Never stop thinking

32-Bit

TC1797

32-Bit Single-Chip Microcontroller

User's Manual

V1.1 2009-05

Microcontrollers

Edition 2009-05

Published by
Infineon Technologies AG
81726 Munich, Germany

Legal Disclaimer

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics. With respect to any examples or hints given herein, any typical values stated herein and/or any information regarding the application of the device, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation, warranties of non-infringement of intellectual property rights of any third party.

Information

For further information on technology, delivery terms and conditions and prices, please contact the nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements, components may contain dangerous substances. For information on the types in question, please contact the nearest Infineon Technologies Office.

Infineon Technologies components may be used in life-support devices or systems only with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

Never stop thinking

32-Bit

TC1797

32-Bit Single-Chip Microcontroller

User's Manual

V1.1 2009-05

Microcontrollers

Revision History: V1.1 2009-05

Previous Version: V1.0

Chapter	Subjects (major changes since last revision)
Page 5-15	PMU: Changed number of DFlash cycles to 30000.
Page 5-40	PMU: Erase Physical Sector command shall not be used for large sectors above S7.
Page 5-69	PMU: BNKSEL bit field description in MARD register is corrected.
Page 5-88	PMU: Footnote related to method of counting single-bit errors is included.
Page 5-96	PMU: Added a section to explain how power failures can be handled in EEPROM emulation.
Page 5-96	PMU: Corrected typo, SPREC is located in UCB1.
Page 6-1	OVC: The introduction text is enhanced.
Page 7-4	Firmware: The ratio of f_{fpi} and f_{OSC} during bootloading upon power-on reset is updated.
Page 7-11	Firmware: The first 16 bytes of LDRAM is overwritten during start-up.
Page 8-5	Memory Map: Foonote for DMI LDRAM section is updated from FPI and LMB point of view .
Page 8-24	Memory Map: Added a new section, Side Effects from Modules to LDRAM.
Page 9-1	Ports: Number of GPIO is corrected.
Page 12-54	EBU: A diagram example for non-multiplexed accesses in asynchronous device section.
Page 12-105, Page 12-108	EBU: The CMDDELAY bit field location in the register description of BUSRAP and BUSWAP is aligned with the register images.
Page 22-1	GPTA: Typos and inconsistencies are fixed.
Page 23-131, Page 23-137	ADC: Reference to port chapter is updated for ADC0 EMUX2, Non-connection for ADC1 EMUX2.
Page 24-1	FADC: Gain calibration feature is removed.
Page 24-58	FADC: Typo is corrected for CR bit field in CRRn register.

Trademarks

We Listen to Your Comments

Any information within this document that you feel is wrong, unclear or missing at all? Your feedback will help us to continuously improve the quality of this document. Please send your proposal (including a reference to this document) to:

mcdocu.comments@infineon.com

Table of Contents

This “Table of Contents” section refers to page numbers of the TC1797 User´s Manual.

1 Introduction

This User´s Manual describes the Infineon TC1797, a 32-bit microcontroller DSP, based on the Infineon TriCore Architecture.

1.1 About this Document

This document is designed to be read primarily by design engineers and software engineers who need a detailed description of the interactions of the TC1797 functional units, registers, instructions, and exceptions.

This TC1797 User´s Manual describes the features of the TC1797 with respect to the TriCore Architecture. Where the TC1797 directly implements TriCore architectural functions, this manual simply refers to those functions as features of the TC1797. In all cases where this manual describes a TC1797 feature without referring to the TriCore Architecture, this means that the TC1797 is a direct implementation of the TriCore Architecture.

Where the TC1797 implements a subset of TriCore architectural features, this manual describes the TC1797 implementation, and then describes how it differs from the TriCore Architecture. Such differences between the TC1797 and the TriCore Architecture are documented in the section covering each such subject.

1.1.1 Related Documentations

A complete description of the TriCore architecture is found in the document entitled “TriCore Architecture Manual”. The architecture of the TC1797 is described separately this way because of the configurable nature of the TriCore specification: Different versions of the architecture may contain a different mix of systems components. The TriCore architecture, however, remains constant across all derivative designs in order to preserve compatibility.

This User´s Manuals together with the “TriCore Architecture Manual” are required to understand the complete TC1797 micro controller functionality.

1.1.2 Text Conventions

This document uses the following text conventions for named components of the TC1797:

- Functional units of the TC1797 are given in plain UPPER CASE. For example: “The SSC supports full-duplex and half-duplex synchronous communication”.
- Pins using negative logic are indicated by an overline. For example: “The external reset pin, $\overline{\text{ESR0}}$, has a dual function.”.
- Bit fields and bits in registers are in general referenced as “Module_Register name.Bit field” or “Module_Register name.Bit”. For example: “The Current CPU Priority Number bit field CPU_ICR.CCPN is cleared”. Most of the

register names contain a module name prefix, separated by an underscore character "_" from the actual register name (for example, "ASC0_CON", where "ASC0" is the module name prefix, and "CON" is the kernel register name). In chapters describing the kernels of the peripheral modules, the registers are mainly referenced with their kernel register names. The peripheral module implementation sections mainly refer to the actual register names with module prefixes.

- Variables used to describe sets of processing units or registers appear in mixed upper and lower cases. For example, register name "MSGCFGn" refers to multiple "MSGCFG" registers with variable n. The bounds of the variables are always given where the register expression is first used (for example, "n = 0-31"), and are repeated as needed in the rest of the text.
- The default radix is decimal. Hexadecimal constants are suffixed with a subscript letter "H", as in 100_H. Binary constants are suffixed with a subscript letter "B", as in: 111_B.
- When the extent of register fields, groups register bits, or groups of pins are collectively named in the body of the document, they are represented as "NAME[A:B]", which defines a range for the named group from B to A. Individual bits, signals, or pins are given as "NAME[C]" where the range of the variable C is given in the text. For example: CFG[2:0] and SRPN[0].
- Units are abbreviated as follows:
  - **MHz** = Megahertz
  - **µs** = Microseconds
  - **kBaud, kbit** = 1000 characters/bits per second
  - **MBaud, Mbit** = 1,000,000 characters/bits per second
  - **Kbyte, KB** = 1024 bytes of memory
  - **Mbyte, MB**= 1048576 bytes of memory

    In general, the k prefix scales a unit by 1000 whereas the K prefix scales a unit by 1024. Hence, the Kbyte unit scales the expression preceding it by 1024. The kBaud unit scales the expression preceding it by 1000. The M prefix scales by 1,000,000 or 1048576, and µ scales by .000001. For example, 1 Kbyte is 1024 bytes, 1 Mbyte is 1024 × 1024 bytes, 1 kBaud/kbit are 1000 characters/bits per second, 1 MBaud/Mbit are 1000000 characters/bits per second, and 1 MHz is 1,000,000 Hz.
- Data format quantities are defined as follows:
  - **Byte** = 8-bit quantity
  - **Half-word** = 16-bit quantity
  - **Word** = 32-bit quantity
  - **Double-word** = 64-bit quantity

1.1.3 Reserved, Undefined, and Unimplemented Terminology

In tables where register bit fields are defined, the following conventions are used to indicate undefined and unimplemented function. Furthermore, types of bits and bit fields are defined using the abbreviations as shown in **Table 1-1**.

Table 1-1 Bit Function Terminology

Function of Bits	Description
Unimplemented, Reserved	Register bit fields named **0** indicate unimplemented functions with the following behavior. • Reading these bit fields returns 0. • These bit fields should be written with 0 if the bit field is defined as r or rh. • These bit fields have to be written with 0 if the bit field is defined as rw. These bit fields are reserved. The detailed description of these bit fields can be found in the register descriptions.
rw	The bit or bit field can be read and written.
rwh	As rw, but bit or bit field can be also set or reset by hardware.
r	The bit or bit field can only be read (read-only).
w	The bit or bit field can only be written (write-only). A read to this register will always give a default value back.
rh	This bit or bit field can be modified by hardware (read-hardware, typical example: status flags). A read of this bit or bit field give the actual status of this bit or bit field back. Writing to this bit or bit field has no effect to the setting of this bit or bit field.
s	Bits with this attribute are “sticky” in one direction. If their reset value is once overwritten by software, they can be switched again into their reset state only by a reset operation. Software cannot switch this type of bit into its reset state by writing the register. This attribute can be combined to “rws” or “rwhs”.
f	Bits with this attribute are readable only when they are accessed by an instruction fetch. Normal data read operations will return other values.

1.1.4 Register Access Modes

Read and write access to registers and memory locations are sometimes restricted. In memory and register access tables, the terms as defined in **Table 1-2** are used.

Table 1-2 Access Terms

Symbol	Description
U	Access Mode: Access permitted in User Mode 0 or 1.
	Reset Value: Value or bit is not changed by a reset operation.
SV	Access permitted in Supervisor Mode.
R	Read-only register.
32	Only 32-bit word accesses are permitted to this register/address range.
E	Endinit-protected register/address.
PW	Password-protected register/address.
NC	No change, indicated register is not changed.
BE	Indicates that an access to this address range generates a Bus Error.
nBE	Indicates that no Bus Error is generated when accessing this address range, even though it is either an access to an undefined address or the access does not follow the given rules.
nE	Indicates that no Error is generated when accessing this address or address range, even though the access is to an undefined address or address range. True for CPU accesses (MTCR/MFCR) to undefined addresses in the CSFR range.

1.1.5 Abbreviations and Acronyms

The following acronyms and terms are used in this document:

ADC	Analog-to-Digital Converter
AGPR	Address General Purpose Register
ALU	Arithmetic and Logic Unit
ASC	Asynchronous/Synchronous Serial Controller
BCU	Bus Control Unit
BROM	Boot ROM & Test ROM
CAN	Controller Area Network
CMEM	PCP Code Memory
CISC	Complex Instruction Set Computing
CPS	CPU Slave Interface
CPU	Central Processing Unit

CSA	Context Save Area
CSFR	Core Special Function Register
DAP	Device Access Port
DAS	Device Access Server
DCACHE	Data Cache
DFLASH	Data Flash Memory
DGPR	Data General Purpose Register
DMA	Direct Memory Access
DMI	Data Memory Interface
EBU	External Bus Interface
EMI	Electro-Magnetic Interference
FADC	Fast Analog-to-Digital Converter
FAM	Flash Array Module
FCS	Flash Command State Machine
FIM	Flash Interface and Control Module
FPI	Flexible Peripheral Interconnect (Bus)
FPU	Floating Point Unit
GPIO	General Purpose Input/Output
GPR	General Purpose Register
GPTA	General Purpose Timer Array
ICACHE	Instruction Cache
I/O	Input / Output
JTAG	Joint Test Action Group = IEEE1149.1
LBCU	Local Memory Bus Control Unit
LDRAM	Local Data RAM
LFI	Local Memory-to-FPI Bus Interface
LMB	Local Memory Bus
LTC	Local Timer Cell
MLI	Micro Link Interface
MMU	Memory Management Unit
MSB	Most Significant Bit
MSC	Micro Second Channel

NC	Not Connected
NMI	Non-Maskable Interrupt
OCDS	On-Chip Debug Support
OVRAM	Overlay Memory
PCP	Peripheral Control Processor
PMU	Program Memory Unit
PLL	Phase Locked Loop
PFLASH	Program Flash Memory
PMI	Program Memory Interface
PMU	Program Memory Unit
PRAM	PCP Parameter RAM
RAM	Random Access Memory
RISC	Reduced Instruction Set Computing
SBCU	System Peripheral Bus Control Unit
SCU	System Control Unit
SFR	Special Function Register
SPB	System Peripheral Bus
SPRAM	Scratch-Pad RAM
SRAM	Static Data Memory
SRN	Service Request Node
SSC	Synchronous Serial Controller
STM	System Timer
WDT	Watchdog Timer

1.2 System Architecture of the TC1797

The TC1797 combines three powerful technologies within one silicon die, achieving new levels of power, speed, and economy for embedded applications:

- Reduced Instruction Set Computing (RISC) processor architecture
- Digital Signal Processing (DSP) operations and addressing modes
- On-chip memories and peripherals

DSP operations and addressing modes provide the computational power necessary to efficiently analyze complex real-world signals. The RISC load/store architecture provides high computational bandwidth with low system cost. On-chip memory and peripherals are designed to support even the most demanding high-bandwidth real-time embedded control-systems tasks.

Additional high-level features of the TC1797 include:

- Efficient memory organization: instruction and data scratch memories, caches
- Serial communication interfaces – flexible synchronous and asynchronous modes
- Peripheral Control Processor – standalone data operations and interrupt servicing
- DMA Controller – DMA operations and interrupt servicing
- General-purpose timers
- High-performance on-chip buses
- On-chip debugging and emulation facilities
- Flexible interconnections to external components
- Flexible power-management

The TC1797 is a high-performance microcontroller with TriCore CPU, program and data memories, buses, bus arbitration, an interrupt controller, a peripheral control processor and a DMA controller and several on-chip peripherals. The TC1797 is designed to meet the needs of the most demanding embedded control systems applications where the competing issues of price/performance, real-time responsiveness, computational power, data bandwidth, and power consumption are key design elements.

The TC1797 offers several versatile on-chip peripheral units such as serial controllers, timer units, and Analog-to-Digital converters. Within the TC1797, all these peripheral units are connected to the TriCore CPU/system via the Flexible Peripheral Interconnect (FPI) Bus and the Local Memory Bus (LMB). Several I/O lines on the TC1797 ports are reserved for these peripheral units to communicate with the external world.

1.2.1 TC1797 Block Diagram

Figure 1-1 shows the block diagram of the TC1797.

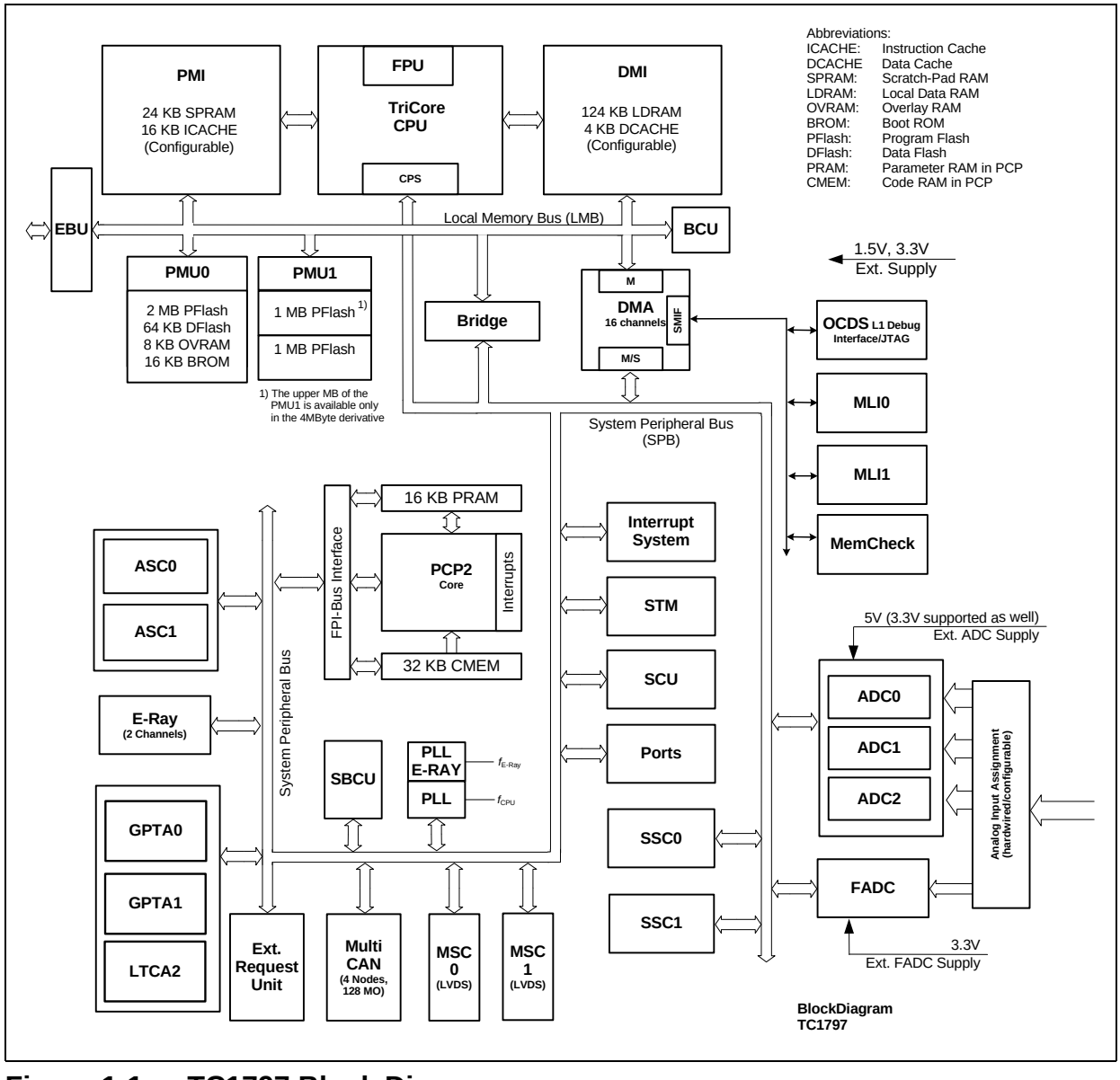


Figure 1-1 TC1797 Block Diagram

1.2.2 System Features

The TC1797 has the following features:

Package

- P-BGA-416 package, 1mm pitch

Clock Frequencies for the 180 MHz derivative

- Maximum CPU clock frequency: 180 MHz[1)]
- Maximum PCP clock frequency: 180 MHz[2)]
- Maximum system clock frequency: 90 MHz[3)]

Clock Frequencies for the 150 MHz derivative

- Maximum CPU clock frequency: 150 MHz[1)]
- Maximum PCP clock frequency: 150 MHz[2)]
- Maximum system clock frequency: 90 MHz[3)]

1) For CPU frequencies > 90 MHz, 2:1 mode has to be enabled. CPU 2:1 mode means: $f_{FPI} = 0.5 * f_{CPU}$

2) For PCP frequencies > 90 MHz, 2:1 mode has to be enabled. PCP 2:1 mode means: $f_{FPI} = 0.5 * f_{PCP}$

3) CPU 1:1 Mode means: $f_{FPI} = f_{CPU}$. PCP 1:1 mode means: $f_{FPI} = f_{PCP}$

1.2.3 CPU Cores of the TC1797

The TC1797 includes a high Performance CPU and a Peripheral Control Processor.

1.2.3.1 High-performance 32-bit CPU

This chapter gives an overview about the TriCore 1 architecture.

TriCore (TC1.3.1) Architectural Highlights

- Unified RISC MCU/DSP
- 32-bit architecture with 4 Gbytes unified data, program, and input/output address space
- Fast automatic context-switching
- Multiply-accumulate unit
- Floating point unit
- Saturating integer arithmetic
- High-performance on-chip peripheral bus (FPI Bus)
- Register based design with multiple variable register banks
- Bit handling
- Packed data operations
- Zero overhead loop
- Precise exceptions
- Flexible power management

High-efficiency TriCore Instruction Set

- 16/32-bit instructions for reduced code size
- Data types include: Boolean, array of bits, character, signed and unsigned integer, integer with saturation, signed fraction, double-word integers, and IEEE-754 single-precision floating point
- Data formats include: Bit, 8-bit byte, 16-bit half-word, 32-bit word, and 64-bit double-word data formats
- Powerful instruction set
- Flexible and efficient addressing mode for high code density

Integrated CPU related On-Chip Memories

- Instruction memory: 40 KB total. After reset, configured into:[1)]
  - 40 Kbyte Scratch-Pad RAM (SPRAM)
  - 0 Kbyte Instruction Cache (ICACHE)
- Data memory: 128 KB total. After reset, configured into:[1)]
  - 128 Kbyte Local Data RAM (LDRAM)

1) Software configurable. Available options are described in the CPU chapter.

  – 0 Kbyte Data Cache (DACHE)
- On-chip SRAMs with parity error detection

1.2.3.2 High-performance 32-bit Peripheral Control Processor

The PCP is a flexible Peripheral Control Processor optimized for interrupt handling and thus unloading the CPU.

Features

- Data move between any two memory or I/O locations
- Data move until predefined limit supported
- Read-Modify-Write capabilities
- Full computation capabilities including basic MUL/DIV
- Read/move data and accumulate it to previously read data
- Read two data values and perform arithmetic or logical operation and store result
- Bit-handling capabilities (testing, setting, clearing)
- Flow control instructions (conditional/unconditional jumps, breakpoint)
- Dedicated Interrupt System
- PCP SRAMs with parity error detection
- PCP/FPI clock mode 1:1 and 2:1 available

Integrated PCP related On-Chip Memories

- 32 Kbyte Code Memory (CMEM)
- 16 Kbyte Parameter Memory (PRAM)

1.3 On-Chip System Units

The TC1797 microcontroller offers several versatile on-chip system peripheral units such as DMA controller, embedded Flash module, interrupt system and ports.

1.3.1 Flexible Interrupt System

The TC1797 includes a programmable interrupt system with the following features:

Features

- Fast interrupt response
- Independent interrupt systems for CPU and PCP
- Each SRN can be mapped to the CPU or PCP interrupt system
- Flexible interrupt-prioritizing scheme with 255 interrupt priority levels per interrupt system

1.3.2 Direct Memory Access Controller

The TC1797 includes a fast and flexible DMA controller with 16 independant DMA channels (two DMA Move Engines).

Features

- 16 independent DMA channels
  - 2 DMA Sub-Blocks with (8 DMA channels per DMA Sub-Block)
  - DMA Sub-Blocks with support of parallel channel execution (1 channel per Sub-Block, both Sub-Blocks in parallel)
  - Up to 16 selectable request inputs per DMA channel
  - 2-level programmable priority of DMA channels within the DMA Sub-Block
  - Software and hardware DMA request
  - Hardware requests by selected on-chip peripherals and external inputs
- 3-level programmable priority of the DMA Sub-Blocks at the on chip bus interfaces
- Buffer capability for move actions on the buses (at least 1 move per bus is buffered)
- Individually programmable operation modes for each DMA channel
  - Single Mode: stops and disables DMA channel after a predefined number of DMA transfers
  - Continuous Mode: DMA channel remains enabled after a predefined number of DMA transfers; DMA transaction can be repeated
  - Programmable address modification
  - Two shadow register modes (with / w/o automatic re-set and direct write access).
- Full 32-bit addressing capability of each DMA channel
  - 4 Gbyte address range
  - Data block move supports > 32 Kbyte moves per DMA transaction
  - Circular buffer addressing mode with flexible circular buffer sizes

- Programmable data width of DMA transfer/transaction: 8-bit, 16-bit, or 32-bit
- Register set for each DMA channel
  – Source and destination address register
  – Channel control and status register
  – Transfer count register
- Flexible interrupt generation (the service request node logic for the MLI channels is also implemented in the DMA modules)
- DMA module is working on SPB frequency, LMB interface on LMB frequency.
- Dependant on the target/destination address, Read/write requests from the Move Engines are directed to the SPB, LMB, MLIs or to the the Cerberus.

1.3.3 System Timer

The TC1797's STM is designed for global system timing applications requiring both high precision and long range.

Features

- Free-running 56-bit counter
- All 56 bits can be read synchronously
- Different 32-bit portions of the 56-bit counter can be read synchronously
- Flexible interrupt generation based on compare match with partial STM content
- Driven by maximum 90 MHz (= f_{SYS}, default after reset = $f_{SYS}/2$)
- Counting starts automatically after a reset operation
- STM registers are reset by an application reset if bit ARSTDIS.STMDIS is cleared. If bit ARSTDIS.STMDIS is set, the STM is not reset.
- STM can be halted in debug/suspend mode

Special STM register semantics provide synchronous views of the entire 56-bit counter, or 32-bit subsets at different levels of resolution.

The maximum clock period is $2^{56} \times f_{STM}$. At f_{STM} = 90 MHz, for example, the STM counts 25.39 years before overflowing. Thus, it is capable of continuously timing the entire expected product life time of a system without overflowing.

In case of a power-on reset, a watchdog reset, or a software reset, the STM is reset. After one of these reset conditions, the STM is enabled and immediately starts counting up. It is not possible to affect the content of the timer during normal operation of the TC1797.

The STM can be optionally disabled for power-saving purposes, or suspended for debugging purposes via its clock control register. In suspend mode of the TC1797 (initiated by writing an appropriate value to STM_CLC register), the STM clock is stopped but all registers are still readable.

Due to the 56-bit width of the STM, it is not possible to read its entire content with one instruction. It needs to be read with two load instructions. Since the timer would continue to count between the two load operations, there is a chance that the two values read are not consistent (due to possible overflow from the low part of the timer to the high part between the two read operations). To enable a synchronous and consistent reading of the STM content, a capture register (STM_CAP) is implemented. It latches the content of the high part of the STM each time when one of the registers STM_TIM0 to STM_TIM5 is read. Thus, STM_CAP holds the upper value of the timer at exactly the same time when the lower part is read. The second read operation would then read the content of the STM_CAP to get the complete timer value.

The content of the 56-bit System Timer can be compared against the content of two compare values stored in the STM_CMP0 and STM_CMP1 registers. Interrupts can be generated on a compare match of the STM with the STM_CMP0 or STM_CMP1 registers.

Figure 1-2 provides an overview on the STM module. It shows the options for reading parts of STM content.

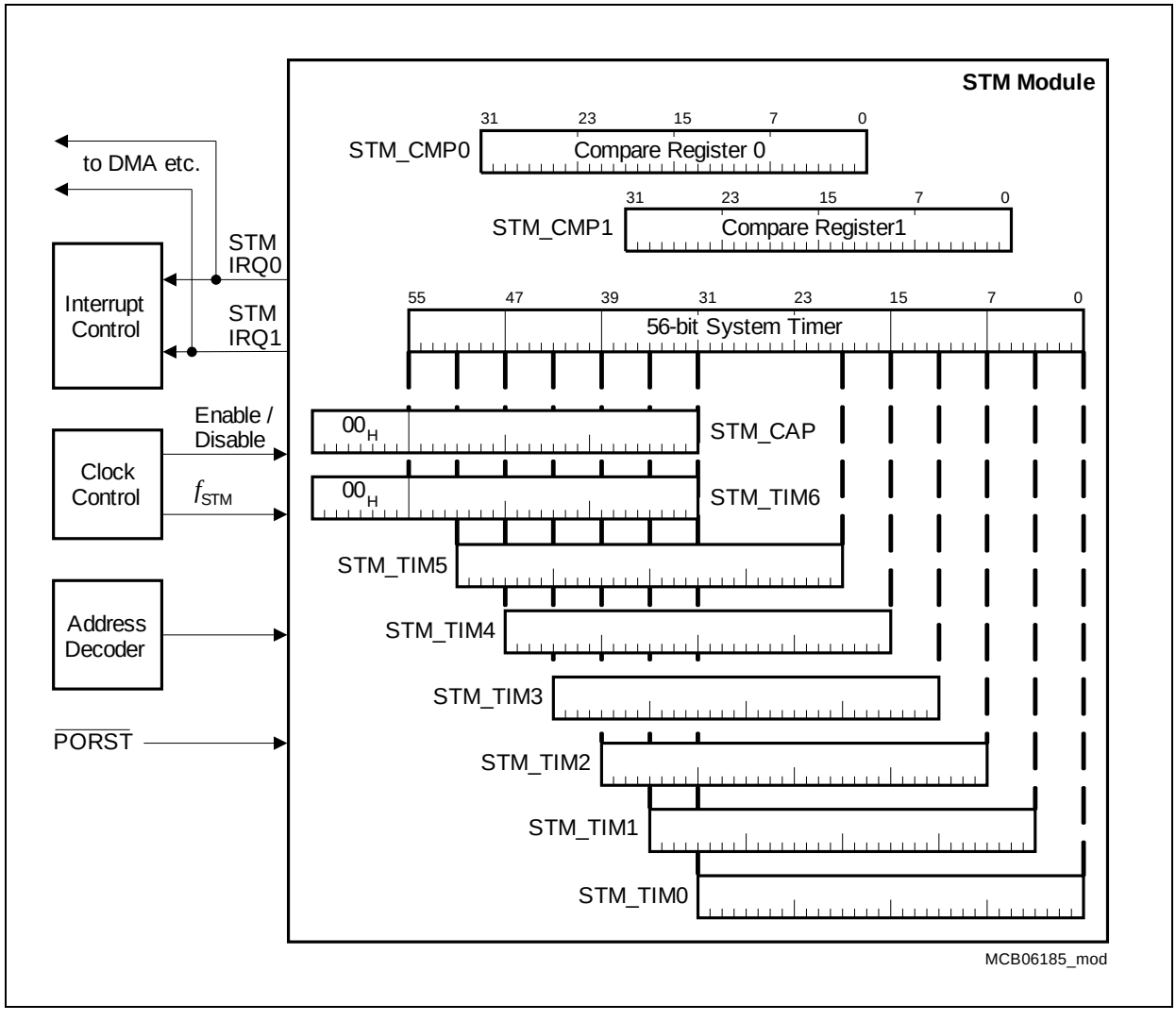


Figure 1-2 General Block Diagram of the STM Module Registers

1.3.4 System Control Unit

The following SCU introduction gives an overview about the TC1797 System Control Unit (SCU) For Information about the SCU see chapter 3.

1.3.4.1 Clock Generation Unit

The Clock Generation Unit (CGU) allows a very flexible clock generation for the TC1797. During user program execution the frequency can be programmed for an optimal ratio between performance and power consumption.

1.3.4.2 Features of the Watchdog Timer

The main features of the WDT are summarized here.

- 16-bit Watchdog counter
- Selectable input frequency: $f_{FPI}/256$ or $f_{FPI}/16384$
- 16-bit user-definable reload value for normal Watchdog operation, fixed reload value for Time-Out and Prewarning Modes
- Incorporation of the ENDINIT bit and monitoring of its modifications
- Sophisticated Password Access mechanism with fixed and user-definable password fields
- Access Error Detection: Invalid password (during first access) or invalid guard bits (during second access) trigger the Watchdog reset generation
- Overflow Error Detection: An overflow of the counter triggers the Watchdog reset generation
- Watchdog function can be disabled; access protection and ENDINIT monitor function remain enabled
- Double Reset Detection

1.3.4.3 Reset Operation

The following reset request triggers are available:

- 1 External power-on hardware reset request trigger; $\overline{\text{PORST}}$, (cold reset)
- 2 External System Request reset triggers; $\overline{\text{ESR0}}$ and $\overline{\text{ESR1}}$,(cold/warm reset)
- Watchdog Timer (WDT) reset request trigger, (warm reset)
- Software reset (SW), (warm reset)
- Debug (OCDS) reset request trigger, (warm reset)
- Resets via the JTAG interface

There are two basic types of reset request triggers:

- Trigger sources that do not depend on a clock, such as the $\overline{\text{PORST}}$. This trigger force the device into an asynchronous reset assertion independently of any clock. The activation of an asynchronous reset is asynchronous to the system clock, whereas its de-assertion is synchronized.

- Trigger sources that need a clock in order to be asserted, such as the input signals ESR0, ESR1, the WDT trigger, the parity trigger, or the SW trigger.

1.3.4.4 External Interface

The SCU provides interface pads for system purpose. Various functions are covered by these pins. Due to the different tasks some of the pads can not be shared with other functions but most of them can be shared with other functions. The following functions are covered by the SCU controlled pads:

- Reset request triggers
- Reset indication
- Trap request triggers
- Interrupt request triggers
- Non SCU module triggers

The first three points are covered by the ESR pads and the last two points by the ERU pads.

1.3.4.5 Die Temperature Measurement

The Die Temperature Sensor (DTS) generates a measurement result that indicates directly the current temperature. The result of the measurement can be read via an DTS register.

1.3.5 General Purpose I/O Ports and Peripheral I/O Lines

The TC1797 includes a flexible Ports structure with the following features:

Features

- Digital General-Purpose Input/Output (GPIO) port lines
- Input/output functionality individually programmable for each port line
- Programmable input characteristics (pull-up, pull-down, no pull device)
- Programmable output driver strength for EMI minimization (weak, medium, strong)
- Programmable output characteristics (push-pull, open drain)
- Programmable alternate output functions
- Output lines of each port can be updated port-wise or set/reset/toggled bit-wise

1.3.6 Program Memory Unit (PMU)

The devices of the AudoF family contain at least one Program Memory Unit. This is named “PMU0”. Some devices contain additional PMUs which are named “PMU1”, …

In the TC1797, the PMU0 contains the following submodules:

- The Flash command and fetch control interface for Program Flash and Data Flash.
- The Overlay RAM interface with Online Data Acquisition (OLDA) support.

- The Boot ROM interface.
- The Emulation Memory interface.
- The Local Memory Bus LMB slave interface.

Following memories are controlled by and belong to the PMU0:

- 2 Mbyte of Program Flash memory (PFlash)
- 64 Kbyte of Data Flash memory (DFlash, represents 16 Kbyte EEPROM)
- 16 Kbyte of Boot ROM (BROM)
- 8 Kbyte Overlay RAM (OVRAM)

In the TC1797 an additional PMU is included with only a subset of PMU0's submodules:

- The Flash command and fetch control interface but only for Program Flash.
- The Local Memory Bus LMB slave interface.

The following memories are controlled and belong to the PMU1:

- 2 Mbyte of Program Flash memory (PFlash).

Because of its independence from PMU0 this second PMU enables additional functionality: Read while Write (RWW), Write while Write (WWW) or concurrent data and instruction accesses, if those are operating on different PMUs.

The following figure shows the block diagram of the PMU0:

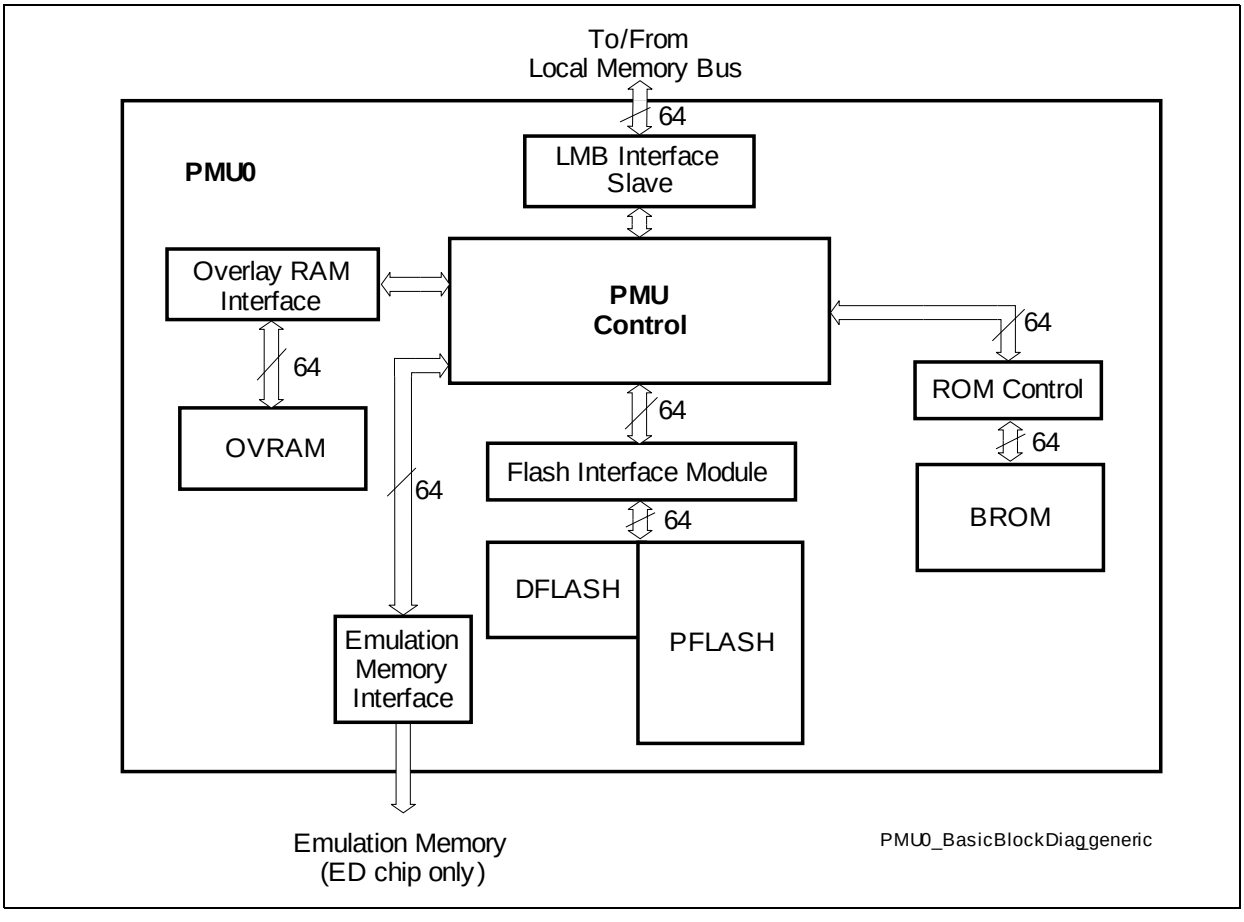


Figure 1-3 PMU0 Basic Block Diagram

As described before the PMU1 is reduced to the PFLASH and its controlling submodules.

1.3.6.1 Boot ROM

The internal 16 Kbyte Boot ROM (BROM) is divided into two parts, used for:

- firmware (Boot ROM), and
- factory test routines (Test ROM).

The different sections of the firmware in Boot ROM provide startup and boot operations after reset. The TestROM is reserved for special routines, which are used for testing, stressing and qualification of the component.

1.3.6.2 Overlay RAM and Data Acquisition

The overlay memory OVRAM is provided in the PMU especially for redirection of data accesses to program memory to the OVRAM by using the data overlay function. The data overlay functionality itself is controlled in the DMI module.

For online data acquisition (OLDA) of application or calibration data a virtual 32 KB memory range is provided which can be accessed without error reporting. Accesses to this OLDA range can also be redirected to an overlay memory.

1.3.6.3 Emulation Memory Interface

In TC1797 Emulation Device, an Emulation Memory (EMEM) is provided, which can fully be used for calibration via program memory or OLDA overlay. The Emulation Memory interface shown in **Figure 1-3** is a 64-bit wide memory interface that controls the CPU-accesses to the Emulation Memory in the TC1797 Emulation Device. In the TC1797 production device, the EMEM interface is always disabled.

1.3.6.4 Tuning Protection

Tuning protection is required by the user to absolutely protect control data (e.g. for engine control), serial number and user software, stored in the Flash, from being manipulated, and to safely detect changed or disturbed data. For the internal Flash, these protection requirements are excellently fulfilled in the TC1797 with

- Flash read and write protection with user-specific protection levels, and with
- dedicated HW and firmware, supporting the internal Flash read protection, and with
- the Alternate Boot Mode.

Special tuning protection support is provided for external Flash, which must also be protected.

1.3.6.5 Program and Data Flash

The embedded Flash modules of PMU0 includes 2 Mbyte of Flash memory for code or constant data (called Program Flash) and additionally 64 Kbyte of Flash memory used for emulation of EEPROM data (called Data Flash). The Program Flash is realized as one independent Flash bank, whereas the Data Flash is built of two Flash banks, allowing the following combinations of concurrent Flash operations:

- Read code or data from Program Flash, while one bank of Data Flash is busy with a program or erase operation.
- Read data from one bank of Data Flash, while the other bank of Data Flash is busy with a program or erase operation.
- Program one bank of Data Flash while erasing the other bank of Data Flash, read from Program Flash.

In TC1797 the PMU1 contains 2 Mbyte of Program Flash realized as one Flash bank. It does not contain any Data Flash.

Since in TC1797 the two PMUs can work in parallel, further combinations of concurrent operations are supported if those are operating on Flash modules in different PMUs, e.g.

- Read data from Flash1 while accessing code from Flash0.
- Read code or data from one Flash while the other Flash is busy with program or erase operation.
- Both Flash modules are concurrently busy with program or erase operation.

Both, the Program Flash and the Data Flash, provide error correction of single-bit errors within a 64-bit read double-word, resulting in an extremely low failure rate. Read accesses to Program Flash are executed in 256-bit width, to Data Flash in 64-bit width (both plus ECC). Single-cycle burst transfers of up to 4 double-words and sequential prefetching with control of prefetch hit are supported for Program Flash.

The minimum programming width is the page, including 256 bytes in Program Flash and 128 bytes in Data Flash. Concurrent programming and erasing in Data Flash is performed using an automatic erase suspend and resume function.

A basic block diagram of the Flash Module is shown in the following figure.

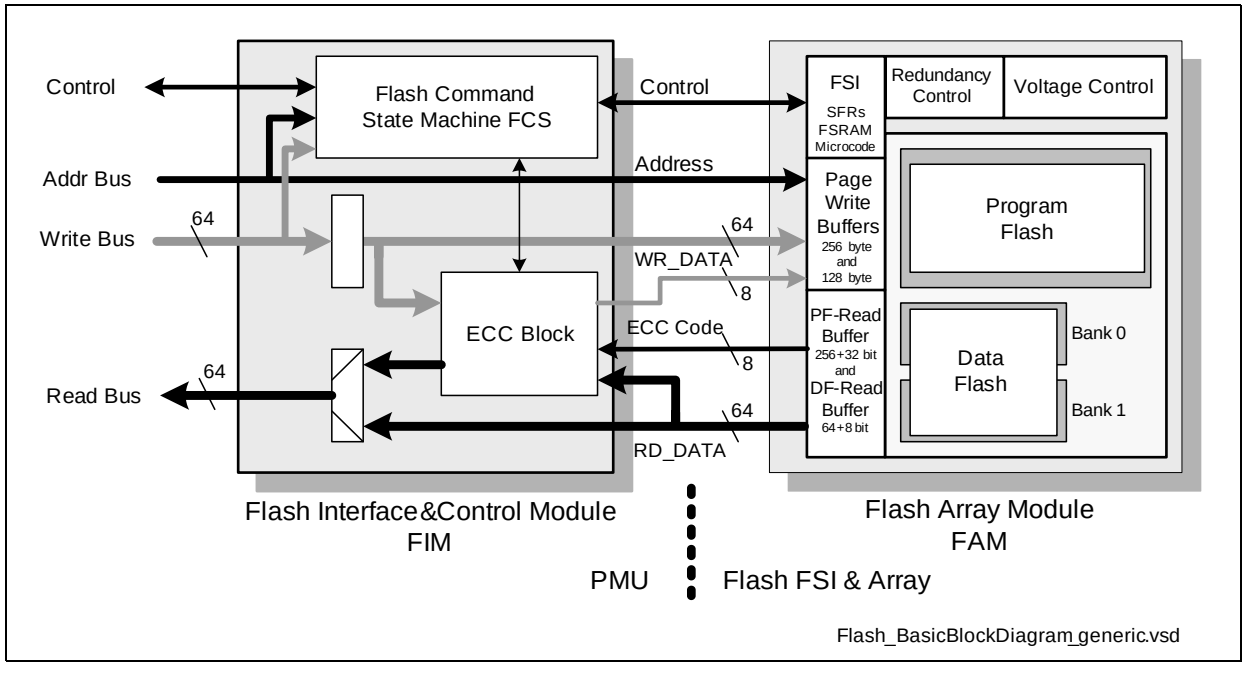


Figure 1-4 Basic Block Diagram of Flash Module

All Flash operations are controlled simply by transferring command sequences to the Flash which are based on JEDEC standard. This user interface of the embedded Flash is very comfortable, because all operations are controlled with high level commands, such as “Erase Sector”. State transitions, such as termination of command execution, or errors are reported to the user by maskable interrupts. Command sequences are

normally written to Flash by the CPU, but may also be issued by the DMA controller (or OCDS).

The Flash also features an advanced read/write protection architecture, including a read protection for the whole Flash array (optionally without Data Flash) and separate write protection for all sectors (only Program Flash). Write protected sectors can be made re-programmable (enabled with passwords), or they can be locked for ever (ROM function). Each sector can be assigned to up to three different users for write protection. The different users are organized hierarchically.

Program Flash Features and Functions

- 2 Mbyte on-chip Program Flash in PMU0.
- 2 Mbyte on-chip Program Flash in PMU1.
- Any use for instruction code or constant data.
- Double Flash module system approach:
  - Concurrent read access of code and data.
  - Read while write (RWW).
  - Concurrent program/erase in both modules.
- 256 bit read interface (burst transfer operation).
- Dynamic correction of single-bit errors during read access.
- Transfer rate in burst mode: One 64-bit double-word per clock cycle.
- Sector architecture:
  - Eight 16 Kbyte, one 128 Kbyte and seven 256 Kbyte sectors.
  - Each sector separately erasable.
  - Each sector lockable for protection against erase and program (write protection).
- One additional configuration sector (not accessible to the user).
- Optional read protection for whole Flash, with sophisticated read access supervision. Combined with whole Flash write protection — thus supporting protection against Trojan horse programs.
- Sector specific write protection with support of re-programmability or locked forever.
- Comfortable password checking for temporary disable of write or read protection.
- User controlled configuration blocks (UCB) in configuration sector for keywords and for sector-specific lock bits (one block for every user; up to three users).
- Pad supply voltage (V_{DDP}) also used for program and erase (no VPP pin).
- Efficient 256 byte page program operation.
- All Flash operations controlled by CPU per command sequences (unlock sequences) for protection against unintended operation.
- End-of-busy as well as error reporting with interrupt and bus error trap.
- Write state machine for automatic program and erase, including verification of operation quality.
- Support of margin check.
- Delivery in erased state (read all zeros).
- Global and sector status information.

- Overlay support with SRAM for calibration applications.
- Configurable wait state selection for different CPU frequencies.
- Endurance = 1000; minimum 1000 program/erase cycles per physical sector; reduced endurance of 100 per 16 KB sector.
- Operating lifetime (incl. Retention): 20 years with endurance=1000.

Data Flash Features and Functions

Note: Only available in PMU0.

- 64 Kbyte on-chip Flash, configured in two independent Flash banks of equal size.
- 64 bit read interface.
- Erase/program one bank while data read access from the other bank.
- Programming one bank while erasing the other bank using an automatic suspend/resume function.
- Dynamic correction of single-bit errors during read access.
- Sector architecture:
  – Two sectors of equal size.
  – Each sector separately erasable.
- 128 byte pages to be written in one step.
- Operational control per command sequences (unlock sequences, same as those of Program Flash) for protection against unintended operation.
- End-of-busy as well as error reporting with interrupt and bus error trap.
- Write state machine for automatic program and erase.
- Margin check for detection of problematic Flash bits.
- Endurance = 30000 (can be device dependent); i.e. 30000 program/erase cycles per sector are allowed, with a retention of min. 5 years.
- Dedicated DFlash status information.
- Other characteristics: Same as Program Flash.

1.3.7 Data Access Overlay

The data overlay functionality provides the capability to redirect data accesses by the TriCore to program memory (segments 8_H and A_H) called “target memory” to a different memory called “overlay memory”.

Depending on the device the following overlay memories can be available:

- Overlay SRAM in the PMU.
- Emulation Memory[1)].
- External memory[2)].

1) Only available in Emulation Device “ED”.

2) Only available in Emulation Device with EBU.

This functionality makes it possible, for example, to modify the application's test and calibration parameters (which are typically stored in the program memory) during run time of a program.

As the address translation is implemented in the DMI, it affects only data accesses (reads and writes) of the TriCore. Instruction fetches by the TriCore or accesses by any other master (including the debug interface) are not redirected.

Summary of Features and Functions

- 16 overlay ranges ("blocks") configurable.
- Support of 8 Kbyte embedded Overlay SRAM (OVRAM) in PMU.
- Support of up to 512 Kbyte overlay/calibration memory (EMEM)[1].
- Support of up to 2 MB overlay memory in external memory (EBU space)[2].
- Support of Online Data Acquisition into range of up to 32 KB and of its overlay.
- Support of different overlay memory selections for every enabled overlay block.
- Sizes of overlay blocks selectable depending on the overlay memory:
  - OVRAM: from 16 byte to 2 Kbyte.
  - EMEM[1] and external memory[2]: 1 Kbyte to 128 Kbyte.
- All configured overlay ranges can be enabled with only one register write access.
- Programmable flush (invalidate) control for data cache in DMI.

1.3.8 Development Support

Overview about the TC1797 development environment:

Complete Development Support

A variety of software and hardware development tools for the 32-bit microcontroller TC1797 are available from experienced international tool suppliers. The development environment for the Infineon 32-bit microcontroller includes the following tools:

- Embedded Development Environment for TriCore Products
- The TC1797 On-chip Debug Support (OCDS) provides a JTAG port for communication between external hardware and the system
- Flexible Peripheral Interconnect Buses (FPI Bus) for on-chip interconnections and its FPI Bus control unit (SBCU)
- The System Timer (STM) with high-precision, long-range timing capabilities
- The TC1797 includes a power management system, a watchdog timer as well as reset logic

1.4 On-Chip Peripheral Units of the TC1797

The TC1797 microcontroller offers several versatile on-chip peripheral units such as serial controllers, timer units, and Analog-to-Digital converters. Several I/O lines on the TC1797 ports are reserved for these peripheral units to communicate with the external world.

On-Chip Peripheral Units

- Two Asynchronous/Synchronous Serial Channels (ASC) with baud-rate generator, parity, framing and overrun error detection
- Two Synchronous Serial Channels (SSC) with programmable data length and shift direction
- Two Micro Second Bus Interfaces (MSC) for serial communication
- One CAN Module with four CAN nodes (MultiCAN) for high-efficiency data handling via FIFO buffering and gateway data transfer
- Two Micro Link Serial Bus Interfaces (MLI) for serial multiprocessor communication
- Two General Purpose Timer Arrays (GPTA) with a powerful set of digital signal filtering and timer functionality to accomplish autonomous and complex Input/Output management. One additional Local Timer Cell Array (LCTA).
- Three Analog-to-Digital Converter Units (ADC) with 8-bit, 10-bit, or 12-bit resolution.
- One fast Analog-to-Digital Converter Unit (FADC)
- One FlexRay™ module with 2 channels (E-Ray).
- One External Bus Interface (EBU)

1.4.1 Asynchronous/Synchronous Serial Interfaces

The TC1797 includes two Asynchronous/Synchronous Serial Interfaces, ASC0 and ASC1. Both ASC modules have the same functionality.

Figure 1-5 shows a global view of the Asynchronous/Synchronous Serial Interface (ASC).

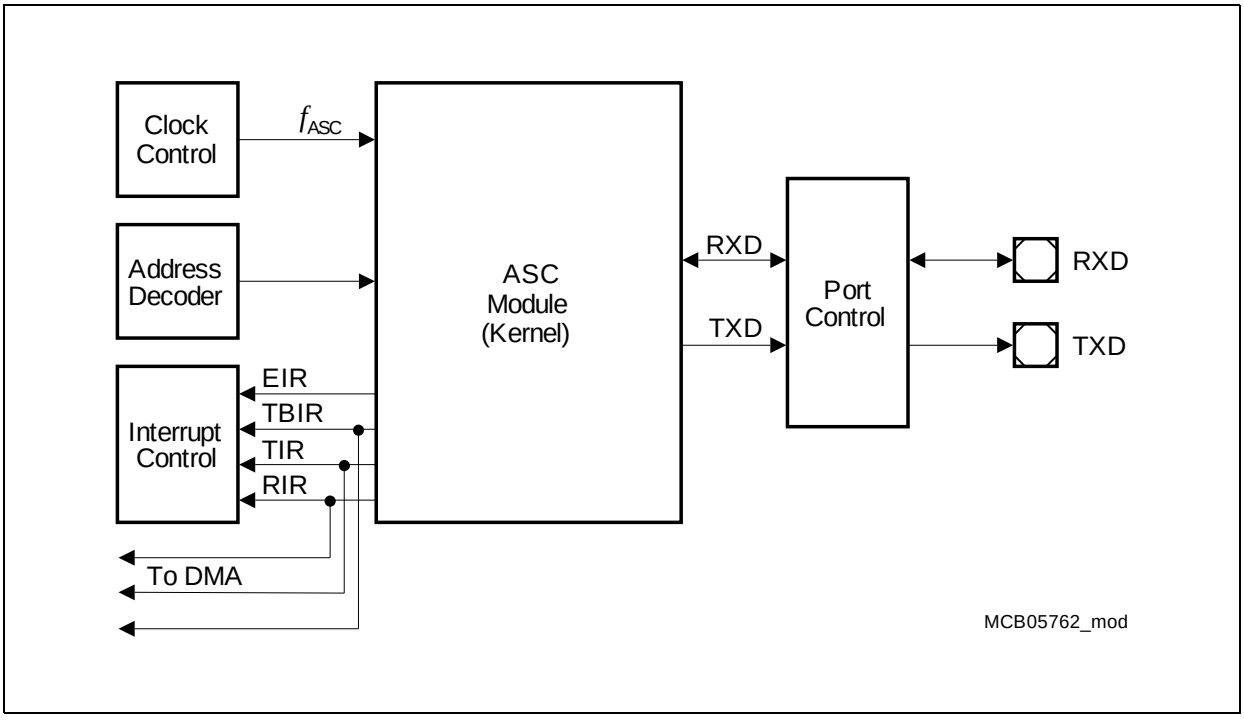


Figure 1-5 General Block Diagram of the ASC Interface

The ASC provides serial communication between the TC1797 and other microcontrollers, microprocessors, or external peripherals.

The ASC supports full-duplex asynchronous communication and half-duplex synchronous communication. In Synchronous Mode, data is transmitted or received synchronous to a shift clock that is generated by the ASC internally. In Asynchronous Mode, 8-bit or 9-bit data transfer, parity generation, and the number of stop bits can be selected. Parity, framing, and overrun error detection are provided to increase the reliability of data transfers. Transmission and reception of data is double-buffered. For multiprocessor communication, a mechanism is included to distinguish address bytes from data bytes. Testing is supported by a loop-back option. A 13-bit baud rate generator provides the ASC with a separate serial clock signal, which can be accurately adjusted by a prescaler implemented as fractional divider.

Features

- Full-duplex asynchronous operating modes
  - 8-bit or 9-bit data frames, LSB first
  - Parity-bit generation/checking
  - One or two stop bits
  - Baud rate from 5.625 Mbit/s to 1.34 bit/s (@ 90 MHz module clock)
  - Multiprocessor mode for automatic address/data byte detection
  - Loop-back capability
- Half-duplex 8-bit synchronous operating mode
  - Baud rate from 11.25 Mbit/s to 915.5 bit/s (@ 90 MHz module clock)
- Double-buffered transmitter/receiver
- Interrupt generation
  - On a transmit buffer empty condition
  - On a transmit last bit of a frame condition
  - On a receive buffer full condition
  - On an error condition (frame, parity, overrun error)
- Implementation features
  - Connections to DMA Controller
  - Connections of receiver input to GPTA (LTC) for baud rate detection and LIN break signal measuring

1.4.2 High-Speed Synchronous Serial Interfaces

The TC1797 includes two High-Speed Synchronous Serial Interfaces, SSC0 and SSC1. Both SSC modules have the same functionality.

Figure 1-6 shows a global view of the Synchronous Serial interface (SSC).

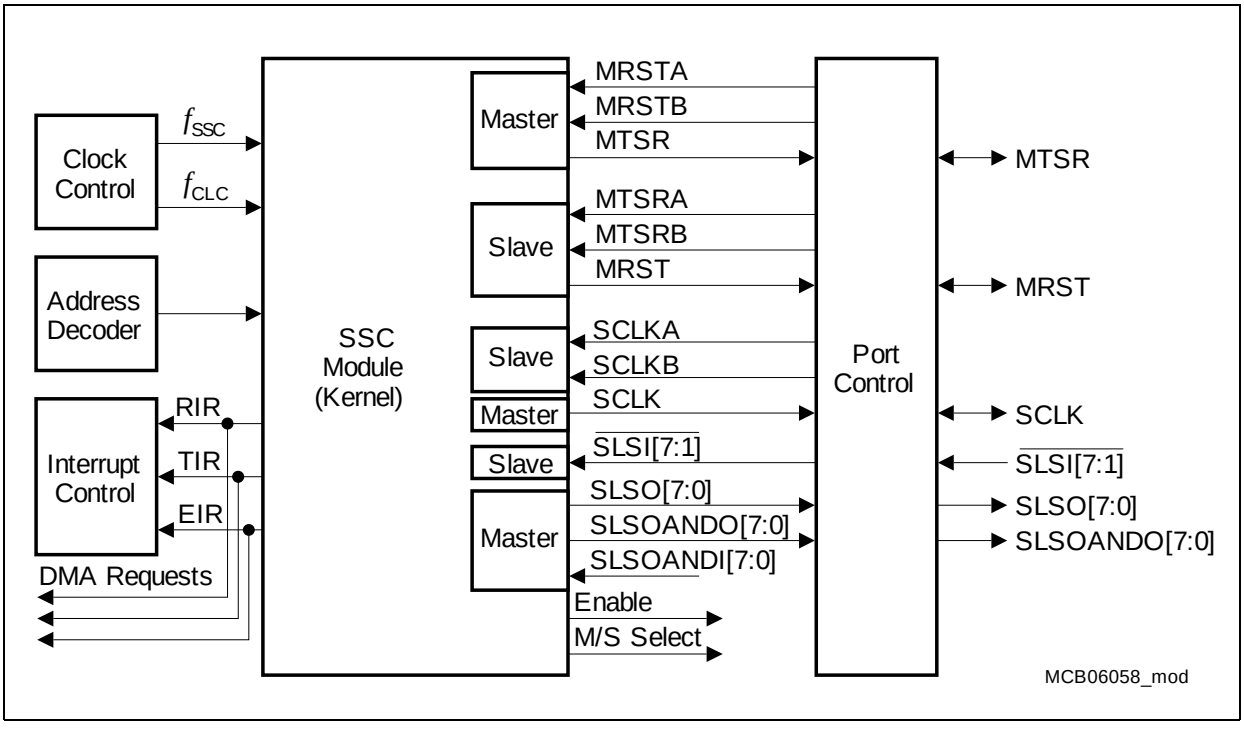


Figure 1-6 General Block Diagram of the SSC Interface

The SSC supports full-duplex and half-duplex serial synchronous communication up to 45 Mbit/s (@ 90 MHz module clock, Master Mode). The serial clock signal can be generated by the SSC itself (Master Mode) or can be received from an external master (Slave Mode). Data width, shift direction, clock polarity and phase are programmable. This allows communication with SPI-compatible devices. Transmission and reception of data are double-buffered. A shift clock generator provides the SSC with a separate serial clock signal. One slave select input is available for slave mode operation. Eight programmable slave select outputs (chip selects) are supported in Master Mode.

Features

- Master and Slave Mode operation
  - Full-duplex or half-duplex operation
  - Automatic pad control possible
- Flexible data format
  - Programmable number of data bits: 2 to 16 bits
  - Programmable shift direction: LSB or MSB shift first
  - Programmable clock polarity: Idle low or idle high state for the shift clock
  - Programmable clock/data phase: Data shift with leading or trailing edge of the shift clock
- Baud rate generation
  - Master Mode:
  - Slave Mode:
- Interrupt generation
  - On a transmitter empty condition
  - On a receiver full condition
  - On an error condition (receive, phase, baud rate, transmit error)
- Flexible SSC pin configuration
- Seven slave select inputs $\overline{\text{SLSI}}$[7:1] in Slave Mode
- Eight programmable slave select outputs SLSO[7:0] in Master Mode
  - Automatic SLSO generation with programmable timing
  - Programmable active level and enable control
  - Combinable with SLSO output signals from other SSC modules

1.4.3 Micro Second Channel Interface

The TC1797 includes two Micro Second Channel interfaces, MSC0 and MSC1. Both MSC modules have the same functionality.

Each Micro Second Channel (MSC) interface provides serial communication links typically used to connect power switches or other peripheral devices. The serial communication link includes a fast synchronous downstream channel and a slow asynchronous upstream channel. **Figure 1-7** shows a global view of the interface signals of an MSC interface.

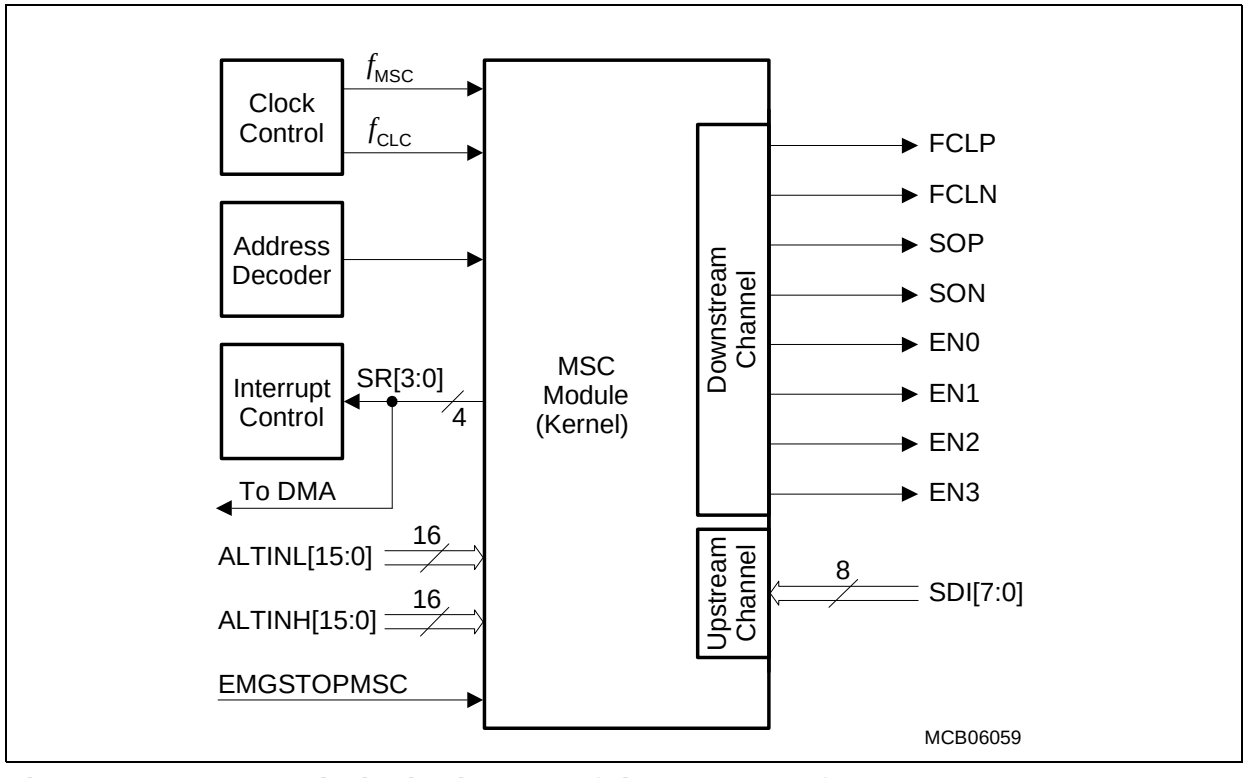


Figure 1-7 General Block Diagram of the MSC Interface

The downstream and upstream channels of the MSC module communicate with the external world via nine I/O lines. Eight output lines are required for the serial communication of the downstream channel (clock, data, and enable signals). One out of eight input lines SDI[7:0] is used as serial data input signal for the upstream channel. The source of the serial data to be transmitted by the downstream channel can be MSC register contents or data that is provided on the ALTINL/ALTINH input lines. These input lines are typically connected with other on-chip peripheral units (for example with a timer unit such as the GPTA). An emergency stop input signal makes it possible to set bits of the serial data stream to dedicated values in an emergency case.

Clock control, address decoding, and interrupt service request control are managed outside the MSC module kernel. Service request outputs are able to trigger an interrupt or a DMA request.

Features

- Fast synchronous serial interface to connect power switches in particular, or other peripheral devices via serial buses
- High-speed synchronous serial transmission on downstream channel
  - Serial output clock frequency: $f_{FCL} = f_{MSC}/2$ (f_{MSCmax} = 90 MHz)
  - Fractional clock divider for precise frequency control of serial clock f_{MSC}
  - Command, data, and passive frame types
  - Start of serial frame: Software-controlled, timer-controlled, or free-running
  - Programmable upstream data frame length (16 or 12 bits)
  - Transmission with or without SEL bit
  - Flexible chip select generation indicates status during serial frame transmission
  - Emergency stop without CPU intervention
- Low-speed asynchronous serial reception on upstream channel
  - Baud rate: f_{MSC} divided by 4, 8, 16, 32, 64, 128, or 256 (f_{MSCmax} = 90 MHz)
  - Standard asynchronous serial frames
  - Parity error checker
  - 8-to-1 input multiplexer for SDI lines
  - Built-in spike filter on SDI lines
- Selectable pin types of downstream channel interface:
  four LVDS differential output drivers or four digital GPIO pins

1.4.4 FlexRay™ Protocol Controller (Intro)

The Intro IP-module performs communication according to the FlexRay™ [1] protocol specification v2.1. With maximum specified clock the bitrate can be programmed to values up to 10 Mbit/s. Additional bus driver (BD) hardware is required for connection to the physical layer.

1.4.4.1 Intro Kernel Description

Figure 1.4.4.1 shows a global view of the Intro interface.

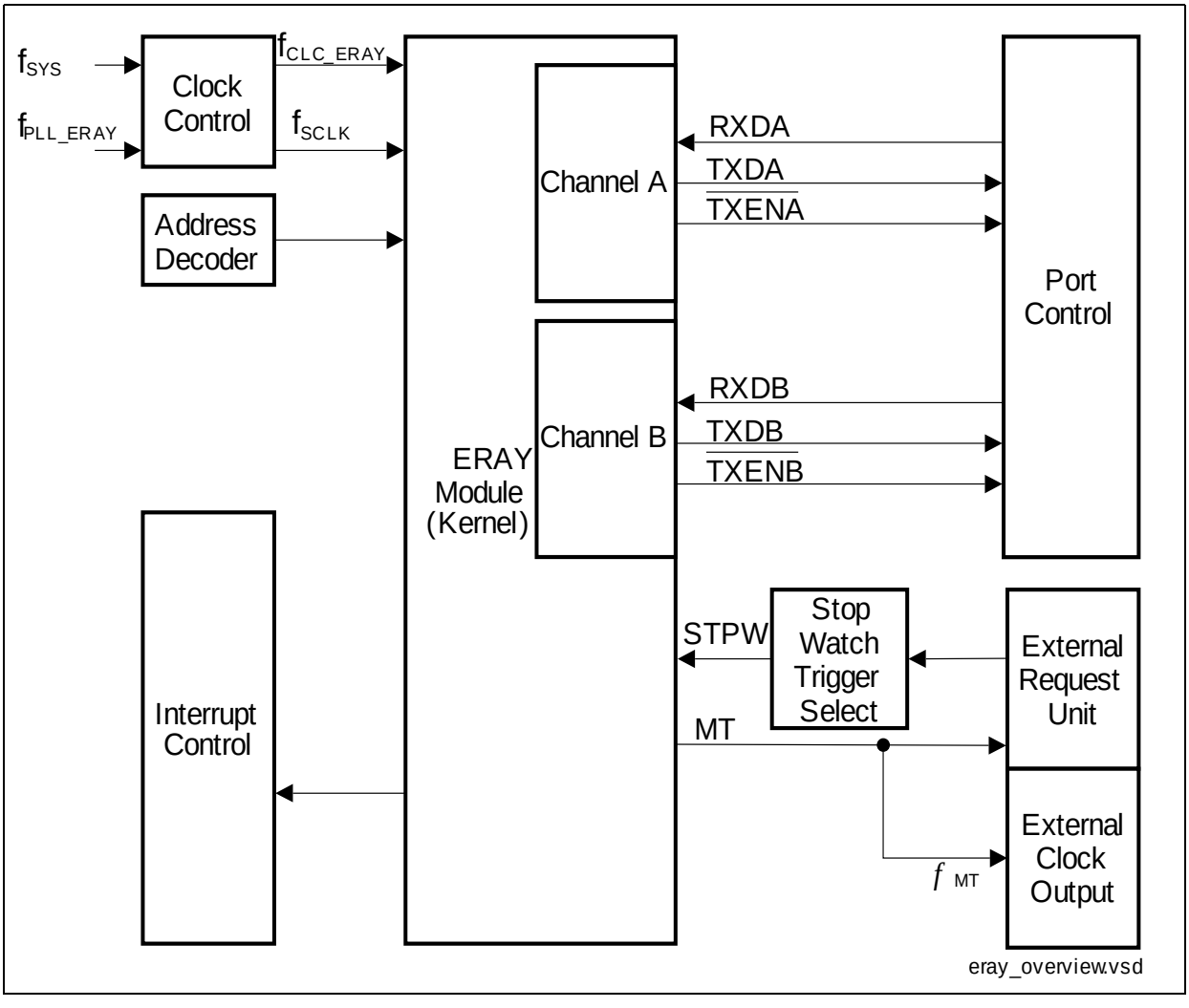


Figure 1-8 General Block Diagram of the Intro Interface

1) *Infineon®, Infineon Technologies®, are trademarks of Infineon Technologies AG. FlexRay™* is a trademark of FlexRay Consortium.

The Intro module communicates with the external world via three I/O lines each channel. The RXDAx and RXDBx lines are the receive data input signals, TXDA and TXDB lines are the transmit output signals, TXENA and TXENB the transmit enable signals.

Clock control, address decoding, and service request control are managed outside the Intro module kernel.

1.4.4.2 Overview

For communication on a FlexRay™ network, individual Message Buffers with up to 254 data byte are configurable. The message storage consists of a single-ported Message RAM that holds up to 128 Message Buffers. All functions concerning the handling of messages are implemented in the Message Handler. Those functions are the acceptance filtering, the transfer of messages between the two FlexRay™ Channel Protocol Controllers and the Message RAM, maintaining the transmission schedule as well as providing message status information.

The register set of the Intro IP-module can be accessed directly by an external Host via the module's Host interface. These registers are used to control/configure/monitor the FlexRay™ Channel Protocol Controllers, Message Handler, Global Time Unit, System Universal Control, Frame and Symbol Processing, Network Management, Service Request Control, and to access the Message RAM via Input / Output Buffer.

The Intro IP-module supports the following features:

- Conformance with FlexRay™ protocol specification v2.1
- Data rates of up to 10 Mbit/s on each channel
- Up to 128 Message Buffers configurable
- 8 Kbyte of Message RAM for storage of e.g. 128 Message Buffers with max. 48 byte data field or up to 30 Message Buffers with 254 byte Data Sections
- Configuration of Message Buffers with different payload lengths possible
- One configurable receive FIFO
- Each Message Buffer can be configured as receive buffer, as transmit buffer or as part of the receive FIFO
- Host access to Message Buffers via Input and Output Buffer.
  Input Buffer: Holds message to be transferred to the Message RAM
  Output Buffer: Holds message read from the Message RAM
- Filtering for slot counter, cycle counter, and channel
- Maskable module service requests
- Network Management supported
- Four service request lines
- Automatic delayed read access to Output Command Request Register (OBCR) if a data transfer from Message RAM to Output Shadow Buffer (initiated by a previous write access to the OBCR) is ongoing.

- Automatic delayed read access to Input Command Request Register (IBCR) if a data transfer from Input Shadow Buffer to Message RAM to (initiated by a previous write access to the IBCR) is ongoing.
- Four Input Buffer for building up transmission Frames in parallel.
- Flag indicating which Input Buffer is currently accessible by the host.

1.4.5 MultiCAN Controller

The MultiCAN module provides four independent CAN nodes, representing four serial communication interfaces. The number of available message objects is 128.

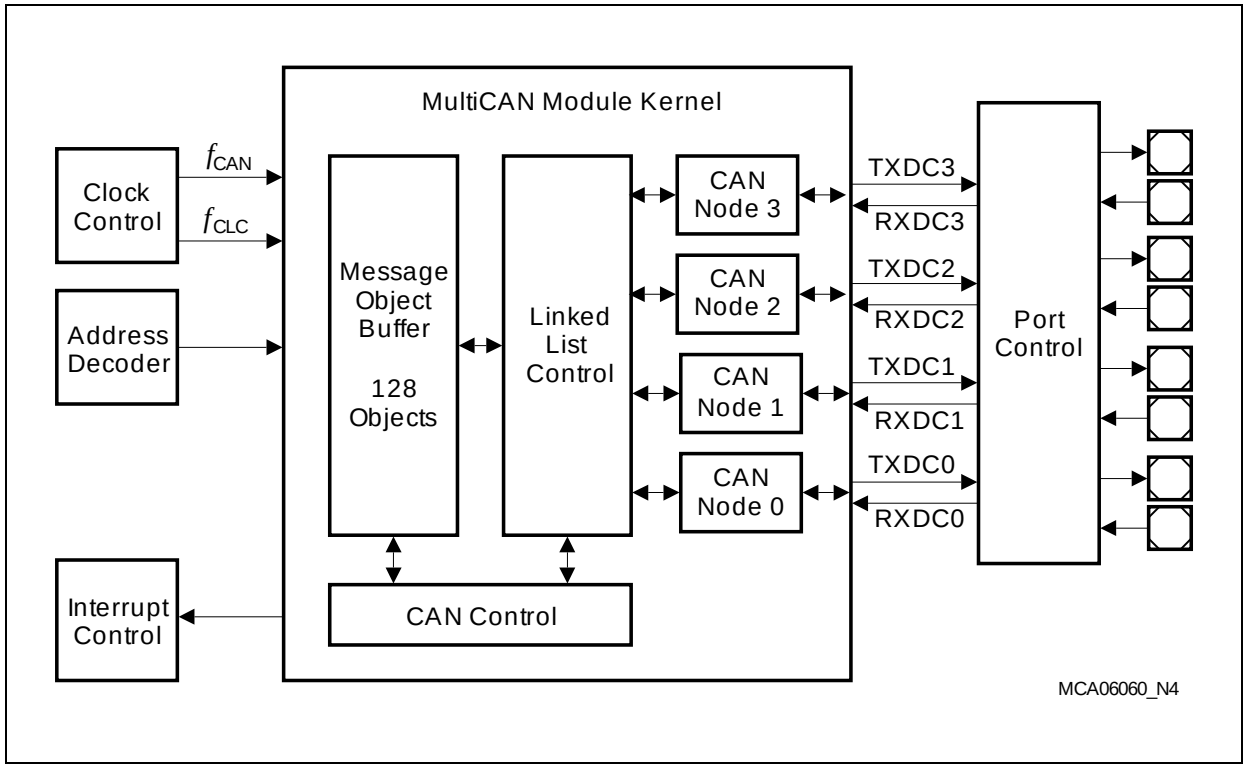


Figure 1-9 Overview of the MultiCAN Module

The MultiCAN module contains four independently operating CAN nodes with Full-CAN functionality that are able to exchange Data and Remote Frames via a gateway function. Transmission and reception of CAN frames is handled in accordance to CAN specification V2.0 B (active). Each CAN node can receive and transmit standard frames with 11-bit identifiers as well as extended frames with 29-bit identifiers.

All four CAN nodes share a common set of message objects. Each message object can be individually allocated to one of the CAN nodes. Besides serving as a storage container for incoming and outgoing frames, message objects can be combined to build gateways between the CAN nodes or to set up a FIFO buffer.

The message objects are organized in double-chained linked lists, where each CAN node has its own list of message objects. A CAN node stores frames only into message objects that are allocated to the message object list of the CAN node, and it transmits only messages belonging to this message object list. A powerful, command-driven list controller performs all message object list operations.

The bit timings for the CAN nodes are derived from the module timer clock (f_{CAN}) and are programmable up to a data rate of 1 Mbit/s. External bus transceivers are connected to a CAN node via a pair of receive and transmit pins.

Features

- Compliant with ISO 11898
- CAN functionality according to CAN specification V2.0 B active
- Dedicated control registers for each CAN node
- Data transfer rates up to 1 Mbit/s
- Flexible and powerful message transfer control and error handling capabilities
- Advanced CAN bus bit timing analysis and baud rate detection for each CAN node via a frame counter
- Full-CAN functionality: A set of 128 message objects can be individually
  - Allocated (assigned) to any CAN node
  - Configured as transmit or receive object
  - Setup to handle frames with 11-bit or 29-bit identifier
  - Identified by a timestamp via a frame counter
  - Configured to remote monitoring mode
- Advanced Acceptance Filtering
  - Each message object provides an individual acceptance mask to filter incoming frames.
  - A message object can be configured to accept standard or extended frames or to accept both standard and extended frames.
  - Message objects can be grouped into four priority classes for transmission and reception.
  - The selection of the message to be transmitted first can be based on frame identifier, IDE bit and RTR bit according to CAN arbitration rules, or on its order in the list.
- Advanced message object functionality
  - Message objects can be combined to build FIFO message buffers of arbitrary size, limited only by the total number of message objects.
  - Message objects can be linked to form a gateway that automatically transfers frames between 2 different CAN buses. A single gateway can link any two CAN nodes. An arbitrary number of gateways can be defined.
- Advanced data management
  - The message objects are organized in double-chained lists.
  - List reorganizations can be performed at any time, even during full operation of the CAN nodes.
  - A powerful, command-driven list controller manages the organization of the list structure and ensures consistency of the list.
  - Message FIFOs are based on the list structure and can easily be scaled in size during CAN operation.
  - Static allocation commands offer compatibility with MultiCAN applications that are not list-based.
- Advanced interrupt handling

– Up to 16 interrupt output lines are available. Interrupt requests can be routed individually to one of the 16 interrupt output lines.
– Message post-processing notifications can be combined flexibly into a dedicated register field of 256 notification bits.

1.4.6 Micro Link Serial Bus Interface

This TC1797 contains two Micro Link Serial Bus Interfaces, MLI0 and MLI1.

The Micro Link Interface (MLI) is a fast synchronous serial interface to exchange data between microcontrollers or other devices, such as stand-alone peripheral components. **Figure 1-10** shows how two microcontrollers are typically connected together via their MLI interfaces.

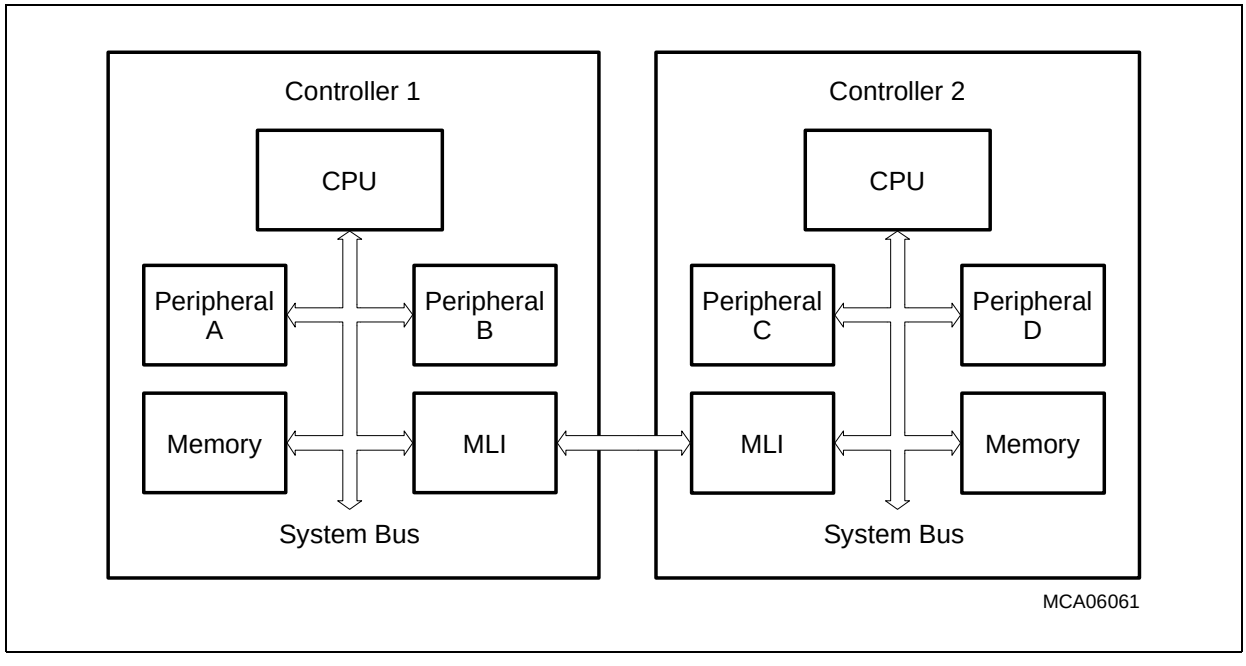


Figure 1-10 Typical Micro Link Interface Connection

Features

- Synchronous serial communication between an MLI transmitter and an MLI receiver
- Different system clock speeds supported in MLI transmitter and MLI receiver due to full handshake protocol (4 lines between a transmitter and a receiver)
- Fully transparent read/write access supported (= remote programming)
- Complete address range of target device available
- Specific frame protocol to transfer commands, addresses and data
- Error detection by parity bit
- 32-bit, 16-bit, or 8-bit data transfers supported
- Programmable baud rate: $f_{MLI}/2$ (max. $f_{MLI} = f_{SYS}$)
- Address range protection scheme to block unauthorized accesses
- Multiple receiving devices supported

Figure 1-11 shows a general block diagram of the MLI module.

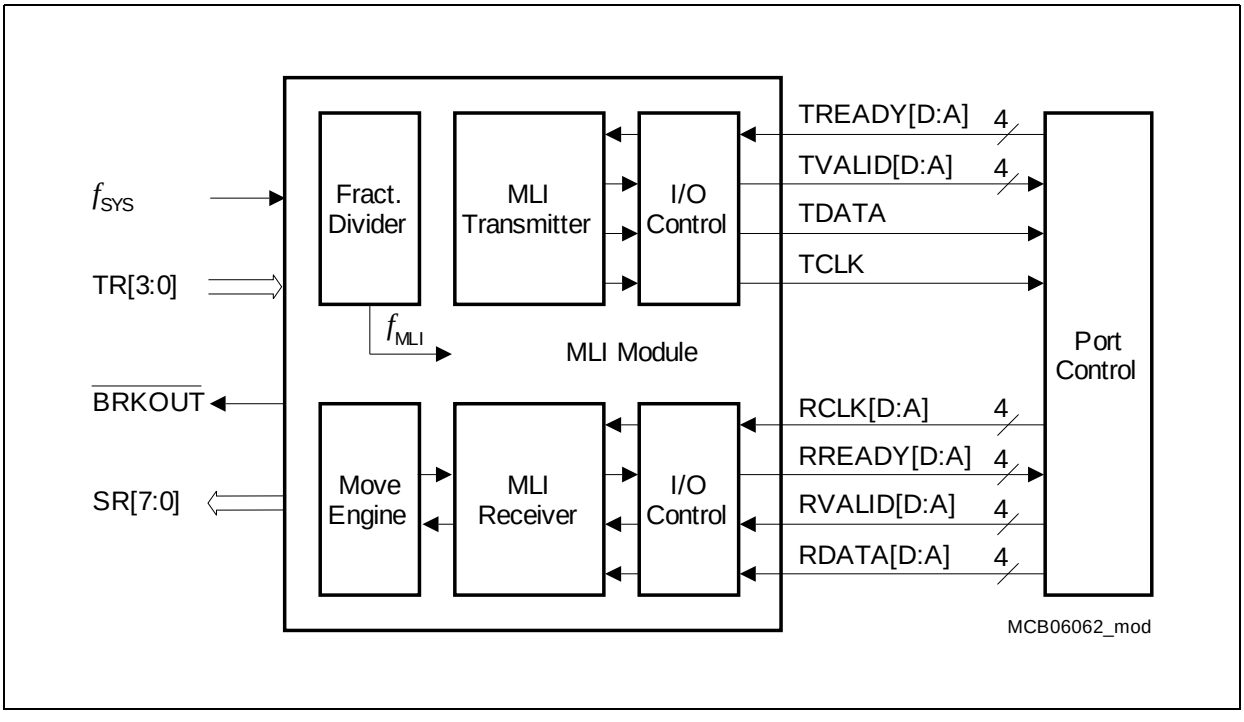


Figure 1-11 General Block Diagram of the MLI Modules

The MLI transmitter and MLI receiver communicate with other MLI receivers and MLI transmitters via a four-line serial connection each. Several I/O lines of these connections are available outside the MLI module kernel as a four-line output or input vector with index numbering A, B, C and D. The MLI module internal I/O control blocks define which signal of a vector is actually taken into account and also allow polarity inversions (to adapt to different physical interconnection means)

1.4.7 General Purpose Timer Array (GPTA)

The TC1797 contains the two General Purpose Timer Arrays (GPTA0 and GPTA1) with identical functionality, plus the additional Local Timer Cell Array (LTCA2). **Figure 1-12** shows a global view of the GPTA modules.

The GPTA provides a set of timer, compare, and capture functionalities that can be flexibly combined to form signal measurement and signal generation units. They are optimized for tasks typical of engine, gearbox, and electrical motor control applications, but can also be used to generate simple and complex signal waveforms required for other industrial applications.

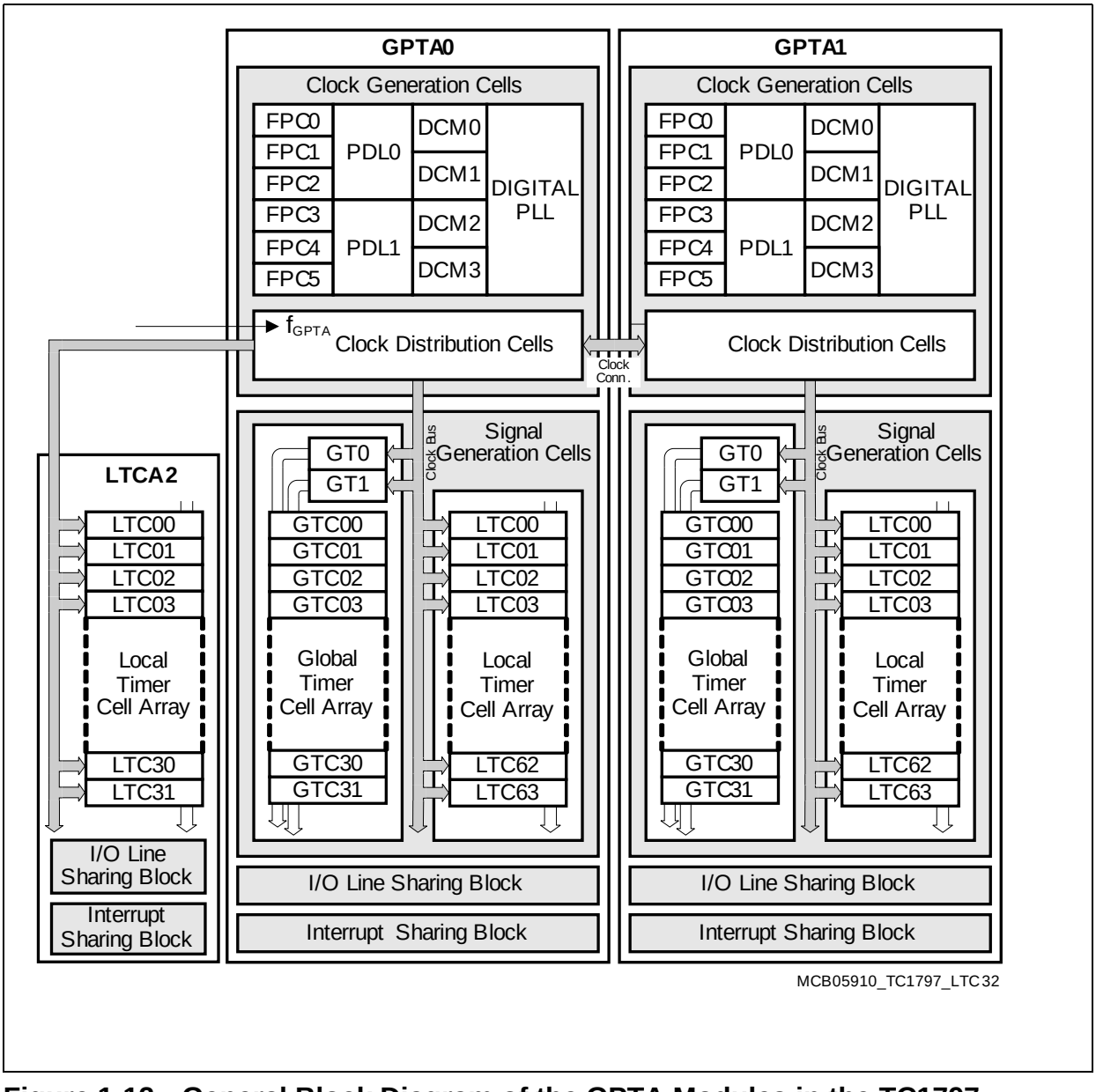


Figure 1-12 General Block Diagram of the GPTA Modules in the TC1797

1.4.7.1 Functionality of GPTA0 and GPTA1

The General Purpose Timer Arrays (GPTA0 and GPTA1) each provides a set of hardware modules required for high-speed digital signal processing:

- Filter and Prescaler Cells (FPC) support input noise filtering and prescaler operation.
- Phase Discrimination Logic units (PDL) decode the direction information output by a rotation tracking system.
- Duty Cycle Measurement Cells (DCM) provide pulse-width measurement capabilities.
- A Digital Phase Locked Loop unit (PLL) generates a programmable number of GPTA module clock ticks during an input signal's period.
- Global Timer units (GT) driven by various clock sources are implemented to operate as a time base for the associated Global Timer Cells.
- Global Timer Cells (GTC) can be programmed to capture the contents of a Global Timer on an external or internal event. A GTC may also be used to control an external port pin depending on the result of an internal compare operation. GTCs can be logically concatenated to provide a common external port pin with a complex signal waveform.
- Local Timer Cells (LTC) operating in Timer, Capture, or Compare Mode may also be logically tied together to drive a common external port pin with a complex signal waveform. LTCs – enabled in Timer Mode or Capture Mode – can be clocked or triggered by various external or internal events.
- On-chip Trigger and Gating Signals (OTGS) can be configured to provide trigger or gating signals to integrated peripherals (GPTA0 only).

Input lines can be shared by an LTC and a GTC to trigger their programmed operation simultaneously.

The following list summarizes the specific features of the GPTA units.

Clock Generation Unit

- Filter and Prescaler Cell (FPC)
  - Six independent units
  - Three basic operating modes:
    Prescaler, Delayed Debounce Filter, Immediate Debounce Filter
  - Selectable input sources:
    Port lines, GPTA module clock, FPC output of preceding FPC cell
  - Selectable input clocks:
    GPTA module clock, prescaled GPTA module clock, DCM clock, compensated or uncompensated PLL clock.
  - $f_{GPTA}/2$ maximum input signal frequency in Filter Modes
- Phase Discriminator Logic (PDL)
  - Two independent units
  - Two operating modes (2- and 3- sensor signals)

  - f_{GPTA}/4 maximum input signal frequency in 2-sensor Mode, f_{GPTA}/6 maximum input signal frequency in 3-sensor Mode
- Duty Cycle Measurement (DCM)
  - Four independent units
  - 0 - 100% margin and time-out handling
  - f_{GPTA} maximum resolution
  - f_{GPTA}/2 maximum input signal frequency
- Digital Phase Locked Loop (PLL)
  - One unit
  - Arbitrary multiplication factor between 1 and 65535
  - f_{GPTA} maximum resolution
  - f_{GPTA}/2 maximum input signal frequency
- Clock Distribution Unit (CDU)
  - One unit
  - Provides nine clock output signals:
    f_{GPTA}, divided f_{GPTA} clocks, FPC1/FPC4 outputs, DCM clock, LTC prescaler clock

Signal Generation Unit

- Global Timers (GT)
  - Two independent units
  - Two operating modes (Free-Running Timer and Reload Timer)
  - 24-bit data width
  - f_{GPTA} maximum resolution
  - f_{GPTA}/2 maximum input signal frequency
- Global Timer Cell (GTC)
  - 32 units related to the Global Timers
  - Two operating modes (Capture, Compare and Capture after Compare)
  - 24-bit data width
  - f_{GPTA} maximum resolution
  - f_{GPTA}/2 maximum input signal frequency
- Local Timer Cell (LTC)
  - 64 independent units
  - Three basic operating modes (Timer, Capture and Compare) for 63 units
  - Special compare modes for one unit
  - 16-bit data width
  - f_{GPTA} maximum resolution
  - f_{GPTA}/2 maximum input signal frequency

Interrupt Sharing Unit

- 286 interrupt sources, generating up to 92 service requests

On-chip Trigger Unit

- 16 on-chip trigger signals

I/O Sharing Unit

- Interconnecting inputs and outputs from internal clocks, FPC, GTC, LTC, ports, and MSC interface

1.4.7.2 Functionality of LTCA2

The Local Timer Cell Array (LTCA2) provides a set of hardware modules required for high-speed digital signal processing:

- Local Timer Cells (LTC) operating in Timer, Capture, or Compare Mode may also be logically tied together to drive a common external port pin with a complex signal waveform. LTCs – enabled in Timer Mode or Capture Mode – can be clocked or triggered by various external or internal events.

The following list summarizes the specific features of the LTCA unit.

The Local Timer Arrays (LTCA2) provides a set of hardware modules required for high-speed digital signal processing:

Signal Generation Unit

- Local Timer Cell (LTC)
  - 32 independent units
  - Three basic operating modes (Timer, Capture and Compare) for 63 units
  - Special compare modes for one unit
  - 16-bit data width
  - f_{GPTA} maximum resolution
  - $f_{GPTA}/2$ maximum input signal frequency

I/O Sharing Unit

- Interconnecting inputs and outputs from internal clocks, LTC, ports, and MSC interface

1.4.8 Analog-to-Digital Converters

The TC1797 includes three Analog to Digital Converter modules (ADC0, ADC1, ADC2) and one Fast Analog to Digital Converter (FADC).

1.4.8.1 ADC Block Diagram

The analog to digital converter module (ADC) allows the conversion of analog input values into discrete digital values based on the successive approximation method.
This module contains 3 independent kernels (ADC0, ADC1, ADC2) that can operate autonomously or can be synchronized to each other. An ADC kernel is a unit used to convert an analog input signal (done by an analog part) and provides means for triggering conversions, data handling and storage (done by a digital part).

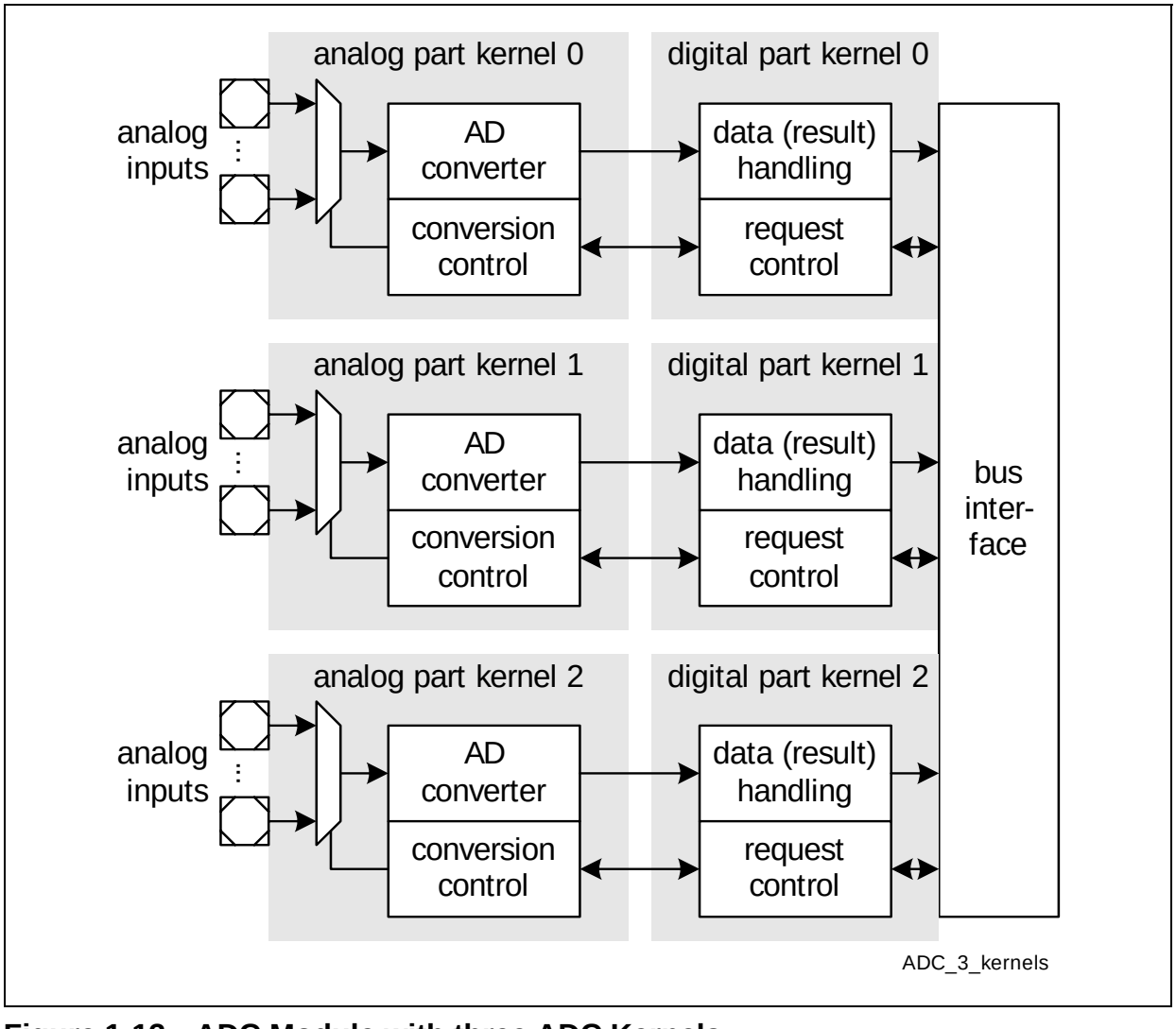


Figure 1-13 ADC Module with three ADC Kernels

Features of the analog part of each ADC kernel:

- Input voltage range from 0V to analog supply voltage
- Analog supply voltage range from 3.3 V to 5 V (single supply)
  (5V nominal supply voltage, performance degradation accepted for lower voltages)
- Input multiplexer width of 16 possible analog input channels (not all of them are necessarily available on pins)
- Performance for 12 bit resolution (@f_{ADCI} = 10 MHz):
  - conversion time about 2µs, TUE[1)] of ±4 LSB_{12} @ operating voltage 5 V
  - conversion time about 2µs, TUE of **tbd** LSB_{12} @ operating voltage 3.3 V
- V_{AREF} and 1 alternative reference input at channel 0
- Programmable sample time (in periods of f_{ADCI})
- Wide range of accepted analog clock frequencies f_{ADCI}
- Multiplexer test mode (channel 7 input can be connected to ground via a resistor for test purposes during run time by specific control bit)
- Power saving mechanisms

Features of the digital part of each ADC kernel:

- Independent result registers (16 independent registers)
- 5 conversion request sources (e.g. for external events, auto-scan, programmable sequence, etc.)
- Synchronization of the ADC kernels for concurrent conversion starts
- Control an external analog multiplexer, respecting the additional set up time
- Programmable sampling times for different channels
- Possibility to cancel running conversions on demand with automatic restart
- Flexible interrupt generation (possibility of DMA support)
- Limit checking to reduce interrupt load
- Programmable data reduction filter by adding conversion results
- Support of conversion data FIFO
- Support of suspend and power down modes
- Individually programmable reference selection for each channel (with exception of dedicated channels always referring to V_{AREF})

1) This value reflects the ADC module capability in an adapted electrical environment, e.g. characterized by “clean” routing of analog and digital signals and separation of analog and digital PCB areas, low noise on analog power supply (< 30mV), low switching activity of digital pins near to the ADC, etc.

1.4.8.2 FADC Short Description

General Features

- Extreme fast conversion, 21 cycles of f_{FADC} clock (262.5 ns @ f_{FADC} = 80 MHz)
- 10-bit A/D conversion (higher resolution can be achieved by averaging of consecutive conversions in digital data reduction filter)
- Successive approximation conversion method
- All differential input channels with impedance control
- All FADC inputs are overlaid with ADC1 inputs
- Each differential input channel can also be used as single-ended input
- Offset calibration support for each channel
- Programmable gain of 1, 2, 4, or 8 for each channel
- Free-running (Channel Timers) or triggered conversion modes
- Trigger and gating control for external signals
- Built-in Channel Timers for internal triggering
- Channel timer request periods independently selectable for each channel
- Selectable, programmable digital anti-aliasing and data reduction filter block with four independent filter units

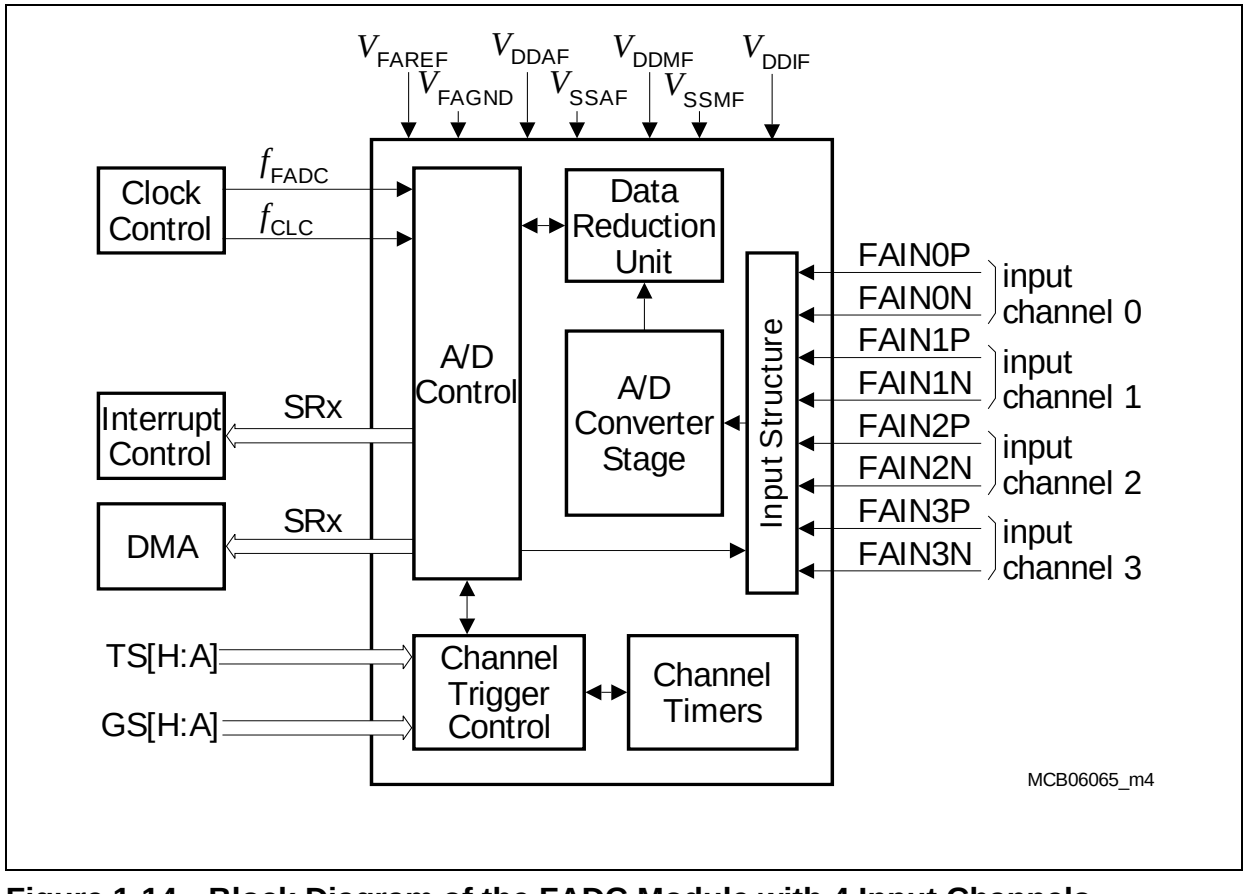


Figure 1-14 Block Diagram of the FADC Module with 4 Input Channels

As shown in **Figure 1-14**, the main FADC functional blocks are:

- An Input Structure containing the differential inputs and impedance control.
- An A/D Converter Stage responsible for the analog-to-digital conversion including an input multiplexer to select between the channel amplifiers
- A Data Reduction Unit containing programmable anti-aliasing and data reduction filters
- A Channel Trigger Control block determining the trigger and gating conditions for the FADC channels
- A Channel Timer for each channel to independently trigger the conversions
- An A/D Control block responsible for the overall FADC functionality

FADC Power Supply and References

The FADC module is supplied by the following power supply and reference voltage lines:

- V_{DDMF} / V_{SSMF}: FADC Analog Channel Amplifier Power Supply (3.3 V)
- V_{DDIF} / V_{SSMF}: FADC Analog Input Stage Power Supply (3.3 - 5 V),
  the V_{DDIF} supply does not appear as supply pin, because it is internally connected to the V_{DDM} supply of the ADC that is sharing the FADC input pins.
- V_{DDAF} / V_{SSAF}: FADC Analog Part Power Supply (1.5 V),
  to be fed in externally
- V_{FAREF} / V_{FAGND}: FADC Reference Voltage (3.3 V max.) and FADC Reference Ground

Input Structure

The input structure of the FADC in the TC1797 contains:

- A differential analog input stage for each input channel to select the input impedance (differential or single-ended measurement) and to decouple the FADC input signal from the pins.
- All input channels are overlaid with ADC1 input signals (AN24 - AN31).
- A channel amplifier for each input channel with a settling time (about 5µs) when changing the characteristics of an input stage (changing between unused, differential, single-ended N, or single-ended P mode).

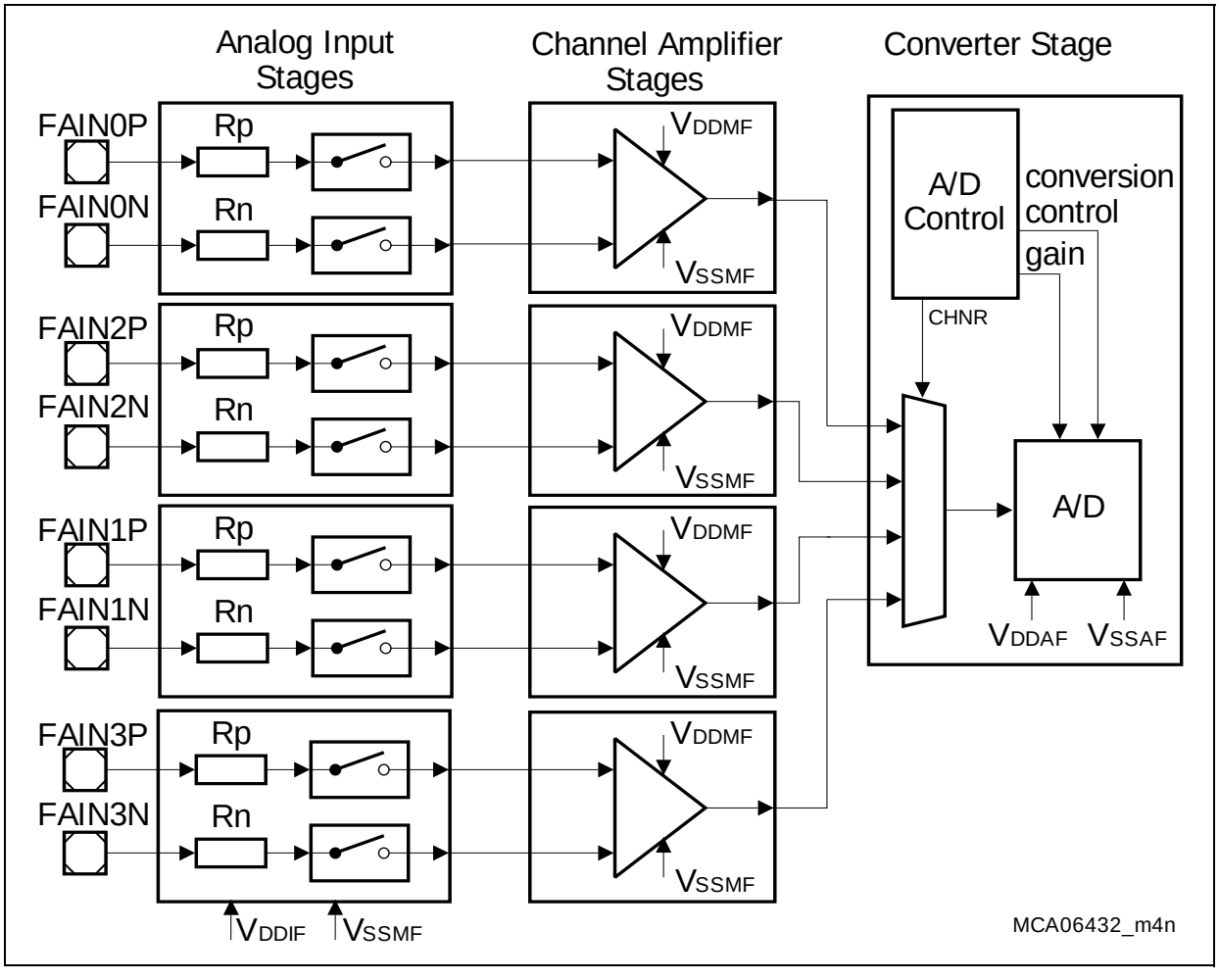


Figure 1-15 FADC Input Structure in TC1797

1.4.9 External Bus Interface

The External Bus Unit (EBU) of the TC1797 controls the accesses from peripheral units to external memories.

Features:

- 64-bit internal LMB interface
- 32-bit demultiplexed / 16-bit multiplexed external bus interface (3.3V, 2.5V)
  - Support for Intel-style and Motorola-style interface signals
  - Support for Burst Flash memory devices
  - Flexibly programmable access parameters
  - Programmable chip select lines
  - Little-endian support
- Examples for memories that has to be supported
  - Burst Flash:
  - Spansion: S29CD016, S29CD032
  - Spansion: S29CL032J1RFAM010 @3,3V
  - ST: M58BW016, M58BW032
  - ST: M58BW032GB B45ZA3T @3,3V
  - Flash (for 16 bit muxed mode):
  - http://www.spansion.com/products/Am29LV160B.html
  - SRAM (for 16 bit muxed mode):
  - http://www.idt.com/products/files/10372/71V016saautomotive.pdf
  - http://213.174.55.51/zmd.biz/pdf/ UL62H1616A.pdf
  - IDT 71V416YS15BEI
- Scalable external bus frequency
  - Derived from LMB frequency (f_{CPU}) divided by 1, 2, 3, or 4
  - Maximum 75 MHz[1)]
- Data buffering supported
  - Code prefetch buffer
  - Read/write buffer

1.5 On-Chip Debug Support (OCDS)

The TC1797 contains resources for different kinds of “debugging”, covering needs from software development to real-time-tuning. These resources are either embedded in specific modules (e.g. breakpoint logic of the TriCore) or part of a central peripheral (known as CERBERUS).

1) Maximum frequency of today available automotive Burst Flash devices.

1.5.1 On-Chip Debug Support

The classic software debug approach (start/stop, single-stepping) is supported by several features labelled “OCDS Level 1”:

- Run/stop and single-step execution independently for TriCore and PCP.
- Means to request all kinds of reset without usage of sideband pins.
- Halt-after-Reset for repeatable debug sessions.
- Different Boot modes to use application software not yet programmed to the Flash.
- A total of four hardware breakpoints for the TriCore based on instruction address, data address or combination of both.
- Unlimited number of software breakpoints (DEBUG instruction) for TriCore and PCP.
- Debug event generated by access to a specific address via the system bus.
- Tool access to all SFRs and internal memories independent of the Cores.
- Two central Break Switches to collect debug events from all modules (TriCore, PCP, DMA, BCU, break input pins) and distribute them selectively to breakable modules (TriCore, PCP, break output pins).
- Central Suspend Switch to suspend parts of the system (TriCore, PCP, Peripherals) instead if breaking them as reaction to a debug event.
- Dedicated interrupt resources to handle debug events inside TriCore (breakpoint trap, software interrupt) and Cerberus (can trigger PCP), e.g. for implementing Monitor programs.
- Access to all OCDS Level 1 resources also for TriCore and PCP themselves for debug tools integrated into the application code.
- Triggered Transfer of data in response to a debug event; if target is programmed to be a device interface simple variable tracing can be done.
- In depth performance analysis and profiling support given by the Emulation Device through MCDS Event Counters driven by a variety of trigger signals (e.g. cache hit, wait state, interrupt accepted).

1.5.2 Real Time Trace

For detailed tracing of the system’s behavior a pin-compatible Emulation Device will be available.[1)]

1.5.3 Calibration Support

Two main use cases are catered for by resources in addition the OCDS Level 1 infrastructure: Overlay of non-volatile on-chip memory and non-intrusive signaling:

- 8 KB SRAM for Overlay.
- Can be split into up to 16 blocks which can overlay independent regions of on-chip Data Flash.

1) The OCDS L2 interface of AudoNG is not available.

- Changing the configuration is triggered by a single SFR access to maintain consistency.
- Overlay configuration switch does not require the TriCore to be stopped or suspended.
- Invalidation of the Data Cache (maintaining write-back data) can be done concurrently with the same SFR.
- 256 KB additional Overlay RAM on Emulation Device.
- The 256 KB Trace memory of the Emulation Device can optionally be used for Overlay also.
- A dedicated trigger SFR with 32 independent status bits is provided to centrally post requests from application code to the host computer.
- The host is notified automatically when the trigger SFR is updated by the TriCore or PCP. No polling via a system bus is required.

1.5.4 Tool Interfaces

Three options exist for the communication channel between Tools (e.g. Debugger, Calibration Tool) and TC1797:

- Two wire DAP (Device Access Port) protocol for long connections or noisy environments.
- Four (or five) wire JTAG (IEEE 1149.1) for standardized manufacturing tests.
- CAN (plus software linked into the application code) for low bandwidth deeply embedded purposes.
- DAP and JTAG are clocked by the tool.
- Bit clock up to 40 MHz for JTAG, up to 80 MHz for DAP.
- Hot attach (i.e. physical disconnect/reconnect of the host connection without reset of the TC1797) for all interfaces.
- Infineon standard DAS (Device Access Server) implementation for seamless, transparent tool access over any supported interface.
- Lock mechanism to prevent unauthorized tool access to critical application code.

1.5.5 Self-Test Support

Some manufacturing tests can be invoked by the application (e.g. after power-on) if needed:

- Hardware-accelerated checksum calculation (e.g. for Flash content).
- RAM tests optimized for the implemented architecture.

1.5.6 FAR Support

To efficiently locate and identify faults after integration of a TC1797 into a system special functions are available:

- Boundary Scan (IEEE 1149.1) via JTAG and DAP.

- SSCM (Single Scan Chain Mode[1)]) for structural scan testing of the chip itself.

1) This function requires access to some device pins (e.g. $\overline{\text{TESTMODE}}$) in addition to those needed for OCDS.

2 CPU Subsystem

The TC1797 processor contains a TriCore 1.3.1 CPU. This chapter describes the implementation-specific options of the CPU, and should be read in conjunction with the TriCore Architecture Manual, which describes the complete TriCore Architecture including the register and instruction set.

2.1 TC1797 Processor Subsystem

The diagram below shows the block diagram of the TC1797 Processor subsystem.

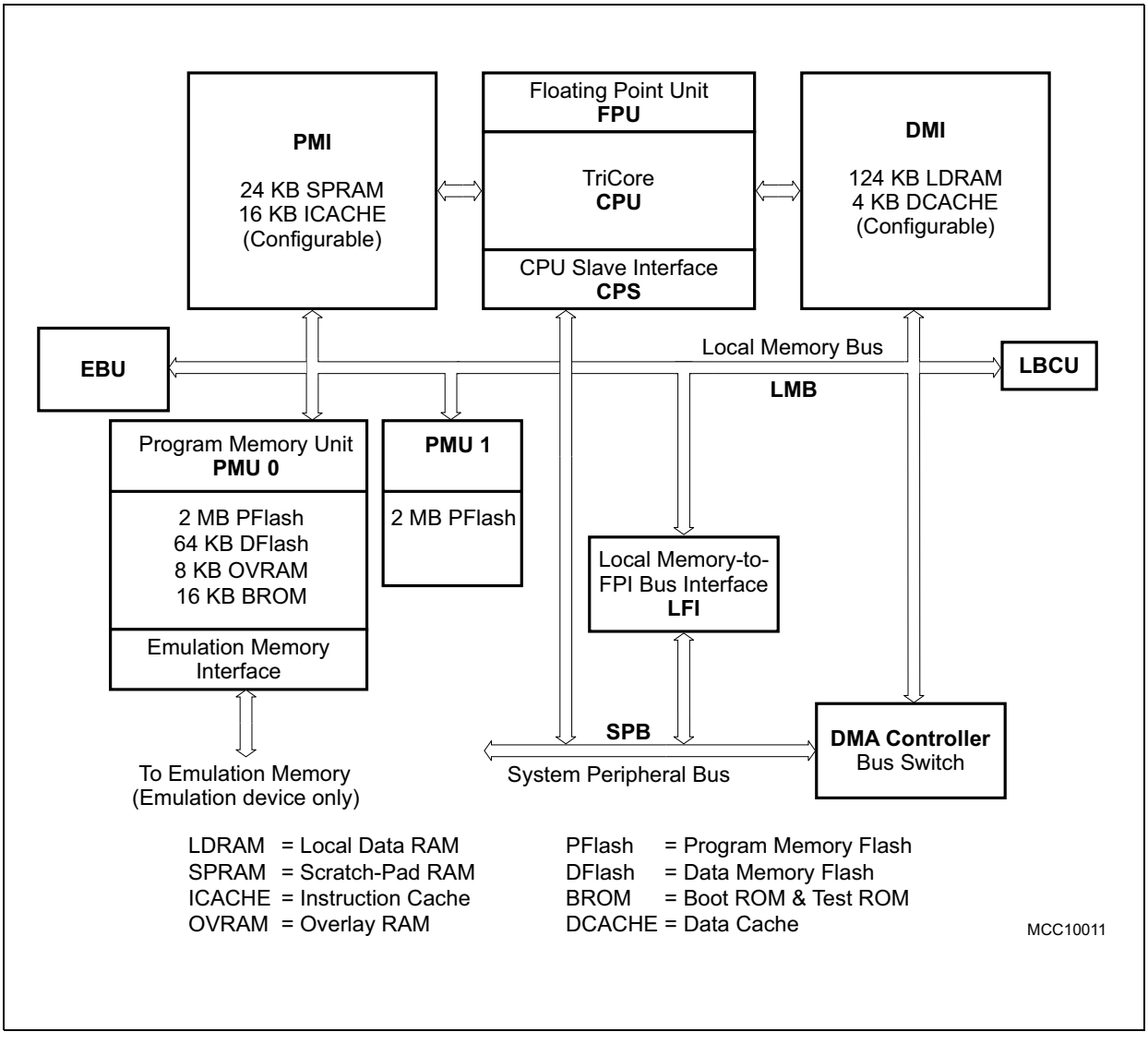


Figure 2-1 TC1797 Processor Subsystem Block Diagram

2.2 Central Processing Unit Features

The 180 MHz TriCore TC1797 CPU includes:

Architecture

- 32-bit load store architecture
- 4 Gbyte address range (2^{32})
- 16-bit and 32-bit instructions for reduced code size
- Data types:
  - Boolean, integer with saturation, bit array, signed fraction, character, double-word integers, signed integer, unsigned integer, IEEE-754 single-precision floating point
- Data formats:
  - Bit, byte (8-bits), half-word (16-bits), word (32-bits), double-word (64-bits)
- Byte and bit addressing
- Little-endian byte ordering for data, memory and CPU registers
- Multiply and Accumulate (MAC) instructions: Dual 16 × 16, 16 × 32, 32 × 32
- Saturation integer arithmetic
- Packed data
- Addressing modes:
  - Absolute, circular, bit reverse, long + short, base + offset with pre- and post-update
- Instruction types:
  - Arithmetic, address arithmetic, comparison, address comparison, logical, MAC, shift, coprocessor, bit logical, branch, bit field, load/store, packed data, system
- General Purpose Register Set (GPRS):
  - Sixteen 32-bit data registers
  - Sixteen 32-bit address registers
  - Three 32-bit status and program counter registers (PSW, PC, PCXI)
- Core Debug support (OCDS):
  - Level 1, supported in conjunction with the CPS block
  - Level 3, supported in conjunction with the MCDS block (Emulation Device only).

Implementation

- Most instructions executed in 1 cycle
- Branch instructions in 1, 2 or 3 cycles (using branch prediction)
- Shadow registers for fast context switch
- Automatic context save-on-entry and restore-on-exit for: subroutine, interrupt, trap
- Four memory protection register sets
- Dual instruction issuing (in parallel into Integer Pipeline and Load/Store Pipeline)
- Third pipeline for loop instruction only (zero overhead loop)
- Optional Floating Point instruction set implemented
- Optional Memory Management Unit (MMU) instruction set not implemented (Memory management configuration registers are always read as MMU not present)

2.2.1 CPU Diagram

The Central Processing Unit (CPU) comprises of an Instruction Fetch Unit, an Execution Unit, a General Purpose Register File (GPR), a CPU Slave interface (CPS), and Floating Point Unit (FPU).

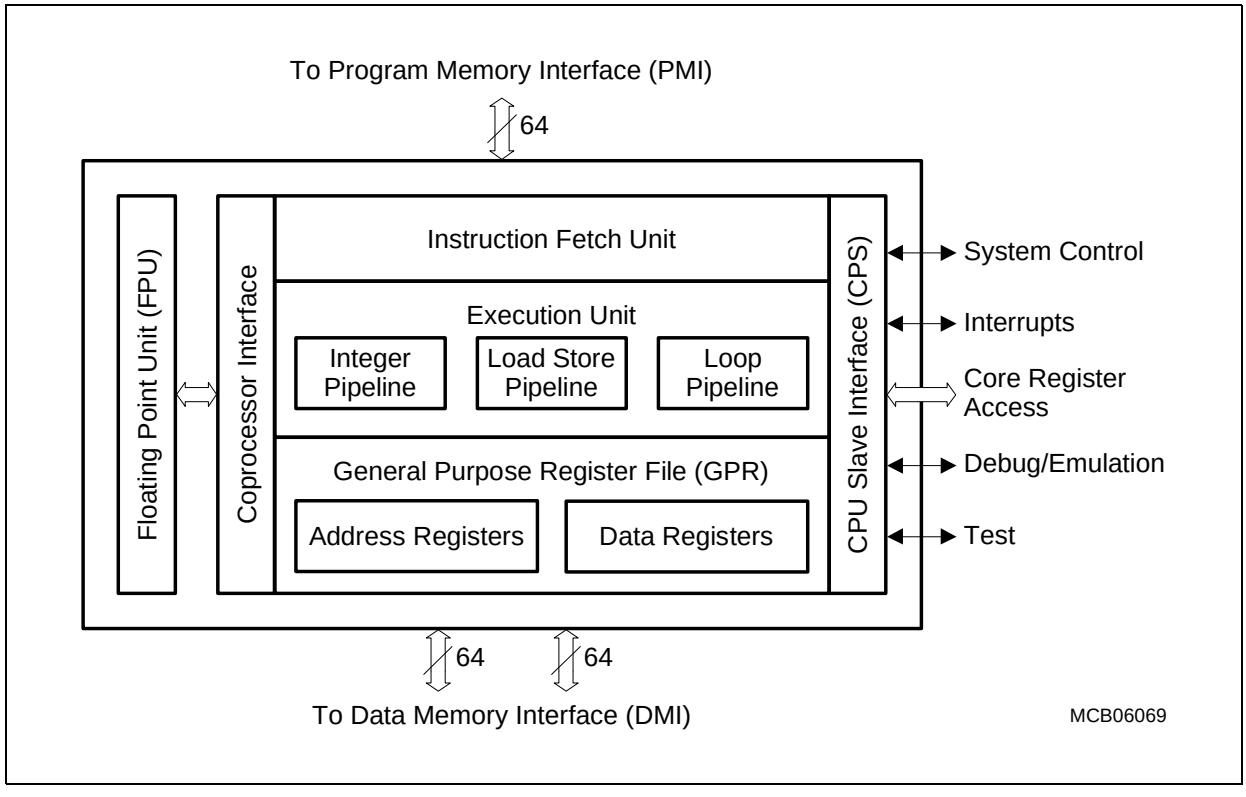


Figure 2-2 CPU Block Diagram

2.2.2 Instruction Fetch Unit

The Instruction Fetch Unit pre-fetches and aligns incoming instructions from the 64-bit wide Program Memory Interface (PMI). It contains an instruction pre-fetch buffer which may contain up to 128-bits of instructions linearly pre-fetched ahead of the current program counter. The Issue Unit directs the instruction to the appropriate pipeline.

The Instruction Protection Unit checks the validity of accesses to the PMI and also checks for instruction breakpoint conditions. The Program Counter Unit (PC) is responsible for updating the program counters.

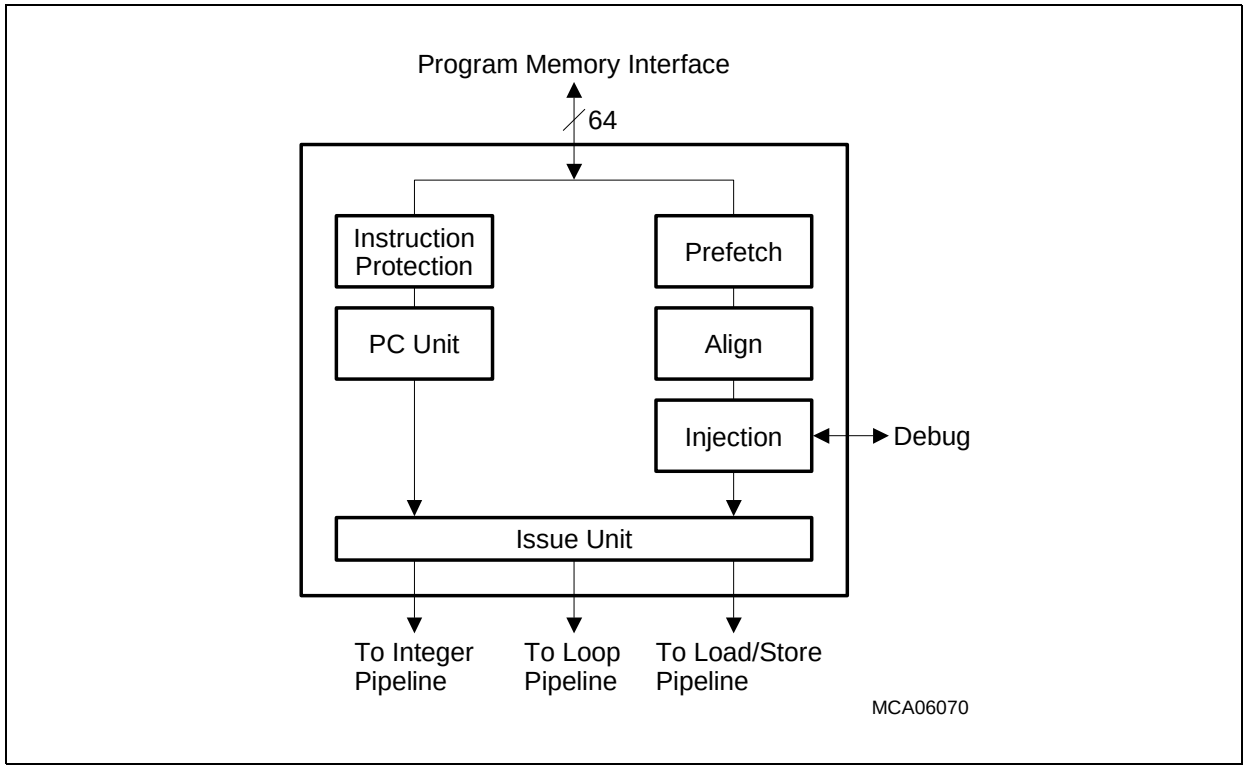


Figure 2-3 Instruction Fetch Unit

2.2.3 Execution Unit

The Execution Unit contains the Integer Pipeline, the Load/Store Pipeline and the Loop Pipeline.

The Integer Pipeline and Load/Store Pipeline have four stages: Fetch, Decode, Execute, and Write-back. The Execute stage may extend beyond one cycle to accommodate multi-cycle operations such as load instructions.

The Loop Pipeline has two stages: Decode and Write-back.

All three pipelines operate in parallel, permitting up to three instructions to be executed in one clock cycle.

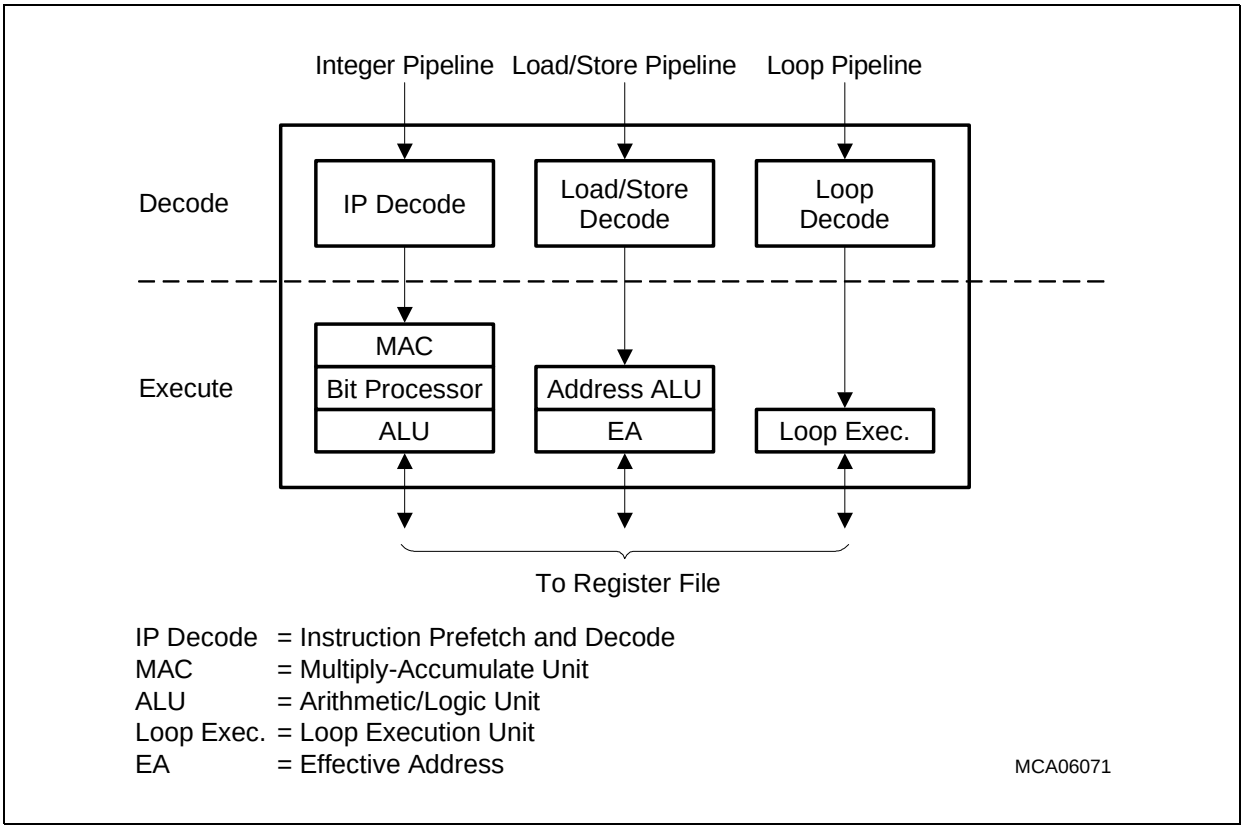


Figure 2-4 Execution Unit

2.2.4 General Purpose Register File

The CPU has a General Purpose Register (GPR) file, divided into an Address Register File (registers A0 through A15) and a Data Register File (registers D0 through D15).

The data flow for instructions issued to the Load/Store Pipeline is steered through the Address Register File.

The data flow for instructions issued to/from the Integer Pipeline and for data load/store instructions issued to the Load/Store Pipeline is steered through the Data Register File.

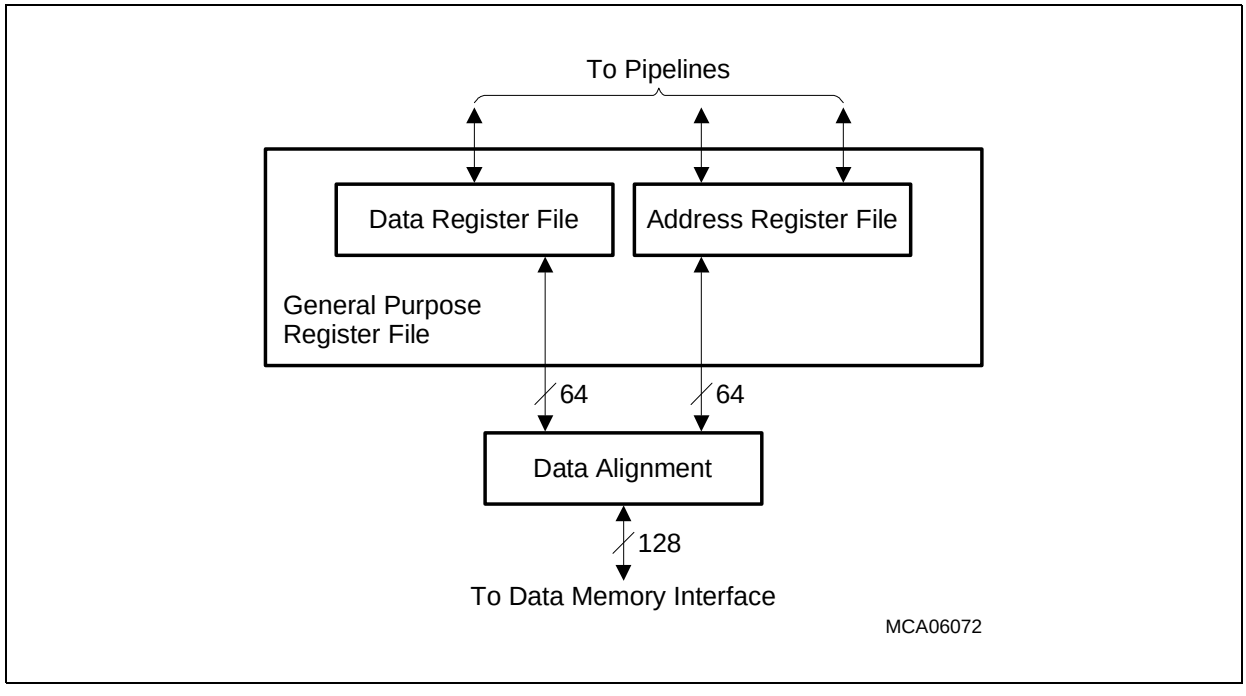


Figure 2-5 General Purpose Register File

2.3 CPU Implementation-Specific Features

This section describes the implementation-specific features of the TC1797 CPU. For a complete description of all registers, refer to the TriCore Architecture Manual.

2.3.1 Context Save Areas

In the TC1797, Context Save Areas (CSA) may be placed in LDRAM or cached external memory.

The TC1797 uses a uniform context-switching method for function calls, interrupts and traps. In all cases the Upper Context of the task is automatically saved and restored by hardware. Saving and restoring of the Lower Context may be optionally performed by software.

Fast context switching is enhanced by the unique memory subsystem design and the usage of shadow registers for the Upper Context. Shadow registers are automatically stored to and restored from memory when required and the presence of these registers is transparent to software.

The actual timing of context operations is dependent upon the placement of the Context Save Areas in either LDRAM or cached external memory and the state of the shadow registers.

CSA Placement in LDRAM

When they are not full, the shadow registers allow a complete Upper Context to be saved in as few as two clock cycles. When the shadow registers are full, the upper context save takes up to five cycles. On average an upper context save takes ~2.7 cycles.

CSA Placement in Cached External Memory

In this case, the timing is also dependent on the state of the Data Cache. The following values assume best case Data Cache operation (context saves do not incur a cache line writeback, context restores hit in the data cache): When they are not full, the shadow registers allow a complete Upper Context to be saved in as few as four clock cycles. When the shadow registers are full, the upper context save takes up to nine cycles. On average an upper context save takes ~5 cycles.

2.3.2 Program Counter Register - PC

The Program Counter (PC) holds the address of the instruction that is currently fetched and forwarded to the CPU pipelines. The CPU handles updates of the PC automatically.

Software can use the current value of the PC for various tasks, such as performing code address calculations. Reading the PC through software executed by the CPU must only be done with an MFCR instruction. Explicit writes to the PC through an MTCR instruction must not be done due to possible unexpected behavior of the CPU.

The CPU must not perform Load/Store instructions to the mapped address of the PC in Segment 15. A MEM trap will be generated in such a case. Bit 0 of the PC register is read-only and hard-wired to 0.

2.3.3 Interrupt System

An interrupt request can be generated by the TC1797 on-chip peripheral units, or it can be generated by external events. Requests can be targeted to either the CPU, or to the Peripheral Control Processor (PCP).

The TC1797 interrupt system evaluates service requests for priority and to identify whether the CPU (or PCP) should receive the request. The highest-priority service request is then presented to the CPU (or PCP) by way of an interrupt.

The term “interrupt” is used generally to mean an event directed to the CPU, while the term “service request” describes an event that can be directed to either the CPU or the PCP.

2.3.4 Trap System

The following traps have implementation-specific properties.

UOPC - Unimplemented Opcode (TIN 2)

The TC1797 UOPC trap is raised on optional MMU instructions, coprocessor two and coprocessor three instructions.

OPD - Invalid Operand (TIN 3)

The TC1797 CPU does not raise OPD traps.

DSE - Data Access Synchronous Error (TIN 2)

The Data Access Synchronous Bus Error (DSE) trap is generated by the DMI module when a load access from the CPU encounters certain error conditions, such as an LMB Bus error, or an out-of-range access to LDRAM. When a DSE trap is generated, the exact cause of the error can be determined by reading the DMI Synchronous Trap Flag Register, DMI_STR. For details of possible error conditions and the corresponding flag bits in DMI_STR, see **“DMI Trap Generation” on Page 2-82**.

DAE - Data Access Asynchronous Error (TIN 3)

The Data Access Asynchronous Error Trap (DAE) is generated by the DMI module when a store or cache management access from the CPU encounters certain error conditions, such as an LMB Bus error, or an out-of-range access to LDRAM. When a DAE trap is generated, the exact cause of the error can be determined by reading the DMI Asynchronous Trap Flag Register, DMI_ATR. For details of possible error conditions and the corresponding flag bits in DMI_ATR, see **“DMI Trap Generation” on Page 2-82**.

2.3.5 Memory Integrity Error Handling

The TriCore 1.3.1 contains integrated support for the detection and handling of memory integrity errors. The handling of memory integrity errors for the various memory types in TriCore 1.3.1 is as follows:

2.3.5.1 Program Side Memories

The program side memories of the TriCore 1.3.1 core support a programmable split between Scratchpad RAM (SPRAM) and Instruction Cache (ICACHE). Both SPRAM and ICACHE are unified within a single memory structure, known as Program Memory (PMEM). The PMEM of TriCore 1.3.1 is protected from memory integrity errors on a per-halfword basis. Any byte write access to the SPRAM from the LMB interface is converted to a halfword Read-Modify-Write sequence by the PMI module.

Scratchpad RAM (SPRAM)

The Scratchpad RAM of TriCore 1.3.1 is protected from memory integrity errors on a per-halfword basis. The SPRAM is parity protected, one parity bit is required per half-word stored. Even parity is used for TriCore 1.3.1. Parity protection of SPRAM is enabled by setting MIECON.PMIEE to one. When MIECON.PMIEE is zero all parity errors are ignored.

For instruction fetch requests from the TriCore CPU to SPRAM, the parity bits are read along with the data bits and an error signal is generated for each half-word. The error signals are passed to the core along with their corresponding instruction half-words. Whenever an attempt is made to issue an instruction containing an integrity error a synchronous PIE trap is raised. The trap handler is then responsible for correcting the memory entry and re-starting program execution.

For SPRAM read operations from the LMB interface, either from the DMI module or another LMB master agent, an access that results in the detection of a parity error in the requested data half-words causes a bus error to be returned for the bus transaction. Since the TriCore CPU may not be involved in the transaction, a separate error is also flagged to the SCU module to optionally generate an NMI trap back to the core.

Writes to program scratchpad memory are only ever performed from the bus interface. For write operations of half-word size or greater, the parity bit values are pre-computed based on the 16-bit granularity and written to the scratch memory in parallel with the data. For byte write operations the memory transaction is transformed into a half-word read-modify-write sequence inside the PMI module. As such, byte write operations may result in the detection of parity errors, which are handled as standard read operations.

Instruction Cache (ICACHE)

Since the instruction cache shares the same physical memory as the Scratchpad RAM, it is similarly configured for memory integrity error protection: a single parity bit is stored

per half-word and even parity is used. Parity protection of the instruction cache is enabled by setting MIECON.PMIEE to one. When MIECON.PMIEE is zero all parity errors are ignored.

For instruction fetch requests from the TriCore CPU to ICACHE, the parity bits are read along with the data bits of both cache ways, and an error signal generated for each half-word of each cache way. In the case of a tag hit the error signals for the corresponding cache way are passed to the core along with their corresponding instruction half-words. The corresponding program tag entry is invalidated such that the next attempt to fetch the instruction cache line will result in a refill. Whenever an attempt is made to issue an instruction containing an integrity error a synchronous PIE trap is raised. The trap handler is then responsible for checking the source of the memory integrity error, and, in the case of the instruction cache, may immediately return to re-fetch the now invalidated cache line.

Program Tag (PTag)

The program tag stores a 22-bit tag address and 1-bit valid field for each of the two cache ways in a set. As such the program tag is written with 23-bit granularity and a single parity bit is associated with each 23-bit tag way. Even parity is used. Parity protection of the program tag is enabled by setting MIECON.PTIEE to one. When MIECON.PTIEE is zero all parity errors are ignored.

For instruction fetch requests from the TriCore CPU to ICACHE, the program tag parity bits are read along with the data bits and an error flag is computed. A way hit is triggered only if the tag address comparison succeeds, the valid bit is set and no parity error in the associated tag way is detected, any other result is considered a miss. In the normal case where no error is detected in either cache way then the cache line is filled/refilled as normal. In the case where an error is detected the cache controller replacement algorithm forces the way indicating an error to be replaced. Since such errors are otherwise transparent to the TriCore CPU, the CCPIE-R counter is incremented to allow counting of such error corrections if required. In the case where one cache way flags a cache hit, and the other cache way detects a parity error, the parity error condition is masked and has no effect on the memory integrity error handling mechanisms.

2.3.5.2 Data Side Memories

The data side memories of the TriCore 1.3.1 core support a programmable split between Local Data RAM (LDRAM) and Data Cache (DCache). Both LDRAM and DCache are unified within a single memory structure, known as Data Memory (DMEM). The DMEM of TriCore 1.3.1 is protected from memory integrity errors on a per-halfword basis. Any byte write access to either LDRAM or DCache is converted to a halfword Read-Modify-Write sequence. The transformation of such byte accesses to atomic sequences is performed within the DMI rather than the CPU core itself. In normal operation isolated byte write transactions to the data memories result in no additional stall cycles.

Local Data RAM (LDRAM)

The Local Data RAM of TriCore 1.3.1 is protected from memory integrity errors on a per-halfword basis. The LDRAM is parity protected, one parity bit is required per half-word stored. Even parity is used for TriCore 1.3.1. Parity protection of LDRAM is enabled by setting MIECON.DMIEE to one. When MIECON.DMIEE is zero all parity errors are ignored.

For data load requests from the TriCore CPU to LDRAM, the parity bits are read along with the data bits and an error signal is generated for each half-word. If an error is detected associated with any of the data half-words passed to the core an error is flagged to the core. If such an error condition is detected an asynchronous DIE trap is raised. The trap handler is then responsible for correcting the memory entry, or for taking alternative action (such as system soft reset) if correction of the data is not possible.

For LDRAM read operations from the LMB interface, either from the PMI module or another LMB master agent, an access that results in the detection of a parity error in the requested data half-words causes a bus error to be returned for the bus transaction. Since the TriCore CPU may not be involved in the transaction, a separate error is also flagged to the SCU module to optionally generate an NMI trap back to the core.

For write operations to LDRAM of half-word size or greater, the check bits are pre-calculated and written to the memory in parallel with the data bits. For byte write operations the memory transaction is transformed into a half-word read-modify-write sequence inside the DMI module. As such, byte write operations may result in the detection of parity errors, which are handled as per standard read operations.

Data Cache (DCache)

Since the data cache shares the same physical memory as the Local Data RAM, it is similarly configured for memory integrity error protection: a single parity bit is stored per half-word and even parity is used. Parity protection of the instruction cache is enabled by setting MIECON.DMIEE to one. When MIECON.DMIEE is zero all parity errors are ignored.

For data load requests from the TriCore CPU to DCache, the parity bits are read along with the data bits of both cache ways, and an error flag computed for each half-word of each cache way. In the case where an error is detected with any of the requested data half-words in a cache way which has a corresponding tag hit, an error is flagged to the core. If such an error condition is detected an asynchronous DIE trap is raised. The trap handler is then responsible for correcting the memory entry, or for taking alternative action (such as system soft reset) if correction of the data is not possible.

For write operations of half-word size or greater, the check bits are pre-calculated and written to the memory in parallel with the data bits. For byte write operations the memory transaction is transformed into a half-word read-modify-write sequence inside the DMI module. As such, byte write operations may result in the detection of parity errors as for read operations.

For cache line writeback, error detection is performed as dirty data is transferred to the bus. In all cases (normal cache line eviction, cachex.xx instruction) where an error condition is detected in a valid cache line a DIE trap is raised. The trap handler is then responsible for taking corrective action (such as system soft reset) since correction of the data is not possible. Since the parity error may not be detected until the bus transaction is in progress, no attempt is made to abort the bus transaction.

Data Tag (DTag)

The data tag stores a 22-bit tag address for each of the two cache ways in a set. As such the data tags are written with 22-bit granularity and a single parity bit is associated with each 22-bit tag address. Even parity is used. Parity protection of the data tag is enabled by setting MIECON.DTIEE to one. When MIECON.DTIEE is zero, all parity errors are ignored.

For data load or store requests from the TriCore CPU to DCache, the data tag parity bits are read along with the data bits and an error flag is computed. A way hit is triggered only if the tag address comparison succeeds, the tag location is valid and no parity error in the associated tag way is detected, any other result is considered a miss. In the normal case where no error is detected in either tag way then the cache line is filled/refilled as normal. In the case of a cache miss where an error is detected in one of the tag ways and the cache line does not contain dirty data the cache controller replacement algorithm forces the way indicating an error to be replaced when the refill operation returns. Since such errors are otherwise transparent to the TriCore CPU, the CCDIE-R counter is incremented to allow counting of such error corrections if required. In the case where one cache way flags a cache hit, and the other way detects a parity error, the error condition is masked and has no effect on the memory integrity error handling mechanisms. If a cache miss occurs, with a parity error detected on the associated data tag way and dirty data detected, then an asynchronous DIE trap is signalled to the core and any writeback / refill sequence aborted. The trap handler is responsible for invalidating the cache line and processing any associated dirty data if possible, or taking other corrective action. Similar action is taken for forced cache writeback using the cache manipulation instructions.

2.3.6 TriCore 1.3 Compatibility

In order to allow code written for existing TriCore 1.3 based devices to be utilised without modification, a compatibility mode is included for both the program and data side memory integrity handling. This compatibility mode is enabled by setting the COMPAT.PIE/DIE bit(s) to one.

When the COMPAT.PIE/DIE bit is set, the memory integrity handling is disabled. Memory integrity errors are never flagged to the TriCore 1.3.1 core, such that:

- PIE/DIE traps are not generated
- The CCPIE-R/CCDIE-R counters are not updated

The one exception is that the MIECON bits are still required to be set in order to enable parity protection of the various memories.

When COMPAT.PIE/DIE is set along with the corresponding MIECON bit(s), any memory integrity error detected results in an error being flagged to the SCU module to optionally generate an NMI trap back to the core.

2.4 CPU Subsystem Registers

This section describes the implementation-specific features of the CPU Subsystem registers listed in **Table 2-1**. For complete descriptions of all registers refer to the TriCore Architecture Manual.

Table 2-1 CPU Subsystem Registers

Registers	Purpose	Description
Core Special Function Registers (CSFRs)	Program state information, context and stack management, interrupt and trap control, system control	see **Page 2-16**
CPU Slave Interface Registers (CPSs)	Software break control and software service request control	see **Page 2-22**
Core General Purpose Registers (GPRs)	Address and data	see **Page 2-26**
Core Special Function Memory Protection Registers (CSFRs)	Memory protection control and mode selection	see **Page 2-29**
New CSFRs	For TriCore 1.3.1 functionality.	see **Page 2-35**
Core Debug Registers	Debug control	see **Page 2-52**
Program Memory Interface Registers (PMI)	PMI instruction cache control and status	see **Page 2-72**
Data Memory Interface Registers (DMI)	DMI status and trap flags	see **Page 2-84**

2.4.1 CPU Core Special Function Registers (CSFR)

Figure 2-6 shows the CSFR registers of the TC1797.

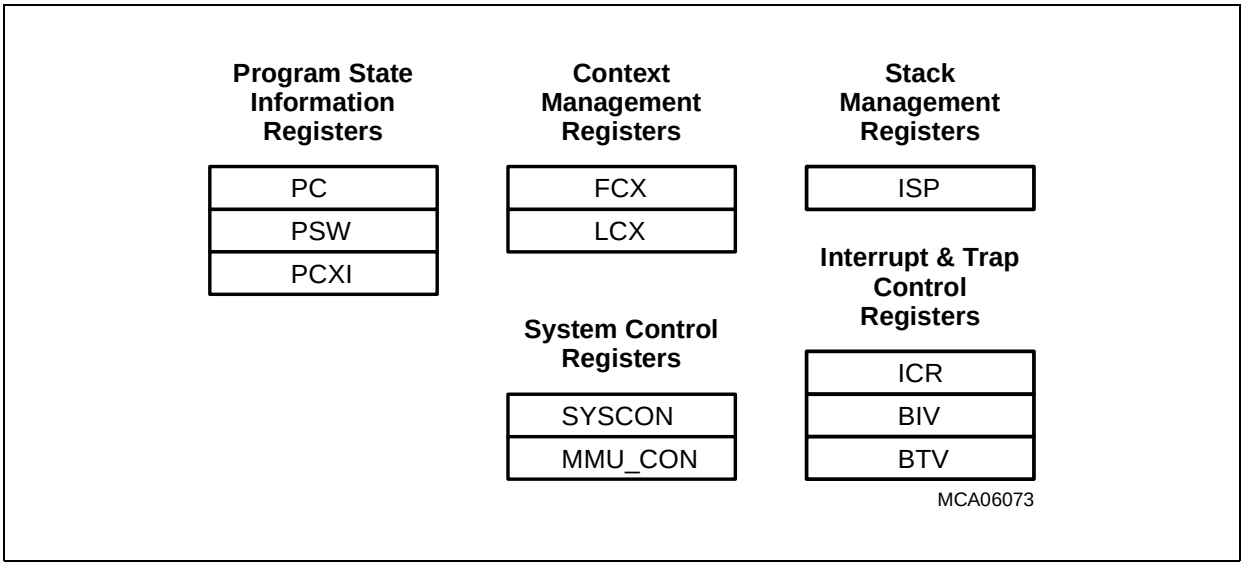


Figure 2-6 CSFR Registers

Table 2-2 Core Special Function Registers

Short Name	Description	Offset Address	Access Mode		Reset Value
			Read	Write	
MMU_CON	MMU Configuration Register	8000_H	U, SV, 32	SV, 32	Class 3 Reset $0000\ 8000_H$
PCXI	Previous Context Information Register	$FE00_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
PSW	Program Status Word Register	$FE04_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0B80_H$
PC	Program Counter Register	$FE08_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
SYSCON	System Configuration Register	$FE14_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
CPU_ID	CPU Identification Register	$FE18_H$	U, SV, 32	U, SV, 32, NC	Class 3 Reset $000A\ C006_H$
BIV	Interrupt Vector Table Pointer Register	$FE20_H$	U, SV, 32	SV, E, 32	Class 3 Reset $0000\ 0000_H$
BTV	Trap Vector Table Pointer Register	$FE24_H$	U, SV, 32	SV, E, 32	Class 3 Reset $A000\ 0100_H$

Table 2-2 Core Special Function Registers (cont'd)

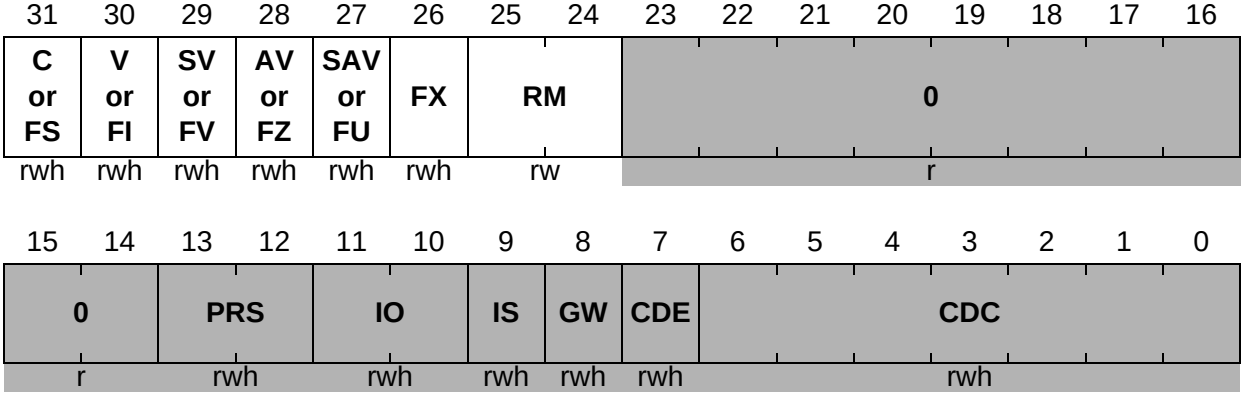
Short Name	Description	Offset Address	Access Mode		Reset Value
			Read	Write	
ISP	Interrupt Stack Pointer Register	$FE28_H$	U, SV, 32	SV, E, 32	Class 3 Reset $0000\ 0100_H$
ICR	ICU Interrupt Control Register	$FE2C_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
FCX	Free Context List Head Pointer Register	$FE38_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
LCX	Free Context List Limit Pointer Register	$FE3C_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$

2.4.1.1 Implementation-Specific Core Special Function Registers

The implementation-specific Program Status Word Register (PSW) is an extension of the PSW description in the TriCore Architecture Manual. The status flags used for FPU operations overlay the status flags used for Arithmetic Logic Unit (ALU) operations.

Program Status Word Register

PSW
Program Status Word Register (F7E1 FE04$_H$) Reset Value: 0000 0B80$_H$



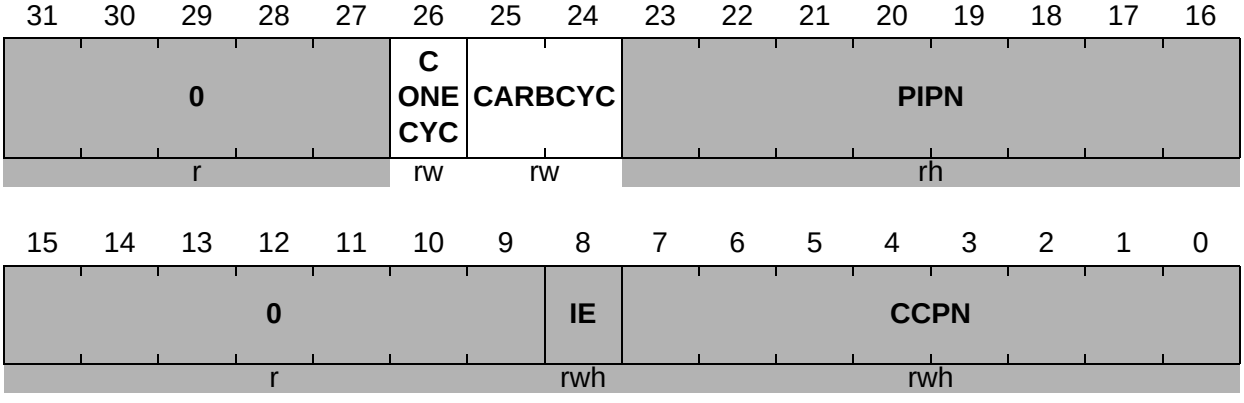
Field	Bits	Type	Description
RM	[25:24]	rw	**FPU Rounding Mode Selection**
FX	26	rwh	**FPU Inexact Flag**
SAV	27	rh	**Sticky Advance Overflow Flag**
FU		rwh	**FPU Underflow Flag**
AV	28	rwh	**Advance Overflow Flag**
FZ			**FPU Divide by Zero Flag**
SV	29	rwh	**Sticky Overflow Flag**
FV			**FPU Overflow Flag**
V	30	rwh	**Overflow Flag**
FI			**FPU Invalid Operation Flag**
C	31	rwh	**Carry Flag**
FS			**FPU Some Exception Flag**

Note: The non-shaded areas are implementation-specific bits/bit fields. The shaded areas are defined in the TriCore Architecture Manual.

Interrupt Control Register

The Interrupt Control Register (ICR) is an implementation-specific CFSR. Its Arbitration Cycle Control implementation-specific details are defined in bits 24 to 26.

ICR
Interrupt Control Register (F7E1 FE2C$_H$) Reset Value: 0000 0000$_H$



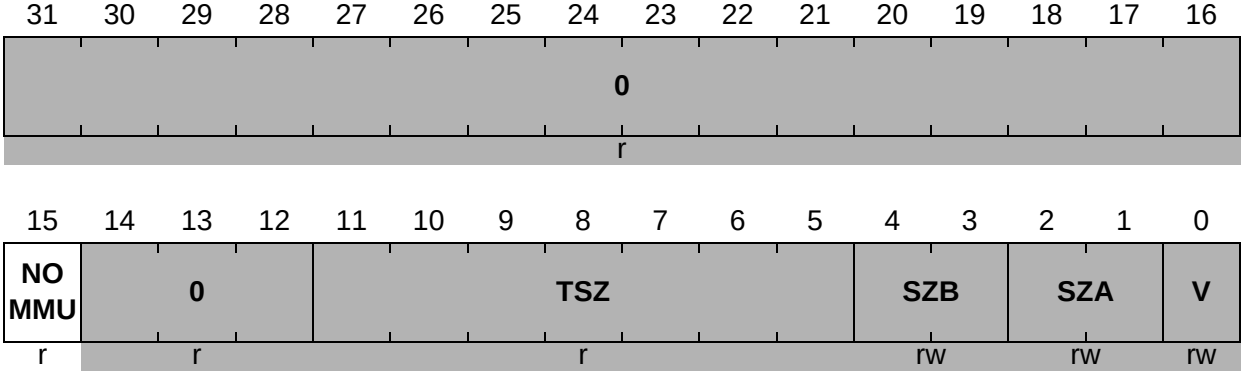
Field	Bits	Type	Description
CARBCYC	[25:24]	rw	**Number of Arbitration Cycles** CARBCYC controls the number of arbitration cycles used to determine the request with the highest priority. 00_B 4 arbitration cycles (default) 01_B 3 arbitration cycles 10_B 2 arbitration cycles 11_B 1 arbitration cycles
CONECYC	26	rw	**Number of Clocks per Arbitration Cycle Control** The CONECYC bit determines the number of system clocks per arbitration cycle. This bit should be set to 1 only for system designs utilizing low system clock frequencies. 0_B 2 clocks per arbitration cycle 1_B 1 clock per arbitration cycle

Note: The non-shaded areas are implementation-specific bits/bit fields. The shaded areas are defined in the TriCore Architecture Manual.

MMU Configuration Register

In the TC1797, the MMU Configuration Register (MMU_CON) register indicates the non-availability of the TriCore Memory Management Unit (bit NO MMU is always set).

MMU_CON
MMU Configuration Register (F7E1 8000$_H$) Reset Value: 0000 8000$_H$



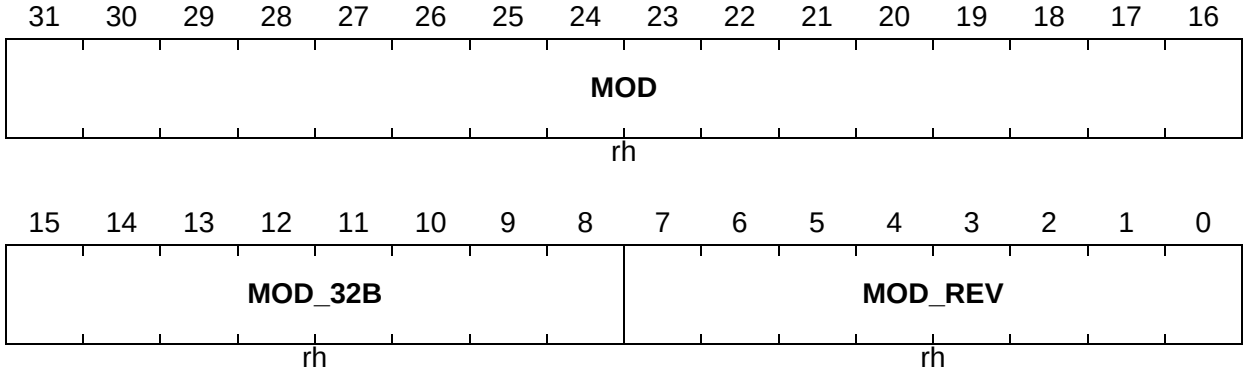
Field	Bits	Type	Description
NO MMU	15	r	**MMU Exists** 0_B MMU is available. 1_B MMU is not available. All other bits of MMU_CON are undefined. The MMU is not available in TC1797.

Note: The non-shaded areas are implementation-specific bits/bit fields. The shaded areas are defined in the TriCore Architecture Manual.

CPU Identification Register

CPU_ID
CPU Identification Register

(F7E1 FE18$_H$) **Reset Value: 000A C006$_H$**



Field	Bits	Type	Description
MOD_REV	[7:0]	rh	**Revision Number**
MOD_32B	[15:8]	rh	**32-Bit Module Enable** C0$_H$ A value of C0$_H$ in this field indicates a 32-bit module with a 32-bit module ID register.
MOD	[31:16]	rh	**Module Identification Number** 0A$_H$ For module identification

2.4.2 CPU Slave Interface (CPS) Registers

In the TC1797, the CPU Slave Interface (CPS) of the TriCore CPU directly accesses the interrupt service request registers in the CPU from the TC1797 System Peripheral Bus. The CPS registers are described in detail in the TriCore Architecture Manual.

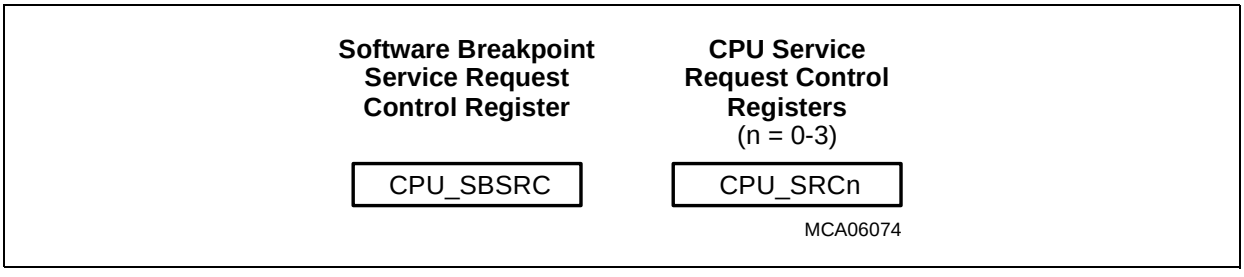


Figure 2-7 CPS Registers

Table 2-3 CPS Registers

Short Name	Description	Offset Address	Access Mode		Reset
			Read	Write	
CPS_ID	CPS Module Identification Register	$FF08_H$	U, SV, 32	U, SV, 32, NC	Class 3 Reset $0015\ C007_H$
CPU_SBSRC	CPU Software Breakpoint Service Request Control Register	$FFBC_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
CPU_SRC3	CPU Service Request Control 3 Register	$FFF0_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
CPU_SRC2	CPU Service Request Control 2 Register	$FFF4_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
CPU_SRC1	CPU Service Request Control 1 Register	$FFF8_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
CPU_SRC0	CPU Service Request Control 0 Register	$FFFC_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$

Note: The registers CPU_SBSRC and CPU_SRC[3:0] are not bit-addressable.

2.4.2.1 Implementation-Specific CPS Registers

All registers from **Table 2-3** have a TC1797-specific implementation detail, the Type of Service Control (TOS) bit/bit field.

CPU Service Request Control Register

CPU_SRCn (n = 0-3)
CPU Service Request Control Register n
(F7E0 FFFC$_H$-n*4) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SETR	CLRR	SRR	SRE	0	TOS	0		SRPN							
w	w	rh	rw	r	rw	r		rw							

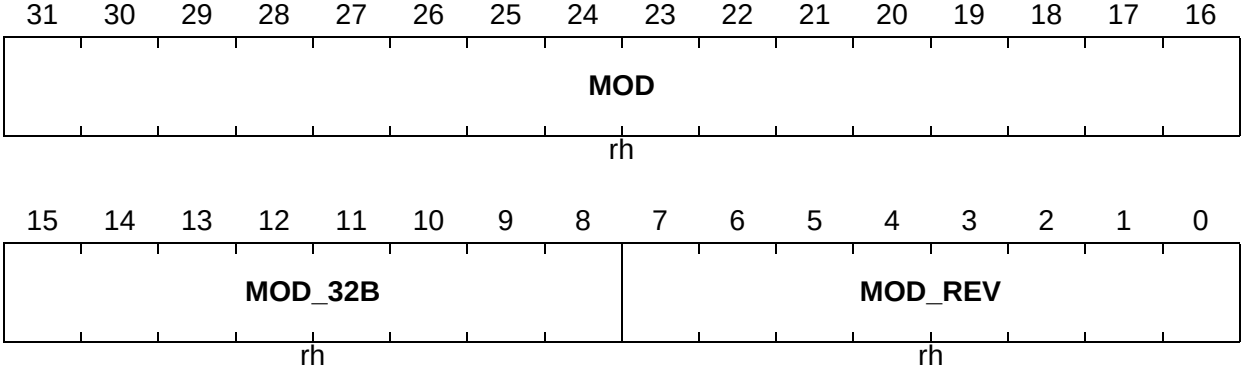
Field	Bits	Type	Description
TOS	10	rw	**Type of Service Control** 0_B Service Provider = CPU 1_B Service Provider = PCP2
0	11	r	**Reserved** Read as 0; should be written with 0.

Note: The non-shaded areas are implementation-specific bits/bit fields. The shaded areas are defined in the TriCore Architecture Manual.

CPS Module Identification Register

CPS_ID
CPS Module Identification Register

(F7E0 FF08$_H$) **Reset Value: 0015 C007$_H$**

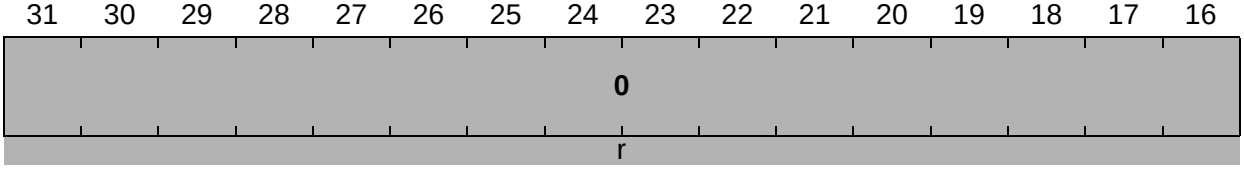


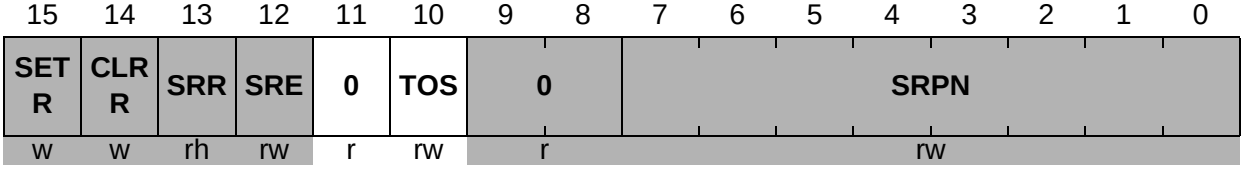
Field	Bits	Type	Description
MOD_REV	[7:0]	rh	**Revision Number**
MOD_32B	[15:8]	rh	**32-Bit Module Enable** $C0_H$ A value of $C0_H$ in this field indicates a 32-bit module with a 32-bit module ID register.
MOD	[31:16]	rh	**Module Identification Number** 15_H For module identification.

CPU Software Breakpoint Service Request Control Register

CPU_SBSRC
CPU Software Breakpoint Service Request Control Register

(F7E0 FFBC$_H$) **Reset Value: 0000 0000$_H$**





Field	Bits	Type	Description
TOS	10	rw	**Type of Service Control** 0_B Service Provider = CPU 1_B Reserved
0	11	r	**Reserved** Read as 0; should be written with 0.

Note: The non-shaded areas are implementation-specific bits/bit fields. The shaded areas are defined in the TriCore Architecture Manual.

2.4.3 CPU General Purpose Registers

Figure 2-8 shows the GPRs of the TC1797.

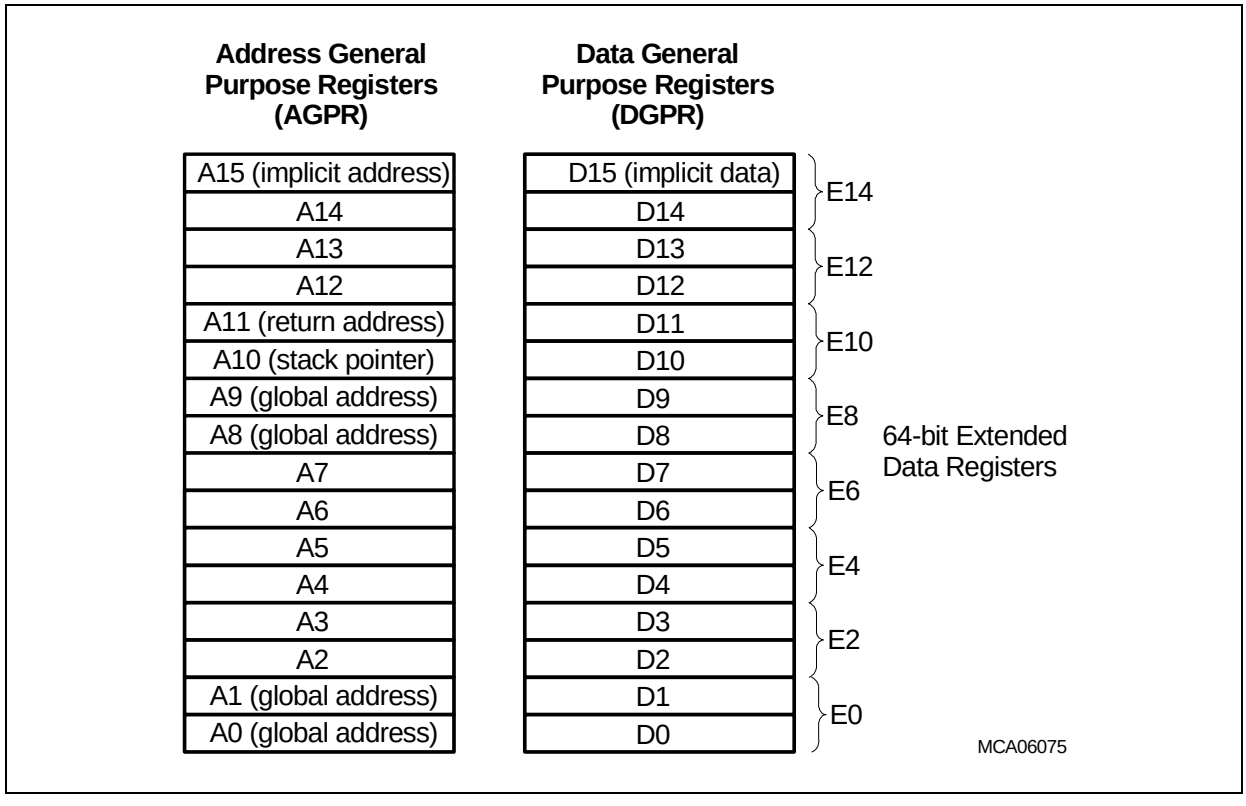


Figure 2-8 GPR Registers

Table 2-4 GPR Registers

Short Name	Description	Offset Address	Access Mode		Reset
			Read	Write	
D0	Data Register 0	$FF00_H$	U, SV, 32	SV, 32	Class 3 Reset XXXX XXXX$_H$
D1	Data Register 1	$FF04_H$	U, SV, 32	SV, 32	Class 3 Reset XXXX XXXX$_H$
D2	Data Register 2	$FF08_H$	U, SV, 32	SV, 32	Class 3 Reset XXXX XXXX$_H$
D3	Data Register 3	$FF0C_H$	U, SV, 32	SV, 32	Class 3 Reset XXXX XXXX$_H$
D4	Data Register 4	$FF10_H$	U, SV, 32	SV, 32	Class 3 Reset XXXX XXXX$_H$

Table 2-4 GPR Registers (cont'd)

Short Name	Description	Offset Address	Access Mode		Reset
			Read	Write	
D5	Data Register 5	$FF14_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
D6	Data Register 6	$FF18_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
D7	Data Register 7	$FF1C_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
D8	Data Register 8	$FF20_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
D9	Data Register 9	$FF24_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
D10	Data Register 10	$FF28_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
D11	Data Register 11	$FF2C_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
D12	Data Register 12	$FF30_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
D13	Data Register 13	$FF34_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
D14	Data Register 14	$FF38_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
D15	Data Register 15	$FF3C_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
A0	Address Register 0 (Global Address Register)	$FF80_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
A1	Address Register 1 (Global Address Register)	$FF84_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
A2	Address Register 2	$FF88_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
A3	Address Register 3	$FF8C_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
A4	Address Register 4	$FF90_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$

Table 2-4 GPR Registers (cont'd)

Short Name	Description	Offset Address	Access Mode		Reset
			Read	Write	
A5	Address Register 5	$FF94_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
A6	Address Register 6	$FF98_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
A7	Address Register 7	$FF9C_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
A8	Address Register 8 (Global Address Register)	$FFA0_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
A9	Address Register 9 (Global Address Register)	$FFA4_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
A10	Address Register 10 (Stack Pointer)	$FFA8_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
A11	Address Register 11 (Return Address)	$FFAC_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
A12	Address Register 12	$FFB0_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
A13	Address Register 13	$FFB4_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
A14	Address Register 14	$FFB8_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$
A15	Address Register 15	$FFBC_H$	U, SV, 32	SV, 32	Class 3 Reset $XXXX\ XXXX_H$

2.4.4 CPU Memory Protection Registers

As shown in **Figure 2-9**, there are four Memory Protection Register Sets in the TC1797, The sets specify memory protection ranges and permissions for code and data. The PSW.PRS bit field determines which of these sets is currently in use by the CPU. The Memory Protection Registers are Core Special function Registers, they are described in detail in the TriCore Architecture Manual.

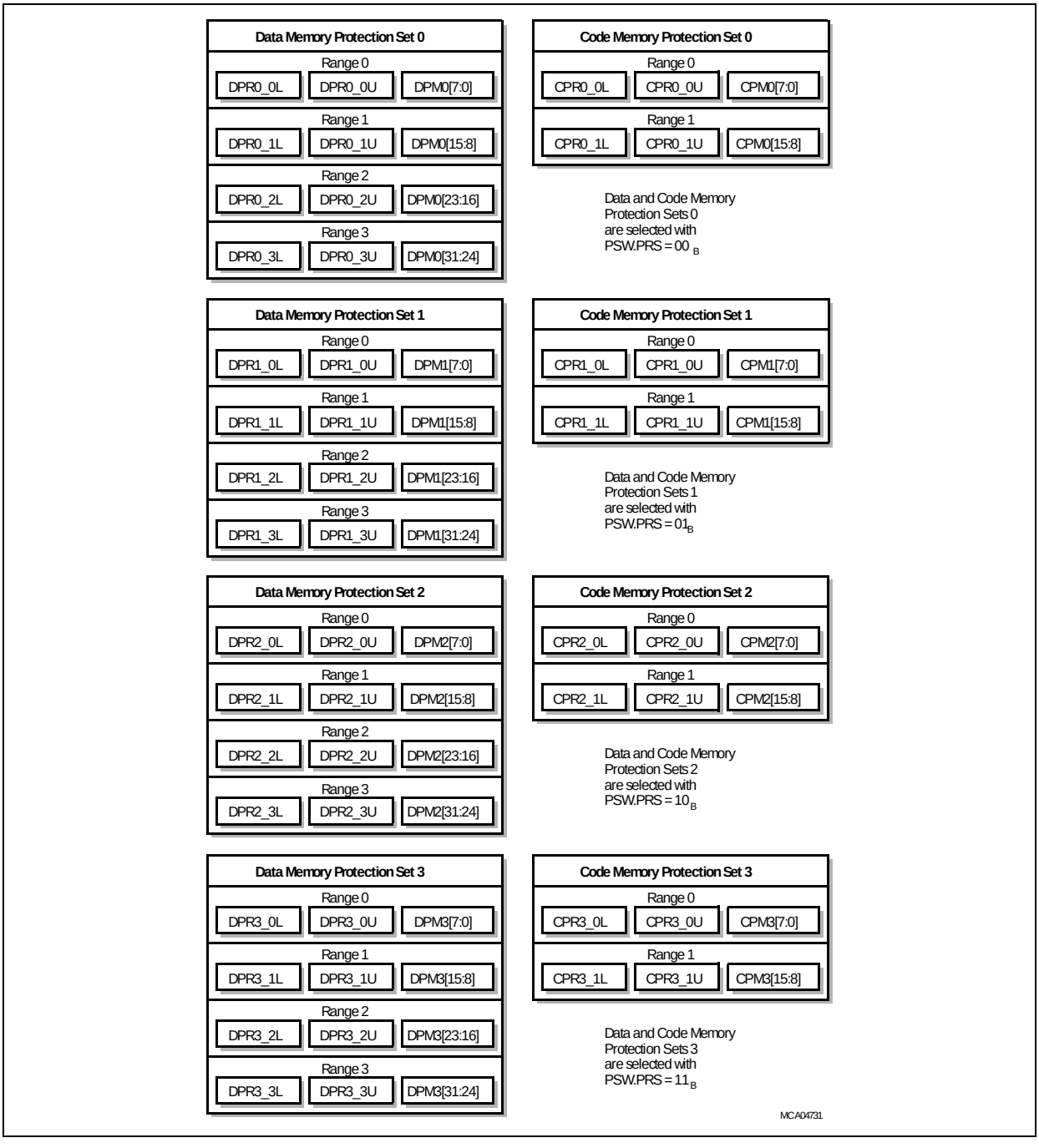


Figure 2-9 Memory Protection Register Sets of the TC1797

Table 2-5 Memory Protection Registers

Short Name	Description	Offset Address	Access Mode		Reset
			Read	Write	
DPR0_0L	Data Segment Protection Register Set 0, Range 0, Lower Boundary	$C000_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR0_0U	Data Segment Protection Register Set 0, Range 0, Upper Boundary	$C004_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR0_1L	Data Segment Protection Register Set 0, Range 1, Lower Boundary	$C008_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR0_1U	Data Segment Protection Register Set 0, Range 1, Upper Boundary	$C00C_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR0_2L	Data Segment Protection Register Set 0, Range 2, Lower Boundary	$C010_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR0_2U	Data Segment Protection Register Set 0, Range 2, Upper Boundary	$C014_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR0_3L	Data Segment Protection Register Set 0, Range 3, Lower Boundary	$C018_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR0_3U	Data Segment Protection Register Set 0, Range 3, Upper Boundary	$C01C_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR1_0L	Data Segment Protection Register Set 1, Range 0, Lower Boundary	$C400_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR1_0U	Data Segment Protection Register Set 1, Range 0, Upper Boundary	$C404_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR1_1L	Data Segment Protection Register Set 1, Range 1, Lower Boundary	$C408_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$

Table 2-5 Memory Protection Registers (cont'd)

Short Name	Description	Offset Address	Access Mode		Reset
			Read	Write	
DPR1_1U	Data Segment Protection Register Set 1, Range 1, Upper Boundary	$C40C_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR1_2L	Data Segment Protection Register Set 1, Range 2, Lower Boundary	$C410_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR1_2U	Data Segment Protection Register Set 1, Range 2, Upper Boundary	$C414_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR1_3L	Data Segment Protection Register Set 1, Range 3, Lower Boundary	$C418_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR1_3U	Data Segment Protection Register Set 1, Range 3, Upper Boundary	$C41C_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR2_0L	Data Segment Protection Register Set 2, Range 0, Lower Boundary	$C800_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR2_0U	Data Segment Protection Register Set 2, Range 0, Upper Boundary	$C804_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR2_1L	Data Segment Protection Register Set 2, Range 1, Lower Boundary	$C808_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR2_1U	Data Segment Protection Register Set 2, Range 1, Upper Boundary	$C80C_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR2_2L	Data Segment Protection Register Set 2, Range 2, Lower Boundary	$C810_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR2_2U	Data Segment Protection Register Set 2, Range 2, Upper Boundary	$C814_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$

Table 2-5 Memory Protection Registers (cont'd)

Short Name	Description	Offset Address	Access Mode		Reset
			Read	Write	
DPR2_3L	Data Segment Protection Register Set 2, Range 3, Lower Boundary	$C818_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR2_3U	Data Segment Protection Register Set 2, Range 3, Upper Boundary	$C81C_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR3_0L	Data Segment Protection Register Set 3, Range 0, Lower Boundary	$CC00_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR3_0U	Data Segment Protection Register Set 3, Range 0, Upper Boundary	$CC04_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR3_1L	Data Segment Protection Register Set 3, Range 1, Lower Boundary	$CC08_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR3_1U	Data Segment Protection Register Set 3, Range 1, Upper Boundary	$CC0C_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR3_2L	Data Segment Protection Register Set 3, Range 2, Lower Boundary	$CC10_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR3_2U	Data Segment Protection Register Set 3, Range 2, Upper Boundary	$CC14_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR3_3L	Data Segment Protection Register Set 3, Range 3, Lower Boundary	$CC18_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPR3_3U	Data Segment Protection Register Set 3, Range 3, Upper Boundary	$CC1C_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
CPR0_0L	Code Segment Protection Register Set 0, Range 0, Lower Boundary	$D000_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$

Table 2-5 Memory Protection Registers (cont'd)

Short Name	Description	Offset Address	Access Mode		Reset
			Read	Write	
CPR0_0U	Code Segment Protection Register Set 0, Range 0, Upper Boundary	$D004_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
CPR0_1L	Code Segment Protection Register Set 0, Range 1, Lower Boundary	$D008_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
CPR0_1U	Code Segment Protection Register Set 0, Range 1, Upper Boundary	$D00C_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
CPR1_0L	Code Segment Protection Register Set 1, Range 0, Lower Boundary	$D400_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
CPR1_0U	Code Segment Protection Register Set 1, Range 0, Upper Boundary	$D404_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
CPR1_1L	Code Segment Protection Register Set 1, Range 1, Lower Boundary	$D408_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
CPR1_1U	Code Segment Protection Register Set 1, Range 1, Upper Boundary	$D40C_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
CPR2_0L	Code Segment Protection Register Set 2, Range 0, Lower Boundary	$D800_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
CPR2_0U	Code Segment Protection Register Set 2, Range 0, Upper Boundary	$D804_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
CPR2_1L	Code Segment Protection Register Set 2, Range 1, Lower Boundary	$D808_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
CPR2_1U	Code Segment Protection Register Set 2, Range 1, Upper Boundary	$D80C_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$

Table 2-5 Memory Protection Registers (cont'd)

Short Name	Description	Offset Address	Access Mode		Reset
			Read	Write	
CPR3_0L	Code Segment Protection Register Set 3, Range 0, Lower Boundary	$DC00_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
CPR3_0U	Code Segment Protection Register Set 3, Range 0, Upper Boundary	$DC04_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
CPR3_1L	Code Segment Protection Register Set 3, Range 1, Lower Boundary	$DC08_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
CPR3_1U	Code Segment Protection Register Set 3, Range 1, Upper Boundary	$DC0C_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPM0	Data Protection Mode Register Set 0	$E000_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPM1	Data Protection Mode Register Set 1	$E080_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPM2	Data Protection Mode Register Set 2	$E0C0_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DPM3	Data Protection Mode Register Set 3	$E100_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
CPM0	Code Protection Mode Register Set 0	$E200_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
CPM1	Code Protection Mode Register Set 1	$E280_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
CPM2	Code Protection Mode Register Set 2	$E300_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
CPM3	Code Protection Mode Register Set 3	$E380_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$

2.5 CPU Core Special Function Registers (CSFRs)

The Core Special Function Registers (CSFRs) are listed below.

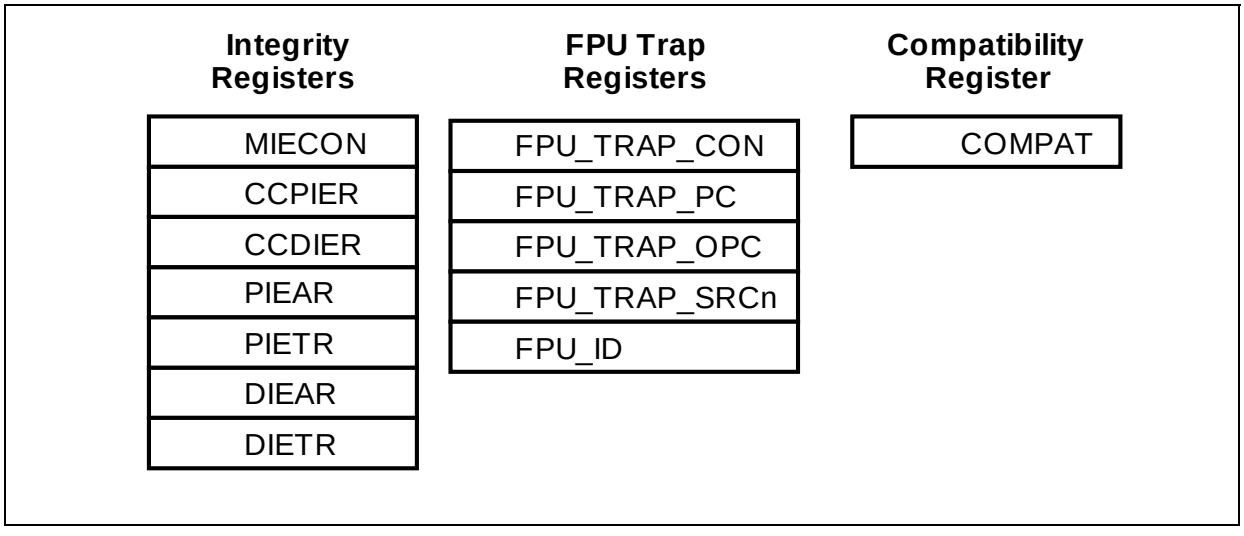


Figure 2-10 CSFR Registers

Table 2-6 Core Special Function Registers (CSFRs)

Short Name	Description	Offset Address	Access Mode		Reset
			Read	Write	
MIECON	Memory Integrity Error Control Register	9044_H	U, SV, 32	SV, E, 32	Class 3 Reset $0000\ 0000_H$
CCPIER	Count of Corrected Program Integrity Errors Register	9218_H	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
CCDIER	Count of Corrected Data Integrity Errors Register	9028_H	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
PIEAR	Program Integrity Error Address Register	9210_H	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
PIETR	Program Integrity Error Trap Register	9214_H	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DIEAR	Data Integrity Error Address Register	9020_H	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
DIETR	Data Integrity Error Trap Register	9024_H	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$

Table 2-6 Core Special Function Registers (CSFRs) (cont'd)

Short Name	Description	Offset Address	Access Mode		Reset
			Read	Write	
SMACON	SIST Mode Access Control Register	$900C_H$	U, SV, 32	SV, E, 32	Class 3 Reset $0000\ 0000_H$
COMPAT	Compatibility Control Register	9400_H	U, SV, 32	SV, E, 32	Class 3 Reset $FFFF\ FFFF_H$

Table 2-7 Floating Point Special Function Registers

Short Name	Description	Offset Address	Access Mode		Reset
			Read	Write	
FPU_TRAP_CON	Trap Control Register	$A000_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
FPU_TRAP_PC	Trapping Instruction Program Counter Register	$A004_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
FPU_TRAP_OPC	Trapping Instruction Opcode Register	$A008_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
FPU_TRAP_SRC1	Trapping Instruction Operand Register	$A010_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
FPU_TRAP_SRC2	Trapping Instruction Operand Register	$A014_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
FPU_TRAP_SRC3	Trapping Instruction Operand Register	$A018_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
FPU_ID	Trapping Identification Register	$A020_H$	U, SV, 32	SV, 32	Class 3 Reset $0054\ C003_H$

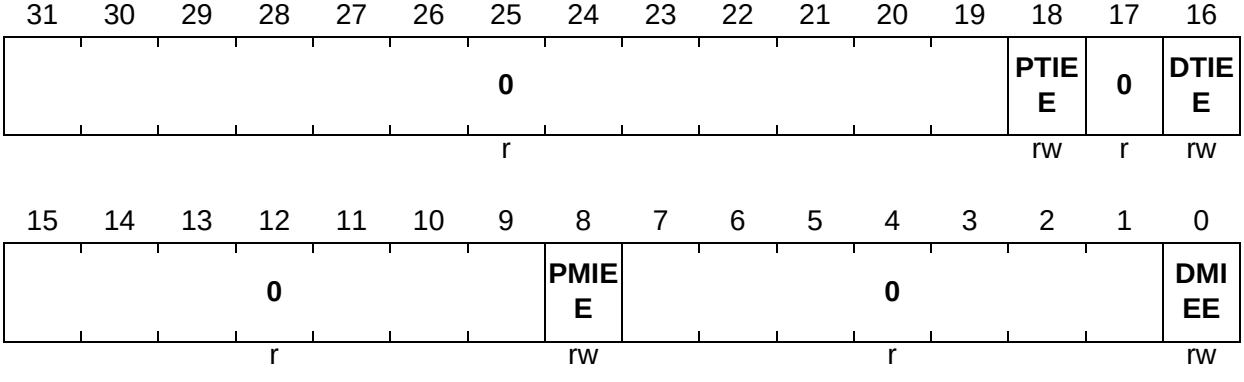
2.5.1 Register Descriptions

Memory Integrity Error Control Register

An architecturally visible register (MIECON) is introduced to allow software to control the memory integrity error detection / correction mechanisms. The existence of the MIECON register is architecturally defined. However, the fields within the MIECON are implementation specific.

The MIECON register may only be written in supervisor mode and is endinit protected.

MIECON
Memory Integrity Error Control Register
(F7E1 9044$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
DMIEE	0	rw	**Data Memory Integrity Error Enable** Enables integrity error handling for the Data Memories. 0_B Integrity error handling disabled - all memory accesses interpreted as error free. 1_B Integrity error handling enabled.
PMIEE	8	rw	**Program Memory Integrity Error Enable** Enables integrity error handling for the Program Memories. 0_B Integrity error handling disabled - all memory accesses interpreted as error free. 1_B Integrity error handling enabled.

Field	Bits	Type	Description
DTIEE	16	rw	**Data Tag Integrity Error Enable** Enables integrity error handling for the Data Tag. 0_B Integrity error handling disabled - all memory accesses interpreted as error free. 1_B Integrity error handling enabled.
0	17	r	**Reserved** Read as 0; should be written with 0.
PTIEE	18	rw	**Program Tag Integrity Error Enable** Enables integrity error handling for the Program Tag 0_B Integrity error handling disabled - all memory accesses interpreted as error free. 1_B Integrity error handling enabled
0	[7:1] [15:9], 17, [31:19]	r	**Reserved** Read as 0; should be written with 0.

2.5.1.1 Program Integrity Error Address Registers

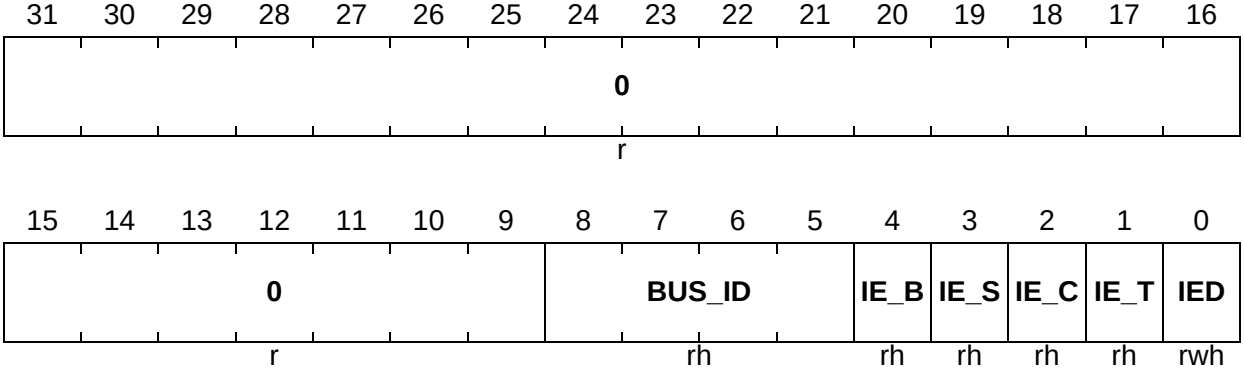
Two architecturally visible registers (PIETR, PIEAR) are introduced to allow software to localise the source of the last detected program integrity error. These registers are updated when an uncorrectable program integrity error condition is detected and the PIETR.IED bit is zero. On update the PIETR.IED bit is set to one and remains set until cleared by software. Whilst PIETR.IED is set further hardware updates of PIETR and PIEAR are inhibited.

PIETR and PIEAR are updated on any uncorrectable program integrity error condition detected, either during a bus access or a CPU instruction pre-fetch. Since instruction pre-fetches are speculative, the PIETR and PIEAR registers may be updated without a corresponding PIE trap.

The Program Integrity Error Trap Register (PIETR) contains flags to support software in localising the source of the last detected program integrity error. Where a program integrity error condition is detected during an instruction pre-fetch, the IE_S, IE_C and IE_T bits are updated to denote in which memory structure the error was detected, whilst BUS_ID and IE_B are cleared. Where the error is detected during a bus access, IE_B is set and BUS_ID updated to denote the master tag ID of the initiating bus master, whilst IE_S, IE_C and IE_T are cleared.

Program Integrity Error Trap Register (PIETR)

PIETR
Program Integrity Error Trap Register
(F7E1 9214$_H$) **Reset Value: 0000 0000$_H$**

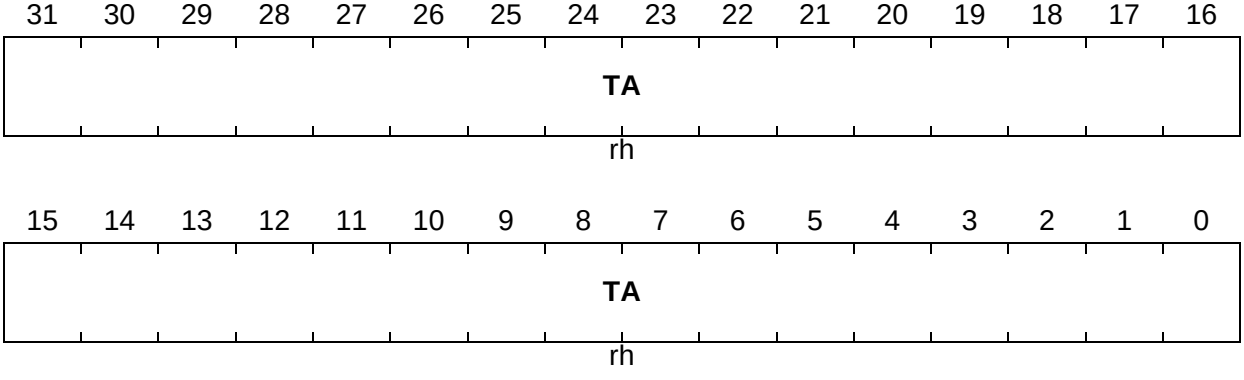


Field	Bits	Type	Description
IED	0	rwh	**Integrity Error Detected** Read Operation: 0_B No program integrity error condition occurred. 1_B Program integrity error condition detected. PIETR and PIEAR contents valid, further PIETR and PIEAR updates disabled. Write Operation: 0_B Clear IED bit, re-enable PIETR and PIEAR updates. 1_B No effect.
IE-T	1	rh	**Integrity Error - Tag Memory**
IE-C	2	rh	**Integrity Error - Cache Memory**
IE-S	3	rh	**Integrity Error - Scratchpad Memory**
IE-B	[4]	rh	**Integrity Error - Bus Access**
BUS_ID	[8:5]	rh	**Bus Master ID.** Bus Master Tag ID, where program integrity error is detected during bus access.
0	[31:9]	r	**Reserved** Read as 0; should be written with 0.

Program Integrity Error Address Register

This register contains the physical address accessed by the operation that encountered a program integrity error. This register is only updated if PIETR.IED is zero.

PIEAR
Program Integrity Error Address Register
(F7E1 9210$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
TA	[31:0]	rh	**Transaction Address** Physical address being accessed by operation that encountered program integrity error.

2.5.1.2 Data Integrity Error Information Registers

Two architecturally visible registers (DIETR, DIEAR) are introduced to allow software to localise the source of the last detected data integrity error. These registers are updated when an uncorrectable data integrity error condition is detected and the DIETR.IED bit is zero. On update the DIETR.IED bit is set to one and remains set until cleared by software. Whilst DIETR.IED is set further hardware updates of DIETR and DIEAR are inhibited.

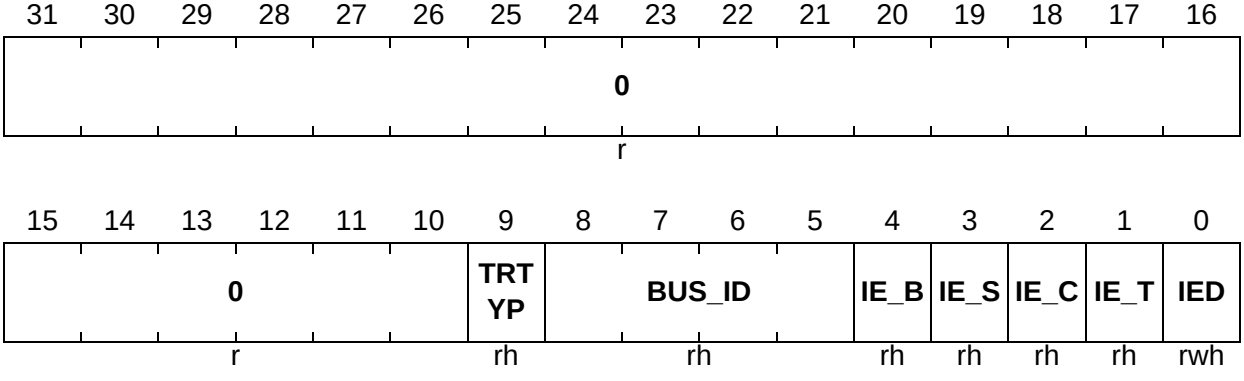
The Data Integrity Error Trap Register (DIETR) contains flags to support software in localising the source of the last detected data integrity error. Where a data integrity error condition is detected during a CPU Load/Store access, the IE_S, IE_C, IE_T and TRTYP bits are updated to denote in which memory structure the error was detected and the nature of the DIE trap, whilst BUS_ID and IE_B are cleared. Where the error is detected during a bus access, IE_B is set and BUS_ID updated to denote the master tag ID of the initiating bus master, whilst IE_S, IE_C, IE_T and TRTYP are cleared.

In TriCore 1.3.1 DIE traps are always asynchronous, independent of the type of transaction which encountered the data integrity error. The type of DIE trap is denoted by the DIETR.TYTYP bit. When a data integrity error is detected which causes an update of the DIE information registers and results in an asynchronous DIE trap, DIETR.IED = 1 and DIETR.TRTYP = 1, further asynchronous DIE traps are disabled until DIETR.IED is cleared by software.

Data Integrity Error Trap Register (DIETR)

DIETR
Data Integrity Error Trap Register

(F7E1 9024$_H$) **Reset Value: 0000 0000$_H$**



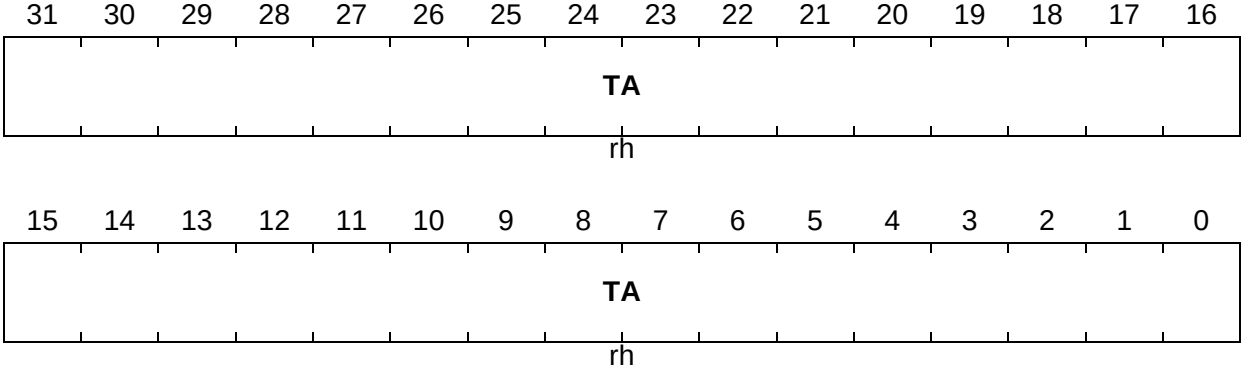
Field	Bits	Type	Description
IED	0	rwh	**Integrity Error Detected** Read Operation: 0_B No data integrity error condition occurred. 1_B Data integrity error condition detected. PIETR and DIEAR contents valid, further DIETR and DIEAR updates disabled. Write Operation: 0_B Clear IED bit, re-enable DIETR and DIEAR update. 1_B No effect.
IE-T	1	rh	**Integrity Error - Tag Memory**
IE-C	2	rh	**Integrity Error - Cache Memory**
IE-S	3	rh	**Integrity Error - Scratchpad Memory**
IE-B	4	rh	**Integrity Error - Bus Access**
BUS_ID	[8:5]	rh	**Bus Master ID** Bus Master Tag ID, where data integrity error is detected during bus access.
TRTYP	9	rh	**Trap Type** Type of trap generated, where data integrity error is detected during CPU Load/Store access. 0_B Synchronous Trap

Field	Bits	Type	Description
0	[31:10]	r	**Reserved** Read as 0; should be written with 0.

Data Integrity Error Address Register

This register contains the physical address accessed by the operation that encountered a data integrity error. This register is only updated if DIETR.IED is zero.

DIEAR
Data Integrity Error Address Register
(F7E1 9020$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
TA	[31:0]	rh	**Transaction Address** Physical address being accessed by operation that encountered data integrity error.

2.5.1.3 Software In-system Test Support

The TriCore 1.3.1 core protects against memory integrity errors by parity protection of the on-core memories. This has the side-effect of requiring memory blocks wider than the normal data access path to the memory. The additional parity storage bits are not easily accessible via the existing data paths, causing problems where SIST based testing of the memories is required. The TriCore 1.3.1 core also includes embedded memory arrays, such as the tag memories, which are not ordinarily accessible by the usual CPU datapaths. In order to address this problem, the TriCore 1.3.1 core includes improved SIST support, allowing all on-core memory arrays to be accessed.

Hidden Memory Arrays

The mapping of hidden memory arrays into the TriCore address space and the enabling of other SIST related features is controlled by the setting of bits within the SIST Mode Access Control Register (SMACON). The SMACON register is architecturally defined, however the fields within the SMACON and the effect of the fields on the memory map of the processor are implementation specific.

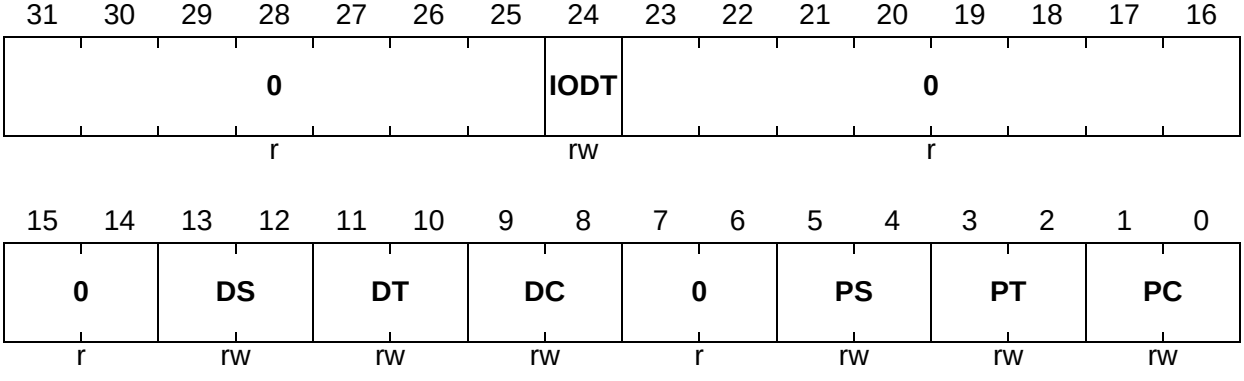
The hidden memory arrays are mapped into the program and data scratch areas (Segments C_H and D_H) of the address map by setting bits in the SMACON register. Program side embedded memories are mapped into the program scratchpad area and located in the address range $C01C0000_H$ - $C01FFFFF_H$ (and mirrored at $D41C0000_H$ - $D41FFFFF_H$). All other embedded memories are mapped into the data scratchpad area and located in the address range $D01C0000_H$ - $D01FFFFF_H$. A memory that has been mapped into the scratch memory area using the SMACON register may not be accessed in its normal operational mode.

The SMACON register is a CSFR with offset address $900C_H$. It may only be written in supervisor mode and is endinit protected.

SIST Mode Access Control Register

SMACON
SIST Mode Access Control Register

(F7E1 900C$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
PC	[1:0]	rw	**Instruction Cache Memory SIST mode access control[1)]** 00_B Normal Operation, No Mapping. 01_B $1x_B$ Instruction cache memory configured as program SPR.
PT	[3:2]	rw	**Program Tag Memory SIST mode access control** 00_B Normal Operation, No Mapping. 01_B Data Array Mapping, no error detection/correction. 10_B Check Array Mapping, no error detection/correction. 11_B Data Array Mapping, error detection/correction enabled.
PS	[5:4]	rw	**Program Scratch Memory SIST mode access control** 00_B Normal Operation, No Mapping. 01_B Data Array Mapping, no error detection/correction. 10_B Check Array Mapping, no error detection/correction. 11_B Data Array Mapping, error detection/correction enabled.

Field	Bits	Type	Description
DC	[9:8]	rw	**Data Cache Memory SIST mode access control** 00_B Normal Operation, No Mapping. 01_B $1x_B$ Data cache memory configured as Data SPR.
DT	[11:10]	rw	**Data Tag Memory SIST mode access control** 00_B Normal Operation, No Mapping. 01_B Data Array Mapping, no error detection/correction. 10_B Check Array Mapping, no error detection/correction. 11_B Data Array Mapping, error detection/correction enabled.
DS	[13:12]	rw	**Data Scratch Memory SIST mode access control** 00_B Normal Operation, No Mapping, Performance Optimised. 01_B Data Array Mapping, no error detection/correction. 10_B Check Array Mapping, no error detection/correction. 11_B Data Array Mapping, error detection/correction enabled.
IODT	24	rw	**In-Order Data Transactions** 0_B Normal operation, Non-dependent loads bypass stores. 1_B In-order operation, Loads always flush preceding stores, processor store buffer disabled.
0	[7:6] [23:14] [31:25]	r	**Reserved** Read as 0; should be written with 0.

1) When the Flash Read Protection mechanism is active, the value of SMACON.PC is overridden and treated as 00_B 'normal operating mode'; however the field can still be read and written normally.

Control Fields

The control fields within the SMACON register allow individual control of the local memories. Each memory may be mapped to operate in a number of different modes.

Normal operation, No Mapping

No mapping of the memories is performed and normal operation is possible. Embedded memories not usually directly addressable are not accessible in the system address map. Performance optimisations are enabled such that loads may read from the physical memory or associated write buffers.

Data Array Mapping, no error detection/correction

The data array (only) of the memory is made visible in the address map. Writes to the memory will not affect the check bits. Error correction/detection for the memory is disabled. Performance optimisations are disabled such that memory accesses are guaranteed to be performed to the actual memory.

Check Array Mapping, no error detection/correction

The check bit array (only) of the memory is made visible in the address map. Writes to the memory will not affect the data bits. Error correction/detection for the memory is disabled. Performance optimisations are disabled such that memory accesses are guaranteed to be performed to the actual memory.

Data Array Mapping, error detection/correction enabled

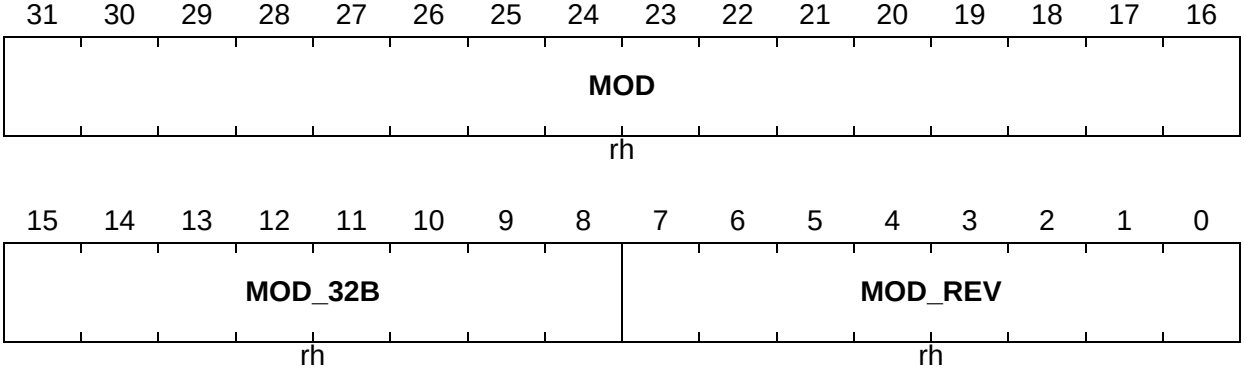
The data array of the memory is made visible in the address map. Writes to the memory will update the check bits as per normal operation. Error correction/detection for the memory is enabled. Performance optimisations are disabled such that memory accesses are guaranteed to be performed to the actual memory.

2.5.1.4 Floating Point CSFR Register

FPU Identification Register

FPU_ID
Trapping Identification Register

(F7E1 A020$_H$) **Reset Value: 0054 C003$_H$**



Field	Bits	Type	Description
MOD_REV	[7:0]	rh	**Revision Number**
MOD_32B	[15:8]	rh	**32-Bit Module Enable** $C0_H$ A value of $C0_H$ in this field indicates a 32-bit module with a 32-bit module ID register.
MOD	[31:16]	rh	**Module Identification Number** 54_H For module identification.

2.5.1.5 Compatibility

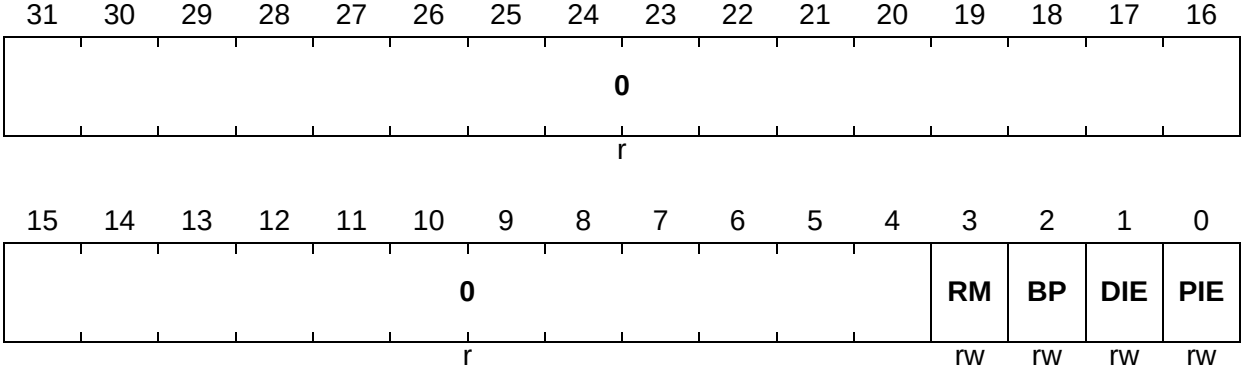
In order to force backwards compatibility with TriCore 1.3 a Compatibility register (COMPAT) is introduced. The existence of the COMPAT register is architecturally defined. However, the fields within the COMPAT register and their effect on the behaviour of the core are implementation specific. The reset value of the COMPAT register ensures that backwards compatibility with TriCore 1.3 is enabled by default.

The COMPAT register may only be written in supervisor mode and is endinit protected. The definition of the COMPAT register is as follows:

Compatibility Control Register

COMPAT
Compatibility Control Register

(F7E1 9400$_H$) **Reset Value: FFFF FFFF$_H$**



Field	Bits	Type	Description
PIE	0	rw	**Program Integrity Error Compatibility** 0_B Errors handled by CPU. 1_B Errors flagged off-core. TriCore 1.3 backwards compatibility.
DIE	1	rw	**Data Integrity Error Compatibility** 0_B Errors handled by CPU. 1_B Errors flagged off-core. TriCore 1.3 backwards compatibility.
BP	2	rw	**Branch Predictor Compatibility** 0_B Bi-model branch prediction. 1_B Static branch prediction. TriCore 1.3 backwards compatibility.

Field	Bits	Type	Description
RM	3	rw	**Rounding Mode Compatibility** 0_B PSW.RM not restored by RET. 1_B PSW.RM restored by RET. TriCore 1.3 backwards compatibility.
0	[31:4]	r	**Reserved.** Read as 0; should be written with 0.

2.6 Core Debug Registers

In the TC1797, TriCore 1.3.1 Core Debug registers are available for debug purposes. For a complete description of all registers, refer to the TriCore Architecture Manual.

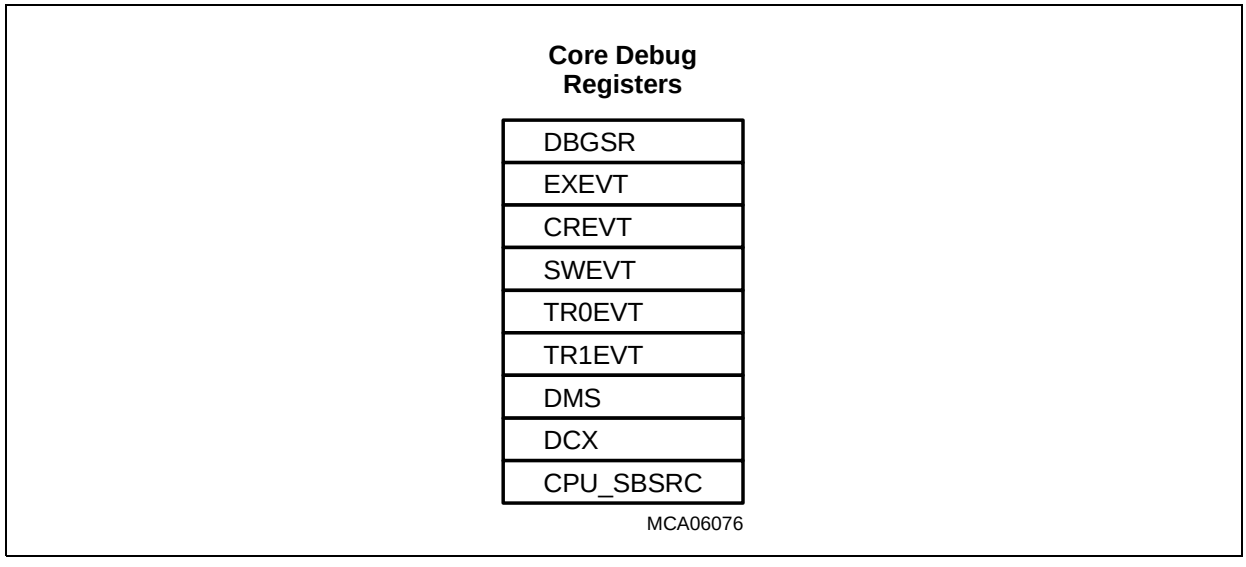


Figure 2-11 Core Debug Registers

Table 2-8 Core Debug Registers

Short Name	Description	Offset Address	Access Mode		Reset
			Read	Write	
DBGSR	Debug Status Register	$FD00_H$	U, SV, 32	SV, 32	Class 1 Reset 0000 0000$_H$
EXEVT	External Break Input Event Register	$FD08_H$	U, SV, 32	SV, 32	Class 1 Reset 0000 0000$_H$
CREVT	Core SFR Access Break Event Register	$FD0C_H$	U, SV, 32	SV, 32	Class 1 Reset 0000 0000$_H$
SWEVT	Software Break Event Register	$FD10_H$	U, SV, 32	SV, 32	Class 1 Reset 0000 0000$_H$
TR0EVT	Trigger Event 0 Register	$FD20_H$	U, SV, 32	SV, 32	Class 1 Reset 0000 0000$_H$
TR1EVT	Trigger Event 1 Register	$FD24_H$	U, SV, 32	SV, 32	Class 1 Reset 0000 0000$_H$
DMS	Debug Monitor Start Address Register	$FD40_H$	U, SV, 32	U, SV, 32, NC	Class 1 Reset 0000 0000$_H$
DCX	Debug Context Save Area Pointer	$FD44_H$	U, SV, 32	SV, 32	Class 1 Reset DE80 0000$_H$

Table 2-8 Core Debug Registers (cont'd)

Short Name	Description	Offset Address	Access Mode		Reset
			Read	Write	
DBGTCR	Debug Trap Control Register	$FD48_H$	U, SV, 32	SV, 32	Class 1 Reset $0000\ 0000_H$
CPU_SBSRC	CPU Software Breakpoint Service Request Control Register	$FFBC_H$ 1)	U, SV	SV, 32	Class 3 Reset $0000\ 0000_H$

1) Located in the CPU slave (CPS) interface register area.

2.7 Implementation Specific Reset Values

This section summarizes the implementation specific reset values of the CPU registers not defined in this chapter.

Table 2-9 Implementation Specific Reset Values

Register	Address	Reset Value
PCXI	F7E1 FE00$_H$	0000 0000$_H$
PCX	F7E1 FE00$_H$	0000 0000$_H$
CPU_ID	F7E1 FE18$_H$	000A C006$_H$
FCX	F7E1 FE38$_H$	0000 0000$_H$
LCX	F7E1 FE3C$_H$	0000 0000$_H$
COMPAT	F7E1 9400$_H$	FFFF FFFF$_H$
ISP	F7E1 FE28$_H$	0000 0100$_H$
BIV	F7E1 FE20$_H$	0000 0000$_H$
BTV	F7E1 FE24$_H$	A000 0100$_H$

2.8 CPU Instruction Timing

This section gives information on CPU instruction timing by execution unit. The Integer Pipeline and Load/Store Pipeline are always present, and the Floating Point Unit (FPU) is optional. The Load/Store unit implements the optional TLB instructions.

Definition of Terms:

- **Repeat Rate**

Assuming the same instruction is being issued sequentially, repeat is the minimum number of clock cycles between two consecutive issues. There may be additional delays described elsewhere due to internal pipeline effects when issuing a different subsequent instruction.

- **Result Latency**

The number of clock cycles from the cycle when the instruction is issued to the cycle when the result value is available to be used as an operand to a subsequent instruction or written into a GPR. Result latency is not meaningful for instructions that do not write a value into a GPR.

- **Address Latency**

The number of clocks cycles from the cycle when the instruction is issued to the cycle when the addressing mode updated value is available as an operand to a subsequent instruction or written into an Address Register.

- **Flow Latency**

The number of clock cycles from the cycle when the instruction is issued to the cycle when the next instruction (located at the target location or the next sequential instruction if the control change is conditional) is issued.

2.8.1 Integer-Pipeline Instructions

These are the Integer-Pipeline instruction timings for each instruction.

2.8.1.1 Simple Arithmetic Instruction Timings

Each instruction is single issued.

Table 2-10 Simple Arithmetic Instruction Timing

Instruction	Result Latency	Repeat Rate	Instruction	Result Latency	Repeat Rate
Integer Pipeline Arithmetic Instructions					
ABS	1	1	**MAX.H**	1	1
ABS.B	1	1	**MAX.HU**	1	1
ABS.H	1	1	**MAX.U**	1	1
ABSDIF	1	1	**MIN**	1	1
ABSDIF.B	1	1	**MIN.B**	1	1
ABSDIF.H	1	1	**MIN.BU**	1	1
ABSDIFS	1	1	**MIN.H**	1	1
ABSDIFS.H	1	1	**MIN.HU**	1	1
ABSS	1	1	**MIN.U**	1	1
ABSS.H	1	1	**RSUB**	1	1
ADD	1	1	**RSUBS**	1	1
ADD.B	1	1	**RSUBS.U**	1	1
ADD.H	1	1	**SAT.B**	1	1
ADDC	1	1	**SAT.BU**	1	1
ADDI	1	1	**SAT.H**	1	1
ADDIH	1	1	**SAT.HU**	1	1
ADDS	1	1	**SEL**	1	1
ADDS.H	1	1	**SELN**	1	1
ADDS.HU	1	1	**SUB**	1	1
ADDS.U	1	1	**SUB.B**	1	1
ADDX	1	1	**SUB.H**	1	1
CADD	1	1	**SUBC**	1	1
CADDN	1	1	**SUBS**	1	1

Table 2-10 Simple Arithmetic Instruction Timing (cont'd)

Instruction	Result Latency	Repeat Rate	Instruction	Result Latency	Repeat Rate
CSUB	1	1	**SUBS.H**	1	1
CSUBN	1	1	**SUBS.HU**	1	1
MAX	1	1	**SUBS.U**	1	1
MAX.B	1	1	**SUBX**	1	1
MAX.BU	1	1			
Compare Instructions					
EQ	1	1	**LT.B**	1	1
EQ.B	1	1	**LT.BU**	1	1
EQ.H	1	1	**LT.H**	1	1
EQ.W	1	1	**LT.HU**	1	1
EQANY.B	1	1	**LT.U**	1	1
EQANY.H	1	1	**LT.W**	1	1
GE	1	1	**LT.WU**	1	1
GE.U	1	1	**NE**	1	1
LT	1	1			
Count Instructions					
CLO	1	1	**CLS.H**	1	1
CLO.H	1	1	**CLZ**	1	1
CLS	1	1	**CLZ.H**	1	1
Extract Instructions					
DEXTR	1	1	**INS.T**	1	1
EXTR	1	1	**INSN.T**	1	1
EXTR.U	1	1	**INSERT**	1	1
IMASK	1	1			
Logical Instructions					
AND	1	1	**OR.EQ**	1	1
AND.AND.T	1	1	**OR.GE**	1	1
AND.ANDN.T	1	1	**OR.GE.U**	1	1
AND.EQ	1	1	**OR.LT**	1	1
AND.GE	1	1	**OR.LT.U**	1	1

Table 2-10 Simple Arithmetic Instruction Timing (cont'd)

Instruction	Result Latency	Repeat Rate	Instruction	Result Latency	Repeat Rate
AND.GE.U	1	1	**OR.NE**	1	1
AND.LT	1	1	**OR.NOR.T**	1	1
AND.LT.U	1	1	**OR.OR.T**	1	1
AND.NE	1	1	**OR.T**	1	1
AND.NOR.T	1	1	**ORN**	1	1
AND.OR.T	1	1	**ORN.T**	1	1
AND.T	1	1	**XNOR**	1	1
ANDN	1	1	**XNOR.T**	1	1
ANDN.T	1	1	**XOR**	1	1
NAND	1	1	**XOR.EQ**	1	1
NAND.T	1	1	**XOR.GE**	1	1
NOR	1	1	**XOR.GE.U**	1	1
NOR.T	1	1	**XOR.LT**	1	1
OR	1	1	**XOR.LT.U**	1	1
OR.AND.T	1	1	**XOR.NE**	1	1
OR.ANDN.T	1	1	**XOR.T**	1	1
Move Instructions					
CMOV	1	1	**MOV.U**	1	1
CMOVN	1	1	**MOVH**	1	1
MOV	1	1			
Shift Instructions					
SH	1	1	**SH.NE**	1	1
SH.AND.T	1	1	**SH.NOR.T**	1	1
SH.ANDN.T	1	1	**SH.OR.T**	1	1
SH.EQ	1	1	**SH.ORN.T**	1	1
SH.GE	1	1	**SH.XNOR.T**	1	1
SH.GE.U	1	1	**SH.XOR.T**	1	1
SH.H	1	1	**SHA**	1	1
SH.LT	1	1	**SHA.H**	1	1
SH.LT.U	1	1	**SHAS**	1	1

Table 2-10 Simple Arithmetic Instruction Timing (cont'd)

Instruction	Result Latency	Repeat Rate	Instruction	Result Latency	Repeat Rate
SH.NAND.T	1	1			
Coprocessor 0 Instructions					
BMERGE	1	1	**DVSTEP**	4	4
BSPLIT	1	1	**DVSTEP.U**	4	4
DVADJ	1	1	**IXMAX**	1	1
DVINIT	1	1	**IXMAX.U**	1	1
DVINIT.U	1	1	**IXMIN**	1	1
DVINIT.B	1	1	**IXMIN.U**	1	1
DVINIT.H	1	1	**PACK**	1	1
DVINIT.BU	1	1	**PARITY**	1	1
DVINIT.HU	1	1	**UNPACK**	1	1

2.8.1.2 Multiply Instruction Timings

Each instruction is single issued.

Table 2-11 Multiply Instruction Timing

Instruction	Result Latency	Repeat Rate	Instruction	Result Latency	Repeat Rate
MUL	2	1	**MUL.Q**	2	1
MUL.U	2	1	**MULM.H**	2	1
MULS	2	1	**MULR.H**	2	1
MULS.U	2	1	**MULR.Q**	2	1
MUL.H	2	1			

2.8.1.3 Multiply Accumulate (MAC) Instruction Timing

Each instruction is single issued.

Table 2-12 Multiply Accumulate Instruction Timing

Instruction	Result Latency	Repeat Rate	Instruction	Result Latency	Repeat Rate
MADD	2	1	**MSUB**	2	1
MADD.U	2	1	**MSUB.U**	2	1
MADDS	2	1	**MSUBS**	2	1
MADDS.U	2	1	**MSUBS.U**	2	1
MADD.H	2	1	**MSUB.H**	2	1
MADD.Q	2	1	**MSUB.Q**	2	1
MADDM.H	2	1	**MSUBM.H**	2	1
MADDMS.H	2	1	**MSUBMS.H**	2	1
MADDR.H	2	1	**MSUBR.H**	2	1
MADDR.Q	2	1	**MSUBR.Q**	2	1
MADDRS.H	2	1	**MSUBRS.H**	2	1
MADDRS.Q	2	1	**MSUBRS.Q**	2	1
MADDS.H	2	1	**MSUBS.H**	2	1
MADDS.Q	2	1	**MSUBS.Q**	2	1
MADDSU.H	2	1	**MSUBAD.H**	2	1
MADDSUM.H	2	1	**MSUBADM.H**	2	1
MADDSUMS.H	2	1	**MSUBADMS.H**	2	1
MADDSUR.H	2	1	**MSUBADR.H**	2	1
MADDSURS.H	2	1	**MSUBADRS.H**	2	1
MADDSUS.H	2	1	**MSUBADS.H**	2	1

For MADD.Q, MADDS.Q, MSUB.Q, MSUBS.Q Instructions:

	Result Latency	Repeat Rate
16 × 16	2	1
16 × 32	2	1
32 × 32	2	1

2.8.1.4 Control Flow Instruction Timing

Note all Integer Pipeline Control flow instructions are conditional.

- Each instruction is single issued.
- All target locations yield a full instruction in one access (i.e. not 16-bits of a 32-bit instruction).
- All code fetches take a single cycle.
- Timing is best case; no cache misses for context operations, no pending stores.

Table 2-13 Integer Pipeline Control Flow Instruction Timing

Instruction	Flow Latency	Repeat Rate	Instruction	Flow Latency	Repeat Rate
Branch Instructions					
JEQ	1/2/3	1/2/3	**JLTZ**	1/2/3	1/2/3
JGE	1/2/3	1/2/3	**JNE**	1/2/3	1/2/3
JGE.U	1/2/3	1/2/3	**JNED**	1/2/3	1/2/3
JGEZ	1/2/3	1/2/3	**JNEI**	1/2/3	1/2/3
JGTZ	1/2/3	1/2/3	**JNZ**	1/2/3	1/2/3
JLEZ	1/2/3	1/2/3	**JNZ.T**	1/2/3	1/2/3
JLT	1/2/3	1/2/3	**JZ**	1/2/3	1/2/3
JLT.U	1/2/3	1/2/3	**JZ.T**	1/2/3	1/2/3

For All Control Flow Instructions:

	Flow Latency	Repeat Rate
Correctly predicted, not taken	1	1
Correctly predicted, taken	2	2
Wrongly predicted	3	3

2.8.2 Load-Store Pipeline Instructions

This section summarizes the Load-Store Pipeline instructions.

2.8.2.1 Address Arithmetic Timing

Each instruction is single issued.

Table 2-14 Address Arithmetic Instruction Timing

Instruction	Result Latency	Repeat Rate	Instruction	Result Latency	Repeat Rate
Load Store Arithmetic Instructions					
ADD.A	1	1	**GE.A**	1	1
ADDIH.A	1	1	**LT.A**	1	1
ADDSC.A	1	1	**NE.A**	1	1
ADDSC.AT	1	1	**NEZ.A**	1	1
EQ.A	1	1	**SUB.A**	1	1
EQZ.A	1	1	**NOP**	1	1
Trap and Interrupt Instructions					
DEBUG	–	1	**TRAPSV**[1)]	–	1
DISABLE	–	1	**TRAPV**[1)]	–	1
ENABLE	–	1	**RSTV**	–	1
Move Instructions					
MFCR	1	1	**MOV.A**	1	1
MTCR	–	1	**MOV.AA**	1	1
MOVH.A	1	1	**MOV.D**	1	1
Sync Instructions					
DSYNC[2)]	–	1	**ISYNC**[3)]	–	1

1) Execution cycles when no TRAP is taken. The execution timing in the case of raising these TRAPs is the same as other TRAPs such as SYSCALL.

2) Repeat rate assumes that no shadow register writeback is pending, otherwise the repeat rate will depend upon the time for all delayed memory operation to occur.

3) Repeat rate assumes that code refetch takes a single cycle.

2.8.2.2 Control Flow Instruction Timing

This section summarizes the timing of Control Flow instructions.

Each instruction is single issued.

- All targets yield a full instruction in one access (not 16-bits of a 32-bit instruction).
- All code fetches take a single cycle. Timing is best case; no cache misses for context operations, no pending stores.
- Latency of CSA related instructions varies according to preceding instruction and status of the shadow register file.

Table 2-15 Load Store Control Flow Instruction Timing

Instruction	Flow Latency	Repeat Rate	Instruction	Flow Latency	Repeat Rate
Branch Instructions					
J	2	2	**JLI**	2	2
JA	2	2	**JEQ.A**	1/2/3	1/2/3
JI	2	2	**JNE.A**	1/2/3	1/2/3
JL	2	2	**JNZ.A**	1/2/3	1/2/3
JLA	2	2	**JZ.A**	1/2/3	1/2/3
CSA Instructions					
CALL[1)]	2-9	2-9	**SYSCALL**[1)]	2-9	2-9
CALLA[1)]	2-9	2-9	**SVLCX**[4)]	4-16	4-16
CALLI[1)]	2-9	2-9	**RSLCX**[2)]	4, 8	4, 8
RET[3)]	2-9	2-9	**RFE**[3)]	2-9	2-9
BISR[4)]	4-16	4-16	**RFM**[5)]	2-5	2-5
Loop Instructions					
LOOP[6)]	2/1/3	2/1/3	**LOOPU**[6)]	2/1/3	2/1/3

1) The range is 2-5 for LDRAM and 3-9 for Cached External Memory. The average latency is ~2.7 cycles for LDRAM and 5 cycles for Cached External Memory.

2) The range is 4 for LDRAM and 8 for Cached External Memory.

3) The range is 2-5 for LDRAM and 2-9 for Cached External Memory.

4) The range is 4-9 for LDRAM and 7-16 for Cached External Memory.

5) Not strictly a CSA operation, but retrieves from memory a subset of context information and changes control flow in a similar manner. The range is 2-3 for LDRAM and 4-5 for Cached External Memory.

6) First time encountered executed in LS pipeline: Flow latency = 2, Repeat rate = 2.
Successive time executed in Loop pipeline: Flow latency = 1: Repeat rate = 1 (nested up to 2 deep).
Last time encountered: Flow latency = 3: Repeat rate = 3.

For JLI, JEQ.A, JNE.A JNZ.A, JZ.A Instructions:

	Flow Latency	Repeat Rate
Correctly predicted, not taken	1	1
Correctly predicted, taken	2	2
Wrongly predicted	3	2

2.8.2.3 Load Instruction Timing

Load instructions can produce two results if they use the pre-increment, post-increment, circular or bit-reverse addressing modes. Hence, in those cases there are two latencies that must be specified, the result latency for the value loaded from memory and the address latency for using the updated address register result.

- Each instruction is single issued.
- The memory references is naturally aligned.
- The memory accessed takes a single cycle to return a data item.
- Timing is best case; no cache misses, no pending stores.

Table 2-16 Load Instruction Timing

Instruction	Address Latency	Result Latency	Repeat Rate	Instruction	Address Latency	Result Latency	Repeat Rate
Load Instructions							
LD.A	1	2	1	**LD.Q**	1	1	1
LD.B	1	1	1	**LD.W**	1	1	1
LD.BU	1	1	1	**LDLCX**	4	4	4
LD.D	1	1	1	**LDUCX**	4	4	4
LD.DA	1	2	1	**SWAP.W**	2	2	2
LD.H	1	1	1	**LEA**[1)]	–	1	1
LD.HU	1	1	1				

1) The addressing mode returning an updated address is not relevant for this instruction.

2.8.2.4 Store Instruction Timing

Cache and Store instructions similar to Load instructions will have a result for the pre-increment, post-increment, circular or bit-reverse addressing modes, but do not produce a ‘memory’ result.

- Each instruction is single issued.
- The memory references is naturally aligned.
- The memory accessed takes a single cycle to accept a data item.
- Timing is best case; no cache misses, no pending stores.

Table 2-17 Cache and Store Instruction Timing

Instruction	Address Latency	Repeat Rate	Instruction	Address Latency	Repeat Rate
Cache Instructions					
CACHEA.I	1	1	**CACHEA.WI**[1)]	1	1
CACHEA.W[1)]	1	1	**CACHEI.W**	1	1
CACHEI.WI	1	1			
Store Instructions					
ST.A	1	1	**ST.T**	2	2
ST.B	1	1	**ST.W**	1	1
ST.D	1	1	**STLCX**	4	4
ST.DA	1	1	**STUCX**	4	4
ST.H	1	1	**LDMST**	2	2
ST.Q	1	1			

1) Repeat rate assumes that no memory writeback operation occurs. Otherwise the repeat rate will depend upon the time for the castout buffers to clear.

2.8.3 Floating Point Pipeline Timing

These instructions are only valid if the optional Floating Point Unit is implemented.

Each instruction is single issued.

Table 2-18 Floating Point Instruction Timing

Instruction	Result Latency	Repeat Rate	Instruction	Result Latency	Repeat Rate
Floating Point Instructions					
ADDF	2	2	**ITOF**	2	2
CMP.F	1	1	**MADD.F**	3	3
DIV.F	15	15	**MSUB.F**	3	3
FTOI	2	2	**MUL.F**	2	2
FTOIZ	2	2	**Q31TOF**	2	2
FTOQ31	2	2	**QSEED.F**	1	1
FTOQ31Z	2	2	**SUB.F**	2	2
FTOU	2	2	**UPDFL**	–	1
FTOUZ	2	2	**UTOF**	2	2

2.9 Program Memory Interface (PMI)

Figure 2-12 shows the block diagram of the Program Memory Interface (PMI) of the TC1797.

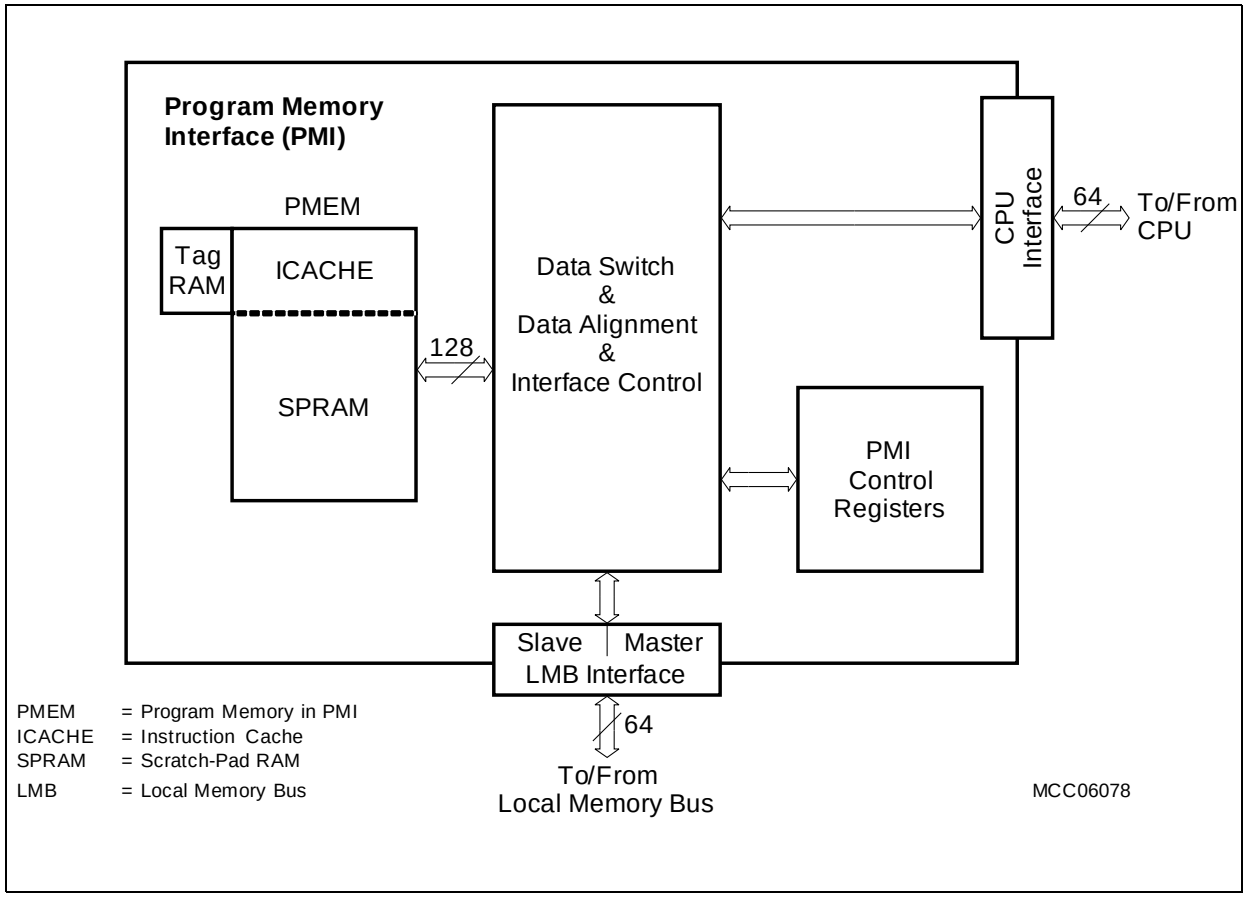


Figure 2-12 PMI Block Diagram

2.9.1 PMI Features

The Program Memory Interface (PMI) has the following features:

- 40 Kbyte total Program Memory (PMEM), with software configurable split between SPRAM and ICACHE. The following configurations are supported:

SPRAM	ICACHE
24	16
32	8
36	4
38	2
40	0

- ICACHE operation features:
  - Two-way set associative cache
  - LRU (Least-Recently Used) replacement algorithm
  - Cache line size: 256 bits (4 double-words)
  - Validity granularity: One valid bit per cache line
  - ICACHE can be globally invalidated to provide support for software cache coherency (to be handled by the programmer)
  - ICACHE can be bypassed to provide a direct fetch from the CPU to on-chip and off-chip resources
  - ICACHE refill mechanism:
    critical double-word first, line wrap around, streaming to CPU
- CPU interface
  - Supporting unaligned accesses (16-bit aligned)
- Local Memory Bus (LMB) Master Interface to PMI, DMI and LFI-Bridge
- Local Memory Bus (LMB) Slave Interface to scratchpad RAM
- PMI SRAMs (SPRAM, ICACHE, and cache tag SRAM) are parity protected

2.9.2 LMB Access Priorities

The TC1797 contains a common local memory bus, shared between the Program Memory Interface (PMI), Data Memory Interface (DMI), DMA controller and LMB-FPI Interface (FPI) bus masters. In such systems, the DMI is the default bus master whilst LMB arbitration priorities are as follows:

1. DMA High
2. LFI
3. DMA Medium
4. DMI
5. PMI
6. DMA Low

2.9.3 Scratchpad RAM

The TC1797 contains up to 40 Kbyte of scratchpad RAM. Scratchpad RAM provides a fast, deterministic program fetch access from the CPU for use by performance critical code sequences.

- CPU program fetch accesses to scratchpad RAM are never cached in the instruction cache and are always directly targeted to the scratchpad RAM.
- The scratchpad RAM has the concept of a scratchpad RAM “line” (similar to the instruction cache). Scratchpad RAM lines are 256-bits long (4 double-words).

The CPU fetch interface will generate unaligned accesses (16-bit aligned), which will normally result in 64-bits of instruction being returned to the CPU.

- If the fetch request is made within a scratchpad RAM line, the 64-bit instruction packet is returned to the CPU in a single cycle.
- If the fetch request is made to the end of a scratchpad RAM line, such that 64-bits would span two scratchpad RAM lines, then only the instruction half-words up to the end of the scratchpad RAM line are returned to the CPU.

Note that the CPU Fetch Unit can only read from the scratchpad RAM and can never write to it.

The scratchpad RAM may also be accessed from the LMB Slave interface by another bus master, such as the Data Memory Interface (DMI). The scratchpad RAM may be both read and written from the LMB. In the TC1797, the PMI LMB Slave interface supports all LMB transaction types.

2.9.4 Instruction Cache

The TC1797 contains up to 16 Kbyte of Instruction Cache (ICACHE). The ICACHE is a two-way set-associative cache with a Least-Recently-Used (LRU) replacement algorithm, and is organized as 512 cache lines, with 256-bits per line. Each ICACHE line has a single associated valid bit.

CPU program fetch accesses which target a cacheable memory segment (and where the ICACHE is not bypassed) target the ICACHE. If the requested address and its associated instruction are found in the cache (Cache Hit), the instruction is passed to the CPU Fetch Unit without incurring any wait states. If the address is not found in the cache (Cache Miss), the PMI cache controller issues a cache refill sequence and wait states are incurred whilst the cache line is refilled. The CPU fetch interface will generate unaligned accesses (16-bit aligned), which will normally result in 64-bits of instruction being returned to the CPU. If the fetch request is made within a ICACHE line, no matter the alignment, and a cache hit occurs, then the 64-bit instruction packet is returned to the CPU. If the fetch request is made to the end of a ICACHE line, such that 64-bits would span two ICACHE lines, then only the instruction half-words up to the end of the ICACHE line are returned to the CPU.

Instruction Cache Refill Sequence

Instruction Cache refills are performed using a critical double-word first strategy with cache line wrapping such that the refill size is always 4 double-words. ICACHE refills are always performed in 64-bit quantities. A refill sequence will always affect only one cache line. There is no prefetching of the next cache line.

ICACHE refills are therefore implemented using an LMB Burst Transfer 4 (BTR4) transfers. The Instruction Cache supports instruction streaming, meaning that it can deliver available instruction half-words to the CPU Fetch Unit whilst the refill operation is ongoing.

Instruction Cache Bypass

The Instruction Cache may be bypassed, under control of PMI_CON0.PCBYP, to provide a direct instruction fetch path for the CPU Fetch Unit. The default value of PMI_CON0.PCBYP is such that the ICACHE is bypassed after reset. ICACHE bypass should be disabled during initialization to enable the ICACHE.

Whilst ICACHE bypass is enabled, a fetch request by the CPU to a cacheable address will result in a forced cache miss, such that the cache controller issues a standard refill sequence and supplies instruction half-words to the CPU using instruction streaming, without updating the cache contents. Any valid cache lines within the ICACHE will remain valid and unchanged whilst the ICACHE is bypassed. As such, instruction fetch requests to cacheable addresses with ICACHE bypass enabled behave identically to instruction fetch requests to non-cacheable addresses.

Instruction Cache Invalidation

The PMI does not have automatic cache coherency support. Changes to the contents of memory areas external to the PMI that may have already been cached in the ICACHE are not detected. Software must provide the cache coherency in such a case. The PMI supports this via the cache invalidation function. The ICACHE contents may be globally invalidated by writing a ‘1’ to PMI_CON1.PCINV. The ICACHE invalidation is performed over multiple cycles by a hardware state machine which cycles through the ICACHE entries marking each as invalid. The status of the ICACHE invalidation sequence may be determined by reading the PMI_CON1.PCINV bit.

2.9.5 Program Line Buffer

The PMI module contains a 256-bit Program Line Buffer (PLB). For accesses to cacheable addresses the PLB acts as a streaming buffer for the main instruction cache. Instruction cache refill sequences transfer data to the PLB and the immediately to the TriCore CPU without updating the main instruction cache. The PLB contents are then transferred to the main instruction cache on the next instruction cache miss.

Program fetch requests to non-cacheable addresses utilize the PLB as a single line cache. A single valid bit is associated with the PLB, denoting that the PLB contents are valid. As such all fetch requests resulting in an update of the PLB, whether to a cacheable address or not, are implemented as LMB Burst Transfer 4 (BTR4) transactions, with the critical double-word of the PLB line being fetched first size.

2.9.6 PMI Registers

Three control registers are implemented in the Program Memory Interface. These registers and their bits are described in this section.

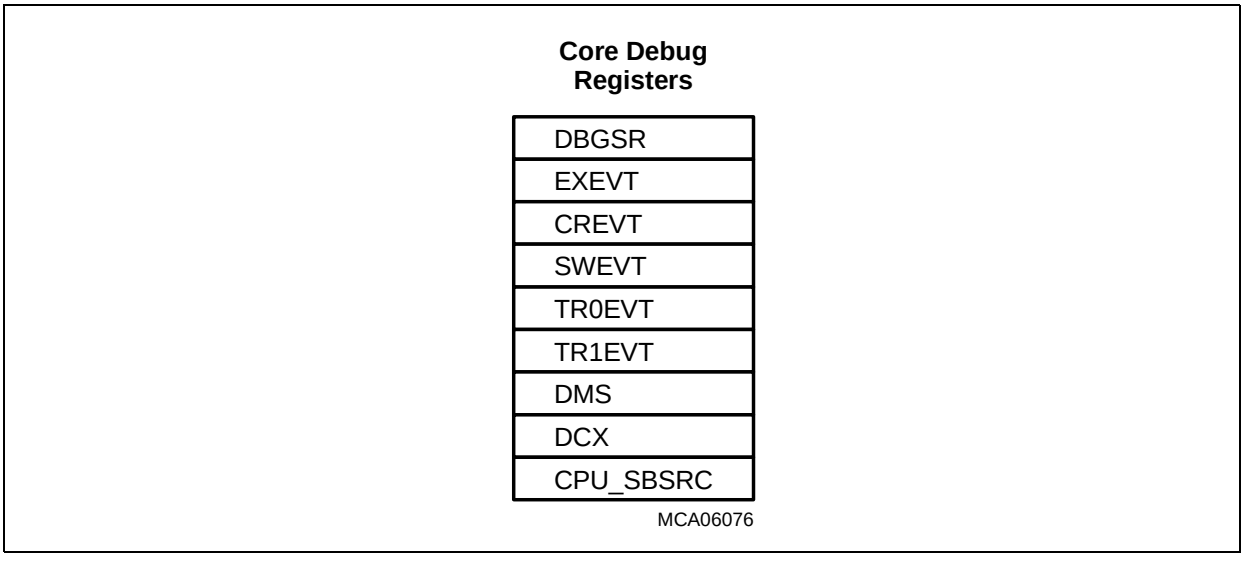


Figure 2-13 PMI Registers

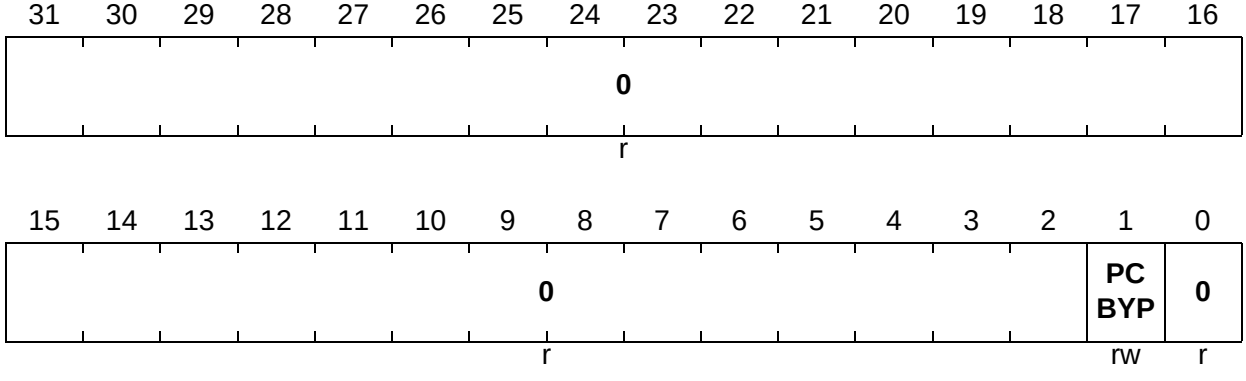
Table 2-19 PMI Registers

Short Name	Description	Offset Address	Access Mode		Reset
			Read	Write	
PMI_ID	PMI Identification Register	$FD08_H$	U, SV, 32	SV, 32	Class 3 Reset $000B\ C005_H$
PMI_CON0	PMI Control Register 0	$FD10_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0002_H$
PMI_CON1	PMI Control Register 1	$FD14_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$
PMI_CON2	PMI Control Register 2	$FD18_H$	U, SV, 32	SV, E, 32	Class 3 Reset $0280\ 0284_H$
PMI_STR	PMI Synchronous Trap Register	$FD20_H$	U, SV, 32	SV, 32	Class 3 Reset $0000\ 0000_H$

2.9.6.1 PMI Register Descriptions

PMI Control Register 0

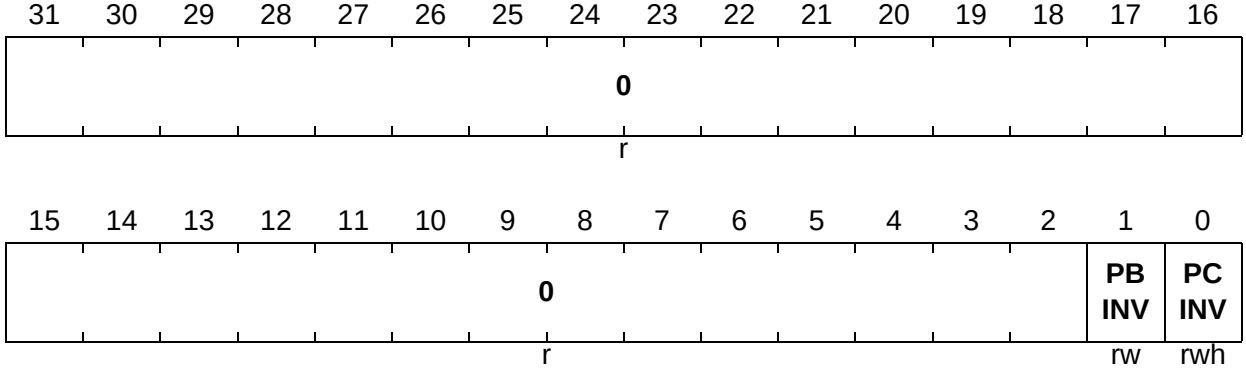
PMI_CON0
PMI Control Register 0 (F87F FD10$_H$) Reset Value: 0000 0002$_H$



Field	Bits	Type	Description
PCBYP	1	rw	**Instruction Cache Bypass** 0_B Cache enabled 1_B Cache bypassed (disabled)
0	[31:2], 0	r	**Reserved** Read as 0; should be written with 0.

PMI Control Register 1

PMI_CON1
PMI Control Register 1 **(F87F FD14$_H$)** **Reset Value: 0000 0000$_H$**

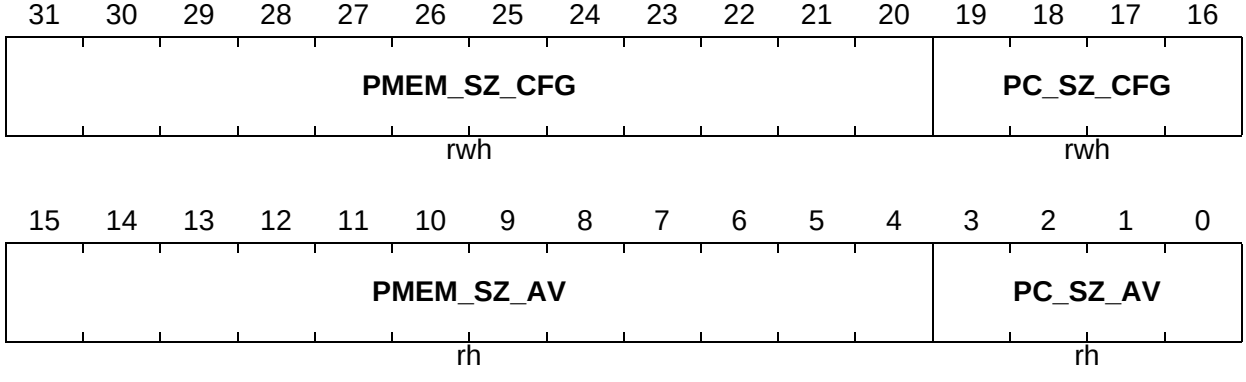


Field	Bits	Type	Description
PCINV	0	rwh	**Instruction Cache Invalidate** Write Operation: 0_B No effect. Normal instruction cache operation. 1_B Initiate invalidation of entire instruction cache. Read Operation: 0_B Normal operation. Instruction cache available. 1_B Instruction cache invalidation in progress. Instruction cache unavailable.
PBINV	1	rw	**Program Buffer Invalidate** Write Operation: 0_B No effect. Normal program line buffer operation. 1_B Invalidate the program line buffer. This field returns 0 when read.
0	[31:2]	r	**Reserved** Read as 0; should be written with 0.

PMI Control Register 2

The PMI_CON2 register may only be written in supervisor mode and is endinit protected. In addition write accesses to PMI_CON2 are also dependent on the status of Flash read protection. Whenever Flash read protection is inactive PMI_CON2 may be written as often as required (bearing in mind operational constraints for changing SRAM and cache sizes). When Flash read protection becomes active after reset, PMI_CON2 may be written only the one time. Any further writes to PMI_CON2 with Flash read protection active are ignored. Setting Flash read protection inactive then active again does not allow further Flash read protection enabled PMI_CON2 writes; only a reset will re-enable the single Flash read protection enabled write.

PMI_CON2
PMI Control Register 2 (F87F FD18$_H$) Reset Value: 0280 0284$_H$



Field	Bits	Type	Description
PC_SZ_AV	[3:0]	rh	**Maximum Instruction Cache Size Available** Size of the maximum available Instruction Cache. Encoding is as per PC_SZ_CFG.
PMEM_SZ_AV	[15:4]	rh	**Maximum Program Memory Size Available** Size of the maximum available Program Memory[1)], where size of PMEM = size of SPRAM + size of ICACHE. Encoding is as per PMEM_SZ_CFG.

Field	Bits	Type	Description
PC_SZ_CFG	[19:16]	rwh	**Instruction Cache Size Configuration** Configuration of the Instruction Cache Size. Any program memory not utilised as instruction cache is configured as SPRAM. After reset this field is set to zero. This field may subsequently be written to select an alternative split of PMEM between ICACHE and SPRAM[2]. The encoding of PC_SZ_CFG is as follows: 0000_B No Instruction cache 0001_B 2KByte Instruction cache 0010_B 4Kbyte Instruction cache 0011_B 8Kbyte Instruction cache 0100_B 16Kbyte instruction cache Others : Reserved
PMEM_SZ_CFG	[31:20]	rwh	**Program Memory Size Configuration** Configuration of the Program Memory (PMEM) size. After reset this field is set to equal the maximum PMEM size available, PMEM_SZ_AV. This field may subsequently be written to force a smaller PMEM size to be visible to software[3]. PMEM_SZ_CFG specifies the configured PMEM size in Kbytes: 000_H Reserved. 004_H 4Kbyte Program Memory. 008_H 8Kbyte Program Memory. $00C_H$ 12Kbyte Program Memory. ... 100_H 256Kbyte Program Memory.

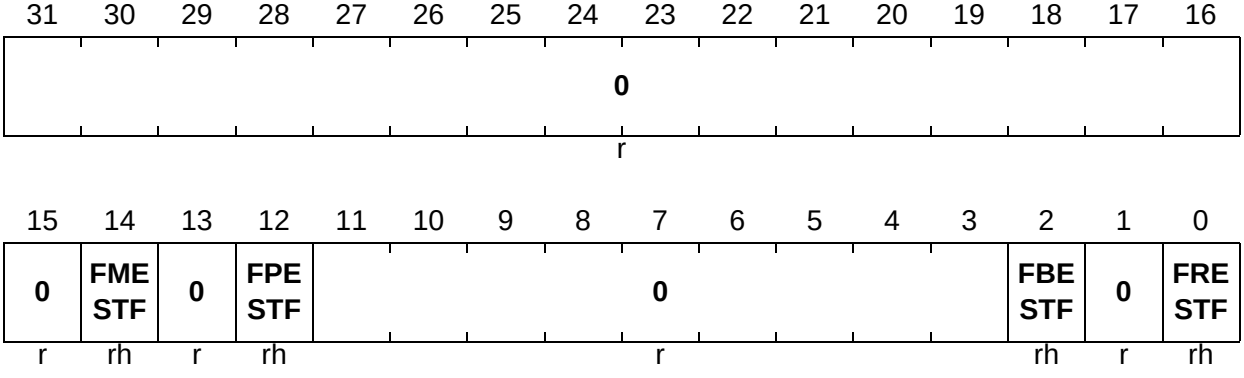
1) Configuration of program memory and instruction cache size is not intended to be performed on-the-fly by application code. The configuration should be selected once during the initialisation sequence, before PMI_CON0.PCBYP is cleared. Instruction fetches from SPRAM or cacheable program regions during switching of the PC_SZ_CFG field will give undefined behaviour.

2) Writing this field with a value larger than the maximum available ICACHE size (PC_SZ_AV) resets this field to PC_SZ_AV.

3) Writing this field with a value larger than the maximum available PMEM size (PMEM_SZ_AV) resets this field to PMEM_SZ_AV.

Program Memory Interface Synchronous Trap Register (PMI_STR)

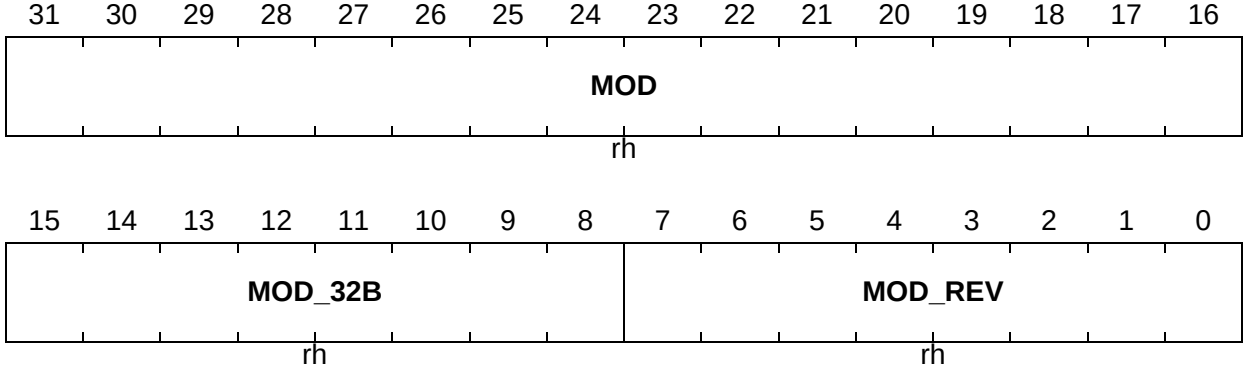
PMI_STR
PMI Synchronous Trap Register (F87F FD20$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
FRESTF	0	rh	**Fetch Range Error Synchronous Trap Flag**
FBESTF	2	rh	**Fetch Bus Error Synchronous Trap Flag**
FPESTF	12	rh	**Fetch Peripheral Error Synchronous Trap Flag**
FMESTF	14	rh	**Fetch MSIST Error Synchronous Trap Flag**
0	1, [11:3], 15, [31:16]	r	**Reserved** Read as 0; should be written with 0.

PMI Identification Register

PMI_ID
PMI Identification Register (F87F FD08$_H$) Reset Value: 000B C005$_H$



Field	Bits	Type	Description
MOD_REV	[7:0]	rh	**Revision Number**
MOD_32B	[15:8]	rh	**32-Bit Module Enable** C0$_H$ A value of C0$_H$ in this field indicates a 32-bit module with a 32-bit module ID register.
MOD	[31:16]	rh	**Module Identification Number** 0B$_H$ For module identification.

2.10 Data Memory Interface (DMI)

This figure shows the block diagram of the Data Memory Interface (DMI) of the TC1797.

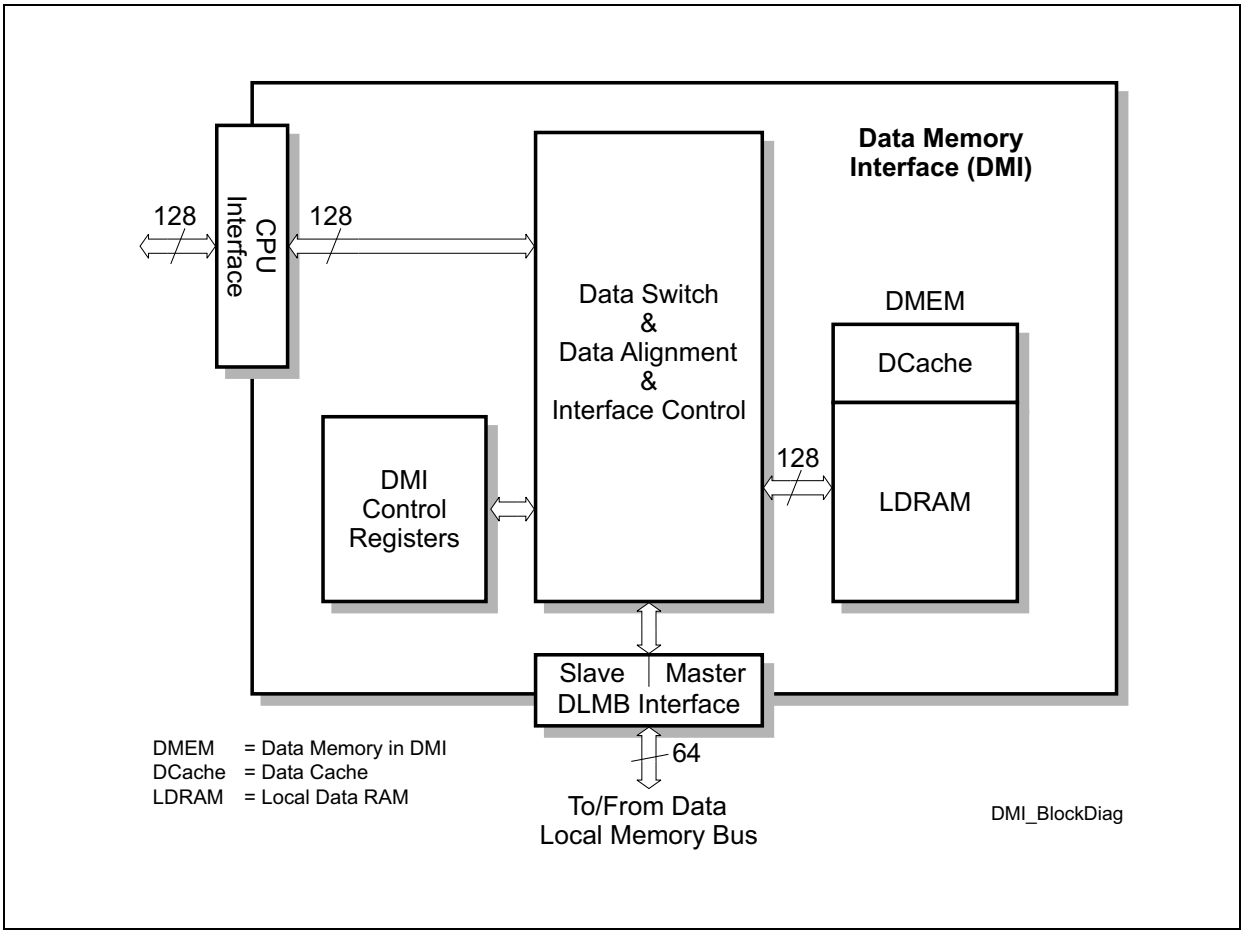


Figure 2-14 DMI Block Diagram

2.10.1 DMI Features

The Data Memory Interface (DMI) has the following features:

- 128 Kbyte total Data Memory (DMEM), with software configurable split between LDRAM and DCache. The following configurations are supported:

LDRAM	DCache
124	4
126	2
128	0

- Data Cache features:

  - Two-way associative cache, LRU (least recently used) replacement algorithm
  - Cache line size: 128 bits
  - Validity granularity: One valid bit per cache line
  - Write-back Cache: Writeback granularity: 64 bits
  - Refill mechanism: full cache line refill
- CPU interface
  - supporting unaligned accesses (16-bit aligned) with a minimum penalty of one cycle for unaligned accesses crossing 2 lines LDRAM or DCache)
- Local Memory Bus (LMB) Master interface
- Local Memory Bus (LMB) Slave interface to LDRAM

2.10.2 LMB Access Priorities

See **“LMB Access Priorities” on Page 2-69**.

2.10.3 Local Data RAM (LDRAM)

The TC1797 contains up to 128 Kbyte of LDRAM. LDRAM provides fast, deterministic data access to the CPU for use by performance critical code sequences.

The LDRAM has the concept of an LDRAM “line”. LDRAM lines are 128-bits long (2 double-words). The CPU load-store interface will generate unaligned accesses (16-bit aligned), which will result in up to 64-bits of data being transferred to or from the CPU (for non-context operations). If the data access is made within an LDRAM line, no matter the alignment, then the requested data is returned to the CPU in a single cycle. If the data access is made to the end of an LDRAM line, such that the requested data would span two LDRAM lines, a single wait cycle is incurred.

The LDRAM may also be accessed from the LMB Slave interface by another bus master, with both read and write transactions supported. The LDRAM may be accessed by the LMB Slave interface using any LMB transaction type, including burst transfers. In accordance with the LMB protocol, accesses to the LMB Slave interface must be naturally aligned.

2.10.4 Data Cache

The TC1797 contains up to 4 KByte of Data Cache (DCache). The DCache is a two-way set-associative cache with a Least-Recently-Used (LRU) replacement algorithm, and is organised as 256 cache lines, with 128-bits per line. Associated with each DCache line is a single valid bit which pertains to the entire line.

CPU data accesses to a cacheable memory segment target the DCache. If the requested address and its associated data are found in the cache (Cache Hit), the data is passed to/from the CPU Load-Store Unit without incurring any wait states. If the address is not found in the cache (Cache Miss), the DMI cache controller issues a cache refill sequence and wait states are incurred whilst the cache line is refilled. The CPU

load-store interface will generate unaligned accesses (16-bit aligned), which will result in up to 64-bits of data being transferred to or from the CPU (for non-context operations). If the data access is made within a DCache line, no matter the alignment, and a cache hit is detected then the requested data is returned to the CPU in a single cycle. If the data access is made to the end of a DCache line, such that the requested data would span two DCache lines, a single wait cycle is incurred (if both cache lines are present in the cache, otherwise a refill sequence is required for the missing cache line(s)).

The TC1797 data cache is of the writeback type. When the CPU writes to a cacheable location the data is merged with the corresponding cache line and not written to main memory immediately. Associated with each cache line is a single ‘dirty’ bit, to denote that the data in the cache line has been modified. Whenever a CPU load-store access results in a cache miss, and each of the potential cache ways that could hold the requested cache line are valid, one of the cache lines is chosen for eviction based upon the LRU replacement algorithm. The line selected for eviction is then checked to determine if it has been modified using its dirty bit. If the line has not been modified the line is discarded and the refill sequence started immediately. If the line has been modified then the dirty data is first written back to main memory before the refill is initiated. Due to the single dirty bit per cache line, 128 bits of data will always be written back, resulting in LMB Burst Transfer 2 (BTR2) transactions.

Data Cache refills always result in the full cache line being refilled, with the critical double-word of the DCache line being fetched first. A refill sequence will always affect only one cache line. There is no prefetching of the next cache line. Due to the uniform size of DCache refill sequences, such refills are always implemented using LMB Burst Transfer 2 (BTR2) transactions.

2.10.5 Data Line Buffer

The DMI module contains a 128-bit Data Line Buffer (DLB).

When the TC1797 device is configured to contain a data cache, the DLB acts as a streaming buffer for the main data cache. Data cache refill sequences transfer data to the DLB and the immediately to the TriCore CPU without updating the main data cache. The DLB contents are then transferred to the main data cache on the next data cache miss. The presence of the DLB is transparent to software but ensures that accesses to cacheable addresses typically[1)] incur no more wait cycles than accesses to non-cacheable addresses.

When the TC1797 device is configured without a data cache, the DLB acts as a single line data cache for accesses to cacheable addresses.

1) No additional wait states are incurred as long as the data cache line to be replaced does not contain dirty information which requires a cache line writeback.

A single valid bit is associated with the DLB, denoting that the DLB contents are valid. As such all accesses updating the DLB, whether data cache is configured or not, are implemented as LMB Burst Transfer 2 (BTR2) transactions, with the critical double-word of the DLB line being fetched first size.

Unlike the PMI modules PLB, data accesses to non-cacheable addresses always bypass the DLB.

2.10.6 DMI Trap Generation

CPU data accesses to the DMI may encounter one of a number of potential error conditions, which result in one of two trap conditions being reported by the DMI back to the CPU. Data Access Synchronous Bus Error (DSE) traps are generated as a result of load accesses, whilst Data Access Asynchronous Error (DAE) traps are generated as a result of store accesses. Since a number of potential error conditions exist, the DMI contains two read-only status registers that hold information about the type of the error. The DMI Synchronous Trap Flag Register (DMI_STR) contains the flags indicating the cause of a DSE trap, whilst the DMI Asynchronous Trap Flag Register (DMI_ATR) holds the flags indicating the cause of a DAE trap.

The possible error conditions and their corresponding trap flag register bits are as follows:

Range Error

Range errors are caused by accesses to LDRAM space (D000 0000_H - D3FF $FFFF_H$) outside the range of the LDRAM. Load accesses which generate a range error will result in the DMI_STR.LRESTF flag being set, store accesses will result in DMI_ATR.SREATF flag being set.

LMB Bus Error

LMB Bus errors are detected when CPU load-store accesses directly target the LMB and where an error condition is encountered on the LMB. Load accesses which generate an LMB Bus error will result in the DMI_STR.LBESTF flag being set, store accesses will result in the DMI_ATR.SBEATF flag being set. Note that accesses to DMI special function registers will also result in this error type, since such accesses are always directed to the LMB (even if performed by the CPU) and handled by the DMI LMB Slave interface.

Cache Refill Error

Cache refill errors are detected when a data cache or DLB refill sequence encounters a bus error on the LMB. Load accesses which generate a cache refill error will result in the DMI_STR.CRLESTF flag being set, whilst store accesses will result in the DMI_ATR.CRSEATF flag being set.

Cache Writeback Error

Cache writeback errors are detected when a data cache or DLB writeback sequence, initiated by a CPU load-store access generating a cache miss, encounters a bus error on the LMB. Note that unlike other error types, the address causing a cache writeback error is not related to the address of the CPU load-store access which caused the writeback.

Load accesses which encounter a cache writeback error will result in the DMI_STR.CWLESTF flag being set, whilst store accesses will result in the DMI_ATR.CWSEATF flag being set.

Cache Flush Error

Cache flush errors are detected when a data cache or DLB writeback sequence, initiated by a cache management instruction (cachea.w, cachea.wi), encounters a bus error on the LMB. Cache management instructions which encounter such errors will result in the DMI_ATR.CFEATF flag being set.

Cache management Error

Cache management errors are detected when a cache management instruction (cachea.w, cachea.i, cachea.wi) targets a non-cacheable address, either a noncacheable physical memory address or where PTE-based translation (via the MMU) results in an access being flagged as non-cacheable. Cache management errors will result in the DMI_ATR.CMEATF flag being set.

2.10.7 DMI Registers

Two control registers and two trap flag registers are implemented in the DMI. These registers and their bits are described in this section.

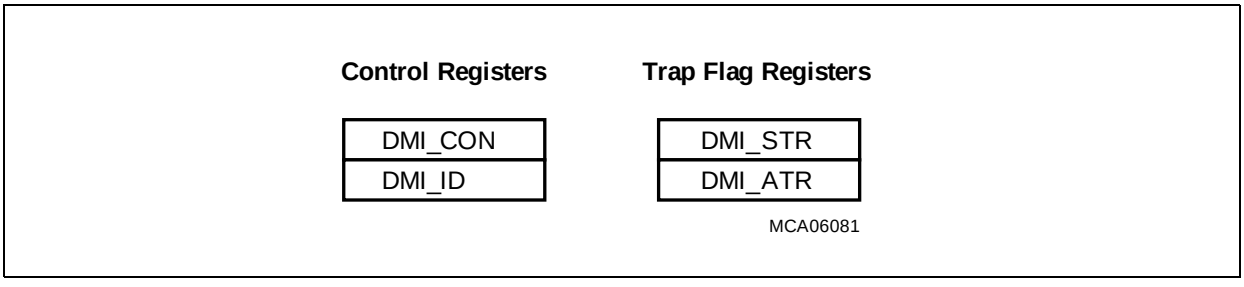


Figure 2-15 DMI Registers

Table 2-20 DMI Registers

Short Name	Description	Offset Address	Access Mode		Reset
			Read	Write	
DMI_ID	DMI Identification Register	$FC08_H$	U, SV, 32	SV, 32	Class 3 Reset 0008 $C005_H$
DMI_CON	DMI Control Register	$FC10_H$	U, SV, 32	SV, E, 32	Class 3 Reset 0800 0802_H
DMI_STR	DMI Synchronous Trap Flag Register	$FC18_H$	U, SV, 32[1)]	SV, 32	Class 3 Reset 0000 0000_H
DMI_ATR	DMI Asynchronous Trap Flag Register	$FC20_H$	U, SV, 32[1)]	SV, 32	Class 3 Reset 0000 0000_H

1) Reading these registers in supervisor mode returns the contents and then clears the register. Reading it in user mode only returns the contents of the register; it is not cleared. No error will be reported in this case.

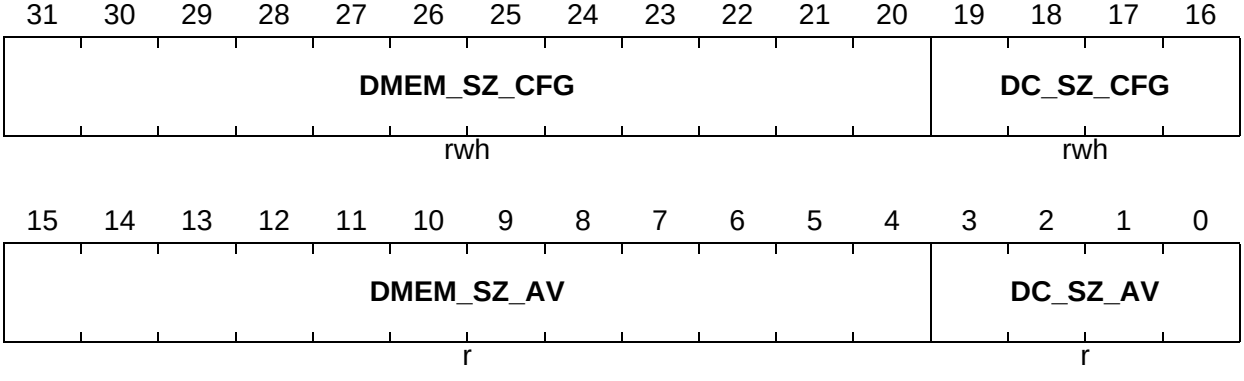
Access to DMI control registers must only be made with double-word aligned word accesses. An access not conforming to this rule, or an access that does not follow the specified privilege mode (Supervisor mode, Endinit-protection), or a write access to a read-only register, will lead to a bus error if the access was from the LMB Bus, or to a trap, flagged in DMI_STR/DMI_ATR register in case of a CPU load/store access.

2.10.7.1 DMI Register Descriptions

DMI Control Register

The DMI control register indicates the DMI data memory size and data cache availability.

DMI_CON
DMI Control Register **(F87F FC10$_H$)** **Reset Value: 0800 0802$_H$**



Field	Bits	Type	Description
DC_SZ_AV	[3:0]	r	**Maximum Data Cache Size Available** Size of the maximum available Data Cache. Encoding is as per DC_SZ_CFG.
DMEM_SZ_AV	[15:4]	r	**Maximum Data Memory Size Available** Size of the maximum available Data Memory, where size of DMEM = size of LDRAM + size of DCACHE. Encoding is as per DMEM_SZ_CFG.
DC_SZ_CFG	[19:16]	rwh	**Data Cache Size Configuration** Configuration of the Data Cache Size. Any data memory not utilised as data cache is configured as a LDRAM. After reset this field is set to zero. This field may subsequently be written to select an alternative split of DMEM between DCache and LDRAM[1)]. The encoding of DC_SZ_CFG is as follows: 0000_B No Data cache. 0001_B 2Kbyte Data cache. 0010_B 4Kbyte Data cache Others : Reserved

Field	Bits	Type	Description
DMEM_SZ_CFG	[31:20]	rwh	**Data Memory Size Configuration** Configuration of the Data Memory (DMEM) size. After reset this field is set to equal the maximum DMEM size available, DMEM_SZ_AV. This field may subsequently be written to force a smaller DMEM size to be visible to software[2]. DMEM_SZ_CFG specifies the configured DMEM size in Kbytes: 000_H Reserved. 004_H 4Kbyte Data Memory. 008_H 8Kbyte Data Memory. $00C_H$ 12Kbyte Data Memory. ... 100_H 256Kbyte Data Memory.

1) Writing this field with a value larger than the maximum available DCache size (DC_SZ_AV) resets this field to DC_SZ_AV.

2) Writing this field with a value larger than the maximum available DMEM size (DMEM_SZ_AV) resets this field to DMEM_SZ_AV.

DMI Synchronous Trap Flag Register

The DMI Synchronous Trap Flag Register, DMI_STR, holds the flags that identify the root cause of a Data-access Synchronous Bus Error (DSE). Reading DMI_STR in supervisor mode returns the register contents and then clears its contents. Reading DMI_STR in user mode returns the contents of the register but does not clear its contents.

DMI_STR
DMI Synchronous Trap Flag Register

(F87F FC18$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	LMESTF	0					CWLESTF	0	CRLESTF	0			LBESTF	0	LRESTF
r	rh	r					rh	r	rh	r			rh	r	rh

Field	Bits	Type	Description
LRESTF	0	rh	**Load Range Synchronous Error. Trap Flag**
LBESTF	2	rh	**Bus Load Synchronous Error. Trap Flag**
CRLESTF	6	rh	**Cache Refill Synchronous Error. Trap Flag**
CWLESTF	8	rh	**Cache Writeback Synchronous Error. Trap Flag**
LMESTF	14	rh	**Load MSIST Synchronous Error. Trap Flag**
0	1, [5:3], 7, [13:9], 15, [31:16]	r	**Reserved** Read as 0; should be written with 0.

DMI Asynchronous Trap Flag Register

The DMI Asynchronous Trap Flag Register, DMI_ATR, holds the flags that inform about the root cause of a Data Access Asynchronous Bus Error (ASE). Reading DMI_ATR in supervisor mode returns the register contents and then clears its contents. Reading DMI_ATR in user mode returns the contents of the register but does not clear its contents.

DMI_ATR
DMI Asynchronous Trap Flag Register

(F87F FC20$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

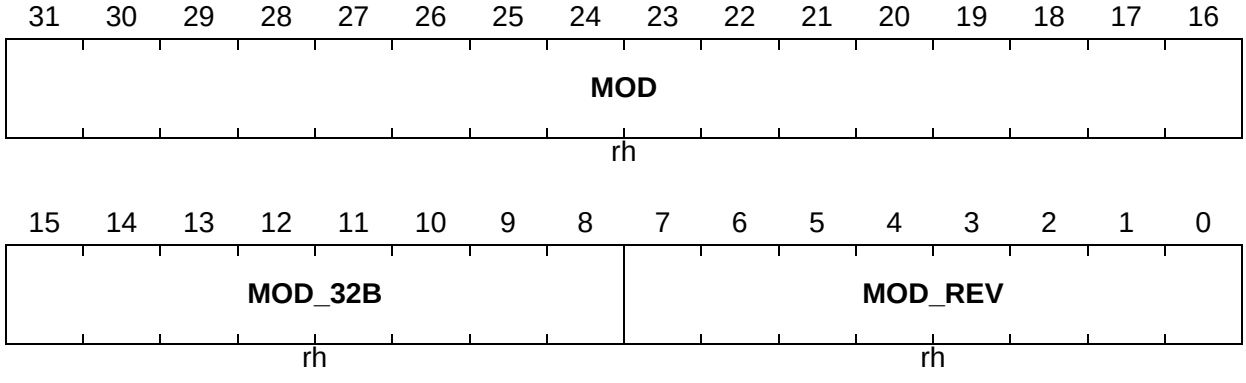
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMEATF	0			CMEATF	CFEATF	CWSEATF	0	CRSEATF	0			SBEATF	0	SREATF	0
rh	r			rh	rh	rh	r	rh	r			rh	r	rh	r

Field	Bits	Type	Description
SREATF	1	rh	**Store Range Asynchronous Error. Trap Flag.**
SBEATF	3	rh	**LMB Bus Store Asynchronous Error. Trap Flag.**
CRSEATF	7	rh	**Cache Refill Store Asynchronous Error. Trap Flag.**
CWSEATF	9	rh	**Cache Writeback Store Asynchronous Error. Trap Flag.**
CFEATF	10	rh	**Cache Flush Store Asynchronous Error. Trap Flag.**
CMEATF	11	rh	**Cache Management Store Asynchronous Error. Trap Flag.**
SMEATF	15	rh	**Store MSIST Store Asynchronous Error. Trap Flag.**
0	0, 2, [6:4], 8, [14:12] [31:16]	r	**Reserved** Read as 0; should be written with 0.

DMI Identification Register

DMI_ID
DMI Identification Register

(F87F FC08$_H$) **Reset Value: 0008 C005$_H$**



Field	Bits	Type	Description
MOD_REV	[7:0]	rh	**Revision Number**
MOD_32B	[15:8]	rh	**32-Bit Module Enable** C0$_H$ A value of C0$_H$ in this field indicates a 32-bit module with a 32-bit module ID register.
MOD	[31:16]	rh	**Module Identification Number** 08$_H$ For module identification.

3 System Control Unit (SCU)

The System Control Unit (SCU) of the TC1797 handles all system control tasks beside the debug related tasks which are controlled by the OCDS/Cerberus.

The SCU contains the following functional sub-blocks:

- Clock Control (see **Section 3.1**)
- Reset Operation (see **Section 3.2**)
- External Interface (see **Section 3.3**)
- Power Management (see **Section 3.4**)
- Software Boot Support (see **Section 3.5**)
- SRAM Parity Control (see **Section 3.6**)
- Die Temperature Measurement (see **Section 3.7**)
- Watchdog Timer (see **Section 3.8**)
- Emergency Stop Control (see **Section 3.9**)
- Interrupt Generation (see **Section 3.10**)
- NMI Trap Generation (see **Section 3.11**)
- SCU registers and Address map (see **Section 3.12**)
- SCU register overview table (see **Table 3-20**)

3.1 Clock System Overview

This section describes the TC1797 clock system. Topics covered include clock generation and the operation of clock circuitry.

The TC1797 clock system provides the following functions:

- Acquires and buffers incoming clock signals to create a master clock frequency
- Distributes in-phase synchronized clock signals throughout the TC1797's entire clock tree
- Divides the master clock frequency into lower frequencies required by the different modules for operation
- Reduces electromagnetic interference (EMI) by switching off unused modules

Figure 3-1 shows the structure of the TC1797 clock system. The master clock f_{PLL} is generated by the oscillator circuit and the PLL (phase-locked loop) unit (see **Section 3-2**).

The functionality of the control blocks shown in **Figure 3-1** varies depending on the functional unit being controlled. Some functional units such as the watchdog timer, are directly driven by the system clock. The implemented clock control register options are described for each module in the module chapter itself.

All clock control registers CLC and the fractional divider registers FDR are Endinit-protected.

Features of the TC1797 Clock System

- PLL operation for multiplying clock source by different factors
- Direct drive capability for direct clocking
- Comfortable state machine for secure switching between Freerunning Mode / Normal Mode and Prescaler Mode

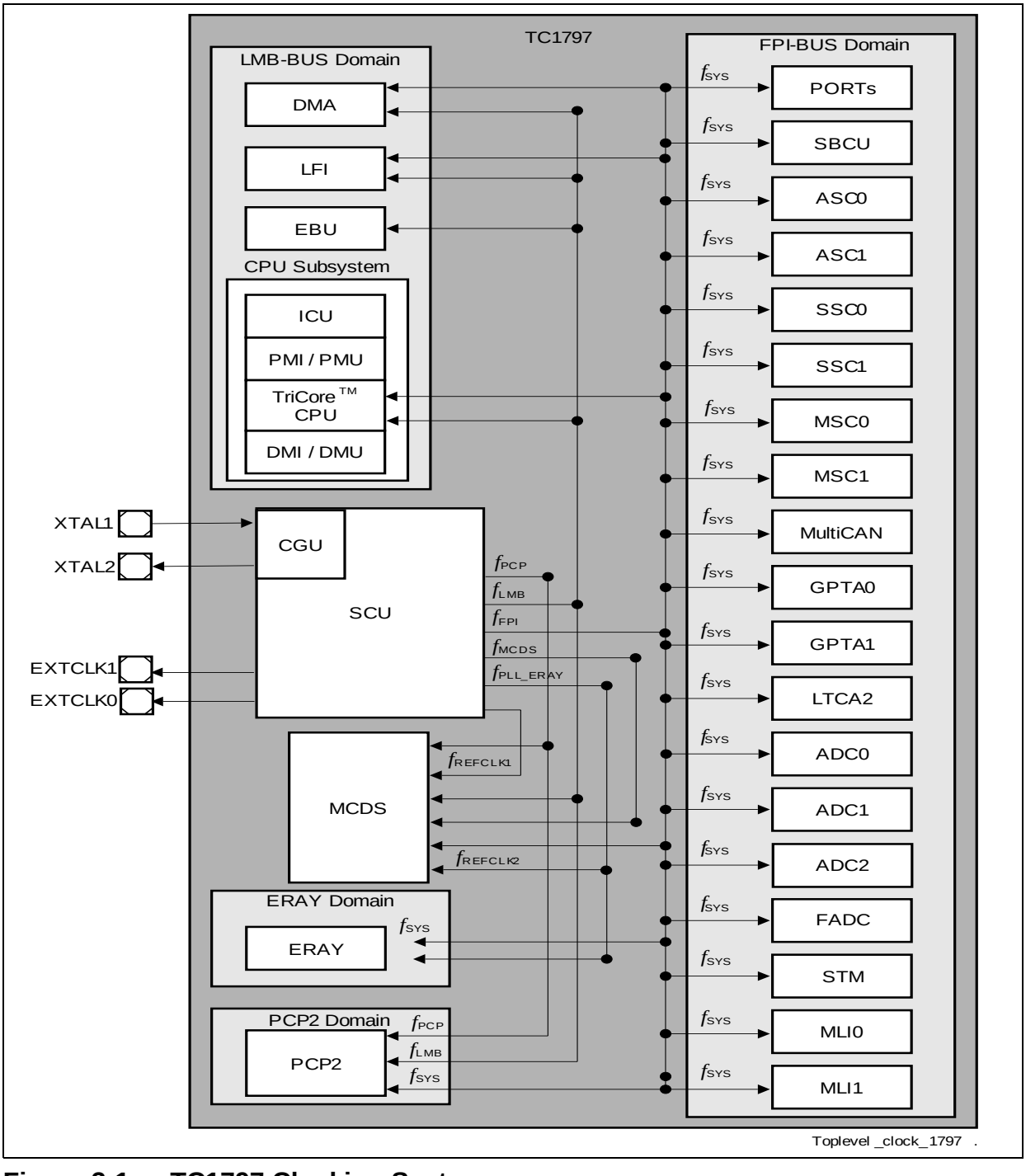


Figure 3-1 TC1797 Clocking System

3.1.1 Clock Generation Unit

The Clock Generation Unit (CGU) allows a very flexible clock generation for the TC1797. During user program execution the frequency can be programmed for an optimal ratio between performance and power consumption.

3.1.1.1 Overview

The CGU in the TC1797 consists of one oscillator circuit (OSC), two Phase-Locked Loop modules (PLL and PLL_ERAY) and a Clock Control Unit (CCU). The CGU can convert a low-frequency external clock signal to a high-speed internal clock.

The CGU provides clock signals for the different parts of the device that can be configured depending on the application needs within certain limits.

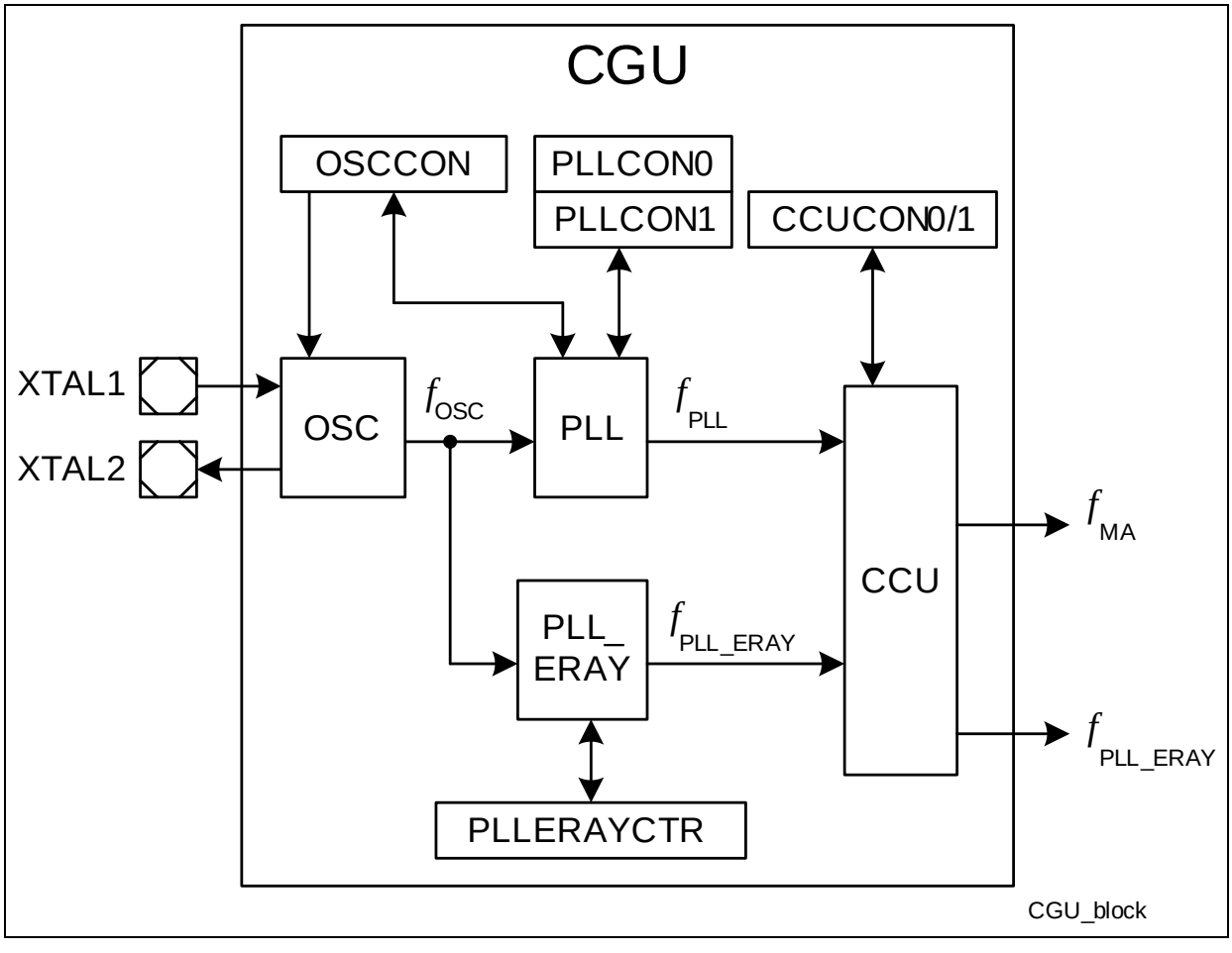


Figure 3-2 Clock Generation Unit Block Diagram

The following sections describe the different parts of the CGU.

3.1.1.2 Oscillator Circuit (OSC)

The oscillator circuit, a Pierce oscillator, is designed to work with both an external crystal oscillator or an external stable clock source, consists of an inverting amplifier with XTAL1 as input, and XTAL2 as output with an integrated feedback resistor.

Figure 3-3 shows the recommended external circuitries for both operating modes, External Crystal Mode and External Input Clock Mode.

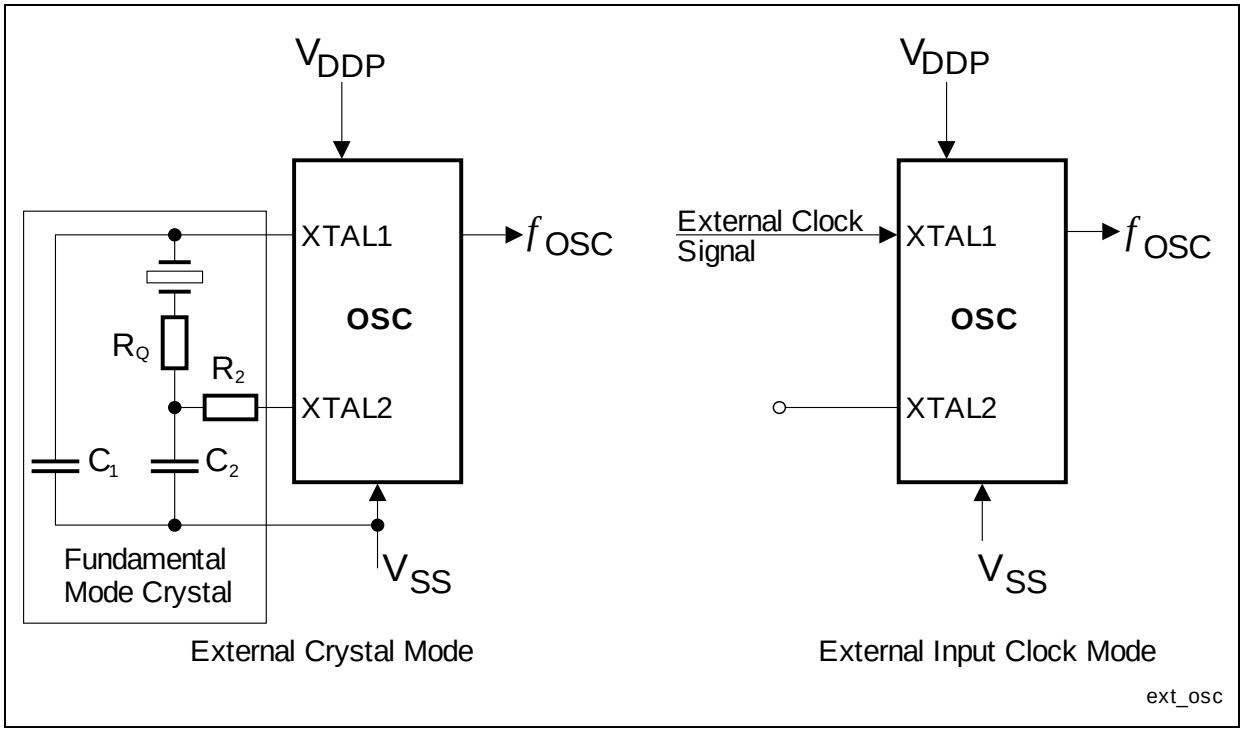


Figure 3-3 TC1797 External Circuitry for the Internal Oscillator

External Input Clock Mode

When supplying the clock signal directly, not using an external crystal and bypassing the oscillator, the input frequency needs to be equal or greater than PLL VCO input frequency (the value is listed in the Data Sheet).

When using an external clock signal it must be connected to XTAL1. XTAL2 is left open (unconnected).

External Crystal Mode

When using an external crystal, its frequency can be within the allowed range. An external oscillator load circuitry must be used, connected to both pins, XTAL1 and XTAL2. Additionally are necessary, two load capacitances C_1 and C_2, and depending on the crystal type, a series resistor R_2 to limit the current. A test resistor R_Q may be temporarily inserted to measure the oscillation allowance (negative resistance) of the

oscillator circuitry. R_Q values are typically specified by the crystal vendor. The C_1 and C_2 values shown in the Data Sheet can be used as starting points for the negative resistance evaluation and for non-productive systems. The exact values and related operating range are dependent on the crystal frequency and have to be determined and optimized together with the crystal vendor using the negative resistance method. Oscillation measurement with the final target system is strongly recommended to verify the input amplitude at XTAL1 and to determine the actual oscillation allowance (margin negative resistance) for the oscillator-crystal system.

The oscillator can also be used in combination with a ceramic resonator. The final circuitry must be also verified by the resonator vendor.

Oscillator Run Detection

See **Oscillator Watchdog**.

3.1.1.3 Phase-Locked Loop (PLL) Module

The PLL can convert a low-frequency external clock signal to a high-speed internal clock for maximum performance. The PLL also has fail-safe logic that detects degenerate external clock behavior such as abnormal frequency deviations or a total loss of the external clock. It can execute emergency actions if it loses its lock on the external clock.

This module is a phase locked loop for integer frequency synthesis. It allows the use of input and output frequencies of a wide range by varying the different divider factors.

Features

- VCO lock detection
- 4-bit input divider **P:** (divide by PDIV+1)
- 7-bit feedback divider **N**: (multiply by NDIV+1)
- 7-bit output divider **K1 or K2:** (divide by either by K1DIV+1 or K2DIV+1)
- Oscillator Watchdog
  - Detecting too low input frequencies
  - Detecting too high input frequencies
- Different operating modes
  - Prescaler Mode
  - Freerunning Mode
  - Normal Mode
- VCO Power Down
- Glitchless switching between both K-Dividers
- Glitchless switching between Normal Mode and Prescaler Mode

PLL Functional Description

The PLL consists of a Voltage Controlled Oscillator (VCO) with a feedback path. A divider in the feedback path (N-Divider) divides the VCO frequency down. The resulting frequency is then compared with the externally provided and divided frequency (P-Divider). The phase detection logic determines the difference between the two clocks and accordingly controls the frequency of the VCO (f_{VCO}). A PLL lock detection unit monitors and signals this condition. The phase detection logic continues to monitor the two clocks and adjusts the VCO clock if required. The PLL output clock f_{PLL} is derived from the VCO clock by the K2-Divider or from the oscillator clock and the K1-Divider.

The following figure shows the PLL block structure.

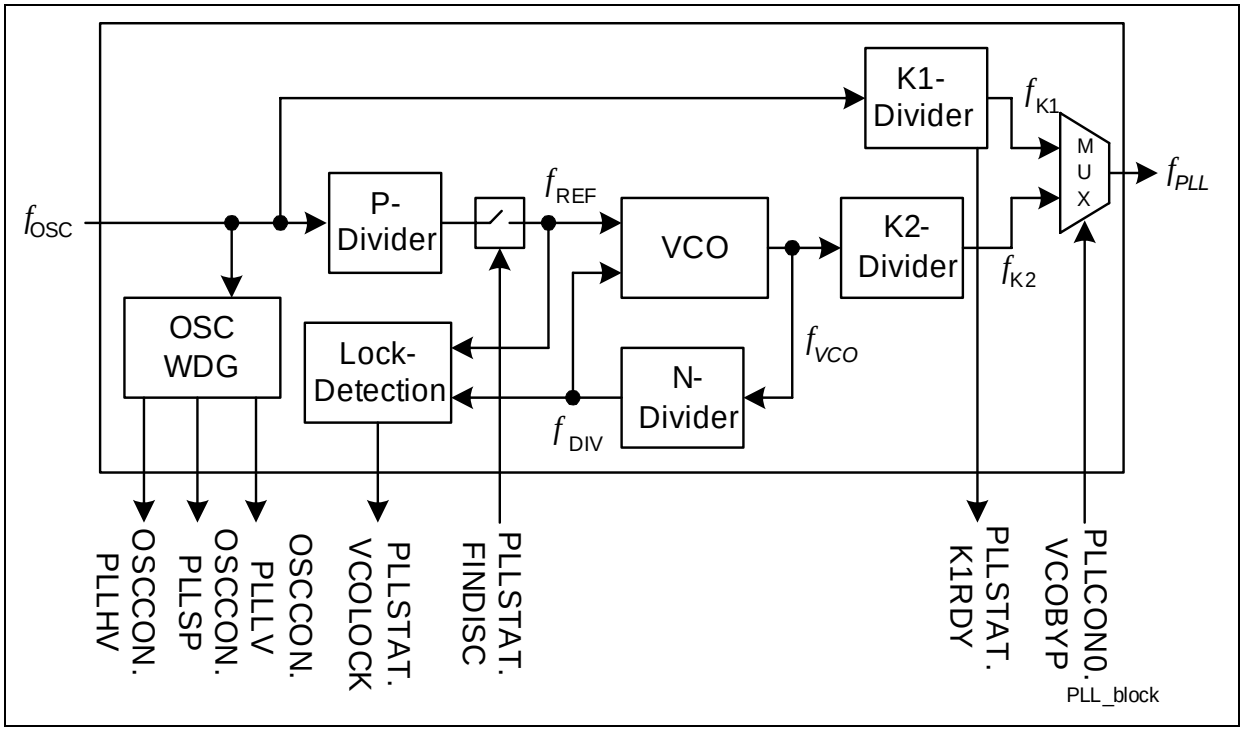


Figure 3-4 PLL Block Diagram

Clock Source Control

The PLL clock f_{PLL} is generated from f_{OSC} in one of three software selectable modes:

- Normal Mode
- Prescaler Mode
- Freerunning Mode

Normal Mode

In Normal Mode the input frequency f_{OSC} is divided down by a factor P, multiplied by a factor N and then divided down by a factor K2.

The output frequency is given by

(3.1)

$$f_{PLL} = \frac{N}{P \cdot K2} \cdot f_{OSC}$$

Prescaler Mode

In Prescaler Mode the reference frequency f_{OSC} is only divided down by a factor K1.

The output frequency is given by

(3.2)

$$f_{PLL} = \frac{f_{OSC}}{K1}$$

Freerunning Mode

In Freerunning Mode the base frequency output of the Voltage Controlled Oscillator (VCO) $f_{VCObase}$ is only divided down by a factor K2.

The output frequency is given by

(3.3)

$$f_{PLL} = \frac{f_{VCObase}}{K2}$$

Oscillator Watchdog (OSC_WDT)

The oscillator watchdog monitors the incoming clock frequency f_{OSC} from OSC. A stable and defined input frequency is a mandatory requirement for operation in both Prescaler Mode and Normal Mode. For operation in Freerunning Mode no f_{OSC} input frequency is required. Therefore this mode is selected automatically after each Application Reset. In addition for the Normal Mode it is required that the input frequency f_{OSC} is in a certain frequency range to obtain a stabile master clock from the VCO part.

The expected input frequency is selected via the bit field OSCCON.OSCVAL. The OSC_WDT checks for too low frequencies and for too high frequencies.

The frequency that is monitored is f_{OSCREF} which is derived for f_{OSC}.

(3.4)

$$f_{OSCREF} = \frac{f_{OSC}}{OSCVAL + 1}$$

The divider value OSCCON.OSCVAL has to be selected in a way that f_{OSCREF} is 2.5 MHz.

Note: f_{OSCREF} has to be within the range of 2 MHz to 3 MHz and should be as close as possible to 2.5 MHz.

Before configuring the OSC_WDT function all the trap options should be disabled in order to avoid unintended traps. Thereafter the value of OSCCON.OSCVAL can be changed. Then the OSC_WDT should be reset by setting OSCCON.OSCRES. This requests the start of OSC_WDT monitoring with the new configuration. When the expected positive monitoring results of OSCCON.PLLLV and / or OSCCON.PLLHV are set the input frequency is within the expected range. As setting OSCCON.OSCRES clears all three bits OSCCON.PLLSP, OSCCON.PLLLV, and OSCCON.PLLHV all three trap status flags will be set. Therefore all three flags should be cleared before the trap generation is enabled again. The trap disabling-clearing-enabling sequence should also be used if only bit OSCCON.OSCRES is set without any modification of OSCCON.OSCVAL.

Configuration and Operation of the Freerunning Mode

In Freerunning Mode, the PLL is running at its VCO base frequency and f_{PLL} is derived from f_{VCO} only by the K2-Divider.

The Freerunning Mode is entered after each Application Reset.

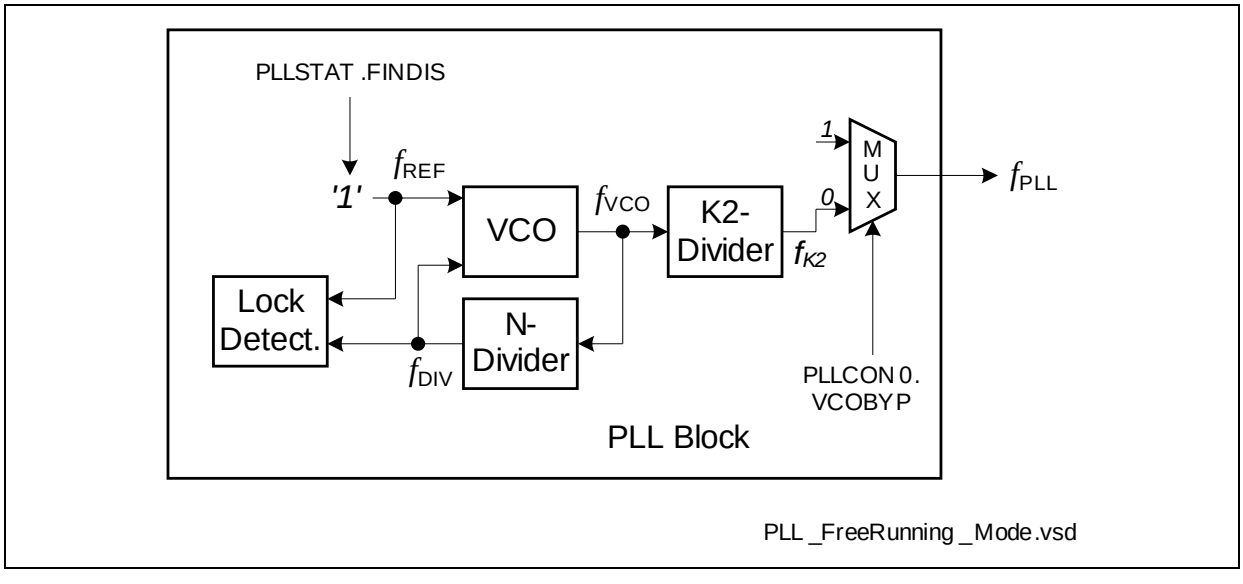


Figure 3-5 PLL Free-Running Mode Diagram

The output frequency is given by

(3.5)

$$f_{PLL} = \frac{f_{VCObase}}{K2}$$

The Freerunning Mode is selected by the following settings

- PLLCON0.VCOBYP = 0
- PLLCON0.SETFINDIS = 1

The Freerunning Mode is entered when

- PLLSTAT.FINDIS = 1
- AND
- PLLSTAT.VCOBYST = 0

Operation on the Freerunning Mode does not require an input clock frequency of f_{OSC}. The Freerunning Mode is automatically entered on a PLL VCO Loss-of-Lock event if bit PLLCON0.OSCDISCDIS is cleared. This mechanism allows a fail-safe operation of the PLL as in emergency cases still a clock is available.

The frequency of the Freerunning Mode $f_{VCObase}$ is listed in the Data Sheet.

Note: Changing the system operation frequency by changing the value of the K2-Divider has a direct coupling to the power consumption of the device. Therefore this has to be done carefully.

Depending on the selected divider value of the K2-Divider the duty cycle of the clock is selected. This can have an impact for the operation with an external communication interface. The duty cycles values for the different K2-divider values are defined in the Data Sheet.

Configuration and Operation of the Prescaler Mode

In Prescaler Mode, the PLL is running at the external frequency f_{OSC} and f_{PLL} is derived from f_{OSC} only by the K1-Divider.

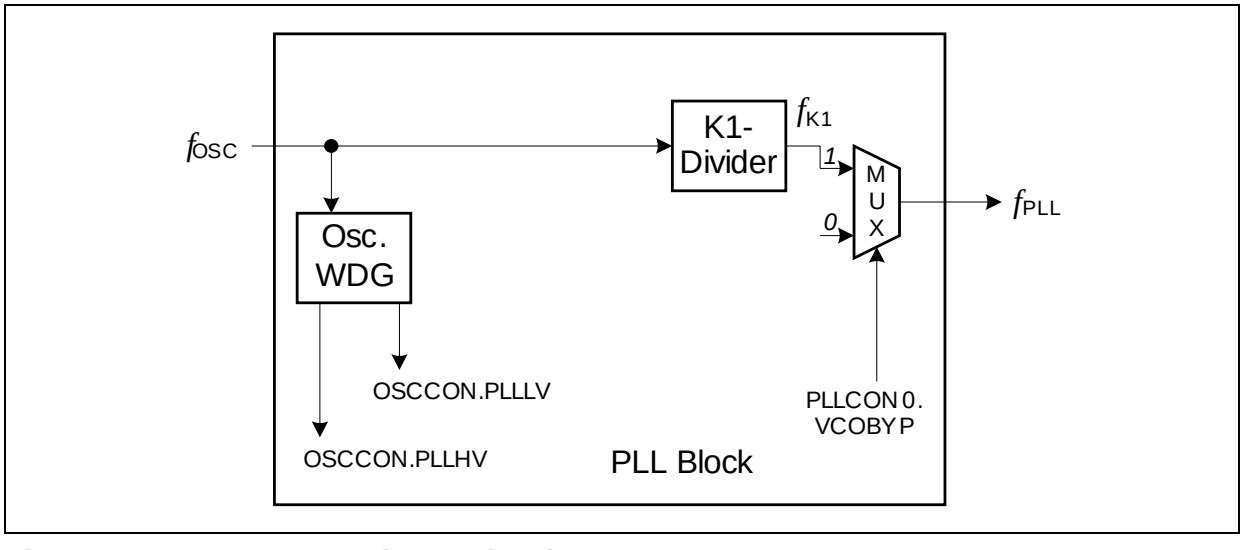


Figure 3-6 PLL Prescaler Mode Diagram

The output frequency is given by:

(3.6)

$$f_{PLL} = \frac{f_{OSC}}{K1}$$

The Prescaler Mode is selected by the following settings

- PLLCON0.VCOBYP = 1

The Prescaler Mode is entered when the following requirements are all together valid:

- PLLSTAT.VCOBYST = 1
- OSCCON.PLLLV = 1

Operation on the Prescaler Mode does require an input clock frequency of f_{OSC}. Therefore it is recommended to check and monitor if an input frequency f_{OSC} is available at all by checking OSCCON.PLLLV. For a better monitoring also the upper frequency can be monitored via OSCCON.PLLHV.

For the Prescaler Mode there are no requirements regarding the frequency of f_{OSC}.

The system operation frequency is controlled in the Prescaler Mode by the value of the K1-Divider. When the value of PLLCON1.K1DIV was changed the next update of this value should not be done before bit PLLSTAT.K1RDY is set.

Note: Changing the system operation frequency by changing the value of the K1-Divider has a direct coupling to the power consumption of the device. Therefore this has to be done carefully.

Depending on the selected divider value of the K1-Divider the duty cycle of the clock is selected. This can have an impact for the operation with an external communication interface. The duty cycles values for the different K1-divider values are defined in the Data Sheet.

The Prescaler Mode is requested from the Freerunning or Normal Mode by setting bit PLLCON.VCOBYP. The Prescaler Mode is entered when the status bit PLLSTAT.VCOBYST is set. Before the Prescaler Mode is requested the K1-Divider should be configured with a value generating a PLL output frequency f_{PLL} that matches the one generated by the Freerunning or Normal Mode as much as possible. In this way the frequency change resulting out of the mode change is reduced to a minimum.

The Prescaler Mode is requested to be left by clearing bit PLLCON.VCOBYP. The Prescaler Mode is left when the status bit PLLSTAT.VCOBYST is cleared.

Configuration and Operation of the Normal Mode

In Normal Mode, the PLL is running at the external frequency f_{OSC} and f_{PLL} is divided down by a factor P, multiplied by a factor N and then divided down by a factor K2.

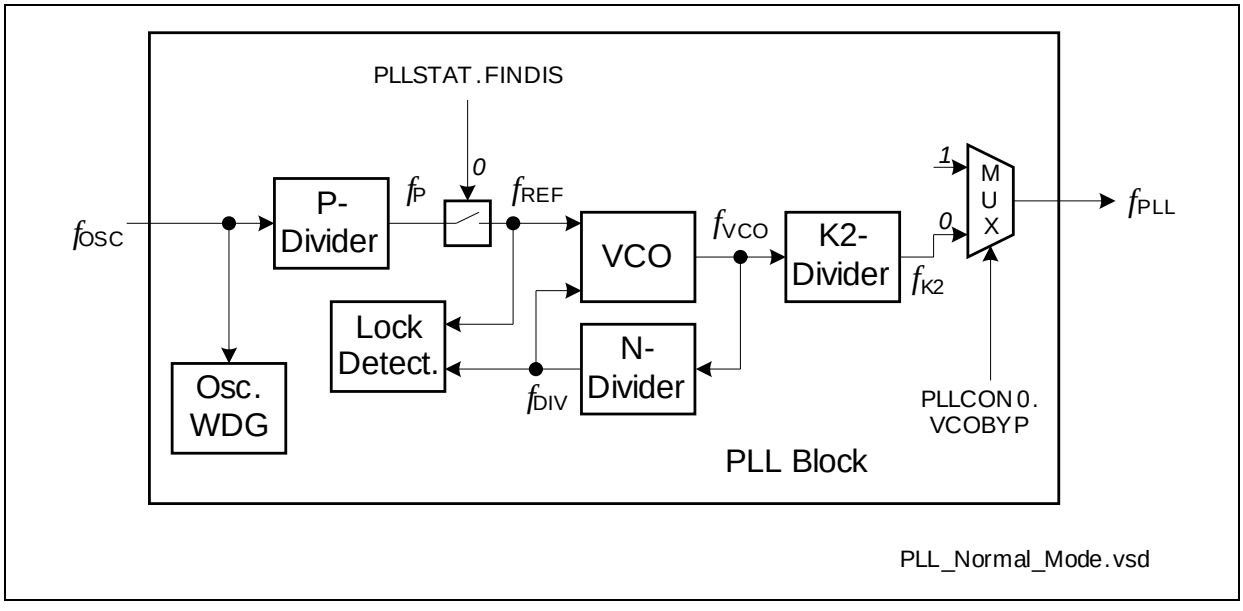


Figure 3-7 PLL Normal Mode Diagram

The output frequency is given by:

(3.7)

$$f_{PLL} = \frac{N}{P \cdot K2} \cdot f_{OSC}$$

The Normal Mode is selected by the following settings

- PLLCON0.VCOBYP = 0
- PLLCON0.CLRFINDIS = 1

The Normal Mode is entered when the following two requirements are all together valid:

- PLLSTAT.FINDIS = 0
- PLLSTAT.VCOBYST = 0
- PLLSTAT.VCOLOCK = 1
- OSCCON.PLLLV = 1
- OSCCON.PLLHV = 1

Operation on the Normal Mode does require an input clock frequency of f_{OSC}. Therefore it is recommended to check and monitor if an input frequency f_{OSC} is available at all by checking OSCCON.PLLLV. For a better monitoring also the upper frequency can be monitored via OSCCON.PLLHV.

The system operation frequency is controlled in the Normal Mode by the values of the three dividers: P, N, and K2. A modification of the two dividers P and N has a direct influence to the VCO frequency and lead to a loss of the VCO Lock status. A modification of the K2-divider has no impact on the VCO Lock status but still changes the PLL output frequency.

Note: Changing the system operation frequency by changing the value of the K2-Divider has a direct coupling to the power consumption of the device. Therefore this has to be done carefully.

When the frequency of the Normal Mode should be modified or entered the following sequence should be followed:

First the Prescaler Mode should be configured and entered. For more details see the Prescaler Mode.

The NMI trap generation for the VCO Lock should be disabled.

While the Prescaler Mode is used the Normal Mode can be configured and checked for a positive VCO Lock status. The first target frequency of the Normal Mode should be selected in a way that it matches or is only slightly higher as the one used in the Prescaler Mode. This avoids big changes in the system operation frequency and therefore power consumption when switching later from Prescaler Mode to Normal Mode. The P and N divider should be selected in the following way:

- Selecting P and N in a way that f_{VCO} is in the lower area of its allowed values leads to a slightly reduced power consumption but to a slightly increased jitter
- Selecting P and N in a way that f_{VCO} is in the upper area of its allowed values leads to a slightly increased power consumption but to a slightly reduced jitter

After the P, N, and K2 dividers are updated for the first configuration the indication of the VCO Lock status should be await (PLLSTAT.VCOLOCK = 1).

Note: It is recommended to reset the VCO Lock detection (PLLCON0.RESLD = 1) after the new values of the dividers are configured to get a defined VCO lock check time.

When this happens the switch from Prescaler Mode to Normal Mode can be done. Normal Mode is requested by clearing PLLCON.VCOBYP. The Normal Mode is entered when the status bit PLLSTAT.VCOBYST is cleared.

Now the Normal Mode is entered. The NMI status flag for the VCO Lock trap should be cleared and then enabled again. The intended PLL output target frequency can now be configured by changing only the K2-Divider.

Depending on the selected divider value of the K2-Divider the duty cycle of the clock is selected. This can have an impact for the operation with an external communication interface. The duty cycles values for the different K2-divider values are defined in the Data Sheet. This can result in multiple changes of the K2-Divider to avoid to big frequency changes. Between the update of two K2-Divider values 6 cycles of f_{PLL} should be waited.

PLL VCO Lock Detection

The PLL has a lock detection that supervises the VCO part of the PLL in order to differentiate between stable and instable VCO circuit behavior. The lock detector marks the VCO circuit and therefore the output f_{VCO} of the VCO as instable if the two inputs f_{REF}

and f_{DIV} differ too much. Changes in one or both input frequencies below a level are not marked by a loss of lock because the VCO can handle such small changes without any problem for the system.

PLL VCO Loss-of-Lock Event

The PLL may become unlocked, caused by a break of the crystal or the external clock line. In such a case, an NMI trap is generated if the according NMI trap is enabled. Additionally, the OSC clock input f_{OSC} is disconnected from the PLL VCO to avoid unstable operation due to noise or sporadic clock pulses coming from the oscillator circuit. Without a clock input f_{OSC}, the PLL gradually slows down to its VCO base frequency and remains there. This automatic feature can be disabled by setting bit PLLCON0.OSCDISCDIS. If this bit is cleared the OSC clock remains connected to the VCO.

VCO Power Down Mode

The PLL offers a VCO Power Down Mode. This mode can be entered to save power within the PLL. The VCO Power Down Mode is entered by setting bit PLLCON0.VCOPWD. While the PLL is in VCO Power Down Mode only the Prescaler Mode is operable. Please note that selecting the VCO Power Down Mode does not automatically switch to the Prescaler Mode. So before the VCO Power Down Mode is entered the Prescaler Mode must be active.

PLL Power Down Mode

The PLL offers a Power Down Mode. This mode can be entered to save power if the PLL is not needed at all. The Power Down Mode is entered by setting bit PLLCON0.PLLPWD. While the PLL is in Power Down Mode no PLL output frequency is generated.

3.1.1.4 ERAY Phase-Locked Loop (PLL_ERAY) Module

The PLL_ERAY can convert a low-frequency external clock signal to a high-speed internal clock for maximum performance. The PLL_ERAY also has fail-safe logic that detects degenerate external clock behavior such as abnormal frequency deviations or a total loss of the external clock. It can execute emergency actions if it loses its lock on the external clock.

This module is a phase locked loop for integer frequency synthesis. It allows the use of input and output frequencies of a wide range by varying the different divider factors.

Features

Here is a brief overview of the functions that are offered by the PLL_ERAY.

- VCO lock detection
- 3-bit input divider **P:** (divide by PDIV+1)

- 5-bit feedback divider **N**: (multiply by NDIV+1)
- 4-bit output divider **K:** (divide by KDIV+1)
- Oscillator Run Detection
- Different operating modes
  - Prescaler Mode
  - Freerunning Mode
  - Normal Mode
- PLL_ERAY Power Down Mode
- Glitchless switching between Normal Mode and Prescaler Mode

PLL_ERAY Functional Description

The PLL_ERAY consists of a Voltage Controlled Oscillator (VCO) with a feedback path. A divider in the feedback path (N-Divider) divides the VCO frequency down. The resulting frequency is then compared with the externally provided and divided frequency (P-Divider). The phase detection logic determines the difference between the two clocks and accordingly controls the frequency of the VCO (f_{VCO}). A PLL_ERAY lock detection unit monitors and signals this condition. The phase detection logic continues to monitor the two clocks and adjusts the VCO clock if required. The PLL_ERAY output clock f_{PLL_ERAY} is derived from the VCO clock or from the oscillator clock by the K-Divider.

The following figure shows the PLL_ERAY block structure.

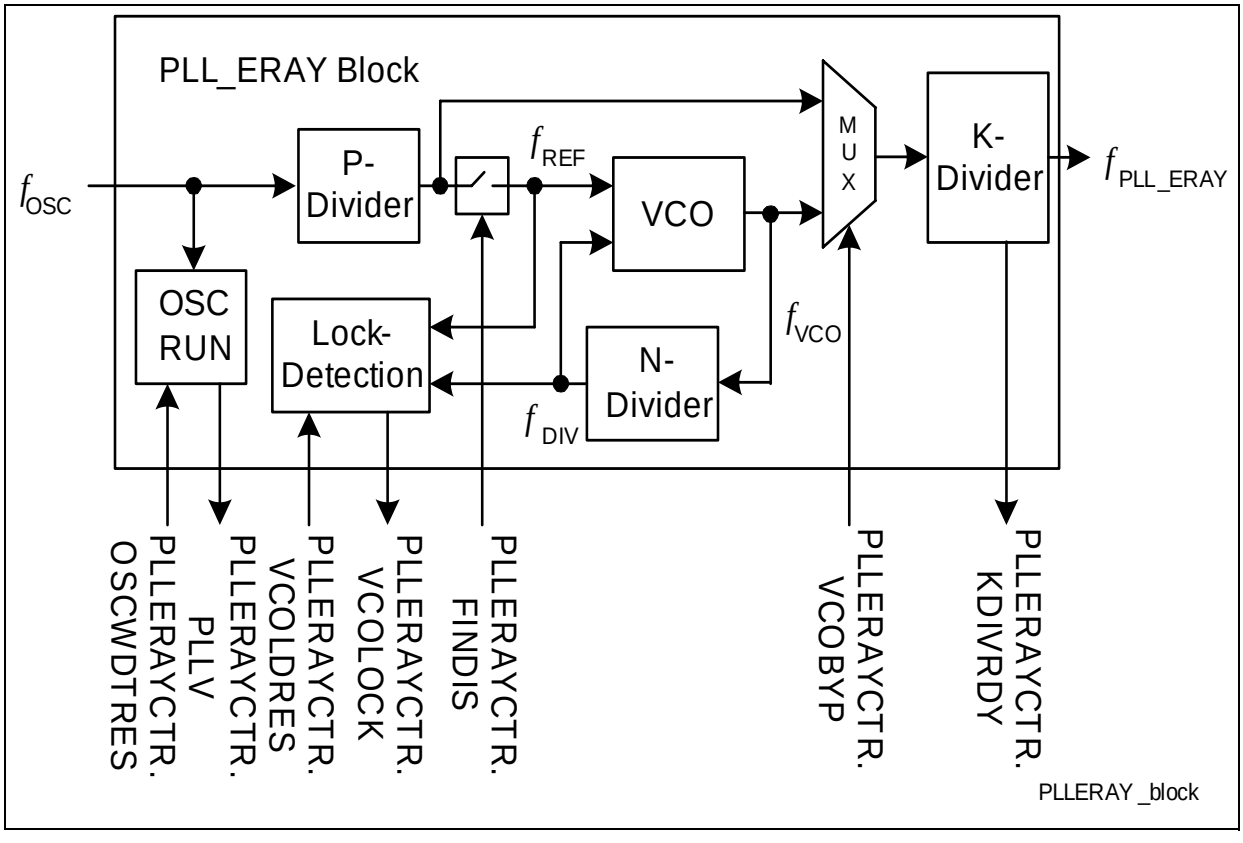


Figure 3-8 PLL_ERAY Block Diagram

Clock Source Control

The PLL_ERAY clock f_{PLL_ERAY} is generated from f_{OSC} in one of three software selectable modes:

- Normal Mode
- Prescaler Mode
- Freerunning Mode

Normal Mode

In Normal Mode the input frequency f_{OSC} is divided down by a factor P, multiplied by a factor N and then divided down by a factor K.

The output frequency is given by

(3.8)

$$f_{PLLERAY} = \frac{N}{P \cdot K} \cdot f_{OSC}$$

Prescaler Mode

In Prescaler Mode the reference frequency f_{OSC} is divided down by a factor P $\times$ K.

The output frequency is given by

(3.9)

$$f_{PLLERAY} = \frac{f_{OSC}}{P \times K}$$

Freerunning Mode

In Freerunning Mode the base frequency output of the Voltage Controlled Oscillator (VCO) $f_{VCObase}$ is only divided down by a factor K.

The output frequency is given by

(3.10)

$$f_{PLLERAY} = \frac{f_{VCObase}}{K}$$

Oscillator Run Detection

The oscillator run detection monitors the incoming clock frequency f_{OSC} from OSC. A stabile and defined input frequency is a mandatory requirement for operation in both Prescaler Mode and Normal Mode. For operation in Freerunning Mode no f_{OSC} input frequency is required.

Before configuring the oscillator run detection function the trap generation should be disabled in order to avoid unintended traps. Then the oscillator run detection should be reset by setting PLLERAYCTR.OSCWDTRES. This requests the start of oscillator run detection monitoring with the new configuration. When the expected positive monitoring results of PLLERAYCTR.PLLV is set the input frequency is above a certain frequency. As setting PLLERAYCTR.OSCWDTRES sets the trap status flag. Therefore the flag should be cleared before the trap generation is enabled again.

Configuration and Operation of the Freerunning Mode

In Freerunning Mode, the PLL_ERAY is running at its VCO base frequency and f_{PLL_ERAY} is derived from f_{VCO} only by the K-Divider.

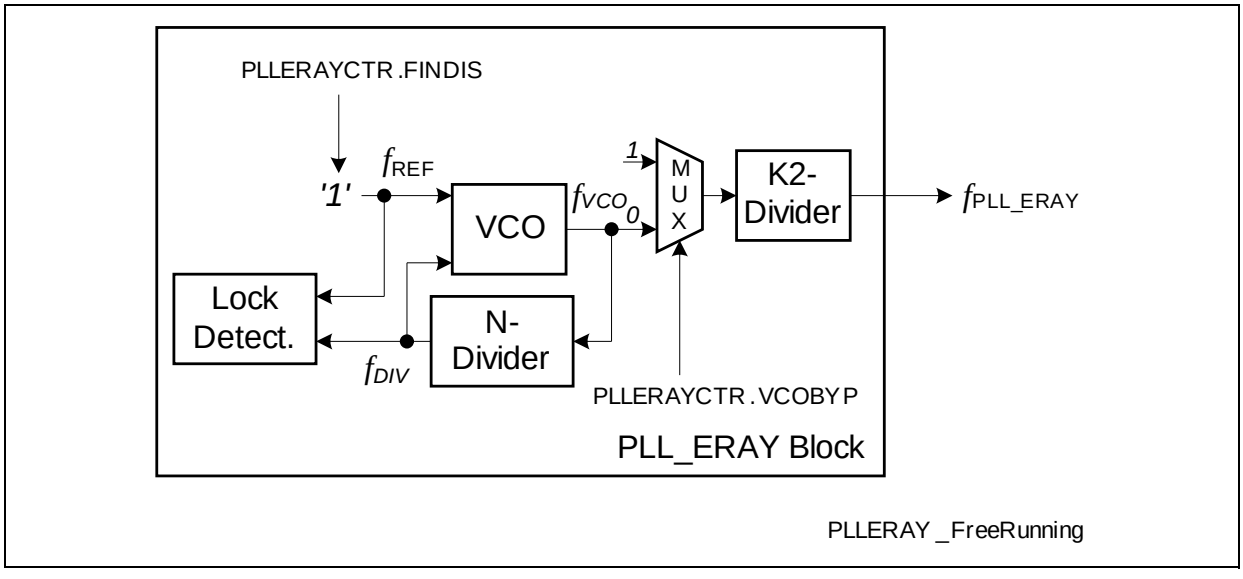


Figure 3-9 PLL_ERAY Free-Running Mode Diagram

The output frequency is given by

(3.11)

$$f_{PLLERAY} = \frac{f_{VCObase}}{K}$$

The Freerunning Mode is selected and entered by the following settings

- PLLERAYCTR.VCOBYP = 0
- PLLERAYCTR.FINDIS = 1

Operation on the Freerunning Mode does not require an input clock frequency of f_{OSC}. The Freerunning Mode is automatically entered on a PLL_ERAY VCO Loss-of-Lock event. This mechanism allows a fail-safe operation of the PLL as in emergency cases still a clock is available.

The frequency of the Freerunning Mode $f_{VCObase}$ is listed in the Data Sheet.

Note: Changing the system operation frequency by changing the value of the K-Divider has a direct coupling to the power consumption of the device. Therefore this has to be done carefully.

Depending on the selected divider value of the K-Divider the duty cycle of the clock is selected. This can have an impact for the operation with an external communication interface. The duty cycles values for the different K-divider values are defined in the Data Sheet. When the value of PLLERAYCTR.KDIV was changed the next update of this value should not be done before bit PLLERAYCTR.KDIVRDY is set.

Configuration and Operation of the Prescaler Mode

In Prescaler Mode, the PLL_ERAY is running at the external frequency f_{OSC} and f_{PLL_ERAY} is derived from f_{OSC} only by the P- and K-Divider.

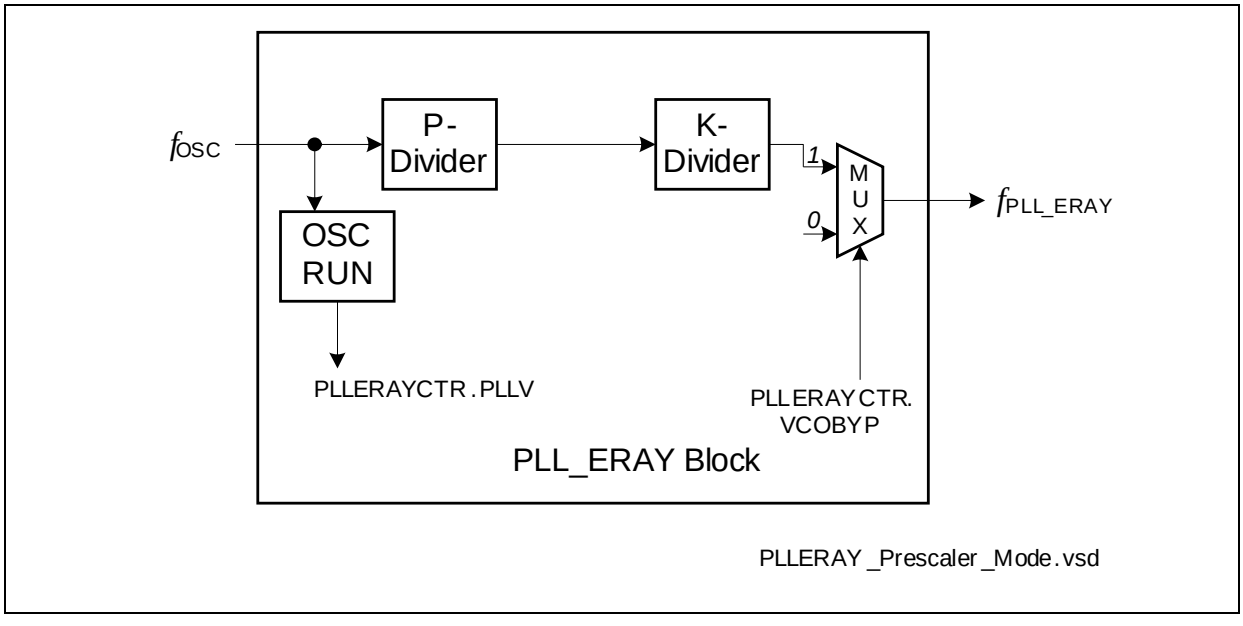


Figure 3-10 PLL_ERAY Prescaler Mode Diagram

The output frequency is given by:

(3.12)

$$f_{PLLERAY} = \frac{f_{OSC}}{P \times K}$$

The Prescaler Mode is selected and entered by the following setting

- PLLERAYCTR.VCOBYP = 1

Operation on the Prescaler Mode does require an input clock frequency of f_{OSC}. Therefore it is recommended to check and monitor if an input frequency f_{OSC} is available at all by checking PLLERAYCTR.PLLV.

For the Prescaler Mode there are no requirements regarding the frequency of f_{OSC}.

The system operation frequency is controlled in the Prescaler Mode by the value of the K-Divider (Please note that the P-Divider has to be set to 1:1 of the PLL_ERAY and therefore the P-Divider can be ignored here). When the value of PLLERAYCTR.KDIV was changed the next update of this value should not be done before bit PLLERAYCTR.KDIVRDY is set.

Note: Changing the system operation frequency by changing the value of the K-Divider has a direct coupling to the power consumption of the device. Therefore this has to be done carefully.

Depending on the selected divider value of the K-Divider the duty cycle of the clock is selected. This can have an impact for the operation with an external communication interface. The duty cycles values for the different K-divider values are defined in the Data Sheet.

The Prescaler Mode is requested from the Freerunning or Normal Mode by setting bit PLLERAYCTR.VCOBYP.

The Prescaler Mode is requested to be left by clearing bit PLLERAYCTR.VCOBYP.

Configuration and Operation of the Normal Mode

In Normal Mode, the PLL_ERAY is running at the external frequency f_{OSC} and $f_{PLLERAY}$ is divided down by a factor P, multiplied by a factor N and then divided down by a factor K.

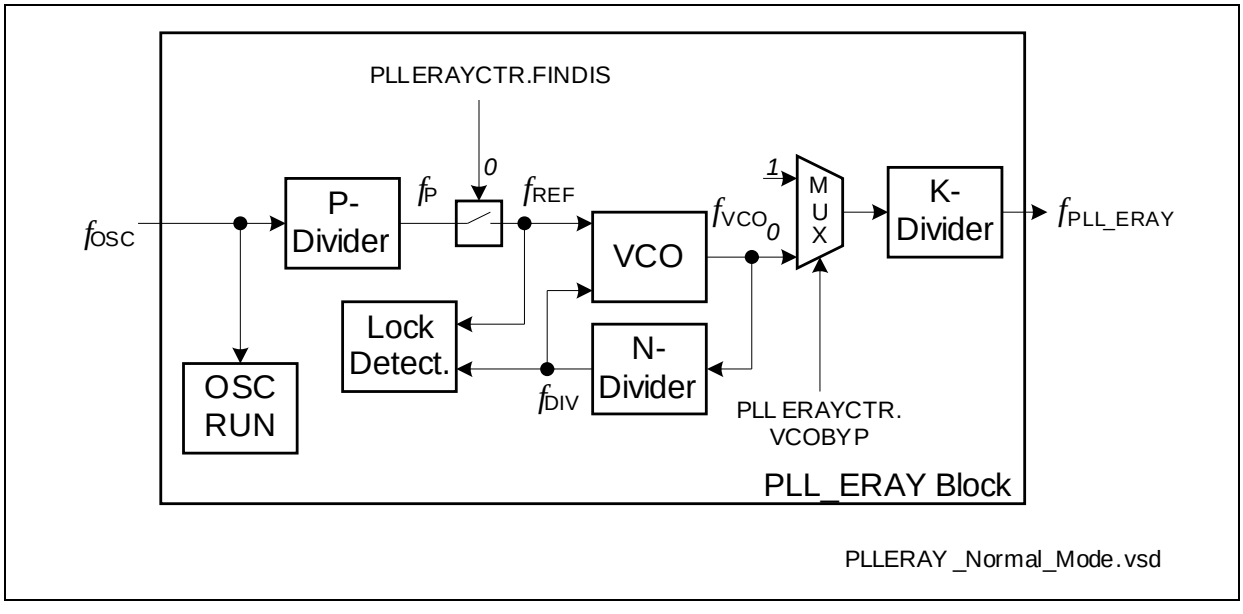


Figure 3-11 PLL_ERAY Normal Mode Diagram

The output frequency is given by:

(3.13)

$$f_{PLLERAY} = \frac{N}{P \cdot K} \cdot f_{OSC}$$

The Normal Mode is selected and entered by the following settings

- PLLERAYCTR.VCOBYP = 0
- PLLERAYCTR.FINDIS = 0

Operation on the Normal Mode does require an input clock frequency of f_{OSC}. Therefore it is recommended to check and monitor if an input frequency f_{OSC} is available at all by checking PLLERAYCTR.PLLV.

The system operation frequency is controlled in the Normal Mode by the values of the three dividers: P, N, and K. Please note that the P-Divider has to be set to 1:1 (PDIV = 0) of the PLL_ERAY and therefore the P-Divider can be ignored here. A modification of the N-Divider as a direct influence to the VCO frequency and lead to a loss of the VCO Lock status. A modification of the K-divider has no impact on the VCO Lock status but still changes the PLL_ERAY output frequency.

Note: Changing the system operation frequency by changing the value of the K-Divider has a direct coupling to the power consumption of the device. Therefore this has to be done carefully.

When the frequency of the Normal Mode should be modified or entered the following sequence should be followed:

First the Prescaler Mode should be configured and entered. For more details see the Prescaler Mode.

The NMI trap generation for the VCO Lock should be disabled.

While the Prescaler Mode is used the Normal Mode can be configured and checked for a positive VCO Lock status. The first target frequency of the Normal Mode should be selected in a way that it matches or is only slightly higher as the one used in the Prescaler Mode. This avoids big changes in the system operation frequency and therefore power consumption when switching later from Prescaler Mode to Normal Mode. The N divider should be selected in the following way:

- Selecting N in a way that f_{VCO} is in the lower area of its allowed values leads to a slightly reduced power consumption but to a slightly increased jitter
- Selecting N in a way that f_{VCO} is in the upper area of its allowed values leads to a slightly increased power consumption but to a slightly reduced jitter

After the N, and K dividers are updated for the first configuration the indication of the VCO Lock status should be await (PLLERAYCTR.VCOLOCK = 1).

Note: It is recommended to reset the VCO Lock detection (PLLERAYCTR.VCOLDRES = 1) after the new values of the dividers a configured in order to get a defined VCO lock check time.

When this happens the switch from Prescaler Mode to Normal Mode can be done. Normal Mode is requested by clearing PLLERAYCTR.VCOBYP.

Now the Normal Mode is entered. The NMI status flag for the VCO Lock trap should be cleared and then enabled again. The intended PLL_ERAY output target frequency can not be configured by changing only the K-Divider.

Depending on the selected divider value of the K-Divider the duty cycle of the clock is selected. This can have an impact for the operation with an external communication interface. The duty cycles values for the different K-divider values are defined in the Data Sheet. This can result in multiple changes of the K-Divider to avoid to big frequency

changes. When the value of PLLERAYCTR.KDIV was changed the next update of this value should not be done before bit PLLERAYCTR.KDIVRDY is set.

PLL_ERAY VCO Lock Detection

The PLL_ERAY has a lock detection that supervises the VCO part of the PLL_ERAY in order to differentiate between stable and instable VCO circuit behavior. The lock detector marks the VCO circuit and therefore the output f_{VCO} of the VCO as instable if the two inputs f_{REF} and f_{DIV} differ too much. Changes in one or both input frequencies below a level are not marked by a loss of lock because the VCO can handle such small changes without any problem for the system.

PLL_ERAY Loss of Oscillator Lock Operation

If PLL_ERAY loses its lock to the external clock/oscillator, it de-asserts its lock bit PLLERAYCTR.VCOLOCK. PLL_ERAY loss of lock detection should be triggered by setting the restart lock detection bit PLLERAYCTR.VCOLDRES = 1. The PLL_ERAY behaves as if the Freerunning Mode is selected (even so the PLLERAYCTR.FINDIS remains cleared). Due to the missing input clock the phase detector pulls the VCO down to the lower limit frequency, i.e. the free-running frequency f_{free}.

If the PLL_ERAY is operating in Normal Mode and a loss of lock event is generated a NMI request is forwarded to the trap control. In addition, the PLLERAYCTR.VCOLOCK flag is cleared. PLL_ERAY VCO gradually slows down or speeds up. The PLL_ERAY may slow down to the minimum frequency being the base frequency or to the upper frequency band limit.

Loss of Oscillator Lock Recovery

If the PLL_ERAY is in loss-of-lock mode, user software can try to re-lock the PLL_ERAY again by executing the following sequence:

1. Restarting the oscillator run detection by setting bit PLLERAYCTR.OSCWDTRES
2. Waiting until PLLERAYCTR.PLLV is set
3. If bit PLLERAYCTR.PLLV is set, then
   a) The Prescaler Mode has to be selected (PLLERAYCTR.VCOBYP = 1_B)
   b) The oscillator has to be connected to the PLL_ERAY if disconnected (PLLERAYCTR.FINDIS = 0)
   c) Setting the restart lock detection bit PLLERAYCTR.VCOLDRES = 1
   d) Waiting until the PLL_ERAY becomes locked (PLLERAYCTR.VCOLOCK = 1)
   e) When the PLLERAYCTR.VCOLOCK is set again, the Prescaler Mode can be deselected (PLLERAYCTR.VCOBYP = 0_B) and Normal Mode operation is resumed.

VCO Power Down Mode

The PLL_ERAY offers a VCO Power Down Mode. This mode can be entered to save power within the PLL_ERAY. The VCO Power Down Mode is entered by setting bit PLLERAYCTR.VCOPWD. While the PLL_ERAY is in VCO Power Down Mode only the

Prescaler Mode is operable. Please note that selecting the VCO Power Down Mode does not automatically switch to the Prescaler Mode. So before the VCO Power Down Mode is enter the Prescaler Mode must be active.

3.1.1.5 Clock Control Unit

The Clock Control Unit (CCU) receives the clock that is created by the two PLLs f_{PLL} and f_{PLL_ERAY}.

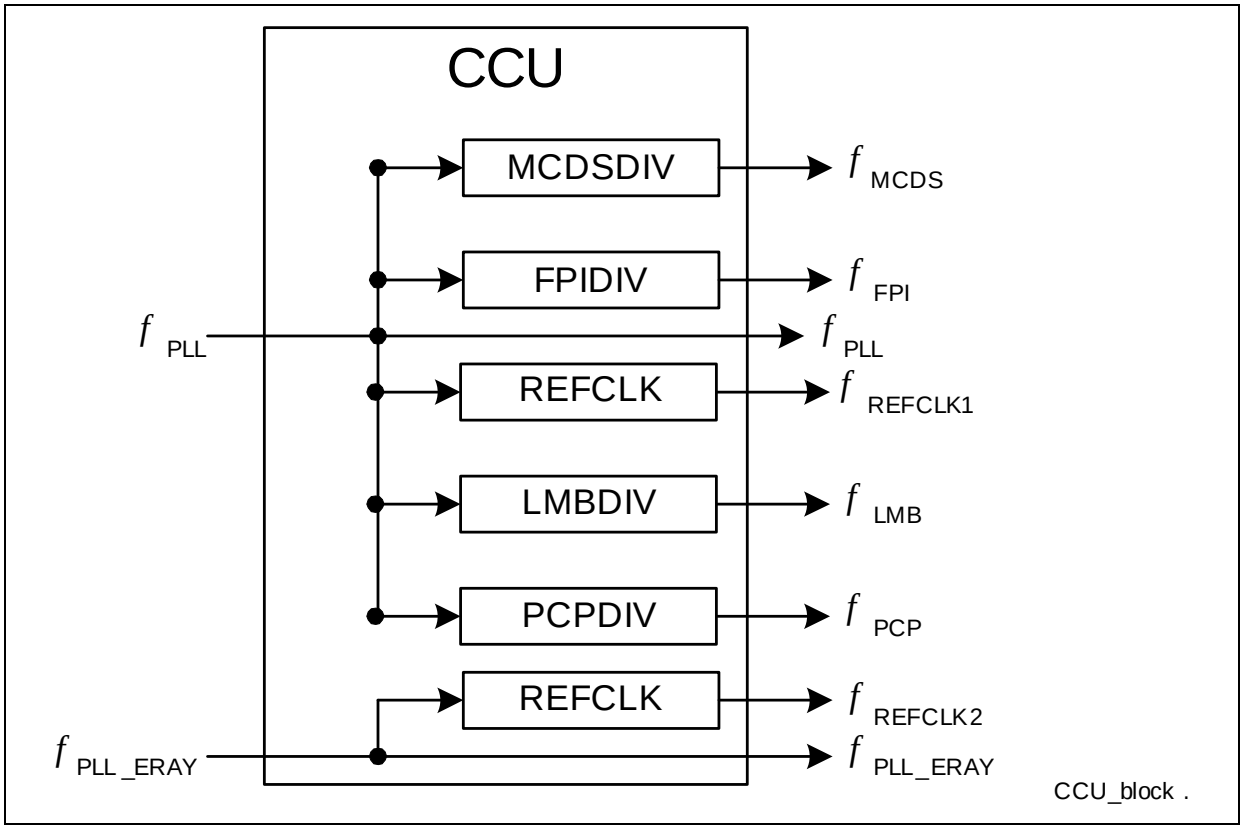


Figure 3-12 Clock Control Unit

The clocking system of the TC1797 consists of the Clock Control Unit (CCU) and the Clock Generation Unit.

There is also a fix reference clock REFCLK1 for the MCDS block which divides the master clock f_{PLL} by 24. This allows the MCDS to generate time stamps independent of the selected LMB-Bus and FPI-Bus clock speeds.

Clock Divider Limitations

There are several limitation and relations between the different clocks that could be configured via the CCUCON0 and CCUCON1 registers:

- $f_{LMB} = f_{FPI}$ or $f_{LMB} = 2 \times f_{FPI}$
- $f_{LMB} = (f_{MCDS})$ OR $(2 \times f_{MCDS})$

- $f_{LMB} = (f_{PCP})$ OR $(2 \times f_{PCP})$
- $f_{PLL} = 24 \times f_{REFCLK1}$
- $f_{PLLERAY} = 24 \times f_{REFCLK2}$
- $f_{MCDS} \geq f_{FPI}$
- $f_{PCP} \geq f_{FPI}$

3.1.1.6 External Clock Output

Two external clock outputs are provided via pins EXTCLK0 and EXTCLK1. These external clocks can be enabled/disabled via bits EXTCON.EN0 for EXTCLK0 and EXTCON.EN1 for EXTCLK1. Each of the clocks that defines a clock domain can individually be selected to be seen at pins EXTCLK0 or EXTCLK1, this is configured via bit field EXTCON.SEL0/1. Changing the content of bit field EXTCON.SEL0/1 can lead to spikes at pins EXTCLK0/1.

Note: Only the burst flash clock of the EBU is not included as the EBU provides a separate pin here.

Additionally a connection to the GPTA module is implemented to support the start-up control of an external crystal for the device clock generation. The first time before the master clock is generated based on a external crystal 1000 cycles of the crystal clock f_{OSC} should be waited before the clock control system is changed to External Crystal Mode. The 1000 cycles can be counted with the GPTA using f_{OSC} as count input for the counter.

Programmable Frequency Output for EXTCLK0

This section describes the external clock generation using the Fractional Divider.

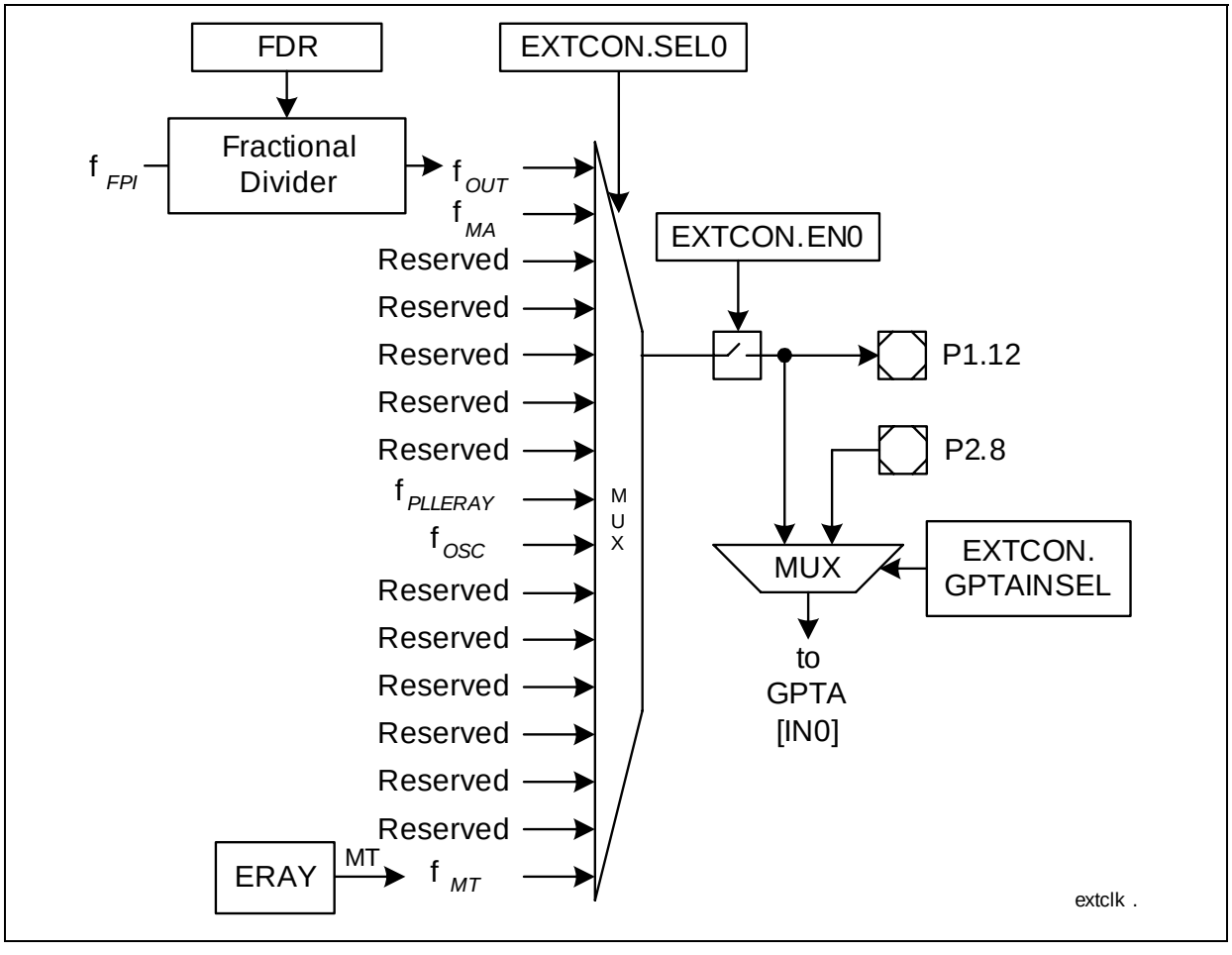


Figure 3-13 EXTCLK0 Generation

Overview

The fractional divider makes it possible to generate a external clock from the FPI-Bus clock using a programmable divider. The fractional divider divides the input clock f_{FPI} either by the factor 1/n or by a fraction of n/1024 for any value of n from 0 to 1023. This clock is thereafter divider additionally by a factor of two to guarantee a 50% duty cycle and outputs the clock, f_{OUT}. The fractional divider is controlled by the FDR register. **Figure 3-14** shows the fractional divider block diagram.

The adder logic of the fractional divider can be configured for two operating modes:

- Reload counter (addition of +1), generating an output clock pulse on counter overflow
- Adder that adds a STEP value to the RESULT value and generates an output clock pulse on counter overflow

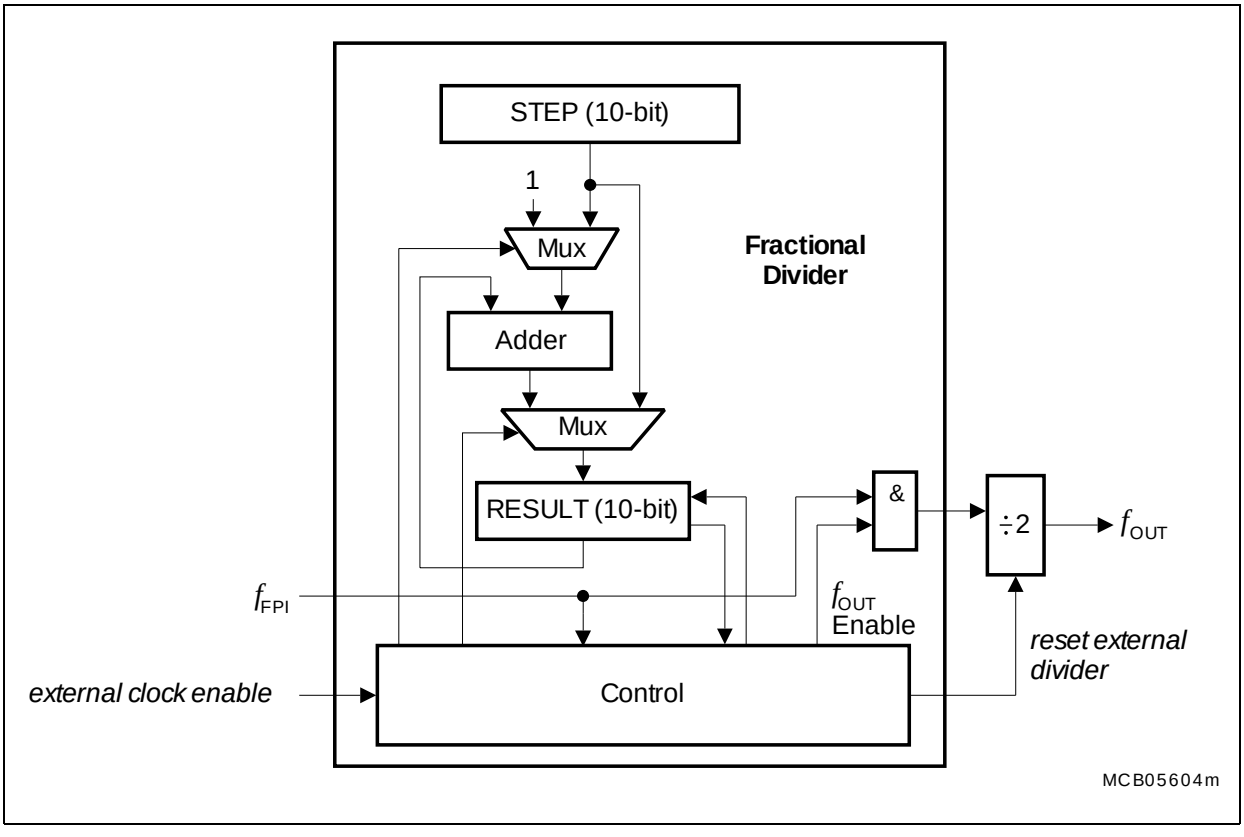


Figure 3-14 Fractional Divider Block Diagram

The adder logic of the fractional divider can be configured for two operating modes:

- **Normal Mode:** Reload counter (RESULT = RESULT + 1), generating an output clock pulse on counter overflow.
- **Fractional Divider Mode:** Adder that adds a STEP value to the RESULT value and generates an output clock pulse on counter overflow.

Fractional Divider Operating Modes

The fractional divider has two operating modes:

- Normal Divider Mode
- Fractional Divider Mode

Normal Divider Mode

In Normal Divider Mode (FDR.DM = 01_B), the fractional divider behaves as a reload counter (addition of +1) that generates an output clock pulse on the transition from $3FF_H$ to 000_H. FDR.RESULT represents the counter value and FDR.STEP determines the reload value.

The output frequencies in Normal Divider Mode are defined according to the following formulas:

$$f_{OUT} = \frac{f_{FPI} \times \frac{1}{n}}{2}, \text{ with } n = 1024 - STEP \quad (3.14)$$

In order to get $f_{OUT} = f_{FPI}/2$ STEP must be programmed with $3FF_H$.

Fractional Divider Mode

When the Fractional Divider Mode is selected (FDR.DM = 10_B), the output is derived from the input clock f_{FPI} by division of a fraction of n/1024 for any value of n from 0 to 1023 followed by the division of two. In general, the Fractional Divider Mode makes it possible to program the average output clock frequency with a higher accuracy than in Normal Divider Mode.

In Fractional Divider Mode, a pulse is generated depending on the result of the addition FDR.RESULT + FDR.STEP. If the addition leads to an overflow over $3FF_H$, a pulse is generated for the divider by two. Note that in Fractional Divider Mode the clock f_{OUT} can have a maximum period jitter of one f_{FPI} clock period.

The output frequencies in Fractional Divider Mode are defined according to the following formulas:

$$f_{OUT} = \frac{f_{FPI} \times \frac{n}{1024}}{2}, \text{ with } n = 0\text{-}1023 \quad (3.15)$$

External Clock Enable

When the external clock generation with the fractional divider has been disabled by software (setting FDR.DISCLK = 1), the disable state can be exited (hardware controlled) when the External Clock Enable input = 1.

Programmable Frequency Output for EXTCLK1

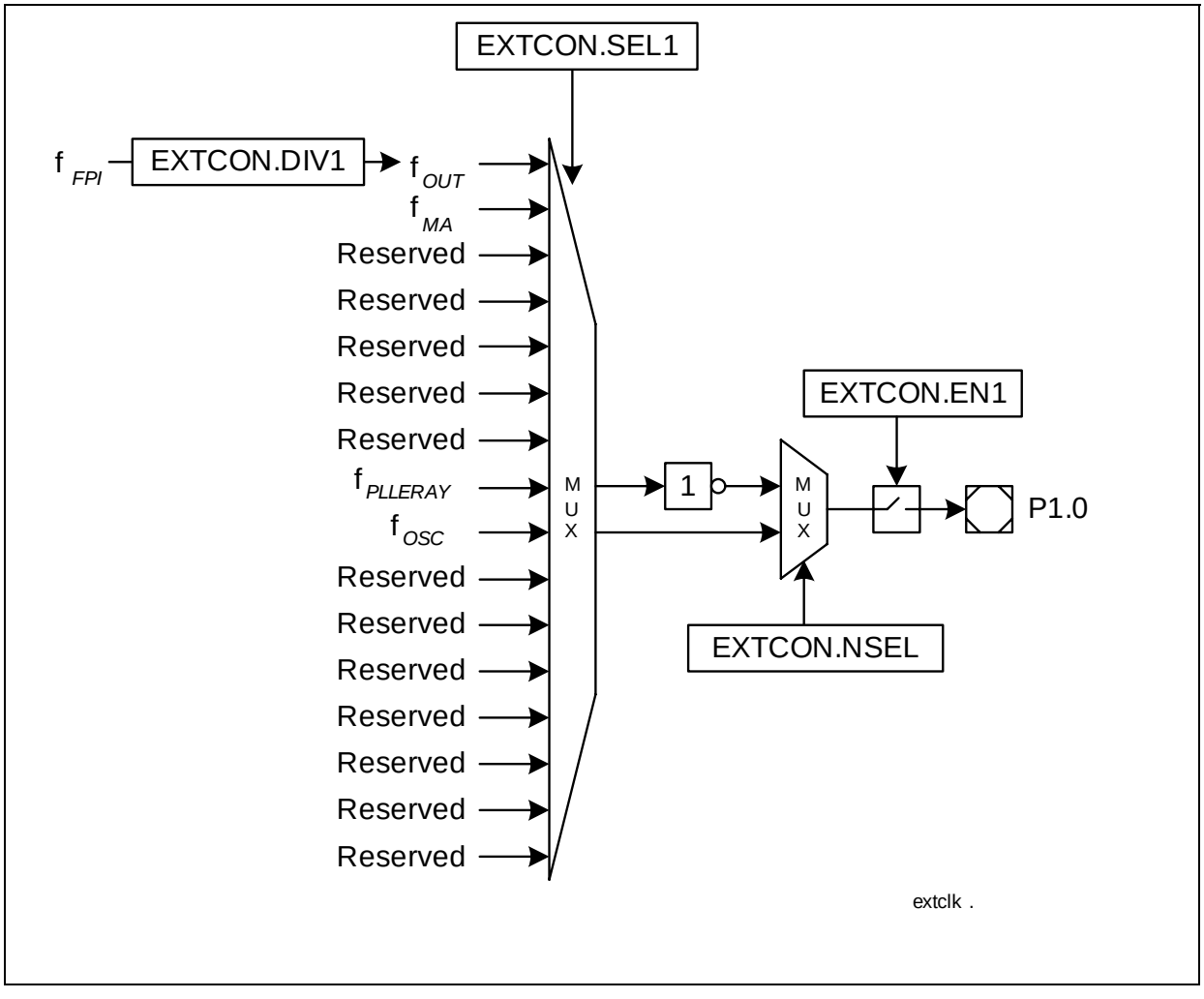


Figure 3-15 EXTCLK1 Generation

Clock f_{OUT} is generated via a counter, so the output frequency can be selected in small steps.

f_{OUT} always provides complete output periods.

Register EXTCON provides control over the output generation (frequency, activation).

3.1.1.7 CGU Registers

System Oscillator Register

This register controls the settings of OSC.

OSCCON
OSC Control Register **(010_H)** **Reset Value: 0000 001C$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0											OSCVAL				
r											rw				

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				X1DEN	X1D	0	PLLHV	0	MODE		GAINSEL		OSCRES	PLLLV	0
r				rw	rh	r	rh	rw	rw		rh		w	rh	

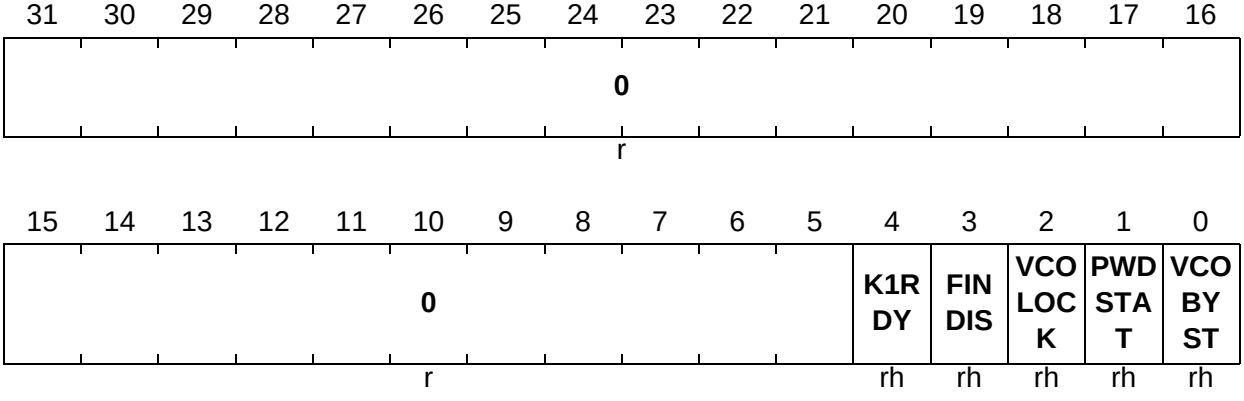
Field	Bits	Type	Description
PLLLV	1	rh	**Oscillator for PLL Valid Low Status Bit** This bit indicates if the frequency output of OSC is usable for the VCO part of the PLL. This is checked by the Oscillator Watchdog of the PLL. 0_B The OSC frequency is not usable. Frequency f_{REF} is too low. 1_B The OSC frequency is usable
OSCRES	2	w	**Oscillator Watchdog Reset** 0_B The Oscillator Watchdog of the PLL is not cleared and remains active 1_B The Oscillator Watchdog of the PLL is cleared and restarted
GAINSEL	[4:3]	rw	**Oscillator Gain Selection** 00_B Reserved, do not use this combination 01_B The gain control is configured for frequencies from 8 MHz to 16 MHz 10_B The gain control is configured for frequencies from 8 MHz to 20 MHz 11_B The gain control is configured for frequencies from 8 MHz to 25 MHz

Field	Bits	Type	Description
MODE	[6:5]	rw	**Oscillator Mode** This bit field defines which mode can be used and if the oscillator entered the Power-Saving Mode or not. 00_B External Crystal Mode and External Input Clock Mode. The oscillator Power-Saving Mode is not entered. 01_B OSC is disabled. The oscillator Power-Saving Mode is not entered. 10_B External Input Clock Mode and the oscillator Power-Saving Mode is entered 11_B OSC is disabled. The oscillator Power-Saving Mode is entered.
PLLHV	8	rh	**Oscillator for PLL Valid High Status Bit** This bit indicates if the frequency output of OSC is usable for the VCO part of the PLL. This is checked by the Oscillator Watchdog of the PLL. 0_B The OSC frequency is not usable. Frequency f_{OSC} is too high. 1_B The OSC frequency is usable
X1D	10	rh	**XTAL1 Data Value** This bit monitors the value (level) of pin XTAL1. If XTAL1 is not used as clock input it can be used as GPI pin. This bit is only updated if X1DEN is set.
X1DEN	11	rw	**XTAL1 Data Enable** 0_B Bit X1D is not updated 1_B Bit X1D can be updated
OSCVAL	[20:16]	rw	**OSC Frequency Value** This bit field defines the divider value that generates the reference clock that is supervised by the oscillator watchdog. f_{OSC} is divided by OSCVAL + 1 in order to generate f_{OSCREF}.
0	7	rw	**Reserved** Should be written with 0.
0	0,9, [15:12], [31:21]	r	**Reserved** Read as 0; should be written with 0.

PLL Registers

These registers control the setting of the PLL.

PLLSTAT
PLL Status Register **(014_H)** **Reset Value: 0000 0009$_H$**



Field	Bits	Type	Description
VCOBYST	0	rh	**VCO Bypass Status** 0_B Freerunning / Normal Mode is entered 1_B Prescaler Mode is entered
PWDSTAT	1	rh	**PLL Power-saving Mode Status** 0_B PLL Power-saving Mode was not entered 1_B PLL Power-saving Mode was entered
VCOLOCK	2	rh	**PLL VCO Lock Status** 0_B The frequency difference of f_{REF} and f_{DIV} is greater than allowed. The VCO part of the PLL can not lock on a target frequency. 1_B The frequency difference of f_{REF} and f_{DIV} is small enough to enable a stable VCO operation. *Note: In case of a loss of VCO lock the f_{VCO} goes to the upper boundary of the VCO frequency if the reference clock input is greater than expected.* *Note: In case of a loss of VCO lock the f_{VCO} goes to the lower boundary of the VCO frequency if the reference clock input is lower than expected.*

Field	Bits	Type	Description
FINDIS	3	rh	**Input Clock Disconnect Select Status** 0_B The input clock from the oscillator is connected to the VCO part 1_B The input clock from the oscillator is disconnected from the VCO part *Note: This bit can be set by setting bit PLLCON0.SETFINDIS.* *Note: This bit can be cleared by setting bit PLLCON0.CLRFINDIS.*
K1RDY	4	rh	**K1 Divider Ready Status** This bit indicates if the K1-divider operates on the configured value or not. this is of interest if the values is changed. 0_B K1-Divider is not ready to operate with the new value 1_B K1-Divider is ready to operate with the new value
0	[31:5]	r	**Reserved** Read as 0; should be written with 0.

PLLCON0
PLL Configuration 0 Register (018$_H$) Reset Value: 0001 C600$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0				**PDIV**				**0**					**RES LD**	**0**	**PLL PWD**
r				rw				r					w	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NDIV							**0**	**0**	**OSC DISC DIS**	**CLR FIN DIS**	**SET FIN DIS**	**0**		**VCO PWD**	**VCO BYP**
rw							rw	r	rw	w	w	rw		rw	rw

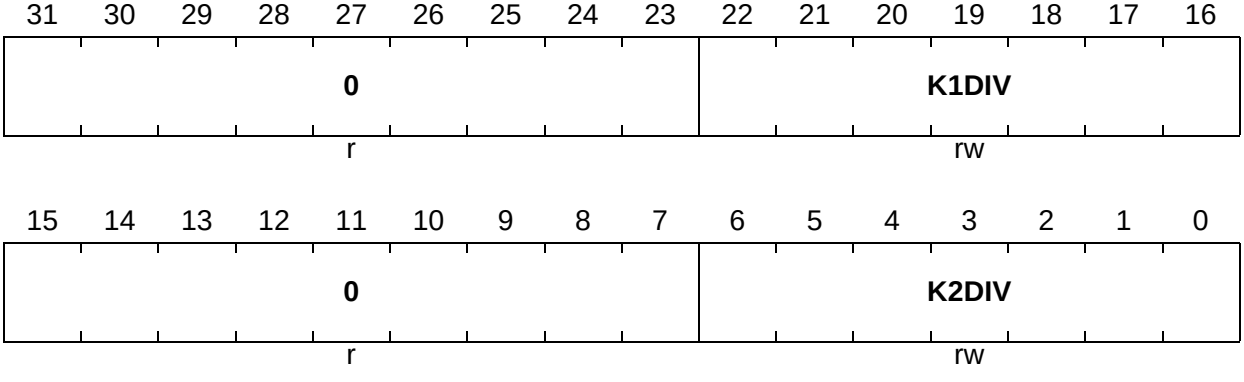
Field	Bits	Type	Description
VCOBYP	0	rw	**VCO Bypass** 0_B Normal operation, VCO is not bypassed 1_B Prescaler Mode; VCO is bypassed

Field	Bits	Type	Description
VCOPWD	1	rw	**VCO Power Saving Mode** 0_B Normal behavior 1_B The VCO is put into a Power Saving Mode and can no longer be used. Only Prescaler Mode are active if previously selected.
SETFINDIS	4	w	**Set Status Bit PLLSTAT.FINDIS** 0_B Bit PLLSTAT.FINDIS is left unchanged 1_B Bit PLLSTAT.FINDIS is set. The input clock from the oscillator is disconnected from the VCO part.
CLRFINDIS	5	w	**Clear Status Bit PLLSTAT.FINDIS** 0_B Bit PLLSTAT.FINDIS is left unchanged 1_B Bit PLLSTAT.FINDIS is cleared. The input clock from the oscillator is connected to the VCO part.
OSCDISCDIS	6	rw	**Oscillator Disconnect Disable** This bit is used to disable the control PLLSTAT.FINDIS in a PLL loss-of-lock case. 0_B In case of a PLL loss-of-lock bit PLLSTAT.FINDIS is set 1_B In case of a PLL loss-of-lock bit PLLSTAT.FINDIS is cleared
NDIV	[15:9]	rw	**N-Divider Value** The value the N-Divider operates is NDIV+1.
PLLPWD	16	rw	**PLL Power Saving Mode** 0_B The complete PLL block is put into a Power Saving Mode and can no longer be used. Only the Bypass Mode is active if previously selected. 1_B Normal behavior
RESLD	18	w	**Restart VCO Lock Detection** Setting this bit will clear bit PLLSTAT.VCOLOCK and restart the VCO lock detection. Reading this bit returns always a zero.
PDIV	[27:24]	rw	**P-Divider Value** The value the P-Divider operates is PDIV+1.
0	[3:2], 8	rw	**Reserved** Have to be written with 0.

Field	Bits	Type	Description
0	7, 17, [23:19], , [31:28]	r	**Reserved** Read as 0; should be written with 0.

PLLCON1

PLL Configuration 1 Register (01C$_H$) Reset Value: 0002 000F$_H$



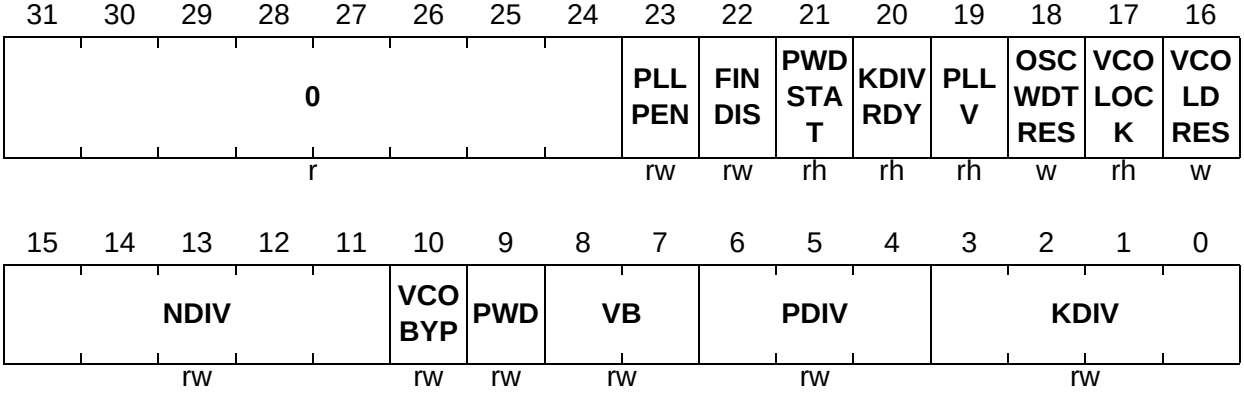
Field	Bits	Type	Description
K2DIV	[6:0]	rw	**K2-Divider Value** The value the K2-Divider operates is K2DIV+1.
K1DIV	[22:16]	rw	**K1-Divider Value** The value the K1-Divider operates is K1DIV+1.
0	[15:7], [31:23]	r	**Reserved** Read as 0; should be written with 0.

PLL_ERAY Register

This register control the setting of the PLL_ERAY.

PLLERAYCTR

PLL_ERAY Control and Status Register(020_H) **Reset Value: 0000 B80F$_H$**



Field	Bits	Type	Description
KDIV	[3:0]	rw	**K-Divider Value** The value the K-Divider operates is KDIV+1.
PDIV	[6:4]	rw	**P-Divider Value** The value the P-Divider operates is PDIV+1.
VB	[8:7]	rw	**VCO Band Select** The VCO output frequency (freerunning frequency) is selected by this bit field. 00_B VCO frequency is 400 to 500 MHz (freerunning frequency is 150 to 320 MHz) 01_B Reserved, do not use this combination 10_B Reserved, do not use this combination 11_B Reserved, do not use this combination
PWD	9	rw	**Power Saving Mode** 0_B The PLL VCO is put into a Power Saving Mode and can no longer be used. 1_B Normal behavior This bit is only valid if PLLPEN is set.
VCOBYP	10	rw	**VCO Bypass** 0_B Normal operation, VCO is not bypassed 1_B Prescaler Mode; VCO is bypassed
NDIV	[15:11]	rw	**N-Divider Value** The value the P-Divider operates is NDIV+1.

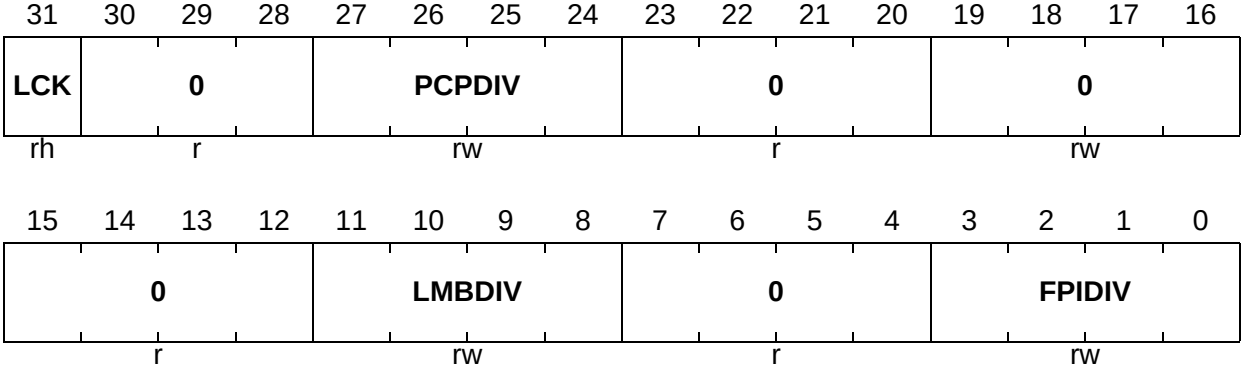
Field	Bits	Type	Description
VCOLDRES	16	w	**Restart VCO Lock Detection** Setting this bit will clear bit VCOLOCK and restart the VCO lock detection. Reading this bit returns always a zero.
VCOLOCK	17	rh	**PLL VCO Lock Status** 0_B The frequency difference of f_{REF} and f_{DIV} is greater than allowed. The VCO part of the PLL can not lock on a target frequency. 1_B The frequency difference of f_{REF} and f_{DIV} is small enough to enable a stable VCO operation. *Note: In case of a loss of VCO lock the f_{VCO} goes to the upper boundary of the selected VCO band if the reference clock input is greater than expected.* *Note: In case of a loss of VCO lock the f_{VCO} goes to the lower boundary of the selected VCO band if the reference clock input is lower than expected.*
OSCWDTRES	18	w	**Oscillator Watchdog Reset** 0_B The Oscillator Watchdog of the PLL is not cleared and remains active 1_B The Oscillator Watchdog of the PLL is cleared and restarted
PLLV	19	rh	**Oscillator for PLL Valid Status Bit** This bit indicates if the frequency output of OSC is usable for the VCO part of the PLL. This is checked by the Oscillator Watchdog of the PLL. 0_B The OSC frequency is not usable 1_B The OSC frequency is usable

Field	Bits	Type	Description
KDIVRDY	20	rh	**K-Divider Running Stable** 0_B KDIV value is currently copied to the k-divider and therefore the value of KDIV may not be altered (avoid programming of the KDIV at this point of time) 1_B k-divider is stable and may be programmed by setting a new KDIV value *Note: If reprogramming KDIV with a new value while KDIVRDY is cleared may result in an incorrect programming of the k-divider.*
PWDSTAT	21	rh	**PLL Power-saving Mode Status** 0_B PLL Power-saving Mode was not entered 1_B PLL Power-saving Mode was entered
FINDIS	22	rw	**Input Clock Disconnect Select Status** 0_B The input clock from the oscillator is connected to the VCO part 1_B The input clock from the oscillator is disconnected from the VCO part
PLLPEN	23	rw	**PLL Power Enable** 0_B The power supply for the PLL is disabled. No PLL operation is possible at all and no clock is generated. 1_B The power supply for the PLL is enabled. This bit has to be set in order to use any PLL function.
0	[31:24]	r	**Reserved** Read as 0; should be written with 0.

CCU Control Registers

CCUCON0

CCU Clock Control Register 0 (030_H) Reset Value: $8000\ 0001_H$



Field	Bits	Type	Function
FPIDIV	[3:0]	rw	**FPI-Bus Divider Reload Value** 0000_B $f_{FPI} = f_{PLL}$ 0001_B $f_{FPI} = f_{PLL}/2$ 0010_B $f_{FPI} = f_{PLL}/3$ 0011_B $f_{FPI} = f_{PLL}/4$ 0100_B $f_{FPI} = f_{PLL}/5$ 0101_B $f_{FPI} = f_{PLL}/6$ 0110_B $f_{FPI} = f_{PLL}/7$ 0111_B $f_{FPI} = f_{PLL}/8$ 1000_B $f_{FPI} = f_{PLL}/9$ 1001_B $f_{FPI} = f_{PLL}/10$ 1010_B $f_{FPI} = f_{PLL}/11$ 1011_B $f_{FPI} = f_{PLL}/12$ 1100_B $f_{FPI} = f_{PLL}/13$ 1101_B $f_{FPI} = f_{PLL}/14$ 1110_B $f_{FPI} = f_{PLL}/15$ 1111_B $f_{FPI} = f_{PLL}/16$

Field	Bits	Type	Function
LMBDIV	[11:8]	rw	**LMB-Bus Divider Reload Value** 0000_B $f_{LMB} = f_{PLL}$ 0001_B $f_{LMB} = f_{PLL}/2$ 0010_B $f_{LMB} = f_{PLL}/3$ 0011_B $f_{LMB} = f_{PLL}/4$ 0100_B $f_{LMB} = f_{PLL}/5$ 0101_B $f_{LMB} = f_{PLL}/6$ 0110_B $f_{LMB} = f_{PLL}/7$ 0111_B $f_{LMB} = f_{PLL}/8$ 1000_B $f_{LMB} = f_{PLL}/9$ 1001_B $f_{LMB} = f_{PLL}/10$ 1010_B $f_{LMB} = f_{PLL}/11$ 1011_B $f_{LMB} = f_{PLL}/12$ 1100_B $f_{LMB} = f_{PLL}/13$ 1101_B $f_{LMB} = f_{PLL}/14$ 1110_B $f_{LMB} = f_{PLL}/15$ 1111_B $f_{LMB} = f_{PLL}/16$
PCPDIV	[27:24]	rw	**PCP Divider Reload Value** 0000_B $f_{PCP} = f_{PLL}$ 0001_B $f_{PCP} = f_{PLL}/2$ 0010_B $f_{PCP} = f_{PLL}/3$ 0011_B $f_{PCP} = f_{PLL}/4$ 0100_B $f_{PCP} = f_{PLL}/5$ 0101_B $f_{PCP} = f_{PLL}/6$ 0110_B $f_{PCP} = f_{PLL}/7$ 0111_B $f_{PCP} = f_{PLL}/8$ 1000_B $f_{PCP} = f_{PLL}/9$ 1001_B $f_{PCP} = f_{PLL}/10$ 1010_B $f_{PCP} = f_{PLL}/11$ 1011_B $f_{PCP} = f_{PLL}/12$ 1100_B $f_{PCP} = f_{PLL}/13$ 1101_B $f_{PCP} = f_{PLL}/14$ 1110_B $f_{PCP} = f_{PLL}/15$ 1111_B $f_{PCP} = f_{PLL}/16$
LCK	31	rh	**Lock Status** This bit indicates if the register can be updated with a new value or if the register is locked and a write action from the bus side has no effect. 0_B The register is unlocked and can be updated 1_B The register is locked and can not be updated

Field	Bits	Type	Function
0	[19:16]	rw	**Reserved** Should be written with 0.
0	[7:4], [15:12], [23:20], [30:28]	r	**Reserved** Read as 0; should be written with 0.

CCUCON1
CCU Clock Control Register 1

(034_H) **Reset Value: 8000 0B01$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LCK	**0**														
rh	r														

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				**REFCLKDIV**				**0**				**MCDSDIV**			
r				r				r				rw			

Field	Bits	Type	Function
MCDSDIV	[3:0]	rw	**MCDS Divider Reload Value** 0000_B $f_{MCDS} = f_{PLL}$ 0001_B $f_{MCDS} = f_{PLL}/2$ 0010_B $f_{MCDS} = f_{PLL}/3$ 0011_B $f_{MCDS} = f_{PLL}/4$ 0100_B $f_{MCDS} = f_{PLL}/5$ 0101_B $f_{MCDS} = f_{PLL}/6$ 0110_B $f_{MCDS} = f_{PLL}/7$ 0111_B $f_{MCDS} = f_{PLL}/8$ 1000_B $f_{MCDS} = f_{PLL}/9$ 1001_B $f_{MCDS} = f_{PLL}/10$ 1010_B $f_{MCDS} = f_{PLL}/11$ 1011_B $f_{MCDS} = f_{PLL}/12$ 1100_B $f_{MCDS} = f_{PLL}/13$ 1101_B $f_{MCDS} = f_{PLL}/14$ 1110_B $f_{MCDS} = f_{PLL}/15$ 1111_B $f_{MCDS} = f_{PLL}/16$
REFCLKDIV	[11:8]	r	**Reference Clock for MCDS Divider Reload Value** 0000_B $f_{REFCLK2} = f_{PLL\_ERAY}/2$ and $f_{REFCLK1} = f_{PLL}/2$ 0001_B $f_{REFCLK2} = f_{PLL\_ERAY}/4$ and $f_{REFCLK1} = f_{PLL}/4$ 0010_B $f_{REFCLK2} = f_{PLL\_ERAY}/6$ and $f_{REFCLK1} = f_{PLL}/6$ 0011_B $f_{REFCLK2} = f_{PLL\_ERAY}/8$ and $f_{REFCLK1} = f_{PLL}/8$ 0100_B $f_{REFCLK2} = f_{PLL\_ERAY}/10$ and $f_{REFCLK1} = f_{PLL}/10$ 0101_B $f_{REFCLK2} = f_{PLL\_ERAY}/12$ and $f_{REFCLK1} = f_{PLL}/12$ 0110_B $f_{REFCLK2} = f_{PLL\_ERAY}/14$ and $f_{REFCLK1} = f_{PLL}/14$ 0111_B $f_{REFCLK2} = f_{PLL\_ERAY}/16$ and $f_{REFCLK1} = f_{PLL}/16$ 1000_B $f_{REFCLK2} = f_{PLL\_ERAY}/18$ and $f_{REFCLK1} = f_{PLL}/18$ 1001_B $f_{REFCLK2} = f_{PLL\_ERAY}/20$ and $f_{REFCLK1} = f_{PLL}/20$ 1010_B $f_{REFCLK2} = f_{PLL\_ERAY}/22$ and $f_{REFCLK1} = f_{PLL}/22$ 1011_B $f_{REFCLK2} = f_{PLL\_ERAY}/24$ and $f_{REFCLK1} = f_{PLL}/24$ 1100_B $f_{REFCLK2} = f_{PLL\_ERAY}/26$ and $f_{REFCLK1} = f_{PLL}/26$ 1101_B $f_{REFCLK2} = f_{PLL\_ERAY}/28$ and $f_{REFCLK1} = f_{PLL}/28$ 1110_B $f_{REFCLK2} = f_{PLL\_ERAY}/30$ and $f_{REFCLK1} = f_{PLL}/30$ 1111_B $f_{REFCLK2} = f_{PLL\_ERAY}/32$ and $f_{REFCLK1} = f_{PLL}/32$

Field	Bits	Type	Function
LCK	31	rh	**Lock Status** This bit indicates if the register can be updated with a new value or if the register is locked and a write action from the bus side has no effect. 0_B The register is unlocked and can be updated 1_B The register is locked and can not be updated
0	[7:4], [30:12]	r	**Reserved** Read as 0; should be written with 0.

Clock Output Control Register

This register controls the setting of external clock for pin 1.0 and 1.12 (EXTCLK0 and EXTCLK1).

EXTCON
External Clock Control Register ($03C_H$) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DIV1								0		SEL1				NSEL	EN1
rw								r		rw				rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0									GPTAINSEL	SEL0				0	EN0
r									rw	rw				r	rw

Field	Bits	Type	Description
EN0	0	rw	**External Clock Enable for EXTCLK0** 0_B No external clock is provided 1_B The configured external clock is provided

Field	Bits	Type	Description
SEL0	[5:2]	rw	**External Clock Select for EXTCLK0** This bit field defines the clock source that is selected as output for pin EXTCLK0. 0000_B f_{OUT} is selected for the external clock 0001_B f_{PLL} is selected for the external clock 0010_B Reserved, do not use this combination ... 0110_B Reserved, do not use this combination 0111_B f_{PLL_ERAY} is selected for the external clock signal 1000_B f_{OSC} is selected for the external clock signal 1001_B Reserved, do not use this combination ... 1110_B Reserved, do not use this combination 1111_B f_{MT} from the ERAY module is selected for the external clock
GPTAINSEL	6	rw	**GPTA Input Select** This value defines if either the input from P2.8 or the configured output frequency for EXTCLK0 is used as input for IN0 of the GPTA module. 0_B P2.8 is selected as input for IN0 of the GPTA 1_B EXTCLK0 output is selected as input for IN0 of the GPTA
EN1	16	rw	**External Clock Enable for EXTCLK1** 0_B No external clock is provided 1_B The configured external clock is provided
NSEL	17	rw	**Negation Selection** 0_B The external clock is inverted 1_B The external clock is not inverted

Field	Bits	Type	Description
SEL1	[21:18]	rw	**External Clock Select for EXTCLK1** This bit field defines the clock source that is selected as output for pin EXTCLK1. 0000_B f_{OUT} is selected for the external clock 0001_B f_{PLL} is selected for the external clock 0010_B Reserved, do not use this combination ... 0110_B Reserved, do not use this combination 0111_B f_{PLL_ERAY} is selected for the external clock signal 1000_B f_{OSC} is selected for the external clock signal 1001_B Reserved, do not use this combination ... 1111_B Reserved, do not use this combination
DIV1	[31:24]	rw	**External Clock Divider for EXTCLK1** This value defines the reload value of the divider that generates f_{OUT} out of f_{FPI} ($f_{OUT} = f_{FPI}/(DIV1+1)$). The divider itself is cleared each time bit EN1 is cleared.
0	1, [15:7], [23:22]	r	**Reserved** Read as 0; should be written with 0.

FDR
Fractional Divider Register **(038_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DIS CLK			**0**			**RESULT**									
rwh			r			rh									

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DM		**0**				**STEP**									
rw		r				rw									

Field	Bits	Type	Description
STEP	[9:0]	rw	**Step Value** In Normal Divider Mode, STEP contains the reload value for RESULT. In Fractional Divider Mode, this bit field determines the 10-bit value that is added to RESULT with each input clock cycle.
DM	[15:14]	rw	**Divider Mode** This bit fields determines the functionality of the fractional divider block. 00_B Fractional divider is switched off; no output clock is generated. The Reset External Divider signal is 1. RESULT is not updated (default after System Reset). 01_B Normal Divider Mode selected. 10_B Fractional Divider Mode selected. 11_B Fractional divider is switched off; no output clock is generated. RESULT is not updated.
RESULT	[25:16]	rh	**Result Value** In Normal Divider Mode, RESULT acts as reload counter (addition +1). In Fractional Divider Mode, this bit field contains the result of the addition RESULT + STEP. If DM is written with 01_B or 10_B, RESULT is loaded with $3FF_H$.
DISCLK	31	rwh	**Disable Clock** 0_B Clock generation of f_{OUT} is enabled according to the setting of bit field DM. 1_B Fractional divider is stopped. No change except when writing bit field DM. This bit is cleared when external clock enable input is asserted. In case of a conflict between hardware clear and software set of DISCLK, the software set wins. Any write or read-modify-write action leads to the described behavior. As a result, read-modify-write operations should be avoided.
0	[13:10], [30:26]	r	**Reserved** Read as 0; should be written with 0.

3.1.2 Module Clock Generation

The TC1797 on-chip modules have two registers for clock control:

- Clock Control Register CLC
- Fractional Divider Register FDR

The following sections describes the general functionality of CLC and FDR. The module-specific implementation details are described in the corresponding module chapters.

3.1.2.1 Clock Control Register CLC

All CLC registers have basically the same bit and bit field layout. However, not all CLC register functions are implemented for each peripheral module. **Table 3-1** defines in detail which bits and bit fields of the CLC registers are implemented for each clock control register.

The CLC register controls the generation of the peripheral module clock which is derived from the system clock. The following functions for the module are associated with the CLC register:

- Peripheral clock static on/off control
- Module clock behavior in Sleep Mode
- Operation during Suspend Mode
- Fast Shut-off Mode control

Module Enable/Disable Control

If a module is not used at all by an application, it can be completely shut off by setting bit DISR in its CLC register. For peripheral modules with a run mode clock divider field RMC, a second option to completely switch off the module is to set bit field RMC to 00_H. This also disables the module's operation.

The status bit DISS always indicates whether a module is currently switched off (DISS = 1) or switched on (DISS = 0).

Write operations to the non CLC registers of disabled modules are not allowed. However, the CLC of a disabled module can be written. An attempt to write to any of the other writable registers of a disabled module except CLC will cause the corresponding Bus Control Unit (BCU) to generate a bus error.

A read operation of registers of a disabled module is allowed and does not generate a bus error.

When a disabled module is switched on by writing an appropriate value to its MOD_CLC register (DISR = 0 and RMC (if implemented) > 0), status bit DISS changes from 1 to 0. During the phase in which the module becomes active, any write access to corresponding module registers (when DISS is still set) will generate a bus error. Therefore, when enabling a disabled module, application software should check after activation of the module once (read back of the CLC register) to find out whether DISS is already cleared, before a module register (including the CLC register) will be written to.

Sleep Mode Control

The EDIS bit in the CLC register controls whether or not a module is stopped during Sleep Mode. If EDIS is 0, a Sleep Mode request can be recognized by the module and, when received, its clock is shut off.

If EDIS is set to 1, a Sleep Mode request is disregarded by the module and the module continues its operation.

Suspend Mode Control

During emulation and debugging of TC1797 applications, the execution of an application program can be suspended. When an application is suspended, normal operation of the application's program is halted, and the TC1797 begins (or resumes) executing a special debug monitor program. If bit SPEN is set, the operation of the peripheral module is stopped when the Suspend Mode request is generated. If SPEN is cleared, the module does not react to the Suspend Mode request and continues its normal operation. This feature allows each peripheral module to be adapted to the unique requirements of the application being debugged. Setting SPEN bits is usually performed by a debugger.

This feature is necessary because application requirements typically determine whether on-chip modules should be stopped or left running when an application is suspended for debugging. For example, a peripheral unit that is controlling the motion of an external device through motors in most cases must not be stopped so as to prevent damage of the external device due to the loss of control through the peripheral. On the other hand, it makes sense to stop the system timer while the debugger is actively controlling the chip because it should only count the time when the user's application is running.

Note that it is never appropriate for application software to set the SPEN bit. The Suspend Mode should only be set by a debug software. To guard against application software accidently setting SPEN, bit SPEN is specially protected by the mask bit SBWE. The SPEN bit can only be written if, during the same write operation, SBWE is set, too. Application software should never set SBWE. In this way, user software can not accidentally alter the value of the SPEN bit that has been set by a debugger.

Entering Disabled Mode

Software can request that a peripheral unit be put into Disabled Mode by setting DISR. A module will also be put into Disabled Mode if the Sleep Mode is requested and the module is configured to allow Sleep Mode.

In Secure Shut-off Mode, a module first finishes any operation in progress, then proceeds with an orderly shut down. When all sub-components of the module are ready to be shut down, the module signals its clock control unit, that turns off the clock to this peripheral unit, that it is now ready for shut down. The status bit DISS is updated by the peripheral unit accordingly.

During emulation and debugging, it may be necessary to monitor the instantaneous state of the machine – including all or most of its modules – at the moment a software breakpoint is reached. In such cases, it may not be desired that the kernel of a module finish whatever transaction is in progress before stopping, because that might cause important states in this module to be lost. Fast Shut-off Mode, controlled by bit FSOE, is available for this situation.

If FSOE = 0, modules are stopped as described above. This is called Secure Shut-off Mode. The module is allowed to finish whatever operation is in progress. If Fast Shut-off Mode is selected (FSOE = 1), clock generation to the unit is stopped as soon as any outstanding bus operation is finished. The clock control unit does not wait until the module has finished its transaction. This option stops the unit's clock as fast as possible, and the state of the unit will be the closest possible to the time of the occurrence of the software breakpoint.

Note: In all TC1797 modules except MultiCAN and DMA, the only shut down operating mode that is available is the Fast Shut-off ModeTC1797, regardless of the state of the FSOE bit.

Whether Secure Shut-off Mode or Fast Shut-off Mode is required depends on the application, the needs of the debugger, and the type of unit. For example, the analog-to-digital converter might allow the converter to finish a running analog conversion before it can be suspended. Otherwise the conversion might be corrupted and a wrong value could be produced when Suspend Mode is exited and the unit is enabled again. This would affect further emulation and debugging of the application's program.

On the other hand, if a problem is observed to relate to the operation of the external analog-to-digital converter itself, it might be necessary to stop the unit as fast as possible in order to monitor its current instantaneous state. To do this, the Fast Shut-off Mode option would be selected. Although proper continuation of the application's program might not be possible after such a step, this would most likely not matter in such a case.

Note that it is never appropriate for application software to set the FSOE bit. Fast Shut-off Mode should only be set by debug software. To guard against application software accidently setting FSOE, bit FSOE is specially protected by the mask bit SBWE. The SPEN bit can only be written if, during the same write operation, SBWE is set, too. Application software should never set SBWE. In this way, user software can not accidentally alter the value of the FSOE bit. Note that this is the same guard mechanism used for the SPEN bit.

Module Clock Divider Control

Peripheral modules of the TC1797 can have a RMC control bit field in their CLC registers. This Run Mode Clock control bit field makes it possible to slow down the CLC clock via a programmable clock divider circuit.

A value of 00_H in RMC disables the clock signals to these modules (CLC clock is switched off). If RMC is not equal to 00_H, the clock for a module register (f_{CLC}) accesses is generated as

(3.16)

$$f_{CLC} = \frac{f_{FPI}}{RMC}$$

where RMC is the content of its CLC register RMC field with a range of 1 to 255. If RMC is not available in a CLC register, the CLC clock frequency f_{CLC} is always equal to the frequency of f_{FPI}.

Note: The number of module clock cycles (wait states) that are required for a "destructive read" access (means: flags/bits are set/cleared by a read access) to a module register of a peripheral unit depends on the selected CLC clock frequency.
Therefore, a slower CLC clock (selected via bit field RMC in the CLC register) may result in a longer read cycle access time on the FPI-Bus for peripheral units with "destructive read" access (e.g. the ASC).

Module Clock Register Implementations

Table 3-1 shows which of the CLC register bits/bit fields are implemented for each peripheral module in the TC1797 and which modules are equipped with a fractional divider.

Table 3-1 Clock Generation Implementation of the TC1797 Peripheral Modules

Module	DISR Bit 0	DISS Bit 1	SPEN Bit 2	EDIS Bit 3	SBWE Bit 4	FSOE Bit 5	RMC	Fract. Divider 1)
ADC0 ADC1 ADC2	✔	✔	✔	✔	✔	✔	–	–
FADC	✔	✔	✔	✔	✔	✔	–	✔
ASC0 and ASC1	✔	✔	✔	✔	✔	✔	8-bit	–
SSC0	✔	✔	✔	✔	✔	✔	–	✔
SSC1	✔	✔	✔	✔	✔	✔	–	✔
MultiCAN	✔	✔	✔	✔	✔	✔	–	✔
DMA	✔	✔	✔	✔	✔	✔	–	–
EBU	✔	✔	–	–	–	–	–	–
GPTA0 GPTA1 LTCA2	✔	✔	✔	✔	✔	✔	–	✔
MLI0	not implemented, MLI is connected to DMA_CLC							✔
MLI1	not implemented, MLI is connected to DMA_CLC							✔
MSC0	✔	✔	✔	✔	✔	✔	–	✔

Table 3-1 Clock Generation Implementation of the TC1797 Peripheral Modules

Module	DISR Bit 0	DISS Bit 1	SPEN Bit 2	EDIS Bit 3	SBWE Bit 4	FSOE Bit 5	RMC	Fract. Divider 1)
MSC1	✔	✔	✔	✔	✔	✔	–	✔
PCP2	different bit definitions 2)							
STM	✔	✔	✔	✔	✔	✔	3-bit	–
ERAY	✔	✔	✔	✔	✔	✔	3-bit	–

1) Further info on FDR implementations see **Table 3-3**.

2) Automatic clock switch-off capability if PCP is idle.

Fractional Divider Operation

The fractional divider divides the input clock f_{CLC} either by the factor 1/n or by a fraction of n/1024 for any value of n from 0 to 1023, and outputs the clock signal, f_{MOD}. The fractional divider is controlled by the MOD_FDR register.

The fractional divider can be configured for two operating modes:

- **Normal Divider Mode:** Reload counter (RESULT = RESULT + 1), generating an output clock pulse on counter overflow.
- **Fractional Divider Mode:** Add a STEP value to the RESULT value and generates an output clock pulse on counter overflow.

Normal Divider Mode

In Normal Divider Mode (MOD_FDR.DM = 01_B), the fractional divider behaves as a reload counter (addition of +1) that generates an output clock pulse at f_{MOD} on the transition from $3FF_H$ to 000_H. MOD_FDR.RESULT represents the counter value and MOD_FDR.STEP determines the reload value.

The output frequencies in Normal Divider Mode are defined according to the following formulas:

$$f_{MOD} = f_{CLC} \times \frac{1}{n} \quad \text{, with n = 1024 - STEP} \qquad (3.17)$$

Note: Each write to register FDR with bit field DM = 01_B or 10_B sets bit field RESULT to $3FF_H$.

In order to get $f_{MOD} = f_{CLC}$ MOD_FDR.STEP must be programmed with $3FF_H$. **Figure 3-16** shows the operation of the Normal Divider Mode with a reload value of MOD_FDR.STEP = $3FD_H$.

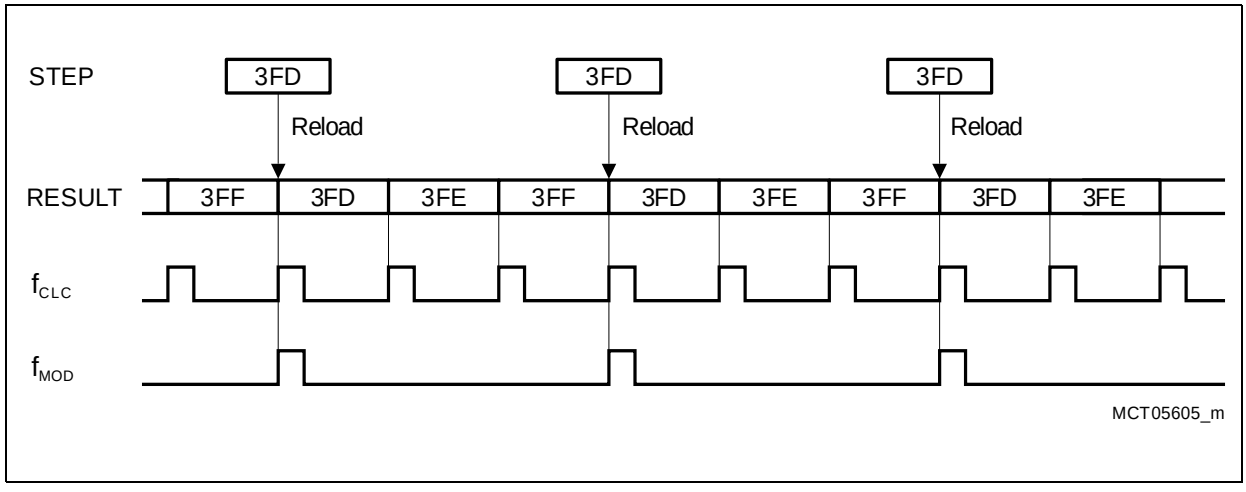


Figure 3-16 Normal Divider Mode

Fractional divider Mode

When the Fractional Divider Mode is selected (MOD_FDR.DM = 10_B), the module clock f_{MOD} is derived from the bus clock f_{CLC} by division of a fraction of n/1024 for any value of n from 0 to 1023. In general, the Fractional Divider Mode makes it possible to program the average module clock frequency with a higher accuracy than in Normal Divider Mode.

In Fractional Divider Mode, an clock pulse at f_{MOD} is generated depending on the result of the addition MOD_FDR.RESULT + MOD_FDR.STEP. If the addition leads to an overflow over $3FF_H$, a pulse is generated at f_{MOD}. Note that in Fractional Divider Mode the clock f_{MOD} can have a maximum period jitter of one f_{CLC} clock period.

The frequencies in Fractional Divider Mode are defined according to the following formula:

$$f_{MOD} = f_{CLC} \times \frac{n}{1024} \quad \text{, with n = 0-1023} \tag{3.18}$$

Note: Each write to register FDR with bit field DM = 01_B or 10_B sets bit field RESULT to $3FF_H$.

Suspend Mode

The operation of the fractional divider can be controlled by the Suspend Mode Request input. This input is activated in Suspend Mode by the on-chip debug control logic. In Suspend Mode, module registers are accessible for read and write actions, but other module internal functions are frozen.

The state of the Suspend Mode Request and Suspend Mode Acknowledge is latched in two status flags of register MOD_FDR, SUSREQ and SUSACK. Suspend Mode Request

and (Suspend Mode Acknowledge or bit SM) must remain set both to maintain the Suspend Mode.

The Kernel Disable Request becomes always active when the Module Disable Request signal is activated, independently of the Suspend Mode settings in the fractional divider.

External Clock Enable

When the module clock generation has been disabled by software (setting MOD_FDR.DISCLK = 1), the disable state can be exited when the External Clock Enable input = 1. This feature is enabled when MOD_FDR.ENHW = 1.

Table 3-2 Fractional Divider Function Table

Mode	SC	DM	Result	f_{MOD}	Operation of Fractional Divider
Normal Mode	–	00_B	unchanged	inactive	switched off
		01_B	continuously updated	active	Normal Divider Mode
		10_B			Fractional Divider Mode
		11_B	unchanged	inactive	switched off
Suspend Mode	00_B	00_B	unchanged	inactive	switched off
		01_B	continuously updated	active	Normal Divider Mode
		10_B			Fractional Divider Mode
		11_B	unchanged	inactive	switched off
	01_B	00_B	unchanged		switched off
		01_B	loaded with $3FF_H$		halted
		10_B			
		11_B	unchanged	inactive	switched off
	10_B	00_B	loaded with $3FF_H$	inactive	switched off
		01_B			halted
		10_B			
		11_B			switched off
	11_B	–	loaded with $3FF_H$	inactive	switched off

Fractional Divider Register Implementations

Table 3-3 shows the implementations specific differences of the fractional divider functionality.

Table 3-3 FDR Register Implementations

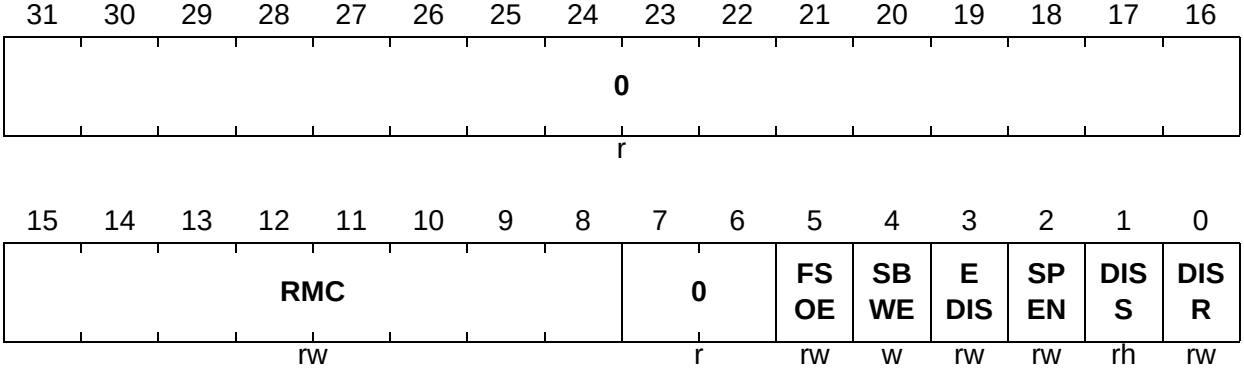
FDR Register	Suspend Mode Acknowledge Operation[1)]	ENHW[2)]
CAN_FDR	Acknowledge depends on module state	–
FADC_FDR	Acknowledge depends on module state	–
GPTA0_FDR	Always immediately acknowledged; independently from module states	from MultiCAN
MLI0_FDR	Acknowledge depends on module state	–
MLI1_FDR	Acknowledge depends on module state	–
MSC0_FDR	Acknowledge depends on module state	from MultiCAN
MSC1_FDR	Acknowledge depends on module state	from MultiCAN
SSC0_FDR	Always immediately acknowledged; independently from module state	from MultiCAN
SSC1_FDR	Always immediately acknowledged; independently from module state	from MultiCAN

1) This column shows whether a suspend acknowledge from a FDR controlled module depends on the module's state or not. If a suspend acknowledge depends from the module state, typically module operations such as serial transmissions are terminated before a suspend request is acknowledged back to the fractional divider. Note that bit FDR.SM must be cleared (granted suspend mode selected) when using the suspend mode acknowledge/grant functionality. If immediate suspend mode is selected (FDR.SM = 1), Suspend Mode is entered at once (if FDR.SC not equal 00_B) independently from the suspend acknowledge answer from the module.

2) This column shows whether the External Clock Enable input of a fractional divider is controlled by on-chip hardware (source module see comment) or not ("–").

Module CLC Register

MOD_CLC
Clock Control Register **(00_H)** **Reset Value: Module-specific**



Field	Bits	Type	Description
DISR	0	rw	**Module Disable Request Bit** Used for enable/disable control of the module. 0_B Module disable is not requested 1_B Module disable is requested
DISS	1	rh	**Module Disable Status Bit** Bit indicates the current status of the module 0_B Module is enabled 1_B Module is disabled If the RMC field is implemented and if it is 0, DISS is set automatically.
SPEN	2	rw	**Module Suspend Enable** Used for enabling the Suspend Mode. 0_B Module cannot be suspended (suspend is disabled) 1_B Module can be suspended (suspend is enabled) This bit can be written only if SBWE is set during the same write operation.
EDIS	3	rw	**Sleep Mode Enable Control** Used for module Sleep Mode control. 0_B Sleep Mode request is regarded. Module is enabled to go into Sleep Mode. 1_B Sleep Mode request is disregarded: Sleep Mode cannot be entered on a request.

Field	Bits	Type	Description
SBWE	4	w	**Module Suspend Bit Write Enable for OCDS** Determines whether SPEN and FSOE are write-protected. 0_B Bits SPEN and FSOE are write-protected 1_B Bits SPEN and FSOE are overwritten by respective value of SPEN or FSOE Reading this bit returns always 0.
FSOE	5	rw	**Fast Switch Off Enable** Used for fast clock switch-off in Suspend Mode. 0_B Clock switch-off in Suspend Mode via Disable Control Feature (Secure Clock Switch Off) selected 1_B Fast clock switch off in Suspend Mode selected This bit can be written only if SBWE is set during the same write operation.
RMC	[15:8]	rw	**8-Bit Clock Divider Value in RUN Mode** This is a maximum 8-bit divider value for clock f_{FPI}. If RMC is set to 0 the module is disabled.
0	7, 6, [31:16]	r	**Reserved** Read as 0; should be written with 0.

MOD_FDR
Fractional Divider Register **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DISCLK	ENHW	SUSREQ	SUSACK	0		RESULT									
rwh	rw	rh	rh	r		rh									

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DM		SC		SM	0	STEP									
rw		rw		rw	r	rw									

Field	Bits	Type	Description
STEP	[9:0]	rw	**Step Value** In Normal Divider Mode, STEP contains the reload value for RESULT. In Fractional Divider Mode, this bit field determines the 10-bit value that is added to RESULT with each input clock cycle.
SM	11	rw	**Suspend Mode** SM selects between granted or immediate Suspend Mode. 0_B Granted Suspend Mode selected 1_B Immediate Suspend Mode selected
SC	[13:12]	rw	**Suspend Control** This bit field determines the behavior of the fractional divider in Suspend Mode (bit SUSREQ and SUSACK set). 00_B Clock generation continues. 01_B Clock generation is stopped and the clock output signal is not generated. RESULT is not changed except when writing bit field DM with 01_B or 10_B. 10_B Clock generation is stopped and the clock output signal is not generated. RESULT is loaded with $3FF_H$. 11_B Same as SC = 10_B but signal Reset External Divider is 1 (independently of bit field DM).
DM	[15:14]	rw	**Divider Mode** This bit fields determines the functionality of the fractional divider block. 00_B Fractional divider is switched off; no output clock is generated. The Reset External Divider signal is 1. RESULT is not updated. 01_B Normal Divider Mode selected. 10_B Fractional Divider Mode selected. 11_B Fractional divider is switched off; no output clock is generated. RESULT is not updated.

Field	Bits	Type	Description
RESULT	[25:16]	rh	**Result Value** In Normal Divider Mode, RESULT acts as reload counter (addition +1). In Fractional Divider Mode, this bit field contains the result of the addition RESULT + STEP. If DM is written with 01_B or 10_B, RESULT is loaded with $3FF_H$.
SUSACK	28	rh	**Suspend Mode Acknowledge** 0_B Suspend Mode is not acknowledged. 1_B Suspend Mode is acknowledged. Suspend Mode is entered when SUSACK and SUSREQ are set.
SUSREQ	29	rh	**Suspend Mode Request** 0_B Suspend Mode is not requested. 1_B Suspend Mode is requested. Suspend Mode is entered when SUSREQ and SUSACK are set.
ENHW	30	rw	**Enable Hardware Clock Control** 0_B Bit DISCLK cannot be cleared by a high level of the External Clock Enable input 1_B Bit DISCLK is cleared while the External Clock Enable input is at high level.
DISCLK	31	rwh	**Disable Clock** 0_B Clock generation of f_{MOD} is enabled according to the setting of bit field DM. 1_B Fractional divider is stopped. Signal f_{MOD} becomes inactive. No change except when writing bit field DM.
0	10, [27:26]	r	**Reserved** Read as 0; should be written with 0.

3.2 Reset Operation

This section describes the conditions under which the TC1797 will be reset and the reset operation configuration and control.

3.2.1 Overview

The following reset request triggers are available:

- 1 External power-on hardware reset request trigger; $\overline{\text{PORST}}$, (cold reset)
- 2 External System Request reset triggers; $\overline{\text{ESR0}}$, and $\overline{\text{ESR1}}$,(warm reset)
- Watchdog Timer (WDT) reset request trigger, (warm reset)
- Software reset (SW), (warm reset)
- Debug (OCDS) reset request trigger, (warm reset)
- Resets via the JTAG interface
- Flash Tuning Protection (TP) (warm reset)
- JTAG reset (special reset)

Note: The JTAG and OCDS resets are described in the OCDS chapter.

There are two basic types of reset request triggers:

- Trigger sources that do not depend on a clock, such as the $\overline{\text{PORST}}$. This trigger force the device into an asynchronous reset assertion independently of any clock. The activation of an asynchronous reset is asynchronous to the system clock, whereas its de-assertion is synchronized.
- Trigger sources that need a clock in order to be asserted, such as the input signals $\overline{\text{ESR0}}$,$\overline{\text{ESR1}}$, the WDT trigger, TP trigger, or the SW trigger.

Note: A $\overline{\text{PORST}}$ reset should only triggered for power related issue and not for pure functional purpose.

3.2.2 Reset Types

The following summary shows the different reset types.

- Power-on Reset:
  This reset leads to a initialization into a defined state of the complete system. This reset should only be requested on a real power-on event and can not by any non power related event.
  A Power-on Reset also generates a System Reset and an Application Reset.
- System Reset:
  This reset leads to a initialization into a defined state of the complete system without the following parts: reset control, power control.
  A System Reset also generates an Application Reset.
- Debug Reset:
  This reset leads to a initialization into a defined state of the complete debug system.

- Application Reset:
  This reset leads to a initialization into a defined state of the complete application system with the following parts: all peripherals, the CPU and partially the SCU and the flash memory.

3.2.3 Reset Sources Overview

The connection of the reset sources and the activated reset signals/domains are shown in **Table 3-4**.

Table 3-4 Effects of Resets for Reset Signal Activation

Reset Request Trigger	Application Reset	Debug Reset	System Reset
PORST	Activated	Activated	Activated
ESR0	Configurable	Not Activated	Configurable
ESR1	Configurable	Not Activated	Configurable
WDT	Configurable	Not Activated	Configurable
SW	Configurable	Not Activated	Configurable
OCDS Reset	Not Activated	Activated	Not Activated
CBS_OJCONF.RSTCL0	Activated	Not Activated	Activated
CBS_OJCONF.RSTCL1	Not Activated	Activated	Not Activated
CBS_OJCONF.RSTCL3	Activated	Not Activated	Not Activated

3.2.4 Module Reset Behavior

Table 3-5 lists how the various functions of the TC1797 are affected through a reset depending on the reset type. A “X” means that this block has at least some register/bits that are affected by this reset.

Table 3-5 Effect of Reset on Device Functions

Module / Function	Application Reset	Debug Reset	System Reset	Power-on Reset
CPU Core	X	X	X	X
Peripherals (except SCU)	X	X	X	X
SCU	X	Not affected	X	X

Table 3-5 Effect of Reset on Device Functions (cont'd)

Module / Function		Application Reset	Debug Reset	System Reset	Power-on Reset
On-chip Static RAMs[1]	**OVRAM**	Not affected, reliable	Not affected, reliable	Affected, un-reliable	Affected, un-reliable
	LDRAM	Not affected, reliable	Not affected, reliable	Affected, un-reliable	Affected, un-reliable
	DCACHE	Not affected, reliable	Not affected, reliable	Affected, un-reliable	Affected, un-reliable
	SPRAM	Not affected, reliable	Not affected, reliable	Affected, un-reliable	Affected, un-reliable
	PRAM	Not affected, reliable	Not affected, reliable	Affected, un-reliable	Affected, un-reliable
	CMEM	Not affected, reliable	Not affected, reliable	Affected, un-reliable	Affected, un-reliable
	ICACHE	Not affected, reliable	Not affected, reliable	Affected, un-reliable	Affected, un-reliable
Flash Memory		X [2]	Not affected, reliable	X	X
JTAG Interface		Not affected	Not affected	Not affected	Affected
OCDS		Not affected	X	Not affected	X
MCDS		Not affected	X	Not affected	X
Oscillators, PLL		Not affected	Not affected	X	X
Port Pins		X	Not affected	X	X
Pins ESRx		Not affected	Not affected	X	X

1) Reliable here means that also the redundancy is not affected by the reset.

2) Parts of the flash memory block are only reset by a class 0 reset. For more detail see the flash chapter.

3.2.5 General Reset Operation

A reset is generated if an enabled reset request trigger is asserted. Most reset request trigger can configured for the reset type that should initiated by it. No action (disabled) is one possible configuration and can be selected for a reset request trigger by setting the adequate bit field in a Reset Configuration Register to 00_B. The Debug Reset can only be requested by dedicated reset request triggers and can not be selected via a Reset Configuration Register. For more information see also register RSTCON.

The duration of a reset is defined by two independent counters. One counter for the Power-on, System, and Application Reset types and one separate counter for the Debug Reset. A separate counter for the Debug Reset was implemented to allow a non-intrusive adaptation of the reset length to the debugger needs without modification of the application setting.

3.2.6 Reset State Machine

There is one central Reset State Machine (RSM) controlling the reset generation for the complete device beside the JTAG reset domain.

Note: The JTAG reset domain is controlled by the $\overline{TRST}$ pin.

The RSM is composed of a control block responsible for the operation flow, two counters with a reload register, and a distribution logic that controls the generation of the three dedicated resets.

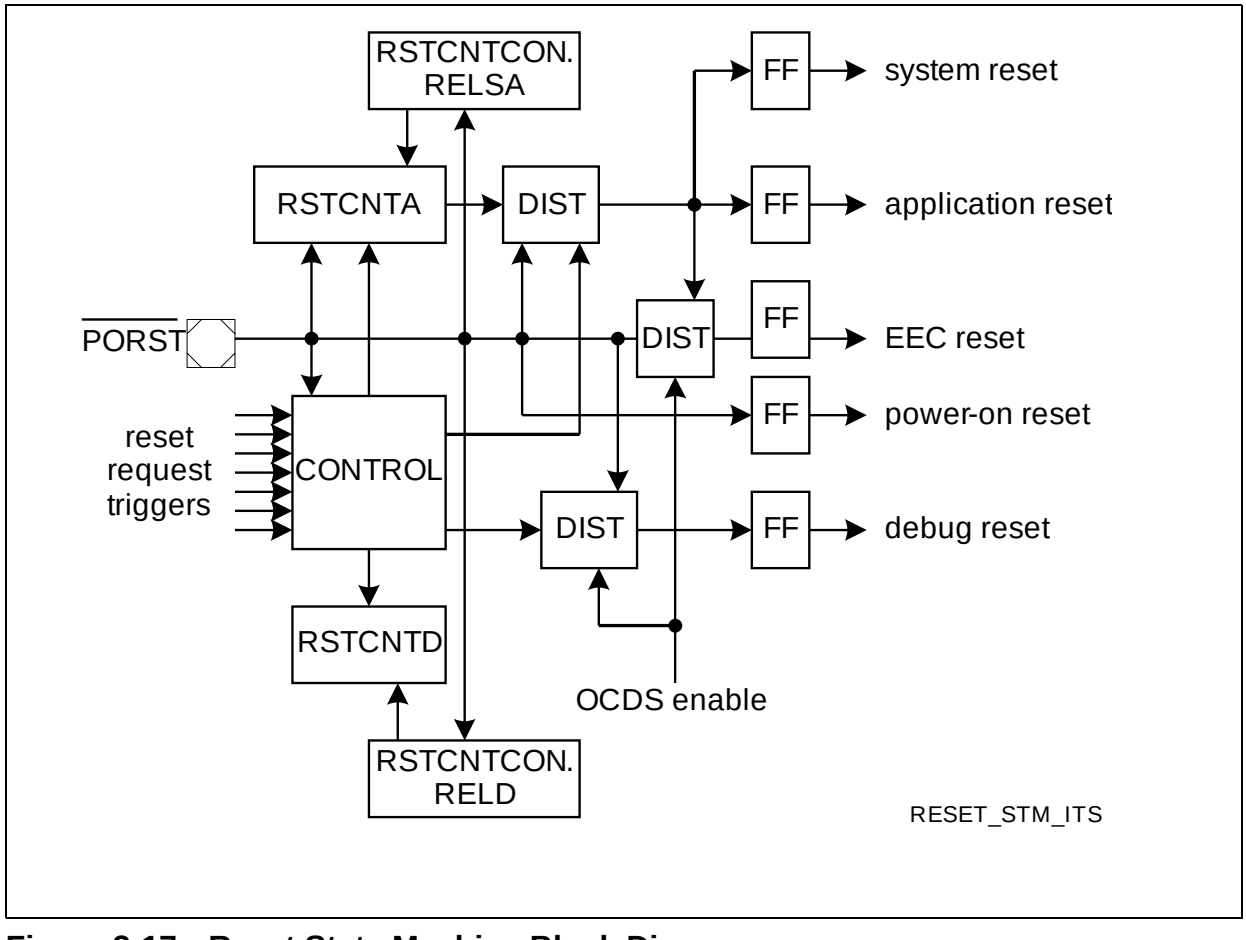


Figure 3-17 Reset State Machine Block Diagram

3.2.7 Reset Counters (RSTCNTA and RSTCNTD)

There are two reset counters implemented. RSTCNTA is the reset counter that controls the reset length for all non debug relevant resets (System Reset and Application Reset). RSTCNTD is the reset counter that controls the reset length for the Debug Reset.

The reset counters can be used for the following purposes:

- First to control the length of the internal resets.
- Second to configure the reset length in a way that the reset outputs via the ESRx pins match with the reset input requirements of external blocks connected with the reset outputs.

A reset counter RSTCNT is an 16-bit counter counting down from the reload value defined by RSTCNTCON.RELx (x = SA or D). The counter is started by the control block as soon as a reset request trigger condition becomes active (for more information see **Table 3-6** and **Table 3-7**). Whether the counter has to be started or not depends on the reset request trigger and whether the counter is already active or not. In case that the counter is inactive, not counting down, it is always started. While the counter is already active it depends on the reset of the new reset request trigger that was asserted anew if the counter is restarted or not. This behavior is summarized in **Table 3-6** and **Table 3-7**.

Table 3-6 Restart of RSTCNTA

Reset Active	New Reset Request			
	Power-on	**System**	**Debug**	**Application**
Power-on Reset	Restart	No Restart	No Restart	No Restart
System	Restart	No Restart	No Restart	No Restart
Application	Restart	Restart	No Restart	No Restart

Table 3-7 Restart of RSTCNTD

Reset Active	New Reset / Reset Type Request			
	Power-on	**System**	**Debug**	**Application**
Debug	Restart	No Restart	No Restart	No Restart

RSTCNTx ensures that a reset request trigger generates a reset of a minimum length which is configurable. But if a reset request trigger is asserted continuously longer than the counter needs for the complete count-down process the reset cannot be deasserted before the reset request trigger is also deasserted. Anyway the counter is not started again, instead the control block informs the distribution logic (DIST) that the reset still has to be asserted until the reset request trigger is deasserted.

The reset state of the control block is implemented such that the RSTCNTA is started and two reset types (System and Application) have to be asserted by the distribution logic.

The reset state of the control block is implemented such that the RSTCNTD is started and the Debug Reset has to be asserted by the distribution logic.

Note: The reset counter should not be configured with a value lower than 20_H due to the implementation.

3.2.8 De-assertion of a Reset

The reset is de-asserted when all of the following conditions are fulfilled.

- The reset counter has expired (reached zero)
- No reset request trigger that is configured to generate the same reset is currently asserted

3.2.8.1 Example1:

Reset request trigger A is asserted and leads to an Application Reset. If the reset request trigger is de-asserted before RSTCNTA reached zero the Application Reset is de-asserted when RSTCNTA reaches zero. If the reset request trigger is de-asserted after RSTCNTA reached zero the Application Reset is de-asserted when the reset request trigger is de-asserted.

3.2.8.2 Example2:

Reset request trigger A is asserted and leads to an Application Reset. Reset request trigger A is de-asserted before RSTCNTA reached zero. Reset request trigger B is asserted after reset request trigger A but before RSTCNTA reaches zero. Reset request trigger B is also configured to result in an Application Reset. If the reset request trigger B is de-asserted before RSTCNTA reached zero the Application Reset is de-asserted when RSTCNTA reaches zero. If the reset request trigger B is de-asserted after RSTCNTA reached zero the Application Reset is de-asserted when the reset request trigger B is de-asserted.

3.2.8.3 Example3:

Reset request trigger A is asserted and leads to a System Reset. Reset request trigger A is de-asserted before RSTCNTA reached zero. Reset request trigger B is asserted after reset request trigger A but before RSTCNTA reaches zero. Reset request trigger B is configured to result in an Application Reset. If the reset request trigger B is de-asserted before RSTCNTA reached zero the System and Application Resets are de-asserted when RSTCNTA reaches zero. If the reset request trigger B is de-asserted after

RSTCNTA reached zero the Application Reset is de-asserted when the reset request trigger B is de-asserted.

3.2.9 Reset Triggers

There are two types of reset triggers for the reset control logic:

- Triggers that lead to a specific reset
- Triggers that lead to a configurable reset

3.2.9.1 Specific Reset Triggers

These triggers lead to a predefined reset if the trigger is asserted. Additionally these triggers can not be enabled / disabled. All specific reset are listed in **Table 3-8**.

Table 3-8 Specific Reset Triggers

Reset Trigger	Reset Type Request
Debug (OCDS from Cerberus)	Debug Reset
Cerberus 0 (CB0)	System Reset
Cerberus 1 (CB1)	Debug Reset
Cerberus 3 (CB3)	Application Reset

3.2.9.2 Configurable Reset Triggers

These triggers lead to a selectable reset if the trigger is asserted. Additionally these triggers can be enabled / disabled. The behavior of the reset triggers action is defined by the configuration of the dedicated bit field for this trigger in register RSTCON.

3.2.10 Debug Specific Behavior

For safety reasons it is required by the debugger that if the OCDS system is disabled a Debug Reset is also asserted every time an Application Reset is asserted.

3.2.11 EEC Reset Specific Behavior

The complete EEC part is reset with the Power-on Reset. For safety reasons it is required by the debugger that if the OCDS system is disabled a EEC reset is also asserted every time an Application Reset is asserted.

3.2.12 Reset Controller Registers

3.2.12.1 Status Registers

After a reset has been executed, the Reset Status registers provide information on the source of the last reset(s). The reset status registers are updated upon each reset cycle. A reset cycle is finished when all resets are de-asserted. Within a reset cycle the status flags are used as sticky flags.

RSTSTAT
Reset Status Register

(050_H) **Reset Value: $0001\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0										TP	CB3	CB1	CB0	OCDS	PORST
r										rh	rh	rh	rh	rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0											SW	WDT	0	ESR1	ESR0
r											rh	rh	r	rh	rh

Field	Bits	Type	Description
ESR0	0	rh	**Reset Request Trigger Reset Status for ESR0** 0_B The last reset was not requested by this reset trigger 1_B The last reset was requested by this reset trigger *Note: This bit is set if the ESR0 pin is configured as open drain for output characteristics ($IOCR.PC0 = 1101_H$ or 1110_H) and ESR0 is configured to generate a Reset ($RSTCON.ESR0 = 01_B$ or 10_B) AND a reset is signaled by the ESR0 pin to the outside. That this bit is set under this condition can be avoided by either setting $IOCR.PC0 = 1001_H$ or 1010_H or $RSTCON.ESR0 = 00_B$.*

Field	Bits	Type	Description
ESR1	1	rh	**Reset Request Trigger Reset Status for ESR1** 0_B The last reset was not requested by this reset trigger 1_B The last reset was requested by this reset trigger *Note: This bit is set if the ESR1 pin is configured as open drain for output characteristics (IOCR.PC1 = 1101_H or 1110_H) and ESR1 is configured to generate a Reset (RSTCON.ESR1 = 01_B or 10_B) AND a reset is signaled by the ESR1 pin to the outside. That this bit is set under this condition can be avoided by either setting IOCR.PC1 = 1001_H or 1010_H or RSTCON.ESR1 = 00_B.*
WDT	3	rh	**Reset Request Trigger Reset Status for WDT** 0_B The last reset was not requested by this reset trigger 1_B The last reset was requested by this reset trigger
SW	4	rh	**Reset Request Trigger Reset Status for SW** 0_B The last reset was not requested by this reset trigger 1_B The last reset was requested by this reset trigger
PORST	16	rh	**Reset Request Trigger Reset Status for PORST** 0_B The last reset was not requested by this reset trigger 1_B The last reset was requested by this reset trigger This bit is also set if the bits CB0, CB1, and CB3 are set in parallel.
OCDS	17	rh	**Reset Request Trigger Reset Status for OCDS** 0_B The last reset was not requested by this reset trigger 1_B The last reset was requested by this reset trigger

Field	Bits	Type	Description
CB0	18	rh	**Reset Request Trigger Reset Status for Cerberus System Reset** 0_B The last reset was not requested by this reset trigger 1_B The last reset was requested by this reset trigger
CB1	19	rh	**Reset Request Trigger Reset Status for Cerberus Debug Reset** 0_B The last reset was not requested by this reset trigger 1_B The last reset was requested by this reset trigger
CB3	20	rh	**Reset Request Trigger Reset Status for Cerberus Application Reset** 0_B The last reset was not requested by this reset trigger 1_B The last reset was requested by this reset trigger
TP	21	rh	**Reset Request Trigger Reset Status for TP** 0_B The last reset was not requested by this reset trigger 1_B The last reset was requested by this reset trigger
0	2, [15:5], [31:22]	r	**Reserved** Read as 0; should be written with 0.

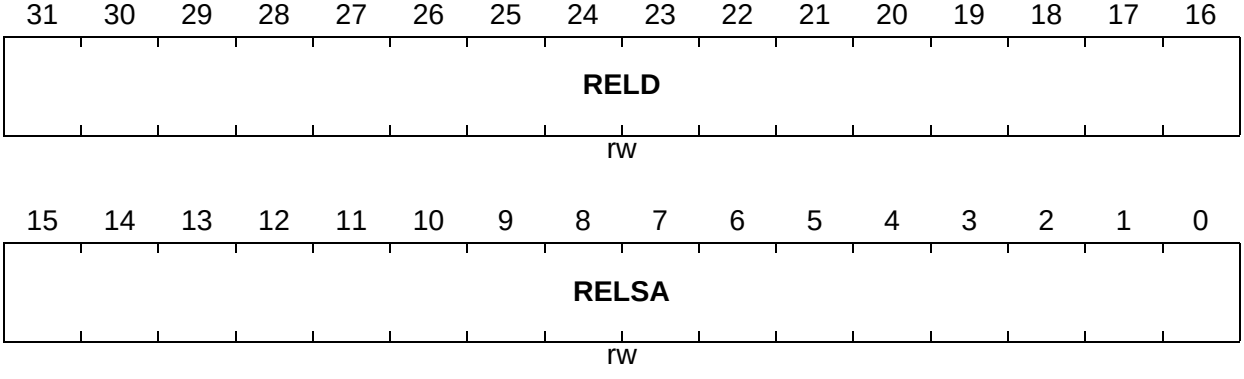
3.2.12.2 Configuration Registers

Reset Counter Control Register

This register controls the reset length settings for the three resets.

RSTCNTCON
Reset Counter Control Register

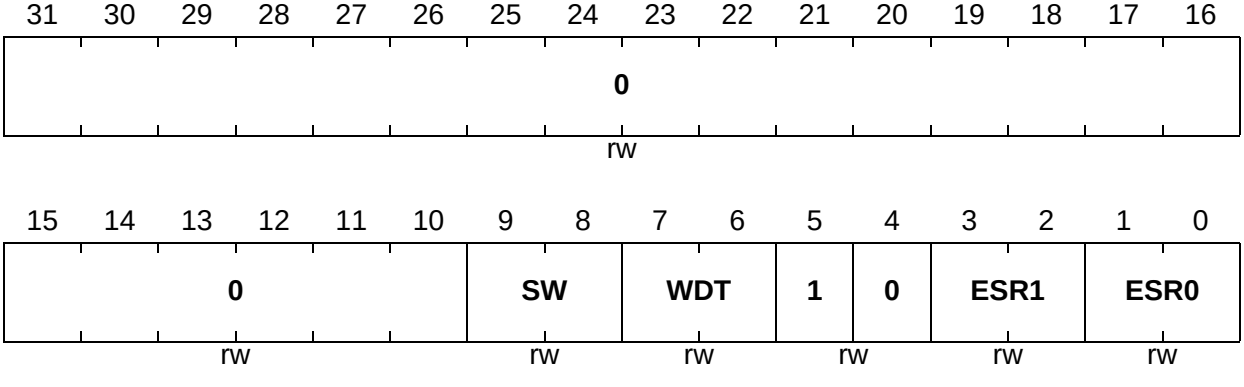
(054_H) **Reset Value: 05BE 05BE$_H$**



Field	Bits	Type	Description
RELSA	[15:0]	rw	**System Application Reset Counter Reload Value** This bit field defines the reload value of RSTCNTA. This value is always used when counter RSTCNTA is started.
RELD	[31:16]	rw	**Debug 1 Reset Counter Reload Value** This bit field defines the reload value of RSTCNTD. This value is always used when counter RSTCNTD is started.

RSTCON
Reset Configuration Register

(058_H) **Reset Value: 0000 02A2$_H$**

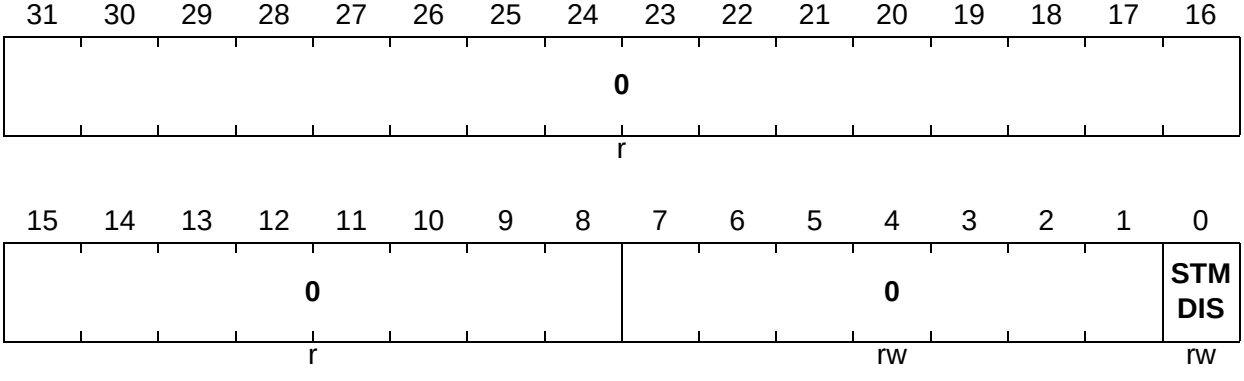


Field	Bits	Type	Description
ESR0	[1:0]	rw	**ESR0 Reset Request Trigger Reset Configuration** This bit field defines which reset is generated by a reset request trigger from ESR0 reset. 00_B No reset is generated for a trigger of ESR0 01_B A System Reset is generated for a trigger of ESR0 reset 10_B An Application Reset is generated for a trigger of ESR0 reset 11_B Reserved, do not use this combination
ESR1	[3:2]	rw	**ESR1 Reset Request Trigger Reset Configuration** This bit field defines which reset is generated by a reset request trigger from ESR1 reset. 00_B No reset is generated for a trigger of ESR1 01_B A System Reset is generated for a trigger of ESR1 reset 10_B An Application Reset is generated for a trigger of ESR1 reset 11_B Reserved, do not use this combination
WDT	[7:6]	rw	**WDT Reset Request Trigger Reset Configuration** This bit field defines which reset is generated by a reset request trigger from WDT reset. 00_B No reset is generated for a trigger of WDT 01_B A System Reset is generated for a trigger of WDT reset 10_B An Application Reset is generated for a trigger of WDT reset 11_B Reserved, do not use this combination
SW	[9:8]	rw	**SW Reset Request Trigger Reset Configuration** This bit field defines which reset is generated by a reset request trigger from software reset. 00_B No reset is generated for a trigger of software reset 01_B A System Reset is generated for a trigger of Software reset 10_B An Application Reset is generated for a trigger of Software reset 11_B Reserved, do not use this combination

Field	Bits	Type	Description
1	5	rw	**Reserved** Should be written with 1.
0	4, [31:10]	rw	**Reserved** Should be written with 0.

ARSTDIS
Application Reset Disable Register

(05C$_H$) **Reset Value: 0000 0000$_H$**



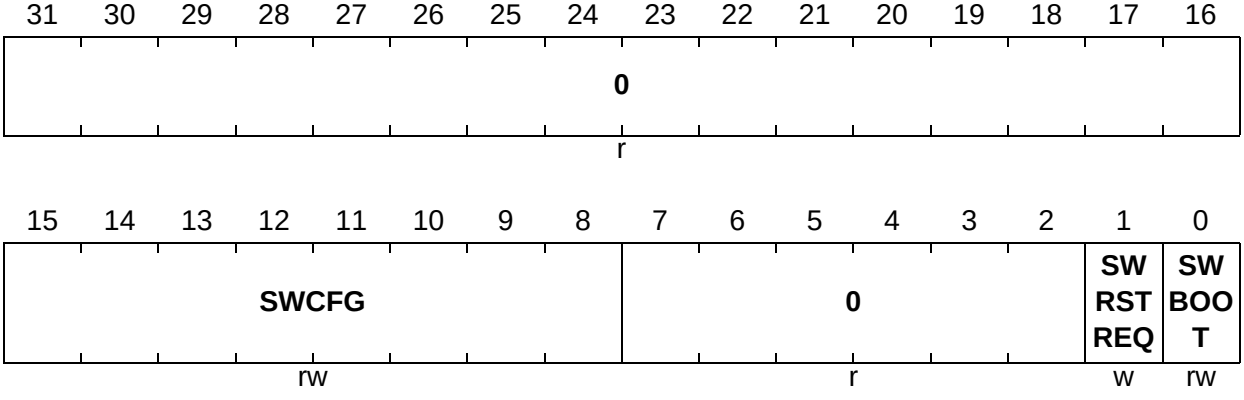
Field	Bits	Type	Description
STMDIS	0	rw	**STM Disable Reset** This bit field defines if an Application Reset leads to an reset for the STM. 0_B An Application Reset resets the STM 1_B An Application Reset has no effect for the STM
0	[7:1]	rw	**Reserved** Should be written with 0.
0	[31:8]	r	**Reserved** Read as 0; should be written with 0.

SW Reset Configuration Register

This register controls the SW Reset operation.

SWRSTCON
Software Reset Configuration Register

(060$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
SWBOOT	0	rw	**Software Boot Configuration Selection** 0_B Bit field STSTAT.HWCFG is not updated with the content of SWCFG upon an Application Reset 1_B Bit field STSTAT.HWCFG is updated with the content of SWCFG upon an Application Reset
SWRSTREQ	1	w	**Software Reset Request** 0_B No SW Reset is requested 1_B A SW Reset request trigger is generated This bit is automatically cleared and read always as zero.
SWCFG	[15:8]	rw	**Software Boot Configuration** A software boot configuration different from the external applied hardware configuration can be specified with these bits. The configuration encoding is equal to the HWCFG encoding in register STSTAT.
0	[7:2], [31:16]	r	**Reserved** Read as 0; should be written with 0.

3.3 External Interface

The SCU provides interface pads for system purpose. Various functions are covered by these pins. Due to the different tasks some of the pads can not be shared with other functions but most of them can. The following functions are covered by the SCU controlled pads:

- Reset request triggers
- Reset indication
- Trap request triggers
- Interrupt request triggers
- Non SCU module triggers

The first three points are covered by the ESR pads and the last two points by the ERU pads.

3.3.1 External Service Requests ($\overline{\text{ESRx}}$)

The ESR pins can be used to trigger either a reset, a trap (NMI), as reset output, or as data pin.

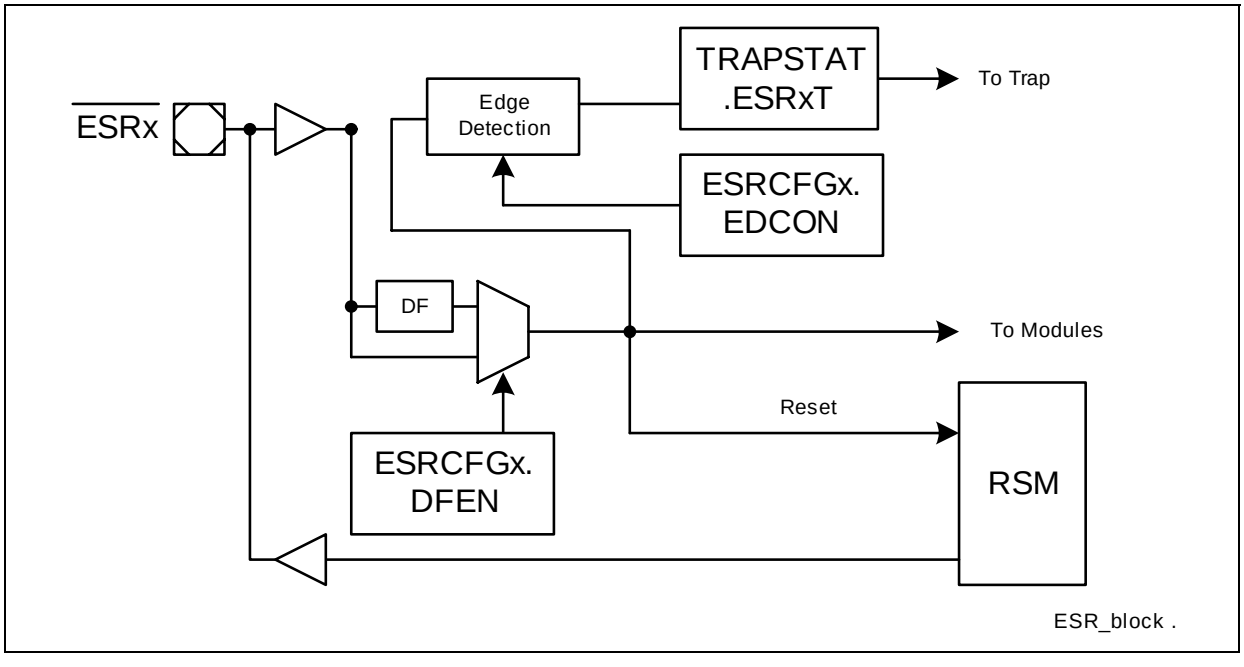


Figure 3-18 ESR Operation

3.3.1.1 $\overline{\text{ESRx}}$ as Reset Request Trigger

An $\overline{\text{ESR0}}$/$\overline{\text{ESR1}}$ reset request trigger leads to a System or Application Reset. The type of the reset can be configured via RSTCON.ESRx.

In order to be safely recognized $\overline{\text{ESR0}}$/$\overline{\text{ESR1}}$ has to be active for a minimum of 2 f_{FPI} clock cycles.

The input signal $\overline{\text{ESR0}}$/$\overline{\text{ESR1}}$/$\overline{\text{ESR2}}$ have digital filters (3-stage median filters), that can be disabled.

A 3-stage median filter samples with f_{FPI} three consecutive clock cycles and the output is defined by the majority of the three sampled values.

The behavior of all three $\overline{\text{ESR0}}$/$\overline{\text{ESR1}}$ pins can be configured by programming the registers ESRCFGn. The pad control functionality can be configured independently for each pin, comprising:

- A selection of the driver type (open-drain or push-pull)
- An enable function for the output driver (input and/or output capability)
- An enable function for the pull-up/down resistance

3.3.1.2 $\overline{\text{ESR}}$x as Reset Output

The external pins $\overline{\text{ESR0}}$/$\overline{\text{ESR1}}$ can serve as an reset output (open drain) for the Application Reset.

Note: The reset output is only asserted for the duration the reset counter RSTCNTA is active. During a possible reset extension the reset output is not longer asserted.

If the pin $\overline{\text{ESR0}}$/$\overline{\text{ESR1}}$ is enabled as reset output and the input level is low while the output stage is disabled (indicating that it is still driven low externally), the reset circuitry holds the chip in reset until a high level is detected on $\overline{\text{ESR0}}$/$\overline{\text{ESR1}}$. The internal output stage drives a low level during reset only while RSTCNTA is active. It deactivates the output stage when the time defined by RSTCNTCON.RELSA has passed.

3.3.1.3 ESR Registers

ESRCFG0
ESR0 Configuration Register

(070_H) **Reset Value: 0000 0110_H**

ESRCFG1
ESR1 Configuration Register

(074_H) **Reset Value: 0000 0090_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							EDCON		0		DFEN	0			
r							rw		r		rw	r			

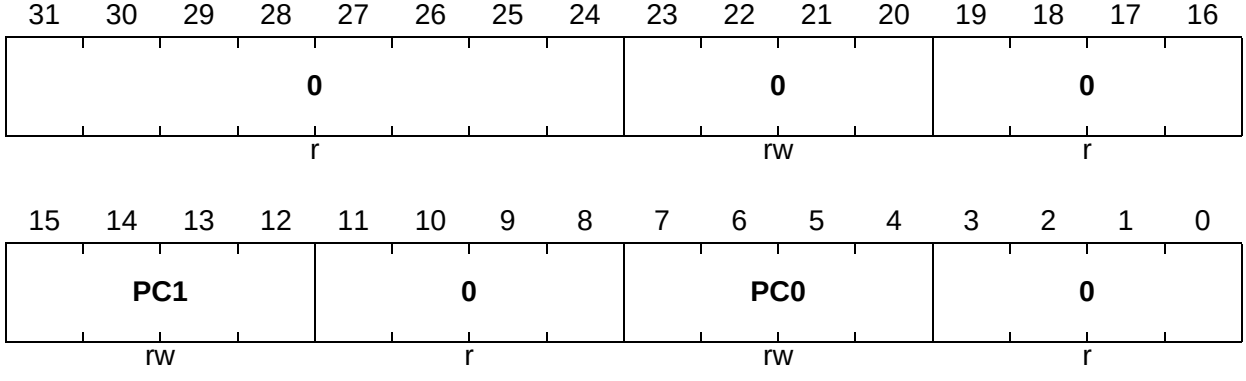
Field	Bits	Type	Description
DFEN	4	rw	**Digital Filter Enable** This bit defines if the 3-stage median filter of the ESR0 is used or bypassed. 0_B The filter is bypassed 1_B The filter is used
EDCON	[8:7]	rw	**Edge Detection Control** This bit field defines the edges that lead to an ESR0 trigger of the synchronous path. 00_B No trigger is generated 01_B A trigger is generated upon a rising edge 10_B A trigger is generated upon a falling edge 11_B A trigger is generated upon a rising OR falling edge
0	[3:0], [6:5], [31:9]	r	**Reserved** Read as 0; should be written with 0.

Input/Output Control Registers

The input/output control registers select the digital output and input driver functionality and characteristics of the pin. Direction (input or output), pull-up or pull-down devices for

inputs, and push-pull or open-drain functionality for outputs can be selected by the corresponding bit fields PCx (x = 0-1).

IOCR
Input/Output Control Register (0A0$_H$) Reset Value: 0020 10E0$_H$



Field	Bits	Type	Description
PC0	[7:4]	rw	**Control for $\overline{\text{ESR0}}$ Pin** This bit field defines the $\overline{\text{ESR0}}$ pin functionality according to the coding table (see **Table 3-9**).
PC1	[15:12]	rw	**Control for $\overline{\text{ESR1}}$ Pin** This bit field defines the $\overline{\text{ESR1}}$ pin functionality according to the coding table (see **Table 3-10**).
0	[23:20]	rw	**Reserved** Have to be written with 0010_B.
0	[3:0], [11:8], [19:16], [31:24]	r	**Reserved** Read as 0; should be written with 0.

Pad Control Coding

Table 3-9 describes the coding of the PC0 bit field that determine the port line functionality.

Table 3-9 PC0 Coding

PC0[3:0]	I/O	Output Characteristics	Selected Pull-up/Pull-down/ Selected Output Function
$0X00_B$	Input is active and not inverted; Output is inactive		No input pull device connected
$0X01_B$			Input pull-down device connected
$0X10_B$			Input pull-up device connected
$0X11_B$			No input pull device connected
1000_B	Input is active and not inverted; Output is active	Push-pull	General-purpose Output
1001_B			Output drives a 0 for System Resets, a weak pull-up is active otherwise
1010_B			Output drives a 0 for Application Resets, a weak pull-up is active otherwise
1011_B			Reserved, do not use this combination
1100_B	the input is active and not inverted; Output is active	Open-drain	General-purpose Output
1101_B			Output drives a 0 for System Resets, a weak pull-up is active otherwise
1110_B			Output drives a 0 for Application Resets, a weak pull-up is active otherwise
1111_B			Reserved, do not use this combination

Pad Control Coding

Table 3-10 describes the coding of the PC1 bit field that determine the port line functionality.

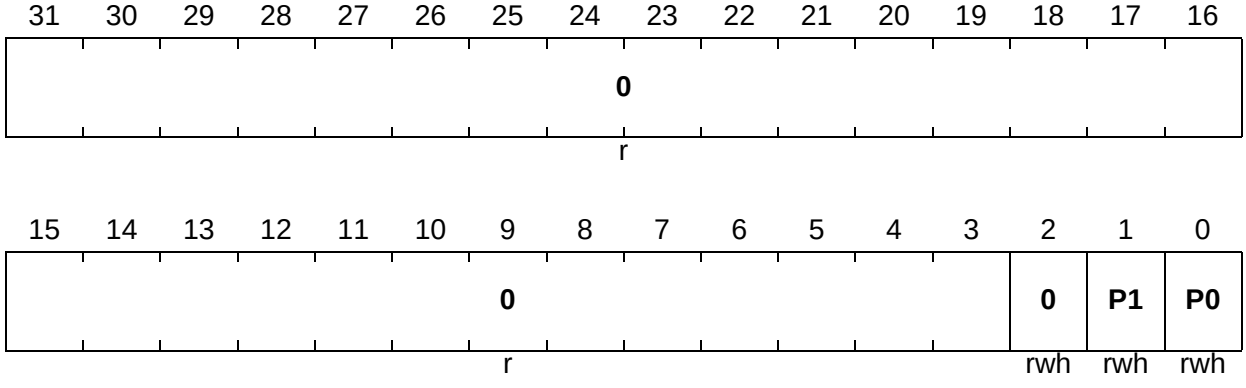
Table 3-10 PC1 Coding

PC1[3:0]	I/O	Output Characteristics	Selected Pull-up/Pull-down/ Selected Output Function
$0X00_B$	Input is active and not inverted; Output is inactive		No input pull device connected
$0X01_B$			Input pull-down device connected
$0X10_B$			Input pull-up device connected
$0X11_B$			No input pull device connected
1000_B	Input is active and not inverted; Output is active	Push-pull	General-purpose Output
1001_B			Output drives a 0 for System Resets, a ’Z’ otherwise
1010_B			Output drives a 0 for Application Resets, a ’Z’ otherwise
1011_B			Reserved, do not use this combination
1100_B	the input is active and not inverted; Output is active	Open-drain	General-purpose Output
1101_B			Output drives a 0 for System Resets, a ’Z’ otherwise
1110_B			Output drives a 0 for Application Resets, a ’Z’ otherwise
1111_B			Reserved, do not use this combination

Output Register

The output register determines the value of a GPIO pin when it is selected by IOCR as output. Writing a 0 to a OUT.Px (x = 0-1) bit position delivers a low level at the corresponding output pin. A high level is output when the corresponding bit is written with a 1. Note that each single bit or group of bits of OUT.Px can be set/cleared by writing appropriate values into the output modification register OMR.

OUT
Output Register **($0A4_H$)** **Reset Value: $0000\ 0000_H$**

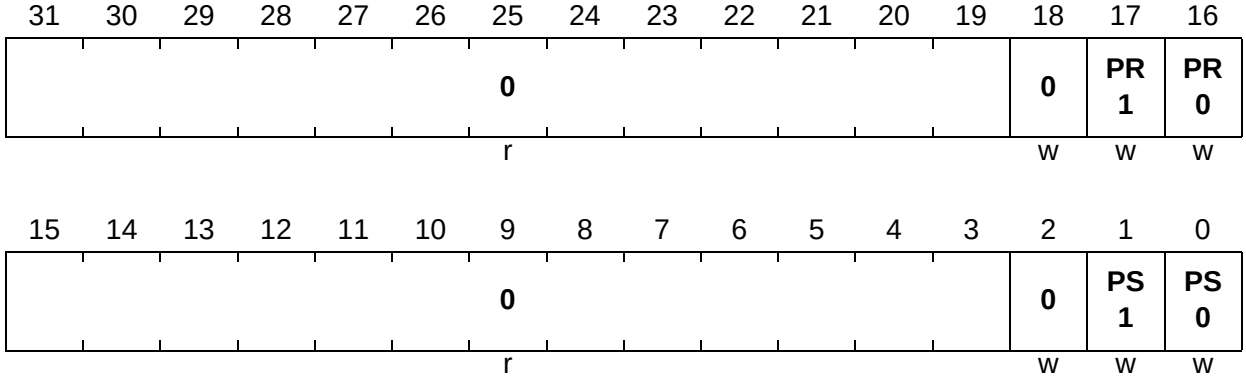


Field	Bits	Type	Description
Px **(x = 0-1)**	x	rwh	**Output Bit x** This bit determines the level at the output pin $\overline{ESR}$x if the output is selected as GPIO output. 0_B The output level of $\overline{ESR}$x is 0 1_B The output level of $\overline{ESR}$x is 1 Px can also be set/cleared by control bits of the OMR register.
0	2	rwh	**Reserved** Have to be written with 0.
0	[31:3]	r	**Reserved** Read as 0; should be written with 0.

Output Modification Register

The output modification register contains control bits that make it possible to individually set, clear, or toggle the logic state of a single pad by manipulating the output register.

OMR
Output Modification Register **($0A8_H$)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
PSx (x = 0-1)	x	w	**Set Bit x** Setting this bit will set or toggle the corresponding bit in the output register OUT. The function of this bit is shown in **Table 3-11**.
PRx (x = 0-1)	x + 16	w	**Clear Bit x** Setting this bit will clear or toggle the corresponding bit in the port output register OUT. The function of this bit is shown in **Table 3-11**.
0	2, 18	w	**Reserved** Read as 0; have to be written with 0.
0	[15:3], [31:19]	r	**Reserved** Read as 0; should be written with 0.

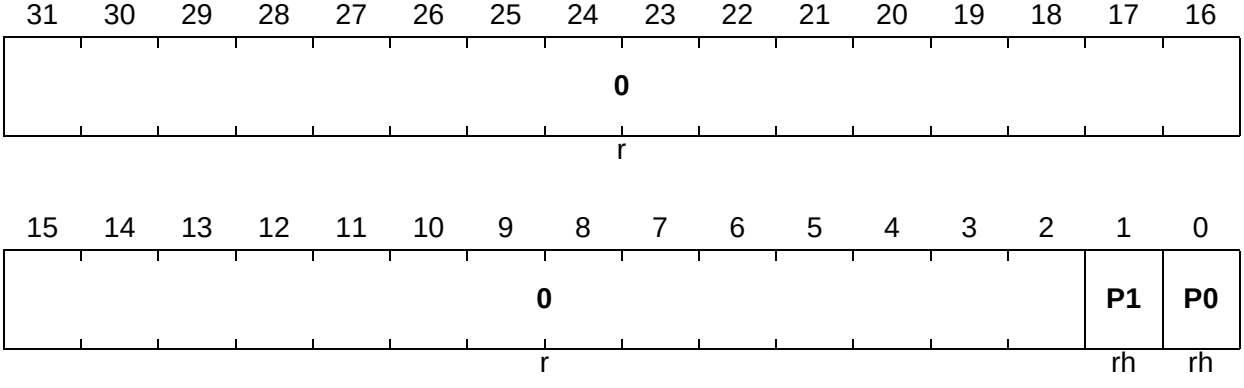
Table 3-11 Function of the Bits PRx and PSx

PRx	PSx	Function
0	0	Bit OUT.Px is not changed
0	1	Bit OUT.Px is set
1	0	Bit OUT.Px is cleared
1	1	Bit OUT.Px is toggled

Input Register

The logic level of a GPIO pin can be read via the read-only port input register IN. Reading the IN register always returns the current logical value at the GPIO pin independently whether the pin is selected as input or output.

IN
Input Register **($0AC_H$)** **Reset Value: 0000 000X_H**



Field	Bits	Type	Description
Px **(x = 0-1)**	x	rh	**Input Bit x** This bit indicates the level at the input pin $\overline{ESR}$x. 0_B The input level of $\overline{ESR}$x is 0 1_B The input level of $\overline{ESR}$x is 1
0	[31:2]	r	**Reserved** Read as 0.

3.3.2 External Request Unit (ERU)

The External Request Unit (ERU) is a versatile event and pattern detection unit. Its major task is the **generation of interrupts based on selectable trigger events at different inputs**, e.g. to generate external interrupt requests if an edge occurs at an input pin. The detected events can also be used by other modules to trigger or to gate module-specific actions.

3.3.2.1 Introduction

The ERU of the TC1797 can be split in three main functional parts:

- 4 independent **Input Channels x** for input selection and conditioning of trigger or gating functions
- Event distribution: A **Connecting Matrix** defines the events of the Input Channel x that lead to a reaction of an Output Channel y.
- 4 independent **Output Channels y** for combination of events, definition of their effects and distribution to the system (interrupt generation, ...)

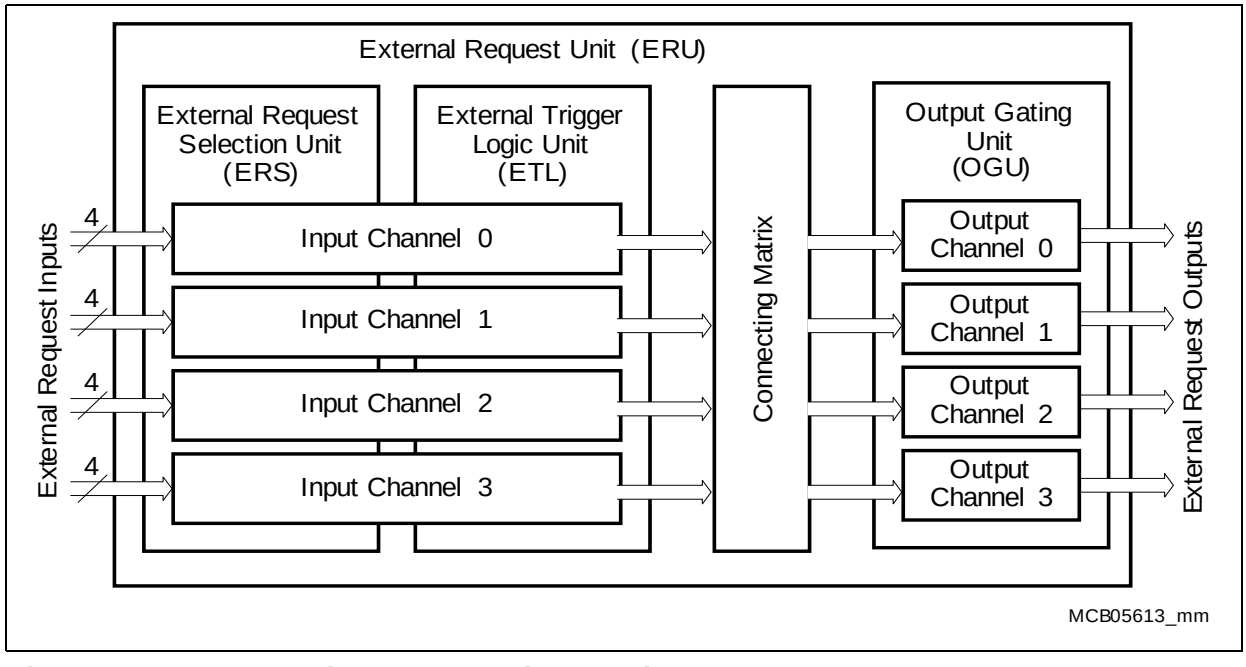


Figure 3-19 External Request Unit Overview

These tasks are handled by the following building blocks:

- An **External Request Select Unit (ERSx)** per Input Channel allows the selection of one input vector out of the 4 possible inputs available.
- An **Event Trigger Logic (ETLx)** per Input Channel allows the definition of the transition (edge selection, or by software) that lead to a trigger event and can also store this status. Here, the input levels of the selected signals are translated into

events (event detected = event flag becomes set, independent of the polarity of the original input signals).

- The **Connecting Matrix** distributes the events and status flags generated by the Input Channels to the Output Channels.
- An **Output Gating Unit (OGUy)** per Output Channel that combines the available trigger events and status information from the Input Channels. An event of one Input Channel can lead to reactions of several Output Channels, or also events of several Input Channels can be combined to a reaction of one Output Channel (pattern detection).
  Different types of reactions are possible, e.g. interrupt generation (based on signals ERU_IOUTy).

3.3.2.2 ERU Pin Connections

Figure 3-20 shows the ERU input connections.

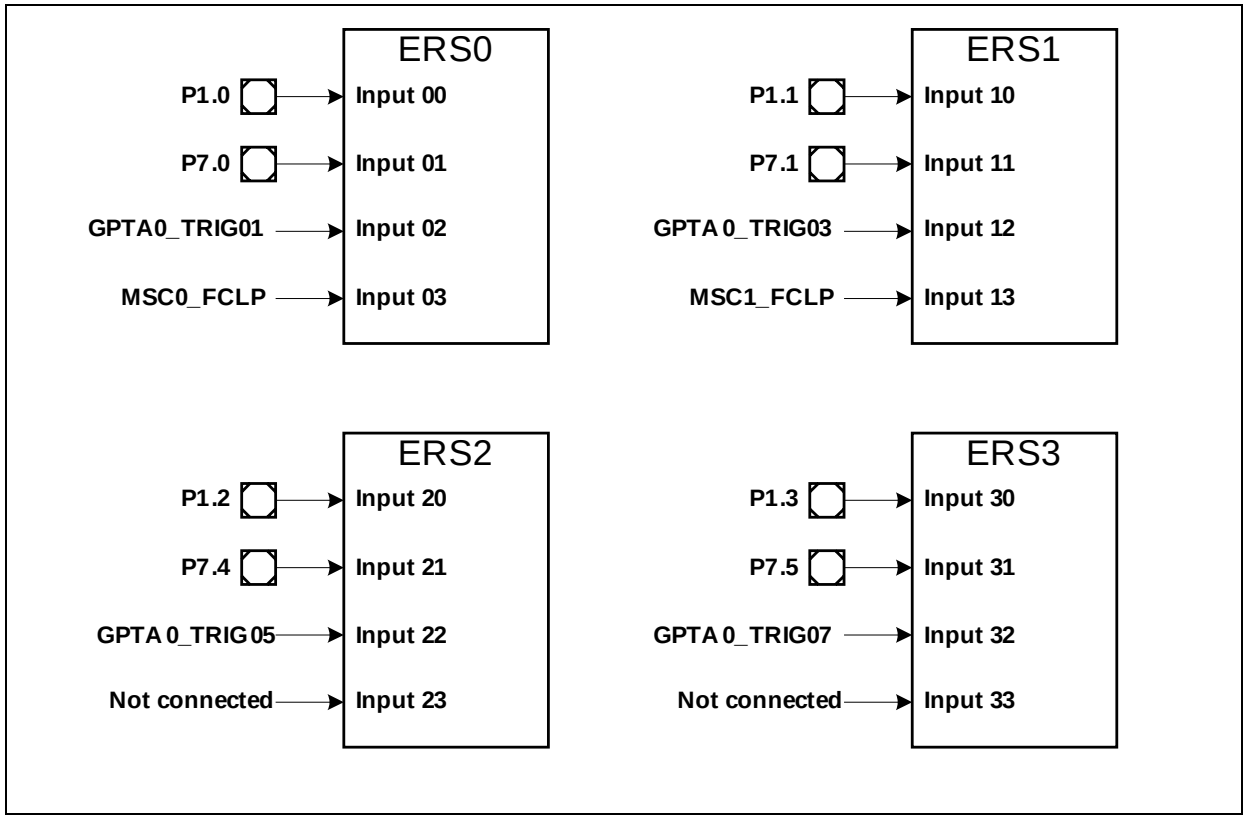


Figure 3-20 ERU Inputs Overview

The inputs to the ERU can be selected from a large number of input signals. While some of the inputs come directly from a pin, other inputs use signals from various peripheral modules.

Usually, such signals would be selected for an ERU function when the input function to the other module is not used otherwise, or the module is not used at all. However, it is also possible to select a input which is actually needed in the other module, and to use it also in the ERU to provide for certain trigger functions, eventually combined with other signals (e.g. to generate an interrupt trigger in case a start of frame is detected at a selected communication).

3.3.2.3 External Request Select Unit (ERS)

Each ERS combines four inputs to the one input signal of the respective input channel. **Figure 3-21** shows the structure of this block.

In addition to the direct choice of either input Ax or Bx or their inverted values, the possible logical combinations for two selected inputs are a logical AND or a logical OR.

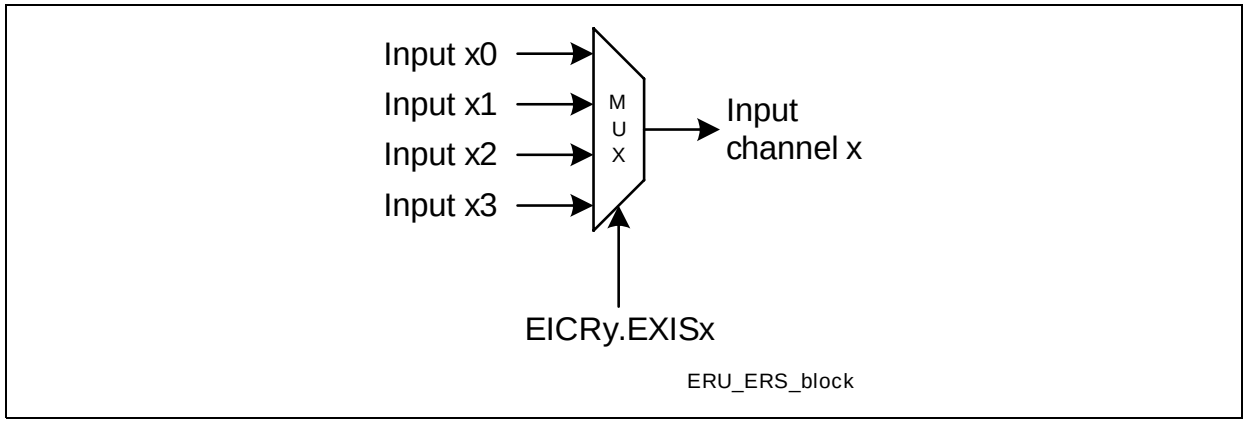


Figure 3-21 External Request Select Unit Overview

The ERS unit for channel x is controlled via bit field ERCIy.EXISx.

3.3.2.4 Event Trigger Logic (ETL)

For each Input Channel x, an event trigger logic ETLx derives a trigger event and a status from the input channel x delivered by the associated ERSx unit. Each ETLx is based on an edge detection block, where the detection of a rising or a falling edge can be individually enabled. Both edges lead to a trigger event if both enable bits are set (e.g. to handle a toggling input).
Each pair of the four ETL units has an associated EICRy register, that controls all options of an ETL (the register also holds control bits for the associated ERS unit pair, e.g. EICR0 to control ESR0 and ESR1 plus and ETL0 and ETL1).

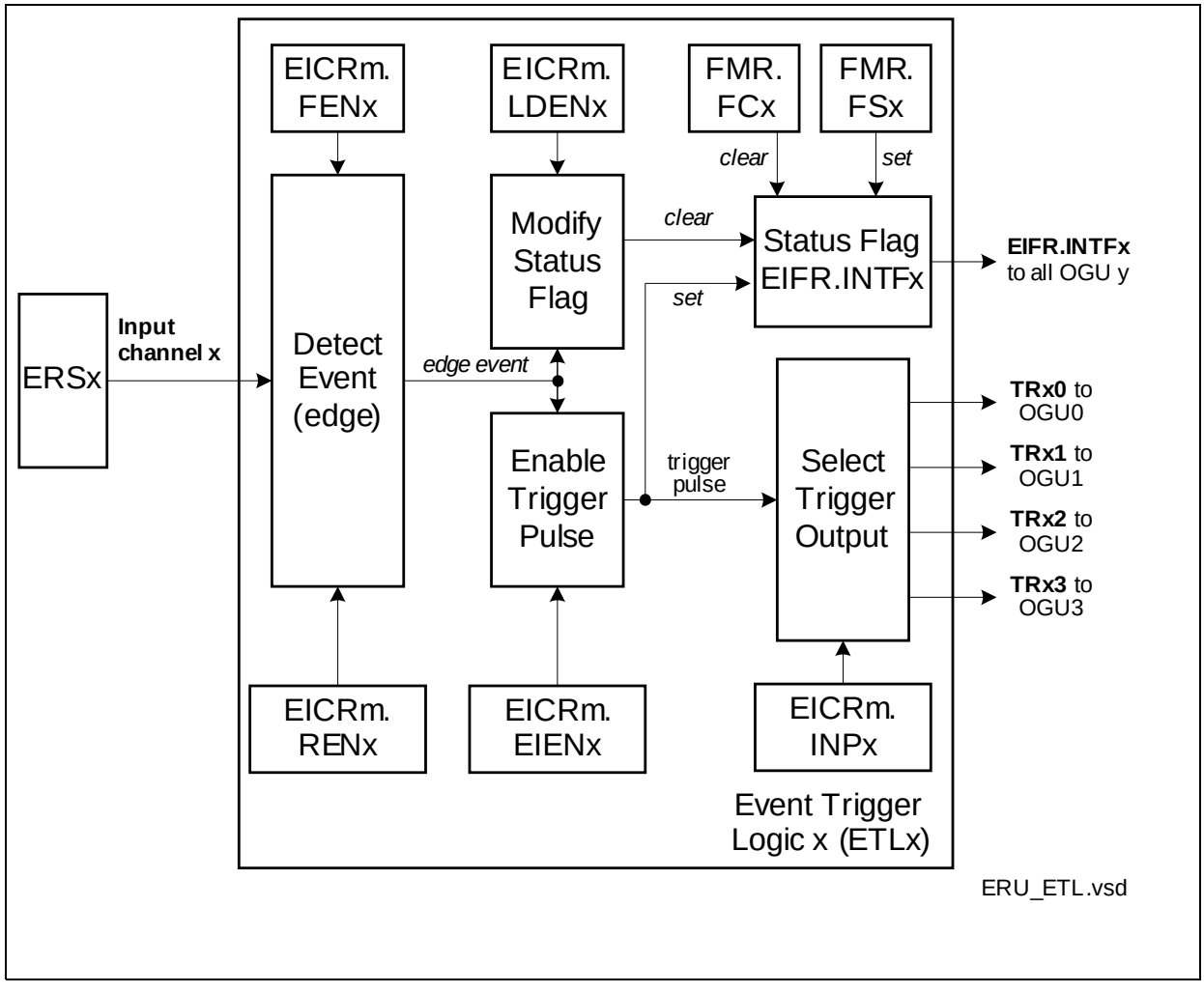


Figure 3-22 Event Trigger Logic Overview

When the selected event (edge) is detected, the status flag EIFR.INTFx becomes set. the status flag is cleared automatically if the “opposite” event is detected if enabled so via bit EICRy.LDENx = 1. For example, if only the falling edge detection is enabled to set the status flag, it is cleared when the rising edge is detected. In this mode, it can be used for pattern detection where the actual status of the input is important (enabling both edge detections is not useful in this mode).
The output of the status flag is connected to all following Output Gating Units (OGUz) in parallel (see **Figure 3-23**) to provide **pattern detection capability of all OGUz** units based on different or the same status flags.

In addition to the modification of the status flag, a trigger pulse output TRxz of ETLx can be enabled (by bit EICRy.EIENx) and selected to **trigger actions in one of the OGUz** units. The target OGUz for the trigger is selected by bit field EICRy.INPx.
The trigger becomes active when the selected edge event is detected, independently from the status flag EIFR.INTFx.

3.3.2.5 Connecting Matrix

The connecting matrix distributes the trigger signals (TRxy) and status signals (EIFR.INPFx) from the different ETLx units between the OGUy units. **Figure 3-23** provides a complete overview of the connections between the ETLx and the OGUz units.

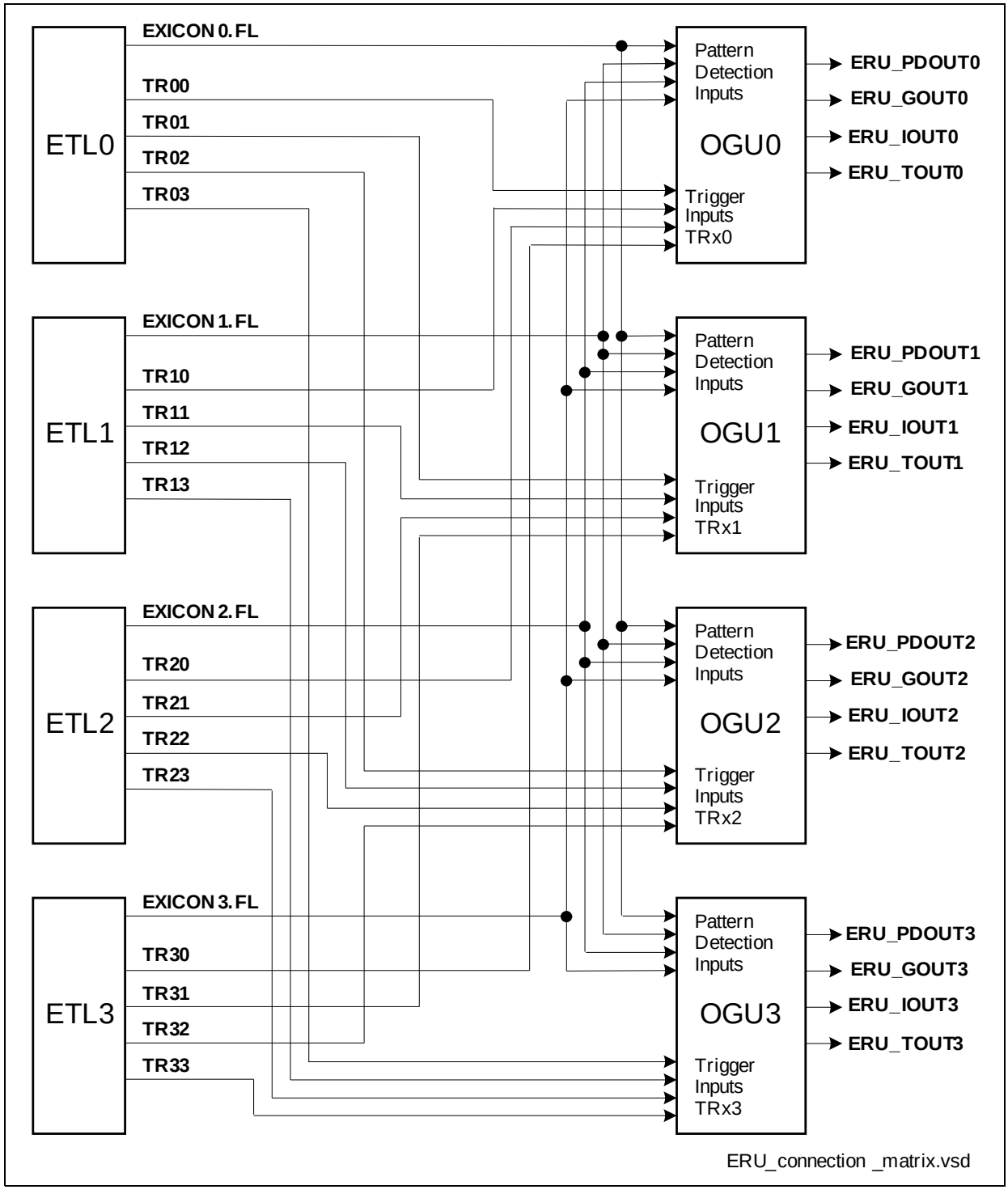


Figure 3-23 Connecting Matrix between ETLx and OGUy

3.3.2.6 Output Gating Unit (OGU)

Each OGUy unit combines the available trigger events and status flags from the Input Channels and distributes the results to the system. **Figure 3-24** illustrates the logic blocks within an OGUy unit. All functions of an OGUy unit are controlled by the associated IGCRm registers, one for each pair of output channels e.g. IGCR1 for OGU2 and OGU3. The function of an OGUy unit can be split into two parts:

- **Trigger combination**:
  All trigger signals TRxy from the Input Channels that are enabled and directed to OGUy and a pattern change event (if enabled) are logically OR-combined.
- **Pattern detection**:
  The status flags EIFR.INTFx of the Input Channels can be enabled to take part in the pattern detection. A pattern match is detected while all enabled status flags are set.

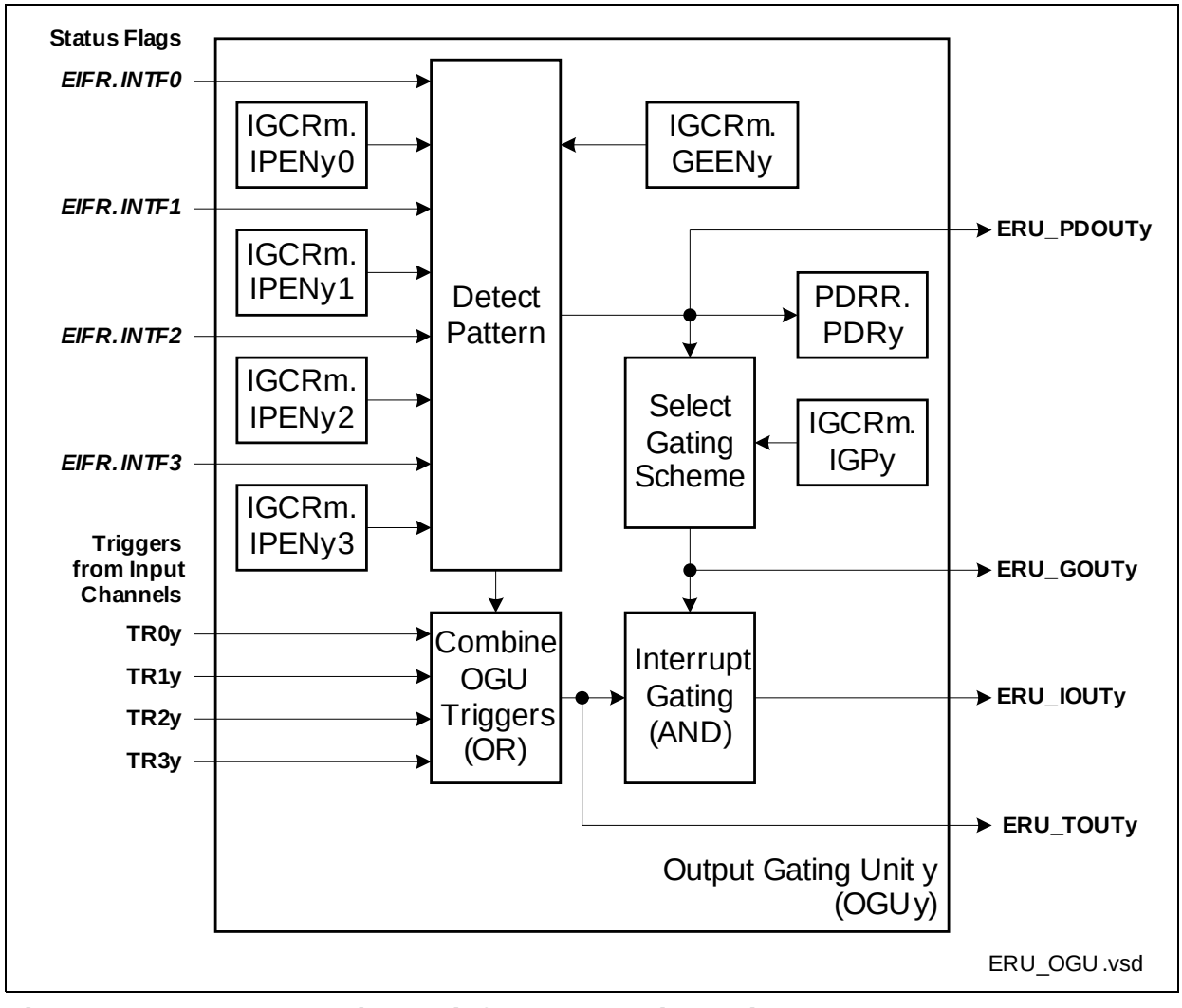


Figure 3-24 Output Gating Unit for Output Channel y

Each OGUy units generates 4 output signals that are distributed to the system.

- **ERU_PDOUTy** to directly output the pattern match information for gating purposes in other modules (pattern match = 1).
- **ERU_GOUTy** to output the pattern match or pattern miss information (inverted pattern match), or a permanent 0 or 1 under software control for gating purposes in other modules.
- **ERU_TOUTy** as combination of a peripheral trigger, a pattern detection result change event, or the ETLx trigger outputs TRxy to trigger actions in other modules.
- **ERU_IOUTy** as gated trigger output (ERU_GOUTy logical AND-combined with ERU_TOUTy) to trigger interrupts (e.g. the interrupt generation can be gated to allow interrupt activation during a certain time window).

Trigger Combination

The trigger combination logically OR-combines different trigger inputs to form a common trigger ERU_TOUTy. Possible trigger inputs are:

- In each ETLx unit of the **Input Channels**, the trigger output TRxy can be enabled and the trigger event can be directed to one of the OGUy units.
- In the case that at least one **pattern detection** input is enabled (IGCRm.IPENxy) and a change of the pattern detection result from pattern match to pattern miss (or vice-versa) is detected, a trigger event is generated to indicate a pattern detection result event (if enabled by IGCRm.GEENy).

The trigger combination offers the possibility to program different trigger criteria for several input signals (independently for each Input Channel) or peripheral signals, and to combine their effects to a single output, e.g. to generate an interrupt or to start e.g. an ADC conversion. This combination capability allows the generation of an interrupt per OGU that can be triggered by several inputs (multitude of request sources -> one reaction).

Pattern Detection

The pattern detection logic allows the combination of the status flags of all ETLx units. Each status flag can be individually included or excluded from the pattern detection for each OGUy, via control bits IGCRm.IPENxy. The pattern detection block outputs the following pattern detection results:

- **Pattern match** (PDRR.PDRy = 1 and ERU_PDOUTy = 1):
  A pattern match is indicated while all status flags that are included in the pattern detection are 1.
- **Pattern miss** (PDRR.PDRy = 0 and ERU_PDOUTy = 0):
  A pattern miss is indicated while at least one of the status flags that are included in the pattern detection is 0.

In addition, the pattern detection can deliver a trigger event if the pattern detection result changes from match to miss or vice-versa (if enabled by IGCRm.GEENy = 1). The pattern result change event is logically OR-combined with the other enabled trigger

events to support interrupt generation or to trigger other module functions (e.g. in an ADC). The event is indicated when the pattern detection result changes and PDRR.PDRy becomes updated.

The interrupt generation in the OGUy is based on the trigger ERU_TOUTy that can be gated (masked) with the pattern detection result ERU_PDOUTy. This allows an automatic and reproducible generation of interrupts during a certain time window, where the request event is elaborated by the trigger combination block and the time window information (gating) is given by the pattern detection. For example, interrupts can be issued on a regular time base while a combination of input signals occurs (pattern detection based on ETLx status bits).

A programmable gating scheme introduces flexibility to adapt to application requirements and allows the generation of interrupt requests ERU_IOUTy under different conditions:

- **Pattern match** (IGCRm.IGPy = 10_B):
  An interrupt request is issued when a trigger event occurs while the pattern detection shows a pattern match.
- **Pattern miss** (IGCRm.IGPy = 11_B):
  An interrupt request is issued when the trigger event occurs while the pattern detection shows a pattern miss.
- **Independent** of pattern detection (IGCRm.IGPy = 01_B):
  In this mode, each occurring trigger event leads to an interrupt request. The pattern detection output can be used independently from the trigger combination for gating purposes of other peripherals (independent use of ERU_TOUTy and ERU_PDOUTy with interrupt requests on trigger events).
- **No interrupts** (IGCRm.IGPy = 00_B, default setting)
  In this mode, an occurring trigger event does not lead to an interrupt request. The pattern detection output can be used independently from the trigger combination for gating purposes of other peripherals (independent use of ERU_TOUTy and ERU_PDOUTy without interrupt requests on trigger events).

3.3.2.7 ERU Output Connections

This section describes the connections of the ERU output signals for gating or triggering other module functions, as well as the connections to the interrupt control registers.

Table 3-12 ERU Output Connections in TC1797

Output	from/to Module	I/O to OGUy	Can be used to/as
OGU0 Outputs			
ERU_ PDOUT0	ERAY input STPW0	O	pattern detection output
ERU_ GOUT0	not connected	O	gated pattern detection output
ERU_ TOUT0	not connected	O	trigger output
ERU_ IOUT0	**Interrupt Generation** DMA channel 00 DMA channel 04	O	interrupt output
OGU1 Outputs			
ERU_ PDOUT1	ERAY input STPW1	O	pattern detection output
ERU_ GOUT1	not connected	O	gated pattern detection output
ERU_ TOUT1	not connected	O	trigger output
ERU_ IOUT1	**Interrupt Generation** DMA channel 01 DMA channel 05 GPTA0 input INT1 GPTA1 input INT1 LTCA2 input INT1	O	interrupt output
OGU2 Outputs			
ERU_ PDOUT2	ERAY input STPW2 ADC gating input FADC input FADC_GSC	O	pattern detection output
ERU_ GOUT2	not connected	O	gated pattern detection output
ERU_ TOUT2	not connected	O	trigger output

Table 3-12 ERU Output Connections in TC1797 (cont'd)

Output	from/to Module	I/O to OGUy	Can be used to/as
ERU_ IOUT2	**Interrupt Generation** DMA channel 02 DMA channel 06 ADC trigger input FADC input FADC_TSC GPTA0 input INT2 GPTA1 input INT2 LTCA2 input INT2	O	interrupt output
OGU3 Outputs			
ERU_ PDOUT3	ERAY input STPW3 ADC gating input FADC input FADC_GSD	O	pattern detection output
ERU_ GOUT3	not connected	O	gated pattern detection output
ERU_ TOUT3	not connected	O	trigger output
ERU_ IOUT3	**Interrupt Generation** DMA channel 03 DMA channel 07 ADC trigger input FADC input FADC_TSD GPTA0 input INT3 GPTA1 input INT3 LTCA2 input INT3	O	interrupt output

3.3.2.8 External Request Unit Registers

The External Input Channel Register EICR0 and EICR1 for the external input channels 0 to 3 contain bits to configure the external request selection ERS and the event trigger logic ETL.

EICR0

External Input Channel Register 0 (080_H) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**INP1**			**EI EN1**	**LD EN1**	**R EN1**	**F EN1**	**0**		**EXIS1**		**0**			
r	rw			rw	rw	rw	rw	r		rw		r			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**INP0**			**EI EN0**	**LD EN0**	**R EN0**	**F EN0**	**0**		**EXIS0**		**0**			
r	rw			rw	rw	rw	rw	r		rw		r			

Field	Bits	Type	Description
EXIS0	[5:4]	rw	**External Input Selection 0** This bit field determines which input line is selected for Input Channel 0. 00_B Input 00 is selected 01_B Input 01 is selected 10_B Input 02 is selected 11_B Input 03 is selected
FEN0	8	rw	**Falling Edge Enable 0** This bit determines if the falling edge of Input Channel 0 is used to set bit INTF0. 0_B The falling edge is not used 1_B The detection of a falling edge of Input Channel 0 generates a trigger event (INTF0 becomes set)
REN0	9	rw	**Rising Edge Enable 0** This bit determines if the rising edge of Input Channel 0 is used to set bit INTF0. 0_B The rising edge is not used 1_B The detection of a rising edge of Input Channel 0 generates a trigger event (INTF0 becomes set)

Field	Bits	Type	Description
LDEN0	10	rw	**Level Detection Enable 0** This bit determines if bit INTF0 is cleared automatically if an edge of the input Input Channel 0 is detected, which has not been selected (rising edge with REN0 = 0 or falling edge with FEN0 = 0). 0_B Bit INTF0 will not be cleared 1_B Bit INTF0 will be cleared
EIEN0	11	rw	**External Input Enable 0** This bit enables the generation of a trigger event for request channel 0 (e.g. for interrupt generation) when a selected edge is detected. 0_B The trigger event is disabled 1_B The trigger event is enabled
INP0	[14:12]	rw	**Input Node Pointer** This bit field determines the destination (output channel) for trigger event 0 (if enabled by EIEN0). 000_B The event of input channel 0 triggers output channel 0 (signal INT00) 001_B The event of input channel 0 triggers output channel 1 (signal INT01) 010_B The event of input channel 0 triggers output channel 2 (signal INT02) 011_B The event of input channel 0 triggers output channel 3 (signal INT03) 100_B Reserved, do not use this combination 101_B Reserved, do not use this combination 110_B Reserved, do not use this combination 111_B Reserved, do not use this combination
EXIS1	[21:20]	rw	**External Input Selection 1** This bit field determines which input line is selected for Input Channel 1. 00_B Input 10 is selected 01_B Input 11 is selected 10_B Input 12 is selected 11_B Input 13 is selected

Field	Bits	Type	Description
FEN1	24	rw	**Falling Edge Enable 1** This bit determines if the falling edge of Input Channel 1 is used to set bit INTF1. 0_B The falling edge is not used 1_B The detection of a falling edge of Input Channel 1 generates a trigger event (INTF1 becomes set)
REN1	25	rw	**Rising Edge Enable 1** This bit determines if the rising edge of Input Channel 1 is used to set bit INTF1. 0_B The rising edge is not used 1_B The detection of a rising edge of Input Channel 1 generates a trigger event (INTF1 becomes set)
LDEN1	26	rw	**Level Detection Enable 1** This bit determines if bit INTF1 is cleared automatically if an edge of the input Input Channel 1 is detected, which has not been selected (rising edge with REN1 = 0 or falling edge with FEN1 = 0). 0_B Bit INTF1 will not be cleared 1_B Bit INTF1 will be cleared
EIEN1	27	rw	**External Input Enable 1** This bit enables the generation of a trigger event for request channel 1 (e.g. for interrupt generation) when a selected edge is detected. 0_B The trigger event is disabled 1_B The trigger event is enabled

Field	Bits	Type	Description
INP1	[30:28]	rw	**Input Node Pointer** This bit field determines the destination (output channel) for trigger event 1 (if enabled by EIEN1). 000_B The event of input channel 1 triggers output channel 0 (signal INT10) 001_B The event of input channel 1 triggers output channel 1 (signal INT11) 010_B The event of input channel 1 triggers output channel 2 (signal INT12) 011_B The event of input channel 1 triggers output channel 3 (signal INT13) 100_B Reserved, do not use this combination 101_B Reserved, do not use this combination 110_B Reserved, do not use this combination 111_B Reserved, do not use this combination
0	[3:0], [7:6], [19:15], [23:22], 31	r	**Reserved** Read as 0; should be written with 0.

EICR1
External Input Channel Register 1 (084_H) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		**INP3**		**EI EN3**	**LD EN3**	**R EN3**	**F EN3**	**0**		**EXIS3**		**0**			
r		rw		rw	rw	rw	rw	r		rw		r			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		**INP2**		**EI EN2**	**LD EN2**	**R EN2**	**F EN2**	**0**		**EXIS2**		**0**			
r		rw		rw	rw	rw	rw	r		rw		r			

Field	Bits	Type	Description
EXIS2	[5:4]	rw	**External Input Selection 2** This bit field determines which input line is selected for Input Channel 2. 00_B Input 20 is selected 01_B Input 21 is selected 10_B Input 22 is selected 11_B Input 23 is selected
FEN2	8	rw	**Falling Edge Enable 2** This bit determines if the falling edge of Input Channel 2 is used to set bit INTF2. 0_B The falling edge is not used 1_B The detection of a falling edge of Input Channel 2 generates a trigger event (INTF3 becomes set)
REN2	9	rw	**Rising Edge Enable 2** This bit determines if the rising edge of signal Input Channel 2 is used to set bit INTF2. 0_B The rising edge is not used 1_B The detection of a falling edge of Input Channel 2 generates a trigger event (INTF2 becomes set)
LDEN2	10	rw	**Level Detection Enable 2** This bit determines if bit INTF2 is cleared automatically if an edge of the input Input Channel 2 is detected, which has not been selected (rising edge with REN2 = 0 or falling edge with FEN2 = 0). 0_B Bit INTF2 will not be cleared 1_B Bit INTF2 will be cleared
EIEN2	11	rw	**External Input Enable 2** This bit enables the generation of a trigger event for request channel 2 (e.g. for interrupt generation) when a selected edge is detected. 0_B The trigger event is disabled 1_B The trigger event is enabled

Field	Bits	Type	Description
INP2	[14:12]	rw	**Input Node Pointer** This bit field determines the destination (output channel) for trigger event 2 (if enabled by EIEN2). 000_B The event of input channel 2 triggers output channel 0 (signal INT20) 001_B The event of input channel 2 triggers output channel 1 (signal INT21) 010_B The event of input channel 2 triggers output channel 2 (signal INT22) 011_B The event of input channel 2 triggers output channel 3 (signal INT23) 100_B Reserved, do not use this combination 101_B Reserved, do not use this combination 110_B Reserved, do not use this combination 111_B Reserved, do not use this combination
EXIS3	[21:20]	rw	**External Input Selection 3** This bit field determines which input line is selected for Input Channel 3. 00_B Input 30 is selected 01_B Input 31 is selected 10_B Input 32 is selected 11_B Input 33 is selected
FEN3	24	rw	**Falling Edge Enable 3** This bit determines if the falling edge of Input Channel 3 is used to set bit INTF3. 0_B The falling edge is not used 1_B The detection of a falling edge of Input Channel 3 generates a trigger event (INTF3 becomes set)
REN3	25	rw	**Rising Edge Enable 3** This bit determines if the rising edge of signal Input Channel 3 is used to set bit INTF3. 0_B The rising edge is not used 1_B The detection of a falling edge of Input Channel 3 generates a trigger event (INTF3 becomes set)

Field	Bits	Type	Description
LDEN3	26	rw	**Level Detection Enable 3** This bit determines if bit INTF3 is cleared automatically if an edge of the input Input Channel 3 is detected, which has not been selected (rising edge with REN3 = 0 or falling edge with FEN3 = 0). 0_B Bit INTF3 will not be cleared 1_B Bit INTF3 will be cleared
EIEN3	27	rw	**External Interrupt Enable 3** This bit enables the generation of a trigger event for request channel 3 (e.g. for interrupt generation) when a selected edge is detected. 0_B The trigger event is disabled 1_B The trigger event is enabled
INP3	[30:28]	rw	**Interrupt Node Pointer** This bit field determines the destination (output channel) for trigger event 3 (if enabled by EIEN3). 000_B The event of input channel 3 triggers output channel 0 (signal INT30) 001_B The event of input channel 3 triggers output channel 1 (signal INT31) 010_B The event of input channel 3 triggers output channel 2 (signal INT32) 011_B The event of input channel 3 triggers output channel 3 (signal INT33) 100_B Reserved, do not use this combination 101_B Reserved, do not use this combination 110_B Reserved, do not use this combination 111_B Reserved, do not use this combination
0	[3:0], [7:6], [19:15], [23:22], 31	r	**Reserved** Read as 0; should be written with 0.

The External Input Flag Register EIFR contains all status flags for the external input channels. The bits in this register can be cleared by software by setting FMR.FCx, and set by setting FMR.FSx.

EIFR
External Input Flag Register **(088_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0												INTF3	INTF2	INTF1	INTF0
r												rh	rh	rh	rh

Field	Bits	Type	Description
INTFx (x = 0-3)	x	rh	**External Interrupt Flag of Channel x** This bit monitors the status flag of the event trigger condition for the input channel x. This bit is automatically cleared when the selected condition (see RENx, FENx) is no longer met (if LDENx = 1) or remains set until it is cleared by software (if LDENx = 0).
0	[31:4]	r	**Reserved** Read as 0; should be written with 0.

FMR
Flag Modification Register **($08C_H$)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0												FC3	FC2	FC1	FC0
r												w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0												FS3	FS2	FS1	FS0
r												w	w	w	w

Field	Bits	Type	Description
FSx (x = 0-3)	x	w	**Set Flag INTFx for Channel x** Setting this bit will set the corresponding bit INTFx in register EIFR. Reading this bit always delivers a 0. 0_B The bit x in register EIFR is not modified 1_B The bit x in register EIFR is set
FCx (x = 0-3)	16 + x	w	**Clear Flag INTFx for Channel x** Setting this bit will clear the corresponding bit INTFx in register EIFR. Reading this bit always delivers a 0. 0_B The bit x in register EIFR is not modified 1_B The bit x in register EIFR is cleared
0	[15:4], [31:20]	r	**Reserved** Read as 0; should be written with 0.

The Pattern Detection Result Register monitors the combinatorial output status of the pattern detection units.

PDRR
Pattern Detection Result Register (090_H) Reset Value: 0000 000F$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0												PDR 3	PDR 2	PDR 1	PDR 0
r												rh	rh	rh	rh

Field	Bits	Type	Description
PDRy (y = 0-3)	y	rh	**Pattern Detection Result of Channel y** This bit monitors the output status of the pattern detection for the output channel y.
0	[31:4]	r	**Reserved** Read as 0; should be written with 0.

The Interrupt Gating Control Registers IGCR0 and IGCR1 contain bits to enable the pattern detection and to control the gating for output channel 0 to 3.

IGCR0
Interrupt Gating Register 0 (094_H) Reset Value: 0000 0000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IGP1		GEEN1	0									IPEN13	IPEN12	IPEN11	IPEN10
rw		rw	r									rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IGP0		GEEN0	0									IPEN03	IPEN02	IPEN01	IPEN00
rw		rw	r									rw	rw	rw	rw

Field	Bits	Type	Description
IPEN0x (x = 0-3)	x	rw	**Interrupt Pattern Enable for Channel 0** Bit IPEN0x determines if the flag INTFx of channel x takes part in the pattern detection for the gating of the requests for the output signals GOUTy and IOUTy. 0_B The bit INTFx does not take part in the pattern detection 1_B The bit INTFx is taken into consideration for the pattern detection
GEEN0	13	rw	**Generate Event Enable 0** Bit GEEN0 enables the generation of a trigger event for output channel 0 when the result of the pattern detection changes. When using this feature, a trigger (e.g. for an interrupt) is generated during the first clock cycle when a pattern is detected or when it is no longer detected. 0_B The trigger generation at a change of the pattern detection result is disabled 1_B The trigger generation at a change of the pattern detection result is enabled

Field	Bits	Type	Description
IGP0	[15:14]	rw	**Interrupt Gating Pattern 0** Bit field IGP0 determines how the pattern detection influences the output lines GOUT0 and IOUT0. 00_B The detected pattern is not taken into account. An activation of IOUT0 is always possible due to a trigger event. 01_B The detected pattern is not taken into account. An activation of IOUT0 is not possible. 10_B The detected pattern is taken into account. An activation of IOUT0 is only possible due to a trigger event while the pattern is detected. 11_B The detected pattern is taken into account. An activation of IOUT0 is only possible due to a trigger event while the pattern is not detected.
IPEN1x (x = 0-3)	16+x	rw	**Interrupt Pattern Enable for Channel 1** Bit IPEN1x determines if the flag INTFx of channel x takes part in the pattern detection for the gating of the requests for the output signals GOUTy and IOUTy. 0_B The bit INTFx does not take part in the pattern detection 1_B The bit INTFx is taken into consideration for the pattern detection
GEEN1	29	rw	**Generate Event Enable 1** Bit GEEN1 enables the generation of a trigger event for output channel 1 when the result of the pattern detection changes. When using this feature, a trigger (e.g. for an interrupt) is generated during the first clock cycle when a pattern is detected, or when it is no longer detected. 0_B The trigger generation at a change of the pattern detection result is disabled 1_B The trigger generation at a change of the pattern detection result is enabled

Field	Bits	Type	Description
IGP1	[31:30]	rw	**Interrupt Gating Pattern 1** Bit field IGP1 determines how the pattern detection influences the output lines GOUT1 and IOUT1. 00_B The detected pattern is not taken into account. An activation of IOUT1 is always possible due to a trigger event. 01_B The detected pattern is not taken into account. An activation of IOUT1 is not possible. 10_B The detected pattern is taken into account. An activation of IOUT1 is only possible due to a trigger event while the pattern is detected. 11_B The detected pattern is taken into account. An activation of IOUT1 is only possible due to a trigger event while the pattern is not detected.
0	[12:4], [28:20]	r	**Reserved** Read as 0; should be written with 0.

IGCR1

Interrupt Gating Register 1 (098_H) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IGP3		GEEN3	0									IPEN33	IPEN32	IPEN31	IPEN30
rw		rw	r									rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IGP2		GEEN2	0									IPEN23	IPEN22	IPEN21	IPEN20
rw		rw	r									rw	rw	rw	rw

Field	Bits	Type	Description
IPEN2x (x = 0-3)	x	rw	**Interrupt Pattern Enable for Channel 2** Bit IPEN2x determines if the flag INTFx of channel x takes part in the pattern detection for the gating of the requests for the output signals GOUTy and IOUTy. 0_B The bit INTFx does not take part in the pattern detection 1_B The bit INTFx is taken into consideration for the pattern detection
GEEN2	13	rw	**Generate Event Enable 2** Bit GEEN2 enables the generation of a trigger event for output channel 2 when the result of the pattern detection changes. When using this feature, a trigger (e.g. for an interrupt) is generated during the first clock cycle when a pattern is detected, or when it is no longer detected. 0_B The trigger generation at a change of the pattern detection result is disabled 1_B The trigger generation at a change of the pattern detection result is enabled
IGP2	[15:14]	rw	**Interrupt Gating Pattern 2** Bit field IGP2 determines how the pattern detection influences the output lines GOUT2 and IOUT2. 00_B The detected pattern is not taken into account. An activation of IOUT2 is always possible due to a trigger event. 01_B The detected pattern is not taken into account. An activation of IOUT2 is not possible. 10_B The detected pattern is taken into account. An activation of IOUT2 is only possible due to a trigger event while the pattern is detected. 11_B The detected pattern is taken into account. An activation of IOUT2 is only possible due to a trigger event while the pattern is not detected.

Field	Bits	Type	Description
IPEN3x (x = 0-3)	16+x	rw	**Interrupt Pattern Enable for Channel 3** Bit IPEN3x determines if the flag INTFx of channel x takes part in the pattern detection for the gating of the requests for the output signals GOUTy and IOUTy. 0_B The bit INTFx does not take part in the pattern detection 1_B The bit INTFx is taken into consideration for the pattern detection
GEEN3	29	rw	**Generate Event Enable 3** Bit GEEN3 enables the generation of a trigger event for output channel 3 when the result of the pattern detection changes. When using this feature, a trigger (e.g. for an interrupt) is generated during the first clock cycle when a pattern is detected, or when it is no longer detected. 0_B The trigger generation at a change of the pattern detection result is disabled 1_B The trigger generation at a change of the pattern detection result is enabled
IGP3	[31:30]	rw	**Interrupt Gating Pattern 3** Bit field IGP3 determines how the pattern detection influences the output lines GOUT3 and IOUT3. 00_B The detected pattern is not taken into account. An activation of IOUT3 is always possible due to a trigger event. 01_B The detected pattern is not taken into account. An activation of IOUT3 is not possible. 10_B The detected pattern is taken into account. An activation of IOUT3 is only possible due to a trigger event while the pattern is detected. 11_B The detected pattern is taken into account. An activation of IOUT3 is only possible due to a trigger event while the pattern is not detected.
0	[12:4], [28:20]	r	**Reserved** Read as 0; should be written with 0.

3.4 Power Management

This section describes the power management system of the TC1797. Topics covered here include the internal system interfaces, external interfaces, and the operations of the CPU and peripherals.

3.4.1 Power Management Overview

The TC1797 power-management system allows software to configure the various processing units so that they automatically adjust to draw the minimum necessary power for the application.

As shown in **Table 3-13**, there are three power management modes available:

- Run Mode
- Idle Mode
- Sleep Mode

Table 3-13 Power Management Mode Summary

Mode	Description
Run Mode	The system is fully operational. CPU and peripherals are enabled, as determined by software.
Idle Mode	The CPU is disabled, waiting for a condition to return it to Run Mode. Idle Mode can be entered by software when the processor has no active tasks to perform. Processor memory is accessible to peripherals. An Application Reset, Watchdog Timer event, an NMI trap, or any interrupt for the CPU will return the system to Run Mode.
Sleep Mode	Only those peripherals programmed to operate in Sleep Mode are active. The other peripheral module will be shut down. Interrupts for the CPU, a Watchdog Timer event, an NMI trap, or an Application Reset will return the system to Run Mode. Entering this state requires an orderly shut-down controlled by the Power Management State Machine.

The power-management modes provide flexible reduction of power consumption through a combination of techniques, including:

- Stopping the CPU
- Stopping other system components individually
- Clock-speed reduction of some peripheral components individually

The Power Management controls the power mode of all system components during Run Mode, Idle Mode, and Sleep Mode. This flexibility in power management provides minimum power consumption for any application.

In typical operation, Idle Mode and Sleep Mode may be entered and exited frequently during the run time of an application. For example, system software will typically cause

the CPU to enter Idle Mode each time it has to wait for an interrupt before continuing its tasks. In Sleep Mode and Idle Mode, wake-up is performed automatically when any interrupt is detected or if an NMI trap is activated.

3.4.2 Power Management Modes

This section describes in more detail the power management modes, their operations, and how power management modes are entered and exited. It also describes the behavior of TC1797 system components in all power management modes.

3.4.2.1 Idle Mode

The Idle Mode is requested by software when writing to register PMCSR.REQSLP = 01_B. The CPU finishes its current operation, sends an acknowledge to the Power Management, and then enters an inactive state in which the CPU and the DMI and PMI memory units are shut off.

Other system components that are able to write to register PMCSR can also request the Idle Mode. For example, the PCP or DMA controller can request Idle Mode by writing to the PMCSR register.

During Idle Mode, memory accesses to the DMI and PMI cause these units to awaken automatically to handle the transactions. When memory transactions are complete, the DMI and PMI return to Idle Mode again.

The system will return to Run Mode through the occurrence of any of the following conditions:

- An interrupt is received from an interrupt source of the CPU
- An NMI trap request is received
- An Application Reset is generated

If any of these conditions arise, the TC1797 immediately awakens and returns to Run Mode. If it is awakened by a reset, the TC1797 system begins its reset sequence. If it is awakened by a Watchdog Timer overflow event, it executes the instruction following the one that was last executed before Idle Mode was entered. If it is awakened by an NMI or interrupt, the CPU will immediately vector to the appropriate handler.

3.4.2.2 Sleep Mode

The Sleep Mode is requested by software when writing to register PMCSR.REQSLP = 10_B.

Entering Sleep Mode

Sleep Mode is entered in two steps:

1. The CPU is put into Idle Mode as described in the previous section. When the Power Management receives the Idle acknowledge back from the CPU, it proceeds with the second step.
2. Each FPI Bus unit is requested to enter the Sleep Mode. The response of each FPI Bus unit to the sleep request is determined by its clock control register. These clock control registers must have been previously configured by software.

TC1797 State During Sleep Mode

Sleep Mode is disabled for a unit if MOD_CLC.EDIS is set. The sleep request is ignored in this case and the corresponding unit continues normal operation. If MOD_CLC.EDIS is cleared, Sleep Mode is enabled for this unit and the unit enters Sleep Mode. Two actions then occur:

- The unit finishes whatever bus transaction was in progress when the signal was received
- The unit functions are suspended

Depending on bit MOD_CLC.FSOE, the module is either immediately stopped (MOD_CLC.FSOE = 1), or the unit is allowed to finish ongoing operations (MOD_CLC.FSOE = 0) before the Sleep Mode is entered. For example, setting MOD_CLC.FSOE to 1 for a serial port will stop all actions in the serial port immediately when the sleep request is received. Ongoing transmissions or receptions will be aborted. If MOD_CLC.FSOE is cleared, ongoing transmissions or receptions will be completed, before Sleep Mode is entered. The purpose of setting MOD_CLC.FSOE = 1 is to allow a debugger to observe the internal state of a peripheral unit immediately.

Exiting Sleep Mode

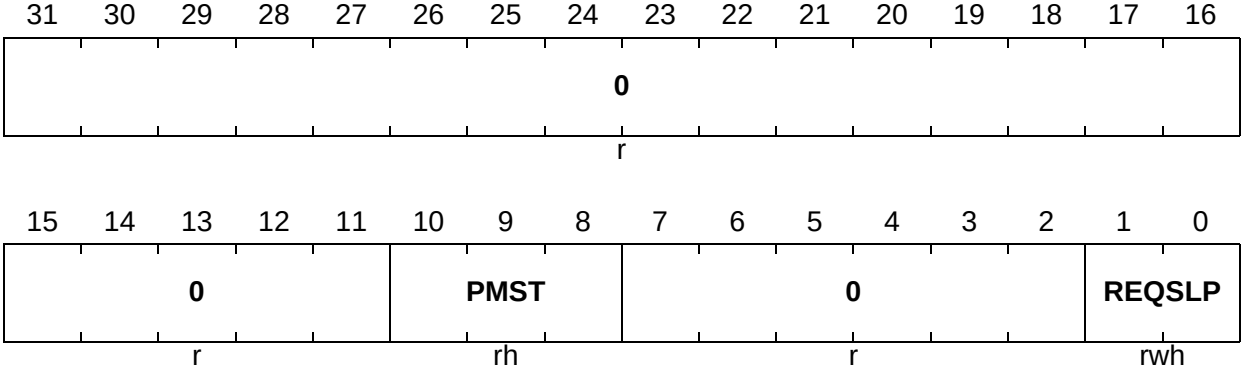
The system will be returned to Run Mode by the same events that exit Idle Mode. The response of the CPU to being awakened is also the same as for Idle Mode. Peripheral units that have entered Sleep Mode will switch back to their selected Run Mode operation.

3.4.3 Power Management Control and Status Register, PMCSR

The set of registers used for power management is divided between central TC1797 components and peripheral components. The PMCSR register provides software control

and status information for the central part. There are individual clock control registers for peripheral components because the Sleep Mode behavior of each peripheral component is programmable. When entering Idle Mode and Sleep Mode, the Power Management directly controls TC1797 components such as the CPU, but indirectly controls peripheral components through their clock control registers.

PMCSR
Power Management Control and Status Register
($0B0_H$) **Reset Value: $0000\ 0100_H$**



Field	Bits	Type	Function
REQSLP	[1:0]	rwh	**Idle Mode and Sleep Mode Request** 00_B Normal Run Mode 01_B Request Idle Mode 10_B Request Sleep Mode 11_B Reserved; do not use this combination In Idle Mode or Sleep Mode, these bits are cleared in response to an interrupt for the CPU, or when bit 15 of the Watchdog Timer count register (the WDT_SR.TIM[15] bit) changes from 0 to 1.
PMST	[10:8]	rh	**Power Management Status** 000_B Waiting for PLL lock condition 001_B Normal Run Mode 010_B Idle Mode requested 011_B Idle Mode acknowledged 100_B Sleep Mode 101_B Reserved, do not use this combination 110_B Reserved, do not use this combination 111_B Reserved, do not use this combination

Field	Bits	Type	Function
0	[7:2], [31:11]	r	**Reserved** Read as 0; should be written with 0.

3.5 Software Boot Support

In order to determine the correct starting point of operation for the software a minimum of hardware support is required. As much as possible is done via software. Some decisions have to be done in hardware because they must be known before any software is operational.

For a startup operation there are two general cases that have to be handled:

- Differentiation between Test Mode and Normal Mode for each Power-on Reset event (see **Section 3.5.1**)
- Configuration of the boot option for each Application Reset event (see **Section 3.5.2**)

3.5.1 Configuration done with Start-up

With the device power-on some basic operating mode selection has to be made. The first decision that has to be made is if the device should operate in Test Mode or in Normal (Customer) Mode. The Test Mode is only for Infineon internal usage not for any customer and has nothing to do with debugging.

If the Normal Mode was selected the next decision is which debug interface type issued for debugging for this session (until the next power-on event).

Table 3-14 Normal Mode / Test Mode Input Selection

Field	Description
TESTL	**Latched TEST Signal** 0 A Test Mode can be selected 1 Normal Mode is selected
TRSTL	**Latched TRST Signal** 0 The JTAG interface is active 1 The DAP interface is active

After these two decisions were made the detailed decision has to be made to define the real startup configuration. Most is made via the software and can be supported by some hardware selections depending on the startup configuration that should be selected.

3.5.2 Start-up Configuration Options

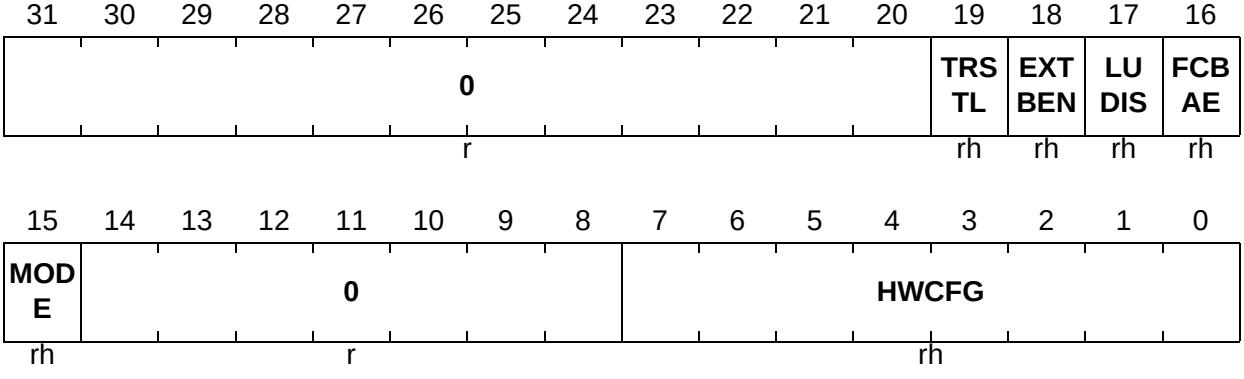
For the support of the start-up pins P0.0 to P0.7 are latched with the rising edge of the Application Reset and stored in register STSTAT.HWCFG. The update of bit field STSTAT.HWCFG with the latched value is only done if bit STSTAT.LUDIS is cleared. If bit STSTAT.LUDIS is set the value of STSTAT.HWCFG is not updated.

3.5.3 Start-up Registers

3.5.3.1 Start-up Status Register

Register STSTAT contains the information required by the boot software to identify the different start-up settings that can be selected.

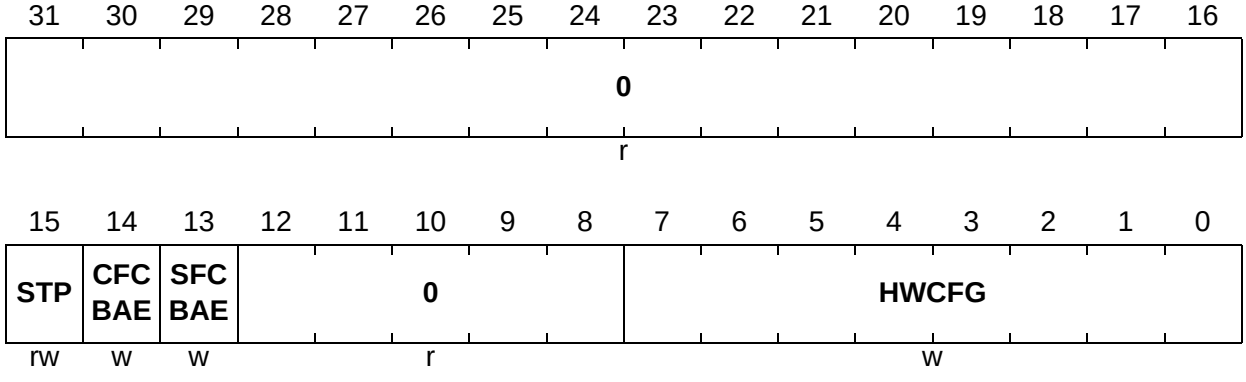
STSTAT
Start-up Status Register **($0C0_H$)** **Reset Value: $0000\ 8000_H$**



Field	Bits	Type	Description
HWCFG	[7:0]	rh	**Hardware Configuration Setting** This bit field contains the value that is used by the boot software. This bit field is updated in case of an Application Reset with the content by register SWRSTCON.SWCFG if bit SWRSTCON.SWBOOT AND RSTSTAT.SW are set. This bit field is updated in case of an Application Reset with the content of the latches of P0.0 to P.0.7 if bit SWRSTCON.SWBOOT OR RSTSTAT.SW are cleared and bit STSTAT.LUDIS is cleared. This bit field is left unchanged in case of an Application Reset and is not updated with the content of the latches of P0.0 to P.0.7 if bit SWRSTCON.SWBOOT OR RSTSTAT.SW are cleared and bit STSTAT.LUDIS is set. This bit field is updated with the value written to bit field STCON.HWCFG on a software write action.

Field	Bits	Type	Description
Mode	15	rh	**Mode** This bit indicates if the Test Mode is entered or not. 0_B A Test Mode can be selected 1_B Normal Mode is selected
FCBAE	16	rh	**Flash Config. Sector Access Enable** 0_B Flash config sector is not accessible. Instead the flash memory area is accessed. 1_B Flash config sector is accessible. The flash memory area can not be accessed. This bit can be cleared by setting bit STCON.CFCBAE. This bit can be set by setting bit STCON.SFCBAE.
LUDIS	17	rh	**Latch Update Disable** 0_B Bit field STSTAT.HWCFG is automatically updated with the latched value of pins P0.0 to P0.7 1_B Bit field STSTAT.HWCFG is not updated with the latched value of pins P0.0 to P0.7 This bit can be set by setting bit SYSCON.SETLUDIS.
EXTBEN	18	rh	**External Boot Enable** 0_B No Boot Configuration Value is fetched by the EBU 1_B A Boot Configuration Value is fetched by the EBU This bit can be set by setting bit SYSCON.SETEXTBEN.
TRSTL	19	rh	**TRSTL Status** This bit simply displays the value of TRSTL.
0	[14:8], [31:20]	r	**Reserved** Read as 0; should be written with 0.

STCON
Start-up Configuration Register ($0C4_H$) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
HWCFG	[7:0]	w	**Hardware Configuration Setting** Writing to this bit field updates bit field STSTAT.HWCFG. Reading this bit field returns zero.
SFCBAE	13	w	**Set Flash Config. Sector Access Enable** Setting this bit sets bit STSTAT.FCBAE. Reading this bit returns always a zero. *Note: This bit may not be set in parallel with bit CFCBAE.*
CFCBAE	14	w	**Clear Flash Config. Sector Access Enable** Setting this bit clears bit STCON.FCBAE. Reading this bit returns always a zero. *Note: This bit may not be set in parallel with bit SFCBAE.*
STP	15	rw	**Start-up Protection Setting** 0_B Start-up code is executed. Start-up protection is disabled. 1_B Start-up code protection is active This bit is also cleared by an Application Reset.
0	[12:8], [31:16]	r	**Reserved** Read as 0; should be written with 0.

3.6 SRAM Parity Control

In TC1797, several on-chip memory blocks are equipped with a parity error detection logic. This logic asserts a parity error signal when a parity error is detected in the related memory block. An active parity error signal sets a parity error flag, PFLx. If enabled by the specific parity enable control bit PENx, a NMI trap can be generated.

Figure 3-25 shows the functionality of the SRAM parity error control in the SCU.

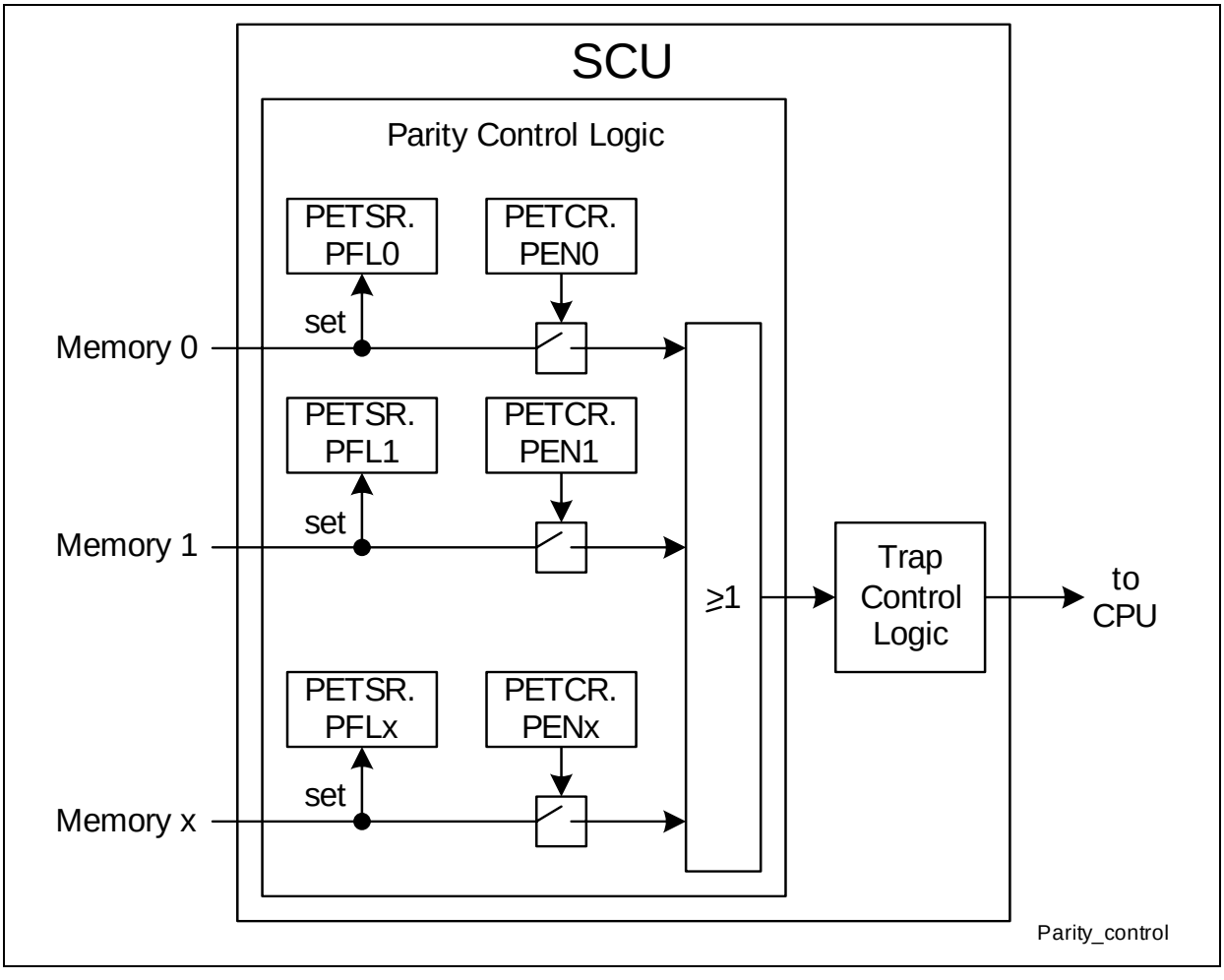
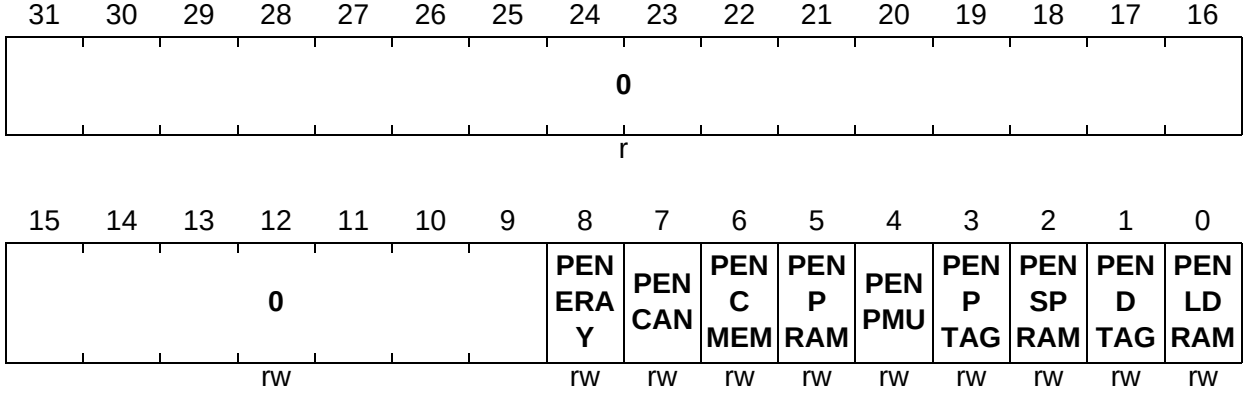


Figure 3-25 Control of SRAM Parity Error Detection

3.6.1 Parity Error Trap Registers

PETCR
Parity Error Trap Control Register (0D0$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
PENLDRAM	0	rw	**Parity Error Trap Enable for LDRAM and DCACHE Memory** This bit determine whether a trap is requested if an uncorrected parity error is detected in the LDRAM / DCACHE memory. 0_B No parity error trap trigger is requested 1_B A parity error trap trigger is requested
PENDTAG	1	rw	**Parity Error Trap Enable for Data Cache TAG RAM Memory** This bit determine whether a trap is requested if an uncorrected parity error is detected in the data cache TAG RAM memory. 0_B No parity error trap trigger is requested 1_B A parity error trap trigger is requested
PENSPRAM	2	rw	**Parity Error Trap Enable for SPRAM and ICACHE Memory** This bit determine whether a trap is requested if an uncorrected parity error is detected in the SPRAM / ICACHE memory. 0_B No parity error trap trigger is requested 1_B A parity error trap trigger is requested

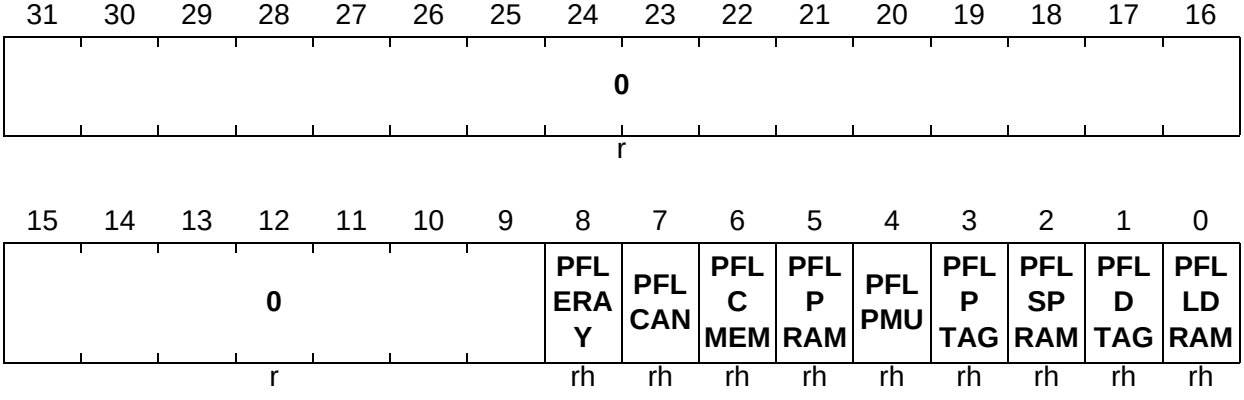
Field	Bits	Type	Description
PENPTAG	3	rw	**Parity Error Trap Enable for Program Cache TAG RAM Memory** This bit determine whether a trap is requested if an uncorrected parity error is detected in the program cache TAG RAM memory. 0_B No parity error trap trigger is requested 1_B A parity error trap trigger is requested
PENPMU	4	rw	**Parity Error Trap Enable for PMU Memory** This bit determine whether a trap is requested if an uncorrected parity error is detected in the PMU memory. 0_B No parity error trap trigger is requested 1_B A parity error trap trigger is requested
PENPRAM	5	rw	**Parity Error Trap Enable for Parameter RAM Memory** This bit determine whether a trap is requested if an uncorrected parity error is detected in the parameter RAM memory. 0_B No parity error trap trigger is requested 1_B A parity error trap trigger is requested
PENCMEM	6	rw	**Parity Error Trap Enable for Code Memory** This bit determine whether a trap is requested if an uncorrected parity error is detected in the code memory. 0_B No parity error trap trigger is requested 1_B A parity error trap trigger is requested
PENCAN	7	rw	**Parity Error Trap Enable for CAN Memory** This bit determine whether a trap is requested if an uncorrected parity error is detected in the CAN memory. 0_B No parity error trap trigger is requested 1_B A parity error trap trigger is requested
PENERAY	8	rw	**Parity Error Trap Enable for ERAY Memory** This bit determine whether a trap is requested if an uncorrected parity error is detected in the ERAY memory. 0_B No parity error trap trigger is requested 1_B A parity error trap trigger is requested

Field	Bits	Type	Description
0	[15:9]	rw	**Reserved** Read as 0; should be written with 0.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

Note: Register PETCR is Endinit-protected for write operations.

PETSR

Parity Error Trap Status Register (0D4$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
PFLLDRAM	0	rh	**Parity Error Flag for LDRAM and DCACHE Memory** This bit indicate whether an uncorrected parity error has been detected in the LDRAM / DCACHE memory. 0_B No parity error detected. 1_B Parity error is detected. The PFLx bits are cleared by hardware after a read access.
PFLDTAG	1	rh	**Parity Error Flag for Data Cache TAG RAM Memory** This bit indicate whether an uncorrected parity error has been detected in the data TAG RAM memory. 0_B No parity error detected. 1_B Parity error is detected. The PFLx bits are cleared by hardware after a read access.
PFLSPRAM	2	rh	**Parity Error Flag for SPRAM and ICACHE Memory** This bit indicate whether an uncorrected parity error has been detected in the SPRAM / ICACHE memory. 0_B No parity error detected. 1_B Parity error is detected. The PFLx bits are cleared by hardware after a read access.

Field	Bits	Type	Description
PFLPTAG	3	rh	**Parity Error Flag for Program Cache TAG RAM Memory** This bit indicate whether an uncorrected parity error has been detected in the program TAG RAM memory. 0_B No parity error detected. 1_B Parity error is detected. The PFLx bits are cleared by hardware after a read access.
PFLPMU	4	rh	**Parity Error Flag for PMU Memory** This bit indicate whether an uncorrected parity error has been detected in the PMU memory. 0_B No parity error detected. 1_B Parity error is detected. The PFLx bits are cleared by hardware after a read access.
PFLPRAM	5	rh	**Parity Error Flag for Parameter RAM Memory** This bit indicate whether an uncorrected parity error has been detected in the Parameter RAM memory. 0_B No parity error detected. 1_B Parity error is detected. The PFLx bits are cleared by hardware after a read access.
PFLCMEM	6	rh	**Parity Error Flag for Code Memory** This bit indicate whether an uncorrected parity error has been detected in the code memory. 0_B No parity error detected. 1_B Parity error is detected. The PFLx bits are cleared by hardware after a read access.
PFLCAN	7	rh	**Parity Error Flag for CAN Memory** This bit indicate whether an uncorrected parity error has been detected in the CAN memory. 0_B No parity error detected. 1_B Parity error is detected. The PFLx bits are cleared by hardware after a read access.

Field	Bits	Type	Description
PFLERAY	8	rh	**Parity Error Flag for ERAY Memory** This bit indicate whether an uncorrected parity error has been detected in the ERAY memory. 0_B No parity error detected. 1_B Parity error is detected. The PFLx bits are cleared by hardware after a read access.
0	[31:9]	r	**Reserved** Read as 0.

Note: PETSR is a read-only register. Writing to PETSR results in a bus error.

3.7 Die Temperature Measurement

The Die Temperature Sensor (DTS) generates a measurement result that indicates directly the current temperature. The result of the measurement is displayed via bit field DTSSTAT.RESULT. In order to start one measurement bit DTSCON.START needs to be set.

The DTS has to be enabled before it can be used via bit DTSCON.PWD. When the DTS is powered after the start-up time of the DTS (defined in the Data Sheet) a temperature measurement can be started.

Note: If bit field DTSSTAT.RESULT is read before the first measurement was finished it will return 0x000.

When a measurement is started the result is available after the measurement time passed. If the DTS is ready to start a measurement can be checked via bit DTSSTAT.RDY. If a started measurement is finished or still in progress is indicated via the status bit DTSSTAT.BUSY. The measurement time is also defined in the Data Sheet.

In order to adjust production variations bit field DTSCON.CAL should be programmed with a predefined value. The value is located at address 0xD000000E for the 7 LSB of bit field DTSCON.CAL (DTS_CON[10:4]). The 5 MSB of bit field DTSCON.CAL (DTS_CON[15:11]) have to be written with zeros.

Note: The first measurement after the DTS was powered delivers a result without calibration adjustment and should be ignored therefore.

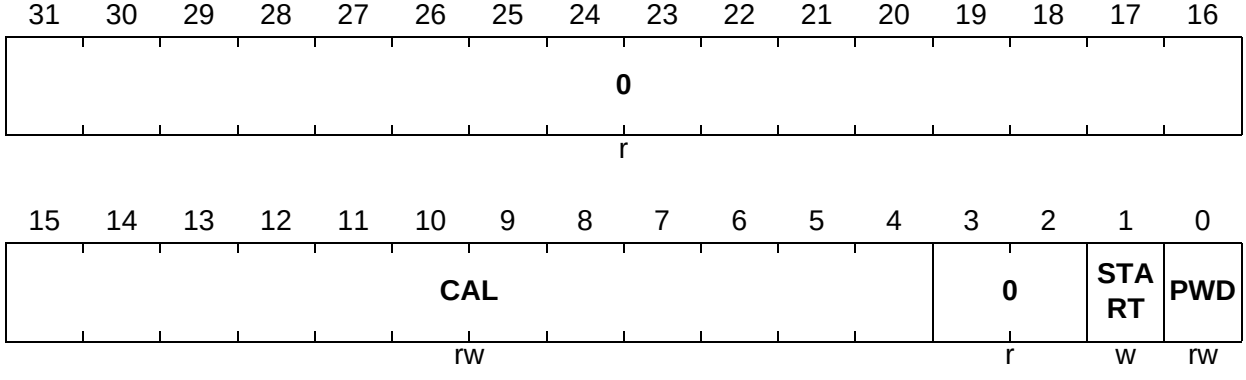
The formula to calculate the die temperature is defined in the Data Sheet.

Note: The maximum resolution is only achieved for a measurement that is part of multiple continuos measurements.

3.7.1 Die Temperature Sensor Register

DTSCON

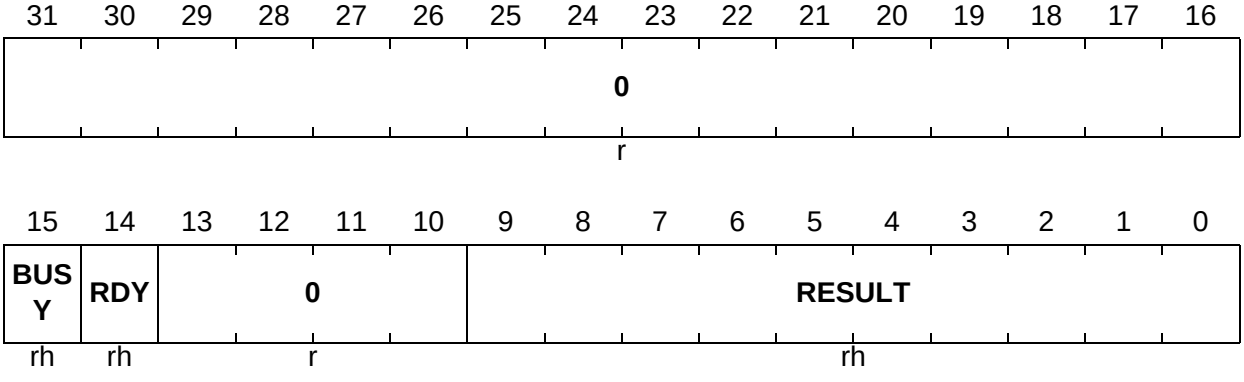
Die Temperature Sensor Control Register(0E4$_H$) **Reset Value: 0000 0001$_H$**



Field	Bits	Type	Description
PWD	0	rw	**Sensor Power Down** This bit defines the DTS power state. 0_B The DTS is powered 1_B The DTS is not powered
START	1	w	**Sensor Measurement Start** This bit starts a measurement of the DTS. 0_B No DTS measurement is started 1_B A DTS measurement is started If set this bit is automatically cleared. This bit always reads as zero.
CAL	[15:4]	rw	**Calibration Value** This bit field interfaces the calibration values to the DTS.
0	[3:2], [31:16]	r	**Reserved** Read as 0; should be written with 0.

DTSSTAT

Die Temperature Sensor Status Register(0E0$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
RESULT	[9:0]	rh	**Result of the DTS Measurement** This bit field shows the result of the DTS measurement. The value given is directly related to the die temperature. Only the bit [9:2] have to be evaluated. The formula for mapping the result to a temperature will follow later.
RDY	14	rh	**Sensor Ready Status** This bit indicate the DTS is ready or not. 0_B The DTS is not ready 1_B The DTS is ready
BUSY	15	rh	**Sensor Busy Status** This bit indicate the DTS is currently busy or not. If the sensor is busy currently a measurement is running and the result should not be used. 0_B The DTS is not busy 1_B The DTS is busy *Note: This bit is updated 2 cycles after bit DTSCON.START is set.*
0	[13:10], [31:16]	r	**Reserved** Read as 0; should be written with 0.

3.8 Watchdog Timer

This section describes the TC1797 Watchdog Timer (WDT). Topics include an overview of the WDT function and descriptions of the registers, the password-protection scheme, accessing registers, modes, and initialization.

3.8.1 Watchdog Timer Overview

The WDT provides a highly reliable and secure way to detect and recover from software or hardware failure. The WDT helps to abort an accidental malfunction of the TC1797 in a user-specified time period. When enabled, the WDT can cause the TC1797 system to be reset if the WDT is not serviced within a user-programmable time period. The CPU must service the WDT within this time interval to prevent the WDT from causing a TC1797 System or Application Reset. Hence, routine service of the WDT confirms that the system is functioning properly.

In addition to this standard “Watchdog” function, the WDT incorporates the End-of-Initialization (Endinit) feature and monitors its modifications.

Because servicing the Watchdog and modifications of the ENDINIT bit are critical functions that must not be allowed in case of a system malfunction, a sophisticated scheme is implemented that requires a password and guard bits during accesses to the WDT control register. Any write access that does not deliver the correct password or the correct value for the guard bits is regarded as a malfunction of the system, and a Watchdog reset is requested. In addition, even after a valid access has been performed and the ENDINIT bit has been cleared to provide access to the critical registers, the Watchdog imposes a time limit for this access window. If bit ENDINIT has not been properly set again before this limit expires, the system is assumed to have malfunctioned, and a Watchdog reset is requested. These stringent requirements, although not guaranteed, nonetheless provide a high degree of assurance of the robustness of system operation.

A further enhancement in the TC1797’s WDT is its reset prewarning operation. Instead of immediately resetting the device on the detection of an error (the way that standard Watchdogs do), the WDT first issues a Non-Maskable Interrupt (NMI) to the CPU before finally resetting the device at a specified time period later.

3.8.2 Features of the Watchdog Timer

The main features of the WDT are summarized here.

- 16-bit Watchdog counter
- Selectable input frequency: $f_{FPI}/256$ or $f_{FPI}/16384$
- 16-bit user-definable reload value for normal Watchdog operation, fixed reload value for Time-Out and Prewarning Modes
- Incorporation of the ENDINIT bit and monitoring of its modifications

- Sophisticated Password Access mechanism with fixed and user-definable password fields
- Access Error Detection: Invalid password (during first access) or invalid guard bits (during second access) trigger the Watchdog reset generation
- Overflow Error Detection: An overflow of the counter triggers the Watchdog reset generation
- Watchdog function can be disabled; access protection and ENDINIT bit monitor function remain enabled
- Double Reset Detection

3.8.3 The Endinit Function

It is a prerequisite to understand the ENDINIT bit and its function for better understanding of the descriptions in the following sections. Hence, its function is explained first.

There are a number of registers in the TC1797 that are usually programmed only once during the initialization sequence of the application. Modification of such registers during normal application run can have a severe impact on the overall operation of modules or the entire system.

While the Supervisor Mode, that allows writes to registers only when it is active, provides a certain level of protection against unintentional modifications, it may not provide enough security for system-critical registers.

The TC1797 provides one more level of protection for such registers via the Endinit feature. This is a highly secure write-protection scheme that makes unintentional modifications of registers protected by this feature nearly impossible.

The Endinit feature consists of an ENDINIT bit incorporated in the WDT control register, WDT_CON0. Registers protected via Endinit determine whether or not writes are enabled. Writes are only enabled if bit ENDINIT = 0 AND Supervisor Mode is active. Write attempts if this condition is not true will be discarded and the register contents will not be modified in this case. The BCU controls the further operation following a discarded write access.

To get the highest level of security, this bit is incorporated in the highly secure access protection scheme implemented in the WDT. This is a complex procedure, that makes it nearly impossible for the ENDINIT bit to be modified unintentionally. In addition, the WDT monitors ENDINIT bit modifications by starting a time-out sequence each time software opens access to the critical registers through clearing bit ENDINIT. If the time out period ends before bit ENDINIT is set again, a malfunction of the software is assumed and a reset request is generated.

The access-protection scheme and the Endinit time-out operation of the WDT is described in the following sections. **Table 3-15** lists the registers that are protected via the Endinit feature in the TC1797.

Note: The clearing of the ENDINIT bit takes some time. Accesses to Endinit-protected registers after the clearing of the ENDINT bit must only be done when bit ENDINIT is really cleared. As a solution, WDT_CON0 (the register with the ENDINIT bit) should be read back once before Endinit-protected registers are accessed the first time after bit ENDINIT has been cleared.

Table 3-15 TC1797 Registers Protected via the Endinit Feature

Register Name	Description
mod_CLC	All clock control registers of the individual peripheral modules are Endinit-protected
mod_FDR	All clock fractional divider registers of the individual peripheral modules are Endinit-protected
BTV, BIV, ISP	Trap and interrupt vector table pointer as well as the interrupt stack pointer are Endinit-protected
FLASH0_FCON **FLASH1_FCON** **FLASH0_MARP** **FLASH1_MARP**	Flash configuration registers
WDT_CON1	The Watchdog Timer Control Register 1, which controls the disabling and the input frequency of the Watchdog Timer, is Endinit-protected. In addition, its bits will only have an effect on the WDT when ENDINIT is properly set to 1 again.
SCU_OSCCON **SCU_PLLCON0** **SCU_PLLCON1** **SCU_CCUCON0** **FDR** **SCU_CCUCON1** **SCU_PLLERAYCTR**	All clock control registers are protected
SCU_RSTCNTCON **SCU_RSTCON** **SCU_ARSTDIS** **SCU_SWRSTCON**	All reset control registers are protected
SCU_ESRCFG0 **SCU_ESRCFG1**	All ESR control registers are protected
SCU_EMSR	The emergency stop register

Table 3-15 TC1797 Registers Protected via the Endinit Feature (cont'd)

Register Name	Description
SCU_TRAPSET **SCU_TRAPDIS**	The trap set and disable register
SCU_PETCR	The parity control register

3.8.3.1 Password Access to WDT_CON0

A correct password must be written to register WDT_CON0 in order to unlock it for modifications. Software must either know the correct password in advance or compute it at runtime. The password required to unlock the register is formed by a combination of bits in registers WDT_CON0 and WDT_CON1, plus a number of guard bits. **Table 3-16** summarizes the requirements for the password.

Table 3-16 Password Access Bit Pattern Requirements

Bit Position	Required Value
0	Current state of bit WDT_CON0.ENDINIT
1	Fixed; must be written with 0
2	Current state of bit WDT_CON1.IR
3	Current state of bit WDT_CON1.DR
[7:4]	Fixed; must be written to 1111_B
[15:8]	Current value of user.definable password field WDT_CON0.PW
[31:16]	Current value of user-definable reload value, WDT_CON0REL

The password is designed such that it is not possible to just read the contents of a register and use this as the password. The password is never identical to the contents of WDT_CON0 or WDT_CON1, it is always required to modify the read value (at least bits 1 and [7:4]) to get the correct password. This prevents a malfunction from accidentally reading a WDT register's contents and writing it to WDT_CON0 as an unlocking password.

If the password matches the requirements, WDT_CON0 will be unlocked as soon as the Password Access is completed. The unlocked condition will be indicated by WDT_CON0.LCK = 0.

If an improper password value is written to WDT_CON0 during the Password Access, a Watchdog Access Error condition exists. Bit WDT_SR.AE is set and the Prewarning Mode is entered.

The user-definable password, WDT_CON0.PW, provides additional options for adjusting the password requirements to the application's needs. It can be used, for

instance, to detect unexpected software loops, or to monitor the execution sequence of routines.

3.8.3.2 Modify Access to WDT_CON0

If WDT_CON0 is successfully unlocked, the following write access to WDT_CON0 can modify it. However, this access must also meet certain requirements in order to be accepted and regarded as valid. **Table 3-17** lists the required bit patterns. If the access does not follow these rules, a Watchdog Access Error condition is detected, bit WDT_SR.AE is set, and the Prewarning Mode is entered.

Table 3-17 Modify Access Bit Pattern Requirements

Bit Position	Value
0	User-definable; desired value for bit WDT_CON0.ENDINIT.
1	Fixed; must be written with 1.
2	Fixed; must be written with 0.
3	Fixed; must be written with 0.
[7:4]	Fixed; must be written with 1111_B.
[15:8]	User-definable; desired value of user-definable password field, WDT_CON0.PW.
[31:16]	User-definable; desired value of user-definable reload value, WDT_CON0.REL.

After the Modify Access has completed, WDT_CON0.LCK is set again, automatically re-locking WDT_CON0. Before the register can be modified again, a valid Password Access must be executed again.

3.8.3.3 Access to Endinit-Protected Registers

If some or all of the system's Endinit-protected registers must be changed during run time of an application, access can be re-opened. To do this, WDT_CON0 must first be unlocked with a Valid Password Access. In the subsequent Valid Modify Access, ENDINIT can be cleared. Access to Endinit-protected registers is now open again. However, when WDT_CON0 is unlocked, the WDT is automatically switched to Time-Out Mode. Thus, the access window is time-limited. Time-Out Mode is only terminated after ENDINIT has been set again, requiring another Valid Password and Valid Modify Access to WDT_CON0.

If the WDT is not used in an application and is therefore disabled (WDT_SR.DS = 1), the above described case is the only occasion when WDT_CON0 must be accessed again after the system is initialized. If there are no further changes to critical system registers

needed, no further accesses to WDT_CON0, WDT_CON1, or WDT_SR are necessary. However, it is recommended that the WDT be used in an application for safety reasons.

For debugging support the Cerberus module can override the ENDINIT control to ease the debug flow. If bit CBS_OSTATE.ENIDIS is set the ENDINIT protection is disabled independent of the current status configured by the WDT. If CBS_OSTATE.ENIDIS is cleared the complete control is within the WDT.

3.8.4 Timer Operation

The timer is automatically active after an Application Reset. The 16-bit counter implementing the timer functionality is either triggered with f_{FPI} / 256 or f_{FPI} / 16384. The two possible counting rates are controlled via bit WDT_CON1.IR.

Determining WDT Periods

The WDT uses the FPI-Bus clock f_{FPI}. A clock divider in front of the WDT provides two output frequencies, f_{FPI} / 256 and f_{FPI} / 16384.

The general formula to calculate a Watchdog period is:

$$period = \frac{(2^{16} - startvalue) \cdot 256 \cdot 2^{(1-IR)\cdot 6}}{f_{FPI}} \quad (3.19)$$

The parameter start value represents the fixed value $FFFC_H$ for the calculation of the Time-Out Period, and the user-programmable reload value WDT_CON0.REL for the calculation of the Normal Period.

3.8.4.1 Timer Modes

The Watchdog Timer can operate in one of four different operating modes:

- Time-Out Mode
- Normal Mode
- Disable Mode
- Prewarning Mode

The following overview describes these modes and how the WDT changes from one mode to the other.

Time-Out Mode

The Time-Out Mode is entered after an Application Reset or when a valid Password Access to register WDT_CON0 is performed (see **Section 3.8.3.1**). The Time-Out Mode is indicated by bit WDT_SR.TO = 1. The timer is set to $FFFC_H$ and starts counting upwards. Time-Out Mode can only be exited properly by setting ENDINIT = 1 with a correct access sequence. If an improper access to the WDT is performed, or if the timer

overflows before ENDINIT is set, a WDT_NMI is requested, and Prewarning Mode is entered.

A proper exit from Time-Out Mode can either be to the Normal or the Disable Mode, depending on the state of the disable request bit WDT_CON1.DR.

Normal Mode

In Normal Mode (DR = 0), the WDT operates in a standard Watchdog fashion. The timer is set to WDT_CON0.REL, and begins counting up. It has to be serviced before the counter overflows. Servicing is performed through a proper access sequence to the control register WDT_CON0. This enters the Time-Out Mode.

If the WDT is not serviced before the timer overflows, a system malfunction is assumed. Normal Mode is terminated, a WDT_NMI is requested, and Prewarning Mode is entered.

Disable Mode

Disable Mode is provided for applications which truly do not require the WDT function. It can be requested from Time-Out Mode when the disable request bit WDT_CON1.DR is set. The Disable Mode is entered when was requested AND bit WDT_CON0.ENDINIT is set. The timer is stopped in this mode. However, disabling the WDT only stops it from performing the standard Watchdog function, eliminating the need for timely service of the WDT. It does not disable Time-Out and Prewarning Mode. If an access to register WDT_CON0 is performed in Disable Mode, Time-Out Mode is entered if the access was valid, and Prewarning Mode is entered if the access was invalid. Thus, the ENDINIT monitor function as well as (a part of) the system malfunction detection will still be active.

Prewarning Mode

Prewarning Mode is entered always when a Watchdog error is detected. This can be due to an overflow of the timer in Normal or Time-Out Mode, or an invalid access to register WDT_CON0. Instead of immediately requesting a reset of the device, the WDT enables the system to enter a secure state by a prewarning before the reset occurs.

In Prewarning Mode, after having generated the NMI request, the WDT counts up from $FFFC_H$, and then generates a Watchdog reset request on the overflow. This reset request cannot be avoided in this mode; the WDT does not react anymore to accesses to its registers, nor will it change its state unless reset by an Application Reset. This is to prevent a malfunction from falsely terminating this mode, disabling the reset, and letting the device to continue to function improperly. Register WDT_CON0 can still be accessed by a valid Password Access.

Note: In Prewarning Mode, it is not required for the part to wait for the end of this mode and the reset. After having saved required state in the NMI routine, software can execute an Application Reset to shorten the time.

Note: The Prewarning Mode is only left by an application Reset and not only on the reset request. Therefore if bit field RSTCON.WDT is set to 00_B the Prewarning Mode is not left.

3.8.4.2 WDT Reset Behavior

WDT reset requests are generated for three cases:

- Invalid password access to register WDT_CON0
- Not finishing a password access before a timer overflow occurs in the Time-Out Mode
- Not serving the WDT before a timer overflow occurs in the Normal Mode

If a reset is generated on a WDT reset request and the kind of the reset can be configured via bit field RSTCON.WDT.

Note: The WDT itself is reset by any Application Reset.

Before a reset is requested the Prewarning Mode is entered, for more details see **Section 3.8.4.1**.

Double WDT Reset

If the Watchdog induced reset occurs twice, a severe system malfunction is assumed and the TC1797 is held in reset until a System Reset occurs. This prevents the device from being periodically reset if, for instance, connection to the external memory has been lost such that even system initialization could not be performed.

If the WDT is configured to request an Application Reset the second reset request will be permanently asserted resulting (without any change in the reset configuration) resulting in a permanent Application Reset. The information about the first WDT reset request is stored in an internal flag which is reset with the System Reset. This flag is set when the Prewarning Mode is finished and the reset request is generated. If a new reset is requested and the internal flag is already set a double reset event has occurred and a permanent request is generated. This internal flag is cleared by any System Reset or when bit WDT_CON1.CLRIRF is set AND bit WDT_CON0.ENDINIT is set too. Please note that a correct service of the WDT does not clear this internal flag. Bit WDT_CON1.CLRIRF can only be set when bit WDT_CON0.ENDINIT is cleared.

Note: It does not matter whether a reset was generated on a WDT reset request or if the reset configuration was changed between the two reset requests.

Note: If for any reason random code is executed bit field RSTCON.WDT can be updated unintentional. This can result that a WDT error does not lead to an reset. To avoid this after the SSW is finished this bit field should be checked and the ENDINIT protection enabled.

Servicing the Watchdog Timer

If the WDT is used in an application and is enabled (WDT_SR.DS = 0), it must be regularly serviced to prevent it from overflowing.

Service is performed in two steps. a Valid Password Access followed by a Valid Modify Access. The Valid Password Access to WDT_CON0 automatically switches the WDT to Time-Out Mode. Thus, the Modify Access must be performed before the Time-out expires or a System Reset will result.

During the next Modify Access, the strict requirement is that WDT_CON0.ENDINIT as well as bit 1 and bits [7:4] are written with 1, while bits [3:2] are written with 0.

Note: ENDINIT must be written with 1 to perform a proper service, even if it is already set to 1.

Changes to the reload value WDT_CON0.REL, or the user-definable password WDT_CON0.PW, are not required. However, changing WDT_CON0.PW is recommended so that software can monitor WDT service operations throughout the duration of an application program (see next section).

When WDT service is properly executed, Time-Out Mode is terminated, and the WDT switches back to its former mode of operation, and WDT service is complete.

3.8.4.3 WDT Operation During Power-Saving Modes

If the CPU is in Idle Mode or Sleep Mode, it cannot service the WDT because no software is running. Excluding the case where the system is running normally, a strategy for managing the WDT is needed while the CPU is in Idle or Sleep Mode. There are two ways to manage the WDT in these cases. First, the Watchdog can be disabled before idling the CPU. The disadvantage of this is that the system will no longer be monitored during the idle period.

A better approach to this problem relies upon a wake-up feature of the WDT. Whenever the CPU is put in Idle or Sleep Mode and the WDT is not disabled, it causes the CPU to be awakened at regular intervals. When the WDT changes its count value (WDT_SR.TIM) from $7FFF_H$ to 8000_H, the CPU is awakened and continues to execute the instruction following the instruction that was last executed before entering the Idle or Sleep Mode.

Note: Before switching into a non-running power-management mode, software should perform a Watchdog service sequence. At the Modify Access, the Watchdog reload value, WDT_CON0.REL, should be programmed such that the wake-up occurs after a period which best meets application requirements. The maximum period between two CPU wake-ups is one-half of the maximum WDT period.

3.8.4.4 Suspend Mode Support

In an enabled and active debug session the Watchdog functionality can lead to unintended resets. Therefore to avoid these resets the OCDS can control if the WDT is enabled or disabled (default after Application Reset) via bit CBS_OSTATE.WDTSUS if it is not already stopped.

Table 3-18 OCDS Behavior of WDT

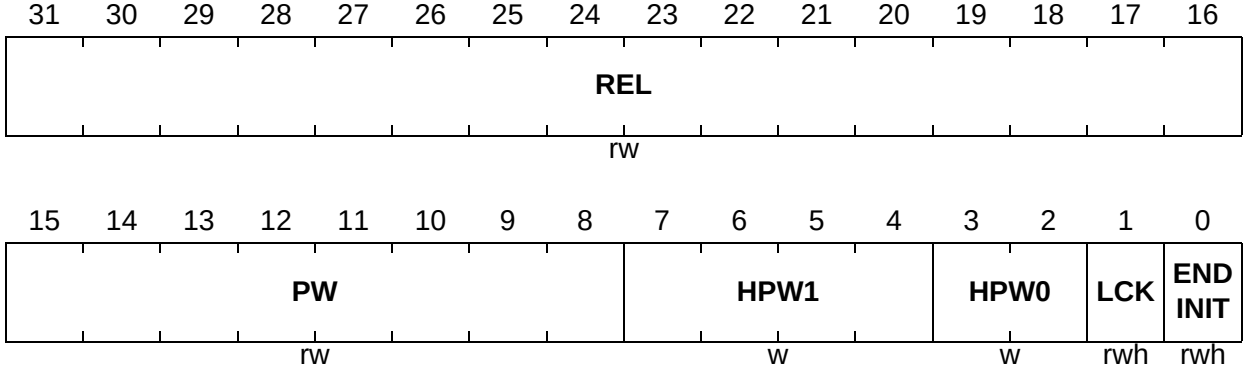
STCON.STP	WDT_SR.DS	CBS_OSTATE.OEN	CBS_OSTATE.SUS	CBS_MCDSSG.SOS	WDT Action
0	X	X	X	X	Stopped
1	1	X	X	X	Stopped
1	0	0	X	X	Running
1	0	1	0	X	Stopped
1	0	1	1	0	Running
1	0	1	1	1	Stopped

3.8.5 Watchdog Timer Registers

3.8.5.1 Watchdog Timer Control Register 0

Register WDT_CON0 manages Password Access to the Watchdog Timer. It also stores the timer reload value, a user-definable password field, a lock bit, and the End-of-Initialization (ENDINIT) control bit.

WDT_CON0
WDT Control Register 0 (F000 05F0$_H$) Reset Value: FFFC 0002$_H$



Field	Bits	Type	Description
ENDINIT	0	rwh	**End-of-Initialization Control Bit** 0_B Access to Endinit-protected registers is permitted (default after Application Reset) 1_B Access to Endinit-protected registers is not permitted
LCK	1	rwh	**Lock Bit to Control Access to WDT_CON0** 0_B Register WDT_CON0 is unlocked 1_B Register WDT_CON0 is locked (default after Application Reset) The actual value of LCK is controlled by hardware. It is cleared after a valid Password Access to WDT_CON0, and automatically set again after a valid Modify Access to WDT_CON0. During a write to WDT_CON0, the value written to this bit is only used for the password-protection mechanism and is not stored. This bit must be cleared during a Password Access to WDT_CON0, and set during a Modify Access to WDT_CON0. That is, the inverted value read from LCK always must be written to itself.
HPW0	[3:2]	w	**Hardware Password 0** This bit field must be written with the value of the bits WDT_CON1.DR and WDT_CON1.IR during a Password Access. This bit field must be written with 0s during a Modify Access to WDT_CON0. When read, these bits always return 0.
HPW1	[7:4]	w	**Hardware Password 1** This bit field must be written with 1111_B during both Password Access and Modify Access to WDT_CON0. When read, these bits always return 0.
PW	[15:8]	rw	**User-Definable Password Field for Access to WDT_CON0** This bit field must be written with its current contents during a Password Access. It can be changed during a Modify Access to WDT_CON0.

Field	Bits	Type	Description
REL	[31:16]	rw	**Reload Value for the WDT** If the Watchdog Timer is enabled and in Normal Timer Mode, it will start counting from this value after a correct Watchdog service. This bit field must be written with its current contents during a Password Access. It can be changed during a Modify Access to WDT_CON0 ($FFFC_H$ = default after Application Reset).

3.8.5.2 Watchdog Timer Control Register 1

WDT_CON1 manages operation of the WDT. It includes the disable request and frequency selection bits. It is ENDINIT-protected.

WDT_CON1
WDT Control Register 1 (F000 05F4$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

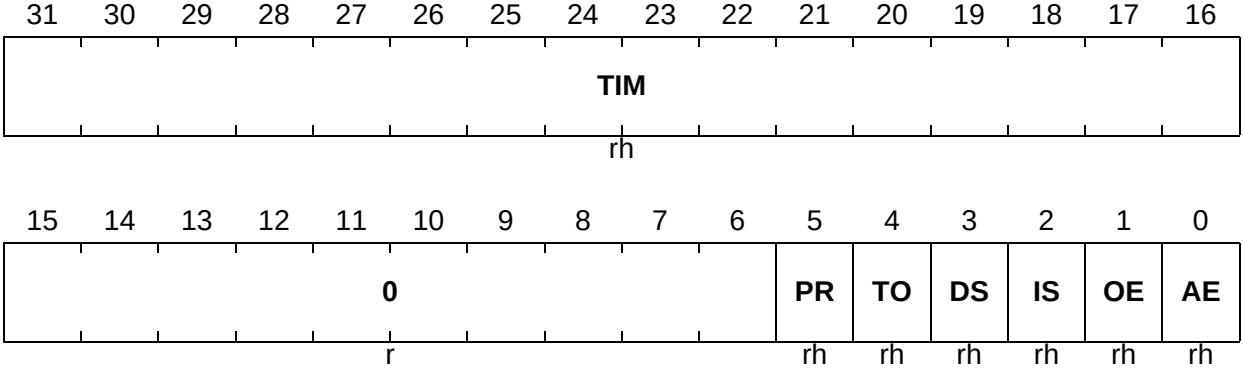
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0												DR	IR	0	CLR IRF
r												rw	rw	r	rwh

Field	Bits	Type	Description
CLRIRF	0	rwh	**Clear Internal Reset Flag** This bit is used to request a clear of the internal flag storing the information about the first WDT reset request. 0_B No action 1_B Request to clear the internal flag This bit can only be modified if WDT_CON0.ENDINIT is cleared. The internal flag is cleared when ENDINIT is set again. As long as ENDINIT is cleared, the internal flag is unchanged and controls the current past error status of the WDT. When ENDINIT is set again with a Valid Modify Access, the internal flag is cleared together with this bit.
IR	2	rw	**Input Frequency Request Control Bit** 0_B Request to set input frequency to f_{FPI}/16384. 1_B Request to set input frequency to f_{FPI}/256. This bit can only be modified if WDT_CON0.ENDINIT is cleared. WDT_SR.IS is updated by this bit only when ENDINIT is set again. As long as ENDINIT is cleared, WDT_SR.IS controls the current input frequency of the Watchdog Timer. When ENDINIT is set again, WDT_SR.IS is updated with the state of IR.
DR	3	rw	**Disable Request Control Bit** 0_B Request to enable the WDT 1_B Request to disable the WDT This bit can only be modified if WDT_CON0.ENDINIT is cleared. WDT_SR.DS is updated when ENDINIT is set again. As long as ENDINIT is cleared, bit WDT_SR.DS controls the current enable/disable status of the WDT. When ENDINIT is set again with a Valid Modify Access, WDT_SR.DS is updated with the state of DR.
0	1, [31:4]	r	**Reserved** Read as 0; should be written with 0.

3.8.5.3 Watchdog Timer Status Register

Register WDT_SR shows the current state of the WDT. Status include bits indicating reset prewarning, Time-Out, enable/disable status, input clock status, and access error status.

WDT_SR
WDT Status Register **(F000 05F8$_H$)** **Reset Value: FFFC 0010$_H$**



Field	Bits	Type	Description
AE	0	rh	**Watchdog Access Error Status Flag** 0_B No Watchdog access error 1_B A Watchdog access error has occurred This bit is set when an illegal Password Access or Modify Access to register WDT_CON0 was attempted. This bit is only cleared when WDT_CON0.ENDINIT is set during a Valid Modify Access However, it is not possible to clear this bit if the WDT is in Prewarning Mode.
OE	1	rh	**Watchdog Overflow Error Status Flag** 0_B No Watchdog overflow error 1_B A Watchdog overflow error has occurred This bit is set when the WDT overflows from FFFF$_H$ to 0000$_H$. This bit is only cleared when WDT_CON0.ENDINIT is set to 1 during a Valid Modify Access. However, it is not possible to clear this bit if the WDT is in Prewarning Mode.

Field	Bits	Type	Description
IS	2	rh	**Watchdog Input Clock Status Flag** 0_B The timer operation clock is f_{FPI}/16384 (default after Application Reset) 1_B The timer operation clock is f_{FPI}/256 This bit is updated with the state of bit WDT_CON1.IR after WDT_CON0.ENDINIT is written with 1 during a Valid Modify Access to register WDT_CON0.
DS	3	rh	**Watchdog Enable/Disable Status Flag** 0_B WDT is enabled (default after Application Reset) 1_B WDT is disabled This bit is updated with the state of bit WDT_CON1.DR after WDT_CON0.ENDINIT is set during a Valid Modify Access to register WDT_CON0.
TO	4	rh	**Watchdog Time-Out Mode Flag** 0_B The Watchdog is not operating in Time-Out Mode 1_B The Watchdog is operating in Time-Out Mode (default after Application Reset) This bit is set when Time-Out Mode is entered. It is automatically cleared when Time-Out Mode is left.
PR	5	rh	**Watchdog Prewarning Mode Flag** 0_B The Watchdog is not operating in Prewarning Mode 1_B The Watchdog is operating in Prewarning Mode This bit is set when a Watchdog error is detected. The WDT has issued a trap trigger and is in Prewarning Mode. A reset of the chip occurs after the prewarning period has expired if it is enabled in bit field RSTCON.WDT.
TIM	[31:16]	rh	**Timer Value** Reflects the current content of the WDT.
0	[15:6]	r	**Reserved** Read as 0.

3.9 Emergency Stop Output Control

The emergency stop feature of the TC1797 allows for a fast emergency reaction on an external event without the intervention of software. In an emergency case, the outputs can be selectively put immediately to a well-defined logic state (for more information see the port chapter).

The emergency case is indicated by an emergency input signal with selectable polarity that has to be connected to input P9.10.

Figure 3-26 shows a diagram of the emergency stop input logic. This logic is controlled by the SCU Emergency Stop Register EMSR.

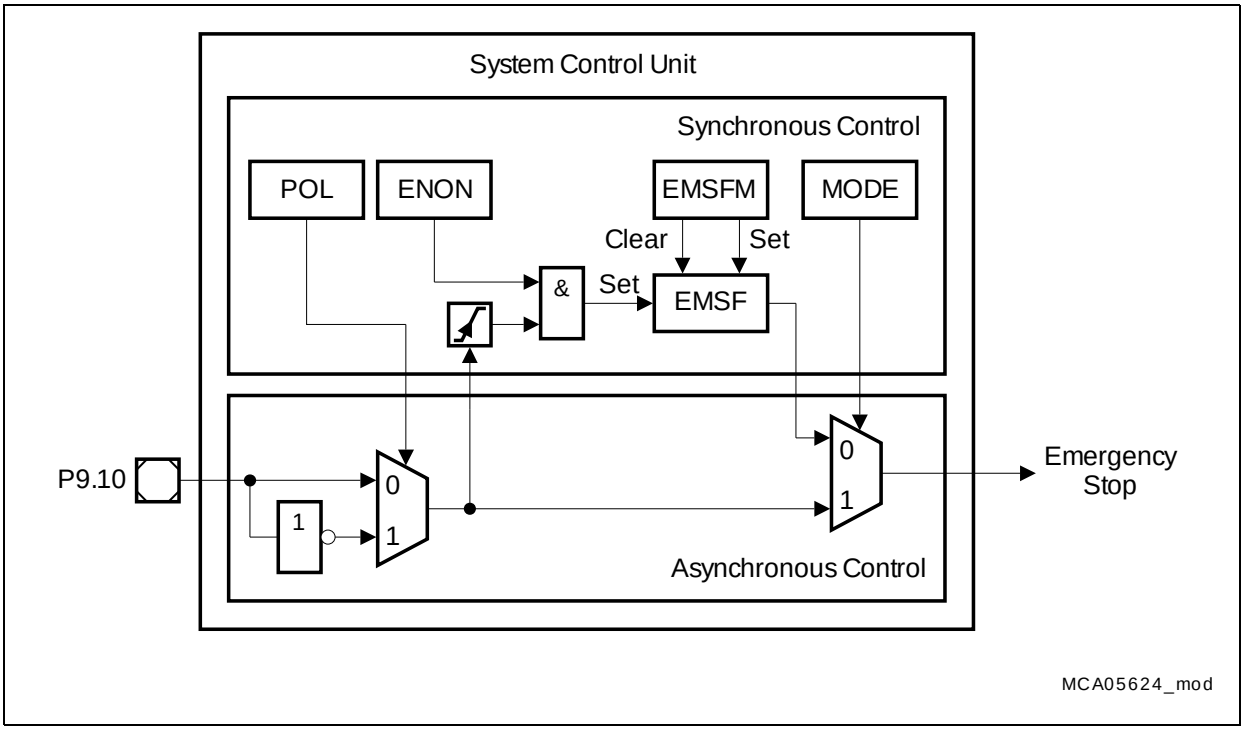


Figure 3-26 Emergency Stop Input Control

The emergency stop control logic for the ports can basically operate in two modes:

- Synchronous Mode (default after reset):
  Emergency case is activated by hardware and released by software.
- Asynchronous Mode:
  Emergency case is activated and released by hardware.

In Synchronous Mode (selected by EMSR.MODE = 0), the port signal is sampled for a inactive-to-active level transition, and an emergency stop flag EMSR.EMSF is set if the transition is detected. The setting of EMSR.EMSF activates the emergency stop signal. An emergency case can only be terminated by clearing EMSR.EMSF via software. The synchronous control logic is clocked by the system clock f_{SYS}. This results in a small delay between the port signal and emergency stop signal generation.

In Asynchronous Mode (selected by EMSR.MODE = 1), the occurrence of an active level at the port input immediately activates the emergency stop signal. Of course, a valid-to-invalid transition of the port input (emergency case is released) also immediately deactivates the emergency stop signal.

The EMSR.POL bit determines the active level of the input signal. The EMSR.MODE bit selects Synchronous or Asynchronous Mode for emergency stop signal generation.

3.9.1 Emergency Stop Register

The Emergency Stop Register EMSR contains control and status bits/flags of the emergency stop input logic.

EMSR
Emergency Stop Register **(100_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0						EMSFM		0							EMSF
r						w		r							rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0													ENON	MODE	POL
r													rw	rw	rw

Field	Bits	Type	Description
POL	0	rw	**Input Polarity** This bit determines the polarity of the input line. 0_B Input is high active 1_B Input is low active
MODE	1	rw	**Mode Selection** This bit determines the operating mode of the emergency stop signal. 0_B Synchronous Mode selected; emergency stop is derived from the state of flag EMSF 1_B Asynchronous Mode selected; emergency stop is directly derived from the state of the input signal
ENON	2	rw	**Enable ON** This bit enables the setting of flag EMSF by an inactive-to-active level transition of input signal. 0_B Setting of EMSF is disabled 1_B Setting of EMSF is enabled

Field	Bits	Type	Description
EMSF	16	rh	**Emergency Stop Flag** This bit indicates if an emergency stop condition has occurred. 0_B An emergency stop has not occurred 1_B An emergency stop has occurred and signal emergency stop becomes active (if MODE = 0)
EMSFM	[25:24]	w	**Emergency Stop Flag Modification** This bit field set or clear flag EMSF via software. 00_B EMSF remains unchanged 01_B EMSF becomes set 10_B EMSF becomes cleared 11_B EMSF remains unchanged EMSFM is always read as 00_B.
0	[15:3], [23:17], [31:26]	r	**Reserved** Read as 0; should be written with 0.

3.10 Interrupt Generation

The interrupt structure is shown in **Figure 3-27**. The interrupt request or the corresponding interrupt set bit (in register INTSET) can trigger the interrupt generation at the selected interrupt node x. The service request pulse is generated independently from the interrupt flag in register INTSTAT. The interrupt flag can be cleared by software by writing to the corresponding bit in register INTCLR.
If more than one interrupt source is connected to the same interrupt node pointer (in register INTNP), the requests are combined to one common line.

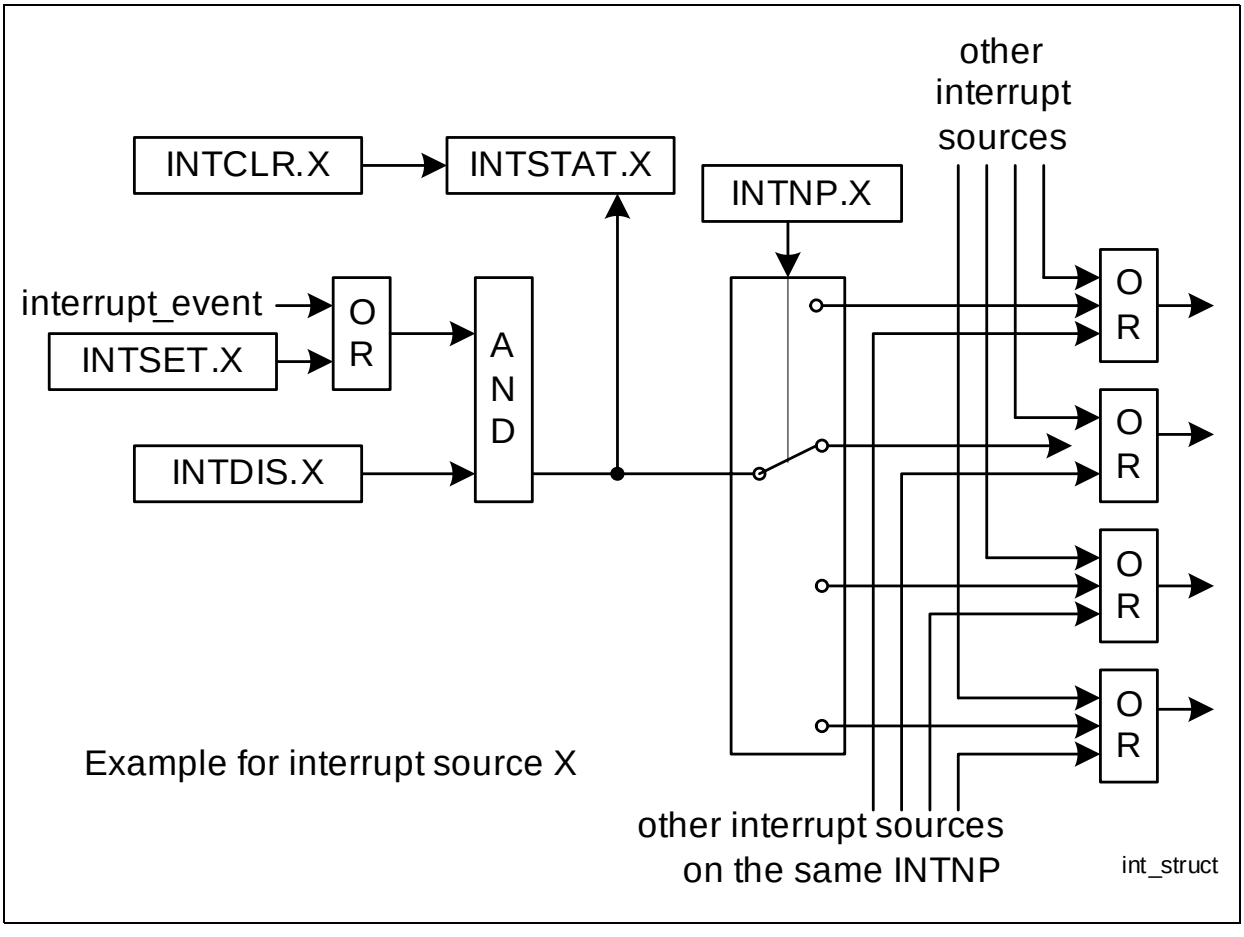
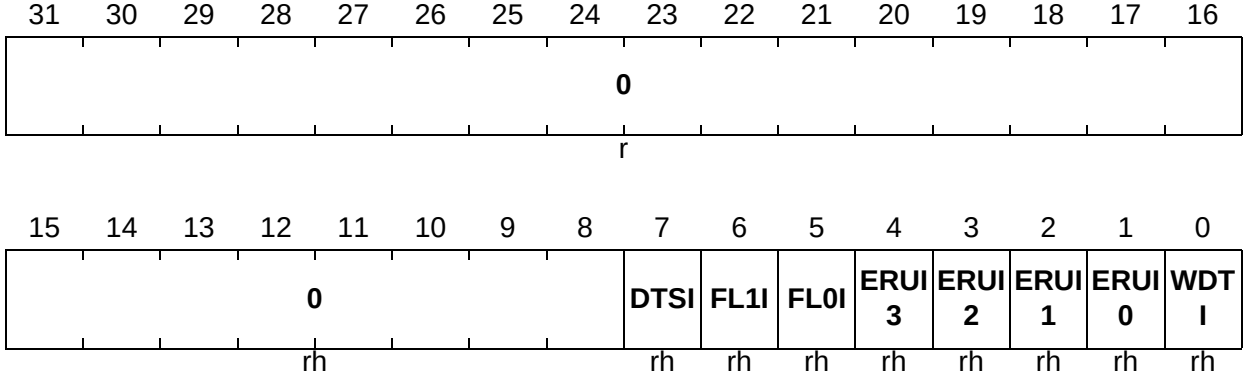


Figure 3-27 Interrupt Generation

3.10.1 Interrupt Control Registers

INTSTAT
Interrupt Status Register **(110_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
WDTI	0	rh	**Watchdog Timer Interrupt Request Flag** This bit is set if the WDT Prewarning Mode is entered and bit is INTDIS.WDTI = 0. 0_B No interrupt was requested since this bit was cleared the last time 1_B An interrupt was requested since this bit was cleared the last time This bit can be cleared by bit INTCLR.WDTI. This bit can be set by bit INTSET.WDTI.
ERUI0	1	rh	**ERU Channel 0 Interrupt Request Flag** This bit is set if the ERU channel 0 is active and bit is INTDIS.ERUI0 = 0. 0_B No interrupt was requested since this bit was cleared the last time 1_B An interrupt was requested since this bit was cleared the last time This bit can be cleared by bit INTCLR.ERUI0. This bit can be set by bit INTSET.ERUI0.

Field	Bits	Type	Description
ERUI1	2	rh	**ERU Channel 1 Interrupt Request Flag** This bit is set if the ERU channel 1 is active and bit is INTDIS.ERUI1 = 0. 0_B No interrupt was requested since this bit was cleared the last time 1_B An interrupt was requested since this bit was cleared the last time This bit can be cleared by bit INTCLR.ERUI1. This bit can be set by bit INTSET.ERUI1.
ERUI2	3	rh	**ERU Channel 2 Interrupt Request Flag** This bit is set if the ERU channel 2 is active and bit is INTDIS.ERUI2 = 0. 0_B No interrupt was requested since this bit was cleared the last time 1_B An interrupt was requested since this bit was cleared the last time This bit can be cleared by bit INTCLR.ERUI2. This bit can be set by bit INTSET.ERUI2.
ERUI3	4	rh	**ERU Channel 3 Interrupt Request Flag** This bit is set if the ERU channel 3 is active and bit is INTDIS.ERUI3 = 0. 0_B No interrupt was requested since this bit was cleared the last time 1_B An interrupt was requested since this bit was cleared the last time This bit can be cleared by bit INTCLR.ERUI3. This bit can be set by bit INTSET.ERUI3.
FL0I	5	rh	**Flash 0 Interrupt Request Flag** This bit is set if the Flash interrupt trigger is active and bit is INTDIS.FL0I = 0. 0_B No interrupt was requested since this bit was cleared the last time 1_B An interrupt was requested since this bit was cleared the last time This bit can be cleared by bit INTCLR.FL0I. This bit can be set by bit INTSET.FL0I.

Field	Bits	Type	Description
FL1I	6	rh	**Flash 1 Interrupt Request Flag** This bit is set if the Flash interrupt trigger is active and bit is INTDIS.FL1I = 0. 0_B No interrupt was requested since this bit was cleared the last time 1_B An interrupt was requested since this bit was cleared the last time This bit can be cleared by bit INTCLR.FL1I. This bit can be set by bit INTSET.FL1I.
DTSI	7	rh	**DTS Interrupt Request Flag** This bit is set if the DTS busy indication changes from 1_B to 0_B and bit is INTDIS.DTSI = 0. 0_B No interrupt was requested since this bit was cleared the last time 1_B An interrupt was requested since this bit was cleared the last time This bit can be cleared by bit INTCLR.DTSI. This bit can be set by bit INTSET.DTSI.
0	[15:8]	rh	**Reserved** Read as 0. This bit can be cleared by bit INTCLR.[x]. This bit can be set by bit INTSET.[x]. *Note: x = 6, [13:8], 15.*
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

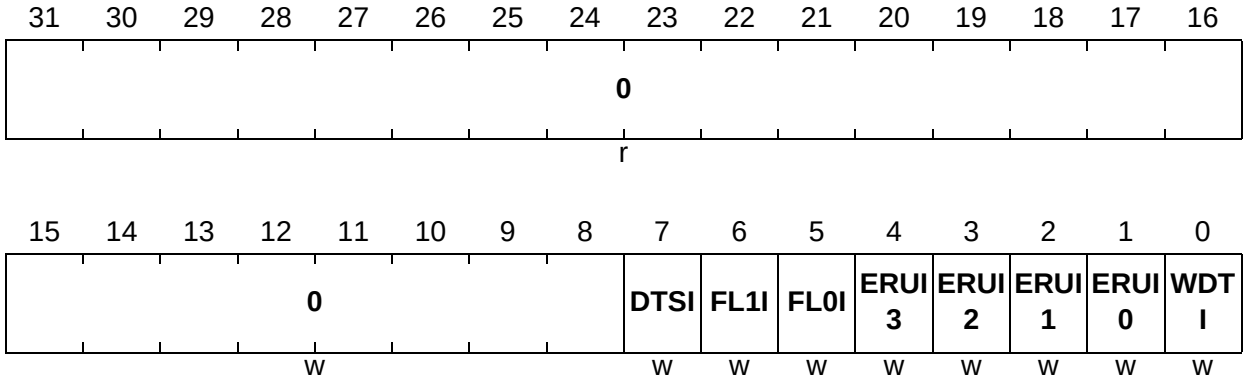
INTSET
Interrupt Set Register **(114_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								DTSI	FL1I	FL0I	ERUI3	ERUI2	ERUI1	ERUI0	WDTI
w								w	w	w	w	w	w	w	w

Field	Bits	Type	Description
WDTI	0	w	**Set Interrupt Request Flag WDTI** Setting this bit set bit INTSTAT.WDTI. Clearing this bit has no effect. Reading this bit returns always zero.
ERUI0	1	w	**Set Interrupt Request Flag ERUI0** Setting this bit set bit INTSTAT.ERUI0. Clearing this bit has no effect. Reading this bit returns always zero.
ERUI1	2	w	**Set Interrupt Request Flag ERUI1** Setting this bit set bit INTSTAT.ERUI1. Clearing this bit has no effect. Reading this bit returns always zero.
ERUI2	3	w	**Set Interrupt Request Flag ERUI2** Setting this bit set bit INTSTAT.ERUI2. Clearing this bit has no effect. Reading this bit returns always zero.
ERUI3	4	w	**Set Interrupt Request Flag ERUI3** Setting this bit set bit INTSTAT.ERUI3. Clearing this bit has no effect. Reading this bit returns always zero.
FL0I	5	w	**Set Interrupt Request Flag FL0I** Setting this bit set bit INTSTAT.FL0I. Clearing this bit has no effect. Reading this bit returns always zero.
FL1I	6	w	**Set Interrupt Request Flag FL1I** Setting this bit set bit INTSTAT.FL1I. Clearing this bit has no effect. Reading this bit returns always zero.
DTSI	7	w	**Set Interrupt Request Flag DTSI** Setting this bit set bit INTSTAT.DTSI. Clearing this bit has no effect. Reading this bit returns always zero.
0	[15:8]	w	**Reserved** Read as 0; have to be written with 0.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

INTCLR
Interrupt Clear Register **(118_H)** **Reset Value: $0000\ 0000_H$**

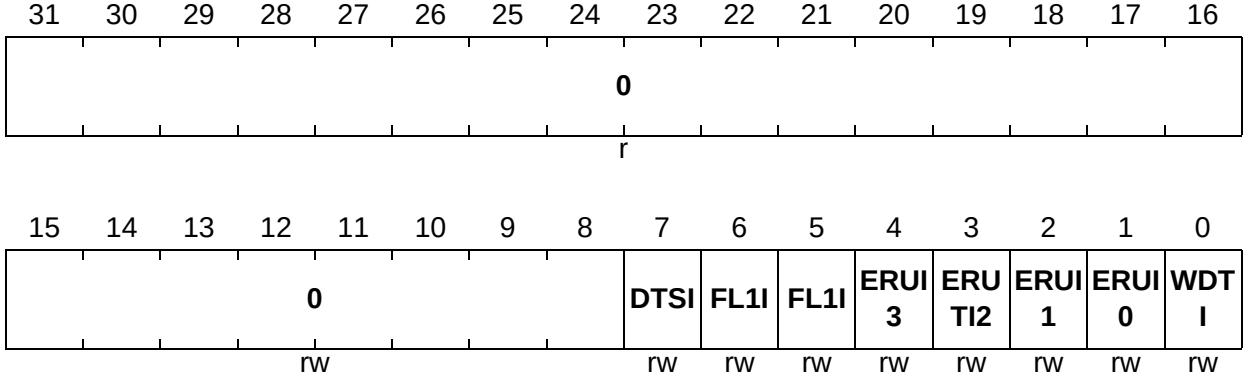


Field	Bits	Type	Description
WDTI	0	w	**Clear Interrupt Request Flag WDTI** Setting this bit clears bit INTSTAT.WDTI. Clearing this bit has no effect. Reading this bit returns always zero.
ERUI0	1	w	**Clear Interrupt Request Flag ERUI0** Setting this bit clears bit INTSTAT.ERUI0. Clearing this bit has no effect. Reading this bit returns always zero.
ERUI1	2	w	**Clear Interrupt Request Flag ERUI1** Setting this bit clears bit INTSTAT.ERUI1. Clearing this bit has no effect. Reading this bit returns always zero.
ERUI2	3	w	**Clear Interrupt Request Flag ERUI2** Setting this bit clears bit INTSTAT.ERUI2. Clearing this bit has no effect. Reading this bit returns always zero.
ERUI3	4	w	**Clear Interrupt Request Flag ERUI3** Setting this bit clears bit INTSTAT.ERUI3. Clearing this bit has no effect. Reading this bit returns always zero.
FL0I	5	w	**Clear Interrupt Request Flag FL0I** Setting this bit clears bit INTSTAT.FL0I. Clearing this bit has no effect. Reading this bit returns always zero.

Field	Bits	Type	Description
FL1I	6	w	**Clear Interrupt Request Flag FL1I** Setting this bit clears bit INTSTAT.FL1I. Clearing this bit has no effect. Reading this bit returns always zero.
DTSI	7	w	**Clear Interrupt Request Flag DTSI** Setting this bit clears bit INTSTAT.DTSI. Clearing this bit has no effect. Reading this bit returns always zero.
0	[15:8]	w	**Reserved** Read as 0; have to be written with 1.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

INTDIS

Interrupt Disable Register **($11C_H$)** **Reset Value: $0000\ 0000_H$**

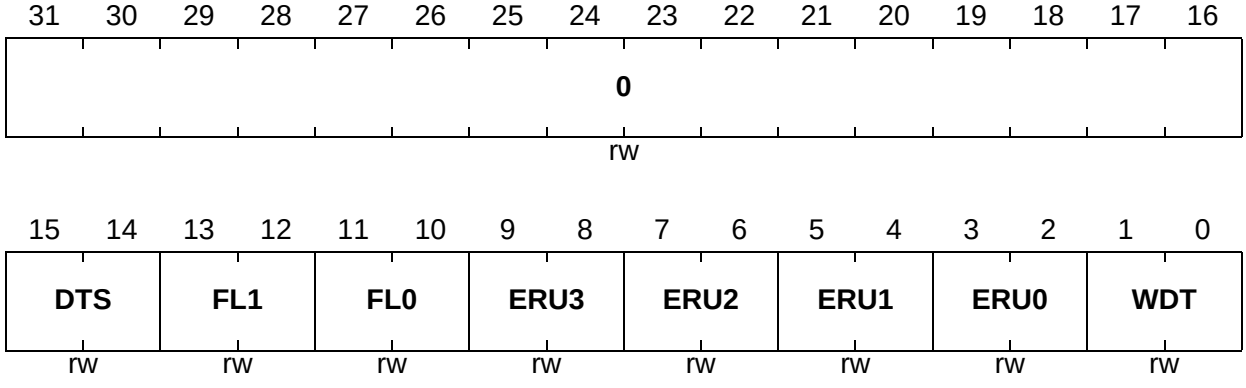


Field	Bits	Type	Description
WDTI	0	rw	**Disable Interrupt Request WDT** 0_B An interrupt request can be generated for this source 1_B No interrupt request can be generated for this source
ERUI0	1	rw	**Disable Interrupt Request ERU0** 0_B An interrupt request can be generated for this source 1_B No interrupt request can be generated for this source

Field	Bits	Type	Description
ERUI1	2	rw	**Disable Interrupt Request ERU1** 0_B An interrupt request can be generated for this source 1_B No interrupt request can be generated for this source
ERUI2	3	rw	**Disable Interrupt Request ERU2** 0_B An interrupt request can be generated for this source 1_B No interrupt request can be generated for this source
ERUI3	4	rw	**Disable Interrupt Request ERU3** 0_B An interrupt request can be generated for this source 1_B No interrupt request can be generated for this source
FL0I	5	rw	**Disable Interrupt Request Flash 0** 0_B An interrupt request can be generated for this source 1_B No interrupt request can be generated for this source
FL1I	6	rw	**Disable Interrupt Request Flash 1** 0_B An interrupt request can be generated for this source 1_B No interrupt request can be generated for this source
DTSI	7	rw	**Disable Interrupt Request DTS** 0_B An interrupt request can be generated for this source 1_B No interrupt request can be generated for this source
0	[15:8]	rw	**Reserved** Have to be written with 1.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

INTNP

Interrupt Node Pointer Register (120_H) Reset Value: $0000\ 0000_H$

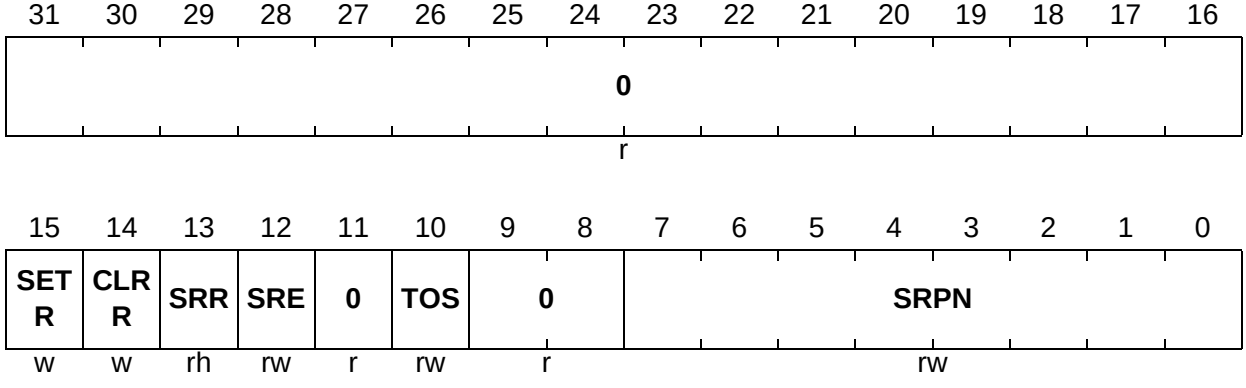


Field	Bits	Type	Description
WDT	[1:0]	rw	**Interrupt Node Pointer for Interrupt WDT** This bit field defines the interrupt node, that is requested due to the set condition for bit INTSTAT.WDTI (if enabled by bit INTDIS.WDTI). 00_B Interrupt node 0 is selected 01_B Interrupt node 1 is selected 10_B Interrupt node 2 is selected 11_B Interrupt node 3 is selected
ERU0	[3:2]	rw	**Interrupt Node Pointer for Interrupt ERU0** This bit field defines the interrupt node, that is requested due to the set condition for bit INTSTAT.ERUI0 (if enabled by bit INTDIS.ERUI0). 00_B Interrupt node 0 is selected 01_B Interrupt node 1 is selected 10_B Interrupt node 2 is selected 11_B Interrupt node 3 is selected
ERU1	[5:4]	rw	**Interrupt Node Pointer for Interrupt ERU1** This bit field defines the interrupt node, that is requested due to the set condition for bit INTSTAT.ERUI1 (if enabled by bit INTDIS.ERUI1). 00_B Interrupt node 0 is selected 01_B Interrupt node 1 is selected 10_B Interrupt node 2 is selected 11_B Interrupt node 3 is selected

Field	Bits	Type	Description
ERU2	[7:6]	rw	**Interrupt Node Pointer for Interrupt ERU2** This bit field defines the interrupt node, that is requested due to the set condition for bit INTSTAT.ERUI2 (if enabled by bit INTDIS.ERUI2). 00_B Interrupt node 0 is selected 01_B Interrupt node 1 is selected 10_B Interrupt node 2 is selected 11_B Interrupt node 3 is selected
ERU3	[9:8]	rw	**Interrupt Node Pointer for Interrupt ERU3** This bit field defines the interrupt node, that is requested due to the set condition for bit INTSTAT.ERUI3 (if enabled by bit INTDIS.ERUI3). 00_B Interrupt node 0 is selected 01_B Interrupt node 1 is selected 10_B Interrupt node 2 is selected 11_B Interrupt node 3 is selected
FL0	[11:10]	rw	**Interrupt Node Pointer for Interrupt FL0** This bit field defines the interrupt node, that is requested due to the set condition for bit INTSTAT.FL0I (if enabled by bit INTDIS.FL0I). 00_B Interrupt node 0 is selected 01_B Interrupt node 1 is selected 10_B Interrupt node 2 is selected 11_B Interrupt node 3 is selected
FL1	[13:12]	rw	**Interrupt Node Pointer for Interrupt FL1** This bit field defines the interrupt node, that is requested due to the set condition for bit INTSTAT.FL1I (if enabled by bit INTDIS.FL1I). 00_B Interrupt node 0 is selected 01_B Interrupt node 1 is selected 10_B Interrupt node 2 is selected 11_B Interrupt node 3 is selected
DTS	[15:14]	rw	**Interrupt Node Pointer for Interrupt DTS** This bit field defines the interrupt node, that is requested due to the set condition for bit INTSTAT.DTSI (if enabled by bit INTDIS.DTSI). 00_B Interrupt node 0 is selected 01_B Interrupt node 1 is selected 10_B Interrupt node 2 is selected 11_B Interrupt node 3 is selected

Field	Bits	Type	Description
0	[31:16]	rw	**Reserved** Should be written with 0.

SRC0
Service Request Control 0 Register (1FC$_H$) **Reset Value: 0000 0000$_H$**
SRC1
Service Request Control 1 Register (1F8$_H$) **Reset Value: 0000 0000$_H$**
SRC2
Service Request Control 2 Register (1F4$_H$) **Reset Value: 0000 0000$_H$**
SRC3
Service Request Control 3 Register (1F0$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
SRPN	[7:0]	rw	**Service Request Priority Number** 00_H Service request is never serviced 01_H Service request is on lowest priority ... FF_H Service request is on highest priority
TOS	10	rw	**Type of Service Control** 0_B CPU service is initiated 1_B PCP request is initiated
SRE	12	rw	**Service Request Enable** 0_B Service request is disabled 1_B Service request is enabled
SRR	13	rh	**Service Request Flag** 0_B No service request is pending 1_B A service request is pending

Field	Bits	Type	Description
CLRR	14	w	**Request Clear Bit** CLRR is required to clear SRR. 0_B No action 1_B Clear SRR; bit value is not stored; read always returns 0; no action if SETR is set also.
SETR	15	w	**Request Set Bit** SETR is required to set SRR. 0_B No action 1_B Set SRR; bit value is not stored; read always returns 0; no action if CLRR is set also.
0	[9:8], 11, [31:16]	r	**Reserved** Read as 0; should be written with 0.

3.11 NMI Trap Generation

The NMI trap structure is shown in **Figure 3-28**. The trap request trigger or the corresponding trap set bit (in register TRAPSET) can trigger the NMI trap generation. The trap flag can be cleared by software by writing to the corresponding bit in register TRAPCLR. A NMI request is only generated if the trap source was not disabled. Otherwise only the trap status flag is set but no NMI request is generated.

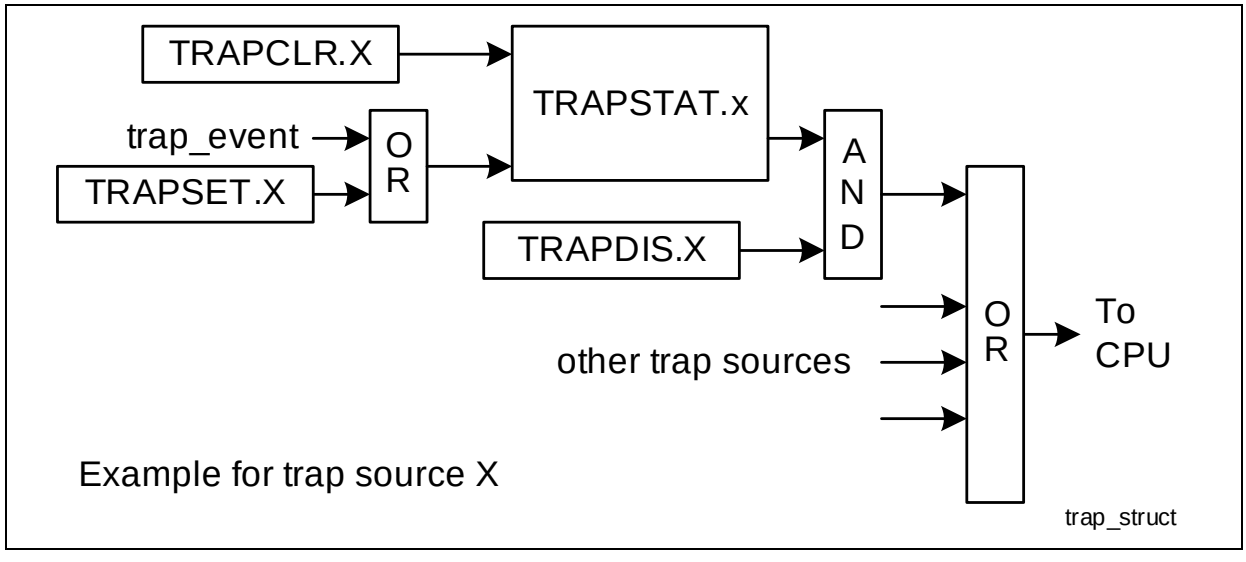


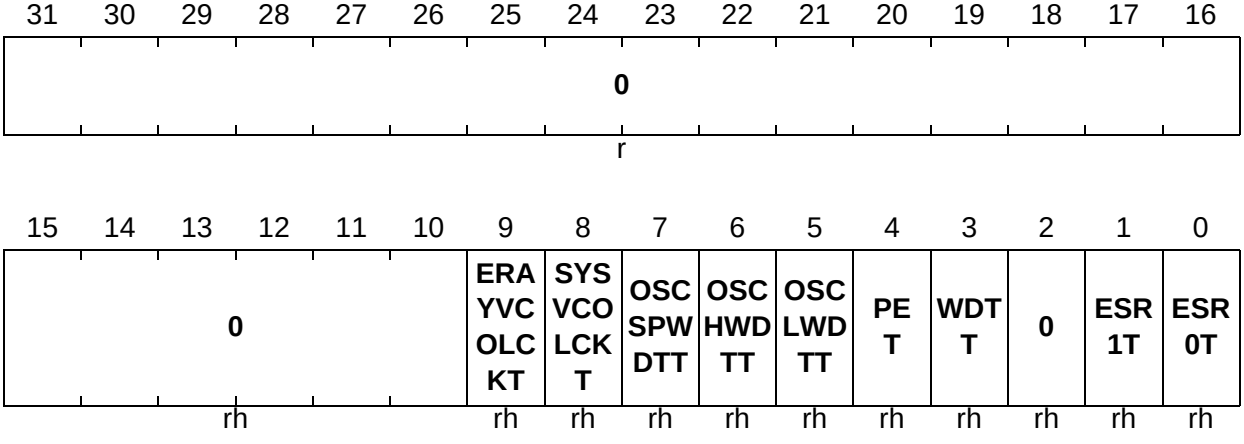
Figure 3-28 NMI Trap Generation

Handling NMI Traps

As an NMI trap is generated while the trap source is enable AND the trap status flag is set it is recommended to clear the trap status flag before the trap source is enabled. The trap status flag can be set before the trap source is enabled and simply enabling the trap source can result in unintended NMI traps. At the end of a NMI trap handing routine the trap status flag should be cleared.

3.11.1 Trap Control Registers

TRAPSTAT
Trap Status Register **(124_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
ESR0T	0	rh	**ESR0 Trap Request Flag** This bit is set if an ESR0 event is triggered and bit is TRAPDIS.ESR0T is cleared. 0_B No trap was requested since this bit was cleared the last time 1_B A trap was requested since this bit was cleared the last time This bit can be cleared by setting bit TRAPCLR.ESR0T. This bit can be set by setting bit TRAPSET.ESR0T.
ESR1T	1	rh	**ESR1 Trap Request Flag** This bit is set if an ESR1 event is triggered and bit is TRAPDIS.ESR1T is cleared. 0_B No trap was requested since this bit was cleared the last time 1_B A trap was requested since this bit was cleared the last time This bit can be cleared by setting bit TRAPCLR.ESR1T. This bit can be set by setting bit TRAPSET.ESR1T.

Field	Bits	Type	Description
WDTT	3	rh	**WDT Trap Request Flag** This bit is set if a WDT trap is indicated and bit is TRAPDIS.WDTT is cleared. 0_B No trap was requested since this bit was cleared the last time 1_B A trap was requested since this bit was cleared the last time This bit can be cleared by setting bit TRAPCLR.WDTT. This bit can be set by setting bit TRAPSET.WDTT.
PET	4	rh	**Parity Error Trap Request Flag** This bit is set if a memory parity error is indicated and bit is TRAPDIS.PET is cleared. 0_B No trap was requested since this bit was cleared the last time 1_B A trap was requested since this bit was cleared the last time This bit can be cleared by setting bit TRAPCLR.PET. This bit can be set by setting bit TRAPSET.PET.
OSCLWDTT	5	rh	**OSCWDT Low Trap Request Flag** This bit is set if a oscillator WDT of the PLL detects a low event and bit is TRAPDIS.OSCLWDTT cleared. 0_B No trap was requested since this bit was cleared the last time 1_B A trap was requested since this bit was cleared the last time This bit can be cleared by setting bit TRAPCLR.OSCLWDTT. This bit can be set by setting bit TRAPSET.OSCLWDTT.

Field	Bits	Type	Description
OSCHWDTT	6	rh	**OSCWDT High Trap Request Flag** This bit is set if a oscillator WDT of the PLL detects a high event and bit is TRAPDIS.OSCHWDTT cleared. 0_B No trap was requested since this bit was cleared the last time 1_B A trap was requested since this bit was cleared the last time This bit can be cleared by setting bit TRAPCLR.OSCHWDTT. This bit can be set by setting bit TRAPSET.OSCHWDTT.
OSCSPWDTT	7	rh	**OSCWDT Spike Trap Request Flag** This bit is set if a oscillator WDT of the PLL detects a spike event and bit is TRAPDIS.OSCSPWDTT cleared. 0_B No trap was requested since this bit was cleared the last time 1_B A trap was requested since this bit was cleared the last time This bit can be cleared by setting bit TRAPCLR.OSCSPWDTT. This bit can be set by setting bit TRAPSET.OSCSPWDTT.
SYSVCOLCKT	8	rh	**SYSVCOWDT Trap Request Flag** This bit is set if a PLL VCO Loss-of-Lock event is triggered and bit is TRAPDIS.SYSVCOLCKT cleared. 0_B No trap was requested since this bit was cleared the last time 1_B A trap was requested since this bit was cleared the last time This bit can be cleared by setting bit TRAPCLR.SYSVCOLCKT. This bit can be set by setting bit TRAPSET.SYSVCOLCKT.

Field	Bits	Type	Description
ERAYVCOLCKT	9	rh	**ERAYVCOWDT Trap Request Flag** This bit is set if a PLL_ERAY VCO Loss-of Oscillator Lock event is triggered and bit is TRAPDIS.ERAYXVCOLCKT cleared. 0_B No trap was requested since this bit was cleared the last time 1_B A trap was requested since this bit was cleared the last time This bit can be cleared by setting bit TRAPCLR.ERAYVCOLCKT. This bit can be set by setting bit TRAPSET.ERAYVCOLCKT.
0	2, [15:10]	rh	**Reserved** Read as 0. This bit can be cleared by bit TRAPCLR.[x]. This bit can be set by bit TRAPSET.[x]. *Note: x = 2, [15:10].*
0	[31:16]	r	**Reserved** Read as 0.

TRAPSET
Trap Set Register **(128_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						ERAYVCOLCKT	SYSVCOLCKT	OSCSPWDTT	OSCHWDTT	OSCLWDTT	PET	WDTT	0	ESR1T	ESR0T
w						w	w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
ESR0T	0	w	**Set Trap Request Flag ESR0T** Setting this bit set bit TRAPSTAT.ESR0T. Clearing this bit has no effect. Reading this bit returns always zero.
ESR1T	1	w	**Set Trap Request Flag ESR1T** Setting this bit set bit TRAPSTAT.ESR1T. Clearing this bit has no effect. Reading this bit returns always zero.
WDTT	3	w	**Set Trap Request Flag WDTT** Setting this bit set bit TRAPSTAT.WDTT. Clearing this bit has no effect. Reading this bit returns always zero.
PET	4	w	**Set Trap Request Flag PET** Setting this bit set bit TRAPSTAT.PET. Clearing this bit has no effect. Reading this bit returns always zero.
OSCLWDTT	5	w	**Set Trap Request Flag OSCLWDTT** Setting this bit set bit TRAPSTAT.OSCLWDTT. Clearing this bit has no effect. Reading this bit returns always zero.
OSCHWDTT	6	w	**Set Trap Request Flag OSCHWDTT** Setting this bit set bit TRAPSTAT.OSCHWDTT. Clearing this bit has no effect. Reading this bit returns always zero.
OSCSPWDTT	7	w	**Set Trap Request Flag OSCSPWDTT** Setting this bit set bit TRAPSTAT.OSCSPWDTT. Clearing this bit has no effect. Reading this bit returns always zero.
SYSVCOLCKT	8	w	**Set Trap Request Flag SYSVCOLCKT** Setting this bit set bit TRAPSTAT.SYSVCOLCKT. Clearing this bit has no effect. Reading this bit returns always zero.
ERAYVCOLCKT	9	w	**Set Trap Request Flag ERAYVCOLCKT** Setting this bit set bit TRAPSTAT.ERAYVCOLCKT. Clearing this bit has no effect. Reading this bit returns always zero.

Field	Bits	Type	Description
0	2, [15:10]	w	**Reserved** Read as 0; have to be written with 0.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

TRAPCLR
Trap Clear Register **(12C$_H$)** **Reset Value: 0000 0000$_H$**

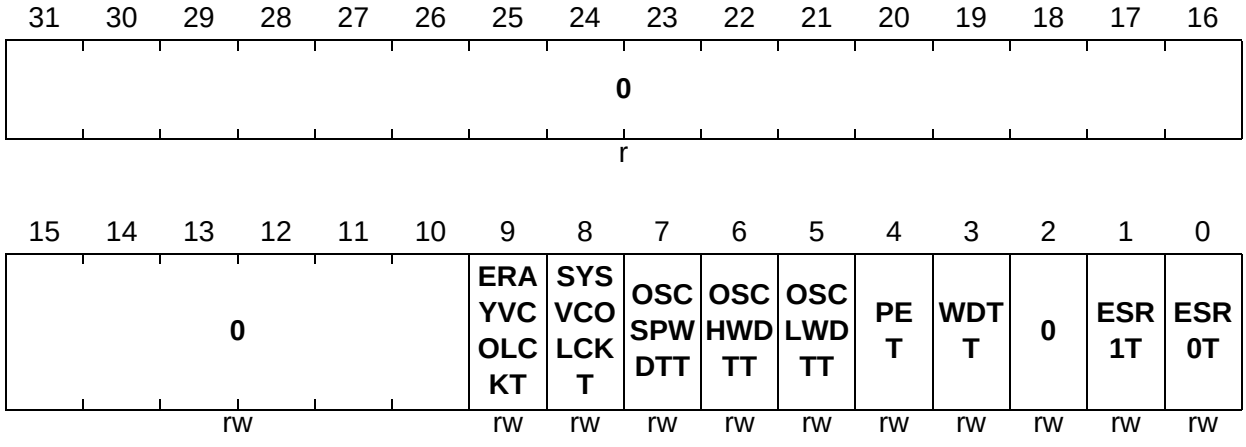
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						ERAYVCOLCKT	SYSVCOLCKT	OSCSPWDTT	OSCHWDTT	OSCLWDTT	PET	WDTT	0	ESR1T	ESR0T
w						w	w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
ESR0T	0	w	**Clear Trap Request Flag ESR0T** Setting this bit clears bit TRAPSTAT.ESR0T. Clearing this bit has no effect. Reading this bit returns always zero.
ESR1T	1	w	**Clear Trap Request Flag ESR1T** Setting this bit clears bit TRAPSTAT.ESR1T. Clearing this bit has no effect. Reading this bit returns always zero.
WDTT	3	w	**Clear Trap Request Flag WDTT** Setting this bit clears bit TRAPSTAT.WDTT. Clearing this bit has no effect. Reading this bit returns always zero.
PET	4	w	**Clear Trap Request Flag PET** Setting this bit clears bit TRAPSTAT.PET. Clearing this bit has no effect. Reading this bit returns always zero.

Field	Bits	Type	Description
OSCLWDTT	5	w	**Clear Trap Request Flag OSCLWDTT** Setting this bit clears bit TRAPSTAT.OSCLWDTT. Clearing this bit has no effect. Reading this bit returns always zero.
OSCHWDTT	6	w	**Clear Trap Request Flag OSCHWDTT** Setting this bit clears bit TRAPSTAT.OSCHWDTT. Clearing this bit has no effect. Reading this bit returns always zero.
OSCSPWDTT	7	w	**Clear Trap Request Flag OSCSPWDTT** Setting this bit clears bit TRAPSTAT.OSCSPWDTT. Clearing this bit has no effect. Reading this bit returns always zero.
SYSVCOLCKT	8	w	**Clear Trap Request Flag SYSVCOLCKT** Setting this bit clears bit TRAPSTAT.SYSVCOLCKT. Clearing this bit has no effect. Reading this bit returns always zero.
ERAYVCOLCKT	9	w	**Clear Trap Request Flag ERAYVCOLCKT** Setting this bit clears bit TRAPSTAT.ERAYVCOLCKT. Clearing this bit has no effect. Reading this bit returns always zero.
0	2, [15:10]	w	**Reserved** Read as 0; should be written with 0.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

TRAPDIS
Trap Disable Register **(130_H)** **Reset Value: 0000 FFFF$_H$**



Field	Bits	Type	Description
ESR0T	0	rw	**Disable Trap Request ESR0T** 0_B A trap request can be generated for this source 1_B No trap request can be generated for this source
ESR1T	1	rw	**Disable Trap Request ESR1T** 0_B A trap request can be generated for this source 1_B No trap request can be generated for this source
WDTT	3	rw	**Disable Trap Request WDTT** 0_B A trap request can be generated for this source 1_B No trap request can be generated for this source
PET	4	rw	**Disable Trap Request PET** 0_B A trap request can be generated for this source 1_B No trap request can be generated for this source
OSCLWDTT	5	rw	**Disable Trap Request OSCLWDTT** 0_B A trap request can be generated for this source 1_B No trap request can be generated for this source

Field	Bits	Type	Description
OSCHWDTT	6	rw	**Disable Trap Request OSCHWDTT** 0_B A trap request can be generated for this source 1_B No trap request can be generated for this source
OSCSPWDTT	7	rw	**Disable Trap Request OSCSPWDTT** 0_B A trap request can be generated for this source 1_B No trap request can be generated for this source
SYSVCOLCKT	8	rw	**Disable Trap Request SYSVCOLCKT** 0_B A trap request can be generated for this source 1_B No trap request can be generated for this source
ERAYVCOLCKT	9	rw	**Disable Trap Request ERAYVCOLCKT** 0_B A trap request can be generated for this source 1_B No trap request can be generated for this source
0	2, [15:10]	rw	**Reserved** Have to be written with 1.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

3.12 Miscellaneous System Control Register

This section collects different register that serve various system aspects.

3.12.1 GPTA Input IN1 Control

In the TC1797, the input line IN1 of the GPTA module can be used to measure the baud rate of an ASC0 or ASC1 receiver input signal with GPTA. This feature is controlled by SYSCON.GPTAIS.

Table 3-19 GPTA0/GPTA1/LTCA2 Input Line IN1 Connections

SYSCON.GPTAIS	GPTA0/GPTA1/LTCA2 Input IN1 Connected to
00_B	P2.9 / IN1 (default after Application Reset)
01_B	P5.0 / RXD0A
10_B	P6.8 / RXD0B
11_B	P6.10 / RXD1B

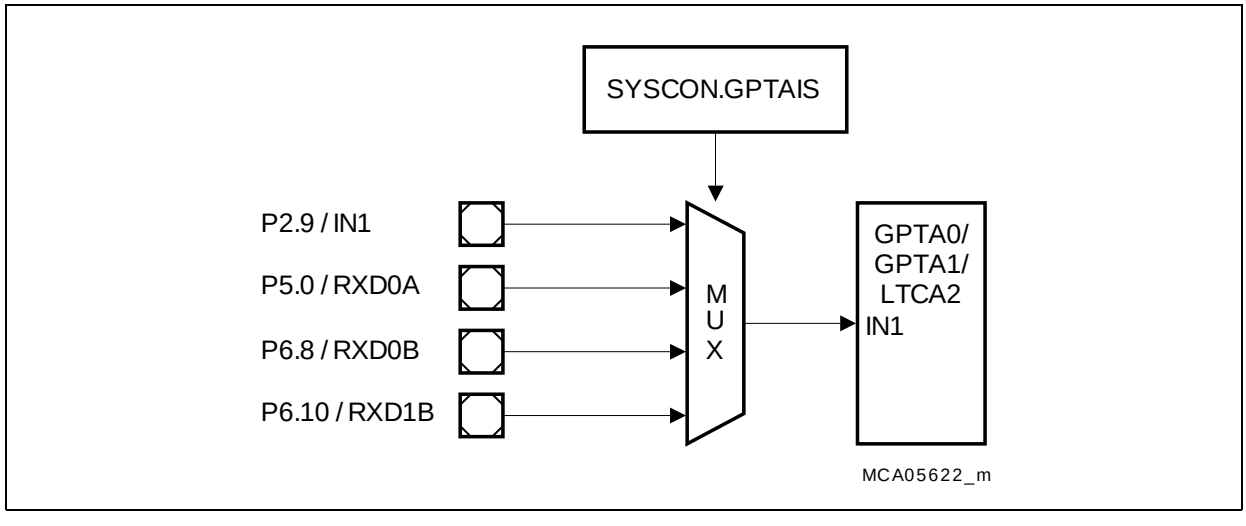
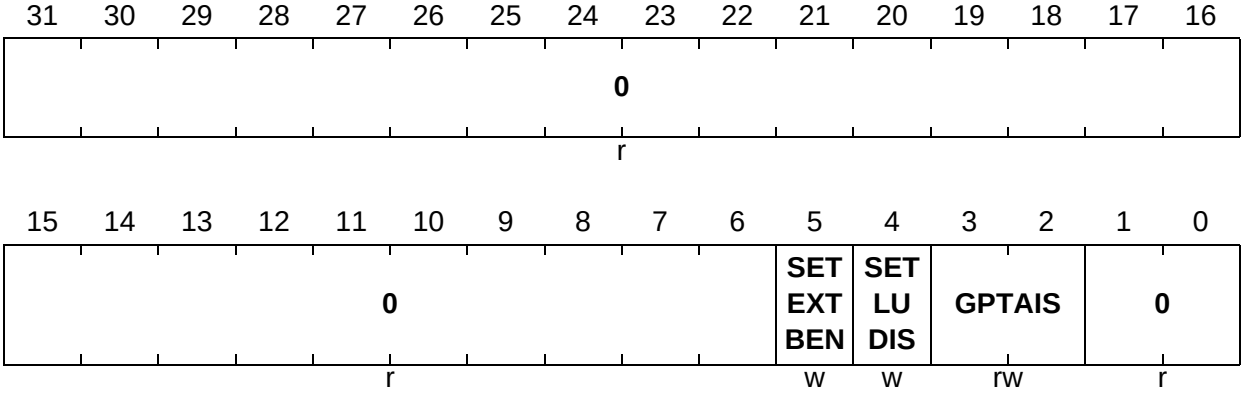


Figure 3-29 GPTA0/GPTA1/LTCA2 Input IN1 Control

3.12.2 System Control Register

This register controls various functionality used by the SCU but that are located outside of the module. Additionally some functions for other modules are included.

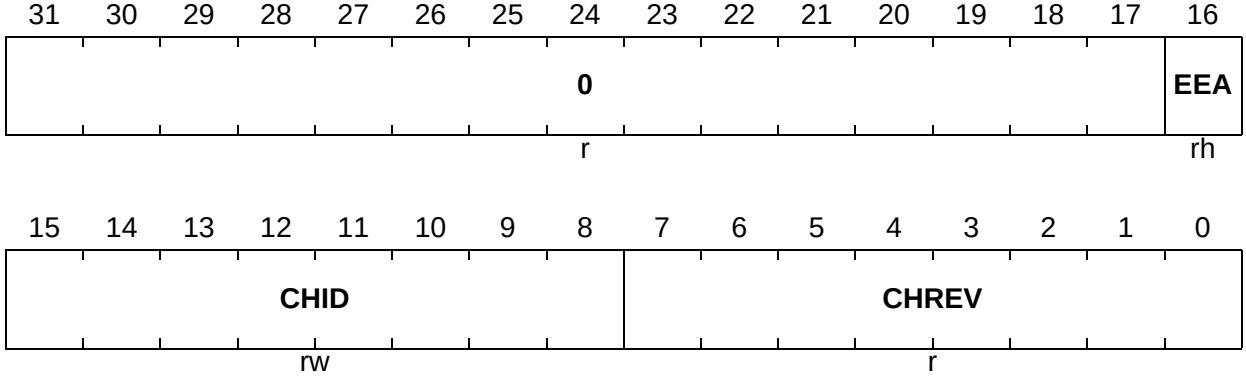
SYSCON
System Control Register (040_H) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
GPTAIS	[3:2]	rw	**GPTA Input Select** This bit field selects the input that is used for IN1 of the GPTA module. For more information see either **Section 3.12.1** or the GPTA chapter. 00_B IN0 is selected 01_B IN1 is selected 10_B IN2 is selected 11_B IN3 is selected
SETLUDIS	4	w	**Set Latch Update Disable** Setting this bit sets bit STSTAT.LUDIS. Clearing this bit has no effect. This bit reads always as zero.
SETEXTBEN	5	w	**Set External Boot Enable** Setting this bit sets bit STSTAT.EXTBEN. Clearing this bit has no effect. This bit reads always as zero.
0	[1:0], [31:6]	r	**Reserved** Read as 0; should be written with 0.

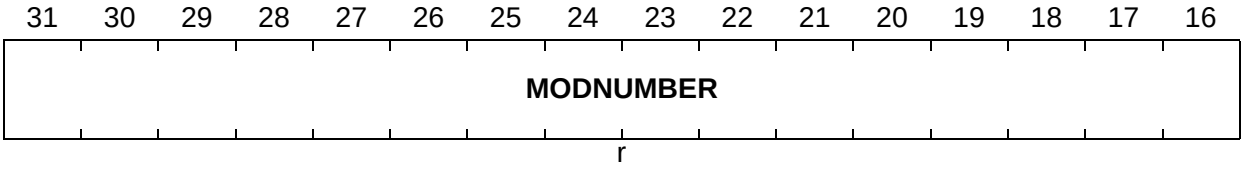
3.12.3 Identification Registers

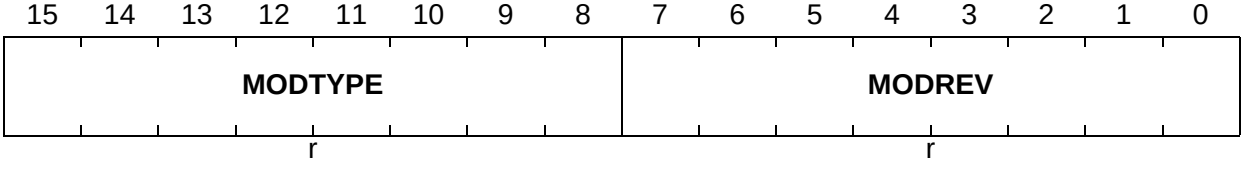
CHIPID
Chip Identification Register **(140_H)** **Reset Value: 0000 9101$_H$**



Field	Bits	Type	Description
CHREV	[7:0]	r	**Chip Revision Number** This bit field indicates the revision number of the TC1797 device (01_H = first revision). CHREV can be used e.g. for major step identification purposes. The value of this bit field is defined in the TC1797 Data Sheet.
CHID	[15:8]	rw	**Chip Identification Number** This bit field defines the product by a unique number. 91_H = 1797
EEA	16	rh	**Emulation Extension Available** Indicates if the emulation extension is available or not. 0_B EEC is not available 1_B EEC is available
0	[31:17]	r	**Reserved** Read as 0; should be written with 0.

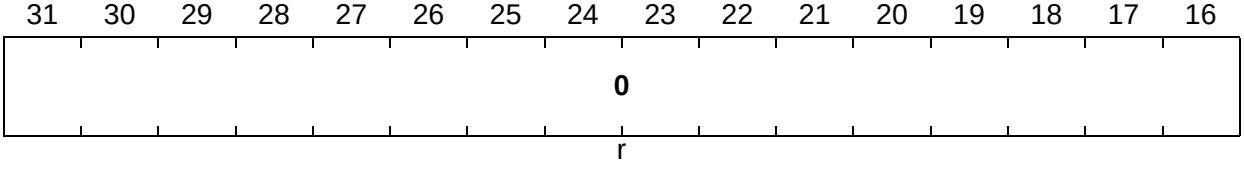
ID
Identification Register **(008_H)** **Reset Value: 0052 C0XX$_H$**

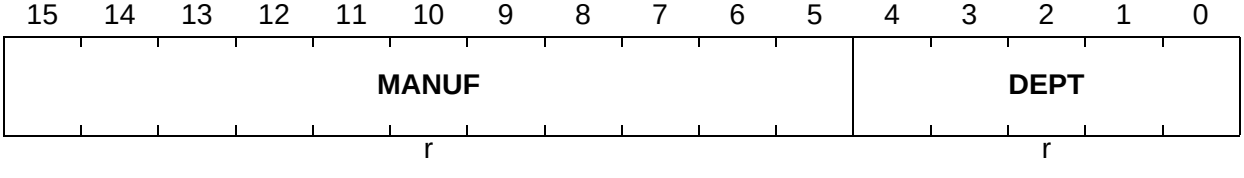




Field	Bits	Type	Description
MODREV	[7:0]	r	**Module Revision Number** This bit field indicates the revision number of the TC1797 module (01_H = first revision).
MODTYPE	[15:8]	r	**Module Type** This bit field is $C0_H$. It defines a 32-bit module
MODNUMBER	[31:16]	r	**Module Number** This bit field defines the module identification number.

MANID
Manufacturer Identification Register **(144_H)** **Reset Value: 0000 1820$_H$**





Field	Bits	Type	Description
DEPT	[4:0]	r	**Department Identification Number** = 00_H: indicates the Automotive & Industrial microcontroller department within Infineon Technologies.
MANUF	[15:5]	r	**Manufacturer Identification Number** This is a JEDEC normalized manufacturer code. MANUF = $C1_H$ stands for Infineon Technologies.
0	[31:16]	r	**Reserved** Read as 0.

The Redesign Tracing Register RTID provides a means of signalling minor redesigns that are not reflected in the CHIPID.CHREV bit field.

RTID
Redesign Tracing Identification Register

(148_H) **Reset Value: 0000 XXXX$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RT 15	**RT 14**	**RT 13**	**RT 12**	**RT 11**	**RT 10**	**RT 9**	**RT 8**	**RT 7**	**RT 6**	**RT 5**	**RT 4**	**RT 3**	**RT 2**	**RT 1**	**RT 0**
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Field	Bits	Type	Description
RTx **(x = 0-15)**	x	r	**Redesign Trace Bit x** 0_B No change indicated 1_B A change has been made (without changing bit field CHIPID.CHREV). RTx can be used, e.g., for minor redesign stepping identification purposes.
0	[31:16]	r	**Reserved** Read as 0

Note: The RTID reset value for a major design step (without modifications) is 0000_H.

3.12.4 SCU Kernel Registers

This section describes the kernel registers of the 32-bit SCU module. Most of 32-bit SCU kernel register names described in this section will be referenced in other parts of the TC1797 User´s Manual by the module name prefix “SCU_”.

SCU Kernel Register Overview

Table 3-20 Register Overview of SCU

Short Name	Long Name	Offset Addr.[1)]	Access Mode		Reset	Description See
			Read	Write		
–	Reserved	000_H - $00C_H$	BE	BE	–	–
OSCCON	OSC Control Register	010_H	U, SV	SV, E	System Reset	Page 3-29
PLLSTAT	PLL Status Register	014_H	U, SV	BE	System Reset	Page 3-31
PLLCON0	PLL Configuration 0 Register	018_H	U, SV	SV, E	System Reset	Page 3-32
PLLCON1	PLL Configuration 1 Register	$01C_H$	U, SV	SV, E	System Reset	Page 3-34
–	Reserved	020_H	BE	BE	–	–
PLLERAY CTR	PLL_ERAY Control and Status Register	020_H	U, SV	SV, E	System Reset	Page 3-35
–	Reserved	024_H - $02C_H$	BE	BE	–	–
CCUCON0	CCU Control Register 0	030_H	U, SV	SV, E	System Reset	Page 3-38
CCUCON1	CCU Control Register 1	034_H	U, SV	SV, E	System Reset	Page 3-40
FDR	Fractional Divider Register	038_H	U, SV	SV, E	System Reset	Page 3-44
EXTCON	External Clock Control Register	$03C_H$	U, SV	U, SV	System Reset	Page 3-42
SYSCON	System Control Register	040_H	U, SV	U, SV	System Reset	Page 3-170

Table 3-20 Register Overview of SCU

Short Name	Long Name	Offset Addr.[1)]	Access Mode		Reset	Description See
			Read	Write		
–	Reserved	044_H - $04C_H$	BE	BE	–	–
RSTSTAT	Reset Status Register	050_H	U, SV	BE	Power-on Reset	Page 3-67
RSTCNTCON	Reset Counter Control Register	054_H	U, SV	SV, E	Power-on Reset	Page 3-70
RSTCON	Reset CON Register	058_H	U, SV	SV, E	Power-on Reset	Page 3-70
ARSTDIS	Application Reset Disable Register	$05C_H$	U, SV	SV, E	Power-on Reset	Page 3-72
SWRSTCON	Software Reset Configuration Register	060_H	U, SV	SV, E	Power-on Reset	Page 3-73
–	Reserved	064_H - $06C_H$	BE	BE	–	–
ESRCFG0	ESR0 Configuration Register	070_H	U, SV	SV, E	System Reset	Page 3-76
ESRCFG1	ESR1 Configuration Register	074_H	U, SV	SV, E	System Reset	Page 3-76
–	Reserved	078_H - $07C_H$	BE	BE	–	–
EICR0	External Input Channel Register 0	080_H	U, SV	U, SV	Application Reset	Page 3-95
EICR1	External Input Channel Register 1	084_H	U, SV	U, SV	Application Reset	Page 3-98
EIFR	External Input Flag Register	088_H	U, SV	U, SV	Application Reset	Page 3-102
FMR	Flag Modification Register	$08C_H$	U, SV	U, SV	Application Reset	Page 3-102
PDRR	Pattern Detection Result Register	090_H	U, SV	U, SV	Application Reset	Page 3-103

Table 3-20 Register Overview of SCU

Short Name	Long Name	Offset Addr.[1)]	Access Mode		Reset	Description See
			Read	Write		
IGCR0	Interrupt Gating Register 0	094_H	U, SV	U, SV	Application Reset	Page 3-104
IGCR1	Interrupt Gating Register 1	098_H	U, SV	U, SV	Application Reset	Page 3-106
–	Reserved	$09C_H$	BE	BE	–	–
IOCR	Input/Output Control Register	$0A0_H$	U, SV	U, SV	System Reset	Page 3-77
OUT	Output Register	$0A4_H$	U, SV	U, SV	System Reset	Page 3-80
OMR	Output Modification Register	$0A8_H$	U, SV	U, SV	System Reset	Page 3-81
IN	Input Register	$0AC_H$	U, SV	BE	System Reset	Page 3-82
PMCSR	Power Management Control and Status Register	$0B0_H$	U, SV	U, SV	Application Reset	Page 3-112
–	Reserved	$0B4_H$ - $0BC_H$	BE	BE	–	–
STSTAT	Start-up Status Register	$0C0_H$	U, SV	BE	Power-on Reset	Page 3-115
STCON	Start-up Configuration Register	$0C4_H$	U, SV	ST	Power-on Reset	Page 3-117
–	Reserved	$0C8_H$ - $0CC_H$	BE	BE	–	–
PETCR	Parity Error Trap Control Register	$0D0_H$	U, SV	SV, E	Application Reset	Page 3-119
PETSR	Parity Error Trap Status Register	$0D4_H$	U, SV	BE	Application Reset	Page 3-122
–	Reserved	$0D8_H$ - $0DC_H$	BE	BE	–	–

Table 3-20 Register Overview of SCU

Short Name	Long Name	Offset Addr.[1)]	Access Mode		Reset	Description See
			Read	Write		
DTSSTAT	Die Temperature Sensor Status Register	$0E0_H$	U, SV	BE	Application Reset	Page 3-127
DTSCON	Die Temperature Sensor Control Register	$0E4_H$	U, SV	U, SV	Application Reset	Page 3-126
–	Reserved	$0E8_H$ - $0EC_H$	BE	BE	–	–
WDT_ CON0	WDT Control Register 0	$0F0_H$	U, SV	U, SV	Application Reset	Page 3-137
WDT_ CON1	WDT Control Register 1	$0F4_H$	U, SV	SV, E	Application Reset	Page 3-139
WDT_ SR	WDT Status Register	$0F8_H$	U, SV	BE	Application Reset	Page 3-141
–	Reserved	$0FC_H$	BE	BE	–	–
EMSR	Emergency Stop Register	100_H	U, SV	SV, E	Application Reset	Page 3-145
–	Reserved	104_H - $10F_H$	BE	BE	–	–
INTSTAT	Interrupt Status Register	110_H	U, SV	BE	Application Reset	Page 3-148
INTSET	Interrupt Set Register	114_H	U, SV	U, SV	Application Reset	Page 3-150
INTCLR	Interrupt Clear Register	118_H	U, SV	U, SV	Application Reset	Page 3-152
INTDIS	Interrupt Disable Register	$11C_H$	U, SV	U, SV	Application Reset	Page 3-153
INTNP	Interrupt Node Pointer Register	120_H	U, SV	U, SV	Application Reset	Page 3-155
TRAPSTAT	Trap Status Register	124_H	U, SV	BE	System Reset	Page 3-160
TRAPSET	Trap Set Register	128_H	U, SV	SV, E	System Reset	Page 3-163

Table 3-20 Register Overview of SCU

Short Name	Long Name	Offset Addr.[1]	Access Mode		Reset	Description See
			Read	Write		
TRAPCLR	Trap Clear Register	$12C_H$	U, SV	U, SV	System Reset	Page 3-165
TRAPDIS	Trap Disable Register	130_H	U, SV	SV, E	Application Reset	Page 3-167
–	Reserved	134_H - $13F_H$	BE	BE	–	–
CHIPID	Chip Identification Register	140_H	U, SV	ST	System Reset	Page 3-171
MANID	Manufacture Identification Register	144_H	U, SV	BE	System Reset	Page 3-172
RTID	Redesign Trace Identification Register	148_H	U, SV	BE	System Reset	Page 3-173
–	Reserved	$14C_H$ - $1EF_H$	BE	BE	–	–
SRC3	Service Request Control Register 3	$1F0_H$	U, SV	SV	Application Reset	Page 3-157
SRC2	Service Request Control Register 2	$1F4_H$	U, SV	SV	Application Reset	Page 3-157
SRC1	Service Request Control Register 1	$1F8_H$	U, SV	SV	Application Reset	Page 3-157
SRC0	Service Request Control Register 0	$1FC_H$	U, SV	SV	Application Reset	Page 3-157

1) The absolute register address is calculated as follows:
Module Base Address + Offset Address (shown in this column)

3.12.5 SCU Address Area

Table 3-21 Registers Address Space - SCU Kernel Registers

Module	Base Address	End Address	Note
SCU	$F000\ 0500_H$	$F000\ 06FF_H$	-

4 On-Chip System Buses and Bus Bridges

The TC1797 has two independent on-chip buses:

- Local Memory Bus (LMB)
- System Peripheral Bus (SPB)

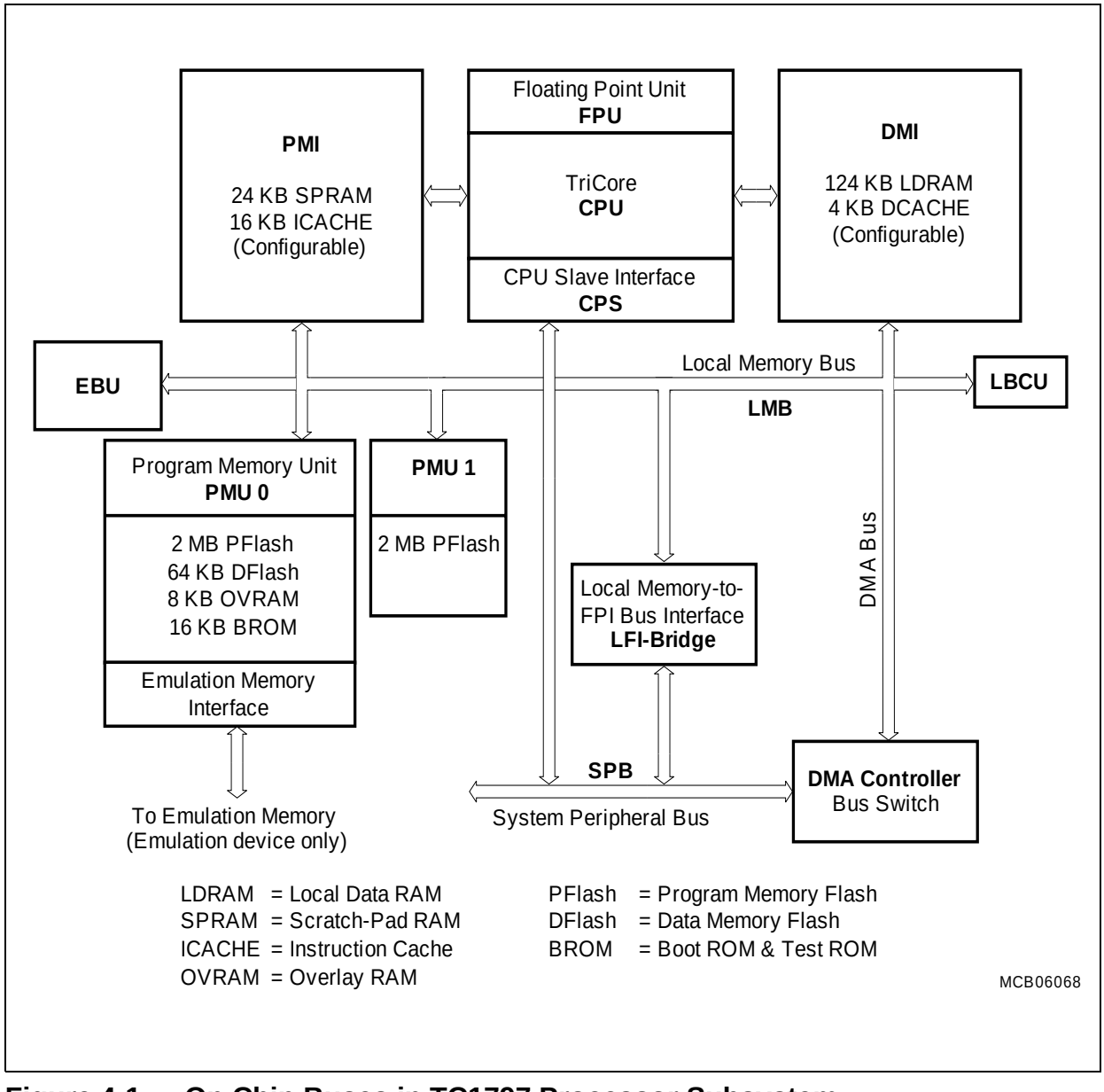


Figure 4-1 On Chip Buses in TC1797 Processor Subsystem

The LMB connects the TriCore CPU to its local resources for instruction fetches and data accesses.

The SPB is accessible by the CPU via the LFI Bridge.

4.1 What is new

Major differences of the AudoFuture On Chip Bus System architecture compared to AudoNG:

- The TC1797 is based on two on chip busses (LMB, SPB). The remote peripheral bus (RPB) was removed.
- The DMA is additionally connected to the LMB bus with a master interface.
- The E-Ray module is additionally connected to the SPB with a slave interface.
- The OCDS module is connected to the DMA. The OCDS module has direct access to the LMB/SPB bus via the DMA-LMB and DMA-SPB master interface.
- DMA decides to forward transactions from DMA internal sources (Move Engine 0/1, MLI0/1 and Cerberus) to LMB/SPB by internal address decoding (see address map chapter).
- The DMA controller can generate single data read and write transactions on the LMB bus. The DMA does not forward directly transactions from the FPI bus to the LMB bus or vice versa (no LMB<->FPI bridge functionality). Further DMA details and/or DMA changes compared to AudoNG can be found in the DMA chapter.
- The DMA module is now able to access the LMB / SPB bus with three priorities (see **Page 4-6** and **Page 4-26**). Priority of DMA access is controlled by the OCDS (for DMA-OCDS accesses) and by the Move Engine Channels for DMA-Move Engine accesses. A detailed description of the DMA internal control of the priorities can be found in the DMA chapter.
- The LMB-to-FPI bridge (LFI) address translation table was adapted to the AudoFuture address space (**Page 4-16**).
- Some clarifications where added (example: Figure of the BCU Breakpoint Trigger Combination Logic, **Page 4-34**).
- The SBCU_DBDAT register was introduced. (see **Page 4-57**).
- The table On Chip Bus Master TAG Assignments was adapted (**Page 4-59**).

4.2 Local Memory Bus

The following terminology is used for the bus:

Table 4-1 LMB Bus Terms

Term	Description
Agent	An LMB agent is any master or slave device which is connected to the LMB Bus.
Master	An LMB master device is an LMB agent which is able to initiate transactions on the LMB. It is also able to react as an LMB slave.
Slave	An LMB slave device is an LMB agent which is not able to initiate transactions on the LMB. It is only able to handle operations that are dedicated to it by an LMB master device.

4.2.1 Overview

The LMB is a synchronous and pipelined bus with variable block size transfer support. The protocol supports 8-, 16-, 32-, and 64-bit single transactions and 2/4 wide 64-bit block transfers.

The LMB has the following features:

- Optimized for high speed and high performance data transfers
- 32-bit address bus, 64-bit data bus
- Simple central arbitration per cycle
- Slave-controlled wait state insertion
- Address pipelining (max. depth - 2)
- Support of atomic operations LDMST, ST.T and SWAP.W
- Block transfers with variable block length (two or four 64-bit data transactions)

4.2.2 Transaction Types

There are three transaction types of the LMB.

4.2.2.1 Single Transfers

Single transfers are all transactions that are initiated by any instruction (code or data) of the TriCore 1 CPU and that require a system resource which is not part of the TriCore 1 PMI or DMI. The only exceptions are the following instructions:

- LDMST, ST.T and SWAP.W generate atomic transfers
- Cache miss instructions generate block transfers

4.2.2.2 Block Transfers

Block transfers are only issued in the following ways:

1. By the PMI and DMI in case of a cache miss.
2. By the PCP if it uses a BCOPY instruction.

Block transfers work in the same way as single transfers, except that only one address phase with consecutive two or four data phases is generated.

4.2.2.3 Atomic Transfers

Atomic transfers are initiated by instructions that require two single transfers (e.g. read-modify-write instructions such as LDMST, ST.T and SWAP.W). During an atomic transfer any other LMB master is blocked for gaining bus ownership.

4.2.3 Address Alignment Rules

Depending on the data size, there are rules that determine the address alignment of an LMB transfer.

1. Byte accesses must always be located on byte address boundaries.
2. Half-word accesses must be aligned to addresses with address line A0 = 0.
3. Word accesses must be aligned to addresses with address lines A[1:0] = 00_B.
4. Double-word accesses must be aligned to addresses with address lines A[2:0] = 000_B.
5. Block transfers must be aligned identical as double-word addresses.

4.2.4 Reaction of a Busy Slave

If an LMB slave is busy at an incoming LMB transaction request, it can delay the execution of the LMB transaction. The requesting LMB master releases the LMB for one cycle after the LMB transaction request in order to allow the LMB slave to indicate if it is ready to handle the requested LMB transaction.

Note: For the LMB default master, the one cycle gap does not result in a performance loss because it is granted the LMB in this cycle as default master if no other master request the LMB for some other reasons.

4.2.5 LMB Basic Operation

Figure 4-2 describes some basic bus operations of the LMB.

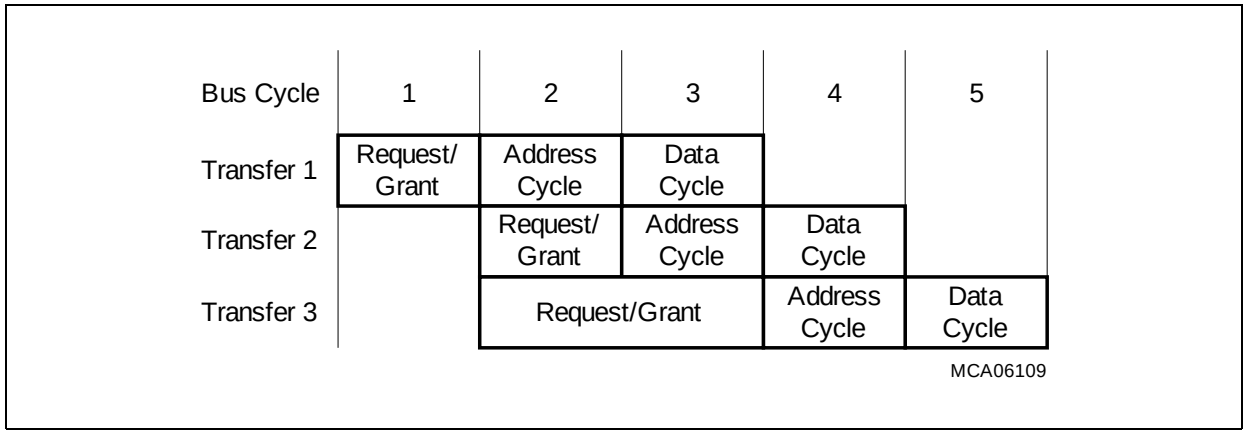


Figure 4-2 Basic LMB Transactions

Transfer 1 displays the three cycles of any LMB transaction:

1. **Request/Grant Cycle:** The LMB master wants to perform a read or write transfer and requests for the LMB. If the LMB is available, it is granted in the same cycle by the LMB controller.
2. **Address Cycle:** After the request/grant cycle the master puts the address on the LMB and all LMB slave devices check whether they are addressed for the following data cycle.
3. **Data Cycle:** In the data cycle either the LMB master puts write data on the LMB which is read by the LMB slave (write cycle) or vice versa (read cycle).

Transfers 2 and 3 show the conflict when two masters try to use the LMB, and how the conflict is resolved. In the example, the LMB master of transfer 2 has a higher priority than the LMB master of transfer 3.

During a block transfer, the address cycle of a second transfer is extended until the data cycles of the block transfer are finished. In the example shown in **Figure 4-3**, transfer 1 is a block transfer while transfer 2 is a single transfer.

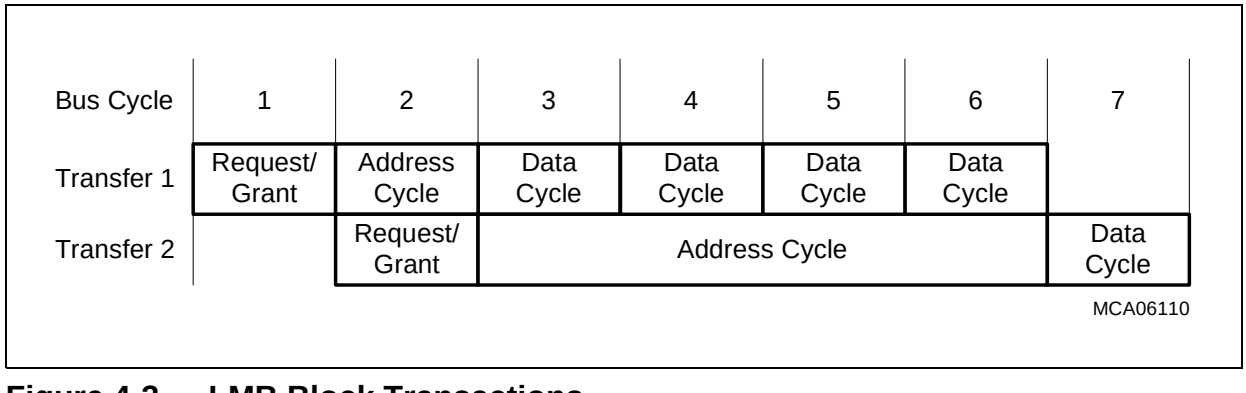


Figure 4-3 LMB Block Transactions

4.3 Local Memory Bus Controller Unit

The LMB in the TC1797 has an LMB Bus Control Unit (LBCU).

4.3.1 Basic Operation

The LBCU handles the cycle sequences of the transfers which have been requested by the LMB master devices. The LBCU is also able to detect bus protocol violations and addressing of un-implemented addresses. In case of a bus error, the LBCU captures all relevant data like bus address, bus data and bus status information in register where the information can be analyzed by software.

4.3.2 LMB Bus Arbitration

All LMB master devices requesting the LMB will participate in an arbitration round. Arbitration rounds are performed in each cycle that precedes a possible address cycle. Each LMB master device has a fixed priority as shown in **Table 4-2**.

Table 4-2 Priority of Master LMB Agents

Priority	LMB Master	Comment
Highest	DMA, high priority	DMA Requests from modules: - Cerberus high priority[1)] - DMA channels with high priority[2)]
	LFI Bridge	–
	DMA, medium priority	DMA Requests from modules: - DMA Channel medium priority[2)]
	Data Memory Interface (DMI)	Default Master
	Program Memory Interface (PMI)	–
Lowest	DMA, low priority	DMA requests from modules: - Cerberus low priority[1)] - DMA channels low priority[2)] - MLI

1) Priority of Cerberus transaction at the On Chip Busses is defined by the register bit IOCONF.FPI_PRIO. The register is defined in the OCDS chapter.

2) Priority of a DMA channel is defined by the corresponding CHCRmn.DMAPRIO bits. The registers are defined in the DMA chapter.

For all the masters requesting the LMB during any one cycle, the granted master is the one with the highest priority.

4.3.2.1 LMB Bus Default Master

When no LMB master is requesting the LMB, it is granted to the LMB default master. This means, if the default master needs the LMB in the next cycle, it can enter the address cycle without running through a request/grant cycle.

4.3.3 LMB Bus Error Handling

When an error occurs on LMB, the LMB controller captures and stores data about the erroneous condition and generates a service request if enabled to do so. The error conditions that force an error capture event are:

- Un-implemented Address: No LMB slave responds to an address target
- Error Acknowledge: An LMB slave responds with an error to a transaction

When a transaction causes an error, the address and data phase signals of the transaction causing the error are captured and stored in the following registers:

- The LMB Error Address Register (LEADDR) stores the LMB address that has been captured during the last erroneous LMB transaction.
- The LMB Error Data Registers (LEDATL/LEDATH) stores the LMB data bus information that has been captured during the last erroneous LMB transaction.
- The LMB Error Attribute Register (LEATT) stores status information of the bus error event.

If more than one LMB transaction generates a bus error, only the first bus error is captured. After a bus error was captured, the capture mechanism must be released again by software.

If a write transaction from the PCP or DMA on the SPB that is forwarded by the LFI module to the LMB causes a bus error on the LMB Bus, the originating masters are not informed about this bus error because these write transactions are finished on SPB when the error happens on LMB. With each bus error-capture event, a service request is generated and interrupt can be generated if enabled and configured in the corresponding service request register.

4.3.4 LMB Bus Control Unit Registers

Figure 4-4 and Table 4-4 are showing the address maps with all registers of LMB Bus Control Unit (LBCU) module.

LBCU Unit Register Overview

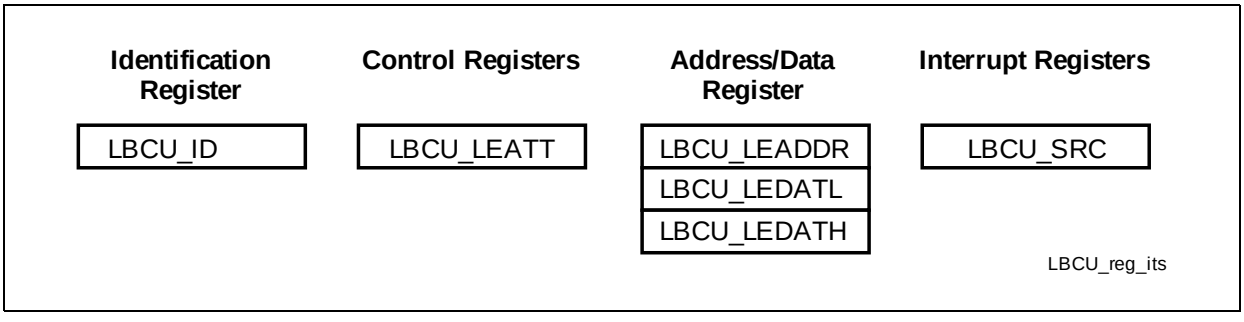


Figure 4-4 LMB Bus Control Unit Registers

Note: Addresses listed in column "Offset Address" of Table 4-4 are word (32-bit) addresses.

Table 4-3 Registers Address Space - LBCU Register Address Space

Module	Base Address	End Address	Note
LBCU	$F87F\ FE00_H$	$F87F\ FEFF_H$	

Table 4-4 Registers Overview - LBCU Module Control Registers

Short Name	Description	Offset Addr.[1)]	Access Mode		Reset Class	Description See
			Read	Write		
-	Reserved	000_H - 004_H	BE	BE	-	-
LBCU_ID	LBCU Module Identification Register	008_H	U, SV	BE	-	Page 4-10
-	Reserved	$00C_H$ - $01C_H$	BE	BE	-	-
LBCU_LEATT	LBCU LMB Error Attribute Register	020_H	U, SV, 32	SV, 32	3	Page 4-11
LBCU_LEADDR	LBCU LMB Error Address Register	024_H	U, SV	BE	3	Page 4-13

Table 4-4 Registers Overview - LBCU Module Control Registers

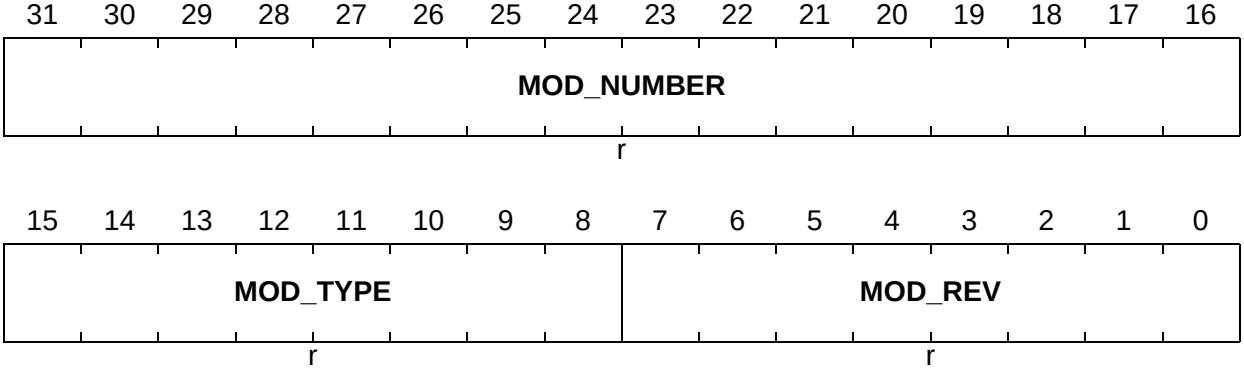
Short Name	Description	Offset Addr.[1]	Access Mode		Reset Class	Description See
			Read	Write		
LBCU_LEDATL	LBCU LMB Error Data Low Register	028_H	U, SV, 64	BE	3	Page 4-14
LBCU_LEDATH	LBCU LMB Error Data High Register	$02C_H$	U, SV	BE	3	Page 4-14
-	Reserved	030_H - $0F8_H$	BE	BE	-	-
LBCU_SRC	LBCU Service Request Control Register	$0FC_H$	U, SV, 32	SV, 32	3	Page 4-15

1) The absolute register address is calculated as follows:
Module Base Address (Table 4-3) + Offset Address (shown in this column)

4.3.4.1 LMB Bus Control Unit Control Registers

The identification register allows the programmer version-tracking of the module. The table below shows the identification register which is implemented in the LBCU module.

LBCU_ID
Module Identification Register (008_H) Reset Value: $000F\ C0XX_H$



Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision Number** This bit field defines the module revision number. The value of a module revision starts with 01_H (first revision).
MOD_TYPE	[15:8]	r	**Module Type** The bit field is set to $C0_H$ which defines the module as a 32-bit module.
MOD_NUMBER	[31:16]	r	**Module Number Value** This bit field defines a module identification number. The value for the LBCU module is $000F_H$.

LBCU_LEATT

LBCU LMB Error Attribute Register (020_H) Reset Value: XXXX XXX0$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OPC				0	TAG			RD	WR	SVM	0	UIS	ACK		
rh				r	rh			rh	rh	rh	r	rh	rh		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LOC	NOS	0						FPITAG				0			LEC
rh	rh	r						rh				r			rwh

Field	Bits	Type	Description
LEC	0	rwh	**Lock Error Capture** This bit indicates and controls whether the error-capture mechanism is unlocked or locked. 0_B The error-capture mechanism is unlocked. The next LMB Bus error will be captured. 1_B The error-capture mechanism is locked. The registers LEADDR and bits [31:4] of LEATT contain valid data. The registers LEADTL and LEDATH contain valid data if LEATT.UIS = ´0´. LEC is automatically set when an LMB bus error has been captured. Any further LMB bus error is not captured if LEC = 1. When writing a 1 to LEC, the error-capture mechanism becomes unlocked and is ready for the next LMB bus error-capture event.
FPITAG	[7:4]	rh	**FPI Bus Master TAG** This bit field indicates the FPI Bus master tag in case of an LMB bus error. Note that the FPI Bus master tag is only of interest if the erroneous LMB transfer was initiated by the DMA or the PCP..
NOS	14	rh	**LMB No Split** This bit indicates the state of the lmb_no_split_n signal in case of an LMB bus error.

Field	Bits	Type	Description
LOC	15	rh	**LMB Bus Lock State** This bit indicates the bus lock state in case of an LMB bus error. 0_B LMB bus error occurred at an atomic transfer. 1_B LMB bus error occurred at a single or block transfer.
ACK	[18:16]	rh	**LMB Bus Slave Response State** This bit indicates status information of the LMB slave device in case of an LMB bus error. 000_B Slave is in normal operation. 010_B Slave is busy. 011_B Slave has an error encountered. OthersReserved
UIS	19	rh	**Un-implemented Address** This bit indicates whether the LMB bus error occurred by an un-implemented address. 0_B LMB slave address is valid. 1_B Invalid LMB slave address occurred.
SVM	21	rh	**LMB Bus Supervisor Mode** This bit indicates whether the LMB bus error occurred in Supervisor Mode or in User Mode. 0_B Transfer was initiated in Supervisor Mode. 1_B Transfer was initiated in User Mode.
WR	22	rh	**LMB Bus Write Error Indication** This bit indicates whether the LMB bus error occurred at a write cycle (see **Table 4-5**).
RD	23	rh	**LMB Bus Read Error Indication** This bit indicates whether the LMB bus error occurred at a read cycle (see **Table 4-5**).
TAG	[26:24]	rh	**LMB Master TAG[1)]** This bit field indicates the LMB master device in case of an LMB bus error (see **Table 4-15**). 000_B LFI 010_B PMI 100_B DMI 101_B DMA Other bit combinations are reserved.

Field	Bits	Type	Description
OPC	[31:28]	rh	**LMB Bus Error Transaction Type** This bit field indicates the type of transfer at which the LMB bus error occurred. 0000_B 8-bit data single transfer 0001_B 16-bit data single transfer 0010_B 32-bit data single transfer 0011_B 64-bit data single transfer 1000_B 2 * 64-bit data block transfer 1001_B 4 * 64-bit data block transfer OthersReserved .
0	[3:1], [13:8], 20, 27	r	**Reserved** Read as 0; should be written with 0.

1) Pls. note that this bit field represents bit 0-2 of the master TAG as shown in **Table 4-15**). This as bit 3 of the On Chip Bus master TAGs is always 0 for master interfaces connected to the LMB Bus.

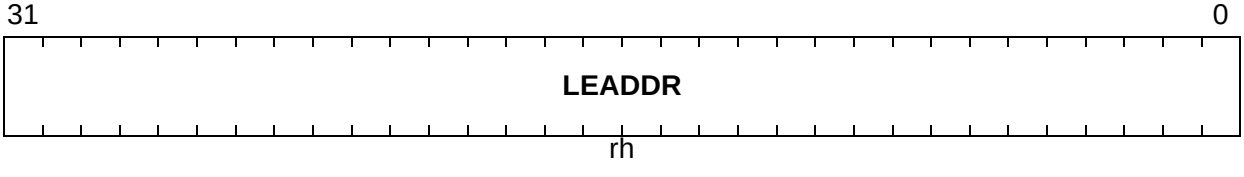
Note: LEATT[31:4] contains valid read data only if its bit LEC is set.

Table 4-5 LMB Bus Read/Write Error Indication

RD	WR	LMB Bus Cycle
0	0	LMB bus error occurred at the read cycle of an atomic transfer.
0	1	LMB bus error occurred at a read cycle of a single transfer.
1	0	LMB bus error occurred at a write cycle of a single transfer or at the write cycle of an atomic transfer.
1	1	Does not occur.

LBCU_LEADDR

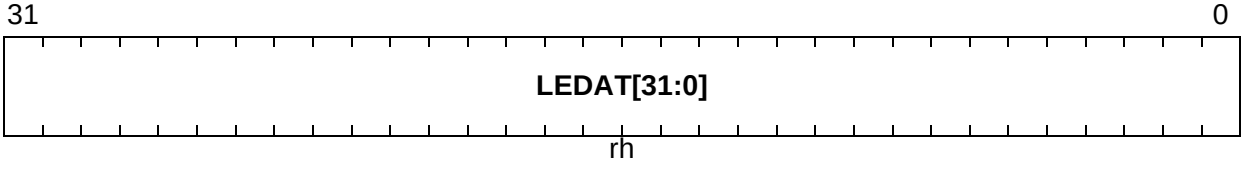
LBCU LMB Error Address Register (024_H) Reset Value: XXXX XXXX$_H$



Field	Bits	Type	Description
LEADDR	[31:0]	rh	**LMB Bus Address** This bit field holds the LMB address that has been captured at an LMB error. LEADDR only contains valid read data when bit LEC in the corresponding register LEATT is set.

LBCU_LEDATL

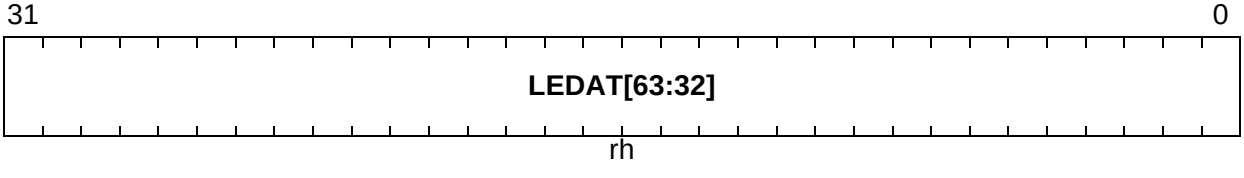
LBCU LMB Error Data Low Register (028$_H$) **Reset Value: XXXX XXXX$_H$**



Field	Bits	Type	Description
LEDAT[31:0]	[31:0]	rh	**LMB Bus Address Bits [31:0]** This bit field holds the lower 32-bit part of the 64-bit LMB data that has been captured at an LMB bus error. LEDAT[31:0] only contains valid read data when bit LEC in the corresponding register LEATT is set.

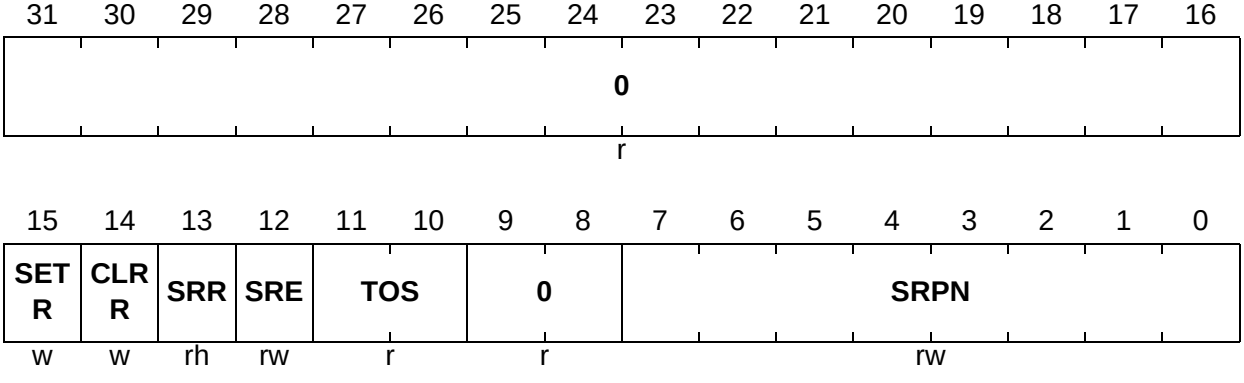
LBCU_LEDATH

LBCU LMB Error Data High Register (02C$_H$) **Reset Value: XXXX XXXX$_H$**



Field	Bits	Type	Description
LEDAT[63:32]	[31:0]	rh	**LMB Bus Address Bits [31:0]** This bit field holds the upper 32-bit part of the 64-bit LMB data that has been captured at an LMB bus error. LEDAT[63:32] only contains valid read data when bit LEC in the corresponding register LEATT is set.

LBCU_SRC
LBCU Service Request Control Register
($0FC_H$) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
SRPN	[7:0]	rw	**Service Request Priority Number**
TOS	[11:10]	r	**Type-of-Service State** Always read as 00_B. This means type-of-service is associated with interrupt bus 0 (CPU interrupt arbitration bus).
SRE	12	rw	**Service Request Enable**
SRR	13	rh	**Service Request Flag**
CLRR	14	w	**Request Flag Clear Bit**
SETR	15	w	**Request Flag Set Bit**
0	[9:8], [31:16]	r	**Reserved** Read as 0; should be written with 0.

4.4 Local Memory Bus to FPI Bus Interface (LFI Bridge)

This section describes the basic functionality of the LFI Bridge.

4.4.1 Functional Overview

The LFI Bridge is a bi-directional bus bridge between the LMB and the System Peripheral FPI Bus (SPB). The bridge supports all transactions types of both the LMB Bus and FPI Bus.

The bridge is not direction-transparent, this means that the master TAG of a bus master is not forwarded to the other side of the bridge and is replaced instead by the master TAG of the LFI Bridge itself.

In order to avoid deadlocks, priority is given to transactions initiated by the DMA or the PCP.

The bridge supports the pipelining of both connected buses. Therefore, no additional delay is created except for bus protocol conversions.

Address Translation

Addresses of SPB transfers (initiated by the DMA or the PCP) via the LFI Bridge that address an LMB slave device, are translated into an LMB address according to **Table 4-6**.

Table 4-6 SPB to LMB Bus Address Translation

Transaction Destination	SPB Access Range	Translated LMB Address
Reserved	$E800\ 0000_H$ - $E83F\ FFFF_H$	$C800\ 0000_H$ - $C83F\ FFFF_H$
DMI LDRAM	$E840\ 0000_H$ - $E841\ FFFF_H$	$D000\ 0000_H$ - $D001\ FFFF_H$
Reserved	$E842\ 0000_H$ - $E84F\ FFFF_H$	$D002\ 0000_H$ - $D00F\ FFFF_H$
PMI SPRAM	$E850\ 0000_H$ - $E850\ 9FFF_H$	$C000\ 0000_H$ - $C000\ 9FFF_H$
Reserved	$E850\ A000_H$ - $E85F\ FFFF_H$	$C000\ A000_H$ - $C000\ FFFF_H$

Bus Errors at Writes via the LFI Bridge

When a write operation has been initiated and directed to the LFI Bridge by an SPB bus master, the LFI Bridge handles the write transaction at the LMB autonomously. If the write operation at the LMB results in a bus error, the LBCU detects the bus error and generates an LMB bus error interrupt. There is no bus error generated at the SPB side in this case because of the posted nature of a write operation.

The equivalent behavior occurs when an LMB master initiates a write to an SPB slave device. In this case, SPB bus errors are detected by the SBCU but not at the LMB side.

Note that this behavior occurs only at write operations via the LFI Bridge. It can also be triggered by an erroneous write cycle of a read-modify-write bus transaction.

4.4.2 LMB to FPI Bridge Control Registers

Table 4-7 and Table 4-8 are showing the address maps with all registers of the LMB to FPI Bridge (LFI) module.

LFI Register

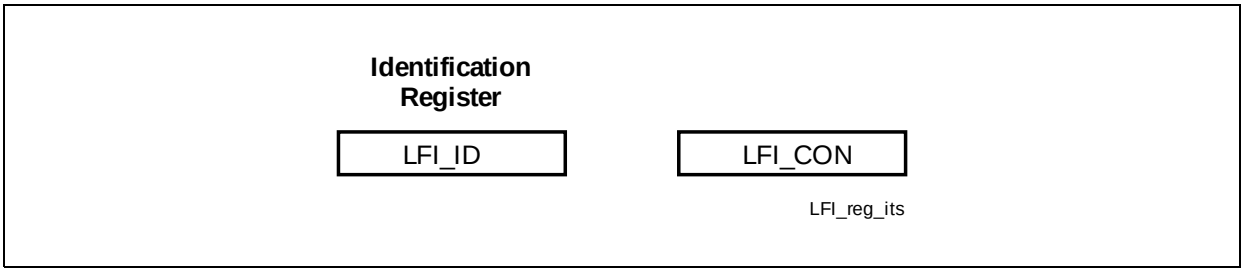


Figure 4-5 LFI Register

Note: Addresses listed in column "Address" of Table 4-8 are word (32-bit) addresses.

Table 4-7 Registers Address Space - LFI Bridge

Module	Base Address	End Address	Note
LFI	F87F FF00$_H$	F87F FFFF$_H$	

Table 4-8 Registers Overview - LFI Bridge Module Control Registers

Short Name	Description	Offset Addr.[1]	Access Mode		Reset Class	Description See
			Read	Write		
–	Reserved	000$_H$ - 004$_H$	BE	BE	-	-
LFI_ID	LFI Module Identification Register	008$_H$	U, SV	BE	-	Page 4-19
–	Reserved	00C$_H$	BE	BE	-	-
LFI_CON	LFI Configuration Register	010$_H$	U, SV	SV	3	Page 4-20
–	Reserved	014$_H$ - 0F4$_H$	BE	BE	-	-
–	Reserved	0F8$_H$	nBE[2]	nBE[2]	-	-
–	Reserved	0FC$_H$ - 0FF$_H$	BE	BE	-	-

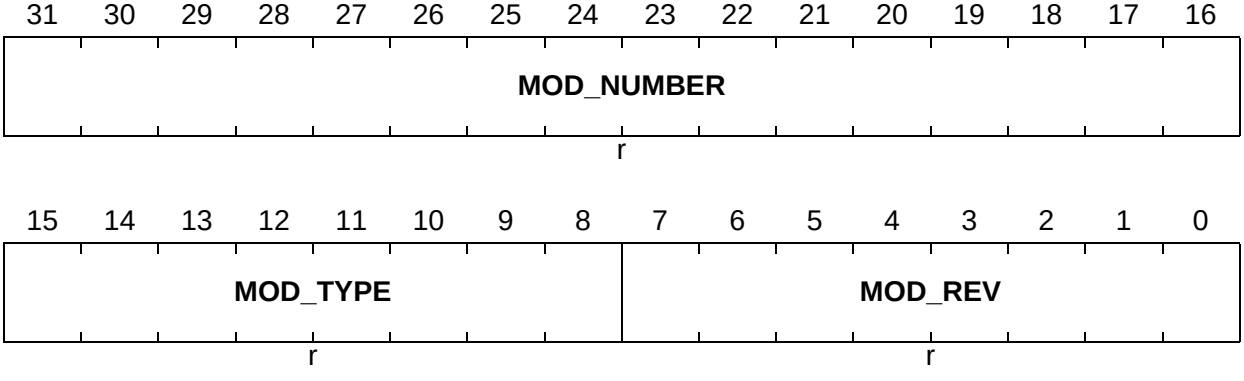
1) The absolute register address is calculated as follows:
Module Base Address (Table 4-7) + Offset Address (shown in this column)

2) Read as 0. Should not be written. If it is written, it must be written with 0.

4.4.2.1 LFI Register Description

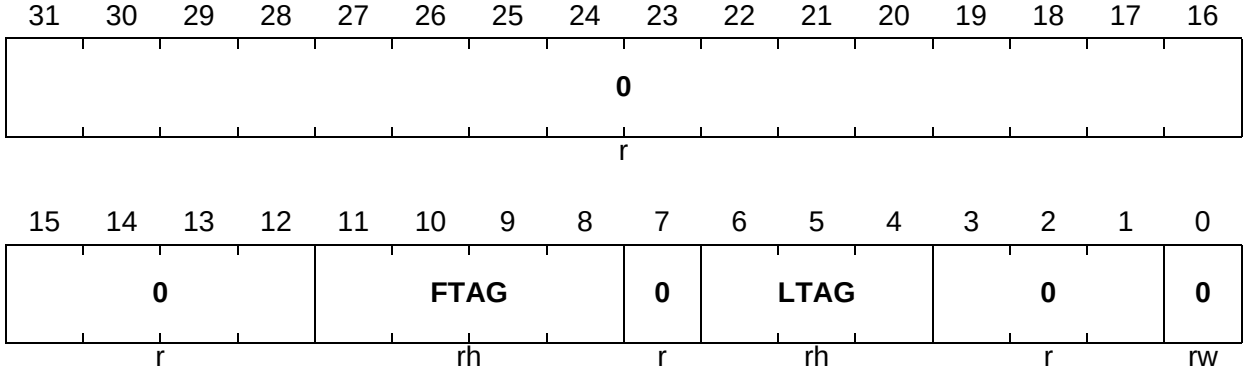
The identification register allows the programmer version-tracking of the module. The table below shows the identification register which is implemented in the LFI module.

LFI_ID
Module Identification Register (008_H) Reset Value: $000C\ C0XX_H$



Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision Number** This bit field defines the module revision number. The value of a module revision starts with 01_H (first revision).
MOD_TYPE	[15:8]	r	**Module Type** The bit field is set to $C0_H$ which defines the module as a 32-bit module.
MOD_NUMBER	[31:16]	r	**Module Number Value** This bit field defines a module identification number. The value for the LFI module is $000C_H$.

LFI_CON
LFI Configuration Register **(010_H)** **Reset Value: 0000 0B00$_H$**



Field	Bits	Type	Description
0	0	rw	**Reserved** Returns 0 if read; must be written with 0.
LTAG	[6:4]	rh	**LMB Bus Tag ID** In the TC1797, the bit field LTAG = 000_B.
FTAG	[11:8]	rh	**FPI Bus (SPB) Tag ID** In the TC1797, the bit field FTAG = 1011_B, which reflects the tag number of the LFI Bridge on the SPB.
0	[3:1], 7, [31:12]	r	**Reserved** Returns 0 if read; must be written with 0.

4.5 System Peripheral Bus

The TC1797 has one on-chip FPI Bus:

- System Peripheral Bus (SPB)
  - System bus for on-chip peripherals

This section gives an overview of the on-chip FPI Bus. It describes its bus control units, the bus characteristics, bus arbitration, scheduling, prioritizing, error conditions, and debugging support.

4.5.1 Overview

The FPI Bus interconnects the on-chip peripheral functional units with the TC1797 processor subsystem.

The FPI Bus is designed to be quick to be acquired by on-chip functional units, and quick to transfer data. The low setup overhead of the FPI Bus access protocol guarantees fast FPI Bus acquisition, which is required for time-critical applications.

The FPI Bus is designed to sustain high transfer rates. For example, a peak transfer rate of up to 320 Mbyte/s can be achieved with the 32-bit data bus at 80 MHz bus clock. Multiple data transfers per bus arbitration cycle allow the FPI Bus to operate close to its peak bandwidth.

Additional features of the FPI Bus include:

- Optimized for high speed and high performance
- Support of multiple bus masters and pipelined transactions
- 32-bit wide address and data buses
- 8-, 16-, and 32-bit data transfers
- 64-, 128-, and 256-bit block transfers
- Central simple per-cycle arbitration
- Slave-controlled wait state insertion
- Support of atomic operations LDMST, ST.T and SWAP.W

The functional units of the TC1797 are connected to the FPI Bus via FPI Bus interfaces. An FPI Bus interfaces acts as bus agents, requesting bus transactions on behalf of their functional unit, or responding to bus transaction requests.

There are two types of bus agents:

- FPI Bus master agents can initiate FPI Bus transactions and can also act as slaves.
- Slave agents can only react and respond to FPI Bus transaction requests in order to read or write internal registers of slave modules as for example memories.

When an FPI Bus master attempts to initiate a transfer on the FPI Bus, it first signals a request for bus ownership to the bus control unit (SBCU). When bus ownership is granted by the SBCU, an FPI Bus read or write transaction is initiated. The unit targeted by the transaction becomes the FPI Bus slave, and responds with the requested action.

Some functional units operate only as slaves, while others can operate as either masters or slaves on the FPI Bus. In the TC1797, DMI and PMI (via the LFI Bridge), PCP and DMA (including Cerberus and MLI´s) operate as FPI Bus masters. On-chip peripheral units are typically FPI Bus slaves.

FPI Bus arbitration is performed by the Bus Control Unit (SBCU) of the FPI Bus. In case of bus errors, the SBCU generates an interrupt request to the CPU and provides debugging information about the actual bus error to the CPU.

4.5.2 Bus Transaction Types

This section describes the SPB transaction types.

Single Transfers

Single transfers are byte, half-word, and word transactions that target any slave connected to SPB. Note that the LFI Bridge operates as an SPB master.

Block Transfers

Block transfers operate in principle in the same way as single transfers do, but one address phase is followed by multiple data phases. Block transfers can be composed of 2 word, 4 word, or 8 word transfers.

Note: In general, block transfers (2 word, 4 word, or 8 word) cannot be executed in the TC1797 with peripheral units that operate as FPI Bus slaves during an FPI Bus transaction.

Block transfers are initiated by the following CPU instructions: LD.D, LD.DA, MOV.D, ST.D and ST.DA. The BCOPY instruction of the PCP also initiates a block transfer transaction on the FPI Bus.

Atomic Transfers

Atomic transfers are generated by LDMST, ST.T and SWAP.W instructions that require two single transfers. The read and write transfer of an atomic transfer are always locked and cannot be interrupted by another bus masters. Atomic transfers are also referenced as read-modify-write transfers.

*Note: See also **Table 4-11** for available FPI Bus transfer types.*

4.5.3 Reaction of a Busy Slave

If an FPI Bus slave is busy at an incoming FPI Bus transaction request, it can delay the execution of the FPI Bus transaction. The requesting FPI Bus master releases the FPI Bus for one cycle after the FPI Bus transaction request, in order to allow the FPI Bus slave to indicate if it is ready to handle the requested FPI Bus transaction. This sequence is repeated as long as the slave indicates that it is busy.

Note: For the FPI Bus default master, the one cycle gap does not result in a performance loss because it is granted the FPI Bus in this cycle as default master if no other master requests the FPI Bus for some other reasons.

4.5.4 Address Alignment Rules

FPI Bus address generation is compliant with the following rules:

- Half-word transactions must have a half-word aligned address (A0 = 0). Half-word accesses on byte lanes 1 and 2 addresses are illegal.
- Word transactions must always have word-aligned addresses (A[1:0] = 00_B).
- Block transactions must always have block-type aligned addresses.

4.5.5 FPI Bus Basic Operations

This section describes some basic transactions on the FPI Bus.

The example in **Figure 4-6** shows the three cycles of an FPI Bus operation:

1. **Request/Grant Cycle:** The FPI Bus master attempts to perform a read or write transfer and requests for the FPI Bus. If the FPI Bus is available, it is granted in the same cycle by the FPI Bus controller.
2. **Address Cycle:** After the request/grant cycle, the master puts the address on the FPI Bus, and all FPI Bus slave devices check whether they are addressed for the following data cycle.
3. **Data Cycle:** In the data cycle, either the master puts write data on the FPI Bus which is read by the FPI Bus slave (write cycle) or vice versa (read cycle).

Transfers 2 and 3 show the conflict when two master try to use the FPI Bus and how the conflict is resolved. In the example, the FPI Bus master of transfer 2 has a higher priority than the FPI Bus master of transfer 3.

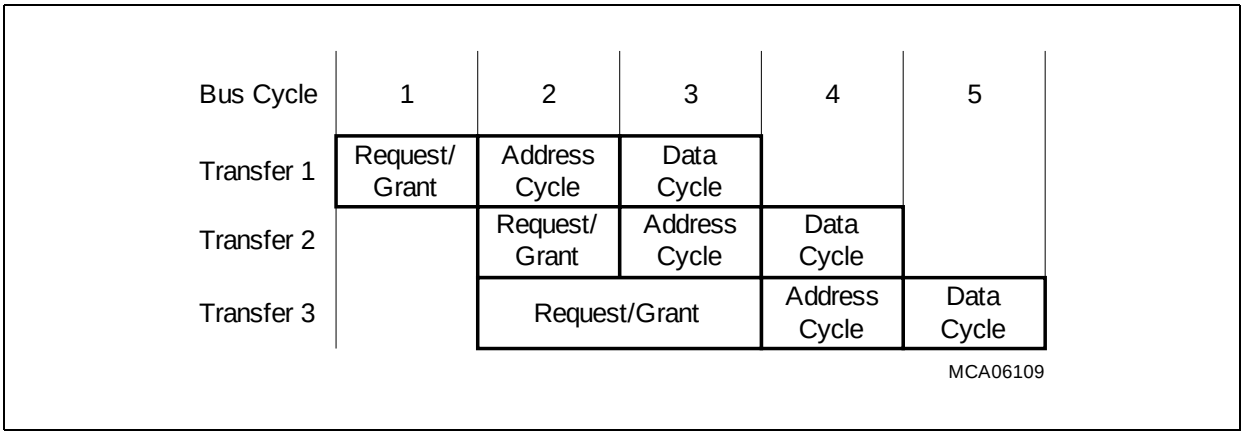


Figure 4-6 Basic FPI Bus Transactions

At a block transfer, the address cycle of a second transfer is extended until the data cycles of the block transfer are finished. In the example of **Figure 4-7**, transfer 1 is a block transfer, while transfer 2 is a single transfer.

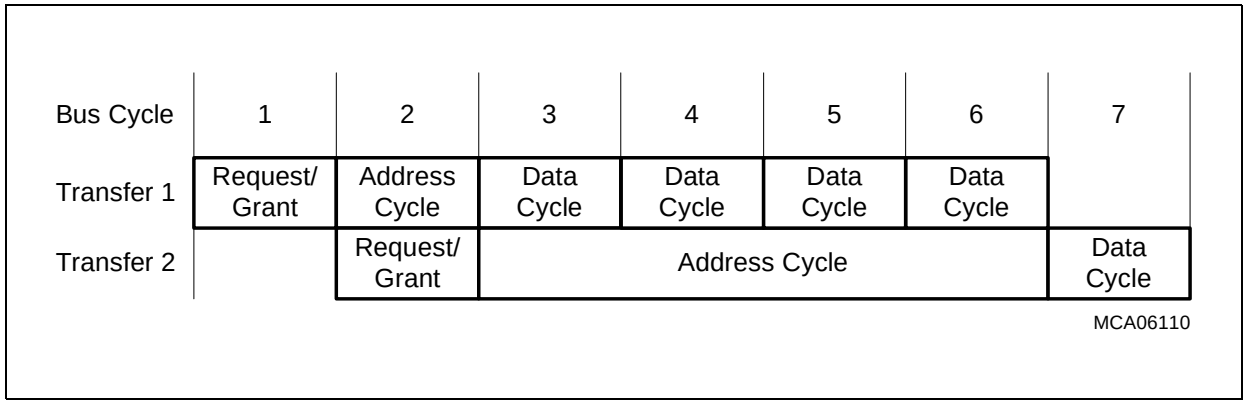


Figure 4-7 FPI Bus Block Transactions

4.6 FPI Bus Control Unit (SBCU)

The TC1797 incorporates one BCU for the SPB, called SBCU.

4.6.1 FPI Bus Arbitration

The arbitration unit of the BCU determines whether it is necessary to arbitrate for FPI Bus ownership, and, if so, which available bus requestor gets the FPI Bus for the next data transfer. During arbitration, the bus is granted to the requesting agent with the highest priority. If no request is pending, the bus is granted to a default master. If no bus master takes the bus, the BCU itself will drive the FPI Bus to prevent it from floating electrically.

4.6.1.1 Arbitration on the System Peripheral Bus

The TC1797 SPB has three bus agents that can become a SPB bus master (DMA, LFI, PCP). Each agent is supplied an arbitration priority as shown in **Table 4-9**. DMA controller agent can be assigned to low, medium or high priority by software (via DMA Channel and OCDS control registers).

Table 4-9 Priority of TC1797 SPB Bus Agents

Priority	Agent	Comment
highest	Any bus requestor meeting the starvation protection criteria is assigned this priority	Highest priority, used only for starvation protection
	DMA, high priority	DMA requests from module: - OCDS high priority[1)] - DMA channel with high priority[2)]
	Peripheral Control Processor (PCP)	Default master 0
	DMA, medium priority	DMA requests from module: - DMA channels with med. priority
	LFI Bridge	Default master 1
lowest	DMA, low priority	DMA requests from modules: - Cerberus low priority[1)] - DMA channels low priority[2)] - MLI

1) Priority of Cerberus transaction at the On Chip Busses is defined by the register bit IOCONF.FPI_PRIO. The register is defined in the OCDS chapter.

2) Priority of a DMA channel is defined by the corresponding CHCRmn.DMAPRIO bits. The registers are defined in the DMA chapter.

If there is no request from an SPB bus master, the SPB is granted to a default master (LFI Bridge or PCP) which has been at last the active master.

4.6.1.2 Starvation Prevention

Starvation prevention is a feature of the SBCU that can take care that even requesting low priority master agents will be granted after a period, where the period length can be controlled by SBCU control registers. Because the priority assignment of the SPB agents is fixed, it is possible that a lower-priority bus requestor may never be granted the bus if a higher-priority bus requestor continuously asks for, and receives, bus ownership. To protect against bus starvation of lower-priority masters, the starvation prevention mechanism of the SBCU will detect such cases and momentarily raise the priority of the lower-priority requestor to the highest priority (above all other priorities), thereby guaranteeing it access.

Starvation protection employs a counter that is decremented each time an arbitration is performed on the connected FPI bus. The counter is re-loaded with the starvation period value in the SBCU_CON.SPC bit field as long it is enabled SBCU_CON.SPE. When this counter is counted down to zero, for each active bus request a request flag is stored in the BCU. This flag is cleared automatically when a master is granted the bus.

When the next period is finished, an active request of a master from which the request flag was set, a starvation event happened. This master will now be set to the highest priority and will be granted service. If there are several masters to which this starvation condition applies, they are served in the order of their hard-wired priority ranking.

If a master that is processing its transaction under starvation condition is retried, its corresponding request flag is automatically again.

Starvation protection can be enabled and disabled through bit SBCU_CON.SPE. The sample period of the counter is programmed through bit field SBCU_CON.SPC. SPC should be set to a value at least greater than or equal to the number of masters. Its reset value is 40_H.

4.6.2 FPI Bus Error Handling

When an error occurs on an FPI Bus, its BCU captures and stores data about the erroneous condition and generates a service request if enabled to do so. The error conditions that force an error-capture are:

- Error Acknowledge: An FPI Bus slave responds with an error to a transaction.
- Un-implemented Address: No FPI Bus slave responds to a transaction request.
- Time-out: A slave does not respond to a transaction request within a certain time window. The number of bus clock cycles that can elapse until a bus time-out is generated is defined by bit field SBCU_CON.TOUT.

When a transaction causes an error, the address and data phase signals of the transaction causing the error are captured and stored in registers.

- The Error Address Capture Register (SBCU_EADD) stores the 32-bit FPI Bus address that has been captured during the erroneous FPI Bus transaction.

- The Error Data Capture Registers (SBCU_EDAT) stores the 32-bit FPI Bus data bus information that has been captured during the erroneous FPI Bus transaction.
- The Error Control Capture Register (SBCU_ECON) stores status information of the bus error event.

If more than one FPI Bus transaction generates a bus error, only the first bus error is captured. After a bus error has been captured, the capture mechanism must be released again by software.

If a write transaction from TriCore causes an error on the SPB, the originating master is not informed about this error as it is not an SPB master agent. With each bus error-capture event, a service request is generated, and an interrupt can be generated if enabled and configured in the corresponding service request register.

Interpreting the BCU Control Register Error Information

Although the address and data values captured in registers SBCU_EADD and SBCU_EDAT, respectively, are self-explanatory, the captured FPI Bus control information needs some more explanation.

Register SBCU_ECON captures the state of the read (RDN), write (WRN), Supervisor Mode (SVM), acknowledge (ACK), ready (RDY), abort (ABT), time-out (TOUT), bus master identification lines (TAG) and transaction operation code (OPC) lines of the FPI Bus.

The SVM signal is set to 1 for an access in Supervisor Mode and set to 0 for an access in User Mode. The time-out signal indicates if there was no response on the bus to an access, and the programmed time (via SBCU_TOUT) has elapsed. TOUT is set to 1 in this case. An acknowledge code has to be driven by the selected slave during each data cycle of an access. These codes are listed in **Table 4-10**.

Table 4-10 FPI Bus Acknowledge Codes

Code (ACK)	Description
00_B	NSC: No Special Condition.
01_B	SPT: Split Transaction (not used in the TC1797).
10_B	RTY: Retry. Slave can currently not respond to the access. Master needs to repeat the access later.
11_B	ERR: Bus Error, last data cycle is aborted.

Transactions on the FPI Bus are classified via a 4-bit operation code (see **Table 4-11**). Note that split transactions (OPC = 1000_B to 1110_B) are not used in the TC1797.

Table 4-11 FPI Bus Operation Codes (OPC)

OPC	Description
0000_B	Single Byte Transfer (8-bit)
0001_B	Single Half-Word Transfer (16-bit)
0010_B	Single Word Transfer (32-bit)
0100_B	2-Word Block Transfer
0101_B	4-Word Block Transfer
0110_B	8-Word Block Transfer
1111	No operation
0011_B, 0111_B, 1000_B - 1110_B	Reserved

4.6.3 BCU Debug Support

For debugging purposes, the BCU has the capability for breakpoint generation support. This OCDS debug capability is controlled by the Cerberus module and must be enabled by it (indicated by bit SBCU_DBCNTL.EO).

When BCU debug support has been enabled (EO = 1), any breakpoint request generated by the BCU to the Cerberus disarms the BCU breakpoint logic for further breakpoint requests. In order to rearm the BCU breakpoint logic again for the next breakpoint request generation, bit SBCU_DBCNTL.RA must be set. The status of the BCU breakpoint logic (armed or disarmed) is indicated by bit SBCU_DBCNTL.OA.

There are three types of trigger events:

- Address triggers
- Signal triggers
- Grant triggers

4.6.3.1 Address Triggers

The address debug trigger event conditions are defined by the contents of the SBCU_DBADR1, SBCU_DBADR2, and SBCU_DBCNTL registers. A wide range of possibilities arise for the creation of debug trigger events based on addresses. The following debug trigger events can be selected:

- Match on one signal address
- Match on one of two signal addresses
- Match on one address area
- Mismatch on one address area

Each pair of DBADRx registers and DBCNTL.ONAx bits determine one possible debug trigger event. The combination of these two possible debug trigger events defined by DBCNTL.CONCOM1 determine the address debug trigger event condition.

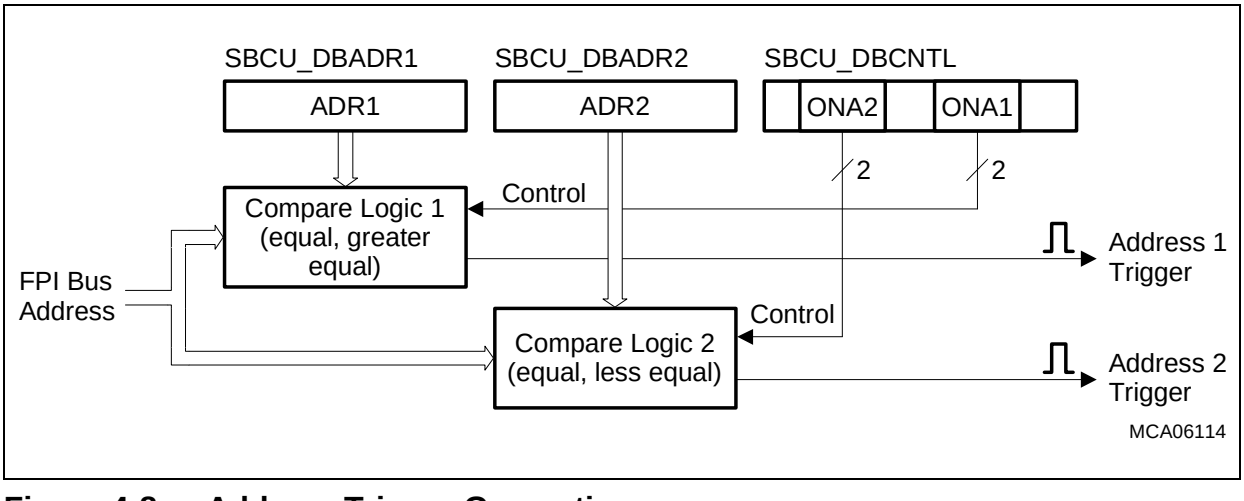


Figure 4-8 Address Trigger Generation

4.6.3.2 Signal Status Triggers

The signal status debug trigger event conditions are defined by the contents of the SBCU_DBBOS and SBCU_DBCNTL registers. Depending on the selected configuration a wide range of possibilities arise for the creation of a debug trigger event based on FPI Bus status signals. Possible combinations are:

- Match on a single signal status
- Match on a multiple signal status

With the multiple signal match conditions, all single signal match conditions are combined with a logical **AND** to the signal status debug trigger event signal. The selection whether or not a single match condition is selected can be enabled/disabled selectively for each condition via the SBCU_DBCNTL.ONBOSx bits.

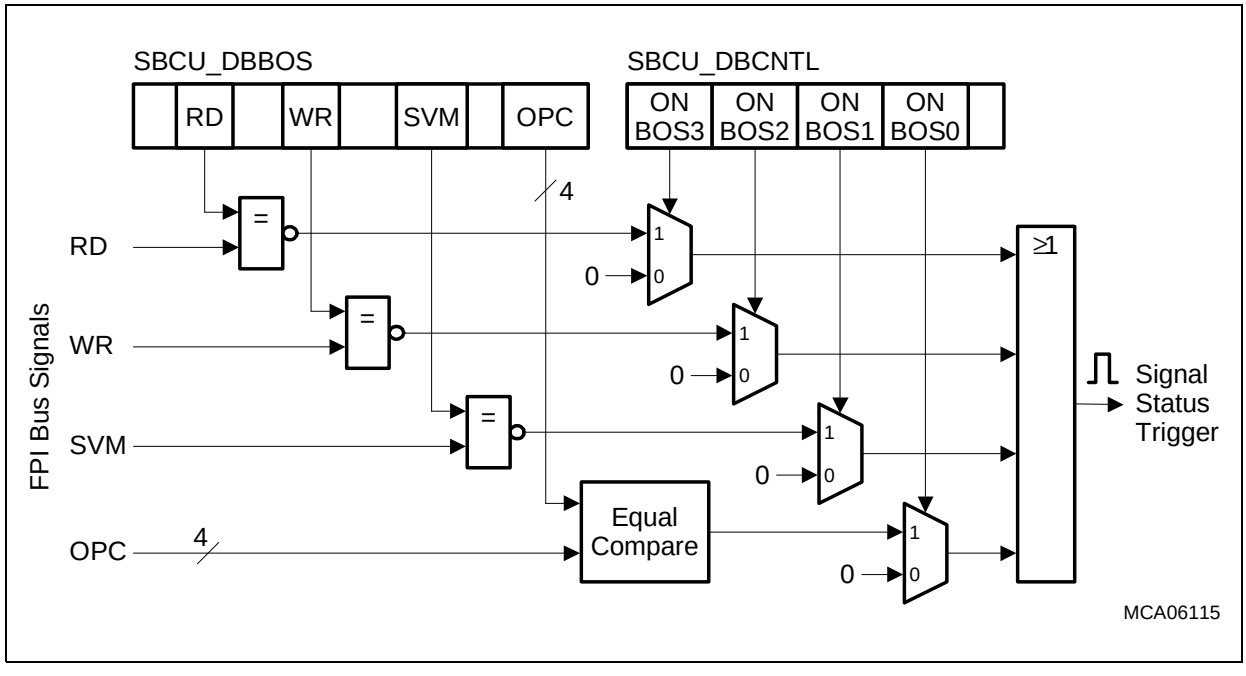


Figure 4-9 Signal Status Trigger Generation

4.6.3.3 Grant Triggers

The signal status debug trigger event conditions are defined via the registers SBCU_DBGRNT and SBCU_DBCNTL. Depending on the configuration of these registers, any combination of FPI Bus master trigger events can be configured. Only the enabled masters in the SBCU_DBGRNT register are of interest for the grant debug trigger event condition. The grant debug trigger event condition can be enabled/disabled via bit SBCU_DBCNTL.ONG (see **Figure 4-10**).

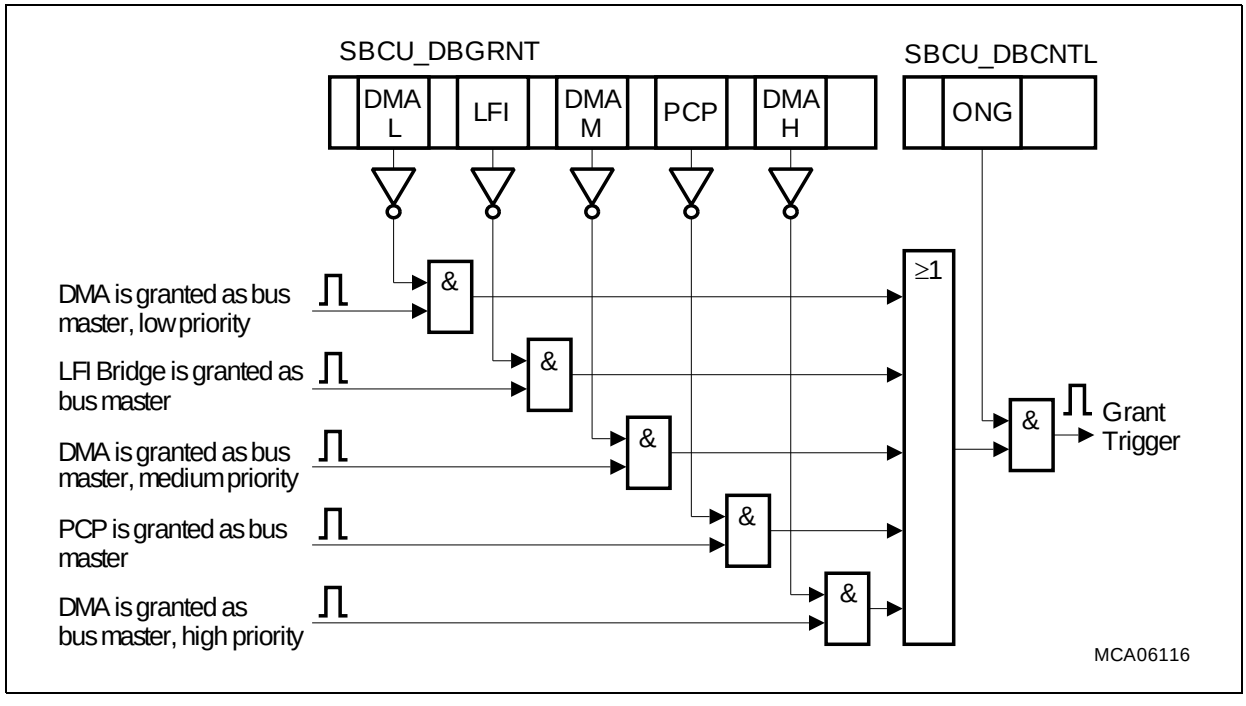


Figure 4-10 Grant Trigger Generation

4.6.3.4 Combination of Triggers

The combination of the four debug trigger signals to the single BCU breakpoint trigger event is defined via the bits CONCOM[2:0] of register SBCU_DBCNTL (see **Figure 4-11**). The two address triggers are combined to one address trigger that is further combined with signal status and grant trigger signals. A logical AND or OR combination can be selected for the BCU breakpoint trigger generation.

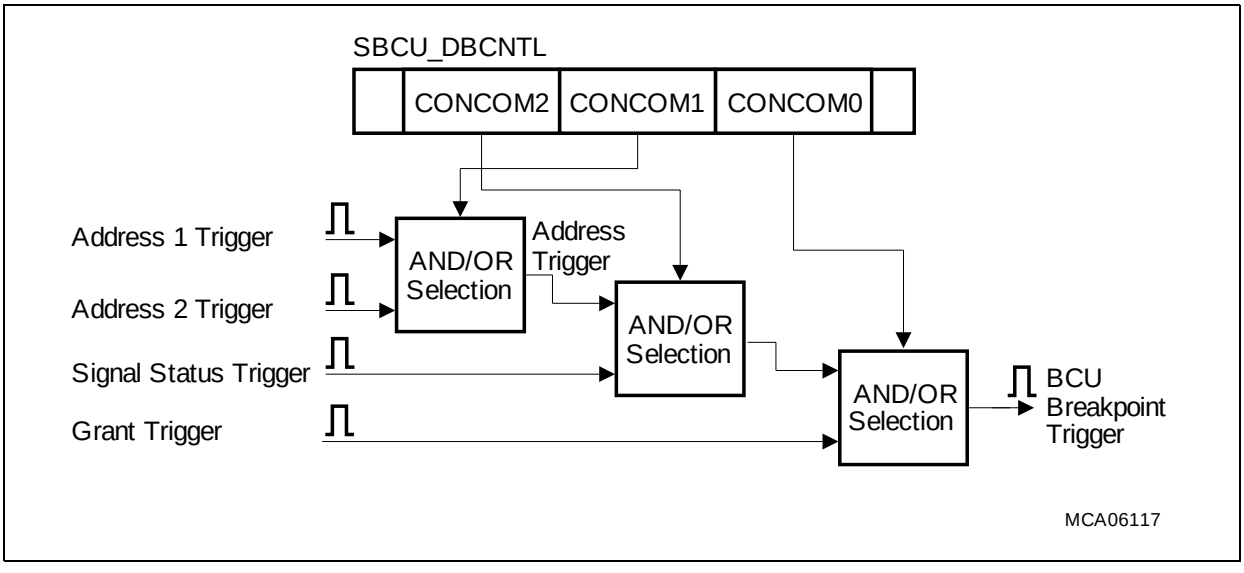


Figure 4-11 BCU Breakpoint Trigger Combination Logic

4.6.3.5 BCU Breakpoint Generation Examples

This section gives three examples of how BCU debug trigger events are programmed.

OCDS Debug Example 1

- Task: Generation of a BCU debug trigger event on any SPB write access to address 00002004_H or $000020A0_H$ by SPB master of the LFI Bridge or the PCP.

For this task, the following programming settings for the BCU breakpoint logic must be executed:

1. Writing SBCU_DBADR1 = $0000\ 2004_H$
2. Writing SBCU_DBADR2 = $0000\ 20A0_H$
3. Writing SBCU_DBCNTL = $C1115010_H$:
   a) ONBOS[3:0] = 1100_B means that no signal status trigger is generated (disabled) for a read signal match AND write signal match condition according to the settings of bits RD and WR in register SBCU_DBBOS. Debug trigger event generation for Supervisor Mode signal match and opcode signal match condition is disabled.
   b) ONA2 = 01_B means that the equal match condition for debug address 2 register is selected.

c) ONA1 = 01_B means that the equal match condition for debug address 1 register is selected.
d) ONG = 1 means that the grant debug trigger is enabled.
e) CONCOM[2:0] = 101_B means that the address trigger is created by address trigger 1 OR address trigger 2 (CONCOM1 = 0), and that the grant trigger is ANDed with the address trigger (CONCOM0 = 1), and that the signal status trigger is ANDed with the address trigger (CONCOM2 = 1).
f) RA = 1 means that the BCU breakpoint logic is rearmed.

4. Writing SBCU_DBGRNT = $FFFFFFD7_H$:
means that the grant trigger for the SPB master of the PCP and LFI Bridge is enabled.
5. Writing SBCU_DBBOS = 00001000_H:
means that the signal status trigger is generated on a write transfer and not on a read transfer.

OCDS Debug Example 2

- Task: generation of a BCU debug trigger event on any half-word access in User Mode to address area $01FFFFFF_H$ to $02FFFFFF_H$ by any master.

For this task, the following programming settings for the BCU breakpoint logic must be executed:

1. Writing SBCU_DBADR1 = $01FFFFFF_H$
2. Writing SBCU_DBADR2 = $02FFFFFF_H$
3. Writing SBCU_DBCNTL = 32206010_H:
a) ONBOS[3:0] = 0011_B means that the signal status trigger is disabled for a read or for write signal status match but enabled for Supervisor Mode match AND opcode match conditions according to the settings of bit SVM and bit field OPC in register SBCU_DBBOS.
b) ONA2 = 10_B means that the address 2 trigger is generated if the FPI Bus address is greater or equal to SBCU_DBADR2.
c) ONA1 = 10_B means that the address 1 trigger is generated if the FPI Bus address is greater or equal to SBCU_DBADR1.
d) ONG = 0 means that the grant debug trigger is disabled.
e) CONCOM[2:0] = 110_B means that the address trigger is created by address trigger 1 AND address trigger 2 (CONCOM1 = 1), and that the grant trigger is OR-ed with the address trigger (CONCOM0 = 0), and that the signal status trigger is AND-ed with the address trigger (CONCOM2 = 1).
f) RA = 1 means that the BCU breakpoint logic is rearmed.
4. Writing SBCU_DBGRNT = $FFFFFFFF_H$:
means that no grant trigger for SPB masters is selected (“don’t care” because also disabled by ONG = 0).
5. Writing SBCU_DBBOS = 00000001_H:
means that the signal status trigger is generated for read (RD = 0) and write (WR = 0) half-word transfers (OPC = 0001_B) in User Mode (SVM = 0).

OCDS Debug Example 3

- Task: Generation of a BCU debug trigger event on any access into address area $01FFFFFF_H$ to $FFFFFFFF_H$ by the PCP.

For this task the following programming settings for the BCU breakpoint logic must be executed:

1. Writing SBCU_DBADR1 = $01FFFFFF_H$
2. Writing SBCU_DBADR2 = don't care
3. Writing SBCU_DBCNTL = 00215010_H:
   a) ONBOS[3:0] = 0000_B means that a signal status trigger is generated for all FPI Bus opcodes not equal to a "no operation" opcode.
   b) ONA2 = 00_B means that no address 2 trigger is generated.
   c) ONA1 = 10_B means that the address 1 trigger is generated if the FPI Bus address is greater or equal to SBCU_DBADR1.
   d) ONG = 1 means that the grant debug trigger is enabled.
   e) CONCOM[2:0] = 101_B means that the address trigger is created by address trigger 1 OR address trigger 2 (CONCOM1 = 0), and that the grant trigger is AND-ed with the address trigger (CONCOM0 = 1), and that the signal status trigger is ANDed with the address trigger (CONCOM2 = 1).
   f) RA = 1 means that the BCU breakpoint logic is rearmed.
4. Writing SBCU_DBGRNT = $FFFFFFF7_H$:
   means that the grant trigger for the SPB bus master of the PCP is enabled.
5. Writing SBCU_DBBOS is "don't care". No signal trigger for SVM, WR, or RD is generated.

4.6.4 System Bus Control Unit Registers

Figure 4-12 and Table 4-13 are showing the address maps with all registers of the System Bus Control Unit (SBCU) module.

SBCU Control Registers Overview

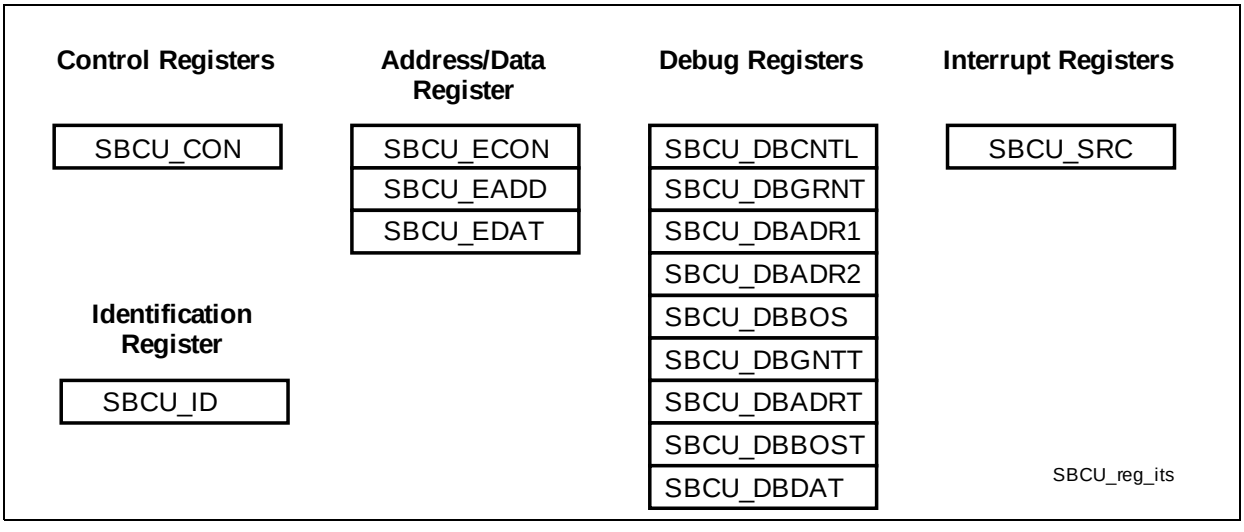


Figure 4-12 SBCU Registers

Table 4-12 Registers Address Space - SBCU Address Space

Module	Base Address	End Address	Note
SBCU	F000 0100$_H$	F000 01FF$_H$	

Table 4-13 Registers Overview - SBCU Control Registers

Short Name	Description	Offset Addr.[1)]	Access Mode		Reset Class	Description See
			Read	Write		
–	Reserved	000$_H$ - 004$_H$	BE	BE	-	-
SBCU_ID	SBCU Module Identification Register	008$_H$	U, SV	BE	-	Page 4-39
–	Reserved	00C$_H$	BE	BE	-	-
SBCU_ CON	SBCU Control Register	010$_H$	U, SV	SV	3	Page 4-40
–	Reserved	014$_H$ - 01C$_H$	BE	BE	3	-

Table 4-13 Registers Overview - SBCU Control Registers

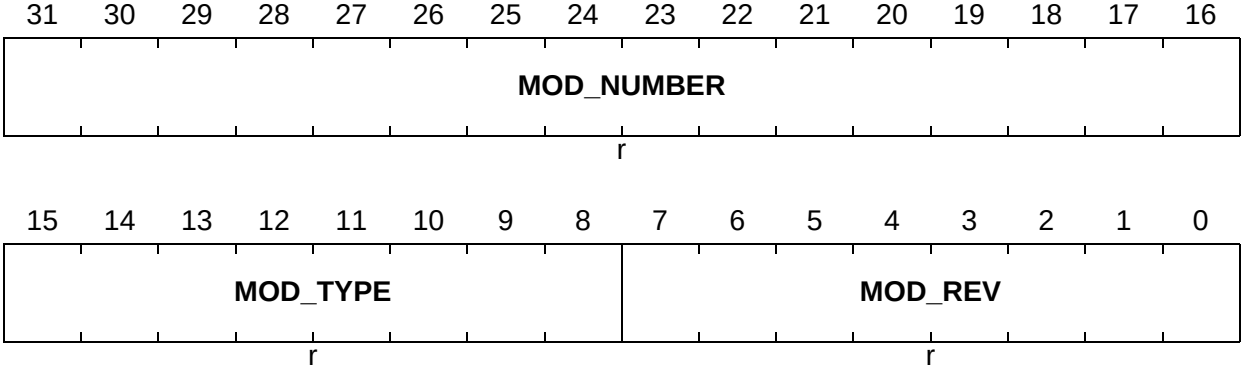
Short Name	Description	Offset Addr.[1]	Access Mode		Reset Class	Description See
			Read	Write		
SBCU_ ECON	SBCU Error Control Capture Register	020_H	U, SV	SV	3	Page 4-41
SBCU_ EADD	SBCU Error Address Capture Register	024_H	U, SV	SV	3	Page 4-43
SBCU_ EDAT	SBCU Error Data Capture Register	028_H	U, SV	SV	3	Page 4-44
–	Reserved	$02C_H$	BE	BE	-	-
SBCU_ DBCNTL	SBCU Debug Control Register	030_H	U, SV	SV	1	Page 4-45
SBCU_ DBGRNT	SBCU Debug Grant Mask Register	034_H	U, SV	SV	1	Page 4-48
SBCU_ DBADR1	SBCU Debug Address Register 1	038_H	U, SV	SV	1	Page 4-49
SBCU_ DBADR2	SBCU Debug Address Register 2	$03C_H$	U, SV	SV	1	Page 4-50
SBCU_ DBBOS	SBCU Debug Bus Operation Signals Register	040_H	U, SV	SV	1	Page 4-50
SBCU_ DBGNTT	SBCU Debug Trapped Master Register	044_H	U, SV	BE	1	Page 4-52
SBCU_ DBADRT	SBCU Debug Trapped Address Register	048_H	U, SV	BE	1	Page 4-54
SBCU_ DBBOST	SBCU Debug Trapped Bus Operation Signals Register	$04C_H$	U, SV	BE	1	Page 4-54
SBCU_ DBDAT	SBCU Debug Data Status Register	050_H	U, SV	BE	1	Page 4-57
–	Reserved	054_H - $0F8_H$	BE	BE	-	-
SBCU_ SRC	SBCU Service Request Control Register	$0FC_H$	U, SV	SV	3	Page 4-58

1) The absolute register address is calculated as follows:
Module Base Address (Table 4-12) + Offset Address (shown in this column)

4.6.4.1 SBCU ID Register Description

The identification register allows the programmer version-tracking of the module. The table below shows the identification register which is implemented in the SBCU module.

SBCU_ID
Module Identification Register (008$_H$) Reset Value: 0000 6AXX$_H$

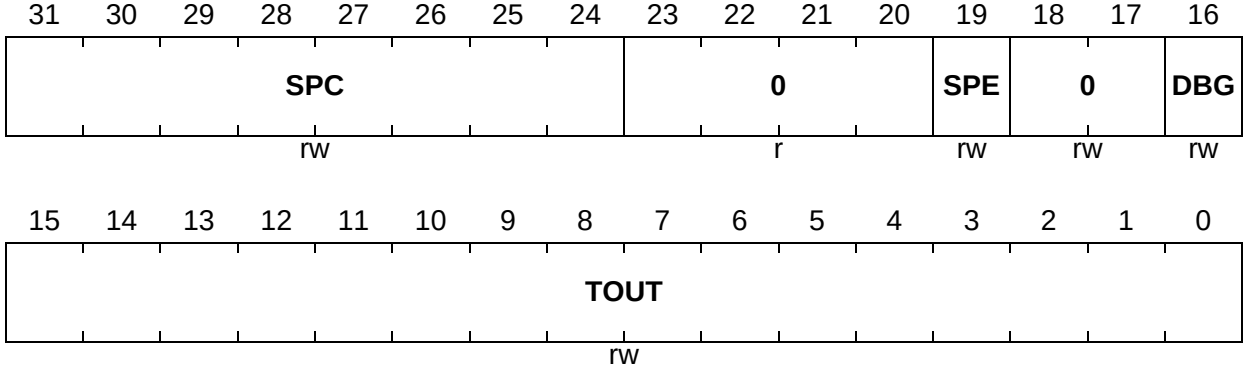


Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision Number** This bit field defines the module revision number. The value of a module revision starts with 01$_H$ (first revision).
MOD_NUMBER	[15:8]	r	**Module Number Value** This bit field defines a module identification number. The value for the LBCU module is 006AH.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

4.6.4.2 SBCU Control Registers Descriptions

The SBCU Control Register controls the overall operation of the SBCU, including setting the starvation sample period, the bus time-out period, enabling starvation-protection mode, and error handling.

SBCU_CON
SBCU Control Register **(010_H)** **Reset Value: $4009\ FFFF_H$**



Field	Bits	Type	Description
TOUT	[15:0]	rw	**SBCU Bus Time-Out Value** The bit field determines the number of System Peripheral Bus time-out cycles. Default after reset is $FFFF_H$ (= 65536 bus cycles).
DBG	16	rw	**SBCU Debug Trace Enable** 0_B SBCU debug trace disabled 1_B SBCU debug trace enabled (default after reset)
SPE	19	rw	**SBCU Starvation Protection Enable** 0_B SBCU starvation protection disabled 1_B SBCU starvation protection enabled (default after reset)
SPC	[31:24]	rw	**Starvation Period Control** Determines the sample period for the starvation counter. Must be larger than the number of masters. The reset value is 40_H.
0	[18:17], [23:20]	r	**Reserved** Read as 0; should be written with 0.

4.6.4.3 SBCU Error Registers Descriptions

The capture of bus error conditions is enabled by setting SBCU_CON.DBG to 1. In case of a bus error, information about the condition will then be stored in the SBCU error capture registers. The SBCU error capture registers can then be examined by software to determine the cause of the FPI Bus error.

If enabled and an FPI Bus error occurs, the SBCU_ECON register holds the captured FPI Bus control information and an error count of the number of bus errors. The SBCU_EADD register stores the captured FPI Bus address. The SBCU_EDAT register stores the captured FPI Bus data.

If the capture of FPI Bus error conditions is disabled (SBCU_CON.DBG = 0), the SBCU error capture registers remain untouched.

Note: The SBCU error capture registers store only the parameters of the first error. In case of multiple bus errors, an error counter SBCU_ECON.ERRCNT shows the number of bus errors since the first error occurred. An application reset clears this bit field to zero, but the counter can be set to any value through software. This counter is prevented from overflowing, so a value of 2^{16} - 1 indicates that at least this many errors have occurred, but there may have been more. After SBCU_ECON has been read, the SBCU_ECON, SBCU_EADD and SBCU_EDAT registers are re-enabled to trace FPI Bus error conditions.

SBCU_ECON
SBCU Error Control Capture Register (020_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OPC				TAG				RDN	WRN	SVM	ACK		ABT	RDY	T OUT
rwh				rwh				rwh	rwh	rwh	rwh		rwh	rwh	rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ERRCNT															
rwh															

Field	Bits	Type	Description
ERRCNT	[15:0]	rwh	**FPI Bus Error Counter** ERRCNT is incremented on every occurrence of an FPI Bus error. ERRCNT is reset to 0000_H after the SBCU_ECON register is read.[1)]
TOUT	16	rwh	**State of FPI Bus Time-Out Signal** This bit indicates the state of the time-out signal at an FBI Bus error. 0_B No time-out occurred 1_B Time-out has occurred
RDY	17	rwh	**State of FPI Bus Ready Signal** This bit indicates the state of the ready signal at an FBI Bus error. 0_B Wait state(s) have been inserted. Ready signal was active 1_B Ready signal was inactive
ABT	18	rwh	**State of FPI Bus Abort Signal** This bit indicates the state of the abort signal at an FBI Bus error. 0_B Master has aborted an FPI Bus transfer. Abort signal was active 1_B Abort signal was inactive
ACK	[20:19]	rwh	**State of FPI Bus Acknowledge Signals** This bit field indicates the acknowledge code that has been output by the selected slave at an FPI Bus error. Coding see **Table 4-10**.
SVM	21	rwh	**State of FPI Bus Supervisor Mode Signal** This bit indicates whether the FPI Bus error occurred in Supervisor Mode or in User Mode. 0_B Transfer was initiated in Supervisor Mode 1_B Transfer was initiated in User Mode
WRN	22	rwh	**State of FPI Bus Write Signal** This bit indicates whether the FPI Bus error occurred at a write cycle (see**Table 4-14**).
RDN	23	rwh	**State of FPI Bus Read Signal** This bit indicates whether the FPI Bus error occurred at a read cycle (see**Table 4-14**).

Field	Bits	Type	Description
TAG	[27:24]	rwh	**FPI Bus Master Tag Number Signals** This bit field indicates the FPI Bus master TAG number (definitions see **Table 4-15**).
OPC	[31:28]	rwh	**FPI Bus Operation Code Signals** The FPI Bus operation codes are defined in **Table 4-11**.

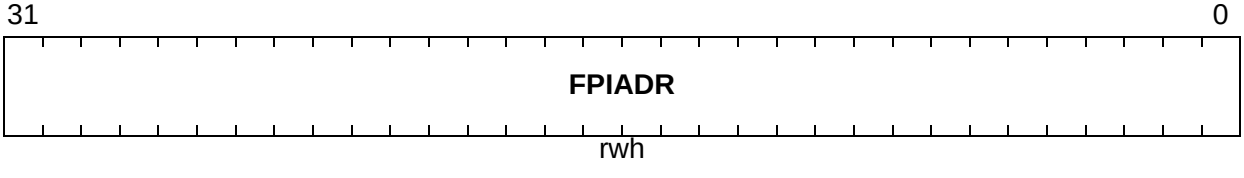
1) In the TC1797, aborted accesses to a 0 wait state SPB slave may also increment ERRCNT when the slave generates an error acknowledge.

Table 4-14 FPI Bus Read/Write Error Indication

$\overline{RD}$	$\overline{WR}$	FPI Bus Cycle
0	0	FPI Bus error occurred at the read transfer of a read-modify-write transfer.
0	1	FPI Bus error occurred at a read cycle of a single transfer.
1	0	FPI Bus error occurred at a write cycle of a single transfer or at the write cycle of a read-modify-write transfer.
1	1	Does not occur.

SBCU_EADD

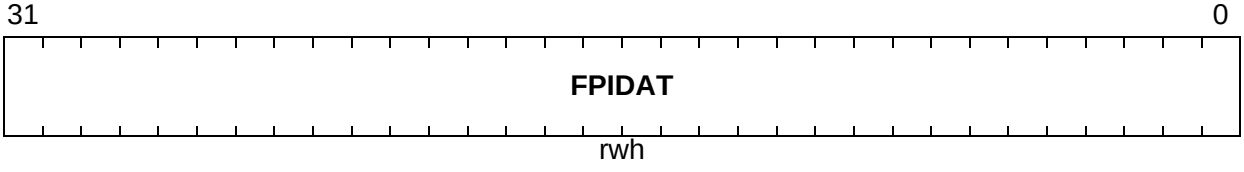
SBCU Error Address Capture Register (024_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
FPIADR	[31:0]	rwh	**Captured FPI Bus Address** This bit field holds the 32-bit FPI Bus address that has been captured at an FPI Bus error. Note that if multiple bus errors occurred, only the address of the first bus error is captured.

SBCU_EDAT

SBCU Error Data Capture Register (028$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
FPIDAT	[31:0]	rwh	**Captured FPI Bus Address** This bit field holds the 32-bit FPI Bus data that has been captured at an FPI Bus error. Note that if multiple bus errors occurred, only the data of the first bus error is captured.

4.6.4.4 SBCU OCDS Registers Descriptions

SBCU_DBCNTL
SBCU Debug Control Register (030_H) Reset Value: 0000 7003$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ONBOS3	ONBOS2	ONBOS1	ONBOS0	0		ONA2		0		ONA1		0			ONG
rw	rw	rw	rw	r		rw		r		rw		r			rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	CONCOM2	CONCOM1	CONCOM0	0							RA	0		OA	EO
r	rw	rw	rw	r							w	r		r	r

Field	Bits	Type	Description
EO	0	r	**Status of SBCU Debug Support Enable** This bit is controlled by the Cerberus and enables the SBCU debug support. 0_B SBCU debug support is disabled 1_B SBCU debug support is enabled (default after reset)
OA	1	r	**Status of SBCU Breakpoint Logic** 0_B The SBCU breakpoint logic is disarmed. Any further breakpoint activation is discarded 1_B The SBCU breakpoint logic is armed The OA bit is set by writing a 1 to bit RA. When OA is set, registers SBCU_DBGNTT, SBCU_DBADRT, and SBCU_DBBOST are reset.
RA	4	w	**Rearm SBCU Breakpoint Logic** Writing a 1 to this bit rearms SBCU breakpoint logic and sets bit OA = 1. RA is always reads as 0.

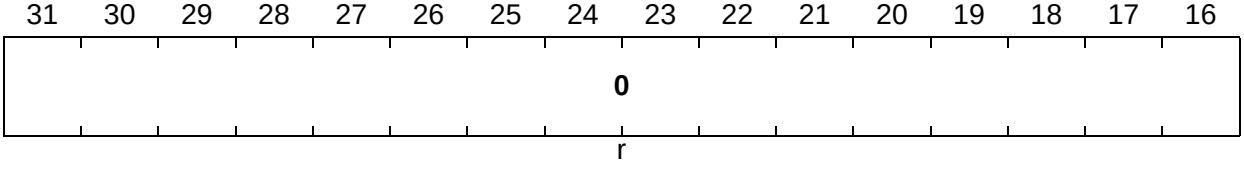
Field	Bits	Type	Description
CONCOM0	12	rw	**Grant and Address Trigger Relation** 0_B The grant phase trigger condition and the address trigger condition (see CONCOM1) are combined with a logical OR for further control 1_B The grant phase trigger condition and the address trigger condition (see CONCOM1) are combined with a logical AND for further control (see **Figure 4-11**)
CONCOM1	13	rw	**Address 1 and Address 2 Trigger Relation** 0_B Address 1 trigger condition and address 2 trigger condition are combined with a logical OR to the address trigger condition for further control 1_B Address 1 trigger condition and address 2 trigger condition are combined with a logical AND to the address trigger condition for further control (see **Figure 4-11**)
CONCOM2	14	rw	**Address and Signal Trigger Relation** 0_B Address trigger condition (see CONCOM1) and signal status trigger conditions are combined with a logical OR for further control 1_B Address phase trigger condition (see CONCOM1) and the signal status trigger conditions are combined with a logical AND for further control (see **Figure 4-11**)
ONG	16	rw	**Grant Trigger Enable** 0_B No grant debug event trigger is generated 1_B The grant debug event trigger is enabled and generated according the settings of register SBCU_DBGRNT (see **Figure 4-10**)
ONA1	[21:20]	rw	**Address 1 Trigger Control** 00_B No address 1 trigger is generated 01_B An address 1 trigger event is generated if the FPI Bus address is equal to SBCU_DBADR1 10_B An address 1 trigger event is generated if FPI Bus address is greater or equal to SBCU_DBADR1 11_B same as 00_B See also **Figure 4-8**.

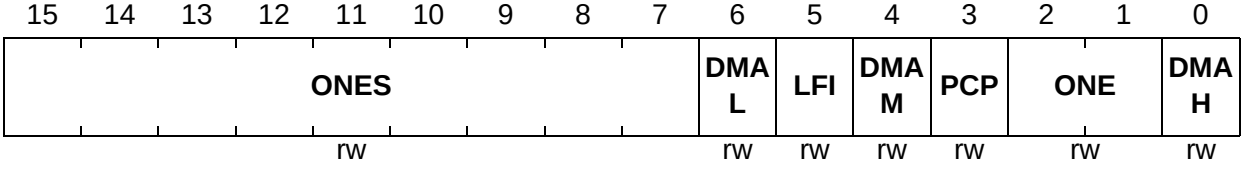
Field	Bits	Type	Description
ONA2	[25:24]	rw	**Address 2 Trigger Control** 00_B No address 2 trigger is generated 01_B An address 2 trigger event is generated if the FPI Bus address is equal to SBCU_DBADR2 10_B An address 2 trigger event is generated if FPI Bus address is greater or equal to SBCU_DBADR2 11_B same as 00_B See also **Figure 4-8**.
ONBOS0	28	rw	**Opcode Signal Status Trigger Condition** 0_B A signal status trigger is generated for all FPI Bus opcodes except a “no operation” opcode 1_B A signal status trigger is generated if the FPI Bus opcode matches the opcode as defined in DBBOS.OPC (see **Figure 4-9**)
ONBOS1	29	rw	**Supervisor Mode Signal Trigger Condition** 0_B The signal status trigger generation for the FPI Bus Supervisor Mode signal is disabled 1_B A signal status trigger is generated if the FPI Bus Supervisor Mode signal state is equal to the value of DBBOS.SVM (see **Figure 4-9**)
ONBOS2	30	rw	**Write Signal Trigger Condition** 0_B The signal status trigger generation for the FPI Bus write signal is disabled 1_B A signal status trigger is generated if the FPI Bus write signal state is equal to the value of DBBOS.WR (see **Figure 4-9**)
ONBOS3	31	rw	**Read Signal Trigger Condition** 0_B The signal status trigger generation for the FPI Bus read signal is disabled 1_B A signal status trigger is generated if the FPI Bus read signal state is equal to the value of DBBOS.RD (see **Figure 4-9**)

Field	Bits	Type	Description
0	[3:2], [11:5], 15, [19:17], [23:22], [27:26]	r	**Reserved** Read as 0; should be written with 0.

SBCU_DBGRNT

SBCU Debug Grant Mask Register (034$_H$) Reset Value: 0000 FFFF$_H$





Field	Bits	Type	Description
DMAH	0	rw	**Cerberus Grant Trigger Enable, High Priority[1)]** 0_B FPI Bus transactions with high-priority DMA as bus master are enabled for grant trigger event generation 1_B FPI Bus transactions with high-priority DMA as bus master are disabled for grant trigger event generation
ONE, ONES	[2:1], [15:7]	rw	**Reserved** Read as 1 after reset; reading these bits will return the value last written.
PCP	3	rw	**PCP Grant Trigger Enable** 0_B FPI Bus transactions with PCP as bus master are enabled for grant trigger event generation 1_B FPI Bus transactions with PCP as bus master are disabled for grant trigger event generation

Field	Bits	Type	Description
DMAM	4	rw	**DMA Grant Trigger Enable, Medium Priority**[2] 0_B FPI Bus transactions with medium-priority DMA channels as bus master are enabled for grant trigger event generation 1_B FPI Bus transactions with medium-priority DMA channels as bus master are disabled for grant trigger event generation
LFI	5	rw	**LFI Bridge Grant Trigger Enable** 0_B FPI Bus transactions with LFI Bridge as bus master are enabled for grant trigger event generation 1_B FPI Bus transactions with LFI Bridge as bus master are disabled for grant trigger event generation
DMAL	6	rw	**DMA Grant Trigger Enable, Low Priority**[3] 0_B FPI Bus transactions with low-priority DMA channels as bus master are enabled for grant trigger event generation 1_B FPI Bus transactions with low-priority DMA channels as bus master are disabled for grant trigger event generation
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

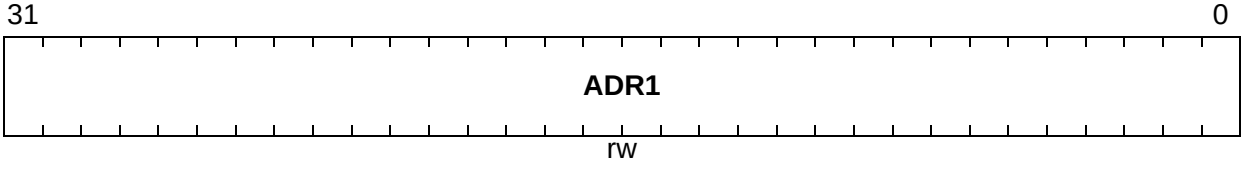
1) Including DMA transactions from DMA channels with high priority and from Cerberus with high priority.

2) Including DMA transactions from DMA channels with medium priority.

3) Including DMA transactions from DMA channels with low priority, MLI and from Cerberus with low priority.

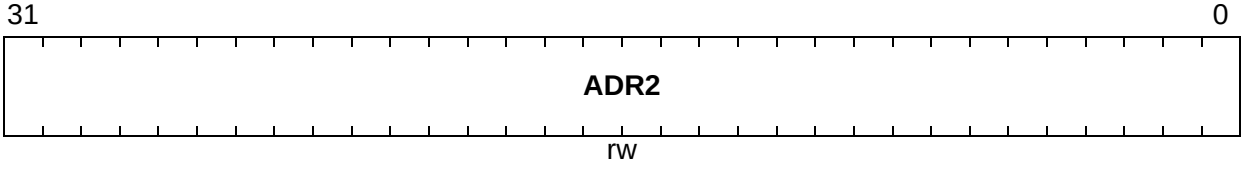
SBCU_DBADR1

SBCU Debug Address 1 Register (038_H) Reset Value: 0000 0000_H



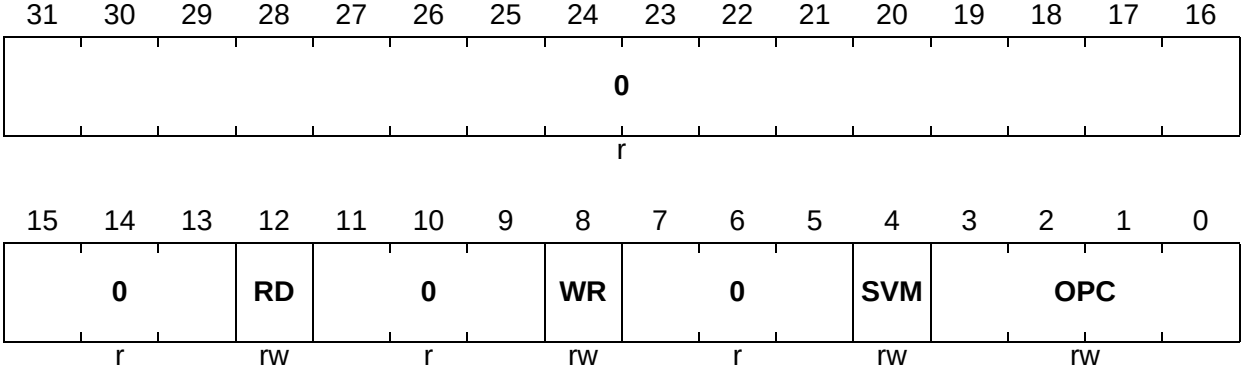
Field	Bits	Type	Description
ADR1	[31:0]	rw	**Debug Trigger Address 1** This register contains the address for the address 1 trigger event generation.

SBCU_DBADR2
SBCU Debug Address 2 Register (03C$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
ADR2	[31:0]	rw	**Debug Trigger Address 2** This register contains the address for the address 2 trigger event generation.

SBCU_DBBOS
SBCU Debug Bus Operation Signals Register
(040$_H$) Reset Value: 0000 0000$_H$

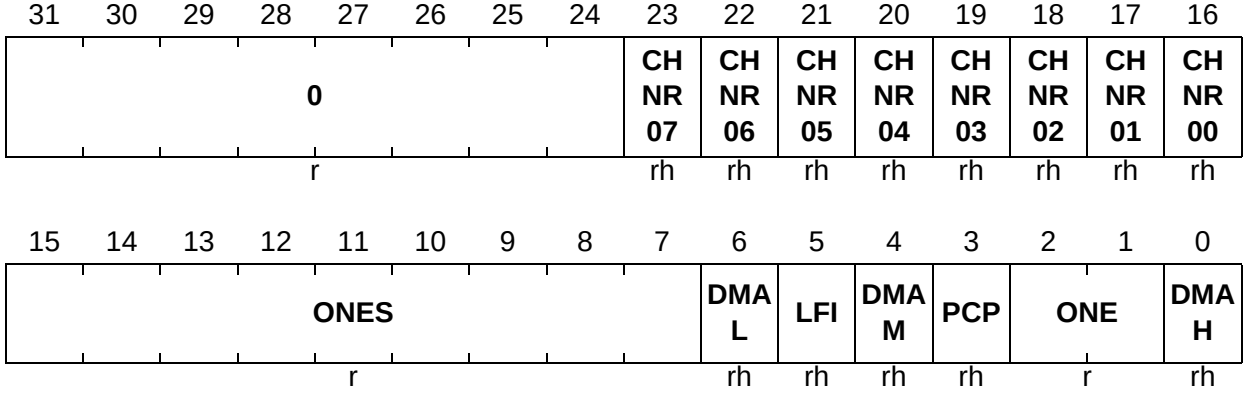


Field	Bits	Type	Description
OPC	[3:0]	rw	**Opcode for Signal Status Debug Trigger** This bit field determines the type (opcode) of an FPI Bus transaction for which a signal status debug trigger event is generated (if enabled by DBCNTL.ONBOS0 = 1). 0000_B Trigger on single byte transfer selected 0001_B Trigger on single half-word transfer selected 0010_B Trigger on single word transfer selected 0100_B Trigger on 2-word block transfer selected 0101_B Trigger on 4-word block transfer selected 0110_B Trigger on 8-word block transfer selected 1111_B Trigger on no operation selected Other bit combinations are reserved.
SVM	4	rw	**SVM Signal for Status Debug Trigger** This bit determines the mode of an FPI Bus transaction for which a signal status debug trigger event is generated (if enabled by DBCNTL.ONBOS1 = 1). 0_B Trigger on User Mode selected 1_B Trigger on Supervisor Mode selected
WR	8	rw	**Write Signal for Status Debug Trigger** This bit determines the state of the $\overline{WR}$ signal of an FPI Bus transaction for which a signal status debug trigger event is generated (if enabled by DBCNTL.ONBOS2 = 1). 0_B Trigger on a single write transfer or write cycle of an atomic transfer selected 1_B No operation or read transaction selected
RD	12	rw	**Write Signal for Status Debug Trigger** This bit determines the state of the $\overline{RD}$ signal of an FPI Bus transaction for which a signal status debug trigger event is generated (if enabled by DBCNTL.ONBOS3 = 1). 0_B Trigger on a single read transfer or read cycle of an atomic transfer selected 1_B No operation or write transfer selected

Field	Bits	Type	Description
0	[7:5], [11:9], [31:13]	r	**Reserved** Read as 0; should be written with 0.

SBCU_DBGNTT
SBCU Debug Trapped Master Register

(044_H) **Reset Value: 0000 FFFF$_H$**



Field	Bits	Type	Description
DMAH	0	rh	**High-Priority DMA FPI Bus Master Status**[1)] This bit indicates whether the DMA with a high priority request was FPI Bus master when the break trigger event occurred. 0_B The high-priority DMA was not the FPI bus master. 1_B The high-priority DMA was the FPI Bus master.
PCP	3	rh	**PCP FPI Bus Master Status** This bit indicates whether the PCP was FPI Bus master when the break trigger event occurred. 0_B The PCP was not an FPI bus master. 1_B The PCP was FPI bus master at the break trigger event.

Field	Bits	Type	Description
DMAM	4	rh	**High-Priority DMA FPI Bus Master Status**[2)] This bit indicates whether the DMA with a medium priority request was FPI Bus master when the break trigger event occurred. 0_B The medium-priority DMA was not the FPI bus master. 1_B The medium-priority DMA was the FPI Bus master.
LFI	5	rh	**LFI Bridge FPI Bus Master Status** This bit indicates whether the LFI Bridge was FPI Bus master when the break trigger event occurred. 0_B The LFI Bridge was not an FPI Bus master. 1_B The LFI Bridge was FPI Bus master.
DMAL	6	rh	**Low-Priority DMA FPI Bus Master Status**[3)] This bit indicates whether the DMA with a low-priority request was the FPI Bus master when the break trigger event occurred. 0_B The low-priority DMA was not the FPI Bus master. 1_B The low-priority DMA was the FPI Bus master.
CHNR0y (y = 0-7)	16+y	rh	**DMA Channel Number Status** These bits indicate which DMA channel with number 0y was active when a DMA break trigger event occurred. 0_B DMA channel 0y was not active at a DMA break trigger event. 1_B DMA channel 0y was active at a DMA break trigger event.
ONE, ONES	[15:7], [2:1]	r	**Reserved** Read as 1; should be written with 0.
0	[31:24]	r	**Reserved** Read as 0; should be written with 0.

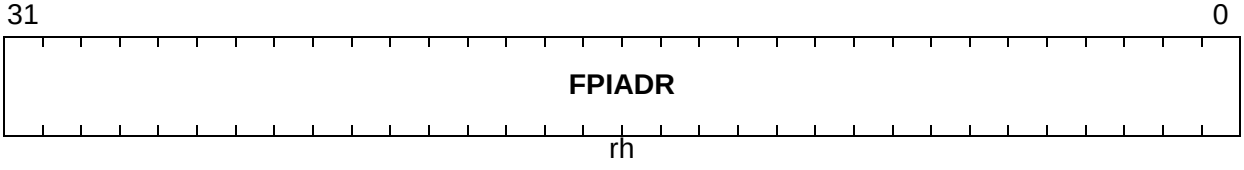
1) Including DMA transactions from DMA channels with high priority and from Cerberus with high priority.

2) Including DMA transactions from DMA channels with medium priority.

3) Including DMA transactions from DMA channels with low priority, MLI and from Cerberus with low priority.

SBCU_DBADRT
SBCU Debug Trapped Address Register

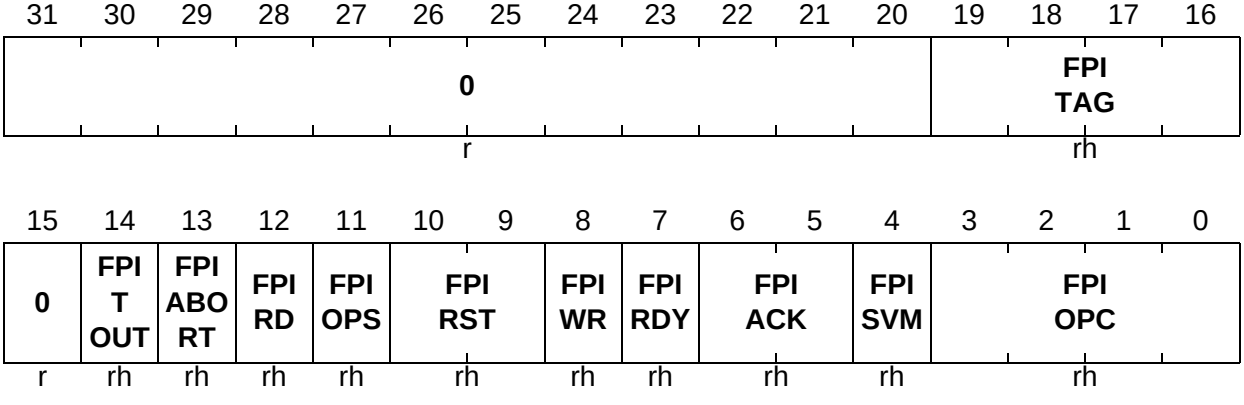
(048_H) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
FPIADR	[31:0]	rh	**FPI Bus Address Status** This register contains the FPI Bus address that was captured when the OCDS break trigger event occurred.

SBCU_DBBOST
SBCU Debug Trapped Bus Operation Signals Register

($04C_H$) **Reset Value: 0000 3180$_H$**



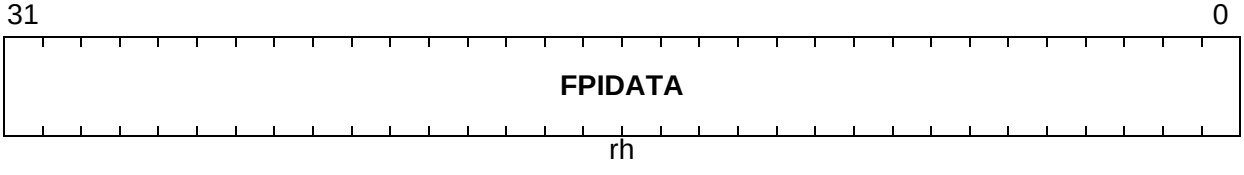
Field	Bits	Type	Description
FPIOPC	[3:0]	rh	**FPI Bus Opcode Status** This bit field indicates the type (opcode) of the FPI Bus transaction captured from the FPI Bus signal lines when the BCU break trigger event occurred. 0000_B Single byte transfer 0001_B Single half-word transfer 0010_B Single word transfer 0100_B 2-word block transfer 0101_B 4-word block transfer 0110_B 8-word block transfer 1111_B No operation Other bit combinations are reserved.
FPISVM	4	rh	**FPI Bus Supervisor Mode Status** This bit indicates the state of the Supervisor Mode signal captured from the FPI Bus signal lines when the BCU break trigger event occurred. 0_B User mode 1_B Supervisor mode
FPIACK	[6:5]	rh	**FPI Bus Acknowledge Status** This bit field indicates the acknowledge signal status captured from the FPI Bus signal lines when the BCU break trigger event occurred. 00_B No special case 01_B Error 10_B Reserved 11_B Retry, slave did not respond
FPIRDY	7	rh	**FPI Bus Ready Status** This bit indicates the ready signal status captured from the FPI Bus signal lines when the BCU break trigger event occurred. 0_B Last cycle of transfer 1_B Not last cycle of transfer

Field	Bits	Type	Description
FPIWR	8	rh	**FPI Bus Write Indication Status** This bit indicates the write signal status captured from the FPI Bus signal lines when the BCU break trigger event occurred. 0_B Single write transfer or write cycle of an atomic transfer 1_B No operation or read transfer
FPIRST	[10:9]	rh	**FPI Bus Reset Status** This bit field indicates the reset signal status captured from the FPI Bus signal lines when the BCU break trigger event occurred. 00_B Reset of all FPI Bus components 11_B No reset Others Reserved
FPIOPS	11	rh	**FPI Bus OCDS Suspend Status** This bit indicates the OCDS suspend signal status captured from the FPI Bus signal lines when the BCU break trigger event occurred. 0_B No OCDS suspend request is pending 1_B An OCDS suspend request is pending
FPIRD	12	rh	**FPI Bus Read Indication Status** This bit indicates the read signal status captured from the FPI Bus signal lines when the BCU break trigger event occurred. 0_B Single read transfer or read cycle of an atomic transfer 1_B No operation or write transfer
FPIABORT	13	rh	**FPI Bus Abort Status** This bit indicates the abort signal status captured from the FPI Bus signal lines when the BCU break trigger event occurred. 0_B A transfer that has already started was aborted 1_B Normal operation
FPITOUT	14	rh	**FPI Bus Time-out Status** This bit indicates the time-out signal status captured from the FPI Bus signal lines when the BCU break trigger event occurred. 0_B Normal operation 1_B A time-out event was generated

Field	Bits	Type	Description
FPITAG	[19:16]	rh	**FPI Bus Master TAG Status** This bit field indicates the master TAG captured from the FPI Bus signal lines when the BCU break trigger event occurred (see **Table 4-15**). The master TAG identifies the master of the transfer which generated BCU break trigger event. 1001_B Peripheral Control Processor (PCP) 1010_B DMA Controller (high-priority channels) 1011_B LFI Bridge Others Reserved
0	15, [31:20]	rh	**Reserved** Read as 0; should be written with 0.

SBCU_DBDAT
SBCU Debug Data Status Register

(050_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
FPIDATA	[31:0]	rh	**FPI Bus Data Status** This register contains the FPI Bus data that was captured when the OCDS break trigger event occurred.

4.6.4.5 SBCU Service Request Control Register Description

In case of a bus error, the SBCU generates an interrupt request to the selected service provider (usually the CPU). This interrupt request is controlled through a standard service request control register.

SBCU_SRC
SBCU Service Request Control Register
($0FC_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SETR	CLRR	SRR	SRE	0	TOS	0		SRPN							
w	w	rh	rw	r	rw	r		rw							

Field	Bits	Type	Description
SRPN	[7:0]	rw	**Service Request Priority Number**
TOS	10	rw	**Type of Service Control** 0_B CPU service is initiated 1_B PCP request is initiated
SRE	12	rw	**Service Request Enable**
SRR	13	rh	**Service Request Flag**
CLRR	14	w	**Request Clear Bit**
SETR	15	w	**Request Set Bit**
0	[9:8], 11, [31:16]	r	**Reserved** Read as 0; should be written with 0.

Note: Further details on interrupt handling and processing are described in the Interrupt Chapter of this TC1797 User´s Manual.

4.7 On Chip Bus Master TAG Assignments

Each master interface on the FPI Bus and on the LMB Bus is assigned to a 4-bit (FPI Bus) or 3-bit (LMB Bus) identification number, the master TAG number (see **Table 4-15**). This makes it possible for software debug and MCDS purposes to distinguish which master has performed the current transaction (see **"LBCU_LEATT" on Page 4-11**[1)] and **"SBCU_DBADRT" on Page 4-54**).

Table 4-15 On Chip Bus Master TAG Assignments

TAG-Number	Module	Location	Description
0000_B	LFI	LMB	LFI Master Interface to LMB
0001_B	-	-	Reserved
0010_B	PMI	LMB	Program Memory Interface
0011_B	-	-	Reserved
0100_B	DMI	LMB	Data Memory Interface
0101_B	-	-	Reserved
0110_B	-	-	Reserved
0111_B	DMA	LMB	DMA Controller Master Interface on LMB
1000_B	-	-	Reserved
1001_B	PCP	SPB	Peripheral Control Processor
1010_B	DMA	SPB	DMA Controller Master Interface on SPB
1011_B	LFI	SPB	LFI Master Interface to SPB
1100_B	-	-	Reserved
1101_B	-	-	Reserved
1110_B	-	-	Reserved
1111_B	-	-	Reserved

1) Pls. note that the TAG bit field in the register **"LBCU_LEATT" on Page 4-11** represents only bit 0-2 of the TAG-Number as the TAG number of all On Chip Bus master interfaces connected to LMB is 0.

5 Program Memory Unit (PMU)

The devices of the AudoF family have at least one Program Memory Unit. This is named “PMU0”. The high-end devices can have additional PMUs which are named “PMU1”, …

The TC1797 has PMU0 and PMU1.

The PMU0 contains the following submodules:

- The Flash command and fetch control interface for Program Flash and Data Flash.
- The Overlay RAM interface with Online Data Acquisition (OLDA) support.
- The Boot ROM interface.
- The Emulation Memory interface.
- The Local Memory Bus LMB slave interface.

Following memories are controlled by and belong to the PMU0:

- 2 Mbyte of Program Flash memory (PFLASH).
- 64 Kbyte of Data Flash memory (DFLASH). It can represent up to 16 Kbyte EEPROM.
- 16 Kbyte of Boot ROM (BROM).
- 8 Kbyte Overlay RAM (OVRAM).

The PMU1 contains the following submodules:

- The Flash command and fetch control interface for Program Flash.
- The Local Memory Bus LMB slave interface.

Following memories are controlled by and belong to the PMU1:

- 2 Mbyte of Program Flash memory (PFLASH).

The PFLASH of PMU1 provides the same features as the PFLASH of PMU0, e.g. small logical sectors and read/write/OTP protection. The combination of two independent PMUs supply the device with additional functionality, such as Read while Write (RWW), Write while Write (WWW) or concurrent data and instruction accesses, if those are operating on different PMUs.

The **Figure 5-1** shows the block diagram of the PMU0:

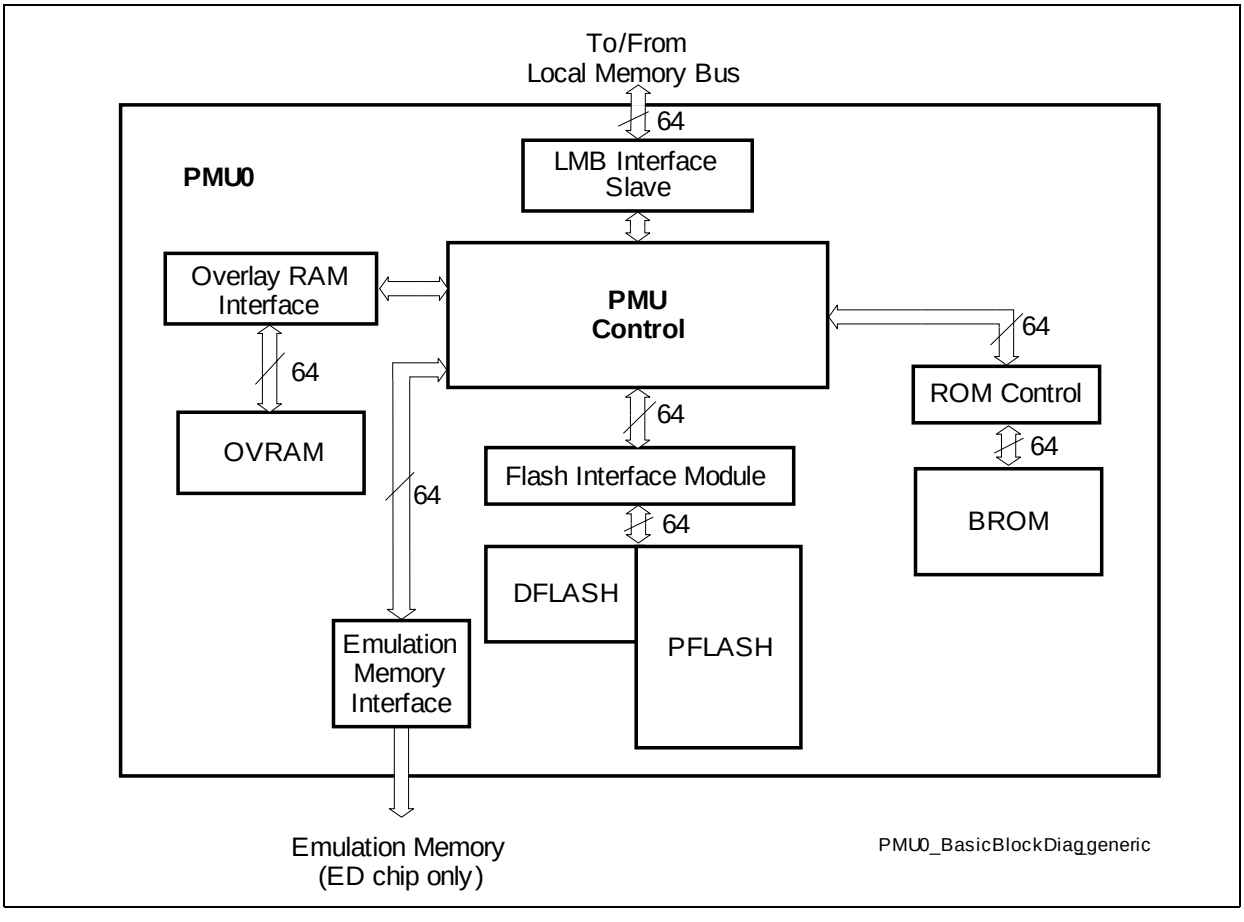


Figure 5-1 PMU0 Basic Block Diagram

The PMU1 and further PMUs have the reduced functionality as depicted in **Figure 5-2**.

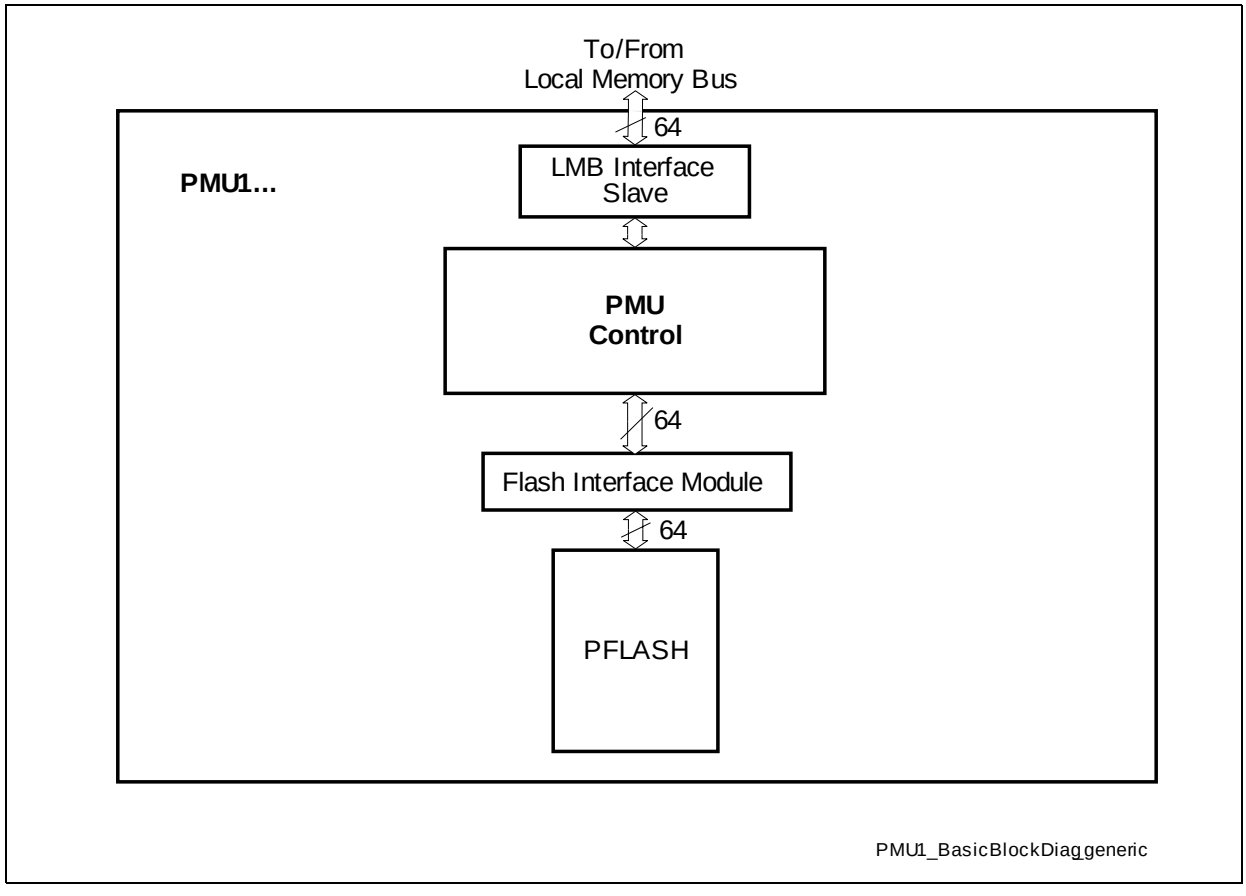


Figure 5-2 PMU1 Basic Block Diagram

5.1 BootROM

The BootROM in PMU0 has a capacity of 16 KB, organized with double-words of 64 bits. The BootROM consists basically of two parts, used for:

- startup and boot SW (also called firmware), and
- factory test routines.

5.1.1 Addressing

The BootROM is visible at two locations, as can be seen in the memory map:

- In segment 8_H (cached space) starting at location $8FFF\ C000_H$
- In segment A_H (non-cached space) starting at location $AFFF\ C000_H$

After any reset, the hardware-controlled start address is $AFFF\ C000_H$. At this location, the first instruction of the startup procedure is stored and started. Another start location after reset is not supported, guaranteeing, that always the only one startup firmware within the BootROM is executed after reset.

5.1.2 Firmware Program Structure

The different sections of the firmware in BootROM provide startup and boot operations after reset, such as:

- The startup SW, which is the main control firmware in the BootROM executed after reset.
- The bootstrap loaders, which load a program code via a serial interface into the Scratchpad RAM (SPRAM) of the PMI and start its execution.
- The alternate boot modes, which provide a CRC checksum test of Flash regions before starting the user program
- The emulation support, if the chip is an emulation device.

The BootROM also includes special routines, which are used for testing, stressing and qualification of the component.

5.2 Overlay RAM and Data Acquisition

The overlay memory OVRAM is provided in the PMU0 especially for redirection of program memory accesses to the OVRAM by using the data overlay function. The data overlay functionality itself is controlled in the DMI module to avoid any performance penalty during the execution of redirection, and to support also external memories.

For online data acquisition (OLDA) of application or calibration data a virtual 32 KB memory range is provided which can be accessed without error reporting. Accesses to this OLDA range can also be redirected to the overlay memory.

5.2.1 Internal Overlay Memory

The capacity of the OVRAM is 8 KB. The base address of the OVRAM is A/8FE8 0000_H. The internal Overlay Memory OVRAM is available in both, the production device and the Emulation Device. Write accesses to the Overlay Memory OVRAM are possible in byte, half-word, word and double-word widths. Read accesses are performed in 64-bit width. Read-Modify-Write accesses are not supported.

As all other SRAMs in the device, also the OVRAM is parity protected. The parity bit generation and detection is enabled via the parity enable bit PEREN in the OVRCON register. After enabling, the OVRAM should be initialized by the user before any read access is performed. Also byte or half-word write accesses to the OVRAM may result in parity error detection (due to read-modify-write) if the memory has not been initialized before the access. A parity error is reported to the SCU for control of error indication and of NMI trap (disabled after reset).

5.2.2 Online Data Acquisition (OLDA)

Calibration is additionally supported by an OLDA memory range of up to 32 Kbyte, which is a virtual memory and physically only available, if it is redirected (by user-controlled overlay function) to the internal OVRAM, or in an Emulation Device to the Emulation Memory EMEM or to the overlay region in external memory. Thus, if not redirected, write accesses to the OLDA range are not really executed, and they do not generate a bus error trap[1)]. Read accesses to the OLDA range in production devices generate a bus error trap, if not redirected to a physically available overlay block. After redirection into the OVRAM parity errors may be reported also for byte/half-word write accesses (see above).

The base address of the virtual OLDA memory range is A/8FE7 0000_H (non-cached/cached space), the end address is A/8FE7 $7FFF_H$.

1) Attention: write accesses to the cached memory range will be executed by the TriCore by first reading this address to fill the cache line before merging the new data in. The read will trigger the bus error trap in this situation.

5.2.3 Access Performance

Write accesses to the PMU Overlay Memory OVRAM are performed with two cycles (read-modify-write because of parity generation). Read accesses to the OVRAM take one cycle. Additionally, one address select cycle is required in the PMU for read and write accesses. Bursts (BTR2) need one (read) or two (write) additional cycles.

5.2.4 Overlay Memory Control Register

The OLDA function and the parity protection of OVRAM are controlled via the Overlay RAM Control register OVRCON. Bit protection allows independent control of OLDA and parity function. Write accesses to this register are permitted only in Supervisor Mode SV, read accesses in User Mode or SV. The OVRCON register is cleared with the application (= class 3) reset.

The register is defined as follows:

PMU0_OVRCON
Overlay RAM Control Register (F800 0520$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PB1ERR	PB0ERR	PB1R	PB0R	PB1W	PB0W	PERENR	PPERCTR	0	0	0	0	0	0	POLDAEN	OLDAEN
rwh	rwh	rh	rh	rw	rw	rw	rw	r	r	r	r	r	r	w	rw

Field	Bits	Type	Description
OLDAEN	0	rw	**Online Data Acquisition Enabled** 0_B Trap generation on write access to virtual OLDA memory is enabled 1_B The write trap is disabled although the OLDA memory is virtual and not available in both, the production device and the Emulation Device.

Field	Bits	Type	Description
POLDAEN	1	w	**Protection Bit for OLDAEN** 0_B Bit protection: Bit OLDAEN remains unchanged with register OVRCON write 1_B OLDAEN can be changed with current write access to register OVRCON This bit enables OLDAEN write during OVRCON write. Return 0 if read.
PPERCTR	8	w	**Protection Bit for Parity Control Bits** 0_B Bit protection: The writable parity control bits (PEREN, PB0W, PB1W, PB0ERR, PB1ERR) remain unchanged with register OVRCON write. 1_B The parity control bits can be changed with current write access to register OVRCON. Return 0 if read.
PEREN	9	rw	**Parity Generation and Error Reporting Enable** 0_B Parity error in Overlay Memory OVRAM is not reported to SCU. Parity test with use of parity write bits PB0/1W for OVRAM write accesses is enabled. 1_B The parity error reporting and thus the activation of error signals to SCU is enabled. Automatic parity generation is active during write accesses to OVRAM. Use of PB0/1W bits for testing is disabled.
PB0W	10	rw	**Parity Bit 0 Write Bit for Parity Test** This bit is used as parity bit for word 0 write accesses to the OVRAM, if PEREN=0 (parity error reporting is disabled / test enabled).
PB1W	11	rw	**Parity Bit 1 Write Bit for Parity Test** This bit is used as parity bit for word 1 write accesses to the OVRAM, if PEREN=0 (parity error reporting is disabled / test enabled).
PB0R	12	rh	**Parity Bit 0 Read Bit** Parity bit of word 0 of last read access to OVRAM
PB1R	13	rh	**Parity Bit 1 Read Bit** Parity bit of word 1 of last read access to OVRAM
PB0ERR	14	rwh	**Parity Bit 0 Error** If set, this bit shows a parity error of word 0 (low word within addressed double-word) during a read access.

Field	Bits	Type	Description
PB1ERR	15	rwh	**Parity Bit 1 Error** If set, this bit shows a parity error of word 1 (high word within addressed double-word) during a read access.
0	[31:16], [7:2]	r	**Reserved** Reserved for future use. Write 0, read 0.

5.3 Emulation Memory Interface

In the Emulation Device, an Emulation Memory (EMEM) is provided, which can be used for calibration via program memory or OLDA overlay. Its base address is A/8FF0 0000_H. As for Flash and OVRAM accesses, cached (segment 8_H) and non-cached (segment A_H) accesses can be used for EMEM accesses via PMU.

The Emulation Memory interface shown in **Figure 5-1** is a 64-bit wide memory interface that controls the CPU-accesses to the Emulation Memory in the Emulation Device. All widths of write accesses are supported (byte, half-word, word, double-word). CPU-controlled Load-Modify-Store accesses (with LDMST instruction) are not supported.

In the TC1797 production device, the EMEM interface is always disabled. A CPU read access from the Emulation Memory region causes a DSE trap and an LMB bus error. If the Emulation Memory region read access is initiated by a SPB master (e.g. PCP), additionally a SPB error interrupt is generated. Per default, write accesses to the Emulation Memory by any master cause an LMB bus error trap in production device.

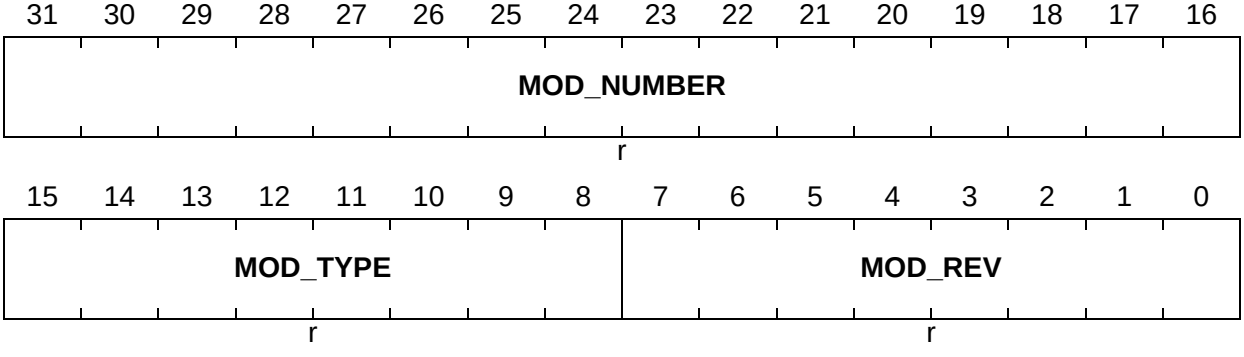
In the Emulation Device, a LMB bus error trap is reported by the PMU, if the CPU access can’t be handled by the EMEM, for example, when the CPU accesses a trace memory tile in EMEM. In this case, the EMEM access is aborted by the PMU.

Similar to the internal 8 KB OVRAM, the EMEM can also be used for overlay blocks dedicated to blocks in the internal Program Flash or to the virtual OLDA memory, redirecting (in DMI) Flash/OLDA addresses to the Emulation Memory. Also the external memory is supported for redirection of blocks into the EMEM.

5.4 PMU ID Register

The PMU_ID register is a read-only register, thus write accesses lead to a bus error trap. Read accesses are permitted in Supervisor Mode SV and in User Mode. The PMU_ID register is defined as follows:

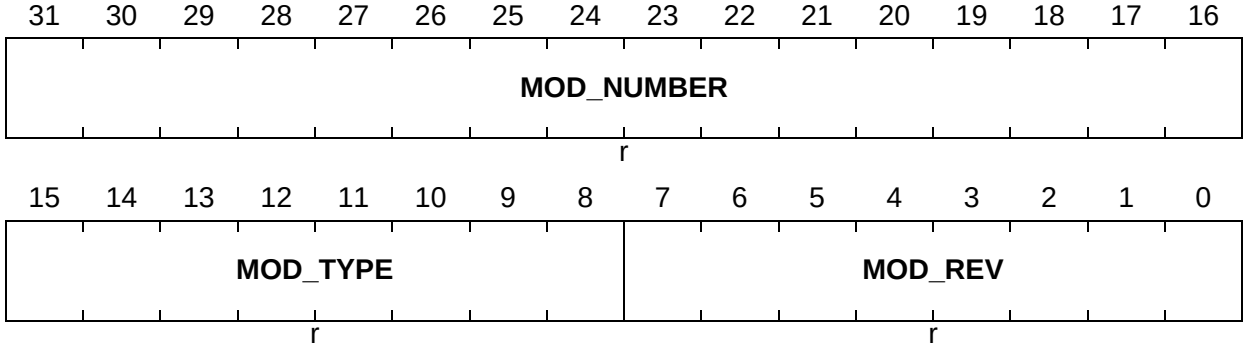
PMU0_ID
PMU0 Identification Register (F800 0508$_H$) Reset Value: 0050 C0XX$_H$



Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision Number** MOD_REV defines the module revision number. The value of a module revision starts with 01$_H$ (first rev.).
MOD_TYPE	[15:8]	r	**Module Type** This bit field is C0$_H$. It defines the module as a 32-bit module.
MOD_NUMBER	[31:16]	r	**Module Number Value** This bit field defines the module identification number for PMU0.

PMU1_ID

PMU1 Identification Register (F800 0608$_H$) Reset Value: 0051 C0XX$_H$



Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision Number** MOD_REV defines the module revision number. The value of a module revision starts with 01$_H$ (first rev.).
MOD_TYPE	[15:8]	r	**Module Type** This bit field is C0$_H$. It defines the module as a 32-bit module.
MOD_NUMBER	[31:16]	r	**Module Number Value** This bit field defines the module identification number for PMU1.

5.5 Tuning Protection

The special tuning protection support represents a security function provided additionally to Flash read/write/OTP protection (see **Page 5-24** and **Chapter 5.6.5**) and additionally to the Alternate Boot Mode (see BootROM spec).

For details on the tuning protection please contact your Infineon representative.

5.6 Program and Data Flash

This chapter describes the embedded Flash module of the TC1797.

Note: In TC1797, two Flash modules are included. In this case, the second Flash module is controlled by PMU1 and it has the same functionality as the first one (only exception: second without Data Flash). The Flash modules of PMU0 and PMU1 can operate in parallel. In the following, only the Flash module FLASH0 of PMU0 is described - with additional hints, concerning the double-Flash system.

5.6.1 Introduction

The embedded Flash module of TC1797 includes 2 MB of Flash memory for code or constant data (called Program Flash) and 64 Kbyte of additional Flash memory used for emulation of EEPROM data (called Data Flash).

The Program Flash is realized as one independent Flash bank, whereas the Data Flash is built of two Flash banks, allowing the following combinations of concurrent Flash operations:

- Read code or data from Program Flash, while one bank of Data Flash is busy with a program or erase operation.
- Read data from one bank of Data Flash, while the other bank of Data Flash is busy with a program or erase operation.
- Program one bank of Data Flash while erasing the other bank of Data Flash, read from Program Flash.

Since in TC1797 the Flash modules of the two PMUs can work in parallel, further combinations of concurrent operations are supported if those are operating on different PMUs, e.g.:

- Read data from Flash1 while accessing code from Flash0.
- Read code or data from one Flash while the other Flash is busy with program or erase operation.
- Both Flash modules are concurrently busy with program or erase operation.

Both, the Program Flash and the Data Flash, provide error correction of single-bit errors within a 64-bit read double-word, resulting in an extremely low failure rate. Read accesses to Program Flash are executed in 256-bit width, to Data Flash in 64-bit width (both plus ECC). Read accesses are very efficiently controlled, supporting a read line buffer for Program Flash with buffer hit control and a separate read buffer for Data Flash. Single-cycle burst transfers of up to 4 double-words and sequential prefetching with control of prefetch hit are additionally supported for Program Flash. Accesses to Data Flash do not disturb buffered data and prefetched data in Program Flash for hit control.

The minimum programming width is one page, consisting of 256 bytes in Program Flash and 128 bytes in Data Flash. Concurrent programming and erasing in Data Flash is performed using an automatic erase suspend and resume function.

The whole Flash module is divided into the following two sub-modules:

- The Flash array module (FAM) with one Flash bank PFLASH and with two Flash banks DFLASH, with separate read buffers for PFLASH and DFLASH, with Flash standard interface FSI, including the Flash array controller, with page write buffers (assembly buffers) and voltage generators.
- The Flash interface and control module (FIM), which controls the execution of Flash commands and which is responsible for the error correction and ECC generation. It is interfaced via LMB bus to the PMI module for instruction accesses and to the DMI and DMA modules for data accesses.

The Flash interface module is main part of the Program Memory Unit PMU. An overview of system integration and system architecture is presented in the TC1797 block diagram.

A basic block diagram of the Flash Module is shown in the following **Figure 5-3**.

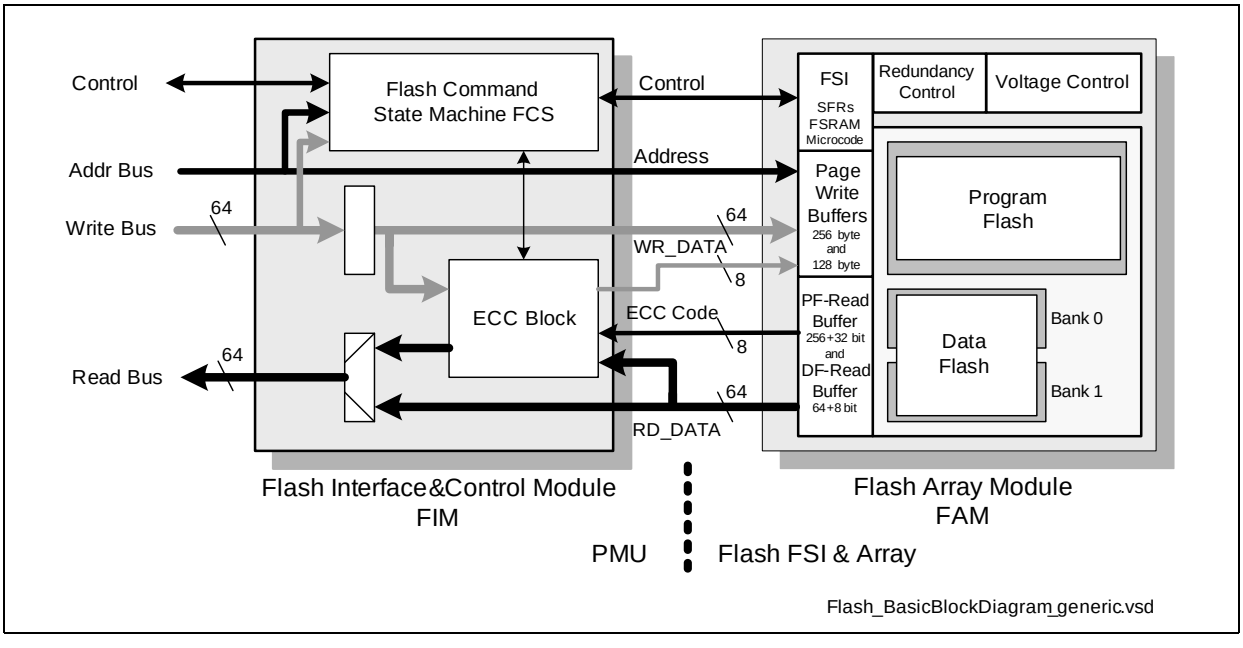


Figure 5-3 Basic Block Diagram of Flash Module

Summary of Program Flash Features and Functions

Note: All performance and quality figures in this document, especially access timings, wait state settings, duration of flash processes, endurance and retention settings are just indicative. Binding figures are published in the data sheet.

- 2 Mbyte on-chip Program Flash in PMU0.
- 2 Mbyte on-chip Program Flash in PMU1.
- Any use for instruction code or constant data.
- Double PMU system approach:
  - Concurrent read access of code and data.
  - Read while write (RWW).

  - Concurrent program/erase in both modules.
- 256 bit read interface (burst transfer operation).
- Dynamic correction of single-bit errors during read access.
- Read access time: 26 ns.
- Transfer rate in burst mode: one 64-bit double-word per cycle.
- Sector architecture:
  - Eight 16 Kbyte[1)], one 128 Kbyte and seven 256 Kbyte sectors.
  - Each sector separately erasable.
  - Each sector lockable for protection against erase and program (write protection).
- One additional configuration sector (not accessible to the user).
- Read protection for whole Flash. Combined with whole Flash write protection - thus supporting protection against Trojan horse programs.
- Multi-level sector write protection with support of re-programmability; one level dedicated to OTP protection with ROM functionality (locked forever).
- Comfortable password checking for temporary disable of write or read protection.
- User controlled configuration blocks (UCB) in configuration sector for keywords and for sector-specific lock bits (one block for every user; up to three users).
- Pad supply voltage also used for program and erase (no VPP pin).
- Efficient 256 byte page program operation.
- Programming time: typ. 5 msec per page.
- All Flash operations controlled by CPU per command sequences (unlock sequences) for protection against unintended operation.
- Write state machine for automatic program and erase, including verification of operation quality.
- End-of-busy as well as error reporting with interrupt and bus error trap.
- Support of margin check.
- Erase time per sector: max. 5 sec.
- Delivery in erased state (read all zeros).
- Global and sector specific status information.
- Overlay support for calibration applications.
- Configurable wait state selection for different CPU frequencies
- Endurance = 1000; allowed number of program/erase cycles per physical sector; reduced endurance of 100 per 16 KB sector.
- Operating lifetime (incl. Retention): 20 years with endurance = 1000.
- For further operating conditions see data sheet.

Summary of Data Flash Features and Functions

- 64 Kbyte on-chip Flash, configured in two independent Flash banks of equal size.
- Sector architecture: one sector per bank.
- 64 bit read interface.

1) The 16 Kbyte sectors are “logical” sectors. See **Page 5-17**.

- Erase/program one bank while data read access from the other bank.
- Programming one bank while erasing the other bank using an automatic suspend/resume function.
- Dynamic correction of single-bit errors during read access.
- Read access time: 26 ns.
- 128 byte pages to be written in one step.
- Selectable read/write protection in combination with PFlash read protection.
- Programming time: typ. 5 msec per page.
- Operational control per command sequences (unlock sequences, same as those of Program Flash) for protection against unintended operation.
- End-of-busy as well as error reporting with interrupt and bus error trap.
- Write state machine for automatic program and erase.
- Margin check for detection of problematic Flash bits.
- Erase time per sector: max. 1 sec. (increased for low frequencies).
- Endurance = 30000 (can be device dependent); i.e. 30000 program/erase cycles per sector are allowed, with a retention of min. 5 years
- Dedicated DFlash status information.
- Other characteristics: Same as Program Flash.

5.6.2 Architectural and Operational Overview

In the following, an overview of the internal structure and of operations is presented.

5.6.2.1 Sector and Page Architecture

The Program Flash as well as the Data Flash memory are characterized by their sector architecture and by their page structure. Sectors are Flash memory partitions of different sizes. Some Flash operations are dedicated to and are only performed on complete sectors, such as sector erase operations and installation of sector write protection. Programming operations are generally dedicated to pages (byte, word or double-word program is not supported).

Page and Wordline Structure

The whole Flash memory is divided into pages of defined and always identical size, which depends on Flash type. Program Flash and Data Flash have different page sizes:

- Program Flash: Page size is 256 Byte (32 double-words).
- Data Flash: Page size is 128 Byte (16 double-words).

The page size is defined by the wordline length of the array: a wordline always consists of two sequential pages with according even and odd page addresses.

Sector Partitioning in Program Flash

The Program Flash memory is divided into the following sectors:

- Eight 16 Kbyte logical sectors, together building two 64 Kbyte physical sectors,
- one 128 Kbyte (physical) sector,
- and seven 256 Kbyte (physical) sectors.

Additionally a configuration sector is implemented, which is not user-accessible.

Physical and Logical Sectors

The maximum number of program/erase cycles is always related to physical sectors. Thus the specified endurance of 1000 belongs to the sectors with 64 Kbyte and more capacity, but not to the small 16K sectors.

The 16K sectors are called logical sectors, because they are not physically separated from their neighboring sectors. They are “logically” separated simply by definition of 16K address ranges within a 64K physical sector, to avoid the area overhead necessary for physical sector separation. Because all logical sectors within a physical sector use the same bit lines, the maximum number of erase cycles without refreshing must be limited to 100 for every logical sector. This means, that after 100 erase operations to a 16K sector the whole 64K sector (which includes the 16K sector) has to be refreshed (erased and re-programmed), before the next 100 erase cycles can be performed on the **same** 16K sector. Thus, all four 16K sectors of a physical 64K sector can be erased 100

times before refreshing the 64K sector. In total, also for each logical sector the max. number of erase cycles is 1000. Erase of the 64K physical sector can be performed with one 64K erase operation or with four 16K erase operations.

Data Flash Sectors

The 64 Kbyte Data Flash is divided into two erasable (physical) sectors of equal size. The two sectors are identical to the two Flash banks, which are defined to support concurrent operations. Only in case of read protection, full write protection of DFlash is supported (can be disabled), but it is not possible to select sector write protection only for DFlash sectors.

5.6.2.2 Data Flash and EEPROM Emulation

The Data Flash consists of two independent Flash banks (contrary to the Program Flash). A Data Flash bank also represents a sector, which can be erased only completely as a unit, and can be programmed page by page. The structure with two independent Data Flash banks is used to allow simultaneous read or program operations on one bank while erasing the other bank in the background.

The general structure of the Data Flash is selected to support the emulation of an EEPROM. In this context, the main difference between a Flash memory and an EEPROM is the endurance, combined with a shorter retention. For EEPROM, an endurance of e.g. 120 000 write/erase cycles is required, what is not supported with the standard Flash memory.

For EEPROM emulation and thus for increasing the endurance, the Data Flash is used like a circular buffer-memory: The newest data updates are programmed always above the last programmed page. When the top of sector is reached, all actual data (representing the actual EEPROM region) are copied to the bottom area of the next sector and the last sector is then erased. This round robin procedure, using multi-fold replications of the emulated EEPROM size (called EEPROM regions), significantly increases the endurance. Thus, the endurance can be selected by changing the size of emulated EEPROM.

Example 1

The DFLASH is logically divided into four regions (i.e. two per bank) that operate as a circular buffer memory. At a time, one of the four regions is always regarded as active EEPROM region. The active EEPROM region is simply identified by the used region with the highest address within the active DFLASH bank.

The active EEPROM region is held as a EEPROM mirror in an on-chip RAM area. After a reset operation, the active EEPROM region is copied into the RAM. When the EEPROM data must be updated, the next consecutive region within the DFLASH circular

buffer becomes the active EEPROM region. The “old” DFLASH bank can be erased, when the active EEPROM region has been switched to the “new” DFLASH bank.

As a result of the continuously changing assignment of the active EEPROM region in a circular buffer, the DFLASH memory cells of one EEPROM region can be erased/programmed 4-times as often as one physical region, resulting in an 4-fold endurance for one EEPROM region. Additionally, a reduced retention is assumed for EEPROM data supporting a higher endurance (e.g. 30000 for Data Flash with retention of 5 years). Thus, for the above described example with four regions and with a retention of 5 years, the endurance for an emulated EEPROM region is increased to 120000 write/erase cycles.

Example 2

For emulation of more complex EEPROM structures, several EEPROM region types can be defined in parallel, with different endurance or update requirements. Each region type is identified by an identification label. The active region of each type is identified by the highest address of all regions with the same label. If a “new” Flash bank is entered, all active regions are copied into the new bank, so that the “old” DFLASH bank can be erased. It is recommended to delay the copy (and erase) operation so that more active regions in the “old” bank are renewed (updated) in the “new” bank anyway.

Depending on the number of pages used for dynamic programming of EEPROM data, the endurance of an EEPROM region can additionally be optimized. If only that part of EEPROM region, which has been changed, is re-programmed during update-operation, the endurance is further increased, because not always a full EEPROM region is wasted when only one wordline (two pages) has to be updated. It is thus necessary to indicate old (invalid) wordlines. Therefore, a wordline can be marked as invalid wordline by re-programming one (or both) pages of the wordline with all-ones (only this kind of twofold programming of a page is allowed in Data Flash and in Program Flash). For this “invalidation stamp”, a correct ECC code is ensured because the ECC algorithm is selected in that way that either for the erased state (all zeros) or for the invalidated state (all ones) the according ECC code (also all zeros or all ones) is correct.

Note: Only two page writes for one wordline are allowed. Therefore, if a third page write is executed for the invalidation stamp, not only this page, but the whole wordline including this page is in an invalid state.

5.6.2.3 Operational Overview

In general, the operations of Program Flash and Data Flash are controlled identically. Therefore, in the following, the operational overview is mainly presented only for the Program Flash. When necessary, additional explanations are made for the Data Flash.

Standard Read

In standard read mode (the normal operating mode) the Flash memory appears like an on-chip ROM. The Flash array module offers an asynchronous read access of 256-bit read data (Data Flash: 64-bit read data) with an access time of 26 ns (Data Flash: 26 ns). Depending on the clock frequency of the PMU it has to wait a defined number of wait cycles (see **Table 5-1**). Instruction read accesses are 64-bit accesses. Because the read data width of Program Flash is 256 bit, always a full burst transfer of four 64-bit double-words is executed. Data operand reads are 64-bit accesses out of PFLASH or DFLASH. In case of data access in cached address space, two double-words are transferred. In case of non-cached data accesses only the addressed DW is accessed and directly transferred to the CPU.

Note: The Flash module delivers always 64-bit read data. The addressed data (word, half-word, byte depending on data type) within the double-word read data is selected by the CPU.

The Flash addresses are mapped into the total address space of the controller with different base addresses (see **"Address Mapping" on Page 5-29**). The base addresses of PFLASH and of the DFLASH banks of all implemented PMUs were defined to allow compatibility between all devices of the AudoF family.

The physical address range of the 16 Kbyte configuration sector starts also with address zero, and it is mapped to the same base address as the Program Flash, but read and direct write accesses to the config-sector are not possible for the user.

To eliminate any possibility of read data corruption, the Flash module provides ECC with SEC-DED (Single Error Correction, Double Error Detection) capability over a 64-bit double-word. For verify operations, the normal read can be combined with a margin check (more critical control of sense amps). Single and double-bit errors are indicated in the Flash status register and, if enabled, reported to the CPU by means of error interrupts; the double-bit error causes a hardware trap (if not disabled for margin checks).

Operation Control with Command Sequences

All operations besides normal read operations are initiated and controlled by command sequences written to the flash interface&command state machine. During normal read operations, a first write cycle to the Flash address space is automatically interpreted as a command cycle, initiating a command sequence. The different write cycles of command sequences are not only used for operation definitions such as program

commands or sector erase commands, but also as fail-safe and unlock cycles in order to protect against inadvertent writes. Some commands which do not directly control Flash array operations are implemented as single cycle commands.

All command cycles are write (store) cycles to the Flash. During command cycles, the low order 16 bits of the address bus (A15–A0) define the Flash command address, and only command cycles with addresses to sectors, pages and wordlines also use the Flash address bits A20–A16. Additionally, the command addresses are mapped into the total address space of the controller by using base addresses dedicated to the Flash bank to be operated on (see **“Address Mapping” on Page 5-29**). This has especially to be considered for the two PMUs and Flash arrays in TC1797, which are mapped to different address spaces. Thus, command sequences belong to that Flash module which is addressed by the command sequence.

The write data of a command cycle define the 8-bit operation code or security pattern in case of unlock cycles, the 32-bit password for protection disable cycles, or it represents the 64 or 32-bit data to be programmed. Improper command cycles or interrupted command sequences are indicated by the sequence error flag in the Flash status register.

Note: User code, that writes command sequences to the Program Flash, should not be executed from the internal Flash; it shall be located in other internal or external program memory, e.g. in the scratchpad SPRAM. But user code, that writes command sequences to the Data Flash, can be located in and executed from the Program Flash.

Note: The write cycles, belonging to a command sequence, may be buffered on its way to the Flash in store/write buffers. To maintain data coherency (strictly in-order sequence of command cycles is mandatory) and to guarantee immediate transfer of the command cycles to the PMU, all write cycles to the Flash must access the Flash in its non-cached address space. Additionally, it is recommended to include a dummy read (ld.w) instruction to a PMU register (e.g. PMU_ID) after the last write cycle of a command sequence to flush the write buffers.

Additional hints are available in chapter **“Application Hints and Guidelines” on Page 5-92**, possible error conditions and their reporting in FSR are summarized in **Chapter 5.6.6.3**.

Programming Control

Programmed Flash bits deliver a ‘1’ value. The Flash module provides a page assembly mechanism for all program (data write) operations in Program Flash and Data Flash. This assembly mechanism requires to assemble 256 bytes (Data Flash: 128 bytes) in a page assembly buffer before this assembly buffer is being written to Flash in one program cycle.

Page write operations are executed by load/store command sequences, comprising the following three steps:

1. Start with the ‘Enter Page Mode’ command. This command’s address determines in its high order address bits also the bank of Flash (Program Flash or Data Flash bank) to be programmed, in TC1797 either belonging to PMU0 or to PMU1.
2. Execute 32 (Data Flash: 16) ‘Load Page’ commands to transfer double-words or execute 64(32) commands to transfer words to the respective page assembly buffer. Mixed transfers of words and double-words are not allowed (error indication). The first double-word is loaded into the page assembly buffer to the location with address zero (starting address of page register). The address of following double-words within page register is internally controlled by incrementing the start address; thus the write cycles have always the same address, pointing to the Flash bank to be programmed. For every double-word (or 2 words), the ECC code is internally generated and also stored in the assembly buffer.
3. ’Write Page’ command sequence (4 cycles) to program the whole 64 (DFlash: 32) words within the page assembly buffer in one step into the flash memory. The page address is defined by the last command cycle. The write data of the last command cycle is a confirm pattern. All base addresses of command cycles have to point to the Flash bank to be programmed.

A write command for a not completely filled buffer is executed (not loaded words of page use ‘old’ contents of assembly buffer and are therefore undefined) but reported to the user by error indication in the status register. If a memory region shall be programmed with always the same pattern, it is possible to use the ‘old’ contents of the assembly buffer for several page write operations, if always the Enter Page Mode command is directly followed by the Write Page command.

Command cycles addressing a busy Flash bank cause a stall of the bus system and the sending master until the busy clears.

After receiving the Write Page command the module executes the program operation. The page of 256 (Data Flash: 128) bytes is programmed within typ. 5 msec (Data Flash: 5 msec). The programming algorithm is followed by a quality check to guarantee the specified retention for each programmed bit. Thereby, the quality check is performed with tightened read conditions on all programmed (’1’) bits as defined by the assembly buffer contents. Weak bits are re-programmed. If re-programming is no more possible, an error flag (VER) is set in the Flash Status Register FSR (see **Chapter 5.6.6.3**).

Termination of operation is indicated in the Flash Status Register and can also be configured to generate an end-of-busy interrupt. During a program operation, the minimum system clock is limited to 1 MHz. Any reset condition stops a program operation within max. 250 µsec. Such an error state can be recognized by proper handling and checking of the PROG status flag in Flash Status Register FSR (except for power-on reset).

Erase Control

This Flash module features a sector erase architecture. Erase is accomplished by executing the six-cycle erase command sequence including the sector address in its last cycle (for sector addressing see **Chapter 5.6.3.5**). All command cycles of the command sequence have to use base addresses which point to the Flash bank to be operated on. If the Flash bank, which is addressed, is still busy, the command cycle stall the bus system and the sending master. After the last cycle of the command sequence, the device automatically starts and controls the erase procedure. Start of operation is delayed, if another bank is busy with a write operation at that time. Termination of operation is indicated in the Flash status register and can also be enabled to generate an end-of-busy interrupt. After erase operation, the Flash memory delivers ‘0’ data with correct ECC code on read access.

For the physical sectors in Program Flash (64K, 128K and 256K) a maximum number (endurance) of 1000 erase and program operations is supported. For the (logical) 16K sectors in PFlash max. 100 erase cycles are allowed to the same sector without refresh (see **Chapter 5.6.2.1**). For the Data Flash sectors a maximum number of 30000 erase cycles are defined (can depend on the device).

Besides sector erase, also a special six-cycle block erase operation is available dedicated only to the User Configuration Blocks (UCB). This operation supports the change of user-specific Flash configuration regarding protection functions, i.e. the lock bits or keywords. Since a UC block comprises four pages, a 1 Kbyte block is erased with this operation. The maximum number of UCB changes is limited to 4. The UCB erase time is shorter than the sector erase time (max. 500 ms).

As the program operation, also the sector erase operation includes an erase quality check that identifies incorrectly erased bits in the Flash sector, and that indicates a verification error if weak bits can no more be corrected (see **Chapter 5.6.6.3**).

An erase operation is executed within max. 5 s (Data Flash: 1 s) (but depending on sector size and on CPU frequency). In Data Flash an automatic erase suspend function is implemented for immediate execution of program operations to the other DFlash bank. During an erase operation, the minimum system clock is limited to 1 MHz. A system reset condition stops an erase operation within max. 250 µsec. Such an error state can be recognized by proper handling and checking of the ERASE status flag in Flash Status Register FSR (except for power-on reset).

In-System Programming

In-system programming is fully supported. No special program voltage VPP is required. Because of the automatic execution of erase and program algorithms, write operations are reduced to transferring commands and data to the Flash and reading the status (see **Chapter 5.6.6.5** for hints). Because of the page assembly function, write data may be written to the Flash very comfortably and fast. User code that writes data to the page assembly buffer can be executed also from the same internal Program Flash (in

non-cached operating mode). User code that writes command sequences to the Program Flash must be executed from memory outside the addressed on-chip Flash memory (on-chip RAM or if available other Flash module or external memory).

Note: Simultaneous read accesses to Program Flash while erasing or programming the Data Flash are supported. Thus, command sequences for the Data Flash can be written by user code accessed from Program Flash. In Data Flash, also parallel write operations (programming one bank while erasing the other bank) are possible.

Register Access Control

Register accesses for polling the status register are allowed in any state, also during erase and program operations (but then executed out of other internal or external memory).

Read/Write/OTP Protection

The Flash module provides sophisticated security functions that protect against unauthorized readout or modification by any third party. For this, the Flash supports read protection for the whole Flash array (including Data Flash, if not separately disabled). The read protection automatically includes a global Flash write protection (Data Flash included, if not separately disabled) for protection against Trojan Horse programs. Additionally, sector-specific write and OTP protection for all sectors in Program Flash is provided. Write protected sectors are re-programmable (with passwords), OTP (One Time Programmable) protected sectors are locked for ever and have ROM functionality.

If read protection is installed and active, any Flash read access is disabled if the instruction execution is started after reset from another memory but from Program Flash itself, e.g. in case of bootstrap loader start after reset. The debug interface is enabled in this case, because the Flash module itself disables code and data accesses. In case of start after reset from internal Flash, Flash accesses are enabled, but the debug interface is locked by the firmware in BootROM, and the user himself has to control the debug interface. In any case the Flash-user can control by himself the access rights for instructions and data and for different masters (e.g. DMA controller). The Flash read protection can be temporarily disabled (with passwords), e.g. to change the access rights to the Flash memory or to perform a programming operation.

Write and OTP protection of Program Flash sectors is provided to protect code and constant data against any manipulation (e.g. against tuning). This feature will disable both, program and erase operations, for any combination of selected sectors. Sector protection is accomplished by sector specific lock bits which are programmed by the user directly into its ‘User Configuration Block’ (UC block) in the configuration sector. The installation of read and/or write protection will become active after the next reset.

Three different user-classes (also called levels) and thus three different UC blocks (UCB0, UCB1 and UCB2) are supported, two with separate keywords, one (UCB2) for

selection of OTP protected sectors, which can never be erased and re-programmed again. The user-classes are organized hierarchically: After disabling the protection with his password user 0 can program or erase all those sectors that are protected by him (even if they are protected also by user 1) but not sectors that are protected by user 2 and sectors that are protected only by user 1. User 1 can change sectors that are protected only by him but not sectors that are protected by user 0 or user 2.

As for read protection, also for short-term disablement of sector write protection a password checking feature is provided. Only if the passwords match the keywords in the UCB of the user, a temporarily unprotected state of his sectors is taken and erase or program commands are enabled for these unlocked sectors. If not finished by the single cycle command ‘Re-Enable Read/Write Protection’, the unprotected state is terminated with the next reset. User 2 protection (OTP sectors with ROM functionality) cannot temporarily be disabled and therefore does not need passwords.

Password checking is based on two keywords (together 64 bit) which are programmed by the user directly into his ‘User Configuration Block’ in the configuration sector (same procedure as used for lock bits). To avoid forever-lock conditions in case of disturbed keywords (e.g. because of power problem during programming of keywords), the keyword-correctness (users 0 and 1) has first to be checked by the user before a special confirmation code has to be written to a second wordline within the user’s configuration block. User 2 confirms with the confirmation code the protection configuration (OTP sectors get ROM functionality) in UCB2.

If any protection is configured and confirmed (thus installed correctly), this state is indicated in the Flash Status Register FSR. Additionally, protection summary bits are provided in FSR for every user indicating the installation of read protection (only user 0) or/and write protection and for indication of a temporarily disabled state (user 0 and 1). The locked state of every sector is indicated by sector specific flags in the Protection Configuration registers PROCON0, PROCON1 and PROCON2.

Note: If any sector of user 2 is locked for ever (OTP protected via PROCON2), investigation of FARs by Infineon is very limited because accesses to the Flash array registers (SFRs) are no more possible.

Note: The physical sector, including a 16 KB OTP sector, can never be erased; thus the endurance of its 16K logical sectors is limited to 100.

Note: If any protection is enabled also the related user block(s) in configuration sector is especially protected.

With the three possibilities for write protection — whole memory or sector specific with or without re-program capability and with all combinations of these possibilities — a flexible installation of write protection is supported to protect the Flash memory or parts of it from unauthorized programming and to provide virus-proof security for all system code sectors.

In TC1797 with its two PMUs and Flash modules, the read/write/OTP protection with all above described functions can separately and in parallel be used for PMU0/Flash0 and PMU1/Flash1, with different or the same passwords of the users. If read protection is installed in PMU0 for Flash0, read protection or all-sector write protection should also be installed in PMU1/Flash1 to inhibit any Trojan Horse attack. Installation of read protection only in Flash1 should not be used because system functions are HW-controlled by the read protection of Flash0. However, the startup-SW in BootROM checks both Flash modules for read protection to control the access rights to the Flash modules and the operation of the debug interface. Note, that in case of program start in internal Flash the debug interface is also locked if read protection is installed only in one of the two PMU/Flash modules.

Power Reduction

The Flash module supports a SCU-controlled sleep mode, globally requested by the Power Management System, and it supports an individual sleep mode, selected by the user as power down feature, if the Flash is not needed for certain processor states. If the sleep state is requested, all active or pending Flash array operations are at first correctly terminated, and only then the power down state is taken.

The wakeup from sleep ramps up the voltage generators, before the Flash read mode is activated again. Its duration is documented in the data sheet.

Additionally, the PMU/Flash module supports a dynamic power reduction mode, where idle states are used to disable the wordline drivers. If this mode is enabled, the prefetch functionality is simplified.

5.6.2.4 Flash Access Control and Performance

In TC1797 with its double-Flash system approach, data accesses with prefetching and buffer/line hit support can be performed in parallel to code accesses and without disturbing prefetched code in read/line buffers, if data and code are located in different Flash modules. Thus, the second PMU/Flash fully utilizes the pipelined operation on the LMB bus, where two addresses can be controlled concurrently; the read data are interleaved on the bus.

The required number of wait states for an initial access to PFlash or DFlash is related to the maximum operating frequency (including PLL jitter). Because the default after reset is a worst case setting sufficient for all frequencies, the access times have to be configured by the user according to the application's frequency for optimum performance. This configuration of wait states must be performed via the 4-bit-fields "WSPFLASH" and "WSDFLASH" in register FCON (Flash Configuration Register, see **Page 5-58**) according to the following table.

Note: If the initial access (either instruction or data access) addresses a double-word in PFlash which is already available in the 256-bit Flash read buffer (either because of earlier initial access or because of automatic prefetching), the defined number of wait states (in FCON register) is disabled and the access is performed without wait state, thus with 0 ns access time ("buffer hit"). In case of prefetch line hit the number of access cycles is reduced to 1 wait state, if the prefetched read-data line is already pending before the read buffer.

Table 5-1 Selection of Wait States in Relation to Operating Frequency for Flash modules with Ta=26 ns

Operating Frequency[1)]	Cycle Time	WS for Initial Access	WS for Read Buffer Hit Access	WS for Prefetch Line Hit
150 MHz up to 180 MHz	Min. 5.55 ns	5	0	min. 1
112 MHz up to 150 MHz	Min. 6.67 ns	4	0	min. 1
75 MHz up to 112 MHz	Min. 8.93 ns	3	0	min. 1
37.5 MHz up to 75 MHz	Min. 13.33 ns	2	0	min. 1
Up to 37.5 MHz	Min. 26.67 ns	1	0	min. 1

1) The maximum operating frequency of a device is documented in its data sheet.

In the table above, also the number of wait states for accesses with read buffer hit and with prefetch line hit (both only in PFlash available) are shown, which are both identical for all frequencies.

The wait state control bit in FCON for additional wait state for the ECC cycle during PFlash and DFlash accesses need not be changed by the user.

Note: The number of wait states may be changed in register FCON at any time, as long as the new number considers the maximum frequency as described in table above; this change can also be performed with code executed from Program Flash.

Mixed Accesses to PFlash and DFlash

Parallel read accesses to the Program Flash and to the Data Flash are not supported. The accesses are serialized on LMB bus and thus also on the Flash array interface. Therefore, a burst transfer from PFlash cannot be interrupted by a DFlash cycle. If a DFlash request is received during prefetching the PFlash, the prefetch access is aborted and the DFlash access is immediately started. Prefetched data or code is only inhibited and prefetching PFlash is aborted in case of a new request (data or code) to PFlash.

5.6.3 Functional Description

In the following chapters, the detailed Flash functions and the related user interface are described.

In devices with a second Flash module (controlled by a second PMU) also a second address range is defined for the other Program Flash. Additionally, a second set of all Flash registers is available in the second Flash module, providing separate status and configuration registers and separate protection control. Since the second Flash module has its own LMB bus interface it can also be controlled in parallel to all operations of the first PMU. If in the following a differentiation of the functions or addresses in the two PMUs shall be shown, the function is separated for or prefixed by PMU0 or PMU1.

5.6.3.1 Address Mapping

The total address range of 4 Gbyte (addresses A31–A0) is divided into 16 segments of each 256 Mbyte, which are addressed by A31–A28. The Program Flash as well as the Data Flash are located in segment A_H for non-cached accesses, and in segment 8_H for cached accesses. Thus the segment address bits are:

- A31–A28 = 8_H for all cached Flash accesses, and
- A31–A28 = A_H for all non-cached Flash accesses.

Note: Data accesses to Overlay Memory shall only be performed in the non-cached address space to bypass the cache/line buffer in the DMI module (necessary for data consistency after write).

Note: Command sequence cycles to the Flash shall be mapped always to the non-cached address space of the Flash which shall be operated on.

The following table **Table 5-2** presents a summary of address mappings for the Program Flash and Data Flash, and of related Flash register address mappings in PMU0. In devices with different amount of Flash memory the start addresses of the PFLASH bank and of each DFLASH bank remain constant to allow binary software compatibility between devices of the AudoF family.

Table 5-2 Flash Memory Map and Access Control in PMU0

Range Description	Size	Segment	Start Address	Access Control	Transaction Control
Program Flash cached space	2 Mbyte	8_H	$8000\ 0000_H$	Instr. Access via LMB	4x64-bit into PMI cache, with prefetch
				Data Access via LMB	2x64-bit into DMI cache, with prefetch
Data Flash cached space	64 Kbyte			Data Access via LMB	2x64-bit into DMI cache, no burst, no prefetch
DFlash bank 0	32 KB		$8FE0\ 0000_H$		
DFlash bank 1	32 KB		$8FE1\ 0000_H$		
Program Flash non-cached	2 Mbyte	A_H	$A000\ 0000_H$	Instr. Access via LMB	4x64-bit into PMI Line Buffer, with prefetch
				Data Access via LMB	1x64-bit; DMI line buffer bypassed, with prefetch
Data Flash non-cached	64 Kbyte			Data Access via LMB	1x64-bit; DMI line buffer bypassed, no prefetch
DFlash bank 0	32 KB		$AFE0\ 0000_H$		
DFlash bank 1	32 KB		$AFE1\ 0000_H$		
Flash Registers	1 Kbyte	F_H	$F800\ 2000_H$	Data Access via LMB	1x64-bit, buffer bypass

Table 5-3 contains the address mapping of the second PMU (PMU1). Its base address is different in AudoFuture and AudoMax devices. In devices with only one PMU this range is reserved. In devices with a different amount of flash memory the start address of the PFLASH bank remains constant for software compatibility.

Table 5-3 Flash Memory Map and Access Control in PMU1

Range Description	Size	Segment	Start Address	Access Control	Transaction Control
Program Flash cached space	2 Mbyte	**8_H**	$8020\ 0000_H$	Instr. Access via LMB	4x64-bit into PMI cache, with prefetch
				Data Access via LMB	2x64-bit into DMI cache, with prefetch
Program Flash non-cached	2 Mbyte	**A_H**	$A020\ 0000_H$	Instr. Access via LMB	4x64-bit into PMI Line Buffer, with prefetch
				Data Access via LMB	1x64-bit; DMI line buffer bypassed, with prefetch
Flash Registers	1 Kbyte	**F_H**	$F800\ 4000_H$	Data Access via LMB	1x64-bit, buffer bypass

5.6.3.2 Basic Operating Modes

Generally, the Flash module distinguishes two basic operating modes, the standard read mode and the command mode. Additionally to the read mode, the page mode can be activated. Since the Flash array is represented by three autonomous Flash banks, one bank of Program Flash and two banks of Data Flash, the operating modes belong to every Flash bank and can partly be active concurrently. Parallel write command execution (program one bank while erasing the other bank) is supported for the two Data Flash banks, but not for the Program Flash and one Data Flash bank.

Note: Register accesses are independent from the operating mode and allowed in any state.

The initial state after power-on and after reset is in all three Flash banks the standard read mode.

Every write access cycle into the memory address space of Program Flash or Data Flash is automatically interpreted as a command cycle, belonging to a command sequence. Thereby a command sequence consists of up to six command cycles with defined address and data values. After the last command of a command sequence, the device enters the command mode. Rules and guidelines for command sequences are provided in the chapter “Operation Control with Command Sequences” on **Page 5-20** and in the chapter “Operation Hints and Guidelines” on **Page 5-92**.

In general, the command mode remains active during the whole command execution, also indicated by the “busy” bits in the status register. Those command sequences which do not affect the Flash array, e.g. the Enter Page Mode command or the Clear Status command, are immediately executed and are therefore not visible through the busy status bit. The command mode is terminated by the correct execution of the command or by an error condition as indicated in the status register. An end-of-busy and/or error interrupt can be enabled to inform the CPU about the new state.

5.6.3.3 Command Sequence Definitions

Flash operations are selected and started by writing specific address and data bus cycles to the Flash module, thus performing the command sequences. All command cycles of command sequences which define Flash bank operations have to use base addresses which point to the Flash bank (in Flash0 or Flash1) to be operated on. All command addresses are aligned to word addresses (A1/A0=0). All command cycles belonging to one command sequence have to be mapped into the same non-cached Flash bank space. All command data (exception write and password data) are right-bounded 8-bit data. The command sequences are described in following table, whereby the shortcuts used in this table have following meanings.

- WD: 64-bit (or 32-bit) write data to be loaded into page assembly buffer.
- y: Flash Type y=0_H selects the Program Flash, y=D_H selects the Data Flash.
- PA: Page address; base address of PFlash/DFlash page to be programmed.

- UCPA: User configuration page address.
- SA: Sector address; base address of sector to be erased.
- UCBA: User configuration block address; base address of the 1 Kbyte UC block.
- UL: User protection level; the command user level is zero (master user) or one.
- PW: 32-bit password.

Table 5-4 Command Sequences for Flash Control

Command Sequence[1)2)]		1. Cycle	2. Cycle	3. Cycle	4. Cycle	5. Cycle	6. Cycle
Reset to Read	Address Data	.5554 ..xx**F0**					
Enter Page Mode[*)]	Address Data	.5554 ..xx**5y**					
Load Page[*)3)]	Address Data	.55F0 **WD**					
Write Page[*)4)5)]	Address Data	.5554 ..xx**AA**	.AAA8 ..xx**55**	.5554 ..xx**A0**	**PA** ..xx**AA**		
Write UC Page[*)5)]	Address Data	.5554 ..xx**AA**	.AAA8 ..xx**55**	.5554 ..xx**C0**	**UCPA** ..xx**AA**		
Erase Sector[*)5)]	Address Data	.5554 ..xx**AA**	.AAA8 ..xx**55**	.5554 ..xx**80**	.5554 ..xx**AA**	.AAA8 ..xx**55**	**SA** ..xx**30**
Erase Phys Sector[*)6)]	Address Data	.5554 ..xx**AA**	.AAA8 ..xx**55**	.5554 ..xx**80**	.5554 ..xx**AA**	.AAA8 ..xx**55**	**SA** ..xx**40**
Erase UC Block[*)5)]	Address Data	.5554 ..xx**AA**	.AAA8 ..xx**55**	.5554 ..xx**80**	.5554 ..xx**AA**	.AAA8 ..xx**55**	**UCBA** ..xx**C0**
Disable Write Protection[7)]	Address Data	.5554 ..xx**AA**	.AAA8 ..xx**55**	.553C **UL**	.AAA8 **PW 0**	.AAA8 **PW 1**	.5558 ..xx**05**
Disable Read Protection[7)]	Address Data	.5554 ..xx**AA**	.AAA8 ..xx**55**	.553C ..xx**00**	.AAA8 **PW 0**	.AAA8 **PW 1**	.5558 ..xx**08**
Resume Protection	Address Data	.5554 ..xx**5E**					
Clear Status	Address Data	.5554 ..xx**F5**					

1) Addresses are byte addresses but are aligned to word addresses (A1/A0=0).
Address bits A20–A16 are not used for most command addresses; however, all command and data addresses to the Flash have to be mapped to the non-cached Flash memory space of Flash0 or Flash1 by using proper base addresses according to mapping of Program Flash or Data Flash memory bank. (DFlash banks are selected with A16).
Within a command sequence all base addresses have to be identical.
*): These command sequences have to use non-cached base addresses pointing to the Flash bank to be operated on.

2) Addresses and data within the table are written in Hex format. Data are shown as right-bounded bytes in 32-bit words; however, their position on the 64-bit data bus is defined by address bit A2.

3) The address “55F0$_H$” is used for load DW (64-bit) operations and for load word (32-bit) operations with word transfer on even half of 64-bit bus. In case of word transfers, for every second word the address has to be “55F4$_H$” because the word is transferred on the high half of the 64-bit data bus (A2=1).
In case of completely filled assembly buffer, overrun data are lost and an error flag and interrupt is generated.
4) The assembly buffer should be filled with Load Page commands before this command is executed. The page address is used from the 4. cycle.
5) This command sequence is only accepted, if read protection and/or write protection for the addressed sector is disabled or not installed (user specific).
6) Phys Sector = Physical Sector. Used to erase a physical 64K Program Flash Sector, if none of its 16K logical sectors is protected. Also used to erase DFlash sectors. This command sequence is only accepted, if read protection or write protection for the addressed sector (user specific) is disabled or not installed
7) Two password cycles are included for unlock operations (PW0 = low order 32-bit of password).

5.6.3.4 Functional Command Description

In the following, the commands are described in their standard functionality and with all the particularities, which have to be considered for application.

Reset to Read

The internal command state machine is reset to initial state and returns to read mode. An already started program or erase operation is not affected and will be continued.

The Reset to Read command is a single cycle command. It can be used at any point during the command sequence to reset the internal state machines and to return the device to the initial state, the read mode. With the Reset to Read command, also all error flags in the Flash Status Register FSR are cleared and an active page mode is aborted. A busy state of array (write operation or voltage ramp-up) is not aborted and the busy flags in FSR are not affected[1)].

Enter Page Mode

The page mode is entered and the pointer for the first write data into the page assembly buffer is cleared to point to the first word location. This command is a single cycle command. Its base address (A31–A16) has to point to the addressed Flash bank. The addressed Flash type (Program Flash or Data Flash) is selected with the parameter “y”. Simultaneous page modes in Program Flash and Data Flash are not supported. In a DFLASH bank the page mode can be entered simultaneously to an erase operation within another DFLASH bank, but not simultaneously to a program operation[2)].

1) The exact behavior of Reset to Read in case of busy Flash banks is: when a busy Flash bank is addressed the writing master is stalled (as for all commands). When Reset to Read addresses a different Flash bank it is executed as described (clearing the documented flags) but in order to note that a command sequence was executed while a Flash bank was busy the SQER flag is set.
2) Attention: trying to program a DFLASH bank while a program operation is still ongoing in the other DFLASH bank is not prevented by hardware and it does not trigger a SQER or other error indication. However both program operations can be disturbed and program erroneous data!

With this command, the ‘page mode’ is entered, indicating that the page assembly buffer is enabled to be filled up with Load Page commands, and that a program operation is in preparation. Selection between assembly buffers of Program Flash and Data Flash is performed with the nibble “y” in the low byte of data word: $y=0_H$ selects the assembly buffer (256 byte) of Program Flash, $y=D_H$ selects the assembly buffer (128 byte) of Data Flash. The page mode is indicated in the status register FSR with the PAGE bit, separately for PFlash and DFlash. The page mode and the read mode are allowed in parallel at the same time and in the same Flash memory. The page mode can be aborted and the related PAGE bit in FSR be cleared with the Reset to Read command. A new Enter Page Mode command during page mode aborts the actual page mode, which is indicated with error flag ‘Sequence Error’, and enters the new page mode as described above. The assembly buffer remains unchanged (not cleared) with a new Enter Page Mode command.

Load Page

Load page assembly buffer with 64-bit double-word or with 32-bit word. The first or a sequential or the last double-word (or word) is loaded into the page assembly buffer with each Load Page command.

Note: For loading one page, the transfer width must always be identical. Mixed transfers of 64-bit and 32-bit write data are not supported by the PMU, because the ECC generation would be disabled in case of single 32-bit transfers (SQER error reporting in case of mixed transfers).

This (sequential) data write access to the assembly buffer belongs to and is only accepted in Page Mode. The data width (64-bit or 32-bit) is selected by the data-source (normally the DMI-unit) and it is indicated to the Flash by a qualifier to the data address. For 64-bit transfers, the address (A15–A0) of this single cycle command is always the same ($55F0_H$), in case of 32-bit transfers, the address must alternate between even or odd ($55F0_H$ or $55F4_H$) word addresses. The low order address bits also identify the Load Page command and the sequential write data to be loaded into the assembly buffer. The page assembly buffer to be accessed, either the 256-byte assembly buffer of the Program Flash or the 128-byte assembly buffer of the Data Flash, is defined by the base address of the command cycle. The location within the page assembly buffer is defined by the internal address pointer (see command Enter Page Mode), which is incremented after every load operation.

The page assembly buffer is a two-stage buffer, with the first stage (in Flash Interface Module) representing a 2-word data latch for ECC generation, and a second stage, representing the page assembly buffer, being filled with the double-words. After every received double-word, an 8-bit error correction code ECC is automatically generated and the double-word plus ECC code is loaded into the page assembly buffer. Not loaded double-words in the assembly buffer are undefined (not cleared). Load Page operations shall not be started before a preceding Write Page operation is terminated.

In case of a completely filled page assembly buffer, an overrun condition is sampled during Load Page operations. In this case, the write data causing the overflow condition are lost. The overflow condition is indicated by the sequence error flag and by an error interrupt (if enabled), but the execution of a following Write Page command is not suppressed (the page mode is not aborted). When a Load Page command is received and the Flash is **not** in page mode, a sequence error is reported in FSR with SQER flag. A sequence error is also indicated if during page mode a new Enter Page command is received. In case of a Reset to Read command during page mode, or in case of a system reset, the page mode is aborted (but the write data in the page assembly buffer are not cleared).

Note: A sequence error generates an error interrupt if enabled in the configuration register FCON.

Write Page

The contents of the selected page assembly buffer (the whole page) is written (programmed) into the Flash array at page address PA. The addressed Flash type has to be the same as in the Enter Page Mode command (otherwise, a sequence error is reported instead of execution).

The complete

- 256-byte Program Flash assembly buffer and the 32 ECC-bytes, or the
- 128-byte Data Flash assembly buffer and the 16 ECC-bytes,

are programmed to Flash in one program operation, autonomously controlled by the Flash array module.

The bits

- A20 to A8 of PA (address of 4. cycle) define the page address which is the location of the page within the Program Flash
- A14 to A7 of PA (address of 4. cycle) define the page address and thus the location of the page within that Data Flash bank, which is addressed with A16.

The page address has to be aligned, therefore the bits A7–A0 (Data Flash: A6 to A0) of PA have to be zero. The data pattern of the 4. cycle is fixed (AA_H) and only used for confirmation of the page address.

If the page assembly buffer is not completely filled when receiving the command, a sequence error is reported, however the command is executed. If the 2-word data latch for ECC generation (in FIM) is not completely filled, the word is lost and again a sequence error is reported.

With the Write Page command, the page mode is terminated, indicated by resetting the related PAGE flag and — if the command is accepted — the command mode is entered and the PROG flag in the status register FSR is activated. The Write Page command is only executed (PROG is only set), if the addressed Flash bank is not busy; otherwise,

the first command cycle is acknowledged with a retry request. After start of program operation also the BUSY flag for the addressed bank is set in FSR.

The start of program operation can be delayed:

- If initiated for one DFlash bank while the other DFlash bank is in erase operation (which will be automatically suspended and resumed). The delay depends on the state of the erase operation and the erase timing. Typically the delay is less or equal to 15 ms.
- Until an active Data Flash erase operation is terminated, if the program operation is initiated for the Program Flash.

Read accesses to Data Flash banks during busy state of Program Flash are not allowed. Read accesses to one Data Flash bank while the other DFlash bank is busy with a program operation is especially supported. A read access to a busy Flash bank is serviced with a retry acknowledge until the addressed Flash bank is not busy anymore. A new Write Page command is only accepted after a new Enter Page Mode command.

Note: Multiple writes to the same page location before erase are not allowed; in this case stored data in adjacent cells (in the second page of the same wordline) may be disturbed. However, there is one exception for a second write to the same page in Data Flash and in Program Flash: A page can once be overwritten with all-ones data (and correct all-ones ECC) for example to create an invalidation stamp for indication of an invalid wordline.

During the program operation, the quality of the programmed bits is autonomously verified by the Flash FSI. Weak bits are re-programmed. A verification error (re-programming is not possible anymore) is indicated with the VER bit in FSR (see also **Page 5-23**).

If write protection is installed for the sector to be programmed or in case of active read protection (including global write protection, if not separately disabled for the Data Flash), the Write Page command is only executed when read/write protection has been disabled before, using a ‘Disable Protection’ command sequence with two 32-bit passwords. If the sector-specific and/or global write protection is not disabled when the Write Page command is received or if an OTP protected sector is addressed, only the page mode is terminated, but command mode and thus the program operation is not started, and the protection error flag PROER is set in the FSR.

Write User Configuration (UC) Page

This command is identical to the Write Page command for the Program Flash with the following exceptions: The addressed page (UCPA) belongs to one User Configuration Block (UCB) in the configuration sector and not to the user Flash array. This command can only be executed on the user’s four pages in his UCB. Only if protection is not installed (e.g. for the very first installation of protection), it is not necessary to disable the user’s protection before execution of this command. After writing the correct protection

confirmation code in the respective UC page, the new protection configuration is valid and active after the next reset.

If the protection configuration in an UC block has to be re-programmed (not possible for UCB2), at first the user's protection must be disabled and then its UC block must be erased. Thus also the UCB's protection confirmation code is erased and the protection is uninstalled. Only in this unprotected state an UC page can be (re-)programmed. A Write UC Page command to a User Configuration Block with active (not disabled) write protection disables the command and generates a protection error (PROER in FSR). In case of installed and active read protection, all three UC Blocks are locked and not accessible (PROER is indicated in FSR).

The structure of the User Configuration Block (comprising four User Configuration Pages UCP0–UCP3 per block) is described in **Chapter 5.6.5.4**.

Note: If the program operation is aborted (e.g. by a reset), written UCB-data may be disturbed and the protection can possibly no more be disabled.

Erase Sector

The addressed sector in the Program Flash is erased. This command cannot be used for Data Flash sectors. It is a six-cycle command sequence.

The sector to be erased is addressed by SA (sector address) in the last command cycle. Sectors can be erased (and re-programmed) as often as defined by the endurance value (max. 1000, for special handling of logical 16K sectors see **Chapter 5.6.2.1**). The timing of erase execution is autonomously controlled by the Flash array module. The max. erase time is five sec., but its exact timing depends on the sector size and the CPU frequency.

The address bits A20 to A14 of SA (address of 6. cycle) define the sector address and thus the sector to be erased. The sector address has to be aligned, therefore the bits A13 to A0 of SA have to be zero. The addresses of command sequence cycles including the sector address have to be mapped to the base address of the non-cached Program Flash (sequence error, if mapped to DFlash).

With the last cycle of the Erase Sector command, the command mode is entered, indicated by activation of the ERASE and by the PBUSY flag in the status register FSR. The start of an erase operation can be delayed until an active erase and/or program operation(s) is terminated. Allowed is only to issue an erase of a DFLASH bank while the other DFLASH bank is busy.

Read accesses to Data Flash banks during busy state of Program Flash are not allowed. Read accesses to one Data Flash bank while the other DFlash bank is busy with erase operation is especially supported. A read access to the busy Flash bank is serviced with a retry acknowledge until the addressed Flash bank is no more busy. A new command sequence to the busy Flash bank is also serviced with a retry acknowledge.

The sector erase algorithm includes an erase quality check that identifies incorrect erased bits in the Flash sector. If re-erasing of weak bits or soft re-programming of over-erased bits is unsuccessful, the verify error flag FSR.VER is set (see **Chapter 5.6.6.3**).

If write protection is installed for the sector to be erased or in case of active read protection (including global write protection, if not separately disabled for Data Flash), the command is only executed when read/write protection has been disabled before, using a ‘Disable Protection’ command sequence with two 32-bit passwords. If write protection is not disabled when the Sector Erase command is received or if an OTP protected sector is addressed, the command mode and thus the erase operation is not started and the protection error flag PROER is set in the FSR.

Erase Physical Sector

The addressed physical sector in the Data Flash or in Program Flash is erased. Cannot be used for 16K sectors in Program Flash. It is a six-cycle command sequence.

The sectors, which can be erased with this command, are the two Data Flash sectors (banks) DS0 and DS1 and the two 64K physical sectors PS0 and PS4 in Program Flash which comprise the logical 16K sectors (see **Table 5-6**). This command shall not be used for the large sectors above S7. Besides the different sector address definitions, the command execution is analogous to the Erase Sector command (including erase quality check).

The sector addresses (see **Page 5-44**) have to be aligned, therefore all low order address bits of SA are zero. All command cycle addresses, including the sector address SA, have to be mapped into that space (PFlash or DFlash range) where the sector to be erased is located.

The start of sector erase operation can be delayed by up to 5 msec if initiated while another bank (DFlash or PFlash) is active with program operation.

Read accesses to Data Flash banks during busy state of Program Flash are not allowed. Read accesses to one Data Flash bank while the other DFlash bank is busy with erase operation is especially supported. A read access to the busy Flash bank is serviced with a retry acknowledge until the addressed Flash bank is no more busy. A new command sequence to the busy Flash bank is also serviced with a retry acknowledge. The busy state of Flash bank is indicated in the FSR separately for PFlash and for bank 0 and bank 1 of DFlash.

If read protection (includes global write protection with or without Data Flash) is installed, the command is only executed when read protection has before been disabled using the unlock command sequence ‘Disable Read Protection’ with two passwords, belonging to user 0. If read protection is not disabled when the Erase Phys Sector command is received, the command mode and thus the erase operation is not started, and the protection error flag PROER is set in the FSR.

If the Erase Phys Sector operation is used to erase a physical 64K sector of Program Flash (including the 16K sectors), this operation is only executed, if none of its 16K sectors is write protected or if protection is disabled (user 0 and/or user 1). If write protection is not disabled, or if one or more of the included 16K sectors are OTP protected, the erase operation is not started, and the protection error flag PROER is set.

The Erase Phys Sector operation in one DFlash bank can be executed simultaneously to program operation in the other Data Flash bank or/and to read accesses to Program Flash.

Erase User Configuration Block

The addressed user configuration block (UCB 0–2) is erased. A UC block has the capacity of four pages (1 Kbyte).

Note: If sector write protection is configured and confirmed (thus installed) in UCB 2, this block can never be erased again. As its write protected sectors, UCB 2 is only one time programmable (OTP). UCB 2 can only be erased before installation of OTP write protection.

The addressed UC block belongs to the configuration sector and not to the universal Flash array. This command can only be executed after disabling of sector write protection of the addressed UCB (representing the user's protection level), and/or after disabling of read protection. If protection is not disabled when the Erase UC Block command is received, the command mode and thus the erase operation is not started, and the protection error flag PROER is set in the FSR.

The UC block to be erased is addressed by UCBA (UCB address) in the last command cycle to the Program Flash. The bits A20 to A10 of UCBA (address of 6. cycle) define the UC block address and thus the two wordlines (four pages) of array to be erased. The UCB address has to be aligned, therefore the bits A9 to A0 of UCBA have to be zero. The addresses of all cycles of the command sequence including the UC block address have to be mapped to the base address of Program Flash.

The command execution is analogous to an Erase Sector operation in Program Flash. With the last cycle of the Erase UC Block command, the command mode is entered, indicated by activation of the ERASE and the PBUSY flag in the status register FSR. Read accesses to Data Flash banks during busy state of Program Flash are not allowed. A read access to the busy Flash is serviced with a retry acknowledge until the addressed Flash bank is no more busy.

Note: If the erase operation is aborted (e.g. by a reset), UCB-data and the whole config sector may be disturbed and the protection can possibly no more be disabled. In this case, also the init code may be affected, resulting in a fatal startup error.

After the erase operation, the new protection configuration (including keywords and protection confirmation code) can be written to the UC pages (see description in **Chapter 5.6.5**).

Disable Sector Write Protection

Sector write protection of all protected sectors belonging to the user's protection level (only in Program Flash) is temporarily disabled.

This is a protected command sequence, using two user defined passwords to release this command. For every password one command cycle is required (see command sequence definitions). The 32-bit passwords are internally compared with the keywords out of the User Configuration Block, which is assigned to the user's protection level. If one or both passwords are not identical to their related keywords, the protected sectors remain in the locked state and a protection error (PROER) is indicated in the Flash Status Register. In this case, a new Disable Sector Write Protection command or a Disable Read Protection command is only accepted after the next application reset.

Note: The Disable Sector Write Protection command is not accepted for user 2 sectors, which are OTP protected and handled as ROM sectors.

After correct execution of this command, all protected sectors (which belong to the user) are unlocked, which is indicated in the Flash Status Register (FSR) with the user's WPRODIS bit. Erase and write operations on the unlocked sectors are now possible, if not coincidently the read protection (including global write protection) is enabled. The protection is resumed, when:

- The command Resume Read/Write Protection is executed
- The next application reset (including HW and SW reset) is received.

Note: This command sequence is also used to check the correctness of keywords before the protection is locked with the confirmation code in the User Configuration Block. A wrong keyword is indicated by the FSR flag PROER.

Disable Read Protection

Flash read protection and global write protection of the whole Flash array including the Data Flash (if also protected) are temporarily disabled.

This is a protected command sequence, using two user defined passwords to release this command. For every password one command cycle is required (see command sequence definitions). The both 32-bit passwords are internally compared with the keywords out of the User Configuration Wordline, which is assigned to the user protection level zero. If one or both passwords are not identical to their related keywords, the protected array remains in the locked state and a protection error (PROER) is indicated in the Flash status register. In this case, a new Disable Read Protection command or a Disable Sector Write Protection command is only accepted after the next application reset.

After correct execution of this command, the whole Flash is unlocked and the read protection disable bit RPRODIS is set in the Flash Status Register (FSR). Erase and write operations on all sectors are now possible, if they are not separately write protected or OTP protected. Additionally, Flash read accesses from any master are

now supported in the PMU. The read protection control flags DCF and DDF in FCON register can now be cleared via FCON register access. The read protection (including global write protection) remains disabled until the command Resume Read/Write Protection is executed, or until the next application reset (including HW and SW reset).

Note: This command sequence is also used to check the correctness of keywords before the protection is locked with the confirmation code in the User Configuration Block zero. A wrong keyword is indicated by the FSR flag PROER.

Resume Protection

Write protection of temporarily unlocked protected sectors and/or read protection is resumed.

This single-cycle command resumes all kinds of temporarily disabled protection installations, as defined in the User Configuration Pages belonging to the UC-blocks. This command is released immediately after execution.

Clear Status

The error flags in Flash status register are cleared. Additionally, the write status bits (PROG, ERASE) are cleared, if Flash is not busy.

The single-cycle command ‘Clear Status’ is accepted only in Read Mode (otherwise ‘Sequence Error’)[1]. Note: Read Mode is also taken after every error detection and indication in FSR.

1) The exact behavior of Clear Status in case of busy Flash banks is: as all commands it stalls the issuing master when addressing a busy Flash bank. When addressing a not busy Flash bank Clear Status is not executed and SQER is set additionally.

5.6.3.5 Sector, Page and Block Addressing

As all command cycles of command sequences, sector, page and block addressing as required in the command sequences shall be performed in the non-cached address space of Program Flash and Data Flash (for definition of address mapping see **Table 5-2**). In the following, the base addresses of Program Flash and Data Flash banks, which depend on this mapping are not considered. Only the physical addresses within the Flash array, which always start with address zero, are defined.

Sector addresses are required for the commands ‘Erase Sector’ for the Program Flash and ‘Erase Phys Sector’ for the Data and Program Flash.

Block addressing is required for the command ‘Erase User Configuration Block’.

Page addressing is necessary for the commands ‘Write Page’ and ‘Write User Configuration Page’.

Sector Addresses

Sector addresses SA are identical to the sector start addresses (represented by lowest address within the sector) which are defined by the address bits A20–A14 in the appropriate command cycles. Depending on the sector size at least the address bits A13–A0 must be zero for sector addresses which have to be aligned. The high order address bits A31–A21 are defined by the base address of the Program Flash memory and the base address of the Data Flash in case of DFlash sectors, as described in **Chapter 5.6.3.1**.

The following two tables define the sectors with their sector numbers, the sector sizes, the sector (start) addresses and the ranges of the sectors for the (standard) sectors in Program Flash and for the physical sectors in Program Flash and Data Flash. The address bits written with bold characters are fixed within the respective range.

All sectors in Program Flash can separately be locked by up to three users for write or OTP protection.

Table 5-5 Addresses and Sizes of Sectors in Program Flash

Sector	Phys. Sector	Sector Size	Sector Addresses SA									Sector Range Physical Addr. (hex)
			A21 – A19			A18– A17		A16 – A14			A13– A00	
S0	**PS0**	16 KB	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	- 0 -	00’0000 – 00’3FFF
S1		16 KB	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	- 0 -	00’4000 – 00’7FFF
S2		16 KB	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	- 0 -	00’8000 – 00’BFFF
S3		16 KB	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	- 0 -	00’C000 – 00’FFFF
S4	**PS4**	16 KB	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	- 0 -	01’0000 – 01’3FFF
S5		16 KB	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	- 0 -	01’4000 – 01’7FFF
S6		16 KB	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	- 0 -	01’8000 – 01’BFFF
S7		16 KB	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	- 0 -	01’C000 – 01’FFFF
S8	**–**	128 KB	**0**	**0**	**0**	**0**	**1**	0	0	0	- 0 -	02’0000 – 03’FFFF
S9	**–**	256 KB	**0**	**0**	**0**	**1**	0	0	0	0	- 0 -	04’0000 – 07’FFFF
S10	**–**	256 KB	**0**	**0**	**1**	**0**	0	0	0	0	- 0 -	08’0000 – 0B’FFFF
S11	**–**	256 KB	**0**	**0**	**1**	**1**	0	0	0	0	- 0 -	0C’0000 – 0F’FFFF
S12	**–**	256 KB	**0**	**1**	**0**	**0**	0	0	0	0	- 0 -	10’0000 – 13’FFFF
S13	**–**	256 KB	**0**	**1**	**0**	**1**	0	0	0	0	- 0 -	14’0000 – 17’FFFF
S14	**–**	256 KB	**0**	**1**	**1**	**0**	0	0	0	0	- 0 -	18’0000 – 1B’FFFF
S15	**–**	256 KB	**0**	**1**	**1**	**1**	0	0	0	0	- 0 -	1C’0000 – 1F’FFFF

For compatibility between devices with different amount of PFLASH all sectors maintain their address and number throughout the family. PMUs with less PFLASH have fewer of the 256 Kbyte sectors:

- 0.5 Mbyte PFLASH: sectors S0 – S9 are implemented.
- 1 Mbyte PFLASH: sectors S0 – S11 are implemented.
- 1.5 Mbyte PFLASH: sectors S0 – S13 are implemented.
- 2 Mbyte PFLASH: sectors S0 – S15 are implemented.
- 2.5 Mbyte PFLASH: sectors S0 – S17 are implemented.

*Note: All addresses defined in the sector table above are internal addresses of and within the Program Flash memory. However, all sector addresses to the Flash have to be assigned to the respective Flash memory base address by using proper **A31–A21** address bits according to the mapping of Program Flash memory.*

Both Data Flash sectors (DS0 and DS1) and the two PFlash sectors (PS0 and PS4), as used in the command sequence ‘Erase Phys Sector’ (see **Table 5-4**) are addressed according to the standard sector addressing, with their start addresses (see table **Table 5-6**). The high order address bits A31 – A21 are again defined by the address mapping of Data Flash or Program Flash.

Table 5-6 Addresses of Phys Sectors in DFlash and in PFlash

Sector Number	Sector Size	Sector Addresses SA			Sector Range Physical Addr. (hex)
		A20 – A17	A16	A15 – A00	
DS0	32 KB	**- 0 -**	**0**	-0-	0’0000 – 0’7FFF
DS1	32 KB	**- 0 -**	**1**	-0-	1’0000 – 1’7FFF
PS0	64 KB	**- 0 -**	**0**	-0-	00’0000 – 00’FFFF
PS4	64 KB	**- 0 -**	**1**	-0-	01’0000 – 01’FFFF

As noted before in all DFLASH implementation throughout the family the start addresses of both DFLASH banks/sectors remains constant.

Page Addresses

Pages in Program Flash (256 byte pages) are addressed via the address bits A20–A8, pages of Data Flash bank (128 byte pages) are addressed with the address bits A14 to A7 during the last command cycle of a ‘Write Page’ command sequence. The low order address bits A7–A0 (Data Flash A6–A0) have to be zero, thus the page address PA must be aligned. As for sector addresses, the high order address bits A31–***A21*** for Program Flash and A31–A16 for Data Flash are defined by the base address of the accessed Flash bank and depend on the Flash mapping (see **Table 5-2** for PMU0 and **Table 5-3** for PMU1). The addresses of User Configuration Pages (UCP) are handled as the PFlash page addresses PA11 – PA0.

Table 5-7 Addresses and Sizes of Pages

Page Number	Page Size	Page Addresses PA							Page Range Physical Addr. (hex)
		A20–A17	A16	A15–A12	A11–A9	A8	A7	A6–A0	
Program Flash									
Pneven	256 Byte	**-X-**	**X**	**-X-**	**-X-**	**0**	- 0 -		xx’xx00 – xx’xxFF
Pnodd	256 Byte	**-X-**	**X**	**-X-**	**-X-**	**1**	- 0 -		xx’xx00 – xx’xxFF

Table 5-7 Addresses and Sizes of Pages (cont'd)

Page Number	Page Size	Page Addresses PA							Page Range Physical Addr. (hex)
		A20–A17	A16	A15–A12	A11–A9	A8	A7	A6–A0	
Data Flash									
DPneven	128 Byte	Base=0	**b**[1]	**XXXX**	**-X-**	**X**	**0**	- 0 -	x'xx00 – x'xx7F
DPnodd	128 Byte	Base=0	**b**[1]	**XXXX**	**-X-**	**X**	**1**	- 0 -	x'xx80 – x'xxFF
User Configuration Block									
UCPneven	256 Byte	**- 0-**	**0**	**- 0-**	**-X-**	**0**	- 0 -		00'0x00 – 00'0xFF
UCPnodd	256 Byte	**- 0-**	**0**	**- 0-**	**-X-**	**1**	- 0 -		00'0x00 – 00'0xFF

1) "b" selects the Data Flash bank. b=0: DFlash bank 0, b=1: DFlash bank 1

In table above, '**X**' indicate relevant address bits or relevant hex-numbers of physical (Flash-internal) addresses. The useful range of addresses depends obviously on the amount of implemented flash memory, i.e. the size of the banks.

User Configuration Block Addresses

Three User Configuration Blocks (UCB) are available in the configuration sector in Program Flash. Each UC block has the capacity of 1 KB (four pages).

Table 5-8 Addresses and Sizes of User Configuration Blocks

UCB Number	UCB Size	UCB Addresses				UCB Range Physical Addr. (hex)
		A20–A15	A14–A12	A11/A10	A9–A0	
UCB0	1 Kbyte	**- 0-**	**- 0-**	**00**	- 0 -	00'0000 – 00'03FF
UCB1	1 Kbyte	**- 0-**	**- 0-**	**01**	- 0 -	00'0400 – 00'07FF
UCB2	1 Kbyte	**- 0-**	**- 0-**	**10**	- 0 -	00'0800 – 00'0BFF

5.6.3.6 Register Addresses and Access Restrictions

Register accesses are supported for CPU controlled accesses via the DMI unit and for accesses via the debug interface. All register addresses are word aligned, independently of the register width. Besides word-read/write accesses, also byte or half-word read/write accesses are supported.

All Flash register read and write accesses are single cycle operations.

For Flash registers separate address spaces are reserved for each PMU (see **Table 5-9**). The detailed register list of PMU0 is contained in **Table 5-2** and if available for PMU1 in **Table 5-3**.

Table 5-9 Registers Address Spaces of Flash Registers

Module	Base Address	End Address	Note
FLASH0	F800 1000_H	F800 $23FF_H$	See **Table 5-11**
FLASH1	F800 3000_H	F800 $43FF_H$	See **Table 5-12**

Within the register address table below, the Access Modes “Read” and “Write” indicate access rights and restrictions as well as values, using symbols as follows:

Table 5-10 Address Map Symbols

Symbol	Description
U	Access permitted in User Mode 0 or 1.
SV	Access permitted in Supervisor Mode.
E	ENDINIT protected register/address.
BE	Indicates that an access to this address range generates a Bus Error.
X	Undefined value or bit.

Note: In the table above, the declaration of access protection with ENDINIT indicates the standard “ENDINIT” protection feature (including watchdog password access control) which is used for write accesses to protected system registers. Using this mechanism, it is possible to change the protected register only before End of Initialization or after ENDINIT with valid password access to WDT_CON0. Read accesses are always possible.

The following table shows the addresses, the access modes and reset types for the Flash registers in PMU0:

Table 5-11 Addresses of Flash0 Registers

Short Name	Description	Address	Access Mode		Reset	See
			Read	Write		
–	Reserved	F800 2000_H – F800 2004_H	BE	BE	–	–
FLASH0_ ID	Flash Module Identification Register	F800 2008_H	U, SV	BE	Class3 Reset	**Page 5-64**
–	Reserved	F800 $200C_H$	BE	BE	–	–

Table 5-11 Addresses of Flash0 Registers

Short Name	Description	Address	Access Mode		Reset	See
			Read	Write		
FLASH0_FSR	Flash Status Register	F800 2010$_H$	U, SV	BE	Class3+ PORST Reset	Page 5-51
FLASH0_FCON	Flash Configuration Register	F800 2014$_H$	U, SV	SV, E	Class3 Reset	Page 5-58
FLASH0_MARP	Flash Margin Control Register PFlash	F800 2018$_H$	U, SV	SV, E	Class3 Reset	Page 5-68
FLASH0_MARD	Flash Margin Control Register DFlash	F800 201C$_H$	U, SV	U, SV	Class3 Reset	Page 5-69
FLASH0_PROCON0	Flash Protection Configuration User 0	F800 2020$_H$	U, SV	BE	Class3 Reset	Page 5-76
FLASH0_PROCON1	Flash Protection Configuration User 1	F800 2024$_H$	U, SV	BE	Class3 Reset	Page 5-78
FLASH0_PROCON2	Flash Protection Configuration User 2	F800 2028$_H$	U, SV	BE	Class3 Reset	Page 5-80
–	Reserved	F800 202C$_H$ – F800 23FC$_H$	BE	BE	–	

The following table shows the addresses, the access modes and reset types for the Flash registers in PMU1:

Table 5-12 Addresses of Flash1 Registers

Short Name	Description	Address	Access Mode		Reset	See
			Read	Write		
–	Reserved	F800 4000$_H$ – F800 4004$_H$	BE	BE	–	–
FLASH1_ID	Flash Module Identification Register	F800 4008$_H$	U, SV	BE	Class3 Reset	Page 5-65
–	Reserved	F800 400C$_H$	BE	BE	–	–
FLASH1_FSR	Flash Status Register	F800 4010$_H$	U, SV	BE	Class3, PORST Reset	Page 5-51
FLASH1_FCON	Flash Configuration Register	F800 4014$_H$	U, SV	SV, E	Class3 Reset	Page 5-58

Table 5-12 Addresses of Flash1 Registers

Short Name	Description	Address	Access Mode		Reset	See
			Read	Write		
FLASH1_MARP	Flash Margin Control Register PFlash	F800 4018$_H$	U, SV	SV, E	Class3 Reset	Page 5-68
FLASH1_MARD	Flash Margin Control Register DFlash	F800 401C$_H$	U, SV	U, SV	Class3 Reset	Page 5-69
FLASH1_PROCON0	Flash Protection Configuration User 0	F800 4020$_H$	U, SV	BE	Class3 Reset	Page 5-76
FLASH1_PROCON1	Flash Protection Configuration User 1	F800 4024$_H$	U, SV	BE	Class3 Reset	Page 5-78
FLASH1_PROCON2	Flash Protection Configuration User 2	F800 4028$_H$	U, SV	BE	Class3 Reset	Page 5-80

5.6.3.7 Flash Status Definition

The Flash Status Register FSR reflects the overall status of the Flash module after Reset and after reception of the different commands. Sector specific protection states are not indicated in the FSR, but in the registers PROCON0, PROCON1 and PROCON2. The status register is a read-only register. Only the error flags and the two status flags (PROG, ERASE) are affected with the “Clear Status” command. The error flags are also cleared with the “Reset to Read” command.

The FSR is defined as follows:

FSR
Flash Status Register **(1010_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
VER	**X**	**0**	**SLM**	**0**	**WPRODIS1**	**WPRODIS0**	**0**	**WPROIN2**	**WPROIN1**	**WPROIN0**	**0**	**RPRODIS**	**RPROIN**	**0**	**PROIN**
rh	rh	r	rh	r	rh	rh	r	rh	rh	rh	r	rh	rh	r	rh
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DFDBER	**PFDBER**	**DFSBER**	**PFSBER**	**PROER**	**SQER**	**DFOPER**	**PFOPER**	**DFPAGE**	**PFPAGE**	**ERASE**	**PROG**	**D1BUSY**	**D0BUSY**	**FABUSY**	**PBUSY**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
PBUSY[1)]	0	rh	**Program Flash Busy** HW-controlled status flag. 0_B PFlash ready, not busy; PFlash in read mode. 1_B PFlash busy; PFlash not in read mode. Indication of busy state of PFlash because of active execution of program or erase operation; PFlash busy state is also indicated during Flash recovery (after reset) and in power ramp-up state or in sleep mode; while in busy state, the PFlash is not in read mode.
FABUSY[1)]	1	rh	**Flash Array Busy** Internal busy flag for testing purposes. Must be ignored by application software. This may only use PBUSY, D0BUSY and D1BUSY.

Field	Bits	Type	Description
D0BUSY[1)]	2	rh	**Data Flash Bank 0 Busy** HW-controlled status flag. 0_B DFlash0 ready, not busy; DFlash0 in read mode. 1_B DFlash0 busy; DFlash0 not in read mode. Indication of busy state of DFlash bank 0 because of active execution of program or erase operation; DFlash0 busy state is also indicated during Flash recovery (after reset) and in power ramp-up state or in sleep mode; while in busy state the DFlash0 is not in read mode.
D1BUSY[1)]	3	rh	**Data Flash Bank 1 Busy** HW-controlled status flag. 0_B DFlash1 ready, not busy; DFlash1 in read mode. 1_B DFlash1 busy; DFlash1 not in read mode. Indication of busy state of DFlash bank 1 because of active execution of program or erase operation; DFlash1 busy state is also indicated during Flash recovery (after reset) and in power ramp-up state or in sleep mode; while in busy state the DFlash0 is not in read mode.
PROG[3)4)]	4	rh	**Programming State** HW-controlled status flag. 0_B There is no program operation requested or in progress or just finished. 1_B Programming operation (write page) requested (from FIM) or in action or finished. Set with last cycle of Write Page command sequence, cleared with Clear Status command (if not busy) or with power-on reset. If one BUSY flag is coincidently set, PROG indicates the type of busy state. If xOPER is coincidently set, PROG indicates the type of erroneous operation. Otherwise, PROG indicates, that operation is still requested or finished.

Field	Bits	Type	Description
ERASE[3)4)]	5	rh	**Erase State** HW-controlled status flag. 0_B There is no erase operation requested or in progress or just finished 1_B Erase operation requested (from FIM) or in action or finished. Set with last cycle of Erase command sequence, cleared with Clear Status command (if not busy) or with power-on reset. Indications are analogous to PROG flag.
PFPAGE[1)2)]	6	rh	**Program Flash in Page Mode** HW-controlled status flag. 0_B Program Flash not in page mode 1_B Program Flash in page mode; assembly buffer of PFlash (256 byte) is in use (being filled up) Set with Enter Page Mode for PFlash, cleared with Write Page command *Note: Concurrent page and read modes are allowed*
DFPAGE[1)2)]	7	rh	**Data Flash in Page Mode** HW-controlled status flag. 0_B Data Flash not in page mode 1_B Data Flash in page mode; assembly buffer of DFlash (128 byte) is in use (being filled up) Set with Enter Page Mode for DFlash, cleared with Write Page command. *Note: Concurrent page and read modes are allowed*
PFOPER[2)3)4)]	8	rh	**Program Flash Operation Error** 0_B No operation error reported by Program Flash 1_B Flash array operation aborted, because of a Flash array failure, e.g. an ECC error in microcode. This bit is not cleared with application reset, but with power-on reset. Registered status bit; must be cleared per command
DFOPER[2)3)4)]	9	rh	**Data Flash Operation Error** Function analogous to Program Flash OPER

Field	Bits	Type	Description
SQER[1)2)3)]	10	rh	**Command Sequence Error** 0_B No sequence error 1_B Command state machine operation unsuccessful because of improper address or command sequence. A sequence error is not indicated if the Reset to Read command aborts a command sequence. Registered status bit; must be cleared per command
PROER[1)2)3)]	11	rh	**Protection Error** 0_B No protection error 1_B Protection error. A Protection Error is reported e.g. because of a not allowed command, for example an Erase or Write Page command addressing a locked sector, or because of wrong password(s) in a protected command sequence such as “Disable Read Protection” Registered status bit; must be cleared per command
PFSBER[1)2)3)]	12	rh	**PFlash Single-Bit Error and Correction** 0_B No Single-Bit Error detected during read access to PFlash 1_B Single-Bit Error detected and corrected Registered status bit; must be cleared per command
DFSBER[1)2)3)]	13	rh	**DFlash Single-Bit Error and Correction** Function analogous to PFlash PFSBER
PFDBER[1)2)3)]	14	rh	**PFlash Double-Bit Error** 0_B No Double-Bit Error detected during read access to PFlash 1_B Double-Bit Error detected in PFlash Registered status bit; must be cleared per command
DFDBER[1)2)3)]	15	rh	**DFlash Double-Bit Error** Function analogous to PFlash PFDBER
PROIN	16	rh	**Protection Installed** 0_B No protection is installed 1_B Read or/and write protection for one or more users is configured and correctly confirmed in the User Configuration Block(s). HW-controlled status flag

Field	Bits	Type	Description
RPROIN	18	rh	**Read Protection Installed** 0_B No read protection installed 1_B Read protection and global write protection (with or without Data Flash) is configured and correctly confirmed in the User Configuration Block 0. Supported only for the master user (user zero). HW-controlled status flag
RPRODIS[1)5)]	19	rh	**Read Protection Disable State** 0_B Read protection (if installed) is not disabled 1_B Read and global write protection is temporarily disabled. Flash read with instructions from other memory, as well as program or erase on not separately write protected sectors is possible. HW-controlled status flag
WPROIN0	21	rh	**Sector Write Protection Installed for User 0** 0_B No write protection installed for user 0 1_B Sector write protection for user 0 is configured and correctly confirmed in the User Configuration Block 0. HW-controlled status flag
WPROIN1	22	rh	**Sector Write Protection Installed for User 1** 0_B No write protection installed for user 1 1_B Sector write protection for user 1 is configured and correctly confirmed in the User Configuration Block 1. HW-controlled status flag
WPROIN2	23	rh	**Sector OTP Protection Installed for User 2** 0_B No OTP write protection installed for user 2 1_B Sector OTP write protection with ROM functionality is configured and correctly confirmed in the UCB2. The protection is locked for ever. HW-controlled status flag

Field	Bits	Type	Description
WPRODIS0[1)5)]	25	rh	**Sector Write Protection Disabled for User 0** 0_B All protected sectors of user 0 are locked if write protection is installed 1_B All write-protected sectors of user 0 are temporarily unlocked, if not coincidently locked by user 2 or via read protection. Hierarchical protection control: User-0 sectors are also unlocked, if coincidently protected by user 1. But not vice versa. HW-controlled status flag
WPRODIS1[1)5)]	26	rh	**Sector Write Protection Disabled for User 1** 0_B All protected sectors of user 1 are locked if write protection is installed 1_B All write-protected sectors of user 1 are temporarily unlocked, if not coincidently locked by user 0 or user 2 or via read protection. HW-controlled status flag
SLM[1)]	28	rh	**Flash Sleep Mode** HW-controlled status flag. Indication of Flash sleep mode taken because of global or individual sleep request; additionally indicates when the Flash is in shut down mode. 0_B Flash not in sleep mode 1_B Flash is in sleep or shut down mode
X	30	rh	**Reserved** Value undefined
VER[1)3)]	31	rh	**Verify Error** 0_B The page is correctly programmed or the sector correctly erased. All programmed or erased bits have full expected quality. 1_B A program verify error or an erase verify error has been detected. Full quality (retention time) of all programmed (“1”) or erased (“0”) bits cannot be guaranteed. See **Chapter 5.6.6.3** and **Chapter 5.6.6.4** for proper reaction. Registered status bit; must be cleared per command

Field	Bits	Type	Description
0	17,20, 24, 27, 29	r	**Reserved** Read zero, no write

Note: The footnote numbers of FSR bits describe the specific reset conditions:
1)Cleared with application reset (class 3 reset)
2)Cleared with command “Reset to Read”
3)Cleared with command “Clear Status”
4)Cleared with power-on reset (PORST)
5)Cleared with command “Resume Protection”

Note: The xBUSY flags as well as the protection flags cannot be cleared with the “Clear Status” command or with the “Reset to Read” command. These flags are controlled by HW.

*Note: The reset value above is indicated after correct execution of Flash rampup. Additionally, errors are possible after rampup (see **Chapter 5.6.6.4**).*

Note: Since in PMU1 a Data Flash is not available, the DFlash bits are not relevant in PMU1_FSR.

5.6.3.8 Flash Configuration Control

The Flash Configuration Register FCON reflects and controls the following general Flash configuration functions:

- Number of wait states for Flash accesses (see also **Table 5-1** for selection).
- Indication of installed and active read protection.
- Instruction and data access control for read protection.
- Interrupt mask bits.
- Power reduction and shut down control.

FCON is a “ENDINIT” protected register. It is defined as follows:

FCON
Flash Configuration Register (1014_H) Reset value: $0007\ 0606_H$ [1)]

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EOBM	**DFDBERM**	**PFDBERM**	**DFSBERM**	**PFSBERM**	**PROERM**	**SQERM**	**VOPERM**	**0**	**0**	**DDFPCP**	**DDFDMA**	**0**	**DDF**	**DCF**	**RPA**
rw	rw	rw	rw	rw	rw	rw	rw	r	r	rw	rw	r	rwh	rwh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SLEEP	**ESLDIS**	**IDLE**	**WSECDF**	**WSDFLASH**				**0**	**0**	**0**	**WSECPF**	**WSPFLASH**			
rw	rw	rw	rw	rw				r	r	r	rw	rw			

1) After Flash rampup and execution of the startup SW in BootROM (after firmware exit), the initial value is $000X\ 0606_H$.

Field	Bits	Type	Description
WSPFLASH	[3:0]	rw	**Wait States for read access to PFlash** This bitfield defines the number of wait states, which are used for an initial read access to the Program Flash memory area (see **Table 5-1** for selection). 0000_B PFlash access in one clock cycle 0001_B PFlash access in one clock cycle 0010_B PFlash access in two clock cycles 0011_B PFlash access in three clock cycles 0100_B PFlash access in four clock cycles 0101_B PFlash access in five clock cycles 0110_B PFlash access in six clock cycles = default 0111_B PFlash access in seven clock cycles. .... PFlash access in eight up to fourteen clock cycles. 1111_B PFlash access in fifteen clock cycles.
WSECPF	4	rw	**Wait State for Error Correction of PFlash** 0_B No additional wait state for error correction 1_B One additional wait state for error correction during read access to Program Flash. If enabled, this wait state is only used for the first transfer of a burst transfer. Set this bit only when requested by Infineon.
WSDFLASH	[11:8]	rw	**Wait States for read access to DFlash** This bitfield defines the number of wait states, which are used for a read access to the Data Flash memory area (see **Table 5-1** for selection). 0000_B DFlash access in one clock cycle 0001_B DFlash access in one clock cycle 0010_B DFlash access in two clock cycles 0011_B DFlash access in three clock cycles 0100_B DFlash access in four clock cycles 0101_B DFlash access in five clock cycles 0110_B DFlash access in six clock cycles = default 0111_B DFlash access in seven clock cycles. 1000_B DFlash access in eight clock cycles. .... DFlash access in nine up to fourteen clock cycles. 1111_B DFlash access in fifteen clock cycles.

Field	Bits	Type	Description
WSECDF	12	rw	**Wait State for Error Correction of DFlash** 0_B No additional wait state for error correction 1_B One additional wait state for error correction during read access to Data Flash
IDLE	13	rw	**Dynamic Flash Idle** 0_B Normal/standard Flash read operation 1_B Dynamic idle of Program Flash enabled for power saving; static prefetching disabled *Note: In Data Flash, dynamic idle is always enabled (prefetching not supported).*
ESLDIS	14	rw	**External Sleep Request Disable** 0_B External sleep request signal input is enabled 1_B Externally requested Flash sleep is disabled The ‘external’ signal input is connected with a global power-down/sleep request signal from SCU.
SLEEP	15	rw	**Flash SLEEP** 0_B Normal state or wake-up 1_B Flash sleep mode is requested, Wake-up from sleep is started with clearing of the SLEEP-bit.
RPA	16	rh	**Read Protection Activated** This bit monitors the status of the Flash-internal read protection. This bit can only be ‘0’ when read protection is not installed or while the read protection is temporarily disabled with password sequence. 0_B The Flash-internal read protection is not activated. Bits DCF, DDF are not taken into account. Bits DCF, DDFx can be cleared 1_B The Flash-internal read protection is activated. Bits DCF, DDF are enabled and evaluated.

Field	Bits	Type	Description
DCF	17	rwh	**Disable Code Fetch from Flash Memory** This bit enables/disables the code fetch from the internal Flash memory area. Once set, this bit can only be cleared when RPA=’0’. This bit is automatically set with reset and is cleared during rampup, if no RP installed, and during startup (BootROM SW) in case of internal start out of Flash. 0_B Code fetching from the Flash memory area is allowed. 1_B Code fetching from the Flash memory area is not allowed. This bit is not taken into account while RPA=’0’.
DDF	18	rwh	**Disable Any Data Fetch from Flash** This bit enables/disables the data read access to the Flash memory area (Program Flash and Data Flash). Once set, this bit can only be cleared when RPA=’0’. This bit is automatically set with reset and is cleared during rampup, if no RP installed, and during startup (BootROM SW) in case of internal start out of Flash. 0_B Data read access to the Flash memory area is allowed. 1_B Data read access to the Flash memory area is not allowed. This bit is not taken into account while RPA=’0’.
DDFDMA	20	rw	**Disable Data Fetch from DMA Controller** This bit enables/disables the data read access to PFlash&DFlash memory from the DMA controller and other bus masters that access the LMB bus via the DMA peripheral interfaces — these are, dependent on the device: Cerberus, MLI and Memcheck. Once set, this bit can only be cleared when RPA=’0’. 0_B The data read access by the DMA controller and its peripheral interfaces to the Flash memory area is allowed. 1_B The data read access to the Flash memory area is not allowed for the DMA controller and its peripheral interfaces.

Field	Bits	Type	Description
DDFPCP	21	rw	**Disable Data Fetch from PCP Controller** This bit enables/disables the data read access to PFlash&DFlash memory via the LFI bridge (used from PCP controller). Once set, this bit can only be cleared when RPA='0'. 0_B The data read access by the PCP controller to the Flash memory area is allowed. 1_B The data read access to the Flash memory area is not allowed for the PCP controller.
VOPERM	24	rw	**Verify and Operation Error Interrupt Mask** 0_B Interrupt not enabled 1_B Flash interrupt because of Verify Error or Operation Error in Flash array (FSI) is enabled
SQERM	25	rw	**Command Sequence Error Interrupt Mask** 0_B Interrupt not enabled 1_B Flash interrupt because of Sequence Error is enabled
PROERM	26	rw	**Protection Error Interrupt Mask** 0_B Interrupt not enabled 1_B Flash interrupt because of Protection Error is enabled
PFSBERM	27	rw	**PFlash Single-Bit Error Interrupt Mask** 0_B No Single-Bit Error interrupt enabled 1_B Single-Bit Error interrupt enabled for PFlash
DFSBERM	28	rw	**DFlash Single-Bit Error Interrupt Mask** 0_B No Single-Bit Error interrupt enabled 1_B Single-Bit Error interrupt enabled for DFlash
PFDBERM	29	rw	**PFlash Double-Bit Error Interrupt Mask** 0_B Double-Bit Error interrupt for PFlash not enabled 1_B Double-Bit Error interrupt for PFlash enabled. Especially intended for margin check
DFDBERM	30	rw	**DFlash Double-Bit Error Interrupt Mask** 0_B Double-Bit Error interrupt for DFlash not enabled 1_B Double-Bit Error interrupt for DFlash enabled. Especially intended for margin check

Field	Bits	Type	Description
EOBM	31	rw	**End of Busy Interrupt Mask** 0_B Interrupt not enabled 1_B EOB interrupt is enabled
0	[7:5], 19, [23:22]	r	**Reserved** Read/write zero

Note: The default numbers of wait states represent the slow cases. This is a general proceeding and additionally opens the possibility to execute higher frequencies without changing the configuration.

Note: After reset and execution of BootROM startup SW, the read protection control bits are coded as follows:
DDF, DCF, RPA = “110”: No read protection installed
DDF, DCF, RPA = “001”: Read protection installed; start in internal Flash
DDF, DCF, RPA = “111”: Read protection installed; start not in internal Flash.

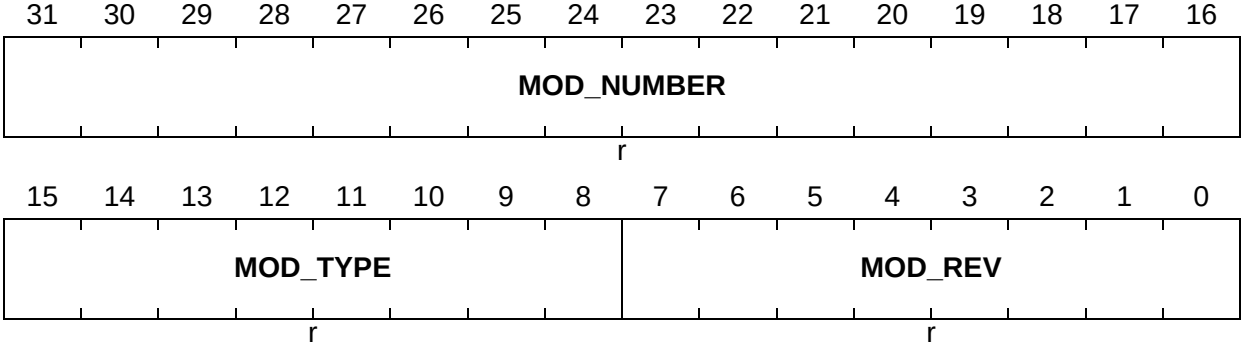
Note: Since in PMU1 a Data Flash is not available, the DFlash bits are not relevant in PMU1_FCON.

5.6.3.9 Flash Identification Register

The module identification register of Flash module is directly accessible by the CPU via PMU access. This register is mapped into the space of the Flash Interface Module's registers (see **Table 5-11**).

FLASH0_ID

Flash Module Identification Register (1008_H) **Reset Value: 0053 C0XX$_H$**

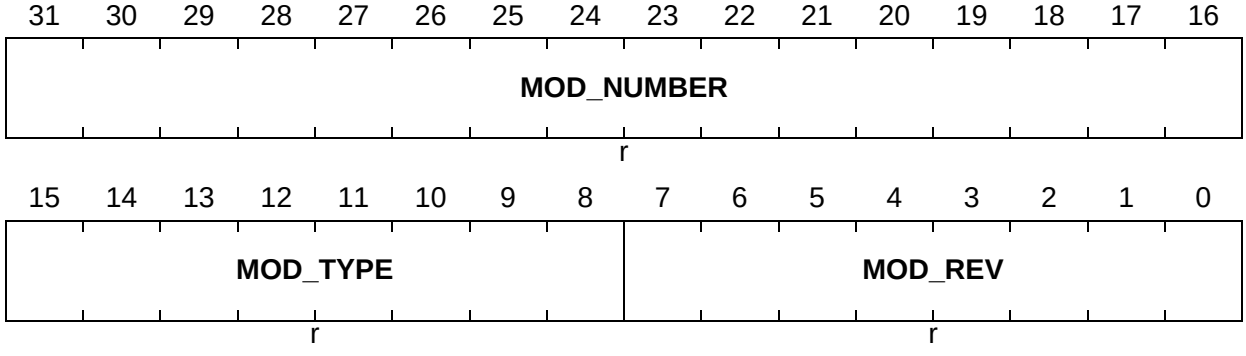


Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision Number** MOD_REV defines the module revision number. The value of a module revision starts with 01_H (first revision).
MOD_TYPE	[15:8]	r	**Module Type** This bit field is $C0_H$. It defines the module as a 32-bit module.
MOD_NUMBER	[31:16]	r	**Module Number Value** This bit field defines a module identification number. For the TC1797 Flash0 this number is 0053_H.

The second Flash array (Flash1) is not identical to the first array, because it does not include the Data Flash. It therefore has another ID number than the Flash0 array:

FLASH1_ID

Flash Module Identification Register (1008_H) **Reset Value: $0055\ C0XX_H$**



Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision Number** MOD_REV defines the module revision number. The value of a module revision starts with 01_H (first revision).
MOD_TYPE	[15:8]	r	**Module Type** This bit field is $C0_H$. It defines the module as a 32-bit module.
MOD_NUMBER	[31:16]	r	**Module Number Value** This bit field defines a module identification number. For the TC1797 Flash1 this number is 0055_H.

5.6.4 Error Correction and Margin Control

Error detection and correction is provided for all read accesses to Program Flash and Data Flash. The combination of error detection with the also available margin check provides an excellent verify function for Flash data safety.

5.6.4.1 Dynamic Error Correction

The Flash module supports the following error detection and correction functions for read accesses to both, the Program Flash as well as the Data Flash:

- Detection of single-bit errors within 64-bit read data and correction on the fly
- Detection of double-bit errors
- Two read error flags in the Flash Status Register FSR, indicating a single-bit error:
  - Flag PFSBER indicates, that one (or more) single-bit error was detected and corrected in the Program Flash
  - Flag DFSBER indicates, that one (or more) single-bit error was detected and corrected in the Data Flash
- Two read error flags in the Flash Status Register FSR, indicating a double-bit error:
  - Flag PFDBER indicates, that one (or more) double-bit error was detected in the Program Flash
  - Flag DFDBER indicates, that one (or more) double-bit error was detected in the Data Flash.
- An error interrupt is generated in case of any single-bit error, if enabled in the FCON register.
- An error interrupt is generated in case of any double-bit error, if enabled in the FCON register. This interrupt shall only be used for margin check, when trap is disabled
- A bus error trap is reported in case of a double-bit error during access to Program Flash or Data Flash, as soon as the disturbed instruction or data is transferred to the PMI or DMI unit via the LMB bus. This trap can be disabled for margin checks.

Note: A single-bit or a double-bit error may also be caused by a disturbed EC-code with correct 64-bit data, or by a wrong selection of access time with wait states.

Error detection and correction is controlled using a SEC-DED algorithm that results in an 8-bit error correction code (ECC) for every 64-bit data in PFlash and in DFlash. This 8-bit ECC is dynamically generated during write operations to the assembly buffer and then programmed to the Flash array with the Write Page command. For every read access to the Flash the 64-bit read data is fetched together with its associated 8-bit ECC.

The ECC algorithm is selected in such a way, that all zero data have an all zero ECC and all one data have an all one ECC. It is thus supported, that erased locations (all zero) have a correct EC-code, and it is also supported to re-program a Data Flash page to all ones to indicate a special page, e.g. with an invalidation stamp. This over-program operation also results in a correct EC-code (all ones).

After an erase operation, a correct ECC code (all zero) is provided for the erased sector in the Program Flash as well as in the Data Flash.

For details about handling ECC errors and other flags see **Chapter 5.6.6.3**.

5.6.4.2 Margin Check Control

Margin control is supported for the sense amplifiers separately of the Program Flash and the Data Flash. With the two margin control registers MARP (for Program Flash) and MARD (for Data Flash) the sensing of array bit lines can be controlled more critical for ‘0’ or ‘1’ values during read operations. The Margin Control Registers MARP and MARD are used to change the margin levels for read operations to find problematic array bits. Since problematic bits always change their value from ‘1’ to ‘0’, it is quite simple to find those bits: The array area to be checked is read with changed margins which are more critical for sensing ‘1’ values (verify operation). A single or double-bit error will be reported to the CPU by an error interrupt or a bus error trap. The double-bit error trap can be disabled for margin checks and also redirected to an error interrupt.

The different margin levels are enabled and selected with the Margin Control Registers MARP and MARD. The high margin levels which can be selected, are one low level margin (coded with ‘0’) and one high level margin (coded with ‘1’).

Note: Only one margin change (high or low level, PFlash or DFlash) is allowed at a time.

*Note: To increase the security and to inhibit unintended write accesses, the standard “ENDINIT” protection feature (including watchdog password access control) is used for write accesses to the margin control register MARP (for Program Flash). The MARD register for Data Flash is **not** especially protected.*

Note: After change of margin level, a wait time of >10µsec. for sense amp adjustment is necessary before read operations with the modified margins shall be executed.

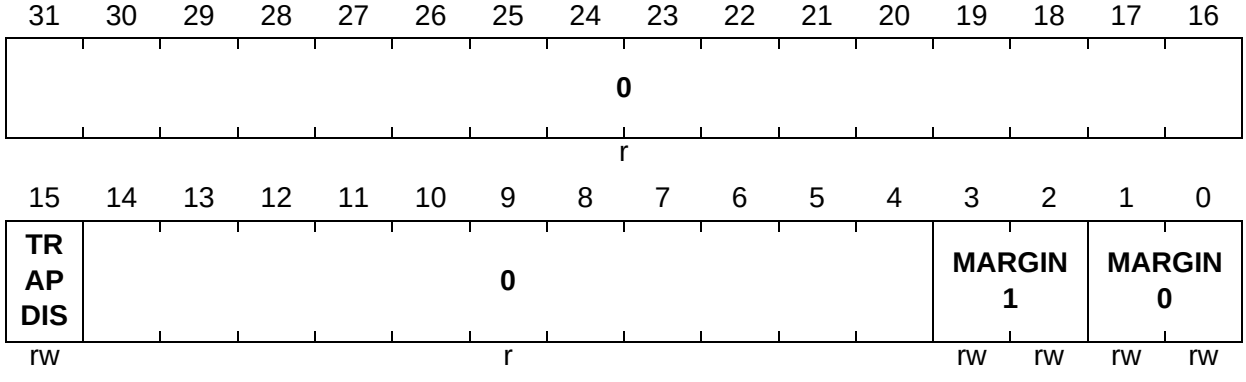
Note: During erase or program operation only the standard (default) margins are allowed (no margin change for parallel read accesses).

Note: Although double-bit error traps are disabled with reset, the traps are enabled by the startup SW (firmware) in Boot ROM before Boot ROM exit.

The Margin Control Registers for Program Flash (MARP) and for Data Flash (MARD) are defined as follows:

MARP

Margin Control Register PFLASH (1018$_H$) Reset Value: 0000 8000$_H$

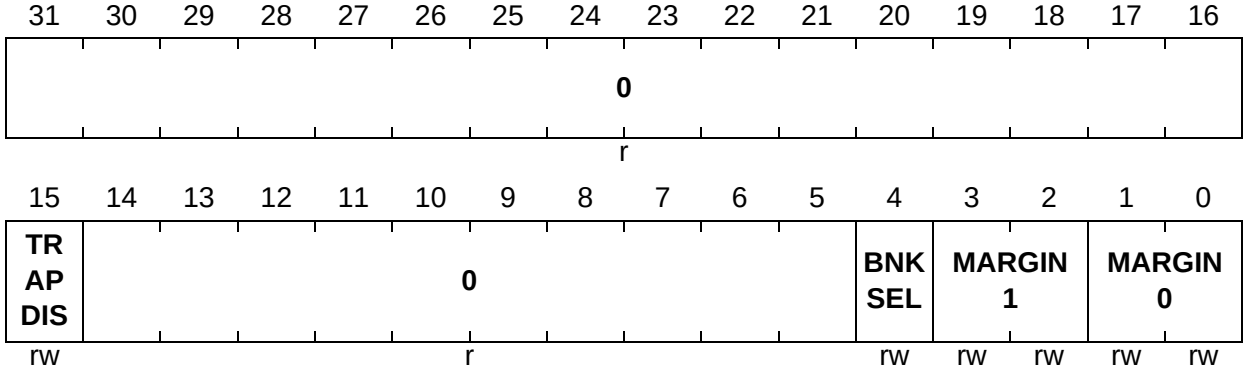


Field	Bits	Type	Description
MARGIN0	[1:0]	rw	**PFLASH Margin Selection for Low Level** 00_B Standard (default) margin 01_B High margin for 0 (low) level 10_B Reserved 11_B Reserved
MARGIN1	[3:2]	rw	**PFLASH Margin Selection for High Level** 00_B Standard (default) margin 01_B High margin for 1 (high) level 10_B Reserved 11_B Reserved
TRAPDIS	15	rw	**PFLASH Double-Bit Error Trap Disable** 0_B If a double-bit error occurs in PFLASH, a bus error trap is generated[1]. 1_B The double-bit error trap is disabled. Shall be used only during margin check
0	[14:4], [31:16]	r	**Reserved;** always read as 0; should be written with 0.

1) After Boot ROM exit, double-bit error traps are enabled (TRAPDIS = 0).

MARD

Margin Control Register DFLASH (101C$_H$) **Reset Value: 0000 8000$_H$**



Field	Bits	Type	Description
MARGIN0	[1:0]	rw	**DFLASH Margin Selection for Low Level** 00_B Standard (default) margin 01_B High margin for 0 (low) level 10_B Reserved 11_B Reserved
MARGIN1	[3:2]	rw	**DFLASH Margin Selection for High Level** 00_B Standard (default) margin 01_B High margin for 1 (high) level 10_B Reserved 11_B Reserved
BNKSEL	4	rw	**Enable DFLASH Margin Control** 0_B The active read margin for both DFLASH banks is determined by MARGIN0 and MARGIN1. 1_B Both DFLASH banks are read with standard (default) margin independent of MARGIN0 and MARGIN1.
TRAPDIS	15	rw	**DFLASH Double-Bit Error Trap Disable** 0_B If a double-bit error occurs in DFLASH, a bus error trap is generated[1]. 1_B The double-bit error trap is disabled. Shall be used only during margin check
0	[14:5], [31:16]	r	**Reserved;** always read as 0; should be written with 0.

1) After Boot ROM exit, double-bit error traps are enabled (TRAPDIS = 0).

Note: Since in PMU1 a Data Flash is not available, the MARD bits have no effect in PMU1.

5.6.5 Read and Write Protection

For an overview please refer to **Chapter 5.6.2.3**

In general, three user levels are supported for installation of protection configuration, and three different types of protection can be assigned to the user levels as follows:

1. User 0: This is the master user. He is able to install read protection for the whole Flash (with or without DFlash). Additionally or alternatively, he can install sector specific write protection. User 0 controls the User Configuration Block UCB0 in the configuration sector and defines his keywords in UCB0.
2. User 1 is able to install sector specific write protection, with lower priority than user 0. User 1 controls the UCB1 in config sector and defines his keywords in UCB1.
3. User 2 is able to install sector specific OTP protection with ROM functionality. Sectors with ROM functionality are locked for ever and are never re-programmable. Keywords for temporary disabling the protection are therefore not necessary. User 2 (who might be identical to user 0 or user 1) controls the UCB2.

Any installation of protection will become active only after next reset.

5.6.5.1 Read Protection

When read protection is installed and active, read accesses to the Flash memory are disabled and suppressed, if the program execution does not start in internal Flash after reset. Thus Flash read accesses by instructions fetched from other memory but internal Flash are initially blocked. The read protection is characterized by the following definitions (identical for Flash0 and Flash1 if not otherwise stated):

- The read protection is installed, if the read protection configuration is programmed (thus the read protection is configured) and confirmed in the User Configuration Block of user 0 (UCB0); the installed read protection is indicated in the FSR register by RPROIN-bit and in PROCON0 by RPRO-bit. If read protection is configured but not confirmed, only RPRO is set after next reset.
- The Data Flash is only excluded from read protection, if the bit DFEXPRO is set additionally to the RPRO-bit in register PROCON0.
- The read protection is active, when it is installed and not temporarily disabled with password protected command sequence ‘Disable Read Protection’; this state is shown with RPA-bit in configuration register FCON.
- If read protection is active (RPA=1), also a global write protection for the whole Flash array is activated. The Data Flash is only excluded from the global write protection, if the bit DFEXPRO is set additionally to the RPRO-bit in register PROCON0. Flash erase/program commands are generally disabled and not executed. But accesses to registers and command cycles (write cycles) to the state machine are enabled (necessary for the disable command sequence).
- If read protection is active, the Flash read accesses are controlled with the disable bits DCF (disable code fetch) and DDF (disable data fetch) in FCON register. After reset, these bits are set if the user program start is not executed from internal Flash

(e.g. start from external memory or from internal scratchpad memory after bootstrap execution).

- If read protection is active and a bootstrap loader is selected by reset configuration, the execution of bootstrap loader is not suppressed because the Flash is fully protected with DCF and DDF (see above).
- If read protection is active in one or both Flash modules and user program is started in internal Flash, the debug interfaces are totally disabled after reset (controlled by BootROM startup).
- If read protection is active in Flash0, only one re-configuration of the instruction cache in PMI unit is possible after reset. The user shall install his SPR/cache configuration quickly after reset to lock the configuration and thus the protection; then, data read accesses to the instruction cache are disabled (because this requires a change of configuration). Subsequent changes of SPR/cache configuration are only possible, if the read protection is temporarily disabled with correct passwords.
- If read protection is active in one or both Flash modules, Flash read accesses are generally **not** disabled in case of internal start after reset out of the Program Flash (DCF and DDF are cleared before start of program execution, controlled by BootROM startup). In this case the user SW in Flash has to handle access restrictions to the Flash by controlling the Disable Flash Fetch bits in the FCON register.

  Examples ('Flash' stands for Flash0 or Flash1):
  - Before jumping to external memory or internal RAM, bit DDF is set by user SW in Flash. In this case, all Flash data accesses are disabled but return to Program Flash instruction execution is possible (because DCF is not set).
  - Before jumping to external memory or internal RAM, bit DDF is set by the user. DCF is also set by the user directly after jumping to internal or external RAM. Now, Flash data accesses and return to code fetch from Program Flash are disabled; return is only possible if the read protection is temporarily disabled (not active) using the password protected disable command sequence. In this case, bits DCF and DDF must be cleared by the user before the read protection is resumed, because otherwise accesses are blocked after resumption.
- If read protection is active or not, Flash data accesses from dedicated bus masters others than the CPU/DMI can be separately disabled with the FCON bits DDFDMA and DDFPCP (bus cycle sources are indicated by tags). These bits cannot be cleared while read protection is active.
- A disabled but installed read protection is indicated by RPRODIS=1 in FSR register.

Note: If read protection is active and DCF and DDF are set, not allowed data or instruction read accesses to Flash result in a LMB bus error (trap) indication. This has also to be considered for jump predictions. The user has to make sure that the prediction does not point into the protected Flash.

Installation of read protection is performed with the "Write User Configuration Page" operation, controlled by the user 0 (separately for Flash0 and Flash1). With this

command, user 0 writes the protection configuration bits RPRO and eventually DFEXPRO, and the two 32-bit keywords into the UCB0-page 0. Additionally, with a second “Write User Configuration Page” command, a special 32-bit confirmation (lock-) code is written into the UCB0-page2. Only this confirmation code enables the protection and thus the keywords. The confirmation write operation to the second wordline of the User Configuration Block shall be executed only after check of keyword-correctness (with command “Disable Read Protection” after next reset). The confirmed state and thus the installation of protection is indicated with the FSR-bit PROIN in Flash Status Register FSR and for read protection with bit RPROIN in FSR. If read protection is not correctly confirmed and thus not enabled, the bits PROIN and RPROIN in the FSR are not set. The configured read protection as fetched from UCB0 is indicated in the protection configuration register PROCON0[1)].

For safety of the information stored in the UCB pages, all keywords, lock bits and the confirmation code are stored two-times in the two wordlines. In case of a disturbed original data detected during rampup, its copy is used. Layout of the four UC pages belonging to the user’s UC block is shown in **Chapter 5.6.5.4** below, the command “Write User Configuration Page” is described in **Chapter 5.6.3.4**.

With the command sequence “Disable Read Protection” a short-term disablement of read protection is provided (separately for Flash0 or Flash1). With this command, it is possible to disable the Flash protection (latest until next reset) for user controlled erase and re-program operations as well as for clearing of DCF and DDF control bits after external program execution. The “Disable Read Protection” command sequence is a protected command, which is only processed by the command state machine, if the included two passwords are identical to the two keywords of user 0. The disabled state of read protection is controlled with the FCON-bit RPA=’0’ and indicated in the Flash Status Register FSR with the RPRODIS bit (see **Chapter 5.6.3.7**). As long as read protection is disabled (and thus not active) in the concerned Flash module, the FCON-bits DDF and DCF of this module can be cleared.

Resumption of read protection after disablement is performed with the “Resume Read/Write Protection” command. After execution of this single cycle command, read protection (if installed) is again active, indicated by the FCON bit RPA=’1’.

Generally, Flash read protection will remain installed as long as it is confirmed (locked) in the User Configuration Block 0. Erase of UC block and re-program of UC pages can be performed up to 4 times. But note, after execution of the Erase UC block command (which is protected and therefore requires the preceding disable command with the user’s specific passwords), all keywords and all protection installations of user 0 are erased; thus, the Flash is no more read protected (beginning with next reset) until re-programming the UC pages. But the division and separation of the protection configuration data and of the confirmation data into two different UCB-wordlines

1) PROCON0.DFEXPRO is only set when in UCB0 the bits for RPRO and DFEXPRO are set to 1.

guarantees, that a disturb of keywords can be discovered and corrected before the protection is confirmed. For this reason, the command sequence “Disable Read Protection” can also be used when protection is programmed (configured) but not confirmed; wrong keywords are then indicated by the error flag PROER.

Read protection can be combined with sector specific write protection. In this case, after execution of the command ‘Disable Read Protection’ only those sectors are unlocked for write accesses, which are not separately write protected.

5.6.5.2 Write and OTP Protection

If sector write protection is installed and active, erasing and programming of write protected sectors is not possible. An installed write protection is indicated per user by the three (because three users are supported) WPROIN bits in FSR register, and sector-specific by the configuration status bits in the Protection Configuration registers PROCON0 (for user 0), PROCON1 (for user 1) and PROCON2 (sectors of user 2 with OTP protection, representing ROM functionality).

*Note: Sectors in **Data Flash** cannot be separately write protected (only generally via Read Protection).*

Note: Beginning with sector S10, only sector-pairs of two 256 KB sectors can be defined for installation of sector-specific write or OTP protection (see PROCON registers).

Note: An installation of OTP protection (for ROM functionality) can be performed only once, because the UCB2 block is locked for ever after the installation.

Note: Full FAR test of protected Flash is only possible, if the customer has de-installed the protection before, or if the passwords are known by the test person. In case of OTP/ROM protection, the Flash test capability of FAR is very limited, because FSI-SFRs and FSI-SRAM are no more accessible.

As read protection, installation of write protection is performed with the “Write User Configuration Page” operation, controlled by the user for that Flash module, which is addressed by the base addresses of the command cycles. With this command, the user defines and writes into the UCBx page 0 the write protection configuration bits for all sectors in the addressed Program Flash module, which shall be locked by the specific user, and the user-specific two keywords (not necessary for user 2). The position of sector lock bits is identical as defined for the PROCON registers. The correctness of keywords shall then (after next reset) be checked with the command ‘Disable Sector Write Protection’, which delivers a protection error PROER in case of wrong passwords. Only if the keywords are correct, the special 32-bit confirmation code must be written into the page 2 of UCBx with a second “Write User Configuration Page” command. Only this confirmation code enables the write protection of the User Control Block UCBx, and only in this case the installation bit(s) in FSR is (are) set during rampup.

Note: If the write protection is configured in the user's UCB page 0 but not confirmed via page 2 (necessary for check of keywords), the state after next reset is as follows:
- The selected sector(s) are protected (good for testing of protection, also of OTP)
- The UCBx is not protected, thus it can be erased without passwords
- The related WPROINx bit in FSR is not set
- The Disable Write Protection command sets the WPRODISx bit
- The Resume command does not clear the WPRODISx bit.

The structure and layout of the three UC blocks is shown in **Chapter 5.6.5.4** below, the command "Write User Configuration Page" is described in **Chapter 5.6.3.4**.

With the command sequence "Disable Sector Write Protection" a short-term disablement of write protection for user 0 or user 1 is provided. This command unlocks temporarily all locked sectors belonging to the user. The "Disable Sector Write Protection" command sequence is a protected command, which is only processed by the command state machine, if the included two passwords are correct. The disabled state of sector protection is indicated in the Flash Status Register FSR with the WPRODIS bit of the user 0 or/and user 1 (see **Chapter 5.6.3.7**). For user 2 who owns the sectors with ROM functionality, a disablement of write protection and thus re-programming is not possible.

Resumption of write protection after disablement is performed with the "Resume Read/Write Protection" command, which is identical for user 0 and user 1.

Generally, sector write protection will remain installed as long as it is configured and confirmed in the User Configuration Block belonging to the user. Erase of UC block and re-program of UC pages can be performed up to 4 times, for user 0 and user 1 only. But note, after execution of the Erase UC block command (which is still protected and therefore requires the preceding disablement of write protection with the user's passwords), the complete protection configuration including the keywords of the specific user (not user 2) is erased; thus, the sectors belonging to the user are totally unprotected until the user's UC pages are re-programmed. Only exception: sectors protected by user 2 are locked for ever because the UCB2 can no more be erased after installation of write protection in UCB2.

Sector specific write protection may be combined with read protection. In this case, after execution of the command 'Disable Sector Write Protection' the protected sectors are only unlocked if read protection is also disabled.

The write protection is hierarchically controlled, thus the user 0 has the highest level of access rights with write access to all 'his' sectors (if they are not protected by user 2): When user 0 disables the write protection for 'his' sectors, the write protection is also disabled for those sectors, which are coincidently protected by user 1. But user 1 can only disable his own protected sectors, if they are not coincidently protected by user 0 (and user 2). If one sector is coincidently protected by user 0 and 1, user 1 can only write-access this sector if unlocked also with the passwords of user 0.

5.6.5.3 Protection Configuration Indication

The configuration of read/write/OTP protection is indicated with registers PROCON0, PROCON1 and PROCON2, thus separately for every user, and it is generally indicated in the status register FSR.

If write protection is installed for user 0 or 1 or OTP protection for user 2, for each sector of the Program Flash it is indicated in the user-specific Protection Configuration register PROCONx, if it is locked or unlocked for program or erase operations.

The Flash Protection Configuration registers PROCONx are loaded by the FIM state machine out of the user's configuration block directly after reset during rampup. The three PROCONx registers are read-only registers. They are defined as follows:

PROCON0
Flash Protection Configuration Register User 0
(1020_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RPRO	DFEXPRO	0	S14/S15L	S12/S13L	S10/S11L	S9L	S8L	S7L	S6L	S5L	S4L	S3L	S2L	S1L	S0L
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
SnL (n=0-9)	n	rh	**Sector n Locked for Write Protection by User 0** These bits indicate whether PFLASH sector n is write-protected by user 0 or not. 0_B No write protection is configured for sector n. 1_B Write protection is configured for sector n.

Field	Bits	Type	Description
S10/S11L	10	rh	**Sectors 10 and 11 Locked for Write Protection by User 0** This bit is only used if PFLASH has more than 0.5 Mbyte. It indicates whether PFLASH sectors 10+11 (together 512 KB) are write-protected by user 0 or not. 0_B No write protection is configured for sectors 10+11. 1_B Write protection is configured for sectors 10+11.
S12/S13L	11	rh	**Sectors 12 and 13 Locked for Write Protection by User 0** This bit is only used if PFLASH has more than 1 Mbyte. It indicates whether PFLASH sectors 12+13 (together 512 KB) are write-protected by user 0 or not. 0_B No write protection is configured for sectors 12+13. 1_B Write protection is configured for sectors 12+13.
S14/S15L	12	rh	**Sectors 14 and 15 Locked for Write Protection by User 0** This bit is only used if PFLASH has more than 1.5 Mbyte. It indicates whether PFLASH sectors 14+15 (together 512 KB) are write-protected by user 0 or not. 0_B No write protection is configured for sectors 14+15. 1_B Write protection is configured for sectors 14+15.

Field	Bits	Type	Description
DFEXPRO	14	rh	**Data Flash Excluded from Read Protection** When read protection is configured this bit indicates whether the DFLASH shall be excluded from read protection and global write protection or not. **Attention**: Even when the corresponding bit in UCB0 is programmed to 1_B this bit is only set to 1_B when RPRO is also programmed to 1_B. 0_B DFLASH not excluded from read protection and global write protection. 1_B DFLASH is excluded from read/write protection; read protection and global write protection is configured by user 0 only for the PFLASH
RPRO	15	rh	**Read Protection Configuration** This bit indicates whether read protection is configured for PFLASH and DFLASH by user 0. 0_B No read protection configured 1_B Read protection and global write protection is configured by user 0 (master user)
0	13	rh	**Reserved;** deliver the corresponding UCB0 entry. Shall be configured to 0.
0	[31:16]	r	**Reserved;** always read as 0.

PROCON1
Flash Protection Configuration Register User 1

(1024_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		**0**	**S14/ S15L**	**S12/ S13L**	**S10/ S11L**	**S9L**	**S8L**	**S7L**	**S6L**	**S5L**	**S4L**	**S3L**	**S2L**	**S1L**	**S0L**
r		rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
SnL (n=0-9)	n	rh	**Sector n Locked for Write Protection by User 1** These bits indicate whether PFLASH sector n is write-protected by user 1 or not. 0_B No write protection is configured for sector n. 1_B Write protection is configured for sector n.
S10/S11L	10	rh	**Sectors 10 and 11 Locked for Write Protection by User 1** This bit is only used if PFLASH has more than 0.5 Mbyte. It indicates whether PFLASH sectors 10+11 (together 512 KB) are write-protected by user 1 or not. 0_B No write protection is configured for sectors 10+11. 1_B Write protection is configured for sectors 10+11.
S12/S13L	11	rh	**Sectors 12 and 13 Locked for Write Protection by User 1** This bit is only used if PFLASH has more than 1 Mbyte. It indicates whether PFLASH sectors 12+13 (together 512 KB) are write-protected by user 1 or not. 0_B No write protection is configured for sectors 12+13. 1_B Write protection is configured for sectors 12+13.
S14/S15L	12	rh	**Sectors 14 and 15 Locked for Write Protection by User 1** This bit is only used if PFLASH has more than 1.5 Mbyte. It indicates whether PFLASH sectors 14+15 (together 512 KB) are write-protected by user 1 or not. 0_B No write protection is configured for sectors 14+15. 1_B Write protection is configured for sectors 14+15.
0	13	rh	**Reserved;** deliver the corresponding UCB1 entry. Shall be configured to 0.

Field	Bits	Type	Description
0	[31:16], 15, 14	r	**Reserved;** always read as 0.

PROCON2
Flash Protection Configuration Register User 2
(1028$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**S14/ S15 ROM**	**S12/ S13 ROM**	**S10/ S11 ROM**	**S9 ROM**	**S8 ROM**	**S7 ROM**	**S6 ROM**	**S5 ROM**	**S4 ROM**	**S3 ROM**	**S2 ROM**	**S1 ROM**	**S0 ROM**
r	r	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
SnROM (n=0-9)	n	rh	**Sector n Locked Forever by User 2** These bits indicate whether PFLASH sector n is an OTP protected sector with read-only functionality, thus if it is locked for ever. 0_B No ROM functionality configured for sector n. 1_B ROM functionality is configured for sector n. Re-programming of this sector is no longer possible.
S10/S11ROM	10	rh	**Sectors 10 and 11 Locked Forever by User 2** This bit is only used if PFLASH has more than 0.5 Mbyte. It indicates whether PFLASH sectors 10+11 (together 512 KB) are read-only sectors or not. 0_B No ROM functionality is configured for sectors 10+11. 1_B ROM functionality is configured for sectors 10+11.

Field	Bits	Type	Description
S12/S13ROM	11	rh	**Sectors 12 and 13 Locked Forever by User 2** This bit is only used if PFLASH has more than 1 MByte. It indicates whether PFLASH sectors 12+13 (together 512 KB) are read-only sectors or not. 0_B No ROM functionality is configured for sectors 12+13. 1_B ROM functionality is configured for sectors 12+13.
S14/S15ROM	12	rh	**Sectors 14 and 15 Locked Forever by User 2** This bit is only used if PFLASH has more than 1.5 Mbyte. It indicates whether PFLASH sectors 14+15 (together 512 KB) are read-only sectors or not. 0_B No ROM functionality is configured for sectors 14+15. 1_B ROM functionality is configured for sectors 14+15.
0	13	r	**Reserved;** deliver the corresponding UCB2 entry. Shall be configured to 0.
0	[31:16], 15, 14	r	**Reserved;** always read as 0.

5.6.5.4 User Configuration Blocks and Pages

In the User Configuration Pages, the installation of read and write protection is configured and confirmed by the user. Three UC blocks of each 1 Kbyte are provided for three different users. With the dedicated commands (see command definitions), the UC blocks can be initially programmed. The UC blocks UCB0 and UCB1 can be modified up to 4 times.

Each of the three User Configuration Blocks UCB0, UCB1 and UCB2 contains four User Configuration Pages UCP0–3; for UCP addresses see **Table 5-8**. The User Configuration Pages are ECC-protected as all other information in the Flash sectors. The User Configuration Blocks and their UC pages are organized and programmed as follows.

User Configuration Block UCB0

- The **UC Page 0** (bytes 255–0) of UCB0 includes the following information
  - Bytes 1–0: Protection configuration bits, for configuration of array read protection and for each sector to be write protected; structure of this configuration info is identical to the structure of the PROCON0 register (see **Chapter 5.6.5.3**).
  - Bytes 9–8: Copy of protection configuration bits
  - Bytes 19–16: First 32-bit keyword of user 0
  - Bytes 23–20: Second 32-bit keyword of user 0
  - Bytes 27–24: Copy of first 32-bit keyword
  - Bytes 31–28: Copy of second 32-bit keyword
  - All other page bytes: zero
- The User Configuration **Page 1** is not used (reserved for future, all zero).
- The User Configuration **Page 2** (bytes 255–0) includes the following information
  - Bytes 3–0: 32-bit confirmation code. The confirmation code (also called lock code) indicates that a protection is correctly installed and enabled. The 32-bit confirmation code is defined as follows: “$8AFE15C3_H$”.
  - Bytes 11–8: Copy of confirmation code
  - All other page bytes: zero
- The User Configuration **Page 3** is not used (reserved for future, all zero).

User Configuration Block UCB1

The structure of this UC block is identical to UCB0 with following exceptions

- Read protection cannot be installed (configuration is identical to PROCON1).
- The bit 0 of byte 2 (position of PROCON1.16) is called SPREC (Soft-Programming Recover). It configures the behavior of the Flash startup, see **“Recovery From Aborted Logical Sector Erase (“ALSE”)” on Page 5-98**. It is however not readable in PROCON1, so its correct content can not be verified.
- The keywords are the keywords from user 1

User Configuration Block UCB2

- The UC **Page 0** (bytes 255–0) of UCB2 includes the following information
  - Bytes 1–0: OTP protection configuration bits, one for each sector with ROM functionality; structure of this configuration info is identical to the structure of the PROCON2 register (see **Chapter 5.6.5.3**).
  - Bytes 9–8: Copy of protection configuration bits
  - All other page bytes: zero
- The User Configuration **Page 1** is not used (reserved for future, all zero)
- The User Configuration **Page 2** (bytes 255–0) includes the following information
  - Bytes 3–0: 32-bit confirmation code. The confirmation code (also called lock code) indicates that a protection is correctly installed and enabled. The 32-bit confirmation code is defined as follows: “$8AFE15C3_H$”.
  - Bytes 11–8: Copy of confirmation code
  - All other page bytes: zero
- The User Configuration **Page 3** is not used (reserved for future, all zero).

Note: The configuration of sectors with ROM functionality can no more be changed after confirmation of its configuration, because the UCB2 block can never be erased.

The confirmation code for user 0 and user 1 shall be programmed only after check of correct programming of keywords, e.g. with the command ‘Disable Read Protection’. The confirmation code is programmed in the 2. wordline to exclude any possibility of disturbing the keywords or the protection configuration while writing the confirmation code. The confirmation code is identical for all users. If a wrong confirmation code with correct ECC is detected after reset, protection is not enabled and the respective installation bits in status register FSR are not set. However, if a double-bit error is detected during fetching the confirmation code, protection is installed and activated.

During rampup after reset, always the original value is fetched at first. Only in case of double-bit error within the original, its copy is fetched. If this is also disturbed and cannot be corrected, the double-bit error is indicated in FSR and additionally the protection error bit PROER.

Note: Accesses to the User Configuration Blocks are only possible, if the protection at first has been disabled with correct passwords.

Note: Besides accesses to the User Configuration Blocks, CPU-controlled accesses to the configuration sector are only supported during Flash Test Mode and during startup procedure out of BootROM.

5.6.6 Interrupt, Error and Operation Control

Access and/or operational errors (e.g. wrong command sequences) may be reported to the user by interrupts, and they are indicated by flags in the Flash Status Register FSR. Additionally, bus errors may be generated resulting in CPU traps (also shortly called bus error traps, although this is not correct).

5.6.6.1 Interrupt Control

The Flash module supports immediate error and status information to the user by interrupt generation. One CPU interrupt request is provided by the Flash module to be controlled in the SCU. The source Flash (in devices with more than one PMU) of Flash interrupts is defined by the related service request input in the SCU. In the SCU, the Flash interrupts are controlled and indicated via bit 5 (FL0) and bit 6 (FL1 for PMU1) in the SCU-registers for interrupt control (registers INTSET, INTCLR, INTDIS, INTNP and INTSTAT).

The Flash interrupt can be issued because of following events:

- End of busy state: program or erase operation finished
- Operational error: program or erase operation aborted
- Verify error: program or erase operation not correctly finished
- Protection error
- Sequence error
- Single-bit error: corrected read data from PFlash or DFlash delivered
- Double-bit error in Program Flash or Data Flash.

Note: In case of an OPER or VER error, the error interrupt is issued not before the busy state of the Flash is deactivated.

The source of interrupt is indicated in the Flash Status Register FSR by the error flags or by the PROG or ERASE flag in case of end of busy interrupt. An interrupt is also generated for a new error event, if the related error flag is still set from a previous error interrupt.

Every interrupt source is masked (disabled) after reset and can be enabled via dedicated mask bits in the Flash Configuration Register FCON.

5.6.6.2 Trap Control

CPU traps are executed because of LMB bus errors, generated by the PMU in case of erroneous Flash accesses from the PMI (Program Fetch Synchronous Error: PSE trap) or DMI (Data Access Synchronous Error: DSE trap). LMB bus errors are generated synchronously to the bus cycle requesting the not allowed Flash access or the disturbed Flash read data. The error attributes are captured in the LMB Bus Control Unit in synchronous capture registers. In devices with more than one PMU the source Flash of

a bus error trap can only be determined via the address in the address capture register. Bus errors are issued because of following events:

- Not correctable double-bit error of 64-bit read data from PFlash or DFlash (if not disabled for margin check)
- Not allowed write access to read only register (see **Table 5-11**)
- Not allowed write access to ENDINIT protected register (see **Table 5-11**)
- Not allowed test register access (see **Table 5-11**)
- Not allowed data or instruction read access in case of active read protection
- Access to not implemented addresses within the register or array space, including accesses to DFlash in PMU1.
- Read-modify-write access to the Flash array.

Write accesses to the Flash array address space are interpreted as command cycles and initiate not a bus error but a sequence error if the address or data pattern is not correct. However, command sequence cycles, which address a busy Flash bank, are serviced with a retry-acknowledge[1)], not with a sequence error.

If the trap event is a double-bit error in PFlash or DFlash, it is indicated in the FSR. With exception of this error trap event, all other trap sources cannot be disabled within the PMU.

Note: A double-bit error trap during margin check can be disabled (via MARP or MARD register) and redirected to an interrupt request.

5.6.6.3 Handling Errors During Operation

The previous sections described shortly the functionality of “error indicating” bits in the flash status register **FSR**. This section elaborates on this with more in-depth explanation of the error conditions and recommendations how these should be handled by customer software. This first part handles error conditions occurring during operation (i.e. after issuing command sequences) and the second part (**Section 5.6.6.4**) error conditions detected during startup.

SQER “Sequence Error”

Fault conditions:

- Improper command cycle address or data, i.e. incorrect command sequence.
- New “Enter Page” in Page Mode.
- “Load Page” and not in Page Mode.
- “Load Page” results in buffer overflow.
- “Load Page” with mixed 32/64 transfers.
- First “Load Page” addresses 2. word.

1) Please note that the CPU (i.e. the initiator of the last command cycle) stalls as long it receives a retry-acknowledge.

- “Write Page” with buffer underflow.
- “Write Page” and not in Page Mode.
- “Write Page” to wrong Flash type.
- Byte transfer to password or data.
- “Clear Status” or “Reset to Read” while busy[1].
- “Erase Sector” command to DFlash.
- Erase UCB with wrong UCBA.

New state:

Read mode is entered with following exceptions:

- “Enter Page” in Page Mode re-enters Page Mode.
- “Write Page” with buffer underflow is executed.
- After “Load Page” causing a buffer overflow the Page Mode is not left, a following “Write Page” is executed.

Proposed handling by software:

Usually this bit is only set due to a bug in the software. Therefore in development code the responsible error tracer should be notified. In production code this error will not occur. It is however possible to clear this flag with “Clear Status” or “Reset to Read” and simply issue the corrected command sequence again.

PFOPER/DFOPER “Operation Error”

Fault conditions:

ECC double-bit error detected in Flash microcode SRAM during a program or erase operation in PFlash or DFlash. This can be a transient event due to alpha-particles or illegal operating conditions or it is a permanent error due to a hardware defect. This situation will practically not occur.

Attention: these bits can also be set during startup (see **Chapter 5.6.6.4**).

New state:

The Flash operation is aborted, the BUSY flag is cleared and read mode is entered.

Proposed handling by software:

The flag should be cleared with “Clear Status”. The last operation can be determined from the PROG and ERASE flags[2]. In case of an erase operation the affected physical sector must be assumed to be in an invalid state, in case of a program operation only the affected page. Other physical sectors can still be read. New program or erase commands must not be issued before the next reset.

1) When the command addresses the busy Flash bank, the access is serviced with retry acknowledge.

2) Only when both DFlash banks were busy, one with program and the other with erase the affected bank and operation can not be determined.

Consequently a reset must be performed. This performs a new Flash rampup with initialization of the microcode SRAM. The application must determine from the context which operation failed and react accordingly. Mostly erasing the addressed sector and re-programming its data is most appropriate. If a “Program Page” command was affected and the sector can not be erased (e.g. in Flash EEPROM emulation) the wordline could be invalidated if needed by marking it with all-one data and the data could be programmed to another empty wordline.

Only in case of a defective microcode SRAM the next program or erase operation will incur again this error.

Note: Although this error indicates a failed operation it is possible to ignore it and rely on a data verification step to determine if the Flash memory has correct data. Before re-programming the Flash the flow must ensure that a new reset is applied.

*Note: Even when the flag is ignored it is recommended to clear it. Otherwise all following operations — including “sleep” — could trigger an interrupt even when they are successful (see **Chapter 5.6.6.1**, interrupt because of operational error).*

PROER “Protection Error”

Fault conditions:

- Password failure.
- Erase/Write to protected sector.
- Erase UCB and protection active.
- Write UC-Page to protected UCB.

Attention: a protection violation can even occur when a protection was not explicitly installed by the user. This is the case when the Flash startup detects an error and starts the user software with read-only Flash (see **Chapter 5.6.6.4**). Trying to change the Flash memory will then cause a PROER.

New state:

Read mode is entered. The protection violating command is not executed.

Proposed handling by software:

Usually this bit is only set during runtime due to a bug in the software. In case of a password failure a reset must be performed in the other cases the flag can be cleared with “Clear Status” or “Reset to Read”. After that the corrected sequence can be executed.

VER “Verification Error”

Fault conditions:

This flag is a warning indication and not an error. It is set when a program or erase operation was completed but with a suboptimal result. This bit is already set when only

a single bit is left over-erased or weakly programmed which would be corrected by the ECC anyhow.

However excessive VER occurrence can be caused by operating the Flash out of the specified limits, e.g. incorrect voltage or temperature. A VER after programming can also be caused by programming a page whose sector was not erased correctly (e.g. aborted erase due to power failure).

Under correct operating conditions a VER after programming will practically not occur. A VER after erasing is not unusual.

Attention: this bit can also be set during startup (see **Chapter 5.6.6.4**).

New state:

No state change. Just the bit is set.

Proposed handling by software:

This bit can be ignored. It should be cleared with “Clear Status” or “Reset to Read”. In-spec operation of the Flash memory must be ensured.

If the application allows (timing and data logistics), a more elaborate procedure can be used to get rid of the VER situation:

- VER after program: erase the sector and program the data again. This is only recommended when there are more than 3 program VERs in the same sector. When programming the DFlash in field (EEPROM emulation) ignoring program VER is normally the best solution because its most likely cause are violated operating conditions. Take care that never a sector is programmed in which the erase was aborted. In the EEPROM emulation the algorithm must ensure this e.g. by programming a marker after finishing successfully the erase.
- VER after erase: the erase operation can be repeated until VER disappears. Repeating the erase more than 3 times consecutively for the same sector is not recommended. After that it is better to ignore the VER, program the data and check its readability. Again for EEPROM emulation its most likely cause are violated operating conditions. Therefore it is recommended to repeat the erase at most once or ignore it altogether.

For optimizing the quality of Flash programming see the following section about handling single-bit ECC errors.

*Note: Even when this flag is ignored it is recommended to clear it. Otherwise all following operations — including “sleep” — could trigger an interrupt even when they are successful (see **Chapter 5.6.6.1**, interrupt because of verify error).*

PFSBER/DFSBER “Single-Bit Error”

Fault conditions:

When reading data or fetching code from PFlash or DFlash the ECC evaluation detected a single-bit error (“SBE”) which was corrected.

This flag is a warning indication and not an error. A certain amount of single-bit errors must be expected because of known physical effects.

New state:

No state change. Just the bit is set.

Proposed handling by software:

This flag can be used to analyze the state of the Flash memory. During normal operation it should be ignored. In order to count single-bit errors it must be cleared by “Clear Status” or “Reset to Read” after each occurrence[1)].

Usually it is sufficient after programming data to compare the programmed data with its reference values ignoring the SBE bits. When there is a comparison error the sector is erased and programmed again.

When programming the PFlash (end-of-line programming or SW updates) customers can further reduce the probability of future read errors by performing the following check after programming:

- Change the read margin to “high margin 0”.
- Verify the data and count the number of SBEs.
- When the number of SBEs exceeds a certain limit (e.g. 10 in 2 MByte) the affected sectors could be erased and programmed again.
- Repeat the check for “high margin 1”.
- Each sector should be reprogrammed at most once, afterwards SBEs can be ignored.

In case of EEPROM emulation using DFlash the verification of programmed data should be done with the normal read level and SBEs should be ignored. When a comparison error is found the sector can usually not be erased because it contains active data in other pages. The emulation algorithm can mark the affected page as invalid and program the data to a following page. As always the number of consecutive repetitions should be limited (e.g. to 3) as protection against violated operating conditions.

To keep the EEPROM emulation alive even when wordline oriented fails occur (e.g. due to over-cycling) the algorithm can implement the following scheme for highest possible robustness:

- Before programming a page save the content of the other page on the same wordline in SRAM.
- Program the new page and compare the content of this page and of the saved page with their reference data. This can be done with normal read margins. Ignore SBEs.

1) Further advice: remember that the ECC is evaluated when the data is read from the PMU. When counting single-bit errors use always the non-cached address range otherwise the error count can depend on cache hit or miss and it refers to the complete cache line. As the ECC covers a block of 64 data bits take care to evaluate the FSR only once per 64-bit block.

- If the data comparison fails program this page and the saved content of the other page to a different wordline.
- This procedure can be repeated if the data comparison fails again. The number of repetitions should be limited (e.g. to 3) in case the programming fails because of out-of-spec operating conditions.
- Wordline oriented fails can also have the effect that the affected wordlines can not be erased anymore (other wordlines stay fully functional). A robust EEPROM emulation is immune against such wordlines (e.g. by identifying old data by version counters).

Due to the specificity of each application the appropriate usage and implementation of these measures (together with the more elaborate VER handling) must be chosen according to the context of the application.

5.6.6.4 Handling Errors During Startup

The FSR flags are not only used to inform about the success of Flash command sequences but they are also used to inform (1) the startup software and (2) the user software about special situations incurred during startup. In order to react on this information these flags must be evaluated after reset before performing any flag clearing sequence as “Clear Status” or “Reset to Read”.

The following two levels of situations are separated:

- Fatal level: the user software is not started. A WDT reset is performed.
- Error level: the user software is started but the Flash memory must not be programmed or erased.
- Warning level: the user software is started but a warning is issued.

Fatal Level (WDT Reset)

These error conditions are evaluated by the startup software which decides that the Flash is not operable and thus waits for a WDT reset. The application sees only a longer startup time followed by a WDT reset.

The reason for a failed Flash startup can be a hardware error or damaged configuration data.

Error Level (Flash Read-Only)

In this condition the user software is started but the Flash memory must not be programmed or erased. If writability of the Flash is mandatory the user software itself has to perform a reset.

Flash microcode error:

FSR bits set: PFOPER and DFOPER.

The user software is started normally but the Flash must not be programmed or erased. Please note that programming or erasing is not blocked by hardware. Issuing program or erase sequences despite this condition is forbidden.

Warning Level

These conditions inform the user software about an internally corrected or past error condition.

Logical sector corrected:

FSR bits set: VER.

The Flash detected that a logical sector erase was apparently aborted by reset or power failure. In order to maintain readability of the other logical sectors this sector was corrected. Its erase operation must be repeated. See also **“Recovery From Aborted Logical Sector Erase (“ALSE”)” on Page 5-98**.

Leftover OPER:

FSR bits set: PFOPER or/and DFOPER.

The OPER flags are only cleared by the command sequence “Clear Status” or with a power-on reset (Class 0). After any other reset a OPER flag can still be set when the user software is started.

Single-bit error in protection:

FSR bits set: PFSBER.

An corrected ECC single-bit error was detected during installation of the protection.

5.6.6.5 Application Hints and Guidelines

Every command execution is started with the last command cycle of the command sequence, and it is indicated by the busy bit of Program Flash or Data Flash in the status register FSR. Depending on the specific operation additionally one of the following status bits may be set:

- PROG indicating a requested, running or just finished program operation (page, User Configuration Page),
- ERASE indicating a requested, running or just finished erase execution (sector, User Configuration Block)

The status bits have to be cleared by SW. It is thus possible to qualify the BUSY bits and the OPER bits with the two status bits PROG and ERASE (see FSR description in **Chapter 5.6.3.7**). Polling the BUSY bits show directly the termination of the specific operation. The termination is additionally reported to the CPU by an end-of-busy interrupt (if enabled in the FCON register). Because the termination could have been also caused by a fault condition, the error flags shall be sampled after each command sequence and after the termination of its execution. All error conditions are additionally indicated to the CPU by an error interrupt, if enabled in the FCON register.

In summary a flash operation shall be controlled by the following process:

- Write command sequence to Flash
- Check for correct command sequence by sampling the SQER and PROER bits in status register FSR, or install a proper error interrupt reaction.
- In case of an indicated fault condition: clear the error flag with a Clear Status or a Reset to Read command and start a specific SW reaction, for example a retry operation. If no error:
- In case of a Flash array operation: check, if the command has been correctly requested by polling the PROG or ERASE bit in status register
- Check, if the command is in operation by sampling the BUSY bit
- When the execution of the command is finished (reported per interrupt or noticed per polling the busy flag) check the VER and OPER flags; if one flag is set: see **Chapter 5.6.6.3** for proper reaction; else: clear PROG or ERASE flag and continue user program execution.

The status flags PROG and ERASE are cleared only with a power-on reset. It is therefore possible to detect, if a program or erase operation has been interrupted by a warm reset. Assumption: These flags are normally cleared by SW (with the Clear Status command) immediately after termination of the operation.

In the following, some additional application hints and guidelines are presented, to be considered by the user (some of them are already noted in other sections of the chapter):

- User code, that writes command sequences to the Program Flash, should not be executed from the same internal Flash module (because the Flash takes the busy state with last command cycle); it shall be located in other internal or external

program memory, e.g. in the scratchpad SPRAM, or in the other Flash module. But user code, that writes command sequences to the Data Flash, can be located in and executed from the Program Flash in the same Flash module.

- The write cycles, belonging to a command sequence, must access the Flash in its non-cached address space (otherwise they will remain in the data cache). Additionally, it is recommended to include a dummy read (ld.w) instruction to a PMU register (e.g. PMU_ID) after the last write cycle of a command sequence to flush the write buffers.
- After change of margin level, a wait time of min. 10 µsec. is necessary for sense amp adjustment before read operations are executed with the modified margins.
- After installation of OTP write protection for sectors with ROM functionality or after installation of tuning protection, accesses to Flash SFRs are no more possible; this greatly reduces the FAR analysis possibility.
- If write protection is configured in the user's UCB page 0 but not confirmed via page 2, the protection is only partially enabled after next reset (see **Chapter 5.6.5.2**).
- The selection of wait states for PFlash and DFlash accesses must be controlled via the FCON register by the user in relation to his application frequency (see **Table 5-1**).
- The performance of data read accesses to the Program Flash can be influenced by support of buffer hit or cache hit mechanisms in PMU and DMI; therefore data locations in Flash shall be located sequentially, so that the read buffer in PMU (32 bytes) and the cache/buffer line in DMI (16 bytes) are optimally used. This has especially to be considered in devices with more than one PMU, where data accesses can be isolated from instruction accesses by proper assignments to different PMUs.

For customer Flash tests it may be wanted to perform checkerboard tests of Program Flash or Data Flash. Programming checkerboard patterns into the Flash can simply be done, because a Flash wordline always consists of two sequential and bitwise-interleaved pages with according even and odd page addresses (see **Table 5-7**). Thus, it is only necessary to program complete pages either with ones or zeros to get a checkerboard pattern. Always four sequential pages must then be programmed as follows: Page 0 with all ones, page 1 remains erased (all zeros), page 2 remains erased (all zeros), page 3 is programmed with all ones. Identically the next four pages are treated, and so on.

5.6.7 Power Supply and Reset

The following chapters describe the required power supplies, the power consumption and its possible reduction, the control of Flash Sleep Mode and the basic control of Reset.

5.6.7.1 Power Supply

The flash module uses the standard V_{DDP} I/O power supply to generate the voltages for both read and write functions. A VPP pin is not used for write operations. Internally generated and regulated voltages are provided for the program and erase operations as well as for read operations. The standard V_{DD} is used for all digital control functions.

5.6.7.2 Flash Power Consumption

For dynamic reduction of power consumption, the PMU controls the activity of the word line drivers of the Flash array, if enabled in the FCON register. If this dynamic power reduction is enabled, the read performance is slightly reduced because the control of prefetching is limited in this case. In DFlash, dynamic power reduction is provided per default, because DFlash prefetching is not supported.

5.6.7.3 Flash Sleep Mode

As power reduction feature, the Flash module provides a Flash sleep mode which can be selected by the user individually for the Flash, if the FCON-bit 15 (SLEEP) is set. Additionally, the sleep mode can be requested by a global SLEEP signal from a Power Management System. This ‘external’ sleep request signal is only accepted by Flash state machines, when it is not disabled with the FCON-bit ESLDIS.

The requested sleep mode is only taken if the Flash is in idle state and when all pending or active requests are processed and terminated. Only then, the Flash array performs the ramp down into the sleep mode: the sense amplifiers are switched off, the voltages are ramped down and the array-oscillator is switched off.

As long as the Flash is in sleep mode, this state is indicated by the SLM bit in the Flash Status Register FSR.

Wake-up from sleep is controlled with clearing of bit FCON.15, if selected via this bit, or wake-up is initialized by trailing edge of the broadcasted (’external’) sleep signal from SCU. After wake-up, the Flash oscillator is switched on, the voltages are ramped up and the Flash takes the read mode. Note: A wake-up is only accepted by the Flash in its sleep mode; it is not accepted while it is ramped down.

Note: During ramp-down, sleep and wake-up, the Flash is reported to be busy. Thus, read and write accesses to the Flash in sleep mode are acknowledged with ‘retry’ and should therefore be avoided; those accesses make sense only during wake-up, when waiting for the Flash read mode.

Note: The wake-up time is documented in the data sheet. This time may fully delay the interrupt response time in sleep mode.

5.6.7.4 Reset Control

The PMU-part of Flash module (FIM) uses

- the application reset (“class 3 reset”), which may include all reset sources (power-on, HW, SW and watchdog reset, if configured), and
- the power-on reset.

If not otherwise stated, always the application/class 3 reset is used.

The flash will be automatically reset to the read mode after every reset.

Note: All implemented PMUs and Flash arrays execute their rampups autonomously and in parallel by accessing their own configuration sectors in PFlash0 and PFlash1. Although the startup SW in BootROM has to check the FSRs of both Flash modules for proper and error-free rampup, the startup time is nearly unaffected by the second PMU and Flash.

Resets During Flash Operation

A reset or power failure during an ongoing Flash operation (i.e. program or erase) must be considered as violation of stable operating conditions. However the Flash was designed to prevent damage to non-addressed Flash ranges when the reset is applied as defined in the data sheet. The exceptions are erasing logical sectors and UCBs. Aborting an erase process of a logical sector can leave the complete physical sector unreadable. An automatic recovery mechanism is implemented (see next section). When an UCB erase is aborted the complete Flash can become unusable. There is no recovery implemented because UCBs are usually only erased in a controlled environment. The addressed Flash range is left in an undefined state.

When an erase operation is aborted the addressed logical or physical sector can contain any data. It can even be in a state that doesn’t allow this range to be programmed.

When a page programming operation is aborted the page can still appear as erased (but contain slightly programmed bits), it can appear as being correctly programmed (but the data has a lowered retention) or the page contains garbage data. It is also possible that the read data is instable so that depending on the operating conditions different data is read.

For the detection of an aborted Flash process the flags FSR.PROG and FSR.ERASE could be used as indicator but only when the reset was an application reset. Power-on resets can not be determined from any flags. It is not possible to detect an aborted operation simply by reading the Flash range. Even the margin reads don’t offer a reliable indication.

When erasing or programming the PFlash usually an external instance can notice the reset and simply restart the operation by erasing the Flash range and programming it again.

However for the case of EEPROM emulation in the DFlash this external instance is not existing. A common solution is detecting an abort by performing two operations in sequence and determine after reset from the correctness of the second the completeness of the first operation.

E.g. after erasing a DFlash sector a page is programmed. After reset the existence of this page proves that the erase process was performed completely.

The detection of aborted programming processes can be handled similarly. After programming a block of data an additional page is programmed as marker. When after reset the block of data is readable and the marker is existent it is ensured that the block of data was programmed without interruption.

Because often very small amounts of data need to be programmed in EEPROM emulation not always a complete page can be spent as marker. The following recipe allows to reduce the marker size to 8 bytes. This recipe violates the rule that a page may be programmed only once. This violation is only allowed for this purpose (EEPROM emulation with DFlash) and only when the algorithm is robust against disturbed pages (see also recommendations for handling single-bit errors) by repeating a programming step when it detects a failure.

Robust programming of a page of data with an 8 byte marker:

1. After reset program preferably always first to an even page (“Target Page”).
2. If the Other Page on the same wordline contains active data save it to SRAM (the page can become disturbed because of the 4 programming operations per wordline).
3. Program the data to the Target Page.
4. Perform strict check of the Target Page (see below).
5. Program 8 byte marker to Target Page.
6. Perform strict check of the Target Page.
7. In case of any error of the strict check go to the next wordline and program the saved data and the target data again following the same steps.
8. Ensure that the algorithm doesn’t repeat unlimited in case of a violation of operating conditions.

Strict checking of programmed data:

1. Ignore single-bit errors and the VER flag.
2. Switch to tight margin 0.
3. If the data (check the complete page) is not equal to the expected data report an error.
4. If a double-bit error is detected report an error.

When this algorithm is performed while the other DFlash bank is busy with erasing an additional rule is violated: margin reads are normally not allowed while any Flash operation is ongoing. In this case this is allowed.

After reset the algorithm has to check the last programmed page if it was programmed completely:

1. Read with normal read level. Ignore single-bit errors.
2. Read 8-byte marker and check for double-bit error.
3. Read data part and verify its consistency (e.g. by evaluating a CRC). Check for double-bit error.
4. If the data part is defective don't use it (e.g. by invalidating the page).
5. If the data part is ok:
   a) If the marker is erased the data part could have been programmed incompletely. Therefore the data part should not be used or alternatively it could be programmed again to a following page.
   b) If the marker contains incorrect data the data part was most likely programmed correctly but the marker was programmed incompletely. The page could be used as is or alternatively the data could be programmed again to a following page.
   c) If the marker is ok the data part was programmed completely and has the full retention. However this is not ensured for the marker part itself. Therefore the algorithm must be robust against the case that the marker becomes unreadable later.

Recovery From Aborted Logical Sector Erase ("ALSE")

When while erasing one of the logical sectors a power failure occurs or a reset is triggered the aborted erase process might leave the complete physical sector unreadable. As often the logical sectors contain important boot code the application might not start anymore. Thus the recommended step to repeat the aborted erase after startup can not be realized.

The FAM implements two recovery algorithms. The selection between the two is done with the SPREC bit ("Soft-Programming Recovery"), i.e. bit 0 of byte 2 in the UCB1 (see **"User Configuration Block UCB1" on Page 5-82**).

- The default algorithm is selected with SPREC = 0. In this mode the FAM searches for a logical sector that is in an over-erased state which prevents reading of any data in the complete physical sector. When such an over-erased logical sector is found this algorithm programs it shortly with all-one data. The other logical sectors become readable again. The execution of this algorithm is noted by setting the VER flag (see **Chapter 5.6.6.4**).
  At least theoretically (especially when operating the device outside of the allowed operating conditions) this algorithm could destroy valid data: when an over-erased logical sector is reported incorrectly during normal startup without a preceding sector erase the data of a logical sector would be overwritten with all-one.
- An alternative algorithm is selected when SPREC = 1. This algorithm searches for an over-erased logical sector as before. For repair a smarter but more time consuming algorithm is performed. The affected logical sector is not overwritten with all-one but only the over-erased 0-bits are slightly programmed so that they become normal 0-bits again. Under all circumstances this algorithm can not destroy any data but when

a lot of data has to be repaired the flash startup time can be increased to over 250 ms.

6 Data Access Overlay (OVC)

The data overlay functionality provides the capability to redirect data accesses by the TriCore to program memory (segments 8_H and A_H) called “target memory” to a different memory called “overlay memory”.

Depending on the device the following overlay memories can be available:

- Overlay SRAM in the PMU.
- Emulation Memory[1)].
- External memory[2)].

This functionality makes it possible, for example, to modify the application’s test and calibration parameters (which are typically stored in the program memory) during run time of a program.

As the address translation is implemented in the DMI, it affects only data accesses (reads and writes) of the TriCore. Instruction fetches by the TriCore or accesses by any other master (including the debug interface) are not redirected.

Summary of Features and Functions

- 16 overlay ranges (“blocks”) configurable.
- Support of 8 Kbyte embedded Overlay SRAM (OVRAM) in PMU.
- Support of up to 512 Kbyte overlay/calibration memory (EMEM)[1)].
- Support of up to 2 MB overlay memory in external memory (EBU space)[2)].
- Support of Online Data Acquisition into range of up to 32 KB and of its overlay.
- Support of different overlay memory selections for every enabled overlay block.
- Sizes of overlay blocks selectable depending on the overlay memory:
  - OVRAM: from 16 byte to 2 Kbyte.
  - EMEM[1)] and external memory[2)]: 1 Kbyte to 128 Kbyte.
- All prepared overlay blocks can be enabled with only one register write access.
- Programmable flush (invalidate) control for data cache in DMI.

6.1 Basic Overlay Control

This chapter describes the complete overlay functionality irrespective of the overlay memory availability.

Per overlay block, there are three possibilities for redirection of the original data address, redirection to the Overlay SRAM in the PMU, redirection to the external memory and redirection to the Emulation Memory EMEM, if the chip includes the Emulation Extension Control EEC for an Emulation Device. In all cases, the same overlay mechanism is used.

The basic overlay scheme is shown in **Figure 6-1**.

1) Only available in Emulation Device “ED”.

2) Only available in Emulation Device with EBU.

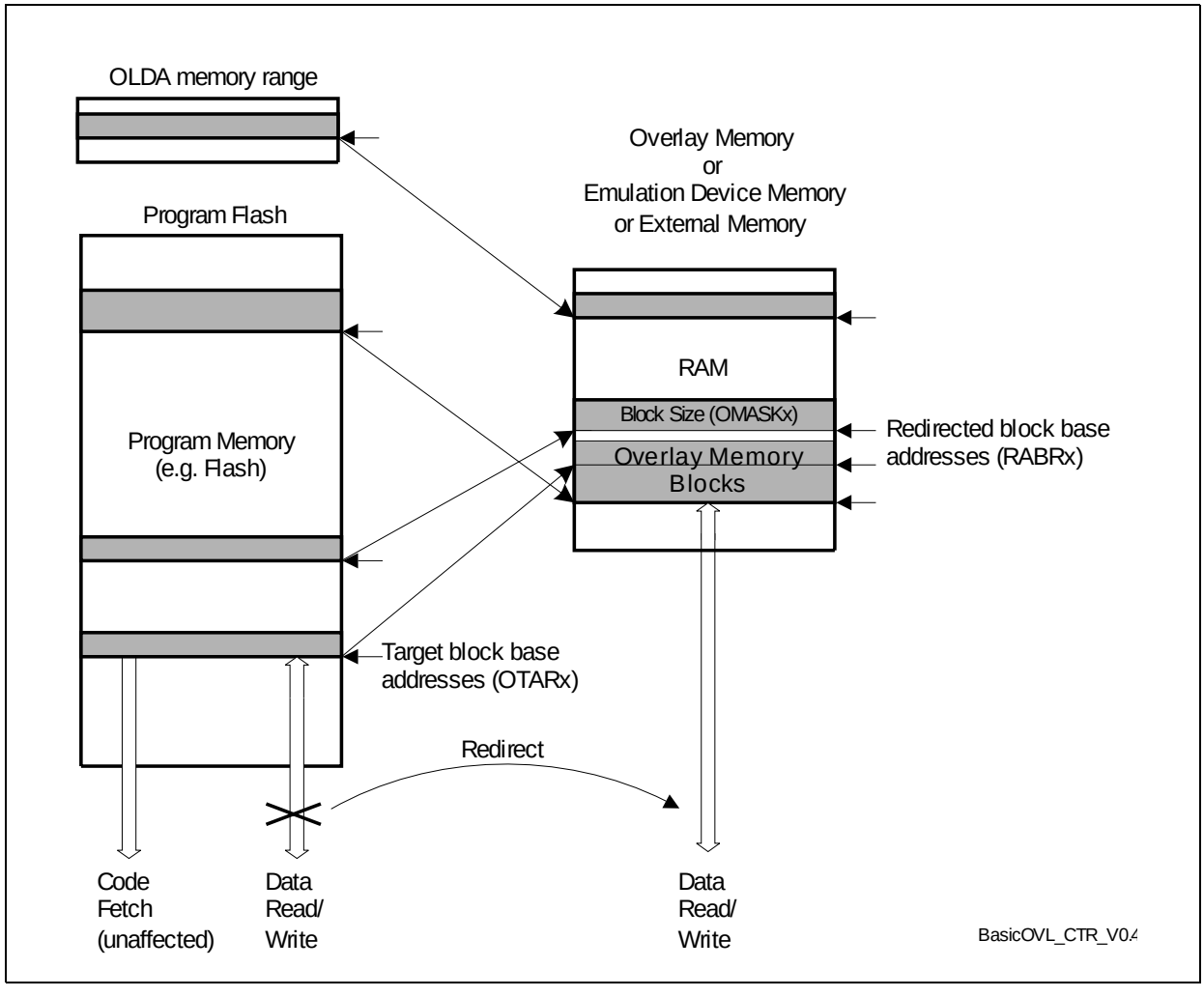


Figure 6-1 Redirection of Data Accesses to/from Code Memory to Internal OVRAM or to Emulation Memory (or to External Memory)

In the TC1797, the target memory (Program Flash, external memory or OLDA range, see **Chapter 6.4.1**) can be divided into a maximum of sixteen memory blocks for redirection into an overlay memory. The base address in target and overlay memory as well as the block size of each overlay block can be individually selected. The possible sizes of overlay blocks depend on the selected overlay memory: Blocks in the internal OVRAM are smaller than overlay blocks in EMEM or external memory. All enabled overlay blocks can be overlaid (redirected) together with only one register access. Concurrently, the data cache in the DMI may be flushed.

The operation of the address translation process is described in **Figure 6-2**, shown for redirection into the internal Overlay RAM OVRAM.

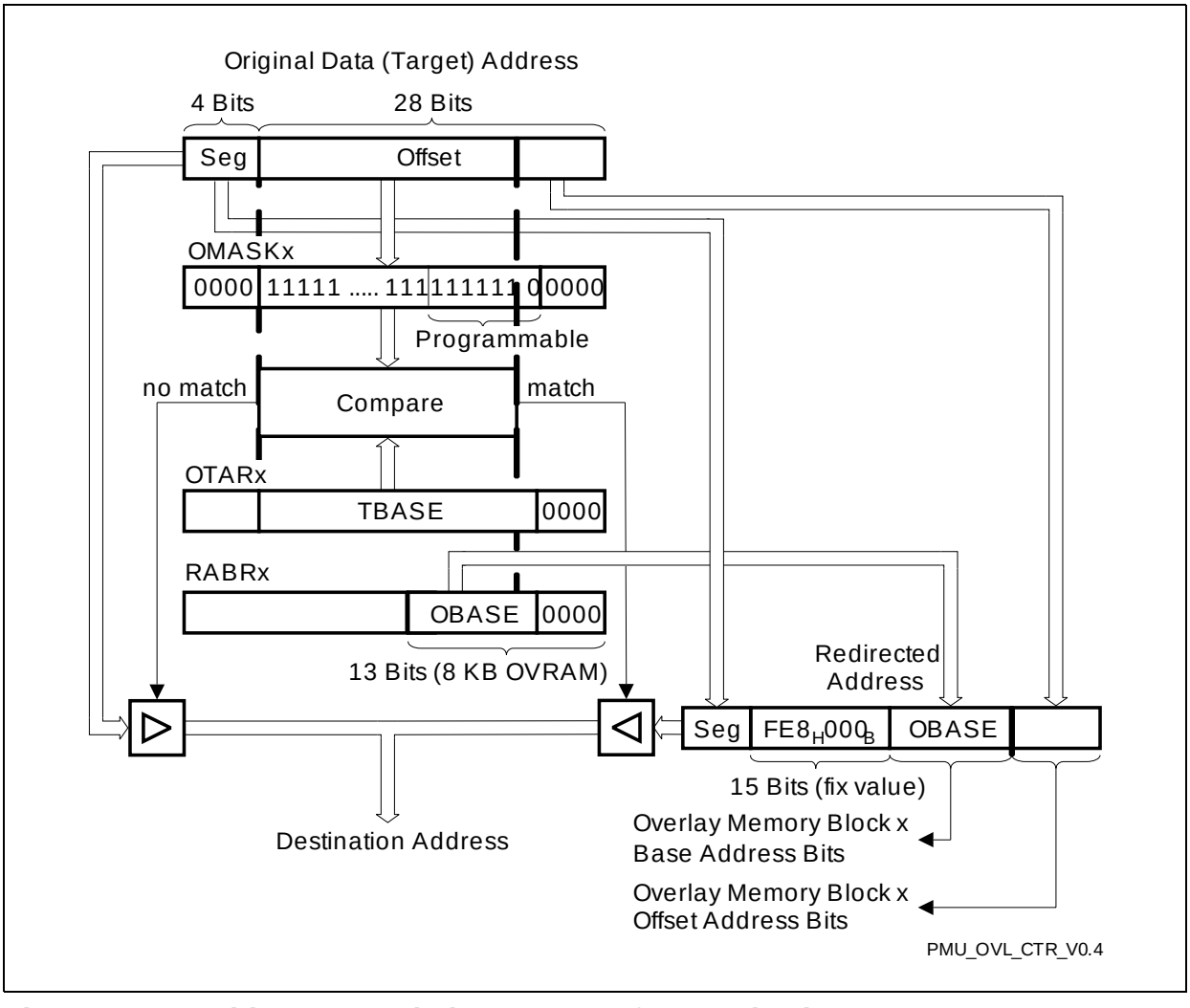


Figure 6-2 Address Translation Process for Overlay into OVRAM

In each enabled overlay block control logic, three registers hold the information to control the overlay functionality:

- The overlay target address in the Overlay Target Address Register OTARx, which determines the base address of the program or OLDA memory data block x to be redirected.
- The base address of the overlay memory block in the OVRAM, external memory or EMEM (if emulation Device) in the Redirected Address Base Register RABRx, and the related enable and control bits.
- A mask in the Overlay Mask Register OMASKx, defining the size of the block, the address bits to be checked for an address match, and which bits are used from the redirected address base and which from the original data address.

The size of the overlay memory blocks can be 2^n times the minimal block size (16 byte for internal memory overlay or 1 Kbyte for Emulation Memory overlay, respectively) with

n=0-7. The start address of the block can be an integer multiple of the programmed block size (natural page boundary).

If the data segment address is A_H or 8_H, the segment offset of the original data address is compared with the target base addresses of all overlay blocks which are enabled in RABRx. This bit-wise comparison is qualified by the content of the mask, ignoring the address bits that form the offset into the overlaid block.

If there is no match, the original data address is taken to perform the access.

If there is a match (see **Figure 6-2**):

- The four segment address bits (A_H or 8_H) are taken directly from the data address.
- The most significant part of the segment offset, that addresses the base address of the overlay memory, is set to predefined values according to the address map (fixed bits in registers RABRx).
- The part of the target block address, that corresponds to the overlay block base address, is replaced by the respective overlay block base address bits (bits OBASE in RABRx, where the corresponding mask bits OMASK in registers OMASKx are set to “1”).
- The address is completed by the original offset into the block; the number of bits used are determined by the bits set to “0” in the mask OMASK.

6.2 Online Data Acquisition (OLDA) and its Overlay

Calibration is additionally supported by an OLDA memory range of up to 32 Kbyte, which is a virtual memory and physically only available, if it is redirected (as described above) to the internal or external overlay memory or to the EMEM in Emulation Device. If OLDA is enabled in PMU, direct write accesses (without redirection) to the OLDA range are not really executed, and they do not generate a bus error trap. This trap suppression works only for accesses to the non-cached range. Read accesses to the OLDA range generate a bus error trap, if not redirected to a physically available overlay block.

The base address of the virtual OLDA memory range is A/8FE7 0000_H, the end address is A/8FE7 $7FFF_H$. Accesses to the OLDA range are also supported in cached address space but there the bus error trap for write accesses is not suppressed.

Note: In OTARx registers, any target address can be selected for redirection, thus also addresses in the OLDA range. However, the handling of direct accesses to the OLDA range is completely controlled in the PMU.

6.3 Enable Control of Overlay Blocks

For basic control of overlay execution, the OCON register is provided with following functionality:

- 16 enable bits (SHOVENx), one for every overlay block configuration, to support concurrent and parallel switching of up to 16 overlay ranges (blocks); shadow

functionality to the single enable bits in the 16 block control registers (RABRx) provide compatibility to enable-control in TC1766/96.
- One common overlay start bit (OVSTRT) to enable all prepared (enabled) overlay configurations by writing all shadow enable bits into the 16 block control registers in parallel (write-only bit); stop-function if all-zeros are written.
- One control flag (DCINVAL), to be set by the CPU or Cerberus, to flush (invalidate) the (clean) data cache lines in the DMI (write-only bit).
- One common overlay stop (OVSTP) bit to disable all overlay configurations in the 16 block control registers (write-only bit), without changing the configuration
- One overlay configured status bit (OVCONF), which may be set when overlay registers are configured by the Cerberus via JTAG interface, and which may be cleared by the CPU after common overlay start (re-direction of all enabled overlay blocks).

The control flag in the high byte of the Overlay Control Register OCON (see **Page 6-20**) is implemented with additional protection bit, supporting write access to OCON by different users without violation of such control bit which shall remain unchanged. Additionally, byte protection is possible by support of byte write accesses to OCON.

6.4 Target and Overlay Memories

In the following, the possible target and overlay memories are described, and for the overlay memories also their block-specific selection and the possible block sizes. The Internal Overlay Memory OVRAM and the interface to the Emulation Memory EMEM are located in the PMU. The EMEM can only be selected for overlay blocks, if the chip is an Emulation Device ED.

6.4.1 Target Memories

Any data read or write access to the segments 8_H and A_H is checked for a valid overlay target address, using all 16 OTARx registers concurrently for comparison (if they are enabled for overlay execution). Since the OTARx registers are writable, any data memory within the segments 8_H and A_H may be used for redirection to an overlay memory. Thus, the following memories can be selected as target memories:

- Program Flash
- Data Flash
- The (virtual) OLDA memory range
- The external memory.

6.4.2 Internal Overlay Memory

The capacity of the Internal Overlay Memory OVRAM is 8 KB. The base address of the OVRAM is A/8FE8 0000_H (non-cached/cached space). The OVRAM is selected for overlay execution, if the bits IEMS and EXOMS in the block-related RABRx register are

zero. During address translation, the upper 19 address bits are set to $A/8FE8_H000_B$ using the same segment address as the original data (target) address. For internal overlay, the size of the overlay blocks can be 2^n x 16 B, with n = 0 to 7 (16 byte to 2 Kbyte). The internal overlay memory OVRAM is available in both the Production Device and the Emulation Device.

6.4.3 Emulation Overlay Memory

In the corresponding emulation device “ED” an Emulation Memory of 512 Kbyte is provided, which can fully be used for calibration via program memory or OLDA overlay. Its base address is $A/8FF0\ 0000_H$. The Emulation Memory EMEM is selected for overlay execution, if the block-related RABRx bits IEMS=1 and EXOMS=0. During address translation, the upper 13 address bits are set to AFF_H0_B (non-cached) or to $8FF_H0_B$ (cached space) using the same segment address as the original data address.

For Emulation Memory (EMEM) overlay, the size of the overlay blocks can be $2^n \times 1$ Kbyte, with n = 0 to 7 (1 Kbyte to 128 Kbyte).

6.4.4 External Overlay Memory

If an external memory is available in the Emulation Device system, it can also be used for calibration via program memory or OLDA overlay. The External Memory is selected for overlay execution, if the block-related RABRx bits IEMS=1 and EXOMS=1. During address translation, the upper 9 address bits are set to $A0_H1_B$ (non-cached) or to 80_H1_B (cached space) using the same segment address as the original data target address.

For redirection into the external EBU memory, the same sizes of the overlay blocks are provided as for Emulation Memory overlay: $2^n \times 1$ Kbyte, with n = 0 to 7 (1 Kbyte to 128 Kbyte). Thus, the maximum space supported for the overlay region is 2 MB.

6.5 Change of Overlay Parameters and Overlay Start

When changing the overlay parameters of a block or when switching a block from one overlay memory to another overlay memory, it must be ensured that the respective OVEN bit in register RABRx is reset, before the block parameters are set properly, and then the overlay block is enabled again. Otherwise, unintended access redirections may occur.

It is especially supported to enable (start) all prepared overlay blocks concurrently, when using the shadow mechanism for the OVEN bits in the one OCON register instead of the single OVEN bits in the different RABRx registers (see **Chapter 6.3**). With this function it is possible to switch directly from one set of overlay blocks to another set of overlay blocks, without any restriction concerning the block-specific use of (available) overlay memories.

Note: The Overlay Control does not prevent configuring the translation logic incorrectly so that memory accesses are translated to not implemented or forbidden memory ranges.

6.6 Block Priority and Access Performance

If concurrent matches in more than one enabled overlay block occur, the block with the lowest order number will win and perform the address translation.

The dynamic address translation for redirection to the overlay memory is executed without performance penalty.

6.7 Overlay Control Registers

Figure 6-3 shows all the overlay control registers associated with control of the overlay memory blocks.

Overlay Control Registers Overview

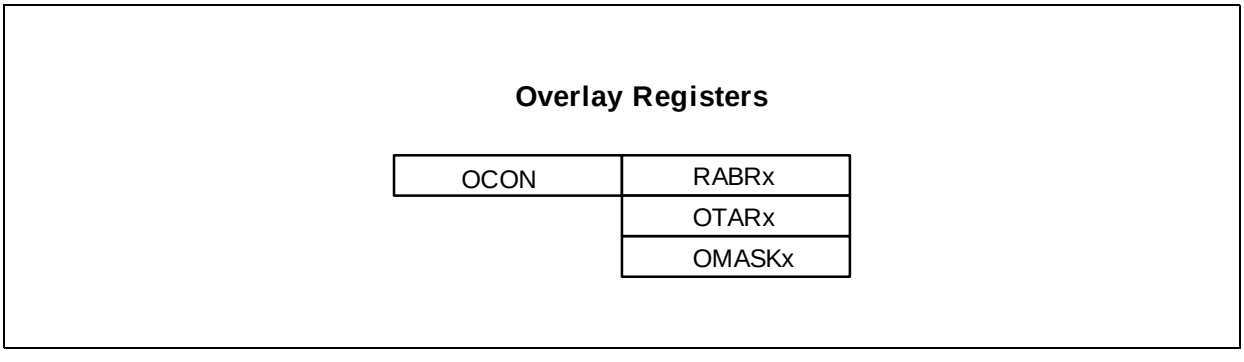


Figure 6-3 Overlay Control Registers

The address space for the overlay control registers is as follows:

Table 6-1 Registers Address Space of OVC Registers

Module	Base Address	End Address	Note
OVC	F87F FB00$_H$	F87F FBFF$_H$	

Table 6-2 Registers Overview

Register[1] Short Name	Register Long Name	Offset Address	Access Mode[2]		Description see
			Read	Write	
OTARx	Overlay Target Address Register x (x = 0-15)	0024_H + x ∗ C_H	U, SV	SV	Page 6-9
RABRx	Redirected Address Base Register x (x = 0-15)	0020_H + x ∗ C_H	U, SV	SV	Page 6-11
OMASKx	Overlay Mask Register x (x = 0-15)	0028_H + x ∗ C_H	U, SV	SV	Page 6-16
OCON	Overlay Control Register	$00E0_H$	U, SV	SV	Page 6-20

1) The OVC register short names are extended with the module name prefix “OVC_”.

2) Symbol U: Access permitted in User Mode 0 or 1
Symbol SV: Access permitted in Supervisor Mode

Note: Accesses to free/not used register addresses within the OVC address space are not executed and not serviced with a bus error trap.

Register Descriptions

For each of the 16 overlay memory blocks (indicated by index x), three registers control the overlay operation and the memory selection:

- The Overlay Target Address Register OTARx, which holds the base address of the memory block in internal Flash, in external memory or in the OLDA memory being overlaid (and being compared with original data address).
- The Redirected Address Base Register RABRx, which holds the base address of the overlay memory to be used (with fixed address bits) and of the overlay memory block within the overlay memory, and some control bits.
- The Overlay Mask Register OMASKx, which determines which bits (from RABRx) are used for the base address (of overlay memory and block) and which bits (of original data address) are directly used as offset within the block (remaining unchanged).

Additionally, for general overlay control the register OCON is provided.

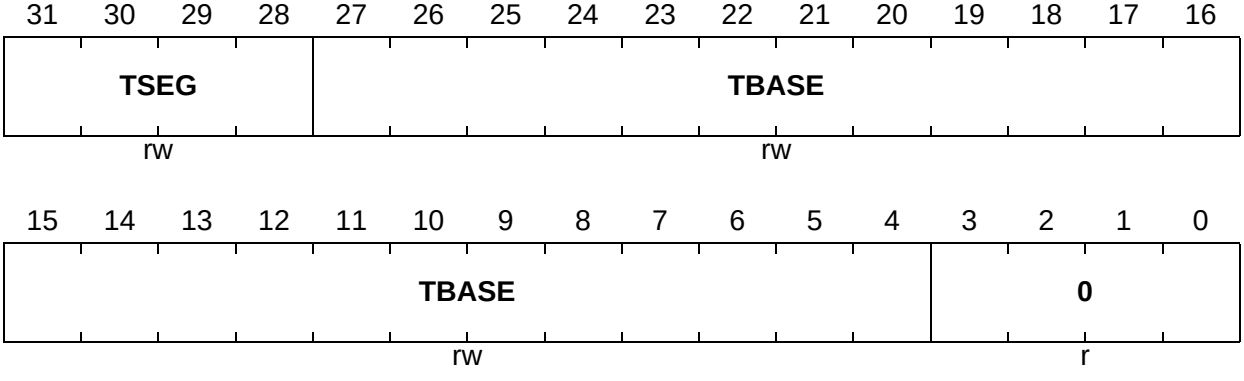
All overlay block and control registers are reset to their default values with the application reset. A special debug reset is not considered.

Note: All overlay block control registers have different definitions for overlay blocks in internal OVRAM (RABRx.IEMS=0), and overlay blocks in EMEM or external memory (RABRx.IEMS=1 or RABRx.EXOMS=1). Therefore, in the following, two (RABRx: three) different definitions are provided for the same register.

If RABRx.IEMS=0, the OTARx register is defined as follows.

OTARx (x=0-15)
Overlay Target Address Register x

(24_H+x*C_H) **Reset Value: 0000 0000_H**

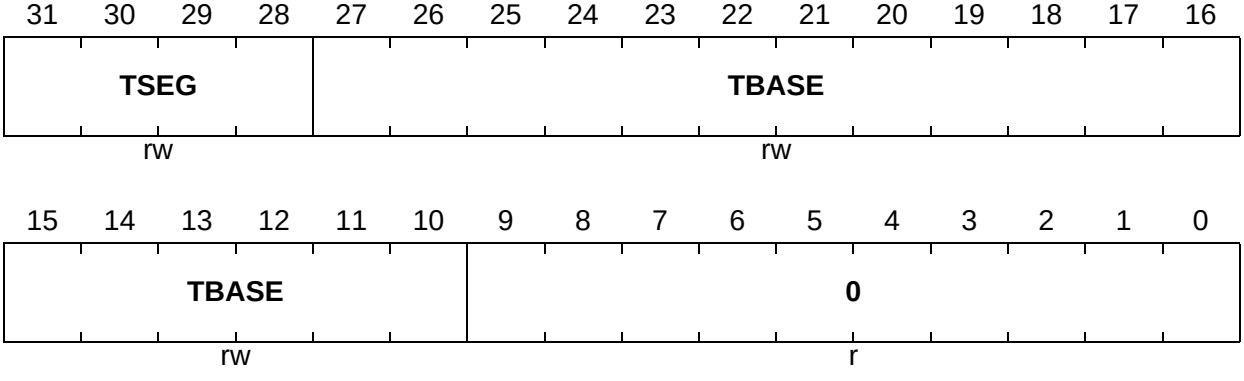


Field	Bits	Type	Description
TBASE	[27:4]	rw	**Target Base** This field holds the base address of the overlay memory block in the target memory (Program Flash or OLDA memory or external memory).
TSEG	[31:28]	rw	**Target Segment (reserved)** This bit field is reserved for future use, to select a segment. In TC1797 implementation, any access to segments 8_H, or A_H will be checked for a valid base address; return 0 if read; should be written with 0.
0	[3:0]	r	**Reserved** Read as 0; should be written with 0.

If RABRx.IEMS=1, the OTARx register is defined as follows.

OTARx (x=0-15)
Overlay Target Address Register x

(24_H+x*C_H) **Reset Value: 0000 0000_H**

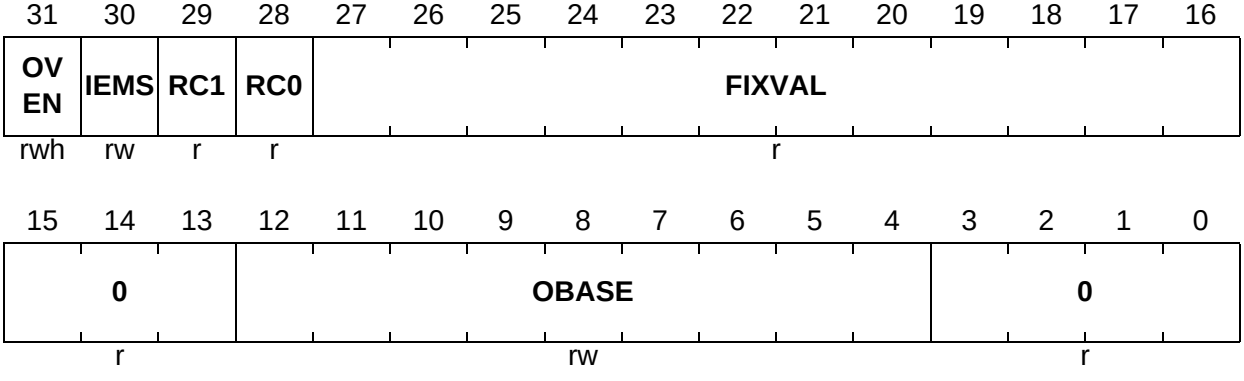


Field	Bits	Type	Description
TBASE	[27:10]	rw	**Target Base** This field holds the base address of the overlay memory block in the target memory (Program Flash or OLDA memory or external memory).
TSEG	[31:28]	rw	**Target Segment (reserved)** This bit field is reserved for future use, to select a segment. In TC1797 implementation, any access to segments 8_H, or A_H will be checked for a valid base address; return 0 if read; should be written with 0.
0	[9:0]	r	**Reserved** Read as 0; should be written with 0.

If RABRx.IEMS=0, the RABRx register is defined as follows.

RABRx (x=0-15)
Redirected Address Base Register x

(20_H+x*C_H) **Reset Value: 0FE8 0000_H**



Field	Bits	Type	Description
OBASE	[12:4]	rw	**Overlay Block Base Address** This bit field holds the base address of the overlay memory block in the overlay memory OVRAM. The largest block base address that is allowed, is ($OBASE_{max}$ - block size[1)])
FIXVAL	[27:16]	r	**Fixed Value** Base address of OVRAM within segment. $FE8_H$ Base address. All other values are reserved. Returns $FE8_H$ if read; should be written with $FE8_H$.
RC0, RC1	28, 29	r	**Reserved Control Bits** Reserved for future control expansions. Read returns 0. Must be written with 0.
IEMS	30	rw	**Internal or Emulation/External Memory Select** IEMS selects the type of the overlay memory and the size-range of overlay blocks. 0_B Internal OVRAM is selected as overlay memory. Block sizes are 2^n Bytes, n = 4-11. 1_B Emulation Memory EMEM is selected as overlay memory if not coincidently EXOMS is set. Block sizes are 2^n x 1 KB, n = 0-7. IEMS must be written with zero in production device.

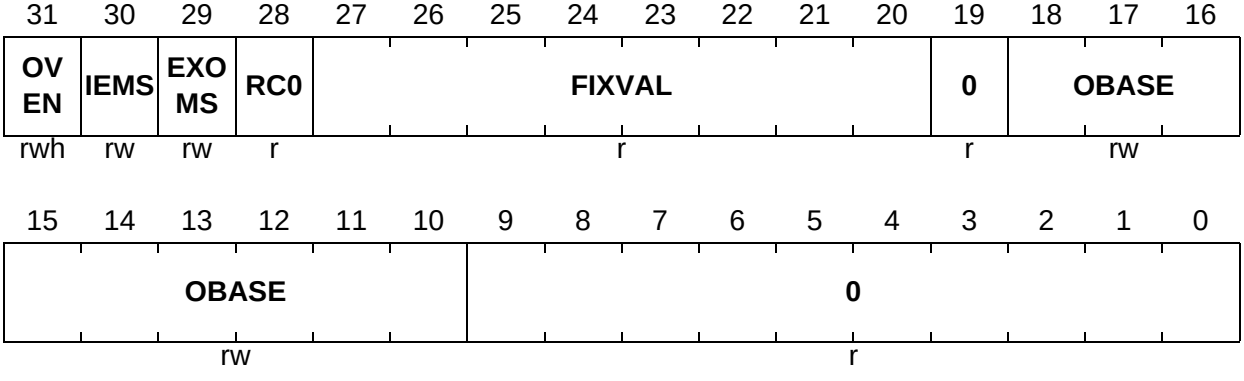
Field	Bits	Type	Description
OVEN	31	rwh	**Overlay Enabled** This bit controls whether or not the overlay function of overlay block x is enabled. 0_B Overlay function of block x is disabled. 1_B Overlay function of block x is enabled. This bit can also be changed via its shadow bit in the OCON register.
0	[3:0], [15:13]	r	**Fixed Value** Read as 0; should be written with 0.

1) The block size is determined by the mask register OMASK.

If RABRx.IEMS=1 and RABRx.EXOMS=0, the RABRx register is defined as follows. The reset value is meaningless in this case because after reset the register has always the layout defined on **Page 6-11**.

RABRx (x=0-15)
Redirected Address Base Register x

(20_H+x*C_H) **Reset Value: XXXX XXXX$_H$**



Field	Bits	Type	Description
OBASE	[18:10]	rw	**Overlay Block Base Address** This bit field holds the base address of the overlay memory block in the Emulation Memory EMEM. The largest block base address that is allowed, is ($OBASE_{max}$ - block size[1)])
FIXVAL	[27:20]	r	**Fixed Value** Base address of EMEM within segment. FF_H Base address. All other values are reserved. Returns FF_H if read; should be written with FF_H.
RC0	28	r	**Reserved Control Bit** Reserved for future control expansions. Read returns 0. Must be written with 0.
EXOMS	29	rw	**External Overlay Memory Select** If set, the external memory is used as overlay memory. 0_B Overlay memory of block x is not the external memory. 1_B Overlay memory of block x is the external memory. The block sizes are identical to EMEM blocks: 2^n x 1 KB, with n = 0-7.

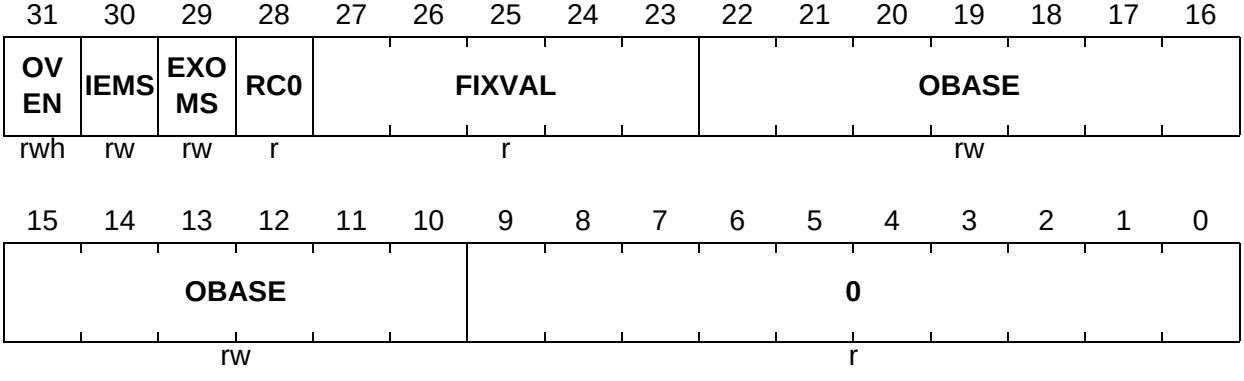
Field	Bits	Type	Description
IEMS	30	rw	**Internal or Emulation/External Memory Select** IEMS selects the type of the overlay memory and the size-range of overlay blocks. 0_B Internal OVRAM is selected as overlay memory. Block sizes are 2^n Bytes, n = 4-11. 1_B Emulation Memory EMEM is selected as overlay memory if not coincidently EXOMS is set. Block sizes are 2^n x 1 KB, n = 0-7.
OVEN	31	rwh	**Overlay Enabled** This bit controls whether or not the overlay function of overlay block x is enabled. 0_B Overlay function of block x is disabled. 1_B Overlay function of block x is enabled. This bit can also be changed via its shadow bit in the OCON register.
0	[9:0], 19	r	**Fixed Value** Read as 0; should be written with 0.

1) The block size is determined by the mask register OMASK.

If RABRx.IEMS=1 and RABRx.EXOMS=1, the RABRx register is defined as follows. The reset value is meaningless in this case because after reset the register has always the layout defined on **Page 6-11**.

RABRx (x=0-15)
Redirected Address Base Register x

(20_H+x*C_H) **Reset Value: XXXX XXXX$_H$**



Field	Bits	Type	Description
OBASE	[22:10]	rw	**Overlay Block Base Address** This bit field holds the base address of the overlay memory block in the external memory. The largest block base address that is allowed, is ($OBASE_{max}$ - block size[1)])
FIXVAL	[27:23]	r	**Fixed Value** Base address of external memory within segment. 00001_BBase address. All other values are reserved. Returns 00001_B if read; should be written with 00001_B.
RC0	28	r	**Reserved Control Bit** Reserved for future control expansions. Read returns 0. Must be written with 0.
EXOMS	29	rw	**External Overlay Memory Select** If set, the external memory is used as overlay memory. 0_B Overlay memory of block x is not the external memory. 1_B Overlay memory of block x is the external memory. The block sizes are identical to EMEM blocks: 2^n x 1 KB, with n = 0-7.

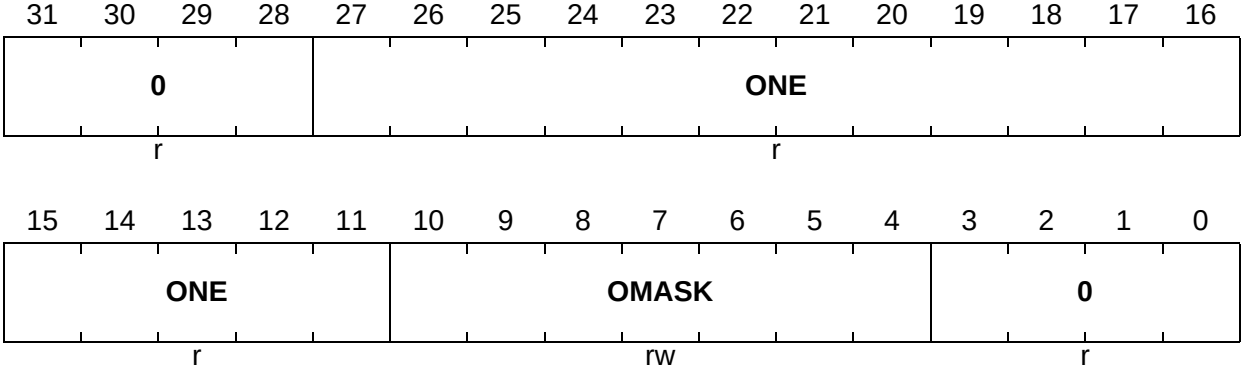
Field	Bits	Type	Description
IEMS	30	rw	**Internal or Emulation/External Memory Select** IEMS selects the type of the overlay memory and the size-range of overlay blocks. 0_B Internal OVRAM is selected as overlay memory. Block sizes are 2^n Bytes, n = 4-11. 1_B Emulation Memory EMEM is selected as overlay memory if not coincidently EXOMS is set. Block sizes are 2^n x 1 KB, n = 0-7.
OVEN	31	rwh	**Overlay Enabled** This bit controls whether or not the overlay function of overlay block x is enabled. 0_B Overlay function of block x is disabled. 1_B Overlay function of block x is enabled. This bit can also be changed via its shadow bit in the OCON register.
0	[9:0]	r	**Fixed Value** Read as 0; should be written with 0.

1) The block size is determined by the mask register OMASK.

The Overlay Mask Register OMASKx determines the size of the overlay memory block x (by the number of least significant zero's). It also determines which address bits will participate in the address compare for the block base address (all high-order one's), and in case of address match which bits are taken from RABRx (as many high order bits as defined in OMASKx for address compare) and which bits are used from the original data address as offset within the block (all low order bits related to zero values in OMASKx).

If RABRx.IEMS=0, the OMASKx register is defined as follows.

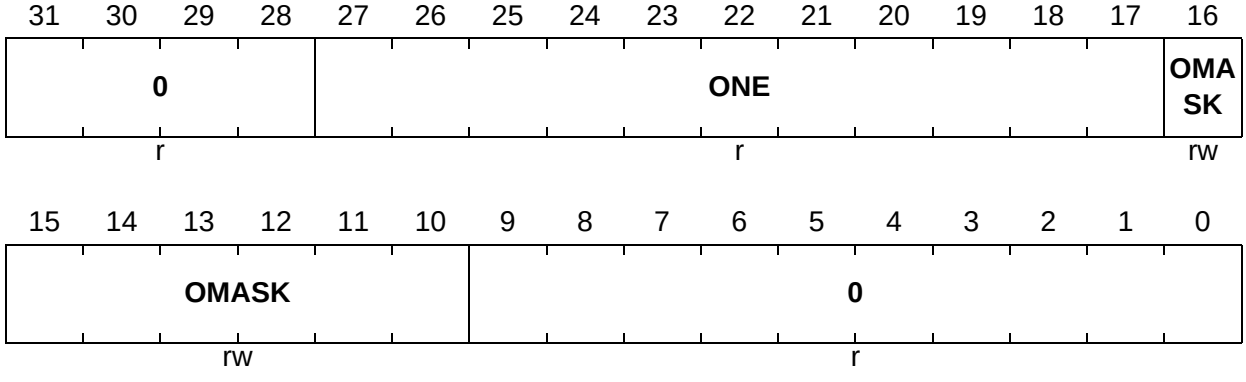
OMASKx (x=0-15)
Overlay Mask Register x **(28_H+x*C_H)** **Reset Value: 0FFF FFE0$_H$**



Field	Bits	Type	Description
OMASK	[10:4]	rw	**Overlay Address Mask** This bitfield determines the overlay block size in OVRAM and the bits used for address comparison and translation. Selectable overlay memory block sizes in OVRAM: 0000000_B 2 Kbyte 1000000_B 1 Kbyte 1100000_B 512 byte 1110000_B 256 byte 1111000_B 128 byte 1111100_B 64 byte 1111110_B 32 byte 1111111_B 16 byte “Zero” bits determine the corresponding address bits which are not used in the address comparison and thus determine the block size; corresponding final address bits are derived from the original data address. “One” bits determine the corresponding address bits which are used for the address comparison; corresponding final address bits are derived from RABRx register in case of address match. All OMASK bits located right of the most significant mask bit which is set to zero, are treated as zeros as well, independent from their actual value. They will be read back as 0.
ONE	[27:11]	r	**Fixed “1” Values** Corresponding address bits are participating in the address comparison. Corresponding final address bits are taken from RABRx.
0	[3:0], [31:28]	r	**Fixed “0” Values** Corresponding address bits are not used in the address comparison. Corresponding final address bits are taken from the original data address.

If RABRx.IEMS=1, the OMASKx register is defined as follows.

OMASKx (x=0-15)
Overlay Mask Register x **(28_H+x*C_H)** **Reset Value: 0FFF FC00$_H$**

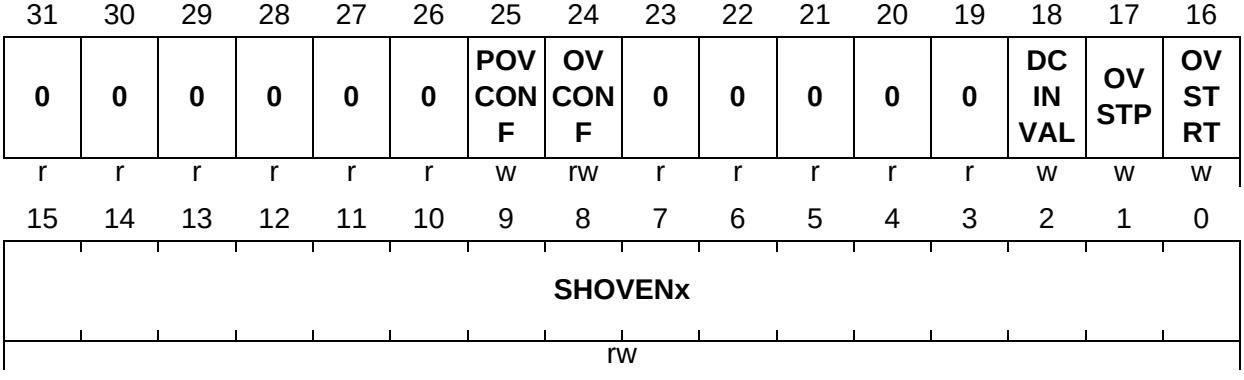


Field	Bits	Type	Description
OMASK	[16:10]	rw	**Overlay Address Mask** This bitfield determines the overlay block size in EMEM or in external memory (if EXOMS=1) and the bits used for address comparison and translation. Selectable overlay memory block sizes in EMEM: 0000000_B 128 Kbyte 1000000_B 64 Kbyte 1100000_B 32 Kbyte 1110000_B 16 Kbyte 1111000_B 8 Kbyte 1111100_B 4 Kbyte 1111110_B 2 Kbyte 1111111_B 1 Kbyte “Zero” bits determine the corresponding address bits which are not used in the address comparison and thus determine the block size; corresponding final address bits are derived from the original data address. “One” bits determine the corresponding address bits which are used for the address comparison; corresponding final address bits are derived from RABRx register in case of address match. All OMASK bits located right of the most significant mask bit which is set to zero, are treated as zeros as well, independent from their actual value. They will be read back as 0.

Field	Bits	Type	Description
ONE	[27:17]	r	**Fixed “1” Values** Corresponding address bits are participating in the address comparison. Corresponding final address bits are taken from RABR.
0	[9:0], [31:28]	r	**Fixed “0” Values** Corresponding address bits are not used in the address comparison. Corresponding final address bits are taken from the original address.

The Overlay Control Register OCON is defined as follows:

OCON
Overlay Control Register **($00E0_H$)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
SHOVENx (x=0-15)	x	rw	**Shadow Overlay Enable x** 0_B Overlay block x is disabled with next OVSTRT 1_B Overlay block x is enabled with next OVSTRT For each of the 16 overlay blocks (indicated by index x), one enable (disable) bit is provided.
OVSTRT	16	w	**Overlay Start** 0_B No action 1_B All 16 shadow overlay enable bits SHOVEN ar loaded into the related OVEN bits in RABRx registers in parallel. Related to the SHOVEN bits state, the overlay blocks are concurrently enabled or disabled. Return 0 if read.
OVSTP	17	w	**Overlay Stop** 0_B No action 1_B All 16 OVEN bits in RABRx registers are cleared in parallel independently of the SHOVEN bits and without changing the SHOVEN bits. Return 0 if read.

Field	Bits	Type	Description
DCINVAL	18	w	**Data Cache Invalidate** No function in devices without without data cache in Tricore. 0_B No action 1_B Data Cache Lines in DMI are invalidated (flushed). *Note: Per write modified cache lines are not invalidated.*[1)] Return 0 if read.
OVCONF	24	rw	**Overlay Configured** Overlay configured status bit 0_B Overlay is not configured or it has been already started by CPU 1_B Overlay block control registers are configured and ready for overlay start This bit may be used as handshake bit between a debug device (via JTAG interface and Cerberus) and the CPU.
POVCONF	25	w	**Protection Bit for OVCONF** 0_B Bit protection: Bit OVCONF remains unchanged with register OCON write 1_B OVCONF can be changed with actual write access to register OCON This bit enables OVCONF-write during OCON write. Return 0 if read.
0	[23:19], [31:26]	r	**Reserved** Read/write 0.

1) Because the data cache is a writeback cache (not a writethrough cache; therefore saving of modified data in cache has to be performed by the user) it is highly recommended to use only non-cached accesses for overlaid (redirected) accesses to the target memory (normally the Program Flash), if write accesses are involved.

7 TC1797BootROM Content

The TC1797 BootROM contains so-called Startup Software (SSW) which is the first software executed after chip reset. The Startup Software contains procedures to initialize the device depending on one or more from the following:

- values applied to external (configuration-) pins;
- the type of event which has triggered the SSW-execution (the last reset event);
- information stored into Flash Configuration Sector.

After SSW the TC1797 executes user code out of an on-chip or off-chip program memory. The initial code source can be selected via hardware configuration (i.e. defined levels on specific pins:

- **Internal Start** Mode: executes code out of the on-chip program Flash
- **External Start** Mode: executes code out of an off-chip (External) memory
- **Bootstrap Loading**Mode: executes code out of the on-chip Instruction Scratchpad Memory SPRAM (PMI). This code is downloaded beforehand via a selectable serial interface

7.1 Start-up Mode Selection

After any device start the currently valid start-up configuration is indicated in bitfield HWCFG of register SCU_STSTAT. **Table 7-1** summarizes the defined start up modes.

The value in this bit field can be updated in different ways:

1. with values latched at the configuration pins P0[7:0] upon the rising reset-edge
   This happens when SCU_STSTAT.LUDIS=0, therefore:
   a) after any system reset
   b) after other resets - depending on LUDIS bit.
2. by executing the following software sequence (using register SCU_SWRSTCON, described in SCU Chapter):
   a) write respective configuration value (refer to **Table 7-1**) to bitfield SWCFG;
   b) set Software Boot Configuration bit: SWBOOT = 1;
   c) trigger a software reset by activating Software Reset Request: SWRSTREQ = 1.

Table 7-1 Startup Modes in TC1797

HWCFG[7:0]	Startup Mode	Pins used
11xxxxxx$_B$	Internal Start from Flash	2
011xxxxx$_B$	Reserved	3
010xxxx0$_B$	Bootstrap Loader Mode, Generic Bootloader at CAN pins	4
10101xx0$_B$	Bootstrap Loader Mode, ASC Bootloader	6
10100xx0$_B$	Alternate Boot Mode, ASC Bootloader on fail	6
1011xxxx$_B$	Alternate Boot Mode, Generic Bootloader at CAN pins on fail	4
1000xxxx$_B$	Reserved	4
1001C1A0$_B$[1)]	External Alternate Boot Modes	8
0010E1A0$_B$[1)]	External Start Modes	8
0011xxxx$_B$	Reserved	4
000xxxxx$_B$	Reserved	3

1) C bit - Pins used for bootloading: 0 - ASC (ASC bootloader);1 - CAN (Generic bootloader)
A bit - EBU Arbitration Mode: 0 - Participant; 1 - Arbiter (previously referred as “External Master”)
E bit - EBU Configuration: 0 - Default; 1 - Automatic

The TC1797 SSW supports an internal “Reset Configuration Updated” flag which is used in Bootstrap Loader modes (refer to **Chapter 7.4**). This flag is installed as follows:

- if the last reset is a software-reset with software-configuration, e.g.
  ((SCU_RSTSTAT.SW=1) AND (SCU_SWRSTCON.SWBOOT=1))
  - the flag is set
- if the last reset is NOT of the above type
  - the flag is set to the inverted value of SCU_STSTAT.LUDIS bit

7.2 Internal Start

This is the basic TC1797 type of operation in which the user code is started out of the Internal Flash Memory.

The User Start Address STADD is set to the beginning of Internal Flash0 Memory Module at address A000’0000$_H$.

Note: Because internal start mode is expected to be the configuration used in most cases, this mode can be selected by pulling high just 2 pins.

7.3 External Start

In TC1797, several External Start modes are supported providing different EBU settings (refer to **Table 10-1**).

The User Start Address STADD is set to address A100'0000$_H$ in external EBU space.

7.3.1 EBU Configuration Settings

Several External Start options are supported by the SSW in TC1797 regarding the two basic EBU (External Bus Unit) settings:

EBU Configuration

- Automatic
  In such a case, before any other activity with EBU the SSW first sets SCU_STSTAT.EXTBEN indirectly by writing SCU_SYSCON.SETEXTBEN=1. This triggers an external fetch of the Boot Configuration Value. This fetch is performed as access to the dedicated address 000004$_H$ of a standard asynchronous memory device attached to the $\overline{CS0}$ chip-select line of the EBU.
  After this operation is performed, the SSW checks either valid data has been fetched. This check will find EBU_EXTBOOT.CFGERR=1 (Configuration Fetch Error), if FFFF$_H$ has been returned for bits[15:0] of the Boot Configuration Value upon the automatic fetch-access. Such a result shows that no valid Configuration word is available at the dedicated location - for example the location is in a Flash which is not (properly) programmed.
  The SSW reaction on invalid auto-configuration word is different upon the current startup mode, as follows:
  i) in External Start Mode - the SSW terminates with an error, as far as a successful start can not be guaranteed due to unknown configuration, while the user expects this configuration should be automatically set to correct values;
  ii) in External Alternate Boot Mode - the situation is considered as an error in ABM Header. Therefore the SSW does not continue with further checks but directly jumps to a Bootloader as selected by the startup configuration.
- Default
  In such a case the SSW writes standard values into EBU control registers
  – EBU_ADDRSEL0:=A000 0005$_H$;
  – EBU_MODCON_GLOBALCS:=0001$_B$.

EBU Arbitration Mode

- Arbiter - the SSW configures EBU_MODCON.ARBMODE:=01$_B$
- Participant - the SSW configures EBU_MODCON.ARBMODE:=10$_B$

Once the SSW writes something - in any case different from 00$_B$ - in ARBMODE bitfield, the EBU is enabled and can start accessing an External Memory.

7.4 Bootstrap Loading

Different Bootstrap Loader routines are used in these modes to download code/data into the Instruction Scratchpad Memory SPRAM (PMI)

The selected Bootstrap Loader is executed only if the SSW-flag “Reset Configuration Updated” is set (refer to **Chapter 7.1**). This is to avoid multiple executions of the Bootstrap Loader and to start directly the code already downloaded after some - intended to be “application only” - reset events, for example after a Watchdog Timer reset which has been configured as application reset.

The supported Bootloader selections are:

- ASC Bootloader - ASC communication protocol via ASC pins, as follows:
  - receive pin RxD at Pin 0 Port 5 (P5.0)
  - transmit pin TxD at Pin 1 Port 5 (P5.1)
- Generic Bootloader via CAN pins - the communication protocol is automatically selected by the SSW between ASC and CAN, the pins are used as follows:
  - receive pin RxD at Pin 8 Port 6 (P6.8)
  - transmit pin TxD at Pin 9 Port 6 (P6.9)

After downloading (in case) the code, the User Start Address STADD is set to the beginning of PMI Scratchpad RAM at D400’0000$_H$.

7.4.1 Common Procedures for all Bootloaders

The first such a common procedure is to reconfigure the clock system in case the last reset is a Power-on. This reconfiguration switches from the initial PLL Freerunning mode (VCO base frequency) to Prescaler Mode with F_{FPI}:F_{OSC}=1:2.

Therefore an external crystal must be connected between XTAL1/XTAL2 pins if a Bootloader mode will be selected upon Power-on. The FPI-peripherals (including MultiCAN and ASC modules) will run then at half the crystal frequency.

This 1:2 relation must be taken into account when selecting the host speed for downloading. For example, when using the MultiCAN bootstrap loader FOSC should be greater or equal to 20 MHz for a baud rate of 1 Mbit/s.

Attention: This clock-switch to external oscillator upon Power-on is overwritten by a switch back to PLL freerunning mode at the end of SSW .
Therefore if it is desired after downloading to continue communication with the same baudrate, the user code must first reinstall:
- PLL prescaler mode (SCU_PLLCON0.VCOBYP:=0) and
- K1 divider 1:1 (SCU_PLLCON1.KDIV:=00$_H$).

Upon any other reset but not power-on, the clock configuration - respectively the system frequency - remains the same as previously selected.

Next, depending on the Bootloader-type currently configured in HWCFG a pin is selected as receive-data input (RxD) which will be evaluated on the following step. Note, that no

further pin-configuration is done here - e.g. no assignment to a specific (ASC/CAN) module functionality - but only the pin input-value is directly checked in this procedure.

The procedure waits to receive a low-level pulse at RxD and measures its duration - the time between the falling and rising edges.

Then, the SSW checks the startup configuration in HWCFG:

- upon ASC Bootloader Mode - a jump to the **ASC Bootstrap Loader** routine (**Chapter 7.4.2**) is directly executed
- upon Generic Bootloader Mode - the type of interface (ASC/CAN) currently used must be detected by the SSW. For this the RxD pin is checked until:
  – low level is found there - then a jump to **CAN Bootstrap Loader**routine (**Chapter 7.4.3**) will be executed; **OR**
  – no edge is found for a time 6 time longer than the value in BL_meas - then **ASC Bootstrap Loader** routine will be executed.

This interface-detection procedure is based on the following principles:

- an ASC-Bootloader Host sends one only start Byte and then waits for a response from the target system
- a CAN-Bootloader Host sends a complete frame, whereas no more than 5 consecutive bits can be sent having equal logical levels - i.e. after two consecutive edges for a given time dT, in any case another edge must follow within the next time-frame of 6*dT.

7.4.2 ASC Bootstrap Loader

The ASC Bootloading routine implements the following steps:

- RxD/TxD pins configuration is done in accordance to the TC1797 definitions, as well as depending either the routine is invoked upon “ASC Bootloader”-startup mode (ASC-only pins are used) or following an ASC-protocol detection upon “Generic Bootloader”-mode (CAN/ASC-shared pins are used but configured to ASC module)
- baudrate calculation is done based on the value already captured
- ASC0 is initialized (without enabling the receiver) to the baudrate as determined, 8 data and 1 stop bit
- acknowledge byte $D5_H$ is sent to the host indicating the device is ready to accept a data transfer
- after the acknowledge byte is transmitted, the receiver is enabled
- the bootloader enters a loop waiting to receive exactly 128 bytes which are stored as 32 words in PMI Scratchpad RAM starting from address $D400'0000_H$

Once 128 bytes are received, the SSW starts the user code from address $D400'0000_H$.

7.4.3 CAN Bootstrap Loader

The CAN bootstrap loader transfers program code/data via node 0 of the MultiCAN module into the PMI Scratchpad RAM. Data is transferred from the external host to the TC1797 using eight-byte data frames. The number of data frames to be received is programmable and determined by the 16-bit data message count value DMSGC.

The communication between TC1797 and external host is based on the following three CAN standard frames:

- Initialization frame - sent by the external host to the TC1797
- Acknowledge frame - sent by the TC1797 to the external host
- Data frame(s) - sent by the external host to the TC1797

The initialization frame is used in the TC1797 for baud rate detection. After a successful baud rate detection is reported to the external host by sending the acknowledge frame, data is transmitted using data frames.

Initialization Phase

The first task is to determine the CAN baud rate at which the external host is communicating. This task requires the external host to send initialization frames continuously to the TC1797. The first two data bytes of the initialization frame include a 2-byte baud rate detection pattern (5555_H), an 11-bit (2-byte) identifier ACKID for the acknowledge frame, a 16-bit data message count value DMSGC, and an 11-bit (2-byte) identifier DMSGID to be used by the data frame(s).

The CAN baud rate is determined by analyzing the received baud rate detection pattern (5555_H) and the baud rate registers of the MultiCAN module are set accordingly. The TC1797 is now ready to receive CAN frames with the baud rate of the external host.

Acknowledge Phase

In the acknowledge phase, the bootstrap loader waits until it receives the next correctly recognized initialization frame from the external host, and acknowledges this frame by generating a dominant bit in its ACK slot. Afterwards, the bootstrap loader transmits an acknowledge frame back to the external host, indicating that it is now ready to receive data frames. The acknowledge frame uses the message identifier ACKID that has been received with the initialization frame.

Data Transmission Phase

In the data transmission phase, data frames are sent by the external host and received by the TC1797. The data frames use the 11-bit data message identifier DMSGID that has been sent with the initialization frame. Eight data bytes are transmitted with each data frame. The first data byte is stored in PMI Scratchpad RAM starting from address $D400'0000_H$. Consecutive data bytes are stored at incrementing addresses.

Both communication partners evaluate the data message count DMSGC until the requested number of CAN data frames has been transmitted.

After the reception of the last CAN data frame, the SSW starts the user code from address D400'0000$_H$.

7.5 Alternate Boot Modes

In these modes, program code is started from user-defined address but only if at least one of the two check-conditions is satisfied. If both the conditions are false - a Bootstrap Loader routine is started to download the code into the device, this code is afterwards started by the SSW.

Check-condition true means a correct program code is available at the defined location. The check condition itself is evaluated by calculating CRC sum over the content of a defined memory range. All the information needed for the SSW to handle ABM startup modes is collected into the so-called Headers. The checks are performed according to two Headers defined inside the Internal Flash memory of the device for Internal ABM as well as two Headers in External EBU address space to support External ABM modes.

Several Alternate Boot Modes (ABM) are available in TC1797, the differences are in:

- where the Header and the user code are located: in Internal Flash or in External Memory;
- which communication channel is used for code downloading upon an error in check-condition: ASC or CAN.

The SSW flow in these modes is:

- evaluate the startup configuration to decide where are the ABM-Headers located
- check the Headers - refer to the description in **Chapter 10.1.4.1** - and react accordingly:
  - if the check is OK for one of the Headers - set the User Start Address STADD to the respective value from this correct header (STADABMx) and continue
  - if both the Header-checks fail - start a Bootloader (ASC/Generic) corresponding to the startup configuration. After downloading the code, the User Start Address STADD is set to the beginning of PMI Scratchpad RAM at D400'0000$_H$.

7.5.1 Header Check in Alternate Boot Modes

The Alternate Boot Modes (ABMs) are intended to start program code already available at arbitrary user-defined address if a check-condition is satisfied. If the check-condition fails - a Bootstrap Loader routine is invoked in accordance to the current startup mode.

The address of the code to be started together with all information needed to verify the check-condition are contained in dedicated memory areas named ABM Headers. In any Alternate Boot Mode of TC1797 two such Headers are defined - Header 0 and Header 1 (referred as ABM.HD0 and ABM.HD1), respectively user code can be started from up to two different start addresses.

The ABM Headers can be located in Internal Flash memory of the device, whereas the locations are defined:

- Header 0 - Base address A001'FFE0$_H$; End address A001'FFFF$_H$
- Header 1 - Base address A000'FFE0$_H$; End address A000'FFFF$_H$

In TC1797 External starts are also supported as Startup Configurations, respectively so-call External Alternate Boot Modes are defined (refer to **Table 10-1**). In this latest case, the ABM Headers are located in External EBU address space as follows:

- Header 0, External - Base address A100'FFE0$_H$; End address A100'FFFF$_H$
- Header 1, External - Base address A108'FFE0$_H$; End address A108'FFFF$_H$

*Note: In all the External ABMs, the SSW first tries to fetch the configuration word (refer to **Chapter 7.3.1**) as check for availability of an external memory. If this fetch fails to return valid data (indicated by EBU_EXTBOOT.CFGERR=1) - the situation is considered as error in the ABM Header and a bootloader is started.*

Any of the Headers is 32 Bytes long, containing information in accordance to **Table 10-2**.

Table 7-2 ABM Headers Structure

Offset Addr.	Size Byte	Field Name	Description
00$_H$	4	STADABM	User Code Start Address
04$_H$	4	ABMHDID	ABM Header ID = DEAD BEEF$_H$ (Confirmation code)
08$_H$	4	ChkStart	Memory Range to be checked - Start Address
0C$_H$	4	ChkEnd	Memory Range to be checked - End Address
10$_H$	4	CRCrange	Check Result for the Memory Range
14$_H$	4	$\overline{\text{CRCrange}}$	Inverted Check Result for the Memory Range
18$_H$	4	CRChead	Check Result for the ABM Header (offset 00$_H$..17$_H$)
1C$_H$	4	$\overline{\text{CRChead}}$	Inverted Check Result for the ABM Header

In any Alternate Boot Mode always the Header 0 is checked first, and if failed - Header 1 is checked next. Where the check gives OK, the full start address is taken from the respective Header field STADABMx (index x=0,1 for Header 0,1).

If check fails for both the Headers, the further execution depends on the startup mode currently configured in HWCFG: a jump either to ASC- or to Generic- (CAN pins) Bootloader is performed by the SSW.

The validation procedure executes CRC calculation based on a 32-bit polynomial:

$$f(x) = x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1 \quad (7.1)$$

This calculation is performed by the SSW using the Memory Checker Module in TC1797.

The complete check-procedure for a Header consists of the following steps:

1. check the ABM Header ID at offsets $04_H..07_H$ (the correct values are given in **Table 10-2**):
   a) if OK - continue with 2.
   b) if Not - exit the check-procedure for this Header with Error.
2. calculate the CRC of the first 24 Bytes from the ABM Header - process the fields STADABM...CRCRange at offsets $00_H...17_H$
   a) compare the result with the CRChead value (offset 18_H)
      - if OK - continue with 2.b)
      - if Not - exit the check-procedure for this Header with Error.
   b) inverse the result value and compare with $\overline{\text{CRChead}}$ (offset $1C_H$)
      - if OK - continue with 3.
      - if Not - exit the check-procedure for this Header with Error.
3. calculate the CRC over the memory address range ChkStart...ChkEnd (start- and end- addresses taken from offsets 08_H and $0C_H$ respectively)
   a) compare the result with the CRCrange value (offset 10_H)
      - if OK - continue with 3.b)
      - if Not - exit the check-procedure for this Header with Error.
   b) inverse the result value and compare with $\overline{\text{CRCrange}}$ (offset 14_H)
      - if Not - exit the check-procedure for this Header with Error.
      - if OK - the check-procedure PASSed for this Header !

According to the result returned from this check procedure, the startup software continues either with user-code start from STADABMx address (on success), or with checking the second header (on error in the first one), or with a Bootstrap loader (on error in the second header).

7.6 Startup Errors Handling

There are a number of check-points during the Startup Software execution where Errors can be raised. The processing upon an Error is as follows:

- an Exit-code according to the error is stored into CBS_COMDATA register - refer to **Table 7-3**
- the Watchdog Timer Reset is configured to class 3
- all code- and data- fetches from Flash are disabled
- access to the Flash Config sector is disabled
- startup protection is activated, which activation starts the Watchdog Timer
- the debug interface is unlocked
- an endless loop is executed (jump to itself)

As far as the Watchdog Timer is already enabled, the endless loop is aborted by a WDT reset which triggers a new SSW-execution. If this new startup fails again, the following error-processing will lead to the same endless loop. Respectively, a second WDT reset will occur being already a locked reset, which can be aborted only by a next power-on sequence.

Table 7-3 Errors reported by the TC1797 SSW

Coding in d12/COMDATA	Description
00000001_H	Bootcode wrongly called after exiting startup mode
00000002_H	Flash error during rampup
00000003_H	Error in Flash Configuration sector
00000004_H	Invalid Startup mode selected
00000008_H	MultiCAN module not available but CAN Bootloader selected
00000009_H	ASC module not available but ASC Bootloader selected
$0000004x_H$	Trap of Class x (0..7) raised during SSW

7.7 Notes and usage hints

The information below is provided to help the customer to use properly the functionality of TC1797 startup software.

7.7.1 Conditions upon user code start

After SSW, the user code execution starts upon the following basic conditions within TC1797:

- system clock
  - after power-on/PORST: PLL in freerunning mode, 16.66 MHz system frequency (nominal)

Note: When in Bootloader mode - keep in mind the Attention-note at the beginning of in **Chapter 7.4.1** *(***Common Procedures for all Bootloaders***)!*

  - otherwise: as previously configured - no change due to the bootcode
- interrupts and traps
  - all maskable interrupts are disabled by ICR.IE=0
  - all NMI-traps are disabled in SCU_TRAPDIS
- Watchdog-Timer - running in time-out mode

7.7.2 RAMs Handling

Two aspect must be taken into account by the user:

LDRAM Locations overwritten by SSW

The startup procedure overwrites the first 16 Bytes from LDRAM - address D000'0000$_H$ to D000'000F$_H$.

Using Memory Control

No RAM initialization is performed by the Startup Software in TC1797.

The user should take care, if Parity Control will be enabled for a RAM module, to assure a correct initial content of this memory.

7.7.3 Influencing the next SSW-execution

By writing 1 to SYSCON.SETLUDIS (no protection), the user software will prevent automatic update of SCU_STSTAT.HWCFG upon the next application reset.

If such setting is active when a Bootstrap Loader mode is configured in STSTAT.HWCFG bitfield, after the next application reset the Bootloader routine will not be executed but directly the application-code will be started from SPRAM.

The LUDIS=1 setting will have effect until a system/power-on reset which will force the SYSCON register to its reset value with LUDIS=0. Therefore, upon system/power-on

reset the startup HW-configuration (SCU_STSTAT.HWCFG) is always updated from pins.

8 Memory Maps

This chapter gives an overview of the TC1797 memory map, and describes the address locations and access possibilities for the units, memories, and reserved areas as “seen” from the two different on-chip buses’ point of view.

The TC1797 has the following memories

- Program Memory Unit (PMU) with
  - 4 Mbyte of Program Flash Memory (PFLASH)
  - 64 Kbyte of Data Flash Memory (DFLASH)
  - 16 Kbyte of Boot ROM (BROM)
  - 8 KB Overlay Memory (OVRAM)
- Program Memory Interface (PMI)
  - 40 Kbyte of Scratch-Pad RAM (SPRAM[1)])
  - 16 Kbyte of Instruction Cache (ICACHE[1)])
- Data Memory Interface (DMI)
  - 128 Kbyte of Local Data RAM (LDRAM[2)])
  - 4 Kbyte of Data Cache (DCACHE[2)])
- PCP memory
  - 32 Kbyte of PCP Code Memory (CMEM)
  - 16 Kbyte of PCP Data Memory (PRAM)

Furthermore, the TC1797 has two on-chip buses:

- System Peripheral Bus (SPB)
- Local Memory Bus (LMB)

1) Up to 16 Kbyte out of the 40 Kbyte SPRAM can be configured as ICache. The supported SPRAM / ICache configurations are described in the CPU sub-chapter: ´Program Memory Interface (PMI)‘.

2) Up to 4 Kbyte out of the 128 Kbyte LDRAM can be configured as DCache. The supported LDRAM / DCache configurations are described in the CPU sub-chapter: ´Data Memory Interface (DMI)‘.

8.1 What is new

The target for the AudoFuture devices memory map is to keep it compatible to AudoNG wherever it is possible. This means to keep memory space/flash segment start addresses and peripheral control register address spaces where they where mapped to in AudoNG.

Major differences of the TC1797 Memory Map compared to AudoNG:

- Address map is adapted to the peripheral set of the products (peripherals where added/removed, number of ports is adapted).
- Target was to keep the start address where possible of all common modules, SRAMs and Flash segments
- Added PMI Byte Read/Write access in **Table 8-5**
- Added Double-Word support for PCP-CRAM and PCP-PRAM accesses. Means that CPU has also 64 bit access to the PCP memories (**Table 8-5**)
- Moved SCU address map inside Segment 15 as SCU requires now 2x256Byte
- Moved ADC and FADC address maps inside Segment 15 as these modules require more 256 byte slices now.
- Adapted memory and flash sizes (SPRAM, LDRAM, CRAM, PRAM, PFlash, DFlash)
- Notes for the Instruction / Data Cache configurations where added.
- OVRAM is moved from Segment 12 to Segment 8 and Segment 10.
- Emulation Device Memory was moved from xFF2-0000 -> xFF5FFF to xFF0-0000 -> xFF3-FFFF
- An PMI memory mirror image was added (SPRAM + configurable ICACHE) to C000. Added map of the mirrored PMI memory image to segment E800.
- Removed Boot ROM Address Space in segment D as it is not necessary any more.
- Added Online Data Acquisition Address Space (OLDA) to segment 8 and A with a footnote that it is controlled via PMU0_OVRCON.OLDAEN bit.
- MultiCAN module address space increased from 8KB -> 16KB
- Changed the SPB view of segment C and D in a way that it is now fully transparent (write accesses to reserved addresses result in LMBBE, read to SPBBE & LMBBE)
- Added Overlay Control Module (OVC) with 256 byte to segment 15 (**Table 8-3**) as this functionality is moved now to DMI.

8.2 How to Read the Address Maps

The bus-specific address maps describe how the different bus master devices react on accesses to on-chip memories and modules, and which address ranges are valid or invalid for the corresponding buses.

The FPI Bus address map shows the system addresses from the point of view of the SPB master agents. SPB master agents are PCP2 and DMA[1)].

The LMB address map shows the system addresses from the point of view of the LMB master agents. LMB master agents are PMI, DMI and DMA[1)].

The LFI is a bi-directional bridge between LMB and SPB and therefore not mentioned here as LMB or SPB master in the Address Map. The LFI includes an SPB to LMB address translation mechanism. The SPB to LMB Bus Address Translation Table can be found in the chapter: ´On-Chip System Buses and Bus Bridges´ / ´Local Memory to FPI Bus Interface (LFI Bridge)´.

1) DMA including: DMA Move Engines and module connected to the DMA Peripheral Interface like MLI modules.

Table 8-1 defines the acronyms and other terms that are used in the address maps (Table 8-2 to Table 8-4).

Table 8-1 Definition of Acronyms and Terms

Term	Description
…BE	Means “Bus error” generation.
…BET	Means “Bus error & trap” generation.
SPBBE	A bus access is terminated with a bus error on the SPB.
SPBBET	A bus access is terminated with a bus error on the SPB and a DSE trap (read access) or DAE trap (write access).
LMBBE	A bus access is terminated with a bus error on the LMB.
LMBBET	A bus access is terminated with a bus error on the LMB and a DSE trap (read access) or DAE trap (write access).
access	A bus access is allowed and is executed.
ignore	A bus access is ignored and is not executed. No bus error is generated.
trap	A DSE trap (read access) or DAE trap (write access) is generated.
32	Only 32-bit word bus accesses are permitted to that register/address range.
nE	A bus access generates no bus error, although the bus access points to an undefined address or address range. This is valid e.g. for CPU accesses (MTCR/MFCR) to undefined addresses in the CSFR range.

8.3 Contents of the Segments

This section summarizes the contents of the segments.

Segments 0-7

These segments are reserved segments in the TC1797.

Segment 8

From the SPB point of view (PCP), this memory segment allows accesses to all PMU memories (PFLASH, DFLASH, EBU, BROM, TROM and OVRAM).

From the LMB point of view (CPU-PMI[1)2)], CPU-DMI[1)2)], DMA including Cerberus and MLI), this memory segment allows cached accesses to all PMU memories (PFLASH, DFLASH, EBU, BROM, TROM and OVRAM).

Segment 9

This memory segment is reserved in the TC1797.

Segment 10

From the SPB point of view (PCP), this memory segment allows non-cached accesses to all PMU memories (PFLASH, DFLASH, BROM, TROM and OVRAM).

From the LMB point of view (CPU-PMI, CPU-DMI, DMA including Cerberus and MLI), this memory segment allows non-cached accesses to all PMU memories (PFLASH, DFLASH, BROM, TROM and OVRAM).

From the DMA point of view, Move Engine, Cerberus and MLI accesses to this segment are processed by the DMA LMB master interface on the LMB Bus.

Segment 11

This memory segment is reserved in the TC1797.

Segment 12

From the SPB point of view (PCP), this memory segment allows non-cached accesses to the PMI scratch-pad RAM (SPRAM).

1) CPU access to OLDA address space via segment 8 (cached) results in LMB Bus Error Trap independent of the PMU0_OVRCON.OLDAEN bit setting.

2) For CPU accesses to segment 8 the specific PMI and DMI features for segment 8 accesses have to be taken into account.

From the LMB point of view (CPU-PMI, CPU-DMI, DMA including. Cerberus and MLI), this memory segment allows non-cached accesses to the PMI scratch-pad RAM (SPRAM).

From the DMA point of view, Move Engine, Cerberus and MLI accesses to this segment are processed by the DMA LMB master interface on the LMB Bus.

Segment 13

From the SPB point of view (PCP), this memory segment allows non-cached accesses to the PMI scratch-pad RAM (SPRAM) and the DMI memory (LDRAM).

From the LMB point of view (CPU-PMI, CPU-DMI, DMA including. Cerberus and MLI), this memory segment allows non-cached accesses to the PMI scratch-pad RAM (SPRAM) and the DMI memory (LDRAM) as well as read access to the boot ROM and test ROM (BROM and TROM).

From the DMA point of view, Move Engine, Cerberus and MLI accesses to this segment are processed by the DMA LMB master interface on the LMB Bus.

Segment 14

From the SPB point of view (PCP, DMA including. Cerberus and MLI), this memory segment allows accesses the DMI Local Data RAM (LDRAM), and the PMI scratch-pad RAM (SPRAM). All accesses to this segment will be translated by the LFI into Segment 12 and Segment 13 accesses. The detailed SPB to LMB Bus Address Translation is described in the Chapter: Local Memory to FPI Bus Interface (LFI).

From the CPU point of view (PMI and DMI), this memory segment is reserved in the TC1797.

From the DMA point of view, Move Engine, Cerberus and MLI accesses to this segment are processed by the DMA FPI master interface on the SPB Bus.

Segment 15

From the SPB point of view (PCP, DMA, Cerberus and MLI), this memory segment allows accesses to all SFRs, CSFRs, the PCP memories and the MLI transfer windows.

From the CPU point of view (PMI and DMI), this memory segment allows accesses to all SFRs, CSFRs, the PCP memories, and the MLI transfer windows.

From the DMA point of view, Move Engine, Cerberus and MLI accesses to this segment are processed by the DMA FPI master interface on the SPB Bus.

8.4 Address Map of the FPI Bus System

This chapter describes the system address map from FPI Bus (SPB) point of view.

8.4.1 Segments 0 to 14

Table 8-2 shows the address map of segments 0 to 14 as it is seen from the SPB bus masters PCP, DMA and OCDS.

Table 8-2 SPB Address Map of Segment 0 to 14

Segment	Address Range	Size	Description	Access Type	
				Read	Write[1]
0-7	$0000\ 0000_H$ - $0000\ 0007_H$	8 byte	Reserved (virtual address space)	MPN trap	MPN trap
	$0000\ 0008_H$ - $7FFF\ FFFF_H$	8 × 256 Mbyte		SPBBE	SPBBE
8	$8000\ 0000_H$ - $801F\ FFFF_H$	2 Mbyte	Program Flash 0 (PFLASH0)	access	access[1]
	$8020\ 0000_H$ - $803F\ FFFF_H$	2 Mbyte	Program Flash 1 (PFLASH1)	access	access[1]
	$8040\ 0000_H$ - $807F\ FFFF_H$	4 Mbyte	Reserved	LMBBE & SPBBE	LMBBE
	$8080\ 0000_H$ - $8FDF\ FFFF_H$	246 Mbyte	External EBU Space	access	access
	$8FE0\ 0000_H$ - $8FE0\ 7FFF_H$	32 Kbyte	Data Flash (DFLASH) Bank 0	access	access[1]
	$8FE0\ 8000_H$ - $8FE0\ FFFF_H$	32 Kbyte	Reserved	LMBBE & SPBBE	LMBBE
	$8FE1\ 0000_H$ - $8FE1\ 7FFF_H$	32 Kbyte	Data Flash (DFLASH) Bank 1	access	access[1]
	$8FE1\ 8000_H$ - $8FE1\ FFFF_H$	32 Kbyte	Reserved	LMBBE & SPBBE	LMBBE
	$8FE2\ 0000_H$ - $8FE6\ FFFF_H$	–	Reserved	LMBBE & SPBBE	LMBBE
	$8FE7\ 0000_H$ - $8FE7\ 7FFF_H$	32 Kbyte	Online Data Acquisition (OLDA)	LMBBE & SPBBE	access[2] / LMBBE & SPBBE
	$8FE7\ 8000_H$ - $8FE7\ FFFF_H$	32 Kbyte	Reserved	LMBBE & SPBBE	LMBBE

Table 8-2 SPB Address Map of Segment 0 to 14 (cont'd)

Segment	Address Range	Size	Description	Access Type	
				Read	Write[1]
	$8FE8\ 0000_H$ - $8FE8\ 1FFF_H$	8 Kbyte	Overlay memory (OVRAM)	access	access
	$8FE8\ 2000_H$ - $8FEF\ FFFF_H$	–	Reserved	LMBBE & SPBBE	LMBBE
	$8FF0\ 0000_H$ - $8FF7\ FFFF_H$	512 Kbyte	Reserved for TC1797 emulation device memory	SPBBE	SPBBE
	$8FF8\ 0000_H$ - $8FFF\ BFFF_H$	–	Reserved	SPBBE	SPBBE
	$8FFF\ C000_H$ - $8FFF\ FFFF_H$	16 Kbyte	Boot ROM (BROM)	access	SPBBE
9	$9000\ 0000_H$ - $9FFF\ FFFF_H$	256 Mbyte	Reserved	SPBBE	SPBBE
10	$A000\ 0000_H$ - $A01F\ FFFF_H$	2 Mbyte	Program Flash 0 (PFLASH0)	access	access[1]
	$A020\ 0000_H$ - $A03F\ FFFF_H$	2 Mbyte	Program Flash 1 (PFLASH1)	access	access[1]
	$A040\ 0000_H$ - $A07F\ FFFF_H$	4 Mbyte	Reserved	LMBBE & SPBBE	LMBBE
	$A080\ 0000_H$ - $AFDF\ FFFF_H$	246 Mbyte	External EBU Space	access	access
	$AFE0\ 0000_H$ - $AFE0\ 7FFF_H$	32 Kbyte	Data Flash (DFLASH) Bank 0	access	access[1]
	$AFE0\ 8000_H$ - $AFE0\ FFFF_H$	32 Kbyte	Reserved	LMBBE & SPBBE	LMBBE
	$AFE1\ 0000_H$ - $AFE1\ 7FFF_H$	32 Kbyte	Data Flash (DFLASH) Bank 1	access	access[1]
	$AFE1\ 8000_H$ - $AFE1\ FFFF_H$	32 Kbyte	Reserved	LMBBE & SPBBE	LMBBE
	$AFE2\ 0000_H$ - $AFE6\ FFFF_H$	–	Reserved	LMBBE & SPBBE	LMBBE
	$AFE7\ 0000_H$ - $AFE7\ 7FFF_H$	32 Kbyte	Online Data Acquisition (OLDA)	LMBBE & SPBBE	access[2] / LMBBE & SPBBE

Table 8-2 SPB Address Map of Segment 0 to 14 (cont'd)

Segment	Address Range	Size	Description	Access Type	
				Read	Write[1]
	AFE7 8000$_H$ - AFE7 FFFF$_H$	32 Kbyte	Reserved	LMBBE & SPBBE	LMBBE
	AFE8 0000$_H$ - AFE8 1FFF$_H$	8 Kbyte	Overlay memory (OVRAM)	access	access
	AFE8 2000$_H$ - AFEF FFFF$_H$	–	Reserved	LMBBE & SPBBE	LMBBE
	AFF0 0000$_H$ - AFF7 FFFF$_H$	512 Kbyte	Reserved for TC1797 emulation device memory	LMBBE & SPBBE	LMBBE
	AFF8 0000$_H$ - AFFF BFFF$_H$	–	Reserved	LMBBE & SPBBE	LMBBE
	AFFF C000$_H$ - AFFF FFFF$_H$	16 Kbyte	Boot ROM (BROM)	access	LMBBE
11	B000 0000$_H$ - BFFF FFFF$_H$	256 Mbyte	Reserved	SPBBE	SPBBE
12	C000 0000$_H$ - C000 5FFF$_H$	24 Kbyte	PMI Scratch-Pad RAM (SPRAM)	access	access
	C000 6000$_H$ - C000 7FFF$_H$	8 Kbyte		access[3]	access[3]
	C000 8000$_H$ - C000 8FFF$_H$	4 Kbyte		access[4]	access[4]
	C000 9000$_H$ - C000 97FF$_H$	2 Kbyte		access[5]	access[5]
	C000 9800$_H$ - C000 9FFF$_H$	2 Kbyte		access[6]	access[6]
	C000 A000$_H$ - CFFF FFFF$_H$	≈ 256 Mbyte	Reserved	LMBBE & SPBBE	LMBBE
13	D000 0000$_H$ - D001 EFFF$_H$	124 Kbyte	DMI Local Data RAM (LDRAM)	access	access
	D001 F000$_H$ - D001 FFFF$_H$	4 Kbyte		access[7]	access[7]
	D002 0000$_H$ - D3FF FFFF$_H$	≈ 64 Mbyte	Reserved	LMBBE & SPBBE	LMBBE

Table 8-2 SPB Address Map of Segment 0 to 14 (cont'd)

Segment	Address Range	Size	Description	Access Type	
				Read	Write[1)]
	D400 0000$_H$ - D400 5FFF$_H$	24 Kbyte	PMI Scratch-Pad RAM (SPRAM)	access	access
	D400 6000$_H$ - D400 7FFF$_H$	8 Kbyte		access[3)]	access[3)]
	D400 8000$_H$ - D400 8FFF$_H$	4 Kbyte		access[4)]	access[4)]
	D400 9000$_H$ - D400 97FF$_H$	2 Kbyte		access[5)]	access[5)]
	D400 9800$_H$ - D400 9FFF$_H$	2 Kbyte		access[6)]	access[6)]
	D400 A000$_H$ - D7FF FFFF$_H$	≈ 64 Mbyte	Reserved	LMBBE & SPBBE	LMBBE
	D800 0000$_H$ - DEFF FFFF$_H$	112 Mbyte	Ext. Peripheral (EBU)	access	access
	DF00 0000$_H$ - DFFF FFFF$_H$	16 Mbyte	Reserved	LMBBE & SPBBE	LMBBE
14	E000 0000$_H$ - E7FF FFFF$_H$	128 MB	Ext. Peripheral (EBU)	access	access
	E800 0000$_H$ - E83F FFFF$_H$	4 MB	Reserved	LMBBE	LMBBE
	E840 0000$_H$ - E841 EFFF$_H$	124 Kbyte	DMI Local Data RAM (LDRAM)	access	access
	E841 F000$_H$ - E841 FFFF$_H$	4 Kbyte		access[7)]	access[7)]
	E842 0000$_H$ - E85F FFFF$_H$	≈ 1 Mbyte	Reserved	LMBBE	LMBBE

Table 8-2 SPB Address Map of Segment 0 to 14 (cont'd)

Seg-ment	Address Range	Size	Description	Access Type	
				Read	Write[1)]
	E850 0000$_H$ - E850 5FFF$_H$	24 Kbyte	PMI Scratch-Pad RAM (SPRAM)	access	access
	E850 6000$_H$ - E850 7FFF$_H$	8 Kbyte		access[3)]	access[3)]
	E850 8000$_H$ - E850 8FFF$_H$	4 Kbyte		access[4)]	access[4)]
	E850 9000$_H$ - E850 97FF$_H$	2 Kbyte		access[5)]	access[5)]
	E850 9800$_H$ - E850 9FFF$_H$	2 Kbyte		access[6)]	access[6)]
	E850 A000$_H$ - E85F FFFF$_H$	≈ 1 MB	Reserved	LMBBE	LMBBE
	E860 C000$_H$ - EFFF FFFF$_H$	≈ 122 Mbyte	Reserved	LMBBE	LMBBE
15	F000 0000$_H$ - FFFF FFFF$_H$	256 Mbyte	see **Table 8-3**		

1) Write access to PFlash / DFlash only applicable when writing Flash command sequences.

2) Online Data Acquisition address space can be disabled/enabled via PMU control register bit PMU0_OVRCON.OLDAEN. CPU access to OLDA address space via segment 8 (cached) results in LMBBET independent of the PMU0_OVRCON.OLDAEN bit setting.

3) Not available when Instruction Cache is configured for 16 Kbyte

4) Not available when Instruction Cache is configured for 8 Kbyte or 16Kbytes

5) Not available when Instruction Cache is configured for 4 Kbyte, 8 Kbytes or 16Kbytes

6) Not available when Instruction Cache is configured for 2 Kbyte, 4 Kbyte, 8 Kbytes or 16Kbytes

7) Not available when Data Cache is configured for 2 Kbyte or 4 Kbyte

8.4.2 Segment 15

Table 8-3 shows the address map of segment 15 as seen from the SPB bus masters PCP, DMA and OCDS. Please note that access in Table 8-3 means only that an access to an address within the defined address range is not automatically incorrect or ignored.

Table 8-3 SPB Address Map of Segment 15

Unit	Address Range	Size	Access Type	
			Read	Write
Reserved	$F000\ 0000_H$ - $F000\ 00FF_H$	–	SPBBE	SPBBE
System Peripheral Bus Control Unit (SBCU)	$F000\ 0100_H$ - $F000\ 01FF_H$	256 byte	access	access
System Timer (STM)	$F000\ 0200_H$ - $F000\ 02FF_H$	256 byte	access	access
Reserved	$F000\ 0300_H$ - $F000\ 03FF_H$	–	SPBBE	SPBBE
On-Chip Debug Support (Cerberus)	$F000\ 0400_H$ - $F000\ 04FF_H$	256 byte	access	access
System Control Unit (SCU) and Watchdog Timer (WDT)	$F000\ 0500_H$ - $F000\ 06FF_H$	2×256 byte	access	access
Reserved	$F000\ 0700_H$ - $F000\ 07FF_H$	–	SPBBE	SPBBE
MicroSecond Bus Controller 0 (MSC0)	$F000\ 0800_H$ - $F000\ 08FF_H$	256 byte	access	access
MicroSecond Bus Controller 1 (MSC1)	$F000\ 0900_H$ - $F000\ 09FF_H$	256 byte	access	access
Async./Sync. Serial Interface 0 (ASC0)	$F000\ 0A00_H$ - $F000\ 0AFF_H$	256 byte	access	access
Async./Sync. Serial Interface 1 (ASC1)	$F000\ 0B00_H$ - $F000\ 0BFF_H$	256 byte	access	access
Port 0	$F000\ 0C00_H$ - $F000\ 0CFF_H$	256 byte	access	access
Port 1	$F000\ 0D00_H$ - $F000\ 0DFF_H$	256 byte	access	access
Port 2	$F000\ 0E00_H$ - $F000\ 0EFF_H$	256 byte	access	access

Table 8-3 SPB Address Map of Segment 15 (cont'd)

Unit	Address Range	Size	Access Type	
			Read	Write
Port 3	F000 0F00$_H$ - F000 0FFF$_H$	256 byte	access	access
Port 4	F000 1000$_H$ - F000 10FF$_H$	256 byte	access	access
Port 5	F000 1100$_H$ - F000 11FF$_H$	256 byte	access	access
Port 6	F000 1200$_H$ - F000 12FF$_H$	256 byte	access	access
Port 7	F000 1300$_H$ - F000 13FF$_H$	256 byte	access	access
Port 8	F000 1400$_H$ - F000 14FF$_H$	256 byte	access	access
Port 9	F000 1500$_H$ - F000 15FF$_H$	256 byte	access	access
Port 10	F000 1600$_H$ - F000 16FF$_H$	256 byte	access	access
Port 11	F000 1700$_H$ - F000 17FF$_H$	256 byte	access	access
General Purpose Timer Array (GPTA0)	F000 1800$_H$ - F000 1FFF$_H$	8 × 256 byte	access	access
General Purpose Timer Array (GPTA1)	F000 2000$_H$ - F000 27FF$_H$	8 × 256 byte	access	access
Local Timer Cell Array (LTCA2)	F000 2800$_H$ - F000 2FFF$_H$	8 × 256 byte	access	access
Reserved	F000 3000$_H$ - F000 31FF$_H$	–	SPBBE	SPBBE
Reserved	F000 3200$_H$ - F000 33FF$_H$	–	SPBBE	SPBBE
Reserved	F000 3400$_H$ - F000 34FF$_H$	–	SPBBE	SPBBE
Reserved	F000 3500$_H$ - F000 35FF$_H$	–	SPBBE	SPBBE

Table 8-3 SPB Address Map of Segment 15 (cont'd)

Unit		Address Range	Size	Access Type	
				Read	Write
Reserved		$F000\ 3600_H$ - $F000\ 37FF_H$	–	SPBBE	SPBBE
Reserved		$F000\ 3800_H$ - $F000\ 3BFF_H$	–	SPBBE	SPBBE
Direct Memory Access Controller (DMA)		$F000\ 3C00_H$ - $F000\ 3EFF_H$	3 × 256 byte	access	access
Reserved		$F000\ 3F00_H$ - $F000\ 3FFF_H$	–	SPBBE	SPBBE
MultiCAN Controller (CAN)		$F000\ 4000_H$ - $F000\ 7FFF_H$	16 Kbyte	access	access
Reserved		$F000\ 8000_H$ - $F000\ FFFF_H$	–	SPBBE	SPBBE
FlexRay™ Protocol Controller (E-Ray)		$F001\ 0000_H$ - $F001\ 7FFF_H$	32 Kbyte	access	access
Reserved		$F001\ 8000_H$ - $F003\ FFFF_H$	–	SPBBE	SPBBE
PCP	Reserved	$F004\ 0000_H$ - $F004\ 3EFF_H$	–	SPBBE	SPBBE
	PCP Registers	$F004\ 3F00_H$ - $F004\ 3FFF_H$	256 byte	access	access
	Reserved	$F004\ 4000_H$ - $F004\ FFFF_H$	–	SPBBE	SPBBE
	PCP Data Memory (PRAM)	$F005\ 0000_H$ - $F005\ 3FFF_H$	16 Kbyte	nE, 32	nE, 32
	Reserved	$F005\ 4000_H$ - $F005\ FFFF_H$	–	SPBBE	SPBBE
	PCP Code Memory (CMEM)	$F006\ 0000_H$ - $F006\ 7FFF_H$	32 Kbyte	nE, 32	nE, 32
	Reserved	$F006\ 8000_H$ - $F007\ FFFF_H$	–	SPBBE	SPBBE
Reserved		$F008\ 0000_H$ - $F00F\ FFFF_H$	–	SPBBE	SPBBE

Table 8-3 SPB Address Map of Segment 15 (cont'd)

Unit	Address Range	Size	Access Type	
			Read	Write
Reserved	$F010\ 0000_H$ - $F010\ 00FF_H$	–	SPBBE	SPBBE
Synchronous Serial Interface 0 (SSC0)	$F010\ 0100_H$ - $F010\ 01FF_H$	256 byte	access	access
Synchronous Serial Interface 1 (SSC1)	$F010\ 0200_H$ - $F010\ 02FF_H$	256 byte	access	access
Reserved	$F010\ 0300_H$ - $F010\ 03FF_H$	-	SPBBE	SPBBE
Fast Analog-to-Digital Converter (FADC)	$F010\ 0400_H$ - $F010\ 05FF_H$	2 × 256 byte	access	access
Reserved	$F010\ 0600_H$ - $F010\ 0FFF_H$	–	SPBBE	SPBBE
Analog-to-Digital Converter 0 (ADC0)	$F010\ 1000_H$ - $F010\ 13FF_H$	4 × 256 byte	access	access
Analog-to-Digital Converter 1 (ADC1)	$F010\ 1400_H$ - $F010\ 17FF_H$	4 × 256 byte	access	access
Analog-to-Digital Converter 2 (ADC2)	$F010\ 1800_H$ - $F010\ 1BFF_H$	4 × 256 byte	access	access
Reserved	$F010\ 1C00_H$ - $F010\ 9FFF_H$	–	SPBBE	SPBBE
Reserved	$F010\ A000_H$ - $F010\ BFFF_H$	–	SPBBE	SPBBE
Micro Link Interface 0 (MLI0)	$F010\ C000_H$ - $F010\ C0FF_H$	256 byte	access	access
Micro Link Interface 1 (MLI1)	$F010\ C100_H$ - $F010\ C1FF_H$	256 byte	access	access
Memory Checker (MCHK)	$F010\ C200_H$ - $F010\ C2FF_H$	256 byte	access	access
Reserved	$F010\ C300_H$ - $F01D\ FFFF_H$	–	SPBBE	SPBBE
MLI0 Small Transfer Windows	$F01E\ 0000_H$ - $F01E\ 7FFF_H$	4 × 8 Kbyte	access	access

Table 8-3 SPB Address Map of Segment 15 (cont'd)

Unit		Address Range	Size	Access Type	
				Read	Write
MLI1 Small Transfer Windows		$F01E\ 8000_H$ - $F01E\ FFFF_H$	4 × 8 Kbyte	access	access
Reserved		$F01F\ 0000_H$ - $F01F\ FFFF_H$	–	SPBBE	SPBBE
MLI0 Large Transfer Windows		$F020\ 0000_H$ - $F023FFFF_H$	4 × 64 Kbyte	access	access
MLI1 Large Transfer Windows		$F024\ 0000_H$ - $F027\ FFFF_H$	4 × 64 Kbyte	access	access
Reserved		$F028\ 0000_H$ - $F02F\ FFFF_H$	–	SPBBE	SPBBE
Port 12		$F030\ 0000_H$ - $F030\ 00FF_H$	256 byte	access	access
Port 13		$F030\ 0100_H$ - $F030\ 01FF_H$	256 byte	access	access
Port 14		$F030\ 0200_H$ - $F030\ 02FF_H$	256 byte	access	access
Port 15		$F030\ 0300_H$ - $F030\ 03FF_H$	256 byte	access	access
Port 16		$F030\ 0400_H$ - $F030\ 04FF_H$	256 byte	access	access
Reserved		$F030\ 0500_H$ - $F7E0\ FEFF_H$	–	SPBBE	SPBBE
CPU	CPU Slave Interface Registers (CPS)	$F7E0\ FF00_H$ - $F7E0\ FFFF_H$	256 byte	access	access
	CPU Core SFRs & GPRs	$F7E1\ 0000_H$ - $F7E1\ FFFF_H$	64 Kbyte	access	access
Reserved		$F7E2\ 0000_H$ - $F7FF\ FFFF_H$	–	SPBBE	SPBBE
External Bus Unit (EBU)		$F800\ 0000_H$ - $F800\ 03FF_H$	1 Kbyte	access	access
Reserved		$F800\ 0400_H$ - $F800\ 04FF_H$	–	LMBBE & SPBBE	LMBBE

Table 8-3 SPB Address Map of Segment 15 (cont'd)

Unit		Address Range	Size	Access Type	
				Read	Write
Program Memory Unit 0 (PMU0)		$F800\ 0500_H$ - $F800\ 05FF_H$	256 byte	access	access
Program Memory Unit 1 (PMU1)		$F800\ 0600_H$ - $F800\ 06FF_H$	256 byte	access	access
Reserved		$F800\ 0700_H$ - $F800\ 0FFF_H$	–	LMBBE & SPBBE	LMBBE
Flash Register (PMU0)		$F800\ 1000_H$ - $F800\ 23FF_H$	5 Kbyte	access	access
Reserved		$F800\ 2400_H$ - $F800\ 2FFF_H$	–	LMBBE & SPBBE	LMBBE
Flash Register PMU1		$F800\ 3000_H$ - $F800\ 43FF_H$	5 Kbyte	access	access
Reserved		$F800\ 4400_H$ - $F801\ 00FF_H$	–	LMBBE & SPBBE	LMBBE
Reserved		$F801\ 0100_H$ - $F801\ 01FF_H$	–	LMBBE & SPBBE	LMBBE
Reserved		$F801\ 0200_H$ - $F87F\ F9FF_H$	–	LMBBE & SPBBE	LMBBE
Reserved		$F87F\ FA00_H$ - $F87F\ FAFF_H$	–	LMBBE & SPBBE	LMBBE
Overlay Control Unit (OVC)		$F87F\ FB00_H$ - $F87F\ FBFF_H$	256 byte	access	access
CPU	DMI Registers	$F87F\ FC00_H$ - $F87F\ FCFF_H$	256 byte	access	access
	PMI Registers	$F87F\ FD00_H$ - $F87F\ FDFF_H$	256 byte	access	access
Local Memory Bus Control Unit (LBCU)		$F87F\ FE00_H$ - $F87F\ FEFF_H$	256 byte	access	access
LFI Bridge		$F87F\ FF00_H$ - $F87F\ FFFF_H$	256 byte	access	access
Reserved		$F880\ 0000_H$ - $FFFF\ FFFF_H$	–	LMBBE & SPBBE	LMBBE

8.5 Address Map of the Local Memory Bus (LMB)

Table 8-4 shows the address map as seen from the LMB bus masters (PMI, DMI, DMA, MLI and Cerberus).

Table 8-4 LMB Address Map

Segment	Address Range	Size	Description	Action	
				Read	Write[1]
0-7	$0000\ 0000_H$ - $0000\ 0007_H$	8 byte	Reserved (virtual address space)	MPN trap	MPN trap
	$0000\ 0008_H$ - $7FFF\ FFFF_H$	8 × 256 Mbyte		SPBBET	SPBBE
8	$8000\ 0000_H$ - $801F\ FFFF_H$	2 Mbyte	Program Flash 0 (PFLASH0)	access	access[1]
	$8020\ 0000_H$ - $803F\ FFFF_H$	2 Mbyte	Program Flash 1 (PFLASH1)	access	access[1]
	$8040\ 0000_H$ - $807F\ FFFF_H$	4 Mbyte	Reserved	LMBBE	LMBBE
	$8080\ 0000_H$ - $8FDF\ FFFF_H$	246 Mbyte	External EBU Space	access	access
	$8FE0\ 0000_H$ - $8FE0\ 7FFF_H$	32 Kbyte	Data Flash (DFLASH) Bank 0	access	access[1]
	$8FE0\ 8000_H$ - $8FE0\ FFFF_H$	32 Kbyte	Reserved	LMBBET	LMBBET
	$8FE1\ 0000_H$ - $8FE1\ 7FFF_H$	32 Kbyte	Data Flash (DFLASH) Bank 1	access	access[1]
	$8FE1\ 8000_H$ - $8FE1\ FFFF_H$	32 Kbyte	Reserved	LMBBET	LMBBET
	$8FE2\ 0000_H$ - $8FE6\ FFFF_H$	–	Reserved	LMBBET	LMBBET
	$8FE7\ 0000_H$ - $8FE7\ 7FFF_H$	32 Kbyte	Online Data Acquisition (OLDA)	LMBBET	access[2] / LMBBET
	$8FE7\ 8000_H$ - $8FE7\ FFFF_H$	32 Kbyte	Reserved	LMBBET	LMBBET
	$8FE8\ 0000_H$ - $8FE8\ 1FFF_H$	8 Kbyte	Overlay memory (OVRAM)	access	access
	$8FE8\ 2000_H$ - $8FEF\ FFFF_H$	–	Reserved	LMBBET	LMBBET

Table 8-4 LMB Address Map (cont'd)

Segment	Address Range	Size	Description	Action	
				Read	Write[1)]
	8FF0 0000$_H$ - 8FF7 FFFF$_H$	512 Kbyte	Reserved for TC1797 emulation device memory	LMBBET	LMBBET
	8FF8 0000$_H$ - 8FFF BFFF$_H$	–	Reserved	LMBBET	LMBBET
	8FFF C000$_H$ - 8FFF FFFF$_H$	16 Kbyte	Boot ROM (BROM)	access	LMBBET
9	9000 0000$_H$ - 9FFF FFFF$_H$	256 Mbyte	Reserved	SPBBET	SPBBE
10	A000 0000$_H$ - A01F FFFF$_H$	2 Mbyte	Program Flash 0 (PFLASH0)	access	access[1)]
	A020 0000$_H$ - A03F FFFF$_H$	2 Mbyte	Program Flash 1 (PFLASH1)	access	access[1)]
	A040 0000$_H$ - A07F FFFF$_H$	4 Mbyte	Reserved	LMBBE	LMBBE
	A080 0000$_H$ - AFDF FFFF$_H$	246 Mbyte	External EBU Space	access	access
	AFE0 0000$_H$ - AFE0 7FFF$_H$	32 Kbyte	Data Flash (DFLASH) Bank 0	access	access[1)]
	AFE0 8000$_H$ - AFE0 FFFF$_H$	32 Kbyte	Reserved	LMBBET	LMBBET
	AFE1 0000$_H$ - AFE1 7FFF$_H$	32 Kbyte	Data Flash (DFLASH) Bank 1	access	access[1)]
	AFE1 8000$_H$ - AFE1 FFFF$_H$	32 Kbyte	Reserved	LMBBET	LMBBET
	AFE2 0000$_H$ - AFE6 FFFF$_H$	–	Reserved	LMBBET	LMBBET
	AFE7 0000$_H$ - AFE7 7FFF$_H$	32 Kbyte	Online Data Acquisition (OLDA)	LMBBET	access[2)] / LMBBET
	AFE7 8000$_H$ - AFE7 FFFF$_H$	32 Kbyte	Reserved	LMBBET	LMBBET
	AFE8 0000$_H$ - AFE8 1FFF$_H$	8 Kbyte	Overlay memory (OVRAM)	access	access

Table 8-4 LMB Address Map (cont'd)

Segment	Address Range	Size	Description	Action	
				Read	Write[1)]
	AFE8 2000$_H$ - AFEF FFFF$_H$	–	Reserved	LMBBET	LMBBET
	AFF0 0000$_H$ - AFF7 FFFF$_H$	512 Kbyte	Reserved for TC1797 emulation device memory	LMBBET	LMBBET
	AFF8 0000$_H$ - AFFF BFFF$_H$	–	Reserved	LMBBET	LMBBET
	AFFF C000$_H$ - AFFF FFFF$_H$	16 Kbyte	Boot ROM (BROM)	access	LMBBET
11	B000 0000$_H$ - BFFF FFFF$_H$	256 Mbyte	Reserved	SPBBET	SPBBE
12	C000 0000$_H$ - C000 5FFF$_H$	24 Kbyte	PMI Scratch-Pad RAM (SPRAM)	access	access
	C000 6000$_H$ - C000 7FFF$_H$	8 Kbyte		access[3)]	access[3)]
	C000 8000$_H$ - C000 8FFF$_H$	4 Kbyte		access[4)]	access[4)]
	C000 9000$_H$ - C000 97FF$_H$	2 Kbyte		access[5)]	access[5)]
	C000 9800$_H$ - C000 9FFF$_H$	2 Kbyte		access[6)]	access[6)]
	C000 A000$_H$ - CFFF FFFF$_H$	≈ 256 Mbyte	Reserved	LMBBET	LMBBET
13	D000 0000$_H$ - D001 EFFF$_H$	124 Kbyte	DMI Local Data RAM (LDRAM)	access	access
	D001 F000$_H$ - D001 FFFF$_H$	4 Kbyte		access[7)]	access[7)]
	D001 B000$_H$ - D3FF FFFF$_H$	≈ 64 Mbyte	Reserved	LMBBET	LMBBET

Table 8-4 LMB Address Map (cont'd)

Seg-ment	Address Range	Size	Description	Action	
				Read	Write[1)]
	$D400\ 0000_H$ - $D400\ 5FFF_H$	24 Kbyte	PMI Scratch-Pad RAM (SPRAM)	access	access
	$D400\ 6000_H$ - $D400\ 7FFF_H$	8 Kbyte		access[3)]	access[3)]
	$D400\ 8000_H$ - $D400\ 8FFF_H$	4 Kbyte		access[4)]	access[4)]
	$D400\ 9000_H$ - $D400\ 97FF_H$	2 Kbyte		access[5)]	access[5)]
	$D400\ 9800_H$ - $D400\ 9FFF_H$	2 Kbyte		access[6)]	access[6)]
	$D400\ A000_H$ - $D7FF\ FFFF_H$	≈ 64 Mbyte	Reserved	LMBBET	LMBBET
	$D800\ 0000_H$ - $DEFF\ FFFF_H$	112 Mbyte	Ext.Peripeheral (EBU)	access	access
	$DF00\ 0000_H$ - $DFFF\ FFFF_H$	16 Mbyte	Reserved	LMBBET	LMBBET
14	$E000\ 0000_H$ - $E7FF\ FFFF_H$	128 Mbyte	Ext. Peripheral (EBU)	access	access
	$E800\ 0000_H$ - $EFFF\ FFFF_H$	128 Mbyte	Reserved	LMBBET	LMBBET
15	$F000\ 0000_H$ - $F7FF\ FFFF_H$	128 Mbyte	Address map is identical to FPI Bus segment 15 address map (see **Table 8-3**) Reserved areas give an bus error.	SPBBET	SPBBE
	$F800\ 0000_H$ - $F800\ 03FF_H$	1 Kbyte	External Bus Unit (EBU)	access	access
	$F800\ 0400_H$ - $F800\ 04FF_H$	256 byte	Reserved	LMBBET	LMBBET
	$F800\ 0500_H$ - $F800\ 05FF_H$	256 byte	Program Memory Unit 0 (PMU0)	access	access
	$F800\ 0600_H$ - $F800\ 06FF_H$	256 byte	Program Memory Unit (PMU)	access	access

Table 8-4 LMB Address Map (cont'd)

Segment	Address Range	Size	Description	Action	
				Read	Write[1)]
	$F800\ 0700_H$ - $F800\ 0FFF_H$	≈ 2 Kbyte	Reserved	LMBBET	LMBBET
	$F800\ 1000_H$ - $F800\ 23FF_H$	5 Kbyte	Flash Registers (PMU0)	access	access
	$F800\ 2400_H$ - $F800\ 2FFF_H$	–	Reserved	LMBBET	LMBBET
	$F800\ 3000_H$ - $F800\ 43FF_H$	5 Kbyte	Flash Registers PMU1	access	access
	$F800\ 2400_H$ - $F87F\ FAFF_H$	≈ 8 Mbyte	Reserved	LMBBET	LMBBET
	$F87F\ FB00_H$ - $F87F\ FBFF_H$	256 byte	Overlay Control Unit (OVC)	access	access
	$F87F\ FC00_H$ - $F87F\ FCFF_H$	256 byte	Data Memory Interface Unit (DMI)	access	access
	$F87F\ FD00_H$ - $F87F\ FDFF_H$	256 byte	Program Memory Interface Unit (PMI)	access	access
	$F87F\ FE00_H$ - $F87F\ FEFF_H$	256 byte	LBCU register space	access	access
	$F87F\ FF00_H$ - $F87F\ FFFF_H$	256 byte	LFI Bus Bridge	access	access
	$F880\ 0000_H$ - $FFFF\ FFFF_H$	≈ 119 Mbyte	Reserved	LMBBET	LMBBET

1) Write access to PFlash / DFlash only applicable when writing Flash command sequences.

2) Online Data Acquisition address space can be disabled/enabled via PMU control register bit PMU0_OVRCON.OLDAEN. CPU access to OLDA address space via segment 8 (cached) results in LMBBET independent of the PMU0_OVRCON.OLDAEN bit setting.

3) Not available when Instruction Cache is configured for 16 Kbyte

4) Not available when Instruction Cache is configured for 8 Kbyte or 16Kbytes

5) Not available when Instruction Cache is configured for 4 Kbyte, 8 Kbytes or 16Kbytes

6) Not available when Instruction Cache is configured for 2 Kbyte, 4 Kbyte, 8 Kbytes or 16Kbytes

7) Not available when Data Cache is configured for 2 Kbyte or 4 Kbyte

8.6 Memory Module Access Restrictions

Table 8-5 describes which type of accesses are possible to the different memories in the TC1797.

Table 8-5 Possible Memory Accesses

Memory		Bit	Byte		Half-word		Word		Double-word	
		rmw	r	w	r	w	r	w	r	w
PMI[1]	SPRAM	✔	✔	✔	✔	✔	✔	✔	✔	✔
DMI[1]	LDRAM	✔	✔	✔	✔	✔	✔	✔	✔	✔
PMU	ROM	–	✔	–	✔	–	✔	–	✔	–
	PFLASH	–	✔	–	✔	–	✔	✔	✔	✔
	DFLASH	–	✔	–	✔	–	✔	✔	✔	✔
	OVRAM[1]	–	✔	✔	✔	✔	✔	✔	✔	✔
PCP[2]	CMEM	✔	–	–	–	–	✔	✔	✔	✔
	PRAM	✔	–	–	–	–	✔	✔	✔	✔

1) The module also supports LMB 2-Word and 4-Word Block read and write accesses.

2) The module also supports FPI 4-Word and 8-Word Block read and write accesses.

8.7 Side Effects from Modules to LDRAM

Please note that the LDRAM can be used by the CPU for system tasks:

LDRAM can be used as context save area (for details see chapter ´CPU Subsystem´)

9 General Purpose I/O Ports and Peripheral I/O Lines (Ports)

The TC1797 has 215 digital General Purpose Input/Output (GPIO) port lines which are connected to the on-chip peripheral units. They are divided into:

Table 9-1 Ports Overview

Port	Pins	Base Address	End Address	Note
0	16	F000 0C00$_H$	F000 0CFF	+hwcfg, +E-Ray, EMSTOP
1	16	F000 0D00$_H$	F000 0DFF	EMSTOP
2	14	F000 0E00$_H$	F000 0EFF	EMSTOP
3	16	F000 0F00$_H$	F000 0FFF	EMSTOP
4	16	F000 1000$_H$	F000 10FF	EMSTOP
5	16	F000 1100$_H$	F000 11FF	extended with LVDS, EMSTOP
6	12	F000 1200$_H$	F000 12FF	6.4 to 6.15
7	8	F000 1300$_H$	F000 13FF	
8	8	F000 1400$_H$	F000 14FF	EMSTOP
9	9 -> 15	F000 1500$_H$	F000 15FF	EMSTOP only 6 added. no change to the old ones. BRKIN, BRKOUT
10	4 -> 6	F000 1600$_H$	F000 16FF	fully changed, - EBU
11	16	F000 1700$_H$	F000 17FF	-
12	8	F030 0000$_H$	F030 00FF$_H$	-
13	16	F030 0100$_H$	F030 01FF$_H$	EMSTOP
14	16	F030 0200$_H$	F030 02FF$_H$	EMSTOP
15	16	F030 0300$_H$	F030 03FF$_H$	-
16	4	F030 0400$_H$	F030 04FF$_H$	-

Additionally, fourteen dedicated input/output lines without GPIO functionality are provided. The External Bus Interface (EBU) has its own set of dedicated signal lines. Further details are described in the port-specific sections of this chapter.

9.1 Basic Port Operation

Figure 9-1 is a general block diagram of a TC1797 GPIO port slice.

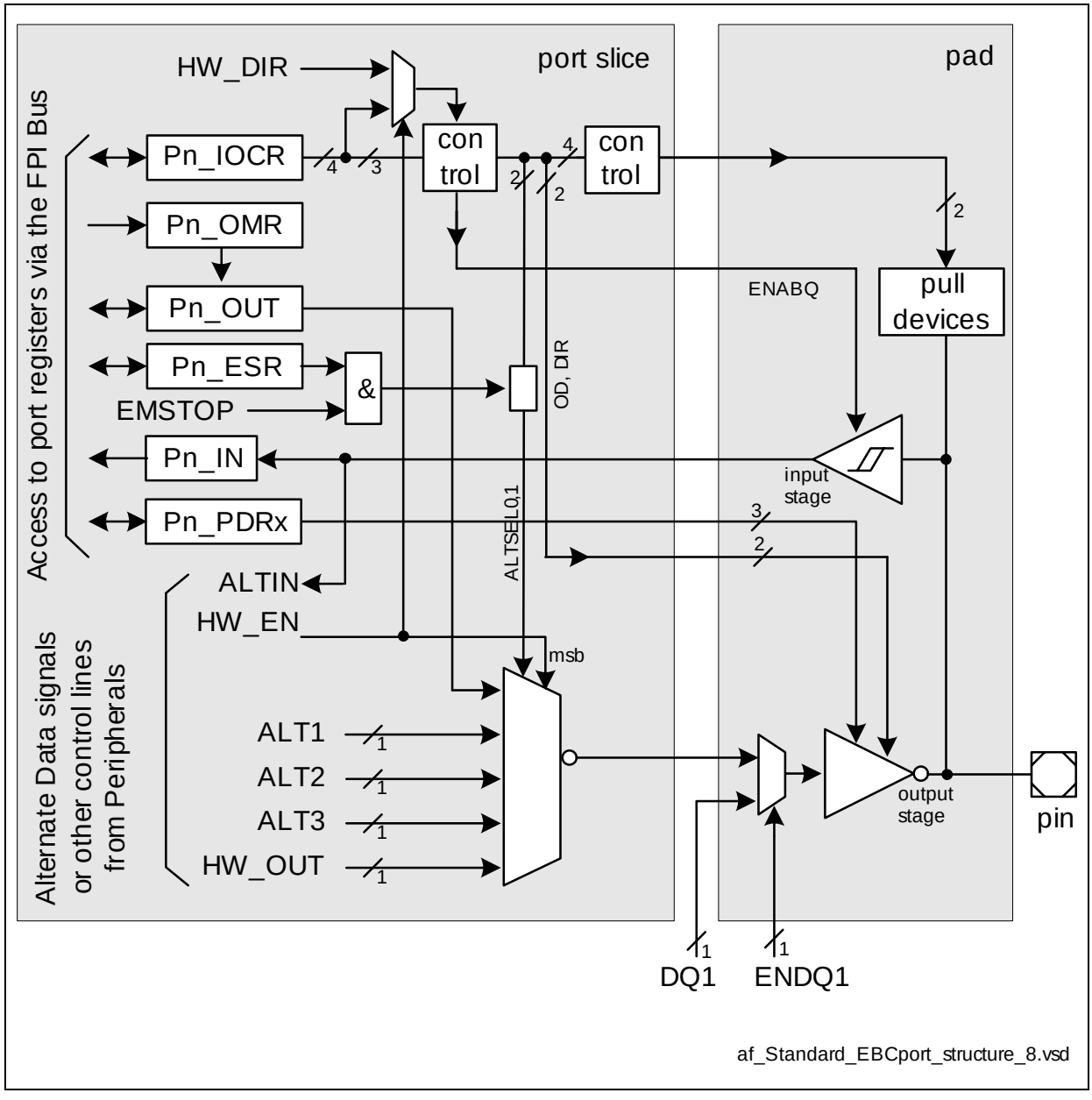


Figure 9-1 General Structure of a Port Pin

Each port line has a number of control and data bits, enabling very flexible usage of the line. Each port pin (except Port 10) can be configured for input or output operation. In input mode (default after reset), the output driver is switched off (high-impedance). The actual voltage level present at the port pin is translated into a logical 0 or 1 via a Schmitt-Trigger device and can be read via the read-only register Pn_IN. An input signal can also be connected directly to the various inputs of the peripheral units (AltDataIn). The

function of the input line from the pin to the input register Pn_IN and to AltDataIn is independent of whether the port pin operates as input or output. This means that when the port is in output mode, the level of the pin can be read by software via Pn_IN or a peripheral can use the pin level as an input.

In output mode, the output driver is activated and drives the value supplied through the multiplexer to the port pin. Switching between input and output mode is accomplished through the Pn_IOCR register, which enables or disables the output driver. If a peripheral unit uses a GPIO port line as a bi-directional I/O line, register Pn_IOCR has to be written for input or output selection. The Pn_IOCR register further controls the driver type of the output driver, and determines whether an internal weak pull-up or pull-down device is alternatively connected to the pin when used as an input. This offers additional advantages in an application.

The output multiplexer in front of the output driver selects the signal source for the GPIO line when used as output. If the pin is used as general-purpose output, the multiplexer is switched by software (Pn_IOCR register) to the Output Data Register Pn_OUT. Software can set or clear the bit in Pn_OUT, and therefore it can directly influence the state of the port pin. If the on-chip peripheral units use the pin for output signals, the alternate output lines ALT1 to ALT3 can be switched via the multiplexer to the output driver. The data written into the output register Pn_OUT by software can be used as input data to an on-chip peripheral. This enables, for example, peripheral tests via software without external circuitry.

When selected as general-purpose output line, the logic state of each port pin can be changed individually by programming the pin-related bits in the Output Modification Register Pn_OMR. The bits in Pn_OMR make it possible to set, reset, toggle, or leave the bits in the Pn_OUT register unchanged.

When selected as general-purpose output line, the actual logic level at the pin can be examined through reading Pn_IN and compared against the applied output level (either applied through software via the output register Pn_OUT, or via an alternate output function of a peripheral unit). This can be used to detect some electrical failures at the pin caused through external circuitry. In addition, software-supported arbitration schemes can be implemented in this way using the open-drain configuration and an external wired-And circuitry. Collisions on the external communication lines can be detected when a high level (1) is output, but a low level (0) is seen when reading the pin value via the input register Pn_IN.

All GPIO lines of the TC1797 that are used by the GPTA modules (GPTA0, GPTA1, LTCA2) have an emergency stop logic. This logic makes it possible to individually disconnect GPTA outputs from the driving GPTA module outputs and to put them onto a well defined logic state in an emergency case. In en emergency case, the content of the port output register Pn_OUT is driven at the output pin instead of the GPTA module output. The Emergency Stop Register Pn_EMR determines whether a GPTA output is enabled or disabled in an emergency case.

9.2 Description Scheme for the Port IO Functions

The following two general building block can be used to describe each GPIO pin:

Table 9-2 Port x Input/Output Functions

Port Pin	I/O	Select	Connected Signal(s)	From / to Module
Px.y	Input		Signal(s)	module(s)
	Output	GPIO	Signal	module
		ALT1	Signal	module
		ALT2	Signal	module
		ALT3	Signal	module
	HW_DIR	HW_Out	Signal	module; group En

or:

Table 9-3 Port x Functions

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
Px.y	I	GPIOt	Px_IN.Py	Px_IOCRz. PC0	$0XXX_B$
	O	GPIO	Px_OUT.Py		$1X00_B$
					$1X01_B$
					$1X10_B$
					$1X11_B$
	HW_DIR	module; group En	Signal		HW_DIR

- HW_DIR:
  The type Alternate Direction signal which is needed if HW_En is active:
  – Out -always output
    DIRx - the pins in one port having the same DIRx (x=0, 1, 2, ...), are controlled as a group by a dedicated HW_DIR signal.
    SDIR- Single DIR- the pin is controlled by its own, dedicated, single HW_DIR signal.
- grouping indicates if the respective pin is controlled by hardware:

  - ENx - the pins in one port having the same ENx (x=0, 1, 2, ...), are controlled as a group by a dedicated HW_EN signal.
  - SEN - Single EN - the pin is controlled by its own, dedicated, single HW_EN signal
- Digital port slices with HW_DIR defined are the ports described in **Figure 9-1**.

Note: HW_EN signal has higher priority then Emergency Stop. Emergency Stop is functional when the pins are set in the GPIO mode.

Note: HW_DIR signal, output case, switches the pad to push-pull output state.
HW_DIR signal, input case, switches the pad to the input state with pull-up/down setting as defined by the IOCR register.

9.3 Port Register Description

The individual control and data bits of each GPIO port are implemented in a number of registers. The registers are used to configure and use the port as general-purpose I/O or alternate function input/output. For some ports, not all registers are implemented. The availability of the registers in the specific ports is described in **Table 9-10** on **Page 9-22** up to **Table 9-28** on **Page 9-80**.

Note: All port-registers have control bits implemented in groups of four (a nibble), starting from the bit position 0. If a port is do not fit to multiple of four without rest, or starts with a bit number other than zero, then some control bits remain unused. These bits behave as standard read-write bits, but do not have any function. The registers of not implemented groups of bits starting on the position 0 are implemented, but do not have any function. For example, Port 6 contains control registers and bitfields for the first nibble P6.0 to P6.3, although there are no such pins physically implemented. The not implemented bits appear in the boundary-scan chain, although they do not have an external connection.

Port Register Overview

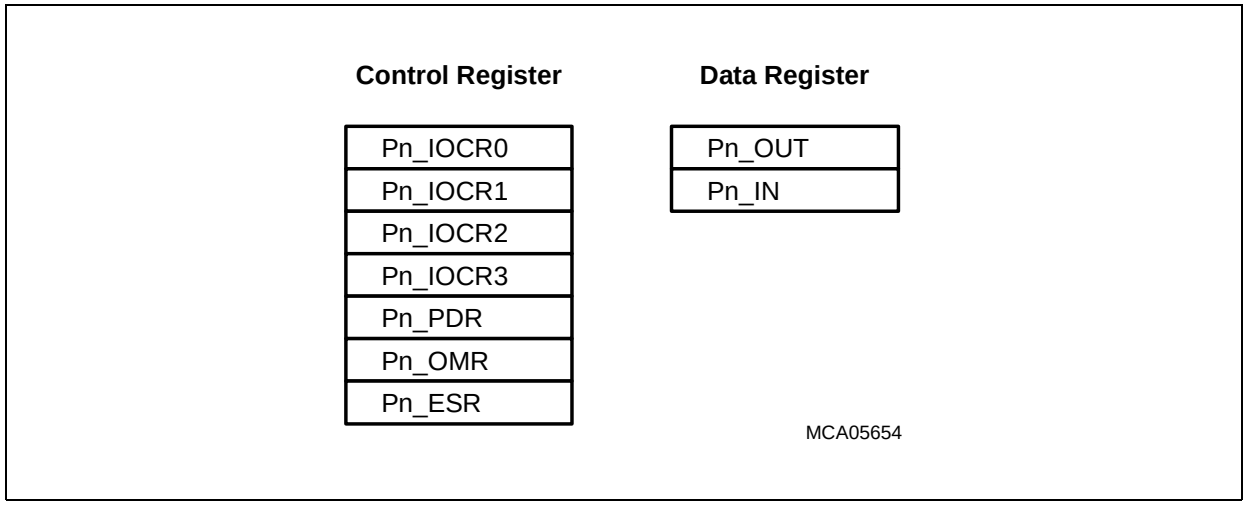


Figure 9-2 Port Registers

Table 9-4 Registers Address Space

Module	Base Address	End Address	Note
P0	F000 0C00$_H$	F000 0CFF$_H$	16
P1	F000 0D00$_H$	F000 0DFF$_H$	16
P2	F000 0E00$_H$	F000 0EFF$_H$	14
P3	F000 0F00$_H$	F000 0FFF$_H$	16
P4	F000 1000$_H$	F000 10FF$_H$	16

Table 9-4 Registers Address Space

Module	Base Address	End Address	Note
P5	$F000\ 1100_H$	$F000\ 11FF_H$	16
P6	$F000\ 1200_H$	$F000\ 12FF_H$	12
P7	$F000\ 1300_H$	$F000\ 13FF_H$	8
P8	$F000\ 1400_H$	$F000\ 14FF_H$	8
P9	$F000\ 1500_H$	$F000\ 15FF_H$	9 -> 15
P10	$F000\ 1600_H$	$F000\ 16FF_H$	4 -> 6
P11	$F000\ 1700_H$	$F000\ 17FF_H$	16
P12	$F030\ 0000_H$	$F030\ 00FF_H$	8
P13	$F030\ 0100_H$	$F030\ 01FF_H$	16
P14	$F030\ 0200_H$	$F030\ 02FF_H$	16
P15	$F030\ 0300_H$	$F030\ 03FF_H$	16
P16	$F030\ 0400_H$	$F030\ 04FF_H$	8

Table 9-5 Registers Overview

Register Short Name	Register Long Name	Offset Address	Description see
Pn_OUT	Port n Output Register	0000_H	Page 9-15
Pn_OMR	Port n Output Modification Register	0004_H	Page 9-16
Pn_IOCR0	Port n Input/Output Control Register 0	0010_H	Page 9-9
Pn_IOCR4	Port n Input/Output Control Register 4	0014_H	Page 9-10
Pn_IOCR8	Port n Input/Output Control Register 8	0018_H	Page 9-10
Pn_IOCR12	Port n Input/Output Control Register 12	$001C_H$	Page 9-11
Pn_IN	Port n Input Register	0024_H	Page 9-19
Pn_PDR	Port n Pad Driver Mode Register	0040_H	Page 9-13
Pn_ESR	Port n Emergency Stop Register	0050_H	Page 9-18

Register Access Rights and Reset Class

Table 9-6 Registers Access Rights and Reset Classes

Register Short Name	Access Rights		Reset Class
	Read	Write	
Pn_OUT	U,SV	U,SV	Class 3
Pn_OMR			
Pn_IOCR0			
Pn_IOCR4			
Pn_IOCR8			
Pn_IOCR12			
Pn_IN			
Pn_PDR		SV,E	
Pn_ESR			

9.3.1 Port Input/Output Control Registers

The port input/output control registers select the digital output and input driver functionality and characteristics of a GPIO port pin. Port direction (input or output), pull-up or pull-down devices for inputs, and push-pull or open-drain functionality for outputs can be selected by the corresponding bit fields PCx (x = 0-15). Each 32-bit wide port input/output control register controls four GPIO port lines:

Register Pn_IOCR0 controls the Pn.[3:0] port lines
Register Pn_IOCR4 controls the Pn.[7:4] port lines
Register Pn_IOCR8 controls the Pn.[11:8] port lines
Register Pn_IOCR12 controls the Pn.[15:12] port lines

The diagrams below show the register layouts of the port input/output control registers with the PCx bit fields. One PCx bit field controls exactly one port line Pn.x.

Pn_IOCR0 (n=0-1)
Port n Input/Output Control Register 0
($F000\ 0C10_H + n*100_H$) **Reset Value: $2020\ 2020_H$**

Pn_IOCR0 (n=3-11)
Port n Input/Output Control Register 0
($F000\ 0C10_H + n*100_H$) **Reset Value: $2020\ 2020_H$**

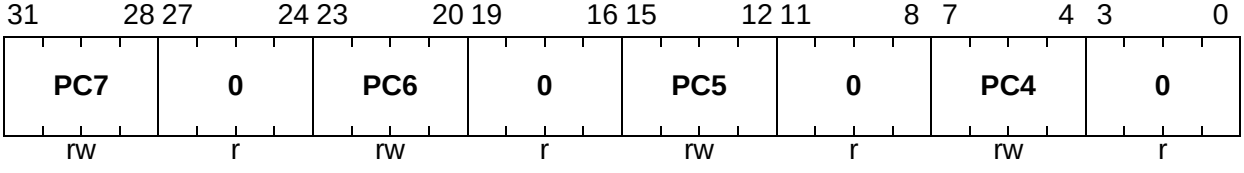
Pn_IOCR0 (n=12-16)
Port n Input/Output Control Register 0
($F02F\ F410_H + n*100_H$) **Reset Value: $2020\ 2020_H$**

31 28	27 24	23 20	19 16	15 12	11 8	7 4	3 0
PC3	0	PC2	0	PC1	0	PC0	0
rw	r	rw	r	rw	r	rw	r

Field	Bits	Type	Description
PC0, PC1, PC2, PC3	[7:4], [15:12], [23:20], [31:28]	rw	**Port Control for Port n Pin 0 to 3** This bit field determines the Port n line x functionality (x = 0-3) according to the coding table (see Table 9-7).
0	[3:0], [11:8], [19:16], [27:24]	r	**Reserved** Read as 0; should be written with 0.

Pn_IOCR4 (n=0-11)
Port n Input/Output Control Register 4
(F000 0C14$_H$ + n*100$_H$) **Reset Value: 2020 2020$_H$**

Pn_IOCR4 (n=12-16)
Port n Input/Output Control Register 4
(F02F F414$_H$ + n*100$_H$) **Reset Value: 2020 2020$_H$**



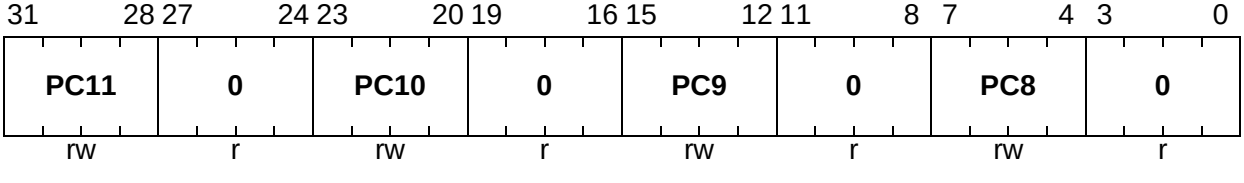
Field	Bits	Type	Description
PC4, PC5, PC6, PC7	[7:4], [15:12], [23:20], [31:28]	rw	**Port Control for Port n Pin 4 to 7** This bit field determines the Port n line x functionality (x = 4-7) according to the coding table (see **Table 9-7**).
0	[3:0], [11:8], [19:16], [27:24]	r	**Reserved** Read as 0; should be written with 0.

Pn_IOCR8 (n=0-6)
Port n Input/Output Control Register 8
(F000 0C18$_H$ + n*100$_H$) **Reset Value: 2020 2020$_H$**

P9_IOCR8
Port 9 Input/Output Control Register 8
(18$_H$) **Reset Value: 2020 2020$_H$**

P11_IOCR8
Port 11Input/Output Control Register 8
(18$_H$) **Reset Value: 2020 2020$_H$**

Pn_IOCR8 (n=13-15)
Port n Input/Output Control Register 8
(F02F F418$_H$ + n*100$_H$) **Reset Value: 2020 2020$_H$**

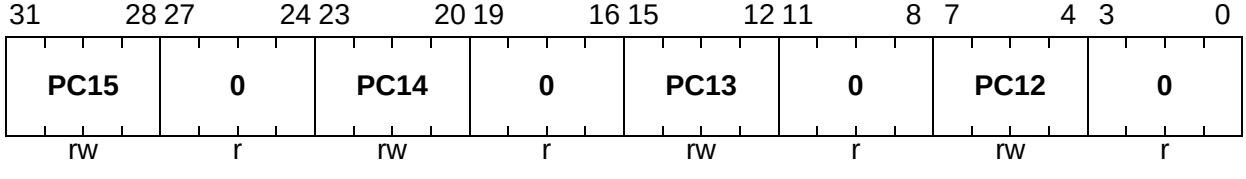


Field	Bits	Type	Description
PC8, PC9, PC10, PC11	[7:4], [15:12], [23:20], [31:28]	rw	**Port Control for Port n Pin 8 to 11** This bit field determines the Port n line x functionality (x = 8-11) according to the coding table (see **Table 9-7**).
0	[3:0], [11:8], [19:16], [27:24]	r	**Reserved** Read as 0; should be written with 0.

Pn_IOCR12 (n=0-6)
Port n Input/Output Control Register 12
(F000 0C1C$_H$ + n*100$_H$) **Reset Value: 2020 2020$_H$**

P11_IOCR12
Port 11 Input/Output Control Register 12
(1C$_H$) **Reset Value: 2020 2020$_H$**

Pn_IOCR12 (n=13-15)
Port n Input/Output Control Register 12
(F02F F41C$_H$ + n*100$_H$) **Reset Value: 2020 2020$_H$**



Field	Bits	Type	Description
PC12, PC13, PC14, PC15	[7:4], [15:12], [23:20], [31:28]	rw	**Port Control for Port n Pin 12 to 15** This bit field determines the Port n line x functionality (x = 12-15) according to the coding table (see **Table 9-7**).
0	[3:0], [11:8], [19:16], [27:24]	r	**Reserved** Read as 0; should be written with 0.

Depending on the GPIO port functionality (number of GPIO lines of a port), not all of the port input/output control registers are implemented.

The structure with one control bit field for each port pin located in different register bytes offers the possibility to configure the port pin functionality of a single pin with byte-oriented accesses without accessing the other PCx bit fields.

Port Control Coding

Table 9-7 describes the coding of the PCx bit fields that determine the port line functionality. Port 4 has a different PCx coding than the other ports (in input mode, no pull devices can be connected).

Table 9-7 PCx Coding

PCx[3:0]	I/O	Output Characteristics	Selected Pull-up / Pull-down / Selected Output Function
$0X00_B$	Input	–	No input pull device connected
$0X01_B$			Input pull-down device connected
$0X10_B$			Input pull-up device connected[1)]
$0X11_B$			No input pull device connected
1000_B	Output	Push-pull	General-purpose output
1001_B			Alternate output function 1
1010_B			Alternate output function 2
1011_B			Alternate output function 3
1100_B		Open-drain	General-purpose output
1101_B			Alternate output function 1
1110_B			Alternate output function 2
1111_B			Alternate output function 3

1) This is the default pull-up state after reset.

9.3.2 Pad Driver Mode Register

Overview

The pad structure of the TC1797 GPIO lines offers the possibility to select the output driver strength and the slew rate. These two parameters are controlled by the bit fields in the pad driver mode register Pn_PDR, independently from input/output and pull-up/pull-down control functionality as programmed in the Pn_IOCR register. One Pn_PDR register is assigned to each port.

The GPIO port lines consist of by two classes of pads:

- Class A1 pins (low speed 3.3V LVTTL outputs)
- Class A2 pins (high speed 3.3V LVTTL outputs. e.g. for serial outputs)

The assignment of each port pin to one of these pad classes is shown in the port configuration figures. Further details about pad driver classes that are available in the TC1797 are summarized in the “Electrical Specification” chapter.

Depending on the assigned pad class, the 3-bit wide pad driver mode selection bit fields PDx in the pad driver mode registers Pn_PDR make it possible to select the port line functionality as shown in **Table 9-8**. Note that the pad driver mode registers are specific for each port. Therefore, the Pn_PDR layout is described for each port in the port-specific sections.

Class A1 pins make it possible to select between medium and weak output drivers. Class A2 pins make it possible to select between strong/medium/weak output drivers. In the case of strong driver type, the signal transition edge can be additionally selected as sharp/medium/soft.

Table 9-8 Pad Driver Mode Selection

Pad Class	PDx.2	PDx.1	PDx.0	Functionality
A1	X	X	0	Medium driver selected
			1	Weak driver selected
A2/B1/B2	0	0	0	Strong driver, sharp edge selected
	0	0	1	Strong driver, medium edge selected
	0	1	0	Strong driver, soft edge selected
	0	1	1	Weak driver selected
	1	0	0	Medium driver selected
	1	0	1	
	1	1	0	
	1	1	1	Weak driver selected

Note: TC1797 Data Sheet describes the DC characteristics of all pad classes.

Pad Driver Mode Register

This is the general description of the PDR register. Each port contains its own specific PDR register, described additionally at each port, that can contain between one and eight PDx fields. Each field controlls a number of pins, normally four, but in general between one and sixteen.

Px_PDR

Port x Pad Driver Mode Register(F000 0C40$_H$+x*100$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**PD7**			**0**	**PD6**			**0**	**PD5**			**0**	**PD4**		
r	rw			r	rw			r	rw			r	rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**PD3**			**0**	**PD2**			**0**	**PD1**			**0**	**PD0**		
r	rw			r	rw			r	rw			r	rw		

Field	Bits	Type	Description
PD0	[2:0]	rw	**Pad Driver Mode for Px.[3:0]** (Class A1 or A2 pads; coding see **Page 9-13**)
PD1	[6:4]	rw	**Pad Driver Mode for Px.[7:4]** (Class A1 or A2 pads; coding see **Page 9-13**)
PD2	[10:8]	rw	**Pad Driver Mode for Px.[11:8]** (Class A1 or A2 pads; coding see **Page 9-13**)
PD3	[14:12]	rw	**Pad Driver Mode for Px.[15:12]** (Class A1 or A2 pads; coding see **Page 9-13**)
PD4	[2:0]	rw	**Pad Driver Mode for Px.[19:16]**
PD5	[6:4]	rw	**Pad Driver Mode for Px.[23:20]**
PD6	[10:8]	rw	**Pad Driver Mode for Px.[27:24]**
PD7	[14:12]	rw	**Pad Driver Mode for Px.[31:28]**
0	3, 7, 11, 15, 19, 23, 27, 31	r	**Reserved** Read as 0; should be written with 0.

9.3.3 Port Output Register

The port output register determines the value of a GPIO pin when it is selected by Pn_IOCRx as output. Writing a 0 to a Pn_OUT.Px (x = 0-15) bit position delivers a low level at the corresponding output pin. A high level is output when the corresponding bit is written with a 1. Note that the bits of Pn_OUT.Px can be individually set/reset by writing appropriate values into the port output modification register Pn_OMR.

Pn_OUT (n=0-11)
Port n Output Register **(F000 0C00$_H$ + n*100$_H$)** **Reset Value: 0000 0000$_H$**
Pn_OUT (n=12-16)
Port n Output Register **(F02F F400$_H$ + n*100$_H$)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
Px **(x = 0-15)**	x	rwh	**Port n Output Bit x** This bit determines the level at the output pin Pn.x if the output is selected as GPIO output. 0_B The output level of Pn.x is 0. 1_B The output level of Pn.x is 1. Pn.x can also be set/reset by control bits of the Pn_OMR register.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

Note: Only Port 0, 1, 3, and 4 are 16-bit wide ports. The Pn_OUT registers of the other ports have a reduced number of Px bits (see Pn_OUT register descriptions in the corresponding port sections).

9.3.4 Port Output Modification Register

The port output modification register contains control bits that make it possible to individually set, reset, or toggle the logic state of a single port line by manipulating the output register.

Pn_OMR (n=0-11)
Port n Output Modification Register (F000 0C04$_H$ + n*100$_H$) Reset Value: 0000 0000$_H$
Pn_OMR (n=12-16)
Port n Output Modification Register (F02F F404$_H$ + n*100$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PR 15	**PR 14**	**PR 13**	**PR 12**	**PR 11**	**PR 10**	**PR 9**	**PR 8**	**PR 7**	**PR 6**	**PR 5**	**PR 4**	**PR 3**	**PR 2**	**PR 1**	**PR 0**
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PS 15	**PS 14**	**PS 13**	**PS 12**	**PS 11**	**PS 10**	**PS 9**	**PS 8**	**PS 7**	**PS 6**	**PS 5**	**PS 4**	**PS 3**	**PS 2**	**PS 1**	**PS 0**
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
PSx (x = 0-15)	x	w	**Port n Set Bit x** Setting this bit will set or toggle the corresponding bit in the port output register Pn_OUT. The function of this bit is shown in **Table 9-9**.
PRx (x = 0-15)	x + 16	w	**Port n Reset Bit x** Setting this bit will reset or toggle the corresponding bit in the port output register Pn_OUT. The function of this bit is shown in **Table 9-9**.

Note: Register Pn_OMR is virtual and does not contain any flip-flop. A read action delivers the value of 0. One 8 or 16-bits write behaves as a 32-bit write padded with zeros.

Table 9-9 Function of the Bits PRx and PSx

PRx	PSx	Function
0	0	Bit Pn_OUT.Px is not changed.
0	1	Bit Pn_OUT.Px is set.
1	0	Bit Pn_OUT.Px is reset.
1	1	Bit Pn_OUT.Px is toggled.

9.3.5 Emergency Stop Register

All GPIO lines which are used by the GPTA modules (GPTA0, GPTA1, LTCA2) have an emergency stop logic implemented (see **Figure 9-1**). These GPTA related GPIO lines are:

- P0.[7:0], P1.[15:8], P2.[15:8], P3.[15:0], P4.[15:0], P5.[15:0], P8.[7:0], P9.[7:0], P13.[15:0], and P14.[15:0]

Each of these GPIO lines has its own emergency stop enable bit ENx that is located in the emergency stop register Pn_ESR of Port n. If the emergency stop signal becomes active, one of two states can be selected:

- Emergency stop function disabled (ENx = 0):
  A GPTA output line remains connected to the GPTA module (alternate function).
- Emergency stop function enabled (ENx = 1):
  A GPTA output line is disconnected from the GPTA module (alternate function) and connected to the corresponding bit of the Pn_OUT output register (the content of the corresponding PCx bit fields in register Pn_IOCR is discarded).

Pn_ESR (n=0-5)
Port n Emergency Stop Register (F000 0C50$_H$ + n*100$_H$) Reset Value: 0000 0000$_H$
Pn_ESR (n=8-9)
Port n Emergency Stop Register (F000 0C50$_H$ + n*100$_H$) Reset Value: 0000 0000$_H$
Pn_ESR (n=13-14)
Port n Emergency Stop Register (F02F F450$_H$ + n*100$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN15	EN14	EN13	EN12	EN11	EN10	EN9	EN8	EN7	EN6	EN5	EN4	EN3	EN2	EN1	EN0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
ENx (x = 0-15)	x	rw	**Emergency Stop Enable for Port n Pin x** This bit enables the emergency stop function for GPIO lines used as GPTA outputs. If the emergency stop condition is met and enabled, the output selection is automatically switched from alternate (GPTA output) function to GPIO output function. 0_B Emergency stop function for Pn.x is disabled. 1_B Emergency stop function for Pn.x is enabled.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

Note: Ports 3, 4, 5, 13, and 14 are 16-bit wide ports entirely used by the GPTA modules. The Pn_ESR registers of the other ports have a reduced number of bits (see Pn_ESR register descriptions in the corresponding port sections).

9.3.6 Port Input Register

The logic level of a GPIO pin can be read via the read-only port input register Pn_IN. Reading the Pn_IN register allways returns the current logical value at the GPIO pin independently whether the pin is selected as input or output.

Pn_IN (n=0-11)
Port n Input Register **(F000 0C24$_H$ + n*100$_H$)** **Reset Value: 0000 XXXX$_H$**
Pn_IN (n=12-16)
Port n Input Register **(F02F F424$_H$ + n*100$_H$)** **Reset Value: 0000 XXXX$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P15	**P14**	**P13**	**P12**	**P11**	**P10**	**P9**	**P8**	**P7**	**P6**	**P5**	**P4**	**P3**	**P2**	**P1**	**P0**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
Px **(x = 0-15)**	x	rh	**Port n Input Bit x** This bit indicates the level at the input pinPn.x. 0_B The input level of Pn.x is 0. 1_B The input level of Pn.x is 1.
0	[31:16]	r	**Reserved** Read as 0.

Note: Only Port 0, 1, 3, and 4 are 16-bit wide ports. The Pn_IN registers of the other ports have a reduced number of Px bits (see Pn_IN register descriptions in the corresponding port sections).

9.4 Port 0

This section describes the Port 0 functionality in detail.

9.4.1 Port 0 Configuration

Port 0 is a general-purpose 16-bit bi-directional port. It serves as GPIO lines without secondary functions.

Table 9-10 summarizes the I/O control selection functions of each Port 0 line.

9.4.2 Port 0 Function Table

Table 9-10 summarizes the I/O control selection functions of each Port 0 line.

Table 9-10 Port 0 Functions

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P0.0	I	General-purpose input	P0_IN.P0	P0_IOCR0.PC0	$0XXX_B$
			HWCFG0		
	O	General-purpose output	P0_OUT.P0		$1X00_B$
		GPTA0 output	OUT56		$1X01_B$
		GPTA1 output	OUT56		$1X10_B$
		LTCA2 output	OUT80		$1X11_B$
P0.1	I	General-purpose input	P0_IN.P1	P0_IOCR0.PC1	$0XXX_B$
			HWCFG1		
	O	General-purpose output	P0_OUT.P1		$1X00_B$
		GPTA0 output	OUT57		$1X01_B$
		GPTA1 output	OUT57		$1X10_B$
		LTCA2 output	OUT81		$1X11_B$
P0.2	I	General-purpose input	P0_IN.P2	P0_IOCR0.PC2	$0XXX_B$
			HWCFG2		
	O	General-purpose output	P0_OUT.P2		$1X00_B$
		GPTA0 output	OUT58		$1X01_B$
		GPTA1 output	OUT58		$1X10_B$
		LTCA2 output	OUT82		$1X11_B$
P0.3	I	General-purpose input	P0_IN.P3	P0_IOCR0.PC3	$0XXX_B$
			HWCFG3		
	O	General-purpose output	P0_OUT.P3		$1X00_B$
		GPTA0 output	OUT59		$1X01_B$
		GPTA1 output	OUT59		$1X10_B$
		LTCA2 output	OUT83		$1X11_B$

Table 9-10 Port 0 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P0.4	I	General-purpose input	P0_IN.P4	P0_IOCR4.PC4	$0XXX_B$
			HWCFG4		
	O	General-purpose output	P0_OUT.P4		$1X00_B$
		GPTA0 output	OUT60		$1X01_B$
		GPTA1 output	OUT60		$1X10_B$
		LTCA2 output	OUT84		$1X11_B$
P0.5	I	General-purpose input	P0_IN.P5	P0_IOCR4.PC5	$0XXX_B$
			HWCFG5		
	O	General-purpose output	P0_OUT.P5		$1X00_B$
		GPTA0 output	OUT61		$1X01_B$
		GPTA1 output	OUT61		$1X10_B$
		LTCA2 output	OUT85		$1X11_B$
P0.6	I	General-purpose input	P0_IN.P6	P0_IOCR4.PC6	$0XXX_B$
			HWCFG6		
	O	General-purpose output	P0_OUT.P6		$1X00_B$
		GPTA0 output	OUT62		$1X01_B$
		GPTA1 output	OUT62		$1X10_B$
		LTCA2 output	OUT86		$1X11_B$
P0.7	I	General-purpose input	P0_IN.P7	P0_IOCR4.PC7	$0XXX_B$
			HWCFG7		
	O	General-purpose output	P0_OUT.P7		$1X00_B$
		GPTA0 output	OUT63		$1X01_B$
		GPTA1 output	OUT63		$1X10_B$
		LTCA2 output	OUT87		$1X11_B$
P0.8	I	General-purpose input	P0_IN.P8	P0_IOCR8.PC8	$0XXX_B$
	O	General-purpose output	P0_OUT.P8		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$

Table 9-10 Port 0 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P0.9	I	General-purpose input	P0_IN.P9	P0_IOCR8.PC9	$0XXX_B$
		E-Ray	RXDA0		
	O	General-purpose output	P0_OUT.P9		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P0.10	I	General-purpose input	P0_IN.P10	P0_IOCR8. PC10	$0XXX_B$
	O	General-purpose output	P0_OUT.P10		$1X00_B$
		E-Ray	TXENA		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P0.11	I	General-purpose input	P0_IN.P11	P0_IOCR8. PC11	$0XXX_B$
	O	General-purpose output	P0_OUT.P11		$1X00_B$
		E-Ray	TXENB		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P0.12	I	General-purpose input	P0_IN.P12	P0_IOCR12. PC12	$0XXX_B$
	O	General-purpose output	P0_OUT.P12		$1X00_B$
		E-Ray	TXDB		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P0.13	I	General-purpose input	P0_IN.P13	P0_IOCR12. PC13	$0XXX_B$
		E-Ray	RXDB0		
	O	General-purpose output	P0_OUT.P13		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$

Table 9-10 Port 0 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P0.14	I	General-purpose input	P0_IN.P14	P0_IOCR12. PC14	$0XXX_B$
	O	General-purpose output	P0_OUT.P14		$1X00_B$
		E-Ray	TXDA		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P0.15	I	General-purpose input	P0_IN.P15	P0_IOCR12. PC15	$0XXX_B$
	O	General-purpose output	P0_OUT.P15		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$

9.4.3 Port 0 Registers

The following registers are available on Port 0:

Table 9-11 Port 0 Registers

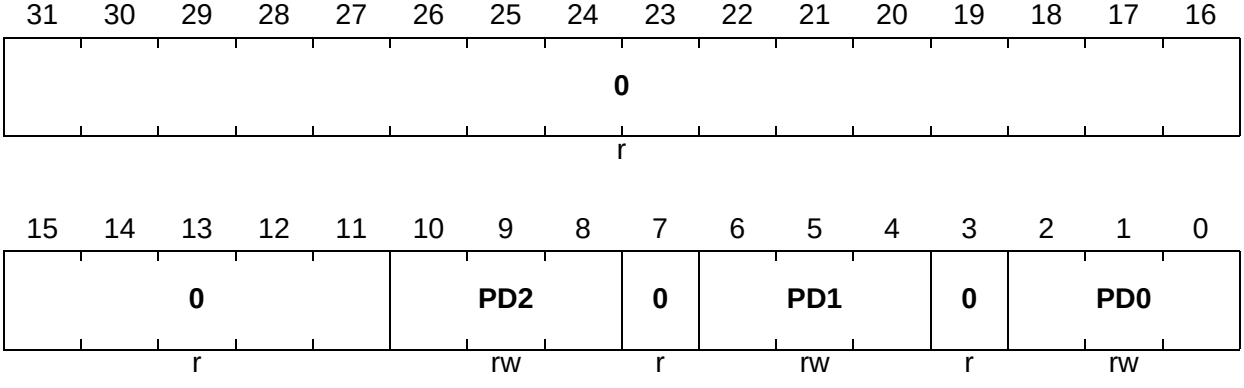
Register Short Name	Register Long Name	Offset Address	Description see
P0_OUT	Port 0 Output Register	0000_H	Page 9-15
P0_OMR	Port 0 Output Modification Register	0004_H	Page 9-16
P0_IOCR0	Port 0 Input/Output Control Register 0	0010_H	Page 9-9
P0_IOCR4	Port 0 Input/Output Control Register 4	0014_H	Page 9-10
P0_IOCR8	Port 0 Input/Output Control Register 8	0018_H	Page 9-10
P0_IOCR12	Port 0 Input/Output Control Register 12	$001C_H$	Page 9-11
P0_IN	Port 0 Input Register	0024_H	Page 9-19
P0_PDR	Port 0 Pad Driver Mode Register	0040_H	Page 9-27[1)]
P0_ESR	Port 0 Emergency Stop Register	0050_H	Page 9-18

1) This register is listed here in the Port 0 section because they differ from the general port register description given in **Section 9.3**.

9.4.3.1 Port 0 Pad Driver Mode Register and Pad Classes

The Port 0 pad driver mode register contains two bit fields that determine the pad driver mode (output driver strength and slew rate) of Port 0 line groups.

P0_PDR
Port 0 Pad Driver Mode Register (40_H) Reset Value: 0000 0000_H



Field	Bits	Type	Description
PD0	[2:0]	rw	**Pad Driver Mode for P0.[7:0]** (Class A1 pads; for coding see **Page 9-13**)
PD1	[6:4]	rw	**Pad Driver Mode for P0.[9:8], P0.13, and P0.15** (Class A1 pads; for coding see **Page 9-13**)
PD2	[10:8]	rw	**Pad Driver Mode for P0.[12:10] and P0.14** (Class A2 pads; for coding see **Page 9-13**)
0	3, 7, [31:11]	r	**Reserved** Read as 0; should be written with 0.

9.4.3.2 Port 0 Emergency Stop Register

The basic P0_ESR register functionality is described on **Page 9-18**. At Port 0, only port lines P0.[7:0] have GPTA outputs. Therefore, the P0_ESR bits EN[15:8] are not implemented. They are always read as 0 and should be written with 0.

9.5 Port 1

This section describes the Port 1 functionality in detail.

9.5.1 Port 1 Configuration

Port 1 is a 16-bit bi-directional general-purpose I/O port that can be alternatively used for the MLI0 I/O lines or for the external trigger inputs REQ[3:0] of the CPU. Furthermore, the system clock output SYSCLK is provided at Port 1.

9.5.2 Port 1 Function Table

Table 9-12 summarizes the I/O control selection functions of each Port 1 line.

Table 9-12 Port 1 Functions

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P1.0	I	General-purpose input	P1_IN.P0	P1_IOCR0.PC0	$0XXX_B$
		SCU input	REQ0		
	O	General-purpose output	P1_OUT.P0		$1X00_B$
		SCU Output	EXTCLK1		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P1.1	I	General-purpose input	P1_IN.P1	P1_IOCR0.PC1	$0XXX_B$
		SCU input	REQ1		
	O	General-purpose output	P1_OUT.P1		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P1.2	I	General-purpose input	P1_IN.P2	P1_IOCR0.PC2	$0XXX_B$
		SCU input	REQ2		
	O	General-purpose output	P1_OUT.P2		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P1.3	I	General-purpose input	P1_IN.P3	P1_IOCR0.PC3	$0XXX_B$
		SCU input	REQ3		
		MLI0 input	TREADY0B		
	O	General-purpose output	P1_OUT.P3		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$

Table 9-12 Port 1 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P1.4	I	General-purpose input	P1_IN.P4	P1_IOCR4.PC4	$0XXX_B$
	O	General-purpose output	P1_OUT.P4		$1X00_B$
		MLI0 output	TCLK0		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P1.5	I	General-purpose input	P1_IN.P5	P1_IOCR4.PC5	$0XXX_B$
		MLI0 input	TREADY0A		
	O	General-purpose output	P1_OUT.P5		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P1.6	I	General-purpose input	P1_IN.P6	P1_IOCR4.PC6	$0XXX_B$
	O	General-purpose output	P1_OUT.P6		$1X00_B$
		MLI0 output	TVALID0A		$1X01_B$
		SSC1 output	SLSO10		$1X10_B$
		Reserved	–		$1X11_B$
P1.7	I	General-purpose input	P1_IN.P7	P1_IOCR4.PC7	$0XXX_B$
	O	General-purpose output	P1_OUT.P7		$1X00_B$
		MLI0 output	TDATA0		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P1.8	I	General-purpose input	P1_IN.P8	P1_IOCR8.PC8	$0XXX_B$
		MLI0 input	RCLK0A		
	O	General-purpose output	P1_OUT.P8		$1X00_B$
		GPTA0 output	OUT64		$1X01_B$
		GPTA1 output	OUT64		$1X10_B$
		LTCA2 output	OUT88		$1X11_B$

Table 9-12 Port 1 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P1.9	I	General-purpose input	P1_IN.P9	P1_IOCR8.PC9	$0XXX_B$
	O	General-purpose output	P1_OUT.P9		$1X00_B$
		MLI0 output	RREADY0A		$1X01_B$
		SSC1 output	SLSO11		$1X10_B$
		GPTA0 output	OUT65		$1X11_B$
P1.10	I	General-purpose input	P1_IN.P10	P1_IOCR8.PC10	$0XXX_B$
		MLI0 input	RVALID0A		
	O	General-purpose output	P1_OUT.P10		$1X00_B$
		GPTA0 output	OUT66		$1X01_B$
		GPTA1 output	OUT66		$1X10_B$
		LTCA2 output	OUT90		$1X11_B$
P1.11	I	General-purpose input	P1_IN.P11	P1_IOCR8.PC11	$0XXX_B$
		MLI0 input	RDATA0A		
	O	General-purpose output	P1_OUT.P11		$1X00_B$
		GPTA0 output	OUT67		$1X01_B$
		GPTA1 output	OUT67		$1X10_B$
		LTCA2 output	OUT91		$1X11_B$
P1.12		General-purpose input	P1_IN.P12	P1_IOCR12. PC12	$0XXX_B$
	O	General-purpose output	P1_OUT.P12		$1X00_B$
		System clock output	EXTCLK0		$1X01_B$
		GPTA0 output	OUT68		$1X10_B$
		GPTA1 output	OUT68		$1X11_B$
P1.13	I	General-purpose input	P1_IN.P13	P1_IOCR12. PC13	$0XXX_B$
		MLI0 input	RCLK0B		
	O	General-purpose output	P1_OUT.P13		$1X00_B$
		GPTA0 output	OUT69		$1X01_B$
		GPTA1 output	OUT69		$1X10_B$
		LTCA2 output	OUT93		$1X11_B$

Table 9-12 Port 1 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P1.14	I	General-purpose input	P1_IN.P14	P1_IOCR12. PC14	$0XXX_B$
		MLI0 input	RVALID0B		
	O	General-purpose output	P1_OUT.P14		$1X00_B$
		GPTA0 output	OUT70		$1X01_B$
		GPTA1 output	OUT70		$1X10_B$
		LTCA2 output	OUT94		$1X11_B$
P1.15	I	General-purpose input	P1_IN.P15	P1_IOCR12. PC15	$0XXX_B$
		MLI0 input	RDATA0B		
	O	General-purpose output	P1_OUT.P15		$1X00_B$
		GPTA0 output	OUT71		$1X01_B$
		GPTA1 output	OUT71		$1X10_B$
		LTCA2 output	OUT95		$1X11_B$

9.5.3 Port 1 Registers

The following registers are available on Port 1:

Table 9-13 Port 1 Registers

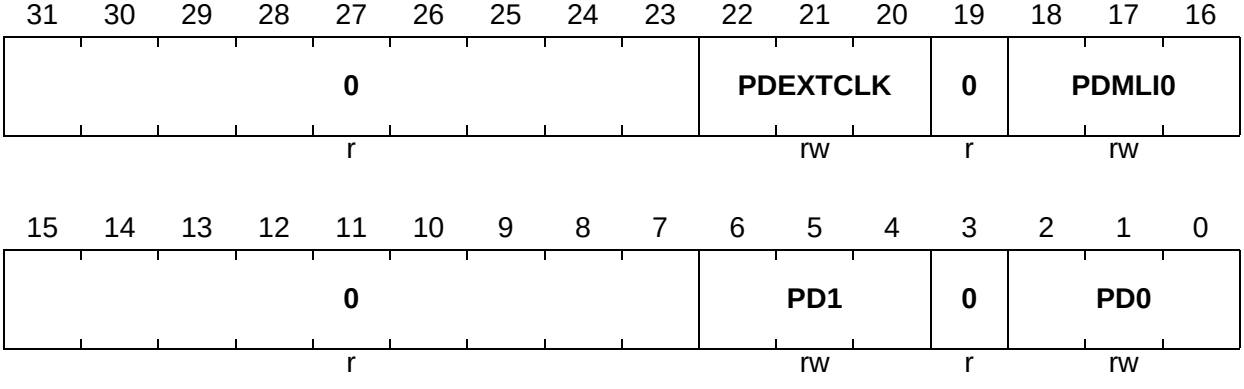
Register Short Name	Register Long Name	Offset Address	Description see
P1_OUT	Port 1 Output Register	0000_H	**Page 9-15**
P1_OMR	Port 1 Output Modification Register	0004_H	**Page 9-16**
P1_IOCR0	Port 1 Input/Output Control Register 0	0010_H	**Page 9-9**
P1_IOCR4	Port 1 Input/Output Control Register 4	0014_H	**Page 9-10**
P1_IOCR8	Port 1 Input/Output Control Register 8	0018_H	**Page 9-10**
P1_IOCR12	Port 1 Input/Output Control Register 12	$001C_H$	**Page 9-11**
P1_IN	Port 1 Input Register	0024_H	**Page 9-19**
P1_PDR	Port 1 Pad Driver Mode Register	0040_H	**Page 9-33**[1)]
P1_ESR	Port 1 Emergency Stop Register	0050_H	**Page 9-18**

1) This register is listed here in the Port 1 section because they differ from the general port register description given in **Section 9.3**.

9.5.3.1 Port 1 Pad Driver Mode Register and Pad Classes

The Port 1 pad driver mode register contains four bit fields that determine the pad driver mode (output driver strength and slew rate) of Port 1 lines and line groups. The Port 1 port lines are assigned to A1 and A2 pad classes.

P1_PDR
Port 1 Pad Driver Mode Register (40_H) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
PD0	[2:0]	rw	**Pad Driver Mode for P1.[3:1] and P1.[15:13]** (Class A1 pads; for coding see **Page 9-13**)
PD1	[6:4]	rw	**Pad Driver Mode for P1.5, P1.8, and P1.[11:10]** (Class A1 pads; for coding see **Page 9-13**)
PDMLI0	[18:16]	rw	**Pad Driver Mode for P1.4, P1.[7:6], and P1.9** (Class A2 pads; for coding see **Page 9-13**)
PDEXTCLK	[22:20]	rw	**Pad Driver Mode for P1.0 and P1.12** (Class A2 pads; for coding see **Page 9-13**)
0	3, 19, [15:7], [31:23]	r	**Reserved** Read as 0; should be written with 0.

9.5.3.2 Port 1 Emergency Stop Register

The basic P1_ESR register functionality is described on **Page 9-18**. At Port 1, only port lines P1.[15:8] have GPTA outputs. Therefore, the P1_ESR bits EN[7:0] are not implemented. They are always read as 0 and should be written with 0.

9.6 Port 2

This section describes the Port 2 functionality in detail.

9.6.1 Port 2 Configuration

Port 2 is a 14-bit bi-directional general-purpose I/O port that can be used either for the SSC0/SSC1 chip select output lines or for MSC0/MSC1 or GPTA0/GPTA1/LTCA2 I/O lines.

9.6.2 Port 2 Function Table

Table 9-14 summarizes the I/O control selection functions of each Port 2 line.

Table 9-14 Port 2 Functions

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P2.2	I	General-purpose input	P2_IN.P2	P2_IOCR0.PC2	$0XXX_B$
	O	General-purpose output	P2_OUT.P2		$1X00_B$
		SSC0 output	SLSO02		$1X01_B$
		SSC1 output	SLSO12		$1X10_B$
		SSC1 output	SLSOANDO12		$1X11_B$
P2.3	I	General-purpose input	P2_IN.P3	P2_IOCR0.PC3	$0XXX_B$
	O	General-purpose output	P2_OUT.P3		$1X00_B$
		SSC0 output	SLSO03		$1X01_B$
		SSC1 output	SLSO13		$1X10_B$
		SSC1 output	SLSOANDO13		$1X11_B$
P2.4	I	General-purpose input	P2_IN.P4	P2_IOCR4.PC4	$0XXX_B$
	O	General-purpose output	P2_OUT.P4		$1X00_B$
		SSC0 output	SLSO04		$1X01_B$
		SSC1 output	SLSO14		$1X10_B$
		SSC1 output	SLSOANDO14		$1X11_B$

Table 9-14 Port 2 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P2.5	I	General-purpose input	P2_IN.P5	P2_IOCR4.PC5	$0XXX_B$
	O	General-purpose output	P2_OUT.P5		$1X00_B$
		SSC0 output	SLSO05		$1X01_B$
		SSC1 output	SLSO15		$1X10_B$
		SSC1 output	SLSOANDO15		$1X11_B$
P2.6	I	General-purpose input	P2_IN.P6	P2_IOCR4.PC6	$0XXX_B$
	O	General-purpose output	P2_OUT.P6		$1X00_B$
		SSC0 output	SLSO06		$1X01_B$
		SSC1 output	SLSO16		$1X10_B$
		SSC1 output	SLSOANDO16		$1X11_B$
P2.7	I	General-purpose input	P2_IN.P7	P2_IOCR4.PC7	$0XXX_B$
	O	General-purpose output	P2_OUT.P7		$1X00_B$
		SSC0 output	SLSO07		$1X01_B$
		SSC1 output	SLSO17		$1X10_B$
		SSC1 output	SLSOANDO17		$1X11_B$
P2.8	I	General-purpose input	P2_IN.P8	P2_IOCR8.PC8	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN0		
	O	General-purpose output	P2_OUT.P8		$1X00_B$
		GPTA0 output	OUT0		$1X01_B$
		GPTA1 output	OUT0		$1X10_B$
		LTCA2 output	OUT0		$1X11_B$
P2.9	I	General-purpose input	P2_IN.P9	P2_IOCR8.PC9	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN1		
	O	General-purpose output	P2_OUT.P9		$1X00_B$
		GPTA0 output	OUT1		$1X01_B$
		GPTA1 output	OUT1		$1X10_B$
		LTCA2 output	OUT1		$1X11_B$

Table 9-14 Port 2 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P2.10	I	General-purpose input	P2_IN.P10	P2_IOCR8. PC10	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN2		
	O	General-purpose output	P2_OUT.P10		$1X00_B$
		GPTA0 output	OUT2		$1X01_B$
		GPTA1 output	OUT2		$1X10_B$
		LTCA2 output	OUT2		$1X11_B$
P2.11	I	General-purpose input	P2_IN.P11	P2_IOCR8. PC11	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN3		
	O	General-purpose output	P2_OUT.P11		$1X00_B$
		GPTA0 output	OUT3		$1X01_B$
		GPTA1 output	OUT3		$1X10_B$
		LTCA2 output	OUT3		$1X11_B$
P2.12	I	General-purpose input	P2_IN.P12	P2_IOCR12. PC12	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN4		
	O	General-purpose output	P2_OUT.P12		$1X00_B$
		GPTA0 output	OUT4		$1X01_B$
		GPTA1 output	OUT4		$1X10_B$
		LTCA2 output	OUT4		$1X11_B$
P2.13	I	General-purpose input	P2_IN.P13	P2_IOCR12. PC13	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN5		
	O	General-purpose output	P2_OUT.P13		$1X00_B$
		GPTA0 output	OUT5		$1X01_B$
		GPTA1 output	OUT5		$1X10_B$
		LTCA2 output	OUT5		$1X11_B$

Table 9-14 Port 2 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P2.14	I	General-purpose input	P2_IN.P14	P2_IOCR12. PC14	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN6		
	O	General-purpose output	P2_OUT.P14		$1X00_B$
		GPTA0 output	OUT6		$1X01_B$
		GPTA1 output	OUT6		$1X10_B$
		LTCA2 output	OUT6		$1X11_B$
P2.15	I	General-purpose input	P2_IN.P15	P2_IOCR12. PC15	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN7		
	O	General-purpose output	P2_OUT.P15		$1X00_B$
		GPTA0 output	OUT7		$1X01_B$
		GPTA1 output	OUT7		$1X10_B$
		LTCA2 output	OUT7		$1X11_B$

9.6.3 Port 2 Registers

The following registers are available on Port 2:

Table 9-15 Port 2 Registers

Register Short Name	Register Long Name	Offset Address	Description see
P2_OUT	Port 2 Output Register	0000_H	below[1]
P2_OMR	Port 2 Output Modification Register	0004_H	
P2_IOCR0	Port 2 Input/Output Control Register 0	0010_H	Page 9-39[1]
P2_IOCR4	Port 2 Input/Output Control Register 4	0014_H	Page 9-10
P2_IOCR8	Port 2 Input/Output Control Register 8	0018_H	Page 9-10
P2_IOCR12	Port 2 Input/Output Control Register 12	$001C_H$	Page 9-11
P2_IN	Port 2 Input Register	0024_H	Page 9-39[1]
P2_PDR	Port 2 Pad Driver Mode Register	0040_H	Page 9-40[1]
P2_ESR	Port 2 Emergency Stop Register	0050_H	Page 9-40[1]

1) These registers are listed and noted here in the Port 2 section because they differ from the general port register description given in **Section 9.3**.

9.6.3.1 Port 2 Output Register

The basic P2_OUT register functionality is described on **Page 9-15**. Port lines P2.[1:0] are not connected to port lines. Therefore, reading the P2_OUT bits P[1:0] returns the value that was last written (0 after reset). These bits can be also set/reset by the corresponding P2_OMR bits.

9.6.3.2 Port 2 Output Modification Register

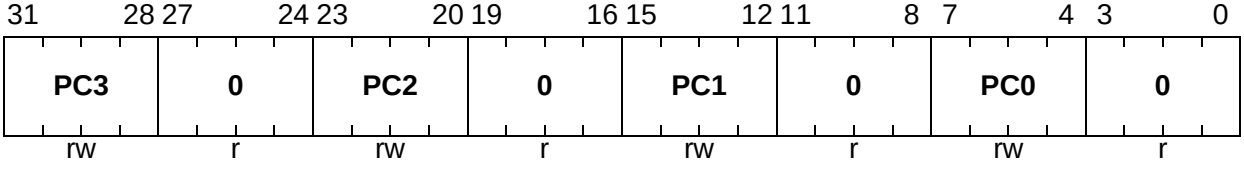
The basic P2_OMR register functionality is described on **Page 9-16**. However, port lines P2.0 and P2.1 are not available. Therefore, the P2_OMR bits PS[1:0] and PR[1:0] have no direct effect on port lines but only on register bits P2_OUT.P[1:0].

9.6.3.3 Port 2 Input/Output Control Register 0

Port lines P2.0 and P2.1 are not available. Therefore, the PC0 and PC1 bit fields in register P2_IOCR0 are not connected to any port lines.

P2_IOCR0
Port 2 Input/Output Control Register 0

(10_H) **Reset Value: 2020 2020_H**



Field	Bits	Type	Description
PC0, PC1	[7:4], [15:12]	rw	**Reserved;** read as 0010_B after reset; returns value that was written.
PC2	[23:20]	rw	**Port Control for Port 2.2** (coding see **Table 9-7** on **Page 9-12**)
PC3	[31:28]	rw	**Port Control for Port 2.3** (coding see **Table 9-7** on **Page 9-12**)
0	[3:0], [11:8], [19:16], [27:24]	r	**Reserved** Read as 0; should be written with 0.

9.6.3.4 Port 2 Input Register

The basic P2_IN register functionality is described on **Page 9-19**. However, port lines P2.0 and P2.1 are not available. Therefore, bits P0 and P1 in register P2_IN are always read as 0.

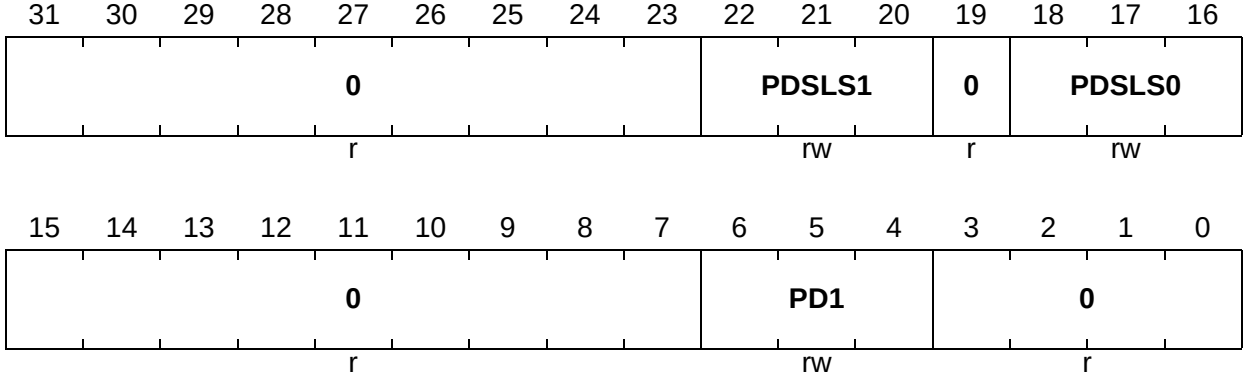
9.6.3.5 Port 2 Emergency Stop Register

The basic P2_ESR register functionality is described on **Page 9-18**. At Port 2, only port lines P2.[15:8] are connected to GPTA I/O lines. Therefore, only these port lines of Port 2 can be controlled for the emergency stop function. The P2_ESR bits EN[7:0] are not implemented. They are always read as 0 and should be written with 0.

9.6.3.6 Port 2 Pad Driver Mode Register and Pad Classes

The Port 2 pad driver mode register contains four bit fields that determine the pad driver mode (output driver strength and slew rate) of Port 2 line groups. The Port 2 port lines are assigned to A1 and A2 pad classes.

P2_PDR
Port 2 Pad Driver Mode Register (40_H) Reset Value: 0000 0000_H



Field	Bits	Type	Description
PD1	[6:4]	rw	**Pad Driver Mode for P2.[15:8]** (Class A1 pads; for coding see **Page 9-13**)
PDSLS0	[18:16]	rw	**Pad Driver Mode for P2.[3:2]** (Class A2 pads; for coding see **Page 9-13**)
PDSLS1	[22:20]	rw	**Pad Driver Mode for P2.[7:4]** (Class A2 pads; for coding see **Page 9-13**)
0	[3:0], [15:7], 19, [31:23]	r	**Reserved** Read as 0; should be written with 0.

9.7 Port 3

This section describes the Port 3 functionality in detail.

9.7.1 Port 3 Configuration

Port 3 is a 16-bit bi-directional general-purpose I/O port that can be used for the GPTA0/GPTA1/LTCA2 I/O lines.

9.7.2 Port 3 Function Table

Table 9-16 summarizes the I/O control selection functions of each Port 3 line.

Table 9-16 Port 3 Functions

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P3.0	I	General-purpose input	P3_IN.P0	P3_IOCR0.PC0	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN8		
	O	General-purpose output	P3_OUT.P0		$1X00_B$
		GPTA0 output	OUT8		$1X01_B$
		GPTA1 output	OUT8		$1X10_B$
		LTCA2 output	OUT8		$1X11_B$
P3.1	I	General-purpose input	P3_IN.P1	P3_IOCR0.PC1	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN9		
	O	General-purpose output	P3_OUT.P1		$1X00_B$
		GPTA0 output	OUT9		$1X01_B$
		GPTA1 output	OUT9		$1X10_B$
		LTCA2 output	OUT9		$1X11_B$
P3.2	I	General-purpose input	P3_IN.P2	P3_IOCR0.PC2	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN10		
	O	General-purpose output	P3_OUT.P2		$1X00_B$
		GPTA0 output	OUT10		$1X01_B$
		GPTA1 output	OUT10		$1X10_B$
		LTCA2 output	OUT10		$1X11_B$

Table 9-16 Port 3 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P3.3	I	General-purpose input	P3_IN.P3	P3_IOCR0.PC3	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN11		
	O	General-purpose output	P3_OUT.P3		$1X00_B$
		GPTA0 output	OUT11		$1X01_B$
		GPTA1 output	OUT11		$1X10_B$
		LTCA2 output	OUT11		$1X11_B$
P3.4	I	General-purpose input	P3_IN.P4	P3_IOCR4.PC4	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN12		
	O	General-purpose output	P3_OUT.P4		$1X00_B$
		GPTA0 output	OUT12		$1X01_B$
		GPTA1 output	OUT12		$1X10_B$
		LTCA2 output	OUT12		$1X11_B$
P3.5	I	General-purpose input	P3_IN.P5	P3_IOCR4.PC5	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN13		
	O	General-purpose output	P3_OUT.P5		$1X00_B$
		GPTA0 output	OUT13		$1X01_B$
		GPTA1 output	OUT13		$1X10_B$
		LTCA2 output	OUT13		$1X11_B$
P3.6	I	General-purpose input	P3_IN.P6	P3_IOCR4.PC6	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN14		
	O	General-purpose output	P3_OUT.P6		$1X00_B$
		GPTA0 output	OUT14		$1X01_B$
		GPTA1 output	OUT14		$1X10_B$
		LTCA2 output	OUT14		$1X11_B$

Table 9-16 Port 3 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P3.7	I	General-purpose input	P3_IN.P7	P3_IOCR4.PC7	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN15		
	O	General-purpose output	P3_OUT.P7		$1X00_B$
		GPTA0 output	OUT15		$1X01_B$
		GPTA1 output	OUT15		$1X10_B$
		LTCA2 output	OUT15		$1X11_B$
P3.8	I	General-purpose input	P3_IN.P8	P3_IOCR8.PC8	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN16		
	O	General-purpose output	P3_OUT.P8		$1X00_B$
		GPTA0 output	OUT16		$1X01_B$
		GPTA1 output	OUT16		$1X10_B$
		LTCA2 output	OUT16		$1X11_B$
P3.9	I	General-purpose input	P3_IN.P9	P3_IOCR8.PC9	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN17		
	O	General-purpose output	P3_OUT.P9		$1X00_B$
		GPTA0 output	OUT17		$1X01_B$
		GPTA1 output	OUT17		$1X10_B$
		LTCA2 output	OUT17		$1X11_B$
P3.10	I	General-purpose input	P3_IN.P10	P3_IOCR8. PC10	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN18		
	O	General-purpose output	P3_OUT.P10		$1X00_B$
		GPTA0 output	OUT18		$1X01_B$
		GPTA1 output	OUT18		$1X10_B$
		LTCA2 output	OUT18		$1X11_B$

Table 9-16 Port 3 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P3.11	I	General-purpose input	P3_IN.P11	P3_IOCR8. PC11	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN19		
	O	General-purpose output	P3_OUT.P11		$1X00_B$
		GPTA0 output	OUT19		$1X01_B$
		GPTA1 output	OUT19		$1X10_B$
		LTCA2 output	OUT19		$1X11_B$
P3.12	I	General-purpose input	P3_IN.P12	P3_IOCR12. PC12	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN20		
	O	General-purpose output	P3_OUT.P12		$1X00_B$
		GPTA0 output	OUT20		$1X01_B$
		GPTA1 output	OUT20		$1X10_B$
		LTCA2 output	OUT20		$1X11_B$
P3.13	I	General-purpose input	P3_IN.P13	P3_IOCR12. PC13	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN21		
	O	General-purpose output	P3_OUT.P13		$1X00_B$
		GPTA0 output	OUT21		$1X01_B$
		GPTA1 output	OUT21		$1X10_B$
		LTCA2 output	OUT21		$1X11_B$
P3.14	I	General-purpose input	P3_IN.P14	P3_IOCR12. PC14	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN22		
	O	General-purpose output	P3_OUT.P14		$1X00_B$
		GPTA0 output	OUT22		$1X01_B$
		GPTA1 output	OUT22		$1X10_B$
		LTCA2 output	OUT22		$1X11_B$

Table 9-16 Port 3 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P3.15	I	General-purpose input	P3_IN.P15	P3_IOCR12. PC15	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN23		
	O	General-purpose output	P3_OUT.P15		$1X00_B$
		GPTA0 output	OUT23		$1X01_B$
		GPTA1 output	OUT23		$1X10_B$
		LTCA2 output	OUT23		$1X11_B$

9.7.3 Port 3 Registers

The following registers are available on Port 3:

Table 9-17 Port 3 Registers

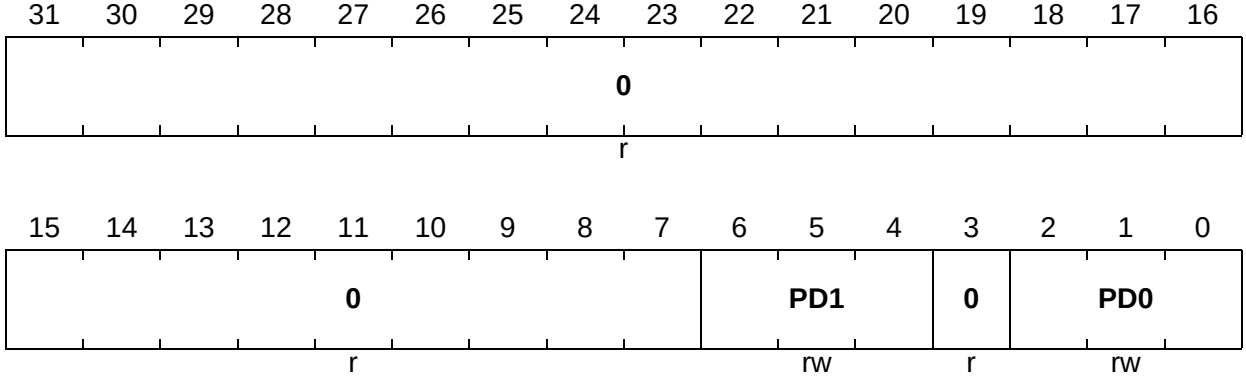
Register Short Name	Register Long Name	Offset Address	Description see
P3_OUT	Port 3 Output Register	0000_H	Page 9-15
P3_OMR	Port 3 Output Modification Register	0004_H	Page 9-16
P3_IOCR0	Port 3 Input/Output Control Register 0	0010_H	Page 9-9
P3_IOCR4	Port 3 Input/Output Control Register 4	0014_H	Page 9-10
P3_IOCR8	Port 3 Input/Output Control Register 8	0018_H	Page 9-10
P3_IOCR12	Port 3 Input/Output Control Register 12	$001C_H$	Page 9-11
P3_IN	Port 3 Input Register	0024_H	Page 9-19
P3_PDR	Port 3 Pad Driver Mode Register	0040_H	Page 9-47[1)]
P3_ESR	Port 3 Emergency Stop Register	0050_H	Page 9-18

1) This register is listed here in the Port 3 section because they differ from the general port register description given in **Section 9.3**.

9.7.3.1 Port 3 Pad Driver Mode Register and Pad Classes

The Port 3 pad driver mode register contains two bit fields that determine the pad driver mode (output driver strength and slew rate) of Port 3 line groups. The Port 3 port lines are all of pad class A1.

P3_PDR
Port 3 Pad Driver Mode Register (40_H) Reset Value: 0000 0000_H



Field	Bits	Type	Description
PD0	[2:0]	rw	**Pad Driver Mode for P3.[7:0]** (Class A1 pads; for coding see **Page 9-13**)
PD1	[6:4]	rw	**Pad Driver Mode for P3.[15:8]** (Class A1 pads; for coding see **Page 9-13**)
0	3, [31:7]	r	**Reserved** Read as 0; should be written with 0.

9.8 Port 4

This section describes the Port 4 functionality in detail.

9.8.1 Port 4 Configuration

Port 4 is a 16-bit bi-directional general-purpose I/O port that can be used for the GPTA0/GPTA1/LTCA2 I/O lines.

9.8.2 Port 4 Function Table

Table 9-18 summarizes the I/O control selection functions of each Port 4 line.

Table 9-18 Port 4 Functions

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P4.0	I	General-purpose input	P4_IN.P0	P4_IOCR0.PC0	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN24		
	O	General-purpose output	P4_OUT.P0		$1X00_B$
		GPTA0 output	OUT24		$1X01_B$
		GPTA1 output	OUT24		$1X10_B$
		LTCA2 output	OUT24		$1X11_B$
P4.1	I	General-purpose input	P4_IN.P1	P4_IOCR0.PC1	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN25		
	O	General-purpose output	P4_OUT.P1		$1X00_B$
		GPTA0 output	OUT25		$1X01_B$
		GPTA1 output	OUT25		$1X10_B$
		LTCA2 output	OUT25		$1X11_B$
P4.2	I	General-purpose input	P4_IN.P2	P4_IOCR0.PC2	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN26		
	O	General-purpose output	P4_OUT.P2		$1X00_B$
		GPTA0 output	OUT26		$1X01_B$
		GPTA1 output	OUT26		$1X10_B$
		LTCA2 output	OUT26		$1X11_B$

Table 9-18 Port 4 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P4.3	I	General-purpose input	P4_IN.P3	P4_IOCR0.PC3	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN27		
	O	General-purpose output	P4_OUT.P3		$1X00_B$
		GPTA0 output	OUT27		$1X01_B$
		GPTA1 output	OUT27		$1X10_B$
		LTCA2 output	OUT27		$1X11_B$
P4.4	I	General-purpose input	P4_IN.P4	P4_IOCR4.PC4	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN28		
	O	General-purpose output	P4_OUT.P4		$1X00_B$
		GPTA0 output	OUT28		$1X01_B$
		GPTA1 output	OUT28		$1X10_B$
		LTCA2 output	OUT28		$1X11_B$
P4.5	I	General-purpose input	P4_IN.P5	P4_IOCR4.PC5	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN29		
	O	General-purpose output	P4_OUT.P5		$1X00_B$
		GPTA0 output	OUT29		$1X01_B$
		GPTA1 output	OUT29		$1X10_B$
		LTCA2 output	OUT29		$1X11_B$
P4.6	I	General-purpose input	P4_IN.P6	P4_IOCR4.PC6	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN30		
	O	General-purpose output	P4_OUT.P6		$1X00_B$
		GPTA0 output	OUT30		$1X01_B$
		GPTA1 output	OUT30		$1X10_B$
		LTCA2 output	OUT30		$1X11_B$

Table 9-18 Port 4 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P4.7	I	General-purpose input	P4_IN.P7	P4_IOCR4.PC7	$0XXX_B$
		GPTA0/GPTA1/LTCA2 input	IN31		
	O	General-purpose output	P4_OUT.P7		$1X00_B$
		GPTA0 output	OUT31		$1X01_B$
		GPTA1 output	OUT31		$1X10_B$
		LTCA2 output	OUT31		$1X11_B$
P4.8	I	General-purpose input	P4_IN.P8	P4_IOCR8.PC8	$0XXX_B$
		GPTA0/GPTA1input	IN32		
	O	General-purpose output	P4_OUT.P8		$1X00_B$
		GPTA0 output	OUT32		$1X01_B$
		GPTA1 output	OUT32		$1X10_B$
		LTCA2 output	OUT0		$1X11_B$
P4.9	I	General-purpose input	P4_IN.P9	P4_IOCR8.PC9	$0XXX_B$
		GPTA0/GPTA1input	IN33		
	O	General-purpose output	P4_OUT.P9		$1X00_B$
		GPTA0 output	OUT33		$1X01_B$
		GPTA1 output	OUT33		$1X10_B$
		LTCA2 output	OUT1		$1X11_B$
P4.10	I	General-purpose input	P4_IN.P10	P4_IOCR8. PC10	$0XXX_B$
		GPTA0/GPTA1 input	IN34		
	O	General-purpose output	P4_OUT.P10		$1X00_B$
		GPTA0 output	OUT34		$1X01_B$
		GPTA1 output	OUT34		$1X10_B$
		LTCA2 output	OUT2		$1X11_B$

Table 9-18 Port 4 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P4.11	I	General-purpose input	P4_IN.P11	P4_IOCR8. PC11	$0XXX_B$
		GPTA0/GPTA1 input	IN35		
	O	General-purpose output	P4_OUT.P11		$1X00_B$
		GPTA0 output	OUT35		$1X01_B$
		GPTA1 output	OUT35		$1X10_B$
		LTCA2 output	OUT3		$1X11_B$
P4.12	I	General-purpose input	P4_IN.P12	P4_IOCR12. PC12	$0XXX_B$
		GPTA0/GPTA1 input	IN36		
	O	General-purpose output	P4_OUT.P12		$1X00_B$
		GPTA1 output	OUT36		$1X10_B$
		GPTA0 output	OUT36		$1X01_B$
		LTCA2 output	OUT4		$1X11_B$
P4.13	I	General-purpose input	P4_IN.P13	P4_IOCR12. PC13	$0XXX_B$
		GPTA0/GPTA1 input	IN37		
	O	General-purpose output	P4_OUT.P13		$1X00_B$
		GPTA0 output	OUT37		$1X01_B$
		GPTA1 output	OUT37		$1X10_B$
		LTCA2 output	OUT5		$1X11_B$
P4.14	I	General-purpose input	P4_IN.P14	P4_IOCR12. PC14	$0XXX_B$
		GPTA0/GPTA1 input	IN38		
	O	General-purpose output	P4_OUT.P14		$1X00_B$
		GPTA0 output	OUT38		$1X01_B$
		GPTA1 output	OUT38		$1X10_B$
		LTCA2 output	OUT6		$1X11_B$

Table 9-18 Port 4 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P4.15	I	General-purpose input	P4_IN.P15	P4_IOCR12. PC15	$0XXX_B$
		GPTA0/GPTA1 input	IN39		
	O	General-purpose output	P4_OUT.P15		$1X00_B$
		GPTA0 output	OUT39		$1X01_B$
		GPTA1 output	OUT39		$1X10_B$
		LTCA2 output	OUT7		$1X11_B$

9.8.3 Port 4 Registers

The following registers are available on Port 4:

Table 9-19 Port 4 Registers

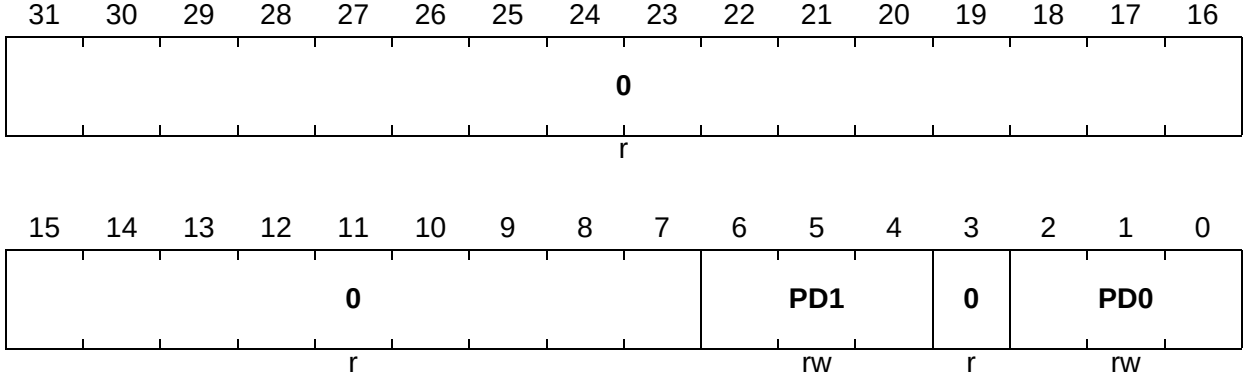
Register Short Name	Register Long Name	Offset Address	Description see
P4_OUT	Port 4 Output Register	0000_H	**Page 9-15**
P4_OMR	Port 4 Output Modification Register	0004_H	**Page 9-16**
P4_IOCR0	Port 4 Input/Output Control Register 0	0010_H	**Page 9-9**
P4_IOCR4	Port 4 Input/Output Control Register 4	0014_H	**Page 9-10**
P4_IOCR8	Port 4 Input/Output Control Register 8	0018_H	**Page 9-10**
P4_IOCR12	Port 4 Input/Output Control Register 12	$001C_H$	**Page 9-11**
P4_IN	Port 4 Input Register	0024_H	**Page 9-19**
P4_PDR	Port 4 Pad Driver Mode Register	0040_H	**Page 9-53**[1)]
P4_ESR	Port 4 Emergency Stop Register	0050_H	**Page 9-18**

1) This register is listed here in the Port 4 section because they differ from the general port register description given in **Section 9.3**.

9.8.3.1 Port 4 Pad Driver Mode Register and Pad Classes

The Port 4 pad driver mode register contains two bit fields that determine the pad driver mode (output driver strength and slew rate) of Port 4 lines and line groups. The Port 4 port lines are assigned to A1 and A2 pad classes.

P4_PDR
Port 4 Pad Driver Mode Register (40_H) Reset Value: 0000 0000_H



Field	Bits	Type	Description
PD0	[2:0]	rw	**Pad Driver Mode for P4.[7:0]** (Class A2 pads; for coding see **Page 9-13**)
PD1	[6:4]	rw	**Pad Driver Mode for P4.[15:8]** (Class A1 pads; for coding see **Page 9-13**)
0	3, [31:7]	r	**Reserved** Read as 0; should be written with 0.

9.9 Port 5

Port 5 is an 16-bit GPIO port. Pins associated to it be used in two ways:

- as a CMOS Port where each pin outputs one signal, as any other port (only exception - no open drain mode available), and
- as an output LVDS port where a pin pair (two pins) outputs one differential MSC signal.

The switching between the two modes is done via the PDR register.

The swithcing between Input/Ouput and Pull-Up/Pull-Down control is done via the IOCR register.

Attention: In the LVDS mode the IOCR.PCx bit field of each pin of the LVDS pair must be programed as output, that is $1xxx_B$.

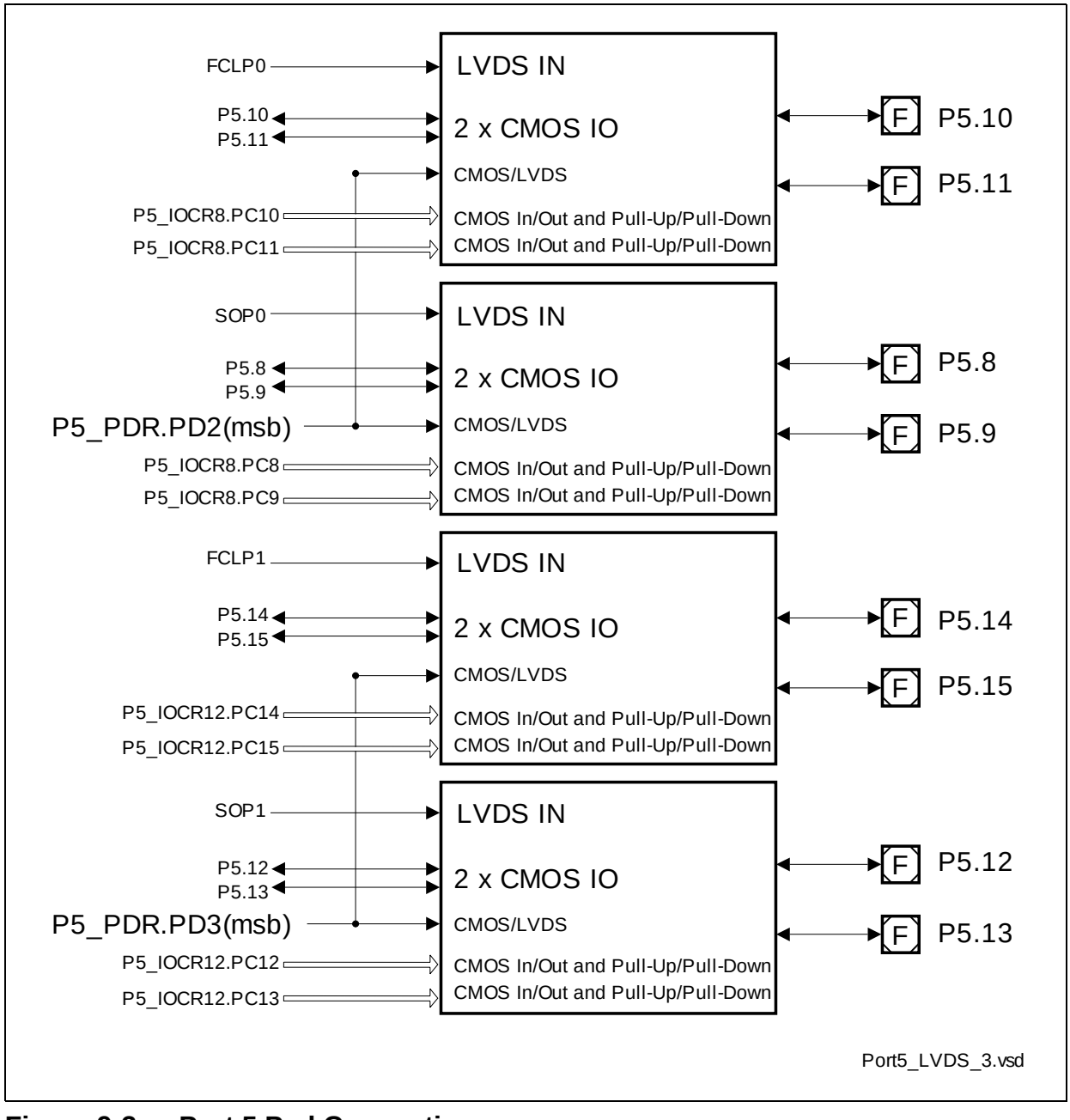


Figure 9-3 Port 5 Pad Connections

Note: The following constraint applies to an LVDS pair used in CMOS mode: only one pin of a pair should be used as output, the other should be used as input, or both pins should be used as inputs. Using both pins as outputs is not recommended because of high crosstalk between them.

9.9.1 Port 5 Configuration

Port 5 is an bi-directional general-purpose I/O port which can be used for the ASC0/ASC1, MSC0/MSC1, or MLI0 interface I/O lines.

9.9.2 Port 5 Function Table

Table 9-20 summarizes the I/O control selection functions of each Port 5 line.

Table 9-20 Port 5 Functions

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P5.0	I	General-purpose input	P5_IN.P0	P5_IOCR0.PC0	$0XXX_B$
		ASC0 input	RXD0A		
	O	General-purpose output	P5_OUT.P0		$1X00_B$
		ASC0 output (sync. mode)	RXD0A		$1X01_B$
		GPTA0 output	OUT72		$1X10_B$
		GPTA1 output	OUT72		$1X11_B$
P5.1	I	General-purpose input	P5_IN.P1	P5_IOCR0.PC1	$0XXX_B$
	O	General-purpose output	P5_OUT.P1		$1X00_B$
		ASC0 output	TXD0		$1X01_B$
		GPTA0 output	OUT73		$1X10_B$
		GPTA1 output	OUT73		$1X11_B$
P5.2	I	General-purpose input	P5_IN.P2	P5_IOCR0.PC2	$0XXX_B$
		ASC1 input	RXD1A		
	O	General-purpose output	P5_OUT.P2		$1X00_B$
		ASC1 output (sync. mode)	RXD1A		$1X01_B$
		GPTA0 output	OUT74		$1X10_B$
		GPTA1 output	OUT74		$1X11_B$
P5.3	I	General-purpose input	P5_IN.P3	P5_IOCR0.PC3	$0XXX_B$
	O	General-purpose output	P5_OUT.P3		$1X00_B$
		ASC1 output	TXD1		$1X01_B$
		GPTA0 output	OUT75		$1X10_B$
		GPTA1 output	OUT75		$1X11_B$

Table 9-20 Port 5 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P5.4	I	General-purpose input	P5_IN.P4	P5_IOCR4.PC4	$0XXX_B$
	O	General-purpose output	P5_OUT.P4		$1X00_B$
		MSC0 output	EN00		$1X01_B$
		MLI0 output	RREADY0B		$1X10_B$
		GPTA0 output	OUT76		$1X11_B$
P5.5	I	General-purpose input	P5_IN.P5	P5_IOCR4.PC5	$0XXX_B$
		MSC0 input	SDI0		
	O	General-purpose output	P5_OUT.P5		$1X00_B$
		GPTA0 output	OUT77		$1X01_B$
		GPTA1 output	OUT77		$1X10_B$
		LTCA2 output	OUT101		$1X11_B$
P5.6	I	General-purpose input	P5_IN.P6	P5_IOCR4.PC6	$0XXX_B$
	O	General-purpose output	P5_OUT.P6		$1X00_B$
		MSC1 output	EN10		$1X01_B$
		MLI0 output	TVALID0B		$1X10_B$
		GPTA0 output	OUT78		$1X11_B$
P5.7	I	General-purpose input	P5_IN.P7	P5_IOCR4.PC7	$0XXX_B$
		MSC1 input	SDI1		
	O	General-purpose output	P5_OUT.P7		$1X00_B$
		GPTA0 output	OUT79		$1X01_B$
		GPTA1 output	OUT79		$1X10_B$
		LTCA2 output	OUT103		$1X11_B$
P5.8	I	General-purpose input	P5_IN.P8	P5_IOCR8.PC8	$0XXX_B$
	O	General-purpose output	P5_OUT.P8		$1X00_B$
		MSC0 Output	SON0		$1X01_B$
		GPTA0 output	OUT80		$1X10_B$
		GPTA1 output	OUT80		$1X11_B$

Table 9-20 Port 5 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P5.9	I	General-purpose input	P5_IN.P9	P5_IOCR8.PC9	$0XXX_B$
	O	General-purpose output	P5_OUT.P9		$1X00_B$
		MSC0 Output	SOP0A		$1X01_B$
		GPTA0 output	OUT81		$1X10_B$
		GPTA1 output	OUT81		$1X11_B$
P5.10	I	General-purpose input	P5_IN.P10	P5_IOCR8. PC10	$0XXX_B$
	O	General-purpose output	P5_OUT.P10		$1X00_B$
		MSC0 Output	FCLN0		$1X01_B$
		GPTA0 output	OUT82		$1X10_B$
		GPTA1 output	OUT82		$1X11_B$
P5.11	I	General-purpose input	P5_IN.P11	P5_IOCR8. PC11	$0XXX_B$
	O	General-purpose output	P5_OUT.P11		$1X00_B$
		MSC0 Output	FCLP0A		$1X01_B$
		GPTA0 output	OUT83		$1X10_B$
		GPTA1 output	OUT83		$1X11_B$
P5.12	I	General-purpose input	P5_IN.P12	P5_IOCR12. PC12	$0XXX_B$
	O	General-purpose output	P5_OUT.P12		$1X00_B$
		MSC1 Output	SON1		$1X01_B$
		GPTA0 output	OUT84		$1X10_B$
		GPTA1 output	OUT84		$1X11_B$
P5.13	I	General-purpose input	P5_IN.P13	P5_IOCR12. PC13	$0XXX_B$
	O	General-purpose output	P5_OUT.P13		$1X00_B$
		MSC1 Output	SOP1A		$1X01_B$
		GPTA0 output	OUT85		$1X10_B$
		GPTA1 output	OUT85		$1X11_B$

Table 9-20 Port 5 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P5.14	I	General-purpose input	P5_IN.P14	P5_IOCR12. PC14	$0XXX_B$
	O	General-purpose output	P5_OUT.P14		$1X00_B$
		MSC1 Output	FCLN1		$1X01_B$
		GPTA0 output	OUT86		$1X10_B$
		GPTA1 output	OUT86		$1X11_B$
P5.15	I	General-purpose input	P5_IN.P15	P5_IOCR12. PC15	$0XXX_B$
	O	General-purpose output	P5_OUT.P15		$1X00_B$
		MSC1 Output	FCLP1A		$1X01_B$
		GPTA0 output	OUT87		$1X10_B$
		GPTA1 output	OUT87		$1X11_B$

9.9.3 Port 5 Registers

The following registers are available on Port 5:

Table 9-21 Port 5 Registers

Register Short Name	Register Long Name	Offset Address	Description see
P5_OUT	Port 5 Output Register	0000_H	below[1]
P5_OMR	Port 5 Output Modification Register	0004_H	
P5_IOCR0	Port 5 Input/Output Control Register 0	0010_H	Page 9-10
P5_IOCR4	Port 5 Input/Output Control Register 4	0014_H	Page 9-10
P5_IOCR8	Port 5 Input/Output Control Register 8	0018_H	Page 9-10
P5_IOCR12	Port 5 Input/Output Control Register 12	$001C_H$	Page 9-10
P5_IN	Port 5 Input Register	0024_H	below[1]
P5_PDR	Port 5 Pad Driver Mode Register	0040_H	Page 9-61[1]
P5_ESR	Port 5 Emergency Stop Register	0050_H	Page 9-18

1) These registers are noted here in the Port 5 section because they differ from the general port register description given in Section 9.3.

9.9.3.1 Port 5 Output Register

The basic P5_OUT register functionality is described on Page 9-15.

9.9.3.2 Port 5 Output Modification Register

The basic P5_OMR register functionality is described on Page 9-16.

9.9.3.3 Port 5 Input Register

The basic P5_IN register functionality is described on Page 9-19.

9.9.3.4 Port 5 Pad Driver Mode Register and Pad Classes

The Port 5 pad driver mode register contains four bit fields that determine the pad driver mode (output driver strength and slew rate) of Port 5 line groups.

P5_PDR
Port 5 Pad Driver Mode Register (40_H) Reset Value: 0000 0000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	PDMSC1			0	PDMSC0			0	PDASC1			0	PDASC0		
r	rw			r	rw			r	rw			r	rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	PD3			0	PD2			0							
r	rw			r	rw			r							

Field	Bits	Type	Description
PD2	[10:8]	rw	**Pad Driver Mode for P5.[11:8]** The msb of PD2 switches between CMOS and LVDS pads. Default CMOS input. $0XX_B$ CMOS Input or Output (depending on the IOCR setting) $1XX_B$ LVDS Output (pull-up/pull-down must be switched off via IOCR)
PD3	[14:12]	rw	**Pad Driver Mode for P6.[15:12]** The msb of PD3 switches between CMOS and LVDS pads. Default CMOS input. $0XX_B$ CMOS Input or Output (depending on the IOCR setting) $1XX_B$ LVDS Output (pull-up/pull-down must be switched off via IOCR)
PDASC0	[18:16]	rw	**Pad Driver Mode for P5.[1:0]** (Class A2 pads; for coding see **Page 9-13**)
PDASC1	[22:20]	rw	**Pad Driver Mode for P5.[3:2]** (Class A2 pads; for coding see **Page 9-13**)

Field	Bits	Type	Description
PDMSC0	[26:24]	rw	**Pad Driver Mode for P5.[5:4]** (Class A2 pads; for coding see **Page 9-13**)
PDMSC1	[30:28]	rw	**Pad Driver Mode for P5.[7:6]** (Class A2 pads; for coding see **Page 9-13**)
0	[7:0], 11, 15, 19, 23, 27, 31	r	**Reserved** Read as 0; should be written with 0.

9.9.3.5 Port 5 Emergency Stop Register

The basic P5_ESR register functionality is described on **Page 9-18**. At Port 5, all port lines P5.[15:0] have GPTA outputs and correspondent ESR lines.

9.10 Port 6

This section describes the Port 6 functionality in detail.

9.10.1 Port 6 Configuration

Port 6 is a 12-bit bi-directional general-purpose I/O port which can be used for the SSC1, ASC0/ASC1, or for the MultiCAN controller I/O lines.

9.10.2 Port 6 Function Table

Table 9-22 summarizes the I/O control selection functions of each Port 6 line.

Table 9-22 Port 6 Functions

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P6.4	I	General-purpose input	P6_IN.P4	P6_IOCR4.PC4	$0XXX_B$
		SSC1 input (slave mode)	MTSR1		
	O	General-purpose output	P6_OUT.P4		$1X00_B$
		SSC1 output (master mode)	MTSR1		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P6.5	I	General-purpose input	P6_IN.P5	P6_IOCR4.PC5	$0XXX_B$
		SSC1 input (master mode)	MRST1		
	O	General-purpose output	P6_OUT.P5		$1X00_B$
		SSC1 output (slave mode)	MRST1		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P6.6	I	General-purpose input	P6_IN.P6	P6_IOCR4.PC6	$0XXX_B$
		SSC1 input (slave mode)	SCLK1		
	O	General-purpose output	P6_OUT.P6		$1X00_B$
		SSC1 output (master mode)	SCLK1		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P6.7	I	General-purpose input	P6_IN.P7	P6_IOCR4.PC7	$0XXX_B$
		SSC1 input	$\overline{\text{SLSI1}}$		
	O	General-purpose output	P6_OUT.P7		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$

Table 9-22 Port 6 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P6.8	I	General-purpose input	P6_IN.P8	P6_IOCR8.PC8	$0XXX_B$
		CAN node 0 rec. input 0 CAN node 3 rec. input 1	RXDCAN0		
		ASC0 input	RXD0B		
	O	General-purpose output	P6_OUT.P8		$1X00_B$
		Reserved	–		$1X01_B$
		ASC0 output (sync. mode)	RXD0B		$1X10_B$
		Reserved	–		$1X11_B$
P6.9	I	General-purpose input	P6_IN.P9	P6_IOCR8.PC9	$0XXX_B$
	O	General-purpose output	P6_OUT.P9		$1X00_B$
		CAN node 0 output	TXDCAN0		$1X01_B$
		ASC0 output	TXD0		$1X10_B$
		Reserved	–		$1X11_B$
P6.10	I	General-purpose input	P6_IN.P10	P6_IOCR8. PC10	$0XXX_B$
		CAN node 1 rec. input 0 CAN node 0 rec. input 1	RXDCAN1		
		ASC1 input	RXD1B		
	O	General-purpose output	P6_OUT.P10		$1X00_B$
		Reserved	–		$1X01_B$
		ASC1 output (sync. mode)	RXD1B		$1X10_B$
		E-Ray output	TXENA		$1X11_B$
P6.11	I	General-purpose input	P6_IN.P11	P6_IOCR8. PC11	$0XXX_B$
	O	General-purpose output	P6_OUT.P11		$1X00_B$
		CAN node 1 output	TXDCAN1		$1X01_B$
		ASC1 output	TXD1		$1X10_B$
		E-Ray output	TXENB		$1X11_B$

Table 9-22 Port 6 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P6.12	I	General-purpose input	P6_IN.P12	P6_IOCR12. PC12	$0XXX_B$
		CAN node 2 rec. input 0 CAN node 1 rec. input 1	RXDCAN2		
		E-Ray input	RXDA1		
	O	General-purpose output	P6_OUT.P12		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P6.13	I	General-purpose input	P6_IN.P13	P6_IOCR12. PC13	$0XXX_B$
	O	General-purpose output	P6_OUT.P13		$1X00_B$
		CAN node 2 output	TXDCAN2		$1X01_B$
		E-Ray output	TXDA		$1X10_B$
		Reserved	–		$1X11_B$
P6.14	I	General-purpose input	P6_IN.P14	P6_IOCR12. PC14	$0XXX_B$
		CAN node 3 rec. input 0 CAN node 2 rec. input 1	RXDCAN3		
		E-Ray Input	RXDB1		
	O	General-purpose output	P6_OUT.P14		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P6.15	I	General-purpose input	P6_IN.P15	P6_IOCR12. PC15	$0XXX_B$
	O	General-purpose output	P6_OUT.P15		$1X00_B$
		CAN node 3 output	TXDCAN3		$1X01_B$
		E-Ray output	TXDB		$1X10_B$
		Reserved	–		$1X11_B$

9.10.3 Port 6 Registers

The following registers are available on Port 6:

Table 9-23 Port 6 Registers

Register Short Name	Register Long Name	Offset Address	Description see
P6_OUT	Port 6 Output Register	0000_H	below[1]
P6_OMR	Port 6 Output Modification Register	0004_H	
P6_IOCR4	Port 6 Input/Output Control Register 4	0014_H	Page 9-10
P6_IOCR8	Port 6 Input/Output Control Register 8	0018_H	Page 9-10
P6_IOCR12	Port 6 Input/Output Control Register 12	$001C_H$	Page 9-11
P6_IN	Port 6 Input Register	0024_H	below[1]
P6_PDR	Port 6 Pad Driver Mode Register	0040_H	Page 9-68[1]

1) These registers are listed and noted here in the Port 6 section because they differ from the general port register description given in Section 9.3.

9.10.3.1 Port 6 Output Register

The basic P6_OUT register functionality is described on Page 9-15. Port lines P6.[3:0] are not connected to port lines. Therefore, reading the P6_OUT bits P[3:0] returns the value that was last written (0 after reset). These bits can be also set/reset by the corresponding P6_OMR bits.

9.10.3.2 Port 6 Output Modification Register

The basic P6_OMR register functionality is described on Page 9-16. Port lines P6.[3:0] are not available. Therefore, they are not implemented. These bits should always be written with 0.

9.10.3.3 Port 6 Input Register

The basic P6_IN register functionality is described on Page 9-19. Port lines P6.[3:0] are not available. Therefore, the P6_IN bits P[3:0] are always read as 0.

9.10.3.4 Port 6 Pad Driver Mode Register and Pad Classes

The Port 6 pad driver mode register contains three bit fields that determine the pad driver mode (output driver strength and slew rate) of Port 6 line groups.

P6_PDR
Port 6 Pad Driver Mode Register (40_H) Reset Value: 0000 0000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	PDCAN23			0	PDCAN01			0	PDSSC1			0	0		
r	rw			r	rw			r	rw			r	rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0													PD0		
r													rw		

Field	Bits	Type	Description
PD0	[2:0]	rw	**Pad Driver Mode for P6.14 and P6.12** (Class A1 pads; for coding see **Page 9-13**)
PDSSC1	[22:20]	rw	**Pad Driver Mode for P6.[7:4]** (Class A2 pads; for coding see **Page 9-13**)
PDCAN01	[26:24]	rw	**Pad Driver Mode for P6.[11:8]** (Class A2 pads; for coding see **Page 9-13**)
PDCAN23	[30:28]	rw	**Pad Driver Mode for P6.15 and P6.13** (Class A2 pads; for coding see **Page 9-13**)
0	[19:3], 23, 27, 31	r	**Reserved** Read as 0; should be written with 0.

9.11 Port 7

This section describes the Port 7 functionality in detail.

9.11.1 Port 7 Configuration

Port 7 is an 8-bit bi-directional general-purpose I/O port that can be used for the external trigger input lines REQ[7:4] or for the ADC0/ADC1 external multiplexer control output lines.

9.11.2 Port 7 Function Table

Table 9-24 summarizes the I/O control selection functions of each Port 7 line.

Table 9-24 Port 7 Functions

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P7.0	I	General-purpose input	P7_IN.P0	P7_IOCR0.PC0	$0XXX_B$
		SCU input	REQ4		
	O	General-purpose output	P7_OUT.P0		$1X00_B$
		ADC2 output	AD2EMUX2		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P7.1	I	General-purpose input	P7_IN.P1	P7_IOCR0.PC1	$0XXX_B$
		SCU input	REQ5		
	O	General-purpose output	P7_OUT.P1		$1X00_B$
		ADC0 output	AD0EMUX2		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P7.2	I	General-purpose input	P7_IN.P2	P7_IOCR0.PC2	$0XXX_B$
	O	General-purpose output	P7_OUT.P2		$1X00_B$
		ADC0 output	AD0EMUX0		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P7.3	I	General-purpose input	P7_IN.P3	P7_IOCR0.PC3	$0XXX_B$
	O	General-purpose output	P7_OUT.P3		$1X00_B$
		ADC0 output	AD0EMUX1		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$

Table 9-24 Port 7 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P7.4	I	General-purpose input	P7_IN.P4	P7_IOCR4.PC4	$0XXX_B$
		SCU input	REQ6		
	O	General-purpose output	P7_OUT.P4		$1X00_B$
		ADC2 output	AD2EMUX0		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P7.5	I	General-purpose input	P7_IN.P5	P7_IOCR4.PC5	$0XXX_B$
		SCU input	REQ7		
		TTCAN input ECTT2			
	O	General-purpose output	P7_OUT.P5		$1X00_B$
		ADC2 output	AD2EMUX1		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P7.6	I	General-purpose input	P7_IN.P6	P7_IOCR4.PC6	$0XXX_B$
	O	General-purpose output	P7_OUT.P6		$1X00_B$
		ADC1 output	AD1EMUX0		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P7.7	I	General-purpose input	P7_IN.P7	P7_IOCR4.PC7	$0XXX_B$
	O	General-purpose output	P7_OUT.P7		$1X00_B$
		ADC1 output	AD1EMUX1		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$

9.11.3 Port 7 Registers

The following registers are available on Port 7:

Table 9-25 Port 7 Registers

Register Short Name	Register Long Name	Offset Address	Description see
P7_OUT	Port 7 Output Register	0000_H	below[1]
P7_OMR	Port 7 Output Modification Register	0004_H	
P7_IOCR0	Port 7 Input/Output Control Register 0	0010_H	Page 9-9
P7_IOCR4	Port 7 Input/Output Control Register 4	0014_H	Page 9-10
P7_IN	Port 7 Input Register	0024_H	below[1]
P7_PDR	Port 7 Pad Driver Mode Register	0040_H	Page 9-73[1]

1) These registers are listed and noted here in the Port 7 section because they differ from the general port register description given in Section 9.3.

9.11.3.1 Port 7 Output Register

The basic P7_OUT register functionality is described on Page 9-15. Port lines P7.[15:8] are not available. Therefore, the P7_OUT bits P[15:8] should be written with 0, and are always read as 0.

9.11.3.2 Port 7 Output Modification Register

The basic P7_OMR register functionality is described on Page 9-16. Port lines P7.[15:8] are not available. Therefore, the P7_OMR bits PS[15:8] and PR[15:8] are not implemented. These bits should always be written with 0.

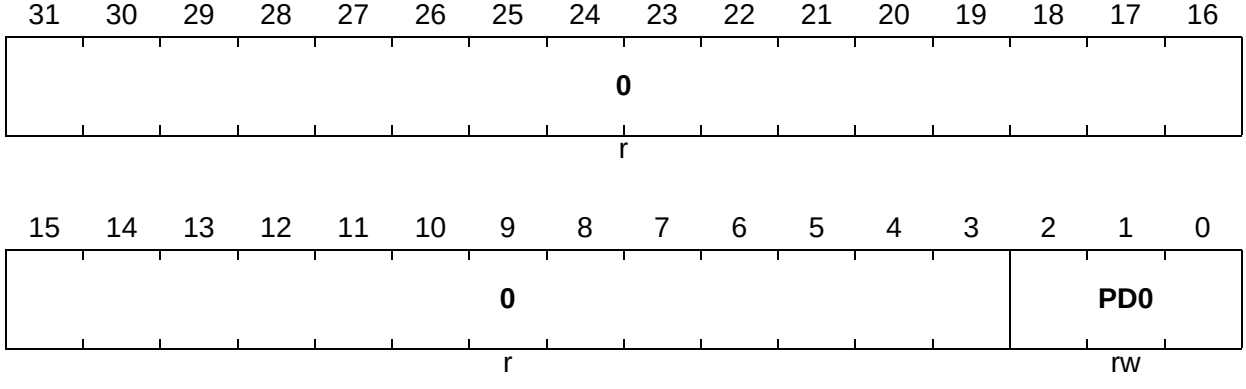
9.11.3.3 Port 7 Input Register

The basic P7_IN register functionality is described on Page 9-19. Port lines P7.[15:8] are not available. Therefore, the P7_IN bits P[15:8] are always read as 0.

9.11.3.4 Port 7 Pad Driver Mode Register and Pad Classes

The Port 7 pad driver mode register contains one bit field that determines the pad driver mode (output driver strength and slew rate) of the Port 7 lines. The port lines are all class A1 pads.

P7_PDR
Port 7 Pad Driver Mode Register (40_H) Reset Value: 0000 0000_H



Field	Bits	Type	Description
PD0	[2:0]	rw	**Pad Driver Mode for P7.[7:0]** (Class A1 pads; for coding see **Page 9-13**)
0	[31:3]	r	**Reserved** Read as 0; should be written with 0.

9.12 Port 8

This section describes the Port 8 functionality in detail.

9.12.1 Port 8 Configuration

Port 8 is an 8-bit bi-directional general-purpose I/O port which can be used for the MLI1 interface lines or for the GPTA0/GPTA1 I/O lines.

9.12.2 Port 8 Function Table

Table 9-26 summarizes the I/O control selection functions of each Port 8 line.

Table 9-26 Port 8 Functions

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P8.0	I	General-purpose input	P8_IN.P0	P8_IOCR0.PC0	$0XXX_B$
		GPTA0/GPTA1 input	IN40		
	O	General-purpose output	P8_OUT.P0		$1X00_B$
		GPTA0 output	OUT40		$1X01_B$
		GPTA1 output	OUT40		$1X10_B$
		MLI1 output	TCLK1		$1X11_B$
P8.1	I	General-purpose input	P8_IN.P1	P8_IOCR0.PC1	$0XXX_B$
		GPTA0/GPTA1 input	IN41		
		MLI1 input	TREADY1A		
	O	General-purpose output	P8_OUT.P1		$1X00_B$
		GPTA0 output	OUT41		$1X01_B$
		GPTA1 output	OUT41		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P8.2	I	General-purpose input	P8_IN.P2	P8_IOCR0.PC2	$0XXX_B$
		GPTA0/GPTA1 input	IN42		
	O	General-purpose output	P8_OUT.P2		$1X00_B$
		GPTA0 output	OUT42		$1X01_B$
		GPTA1 output	OUT42		$1X10_B$
		MLI1 output	TVALID1A		$1X11_B$
P8.3	I	General-purpose input	P8_IN.P3	P8_IOCR0.PC3	$0XXX_B$
		GPTA0/GPTA1 input	IN43		
	O	General-purpose output	P8_OUT.P3		$1X00_B$
		GPTA0 output	OUT43		$1X01_B$
		GPTA1 output	OUT43		$1X10_B$
		MLI1 output	TDATA1		$1X11_B$

Table 9-26 Port 8 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P8.4	I	General-purpose input	P8_IN.P4	P8_IOCR4.PC4	$0XXX_B$
		GPTA0/GPTA1 input	IN44		
		MLI1 input	RCLK1A		
	O	General-purpose output	P8_OUT.P4		$1X00_B$
		GPTA0 output	OUT44		$1X01_B$
		GPTA1 output	OUT44		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P8.5	I	General-purpose input	P8_IN.P5	P8_IOCR4.PC5	$0XXX_B$
		GPTA0/GPTA1 input	IN45		
	O	General-purpose output	P8_OUT.P5		$1X00_B$
		GPTA0 output	OUT45		$1X01_B$
		GPTA1 output	OUT45		$1X10_B$
		MLI1 output	RREADY1A		$1X11_B$
P8.6	I	General-purpose input	P8_IN.P6	P8_IOCR4.PC6	$0XXX_B$
		GPTA0/GPTA1 input	IN46		
		MLI1 input	RVALID1A		
	O	General-purpose output	P8_OUT.P6		$1X00_B$
		GPTA0 output	OUT46		$1X01_B$
		GPTA1 output	OUT46		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P8.7	I	General-purpose input	P8_IN.P7	P8_IOCR4.PC7	$0XXX_B$
		GPTA0/GPTA1 input	IN47		
		MLI1 input	RDATA1A		
	O	General-purpose output	P8_OUT.P7		$1X00_B$
		GPTA0 output	OUT47		$1X01_B$
		GPTA1 output	OUT47		$1X10_B$
		Reserved[1)]	–		$1X11_B$

1) The port I/O control values P8_IOCRx.Py that are assigned to this reserved alternate output control selection should not be used. Otherwise, unpredictable output port line behavior may occur.

9.12.3 Port 8 Register

The following registers are available on Port 8:

Table 9-27 Port 8 Registers

Register Short Name	Register Long Name	Offset Address	Description see
P8_OUT	Port 8 Output Register	0000_H	below[1]
P8_OMR	Port 8 Output Modification Register	0004_H	
P8_IOCR0	Port 8 Input/Output Control Register 0	0010_H	Page 9-9
P8_IOCR4	Port 8 Input/Output Control Register 4	0014_H	Page 9-10
P8_IN	Port 8 Input Register	0024_H	below[1]
P8_PDR	Port 8 Pad Driver Mode Register	0040_H	Page 9-78[1]
P8_ESR	Port 8 Emergency Stop Register	0050_H	below[1]

1) These registers are listed here in the Port 8 section because they differ from the general port register description given in Section 9.3.

9.12.3.1 Port 8 Output Register

The basic P8_OUT register functionality is described on Page 9-15. Port lines P8.[15:8] are not available. Therefore, the P8_OUT bits P[15:8] should be written with 0 and are always read as 0.

9.12.3.2 Port 8 Output Modification Register

The basic P8_OMR register functionality is described on Page 9-16. Port lines P8.[15:8] are not available. Therefore, the P8_OMR bits PS[15:8] and PR[15:8] are not implemented. These bits should always be written with 0.

9.12.3.3 Port 8 Input Register

The basic P8_IN register functionality is described on Page 9-19. Port lines P8.[15:8] are not available. Therefore, the P8_IN bits P[15:8] are always read as 0.

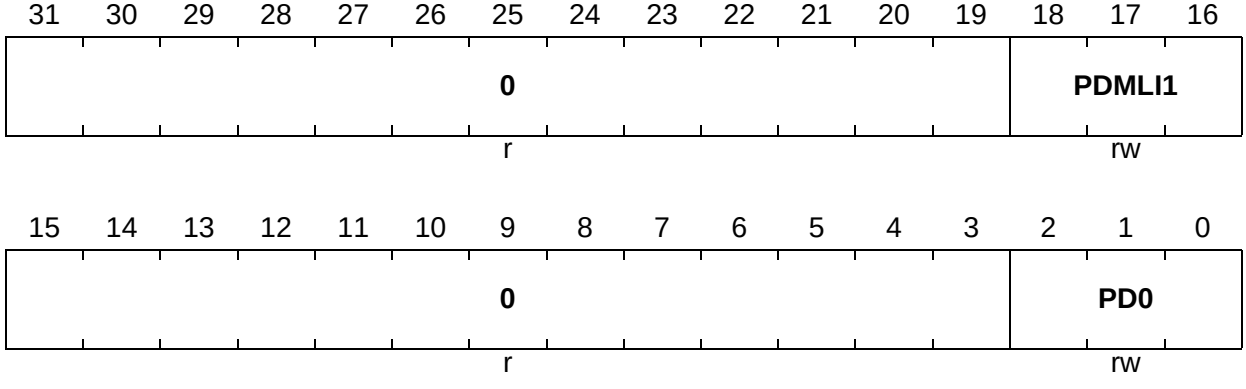
9.12.3.4 Port 8 Emergency Stop Register

The basic P8_ESR register functionality is described on Page 9-18. At Port 8, only port lines P8.[7:0] are implemented. Therefore, the P8_ESR bits EN[15:8] are not implemented. They are always read as 0 and should be written with 0.

9.12.3.5 Port 8 Pad Driver Mode Register and Pad Classes

The Port 8 pad driver mode register contains two bit fields that determine the pad driver mode (output driver strength and slew rate) of Port 8 lines and line groups. The port lines are assigned to A1 and A2 pad classes.

P8_PDR
Port 8 Pad Driver Mode Register (40_H) Reset Value: 0000 0000_H



Field	Bits	Type	Description
PD0	[2:0]	rw	**Pad Driver Mode for P8.1, P8.4, P8.6, and P8.7** (Class A1 pads; for coding see **Page 9-13**)
PDMLI1	[18:16]	rw	**Pad Driver Mode for P8.0, P8.2, P8.3, and P8.5** (Class A2 pads; for coding see **Page 9-13**)
0	[15:3], [31:19]	r	**Reserved** Read as 0; should be written with 0.

9.13 Port 9

This section describes the Port 9 functionality in detail.

9.13.1 Port 9 Configuration

Port 9 is a 15-bit bi-directional general-purpose I/O port which can be used for the MSC0/MSC1 interface output lines or for the GPTA0/GPTA1 I/O lines.

9.13.2 Port 9 Function Table

Table 9-28 summarizes the I/O control selection functions of each Port 9 line.

Table 9-28 Port 9 Functions

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P9.0	I	General-purpose input	P9_IN.P0	P9_IOCR0.PC0	$0XXX_B$
		GPTA0/GPTA1 input	IN48		
	O	General-purpose output	P9_OUT.P0		$1X00_B$
		GPTA0 output	OUT48		$1X01_B$
		GPTA1 output	OUT48		$1X10_B$
		MSC1 output	EN12		$1X11_B$
P9.1	I	General-purpose input	P9_IN.P1	P9_IOCR0.PC1	$0XXX_B$
		GPTA0/GPTA1 input	IN49		
	O	General-purpose output	P9_OUT.P1		$1X00_B$
		GPTA0 output	OUT49		$1X01_B$
		GPTA1 output	OUT49		$1X10_B$
		MSC1 output	EN11		$1X11_B$
P9.2	I	General-purpose input	P9_IN.P2	P9_IOCR0.PC2	$0XXX_B$
		GPTA0/GPTA1 input	IN50		
	O	General-purpose output	P9_OUT.P2		$1X00_B$
		GPTA0 output	OUT50		$1X01_B$
		GPTA1 output	OUT50		$1X10_B$
		MSC1 output	SOP1B		$1X11_B$
P9.3	I	General-purpose input	P9_IN.P3	P9_IOCR0.PC3	$0XXX_B$
		GPTA0/GPTA1 input	IN51		
	O	General-purpose output	P9_OUT.P3		$1X00_B$
		GPTA0 output	OUT51		$1X01_B$
		GPTA1 output	OUT51		$1X10_B$
		MSC1 output	FCLP1B		$1X11_B$

Table 9-28 Port 9 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P9.4	I	General-purpose input	P9_IN.P4	P9_IOCR4.PC4	$0XXX_B$
		GPTA0/GPTA1 input	IN52		
	O	General-purpose output	P9_OUT.P4		$1X00_B$
		GPTA0 output	OUT52		$1X01_B$
		GPTA1 output	OUT52		$1X10_B$
		MSC0 output	EN03		$1X11_B$
P9.5	I	General-purpose input	P9_IN.P5	P9_IOCR4.PC5	$0XXX_B$
		GPTA0/GPTA1 input	IN53		
	O	General-purpose output	P9_OUT.P5		$1X00_B$
		GPTA0 output	OUT53		$1X01_B$
		GPTA1 output	OUT53		$1X10_B$
		MSC0 output	EN02		$1X11_B$
P9.6	I	General-purpose input	P9_IN.P6	P9_IOCR4.PC6	$0XXX_B$
		GPTA0/GPTA1 input	IN54		
	O	General-purpose output	P9_OUT.P6		$1X00_B$
		GPTA0 output	OUT54		$1X01_B$
		GPTA1 output	OUT54		$1X10_B$
		MSC0 output	EN01		$1X11_B$
P9.7	I	General-purpose input	P9_IN.P7	P9_IOCR4.PC7	$0XXX_B$
		GPTA0/GPTA1 input	IN55		
	O	General-purpose output	P9_OUT.P7		$1X00_B$
		GPTA0 output	OUT55		$1X01_B$
		GPTA1 output	OUT55		$1X10_B$
		MSC0 output	SOP0B		$1X11_B$

Table 9-28 Port 9 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P9.8	I	General-purpose input	P9_IN.P8	P9_IOCR8.PC8	$0XXX_B$
	O	General-purpose output	P9_OUT.P8		$1X00_B$
		MSC0 output	FCLP0B		$1X01_B$
		MSC0 output	FCLP0B		$1X10_B$
		MSC0 output	FCLP0B		$1X11_B$
P9.9	I	General-purpose input	P9_IN.P9	P9_IOCR8.PC9	$0XXX_B$
	O	General-purpose output	P9_OUT.P9		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P9.10	I	General-purpose input	P9_IN.P10	P9_IOCR8.PC10	$0XXX_B$
		SCU	EMGSTOP		
	O	General-purpose output	P9_OUT.P10		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P9.11	I	General-purpose input	P9_IN.P11	P9_IOCR8.PC11	$0XXX_B$
	O	General-purpose output	P9_OUT.P11		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P9.12	I	General-purpose input	P9_IN.P12	P9_IOCR12.PC12	$0XXX_B$
	O	General-purpose output	P9_OUT.P12		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$

Table 9-28 Port 9 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P9.13	I	General-purpose input	P9_IN.P13	P9_IOCR12. PC13	$0XXX_B$
		OCDS	$\overline{\text{BRKIN}}$		
	O	General-purpose output	P9_OUT.P13		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_OUT	OCDS;SEN	$\overline{\text{BRKOUT}}$		SDIR
P9.14	I	General-purpose input	P9_IN.P14	P9_IOCR12. PC14	$0XXX_B$
		OCDS	$\overline{\text{BRKIN}}$		
	O	General-purpose output	P9_OUT.P14		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_OUT	OCDS;SEN	$\overline{\text{BRKOUT}}$		SDIR

9.13.3 Port 9 Registers

The following registers are available on Port 9:

Table 9-29 Port 9 Registers

Register Short Name	Register Long Name	Offset Address	Description see
P9_OUT	Port 9 Output Register	0000_H	below[1]
P9_OMR	Port 9 Output Modification Register	0004_H	
P9_IOCR0	Port 9 Input/Output Control Register 0	0010_H	Page 9-9
P9_IOCR4	Port 9 Input/Output Control Register 4	0014_H	Page 9-10
P9_IOCR8	Port 9 Input/Output Control Register 8	0018_H	Page 9-10
P9_IOCR12	Port 9 Input/Output Control Register 12	$001C_H$	Page 9-85[1]
P9_IN	Port 9 Input Register	0024_H	Page 9-85[1]
P9_PDR	Port 9 Pad Driver Mode Register	0040_H	Page 9-86[1]
P9_ESR	Port 9 Emergency Stop Register	0050_H	Page 9-86[1]

1) These registers are listed and noted here in the Port 9 section because they differ from the general port register description given in Section 9.3.

9.13.3.1 Port 9 Output Register

The basic P9_OUT register functionality is described on Page 9-15. Port line P9.15 is not available. Therefore, the P9_OUT bit 15 should be written with 0 and is always read as 0.

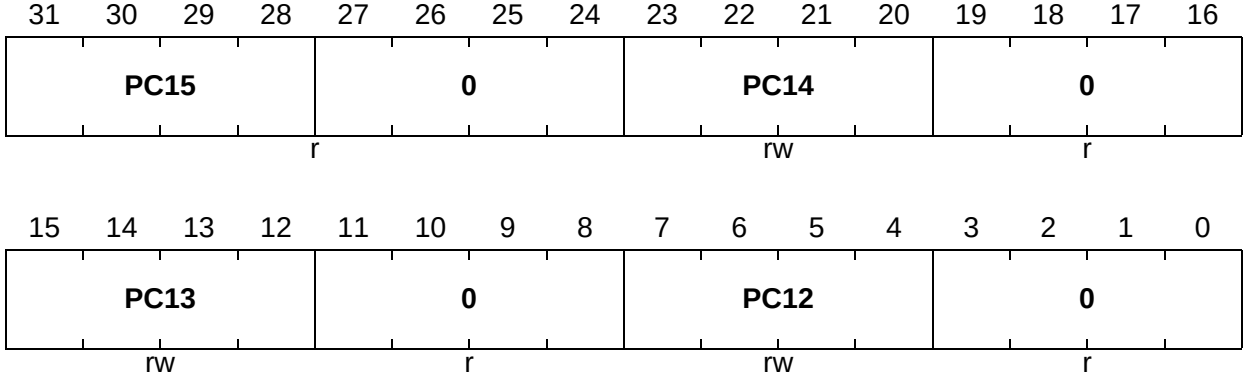
9.13.3.2 Port 9 Output Modification Register

The basic P9_OMR register functionality is described on Page 9-16. Port line P9.15 is not available. Therefore, the P9_OMR bits PS15 and PR15 are not implemented. These bits should always be written with 0.

9.13.3.3 Port 9 Input/Output Control Register 8

P9_IOCR12
Port 9 Input/Output Control Register 12

($1C_H$) **Reset Value: 2020 2020_H**



Field	Bits	Type	Description
PC12	[7:4]	rw	**Port Control for P9.12** This bit field determines the P9.12 functionality.
PC13	[15:12]	rw	**Port Control for P9.13** This bit field determines the P9.13 functionality.
PC14	[23:20]	rw	**Port Control for P9.14** This bit field determines the P9.14 functionality.
PC15	[31:28]	rw	**Reserved;** read as 0010_B after reset; returns value that was written.
0	[3:0], [11:8], [19:16], [27:24]	r	**Reserved** Read as 0; should be written with 0.

9.13.3.4 Port 9 Input Register

The basic P9_IN register functionality is described on **Page 9-19**. Port line P9.15 is not available. Therefore, the P9_IN bit P9.15 is always read as 0.

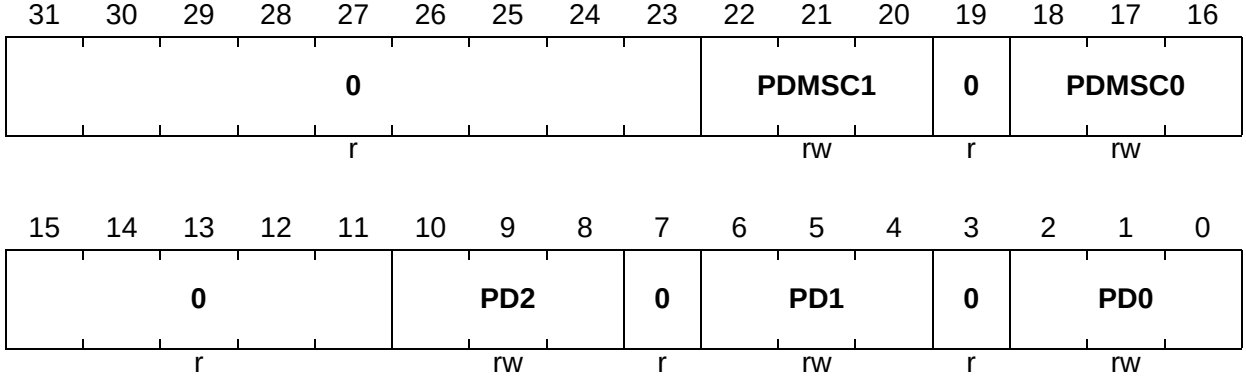
9.13.3.5 Port 9 Emergency Stop Register

The basic P9_ESR register functionality is described on **Page 9-18**. At Port 9, only port lines P9.[7:0] have GPTA outputs. Therefore, the P9_ESR bits EN[15:8] are not implemented. They are always read as 0 and should be written with 0.

9.13.3.6 Port 9 Pad Driver Mode Register and Pad Classes

The Port 9 pad driver mode register contains three bit fields that determine the pad driver mode (output driver strength and slew rate) of Port 9 lines. The port lines are class A1 and class A2 pads.

P9_PDR
Port 9 Pad Driver Mode Register (40_H) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
PD0	[2:0]	rw	**Pad Driver Mode for MSC1/GPIO P9.8** (Class A2 pads; for coding see **Page 9-13**)
PD1	[6:4]	rw	**Pad Driver Mode for GPIO P9.[12:9]** (Class A1 pads; for coding see **Page 9-13**)
PD2	[10:8]	rw	**Pad Driver Mode for GPIO P9.[14:13]** (Class A2 pads; for coding see **Page 9-13**)
PDMSC0	[18:16]	rw	**Pad Driver Mode for GPTA/MSC0 Outputs P9.[7:4]** (Class A2 pads; for coding see **Page 9-13**)
PDMSC1	[22:20]	rw	**Pad Driver Mode for GPTA/MSC1 Outputs P9.[3:0]** (Class A2 pads; for coding see **Page 9-13**)
0	3, 7, [15:11], 19, [31:23]	r	**Reserved** Read as 0; should be written with 0.

9.14 Port 10

This section describes the Port 10 functionality in detail.

9.14.1 Port 10 Configuration

Port 10 is a 6-bit port.

9.14.2 Port 10 Function Table

Table 9-26 summarizes the I/O control selection functions of each line.

Table 9-30 Port 10 Functions

Port Pin	I/O	Pin Functionality	Associated Reg./I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P10.0	I	General-purpose input	P10_IN.P0	P10_IOCR0 .PC0	$0XXX_B$
		SSC0 Input, master mode	MRST0		
	O	General-purpose output	P10_OUT.P0		$1X00_B$
		SSC0 Output, slave mode	MRST0		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P10.1	I	General-purpose input	P10_IN.P1	P10_IOCR0 .PC1	$0XXX_B$
		SSC0 Input, slave mode	MTSR0		
	O	General-purpose output	–		$1X00_B$
		SSC0 Output, master mode	MTSR0		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P10.2	I	General-purpose input	P10_IN.P2	P10_IOCR0 .PC2	$0XXX_B$
		SSC0 Input	$\overline{\text{SLSI0}}$		
	O	General-purpose output	P10_OUT.P2		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$

Table 9-30 Port 10 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P10.3	I	General-purpose input	P10_IN.P3	P10_IOCR0 .PC3	$0XXX_B$
		SSC0 Input	SCLK0		
	O	General-purpose output	P10_OUT.P3		$1X00_B$
		SSC0 Output	SCLK0		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P10.4	I	General-purpose input	P10_IN.P4	P10_IOCR4 .PC4	$0XXX_B$
	O	General-purpose output	P10_OUT.P4		$1X00_B$
		SSC0 Output	SLSO0		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P10.5	I	General-purpose input	P10_IN.P5	P10_IOCR4 .PC5	$0XXX_B$
	O	General-purpose output	P10_OUT.P5		$1X00_B$
		SSC0 Output	SLSO1		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$

1) The port I/O control values P10_IOCRx.Py that are assigned to this reserved alternate output control selection should not be used. Otherwise, unpredictable output port line behavior may occur.

9.14.3 Port 10 Registers

The following registers are available on Port 10:

Table 9-31 Port 10 Registers

Register Short Name	Register Long Name	Offset Address[1)]	Description see
P10_OUT	Port 10 Output Register	0000_H	
P10_OMR	Port 10 Output Modification Register	0004_H	
P10_IOCR0	Port 10 Input/Output Control Register 0	0010_H	Page 9-9
P10_IOCR4	Port 10 Input/Output Control Register 4	0014_H	Page 9-10
P10_IN	Port 10 Input Register	0024_H	Page 9-89[2)]
P10_PDR	Port 10 Pad Driver Mode Register	0040_H	Page 9-90

1) The absolute addresses are calculated by adding the offset address to the module base address (see Table 9-5)

2) This register is listed here in the Port 10 section because it differs from the general port register description given in Section 9.3.

Note: Register P10_IN makes it possible to read the actual logic levels of the Port 10 inputs.

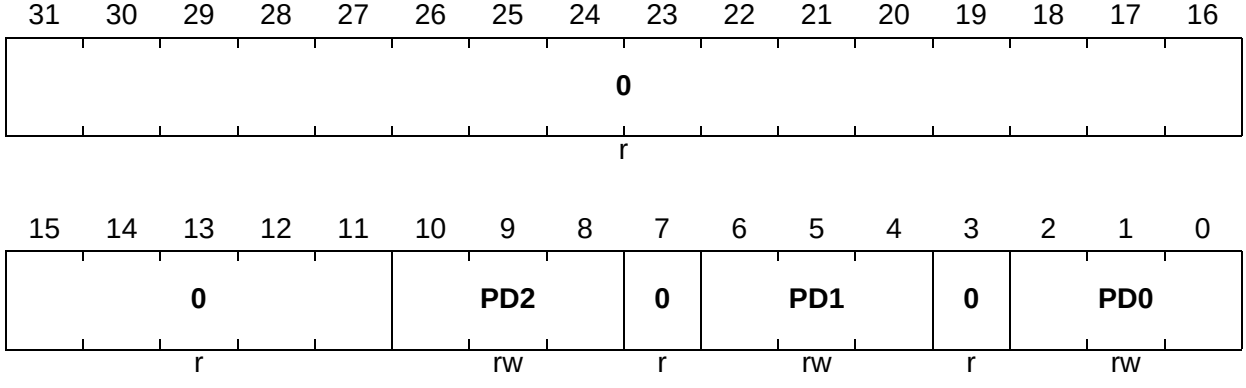
9.14.3.1 Port 10 Input Register

The basic P10_IN register functionality is described on Page 9-19. Port lines P10.[15:6] are not available. Therefore, the P10_IN bits P[15:6] are always read as 0.

9.14.3.2 Port 10 Pad Driver Mode Register and Pad Classes

The Port 10 pad driver mode register contains three bit fields that determine the pad driver mode (output driver strength and slew rate) of the Port 10 lines.

P10_PDR
Port 10 Pad Driver Mode Register (40_H) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
PD0	[2:0]	rw	**Pad Driver Mode for P10.0, P10.1, and P10.3** (Class A2 pads; for coding see **Page 9-13**)
PD1	[6:4]	rw	**Pad Driver Mode for P10.2** (Class A1 pads; for coding see **Page 9-13**)
PD2	[10:8]	rw	**Pad Driver Mode for P10.[5:4]** (Class A2 pads; for coding see **Page 9-13**)
0	3, 7, [31:11]	r	**Reserved** Read as 0; should be written with 0.

9.15 Port 11

This section describes the Port 11 functionality in detail.

9.15.1 Port 11 Configuration

Port 11 is a general-purpose 16-bit bi-directional port. It serves as GPIO lines without secondary functions.

Table 9-10 summarizes the I/O control selection functions of each Port 11 line.

9.15.2 Port 11 Function Table

Table 9-10 summarizes the I/O control selection functions of each Port 11 line.

Table 9-32 Port 11 Functions

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P11.0	I	General-purpose input	P11_IN.P0	P11_IOCR0.PC0	$0XXX_B$
	O	General-purpose output	P11_OUT.P0		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	A0		DIR
P11.1	I	General-purpose input	P11_IN.P1	P11_IOCR0.PC1	$0XXX_B$
	O	General-purpose output	P11_OUT.P1		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	A1		DIR
P11.2	I	General-purpose input	P11_IN.P2	P11_IOCR0.PC2	$0XXX_B$
	O	General-purpose output	P11_OUT.P2		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	A2		DIR
P11.3	I	General-purpose input	P11_IN.P3	P11_IOCR0.PC3	$0XXX_B$
	O	General-purpose output	P11_OUT.P3		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	A3		DIR

Table 9-32 Port 11 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P11.4	I	General-purpose input	P11_IN.P4	P11_IOCR4. PC4	$0XXX_B$
	O	General-purpose output	P11_OUT.P4		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	A4		DIR
P11.5	I	General-purpose input	P11_IN.P5	P11_IOCR4. PC5	$0XXX_B$
	O	General-purpose output	P11_OUT.P5		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	A5		DIR
P11.6	I	General-purpose input	P11_IN.P6	P11_IOCR4. PC6	$0XXX_B$
	O	General-purpose output	P11_OUT.P6		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	A6		DIR
P11.7	I	General-purpose input	P11_IN.P7	P11_IOCR4. PC7	$0XXX_B$
	O	General-purpose output	P11_OUT.P7		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	A7		DIR

Table 9-32 Port 11 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P11.8	I	General-purpose input	P11_IN.P8	P11_IOCR8.PC8	$0XXX_B$
	O	General-purpose output	P11_OUT.P8		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	A8		DIR
P11.9	I	General-purpose input	P11_IN.P9	P11_IOCR8.PC9	$0XXX_B$
	O	General-purpose output	P11_OUT.P9		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	A9		DIR
P11.10	I	General-purpose input	P11_IN.P10	P11_IOCR8.PC10	$0XXX_B$
	O	General-purpose output	P11_OUT.P10		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	A10		DIR
P11.11	I	General-purpose input	P11_IN.P11	P11_IOCR8.PC11	$0XXX_B$
	O	General-purpose output	P11_OUT.P11		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	A11		DIR

Table 9-32 Port 11 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P11.12	I	General-purpose input	P11_IN.P12	P11_IOCR12. PC12	$0XXX_B$
	O	General-purpose output	P11_OUT.P12		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	A12		DIR
P11.13	I	General-purpose input	P11_IN.P13	P11_IOCR12. PC13	$0XXX_B$
	O	General-purpose output	P11_OUT.P13		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	A13		DIR
P11.14	I	General-purpose input	P11_IN.P14	P11_IOCR12. PC14	$0XXX_B$
	O	General-purpose output	P11_OUT.P14		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	A14		DIR
P11.15	I	General-purpose input	P11_IN.P15	P11_IOCR12. PC15	$0XXX_B$
	O	General-purpose output	P11_OUT.P15		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	A15		DIR

9.15.3 Port 11 Registers

The following registers are available on Port 11:

Table 9-33 Port 11 Registers

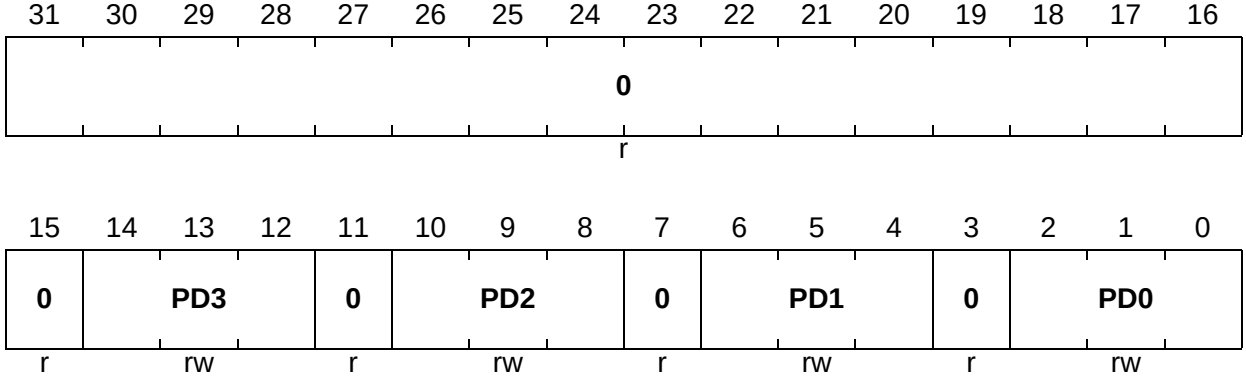
Register Short Name	Register Long Name	Offset Address	Description see
P11_OUT	Port 11 Output Register	0000_H	Page 9-15
P11_OMR	Port 11 Output Modification Register	0004_H	Page 9-16
P11_IOCR0	Port 11 Input/Output Control Register 0	0010_H	Page 9-9
P11_IOCR4	Port 11 Input/Output Control Register 4	0014_H	Page 9-10
P11_IOCR8	Port 11 Input/Output Control Register 8	0018_H	Page 9-10
P11_IOCR12	Port 11 Input/Output Control Register 12	$001C_H$	Page 9-11
P11_IN	Port 11 Input Register	0024_H	Page 9-19
P11_PDR	Port 11 Pad Driver Mode Register	0040_H	Page 9-97[1)]

1) This register is listed here in the Port 0 section because they differ from the general port register description given in **Section 9.3**.

9.15.3.1 Port 11 Pad Driver Mode Register and Pad Classes

The Port 11 pad driver mode register contains two bit fields that determine the pad driver mode (output driver strength and slew rate) of Port 11 line groups. The port lines are all class B1 pads.

P11_PDR
Port 11 Pad Driver Mode Register (40_H) Reset Value: 0000 0000_H



Field	Bits	Type	Description
PD0	[2:0]	rw	**Pad Driver Mode for P11.[3:0]** (Class B1 pads; for coding see **Page 9-13**)
PD1	[6:4]	rw	**Pad Driver Mode for P11.[7:4]** (Class B1 pads; for coding see **Page 9-13**)
PD2	[10:8]	rw	**Pad Driver Mode for P11.[11:8]** (Class B1 pads; for coding see **Page 9-13**)
PD3	[14:12]	rw	**Pad Driver Mode for P11.[15:12]** (Class B1 pads; for coding see **Page 9-13**)
0	3, 7, 11, [31:15]	r	**Reserved** Read as 0; should be written with 0.

9.16 Port 12

This section describes the Port 12 functionality in detail.

9.16.1 Port 12 Configuration

Port 12 is an 8-bit bi-directional general-purpose I/O port.

9.16.2 Port 12 Function Table

Table 9-26 summarizes the I/O control selection functions of each Port 12 line.

Table 9-34 Port 12 Functions

Port Pin	I/O	Pin Functionality	Associated Reg./I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P12.0	I	General-purpose input	P12_IN.P0	P12_IOCR 0.PC0	$0XXX_B$
	O	General-purpose output	P12_OUT.P0		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	A16		DIR
P12.1	I	General-purpose input	P12_IN.P1	P12_IOCR 0.PC1	$0XXX_B$
	O	General-purpose output	P12_OUT.P1		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved[1)]	–		$1X11_B$
	HW_Out	EBU;EN	A17		DIR
P12.2	I	General-purpose input	P12_IN.P2	P12_IOCR 0.PC2	$0XXX_B$
	O	General-purpose output	P12_OUT.P2		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	A18		DIR
P12.3	I	General-purpose input	P12_IN.P3	P12_IOCR 0.PC3	$0XXX_B$
	O	General-purpose output	P12_OUT.P3		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	A19		DIR

Table 9-34 Port 12 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P12.4	I	General-purpose input	P12_IN.P4	P12_IOCR 4.PC4	$0XXX_B$
	O	General-purpose output	P12_OUT.P4		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	A20		DIR
P12.5	I	General-purpose input	P12_IN.P5	P12_IOCR 4.PC5	$0XXX_B$
	O	General-purpose output	P12_OUT.P5		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	A21		DIR
P12.6	I	General-purpose input	P12_IN.P6	P12_IOCR 4.PC6	$0XXX_B$
	O	General-purpose output	P12_OUT.P6		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	A22		DIR
P12.7	I	General-purpose input	P12_IN.P7	P12_IOCR 4.PC7	$0XXX_B$
	O	General-purpose output	P12_OUT.P7		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	A23		DIR

1) The port I/O control values P12_IOCRx.Py that are assigned to this reserved alternate output control selection should not be used. Otherwise, unpredictable output port line behavior may occur.

9.16.3 Port 12 Registers

The following registers are available on Port 12:

Table 9-35 Port 8 Registers

Register Short Name	Register Long Name	Offset Address	Description see
P12_OUT	Port 12 Output Register	0000_H	below[1]
P12_OMR	Port 12 Output Modification Register	0004_H	
P12_IOCR0	Port 12 Input/Output Control Register 0	0010_H	Page 9-9
P12_IOCR4	Port 12 Input/Output Control Register 4	0014_H	Page 9-10
P12_IN	Port 12 Input Register	0024_H	below[1]
P12_PDR	Port 12 Pad Driver Mode Register	0040_H	Page 9-102[1]

1) These registers are listed here in the Port 8 section because they differ from the general port register description given in **Section 9.3**.

9.16.3.1 Port 12 Output Register

The basic P12_OUT register functionality is described on **Page 9-15**. Port lines P8.[15:8] are not available. Therefore, the P12_OUT bits P[15:8] should be written with 0 and are always read as 0.

9.16.3.2 Port 12 Output Modification Register

The basic P12_OMR register functionality is described on **Page 9-16**. Port lines P8.[15:8] are not available. Therefore, the P12_OMR bits PS[15:8] and PR[15:8] are not implemented. These bits should always be written with 0.

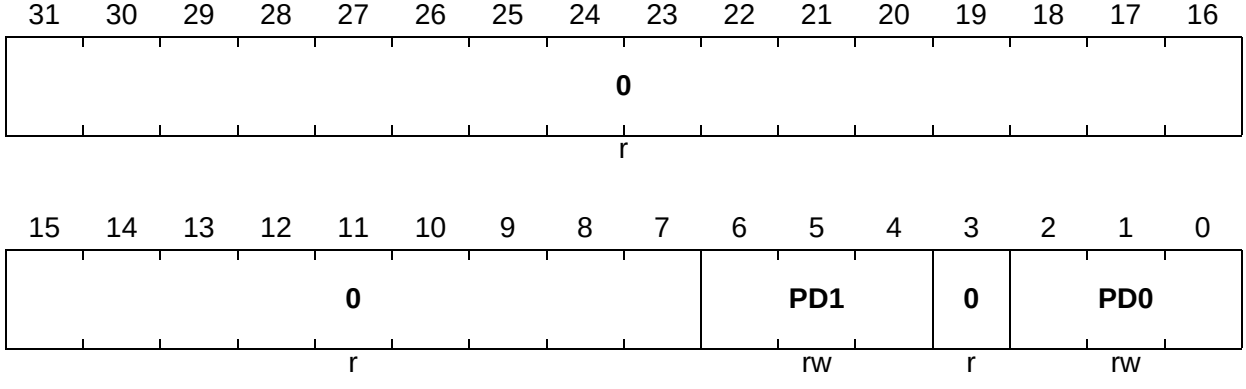
9.16.3.3 Port 12 Input Register

The basic P12_IN register functionality is described on **Page 9-19**. Port lines P12.[15:8] are not available. Therefore, the P12_IN bits P[15:8] are always read as 0.

9.16.3.4 Port 12 Pad Driver Mode Register and Pad Classes

The Port 12 pad driver mode register contains two bit fields that determine the pad driver mode (output driver strength and slew rate) of Port 12 lines and line groups. The port lines are assigned to B1 pad classes.

P12_PDR
Port 12 Pad Driver Mode Register (40_H) Reset Value: 0000 0000_H



Field	Bits	Type	Description
PD0	[2:0]	rw	**Pad Driver Mode for P12.[3:0]** (Class B1 pads; for coding see **Page 9-13**)
PD1	[6:4]	rw	**Pad Driver Mode for P12.[7:4]** (Class B1 pads; for coding see **Page 9-13**)
0	3, [31:7]	r	**Reserved** Read as 0; should be written with 0.

9.17 Port 13

This section describes the Port 13 functionality in detail.

9.17.1 Port 13 Configuration

Port 13 is a general-purpose 16-bit bi-directional port. It serves as GPIO lines without secondary functions.

Table 9-10 summarizes the I/O control selection functions of each Port 0 line.

9.17.2 Port 13 Function Table

Table 9-10 summarizes the I/O control selection functions of each Port 13 line.

Table 9-36 Port 13 Functions

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P13.0	I	General-purpose input	P13_IN.P0	P13_IOCR0.PC0	$0XXX_B$
		EBU Input	D0		
	O	General-purpose output	P13_OUT.P0		$1X00_B$
		GPTA0 output	OUT88		$1X01_B$
		GPTA1 output	OUT88		$1X10_B$
		LTCA2 output	OUT80		$1X11_B$
	HW_Out	EBU;EN	D0		DIR
P13.1	I	General-purpose input	P13_IN.P1	P13_IOCR0.PC1	$0XXX_B$
		EBU Input	D1		
	O	General-purpose output	P13_OUT.P1		$1X00_B$
		GPTA0 output	OUT89		$1X01_B$
		GPTA1 output	OUT89		$1X10_B$
		LTCA2 output	OUT81		$1X11_B$
	HW_Out	EBU;EN	D1		DIR
P13.2	I	General-purpose input	P13_IN.P2	P13_IOCR0.PC2	$0XXX_B$
		EBU Input	D2		
	O	General-purpose output	P13_OUT.P2		$1X00_B$
		GPTA0 output	OUT90		$1X01_B$
		GPTA1 output	OUT90		$1X10_B$
		LTCA2 output	OUT82		$1X11_B$
	HW_Out	EBU;EN	D2		DIR

Table 9-36 Port 13 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P13.3	I	General-purpose input	P13_IN.P3	P13_IOCR0.PC3	$0XXX_B$
		EBU Input	D3		
	O	General-purpose output	P13_OUT.P3		$1X00_B$
		GPTA0 output	OUT91		$1X01_B$
		GPTA1 output	OUT91		$1X10_B$
		LTCA2 output	OUT83		$1X11_B$
	HW_Out	EBU;EN	D3		DIR
P13.4	I	General-purpose input	P13_IN.P4	P13_IOCR4.PC4	$0XXX_B$
		EBU Input	D4		
	O	General-purpose output	P13_OUT.P4		$1X00_B$
		GPTA0 output	OUT92		$1X01_B$
		GPTA1 output	OUT92		$1X10_B$
		LTCA2 output	OUT84		$1X11_B$
	HW_Out	EBU;EN	D4		DIR
P13.5	I	General-purpose input	P13_IN.P5	P13_IOCR4.PC5	$0XXX_B$
		EBU Input	D5		
	O	General-purpose output	P13_OUT.P5		$1X00_B$
		GPTA0 output	OUT93		$1X01_B$
		GPTA1 output	OUT93		$1X10_B$
		LTCA2 output	OUT85		$1X11_B$
	HW_Out	EBU;EN	D5		DIR
P13.6	I	General-purpose input	P13_IN.P6	P13_IOCR4.PC6	$0XXX_B$
		EBU Input	D6		
	O	General-purpose output	P13_OUT.P6		$1X00_B$
		GPTA0 output	OUT94		$1X01_B$
		GPTA1 output	OUT94		$1X10_B$
		LTCA2 output	OUT86		$1X11_B$
	HW_Out	EBU;EN	D6		DIR

Table 9-36 Port 13 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P13.7	I	General-purpose input	P13_IN.P7	P13_IOCR4. PC7	$0XXX_B$
		EBU Input	D7		
	O	General-purpose output	P13_OUT.P7		$1X00_B$
		GPTA0 output	OUT95		$1X01_B$
		GPTA1 output	OUT95		$1X10_B$
		LTCA2 output	OUT87		$1X11_B$
	HW_Out	EBU;EN	D7		DIR
P13.8	I	General-purpose input	P13_IN.P8	P13_IOCR8. PC8	$0XXX_B$
		EBU Input	D8		
	O	General-purpose output	P13_OUT.P8		$1X00_B$
		GPTA0 output	OUT96		$1X01_B$
		GPTA1 output	OUT96		$1X10_B$
		LTCA2 output	OUT88		$1X11_B$
	HW_Out	EBU;EN	D8		DIR
P13.9	I	General-purpose input	P13_IN.P9	P13_IOCR8. PC9	$0XXX_B$
		EBU Input	D9		
	O	General-purpose output	P13_OUT.P9		$1X00_B$
		GPTA0 output	OUT97		$1X01_B$
		GPTA1 output	OUT97		$1X10_B$
		LTCA2 output	OUT89		$1X11_B$
	HW_Out	EBU;EN	D9		DIR
P13.10	I	General-purpose input	P13_IN.P10	P13_IOCR8. PC10	$0XXX_B$
		EBU Input	D10		
	O	General-purpose output	P13_OUT.P10		$1X00_B$
		GPTA0 output	OUT98		$1X01_B$
		GPTA1 output	OUT98		$1X10_B$
		LTCA2 output	OUT90		$1X11_B$
	HW_Out	EBU;EN	D10		DIR

Table 9-36 Port 13 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P13.11	I	General-purpose input	P13_IN.P11	P13_IOCR8.PC11	$0XXX_B$
		EBU Input	D11		
	O	General-purpose output	P13_OUT.P11		$1X00_B$
		GPTA0 output	OUT99		$1X01_B$
		GPTA1 output	OUT99		$1X10_B$
		LTCA2 output	OUT91		$1X11_B$
	HW_Out	EBU;EN	D11		DIR
P13.12	I	General-purpose input	P13_IN.P12	P13_IOCR12.PC12	$0XXX_B$
		EBU Input	D12		
	O	General-purpose output	P13_OUT.P12		$1X00_B$
		GPTA0 output	OUT100		$1X01_B$
		GPTA1 output	OUT100		$1X10_B$
		LTCA2 output	OUT92		$1X11_B$
	HW_Out	EBU;EN	D12		DIR
P13.13	I	General-purpose input	P13_IN.P13	P13_IOCR12.PC13	$0XXX_B$
		EBU Input	D13		
	O	General-purpose output	P13_OUT.P13		$1X00_B$
		GPTA0 output	OUT101		$1X01_B$
		GPTA1 output	OUT101		$1X10_B$
		LTCA2 output	OUT93		$1X11_B$
	HW_Out	EBU;EN	D13		DIR
P13.14	I	General-purpose input	P13_IN.P14	P13_IOCR12.PC14	$0XXX_B$
		EBU Input	D14		
	O	General-purpose output	P13_OUT.P14		$1X00_B$
		GPTA0 output	OUT102		$1X01_B$
		GPTA1 output	OUT102		$1X10_B$
		LTCA2 output	OUT94		$1X11_B$
	HW_Out	EBU;EN	D14		DIR

Table 9-36 Port 13 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P13.15	I	General-purpose input	P13_IN.P15	P13_IOCR12.PC15	$0XXX_B$
		EBU Input	D15		
	O	General-purpose output	P13_OUT.P15		$1X00_B$
		GPTA0 output	OUT103		$1X01_B$
		GPTA1 output	OUT103		$1X10_B$
		LTCA2 output	OUT95		$1X11_B$
	HW_Out	EBU;EN	D15		DIR

9.17.3 Port 13 Registers

The following registers are available on Port 13:

Table 9-37 Port 13 Registers

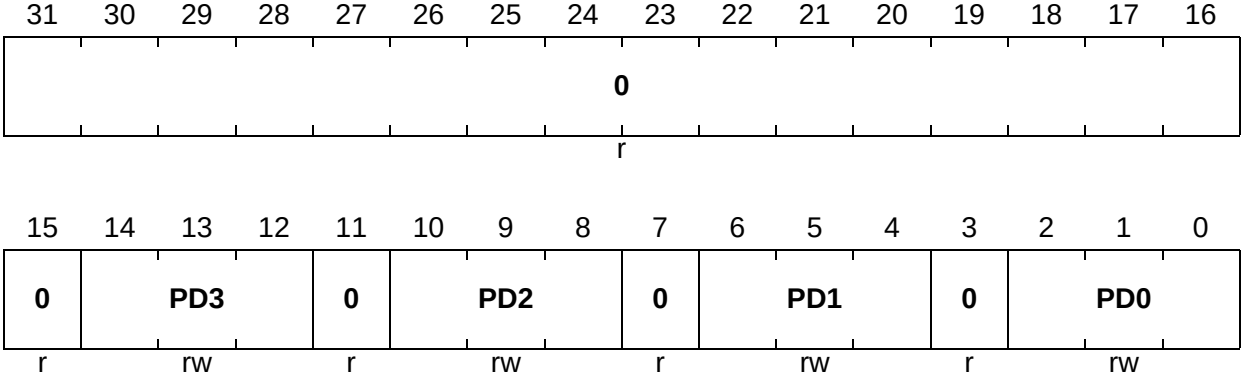
Register Short Name	Register Long Name	Offset Address	Description see
P13_OUT	Port 13 Output Register	0000_H	Page 9-15
P13_OMR	Port 13 Output Modification Register	0004_H	Page 9-16
P13_IOCR0	Port 13 Input/Output Control Register 0	0010_H	Page 9-9
P13_IOCR4	Port 13 Input/Output Control Register 4	0014_H	Page 9-10
P13_IOCR8	Port 13 Input/Output Control Register 8	0018_H	Page 9-10
P13_IOCR12	Port 13 Input/Output Control Register 12	$001C_H$	Page 9-11
P13_IN	Port 13 Input Register	0024_H	Page 9-19
P13_PDR	Port 13 Pad Driver Mode Register	0040_H	Page 9-110[1)]
P13_ESR	Port 13 Emergency Stop Register	0050_H	Page 9-18

1) This register is listed here in the Port 0 section because they differ from the general port register description given in **Section 9.3**.

9.17.3.1 Port 13 Pad Driver Mode Register and Pad Classes

The Port 13 pad driver mode register contains four bit fields that determine the pad driver mode (output driver strength and slew rate) of Port 13 line groups. The Port 13 port lines are all class B1 pads.

P13_PDR
Port 13 Pad Driver Mode Register (40_H) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
PD0	[2:0]	rw	**Pad Driver Mode for P13.[3:0]** (Class B1 pads; for coding see **Page 9-13**)
PD1	[6:4]	rw	**Pad Driver Mode for P13.[7:4]** (Class B1 pads; for coding see **Page 9-13**)
PD2	[10:8]	rw	**Pad Driver Mode for P13.[11:8]** (Class B1 pads; for coding see **Page 9-13**)
PD3	[14:12]	rw	**Pad Driver Mode for P13.[15:12]** (Class B1 pads; for coding see **Page 9-13**)
0	3, 7, 11, [31:15]	r	**Reserved** Read as 0; should be written with 0.

9.17.3.2 Port 13 Emergency Stop Register

The basic P13_ESR register functionality is described on **Page 9-18**. At Port 13, all port lines P13.[15:0] have GPTA outputs and correspondent ESR lines.

9.18 Port 14

This section describes the Port 14 functionality in detail.

9.18.1 Port 14 Configuration

Port 14 is a general-purpose 16-bit bi-directional port. It serves as GPIO lines without secondary functions.

Table 9-10 summarizes the I/O control selection functions of each Port 14 line.

9.18.2 Port 14 Function Table

Table 9-10 summarizes the I/O control selection functions of each Port 14 line.

Table 9-38 Port 14 Functions

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P14.0	I	General-purpose input	P14_IN.P0	P14_IOCR0.PC0	$0XXX_B$
		EBU Input	D16		
	O	General-purpose output	P14_OUT.P0		$1X00_B$
		GPTA0 output	OUT96		$1X01_B$
		GPTA1 output	OUT96		$1X10_B$
		LTCA2 output	OUT96		$1X11_B$
	HW_Out	EBU;EN	D16		DIR
P14.1	I	General-purpose input	P14_IN.P1	P14_IOCR0.PC1	$0XXX_B$
		EBU Input	D17		
	O	General-purpose output	P14_OUT.P1		$1X00_B$
		GPTA0 output	OUT97		$1X01_B$
		GPTA1 output	OUT97		$1X10_B$
		LTCA2 output	OUT97		$1X11_B$
	HW_Out	EBU;EN	D17		DIR
P14.2	I	General-purpose input	P14_IN.P2	P14_IOCR0.PC2	$0XXX_B$
		EBU Input	D18		
	O	General-purpose output	P14_OUT.P2		$1X00_B$
		GPTA0 output	OUT98		$1X01_B$
		GPTA1 output	OUT98		$1X10_B$
		LTCA2 output	OUT98		$1X11_B$
	HW_Out	EBU;EN	D18		DIR

Table 9-38 Port 14 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P14.3	I	General-purpose input	P14_IN.P3	P14_IOCR0.PC3	$0XXX_B$
		EBU Input	D19		
	O	General-purpose output	P14_OUT.P3		$1X00_B$
		GPTA0 output	OUT99		$1X01_B$
		GPTA1 output	OUT99		$1X10_B$
		LTCA2 output	OUT99		$1X11_B$
	HW_Out	EBU;EN	D19		DIR
P14.4	I	General-purpose input	P14_IN.P4	P14_IOCR4.PC4	$0XXX_B$
		EBU Input	D20		
	O	General-purpose output	P14_OUT.P4		$1X00_B$
		GPTA0 output	OUT100		$1X01_B$
		GPTA1 output	OUT100		$1X10_B$
		LTCA2 output	OUT100		$1X11_B$
	HW_Out	EBU;EN	D20		DIR
P14.5	I	General-purpose input	P14_IN.P5	P14_IOCR4.PC5	$0XXX_B$
		EBU Input	D21		
	O	General-purpose output	P14_OUT.P5		$1X00_B$
		GPTA0 output	OUT101		$1X01_B$
		GPTA1 output	OUT101		$1X10_B$
		LTCA2 output	OUT101		$1X11_B$
	HW_Out	EBU;EN	D21		DIR
P14.6	I	General-purpose input	P14_IN.P6	P14_IOCR4.PC6	$0XXX_B$
		EBU Input	D22		
	O	General-purpose output	P14_OUT.P6		$1X00_B$
		GPTA0 output	OUT102		$1X01_B$
		GPTA1 output	OUT102		$1X10_B$
		LTCA2 output	OUT102		$1X11_B$
	HW_Out	EBU;EN	D22		DIR

Table 9-38 Port 14 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P14.7	I	General-purpose input	P14_IN.P7	P14_IOCR4.PC7	$0XXX_B$
		EBU Input	D23		
	O	General-purpose output	P14_OUT.P7		$1X00_B$
		GPTA0 output	OUT103		$1X01_B$
		GPTA1 output	OUT103		$1X10_B$
		LTCA2 output	OUT103		$1X11_B$
	HW_Out	EBU;EN	D23		DIR
P14.8	I	General-purpose input	P14_IN.P8	P14_IOCR8.PC8	$0XXX_B$
		EBU Input	D24		
	O	General-purpose output	P14_OUT.P8		$1X00_B$
		GPTA0 output	OUT104		$1X01_B$
		GPTA1 output	OUT104		$1X10_B$
		LTCA2 output	OUT104		$1X11_B$
	HW_Out	EBU;EN	D24		DIR
P14.9	I	General-purpose input	P14_IN.P9	P14_IOCR8.PC9	$0XXX_B$
		EBU Input	D25		
	O	General-purpose output	P14_OUT.P9		$1X00_B$
		GPTA0 output	OUT105		$1X01_B$
		GPTA1 output	OUT105		$1X10_B$
		LTCA2 output	OUT105		$1X11_B$
	HW_Out	EBU;EN	D25		DIR
P14.10	I	General-purpose input	P14_IN.P10	P14_IOCR8.PC10	$0XXX_B$
		EBU Input	D26		
	O	General-purpose output	P14_OUT.P10		$1X00_B$
		GPTA0 output	OUT106		$1X01_B$
		GPTA1 output	OUT106		$1X10_B$
		LTCA2 output	OUT106		$1X11_B$
	HW_Out	EBU;EN	D26		DIR

Table 9-38 Port 14 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P14.11	I	General-purpose input	P14_IN.P11	P14_IOCR8. PC11	$0XXX_B$
		EBU Input	D27		
	O	General-purpose output	P14_OUT.P11		$1X00_B$
		GPTA0 output	OUT107		$1X01_B$
		GPTA1 output	OUT107		$1X10_B$
		LTCA2 output	OUT107		$1X11_B$
	HW_Out	EBU;EN	D27		DIR
P14.12	I	General-purpose input	P14_IN.P12	P14_IOCR12. PC12	$0XXX_B$
		EBU Input	D28		
	O	General-purpose output	P14_OUT.P12		$1X00_B$
		GPTA0 output	OUT108		$1X01_B$
		GPTA1 output	OUT108		$1X10_B$
		LTCA2 output	OUT108		$1X11_B$
	HW_Out	EBU;EN	D28		DIR
P14.13	I	General-purpose input	P14_IN.P13	P14_IOCR12. PC13	$0XXX_B$
		EBU Input	D29		
	O	General-purpose output	P14_OUT.P13		$1X00_B$
		GPTA0 output	OUT109		$1X01_B$
		GPTA1 output	OUT109		$1X10_B$
		LTCA2 output	OUT109		$1X11_B$
	HW_Out	EBU;EN	D29		DIR
P14.14	I	General-purpose input	P14_IN.P14	P14_IOCR12. PC14	$0XXX_B$
		EBU Input	D30		
	O	General-purpose output	P14_OUT.P14		$1X00_B$
		GPTA0 output	OUT110		$1X01_B$
		GPTA1 output	OUT110		$1X10_B$
		LTCA2 output	OUT110		$1X11_B$
	HW_Out	EBU;EN	D30		DIR

Table 9-38 Port 14 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P14.15	I	General-purpose input	P14_IN.P15	P14_IOCR12. PC15	$0XXX_B$
		EBU Input	D31		
	O	General-purpose output	P14_OUT.P15		$1X00_B$
		GPTA0 output	OUT111		$1X01_B$
		GPTA1 output	OUT111		$1X10_B$
		LTCA2 output	OUT111		$1X11_B$
	HW_Out	EBU;EN	D31		DIR

9.18.3 Port 14 Registers

The following registers are available on Port 14:

Table 9-39 Port 14 Registers

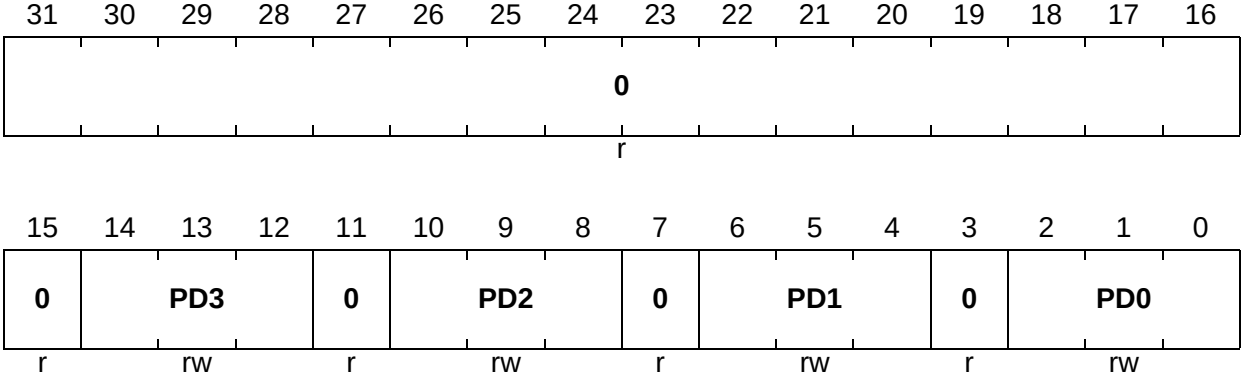
Register Short Name	Register Long Name	Offset Address	Description see
P14_OUT	Port 14 Output Register	0000_H	Page 9-15
P14_OMR	Port 14 Output Modification Register	0004_H	Page 9-16
P14_IOCR0	Port 14 Input/Output Control Register 0	0010_H	Page 9-9
P14_IOCR4	Port 14 Input/Output Control Register 4	0014_H	Page 9-10
P14_IOCR8	Port 14 Input/Output Control Register 8	0018_H	Page 9-10
P14_IOCR12	Port 14 Input/Output Control Register 12	$001C_H$	Page 9-11
P14_IN	Port 14 Input Register	0024_H	Page 9-19
P14_PDR	Port 14 Pad Driver Mode Register	0040_H	Page 9-118[1)]
P14_ESR	Port 14 Emergency Stop Register	0050_H	Page 9-18

1) This register is listed here in the Port 14 section because they differ from the general port register description given in **Section 9.3**.

9.18.3.1 Port 14 Pad Driver Mode Register and Pad Classes

The Port 0 pad driver mode register contains two bit fields that determine the pad driver mode (output driver strength and slew rate) of Port 14 line groups. The port lines are all class B1 pads.

P14_PDR
Port 14 Pad Driver Mode Register (40_H) Reset Value: 0000 0000_H



Field	Bits	Type	Description
PD0	[2:0]	rw	**Pad Driver Mode for P14.[3:0]** (Class B1 pads; for coding see **Page 9-13**)
PD1	[6:4]	rw	**Pad Driver Mode for P14.[7:4]** (Class B1 pads; for coding see **Page 9-13**)
PD2	[10:8]	rw	**Pad Driver Mode for P14.[11:8]** (Class B1 pads; for coding see **Page 9-13**)
PD3	[14:12]	rw	**Pad Driver Mode for P14.[15:12]** (Class B1 pads; for coding see **Page 9-13**)
0	3, 7, 11, [31:15]	r	**Reserved** Read as 0; should be written with 0.

9.18.3.2 Port 14 Emergency Stop Register

The basic P14_ESR register functionality is described on **Page 9-18**. At Port 14, all port lines P14.[15:0] have GPTA outputs and correspondent ESR lines.

9.19 Port 15

This section describes the Port 15 functionality in detail.

9.19.1 Port 15 Configuration

Port 15 is a general-purpose 16-bit bi-directional port. It serves as GPIO lines without secondary functions.

Table 9-10 summarizes the I/O control selection functions of each Port 15 line.

9.19.2 Port 15 Function Table

Table 9-10 summarizes the I/O control selection functions of each Port 0 line.

Table 9-40 Port 15 Functions

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P15.0	I	General-purpose input	P15_IN.P0	P15_IOCR0. PC0	$0XXX_B$
	O	General-purpose output	P15_OUT.P0		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	$\overline{CS0}$		DIR
P15.1	I	General-purpose input	P15_IN.P1	P15_IOCR0. PC1	$0XXX_B$
	O	General-purpose output	P15_OUT.P1		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	$\overline{CS1}$		DIR
P15.2	I	General-purpose input	P15_IN.P2	P15_IOCR0. PC2	$0XXX_B$
	O	General-purpose output	P15_OUT.P2		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	$\overline{CS2}$		DIR
P15.3	I	General-purpose input	P15_IN.P3	P15_IOCR0. PC3	$0XXX_B$
	O	General-purpose output	P15_OUT.P3		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	$\overline{CS3}$		DIR

Table 9-40 Port 15 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P15.4	I	General-purpose input	P15_IN.P4	P15_IOCR4. PC4	$0XXX_B$
	O	General-purpose output	P15_OUT.P4		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	$\overline{BC0}$		DIR
P15.5	I	General-purpose input	P15_IN.P5	P15_IOCR4. PC5	$0XXX_B$
	O	General-purpose output	P15_OUT.P5		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	$\overline{BC1}$		DIR
P15.6	I	General-purpose input	P15_IN.P6	P15_IOCR4. PC6	$0XXX_B$
	O	General-purpose output	P15_OUT.P6		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	$\overline{BC2}$		DIR
P15.7	I	General-purpose input	P15_IN.P7	P15_IOCR4. PC7	$0XXX_B$
	O	General-purpose output	P15_OUT.P7		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	$\overline{BC3}$		DIR

Table 9-40 Port 15 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P15.8	I	General-purpose input	P15_IN.P8	P15_IOCR8. PC8	$0XXX_B$
	O	General-purpose output	P15_OUT.P8		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	$\overline{RD}$		DIR
P15.9	I	General-purpose input	P15_IN.P9	P15_IOCR8. PC9	$0XXX_B$
	O	General-purpose output	P15_OUT.P9		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	RD/$\overline{WR}$		DIR
P15.10	I	General-purpose input	P15_IN.P10	P15_IOCR8. PC10	$0XXX_B$
	O	General-purpose output	P15_OUT.P10		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	$\overline{ADV}$		DIR
P15.11	I	General-purpose input	P15_IN.P11	P15_IOCR8. PC11	$0XXX_B$
		EBU Input	$\overline{WAIT}$		
	O	General-purpose output	P15_OUT.P11		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$

Table 9-40 Port 15 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P15.12	I	General-purpose input	P15_IN.P12	P15_IOCR12. PC12	$0XXX_B$
	O	General-purpose output	P15_OUT.P12		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	MR/$\overline{W}$		DIR
P15.13	I	General-purpose input	P15_IN.P13	P15_IOCR12. PC13	$0XXX_B$
	O	General-purpose output	P15_OUT.P13		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	$\overline{BAA}$		DIR
P15.14	I	General-purpose input	P15_IN.P14	P15_IOCR12. PC14	$0XXX_B$
		EBU Input	BFCLKI		
	O	General-purpose output	P15_OUT.P14		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P15.15	I	General-purpose input	P15_IN.P15	P15_IOCR12. PC15	$0XXX_B$
	O	General-purpose output	P15_OUT.P15		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	BFCLKO		DIR

9.19.3 Port 15 Registers

The following registers are available on Port 15:

Table 9-41 Port 15 Registers

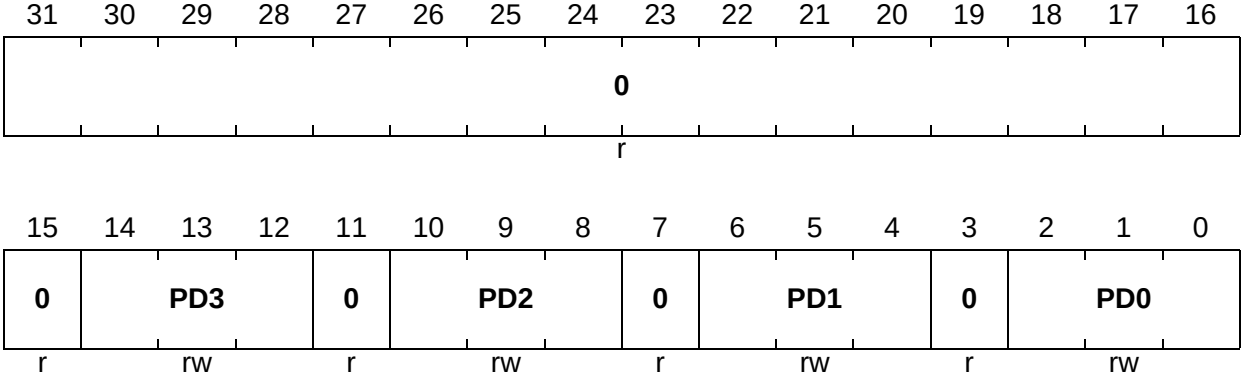
Register Short Name	Register Long Name	Offset Address	Description see
P15_OUT	Port 15 Output Register	0000_H	Page 9-15
P15_OMR	Port 15 Output Modification Register	0004_H	Page 9-16
P15_IOCR0	Port 15 Input/Output Control Register 0	0010_H	Page 9-9
P15_IOCR4	Port 15 Input/Output Control Register 4	0014_H	Page 9-10
P15_IOCR8	Port 15 Input/Output Control Register 8	0018_H	Page 9-10
P15_IOCR12	Port 15 Input/Output Control Register 12	$001C_H$	Page 9-11
P15_IN	Port 15 Input Register	0024_H	Page 9-19
P15_PDR	Port 15 Pad Driver Mode Register	0040_H	Page 9-125[1)]

1) This register is listed here in the Port 15 section because they differ from the general port register description given in **Section 9.3**.

9.19.3.1 Port 15 Pad Driver Mode Register and Pad Classes

The Port 15 pad driver mode register contains four bit fields that determine the pad driver mode (output driver strength and slew rate) of Port 15 line groups. The port lines are all class B1 pads.

P15_PDR
Port 15 Pad Driver Mode Register (40_H) Reset Value: 0000 0000_H



Field	Bits	Type	Description
PD0	[2:0]	rw	**Pad Driver Mode for P15.[3:0]** (Class B1 pads; for coding see **Page 9-13**)
PD1	[6:4]	rw	**Pad Driver Mode for P15.[7:4]** (Class B1 pads; for coding see **Page 9-13**)
PD2	[10:8]	rw	**Pad Driver Mode for P15.[11:8]** (Class B1 pads; for coding see **Page 9-13**)
PD3	[14:12]	rw	**Pad Driver Mode for P15.[15:12]** (Class B1 pads; for coding see **Page 9-13**)
0	3, 7, 11, [31:15]	r	**Reserved** Read as 0; should be written with 0.

9.20 Port 16

This section describes the Port 16 functionality in detail.

9.20.1 Port 16 Configuration

Port 16 is an 4-bit bi-directional general-purpose I/O port which can be used for the EBU I/O lines.

9.20.2 Port 16 Function Table

Table 9-26 summarizes the I/O control selection functions of each Port 16 line.

Table 9-42 Port 16 Functions

Port Pin	I/O	Pin Functionality	Associated Reg./I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P16.0	I	General-purpose input	P16_IN.P0	P16_IOCR 0.PC0	$0XXX_B$
		EBU	HOLD		
	O	General-purpose output	P16_OUT.P0		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
P16.1	I	General-purpose input	P16_IN.P1	P16_IOCR 0.PC1	$0XXX_B$
		EBU Input	HLDA		
	O	General-purpose output	P16_OUT.P1		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	HLDA		
P16.2	I	General-purpose input	P16_IN.P2	P16_IOCR 0.PC2	$0XXX_B$
	O	General-purpose output	P16_OUT.P2		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	BREQ		DIR
P16.3	I	General-purpose input	P16_IN.P3	P16_IOCR 0.PC3	$0XXX_B$
	O	General-purpose output	P16_OUT.P3		$1X00_B$
		Reserved	–		$1X01_B$
		Reserved	–		$1X10_B$
		Reserved	–		$1X11_B$
	HW_Out	EBU;EN	$\overline{\text{CSCOMB}}$		DIR

9.20.3 Port 16 Registers

The following registers are available on Port 16:

Table 9-43 Port 8 Registers

Register Short Name	Register Long Name	Offset Address	Description see
P16_OUT	Port 16 Output Register	0000_H	below[1]
P16_OMR	Port 16 Output Modification Register	0004_H	
P16_IOCR0	Port 16 Input/Output Control Register 0	0010_H	Page 9-9
P16_IN	Port 16 Input Register	0024_H	below[1]
P16_PDR	Port 16 Pad Driver Mode Register	0040_H	Page 9-129[1]

1) These registers are listed here in the Port 16 section because they differ from the general port register description given in Section 9.3.

9.20.3.1 Port 16 Output Register

The basic P16_OUT register functionality is described on Page 9-15. Port lines P16.[15:8] are not available. Therefore, the P16_OUT bits P[15:8] should be written with 0 and are always read as 0.

9.20.3.2 Port 16 Output Modification Register

The basic P16_OMR register functionality is described on Page 9-16. Port lines P8.[15:8] are not available. Therefore, the P8_OMR bits PS[15:8] and PR[15:8] are not implemented. These bits should always be written with 0.

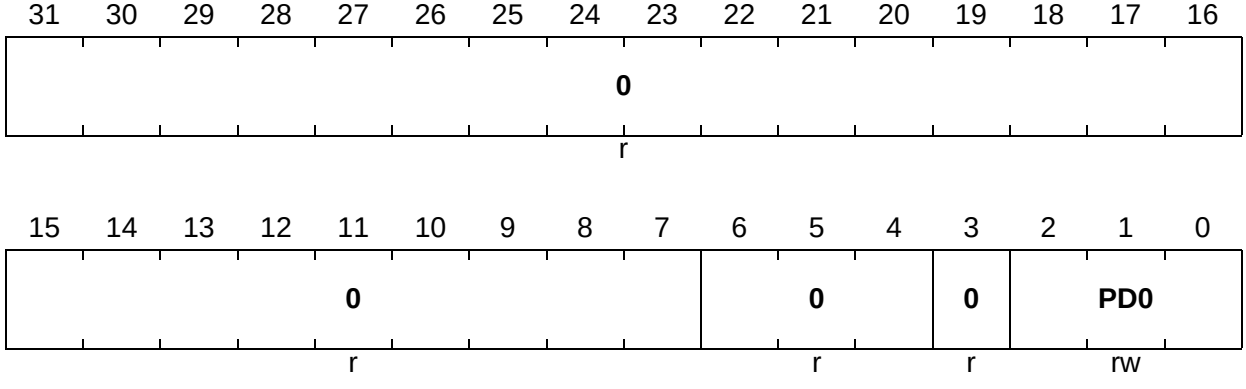
9.20.3.3 Port 16 Input Register

The basic P16_IN register functionality is described on Page 9-19. Port lines P16.[15:8] are not available. Therefore, the P16_IN bits P16.[15:8] are always read as 0.

9.20.3.4 Port 16 Pad Driver Mode Register and Pad Classes

The Port 16 pad driver mode register contains two bit fields that determine the pad driver mode (output driver strength and slew rate) of Port 8 lines and line groups. The port lines are assigned to B1 pad classes.

P16_PDR
Port 16 Pad Driver Mode Register (40_H) Reset Value: 0000 0000_H



Field	Bits	Type	Description
PD0	[2:0]	rw	**Pad Driver Mode for P16.[3:0]** (Class B1 pads; for coding see **Page 9-13**)
0	3, [31:4]	r	**Reserved** Read as 0; should be written with 0.

9.20.3.5 Port 16 Emergency Stop Register

The P16_ESR register is not implemented. Nevertheless, access to its address 0xF0300450 does not generate a Bus Error.

10 Peripheral Control Processor (PCP)

This chapter describes the Peripheral Control Processor (PCP), its architecture, programming model, registers, and instructions. The TC1797's PCP is an enhanced version of the TC1767's and TC1797's PCP peripheral control processor, which is an enhanced version of the TC1766's and TC1796's PCP, which is, in turn, an enhanced version of TC1775's PCP.

Section 10.2 to **Section 10.21** of this chapter describe the TC1797's PCP in general. TC1797 implementation-specific details are described in **Section 10.22**.

10.1 PCP Feature/Enhancement History List

The following table lists all PCP enhancements sequentially, for all versions of the PCP. The table will therefore list enhancements that may not apply to the PCP version included in TC1797.

Table 10-1 PCP Feature/Enhancement History List

Version	Enhancement
TC1775	First released version of PCP
TC1766, TC1796	• Optimised Context switching • Support for nested interrupts • Enhanced instruction set • Enhanced instruction execution speed • Enhanced interrupt queueing
TC1767, TC1797 Only enhancements in this row and above are included in TC1797	• Enhanced PCP core to support higher clock frequencies • Multiple clock ratios PCP:FPI 1:1 and 2:1 • Adaption of PCP Trace Interface to PAL-MCDS • Implementation of Parity
TC1387, TC1784, TC1798	• Implementation of ECC (replacing Parity) • Various Errata fixes.

10.1.1 Switchable Core Clock Ratio

For shorter interrupt latency, shorter uninterruptible task time and increased computing power the PCP is improved to allow operation with a core clock that is faster than the FPI clock (of the FPI Bus to which the PCP is connected). The available clocking ratios (f_{FPI}/f_{PCP}) are 1:1 and 1:2. The clock is controlled and generated in the SCU. The PCP has separate clock inputs for the core and for the FPI-bus related circuits.

10.2 Peripheral Control Processor Overview

The PCP in the TC1797 performs tasks that would normally be performed by the combination of a DMA controller and its supporting CPU interrupt service routines in a traditional computer system. It could easily be considered as the host processor's first line of defence as an interrupt-handling engine. The PCP can unload the CPU from having to service time-critical interrupts. This provides many benefits, including:

- Avoiding large interrupt-driven task context-switching latencies in the host processor
- Reducing the cost of interrupts in terms of processor register and memory overhead
- Improving the responsiveness of interrupt service routines to data-capture and data-transfer operations
- Easing the implementation of multitasking operating systems.

The PCP has an architecture that efficiently supports DMA-type transactions to and from arbitrary devices and memory addresses within the TC1797 and also has reasonable stand-alone computational capabilities.

10.3 PCP Architecture

The PCP is made up of several modular blocks as follows. Please refer to **Figure 10-1**.

- PCP Processor Core
- Code Memory (CMEM)
- Parameter Memory (PRAM)
- PCP Interrupt Control Unit (PICU)
- PCP Service Request Nodes (PSRN)
- System bus interface to the Flexible Peripheral Interface (FPI Bus)

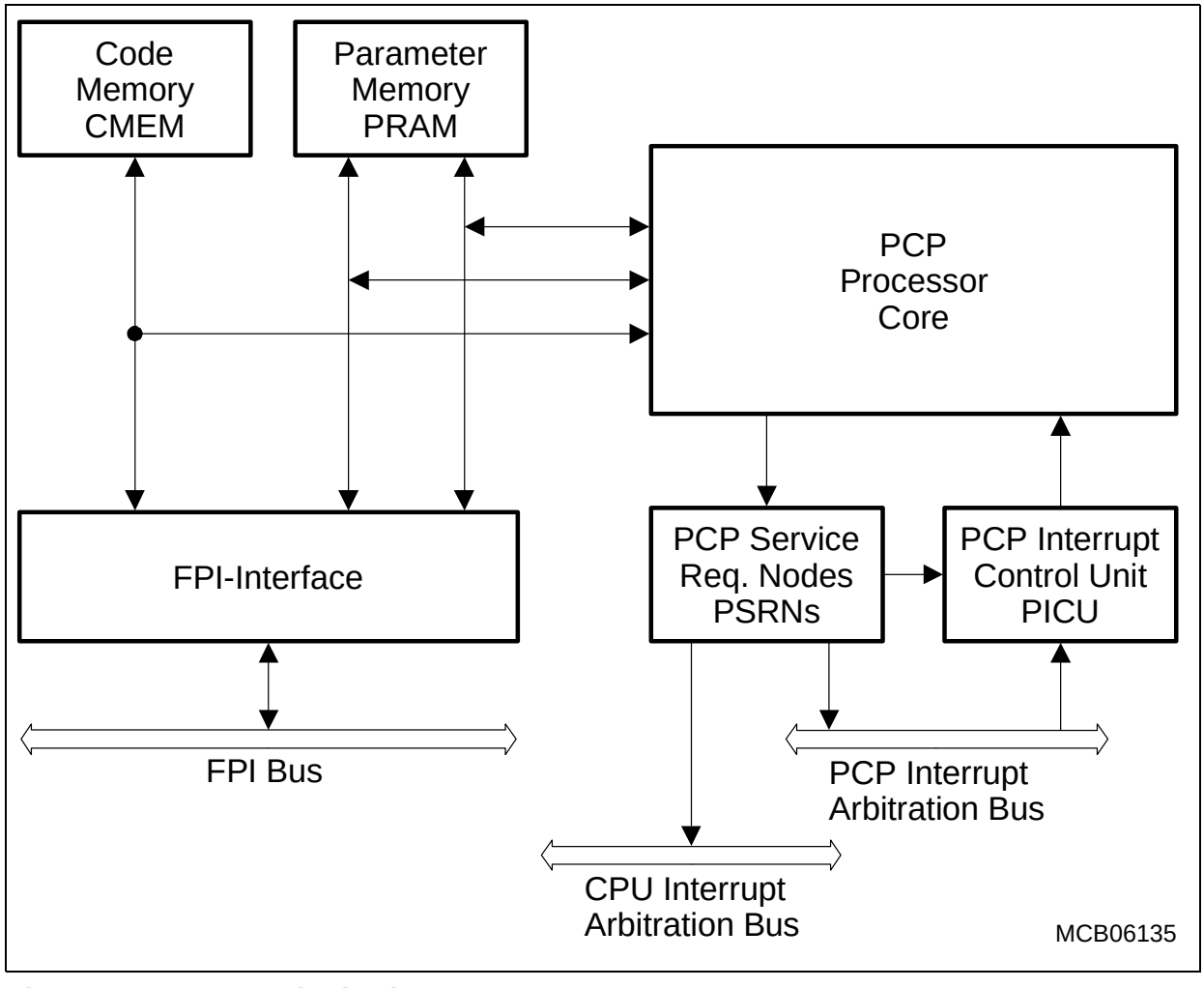


Figure 10-1 PCP Block Diagram

10.3.1 PCP Processor

The PCP Processor is the main engine of the PCP. It contains an instruction pipeline, a set of GPRs, an arithmetic/logic unit (ALU), as well as control and status registers and logic. Its instruction set is optimized especially for the tasks it has to perform. **Table 10-2** provides an overview of the PCP instruction set.

The PCP Processor Core receives service requests from peripherals or other modules in the system via its PCP Interrupt Control Unit (PICU) and executes a channel program (see **Page 10-7**) selected via the priority number of each service request. It first restores the channel program's context from the PRAM and then starts to execute the channel program's instructions stored in the Code Memory (CMEM). Upon an exit condition, it terminates the channel program and saves its context into PRAM. It is then ready to receive the next service request.

The PCP Processor Core is capable of suspending execution of a Channel Program on receipt of a service request with a higher priority than the channel currently being executed. The Core will automatically resume processing of the original Channel Program once the higher-priority request (or requests) has been processed. A channel that has been suspended in this way is termed as "Suspended Channel".

The PCP is fully interrupt-driven, meaning it is only activated through service requests; there is no main program running in the background as with a conventional processor.

Table 10-2 PCP Instruction Set Overview

Instruction Group	Description
DMA Primitives	Efficient DMA channel implementation
Load/Store	Transfer data between PRAM or FPI memory and the GPRs, as well as move or exchange values between registers
Arithmetic	Add, subtract, compare and complement
Divide/Multiply	Divide and multiply
Logical	And, Or, Exclusive Or, Negate
Shift	Shift right or left, rotate right or left, prioritize
Bit Manipulation	Set, clear, insert, and test bits
Flow Control	Jump conditionally, jump long, exit
Miscellaneous	No operation, Debug

10.3.2 PCP Code Memory

The Code Memory (CMEM) of the PCP holds the channel programs, consisting of PCP instructions. All instructions of the PCP are 16 bits long; thus, the PCP accesses its CMEM in 16-bit (half-word) quantities. With the 16-bit Program Counter (PC) of the PCP, a maximum of 64 K instructions can be addressed. This results in a maximum size of the PCP code memory of 128 Kbytes. The actual type (Flash, ROM, SRAM, etc.) and size of the code memory is implementation-specific; see **Page 10-138** for the implemented type and size of the code memory in the TC1797.

The PCP CMEM is viewed from the FPI Bus as a 32-bit wide memory, that must be accessed with 32-bit (word) accesses, and is addressed with byte addresses. Thus, care has to be taken when calculating PCP instruction FPI addresses. See **Page 10-52** for details.

Note: The PCP has a “Harvard” architecture and therefore cannot directly access the CMEM other than reading instructions from it. It is recommended that the PCP should not access CMEM via the FPI Bus.

10.3.3 PCP Parameter RAM

The PCP Parameter RAM (PRAM) is the local holding place for each channel program’s context, and for general data storage. It is also an area that the PCP and the host processor or other FPI Bus masters can use to communicate and share data.

While a portion of the PRAM is always implicitly used for the Context Save Areas (CSAs) of the channel programs (see **Chapter 10.4.2.2**), the remaining area can be used for channel-specific or general data storage. A programmable 8-bit Data Pointer (DPTR), concatenated with a 6-bit offset, is provided for arbitrary access to the PRAM. The effective address is a 14-bit word address, allowing a PRAM size of up to 64 Kbytes. The actual type (SRAM, DRAM, etc.) and size of the parameter RAM is implementation-specific; see **Page 10-138** for the implemented size of the PRAM in this derivative.

Both the PCP and FPI Bus masters address the PRAM as 32-bit words. There is no concept of half-word or byte accesses to PRAM. FPI Bus masters must, however, use byte addresses in order to access the PRAM. As for the CMEM, care has to be taken when calculating PRAM FPI addresses. See **Page 10-52** for details.

10.3.4 FPI Bus Interface

The PCP can access all peripheral units on the FPI Bus and other resources through the FPI Bus interface. The PCP can become an FPI Bus slave, so that other FPI Bus master may access CMEM and PRAM and the control and status registers in the PCP.

The CMEM and PRAM blocks are visible to FPI Bus masters as a block of memory on the FPI Bus. If an FPI Bus master accesses CMEM or PRAM memory concurrently with the PCP, the external FPI Bus master is given precedence over the PCP to avoid deadlocks. The PCP access is stalled for several cycles until the FPI Bus master has

completed its access. If an FPI Bus master performs an atomic read-modify-write access to a PCP memory block, any concurrent PCP access to that block is stalled for the duration of the atomic operation.

10.3.5 PCP Interrupt Control Unit and Service Request Nodes

The PCP is activated in response to an interrupt request programmed for PCP service in one of the Service Request Nodes (SRNs) of the system (nodes associated with a peripheral, the CPU, external interrupts, etc.). The PCP Interrupt Control Unit (PICU) determines the request with the currently highest priority and routes the request together with its priority number to the PCP Processor Core. It also acknowledges the requesting source when the PCP starts the service of this interrupt.

The PCP itself can generate service requests to either the CPU or itself through a number of PCP Service Request Nodes (PSRNs). The PSRNs are also used to store all information required by the PCP Processor Core to allow the later restart of a channel program when it is suspended in favor of a higher-priority Service Request. Please refer to **Section 10.6.3** for more detailed information on the operation of these nodes.

10.4 PCP Programming Model

The PCP programming model can be viewed as a set of autonomous programs, or tasks, called channel programs, that share the processing resources of the PCP. Channel programs may be short and simple, or very complex; but they can coexist persistently within the PCP.

From the programming point of view, the individual parts of a channel program are its instruction sequence in the CMEM and its context in the PRAM. It uses the instruction set and the GPRs (R0 - R7) of the PCP Processor Core to perform the necessary operations, and to communicate with the various resources of the on-chip and off-chip system depending on its task in the application.

These parts of the programming model are discussed in the following sections (with the obvious exception of the system environment outside of the scope of the PCP).

10.4.1 General Purpose Register Set of the PCP

The program-accessible register file of the PCP is composed of eight 32-bit General Purpose Registers (GPRs). These registers are all accessible by PCP programs directly as part of the PCP instruction set. Source and destination registers must be specified in most instructions. These registers are referenced to in this document as Rn or R[n], where n is in the range 0 to 7.

Table 10-3 Directly Accessible Registers

Register	Implicit Use	Description
R0	Accumulator	Implicit target for some arithmetic and logical instructions
R1	–	32-bit general-use register
R2	Return Address	32-bit general-use register
R3	–	32-bit general-use register
R4	SRC (Source)	Source Pointer for BCOPY/COPY instructions
R5	DST (Destination)	Destination Pointer for BCOPY/COPY instruction
R6	CPPN/SRPN/ TOS/CNT1	CNT1: Transfer Count for COPY TOS: Type-of-Service SRPN: 8-bit field used for posting interrupt on EXIT instruction CPPN: Current PCP Priority Number
R7	DPTR/Flags	PRAM Data Pointer (DPTR) and Status Flags

R7 is the only one of the eight registers that may not be used as a full GPR. The most significant 16 bits of R7 may not be written, and will always read back as 0. However, no error will occur when writing to the most significant 16 bits.

Note: The GPRs of the PCP are not memory-mapped into the overall address space. They can only be directly accessed through PCP instructions. The contents of all or some of the registers are part of a channel program's context stored in the PRAM between executions of the channel program. This context is then accessible from outside the PCP.

10.4.1.1 Register R0

R0 is used as an implicit operand destination for some instructions. These are detailed in the individual instruction descriptions.

10.4.1.2 Registers R1, R2, and R3

R1, R2, and R3 are general-use registers. It is recommended that, by convention, R2 should be used as a return address register when call and return program structures are used.

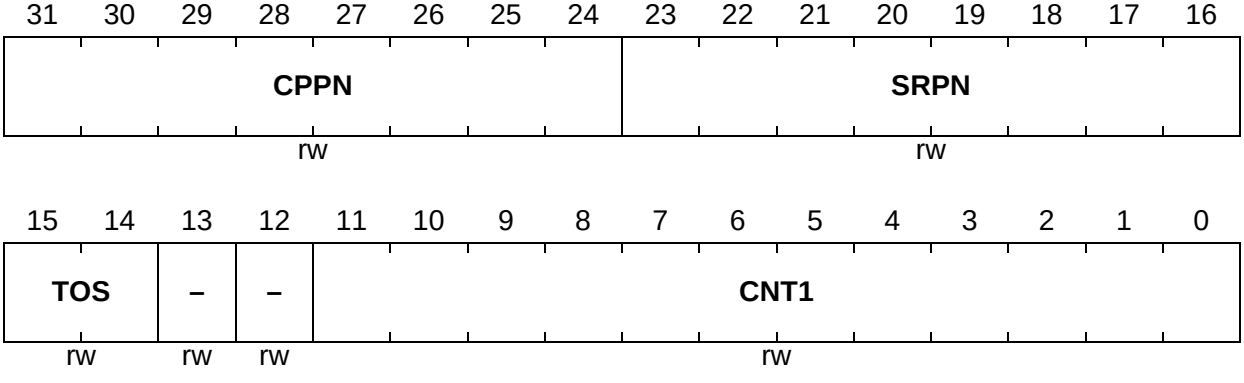
10.4.1.3 Registers R4 and R5

Registers R4 and R5 are also general-use registers. However, the BCOPY/COPY instructions implicitly use R4 and R5 as full 32-bit address pointers (R4 is used as the source address and R5 as the destination address). As the BCOPY/COPY instructions use these registers to maintain the address pointers, either or both R4 and R5 values may or may not be modified by the BCOPY/COPY instructions, depending on the options used in the instructions.

10.4.1.4 Register R6

Register R6 may also be used as a general-use register. Again however, there are some instructions that use fields within R6. If the COPY or EXIT instructions are used, then the field R6.CNT1 can optionally be used implicitly as a counter. If an EXIT instruction is used that causes an interrupt, R6.SRPN and R6.TOS must be configured properly prior to execution of the EXIT. If interrupt priority management is used, then R6.CPPN must be set to the priority level at which the channel shall run at its next invocation, before the EXIT is executed. The fields for R6 are shown below.

PCP Register R6 **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
CPPN	[31:24]	rw	**General-use/PCP Priority Number Posted to PICU**
SRPN	[23:16]	rw	**General-use/Service Request Priority Number for EXIT Interrupt**
TOS	[15:14]	rw	**General-use/Type-of-Service for EXIT Interrupt** Upper bit of TOS is always forced to 0 when transferred into the PCP SRNs, regardless of the value specified in R6[15].
–	13	rw	**General-use**
–	12	rw	**General-use**
CNT1	[11:0]	rw	**General-use/Outer Loop count for COPY Instruction or EXIT Instruction**

10.4.1.5 Register R7

Register R7 is an exception with respect to the other registers in that not all bits within the register can be written, and the implicit use of the remaining bits virtually excludes the use of R7 as a GPR. R7 serves similar purposes as those in the Program Status Word found in traditional processors.

R7 holds the flag bits, a channel enable/disable control bit, and the PRAM Data Pointer (DPTR). The upper 16 bits of R7 are reserved.

Most instructions of the PCP update the flags (CN1Z, V, C, N, Z) in R7 according to the result of their operation. See **Table 10-15** on **Page 10-115** for details on the flag updates of the individual instructions. The values of the flag bits in R7 maintain their state until another instruction that updates their state is executed.

Note: Implicit updates to the flags caused by instruction take precedence over any bits that are explicitly moved to R7. For example, if a MOV instruction is used to place $0000FF07_H$ in R7, then the bit positions for the C (carry), Z (zero) and N (negative) flags are being written with 1. The MOV instruction, however, implicitly updates the Z and N flag bits in R7 as a result of its operation. Because the number is not negative, and not zero, it will update the Z and N flags to 0. As a result, the value left in R7 after the MOV is complete will be $0000FF04_H$ (i.e C = 1, Z = 0, N = 0). It is recommended that only SET and CLR instructions should be used to explicitly modify flags in R7.

The Data Pointer, R7.DPTR, is the means of accessing PRAM variables programmatically. It points to a 64-word PRAM segment that may be addressed by instructions that can use the PRAM for source or destination operands (xx.P and xx.PI instructions). The 8 bits of the DPTR are concatenated with a 6-bit offset value (either specified in the instruction as an immediate value or contained in one of the registers) to give a 14-bit (word) address. A program is able to update the DPTR value dynamically, in order to index more than 64 words of PRAM.

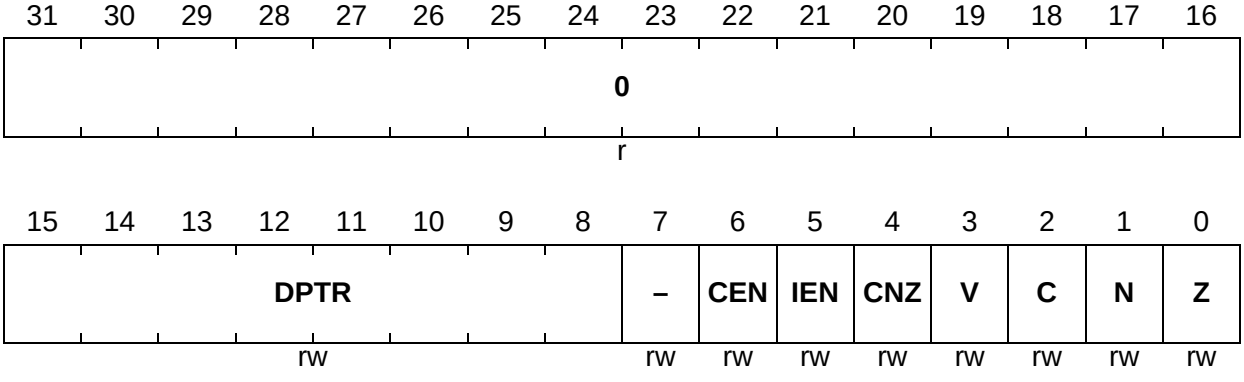
Note: Care must be taken when updating R7.DPTR to ensure that other bits within R7 (e.g. R7.CEN) are not inadvertently corrupted.

The channel enable control bit, R7.CEN, allows the enabling or disabling of specific channel programs. If an interrupt request is received for a channel that is disabled an error exit is forced, and an error interrupt to the CPU is activated.

The interrupt enable control bit, R7.IEN, allows the enabling or disabling of channel interruption on a channel to channel basis. When R7.IEN is 0, the channel will continue its execution regardless of the priority of any new service requests. When R7.IEN is 1, and conditions allow, the channel will be suspended on receipt of a higher-priority service request.

*Note: See **Section 10.21.3.1** regarding the use of R7.IEN.*

PCP Register R7 **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
0	[31:16]	r	**Reserved;** read as 0; should be written with 0.
DPTR	[15:8]	rw	**Data Pointer Segment Address for PRAM accesses**
–	7	rw	**Reserved;** should always be written with 0.
CEN	6	rw	**Channel Enable Control Bit**
IEN	5	rw	**Interrupt Enable** 0 Channel is not interruptible 1 Channel can be suspended in favor of a higher-priority service request
CNZ	4	rw	**Outer Loop Counter 1 Zero Flag**
V	3	rw	**Overflow**
C	2	rw	**Carry**
N	1	rw	**Negative**
Z	0	rw	**Zero**

10.4.2 Contexts and Context Models

After initialization, the instruction sequence of a PCP channel program is permanently stored (i.e. usually at least as long as the application is running) in the CMEM, and data parameters are held in the PRAM. These will remain stored regardless of whether a particular channel program is currently idle or is executing (although, of course, the value of data variables in the PRAM might be modified by the PCP or external FPI Bus masters). The contents of the GPRs of the PCP (used as address pointers, data variables, intermediate results, etc.) however, are usually only valid for a given channel program as long as it is executing. If another channel program is executed, the channel program will re-use the GPRs according to its needs.

Thus, the state of the GPRs of a channel program (termed the “Context” of the channel) needs to be preserved while a channel program is not being executed. The content of the registers needs to be saved when execution of a channel program finishes, and needs to be restored before execution starts again.

The PCP implements automatic handling of these context save and restore operations. On termination of a channel program, the state of all or some of the GPRs is automatically copied to a defined area in the PRAM (Context Save). If the same channel program is re-activated, the contents of the registers are restored by copying the values from the same defined PRAM area into the appropriate registers (Context Restore).

The defined area in the PRAM for the context save and restore operations is called the CSA. Each channel program has its own individual, predefined region in the CSA. When a service request is accepted by the PCP, the service request priority number (SRPN) associated with the request is used to select the channel program and its respective CSA region.

10.4.2.1 Context Models

A Context Model is a means of selecting whether some or all of the registers are saved and restored when a context switch occurs. In order to serve different application needs in terms of PRAM space usage, the PCP offers a choice between three different Context Models:

- Full Context Model: Eight Registers (8 × 32-bit words) are saved/restored per channel.
- Small Context Model: Four Registers (4 × 32-bit words) are saved/restored per channel.
- Minimum Context Model: Two Registers (2 × 32-bit words) are saved/restored.

As illustrated in **Figure 10-2**, the contents of R0 through R7 constitute the Full Context of a channel program. A Small Context consists of R4 through R7. Use of the Small Context Model allows for correct operation of DMA channels, as well as channels which are not required to save large amounts of data in their contexts between invocations. A Minimum Context saves and restores only R6 and R7.

To distinguish the actual register contents from the copies stored in the PRAM context regions, the term CRx is used throughout the rest of this document to refer to the register values in the context regions. Registers R6 and R7 are always handled in a special way during context save and restore operations, this is described in detail in **Section 10.4.2.3**.

The Context Model is selected via bit field (PCP_CS.CS), this is a global setting (i.e. the selected Context Model is used for all channels). Once a context model has been selected (during PCP configuration) and the PCP has been started the PCP must continue to use that Context Model. Attempting to change the Context Model in use during PCP operation will lead to invalid context restore operations which will in turn lead to invalid PCP operation.

In the case of Small and Minimum Context Models, the unsaved and un-restored registers (shaded in **Figure 10-2**) can be thought of as global registers that any channel program can use or change, or reference as constants – for example as base address pointers (see **Section 10.20.2** for more detail).

*Note: Special care must be taken when using Minimum or Small Context Model with nested interrupts (see **Page 10-132**).*

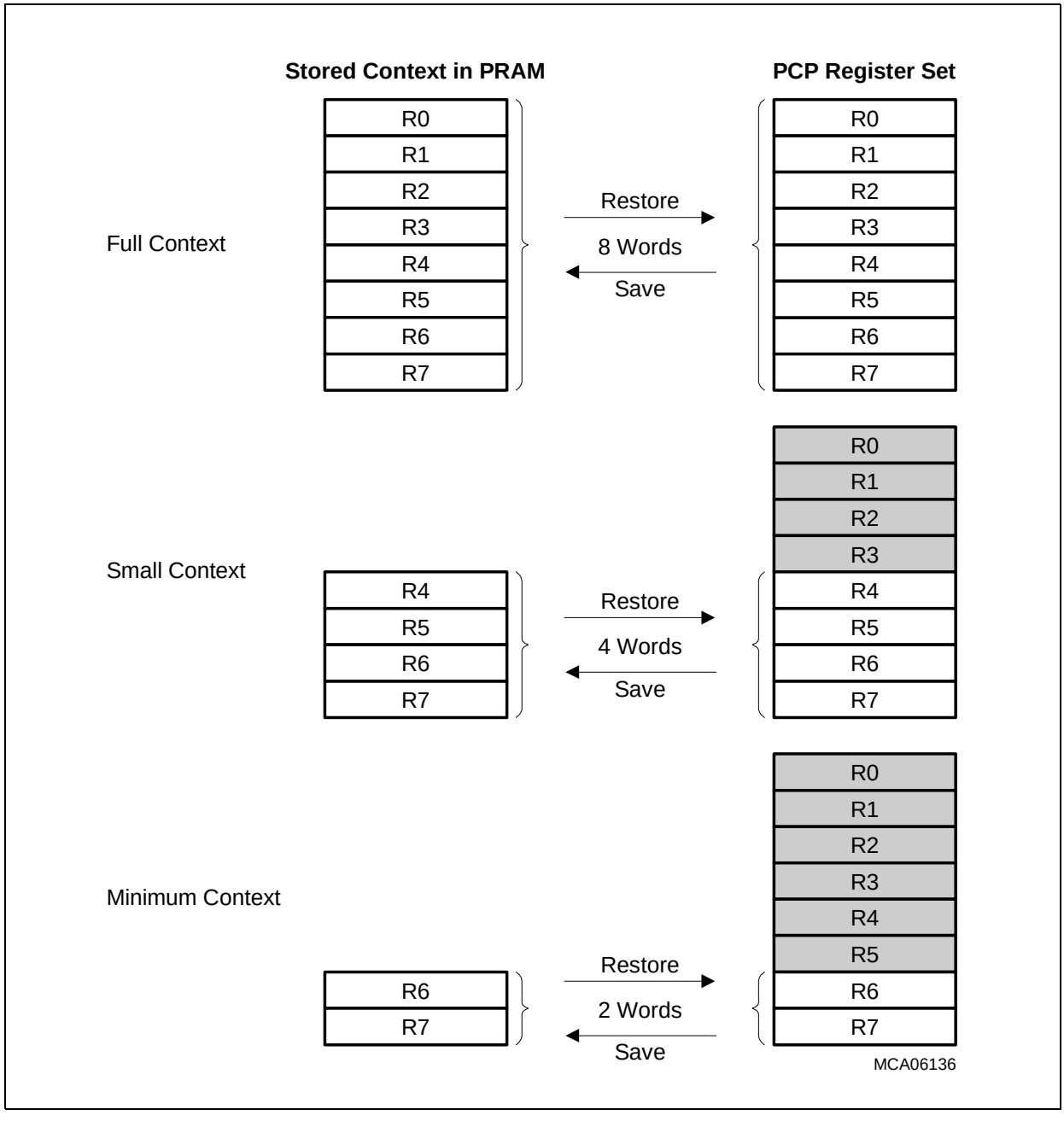


Figure 10-2 PCP Context Models

10.4.2.2 Context Save Area

The Context Save Area (CSA) is a region in PRAM reserved for storing the contexts for all channel programs (while any particular channel is not executing). Each channel's context is stored in a region of the CSA based on the channel number. The channel number is equal to the priority number (SRPN) of the service request. The PCP uses this number to calculate the start address of the context of the associated channel program. The size of a context is determined by the Context Model that the PCP has been initialized to use. As all channels use the same context size, the PRAM address (word address) of the context for a particular channel is simply calculated by multiplying the channel number by the number of registers in the context (8 for Full Context, 4 for Small Context and 2 for Minimum Context). **Figure 10-3** shows the resulting PRAM layout, and from this it can be seen that changing the Context Model also changes the base address for all regions within the CSA. Thus, the chosen Context Model may only be set when the PCP is initialized, and may not be changed during operation.

The CSA in the PRAM starts at address 00_H and grows upward. It is partitioned into equally sized regions, where the size of these regions is determined by the selected Context Model. The priority number (SRPN) of a service request is used to access the appropriate context region for the associated channel program. Since a request with an SRPN of 00_H is not considered as valid request in the TriCore Architecture, the bottom region (context region 0) of the CSA is never used for an actual context.

The total size of the CSA depends on the Context Model and the number of service request numbers used in a given system. Each priority number used in a service request node which can activate interrupts to the PCP must be represented through a dedicated context region in the PRAM. The highest address range in the PRAM used for a context region is determined by the highest priority number presented to the PCP with a service request.

The range of usable priority numbers is further determined by the size of the implemented PRAM and by the space required for other variables and global data located in the PRAM. See **Page 10-138** for the implemented size of the PRAM in the TC1797. As an example, a PRAM of 2 Kbytes, solely used for the CSA, can store up to 255 Minimum Contexts, allowing the highest SRPN used for a PCP service request to be 255 (remember, an SRPN of 0 and an associated context region is never used; the valid SRPNs and the context and channel numbers range from 1 to 255). With a Small Context Model, 127 contexts can be stored, resulting in 127 being the highest usable SRPN in this configuration. Finally, a Full Context Model allows 63 context areas, with 63 being the highest usable SRPN. Interrupt requests to the PCP with priority numbers that would cause loading of a context from outside the available PRAM area must not be generated. Invalid PCP operation will result should this situation be allowed to occur. The PCP can be optionally configured such that if an interrupt request is received that would cause loading of a context from outside the available PRAM area then an error exit is forced, and an error interrupt to the CPU is activated (see **Page 10-40**).

If portions of the PRAM are used for other variables and global data, the space available for the CSA and the range of valid SRPNs is reduced by the memory space required for this data. For best utilization of PRAM, it is advisable to have the CSA grow upwards as a contiguous area without any “holes”, meaning that all SRPNs in the range 1 … max. are actually used to place interrupt requests on the PCP. Unused regions within the CSA (that is, the unused region at the base of the CSA and any context regions associated with unused channels) cannot be used for general variable storage.

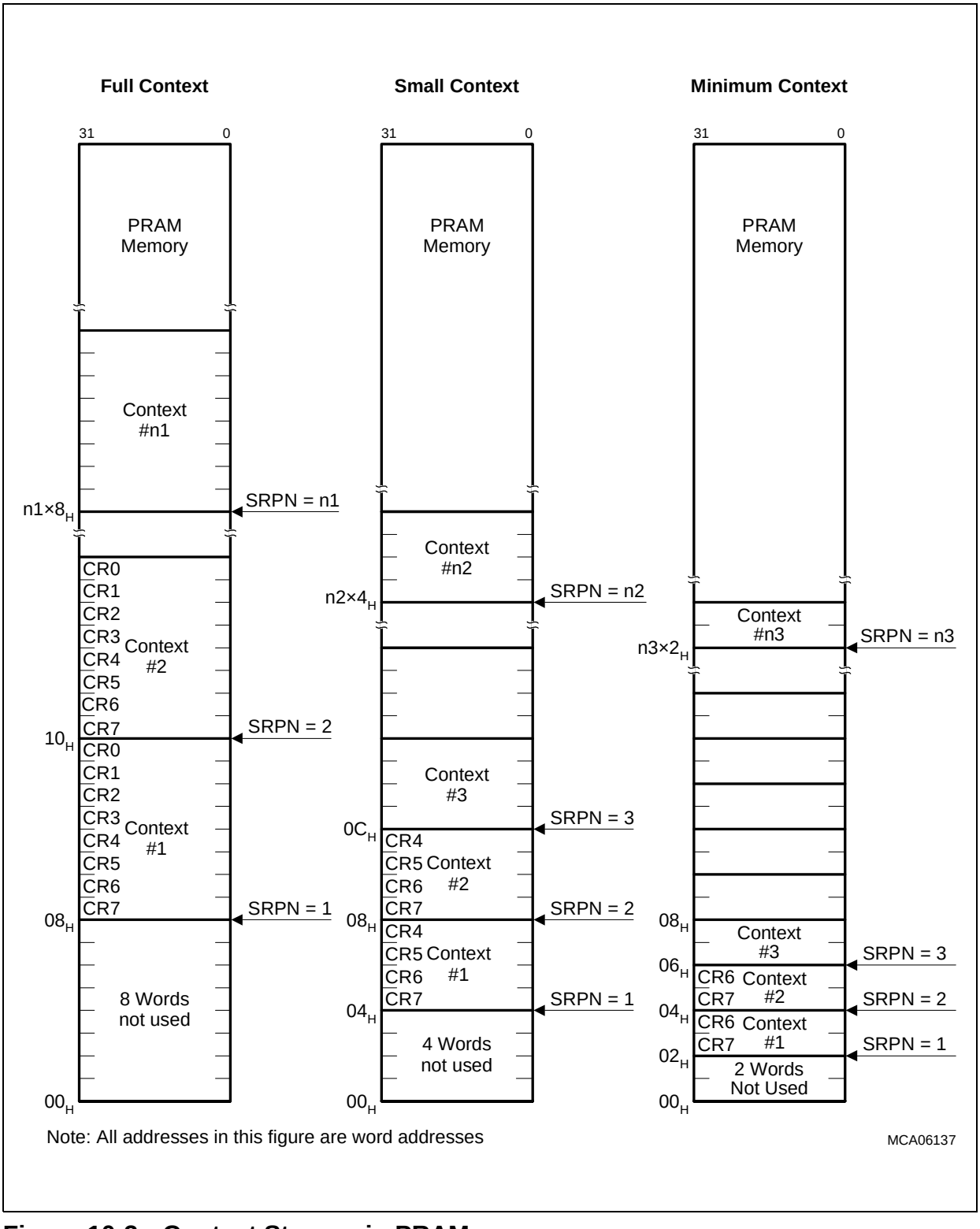


Figure 10-3 Context Storage in PRAM

When choosing the Context Model for a given application, the following considerations can be helpful. When choosing the Small or the Minimum Context Models, save and restore operations for registers not handled in the automatic context operations can still be handled through explicit load and store instructions under control of the user. This may be advantageous for applications in which the majority of the channels do not need the Full Context, and only some would require more context to be saved. In this case, a Smaller Context Model can be used, and the channels which would require more register to be saved/restored would do this via explicit load and store instructions. This is especially advantageous if the channel program can be designed such that the initial real-time response operations can be executed using only the registers which have been automatically restored. Then, as the timing requirements of the service are relaxed, further register contents can be restored from PRAM through regular load instructions. Of course, the contents of these registers needs to be explicitly saved, through regular store instructions, before the exit of the channel program.

*Note: Special care must be exercised when using Minimum or Small Context Models in conjunction with nested interrupts (see **Page 10-132**).*

The criteria for choosing the Context Model are listed in the following:

- Size of PRAM implemented in a given derivative
- Amount of channels (= SRPNs) that need to be used in a system
- Amount of PRAM used for general variables and global’s
- Amount of context (register content) which need to be saved and restored quickly by most of the most important channels

While registers R0 through R5 are always restored in a normal manner (according to the context size), registers R6 and R7 merit discussion regarding context restore operations. The memory location CR7 in a context region is used to hold two different pieces of information: The lower part of register R7, and the PC value of the channel. Similarly, the memory location CR6 in a context region can also be used to hold two different pieces of information: The value to be restored to register R6, and the Operating Priority (CPPN) value of the channel. This leads to the Restore/Save operations described in the following two sections.

10.4.2.3 Context Restore Operation for CR6 and CR7

The operation of R6 and R7 context restore varies according to whether the channel program that is starting is a “new” channel program (i.e. a channel program that is starting in response to the receipt of a new service request) or is a “suspended” channel program (i.e. a channel program that is re-starting after being suspended in favor of a higher-priority channel program). In addition, when a “new” channel program is starting, the context restore operation depends on the channel start mode that has been selected (see **Page 10-26**).

Channel Resume Mode

Figure 10-4 illustrates the operation of a context restore for a “new” channel program when Channel Resume Mode has been selected (see **Page 10-27**). The PC is loaded from CR7[31:16], and the lower half of R7 is loaded from CR7[15:0]. The operating priority of the channel is taken from CR6[31:24] and all of R6 is loaded from CR6.

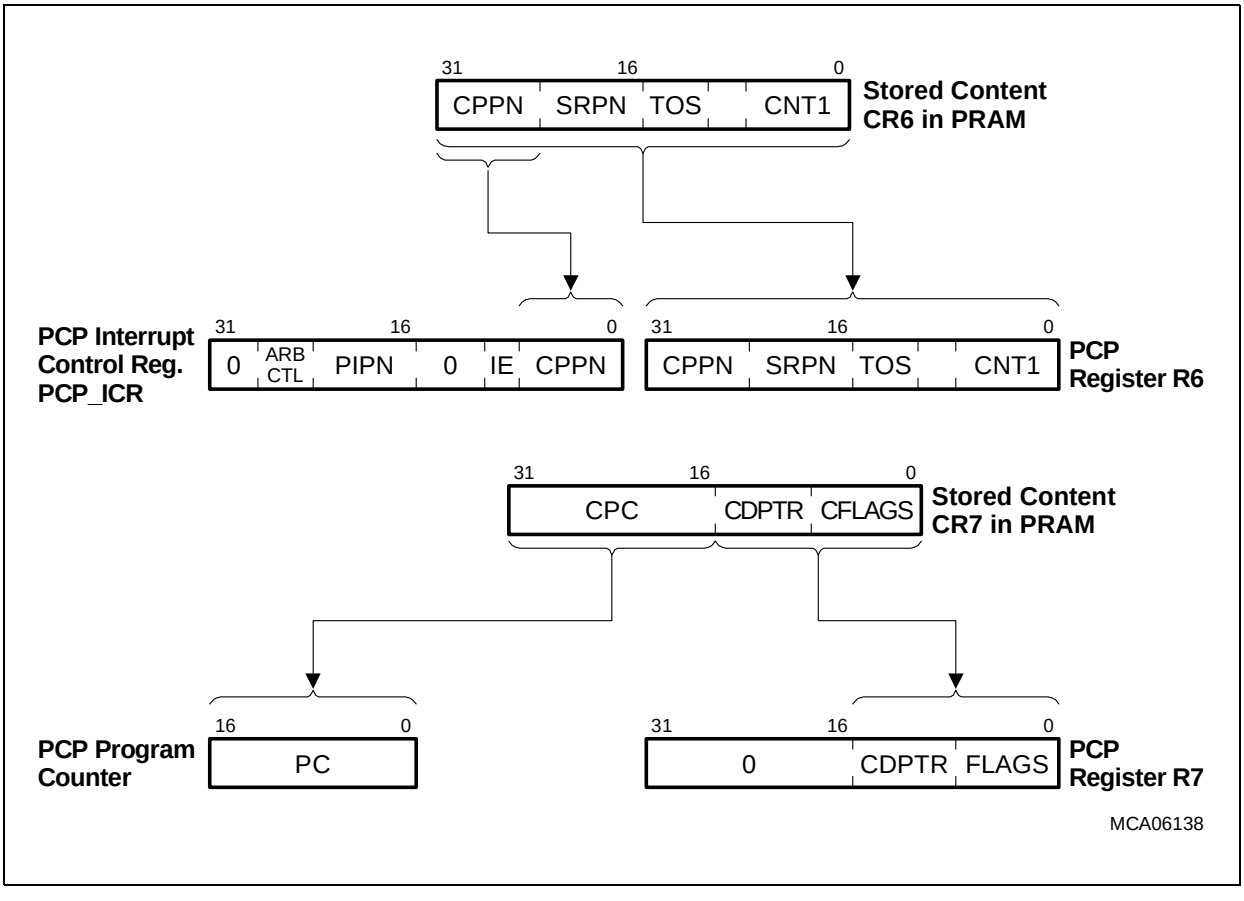


Figure 10-4 Context Restore: Channel Start in “Channel Resume Mode”

Channel Restart Mode

Figure 10-5 illustrates the operation of a context restore for a “new” channel program when Channel Restart Mode has been selected (see **Page 10-26**). The PC is loaded with the channel entry table address, and the lower half of R7 is loaded from CR7[15:0]. The upper half of CR6 is discarded. The operating priority of the channel is taken from CR6[31:24] and all of R6 is loaded from CR6.

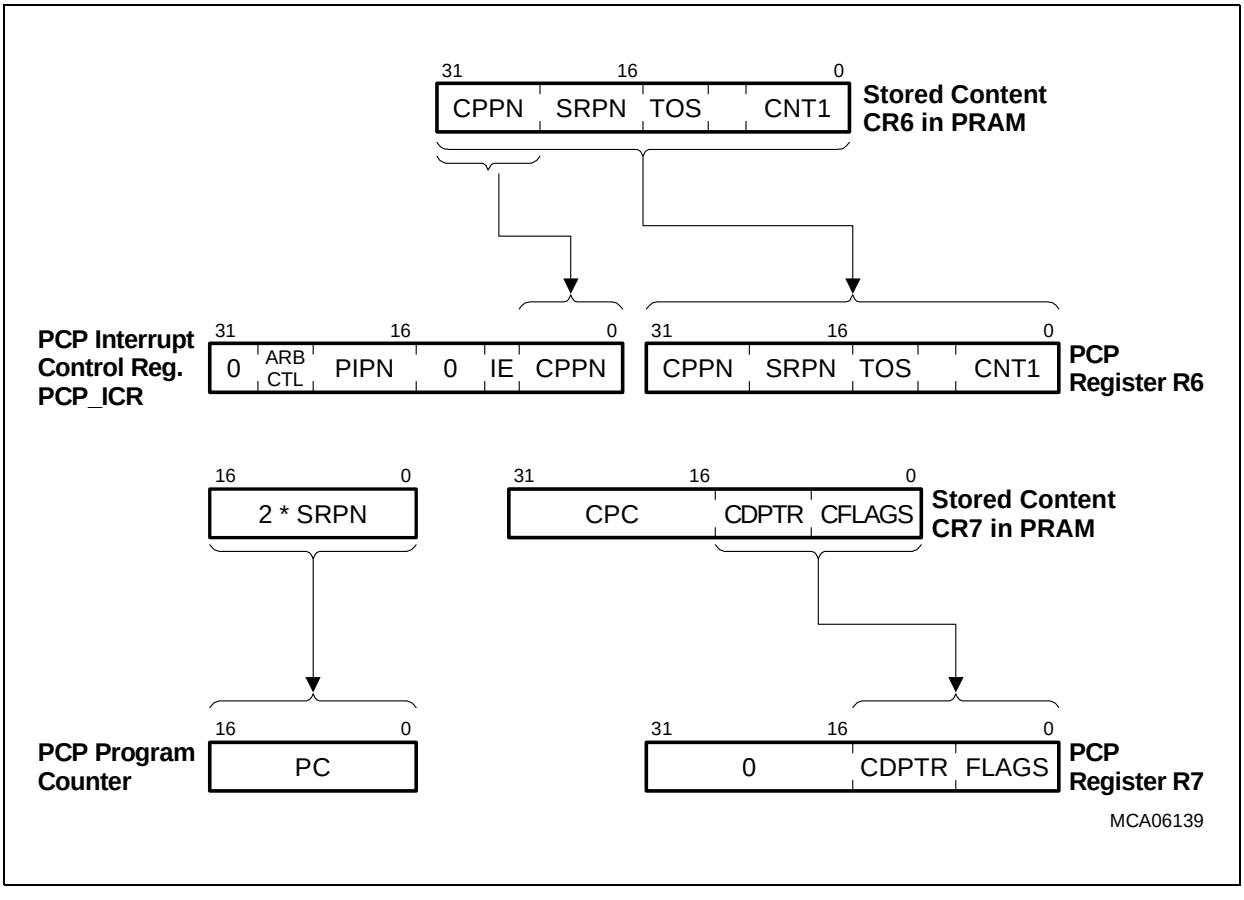
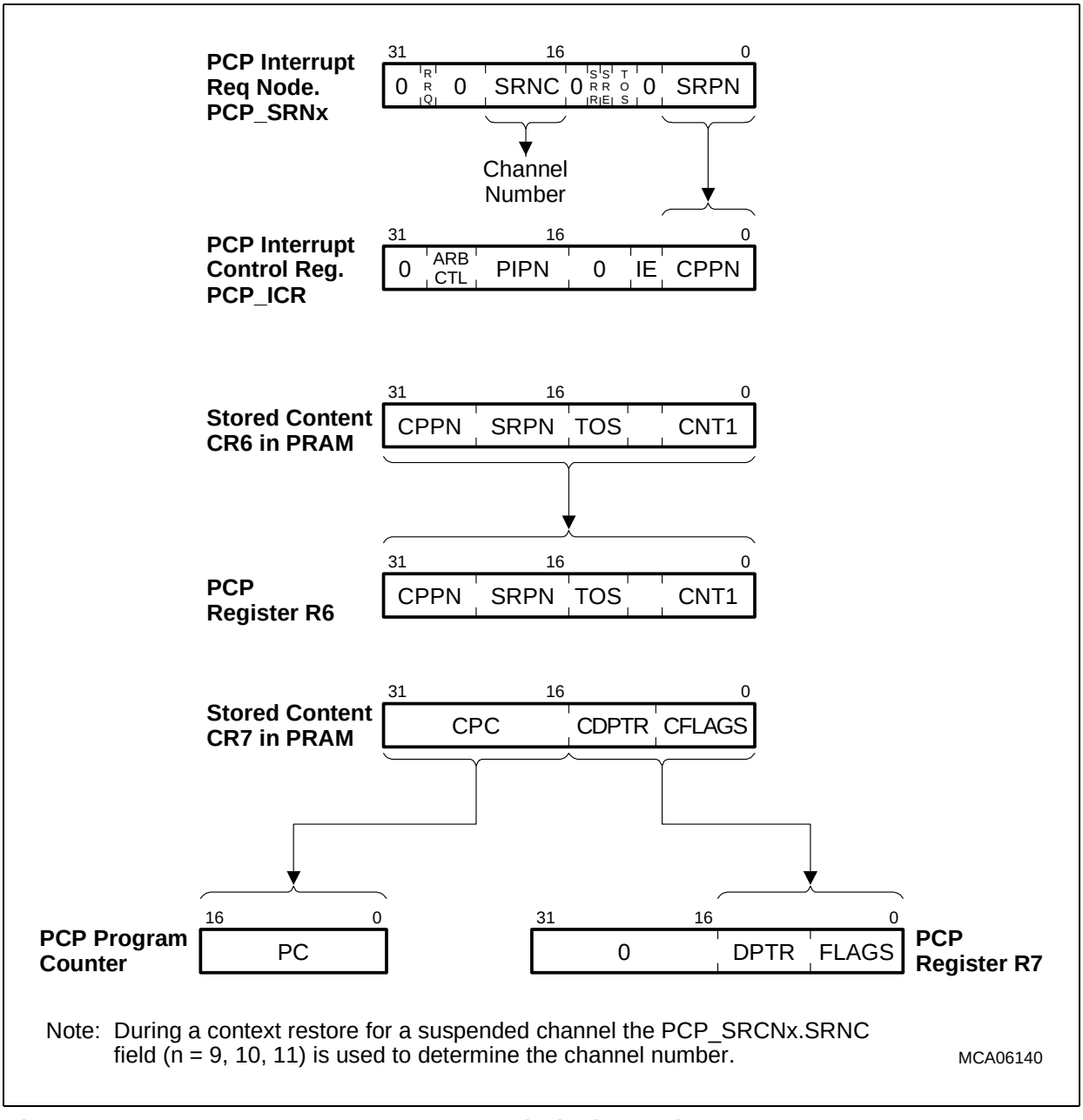


Figure 10-5 Context Restore: Channel Start in Channel Restart Mode

Suspended Channel Restart

Figure 10-6 illustrates the operation of a context restore for a “suspended” channel program. The PC is loaded from CR7[31:16] (regardless of the Channel Start Mode), and the lower half of R7 is loaded from CR7[15:0]. All of R6 is loaded from CR6. The figure also shows how the operating priority of the channel (PCP_IR.CPPN) is restored from the Service Request Node that was used to store the Suspended Interrupt Request (see **Page 10-77**).



Note: During a context restore for a suspended channel the PCP_SRCNx.SRNC field (n = 9, 10, 11) is used to determine the channel number.

MCA06140

Figure 10-6 Context Restore: Suspended Channel Restart

10.4.2.4 Context Save Operation for CR6 and CR7

The operation of R6 and R7 context save varies according to whether the save operation is the result of a channel exit condition, or whether the channel is being suspended in favor of a higher-priority channel program.

Channel Resume Mode

Figure 10-7 illustrates the operation of a context save for a channel exit when Channel Resume Mode has been selected. The value written to CR7 is created by concatenating the 16-bit PC value with the lower 16 bits of R7. CR6 is written with the value taken from R6.

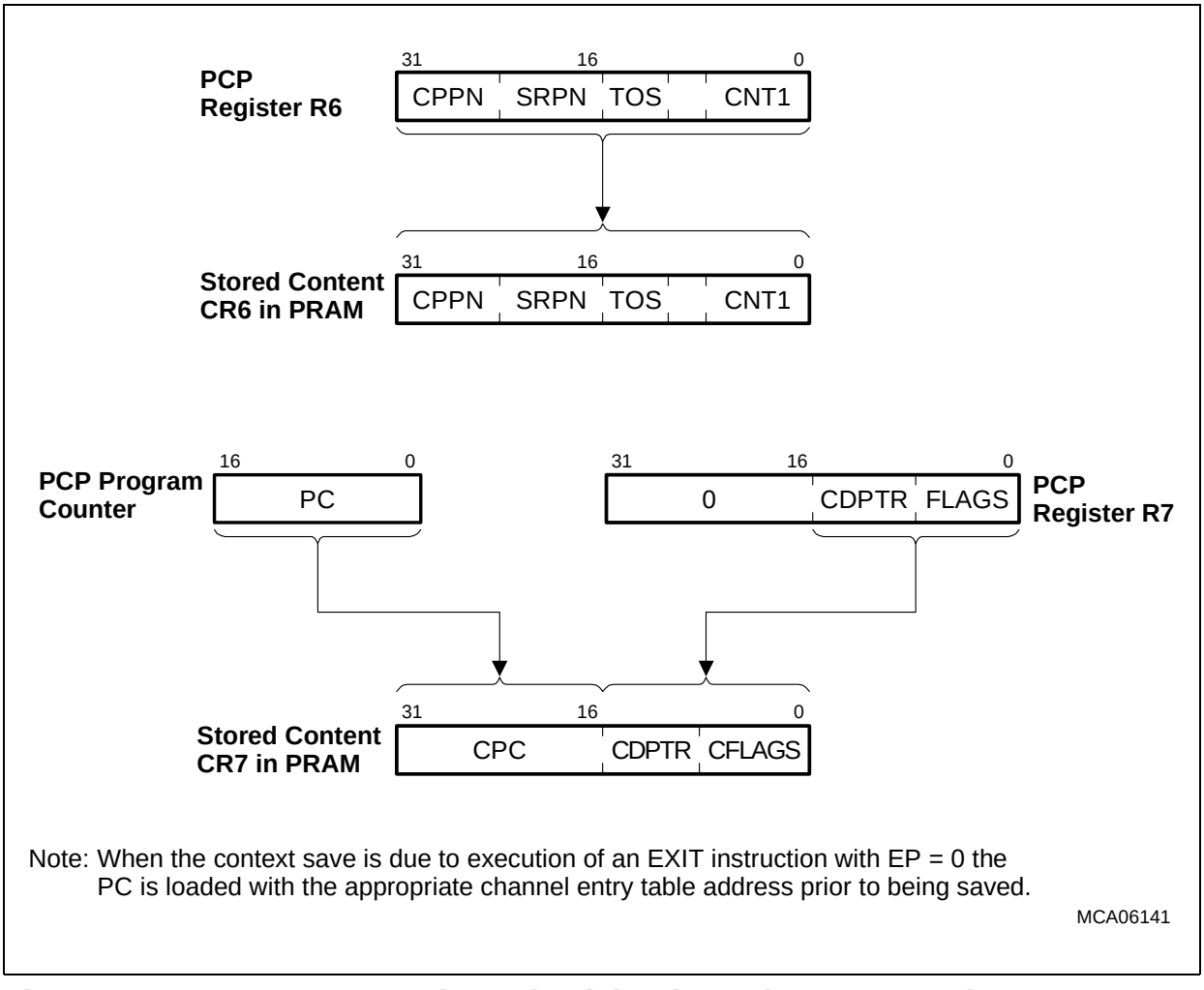


Figure 10-7 Context Save: Channel Exit in Channel Resume Mode

Channel Restart Mode

Figure 10-8 illustrates the operation of a context save for a channel exit when Channel Restart Mode has been selected. This is the same as for Channel Resume mode except that the PC value is discarded, and the appropriate Channel Entry Table address is written to CR7[31:16].

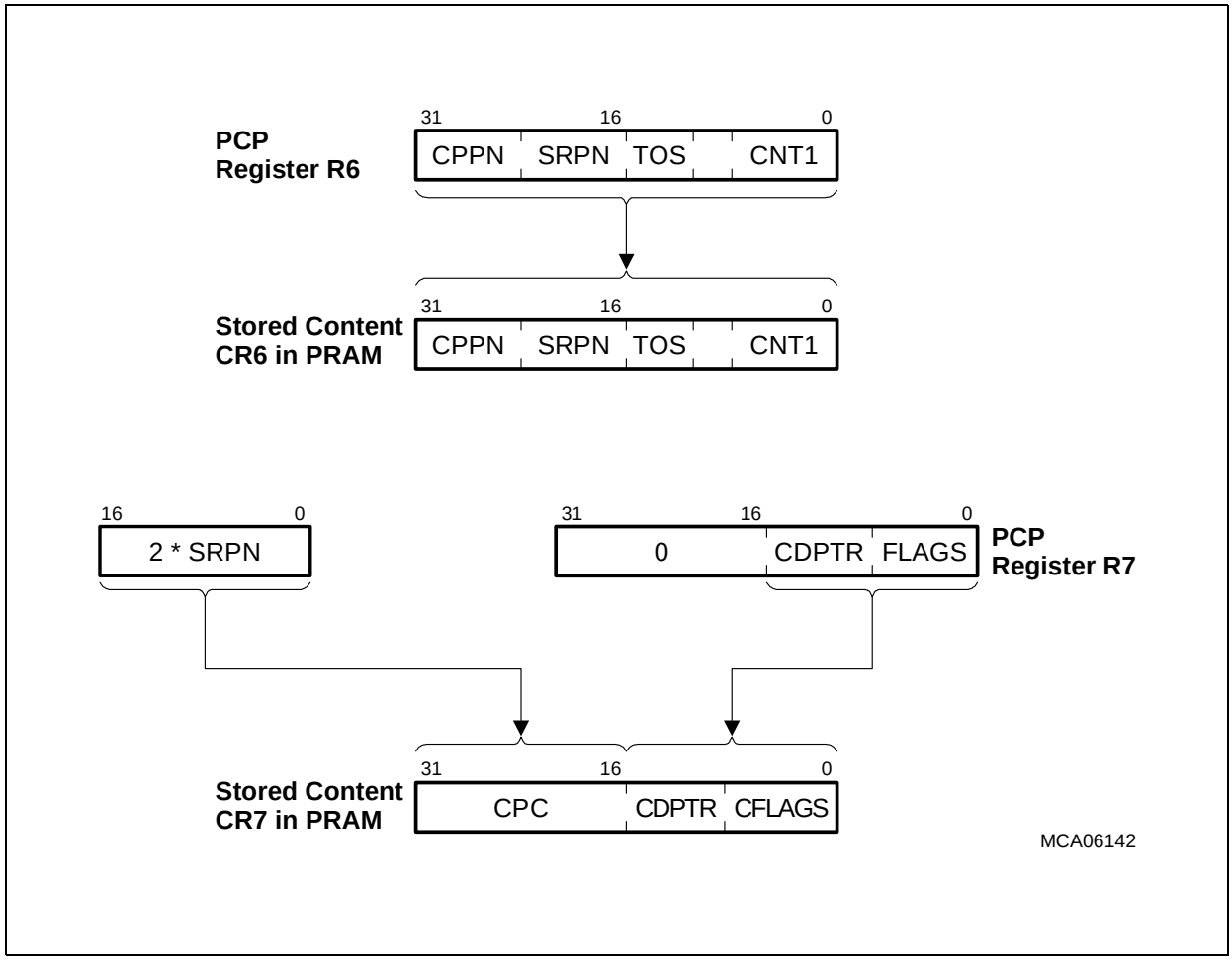


Figure 10-8 Context Save: Channel Exit in Channel Restart Mode

Channel Suspend

Figure 10-9 illustrates the operation of a context save for a channel that is being suspended. This is the same as for Channel Resume mode except that an interrupt request is created to allow the channel to be restarted at a later time. This restore operation utilizes one of three specially extended SRNs (see **Page 10-77**) to store the interrupt request. The information stored as part of the interrupt request is the channel number (SRPN), and the operating priority (CPPN) with which the channel was operating prior to being suspended. This operation in conjunction with the suspended channel restore operation shown in **Figure 10-6** allows the temporary suspension of a channel in favor of a higher-priority channel.

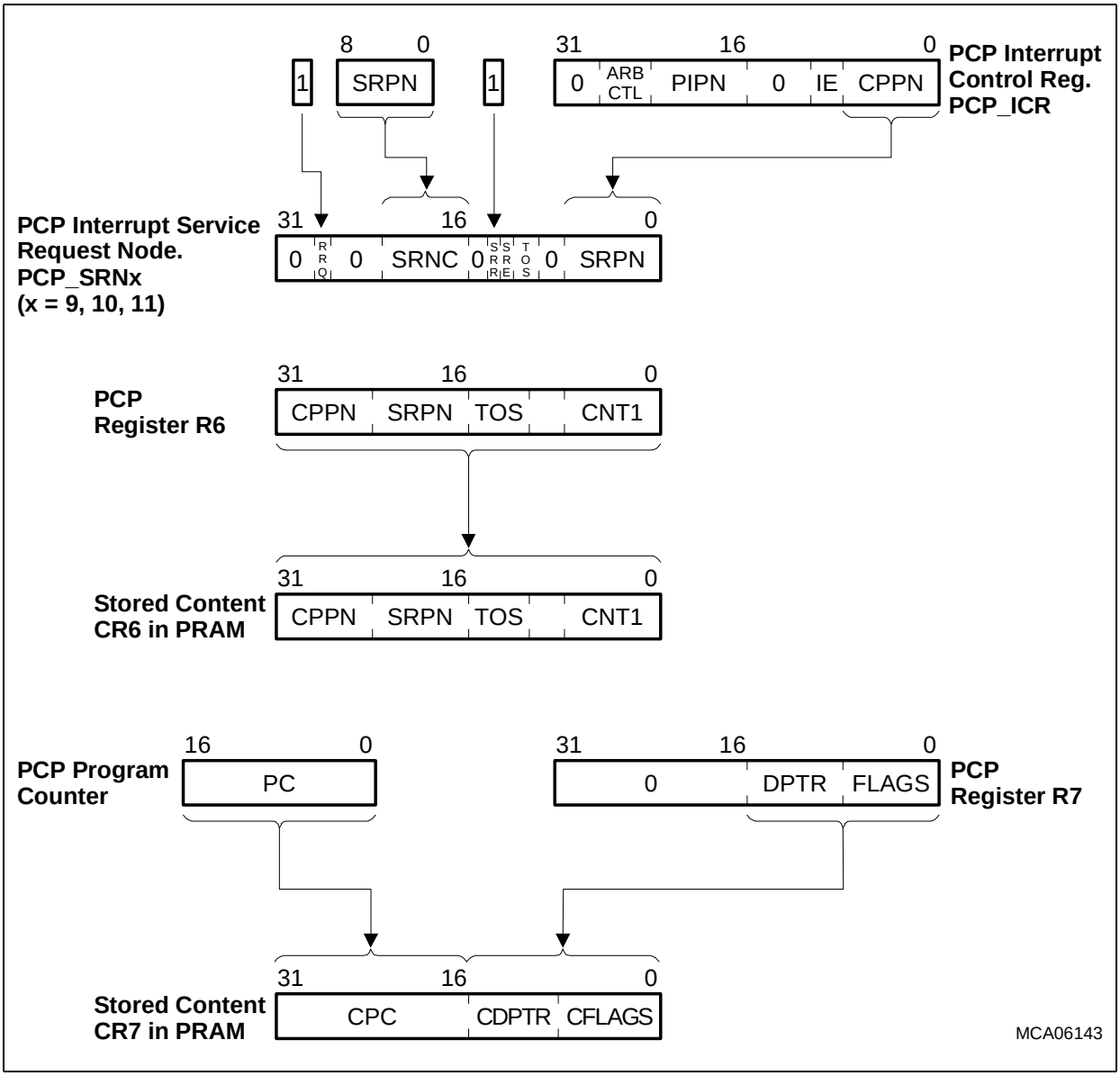


Figure 10-9 Context Save: Channel Suspend

10.4.2.5 Initialization of the Contexts

The programmer is responsible for configuring each channel program's context before commencing operation. Because this must be done by writing to the PCP across the FPI Bus, it is important to understand exactly where each channel program's context is from the FPI Bus perspective (see **Page 10-52** for details).

10.4.2.6 Context Save Optimization

The PCP has an optimized context-switching strategy consisting of optimization of both context load and store. During a context load in which the channel that is starting is also the last channel that the PCP was running then the PCP GPRs already contain the values appropriate to the channel. In this case there is no need to reload the context (i.e. the PCP can immediately continue operation at the appropriate point in the code without having to perform a context load). During a Context Store (i.e. the PCP exits a channel as a result of EXIT or DEBUG instructions or exits in response to a higher-priority channel interrupt), only those registers that have been updated (i.e. written to) since the context was loaded are saved to the CSA.

10.4.3 Channel Programs

The PCP CMEM is used to store the instruction sequences, the channel programs, for each of the PCP channels. The individual channel programs for the individual PCP service requests can usually be viewed as independent and separate programs. There is no background program defined and running for the PCP in TC1797 as there would be with traditional processors.

When the PCP receives a service request for a specific channel program, it needs to determine exactly which channel program to activate and where to start its execution. To accommodate different application needs, the PCP architecture allows the selection of two different entry methods into the channel programs:

- Channel Restart Mode
- Channel Resume Mode

Channel Restart Mode forces the PCP to begin each channel program from a known fixed point in the CMEM that is related to the interrupt number. At the entry point related to the interrupt number in question, there will typically be a jump instruction which vectors the PCP to the main body of the channel program. This is identical to the traditional interrupt vector jump table. In Channel Restart Mode, channel code execution will always start at the same address in the interrupt entry table each time the channel is requested.

Channel Resume Mode allows the PCP to begin execution at the PC address restored as part of the channel program context. This mode allows code to be contiguous and start at any arbitrary address. It also allows for the implementation of interrupt-driven state machines, and even the sharing of code across multiple programs with different context.

The selection of one of the two modes is a global PCP setting, that is, it applies to all channels. Selection is made via the PCP_CS.RCB bit in the PCP configuration register PCP_CS (see **Page 10-61**).

10.4.3.1 Channel Restart Mode

Channel Restart Mode is selected with PCP_CS.RCB = 1. In this mode, the PCP views the CMEM as being partitioned into an interrupt entry table at the beginning of the CMEM, and a general code storage area above this table.

The interrupt entry table consists of two instruction slots (2 × 16-bit) for each channel. When a PCP service request is received, the PCP calculates the start PC for the requested channel by a simple equation based on the SRPN of that request (PC = 2 × SRPN). It then executes the instruction found on that address. If more than two instructions are required for the operation of the channel program, then one of the instructions within the interrupt entry table must be a jump to the remainder of the channel's code. The PCP executes the channel's code until an exit condition or higher-priority interrupt is detected.

It is recommended that all EXIT instructions for all channels should use the EP = 0 setting when the PCP is operated in Channel Restart Mode (see **Page 10-96**).

Note that when Channel Restart Mode is in use a Channel Entry Table must be provided with a valid entry for every channel being used. **Figure 10-10** shows an example of CMEM organization when Channel Restart Mode has been selected. Failure to provide a valid entry for all channels that are in use will lead to invalid PCP operation.

10.4.3.2 Channel Resume Mode

Channel Resume Mode is selected with PCP_CS.RCB = 0. In this mode, the user can arbitrarily determine the address at which the channel program will be started the next time it is invoked. For this purpose, the PC is saved and restored as part of the context of a PCP channel.

Additional flexibility is available when Channel Resume Mode is globally selected by configuring each EXIT instruction to determine the channel start address to be used on the next invocation of a channel (see **Page 10-96**). When the EP = 0 setting is used, the PC value saved in the channel's context (saved in CPC) is the address of the appropriate location in the channel entry table. This forces the channel to start at the appropriate location in the interrupt entry table at next invocation. When the EP = 1 setting is used, the PC value saved in the channel's context is the address of the instruction immediately following the EXIT instruction. The use of the EP = x setting with the EXIT instruction allows the mixture of channels that use a Channel Restart strategy with others using a Channel Resume strategy, and also allows individual channels to use either strategy as appropriate on different invocations.

Note: A valid entry within a Channel Entry Table must be provided for every channel that uses an EXIT instruction with the EP = 0 setting when Channel Resume Mode has been selected. Failure to provide a valid entry for such channels will lead to invalid PCP operation.

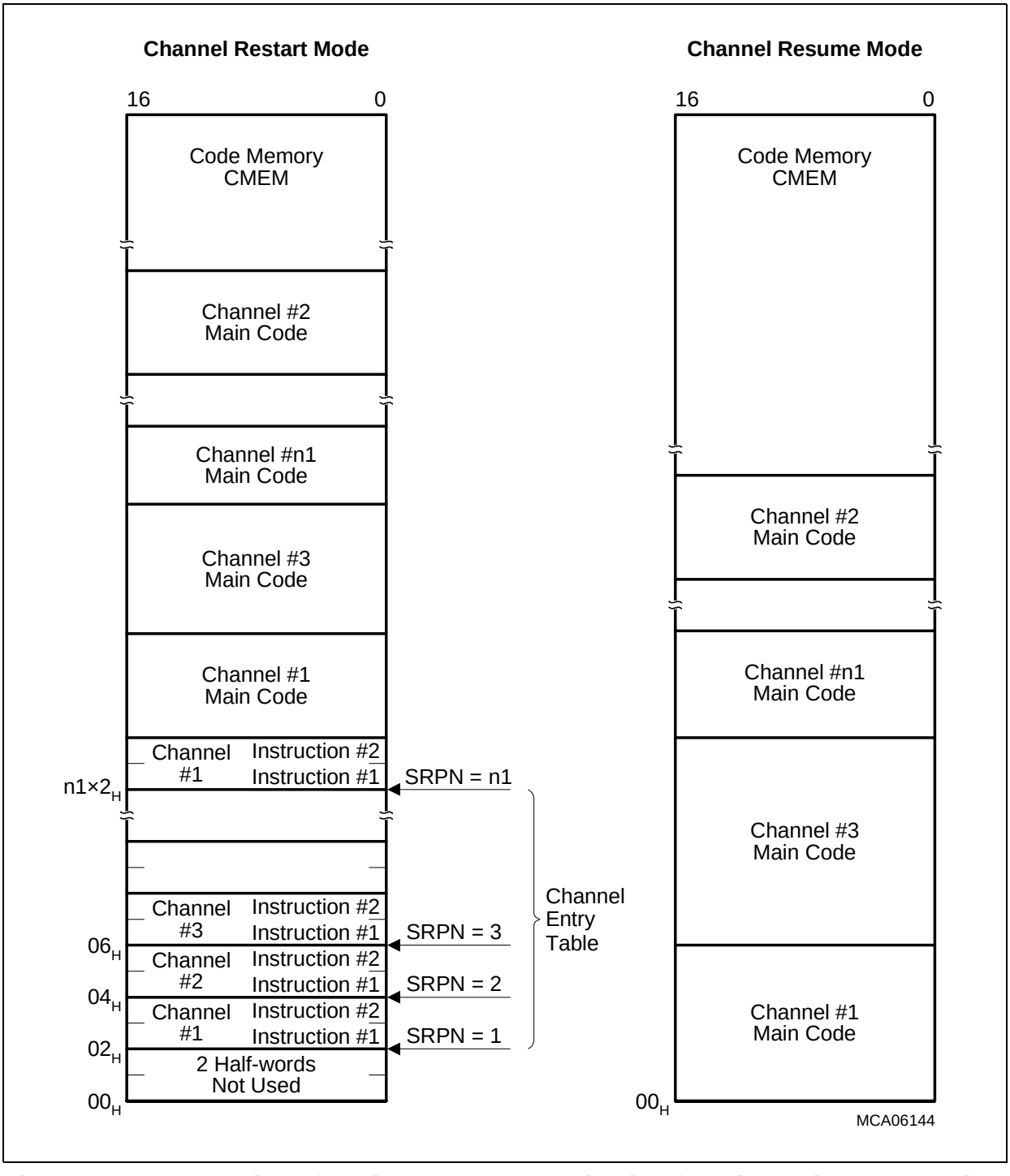


Figure 10-10 Examples of Code Memory Organization for Channel Restart and Channel Resume Modes

Note: The CMEM address offsets in the above figure are shown as PCP instruction (half-word) offsets. To obtain FPI address offsets (byte offset) multiply each offset by two.

10.5 PCP Operation

This section describes how to initialize the PCP, how to invoke a channel program, and the general operation of the PCP.

10.5.1 PCP Initialization

The PCP is placed in a quiescent state when the TC1797 is first powered-on or reset. Before a channel program can be enabled, the PCP as a whole must be initialized by some other FPI Bus master, typically the CPU. Initialization steps include:

- Configure global PCP registers.
  - Initialize PCP Control and Status Register (with PCP_CS.EN = 0).
  - Configure interrupt system via PCP_ICR.
- Load channel programs into the CMEM.
- Load initial context (if/as required) of channel programs in PRAM (R0 - R7 for Maximum context, R4 - R7 for Small Context, R6 - R7 for Minimum Context). Only those registers in each channel whose initial content is required on first invocation of the channel need to be loaded. This may need to include the initial PC, depending on the value of PCP_CS.RCB.
- Clear R7 in the context for unused channels.
- Enable PCP operation PCP_CS.EN = 1.

Now, the PCP is able to begin accepting interrupts and executing channel programs.

10.5.2 Channel Invocation and Context Restore Operation

A channel program is started when one or other of the following conditions occurs:

- The current round of PCP interrupt arbitration results in a winning interrupt number (SRPN) and the PCP is currently quiescent (has exited the previous channel and stored the context for that channel).
- The current round of PCP interrupt arbitration results in a winning interrupt number (SRPN) that has to be greater than the current channel priority (SRPN > CPPN), if suitable Service Request Node space is available in the PSRN to store a suspended interrupt request and the current channel allows interrupts (R7.IEN = 1). See also **Page 10-34**).

When this happens the winning SRPN becomes the current interrupt, and a context restore operation occurs before the new channel program can begin operation, as follows:

- The context of the channel (= winning SRPN) is restored from PRAM into the GPRs from the appropriate address within the CSA. Depending on the value of PCP_CS.CS, a Full, Small, or Minimum Context restore is performed.
- The new priority level of the PCP is taken from R6.CPPN field and is written to PCP_ICR.CPPN. This value can be useful during debugging, as the CPPN of the currently executing or last-executed channel program can be read from PCP_ICR.

After the channel program starts, the value of R6 may be changed without altering the value of the effective CPPN, because updates to the value of R6.CPPN have no effect until the next invocation of the channel program.

- If the R7.CEN bit is clear (0), then an error has occurred because a disabled channel program has been invoked, the PCP_ES.DCR bit is set to flag the error, and the channel program performs an error exit (see **Page 10-30**).
- If the R7.CEN bit is set (1), then code execution begins at the value of the restored PC or at the address of the interrupt routine in the Channel Entry Table, depending on the value of PCP_CS.RCB.

Special care needs to be taken regarding the number of clock cycles required to switch context when the last state of the PCP before channel exit was execution of channel “N” (SRPN = “N”), and the current service request is also SRPN = “N”. In this case, the PCP Processor Core should not load the context (as the PCP GPRs already contain the correct content), but continue operation from the current point and state (noting that the PC value should be set to the appropriate channel entry point if PCP_CS.RCB = 1).

10.5.3 Channel Exit and Context Save Operation

The context of a channel program must be saved when it terminates. Three events can cause the termination of a channel program:

- Execution of the EXIT instruction (normal termination)
- Occurrence of an error
- Execution of the DEBUG instruction (channel termination is optional). The DEBUG instruction must only be used in DEBUG mode; otherwise an “Illegal Operation” (IOP) error will be generated

These channel termination possibilities are described in the next sections.

10.5.3.1 Normal Exit

Under normal circumstances, a channel program finishes by executing an EXIT instruction. This instruction has several setting fields that allow the user to specify a number of optional actions to be performed during the channel exit sequence (see **Page 10-96**). These optional actions are:

- Decrement counter CNT1
- Set the start PC for the next channel invocation to the next instruction address (Channel Resume) or to the channel entry address (Channel Restart)
- Disable further invocations of this channel
- Generate an interrupt request to the CPU or to the PCP itself

When the EXIT instruction is executed, the following sequence occurs:

- If EC = 1 is specified Counter R6.CNT1 is decremented and the CN1Z flag is updated.

- If ST = 1 is specified bit R7.CEN (Channel Enable) is cleared (i.e. the channel is disabled).
- If EP = 0 is specified **or** PCP_CS.RCB = 1 (Channel Restart Mode has been selected), the PCP program counter to be saved to context location CR7.PC is set to the appropriate channel entry table address. If EP = 1 is specified **and** PCP_CS.RCB = 1 (Channel Resume Mode has been selected), the PCP program counter to be saved to context location CR7.PC is set to the address of the instruction immediately following the EXIT instruction.
- If INT = 1 is specified **and** the specified condition cc_B is True, then an interrupt request is raised according to the SRPN value held in R6.SRPN. The interrupt is asserted via one of the PCP_SRCx registers, where x is determined by the combination of the value of R6.TOS and the list of free entries. This allows the conditional creation of a service request to the CPU or PCP with the SRPN value indicated in register R6.SRPN.
- The channel program's context (including all register modifications caused within this EXIT sequence) is saved to the appropriate region in the PRAM Context Save Area. Depending on the value of PCP_CS.CS, either a Full, Small, or Minimum Context save is used.

Special care needs to be taken to optimize the number of clock cycles required to perform a Context Save. During a Context Save, the PCP Processor Core needs only to save those registers that have been written since the last Context Restore was performed.

Note: Particular attention must be paid to the values of R6 and R7 prior to execution of the EXIT instruction. When posting an interrupt request, the user must ensure that R6.SRPN and R6.TOS contain the correct values to generate the required interrupt request. When using the Outer Loop Counter (CNT1), the user must ensure that the value in R6.CNT1 will provide the required function. When using interrupt priority management, the user must ensure that R6.CPPN contains the interrupt priority with which the channel is to run on next invocation. If the channel is to be subsequently re-invoked, the user must ensure that the Channel Enable Bit (R7.CEN) is set.

10.5.3.2 Error Condition Channel Exit

PCP error conditions can occur for a variety of reasons (e.g. an invalid operation code was executed by a channel program, or an FPI Bus error occurred). When an error condition occurs, the PCP Error Status Register (PCP_ES) is updated to reflect the error, and the channel program is aborted. The error exit sequence is as follows:

- The channel enable bit R7.CEN is cleared. This means the channel program will be unable to restart until another FPI Bus master has re-configured the channel program's stored context to set CR7.CEN to 1 again.

- The PC of the instruction that was executing when the error occurred is stored in PCP_ES.EPC.
- The number of the channel program that was executing when the error occurred is stored in PCP_ES.EPN.
- The error type is set in the appropriate field of register PCP_ES.
- The context is saved back to the PRAM CSA. Depending on the chosen context size (PCP_ES.CS) a Full, Small, or Minimum Context save is performed.
- If the error condition was not due to an FPI Bus error or a DEBUG instruction, then an interrupt request to the CPU is generated with the priority number stored in register PCP_CS.ESR.

The repetitive posting of PCP error interrupts will not cause an overwhelming number of interrupts to the CPU. In this situation, the PCP's CPU service request queue (see **Page 10-34**) will quickly fill, and force the PCP to stall until the CPU can resolve the situation.

Note: An error condition (other than an FPI Bus error) will result in an interrupt being sent to the CPU. The interrupt routine that responds to this interrupt must be capable of dealing with the cause as recorded in PCP_ES, and it must be able to restore the channel program to operation. The minimum required to restart the channel program is to set the context value of CR7.CEN = 1.

10.5.3.3 Debug Exit

If the DEBUG instruction is programmed to stop the channel program execution (SDB = 1 has been specified), the PCP performs an exit sequence that is very similar to the error exit sequence, with the exception that no interrupt request to the CPU is generated. This sequence is:

- If RTA = 0, then the channel enable bit R7.CEN is cleared. This means the channel program will be unable to restart until another FPI Bus master has re-configured the channel program's stored context to set CR7.CEN to 1 again. Otherwise, the R7.CEN bit remains unchanged, and the PC is decremented (such that it points to the DEBUG instruction)
- If EDA = 1, a break-point event is generated
- If DAC = 1, then the PCP_CS.EN bit is cleared. This means that the PCP will not execute any further channel programs until the PCP_CS.EN bit is set by another FPI Bus master
- The address of the DEBUG instruction (i.e. the current PC) is stored in register PCP_ES.PC
- The current channel number is stored in register PCP_ES.PN

The execution of the current channel program is stopped at the point of the DEBUG instruction. This instruction only disables the current channel; the PCP will continue to operate, accepting service requests for other channels as they arise.

Note: The DEBUG instruction must be only used in DEBUG mode; otherwise an "Illegal Operation" (IOP) error will be generated.

10.6 PCP Interrupt Operation

The PICU and the PSRNs (PCP_SRC[11:0]) are similar to the CPU's ICU and all other SRNs in the system. They do, however, have some special characteristics, which are described in the following sections. **Figure 10-11** shows an overview of the PCP interrupt scheme.

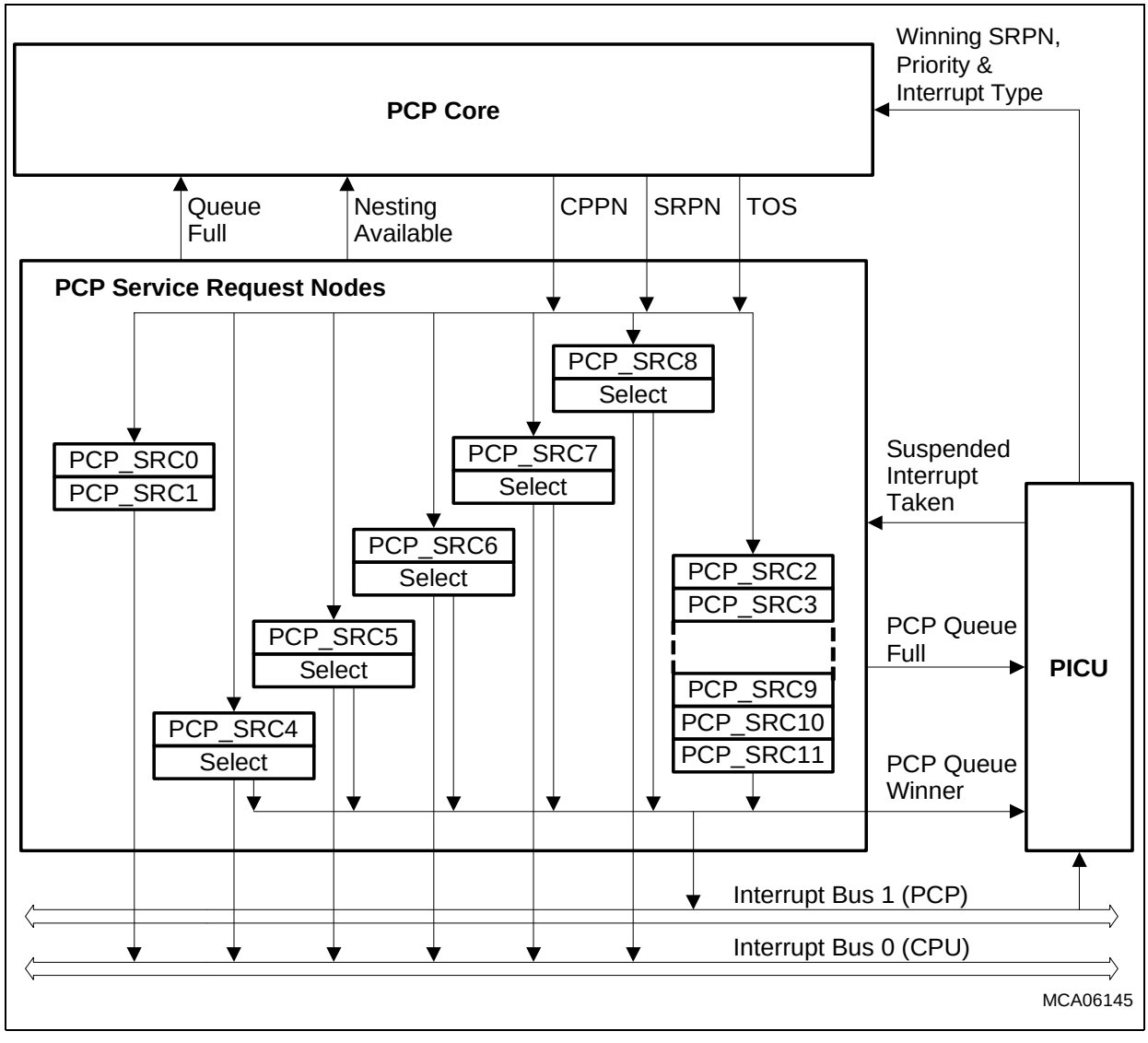


Figure 10-11 PCP Interrupt Block Diagram

10.6.1 Issuing Service Requests to CPU or PCP

The PCP may use one of two mechanisms to raise an interrupt request to the CPU or itself. The first, and most inefficient, method for a PCP channel program is to issue service requests by performing an FPI Bus write operation to an external service request node (SRN). Alternately, the PCP can raise a service request using one of its own internal SRNs. An interrupt can only be generated by the PCP via an internal SRN when executing an EXIT instruction, or when an error condition occurs. In the following descriptions, PCP service requests triggered through an EXIT instruction or the occurrence of an error are called “implicit” PCP service requests to distinguish them from the “explicit” way of generating a service request through an FPI Bus write to a service request node external to the PCP.

10.6.2 PCP Interrupt Control Unit

The PICU operates in a similar manner to the ICU of the CPU. The PICU manages the PCP service request arbitration bus, and handles the communication of service requests and priority numbers to and from the PCP kernel. The PCP_ICR register is provided to control and monitor the arbitration process.

When one or more service requests to the PCP are activated, the PICU performs an arbitration round to determine the request with the highest priority. It then places the priority number of this “winning” service request into the PIPN field of register PCP_ICR, and generates a service request to the PCP kernel.

If the PCP kernel is currently busy processing a channel program, the new request is left pending until the current channel program has finished.

When the PCP kernel is ready to accept a new service request, it calculates the context start address from the Pending Interrupt Priority Number, PIPN, stored in register ICR, and begins with the context restore. It notifies the PICU of the acceptance of this request, and in turn the PICU acknowledges the winner of the last arbitration round. This service request node then resets its Service Request Flag, SRR.

There is one special condition in which the PICU operates differently from the way that the CPU Interrupt Control Unit operates. This special operation is described on **Page 10-37**.

The PCP interrupt arbitration can be adapted to the application’s needs and characteristics through controls in register PCP_ICR. Bit field PCP_ICR.PARBCYC controls the number of arbitration cycles per arbitration round (one through four cycles), while bit PCP_ICR.PONECYC controls whether one arbitration cycle equals one or two system (FPI Bus) clock cycles.

10.6.3 PCP Service Request Nodes

The PCP contains twelve service request nodes, including twelve service request control registers, PCP_SRC0 to PCP_SRC11, which are provided for implicit PCP service

requests. The service request control registers differ from standard SRC registers in that they are fully controlled by the PCP kernel; they are read-only registers during PCP operation. The user cannot generate interrupts by writing to them.

The twelve service request nodes are split into four groups.

- The first group, containing registers PCP_SRC0 and PCP_SRC1, handles implicit PCP service requests targeted to the CPU. The Type-of-Service control fields, TOS, of these registers are hard-wired to 00_B, directing the requests to the CPU.
- The second group, registers PCP_SRC2 and PCP_SRC3, handles the service requests targeted to the PCP itself. The respective TOS field of these registers are hard-wired to 01_B, directing the requests to the PCP.
- The third group, containing registers PCP_SRC4 to PCP_SRC8, have programmable TOS fields which allow these registers to be assigned (at configuration time) to any of the available interrupt buses.
- The fourth group, containing registers PCP_SRC9, PCP_SRC10 and PCP_SRC11, are an extended version of a standard Service Request Node. These handle service requests targeted to the PCP itself, including service requests representing a suspended interrupt. The respective TOS field of these registers are hard-wired to 01_B, directing the requests to the PCP.

The service request enable bits, SRE, of the PCP_SRCx registers are hard-wired to 1, meaning these service requests are always enabled.

Note: The number of interrupt buses is device-dependent. Programming a PCP_SRCx register (x = 4 to 8) with a TOS value representing a non-available interrupt bus (10_B or 11_B in the TC1797) will disable Service Request Node x.

The actual service request flag and the service request priority number of the PCP_SRCx registers are updated by the PCP when it generates an implicit service request. The way this is performed is described in the following section.

The service request nodes in each of the groups described above are implemented as queues with the appropriate number of entries. When the PCP generates an implicit service request, it places the request into the next available free entry of the appropriate queue rather than writing it into a specific register. Queue management logic automatically ensures proper handling of the queue. If all entries of a queue are filled with pending service requests, the queue management reports this condition to the PCP kernel via a “queue full” signal.

In the following descriptions, the terms “CPU Queue” and “PCP Queue” are used to refer to the queues in the two groups of PCP service request nodes.

10.6.4 Issuing PCP Service Requests

The PCP can issue implicit service requests on the execution of an EXIT instruction, when suspending a channel, or when an error occurs during a channel program execution. While the service request generation for the EXIT instruction is optional, a

service request is always generated when a channel is suspended or an error occurs. Further differences between these three mechanisms are detailed in the following sections.

10.6.4.1 Service Request on EXIT Instruction

An implicit PCP service request is issued when the INT field of the EXIT instruction is set to 1 and the specified condition code, cc_B, of this instruction is true. Such a service request can be issued to any of the available interrupt buses, depending on the programmed value in the TOS field of register R6. The PCP examines the TOS field in register R6 and issues a service request to the appropriate queue of the service request nodes. Along with this request, it passes the service request priority number stored in the SRPN field of register R6 to the queue. If the queue has a free entry left, the service request flag, SRR, of the associated service request register, PCP_SRCx, will be set, and the service request priority number will be written to the SRPN field of the SRC register. Please see **Page 10-37** for the case where there is no free entry in the queue.

Because the desired service request is programmed through the TOS and SRPN fields in register R6, each channel program can issue its individual service request. Note that this register needs to be programmed properly if a service request is to be generated by the EXIT instruction.

10.6.4.2 Service Request on Suspension of Interrupt

An implicit PCP service request is issued when the PCP suspends execution of the ongoing channel program in favor of a service request with a higher priority. Such a service request is always issued to the PCP's own interrupt bus and is stored in one of the three extended Service Request Nodes (PCP_SCR9, PCP_SRC10, PCP_SRC11). Along with this request, it passes the current channel operating priority (CPPN) as an SRPN and also the channel number (the original SRPN). The service request flag, SRR, and the Restart Request flag, RRQ, of the associated service request register, PCP_SRCx, will be set, the Operating Priority will be written to the SRPN field, and the channel number will be written to the SRNC field of the SRC register.

Use of the Operating Priority as the SRPN for resumption of the channel program ensures that during subsequent arbitration rounds the PCP will resume execution of the suspended channel program at the appropriate time.

The PCP treats an interrupt request with the RRQ bit set in a special fashion. In this case the PCP clears the interrupt request bit in the appropriate internal Service Request Node but does not issue an interrupt acknowledge to any external nodes. This prevents the unwanted clearing of external service requests with an SRPN that matches the priority of a suspended channel.

Note: The PCP will only suspend channel operation when there are two or more free SRNs with the appropriate TOS value for the PCP, and one of the free SRNs is an

Extended Service Request Node. This allows for the posting of an interrupt request to the PCP on exit from the new channel program.

10.6.4.3 Service Request on Error

While a service request triggered through an EXIT instruction is optional and can be issued either to the CPU or to the PCP itself, a service request due to an error condition will always be automatically issued and will always be directed to the CPU. The PCP issues a service request to the CPU queue of the service request nodes. Along with this request, it passes the SRPN stored in the ESR field of register PCP_CS to the queue. If the queue has a free entry left, the SRR flag of the associated service request register, PCP_SRCx, will be set, and the SRPN will be written to the SRPN field of the SRC register. See next section for the case where there is no free entry in the queue (queue full).

Due to the fact that the priority number is stored in the global control register PCP_CS, all channel programs share the same service request routine in case of an error. The exact cause of the error and the channel number of the program that was executed when the error occurred can be determined through examination of the contents of the Error/Debug Status Register, PCP_ES.

10.6.4.4 Queue Full Operation

Queuing the implicit service requests typically allows the PCP to continue with the next service request without stalling. The depth of the queue and the number of channel programs using them determines the stall rate. Depending on the selected service provider (via R6.TOS in case of an EXIT interrupt or always to the CPU in case of an error interrupt), the request is routed to a free entry in the appropriate queue.

If no free entries are available in a queue at the time the PCP wants to post a request to that queue, the PCP is forced to stall until an entry becomes clear. This ensures that the PCP does not lose any interrupts. An entry in a queue becomes free when its SRR flag, is cleared through an acknowledge from the PICU (that is, the CPU or PCP, as appropriate, has started to service this request).

One special case needs to be resolved for the PCP-related queue through special operation of the PICU. Consider the case in which the PCP queue is full, meaning registers PCP_SRC2, PCP_SRC3 and PCP_SRC9 to PCP_SRC11 are already loaded with pending service requests to the PCP. If the PCP kernel now needed to post an additional service request into that queue, a deadlock situation would be generated: The PCP would stall, since there is not a free entry in the PCP queue in which to place the request. In turn, as the PCP is stalled, it cannot accept new service requests and so the PCP service request queue cannot be emptied. This would result in a deadlock of the PCP.

To avoid such a deadlock, the PICU performs a special arbitration round as soon as the PCP queue becomes full. In this arbitration round, only the service request nodes assigned to the PCP queue are allowed to participate; all service requests from nodes external to the PCP are excluded, regardless of whether their priorities are higher or lower than those of the PCP queue. In this way, it is guaranteed that one entry in the PCP queue gets serviced, freeing one slot in the queue.

The PCP programmer needs to carefully consider this special operation. It ensures that deadlocks are avoided, but it implies that if too many PCP channel programs post service requests to the PCP (self-interrupt), the PCP will have to service these rather than outside interrupt sources. Depending on the priority given to these requests, this could undermine an otherwise appropriate use of the interrupt priority scheme. It is recommended that the system be designed such that in most cases, high-priority numbers can be assigned to these self-interrupts, so that they can win normal arbitration rounds, avoiding the situation where the PCP queue becomes full.

Note: If the CPU queue is full, the PCP can continue to operate until it needs to post another service request to the CPU queue.

10.7 PRAM Protection

10.7.1 Protection of PRAM against Internally Generated PRAM Writes

10.7.1.1 Context Save Region Protection

The Context Save Region can be protected by the PRAM Partitioning Scheme (see **“PCP Control and Status Register, PCP_CS” on Page 10-61**). This protection is enabled via PCP_CS.PPE and the size of the context area being protected is defined by PCP_CS.PPS. When enabled the context region is protected from writes by any Channel Program (i.e. regardless of whether Protected or Unprotected).

Note: This scheme also limits the number of Channel Programs that can be invoked.

Note: FPI PRAM write accesses to PRAM are unaffected by this protection.

10.8 FPI Interface

The PCP operates both as an FPI Master and an FPI Slave.

10.8.1 Operation as an FPI Master

The PCP generates FPI read and write transactions in response to execution of PCP FPI instructions. The PCP can generate Byte, Half-word and Word single transactions and

bursts of length two, four and eight. All FPI transactions are generated in Supervisor Mode.

10.8.2 Operation as an FPI Slave

The PCP is visible to FPI Masters as a 256 Kbyte R/W block of memory on the System Bus. The PCP must be accessed only with word (32 bit) accesses. Accessing an unassigned address (i.e. an address outside the range of Control & Status Registers, PRAM and CMEM) will generate an FPI bus error. All PCP locations can be read in either User or Supervisor Modes. All writes must be performed in Supervisor Mode. Attempting to write to any PCP location in User Mode will generate an FPI error. Some Control and Status Registers are ENDINIT protected.

10.9 PCP Error Handling

The PCP contains a number of fail-safe mechanisms to ensure that error conditions are handled gracefully and predictably. In addition to providing an extra level of system robustness suitable for high integrity and safety-critical systems, these mechanisms can often ease the task of finding programming errors during the development process. Whenever an error is detected, the channel program that was executing exits and the PCP_ES register is updated with information to allow determination of the error that occurred, the instruction address, and the channel program that was executing when the error occurred (see **Page 10-31**).

10.9.1 PRAM Protection Violation

The default configuration of the PCP allows the PCP to use PRAM as a single area. While this default configuration allows complete flexibility regarding the use of PRAM, this flexibility also introduces the possibility of invalid PCP operation as a result of the following issues:

- Any channel program is allowed to write to any PRAM location (including any location in the CSA). This means that a channel program may be inadvertently programmed to corrupt the context save region or PRAM storage belonging to another channel, causing invalid operation of the corrupted channel when it next executes.
- Generation of an interrupt request to the PCP with a priority number that would cause loading of a context from outside the CSA will cause the spurious execution of a channel program with an invalid context loaded from outside the CSA.

10.9.1.1 Enforced PRAM Partitioning

The lowest of the PRAM areas is the CSA (see **Page 10-15**) which is used for storing context information for each active channel while the channel program is not actually executing.

To avoid spurious PCP operation as a result of programming errors, the PCP can be optionally configured via the global PCP control and status register (PCP_CS) to enforce strict partitioning of PRAM between the CSA and the remainder of PRAM. PRAM partitioning is selected by programming PCP_CS.PPE = 1 and the size of the CSA in use is selected via the PCP_CS.PPS bit field (see **Page 10-61**). When PRAM partitioning has been enabled, a PCP Error will be generated on either one of the following events:

- A channel program executes a PRAM write instruction with a target area within the CSA. This prevents a channel from corrupting the context save region of any other channel.
- An incoming interrupt request causes the PCP to attempt to load a context from outside the CSA. This prevents the PCP from running an invalid channel program as a result of an invalid interrupt request.

Note: Enabling PRAM partitioning (PCP_CS.PPE = 1) with a CSA size of zero (PCP_CS.PPS = 0) is an invalid setting and will cause a PCP error event whenever any interrupt request is received by the PCP.

10.9.2 Channel Watchdog

The Channel Watchdog is a PCP internal watchdog that optionally allows the user to ensure that the PCP will not become locked into executing a single channel due to an endless loop or unexpected software sequence. As each channel executes, the PCP maintains an internal count of the number of instructions that have been read from CMEM since the channel started. If the watchdog function is enabled (by programming PCP_CS.CWE = 1) and the internal instruction fetch counter reaches the threshold programmed by the user (programmed via PCP_CS.CWT), a PCP Error is generated. The threshold setting (PCP_CS.CWT) is global to all channels. From this it follows that the threshold must be selected to be greater than the maximum number of instructions that can be fetched by any channel program, taking all channels into consideration. It should be noted that the instruction width of the PCP is 16 bits and that therefore execution of an instruction that is encoded into 32 bits (e.g. LDL.IL) will generate two CMEM instruction reads. That will therefore cause the internal watchdog counter to be incremented twice.

Note: Enabling the Channel Watchdog function (PCP_CS.CWE = 1) with a threshold of zero (PCP_CS.CWT = 0) is an invalid setting and will cause a PCP error event whenever any interrupt request is received by the PCP.

10.9.3 Invalid Opcode

The PCP includes the Invalid Opcode mechanism to check that each instruction fetched from CMEM is a legal instruction. If the PCP attempts to execute an illegal instruction, then a PCP error is generated.

Note: The DEBUG instruction must be only used in DEBUG mode otherwise it will be considered to be an illegal operation and will generate an IOP error.

10.9.4 Instruction Address Error

An Instruction Address Error is generated if the PCP attempts to execute an instruction from an illegal address. An address is considered to be illegal if:

- The address is outside the available CMEM area (see **Page 10-138** for the CMEM size implemented in this derivative)

and/or

- The specified address could not be contained in the 16-bit PC (i.e. an address calculation yielded a 16-bit unsigned overflow).

The second of these cases can result from an address calculation either from the execution of a PC relative jump instruction (either a JC, JC.I, or JL instruction), or the PC being incremented following execution of the previous instruction.

10.10 Software In-System Test Support

The PCP protects against memory integrity errors by Parity protection of the PCP memories. This has the unfortunate side-effect of requiring memory blocks wider than the normal data access path to the memory. The additional Parity storage bits are not easily accessible via the existing data paths, causing significant problems where SIST based testing of the memories is required. In order to address this problem the PCP includes improved SIST support, allowing all PCP memory arrays to be accessedto allow the test and debug of the fault tolerant memory systems.

The mapping of hidden memory Parity bits into the PCP address space is controlled by the setting of bits within the SIST Mode Access Control Register (SMACON). The existence of the SMACON register is architecturally defined. However, the fields within SMACON and the effect of the fields on the memory map of the PCP are implementation specific.

Hidden Parity bits are mapped into the CMEM and PRAM areas by setting bits in the SMACON register. The SMACON register is a SFR which can only be written in supervisor mode and is endinit protected. The definition of the SMACON register for the PCP is shown on **Page 10-73**.

The control fields within the SMACON register allow individual control of the local memories. Each memory may be mapped to operate in a number of different modes.

Normal Operation, No Mapping, error detection enabled

No mapping of the Parity bits is performed and normal operation is possible. Error detection is active.

Data Array Mapping, no error detection

No mapping of the Parity bits is performed. Writes to the memory will not affect the Parity bits. Error detection for the memory is disabled. Normal operation (with the exception that Parity protection is not operational) is possible.

Check Array Mapping, no error detection

The Parity bit array (only) of the memory is made visible in the address map (i.e. the memory data array cannot be accessed). Writes to the memory will not affect the data bits. Error detection for the memory is disabled. The PCP must be disabled before, and remain disabled while, this mode is selected for any of the PCP memories.

When this mode is selected the memory Parity bit array replaces the memory data array in the address map. During a read the value of the Parity bit for a location can be

determined from bit 0 of the value read from the location (all other bits read as ‘0’). During a write the Parity bits are updated from bit 0 of the written value. The written value of all bits other than bit 0 is discarded.

Data Array Mapping, error detection enabled.

No mapping of the Parity bits is performed and normal operation is possible.

Note: Unlike TriCore the PCP has no hidden memory arrays (i.e. all PCP memories reside permanently in the system address map). As a result, for the PCP, this mode is identical to “Normal Operation”. This mode is retained for compatibility with TriCore SIST.

10.11 Memory Integrity Error Detection

The PCP includes support for Memory Integrity Error Detection logic to address the increasing occurrence of memory errors in deep sub-micron CMOS technologies. Soft errors are transient errors in state (not permanent) typically caused by alpha particle hits to the die, causing induced charge which flips the state of storage elements. In current process geometries SRAM arrays are most susceptible to soft error events due to their small size and low stored charge. The detection and deterministic recovery from such memory integrity errors is especially important for automotive applications, such as electronic braking systems and engine management.

The PCP includes the following SRAM arrays:

- Code Memory (CMEM)
- Parameter Memory (PRAM)

Each of these memory arrays will be protected from memory integrity errors, with the specific mechanisms described in the following sections.

Note: Before enabling Error Detection on an SRAM (i.e. CMEM or PRAM) the user must ensure that all locations within the SRAM have been initialised. If this is not done then unwanted spurious memory errors can be generated.

10.12 Instruction Set Overview

The following sections present an overview of the instruction set and the available addressing modes of the PCP in the TC1797.

10.12.1 DMA Primitives

Table 10-3 describes the two DMA instructions of the PCP.

Table 10-4 DMA Transfer Instructions

DMA Transfer	BCOPY	Move block of data value from FPI Bus source address location to FPI Bus destination address location. Optionally increment or decrement source and destination pointer registers. Optionally repeat instruction until counter CNT1 reaches 0.
	COPY	Move value from FPI Bus source address location to FPI Bus destination address location. Optionally increment or decrement source and destination pointer registers. Optionally repeat instruction until counter CNT1 reaches 0.

10.12.2 Load and Store

Table 10-4 describes the load and store instructions of the PCP.

Note: If a conditional instruction's condition code is false, the operation will be treated as a "No Operation". Register values will not be changed and the flags will not be updated.

Table 10-5 Load and Store Instructions

Load	LD.F	Load value from FPI Bus address location into register (FPI Bus address = register content)
	LD.I	Load immediate value into register
	LD.IF	Load value from FPI Bus address location into register (FPI Bus address = register content + immediate offset)
	LD.P	Load value from PRAM address location into register (PRAM address = DPTR + register offset)
	LD.PI	Load value from PRAM address location into register (PRAM address = DPTR + immediate offset)
	LDL.IL	Load 16-bit immediate value into lower bits [15:0] of register
	LDL.IU	Load 16-bit immediate value into upper bits [31:16] of register
Store	ST.F	Store register value to FPI Bus address location (FPI Bus address = register content)
	ST.IF	Store register value to FPI Bus address location (FPI Bus address = register content + immediate offset)
	ST.P	Store register value to PRAM address location (PRAM address = DPTR + register offset)
	ST.PI	Store register value to PRAM address location (PRAM address = DPTR + immediate offset)
Move	MOV	Conditionally move register value to register

10.12.3 Arithmetic and Logical Instructions

Arithmetic instructions that are fully register-based execute conditionally depending on the specified Condition Code A (see **Page 10-80**). All other arithmetic instructions such as PRAM (.PI), indirect (.I), and FPI (.F and .IF) execute unconditionally.

Note: If a conditional instruction's condition code is false, the operation will be treated as a "No Operation". Register values will not be changed and the flags will not be updated.

Table 10-6 Arithmetic Instructions

Add	ADD	Add register to register (conditionally)
	ADD.I	Add immediate value to register
	ADD.F	Add content of FPI Bus address location to register (byte, half-word or word)
	ADD.PI	Add content of PRAM address location to register
Subtract	SUB	Subtract register from register (conditionally)
	SUB.I	Subtract immediate value from register
	SUB.F	Subtract content of FPI Bus address location from register (byte, half-word or word)
	SUB.PI	Subtract content of PRAM address location from register
Compare	COMP	Compare register to register (conditionally)
	COMP.I	Compare immediate value to register
	COMP.F	Compare content of FPI Bus address location to register (byte, half-word or word)
	COMP.PI	Compare content of PRAM address location to register
Negate	NEG	Negate register (2's complement, conditionally)

Logical instructions that are fully register-based execute conditionally as determined by the specified Condition Code A. All other logical instructions, such as PRAM (.PI), indirect (.I), and FPI (.F and .IF) execute unconditionally.

Table 10-7 Logical Instructions

Logical And	AND	Register AND register (conditionally)
	AND.F	Content of FPI Bus address location AND register (byte, half-word or word)
	AND.PI	Content of PRAM address location AND register
	MCLR.PI	Clear specified bits within a PRAM location

Table 10-7 Logical Instructions

Logical Or	OR	Register OR register (conditionally)
	OR.F	Content of FPI Bus address location OR register (byte, half-word or word)
	OR.PI	Content of PRAM address location OR register
	MSET.PI	Set specified bits within a PRAM location
Logical Exclusive-Or	XOR	Register XOR register (conditionally)
	XOR.F	Content of FPI Bus address location XOR register (byte, half-word or word)
	XOR.PI	Content of PRAM address location XOR register
Logical Not	NOT	Invert register (1’s complement, conditionally)
Shift	SHL	Shift left register
	SHR	Shift right register
Rotate	RL	Rotate left register
	RR	Rotate right register
Prioritize	PRI	Calculate position of first set bit (1-bit) in register, from left

Note: If a conditional instruction’s condition code is false, the operation will be treated as a “No Operation”. Register values will not be changed and the flags will not be updated.

10.12.4 Bit Manipulation

All bit manipulation instructions except INB are executed unconditionally. If conditional bit handling is required, INB should be used.

Table 10-8 Bit Manipulation Instructions

Set Bit	SET	Set bit in register
	SET.F	Set bit in FPI Bus address location
Clear Bit	CLR	Clear bit in register
	CLR.F	Clear bit in FPI Bus address location
Insert Bit	INB	Insert carry flag into register (position given by content of a register)
	INB.I	Insert carry flag into register (position given by immediate value)
Check Bit	CHKB	Set carry flag depending on value of specified register bit

10.12.5 Flow Control

Table 10-8 describes flow control instructions of the PCP in the TC1797.

Table 10-9 Flow Control Instructions

Jump	JC	Jump conditionally to PC + short immediate offset address
	JC.A	Jump conditionally to immediate absolute address
	JC.I	Jump conditionally to PC + register offset address
	JC.IA	Jump conditionally to register absolute address
	JL	Jump unconditionally to PC + long immediate offset address
Exit Channel	EXIT	Unconditionally exit channel program execution (optionally generate interrupt request and/or inhibit channel)
No Operation	NOP	Low-power No-Operation
Debug	DEBUG	Conditionally generate debug event (optionally stop channel program execution)

10.12.6 Addressing Modes

The PCP needs to address locations in memory in different ways, as determined by the type of memory being accessed and the type of action being performed on that location.

10.12.6.1 FPI Bus Addressing

All FPI Bus accesses from the PCP are indirect to some extent. The main indirect addressing on the FPI Bus uses a 32-bit absolute address located in the GPR indicated in the instruction. This address must be properly aligned for the type of data access – byte, half-word or word. If it is not aligned, the results are undefined.

- Effective Target Address [31:0] = <R[a]>

where a is the number of the register, for instance, R2. Instructions using this address mode are indicated through the ".F" suffix.

For indirect-plus-immediate addressing on the FPI Bus, the 32-bit absolute address located in the GPR indicated in the instruction is added to the immediate 5-bit offset value encoded in the instruction. This address must be properly aligned for the type of data access (byte, half-word or word). If it is not aligned, the results are undefined.

- Effective Target Address [31:0] = <R[a]> + #offset5

where a is the number of the register and #offset5 is a 5-bit immediate offset value. Instructions using this addressing mode are indicated through the ".IF" suffix (only available for load and store, LD.IF and ST.IF).

This addressing mode is particularly useful for managing peripherals, where the peripheral base address is held in R[a], and the offset can index directly into a specific control register.

The BCOPY and COPY instructions use the indirect absolute addressing with predefined PCP registers. Register R4 is used as the source address pointer, while R5 represents the destination address pointer.

- Effective Source Address [31:0] = <R4>
- Effective Destination Address [31:0] = <R5>

Note: All FPI Bus accesses by the PCP are performed in Supervisor mode.

Note: The PCP is not allowed to access its own registers via instructions executed in the PCP.

10.12.6.2 PRAM Addressing

The PRAM is always addressed indirectly by the PCP. The normal address used is the value of the R7.DPTR field (8 bits) concatenated with an immediate 6-bit offset value encoded in the instruction, yielding a 14-bit word address. This enables access to 16 Kwords (64 Kbytes). Because R7.DPTR is part of a channel program's context, a channel program may alter the DPTR value at any time.

- Effective PRAM Address[13:0] = <R7.DPTR> << 6 + #offset6

Instructions using this addressing mode are indicated through the ".PI" suffix.

To provide effective indexing into large tables or stores of data, an alternate form of indirect addressing can also be used on load and store operations to PRAM. The value of the DPTR field (8 bits) is concatenated with the least significant 6 bits of R[a], again yielding a 14-bit word address. The most significant bits [31:6] of R[a] are ignored.

- Effective PRAM Address[13:0] = <R7.DPTR> << 6 + <R[a][5:0]>

Instructions using this addressing mode are indicated through the ".P" suffix (load and store only, LD.P and ST.P).

10.12.6.3 Bit Addressing

Single bits can be addressed in the PCP GPRs or in FPI Bus address locations. A 5-bit value indicates the location of a bit in the register specified in the instruction. This bit location is either given through an immediate value in the instruction or through the lower five bits of a second register (indirect addressing).

- Effective Bit Position[31:0] = #imm5
- Effective Bit Position[31:0] = <R[a][5:0]>

The immediate bit addressing is used by instructions SET and CLR and their variants as well as by INB.I and CHKB. Indirect bit addressing is used by the INB instruction only.

10.12.6.4 Flow Control Destination Addressing

The jump instructions are split into two groups: PC-relative jumps, and jumps to an absolute address.

For PC-relative jumps, the destination address is a positive or negative offset from the PC of the next instruction. The offset is either contained in the lower 16 bits of a register (the upper 16 bits are ignored), or is given as immediate value of the instruction. The immediate values are sign-extended to 16 bits. If the effective jump address is outside the available CMEM area (or the jump address calculation caused an overflow), then a PCP error condition has occurred.

- Effective JUMP Address[15:0] = NextPC + Signed(R[a][15:0]); +/- 32K instructions
- Effective JUMP Address[15:0] = NextPC + Sign-Extend(#offset10); +/- 512 instructions

- Effective JUMP Address[15:0] = NextPC + Sign-Extend(#offset6);
  +/- 32 instructions

The function NextPC indicates the instruction that would be fetched next by the program counter. Instructions using this addressing are JL, JC and JC.I.

For absolute addressing, the actual address in CMEM where program flow is to resume is either an immediate value #imm16 in the CMEM location immediately following the jump instruction, or it is contained in the lower 16 bits of a register. If the value is greater than the PC size implemented, an error condition has occurred.

- Effective JUMP Address[15:0] = #imm16
- Effective JUMP Address[15:0] = <R[a]>

Instructions using these addressing modes are JC.A (immediate absolute address) and JC.IA (indirect absolute address).

10.13 FPI Interface

Any FPI Bus master (on the TC1797's System Peripheral Bus) can access the three distinct PCP address ranges from the FPI Bus side, on the other hand the PCP master interface can also access any address on the FPI bus. Normally, the CPU initializes the control registers via FPI Bus access. Thereafter, the PCP should not access its control registers itself through PCP instructions. Apart from the access via FPI Bus, there is no direct way to the PCP control registers.

Accesses to the PCP control and status register, the PRAM, and the CMEM are detailed in the following sections.

10.13.1 Access to the PCP Control Registers from the FPI Bus

FPI Bus accesses to the PCP control registers must always be performed in Supervisor Mode with word accesses; byte or half-word accesses will result in a bus error.

All PCP control registers can be read at any time. Register PCP_CS can be optionally Endinit-protected via bit PCP_CS.EIE (see **Page 10-61**). If CS.EIE & ENDINIT = True, then any write access to this register is inhibited. The clock control register PCP_CLC is endinit-protected. The Software In-System Test register is also endinit-protected.

10.13.2 Access to the PRAM from the FPI Bus

FPI Bus accesses to the PRAM must always be performed with word accesses; byte or half-word accesses will result in a bus error.

Attention needs to be paid when accessing the CSAs and data sections of the PCP channel programs. The location of a specific channel's CSA is dependent on the chosen Context Model, Full, Small or Minimum Context. **Table 10-10** shows these addresses.

Table 10-10 FPI Bus Access to CSAs

Channel	Full Context	Small Context	Minimum Context
0 (see note)	PRAM Base Address + 00_H	PRAM Base Address + 00_H	PRAM Base Address + 00_H
1	PRAM Base Address + 20_H	PRAM Base Address + 10_H	PRAM Base Address + 08_H
2	PRAM Base Address + 40_H	PRAM Base Address + 20_H	PRAM Base Address + 10_H
3	PRAM Base Address + 60_H	PRAM Base Address + 30_H	PRAM Base Address + 18_H
n	PRAM Base Address + $n \times 20_H$	PRAM Base Address + $n \times 10_H$	PRAM Base Address + $n \times 08_H$

Note: Since channel 0 is not defined (no service request with SRPN = 0), the first area is not an actual CSA. It is recommended that this area should not be used by PCP channel programs.

The FPI Bus address of a word location pointed to by the Data Pointer R7_DPTR is calculated by the following formula:

- Effective FPI Bus address[31:0] = (PRAM Base Address) + (<DPTR> << 6)

10.13.3 Access to the CMEM from the FPI Bus

FPI Bus accesses to the CMEM must always be performed with word accesses; byte or half-word accesses will result in a bus error.

When using a channel entry table, the FPI Bus address of a specific channel's entry location is given by the following formula:

- Effective FPI Bus address[31:0] = (CMEM Base Address) + $04_H \times$ channel number

The FPI Bus address of an instruction pointed to by the PCP program counter, PC, is calculated by the following formula:

- Effective FPI Bus address[31:0] = (CMEM Base Address) + <PC> << 1

10.14 Debugging the PCP

For debugging the PCP, a special instruction, DEBUG, is provided. This instruction can only be used when the PCP is in Debug Mode. It can be placed at important locations inside the code to track and trace program execution. The execution of the instruction depends on a condition code specified with the instruction. The actions programmed for this instruction will only take place if the specified condition is true.

The following actions are performed when the DEBUG instruction is executed and the condition code is true:

- Store the current PC, i.e. the address of the DEBUG instruction, in register PCP_ES.EPC
- Store the current channel number in register PCP_ES.EPN

In addition, the following operations can be programmed through fields in the DEBUG instruction:

- Optionally stop the channel program execution (instruction field SDB)
- Optionally generate an external debug event at pin BRKOUT (instruction field EDA)
- Optionally prevent the PCP from executing any further channel programs (instruction field DAC)
- Optionally cause the PCP to decrement the PC prior to saving the channel context (instruction field RTA)

If the DEBUG instruction is programmed to stop the channel program execution, the action taken by the PCP depends on the value of the RTA instruction field:

- If RTA = 0, the PCP disables further invocations of the current channel through clearing bit R7.CEN, and then performs a context save. The execution of this channel is stopped at the point of the DEBUG instruction. If the DAC instruction field = 0, the PCP will continue to operate, accepting service requests for other channels as they arise. Since the stopped channel was disabled before saving its context, service requests for this channel will result in an error exit (see **Page 10-31**). When re-enabling the channel, its enable bit CEN in the saved context location CR7 must be set.
- If RTA = 1, the PCP does not modify bit R7.CEN (i.e. the channel remains enabled), decrements the PC (so that it again points to the DEBUG instruction), and then performs a context save. The execution of this channel is stopped at the point of the DEBUG instruction. If the DAC instruction field = 0, the PCP will continue to operate, accepting service requests for channels as they arise. Since the stopped channel was not disabled before saving its context, service requests for this channel will not result in an error exit, but will simply cause re-execution of the DEBUG instruction and hence a repeat of the channel exit.

Note: When a channel is stopped by DEBUG, the context of the stopped channel will be saved to the appropriate region of the CSA before the channel terminates. Where a Small or Minimum Context model is being used, the values of the GPRs not

included in the context will not be saved, and indeed these register values may be changed by the operation of another active channel. In this case, the required registers should be explicitly saved to PRAM by store instructions prior to execution of the DEBUG instruction.

If the DEBUG instruction is programmed to stop all channel program execution, the PCP disables further invocations of any channel by clearing bit PCP_CS.EN. The execution of this channel is only stopped according to the SDB instruction field value. The PCP will only start to reaccept service requests when PCP_CS.EN is written to 1.

Note: The DEBUG instruction must be only used in DEBUG mode; otherwise it will generate an IOP error.

Note: If PCP_CS.RCB = 0 (Channel Resume Mode), then the channel program will begin executing at whichever PC is restored from the context location CR7.PC. If PCP_CS.RCB = 1 (Channel Restart Mode), then the channel program is forced to always start at its channel entry table location, no matter what the restored context value is for the PC. This means that in Channel Restart Mode, it is not possible to restart the channel program from where it was halted by the debug event. It is recommended that when using Channel Restart Mode, the user should also program all EXIT instructions with the "EP = 0" setting to allow selection of Channel Resume Mode for debugging without changing operation of the channel programs.

10.15 PCP Registers

The PCP can be viewed as being a peripheral on the FPI Bus. As with any other peripheral, there are control registers, normally set by the CPU acting as an external FPI Bus master to the PCP during initialization. Control registers select the operating modes of the PCP, and status registers provide information about the current state of the PCP to the external FPI Bus master.

Accessing of Control Registers

The control registers are accessible by any master via the FPI Bus. The control registers must be configured at initialization and then left unaltered. This is typically done by the CPU. The only setting that can be dynamically modified is the PCP_CS.EN bit. All other bits must only be modified when PCP_CS.EN = 0 and PCP_CS.RS = 0.

The PCP control and status registers are accessible only to the CPU when it is operating in Supervisor mode. PCP control and status registers must be accessed with 32-bit read and write operations only.

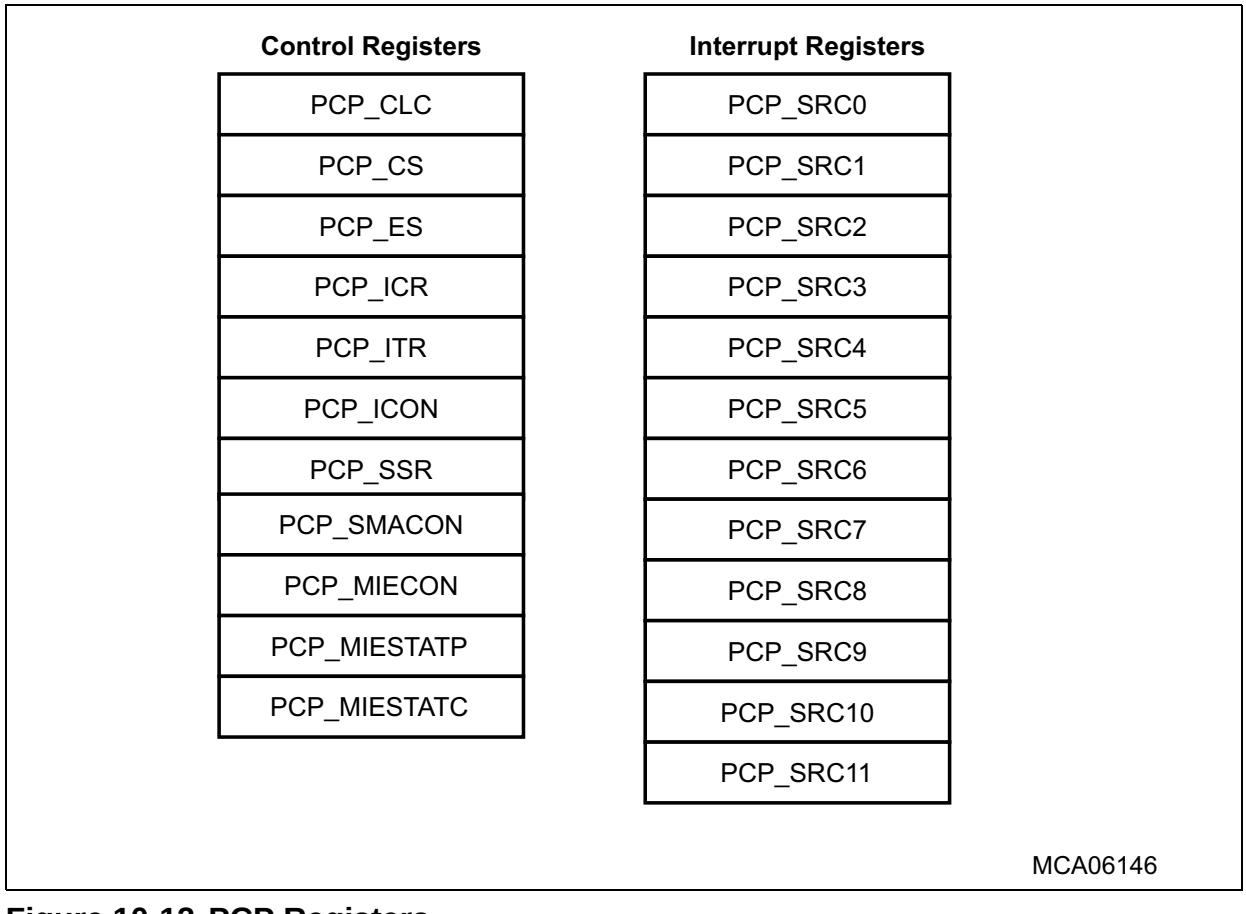


Figure 10-12 PCP Registers

PCP Register Overview

The address map of the PCP starts at its base address as shown in **Table 10-19**. The address offsets of the PCP registers are described in **Table 10-11**.

Table 10-11 Register Overview of PCP

Short Name	Description	Offset Addr.[1]	Access Mode		Reset	Description See
			Read	Write		
PCP_CLC	Clock Control Register	00_H	U, SV, 32	SV, 32, E	FPI Reset	**Page 10-59**
PCP_ID	Module Identification Register	08_H	U, SV, 32	SV, 32[2]	FPI Reset	**Page 10-60**
PCP_CS	Control/Status Register	10_H	U, SV, 32	SV, 32, E[3]	FPI Reset	**Page 10-61**
PCP_ES	Error/Debug Status Register	14_H	U, SV, 32	SV, 32[2]	FPI Reset	**Page 10-63**
PCP_ICR	Interrupt Control Register	20_H	U, SV, 32	SV, 32	FPI Reset	**Page 10-66**
PCP_ITR	Interrupt Threshold Control Register	24_H	U, SV, 32	SV, 32	FPI Reset	**Page 10-68**
PCP_ICON	Interrupt Configuration Register	28_H	U, SV, 32	SV, 32[2]	FPI Reset	**Page 10-69**
PCP_SSR	Stall Status Register	$2C_H$	U, SV, 32	SV, 32[2]	FPI Reset	**Page 10-71**
PCP_SMACON	SIST Mode Access Control Register	40_H	U, SV, 32	SV, 32	FPI Reset	**Page 10-73**
PCP_SRC11	Service Request Control Register 11	$D0_H$	U, SV, 32	SV, 32[2]	FPI Reset	**Page 10-77**
PCP_SRC10	Service Request Control Register 10	$D4_H$	U, SV, 32	SV, 32[2]	FPI Reset	**Page 10-77**
PCP_SRC9	Service Request Control Register 9	$D8_H$	U, SV, 32	SV, 32[2]	FPI Reset	**Page 10-77**
PCP_SRC8	Service Request Control Register 8	DC_H	U, SV, 32	SV, 32	FPI Reset	**Page 10-76**
PCP_SRC7	Service Request Control Register 7	$E0_H$	U, SV, 32	SV, 32	FPI Reset	**Page 10-76**

Table 10-11 Register Overview of PCP

Short Name	Description	Offset Addr.[1]	Access Mode		Reset	Description See
			Read	Write		
PCP_SRC6	Service Request Control Register 6	$E4_H$	U, SV, 32	SV, 32	FPI Reset	Page 10-76
PCP_SRC5	Service Request Control Register 5	$E8_H$	U, SV, 32	SV, 32	FPI Reset	Page 10-76
PCP_SRC4	Service Request Control Register 4	EC_H	U, SV, 32	SV, 32	FPI Reset	Page 10-76
PCP_SRC3	Service Request Control Register 3	$F0_H$	U, SV, 32	SV, 32[2]	FPI Reset	Page 10-75
PCP_SRC2	Service Request Control Register 2	$F4_H$	U, SV, 32	SV, 32[2]	FPI Reset	Page 10-75
PCP_SRC1	Service Request Control Register 1	$F8_H$	U, SV, 32	SV, 32[2]	FPI Reset	Page 10-74
PCP_SRC0	Service Request Control Register 0	FC_H	U, SV, 32	SV, 32[2]	FPI Reset	Page 10-74

1) The absolute register address is calculated as follows:
Module Base Address + Offset Address (shown in this column)

2) A write access is allowed to this register (i.e. no bus error is generated) but as there are no writable bits the register value is unaffected by the write.

3) Endinit protection is controlled by the EIE bit.

10.16 PCP Registers Address Space

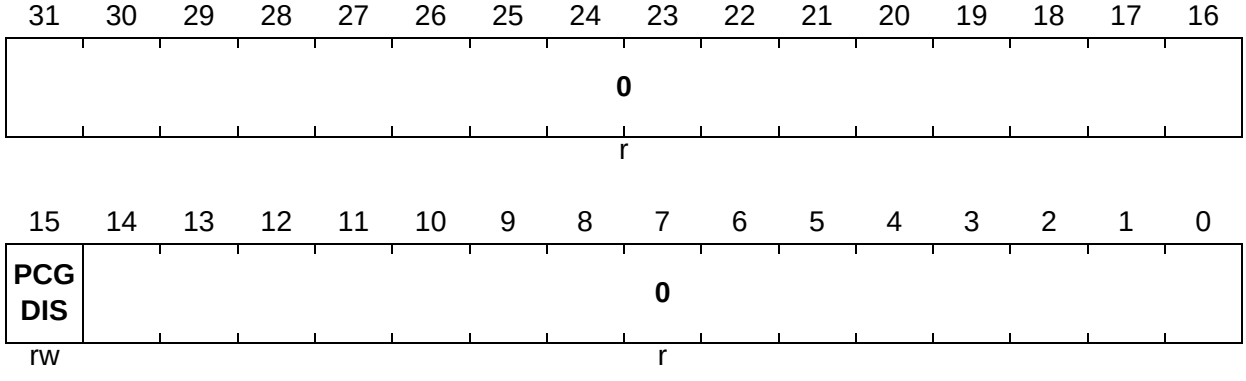
Table 10-12 Registers Address Space

Module	Base Address	End Address	Note
PCP	$F004\ 3F00_H$	$F004\ 3FFF_H$	

10.17 Registers

10.17.1 PCP Clock Control Register, PCP_CLC

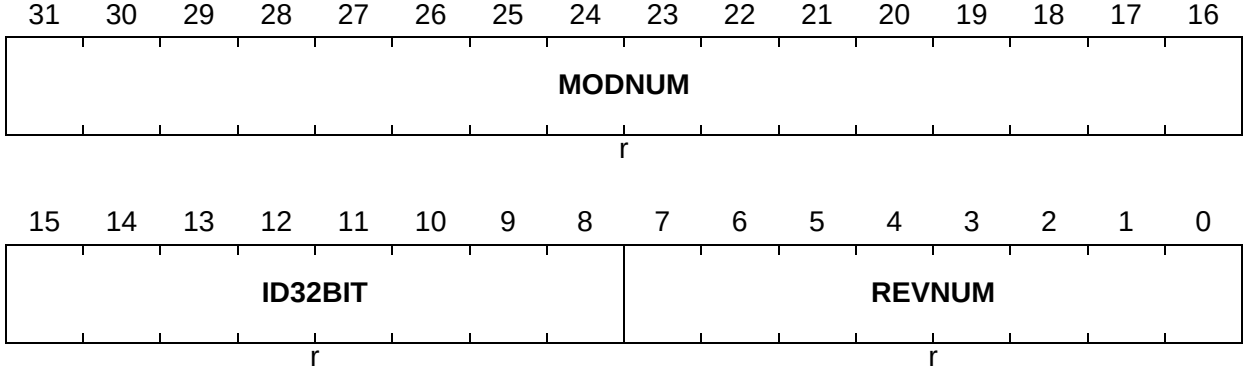
PCP_CLC
PCP Clock Control Register **(00_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.
PCGDIS	15	rw	**Clock Gating Disable Bit** Allows clock gating to be disabled. 0_B PCP Internal Clock stops when PCP is idle (default after reset) 1_B PCP Internal Clock always runs
0	[14:0]	r	**Reserved** Read as 0; should be written with 0.

10.17.2 PCP Module Identification Register, PCP_ID

PCP_ID
PCP Module Identification Register (08_H) Reset Value: 0020 C0XX$_H$



Field	Bits	Type	Description
MODNUM	[31:16]	r	**PCP Identification Number** value = 0020_H
ID32BIT	[15:8]	r	**32-bit Module Identification Number Marker** value = $C0_H$
REVNUM	[7:0]	r	**PCP Revision Number** **Implementation specific**

10.17.3 PCP Control and Status Register, PCP_CS

This register can be Endinit-protected via bit EIE.

PCP_CS
PCP Control/Status Register (10_H) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ESR								CWT							CWE
rw								rw							rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PPS							PPE	CS		EIE	RCB	0	RS	0	EN
rw							rw	rw		rw	rw	r	rh	r	rw

Field	Bits	Type	Description
ESR	[31:24]	rw	**Error Service Request Number** SRPN for interrupt to CPU on an error condition. 00_H No interrupt request posted (default) nn_H Post an as SRPN interrupt to CPU on an error condition (n not equal 00_H)
CWT	[23:17]	rw	**Channel Watchdog Threshold** 0_D Reserved 1_D Threshold = 16 instructions $_D$ … … n_D Threshold = 16 × ‘n’ instructions
CWE	16	rw	**Channel Watchdog Enable** 0_B Disable Channel Watchdog 1_B Enable Channel Watchdog *Note: When enabled, the Channel Watchdog counts the number of instructions executed since the channel started. If this number exceeds the Channel Watchdog Threshold, a PCP error is generated.*

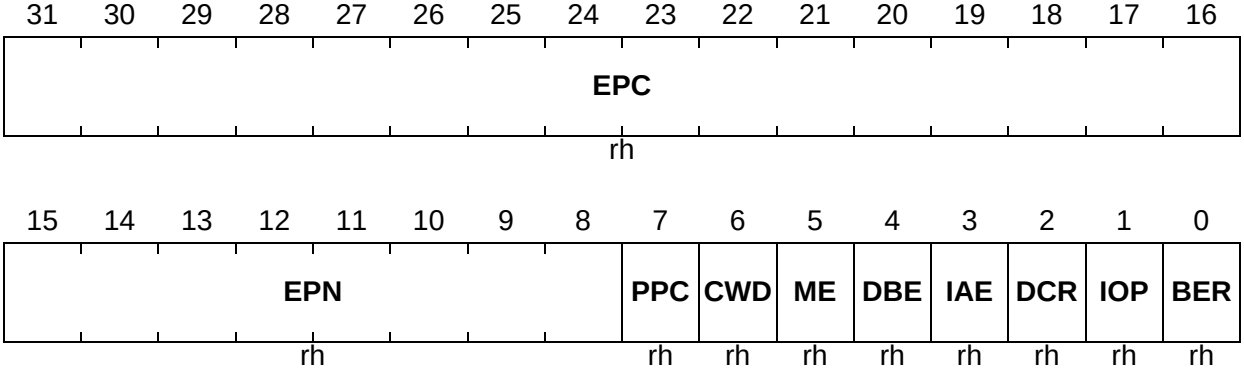
Field	Bits	Type	Description
PPS	[15:9]	rw	**PRAM Partition Size** 0_D Default, only allowed with PPE = 0 1_D CSA contains 3 context save regions $_D$ … … n_D CSA contains 1 + 2 × n context save regions *Note: The actual size of the CSA (in words) is given by the formula (2 × n + 1) × m, where m is the number of registers in the selected Context Model.* *If PPE = 1 and the PCP attempts to perform a data write to PRAM addresses below the CSA, an error condition has occurred.* *This setting also controls the maximum channel number (MCN) used in system. The maximum channel number is MCN = 2 × n. If the SRPN is greater than MCN, an error condition has occurred.* *For example, setting PPS to n = 3 will give a CSA containing 7 context save regions. As channel 0 cannot be used and MCN = 6, channels 1 to 6 are allowed.*
PPE	8	rw	**PRAM Partitioning Enable** 0_B PRAM is not partitioned 1_B PRAM is partitioned *Note: When partitioned, the PRAM is divided into two areas (CSA and remainder). A PCP error will be generated on an inappropriate action in either region (PCP write operation with a target address in the CSA, or context restore from outside the CSA).*
CS	[7:6]	rw	**Context Size Selection** 00_B Use Full Context for all channels 01_B Use Small Context for all channels 10_B Use Minimum Context for all channels 11_B Reserved
EIE	5	rw	**Endinit Enable** 0_B PCP_CS is not Endinit protected. 1_B PCP_CS is Endinit protected.

Field	Bits	Type	Description
RCB	4	rw	**Channel Start Mode Control** 0_B Channel resume operation mode selected; channel start PC is taken from restored context 1_B Channel restart operation mode selected; channel start PC is derived from the requested channel number (= priority number of service request) *Note: This is a global control bit and applies to all channels.*
0	3	r	**Reserved** Read as 0; should be written with 0.
RS	2	rh	**PCP Run/Stop Status Flag** 0_B PCP is stopped or idle (default) 1_B PCP is currently running
0	1	r	**Reserved** Read as 0; should be written with 0.
EN	0	rw	**PCP Enable** 0_B PCP is disabled for operation (default) 1_B PCP is enabled for operation *Note: This bit does not enable/disable clocks for power saving. It stops the PCP from accepting new service requests.*

10.17.4 PCP Error/Debug Status Register, PCP_ES

This is a read-only register, providing state information about error and debug conditions.

PCP_ES
PCP Error/Debug Status Register (14_H) Reset Value: 0000 0000_H



Field	Bits	Type	Description
EPC	[31:16]	rh	**Error PC** PC value of the instruction that was executing when an error or debug event occurred. Default = 0000_H.
EPN	[15:8]	rh	**Error Service Request Priority Number** Channel number of the channel that was operating when the last error/debug event occurred. The value stored is the SRPN which invoked this channel (= channel number), NOT the current PCP priority number stored in field CPPN in register PICR. Default = 00_H.
PPC	7	rh	**PRAM Partitioning Check** Set if the last error/debug event was an error generated by a channel program attempting to perform a write to a PRAM address within the CSA, or receipt of an interrupt request that would have caused a context restore from outside the CSA.
CWD	6	rh	**Channel Watchdog Triggered** Set if the last error/debug event was an error generated by a channel program attempting to execute more instructions than allowed by PCP_CS.CWT.
ME	5	rh	**Memory Error** This bit is set if a PCP internal memory error has occurred. See **Table 10.22 “Implementation of the PCP in the TC1797” on Page 10-138** for TC1797 specific implementation.
DBE	4	rh	**Debug Event Flag** Set if the last error/debug event was a debug event. *Note: A debug event does not cause the posting of an interrupt to the CPU.*
IAE	3	rh	**Instruction Address Error** Set if the last error/debug event was an error generated by the PCP attempting to fetch an instruction from an address outside the implemented CMEM range as a result of a jump or branch instruction; otherwise, clear.

Field	Bits	Type	Description
DCR	2	rh	**Disabled Channel Request Flag** Set if the last error/debug event was an error generated by receipt of an interrupt request with an SRPN that attempted to start a disabled PCP channel; otherwise, clear.
IOP	1	rh	**Invalid Opcode** Set if the last error/debug event was an error generated by the PCP attempting to execute an Invalid Opcode (i.e. the value fetched from CMEM for execution by the PCP did not represent a valid instruction), otherwise clear.
BER	0	rh	**Bus Error Flag** Set if the last error/debug event was an error generated by an FPI Bus error or an invalid address access; otherwise, clear. *Note: An FPI Bus error event does not cause the PCP to post an error interrupt to the CPU. An FPI Bus error interrupt is, however, generated by the FPI control logic.*

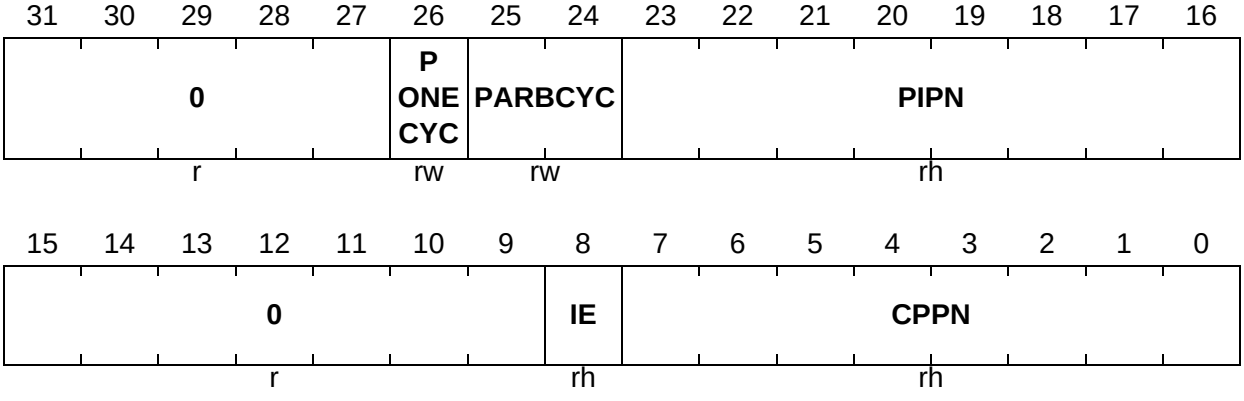
Note: An interrupt request with the SRPN held in PCP_CS.ESR is posted to the CPU whenever a PCP error event, other than an FPI Bus error occurs. FPI Bus error interrupt generation is automatically handled by the FPI Bus control logic, rather than by the PCP. The execution of a DEBUG instruction is not classed as an error event and does not therefore generate an interrupt request to the CPU. The entire contents of the register are updated whenever there is a debug or an error event detected (i.e. all status/error bits, other than the bit representing the last PCP error/debug event, are cleared). This register therefore only provides a record of the last error/debug event encountered. The only way to clear this register is to reset the PCP.

10.17.5 PCP Interrupt Control Register, PCP_ICR

This register controls the operation of the PCP Interrupt Control Unit (PICU).

PCP_ICR

PCP Interrupt Control Register (20_H) Reset Value: $0000\ 0000_H$

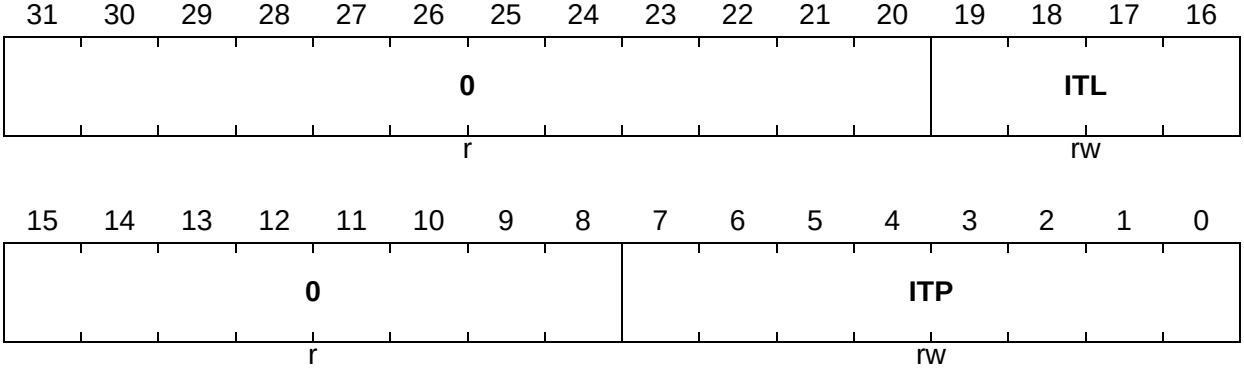


Field	Bits	Type	Description
0	[31:27]	r	**Reserved** Read as 0; should be written with 0.
PONECYC	26	rw	**Clocks per Arbitration Cycle Control** This bit determines the number of clocks per arbitration cycle. 0_B Two clocks per arbitration cycle (default) 1_B One clock per arbitration cycle
PARBCYC	[25:24]	rw	**Number of Arbitration Cycles Control** This bit field controls the number of arbitration cycles used to determine the request with the highest priority. It follows the same coding scheme as described for the CPU interrupt arbitration. 00_B Four arbitration cycles (default) 01_B Three arbitration cycles 10_B Two arbitration cycles 11_B One arbitration cycle
PIPN	[23:16]	rh	**Pending Interrupt Priority Number** This read-only field is updated by the PICU at the end of each arbitration process, and indicates the priority number of a pending request. PIPN is set to 00_H when no request is pending and at the beginning of a new arbitration process.
0	[15:9]	r	**Reserved** Read as 0; should be written with 0.

Field	Bits	Type	Description
IE	8	rh	**Reserved**
CPPN	[7:0]	rh	**Current PCP Priority Number** This field indicates the current priority level of the PCP and is automatically updated by hardware on entry into an interrupt service routine.

10.17.6 PCP Interrupt Threshold Register, PCP_ITR

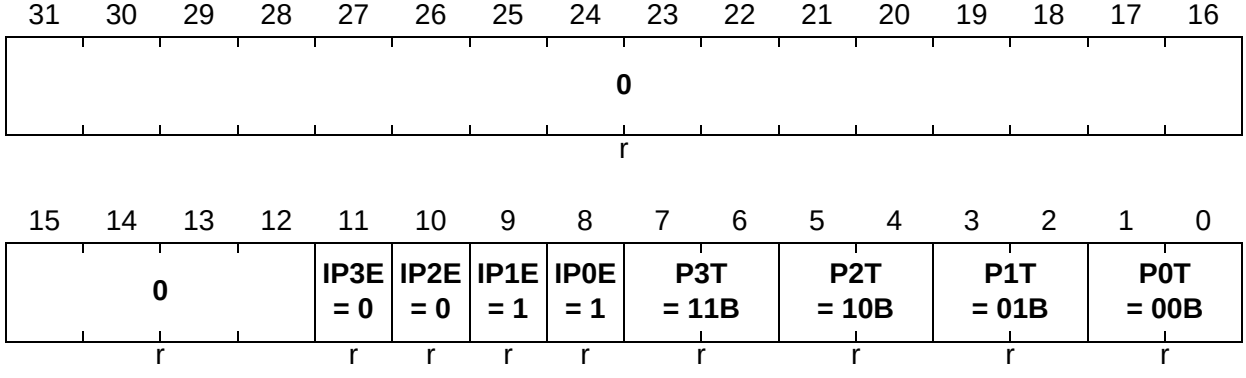
PCP_ITR
PCP Interrupt Threshold Control Register
(24_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
0	[31:20]	r	**Reserved** Read as 0; should be written with 0.
ITL	[19:16]	rw	**Interrupt Threshold Level** This bit field specifies the number of active interrupt entries at which an warning interrupt should be issued to the interrupt queue associated with interrupt bus 0 (i.e. when the number of active port 0 interrupt requests stored in all SRCx registers reaches this value then an interrupt is posted to port 0 with the priority programmed into the ITP field). When ITL is programmed to 0 or is ≥ the number of SRCx registers that can contain port 0 interrupt requests, the threshold warning mechanism is disabled).
0	[15:8]	r	**Reserved** Read as 0; should be written with 0.
ITP	[7:0]	rw	**PCP Interrupt Threshold Service Request Priority Number** This field contains the interrupt priority that is to be posted to the interrupt queue associated with interrupt bus 0 when the threshold condition is reached (setting this value to 0 or disables the threshold detection mechanism).

10.17.7 PCP Interrupt Configuration Register, PCP_ICON

PCP_ICON
PCP Interrupt Configuration Register (28_H) **Reset Value: 0000 03E4$_H$**

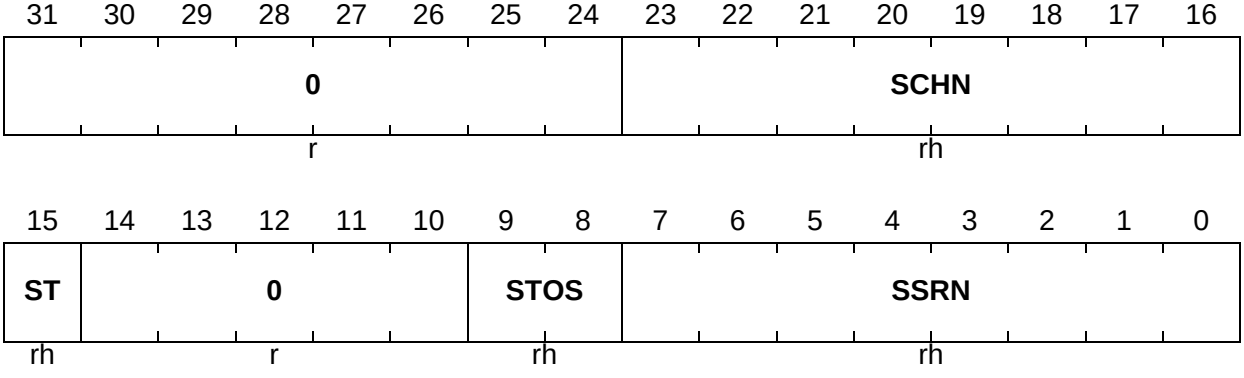


Field	Bits	Type	Description
0	[31:12]	r	**Reserved** Read as 0.
IP3E	11	r	**PCP Interrupt Bus 3 Enable** This bit reflects the status of interrupt bus 3. Interrupt bus 3 is always disabled (not implemented in the TC1797).
IP2E	10	r	**PCP Interrupt Bus 2 Enable** This bit reflects the status of interrupt bus 2. Interrupt bus 2 is always disabled (not implemented in the TC1797).
IP1E	9	r	**PCP Interrupt Bus 1 Enable** This bit reflects the status of interrupt bus 1 (PCP interrupt arbitration bus). Interrupt bus 1 is always enabled.
IP0E	8	r	**PCP Interrupt Bus 0 Enable** This bit reflects the status of interrupt bus 0 (CPU interrupt arbitration bus). Interrupt bus 0 is always enabled.
P3T	[7:6]	r	**PCP Interrupt Bus 3 TOS Mapping** This field reflects the TOS associated with interrupt bus 3. *Note: Interrupt bus 3 is not available in the TC1797.*

Field	Bits	Type	Description
P2T	[5:4]	r	**PCP Interrupt Bus 2 TOS Mapping** This field reflects the TOS associated with interrupt bus 2. *Note: Interrupt bus 2 is not available in the TC1797.*
P1T	[3:2]	r	**PCP Interrupt Bus 1 TOS Mapping** This field reflects the TOS associated with interrupt bus 1 (PCP interrupt arbitration bus). The PCP should use this value in R6.TOS when it wishes to raise an interrupt request to itself (the PCP is always connected to interrupt bus 1).
P0T	[1:0]	r	**PCP Interrupt Bus 0 TOS Mapping** This field reflects the TOS associated with interrupt bus 0 (CPU interrupt arbitration bus). The PCP should use this value in R6.TOS when it wishes to raise an interrupt request via interrupt bus 0.

10.17.8 PCP Stall Status Register, PCP_SSR

PCP_SSR
PCP Stall Status Register **($2C_H$)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
0	[31:24]	r	**Reserved** Read as 0.
SCHN	[23:16]	rh	**PCP Stalled Channel Number** This field shows the channel number of the channel that was executing when the last (or present) stall condition occurred. This field can only be cleared by a reset.
ST	15	rh	**PCP Stalled Status** This bit shows the stalled status of the PCP 0_B PCP is not stalled 1_B PCP is stalled
0	[14:10]	r	**Reserved** Read as 0.
STOS	[9:8]	rh	**PCP Stalled Type-of-Service** This field shows the Type-Of-Service to which an interrupt was being posted that caused the last (or present) stall condition (i.e. the service request queue that was full when the PCP attempted to post a request to it). This field can only be cleared by a reset.

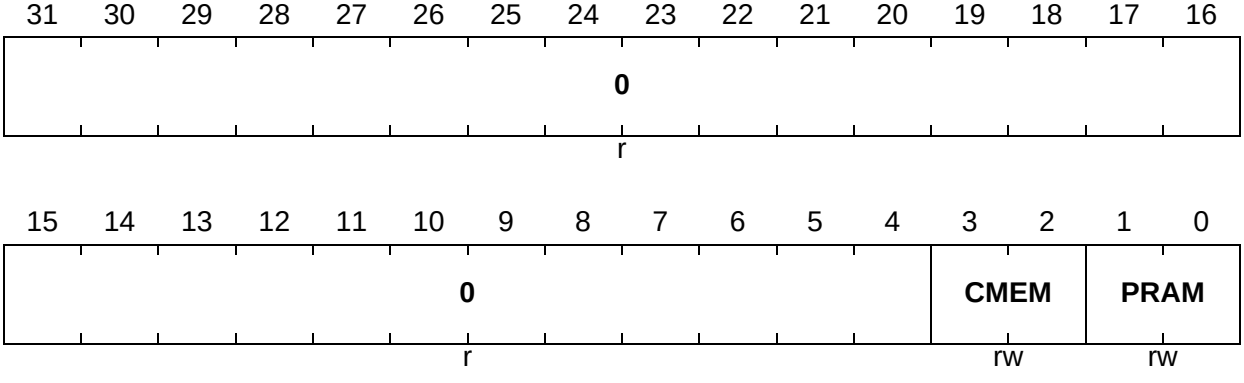
Field	Bits	Type	Description
SSRN	[7:0]	rh	**PCP Stalled Service Request Number** This field shows the Service Request Number that was being posted when the last (or present) stall condition occurred. This field can only be cleared by a reset.

10.17.9 SIST Mode Access Control Register, PCP_SMACON

Note: Please see **Section 10.10** *on* **Page 10-42** *for more information regarding the use of this register.*

This register is ENDINIT protected.

PCP_SMACON
SIST Mode Access Control Register (40_H) **Reset Value: $0000\ 0000_H$**

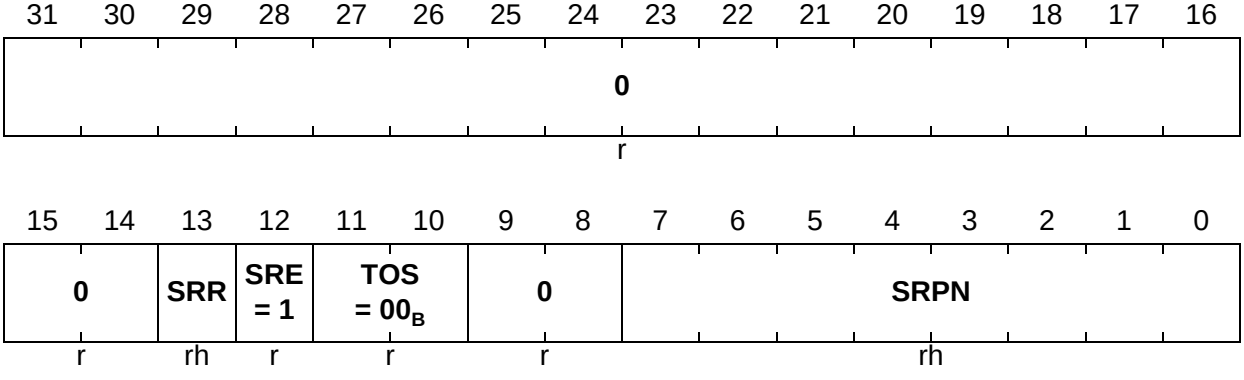


Field	Bits	Type	Description
0	[31:4]	r	**reserved:** returns ‘0’ if read; should be written with ‘0’.
CMEM	[3:2]	rw	**CMEM SIST mode access control** 00_B Normal Operation, No Mapping 01_B Data Array Mapping, no error detection/correction 10_B Check Array Mapping, no error detection/correction 11_B Data Array Mapping, error detection/correction enabled
PRAM	[1:0]	rw	**PRAM SIST mode access control** 00_B Normal Operation, No Mapping 01_B Data Array Mapping, no error detection/correction 10_B Check Array Mapping, no error detection/correction 11_B Data Array Mapping, error detection/correction enabled

10.17.10 PCP Service Request Control Registers m, PCP_SRC[1:0]

Service request nodes for interrupt bus 0 (CPU interrupt arbitration bus).

PCP_SRCm (m = 0-1)
PCP Service Request Control Register m
(FC_H-m*4_H) **Reset Value: 0000 1000$_H$**

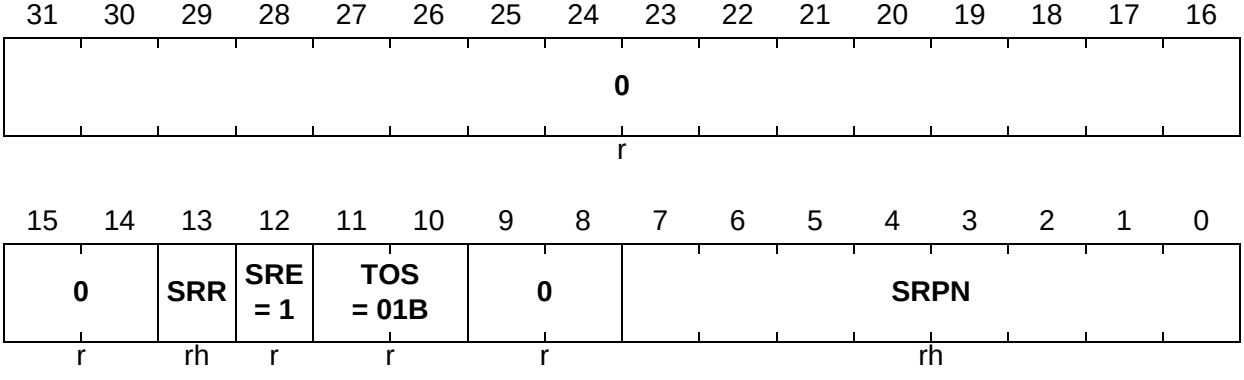


Field	Bits	Type	Description
0	[31:14]	r	**Reserved** Read as 0.
SRR	13	rh	**PCP Node m Service Request Flag** 0_B No service requested (default). 1_B Valid active service requested.
SRE	12	r	**PCP Node m Service Request Enable** Always read as 1 (enabled).
TOS	[11:10]	r	**PCP Node m Type-of-Service State** Always read as 00_B. This means TOS is associated with interrupt bus 0 (CPU interrupt arbitration bus).
0	[9:8]	r	**Reserved** Read as 0.
SRPN	[7:0]	rh	**PCP Node m Service Request Priority Number** This number is automatically set by the PCP if it needs to place a service request on interrupt bus 0 (CPU interrupt arbitration bus). Default after reset is 00_H.

10.17.11 PCP Service Request Control Registers m, PCP_SRC[3:2]

Service request nodes for interrupt bus 1 (PCP interrupt arbitration bus).

PCP_SRCm (m = 2-3)
PCP Service Request Control Register m
(FC_H-m*4_H) **Reset Value: 0000 1400$_H$**

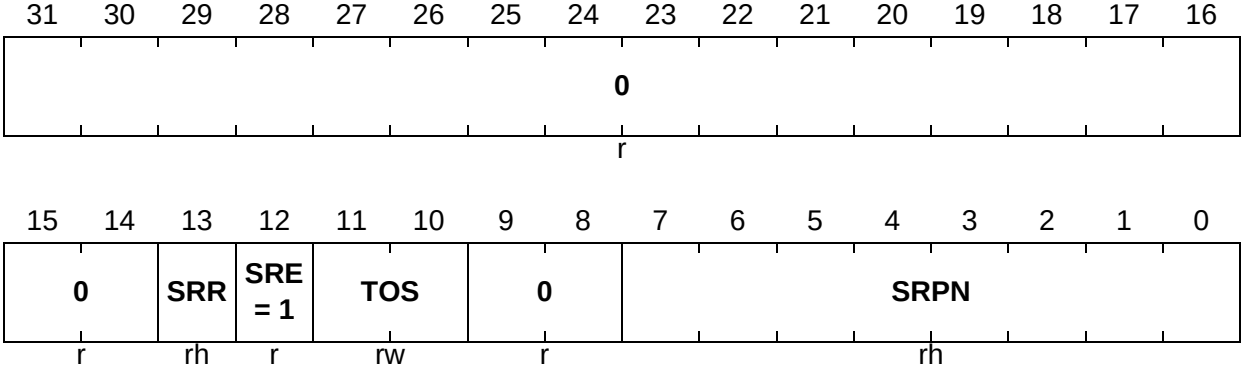


Field	Bits	Type	Description
0	[31:14]	r	**Reserved** Read as 0.
SRR	13	rh	**PCP Node m Service Request Flag** 0_B No service requested (default). 1_B Valid active service requested.
SRE	12	r	**PCP Node m Service Request Enable** Always read as 1 (enabled).
TOS	[11:10]	r	**PCP Node m Type-of-Service State** Always read as 01_B. This means TOS is associated with interrupt bus 1 (PCP interrupt arbitration bus).
0	[9:8]	r	**Reserved** Read as 0.
SRPN	[7:0]	rh	**PCP Node m Service Request Priority Number** This number is automatically set by the PCP if it needs to place a service request on interrupt bus 1 (PCP interrupt arbitration bus). Default after reset is 00_H.

10.17.12 PCP Service Request Control Registers m, PCP_SRC[8:4]

Service request nodes programmable for interrupt bus 0 (CPU interrupt arbitration bus) or 1 (PCP interrupt arbitration bus).

PCP_SRCm (m = 4-8)
PCP Service Request Control Register m
(FC_H-m*4_H) **Reset Value: 0000 1000$_H$**



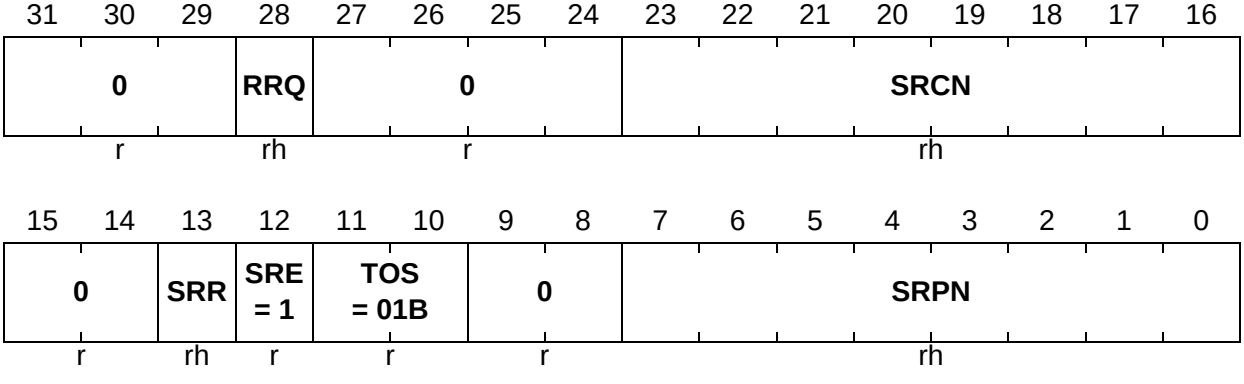
Field	Bits	Type	Description
0	[31:14]	r	**Reserved** Read as 0. Should be written with 0.
SRR	13	rh	**PCP Node m Service Request Flag** 0_B No service requested (default). 1_B Valid active service requested.
SRE	12	r	**PCP Node m Service Request Enable** Always read as 1 (enabled).
TOS	[11:10]	rw	**PCP Node m Type-of-Service State/Control** TOS value depends on the interrupt mapping that has been selected (see **Page 10-132**). This bit field must be written at PCP configuration time and should not subsequently modified while the PCP is operating.
0	[9:8]	r	**Reserved** Read as 0. Should be written with 0.

Field	Bits	Type	Description
SRPN	[7:0]	rh	**PCP Node m Service Request Priority Number** This number is automatically set by the PCP if it needs to place a service request on interrupt bus 0 (CPU interrupt arbitration bus) or 1 (PCP interrupt arbitration bus). Default after reset is 00_H.

10.17.13 PCP Service Request Control Registers m, PCP_SRC[11:9]

Service request nodes for interrupt bus 1 (PCP interrupt arbitration bus) with suspended interrupt capability.

PCP_SRCm (m = 9-11)
PCP Service Request Control Register m
(FC_H-m*4_H) **Reset Value: 0000 1400$_H$**



Field	Bits	Type	Description
0	[31:29]	r	**Reserved** Read as 0.
RRQ	28	rh	**PCP Node m Channel Restart Request** Set when this service request register n contains an active service request that is associated with a suspended interrupt (i.e. a channel that has been interrupted by a higher-priority channel). 0_B The interrupt is not suspended 1_B The interrupt is suspended RRQ is always 0 when SRR is 0.
0	[27:24]	r	**Reserved** Read as 0.

Field	Bits	Type	Description
SRCN	[23:16]	rh	**PCP Node m Service Request Channel Number** Channel Number Entry (default = 0). When the PCP interrupt request was raised by the PCP Processor Core when executing an EXIT instruction, then this bit field contains the SRPN value taken from R6 when the exit instruction was executed. When the PCP interrupt request was raised by the PCP Processor Core when suspending execution of a channel program in order to service a higher-priority interrupt then this bit field contains the channel number of the channel that was suspended.
0	[15:14]	r	**Reserved** Read as 0.
SRR	13	rh	**PCP Node m Service Request Flag** 0_B No service requested (default) 1_B Valid active service requested
SRE	12	r	**PCP Node m Service Request Enable** Always read as 1 (enabled).
TOS	[11:10]	r	**PCP Node m Type-of-Service State** Always read as 01_B. This means TOS is associated with interrupt bus 1 (PCP interrupt arbitration bus).
0	[9:8]	r	**Reserved** Read as 0.
SRPN	[7:0]	rh	**PCP Node m Service Request Priority Number** This number is automatically set by the PCP if it needs to place a service request on interrupt bus 0 (CPU interrupt arbitration bus) or 1 (PCP interrupt arbitration bus). When the PCP interrupt request contained was raised by the PCP Processor Core when executing an EXIT instruction then this bit field contains the SRPN value taken from R6 when the exit instruction was executed. When the PCP interrupt request was raised by the PCP Processor Core when suspending execution of a channel program in order to service a higher-priority interrupt then this bit field contains the CPPN value of the PCP when the interrupt was suspended.

10.18 PCP Instruction Set Details

This section describes the instruction set architecture of the PCP in detail.

10.18.1 Instruction Codes and Fields

All PCP instructions use a common set of fields to describe such things as the source register, and the state of flags. Additionally, many instructions (including arithmetic and many flow control instructions), are conditionally executed.

The descriptions of the PCP instructions are based on the following conventions.

>>, <<	Shift left or right, respectively.
[ ]	Indirect access based on contents of brackets (de-reference).
#immNN	Immediate value encoded into an instruction with width NN.
#offsetNN	Address offset immediate value with width NN.
NextPC	The current executing instruction’s address + 1. (The next instruction to be fetched.)
cc_A, cc_B	Condition Code CONDCA/CONDCB.

10.18.1.1 Conditional Codes

Many PCP instructions have the option of being executed conditionally. The condition code of an instruction is the field that specifies the condition to be tested before the instruction is executed. Depending on the type of instruction there are 8 or 16 condition codes available. The set of 8-condition codes is referred to as CONDCA, while the set of 16-condition codes is referred to as CONDCB. The condition codes are based on an equation of the Flags held in R7. See **Table 10-13**.

Table 10-13 Condition Codes

Description	CONDCA/B	Test (Flag Bits)	Code	Mnemonic
Unconditional	A / B	–	0_H	cc_UC
Zero/Equal	A / B	Z = 1	1_H	cc_Z
Not Zero/Not Equal	A / B	Z = 0	2_H	cc_NZ
Overflow	A / B	v = 1	3_H	cc_V
Carry/Unsigned Less Than/ Check Bit True	A / B	C = 1	4_H	cc_C, cc_ULT
Unsigned Greater Than	A / B	C OR Z = 0	5_H	cc_UGT
Signed Less Than	A / B	N XOR V = 1	6_H	cc_SLT
Signed Greater Than	A / B	(N XOR V) OR Z = 0	7_H	cc_SGT
Negative	B	N = 1	8_H	cc_N
Not Negative	B	N = 0	9_H	cc_NN
Not Overflow	B	V = 0	A_H	cc_NV
No Carry/Unsigned Greater than or Equal	B	C = 0	B_H	cc_NC, cc_UGE
Signed Greater Than or Equal	B	N XOR V = 0	C_H	cc_SGE
Signed Less than or Equal	B	(N XOR V) OR Z = 1	D_H	cc_SLE
CNT1 Equal Zero	B	CNZ1 = 1	E_H	cc_CNZ
CNT1 Not Equal Zero	B	CNZ1 = 0	F_H	cc_CNN

10.18.1.2 Instruction Fields

Table 10-14 lists the instruction field definitions of the PCP instruction set architecture.

Note: The exact syntax for these fields may be different depending on which tool (e.g. assembler) is used. Please refer to the respective tool descriptions.

Table 10-14 Instruction Field Definitions

Symbol	Syntax	Description
cc_A, cc_B	see Table 10-13	**Condition Code** Specifies conditional execution of instruction according to CONDCA or CONDCB.
CNC		**Counter Control** This field is used by the BCOPY/COPY instructions to control the number of repetitions of the data transfer. For BCOPY operation see Figure 10 - 2 on Page 10-85. For COPY operation see Figure 10 - 1 on Page 10-84.
	CNC = 00_B	Perform the number of transfers specified by CNT0 then proceed to next instruction.
	CNC = 01_B	Perform the number of transfers specified by CNT0 then decrement CNT1 and proceed to next instruction.
	CNC = 10_B	Perform the number of transfers specified by CNT0 then decrement CNT1. Repeat until CNT1 = 0, then proceed to next instruction.
	CNC = 11_B	Reserved.
CNT0		**Counter Reload Value (COPY)** The COPY instruction uses an implicit counter to generate multiple data transfers. The CNT0 value given in the instruction specifies how many data transfers are to be performed by the instruction. See also Figure 10 - 1 on Page 10-84.
	CNT0 = $001_B..111_B$	Perform 1..7 data transfers
	CNT0 = 000_B	Perform 8 data transfers
		Block Size (BCOPY) Selects the FPI block size used for a BCOPY instruction.
	CNT0 = 00_B	Use block size of 8 words.
	CNT0 = 10_B	Use block size of 2 words.
	CNT0 = 11_B	Use block size of 4 words.
	Others	Reserved

Table 10-14 Instruction Field Definitions

Symbol	Syntax	Description
DAC		**Stop PCP**
	DAC = 0	Allow the PCP to continue to execute channel programs in response to service requests.
	DAC = 1	Prevent the PCP from executing further channel programs (PCP_CS.EN = 0).
DST+-		**Destination Address Pointer Control**
	DST (00_B)	No Change (DST)
	DST+ (01_B)	Post Increment by Size (DST+)
	DST- (10_B)	Post Decrement by Size (DST-)
	(11_B)	Reserved
EC		**Exit Count Control**
	EC = 0	No action
	EC = 1	Decrement CNT1
EDA		**External Debug Action**
	EDA = 0	No External Debug Action caused
	EDA = 1	Cause an External Debug Action (breakpoint pin etc.)
EP		**Entry Point Control**
	EP = 0	Set the PC to channel program Start. EP = 0 assumes that a Channel Entry Table exists in the base of CMEM. Failure to provide such a table will cause improper operation.
	EP = 1	Set the PC to the address contained in NextPC (next instruction) address.
INT		**Interrupt Control**
	INT = 0	No Interrupt
	INT = 1	INT = 1 AND (cc_B = True) means Issue Interrupt
RTA		**Action on Debug Exit**
	RTA = 0	Stop channel program from accepting new service requests (clear R7.CEN)
	RTA = 1	Allow further channel program execution and decrement PC (so that DEBUG instruction is re-executed on next invocation)
		Note: This field has no effect if SDB = 0 (see below)
S/C		**Test Bit Control**
	S/C = 0	Check for Clear (0)
	S/C = 1	Check for Set (1)
SDB		**Stop on Debug**
	SDB = 0	Continue running if debug event triggered
	SDB = 1	Stop PCP if debug event triggered

Table 10-14 Instruction Field Definitions

Symbol	Syntax	Description
SIZE		**Data Size Control**
	SIZE = 00_B	Byte (8-bit)
	SIZE = 01_B	Half-word (16-bit)
	SIZE = 10_B	Word (32-bit)
	SIZE = 11_B	Reserved
SRC+-		**Source Address Pointer Control**
	SRC (00_B)	No Change (SRC)
	SRC+ (01_B)	Post Increment by Size (SRC+)
	SRC- (10_B)	Post Decrement by Size (SRC-)
	(11_B)	Reserved
ST		**Stop Channel**
	ST = 0	Continue channel execution, leave channel program enabled
	ST = 1	Stop Channel Execution, perform actions according to RTA setting (see above)

10.18.2 Counter Operation for COPY Instruction

Figure 10 - 1 shows the flow of a COPY instruction.

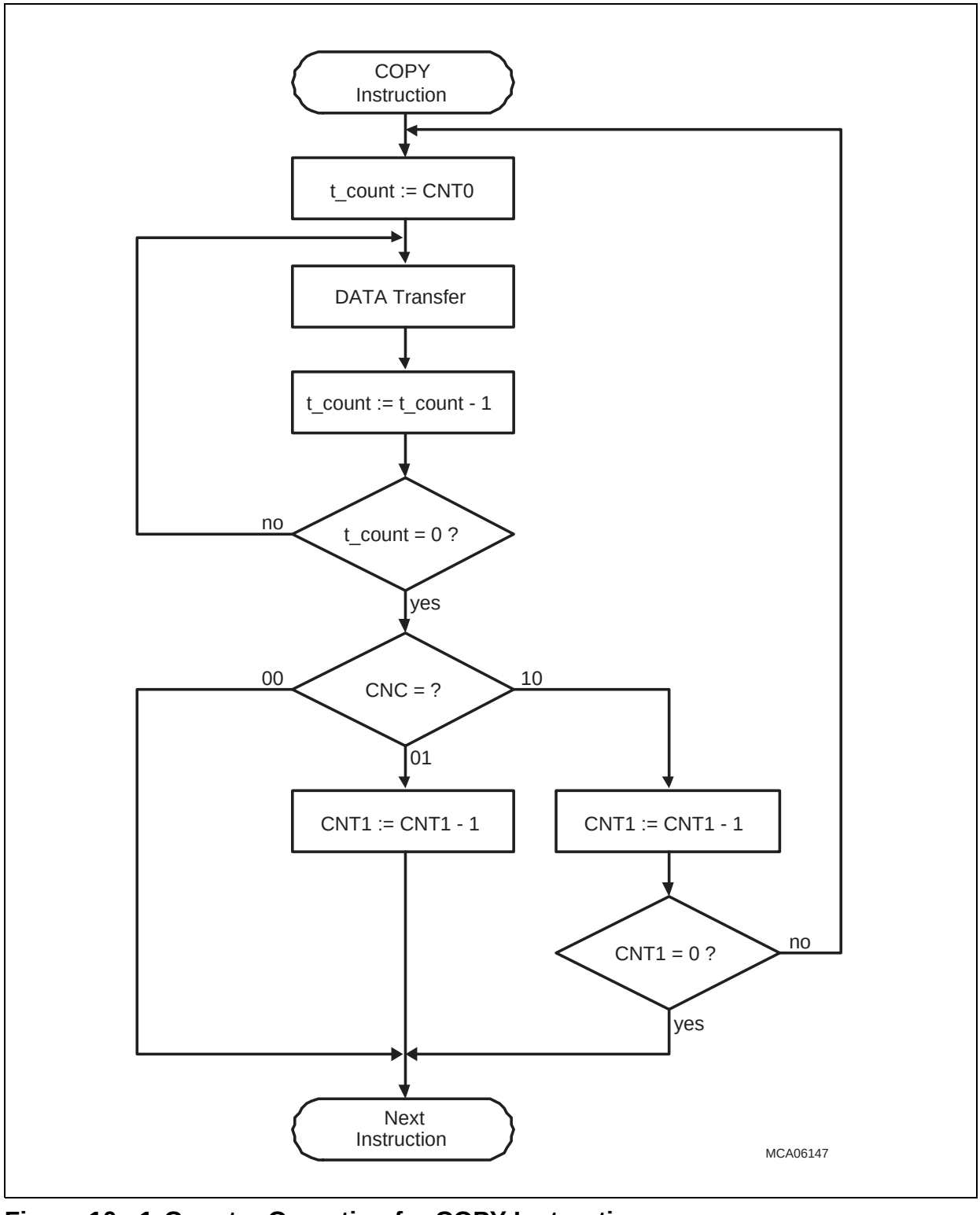


Figure 10 - 1 Counter Operation for COPY Instruction

10.18.3 Counter Operation for BCOPY Instruction

Figure 10 - 2 shows the flow of a BCOPY instruction.

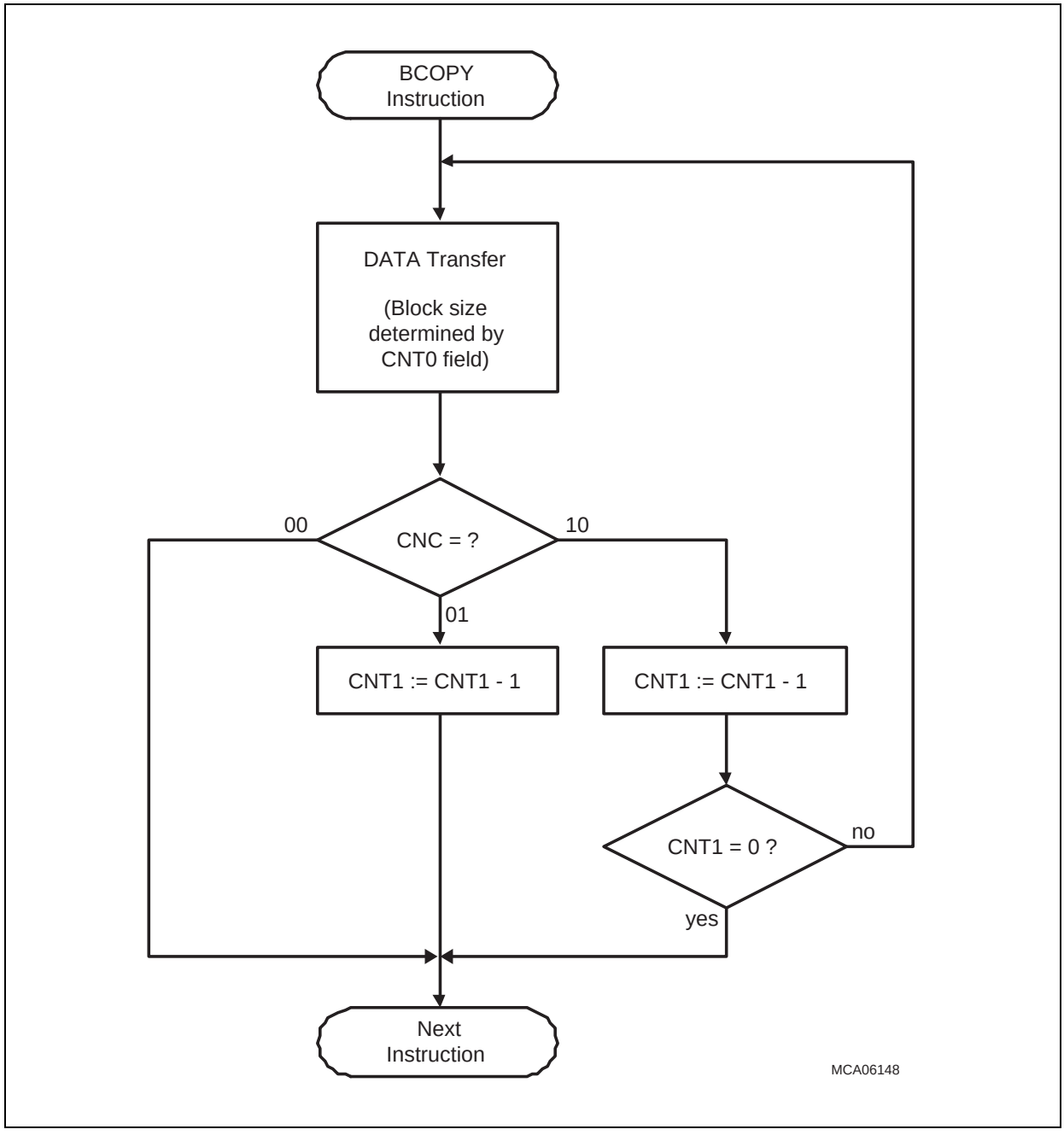


Figure 10 - 2 Counter Operation for BCOPY Instruction

10.18.4 Divide and Multiply Instructions

The PCP has Multiply and Divide capabilities (unsigned values only). All Multiply and divide instructions operate on 8 bits of data (taken from the dividend for divide, from the multiplicand for multiply). This strategy allows the user to implement the appropriate number of instructions (“steps”) as required for the user’s data format.

Each execution of a divide instruction (DSTEP) performs a division which generates 8 bits of result, and also manipulates the registers being used to allow the execution of consecutive divide (DSTEP) instructions to build divide algorithms in multiples of 8 bits (see **Page 10-135** for more details).

Each execution of a multiply instruction (MSTEP32 and MSTEP64) performs a multiplication on 8 bits of data (taken from the multiplicand) and also manipulates the registers to allow execution of consecutive multiply instructions to build multiply algorithms in multiples of 8 bits (see **Page 10-136** for more details).

The following restrictions apply to the use of Divide and Multiply instructions:

- The first instruction of any divide sequence must be the DINIT (initialization) instruction. Any additional instructions other than MINIT, MSTEP32 or MSTEP64 may also be used within the sequence as long as they do not modify any of the registers used for division (R0, Ra and Rb). All subsequent divide instructions within the sequence (DSTEP) must use the same register for dividend and the same register for divisor as used in the preceding DINIT instruction.
- The first instruction of any multiply sequence must be the MINIT (initialization) instruction. Any additional instructions other than DINIT or DSTEP may also be used within the sequence as long as they do not modify any of the registers used for multiplication (R0, Ra and Rb). All subsequent multiply instructions within the sequence (MSTEP32 and MSTEP64) must use the same register for multiplicand and the same register for multiplier as used in the preceding MINIT instruction.
- Neither of the operand registers (Ra or Rb) may be R0 (which is used implicitly within all the instructions), and the same register may not be supplied as both operand registers of an instruction (e.g. DSTEP R3, R3 is invalid).

Note: Failure to adhere to these restrictions will yield undefined results.

1. *Special care must be taken when using multiply and divide sequences when a channel program is interruptible. In this case it must be ensured that a sequence cannot be corrupted by the execution of multiply or divide instructions executed by a higher-priority channel. The R7.IEN bit can be used to ensure that a sequence is not interruptible (see* ***Page 10-134****).*

In the descriptions attached to each multiply and divide instruction, a pseudo-code model is supplied to provide an unambiguous definition of the function of the instruction. The models supplied for the DSTEP and MSTEP32 instructions use 32 bit unsigned integer arithmetic, ignoring any possible overflows. The model supplied for the MSTEP64 uses

a 40-bit unsigned multiply and then shifts this result right by 8 bits (discards the least significant 8 bits of the 40-bit result).

The DSTEP instruction also has some conditions stipulated regarding input values to the instruction. Use of the pseudo-code model for the DSTEP instruction with invalid input values will yield an invalid result.

10.18.5 ADD, 32-bit Addition

This section describes the ADD instructions of the PCP.

ADD	Syntax	**ADD Rb, Ra, cc_A**
	Description	If the condition CONDCA is true, then add the contents of register Ra to the contents of register Rb; place the result in Rb. If CONDCA is false, no operation is performed.
	Operation	if (CONDCA = True) then R[b] = R[b] + R[a] else NOP
	Flags	N, Z, V, C
ADD.I	Syntax	**ADD.I Ra, #imm6**
	Description	Add the zero-extended immediate value imm6 to the contents of register Ra; place the result in Ra.
	Operation	R[a] = R[a] + zero_ext(imm6)
	Flags	N, Z, V, C
ADD.F	Syntax	**ADD.F Rb, [Ra], Size**
	Description	Add the contents of the address location specified by the contents of register Ra to the contents of register Rb; place the result in Rb. *Note: Byte and Half-word values are zero-extended.*
	Operation	R[b] = R[b] + zero_ext(FPI[R[a]])
	Flags	N, Z, V, C
ADD.PI	Syntax	**ADD.PI Ra, [#offset6]**
	Description	Add the contents of the PRAM location specified by the addition of contents of the PRAM Data Pointer, shifted left by six bits, and the zero-extended 6-bit value offset6 to the contents of register Ra; place the result in Ra.
	Operation	R[a] = R[a] + PRAM[(DPTR<<6) + zero_ext(#offset6)]
	Flags	N, Z, V, C

10.18.6 AND, 32-bit Logical AND

This section describes the AND instructions of the PCP.

AND	Syntax	**AND Rb, Ra, cc_A**
	Description	If the condition CONDCA is true, then perform a bit-wise logical AND of the contents of register Ra and the contents of register Rb; place the result in Rb. If CONDCA is false, no operation is performed.
	Operation	if (CONDCA = True) then R[b] = R[b] AND R[a] else NOP
	Flags	N, Z
AND.F	Syntax	**AND.F Rb, [Ra], Size**
	Description	Perform a bit-wise logical AND of the contents of the address location, specified by the contents of register Ra, and the contents of register Rb; place the result in Rb.
	Operation	R[b] = R[b] AND zero_ext(FPI[R[a]])
	Flags	N, Z
AND.PI	Syntax	**AND.PI Ra, [#offset6]**
	Description	Perform a bit-wise logical AND of the contents of the PRAM location specified by the addition of contents of the PRAM Data Pointer, shifted left by six bits, and the zero-extended 6-bit value offset6, and the contents of register Ra; place the result in Ra.
	Operation	R[a] = R[a] AND PRAM[(DPTR<<6) + zero_ext(#offset6)]
	Flags	N, Z

10.18.7 BCOPY, DMA Operation

This section describes the BCOPY instruction of the PCP in the TC1797.

BCOPY	Syntax	**BCOPY DST+-, SRC+-, CNC, CNT0**
	Description	Allows the PCP to perform DMA type transfers using FPI block transfers. Moves the contents of FPI Bus source location to FPI Bus destination location. Source location is pointed to by the contents of register R4; destination location is pointed to by the contents of register R5. Options (see also **Table 10-14** at **Page 10-81**): Source pointer (SRC+-): Increment, decrement or unchanged Destination pointer (SRC+-): Increment, decrement or unchanged Counter control (CNC): see **Table 10-14** Block size value (CNT0): see **Table 10-14**
	Operation	temp = zero_ext(FPI[R[4]]); value loaded and extended depending on SIZE FPI(R[5]) = temp R4 = R4 +/- n; n depending on SRC+- and CNT0 R5 = R5 +/- n; n depending on DST+- and CNT0 For counter operation see **Figure 10 - 2** on **Page 10-85** and **Table 10-14** on **Page 10-81**.
	Flags	CN1Z

See also **Page 10-139** for TC1797 specific details of the BCOPY instruction.

10.18.8 CHKB, Check Bit

This section describes the CHKB instruction of the PCP.

CHKB	Syntax	**CHKB Ra, #imm5, S/C**
	Description	If bit imm5 of register Ra is equal to the specified test value S/C then set the carry flag R7.C, else clear the carry flag.
	Operation	if (R[a][imm5] = S/C) then R7_C = 1 else R7_C = 0
	Flags	C

10.18.9 CLR, Clear Bit

This section describes the CLR instructions of the PCP.

CLR	Syntax	**CLR Ra, #imm5**
	Description	Clear bit imm5 of register Ra to 0.
	Operation	R[a][imm5] = 0
	Flags	None
CLR.F	Syntax	**CLR.F [Ra], #imm5, Size**
	Description	Clear bit imm5 of the address location specified through the contents of register Ra to 0. This instruction is executed using a locked read-modify-write FPI Bus transaction.
	Operation	FPI[(R[a])][imm5] = 0
	Flags	None

10.18.10 COMP, 32-bit Compare

This section describes the COMP instructions of the PCP.

COMP	Syntax	**COMP Rb, Ra, cc_A**
	Description	If the condition CONDCA is true, then subtract the contents of register Ra from the contents of register Rb; set the flags in register R7 according to the result of the subtraction; discard the subtraction result. If CONDCA is false, no operation is performed.
	Operation	if (CONDCA = True) then R7_FLAGS = Flags(R[b] - R[a])
	Flags	N, Z, V, C
COMP.I	Syntax	**COMP.I Ra, #imm6**
	Description	Subtract the immediate value imm6 from the contents of register Ra; set the flags in register R7 according to the result of the subtraction; discard the subtraction result.
	Operation	R7_FLAGS = Flags(R[a] - zero_ext(imm6))
	Flags	N, Z, V, C
COMP.F	Syntax	**COMP.F Rb, [Ra], Size**
	Description	Subtract the contents of the address location specified by the contents of register Ra from the contents of register Rb; set the flags in register R7 according to the result of the subtraction; discard the subtraction result.
	Operation	R7_FLAGS = Flags(R[b] - zero_ext(FPI[R[a]]))
	Flags	N, Z, V, C
COMP.PI	Syntax	**COMP.PI Ra, [#offset6]**
	Description	Subtract the contents of the PRAM location specified by the addition of contents of the PRAM Data Pointer, shifted left by six bits, and the zero-extended 6-bit value offset6, from the contents of register Ra; set the flags in register R7 according to the result of the subtraction; discard the subtraction result.
	Operation	R7_FLAGS = Flags(R[a] - PRAM[(DPTR<<6) + zero_ext(#offset6)])
	Flags	N, Z, V, C

10.18.11 COPY, DMA Instruction

This section describes the COMP instruction of the PCP.

COPY	Syntax	**COPY DST+-, SRC+-, CNC, CNT0, SIZE**
	Description	Moves the contents of FPI Bus source location to FPI Bus destination location. Source location is pointed to by the contents of register R4; destination location is pointed to by the contents of register R5. Options (see also **Table 10-14** on **Page 10-81**): Source pointer (SRC+-): Increment, decrement or unchanged Destination pointer (SRC+-): Increment, decrement or unchanged Counter control (CNC): see **Table 10-14** Counter 0 reload value (CNT0): see **Table 10-14** Data transfer width (SIZE): byte, half-word, word (pointers are incremented/decremented based upon SIZE).
	Operation	temp = zero_ext(FPI[R[4]]); value loaded and extended depending on SIZE FPI(R[5]) = temp R4 = R4 +/- n; n depending on SRC+- and SIZE R5 = R5 +/- n; n depending on DST+- and SIZE For counter operation see **Figure 10 - 1** on **Page 10-84** and **Table 10-14** on **Page 10-81**.
	Flags	CN1Z

10.18.12 DEBUG, Debug Instruction

This section describes the DEBUG instruction of the PCP.

DEBUG	Syntax	**DEBUG EDA, DAC, RTA, SDB, cc_B**
	Description	Conditionally cause a debug event if condition CONDCB is true. Optionally stop channel execution (SDB = 1) and/or generate an external debug event (EDA = 1).

	Operation	if (CONDCB = True) then
		  if (EDA = 1) then activate BRK_OUT pin
		  if (SDB = 1) then
		  if (RTA = 0) then
		  R7_CEN = 0; disable further channel invocation
		  else
		  PC = PC - 1
		  endif
		  save_context
		  idle
		  endif
		  if (DAC = 1)
		  PCP_CS>EN = 0
		  endif
		  set ES.DBE; indicate debug event
		  ES.PC = NextPC
		  ES.PN = channel_number
		endif

	Flags	none

Note: Execution of the DEBUG command when not in Debug Mode will cause the PCP to generate an “Illegal Operation” Error Exit.

10.18.13 DINIT, Divide Initialization

This section describes the DINIT instruction of the PCP.

DINIT	Syntax	**DINIT <R0>, Rb, Ra**
	Description	Initialize Divide logic ready for divide sequence (Rb / Ra) and Clear R0. If value of Ra is 0 then set V (to flag divide by 0 error); otherwise, clear V. If value of Rb is 0 and value of Ra is not 0 then set Z (to flag a zero result); otherwise, clear Z.
	Operation	R0 = 0
	Flags	Z, V

10.18.14 DSTEP, Divide Instruction

This section describes the DSTEP instruction of the PCP.

DSTEP	Syntax	**DSTEP <R0>, Rb, Ra**
	Description	Perform 1 step (eight bits) of an unsigned 32- by 32-bit divide (Rb / Ra). Shift R0 left by 8 bits, copy the most significant byte of Rb into LS byte of R0. Shift Rb left by 8 bits and add (R0 divided by Ra). Load R0 with (the remainder of R0 divided by Ra).
	Operation	R0 = (R0 << 8) + (Rb >> 24) Rb = (Rb << 8) + R0 / Ra R0 = R0 % Ra
	Flags	Z

Note: The value in Ra must always be greater than the value in R0 prior to execution of the DSTEP instruction. If the rules specified on ***Page 10-86*** *are followed, then the above description and operation are correct. Failure to adhere to these rules will yield undefined results.*

10.18.15 EXIT, Exit Instruction

This section describes the EXIT instruction of the PCP.

EXIT	Syntax	**EXIT EC, ST, INT, EP, cc_B**
	Description	Unconditionally exit channel program execution. Optionally decrement counter CNT1 (EC = 1), disable further channel invocation (ST = 1), generate an interrupt request (INT = 1) if condition CONDCB is true. Field EP is used to set the channel code entry point in Channel Resume Mode to either the address of the next instruction (EP = 1) or to the start address of the channel (EP = 0). The EXIT instruction is finished with a context save operation. The EP option is only in effect when Channel Resume operation is globally selected through PCP_CS.RCB = 0. If PCP_CS.RCB = 1, Channel restart mode is selected for all channels, and the EP field of the EXIT instruction is disregarded.

	Operation	if (EC = 1) then CNT1 = CNT1 - 1
		if (ST = 1) then R7_CEN = 0
		if ((INT = 1) AND (cc_B = True)) then activate_interrupt_request
		if (EP = 1) then R7_PC = NextPC else R7_PC = channel_entry_point
		save_context

	Flags	CN1Z

10.18.16 INB, Insert Bit

This section describes the INB instructions of the PCP.

INB	Syntax	**INB Rb, Ra, cc_A**
	Description	If CONDCA is true, then insert the carry flag R7.C into register Rb at the bit position specified through bits [4..0] of register Ra. If CONDCA is false, no operation is performed.
	Operation	if (CONDCA = True) then R[b][R[a][4:0]] = R7_C else NOP
	Flags	None
INB.I	Syntax	**INB.I Ra, #imm5**
	Description	Insert the carry flag R7.C into register Ra at the bit position specified through the immediate value imm5.
	Operation	R[a][imm5] = R7_C
	Flags	None

10.18.17 JC, Jump Conditionally

This section describes the conditional jump instructions of the PCP.

JC	Syntax	**JC offset6, cc_B**
	Description	If CONDCB is true, then add the sign-extended value specified by offset6 to the contents of the PC, and jump to that address. If CONDCB is false, no operation is performed.
	Operation	if (CONDCB = True) then (PC = PC + sign_ext(offset6)) else NOP
	Flags	None
JC.A	Syntax	**JC.A #address16, cc_B**
	Description	If CONDCB is true, then load the value specified by address16 into the PC, and jump to that address. If CONDCB is false, no operation is performed.
	Operation	if (CONDCB = True) then (PC = address16) else NOP
	Flags	None
JC.I	Syntax	**JC.I Ra, cc_B**
	Description	If CONDCB is true, then add the value specified by Ra[15:0] to the contents of the PC, and jump to that address. Value Ra[15:0] is treated as a signed 16-bit number. If CONDCB is false, no operation is performed.
	Operation	if (CONDCB = True) then (PC = PC + (R[a][15:0])) else NOP
	Flags	None
JC.IA	Syntax	**JC.IA Ra, cc_B**
	Description	If CONDCB is true, then load the value specified by Ra[15:0] into the PC, and jump to that address. Value Ra[15:0] is treated as an unsigned 16-bit number. If CONDCB is false, no operation is performed.
	Operation	if (CONDCB = True) then (PC = (R[a][15:0])) else NOP
	Flags	None

10.18.18 JL, Jump Long Unconditional

This section describes the long jump instruction JL of the PCP.

JL	Syntax	**JL offset10**
	Description	Add the sign-extended value specified by offset10 to the contents of the PC, and jump to that address.
	Operation	PC = PC + sign_ext(offset10)
	Flags	None

10.18.19 LD, Load

This section describes the LD instructions of the PCP.

LD.F	Syntax	**LD.F Rb, [Ra], Size**
	Description	Load the zero-extended contents of the address location specified by the contents of register Ra into register Rb.
	Operation	R[b] = zero_ext(FPI[R[a]])
	Flags	N, Z
LD.I	Syntax	**LD.I Ra, #imm6**
	Description	Load the zero-extended value specified by imm6 into register Ra.
	Operation	R[a] = zero_ext(imm6)
	Flags	N, Z
LD.IF	Syntax	**LD.IF [Ra], #offset5, Size**
	Description	Load the zero-extended contents of the address location, specified by the addition of the contents of register Ra and the value specified by imm5, into register R0.
	Operation	R[0] = zero_ext(FPI[R[a] + zero_ext(imm5)])
	Flags	N, Z

LD.P	Syntax	**LD.P Rb, [Ra], cc_A**
	Description	If condition CONDCA is true, then load the contents of the PRAM address location, specified by the addition of contents of the PRAM Data Pointer, shifted left by six bits, and the zero-extended 6-bit value Ra[5:0] into register Rb. If condition CONDCA is false, no operation is performed.
	Operation	if (CONDCA = True) then R[b] = PRAM[(DPTR<<6) + zero_ext(R[a][5:0])] else NOP
	Flags	N, Z
LD.PI	Syntax	**LD.PI Ra, [#offset6]**
	Description	Load the contents of the PRAM location specified by the addition of contents of the PRAM Data Pointer, shifted left by six bits, and the zero-extended 6-bit value offset6 into register Ra.
	Operation	R[a] = PRAM[(DPTR<<6) + zero_ext(offset6)]
	Flags	N, Z

10.18.20 LDL, Load 16-bit Value

This section describes the LDL instructions of the PCP.

LDL.IL	Syntax	**LDL.IL Ra, #imm16**
	Description	Load the immediate value imm16 into the lower bits of register Ra (bits [15:0]). Bits [31:16] of register Ra are unaffected. Value imm16 is treated as an unsigned 16-bit number.
	Operation	R[a][15:0] = imm16
	Flags	N, Z
LDL.IU	Syntax	**LDL.IU Ra, #imm16**
	Description	Load the immediate value imm16 into the upper bits of register Ra (bits [31:16]). Bits [15:0] of register Ra are unaffected.
	Operation	R[a][31:16] = imm16
	Flags	N, Z

10.18.21 MINIT, Multiply Initialization

This section describes the MINIT instruction of the PCP.

MINIT	Syntax	**MINIT <R0>, Rb, Ra**
	Description	Initialize Multiply logic ready for multiply sequence. Clear R0. If value of Ra is zero or value of Rb is zero then set Z (to flag zero result) else clear Z.
	Operation	R0 = 0
	Flags	Z

10.18.22 MOV, Move Register to Register

This section describes the MOV instruction of the PCP.

MOV	Syntax	**MOV Rb, Ra, cc_A**
	Description	If condition CONDCA is true, then move the contents of register Ra into register Rb. If CONDCA is false, no operation is performed.
	Operation	if (CONDCA = True) then R[b] = R[a] else NOP
	Flags	N, Z

10.18.23 Multiply Instructions

This section describes the multiply instructions of the PCP.

MSTEP32	Syntax	**MSTEP32 <R0>, Rb, Ra**
	Description	Perform an unsigned multiply step, using eight bits of data taken from Rb, keeping the least significant 32 bits of a potential 64-bit result. Left rotate Rb by 8 bits. Shift R0 left by 8 bits. Add (Ra multiplied by the least significant 8 bits of Rb) to R0. If value of R0 is zero then set Z (to signal zero result) else clear Z.
	Operation	Rb = (Rb << 8) + (Rb >> 24) R0 = (R0 << 8) + (Rb & 0xff) × Ra
	Flags	Z
MSTEP64	Syntax	**MSTEP64 <R0>, Rb, Ra**
	Description	Perform an unsigned multiply step, using eight bits of data taken from Rb, keeping 40 bits of a potential 64-bit result. Add (Ra multiplied by the least significant 8 bits of Rb) to R0 and retain the 40 bit result (shown as temp below). Store the most significant 32 bits of the result (temp) in R0. Shift Rb right by 8 bits. Store the least significant 8 bits of the first result (temp) in the most significant 8 bits of Rb. If value of R0 is zero then set Z (to signal zero result) else clear Z.
	Operation	temp = R0 + Ra × (Rb & 0xff) R0 = temp >> 8 Rb = (Rb >> 8) + ((temp & 0xff) << 24)
	Flags	Z

Note: In the case of the MSTEP64 instruction above, the “temp” variable is a 40-bit variable and all calculations are performed using 40-bit unsigned arithmetic. All other calculations use 32-bit unsigned arithmetic.

10.18.24 NEG, Negate

This section describes the NEG instruction of the PCP.

NEG	Syntax	**NEG Rb, Ra, cc_A**
	Description	If condition CONDCA is true, then move the 2’s complement of the contents of register Ra into register Rb. If CONDCA is false, no operation is performed.
	Operation	if (CONDCA = True) then R[b] = (- R[a]) else NOP
	Flags	N, Z, V, C

10.18.25 NOP, No Operation

This section describes the NOP instruction of the PCP.

NOP	Syntax	**NOP**
	Description	No operation. The NOP instruction puts the PCP in low-power operation.
	Operation	no operation
	Flags	None

10.18.26 NOT, Logical NOT

This section describes the NOT instruction of the PCP.

NOT	Syntax	**NOT Rb, Ra, cc_A**
	Description	If condition CONDCA is true, then move the 1’s complement of the contents of register Ra into register Rb. If CONDCA is false, no operation is performed.
	Operation	if (CONDCA = True) then R[b] = NOT(R[a]) else NOP
	Flags	N, Z

10.18.27 OR, Logical OR

This section describes the OR instructions of the PCP.

OR	Syntax	**OR Rb, Ra, cc_A**
	Description	If the condition CONDCA is true, then perform a bit-wise logical OR of the contents of register Ra and the contents of register Rb; place the result in Rb. If CONDCA is false, no operation is performed.
	Operation	if (CONDCA = True) then R[b] = R[b] OR R[a] else NOP
	Flags	N, Z
OR.F	Syntax	**OR.F Rb, [Ra], Size**
	Description	Perform a bit-wise logical OR of the contents of the address location, specified by the contents of register Ra, and the contents of register Rb; place the result in Rb.
	Operation	R[b] = R[b] OR zero_ext(FPI[R[a]])
	Flags	N, Z
OR.PI	Syntax	**OR.PI Ra, [#offset6]**
	Description	Perform a bit-wise logical OR of the contents of the PRAM location specified by the addition of contents of the PRAM Data Pointer, shifted left by six bits, and the zero-extended 6-bit value offset6, and the contents of register Ra; place the result in Ra.
	Operation	R[a] = R[a] OR PRAM[(DPTR<<6) + zero_ext(#offset6)]
	Flags	N, Z

10.18.28 PRAM Bit Operations

This section describes the MCLR and MSET instructions of the PCP.

MCLR	Syntax	**MCLR.PI Ra, [#offset6]**
	Description	Perform an ‘AND’ of the contents of the specified register with the contents of the PRAM location specified by the addition of contents of the PRAM Data Pointer, shifted left by six bits, and the zero-extended 6-bit value offset. Write the result back to the PRAM location.
	Operation	R[a] = R[a].AND.PRAM[DPTR<<6 + #offset6] PRAM[DPTR<<6 + #offset6] = R[a]
	Flags	N, Z
MSET	Syntax	**MSET.PI Ra, [#offset6]**
	Description	Perform an ‘OR’ of the contents of the specified register with the contents of the PRAM location specified by the addition of contents of the PRAM Data Pointer, shifted left by six bits, and the zero-extended 6-bit value offset. Write the result back to the PRAM location.
	Operation	R[a] = R[a].OR.PRAM[DPTR<<6 + #offset6] PRAM[DPTR<<6 + #offset6] = R[a]
	Flags	N, Z

Note: MCLR and MSET are read/modify/write operations that cannot be interrupted by an access from another FPI master. They can be used to implement semaphore systems.

10.18.29 PRI, Prioritize

This section describes the PRI instruction of the PCP.

PRId	Syntax	**PRI Rb, Ra, cc_A**
	Description	If condition CONDCA is true, then find the bit position of the most significant 1 in register Ra and put the number into register Rb. The bit location, 31-0, is encoded as a 5-bit number stored in Rb[4:0]. If the contents of Ra is zero, bit Rb[5] is set, while all other bits in Rb are cleared. If CONDCA is false, no operation is performed.

	Operation	if (CONDCA = False) then
		  NOP
		else
		  if (R[a] = 0) then
		  R[b] = 0x20
		  else
		  R[b] = bit_pos(most_significant_1(R[a]))

	Flags	N, Z

10.18.30 RL, Rotate Left

This section describes the RL instruction of the PCP.

RL	Syntax	**RL Ra, #imm5**
	Description	Rotate the contents of register Ra to the left by the number of bit positions specified through the 5-bit value imm5. The values defined for imm5 are 1, 2, 4 and 8. The carry flag, R7.C, is set to the last bit shifted out of bit 31 of register Ra.
	Operation	tmp = R[a] R[a] = R[a] << imm5; imm5 = 1, 2, 4, 8 R7_C = last bit shifted out of R[a] tmp = tmp >> 32 - imm5 R[a] = tmp OR R[a]
	Flags	N, Z

10.18.31 RR, Rotate Right

This section describes the RR instruction of the PCP.

RR	Syntax	**RR Ra, #imm5**
	Description	Rotate the contents of register Ra to the right by the number of bit positions specified through the 5-bit value imm5. The values allowed for imm5 are 1, 2, 4 and 8.
	Operation	tmp = R[a] R[a] = R[a] >> imm5; imm5 = 1, 2, 4, 8 tmp = tmp << 32 - imm5 R[a] = tmp OR R[a]
	Flags	N, Z

10.18.32 SET, Set Bit

This section describes the SET bit instruction of the PCP.

SET	Syntax	**SET Ra, #imm5**
	Description	Set bit imm5 of register Ra to 1.
	Operation	R[a][imm5] = 1
	Flags	None
SET.F	Syntax	**SET.F [Ra], #imm5, Size**
	Description	Set bit imm5 of the address location specified through the contents of register Ra to 1. This instruction is executed using a locked read-modify-write FPI Bus transaction.
	Operation	FPI[(R[a])][imm5] = 1
	Flags	None

10.18.33 SHL, Shift Left

This section describes the SHL instruction of the PCP.

SHL	Syntax	**SHL Ra, #imm5**
	Description	Shift the contents of register Ra to the left by the number of bit positions specified through the 5-bit value imm5. The values allowed for imm5 are 1, 2, 4 and 8. The carry flag, R7.C, is set to the last bit shifted out of bit 31 of register Ra. Zeros are shifted in from right.
	Operation	R[a] = R[a] << imm5; imm5 = 1, 2, 4, 8 R7_C = last bit shifted out of R[a]
	Flags	N, Z, C

10.18.34 SHR, Shift Right

This section describes the SHR instruction of the PCP.

SHR	Syntax	**SHR Ra, #imm5**
	Description	Shift the contents of register Ra to the right by the number of bit positions specified through the 5-bit value imm5. The values allowed for imm5 are 1, 2, 4 and 8. Zeros are shifted in from left.
	Operation	R[a] = R[a] >> imm5; imm5 = 1, 2, 4, 8
	Flags	N, Z

10.18.35 ST, Store

This section describes the ST instructions of the PCP.

ST.F	Syntax	**ST.F Rb, [Ra], Size**
	Description	Store the contents of register Rb to the address location specified by the contents of register Ra. When the Size is byte or half-word, the data is stored with the internal LSB (bit 0) properly aligned to the correct FPI Bus byte or half-word lane.
	Operation	FPI[R[a]] = R[b]
	Flags	None
ST.IF	Syntax	**ST.IF [Ra], #offset5, Size**
	Description	Store the contents of R0 to the address location specified by the addition of the contents of register Ra and the value specified by imm5. When the Size is byte or half-word, the data is stored with the internal LSB (bit 0) properly aligned to the correct FPI Bus byte or half-word lane.
	Operation	FPI[R[a] + zero_ext(imm5)] = R[0]
	Flags	None
ST.P	Syntax	**ST.P Rb, [Ra], cc_A**
	Description	If condition CONDCA is true, then store the contents of Rb to the PRAM address location specified by the addition of the contents of the PRAM Data Pointer, shifted left by six bits, and the zero-extended 6-bit value Ra[5:0]. If condition CONDCA is false, no operation is performed.
	Operation	if (CONDCA = True) then PRAM[(DPTR<<6) + zero_ext(R[a][5:0])] = R[b] else NOP
	Flags	None
ST.PI	Syntax	**ST.PI Rb, [#offset6]**
	Description	Store the contents of register Rb to the PRAM location specified by the addition of the contents of the PRAM Data Pointer, shifted left by six bits, and the zero-extended 6-bit value offset6.
	Operation	PRAM[(DPTR<<6) + zero_ext(offset6)] = R[b]
	Flags	None

10.18.36 SUB, 32-bit Subtract

This section describes the SUB instructions of the PCP.

SUB	Syntax	**SUB Rb, Ra, cc_A**
	Description	If the condition CONDCA is true, then subtract the contents of register Ra from the contents of register Rb; place the result in Rb. If CONDCA is false, no operation is performed.
	Operation	if (CONDCA = True) then R[b] = R[b] - R[a] else NOP
	Flags	N, Z, V, C
SUB.I	Syntax	**SUB.I Ra, #imm6**
	Description	Subtract the zero-extended immediate value imm6 from the contents of register Ra; place the result in Ra.
	Operation	R[a] = R[a] - zero_ext(imm6)
	Flags	N, Z, V, C
SUB.F	Syntax	**SUB.F Rb, [Ra], Size**
	Description	Subtract the zero-extended contents of the address location specified by the contents of register Ra from the contents of register Rb; place the result in Rb.
	Operation	R[b] = R[b] - zero_ext(FPI[R[a]])
	Flags	N, Z, V, C
SUB.PI	Syntax	**SUB.PI Ra, [#offset6]**
	Description	Subtract the contents of the PRAM location specified by the addition of contents of the PRAM Data Pointer, shifted left by six bits, and the zero-extended 6-bit value offset6 from the contents of register Ra; place the result in Ra.
	Operation	R[a] = R[a] - PRAM[(DPTR<<6) + zero_ext(#offset6)]
	Flags	N, Z, V, C

10.18.37 XCH, Exchange

This section describes the XCH instructions of the PCP.

XCH.F	Syntax	**XCH.F Rb, [Ra], Size**
	Description	Exchange contents of R[b] and FPI[R[a]] when Size is byte or half-word, the value is stored with the internal LSB (bit 0) properly aligned to the correct FPI byte or half-word lane. The exchange is done via a locked FPI bus transfer.
	Operation	temp = R[b] R[b] = zero_ext(FPI[R[a]]) FPI[R[a]] = temp
	Flags	N, Z
XCH.PI	Syntax	**XCH.PI Ra, [#offset6]**
	Description	Exchange contents of R[a] and PRAM[DPTR <<6 + #offset6]. *Note: The exchange is un-interruptible, and locks out external accesses; it will not be interrupted by any external FPI bus master transfer requests.*
	Operation	temp = R[a] R[a] = PRAM[(DPTR<<6) + zero_ext(#offset6)] PRAM[(DPTR<<6) + zero_ext(#offset6)] = temp
	Flags	N, Z

10.18.38 XOR, 32-bit Logical Exclusive OR

This section describes the XOR instructions of the PCP.

XOR	Syntax	**XOR Rb, Ra, cc_A**
	Description	If the condition CONDCA is true, then perform a bit-wise logical Exclusive-OR of the contents of register Ra and the contents of register Rb; place the result in Rb. If CONDCA is false, no operation is performed.
	Operation	if (CONDCA = True) then R[b] = R[b] XOR R[a] else NOP
	Flags	N, Z
XOR.F	Syntax	**XOR.F Rb, [Ra], Size**
	Description	Perform a bit-wise logical Exclusive-OR of the contents of the address location, specified by the contents of register Ra, and the contents of register Rb; place the result in Rb.
	Operation	R[b] = R[b] XOR zero_ext(FPI[R[a]])
	Flags	N, Z
XOR.PI	Syntax	**XOR.PI Ra, [#offset6]**
	Description	Perform a bit-wise logical Exclusive-OR of the contents of the PRAM location specified by the addition of contents of the PRAM Data Pointer, shifted left by six bits, and the zero-extended 6-bit value offset6, and the contents of register Ra; place the result in Ra.
	Operation	R[a] = R[a] XOR PRAM[(DPTR<<6) + zero_ext(#offset6)]
	Flags	N, Z

10.18.39 Flag Updates of Instructions

Most instructions update the state flags in R7. In **Table 10-15**, each instruction is shown with the flags that it updates.

Table 10-15 Flag Updates

Instruction	CN1Z	V	C	N	Z
ADD	–	yes	yes	yes	yes
AND	–	–	–	yes	yes
BCOPY	yes[1)]	–	–	–	–
CHKB	–	–	yes	–	–
CLR	–	–	–	–	–
COMP	–	yes	yes	yes	yes
COPY	yes[1)]	–	–	–	–
DEBUG	–	–	–	–	–
DINIT	–	yes	–	–	yes
DSTEP	–	–	–	–	yes
EXIT	yes[1)]	–	–	–	–
INB	–	–	–	–	–
JC	–	–	–	–	–
JL	–	–	–	–	–
LD	–	–	–	yes[2)]	yes
LDL	–	–	–	yes	yes
MCLR	–	–	–	yes	yes
MSET	–	–	–	yes	yes
MOV	–	–	–	yes	yes
NEG	–	yes	yes	yes	yes
NOP	–	–	–	–	–
NOT	–	–	–	yes	yes
OR	–	–	–	yes	yes
PRI	–	–	–	yes[3)]	yes
RR	–	–	–	yes	yes
RL	–	–	yes	yes	yes
SET	–	–	–	–	–

Table 10-15 Flag Updates

Instruction	CN1Z	V	C	N	Z
SHR	–	–	–	yes[3)]	yes
SHL	–	–	yes	yes	yes
ST	–	–	–	–	–
SUB	–	yes	yes	yes	yes
XCH	–	–	–	yes	yes
XOR	–	–	–	yes	yes

1) CN1Z is only modified by the BCOPY, COPY or EXIT instructions if the instruction has been configured to decrement R6.CNT1 (for BCOPY/COPY CNC = 1 or CNC = 2, for EXIT EC = 1). All other instructions have no effect on the CN1Z flag.

2) For the LD.I type of instruction, flag N is always cleared, as bit 31 of the result is always 0.

3) Flag N is always cleared, as bit 31 of the result is always 0.

10.18.40 Instruction Timing

This section gives some information about the duration of PCP instructions. Please note that there are various conditions that can further affect the duration of PCP instructions (e.g. external FPI accesses from another FPI Bus master to the PCP memories stall the PCP Processor Core).

*Note: The clock cycles listed in **Table 10-16** are PCP core clock cycles. When running in 2:1 clocking mode the maximum allowed PCP core clock frequency is 180 MHz (resulting in a minimum PCP core clock cycle time of 5.6ns). When running in 1:1 clocking mode the maximum PCP core clock frequency is the same as the maximum System Peripheral Bus frequency. In the TC1797 the System Peripheral Bus (which is an FPI Bus) is clocked with f_{SYS}, resulting in a minimum PCP core clock cycle time of 11.1ns (in 1:1 clocking mode).*

*Note: Where an execution time is stated for an FPI instruction this is always a **minimum** value. A number of FPI instructions must wait for the completion of an FPI transaction (or transactions). When running in 2:1 mode each FPI clock cycle consists of two core clock frequencies. Where this applies the cycle count is shown with an "FPI" superscript designation (e.g. "3^{FPI}"). In addition there may be an additional single core clock cycle taken according to alignment of execution of an FPI instruction relative to an FPI clock edge.*

Table 10-16 Instruction Timing

Instruction	Number of Clock Cycles	Comments	Notes
Control			
NOP	1	–	–

Table 10-16 Instruction Timing

Instruction	Number of Clock Cycles	Comments	Notes
COPY	–	–	1)
EXIT	f = 9, s = 7, m = 6	–	2)
BCOPY	–	–	1)
FPI Access			
ADD.F	8 min.	5 int. + 3^{FPI} min. for FPI read	3)
AND.F	8 min.	5 int. + 3^{FPI} min. for FPI read	3)
COMP.F	8 min.	5 int. + 3^{FPI} min. for FPI read	3)
LD.F	8 min.	5 int. + 3^{FPI} min. for FPI read	3)
OR.F	8 min.	5 int. + 3^{FPI} min. for FPI read	3)
ST.F	5 min.	2 int. + 3^{FPI} min. for FPI write	3)
SUB.F	8 min.	5 int. + 3^{FPI} min. for FPI read	3)
XCH.F	8 min.	4 int. + 4^{FPI} min. for FPI read and write	3)
XOR.F	8 min.	5 int. + 3^{FPI} min. for FPI read	3)
PRAM Access			
ADD.PI	2	–	–
AND.PI	2	–	–
COMP.PI	2	–	–
LD.PI	2	–	–
MCLR.PI	6	–	–
MSET.PI	6	–	–
OR.PI	2	–	–
ST.PI	4	–	–
SUB.PI	2	–	–
XCH.PI	5	–	–
XOR.PI	2	–	–
Arithmetic (Conditional)			
ADD	1	–	–
AND	1	–	–
COMP	1	–	–
INB	1	–	–

Table 10-16 Instruction Timing

Instruction	Number of Clock Cycles	Comments	Notes
LD.P	2	–	–
MOV	1	–	–
NEG	1	–	–
NOT	1	–	–
OR	1	–	–
PRI	1	–	–
ST.P	4	–	–
SUB	1	–	–
XOR	1	–	–
Immediate Access			
ADD.I	1	–	–
CHKB	1	–	–
CLR	1	–	–
COMP.I	1	–	–
INB.I	1	–	–
LD.I	1	–	–
LDL.IL	1	–	–
LDL.IU	1	–	–
RL	1	–	–
RR	1	–	–
SET	1	–	–
SHL	1	–	–
SHR	1	–	–
SUB.I	1	–	–
FPI + Immediate Access			
CLR.F	8 min.	4 int. + 4^{FPI} min. for locked FPI RMW (Read, Modify, Write)	4)
LD.IF	8 min.	5 int. + 3^{FPI} min. for FPI read	3)
SET.F	8 min.	4 int. + 4^{FPI} min. for locked FPI RMW (Read, Modify, Write)	4)
ST.IF	5 min.	2 int. + 3^{FPI} min. for FPI write	5)

Table 10-16 Instruction Timing

Instruction	Number of Clock Cycles	Comments	Notes
Complex Math			
DINIT	1	–	6)
DSTEP	10	–	6)
MINIT	1	–	7)
MSTEP.L	10	–	7)
MSTEP.U	11	–	7)
Jump			
DEBUG	sdb - 0 = 2 sdb - 1 = exit_time	–	8)
JC	y = 4, n = 2	–	9)
JC.A	y = 4, n = 2	–	8)
JC.I	y = 4, n = 2	–	8)
JC.IA	y = 4, n = 2	–	8)
JL	4	–	–

1) The number of clock cycles for these instructions depends on several parameters such as the amount of data to be copied, the type of memory, and the effective bus load.

2) f = Full Context save, s = Small Context save, m = Minimum Context save time extended until any previous ST.F instruction has completed.

3) Cycles = 5 internal + 3 minimum for FPI read (with 0 wait cycles + 1 cycle for bus arbitration and assuming 1:1 clocking mode)

4) Cycles = 4 internal + 4 minimum for FPI read/modify/write (with 0 wait cycles + 1 cycle bus arbitration and assuming 1:1 clocking mode)

5) Cycles = 2 internal + 3 minimum for FPI write (with 0 wait cycles + 1 cycle for bus arbitration and assuming 1:1 clocking mode)
Time starts after any previous ST.F instruction has completed.

6) 32/32 bit divide requires instruction DINIT + JC + 4 × DSTEP = 1 + 4(2) + 4 × 10 = 45 cycles
8/32 bit divide requires instruction RR + DINIT + JC + DSTEP = 1 + 1 + 4(2) + 10 = 16 cycles

7) 32 × 8 bit multiply requires instructions RR + MINIT + MSTEP.L = 1 + 1 + 10 = 12 cycles
32 × 16 bit multiply requires instructions 2 × RR + MINIT + 2 × MSTEP.L = 2 × 1 + 1 + 2 × 10 = 23 cycles
32 × 32 bit multiply requires instructions MINIT + 4 × MSTEP.U= 1 + 4 × 11 = 45 cycles

8) sdb - 0 = Stop_on_Debug bit in instruction = 0 (disabling stop)
sdb - 1 = Stop_on_Debug bit in instruction = 1 (enabling stop)
exit_time = same time as for an exit instruction

9) y = jump taken, n = jump not taken

10.19 Instruction Encoding

Most instructions are encoded in 16 bits.

This allows two instruction to be fetched out of 32 bit instruction memory per access.

For example, a COPY and an EXIT instruction can be fetched simultaneously, performing a simple DMA transaction.

Table 10-17 Field Definitions

Symbol	Name	Description
CNC	Counter Control	00: Perform the number/type of transfers appropriate to the instruction, and proceed to the next instruction. 01: Perform the number/type of transfers appropriate to the instruction, then decrement CNT1 and proceed to next instruction. 10: Use CNT1 as an “outer loop counter”. Perform the number/type of transfers appropriate to the instruction, then decrement CNT1. Repeat until CNT1 = 0, then proceed to next instruction. If R[7].IEN = True (interrupts enabled) the instruction may be interrupted at the end of each iteration of the “outer loop”. 11: N/A (error)
CNT0	Internal Loop Counter /Block Size Control	Internal Loop Counter for COPY: 000: Perform a sequence of 8 read/write transfers n: Perform a sequence of n (1 <= n <= 7) read/write transfers Block Size control for BCOPY: 000: 8 Words 001: N/A (error) 010: 2 Words 011: 4 Words 1xx: N/A (error)
CONDCA/B	Condition Code	Condition Code for conditional execution of instruction.
DAC	Disable All Channels Control	0: no action 1: Clear CS.EN which stops the PCP executing and further channel programs.
DST+/-	Destination Address Increment / Decrement	00: No Change 01: Post Increment by Size 10: Post Decrement by Size 11: N/A (error)

Table 10-17 Field Definitions

Symbol	Name	Description
EC	Exit Count Control	0: no action 1: decrement CNT1
EDA	External Debug Action	0: No External Debug Action caused 1: Cause an External Debug Action (breakpoint pin etc.)
EP	Entry Point	Entry Point on next Channel Invocation: 0: Set the PC to Channel Start 1: Set the PC to the address contained in NextPC (next instruction) address. Note: EP=0 assumes that a Channel Entry Table exists in the base of Code Memory. Failure to provide one will cause improper operation.
INT	Interrupt	Issue interrupt 0: No 1: Yes if (CONDCB = True)
RTA	Return to This Address	0: The channel is disabled (R7.CEN=0) and the PC value stored in the context is NextPC. 1: The channel remains enabled and the PC value stored in the context is NextPC - 1.
SDB	Stop on Debug	0: Continue running if Debug Event Triggered 1: Stop PCP Channel if Debug Event Triggered
Set/Clr	Set/Clear	0: Check for Clear (0) 1: Check for Set (1)
SIZE	Data Size	00: Byte (8-bit) 01: Half Word (16-bit) 10: Word (32-bit) 11: Reserved
SRC+/-	Source Address Increment / Decrement	00: No Change 01: Post Increment by Size 10: Post Decrement by Size 11: N/A (error)
ST	Stop Channel	0: Leave Channel enabled 1: Stop Channel from accepting new Service Requests (clear R[7].CEN)

Table 10-18 Instruction Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0 -:Control			fmt	Instr	DST + -		SRC + -		CNC		CNT0			Size	
NOP			0	0											
COPY			0	1											
					ST	INT	EP	EC	-		-	CONDC B			
EXIT			1	0								Condition for Interrupt			
			fmt	Instr	DST + -		SRC + -		CNC		-	CNT0		-	
BCOPY			1	1											

1 -:FPI	Instruction	R[b]	R[a]	-	Size
ADD.F	0				
SUB.F	1				
COMP.F	2				
error	3				
error	4				
AND.F	5				
error	6				
OR.F	7				
XOR.F	8				
LD.F	9				
ST.F	A				
XCH.F	B				
error	C				
error	D				
error	E				
error	F				

2 -:PRAM	Instruction	R[a]	Offset 6 - bit
ADD.PI	0		
SUB.PI	1		
COMP.PI	2		

Table 10-18 Instruction Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
error			3												
MCLR.PI			4												
AND.PI			5												
MSET.PI			6												
OR.PI			7												
XOR.PI			8												
LD.PI			9												
ST.PI			A												
XCH.PI			B												
error			C												
error			D												
error			E												
error			F												

3 -:Arithmetic	Instruction	R[b]	R[a]	CONDC A
ADD	0			
SUB	1			
COMP	2			
NEG	3			
NOT	4			
AND	5			
error	6			
OR	7			
XOR	8			
LD.P	9			
ST.P	A			
error	B			
MOV	C			
INB	D			
PRI	E			
error	F			

Table 10-18 Instruction Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
4 - Immediate			Instruction				R[a]			Immediate 6 - bit					
SUB.I			1												
ADD.I			0												
COMP.I			2												
error			3												
SHR			4												
SHL			5												
RR			6												
RL			7												
LDL.IU			8							following #imm16 instruction					
LDL.IL			9							following #imm16 instruction					
SET			A												
CLR			B												
LD.I			C												
INB.I			D												
CHKB			E							S/C					
error			F												
5 -:FPI Immediate			Instruction			Size1	R[a]			Size0	Immediate 5 - bit				
error			0												
error			1												
error			2												
SET.F			3												
CLR.F			4												
LD.IF			5												
ST.IF			6												
error			7												
6 -:Complex Maths			op2:Instruction				R[b]			R[a]			Reserved		
DINIT			0												
DSTEP			1												

Table 10-18 Instruction Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MINIT			2												
MSTEP.L			3												
MSTEP.U			4												
error			5												
error			6												
error			7												
error			8												
error			9												
error			A												
error			B												
error			C												
error			D												
error			E												
error			F												
7 -:JUMP			op2:Instr			Offset 10 - bit									
JL			0	0	0										
			op2:Instr			CONDC B				Offset 6 - bit					
JC			0	0	1										
JC.A			0	1	0					Absolute Destination in next 16 bits					
error			0	1	1										
			op2:Instr			CONDCB				R[a]			-	-	-
JC.I			1	0	0										
JC.IA			1	0	1										
error			1	1	0										
			op2:Instr			CONDCB				-	-	DAC	RTA	EDA	SDB
DEBUG			1	1	1										

10.20 Programming of the PCP

In this section, several techniques are outlined to help design channel programs. There are also examples on configuring a channel program's context.

10.20.1 Initial PC of a Channel Program

A channel program can begin operation at the Channel Entry Table location corresponding to the priority of the interrupt. This is much like an interrupt vector location for that channel in a traditional processor architecture. When the channel program is started, the PC is set to two-times the channel number (SRPN). Since the base of the Channel Entry Table is the bottom of the CMEM address range, and since each entry in the table is two instructions long, this address computation results in the first instruction of the channel program for that SRPN being fetched from memory for execution.

Alternately, the channel program can be made to begin executing at whatever address its restored context holds in R7.PC.

If PCP_CS.RCB = 1, then the channel program is forced to always start at its Channel Entry Table location regardless of the PC value stored in the CSA. If PCP_CS.RCB = 0, then the channel program will simply begin executing at whatever PC value is restored in the context R7.PC.

It is important to be aware of the implications of these two approaches on how CMEM should be configured, and what the initial value of the PC should be in the channel program's context that is loaded in the PRAM CSA at boot time.

10.20.1.1 Channel Entry Table

When PCP_CS.RCB = 1, the program counter of the PCP is vectored to the appropriate channel entry table each time a channel program is invoked by the receipt of an interrupt. The PCP is forced to start executing from its channel entry table location, regardless of its previous context or PC state.

If the EXIT instruction is executed with EP = 0, the PC saved during the context save operation will be the channel entry table location for that channel. That means that the next time the channel program is started, it will begin operation at the appropriate location in the Channel Entry Table.

Note: If EP = 0 is set in any channel program, or if PCP_CS.RCB = 1, a Channel Entry Table must be provided at the base of CMEM. Otherwise this table is not needed.

10.20.1.2 Channel Resume

When PCP_CS.RCB = 0, the program counter of the PCP is vectored to the address that is restored from the channel program's context. This means that before exiting, a channel program must itself arrange for where it will resume execution by configuring the value of its PC in its saved context so that it restarts at the desired location.

In this way, arbitrarily complex interrupt-driven state machines can be created as individual channel programs. Channel programs can be constructed that always start at their beginning, pick up where they left off, or pick up elsewhere, or have a mix of these approaches.

An example of a restarting channel program is shown below. Before exiting, the channel branches back to the address of the START label minus 1 (note that START - 1 = CH16) and then exits. This will leave the next value of the PC in the channel program's context as the address of the START label.

```
CH16:                              ;channel program 16
     EXIT EC=1 ST=0 INT=0 EP=1 cc_UC
                                   ;exit, no intr., leave PC @ next
START:                             ;nominal channel start address
     ST.IF base #0x8 SIZE=32       ;output note from R0
     JC   CH16, cc_UC              ;loop back before exit
```

Note that when the channel program is originally configured by the programmer, the PC field in the R7 context of this channel program should also be set to the address of the START label.

Similarly, an interrupt-driven state machine can be created by exiting with the next PC value pointing to the start of the next state in a state machine implemented by the channel program. The next example below shows a program starting at the address to the STATE0 label. It proceeds after the first interrupt to STATE1 - 1, where the channel program is left ready for the next state, STATE1 in the state machine. After the next interrupt, it executes to address STATE2 - 1 and the channel program is left ready for the next state, STATE2. After another interrupt, it proceeds through STATE2. The channel program jumps back to START, which is STATE0 - 1. The state machine has gone through one cycle and it is ready to restart in STATE0.

```
;This program is intended to test the sequence of exit/operate
;just as if you were implementing an interrupt driven
;state machine.
;It requires a periodic sequence of interrupts.

START:
     EXIT EC=1,ST=0,INT=0,EP=1,cc_UC;begin exit
STATE0:
     COMP.I    R5,#0x0          ;compare to interrupt number
                                ;it should be
```

```
      JC          ERROR,cc_NZ     ;jump to error routine
                                  ;if not correct
      ADD.I       R5,#0x1         ;increment state number
      EXIT EC=1,ST=0,INT=0,EP=1,cc_UC      ;begin exit
STATE1:
      COMP.I      R5,#0x1         ;compare to interrupt number
                                  ;it should be
      JC          ERROR,cc_NZ     ;jump to error routine
                                  ;if not correct
      ADD.I       R5,#0x1         ;increment state number
      EXIT EC=1,ST=0,INT=0,EP=1,cc_UC      ;begin exit
STATE2:
      COMP.I      R5,#0x2         ;compare to interrupt number
                                  ;it should be
      JC          ERROR,cc_NZ     ;jump to error routine if
                                  ;not correct
      LD.I        R5,#0x0         ;reset state number
      JC          START,cc_UC     ;jump back to start of
                                  ;state machine
```

The last state could just as easily have ended with an EXIT that resets the PC to the Channel Entry Table (EP = 0) rather than jumping back to START.

10.20.2 Channel Management for Small and Minimum Contexts

If Small or Minimum Contexts are being used, only some of the registers are saved and restored. The integrity of the GPRs that are **not** included in the context must be handled explicitly by channel programs, since these are not saved and restored with the context of the interrupted channel program.

Channel programs may still use all registers reliably. Channel programs can be so designed that they either ignore the values in unsaved registers, or use those registers to store constants that no channel program changes. Hence, they never need to be saved and restored. Alternately, channel programs can use these unused GPRs as temporary variables as long as the values of such registers cannot be corrupted by the interrupt of the channel program by a higher-priority channel (see **Page 10-132**).

10.20.3 Unused Registers as Global’s or Constants

Registers R0 through R3 (for the Small Context Model), or R0 through R5 (for the Minimum Context Model) can be used to store constants such as addresses that are available to all channel programs. Hence, these registers hold global data, and no channel program is allowed to change them.

Since the GPRs of the PCP are not directly accessible from the FPI Bus, there does need to be an initial channel program that sets these values at or near boot time. There are

two choices here. A boot-time interrupt channel program can be invoked once to perform initialization, or there can be a program that routinely loads these values as a matter of course, and is invoked at boot time or as upon receipt of the very first interrupt.

10.20.4 Dispatch of Low Priority Tasks

A higher-priority channel program may wish to start a low-priority background task, or periodically pause and re-start itself later when there is no other action required at the moment. This can be accomplished in several ways:

- Post an SRPN to a free SRN on the FPI Bus, then EXIT.
- Perform an EXIT, posting the interrupt to the PCP, and indicating the channel number to be started.
- Use a single channel program as a list-driven or state-driven task dispatcher.

The first approach is straightforward to program, but uses a system SRN resource. Its advantage is that it allows continuous channel operation without using the interrupt queue, or risk blocking other uses of the PCP.

The second approach can be implemented by having a looping channel program continue operation in the background. It will also always be superseded by any higher-priority tasks.

The third approach uses a channel program to dispatch other non-interrupt-driven channel programs in an arbitrary order determined by the channel program dispatcher. In this way, multiple tasks could be continuously operated without over-using the PCP service-request queue. This approach would be useful when the aim is to poll for service requests in the peripheral SRNs rather than having them started by PCP hardware.

10.20.5 Code Reuse Across Channels (Call and Return)

A special jump instruction is included in the PCP instruction set to allow subroutines to be called from multiple channel programs. A routine may be jumped to directly, and then returned from using the JC.IA instruction. JC.IA allows a calling channel program to set aside a register for its return address, which will typically be the value of the next PC. The called subprogram can then execute a JC.IA, to the address stored in the register specified, causing a return-from-subroutine operation. The programmer must adopt and enforce a calling convention to determine which register holds the return address. Register R2 is normally used for this purpose.

For example:

```
Main Routine:                    Subroutine:
     LD.IL     R2,#RETURN            SUB:  MOV...
     JC.A      #SUB                        ADD...
RETURN: MOV    ...                         ...
     ...                                   JC.IA R2
```

10.20.6 Case-like Code Switches (Computed Go-To)

The JC.I instruction can be used to implement a multi-way branch for branch-on-bit or branch-on-state conditional branches. This instruction allows a conditional relative jump based on an index held in a register. If this instruction is combined with a table of jump addresses, a switch-type statement can be implemented. The default case, that is when the condition code = False, is the next instruction, as is the jump with register index = 0. The table can be any arbitrary length. The index register should be checked for range before the jump into the table is performed.

For Example:

```
          COMP  R3,#5      ;compare R3 to #5 - the number
                           ;of entries in the table
          JC.I  R3,cc_ULE
DEFAULT:  JL    #case_0    ;destination if R3 = 0 or
                           ;condition = false
          JL    #case_1    ;destination if R3 = 1
          JL    #case_2    ;destination if R3 = 2
          JL    #case_3    ;destination if R3 = 3
          JL    #case_4    ;destination if R3 = 4
          JL    #case_5    ;destination if R3 = 5
```

10.20.7 Simple DMA Operation

A simple interrupt-driven DMA requires at least the Small Context Model to operate properly. Its operation is consists of three stages:

- The device interrupts the PCP to indicate it can receive or provide data.
- The PCP moves the amount of data it is programmed to move.
- The PCP eventually finishes and interrupts the CPU to notify it that the DMA is complete.

There are two options for implementing a simple DMA operation, copy and burst copy.

10.20.7.1 COPY Instruction

A simple DMA channel program can consist of only two instructions. In the example below, a device interrupts the PCP to notify it that it has data in its output buffer, which is 4 words deep. The COPY instruction copies 4 words to memory at a time. It decrements CNT1 (which is initialized by the CPU in CR6_CNT1 context) after each 4 word transfer. The EXIT command then executes, and if CNT1 was decremented to 0, the condition code causes it to issue an interrupt with the value held R6_SRPN.

```
COPY  DST+,SRC,CNC=1,BRST=4,SIZE=32   ;do peripheral -> memory DMA
EXIT  EC=0,ST=0,INT=1,EP=0,cc_CNZ     ;transfer done, so exit
```

In the example above, the COPY instruction increments the destination held in R5 (DST+), and the source address is left constant in R4 (SRC). All permutations of decrement, increment or do not modify can be applied to either pointer register (R4 and R5) by use of the SRC and DST fields (SRC-, SRC+ or SRC and DST-, DST+ or DST).

Building on this basic DMA method, scatter-gather DMA channels can be created.

10.20.7.2 BCOPY Instruction (Burst Copy)

The BCOPY instruction is in principle similar to the COPY instruction except that it uses the FPI Burst mode to perform the transfers rather than performing individual reads/writes. As for the COPY instruction, the FPI Bus is locked between the burst read and burst write to ensure that a valid set of data is transferred. The BCOPY instruction allows support of all burst sizes supported by FPI Burst Mode except a burst size of 1 (i.e. 2, 4 or 8 words). The CNT0 field is used to control the burst size. Both the source and destination addresses (R4 and R5) must be correctly aligned for the burst size being used (see the FPI Bus description for details). If either address is incorrectly aligned, the PCP will generate an Illegal Operation Error Exit.

See also **Page 10-139** for TC1797 specific details of the BCOPY instruction.

10.21 PCP Programming Notes and Tips

This section discusses constraints on the use of the PCP and points out some non-obvious issues.

10.21.1 Notes on PCP Configuration

For configuring of the PCP, some notes should be regarded.

- Only one Context Model may be used at a time for all channels, and the PCP must remain in that Context Model once started and configured.
- In order for a specific channel program to be enabled, its context must have R7.CEN = 1. If R7.CEN = 0, the channel program will terminate when invoked, and cause a Disabled Channel Request error.
- The Channel Context Address from the FPI Bus as viewed during channel configuration is as follows:
- Full Context Model: PRAM Base + $20_H \times n$
- Small Context Model: PRAM Base + $10_H \times n$
- Minimum Context Model: PRAM Base + $08_H \times n$
  where n is the channel number.
- PCP_CS.RCB and context must be consistent. If RCB is configured to 0, then each channel program will start at the PC restored from its context. If the wrong address is pre-configured in the context, the channel program will not operate properly.
- The programmer of the PCP may lock PCP_CS by setting PCP_CS.EIE = 1. When the global ENDINIT bit is set, the PCP_CS register will no longer be writable, and attempting to do so will cause an FPI Bus error.
- An error condition will result in an interrupt being sent to the local FPI Bus master. The targeted interrupt service routine must be capable of dealing with the cause as recorded in PCP_ES, and, if required, it must be able to return the halted channel program to operation. The minimum required to do that is to set the context value of R7.CEN = 1.
- The only PCP Register bit that can be dynamically modified during PCP operation is the PCP_CS.EN bit. When writing to any other PCP Register bits, the user must ensure that the PCP is disabled (PCP_CS.EN = 0) and that the PCP is quiescent (PCP_CS.RS = 0).

10.21.2 General Purpose Register Use

When using the general purpose registers of the PCP, some notes should be regarded.

- The most significant 16 bits of R7 may not be written, and will always read back as 0. However, no error will occur if a write to the most significant 16 bits occurs.
- Care must be taken with the use of R6 as a general-use register to ensure that R6 contains the correct value prior to execution of the EXIT command. As R6 contains the CNT1 (counter used in COPY and optionally in EXIT instructions), SRPN and

TOS (service request number to use during optional interrupt at channel program EXIT) fields, R6 should not be used to pass values from one invocation of a channel program to the next invocation.

- If PRAM is to be accessed programmatically, then R7.DPTR must be configured properly as a pointer into the PRAM. This points to the 64-word segment that may be addressed by the xx.P instructions and the xx.PI instructions. It is not recommended to set R7.DTPR to point into the CSA. Special care must be taken that the Context PRAM is not overwritten.
- The programmer must be careful not to inadvertently clear R7.CEN when updating R7.DTPR (or any other field in R7). This would cause the channel program to generate a disabled channel interrupt to the CPU when the next interrupt request to the channel occurs.
- Any update to the Flags that is caused by an instruction (e.g. MOV R7, R0 which updates Z and N) takes precedence over any explicit bits that are moved to R7. See **Page 10-10**.
- The interrupt system assumes SRPN 0 is not a request. Full Context packing leaves the least significant 8 × 32-bit entries where channel 0 would normally be unused. That is, PRAM Base -> PRAM Base + 1 channel. In addition, for Small Context, the least significant 4 × 32-bit entries are unused, and for Minimum Context the least significant 2 × 32-bit entries are un-used. These “unused” entries should not be used by channel programs.
- If EP = 0 is used, or if PCP_CS.RCB = 1, a Channel Entry Table must be provided at the base of CMEM.
- If there is a plan to use the Small or Minimum Context Model, and the lower registers are to hold global values, then there needs to be an initial channel program that sets these values at or near boot time. There are at least two choices for implementing this. For instance, a boot interrupt channel program can be invoked once to perform initialization, or there can be a program that routinely loads these values as a matter of course, and it is invoked at boot time, or at the very first interrupt. See **Page 10-128**.
- When using Small or Minimum Context models and allowing a channel to be interrupted, care must be taken to ensure that the value of any registers that are not included in the context but are being used by a channel are not corrupted by interruption of the channel and subsequent operation of a higher-priority channel. Particular care must be taken when using instructions that use R0 implicitly. If necessary, critical instruction sequences should be protected by use of the R7.IEN bit (see **Page 10-134**).

10.21.3 Use of Channel Interruption

For channel interruption, the following note should be regarded.

- When a channel program consists of only a few instructions, it is best to configure the channel to be non-interruptible. This increases overall efficiency by removing the context save/restore overhead that would be incurred if the channel were to be interruptible.

10.21.3.1 Dynamic Interrupt Masking

A channel program can dynamically control whether it can be interrupted by use of the R7.IEN bit. When masking interrupts (by clearing R7.IEN), it must be noted that there is a delay of one instruction before the mask becomes effective. As a result the instruction that clears R7.IEN must be placed at least one instruction before the instruction sequence that is to be un-interruptible. As an example, consider the following sequence:

```
CLR   R7,IEN     ;Clear the R7.IEN bit
                 ;<< Interrupt can occur here
NOP
                 ;<< Interrupt can occur here
                 ;First instruction of non-interruptible
                 ;code sequence
```

10.21.3.2 Control of Channel Priority (CPPN)

The PCP has three extended Service Request Nodes (PCP_SRC9, PCP_SRC10 and PCP_SCR11) that allow storage of suspended channel interrupt requests. This allows interrupt nesting to a depth of four. This limit on the nesting depth carries the danger that a high-priority service request will not be serviced because the PCP’s interrupt nesting depth has been exceeded.

It is recommended that a four-level “grouping” scheme should be adopted to avoid this problem. All PCP interrupt sources should be listed in order of their SRPNs.This list should then be subdivided into four contiguous groups, Group 0 being the lowest priority and Group 3 the highest. The CSA for each channel program should be configured such that CR6.CPPN contains the SRPN value of the highest channel program within the group to which the channel belongs. As each channel starts, the Operating Priority (CPPN) of the channel is loaded from the context. Using the scheme recommended above, any channel program will run with the priority of the highest SRPN within the group. As a result, the channel can only be interrupted by a service request from a higher-priority group (e.g. a Group 0 channel program can be interrupted by a new service request for a channel in any group from 1 to 3, a group 2 channel program can only be interrupted by a new service request for a channel in group 3).

Note: When using this scheme, each channel program must ensure prior to channel exit that the R6.CPPN field contains the appropriate value, so that when the channel is next invoked, it will run at the correct priority.

10.21.4 Implementing Divide Algorithms

As discussed in **Section 10.18.4**, a divide algorithm **must** always start with a DINIT instruction followed by a number of DSTEP instructions (up to four depending on the data width that is required). Prior to execution of any DSTEP instruction, R0 always contains either 0 (if this is the first DSTEP instruction in a divide sequence R0 contains 0 due to the preceding DINIT instruction, or the remainder from the previous DSTEP instruction). The dividend to be used in this step is generated in R0 by taking 256 × the remainder of the last DSTEP instruction (R0 << 8) and adding the most significant byte of Rb (Rb >> 24) as the LSB of the new dividend.

Since the remainder of the last DSTEP instruction is by definition always less than the divisor (Ra), it can be guaranteed that the result of the division of the dividend (calculated as above) by the divisor (Ra) can always be contained within an 8-bit result. The description given on **Page 10-95** only holds true under this condition. If the restrictions on the use of the DSTEP instruction (specified on **Page 10-86**) are adhered to, the above condition will always be met and this description of the instruction is correct. Failure to adhere to these conditions will lead to invalid results, which are outside the scope of this document.

During execution of a divide sequence, Rb is used both to compile the final divide result and to hold the remnants of the original dividend. For example, in a 32-/32-bit divide sequence (which consists of 4 DSTEP instructions - see below), Rb will have the following content:

- After the 1st DSTEP instruction:
  The least significant 3 bytes (24 bits) of the original 32-bit dividend (held in the most significant 3 bytes of Rb) and the most significant byte of the final result (held in the least significant byte of Rb).
- After the 2nd DSTEP instruction:
  The least significant 2 bytes (16 bits) of the original 32-bit dividend (held in the most significant 2 bytes of Rb) and the most significant 2 bytes of the final result (held in the least significant 2 bytes of Rb).
- After the 3rd DSTEP instruction:
  The least significant byte of the original 32-bit dividend (held in the most significant byte of Rb) and the most significant 3 bytes of the final result (held in the least significant 3 bytes of Rb).
- After the final DSTEP instruction:
  The 32 bit final result.

Note that the DSTEP instruction **always** uses the divisor as a 32 bit value. In any divide sequence, the dividend can be 8, 16, 24 or 32 bits (according to the number of DSTEP

instructions in the sequence) but the divisor is **always** 32 bits. Prior to the DINIT instruction, the dividend must always occupy the appropriate most significant bits within the 32-bit dividend register (Rb).

Divide Examples

Example of a 32/32 bit divide (R5 / R3):

```
DINIT     R5, R3     ;Initialize ready for the divide
JC        HANDLE_DIVIDE_BY_ZERO, cc_V     ;V flag was set
                     ;so jump to divide
                     ;by zero error handler
DSTEP     R5, R3     ;4 DSTEP instructions
                     ;(4 * 8 = 32 bit
DSTEP     R5, R3     ;divide)
DSTEP     R5, R3
DSTEP     R5, R3
```

After this sequence, R5 holds the result, R0 the remainder and R3 is unchanged.

Example of a 8/32 bit divide (R4 / R2):

```
RR        R4, 8      ;Rotate R4 right by 8 to move
                     ;least significant byte into
                     ;most significant byte
DINIT     R4, R2     ;Initialize ready for the divide
JC        HANDLE_DIVIDE_BY_ZERO, cc_V     ;V flag was set
                     ;so jump to divide
                     ;by zero error handler
DSTEP     R4, R2     ;DSTEP instruction
                     ;(1 * 8 = 8 bit divide)
```

After this sequence, R4 holds the result, R0 the remainder and R2 is unchanged.

Note that the above example is specified as being a 8/32 bit divide rather than an 8/8 bit divide (see comments above).

10.21.5 Implementing Multiply Algorithms

As discussed in **Section 10.18.4**, a multiply algorithm **must** always start with a MINIT instruction, followed by a number of MSTEP32 or MSTEP64 instructions. The MSTEP32 instruction is used to compile a multiplication result contained in 32 bits, discarding any overflows. The MSTEP64 instruction is used to compile a 64-bit multiplication result with the least significant 32 bits of the result contained in Rb and the most significant 32 bits of the result contained in R0.

Multiply Examples

Example of a 32 × 8 bit multiply (R4 × R1) yielding a 32 bit result (R4 = 32 bit, R1 = 8 bit):

```
RR        R1, 8     ;Rotate least significant byte of R1
                    ;to most significant byte
MINIT     R1, R4    ;Initialize ready for multiply
MSTEP32   R1, R4    ;Perform one MSTEP32 instruction
                    ;(8 bit multiply)
```

After this sequence, R0 holds the result, R1 is left unchanged (right rotated by RR instruction then left rotated by MSTEP32 instruction), and R4 is unchanged. The result is only valid if there is no overflow (i.e. the product of the 8-bit number in R1 multiplied by the 32-bit number in R4 can be contained within 32 bits). It is the user's responsibility to ensure that this is the case. The overflow condition cannot be detected after execution of the multiply sequence.

Example of a 32 × 16 bit multiply (R3 × R2) yielding a 32 bit result
(R3 = 32 bit, R2 = 16 bit):

```
RR        R2, 8     ;Perform two 8 bit rotations
                    ;(RR instructions) to get original
                    ;least significant 16 bits into
                    ;most significant 16 bits
RR        R2, 8
MINIT     R2, R3    ;Initialize ready for multiply
MSTEP32   R2, R3    ;Perform two MSTEP32 instructions
                    ;(16 bit multiply)
MSTEP32   R2, R3
```

After this sequence, R0 holds the result, R2 is left unchanged (right rotated by two RR instructions, then left rotated by two MSTEP32 instructions), R3 is unchanged. The comment above regarding overflow also applies to this sequence.

Example of a 32 × 32 bit multiply (R5 × R2) yielding a 64 bit result
(R5 = 32 bit, R2 = 32 bit):

```
MINIT     R2, R5    ;Initialize ready for multiply
MSTEP64   R2, R5    ;Perform 4 MSTEP64 instructions
                    ;(64-bit multiply)
MSTEP64   R2, R5
MSTEP64   R2, R5
MSTEP64   R2, R5
```

After this sequence R0 and R2 hold the result (most significant word in R0, least significant word in R2), R5 is unchanged. There is no possibility of overflow as the result of 32 × 32 bits can always be contained in 64 bits.

10.22 Implementation of the PCP in the TC1797

The addresses of the PCP registers and memories in the TC1797 are given in the following subsections:

10.22.1 PCP Memories

In the TC1797, the location of the registers and the memories sizes of the PRAM and the CMEM are given in **Table 10-19**.

Table 10-19 General Block Address Map

Unit		Address Range	Access Mode		Size
			Read	Write	
PCP	Reserved	$F004\ 0000_H$ - $F004\ 3EFF_H$	BE	BE	–
	PCP Registers	$F004\ 3F00_H$ - $F004\ 3FFF_H$	see **Page 10-59**		256 byte
	Reserved	$F004\ 4000_H$ - $F004\ FFFF_H$	BE	BE	–
	PRAM	F005 0000 - F005 3FFF	U, SV, 32	SV, 32	16k
	Reserved	F005 4000 - F005 FFFF	BE	BE	-
	CMEM	F006 0000 - F006 5FFF	U, SV, 32	SV, 32	
	Reserved	F006 6000 - F006 FFFF	BE	BE	-
	PCP Data Memory (PRAM)	$F005\ 0000_H$ - $F005\ 3FFF_H$	nE, 32	nE, 32	16 Kbyte
	Reserved	$F005\ 4000_H$ - $F005\ FFFF_H$	BE	BE	–
	PCP Code Memory (CMEM)	$F006\ 0000_H$ - $F006\ 7FFF_H$	nE, 32	nE, 32	32 Kbyte

Note: “BE” means that in case of an access to this address region a bus error is generated.

10.22.2 BCOPY Instruction

In the TC1797, the BCOPY instruction can be used to perform burst transfers (2, 4, or 8 words) with DMI memories (Local data RAM) and the PCP memories. Other internal and external memories can be accessed using a burst size of 2 words only (CNT0 = 10_B).

10.22.3 PCP Reset Operation

The PCP module can be reset by a system hardware signal (hard reset).

PCP Hard Reset

A PCP hard reset is always triggered if at least one of these TC1797 reset sources becomes active:

- Watchdog Timer Reset
- Software Reset
- Hardware Reset
- Power-on Reset

Each of these reset sources forces a hardware reset of the functional blocks within the PCP module. The effect of hard reset within the PCP is to:

- Halt any operating channel
- Reset all control registers to their reset values
- Reset the PCP Processor Core to its default state
- Reset the FPI Bus interface

11 Direct Memory Access Controller (DMA)

This chapter describes the Direct Memory Access (DMA) Controller and the Memory Checker Module (MCHK) of the TC1797. It contains the following sections:

- Functional description of the DMA controller kernel (see **Section 11.2**)
- DMA controller module register description (see **Section 11.3**)
- TC1797 implementation-specific details of the DMA controller (interrupt control, address decoding, clock control, see **Section 11.4**)
- Functional description of the Memory Checker (MCHK) module (see **Section 11.5**)
- Memory Checker module register description (see **Section 11.5.2**)

*Note: The DMA kernel register names described in **Section 11.3** are referenced in the TC1797 User's Manual by the module name prefix "DMA_".*

11.1 What is new

Major differences of the AudoFuture DMA compared to AudoNG:

- A new mode for the shadow address register was introduced to support endless channel re-starts without CPU intervention. In this new mode, the shadow address register can be written directly and it is not re-set automatically when loaded into target or destination address register (**Page 11-7** and **Page 11-93** / **Page 11-99**). The new Shadow Register Write Enable bit was added to the register DMA_ADRCR1/0x.
- The DMA channels are supporting now transactions with data moves > 32 KB with wrap around after 32 KB (bit fields CHCR.TREL and CHSR.TCOUNT are extended to 10 bit. See **Page 11-86** / **Page 11-86** and **Page 11-90** / **Page 11-90**. See also DMA_ADRCR.CBLS/CBLD **Page 11-93** / **Page 11-93**).
- As a central module of the AudoFuture system on chip architecture, the DMA is now directly connected to the LMB with an own LMB master interface. The DMA master interface to the RPB was removed as the AudoFuture system architecture is based on a single LMB and single FPI (SPB) bus.
- The RPB BCU control registers where removed as there is no remote peripheral bus any more in AudoFuture)
- The Cerberus module is now connected to the DMA Peripheral Interface. The enables the Cerberus module to have direct access to the FPI and to the LMB bus via the DMA master interfaces either with the highest or with the lowest priority on LMB / FPI.
- 3-level programmable priority of the DMA Sub-Block at the on chip bus interfaces (2-level in AudoNG). The bit field DMAPRIO is expanded to 2 bits in the Channel Control register to support the three DMA on chip bus priorities (**Page 11-86**). The new structure is on the one hand compatible to the AudoNG channel priorities on the SPB, on the other hand it allows to use DMA channels for low priority background tasks on LMB like memory scrubbing. Additionally the DMA On Chip Bus priorities (**Page 11-24**) and the DMA bus switch priorities are adapted (**Page 11-22**).

- The System Interrupt Registers are removed from the DMA module (moved to CPU, SCU and PMU).
- The DMA module has now 8 Service Requests nodes in general. *DMA interrupt outputs DMA SR[7:0] are connected to interrupt nodes. SR[15:8] are used as DMA channel request inputs* (DMA_SRCn (n = 0-7)).
- Up to 16 selectable request inputs per DMA channel (upt to 8 in AudoNG). To allow a more flexible usage of the Move Engine Channels, the number of channel request inputs was increased to 16 (DMA_CHRC0x/1x.PRSEL expanded from 3 bit to 4 bit, **Page 11-86**) and the DMA request wiring matrix was re-defined (**Page 11-102**).
- Adapted the DMA access protection assignment to the AudoFuture address map (**Page 11-112**). The address protection sub-ranges are mapped to OVRAM, PCP PRAM, LDRAM, and SPRAM.
- In general the DMA control registers where adapted to the new structure (one FPI master interface and one LMB master interface instead of two FPI master interfaces).
- A figure with the detailed implementation of the Memory Checker algorithm was added to the Memory Checker chapter (**Page 11-128**).

11.2 DMA Controller Kernel Description

The DMA Controller of the TC1797 transfers data from data source locations to data destination locations without intervention of the CPU or other on-chip devices. One data move operation is controlled by one DMA channel. Eight DMA channels are provided in one DMA Sub-Block. The Bus Switch provides the connection of the DMA Sub-Blocks to the two On Chip Bus interfaces and a DMA Peripheral interface. In the TC1797, the two On Chip Bus interfaces are connected to the System Peripheral Bus and the LMB Bus. The DMA Peripheral interface provides a connection to the Cerberus module, Micro Link Interface modules and other DMA-related devices (Memory Checker module in the TC1797). Clock control, address decoding, DMA request wiring, and DMA interrupt service request control are implementation-specific and are managed outside the DMA controller kernel.

The index “m” in the following block diagram refers to the DMA Sub-Block number (m = 0-1).

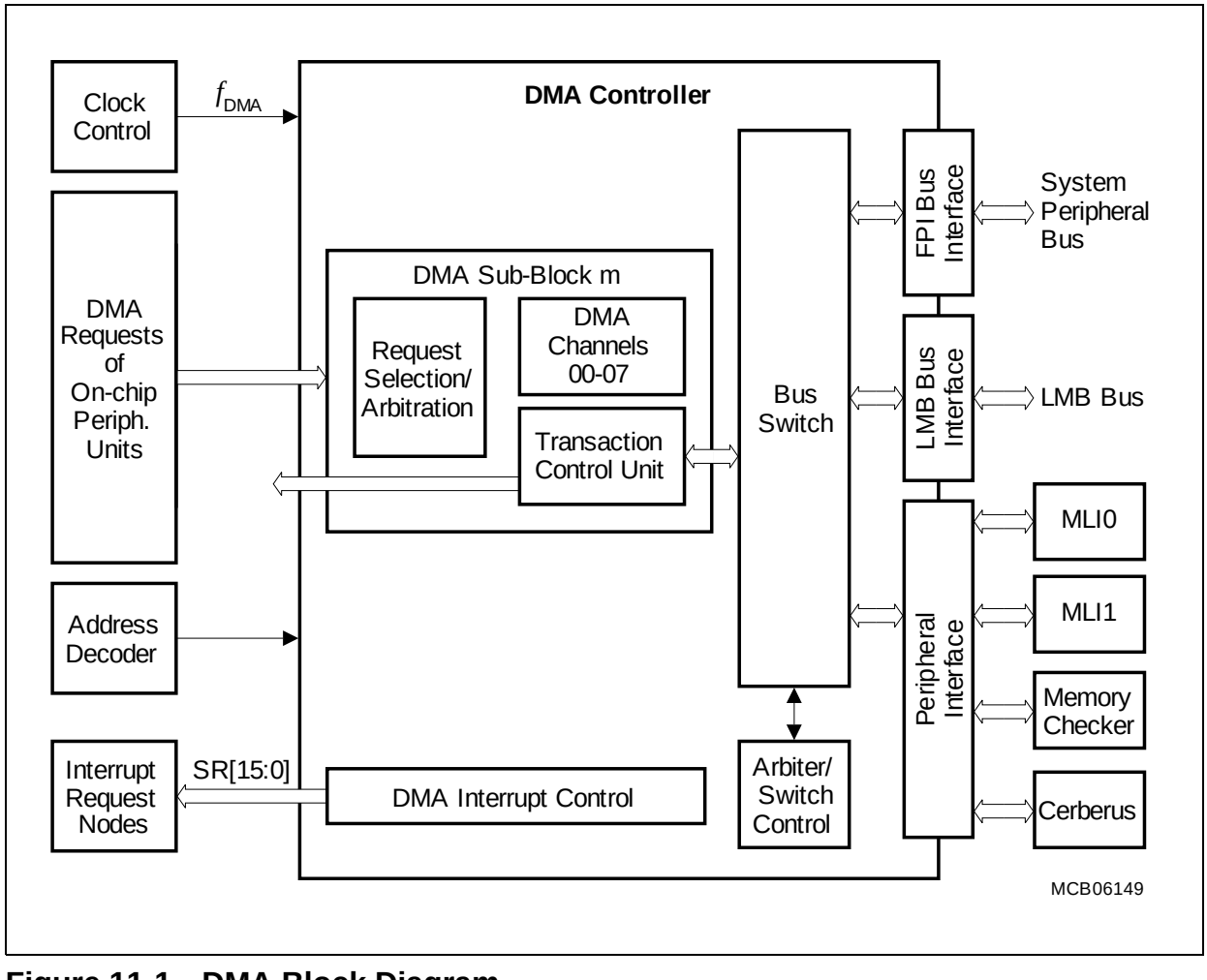


Figure 11-1 DMA Block Diagram

11.2.1 Features

The DMA controller has the following features:

- 16 independent DMA channels
  - 2 DMA Sub-Blocks with (8 DMA channels per DMA Sub-Block)
  - DMA Sub-Blocks with support of parallel channel execution (1 channel per Sub-Block, both Sub-Blocks in parallel)
  - Up to 16 selectable request inputs per DMA channel
  - 2-level programmable priority of DMA channels within the DMA Sub-Block
  - Software and hardware DMA request
  - Hardware requests by selected on-chip peripherals and external inputs
- Programmable priority of the DMA Sub-Blocks on the bus interfaces
- LMB Master Interface with 64-bit read buffer for read accesses to cached areas (Segment 8).
- Individually programmable operation modes for each DMA channel
  - Single Mode: stops and disables DMA channel after a predefined number of DMA transfers
  - Continuous Mode: DMA channel remains enabled after a predefined number of DMA transfers; DMA transaction can be repeated
  - Programmable address modification
  - Two shadow register modes (with / w/o automatic reset and direct write access).
- Full 32-bit addressing capability of each DMA channel
  - 4 Gbyte address range
  - Data block move > 32 Kbyte per DMA transaction
  - Circular buffer addressing mode with flexible circular buffer sizes
- Programmable data width of DMA transfer/transaction: 8-bit, 16-bit, or 32-bit
- Register set for each DMA channel
  - Source and destination address register
  - Channel control and status register
  - Transfer count register
- Flexible interrupt generation (the service request node logic for the MLI channels is also implemented in the DMA modules)
- DMA module is working on FPI frequency, LMB interface on LMB frequency.
- Dependant on the target/destination address, Read/write requests from the Move Engines are directed to the FPI Bus, LMB Bus, MLI modules or to the Cerberus module.

11.2.2 Definition of Terms

Some basic terms must be defined for the functional description of the DMA controller.

DMA Move

A DMA move is an operation that always consists of two parts:

1. A read move that loads data from a data source into the DMA controller
2. A write move that puts data from the DMA controller to a data destination

Within a DMA move, data is always moved from the data source via the DMA controller to the data destination. Data is temporarily stored in the DMA controller. The data widths of read move and write move are always identical (8-bit, 16-bit or 32-bit). Data assembly or disassembly is not supported.

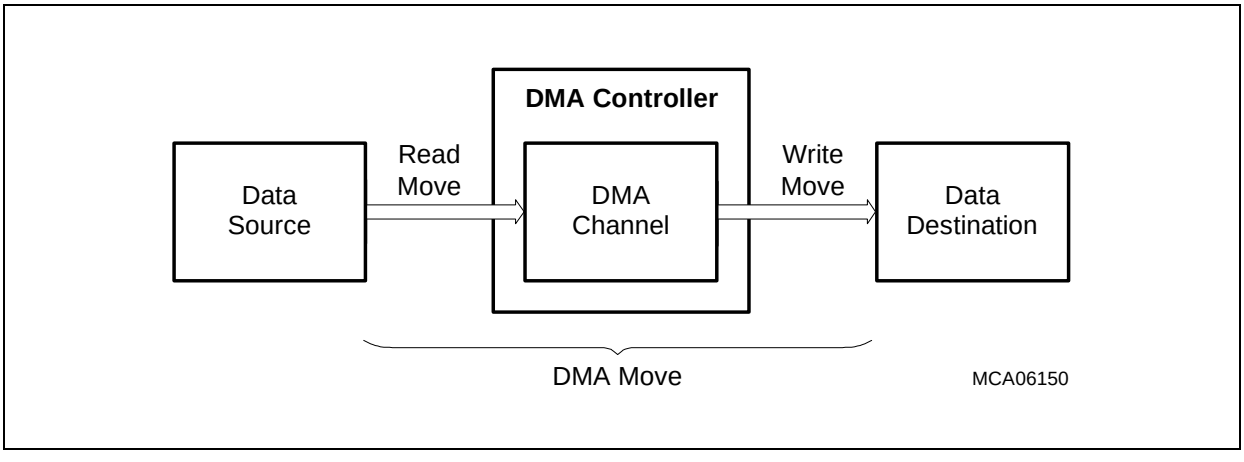


Figure 11-2 DMA Definition of Terms

DMA Transfer

A DMA transfer can be composed of 1, 2, 4, 8 or 16 DMA moves.

DMA Transaction

A DMA transaction is composed of several (at least one) DMA transfers. The Transfer Count determines the number of DMA transfers within one DMA transaction.

Example:
1024 word (32-bit wide) transactions can be composed of 256 transfers of four DMA word moves, or 128 transfers of eight DMA word moves.

11.2.3 DMA Principles

The DMA controller supports DMA moves from one address location to another one. DMA moves can be requested either by hardware or by software. DMA hardware requests are triggered by specific request lines from the peripheral modules or from other DMA channels (see **Figure 11-3**). The number of available DMA request lines from a peripheral module varies depending on the module functionality. Typically, the parallel occurrence of DMA requests and interrupts requests for DMA channels is possible. Therefore, the interrupt control unit and the DMA controller can react independently to interrupt and DMA requests that have been generated by one source.

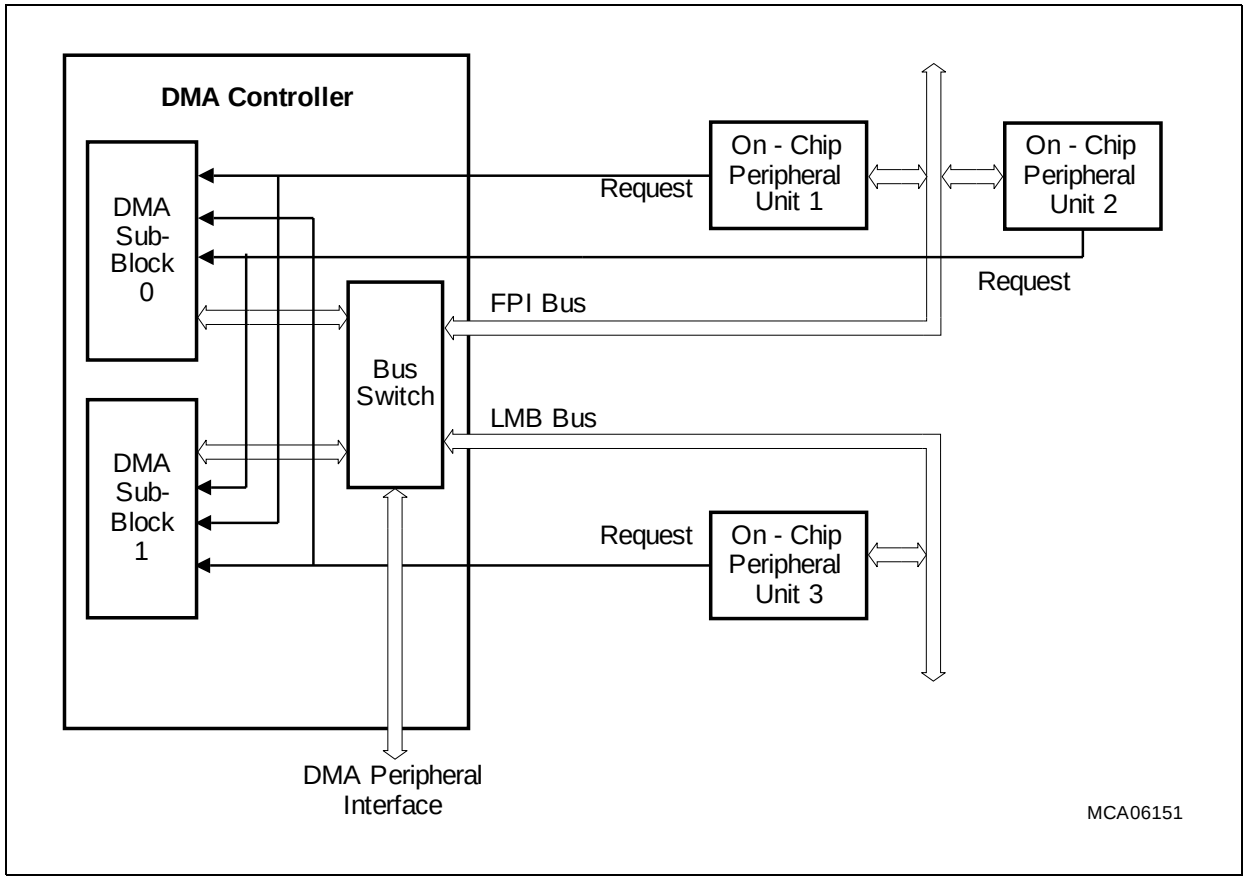


Figure 11-3 DMA Principle

The DMA controller mainly consists of two DMA Sub-Blocksand a Bus Switch. Once configured, the DMA Sub-Blocks are able to act as a master on the FPI Bus and on the LMB Bus.

11.2.4 DMA Channel Functionality

Each of the 16 DMA channels has one associated register set containing seven 32-bit registers. These registers are numbered by one index to indicate the related DMA Sub-Block and one index to indicate the related DMA channel: Index “m” refers to the DMA Sub-Block number (m = 0-1) and Index “n” refers to the channel number (n = 0-7) within the DMA Sub-Block.

Example: CHCR04 is the Control Register of DMA channel 4 in Sub-Block 0.

The register set of a DMA channel register contains the following registers:

- Channel mn Control Register CHCR0n (for details, see **Page 11-86**)
- Channel mn Status Register CHSR0n (for details, see **Page 11-90**)
- Channel mn Interrupt Control Register CHICR0n (for details, see **Page 11-91**)
- Channel mn Address Control Register ADRCR0n (for details, see **Page 11-93**)
- Channel mn Source Address Register SADR0n (for details, see **Page 11-98**)
- Channel mn Destination Address Register DADR0n (for details, see **Page 11-99**)
- Channel mn Shadow Address Register SHADR0n (for details, see **Page 11-100**)

11.2.4.1 Shadowed Source or Destination Address

As a typical application, an ASC module that receives data (fixed source address) has to deliver it to a memory buffer using a DMA transaction (variable destination address). After a certain amount of data has been transferred, a new DMA transaction should be initiated to deliver further ASC data into another memory buffer. While the destination address register is updated during a running DMA transaction with the actual destination address, a shadow mechanism allows programming of a new destination address without disturbing the content of the destination address register. In this case, the new destination address is written into a buffer register, i.e. the shadow address register. At the start of the next DMA transaction, the new address is transferred from this shadow address register to the destination address register without CPU intervention. This shadow mechanism avoids the CPU having to check for the end of a DMA transaction before reprogramming address registers.

The shadow address register can be used also to store a source address. However, it cannot store source and destination address at the same time. This means that the shadow mechanism makes it possible to automatically update either a new source address, or a new destination address at the start of a DMA transaction. If both address registers (for source and destination address) have to be updated for the next DMA transaction, a running DMA transaction for this channel must be finished. After that, source and destination address registers can be written before the next DMA transaction is started.

Figure 11-4 shows the actions that take place when a source address register is updated. The update of a destination register happens in an equivalent manner.

When writing a new address to the (address of) the source or destination address register and no DMA transaction is running, the new address value is directly written into the source or destination address register. In this case, no buffering of the address is required. When writing a new address to the (address of) the source or destination address register and a DMA transaction is running, no transfer to an address register can take place and SHADRmn holds the new address value that was written. For this operation, bit field ADRCRmn.SHCT must be set either to 01_B (address is a source address) or 10_B (new address is a destination address). At the start of the next DMA transaction, the shadow transfer takes place and the content of SHADRmn is written either into SADRmn or DADRmn (ADRCRmn.SHCT must be set accordingly). After the shadow transfer, SHADRmn is set to 0000 0000_H if the shadow register write enable bit is set to 0 (ADRCRmn.SHWEN = 0). In this case (ADRCRmn.SHWEN = 0), the software can check by reading the shadow address register whether or not the shadow transfer has already taken place.

Only one address register can be shadowed while a transaction is running, because the shadow register can only be assigned either to the source or to the destination address register. Note that the shadow address register transfer has the same behavior in Single and Continuous Mode. When the shadow mechanism is disabled (ADRCRmn.SHCT = 00_B), SHADRmn is always read as 0000 0000_H.

If the shadow address register write enable bit is set to 1 (ADRCRmn.SHWEN = 1), the shadow register SHADRmn can be directly written. In this case (ADRCRmn.SHWEN = 1) the value stored in the SHADRmn is not modified when the shadow transfer takes place, and the shadow mechanism remains active and the shadow transfer will be repeated until Channel mn is reset or until the value in SHADR is 0000 0000_H, is written into the shadow register (direct or indirect by writing to the source or destination address register according to the shadow control register ADRCRmn.SHCT).

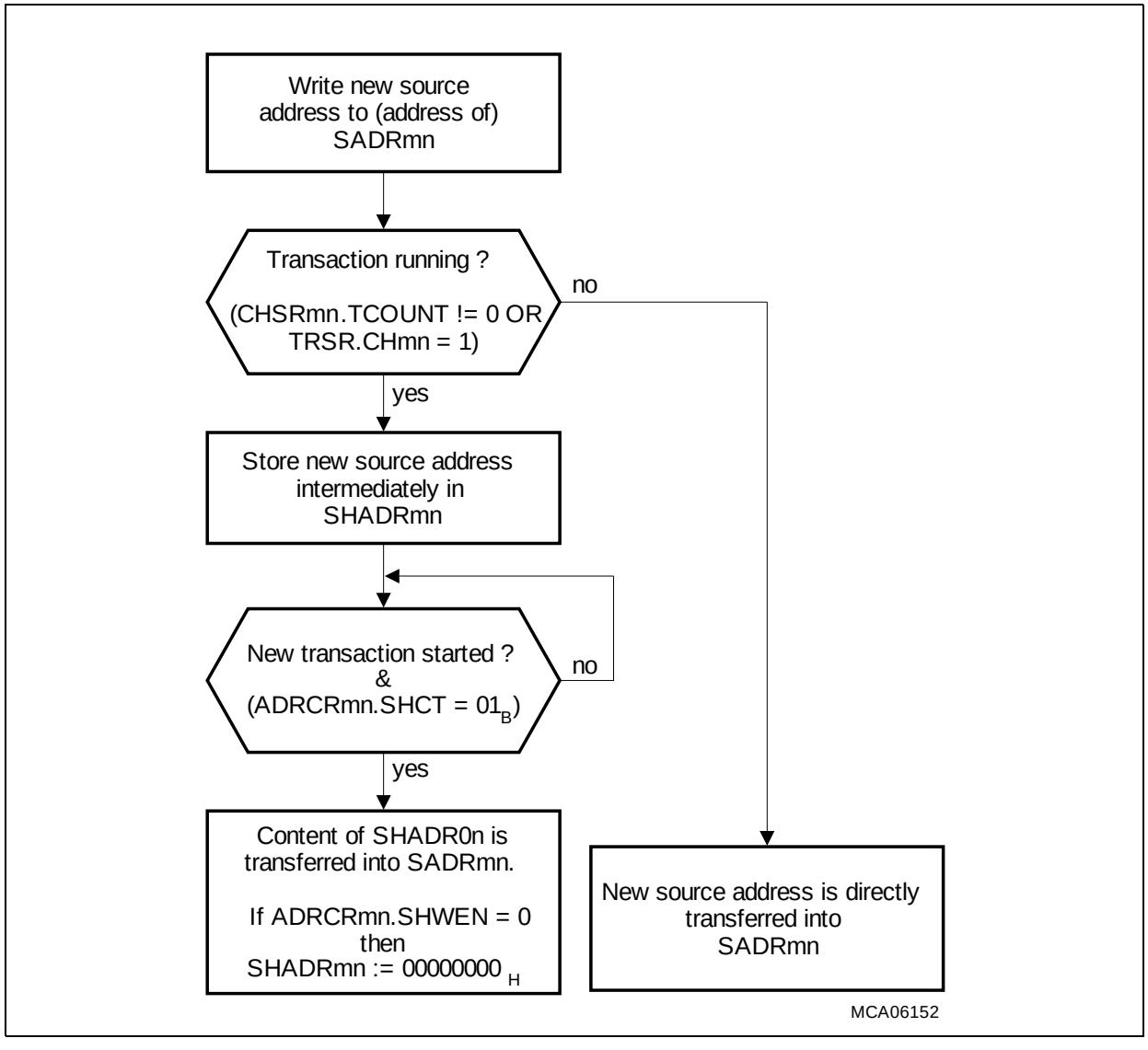


Figure 11-4 Source Address Update (m = 0-1)

The transfer count of a DMA transaction, stored in bit field CHCRmn.TREL, can also be programmed if the DMA transaction is running. At the start of a DMA transaction, TREL is transferred to bit field CHSRmn.TCOUNT, which is then updated during the DMA transaction.

No reload of address or counter will be done if TCOUNT is not equal to 0.

The reprogramming of channel specific values (except for the selected address shadow register) should be avoided while a DMA channel is active.

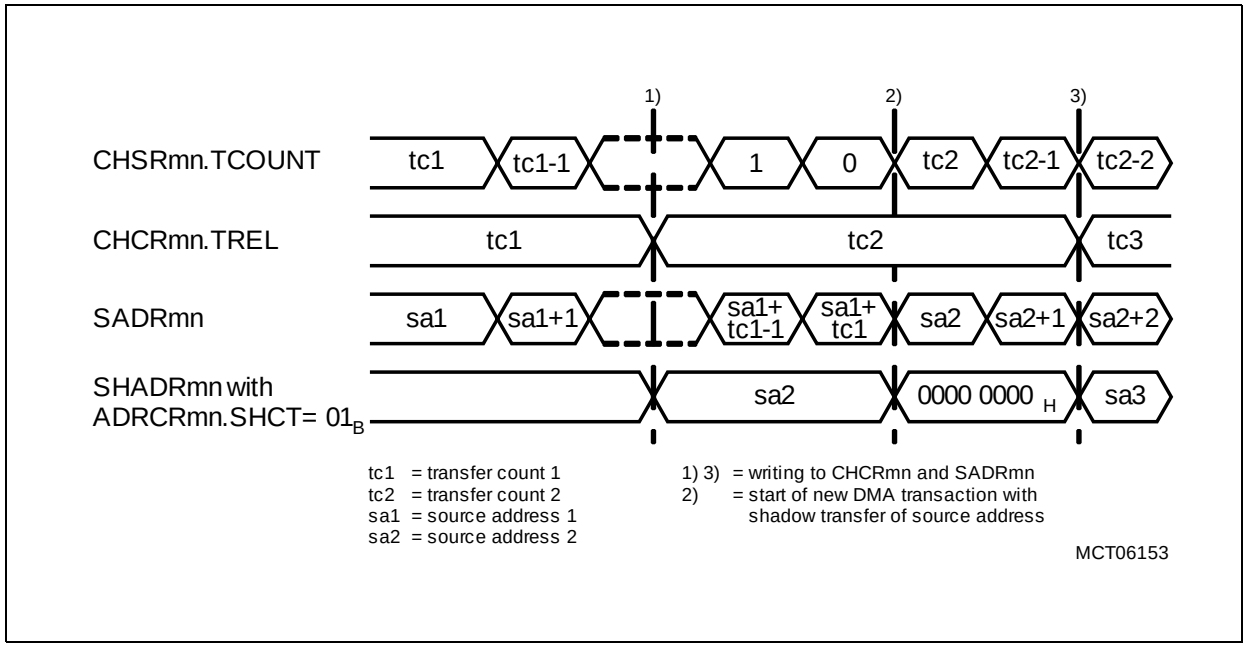


Figure 11-5 Shadow Source Address and Transfer Count Update with ADRCRmn.SHWEN = 0 (m = 0-1)

Figure 11-5 shows how the contents of the source address register SADRmn and the transfer count CHSRmn.TCOUNT are updated during two DMA transactions with a shadowed source address and transfer count update.

At reference point 2) the DMA transaction 1 is finished and DMA transaction 2 is started. At 1) the DMA channel is reprogrammed with two new parameters for the next DMA transaction: Transfer count tc2 and source address sa2. Source address sa2 is buffered in SADRmn and transferred to SADRmn when the new DMA transaction is started at 2). At this time, transfer count tc2 is also transferred to CHSRmn.TCOUNT. Pls. note that the shadow address register is only reset by hardware to 0000 0000_H as shown in this example, if the write enable bit is set to 0 (ADRCRmn.SHWEN = 0).

11.2.4.2 DMA Channel Request Control

Figure 11-6 shows the control logic for DMA requests that is implemented for each DMA channel.

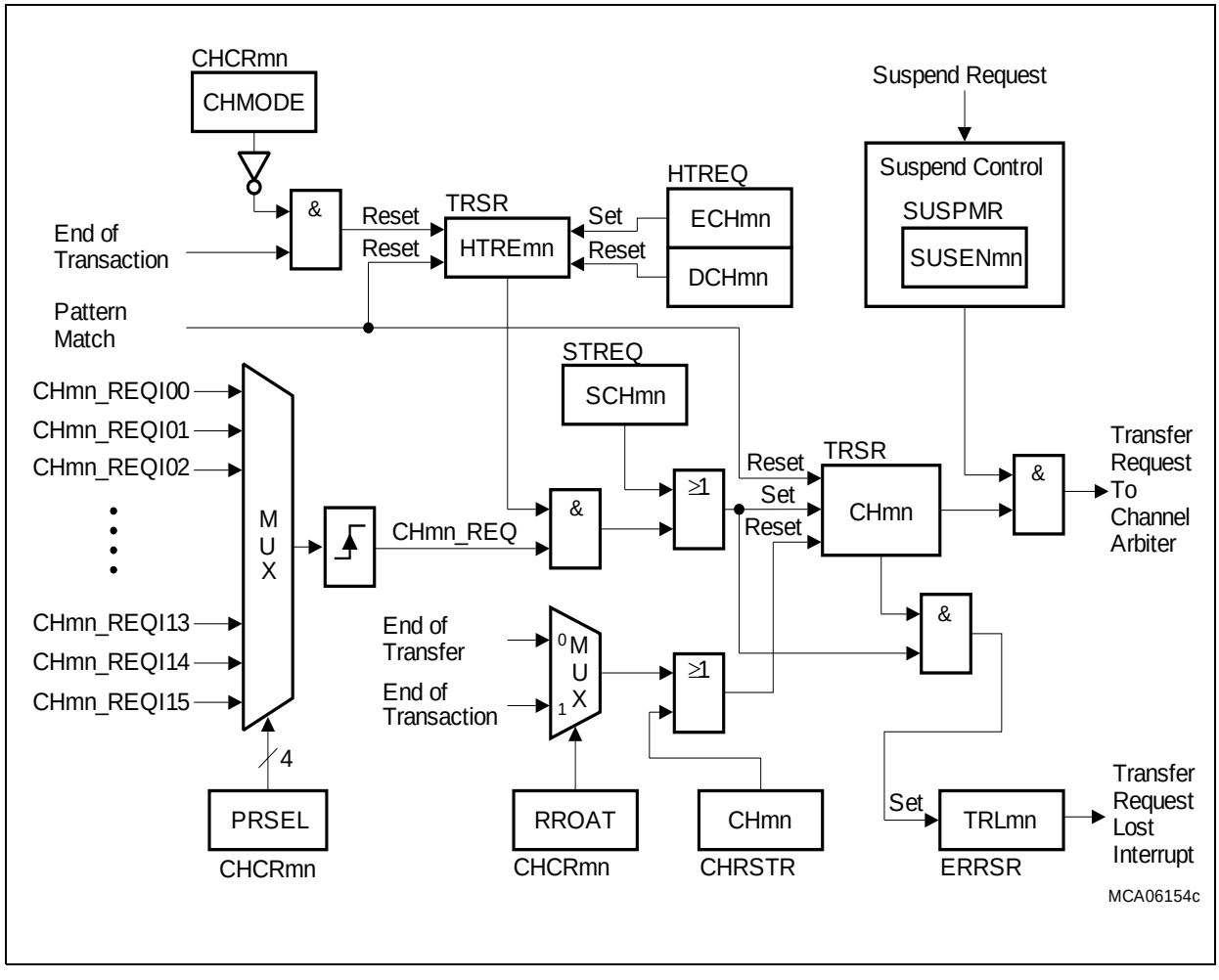


Figure 11-6 Channel Request Control (m = 0-1)

Two different types of DMA requests are possible:

- Hardware DMA requests
- Software DMA requests

The hardware request CHmn_REQ can be connected to one of sixteen possible hardware request input lines as selected by bit field CHCRmn.PRSEL. The hardware request input structure for CHCRmn.PRSEL includes a ´positive edge detector´ as the DMA channels requires single pulse requests.

Hardware requests are enabled/disabled by status bit TRSR.HTREmn. HTREmn can be set/reset by software or by hardware in Single Mode at the end of a DMA transaction. A software request can be generated by setting bit STREQ.SCHmn.

Status flag TRSR.CHmn indicates whether or not a software or hardware generated DMA request for DMA channel mn is pending. TRSR.CHmn can be reset by software or by hardware at the end of a DMA transfer (RROAT = 0) or at the end of a DMA transaction (RROAT = 1).

If a software or a hardware DMA request is detected for channel mn while TRSR.CHmn is set, a request lost event occurs. This error event indicates that the DMA is already processing a transfer and that another transfer has been requested before the end of the previous one. In this case, bit ERRSR.TRLmn will be set and a transfer lost interrupt can be generated.

11.2.4.3 DMA Channel Operation Modes

The operation mode of a DMA channel is individually programmable for each DMA channel mn. Basically, a DMA channel can operate in the following modes:

- Software controlled mode
- Hardware controlled mode, in Single or Continuous Mode

In software-controlled mode, a DMA channel request is generated by setting a control bit. In hardware-controlled mode, a DMA channel request is generated by request signals typically generated by on-chip peripheral units.

In hardware-controlled Single Mode, a DMA channel mn becomes disabled by hardware after the last DMA transfer of its DMA transaction. In hardware-controlled Continuous Mode, a DMA channel mn remains enabled after the last DMA transfer of its DMA transaction.

In hardware- and software-controlled mode, a DMA request signal can be configured to trigger a complete DMA transaction or one single transfer.

Software-controlled Modes

In software-controlled mode, one software request starts one complete DMA transaction or one single DMA transfer. Software-controlled modes are selected by writing HTREQ.DCHmn = 1. This forces status flag TRSR.HTREmn = 0 (hardware request of DMA channel mn is disabled).

The software-controlled mode that initiates one complete DMA transaction to be executed is selected for DMA channel mn by the following write operations:

- CHCRmn.RROAT = 1
- STREQ.SCHmn = 1

Setting STREQ.SCHmn to 1 (this is the software request) causes the DMA transaction of DMA channel mn to be started and TRSR.CHmn to be set. At the start of the DMA transaction, the value of CHCRmn.TREL is loaded into CHSRmn.TCOUNT (transfer count or tc) and the DMA transfers are executed. After each DMA transfer, TCOUNT becomes decremented and next source and destination addresses are calculated.

When TCOUNT reaches the 0, DMA channel mn becomes disabled and status flag TRSR.CHmn is reset. Setting STREQ.SCHmn again starts a new DMA transaction of DMA channel mn with the parameters as actually defined in the channel register set.

The software-controlled mode that initiates a single DMA transfer to be executed is selected for DMA channel mn by the following write operations:

- CHCRmn.RROAT = 0
- STREQ.SCHmn = 1, repeated for each DMA transfer

When CHCRmn.RROAT = 0, TRSR.CHmn becomes reset after each DMA transfer of the DMA transaction and a new software request (writing STREQ.SCHmn = 1) must be generated for starting the next DMA transfer.

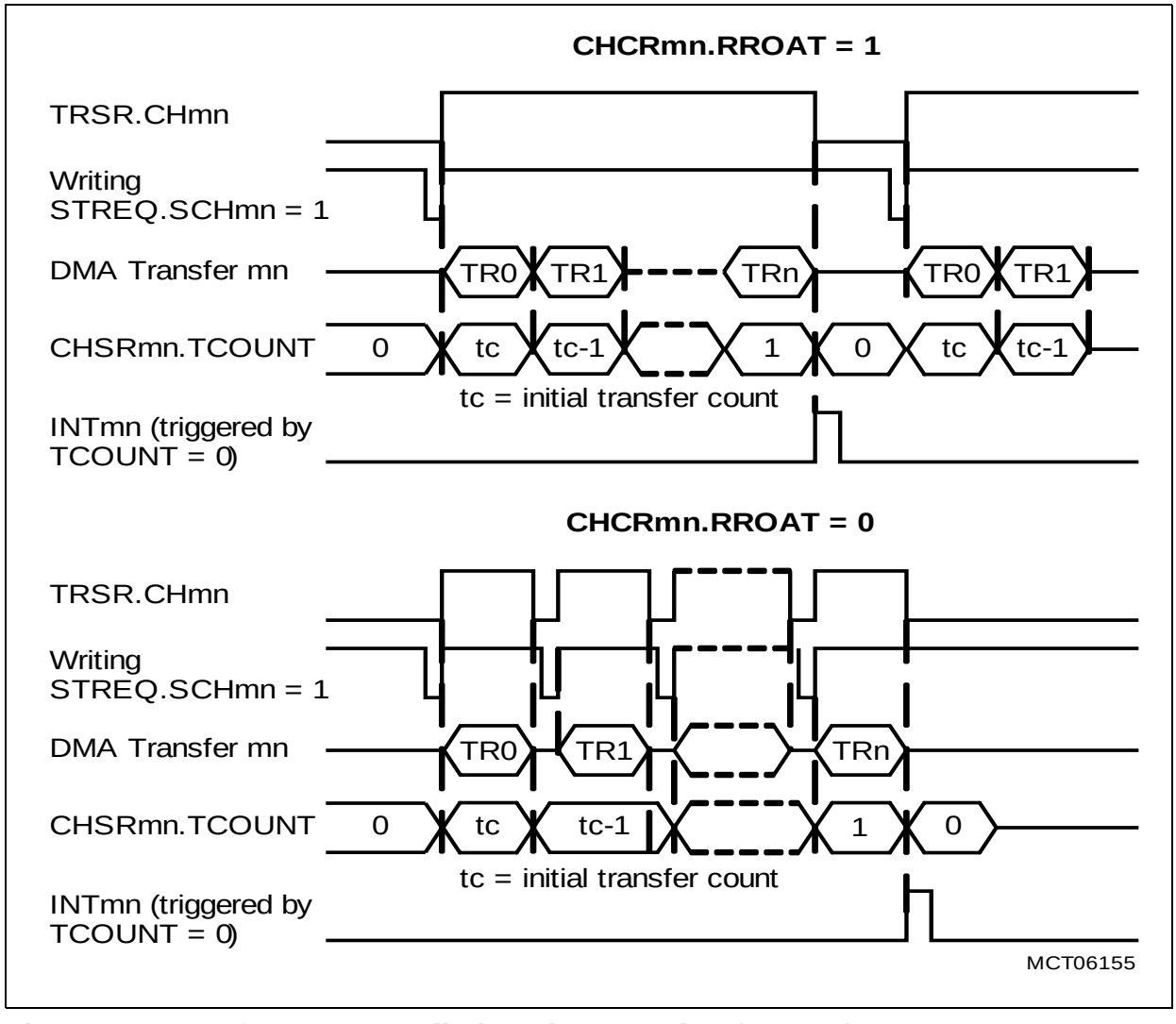


Figure 11-7 Software Controlled Mode Operation (m = 0-1)

Hardware-controlled Modes

In hardware-controlled modes, a hardware request signal starts a DMA transaction or a single DMA transfer. There are two hardware-controlled modes available:

- **Single Mode:**
  Hardware requests are disabled by hardware after a DMA transaction
- **Continuous Mode:**
  Hardware requests are not disabled by hardware after a DMA transaction

Hardware-controlled Single Mode

In hardware-controlled Single Modes, one hardware request starts one complete DMA transaction or one single DMA transfer. The hardware-controlled Single Mode that initiates one complete DMA transaction to be executed for DMA channel mn is selected by the following operations:

- CHCRmn.CHMODE = 0
- CHCRmn.RROAT = 1
- Selecting one of the sixteen hardware request inputs via CHCRmn.PRSEL
- HTREQ.ECHmn = 1

Setting HTREQ.ECHmn to 1 causes the hardware request CHmn_REQ of channel mn to be enabled (TRSR.HTREmn = 1). Whenever the hardware request CHmn_REQ becomes active, the value of CHCRmn.TREL is loaded into CHSRmn.TCOUNT and the DMA transaction is started by executing its first DMA transfer. After each DMA transfer, TCOUNT becomes decremented and next source and destination addresses are calculated. When TCOUNT reaches the 0, DMA channel 0n becomes disabled and status flags TRSR.CHmn and TRSR.HTREmn are reset. In order to start a new hardware-controlled DMA transaction, hardware requests must be enabled again by setting TRSR.HTREmn through HTREQ.ECHmn = 1. The hardware request disable function in Single Mode is typically needed when a reprogramming of the DMA channel register set (addresses, transfer count) is required before the next hardware triggered DMA transaction is started.

The hardware-controlled Single Mode in which each single DMA transfer has to be requested by a hardware request signal is selected as described above, with one difference:

- CHCRmn.RROAT = 0

In this operation mode, TRSR.CHmn becomes reset after each DMA transfer of the DMA transaction, and a new hardware request at CHmn_REQ must be generated for starting the next DMA transfer.

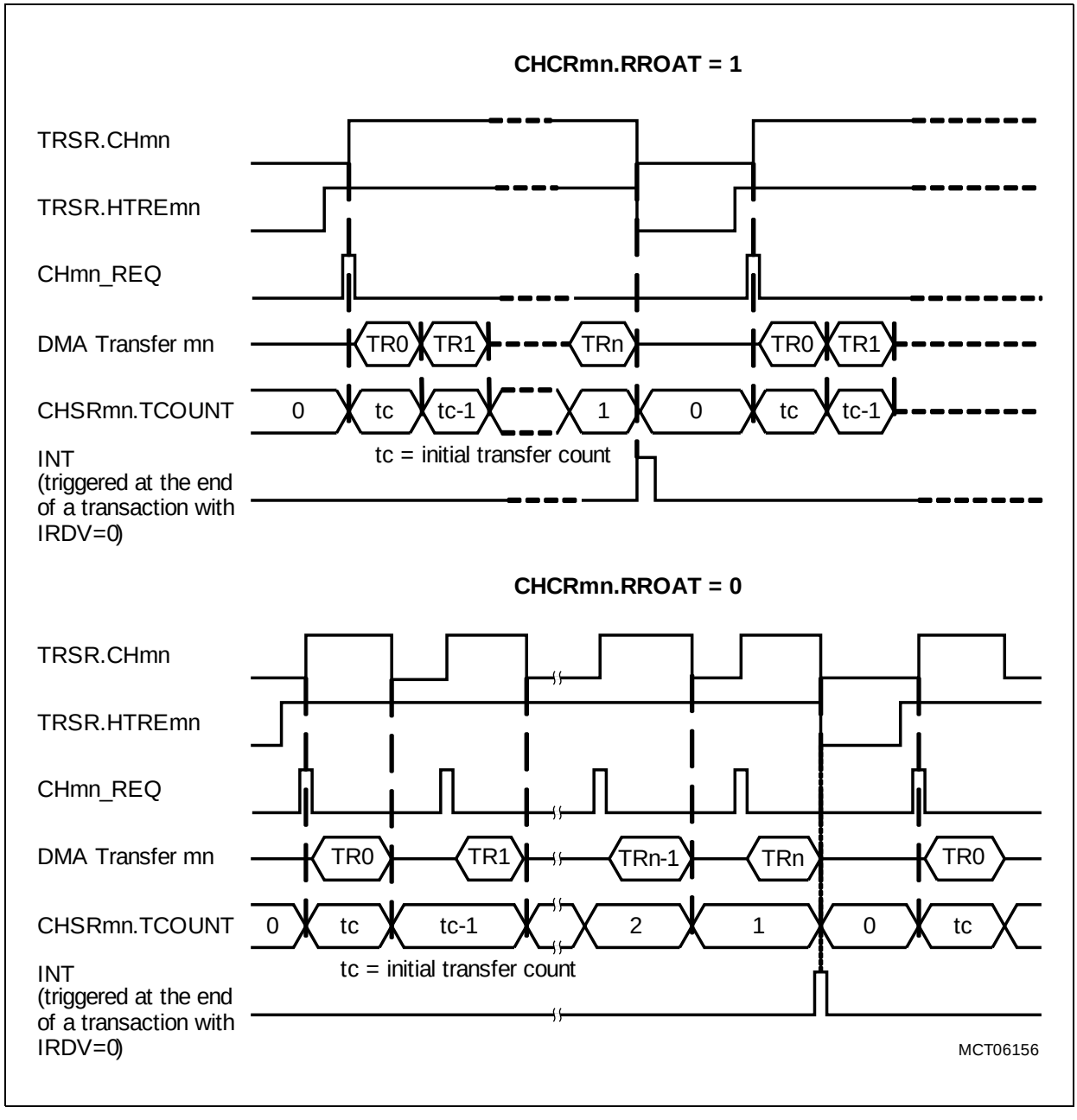


Figure 11-8 Hardware-controlled Single Mode Operation (m = 0-1)

Hardware-controlled Continuous Mode

In hardware-controlled Continuous Mode (CHCRmn.CHMODE = 1), the hardware transaction request enable bit HTREmn is not reset at the end of a DMA transaction. A new transaction of DMA channel mn with the parameters actually stored in the channel register set of DMA channel mn is started each time when CHSRmn.TCOUNT = 0 at the end of the DMA transaction. No software re-enable for a hardware request at CHmn_REQ is required.

Combined Software/Hardware-controlled Mode

Figure 11-9 shows how software- and hardware-controlled modes can be combined. In the example, the first DMA transfer is triggered by software when setting STREQ.SCHmn. Hardware requests are still disabled. After hardware requests have been enabled by setting HTREQ.ECHmn, subsequent DMA transfers are triggered now by hardware request coming from the CHmn_REQ line.

In the example, DMA channel mn operates in Single Mode (CHCRmn.CHMODE = 0). In this mode, TRSR.HTREmn becomes reset by hardware when CHSRmn.TCOUNT = 0 at the end of the DMA transaction.

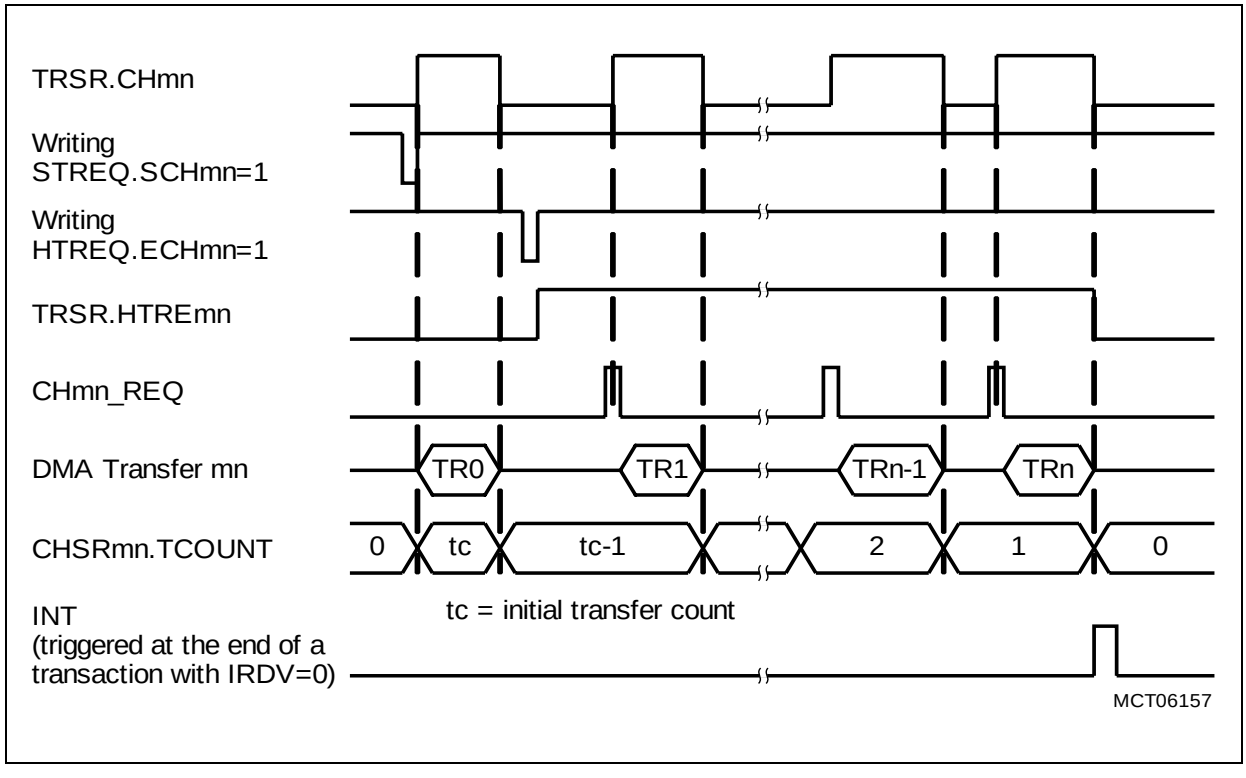


Figure 11-9 Transaction Start by Software, Continuation by Hardware (m = 0-1)

11.2.4.4 Error Conditions

The bus error flag ERRSR.FPIER indicates an FPI Bus error (SPB) that occurred during a source move (read or write) of a DMA module transaction. The bus error flag ERRSR.LMBER indicates an LMB Bus error that occurred during a source move (read or write) of a DMA module transaction.

The source error flags ERRSR.MEmSER indicates than an error occurred during source move (read) of a DMA transaction of DMA Sub-Block m.

The destination error flags ERRSR.MEmDER indicates than an error occurred during destination move (write) of a DMA transaction of DMA Sub-Block m

The transaction lost error flag ERRSR.TRLmn indicates if a DMA request for a DMA channel mn has been lost.

In the case of a read error, the write action is not executed, but the destination address is updated.

In the case of multiple errors, the error bits are set according to the error situations. This means that more than bus error flag can be set and that source/destination flags can be set.

11.2.4.5 Channel Reset Operation

A DMA transaction of DMA channel mn can be stopped (channel is reset) by setting bit CHRSTR.CHmn. When a read or write On Chip Bus transaction of DMA channel mn is executed at the time when CHRSTR.CH0n is set, this On Chip Bus transaction is finished normally. This behavior guarantees data consistency.

When CHRST.CHmn is set to 1:

- Bits TRSR.HTREmn, TRSR.CHmn, ERRSR.TRLmn, INTSR.ICHmn, INTSR.IPMmn, WRPSR.WRPDmn, WRPSR.WRPSmn, CHSRmn.LXO, and bit field CHSRmn.TCOUNT are reset.
- Source and destination address register will be set to the wrap boundary. SHADRmn will be cleared.
- All automatic functions are stopped for channel mn.

A user program must execute the following steps for resetting a DMA channel:

1. If hardware requests are enabled for the DMA channel mn, disable the DMA channel mn hardware requests by setting HTREQ.ECHmn = 0.
2. Writing a 1 to CHRST.CHmn.
3. Waiting (polling) until CHRST.CHmn = 0.

A user program should execute the following steps for restarting a DMA channel after it was reset:

1. Optionally (re-)configuring the address and other channel registers.
2. Restarting the DMA channel mn by setting HTREQ.ECHmn = 1 for hardware requests or STREQ.SCHmn = 1 for software requests.

The value of CHCRmn.TREL is copied to CHSRmn.TCOUNT when a new DMA transaction is requested and shadow address register contents is not equal $0000\ 0000_H$.

11.2.4.6 Transfer Count and Move Count

The move count determines the number of moves (consisting of one read and one write each) to be done in each transfer. It allows the user to indicate to the DMA the number of moves to be done after one request. The number of moves per transfer is selected by the block mode settings (CHCRmn.BLKM).

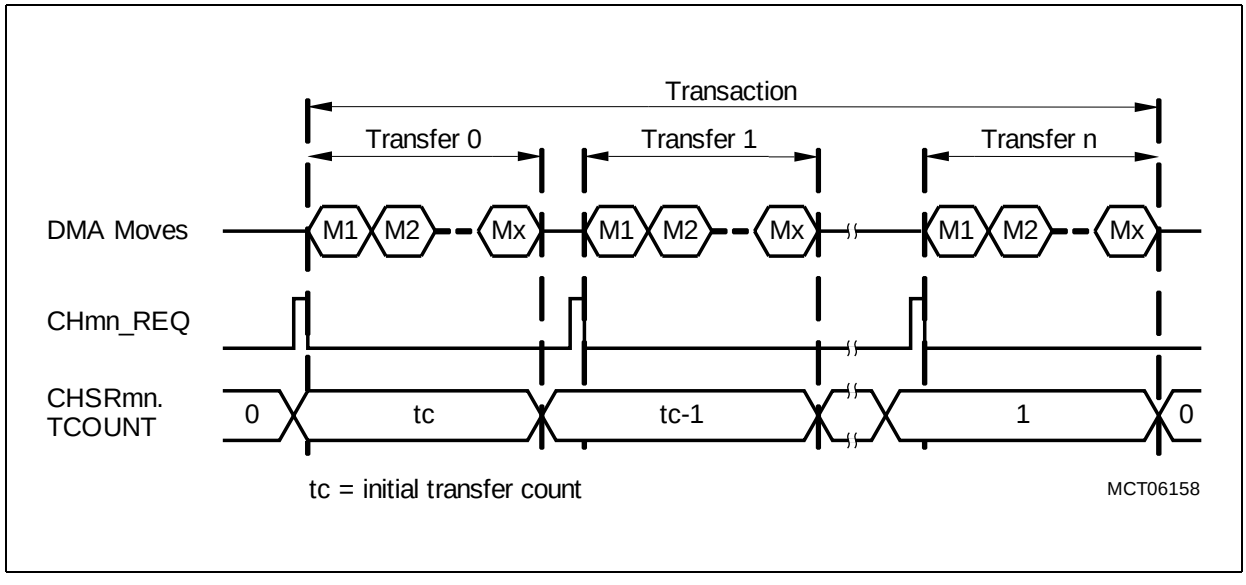


Figure 11-10 Transfer and Move Count (m = 0-1)

After a DMA move, the next source and destination addresses are calculated. Source and destination addresses are calculated independently of each other. The following address calculation parameters can be selected:

- The address offset, which is a multiple of the selected data width
- The offset direction: addition, subtraction, or none (unchanged address)

Control bits in address control register ADRCRmn determine how the addresses are incremented/decremented. Further, the data width as defined in CHCRmn.CHDW is taken into account for the address calculation.

Figure 11-11 and **Figure 11-12** show two examples of address calculation. In both examples, a data width of 16-bit (CHCRmn.CHDW = 01_B) is assumed.

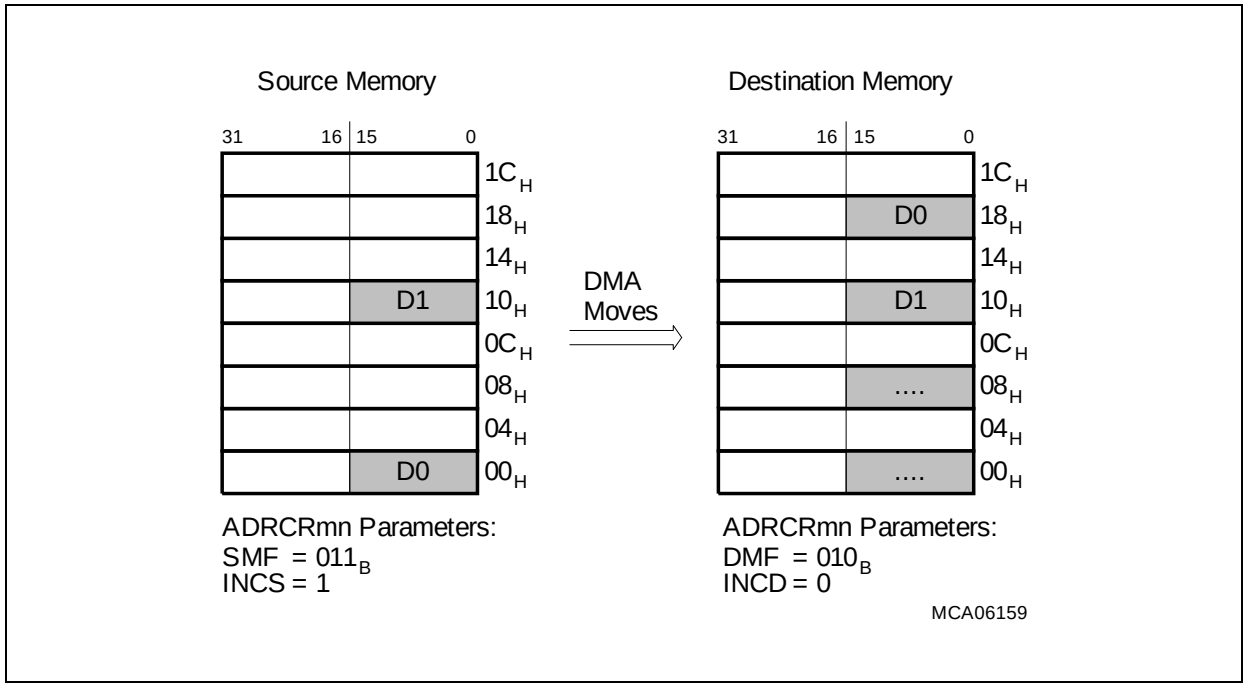


Figure 11-11 Programmable Address Modification - Example 1 (m = 0-1)

In **Figure 11-11**, 16-bit half-words are transferred from a source memory with an incrementing source address offset of 10_H to a destination memory with decrementing destination addresses offset of 08_H.

In **Figure 11-12**, 16-bit half-words are transferred from a source memory with an incrementing source address offset of 02_H to a destination memory with incrementing destination addresses offset of 04_H.

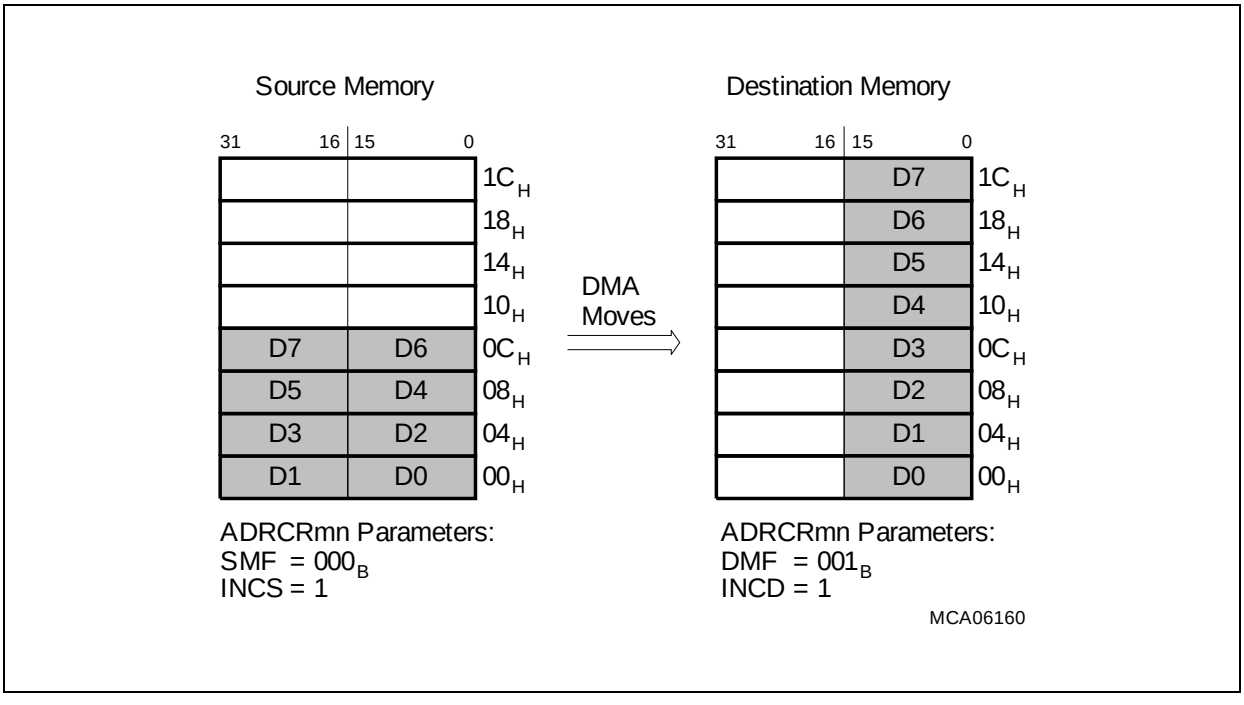


Figure 11-12 Programmable Address Modification - Example 2 (m = 0-1)

11.2.4.7 Circular Buffer

Destination and source address can be configured to build a circular buffer separately for source and destination data. Within this circular buffer, addresses are updated as defined in **Figure 11-11** and **Figure 11-12** with a wrap-around at the buffer limits. The circular buffer length is determined by bit fields ADRCRmn.CBLS (for the source buffer) and ADRCRmn.CBLD (for the destination buffer). These 4-bit wide bit fields determine which bits of the 32-bit address remain unchanged at an address update. Possible buffer sizes of the circular buffers can be 2^{CBLS} or 2^{CBLD} bytes (= 1, 2, 4, 8, 16, … up to 32k bytes).

When source or destination addresses are updated (incremented or decremented) after a DMA move, all upper bits [31:CBLS] of source address and [31:CBLD] of destination address are frozen and remain unchanged, even if a wrap-around from the lower address bits [CBLS:0] or [CBLD:0] occurred. This address-freezing mechanism always causes the circular buffers to be aligned to a multiple integer value of its size.

If the circular buffer size is less or equal than the selected address offset (see **Table 11-10**), the same circular buffer address will always be accessed.

11.2.5 Transaction Control Engine

Each DMA Sub-Block has a Transaction Control Unit. The Transaction Control Unit in the DMA Sub-Block, as shown in the DMA Controller block diagram in **Figure 11-1**, contains a Channel Arbiter and a Move Engine.

The Channel Arbiter arbitrates the transfer requests of the DMA channels, and submits the transfers parameters of the DMA channel with the highest channel priority that are needed for a DMA transfer to the Move Engine. DMA channels within a DMA Sub-Block have a two-level programmable channel priority as defined by bit CHCRmn.CHPRIO. When two transfer requests of two different DMA channels with identical channel priority become active at the same time, the DMA channel with the lowest channel number (n) is serviced first.

The Move Engine handles the execution of a DMA transfer that has been detected by the Channel Arbiter to be the next one. The Move Engine requests the required buses and loads or stores data according to the parameters of a DMA transfer. It is able to wait if a targeted bus is not available. In the Move Engine, a DMA transfer of a DMA transaction cannot be interrupted and always get finished. This means that a DMA transfer, which can also be composed of several data moves (read move and write move), cannot be interrupted by a transfer of another DMA channel.

After a DMA transfer is finished, the Move Engine will send back the actualized address register information to the related DMA channel. Possible error conditions are also reported.

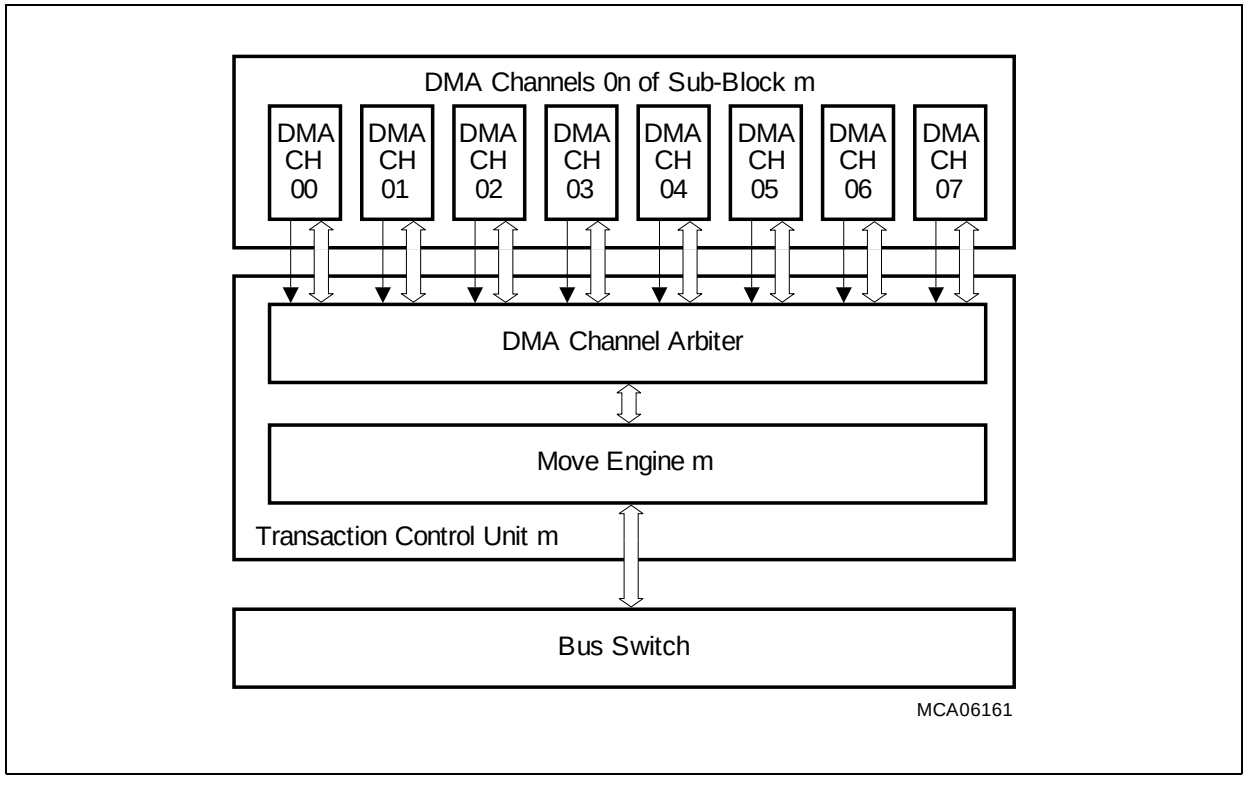


Figure 11-13 Transaction Control Engine (m = 0-1)

11.2.6 Bus Switch, Bus Switch Priorities

The Bus Switch of the DMA controller provides the connection from the DMA Sub-Blocks to the two On Chip Bus master interfaces (connected to System Peripheral Bus and LMB Bus) and to the DMA Peripheral Interface (see **Figure 11-14**).

The FPI Bus interface of the DMA includes a slave interface which provides the access to the DMA and the peripherals connected to the DMA Peripheral Interface (MLI, Cerberus and Memory Checker modules).

The LMB Bus interface of the DMA is a master interface.

The DMA module, the DMA Sub-Blocks as well as the MLI, the Memory Checker and the Cerberus module working frequencies are identical to the FPI Bus frequency. The working frequency of the LMB master interface is identical to the LMB/CPU working frequency.

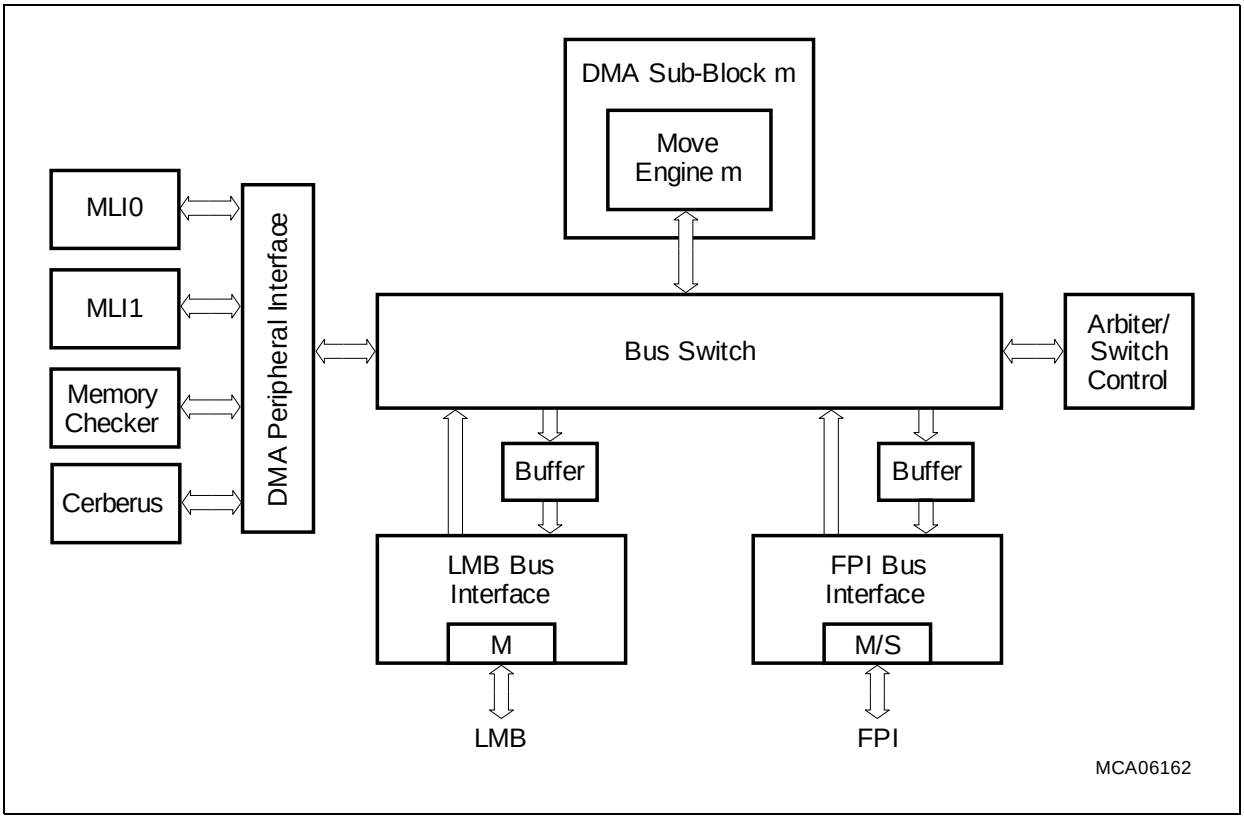


Figure 11-14 Bus Switch

One access can be buffered in the bus interfaces.

Note: The accesses of the DMA Move Engine's bus interfaces to the On Chip Bus interfaces are always done in Supervisor Mode.

The arbiter/switch control unit arbitrates the requests from the connected active interfaces (FPI Bus Interface, DMA Move Engines, MLI, Cerberus, ...) and grants the buses connected to the switch for data transfers. **Table 11-1** and **Table 11-2** are defining the Bus Switch priorities for requests to the same On Chip Bus Interface. The arbitration scheme is valid in case of a collision of requests from active peripherals connected to the DMA Bus Switch (Move Engine, MLI, Cerberus) for the same resource (FPI Bus Interface, LMB Bus Interface, DMA Peripheral Interface, Move Engine). The arbitration is done for each Bus Switch request.

The general overview of the Bus Switch Priorities is given in **Table 11-1**. Additional detailed information about the Move Engine priorities for concurrent on the Bus Switch is described in **Table 11-2**.

In case of a collision of the DMA Move Engine requests on the DMA Bus switch for the the same resource, a Move Engine Write has priority over a Move Engine Read. In case of a collision of both Move Engines with concurrent read requests or concurrent write requests for the same resource, the Move Engine number together with the

CHCRmn.DMAPRIO value determines the priority on the DMA Bus Switch (see **Table 11-2**).

Table 11-1 DMA Bus Switch Priorities

Priority	Agent Requests	Comment
Highest	Cerberus to On Chip Bus High	Priority selection by software in Cerberus.
	FPI Bus to DMA Peripheral Interface accesses	Reason: minimizing wait states on the FPI Bus.
	DMA Move Engine Write	The detailed Bus Switch priorities for the two Engines with concurrent reads or concurrent writes are listed in **Table 11-2**.
	DMA Move Engine Read	The detailed Bus Switch priorities for the two Engines with concurrent reads or concurrent writes are listed in **Table 11-2**.
	MLI0 access	–
Lowest	Cerberus to On Chip Bus Low	Priority selection by software in Cerberus.

Table 11-2 DMA Bus Switch Priorities of DMA Move Engines

Priority	DMA Move Engine Request	Comment
Highest	Move Engine 0 CHCR0x.DMAPRIO = “11” or CHCR0x.DMAPRIO = “01”	(x=7-0). x reflects the channel of Move 0 that won the Move Engine internal arbitration.
	Move Engine 1 CHCR1y.DMAPRIO = “11” or CHCR1y.DMAPRIO = “01”	(y=7-0). y reflects the channel of Move 1 that won the Move Engine internal arbitration.
	Move Engine 0 CHCR0x.DMAPRIO = “00”	(x=7-0). x reflects the channel of Move 0 that won the Move Engine internal arbitration.
Lowest	Move Engine 1 CHCR1x.DMAPRIO = “00”	(y=7-0). y reflects the channel of Move 1 that won the Move Engine internal arbitration.

11.2.7 DMA Module Priorities on On Chip Busses (FPI Bus, LMB Bus)

Every active peripheral connected to the DMA Bus Switch that requests for access to FPI Bus or LMB Bus has to go through two arbitration stages before accessing the On Chip Bus: DMA Module internal arbitration at the DMA Bus Switch and DMA Module external arbitration at the On Chip Bus.

The DMA Module is connected to the FPI Bus and to the LMB Bus with master interfaces. The DMA LMB Master and the DMA FPI Master is each connected with three priorities to its On Chip Bus (low, medium and high priority), where it competes against the other bus masters connected to the On Chip Bus for bus access. The mapping of the Move Engines and the peripherals connected to the DMA Peripheral Interface to the DMA Module priorities on the FPI Bus and on the LMB Bus is described in **Table 11-3**.

The MLI modules are mapped to the low priority On Chip Bus requests of the DMA Module, while the mapping of the Cerberus and the Move Engines to the On Chip Bus requests is selected by software (control register bits).

The complete list of FPI master priorities can be found in the FPI Bus Control Unit Chapter.

The complete list of LMB master priorities can be found in the Local Memory Bus Controller Unit Chapter.

Table 11-3 DMA Module Priorities on On Chip Busses

On Chip Bus Priority	DMA On Chip Bus Request	Comment
High	Cerberus High	Priority selection by SW in Cerberus.
	Move Engine m: CHCRmx.DMAPRIO = “11”	Priority selection by SW in Move Engine. (x=7-0) (m=1-0)
Medium	Move Engine m: CHCRmx.DMAPRIO = “01”	Priority selection by SW in Move Engine. (x=7-0) (m=1-0)
Low	Move Engine m: CHCRmx.DMAPRIO = “00”	Priority selection by SW in Move Engine. (x=7-0) (m=1-0)
	MLI0	-
	Cerberus Low	Priority selection by SW in Cerberus.

11.2.8 DMA Module: On Chip Bus Access Rights, RMW support

All accesses triggered by the DMA Move Engines, the MLI modules or the Cerberus module are always done in SV mode.

The DMA module does not support read/modify write instructions to the peripherals connected to the DMA Peripheral Interface (the MLI, Memory Checker and Cerberus modules).

11.2.9 DMA Module On Chip Bus Master Interfaces

This chapter describes the features of the DMA On Chip Bus Master Interfaces to the FPI Bus and to the LMB Bus.

The DMA FPI master interface supports:

- single data read and write transactions (8bit, 16bit, 32bit)
- generation of pipelined FPI transactions from different sources (Move Engines, Cerberus, MLI)
- de-assertion of request after retry in order to prevent bus blocking.
- out of order transactions from different sources in order to avoid side effects (blocking) between the different sources (Move Engines, Cerberus, MLI)
- three dedicated FPI requests (medium, low, high priority. See **Table 11-3**)[1].

A single move engine supports only one transaction at a time. Due to the fact that the move engines do generate read - write sequences, it is unlikely that the DMA module generates permanent, pipelined, high priority requests.

The DMA LMB master interface supports:

- single data read and write transactions (8bit, 16bit, 32bit)
- single data read transactions (64bit) for read accesses to Segment 8 (cached area)
- pipelined transactions from different sources (Move Engines, Cerberus, MLI)
- de-assertion of request after retry in order to prevent bus blocking.
- out of order transactions from different sources in order to avoid side effects (blocking) between the different sources (Move Engines, Cerberus, MLI)
- three dedicated LMB requests (medium, low, high priority. See **Table 11-3**)[2].

A single move engine supports only one transaction at a time. Due to the fact that the move engines do generate read - write sequences, it is unlikely that the DMA module generates permanent, pipelined, high priority requests.

DMA LMB Master Read Buffer:

The DMA LMB master interface includes a 64bit buffer for read accesses to cached addresses (Segment 8). The DMA LMB Master Interface contains a data read buffer for read accesses to cached addresses. The read buffer allows to read one line of 8byte (=64 bit) of data read from specific memory areas on LMB side (Segment 8: $8000\ 0000_H$ - $8FFF\ FFFF_H$)

A read request to an Segment 8 address (8bit, 16bit or 32bit) will be translated by the DMA LMB master interface into an LMB 64 bit single data read. The DMA LMB master will forward the requested 8bit, 16bit or 32bit data to the DMA Bus switch and save the 64bit read data together with the related 64bit aligned address in the DMA LMB master read buffer. If the next and subsequent read access to a segment 8 address is identical (64bit aligned) to the actual read buffer contents, the requested read data will be read from the read buffer by the DMA LMB master instead of reading it from the LMB bus.

1) The complete list of FPI master priorities can be found in the FPI Bus Control Unit Chapter.

2) The complete list of LMB master priorities can be found in the Local Memory Bus Controller Unit Chapter.

If the next read to a read from a segment 8 address is not identical (64bit aligned) to the actual read buffer contents, the contents of the read buffer is invalidated. A 64bit LMB read is generated by the DMA LMB master interface, the requested 8bit, 16bit or 32 bit data is forwarded to the DMA Bus switch and the read buffer is updated with the new 64bit data and its related address.

A DMA write to a segment 8 address (8bit, 16bit, 32bit write, 64bit write is not supported) invalidates the read buffer.

11.2.10 DMA Module Bridge Functionality

The DMA module includes bridge functionality:

- from the FPI Bus to the DMA Peripheral Interface
- from the DMA Peripheral Interface to the FPI Bus and LMB Bus.

FPI Bus -> LMB Bus

The DMA module does not forward transaction from the FPI Bus to the LMB Bus. The DMA module does not support / include bridge functionality between FPI Bus and LMB Bus.

FPI Bus -> DMA Peripheral Interface (MLI, Memory Check, Cerberus, ...)

The DMA module forwards transactions from the FPI bus to the DMA Peripheral Interface (FPI -> MLI, Memory Check, Cerberus, ...). The identification of the target module on the DMA Peripheral Interface is done by address decoding.

DMA Peripheral Interface -> On Chip Bus (FPI Bus, LMB Bus)

The DMA module forwards transactions from the active modules that are connected to the DMA Peripheral Interface (Cerberus, MLI, ...) to the FPI Bus and LMB Bus. The identification of the target On Chip Bus is done by address decoding.

11.2.11 On-Chip Debug Capabilities

The DMA controller in the TC1797 provides some debugging capabilities. These debug features support:

- Soft-suspend Mode of DMA channels
- Break signal generation
- Trace signal generation

In Soft-suspend mode, the operations of DMA channels are stopped. Pending read or write transfers in the DMA module On Chip Bus Master Interfaces (LMB Master Interface, FPI Master Interface) are finished. Under certain conditions also a break signal is generated for the on-chip debug support logic. Further, DMA trace information can be output.

In Soft-suspend mode, the DMA module provides access to all control registers of the DMA module (incl. Move Engines and Memory Checker Module) and to the peripherals connected to the DMA Peripheral Interface.

11.2.11.1 Hard-suspend Mode

The Hard-Suspend Mode is controlled in the TC1797 DMA module CLC register but should not be used in order to guarantee access to the device via JTAG (Cerberus). Possible support of the Hard-suspend mode by the peripherals connected to the DMA Peripheral Interface is described in the related module chapters.

11.2.11.2 Soft-suspend Mode

The TC1797 on-chip debug control unit is able to generate a Soft-suspend Mode request (SUSREQ) for the DMA controller. When this soft-suspend request becomes active, the state of a DMA channel becomes frozen regarding hardware changes to ensure that the state of the DMA channels can be analyzed by reading the register contents. Pending read or write transfers in the DMA module On Chip Bus Master Interfaces (LMB Master Interface, FPI Master Interface) are finished. The DMA controller signals its soft suspend mode back to the on-chip debug control via an Soft-suspend acknowledge. The Soft-suspend acknowledge becomes active when all DMA channels mnn that are enabled for the Soft-suspend Mode have set its suspend active status flag SUSPMR.SUSACmn.

Soft-suspend Mode of DMA channel mn is entered if its suspend enable bit SUSENmn in the Suspend Mode Register SUSPMR is set. When SUSREQ becomes active, the operation of all DMA channels mnn that are enabled for Soft-suspend Mode is stopped automatically after its current DMA transfers have been finished in the transaction control unit. Afterwards, the suspend active status flag SUSPMR.SUSACmn is set, indicating that DMA channel mn is in Soft-suspend Mode. DMA channels that are disabled for Suspend Mode (SUSENmn = 0) continue with its normal operation.

In Soft-suspend Mode, register contents can be modified. These modifications are taken into account for further DMA transactions or DMA transfers of the related DMA channel

after Suspend Mode has been left again. Suspend Mode of DMA channel mn is left and its normal operation continues if either the SUSREQ signal becomes inactive, or if the enable bit SUSENmn is reset by software.

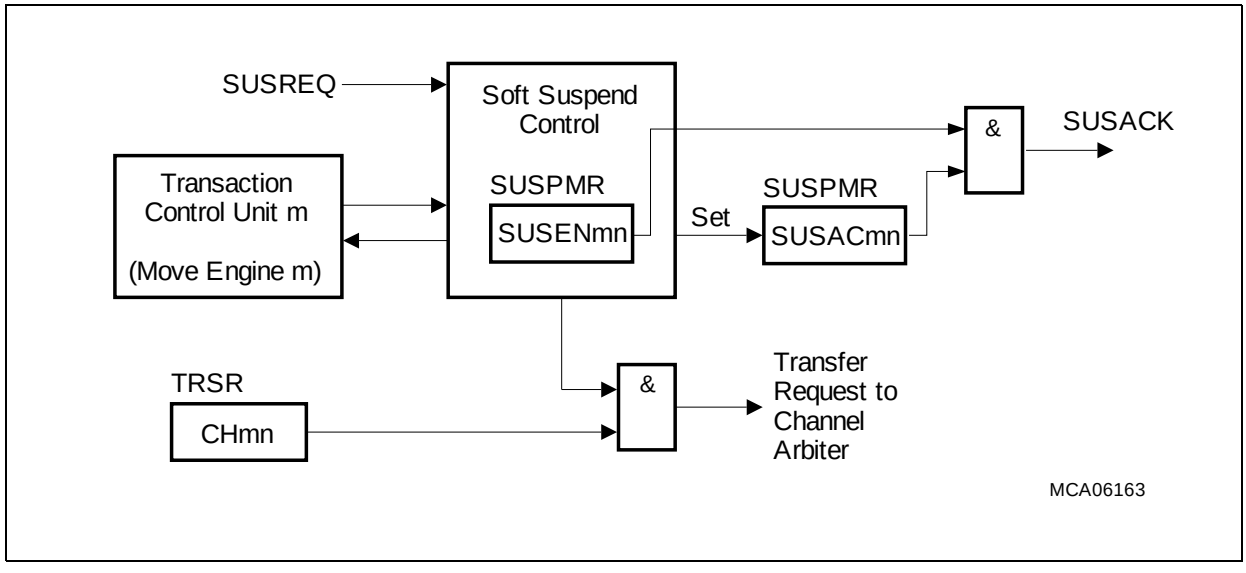


Figure 11-15 Soft-suspend Mode Control (m = 0-1)

11.2.11.3 Break Signal Generation

The DMA controller provides one BREAK output signal that is generated for the on-chip debug support logic (see **Figure 11-16**). The DMA sub-block is able to detect two break conditions:

- Transaction lost interrupt has occurred
- DMA request transitions, indicated by bits TRSR.CHmn

The output lines of the two break conditions in the DMA sub-block are OR-ed together to the BREAK output signal.

A transaction lost break condition occurs in DMA Sub-Block m whenever at least one of its eight transaction lost interrupts becomes active, and when enable bit OCDSR.BRL0 is set. The transaction lost interrupts do not generate a break condition if OCDSR.BRL0 = 0. Transaction interrupt control is described in **Section 11.2.12.2**.

The second break condition of DMA Sub-Block m becomes active when the transaction request bit TRSR.CHmn of one of its eight DMA channels n (as selected by OCDSR.BCHSn) indicates a transition of its state. The CHmn transition type (change from ´no request is pending´ to ´request is pending´, change from ´request is pending´ to ´no request is pending´, changes in both directions) is selected by bit field OCDSR.BTCRn.

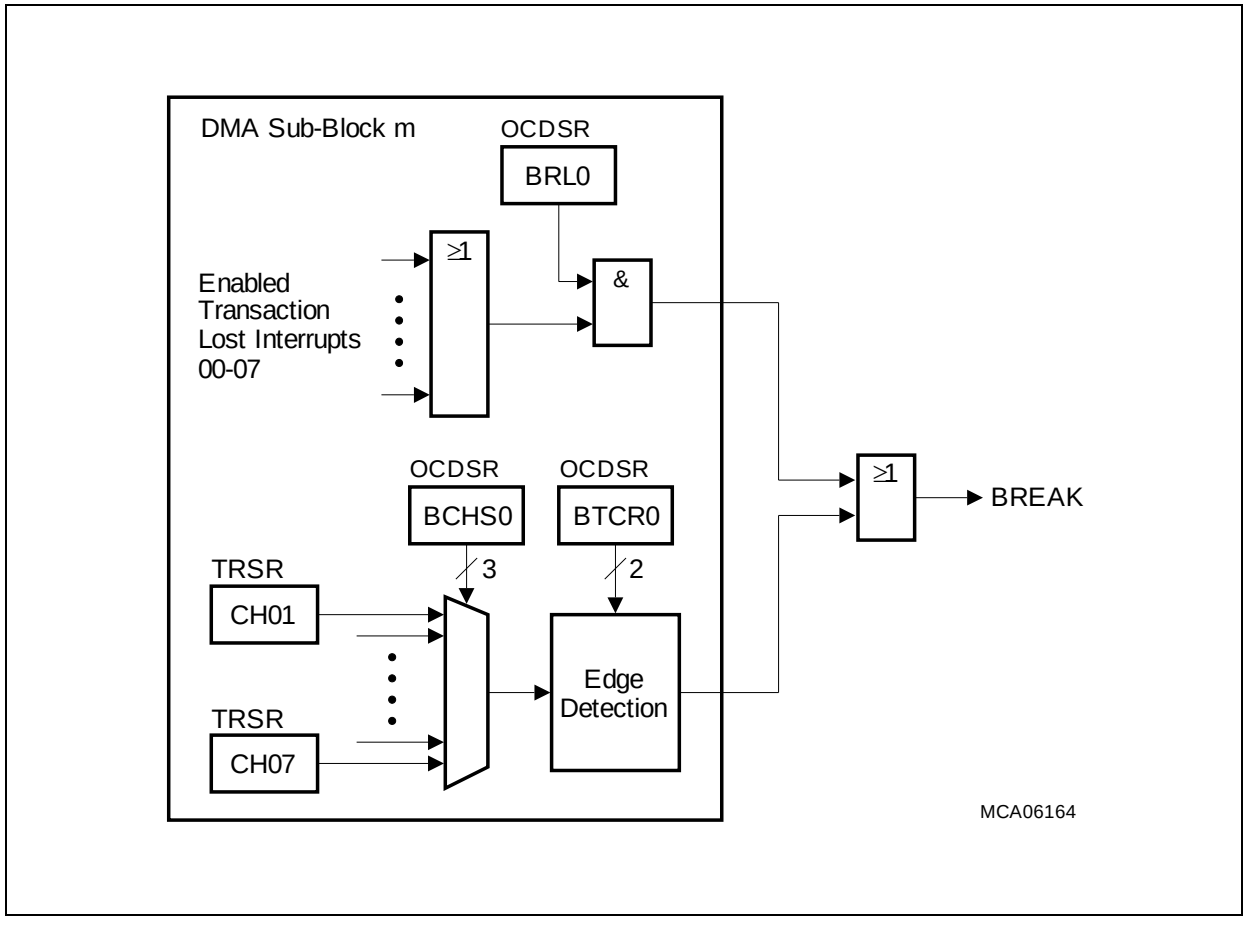


Figure 11-16 DMA Break Event Generation (m = 0-1)

11.2.12 Interrupts

The interrupt structure of the DMA controller is a very flexible control logic that allows an interrupt coming from an interrupt source within four interrupt source types to be connected to each of the sixteen interrupt outputs. This permits, for example, DMA channels that very rarely generate interrupts to share one interrupt node. The remaining interrupt nodes can be assigned to dedicated DMA channels to reduce the interrupt overhead for these channels. The four interrupt source types are:

- Channel interrupts
- Transaction lost interrupt
- Move Engine interrupts
- Wrap buffer interrupts

Some of the interrupt functions are common to all of the four interrupt source types. An interrupt event, internally generated as a request pulse, is always stored in an interrupt status flag. This interrupt status flag can be reset by software. Further, the interrupt event can be enabled or disabled. When an interrupt event is enabled, a 4-bit Interrupt Node Pointer determines which of the sixteen interrupt outputs will be activated.

The following sections describe each of the four interrupt source types in more detail.

11.2.12.1 Channel Interrupts

Each DMA channel mn has one associated channel interrupt. It can always be activated after a DMA transfer, or when CHSRmn.TCOUNT matches with the value of bit field CHSRmn.IRDV after it has been decremented after a DMA transfer. The pattern detection interrupts that are combined with the channel interrupts (one common Interrupt Node Pointer CHICRmn.INTP) are activated when the pattern detection interrupt of DMA channel mn becomes active (when enabled by CHCRmn.PATSEL not equal 00_B).

A channel interrupt of DMA channel mn is indicated when status flag INTSR.ICHmn is set. The status flags ICHmn and IPMmn can be reset together by software when setting bit INTCR.CICHmn (or CHRSTR.CHmn). The channel interrupt of DMA channel mn is enabled when bit CHICRmn.INTCT[1] is set. The channel interrupt pointer CHICRmn.INTP determines which of the interrupt outputs SR[15:0][1)] will be activated on an active channel interrupt or pattern detection interrupt. Note that the signal that is set signal for the ICHmn flag is available as CHmn_OUT signal at the DMA module boundary.

Bit CHICRmn.INTCT[0] selects these two types of interrupt sources. For the compare operation, bit field IRDV (4-bit) is zero-extended to 10-bit and then compared with the 10-bit TCOUNT value. This means that a TCOUNT match interrupt can be generated after one of the last 16 DMA transfers of a DMA transaction. Note that with

1) In the TC1797, only SR[7:0] are connected to interrupt nodes. SR[8:15] are used for DMA channel triggering/connections.

IRDV = 0000_B, the match interrupt is generated at the end of a DMA transaction (after the last DMA transfer).

The pattern detection interrupt is indicated when status flag INTSR.IPMmn is set. The status flags IPMmn and ICHmn can be reset together by software when setting bit INTCR.CICHmn (or CHRSTR.CHmn). The pattern detection interrupt of DMA channel mn is enabled when bit CHICRmn.PATSEL is set to a value not equal to 00_B. The channel interrupt pointer CHICRmn.INTP defines which of the interrupt outputs SR[15:0] will be activated on a pattern detection interrupt or the channel interrupt pointer CHICRmn.INTP determines which of the interrupt outputs SR[15:0][1] will be activated on a pattern detection or channel interrupt.

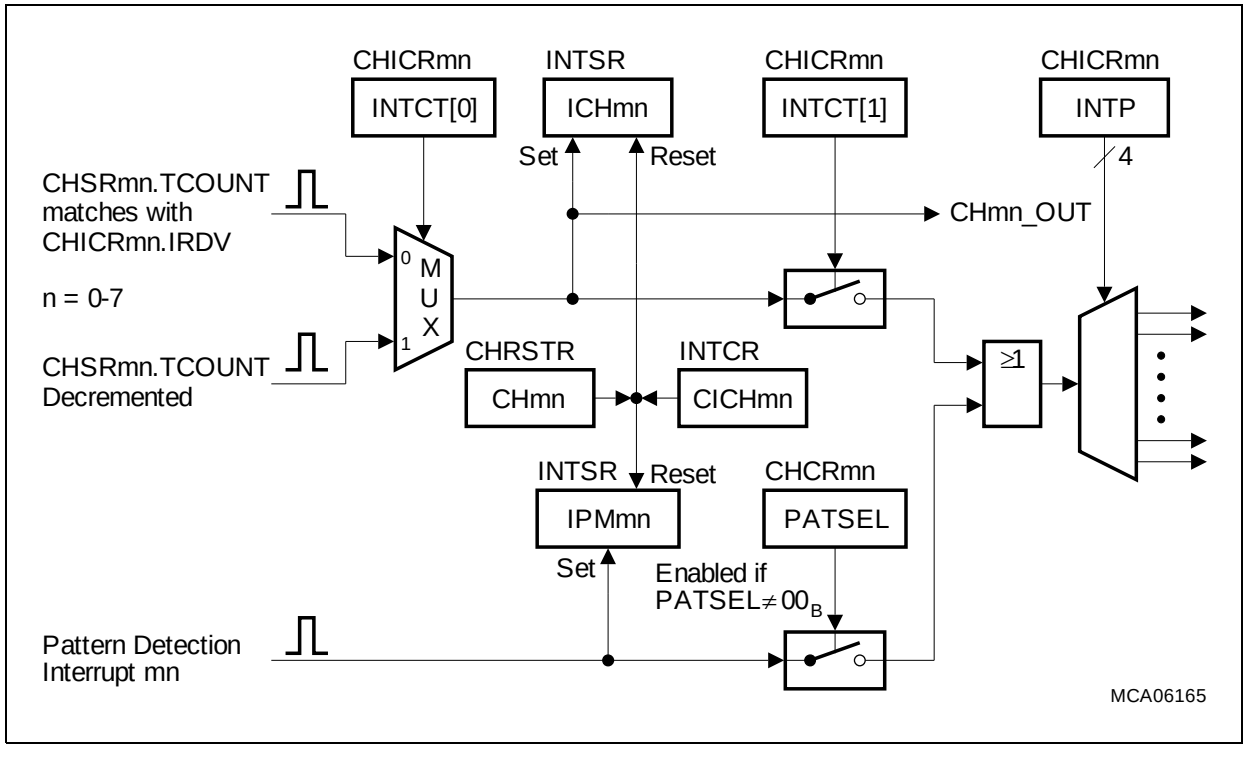


Figure 11-17 Channel Interrupts (m = 0-1)

1) In the TC1797, SR[7:0] are connected to interrupt nodes. SR[8:15] are used for DMA channel triggering/connections.

11.2.12.2 Transaction Lost Interrupt

Each DMA channel mn is able to detect a transaction request lost condition. This condition becomes true when a new hardware or software DMA request occurs while the previous transaction or transfer on DMA channel mn is not finished, indicated by TRSR.CHmn still set. If such a transaction request lost condition occurs, bit ERRSR.TRLmn is set. The transaction lost interrupts of all DMA channels are OR-ed together to one common transaction lost interrupt that can be directed to one of the interrupt outputs SR[15:0][1)] by setting the transaction lost interrupt pointer EER.TRLINP with a corresponding value.

A transaction request lost condition of DMA channel mn is indicated by status flag ERRSR.TRLmn, which can be reset by setting bit CLRE.CTLmn or CHRSTR.CHmn. The transaction lost interrupt for DMA channel mn is enabled when bit EER.ETRLmn is set.

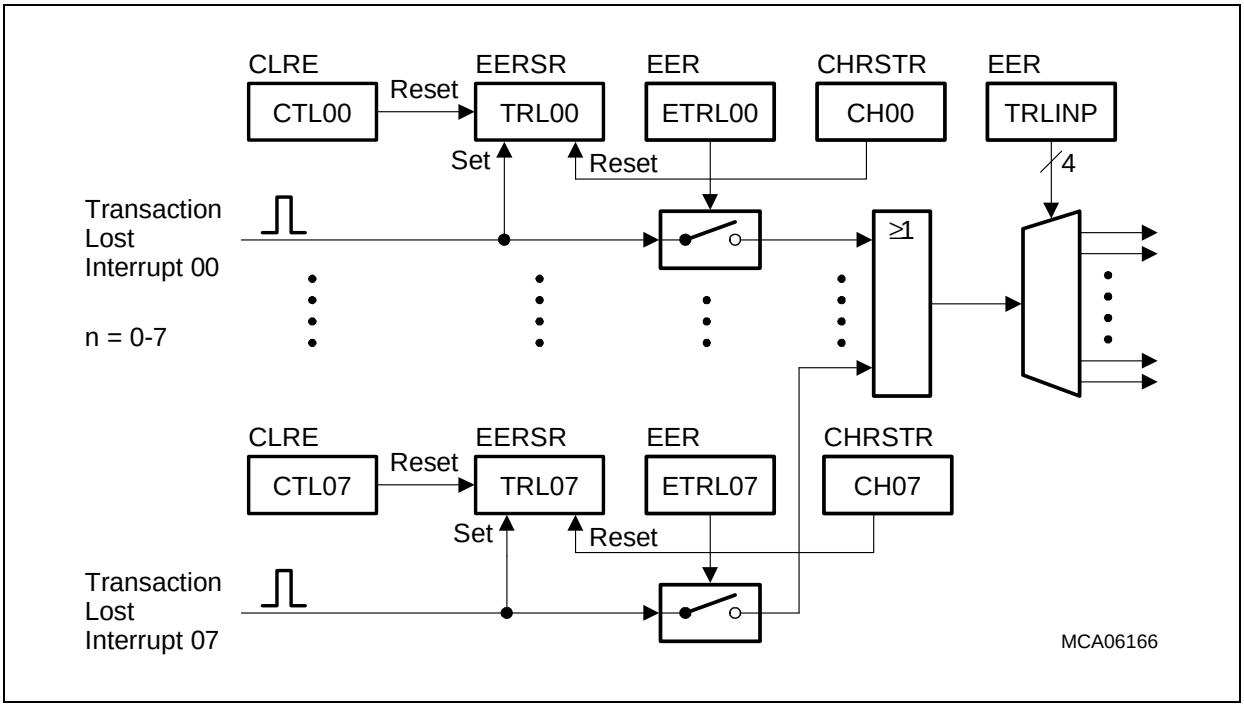


Figure 11-18 Transaction Lost Interrupt

1) In the TC1797 SR[7:0] are connected to interrupt nodes. SR[15:8] are used as DMA channel trigger signals.

11.2.12.3 Move Engine Interrupts

The Move Engine is able to detect error conditions that occur during accesses to the FPI Bus and LMB Bus interfaces of the Bus Switch (see **Figure 11-14**). Two error conditions can be detected:

- Source error
- Destination error

A source error indicates an FPI Bus or LMB Bus error that occurred during a read move from the data source. A destination error indicates an FPI Bus or LMB Bus error that occurred during a write move to the data destination.

A source error of Move Engine 0 is indicated by the status flag ERRSR.ME0SER. Status flag ME0SER can be reset by software when setting bit CLRE.CME0SER. The source error interrupt of Move Engine 0 is enabled when bit EER.EME0SER is set. Separate reset, status, and enable bits are available in the Move Engines for source error condition, as well as for destination error condition. The Move Engine's interrupts can be directed to one of the interrupt outputs SR[15:0][1)] by setting the Move Engine interrupt pointer EER.ME0INP with a corresponding value.

Note that in case of a read move error, the write move is not executed but the destination address is updated.

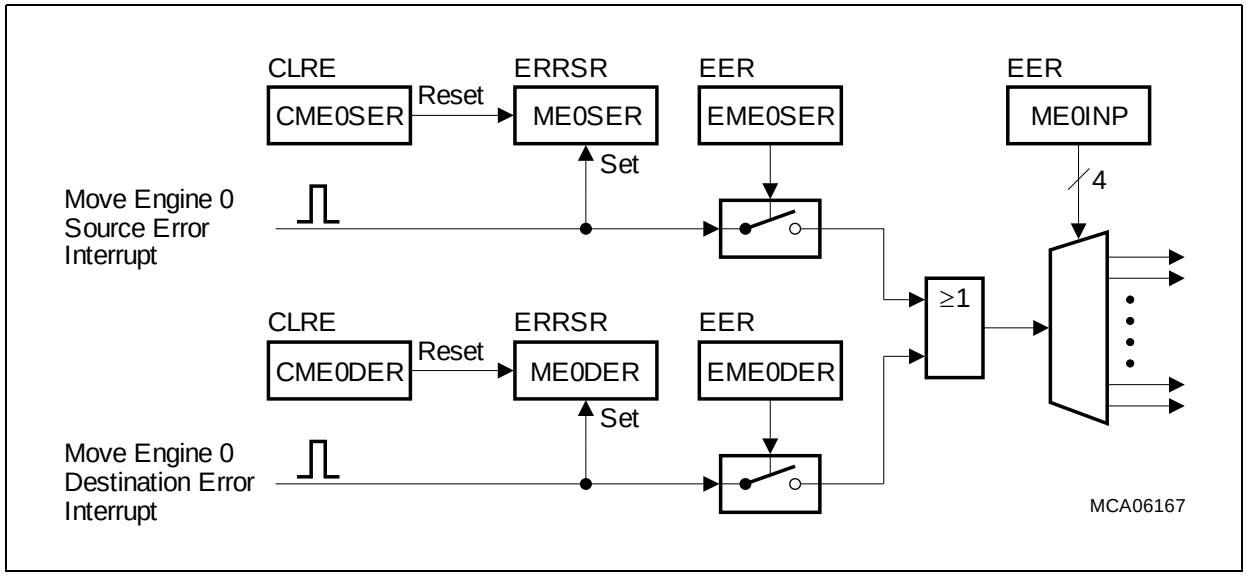


Figure 11-19 Move Engine Interrupts

1) In the TC1797 SR[7:0] are connected to interrupt nodes. SR[15:8] are used as DMA channel trigger signals.

When a Move Engine 0 source or destination error occurs, additional status bits and bit fields are provided in the error status register ERRSR to indicate the following two status conditions:

- At which On Chip Bus interface a Move Engine 0 error occurred (FPI or LMB)
- For which DMA channel a Move Engine 0 read or write move error was reported (LECME0)

These error status bits and bit fields are required by error handler software to detect in detail at which On Chip Bus interface and DMA channel the Move Engine error has been generated. ERRSR.FPIER or ERRSR.LMBER is reset when bits CLRE.CFPI0ER or CLRE.CFPI1ER is respectively set.

11.2.12.4 Wrap Buffer Interrupts

Each DMA channel mn is able to generate a wrap buffer interrupt for source buffer or destination buffer overflow. Further details on the pattern detection are described in **Section 11.2.13**.

A wrap source buffer interrupt of DMA channel mn is indicated by status flag WRPSR.WRPSmn. A wrap destination buffer interrupt of DMA channel mn is indicated by the status flag WRPSR.WRPDmn. Both interrupt status flags can be reset by software when bit INTCR.CWRPmn (or CHRSTR.CHmn becomes set). The wrap source buffer interrupt is enabled when bit CHICRmn.WRPSE is set. The wrap destination buffer interrupt is enabled when bit CHICRmn.WRPDE is set. The two interrupts for wrap source buffer and wrap destination buffer are OR-ed together to one common wrap buffer interrupt of DMA channel mn that can be directed to one of the interrupt outputs SR[15:0][1)] by setting the wrap buffer interrupt pointer CHICRmn.WRPP with a corresponding value. Note that the pattern match should not be enabled while a wrap interrupt is enabled for the same channel.

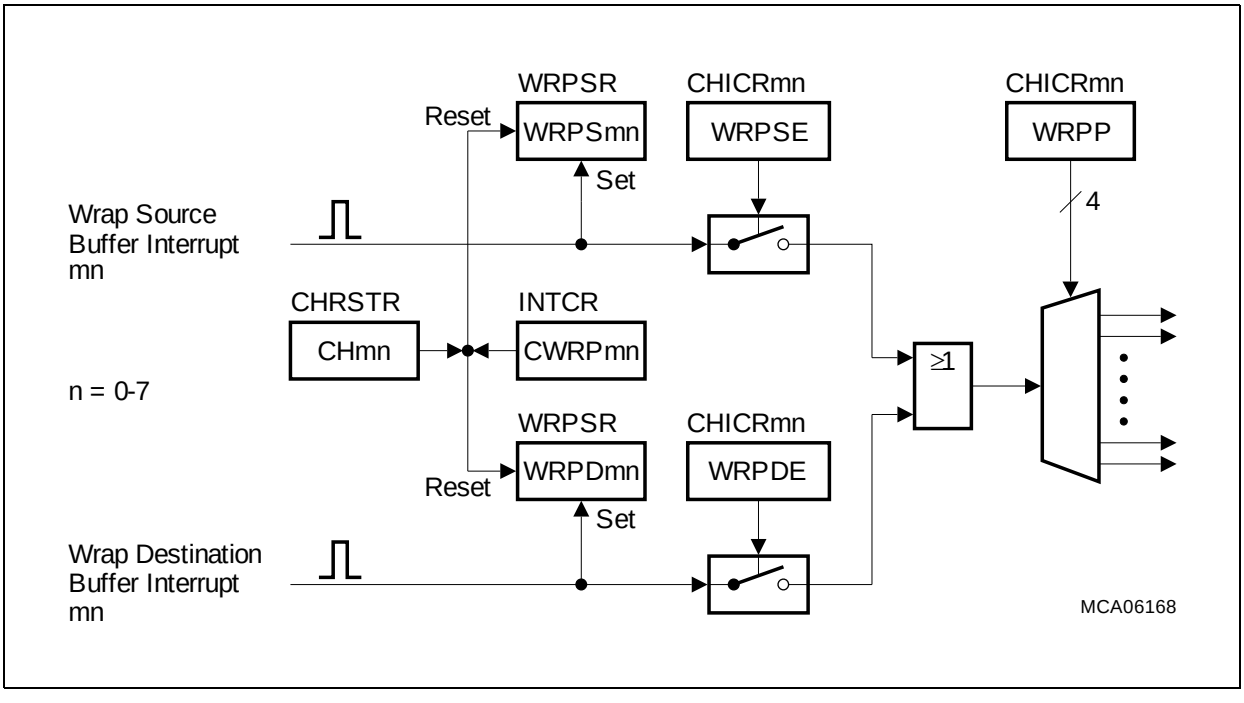


Figure 11-20 DMA Wrap Buffer Interrupts(m = 0-1)

1) In the TC1797 SR[7:0] are connected to interrupt nodes. SR[15:8] are used as DMA channel trigger signals.

11.2.12.5 Interrupt Request Compressor

The interrupt control logic of the DMA controller uses an interrupt compressing scheme that allows high flexibility in interrupt processing. The request compressor logic as shown in **Figure 11-21** condenses the 8 + 1 + 1 + 8 = 18 interrupt sources to the sixteen interrupt outputs. Each internal interrupt source can be directed to one of the sixteen interrupt outputs SR[15:0][1)] by using a 4-bit Interrupt Node Pointer. This also allows the connection of more than one interrupt source to one interrupt output SRx. Each interrupt output SR[15:0][1)] can also be activated by writing a 1 to the corresponding bit GINTR.SIDMAx.

1) In the TC1797 SR[7:0] are connected to interrupt nodes. SR[15:8] are used as DMA channel trigger signals.

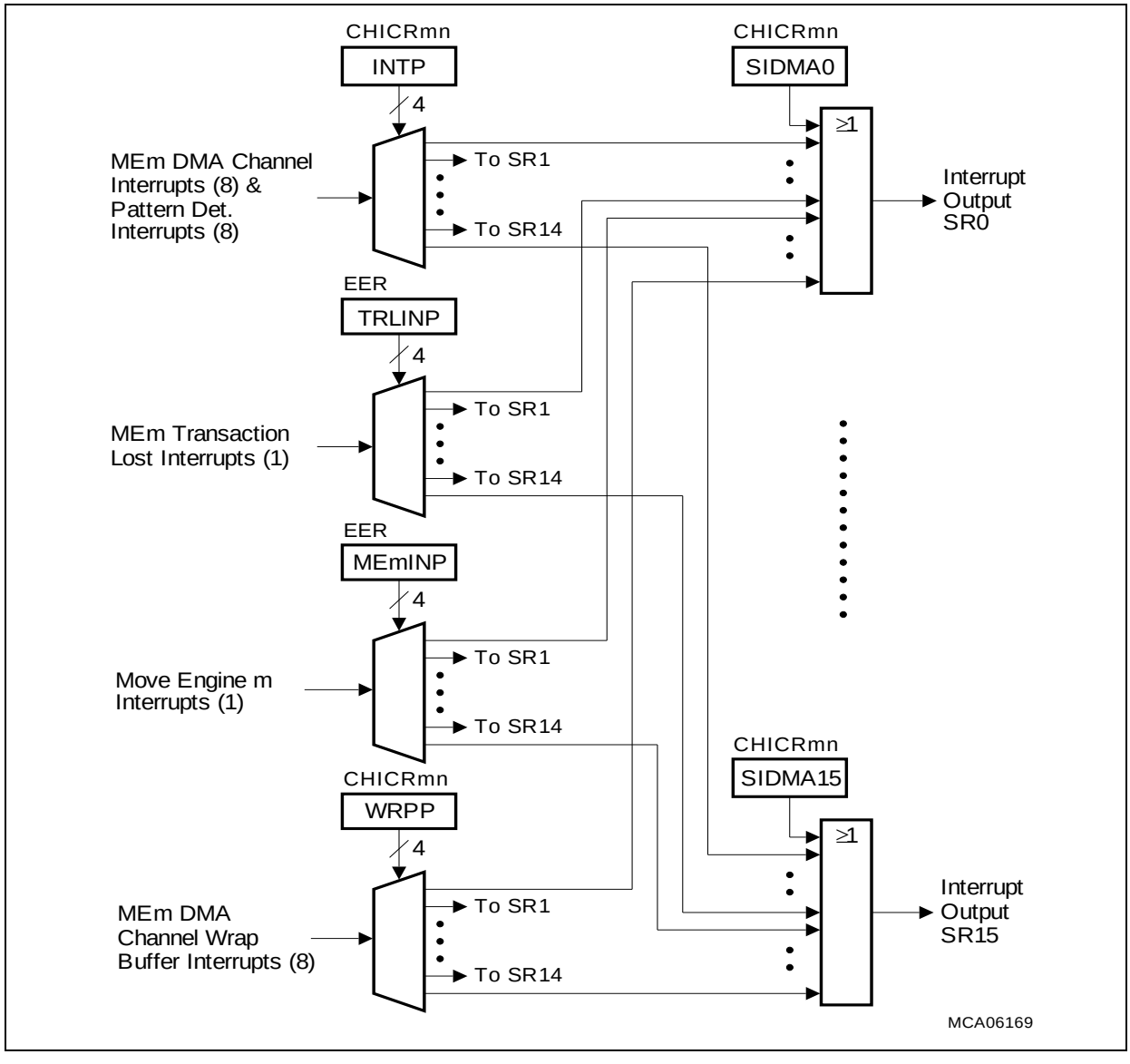


Figure 11-21 DMA Interrupt Request Compressor (m = 0-1)

11.2.13 Pattern Detection

The Move Engine in the DMA Sub-Block provides a register ME0R that contains the data that was read during the last read move. Parts of this read move data can be compared after the read move to data that is stored in the Move Engine pattern register ME0PR of DMA Sub-Block 0. The result of this pattern compare match is always stored in a bit (LXO) of the channel status register of the DMA channel mn that is currently executing the DMA move. Therefore, the pattern match result LXO of the previous read move can also be combined together with the pattern match result of the actual read move. ME0R is overwritten with each read move.

As the compare match patterns are stored in the Move Engine 0 (register ME0PR), its compare patterns are used for all DMA channels that are assigned to Move Engine 0 (all DMA channels of the DMA Sub-Block 0.

The configuration and capabilities of the pattern detection logic further depends on the settings of CHCRmn.CHDW. CHDW determines the data width for the read and write moves individually for each DMA channel mn. Another control bit, CHCRmn.PATSEL, selects among the different operating modes for a specific value of CHDW.

Depending on CHCRmn.PATSEL and on the positive result of the comparison, two actions follow (if CHCRmn.PATSEL=00, no action will be taken when a pattern match is detected, so the wrap interrupt can be used):

- The activation of the interrupt corresponding to the current active channel mn using the Interrupt Pointer defined in CHICRmn.INTP.
- Reset TRSR.HTREmn and TRSR.CHmn in order to stop the current transaction (Hardware and Software request enable). The value of CHSRmn.TCOUNT can be read out by the interrupt software.

The software will have to service the interrupt and to activate again the channel.

11.2.13.1 Pattern Compare Logic

Read move data and compare match patterns are compared on a bit-wise level. The logic as shown in **Figure 11-22** is implemented in each COMP block of **Figure 11-23**, **Figure 11-24**, and **Figure 11-25**. One COMP block controls either 8 bits or 16 bits of data and makes it possible to mask each data bit for the compare operation.

In the compare logic for one bit of the COMP block, a data bit from register ME0R is compared to the corresponding pattern bit stored in register ME0PR. If both bits are equal and a pattern mask bit stored in another part of register ME0PR is 0, the compare matched condition becomes active. When the pattern mask bit is set to 1, the compare matched condition is always active (set) for the related bit. When the compare matched conditions for each bit within a COMP block are true, the compare match output line of the COMP block becomes active.

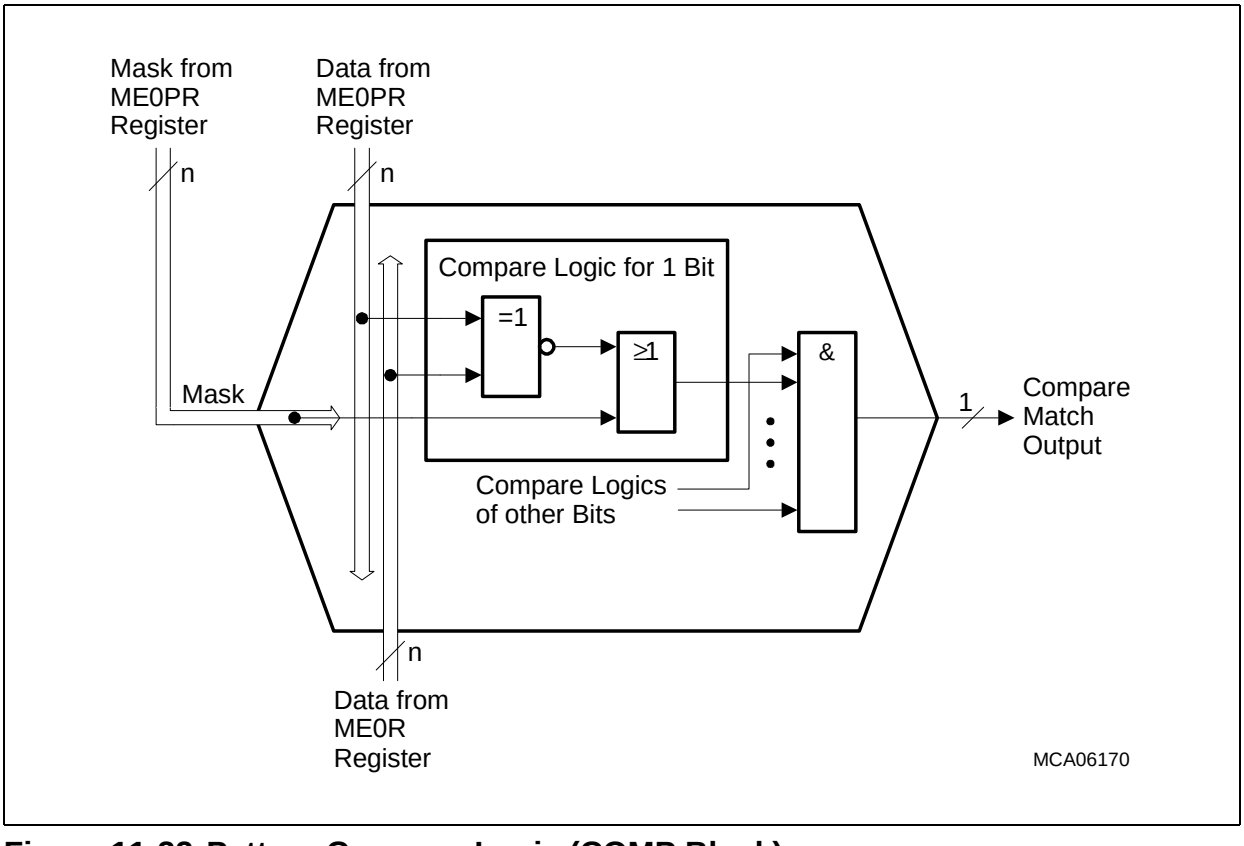


Figure 11-22 Pattern Compare Logic (COMP Block)

11.2.13.2 Pattern Detection for 8-bit Data Width

When 8-bit channel data width is selected (CHCRmn.CHDW = 00_B), the pattern detection logic is configured as shown in **Figure 11-23**. Three compare match configurations are possible.

Table 11-4 Pattern Detection for 8-bit Data Width

CHCRmn**. PATSEL**	**Pattern Detection Operating Modes**
00_B	Pattern detection disabled
01_B	Pattern compare of RD00 to PAT00, masked by PAT02
10_B	Pattern compare of RD00 to PAT01, masked by PAT03
11_B	Pattern compare of RD00 to PAT00, masked by PAT02 of the actual read move **and** Pattern compare of RD00 to PAT01, masked by PAT03 of the previous read move of DMA channel mn

When 8-bit channel data width is selected, the pattern detection logic allows the byte of one read move to be compared with two different patterns. Further, after each read move the pattern match result “RD00 with PAT01, masked by PAT03” is stored in bit CHCRmn.LXO. This operating mode allows, for example, two-byte sequences to be detected in an 8-bit data stream coming from a serial peripheral unit with 8-bit data width (e.g.: recognition of carriage-return, line-feed characters). A mask operation of each compared bit is possible.

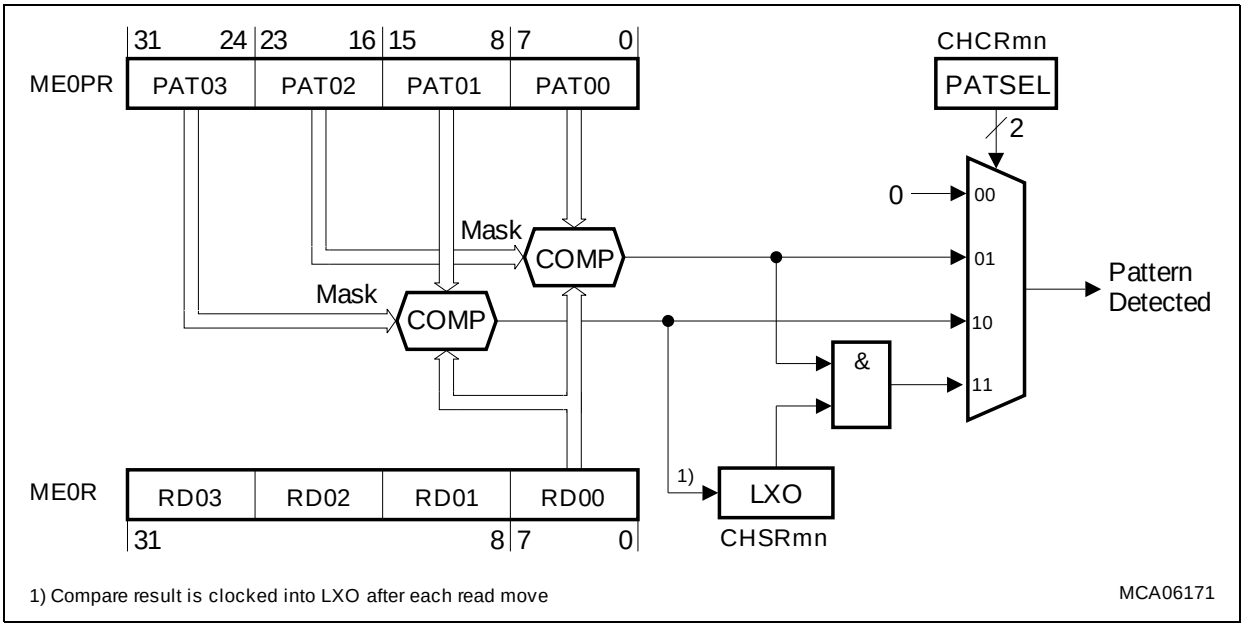


Figure 11-23 Pattern Detection for 8-bit Data Width (CHCRmn**.CHDW = 00_B)** (m = 0-

1)

11.2.13.3 Pattern Detection for 16-bit Data Width

When 16-bit channel data width is selected (CHCRmn.CHDW = 01_B) the pattern detection logic can be configured as shown in **Figure 11-24**. Three compare match configurations are possible.

Table 11-5 Pattern Detection for 16-bit Data Width

CHCRmn**. PATSEL**	**ADRCR**mn**. INCS**	**Pattern Detection Operating Modes**
00_B	–	Pattern detection disabled
01_B	–	**Aligned Mode:** Pattern compare of RD0[1:0] to PAT0[1:0], masked by PAT0[3:2]
10_B	0	**Unaligned Mode 1 (Source Address Decrement):** Pattern compare of RD01 to PAT00, masked by PAT02 of the actual read move **and** Pattern compare of RD00 to PAT01, masked by PAT03 (LXO) of the previous read move of DMA channel mn
	1	**Unaligned Mode 2 (Source Address Increment):** Pattern compare of RD00 to PAT01, masked by PAT03 of the actual read move **and** Pattern compare of RD01 to PAT00, masked by PAT02 (LXO) of the previous read move of DMA channel mn
11_B	0 or 1	**Combined Mode:** Pattern compare for aligned mode (PATSEL = 01_B) **or** unaligned modes (PATSEL = 10_B)

When 16-bit channel data width is selected, the pattern detection logic makes it possible to compare the complete half-word of one read move only (aligned mode) or to compare upper and lower byte of two consecutive read moves (unaligned modes). Both modes can be combined (combined mode) too. A mask operation of each compared bit is possible.

In unaligned mode 1 (source address decremented), the high byte (RD01) of the current and the low byte (RD00) of the previous 16-bit read move are compared.

In unaligned mode 2 (source address incremented), the low byte (RD00) of the current and the high byte (RD01) of the previous 16-bit read move are compared.

If it is not known on which byte boundary (even or odd address) the 16-bit pattern to be detected is located, the combined mode should be used. This mode is the most flexible

mode that combines the pattern search capability for aligned and unaligned 16-bit data searches.

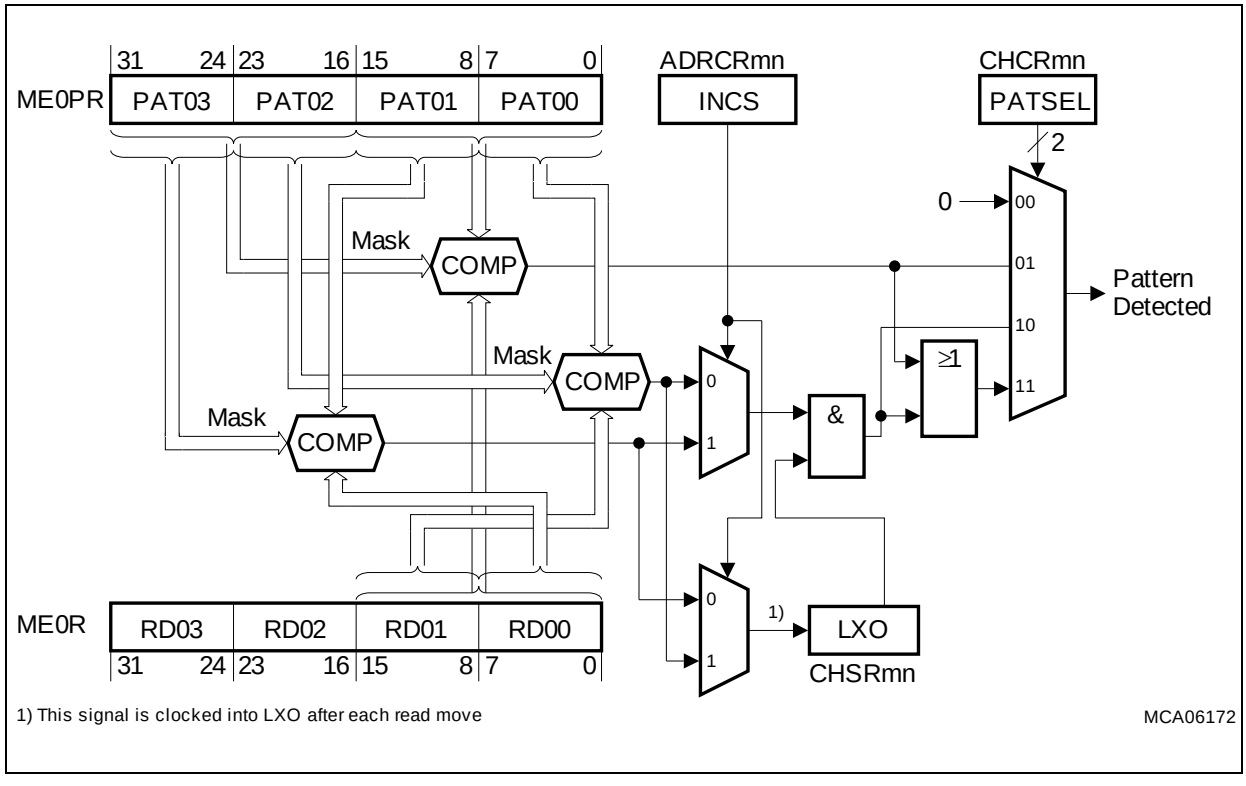


Figure 11-24 Pattern Detection for 16-bit Data Width (CHCRmn.CHDW = 01_B)
(m = 0-1)

11.2.13.4 Pattern Detection for 32-bit Data Width

When 32-bit channel data width is selected (CHCRmn.CHDW = 10_B) the pattern detection logic is configured as shown in **Figure 11-25**. Three compare match configurations are possible.

Table 11-6 Pattern Detection for 32-bit Data Width

CHCRmn**. PATSEL**	**Pattern Detection Operating Modes**
00_B	Pattern detection disabled
01_B	Unmasked pattern compare of RD0[1:0] to PAT0[1:0]
10_B	Unmasked pattern compare of RD0[3:2] to PAT0[3:2]
11_B	Unmasked pattern compare of RD0[3:0] to PAT0[3:0]

In 32-bit channel data width mode, the pattern detection logic makes it possible to compare the lower half-word only, the upper half-word only, or the complete 32-bit word with a pattern stored in the ME0PR register. A mask operation is not possible.

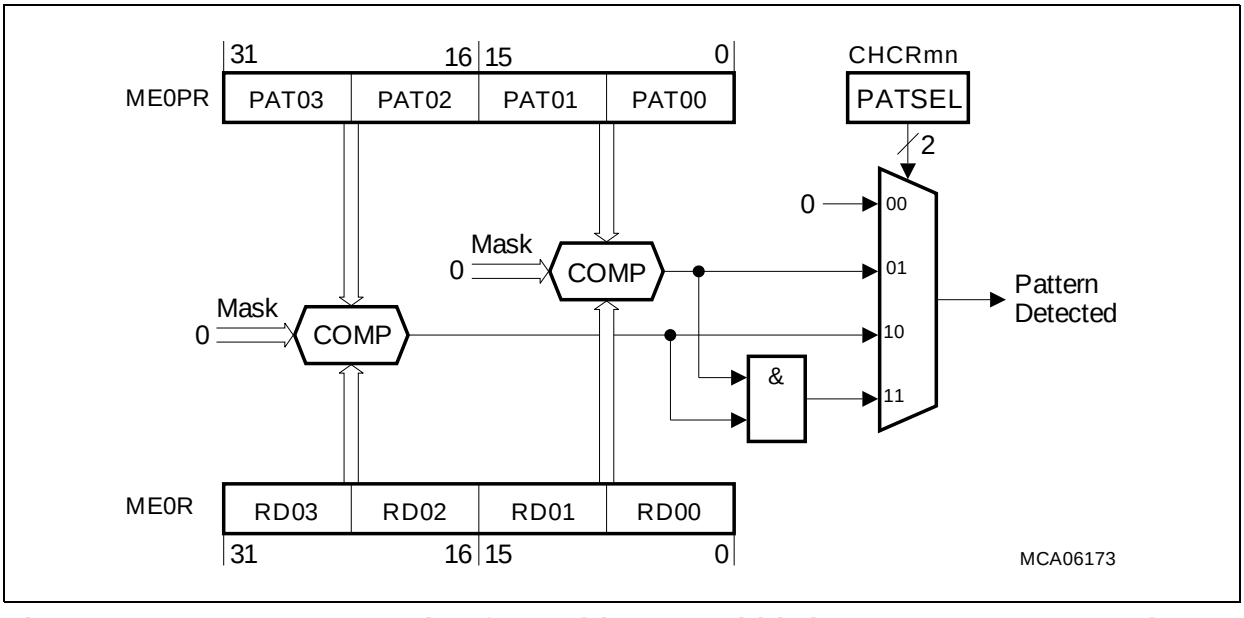


Figure 11-25 Pattern Detection for 32-bit Data Width (CHCRmn**.CHDW = 10_B)**
(m = 0-1)

11.2.14 Access Protection

The DMA controller provides an access protection logic that makes it possible to disable read and write accesses of the Move Engines to specific parts of the memory map. Each address of a read move and a write move is always checked to determine if it is within an address range that is enabled for read/write access. If no address range is valid for an actual move address, a Move Engine interrupt can be generated.

The access protection logic handles two levels of address range definitions:

- Fixed address range
- Programmable address range extension

There are 32 fixed address ranges available that can be individually enabled/disabled in the Move Engine by the address range enable bits AENx (x = 0-31): These bits are located in the Move Engine 0/1 Access Enable Register ME0AENR/ME1AENR. If bit AENx is set, read/write accesses to the associated address range x are allowed. If bit AENx is cleared (default after reset), read/write accesses to the associated address range x are not executed and a Move Engine interrupt for source or destination move is generated (see also **Section 11.2.12.3**).

Additionall four programmable address range extensions are available for each of the two Move Engines that are fixed assigned to the OVRAM, the PCP PRAM, the PMI SPRAM and the DMI LDRAM (see also **Section 11.4.2**). Each programmable address range extension makes it possible to define a sub-range within the corresponding address range where an access will be executed by the corresponding Move Engine if the address range is not disabled by the corresponding AENx bit. An access to the address range outside the defined sub-range will not be executed by the corresponding Move Engine. The parameters for the sub-ranges are stored in the Move Engine 0/1 Access Range Register ME0ARR/ME1ARR. The programmable address range extension is a feature that is applicable for memory access protection of memory blocks. In such an application, several memory sections are defined as sub-ranges of a complete memory block.

Figure 11-26 shows the two levels of address range definitions with the resulting address sub-ranges of the programmable address range extension. In a fixed address range, the width of fixed and variable address bits is constant. Number “a” determines the lowest bit position of the fixed address, and is fixed individually and product-specific for each of the 32 fixed address ranges. With the programmable address range extension, the variable address part of the fixed address range definition (as defined by AENx) is reduced by the definition of a programmable number (up to 32) of sub-ranges. Bit field ME0ARR.SIZE/ME1ARR.SIZE determines the sub-range size and bit field ME0ARR.SLICE/ME1ARR.SLICE determines which of the sub-ranges is currently selected for access protection control. The two parameters (SIZE, SLICE) of the four address range extensions of a move engine are numbered by index “n” (n = 0-3).

In the TC1797 the number “a” is defined in the followign way:

- SIZE0/SLIZE0 covering the SPRAM , “a” = 17
- SIZE1/SLIZE1 covering the OVRAM , “a” = 16
- SIZE2/SLIZE2 covering the LDRAM , “a” = 17
- SIZE3/SLIZE3 covering the PRAM, “a” = 16

Two sub-range examples (see **Figure 11-26**):

- 2^3 = 8 sub-ranges are available with SIZE = 100_B. SLICE[2:0] selects one out of the eight sub-ranges. SLICE[4:3] is “don’t care”.
- 2^7 = 128 sub-ranges are basically available with SIZEn = 000_B. SLICEn[4:0] selects one out of the lowest 32 sub-ranges. The upper 3 x 32 = 96 sub-ranges are not selectable (fixed address bits a-1 and a-2).

Note: The definition of the fixed address ranges x and the assignment of each sub-range to one of the fixed address ranges is product-specific. The definitions of the address ranges for the DMA controller as implemented in the TC1797 are defined on ***Page 11-112****.*

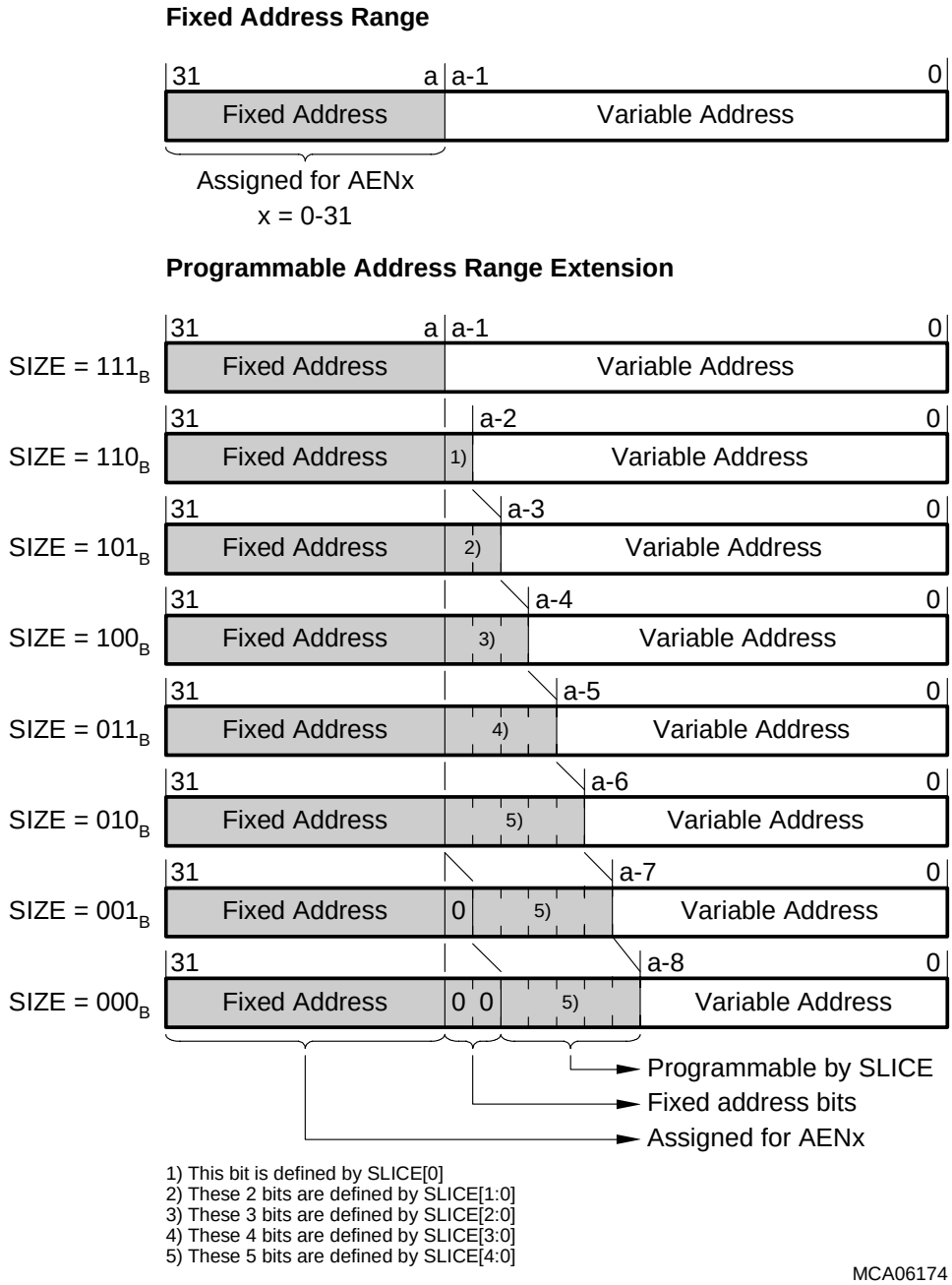


Figure 11-26 Access Protection Address Range Definitions

11.3 DMA Module Registers

Figure 11-27 and **Table 11-8** show all registers associated with the DMA Controller Kernel. Additionally, **Table 11-8** includes the DMA module specific registers that are also showed in **Table 11-29**. All DMA kernel register names described in this section are also referenced in other parts of the TC1797 User's Manual by the module name prefix "DMA_".

The registers are numbered by one index to indicate the related DMA Sub-Block and one index to indicate the related DMA channel: Index "m" refers to the DMA Sub-Block number (m = 0-1) and Index "n" or "x" refers to the channel number (n = 0-7 or x = 0-7) within the DMA Sub-Block.

DMA Registers Overview

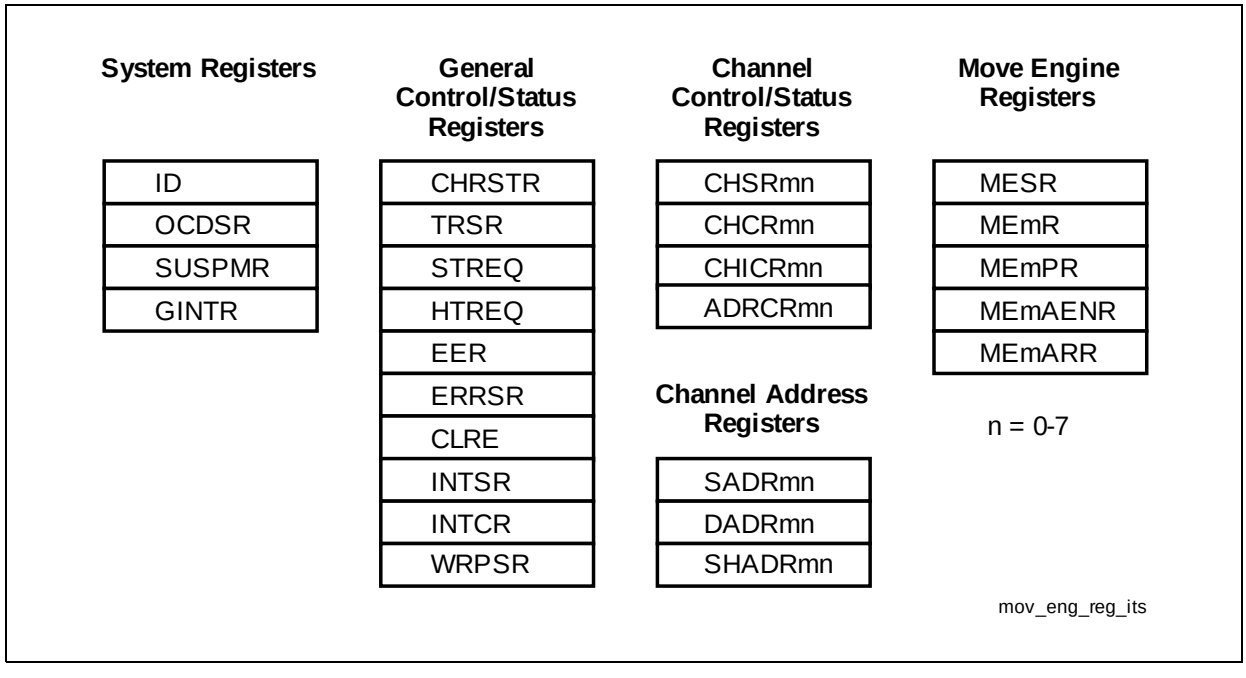


Figure 11-27 DMA Kernel Registers

Table 11-7 Registers Address Space - DMA Module

Module	Base Address	End Address	Note
DMA	F000 3C00$_H$	F000 3EFF$_H$	

Table 11-8 Registers Overview - DMA Control Registers

Short Name	Description	Offset Addr.[1)]	Access Mode		Reset Class	Description See
			Read	Write		
DMA_CLC	DMA Clock Control Register	000_H	U, SV	SV, E	3	Page 11-122
-	Reserved	004_H	nBE	SV	-	-
DMA_ID	DMA Module Identification Register	008_H	U, SV	BE	-	Page 11-54
-	Reserved	$00C_H$	BE	BE	-	-
DMA_CHRSTR	DMA Channel Reset Request Register	010_H	U, SV	SV	3	Page 11-60
DMA_TRSR	DMA Transaction Request State Register	014_H	U, SV	BE	3	Page 11-61
DMA_STREQ	DMA Software Transaction Request Register	018_H	U, SV	SV	3	Page 11-63
DMA_HTREQ	DMA Hardware Transaction Request Register	$01C_H$	U, SV	SV	3	Page 11-64
DMA_EER	DMA Enable Error Register	020_H	U, SV	SV	3	Page 11-66
DMA_ERRSR	DMA Error Status Register	024_H	U, SV	BE	3	Page 11-69
DMA_CLRE	DMA Clear Error Register	028_H	U, SV	SV	3	Page 11-72
DMA_GINTR	DMA Global Interrupt Set Register	$02C_H$	U, SV	SV	3	Page 11-59
DMA_MESR	DMA Move Engine Status Register	030_H	U, SV	BE	3	Page 11-79
DMA_ME0R	DMA Move Engine 0 Read Register	034_H	U, SV	BE	3	Page 11-81
DMA_ME1R	DMA Move Engine 1 Read Register	038_H	U, SV	BE	3	Page 11-81
DMA_ME0PR	DMA Move Engine 0 Pattern Register	$03C_H$	U, SV	SV	3	Page 11-81

Table 11-8 Registers Overview - DMA Control Registers

Short Name	Description	Offset Addr.[1)]	Access Mode		Reset Class	Description See
			Read	Write		
DMA_ME1PR	DMA Move Engine 1 Pattern Register	040_H	U, SV	SV	3	Page 11-81
DMA_ME0AENR	DMA Move Engine 0 Access Enable Register	044_H	U, SV	SV, E	3	Page 11-82
DMA_ME0ARR	DMA Move Engine 0 Access Range Register	048_H	U, SV	SV, E	3	Page 11-84
DMA_ME1AENR	DMA Move Engine 1 Access Enable Register	$04C_H$	U, SV	SV, E	3	Page 11-82
DMA_ME1ARR	DMA Move Engine 1 Access Range Register	050_H	U, SV	SV, E	3	Page 11-84
DMA_INTSR	DMA Interrupt Status Register	054_H	U, SV	BE	3	Page 11-74
DMA_INTCR	DMA Interrupt Clear Register	058_H	U, SV	SV	3	Page 11-78
DMA_WRPSR	DMA Wrap Status Register	$05C_H$	U, SV	BE	3	Page 11-76
-	Reserved	060_H	BE	BE	-	-
DMA_OCDSR	DMA OCDS Register	064_H	U, SV	SV, E	1	Page 11-55
DMA_SUSPMR	DMA Suspend Mode Register	068_H	U, SV	SV, E	1	Page 11-57
-	Reserved	$06C_H$-$07C_H$	BE	BE	-	-
DMA_CHSRmn	DMA Channel mn Status Register (n = 0-7, m = 0-1)	(n x 20_H) + (m x 100_H) + 080_H	U, SV	BE	3	Page 11-90

Table 11-8 Registers Overview - DMA Control Registers

Short Name	Description	Offset Addr.[1]	Access Mode		Reset Class	Description See
			Read	Write		
DMA_CHCRmn	DMA Channel mn Control Register (n = 0-7, m = 0-1)	(n x 20_H) + (m x 100_H) + 084_H	U, SV	SV	3	Page 11-86
DMA_CHICRmn	DMA Channel mn Interrupt Control Register (n = 0-7, m = 0-1)	(n x 20_H) + (m x 100_H) + 088_H	U, SV	SV	3	Page 11-91
DMA_ADRCRmn	DMA Channel mn Address Control Register (n = 0-7, m = 0-1)	(n x 20_H) + (m x 100_H) + $08C_H$	U, SV	SV	3	Page 11-93
DMA_SADRmn	DMA Channel mn Source Address Register (n = 0-7, m = 0-1)	(n x 20_H) + (m x 100_H) + 090_H	U, SV	SV	3	Page 11-98
DMA_DADRmn	DMA Channel mn Destination Address Register (n = 0-7, m = 0-1)	(n x 20_H) + (m x 100_H) + 094_H	U, SV	SV	3	Page 11-99
DMA_SHADRmn	DMA Channel mn Shadow Address Register (n = 0-7, m = 0-1)	(n x 20_H) + (m x 100_H) + 098_H	U, SV	BE / SV [2]	3	Page 11-100
-	Reserved (n = 0-7, m = 0-1)	(n x 20_H) + (m x 100_H) + $09C_H$	BE	BE	-	-

Table 11-8 Registers Overview - DMA Control Registers

Short Name	Description	Offset Addr.[1]	Access Mode		Reset Class	Description See
			Read	Write		
-	Reserved	280_H-$29C_H$	BE	BE	-	-
DMA_MLI0SRC3	DMA MLI0 Service Request Control Reg. 3	$2A0_H$	U, SV	SV	3	Page 11-124
DMA_MLI0SRC2	DMA MLI0 Service Request Control Reg. 2	$2A4_H$	U, SV	SV	3	Page 11-124
DMA_MLI0SRC1	DMA MLI0 Service Request Control Reg. 1	$2A8_H$	U, SV	SV	3	Page 11-124
DMA_MLI0SRC0	DMA MLI0 Service Request Control Reg. 0	$2AC_H$	U, SV	SV	3	Page 11-124
–	Reserved	$2B0_H$ - $2DC_H$	BE	BE	-	-
DMA_SRC7	DMA Service Request Control Register 7	$2E0_H$	U, SV	SV	3	Page 11-123
DMA_SRC6	DMA Service Request Control Register 6	$2E4_H$	U, SV	SV	3	Page 11-123
DMA_SRC5	DMA Service Request Control Register 5	$2E8_H$	U, SV	SV	3	Page 11-123
DMA_SRC4	DMA Service Request Control Register 4	$2EC_H$	U, SV	SV	3	Page 11-123
DMA_SRC3	DMA Service Request Control Register 3	$2F0_H$	U, SV	SV	3	Page 11-123
DMA_SRC2	DMA Service Request Control Register 2	$2F4_H$	U, SV	SV	3	Page 11-123
DMA_SRC1	DMA Service Request Control Register 1	$2F8_H$	U, SV	SV	3	Page 11-123
DMA_SRC0	DMA Service Request Control Register 0	$2FC_H$	U, SV	SV	3	Page 11-123

1) The absolute register address is calculated as follows:
Module Base Address (Table 11-7) + Offset Address (shown in this column)
Further, the following ranges for parameters i, k, x, and n are valid: i = 0-7, k = 0-7, x = 0-1, n = 0-63.

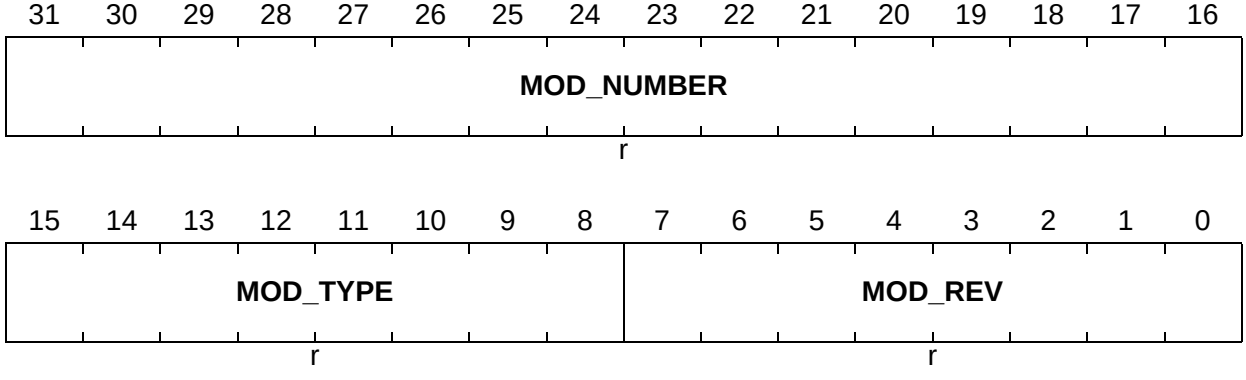
2) Write access mode to DMA_SHADRmn is controlled by the register bit DMA_ADRCRmn.SHWEN. DMA_ADRCRmn.SHWEN=´0´ -> Access Mode Write for DMA_SHADRmn is BE. DMA_ADRCRmn.SHWEN=´1´ -> Access Mode Write for DMA_SHADRmn is SV.

Note: Register bits marked “w” in the following register description are virtual registers and do not contain flip-flops. They are always read as 0.

11.3.1 System Registers

DMA Module Identification Register.

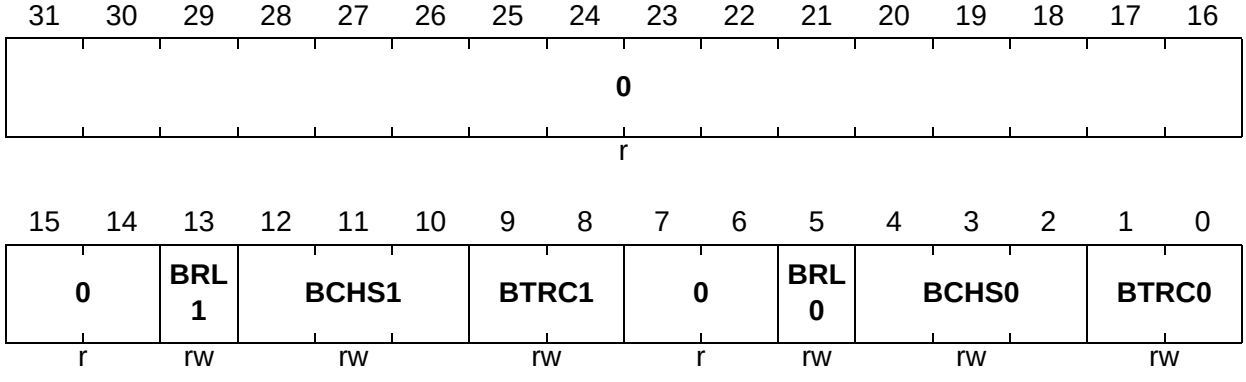
DMA_ID
Module Identification Register (008$_H$) Reset Value: 001A C0XX$_H$



Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision Number** This bit field defines the module revision number. The value of a module revision starts with 01$_H$ (first revision).
MOD_TYPE	[15:8]	r	**Module Type** The bit field is set to C0$_H$ which defines the module as a 32-bit module.
MOD_NUMBER	[31:16]	r	**Module Number Value** This bit field defines a module identification number. The value for the Memory Checker module is 001A$_H$.

The OCDS Register describes the break capability of the DMA module. OCDSR is only reset with the OCDS Reset.

DMA_OCDSR
DMA OCDS Register **(064_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
BTRC0	[1:0]	rw	**Break Trigger Condition In Sub-Block 0** This bit field determines the transition type for the transaction request bit TRSR.CH0n that leads to a break condition in DMA Sub-Block 0. 00_B No break condition is generated 01_B A break condition is generated when TRSR.CH0n changes from 0 to 1 10_B A break condition is generated when TRSR.CH0n changes from 1 to 0 11_B A break condition is generated when TRSR.CH0n changes its state
BCHS0	[4:2]	rw	**Break Channel Select In Sub-Block 0** This bit field determines the DMA channel n of DMA Sub-Block 0 whose transaction request bit TRSR.CH0n is observed for signal transitions as defined by BTRC0. 000_B DMA channel 00 selected 001_B DMA channel 01 selected $\ldots_B$ ... 110_B DMA channel 06 selected 111_B DMA channel 07 selected

Field	Bits	Type	Description
BRL0	5	rw	**Break On Request Lost in Sub-Block 0** This bit field determines whether a BREAK signal is generated for DMA Sub-Block 0 when at least one of its eight transaction lost interrupts becomes active. 0_B No break condition is generated 1_B A break condition is generated for DMA Sub-Block 0 when at least one of its eight transaction lost interrupts becomes active
BTRC1	[9:8]	rw	**Break Trigger Condition In Sub-Block 1** This bit field determines the transition type for the transaction request bit TRSR.CH1n that leads to a break condition in DMA Sub-Block 1. 00_B No break condition is generated 01_B A break condition is generated when TRSR.CH1n changes from 0 to 1 10_B A break condition is generated when TRSR.CH1n changes from 1 to 0 11_B A break condition is generated when TRSR.CH1n changes its state
BCHS1	[12:10]	rw	**Break Channel Select In Sub-Block 1** This bit field determines the DMA channel n of DMA Sub-Block 1 whose transaction request bit TRSR.CH1n is observed for signal transitions as defined by BTRC1. 000_B DMA channel 10 selected 001_B DMA channel 11 selected $...{}_B$ ... 110_B DMA channel 16 selected 111_B DMA channel 17 selected
BRL1	13	rw	**Break On Request Lost in Sub-Block 1** This bit field determines whether a BREAK signal is generated for DMA Sub-Block 1 when at least one of its eight transaction lost interrupts becomes active. 0_B No break condition is generated 1_B A break condition is generated for DMA Sub-Block 1when at least one of its eight transaction lost interrupts becomes active
0	[7:6], [31:14]	r	**Reserved** Read as 0; should be written with 0.

The Suspend Mode Register contains bits for each DMA channel that make it possible to enable/disable its Soft-suspend Mode capability and that indicate its suspend status.

DMA_SUSPMR
DMA Suspend Mode Register (068_H) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SUSAC17	**SUSAC16**	**SUSAC15**	**SUSAC14**	**SUSAC13**	**SUSAC12**	**SUSAC11**	**SUSAC10**	**SUSAC07**	**SUSAC06**	**SUSAC05**	**SUSAC04**	**SUSAC03**	**SUSAC02**	**SUSAC01**	**SUSAC00**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SUSEN17	**SUSEN16**	**SUSEN15**	**SUSEN14**	**SUSEN13**	**SUSEN12**	**SUSEN11**	**SUSEN10**	**SUSEN07**	**SUSEN06**	**SUSEN05**	**SUSEN04**	**SUSEN03**	**SUSEN02**	**SUSEN01**	**SUSEN00**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
SUSEN0n (n = 0-7)	n	rw	**Suspend Enable for DMA Channel 1n** This bit enables the soft suspend capability individually for each DMA channel 0n. 0_B DMA channel 0n is disabled for Soft-suspend Mode. The DMA channel 0n does not react on an active suspend request signal SUSREQ. 1_B DMA channel 0n is enabled for Soft-suspend Mode. If the suspend request signal SUSREQ becomes active, a DMA transaction of DMA channel 0n is stopped after the current DMA transfer has been finished Soft-suspend Mode can be terminated when SUSEN0n is written with 0.

Field	Bits	Type	Description
SUSEN1n (n = 0-7)	8+n	rw	**Suspend Enable for DMA Channel 1n** This bit enables the soft suspend capability individually for each DMA channel 1n. 0_B DMA channel 1n is disabled for Soft-suspend Mode. The DMA channel 1n does not react on an active suspend request signal SUSREQ 1_B DMA channel 1n is enabled for Soft-suspend Mode. If the suspend request signal SUSREQ becomes active, a DMA transaction of DMA channel 1n is stopped after the current DMA transfer has been finished Soft-suspend Mode can be terminated when SUSENmn is written with 0.
SUSAC0n (n = 0-7)	16+n	rh	**Suspend Active for DMA Channel 0n** This status bit indicates whether DMA channel 0n is in Soft-suspend Mode or not. 0_B DMA channel 0n is not in Soft-suspend Mode or internal actions are not yet finished after the Soft-suspend Mode was requested 1_B DMA channel 0n is in Soft-suspend Mode
SUSAC1n (n = 0-7)	24+n	rh	**Suspend Active for DMA Channel 1n** This status bit indicates whether DMA channel 1n is in Soft-suspend Mode or not. 0_B DMA channel 1n is not in Soft-suspend Mode or internal actions are not yet finished after the Soft-suspend Mode was requested 1_B DMA channel 1n is in Soft-suspend Mode

Note: This register is only reset by the Debug reset.

The Global Interrupt Set Register allows the interrupt output lines of the DMA to be activated by software.

DMA_GINTR
DMA Global Interrupt Set Register (02C$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SIDMA15	SIDMA14	SIDMA13	SIDMA12	SIDMA11	SIDMA10	SIDMA9	SIDMA8	SIDMA7	SIDMA6	SIDMA5	SIDMA4	SIDMA3	SIDMA2	SIDMA1	SIDMA0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
SIDMAx (x = 0-15)	x	w	**Set DMA Interrupt Output Line x** 0_B No action 1_B DMA interrupt output line SRx will be activated. Reading this bit returns a 0
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

*Note: In the TC1797, SR[7:0] are connected to interrupt nodes. SR[15:8] are used as DMA channel request inputs (**Page 11-102**).*

11.3.2 General Control/Status Registers

The bits in the Channel Reset Request Register are used to reset DMA channel mn.

DMA_CHRSTR
DMA Channel Reset Request Register

(010_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH 17	**CH 16**	**CH 15**	**CH 14**	**CH 13**	**CH 12**	**CH 11**	**CH 10**	**CH 07**	**CH 06**	**CH 05**	**CH 04**	**CH 03**	**CH 02**	**CH 01**	**CH 00**
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
CH0n (n = 0-7)	n	rwh	**Channel 0n Reset** These bits force the DMA channel 0n to stop its current DMA transaction. Once set by software, this bit will be automatically cleared when the channel has been reset. Writing a 0 to CH0n has no effect. 0_B No action (write) or the requested channel reset has been reset (read). 1_B DMA channel 0n is stopped. More details see **Page 11-17**.
CH1n (n = 0-7)	8+n	rwh	**Channel 1n Reset** These bits force the DMA channel 1n to stop its current DMA transaction. Once set by software, this bit will be automatically cleared when the channel has been reset. Writing a 0 to CH1n has no effect. 0_B No action (write) or the requested channel reset has been reset (read). 1_B DMA channel 1n is stopped. More details see **Page 11-17**.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

The bits in the Transaction Request State Register indicates which DMA channel is processing a request, and which DMA channel has hardware transaction requests enabled.

DMA_TRSR
DMA Transaction Request State Register
(014_H) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
HTRE17	**HTRE16**	**HTRE15**	**HTRE14**	**HTRE13**	**HTRE12**	**HTRE11**	**HTRE10**	**HTRE07**	**HTRE06**	**HTRE05**	**HTRE04**	**HTRE03**	**HTRE02**	**HTRE01**	**HTRE00**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH17	**CH16**	**CH15**	**CH14**	**CH13**	**CH12**	**CH11**	**CH10**	**CH07**	**CH06**	**CH05**	**CH04**	**CH03**	**CH02**	**CH01**	**CH00**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
CH0n (n = 0-7)	n	rh	**Transaction Request State of DMA Channel 0n** 0_B No DMA request is pending for channel 0n. 1_B A DMA request is pending for channel 0n.
CH1n (n = 0-7)	8+n	rh	**Transaction Request State of DMA Channel 1n** 0_B No DMA request is pending for channel 1n. 1_B A DMA request is pending for channel 1n.
HTRE0n (n = 0-7)	16+n	rh	**Hardware Transaction Request Enable State of DMA Channel 0n** 0_B Hardware transaction request for DMA Channel 0n is disabled. An input DMA request will not trigger the channel 0n. 1_B Hardware transaction request for DMA Channel 0n is enabled. The transfers of a DMA transaction are controlled by the corresponding channel request line of the DMA requesting source. HTRE0n is set to 0 when CHSR0n.TCOUNT is decremented and CHSR0n.TCOUNT = 0. HTRE0n can be enabled and disabled with HTREQ.ECH0n or HTREQ.DCH0n.

Field	Bits	Type	Description
HTRE1n (n = 0-7)	24+n	rh	**Hardware Transaction Request Enable State of DMA Channel 1n** 0_B Hardware transaction request for DMA Channel 1n is disabled. An input DMA request will not trigger the channel 1n. 1_B Hardware transaction request for DMA Channel 1n is enabled. The transfers of a DMA transaction are controlled by the corresponding channel request line of the DMA requesting source. HTRE1n is set to 0 when CHSR1n.TCOUNT is decremented and CHSR1n.TCOUNT = 0. HTRE1n can be enabled and disabled with HTREQ.ECH1n or HTREQ.DCH1n.

The bits in the Software Transaction Request Register are used to generate a DMA transaction request by software.

DMA_STREQ
DMA Software Transaction Request Register
(018_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SCH 17	SCH 16	SCH 15	SCH 14	SCH 13	SCH 12	SCH 11	SCH 10	SCH 07	SCH 06	SCH 05	SCH 04	SCH 03	SCH 02	SCH 01	SCH 00
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
SCH0n (n = 0-7)	n	w	**Set Transaction Request for DMA Channel 0n** 0_B No action. 1_B A transaction for DMA channel 0n is requested. When setting SCH0n, TRSR.CH0n becomes set to indicate that a DMA request is pending for DMA channel 0n.
SCH1n (n = 0-7)	8+n	w	**Set Transaction Request for DMA Channel 1n** 0_B No action. 1_B A transaction for DMA channel 1n is requested. When setting SCH1n, TRSR.CH1n becomes set to indicate that a DMA request is pending for DMA channel 1n.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

Note: Register bits marked with “w” are virtual and are not stored in flip-flops. Reading STREQ returns 0 when read.

The bits in the Hardware Transaction Request Register enable or disable DMA hardware requests.

DMA_HTREQ
DMA Hardware Transaction Request Register
($01C_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DCH 17	**DCH 16**	**DCH 15**	**DCH 14**	**DCH 13**	**DCH 12**	**DCH 11**	**DCH 10**	**DCH 07**	**DCH 06**	**DCH 05**	**DCH 04**	**DCH 03**	**DCH 02**	**DCH 01**	**DCH 00**
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ECH 17	**ECH 16**	**ECH 15**	**ECH 14**	**ECH 13**	**ECH 12**	**ECH 11**	**ECH 10**	**ECH 07**	**ECH 06**	**ECH 05**	**ECH 04**	**ECH 03**	**ECH 02**	**ECH 01**	**ECH 00**
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
ECH0n (n = 0-7)	n	w	**Enable Hardware Transfer Request for DMA Channel 0n** see table below
ECH1n (n = 0-7)	8+n	w	**Enable Hardware Transfer Request for DMA Channel 1n** see table below
DCH0n (n = 0-7)	16+n	w	**Disable Hardware Transfer Request for DMA Channel 0n** see table below
DCH1n (n = 0-7)	24+n	w	**Disable Hardware Transfer Request for DMA Channel 1n** see table below

Table 11-9 Conditions to Set/Reset the Bits TRSR.HTREmn

HTREQ.ECHmn	HTREQ.DCHmn	Transaction Finishes[1)] for Channel mn	Modification of TRSR.HTREmn
0	0	0	Unchanged
1	0	0	Set

Table 11-9 Conditions to Set/Reset the Bits TRSR.HTREmn (cont'd)

HTREQ.ECHmn	HTREQ.DCHmn	Transaction Finishes[1] for Channel mn	Modification of TRSR.HTREmn
X	1	X	Reset
X	X	1	Reset

1) In Single Mode only. In Continuous Mode, the end of a transaction has no impact.

The Enable Error Register describes how the DMA controller reacts to errors. It enables the interrupts for the loss of a transaction request or Move Engine errors.

DMA_EER
DMA Enable Error Register (020_H) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TRLINP				ME1INP				ME0INP				EME1DER	EME1SER	EME0DER	EME0SER
rw				rw				rw				rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ETRL17	ETRL16	ETRL15	ETRL14	ETRL13	ETRL12	ETRL11	ETRL10	ETRL07	ETRL06	ETRL05	ETRL04	ETRL03	ETRL02	ETRL01	ETRL00
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
ETRL0n (n = 0-7)	n	rw	**Enable Transaction Request Lost for DMA Channel 0n** This bit enables the generation of an interrupt when the set condition for ERRSR.TRL0n is detected. 0_B The interrupt generation for a request lost event for channel 0n is disabled. 1_B The interrupt generation for a request lost event for channel 0n is enabled.
ETRL1n (n = 0-7)	8+n	rw	**Enable Transaction Request Lost for DMA Channel 1n** This bit enables the generation of an interrupt when the set condition for ERRSR.TRL1n is detected. 0_B The interrupt generation for a request lost event for channel 1n is disabled. 1_B The interrupt generation for a request lost event for channel 1n is enabled.
EME0SER	16	rw	**Enable Move Engine 0 Source Error** This bit enables the generation of a Move Engine 0 source error interrupt. 0_B Move Engine 0 source error interrupt is disabled. 1_B Move Engine 0 source error interrupt is enabled.

Field	Bits	Type	Description
EME0DER	17	rw	**Enable Move Engine 0 Destination Error** This bit enables the generation of a Move Engine 0 destination error interrupt. 0_B Move Engine 0 destination error interrupt is disabled. 1_B Move Engine 0 destination error interrupt is enabled.
EME1SER	18	rw	**Enable Move Engine 1 Source Error** This bit enables the generation of a Move Engine 1 source error interrupt. 0_B Move Engine 1 source error interrupt is disabled. 1_B Move Engine 1 source error interrupt is enabled.
EME1DER	19	rw	**Enable Move Engine 0 Destination Error** This bit enables the generation of a Move Engine 0 destination error interrupt. 0_B Move Engine 1 destination error interrupt is disabled. 1_B Move Engine 1 destination error interrupt is enabled.
ME0INP	[23:20]	rw	**Move Engine 0 Error Interrupt Node Pointer** ME0INP determines the number n (n = 0-15) of the service request output SRn that becomes active on a Move Engine 0 source or destination interrupt. 0000_B SR0 selected for Move Engine 0 interrupt 0001_B SR1 selected for Move Engine 0 interrupt $\ldots_B$ … 1111_B SR15 selected for Move Engine 0 interrupt *Note: In the TC1797, SR[7:0] are connected to interrupt nodes. SR[15:8] are used as DMA channel request inputs (**Page 11-102**).*

Field	Bits	Type	Description
ME1INP	[27:24]	rw	**Move Engine 1 Error Interrupt Node Pointer** ME1INP determines the number n (n = 0-15) of the service request output SRn that becomes active on a Move Engine 1 source or destination interrupt. 0000_B SR0 selected for Move Engine 1 interrupt 0001_B SR1 selected for Move Engine 1 interrupt $\ldots_B$ … 1111_B SR15 selected for Move Engine 1 interrupt *Note: In the TC1797, SR[7:0] are connected to interrupt nodes. SR[15:8] are used as DMA channel request inputs (**Page 11-102**).*
TRLINP	[31:28]	rw	**Transaction Lost Interrupt Node Pointer** TRLINP determines the number n (n = 0-15) of the service request output SRn that becomes active on a transaction lost interrupt. 0000_B SR0 selected for transaction lost interrupt 0001_B SR1 selected for transaction lost interrupt $\ldots_B$ … 1111_B SR15 selected for transaction lost interrupt *Note: In the TC1797, SR[7:0] are connected to interrupt nodes. SR[15:8] are used as DMA channel request inputs (**Page 11-102**).*

The Error Status Register indicates if the DMA controller could not answer to a request because the previous request was not terminated (see **Section 11.2.4.4**). It indicates also the FPI Bus accesses that have been terminated with errors.

DMA_ERRSR
DMA Error Status Register (024$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	LEC ME1			MLI0	LEC ME0			0	CERBERUSER	LMBER	FPIER	ME1DER	ME1SER	ME0DER	ME0SER
r	rh			rh	rh			r	rh	rh	rh	rh	rh	rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRL 17	TRL 16	TRL 15	TRL 14	TRL 13	TRL 12	TRL 11	TRL 10	TRL 07	TRL 06	TRL 05	TRL 04	TRL 03	TRL 02	TRL 01	TRL 00
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
TRL0n (n = 0-7)	n	rh	**Transaction/Transfer Request Lost of DMA Channel 0n** 0_B 0 No request lost event has been detected for channel 0n. 1_B 1 A new DMA request was detected while TRSR.CH0n=1 (request lost event). This bit is reset by software when writing a 1 to CLRE.CTL0n, or by a channel reset (writing CHRSTR.CH0n = 1).
TRL1n (n = 0-7)	8+n	rh	**Transaction/Transfer Request Lost of DMA Channel 1n** 0_B No request lost event has been detected for channel 1n. 1_B A new DMA request was detected while TRSR.CH1n=1 (request lost event). This bit is reset by software when writing a 1 to CLRE.CTL1n, or by a channel reset (writing CHRSTR.CH1n = 1).

Field	Bits	Type	Description
ME0SER	16	rh	**Move Engine 0 Source Error** This bit is set whenever a Move Engine 0 error occurred during a source (read) move of a DMA transfer, or a request could not been serviced due to the access protection. 0_B No Move Engine 0 source error has occurred. 1_B A Move Engine 0 source error has occurred.
ME0DER	17	rh	**Move Engine 0 Destination Error** This bit is set whenever a Move Engine 0 error occurred during a destination (write) move of a DMA transfer, or a request could not been serviced due to the access protection. 0_B No Move Engine 0 destination error has occurred. 1_B A Move Engine 0 destination error has occurred.
ME1SER	18	rh	**Move Engine 1 Source Error** This bit is set whenever a Move Engine 1 error occurred during a source (read) move of a DMA transfer, or a request could not been serviced due to the access protection. 0_B No Move Engine 1 source error has occurred. 1_B A Move Engine 1 source error has occurred.
ME1DER	19	rh	**Move Engine 1 Destination Error** This bit is set whenever a Move Engine 1 error occurred during a destination (write) move of a DMA transfer, or a request could not been serviced due to the access protection. 0_B No Move Engine 1 destination error has occurred. 1_B A Move Engine 1 destination error has occurred.
FPIER	20	rh	**SPB Error** This bit is set whenever a move that has been started by the DMA/MLI FPI master interface leads to an error on the FPI Bus. 0_B No error occurred. 1_B An error occurred on FPI Bus interface.

Field	Bits	Type	Description
LMBER	21	rh	**LMB Error** This bit is set whenever a move that has been started by the DMA/MLI LMB master interface leads to an error on the LMB Bus. 0_B No error occurred. 1_B An error occurred on LMB Bus interface.
CERBERUSER	22	rh	**Cerberus Error Source** This bit is set whenever an On Chip Bus error occurred due to an action of Cerberus. 0_B No On Chip Bus error occurred due to Cerberus. 1_B An On Chip Bus error occurred due to Cerberus.
LECME0	[26:24]	rh	**Last Error Channel Move Engine 0** This bit field indicates the channel number of the last channel of Move Engine 0 leading to an On Chip Bus error that has occurred.
MLI0	27	rh	**MLI0 Error Source** This bit is set whenever an On Chip Bus error occurred due to an action of MLI0. 0_B No On Chip Bus error occurred due to MLI0. 1_B An On Chip Bus error occurred due to MLI0.
LECME1	[30:28]	rh	**Last Error Channel Move Engine 1** This bit field indicates the channel number of the last channel of Move Engine 1 leading to an On Chip Bus error that has occurred.
0	23, 31	r	**Reserved** Read as 0; should be written with 0.

The Clear Error contains bits that make it possible to clear the Transaction Request Lost flags or the Move Engine error flags.

DMA_CLRE
DMA Clear Error Register (028$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	0			CLRMLI0	0				CLCERBERUS	CLMBER	CFPIER	CME1DER	CME1SER	CME0DER	CME0SER
r	r			w	r				w	w	w	rw	rw	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTL17	CTL16	CTL15	CTL14	CTL13	CTL12	CTL11	CTL10	CTL07	CTL06	CTL05	CTL04	CTL03	CTL02	CTL01	CTL00
rw	rw	rw	rw	rw	rw	rw	rw	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
CTL0n (n = 0-7)	n	w	**Clear Transaction Request Lost for DMA Channel 0n** 0_B No action 1_B Clear DMA channel 0n transaction request lost flag ERRSR.TRL0n
CTL1n (n = 0-7)	n+8	w	**Clear Transaction Request Lost for DMA Channel 1n** 0_B No action 1_B Clear DMA channel 1n transaction request lost flag ERRSR.TRL1n
CME0SER	16	w	**Clear Move Engine 0 Source Error** 0_B No action 1_B Clear source error flag ERRSR.ME0SER.
CME0DER	17	w	**Clear Move Engine 0 Destination Error** 0_B No action 1_B Clear destination error flag ERRSR.ME0DER.
CME1SER	18	w	**Clear Move Engine 1 Source Error** 0_B No action 1_B Clear source error flag ERRSR.ME1SER.

Field	Bits	Type	Description
CME1DER	19	w	**Clear Move Engine 1 Destination Error** 0_B No action 1_B Clear destination error flag ERRSR.ME1DER.
CFPIER	20	w	**Clear FPI Error** 0_B No action 1_B Clear error flag ERRSR.FPIER.
CLMBER	21	w	**Clear LMB Error** 0_B No action 1_B Clear error flag ERRSR.LMBER.
CLCERBERUS	22	w	**Clear Cerberus Error** 0_B No action 1_B Clear error flag ERRSR.Cerberus.
CLRMLI0	27	w	**Clear MLI0 Error** 0_B No action 1_B Clear error flag ERRSR.MLI0.
0	[26:23], [30:28], 31	r	**Reserved** Read as 0; should be written with 0.

The Interrupt Status Register indicates if CHSRmn.TCOUNT matches with CHCRmn.IRDV, or if CHSRmn.TCOUNT has been decremented (depending on CHICRmn.INTCT[0]),or if a pattern has been detected. These conditions can also generate an interrupt if enabled (see **Figure 11-17** on **Page 11-32**).

DMA_INTSR

DMA Interrupt Status Register (054$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IPM 17	**IPM 16**	**IPM 15**	**IPM 14**	**IPM 13**	**IPM 12**	**IPM 11**	**IPM 10**	**IPM 07**	**IPM 06**	**IPM 05**	**IPM 04**	**IPM 03**	**IPM 02**	**IPM 01**	**IPM 00**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICH 17	**ICH 16**	**ICH 15**	**ICH 14**	**ICH 13**	**ICH 12**	**ICH 11**	**ICH 10**	**ICH 07**	**ICH 06**	**ICH 05**	**ICH 04**	**ICH 03**	**ICH 02**	**ICH 01**	**ICH 00**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
ICH0n (n = 0-7)	n	rh	**Interrupt from Channel 0n** This bit indicates that channel 0n has raised an interrupt for TCOUNT = IRDV or if TCOUNT has been decremented (depending on CHICR.INTCT[0]. This bit (and IP0n) is reset by software when writing a 1 to INTCR.CICH0n or by a channel reset (writing CHRSTR.CH0n = 1). 0_B A channel interrupt has not been detected. 1_B A channel interrupt has been detected.
ICH1n (n = 0-7)	8+n	rh	**Interrupt from Channel 1n** This bit indicates that channel 1n has raised an interrupt for TCOUNT = IRDV or if TCOUNT has been decremented (depending on CHICR.INTCT[0]. This bit (and IP1n) is reset by software when writing a 1 to INTCR.CICH1n or by a channel reset (writing CHRSTR.CH1n = 1). 0_B A channel interrupt has not been detected. 1_B A channel interrupt has been detected.

Field	Bits	Type	Description
IPM0n (n = 0-7)	16+n	rh	**Pattern Detection from Channel 0n** This bit indicates that a pattern has been detected for channel 0n while the pattern detection has been enabled. This bit (and ICH0n) is reset by software when writing a 1 to INTCR.CICH0n or by a channel reset (writing CHRSTR.CH0n = 1). 0_B A pattern has not been detected. 1_B A pattern has been detected.
IPM1n (n = 0-7)	24+n	rh	**Pattern Detection from Channel 1n** This bit indicates that a pattern has been detected for channel 1n while the pattern detection has been enabled. This bit (and ICH1n) is reset by software when writing a 1 to INTCR.CICH1n or by a channel reset (writing CHRSTR.CH1n = 1). 0_B A pattern has not been detected. 1_B A pattern has been detected.

The Wrap Status Register gives information about the channels that did a wrap-around on their source or destination buffer(s). This condition can also lead to an interrupt if it is enabled.

DMA_WRPSR
DMA Wrap Status Register (05C$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
WRPD17	WRPD16	WRPD15	WRPD14	WRPD13	WRPD12	WRPD11	WRPD10	WRPD07	WRPD06	WRPD05	WRPD04	WRPD03	WRPD02	WRPD01	WRPD00
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WRPS17	WRPS16	WRPS15	WRPS14	WRPS13	WRPS12	WRPS11	WRPS10	WRPS07	WRPS06	WRPS05	WRPS04	WRPS03	WRPS02	WRPS01	WRPS00
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
WRPS0n (n = 0-7)	n	rh	**Wrap Source Buffer for Channel 0n** These bits indicate which channels have done a wrap-around of their source buffer(s). 0_B No wrap-around occurred for channel 0n. 1_B A wrap-around occurred for channel 0n. This bit is reset by software by writing a 1 to INTCR.CWRP0n or CHRSTR.CH0n.
WRPS1n (n = 0-7)	8+n	rh	**Wrap Source Buffer for Channel 1n** These bits indicate which channels have done a wrap-around of their source buffer(s). 0_B No wrap-around occurred for channel 1n. 1_B A wrap-around occurred for channel 1n. This bit is reset by software by writing a 1 to INTCR.CWRP1n or CHRSTR.CH1n.
WRPD0n (n = 0-7)	16+n	rh	**Wrap Destination Buffer for Channel 0n** These bits indicate which channels have done a wrap-around of their destination buffer(s). 0_B No wrap-around occurred for channel 0n. 1_B A wrap-around occurred for channel 0n. This bit is reset by software by writing a 1 to INTCR.CWRP0n or CHRSTR.CH0n.

Field	Bits	Type	Description
WRPD1n (n = 0-7)	24+n	rh	**Wrap Destination Buffer for Channel 1n** These bits indicate which channels have done a wrap-around of their destination buffer(s). 0_B No wrap-around occurred for channel 1n. 1_B A wrap-around occurred for channel 1n. This bit is reset by software by writing a 1 to INTCR.CWRP1n or CHRSTR.CH1n.

The bits in the Interrupt Clear Register make it possible to reset the channel interrupt flags and the wrap buffer interrupt flags for DMA Channels mn.

DMA_INTCR
DMA Interrupt Clear Register (058_H) Reset Value: 0000 0000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CWRP17	**CWRP16**	**CWRP15**	**CWRP14**	**CWRP13**	**CWRP12**	**CWRP11**	**CWRP10**	**CWRP07**	**CWRP06**	**CWRP05**	**CWRP04**	**CWRP03**	**CWRP02**	**CWRP01**	**CWRP00**
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CICH17	**CICH16**	**CICH15**	**CICH14**	**CICH13**	**CICH12**	**CICH11**	**CICH10**	**CICH07**	**CICH06**	**CICH05**	**CICH04**	**CICH03**	**CICH02**	**CICH01**	**CICH00**
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
CICH0n (n = 0-7)	n	w	**Clear Interrupt for DMA Channel 0n** These bits make it possible to reset the channel interrupt flags INTSR.ICH0n and INTSR.IPM0n of DMA channel 0n by software. 0_B No action. 1_B Bits INTSR.ICH0n and INTSR.IPM0n are reset.
CICH1n (n = 0-7)	8+n	w	**Clear Interrupt for DMA Channel 1n** These bits make it possible to reset the channel interrupt flags INTSR.ICH1n and INTSR.IPM1n of DMA channel 1n by software. 0_B No action. 1_B Bits INTSR.ICH1n and INTSR.IPM1n are reset.
CWRP0n (n = 0-7)	16+n	w	**Clear Wrap Buffer Interrupt for DMA Channel 0n** These bits make it possible to reset the wrap source buffer interrupt flag WRPSR.WRPS0n and the wrap destination buffer interrupt flag WRPSR.WRPD0n (both together) of DMA channel 0n by software. 0_B No action. 1_B Bits WRPSR.WRPS0n and WRPSR.WRPD0n are reset.

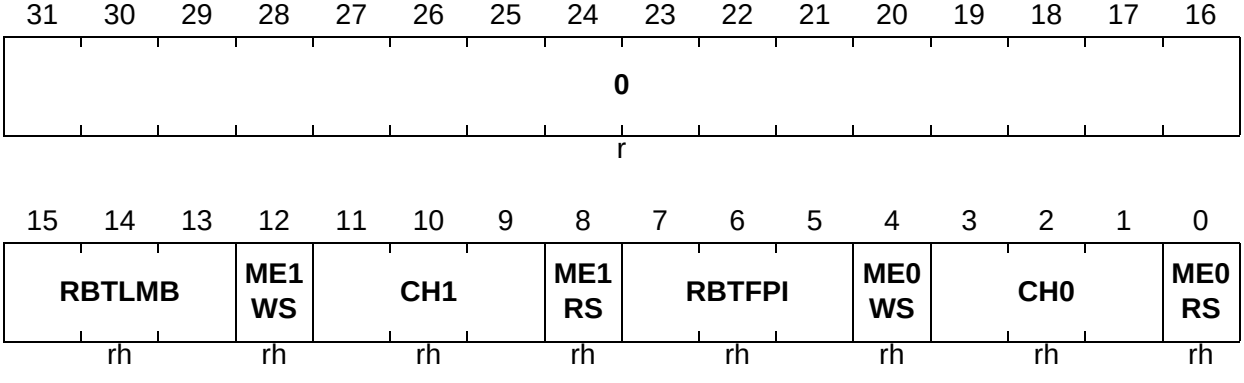
Field	Bits	Type	Description
CWRP1n (n = 0-7)	24+n	w	**Clear Wrap Buffer Interrupt for DMA Channel 1n** These bits make it possible to reset the wrap source buffer interrupt flag WRPSR.WRPS1n and the wrap destination buffer interrupt flag WRPSR.WRPD1n (both together) of DMA channel 1n by software. 0_B No action. 1_B Bits WRPSR.WRPS1n and WRPSR.WRPD1n are reset.

11.3.3 Move Engine Registers

The Move Engine Status Register is a read-only register that holds status information about the transaction handled by the Move Engines.

DMA_MESR

DMA Move Engine Status Register (030_H) Reset Value: 0000 0000$_H$

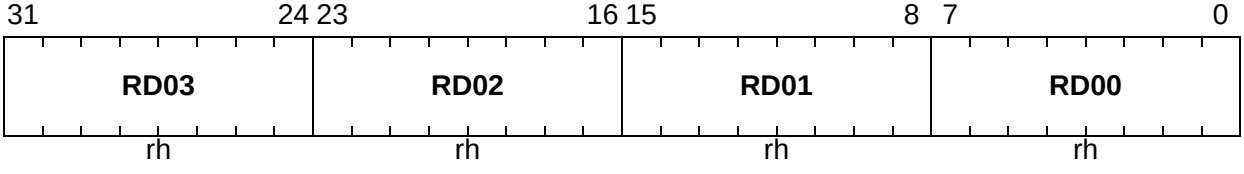


Field	Bits	Type	Description
ME0RS	0	rh	**Move Engine 0 Read Status** 0_B Move Engine 0 is not performing a read. 1_B Move Engine 0 is performing a read.
CH0	[3:1]	rh	**Reading Channel in Move Engine 0** This bit field indicates which channel number is currently being processed by the Move Engine 0.
ME0WS	4	rh	**Move Engine 0 Write Status** 0_B Move Engine 0 is not performing a write. 1_B Move Engine 0 is performing a write.

Field	Bits	Type	Description
RBTFPI	[7:5]	rh	**Read Buffer Trace for FPI Bus Interface** This bit field contains trace information from the buffer in the FPI Bus Interface. In the TC1797 it indicates the source of a bus access to the FPI Bus. 000_B Default value. 001_B DMA Move Engine 0 010_B DMA Move Engine 1 011_B MLI0 101_B Cerberus Other bit combinations are reserved. RBTFPI is useful for emulation purposes. It is not recommended to evaluate this bit field during normal operation of the TC1797.
ME1RS	8	rh	**Move Engine 1 Read Status** 0_B Move Engine 1 is not performing a read. 1_B Move Engine 1 is performing a read.
CH1	[11:9]	rh	**Reading Channel in Move Engine 1** These bit field indicates which channel number is currently being processed by the Move Engine 1.
ME1WS	12	rh	**Move Engine 1 Write Status** 0_B Move Engine 1 is not performing a write. 1_B Move Engine 1 is performing a write.
RBTLMB	[15:13]	rh	**Read Buffer Trace for LMB Bus Interface** This bit field contains trace information from the buffer in the LMB Bus Interface. In the TC1797, it indicates the source of a bus access to the LMB Bus. 000_B Default value 001_B DMA Move Engine 0 010_B DMA Move Engine 1 011_B MLI0 Others Reserved RBTLMB is useful for emulation purposes. It is not recommended to evaluate this bit field during normal operation of the TC1797.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

The Move Engine 0 Read Register indicates the value that has just been read by Move Engine 0. The value in this register is compared to the bits in register ME0PR according to the bit fields CHCRmn.PATSEL.

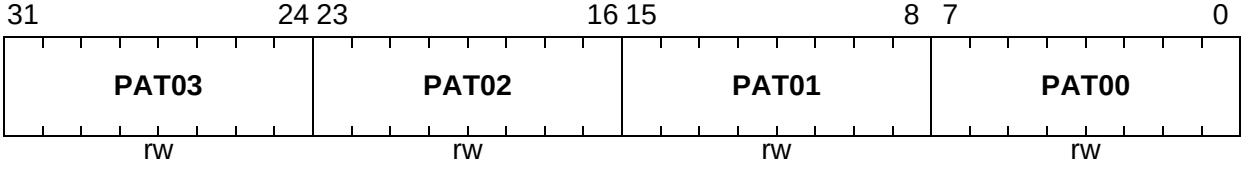
DMA_ME0R
DMA Move Engine 0 Read Register (034$_H$) Reset Value: 0000 0000$_H$
DMA_ME1R
DMA Move Engine 1 Read Register (038$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
RD00, RD01, RD02, RD03	[7:0], [15:8], [23:16], [31:24]	rh	**Read Value for Move Engine 0** Contains the 32-bit read data (four bytes RD0[3:0]) that is stored in the Move Engine 0 after each read move. The content of ME0R is overwritten after each read move of a DMA channel belonging to DMA Sub-block 0.

The Move Engine 0 Pattern Register contains the patterns (mask and/or compare bits) to be processed by the pattern detection logic in Move Engine 0.

DMA_ME0PR
DMA Move Engine 0 Pattern Register (03C$_H$) Reset Value: 0000 0000$_H$
DMA_ME1PR
DMA Move Engine 1Pattern Register (040$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
PAT00, PAT01, PAT02, PAT03	[7:0], [15:8], [23:16], [31:24]	rw	**Pattern for Move Engine 0** Determines up to four 8-bit compare patterns/mask patterns to be processed by the pattern detection logic in Move Engine 0. Depending on the pattern detection configuration (CHCR0n.PATSEL) and channel data width (CHCR0n.CHDW), the patterns are processed as bytes or half-words.

The DMA Move Engine 0 Access Enable Register controls the access protection. It enables/disables the address protection ranges x (x = 0-31) for Move Engine 0.

DMA_ME0AENR
DMA Move Engine 0 Access Enable Register
(044_H) **Reset Value: 0000 0000_H**

DMA_ME1AENR
DMA Move Engine 1 Access Enable Register
($04C_H$) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AEN 31	**AEN 30**	**AEN 29**	**AEN 28**	**AEN 27**	**AEN 26**	**AEN 25**	**AEN 24**	**AEN 23**	**AEN 22**	**AEN 21**	**AEN 20**	**AEN 19**	**AEN 18**	**AEN 17**	**AEN 16**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AEN 15	**AEN 14**	**AEN 13**	**AEN 12**	**AEN 11**	**AEN 10**	**AEN 9**	**AEN 8**	**AEN 7**	**AEN 6**	**AEN 5**	**AEN 4**	**AEN 3**	**AEN 2**	**AEN 1**	**AEN 0**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

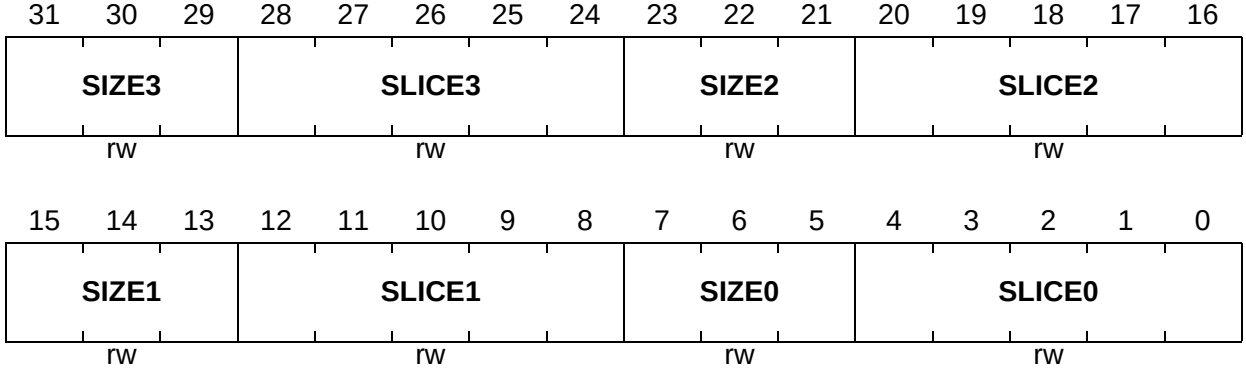
Field	Bits	Type	Description
AENx (x = 0-31)	x	rw	**Address Range x Enable** This bit enables the read and write capability of the DMA Move Engines for address range x (x = 0-31). 0_B DMA read and write moves to address range x are disabled 1_B DMA read and write moves to address range x are enabled If AENx = 0 for a read/write move to address range x, the read/write move is not executed and a source/destination Move Engine interrupt is generated.

*Note: See **Table 11-13** on **Page 11-112** for the TC1797-specific address range definition.*

The DMA Move Engine 0 Access Range Register determines number and size of the sub-ranges for address range extension n (n = 0-3). See also **Figure 11-26** for bit field definitions.

DMA_ME0ARR
DMA Move Engine 0 Access Range Register
(048_H) **Reset Value: 0000 0000_H**

DMA_ME1ARR
DMA Move Engine 1Access Range Register
(050_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
SLICE0	[4:0]	rw	**Address Slice 0** SLICE0 selects a specific sub-range within address range extension 0.
SIZE0	[7:5]	rw	**Address Size 0** SIZE0 determines the sub-range size within address range extension 0.
SLICE1	[12:8]	rw	**Address Slice 1** SLICE1 selects a specific sub-range within address range extension 1.
SIZE1	[15:13]	rw	**Address Size 1** SIZE1 determines the sub-range size within address range extension 1.
SLICE2	[20:16]	rw	**Address Slice 2** SLICE2 selects a specific sub-range within address range extension 2.

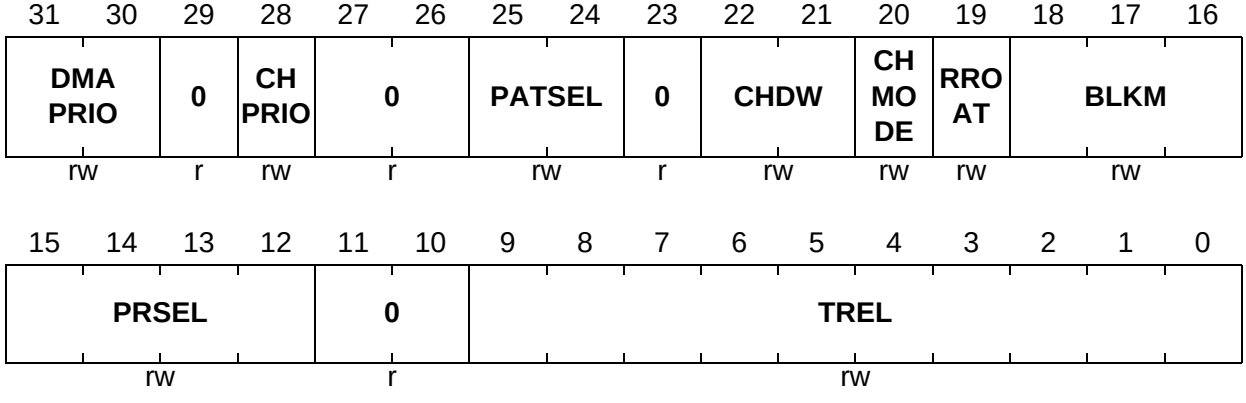
Field	Bits	Type	Description
SIZE2	[23:21]	rw	**Address Size 2** SIZE2 determines the sub-range size within address range extension 2.
SLICE3	[28:24]	rw	**Address Slice 3** SLICE3 selects a specific sub-range within address range extension 3.
SIZE3	[31:29]	rw	**Address Size 3** SIZE3 determines the sub-range size within address range extension 3.

*Note: See **Section 11.4.2** on **Page 11-112** for the TC1797-specific address range and address range extension definitions.*

11.3.4 Channel Control/Status Registers

The Channel Control Register for DMA channel mn contains its configuration and its control bits and bit fields.

DMA_CHCR0x (x = 0-7)
DMA Channel 0x Control Register (084_H+x*20_H) **Reset Value: 0000 0000_H**
DMA_CHCR1x (x = 0-7)
DMA Channel 1x Control Register (184_H+x*20_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
TREL	[9:0]	rw	**Transfer Reload Value** This bit field contains the number of DMA transfers for s DMA transaction of DMA channel mn. This10-bit transfer count value is loaded into CHSRmn.TCOUNT at the start of a DMA transaction (when TRSR.CHmn becomes set and CHSRmn.TCOUNT = 0). A write to CHSRmn.TREL during a running DMA transaction has no influence to the running DMA transaction. If CHSRmn.TREL = 0 or if CHSRmn.TREL = 1, CHSRmn.TCOUNT will be loaded with 1 when a new transaction is started (at least one DMA transfer must be executed per DMA transaction).

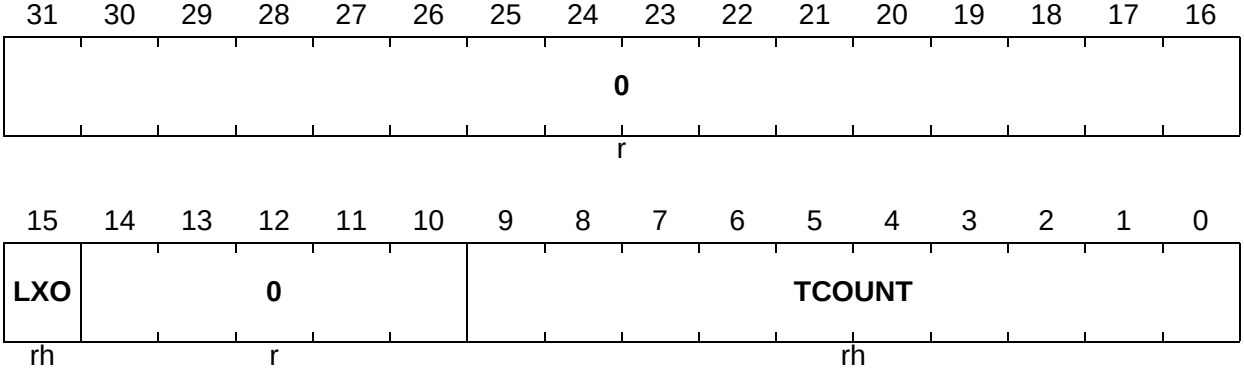
Field	Bits	Type	Description
PRSEL	[15:12]	rw	**Peripheral Request Select** This bit field controls the hardware request input multiplexer of DMA channel mn (see **Figure 11-6** on **Page 11-11**). 0000_B Input CHmn_REQI0 selected 0001_B Input CHmn_REQI1 selected 0010_B Input CHmn_REQI2 selected 0011_B Input CHmn_REQI3 selected 0100_B Input CHmn_REQI4 selected 0101_B Input CHmn_REQI5 selected 0110_B Input CHmn_REQI6 selected 0111_B Input CHmn_REQI7 selected 1000_B Input CHmn_REQI8 selected 1001_B Input CHmn_REQI9 selected 1010_B Input CHmn_REQI10 selected 1011_B Input CHmn_REQI11 selected 1100_B Input CHmn_REQI12 selected 1101_B Input CHmn_REQI13 selected 1110_B Input CHmn_REQI14 selected 1111_B Input CHmn_REQI15 selected
BLKM	[18:16]	rw	**Block Mode** BLKM determines the number of DMA moves executed during one DMA transfer. 000_B One DMA transfer has 1 DMA move 001_B One DMA transfer has 2 DMA move 010_B One DMA transfer has 4 DMA move 011_B One DMA transfer has 8 DMA move 100_B One DMA transfer has 16 DMA move Other bit combinations are reserved and must not be used. See also **Figure 11-10** on **Page 11-18**.
RROAT	19	rw	**Reset Request Only After Transaction** RROAT determines whether or not the TRSR.CHmn transfer request state flag is reset after each transfer. 0_B TRSR.CHmn is reset after each transfer. A transfer request is required for each transfer. 1_B TRSR.CHmn is reset when CHSRmn.TCOUNT = 0 after a transfer. One transfer request starts a complete DMA transaction

Field	Bits	Type	Description
CHMODE	20	rw	**Channel Operation Mode** CHMODE determines the reset condition for control bit TRSR.HTREmn of DMA channel mn. 0_B Single Mode operation is selected for DMA channel mn. After a transaction, DMA channel mn is disabled for further hardware requests (TRSR.HTREmn is reset by hardware) TRSR.HTREmn must be set again by software for starting a new transaction. 1_B Continuous Mode operation is selected for DMA channel mn. After a transaction, bit TRSR.HTREmn remains set
CHDW	[22:21]	rw	**Channel Data Width** CHDW determines the data width for the read and write moves of DMA channel mn. 00_B 8-bit (byte) data width for moves selected 01_B 16-bit (half-word) data width for moves selected 10_B 32-bit (word) data width for moves selected 11_B Reserved
PATSEL	[25:24]	rw	**Pattern Select** This bit field selects the mode of the pattern detection logic. Depending on the channel data width, PATSEL selects different pattern detection configurations. If pattern detection is enabled (PATSEL not equal 00_B), the pattern detection interrupt line will be activated on the selected pattern match. **8-bit channel data width (CHDW = 00_B):** Selected pattern detection configuration see **Table 11-4** on **Page 11-41**. **16-bit channel data width (CHDW = 01_B):** Selected pattern detection configuration see **Table 11-5** on **Page 11-42**. **32-bit channel data width (CHDW = 10_B):** Selected pattern detection configuration see **Table 11-6** on **Page 11-44**.

Field	Bits	Type	Description
CHPRIO	28	rw	**Channel Priority** CHPRIO determines the priority of DMA channel n for the Move Engine minternal channel arbitration. This priority is used for the case when multiple channels of Move Engine m are triggered in parallel. 0_B DMA channel mn has a low channel priority 1_B DMA channel mn has a high channel priority
DMAPRIO	[31:30]	rw	**DMA Priority** This bit determines the DMA the request priority that is used when a move operation related to channel mn is requesting an On Chip Bus. This bit has no effect in channel prioritization inside the Move Engine m in. 00_B Low priority selected 01_B Medium priority selected 10_B Reserved 11_B High priority selected
0	[11:10], 23, [27:26], 29	r	**Reserved** Read as 0; should be written with 0.

The Channel Status Register contains the current transfer count and a pattern detection compare result.

DMA_CHSR0x (x = 0-7)
DMA Channel 0x Status Register (080_H+x*20_H) Reset Value: 0000 0000_H
DMA_CHSR1x (x = 0-7)
DMA Channel 1x Status Register (180_H+x*20_H) Reset Value: 0000 0000_H



Field	Bits	Type	Description
TCOUNT	[9:0]	rh	**Transfer Count Status** TCOUNT holds the actual value of the DMA transfer count for DMA channel mx. TCOUNT is loaded with the value of CHCRmx.TREL when TRSR.CHmx becomes set (and TCOUNT = 0). After each DMA transfer, TCOUNT is decremented by 1.
LXO	15	rh	**Old Value of Pattern Detection** This bit contains the compare result of a pattern compare operation when 8-bit or 16-bit data width is selected. 8-bit data width: see **Table 11-4** and **Figure 11-23** 16-bit data width: see **Table 11-5** and **Figure 11-24** 0_B The corresponding pattern compare operation did not find a pattern match on the last move 1_B The corresponding pattern compare operation found a pattern match at the last move
0	[14:10], [31:16]	r	**Reserved** Read as 0; should be written with 0.

The Channel Interrupt Control Register controls the interrupts generation.

DMA_CHICR0x (x = 0-7)
DMA Channel 0x Interrupt Control Register
(088_H+x*20_H) **Reset Value: 0000 0000_H**

DMA_CHICR1x (x = 0-7)
DMA Channel 1x Interrupt Control Register
(188_H+x*20_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IRDV				INTP				WRPP				INTCT		WRPDE	WRPSE
rw				rw				rw				rw		rw	rw

Field	Bits	Type	Description
WRPSE	0	rw	**Wrap Source Enable** 0_B Wrap source buffer interrupt disabled 1_B Wrap source buffer interrupt enabled
WRPDE	1	rw	**Wrap Destination Enable** 0_B Wrap destination buffer interrupt disabled 1_B Wrap destination buffer interrupt enabled
INTCT	[3:2]	rw	**Interrupt Control** 00_B No interrupt will be generated on changing the TCOUNT value. The bit INTSR.ICHmx is set when TCOUNT equals IRDV. 01_B No interrupt will be generated on changing the TCOUNT value. The bit INTSR.ICHmx is set when TCOUNT is decremented 10_B An interrupt is generated and bit INTSR.ICHmx is set each time TCOUNT equals IRDV 11_B Interrupt is generated and bit INTSR.ICHmx is set each time TCOUNT is decremented *Note: see **Figure 11-17**.*

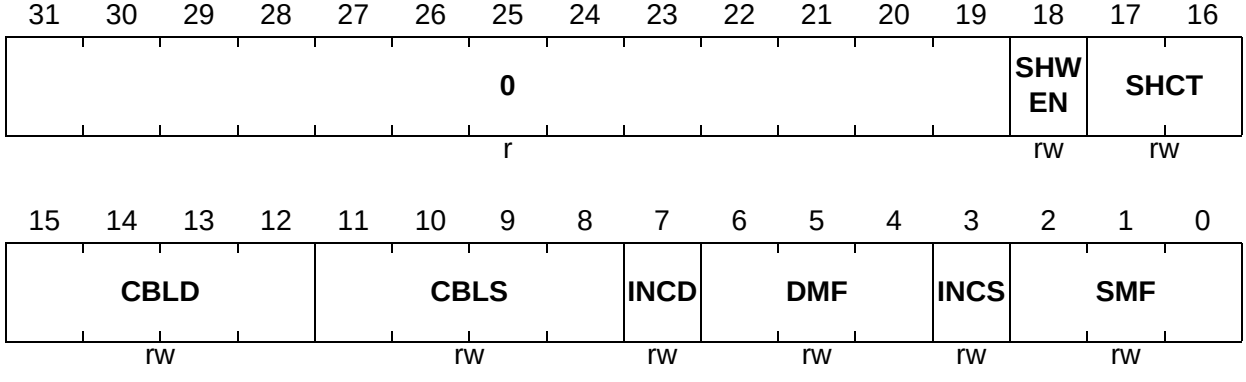
Field	Bits	Type	Description
WRPP	[7:4]	rw	**Wrap Pointer** WRPP determines the number n (n = 0-15) of the service request output SRn that becomes active on a wrap buffer interrupt. 0000_B SR0 selected for channel mx wrap buffer interrupt 0001_B SR1 selected for channel mx wrap buffer interrupt $\ldots_B$ … 1111_B SR15 selected for channel mx wrap buffer interrupt *Note: In the TC1797, SR[7:0] are connected to interrupt nodes. SR[15:8] are used as DMA channel request inputs (**Page 11-102**).*
INTP	[11:8]	rw	**Interrupt Pointer** INTP determines the number n (n = 0-15) of the service request output SRn that becomes active on a channel interrupt. 0000_B SR0 selected for channel mx interrupt 0001_B SR1 selected for channel mx interrupt $\ldots_B$ … 1111_B SR15 selected for channel mx interrupt *Note: In the TC1797, SR[7:0] are connected to interrupt nodes. SR[15:8] are used as DMA channel request inputs (**Page 11-102**).*
IRDV	[15:12]	rw	**Interrupt Raise Detect Value** These bits specify the value of CHSRmx.TCOUNT for which the Interrupt Threshold Limit should be raised.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

Note: The interrupt node of the wrap-around interrupts is shared with the pattern match interrupt. In order to support interrupt generation in case of a pattern match, the wrap-around interrupt should be disabled. If the wrap-around interrupts are used, the pattern match interrupt should not be used. The settings are independent for each DMA channel.

The Address Control Register controls how source and destination addresses are updated after a DMA move. Furthermore, it determines whether or not a source or destination address register update is shadowed.

DMA_ADRCR0x (x = 0-7)
DMA Channel 0x Address Control Register
($08C_H$+x*20_H) **Reset Value: 0000 0000_H**

DMA_ADRCR1x (x = 0-7)
DMA Channel 1x Address Control Register
($18C_H$+x*20_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
SMF	[2:0]	rw	**Source Address Modification Factor** This bit field and the data width as defined in CHCRmx.CHDW determine an address offset value by which the source address is modified after each DMA move. See also **Table 11-10**. 000_B Address offset is 1 x CHCRmx.CHDW 001_B Address offset is 2 x CHCRmx.CHDW 010_B Address offset is 4 x CHCRmx.CHDW 011_B Address offset is 8 x CHCRmx.CHDW 100_B Address offset is 16 x CHCRmx.CHDW 101_B Address offset is 32 x CHCRmx.CHDW 110_B Address offset is 64 x CHCRmx.CHDW 111_B Address offset is 128 x CHCRmx.CHDW

Field	Bits	Type	Description
INCS	3	rw	**Increment of Source Address** This bit determines whether the address offset as selected by SMF will be added to or subtracted from the source address after each DMA move. The source address is not modified if CBLS = 0000_B. 0_B Address offset will be subtracted 1_B Address offset will be added.
DMF	[6:4]	rw	**Destination Address Modification Factor** This bit field and the data width as defined in CHCRmx.CHDW determines an address offset value by which the destination address is modified after each DMA move. The destination address is not modified if CBLD = 0000_B. See also **Table 11-10**. 000_B Address offset is 1 x CHDW 001_B Address offset is 2 x CHDW 010_B Address offset is 4 x CHDW 011_B Address offset is 8 x CHDW 100_B Address offset is 16 x CHDW 101_B Address offset is 32 x CHDW 110_B Address offset is 64 x CHDW 111_B Address offset is 128 x CHDW
INCD	7	rw	**Increment of Destination Address** This bit determines whether the address offset as selected by DMF will be added to or subtracted from the destination address after each DMA move. The destination address is not modified if CBLD = 0000_B. 0_B Address offset will be subtracted 1_B Address offset will be added

Field	Bits	Type	Description
CBLS	[11:8]	rw	**Circular Buffer Length Source** This bit field determines which part of the 32-bit source address register remains unchanged and is not updated after a DMA move operation (see also **Section 11.2.4.7**). Therefore, CBLS also determines the size of the circular source buffer. 0000_B Source address SADR[31:0] is not updated 0001_B Source address SADR[31:1] is not updated 0010_B Source address SADR[31:2] is not updated 0011_B Source address SADR[31:3] is not updated $...B$ ... 1110_B Source address SADR[31:14] is not updated 1111_B Source address SADR[31:15] is not updated
CBLD	[15:12]	rw	**Circular Buffer Length Destination** This bit field determines which part of the 32-bit destination address register remains unchanged and is not updated after a DMA move operation (see also **Page 11-20**). Therefore, CBLD also determines the size of the circular destination buffer. 0000_B Destination address DADR[31:0] is not updated 0001_B Destination address DADR[31:1] is not updated 0010_B Destination address DADR[31:2] is not updated 0011_B Destination address DADR[31:3] is not updated $...B$ ... 1110_B Destination address DADR[31:14] is not updated 1111_B Destination address DADR[31:15] is not updated

Field	Bits	Type	Description
SHCT	[17:16]	rw	**Shadow Control** This bit field determines whether an address is transferred into the shadow address register when writing to source or destination address register. 00_B Shadow address register not used. Source and destination address register are written directly 01_B Shadow address register used for source address buffering. When writing to SADRmx, the address is buffered in SHADRmx and transferred to SADRmx with the start of the next DMA transaction 10_B Shadow address register used for destination address buffering. When writing to DADRmx, the address is buffered in SHADRmx and transferred to DADRmx with the start of the next DMA transaction 11_B Reserved In case of SHCT = 01_B or 10_B, SHCT must not be changed until the next DMA transaction has been started.
SHWEN	18	rw	**Shadow Address Register Write Enable** This bit determines whether the shadow address register SHADRmx is read only and automatically set to 0000 0000_H or if the shadow register can also be directly written and not modified when and shadow transfer takes place. 0_B Shadow address register is read only and the value stored in the SHADRmx is automatically set to 0000 0000_H when the shadow transfer takes place 1_B Shadow address register SHADRmx can be read and can be directly written. The value stored in the SHADRmx is not automatically modified when the shadow transfer takes place
0	[31:19]	r	**Reserved** Read as 0; should be written with 0.

Table 11-10 shows the offset values that are added or subtracted to/from a source or destination address register after a DMA move. Bit field SMF and bit INCS determine the offset value for the source address. Bit field DMF and bit INCD determine the offset value for the destination address.

Table 11-10 Address Offset Calculation Table

CHCRmn.CHDW = 00_B (8-bit Data Width)			CHCRmn.CHDW = 01_B (16-bit Data Width)			CHCRmn.CHDW = 10_B (32-bit Data Width)		
SMF DMF	**INCS INCD**	**Address Offset**	**SMF DMF**	**INCS INCD**	**Address Offset**	**SMF DMF**	**INCS INCD**	**Address Offset**
000_B	0	-1	000_B	0	-2	000_B	0	-4
	1	+1		1	+2		1	+4
001_B	0	-2	001_B	0	-4	001_B	0	-8
	1	+2		1	+4		1	+8
010_B	0	-4	010_B	0	-8	010_B	0	-16
	1	+4		1	+8		1	+16
011_B	0	-8	011_B	0	-16	011_B	0	-32
	1	+8		1	+16		1	+32
100_B	0	-16	100_B	0	-32	100_B	0	-64
	1	+16		1	+32		1	+64
101_B	0	-32	101_B	0	-64	101_B	0	-128
	1	+32		1	+64		1	+128
110_B	0	-64	110_B	0	-128	110_B	0	-256
	1	+64		1	+128		1	+256
111_B	0	-128	111_B	0	-256	111_B	0	-512
	1	+128		1	+256		1	+512

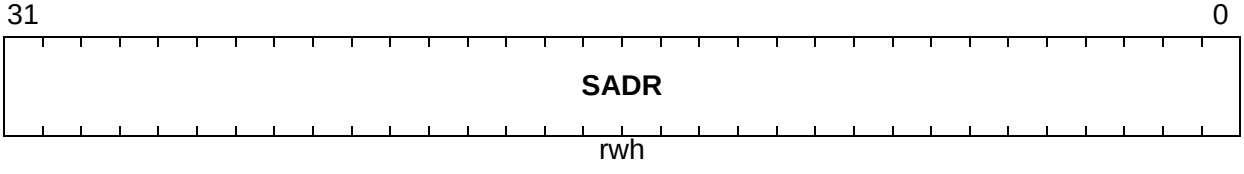
Note: CHCRmn.CHDW = 11_B is reserved and should not be used.

11.3.5 Channel Address Registers

The Source Address Register contains the 32-bit source address. If a DMA channel mn is active, SADRmn is updated continuously (if programmed) and shows the actual source address that is used for read moves within DMA transfers.

DMA_SADR0x (x = 0-7)
DMA Channel 0x Source Address Register
(090_H+x*20_H) **Reset Value: 0000 0000_H**

DMA_SADR1x (x = 0-7)
DMA Channel 1x Source Address Register
(190_H+x*20_H) **Reset Value: 0000 0000_H**



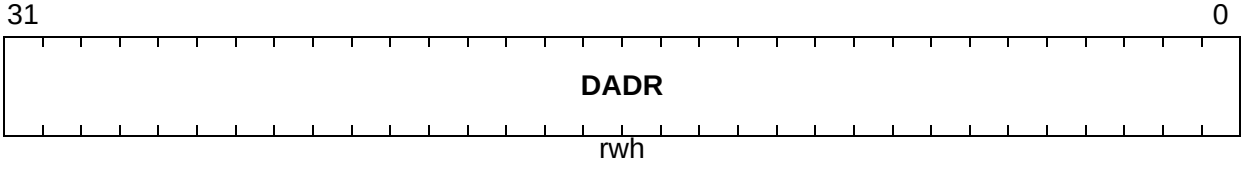
Field	Bits	Type	Description
SADR	[31:0]	rwh	**Source Start Address** This bit field holds the actual 32-bit source address of DMA channel mx that is used for read moves.

A write to SADRmn is executed directly only when the DMA channel mn is inactive (CHSRmn.TCOUNT = 0 and TRSR.CHmn = 0). If DMA channel mn is active when writing to SADRmn, the source address will not be written into SADRmn directly but will be buffered in the shadow register SHADRmn until the start of the next DMA transaction. During this shadowed address register operation, bit field ADRCRmn.SHCT must be set to 01_B.

The Destination Address Register contains the 32-bit destination address. If a DMA channel is active, DADRmn is updated continuously (if programmed) and shows the actual destination address that is used for write moves within DMA transfers.

DMA_DADR0x (x = 0-7)
DMA Channel 0x Destination Address Register
(094_H+x*20_H) **Reset Value: 0000 0000_H**

DMA_DADR1x (x = 0-7)
DMA Channel 1x Destination Address Register
(194_H+x*20_H) **Reset Value: 0000 0000_H**



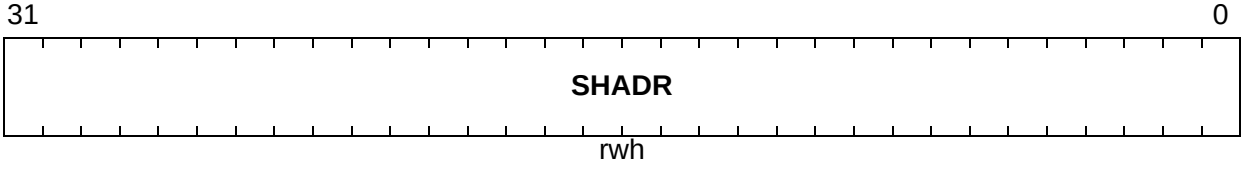
Field	Bits	Type	Description
DADR	[31:0]	rwh	**Destination Address** This bit field holds the actual 32-bit destination address of DMA channel mx that is used for write moves.

A write to DADRmn is executed directly only when the DMA channel mn is inactive (CHSRmn.TCOUNT = 0 and TRSR.CHmn = 0). If DMA channel mn is active when writing to DADRmn, the source address will not be written into DADRmn directly but will be buffered in the shadow register SHADRmn until the start of the next DMA transaction. During this shadowed address register operation, bit field ADRCRmn.SHCT must be set to 10_B.

The Shadow Address Register holds the shadowed source or destination address before it is written into the source or destination address register. SHADRmn can be read only.

DMA_SHADR0x (x = 0-7)
DMA Channel 0x Shadow Address Register
(098_H+x*20_H) **Reset Value: 0000 0000_H**

DMA_SHADR1x (x = 0-7)
DMA Channel 1x Shadow Address Register
(198_H+x*20_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
SHADR	[31:0]	rwh	**Shadowed Address** This bit field holds the shadowed 32-bit source or destination address of DMA channel mx.

SHADRmn is written when source or destination address buffering is selected (ADRCRmn.SHCT = 01_B or ADRCRmn.SHCT = 10_B) and a transaction is running. While the shadow mechanism is disabled, SHADR is set to 0000 0000_H.

If ADRCRmn.SHWEN = 0 the value stored in the SHADR is automatically set to 0000 0000_H when the shadow transfer takes place. The user can read the shadow register in order to detect if the shadow transfer has already taken place. If the value in SHADR is 0000 0000_H, no shadow transfer can take place and the corresponding address register is modified according to the circular buffer rules.

If ADRCRmn.SHWEN = 1 shadow register SHADRmn can be directly written. The value stored in the SHADRmn is not modified when the shadow transfer takes place, the shadow mechanism remains active and the shadow transfer will be repeated until Channel mn is reset or until the value in SHADR is 0000 0000_H, is written into the shadow register.

11.4 DMA Module Implementation

This section describes the TC1797 DMA module interfaces with the clock control, interrupt control, and address decoding.

Figure 11-28 shows the TC1797-specific implementation details and interconnections of the DMA module. The DMA module is supplied with a separate clock control, address decoding, interrupt control, and the request input wiring matrix.

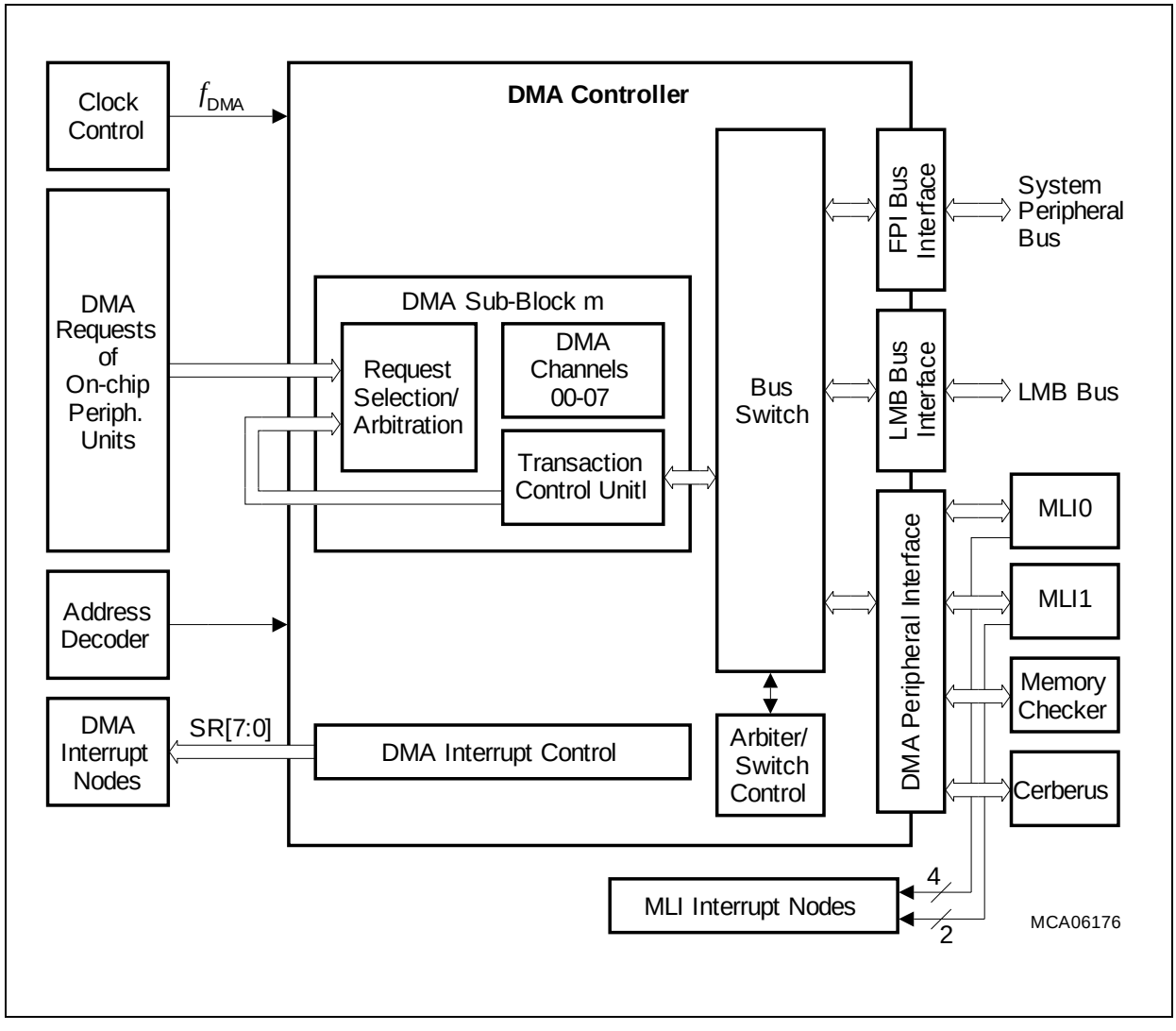


Figure 11-28 DMA Module Implementation and Interconnections

The request sources of the peripheral modules (ADC0, MSC0, MLI0, FADC, MultiCAN, and SCU) are associated with Interrupt Node Pointers and individual interrupt enable bits. As a result, each of the internal requests of a module can be routed independently to any of the interrupt output lines (INT_Ox) of the module.

11.4.1 DMA Request Wiring Matrix

The DMA request input lines of each DMA channel within DMA Sub-Block 0 and DMA Sub-Block 1 are connected to request output lines from the peripheral modules according to **Table 11-11**.

Table 11-11 DMA Request Assignment for DMA Sub-Block 0

DMA Channel	DMA Request Line	DMA Requesting Unit	Selected by
00	DMA_SR08	DMA(INT_O08)	CHCR00.PRSEL = 0000_B
	IOUT0	SCU (ERU)	CHCR00.PRSEL = 0001_B
	FADC_SR00	FADC	CHCR00.PRSEL = 0010_B
	ADC_SR00	ADC	CHCR00.PRSEL = 0011_B
	SSC0_RDR	SSC0	CHCR00.PRSEL = 0100_B
	ASC0_RDR	ASC0	CHCR00.PRSEL = 0101_B
	CAN_INT_O0	MultiCAN	CHCR00.PRSEL = 0110_B
	MLI0_SR4	MLI0	CHCR00.PRSEL = 0111_B
	STMIRQ0	STM	CHCR00.PRSEL = 1000_B
	GPTA_TRIG00	GPTA[1)]	CHCR00.PRSEL = 1001_B
	GPTA_TRIG10	GPTA[1)]	CHCR00.PRSEL = 1010_B
	INT1SRC	ERAY	CHCR00.PRSEL = 1011_B
	IBUSY	ERAY	CHCR00.PRSEL = 1100_B
	Reserved[2)]	-	CHCR00.PRSEL = 1101_B
	Reserved[2)]	-	CHCR00.PRSEL = 1110_B
	Reserved[2)]	-	CHCR00.PRSEL = 1111_B
01	DMA_SR09	DMA(INT_O09)	CHCR01.PRSEL = 0000_B
	IOUT1	SCU (ERU)	CHCR01.PRSEL = 0001_B
	FADC_SR01	FADC	CHCR01.PRSEL = 0010_B
	ADC_SR01	ADC	CHCR01.PRSEL = 0011_B
	SSC1_RDR	SSC1	CHCR01.PRSEL = 0100_B
	ASC1_RDR	ASC1	CHCR01.PRSEL = 0101_B
	CAN_INT_O1	MultiCAN	CHCR01.PRSEL = 0110_B
	MLI0_SR5	MLI0	CHCR01.PRSEL = 0111_B
	STMIRQ0	STM	CHCR01.PRSEL = 1000_B

Table 11-11 DMA Request Assignment for DMA Sub-Block 0 (cont'd)

DMA Channel	DMA Request Line	DMA Requesting Unit	Selected by
	GPTA_TRIG01	GPTA[1)]	CHCR01.PRSEL = 1001_B
	GPTA_TRIG11	GPTA[1)]	CHCR01.PRSEL = 1010_B
	TINT0SRC	ERAY	CHCR01.PRSEL = 1011_B
	Reserved[2)]	-	CHCR01.PRSEL = 1100_B
	Reserved[2)]	-	CHCR01.PRSEL = 1101_B
	Reserved[2)]	-	CHCR01.PRSEL = 1110_B
	Reserved[2)]	-	CHCR01.PRSEL = 1111_B
02	DMA_SR10	DMA(INT_O10)	CHCR02.PRSEL = 0000_B
	IOUT2	SCU (ERU)	CHCR02.PRSEL = 0001_B
	FADC_SR02	FADC	CHCR02.PRSEL = 0010_B
	ADC_SR02	ADC	CHCR02.PRSEL = 0011_B
	SSC0_TDR	SSC0	CHCR02.PRSEL = 0100_B
	ASC0_TDR	ASC0	CHCR02.PRSEL = 0101_B
	MSC0_SR2	MSC0	CHCR02.PRSEL = 0110_B
	MLI0_SR6	MLI0	CHCR02.PRSEL = 0111_B
	STMIRQ0	STM	CHCR02.PRSEL = 1000_B
	GPTA_TRIG02	GPTA[1)]	CHCR02.PRSEL = 1001_B
	GPTA_TRIG12	GPTA[1)]	CHCR02.PRSEL = 1010_B
	NDAT1SRC	ERAY	CHCR02.PRSEL = 1011_B
	ASC0_TBDR	ASC0	CHCR02.PRSEL = 1100_B
	Reserved[2)]	-	CHCR02.PRSEL = 1101_B
	MSC1_SR2	MSC1	CHCR02.PRSEL = 1110_B
	Reserved[2)]	-	CHCR02.PRSEL = 1111_B
03	DMA_SR11	DMA(INT_O11)	CHCR03.PRSEL = 0000_B
	IOUT3	SCU (ERU)	CHCR03.PRSEL = 0001_B
	FADC_SR03	FADC	CHCR03.PRSEL = 0010_B
	ADC_SR03	ADC	CHCR03.PRSEL = 0011_B
	SSC1_TDR	SSC1	CHCR03.PRSEL = 0100_B
	ASC1_TDR	ASC1	CHCR03.PRSEL = 0101_B
	MSC0_SR3	MSC0	CHCR03.PRSEL = 0110_B

Table 11-11 DMA Request Assignment for DMA Sub-Block 0 (cont'd)

DMA Channel	DMA Request Line	DMA Requesting Unit	Selected by
	MLI0_SR7	MLI0	CHCR03.PRSEL = 0111_B
	STMIRQ0	STM	CHCR03.PRSEL = 1000_B
	GPTA_TRIG03	GPTA[1)]	CHCR03.PRSEL = 1001_B
	GPTA_TRIG13	GPTA[1)]	CHCR03.PRSEL = 1010_B
	MBSC1SRC	ERAY	CHCR03.PRSEL = 1011_B
	ASC1_TBDR	ASC1	CHCR03.PRSEL = 1100_B
	Reserved[2)]	-	CHCR03.PRSEL = 1101_B
	MSC1_SR3	MSC1	CHCR03.PRSEL = 1110_B
	Reserved[2)]	-	CHCR03.PRSEL = 1111_B
04	DMA_SR12	DMA(INT_O12)	CHCR04.PRSEL = 0000_B
	IOUT0	SCU (ERU)	CHCR04.PRSEL = 0001_B
	FADC_SR00	FADC	CHCR04.PRSEL = 0010_B
	ADC_SR04	ADC	CHCR04.PRSEL = 0011_B
	SSC0_TDR	SSC0	CHCR04.PRSEL = 0100_B
	ASC0_TDR	ASC0	CHCR04.PRSEL = 0101_B
	MSC0_SR2	MSC0	CHCR04.PRSEL = 0110_B
	MLI0_SR4	MLI0	CHCR04.PRSEL = 0111_B
	STMIRQ0	STM	CHCR04.PRSEL = 1000_B
	GPTA_TRIG04	GPTA[1)]	CHCR04.PRSEL = 1001_B
	GPTA_TRIG14	GPTA[1)]	CHCR04.PRSEL = 1010_B
	INT1SRC	ERAY	CHCR04.PRSEL = 1011_B
	ASC0_TBDR	ASC0	CHCR04.PRSEL = 1100_B
	OBUSY	ERAY	CHCR04.PRSEL = 1101_B
	MSC1_SR2	MSC1	CHCR04.PRSEL = 1110_B
	Reserved[2)]	-	CHCR04.PRSEL = 1111_B
05	DMA_SR13	DMA(INT_O13)	CHCR05.PRSEL = 0000_B
	IOUT1	SCU (ERU)	CHCR05.PRSEL = 0001_B
	FADC_SR01	FADC	CHCR05.PRSEL = 0010_B
	ADC_SR05	ADC	CHCR05.PRSEL = 0011_B
	SSC1_TDR	SSC1	CHCR05.PRSEL = 0100_B

Table 11-11 DMA Request Assignment for DMA Sub-Block 0 (cont'd)

DMA Channel	DMA Request Line	DMA Requesting Unit	Selected by
	ASC1_TDR	ASC1	CHCR05.PRSEL = 0101_B
	MSC0_SR3	MSC0	CHCR05.PRSEL = 0110_B
	MLI0_SR5	MLI0	CHCR05.PRSEL = 0111_B
	STMIRQ0	STM	CHCR05.PRSEL = 1000_B
	GPTA_TRIG05	GPTA[1)]	CHCR05.PRSEL = 1001_B
	GPTA_TRIG15	GPTA[1)]	CHCR05.PRSEL = 1010_B
	TINT1SRC	ERAY	CHCR05.PRSEL = 1011_B
	ASC1_TBDR	ASC1	CHCR05.PRSEL = 1100_B
	Reserved[2)]	-	CHCR05.PRSEL = 1101_B
	MSC1_SR3	MSC1	CHCR05.PRSEL = 1110_B
	Reserved[2)]	-	CHCR05.PRSEL = 1111_B
06	DMA_SR14	DMA(INT_O14)	CHCR06.PRSEL = 0000_B
	IOUT2	SCU (ERU)	CHCR06.PRSEL = 0001_B
	FADC_SR02	FADC	CHCR06.PRSEL = 0010_B
	ADC_SR06	ADC	CHCR06.PRSEL = 0011_B
	SSC0_RDR	SSC0	CHCR06.PRSEL = 0100_B
	ASC0_RDR	ASC0	CHCR06.PRSEL = 0101_B
	CAN_INT_O0	MultiCAN	CHCR06.PRSEL = 0110_B
	MLI0_SR6	MLI0	CHCR06.PRSEL = 0111_B
	STMIRQ0	STM	CHCR06.PRSEL = 1000_B
	GPTA_TRIG06	GPTA[1)]	CHCR06.PRSEL = 1001_B
	GPTA_TRIG16	GPTA[1)]	CHCR06.PRSEL = 1010_B
	NDAT1SRC	ERAY	CHCR06.PRSEL = 1011_B
	Reserved[2)]	-	CHCR06.PRSEL = 1100_B
	Reserved[2)]	-	CHCR06.PRSEL = 1101_B
	ADC_SR08	ADC	CHCR06.PRSEL = 1110_B
	Reserved[2)]	-	CHCR06.PRSEL = 1111_B
07	DMA_SR15	DMA(INT_O15)	CHCR07.PRSEL = 0000_B
	IOUT3	SCU (ERU)	CHCR07.PRSEL = 0001_B
	FADC_SR03	FADC	CHCR07.PRSEL = 0010_B

Table 11-11 DMA Request Assignment for DMA Sub-Block 0 (cont'd)

DMA Channel	DMA Request Line	DMA Requesting Unit	Selected by
	ADC_SR07	ADC	CHCR07.PRSEL = 0011_B
	SSC1_RDR	SSC1	CHCR07.PRSEL = 0100_B
	ASC1_RDR	ASC1	CHCR07.PRSEL = 0101_B
	CAN_INT_O1	MultiCAN	CHCR07.PRSEL = 0110_B
	MLI0_SR7	MLI0	CHCR07.PRSEL = 0111_B
	STMIRQ0	STM	CHCR07.PRSEL = 1000_B
	GPTA_TRIG07	GPTA[1)]	CHCR07.PRSEL = 1001_B
	GPTA_TRIG17	GPTA[1)]	CHCR07.PRSEL = 1010_B
	MBSC1SRC	ERAY	CHCR07.PRSEL = 1011_B
	Reserved[2)]	-	CHCR07.PRSEL = 1100_B
	Reserved[2)]	-	CHCR07.PRSEL = 1101_B
	ADC_SR09	ADC	CHCR07.PRSEL = 1110_B
	Reserved[2)]	-	CHCR07.PRSEL = 1111_B

1) GPTA_TRIG signals are per default level sensitive signals while a DMA channel is activated with every active request signal cycle. The DMA internal positive edge detection will generate the channel request with the rising edge of the GPTA_TRIG signal, if selected. If channel requests for the positive and/or negative GPTA_TRIG signal is required, this can be realized either via the ERU (some GPTA_TRIG signals are mapped to it) or via GPTA programming by using additional GPTA cells.

2) Reserved PRSEL combinations do not result to DMA Channel requests.

Table 11-12 DMA Request Assignment for DMA Sub-Block 1

DMA Channel	DMA Request Line	DMA Requesting Unit	Selected by
10	DMA_SR08	DMA(INT_O08)	CHCR00.PRSEL = 0000_B
	IOUT0	SCU (ERU)	CHCR00.PRSEL = 0001_B
	FADC_SR00	FADC	CHCR00.PRSEL = 0010_B
	ADC_SR00	ADC	CHCR00.PRSEL = 0011_B
	SSC0_RDR	SSC0	CHCR00.PRSEL = 0100_B
	ASC0_RDR	ASC0	CHCR00.PRSEL = 0101_B
	CAN_INT_O0	MultiCAN	CHCR00.PRSEL = 0110_B
	MLI0_SR4	MLI0	CHCR00.PRSEL = 0111_B
	STMIRQ0	STM	CHCR00.PRSEL = 1000_B

Table 11-12 DMA Request Assignment for DMA Sub-Block 1 (cont'd)

DMA Channel	DMA Request Line	DMA Requesting Unit	Selected by
	GPTA_TRIG00	GPTA[1)]	CHCR00.PRSEL = 1001_B
	GPTA_TRIG10	GPTA[1)]	CHCR00.PRSEL = 1010_B
	INT1SRC	ERAY	CHCR00.PRSEL = 1011_B
	IBUSY	ERAY	CHCR00.PRSEL = 1100_B
	Reserved[2)]	-	CHCR00.PRSEL = 1101_B
	Reserved[2)]	-	CHCR00.PRSEL = 1110_B
	Reserved[2)]	-	CHCR00.PRSEL = 1111_B
11	DMA_SR09	DMA(INT_O09)	CHCR01.PRSEL = 0000_B
	IOUT1	SCU (ERU)	CHCR01.PRSEL = 0001_B
	FADC_SR01	FADC	CHCR01.PRSEL = 0010_B
	ADC_SR01	ADC	CHCR01.PRSEL = 0011_B
	SSC1_RDR	SSC1	CHCR01.PRSEL = 0100_B
	ASC1_RDR	ASC1	CHCR01.PRSEL = 0101_B
	CAN_INT_O1	MultiCAN	CHCR01.PRSEL = 0110_B
	MLI0_SR5	MLI0	CHCR01.PRSEL = 0111_B
	STMIRQ0	STM	CHCR01.PRSEL = 1000_B
	GPTA_TRIG01	GPTA[1)]	CHCR01.PRSEL = 1001_B
	GPTA_TRIG11	GPTA[1)]	CHCR01.PRSEL = 1010_B
	TINT0SRC	ERAY	CHCR01.PRSEL = 1011_B
	Reserved[2)]	-	CHCR01.PRSEL = 1100_B
	Reserved[2)]	-	CHCR01.PRSEL = 1101_B
	Reserved[2)]	-	CHCR01.PRSEL = 1110_B
	Reserved[2)]	-	CHCR01.PRSEL = 1111_B
12	DMA_SR10	DMA(INT_O10)	CHCR02.PRSEL = 0000_B
	IOUT2	SCU (ERU)	CHCR02.PRSEL = 0001_B
	FADC_SR02	FADC	CHCR02.PRSEL = 0010_B
	ADC_SR02	ADC	CHCR02.PRSEL = 0011_B
	SSC0_TDR	SSC0	CHCR02.PRSEL = 0100_B
	ASC0_TDR	ASC0	CHCR02.PRSEL = 0101_B
	MSC0_SR2	MSC0	CHCR02.PRSEL = 0110_B

Table 11-12 DMA Request Assignment for DMA Sub-Block 1 (cont'd)

DMA Channel	DMA Request Line	DMA Requesting Unit	Selected by
	MLI0_SR6	MLI0	CHCR02.PRSEL = 0111_B
	STMIRQ0	STM	CHCR02.PRSEL = 1000_B
	GPTA_TRIG02	GPTA[1)]	CHCR02.PRSEL = 1001_B
	GPTA_TRIG12	GPTA[1)]	CHCR02.PRSEL = 1010_B
	NDAT1SRC	ERAY	CHCR02.PRSEL = 1011_B
	ASC0_TBDR	ASC0	CHCR02.PRSEL = 1100_B
	Reserved[2)]	-	CHCR02.PRSEL = 1101_B
	MSC1_SR2	MSC1	CHCR02.PRSEL = 1110_B
	Reserved[2)]	-	CHCR02.PRSEL = 1111_B
13	DMA_SR11	DMA(INT_O11)	CHCR03.PRSEL = 0000_B
	IOUT3	SCU (ERU)	CHCR03.PRSEL = 0001_B
	FADC_SR03	FADC	CHCR03.PRSEL = 0010_B
	ADC_SR03	ADC	CHCR03.PRSEL = 0011_B
	SSC1_TDR	SSC1	CHCR03.PRSEL = 0100_B
	ASC1_TDR	ASC1	CHCR03.PRSEL = 0101_B
	MSC0_SR3	MSC0	CHCR03.PRSEL = 0110_B
	MLI0_SR7	MLI0	CHCR03.PRSEL = 0111_B
	STMIRQ0	STM	CHCR03.PRSEL = 1000_B
	GPTA_TRIG03	GPTA[1)]	CHCR03.PRSEL = 1001_B
	GPTA_TRIG13	GPTA[1)]	CHCR03.PRSEL = 1010_B
	MBSC1SRC	ERAY	CHCR03.PRSEL = 1011_B
	ASC1_TBDR	ASC1	CHCR03.PRSEL = 1100_B
	Reserved[2)]	-	CHCR03.PRSEL = 1101_B
	MSC1_SR3	MSC1	CHCR03.PRSEL = 1110_B
	Reserved[2)]	-	CHCR03.PRSEL = 1111_B
14	DMA_SR12	DMA(INT_O12)	CHCR04.PRSEL = 0000_B
	IOUT0	SCU (ERU)	CHCR04.PRSEL = 0001_B
	FADC_SR00	FADC	CHCR04.PRSEL = 0010_B
	ADC_SR04	ADC	CHCR04.PRSEL = 0011_B
	SSC0_TDR	SSC0	CHCR04.PRSEL = 0100_B

Table 11-12 DMA Request Assignment for DMA Sub-Block 1 (cont'd)

DMA Channel	DMA Request Line	DMA Requesting Unit	Selected by
	ASC0_TDR	ASC0	CHCR04.PRSEL = 0101_B
	MSC0_SR2	MSC0	CHCR04.PRSEL = 0110_B
	MLI0_SR4	MLI0	CHCR04.PRSEL = 0111_B
	STMIRQ0	STM	CHCR04.PRSEL = 1000_B
	GPTA_TRIG04	GPTA[1)]	CHCR04.PRSEL = 1001_B
	GPTA_TRIG14	GPTA[1)]	CHCR04.PRSEL = 1010_B
	INT1SRC	ERAY	CHCR04.PRSEL = 1011_B
	ASC0_TBDR	ASC0	CHCR04.PRSEL = 1100_B
	OBUSY	ERAY	CHCR04.PRSEL = 1101_B
	MSC1_SR2	MSC1	CHCR04.PRSEL = 1110_B
	Reserved[2)]	-	CHCR04.PRSEL = 1111_B
15	DMA_SR13	DMA(INT_O13)	CHCR05.PRSEL = 0000_B
	IOUT1	SCU (ERU)	CHCR05.PRSEL = 0001_B
	FADC_SR01	FADC	CHCR05.PRSEL = 0010_B
	ADC_SR05	ADC	CHCR05.PRSEL = 0011_B
	SSC1_TDR	SSC1	CHCR05.PRSEL = 0100_B
	ASC1_TDR	ASC1	CHCR05.PRSEL = 0101_B
	MSC0_SR3	MSC0	CHCR05.PRSEL = 0110_B
	MLI0_SR5	MLI0	CHCR05.PRSEL = 0111_B
	STMIRQ0	STM	CHCR05.PRSEL = 1000_B
	GPTA_TRIG05	GPTA[1)]	CHCR05.PRSEL = 1001_B
	GPTA_TRIG15	GPTA[1)]	CHCR05.PRSEL = 1010_B
	TINT1SRC	ERAY	CHCR05.PRSEL = 1011_B
	ASC1_TBDR	ASC1	CHCR05.PRSEL = 1100_B
	Reserved[2)]	-	CHCR05.PRSEL = 1101_B
	MSC1_SR3	MSC1	CHCR05.PRSEL = 1110_B
	Reserved[2)]	-	CHCR05.PRSEL = 1111_B
16	DMA_SR14	DMA(INT_O14)	CHCR06.PRSEL = 0000_B
	IOUT2	SCU (ERU)	CHCR06.PRSEL = 0001_B
	FADC_SR02	FADC	CHCR06.PRSEL = 0010_B

Table 11-12 DMA Request Assignment for DMA Sub-Block 1 (cont'd)

DMA Channel	DMA Request Line	DMA Requesting Unit	Selected by
	ADC_SR06	ADC	CHCR06.PRSEL = 0011_B
	SSC0_RDR	SSC0	CHCR06.PRSEL = 0100_B
	ASC0_RDR	ASC0	CHCR06.PRSEL = 0101_B
	CAN_INT_O0	MultiCAN	CHCR06.PRSEL = 0110_B
	MLI0_SR6	MLI0	CHCR06.PRSEL = 0111_B
	STMIRQ0	STM	CHCR06.PRSEL = 1000_B
	GPTA_TRIG06	GPTA[1)]	CHCR06.PRSEL = 1001_B
	GPTA_TRIG16	GPTA[1)]	CHCR06.PRSEL = 1010_B
	NDAT1SRC	ERAY	CHCR06.PRSEL = 1011_B
	Reserved[2)]	-	CHCR06.PRSEL = 1100_B
	Reserved[2)]	-	CHCR06.PRSEL = 1101_B
	ADC_SR08	ADC	CHCR06.PRSEL = 1110_B
	Reserved[2)]	-	CHCR06.PRSEL = 1111_B
17	DMA_SR15	DMA(INT_O15)	CHCR07.PRSEL = 0000_B
	IOUT3	SCU (ERU)	CHCR07.PRSEL = 0001_B
	FADC_SR03	FADC	CHCR07.PRSEL = 0010_B
	ADC_SR07	ADC	CHCR07.PRSEL = 0011_B
	SSC1_RDR	SSC1	CHCR07.PRSEL = 0100_B
	ASC1_RDR	ASC1	CHCR07.PRSEL = 0101_B
	CAN_INT_O1	MultiCAN	CHCR07.PRSEL = 0110_B
	MLI0_SR7	MLI0	CHCR07.PRSEL = 0111_B
	STMIRQ0	STM	CHCR07.PRSEL = 1000_B
	GPTA_TRIG07	GPTA[1)]	CHCR07.PRSEL = 1001_B
	GPTA_TRIG17	GPTA[1)]	CHCR07.PRSEL = 1010_B
	MBSC1SRC	ERAY	CHCR07.PRSEL = 1011_B
	Reserved[2)]	-	CHCR07.PRSEL = 1100_B
	Reserved[2)]	-	CHCR07.PRSEL = 1101_B
	ADC_SR09	ADC	CHCR07.PRSEL = 1110_B
	Reserved[2)]	-	CHCR07.PRSEL = 1111_B

1) GPTA_TRIG signals are per default level sensitive signals while a DMA channel is activated with every active request signal cycle. The DMA internal positive edge detection will generate the channel request with the rising edge of the GPTA_TRIG signal, if selected. If channel requests for the positive and/or negative GPTA_TRIG signal is required, this can be realized either via the ERU (some GPTA_TRIG signals are mapped to it) or via GPTA programming by using additional GPTA cells.
2) Reserved PRSEL combinations do not result to DMA Channel requests.

11.4.2 Access Protection Assignment

DMA access protection as described on **Page 11-45** requires the assignment of 32 fixed address range. **Table 11-13** shows this address range assignment as implemented in the TC1797 (see also: **Page 11-82**, **Page 11-82**).

Table 11-13 DMA Access Protection Address Ranges

Access Protection Range			Related Module(s)
No. n	**Enable Bit in MEmAENR**	**Selected Address Range**	
0	AEN0	$F000\ 0500_H$ - $F000\ 06FF_H$ $F010\ C200_H$ - $F010\ C2FF_H$	SCU MEMCHK
1	AEN1	$F000\ 0100_H$ - $F000\ 01FF_H$	SBCU
2	AEN2	$F000\ 0200_H$ - $F000\ 02FF_H$	STM
3	AEN3	$F000\ 0400_H$ - $F000\ 04FF_H$	OCDS
4	AEN4	$F000\ 0800_H$ to $F000\ 09FF_H$	MSC0 MSC1
5	AEN5	$F000\ 0A00_H$ to $F000\ 0AFF_H$	ASC0
6	AEN6	$F000\ 0B00_H$ to $F000\ 0BFF_H$	ASC1
7	AEN7	$F000\ 0C00_H$ - $F000\ 17FF_H$	Port 0 - Port 5, Port 6, Port7, Port8, Port9, Port10, Port11
8	AEN8	$F030\ 0000_H$ - $F030\ 04FF_H$	-Port 12 - Port 16
9	AEN9	$F000\ 1800_H$ - $F000\ 37FF_H$	GPTA (GPTA0, GPTA1, LTCA2)
10	AEN10	$F000\ 3C00_H$ - $F000\ 3EFF_H$	DMA
11	AEN11	$F000\ 4000_H$ - $F000\ 7FFF_H$	MultiCAN
12	AEN12	$F004\ 0000_H$ - $F004\ FFFF_H$	PCP Registers
13	AEN13	$F005\ 0000_H$ - $F005\ FFFF_H$	PCP Data Memory (PRAM)
14	AEN14	$F006\ 0000_H$ - $F007\ FFFF_H$	PCP CodeMemory (CMEM)
15	AEN15	$F010\ 0100_H$ - $F010\ 01FF_H$	SSC0
16	AEN16	$F010\ 0200_H$ - $F010\ 02FF_H$	SSC1
17	AEN17	$F010\ 0400_H$ - $F010\ 05FF_H$	FADC
18	AEN18	$F010\ 1000_H$ - $F010\ 1BFF_H$	ADC0, ADC1, ADC2
19	AEN19	$F010\ C000_H$ - $F010\ C0FF_H$ $F01E\ 0000_H$ - $F01E\ 7FFF_H$ $F020\ 0000_H$ - $F023\ FFFF_H$	MLI0 Module, MLI0 Small TWs, MLI0 Large TWs

Table 11-13 DMA Access Protection Address Ranges (cont'd)

Access Protection Range			Related Module(s)
No. n	**Enable Bit in MEmAENR**	**Selected Address Range**	
20	AEN20	F010 C100$_H$ - F010 C1FF$_H$ F01E 8000$_H$ - F01E FFFF$_H$ F024 0000$_H$ - F027 FFFF$_H$	-
21	AEN21	F7E0 FF00$_H$ - F7E0 FFFF$_H$ F7E1 0000$_H$ - F7E1 FFFF$_H$ F800 0400$_H$ - F87F FFFF$_H$	CPS, CPU SFRs & GPRs, PMU, Flash Regs, LBCU, DMI, PMI, LFI
22	AEN22	F800 0000$_H$ - F800 03FF	EBU
23	AEN23	8000 0000$_H$ - 807F FFFF$_H$ A000 0000$_H$ - A07F FFFF$_H$	Program Flash Space
24	AEN24	8080 0000$_H$ - 8FDF FFFF$_H$ A080 0000$_H$ - AFDF FFFF$_H$	Ext. EBU Space
25	AEN25	8FE0 0000$_H$ - 8FE1 FFFF$_H$ AFE0 0000$_H$ - AFE1 FFFF$_H$	Data Flash Space
26	AEN26	8FF0 0000$_H$ - 8FFF BFFF$_H$ AFF0 0000$_H$ - AFFF BFFF$_H$	Emulation Device Memory Space
27	AEN27	8FFF C000$_H$ - 8FFF FFFF$_H$ AFFF C000$_H$ - AFFF FFFF$_H$	Boot ROM
28	AEN28	F001 0000$_H$ - F001 7FFF$_H$	ERAY
29	AEN29	8FE8 0000$_H$ - 8FE8 1FFF$_H$ AFE8 0000$_H$ - AFE8 1FFF$_H$	OVRAM
30	AEN30	D000 0000$_H$ - D001 EFFF$_H$ E840 0000$_H$ - E84F FFFF$_H$	DMI DMI Image (E84x translated to D00x)
31	AEN31	C000 0000$_H$ - C400 9FFF$_H$ D400 0000$_H$ - D400 9FFFF$_H$ E850 0000$_H$ - E85F FFFF$_H$ E800 0000$_H$ - E83F FFFF$_H$	PMI PMI Image (E85x translated to D40x)

In the TC1797, four internal memory areas (SPRAM and LDRAM, PRAM, and OVRAM) are protected by an address range verification in addition to the access enable bits. The address range verification is based on the bit fields SIZEx and SLIZEx (x = 3-0), which are located in the registers MEmARR (m = 1-0). An access to one of these four memory areas is only processed if it is enabled by the corresponding AENx bit and if the address is inside the sub-range defined by the corresponding SIZEx and SLIZEx. If the address is outside of the defined sub-range, the transfer will not be processed and an error interrupt is generated (indicated by the corresponding MExDER, MExSER bit). If a protected memory is available from DMA under multiple address ranges (LMB, FPI, cached and or un-cached segments), the access protection is valid for all memory views in parallel, starting always with the base address of the memory views, ending always with the end address of the address range.

The address ranges described by SLIZEx and SIZEx are defined as follows:

- MEmARR.SLICE0, MEmARR.SIZE0:
  40-KB PMI RAM (SPRAM), assigned to address range 31 (AEN31). The sub-ranges are controlled by bit fields MEmARR.SIZE0 and ME0ARR.SLICE0 with a minimum granularity of 0,5 KB (see **Table 11-14**).
- MEmARR.SIZE1 and MEmARR.SLICE1:
  8-KB OVRAM, assigned to address range 29 (AEN29). The sub-ranges are controlled by bit fields MEmARR.SIZE1 and MEmARR.SLICE1 with a minimum granularity of 0.5 KB (see **Table 11-15**).
- MEmARR.SIZE2 and MEmARR.SLICE2:
  128-KB DMI RAM (LDRAM), assigned to address range 30 (AEN30). The sub-ranges are controlled by bit fields MEmARR.SIZE2 and MEmARR.SLICE2 with a minimum granularity of 1 KB (see **Table 11-16**).
- MEmARR.SIZE3 and MEmARR.SLICE3:
  16-Kbyte PCP PRAM, assigned to address range 13 (AEN13). The sub-range is controlled by bit fields ME0mARR.SIZE3 and ME0mARR.SLICE3 with a minimum granularity of 0.5 KB (see **Table 11-17**).

SIZE0 and SLICE0 bit fields: PMI sub-range access protection

Bit fields SIZE0 and SLICE0 for the PMI memory sub-range access protection (40 KB SPRAM) as shown in **Table 11-14**.

The PMI memory is protected with a min. granularity of 0.5 KB up to the end address xxx0 $9FFF_H$[1)].

PMI address ranges that are protected by the bit fields SIZE0 and SLICE0:

- $D400\ 0000_H$ - $D400\ 9FFF_H$ (PMI SPRAM on LMB)
- $E850\ 0000_H$ - $E850\ 9FFF_H$ (LFI translation from E850 to C000)
- $C000\ 0000_H$ - $C000\ 9FFF_H$ (PMI SPRAM on LMB)
- $E800\ 0000_H$ - $E800\ 9FFF_H$ (LFI translation from E800 to D400)

Table 11-14 PMI Address Protection Sub-Range Definition

SIZE0	Sub-Ranges	SLICE0	Selected Address Range[1)]
000_B	32 sub-ranges of 512 bytes	00000_B 00001_B … 11111_B	xxx0 0000_H - xxx0 $01FF_H$ xxx0 0200_H - xxx0 $03FF_H$ … xxx0 $3E00_H$ - xxx0 $3FFF_H$
001_B	32 sub-ranges of 1 Kbyte	00000_B 00001_B … 11111_B	xxx0 0000_H - xxx0 $03FF_H$ xxx0 0400_H - xxx0 $07FF_H$ … xxx0 $7C00_H$ - xxx0 $7FFF_H$
010_B	32 sub-ranges of 2 Kbytes	00000_B 00001_B … 11111_B	xxx0 0000_H - xxx0 $07FF_H$ xxx0 0800_H - xxx0 $0FFF_H$ … xxx0 $F800_H$ - xxx0 $FFFF_H$
011_B	16 sub-ranges of 4 Kbytes	$X0000_B$ $X0001_B$ … $X1111_B$	xxx0 0000_H - xxx0 $0FFF_H$ xxx0 1000_H - xxx0 $1FFF_H$ … xxx0 $F000_H$ - xxx0 $FFFF_H$
100_B	8 sub-ranges of 8 Kbytes	$XX000_B$ $XX001_B$ … $XX111_B$	xxx0 0000_H - xxx0 $1FFF_H$ xxx0 2000_H - xxx0 $3FFF_H$ … xxx0 $E000_H$ - xxx0 $FFFF_H$
101_B	4 sub-ranges of 16 Kbytes	$XXX00_B$ $XXX01_B$ $XXX10_B$ $XXX11_B$	xxx0 0000_H - xxx0 $3FFF_H$ xxx0 4000_H - xxx0 $7FFF_H$ xxx0 8000_H - xxx0 $BFFF_H$ xxx0 $C000_H$ - xxx0 $FFFF_H$

1) xxx in **Table 11-14**, column ´Selected Address Range´ is the place holder for $D40_H$, $E85_H$, $C00_H$, $E80_H$,.

Table 11-14 PMI Address Protection Sub-Range Definition (cont'd)

SIZE0	Sub-Ranges	SLICE0	Selected Address Range[1)]
110_B	2 sub-ranges of 32 Kbytes	$XXXX0_B$ $XXXX1_B$	xxxx 0000_H - xxxx $7FFF_H$ xxxx 8000_H - xxxx $FFFF_H$
111_B	64 Kbytes	$XXXXX_B$	xxxx 0000_H - xxxx $FFFF_H$

SIZE1 and SLICE1 bit fields: OVRAM sub-range access protection

Bit fields SIZE1 and SLICE1 for the OVRAM sub-range access protection (with address translation from E800 to C000 in the LFI) are coded as shown in **Table 11-15**.

The 8 KB OVRAM memory is protected with a min. granularity of 0.5 KB up to it´s end address xFE8 $1FFF_H$[1)].

OVRAM address ranges that are protected by the bit fields SIZE1 and SLICE1:

- 8FE8 0000_H - 8FE8 $1FFF_H$ (OVRAM on LMB, cached segment)
- AFE8 0000_H - AFE8 $1FFF_H$ (OVRAM on LMB, un-cached segment)

Table 11-15 OVRAM Address Protection Sub-Range Definition

SIZE1	Sub-Ranges	SLICE1	Selected Address Range[1)]
000_B	32 sub-ranges of 512 bytes	00000_B 00001_B … 11111_B	xFE0 0000_H - xFE0 $01FF_H$ xFE0 0200_H - xFE0 $03FF_H$ … xFE0 $3E00_H$ - xFE0 $3FFF_H$
001_B	32 sub-ranges of 1 Kbyte	00000_B 00001_B … 11111_B	xFE0 0000_H - xFE0 $03FF_H$ xFE0 0400_H - xFE0 $07FF_H$ … xFE0 $7C00_H$ - xFE0 $7FFF_H$
010_B	32 sub-ranges of 2 Kbytes	00000_B 00001_B … 11111_B	xFE0 0000_H - xFE0 $07FF_H$ xFE0 0800_H - xFE0 $0FFF_H$ … xFE0 $F800_H$ - xFE0 $FFFF_H$
011_B	16 sub-ranges of 4 Kbytes	$X0000_B$ $X0001_B$ … $X1111_B$	xFE0 0000_H - xFE0 $0FFF_H$ xFE0 1000_H - xFE0 $1FFF_H$ … xFE0 $F000_H$ - xFE0 $FFFF_H$

1) x in **Table 11-15**, column ´Selected Address Range´ is the place holder for D_H, and A_H.

Table 11-15 OVRAM Address Protection Sub-Range Definition (cont'd)

SIZE1	Sub-Ranges	SLICE1	Selected Address Range[1)]
100_B	8 sub-ranges of 8 Kbytes	$XX000_B$ $XX001_B$ … $XX111_B$	xxx0 0000_H - xxx0 $1FFF_H$ xxx0 2000_H - xxx0 $3FFF_H$ … xxx0 $E000_H$ - xxx0 $FFFF_H$
101_B	4 sub-ranges of 16 Kbytes	$XXX00_B$ $XXX01_B$ $XXX10_B$ $XXX11_B$	xxx0 0000_H - xxx0 $3FFF_H$ xxx0 4000_H - xxx0 $7FFF_H$ xxx0 8000_H - xxx0 $BFFF_H$ xxx0 $C000_H$ - xxx0 $FFFF_H$
110_B	2 sub-ranges of 32 Kbytes	$XXXX0_B$ $XXXX1_B$	xxxx 0000_H - xxxx $7FFF_H$ xxxx 8000_H - xxxx $FFFF_H$
111_B	64 Kbytes	$XXXXX_B$	xxxx 0000_H - xxxx $FFFF_H$

SIZE2 and SLICE2 bit fields: DMI sub-range access protection

Bit fields SIZE2 and SLICE2 for the DMI RAM sub-range access protection. These bit fields are covering the 128 KB DMI memory (LDRAM).

The DMI memory is protected with a min. granularity of 1 KB (**Table 11-16**) up to it´s end address D001 D001 $FFFF_H$ and E841 $FFFF_H$.

DMI address ranges that are protected by the bit fields SIZE2 and SLICE2:

- D000 0000_H - D001 $FFFF_H$ (DMI LDRAM on LMB)
- E840 0000_H - E841 $FFFF_H$ (LFI translation from E840 to D000)

Table 11-16 DMI Address Protection Sub-Range Defintions

SIZE2	Sub-Ranges	SLICE2	Selected Address Range[1)]
000_B	32 sub-ranges of 1 Kbytes	00000_B 00001_B … 11111_B	xxx0 0000_H - xxx0 $03FF_H$ xxx0 0400_H - xxx0 $07FF_H$ … xxx0 $7C00_H$ - xxx0 $7FFF_H$
001_B	32 sub-ranges of 2 Kbyte	00000_B 00001_B … 11111_B	xxx0 0000_H - xxx0 $07FF_H$ xxx0 0800_H - xxx0 $0FFF_H$ … xxx0 $F800_H$ - xxx0 $FFFF_H$
010_B	32 sub-ranges of 4Kbytes	00000_B 00001_B … 11111_B	xxx0 0000_H - xxx0 $0FFF_H$ xxx0 1000_H - xxx0 $1FFF_H$ … xxx1 $F000_H$ - xxx1 $FFFF_H$

Table 11-16 DMI Address Protection Sub-Range Defintions (cont'd)

SIZE2	Sub-Ranges	SLICE2	Selected Address Range[1)]
011_B	16 sub-ranges of 8 Kbytes	$X0000_B$ $X0001_B$ … $X1111_B$	$xxx0\ 0000_H$ - $xxx0\ 1FFF_H$ $xxx0\ 2000_H$ - $xxx0\ 3FFF_H$ … $xxx1\ E000_H$ - $xxx1\ FFFF_H$
100_B	8 sub-ranges of 16 Kbytes	$XX000_B$ $XX001_B$ … $XX111_B$	$xxx0\ 0000_H$ - $xxx0\ 3FFF_H$ $xxx0\ 4000_H$ - $xxx0\ 7FFF_H$ .... $xxx1\ C000_H$ - $xxx1\ FFFF_H$
101_B	4 sub-ranges of 32 Kbytes	$XXX00_B$ $XXX01_B$ $XXX10_B$ $XXX11_B$	$xxx0\ 0000_H$ - $xxx0\ 7FFF_H$ $xxx0\ 8000_H$ - $xxx0\ FFFF_H$ $xxx1\ 0000_H$ - $xxx1\ 7FFF_H$ $xxx1\ 8000_H$ - $xxx1\ FFFF_H$
110_B	2 sub-ranges of 64 Kbytes	$XXXX0_B$ $XXXX1_B$	$xxx0\ 0000_H$ - $xxx0\ FFFF_H$ $xxx1\ 0000_H$ - $xxx1\ FFFF_H$
111_B	128 Kbytes	$XXXXX_B$	$xxx0\ 0000_H$ - $xxx1\ FFFF_H$

1) xxx in **Table 11-16**, column ´Selected Address Range´ is the place holder of $D00_H$ and $E84_H$

SIZE3 and SLICE3 bit fields: PCP PRAM sub-range access protection

Bit fields SIZE3 and SLICE3 for the PCP Parameter RAM (PRAM) sub-range access protection. These bit fields are covering PCP PRAM memory.

The PCP PRAM memory is protected with a min. granularity of 0.5 KB (**Table 11-17**) up to it´s end address $F005\ 1FFF_H$.

PCP PRAM address range that is protected by the bit fields SIZE3 and SLICE3:

- $F005\ 0000_H$ - $F005\ 3FFF_H$ (PCP PRAM on FPI Bus, 16 KB)

Table 11-17 PCP PRAM Address Protection Sub-Range DefintionScheme

SIZE3	Sub-Ranges	SLICE3	Selected Address Range
000_B	32 sub-ranges of 512 bytes	00000_B 00001_B … 11111_B	$F005\ 0000_H$ - $F005\ 01FF_H$ $F005\ 0200_H$ - $F005\ 03FF_H$ … $F005\ 3E00_H$ - $F005\ 3FFF_H$
001_B	32 sub-ranges of 1 Kbyte	00000_B 00001_B … 11111_B	$F005\ 0000_H$ - $F005\ 03FF_H$ $F005\ 0400_H$ - $F005\ 07FF_H$ … $F005\ 7C00_H$ - $F005\ 7FFF_H$

Table 11-17 PCP PRAM Address Protection Sub-Range DefintionScheme

SIZE3	Sub-Ranges	SLICE3	Selected Address Range
010_B	32 sub-ranges of 2 Kbytes	00000_B 00001_B … 11111_B	$F005\ 0000_H$ - $F005\ 07FF_H$ $F005\ 0800_H$ - $F005\ 0FFF_H$ … $F005\ F800_H$ - $F005\ FFFF_H$
011_B	16 sub-ranges of 4 Kbytes	$X0000_B$ $X0001_B$ … $X1111_B$	$F005\ 0000_H$ - $F005\ 0FFF_H$ $F005\ 1000_H$ - $F005\ 1FFF_H$ … $F005\ F000_H$ - $F005\ FFFF_H$
100_B	8 sub-ranges of 8 Kbytes	$XX000_B$ $XX001_B$ … $XX111_B$	$F005\ 0000_H$ - $F005\ 1FFF_H$ $F005\ 2000_H$ - $F005\ 3FFF_H$ … $F005\ E000_H$ - $F005\ FFFF_H$
101_B	4 sub-ranges of 16 Kbytes	$XXX00_B$ $XXX01_B$ $XXX10_B$ $XXX11_B$	$F005\ 0000_H$ - $F005\ 3FFF_H$ $F005\ 4000_H$ - $F005\ 7FFF_H$ $F005\ 8000_H$ - $F005\ BFFF_H$ $F005\ C000_H$ - $F005\ FFFF_H$
110_B	2 sub-ranges of 32 Kbytes	$XXXX0_B$ $XXXX1_B$	$F005\ 0000_H$ - $F005\ 7FFF_H$ $F005\ 8000_H$ - $F005\ FFFF_H$
111_B	64 Kbytes	$XXXXX_B$	$F005\ 0000_H$ - $F005\ FFFF_H$

11.4.3 Implementation-specific DMA Registers

The DMA controller as implemented in the TC1797 contains the following additional registers:

- DMA clock control register
- Service request control registers for DMA controller interrupts (DMA_SRCx)
- Service request control registers for MLI module interrupts (DMA_MLI0ySRC.x)

Figure 11-29 provides an overview of these registers.

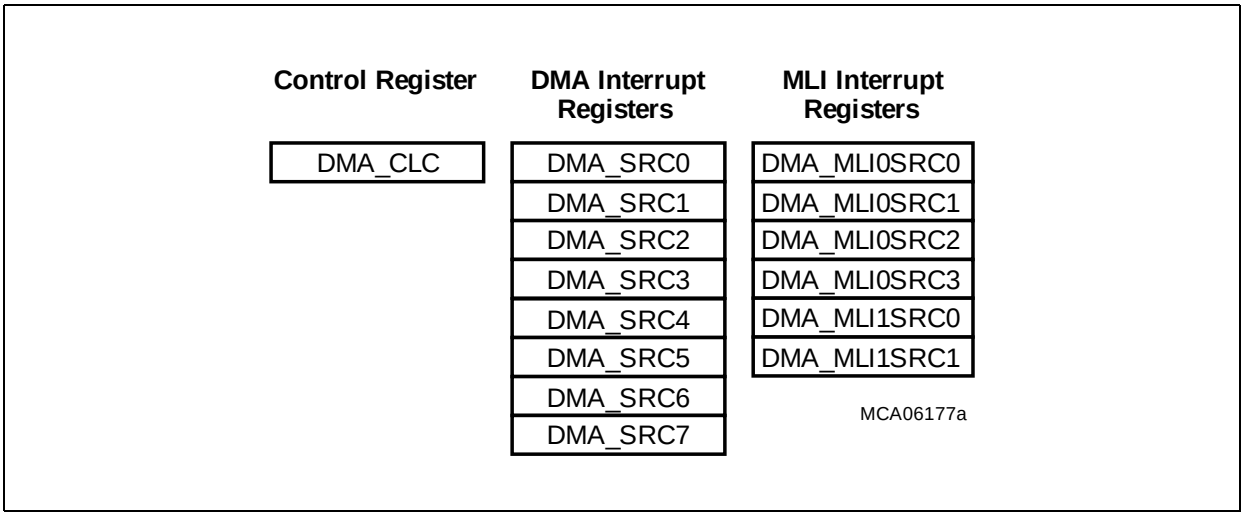


Figure 11-29 DMA Implementation-specific Registers

Note: Further details on interrupt handling and processing are described in the "Interrupt System" chapter of the TC1797 System Units User's Manual.

The clock generation and interrupt control configuration as implemented in the DMA controller module is shown in **Figure 11-29**.

The DMA controller, the Cerberus and the two MLI modules (MLI0 and MLI1) are supplied from a common module clock f_{DMA} that has the frequency of the system clock f_{FPI} and is controlled via the DMA_CLC clock control register. The MLI modules nor the Cerberus module do not have their own clock control registers. Their input clock is derived from the DMA clock divided by separate fractional divider registers.

The control of the suspend and break features is done independently inside each module.

The DMA controller module contains in total 14 interrupt request nodes with its interrupt service request control registers:

- Eight interrupt requests SR[7:0] = INT_O[7:0] from the DMA controller; upper eight interrupt requests of the DMA controller INT_O[15:8] are used ad DMA channel trigger inputs.
- Four interrupt requests SR[3:0] = INT_O[3:0] from the MLI0 module; upper four interrupt requests of the MLI0 module INT_O[7:4] are not connected.

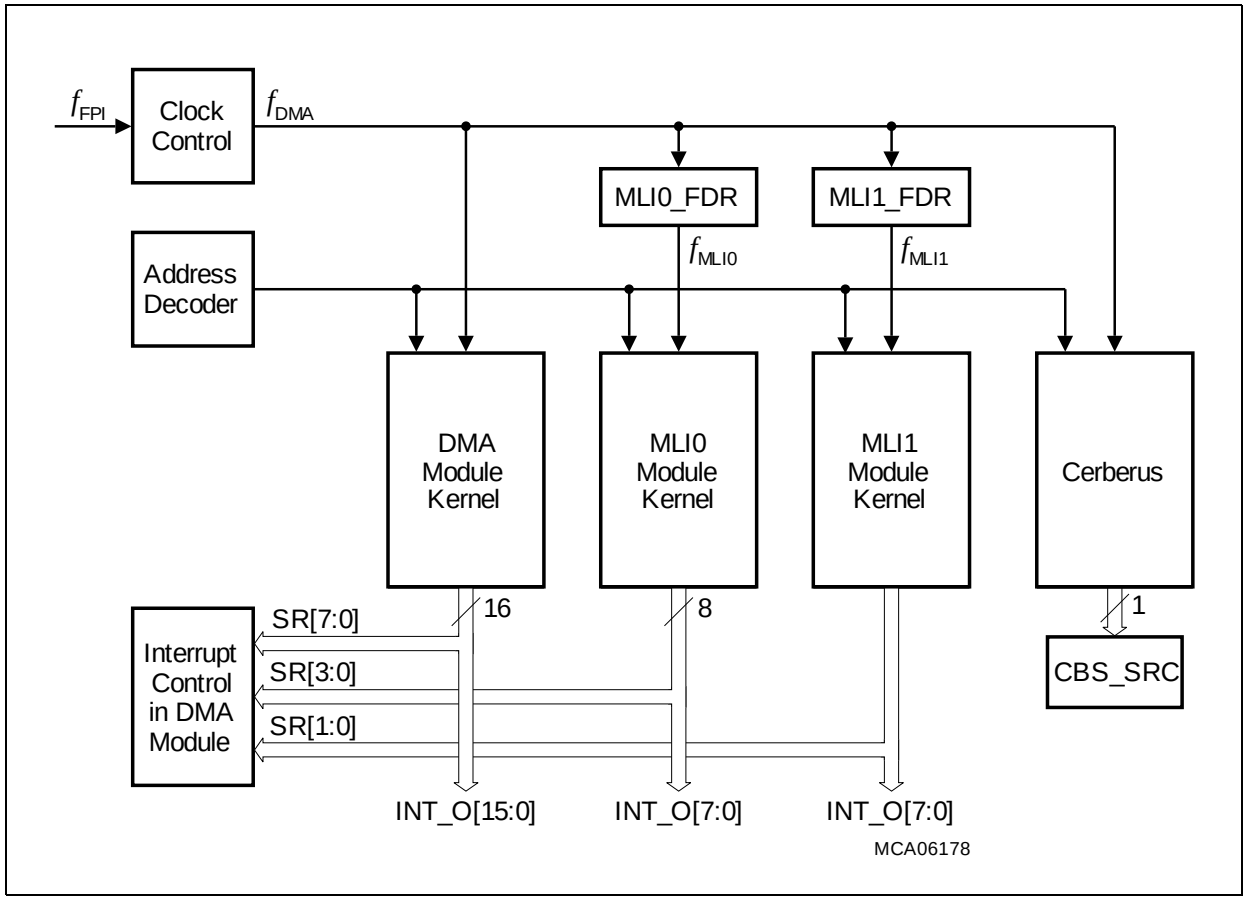


Figure 11-30 Implementation of the DMA Module and the MLI Modules

11.4.3.1 Clock Control Register

The Clock Control Register controls the DMA module internal f_{DMA} clock signal. This clock is also used for the MLI modules as a common clock that can be individually divided for the MLI modules.

DMA_CLC
DMA Clock Control Register (000_H) Reset Value: $0000\ 0008_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0										FSOE	SBWE	ONE	SPEN	DISS	DISR
r										rw	w	rw	rw	r	rw

Field	Bits	Type	Description
DISR	0	rw	**Module Disable Request Bit** Used for enable/disable control of the module
DISS	1	r	**Module Disable Status Bit** Bit indicates the current status of the module
SPEN	2	rw	**Module Suspend Enable for OCDS** Used to enable the Suspend Mode
ONE	3	rw	**Reserved;** returns 1if read; must be written with 1.
SBWE	4	w	**Module Suspend Bit Write Enable for OCDS** Determines whether SPEN and FSOE are write-protected.
0	5	rw	**Reserved;** returns 0 if read; must be written with 0.
0	[31:6]	r	**Reserved** Read as 0; should be written with 0.

Note: After a hardware reset operation, the DMA module is enabled.

Note: The suspend mode does not modify any of the registers.

11.4.3.2 DMA Interrupt Registers

In the TC1797, the lower eight DMA controller interrupts SR[7:0] are connected to service request control registers. The upper eight DMA controller interrupt outputs SR[15:8] are used as DMA channel request inputs (**Page 11-102**).

DMA_SRCx (x = 0-7)
DMA Service Request Control Register x
($2FC_H$ - $x*4_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SETR	CLRR	SRR	SRE	0	TOS	0		SRPN							
w	w	rh	rw	r	rw	r		rw							

Field	Bits	Type	Description
SRPN	[7:0]	rw	**Service Request Priority Number**
TOS	10	rw	**Type of Service Control** 0_B CPU service is initiated 1_B PCP request is initiated
SRE	12	rw	**Service Request Enable**
SRR	13	rh	**Service Request Flag**
CLRR	14	w	**Request Clear Bit**
SETR	15	w	**Request Set Bit**
0	[9:8], 11, [31:16]	r	**Reserved** Read as 0; should be written with 0.

11.4.3.3 MLI Interrupt Registers

The Service Request Control Registers of the MLI modules are located inside the DMA address area, because the MLI modules do not have own FPI Bus interfaces. The MLI modules shares one FPI Bus slave interface with the DMA controller.

The MLI0 module has eight interrupt output lines. Only four of them [3:0] are controlled by the MLI0 service request registers. The MLI1 module has also eight interrupt output lines, but only two of them [1:0] are controlled by the MLI1 service request registers.

DMA_MLI0SRCx (x = 0-3)
DMA MLI0 Service Request Control Register x
($2AC_H$ - $x*4_H$) **Reset Value: 0000 0000_H**

DMA_MLI1SRCy (y = 0-1)
DMA MLI1 Service Request Control Register y
($2BC_H$ - $y*4_H$) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SETR	CLRR	SRR	SRE	0	TOS	0		SRPN							
w	w	rh	rw	r	rw	r		rw							

Field	Bits	Type	Description
SRPN	[7:0]	rw	**Service Request Priority Number**
TOS	10	rw	**Type of Service Control** 0_B CPU service is initiated 1_B PCP request is initiated
SRE	12	rw	**Service Request Enable**
SRR	13	rh	**Service Request Flag**
CLRR	14	w	**Request Clear Bit**
SETR	15	w	**Request Set Bit**
0	[9:8], 11, [31:16]	r	**Reserved** Read as 0; should be written with 0.

Note: The bit coding of the MLI0/MLI1 service request registers is identical to that of the DMA Service Request Control Registers shown on the previous page.

11.4.4 Address Map

The DMA controller register block address map is shown in **Figure 11-31**. It shows how the different register blocks are arranged and adds the absolute address information.

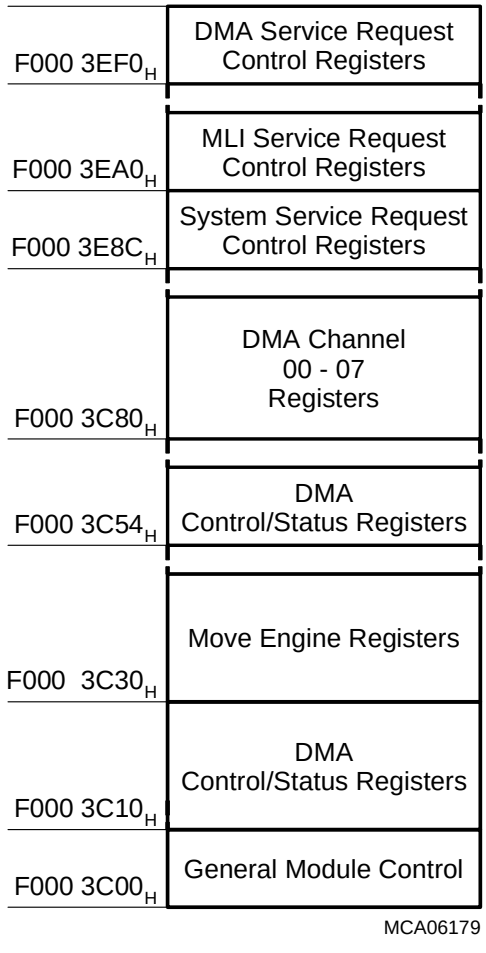


Figure 11-31 DMA Controller Register Block Address Map

11.5 Memory Checker Module

The Memory Checker Module (MCHK) includes two parallel Cyclic Redundancy Checkers (CRCs) that can be used to check the data consistency of two memories in parallel.

11.5.1 Functional Description

The Memory Checker module is connected to the DMA Peripheral Interface and can be accessed via the SPB. Preferable the module is used in combination with the DMA as it as described hereafter: a DMA channel can be used to read 8-bit, 16-bit, or 32-bit data from an address area and to write the data in the memory checker input register. With each write operation to the memory checker input register a polynomial checksum calculation is triggered and the result of the calculation is stored in the memory checker result register.

In order to start a memory check sequence, the memory checker result register must be initialized (e.g. written with $FFFF_H$ or with a desired start value) and a DMA transaction must be set up (start address, length, etc.). When programming the DMA channel for the memory checker with CHCRmn.RROAT = 1, one DMA transfer request (software or hardware triggered) starts the DMA transaction.

During the read move operations of the DMA transaction, data is always read from the memory and then written into the memory checker input register for the polynomial checksum calculation. At the end of the transaction (CHSRmn.TCOUNT = 0), an interrupt can be generated by the DMA channel (if CHCRmn.RROAT = 1), and the memory checker result register can be read out by software.

The memory checker uses the standard Ethernet polynomial, which is given by:

$$G^{32} = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1 \quad (11.1)$$

Note: Although the polynomial above is used for generation, the generation algorithm differs from the one that is used by the Ethernet protocol.

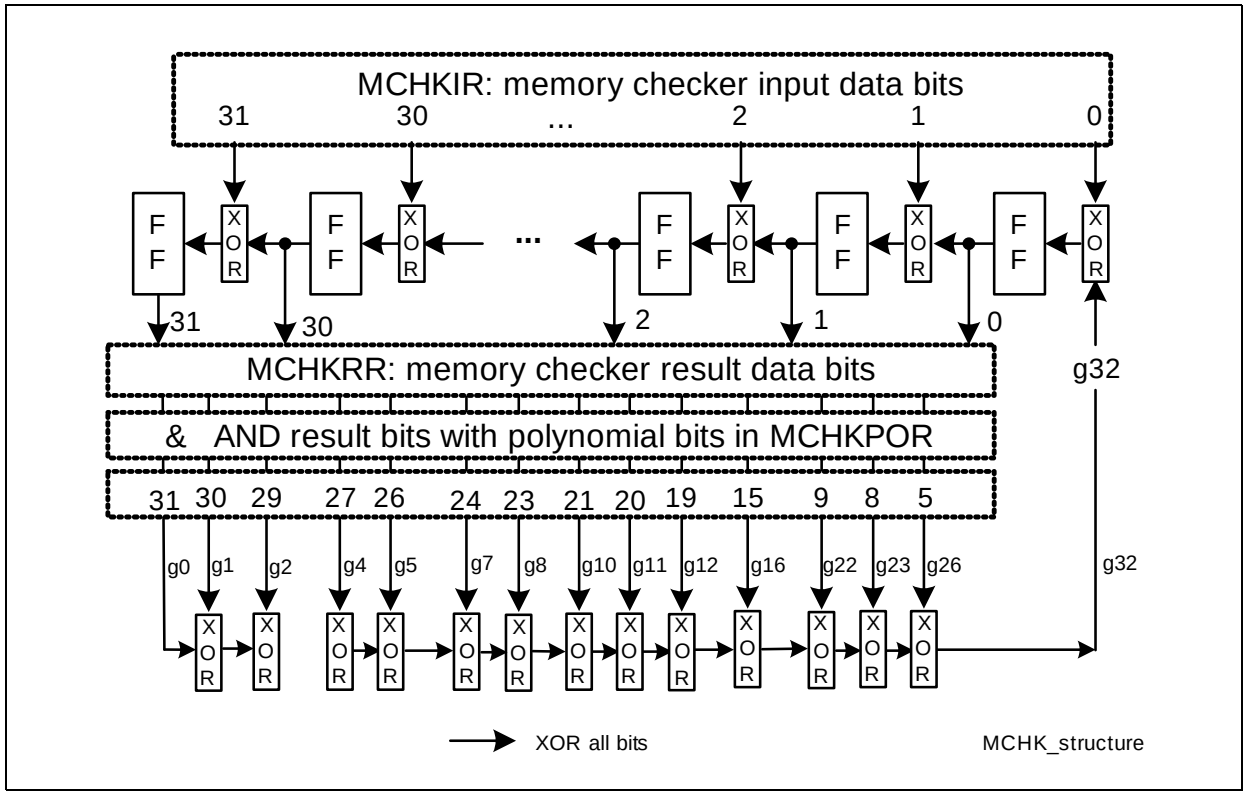


Figure 11-32 Implementation of the Memory Checker algorithm

11.5.2 Memory Checker Module Registers

This section describes the kernel registers of the Memory Checker module.

MCHK Register Overview

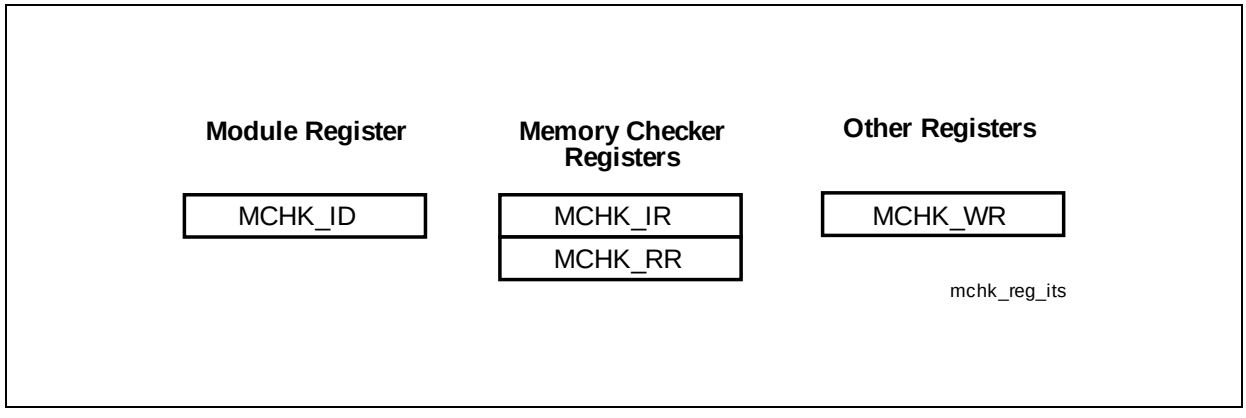


Figure 11-33 Memory Checker Registers

Table 11-18 Registers Address Space - Memory Checker Module Address Space

Module	Base Address	End Address	Note
MCHK	$F010\ C200_H$	$F010\ C2FF_H$	

Table 11-19 Registers Overview - Memory Checker Module Control Registers

Short Name	Description	Offset Addr.[1)]	Access Mode		Reset Class	Description See
			Read	Write		
-	Reserved	000_H - 004_H	BE	BE	-	-
MCHK_ID	Module Identification Register	008_H	US, V	BE	-	Page 11-130
-	Reserved	$00C_H$	BE	BE	-	-
MCHK_IR	Memory Checker Input Register	010_H	U, SV	U, SV	3	Page 11-131
MCHK_RR	Memory Checker Result Register	014_H	U, SV	U, SV	3	Page 11-131
-	Reserved	018_H - $01C_H$	BE	BE	-	-

Table 11-19 Registers Overview - Memory Checker Module Control Registers

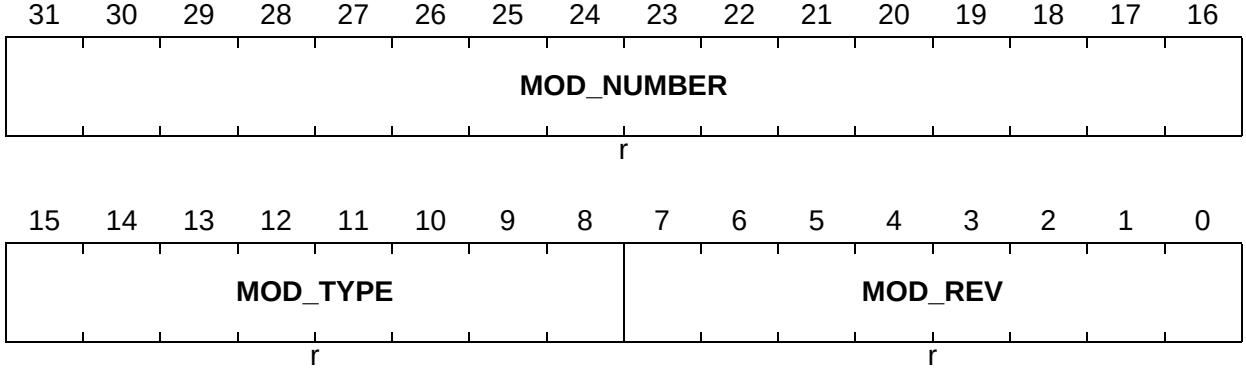
Short Name	Description	Offset Addr.[1]	Access Mode		Reset Class	Description See
			Read	Write		
MCHK_WR	Memory Checker Write Register	020_H	U, SV	U, SV	3	Page 11-132
-	Reserved	024_H - $0FC_H$	BE	BE	-	-

1) The absolute register address is calculated as follows:
Module Base Address (Table 11-18) + Offset Address (shown in this column)

11.5.2.1 Memory Checker Module Control Registers

The identification register allows the programmer version-tracking of the module. The table below shows the identification register which is implemented in the MCHK module.

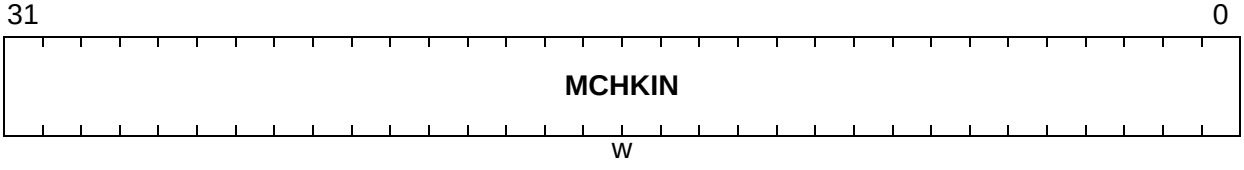
MCHK_ID
Module Identification Register (008_H) Reset Value: 001B C0XX$_H$



Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision Number** This bit field defines the module revision number. The value of a module revision starts with 01_H (first revision).
MOD_TYPE	[15:8]	r	**Module Type** The bit field is set to $C0_H$ which defines the module as a 32-bit module.
MOD_NUMBER	[31:16]	r	**Module Number Value** This bit field defines a module identification number. The value for the Memory Checker module is $001B_H$.

A Memory Checker Input Register is used during write moves of a memory checker related DMA transaction as data destination with its fixed register address. If the DMA moves to registerMCHK_IR0 are 8-bit or 16-bit wide, the unused register bits of the 32-bit MCHKIN value are taken as 0s for the current result calculation.

MCHK_IR
Memory Checker Input Register (010_H) Reset Value: 0000 0000_H

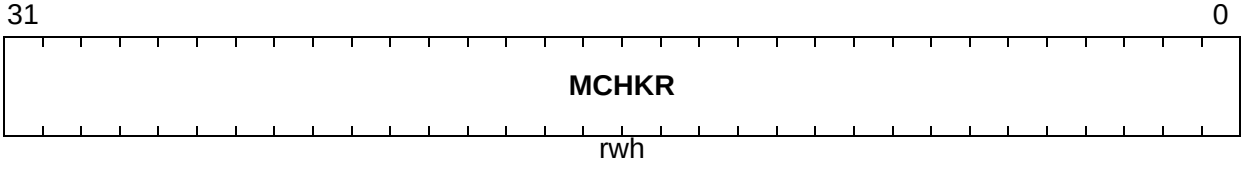


Field	Bits	Type	Description
MCHKIN	[31:0]	w	**Memory Checker Input** The value written to MCHKIN is used for the next checksum calculation. Any read action will deliver 0.

Note: MCHK_IR is a write-only register. Any read action will deliver 0000 0000_H.

A Memory Checker Result Register contains the result of the memory check operation. Before starting a checksum calculation operation, it should be written with an initial checksum calculation value.

MCHK_RR
Memory Checker Result Register (014_H) Reset Value: 0000 0000_H

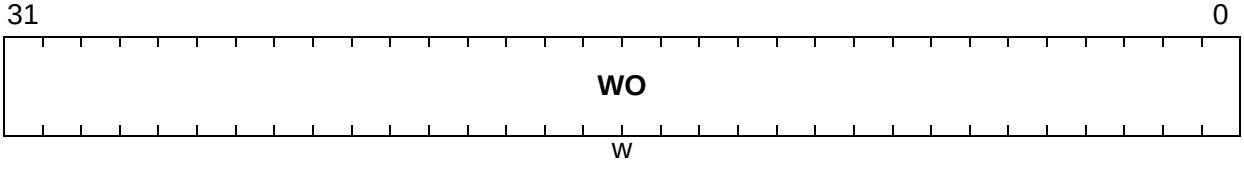


Field	Bits	Type	Description
MCHKR	[31:0]	rwh	**Memory Checker Result** This bit field contains the current result of the memory checksum calculation operation.

The Memory Checker Write Register is a dummy write-only register that is located within the memory checker address range.

The Memory Checker Write Register can be used as dummy write register at the write back action of the DMA or MLI controller Move Engine when the pattern detection feature of the DMA controller is used. Accessing MCHK_WR with the Move Engine of the MLI or DMA controller via the Bus Switch of the DMA controller (see **Figure 11-14**) does not request the two FPI buses of the TC1797, SPB and DMA, because it is near the MLI modules address ranges.

MCHK_WR
Memory Checker Write Register (020_H) Reset Value: 0000 0000_H



Field	Bits	Type	Description
WO	[31:0]	w	**Write-Only** This write-only bit field is used to write dummy data during DMA pattern detection. The data written to WO is not taken into account for any action. Any read action of WO will deliver 0000 0000_H.

12 LMB External Bus Unit

The External Bus Unit (EBU) of the TC1797 controls the transactions between external memories or peripheral units, and the internal memories and peripheral units. The basic interfaces of the EBU are shown in **Figure 12-1**.

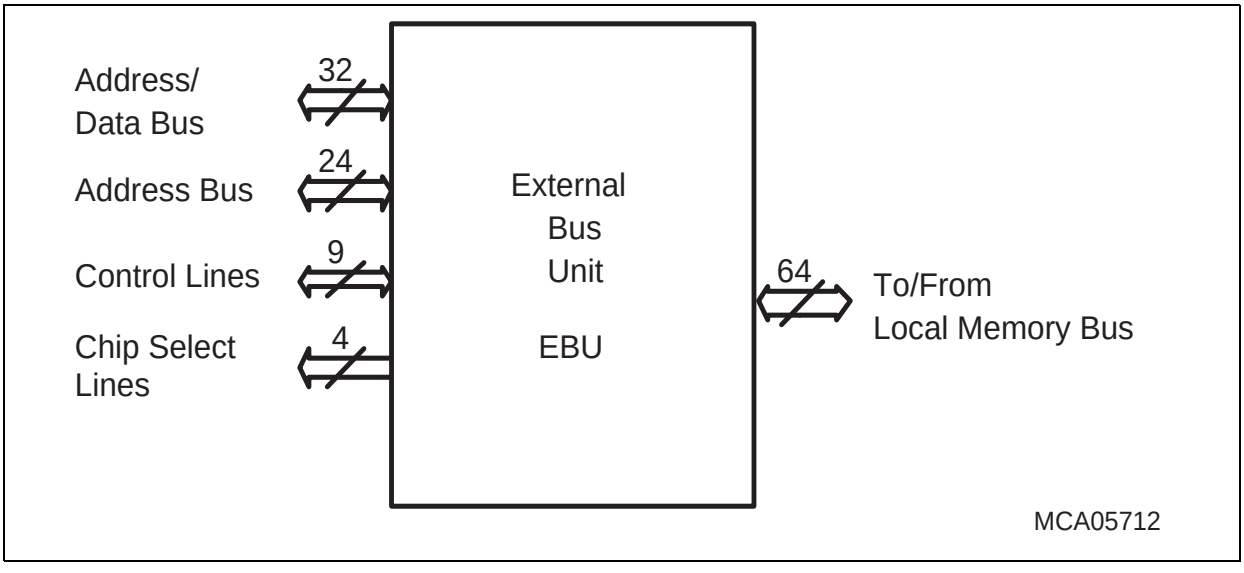


Figure 12-1 EBU Interface Diagram

The EBU is internally connected to the Local Memory Bus (LMB). This configuration makes it possible to get fast access times, especially when using external Burst Flash memory devices.

12.1 Feature List

The following features are supported by the EBU:

- 64-bit internal LMB interface
- 32-bit external bus interface
  - Support for 16 or 32 bit devices directly. Support for 8 bit devices only with assistance from software
  - Support for devices with multiplexed address/data bus as well as devices with separate address and data buses
  - Support for Burst Flash memory devices
  - Flexibly programmable access parameters
  - Different Device settings available for read and write accesses
  - Programmable chip select lines
  - Arbitration support for external EBU bus master devices
- Scalable external bus frequency
  - Derived from LMB frequency (f_{CPU}) divided by 2, 3, or 4
  - Maximum 75 MHz

- Data buffering supported
  – Read/write buffer

12.2 Block Diagram

Figure 12-2 shows the building blocks of the EBU.

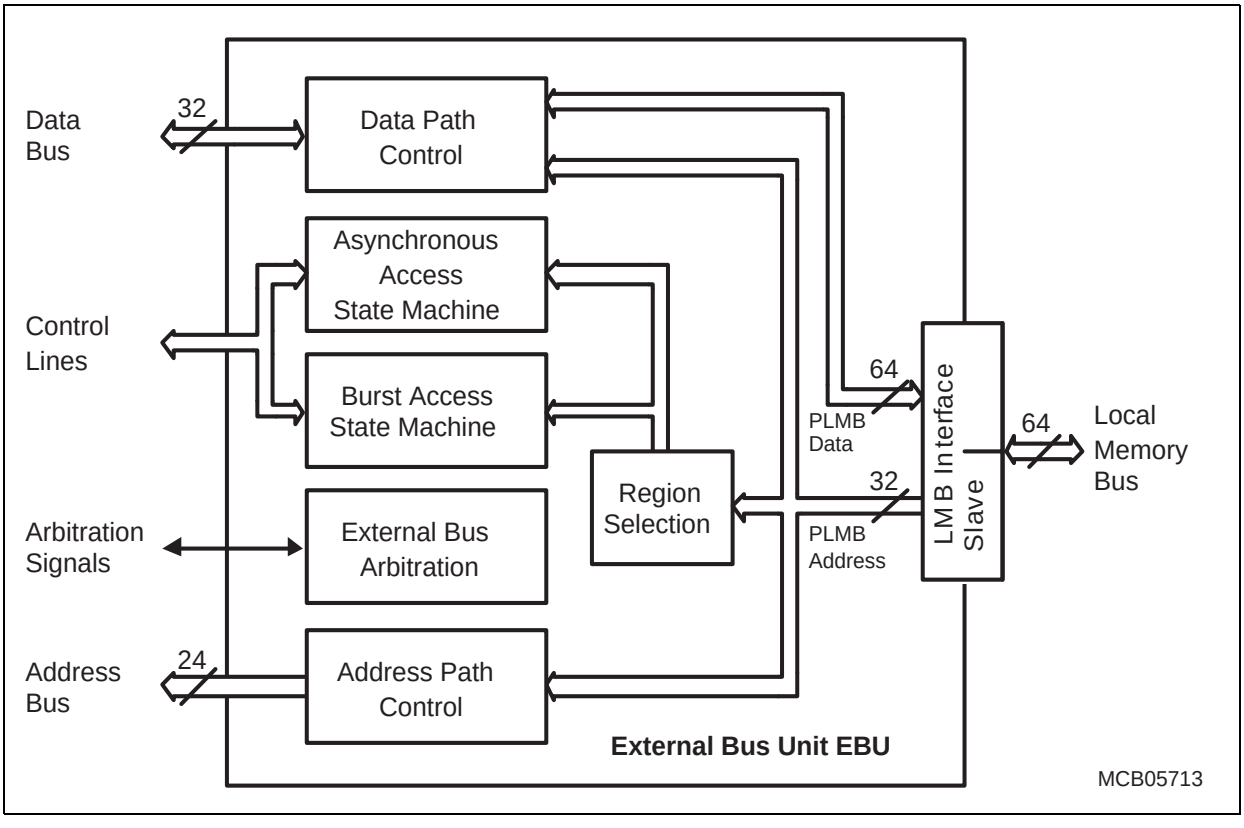


Figure 12-2 EBU Block Diagram

The EBU takes LMB transactions and translates them into the appropriate external access(es). Two dedicated state machines control either asynchronous external accesses or Burst Mode external accesses. The EBU further contains blocks that control the external bus arbitration, the region selection, as well as the data and address paths.

12.3 EBU Interface Signals

The external EBU interface signals are listed in **Table 12-1** below.

Table 12-1 EBU Interface Signals

Signal/Pin	Type	Function
AD[31:0]	O	Multiplexed Address/Data bus lines 0-31
A[23:0]	O	Address bus lines 0-23
$\overline{\text{CS}}$[3:0]	O	Chip select 0-3
$\overline{\text{CSCOMB}}$	O	Combined chip select for global select
$\overline{\text{RD}}$	O	Read control line
RD/$\overline{\text{WR}}$	O	Write control line
$\overline{\text{ADV}}$	O	Address valid output
$\overline{\text{BC}}$[3:0]	O	Byte control lines 0-3
BFCLKO	O	Clock Output for Synchronous Accesses
BFCLKI	I	Feedback clock input for Synchronous Accesses
$\overline{\text{WAIT}}$	I	Wait input
MR/$\overline{\text{W}}$	O	Write Control line with timing suitable for Motorola Peripherals
$\overline{\text{BAA}}$	O	Burst address advance output
$\overline{\text{HOLD}}$	I	Hold request input
$\overline{\text{HLDA}}$	O	Hold acknowledge output
$\overline{\text{BREQ}}$	O	Bus request output
V_{DDEBU}	–	EBU power supply lines (2.3 - 3.3 V)

12.3.1 Address/Data Bus, AD[31:0]

This bus transfers address and data information. The width of this bus is 32 bits. External devices with 8, 16 or 32 bits of data width can be connected to the data bus. Burst Mode instruction fetches can be performed with only 32 or 16 bit data bus width. 8bit devices have to be treated as 16 bit devices and data alignment accomplished using software.

The EBU adjusts the data on the data bus to the width of the external device, according to the programmed parameters in its control registers. The byte control signals $\overline{\text{BC}}$[3:0] specify which parts of the data bus carry valid data.

12.3.2 Address Bus, A[23:0]

The total address bus of the EBU consists of 24 address output lines, giving a directly addressable range of 64 Mbytes (2^{24}, 32 bit words). An external device can be selected via one of the chip select lines. Since there are four chip select lines, four such devices with up to 256 Mbytes of address range can be used in the external system.

12.3.3 Chip Selects, $\overline{\text{CS}}$[3:0]

The EBU provides four chip select outputs, $\overline{\text{CS0}}$, $\overline{\text{CS1}}$, $\overline{\text{CS2}}$ and $\overline{\text{CS3}}$. The address ranges for which these chip selects are generated are programmed separately for each chip select line in a very flexible way via the address select registers EBU_ADDRSELx. More details are described on **Page 12-12**.

12.3.4 Global Chip Select, $\overline{\text{CSCOMB}}$

The EBU provides a combined or global chip select line. This can be programmed to be asserted with any combination of the normal chip selects $\overline{\text{CS}}$[3:0]. See **Chapter 12.4.3** for more details.

12.3.5 Read/Write Control Lines, RD, RD/$\overline{\text{WR}}$

Two lines are provided to trigger the read ($\overline{\text{RD}}$) and write (RD/$\overline{\text{WR}}$) operations of external devices. While some read/write devices require both signals, there are devices with only one control input. The RD/$\overline{\text{WR}}$ line is then used for these devices. This line will go to an active-low level on a write, and will stay inactive high on a read. The external device should only evaluate this signal in conjunction with an active chip select. Thus, an active chip select in combination with a high level on the RD/$\overline{\text{WR}}$ line indicates a read access to this device.

12.3.6 Address Valid, $\overline{\text{ADV}}$

The address valid signal, $\overline{\text{ADV}}$, validates the address lines A[23:0] (and also the address placed on the data bus AD[31:0] when attaching multiplexed address/data devices). It can be used to latch these addresses externally.

12.3.7 Byte Controls, $\overline{\text{BC}}$[3:0]

The byte control signals $\overline{\text{BC}}$[3:0] select the appropriate byte lanes of the data bus for both read and write accesses. **Table 12-2** shows the activation on access to a 16-bit or 8-bit external device. Please note that this scheme supports little-endian devices.

Table 12-2 Byte Control Pin Usage

Width of External Device	$\overline{BC3}$	$\overline{BC2}$	$\overline{BC1}$	$\overline{BC0}$
32-bit device with byte write capability	D[31:24]	D[23:16]	D[15:8]	D[7:0]
16-bit device with byte write capability	inactive (high)	inactive (high)	D[15:8]	D[7:0]
8-bit device	inactive (high)	inactive (high)	inactive (high)	D[7:0]

Signals $\overline{BCx}$ can be programmed for different timing. The available modes cover a wide range of external devices, such as RAM with separate byte write-enable signals, and RAM with separate byte chip select signals. This allows external devices to connect without any external “glue” logic.

Table 12-3 Byte Control Signal Timing Options

Mode	EBU_BUSCONx. BCGEN	Description
Chip Select Mode	00_B	$\overline{BCx}$ signals have the same timing as the generated chip select $\overline{CS}$.
Control Mode	01_B	$\overline{BCx}$ signals have the same timing as the generated control signals $\overline{RD}$ or $\overline{RD/WR}$.
Write Enable Mode	10_B	$\overline{BCx}$ signals have the same timing as the generated control signal $\overline{RD/WR}$.

12.3.8 Burst Flash Clock Output/Input, BFCLKO/BFCLKI

The clock output signal of the EBU is provided at pin BFCLKO. It is used for timing purposes (timing reference) during Burst Mode accesses. BFCLKO is, by default, only generated during synchronous accesses.

The clock input BFCLKI of the EBU is used to latch read data into the EBU. Normally BFCLKI is directly feedback and connected to BFCLKO. This feedback path can be configured externally to maximize the operating frequency for a given Flash device or to compensate the BFCLKO clock pad delay. More details are given on **Page 12-72**.

12.3.9 Wait Input, $\overline{WAIT}$

This is an input signal to the EBU that is used to dynamically insert wait states into read or write data cycles controlled by the device on the external bus.

12.3.10 Burst Address Advance, $\overline{\text{BAA}}$

The Burst Address Advance output, $\overline{\text{BAA}}$, is provided for advancing the addresses during a Burst Flash access cycle.

12.3.11 Motorola Peripheral Write Signal, MR/$\overline{\text{W}}$

A write control signal held valid for the duration of an access (from chip select asserted to chip select negated).

12.3.12 Bus Arbitration Signals, $\overline{\text{HOLD}}$, $\overline{\text{HLDA}}$, and $\overline{\text{BREQ}}$

The $\overline{\text{HOLD}}$ input signal is the external bus arbitration signal that indicates to the EBU when an external bus master requests to obtain ownership of the external bus.

The $\overline{\text{HLDA}}$ output signal is the external bus arbitration signal that indicates to a external bus master that it has obtained ownership of the external bus from the EBU.

The $\overline{\text{BREQ}}$ output signal is the external bus arbitration signal that is asserted by the EBU when it requests to obtain back the ownership of the external bus.

12.3.13 EBU Power Supply

The V_{DDEBU} lines provide the power supply (2.3 - 3.6 V) for the dedicated pins of the EBU interface.

12.3.14 EBU Reset

The EBU is asynchronously initialised by the application reset.

12.3.15 Allocation of Unused Signals as GPIO

The EBU will allow pins not required for its programmed configuration to be allocated for use as GPIO. The signals required are as defined below:

Table 12-4 EBU Interface Signals Required by Operating Mode

Signal/Pin	When Needed by EBU
AD[15:0]	Always needed when the EBU is enabled. EBU_MODCON.ARBMODE != 00_B [1)]
A[23:0]	
$\overline{RD}$	
RD/$\overline{WR}$	
$\overline{ADV}$	
$\overline{BC}$[1:0]	
BFCLKO	
BFCLKI	
$\overline{WAIT}$	
MR/$\overline{W}$	
$\overline{BAA}$	
$\overline{HOLD}$	
$\overline{HLDA}$	
$\overline{BREQ}$	
AD[31:16]	These signals are required by the EBU when the EBU is enabled and an active region is configured for accessing 32 bit memory (EBU_MODCON.ARBMODE != 00_B) AND (EBU_ADDRSELx.REGENAB=1_B OR EBU_ADDRSELx.ALTENAB=1_B) AND (BUSRCONx.PORTW = 10_B or 11_B)
$\overline{BC}$[3:2]	
$\overline{CS}$[3:0]	The Chip select lines are individually controlled and will be required for EBU operation when the associated memory region is active. (EBU_MODCON.ARBMODE != 00_B) AND (EBU_ADDRSELx.REGENAB=1_B OR EBU_ADDRSELx.ALTENAB=1_B)
$\overline{CSCOMB}$	The Global Chip Select line is required by the EBU when the EBU_MODCON.GLOBALCS != 0000_B and the EBU is enabled (EBU_MODCON.ARBMODE != 00_B)

1) If the EBU is disabled by writing 00 to the EBUCON.ARBMODE field, there will be a delay before the signals become available for GPIO usage as the EBU will wait for all pending external memory accesses to be completed and the arbitration logic to return to the “nobus” state before releasing the signals.

12.3.16 Control of Pad Pull Up and Pull Down.

The EBU has no control of the Pull Up and Pull Down components in its pads. These are all controlled by the GPIO blocks associated with the pads. The default, after reset, is for the pull up component to be active. This default should be reprogrammed as required by the application during the initialisation of the EBU.

Table 12-5 Suggested Pull Component Configuration for EBU Signals

Signal/Pin	Configuration of Pull Component	
	Sole Master	Arbitration enabled
AD[31:0]	Pull Up	
A[23:0]	None	Pull Up
$\overline{RD}$	None	Pull Up
RD/$\overline{WR}$	None	Pull Up
$\overline{ADV}$	None	Pull Up
$\overline{BC}$[3:0]	None	Pull Up
BFCLKO	None[1)]	
BFCLKI	None	
$\overline{WAIT}$	Pull Up	
MR/$\overline{W}$	None	Pull Up
$\overline{BAA}$	None	Pull Up
$\overline{HOLD}$	Pull Up	
$\overline{HLDA}$	Pull Up	
$\overline{BREQ}$	Pull Up	
$\overline{CS}$[3:0]	None	Pull Up
$\overline{CSCOMB}$	None	Pull Up

1) The BFCLKO output is always driven by the EBU and is not arbitrated.

12.4 External Bus Operation

The EBU supports interconnection to a wide variety of memory/peripheral devices with flexible programming of the access parameters. In the following sections, the basic features for these access modes are described. The types of external access cycles provided by the EBU are:

- Asynchronous and synchronous devices with demultiplexed access
  - ROMs, EPROMs
  - NOR flash devices
  - Static RAMs and PSRAMs/SSRAMS
- Asynchronous and synchronous devices with multiplexed access
  - NOR flash devices
  - PSRAMs/SSRAMS

Note: Not all memory types supported by the memory controller are known to be available in Automotive quality grades.

Each TC1797 internal LMB master can access external devices via the EBU. The EBU provides four user-programmable external memory regions. Each of these regions is provided with a set of registers that determine the parameters of the external bus transaction and one chip select signal. A LMB transaction that matches one of these user-programmable external memory regions is translated by the EBU to the appropriate external access(es).

In the TC1797, the EBU responds to the address ranges as defined in **Table 12-6**.

Table 12-6 EBU External Address Ranges

Address Range	Description	Action
$8080\ 0000_H$ - $8FDF\ FFFF_H$	External memory space (cached area)	compare
$A080\ 0000_H$ - $AFDF\ FFFF_H$	External memory space (non-cached area)	
$D800\ 0000_H$ - $DDFF\ FFFF_H$	External Peripheral Space (non-cached area)	
$DE00\ 0000_H$ - $DEFF\ FFFF_H$	External Emulator Space (non-cached area)	
$E000\ 0000_H$ - $E7FF\ FFFF_H$	External Peripheral Space (non-cached area)	
$F800\ 0000_H$ - $F800\ 03FF_H$	EBU Registers	access register

The “compare” action means that the EBU compares the supplied LMB address to all its external regions. If a match is found, the EBU performs the appropriate external bus access. Otherwise, the EBU generates a LMB Error Acknowledge.

The “access registers” action means that the EBU is selected for a control/status register access. The EBU performs the requested register access (or generates an LMB Error Acknowledge if there is no register at the supplied address).

For all address ranges that not listed in **Table 12-6** the EBU is not selected and LMB requests are ignored.

12.4.1 External Memory Regions

The EBU of the TC1797 supports four memory regions, which have its own associated chip select outputs $\overline{CS}$[3:0]. Each of these regions has a set of control registers to specify the type of memory/peripheral device and the access parameters.

Each of the four user-programmable regions (x = 0-3 is the numbering index of these regions) is can be configured to respond to a particular address space through registers EBU_ADDRSELx,

The access parameters for each of the regions can be programmed individually to accommodate different types of external devices. Separate control registers are available to control read and write accesses. This allows optimal access types, speeds and parameters to be chosen. Access type is configured via EBU_BUSRCONx and EBU_BUSWCONx. Access parameters are configured via EBU_BUSRAPx and EBU_BUSWAPx.

Throughout this document the generic term EBU_BUSCONx is used when either of EBU_BUSRCONx or EBU_BUSWCONx is applicable and EBU_BUSAPx is used when either of EBU_BUSRAPx or EBU_BUSWAPx is applicable.

Table 12-7 EBU Address Regions, Registers and Chip Selects

Region	Associated Chip Select	Address Select Registers	Bus Configuration Registers	Bus Access Parameters Registers
Region 0	$\overline{CS0}$	EBU_ADDRSEL0	EBU_BUSRCON0 EBU_BUSWCON0	EBU_BUSRAP0 EBU_BUSWAP0
Region 1	$\overline{CS1}$	EBU_ADDRSEL1	EBU_BUSRCON1 EBU_BUSWCON1	EBU_BUSRAP1 EBU_BUSWAP1
Region 2	$\overline{CS2}$	EBU_ADDRSEL2	EBU_BUSRCON2 EBU_BUSWCON2	EBU_BUSRAP2 EBU_BUSWAP2
Region 3	$\overline{CS3}$	EBU_ADDRSEL3	EBU_BUSRCON3 EUB_BUSWCON3	EBU_BUSRAP3 EBU_BUSWAP3

Table 12-8 lists the programmable parameters that are available for the four external regions (regions 0 to 3).

Table 12-8 Programmable Parameters of Regions

Register	Parameter (Bit/Bit field)	Function
EBU_ADDRSELx	ALTSEG	Alternate segment of region to be compared to LMB address bits [31:28].
	BASE	Region base address to be compared with LMB address in conjunction with the MASK parameter.
	MASK	Address mask for each external region. Specifies the number of right-most bits in the base address starting from bit 26.
	WPROT	Write Protect bit for each region.
	ALTENAB	Alternate segment comparison enable of a region: Determines whether or not parameter ALTSEG is always compared to LMB address.
	REGENAB	Enable bit for each region. A disabled region will always generate a miss during address comparison.
EBU_BUSCONx	AGEN	Region access type: Demultiplexed (asynchronous) or Burst Flash (synchronous) access

12.4.2 Chip Select Control

The EBU generates four chip select signals, $\overline{CSx}$, which are all available at dedicated chip select outputs.

12.4.3 Combined Chip Select ($\overline{CSCOMB}$)

The EBU can also generate a combined, global chip select. This is controlled using the EBU_CON.GLOBALCS register field. This is a four bit field and each bit corresponds to one of the normal chip selects. If the appropriate bit is set, then the CSCOMB chip select is asserted whenever the relevant normal chip select is asserted. Multiple bits in the GLOBALCS field can be set simultaneously.

Table 12-9 GLOBALCS bit mapping to $\overline{CSx}$

Bit Function	Bit
$\overline{CSCOMB}$ asserted with $\overline{CS0}$	
$\overline{CSCOMB}$ asserted with $\overline{CS1}$	
$\overline{CSCOMB}$ asserted with $\overline{CS2}$	
$\overline{CSCOMB}$ asserted with $\overline{CS3}$	

12.4.4 Programmable Device Types

Each CS region (0 to 3) can be individually configured using the EBU_BUSCONx.agen register field, to be connected to one of the following external memory/device types:

Table 12-10 agen description

agen value	Device Type
0	Muxed Asynchronous Type External Memory (default after reset)
1	Muxed Burst Type External Memory
2	NAND flash (page optimised, asynchronous)
3	Muxed Cellular RAM External Memory
4	Demuxed Asynchronous Type External Memory
5	Demuxed Burst Type External Memory
6	Demuxed Page Mode External Memory
7	Demuxed Cellular RAM External Memory
8	reserved
9	reserved
10	reserved

Table 12-10 agen description (cont'd)

agen value	Device Type
11	reserved
12	reserved
13	reserved
14	reserved
15	reserved

12.4.5 Support for Multiplexed Device Configurations

Memory Controller supports a number of configurations of Multiplexed memory/peripheral devices using different values of the EBU_BUSRCONx.PORTW bit-field. The EBU_BUSWCONx registers also contain the PORTW field but in this case the field is read only and reflects the value set in the related one of the EBU_BUSRCONx registers. The values set in the EBU_BUSRCONx registers are used for both read and write accesses.

Note: When using multiplexed devices a non-zero recovery phase is mandatory for all devices to prevent read data from one access conflicting with the address for the multiplexed memory.

Table 12-11 Pins used to connect Multiplexed Devices to Memory Controller

Memory Device Configuration	Memory Controller Pins			Section
	A(23:0)[1]	AD(31:16)[2]	AD(15:0)	
16-bit MUX	A(23:0)	-	A(15:0)/ D(15:0)	**16-bit Multiplexed Memory/Peripheral Configuration**
Twin 16-bit MUX	A(23:0)	A(15:0)/ D(31:16)	A(15:0)/ D(15:0)	**Twin 16-bit Multiplexed Device Configuration**
32-bit MUX	A(23:0)	A(31:16)/ D(31:16)	A(15:0)/ D(15:0)	**32-bit Multiplexed Memory/Peripheral Configuration**

1) These pins are always outputs which are connected to address pins on the Multiplexed device(s)

2) These pins are dual function and act as AD(31:16) when required for 32-bit, multiplexed devices

Table 12-12 Selection of Multiplexed Device Configuration

PORTW value	
00_B	reserved
01_B	16-bit multiplexed[1)]
10_B	Twin, 16-bit Multiplexed[2)]
11_B	32 bit multiplexed[3)]

1) Address will only be driven onto AD(15:0) during the address and address hold phases. A(15:0) will be driven with address for duration of access

2) Lower 16 bits of address will be driven onto both A(15:0) and AD(15:0) during the address and address hold phases

3) Full address will be driven onto A(15:0) and AD(15:0) during the address and address hold phases

16-bit Multiplexed Memory/Peripheral Configuration

Throughout the complete external bus cycle the address[1)] is driven onto Memory Controller pins A(23:0). During the address phase the low 16 bits of the address are driven to Memory Controller pins AD(15:0). Data (16-bit) is driven to/read back from the AD(15:0) pins during the data phase. The interconnect between Memory Controller and a 16-bit Multiplexed device in this mode is shown below (note: for clarity only the address/data signals are shown):-

1) This address is pre-aligned according to the bus width as detailed in **Section 12.4.11**.

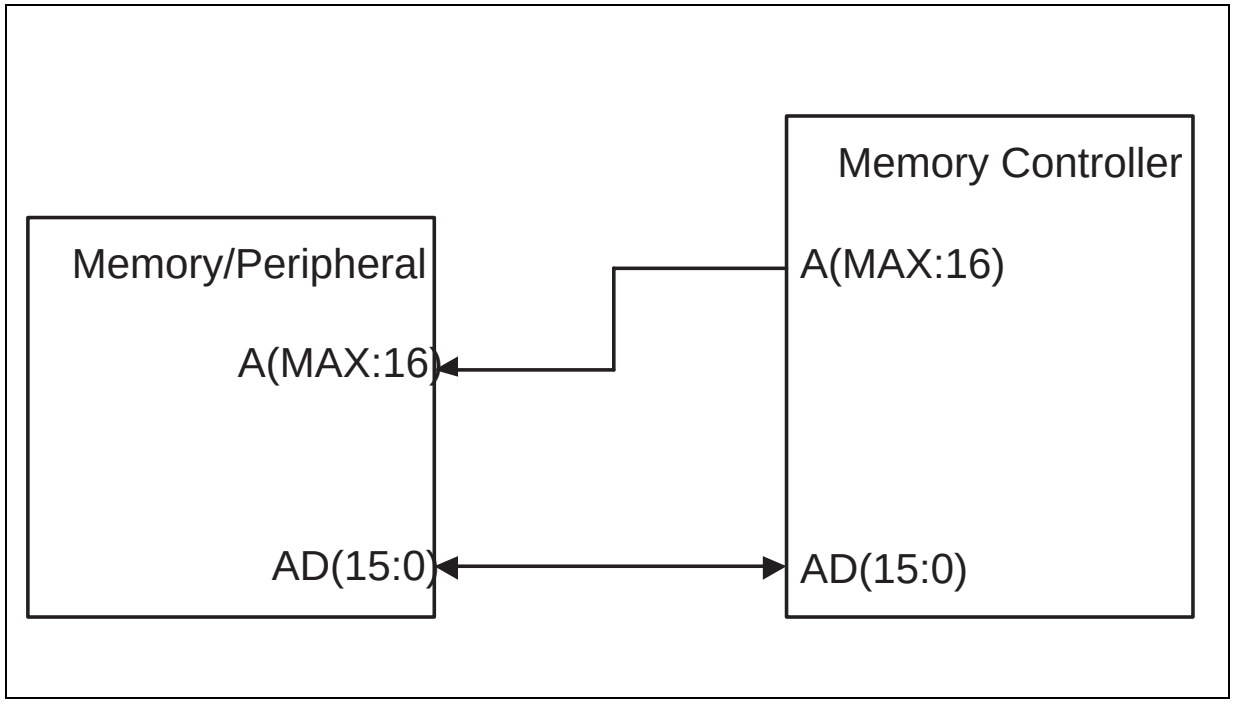


Figure 12-3 Connection of a 16-bit Multiplexed Device to Memory Controller

Twin 16-bit Multiplexed Device Configuration

This mode allows the use of two 16-bit multiplexed devices to create a 32-bit wide bus. Throughout the complete external bus cycle the address[1)] is driven onto Memory Controller pins A(23:0). During the address phase the low 16 bits of the address are driven (in parallel) to Memory Controller pins AD(15:0) and AD(31:16). This ensures that both multiplexed devices are issued with the same address during the address phase. Data (32-bit) is written to/read from the AD(31:16) pins for MSW and the AD(15:0) pins for LSW during the data phase. The interconnect between Memory Controller and two 16-bit Multiplexed devices in this mode is shown below (note: for clarity only the address/data signals are shown):-

1) This address is pre-aligned according to the bus width as detailed in **Section 12.4.11**.

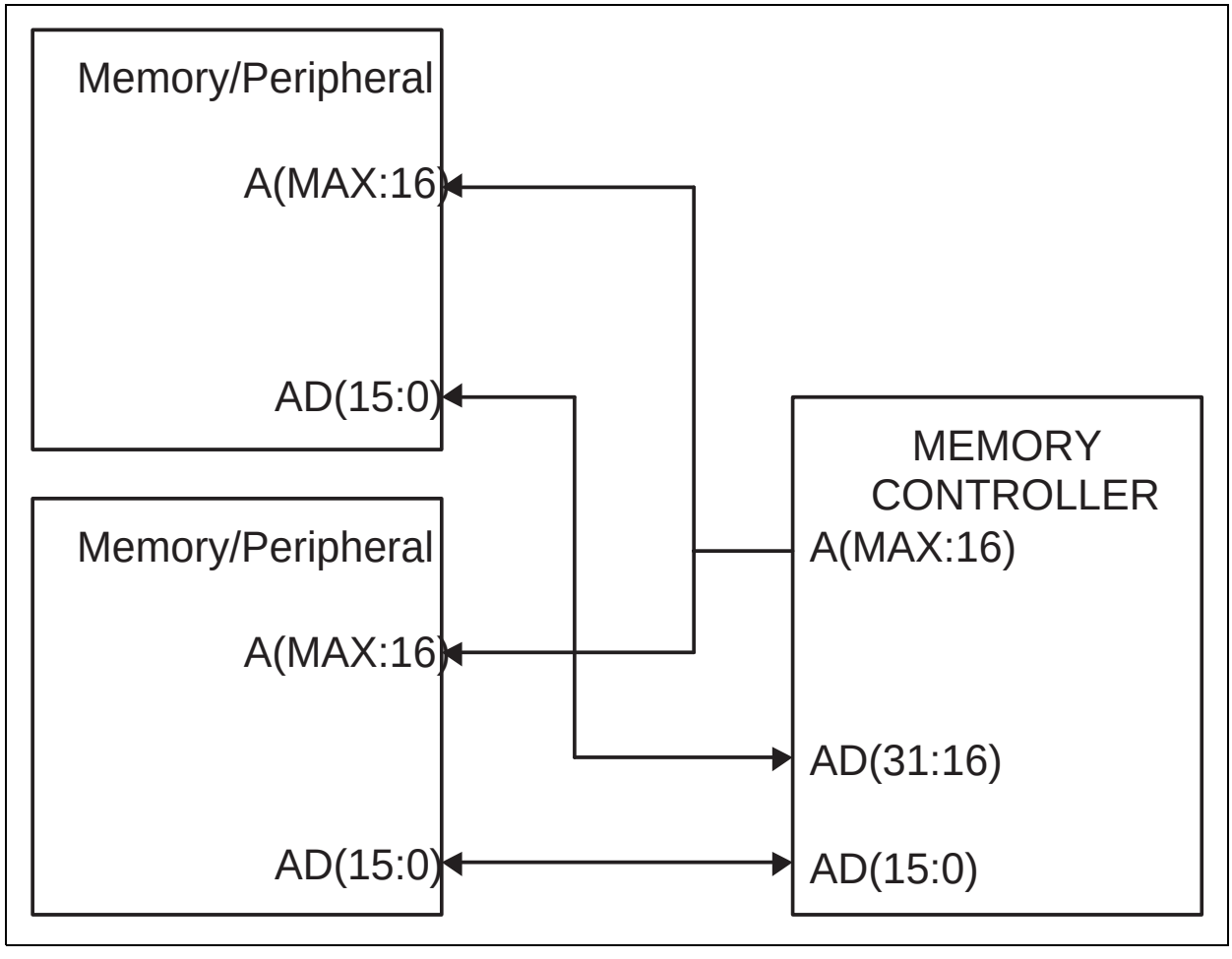


Figure 12-4 Connection of twin 16-bit Multiplexed Device's to Memory Controller

32-bit Multiplexed Memory/Peripheral Configuration

During the address phase the lower 16 bits of the 25 bit address are driven to Memory Controller pins AD(15:0), the most significant 9 bits of the address are driven to pins AD(24:16) and pins AD(31:17) are driven with 0 (zero). Data (32-bit) is driven to/read from the AD(31:0) pins during the data phase. The interconnect between Memory Controller and a 32-bit Multiplexed device in this mode is shown below (note: for clarity only the address/data signals are shown):-

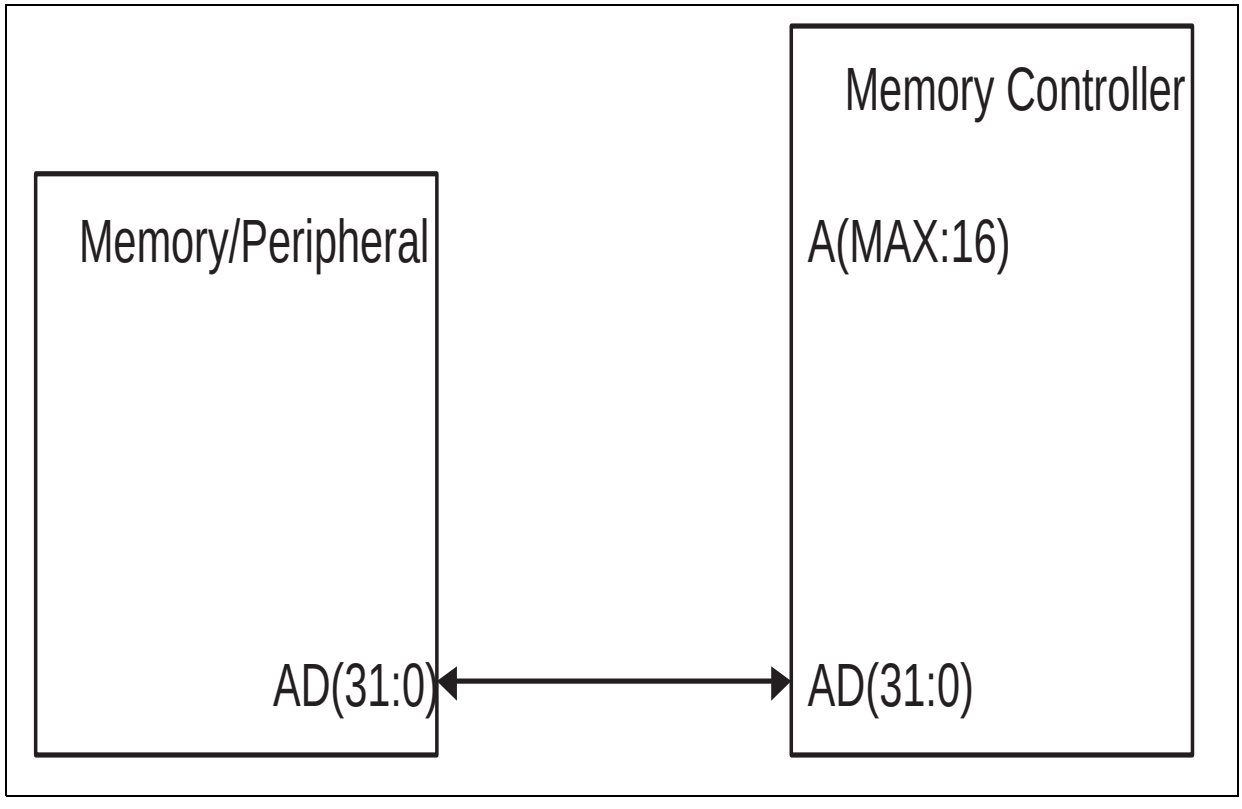


Figure 12-5 Connection of a 32-bit Multiplexed Device to Memory Controller

12.4.6 Support for Non-Multiplexed Device Configurations

The Memory Controller supports 16-bit and 32-bit non-multiplexed memory devices.

Table 12-13 Pins used to connect non-multiplexed Devices to Memory Controller

Memory Device Configuration	Memory Controller Pins			Section
	A(23:0)[1)]	AD(31:16)	AD(15:0)	
16-bit non-MUX	A(23:0)	D(31:16)	D(15:0)	**16-bit non-Multiplexed Memory/Peripheral Configuration**
32-bit non-MUX	A(23:0)	D(31:16)	D(15:0)	**32-bit non-Multiplexed Memory/Peripheral Configuration**

1) These pins are always outputs which are connected to address pins on the Multiplexed device(s)

Table 12-14 Selection of non-Multiplexed Device Configuration

PORTW value	
00_B	reserved
01_B	16-bit[1)]
10_B	reserved
11_B	32-bit

1) Address will only be driven onto AD(15:0) during the address and address hold phases. A(15:0) will be driven with address for duration of access

16-bit non-Multiplexed Memory/Peripheral Configuration

Throughout the complete external bus cycle the address[1)] is driven onto Memory Controller pins A(23:0). Data (16-bit) is driven to/read back from the AD(15:0) pins during the data phase. The interconnect between Memory Controller and a 16-bit non-Multiplexed device in this mode is shown below (note: for clarity only the address/data signals are shown):-

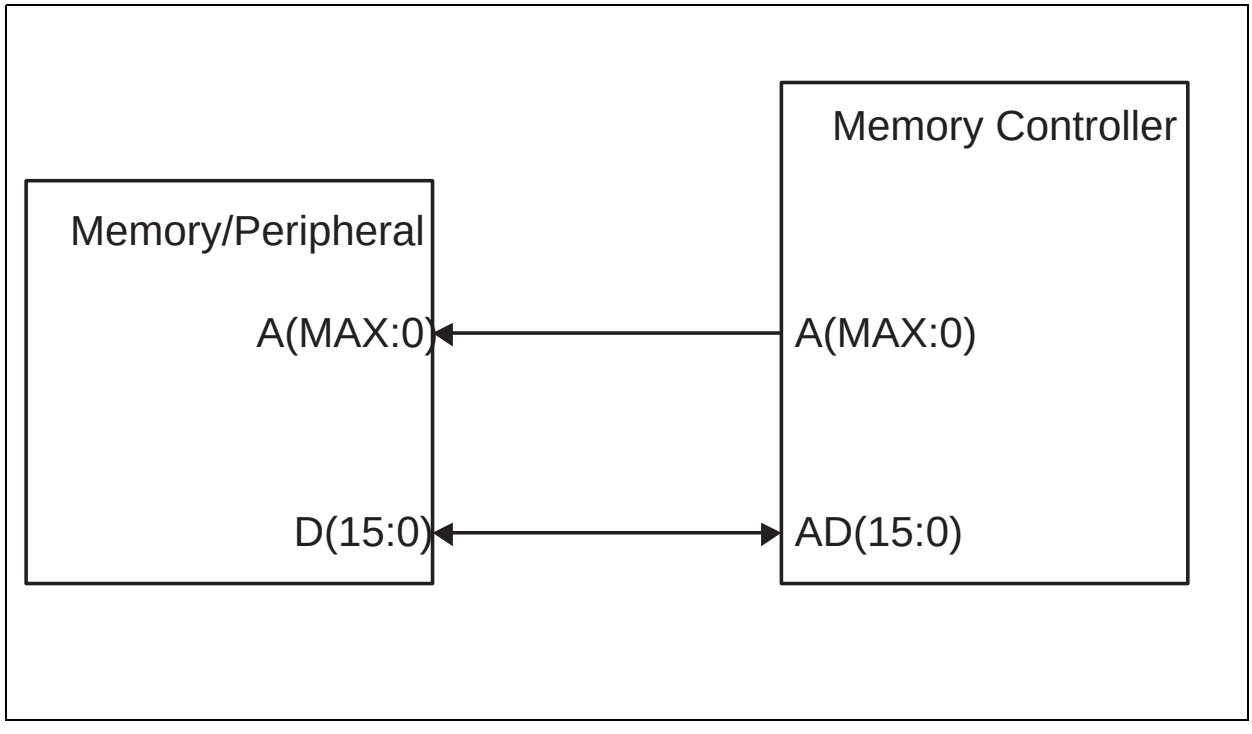


Figure 12-6 Connection of a 16-bit non-Multiplexed Device to Memory Controller

1) This address is pre-aligned according to the bus width as detailed in **Section 12.4.11**.

32-bit non-Multiplexed Memory/Peripheral Configuration

Throughout the complete external bus cycle the address[1)] is driven onto Memory Controller pins A(23:0). Data (32-bit) is driven to/read back from the AD(31:0) pins during the data phase. The interconnect between Memory Controller and a 16-bit non-Multiplexed device in this mode is shown below (note: for clarity only the address/data signals are shown):-

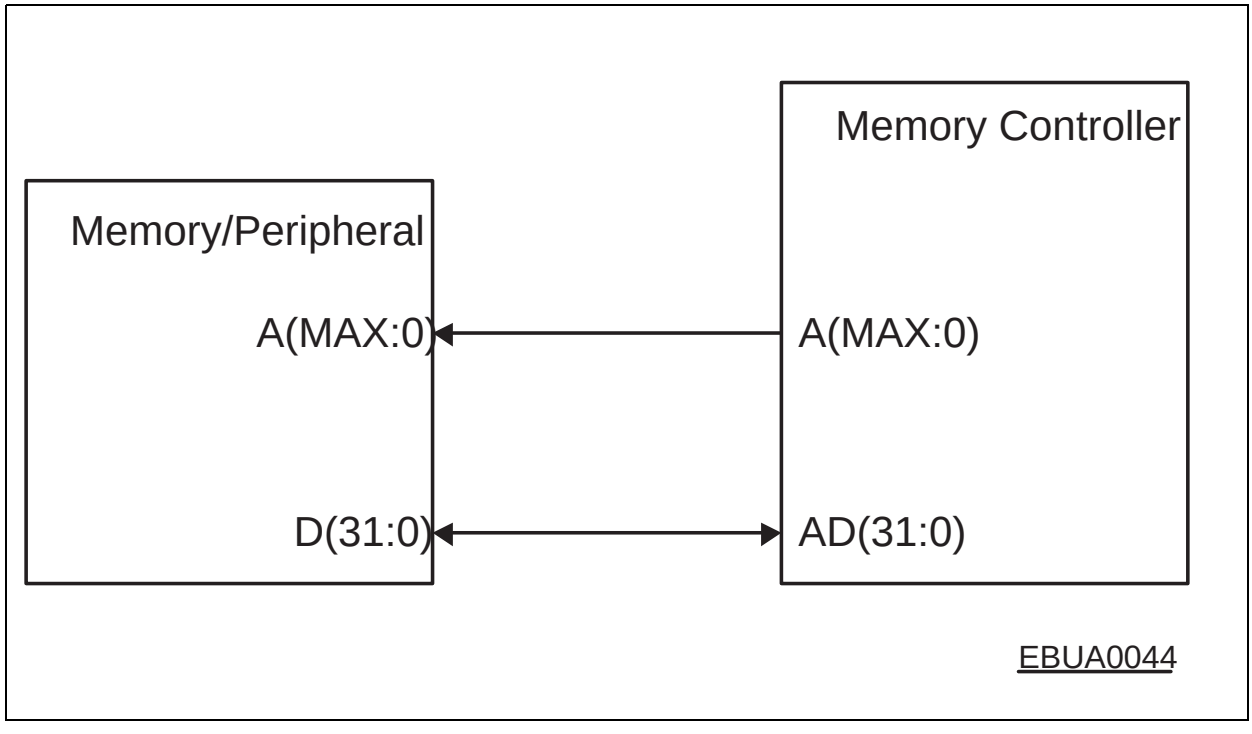


Figure 12-7 Connection of a 32-bit non-Multiplexed Device to Memory Controller

1) This address is pre-aligned according to the bus width as detailed in **Section 12.4.11**.

12.4.7 Address Comparison

Standard Address Comparison Mode

When an Addressing Override Mode is not active, each of the four EBU regions can be programmed for independent base addresses and lengths by bits and bit fields in registers EBU_ADDRSELx.

- Bit REGEN is the enable control of a region. If the region is disabled (REGEN = 0), no address comparison will take place for the region.
- Bit field BASE specifies address bits A[31:12] of region x, where A[31:28] must only point to segments 8, 10, 13, and 14 (see **Table 12-15** on **Page 12-23**).[1)]
- Bit field MASK determines the length of a region. It specifies how many bits of a LMB address must match the contents of the BASE(x) bit field (to a maximum of 15, starting with A[26]). Note that address bits A[31:27] must always match.
- Bit WPROT write protects a region. If the region is protected (WPROT = 1), no address comparison will take place for that region on a write access[2)]
- Bit field ALTSEG determines the number of an alternate segment that can be used for address comparison with A[31:28] (if enabled by ALTENAB = 1).
- Bit ALTENAB determines whether an additional alternate segment number as defined by ALTSEG is used for address comparison.

The address comparison scheme is shown in **Figure 12-8**.

1) There is no hardware lockout preventing other values being written to A[31:28], but in most cases this will result in an inaccessible memory. Enabling a memory region at segment $F8_H$ will result in the memory region conflicting with the registers. The EBU will not work correctly if this is done.

2) The WPROT bit also applies to the read phase of a read modify write, LMB transaction. This is to prevent two possible error conditions. The first would be where the read phase was accepted and the write phase errored. This is an LMB protocol violation. The second would be where the read and write accesses occurred to different memory addresses (if the read took place to a write protected region and there was an unprotected, lower priority, region mapped to the same address space).

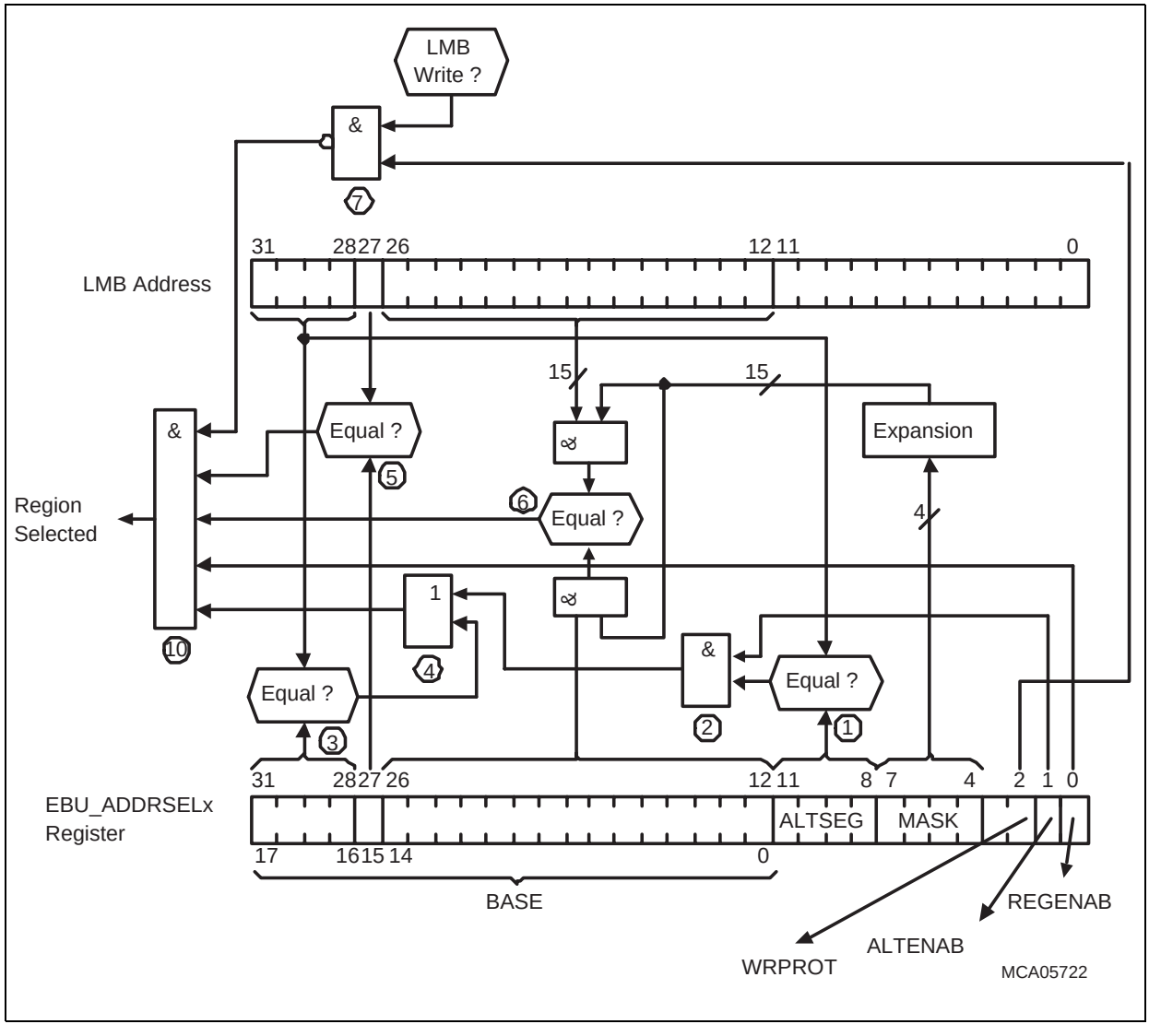


Figure 12-8 Address Comparison to Detect Access to External Region

When the EBU is processing an LMB access, the address is compared in parallel to the parameters of all four regions. The address comparison process for one region is shown in **Figure 12-8**. This process is as follows:

1. The most significant four bits of the LMB address are compared with the ALTSEG bit field (“alternate” segment address). The result of the comparison (1 if equal, otherwise 0) is fed to an AND gate.
2. If ALTENAB = 0, the alternate segment function is disabled and the output of the AND gate is 0. With ALTENAB = 1, the alternate segment function is enabled, and the result of the comparison between ALTSEG and the segment part of the LMB address is fed to an OR gate.

3. The most significant four bits of the LMB address (“main” segment address) are compared to the most significant four bits of the BASE bit field. The result of the comparison (1 if equal, otherwise 0) is fed to the OR gate.
4. The OR gate combines the result of the “main” and “alternate” segment comparisons generates a 1 if the LMB address is in the selected segment(s) for the region, otherwise it generates 0. The output of the OR gate is fed to the final AND gate.
5. Bit 27 of the LMB address is (unconditionally) compared with bit 15 of the BASE bit field. The result of the comparison (1 if equal, otherwise 0) is fed to the final AND gate.
6. The appropriate number of LMB address bits from bit 26 downwards is compared with the corresponding bits from bit field BASE bit 14 downwards. The number of bits used for the comparison is controlled by the MASK bit field. The result of the comparison (1 if the appropriate bits are equal, otherwise 0) is fed to the final AND gate.
7. The NAND gate delivers a 0 if a write is performed to a read-only region (WRPROT=1), and prevents the region from being selected. The output of the NAND gate is fed to the final AND gate.
8. The final AND gate delivers a 1 if a match occurs at the address comparison, and the region x is enabled by REGENAB = 1, and the access is not a write access when the region is defined as read-only access.

This address decoding scheme has the following effects:

- The smallest possible address region is 2^{12} bytes (4 Kbyte)
- The largest possible address region is 2^{27} bytes (128 Mbyte)
- The start address of a region depends on the size of the region. It must be at an address that is a multiple of the size of a region; for example, the smallest region can be placed on any 4-Kbyte boundary, while the largest region can be placed on 8-Mbyte boundaries only.

Table 12-15 shows the possible region sizes and start granularity, as determined by the programming of the MASK bit field. The range of the offset address within such a region is also given.

Table 12-15 EBU Address Regions Size and Start Address Relations

MASK	No. of Address Bits compared to BASE[26:12]	Range of Address Bits compared to BASE[26:12]	Region Size and Start Address Granularity	Range of Offset Address Bits within Region
1111_B	15	A[26:12]	4 Kbyte	A[11:0]
1110_B	14	A[26:13]	8 Kbyte	A[12:0]
1101_B	13	A[26:14]	16 Kbyte	A[13:0]
1100_B	12	A[26:15]	32 Kbyte	A[14:0]
1011_B	11	A[26:16]	64 Kbyte	A[15:0]

Table 12-15 EBU Address Regions Size and Start Address Relations (cont'd)

MASK	No. of Address Bits compared to BASE[26:12]	Range of Address Bits compared to BASE[26:12]	Region Size and Start Address Granularity	Range of Offset Address Bits within Region
1010_B	10	A[26:17]	128 Kbyte	A[16:0]
1001_B	9	A[26:18]	256 Kbyte	A[17:0]
1000_B	8	A[26:19]	512 Kbyte	A[18:0]
0111_B	7	A[26:20]	1 Mbyte	A[19:0]
0110_B	6	A[26:21]	2 Mbyte	A[20:0]
0101_B	5	A[26:22]	4 Mbyte	A[21:0]
0100_B	4	A[26:23]	8 Mbyte	A[22:0]
0011_B	3	A[26:24]	16 Mbyte	A[23:0]
0010_B	2	A[26:25]	32 Mbyte	A[24:0]
0001_B	1	A[26]	64 Mbyte[1)]	A[25:0]
0000_B	0	–	128 Mbyte[1)]	A[26:0]

1) These region size selections do not affect the external address bus of the TC1797 because and A26 is not output at pins (only A[25:1] are available at TC1797 pins).

The EBU uses the four region select outputs from the above scheme, in conjunction with its own address decode logic, to react to LMB accesses as follows:

1. **Address is in the EBU register space:**
   An EBU register access is executed, and in case of a illegal register address, LMB Error Acknowledge is retuned.
2. **Address matches exactly one enabled external region:**
   The requested access is performed to external memory.
3. **Address matches more than one enabled region (overlapping regions):**
   The requested access is performed using the parameters from the region with highest priority. The region with the lowest region number x (region 0) has the highest priority. The emulator region has lowest priority).
4. **Address matches disabled region(s) or no address match:**
   The EBU returns a LMB Error Acknowledge.

When defining mirrored segments, the user is responsible for ensuring that there is no collision. There is no checking mechanism in hardware that ensures that each segment defined (either in BASE[31:28] or ALTSEG[11:8] or both) is exclusive. Therefore, the user must ensure that each mapping from region 0 to 3 and the emulator region does not interfere with any other; otherwise, only the mapping with the highest priority will take effect.

12.4.8 Access Parameter Selection

Selection of the appropriate access parameters is shown in **Figure 12-9**:

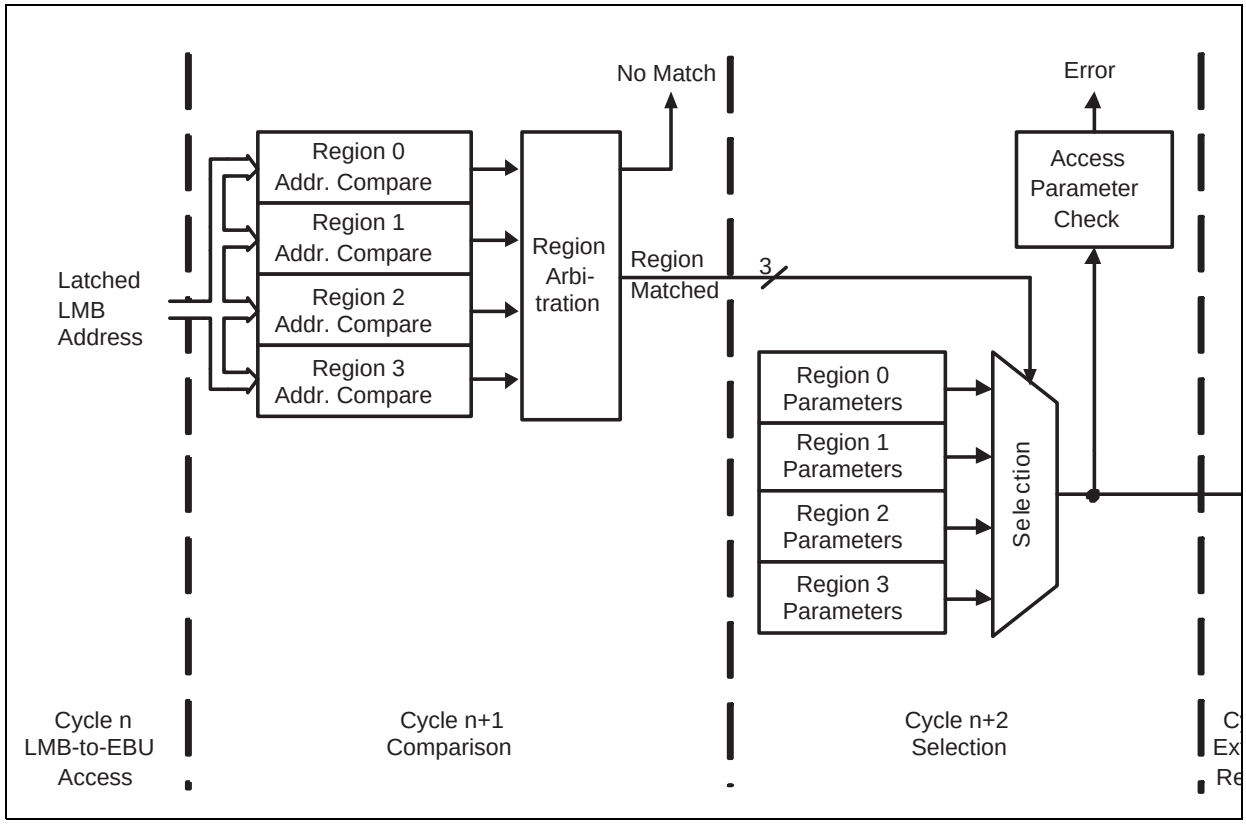


Figure 12-9 Access Parameter Selection Phases

At the end of the LMB address phase (Cycle n), the 32-bit LMB address is made available to the region comparison logic.

In the second phase (Cycle n + 1), the region comparison logic determines whether the LMB address lies within an active region (or regions). If the address is not matched (i.e. is not within an active region), the “No Match” signal causes bits [31:28] or ALTSEG[11:8] to return an LMB Error Acknowledge and discard the access. Otherwise the “Region” signal passes the region index number (in the case of multiple region matches this is the index number of the lowest matching region) to the parameter selection logic.

During the third phase (Cycle n + 2), the parameter selection logic selects the appropriate access cycle parameters. At the end of this cycle, the access parameters associated with the highest priority region are checked for an invalid access (e.g. a write access that does not lie within a defined, writable region). If the access is invalid, the EBU returns an LMB Error Acknowledge and discards the access. Otherwise the parameters are made available to the external bus driver logic.

The next cycle (Cycle n + 3) sees these parameters being passed to the appropriate external access cycle state machine (used to select/initialize the appropriate external access cycle, and also to select the external bus pins to be used for the access cycle).

12.4.9 Programming Sequence Locking

Programming sequences for some Burst Flash devices require that the source of the programming transactions has exclusive access to the external memory device for the duration of the write sequence. If such devices are used and can be accessed by multiple transaction sources then the “Locked Programming Sequence” feature can be enabled via register bit EBU_BUSWCON.LOCKCS.

A programming sequence is considered to start when the processor data port carries out an initial write transaction.

Note: It has been assumed that the processor data port is the only port that will ever attempt programming operations to a NOR flash.

A programming sequence is considered to have ended when the processor data port carries out a subsequent read transaction to a CS which it has locked, or when the processor data port, which has locked CSx, carries out a write to CSy resulting in the locking of CSy, or when an internal fail-safe timeout expires.

A programming sequence lock is aborted if an access is attempted from the processor instruction port to the locked device. An aborted sequence will be flagged by the status bit. LCKABRT, in the EBU_MODCON register. This flag will be cleared by writing 1_B to the LCKABRT field.

Note: This is to prevent a system livelock in the event that the code running the programming sequence is interrupted.

If programming sequence locking is enabled for $\overline{CSx}$, then once a write transaction to $\overline{CSx}$ is accepted from the processor data port, the LMB interface automatically “locks” ownership of the device on $\overline{CSx}$ to that port. Any transaction requests (read or write) to the same $\overline{CSx}$ from other transaction sources are retried until the end of the programming sequence. Accesses to other $\overline{CSy}$ or $\overline{CSz}$ devices from the other initiators will be permitted.

Sequence “locking” takes place in the LMB interface block.

The fail-safe timeout timer defaults to a count of 00_H and is decremented at a frequency of EBU_CLK/16. It commences counting from the preload value every time a write completes. The default value can be changed by writing to the EBU_BUSCON.LOCKTIMEOUT register field.

12.4.10 LMB Bus Width Translation

If the internal access width is wider than the external bus width specified for the selected external region, the internal access is split in the EBU into several external accesses. For

example, if the LMB requests to read a 64-bit word and the external device is only 16-bit wide, the EBU will automatically perform four external 16-bit accesses. When multiple accesses are generated in this way, external bus arbitration is blocked until the multiple access is complete. This means that the EBU remains the owner of the external bus for the duration of the access sequence. The external accesses are performed in ascending LMB address order.

To allow proper bus width translation, the EBU has the capability to re-align data between the external bus and the LMB as shown in **Figure 12-10**.

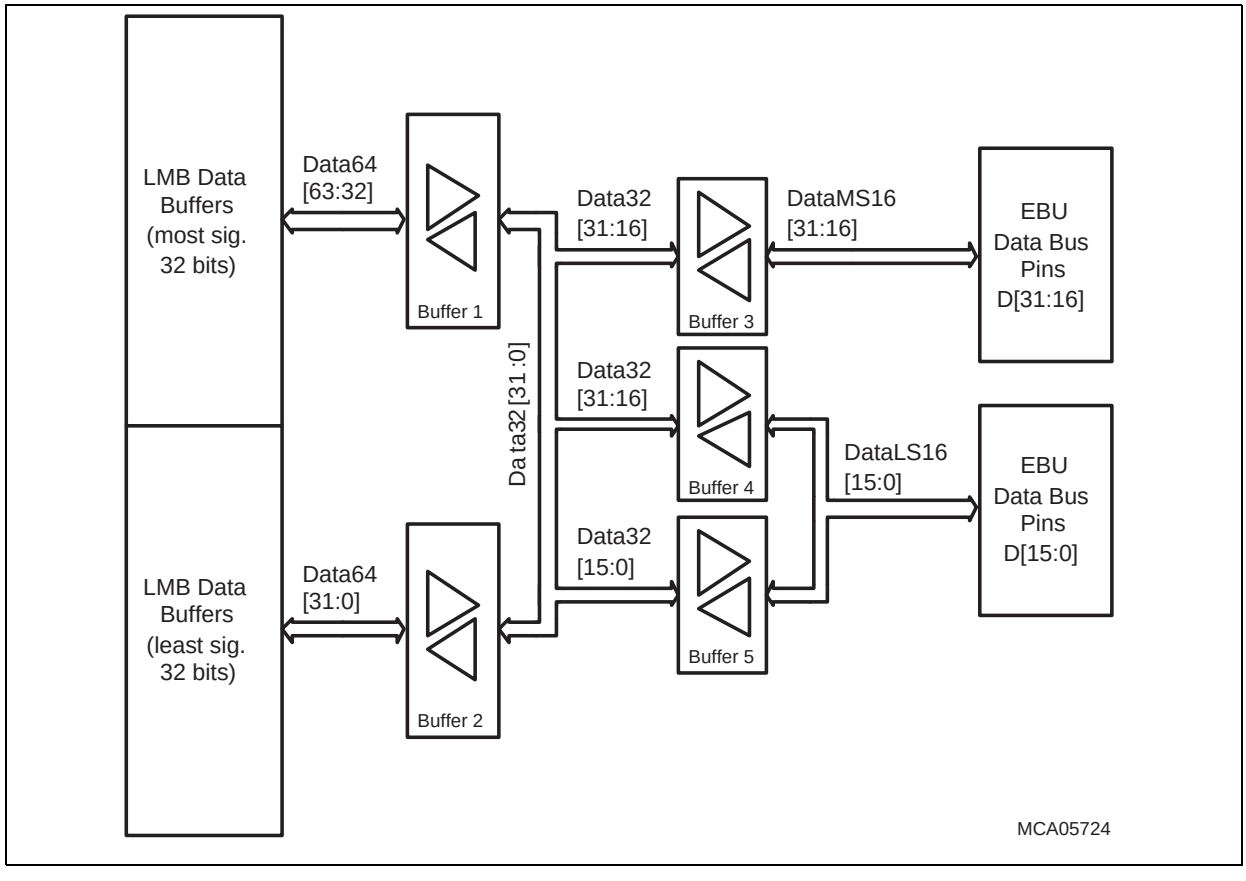


Figure 12-10 LMB to External Bus Data Re-Alignment

- During an access to a 32-bit wide external region, either Buffer 1 or Buffer 2 is enabled (according to bit 2 of the LMB address being accessed) to perform the required 64-bit (LMB) data to 32-bit bus alignment (signified by Data32[31:0] above). To generate a 32-bit access to the external data bus D[31:0], Buffer 3 and Buffer 5 are enabled together.
- During an access to a 16-bit wide external region, either Buffer 1 or Buffer 2 is enabled (according to bit 2 of the LMB address being accessed) and either Buffer 4 or Buffer 5 is enabled (according to bit 1 of the LMB address being accessed). This allows any LMB channel byte pair (i.e. any properly aligned 16-bit data) to be re-aligned to the lower 16 bits of the external data bus D[15:0].

12.4.11 Address Alignment During Bus Accesses

During an external bus access, the EBU will align the internal byte address to generate the appropriate external word or half-word address aligned to the external address pins. The address alignment will be done as follows:

- For 16 bit memory accesses
  - AD[15:0] will be driven with the value from LMBA[16:1] (multiplexed accesses only)
  - A[23:0] will be driven with the value from LMBA[24:1]
- For 32 bit memory accesses
  - AD[15:0] and AD[31:16] will be driven with the value from LMBA[17:2] (accesses to paired 16-bit multiplexed devices only)
  - AD[31:0] will be driven with the right-justified and zero-padded value from LMBA[31:2] (accesses to paired 32-bit multiplexed devices only)
  - A[23:0] will be driven with the value from LMBA[25:2]

12.5 External Bus Arbitration

External bus arbitration is provided to allow the EBU to share its external bus with other master devices. This capability allows other external master devices to obtain ownership of the external bus, and to use the bus to access external devices connected to this bus. The scheme provided by the EBU is compatible with other TriCore and C166 devices and therefore allows the use of such devices as (external bus) masters together with the TC1797.

*Note: In this section, the term “external master” is used to denote a device which is located on the **external** bus and is capable of generating accesses across the external bus (i.e. is capable of driving the external bus). An external master is not able to access units that are located inside the TC1797.*

12.5.1 External Bus Modes

The EBU can operate in two bus modes on the external bus:

- Owner Mode
- Hold Mode

When in Owner Mode, the EBU operates as the master of the external bus. In other words, the EBU drives the external bus as required in order to access devices located on the external bus. While the EBU is in Owner Mode it is not possible for any other master to perform any accesses on the external bus.

During Hold Mode, the EBU tri-states the appropriate signal on the external bus in order to allow another external bus master to perform accesses on the external bus (i.e. to allow another master to drive the various external bus signals without contention with EBU).

12.5.2 Arbitration Signals and Parameters

The arbitration scheme consists of an external bus master that is responsible for controlling the allocation of the external bus. This master is referred to as the “Arbiter” within this document. The other external bus master (termed Participant within this document) requests ownership of the bus, and when necessary, from the Arbiter. The EBU can be programmed to operate either as an Arbiter or as a Participant (see **Page 12-32**). The following three lines are used by the EBU to arbitrate the external bus.

Table 12-16 EBU External Bus Arbitration Signals

Signal	Direction	Function
$\overline{\text{HOLD}}$	In	$\overline{\text{HOLD}}$ is asserted (low) by an external bus master when the external bus master requests to obtain ownership of the external bus from the EBU.
$\overline{\text{HLDA}}$	In/Out[1)]	$\overline{\text{HLDA}}$ is asserted (low) by the Arbiter to signal that the external bus is available for use by the Participant (i.e. the bus is not being used by the Arbiter). $\overline{\text{HLDA}}$ is sampled by the Participant to detect when it may use the external bus.
$\overline{\text{BREQ}}$	Out	$\overline{\text{BREQ}}$ is asserted (low) by the EBU when the EBU requests to obtain ownership of the external bus.

1) The direction of this signal depends upon the mode in which the EBU is operating (see **Page 12-32**).

Two components that are equipped with the EBU arbitration protocol can be directly connected together (without additional external logic) as shown below:

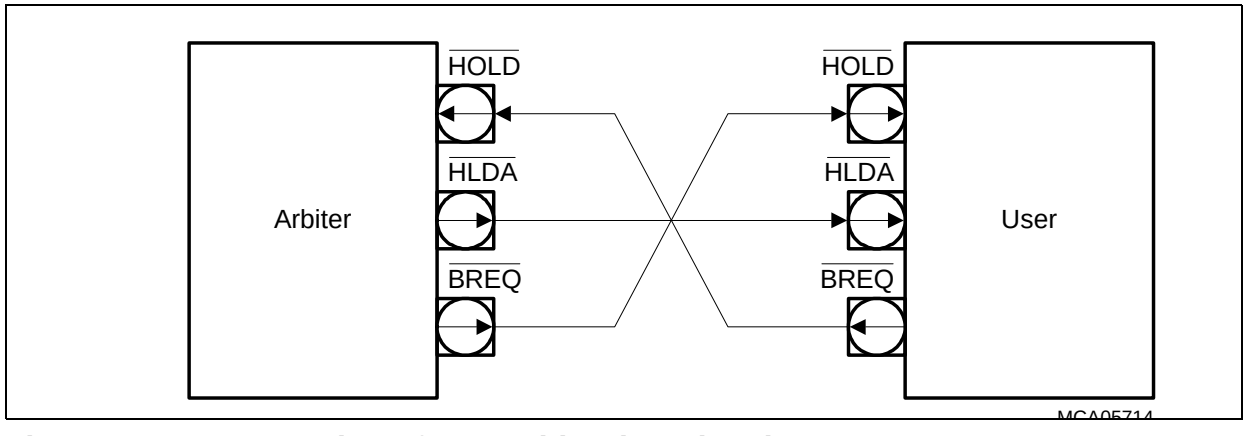


Figure 12-11 Connection of Bus Arbitration Signals

*Note: In the example of **Figure 12-11**, it is possible for the EBU to perform the function or either Arbiter or Participant (or indeed both the Arbiter and Participant may be the EBU).*

Table 12-17 lists the programmable parameters for the external bus arbitration.

Table 12-17 External Bus Arbitration Programmable Parameters

Parameter	Function	Description see
EBU_MODCON.ARBMODE	Arbitration mode selection	Page 12-32
EBU_MODCON.ARBSYNC	Arbitration input signal sampling control	
EBU_MODCON.EXTLOCK	External bus ownership locking control	
EBU_MODCON.TIMEOUTC	External bus time-out control	

12.5.3 Arbitration Modes

The arbitration mode of the EBU can be selected through configuration pins during reset or by programming the EBU_MODCON.ARBMODE bit field (see **Page 12-32**) after reset. Four different modes are available:

- No Bus Mode
- Sole Master Mode
- Arbiter Mode
- Participant Mode

12.5.3.1 No Bus Arbitration Mode

All accesses of the EBU to devices on the external bus are prohibited and will generate a LMB bus error. The EBU operates in Hold Mode all the time. The HOLD and HLDA arbitration inputs are ignored and BREQ arbitration output remains at high (inactive) level.

No Bus Mode is selected by EBU_MODCON.ARBMODE = 00_B.

Note: The EBU can be configured for optimal power saving by entering standby mode (EBU_CLC.DISR=1_B) once ARBMODE has been set to 00_B.

12.5.3.2 Sole Master Arbitration Mode

The EBU is the only master on the external bus; therefore no arbitration is necessary and the EBU has access to the external bus at any time. The EBU operates in Owner Mode all the time. The HOLD arbitration input is ignored, and the HLDA and BREQ arbitration outputs remain at high (inactive) level.

Sole Master Mode is selected by EBU_MODCON.ARBMODE = 11_B.

12.5.3.3 Arbiter Mode Arbitration Mode

The EBU is the default owner of the external bus (e.g. applicable when operating from external memory). Arbitration is performed if an external master (e.g. second TriCore) needs to access the external bus.

The EBU is cooperative in relinquishing ownership of the external bus while operating in Arbiter Mode. When the HOLD input is active, the EBU will generate a “retry” in response to any attempt to access the external bus from the internal LMB. However, the EBU is aggressive in regaining ownership of the external bus while operating in Arbiter Mode. The EBU, having yielded ownership of the bus, will always request return of ownership even if there is no EBU external bus access pending.

Arbiter Mode is selected by EBU_MODCON.ARBMODE = 01_B.

Table 12-18 and **Figure 12-12** show the functionality of the arbitration signals in Arbiter Mode.

Table 12-18 Function of Arbitration Pins in Arbiter Mode

Pin	Type	Pin Function in Arbiter Mode
$\overline{\text{HOLD}}$	In	In Owner Mode (EBU is the owner of the external bus), a low level at $\overline{\text{HOLD}}$ indicates a request for bus ownership from the external master. In Hold Mode (EBU is not the owner of the external bus), a high level at $\overline{\text{HOLD}}$ indicates that the external master has relinquished bus ownership, which causes the EBU to exit Hold Mode.
$\overline{\text{HLDA}}$	Out	While $\overline{\text{HLDA}}$ is high, the EBU is operating in Owner Mode. A high-to-low transition indicates that the EBU has entered Hold Mode and that the external bus is available to the external master. While $\overline{\text{HLDA}}$ is low, the EBU is operating in Hold Mode. A low-to-high transition indicates that the EBU has exited Hold Mode, and has retaken ownership of the external bus.
$\overline{\text{BREQ}}$	Out	$\overline{\text{BREQ}}$ is high during normal operation. The EBU drives $\overline{\text{BREQ}}$ low for two EBU clock cycles after entering Hold Mode (after asserting $\overline{\text{HLDA}}$ low). $\overline{\text{BREQ}}$ returns high one clock cycle after the EBU has exited Hold Mode (after driving $\overline{\text{HLDA}}$ high).

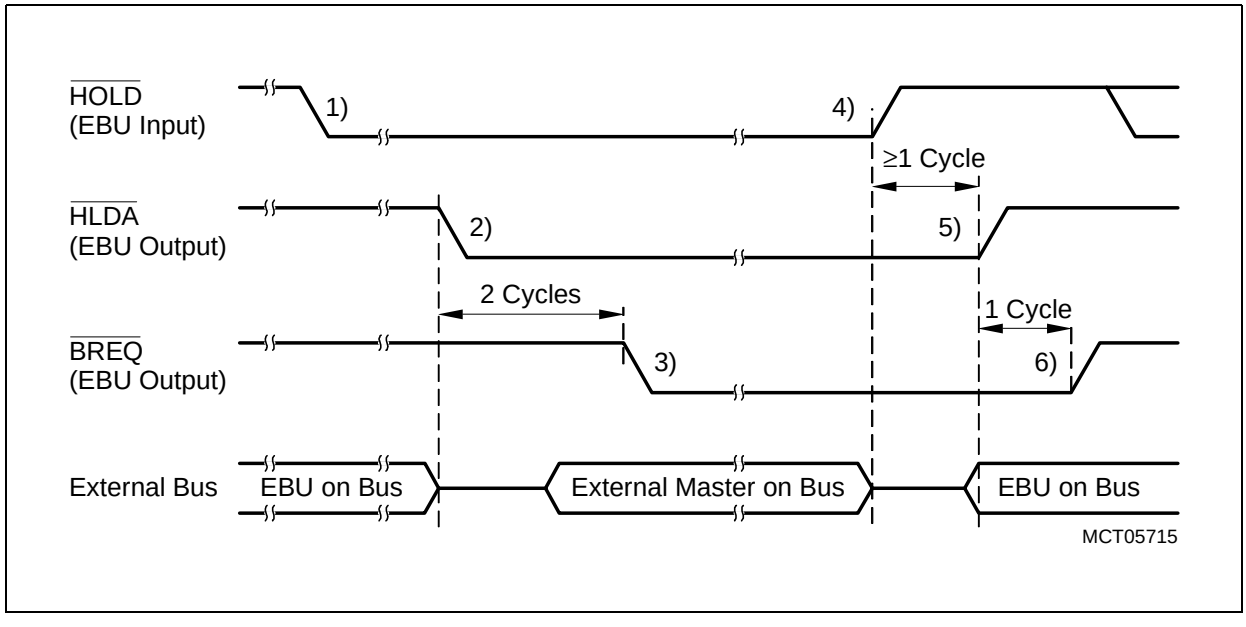


Figure 12-12 Arbitration Sequence with the EBU in Arbiter Mode

In Arbiter Mode, the arbitration sequence starts with the EBU as owner of the external bus.

1. The external master wants to perform an external bus access by asserting a low signal on the HOLD input.
2. When the EBU is able to release bus ownership, it enters Hold Mode by tri-stating its bus interface lines and drives HLDA = 0 to indicate that it has released the bus. At this point, the external master ia allowed to drive the bus.
3. Two clock (EBU_CLK) cycles minimum after issuing HLDA low, the EBU drives BREQ low in order to regain bus ownership. This bus request is issued whether or not the EBU has a pending external bus access. However, the external master will ignore this signal until it has finished its bus access. This scheme assures that the external master can perform at least one complete external bus access.
4. When the external master has completed its access, it tri-states its bus interface and sets HOLD to inactive (high) level to signal that it has released the bus back to the EBU.
5. When the EBU detects that the bus has been released, it returns HLDA to high level and returns to Owner Mode by actively driving the bus interface lines. There is always at least one clock (EBU_CLK) cycle delay from the release of the HOLD input to the EBU driving the bus.
6. Finally, the EBU deactivates the BREQ signal a minimum of one clock (EBU_CLK) cycle after deactivation of HLDA. Now (and not earlier) the external master can generate a new hold request to the EBU.

This sequence assures that the EBU can perform at least one complete bus cycle before it re-enters Hold Mode as a result of a request from the external master.

The conditions that cause change of bus ownership when the EBU is operating in Arbiter Mode are shown in **Figure 12-13**.

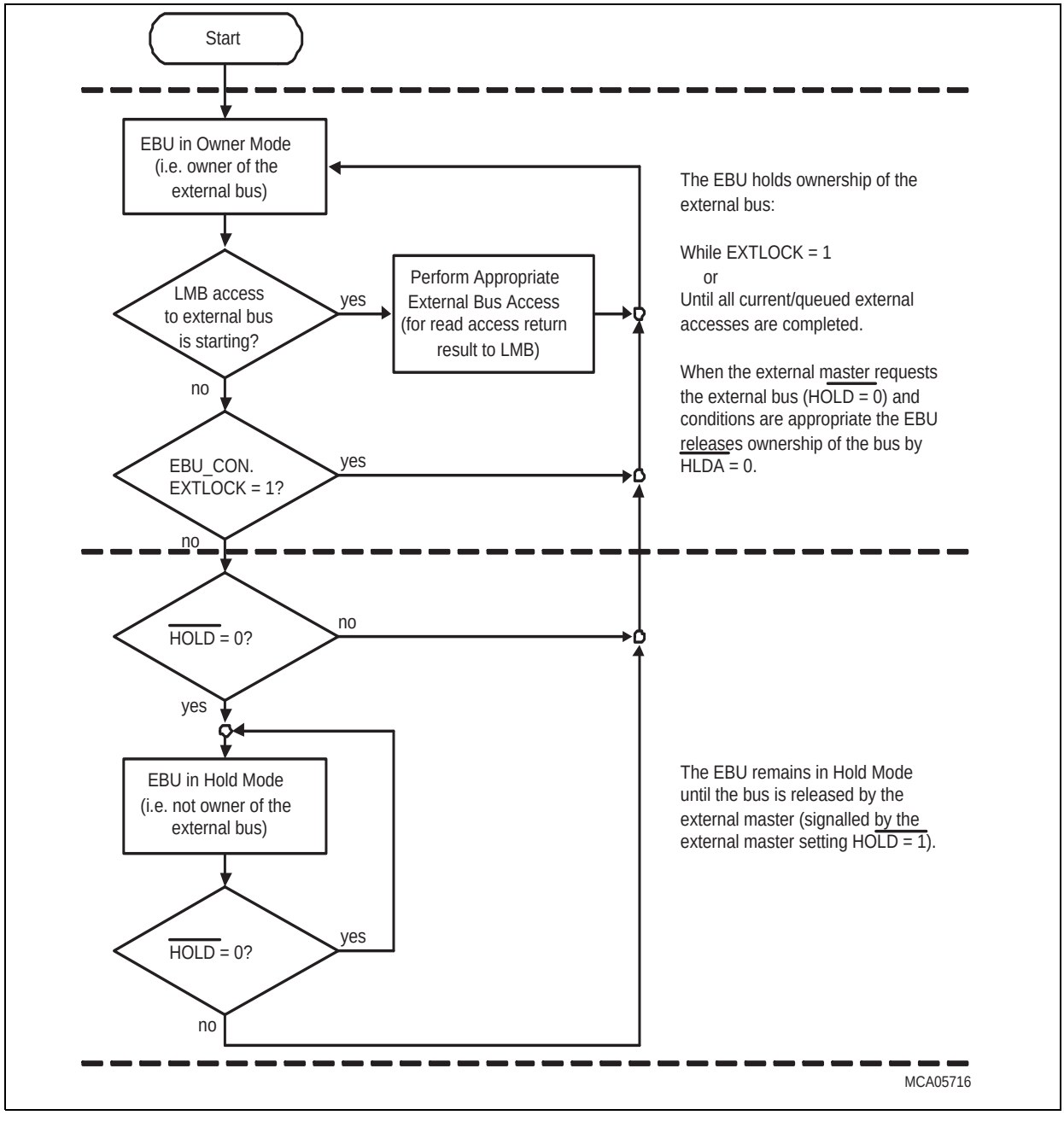


Figure 12-13 Bus Ownership Control in Arbiter Mode

12.5.3.4 “Participant Mode” Arbitration Mode

The EBU tries to gain bus ownership only in case of pending transfers (e.g. when operating from internal memory and performing stores to external memory). While the EBU is not the owner of the external bus (default state), any LMB access to the external bus will be issued with a retry by the EBU. Any such access will, however, cause the EBU to arbitrate for ownership of the external bus.

Once the EBU has gained ownership of the external bus, it will wait until either the occurrence of an external bus access (e.g. the repeat of the request that originally caused the arbitration to occur) or for a programmable time-out (see **Page 12-41**). Once the first access has been completed, the EBU will continue to accept requests from the LMB bus until the external master asserts $\overline{HOLD}$ = 0. After the external master has asserted $\overline{HOLD}$ = 0, the EBU will respond to subsequent LMB accesses to external memory with a retry, and will return ownership of the bus to the external master once any ongoing transaction is complete.

Note: Regardless of the state of the $\overline{HOLD}$ input, the EBU will always perform at least one external bus access (assumed that there is not a time-out) before returning ownership of the bus to the external master.

The use of the arbitration signals in Participant Mode is:

Table 12-19 Function of Arbitration Pins in Participant Mode

Pin	Type	Pin Function in Participant Mode
$\overline{HOLD}$	In	When the EBU is not in Hold Mode ($\overline{HLDA}$ = 0) and has completely taken over control of the external bus, a low level at $\overline{HOLD}$ requests the EBU to return to Hold Mode.
$\overline{HLDA}$	In	When the $\overline{HLDA}$ signal is high, the EBU is in Hold Mode. When the EBU has requested ownership of the bus by a high-to-low transition at $\overline{HLDA}$, the EBU is released from Hold Mode.
$\overline{BREQ}$	Out	$\overline{BREQ}$ remains high as long as the EBU does not need to access the external bus. When the EBU detects that an external access is required, it sets $\overline{BREQ}$ = 0 and waits for signal $\overline{HLDA}$ to become low. When the EBU has completed the external bus access (and has re-entered Hold Mode), it will set $\overline{BREQ}$ = 1 to signal that it has relinquished ownership of the external bus.

Participant Mode arbitration mode is selected by EBU_CON.ARBMODE = 10_B.

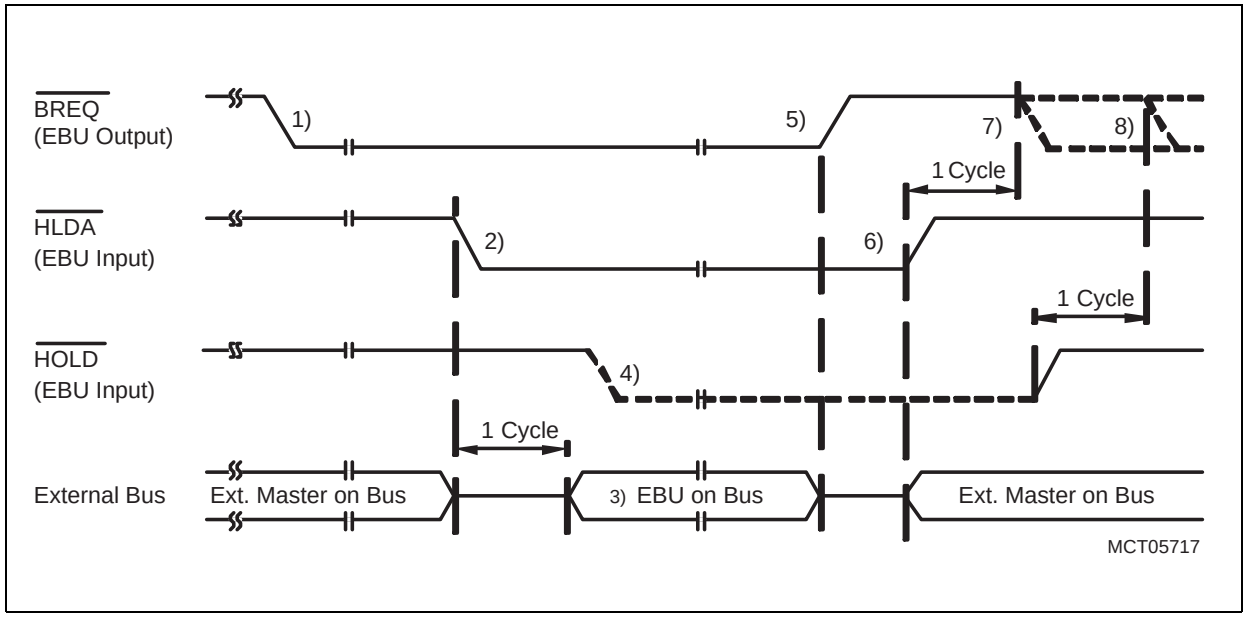


Figure 12-14 Arbitration Sequence with the EBU in Participant Mode

In Participant Mode, the arbitration sequence starts with the EBU in Hold Mode.

1. The EBU detects that it has to perform an external bus access by asserting a low signal on the $\overline{\text{BREQ}}$ output.
2. When the external master is able to release bus ownership, the external master releases the external bus by tri-stating its bus interface lines and drives the $\overline{\text{HLDA}}$ = 0.
3. At least one clock (EBU_CLK) cycle after detecting $\overline{\text{HLDA}}$ = 0, the EBU will start to drive the external bus.
4. When the EBU is in Owner Mode, the external master may optionally drive $\overline{\text{HOLD}}$ = 0 to signal that it wants to regain ownership of the external bus.
5. When the criteria are met for the EBU to release the bus ownership, the EBU enters Hold Mode and drives $\overline{\text{BREQ}}$ = 1 output high to signal that it has released the bus.
6. When the external master detects that the EBU has released the bus ($\overline{\text{BREQ}}$ = 1), it returns $\overline{\text{HLDA}}$ to high level and takes ownership of the external bus.
7. The EBU will not request ownership of the external bus again ($\overline{\text{BREQ}}$ = 0) at least one clock (EBU_CLK) cycle after $\overline{\text{HLDA}}$ has been driven high).
8. In Owner Mode, the EBU will not request ownership of the external bus $\overline{\text{BREQ}}$ = 0) for at least one clock (EBU_CLK) cycle after its $\overline{\text{HOLD}}$ input has been driven high.

The conditions that cause change of bus ownership when the EBU is operating in Participant Mode is shown in **Figure 12-13**.

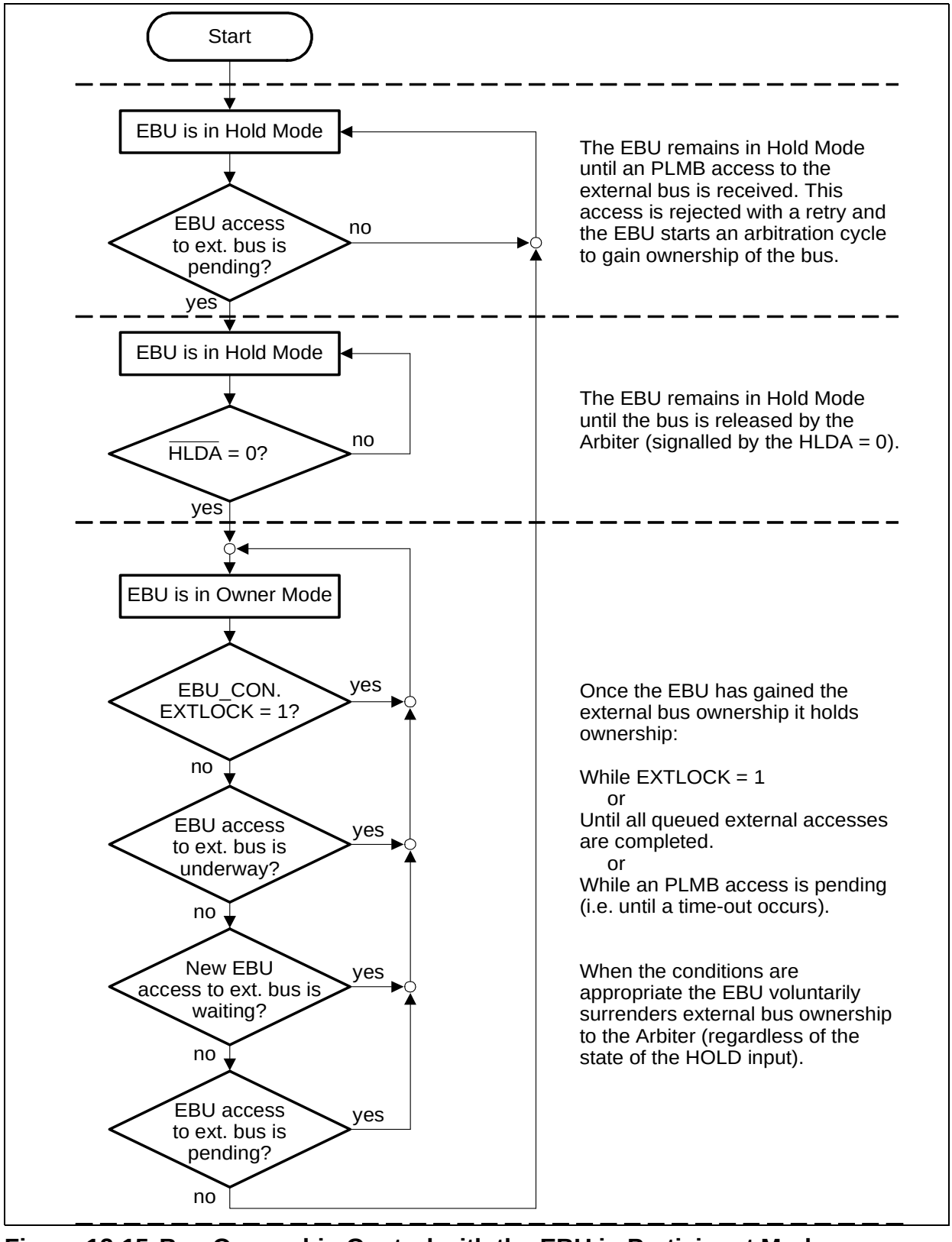


Figure 12-15 Bus Ownership Control with the EBU in Participant Mode

12.5.4 Arbitration Input Signal Sampling

The sampling of the arbitration inputs can be programmed for two modes:

- Synchronous Arbitration
- Asynchronous Arbitration

When synchronous arbitration signal sampling is selected (ARBSYNC = 0), the arbitration input signals are sampled and evaluated in the same clock cycle. This mode provides the least overhead during arbitration (i.e. when changing bus ownership). The disadvantage is that the input signals must adhere to setup and hold times with respect to EBU_CLK to prevent the propagation of meta-stable signals in the EBU.

When asynchronous arbitration signal sampling is selected (ARBSYNC = 1), the arbitration signals are sampled and then fed to an additional latch to be evaluated in the cycle following that in which they were sampled. This provides the EBU with good immunity to signals changing state at or around the time at which they are sampled. The disadvantage is the introduction of additional latency during arbitration (i.e. when changing bus ownership).

12.5.5 Locking the External Bus

The external bus can be locked to allow the EBU to perform uninterrupted sequences of external bus accesses. The EBU allows two methods of locking the external bus:

- Locked LMB accesses
- Lock bit EXTLOCK

When the EBU has ownership of the external bus and is performing external bus accesses in response to a locked LMB access sequence, the ownership of the external bus will not be relinquished until the locked LMB access sequence has been completed.

When lock bit EXTLOCK = 1, the EBU will hold the ownership of the external bus until EXTLOCK is subsequently cleared. If EXTLOCK is written to 1 while the EBU is the owner of the external bus, the EBU is immediately prevented from responding to requests for the external bus until EXTLOCK is cleared[1)]. If EXTLOCK is written to 1 while the EBU is not the owner of the external bus, the EBU will immediately attempt to gain ownership. When the EBU gains the ownership of the external bus the next time, the external master is prevented from regaining ownership of the external bus until EXTLOCK is again cleared.

Note: There is no time-out mechanism available for the EXTLOCK bit. When the EBU is owner of the external bus with EXTLOCK bit set, the external master will remain locked off the bus until the EXTLOCK bit is cleared by software.

1) Requests for the external bus already pending when EXTLOCK is set will not be cancelled so the EBU can give up control of the external bus after EXTLOCK is set provided that the request occurs before the EXTLOCK bit is set.

12.5.6 Reaction to an LMB Access to the External Bus

The reaction of the memory controller to an external bus request from an LMB master is controlled as shown in **Figure 12-16**.

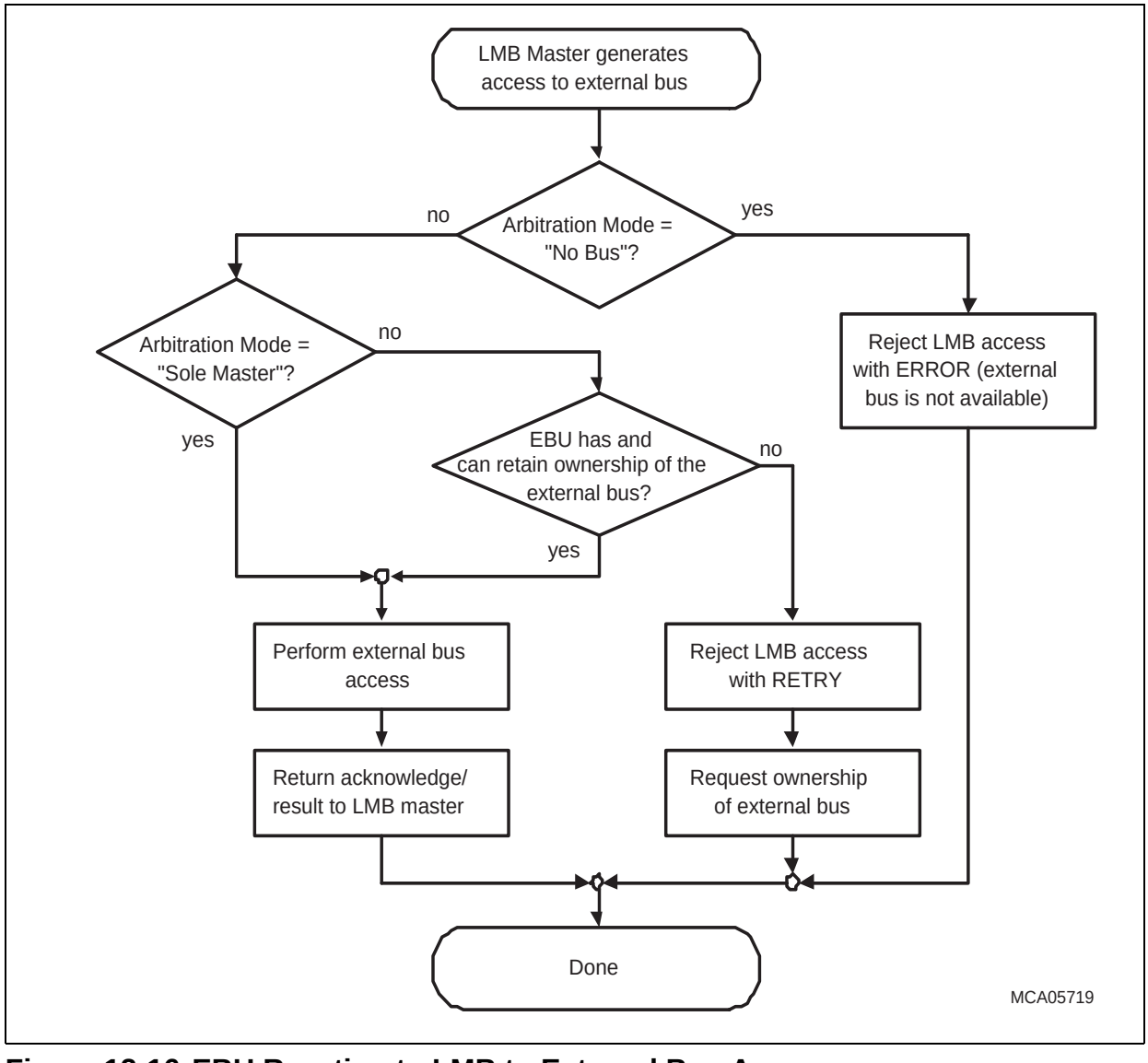


Figure 12-16 EBU Reaction to LMB to External Bus Access

If the EBU is operating in No Bus Mode, it is not possible for an LMB master to access the external bus. For this reason, the EBU generates an LMB error whenever an attempt is made to access the external bus while in No Bus Mode.

If the EBU is operating in Sole Master Mode, it has access to the external bus at all times and as a result it is possible for the EBU to immediately perform the required external bus access.

If the EBU is operating in Arbiter or Participant Modes and receives a request for an external access from an LMB when it is not the owner of the external bus (or is not able to retain ownership of the bus), the request is rejected with a “retry”. As shown in **Figure 12-16**, this event also triggers the EBU to arbitrate with the external master in order to attempt to gain ownership of the external bus so that the request can be serviced when it is re-submitted by the master. This strategy ensures that the LMB remains available while the EBU arbitrates for the external bus.

12.5.6.1 Pending Access Time-Out

The strategy of issuing a retry (when the EBU is not the owner of the external bus) as described in the previous section may result in the occurrence of a time-out condition.

When a LMB master issues a request for an external bus access that is rejected by the EBU with a retry (in order to retain LMB availability) and arbitration for external bus ownership is started, the external bus ownership is given once to the EBU. Now the EBU waits until the next external bus access occurs on the LMB. If the requesting LMB master or any other LMB master subsequently performs no external bus accesses (e.g. fails to re-submit the original access request), the EBU has ownership of the bus for an indefinite time and it would become impossible for an external master to access the external bus.

To avoid such a bus-locking condition, the EBU contains a time-out mechanism. When the EBU has gained ownership of the external bus, it will retain ownership only until a LMB-to-external bus access occurs or a programmable number of EBU_CLK clock cycles has elapsed. If one of these conditions has occurred, the pending access is cancelled and the EBU will continue to arbitrate the external bus in the normal fashion. The desired time-out time (number of EBU_CLK cycles) is programmed using bit field TIMEOUTC. The time-out value can be in the range 1×8 up to 255×8 EBU_CLK clock cycles.

12.6 Start-Up/Boot Process

The EBU can start up in one of two modes after a reset or boot operation.

Table 12-20 EBUT13L-AB Start-Up Modes

Start-Up Modes	Arbitration Mode
Disabled EBU is disabled	No Bus Mode
External Boot Mode Region 0 device characteristics are auto-configured by performing an external Boot Configuration Value fetch from a “Standard” asynchronous device (see **Page 12-42**).	Arbiter Mode

12.6.1 Disabled

The EBU will come up with access to the external bus disabled after reset (i.e. no access from LMB to external memory is possible without EBU re-configuration).

12.6.2 External Boot Mode

The External Boot Mode of the EBU allows the EBU to boot (i.e. run all start-up code) from external memory. Immediately after reset a system may have no knowledge as to the type of memory connected to the external bus. When External Boot Mode is selected, the EBU will automatically read a 32-bit Boot Configuration Value from an external memory (connected to $\overline{CS0}$, chip select region 0). The Boot Configuration Value in the external boot memory makes it possible to initialize the EBU with appropriate configuration values for the external boot memory. These configuration values will, in turn, be used for the subsequent read accesses from the external boot memory (i.e. instruction fetches).

The boot configuration fetch can be triggered using two methods:

- Setting the SYSCON.SETEXTBEN bit and then resetting the EBU. In this case all accesses to the EBU will be retried while configuration fetch is in progress
- Writing 1_B to the EBU_EXTBOOT.EBUCFG bit before the end of system boot. In this case the EBU_EXTBOOT.CFGEND bit should be polled to determine when the fetch has ended and the registers updated.

In both cases the EBU_EXTBOOT.CFGERR bit will be set if the configuration word fetch fails to retrieve valid data

12.6.2.1 Boot Process

If External Boot Mode is selected, the EBU will perform one external bus read access to the dedicated address 000004_H[1)] of the memory device attached to chip select line $\overline{CS0}$.

The data read by this read access is used to configure the EBU with parameters (see Page 12-44). The boot read access itself is performed as an asynchronous access cycle with all timing parameters set to their maximum values. This access scheme supports ROMs, EPROMs or Flash memories with both separate address and data connections and also multiplexed address and data connections. Figure 12-17 shows a timing example of booting from a standard demultiplexed asynchronous memory device.

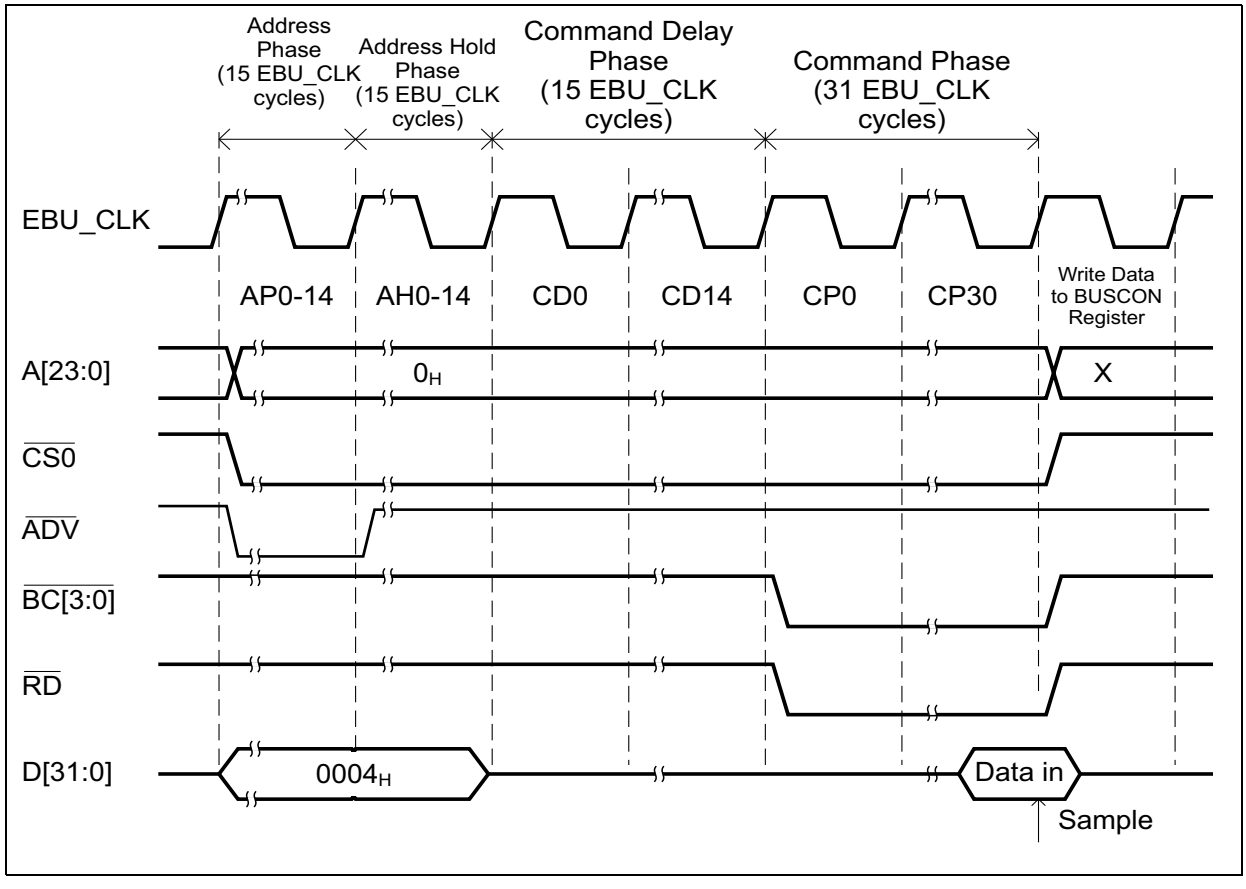


Figure 12-17 Boot Read Access Cycle

When a configuration fetch is triggered, the EBU waits 256 EBU_CLK clock cycles until the access is initiated, as shown in Figure 12-17. This gap is inserted to fulfill the recovery times needed by external synchronous devices such as Flash ROMs. During the read access, the maximum number of programmable EBU_CLK cycles is inserted (WAITRDC = 31), and the evaluation of the WAIT signal is inhibited.

Note: The boot memory must be connected to chip select line CS0. The EBU assumes that the boot memory is 32-bits wide and, therefore, always reads a 32-bit configuration word at the "Data in" point shown above.

1) This is the address presented on the address pins of the EBU and as such is a word address. As the reset value of BUSRCONx.PORTW is 11_H (32 bit) and the external bus address is automatically aligned, this translates to a byte address of 000010_H.

*Note: If $FFFF_H$ is returned as on bits 15 downto 0 of the configuration word during the boot read cycle (e.g. by reading the configuration word from an erased external boot memory device), the arbitration mode is set to No Bus Mode (ARBMODE = 00_B, see **"No Bus Arbitration Mode" on Page 12-32**). However since a value for the BCGEN field of 11_B is not legal, at least one bit of the fetched word will be 0_B for valid data*

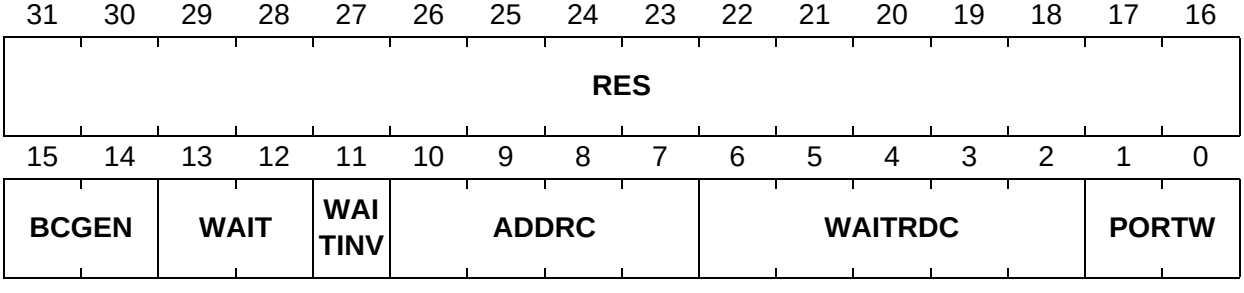
Booting from an External Memory Device

After reset, the memory controller registers are configured for accesses to a 32-bit, multiplexed, asynchronous memory. These settings also allow booting from a non-multiplexed memory device as the dedicated address pins will also be driven.

12.6.2.2 Boot Configuration Value

The EBU supports boot operation from 32-bit wide memories. The format of the Boot Configuration Value is as follows (Bits 31 to 16 are reserved for future expansion):

Boot Configuration Value



Field	Bits	Description
PORTW	1:0	**Port Width** 00_B external memory is 16 bit 01_B external memory is 16 bit 10_B external memory is two 16 bit memories in parallel to make a 32 bit memory 11_B external memory is 32 bit
WAITRDC	[6:2]	**Number of Wait States for Read Accesses** Loaded into EBU_BUSRAP0.WAITRDC.
ADDRC	[10:7]	**Number of Cycles in the Address Phase** Loaded into EBU_BUSRAP0.ADDRC.

Field	Bits	Description
WAITINV	11	**WAIT Input Polarity Control** Loaded into EBU_BUSRCON0.WAITINV.
WAIT	[13:12]	**External Wait State Control** Loaded into EBU_BUSRCON0.WAIT.
BCGEN	[15:14]	**Byte Control Signal Control** Loaded into EBU_BUSRCON0.BCGEN.
RES	[31:16]	**Reserved** This bit field is reserved for future use. In the TC1797, RES should always be set to 0_H.

12.7 Clocking Strategy and Local Clock Generation

The memory controller is designed to have a flexible internal clocking strategy to allow the power dissipation of the core to be minimised

12.7.1 Local Clock Divider

A local divider can be used to reduce the frequency of the clock signal used to drive the core logic of the memory controller (EBU_CLK). The divider can be programmed for divide ratios of 1:1, 2:1, 3:1 or 4:1 using the EBUCLC.EBUDIV field.

The purposes of the divider is to allow the memory controller core to operate synchronously at an integer divide ratio of the LMB clock.

As the local clock division circuit uses pulse swallowing to generate EBU_CLK, the duty cycle will be distorted when a ratio other than 1:1 is selected. This will affect the timing of signals generated using the negative edge of the internal clock and they will be delayed by half an LMB_CLK period from the rising edge and not half an internal clock period.

Clock ratios can be switched dynamically while the memory controller continues to process accesses to external memory (provided that the register settings for the memory are valid for both clock frequencies). To achieve this, the memory controller will wait for running accesses to complete and the retry incoming accesses until the new clock ratio is selected and stable. Once the new ratio is selected, the EBU_CLC.DIVACK field will be updated to show the new ratio.

12.7.2 Standby Mode

The standby mode is enabled by writing 1_B to the EBU_CLC.DISR field.

Once the bit is set, the memory controller will wait for any running accesses to finish and will then disable the clock to the core logic.

Exit from standby mode is triggered by writing 0_B to the EBU_CLC.DISR field

Alternatively, once in standby mode, any valid access to external memory or memory controller registers arriving on the LMB interface will trigger an automatic exit from standby mode to service the access request. This condition may also prevent standby mode being entered at all depending upon when the new access arrives at the interface

Note: Once a pending access has triggered an exit from standby mode, the memory controller will not return to standby mode until another rising edge is detected on the EBU_CLC.DISR bit.

An automatic exit from standby mode will not clear the EBU_CLC.DISR field. If automatic exit is used, then the EBU_CLC.DISR field will have to be cleared by an explicit write of 0_B before standby mode can be used again.

12.8 LMB Data Buffering

The data for all LMB writes are “posted” into a buffer in the LMB interface before the access is passed to the state machine blocks for execution.

12.9 Standard Access Phases

Accesses to asynchronous devices are composed of a number of standard access phases (according to the type of device and the type of access).

There are six access phases defined:

- Address Phase AP (mandatory for read and write cycles of both device types)
- Command Delay Phase CD (optional)
- Command Phase CP (mandatory)
- Data Hold Phase DH (optional, only applies to write cycles)
- Recovery Phase RP (optional)

Throughout the remainder of this document, a short-hand notation is adopted to represent any clock cycle in any phase. This notation consists of two or three letters followed by a number. The letters identify the access phase within which the clock cycle is located (e.g. AP for Address Phase). The number denotes the number of EBU_CLK clock cycles within the phase (i.e. 1 = first, etc.). In the case of delays that can be extended by external control inputs the lower case letters “e” and “i” are inserted following the two letter phase identifier to differentiate between internally (“i”) and externally (“e”) generated delays. For example, AP2 identifies the second clock in the Address Phase. CPe3 identifies the third clock in the Command Phase which is being extended by external wait-states.

12.9.1 Address Phase (AP)

The Address Phase is mandatory. It always consists of at least one or more EBU_CLK cycles. The phase can be optionally extended to accommodate slower devices.

At the start of the Address Phase, the EBU:

- Selects the device to be accessed by asserting the appropriate $\overline{CSx}$ signal,
- Issues the address which is to be accessed on the address bus,
- Asserts the $\overline{ADV}$ signal low,[1)]
- Asserts the appropriate $\overline{BCx}$ signals if these are programmed to be asserted with the $\overline{CSx}$ signal,
- Asserts the MR/$\overline{W}$ signal according to the type of access to be performed (low in the case of a write access, not used in burst read cycles). This level is retained until the start of the next Address Phase.
- At the end of the Address Phase the EBU returns the $\overline{ADV}$ signal to high.

The length (number of EBU_CLK cycles) of the Address Phase is programmed via the EBU_BUSAPx.ADDRC bit field parameter.

12.9.2 Address Hold Phase (AH)

The Address Hold Phase is optional. It consists of zero or more LMB_CLK cycles. It is intended to provide hold time for the multiplexed address bits after the $\overline{ADV}$ signal has returned to the inactive state.

At the end of the address hold phase, the multiplexed address can be removed from the bus:

- During a read access, the multiplexed address/data bus can return to the high impedance condition to allow the read data to be driven by the external memory
- During a write access, the write data can be driven onto the multiplexed address/data bus

12.9.3 Command Delay Phase (CD)

The Command Delay phase is optional. This means that it can also be programmed for a length of zero EBU_CLK clock cycles. The CD phase allows for the insertion of a delay between Address Phase (or optional Address Hold phase) and Command Phase(s). This phase accommodates devices that are not fast enough to receive commands immediately after getting the address or multiplexed devices which require a bus turnaround delay on reads.

The length (number of EBU_CLK cycles) of the Command Delay phase is programmed via the EBU_BUSAPx.CMDDELAY bit field. This parameter makes it possible to select between zero to seven Command Delay phases.

1) If an active high, ALE, signal is required, the polarity of the $\overline{ADV}$ output can be inverted by setting the ALE filed of the EBU_MODCON register.

12.9.4 Command Phase (CP)

The Command Phase is mandatory. It always consists of at least one or more EBU_CLK cycles. The phase can optionally be extended to accommodate slower devices.

The length (number of EBU_CLK cycles) of the Command Phase is separately programmable for read and write accesses. Bit field EBU_BUSAPx.WAITRDC determines the basic length of Command Phases during read cycles and bit field EBU_BUSAPx.WAITWRC determines the basic length of Command Phases during write cycles.

The calculation of the number of Command Phase EBU_CLK cycles is described in the following table:

Additionally, when accessing asynchronous devices, a Command Phase can also be extended externally using the WAIT signal when the region being accessed is programmed for external command delay control via bit EBU_BUSCONx.WAIT or EBU_EMUBC.WAIT.

The Command Phase is further subdivided into:

- CPi (= internally-programmed Command Phase)
- CPe (= externally-prolonged Command Phase, i.e. prolonged by the assertion of the WAIT signal).

At the start of the Command Phase, the EBU:

- Asserts the appropriate control signal RD or RD/WR low according to the access type (read or write),
- Issues the data to be written on the data bus (in the case of a write cycle),
- Asserts the appropriate BCx low (in the case where BCx is programmed to be asserted with the RD or RD/WR signals).

At the end of the Command Phase during an asynchronous access, the EBU:

- Returns the appropriate control signal RD or RD/WR high according to the type of access type (read or write),
- Latches the data from the data bus AD[15:0] (in the case of a read cycle),
- Returns the appropriate BCx high (in the case where BCx is programmed to be asserted with the RD or RD/WR signals).
- Returns the CS high and removes the write data (if no Data Hold Phase is programmed).

12.9.5 Data Hold Phase (DH)

The Data Hold phase is optional. This means that it can also be programmed for a length of zero EBU_CLK clock cycles. Furthermore, it is only available for write accesses. The Data Hold phase extends the amount of time for which data is still held on the bus after the rising edge of the RD/WR signal occurred. The Data Hold phase is used to accommodate external devices that require a data hold time after the rising edge of the

RD/$\overline{WR}$ signal. The length (number of EBU_CLK cycles) of the Data Hold phase is programmed via the EBU_BUSAPx.DATAC bit field.

If a synchronous access is programmed to use the Data Hold Phase, it will be inserted after the final Burst Phase has completed.

At the end of the Data Hold Phase, the EBU will:

- Remove the write data from the data bus (in the case of a write cycle),
- Return the $\overline{CS}$ high.

Note: If an attached asynchronous memory device latches write data on the rising edge of any of the control lines; $\overline{CS}$, RD/$\overline{WR}$ or $\overline{BC}$, then a Data Hold phase will be required to ensure that data is latched correctly.

12.9.6 Burst Phase (BP)

The Burst Phase is mandatory during burst accesses. At the end of the Burst Phase the EBU reads data from the data bus or updates the write data. During a burst access, Burst Phases are repeated as many times as required in order to read or write the required amount of data from or to the external memory device.

At the start of the first Burst Phase during a burst read access, the EBU:

- Drives the $\overline{BAA}$ signal low to cause the Burst Flash device to advance the address with each subsequent BFCLKO positive edge.

The first burst phase of an access will always start on arising edge of BFCLKO. If necessary, the length of the previous phase will be extended to ensure that this happens.

At the end of the last Burst Phase during a burst read access, the EBU:

- Returns the $\overline{BAA}$ signal high,
- Returns the $\overline{CS}$x signal high,
- Returns the $\overline{RD}$ signal high.

During accesses to Burst Flash devices the length of the Burst Phase must be programmed such that the end of the Burst Phase always coincides with a positive edge of the appropriate BFCLKO (Burst Flash Clock) signal.

A Burst Phase is always at least one clock cycle in length. When BUSRCONx.agen is not equal to 1101_B, Demuxed Burst Type External Memory (DDR flash protocol), then the length of each Burst Phase (i.e. the number of EBU_CLK cycles) is derived from the value of the EXTCLOCK and EXTDATA fields in the EBU_BUSAPx register. The length of the burst phase will be either be:

- one period of BFCLKO if EXTDATA is 00_B,
- two periods of BFCLKO if EXTDATA is 01_B.
- four periods of BFCLKO if EXTDATA is 10_B.
- eight periods of BFCLKO if EXTDATA is 11_B.

If EBU_BUSCONx.agen is equal to 1101_B, then the length of the burst phase will be controlled by the EXTDATA field only and the burst phase length will be equal to

EXTDATA+1. This allows the external clock period to be an integer multiple of the burst phase length. This will allow support for devices which return data on both edges of the device clock. (e.g. setting EXTDATA to 00_B and EXTCLOCK to 01_B will support a DDR style flash device)

12.9.7 Recovery Phase (RP)

The Recovery Phase is optional (although for access types which would cause a bus contention a single cycle of recovery is normally forced by the memory controller logic). This means that it can also be programmed for a length of zero EBU_CLK clock cycles. This phase allows the insertion of a delay following an external bus access that delays the start of the Address Phase for the next external bus access. This permits flexible adjustment of the delay between accesses to the various external devices. The following individually programmable delays are provided on a region by region basis for the following conditions:

- Bit fields EBU_BUSAPx.RDRECOVC determine the basic length of the Recovery Phase after a read access.
- Bit fields EBU_BUSAPx.WRRECOVC determine the basic length of the Recovery Phase after a write access.
- Bit fields EBU_BUSAPx.DTACS determine the length (basic number of EBU_CLK clock cycles) of the Recovery Phase after a read/write access of one region that is followed by a read/write access of another region or a read to one region is followed by a write to the same region (BUSRAPx.DTACS) or a write to one region is followed by a read to the same region (BUSWAPx.DTACS).

The EBU implements a “highest wins” algorithm to ensure that the longest applicable recovery delay is always used between consecutive accesses to the external bus. **Table 12-21** shows the scheme for determining this delay for all possible circumstances. For example, if a read access to a region associated with $\overline{CS1}$ is followed by a write to a region associated with $\overline{CS2}$, the delay will be the highest of BUSRAP1.DTACS and BUSRAP1.RDRECOVC. In this case, if BUSRAP1.DTACS is greater than BUSRAP1.RDRECOVC, then the number of recovery cycles between the two accesses is BUSRAP1.DTACS clock cycles (minimum).

Table 12-21 Parameters for Recovery Phase

Case			Parameter(s) used to calculate "Highest Wins" Recovery Phase
Region	**Current Access**	**Next Access**	
Same $\overline{CS}$n	Read	Read	RDRECOVC
	Write	Write	WRRECOVC
	Read	Write	BUSRAPx.DTACS, RDRECOVC
	Write	Read	BUSWAPx.DTACS, WRRECOVC
Different $\overline{CS}$n	Read	Read	BUSRAPx.DTACS, RDRECOVC
	Write	Write	BUSWAPx.DTACS, WRRECOVC
	Read	Write	BUSRAPx.DTACS, RDRECOVC
	Write	Read	BUSWAPx.DTACS, WRRECOVC

12.10 Asynchronous Read/Write Accesses

Asynchronous read/write access of the EBU support the following features:

- EBU_CLK clock-synchronous signal generation
- Support for 16-bit and 32-bit bus width
  Performing an LMB access with a data width greater than that of the external device automatically triggers a sequence of the appropriate number of external accesses to match the LMB access width.
- Demultiplexed address/data lines
- Programmable access parameters
  – Internal control of command delay cycles
  – External and/or internal control of wait states
  – Variable data hold cycles for write operation (to allow flexible hold time adjustment)
  – Variable inactive/recovery cycles when:
    Switching between different memory regions (CS),
    Switching between read and write operations,
    After each read cycle,
    After each write cycle.

Software driver routines are required in order to support Nand Flash devices using asynchronous device accesses. A single Nand Flash access sequence is performed by generating the appropriate sequence of discrete asynchronous device accesses in software.

The EBU does not provide support 8-bit bus width. When 8-bit SRAM devices are used, they should be used in pairs to implement a 16-bit wide memory region.

12.10.1 Signal List

The following signals of the EBU are used for asynchronous accesses:

Table 12-22 Asynchronous Mode Signal List

Signal/Pin	Type	Function
AD[15:0]	O	Address/Data bus lines 0-15
A[23:0]	O	Address bus lines 0-23
$\overline{\text{CS}}$[3:0]	O	Chip select 0-3
$\overline{\text{RD}}$	O	Read control line
RD/$\overline{\text{WR}}$	O	Write control line
$\overline{\text{BC}}$[3:0]	O	Byte control lines 0-3
$\overline{\text{WAIT}}$	I	Wait input

12.10.2 Standard Asynchronous Access Phases

Accesses to asynchronous devices are composed of a subset of the standard access phases which are detailed in **Section 12.9**. The standard access phases for asynchronous devices are:

- AP: Address Phase (compulsory - see **Page 12-46**)
- AH: Address Hold Phase (optional - see **Page 12-47**)
- CD: Command Delay Phase (optional - see **Page 12-47**)
- CP: Command Phase (compulsory - see **Page 12-48**)
- DH: Data Hold Phase (optional - see **Page 12-48**)
- RP: Recovery Phase (optional - see **Page 12-51**)

12.10.3 Configuring the Asynchronous Access Cycle

This section provides an overview of the configuration options for the asynchronous access cycle.

12.10.3.1 Programmable Parameters

Table 12-23 lists the programmable parameters applicable to asynchronous accesses. (See **"Programmable Device Types" on Page 12-12** for AGEN values applicable to asynchronous memories.) Note that the "EBU_BUS…x" registers include parameters that control the four CS[3:0] chip select regions x.

Table 12-23 Asynchronous Access Programmable Parameters

Register	Parameter (Bit/Bit field)	Function
EBU_BUSAPx	ADDRC	Number of cycles in address phase (AP)
	AHOLDC	Number of Cycles in address hold phase (AH)
	CMDDELAY	Number of programmed command delay cycles. (CD)
	WAITRDC	Number of programmed wait states for read accesses. (CP)
	WAITWRC	Number of programmed wait states for write accesses. (CP)
	DATAC	Number of Data Hold cycles. (DH)

Table 12-23 Asynchronous Access Programmable Parameters (cont'd)

Register	Parameter (Bit/Bit field)	Function
EBU_BUSAPx	RDRECOVC	Number of minimum recovery cycles after a read access. (RP)
	WRRECOVC	Number of minimum recovery cycles after a write access. (RP)
	DTARDWR	Number of minimum recovery cycles between a read access and a write access. (RP)
	DTACS	Number of minimum recovery cycles when the next access going to a different memory region. (RP)
EBU_BUSAPx	EXTCLOCK	Controls the delays applied to the control signals by the EBSE and ECSE parameters[1)].
EBU_BUSCONx	WAIT	External Wait State control (OFF, asynchronous, synchronous)
	WAITINV	Reversed polarity at $\overline{\text{WAIT}}$: active low or active high
	ECSE	Timing mode for CS, $\overline{\text{RD}}$ and RD/$\overline{\text{WR}}$[1)].
	EBSE	Timing Mode for $\overline{\text{ADV}}$[1)].

1) See **"Control of ADV & Other Signal Delays During Asynchronous Accesses" on Page 12-62**

12.10.3.2 Accesses to Multiplexed Devices

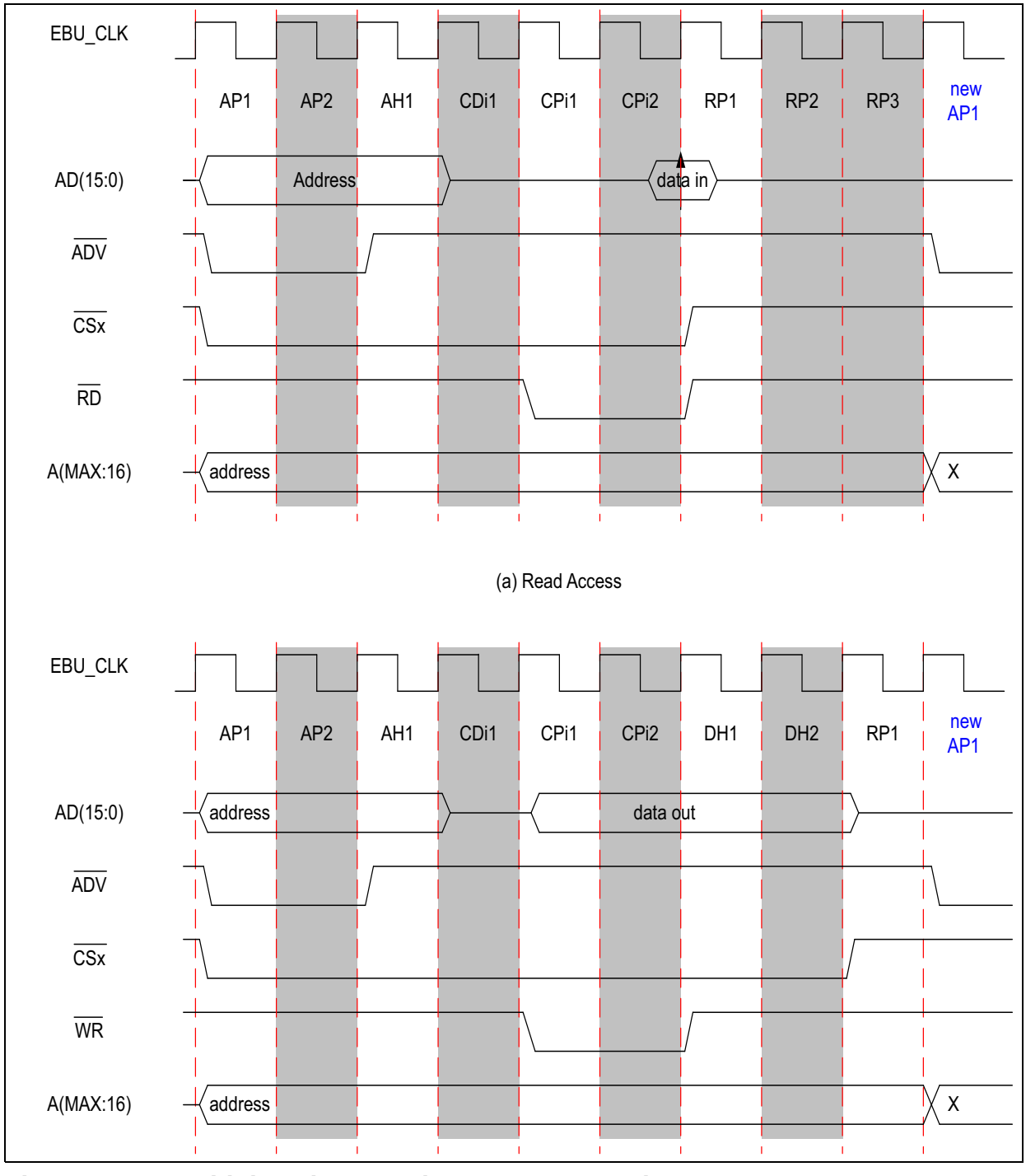


Figure 12-18 Multiplexed External Bus Access Cycles

Figure 12-18 above shows an example of a read access to a multiplexed device. This type of access cycle consists of two to six phases as follows:

- Address Phase (compulsory)
- Address Hold Phase (optional)

- Command Delay Phase (optional)
- Command Phase (compulsory)
- Data Hold Phase (optional)
- Recovery Phase (optional)

12.10.3.3 Accesses to Non-Multiplexed Devices

The following, idealised diagram (**Figure 12-19**) shows the relationship between EBU outputs and access phase. Note that, due to pad delays, the timing of the output signals will be delayed from the internal EBU clock signal, EBU_CLK.

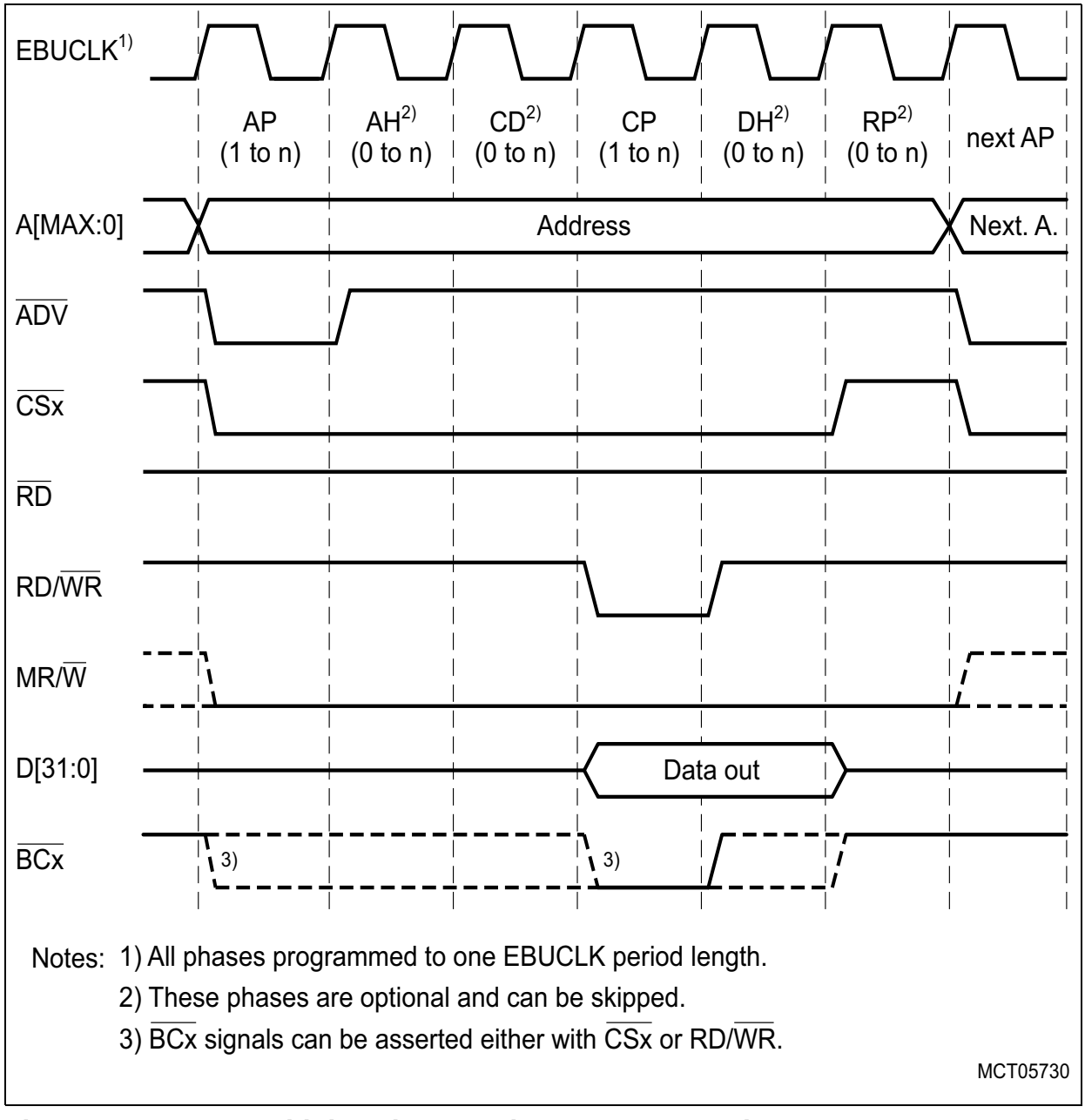


Figure 12-19 Non-Multiplexed External Bus Access Cycles

Figure 12-19 above shows an example of a write access to a non-multiplexed device. This type of access cycle consists of two to six phases as follows:

- Address Phase (compulsory)
- Address Hold Phase (optional)[1)]
- Command Delay Phase (optional)
- Command Phase (compulsory)

1) For non-multiplexed devices, the Address Hold phase is identical in function to Command Delay

- Data Hold Phase (optional)
- Recovery Phase (optional)

Read accesses are identical except that the data bus D[31:0] is not driven and the RD/WR and MR/W signals remain high for th eduration of the access.

12.10.3.4 Dynamic Command Delay and Wait State Insertion

In general, there are two critical phases during asynchronous device accesses. These phases are:

- **Command Delay Phase** (see **Page 12-47**).
- **Command Phase** (see **Page 12-48**).

In the EBU, internal length programming for the Command Delay Phase is available via bit fields EBU_BUSAPx.CMDDELAY.

The equivalent control capability for the Command Phase is available for bit fields EBU_BUSRAPx.WAITRDC and EBU_BUSWAPx.WAITWRC.

External Extension of the Command Phase by WAIT

The WAIT input can be used to cause the EBU to extend the Command Phase by inserting additional cycles prior to deactivation of the RD and RD/WR lines. This signal can be programmed separately for each region to be ignored or sampled either synchronously or asynchronously (selected via the EBU_BUSCONx.WAIT bit field). Additionally, the polarity of WAIT can be programmed for active low (default after reset) or active high function via bit EBU_BUSCONx.WAITINV. The signal will only take effect after the programmed number of Command Phase cycles has passed. This means that the signal can only be used to extend the phase, not to shorten it.

When programmed for synchronous operation, WAIT is sampled on every rising edge of EBU_CLK during the Command Phase. The sampled value is then used on the next rising edge of EBU_CLK to decide whether to prolong the Command Phase or to start the next phase. **Figure 12-20** shows an example of WAIT used in Synchronous Mode.

Note: Due to the one-cycle delay in Synchronous Mode between the sampling of the WAIT input and its evaluation by the EBU, the Command Phase must always be programmed to be at least one EBU_CLK cycle (via EBU_BUSAPx.WAITRDC or EBU_BUSAPx.WAITWRC) in this mode.

When programmed for asynchronous operation, WAIT is also sampled at each rising edge of EBU_CLK during the Command Phase. However, an extra synchronization cycle is inserted prior to the use of the sampled value. This means that the sampled value is not used until the second following rising edge of EBU_CLK. **Figure 12-21** shows an example of WAIT used in Asynchronous Mode.

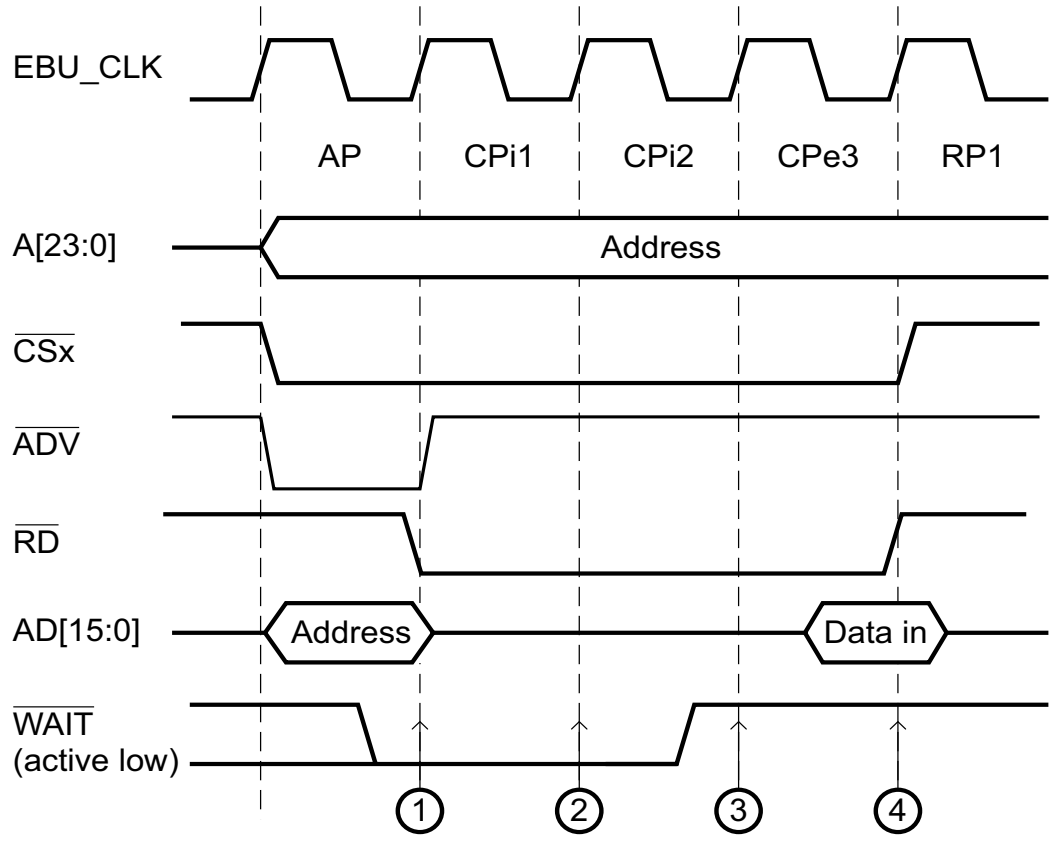
Note: Due to the two-cycle delay in Asynchronous Mode between the sampling of the WAIT input and its evaluation by the EBU, the Command Phase must always be

programmed to be at least two EBU_CLK cycles (via EBU_BUSAPx.WAITRDC or EBU_BUSAP.WAITWRC) in this mode.

Figure 12-20 shows an example of the extension of the Command Phase through the WAIT input in synchronous mode:

- At EBU_CLK edge 1 (at the end of the Address Phase), the EBU samples the WAIT input as low and starts the first cycle of the Command Phase (CPi1 - internally programmed).
- At EBU_CLK edge 2, the EBU samples the WAIT input as low and starts an additional Command Phase cycle (CPe2 - externally generated) as a result of the WAIT input sampled as low at EBU_CLK edge 1.
- At EBU_CLK edge 3, the EBU samples the WAIT input as high and starts an additional Command Phase cycle (CPe3 - externally generated) as a result of the WAIT input sampled as low at EBU_CLK edge 2.
- Finally at EBU_CLK edge 4, as a result of the WAIT input sampled as high at point 3, the EBU terminates the Command Phase, reads the input data from D[31:0] and starts the Recovery Phase.

Note: Synchronous operation means that even though access to the device may be asynchronous, the control logic generating the control signals must meet setup and hold time requirements with respect to EBU_CLK.



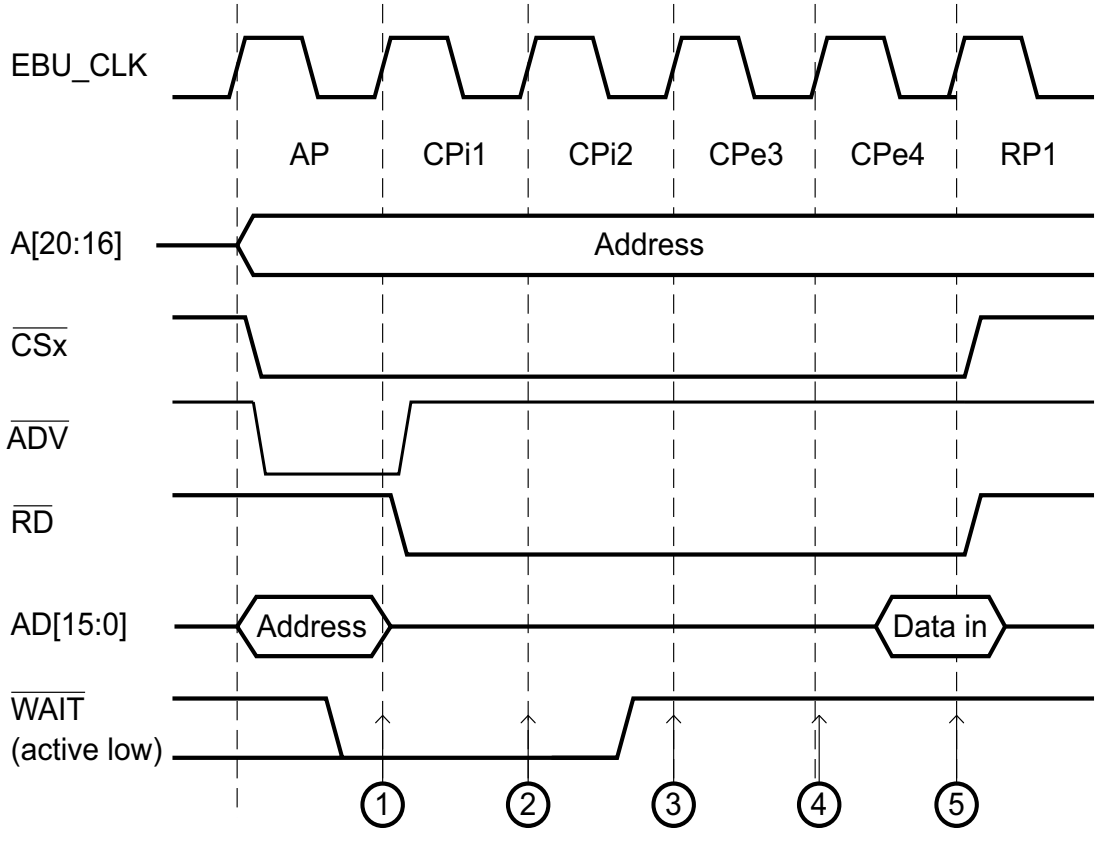
Note: In the example above, the Command Delay phase is internally programmed to zero EBU_CLK cycles (no Command Delay phase). The Command Phase is internally programmed to one EBU_CLK cycle. All other phases are programmed for one EBU_CLK cycle.

Figure 12-20 External Wait Insertion (Synchronous Mode)

Figure 12-21 shows an example of the extension of the Command Phase through the WAIT input in asynchronous mode:

- At EBU_CLK edge 1 (at the end of the Address Phase), the EBU samples the WAIT input as low and starts the first cycle of the Command Phase (CPi1 - internally programmed).
- At EBU_CLK edge 2, the EBU samples the WAIT input as low and starts the second cycle of the Command Phase (CPi2 - internally programmed).
- At EBU_CLK edge 3, the EBU samples the WAIT input as high and starts an additional Command Phase cycle (CPe3 - externally generated) as a result of the WAIT input sampled as low at EBU_CLK edge 1.

- At EBU_CLK edge 4, the EBU starts an additional Command Phase cycle (CPe4 - externally generated) as a result of the WAIT input sampled as low at EBU_CLK edge 2.
- Finally at EBU_CLK edge 5, as a result of the WAIT input sampled as high at EBU_CLK edge 3, the EBU terminates the Command Phase, reads the input data from AD[15:0],and starts the Recovery Phase.



Note: In the example above, the Command Delay phase is internally programmed to zero EBU_CLK cycles (no Command Delay phase). The Command Phase is internally programmed to two EBU_CLK cycles. All other phases are programmed for one EBU_CLK cycle.

Figure 12-21 External Wait Insertion (Asynchronous Mode)

12.10.3.5 Control of ADV & Other Signal Delays During Asynchronous Accesses

For asynchronous accesses, the Memory Controller output signals: ADV, CS, RD, RD/WR, and BC signals can be delayed with respect to the start of the access phases they are asserted in. The amount by which the signal is delayed depends on the setting of the EXTCLOCK field of EBU_BUSAPx :-

- When EXTCLOCK is set to 00_B, control signals are asserted on the negative edge of EBU_CLK. i.e. they are in effect delayed by an EBU_CLK high pulse width (T_{PH})with respect to the other signals.
- When EXTCLOCK is not set to 00_B, control signals are asserted on the next positive edge of EBU_CLK. i.e. they are in effect delayed by an EBU_CLK cycle (T_{CLK}) with respect to the other signals.

The Memory Controller allows these delays to be enabled and disabled independently via the register bits EBU_BUSCONx.EBSE for $\overline{ADV}$ and EBU_BUSCONCx.ECSE for the other control signals. The default setting after reset has the delay disabled.

Table 12-24 $\overline{ADV}$ Signal Timing

EXTCLOCK is set to	ADV Falling Edge Position		ADV Rising Edge Position	
	Delay Disabled[1)]	Delay Enabled	Delay Disabled[1)]	Delay Enabled
00_B	Start of AP1	Start of AP1 + T_{PH}	Start of AP1	Start of AP1 + T_{PH}
01_B, 10_B, 11_B	Start of AP1	End of AP1	Start of AP1	End of AP1

1) See **Figure 12-18** for details of this signal positioning.

Table 12-25 $\overline{RD}$ and RD/$\overline{WR}$ Signal Timing

EXTCLOCK is set to	Set at:		Cleared at:	
	Delay Disabled[1)]	Delay Enabled	Delay Disabled[1)]	Delay Enabled
00_B	Start of CP1	Start of CP1 + T_{PH}	End of CPn[2)]	End of CPn + T_{PH}
01_B, 10_B, 11_B	Start of CP1	End of CP1	End of CPn	End of CPn + T_{CLK}

1) See **Figure 12-18** for details of this signal positioning.

2) CPn indicates the final Command Phase.

Table 12-26 $\overline{CS}$ Signal Timing

EXTCLOCK is set to	Set at:		Cleared at:	
	Delay Disabled[1]	Delay Enabled	Delay Disabled[1]	Delay Enabled
00_B	Start of AP1	Start of AP1 + T_{PH}	End of DHn[2]	End of DHn + T_{PH}
01_B, 10_B, 11_B	Start of AP1	End of AP1	End of DHn	End of DHn + T_{CLK}

1) See **Figure 12-18** for details of this signal positioning.

2) DHn indicates the final Data Hold Phase. This is replaced by CPn, the final Command Phase, if the programmed Data Hold phase length is zero clocks.

The byte control signals, $\overline{BC}x$, can either use the timing in **Table 12-25** if EBU_BUSCONx.BCGEN is set to 01_B or 10_B or the timing in **Table 12-26** if EBU_BUSCONx.BCGEN is set to 00_B.

Write data is handled differently to the other signals. When delays are enabled using ECSE, the following determines the amount of delay applied:

- When EXTCLOCK is set to 00_B, the time at which write data is output is not delayed but the time at which it is removed from the bus is delayed by one EBU_CLK cycle.
- When EXTCLOCK is not set to 00_B, the time when write data bus is enabled is not delayed but the write data is driven on the next positive edge of EBU_CLK (i.e. the valid write data is driven an EBU_CLK cycle after the bus is enabled). The time at which write data is removed from the bus is delayed by one EBU_CLK.

Table 12-27 Write Data Signal Timing

EXTCLOCK is set to	Driven at:		Removed at:	
	Delay Disabled[1]	Delay Enabled	Delay Disabled[1]	Delay Enabled
00_B	Start of CP1	Start of CP1	End of DHn[2]	End of DHn + T_{CLK}
01_B, 10_B, 11_B	Start of CP1	End of CP1[3]	End of DHn	End of DHn + T_{CLK}

1) See **Figure 12-18** for details of this signal positioning.

2) DHn indicates the final Data Hold Phase. If no Data Hold is programmed, this will be CPn, the final Command Phase.

3) Data bus will be enabled at the beginning of CP1

These delay options apply to write data only. Addresses which are output on the data bus to support devices with multiplexed address and data connections are not delayed.

Note: If the control signals are delayed a recovery phase must be used to prevent conflicts between accesses as the rising edge of the control signals will be delayed past the end of the command phase. If a multiplexed access is used without a recovery phase, the address for the next access will be delayed by one clock cycle to enforce a bus turnaround time on the data bus, resulting in the valid address being driven one clock after $\overline{ADV}$ is asserted.

12.10.4 Interfacing to Nand Flash Devices

The memory controller provides limited support for specific Nand Flash devices. The required access sequences (read or write) are generated by connecting the Nand Flash device as an Asynchronous Device and using appropriate processor generated access sequences to emulate the NAND flash commands. **Figure 12-22** Shows an example of Memory Controller connected to a Nand Flash device:-

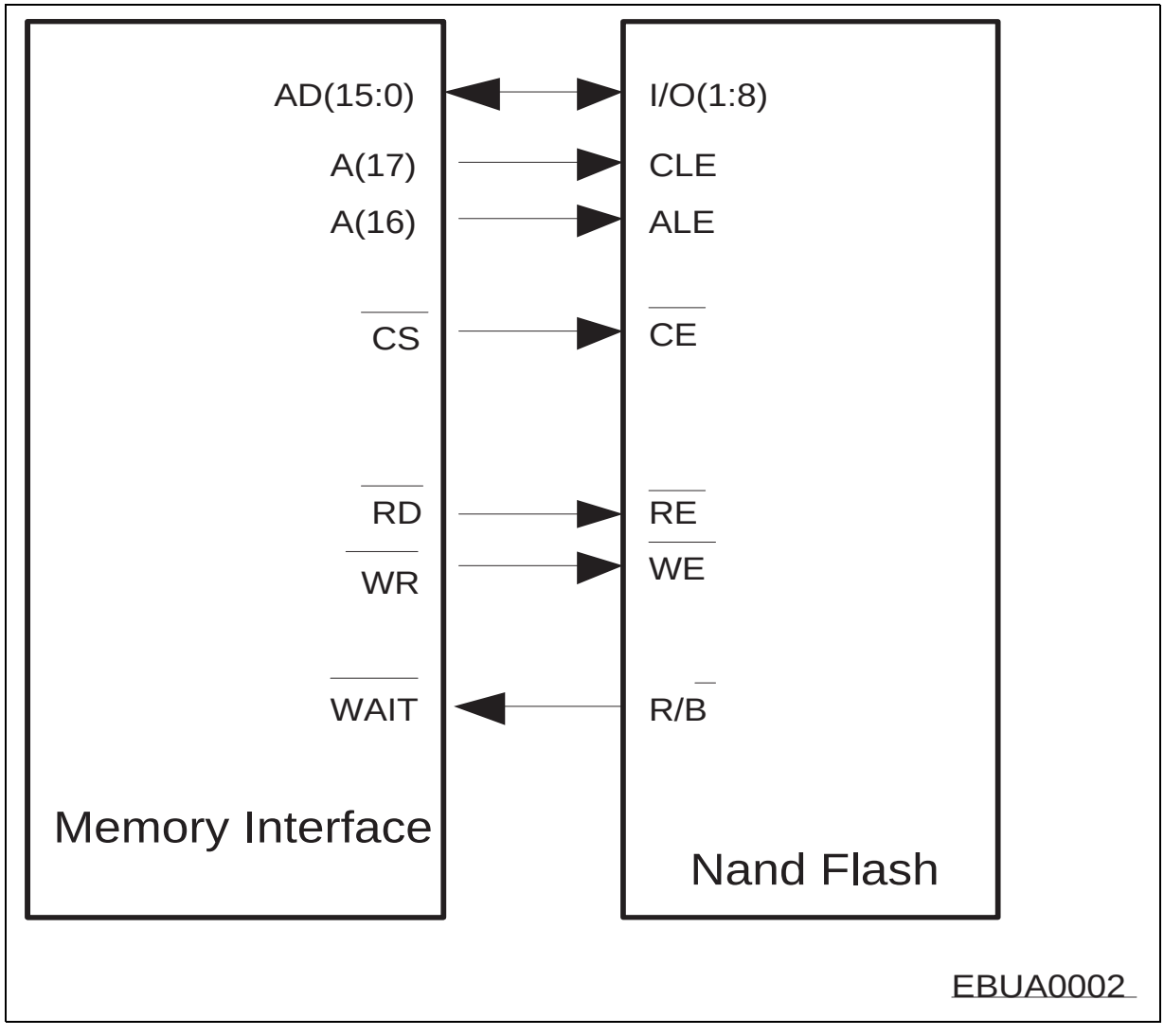


Figure 12-22 Example of interfacing a Nand Flash device to the Memory Controller

The R/$\overline{B}$ input from the NAND flash is connected to the memory controller $\overline{WAIT}$ input and is available as the EBU_MODCON.STS. This enables a NAND flash to be driven by software from the processor.

As shown above only two address lines are connected to the Nand Flash, and rather than being connected to address inputs, they are connected to control inputs. This allows access to three “registers” in the Nand Flash as follows:-

Table 12-28 Nand Flash “Registers”

LMB Address	“Register”	Comment
Base + 00000_H	Data Register	Read/Write: Used to read data from and write data to the device.
Base + 20000_H	Address Register	Write only: Used to write the required access address to the device.
Base + 40000_H	Command Register	Write only: Used to write the required command to the device.

Note: LMB addresses are byte addresses and addresses on the external bus are 16 bit word addresses. Therefore [LMB address(18)]->[external address(17)] and [LMB address(17)]->[external address(16)].

Note that the Memory Controller does not directly support byte wide devices. Writes to 8 bit, NAND Flash devices must therefore be done as 16-bit word writes with the valid byte in the lower part and the upper-byte padded.

12.10.4.1 NAND flash page mode

NAND flash memories are page oriented devices capable of extended read operations with a single setup phase for command signals at the beginning of the access. The asynchronous controller of the Memory Controller will split a large transfer into multiple accesses to external memory but each of these accesses will have the overhead of the initial setup phase. Enabling page mode, using the agen field in EBU_BUSCONx will cause the standard flow of the controller to be modified as follows:

- For a read, if data remains to be fetched at the end of a command phase, the controller will start a new command delay phase, instead of a new address phase or recovery phase and the address will not be incremented. If EBU_BUSRAPx.cmddelay is set to zero, the command delay phase will have a duration of one clock cycle but in this case the command delay phase is mandatory to ensure that the $\overline{RD}$ and RD/$\overline{WR}$ signals return to the high state.
- For a write, if data remains to be written at the end of a data hold phase (or command phase if the length of data hold is zero), the controller will start a new command phase, instead of a new address phase or recovery phase and the address will not be incremented. If EBU_BUSWAPx.datac is set to zero, the data hold phase will have a duration of one clock cycle as in this case the data hold phase is mandatory to ensure that the $\overline{RD}$ and RD/$\overline{WR}$ signals return to the high state. The command phase will be forced to have a minimum length of two clocks.

See **Figure 12-23** for example waveforms.

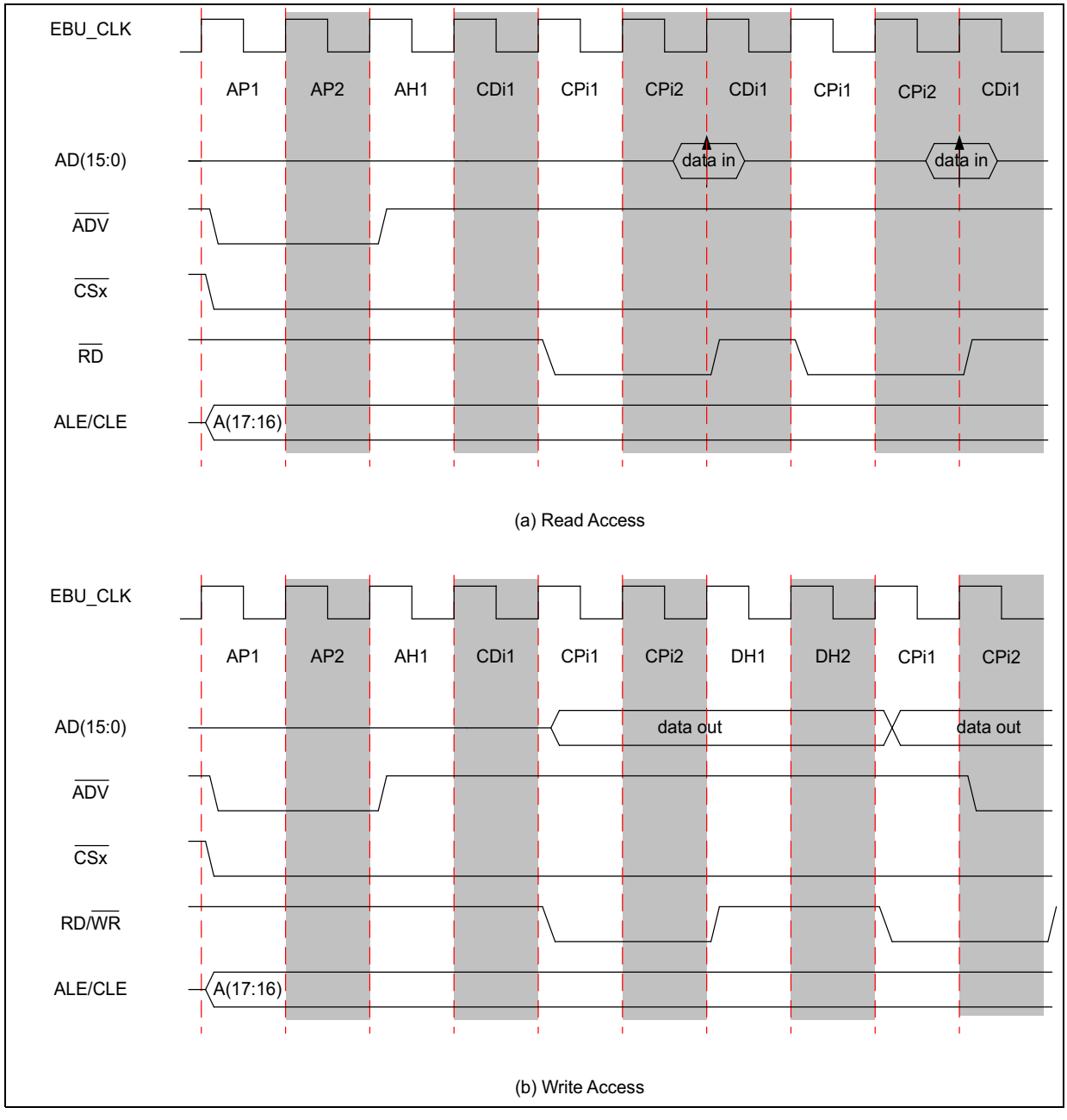


Figure 12-23 NAND Flash Page Mode Accesses

Example Nand Flash Read Sequence

Figure 12-24 shows an example of how the processor can generate a Nand Flash read access sequence given this configuration:-

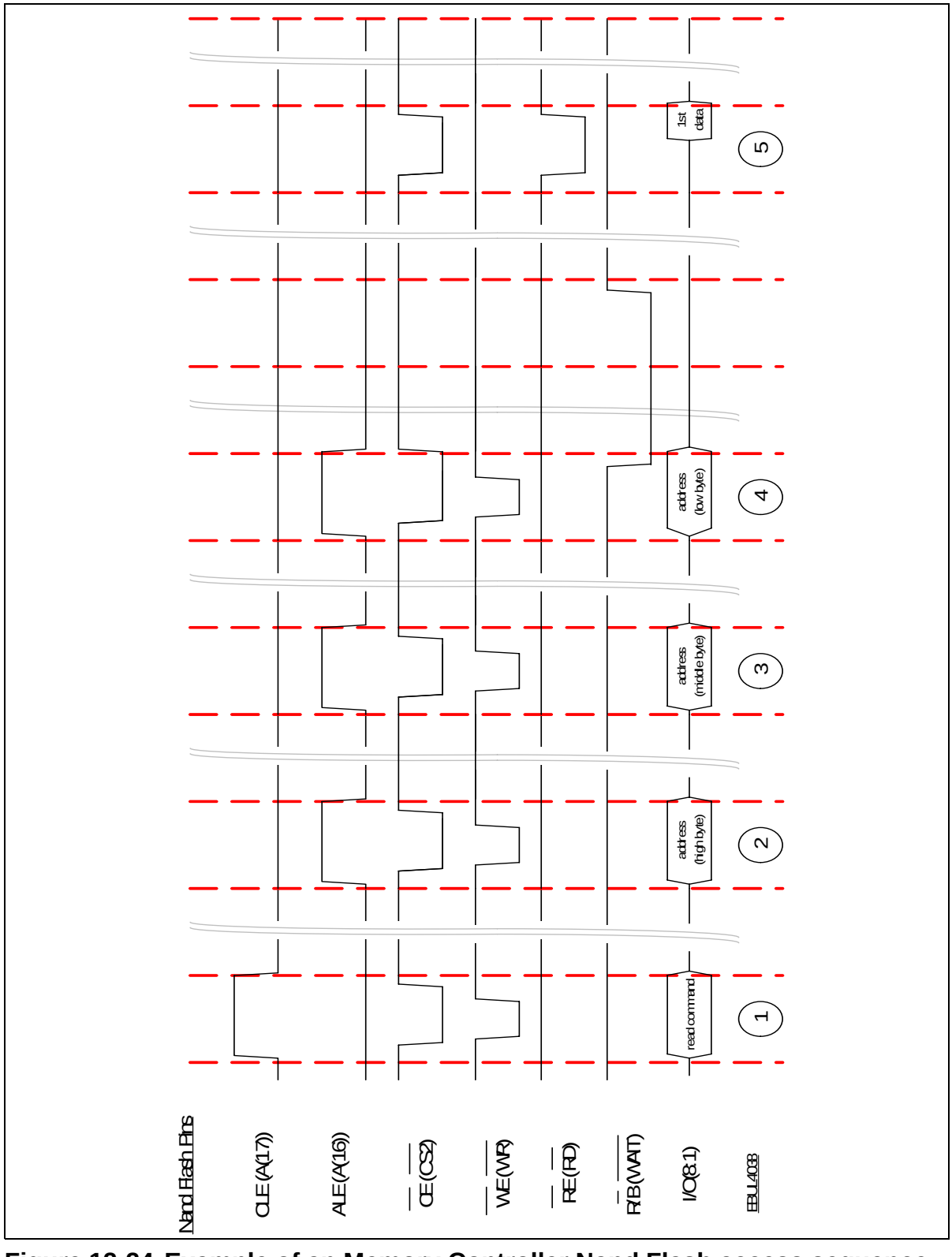


Figure 12-24 Example of an Memory Controller Nand Flash access sequence (read)

1. In the cycle marked ‘1’ in **Figure 12-24** the processor initiates a read sequence by writing the “Read Command” value to address “NAND_FLASH_BASE + 0x40000”. This generates a write sequence with CLE (A(17)) driven high and ALE (A(16)) driven low.
2. In the cycle marked ‘2’ the processor loads the most significant byte of the read address by writing to address “NAND_FLASH_BASE + 0x20000”. This generates a write sequence with CLE (A(17)) driven low and ALE (A(16)) driven high.
3. In the cycle marked ‘3’ the processor loads the middle significant byte of the read address by repeating the access specified in ‘2’ above.
4. In the cycle marked ‘4’ the processor loads the least significant byte of the read address by repeating the access specified in ‘2’ above. The Nand Flash responds to this final address byte by driving it’s R/$\overline{B}$ output low. The processor monitors this pin (using the EBU_MODCON.sts bit) until the Nand Flash has completed it’s internal data fetch.
5. In the cycle marked ‘5’ the processor reads the first byte of data by reading address “NAND_FLASH_BASE + 0x00000”. The processor can subsequently read any additionally required (sequential) data bytes by repeating cycle ‘5’.

Note: A similar scheme can be used to generate write access sequences.

12.11 Synchronous Read/Write Accesses

The Memory Controller is designed to generate waveforms compatible with the burst modes of:

1. INTEL and compatible burst flash devices
2. SPANSION and compatible burst flash devices
3. INFINEON and MICRON cellular RAM
4. Fujitsu and Compatible FCRAM/uTRAM/CosmoRAM
5. Samsung OneNAND burst capable NAND flash and compatible devices
6. M-Systems DiskOnchipG3 and compatible devices
7. GSI SSRAM

Note: Not all of the supported synchronous memory types are known to be available in automotive grade

Features

The Synchronous Access Controller is primarily designed to perform burst mode read cycles for an external instruction memory and read and write cycles for an external Cellular RAM or FCRAM data memory. In general, the features are:-

- Fully synchronous timing with flexible programmable timing parameters (address cycles, read wait cycles, data cycles).
- Programmable $\overline{\text{WAIT}}$ function.
- Programmable burst (mode and length)
- 16-bit device width.

- 32-bit device width
- Page mode read accesses.
- Resynchronisation of read data to a feedback clock to maximise the frequency of operation (optional).

12.11.1 Signals

The following signals are used for the Burst FLASH interface:-

Table 12-29 Burst Flash Signal List

Signal	Type	Function
AD(31:0)	I/O	Multiplexed Address/Data bus
$\overline{RD}$	O	Read control
$\overline{WR}$	O	Write control
A(23:0)	O	Most Significant Part of Address bus
$\overline{ADV}$	O	Address valid strobe
$\overline{WAIT}$	I	Wait/terminate burst control
$\overline{CS}$(3:0)	O	Chip select
BFCLKO	O	Burst FLASH Clock, running equal to, 1/2, 1/3 or 1/4 of the frequency of EBU_CLK.
BFCLKI	I	Burst FLASH Clock Feedback.
$\overline{BAA}$	O	Advance Burst Address
STS	I	Burst FLASH Status Input (Optional, value of $\overline{WAIT}$ pin available through status register)

12.11.2 Support for four Burst FLASH device types

Support is provided for a maximum of four different Burst FLASH configurations on the external bus - i.e. one on each external chip select.

Bit-fields EBU_BUSCONx.EBSE, EBU_BUSCONx.ECSE, EBU_BUSCONx.wait, EBU_BUSCONx.FBBMSEL, EBU_BUSCONx.BFCMSEL and EBU_BUSCONx.FETBLEN are used to configure specific characteristics for burst access cycles.

12.11.3 Typical Burst Flash Connection

The figure below shows a typical burst flash connection.

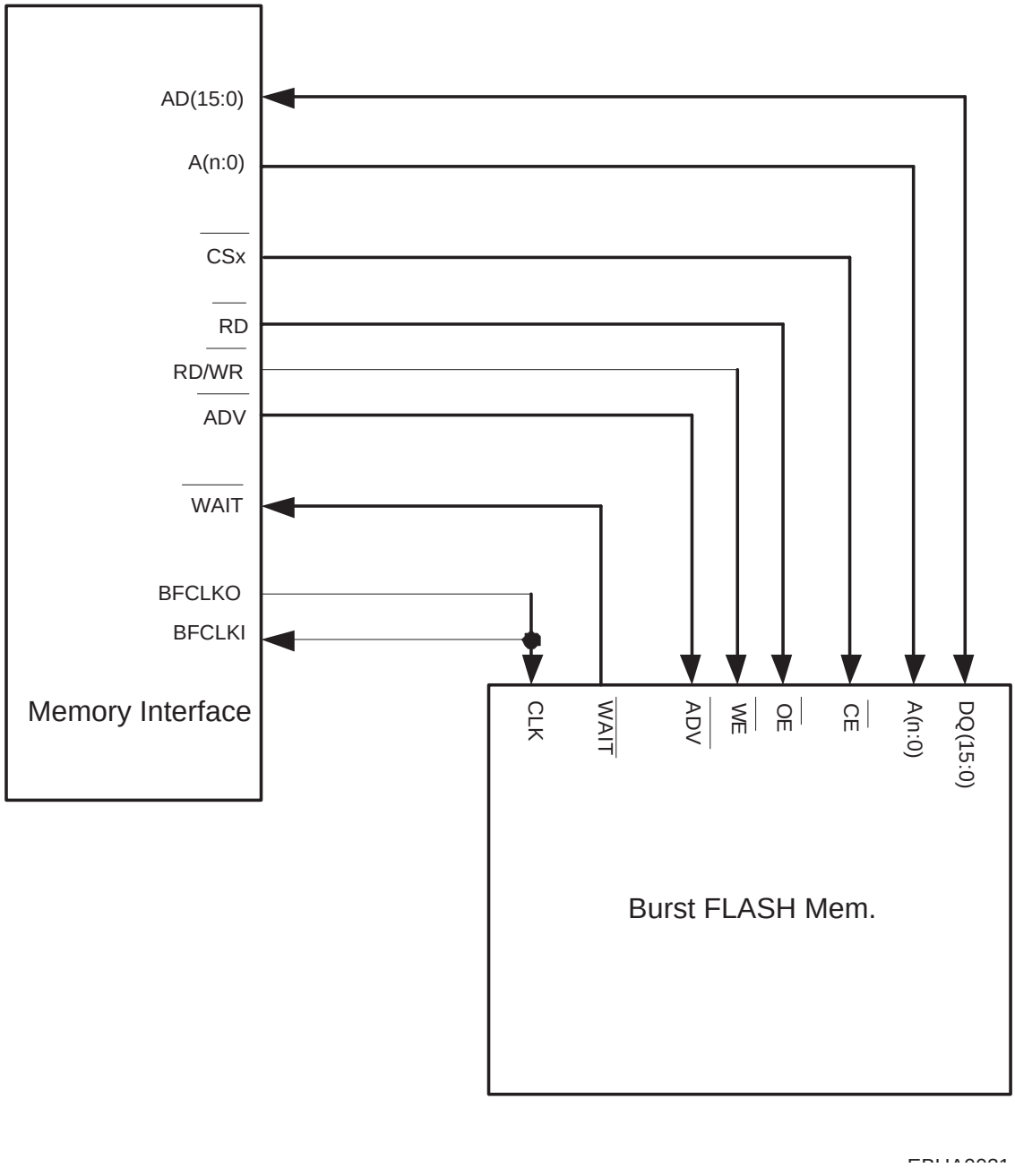


Figure 12-25 Typical Burst Flash Connection

12.11.4 Burst Flash Clock

Since the EBU_CLK can run too fast for clocking Burst FLASH devices, the Memory Controller provides an additional clock source (BFCLKO). This signal is generated by a programmable clock divider driven by EBU_CLK and allows EBU_CLK to BFCLKO ratios of 1:1, 2:1, 3:1 and 4:1 to be selected. The frequency of the signal is determined by bit-field EBU_BUSRP.EXTCLOCK. Note that it is possible to set a different clock rate

for synchronous writes to the same device by programming EBU_BUSWP.EXTCLOCK to a different value.

If a continuously running BFCLKO is required, then the BUSRCONx.BFCMSEL field can be used to enable an ungated flash clock. This bit is normally set to 1_B in all the BUSRCONx registers after reset. If cleared, the related BUSRAPx.EXTCLOCK field will be used to generate a stable BFCLKO. If multiple BUSRCONx.BFCMSEL fields are set to 0_B, then the highest priority (lowest index) BUSRAPx.EXTCLOCK field will be used.

During a burst access to a synchronous device, BFCLKO will generate correctly aligned clock edges as shown in **Figure 12-26**. The BFCLKO signal is gated to ensure that it is low (zero) at all other times (including asynchronous read/writes of/to synchronous devices). This provides power savings and ensures correct asynchronous accesses to Burst FLASH device(s).

The start of the address and burst phases are synchronised, by hardware, to the rising edge of BFCLKO. Exiting from a phase extended by the WAIT input will also be synchronised to the rising edge of BFCLKO. This is intended to ensure that control signal changes will normally occur at the rising edge of BFCLKO unless configured otherwise by register settings.

Note: The length of the standard accesses phases during Burst FLASH accesses are programmed as a multiple of EBU_CLK independent of the BFCLKO frequency. It is the users responsibility to program the access phases to ensure that the sampling of data by Memory Controller guarantees valid sampling of the data from the Burst FLASH device.

The EBU uses the EBU_CLK clock to generate all external bus access sequences.

Table 12-30 EXTCLOCK to clock ratio mapping

EXTCLOCK value	BFCLKO divide ratio
00	1:1
01	1:2
10	1:3
11	1:4

As the BFCLKO generation logic uses the EBU_CLK for timing, the duty cycle of BFCLKO will be affected by the pulse swallowing used by the local clock division circuit (See **“Local Clock Divider” on Page 12-45**).

Table 12-31 BFCLKO Duty Cycle

Local Clock Divider ratio	Programmed BFCLKO Divider Ratio			
	1:1	1:2	1:3	1:4
1:1	50%	50%	50%	50%
1:2	25%	50%	41.7%	50%
1:3	16.7%	50%	38.9%	50%
1:4	12.5%	50%	37.5%	50%

12.11.5 Standard Access Phases

Accesses to burst FLASH devices are composed of a number of “Standard Access Phases” (which are detailed in **Section 12.9**). The Standard Access Phases for Burst FLASH devices are:-

- AP: Address Phase (compulsory - see **“Address Phase (AP)” on Page 12-46**).
- AH: Address Hold Phase (optional see **“Address Hold Phase (AH)” on Page 12-47**).
- CD: Command Delay Phase (optional - see **“Command Delay Phase (CD)” on Page 12-47**).
- CP: Command Phase (compulsory - see **“Command Phase (CP)” on Page 12-48**).
- BP: Burst Phase (compulsory - see **“Burst Phase (BP)” on Page 12-49**).
- DH: Data Hold Phase[1)] (optional - see **“Data Hold Phase (DH)” on Page 12-48**).
- RP: Recovery Phase (optional - see **“Recovery Phase (RP)” on Page 12-51**).

Note: During a burst access the Burst Phase (BP) is repeated the required number of times to complete the burst length.

12.11.6 Burst Length Control

The maximum number of valid data samples that can be generated by a flash device in a single read access is set by the EBU_BUSCONx.FBBMSEL bit and the EBU_BUSCONx.FETBLEN bit field.

The EBU_BUSCONx.FBBMSEL bit is used to select Continuous Burst Mode where there is no limit to the number of data samples in a burst read access.

The EBU_BUSCONx.FBBMSEL and EBU_BUSCONx.FETBLEN bit-fields are used to select the maximum number of data samples in a single access. Where an LMB request exceeds the amount of data that can be fetched or stored by the programmed number of data samples, the EBU will automatically generate the appropriate number of burst accesses to transfer the required amount of data.

1) Data Hold can be used for synchronous accesses and will be inserted after the final Burst Phase.

Note: Selection of Continuous Burst Mode (by use of the 'FBBMSEL' bit) overrides the maximum burst setting (specified by the FETBLEN bit-field).

12.11.7 Control of $\overline{ADV}$ & Control Signal Delays During Synchronous Accesses

The Memory Controller output signals: $\overline{ADV}$, $\overline{CS}$, $\overline{RD}$, RD/$\overline{WR}$, $\overline{BC}$ and AD signals can be delayed with respect to the BFCLKO clock signal.

The delays can be enabled and disabled by the register bits EBU_BUSCONx.EBSE for the $\overline{ADV}$ signal and by EBU_BUSCON.ECSE for the $\overline{CS}$, $\overline{RD}$, RD/$\overline{WR}$, $\overline{BAA}$ and write data signals

The amount by which the signal is delayed depends on the ratio of EBU_CLK to the Burst FLASH clock as follows:-

- When the ratio of EBU_CLK to BFCLKO is 1:1, signals are asserted on the negative edge of EBU_CLK. i.e. it is in effect delayed by an EBU_CLK high pulse width (T_{PH}) with respect to BFCLKO.
- When the ratio of EBU_CLK to BFCLKO is 1:2 or 1:3, control signals are asserted on the next positive edge of EBU_CLK. i.e. it is in effect delayed by an EBU_CLK cycle (T_{CLK}) with respect to BFCLKO.
- When the ratio of EBU_CLK to BFCLKO is 1:4, control signals are asserted on the negative edge of BFCLKO. i.e. it is in effect delayed by two EBU_CLK cycles ($2*T_{CLK}$) with respect to BFCLKO.

The default setting after reset has the delays disabled.

If the delay is disabled, then the signals will not be delayed in 1:1 mode (except for ADV which will be guaranteed to be after the edge of BFCLKO). In 2:1, 3:1 and 4:1 mode, the signals will be delayed by an EBU_CLK high pulse width (T_{PH}) from the start of the cycle in which they are asserted.

Table 12-32 $\overline{ADV}$ Signal Timing

EXTCLOCK is set to	$\overline{ADV}$ Falling Edge position		$\overline{ADV}$ Rising Edge position	
	Delay Disabled[1)]	Delay Enabled	Delay Disabled[1)]	Delay Enabled
00_B	Start of AP1	Start of AP1+T_{PH}	End of APn	End of APn+T_{PH}
01_B, 10_B	Start of AP1+T_{PH}	End of AP1	End of APn+T_{PH}	End of APn+ T_{CLK}
11_B	Start of AP1+T_{PH}	End of AP1 + T_{CLK}	End of APn+T_{PH}	End of APn + $2*T_{CLK}$

1) See **Figure 12-26** for details of this signal positioning.

Table 12-33 $\overline{RD}$ and RD/$\overline{WR}$ Signal Timing

EXTCLOCK is set to	Set at:		Cleared at:	
	Delay Disabled[1]	Delay Enabled	Delay Disabled	Delay Enabled
00_B	Start of CP1	Start of CP1 + T_{PH}	End of CPn[2]	End of CPn + T_{PH}
01_B, 10_B	Start of CP1 + T_{PH}	End of CP1	End of CPn + T_{PH}	End of CPn + T_{CLK}
11_B	Start of CP1 + T_{PH}	End of CP1 + T_{CLK}	End of CPn + T_{PH}	End of CPn + 2*T_{CLK}

1) See **Figure 12-26** for details of this signal positioning.

2) CPn indicates the final Command Phase.

Table 12-34 $\overline{CS}$ Data Signal Timing

EXTCLOCK is set to	Set at:		Cleared at:	
	Delay Disabled[1]	Delay Enabled	Delay Disabled	Delay Enabled
00_B	Start of AP1	Start of AP1 + T_{PH}	End of DHn[2]	End of DHn + T_{PH}
01_B, 10_B	Start of AP1 + T_{PH}	End of AP1	End of DHn + T_{PH}	End of DHn + T_{CLK}
11_B	Start of AP1 + T_{PH}	End of AP1 + T_{CLK}	End of DHn + T_{PH}	End of DHn + 2*T_{CLK}

1) See **Figure 12-26** for details of this signal positioning.

2) DHn indicates the final Data Hold Phase. This is replaced by CPn if the programmed Data Hold phase length is zero clocks.

The byte control signals, $\overline{BC}$x, can either use the timing in **Table 12-33** if EBU_BUSCONx.BCGEN is set to 01_B or 10_B or the timing in **Table 12-34** if EBU_BUSCONx.BCGEN is set to 00_B.

Table 12-35 Write Data Signal Timing

EXTCLOCK is set to	Data Driven at:		Data Cleared at:		Data Changes at:	
	Delay Disabled[1)]	Delay Enabled	Delay Disabled	Delay Enabled	Delay Disabled	Delay Enabled
00_B	Start of CP1	Start of CP1 + T_{PH}	End of DHn[2)]	End of DHn + T_{PH}	Start of BP1	Start of BP1 + T_{PH}
01_B, 10_B	Start of CP1 + T_{PH}[3)]	End of CP1[3)]	End of DHn + T_{PH}	End of DHn + T_{CLK}	Start of BP1 + T_{PH}	End of BP1
11_B	Start of CP1 + T_{PH}[3)]	End of CP1 + T_{CLK}[3)]	End of DHn + T_{PH}	End of DHn + 2*T_{CLK}	Start of BP1 + T_{PH}	End of BP1 + T_{CLK}

1) See **Figure 12-26** for details of this signal positioning.

2) DHn indicates the final Data Hold Phase. This is replaced by CPn if the programmed Data Hold phase length is zero clocks.

3) Data bus will be enabled at the start of CP1

These delay options apply to write data only. Addresses which are output on the data bus to support devices with multiplexed address and data connections are not delayed.

Note: If the control signals are delayed a recovery phase must be used to prevent conflicts between accesses as the rising edge of the control signals will be delayed past the end of the command phase. If a multiplexed access is used without a recovery phase, the address for the next access will be delayed by one clock cycle to enforce a bus turnaround time on the data bus, resulting in the valid address being driven one clock after ADV is asserted.

12.11.8 Burst Flash Clock Feedback

The Memory Controller can be configured to use clock feedback to optimise the operating frequency for a given flash device. This is enabled by setting the EBU_BUSCONx.FDBKEN bit to one. With this bit enabled the first sampling stage for read data has its own clock (PD_BFCLKFEEDBK_I). This will be derived from the BFCLKO output by using a second pad (BFCLKI) to monitor the BFCLKO signal after the output pad delay.

Clock feedback should be used whenever possible as it allows the best possible performance

In addition the number of synchronisation stages (one or two) used to transfer the read-data into the EBU_CLK domain can also be selected via the EBUCON_BUSAPx.BFSSS bit.

When using two synchronisation stages (default) the data is initially resynchronised to PD_BFCLKFEEDBK_I and then is additionally internally resynchronised to BFCLKO before being passed to the normal logic. In this mode PD_BFCLKFEEDBK_I can therefore be skewed by almost an entire BFCLK cycle relative to the EBU_CLK clock without losing data integrity. A side effect of using this mode is an increase in data latency of two cycles of BFCLK (compared to not using clock feedback).

When using a single synchronisation stage the data is resynchronised to PD_BFCLKFEEDBK_I before being passed to the normal logic. This provides a compromise setting for operating frequency and latency for 1:1 clocking mode where the second resynchronisation stage offers no advantage.

Note: If LMB_CLK:BFCLKO = 1:1, then the second and third resynchronisation stages have identical clock signals. There is therefore no advantage to having the second resynchronisation stage and it can be bypassed without loss of performance.

As above, a side effect of using this mode is an increase in data latency. In this case addition of one BFCLK cycle (compared to not using clock feedback).

Note: Clock feedback will be automatically disabled for burst writes as the additional latency on the $\overline{WAIT}$ input cannot be tolerated

12.11.9 Asynchronous Address Phase

As operating frequency increases, it becomes increasingly hard to avoid violating some timing parameters. The asynchronous address phase allows the address to be latched into the flash memory using the ADV signal before the clock is enabled. This is only possible if explicitly allowed by the flash data sheet

If the EBU_BUSCONx.AAP is set, then the clock will not start until the end of the address hold phase of the access. The rising edge of the clock will always be co-incident with the transition from the address hold phase. If the address hold phase has zero length then the first rising edge of the clock will coincide with the transition from the address phase. If this mode is enabled a recovery phase of one BFCLK period will be enforced at the end of the previous transaction to ensure that the clock has time to turn off before the start of the next access.

Setting this mode for any region will force the clocks on all synchronous accesses to be disabled in the recovery phase. Only one clock pulse will occur during the recovery phase.

AAP mode is incompatible with the continuous clock mode and will be disabled automatically is continuous clocking is enabled by setting any BUSRCONx.BFCMSEL bit to 0_B.

12.11.10 Critical Word First Read Accesses

In the default case, the memory controller will always start a burst at the lowest address possible and wrapping of the burst data is handled internally to the memory controller. However, some burst devices implement a wrapping feature which is compatible with the wrapped bursts used by the processor cache fill requests. If this is the case, there is an advantage to using the wrapping mode in the device as the instruction required by the processor (the critical word) can be fetched first from the external memory.

The mode is enabled by setting the EBU_BUSCONx.dba bit for the appropriate region. Once enabled, the memory controller will not align the start address for the burst and the device will be relied on to return data in the correct order. The memory controller must fetch all the data in a single burst. If the transaction is split into multiple accesses on the external bus, the issued addresses will be incorrect.

Note: The cache line fill will use an LMB, BTR4 transfer. This translates to a 16 word burst for a 16 bit device. The device must therefore support a 16 word wrap setting. A 32 bit memory must support a 8 word wrap setting. The memory controller supports a 16 word burst using the continuous burst setting for FBBMSEL. Other BTR opcodes must not be generated for accesses to the external memory by the system if this mode is enabled, otherwise data corruption will occur.

12.11.11 Example Burst Flash Access Cycle

The figure below shows an example burst flash access.

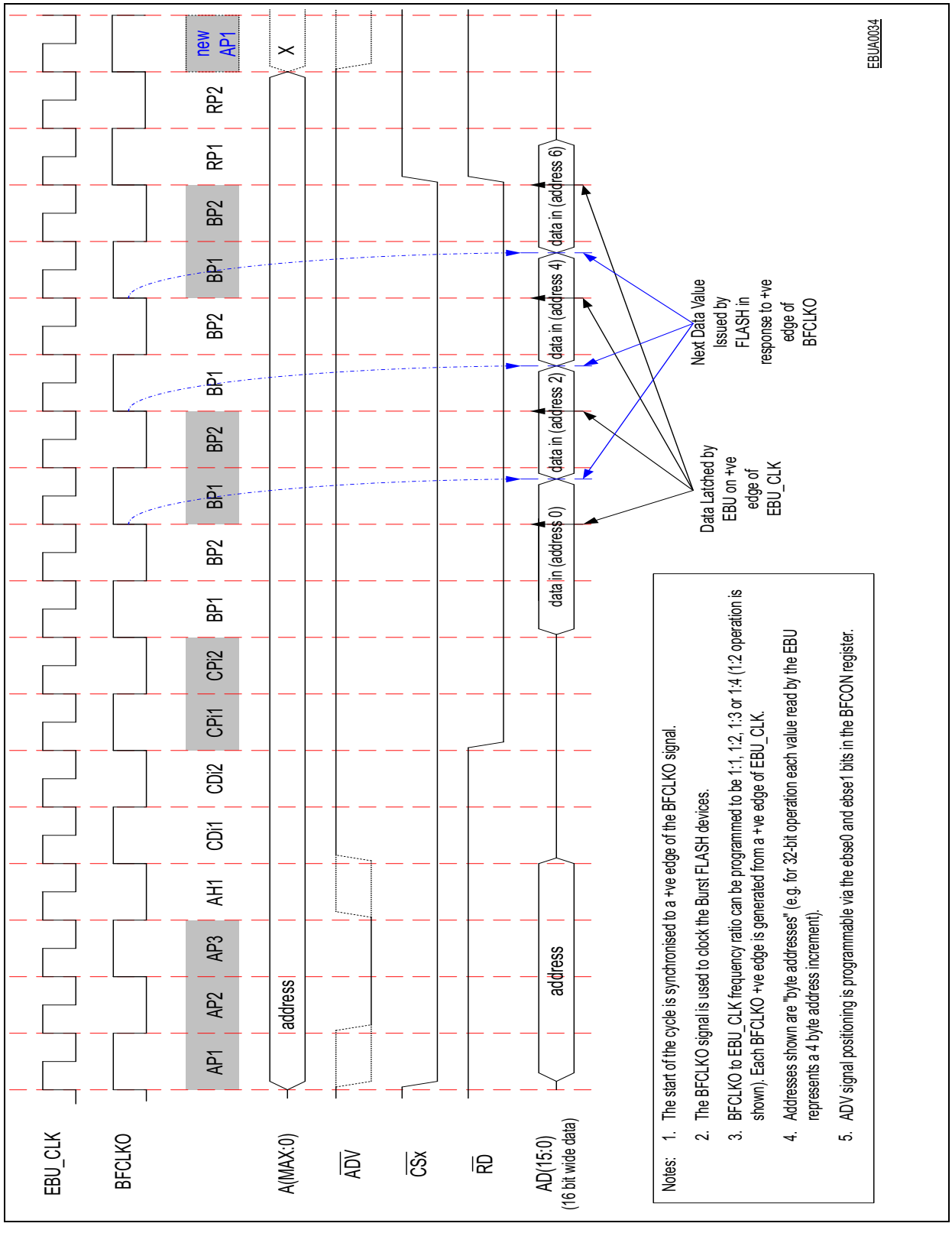


Figure 12-26 Burst FLASH Read without Clock Feedback (burst length of 4)

Figure 12-26 shows an example of a burst read access (burst length of four) to a Burst FLASH device with WAIT and clock feedback functions disabled.

Programmability of the length of the Address, Command Delay and Command phases allows flexible configuration to meet the initial read access time of a Burst FLASH device.

Data is sampled at the end of each Burst Phase cycle. The Burst Phase is repeated the appropriate number of times for the programmed burst length (programmable for lengths of 1, 2, 4 or 8 via the EBU_BUSCONx.FETBLEN bit-field).

Figure 12-26 shows an access cycle with the following settings:-

- Clock Feedback disabled.
- Address Phase length = 3 EBU_CLK cycles (see ADDRC and **"Address Phase (AP)" on Page 12-46**).
- Command Delay Phase length = 3 EBU_CLK cycles (see CMDDELAY and **"Command Delay Phase (CD)" on Page 12-47**).
- Command Phase length = 2 EBU_CLK cycles (see WAITRDC and **"Command Phase (CP)" on Page 12-48**).
- Burst Phase length = 2 EBU_CLK cycles (see EXTCLOCK, EXTDATA and **"Burst Phase (BP)" on Page 12-49**).
- Recovery Phase length = 2 EBU_CLK cycles (see **"Recovery Phase (RP)" on Page 12-51**).
- Burst Length = 4 (see FETBLEN).
- BFCLKO frequency = 1/2 of EBU_CLK frequency (see EXTCLOCK).

12.11.12 External Cycle Control via the WAIT Input

Memory Controller provides control of the Burst FLASH device via the WAIT input. This allows Memory Controller to support operation of Burst FLASH while crossing Burst FLASH page boundaries. During a Burst FLASH access the WAIT input operates in one of four modes:-

- Disabled
- Early Wait for Page Load.
- Wait for Page Load.
- Abort and Retry Access.

Selection of the mode in which the WAIT input operates during Burst FLASH reads is selected via the EBU_BUSCONx.wait bits.

Note: Selection of "Disabled" via the wait bit-field prevents the WAIT input having any effect on a Burst FLASH access cycle

Wait for Page Load Mode

This mode supports devices which assert a WAIT output for the duration of clock cycles in which the data output by the device is invalid or, alternatively, one clock cycle earlier than the data output is invalid. This includes Intel and AMD Burst FLASH devices (and

compatibles) configured for Early Wait Generation Mode (EBU_BUSCONx.wait=01_B) and standard wait generation (EBU_BUSCONx.wait=10_B).

In operation, the burst flash controller loads a counter with the required number of samples at the start of each burst. At the end of each burst phase, the burst flash controller samples the WAIT input and the data bus at the end of each Burst phase. If WAIT is inactive, the sample is valid, the sample counter is decremented and the sampled data is passed to the datapath of the Memory Controller. This synchronous sampling means that the validity of the sample can not be determined until the clock cycle after the end of the burst phase. The Burst Flash controller will therefore overrun and generate extra burst phases until the sample counter is decremented to zero. Extra data samples returned after the sample counter is zero will be discarded.

The only difference if early wait is used is that the validity of data in burst phase “n” is determined by the value of WAIT in burst phase “n-1”.

This mode of operation is compatible with the use of clock feedback as, with feedback enabled, WAIT is fed through the same resynchronisation signals as the data bus. The only effect on operation is that the number of overrun cycles will increase as the decrementing of the sample counter will be lagged by the resynchronisation stages.

During the initial phases of an access, WAIT is sampled on every edge of EBU_CLK. This is so the first burst phase is working with an accurate value for the WAIT signal. To ensure this is the case, the command phase should be of sufficient length to allow the device to drive WAIT and for the signal to propagate to the controller.

12.11.13 Termination of a Burst Access

A burst read operation is terminated by de-asserting CSx signal followed by the appropriate length Recovery Phase. **Figure 12-27** shows an example of termination of a burst access following the read of two locations (i.e. two Burst Phases) from a 16-bit non-multiplexed Burst FLASH device.

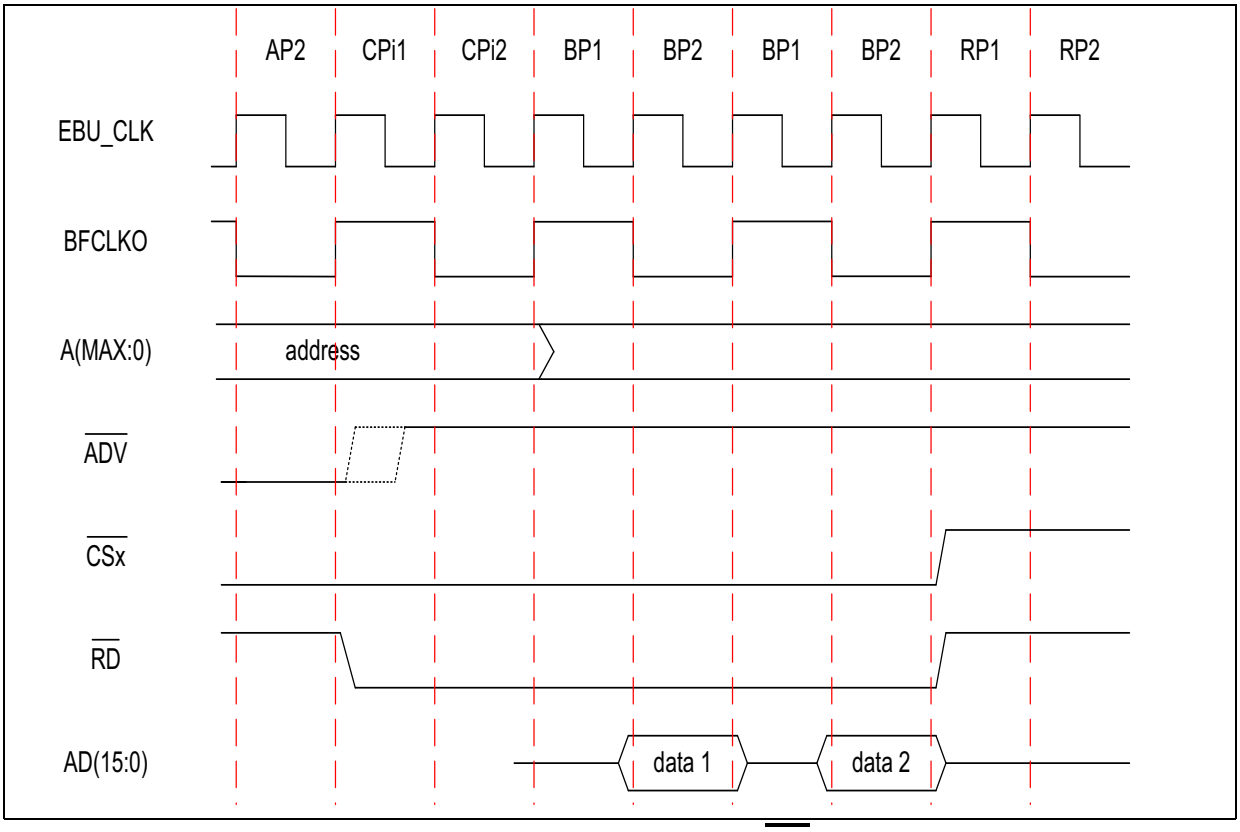


Figure 12-27 Terminating a Burst by de-asserting $\overline{CS}$

12.11.14 Burst Flash Device Programming Sequences

Programming sequences for some Burst Flash devices must not be interrupted by other read/write operations to the same device. There is an optional hardware lock feature to guarantee that programming sequences are not interrupted. See **Section 12.4.9** for details.

12.11.15 Cellular RAM

Cellular RAM devices have been designed to meet the growing memory and bandwidth demands of modern cellular phone designs. The devices have been designed with a “multi-protocol” interface to allow use of the devices with existing memory interfaces (i.e. by re-use of existing memory protocols). The supported interface protocols supported by Cellular RAM devices are:-

1. SRAM (Asynchronous Read and Write).
2. NOR Flash (Synchronous Burst Read, Asynchronous Write).
3. Synchronous (Synchronous Burst Read and Write).

In principle, when using previous versions of Memory Controller, the first two of the above modes (1 and 2 above) provided Cellular RAM support. For maximum

performance, the Memory Controller now supports Synchronous Mode (3 above) for Cellular RAM (Synchronous Burst Read and Write).

As Cellular RAM Synchronous Mode consists of a Burst FLASH compatible Burst Read access, Cellular RAM support has been provided by enhancing the Burst FLASH interface by the inclusion of a Burst Write capability. For this reason Cellular RAM is treated as a special type of Burst FLASH device.

Cellular RAM support is selected by programming the desired region as cellular RAM via the EBU_BUSCONx.agen bit-field.

Synchronous Read Access

A Synchronous Cellular RAM Burst Read Access is compatible with a Burst FLASH Burst Read Access. As a result preceding sections applying to Burst FLASH devices apply and should be consulted for details of Cellular RAM Burst Read Accesses.

Synchronous Write Access

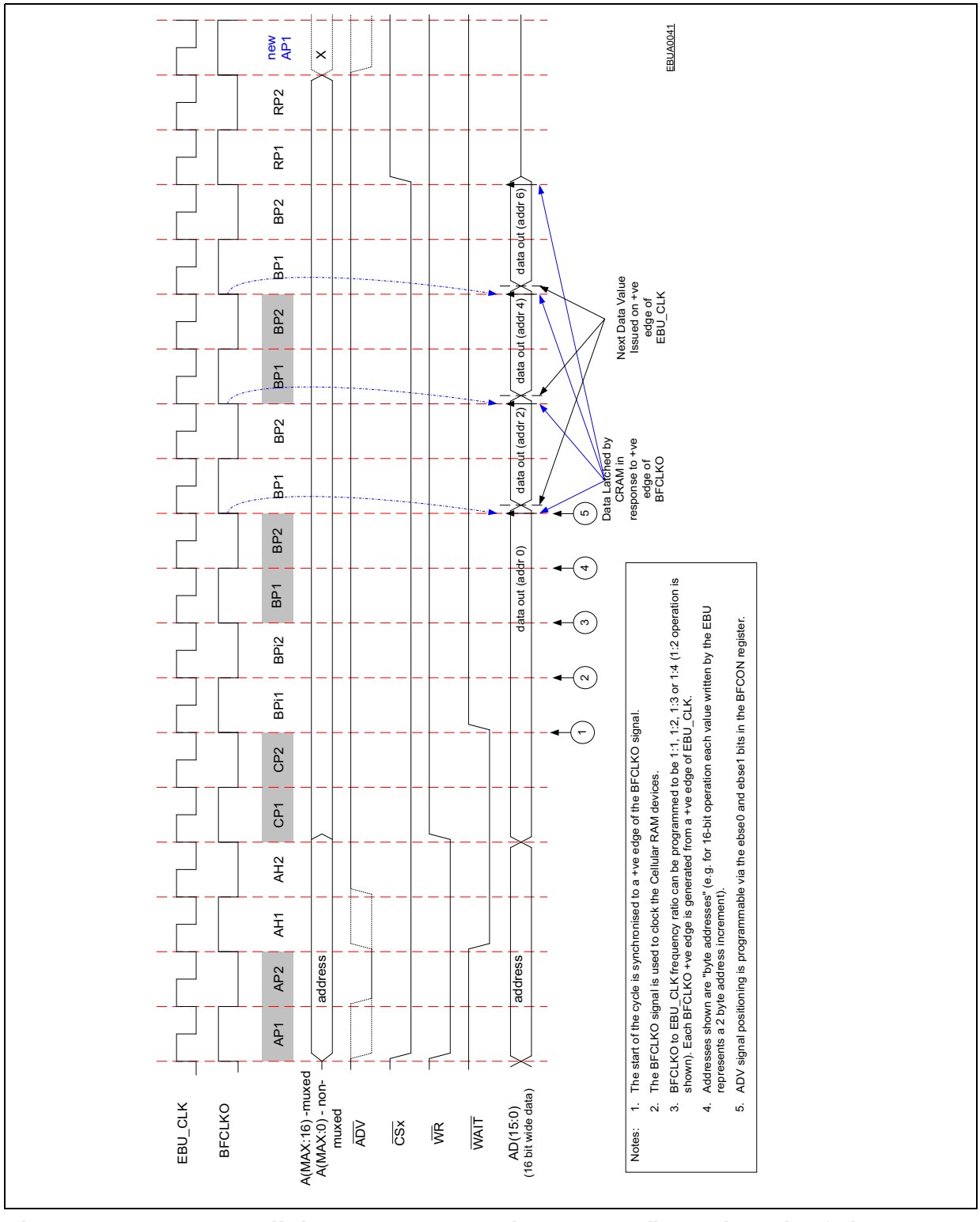


Figure 12-28 Burst Cellular RAM Burst Write Access (burst length of 4)

Figure 12-28 shows an example of a Cellular RAM burst write access.

*Note: **Figure 12-28** shows operation with a BFCLKO to EBU_CLK ratio of 1:2.*

The Start of the access cycle is the same as for a Synchronous Read access (see **Figure 12-26**) except that the WR signal is treated as an address phase signal (i.e. it is asserted active during the Address Phase and Address Hold Phase and is then deasserted). See **"Fujitsu FCRAM Support (burst write with WR active during data phase)" on Page 12-86** for alternative WR timing during burst write.

The remaining sequence is as follows (with reference to the figure above):-

1. At the positive edge of EBU_CLK labelled as '1' above the first Burst Phase starts. As the state machine is currently in the command phase, the interface samples the WAIT input. This is sampled as "active". By coincidence, in this example, the Cellular RAM also deasserts it's WAIT output as a response to this clock edge to signal that it will start to take the data from the data bus on the BFCLKO rising clock edge after the next (i.e. the rising edge of BFCLKO labelled as '5' above) - this need not be the case.
2. At the positive edge of EBU_CLK labelled as '2' above the second programmed EBU_CLK period of the Burst Phase begins.
3. At the positive edge of EBU_CLK labelled as '3' above the Burst FLASH evaluates the WAIT sample from '1' above. As this sample was "active" the write data is not updated. As this clock edge is coincident with the end of a burst phase the WAIT input is resampled. The value of this new WAIT sample is "inactive".
4. At the positive edge of EBU_CLK labelled as '5' above the Burst FLASH again evaluates the WAIT sample from '3' above. As this sample was "in-active", and the edge is coincident with the end of a burst phase, the next data value is issued to the AD(15:0) pins and the next Burst Phase is started.

This process continues until all the data is written.

Fujitsu FCRAM Support (burst write with WR active during data phase)

The FCRAM device type can be supported in two ways. Later FCRAMs have a compatibility bit in the device configuration register which programmes the device to expect the WR signal to be active with the address and to be latched with the ADV signal. In this mode, FCRAM can be treated as an Infineon/Micron cellular RAM.

Alternatively, if a write is attempted to a region configured as a burst flash, the memory controller will generate a burst write with the WR signal asserted with the write data. This should be directly compatible with an FCRAM operating in its native mode.

12.11.16 Programmable Parameters

The following table lists the programmable parameters for burst flash accesses. These parameters only apply when the EBU_BUSCONx.agen parameter for a particular memory region is set for access to synchronous burst devices (page mode or otherwise).

Table 12-36 Burst Flash Access Programmable Parameters

Parameter	Function	Register
ADDRC	Number of cycles in Address Phase.	EBU_BUSAPx
AHOLDC	Number of cycles in Address Hold.	EBU_BUSAPx
CMDDELAY	Number of programmed Command Delay cycles.	EBU_BUSAPx
WAITRDC	Number of programmed wait states for read accesses.	EBU_BUSRAPx
WAITWRC	Number of programmed wait states for write accesses.	EBU_BUSWAPx
EXTDATA	Extended data	EBU_BUSAPx
RDRECOVC	Number of minimum recovery cycles after a read access when the next access is to the same region.	EBU_BUSRAPx
WRRECOVC	Number of minimum recovery cycles after a write access when the next access is to the same region.	EBU_BUSWAPx
RDDTACS	Number of minimum recovery cycles after a read access when the next access is to a different region.	EBU_BUSRAPx
WRDTACS	Number of minimum recovery cycles after a write access when the next access is to a different region.	EBU_BUSWAPx
WAIT	Sampling of $\overline{\text{WAIT}}$ input: OFF, SYNCHRONOUS, ASYNCHRONOUS or WAIT_CELLULAR_RAM	EBU_BUSCONx
FBBMSEL	Flash synchronous burst mode: CONTINUOUS or DEFINED (as in FETBLEN)	EBU_BUSCONX
FETBLEN	Synchronous burst length: SINGLE, BURST2, BURST4 or BURST8	EBU_BUSCONx
BFCMSEL	Flash Clock Mode, continuous or gated	EBU_BUSRCONx

Table 12-36 Burst Flash Access Programmable Parameters (cont'd)

Parameter	Function	Register
EXTCLOCK	Frequency of external clock at pin BFCLKO: equal, 1/2, 1/3 or 1/4 of EBU_CLK	EBU_BUSCONx
EBSE	delay $\overline{\text{ADV}}$ output to improve hold margin	EBU_BUSCONx
ECSE	delay $\overline{\text{CS}}$, $\overline{\text{WR}}$ and write data outputs to improve hold margin	EBU_BUSCONx
LOCKCS	enable locked write sequences for this region	EBU_BUSWCONx
FDBKEN	enable clock feedback to improve read data margins	EBU_BUSRCONx
BFSSS	disable second pipeline stage for clock feedback	EBU_BUSRCONx
DBA	disable alignment of read bursts on external bus	EBU_BUSRCONx
AAP	enable the "asynchronous address phase" mode.	EBU_BUSCONx
PORTW	memory port width	EBU_BUSRCONx
DATAC	Number of clock cycles of data hold	EBU_BUSWCONx

12.12 EBU Registers

This section describes the registers and programmable parameters of the EBU. All these registers can be read in User Mode, but can only be written in Supervisor Mode.

All registers are reset by the module reset.

Table 12-37 Registers Address Space -Flash Registers

Module	Base Address	End Address	Note
EBU	F800 0000_H	F800 $03FF_H$	-

Table 12-38 Registers Overview EBU Control Registers

Register Short Name	Register Long Name	Offset Address	Access Mode		Description see
			Read	Write	
EBU_CLC	EBU Clock Control Register	0000_H	U, SV, 32, 64	SV, 32, 64, E	Page 12-91
EBU_MODCON	EBU Configuration Register	0004_H	U, SV, 32, 64	SV, 32, 64	Page 12-93
EBU_MODID	EBU Module ID Register	0008_H	U, SV, 32, 64	SV, 32, 64	Page 12-111
EBU_USERCON	EBU Test/Control Configuration Register	$000C_H$	U, SV, 32, 64	SV, 32, 64	Page 12-111
EBU_EXTBOOT	EBU External Boot Control Register	0010_H	U,SV, 32	SV, 32[1)]	Page 12-95
	reserved	0014_H	BE	BE	
EBU_ADDRSEL0	EBU Address Select Register 0	0018_H	U, SV, 32, 64	SV, 32, 64	Page 12-96
EBU_ADDRSEL1	EBU Address Select Register 1	$001C_H$	U, SV, 32, 64	SV, 32, 64	Page 12-96
EBU_ADDRSEL2	EBU Address Select Register 2	0020_H	U, SV, 32, 64	SV, 32, 64	Page 12-96
EBU_ADDRSEL3	EBU Address Select Register 3	0024_H	U, SV, 32, 64	SV, 32, 64	Page 12-96
EBU_BUSRCON0	EBU Bus Configuration Register 0	0028_H	U, SV, 32, 64	SV, 32, 64	Page 12-98
EBU_BUSRAP0	EBU Bus Access Parameter Register 0	$002C_H$	U, SV, 32, 64	SV, 32, 64	Page 12-105

Table 12-38 Registers Overview EBU Control Registers

Register Short Name	Register Long Name	Offset Address	Access Mode		Description see
			Read	Write	
EBU_BUSWCON0	EBU Bus Configuration Register 0	0030_H	U, SV, 32, 64	SV, 32, 64	Page 12-102
EBU_BUSWAP0	EBU Bus Access Parameter Register 0	0034_H	U, SV, 32, 64	SV, 32, 64	Page 12-108
EBU_BUSRCON1	EBU Bus Configuration Register 1	0038_H	U, SV, 32, 46	SV, 32, 64	Page 12-98
EBU_BUSRAP1	EBU Bus Access Parameter Register 1	$003C_H$	U, SV, 32, 64	SV, 32, 64	Page 12-105
EBU_BUSWCON1	EBU Bus Configuration Register 1	0040_H	U, SV, 32, 64	SV, 32, 64	Page 12-102
EBU_BUSWAP1	EBU Bus Access Parameter Register 1	0044_H	U, SV, 32, 64	SV, 32, 64	Page 12-108
EBU_BUSRCON2	EBU Bus Configuration Register 2	0048_H	U, SV, 32, 64	SV, 32, 64	Page 12-98
EBU_BUSRAP2	EBU Bus Access Parameter Register 2	$004C_H$	U, SV, 32, 64	SV, 32, 64	Page 12-105
EBU_BUSWCON2	EBU Bus Configuration Register 2	0050_H	U, SV, 32, 64	SV, 32, 64	Page 12-102
EBU_BUSWAP2	EBU Bus Access Parameter Register 2	0054_H	U, SV, 32, 64	SV, 32, 64	Page 12-108
EBU_BUSRCON3	EBU Bus Configuration Register 3	0058_H	U, SV, 32, 64	SV, 32, 64	Page 12-98
EBU_BUSRAP3	EBU Bus Access Parameter Register 3	$005C_H$	U, SV, 32, 64	SV, 32, 64	Page 12-105
EBU_BUSWCON3	EBU Bus Configuration Register 3	0060_H	U, SV, 32, 64	SV, 32, 64	Page 12-102
EBU_BUSWAP3	EBU Bus Access Parameter Register 3	0064_H	U, SV, 32, 64	SV, 32, 64	Page 12-108
-	reserved	0068_H - $03FC_H$	BE	BE	

1) The EXTBOOT register is only writable when the BOOT_ACTIVE flag from the SCU is set.

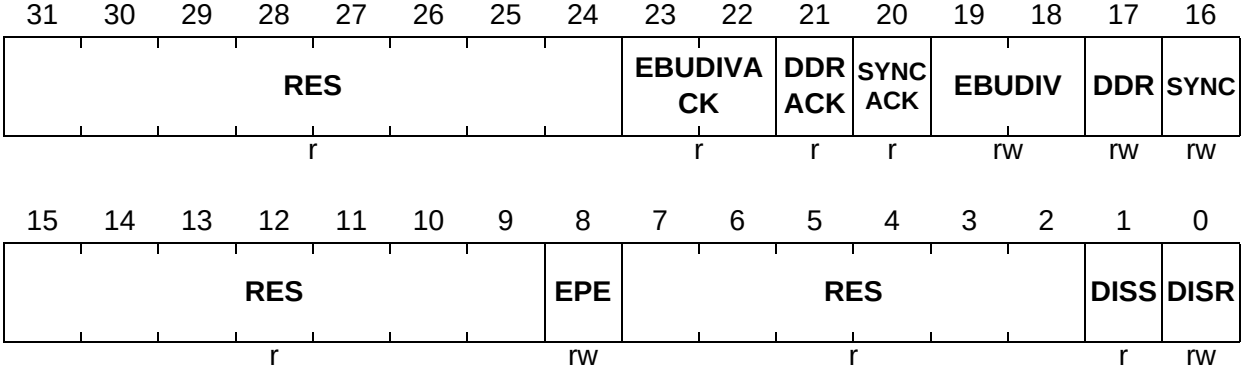
Access Restrictions

Note: The EBU registers are accessible only through word accesses or double word accesses. Half-word and byte accesses on EBU registers will generate a bus error. Accesses to a 64bit word only partially populated with accessible registers will cause a bus error

Note: Writing reserved values to register fields may cause unexpected system operation as the behaviour of the EBU using reserved values has not been verified.

12.12.1 Clock Control Register, CLC

EBU_CLC
EBU Clock Control Register (000_H) Reset Value: 0011 0000$_H$



Field	Bits	Type	Description
DISR	0	rw	**EBU Disable Request Bit** This bit is used for enable/disable control of the EBU. 0_B EBU disable is not requested 1_B EBU disable is requested
DISS	1	r	**EBU Disable Status Bit** DISS is always read as 0, as accessing the EBU will automatically enable it. 0_B EBU is enabled (default after reset) 1_B EBU is disabled
EPE	8	rw	**Endinit Protection Enable** 0_B Disable Endinit protection of the CLC register (default after reset). 1_B Enable Endinit protection of the CLC register.
RES	15:9	r	**Reserved** Read as 0; should be written with 0.

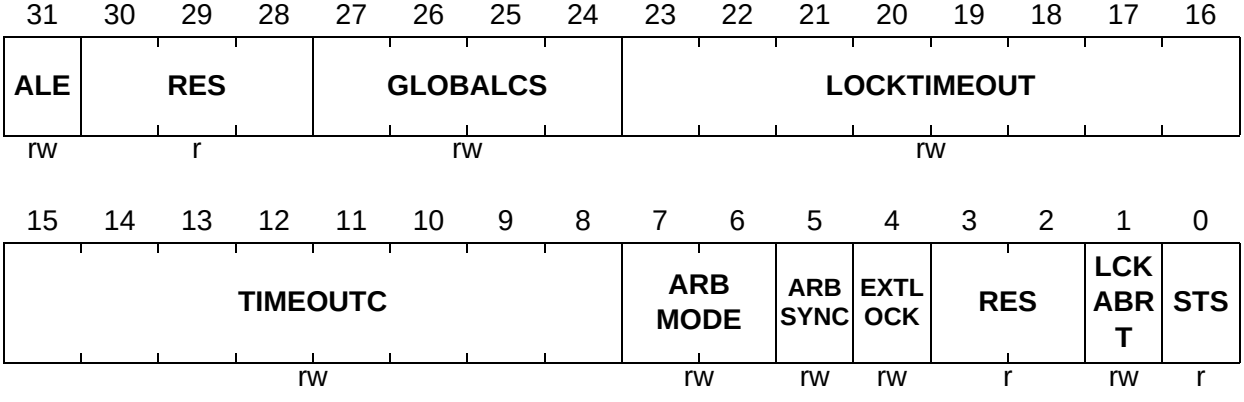
Field	Bits	Type	Description
SYNC	16	rw	**EBU Clocking Mode** Read as 1; must be written with 1.
DDR	17	rw	**DDR Clocking Mode** Read as 0; must be written with 0.
EBUDIV	19:18	rw	**EBU Clock Divide Ratio** 00_B request EBU to run off input clock (default after reset) 01_B request EBU to run off input clock divided by 2 10_B request EBU to run off input clock divided by 3 11_B request EBU to run off input clock divided by 4
SYNCACK	20	r	**EBU Clocking Mode Status** Read as 1; should be written with 1.
DDRACK	21	r	**DDR Clocking Mode Status** Read as 0; should be written with 0.
EBUDIVACK	23:22	r	**EBU Clock Divide Ratio Status** 00_B EBU is running off input clock (default after reset) 01_B EBU is running off input clock divided by 2 10_B EBU is running off input clock divided by 3 11_B EBU is running off input clock divided by 4
RES	31:24	r	**Reserved** Read as 0; should be written with 0.

Note: While the DISR bit is implemented in the EBU, it connects to the standby logic which will disable the clock tree. Standby mode will be exited automatically when an attempt is made to access the EBU. This register can be Endinit-protected after initialization. Writing to this register in this state will cause the EBU to generate an LMB Error.

12.12.2 Configuration Register, MODCON

EBU_MODCON
EBU Configuration Register **(004_H)**
(Internal boot) Reset Value: 0000 0020_H

EBU_MODCON
EBU Configuration Register **(004_H)**
(External boot) Reset Value: 0000 0060_H



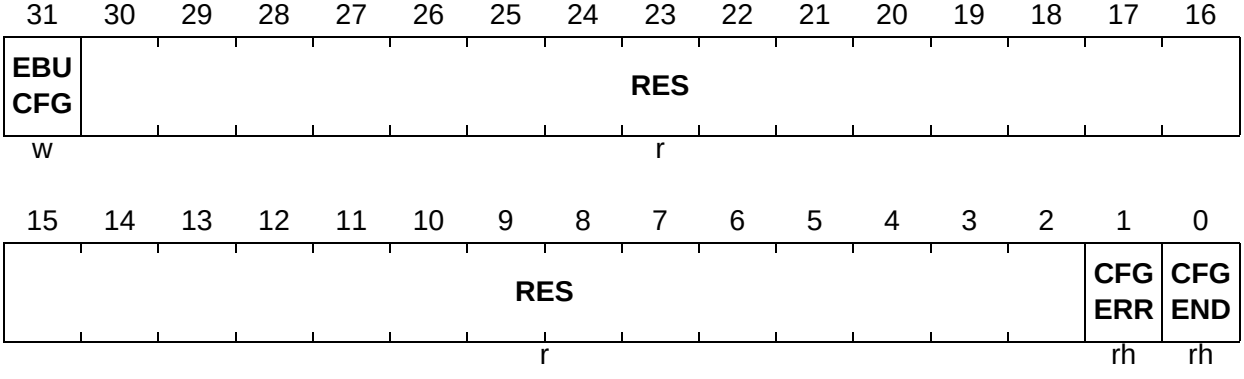
Field	Bits	Type	Description
STS	0	r	**Memory Status Bit** Software access to the $\overline{\text{WAIT}}$ input pin to the EBU.
LCKABRT	1	r	**Lock Abort** This is a status bit which is set when a chip select lock has been cancelled by a program fetch. The flag is cleared by writing 1_B to the field. See **Section 12.4.9** 0_B Lock function is operating normally 1_B Lock function has been aborted by processor instruction read from locked device
RES	3:2	r	**Reserved** Read as 0; should be written with 0.
EXTLOCK	4	rw	**External Bus Lock Control** 0_B External bus is not locked after the EBU gains ownership 1_B External bus is locked after the EBU gains ownership

Field	Bits	Type	Description
ARBSYNC	5	rw	**Arbitration Signal Synchronization Control** 0_B Arbitration inputs are synchronous. (one stage resynch) 1_B Arbitration inputs are asynchronous. (two stage resynch)
ARBMODE	7:6	rw	**Arbitration Mode Selection** 00_B No Bus arbitration mode selected 01_B Arbiter Mode arbitration mode selected 10_B Participant arbitration mode selected 11_B Sole Master arbitration mode selected
TIMEOUTC	15:8	rw	**Bus Time-out Control** This bit field determines the number of inactive cycles leading to a bus time-out after the EBU gains ownership. 00_H Time-out is disabled. 01_H Time-out is generated after 1 × 8 clock cycles. $...{}_H$ ... FF_H Time-out is generated after 255 × 8 clock cycles.
LOCKTIMEOUT	23:16	rw	**Lock Timeout Counter Preload** Value to be preloaded into the timeout counter for the arbitration lock. *Note: (see **Section 12.5.5**)*
GLOBALCS	27:24	rw	**Global Chip Select Enable** The bits of this bit field are used to enable the EBU chip select lines for the global chip select $\overline{CSGLB}$ generation. With x = 0-3, the GLOBALCS bits are defined as follows (see also **Page 12-12**): 0_B $\overline{CSCOMB}$ is not activated by an active $\overline{CSx}$. 1_B $\overline{CSCOMB}$ is activated when $\overline{CSx}$ becomes active.
RES	30:24	r	**Reserved** Read as 0; must be written with 0.
ALE	31	rw	**ALE Mode** Switch the $\overline{ADV}$ output to be an active high ALE signal instead of active low $\overline{ADV}$. 0_B Output is $\overline{ADV}$ 1_B Output is ALE

12.12.3 External Boot Configuration Control Register, EXTBOOT

EBU_EXTBOOT
EBU External Boot Configuration Register(00010_H)

Reset Value: $0000\ 0001_H$



Field	Bits	Type	Description
CFGEND	0	rh	**Configuration End** 0_B Configuration fetch is running 1_B Configuration fetch has completed or not started
CFGERR	1	rh	**Configuration Fetch Error** 0_B No error 1_B The configuration fetch has returned a word with all bits set. This indicates an unprogrammed or missing flash
EBUCFG	31	w	**Configuration Word Fetch** Write 1_B to trigger automatically set the EBU to arbiter arbitration mode and fetch a configuration word from external memory. Always reads 0_B
RES	30:2	r	**Reserved** Read as 0; must be written with 0.

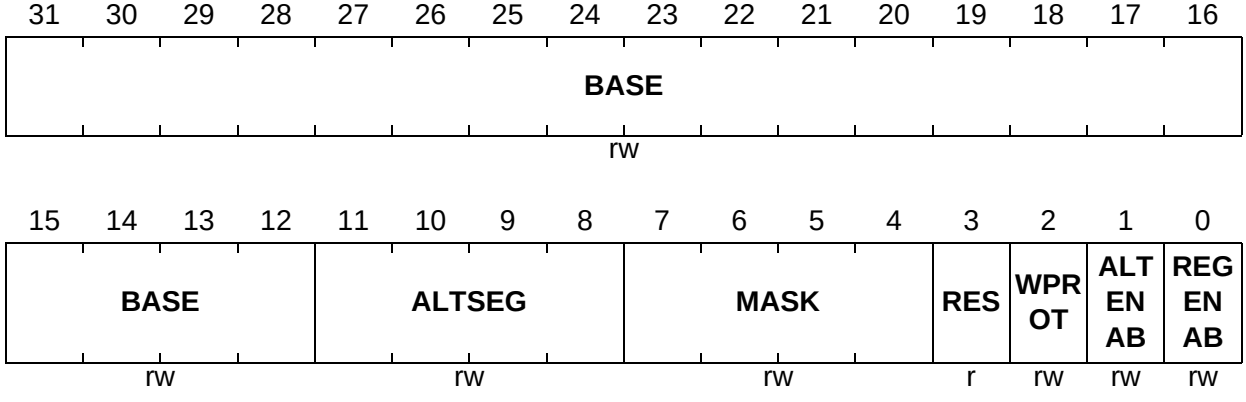
12.12.4 Address Select Register, ADDRSELx

EBU_ADDRSELx (x = 1-3)
EBU Address Select Register x (018_H+x*4_H)

Reset Value: 0000 0000_H

EBU_ADDRSEL0
EBU Address Select Register 0 (018_H)

Reset Value: A000 0001_H



Field	Bits	Type	Description
REGENAB	0	rw	**Memory Region Enable** 0_B Memory region is disabled (default after reset[1)]). 1_B Memory region is enabled.
ALTENAB	1	rw	**Alternate Segment Comparison Enable** 0_B ALTSEG is never compared to LMB address (default after reset). 1_B ALTSEG is always compared to LMB address.
WPROT	2	rw	**Memory Region Write Protect** 0_B Region is enabled for write accesses 1_B Region is write protected.
MASK	7:4	rw	**Memory Region Address Mask** Specifies the number of right-most bits in the base address starting at bit 26, which should be included in the address comparison. Bits [31:27] will always be part of the comparison.
ALTSEG	11:8	rw	**Memory Region Alternate Segment** Alternate segment to be compared to LMB address bit [31:28].

Field	Bits	Type	Description
BASE	31:12	rw	**Memory Region Base Address** Base address to be compared to LMB address in conjunction with the mask control.
RES	3	r	**Reserved** Read as 0; should be written with 0.

1) except for ADDRSEL0, **REGENAB in register ADDRSEL0 is 1_B after reset.**

12.12.5 Bus Configuration Register, BUSRCONx

EBU_BUSRCONx (x = 0-3)
EBU Bus Configuration Register (028_H+x*10_H)

Reset Value: 00D3 0040$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AGEN				RES	AAP	WAIT		PORTW		BCGEN		WAI TINV	DBA	EBSE	ECSE
rw				r	rw	rw		rw		rw		rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RES								NAA	BFC MSEL	FDB KEN	BFS SS	FBB MSEL	FETBLEN		
r								rw	rw	rw	rw	rw	rw		

Field	Bits	Type	Description
FETBLEN	2:0	rw	**Burst Length for Synchronous Burst** Defines maximum number of burst data cycles which are executed by Memory Controller during a burst access to a Synchronous Burst device. 000_B 1 data access (default after reset). 001_B 2 data accesses. 010_B 4 data accesses. 011_B 8 data accesses. $1xx_B$ reserved.
FBBMSEL	3	rw	**Synchronous burst buffer mode select** 0_B Burst buffer length defined by value in fetblen (default after reset). 1_B Continuous mode. All data required for transaction is transferred in a single burst.
BFSSS	4	rw	**Read Single Stage Synchronization:** The second read-data synchronization stage in the pad-logic can be bypassed. Reduces access latency at the expense of the maximum achievable operating frequency. 0_B Two stages of synchronisation used. (maximum margin) 1_B One stage of synchronisation used. (minimum latency)

Field	Bits	Type	Description
FDBKEN	5	rw	**Burst FLASH Clock Feedback Enable** 0_B BFCLK feedback not used. 1_B Incoming data and control signals (from the Burst FLASH device) are re-synchronised to the BFCLKI input.
BFCMSEL	6	rw	**Burst Flash Clock Mode Select** 0_B Burst Flash Clock runs continously with values selected by this register 1_B Burst Flash Clock is disabled between accesses
NAA	7	rw	**Reserved**
RES	15:8	r	**Reserved** 00_H Reserved Value
ECSE	16	rw	**Early Chip Select for Synchronous Burst** 0_B CS is delayed. 1_B CS is not delayed. (see **Control of ADV & Other Signal Delays During Asynchronous Accesses** and **Control of ADV & Control Signal Delays During Synchronous Accesses**)
EBSE	17	rw	**Early Burst Signal Enable for Synchronous Burst** 0_B ADV is delayed. 1_B ADV is not delayed. (see **Control of ADV & Other Signal Delays During Asynchronous Accesses** and **Control of ADV & Control Signal Delays During Synchronous Accesses**)

Field	Bits	Type	Description
DBA	18	rw	**Disable Burst Address Wrapping** 0_B Memory Controller automatically re-aligns any non-aligned synchronous burst access so that data can be fetched from the device in a single burst transaction. 1_B Memory Controller always starts any burst access to a synchronous burst device at the address specified by the LMB request. Any required address wrapping must be automatically provided by the Burst FLASH device. Care must be taken with the use of this feature. The Burst capable device must be programmed to wrap at the appropriate address boundary prior to selection of this mode. **“Critical Word First Read Accesses” on Page 12-79**
WAITINV	19	rw	**Reversed polarity at WAIT** 0_B **OFF**, input at WAIT pin is active low (default after reset). 1_B **Polarity reversed**, input at WAIT pin is active high. This bit has no effect when using Burst FLASH Data Handshake Mode
BCGEN	21:20	rw	**Byte Control Signal Control** This bit field selects the timing mode of the byte control signals. 00_B Byte control signals follow chip select timing. 01_B Byte control signals follow control signal timing ($\overline{RD}$, $RD/\overline{WR}$) (default after reset). 10_B Byte control signals follow write enable signal timing ($RD/\overline{WR}$ only). 11_B Reserved.
PORTW	23:22	rw	**Device Addressing Mode** See **Table 12-12** and **Table 12-14**

Field	Bits	Type	Description
WAIT	25:24	rw	**External Wait Control** Function of the WAIT input. This is specific to the device type (i.e. the agen field). **For Asynchronous Devices:** 0_D OFF (default after reset). 1_D Asynchronous input at WAIT. 2_D Synchronous input at WAIT. 3_D reserved. See **"External Extension of the Command Phase by WAIT" on Page 12-59** **For Synchronous Burst Devices:** 0_D OFF (default after reset). 1_D Wait for page load (Early WAIT). 2_D Wait for page load (WAIT with data). 3_D Abort and retry access. See **"External Cycle Control via the WAIT Input" on Page 12-81**
AAP	26	rw	**Asynchronous Address phase:** Enables an access mode for synchronous memories where the clock is not started until after the address hold phase. 0_B Clock is enabled at beginning of access. 1_B Clock is enabled at after address phase.
0	27	r	**Reserved, always 0**
AGEN	31:28	rw	**Device Type for Region** See **Section 12.4.4 Programmable Device Types**

*Note: When in external boot mode (see **Page 12-42**) the reset value of EBU_BUSRCON0 is overwritten automatically (subsequent to the release of reset) as a result of the external Boot Configuration Value fetch.*

12.12.6 Bus Write Configuration Register, BUSWCONx

EBU_BUSWCONx (x = 0-3)
EBU Bus Write Configuration Register(030_H+x*10_H)

Reset Value: 00D3 0000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AGEN				LOCKCS	AAP	WAIT		PORTW		BCGEN		WAITINV	RES	EBSE	ECSE
rw				rw	rw	rw		r		rw		rw	r	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RES								NAA	RES			FBBMSEL	FETBLEN		
r								r	r			rw	rw		

Field	Bits	Type	Description
FETBLEN	2:0	rw	**Burst Length for Synchronous Burst** Defines maximum number of burst data cycles which are executed by Memory Controller during a burst access to a Synchronous Burst device. 000_B 1 data access (default after reset). 001_B 2 data accesses. 010_B 4 data accesses. 011_B 8 data accesses. $1xx_B$ reserved.
FBBMSEL	3	rw	**Synchronous burst buffer mode select** 0_B Burst buffer length defined by value in FETBLEN (default after reset). 1_B Continuous mode. All data required for transaction transferred in single burst
RES	6:4	r	**Reserved, always reads 0**
NAA	7	r	**Reserved**
RES	15:8	r	**Reserved** 00_H Reserved Value

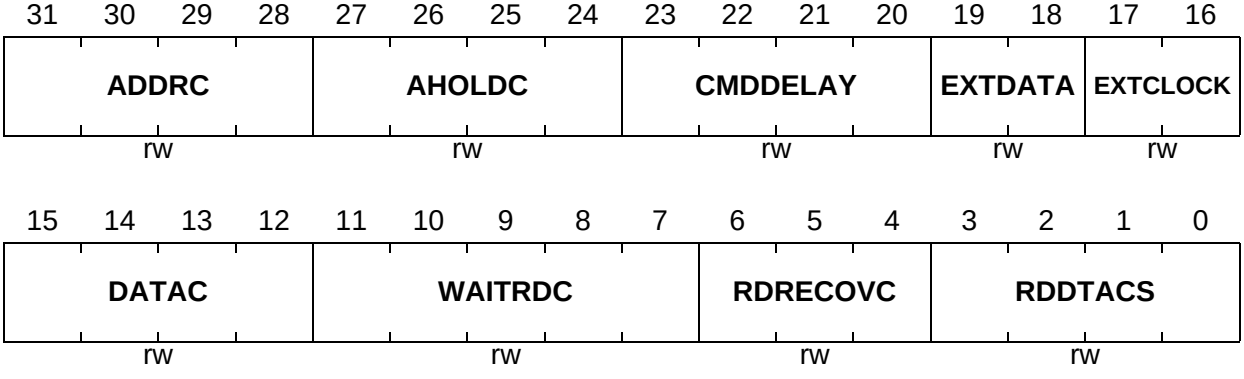
Field	Bits	Type	Description
ECSE	16	rw	**Early Chip Select for Synchronous Burst** 0_B CS is delayed. 1_B CS is not delayed. (see **Control of ADV & Other Signal Delays During Asynchronous Accesses** and **Control of ADV & Control Signal Delays During Synchronous Accesses**)
EBSE	17	rw	**Early Burst Signal Enable for Synchronous Burst** 0_B ADV is delayed. 1_B ADV is not delayed. (see **Control of ADV & Other Signal Delays During Asynchronous Accesses** and **Control of ADV & Control Signal Delays During Synchronous Accesses**)
RES	18	r	**Reserved, always reads 0**
WAITINV	19	rw	**Reversed polarity at WAIT** 0_B **OFF**, input at WAIT pin is active low (default after reset). 1_B **Polarity reversed**, input at WAIT pin is active high. This bit has no effect when using Burst FLASH Data Handshake Mode
BCGEN	21:20	rw	**Byte Control Signal Control** This bit field selects the timing mode of the byte control signals. 00_B Byte control signals follow chip select timing. 01_B Byte control signals follow control signal timing ($\overline{RD}$, $\overline{RD/WR}$) (default after reset). 10_B Byte control signals follow write enable signal timing ($\overline{RD/WR}$ only). 11_B Reserved.
PORTW	23:22	r	**Device Addressing Mode** See **Table 12-12** and **Table 12-13**

Field	Bits	Type	Description
WAIT	25:24	rw	**External Wait Control** Function of the WAIT input. This is specific to the device type (i.e. the agen field). **For Asynchronous Devices:** 0_D OFF (default after reset). 1_D Asynchronous input at WAIT. 2_D Synchronous input at WAIT. 3_D reserved. See **"External Extension of the Command Phase by WAIT" on Page 12-59** **For Synchronous Burst Devices:** 0_D OFF (default after reset). 1_D Wait for page load (Early WAIT). 2_D Wait for page load (WAIT with data). 3_D Abort and retry access. See **"External Cycle Control via the WAIT Input" on Page 12-81**
AAP	26	rw	**Asynchronous Address phase:** Enables an access mode for synchronous memories where the clock is not started until after the address hold phase. 0_B Clock is enabled at beginning of access. 1_B Clock is enabled at after address phase.
LOCKCS	27	rw	**Lock Chip Select** Enable Chip Select for Automatic Locking in the event of a write access 0_B Chip Select cannot be locked (default after reset). 1_B Chip Select will be automatically locked when written to from the processor data port.
AGEN	31:28	rw	**Device Type for Region** See **Section 12.4.4 Programmable Device Types**

*Note: When in external boot mode (see **Page 12-42**) the reset value of BUSCON0 is overwritten automatically (subsequent to the release of reset) as a result of the external Boot Configuration Value fetch.*

12.12.7 Bus Read Access Parameter Register, BUSRAPx

EBU_BUSRAPx (x = 0-3)
EBU Bus Read Access Parameter Register
($02C_H$+x*10_H) **Reset Value: FFFF $FFFF_H$**



Field	Bits	Type	Description
RDDTACS	3:0	rw	**Recovery Cycles between Different Regions** This bit field determines the number of clock cycles of the Recovery Phase between consecutive accesses directed to different regions or different types of access. See **"Recovery Phase (RP)" on Page 12-51** 0000_B No Recovery Phase clock cycles available. 0001_B 1 clock cycle selected. $\ldots_B$ … 1110_B 14 clock cycles selected. 1111_B 15 clock cycles selected.
RDRECOVC	6:4	rw	**Recovery Cycles after Read Accesses** This bit field determines the basic number of clock cycles of the Recovery Phase at the end of read accesses. 000_B No Recovery Phase clock cycles available. 001_B 1 clock cycle selected. $\ldots_B$ … 110_B 6 clock cycles selected. 111_B 7 clock cycles selected.

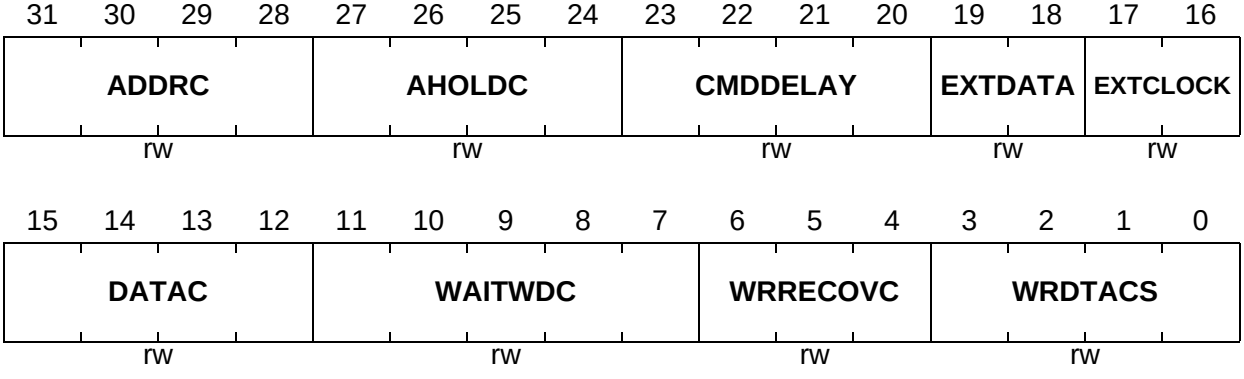
Field	Bits	Type	Description
WAITRDC	11:7	rw	**Programmed Wait States for read accesses** Number of programmed wait states for read accesses. For synchronous accesses, this will always be adjusted so that the phase exits on a rising edge of the external clock. 00000_B1 wait state. 00001_B1 wait states. 00010_B2 wait state. $\ldots_B$ … 11110_B30 wait states. 11111_B31 wait states.
DATAC	15:12	rw	**Data Hold Cycles for Read Accesses** This bit field determines the basic number of Data Hold phase clock cycles during read accesses. It has no effect in the current implementation $xxxx_B$ No data hold clock cycles available.
EXTCLOCK	17:16	rw	**Frequency of external clock at pin BFCLKO** 00_B Equal to EBU_CLK frequency. 01_B 1/2 of EBU_CLK frequency. 10_B 1/3 of EBU_CLK frequency. 11_B 1/4 of EBU_CLK frequency (default after reset). See **“Burst Flash Clock” on Page 12-72**.
EXTDATA	19:18	rw	**Extended data** See **Burst Phase (BP)** 00_B external memory outputs data every BFCLK cycle 01_B external memory outputs data every two BFCLK cycles 10_B external memory outputs data every four BFCLK cycles 11_B external memory outputs data every eight BFCLK cycles
CMDDELAY	23:20	rw	**Command Delay Cycles** This bit field determines the basic number of Command Delay phase clock cycles. 0000_B0 clock cycle selected. 0001_B1 clock cycle selected. $\ldots_B$ … 1110_B14 clock cycles selected. 1111_B15 clock cycles selected.

Field	Bits	Type	Description
AHOLDC	27:24	rw	**Address Hold Cycles** This bit field determines the number of clock cycles of the address hold phase.. 0000_B0 clock cycles selected 0001_B1 clock cycle selected $\cdots_B$ … 1110_B14 clock cycles selected 1111_B15 clock cycles selected
ADDRC	31:28	rw	**Address Cycles** This bit field determines the number of clock cycles of the address phase. 0000_B1 clock cycle selected 0001_B1 clock cycle selected $\cdots_B$ … 1110_B14 clock cycles selected 1111_B15 clock cycles selected

*Note: When in external boot mode (see **Page 12-42**), the reset value of EBU_BUSRAP0 is overwritten automatically (subsequent to the release of reset) as a result of the external Boot Configuration Value fetch.*

12.12.8 Bus Write Access Parameter Register, BUSWAPx

EBU_BUSWAPx (x = 0-3)
EBU Bus Write Access Parameter Register
(034_H+x*10_H) **Reset Value: FFFF $FFFF_H$**



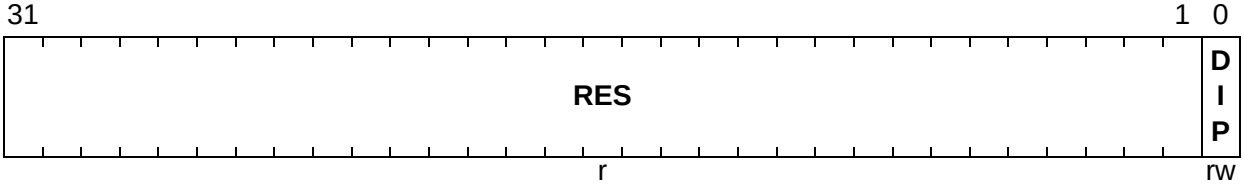
Field	Bits	Type	Description
WRDTACS	3:0	rw	**Recovery Cycles between Different Regions** This bit field determines the number of clock cycles of the Recovery Phase between consecutive accesses directed to different regions or different types of access. See **"Recovery Phase (RP)" on Page 12-51** 0000_B No Recovery Phase clock cycles available. 0001_B 1 clock cycle selected. $\ldots_B$ … 1110_B 14 clock cycles selected. 1111_B 15 clock cycles selected.
WRRECOVC	6:4	rw	**Recovery Cycles after Write Accesses** This bit field determines the basic number of clock cycles of the Recovery Phase at the end of write accesses. 000_B No Recovery Phase clock cycles available. 001_B 1 clock cycle selected. $\ldots_B$ … 110_B 6 clock cycles selected. 111_B 7 clock cycles selected.

Field	Bits	Type	Description
WAITWRC	11:7	rw	**Programmed Wait States for write accesses** Number of programmed wait states for write accesses. For synchronous accesses, this will always be adjusted so that the phase exits on a rising edge of the external clock. 00000_B 1 wait state. 00001_B 1 wait states. 00010_B 2 wait state. $\ldots_B$ … 11110_B 30 wait states. 11111_B 31 wait states.
DATAC	15:12	rw	**Data Hold Cycles for Write Accesses** This bit field determines the basic number of Data Hold phase clock cycles during write accesses. 0000_B No Recovery Phase clock cycles available. 0001_B 1 clock cycle selected. $\ldots_B$ … 1110_B 14 clock cycles selected. 1111_B 15 clock cycles selected.
EXTCLOCK	17:16	rw	**Frequency of external clock at pin BFCLKO** 00_B Equal to EBU_CLK frequency. 01_B 1/2 of EBU_CLK frequency. 10_B 1/3 of EBU_CLK frequency. 11_B 1/4 of EBU_CLK frequency (default after reset). See **“Burst Flash Clock” on Page 12-72**.
EXTDATA	19:18	rw	**Extended data** See **Burst Phase (BP)** 00_B external memory outputs data every BFCLK cycle 01_B external memory outputs data every two BFCLK cycles 10_B external memory outputs data every four BFCLK cycles 11_B external memory outputs data every eight BFCLK cycles

Field	Bits	Type	Description
CMDDELAY	23:20	rw	**Command Delay Cycles** This bit field determines the basic number of Command Delay phase clock cycles. 0000_B0 clock cycle selected. 0001_B1 clock cycle selected. $\ldots_B$ … 1110_B14 clock cycles selected. 1111_B15 clock cycles selected.
AHOLDC	27:24	rw	**Address Hold Cycles** This bit field determines the number of clock cycles of the address hold phase.. 0000_B0 clock cycles selected 0001_B1 clock cycle selected $\ldots_B$ … 1110_B14 clock cycles selected 1111_B15 clock cycles selected
ADDRC	31:28	rw	**Address Cycles** This bit field determines the number of clock cycles of the address phase. 0000_B1 clock cycle selected 0001_B1 clock cycle selected $\ldots_B$ … 1110_B14 clock cycles selected 1111_B15 clock cycles selected

12.12.9 Test/Control Configuration Register, USERCON

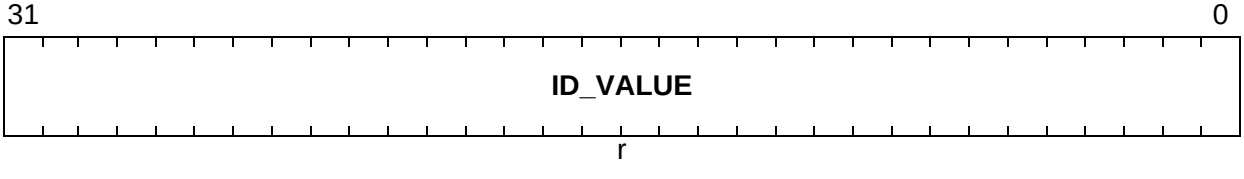
EBU_USERCON
EBU Test/Control Configuration Register
($00C_H$) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
DIP	0	rw	**Disable Internal Pipelining** 0_B The EBU can accept a new LMB transaction before the previous LMB transaction has been completed. 1_B The EBU will issue an LMB re-try if an LMB access is received while a previous LMB transaction is still not completed.
RES	[31:1]	r	**Reserved** All bits read as 0_B and should be written with 0_B.

12.12.10 Module Identification Register

EBU_ID
EBU Module Identification Register **(08_H)** **Reset Value: $0014\ C009_H$**



Field	Bits	Type	Description
ID_VALUE	[31:0]	r	**Module Identification Value**

13 Interrupt System

The TC1797 interrupt system provides a flexible and time-efficient means of processing interrupts. This chapter describes the interrupt system for the TC1797. Topics covered include the architecture of the interrupt system, interrupt system configuration, and the interrupt operations of the TC1797 peripherals and Central Processing Unit (CPU). General information is also given about the Peripheral Control Processor (PCP).

13.1 Overview

An interrupt request can be serviced either by the CPU or by the PCP. Interrupt requests are called “service requests” rather than “interrupt requests” in this document because they can be serviced by either one of the service providers.

Each peripheral in the TC1797 can generate service requests. Additionally, the Bus Control Units, the Debug Unit, the PCP, and even the CPU itself can generate service requests to either of the two service providers.

As shown in **Figure 13-1**, each TC1797 unit that can generate service requests is connected to one or more Service Request Nodes (SRNs). Each SRN contains a Service Request Control Register mod_SRCx, where “mod” is the identifier of the service requesting unit and “x” an optional index. Two arbitration buses connect the SRNs with two Interrupt Control Units (ICUs), which handle interrupt arbitration among competing interrupt service requests, as follows:

- The Interrupt Control Unit (ICU) arbitrates service requests for the CPU and administers the CPU interrupt arbitration bus.
- The Peripheral Interrupt Control Unit (PICU) arbitrates service requests for the PCP and administers the PCP interrupt arbitration bus.

The PCP can make service requests directly to itself (via the PICU), or it can make service requests to the CPU. The Debug Unit can generate service requests to the PCP or the CPU. The CPU can make service requests directly to itself (via the ICU), or it can make service requests to the PCP. The CPU Service Request Nodes are activated through software.

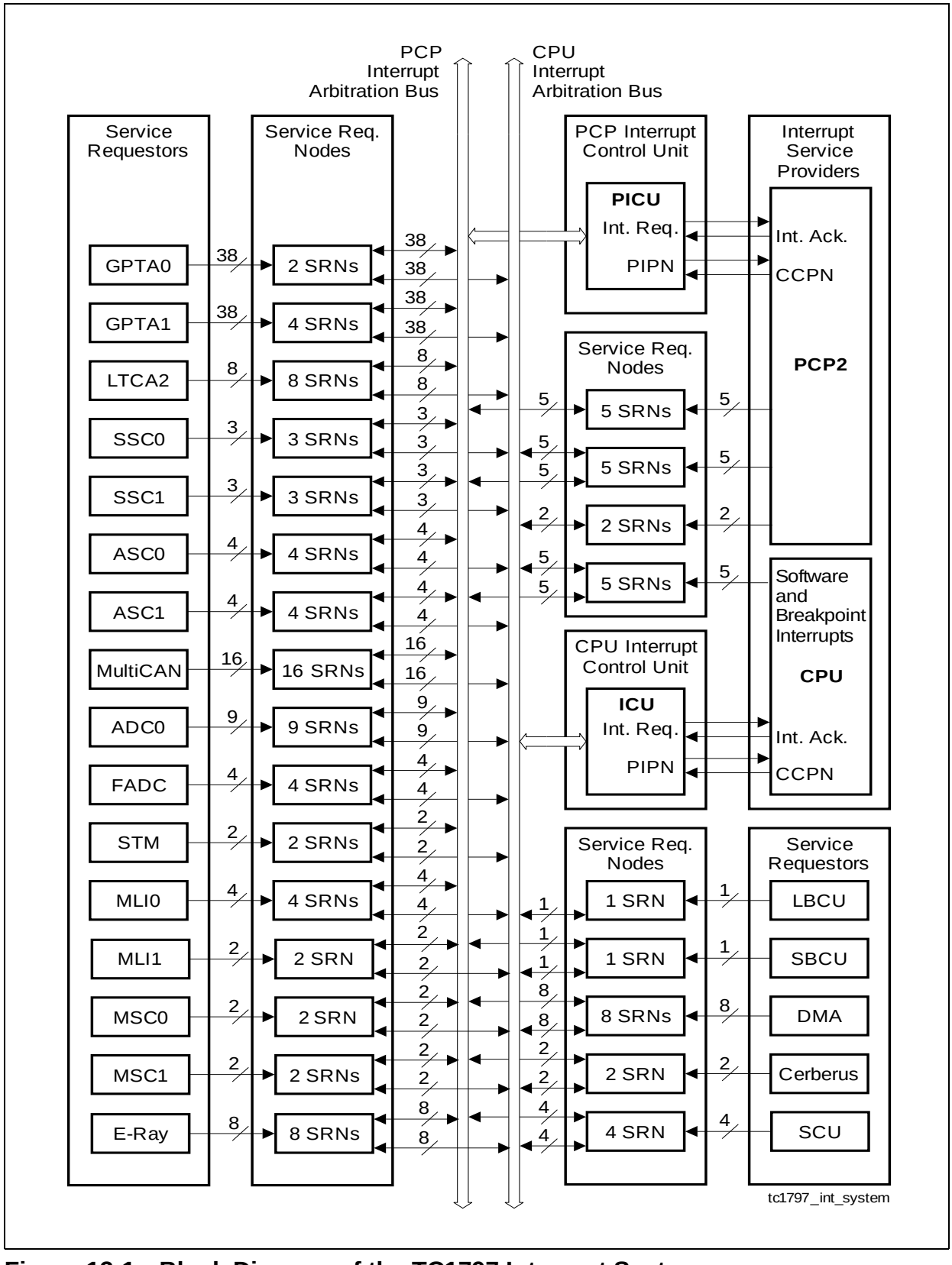


Figure 13-1 Block Diagram of the TC1797 Interrupt System

13.2 Service Request Nodes

Each SRN contains a Service Request Control Register and interface logic that connects it to the triggering unit and to the two interrupt arbitration buses. Some peripheral units of the TC1797 have multiple SRNs.

13.2.1 Service Request Control Registers

All Service Request Control Registers in the TC1797 have the same format. In general, these registers contain:

- Enable/disable information
- Priority information
- Service provider destination
- Service request status bit
- Software-initiated service request set and reset bits

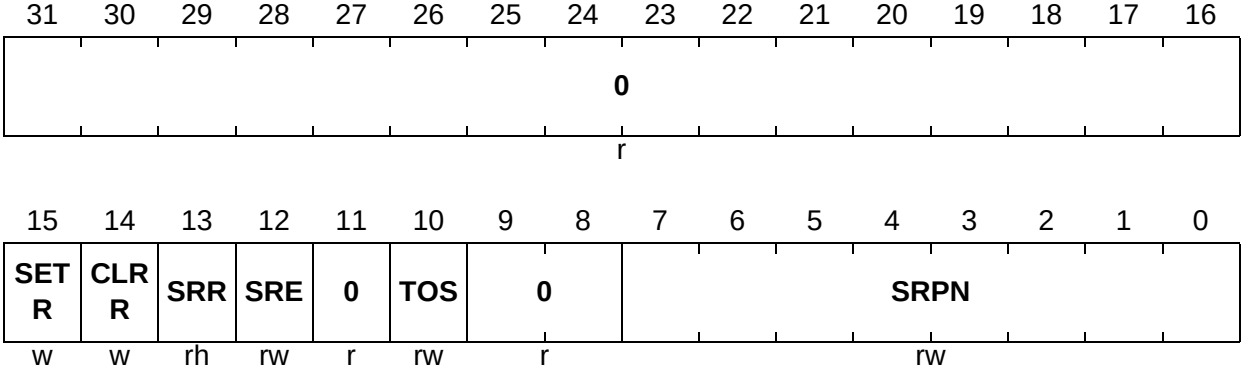
Besides being activated by the associated triggering unit through hardware, each SRN can also be set or reset by software via two software-initiated service request control bits.

The description given in this chapter characterizes all Service Request Control Registers of the TC1797. Information on peripheral module interrupt functions such as enable or request flags is provided in the corresponding sections of the module chapters.

13.2.1.1 General Service Request Control Register Format

The description given in this chapter characterizes all Service Request Control Registers of the TC1797. Information on peripheral module interrupt functions such as enable or request flags is provided in the corresponding sections of the module chapters.

mod_SRC
Service Request Control Register (00_H) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
SRPN	[7:0]	rw	**Service Request Priority Number** 00_H Service request is never serviced 01_H Service request is on lowest priority … … FF_H Service request is on highest priority
TOS	10	rw	**Type of Service Control** 0 CPU service is initiated 1 PCP request is initiated
SRE	12	rw	**Service Request Enable** 0 Service request is disabled 1 Service request is enabled
SRR	13	rh	**Service Request Flag** 0 No service request is pending 1 A service request is pending
CLRR	14	w	**Request Clear Bit** CLRR is required to reset SRR. 0 No action 1 Clear SRR; bit value is not stored; read always returns 0; no action if SETR is set also.
SETR	15	w	**Request Set Bit** SETR is required to set SRR. 0 No action 1 Set SRR; bit value is not stored; read always returns 0; no action if CLRR is set also.
0	[9:8], 11, [31:16]	r	**Reserved** Read as 0; should be written with 0.

13.2.1.2 Request Set and Clear Bits (SETR, CLRR)

The SETR and CLRR bits allow software to set or clear the service request bit SRR. Writing a 1 to SETR causes bit SRR to be set to 1. Writing a 1 to CLRR causes bit SRR to be cleared to 0. If hardware attempts to modify SRR during a read-modify-write software operation (such as the bit-set or bit-clear instructions), the software operation will succeed and the hardware operation will have no effect.

The value written to SETR or CLRR is not stored. Writing a 0 to these bits has no effect. These bits always return 0 when read. If both, SETR and CLRR, are set to 1 at the same time, SRR is not changed.

13.2.1.3 Enable Bit (SRE)

The SRE bit enables an interrupt to take part in the arbitration for the selected service provider. It does not enable or disable the setting of the request flag SRR; the request flag can be set by hardware or by software (via SETR) independent of the state of the SRE bit. This allows service requests to be handled automatically by hardware or through software polling.

If SRE = 1, pending service requests are passed on to the designated service provider for interrupt arbitration. The SRR bit is automatically set to 0 by hardware when the service request is acknowledged and serviced. It is recommended that in this case, software should not modify the SRR bit to avoid unexpected behavior due to the hardware controlling this bit.

If SRE = 0, pending service requests are not passed on to service providers. Software can poll the SRR bit to check whether a service request is pending. To acknowledge the service request, the SRR bit must then be reset by software by writing a 1 to CLRR.

Note: In this document, ‘active source’ means an SRN whose Service Request Control Register has its request enable bit SRE set to 1 to allow its service requests to participate in interrupt arbitration.

13.2.1.4 Service Request Flag (SRR)

When set, the SRR flag indicates that a service request is pending. It can be set or reset directly by hardware or indirectly through software using the SETR and CLRR bits. Writing directly to this bit via software has no effect.

The SRR status bit can be directly set or reset by the associated hardware. For instance, in the General Purpose Array Unit, an associated timer event can cause this bit to be set to 1. The details of how hardware events can cause the SRR bit to be set are defined in the individual peripheral/module chapters.

The acknowledgment of the service request by either the Interrupt Control Unit (ICU) or the PCP Interrupt Control Unit (PICU) causes the SRR bit to be cleared.

SRR can be set or cleared either by hardware or by software regardless of the state of the enable bit SRE. However, the request is only forwarded for service if the enable bit is set. If SRE = 1, a pending service request takes part in the interrupt arbitration of the service provider selected by the device's TOS bit. If SRE = 0, a pending service request is excluded from interrupt arbitrations.

SRR is automatically reset by hardware when the service request is acknowledged and serviced. Software can poll SRR to check for a pending service request. SRR must be reset by software in this case by writing a 1 to CLRR.

It is not advisable to clear a pending service request flag SRR (writing CLRR = 1) and enable the corresponding service request node SRN (writing SRE = 1) simultaneously at the same write access to the Service Request Control Register. If this should happen, an unintended interrupt request may be generated. Instead of executing one write access, it is recommended to split the two actions into two consecutive write accesses to the corresponding Service Request Control Register, starting with the clearing of the pending interrupt flag and followed by the enabling of the service request node.

13.2.1.5 Type-Of-Service Control (TOS)

There are two service providers for service requests in the TC1797, the CPU and the PCP. The TOS bit is used to select whether a service request generates an interrupt to the CPU (TOS = 0) or to the PCP (TOS = 1). Bit 11 of the Service Request Control Register is read-only, returning 0 when read. Writing to this bit position has no effect. However, to ensure compatibility with future extensions, bit 11 should always be written with a 0.

Note that several Service Request Control Registers (e.g. in the PCP) have a hardwired TOS bit (0 or 1) that cannot be written. These registers can only generate an interrupt to one dedicated service provider (PCP or CPU).

Note: Before modifying the content of a TOS bit, the corresponding SRN must be disabled (SRE = 0).

13.2.1.6 Service Request Priority Number (SRPN)

The 8-bit Service Request Priority Number (SRPN) indicates the priority of a service request with respect to other sources requesting service from the same service provider, and with respect to the priority of the service provider itself.

Each active source selecting the same service provider must have a unique SRPN value to differentiate its priority. The special SRPN value of 00_H excludes an SRN from taking part in arbitration, regardless of the state of its SRE bit. The SRPN values for active sources selecting different service providers (CPU vs. PCP) may overlap. If a source is not active – meaning its SRE bit is 0 – no restrictions are applied to the service request priority number.

The SRPN is used by service providers to select an Interrupt Service Routine (ISR)) or Channel Program (in case of the PCP) to service the request. ISRs are associated with Service Request Priority Numbers by an Interrupt Vector Table located in each service provider. This means that the TC1797 Interrupt Vector Table is ordered by priority number. This is unlike traditional interrupt architectures in which their interrupt vector tables are ordered by the source of the interrupt. The TC1797 Interrupt Vector Table allows a single peripheral to have multiple priorities for different purposes.

The range of values for SRPNs used in a system depends on the number of possible active service requests and the user-definable organization of the Interrupt Vector Table. The 8-bit SRPNs permit up to 255 sources to be active at one time (remembering that the special SRPN value of 00_H excludes an SRN from taking part in arbitration).

Note: Before modifying the content of an SRPN bit field, the corresponding SRN must be disabled (SRE = 0).

SRPNs in the TC1797

In the TC1797, interrupt sources selecting the same Service Provider are also allowed to have identical SRPN values. In this case, the software (interrupt service routine) must check which of the interrupt sources with identical SRPN has become active.

Note that module-specific interrupt request flags must be available because the SRR flags cannot be used for this check. SRR flags (meaning all SRR flags of interrupts with identical SRPN values) are in general automatically reset by hardware when a service request is acknowledged and serviced.

Note: This practice with identical SRPN values is not recommended as it is not portable to other TriCore devices.

13.3 Interrupt Control Units

The Interrupt Control Units manage the interrupt system, arbitrate incoming service requests, and determine whether and when to interrupt the service provider. The TC1797 contains two interrupt control units, one for the CPU (called ICU), and one for the PCP (called PICU). Each one controls its associated interrupt arbitration bus and manages the communication with its service provider (see **Figure 13-1**).

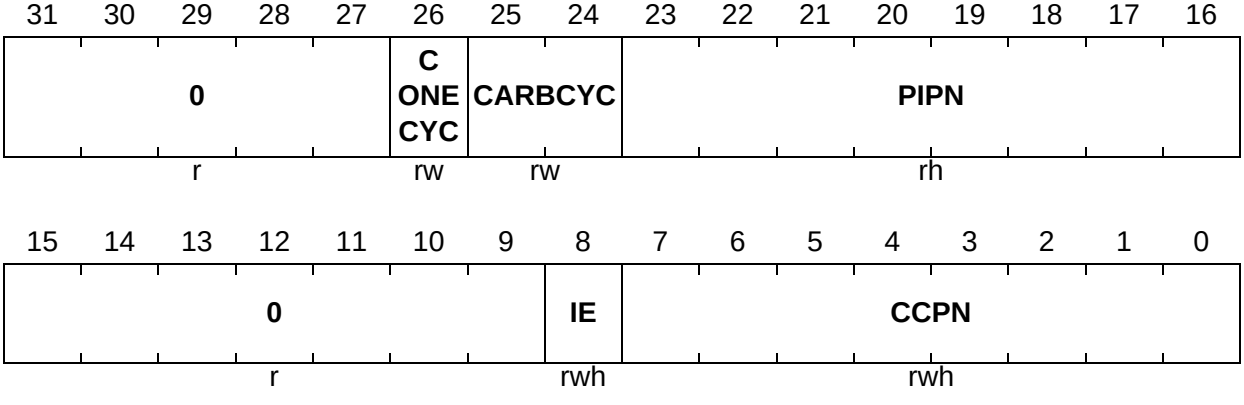
13.3.1 Interrupt Control Unit (ICU)

This section describes the interrupt control unit (ICU) for the CPU.

13.3.1.1 ICU Interrupt Control Register (ICR)

The ICU Interrupt Control Register ICR holds the current CPU priority number (CCPN), the global interrupt enable/disable bit (IE), the pending interrupt priority number (PIPN), and bit fields which control the interrupt arbitration process.

ICR
ICU Interrupt Control Register (F7E1FE2C$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
CCPN	[7:0]	rwh	**Current CPU Priority Number** The Current CPU Priority Number (CCPN) bit field indicates the current priority level of the CPU. It is automatically updated by hardware on entry and exit of interrupt service routines, and through the execution of a BISR instruction. CCPN can also be updated through an MTCR instruction.

Field	Bits	Type	Description
IE	8	rwh	**Global Interrupt Enable Bit** The interrupt enable bit globally enables the CPU service request system. Whether or not a service request is delivered to the CPU depends on the individual Service Request Enable Bits (SRE) in the SRNs, and the current state of the CPU. IE is automatically updated by hardware on entry and exit of an Interrupt Service Routine (ISR). IE is cleared to 0 when an interrupt is taken, and is restored to the previous value when the ISR executes an RFE instruction to terminate itself. IE can also be updated through the execution of the ENABLE, DISABLE, MTCR, and BISR instructions. 0 Interrupt system is globally disabled 1 Interrupt system is globally enabled
PIPN	[23:16]	rh	**Pending Interrupt Priority Number** PIPN is a read-only bit field that is updated by the ICU at the end of each interrupt arbitration process. It indicates the priority number of the pending service request. PIPN is set to 0 when no request is pending, and at the beginning of each new arbitration process. 00_H No valid pending request YY_H A request with priority YY_H is pending
CARBCYC	[25:24]	rw	**Number of Arbitration Cycles** CARBCYC controls the number of arbitration cycles used to determine the request with the highest priority. 00_B 4 arbitration cycles (default) 01_B 3 arbitration cycles 10_B 2 arbitration cycles 11_B 1 arbitration cycle
CONECYC	26	rw	**Number of Clocks per Arbitration Cycle Control** The CONECYC bit determines the number of system clocks per arbitration cycle. This bit should only be set to 1 for system designs utilizing low system clock frequencies. 0 2 clocks per arbitration cycle (default) 1 1 clock per arbitration cycle
0	[15:9], [31:27]	r	**Reserved** Read as 0; should be written with 0.

13.3.1.2 Operation of the Interrupt Control Unit (ICU)

Service-request arbitration is performed in the ICU in parallel with normal CPU operation. When a triggering event occurs in one or more interrupt sources, the associated SRNs, if enabled, send service requests to the CPU through the ICU. The ICU determines which service request has the highest priority. The ICU will then forward the service request to the CPU. The service request will be acknowledged by the CPU and serviced, depending upon the state of the CPU.

The ICU arbitration process takes place in one or more arbitration cycles over the CPU interrupt arbitration bus. The ICU begins a new arbitration process when a new service request is detected. At the end of the arbitration process, the ICU will have determined the service request with the highest priority number. This number is stored in the ICR.PIPN bit field and becomes the pending service request.

After the arbitration process, the ICU forwards the pending service request to the CPU by attempting to interrupt it. The CPU can be interrupted only if interrupts are enabled globally (that is, ICR.IE = 1) and if the priority of the service request is higher than the current processor priority (ICR.PIPN > ICR.CCPN). Also, the CPU may be temporarily blocked from taking interrupts, for example, if it is executing a multi-cycle instruction such as an atomic read-modify-write operation. The full list of conditions which could block the CPU from immediately responding to an interrupt request generated by the ICU is:

- Current CPU priority, ICR.CCPN, is equal to or higher than the pending interrupt priority, ICR.PIPN
- Interrupt system is globally disabled (ICR.IE = 0)
- CPU is in the process of entering an interrupt- or trap-service routine
- CPU is executing non-interruptible trap services
- CPU is executing a multi-cycle instruction
- CPU is executing an instruction which modifies the conditions of the global interrupt system, such as modifying the ICR
- CPU detects a trap condition (such as context depletion) when trying to enter a service routine

When the CPU is not otherwise prevented from taking an interrupt, the CPU’s program counter will be directed to the Interrupt Service Routine entry point associated with the priority of the service request. Next, the CPU saves the value of ICR.PIPN internally, and acknowledges the ICU. The ICU then forwards the acknowledge signal back to the SRN that is requesting service in order to inform it that it will be serviced by the CPU. The SRR bit in this SRN is then reset to 0.

After sending the acknowledgement, the ICU resets ICR.PIPN to 0 and may start a new arbitration process if there is another pending interrupt request. If not, ICR.PIPN remains at 0 and the ICU enters an idle state, waiting for the next interrupt request to awaken it. If there is a new service request waiting, the priority number of the new request will be written to ICR.PIPN at the end of the new arbitration process and the ICU will deliver the pending interrupt to the CPU according to the rules described in this section.

If a new service request is received by the ICU before the CPU has acknowledged the pending interrupt request, the ICU deactivates the pending request and starts a new arbitration process. This reduces the latency of service requests posted before the current request is acknowledged. The ICU deactivates the current pending interrupt request by setting the ICR.PIPN bit field to 0, indicating that the ICU has not yet found a new valid pending request. It then executes its arbitration process again. If the new service request has a higher priority than the previous one, its priority will be written to ICR.PIPN. If the new interrupt has a lower priority, the priority of the previous interrupt request will again be written to ICR.PIPN. In any case, the ICU will deliver a new interrupt request to the CPU according to the rules described in this section.

Once the CPU has acknowledged the current pending interrupt request, any new service request generated by an SRN must wait at least until the end of the next service request arbitration process to be serviced.

Essentially, arbitration in the ICU is performed whenever a new service request is detected, regardless of whether or not the CPU is servicing interrupts. Because of this, the ICR.PIPN bit field always reflects the pending service request with the highest priority. This can, for example, be used by software polling techniques to determine high-priority requests while leaving the interrupt system disabled.

13.3.2 PCP Interrupt Control Unit (PICU)

The PCP Interrupt Control Unit (PICU) is closely coupled with the PCP and its Interrupt Control Register (PCP_PICR). The operation of the PICU is very similar to the ICU of the CPU with respect to the overall scheme.

Note: Details of the PICU and the PCP_ICR register are described in the chapter about the PCP.

13.4 Arbitration Process

The arbitration process implemented in the TC1797 uses a number of arbitration cycles to determine the pending interrupt request with the highest priority number, SRPN. In each of these cycles, two bits of the SRPNs of all pending service requests are compared against each other. The sequence starts with the high-order bits of the SRPNs and works downwards, such that in the last cycle, bits [1:0] of the SRPNs are compared. Thus, to perform an arbitration through all 8 bits of an SRPN, four arbitration cycles are required. There are two factors determining the duration of the arbitration process:

- Number of arbitration cycles, and
- Duration of arbitration cycles.

Both of these can be controlled by the user.

13.4.1 Controlling the Number of Arbitration Cycles

In a real-time system where responsiveness is critical, arbitration must be as fast as possible. However, to maintain flexibility, the TC1797 system is designed to have a large range of service priorities. If not all priorities are needed in a system, arbitration can be accelerated by not examining all the bits used to identify all 255 unique priorities.

For instance, if a 6-bit number is enough to identify all priority numbers used in a system (meaning that bits [7:6] of all SRPNs are always 0), it is not necessary to perform arbitration on these two bits. Three arbitration cycles will be enough to find the highest number in bits [5:0] of the SRPNs of all pending requests. Similarly, the number of arbitration cycles can be reduced to two if only bits [3:0] are used in all SRPNs, and the number of arbitration cycles can be reduced to one cycle if only bits [1:0] are used.

The ICR.CARBCYC bit field controls the number of cycles in the arbitration process. Its default value is 0, which selects four arbitration cycles. **Table 13-1** gives the options for arbitration cycle control.

Table 13-1 Arbitration Cycle Control

Number of Arbitration Cycles	4	3	2	1
ICR.CARBCYC	00_B	01_B	10_B	11_B
Relevant bits of the SRPNs	[7:0]	[5:0]	[3:0]	[1:0]
Range of priority numbers covered	1..255	1..63	1..15	1..3

Note: If less than four arbitration cycles are selected, the corresponding upper bits of the SRPNs are not examined, even if they do not contain zeros.

13.4.2 Controlling the Duration of Arbitration Cycles

During each arbitration cycle, the rate of information flow between the SRNs and the ICU can become limited by propagation delays within the TC1797 when it is executing at high system clock frequencies. At high frequencies, arbitration cycles may require two system clocks to execute properly. In order to optimize the arbitration scheme at lower system frequencies, an additional control bit, ICR.CONECYC, is implemented. The default value of 0 of this bit selects two clock cycles per arbitration cycle. Setting this bit to 1 selects one clock cycle per arbitration cycle. This bit should only be set to 1 for lower system frequencies. Setting this bit for system frequencies above the specified limit leads to unpredictable behavior of the interrupt system. Correct operation is then not guaranteed.

13.5 Entering an Interrupt Service Routine

When an interrupt request from the ICU is pending and all conditions are met such that the CPU can now service the interrupt request, the CPU performs the following actions in preparation for entering the designated Interrupt Service Routine (ISR):

1. Upper context of the current task is saved[1)]. The current CPU priority number, ICR.CCPN, and the state of the global interrupt enable bit, ICR.IE, are automatically saved with the PCXI register (bit field PCPN and bit PIE).
2. Interrupt system is globally disabled (ICR.IE is set to 0).
3. Current CPU priority number (ICR.CCPN) is set to the value of ICR.PIPN.
4. PSW is set to a default value:
   a) All permissions are enabled, that is, PSW.IO = 10_B.
   b) Memory protection is switched to PRS0, that is, PSW.PRS = 0.
   c) The stack pointer bit is set to the interrupt stack, that is, PSW.IS = 1.
   d) The call depth counter is cleared, the call depth limit is set to 63, that is, PSW.CDC = 0.
5. Stack pointer, A10, is reloaded with the contents of the Interrupt Stack Pointer, ISP, if the PSW.IS bit of the interrupted routine was set to 0 (using the user stack); otherwise it is left unaltered.
6. CPU program counter is assigned with an effective address consisting of the contents of the BIV register OR-ed with the ICR.PIPN number left-shifted by 5. This indexes the Interrupt Vector Table entry corresponding to the interrupt priority.
7. Contents at the effective address of the program counter in the Interrupt Vector Table are fetched as the first instruction of the Interrupt Service Routine (ISR). Execution continues linearly from there until the ISR branches or exits.

1) Note that, if a context-switch trap occurs while the CPU is in the process of saving the upper context of the current task, the pending ISR will not be entered, the interrupt request will be left pending, and the CPU will enter the appropriate trap handling routine instead.

As explained, receipt of further interrupts is disabled (ICR.IE = 0) when an Interrupt Service Routine is entered. At the same time, the current CPU priority ICR.CCPN is set by hardware to the priority of the interrupting source (ICR.PIPN).

Clearly, before the processor can receive any more interrupts, the ISR must eventually re-enable the interrupt system again by setting ICR.IE = 1. Furthermore, the ISR can also modify the priority number ICR.CCPN to allow effective interrupt priority levels. It is up to the user to enable the interrupt system again and optionally modify the priority number CCPN to implement interrupt priority levels or handle special cases.

To simply enable the interrupt system again, the ENABLE instruction can be used, which sets ICR.IE bit to 1. The BISR instruction offers a convenient way to re-enable the interrupt system, to set ICR.CCPN to a new value, and to save the lower context of the interrupted task. It is also possible to use an MTCR instruction to modify ICR.IE and ICR.CCPN. However, this should be performed together with an ISYNC instruction (which synchronizes the instruction stream) to ensure completion of this operation before the execution of following instructions.

Note: The lower context can also be saved through execution of an SVLCX (Save Lower Context) instruction.

13.6 Exiting an Interrupt Service Routine

When an ISR exits with an RFE (Return From Exception) instruction, the hardware automatically restores the upper context. Register PCXI, which holds the Previous CPU Priority Number (PCPN) and the Previous Global Interrupt Enable Bit (PIE), is a part of this upper context. The value saved in PCPN is written to ICR.CCPN to set the CPU priority number to the value before the interruption, and bit PIE is written to ICR.IE to restore the state of this bit. The interrupted routine then continues.

Note: There is no automatic restoring of the lower context on an exit from an Interrupt Service Routine. If the lower context was saved during the execution of the ISR, either through execution of the BISR instruction or an SVLCX instruction, the ISR must restore the lower context again via the RSLCX (Restore Lower Context) instruction before it exits through RFI execution.

13.7 Interrupt Vector Table

Interrupt Service Routines (ISRs) are associated with interrupts at a particular priority by way of the Interrupt Vector Table. The Interrupt Vector Table is an array of ISR entry points.

When the CPU takes an interrupt, it calculates an address in the Interrupt Vector Table that corresponds with the priority of the interrupt (the ICR.PIPN bit field). This address is loaded in the program counter. The CPU begins executing instructions at this address in the Interrupt Vector Table. The code at this address is the start of the selected ISR. Depending on the code size of the ISR, the Interrupt Vector Table may only store the initial portion of the ISR, such as a jump instruction that vectors the CPU to the rest of the ISR elsewhere in memory.

The Interrupt Vector Table is stored in code memory. The BIV register specifies the base address of the Interrupt Vector Table. Interrupt vectors are ordered in the table by increasing priority.

The Base of Interrupt Vector Table register (BIV) stores the base address of the Interrupt Vector Table. It can be assigned to any available code memory. Its default on power-up is fixed at $0000\ 0000_H$. However, the BIV register can be modified using the MTCR instruction during the initialization phase of the system, before interrupts are enabled. With this arrangement, it is possible to have multiple Interrupt Vector Tables and switch between them by changing the contents of the BIV register.

Note: The BIV register is protected by the ENDINIT bit (see chapter describing the watchdog timer). Modifications should only be done while the interrupt system is globally disabled (ICR.IE = 0). Also, an ISYNC instruction should be issued after modifying BIV to ensure completion of this operation before execution of following instructions.

When interrupted, the CPU calculates the entry point of the appropriate ISR from the PIPN and the contents of the BIV register. The PIPN is left-shifted by five bits and OR-ed with the address in the BIV register to generate a pointer into the Interrupt Vector Table. Execution of the ISR begins at this address. Due to this operation, it is recommended that bits [12:5] of register BIV are set to 0 (see **Figure 13-2**). Note that bit 0 of the BIV register is always 0 and cannot be written to (instructions have to be aligned on even byte boundaries).

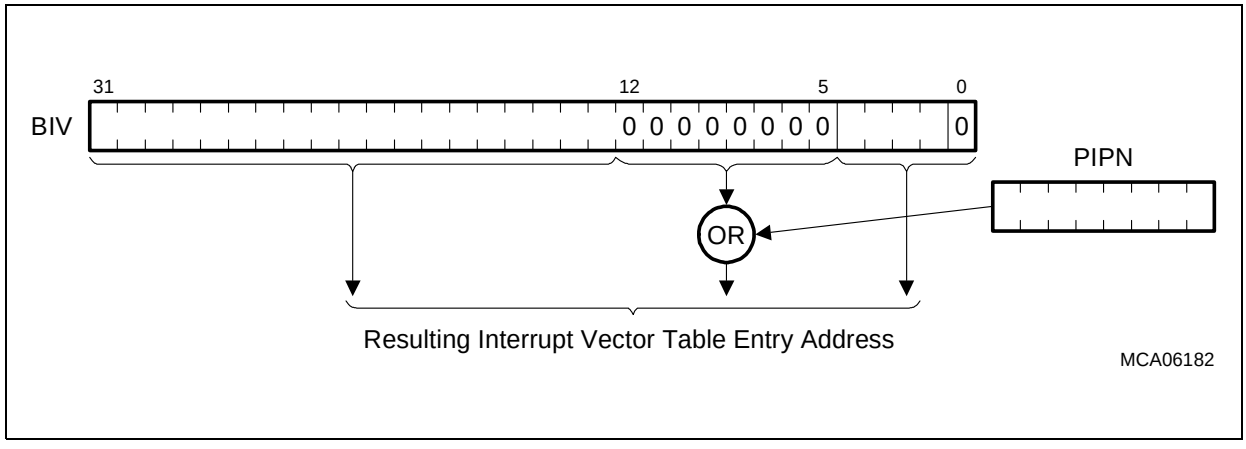


Figure 13-2 Interrupt Vector Table Entry Address Calculation

Left-shifting the PIPN by 5 bits creates entries into the Interrupt Vector Table which are evenly spaced 8 words apart. If an ISR is very short, it may fit entirely within the eight words available in the vector table entry. Otherwise, the code at the entry point must ultimately cause a jump to the rest of the ISR residing elsewhere in memory. Due to the way the vector table is organized according to the interrupt priorities, the TC1797 offers an additional option by allowing spanning several Interrupt Vector Table entries as long as those entries are otherwise unused. **Figure 13-3** illustrates this.

The required size of the Interrupt Vector Table depends only on the range of priority numbers actually used in a system. Of the 256 vector entries, 255 may be used. Vector entry 0 is never used, because if ICR.PIPN is 0, the CPU is not interrupted. Distinct interrupt handlers are supported, but systems requiring fewer entries need not dedicate the full memory area required by the largest configurations.

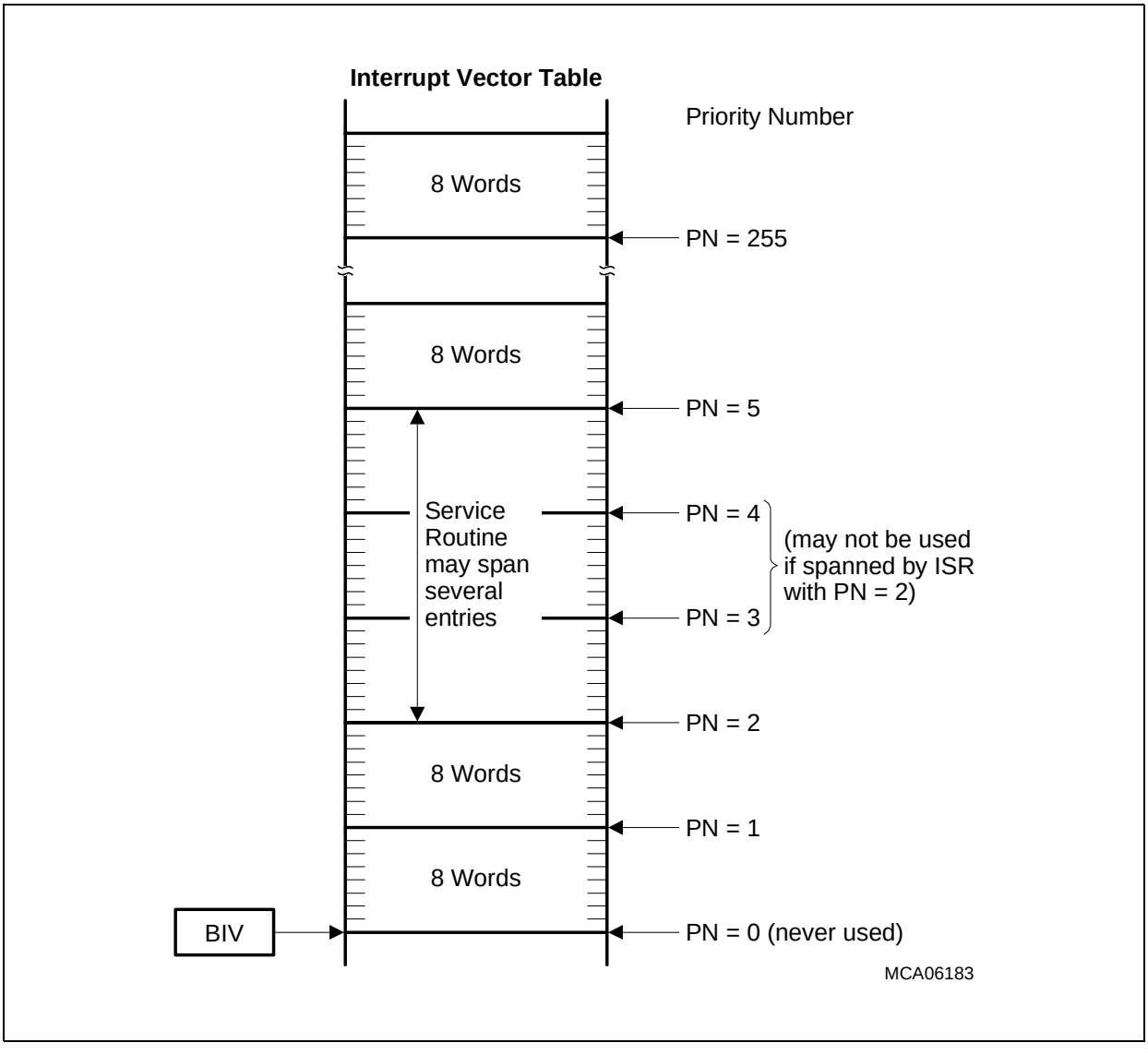


Figure 13-3 Interrupt Vector Table

13.8 Usage of the TC1797 Interrupt System

The following sections provide examples of using the TC1797 interrupt system to solve both typical and special application requirements.

13.8.1 Spanning Interrupt Service Routines Across Vector Entries

Each Interrupt Vector Table entry consists of eight words of memory. If an ISR can be made to fit directly in the Interrupt Vector Table there is no need for a jump instruction to vector to the rest of the interrupt handler elsewhere in memory. However, only the simplest ISRs can fit in the eight words available to a single entry in the table. But it is easy to arrange for ISRs to span across multiple entries, since the Interrupt Vector Table is ordered not by the interrupt source but by interrupt priority. This technique is explained in this section.

In the example of **Figure 13-3**, entry locations 3 and 4 are occupied by the ISR for entry 2. In **Figure 13-3**, the next available entry after entry 2 is entry 5. Of course, if this technique is used, it would be improper to allow any SRN to request service at any of the spanned vector priorities. Thus, priority levels 3 and 4 must not be assigned to SRNs requesting CPU service. They can, however, be used to request PCP service.

There is a performance trade-off that may arise when using this technique because the range of priority numbers used increases. This may have an impact on the number of arbitration cycles required to perform arbitration. Consider the case in which a system uses only three active interrupt sources, that is, where there are only three SRNs enabled to request service. If these three active sources are assigned to priority numbers 1, 2, and 3, it would be sufficient to perform the arbitration in just one cycle. However, if the ISR for interrupt priority 2 is spanned across three Interrupt Vector Table entries as shown in **Figure 13-3**, the priority numbers 1, 2 and 5 would have to be assigned. Thus, two arbitration cycles would have to be used to perform the full arbitration process.

The trade-off between the performance impact of the number of arbitration cycles and the performance gain through spanning service routines can be made by the system designer depending on system needs. Reducing the number of arbitration cycles reduces the service request arbitration latency - spanning service routines reduces the run time of service routines (and therefore also the latency for further interrupts at that priority level or below). For example, if there are multiple fleeting measurements to be made by a system, reducing arbitration latency may be most important. But if keeping total interrupt response time to a minimum is most urgent, spanning Interrupt Vector Table entries may be a solution.

13.8.2 Configuring Ordinary Interrupt Service Routines

When the CPU starts to service an interrupt, the interrupt system is globally disabled and the CPU priority ICR.CCPN is set to the priority of the interrupt now being serviced. This blocks all further interrupts from being serviced until the interrupt system is enabled again.

After an ordinary ISR begins execution, it is usually desirable for the ISR to re-enable global interrupts so that higher-priority interrupts (that is, interrupts that are greater than the current value of ICR.CCPN) can be serviced even during the current ISR's execution. Thus, such an ISR may set ICR.IE = 1 again with, for instance, the ENABLE instruction.

If the ISR enables the interrupt system again by setting ICR.IE = 1 but does not change ICR.CCPN, the effect is that from that point on the hardware can be interrupted by higher-priority interrupts but will be blocked from servicing interrupt requests with the same or lower priority than the current value of bit field ISR.CCPN. Since the current ISR is clearly also at this priority level, the hardware is also blocked from delivering further interrupts to it as well. (This condition is clearly necessary so that the ISR can service the interrupt request automatically.)

When the ISR is finished, it exits with an RFE instruction. Hardware then restores the values of ICR.CCPN and ICR.IE to the values of the interrupted program.

13.8.3 Interrupt Priority Groups

It is sometimes useful to create groups of interrupts at the same or different interrupt priorities that cannot interrupt each other's ISRs. For instance, devices that can generate multiple interrupts may need to have interrupts at different priorities interlocked in this way. The TC1797 interrupt architecture can be used to create such interrupt priority groups. It is effected by managing the current CPU priority level ICR.CCPN in a way described in this section.

For example, in order to make an interrupt priority group out of priority numbers 11 and 12, one would not want an ISR executing at priority 11 to be interrupted by a service request at priority 12, since this would be in the same priority group. Only interrupts above 12 should be allowed to interrupt the ISRs in this interrupt priority group. However, under ordinary ISR usage, the ISR at priority 11 would be interrupted by any request with a higher priority number, including priority 12.

If, however, all ISRs in the interrupt priority group set the value of ICR.CCPN to the highest priority level within their group before they re-enable interrupts, then the desired interlocking will occur.

Figure 13-4 shows an example for interrupt priority grouping. The interrupt requests with the priority numbers 11 and 12 form one group, while the requests with priority numbers 14 through 17 form another group. Each ISR in group 1 sets the value of ICR.CCPN to 12, the highest number in that group, before re-enabling the interrupt system. Each ISR in group 2 sets the value of ICR.CCPN to 17 before re-enabling the interrupt system. If,

for example, interrupt 14 is serviced, it can only be interrupted by requests with a priority number higher than 17; therefore it will not be interrupted by requests from its own priority group or requests with lower priority.

In **Figure 13-4**, the interrupt request with priority number 13 can be said to form an interrupt priority group with just itself as a member.

Setting ICR.CCPN to the maximum value 255 in each service routine has the same effect as not re-enabling the interrupt system; all interrupt requests can then be considered to be in the same group.

Interrupt priority groups demonstrate the power of the TC1797 priority-based interrupt-ordering system. Thus the flexibility of interrupt priority levels ranges from all interrupts in one group to each interrupt request building its own group, and to all possible combinations in between.

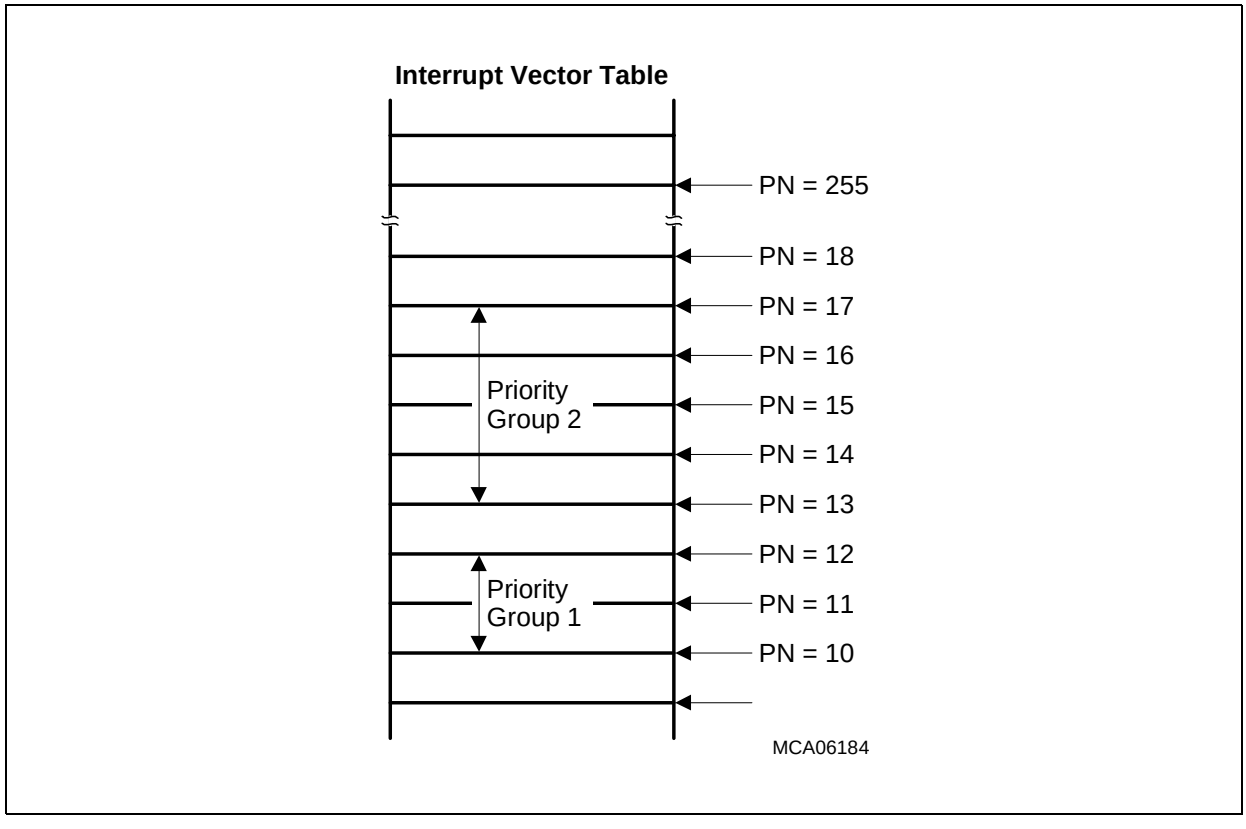


Figure 13-4 Interrupt Priority Groups

13.8.4 Splitting Interrupt Service Across Different Priority Levels

Interrupt service can be divided into multiple ISRs that execute at different priority levels. For example, the beginning stage of interrupt service may be very time-critical, such as reading a data value within a limited time window after the interrupt request activation. However, once the time-critical phase is past, there may still be more to do – for instance,

to process the observation. During this second phase, it may be acceptable for this ISR to be interrupted by lower-level interrupts. This can be performed as follows.

For example, the initial interrupt priority is fixed very high because response time is critical. The necessary actions are carried out immediately by the ISR at that high-priority level. Then the ISR prepares to invoke another ISR at a lower priority level through software to perform the lower-priority actions.

To invoke an ISR through software, the high-priority ISR directly sets an interrupt request bit in an SRN that will invoke the appropriate low-priority ISR. Then the high-priority ISR exits.

When the high-priority ISR exits, the pending low-priority interrupt will eventually be serviced (depending on the priority of other pending interrupts). When the low-priority ISR eventually executes, the low-priority actions of the interrupt will be performed.

The inverse of this method can also be employed, wherein a low-priority ISR raises its own priority level, or leaves interrupts turned off while it executes. For instance, the priority of a service request might be low because the time to respond to the event is not critical, but once it has been granted service, this service should not be interrupted. In this case, the ISR could raise the value of ICR.CCPN to a priority that would exclude some or all other interrupts, or simply leave interrupts disabled.

13.8.5 Using different Priorities for the same Interrupt Source

For some applications, the urgency of a service request may vary, depending on the current state of the system. To handle this, different priority numbers (SRPNs) can be assigned at different times to a service request depending on the application needs.

Of course, Interrupt Service Routines must be placed in the Interrupt Vector Table at all addresses corresponding to the range of priorities used. If service remains the same at different priorities, copies of the ISR can be placed at the possible different entries, or the entries can all vector to a common ISR. If the ISR should execute different code depending on its priority, one need merely put the appropriate ISR in the appropriate entry of the Interrupt Vector Table.

This flexibility is another advantage of the TC1797 interrupt architecture. In traditional interrupt systems where the interrupt vectors are ordered by interrupting source, the ISR would have to check the current priority of the interrupt request and perform a branch to the appropriate code section, causing a delay in the response to the request. In the TC1797, however, the extra check and branch in the ISR are not necessary, hence reduces the interrupt latency.

Because this approach may necessitate an increase in the range of interrupt priorities, the system designer must trade off this advantage against any possible increase in the number of arbitration cycles.

13.8.6 Interrupt Priority 1

Interrupt Priority 1 is the first and lowest-priority entry in the Interrupt Vector Table. It is generally reserved for ISRs which perform task management. ISRs whose actions cause software-managed tasks to be created post a software interrupt request at priority level 1 to signal the event.

The ISR that triggers this event can then execute a normal return from interrupt. There is no need for it to check whether the ISR is returning to the background-task priority level (priority 0) or is returning to a lower-priority ISR that it interrupted. When there is a pending interrupt at a priority higher than the return context for the current interrupt, this interrupt will then be serviced. When a return to the background-task priority level (level 0) is performed, the software-posted interrupt at priority level 1 will be serviced automatically.

13.8.7 Software-Initiated Interrupts

Software can set the service request bit (SRR) in a SRN by writing to its Service Request Control Register. Thus, software can initiate interrupts that are handled by the same mechanism as hardware interrupts.

After the SRR bit is set in an active SRN, there is no way to distinguish between a software-initiated interrupt request and a hardware interrupt request. For this reason, software should only use SRNs and interrupt priority numbers that are not being used for hardware interrupts.

The TC1797 contains four SRNs that support software-initiated interrupts. These SRNs are not connected to peripheral modules and can only cause interrupts when software sets its SRR bit. These SRNs are called the CPU Service Request Nodes (CPU_SRC[3:0]). The PCP can also cause these four SRNs to generate service requests. See also the TriCore chapter for TC1797-specific implementation details of the four CPU Service Request Control Registers.

Additionally, any otherwise unused SRN can be employed to generate software interrupts.

13.8.8 External Interrupts

Four SRNs (SCU_SRC[3:0]) are reserved to handle external interrupts. The setup for external GPIO port input signals (edge/level triggering, gating etc.) that are able to generate an interrupt request is controlled in the External Request Unit (ERU). The ERU functionality is described in detail in the SCU chapter.

13.9 Service Request Node Table

Table 13-2 shows all TC1797 Service Request Nodes.

Table 13-2 Service Request Nodes in the TC1797

Module	No. of Nodes	Description	SRC Register
CPU	4	CPU Service Request Nodes [3:0]	CPU_SRC[3:0]1)
	1	Software Breakpoint Request Node	CPU_SBSRC1)
Cerberus	2	Cerberus/OCDS Request Node[1:0]	CBS_SRC[1:0]
LBCU	1	LBCU Request Node	LBCU_SRC1)
SBCU	1	SBCU Request Node	SBCU_SRC
DMA	8	DMA Service Request Nodes [7:0]	DMA_SRC[7:0]
	4	MLI0 Service Request Nodes [3:0]	DMA_MLI0SRC [3:0]
	2	MLI1 Service Request Nodes [1:0]	DMA_MLI1SRC [1:0]
PCP	12	PCP Service Request Nodes [11:0]	PCP_SRC[11:0]1)
STM	2	STM Service Request Nodes [1:0]	STM_SRC[1:0]
SCU	4	SCU Service Request Nodes [3:0]	SCU_SRC[3:0]
ASC0	4	ASC0 Transmit Interrupt Service Request Node	ASC0_TSRC
		ASC0 Receive Interrupt Service Request Node	ASC0_RSRC
		ASC0 Error Interrupt Service Request Node	ASC0_ESRC
		ASC0 Transmit Buffer Interrupt Service Request Node	ASC0_TBSRC
ASC1	4	ASC1 Transmit Interrupt Service Request Node	ASC1_TSRC
		ASC1 Receive Interrupt Service Request Node	ASC1_RSRC
		ASC1 Error Interrupt Service Request Node	ASC1_ESRC
		ASC1 Transmit Buffer Interrupt Service Request Node	ASC1_TBSRC

Table 13-2 Service Request Nodes in the TC1797 (cont'd)

Module	No. of Nodes	Description	SRC Register
SSC0	3	SSC0 Transmit Interrupt Service Request Node	SSC0_TSRC[1)]
		SSC0 Receive Interrupt Service Request Node	SSC0_RSRC[1)]
		SSC0 Error Interrupt Service Request Node	SSC0_ESRC[1)]
SSC1	3	SSC1 Transmit Interrupt Service Request Node	SSC1_TSRC[1)]
		SSC1 Receive Interrupt Service Request Node	SSC1_RSRC[1)]
		SSC1 Error Interrupt Service Request Node	SSC1_ESRC[1)]
MSC0	2	MSC0 Service Request Nodes [1:0]	MSC0_SRC[1:0]
MSC1	2	MSC1 Service Request Nodes [1:0]	MSC1_SRC[1:0]
CAN	16	CAN Service Request Nodes [15:0]	CAN_SRC[15:0][1)]
GPTA0	38	GPTA0 Service Request Nodes [37:00]	GPTA0_SRC [37:0]
GPTA1	38	GPTA1 Service Request Nodes [37:00]	GPTA1_SRC [37:0]
LTCA2	8	LTCA Service Request Nodes [07:00]	LTCA2_SRC[07:00]
ADC0	9	ADC0 Service Request Nodes [8:0]	ADC0_SRC[8:0][1)]
FADC	4	FADC Service Request Nodes [3:0]	FADC_SRC[3:0][1)]

Table 13-2 Service Request Nodes in the TC1797 (cont'd)

Module	No. of Nodes	Description	SRC Register
E-Ray	8	Interrupt 0 Service Request Control Register	ERAY_INT0SRC[1)]
		Interrupt 1 Service Request Control Register	ERAY_INT0SRC[1)]
		Timer Interrupt 0 Service Request Control Register	ERAY_TINT0SRC[1)]
		Timer Interrupt 1 Service Request Control Register	ERAY_TINT0SRC[1)]
		New Data 0 Service Request Control Register	ERAY_NDAT0SRC[1)]
		New Data 1 Service Request Control Register	ERAY_NDAT0SRC[1)]
		Message Buffer Status Changed 0 Service Request Control Register	ERAY_MBSC0SRC[1)]
		Message Buffer Status Changed 1Service Request Control Register	ERAY_MBSC0SRC[1)]

1) These service request registers are not bit-addressable because its register address is outside the first 16 Kbyte of a segment.

14 System Timer

This chapter describes the System Timer (STM). The TC1797's STM is designed for global system timing applications requiring both high precision and long period.

14.1 Overview

The STM has the following features:

- Free-running 56-bit counter
- All 56 bits can be read synchronously
- Different 32-bit portions of the 56-bit counter can be read synchronously
- Flexible service request generation based on compare match with partial STM content
- Driven by maximum 90 MHz (= f_{SYS}, default after reset = $f_{SYS}/2$)
- Counting starts automatically after a reset operation
- STM registers are reset by an application reset if bit ARSTDIS.STMDIS is cleared. If bit ARSTDIS.STMDIS is set, the STM registers are not reset.[1)]
- STM can be halted in debug/suspend mode (via STM_CLC register)

Special STM register semantics provide synchronous views of the entire 56-bit counter, or 32-bit subsets at different levels of resolution.

The maximum clock period is $2^{56} \times f_{STM}$. At f_{STM} = 90 MHz, for example, the STM counts 25.39 years before overflowing. Thus, it is capable of continuously timing the entire expected product life time of a system without overflowing.

14.2 Operation

The STM is an upward counter, running either at the system clock frequency f_{SYS} or at a fraction of it. The STM clock frequency is $f_{STM} = f_{SYS}/RMC$ with RMC = 0-7 (default after reset is $f_{STM} = f_{SYS}/2$, selected by RMC = 010_B). RMC is a bit field in register STM_CLC. In case of an application reset, the STM is reset if bit SCU_ARSTDIS.DIS0 is set. After reset, the STM is enabled and immediately starts counting up. It is not possible to affect the content of the timer during normal operation of the TC1797. The timer registers can only be read but not written to.

The STM can be optionally disabled for power-saving purposes, or suspended for debugging purposes via its clock control register. In suspend mode of the TC1797 (initiated by writing an appropriate value to STM_CLC register), the STM clock is stopped but all registers are still readable.

Due to the 56-bit width of the STM, it is not possible to read its entire content with one instruction. It needs to be read with two load instructions. Since the timer would continue to count between the two load operations, there is a chance that the two values read are

1) "STM registers" means all registers except STM_CLC, STM_SRC0, and STM_SRC1.

not consistent (due to possible overflow from the low part of the timer to the high part between the two read operations). To enable a synchronous and consistent reading of the STM content, a capture register (STM_CAP) is implemented. It latches the content of the high part of the STM each time when one of the registers STM_TIM0 to STM_TIM5 is read. Thus, STM_CAP holds the upper value of the timer at exactly the same time when the lower part is read. The second read operation would then read the content of the STM_CAP to get the complete timer value.

The STM can also be read in sections from seven registers, STM_TIM0 through STM_TIM6, that select increasingly higher-order 32-bit ranges of the STM. These can be viewed as individual 32-bit timers, each with a different resolution and timing range.

The content of the 56-bit System Timer can be compared against the content of two compare values stored in the STM_CMP0 and STM_CMP1 registers. Service requests can be generated on a compare match of the STM with the STM_CMP0 or STM_CMP1 registers.

Figure 14-1 provides an overview on the STM module. It shows the options for reading parts of STM content.

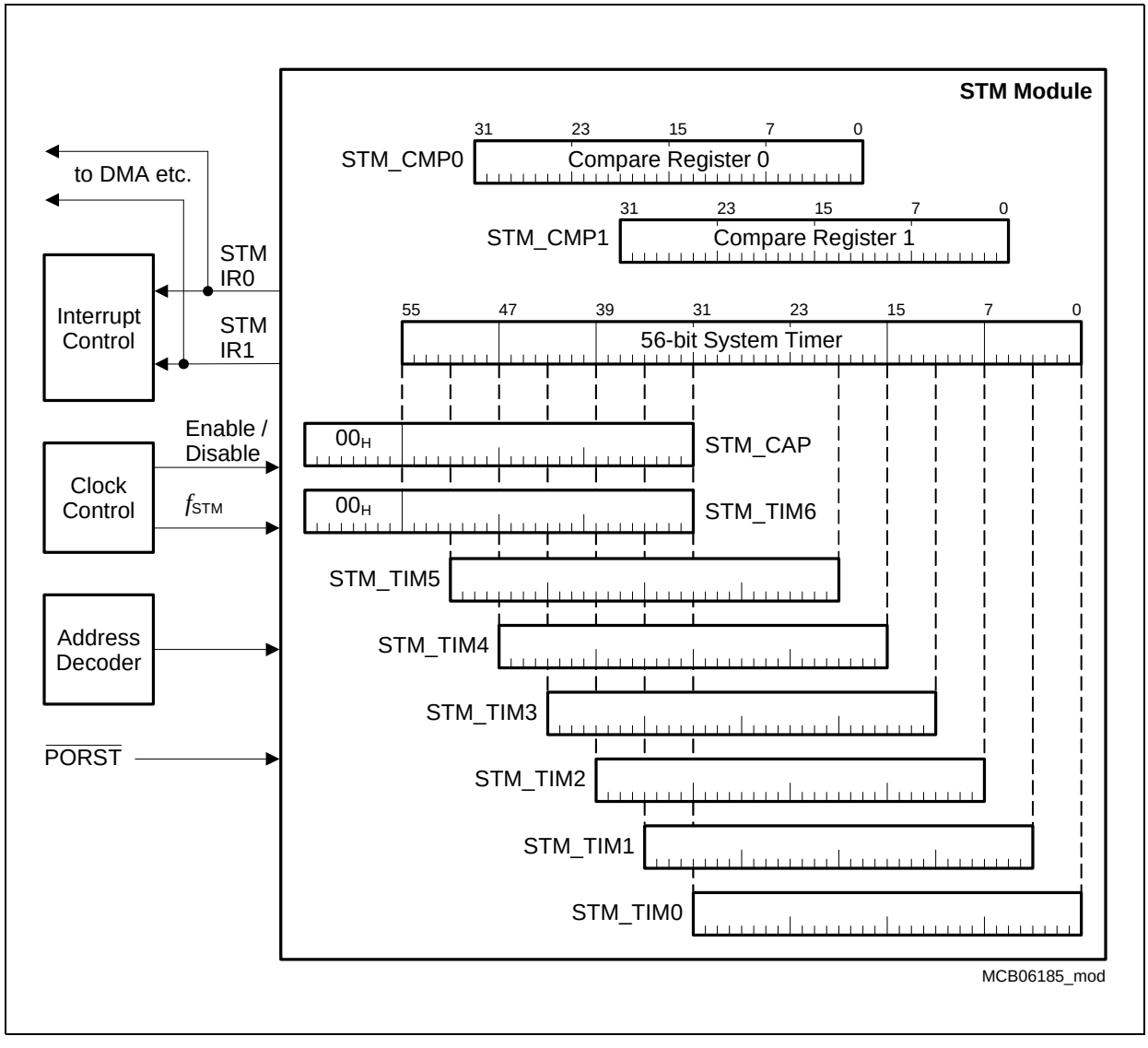


Figure 14-1 General Block Diagram of the STM Module Registers

14.2.1 Resolution and Ranges

Table 14-1 is an overview on the individual timer registers with their resolutions and timing ranges. As an example, the values for a 90 and 45 MHz STM input clock frequency f_{STM} are given.

Table 14-1 System Timer Resolutions and Ranges

Register	STM Bits	Resolution [s]	Range [s]	Resolution	Range	f_{STM} [MHz]
STM_TIM0	[31:0]	1 / f_{STM}	2^{32} / f_{STM}	11.1 ns	47.7 s	90
STM_TIM1	[35:4]	16 / f_{STM}	2^{36} / f_{STM}	178 ns	764 s	
STM_TIM2	[39:8]	256 / f_{STM}	2^{40} / f_{STM}	2.8 µs	203.6 min	
STM_TIM3	[43:12]	4096 / f_{STM}	2^{44} / f_{STM}	45.5 µs	54.3 h	
STM_TIM4	[47:16]	65536 / f_{STM}	2^{48} / f_{STM}	0.728 ms	36.2 days	
STM_TIM5	[51:20]	2^{20} / f_{STM}	2^{52} / f_{STM}	11.7 ms	1.59 yr	
STM_TIM6	[55:32]	2^{32} / f_{STM}	2^{56} / f_{STM}	47.7 s	25.39 yr	
STM_CAP	[55:32]	2^{32} / f_{STM}	2^{56} / f_{STM}	47.7 s	25.39 yr	
STM_TIM0	[31:0]	1 / f_{STM}	2^{32} / f_{STM}	22.2 ns	95.4 s	45
STM_TIM1	[35:4]	16 / f_{STM}	2^{36} / f_{STM}	356 ns	1527.1 s	
STM_TIM2	[39:8]	256 / f_{STM}	2^{40} / f_{STM}	5.7 µs	407.2 min	
STM_TIM3	[43:12]	4096 / f_{STM}	2^{44} / f_{STM}	91.0 µs	108.6 h	
STM_TIM4	[47:16]	65536 / f_{STM}	2^{48} / f_{STM}	1.5 ms	72.4 days	
STM_TIM5	[51:20]	2^{20} / f_{STM}	2^{52} / f_{STM}	23.3 ms	3.17 yr	
STM_TIM6	[55:32]	2^{32} / f_{STM}	2^{56} / f_{STM}	95.4 s	50.78 yr	
STM_CAP	[55:32]	2^{32} / f_{STM}	2^{56} / f_{STM}	95.4 s	50.78 yr	

Note: The maximum input clock f_{STM} is 90 MHz.

14.2.2 Compare Register Operation

The content of the 56-bit STM can be compared against the content of two compare values stored in the STM_CMP0 and STM_CMP1 registers. Service requests can be generated on a compare match of the STM with the STM_CMP0 or STM_CMP1 registers.

Two parameters are programmable for the compare operation:

1. The width of the relevant bits in registers STM_CMP0/STM_CMP1 (compare width MSIZEx) that is taken for the compare operation can be programmed from 1 to 32.
2. The first bit location in the 56-bit STM that is taken for the compare operation can be programmed from 0 to 24.

These programming capabilities make compare functionality very flexible. It even makes it possible to detect bit transitions of a single bit n (n = 0 to 24) within the 56-bit STM by setting MSIZE = 0 and MSTART = n.

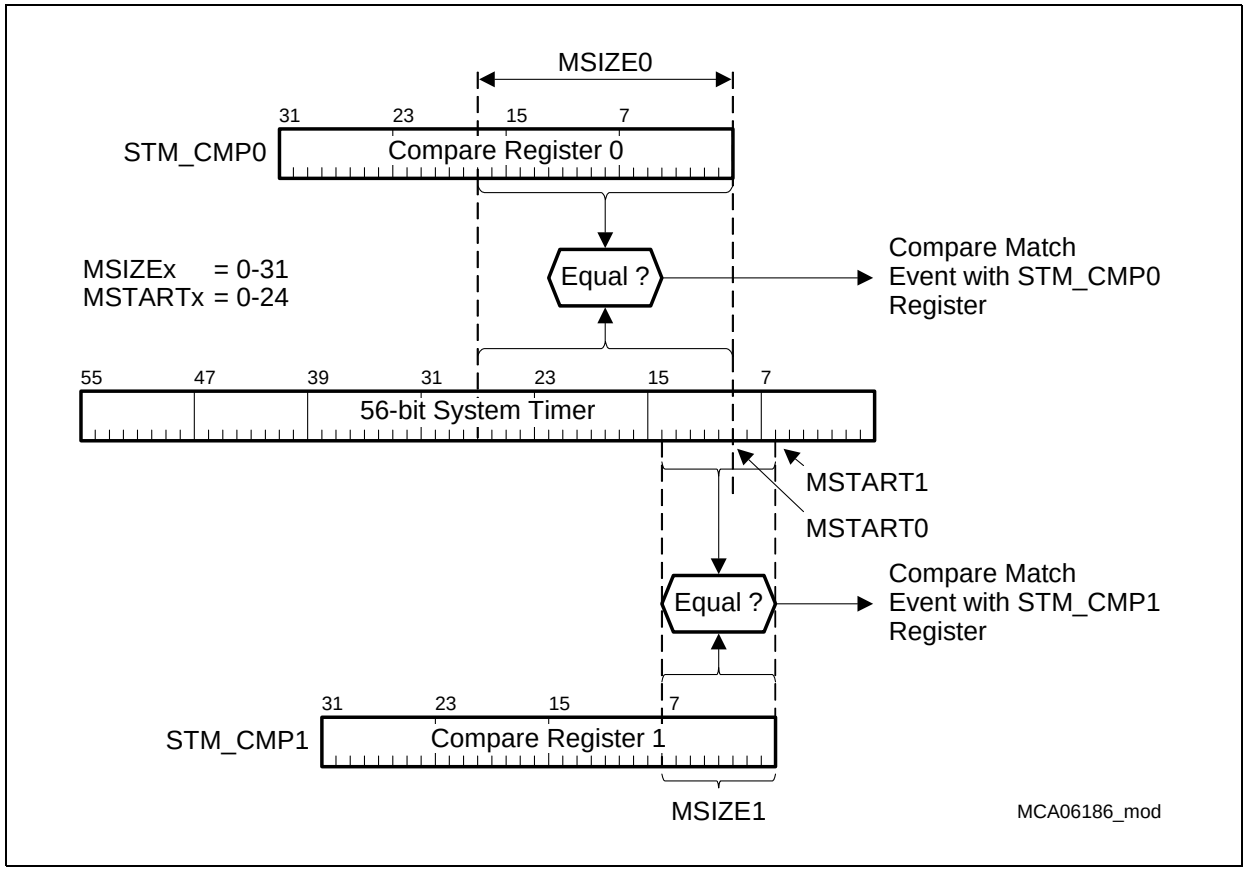


Figure 14-2 Compare Mode Operation

Figure 14-2 shows an example of the compare operation. In this example the following parameters are programmed:

- MSIZE0 = $10001_B = 17_D$; MSTART0 = $01010_B = 9_D$
- MSIZE1 = $00111_B = 7_D$; MSTART1 = $00111_B = 6_D$

A compare operation with MSIZE not equal 0 always implies that the compared value as stored in the CMP register is right-extended with zeros. This means that in the example of **Figure 14-2**, the compare register content STM_CMP0[17:0] plus nine zero bits right-extended is compared with STM[27:0] with STM[8:0] = 000_H. In case of register STM_CMP1, STM[14:0] with STM[5:0] = 00_H are compared with STM_CMP1[8:0] plus six zero bits right-extended.

14.2.3 Compare Match Interrupt Control

The compare match interrupt control logic is shown in **Figure 14-3**. Each STM_CMPx register has its compare match interrupt request flag (STM_ICR.CMPxIR) that is set by hardware on a compare match event. The interrupt request flags can be set (STM_ISSR.CMPxIRS) or cleared (STM_ISSR.CMPxIRR) by software. Note that setting STM_ICR.CMPxIR by writing a 1 into STM_ISSR.CMPxIRS does not generate an interrupt at STMIRx. The compare match interrupts from CMP0 and CMP1 can be further directed by STM_ICR.CMPxOS to either output signal STMIR0 or STMIR1. The STMIR0 and STMIR1 outputs are each connected to interrupt service request control registers, STM_SRC0 and STM_SCR1, respectively.

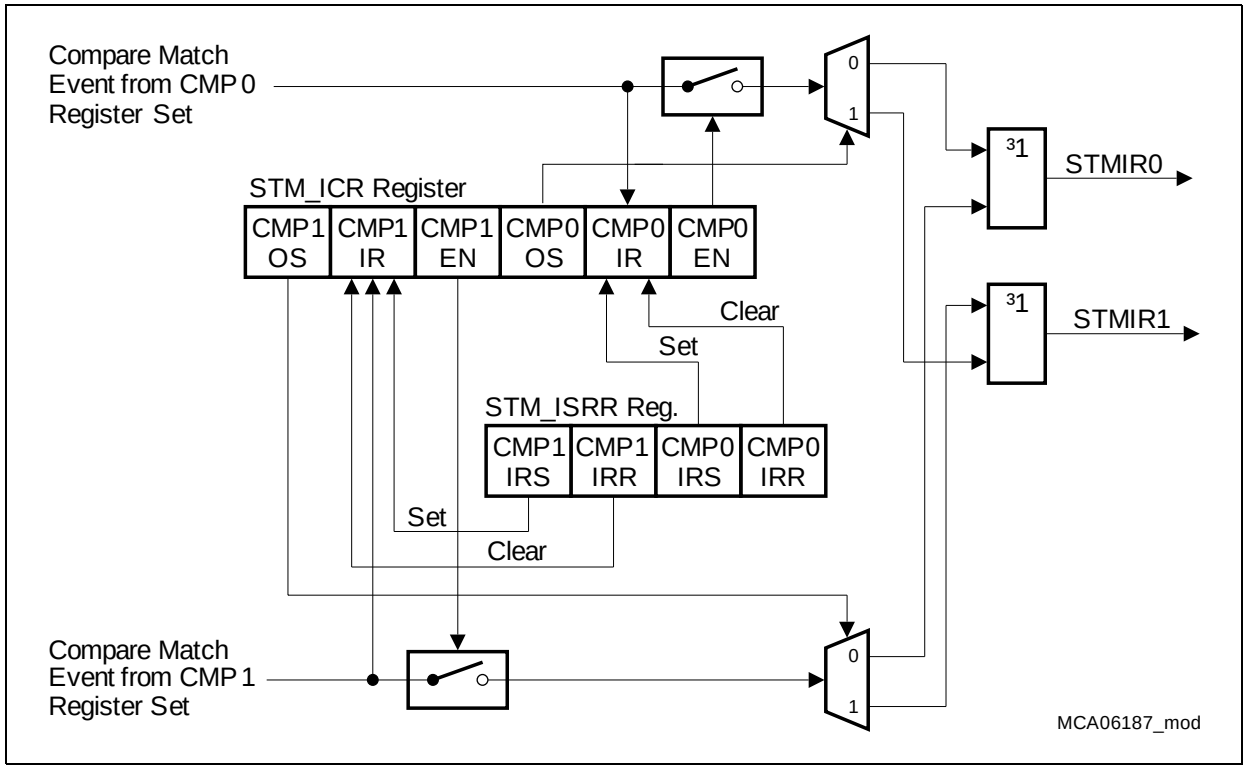


Figure 14-3 STM Interrupt Control

The compare match interrupt flags STM_ICR.CMPxIR are immediately set after an STM reset operation, caused by a compare match event with the reset values of the STM and the compare registers STM_CMPx. This setting of the CMPxIR flags does not directly

generate compare match interrupts because the compare match interrupts are automatically disabled after a STM reset operation (CMPxEN = 0). Therefore, before enabling a compare match interrupt after a STM reset operation, the CMPxIR flags should be cleared by software (writing register STM_ISSR with CMPxIRR set). Otherwise, undesired compare match interrupt events are triggered. Details about DMA connections of STMIR0 and STMIR1 are given in **Table 14-4** on **Page 14-21**.

14.3 STM Registers

This section describes the STM registers of the STM. The STM registers can be divided into four types, as shown in **Figure 14-4**.

STM Registers Overview

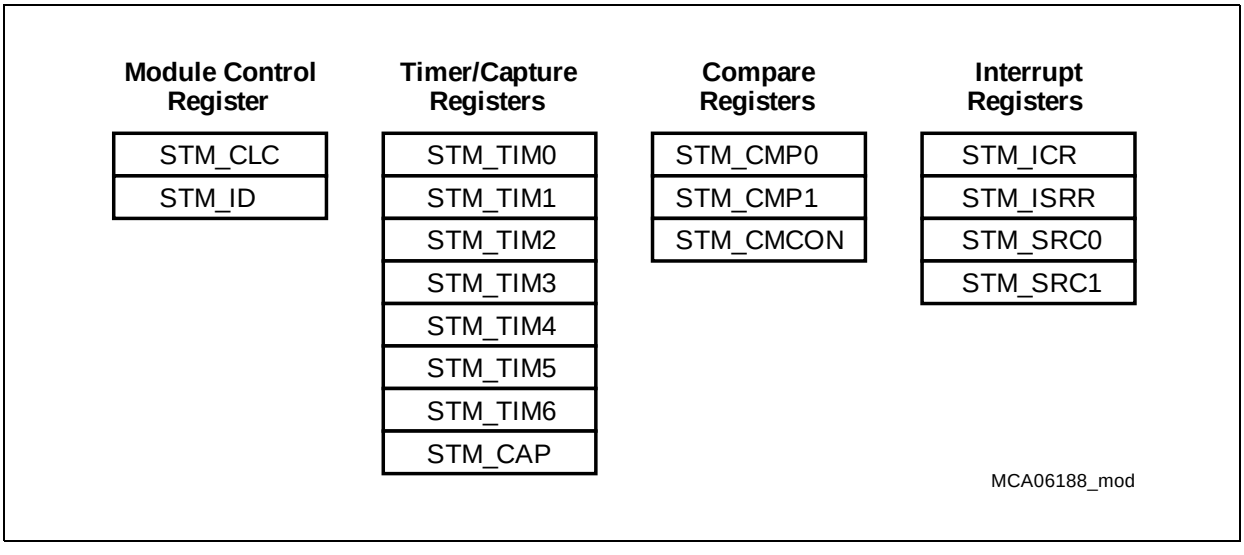


Figure 14-4 STM Registers

In TC1797 all registers are readable is suspend mode. The complete and detailed address map of the STM module with its registers is shown in **Table 14-5** on **Page 14-22**.

Table 14-2 Registers Address Space

Module	Base Address	End Address	Note
STM	$F000\ 0200_H$	$F000\ 02FF_H$	-

Table 14-3 Registers Overview - STM Registers

Register Short Name	Register Long Name	Offset Address	Description see
STM_CLC[1)]	STM Clock Control Register	00_H	Page 14-9
STM_ID	STM Module Identification Register	08_H	Page 14-10
STM_TIM0	STM Timer Register 0	10_H	Page 14-11
STM_TIM1	STM Timer Register 1	14_H	Page 14-11
STM_TIM2	STM Timer Register 2	18_H	Page 14-12
STM_TIM3	STM Timer Register 3	$1C_H$	Page 14-12
STM_TIM4	STM Timer Register 4	20_H	Page 14-12
STM_TIM5	STM Timer Register 5	24_H	Page 14-13
STM_TIM6	STM Timer Register 6	28_H	Page 14-13
STM_CAP	STM Timer Capture Register	$2C_H$	Page 14-14
STM_CMP0	STM Compare Register 0	30_H	Page 14-14
STM_CMP1	STM Compare Register 1	34_H	Page 14-14
STM_CMCON	STM Compare Match Control Register	38_H	Page 14-15
STM_ICR	STM Interrupt Control Register	$3C_H$	Page 14-17
STM_ISRR	STM Interrupt Set/Reset Register	40_H	Page 14-19
STM_SRC1[1)]	STM Service Request Control Register 1	$F8_H$	Page 14-20
STM_SRC0[1)]	STM Service Request Control Register 0	FC_H	Page 14-20

1) These registers are reset by an application reset if bit ARSTDIS.STMDIS is set.

14.3.1 Clock Control Register

The STM clock control register is used to switch the STM on or off and to control its input clock rate. After a power-on reset, the STM is always enabled and starts counting. The STM can be disabled by setting bit DISR to 1.

STM_CLC
STM Clock Control Register (00_H) Reset Value: $0000\ 0200_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

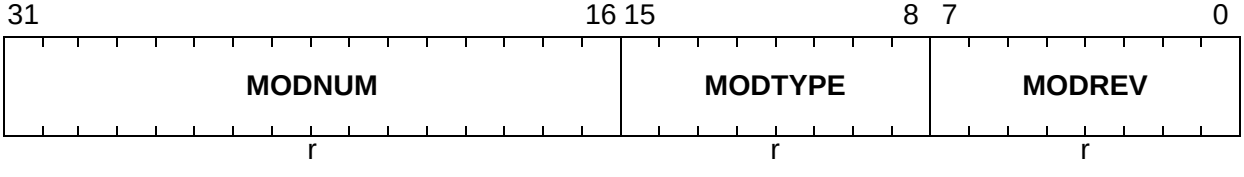
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					RMC			0		FSOE	SBWE	EDIS	SPEN	DISS	DISR
r					rw			r		rw	w	rw	rw	r	rw

Field	Bits	Type	Description
DISR	0	rw	**Module Disable Request Bit** Used for enable/disable control of the STM module. 0_B No disable requested 1_B Disable requested
DISS	1	r	**Module Disable Status Bit** Bit indicates the current status of the STM module. 0_B STM module is enabled 1_B STM module is disabled
SPEN	2	rw	**Module Suspend Enable for OCDS** Used for enabling the suspend mode.
EDIS	3	rw	**Sleep Mode Enable Control** Used for module sleep mode control.
SBWE	4	w	**Module Suspend Bit Write Enable for OCDS** Determines whether SPEN and FSOE are write-protected.
FSOE	5	rw	**Fast Switch Off Enable** Used for fast clock switch off in OCDS suspend mode.

Field	Bits	Type	Description
RMC	[10:8]	rw	**Clock Divider in Run Mode** 000_B No clock signal f_{STM} generated 001_B Clock signal $f_{STM} = f_{SYS}$ selected 010_B Clock signal f_{STM} / 2 selected (default after reset) 011_B Clock signal f_{STM} / 3 selected $...{}_B$ ... 111_B Clock signal f_{STM} / 7 selected
0	[7:6], [31:11]	r	**Reserved** Read as 0; should be written with 0.

The STM Module Identification Register ID contains read-only information about the module version.

STM_ID
STM Module Identification Register (08_H) Reset Value: 0000 C0XX$_H$

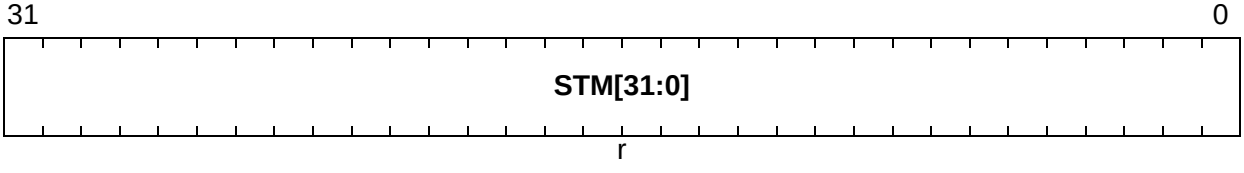


Field	Bits	Type	Description
MODREV	[7:0]	r	**Module Revision Number** MODREV defines the module revision number. The value of a module revision starts with 01_H (first revision).
MODTYPE	[15:8]	r	**Module Type** This bit field defines the module as a 32-bit module: $C0_H$
MODNUM	[31:16]	r	**Module Number Value** This bit field defines the module identification number for the STM: 0000_H

14.3.2 Timer/Capture Registers

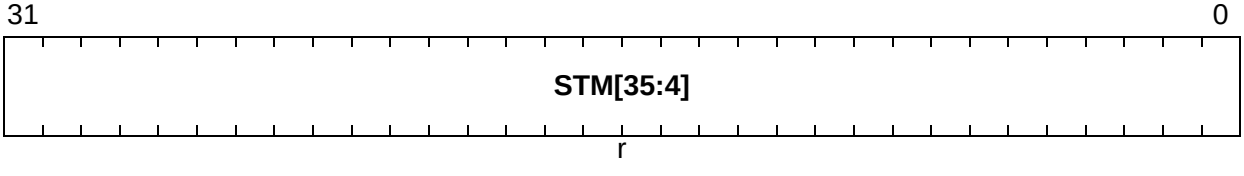
Registers STM_TIM1 to STM_TIM6 provide 32-bit views at varying resolutions of the underlying STM counter.

STM_TIM0
STM Timer Register 0 **(10_H)** **Reset Value: 0000 0000_H**



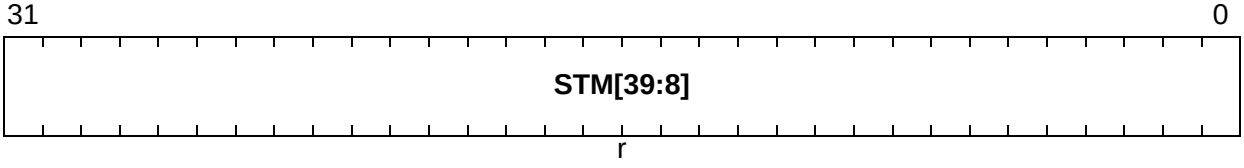
Field	Bits	Type	Description
STM[31:0]	[31:0]	r	**System Timer Bits [31:0]** This bit field contains bits [31:0] of the 56-bit STM.

STM_TIM1
STM Timer Register 1 **(14_H)** **Reset Value: 0000 0000_H**



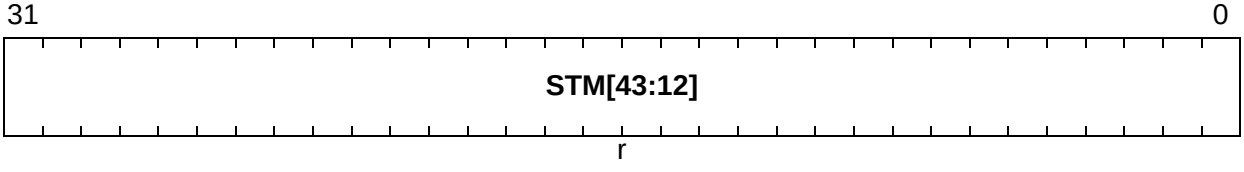
Field	Bits	Type	Description
STM[35:4]	[31:0]	r	**System Timer Bits [35:4]** This bit field contains bits [35:4] of the 56-bit STM.

STM_TIM2
STM Timer Register 2 **(18_H)** **Reset Value: 0000 0000$_H$**



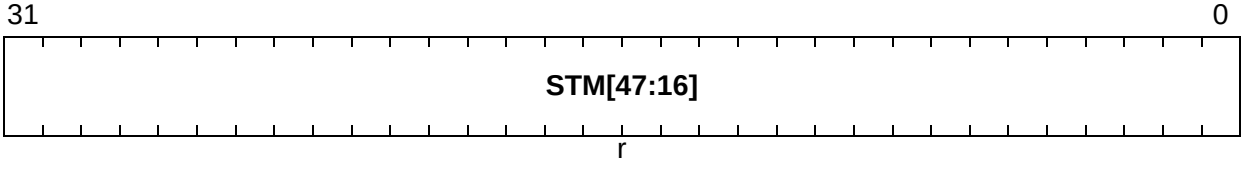
Field	Bits	Type	Description
STM[39:8]	[31:0]	r	**System Timer Bits [39:8]** This bit field contains bits [39:8] of the 56-bit STM.

STM_TIM3
STM Timer Register 3 **($1C_H$)** **Reset Value: 0000 0000$_H$**



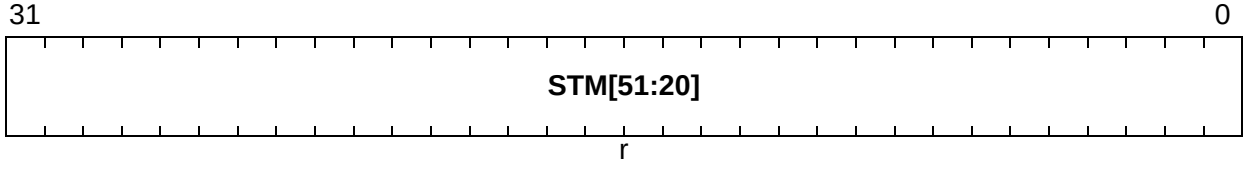
Field	Bits	Type	Description
STM[43:12]	[31:0]	r	**System Timer Bits [43:12]** This bit field contains bits [43:12] of the 56-bit STM.

STM_TIM4
STM Timer Register 4 **(20_H)** **Reset Value: 0000 0000$_H$**



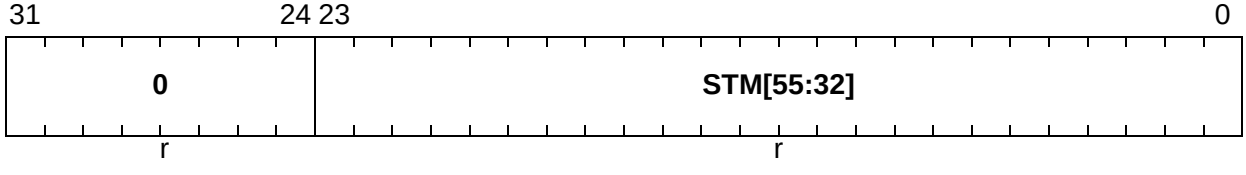
Field	Bits	Type	Description
STM[47:16]	[31:0]	r	**System Timer Bits [47:16]** This bit field contains bits [47:16] of the 56-bit STM.

STM_TIM5
STM Timer Register 5 **(24_H)** **Reset Value: 0000 0000$_H$**



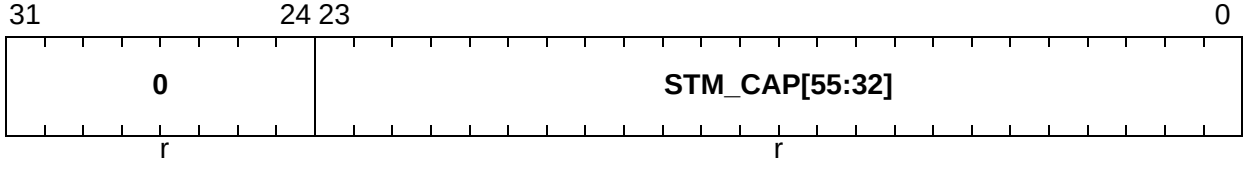
Field	Bits	Type	Description
STM[51:20]	[31:0]	r	**System Timer Bits [51:20]** This bit field contains bits [51:20] of the 56-bit STM.

STM_TIM6
STM Timer Register 6 **(28_H)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
STM[55:32]	[23:0]	r	**System Timer Bits [55:32]** This bit field contains bits [55:32] of the 56-bit STM.
0	[31:24]	r	**Reserved** Read as 0.

STM_CAP
STM Timer Capture Register **($2C_H$)** **Reset Value: 0000 0000$_H$**



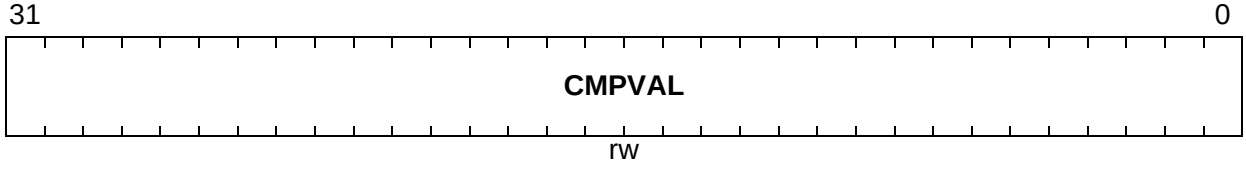
Field	Bits	Type	Description
STM[55:32]	[23:0]	r	**Captured System Timer Bits [55:32]** The capture register STM_CAP always captures the STM bits [55:32] when one of the registers STM_TIM0 to STM_TIM5 is read. This capture operation is performed in order to enable software to operate with a coherent value of all the 56 STM bits at one time stamp.This bit field contains bits [55:32] of the 56-bit STM.
0	[31:24]	r	**Reserved** Read as 0.

Note: The bits in registers STM_CAP to STM_TIM0 are all read-only bits.

14.3.3 Compare Registers

The compare register CMPx holds up to 32-bits; its value is compared to the value of the STM.

STM_CMPx (x = 0-1)
STM Compare Register x **(30_H+x*4_H)** **Reset Value: 0000 0000$_H$**

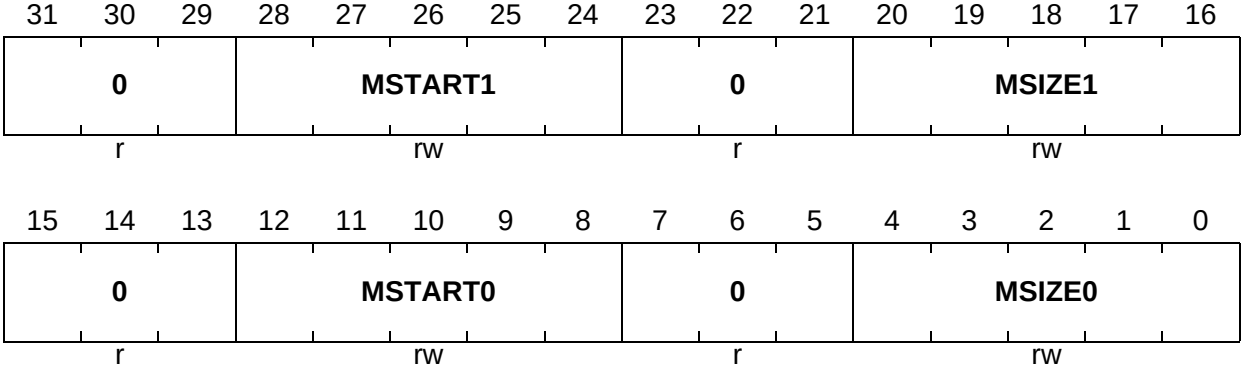


Field	Bits	Type	Description
CMPVAL	[31:0]	rw	**Compare Value of Compare Register x** This bit field holds up to 32 bits of the compare value (right-adjusted).

The STM Compare Match Control Register controls the parameters of the compare logic.

STM_CMCON
STM Compare Match Control Register

(38_H) **Reset Value: $0000\ 0000_H$**



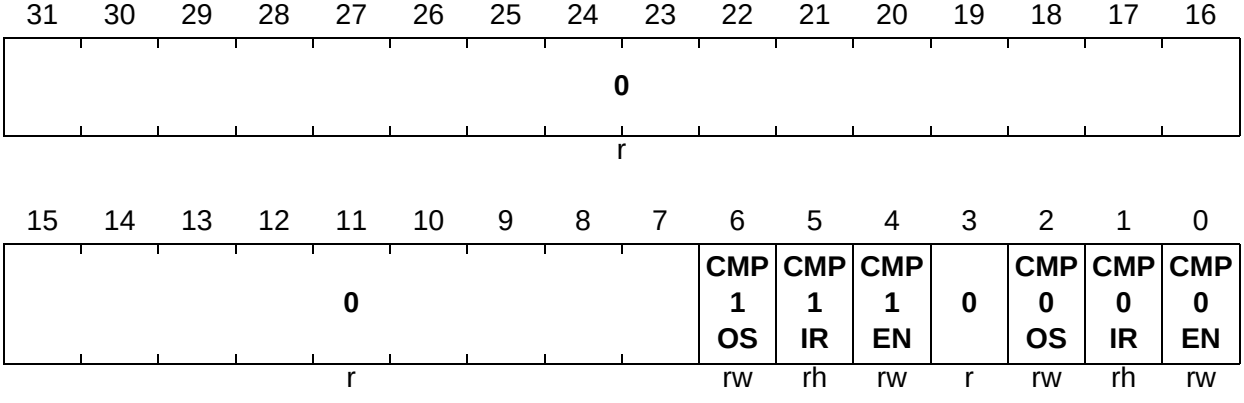
Field	Bits	Type	Description
MSIZE0	[4:0]	rw	**Compare Register Size for CMP0** This bit field determines the number of bits in register CMP0 (starting from bit 0) that are used for the compare operation with the System Timer. 00000_B CMP0[0] used for compare operation 00001_B CMP0[1:0] used for compare operation $\ldots_B$ … 11110_B CMP0[30:0] used for compare operation 11111_B CMP0[31:0] used for compare operation
MSTART0	[12:8]	rw	**Start Bit Location for CMP0** This bit field determines the lowest bit number of the 56-bit STM that is compared with the content of register CMP0 bit 0. The number of bits to be compared is defined by bit field MSIZE0. 00000_B STM[0] is the lowest bit number 00001_B STM[1] is the lowest bit number $\ldots_B$ … 10111_B STM[23] is the lowest bit number 11000_B STM[24] is the lowest bit number Bit combinations 11001_B to 11111_B are reserved and must not be used.

Field	Bits	Type	Description
MSIZE1	[20:16]	rw	**Compare Register Size for CMP1** This bit field determines the number of bits in register CMP1 (starting from bit 0) that are used for the compare operation with the System Timer. 00000_B CMP1[0] used for compare operation 00001_B CMP1[1:0] used for compare operation $\ldots_B$ … 11110_B CMP1[30:0] used for compare operation 11111_B CMP1[31:0] used for compare operation
MSTART1	[28:24]	rw	**Start Bit Location for CMP1** This bit field determines the lowest bit number of the 56-bit STM that is compared with the content of register CMP1 bit 0. The number of bits to be compared is defined by bit field MSIZE1. 00000_B STM[0] is the lowest bit number 00001_B STM[1] is the lowest bit number $\ldots_B$ … 10111_B STM[23] is the lowest bit number 11000_B STM[24] is the lowest bit number Bit combinations 11001_B to 11111_B are reserved and must not be used.
0	[7:5], [15:13], [23:21], [31:29]	r	**Reserved** Read as 0; should be written with 0.

14.3.4 Interrupt Registers

The two compare match interrupts of the STM are controlled by the STM Interrupt Control Register.

STM_ICR
STM Interrupt Control Register ($3C_H$) Reset Value: 0000 0000_H

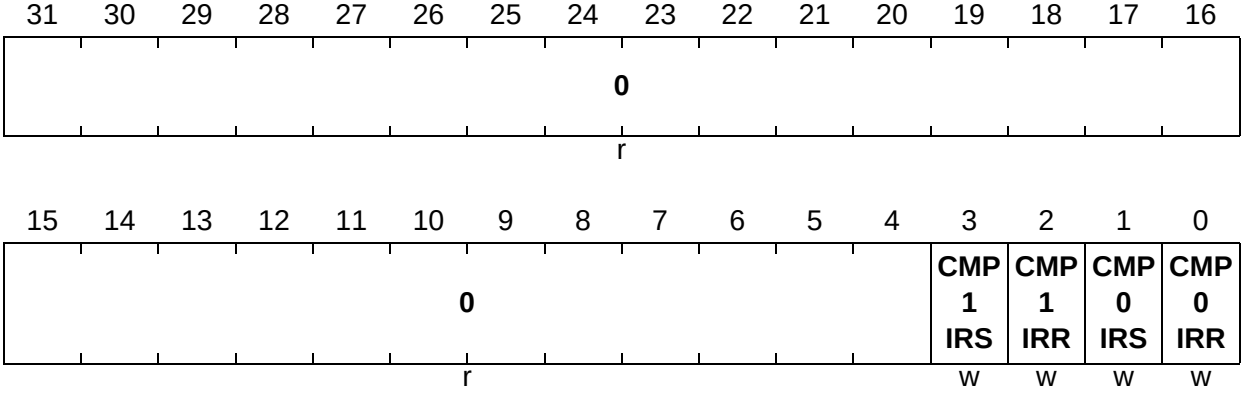


Field	Bits	Type	Description
CMP0EN	0	rw	**Compare Register CMP0 Interrupt Enable Control** This bit enables the compare match interrupt with compare register CMP0. 0_B Interrupt on compare match with CMP0 disabled 1_B Interrupt on compare match with CMP0 enabled
CMP0IR	1	rh	**Compare Register CMP0 Interrupt Request Flag** This bit indicates whether or not a compare match interrupt request of compare register CMP0 is pending. CMP0IR must be cleared by software. 0_B A compare match interrupt has not been detected since the bit has been cleared for the last time. 1_B A compare match interrupt has been detected. CMPIR0 must be cleared by software and can be set by software, too (see CMPISRR register). After a STM reset operation, CMP0IR is immediately set as a result of a compare match event with the reset values of the STM and the compare registers CMP0.

Field	Bits	Type	Description
CMP0OS	2	rw	**Compare Register CMP0 Interrupt Output Selection** This bit determines the interrupt output that is activated on a compare match event of compare register CMP0. 0_B Interrupt output STMIR0 selected 1_B Interrupt output STMIR1 selected
CMP1EN	4	rw	**Compare Register CMP1 Interrupt Enable Control** This bit enables the compare match interrupt with compare register CMP1. 0_B Interrupt on compare match with CMP1 disabled 1_B Interrupt on compare match with CMP1 enabled
CMP1IR	5	rh	**Compare Register CMP1 Interrupt Request Flag** This bit indicates whether or not a compare match interrupt request of compare register CMP1 is pending. CMP1IR must be cleared by software. 0_B A compare match interrupt has not been detected since the bit has been cleared for the last time. 1_B A compare match interrupt has been detected. CMPIR1 must be cleared by software and can be set by software, too (see CMPISRR register). After a STM reset, CMP1IR is immediately set as a result of a compare match event with the reset values of the STM and the compare register CMP1.
CMP1OS	6	rw	**Compare Register CMP1 Interrupt Output Selection** This bit determines the interrupt output that is activated on a compare match event of compare register CMP1. 0_B Interrupt output STMIR0 selected 1_B Interrupt output STMIR1 selected
0	3, [31:7]	r	**Reserved** Read as 0; should be written with 0.

The bits in the STM Interrupt Set/Reset Register make it possible to set or cleared the compare match interrupt request status flags of register ICR.

STM_ISRR
STM Interrupt Set/Reset Register (40_H) Reset Value: 0000 0000_H



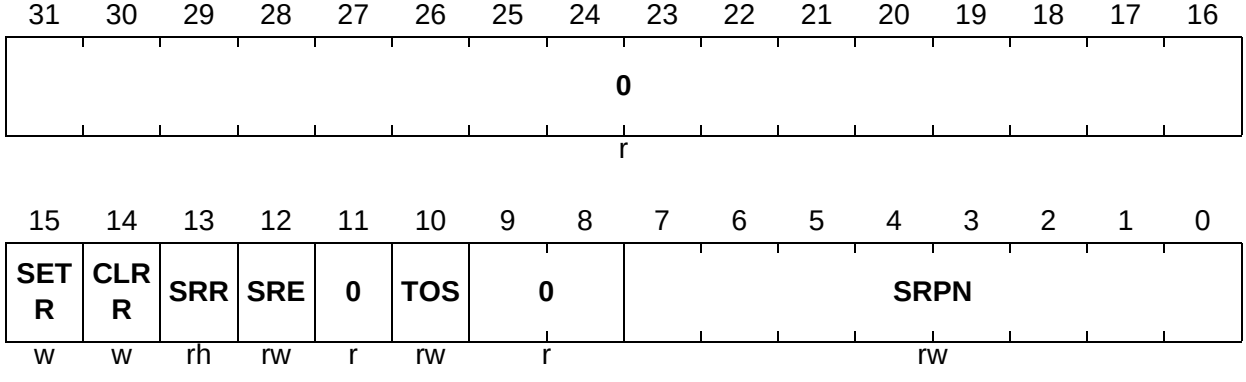
Field	Bits	Type	Description
CMP0IRR	0	w	**Reset Compare Register CMP0 Interrupt Flag** 0_B Bit ICR.CMP0IR is not changed. 1_B Bit ICR.CMP0IR is cleared.
CMP0IRS	1	w	**Set Compare Register CMP0 Interrupt Flag** 0_B Bit ICR.CMP0IR is not changed. 1_B Bit ICR.CMP0IR is set. The state of bit CMP0IRR is “don’t care” in this case.
CMP1IRR	2	w	**Reset Compare Register CMP1 Interrupt Flag** 0_B Bit ICR.CMP1IR is not changed. 1_B Bit ICR.CMP1IR is cleared.
CMP1IRS	3	w	**Set Compare Register CMP1 Interrupt Flag** 0_B Bit ICR.CMP1IR is not changed. 1_B Bit ICR.CMP1IR is set. The state of bit CMP1IRR is “don’t care” in this case.
0	[31:4]	r	**Reserved** Read as 0; should be written with 0.

Note: Reading register CMISRR always returns 0000 0000_H.

In the TC1797, the compare match interrupt output signals of the STM, STMIR0 and STMIR1 are controlled by the STM Service Request Control Registers STM_SRC0 and STM_SRC1.

STM_SRC0
STM Service Request Control Register 0
(FC_H) **Reset Value: 0000 0000$_H$**

STM_SRC1
STM Service Request Control Register 1
($F8_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
SRPN	[7:0]	rw	**Service Request Priority Number**
TOS	10	rw	**Type of Service Control**
SRE	12	rw	**Service Request Enable**
SRR	13	rh	**Service Request Flag**
CLRR	14	w	**Request Clear Bit**
SETR	15	w	**Request Set Bit**
0	[9:8], 11, [31:16]	r	**Reserved** Read as 0; should be written with 0.

14.4 STM Module Implementation

This section defines implementation specific details of the STM in the TC1797.

14.4.1 On-chip Service Request Connections

The two compare match service request outputs STMIR0 and STMIR1 are connected in the TC1797 to on-chip devices as described in **Table 14-4**.

Table 14-4 System Timer On-Chip Interconnections

Service Request Signal	Connected to
STMIR0	DMA Channel 00 Request Input 8 DMA Channel 01 Request Input 8 DMA Channel 02 Request Input 8 DMA Channel 03 Request Input 8 DMA Channel 04 Request Input 8 DMA Channel 05 Request Input 8 DMA Channel 06 Request Input 8 DMA Channel 07 Request Input 8 DMA Channel 10 Request Input 8 DMA Channel 11 Request Input 8 DMA Channel 12 Request Input 8 DMA Channel 13 Request Input 8 DMA Channel 14 Request Input 8 DMA Channel 15 Request Input 8 DMA Channel 16 Request Input 8 DMA Channel 17 Request Input 8
STMIR1	ADC0_REQGT[4:0]_5 ADC1_REQGT[4:0]_5 ADC2_REQGT[4:0]_5

14.4.2 STM Address Map

Table 14-5 defines the complete address range of the STM with absolute addresses and the read/write access rights.

Table 14-5 Address Map of STM

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
System Timer (STM)					
STM_CLC	STM Clock Control Register	F000 0200$_H$	U, SV	SV, E	0000 0200$_H$
–	Reserved	F000 0204$_H$	BE	BE	–
STM_ID	STM Module Identification Register	F000 0208$_H$	U, SV	BE	0000 C0XX$_H$
–	Reserved	F000 020C$_H$	BE	BE	–
STM_TIM0	STM Timer Register 0	F000 0210$_H$	U, SV	U, SV	0000 0000$_H$
STM_TIM1	STM Timer Register 1	F000 0214$_H$	U, SV	U, SV	0000 0000$_H$
STM_TIM2	STM Timer Register 2	F000 0218$_H$	U, SV	U, SV	0000 0000$_H$
STM_TIM3	STM Timer Register 3	F000 021C$_H$	U, SV	U, SV	0000 0000$_H$
STM_TIM4	STM Timer Register 4	F000 0220$_H$	U, SV	U, SV	0000 0000$_H$
STM_TIM5	STM Timer Register 5	F000 0224$_H$	U, SV	U, SV	0000 0000$_H$
STM_TIM6	STM Timer Register 6	F000 0228$_H$	U, SV	U, SV	0000 0000$_H$
STM_CAP	STM Timer Capture Reg.	F000 022C$_H$	U, SV	U, SV	0000 0000$_H$
STM_CMP0	STM Compare Register 0	F000 0230$_H$	U, SV	U, SV	0000 0000$_H$
STM_CMP1	STM Compare Register 1	F000 0234$_H$	U, SV	U, SV	0000 0000$_H$
STM_ CMCON	STM Compare Match Control Register	F000 0238$_H$	U, SV	U, SV	0000 0000$_H$
STM_ICR	STM Interrupt Control Register	F000 023C$_H$	U, SV	U, SV	0000 0000$_H$
STM_ISRR	STM Interrupt Set/Reset Register	F000 0240$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 0244$_H$ - F000 02F4$_H$	BE	BE	–
STM_ SRC1	STM Service Request Control Register 1	F000 02F8$_H$	U, SV	U, SV	0000 0000$_H$
STM_ SRC0	STM Service Request Control Register 0	F000 02FC$_H$	U, SV	U, SV	0000 0000$_H$

15 On-Chip Debug Support

This chapter gives an overview about the debug features of the TC1797 device. This chapter does not describe the TC1797 debug functionality and capabilities in detail. For detailed information about the On-Chip Debug Support (OCDS) functionality as required by tool suppliers please contact local Infineon representatives.

15.1 Overview

TC1797 supports OCDS Level 1 and 3.

OCDS Level 1

The OCDS Level 1 is mainly assigned for system software debugging purposes which have a demand for low-cost standard debugger hardware.

The OCDS Level 1 is based on a Debug Interface that is used by the external debug hardware to communicate with the system. The on-chip Cerberus module controls the interactions between the Debug Interface and the on-chip modules and allows in particular to access the whole address space of the device. The memory mapped on-chip debug resources make it possible to trigger on instruction and data addresses as well as to control user program execution (run/stop, breakpoint, single-step).

OCDS Level 3

The OCDS Level 3 is based on a Multi Core Debug Solution (MCDS) using a special Emulation Device. This device has the following features required for high-end emulation purposes:

- TriCore program trace
- TriCore load/store data trace (no register file trace)
- PCP ownership trace
- PCP program trace
- PCP data write to PRAM trace (no register file trace)
- Full visibility of internal peripheral bus (SPB)
- Full visibility of Local Memory Bus (LMB)
- Time aligned trace of all sources
- Breakpoints and watchpoints based on common event generation logic
- Magnitude comparators working on instruction pointers and memory addresses: A <= IP <= B
- Masked magnitude comparators working on the data busses: DATA = “xxxx55xx”
- Sequential event logic: Counters driven by events and equipped with limit comparators are used as event sources again for breakpoint or trace qualification
- Optimized compression of buffered trace data
- Code and data fetch from Emulation Memory
- Highly sophisticated complex qualification- and trigger mechanism

- Pre- and post event trace buffering (“digital oscilloscope”)
- Performance counters
- Continuous trace logging and trace data acquisition up to the bandwidth of the used host interface
- Central time stamp unit to correlate traces from different cores or other sources
- Central mechanism to start and stop all cores simultaneously or selectively
- Halt the system when trace memory is full

The Emulation Device includes the product chip part extended with additional emulation hardware. For detailed information about the Emulation Device functionality (e.g. as required by tool suppliers), please contact local Infineon representatives.

Figure 15-1 shows a block diagram of the TC1797 OCDS system.

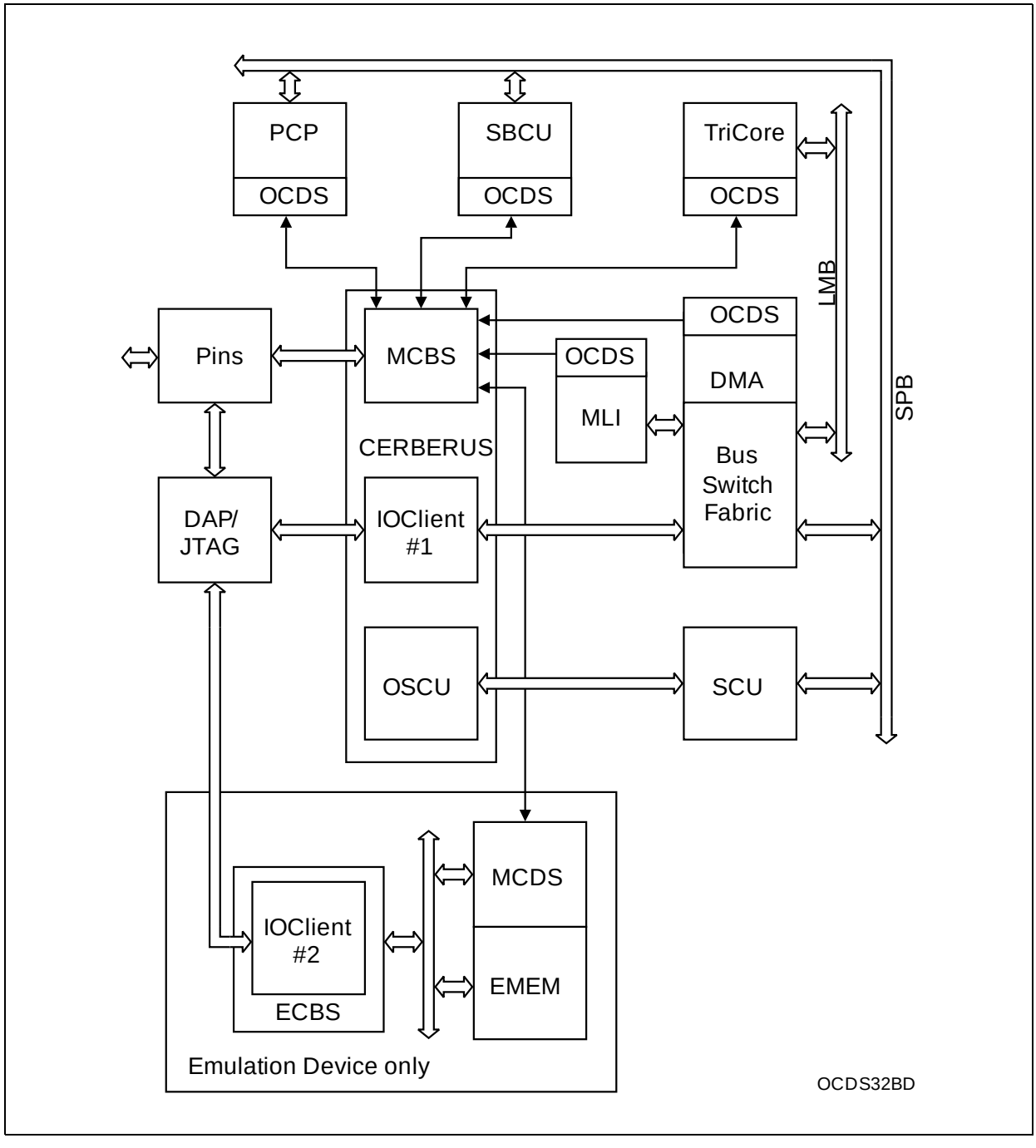


Figure 15-1 OCDS System Block Diagram

Components

The OCDS of the TC1797 consists of the following building blocks:

- OCDS System Control Unit (OSCU)
- TriCore OCDS (see “TriCore Architecture manual”)
- PCP OCDS
- DMA OCDS
- SBCU OCDS
- LMB OCDS
- Multi Core Break Switch (MCBS)
- Break Pin Control
- IOClient
- Device Access Port (DAP)
- JTAG Interface
- Overlay Control

Summary of OCDS Features

- OCDS System Control Unit (OSCU)
  - Controls OCDS enabling
  - Automatic Power Saving
  - Reset control of debug resources
  - Halt After Reset
  - Hot attach of a debugger to a running system
  - Key mechanism allows control the device access in a secure way.
  - State-aware watchdog timer suspension during debugging
  - Control of SoC specific OCDS features
  - Interrupt service request node for debug purposes
- TriCore OCDS features
  - Hardware event generation
  - Break by DEBUG instruction or Break signal from break switch
  - Suspend by Suspend bus or Break signal from break switch
  - Full hardware supported single step
  - Software Single-Step (code patching) is also possible
  - Concurrent access to memory and SFRs via Cerberus
- PCP OCDS features
  - Break by DEBUG instruction or Break signal from break switch
  - Concurrent access to memory and SFRs via Cerberus
- DMA OCDS features
  - Soft-suspend Mode of DMA channels
  - Break signal generation
  - Trace signal generation
- SBCU OCDS features
  - Event generation on specified transactions

- LMB OCDS features
  – Error recording and service request on bus error
- Multi-Core Break Switch (Cerberus MCBS)
  – TriCore, PCP, DMA, break pins and BCU available as break sources
  – TriCore and PCP available as break targets; other parts can be suspended in addition
  – Synchronous stop and restart of the system
  – Break to Suspend converter

15.2 OCDS Level 1

The basic principle of the TriCore OCDS Level 1 is that all relevant user and debug resources are memory mapped. These resources include on-chip memories, CPU core registers and registers of the peripheral units.

A typical OCDS Level 1 debugging configuration is shown in **Figure 15-2**. It comprises two parts:

- The tool software
- The tool access hardware interface adapter

This configuration makes it possible to realize a cost effective debugging environment that permits comprehensive real-time debugging tasks to be performed.

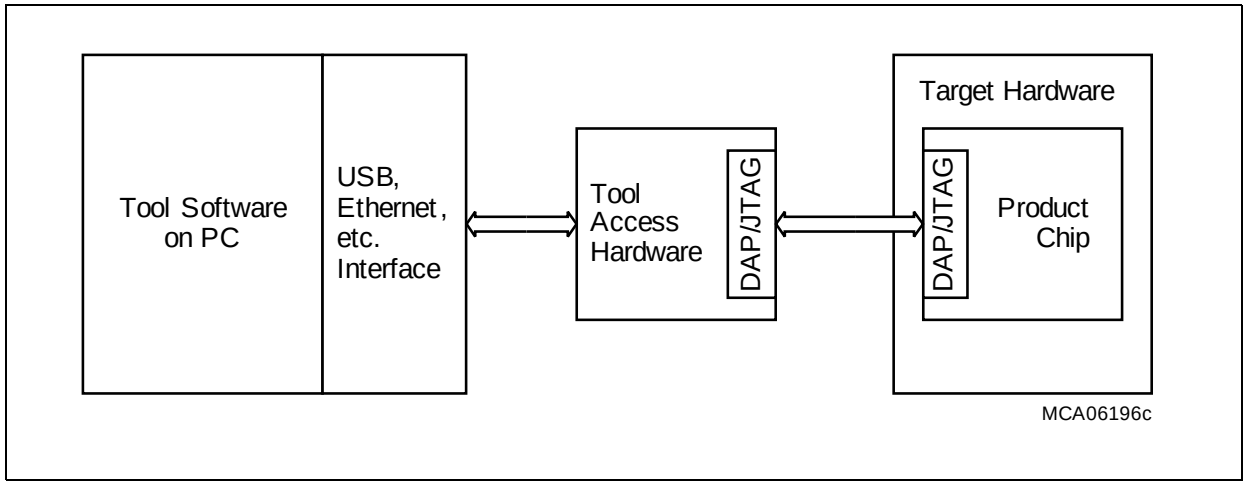


Figure 15-2 Typical OCDS Level 1 Hardware Connections

15.2.1 TriCore CPU OCDS Level 1

This section describes the basic features of the TriCore OCDS Level 1 hardware.

Features

- Single-step support by hardware

- Up to 4 programmable hardware breakpoints. Each one can be defined as a combination of program counter and data address:
  - Breaks on program counter (PC) value
    Two precise PC values or one PC range
    Break before make (BBM) possible
  - Breaks on data address
    Two precise data addresses or one data address range
    No break before make possible (due to pipelined architecture)
  - Combinations of the above break conditions
- Suspend features
  - Core Suspend-Out to suspend bus
  - Configurable Core Suspend-In
- Real-time features
  - Read and write of memory/registers quasi-concurrently, with minimum intrusion (stealing bus cycles by Cerberus)
  - High-priority requests can still be serviced when the core is in emulation mode - by interrupting the monitor program

15.2.1.1 Basic Concept

The TriCore CPU in the TC1797 provides OCDS with the following two basic parts:

- Debug Event Trigger Generation
- Debug Event Trigger Processing

The first part controls the generation of debug events and the second part controls what actions are taken when a debug event is generated.

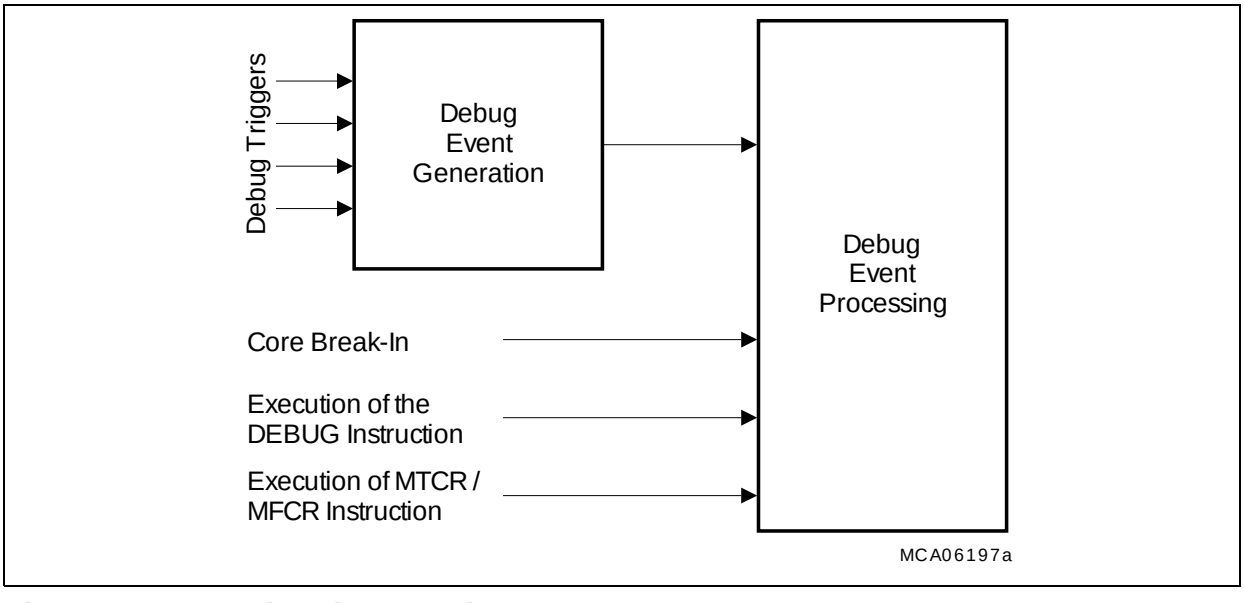


Figure 15-3 Basic TriCore Debug Concept

15.2.1.2 Debug Event Generation

If debug mode is enabled, debug events can be generated by:

- Debug event generation from debug triggers
- Activation of the external Core Break-In signal to the core
- Execution of a DEBUG instruction
- Execution of an MTCR/MFCR instruction

Debug Event Generation from Debug Triggers

The debug event generation unit is responsible for generating debug events when a programmable set of debug triggers is active. Debug triggers can be generated by:

- The code protection logic
- The data protection logic

These debug triggers provide the inputs to a programmable block of combinational logic that outputs debug events. The aim is to be able to specify the breakpoints that use fairly simple criteria purely in the on-chip debug event generation unit, and to rely on help from the external debug system or debug monitor to implement more complex breakpoints.

Activation of the External Core Break-in signal

When activating the TC1797 device pin $\overline{\text{BRKIN}} = 0$ and if MCBS and port control is configured to forward this event to the Core Break-In signal, a break event is induced as specified in an External Break Input Event Specifier Register EXEVT (**Figure 15-5**).

Execution of a DEBUG Instruction

The TriCore architecture supports a mechanism through which software can explicitly generate a debug event. This can be used, for instance, by a debugger to patch code held in RAM in order to implement breakpoints. A special DEBUG instruction is defined which is a user mode instruction, and its operation depends on whether the debug mode is enabled. 16-bit and 32-bit forms of the DEBUG instruction are provided.

If debug mode is enabled, the DEBUG instruction causes a debug event to be raised and the action defined in the Software Break Event Specifier Register SWEVT is taken. If the debug mode is not enabled, then the DEBUG instruction is treated as a NOP instruction.

Execution of an MTCR/MFCR Instruction

In order to protect the emulator resource, a debug event is raised whenever an MTCR or MFCR instruction is used to read or modify a user core SFR, but an event is not raised when the user reads or modifies one of the dedicated core debug registers: DBSR, CREVT, SWEVT, EXEVT, TR0EVT, TR1EVT, DMS, DCX or DBGTCR.

The action that is performed when an MTCR or MFCR instruction is executed on user core SFRs defined by the content of the Emulator Resource Protection Event Specifier Register CREVT.

15.2.1.3 Debug Actions

Four types of debug actions are available:

- Assert Core Break-out signal and $\overline{\text{BRKOUT}}$ (**Figure 15-5**) via MCBS unit and port control
- Halt the CPU core
- Cause a breakpoint trap
- Generate an interrupt request

These debug actions are selected by programming the corresponding Event Specifier registers. Their contents determine which action shall be taken when the corresponding debug event occurs. In parallel the Core Suspend-Out signal can be activated.

15.2.1.4 TriCore OCDS Registers

Figure 15-4 shows the TriCore OCDS registers in an overview.

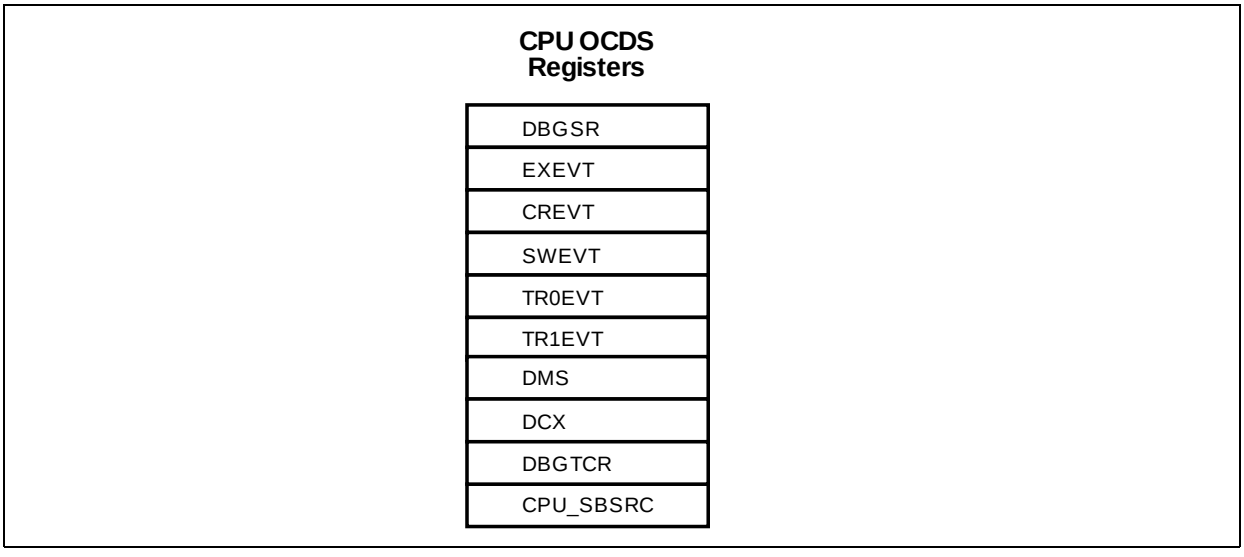


Figure 15-4 TriCore Core Debug Registers

Table 15-1 TriCore OCDS Registers

Register Short Name	Register Long Name	Address
DBGSR	Debug Status Register	F7E1 FD00$_H$
EXEVT	External Break Input Event Specifier Register	F7E1 FD08$_H$
CREVT	Core SFR Access Break Event Specifier Register	F7E1 FD0C$_H$
SWEVT	Software Break Event Specifier Register	F7E1 FD10$_H$
TR0EVT	Trigger Event 0 Register	F7E1 FD20$_H$
TR1EVT	Trigger Event 1 Register	F7E1 FD24$_H$
DMS	Debug Monitor Start Address Register	F7E1 FD40$_H$
DCX	Debug Context Save Area Pointer	F7E1 FD44$_H$
DBGTCR	Debug Trap Control Register	F7E1 FD48$_H$
CPU_SBSRC	CPU Software Breakpoint Service Request Control Register	F7E0 FFBC$_H$ [1)]

1) Located in the CPU slave (CPS) interface register area.

15.2.2 PCP OCDS Level 1

PCP has no hardware means of generating trigger events.

To set breakpoints, the debugger is expected to patch the code in the PCP code memory (CMEM) with DEBUG instructions. If a DEBUG instruction is executed, the running channel program is terminated, either with Debug Exit or with Error Exit.

The PCP has a suspend input, a break input and a break output. The Break Switch can send an external break request to the PCP, with the same consequence as above (Debug Exit or Error Exit).

15.2.3 SBCU OCDS Level 1

The BCU of the SPB bus in the TC1797 offers very powerful means for trigger generation. The BCU contains comparators for

- the arbitration phase (look for specific bus master)
- the address phase (look for specific address or range)
- the data phase (look for read, write, supervisor mode, etc.)

The results can be combined to generate a break request signal, which is sent to the Break Switch.

The OCDS registers of SBCU are described in chapter “On-Chip System Buses and Bus Bridges” starting from section “System Bus Control Unit Registers”.

15.2.4 DMA OCDS Level 1

The DMA controller in the TC1797 provides the following debugging capabilities:

- Hard suspend mode of the DMA controller (for test purposes only)
- Soft suspend mode of DMA channels
- Break signal generation

In suspend modes, the operations of DMA channels or the complete DMA module are stopped. Under certain conditions, a break signal is also generated for the on-chip debug support logic.

More details on the OCDS Level 1 debug capabilities of the DMA controller are provided in chapter “Direct Memory Access Controller (DMA)” in section “On-Chip Debug Capabilities”.

15.3 Debug Interface (Cerberus)

The Cerberus module is the on-chip unit that controls all OCDS main debug functions. Generally, the Cerberus may not be used by any application software, since this could disturb the emulation tool behavior.

The Cerberus module is built up by three parts (see also **Figure 15-1**):

- OCDS System Control Unit - OSCU
- IOClient
- Multi Core Break Switch - MCBS

A tool can be connected to the device in two ways:

- A two line DAP interface via the DAP module receives the debugger commands, converts them and outputs them to the IOClient interface.
- Standard JTAG interface is connected via the JTAG controller with the IOClient interface

Two additional pins are available to handle an external break condition. The external debug hardware can access the Cerberus registers and arbitrary memory locations across the System Peripheral Bus.

Features

- OCDS Level 1 control via
  - 5-pin standard JTAG interface
  - 2-pin DAP interface
- Generation of external break condition via $\overline{\text{BRKIN}}$/$\overline{\text{BRKOUT}}$ pins possible
- Full access to the complete SPB Bus address space via DAP/JTAG
- No user resources (hardware/software) are required
- No or minimum run-time impact (Cerberus bus priority can be controlled)
- Generic memory read/write functionality
- Write word, half-word and byte
- Block read and write
- Full support for communication between an on-chip monitor program and the external debugger
- Pending reads/writes can be optionally triggered by the OCDS module (memory tracing)
- Download of programs and data via DAP/JTAG
- Control of the OCDS blocks
- Data acquisition

15.3.1 RW Mode

As the name implies, the RW mode is used by a DAP/JTAG host to read or write arbitrary memory locations via the DAP/JTAG interface. The RW mode needs the SPB Bus master interface of the Cerberus to actively request data reads or data writes.

Data Types Supported

- WORD (32-bit): The default data type; used for single word transfers and block transfers.
- HWORD (16-bit): For reading 16-bit registers without getting a bus error, a dedicated IOClient instruction is provided.
- BYTE (8-bit): If the DAP/JTAG host wants to read a byte, it has to read the associated word or half-word. Then the host has to extract the part needed itself.

15.3.2 Communication Mode

In Communication Mode, the Cerberus has no access to the internal buses and communication is established between the external DAP/JTAG host and a software monitor (embedded into the application program) via the Cerberus registers. The communication mode is the default mode after reset.

In Communication Mode, the external DAP/JTAG host is master of all transactions. The host requests the monitor to write or read a value to/from the Cerberus register COMDATA. The difference to RW Mode is that the read or write request is not actively executed by Cerberus. The request just sets bits in the CPU accessible IOSR register to signal the monitor that the debugger wants to send or receive a value. The software monitor has to poll the IOSR register for that.

15.3.3 Triggered Transfers

Triggered transfers can be used to read from or write to a certain memory location when an OCDS trigger becomes active.

The main application for Triggered Transfers is to trace a certain memory location. This can be done, when the OCDS of the CPU activates its break out signal, if this memory location is written by the user program. This event is used as a transfer trigger through the configuration of the MCBS. Cerberus is configured to read the location on this trigger.

15.3.4 Multi Core Break Switch

In TC1797, there are several sources and targets for break signals. For instance the OCDS run control of one processor can break the other processor unit. The Multi Core Break Switch (MCBS) is a part of the Cerberus and allows to propagate break requests from sources to targets in a generic and flexible way. **Figure 15-5** shows the break signal interfaces of the MCBS.

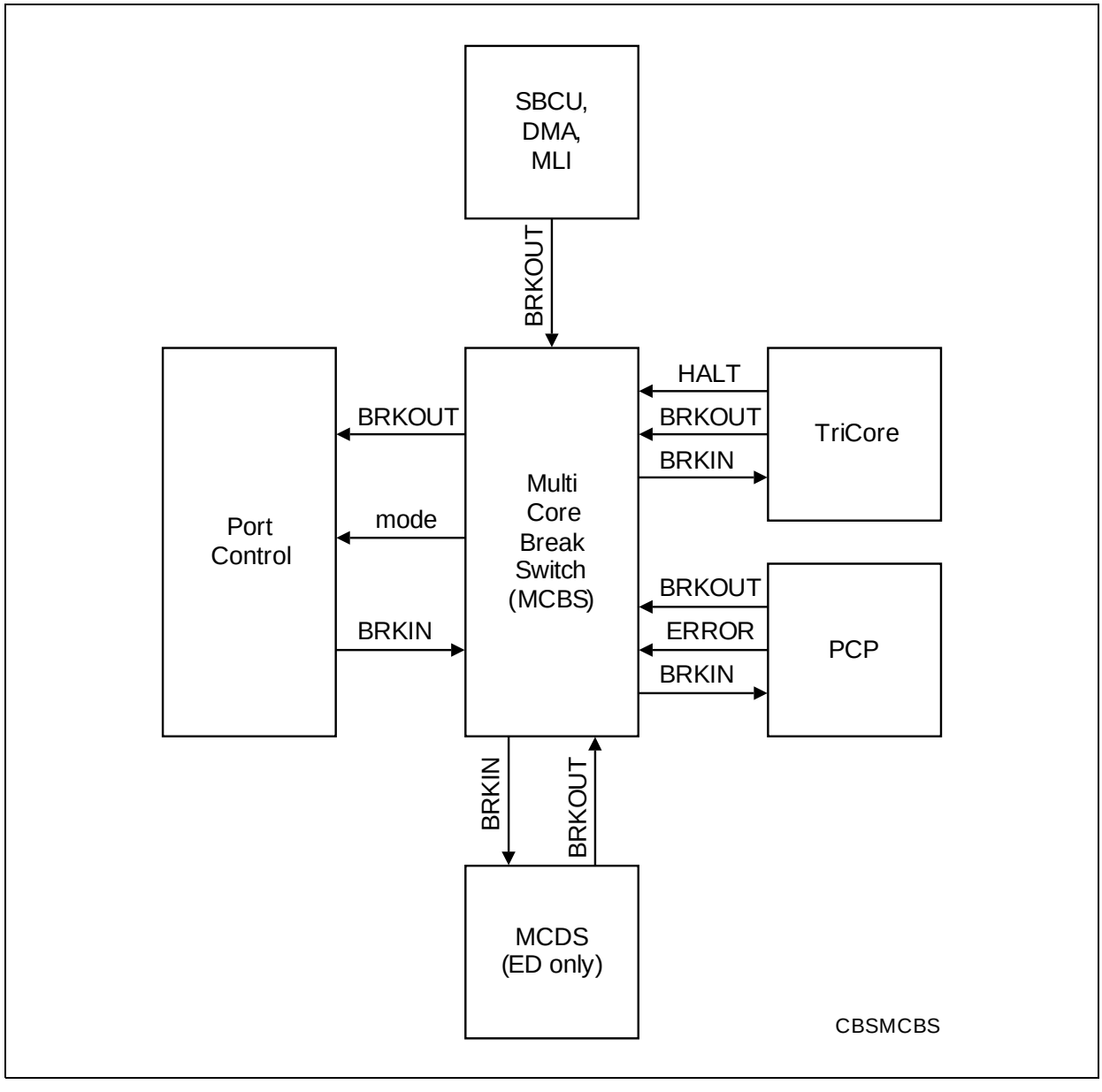


Figure 15-5 Break Switch Interfaces

The MCBS unit supports the following features:

- Six break-in sources (TriCore, PCP, DMA, SBCU, MLI0, MLI1)
- Up to two device pins configurable as BRKIN/BRKOUT pins
- Two independent break buses
- Suspend generation supports delayed suspend
- Break-to-suspend converter
- Create interrupt request with a break coming from a source
- Synchronous restart of the system

15.4 JTAG Interface

The JTAG interface is a standardized unit that is typically used for boundary scan and internal device tests. Because both of these applications are not active during normal device operation in a system, the JTAG port can be used during normal device operation as an ideal interface for debugging tasks.

15.5 Device Access Port (DAP)

The cost inferred by each non-functional pin is a strong argument to reduce the tool access port to as few pins as possible. The standardized Device Access Port (DAP) of Infineon's latest micro controllers offers a convenient method to get the required functionality at the least possible cost. With DAP only two pins (DAP0 for the clock, DAP1 for the bidirectional data) are needed to communicate with the tool.

DAP uses a straightforward half-duplex protocol, i.e. the DAP1 pin is used for data transfer from tool to device and from device to tool at different periods of time while the clock is provided by the tool to the DAP0 input.

15.5.1 DAP Telegram Format

All information transport between tool and device is done in telegrams. Mandatory 6 bit CRC check sums assure secure transport even in noisy environments. Splitting command and reply into separate units transported sequentially allows half-duplex transmission over a single bidirectional line. The physical interface medium can be chosen independently as long as the serial bit stream can be transported.

15.5.2 DAP Telegram Catalog

This chapter lists the telegrams implemented by the TC1797. Other telegrams are silently ignored, resulting in a time-out condition on the tool side.

Three groups of telegrams can be distinguished:

Control Telegrams

These four telegrams are needed to establish and maintain the connection from tool to device as such. No data is transported.

- sync - request synchronization pattern
- turn_off - shut down DAP
- poll - get the current service request
- set_maxwait - adjust the parameter for time out

JTAG Telegrams

The telegrams of this group are simple wrappers around the standard JTAG commands, adding the relaxed timing and increased transmission speed and quality of DAP.

- jtag_mode - switch DAP to BYPASS mode
- jtag_reset - reset the TAP controller
- jtag_setIR - write the TAP's INSTRUCTION register
- jtag_swapIR - write and read the TAP's INSTRUCTION register
- jtag_setDR - write the current JTAG data register
- jtag_swapDR - write and read the current JTAG data register
- jtag_moreDR - write and read part of a long JTAG data register

Client Telegrams

The last group allows direct access to IOClients like Cerberus of the device, completely hiding the asynchronous timing between tool clock and system clock of the device.

The following five telegrams belong to this group:

- client_set - define the current IOClient
- client_get - ask for the index of the current IOClient
- client_reset - reset the current IOClient
- client_write - write to the current IOClient
- client_read - read from the current IOClient

15.6 Cerberus and JTAG Registers

This section summarizes all Cerberus and JTAG registers for reference purposes. Details on these registers are contained in OCDS documents that are available for tool suppliers on request (please contact local Infineon representatives). All CERBERUS registers are prefixed "CBS_" in the register map of a device.

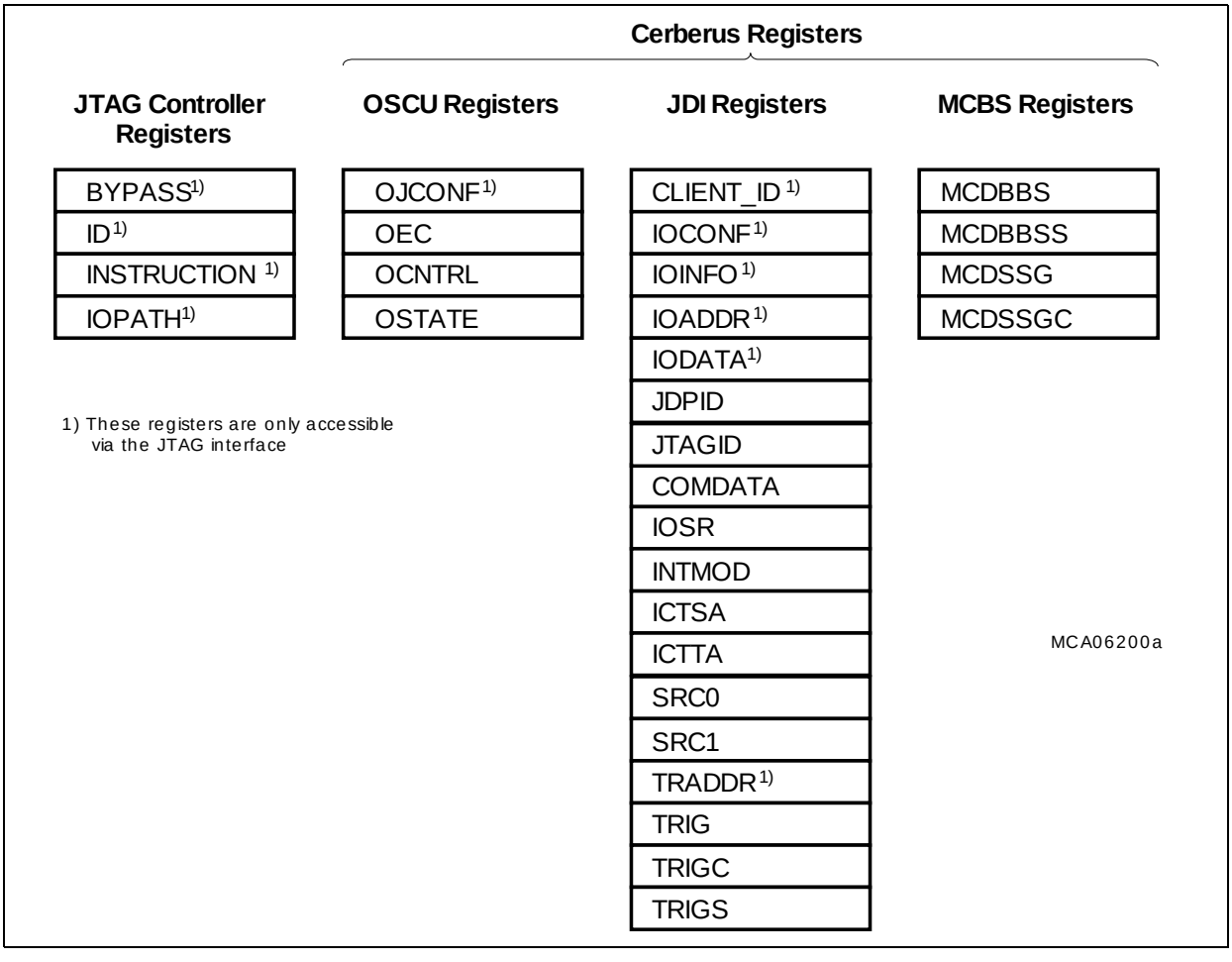


Figure 15-6 JTAG/Cerberus Register Overview

Table 15-2 JTAG/Cerberus Register Overview

Register Short Name	Register Long Name	Address
JTAG Controller Registers		
BYPASS	JTAG Bypass Register (1-bit)	1)
ID	JTAG Device Identification Register (32-bit)	1)
INSTRUCTION	JTAG Instruction Register (8-bit)	1)
IOPATH	IO Client Selection Register (2-bit)	1)
Cerberus Registers		
OJCONF	OSCU Configuration by JTAG Register	1)
OEC	OCDS Enable Control Register	F000 0478$_H$
OCNTRL	OSCU Configuration and Control Register	F000 047C$_H$

Table 15-2 JTAG/Cerberus Register Overview (cont'd)

Register Short Name	Register Long Name	Address
OSTATE	OSCU Status Register	F000 0480$_H$
CLIENT_ID	JTAG Client Identification Register (32-bit)	1)
IOCONF	Configuration Register (12-bit)	1)
IOINFO	State Information for Error Analysis Register (16-bit)	1)
IOADDR	Address for Data Access Register (32-bit)	1)
IODATA	RW Mode Data Register (32-bit)	1)
JDPID	Cerberus Module Identification Register	F000 0408$_H$
JTAGID	JTAG Device Identification Register	F000 0464$_H$
COMDATA	Communication Mode Data Register	F000 0468$_H$
IOSR	Status Register	F000 046C$_H$
INTMOD	Internal Mode Status and Control Register	F000 0484$_H$
ICTSA	Internal Controlled Trace Source Address Register	F000 0488$_H$
ICTTA	Internal Controlled Trace Target Address Register	F000 048C$_H$
MCDBBS	Break Bus Switch Configuration Register	F000 0470$_H$
MCDBBSS	Break Bus Switch Status Register	F000 0490$_H$
MCDSSG	Suspend Signal Generation Status and Control Register	F000 0474$_H$
MCDSSGC	Suspend Signal Generation Configuration Register	F000 0494$_H$
SRC0	Service Request Control Register 0	F000 04FC$_H$
SRC1	Service Request Control Register 1	F000 04F8$_H$
TRADDR	Triggered Transfer Destination Address	1)
TRIG	Trigger to Host	F000 04A8$_H$
TRIGC	Clear Trigger to Host	F000 04A4$_H$
TRIGS	Set Trigger to Host	F000 04A0$_H$

1) These registers are only accessible via the JTAG interface.

16 Asynchronous/Synchronous Serial Interface (ASC)

This chapter describes the two ASC Asynchronous/Synchronous Serial Interfaces, ASC0 and ASC1, of the TC1797. It contains the following sections:

- Functional description of the ASC kernel, valid for ASC0 and ASC1 (see **Page 16-1**)
- ASC kernel register description, describes all ASC kernel specific registers (see **Page 16-19**)
- TC1797 implementation-specific details and registers of the ASC0/ASC1 modules (see **Page 16-30**).

*Note: The ASC kernel register names described in **Section 16.2** are referenced in the TC1797 User´s Manual by the module name prefix "ASC0_" for the ASC0 interface and by "ASC1_" for the ASC1 interface.*

16.1 ASC Kernel Description

Figure 16-1 shows a global view of the ASC interface.

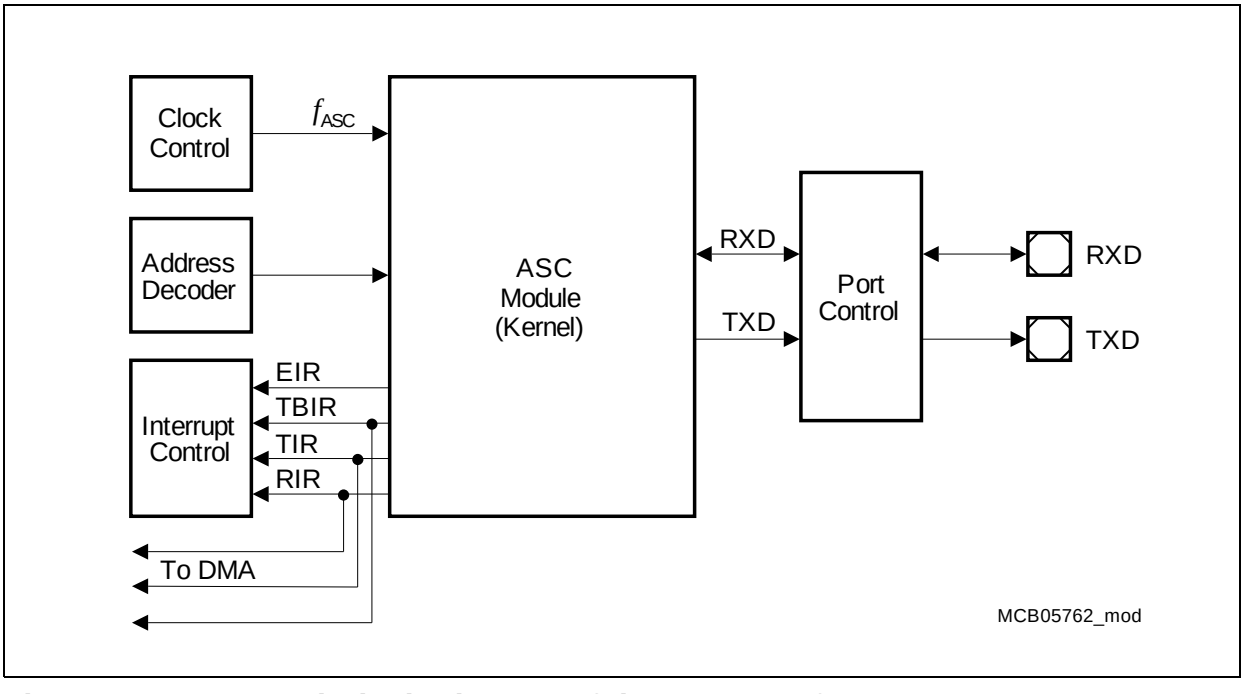


Figure 16-1 General Block Diagram of the ASC Interface

The ASC module communicates with the external world via two I/O lines. The RXD line is the receive data input signal (and also output signal in Synchronous Mode), and TXD is the transmit output signal.

Clock control, address decoding, and interrupt service request control are managed outside the ASC module kernel.

16.1.1 Overview

The ASC provides serial communication between the TC1797 and other microcontrollers, microprocessors, or external peripherals.

The ASC supports full-duplex asynchronous communication and half-duplex synchronous communication. In Synchronous Mode, data is transmitted or received synchronous to a shift clock that is generated by the ASC internally. In Asynchronous Mode, 8-bit or 9-bit data transfer, parity generation, and the number of stop bits can be selected. Parity, framing, and overrun error detection are provided to increase the reliability of data transfers. Transmission and reception of data is double-buffered. For multiprocessor communication, a mechanism is included to distinguish address bytes from data bytes. Testing is supported by a loop-back option. A 13-bit baud rate generator provides the ASC with a separate serial clock signal, which can be accurately adjusted by a prescaler implemented as fractional divider.

Features

- Full-duplex asynchronous operating modes
  - 8-bit or 9-bit data frames, LSB first
  - Parity-bit generation/checking
  - One or two stop bits
  - Baud rate from 5.625 Mbit/s to 1.34 bit/s (@ 90 MHz module clock)
  - Multiprocessor mode for automatic address/data byte detection
  - Loop-back capability
- Half-duplex 8-bit synchronous operating mode
  - Baud rate from 11.25 Mbit/s to 915.5 bit/s (@ 90 MHz module clock)
- Double-buffered transmitter/receiver
- Interrupt generation
  - On a transmit buffer empty condition
  - On a transmit last bit of a frame condition
  - On a receive buffer full condition
  - On an error condition (frame, parity, overrun error)
- Implementation features
  - Connections to DMA Controller
  - Connections of receiver input to GPTA (LTC) for baud rate detection and LIN break signal measuring

16.1.2 General Operation

The ASC supports full-duplex asynchronous communication up to 5.625 Mbit/s and half-duplex synchronous communication up to 11.25 Mbit/s (@ 90 MHz module clock). In Synchronous Mode, data is transmitted or received synchronous to a shift clock generated by the microcontroller. In Asynchronous Mode, 8-bit or 9-bit data transfer, parity generation, and the number of stop bits can be selected. Parity, framing, and overrun error detection are provided to increase the reliability of data transfers. Transmission and reception of data are double-buffered. For multiprocessor communication, a mechanism is included to distinguish address bytes from data bytes. Testing is supported by a loop-back option. A 13-bit baud rate generator provides the ASC with a separate serial clock signal, which can be accurately adjusted by a prescaler implemented as fractional divider.

A transmission is started by writing to the Transmit Buffer Register, TBUF. Only the number of data bits determined by the selected operating mode will actually be transmitted; that is, bits written to positions 9 through 15 of register TBUF are always insignificant. Data transmission is double-buffered, so a new character may be written to TBUF before the transmission of the previous character is complete. This allows a back-to-back transmission of characters to take place without gaps.

Data reception is enabled by the receiver enable bit CON.REN. After a reception has been completed, the received data and, if provided by the selected operating mode, the received parity bit can be read from the (read-only) receive buffer register RBUF. Unused bits in the upper half of RBUF that are not required in the selected operating mode will be read as zeros.

Data reception is double-buffered, so that reception of a second character may already begin before the previously received character has been read out of the receive buffer register. In all modes, receive buffer overrun error detection can be selected through bit CON.OEN. When enabled, the overrun error status flag CON.OE and the error interrupt request line EIR will be activated when the receive buffer register has not been read by the time reception of a second character is complete. In this case, the previously received character in the receive buffer is overwritten.

The loop-back option (selected by bit CON.LB) allows the data currently being transmitted to be received simultaneously in the receive buffer. This may be used to test serial communication routines at an early stage without having to provide an external network. In loop-back mode, the alternate input/output function of port pins is not required.

16.1.3 Asynchronous Operation

Asynchronous Mode supports full-duplex communication, in which both transmitter and receiver use the same data frame format and have the same baud rate. Data is transmitted on pin TXD and received on pin RXD. **Figure 16-2** shows the block diagram of the ASC when operating in Asynchronous Mode.

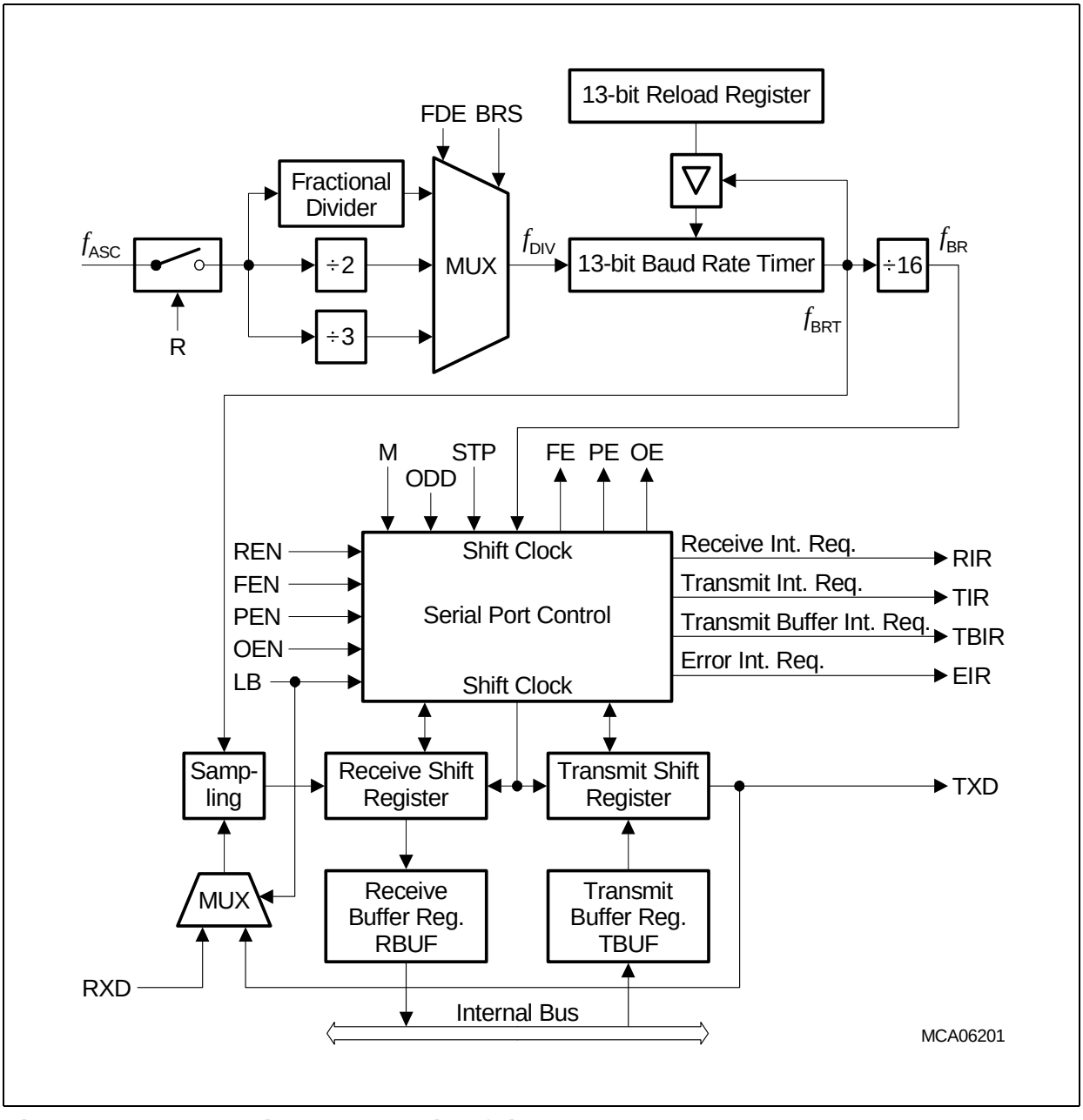


Figure 16-2 Asynchronous Mode of the ASC

16.1.3.1 Asynchronous Data Frames

Asynchronous data frames can consist of 8-bit or 9-bit data frames.

8-bit Data Frames

The 8-bit data frames consist of either eight data bits D7 … D0 (CON.M = 001_B), or of seven data bits D6 … D0 plus an automatically generated parity bit (CON.M = 011_B). Parity may be odd or even, depending on bit CON.ODD. An even parity bit will be set if the modulo-2 sum of the seven data bits is 1. An odd parity bit will be cleared in this case. Parity checking is enabled via bit CON.PEN (always OFF in 8-bit data mode). The parity error flag CON.PE will be set, along with the error interrupt request flag, if a wrong parity bit is received. The received parity bit itself will be stored in RBUF too.

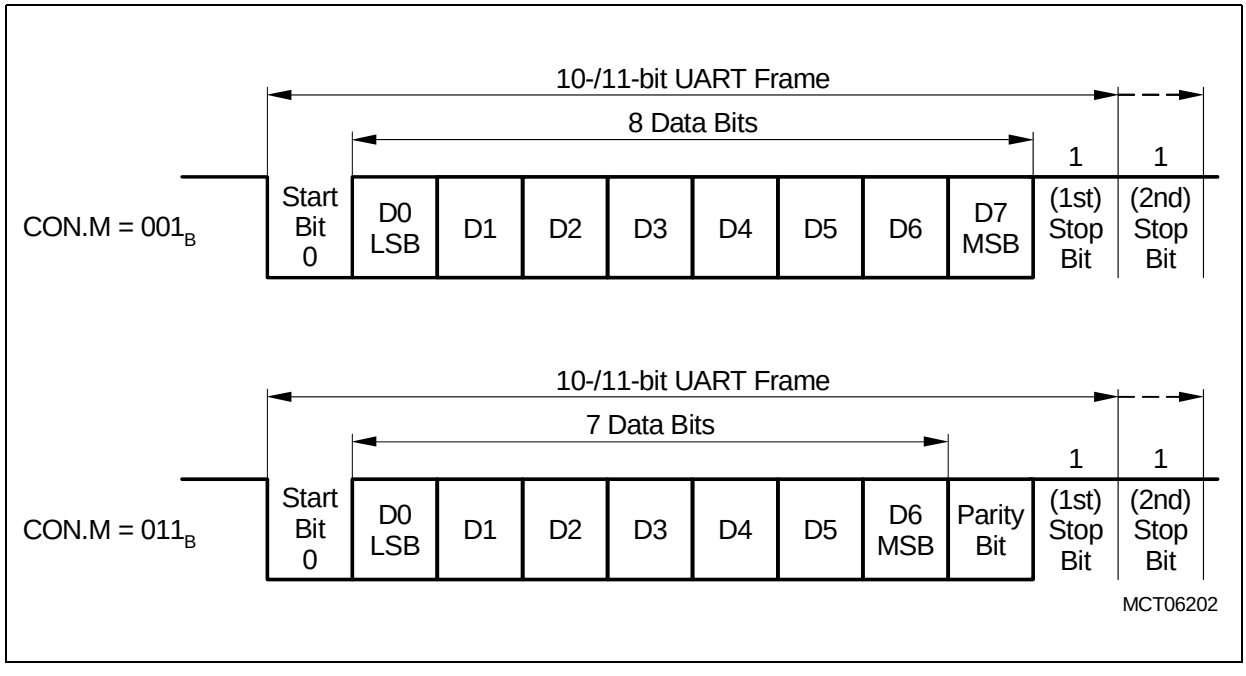


Figure 16-3 Asynchronous 8-bit Frames

9-bit Data Frames

The 9-bit data frames consist of nine data bits D8 … D0 (CON.M = 100_B), or of eight data bits D7 … D0 plus an automatically generated parity bit (CON.M = 111_B) or of eight data bits D7 … D0 plus wake-up bit (CON.M = 101_B). Parity may be odd or even, depending on bit CON.ODD. An even parity bit will be set if the modulo-2-sum of the eight data bits is 1. An odd parity bit will be cleared in this case. Parity checking is enabled via bit CON.PEN (always OFF in 9-bit data and wake-up mode). The parity error flag CON.PE will be set along with the error interrupt request flag if a wrong parity bit is received. The received parity bit itself will be stored in RBUF too.

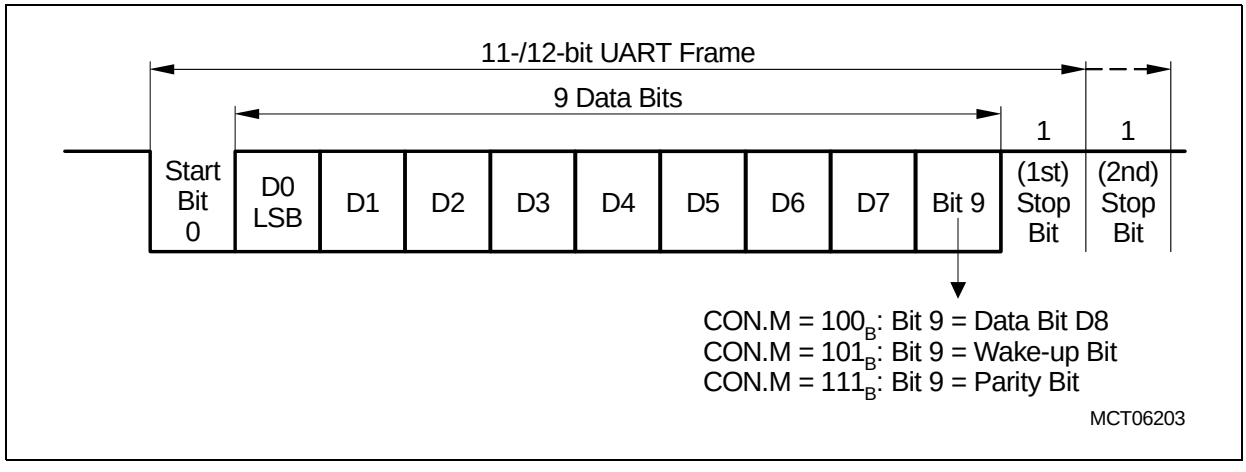


Figure 16-4 Asynchronous 9-bit Frames

In Wake-up Mode (CON.M = 101_B), received frames are transferred to the receive buffer register only if the 9^{th} bit (the wake-up bit) of the frame is 1. If this bit is 0, no receive interrupt request will be activated and no data will be transferred.

This feature can be used to control communication in multi-processor systems, for example:
When the master processor aims to transmit a block of data to one of several slaves, it first sends out an address ‘byte’ (in this case, a ‘byte’ consists of nine bits) that identifies the target slave. An address ‘byte’ differs from a data ‘byte’ in that the additional 9^{th} bit is a 1 for an address ‘byte’ but is a 0 for a data ‘byte’, so, no slave will be interrupted by a data ‘byte’. An address ‘byte’ will interrupt all slaves (operating in 8-bit data + wake-up bit mode), so each slave can examine the eight LSBs of the received character (the address). The addressed slave will switch to 9-bit data mode (for example, by clearing bit CON.M[0]), which enables it to also receive the data bytes that will be coming (having the wake-up bit cleared). The slaves that were not being addressed remain in 8-bit data + wake-up bit mode, ignoring the following data ‘bytes’.

16.1.3.2 Asynchronous Transmission

Asynchronous transmission begins when the next overflow of the divide-by-16 baud rate timer (transition of the baud rate clock f_{BR}) occurs, if bit CON.R is set and data has been loaded into TBUF. The transmitted data frame consists of three elements:

1. The start bit
2. The data field (8 or 9 bits, LSB first, including a parity bit, if selected)
3. The delimiter (1 or 2 stop bits)

Data transmission is double-buffered. When the transmitter is idle, the transmit data loaded into TBUF is immediately moved to the transmit shift register; thus, freeing TBUF for the next transmit data to be loaded. This is indicated by the transmit buffer interrupt request line TBIR being activated. TBUF may then be loaded with the next transmit data while transmission of the previous one continues.

The Transmit Interrupt Request line TIR will be activated before the last bit of a frame is transmitted, that is, before the first or the second stop bit is shifted out of the transmit shift register.

Note: A dedicated GPIO device pin which is connected to the module output pin TXD must be configured by software as alternate data output for asynchronous transmission.

16.1.3.3 Asynchronous Reception

Asynchronous reception is initiated by a falling edge (1-to-0 transition) on pin RXD, on the condition that bits CON.R and CON.REN are set. The receive data input pin RXD is sampled at sixteen times the rate of the selected baud rate. A majority decision of the 7^{th}, 8^{th} and 9^{th} sample determines the effective sampled bit value. This avoids erroneous results that may be caused by noise.

If the detected value is not a 0 when the start bit is sampled, the receive circuit is reset and waits for the next 1-to-0 transition at pin RXD. If the start bit proves valid, the receive circuit continues sampling and shifts the incoming data frame into the receive shift register.

When the last stop bit has been received, the contents of the receive shift register are transferred to the Receive Data Buffer Register RBUF. Simultaneously, the receive interrupt request line RIR is activated after the 9^{th} sample in the last stop bit time-slot (as programmed), regardless whether valid stop bits have been received or not. The receive circuit then waits for the next start bit (1-to-0 transition) at the receive data input line.

Note: A dedicated GPIO pin that is connected to the module input pin RXD must be configured by software as input for asynchronous reception.

Asynchronous reception is stopped by clearing bit CON.REN. A currently received frame is completed including generation of the receive interrupt request and an error interrupt request, if appropriate. Start bits that follow this frame will not be recognized.

Note: In wake-up mode, received frames are transferred to the receive buffer register only if the 9th bit (the wake-up bit) is 1. If this bit is 0, no receive interrupt request will be activated and no data will be transferred.

16.1.3.4 RXD/TXD Data Path Selection in Asynchronous Modes

The data paths for the serial input and output data in Asynchronous Modes are affected by control bit CON.LB (loop-back) as shown in **Figure 16-5**.

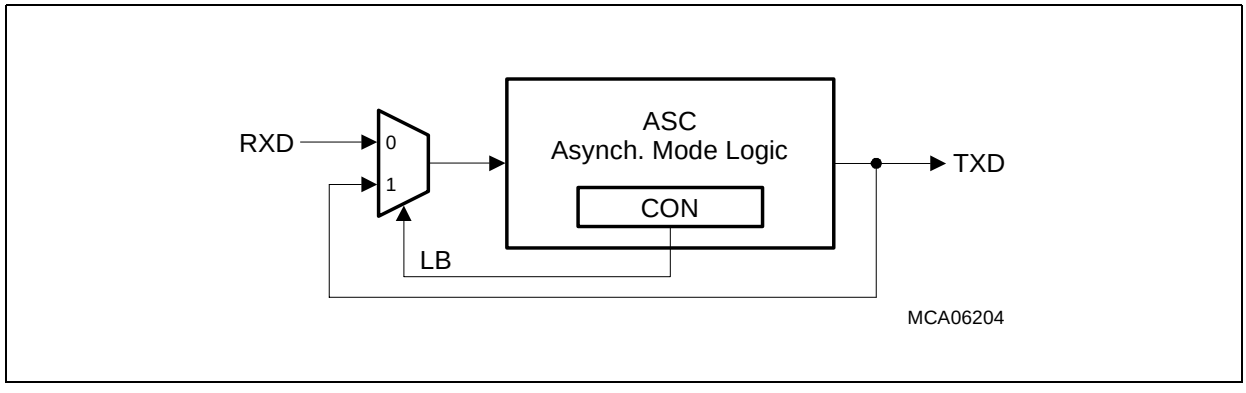


Figure 16-5 RXD/TXD Data Path Selection in Asynchronous Modes

16.1.4 Synchronous Operation

Synchronous Mode supports half-duplex communication, usable for simple I/O expansion via shift registers. Data is transmitted and received via pin RXD while pin TXD outputs the shift clock. These signals are typically connected as alternate functions with GPIO port pins. Synchronous Mode is selected with CON.M = 000_B.

Eight data bits are transmitted or received synchronous to a shift clock generated by the internal baud rate generator. The shift clock is active only as long as data bits are transmitted or received.

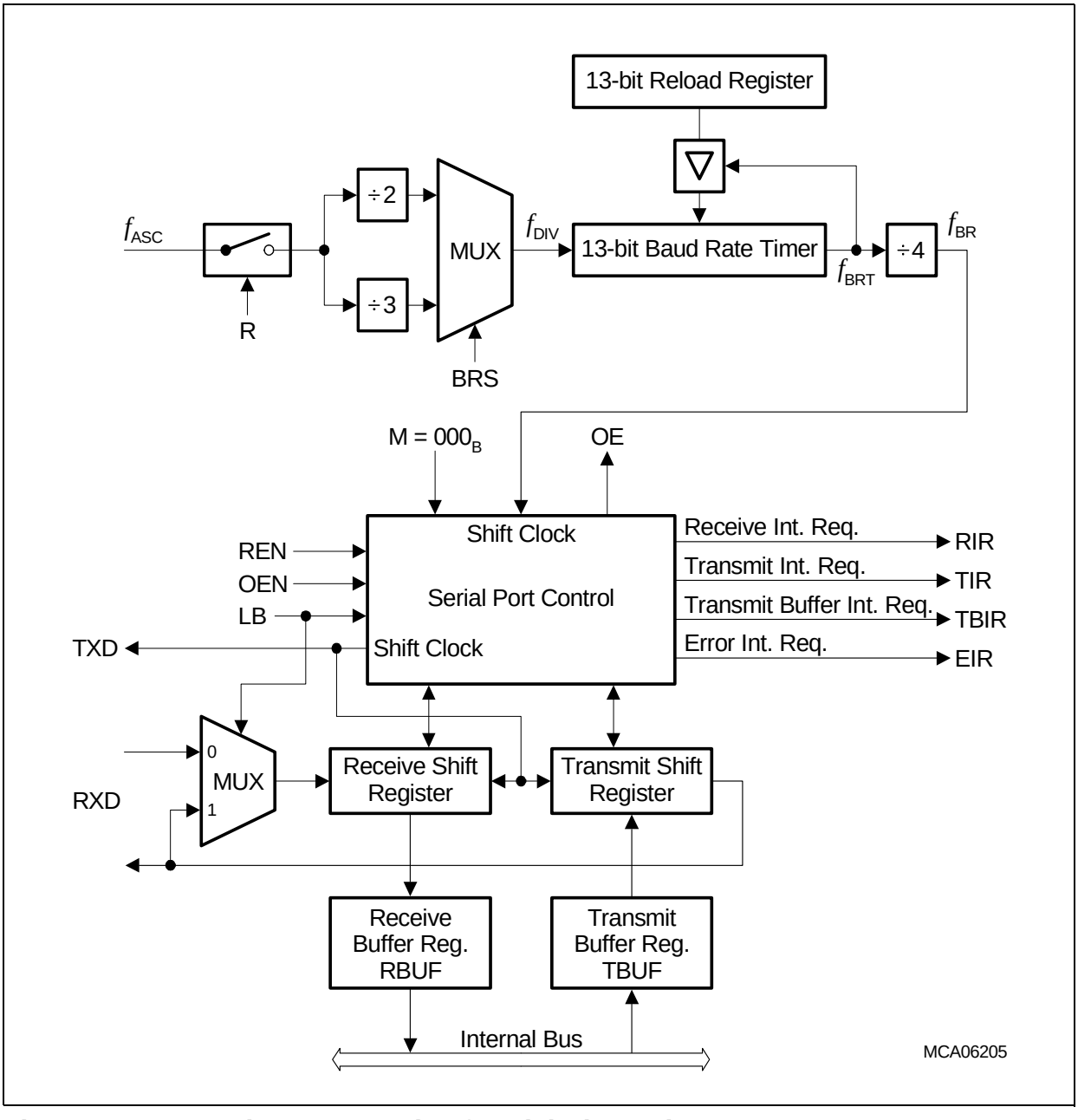


Figure 16-6 Synchronous Mode of Serial Channel ASC

16.1.4.1 Synchronous Transmission

Synchronous transmission begins within four state times after data has been loaded into TBUF, provided that CON.R is set and CON.REN = 0 (half-duplex, no reception), with one exception: in Loop-back Mode (bit CON.LB set), CON.REN must be set for reception of the transmitted byte. Data transmission is double-buffered. When the transmitter is idle, the transmit data loaded into TBUF is immediately moved to the transmit shift register, thus freeing TBUF for the next data to be sent. This is indicated by the transmit buffer interrupt request line TBIR being activated. TBUF may now be loaded with the next data, while transmission of the previous one continues. The data bits are transmitted synchronously with the shift clock. After the bit time for the 8th data bit, both TXD and RXD will be set to high level, the transmit interrupt request line TIR is activated, and serial data transmission stops.

Note: The dedicated GPIO device pins that are connected to TXD and RXD must be configured by software as alternate data outputs in order to provide the shift clock and the output data during synchronous transmission.

16.1.4.2 Synchronous Reception

Synchronous reception is initiated by setting bit CON.REN = 1. If bit CON.R = 1, the data applied at RXD is clocked into the receive shift register synchronously to the clock which is output at TXD. After the 8th bit has been shifted in, the contents of the receive shift register are transferred to the receive data buffer RBUF, the receive interrupt request line RIR is activated, the receiver enable bit CON.REN is reset, and serial data reception stops.

Synchronous reception is stopped by clearing bit CON.REN. Any byte that is currently being received is completed, including the generation of the receive interrupt request and an error interrupt request, if appropriate. Writing to the transmit buffer register while a reception is in progress has no effect on reception and will not start a transmission.

If a previously received byte has not been read out of the receive buffer register by the time the reception of the next byte is complete, both the error interrupt request line EIR and the overrun error status flag CON.OE will be activated/set, provided that the overrun check has been enabled by bit CON.OEN.

Note: The dedicated GPIO device pin that is connected to TXD must be configured by software as alternate data output in order to provide the shift clock. The dedicated GPIO device pin that is connected to RXD must be configured by software as input during synchronous reception.

16.1.4.3 Synchronous Timing

Figure 16-7 shows timing diagrams of the ASC Synchronous Mode data reception and data transmission. In idle state, the shift clock is at high level. With the beginning of a synchronous transmission of a data byte, the data is shifted out at RXD with the falling edge of the shift clock. If a data byte is received through RXD, data is latched with the rising edge of the shift clock.

One shift clock cycle (f_{BR}) delay is inserted between two consecutive receive or transmit data bytes.

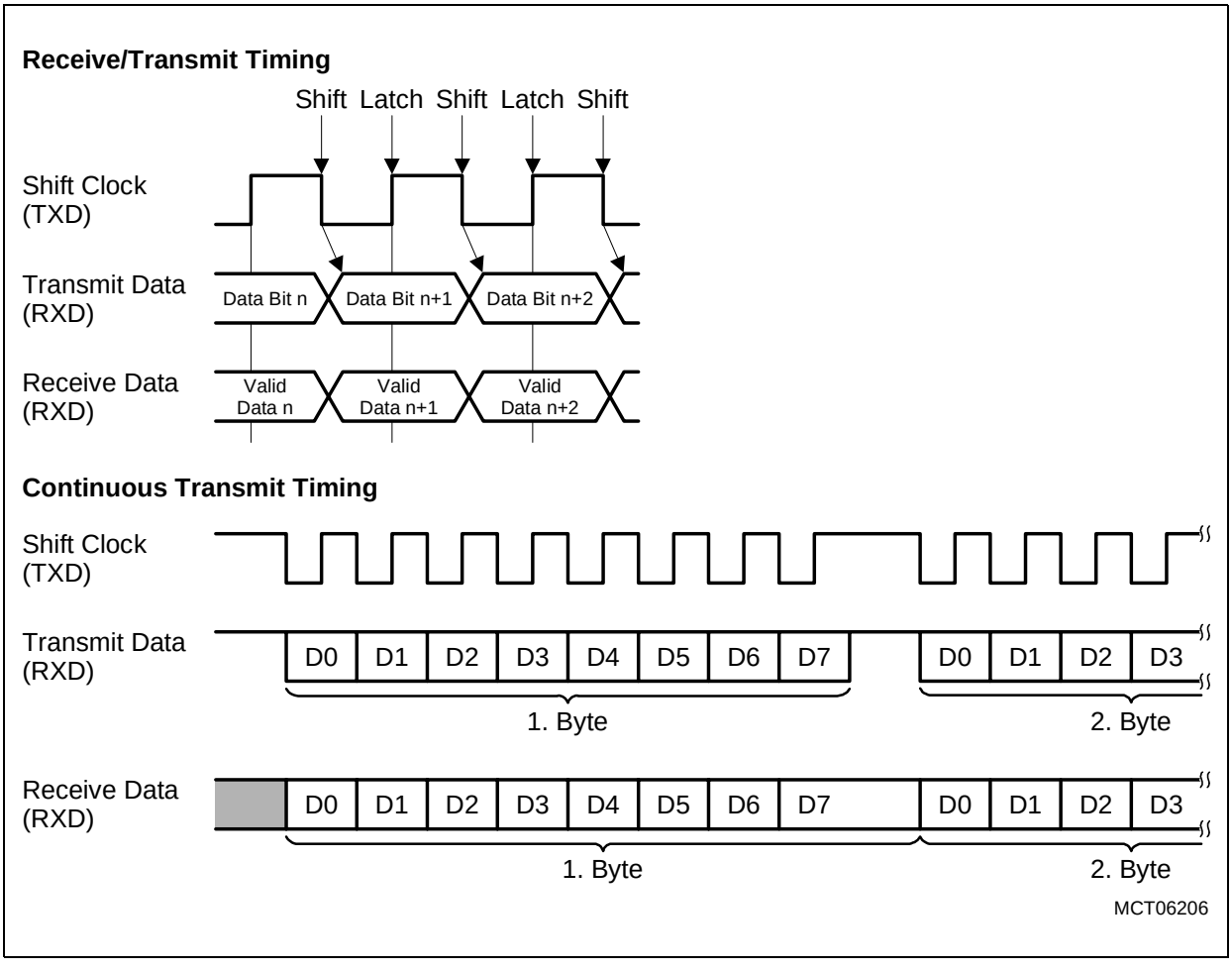


Figure 16-7 ASC Synchronous Mode Waveforms

16.1.5 Baud Rate Generation

The ASC has its own dedicated 13-bit baud rate generator with 13-bit reload capability, allowing baud rate generation independent of other timers.

The baud rate generator is clocked with a clock (f_{DIV}) which is derived via a prescaler from the ASC module clock f_{ASC}. The baud rate timer is counting downwards and can be started or stopped through the baud rate generator run bit CON.R. Each underflow of the timer generates one clock pulse to the serial channel. The timer is reloaded with the value stored in its 13-bit reload register at each underflow. The resulting clock f_{BRT} is again divided by a factor for the baud rate clock (÷ 16 in asynchronous operating modes and ÷ 4 in synchronous operating mode). The prescaler is selected by the bits CON.BRS and CON.FDE. In the asynchronous operating modes, a fractional divider prescaler unit is available (in addition to the two fixed dividers) that allows selection of prescaler divider ratios of n/512 with n = 0-511. Therefore, the baud rate of ASC is determined by the module clock, the content of register FDV, the reload value in register BG, and the operating mode (asynchronous or synchronous).

Register BG is the dual-function baud rate generator/reload register. Reading BG returns the contents of the timer in bit field BR_VALUE (bits 31:13 return zero), while writing to BG always updates the reload register (bits 31:13 are insignificant).

An auto-reload of the timer with the contents of the reload register is performed each time BG is written to. However, if CON.R = 0 at the time the write operation to BG is performed, the timer will not be reloaded until the first instruction cycle after CON.R = 1. For a clean baud rate initialization, BG should only be written if CON.R = 0. If BG is written with CON.R = 1, an unpredictable behavior of the ASC may occur during running transmit or receive operations.

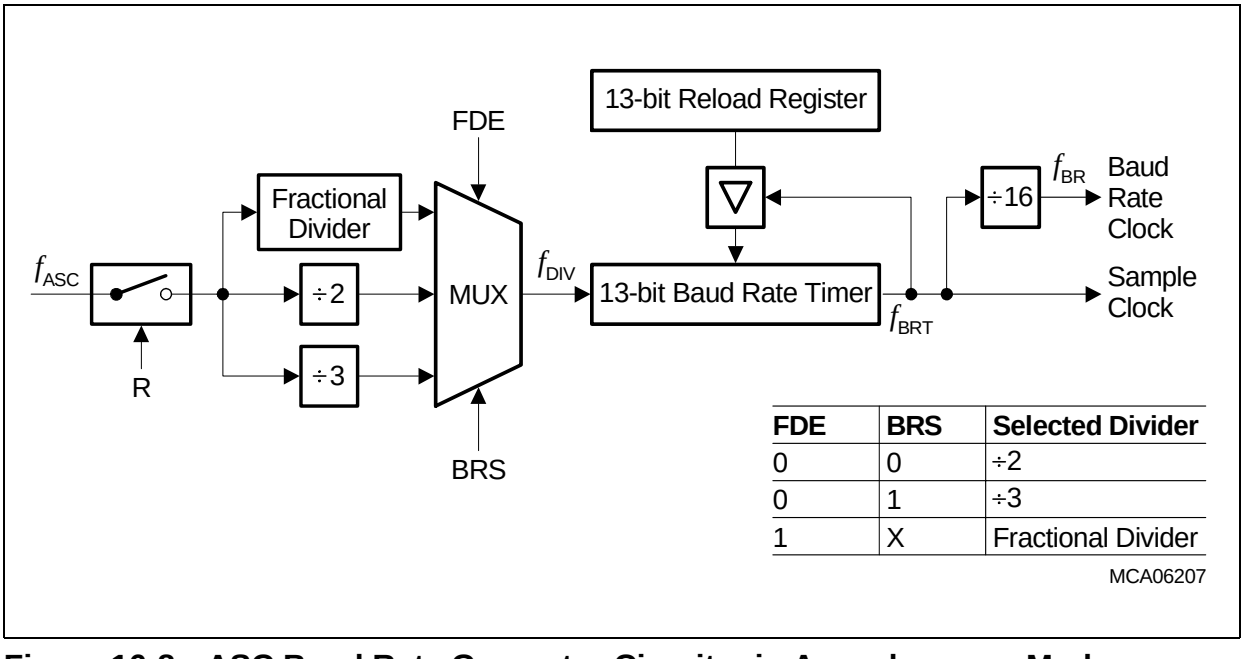
16.1.5.1 Baud Rates in Asynchronous Mode

For asynchronous operation, the baud rate generator provides a clock f_{BRT} with sixteen times the rate of the established baud rate. Every received bit is sampled on the 7th, 8th and 9th cycle of this clock. The clock divider circuitry, which generates the input clock f_{DIV} for the 13-bit baud rate timer, is extended by a fractional divider circuitry that allows the adjustment of more accurate baud rates and the extension of the baud rate range.

The baud rate of the baud rate generator depends on the settings of the following bits and register values:

- Input clock f_{ASC}
- Selection of the baud rate timer input clock f_{DIV} by bits CON.FDE and CON.BRS
- If bit CON.FDE = 1 (fractional divider): value of register FDV
- Value of the 13-bit reload register BG

The output clock of the baud rate timer with the reload register is the sample clock in the asynchronous operating modes of the ASC. For baud rate calculations, this baud rate clock f_{BR} is derived from the sample clock f_{BRT} by a division of sixteen.



FDE	BRS	Selected Divider
0	0	÷2
0	1	÷3
1	X	Fractional Divider

Figure 16-8 ASC Baud Rate Generator Circuitry in Asynchronous Modes

Using the Fixed Input Clock Divider

The baud rate for asynchronous operation of the serial channel ASC when using the fixed input clock divider ratios (CON.FDE = 0) and the required BG reload value for a given baud rate can be determined by the following formulas:

Table 16-1 Asynchronous Baud Rate Formulas using the Fixed Input Clock Dividers

FDE	BRS	BG	Formula
0	0	0 … 8191	$\text{Baud rate} = \frac{f_{ASC}}{32 \times (BG + 1)}$ $BG = \frac{f_{ASC}}{32 \times \text{Baud rate}} - 1$
	1		$\text{Baud rate} = \frac{f_{ASC}}{48 \times (BG + 1)}$ $BG = \frac{f_{ASC}}{48 \times \text{Baud rate}} - 1$

BG represents the content of the reload register bit field BG.BR_VALUE, taken as an unsigned 13-bit integer.

The maximum baud rate that can be achieved for the asynchronous operating modes when using the two fixed clock dividers and a module clock of 90 MHz is 2.8125 Mbit/s. **Table 16-2** lists various commonly used baud rates together with the required reload values and the deviation errors compared to the intended baud rate.

Table 16-2 Typical Asynchronous Baud Rates using Fixed Input Clock Dividers

Baud Rate	CON.BRS = 0, f_{ASC} = 90 MHz		CON.BRS = 1, f_{ASC} = 90 MHz	
	Deviation Error	Reload Value	Deviation Error	Reload Value
2.8125 Mbit/s	–	0000_H	–	–
19.2 kbit/s	+0.3% / -0.4%	0091_H / 0092_H	+0.9% / -0.4%	0060_H / 0061_H
9600 bit/s	+0.0% / -0.3%	0123_H / 0124_H	+0.2% / -0.4%	$00C2_H$ / $00C3_H$
4800 bit/s	+0.2% / -0.0%	0248_H / 0249_H	+0.2% / -0.1%	0185_H / 0186_H

Note: CON.FDE must be 0 to achieve the baud rates in the table above. The deviation errors given in the table are rounded. Using a baud rate crystal will provide correct baud rates without deviation errors.

Using the Fractional Divider

When the fractional divider is selected, the input clock f_{DIV} for the baud rate timer is derived from the module clock f_{ASC} by a programmable fractional divider. If CON.FDE = 1, the fractional divider is activated. It divides f_{ASC} by a fraction of n/512 for any value of n from 0 to 511. If n = 0, the divider ratio is 1, which means that $f_{DIV} = f_{ASC}$. In general, the fractional divider allows the baud rate to be programmed with much better accuracy than with the two fixed prescaler divider stages.

Note: In fractional divider mode, the clock f_{DIV} can have a maximum period jitter of one f_{ASC} clock period.

Table 16-3 Asynchronous Baud Rate Formulas using the Fractional Input Clock Divider

FDE	BRS	BG	FDV	Formula
1	–	0 … 8191	1 … 511	$\text{Baud rate} = \frac{FDV}{512} \times \frac{f_{ASC}}{16 \times (BG + 1)}$
			0	$\text{Baud rate} = \frac{f_{ASC}}{16 \times (BG + 1)}$

BG represents the content of the reload register bit field BG.BR_VALUE, taken as an unsigned 13-bit integer. FDV represents the contents of the fractional divider register bit field FDV.FD_VALUE, taken as an unsigned 9-bit integer.

Table 16-4 Typical Asynchronous Baud Rates using the Fractional Input Clock Divider

f_{ASC}	Desired Baud Rate	BG	FDV	Resulting Baud Rate	Deviation
90 MHz	115.2 kbit/s	0022_H	$16F_H$	115.199 kbit/s	0%
	57.6 kbit/s	0045_H	$16F_H$	57.6 kbit/s	0%
	38.4 kbit/s	0068_H	$16F_H$	38.4 kbit/s	0%
	19.2 kbit/s	$006A_H$	$0BB_H$	57.6 kbit/s	0%

16.1.5.2 Baud Rates in Synchronous Mode

For synchronous operation, the baud rate generator provides a clock f_{BRT} that runs with four times the established baud rate (see **Figure 16-9**).

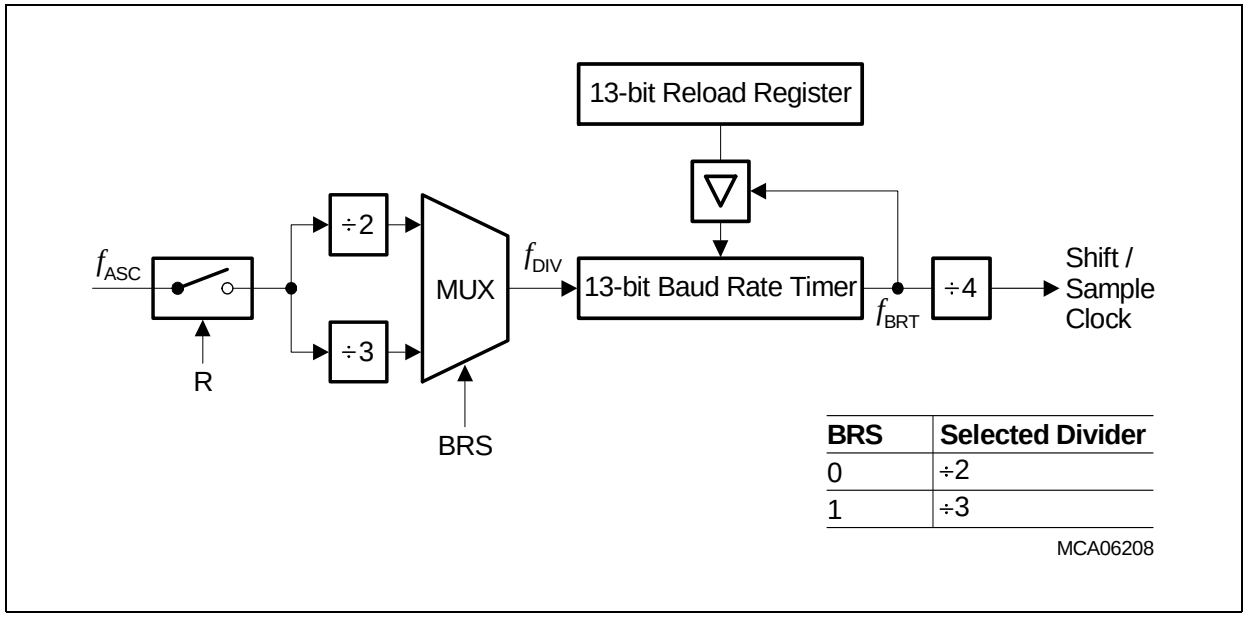


Figure 16-9 ASC Baud Rate Generator Circuitry in Synchronous Mode

The baud rate for synchronous operation of the serial channel ASC can be determined by the formulas as shown in **Table 16-5**.

Table 16-5 Synchronous Baud Rate Formulas

BRS	BG	Formula
0	0 … 8191	$\text{Baud rate} = \frac{f_{ASC}}{8 \times (BG + 1)}$ $\quad BG = \frac{f_{ASC}}{8 \times \text{Baud rate}} - 1$
1		$\text{Baud rate} = \frac{f_{ASC}}{12 \times (BG + 1)}$ $\quad BG = \frac{f_{ASC}}{12 \times \text{Baud rate}} - 1$

BG represents the content of the reload register bit field BG.BR_VALUE, taken as an unsigned 13-bit integer.

The maximum baud rate that can be achieved in Synchronous Mode when using a module clock of 90 MHz is 11.25 Mbit/s.

16.1.6 Hardware Error Detection Capabilities

To improve the reliability of serial data exchange, the serial channel ASC provides an error interrupt request flag that indicates the presence of an error and three (selectable) error status flags in register CON that indicate which error has been detected during reception. Upon completion of a reception, the error interrupt request line EIR will be activated simultaneously with the receive interrupt request line RIR, if one or more of the following conditions are met:

- If the framing error detection enable bit CON.FEN is set and any of the expected stop bits is not high, the framing error flag CON.FE is set, indicating that the error interrupt request is due to a framing error (asynchronous operating modes only).
- If the parity error detection enable bit CON.PEN is set in the modes where a parity bit is received and the parity check on the received data bits proves false, the parity error flag CON.PE is set, indicating that the error interrupt request is due to a parity error (asynchronous operating modes only).
- If the overrun error detection enable bit CON.OEN is set and the last character received was not read out of the receive buffer by software or DMA transfer at the time the reception of a new frame is complete, the overrun error flag CON.OE is set indicating that the error interrupt request is due to an overrun error (Asynchronous and Synchronous Modes).

16.1.7 Interrupts

Four interrupt sources are provided for serial channel ASC. Line TIR indicates a transmit interrupt, TBIR indicates a transmit buffer interrupt, RIR indicates a receive interrupt, and EIR indicates an error interrupt of the serial channel. The interrupt output lines TBIR, TIR, RIR, and EIR are activated (active state) for two periods of the module clock f_{ASC}.

The cause of an error interrupt request EIR (framing, parity, overrun error) can be identified by the error status flags CON.FE, CON.PE, and CON.OE.

Note: By contrast to the error interrupt request line EIR, the error status flags CON.FE/CON.PE/CON.OE are not reset automatically but must be cleared by software.

For normal operation (that is, other than error interrupt), the ASC provides three interrupt requests to control data exchange via this serial channel:

- TBIR is activated when data is moved from TBUF to the transmit shift register.
- TIR is activated before the last bit of an asynchronous frame is transmitted, or after the last bit of a synchronous frame has been transmitted.
- RIR is activated when the received frame is moved to RBUF.

While the task of the receive interrupt handler is quite clear, the transmitter is serviced by two interrupt handlers. This provides advantages for the servicing software.

For single transfers, it is sufficient to use the transmitter interrupt (TIR), which indicates that the previously loaded data has been transmitted, except for the last bit of an asynchronous frame.

For multiple back-to-back transfers, it is necessary to load the following piece of data at least before the last bit of the previous frame has been transmitted. In Asynchronous Mode, this leaves just one bit-time for the handler to respond to the transmitter interrupt request; in Synchronous Mode, it is entirely impossible.

Using the Transmit Buffer Interrupt (TBIR) to reload transmit data provides the time necessary to transmit a complete frame for the service routine, as TBUF may be reloaded while the previous data is still being transmitted.

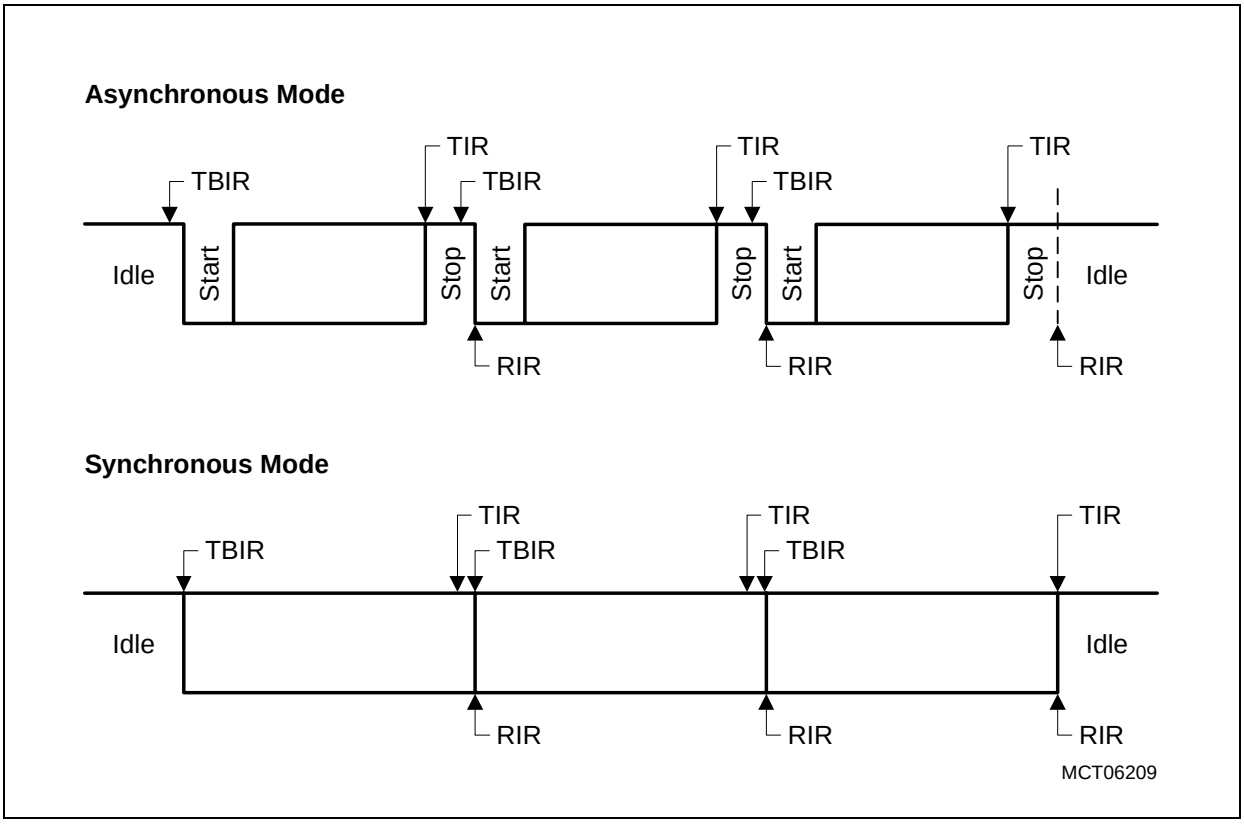


Figure 16-10 ASC Interrupt Generation

As shown in **Figure 16-10**, TBIR is an early trigger for the reload routine, while TIR indicates the completed transmission. Software using handshake should, therefore, rely on TIR at the end of a data block to ensure that all data has been transmitted.

16.2 ASC Kernel Registers

This section describes the kernel registers of the ASC module. All ASC kernel register names described in this section will be referenced in other parts of the TC1797 User´s Manual by the module name prefix “ASC0_” for the ASC0 interface and “ASC1_” for the ASC1 interface.

All registers in the ASC address spaces are reset with the application reset.

ASC Kernel Register Overview

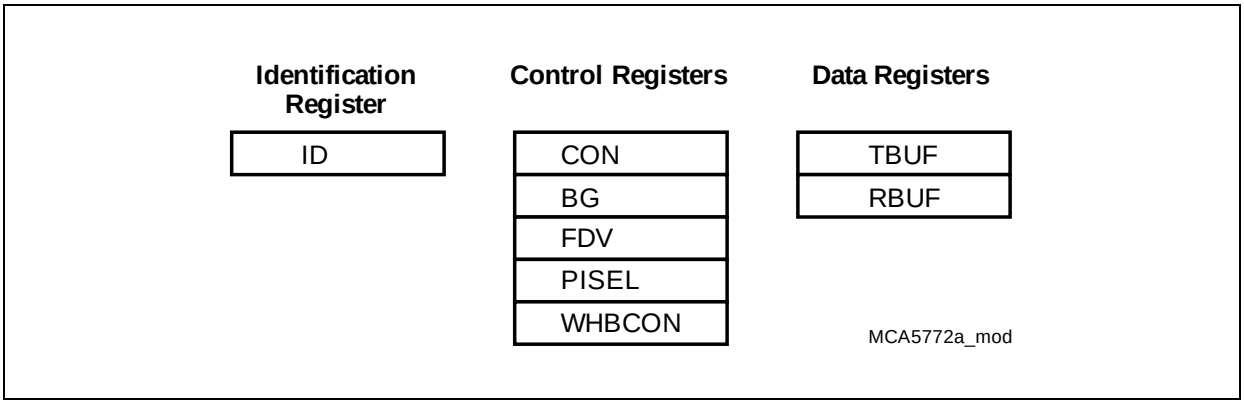


Figure 16-11 ASC Kernel Registers

The complete and detailed address map of the of the ASC module and its registers is described in **Table 16-10** on **Page 16-42**.

Table 16-6 Registers Address Space

Module	Base Address	End Address	Note
ASC0	F000 0A00$_H$	F000 0AFF$_H$	–
ASC1	F000 0B00$_H$	F000 0BFF$_H$	–

Table 16-7 Registers Overview - ASC Kernel Registers

Register Short Name	Register Long Name	Offset Address[1)]	Description see
PISEL	Peripheral Input Select Register	0004$_H$	**Page 16-20**
ID	Module Identification Register	0008$_H$	**Page 16-21**
CON	Control Register	0010$_H$	**Page 16-22**
BG	Baud Rate Timer Reload Register	0014$_H$	**Page 16-27**
FDV	Fractional Divider Register	0018$_H$	**Page 16-27**
TBUF	Transmit Buffer Register	0020$_H$	**Page 16-28**

Table 16-7 Registers Overview - ASC Kernel Registers (cont'd)

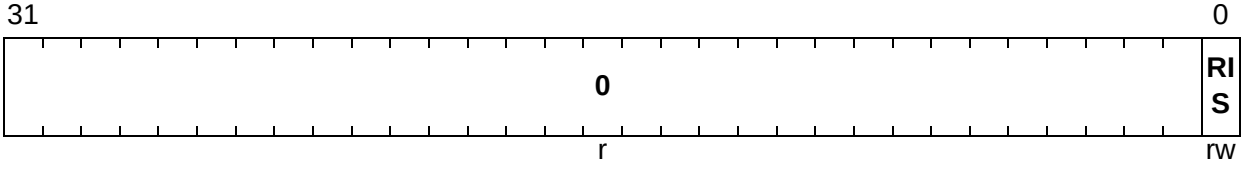
Register Short Name	Register Long Name	Offset Address[1)]	Description see
RBUF	Receive Buffer Register	0024_H	Page 16-29
WHBCON	Write Hardware Bits Control Register	0050_H	Page 16-25

1) The absolute register address is calculated as follows:
Module Base Address ((Table 16-6 on Page 16-19) + Offset Address (shown in this column)

16.2.1 Control Registers

The ASC module kernel provides two receive input lines, RXD_I0 and RXD_I1. Bit RIS in the Peripheral Input Select Register PISEL determines which of theses two input lines is taken for RXD receive input purposes.

PISEL
Peripheral Input Select Register (04_H) Reset Value: 0000 0000$_H$

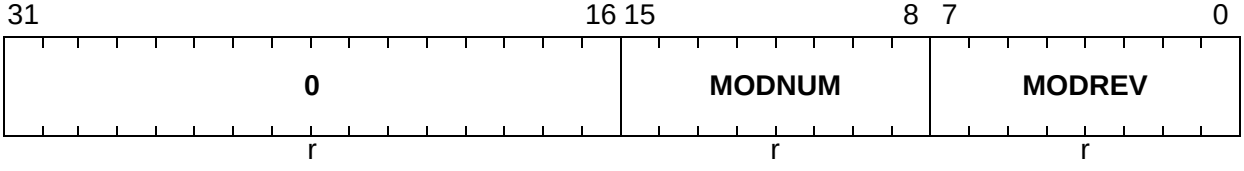


Field	Bits	Type	Description
RIS	0	rw	**Receive Input Select** 0_B ASC receiver input RXD_I0 selected 1_B ASC receiver input RXD_I1 selected
0	[31:1]	r	**Reserved** Read as 0; should be written with 0.

*Note: Implementation specific details (RIS functionality) see **"Peripheral Input Select Register" on Page 16-35**.*

The ASC Module Identification Register ID contains read-only information about the module version.

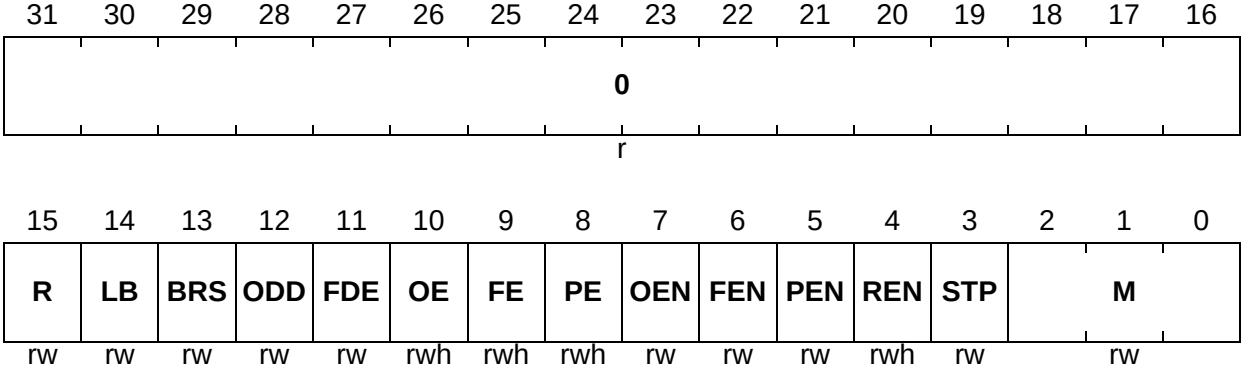
ID
Module Identification Register (08$_H$) Reset Value: 0000 44XX$_H$



Field	Bits	Type	Description
MODREV	[7:0]	r	**Module Revision Number** MOD_REV defines the module revision number. The value of a module revision starts with 01$_H$ (first revision).
MODNUM	[15:8]	r	**Module Number Value** This bit field defines the module identification number for the ASC: 44$_H$
0	[31:16]	r	**Reserved** Read as 0.

The serial operating modes of the ASC module are controlled by its Control Register CON. This register contains control bits for mode and error check selection, and status flags for error identification.

CON
Control Register **(10_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
M	[2:0]	rw	**Mode Selection** 000_B 8-bit data Synchronous Mode 001_B 8-bit data Asynchronous Mode 010_B Reserved. Do not use this combination. 011_B 7-bit data + parity Asynchronous Mode 100_B 9-bit data Asynchronous Mode 101_B 8-bit data + wake up bit Asynchronous Mode 110_B Reserved. Do not use this combination. 111_B 8-bit data + parity Asynchronous Mode
STP	3	rw	**Number of Stop Bit Selection** 0_B One stop bit 1_B Two stop bits
REN	4	rwh	**Receiver Enable Control** 0_B Receiver disabled 1_B Receiver enabled Bit is reset by hardware after reception of a byte in Synchronous Mode.
PEN	5	rw	**Parity Check Enable (asynchronous mode only)** 0_B Ignore parity 1_B Check parity

Field	Bits	Type	Description
FEN	6	rw	**Framing Check Enable (asynchronous mode only)** 0_B Ignore framing errors 1_B Check framing errors
OEN	7	rw	**Overrun Check Enable** 0_B Ignore overrun errors 1_B Check overrun errors
PE	8	rwh	**Parity Error Flag** Set by hardware on a parity error (PEN = 1). Must be reset by software.
FE	9	rwh	**Framing Error Flag** Set by hardware on a framing error (FEN = 1). Must be reset by software.
OE	10	rwh	**Overrun Error Flag** Set by hardware on an overrun error (OEN = 1). Must be reset by software.
FDE	11	rw	**Fractional Divider Enable** 0_B Fractional divider disabled 1_B Fractional divider is enabled and used as prescaler for baud rate timer (bit BRS is don't care)
ODD	12	rw	**Parity Selection** 0_B Even parity selected (parity bit = 1 on odd number of 1s in data, parity bit = 0 on even number of 1s in data) 1_B Odd parity selected (parity bit = 1 on even number of 1s in data, parity bit = 0 on odd number of 1s in data)
BRS	13	rw	**Baud Rate Selection** 0_B Baud rate timer prescaler divide-by-2 selected 1_B Baud rate timer prescaler divide-by-3 selected BRS is don't care if FDE = 1 (fractional divider enabled)
LB	14	rw	**Loop-back Mode Enable** 0_B Loop-Back mode disabled 1_B Loop-Back mode enabled

Field	Bits	Type	Description
R	15	rw	**Baud Rate Generator Run Control** 0_B Baud rate generator disabled (ASC inactive) 1_B Baud rate generator enabled Register BG should only be written if R = 0.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

Serial data transmission or reception is possible only when the run bit CON.R is set to 1. Otherwise, the serial interface is idle. To avoid unpredictable behavior of the serial interface, do not program the mode control field CON.M to one of the reserved combinations.

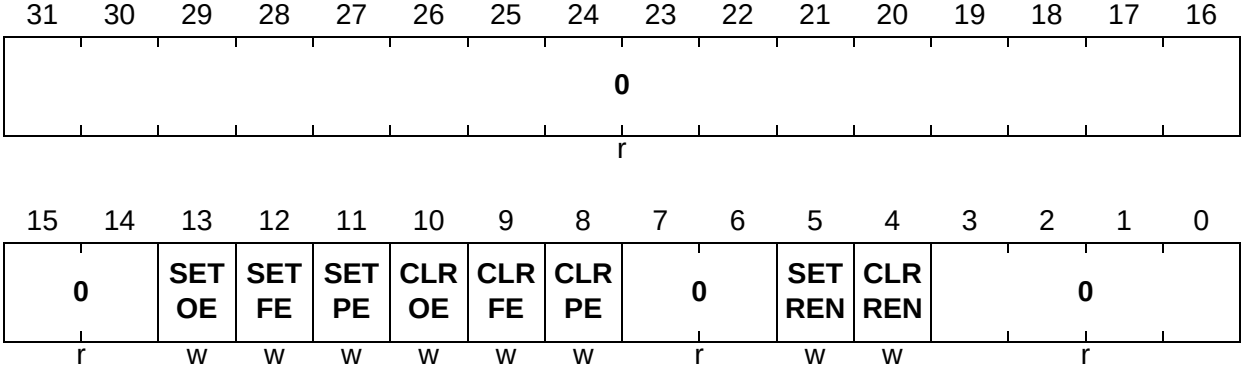
Critical “rwh” Bits

Register CON contains three error flags: PE, FE, and OE. If the software modifies only one of these error flags, it uses typically a Read-Modify-Write (RMW) instruction. When one of the other error flags that is not intended to be modified by the RMW instruction is changed by hardware after the read access but before the write back access of the RMW instruction, it is overwritten with the old bit value, and the hardware change of the bit gets lost. This problem does not affect the bits that are intended to be modified by the RMW instruction. It only affects bits that were not intended to be changed with the RMW instruction.

The three error flags in register CON and the REN bit can be additionally set or reset by software via register WHBCON. This capability avoids the problem with the CON register RMW instruction access to the error flags. WHBCON is a write-only register. Reading WHBCON always returns 0000 0000_H.

WHBCON

Write Hardware Bits Control Register (50_H) **Reset Value: $0000\ 0000_H$**



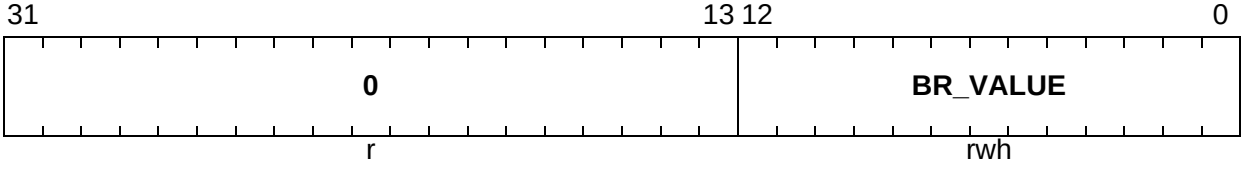
Field	Bits	Type	Description
CLRREN	4	w	**Clear Receiver Enable Bit** 0_B No effect 1_B Bit CON.REN is cleared. Bit is always read as 0.
SETREN	5	w	**Set Receiver Enable Bit** 0_B No effect 1_B Bit CON.REN is set. Bit is always read as 0.
CLRPE	8	w	**Clear Parity Error Flag** 0_B No effect 1_B Bit CON.PE is cleared. Bit is always read as 0.
CLRFE	9	w	**Clear Framing Error Flag** 0_B No effect 1_B Bit CON.FE is cleared. Bit is always read as 0.
CLROE	10	w	**Clear Overrun Error Flag** 0_B No effect 1_B Bit CON.OE is cleared. Bit is always read as 0.
SETPE	11	w	**Set Parity Error Flag** 0_B No effect 1_B Bit CON.PE is set. Bit is always read as 0.

Field	Bits	Type	Description
SETFE	12	w	**Set Framing Error Flag** 0_B No effect 1_B Bit CON.FE is set. Bit is always read as 0.
SETOE	13	w	**Set Overrun Error Flag** 0_B No effect 1_B Bit CON.OE is set. Bit is always read as 0.
0	[3:0], [7:6], [31:14]	r	**Reserved** Read as 0; should be written with 0.

Note: When the set and clear bits for an error flag are set at the same time during a WHBCON write operation (e.g SETPE = CLRPE = 1), the error flag in CON is not affected.

The Baud Rate Timer Reload Register BG of the ASC module contains the 13-bit reload value for the baud rate timer in Asynchronous and Synchronous Modes.

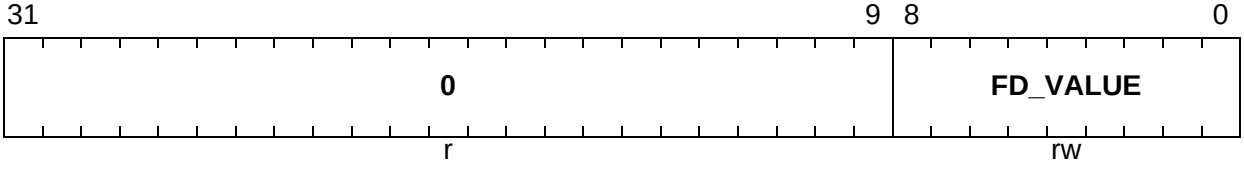
BG
Baud Rate Timer/Reload Register (14$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
BR_VALUE	[12:0]	rwh	**Baud Rate Timer/Reload Register Value** Reading BR_VALUE returns the 13-bit content of the baud rate timer. Writing BR_VALUE loads the baud rate timer reload register. BG should only be written if CON.R = 0.
0	[31:13]	r	**Reserved** Read as 0; should be written with 0.

The Fractional Divider Register FDV of the ASC module contains the 9-bit divider value for the fractional divider (asynchronous mode only).

FDV
Fractional Divider Register (18$_H$) Reset Value: 0000 0000$_H$

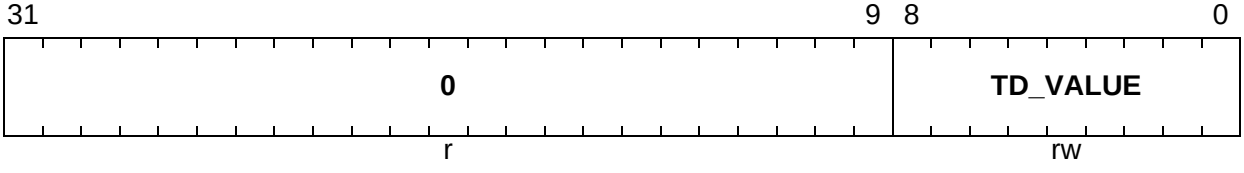


Field	Bits	Type	Description
FD_VALUE	[8:0]	rw	**Fractional Divider Register Value** FD_VALUE contains the 9-bit value n of the fractional divider which determines the fractional divider ratio n/512 (n = 0-511). With n = 0, the fractional divider is switched off (divider ratio = 1).
0	[31:9]	r	**Reserved** Read as 0; should be written with 0.

16.2.2 Data Registers

The Transmit Buffer Register TBUF of the ASC module contains the transmit data value in Asynchronous And Synchronous Modes.

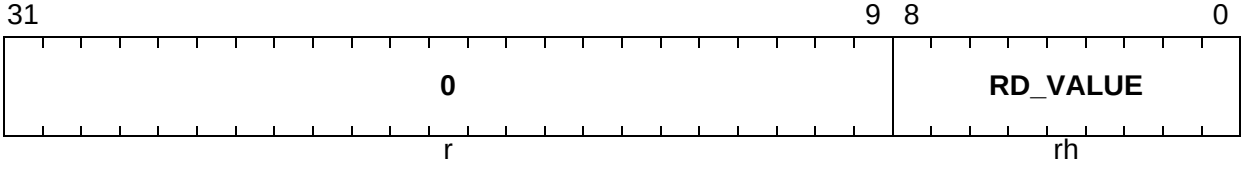
TBUF
Transmit Buffer Register **(20_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
TD_VALUE	[8:0]	rw	**Transmit Data Register Value** TBUF contains the data to be transmitted in the asynchronous and synchronous operating modes of the ASC. Data transmission is double-buffered; therefore, a new value can be written to TBUF before the transmission of the previous value is complete.
0	[31:9]	r	**Reserved** Read as 0; should be written with 0.

The receive buffer register RBUF of the ASC module contains the receive data value in Asynchronous and Synchronous Modes.

RBUF
Receive Buffer Register **(24_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
RD_VALUE	[8:0]	rh	**Receive Data Register Value** RBUF contains the received data bits and, depending on the selected mode, the parity bit in the asynchronous and synchronous operating modes of the ASC. In Asynchronous Mode, with CON.M = 011_B (7-bit data + parity), the received parity bit is written into RBUF.7. In Asynchronous Mode, with CON.M = 111_B (8-bit data + parity), the received parity bit is written into RBUF.8.
0	[31:9]	r	**Reserved** Read as 0.

16.3 ASC0/ASC1 Module Implementation

This section describes ASC0/ASC1 module interfaces with the clock control, port connections, interrupt control, and address decoding.

16.3.1 Interfaces of the ASC Modules

The serial I/O lines of both modules are connected either to Port 5 or Port 6. Each of the ASC modules is further supplied with interrupt control, address decoding, and port control logic. Two DMA requests can be generated by each ASC module. Both ASC modules are supplied by one common clock control unit.

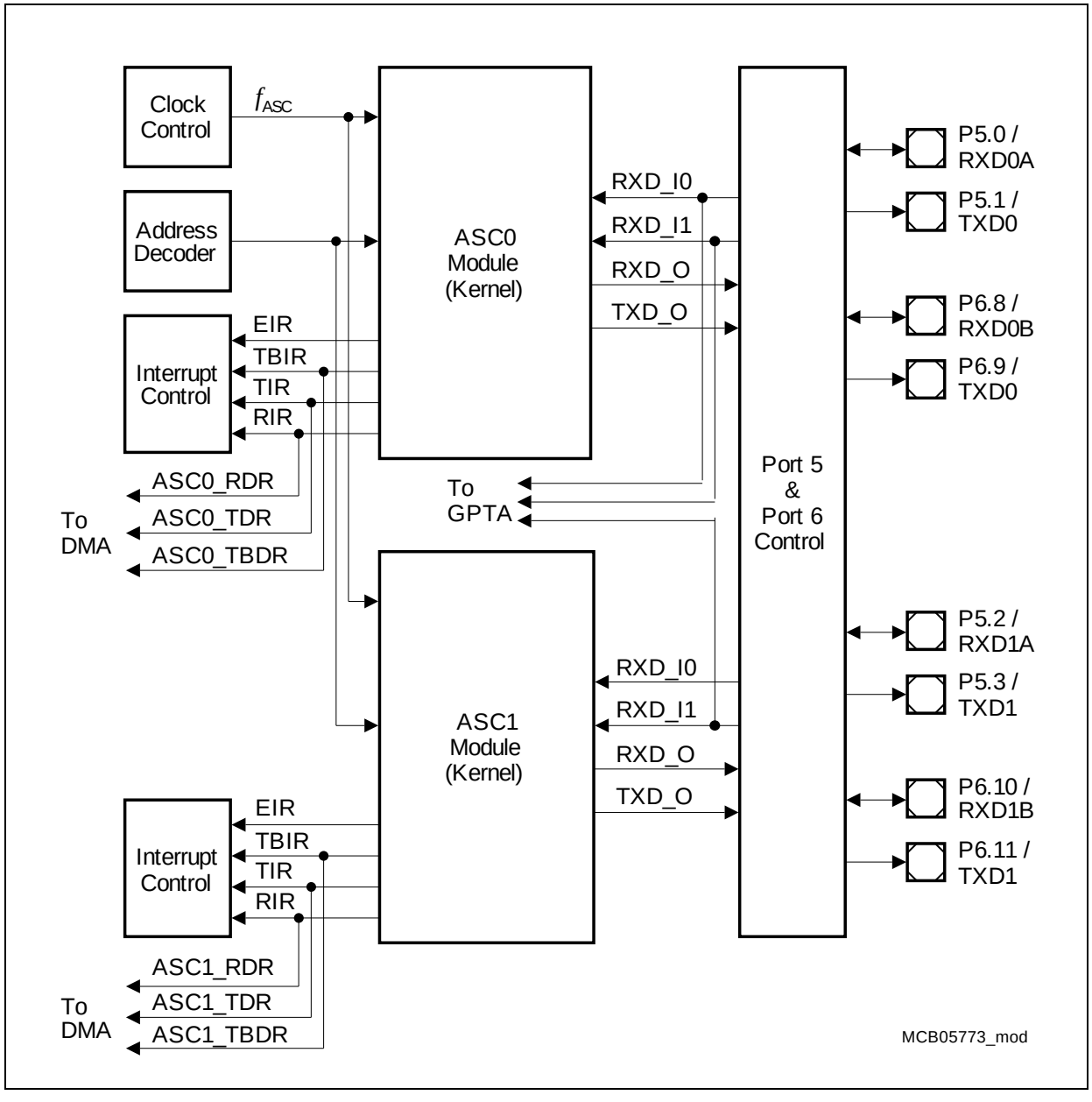


Figure 16-12 ASC0/ASC1 Module Implementation and Interconnections

Some of the receive inputs of the ASC0 and ASC1 are connected via a multiplexer to an LTC input of the GPTA module. Details are described in the SCU and the GPTA chapters.

16.3.2 ASC0/ASC1 Module Related External Registers

Figure 16-13 summarizes the module-related external registers which are required for ASC0/ASC1 programming (see also **Figure 16-11** for the module kernel-specific registers).

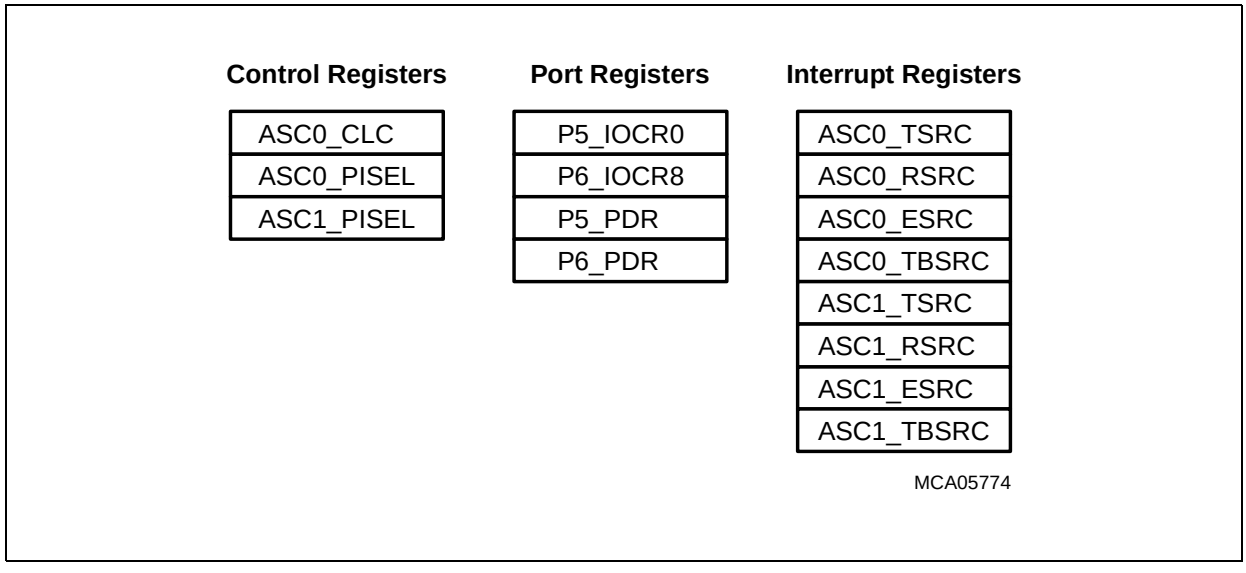
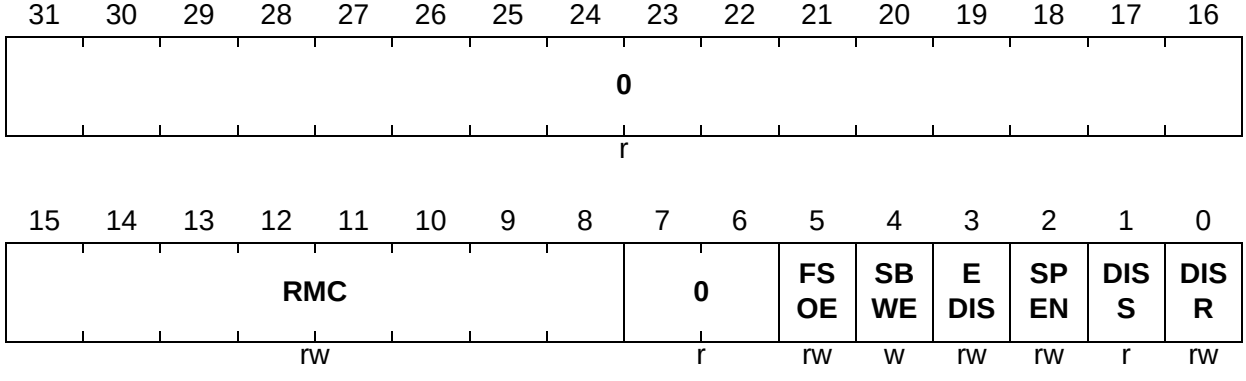


Figure 16-13 ASC0/ASC1 Implementation-specific Special Function Registers

16.3.2.1 Clock Control Register

The Clock Control Register ASC0_CLC allows the programmer to adapt the functionality and power consumption of the ASC modules to the requirements of the application. The description below shows the clock control register functionality which is implemented for the ASC modules. Because ASC0 and ASC1 share one common clock control interface, ASC0_CLC controls the f_{ASC} module clock signal, sleep mode, suspend mode and fast shut-off mode for both modules.

ASC0_CLC
ASC0 Clock Control Register (00_H) Reset Value: 0000 0003$_H$



Field	Bits	Type	Description
DISR	0	rw	**Module Disable Request Bit** Used for enable/disable control of the module.
DISS	1	r	**Module Disable Status Bit** Bit indicates the current status of the module.
SPEN	2	rw	**Module Suspend Enable for OCDS** Used to enable the suspend mode.
EDIS	3	rw	**Sleep Mode Enable Control** Used to control module's sleep mode.
SBWE	4	w	**Module Suspend Bit Write Enable for OCDS** Determines whether SPEN and FSOE are write-protected.
FSOE	5	rw	**Fast Switch Off Enable** Used to switch off fast clock in Suspend Mode.
RMC	[15:8]	rw	**8-bit Clock Divider Value in RUN Mode**
0	[7:6], [31:16]	r	**Reserved** Read as 0; should be written with 0.

Note: After a hardware reset operation, the two ASC modules are disabled.

Note: The number of module clock cycles (wait states) which are required for a “destructive read” access (means: flags/bits are set/reset by one read access) to ASC module register depends on the selected CLC clock frequency, which is selected via bit field RMC in the CLC register. Therefore, increasing ASC0_CLC.RMC may result in a longer FPI Bus read cycle access time.

16.3.2.2 Peripheral Input Select Register

The ASC0/ASC1 modules include a peripheral input select registers that are used to switch the RXD input lines of the ASC0/ASC1 module kernels between different pairs of pins of Port 5 and Port 6 as shown in **Figure 16-14**. Register ASC0_PISEL controls the RXD input selection for ASC0, and ASC1_PISEL controls the RXD input selection for ASC1.

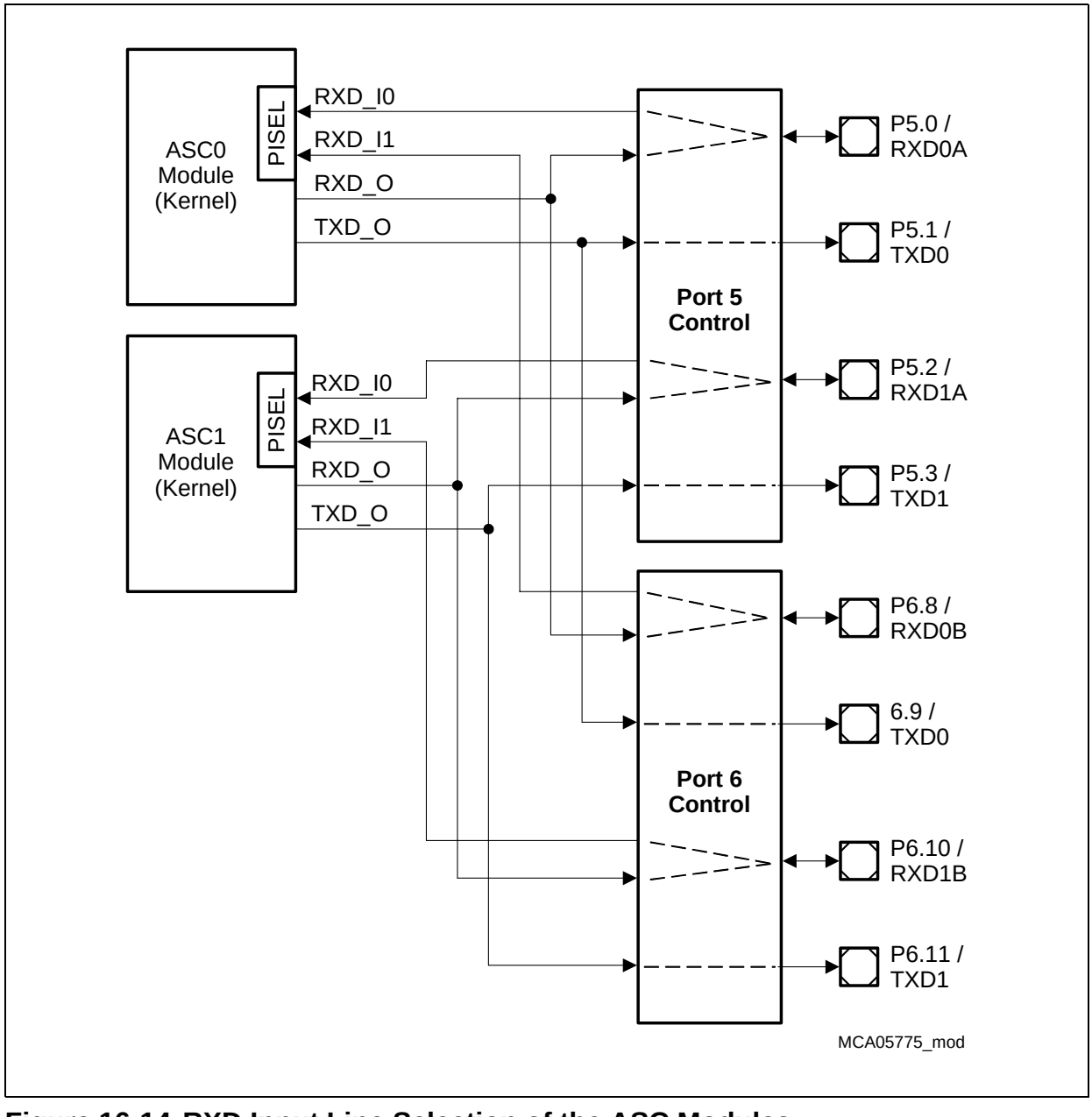
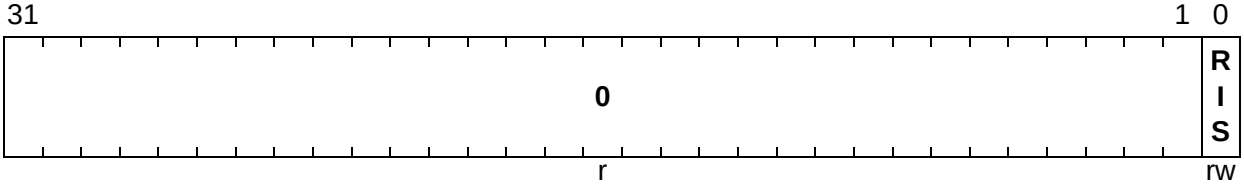


Figure 16-14 RXD Input Line Selection of the ASC Modules

ASC0_PISEL
ASC0 Peripheral Input Select Register

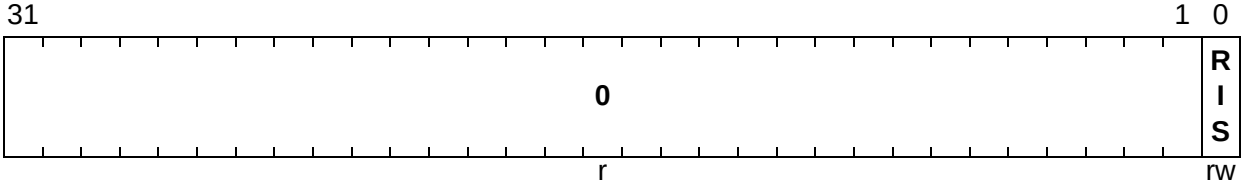
(04_H) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
RIS	0	rw	**Receive Input Select** 0_B ASC0 receiver input RXD0A (P5.0) selected 1_B ASC0 receiver input RXD0B (P6.8) selected
0	[31:1]	0	**Reserved** Read as 0; should be written with 0.

ASC1_PISEL
ASC1 Peripheral Input Select Register

(04_H) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
RIS	0	rw	**Receive Input Select** 0_B ASC1 receiver input RXD1A (P5.2) selected 1_B ASC1 receiver input RXD1B (P6.10) selected
0	[31:1]	0	**Reserved** Read as 0; should be written with 0.

16.3.2.3 Port Control Registers

As shown in **Figure 16-14**, the I/O lines of the ASC modules are connected to Class A2 port pins of Port 3. Additionally to the PISEL register programming, the required ASC port lines must be programmed by software for the desired ASC input/output functionality. Two selections must be executed:

- Input/output function selection
  (controlled by the port input/output control registers IOCR)
- Pad driver characteristics selection for the outputs
  (controlled by the port pad driver mode register PDR)

Input/Output Function Selection

The port input/output control registers contain the 4-bit wide bit fields that select the digital output and input driver characteristics such as pull-up/down devices, port direction (input/output), open-drain, and alternate output selections individually for each pin. The I/O lines for the ASC modules are controlled by the port input/output control registers P5_IOCR0 and P6_IOCR8.

Table 16-8 shows how bits and bit fields must be programmed for the required I/O functionality of the ASC I/O lines. This table also shows the values of the peripheral input select registers.

Table 16-8 ASC0/ASC1 I/O Control Selection and Setup

Module	Port Lines	PISEL Register	Input/Output Control Register Bits	I/O
ASC0	P5.0/RXD0A	ASC0_PISEL.RIS = 0	P5_IOCR0.PC0 = $0XXX_B$	Input
		–	P5_IOCR0.PC0 = $1X01_B$	Output[1)]
	P6.8/RXD0B	ASC0_PISEL.RIS = 1	P6_IOCR8.PC8 = $0XXX_B$	Input
		–	P6_IOCR8.PC8 = $1X10_B$	Output[1)]
	P5.1/TXD0	–	P5_IOCR0.PC1 = $1X01_B$	Output
	P6.9/TXD0		P6_IOCR8.PC9 = $1X10_B$	Output
ASC1	P5.2/RXD1A	ASC1_PISEL.RIS = 0	P5_IOCR0.PC2 = $0XXX_B$	Input
		–	P5_IOCR0.PC2 = $1X01_B$	Output[1)]
	P6.10/RXD1B	ASC1_PISEL.RIS = 1	P6_IOCR8.PC10 = $0XXX_B$	Input
		–	P6_IOCR8.PC10 = $1X10_B$	Output[1)]
	P5.3/TXD1	–	P5_IOCR0.PC3 = $1X01_B$	Output
	P6.11/TXD1		P6_IOCR8.PC11 = $1X10_B$	Output

1) Applicable in Synchronous Mode only.

*Note: In synchronous operating mode of the ASC, the type of the selected RXD port pin (input or output) is **not** automatically controlled by the ASC but must be defined by a user program by writing the appropriate bit field in the IOCR registers.*

16.3.2.4 Interrupt Control Registers

The eight interrupts of the ASC0 and ASC1 modules are controlled by the following service request control registers:

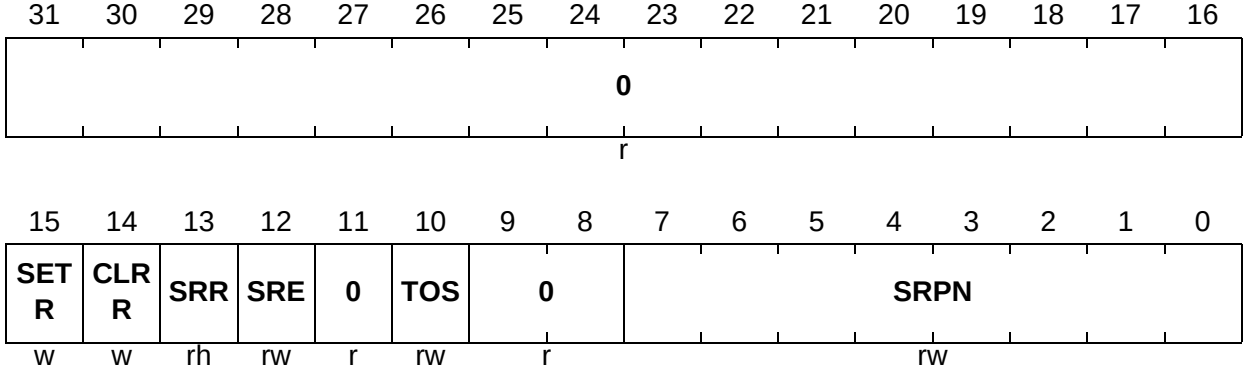
- ASC0_TSRC, ASC1_TSRC: control the transmit interrupts
- ASC0_RSRC, ASC1_RSRC: control the receive interrupts
- ASC0_ESRC, ASC1_ESRC: control the error interrupts
- ASC0_TBSRC, ASC1_TBSRC: control the transmit buffer empty interrupts

TSRC
Transmit Interrupt Service Request Control Register
($F0_H$) **Reset Value: 0000 0000$_H$**

RSRC
Receive Interrupt Service Request Control Register
($F4_H$) **Reset Value: 0000 0000$_H$**

ESRC
Error Interrupt Service Request Control Register
($F8_H$) **Reset Value: 0000 0000$_H$**

TBSRC
Transmit Buffer Interrupt Service Request Control Register
(FC_H) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
SRPN	[7:0]	rw	**Service Request Priority Number**
TOS	10	rw	**Type of Service Control**
SRE	12	rw	**Service Request Enable**
SRR	13	rh	**Service Request Flag**
CLRR	14	w	**Request Clear Bit**
SETR	15	w	**Request Set Bit**

Field	Bits	Type	Description
0	[9:8], 11, [31:16]	r	**Reserved** Read as 0; should be written with 0.

16.3.2.5 DMA Requests

The DMA request output lines of the ASC0/ASC1 modules become active whenever the related ASC interrupt line becomes activated. The DMA request lines are connected to the DMA controller as shown in **Table 16-9**.

Table 16-9 DMA Request Lines of ASC0/ASC1

Module	Related ASC Interrupt	DMA Request Line	Description
ASC0	RIR	ASC0_RDR	DMA Channel 00 Request Input 5
			DMA Channel 10 Request Input 5
			DMA Channel 06 Request Input 5
			DMA Channel 16 Request Input 5
	TIR	ASC0_TDR	DMA Channel 02 Request Input 5
			DMA Channel 12 Request Input 5
			DMA Channel 04 Request Input 5
			DMA Channel 14 Request Input 5
	TBIR	ASC0_TBDR	DMA Channel 02 Request Input 12
			DMA Channel 12 Request Input 12
			DMA Channel 04 Request Input 12
			DMA Channel 14 Request Input 12

Table 16-9 DMA Request Lines of ASC0/ASC1 (cont'd)

Module	Related ASC Interrupt	DMA Request Line	Description
ASC1	RIR	ASC1_RDR	DMA Channel 01 Request Input 5
			DMA Channel 11 Request Input 5
			DMA Channel 07 Request Input 5
			DMA Channel 17 Request Input 5
	TIR	ASC1_TDR	DMA Channel 03 Request Input 5
			DMA Channel 13 Request Input 5
			DMA Channel 05 Request Input 5
			DMA Channel 15 Request Input 5
	TBIR	ASC1_TBDR	DMA Channel 03 Request Input 12
			DMA Channel 13 Request Input 12
			DMA Channel 05 Request Input 12
			DMA Channel 15 Request Input 12

Note: Further details on DMA handling and processing are described in the chapter "DMA Controller" of the TC1797 System Units User´s Manual.

16.3.3 Address Map

An absolute register address is given by the offset address of the register (given in **Table 16-7**) plus the module base address (given in **Table 16-6**).

Table 16-10 Address Map of ASC0/ASC1

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
Async./Sync. Serial Interface 0 (ASC0)					
ASC0_ CLC	ASC0 Clock Control Register	F000 0A00$_H$	U, SV	SV, E	0000 0003$_H$
ASC0_ PISEL	ASC0 Peripheral Input Select Register	F000 0A04$_H$	U, SV	U, SV	0000 0000$_H$
ASC0_ ID	ASC0 Module Identification Register	F000 0A08$_H$	U, SV	BE	0000 44XX$_H$
–	Reserved	F000 0A0C$_H$	BE	BE	–
ASC0_ CON	ASC0 Control Register	F000 0A10$_H$	U, SV	U, SV	0000 0000$_H$
ASC0_ BG	ASC0 Baud Rate/Timer Reload Register	F000 0A14$_H$	U, SV	U, SV	0000 0000$_H$
ASC0_ FDV	ASC0 Fractional Divider Register	F000 0A18$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 0A1C$_H$	BE	BE	–
ASC0_ TBUF	ASC0 Transmit Buffer Register	F000 0A20$_H$	U, SV	U, SV	0000 0000$_H$
ASC0_ RBUF	ASC0 Receive Buffer Register	F000 0A24$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 0A28$_H$ - F000 0A4C$_H$	BE	BE	–
ASC0_ WHBCON	ASC0 Write Hardware Bits Control Register	F000 0A50$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 0A54$_H$ - F000 0AEC$_H$	BE	BE	–
ASC0_ TSRC	ASC0 Transmit Interrupt Service Req. Control Reg.	F000 0AF0$_H$	U, SV	U, SV	0000 0000$_H$
ASC0_ RSRC	ASC0 Receive Interrupt Service Req. Control Reg.	F000 0AF4$_H$	U, SV	U, SV	0000 0000$_H$

Table 16-10 Address Map of ASC0/ASC1 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
ASC0_ ESRC	ASC0 Error Interrupt Service Req. Control Reg.	F000 0AF8$_H$	U, SV	U, SV	0000 0000$_H$
ASC0_ TBSRC	ASC0 Transmit Buffer Interrupt Service Req. Control Reg.	F000 0AFC$_H$	U, SV	U, SV	0000 0000$_H$
Async./Sync. Serial Interface 1 (ASC1)					
–	Reserved	F000 0B00$_H$	BE	BE	–
ASC1_ PISEL	ASC1 Peripheral Input Select Register	F000 0B04$_H$	U, SV	U, SV	0000 0000$_H$
ASC1_ ID	ASC1 Module Identification Register	F000 0B08$_H$	U, SV	BE	0000 44XX$_H$
–	Reserved	F000 0B0C$_H$	BE	BE	–
ASC1_ CON	ASC1 Control Register	F000 0B10$_H$	U, SV	U, SV	0000 0000$_H$
ASC1_ BG	ASC1 Baud Rate/Timer Reload Register	F000 0B14$_H$	U, SV	U, SV	0000 0000$_H$
ASC1_ FDV	ASC1 Fractional Divider Register	F000 0B18$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 0B1C$_H$	BE	BE	–
ASC1_ TBUF	ASC1 Transmit Buffer Register	F000 0B20$_H$	U, SV	U, SV	0000 0000$_H$
ASC1_ RBUF	ASC1 Receive Buffer Register	F000 0B24$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 0B28$_H$ - F000 0B4C$_H$	BE	BE	–
ASC1_ WHBCON	ASC1 Write Hardware Bits Control Register	F000 0B50$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 0B54$_H$ - F000 0BEC$_H$	BE	BE	–
ASC1_ TSRC	ASC1 Transmit Interrupt Service Req. Control Reg.	F000 0BF0$_H$	U, SV	U, SV	0000 0000$_H$
ASC1_ RSRC	ASC1 Receive Interrupt Service Req. Control Reg.	F000 0BF4$_H$	U, SV	U, SV	0000 0000$_H$

Table 16-10 Address Map of ASC0/ASC1 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
ASC1_ ESRC	ASC1 Error Interrupt Service Req. Control Reg.	F000 0BF8$_H$	U, SV	U, SV	0000 0000$_H$
ASC1_ TBSRC	ASC1 Transmit Buffer Interrupt Service Req. Control Reg.	F000 0BFC$_H$	U, SV	U, SV	0000 0000$_H$

17 Synchronous Serial Interface (SSC)

This chapter describes the two SSC Synchronous Serial Interfaces SSC0 and SSC1 of the TC1797. It contains the following sections:

- Functional description of the SSC kernel, valid for SSC0 and SSC1(see **Page 17-1**).
- SSC kernel register description, describes all SSC kernel specific registers (see **Page 17-21**).
- TC1797 implementation-specific details and registers of the SSC0/SSC1/SSC2 modules (port connections and control, interrupt control, address decoding, clock control, see **Page 17-37**).

*Note: The SSC kernel register names described in **Section 17.2** are referenced in the TC1797 User´s Manual by the module name prefix "SSC0_" for the SSC0 interface and by "SSC1_" for the SSC1 interface.*

17.1 SSC Kernel Description

Figure 17-1 shows a global view of the SSC interface.

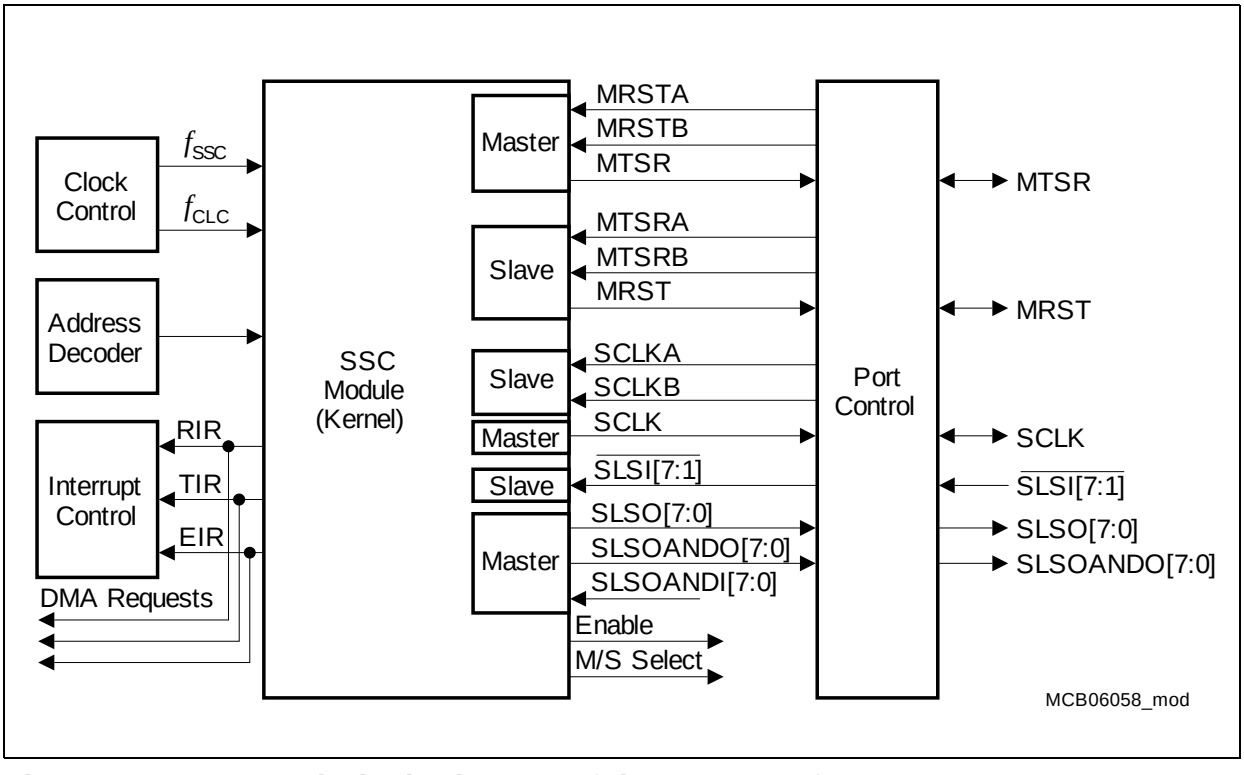


Figure 17-1 General Block Diagram of the SSC Interface

17.1.1 Overview

The SSC supports full-duplex and half-duplex serial synchronous communication up to 45.0 Mbit/s (@ 90 MHz module clock, Master Mode). The serial clock signal can be

generated by the SSC itself (Master Mode) or can be received from an external master (Slave Mode). Data width, shift direction, clock polarity and phase are programmable. This allows communication with SPI-compatible devices. Transmission and reception of data is double-buffered. A shift clock generator provides the SSC with a separate serial clock signal. Seven slave select inputs are available for Slave Mode operation. Eight programmable slave select outputs (chip selects) are supported in Master Mode.

Features:

- Master and Slave Mode operation
  - Full-duplex or half-duplex operation
  - Automatic pad control possible
- Flexible data format
  - Programmable number of data bits: 2 to 16 data bits
  - Programmable shift direction: LSB or MSB shift first
  - Programmable clock polarity: Idle low or idle high state for the shift clock
  - Programmable clock/data phase: Data shift with leading or trailing edge of the shift clock
- Baud rate generation
  - Master Mode: 45.0 Mbit/s to 686.65 bit/s (@ 90 MHz module clock)
  - Slave Mode: 22.5 Mbit/s to 686.65 bit/s (@ 90 MHz module clock)
- Interrupt generation
  - On a transmitter empty condition
  - On a receiver full condition
  - On an error condition (receive, phase, baud rate, transmit error)
- Flexible SSC pin configuration
- Seven slave select inputs $\overline{\text{SLSI}}$[7:1] in Slave Mode
- Eight programmable slave select outputs SLSO[7:0] in Master Mode
  - Automatic SLSO generation with programmable timing
  - Programmable active level and enable control
  - Combinable with SLSO output signals from other SSC modules

17.1.2 General Operation

The SSC supports full-duplex and half-duplex synchronous communication up to 45.0 Mbit/s (@ 90 MHz module clock). The serial clock signal can be generated by the SSC itself (Master Mode) or be received from an external master (Slave Mode). Data width, shift direction, clock polarity and phase are programmable. This allows communication with SPI-compatible devices. Transmission and reception of data are double-buffered. A shift clock generator provides the SSC with a separate serial clock signal.

Configuration of the high-speed synchronous serial interface is very flexible, so it can work with other synchronous serial interfaces, can serve master/slave or multi-master interconnections, or can operate compatibly with the popular SPI interface. It can be used to communicate with shift registers (I/O expansion), peripherals (e.g. EEPROMs etc.), or other controllers (networking). The SSC supports half-duplex and full-duplex communication. Data is transmitted or received on pins MTSR (Master Transmit/Slave Receive) and MRST (Master Receive/Slave Transmit). The clock signal is output or input via pin SCLK (Serial Clock). These three pins are typically used for alternate output functions of port pins. If they are implemented as dedicated bi-directional pins, they can be directly controlled by the SSC. In Slave Mode, the SSC can be selected from a master via dedicated slave select input lines (SLSI). In Master Mode, automatic generation of slave select output lines (SLSO) is supported..

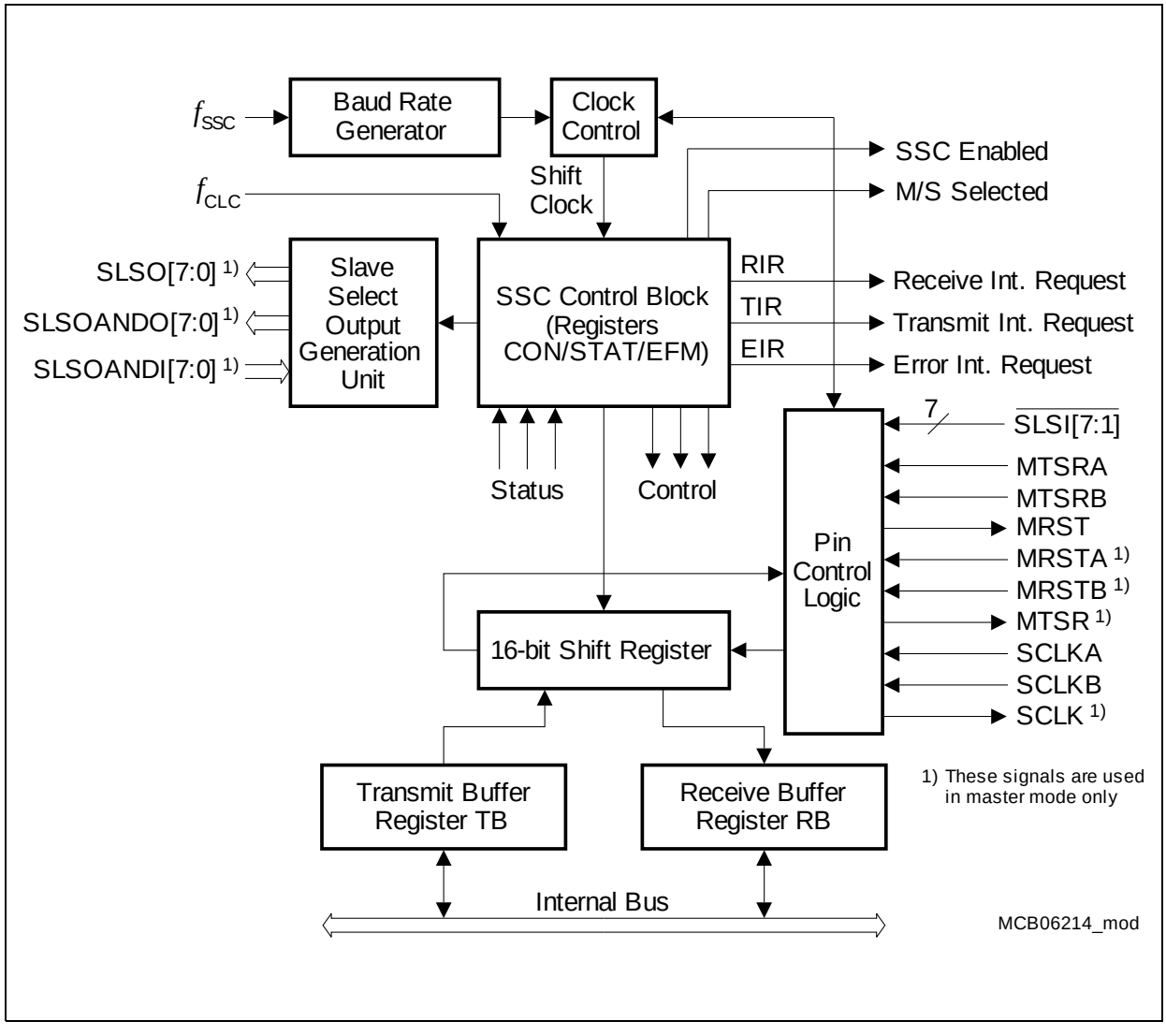


Figure 17-2 Synchronous Serial Channel SSC Block Diagram

17.1.2.1 Operating Mode Selection

The operating mode of the serial channel SSC is controlled by its Control Register, CON. Status information is contained in its Status Register, STAT.

The shift register of the SSC is connected to both the transmit pin and the receive pin via the pin control logic (see block diagram in **Figure 17-2**). Transmission and reception of serial data are synchronized and take place at the same time, that is, the same number of transmitted bits is also received. Transmit data is written into the Transmit Buffer (TB). It is moved to the shift register as soon as this is empty. An SSC master (CON.MS = 1) immediately begins transmitting, while an SSC slave (CON.MS = 0) will wait for an active shift clock. When the transfer starts, the busy flag STAT.BSY is set, and the transmit interrupt request line (TIR) will be activated to indicate that the Transmit Buffer Register (TB) may be reloaded. When the number of bits as programmed in CON.BM have been received, the data bits of the shift register are moved to the Receive Buffer Register (RB) right-aligned, and the receive interrupt request line (RIR) will be activated. If no further transfer is to take place (TB is empty), STAT.BSY will be cleared at the same time. Software should not modify STAT.BSY, as this flag is hardware-controlled.

Note: Only one SSC can be master at a given time.

The following features of the serial data bit transfer can be programmed:

- The data width can be selected from 2 to 16 data bits
- A transfer may start with the LSB or the MSB of the data bits
- The shift clock may be idle low or idle high
- The data bits may be shifted with the leading or trailing edge of the clock signal
- The baud rate (shift clock) can be set from 686.65 bit/s up to 45.0Mbit/s (@ 90 MHz module clock)
- The shift clock can be generated (master) or received (slave)

These features allow the SSC to be adapted to a wide range of applications that require serial data transfer.

The Data Width Selection supports the transfer of frames of any data length from 2-bit “characters” up to 16-bit “characters”.

Starting the serial data bit transfer with the LSB (CON.HB = 0) allows communication with devices such as an SSC device in Synchronous Mode, or 8051-like serial interfaces.

Starting with the MSB (CON.HB = 1) allows operation compatible with the SPI interface.

Regardless of the selected data width and whether the MSB or the LSB is transmitted first, the transfer data is always right-aligned in registers TB and RB, with the LSB of the transfer data in bit 0 of these registers. The data bits are rearranged for transfer by the internal shift register logic. The unselected bits of TB are ignored, and the unselected bits of RB will not be valid and should be ignored by the receiver service routine.

The Clock Control allows the adaptation of transmit and receive behavior of the SSC to a variety of serial interfaces. A specific clock edge (rising or falling) is used to shift out

transmit data, while the other clock edge is used to latch in receive data. Bit CON.PH selects the leading edge or the trailing edge for each function. Bit CON.PO selects the level of the clock line in the idle state. For an idle-high clock, the leading edge is a falling one, a 1-to-0 transition (see **Figure 17-3**).

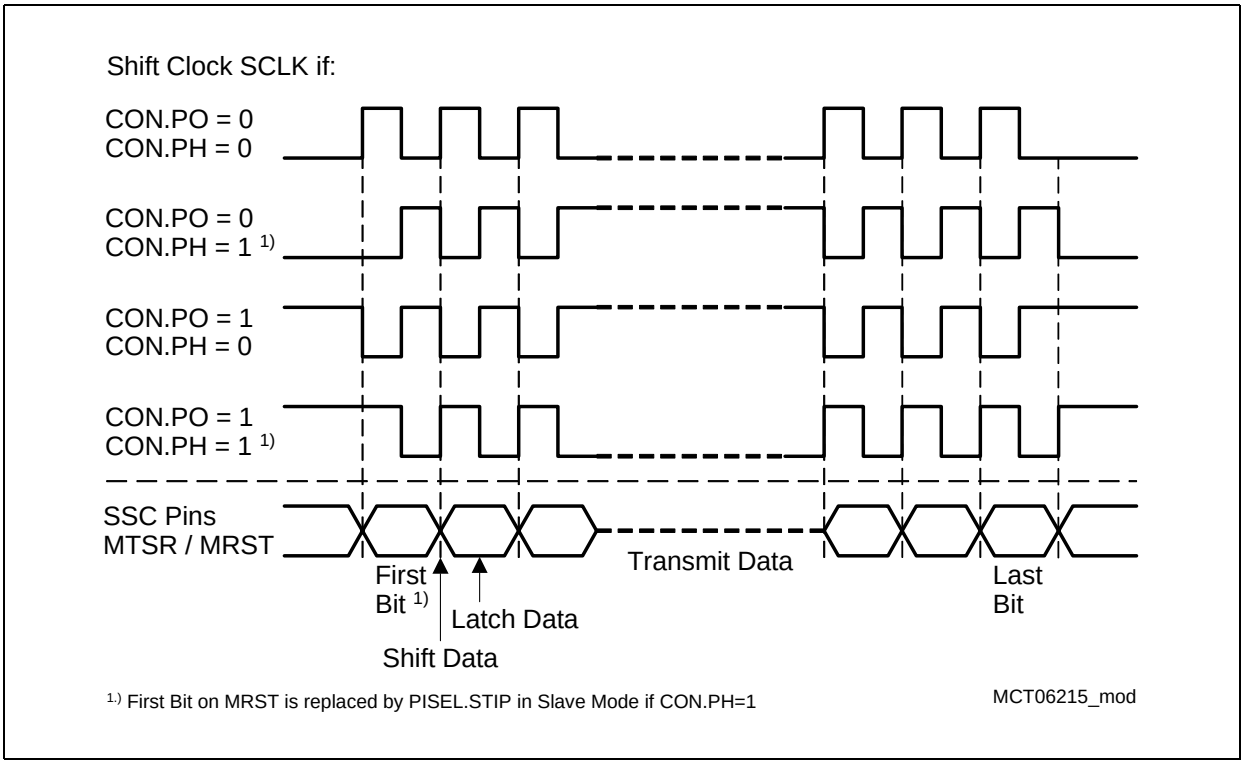


Figure 17-3 Serial Clock SCLK Phase and Polarity Options

17.1.2.2 Full-Duplex Operation

The description in this section assumes that the SSC is used with software controlled bi-directional GPIO port lines that have open-drain capability (see also **Section 17.1.2.5**).

The various devices are connected through three lines. The definition of these lines is always determined by the master. The line connected to the master's data output pin MTSR is the transmit line, the receive line is connected to its data input line MRST, and the clock line is connected to pin SCLK. Only the device selected for master operation generates and outputs the serial clock on pin SCLK. All slaves receive this clock, so their pin SCLK must be switched to input mode. The output of the master's shift register is connected to the external transmit line, which in turn is connected to the slaves' shift register input. The output of the slaves' shift register is connected to the external receive line in order to enable the master to receive the data shifted out of the slave. The external connections are hard-wired, with the function and direction of these pins determined by the master or slave operation of the individual device.

*Note: The shift direction shown in **Figure 17-4** applies to both MSB-first and LSB-first operation.*

When initializing the devices in this configuration, one device must be selected for master operation while all other devices must be programmed for slave operation. Initialization includes the operating mode of the device's SSC and also the function of the respective port lines.

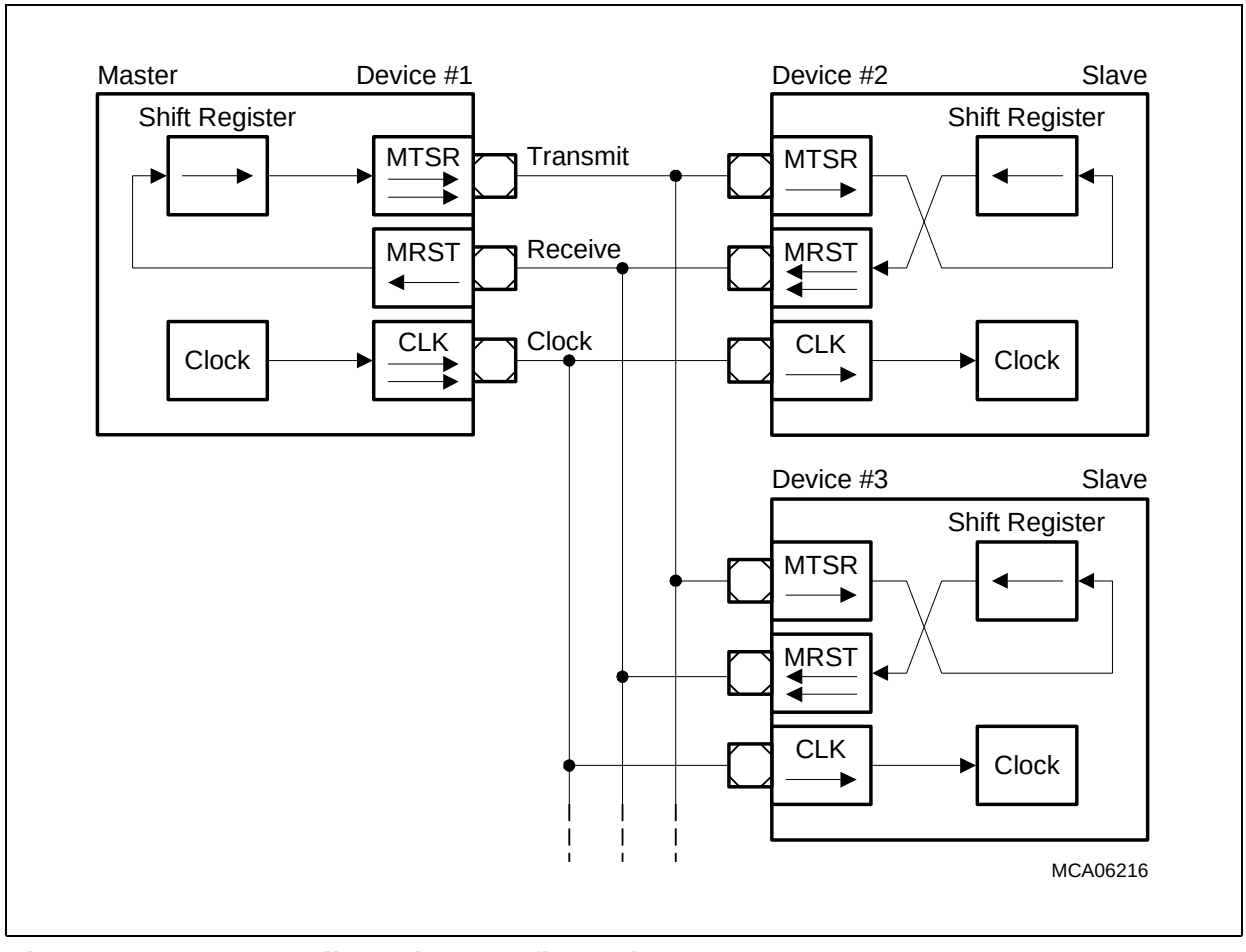


Figure 17-4 SSC Full-Duplex Configuration

The data output pins MRST of all slave devices are connected onto one receive line in this configuration. During a transfer, each slave shifts out data from its shift register. There are two ways to avoid collisions on the receive line due to different slave data:

- **Only one slave drives the line** and enables the driver of its MRST pin. All the other slaves must program their MRST pins to input. Therefore, only one slave can put its data onto the master's receive line. Only reception of data from the master is possible. The master selects the slave device from which it expects data either by separate select lines, or by sending a special command to this slave. The selected slave then switches its MRST line to output until it gets a de-selection signal or command.

- **The slaves use open drain output on MRST.** This forms a wired-AND connection. The receive line needs an external pull-up in this case. Corruption of the data on the receive line sent by the selected slave is avoided when all slaves not selected for transmission to the master send only 1s. Since this high level is not actively driven onto the line, but is only held through the pull-up device, the selected slave can pull this line actively to a low level when transmitting a zero bit. The master selects the slave device from which it expects data either by separate select lines, or by sending a special command to this slave.

After performing all necessary initializations of the SSC, the serial interfaces can be enabled. For a master device, the alternate clock line will now go to its programmed polarity. The alternate data line will go to either 0 or 1, until the first transfer starts. After a transfer, the alternate data line will always remain at the logic level of the last transmitted data bit.

When the serial interfaces are enabled, the master device can initiate the first data transfer by writing the transmit data into register TB. This value is copied into the shift register (assumed to be empty at this time), and the selected first bit of the transmit data will be placed onto the MTSR line on the next clock from the shift clock generator (transmission only starts, if CON.EN = 1). Depending on the selected clock phase, a clock pulse is generated on the SCLK line. With the opposite clock edge, the master simultaneously latches and shifts in the data detected at its input line MRST. This “exchanges” the transmit data with the receive data. Because the clock line is connected to all slaves, their shift registers will be shifted synchronously with the master’s shift register, shifting out the data contained in the registers, and shifting in the data detected at the input line. After the pre-programmed number of clock pulses (via the data width selection), the data transmitted by the master is contained in all slaves’ shift registers, while the master’s shift register holds the data of the selected slave. In the master and all slaves, the content of the shift register is copied into the Receive Buffer (RB) and the receive interrupt line (RIR) is activated.

A slave device will immediately output the selected first bit (MSB or LSB of the transfer data) at pin MRST when the contents of the transmit buffer are copied into the slave’s shift register. Bit STAT.BSY is not set until the first clock edge at SCLK appears. The slave device will not wait for the next clock from the shift clock generator – as the master does – because the first clock edge generated by the master may be already used to clock in the first data bit, depending on the selected clock phase. So the slave’s first data bit must already be valid at this time.

*Note: On the SSC, a transmission **and** a reception always take place at the same time, regardless whether valid data has been transmitted or received.*

17.1.2.3 Half-Duplex Operation

The description in this section assumes that the SSC is used with software controlled bi-directional GPIO port lines that provide open-drain capability (see also **Section 17.1.2.5**).

In a half-duplex configuration, only one data line is necessary for both receiving **and** transmitting data. The data exchange line is connected to both pins MTSR and MRST of each device, and the clock line is connected to the SCLK pin.

The master device controls the data transfer by generating the shift clock, while the slave devices receive it. Due to the fact that all transmit and receive pins are connected to the one data exchange line, serial data may be moved between arbitrary stations.

As in full-duplex mode, there are two ways to avoid collisions on the data exchange line:

- Only the transmitting device may enable its transmit pin driver
- The non-transmitting devices use open-drain output and send only 1s

Because the data inputs and outputs are connected together, a transmitting device will clock in its own data at the input pin (MRST for a master device, MTSR for a slave). In this way, any corruption is detected on the common data exchange line when the received data is not equal to the transmitted data.

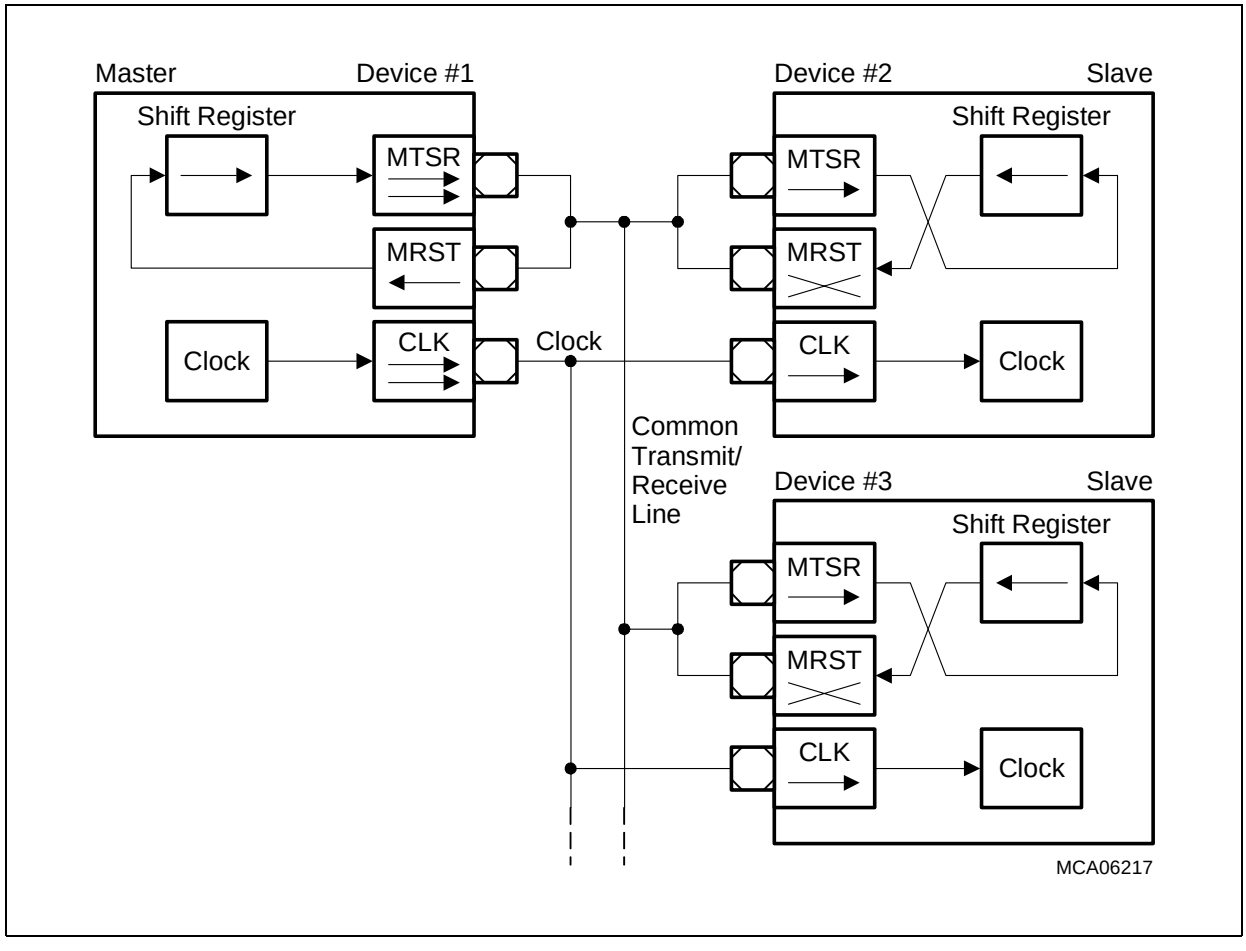


Figure 17-5 SSC Half-Duplex Configuration

17.1.2.4 Continuous Transfers

When the transmit interrupt request flag is set, it indicates that the Transmit Buffer (TB) is empty and is ready to be loaded with the next transmit data. If the TB has been reloaded by the time the current transmission is finished, the data is immediately transferred to the shift register and the next transmission can start without any additional delay (according to the selected SLSO timings). On the data line, there is no gap between the two successive frames if no delays are selected. For example, two byte transfers would look the same as one word transfer. This feature can be used to interface with devices that can operate with (or require more than) 16 data bits per transfer. It is just a matter for software how long a total data frame length can be. This option can also be used, e.g., to interface to byte-wide and word-wide devices on the same serial bus.

Note: This option can only happen in multiples of the selected basic data width, because it would require disabling/enabling of the SSC to reprogram the basic data width on-the-fly.

Note: In Master Mode, the Transmit Buffer register TB is loaded with new data for the following transmission just at the end of the current transmission and a leading delay > 0 is selected (SSOTC.LEAD not equal 00_B), a slightly enlarged leading delay (< one SCLK shift clock period) is generated for the following transmission.

17.1.2.5 Port Control

The SSC uses three lines to communicate with the external world. Pin SCLK serves as the clock line, while pins MRST (Master Receive/Slave Transmit) and MTSR (Master Transmit/Slave Receive) serve as the serial data input/output lines. As shown in **Figure 17-1** these three lines (SCLK as input, Master Receive, Slave Receive) have two inputs each at the SSC Module kernel. Three bits in register PISEL determine which of the two kernel inputs (A or B) are connected. This feature allows for each of the three SSC communication lines to be connected to two inputs coming from different port pins.

Operation of the SSC I/O lines depends on the selected operating mode (master or slave). The direction of the port lines depends on the operating mode. The SSC will automatically use the correct kernel output or kernel input line of the ports when switching modes. Port pins assigned as SSC I/O lines can be controlled either by hardware or by software.

When the SSC I/O lines are connected to dedicated pins, hardware I/O control should typically be used. In this case, two output signals reflect the state of the CON.EN and CON.MS bits directly (the M/S select line is inverted to the CON.MS bit definition).

When the SSC I/O lines are connected with bi-directional lines of general purpose I/O ports, software I/O control should be typically used. In this case port registers must be programmed for alternate output and input selection. When switching between master and slave mode port registers must be reprogrammed.

Using the open-drain output feature of port lines helps to avoid bus contention problems and reduces the need for hard-wired hand-shaking or slave select lines. In open-drain output mode, it is not always necessary to switch the direction of a port pin. Note that in hardware-controlled I/O mode, the availability of open-drain outputs depends on the type of the dedicated output pins that are used. The SSC module itself does not provide any control capability for open-drain control.

*Note: For details of SSC port connections and configuration, see **Page 17-45**.*

17.1.2.6 Baud Rate Generation

The serial channel SSC has its own dedicated 16-bit baud rate generator with 16-bit reload capability, allowing baud rate generation independent of timers. In addition to **Figure 17-2**, **Figure 17-6** shows the baud rate generator of the SSC in more detail.

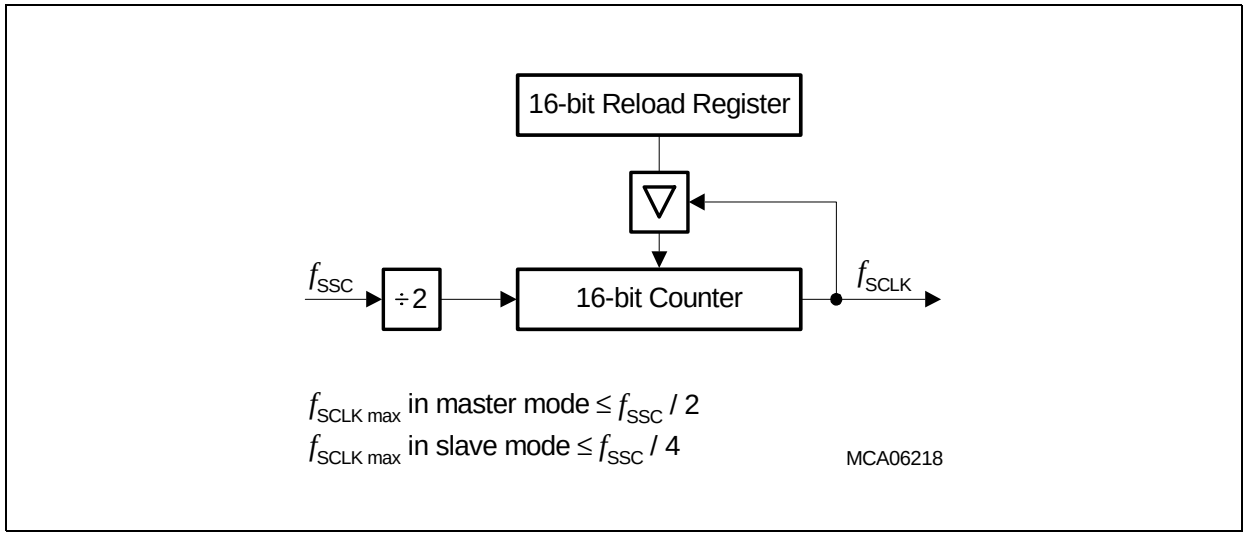


Figure 17-6 SSC Baud Rate Generator

The baud rate generator is clocked with f_{SSC}. The timer counts downwards. Register BR is the dual-function Baud Rate Generator/Reload register. Reading BR while the SSC is enabled returns the contents of the timer. Reading BR while the SSC is disabled returns the programmed reload value. In this mode, the desired reload value can be written to BR.

Note: Never write to BR while the SSC is enabled.

The formulas below calculate either the resulting baud rate for a given reload value, or the required reload value for a given baud rate:

$$\text{Baud rate}_{SSC} = \frac{f_{SSC}}{2 \times (\text{BR\_VALUE} + 1)} \qquad \text{BR\_VALUE} = \frac{f_{SSC}}{2 \times \text{Baud rate}_{SSC}} - 1 \qquad (17.1)$$

BR_VALUE represents the content of the reload register, taken as an unsigned 16-bit integer, while Baud rate_{SSC} is equal to f_{SCLK} as shown in **Figure 17-6**.

The maximum baud rate that can be achieved with f_{SSC} = 90 MHz is 45.0 Mbit/s in Master Mode (with BR_VALUE = 0000_H) and 22.5 Mbit/s in Slave Mode (with BR_VALUE = 0001_H).

Table 17-1 lists some possible baud rates together with the required reload values and the resulting bit times, assuming a module clock f_{SSC} of 90 MHz.

Table 17-1 Typical Baud Rates of the SSC (f_{SSC} = 90 MHz)

Reload Value	Baud Rate (= f_{SCLK})	Deviation
0000_H	45 Mbit/s (only in Master Mode)	0.0%
0001_H	22.5 Mbit/s	0.0%
0003_H	11.25 Mbit/s	0.0%
$002C_H$	1 Mbit/s	0.0%
0063_H	450 kbit/s	0.0%
$01C1_H$	100 kbit/s	0.0%
1194_H	10 kbit/s	0.0%
$AFC7_H$	1 kbit/s	0.0%
$FFFF_H$	686.65 bit/s	0.0%

In the TC1797, the module clock f_{SSC} is generated outside the SSC module kernel. Therefore, for baud rate calculations the dependencies of f_{SSC} from f_{SYS} must be taken into account. **Section 17.3.4.1** on **Page 17-41** describes these dependencies in detail.

17.1.2.7 Slave Select Input Operation

For systems with multiple slaves, the SSC module provides seven $\overline{\text{SLSI}}$ slave select input lines, that permit enabling or disabling of the SCLK, MTSR, and MRST signals in Slave Mode. Slave Mode is selected by CON.MS = 0. The $\overline{\text{SLSI}}$ input logic shown in **Figure 17-7** is controlled by register PISEL and CON.

Note: In the following description, only one of the seven $\overline{\text{SLSI}}$ input lines is mentioned. The remaining six $\overline{\text{SLSI}}$ input lines are connected to the other six inputs of the input multiplexer, which is controlled by PISEL.SLSIS.

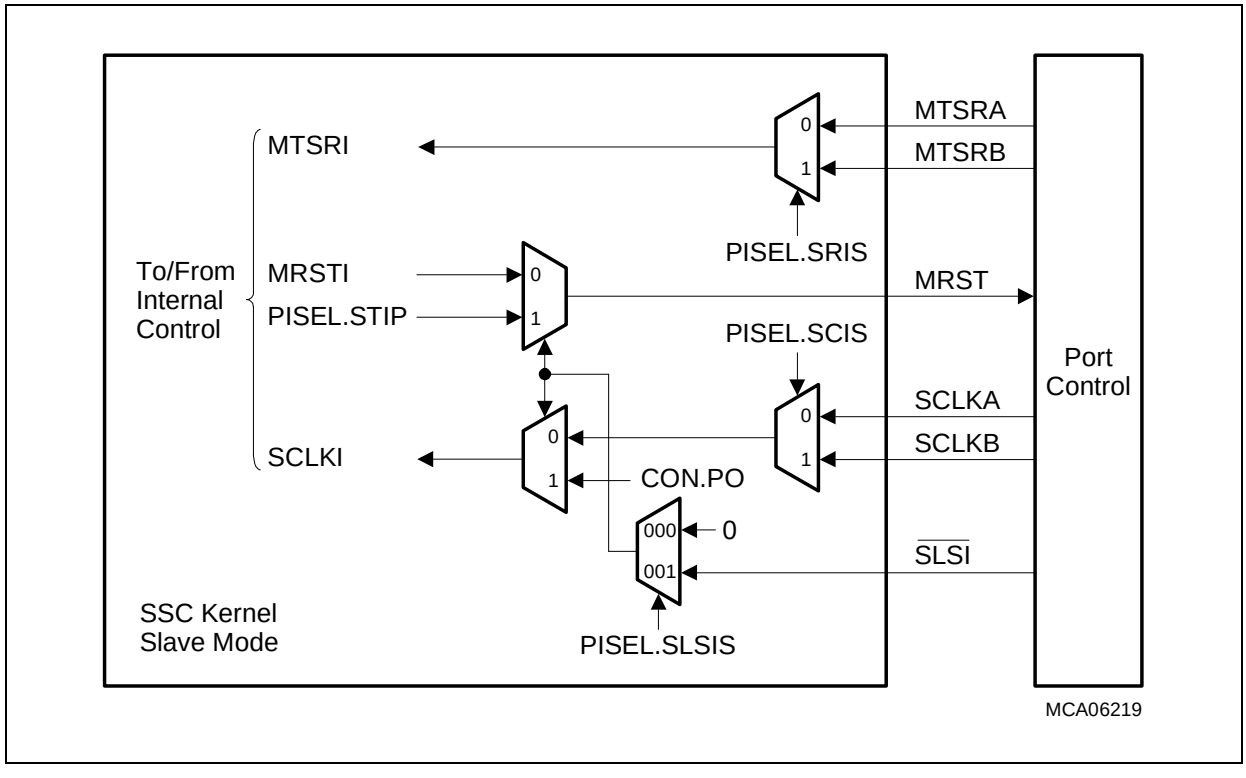


Figure 17-7 Slave Select Input Logic

With PISEL.SLSIS = 000_B and Slave Mode selected, the $\overline{\text{SLSI}}$ input line does not control the SSC I/O lines. The slave receive input signal MTSRA or MTSRB (selected by PISEL.SRIS) and the slave clock input signal SCLKA or SCLKB (selected by PISEL.SCIS) are passed further as MTRSI and SCLKI to the internal SSC control logic. The slave transmit signal MRSTI from the internal SSC control logic MRSTI is passed directly to MRST.

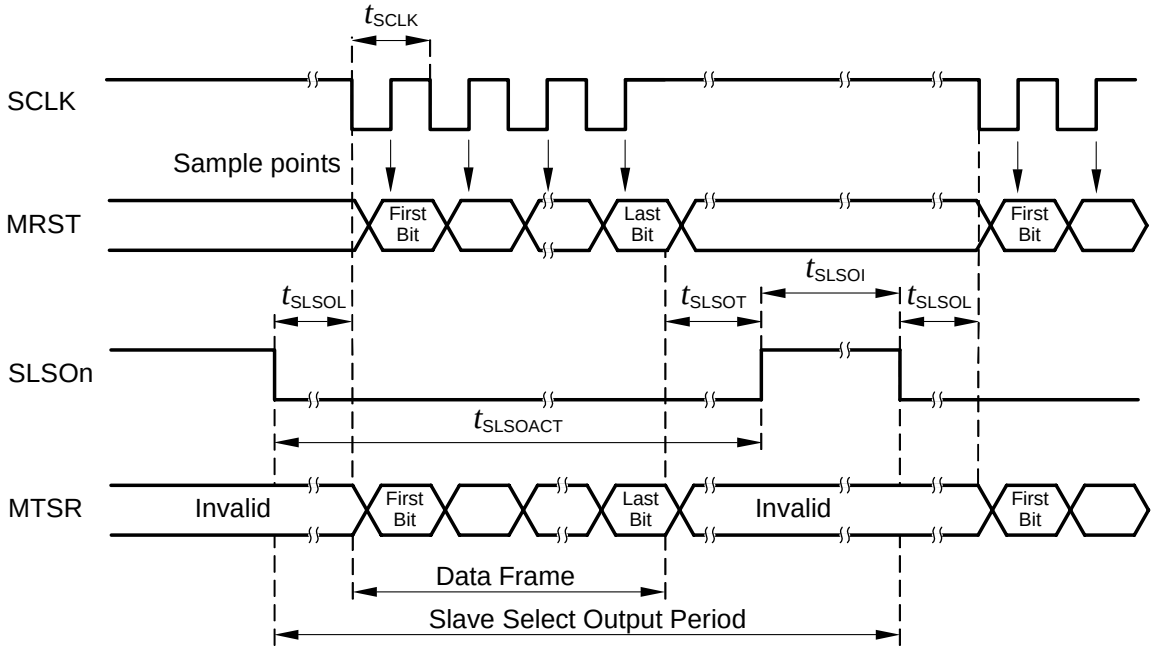
With PISEL.SLSIS = 001_B, input signal $\overline{\text{SLSI}}$ controls the operation of the SSC I/O lines as a slave select signal as follows:

- $\overline{\text{SLSI}}$ = 1: SSC slave is not selected.
  - The slave receive input signals, MTSRA or MTSRB are connected to MTSRI, depending on PISEL.SRIS (Slave Mode receive input select).

  - MRST is driven with the logic level of bit PISEL.STIP (slave transmit idle state).
  - SCLKI is driven with the logic level of CON.PO (clock polarity control).
- SLSI = 0: SSC is selected as slave.
  - The slave receive input signals MTSRA or MTSRB are connected to MTSRI, depending on PISEL.SRIS (Slave Mode receive input select).
  - MRST is directly driven with the slave transmit output signal MRSTI.
  - The slave clock input signals SCLKA or SCLKB are connected to SCLKI, depending on PISEL.SCIS (Slave Mode clock input select).

17.1.2.8 Slave Select Output Generation Unit

In Master Mode, the slave select output generation unit of the SSC automatically generates up to eight slave select output lines SLSO[7:0] for serial transmit operations. The slave select output generation unit further makes it possible to adjust the chip select timing parameters. The active/inactive state of a slave select output as well as the enable/disable state can be controlled individually for each slave select output (see **Figure 17-9**). The basic slave select output timing is shown in **Figure 17-8**, assuming a low active level of the SLSOn lines.



Note: This timing example is based on the following setup: CON.PH = 0; CON.PO = 1

MCT06220_mod

Figure 17-8 SSC Slave Select Output Timing

A slave select output period always starts after a write operation to register TB. With a TB write operation, all timing parameters stored in register SSOTC (LEAD, TRAIL,

INACT, and SLSO7MOD) and register SSOC (AOLn and OENn) are latched and remain valid for the consecutive transmission. Following that, SLSOn becomes active (low) for a number of SCLK cycles (leading delay cycles) before the first bit of the serial data stream occurs at MTSR. After the transmission of the data frame, SLSOn remains active (low) for a number of SCLK cycles (trailing delay cycles) before it becomes inactive again. This inactive state of SLSOn is valid at least for a number of SCLK cycles (inactive delay cycles) before a new chip select period can be started.

Note: When operating in Master Mode with CON.PH = 1 and sampling data from a slave device that becomes enabled by an SLSOn output, a leading delay of at least one leading delay clock cycle should be selected. The reason is that with CON.PH = 1, the first SCLK edge already latches the first data bit at MRST.

The three parameters of a chip select period are controlled by bit fields in the Slave Select Output Timing Control Register SSOTC. Each of these bit fields can contain a value from 0 to 3 defining delay cycles of 0 to 3 multiples of the t_{SCLK} shift clock period. The three parameters are:

1. Number of leading delay cycles (t_{SLSOL} = SSOTC.LEAD × t_{SCLK})
2. Number of trailing delay cycles (t_{SLSOT} = SSOTC.TRAIL × t_{SCLK})
3. Number of inactive delay cycles (t_{SLSOI} = SSOTC.INACT × t_{SCLK})

If SSOTC.INACT = 00_B and register TB has already been loaded with the data for the next data frame, the next chip select period is started with its leading delay phase without SLSOn going inactive. If, in this case, TB has not been loaded in time with the data for the next data frame, SLSOn becomes inactive again.

Slave Select Output Control

Each slave select output SLSOn can be enabled individually. When SSOC.OENn = 1, SLSOn is enabled. Furthermore, active and inactive levels of the SLSOn outputs are programmable. Bit SSOC.AOLn determines the state of the active level of SLSOn.

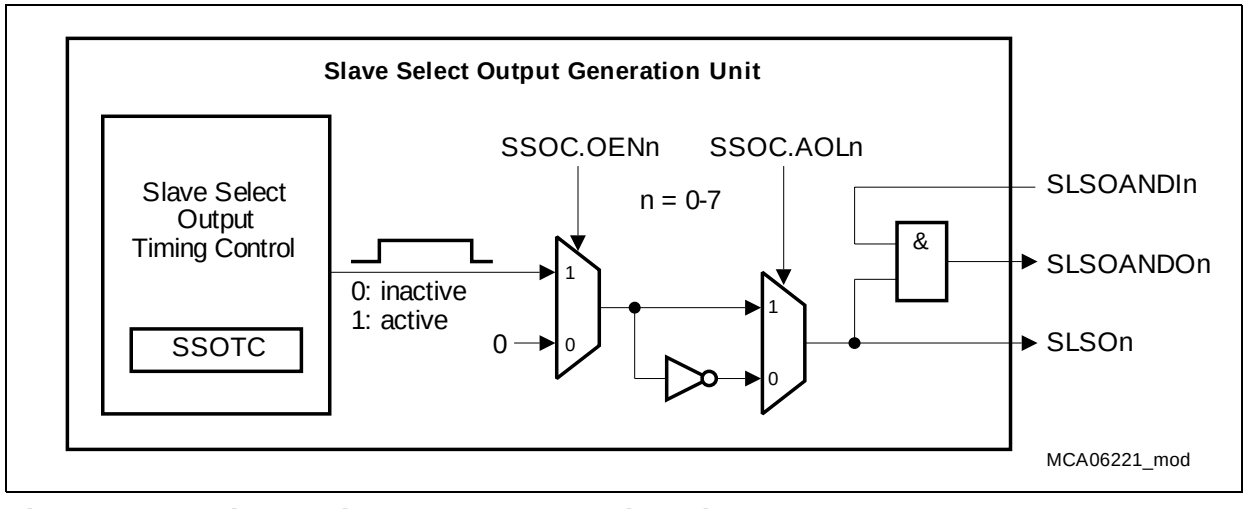
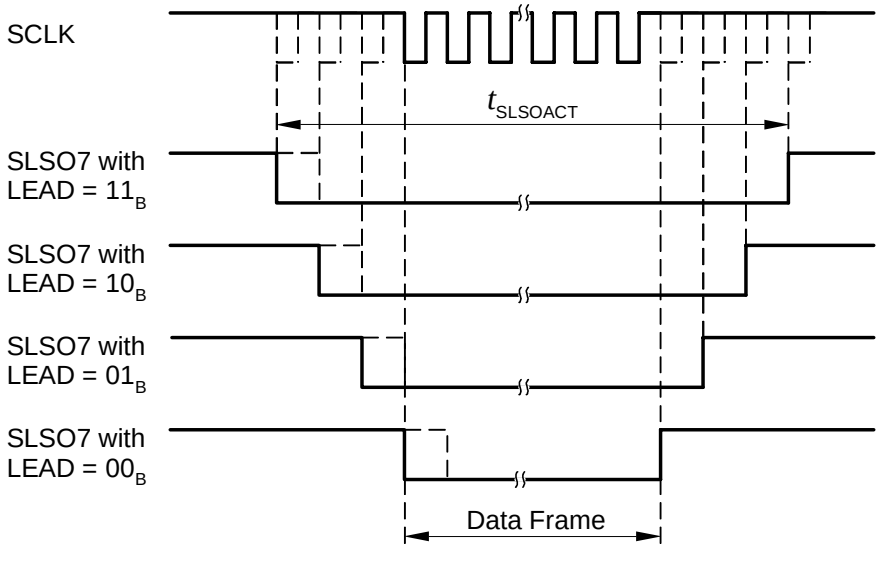


Figure 17-9 Slave Select Output Control Logic

As a special feature, each SLSOn output signal can be combined (ANDed) by an external signal SLSOANDIn coming from another SSC to an output signal SLSANDOn. This AND gate can be used for example to combine two slave select output signals from two SSCs to one common SLSOn output signal. Note that this functionality only works for low active SLSOn signals (SSOC.AOLn = 0).

Slave Select Output 7 Delayed Mode

In the SLSO7 delayed mode (SSOTC.SLSO7MOD = 1), the timing of the slave select output SLSO7 as programmed by the three parameters in SSOTC (number of trailing, leading, and inactive delay clock cycles) is delayed by one shift clock period for the inactive-to-active edge. The active-to-inactive edge is not delayed. The timing of SLSO7 in the delayed mode is shown in **Figure 17-10**. The bold lines show the timing of SLSO7 in normal operating mode, and the dotted lines show the timing of SLSO7 in delayed mode.



Note: The timing is valid for clock polarity control bit CON.PO = 1

MCT06222

Figure 17-10 SLSO7 Delayed Mode

Slave Select Register Update

At the start of an internal transmit sequence (with the TB register write operation), the parameters in registers SSOC and SSOTC are latched. This means that they remain stable while a serial transmission is in progress. Therefore, it is always guaranteed that the data of one serial transmission is always transmitted with a constant slave select configuration setup. A configuration change by reprogramming SSOC or SSOTC during a serial transmission will first become valid with the start of the subsequent serial transmission.

17.1.2.9 Error Detection Mechanisms

The SSC is able to detect four different error conditions. Receive Error and Phase Error are detected in all modes, while Transmit Error and Baud Rate Error apply to Slave Mode only. In case of a Transmit Error or Receive Error, the respective error flags are always set and the error interrupt requests will be generated by activating the EIR line only if the corresponding error enable bits have been set (see **Figure 17-11**). The error interrupt handler may then check the error flags to determine the cause of the error interrupt. The error flags are not cleared automatically, but must be cleared via register EFM after servicing. This allows servicing of some error conditions via interrupt, while others may be polled by software. The error status flags can be set and cleared by software via the error flag modification register EFM.

Note: The error interrupt handler must clear the associated (enabled) error flag(s) to prevent repeated interrupt requests. The setting of an error flag by software does not generate an interrupt request.

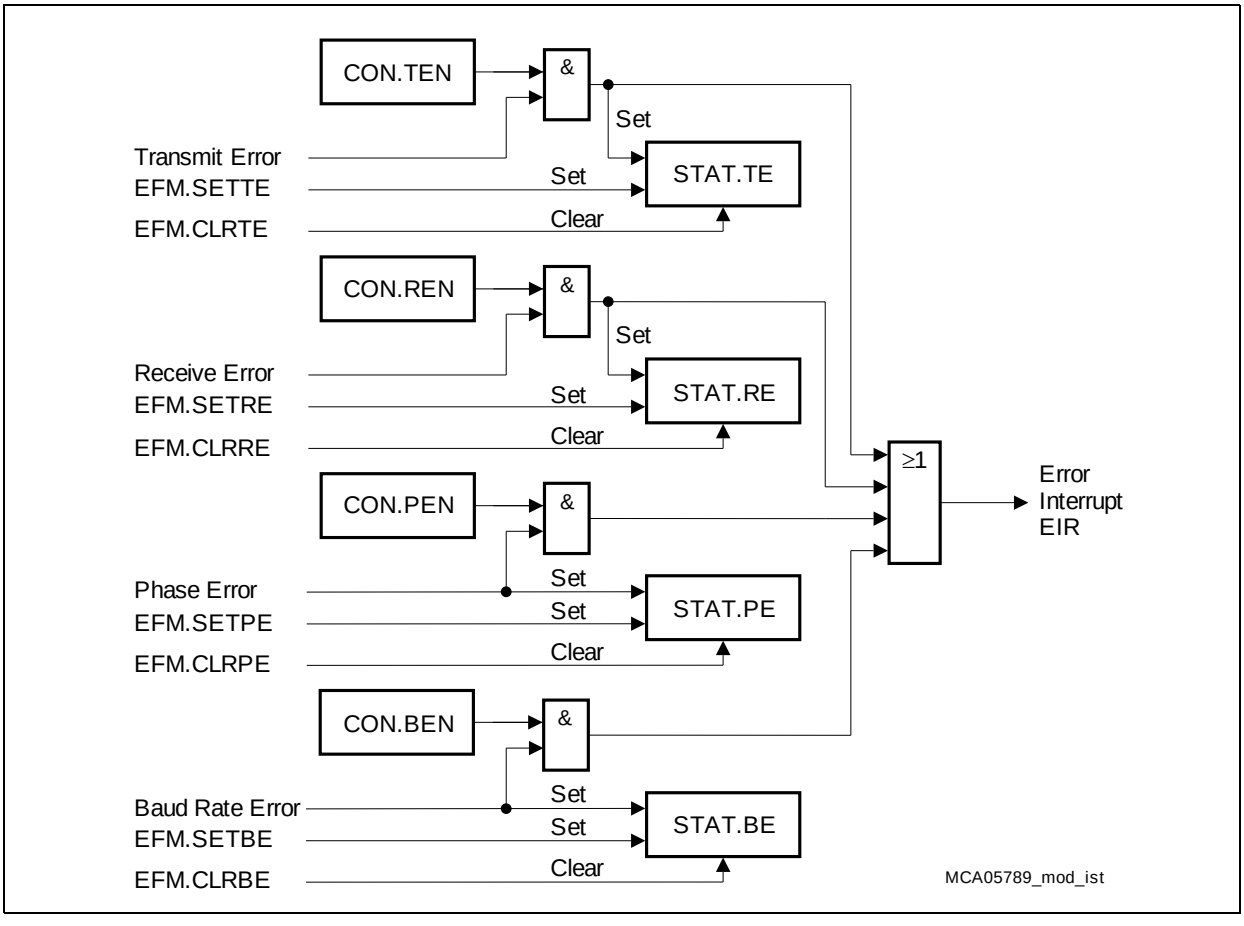


Figure 17-11 SSC Error Interrupt Control

A **Receive Error** (Master or Slave mode) is detected when a new data frame is completely received, but the previous data was not read out of the receive buffer register

RB. If enabled via CON.REN, this condition sets the error flag STAT.RE and activates the error interrupt request line EIR. This condition sets the error flag STAT.RE and, if enabled via CON.REN, sets the error interrupt request line EIR. The old data in the receive buffer RB will be overwritten with the new value and is irretrievably lost.

A **Phase Error** (Master or Slave Mode) is detected when the incoming data at pin MRST (Master Mode) or MTSR (Slave Mode), sampled with the same frequency as the module clock, changes between one cycle before and two cycles after the latching edge of the shift clock signal SCLK. This condition sets the error status flag STAT.PE and, if enabled via CON.PEN, the error interrupt request line EIR.

Note: When CON.PH = 1, the data output signal may be disturbed shortly when the slave select input signal is changed after a serial transmission, resulting in a phase error.

A **Baud Rate Error** (Slave Mode) is detected when the incoming clock signal deviates from the programmed baud rate (shift clock) by more than 100%, meaning it is either more than double or less than half the expected baud rate. This condition sets the error status flag STAT.BE and, if enabled via CON.BEN, the EIR line. Using this error detection capability requires that the slave's shift clock generator is programmed to the same baud rate as the master device. This feature detects false additional pulses or missing pulses on the clock line (within a certain frame).

Note: If this error condition occurs and bit CON.AREN = 1, an automatic reset of the SSC will be performed. This is done to re-initialize the SSC, if too few or too many clock pulses have been detected.

Note: The baud rate error can occur in slave mode after any transfer if the communication is stopped by the master. This is the case due to the fact that SSC module supports back-to-back transfers for multiple transfers. In order to handle this the baud rate detection logic expects after a finished transfer immediately a next clock cycle for a new transfer.

If baud rate error is enabled and the transmit buffer of the slave SSC is loaded with a new value for the next data frame while the current data frame is not yet finished (while STAT.BSY = 1), the slave SSC expects continuation of the clock pulses for the next data frame transmission immediately after finishing the current data frame. Any write to TBUF of the slave SSC while STAT.BSY = 1 initiates or sustains a continuous transmission in the slave. Therefore, the master (shift) clock must be continued after the current frame transmission. Otherwise, the slave SSC will detect a baud rate error. Note that the master SSC does not necessarily send out a continuous shift clock in the case that its transmit buffer is not yet filled with new data or transmission delays occur. Further details on continuous transfers are described in **Section 17.1.2.4** on **Page 17-10**.

A **Transmit Error** (Slave Mode) is detected when a transfer was initiated by the master (shift clock gets active), but the transmit buffer (TB) of the slave was not updated since the last transfer. If enabled via CON.TEN, this condition sets the error status flag

STAT.TE and activates the EIR line. This condition sets the error status flag STAT.TE and, if enabled via CON.TEN, the EIR line. If a transfer starts while the transmit buffer is not updated, the slave will shift out the ‘old’ contents of the shift register, which is normally the data received during the last transfer. This may lead to the corruption of the data on the transmit/receive line in half-duplex mode (open drain configuration) if this slave is not selected for transmission. This mode requires that slaves not selected for transmission only shift out ones; thus, their transmit buffers must be loaded with $FFFF_H$ prior to any transfer.

Note: A slave with push/pull output drivers not selected for transmission will normally have its output drivers switched off. However, to avoid possible conflicts or misinterpretations, it is recommended to always load the slave's transmit buffer prior to any transfer.

The cause of an error interrupt request (receive, phase, baud rate, transmit error) can be identified by the error status flags in control register CON.

Note: In contrast to the EIR line, the error status flags STAT.TE, STAT.RE, STAT.PE, and STAT.BE, are not automatically cleared upon entry into the error interrupt service routine, but must be cleared by software.

17.2 SSC Kernel Registers

This section describes the kernel registers of the SSC module. All SSC kernel register names described in this section will be referenced in other parts of the TC1797 User´s Manual by the module name prefix “SSC0_” for the SSC0 interface and “SSC1_” for the SSC1 interface.

All registers in the SSC address spaces are reset with the application reset (definition see SCU section “Reset Operation”).

SSC Kernel Register Overview

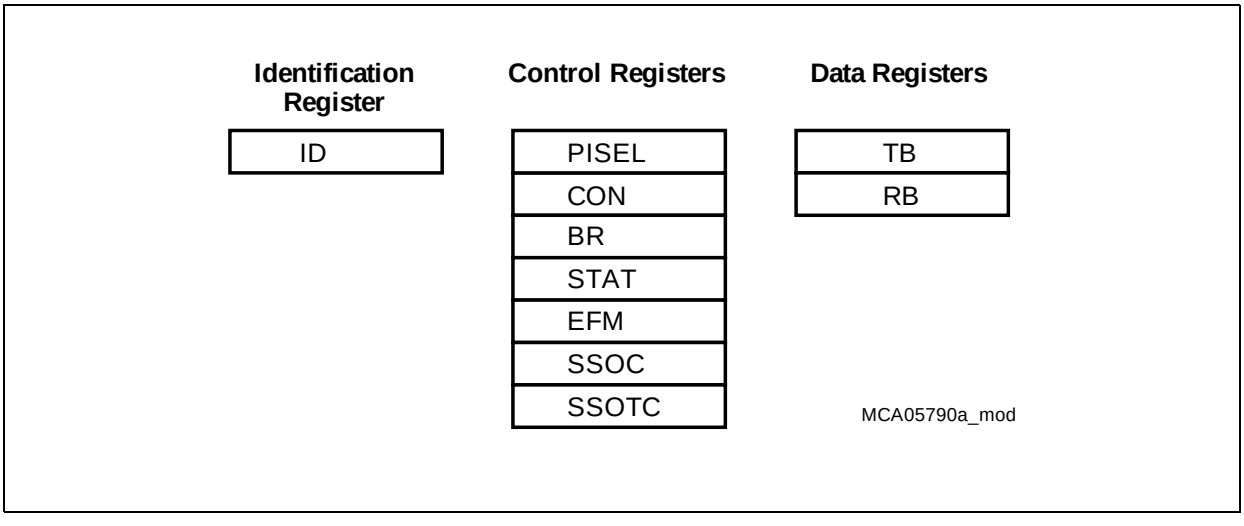


Figure 17-12 SSC Kernel Registers

The complete and detailed address map of the SSC modules is described in **Table 17-6** on **Page 17-50**.

Table 17-2 Registers Address Space - SSC Kernel Registers

Module	Base Address	End Address	Note
SSC0	F010 0100$_H$	F010 01FF$_H$	–
SSC1	F010 0200$_H$	F010 02FF$_H$	–

Table 17-3 Registers Overview - SSC Kernel Registers

Register Short Name	Register Long Name	Offset Address[1)]	Description see
PISEL	Port Input Select Register	04$_H$	**Page 17-23**
ID	Module Identification Register	08$_H$	**Page 17-22**
CON	Control Register	10$_H$	**Page 17-25**

Table 17-3 Registers Overview - SSC Kernel Registers (cont'd)

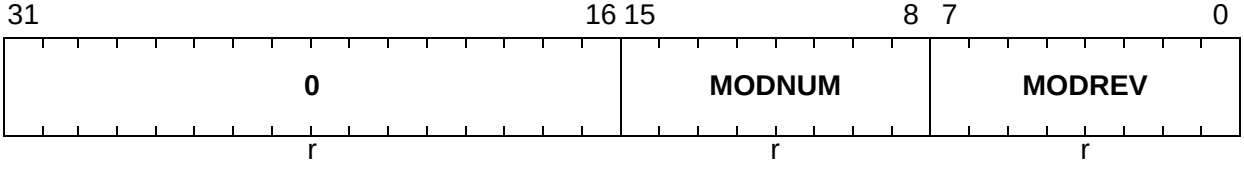
Register Short Name	Register Long Name	Offset Address[1)]	Description see
BR	Baud Rate Timer Reload Register	14_H	Page 17-35
STAT	Status Register	28_H	Page 17-28
EFM	Error Flag Modification Register	$2C_H$	Page 17-30
SSOC	Slave Select Output Control Register	18_H	Page 17-32
SSOTC	Slave Select Output Timing Control Register	$1C_H$	Page 17-33
TB	Transmit Buffer Register	20_H	Page 17-36
RB	Receive Buffer Register	24_H	Page 17-36

1) The absolute register address is calculated as follows:
Module Base Address (Table 17-2) + Offset Address (shown in this column)

17.2.1 Module Identification Register

The SSC Module Identification Register ID contains read-only information about the module version.

ID
Module Identification Register (08_H) Reset Value: 0000 45XX$_H$



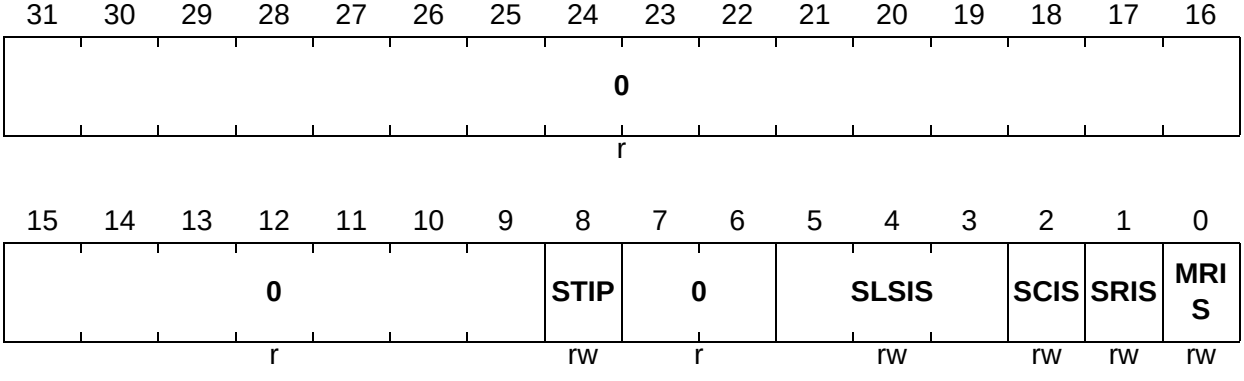
Field	Bits	Type	Description
MODREV	[7:0]	r	**Module Revision Number** MODREV defines the module revision number. The value of a module revision starts with 01_H (first revision).
MODNUM	[15:8]	r	**Module Number Value** This bit field defines the module identification number for the SSC: 45_H
0	[31:16]	r	**Reserved** Read as 0.

Note: Implementation specific details (e.g. reset value) see ***"Module Identification Registers" on Page 17-37****.*

17.2.2 Control Registers

The PISEL register controls the input signal selection of the SSC module. Each input of the module kernel receive, transmit and clock signals has associated two input lines (marked by suffix A and B).

PISEL
Port Input Select Register **(04_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
MRIS	0	rw	**Master Mode Receive Input Select** MRIS selects the receive input line in Master Mode. 0_B Receive input line MRSTA is selected 1_B Receive input line MRSTB is selected
SRIS	1	rw	**Slave Mode Receive Input Select** SRIS selects receive input line in Slave Mode. 0_B Receive input line MTSRA is selected 1_B Receive input line MTSRB is selected
SCIS	2	rw	**Slave Mode Clock Input Select** SCIS selects the module kernel SCLK input line that is used as clock input line in slave mode. 0_B Slave Mode clock input line SCLKA is selected 1_B Slave Mode clock input line SCLKB is selected

Field	Bits	Type	Description
SLSIS	[5:3]	rw	**Slave Mode Slave Select Input Selection** 000_B Slave select input lines are deselected; SSC is operating without slave select input functionality. 001_B SLSI input line 1 is selected for operation. 010_B SLSI input line 2 is selected for operation. 011_B SLSI input line 3 is selected for operation. 100_B SLSI input line 4 is selected for operation. 101_B SLSI input line 5 is selected for operation. 110_B SLSI input line 6 is selected for operation. 111_B SLSI input line 7 is selected for operation. In the TC1797, other combinations of SLSIS except 000_B and 001_B are reserved and must not be used.
STIP	8	rw	**Slave Transmit Idle State Polarity** This bit determines the logic level of the Slave Mode transmit signal MRST when the SSC slave select input signals are inactive (PISEL.SLSIS $\neq 000_B$). 0_B MRST = 0 when SSC is deselected in Slave Mode. 1_B MRST = 1 when SSC is deselected in Slave Mode.
0	[7:6], [31:9]	r	**Reserved** Read as 0; should be written with 0.

The operating modes of the SSC are controlled by the Control Register CON. This register contains control bits for mode and error check selection.

Note: Whenever operating mode parameters in the CON register are changed by software, no transfer should be in progress (STAT.BSY = 0) and the SSC should be disabled (CON.EN = 0) and afterwards enabled again (CON.EN = 1).

CON
Control Register **(10_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN	MS	0	A REN	BEN	PEN	REN	TEN	LB	PO	PH	HB	BM			
rw	rw	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw			

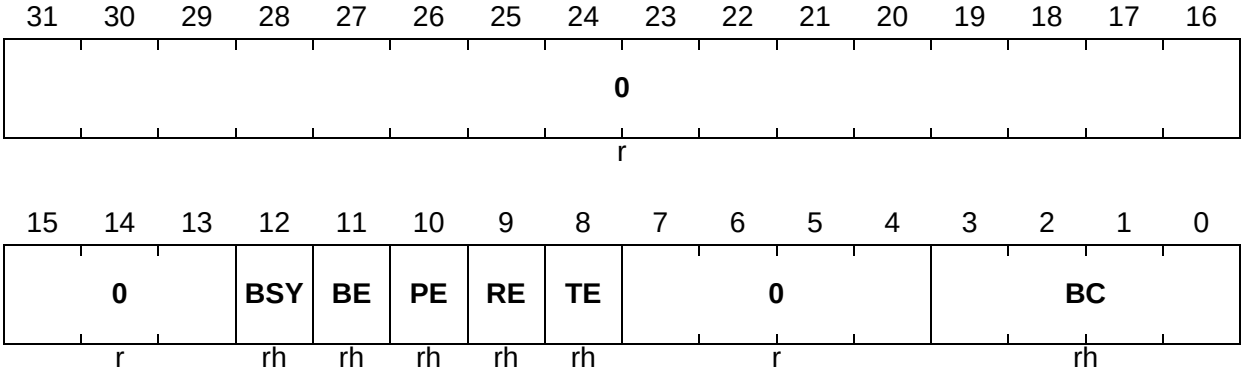
Field	Bits	Type	Description
BM	[3:0]	rw	**Data Width Selection** BM determines the number of data bits of the serial frame. 0000_B Reserved; do not use this combination. 0001_B Transfer Data Width is 2 bit. 0010_B Transfer Data Width is 3 bit. $\ldots_B$ … 1110_B Transfer Data Width is 15 bit. 1111_B Transfer Data Width is 16 bit.
HB	4	rw	**Heading Bit Control** 0_B Transmit/Receive LSB First 1_B Transmit/Receive MSB First
PH	5	rw	**Clock Phase Control** 0_B Shift transmit data on the leading clock edge, latch on trailing edge 1_B Latch receive data on leading clock edge, shift on trailing edge

Field	Bits	Type	Description
PO	6	rw	**Clock Polarity Control** 0_B Idle clock line is low, the leading clock edge is low-to-high transition 1_B Idle clock line is high, the leading clock edge is high-to-low transition
LB	7	rw	**Loop-Back Control** 0_B Normal output 1_B Receive input is connected to transmit output (Half-duplex Mode)
TEN	8	rw	**Transmit Error Enable** 0_B Ignore transmit errors 1_B Check transmit errors
REN	9	rw	**Receive Error Enable** 0_B Ignore receive errors 1_B Check receive errors
PEN	10	rw	**Phase Error Enable** 0_B Ignore phase errors 1_B Check phase errors
BEN	11	rw	**Baud Rate Error Enable** 0_B Ignore baud rate errors 1_B Check baud rate errors
AREN	12	rw	**Automatic Reset Enable** 0_B No additional action upon a baud rate error 1_B SSC is automatically reset on a baud rate error
MS	14	rw	**Master Select** 0_B Slave Mode. Operate on shift clock received via SCLK 1_B Master Mode. Generate shift clock and output it via SCLK The inverted state of this bit is available on module output line “M/S selected” (see **Figure 17-2**).
EN	15	rw	**Enable Bit** 0_B Transmission and reception are disabled. 1_B Transmission and reception are enabled. This bit is available as module output line “SSC enabled” (see **Figure 17-2**). Note that EN should only be cleared by software while no transfer is in progress (STAT.BSY = 0).

Field	Bits	Type	Description
0	13, [31:16]	r	**Reserved** Read as 0; should be written with 0.

The Status Register STAT contains status flags for error identification, the busy flag, and a bit field that indicates the current shift counter status.

STAT
Status Register **(28_H)** **Reset Value: 0000 0000$_H$**

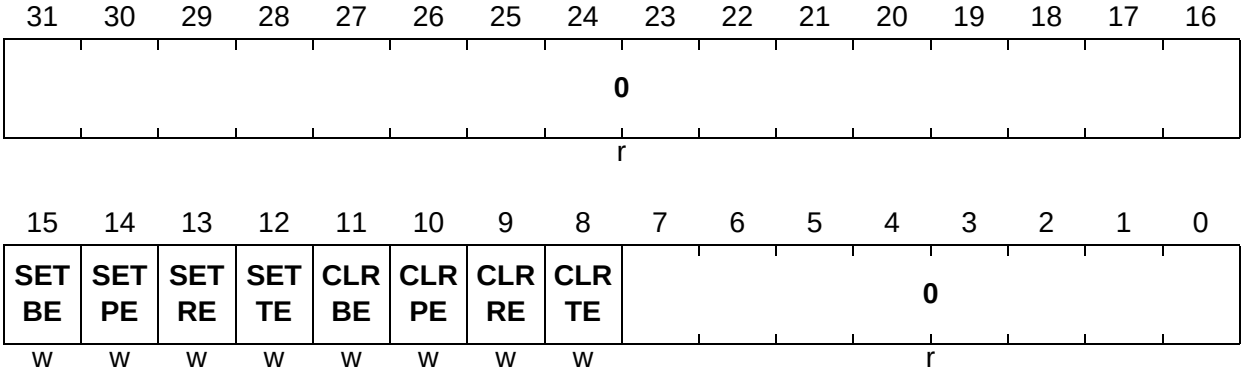


Field	Bits	Type	Description
BC	[3:0]	rh	**Bit Count Status** BC indicates the current status of the shift counter. The shift counter is updated with every shifted bit.
TE	8	rh	**Transmit Error Flag** 0_B No error 1_B Transfer starts with the slave's transmit buffer not being updated
RE	9	rh	**Receive Error Flag** 0_B No error 1_B Reception completed before the receive buffer was read
PE	10	rh	**Phase Error Flag** 0_B No error 1_B Received data changes during the sampling clock edge
BE	11	rh	**Baud Rate Error Flag** 0_B No error 1_B There is more than factor 2 or less than factor 0.5 between the slave's actual and the expected baud rate.
BSY	12	rh	**Busy Flag** BSY is set while a transfer is in progress.

Field	Bits	Type	Description
0	[7:4], [31:13]	r	**Reserved** Read as 0; should be written with 0.

The Error Flag Modification Register EFM is required for clearing or setting the four error flags which are located in register STAT.

EFM
Error Flag Modification Register (2C$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
CLRTE	8	w	**Clear Transmit Error Flag** 0_B No effect 1_B Bit STAT.TE is cleared. Bit is always read as 0.
CLRRE	9	w	**Clear Receive Error Flag** 0_B No effect 1_B Bit STAT.RE is cleared. Bit is always read as 0.
CLRPE	10	w	**Clear Phase Error Flag** 0_B No effect 1_B Bit STAT.PE is cleared. Bit is always read as 0.
CLRBE	11	w	**Clear Baud Rate Error Flag** 0_B No effect 1_B Bit STAT.BE is cleared. Bit is always read as 0.
SETTE	12	w	**Set Transmit Error Flag** 0_B No effect 1_B Bit STAT.TE is set. Bit is always read as 0.

Field	Bits	Type	Description
SETRE	13	w	**Set Receive Error Flag** 0_B No effect 1_B Bit STAT.RE is set. Bit is always read as 0.
SETPE	14	w	**Set Phase Error Flag** 0_B No effect 1_B Bit STAT.PE is set. Bit is always read as 0.
SETBE	15	w	**Set Baud Rate Error Flag** 0_B No effect 1_B Bit STAT.BE is set. Bit is always read as 0.
0	[7:0], [31:16]	r	**Reserved** Read as 0; should be written with 0.

Note: When the set and clear bits for an error flag are set at the same time during an EFM write operation (e.g SETPE = CLRPE = 1), the error flag in STAT is not affected.

The Slave Select Output Control Register controls the operation of the Chip Select Output Generation Unit.

SSOC

Slave Select Output Control Register (18$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

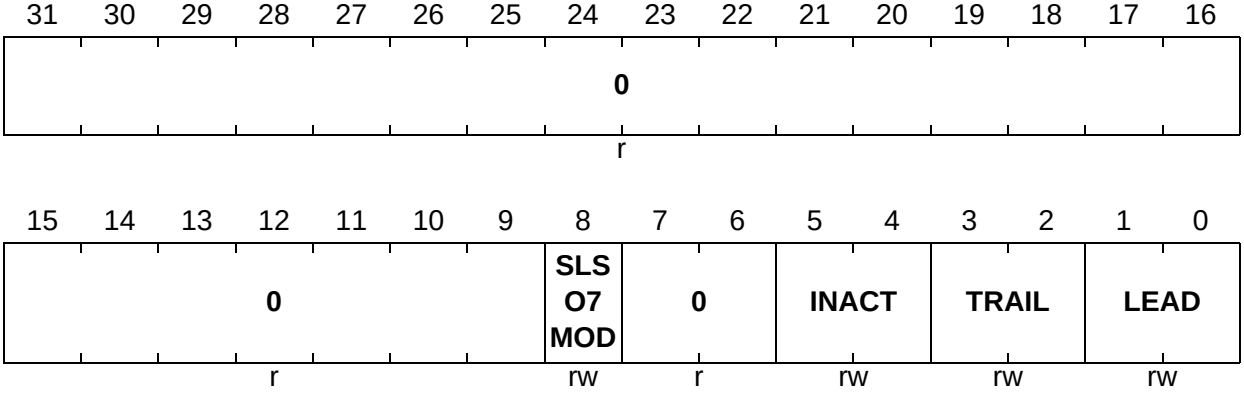
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OEN 7	OEN 6	OEN 5	OEN 4	OEN 3	OEN 2	OEN 1	OEN 0	AOL 7	AOL 6	AOL 5	AOL 4	AOL 3	AOL 2	AOL 1	AOL 0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
AOLn (n = 0-7)	n	rw	**Active Output Level** 0_B SLSOn is at low level during the chip select active time $t_{SLSOACT}$. The high level is the inactive level of SLSOn. 1_B SLSO line n is at high level during the chip select active time $t_{SLSOACT}$. The low level is the inactive level of SLSOn.
OENn (n = 0-7)	8 + n	rw	**Output n Enable Control** 0_B SLSOn output is disabled; SLSOn is always at inactive level as defined by AOLn. 1_B SLSOn output is enabled.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

Note: The SSOC register content is latched by each TB register write operation and remains latched during the consecutive serial transmission.

The Slave Select Output Timing Control Register controls the operation of the Slave Select Output Generation Unit.

SSOTC
Slave Select Output Timing Control Register
($1C_H$) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
LEAD	[1:0]	rw	**Slave Output Select Leading Delay** This bit field determines the number of leading delay clock cycles. A leading delay clock cycle is always a multiple of an SCLK shift clock period. 00_B Zero leading delay clock cycle selected[1)] 01_B One leading delay clock cycle selected 10_B Two leading delay clock cycles selected 11_B Three leading delay clock cycles selected
TRAIL	[3:2]	rw	**Slave Output Select Trailing Delay** This bit field determines the number of trailing delay clock cycles. A trailing delay clock cycle is always a multiple of an SCLK shift clock period. 00_B Zero trailing delay clock cycle selected[1)] 01_B One trailing delay clock cycle selected 10_B Two trailing delay clock cycles selected 11_B Three trailing delay clock cycles selected

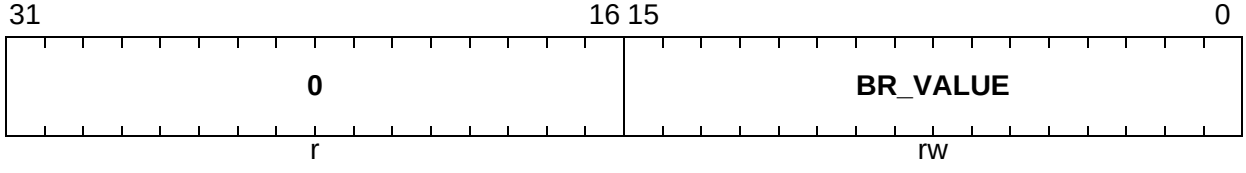
Field	Bits	Type	Description
INACT	[5:4]	rw	**Slave Output Select Inactive Delay** This bit field determines the number of inactive delay clock cycles. An inactive delay clock cycle is always a multiple of an SCLK shift clock period. 00_B Zero inactive delay clock cycle selected[1)] 01_B One inactive delay clock cycle selected 10_B Two inactive delay clock cycles selected 11_B Three inactive delay clock cycles selected
SLSO7MOD	8	rw	**SLSO7 Delayed Mode Selection** This bit selects the delayed mode for the SLSO7 slave select output. 0_B Normal mode selected for SLSO7 1_B Delayed mode selected for SLSO7
0	[7:6], [31:9]	r	**Reserved** Read as 0; should be written with 0.

1) For getting a best case timing with no timing delays (see **Figure 17-8**), this bit field value should be set when the SLSOn outputs are disabled (SSOC.OENn bits set to 0).

Note: The SSOTC register timing parameters LEAD, TRAIL, INACT, and SLSO7MOD are latched by each TB register write operation and remain latched during a consecutive serial transmission.

The Baud Rate Timer Reload Register BR contains the 16-bit reload value for the baud rate timer.

BR
Baud Rate Timer Reload Register (14_H) Reset Value: $0000\ 0000_H$

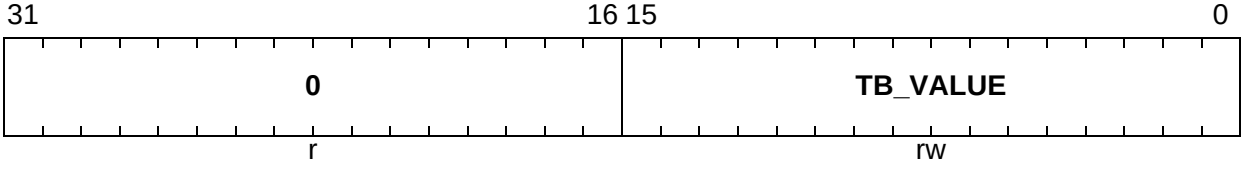


Field	Bits	Type	Description
BR_VALUE	[15:0]	rw	**Baud Rate Timer/Reload Register Value** Reading BR returns the 16-bit content of the baud rate timer. Writing BR loads the baud rate timer reload register with BR_VALUE.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

17.2.3 Data Registers

The Transmit Buffer Register TB contains the transmit data value. A TB write operation latches all timing parameters stored in register SSOTC.

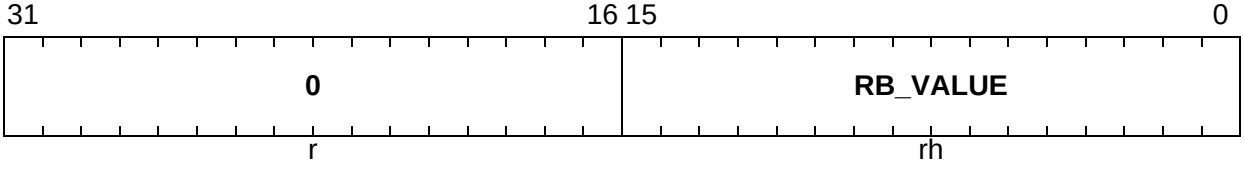
TB
Transmit Buffer Register **(20_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
TB_VALUE	[15:0]	rw	**Transmit Data Register Value** Register TB stores the data value to be transmitted TB_VALUE. Unused bits of TB_VALUE (as defined by CON.BM) are ignored during transmission.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

The Receive Buffer Register RB contains the receive data value.

RB
Receive Buffer Register **(24_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
RB_VALUE	[15:0]	rh	**Receive Data Register Value** Register RB contains the received data value RB_VALUE right aligned. Unused bits of RB_VALUE (as defined by CON.BM) will not be valid and should be ignored.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

17.3 SSC0/SSC1 Module Implementation

This section describes SSC0/SSC1 module interfaces with the clock control, port connections, interrupt control, and address decoding.

17.3.1 Module Identification Registers

The reset values of the SSCx_ID module identification registers are 0000 4511_H.

17.3.2 Interfaces of the SSC Modules

Figure 17-13 shows the TC1797-specific implementation details and interconnections of the SSC0/SSC1 modules.

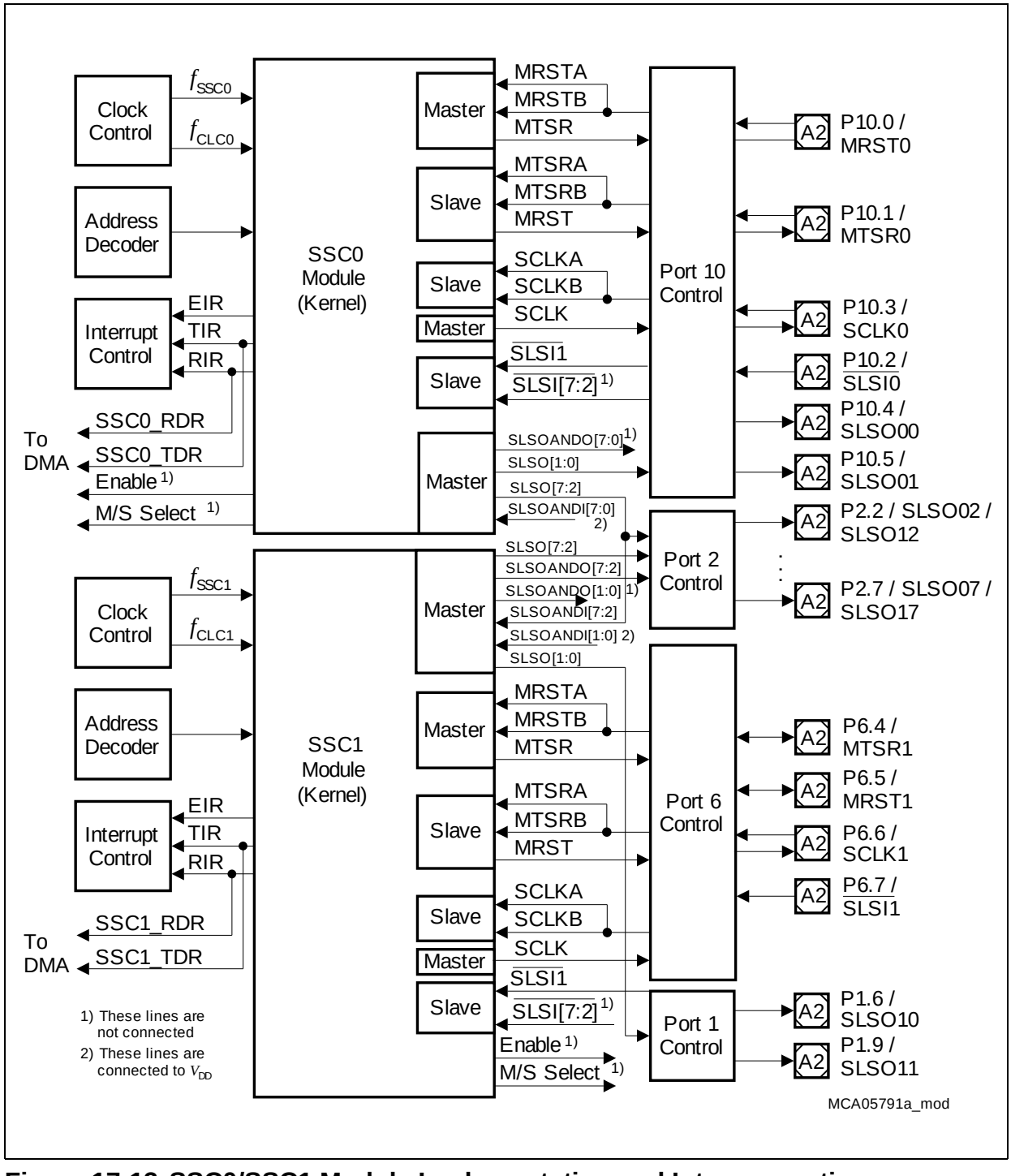


Figure 17-13 SSC0/SSC1 Module Implementation and Interconnections

Each of the SSC modules is supplied by a separate clock control, interrupt control, and address decoding logic. Two interrupt outputs can be used to generate DMA requests. The SSC0 I/O lines are connected to Port 10 and Port 2. The SSC1 I/O lines are connected to Port 1, Port 2, and Port 6. The SLSOn outputs of SSC0 and SSC1 are

wired as alternate function to six I/O lines of Port 2 and two I/O lines of each, Port 1 and Port 10.

17.3.3 On-Chip Connections

This section describes the on-chip connections of the SSC0/SSC1 modules.

DMA Requests

The DMA request lines of the SSC0/SSC1 modules become active whenever the related interrupt line is activated. The DMA request lines are connected to the DMA controller as shown in **Table 17-4**.

Table 17-4 DMA Request Lines of SSC0/SSC1

Module	SSC Interrupt Request Line	DMA Request Line	Description
SSC0	RIR	SSC0_RDR	DMA Channel 00 Request Input 4
			DMA Channel 10 Request Input 4
			DMA Channel 06 Request Input 4
			DMA Channel 16 Request Input 4
	TIR	SSC0_TDR	DMA Channel 02 Request Input 4
			DMA Channel 12 Request Input 4
			DMA Channel 04 Request Input 4
			DMA Channel 14 Request Input 4
SSC1	RIR	SSC1_RDR	DMA Channel 01 Request Input 4
			DMA Channel 11 Request Input 4
			DMA Channel 07 Request Input 4
			DMA Channel 17 Request Input 4
	TIR	SSC1_TDR	DMA Channel 03 Request Input 4
			DMA Channel 13 Request Input 4
			DMA Channel 05 Request Input 4
			DMA Channel 15 Request Input 4

17.3.4 SSC0/SSC1 Module Related External Registers

Figure 17-14 summarizes the module-related external registers which are required for SSC0/SSC1 programming (see also **Figure 17-12** for the module kernel specific registers).

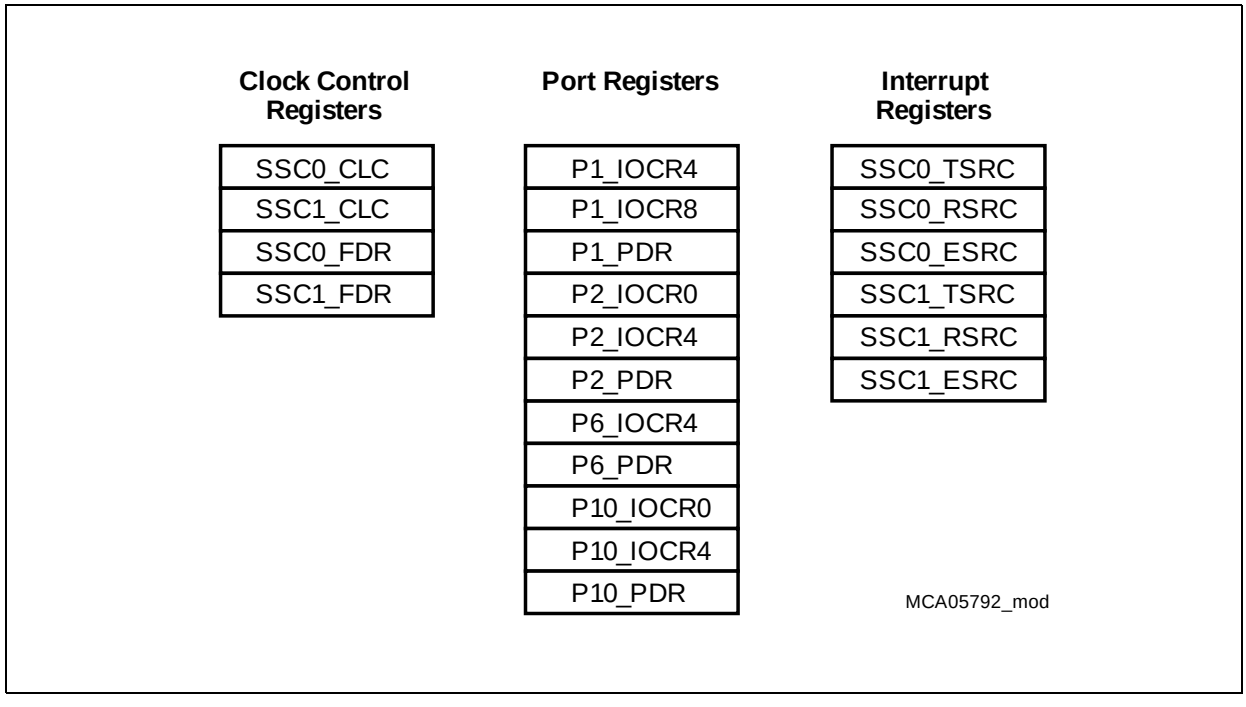


Figure 17-14 SSC0/SSC1 Implementation-specific Special Function Registers

17.3.4.1 Clock Control

Each SSC module has two clock signals:

- f_{CLC0} and f_{CLC1}
  This is the module clock that is used inside the SSC kernel for control purposes such as clocking of control logic and register operations. The frequency of f_{CLC0} and f_{CLC1} is always identical to the system clock frequency f_{SYS} (= f_{FPI} in TC1797). The clock control registers SSC0_CLC and SSC1_CLC make it possible to enable/disable f_{CLC0} and f_{CLC1} under certain conditions.
- f_{SSC0} and f_{SSC1}
  This clock is the module clock that is used in the SSC as input clock of the baud rate generator, which finally determines the baud rate of the serial data. The fractional divider registers SSC0_FDR and SSC1_FDR control the frequency of f_{SSC0} and f_{SSC1} and make it possible to enable/disable it independently of f_{CLC0} and f_{CLC1}.
  The Baud Rate Timer Reload Register SSC0_BR and SSC1_BR define serial data baud rate dependent from the frequency of f_{SSC0} and f_{SSC1}.

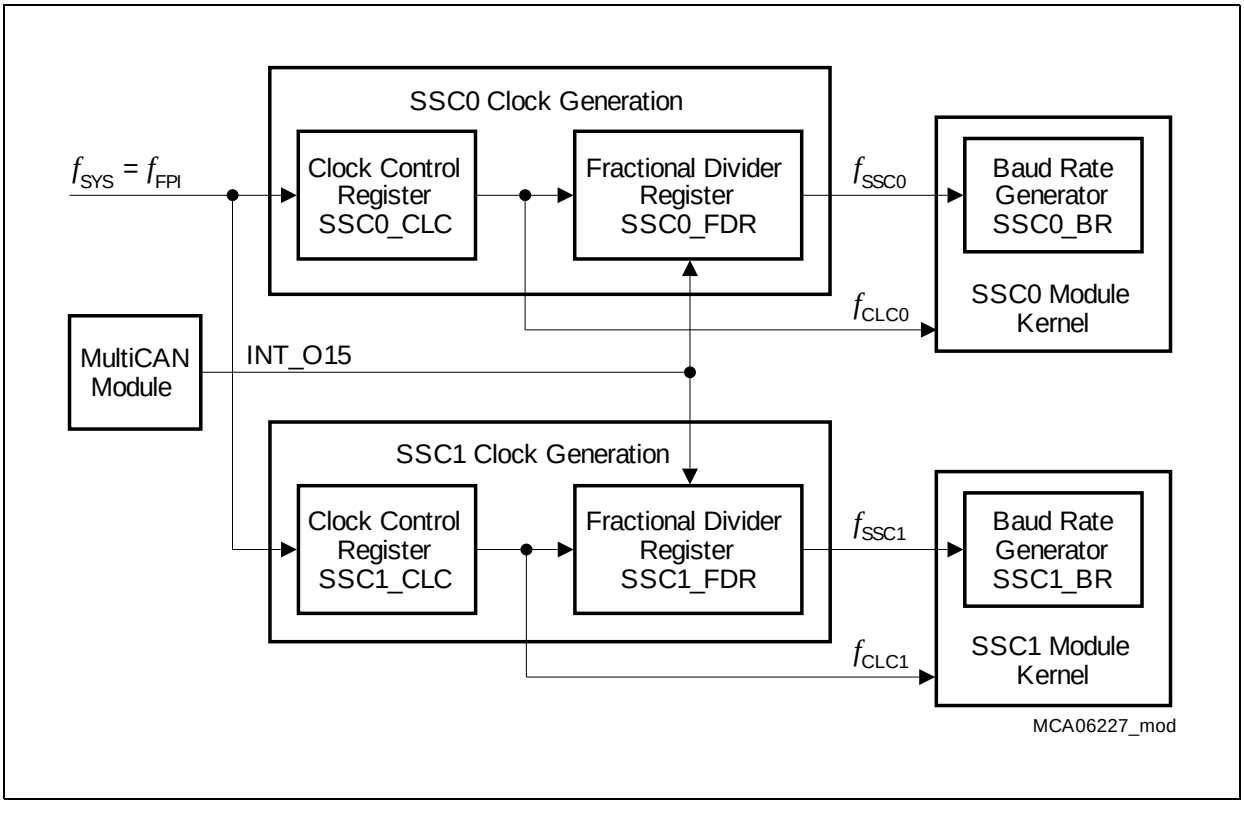


Figure 17-15 SSC Clock Generation

Output signal CAN_INT_O15 of the MultiCAN module can be used for external clock enable control of the fractional divider.

The following formulas define the frequency of f_{SSC0} or f_{SSC1}

$$f_{SSCx} = f_{SYS} \times \frac{1}{n} \text{ with n = 1024 - FDR.STEP} \quad (17.2)$$

$$f_{SSCx} = f_{SYS} \times \frac{n}{1024} \text{ with n = 0-1023} \quad (17.3)$$

Note: In SSC Master Mode, the maximum shift clock frequency is $f_{SSCx}/2$. In SSC Slave Mode, the maximum shift clock frequency is $f_{SSCx}/4$.

Combined with the formulas of the baud rate generator (see **Page 17-12**) and the fractional divider, the resulting serial data baud rate is defined by:

$$\text{Baud rate}_{SSC} = \frac{f_{SYS}}{2 \times (\text{BR.BR\_VALUE} + 1) \times (1024 - \text{FDR.STEP})} \quad (17.4)$$

$$\text{Baud rate}_{SSC} = \frac{f_{SYS} \times \text{FDR.STEP}}{2 \times (\text{BR.BR\_VALUE} + 1) \times 1024} \text{ with FDR.STEP = 0-1023} \quad (17.5)$$

*Note: **Equation (17.2)** and **Equation (17.4)** apply to normal divider mode of the fractional divider (FDR.DM = 01_B). **Equation (17.3)** and **Equation (17.5)** apply to fractional divider mode (FDR.DM = 10_B).*

Clock Control Register

The Clock Control Registers SSC0_CLC and SSC1_CLC make it possible to control (enable/disable) the clock signals f_{CLC0} and f_{CLC1} under certain conditions. Each SSC has its own clock control register.

SSC0_CLC
Clock Control Register **(00_H)** **Reset Value: 0000 0003$_H$**
SSC1_CLC
SSC1 Clock Control Register **(00_H)** **Reset Value: 0000 0003$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0										FSOE	SBWE	EDIS	SPEN	DISS	DISR
r										rw	w	rw	rw	r	rw

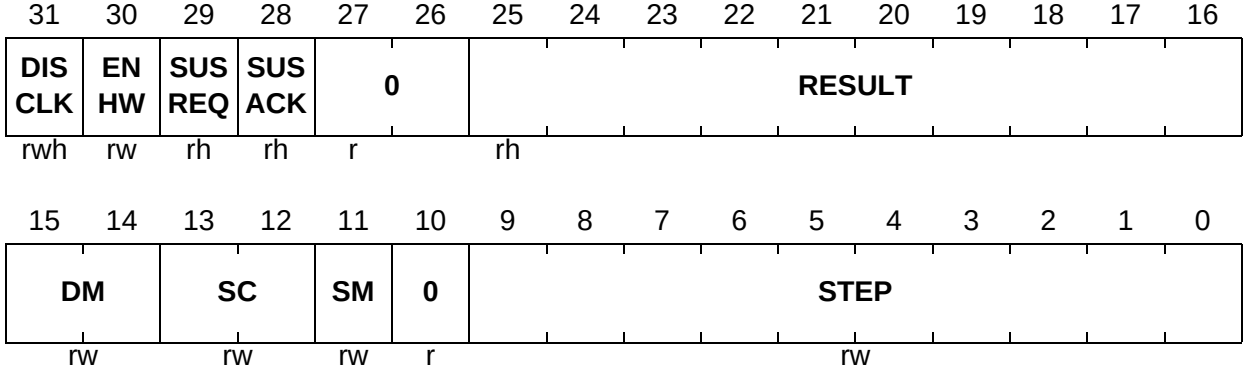
Field	Bits	Type	Description
DISR	0	rw	**Module Disable Request Bit** Used for enable/disable control of the module.
DISS	1	r	**Module Disable Status Bit** Bit indicates the current status of the module.
SPEN	2	rw	**Module Suspend Enable for OCDS** Used to enable the suspend mode.
EDIS	3	rw	**Sleep Mode Enable Control** Used to control module's sleep mode.
SBWE	4	w	**Module Suspend Bit Write Enable for OCDS** Determines whether SPEN and FSOE are write-protected.
FSOE	5	rw	**Fast Switch Off Enable** Used to switch off fast clock in Suspend Mode.
0	[31:6]	r	**Reserved** Read as 0; should be written with 0.

Note: After a hardware reset operation, the f_{CLCx} clocks are disabled, and therefore the SSC modules are disabled (DISS set) also.

Fractional Divider Register

The Fractional Divider Registers SSC0_FDR and SSC1_FDR control the clock rate of the shift clock f_{SSC0} and f_{SSC1}. Each SSC has its own fractional divider register.

SSC0_FDR
SSC0 Fractional Divider Register (0C$_H$) Reset Value: 1000 0000$_H$
SSC1_FDR
SSC1 Fractional Divider Register (0C$_H$) Reset Value: 1000 0000$_H$



Field	Bits	Type	Description
STEP	[9:0]	rw	**Step Value** Reload or addition value for RESULT.
SM	11	rw	**Suspend Mode** SM selects between granted or immediate suspend mode.
SC	[13:12]	rw	**Suspend Control** This bit field determines the behavior of the fractional divider in suspend mode.
DM	[15:14]	rw	**Divider Mode** This bit field selects normal divider mode, fractional divider mode, and off-state.
RESULT	[25:16]	rh	**Result Value** Bit field for the addition result.
SUSACK	28	rh	**Suspend Mode Acknowledge** Indicates state of SPNDACK signal.
SUSREQ	29	rh	**Suspend Mode Request** Indicates state of SPND signal.

Field	Bits	Type	Description
ENHW	30	rw	**Enable Hardware Clock Control** Controls operation of ECEN input and DISCLK bit.
DISCLK	31	rwh	**Disable Clock** Hardware-controlled disable for f_{OUT} signal.
0	10, [27:26]	r	**Reserved** Read as 0; should be written with 0.

17.3.4.2 Port Control

Note: The SSCs in the TC1797 do not directly control the I/O functionality of pins. Therefore, control bits CON.EN and CON.MS have no functionality.

The interconnections between the SSC modules and the I/O lines/pins are controlled by software in the port logics. The **SSC0/SSC1** I/O functionality must be selected by the following port control operations (additionally to the PISEL programming):

- Input/output function selection (IOCR registers)
- Pad driver characteristics selection for the outputs (PDR registers)

The SSC1 port input/output control registers contain the bit fields that select the digital output and input driver characteristics such as pull-up/down devices, port direction (input/output), open-drain, and alternate output selections. The master/slave I/O lines and the slave select input of the SSC1 module are class A2 GPIO pins that are controlled by a port input/output control register of Port 6.

Each of six **SLSOn** outputs of SSC0 and SSC1 (SLSO[7:2]) are wired to six class A2 GPIO pins of Port 2 (P2.[7:2]) and can be selected as alternate function via the P2_IOCR0 and P2_IOCR4 registers. Additionally, the logical AND combination outputs SLSOANDOx with identical index “x” are connected to P2.[7:2] and can be selected as alternate function (see **Figure 17-16**).
The outputs SLSO [1:0] of SSC0 are connected as alternate function to the two port lines P10.4 and P10.5. The outputs SLSO [1:0] of SSC1 are connected as alternate function to the two port lines P1.6 and P1.9.

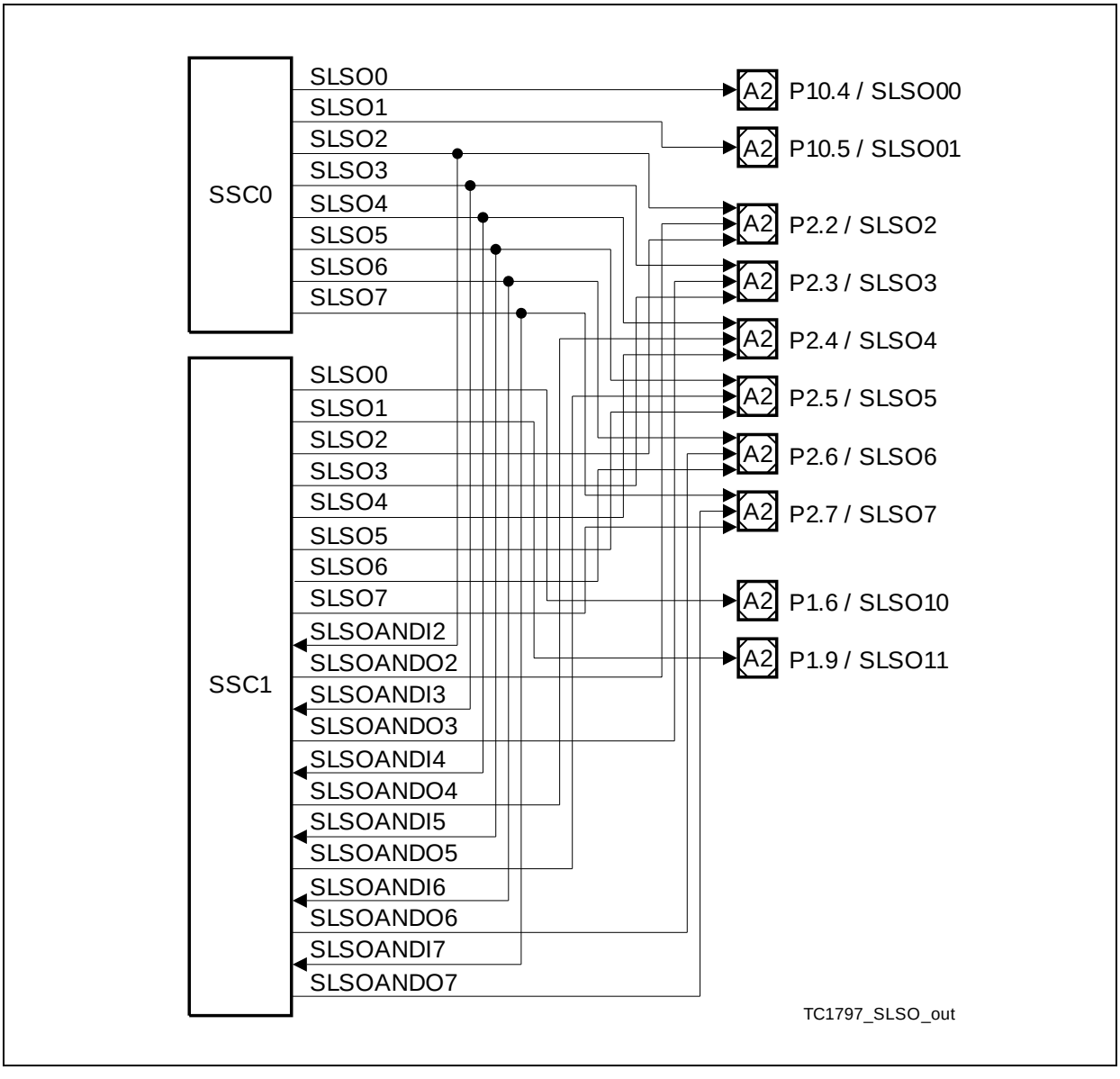


Figure 17-16 SLSO Output Selection

Table 17-5 shows how bits and bit fields must be programmed for the required I/O functionality of the SSC I/O lines.

Table 17-5 SSC0 and SSC1 I/O Line Selection and Setup

Module	Port Lines	Input/Output Control Register Bits	I/O
SSC0	P10.1 / MTSR0	P10_IOCR0.PC1 = $0XXX_B$	Input
		P10_IOCR0.PC1 = $1X01_B$	Output
	P10.0 / MRST0	P10_IOCR0.PC0 = $0XXX_B$	Input
		P10_IOCR0.PC0 = $1X01_B$	Output
	P10.3 / SCLK0	P10_IOCR0.PC3 = $0XXX_B$	Input
		P10_IOCR0.PC3 = $1X01_B$	Output
	P10.2 / SLSI0	P10_IOCR0.PC4 = $0XXX_B$	Input
SSC1	P6.4 / MTSR1	P6_IOCR4.PC4 = $0XXX_B$	Input
		P6_IOCR4.PC4 = $1X01_B$	Output
	P6.5 / MRST1	P6_IOCR4.PC5 = $0XXX_B$	Input
		P6_IOCR4.PC5 = $1X01_B$	Output
	P6.6 / SCLK1	P6_IOCR4.PC6 = $0XXX_B$	Input
		P6_IOCR4.PC6 = $1X01_B$	Output
	P6.7 / $\overline{\text{SLSI1}}$	P6_IOCR4.PC7 = $0XXX_B$	Input
Slave Select Outputs			
SSC0	P10.4 / SLSO00	P10_IOCR4.PC4 = $1X01_B$	Output
	P10.5 / SLSO01	P10_IOCR4.PC5 = $1X01_B$	Output
SSC0	P2.2 / SLSO2	P2_IOCR0.PC2 = $1X01_B$	Output
SSC1		P2_IOCR0.PC2 = $1X10_B$	
SSC0 & SSC1		P2_IOCR0.PC2 = $1X11_B$	
SSC0	P2.3 / SLSO3	P2_IOCR0.PC3 = $1X01_B$	Output
SSC1		P2_IOCR0.PC3 = $1X10_B$	
SSC0 & SSC1		P2_IOCR0.PC3 = $1X11_B$	
SSC0	P2.4 / SLSO4	P2_IOCR4.PC4 = $1X01_B$	Output
SSC1		P2_IOCR4.PC4 = $1X10_B$	
SSC0 & SSC1		P2_IOCR4.PC4 = $1X11_B$	
SSC0	P2.5 / SLSO5	P2_IOCR4.PC5 = $1X01_B$	Output
SSC1		P2_IOCR4.PC5 = $1X10_B$	
SSC0 & SSC1		P2_IOCR4.PC5 = $1X11_B$	

Table 17-5 SSC0 and SSC1 I/O Line Selection and Setup (cont'd)

Module	Port Lines	Input/Output Control Register Bits	I/O
SSC0	P2.6 / SLSO6	P2_IOCR4.PC6 = $1X01_B$	Output
SSC1		P2_IOCR4.PC6 = $1X10_B$	
SSC0 & SSC1		P2_IOCR4.PC6 = $1X11_B$	
SSC0	P2.7 / SLSO7	P2_IOCR4.PC7 = $1X01_B$	Output
SSC1		P2_IOCR4.PC7 = $1X10_B$	
SSC0 & SSC1		P2_IOCR4.PC7 = $1X11_B$	
SSC1	P1.6 / SLSO10	P1_IOCR4.PC6 = $1X10_B$	Output
	P1.9 / SLSO11	P1_IOCR8.PC9 = $1X10_B$	Output

17.3.4.3 Interrupt Control Registers

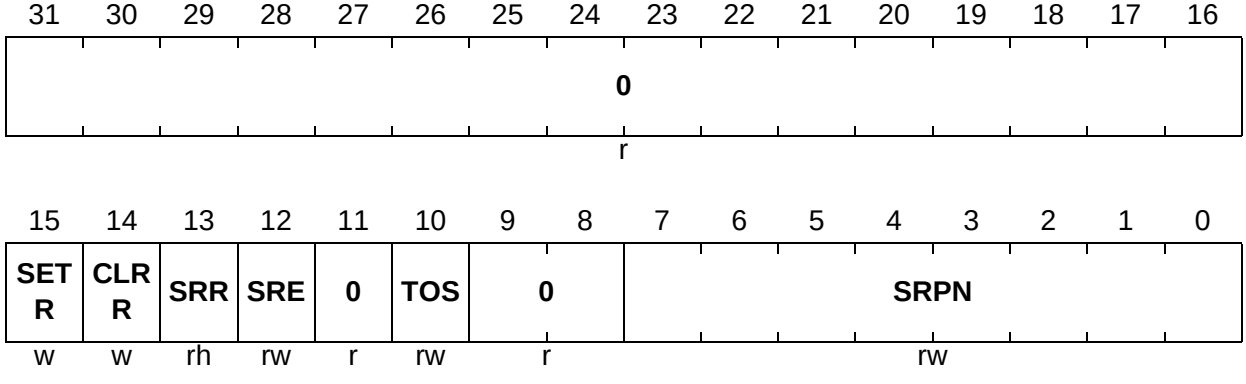
The 2 × 3 interrupts of the SSC0 and SSC1 modules are controlled by the following service request control registers:

- SSC0_TSRC, SSC1_TSRC controls the transmit interrupts
- SSC0_RSRC, SSC1_RSRC controls the receive interrupts
- SSC0_ESRC, SSC1_ESRC controls the error interrupts

TSRC
Transmit Interrupt Service Request Control Register
($F4_H$) **Reset Value: 0000 0000_H**

RSRC
Receive Interrupt Service Request Control Register
($F8_H$) **Reset Value: 0000 0000_H**

ESRC
Error Interrupt Service Request Control Register
(FC_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
SRPN	[7:0]	rw	**Service Request Priority Number**
TOS	10	rw	**Type of Service Control**
SRE	12	rw	**Service Request Enable**
SRR	13	rh	**Service Request Flag**
CLRR	14	w	**Request Clear Bit**
SETR	15	w	**Request Set Bit**
0	[9:8], 11, [31:16]	r	**Reserved** Read as 0; should be written with 0.

17.3.5 SSC0/SSC1 Address Map

An absolute register address is given by the offset address of the register (given in **Table 17-3**) plus the module base address (given in **Table 17-2**).

Table 17-6 Address Map of SSC0/SSC1

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
Synchronous Serial Interface 0 (SSC0)					
SSC0_ CLC	SSC0 Clock Control Register	F010 0100$_H$	U, SV	SV, E	0000 0003$_H$
SSC0_ PISEL	SSC0 Port Input Select Register	F010 0104$_H$	U, SV	U, SV	0000 0000$_H$
SSC0_ ID	SSC0 Module Identification Register	F010 0108$_H$	U, SV	BE	0000 45XX$_H$
SSC0_ FDR	SSC0 Fractional Divider Register	F010 010C$_H$	U, SV	SV, E	1000 0000$_H$
SSC0_CON	SSC0 Control Register	F010 0110$_H$	U, SV	U, SV	0000 0000$_H$
SSC0_ BR	SSC0 Baud Rate Timer Reload Register	F010 0114$_H$	U, SV	U, SV	0000 0000$_H$
SSC0_ SSOC	SSC0 Slave Select Output Control Register	F010 0118$_H$	U, SV	U, SV	0000 0000$_H$
SSC0_ SSOTC	SSC0 Slave Select Output Timing Control Register	F010 011C$_H$	U, SV	U, SV	0000 0000$_H$
SSC0_ TB	SSC0 Transmit Buffer Register	F010 0120$_H$	U, SV	U, SV	0000 0000$_H$
SSC0_ RB	SSC0 Receive Buffer Register	F010 0124$_H$	U, SV	U, SV	0000 0000$_H$
SSC0_ STAT	SSC0 Status Register	F010 0128$_H$	U, SV	U, SV	0000 0000$_H$
SSC0_ EFM	SSC0 Error Flag Modification Register	F010 012C$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 0130$_H$ - F010 01F0$_H$	BE	BE	–
SSC0_ TSRC	SSC0 Transmit Interrupt Service Req. Control Reg.	F010 01F4$_H$	U, SV	SV	0000 0000$_H$

Table 17-6 Address Map of SSC0/SSC1 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
SSC0_ RSRC	SSC0 Receive Interrupt Service Req. Control Reg.	F010 01F8$_H$	U, SV	SV	0000 0000$_H$
SSC0_ ESRC	SSC0 Error Interrupt Service Req. Control Reg.	F010 01FC$_H$	U, SV	SV	0000 0000$_H$
Synchronous Serial Interface 1 (SSC1)					
SSC1_CLC	SSC1 Clock Control Register	F010 0200$_H$	U, SV	SV, E	0000 0003$_H$
SSC1_ PISEL	SSC1 Port Input Select Register	F010 0204$_H$	U, SV	U, SV	0000 0000$_H$
SSC1_ ID	SSC1 Module Identification Register	F010 0208$_H$	U, SV	BE	0000 45XX$_H$
SSC1_ FDR	SSC1 Fractional Divider Register	F010 020C$_H$	U, SV	SV, E	1000 0000$_H$
SSC1_CON	SSC1 Control Register	F010 0210$_H$	U, SV	U, SV	0000 0000$_H$
SSC1_ BR	SSC1 Baud Rate Timer Reload Register	F010 0214$_H$	U, SV	U, SV	0000 0000$_H$
SSC1_ SSOC	SSC1 Slave Select Output Control Register	F010 0218$_H$	U, SV	U, SV	0000 0000$_H$
SSC1_ SSOTC	SSC1 Slave Select Output Timing Control Register	F010 021C$_H$	U, SV	U, SV	0000 0000$_H$
SSC1_ TB	SSC1 Transmit Buffer Register	F010 0220$_H$	U, SV	U, SV	0000 0000$_H$
SSC1_ RB	SSC1 Receive Buffer Register	F010 0224$_H$	U, SV	U, SV	0000 0000$_H$
SSC1_STAT	SSC1 Status Register	F010 0228$_H$	U, SV	U, SV	0000 0000$_H$
SSC1_ EFM	SSC1 Error Flag Modification Register	F010 022C$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 0230$_H$ - F010 02F0$_H$	BE	BE	–
SSC1_ TSRC	SSC1 Transmit Interrupt Service Req. Control Reg.	F010 02F4$_H$	U, SV	SV	0000 0000$_H$

Table 17-6 Address Map of SSC0/SSC1 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
SSC1_ RSRC	SSC1 Receive Interrupt Service Req. Control Reg.	F010 02F8$_H$	U, SV	SV	0000 0000$_H$
SSC1_ ESRC	SSC1 Error Interrupt Service Req. Control Reg.	F010 02FC$_H$	U, SV	SV	0000 0000$_H$

18 Micro Second Channel (MSC)

This chapter describes the Micro Second Channel Interface s, MSC0 and MSC1, of the TC1797. It contains the following sections:

- Functional description of the MSC kernel (see **Page 18-3**)
- MSC kernel register descriptions (see **Page 18-36**)
- TC1797 implementation-specific details and registers of the MSC module (port connections and control, interrupt control, address decoding, and clock control, see **Page 18-62**)

*Note: The MSC kernel register names described in **Section 18.2** are also referenced in the TC1797 User´s Manual by the module name prefix “MSC0_” for the MSC0 module and by “MSC1_” for the MSC1 module.*

MSC Applications

The MSC is a serial interface that is especially designed to connect external power devices to the TC1797. The serial data transmission capability minimizes the number of pins required to connect such external power devices. Parallel data information (coming from the timer units) or command information is sent out to the power device via a high-speed synchronous serial data stream (downstream channel). The MSC receives data and status back from the power device via a low-speed asynchronous serial data stream (upstream channel).

Figure 18-1 shows a typical TC1797 application in which an MSC interface controls two power devices. Output data is provided by the GPTA module.

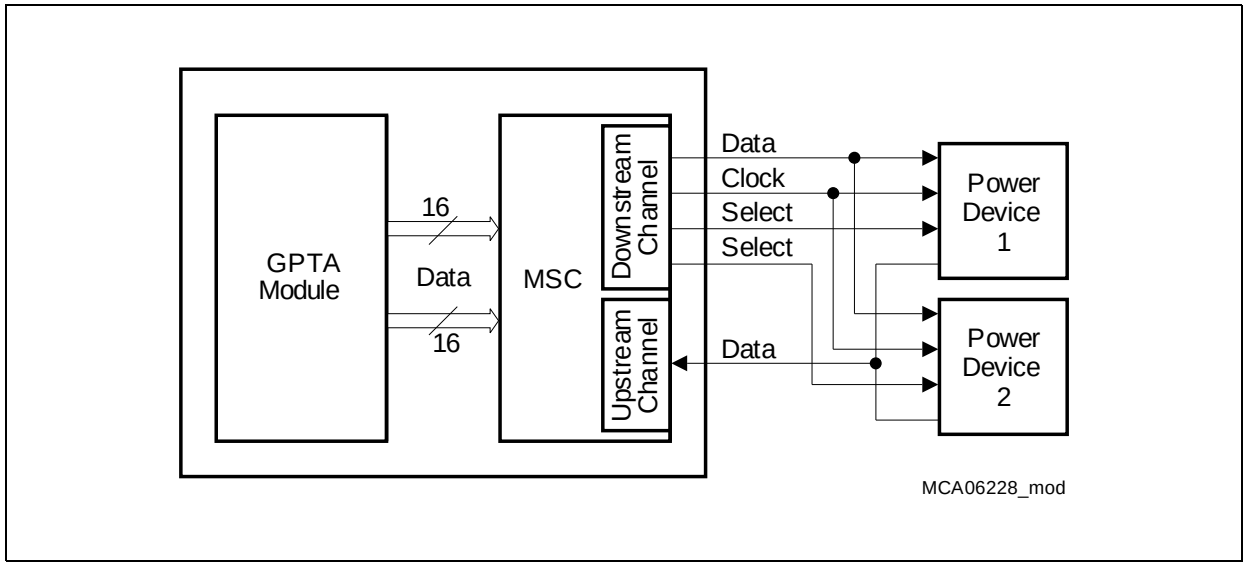


Figure 18-1 MSC to External Power Device Connection

Some applications are:

- Control of the external power switching unit via the downstream channel
- Receiving information back from power switching unit
- Serial connections of the TC1797 to other peripheral devices

18.1 MSC Kernel Description

This section describes the functionality of the MSC kernel.

18.1.1 Overview

The MSC interface provides a serial communication link typically used to connect power switches or other peripheral devices. The serial communication link includes a fast synchronous downstream channel and a slow asynchronous upstream channel. **Figure 18-2** shows a global view of the MSC interface signals.

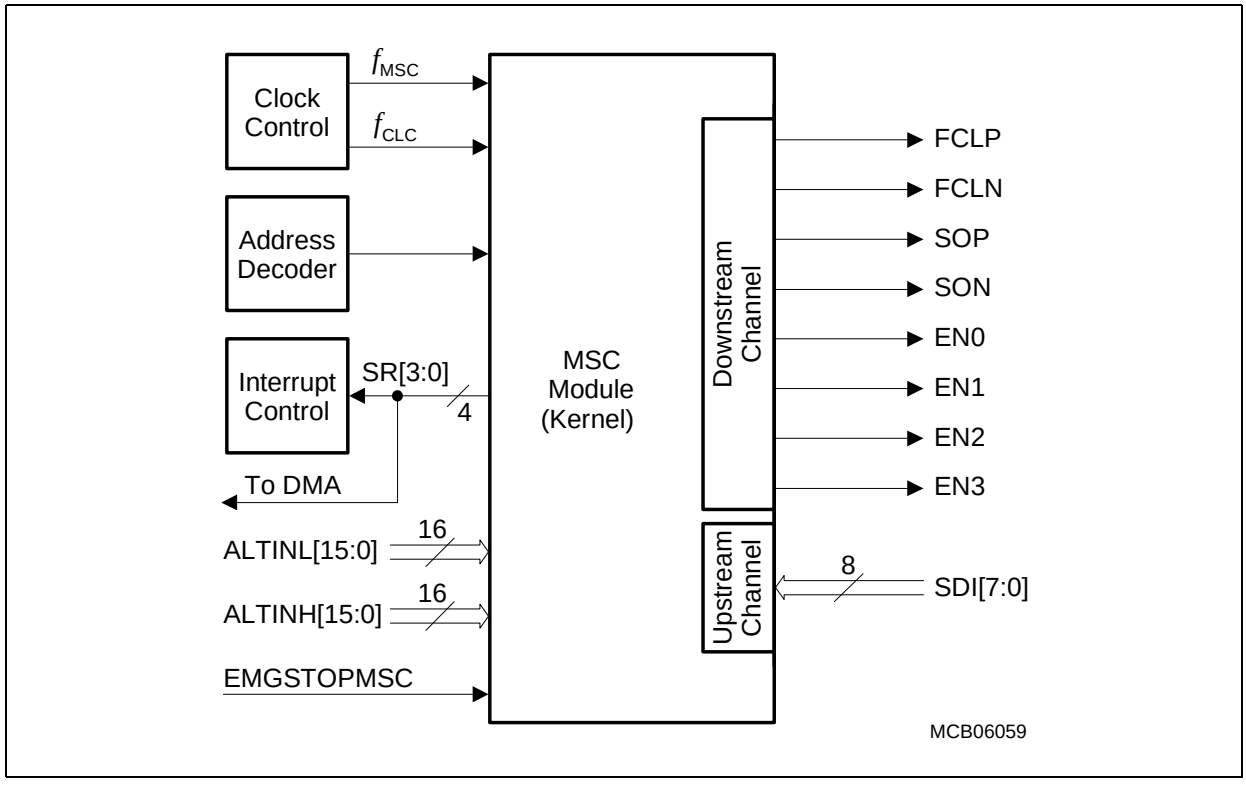


Figure 18-2 General Block Diagram of the MSC Interface

The downstream and upstream channels of the MSC module communicate with the external world via nine I/O lines. Eight output lines are required for the serial communication of the downstream channel (clock, data, and enable signals). One out of eight input lines SDI[7:0] is used as serial data input signal for the upstream channel. The source of the serial data to be transmitted by the downstream channel can be MSC register contents or data that is provided at the ALTINL/ALTINH input lines. These input lines are typically connected to other on-chip peripheral units (for example with a timer unit like the GPTA). An emergency stop input signal makes it possible to set bits of the serial data stream to dedicated values in emergency case.

Clock control, address decoding, and interrupt service request control are managed outside the MSC module kernel. Service request outputs are able to trigger an interrupt or a DMA request.

Features

- Fast synchronous serial interface to connect power switches in particular, or other peripheral devices via serial buses
- High-speed synchronous serial transmission on downstream channel
  - Serial output clock frequency: $f_{FCL} = f_{MSC}/2$ (f_{MSCmax} = 90 MHz)
  - Fractional clock divider for precise frequency control of serial clock f_{MSC}
  - Command, data, and passive frame types
  - Start of serial frame: Software-controlled, timer-controlled, or free-running
  - Transmission with or without SEL bit
  - Flexible chip select generation indicates status during serial frame transmission
  - Emergency stop without CPU intervention
- Low-speed asynchronous serial reception on upstream channel
  - Baud rate: f_{MSC} divided by 4, 8, 16, 32, 64, 128, or 256 (f_{MSCmax} = 90 MHz)
  - Standard asynchronous serial frames
  - Programmable upstream data frame length (16 or 12 bits)
  - Parity error checker
  - 8-to-1 input multiplexer for SDI lines
  - Built-in spike filter on SDI lines
- Selectable pin types of downstream channel interface:
  four LVDS differential output drivers or four digital GPIO pins

18.1.2 Downstream Channel

The downstream channel performs a high-speed synchronous serial transmission of data to external devices. Its 32-bit shift register is divided into two 16-bit parts, SRL and SRH. Each bit of SRL and SRH can be selected to be delivered by the downstream data register DD, by the Downstream Command Register DC, or by two 16-bit wide input signal buses ALTINL and ALTINH.

Figure 18-3 is a diagram of the MSC downstream channel.

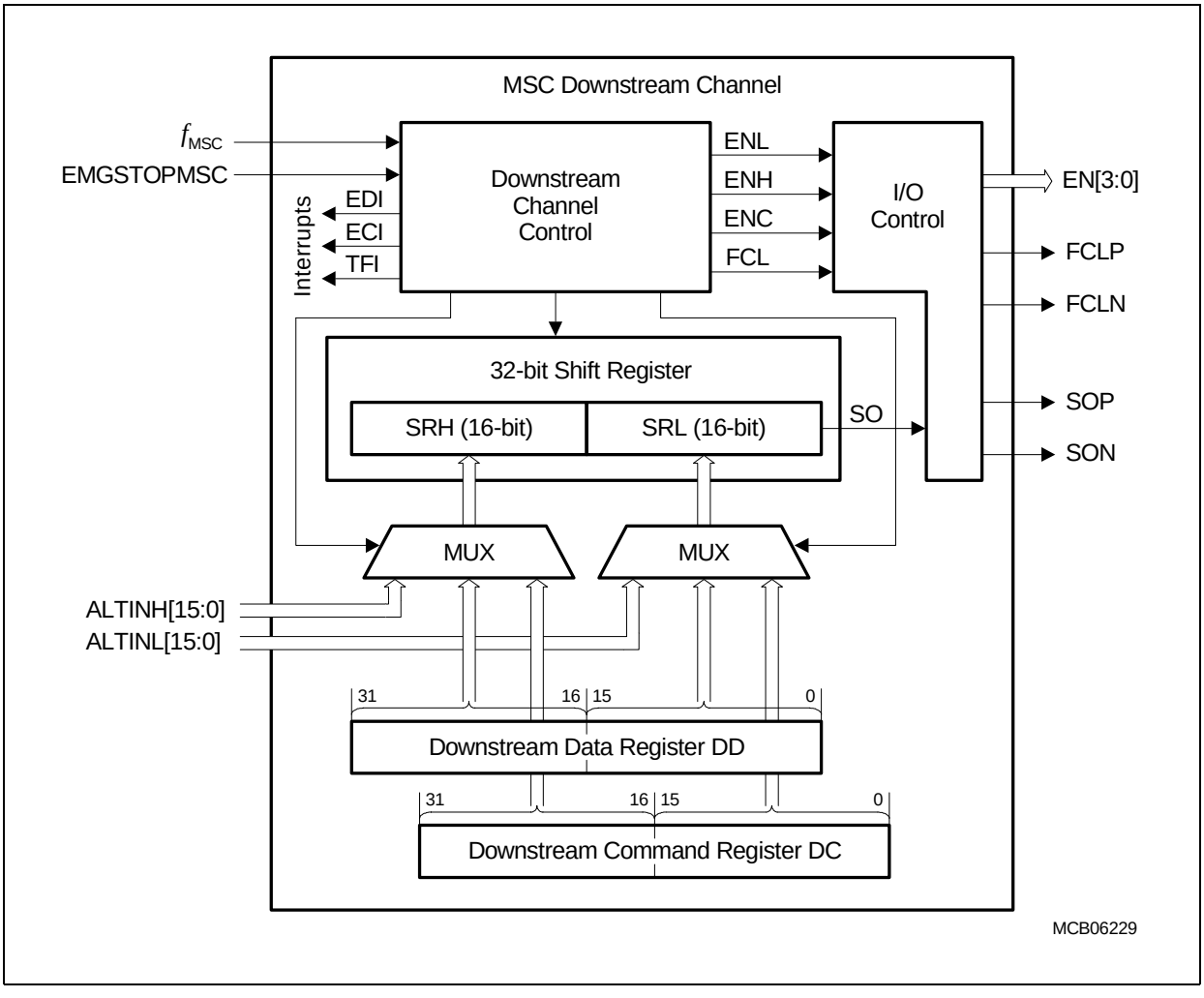


Figure 18-3 Downstream Channel Block Diagram

The enable signals ENL, ENH, and ENC indicate certain phases of the serial transmission in relation to the serial clock FCL. In the I/O control logic, these signals can be combined to four enable/select outputs EN[3:0]. For supporting differential output drivers, the serial clock output FCL and the serial data output SO are available in both polarities, indicated by the signal name suffix “P” and “N”.

The emergency stop input line EMGSTOPMSC is used to indicate an emergency stop condition of a power device. In emergency case, shift register bits can be loaded bit-wise from the downstream data register instead from the ALTINL and ALTINH buses.

18.1.2.1 Frame Formats and Definitions

This section describes the frame formats and definitions of the MSC.

Basic Definitions

Figure 18-4 shows the layout and definitions of a downstream frame. A downstream frame is composed of an active phase and a passive phase. During the active phase, data transmission takes place and during the passive phase no data is transmitted at SO. The active phase is split into two parts: The SRL active phase in which the content of the shift register low part SRL is transmitted, and the SRH active phase in which the content of the shift register high part SRH is transmitted. At the beginning of the SRL and SRH active phase, a selection bit (SELL) can be optionally inserted into the serial data stream. In the frame shown in **Figure 18-4**, SELL is generated at the beginning of the SRL active phase (not for the SRH active phase). The least significant bits of SRL and SRH are sent out first.

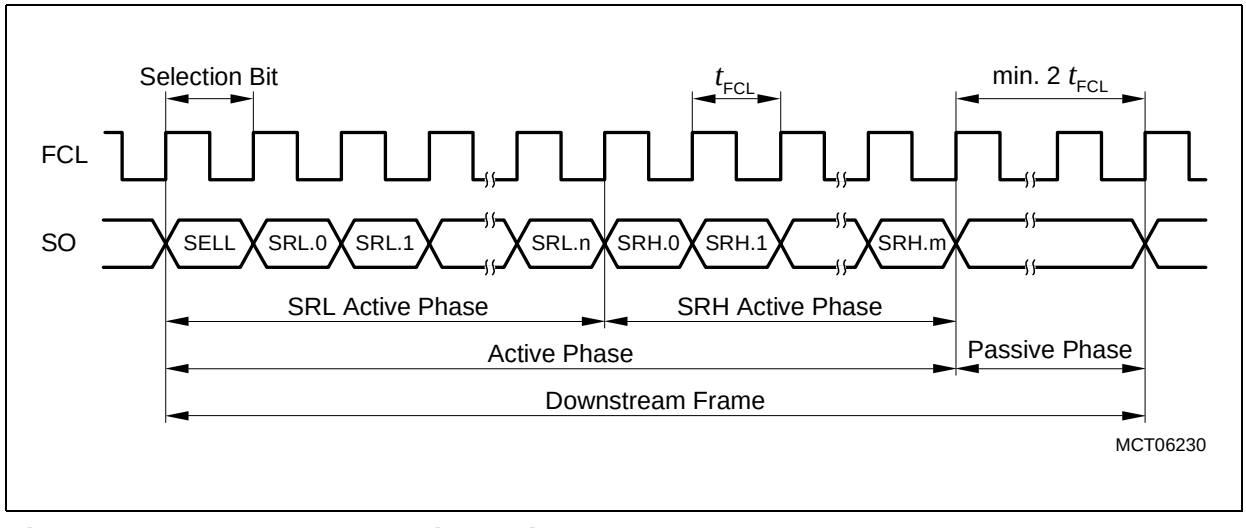


Figure 18-4 Downstream Channel Frame

The MSC downstream channel uses three types of frame formats for operation:

- Command frames, indicated by SELL = 1
- Data frames, indicated by SELL = 0 or SELL bit insertion disabled
- Passive time frame, indicated by ENL = ENH = 0

Command Frames

A command frame has two active phase parts, SRL active phase and SRH active phase. The command frame always starts with a high-level selection bit, independently whether the selection bit insertion (as defined by bit DSC.ENSELL) is enabled or not. The number of the bits transmitted during SRL and SRH active phases (except the selection bit) is defined by bit field DSC.NBC. SRL and SRH are combined to a 32-bit value whose length can be selected from 0 up to 32 bits. In other words, whenever bits of SRH are transmitted, they are always preceded by the transmission of the complete SRL content.

During the active phase of a command frame, the enable output signal ENC becomes active. The enable output signals ENL and ENH remain inactive.

The passive phase of a command frame always has a fixed length of $2 \times t_{FCL}$. The diagram shown in **Figure 18-5** assumes that the FCL clock is only generated during the active phase of the command frame (OCR.CLKCTRL = 0).

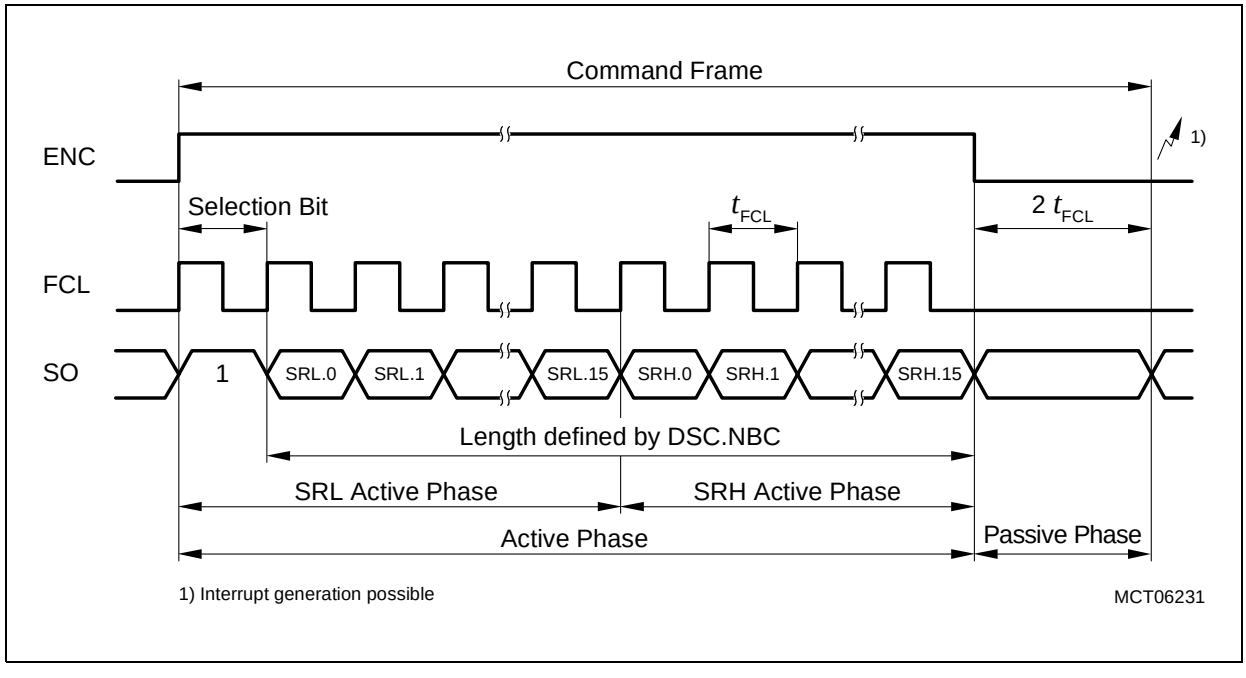


Figure 18-5 Command Frame Layout

Table 18-1 shows the programming of the bits to be transmitted and the resulting length of the complete command frame.

Table 18-1 Command Frame Length

DSC.NBC	SRL/SRH Bits that are Transmitted in Active Phase	Command Frame Length in t_{FCL} Periods
000000_B	No bit shifted out	1 + 0 + 2 = 3
000001_B	SRL[0] shifted out	1 + 1 + 2 = 4
000010_B	SRL[1:0] shifted out	1 + 2 + 2 = 5
000011_B	SRL[2:0] shifted out	1 + 3 + 2 = 6
…	…	…
001111_B	SRL[14:0] shifted out	1 + 15 + 2 = 18
010000_B	SRL[15:0] shifted out	1 + 16 + 2 = 19
010001_B	SRL[15:0] and SRH[0] shifted out	1 + 17 + 2 = 20
010010_B	SRL[15:0] and SRH[1:0] shifted out	1 + 18 + 2 = 21
010011_B	SRL[15:0] and SRH[2:0] shifted out	1 + 19 + 2 = 22
…	…	…
011111_B	SRL[15:0] and SRH[14:0] shifted out	1 + 31 + 2 = 34
100000_B	SRL[15:0] and SRH[15:0] shifted out	1 + 32 + 2 = 35
Other NBC combinations	Reserved; do not use these bit combinations.	

Data Frames

A data frame has two active phase parts, SRL active phase and SRH active phase. The number of bits that are transmitted can be programmed separately for each of these two phases. Bit field DSC.NDBL determines the number of SRL bits that are transmitted during the SRL active phase and DSC.NDBH determines the number of SRH bits that are transmitted during the SRH active phase.

SRL and SRH active phases can start with a low-level selection bit when enabled by bits DSC.ENSELL or DSC.ENSELH.

During the SRL active phase of a data frame, the enable output signal ENL becomes active and during the SRH active phase of a data frame, the enable output signal ENH becomes active. The enable output signal ENC remains inactive.

The length of the data frame's passive phase is variable and is defined by bit field DSC.PPD. It can be within a range of $2 \times t_{FCL}$ up to $31 \times t_{FCL}$. The diagram shown in **Figure 18-6** assumes that the FCL clock is only generated during the active phase of the data frame (OCR.CLKCTRL = 0).

Table 18-2, **Table 18-3**, and **Table 18-4** show the definitions of the five data frame parameters that determine the layout of the data frame.

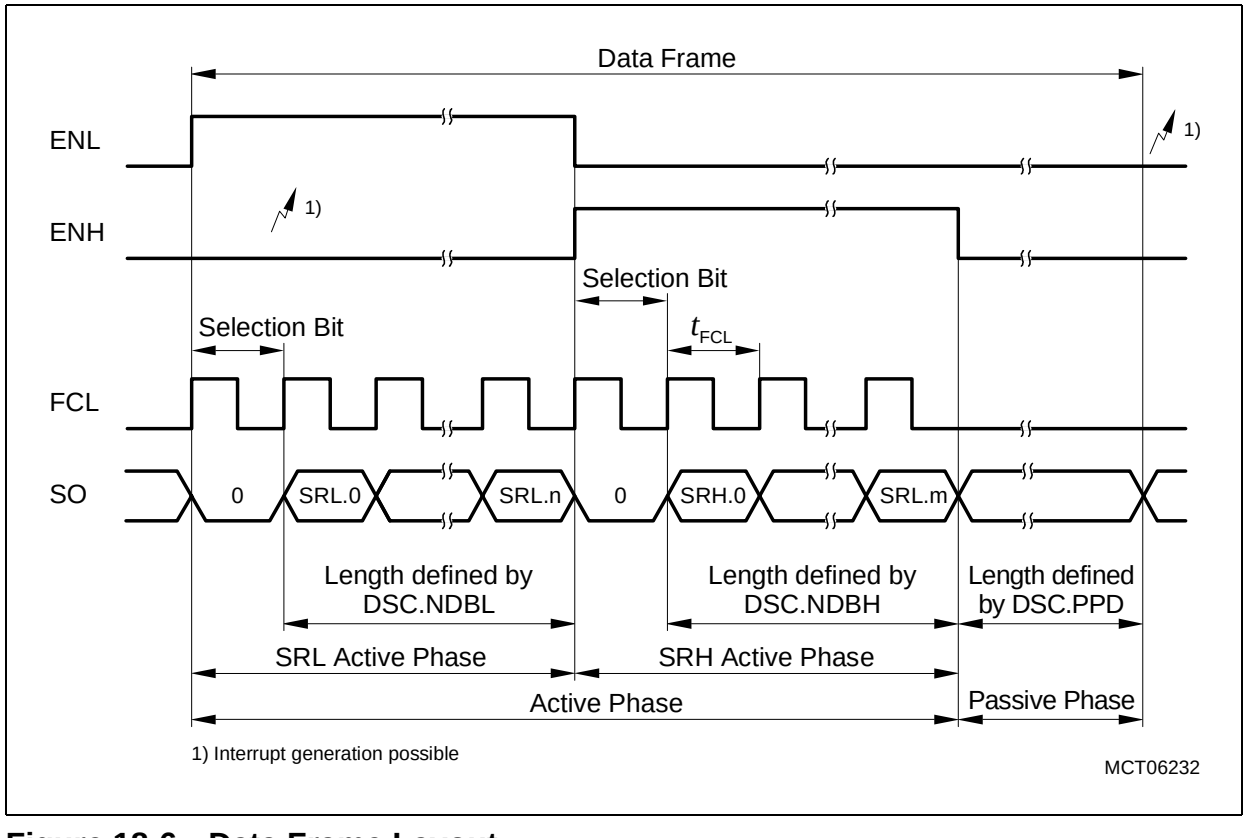


Figure 18-6 Data Frame Layout

Table 18-2 Data Frame Selection Bit Parameters

DSC.ENSELL	Selection Bit	DSC.ENSELH	Selection Bit
0	No selection bit inserted at the beginning of the SRL active phase	0	No selection bit inserted at the beginning of the SRH active phase
1	A low level selection bit is inserted at the beginning of the SRL active phase	1	A low level selection bit is inserted at the beginning of the SRH active phase

Table 18-3 Data Frame SRL/SRH Length Parameters

DSC.NDBL	SRL Bits Transmitted in SRL Active Phase	DSC.NDBH	SRH Bits Transmitted in SRH Active Phase
00000_B	No SRL bit transmitted	00000_B	No SRH bit transmitted
00001_B	SRL[0]	00001_B	SRHL[0]
00010_B	SRL[1:0]	00010_B	SRH[1:0]
00011_B	SRL[2:0]	00011_B	SRH[2:0]
…	…	…	…
01111_B	SRL[14:0]	01111_B	SRH[14:0]
10000_B	SRL[15:0]	10000_B	SRH[15:0]
Other bit combinations	Reserved; do not use these bit combinations.	Other bit combinations	Reserved; do not use these bit combinations.

Table 18-4 Data Frame Passive Phase Length

DSC.PPD	Passive Phase Length
00000_B	$2 \times t_{FCL}$
00001_B	$2 \times t_{FCL}$
00010_B	$2 \times t_{FCL}$
00011_B	$3 \times t_{FCL}$
…	…
01110_B	$30 \times t_{FCL}$
01111_B	$31 \times t_{FCL}$

The following formula determines the number of t_{FCL} cycles of a data frame: All parameters (bits and bit fields) are located in register DSC.

Number of cycles = ENSELL + NDBL + ENSELH + NDBH + PPD (18.1)

Note that in the formula above, PPD must be set to 2 when DSC.PPD $\leq 00010_B$.

Passive Time Frames

A passive time frame has the length defined by the five data frame parameters according **Equation (18.1)**. They are generated only in Data Repetition Mode. Under special conditions (command frame insertion), passive time frames can be shortened (see **Figure 18-9**).

During passive time frames, the data output SO have to be considered as invalid at the receiving device and the clock output FCL may toggle or not (as selected by bit OCR.CLKCTRL). The ENL and ENH enable signals remain at low level during a passive time frame.

18.1.2.2 Shift Register Operation

This section describes the SRL and SRH shift register loading.

SRL Shift Register Loading

During the SRL/SRH shift register load operation at the beginning of each downstream frame transmission, several parameters determine which information is loaded into the bits of the shift register. **Figure 18-7** shows the logic that is implemented for the SRL shift register loading operation. The logic for the SRH shift register loading operation is equivalent to the one for the SRL register. Its differences in data sources and register controls are described later in this section.

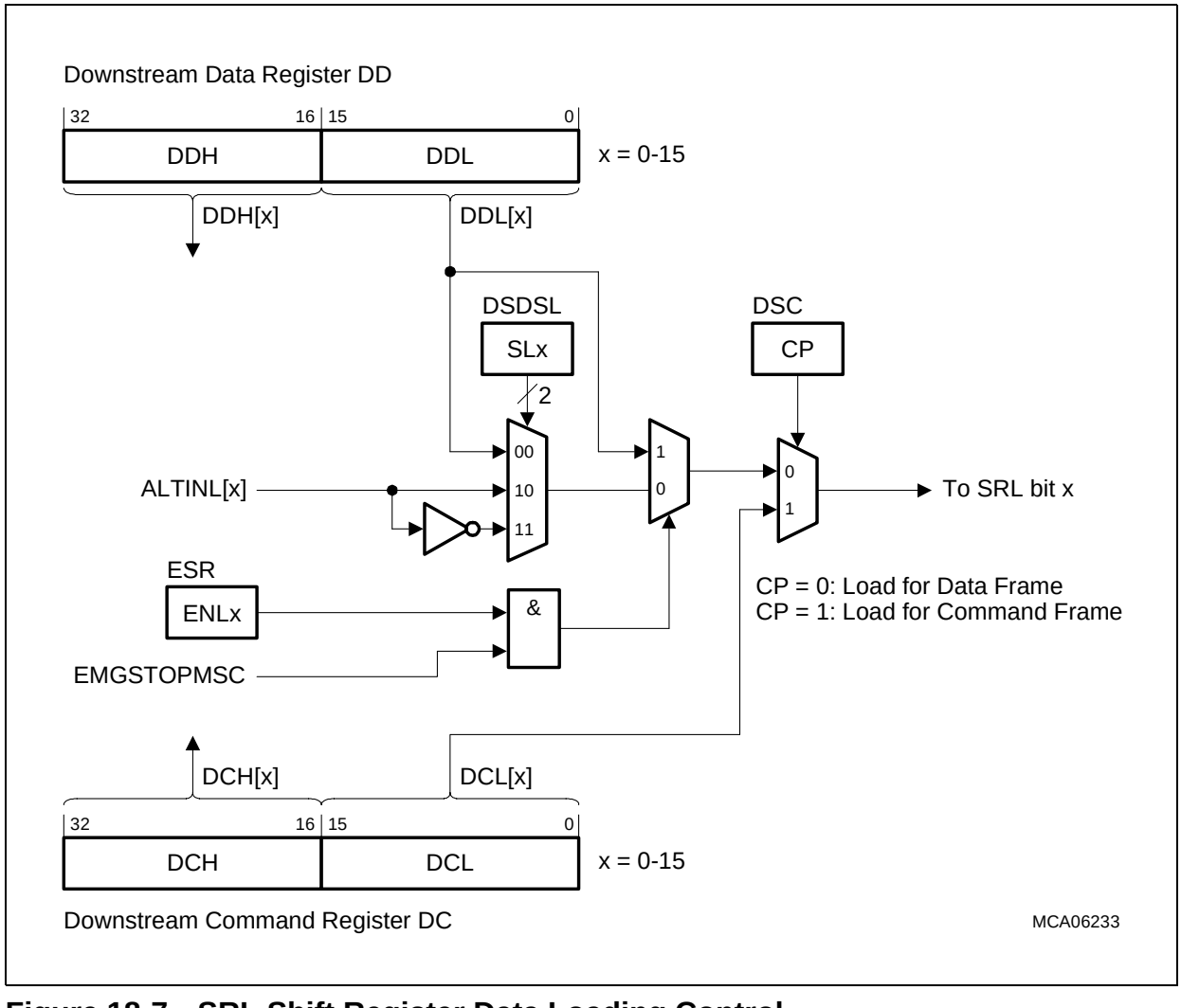


Figure 18-7 SRL Shift Register Data Loading Control

Four data sources can be selected for each SRL bit by using several control bits and one control signal:

- ALTINL input line (non-inverted)
- ALTINL input line (inverted)
- Bit of DD.DDL (downstream data register)
- Bit of DC.DCL (downstream control register)

When SRL is loaded for data frame transmission (DSC.CP = 0), bit fields DSDSL.SLx determine bit-wise which data is loaded into SRL bit x. The data source selection as controlled by DSDSL.SLx will only be effective when EMGSTOPMSC is inactive (at low level). When EMGSTOPMSC = 1 (active) during the load operation, all SRL[x] bits that are enabled for the emergency stop feature (bit ESR.ENLx = 1) are loaded directly with the corresponding bit DDL[x] of the downstream data register DD.

When SRL is loaded for command frame transmission (DSC.CP = 1), always the lower 16-bit part DCL of the downstream control register is loaded completely into SRL.

Table 18-5 summarizes all SRL data source selection capabilities (x = 0-15).

Table 18-5 SRL Data Source Selection Capabilities

DSC. CP	DSDSL. SLx	ESR. ENLx	EMGSTOPMSC	Selection
0	00_B	0	–	Bit DD.DDL[x] is loaded into SRL[x].
	01_B			Reserved.
	10_B			State of ALTINL[x] input is loaded into SRL[x].
	11_B			Inverted state of ALTINL[x] input is loaded into SRL[x].
	XX_B	1	1	Bit DD.DDL[x] is loaded into SRL[x].
1	XX_B	X	X	Bit fields DCL and DCH are completely loaded into SRL and SRH, respectively.

SRH Shift Register Loading

The SRH shift register load operation is equivalent to the SRL shift register load operation. The following differences must be taken into account for SRH shift register loading:

- Input lines ALTINH are connected instead of ALTINL input lines.
- DSDSH register bits control data source selection instead of DSDSL register.
- Emergency stop is enabled by ESR.ENHx bits instead of ESR.ENLx bits.
- Bits of the downstream data register high part DDH are selected instead of DDL.
- Downstream control register high part DCH is selected instead of DCL.

18.1.2.3 Transmission Modes

The downstream channel of the MSC makes it possible to select between two transmission modes:

- Triggered Mode, selected by DSC.TM = 0, or
- Data Repetition Mode, selected by DSC.TM = 1

Triggered Mode

In Triggered Mode, command frames or data frames are sent out as a result of a software event. When a frame transmission has been finished and no further frame transmission has been requested, the downstream channel returns to idle state and waits for the next frame transmission to be triggered by software.

When the Downstream Command Register DC is written, the command pending bit DSC.CP becomes set and a command frame will be immediately started and sent out if the downstream channel is idle. If a data or command frame is currently processed and output, the command frame transmission is delayed, and started when the active downstream frame has been finished. The command pending bit DSC.CP becomes cleared by hardware when the first bit of the command frame is sent out.

If the downstream channel is idle and the data pending bit DSC.DP is set by writing bit ISC.SDP with 1, a data frame will be immediately started and sent out if the downstream channel is idle. If a data frame or a command frame is currently processed and output, the data frame transmission is delayed and started when the active downstream frame has been finished. The data pending bit DSC.DP becomes cleared by hardware when the first bit of the data frame is sent out.

A command frame always has priority over the data frame. This means that if both frame pending bits are set (DSC.DP = DSC.CP = 1), the command frame will always be sent first. Therefore, a pending data frame transmission will be delayed as long as no further command frame transmission is running or requested.

Figure 18-8 is a flow diagram of the Triggered Mode. This diagram especially shows the behavior of the data and command pending bits DSC.DP and DSC.CP. If both frame pending bits are set (DSC.DP = DSC.CP = 1), the command frame will always be sent first, followed by the data frame (assuming no further command frame has been requested).

The type of the active frame that is currently processed and output is indicated by two status flags: DSS.DFA is set during a data frame transmission and DSS.CFA is set during a command frame transmission. Further, the downstream counter DSS.DC indicates the number of shift clock periods that have been elapsed since the start of the current frame.

In Triggered Mode, the shift register loading event as described in **Section 18.1.2.2** occurs just before a command or data frame transmission is started.

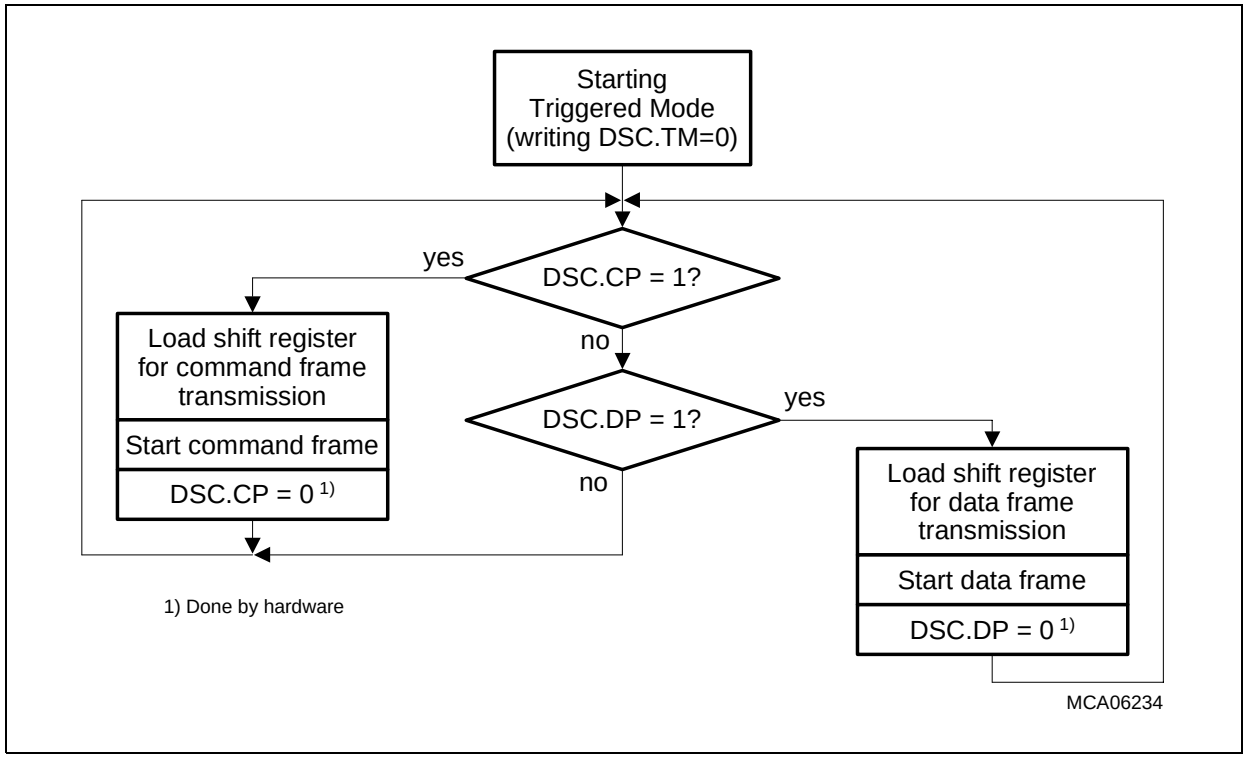


Figure 18-8 Triggered Mode Flow Diagram

Data Repetition Mode

In Data Repetition Mode, data frames are sent out continuously without any software interaction. In the time gap between two consecutive data frames, passive time frames can be inserted. The number of passive time frames to be inserted (0 to 15) is defined by bit field DSS.NPTF. The duration of data frame (t_{DF}) and passive time frames (t_{PTF}) is determined by the five data frame parameters (see **Equation (18.1)**). These parameters determine time reference points (TRP) at which a data or passive time frames is started (see diagram A in **Figure 18-9**).

The automatic data frame generation is controlled by the data pending bit DSC.DP. This bit is set near the end of the last transmitted passive time frame. At the next TRP, a data frame is started (if no command frame has been requested) and DSC.DP is cleared again by hardware after the data frame has been started. Data Frames are always aligned to time reference points. This means they always start at a TRP. Passive time frames can be shortened. This is especially the case when command frames are inserted.

Continuous data frame transmission can be interrupted by insertion of command frames. Command frames are initiated by software. When the downstream control register DSC is written, the command pending bit DSC.CP is set by hardware. CP = 1 indicates that the MSC starts a command frame at the next TRP, independently of whether a data

frame (indicated by DSC.DP = 1) or passive time frame should be started with the next TRP. This means also that command frames are always aligned to time reference points.

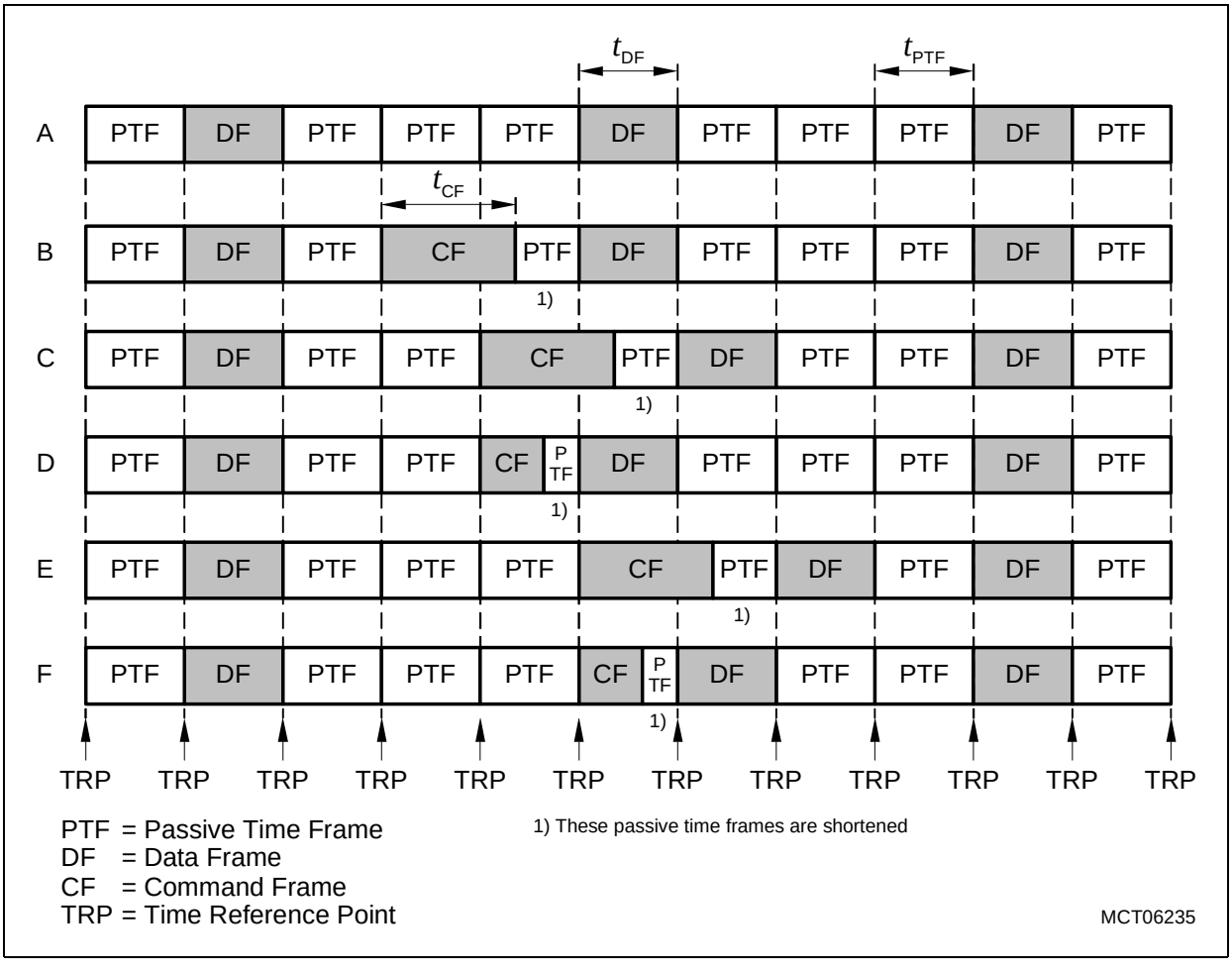


Figure 18-9 Data Repetition Mode Frame Examples with DSS.NPTF = 0011_B

Diagrams B to F in **Figure 18-9** show the command frame insertion in Data Repetition Mode.

In diagram B, a command frame has been requested during the first passive time frame after the data frame, and is inserted at the next TRP. In diagrams C and D, a command frame has been requested during the second passive time frame, and is inserted at the time reference point of the last nominal passive time frame.

When the command frame and data frame is not of the same length (this is the case in diagram B to F), a shortened passive time frame is inserted until the next TRP is reached. This ensures that the next data or normal passive time frame is again aligned to a TRP.

Figure 18-10 is a flow diagram of the Data Repetition Mode. This diagram especially shows the behavior of the data and command pending bits DSC.DP and DSC.CP. If both frame pending bits are set (DSC.DP = DSC.CP = 1), the command frame will always be

sent first, followed by the data frame when the next TRP is reached (assuming no further command frame has been requested).

When the last passive frame is transmitted, DSC.DP becomes set by hardware. This triggers the start of a data frame when the next TRP is reached.

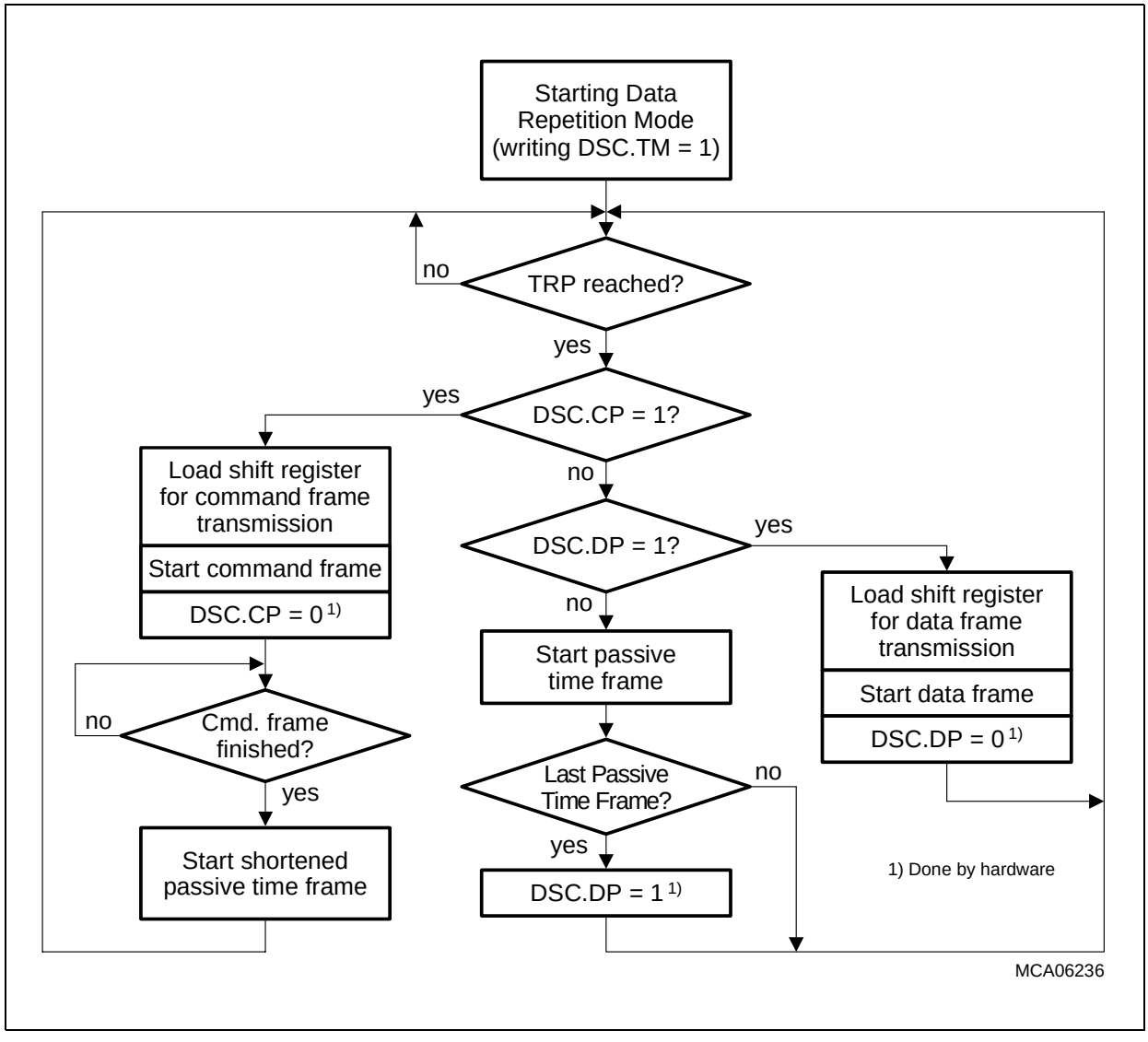


Figure 18-10 Data Repetition Mode Flow Diagram

The type of the active frame (data or command frame) that is currently processed and output is indicated by two status flags: DSS.DFA is set during a data frame transmission and DSS.CFA is set during a command frame transmission. Further, the downstream counter DSS.DC indicates the number of shift clock periods that have been elapsed since the start of the current data, command, or passive time frame.

As in Triggered Mode, the shift register loading event as described in **Section 18.1.2.2** occurs in Data Repetition Mode just before a TRP, this means shortly before a command or data frame transmission is started.

Passive Frame Counter in Data Repetition Mode

In Data Repetition Mode, a passive time frame counter DSS.PFC indicates how many time frames have been already transmitted after the last regular data frame occurrence. The passive time frame counter counts up from 0000_B to the value which has been written into bit field DSS.NPTF (number of passive time frames). DSS.PFC = 0000_B indicates that a data frame is requested for transmission.

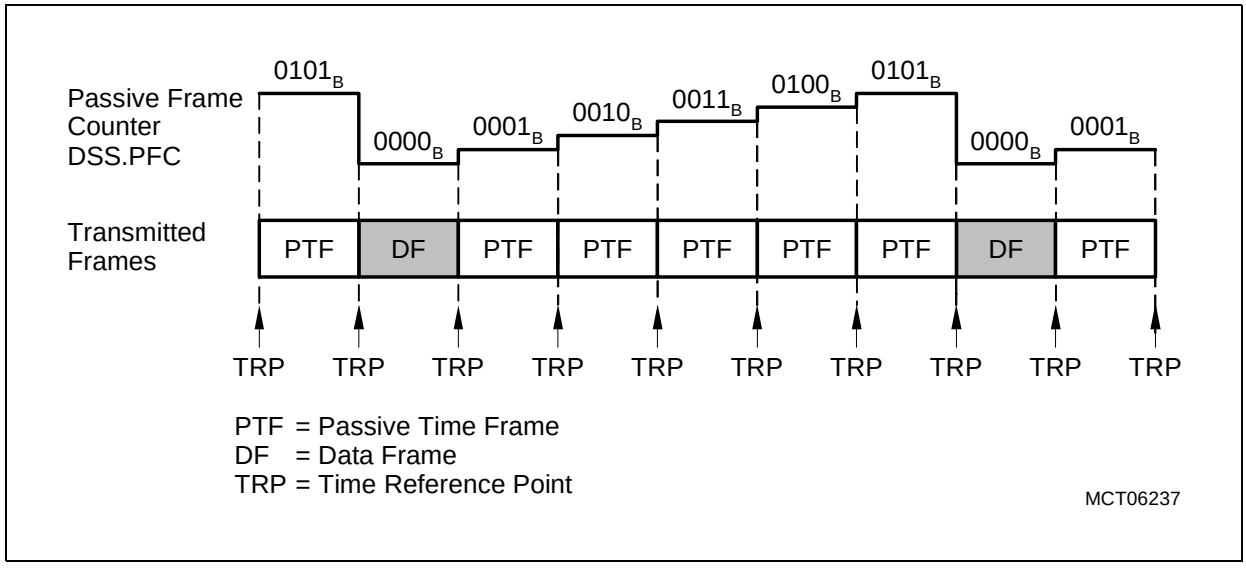


Figure 18-11 Passive Frame Counter Operation (with DSS.NPTF = 0101_B)

18.1.2.4 Downstream Counter and Enable Signals

During downstream channel operation, a 7-bit downstream counter DSS.DC is counting FCL shift clock periods. With the loading of the shift register, the downstream counter is reset to 00_H and started for counting up to the end of the downstream frame (end of passive phase).

In Triggered Mode, the downstream counter stops counting at the end of the passive phase and waits until a new downstream frame is started.

In Repetition Mode, the downstream counter does not stop at the end of the passive phase but is reset and starts counting up again with the next frame, independently whether a data frame, command frame, or passive time frame is started as next frame.

Figure 18-12 shows an example of downstream channel data frame transmission. In this example, the selection bit for the SRL active frame is enabled (ENSELL = 1), and the selection bit for the SRH active frame is disabled (ENSELH = 0). With loading of the shift register SRL/SRH, the downstream counter is reset and then starts counting up with each FCL clock until the end of the passive phase. ENL is set to high level at the beginning of the SRL active frame selection bit.

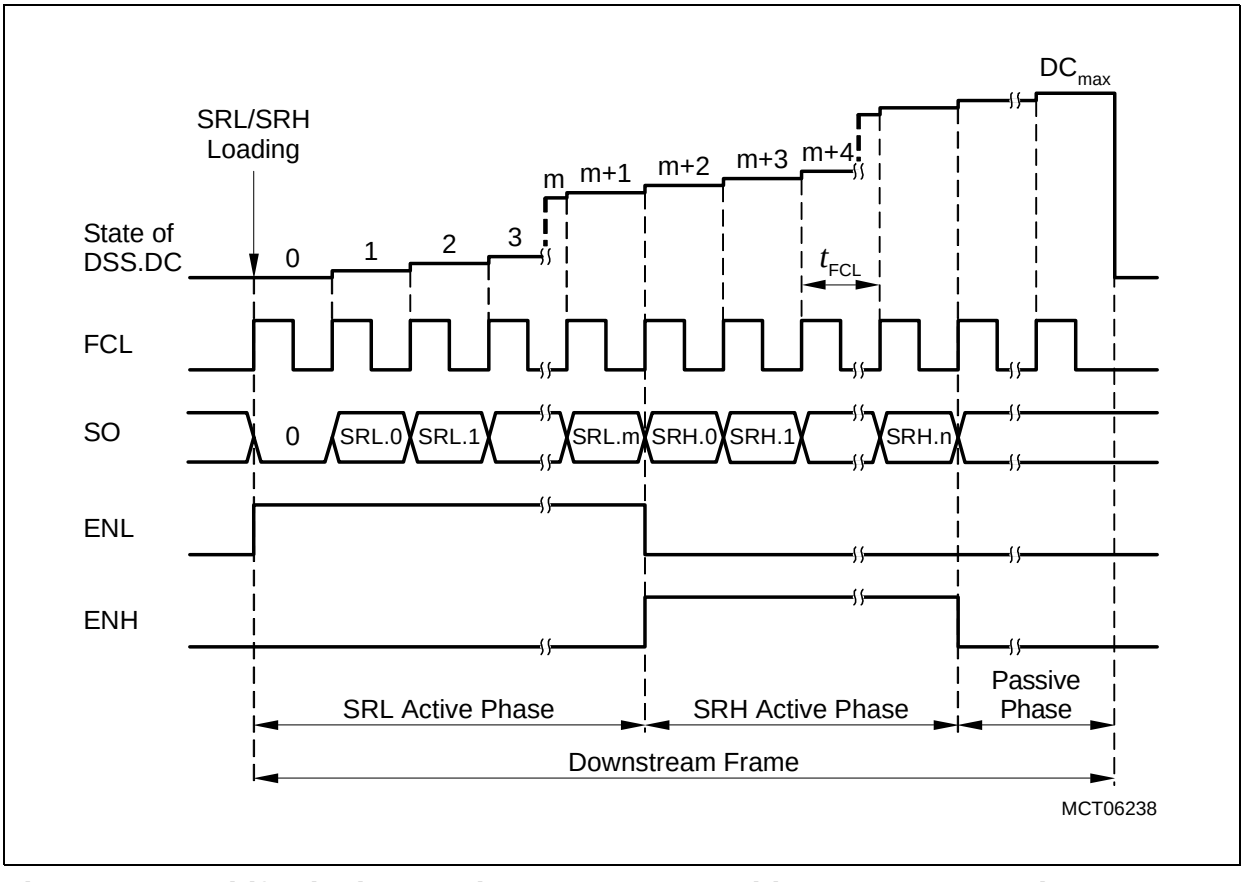


Figure 18-12 Shift Clock Counting: Data Frame with ENSELL = 1 and ENSELH = 0

When the selection bit for the SRL active frame is disabled (ENSELL = 0, see **Figure 18-13**), the loading of the shift register SRL/SRH (and reset of the downstream counter) occurs one FCL clock cycle before the first data bit SRL.0 is output. ENL is set to high level with the beginning of the first data bit SRL.0.

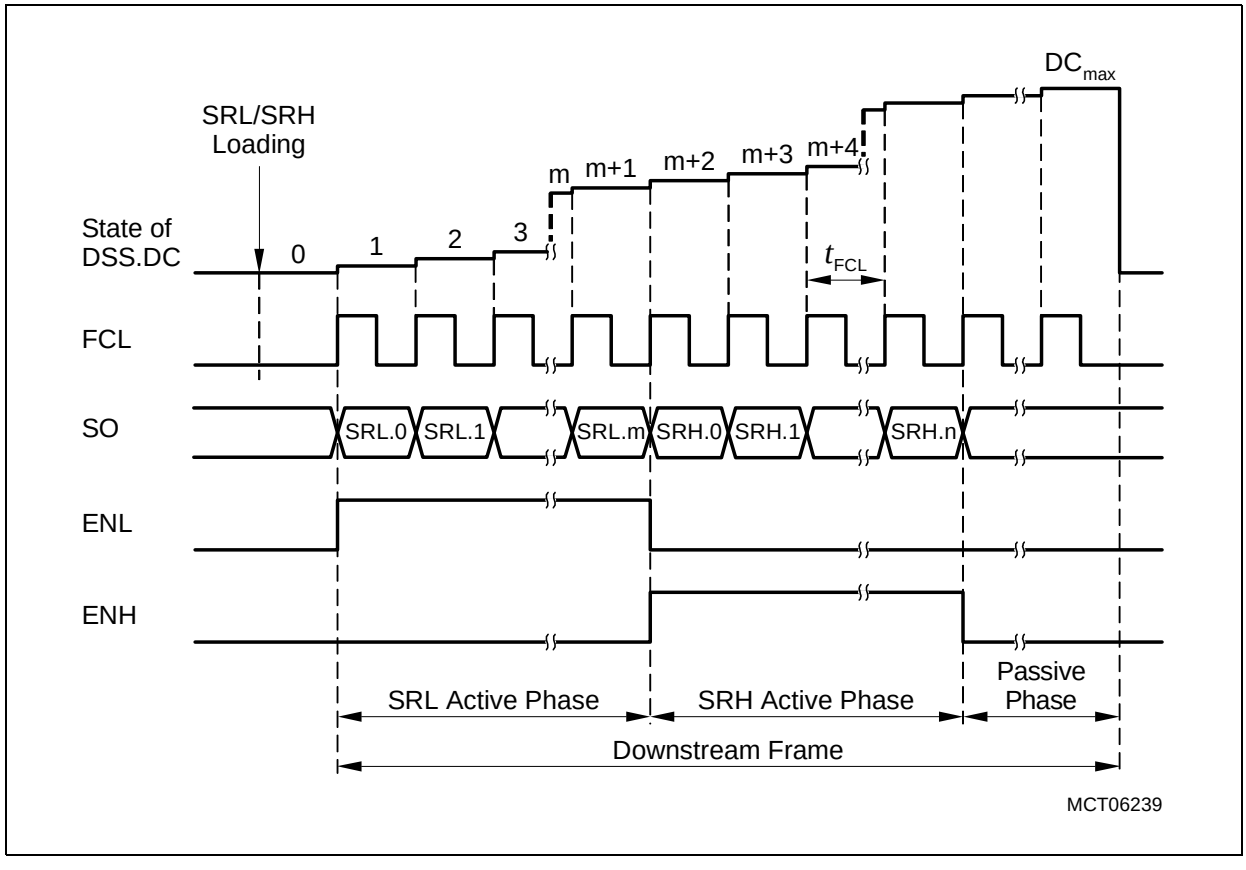


Figure 18-13 Shift Clock Counting: Data Frame with ENSELL = 0 and ENSELH = 0

18.1.2.5 Baud Rate

The baud rate of the downstream channel's serial transmission is defined by the frequency of the serial clock FCL, and is always $f_{MSC}/2$. The f_{MSC} generation is device specific and depends on the implementation of the MSC module. The TC1797 specific clock generation is described on **Page 18-65**.

18.1.2.6 Abort of Frames

Only a reset condition of the device can abort a current transmission. The MSC module does not start a new frame transmission when the downstream channel becomes disabled, the suspend mode is requested, or the sleep mode is entered. If one of these three conditions becomes active during a running frame transmission, the frame transmission is completely finished before the requested abort state is entered. Note that in this case no time frame finished interrupt is generated any more.

18.1.3 Upstream Channel

The MSC upstream channel is an asynchronous serial receiver based on the standard asynchronous data transfer protocol. It is dedicated to receive a serial data stream from a peripheral device via its serial data input SDI, using two specific data frame formats.

Figure 18-14 is a block diagram of the MSC upstream channel.

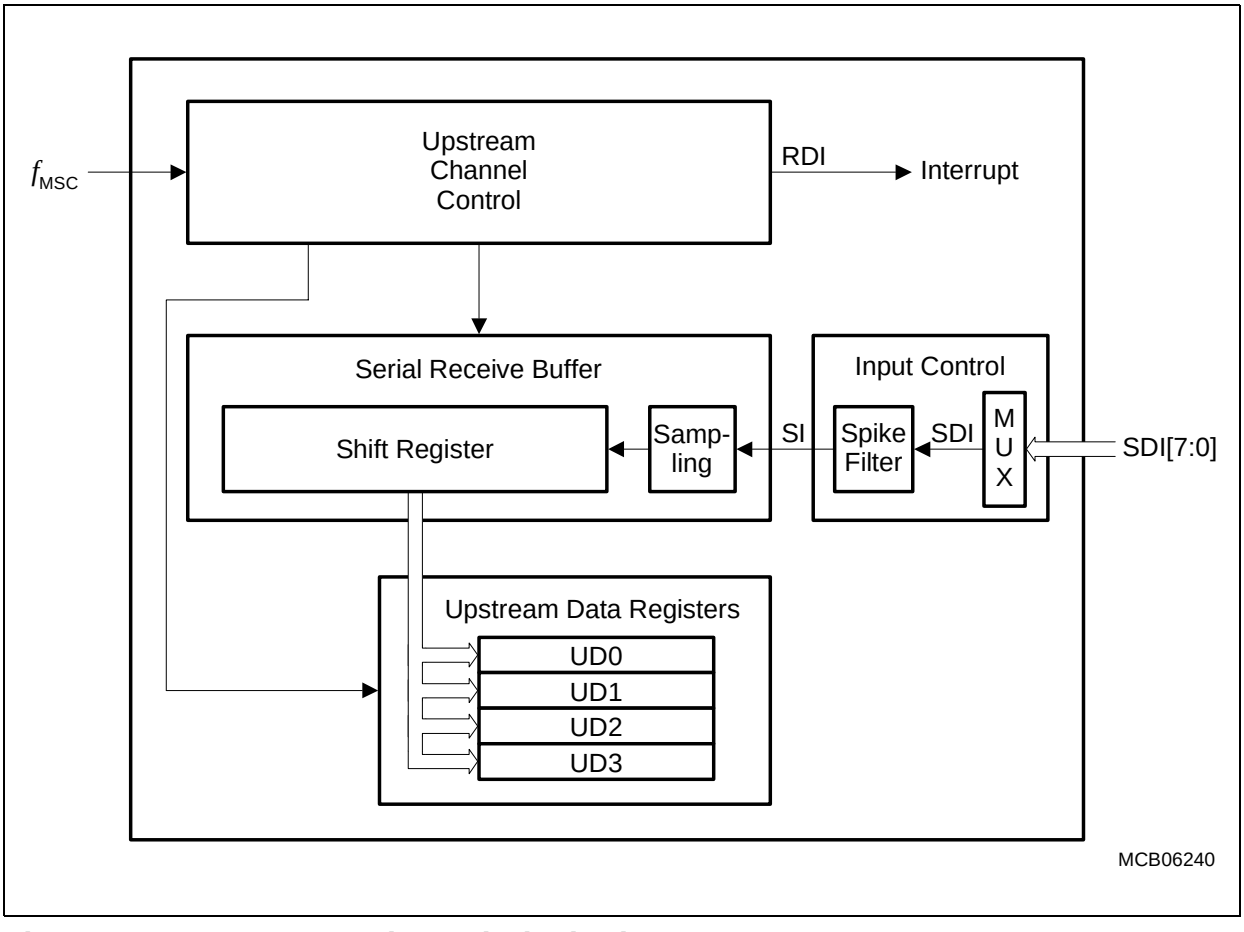


Figure 18-14 Upstream Channel Block Diagram

The incoming data at SI is sampled after it has been filtered for spikes. The detected logic states of the serial input are clocked into a shift register. After the complete reception of the serial data frame, the content of the shift register is transferred into one of the four data registers, and an interrupt can be generated optionally.

The reception baud rate is directly coupled to the module clock f_{MSC}, and can be within a range of $f_{MSC}/4$ up to $f_{MSC}/256$.

18.1.3.1 Data Frames

The asynchronous data frames used by the upstream channel include four basic parts:

1. One start bit, always at low level
2. An 8-bit data field D[7:0] with LSB first
3. An optional 4-bit address field A[3:0] with LSB first
4. One parity bit and two stop bits, that are always at high level

As shown in **Figure 18-15**, the 16-bit upstream data frame includes an additional 4-bit address field. The upstream frame type is selected by bit USR.UFT.

- USR.UFT = 0: 12-bit upstream data frame selected
- USR.UFT = 1: 16-bit upstream data frame with 4-bit address field selected

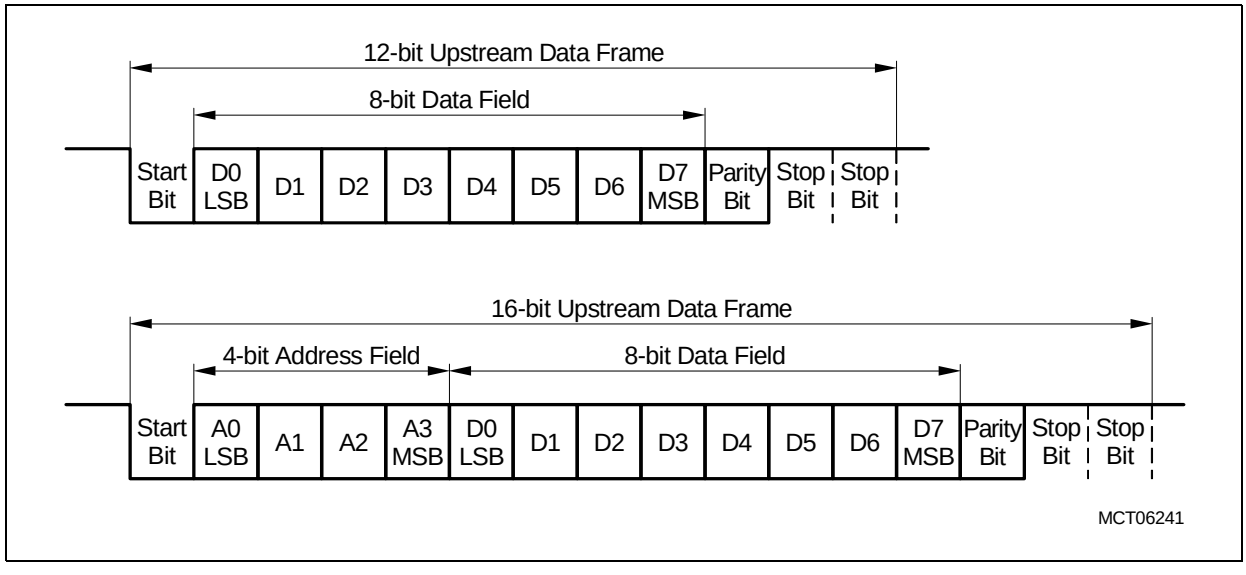


Figure 18-15 Upstream Channel Frame Types

18.1.3.2 Parity Checking

The incoming parity bit of the data frames can be checked by the upstream channel. When a parity error is detected, the parity error flag PERR in the related Upstream Data Register UDx is set. Note that a setting of the parity error flag PERR does not generate an interrupt. The PERR bits must be checked by software. The UDx registers also store the parity bit of the incoming data frame (UDx.P) and the parity bit that is generated internally (UDx.IPF).

Bit USR.PCTR determines the parity mode, even or odd, that is selected for parity checking. With USR.PCTR = 0, even parity mode is selected. Even parity means that the parity bit is set on an odd number of 1s in the data field (12-bit upstream data frame) or in the address plus data field (16-bit upstream data frame). With USR.PCTR = 1, odd parity mode is selected. In odd parity mode, the parity bit is set on an even number of 1s of the related data.

The parity checking logic in the upstream channel also controls whether start bit and the two stop bits of the upstream data frame are at correct logic level. If the start bit is not at low level and the two stop bits are not at high level at the end of the frame reception, the parity error flag UDx.PERR is set, too.

18.1.3.3 Data Reception

The reception of the upstream frame is started with a falling edge (1-to-0 transition) on the SI line. When the start bit is detected, serial reception is enabled and the receive circuit begins to sample the incoming serial data and to buffer it in the receive buffer. After the second stop bit has been detected, the content of the receive buffer is transferred to one of four upstream data registers UDx. The receive circuit then waits for the next start bit (1-to-0 transition) at the SI line. When the content of the receive buffer has been transferred to UDx, the valid bit UDx.V is set by hardware, and a receive interrupt can be generated.

Note: The SI input line is the filtered non-inverted (OCR.ILP = 0) or inverted (OCR.ILP = 1) SDI input signal. The SI input signal selection is described on ***Page 18-30****).*

Frame Reception with Address Field

Frame reception for a 16-bit data frame (see **Figure 18-16**) is selected by USR.UFT = 1. When the content of the receive buffer has been received completely, it is transferred to one of the four UDx registers. The two most significant address bits A[3:2] of the received 4-bit address field select the number x of register UDx in which the received frame content is stored. Register UDx is loaded with the two least significant address bits A0 and A1 (UDx.LABF), the 8-bit data (UDx.DATA), the received parity bit (UDx.P), the calculated parity bit (UDx.IPF), and the parity checking result (UDx.PERR). Finally, the valid bit UDx.V is set to indicate that the UDx register contains valid data.

The current state of the frame reception is indicated by the content of an upstream counter that is readable via bit field USR.UC. The upstream counter is a 5-bit counter that counts the upstream frame bits during reception. As shown in **Figure 18-16**, the upstream counter is loaded with 10000_B at the detection of a start bit. It counts down and is again at 00000_B when the second stop bit has been detected and the frame reception is finished.

The state of the serial input data line SI is sampled in the middle of a bit cell and shifted into the receive buffer at the end of the bit cell. The frequency of the shift clock f_{SHIFT} depends the selected baud rate (see **Page 18-25**).

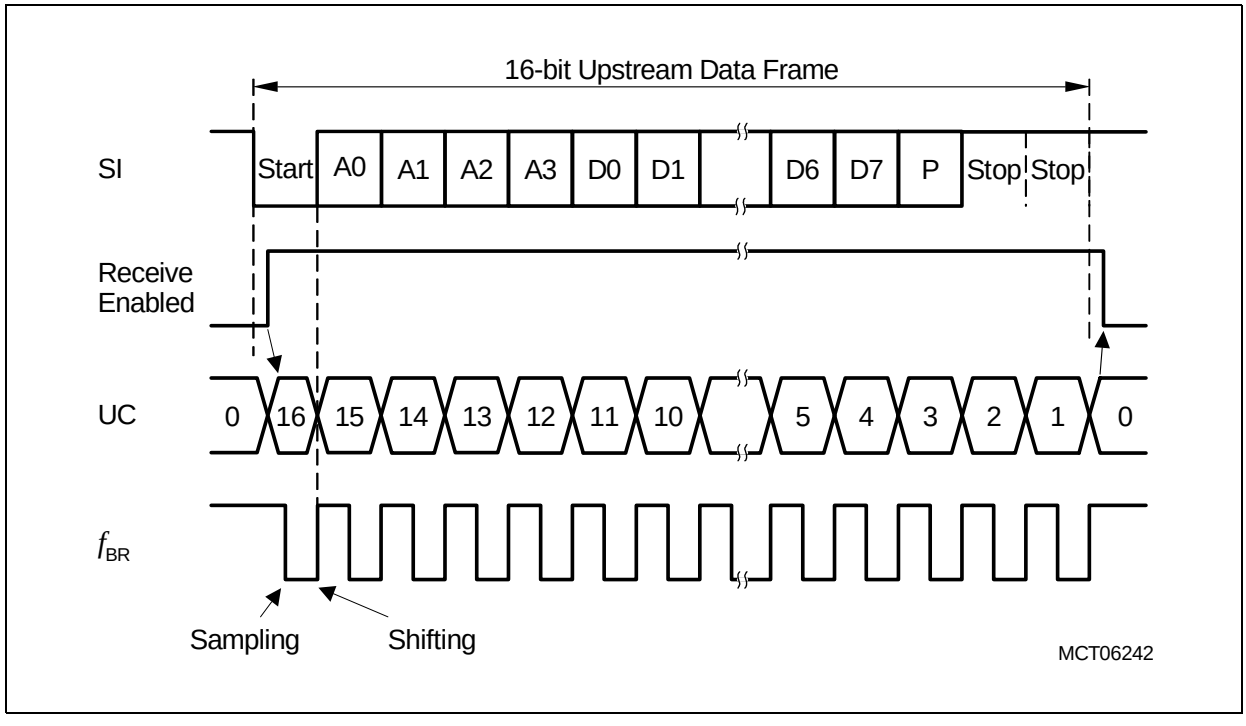


Figure 18-16 16-bit Upstream Reception

Data Reception without Address Field

Frame reception for a 12-bit data frame is selected by USR.UFT = 0. The reception scheme is comparable with that of the 16-bit data frame reception but there are a few differences:

- The upstream counter is initially loaded with 01100_B.
- The received frame content is always stored in register UD0.
- Bit field UD0.LABF is always loaded with 00_B when the frame is stored.

18.1.3.4 Baud Rate

The baud rate of the upstream channel is derived from the MSC module clock f_{MSC}. **Figure 18-17** shows the configuration of the upstream channel clock circuitry.

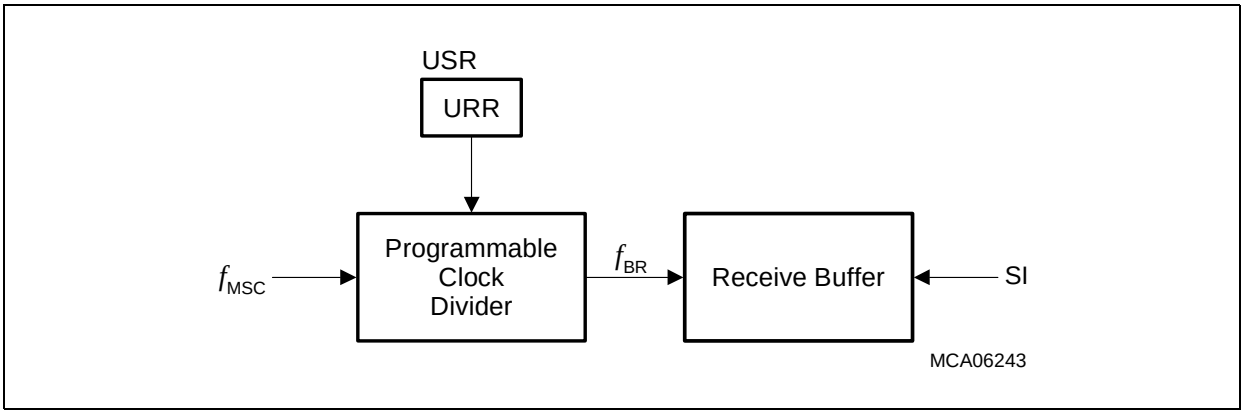


Figure 18-17 Upstream Channel Clock Circuitry

The serial data input SI is evaluated with the baud rate clock f_{BR} in the middle of each bit cell, and latched in case of a data bit. The baud rate clock f_{BR} is derived from f_{MSC} by a programmable clock divider. The frequency of f_{BR} determines the width of a received bit cell and therefore the baud rate for the received data. The content of bit field USR.URR selects the baud rate according **Table 18-6**. The resulting baud rate formula is:

$$\text{Baud rate}_{\text{MSC Upstream Channel}} = \frac{f_{MSC}}{DF} \qquad (18.2)$$

Table 18-6 Upstream Channel Divide Factor DF Selection & Baud Rate

USR.URR	Divide Factor DF	Baud Rate
000_B	reception disabled	–
001_B	4	$f_{MSC}/4$
010_B	8	$f_{MSC}/8$
011_B	16	$f_{MSC}/16$
100_B	32	$f_{MSC}/32$
101_B	64	$f_{MSC}/64$
110_B	128	$f_{MSC}/128$
111_B	256	$f_{MSC}/256$

Note: With the USR.URR = 000_B the upstream channel is disabled and data reception is not possible.

The content of bit field USR.URR determines the operation of an internal sampling reload counter that is clocked with f_{MSC}. **Figure 18-18** shows the operation of the sampling counter at the beginning of an upstream frame with a divide factor DF of 8 (USR.URR = 010_B is equal to DF = 8) which means eight sampling clocks per each frame bit cell.

When the upstream channel is in idle state, it waits for a falling edge (1-to-0 transition) at SI. Therefore, the sample counter starts counting up and is reset when the selected divide factor DF as shown in **Table 18-6** is reached. In the middle of the sampling counter's count range, the logic state at SI is evaluated and, in case of a data bit, latched in the receive buffer's shift register. With the reload of the sampling counter, the shift register is shifted by one bit position.

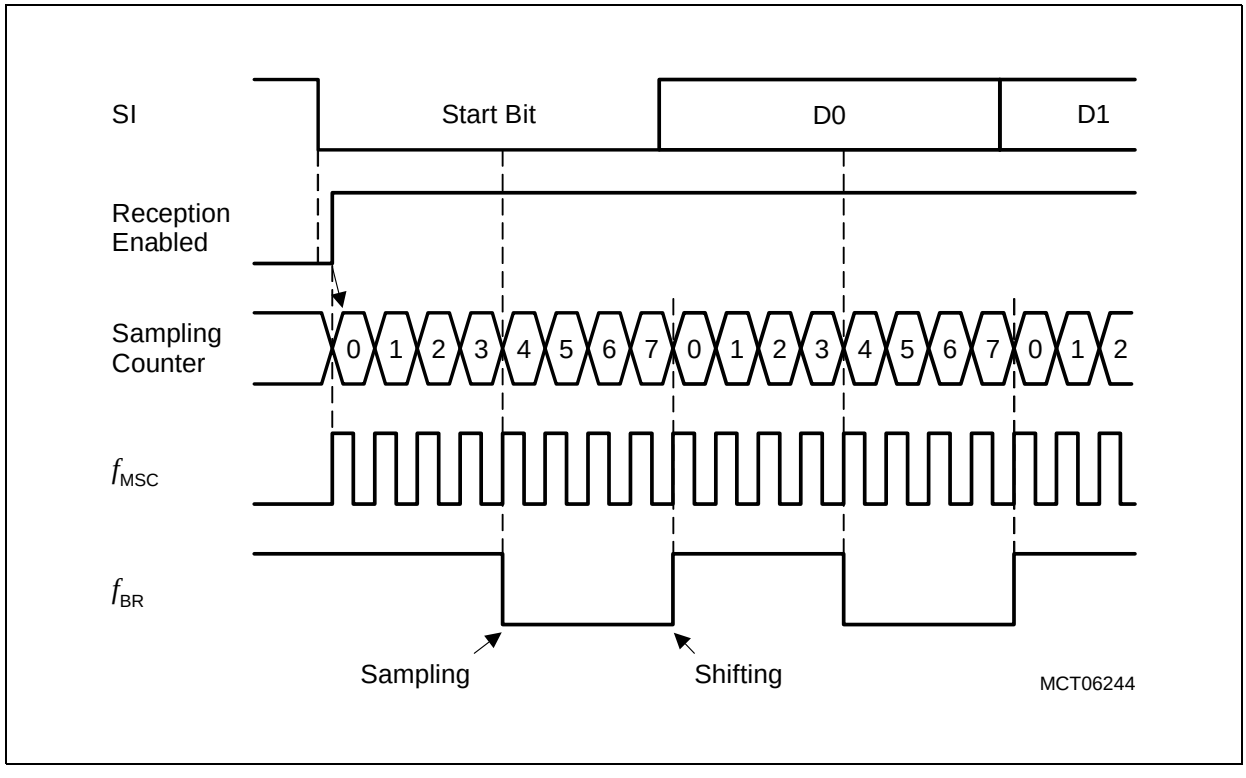


Figure 18-18 Upstream Channel Sampling with URR = 010_B

18.1.3.5 Spike Filter

The upstream channel input line SDI is sampled using a built-in spike filter with synchronization stage, both clocked with f_{MSC}. The spike filter is a chain of flip-flops with a majority decision logic (2 out of 3). A sampled value that is found at least twice in three samples is taken as data input value for SI.

18.1.4 I/O Control

The types of I/O control logic for the MSC module I/O lines are shown in **Figure 18-19**. The downstream channel generates five output signals that control eight MSC module outputs, split into four chip select outputs, two clock outputs, and two serial data outputs. The upstream channel has one input signal.

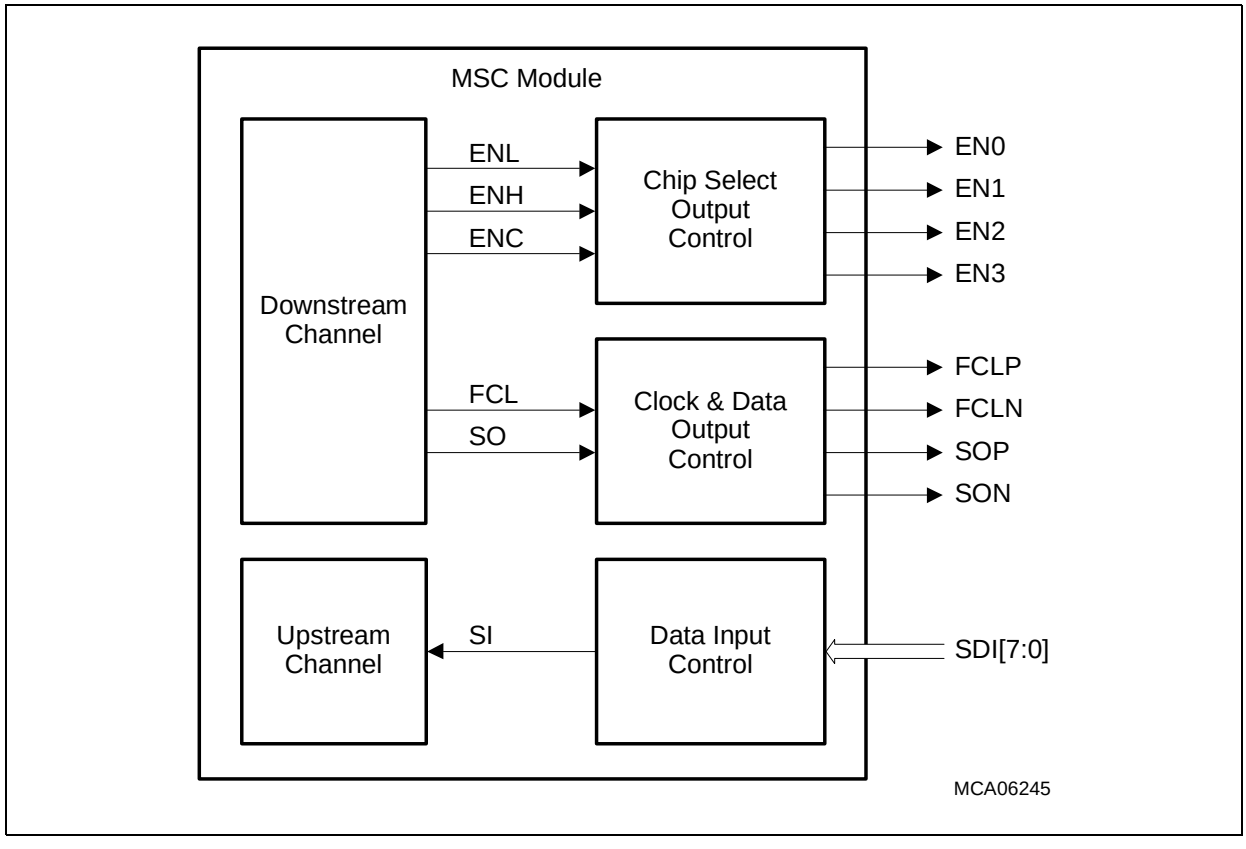


Figure 18-19 I/O Control

The MSC module I/O signals is controlled by bit fields that are located in the Output Control Register OCR.

18.1.4.1 Downstream Channel Output Control

As shown in **Figure 18-5** and **Figure 18-6**, the active phases during downstream channel operation are indicated by three enable signals:

- ENL indicates the SRL active phase of a data frame
- ENH indicates the SRH active phase of a data frame
- ENC indicates the active phase of a command frame

The chip select output control logic of the MSC uses a signal compressing scheme (similar to the interrupt request compressing scheme in **Figure 18-27**) that allows each of the three enable signals to be directed via a 2-bit selector to one of the four chip enable

outputs EN[3:0]. This also makes it possible to connect more than one internal enable signal (ENL, ENH, ENC) to one chip enable output ENx. Three bit fields in register OCR (CSL, CSH, and CSC) determine which chip enable output becomes active on a valid internal enable signal.

In the MSC, enable signals are high-level active signals. If required in a specific application, all chip enable outputs ENx can be assigned for low-level active polarity by setting bit OCR.CSLP.

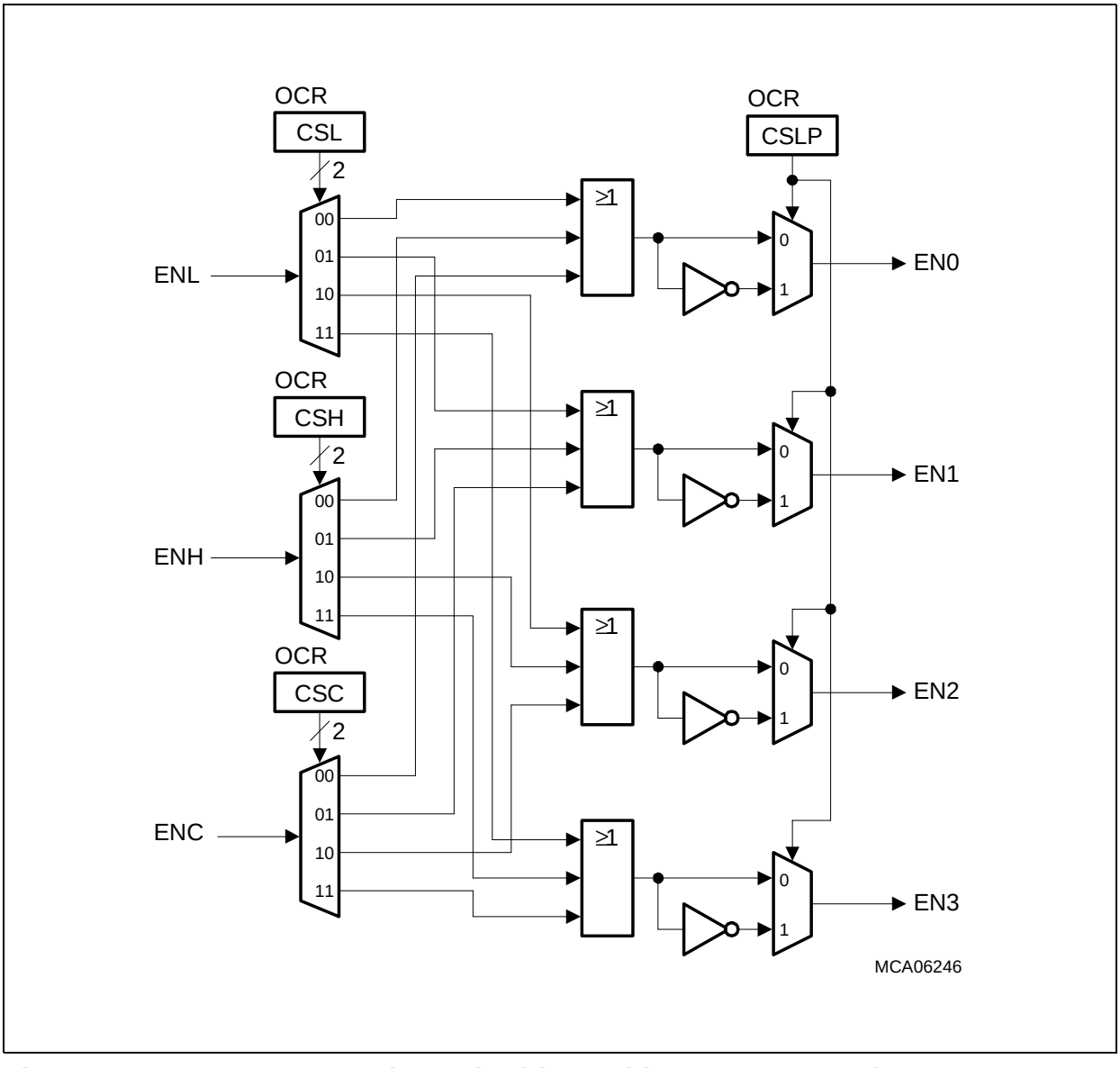


Figure 18-20 Downstream Channel: Chip Enable Output Control

At the MSC downstream channel, the internal serial clock output FCL and data output line SO are available outside the MSC module as two signal pairs with inverted signal polarity, FCLP/FCLN and SOP/SON. Both, clock and data outputs, are generated from the module internal signals FCL and SO according to **Figure 18-21**.

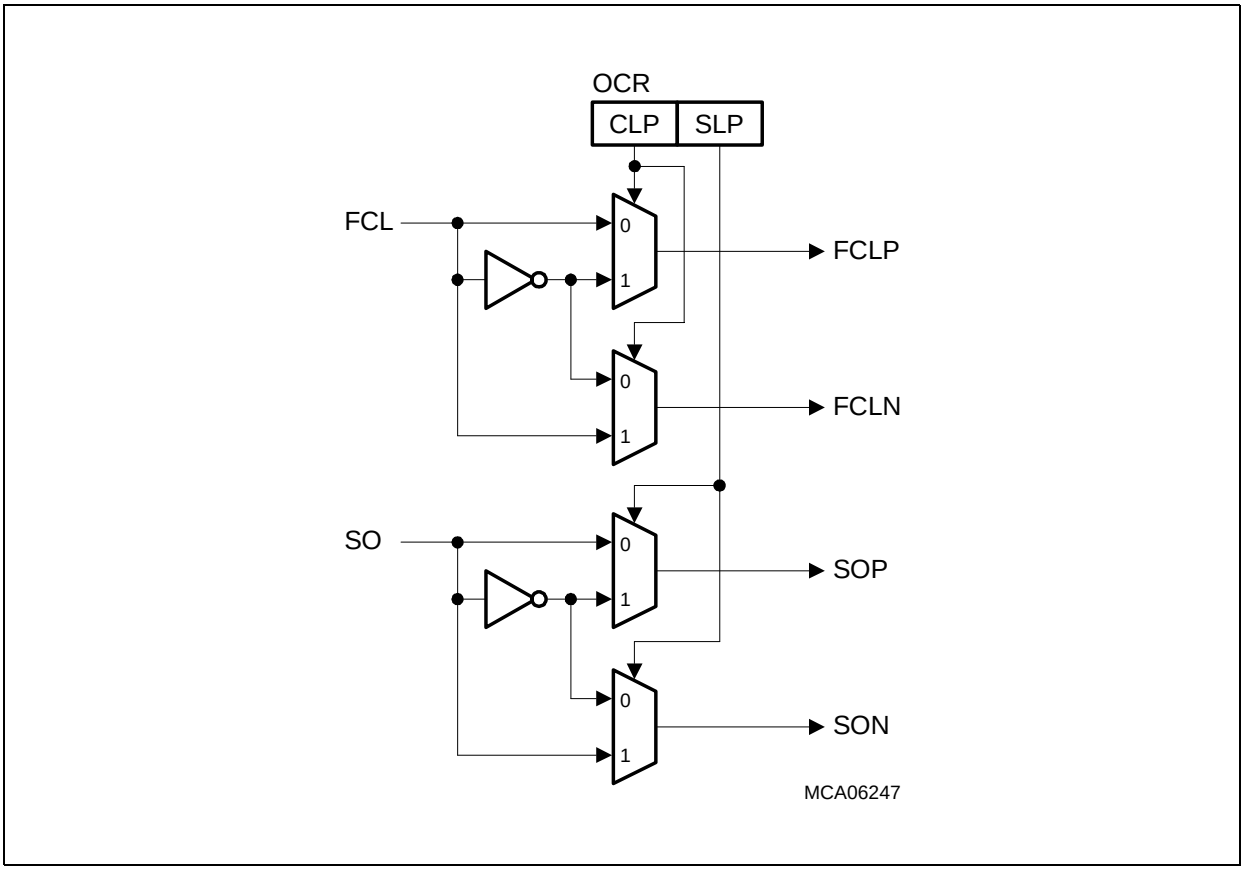


Figure 18-21 Downstream Channel: Clock and Data Output Control

With OCR.CLP = 0, FCLP has identical and FCLN has inverted polarity compared to FCL. Setting OCR.CLP, exchanges the signal polarities of FCLP and FCLN. An equivalent control capability is available for the SOP and SON data outputs (controlled by OCR.SLP).

One additional control capability not shown in **Figure 18-21** is available for the FCL signal. With OCR.CLKCTRL = 1, the FCL clock signal will always be generated, independently whether a downstream frame is currently transmitted or not. If OCR.CLKCTRL = 0, FCL becomes only active during the active phases of data or command frames (not during passive time frames).

18.1.4.2 Upstream Channel

As shown in **Figure 18-22**, the MSC upstream channel can be connected to up to eight SDI[7:0] serial inputs. Bit field OCR.SDISEL selects one out of these input lines (input signal SDI). If OCR.ILP = 0, SDI is directly connected to the serial receive buffer input SI. If OCR.ILP = 1, SDI is connected to input SI via an inverter.

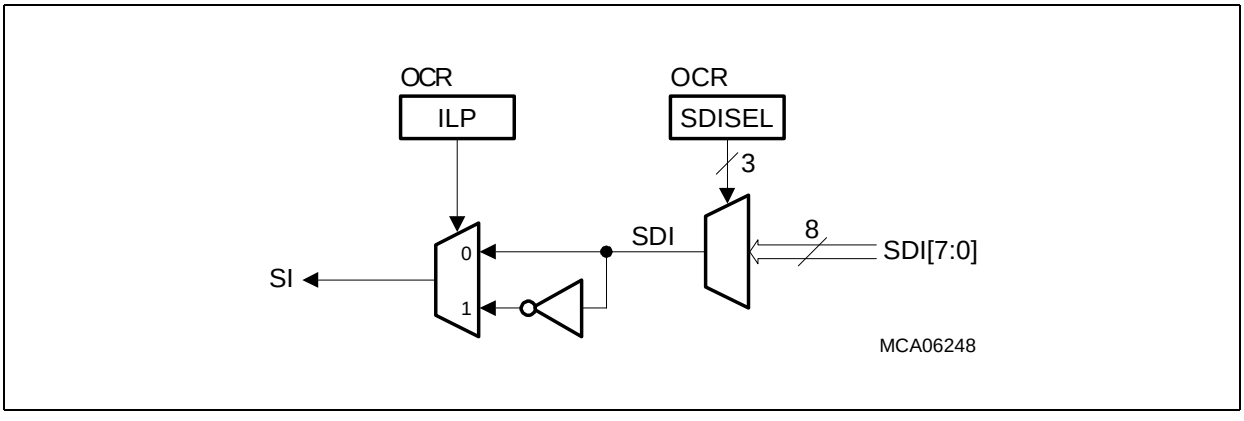


Figure 18-22 Upstream Channel Serial Data Input Control

18.1.5 MSC Interrupts

The MSC module has four interrupt sources and four service request outputs. A service request output is able to generate interrupts (controlled by a service request control register) or DMA requests. The service request output assignment, interrupt or DMA request, is specific for each microcontroller that is using the MSC. In this section, the term “interrupt request” has the meaning of “service request” that is able to handle interrupt or DMA requests.

Each interrupt source is provided with a status flag, enable bit(s) with software set/clear capability, and an interrupt node pointer. An interrupt event, internally generated as a request pulse, is always stored in an interrupt status flag that is located in the Interrupt Status Register ISR. All interrupt status flag can be set or cleared individually by software via the interrupt Set Clear Register ISC. Software-controlled interrupt generation can be initiated by setting the interrupt status flag of the corresponding interrupt. Each interrupt source can be enabled or disabled individually. When an interrupt event is enabled, a 2-bit interrupt node pointer determines which of the service request outputs will be activated.

Table 18-7 shows the four MSC interrupt sources.

Table 18-7 MSC Interrupts

Interrupt Type	Generated by
Data frame interrupt	Downstream Channel
Command frame interrupt	
Time frame finished interrupt	
Receive data interrupt	Upstream Channel

18.1.5.1 Data Frame Interrupt

A data frame interrupt can be generated when either the first or the last data bit of the downstream channel is shifted out and becomes available at the SO output line (see also **Figure 18-6**). Bit ICR.EDIEI selects which case is selected.

Note: If ICR.EDIE = 10_B, an interrupt at the first data bit is only generated if DSC.NDBL is not equal 00000_B. This means, at least one SRL bit must be shifted out for the first data bit shifted interrupt to become active.

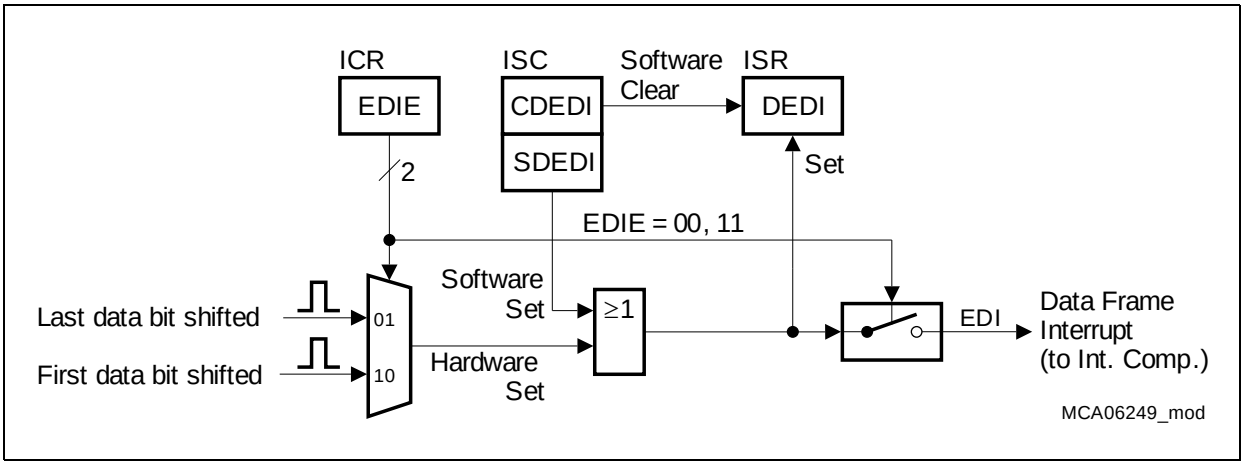


Figure 18-23 Data Frame Interrupt Control

18.1.5.2 Command Frame Interrupt

A command frame interrupt can be generated at the end of a downstream channel command frame (see also **Figure 18-5**).

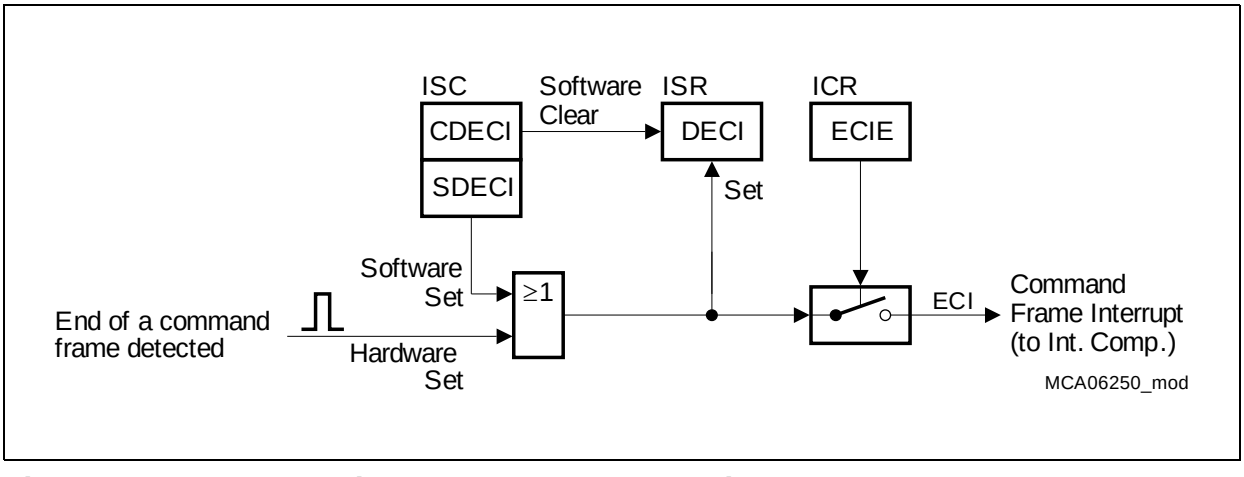


Figure 18-24 Command Frame Interrupt Control

18.1.5.3 Time Frame Finished Interrupt

A time frame finished interrupt can be generated at the end of a downstream channel passive time phase.

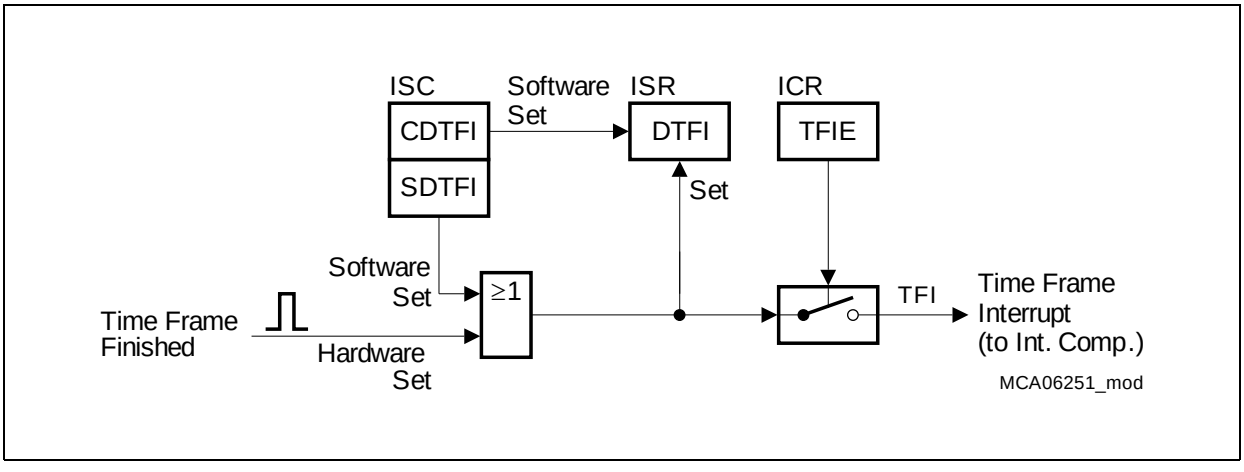


Figure 18-25 Time Frame Interrupt Control

18.1.5.4 Receive Data Interrupt

Whenever the upstream channel receives data in registers UDx (x = 0-3), the MSC is able to generate an interrupt. Three interrupt generation conditions can be selected for the receive data interrupt:

- Each update of UDx (x = 0-3) generates a receive data interrupt.
- Each update of UDx (x = 0-3) generates a receive data interrupt when the updated value is not equal 00_H.
- Only an update of register UD3 generates a receive data interrupt.

The selection of the interrupt generation condition is controlled by bit field ICR.RDIE. Setting ICR.RDIE = 0 disables the receive data interrupt in general. ISR.URDI is the interrupt status flag that can be set or clear when writing bits ISC.SURDI or ISC.CURDI with a 1.

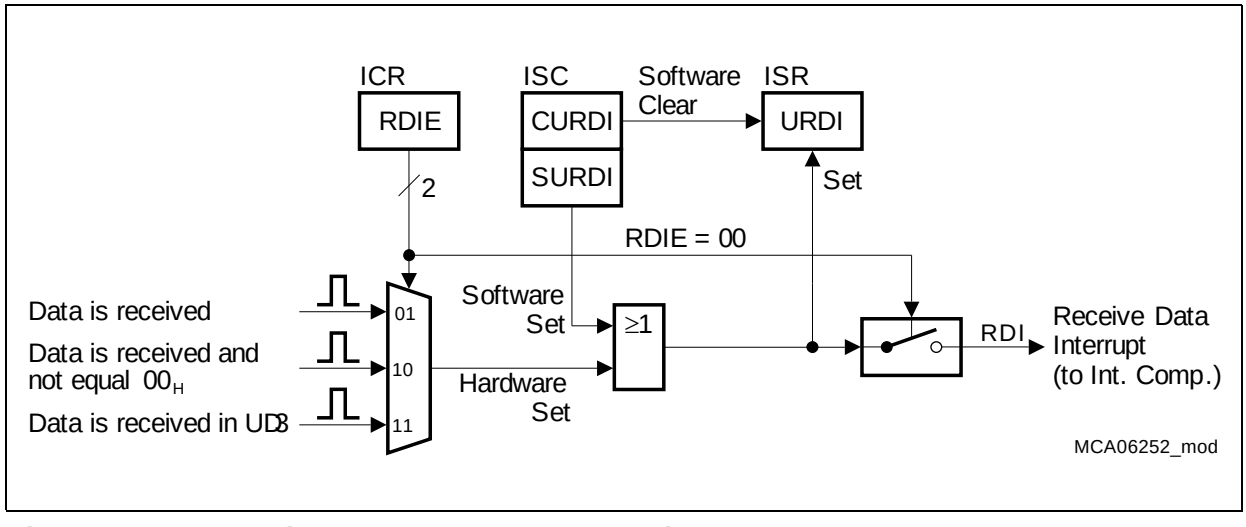


Figure 18-26 Receive Data Interrupt Control

18.1.5.5 Interrupt Request Compressor

The interrupt control logic of the MSC uses an interrupt compressing scheme that allows high flexibility in interrupt processing. Each of the four interrupt sources can be directed via a 2-bit interrupt node pointer to one of the four service request outputs SR[3:0]. This also makes it possible to connect more than one interrupt source to one interrupt output SRx.

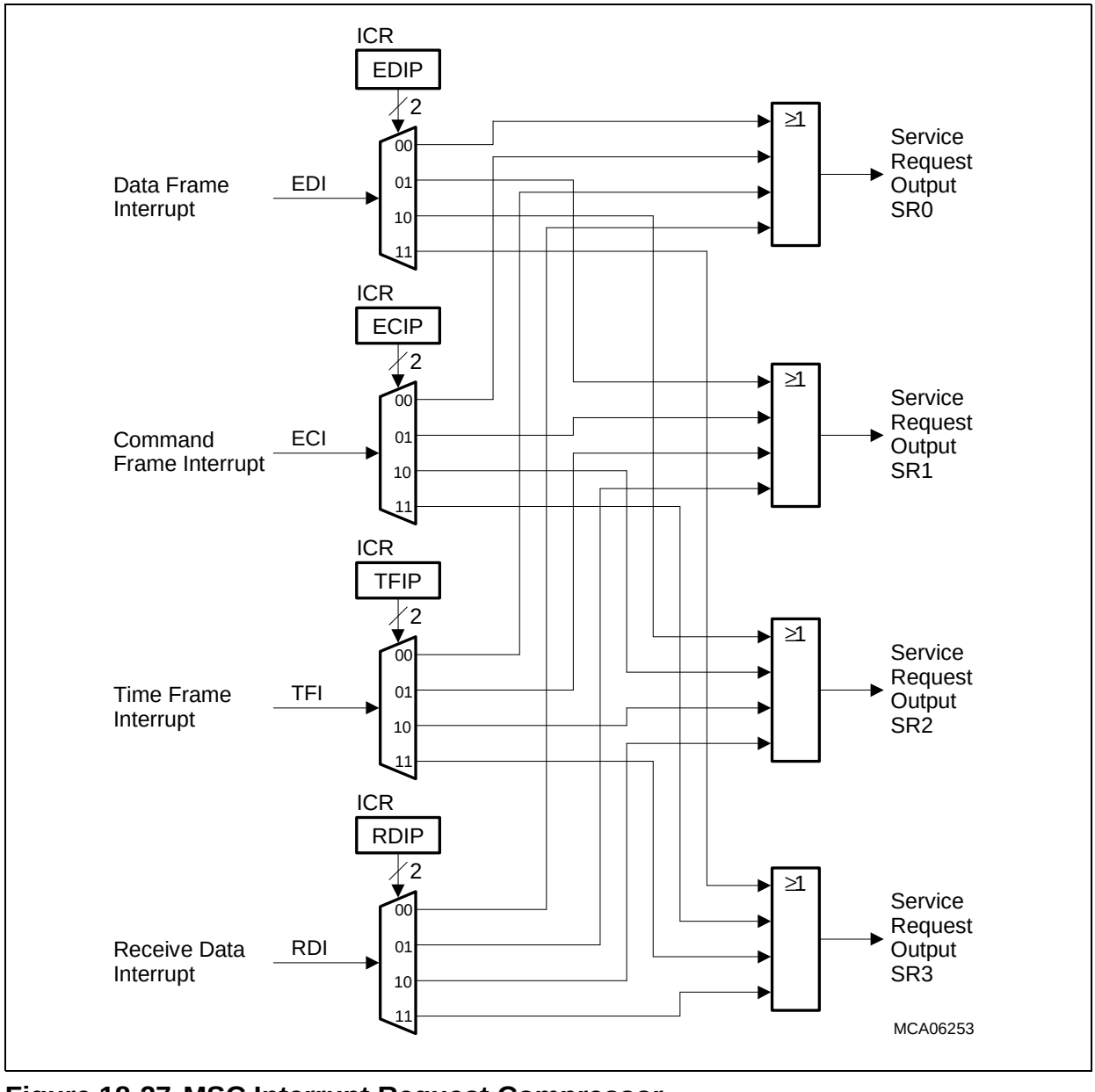


Figure 18-27 MSC Interrupt Request Compressor

*Note: The number of available MSC interrupt outputs depends on the implementation of the MSC module(s) in the specific product (see **Page 18-74** for TC1797 details).*

18.2 MSC Kernel Registers

This section describes the kernel registers of the MSC module. All MSC kernel register names described in this section will be referenced in other parts of the TC1797 User´s Manual by the module name prefix “MSC0_” for the MSC0 interface and “MSC1_” for the MSC1 interface.

All registers in the MSC address spaces are reset with the application reset (definition see SCU section “Reset Operation”).

MSC Kernel Register Overview

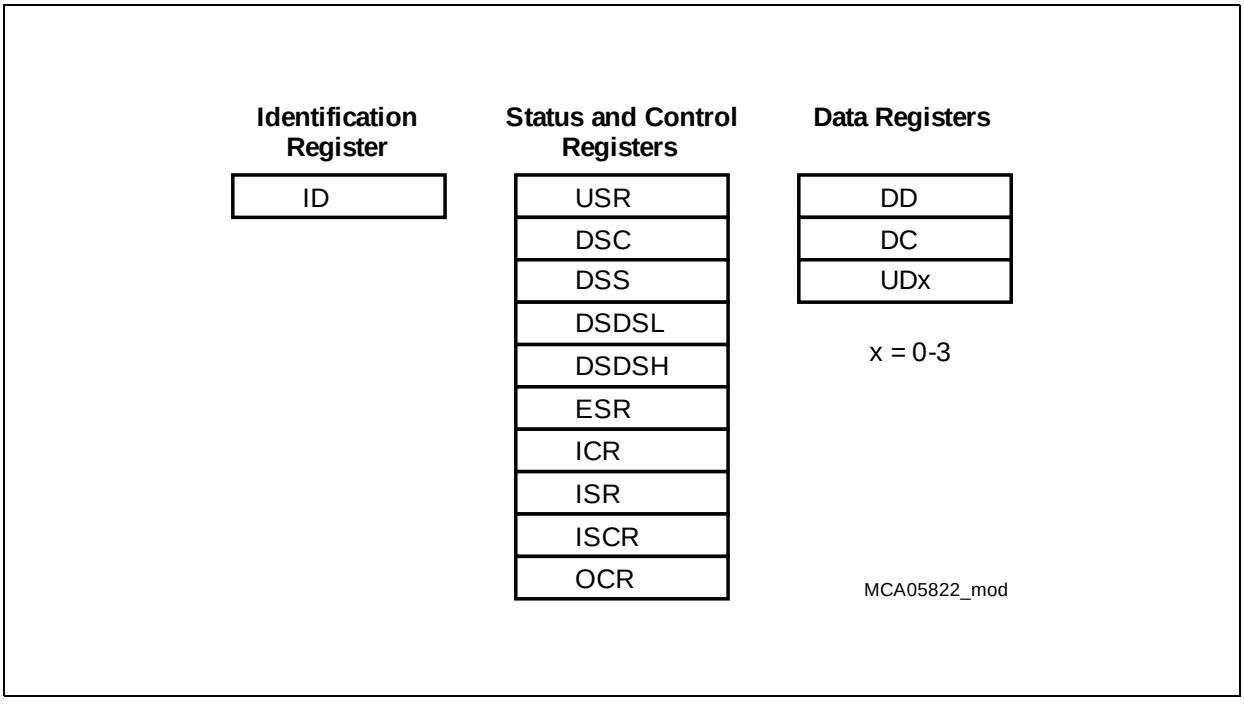


Figure 18-28 MSC Kernel Registers

The complete and detailed address map of the MSC0 module is described in **Table 18-12** on **Page 18-75**.

Table 18-8 Registers Address Space - MSC0 Kernel Registers

Module	Base Address	End Address	Note
MSC0	$F000\ 0800_H$	$F000\ 08FF_H$	–
MSC1	$F000\ 0900_H$	$F000\ 09FF_H$	–

Table 18-9 Registers Overview - MSC Kernel Registers

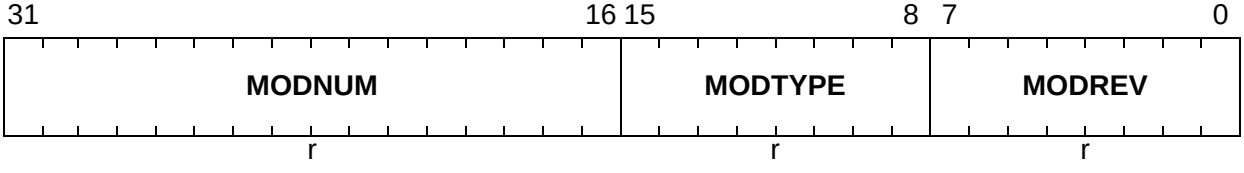
Register Short Name	Register Long Name	Offset Address[1)]	Description see
ID	Module Identification Register	08_H	Page 18-38
USR	Upstream Status Register	10_H	Page 18-39
DSC	Downstream Control Register	14_H	Page 18-41
DSS	Downstream Status Register	18_H	Page 18-44
DD	Downstream Data Register	$1C_H$	Page 18-59
DC	Downstream Command Register	20_H	Page 18-59
DSDSL	Downstream Select Data Source Low Register	24_H	Page 18-46
DSDSH	Downstream Select Data Source High Register	28_H	Page 18-47
ESR	Emergency Stop Register	$2C_H$	Page 18-48
UD0	Upstream Data Register 0	30_H	Page 18-60
UD1	Upstream Data Register 1	34_H	
UD2	Upstream Data Register 2	38_H	
UD3	Upstream Data Register 3	$3C_H$	
ICR	Interrupt Control Register	40_H	Page 18-49
ISR	Interrupt Status Register	44_H	Page 18-52
ISC	Interrupt Set Clear Register	48_H	Page 18-54
OCR	Output Control Register	$4C_H$	Page 18-56

1) The absolute register address is calculated as follows:
Module Base Address (Table 18-8) + Offset Address (shown in this column)

18.2.1 Module Identification Register

The MSC Module Identification Register ID contains read-only information about the module version.

ID
Module Identification Register **(08_H)** **Reset Value: $0028\ C0XX_H$**

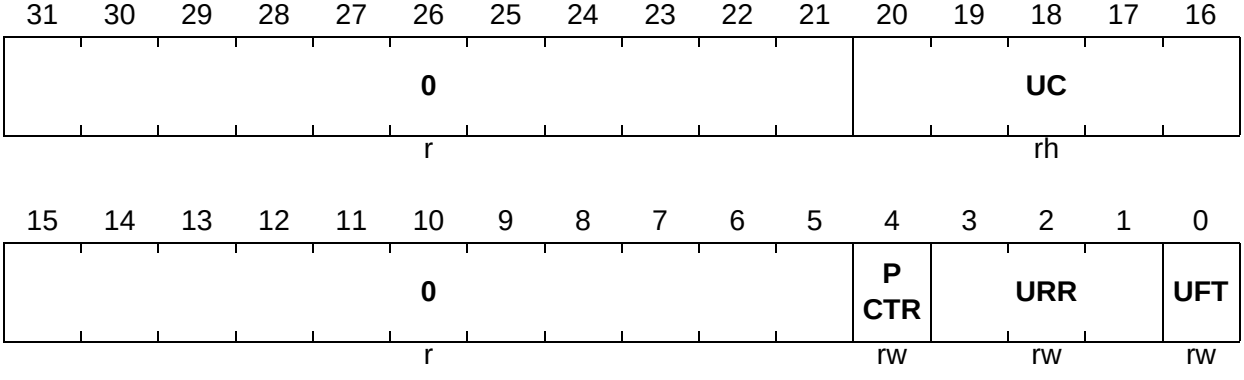


Field	Bits	Type	Description
MODREV	[7:0]	r	**Module Revision Number** This bit field defines the module revision number. The value of a module revision starts with 01_H (first revision).
MODTYPE	[15:8]	r	**Module Type** This bit field defines the module as a 32-bit module: $C0_H$
MODNUM	[31:16]	r	**Module Number Value** This bit field defines the module identification number for the MSC: 0028_H

18.2.2 Status and Control Registers

The Upstream Status Register is used to configure the upstream channel data format, baud rate, and parity type. It also provides the status information of the upstream counter (UC).

USR
Upstream Status Register (10_H) Reset Value: 0000 0000_H

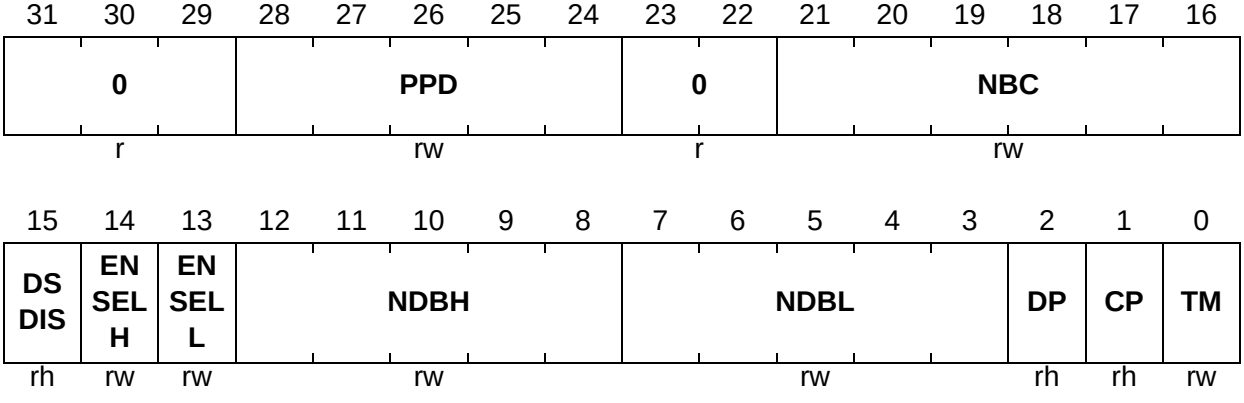


Field	Bits	Type	Description
UFT	0	rw	**Upstream Channel Frame Type** This bit determines the frame type used by the upstream channel for data reception. 0_B 12-bit upstream frame selected 1_B 16-bit upstream frame selected (with 4-bit address field)
URR	[3:1]	rw	**Upstream Channel Receiving Rate** This bit field determines the baud rate for the upstream channel. 000_B Upstream channel disabled; no reception is possible 001_B Baud rate = $f_{MSC}/4$ 010_B Baud rate = $f_{MSC}/8$ 011_B Baud rate = $f_{MSC}/16$ 100_B Baud rate = $f_{MSC}/32$ 101_B Baud rate = $f_{MSC}/64$ 110_B Baud rate = $f_{MSC}/128$ 111_B Baud rate = $f_{MSC}/256$

Field	Bits	Type	Description
PCTR	4	rw	**Parity Control** This bit determines the parity mode used by the upstream channel for data reception. 0_B Even parity mode is selected. A parity bit is set on an odd number of 1s in the serial address/data stream. 1_B Odd parity mode is selected. A parity bit is set on an even number of 1s in the serial address/data stream.
UC	[20:16]	rh	**Upstream Counter** This bit field indicates the content of the upstream counter that counts the bits during upstream channel reception.
0	[15:5], [31:21]	r	**Reserved** Read as 0; should be written with 0.

The Downstream Control Register is used to control the operation mode and frame layout of the downstream channel transmission. It also contains the two pending status bits.

DSC
Downstream Control Register (14_H) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
TM	0	rw	**Transmission Mode** This bit selects the transmission mode of the downstream channel. 0_B Triggered Mode selected 1_B Data Repetition Mode selected
CP	1	rh	**Command Pending** This bit is set when the downstream command register DC is written. CP is cleared when the first bit of the related command frame is sent out.
DP	2	rh	**Data Pending** In Triggered Mode, this bit is set when the set data pending bit ISC.SDP is set by software. In Data Repetition Mode, this bit is set by hardware at the last passive time frame. At the start of the data frame, DP is cleared by hardware.

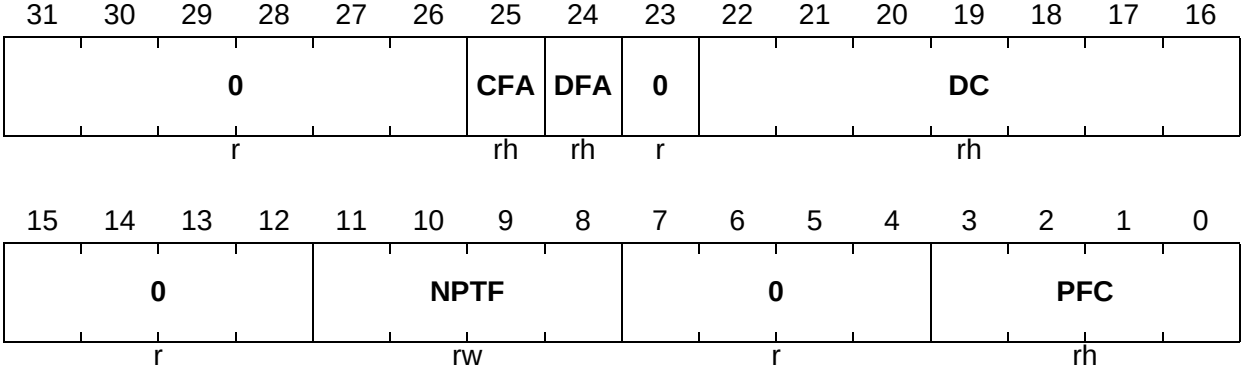
Field	Bits	Type	Description
NDBL	[7:3]	rw	**Number of SRL Bits Shifted at Data Frames** NDBL determines the number of shift register low part (SRL) bits that are shifted out on SO during a data frame. 00000_B No SRL bit shifted 00001_B SRL[0] shifted 00010_B SRL[1:0] shifted $…_B$ … 01111_B SRL[14:0] shifted 10000_B SRL[15:0] shifted Other bit combinations are reserved; do not use these bit combinations.
NDBH	[12:8]	rw	**Number of SRH Bits Shifted at Data Frames** NDBH determines the number of shift register high part (SRH) bits that are shifted out on SO during a data frame. 00000_B No SRH bit shifted; no selection bit is generated, the SRH active phase is completely skipped. 00001_B SRH[0] shifted 00010_B SRH[1:0] shifted $…_B$ … 01111_B SRH[14:0] shifted 10000_B SRH[15:0] shifted Other bit combinations are reserved; do not use these bit combinations.
ENSELL	13	rw	**Enable SRL Active Phase Selection Bit** This bit determines whether a low level selection bit is inserted at the beginning of a data frame's SRL active phase. 0_B No selection bit inserted. 1_B Low level selection bit inserted.
ENSELH	14	rw	**Enable SRH Active Phase Selection Bit** This bit determines whether a low level selection bit is inserted at the beginning of a data frame's SRH active phase. 0_B No selection bit inserted. 1_B Low level selection bit inserted.

Field	Bits	Type	Description
DSDIS	15	rh	**Downstream Disable** This bit indicates the state of the downstream channel operation. 0_B The downstream channel is enabled. A frame transmission can take place (Triggered Mode) or takes place (Data Repetition Mode). 1_B Downstream Counter becomes disabled. No new frame transmission is started. A running frame transmission is always completed.
NBC	[21:16]	rw	**Number of Bits Shifted at Command Frames** This bit field determines how many bits of the SRL/SRH shift registers are shifted out during transmission of a command frame. 000000_B No bit shifted 000001_B SRL[0] shifted 000010_B SRL[1:0] shifted 000011_B SRL[2:0] shifted $\ldots_B$ … 010000_B SRL[15:0] shifted 010001_B SRL[15:0] and SRH[0] shifted 010010_B SRL[15:0] and SRH[1:0] shifted $\ldots_B$ … 011111_B SRL[15:0] and SRH[14:0] shifted 100000_B SRL[15:0] and SRH[15:0] shifted Other bit combinations are reserved; do not use these bit combinations
PPD	[28:24]	rw	**Passive Phase Length at Data Frames** This bit field determines the length of the passive phase of a data frame. 00000_B Passive phase length is $2 \times t_{FCL}$ 00001_B Passive phase length is $2 \times t_{FCL}$ 00010_B Passive phase length is $2 \times t_{FCL}$ 00011_B Passive phase length is $3 \times t_{FCL}$ $\ldots_B$ … 11111_B Passive phase length is $31 \times t_{FCL}$
0	[23:22], [31:29]	r	**Reserved** Read as 0; should be written with 0.

Note: The “rw” bits in the DSC register are buffered in a shadow buffer at the start of a corresponding frame transmission.

The Downstream Status Register DSS contains counter bit fields, status bits, and indicates the number of passive time frames.

DSS
Downstream Status Register (18_H) Reset Value: 0000 0000_H

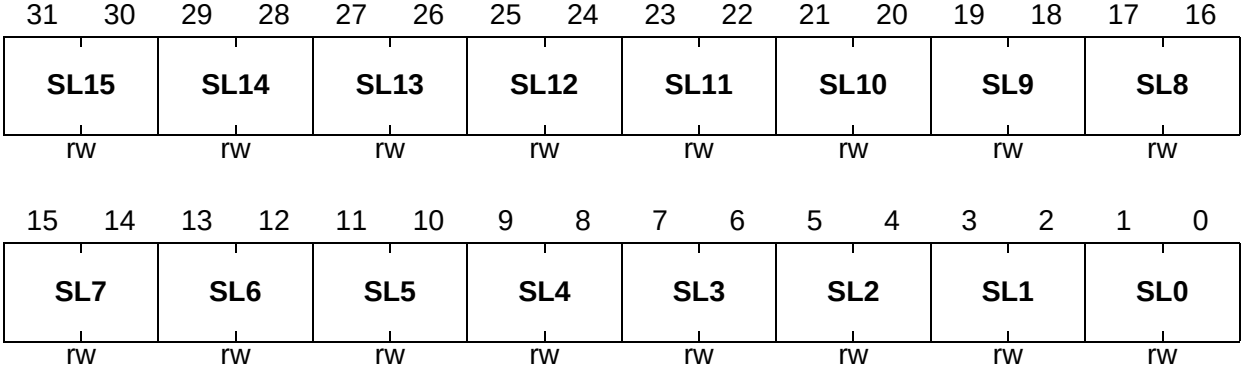


Field	Bits	Type	Description
PFC	[3:0]	rh	**Passive Time Frame Counter** In Data Repetition Mode, this bit field indicates the count of passive time frames that are currently transmitted. In Triggered Mode PFC remains at 0000_B. 0000_B Data frame is transmitted. 0001_B First passive time frame is transmitted. 0010_B Second passive time frame is transmitted. $\ldots_B$ … 1111_B Fifteenth passive time frame is transmitted.
NPTF	[11:8]	rw	**Number Of Passive Time Frames** This bit field indicates the number of passive time frames that are inserted in Data Repetition Mode between two data frames. 0000_B No passive time frame inserted. 0001_B One passive time frame inserted. 0010_B Two passive time frames inserted. $\ldots_B$ … 1111_B Fifteen passive time frames inserted. *Note: NPTF is buffered in a shadow buffer at the start of each data frame.*

Field	Bits	Type	Description
DC	[22:16]	rh	**Downstream Counter** This bit field indicates the number of downstream shift clock periods that have been elapsed since the start of the current frame. 00_H No shift clock elapsed (after counter reset). 01_H 1 shift clock elapsed. $\ldots_H$ … $7F_H$ 127 shift clocks elapsed. DC is reset at the end of a downstream frame.
DFA	24	rh	**Data Frame Active** This bit indicates if a data frame is currently sent out. 0_B No data frame is currently sent out. 1_B A data frame is currently sent out.
CFA	25	rh	**Command Frame Active** This bit indicates if a command frame is currently sent out. 0_B No command frame is currently sent out. 1_B A command frame is currently sent out.
0	[7:4], [15:12], 23, [31:26]	r	**Reserved** Read as 0; should be written with 0.

The bit fields of the Downstream Select Data Low Register DSDSL determine the data source for each bit in shift register SRL.

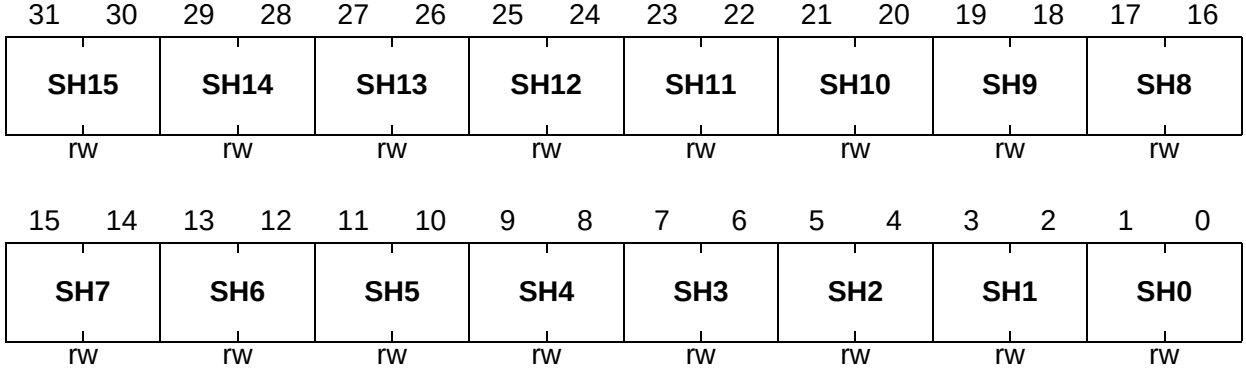
DSDSL
Downstream Select Data Source Low Register
(24_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
SLx **(x = 0-15)**	[2*x+1: 2*x]	rw	**Select Source for SRL** SLx determines which data source is used for the shift register bit SRL[x] during data frame transmission. 00_B SRL[x] is taken from data register DD.DDL[x]. 01_B Reserved. 10_B SRL[x] is taken from the ALTINL input line x. 11_B SRL[x] is taken from the ALTINL input line x in inverted state.

The bit fields of the Downstream Select Data Source High Register DSDSH determine the data source for each bit in shift register SRH.

DSDSH
Downstream Select Data Source High Register
(28_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
SHx **(x = 0-15)**	[2*x+1: 2*x]	rw	**Select Source for SRH** SHx determines which data source is used for the shift register bit SRH[x] during data frame transmission. 00_B SRH[x] is taken from data register DD.DDH[x]. 01_B Reserved. 10_B SRH[x] is taken from the ALTINH input line x. 11_B SRH[x] is taken from the ALTINH input line x in inverted state.

The Emergency Stop Register ESR determines which bits of SRL and SRH are enabled for emergency operation.

ESR
Emergency Stop Register **($2C_H$)** **Reset Value: 0000 0000_H**

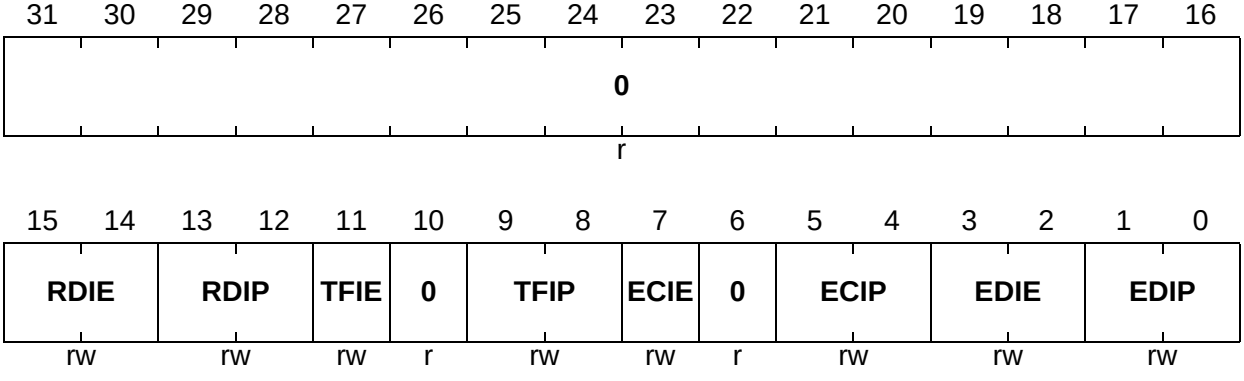
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ENH 15	**ENH 14**	**ENH 13**	**ENH 12**	**ENH 11**	**ENH 10**	**ENH 9**	**ENH 8**	**ENH 7**	**ENH 6**	**ENH 5**	**ENH 4**	**ENH 3**	**ENH 2**	**ENH 1**	**ENH 0**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ENL 15	**ENL 14**	**ENL 13**	**ENL 12**	**ENL 11**	**ENL 10**	**ENL 9**	**ENL 8**	**ENL 7**	**ENL 6**	**ENL 5**	**ENL 4**	**ENL 3**	**ENL 2**	**ENL 1**	**ENL 0**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
ENLx (x = 0-15)	x	rw	**Emergency Stop Enable for Bit x in SRL** This bit enables the emergency stop feature selectively for each SRL bit. If the emergency stop condition is met and enabled (ENLx = 1), the SRL[x] bit is of the data register DD.DDL[x] is used for the shift register load operation. 0_B Emergency stop feature for bit SRL[x] is disabled. 1_B The emergency stop feature for bit SRL[x] is enabled.
ENHx (x = 0-15)	x+16	rw	**Emergency Stop Enable for Bit x in SRH** This bit enables the emergency stop feature selectively for each SRH bit. If the emergency stop condition is met and enabled (ENHx = 1), the SRH[x] bit of the data register DD.DDH[x] is used for the shift register load operation. 0_B Emergency stop feature for bit SRH[x] is disabled. 1_B The emergency stop feature for bit SRH[x] is enabled.

The Interrupt Control Register ICR holds the interrupt enable bits and interrupt pointers of all four MSC interrupts.

ICR
Interrupt Control Register **(40_H)** **Reset Value: 0000 0000_H**



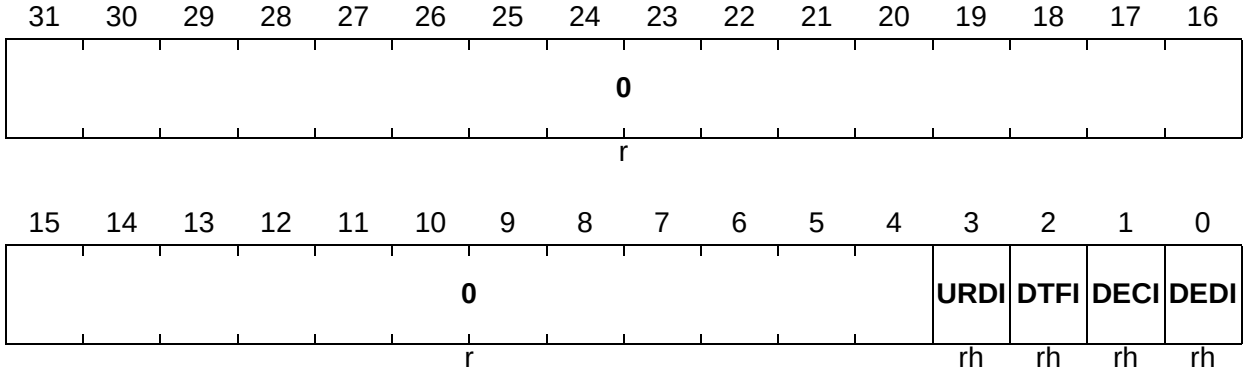
Field	Bits	Type	Description
EDIP	[1:0]	rw	**Data Frame Interrupt Node Pointer** EDIP selects the service request output line SRn (n = 0-3) for the data frame interrupt. 00_B Service request output SR0 selected 01_B Service request output SR1 selected 10_B Service request output SR2 selected 11_B Service request output SR3 selected
EDIE	[3:2]	rw	**Data Frame Interrupt Enable** This bit field determines the enable conditions for the data frame interrupt. 00_B Interrupt generation disabled 01_B An interrupt is generated when the last data bit has been shifted out. 10_B An interrupt is generated when the first data bit has been shifted out, but only if DSC.NDBL is not equal 00000_B. This means, at least one SRL bit must be shifted out for the first data bit shifted interrupt to become active. 11_B Interrupt generation disabled

Field	Bits	Type	Description
ECIP	[5:4]	rw	**Command Frame Interrupt Node Pointer** ECIP selects the service request output line SRn (n = 0-3) for the command frame interrupt. 00_B Service request output SR0 selected 01_B Service request output SR1 selected 10_B Service request output SR2 selected 11_B Service request output SR3 selected
ECIE	7	rw	**Command Frame Interrupt Enable** This bit enables the command frame interrupt. 0_B Interrupt generation disabled. 1_B Interrupt generation enabled.
TFIP	[9:8]	rw	**Time Frame Interrupt Pointer** TFIP selects the service request output line SRn (n = 3-0) for the time frame interrupt. 00_B Service request output SR0 selected 01_B Service request output SR1 selected 10_B Service request output SR2 selected 11_B Service request output SR3 selected
TFIE	11	rw	**Time Frame Interrupt Enable** This bit enables the time frame interrupt. 0_B Interrupt generation disabled. 1_B Interrupt generation enabled.
RDIP	[13:12]	rw	**Receive Data Interrupt Pointer** RDIP selects the service request output line SRn (n = 3-0) for the receive data interrupt. 00_B Service request output SR0 selected 01_B Service request output SR1 selected 10_B Service request output SR2 selected 11_B Service request output SR3 selected

Field	Bits	Type	Description
RDIE	[15:14]	rw	**Receive Data Interrupt Enable** This bit field determines the enable conditions for the receive data interrupt. 00_B Interrupt generation disabled. 01_B An interrupt is generated when data is received and written into the upstream data registers UDx (x = 0-3). 10_B An interrupt is generated as with RDIE = 01_B but only if the received data is not equal to 00_H. 11_B An interrupt is generated when data is received and written into register UD3.
0	6, 10, [31:16]	r	**Reserved** Read as 0; should be written with 0.

The Interrupt Status Register ISR holds the interrupt status flags that indicate an interrupt occurrence in downstream and upstream channels.

ISR
Interrupt Status Register **(44_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
DEDI	0	rh	**Data Frame Interrupt Flag** This flag is always set by hardware when a downstream channel data frame interrupt is generated. DEDI can be set or cleared by software when writing to register ISC with the appropriate bits ISC.SDEDI or ISC.CDEDI set.
DECI	1	rh	**Command Frame Interrupt Flag** This flag is always set by hardware when a downstream channel command frame interrupt is generated, whether or not it is enabled. DECI can be set or cleared by software when writing to register ISC with the appropriate bits SDECI or CDECI set.
DTFI	2	rh	**Time Frame Interrupt Flag** This flag is always set by hardware when a downstream channel time frame interrupt is generated, whether or not it is enabled. DTFI can be set or cleared by software when writing to register ISC with the appropriate bits SDTFI or CDTFI set.

Field	Bits	Type	Description
URDI	3	rh	**Receive Data Interrupt Flag** This flag is always set by hardware when an upstream channel receive data interrupt is generated, whether or not it is enabled. URDI can be set or cleared by software when writing to register ISC with the appropriate bits SURDI or CURDI set.
0	[31:4]	r	**Reserved** Read as 0; should be written with 0.

The Interrupt Set Clear Register ISC is used to set or clear the MSC interrupt flags located in the Interrupt Status Register ISR. Reading ISC always returns 0000 0000$_H$.

ISC
Interrupt Set Clear Register (48$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0									SDDIS	SCP	SDP	SURDI	SDTFI	SDECI	SDEDI
r									w	w	w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0									CDDIS	CCP	CDP	CURDI	CDTFI	CDECI	CDEDI
r									w	w	w	w	w	w	w

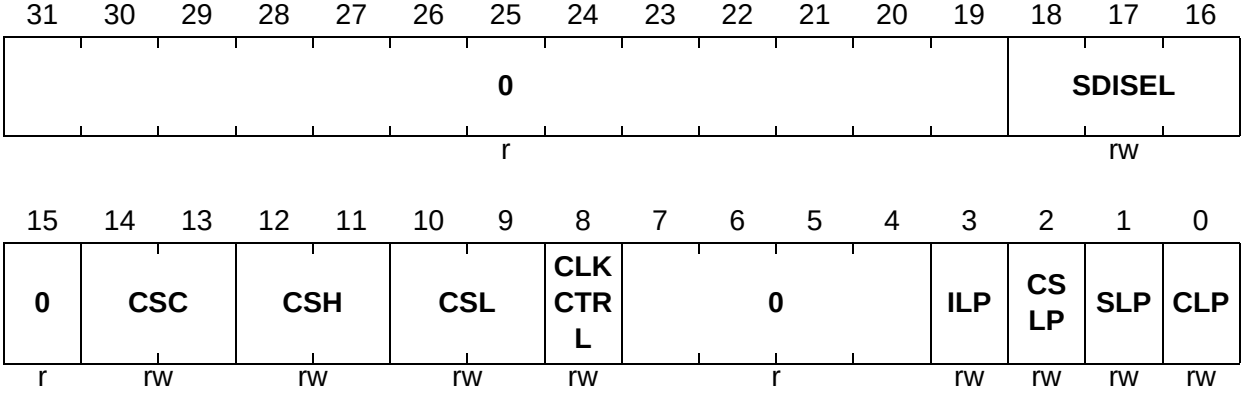
Field	Bits	Type	Description
CDEDI	0	w	**Clear DEDI Flag** 0_B No operation 1_B Bit ISR.DEDI is cleared.
CDECI	1	w	**Clear DECI Flag** 0_B No operation 1_B Bit ISR.DECI is cleared.
CDTFI	2	w	**Clear DTFI Flag** 0_B No operation 1_B Bit ISR.DTFI is cleared.
CURDI	3	w	**Clear URDI Flag** 0_B No operation 1_B Bit ISR.URDI is cleared.
CDP	4	w	**Clear DP Flag** 0_B No operation 1_B Bit DSC.DP is cleared.
CCP	5	w	**Clear CP Flag** 0_B No operation 1_B Bit DSC.CP is cleared.
CDDIS	6	w	**Clear DSDIS Flag** 0_B No operation 1_B Bit DSC.DSDIS is cleared.

Field	Bits	Type	Description
SDEDI	16	w	**Set DEDI Flag** 0_B No operation 1_B Bit ISR.DEDI is set.
SDECI	17	w	**Set DECI Flag** 0_B No operation 1_B Bit ISR.DECI is set.
SDTFI	18	w	**Set DTFI Flag** 0_B No operation 1_B Bit ISR.DTFI is set.
SURDI	19	w	**Set URDI Flag** 0_B No operation 1_B Bit ISR.URDI is set.
SDP	20	w	**Set DP Bit** 0_B No effect 1_B Bit DSC.DP is set.
SCP	21	w	**Set CP Flag** 0_B No operation 1_B Bit DSC.CP is set.
SDDIS	22	w	**Set DSDIS Flag** 0_B No operation 1_B Bit DSC.DSDIS is set.
0	[15:7], [31:23]	r	**Reserved** Read as 0; should be written with 0.

Note: When the ISC register is written with both bits (set and reset bit) for a specific interrupt flag, the clear operation takes place and the set operation is ignored.

The Output Control Register OCR determines the MSC input/output signal polarities, the chip select output signal assignment, and the serial output clock generation.

OCR
Output Control Register **($4C_H$)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
CLP	0	rw	**FCLP Line Polarity** 0_B FCLP and FCL signal polarity is identical. FCLN signal has inverted FCL signal polarity. 1_B FCLP signal has inverted FCL signal polarity. FCLN and FCL signal polarities are identical.
SLP	1	rw	**SOP Line Polarity** 0_B SOP and SO signal polarity is identical. SON signal has inverted SO signal polarity. 1_B SOP signal has inverted SO signal polarity. SON and SO signal polarities are identical.
CSLP	2	rw	**Chip Selection Lines Polarity** 0_B EN[3:0] and ENL, ENH, ENC signal polarities are identical (high active). 1_B EN[3:0] signal polarities are inverted (low active) to the ENL, ENH, ENC signal polarities. Bit CSLP is buffered during a frame transmission. This means that any change of CSLP becomes valid first with the start of the next frame transmission.
ILP	3	rw	**SDI Line Polarity** 0_B SDI and SI signal polarities are identical. 1_B SDI and SI signal polarities are inverted.

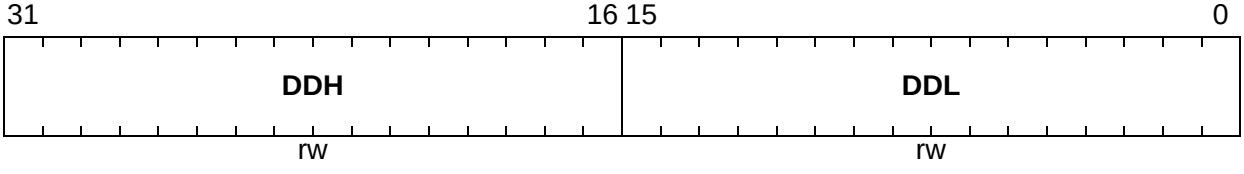
Field	Bits	Type	Description
CLKCTRL	8	rw	**Clock Control** This bit determines the activation of clock output FCL. 0_B FCL is activated only during the active phases of data or command frames (not during passive time frames). 1_B FCL is always active whether or not a downstream frame is currently transmitted.
CSL	[10:9]	rw	**Chip Enable Selection for ENL** This bit field selects the chip enable output ENx that becomes active during the SRL active phase (ENL = 1) of a data frame. The active level of ENx is defined by bit CSLP. 00_B EN0 line is selected for ENL. 01_B EN1 line is selected for ENL. 10_B EN2 line is selected for ENL. 11_B EN3 line is selected for ENL.
CSH	[12:11]	rw	**Chip Enable Selection for ENH** This bit field selects the chip enable output ENx that becomes active during the SRH active phase (ENH = 1) of a data frame. The active level of ENx is defined by bit CSLP. 00_B EN0 line is selected for ENH. 01_B EN1 line is selected for ENH. 10_B EN2 line is selected for ENH. 11_B EN3 line is selected for ENH.
CSC	[14:13]	rw	**Chip Enable Selection for ENC** This bit field selects the chip enable output ENx that becomes active during the active phase (ENC = 1) of a command frame. The active level of ENx is defined by bit CSLP. 00_B EN0 line is selected for ENC. 01_B EN1 line is selected for ENC. 10_B EN2 line is selected for ENC. 11_B EN3 line is selected for ENC.

Field	Bits	Type	Description
SDISEL	[18:16]	rw	**Serial Data Input Selection** This bit field selects the source for the serial data input SDI of the upstream channel. 000_B SDI[0] input is selected for SDI. 001_B SDI[1] input is selected for SDI. 010_B SDI[2] input is selected for SDI. 011_B SDI[3] input is selected for SDI. 100_B SDI[4] input is selected for SDI. 101_B SDI[5] input is selected for SDI. 110_B SDI[6] input is selected for SDI. 111_B SDI[7] input is selected for SDI.
0	[7:4], 15, [31:19]	r	**Reserved** Read as 0; should be written with 0.

18.2.3 Data Registers

The Downstream Data Register DD contains data to be transmitted during data frames.

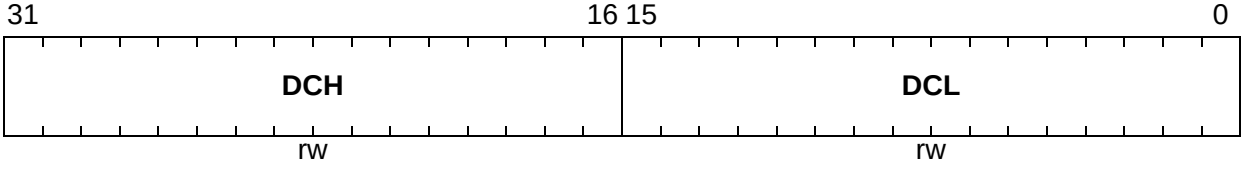
DD
Downstream Data Register (1C$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
DDL	[15:0]	rw	**Downstream Data for SRL Shift Register** Contains the data bits to be transmitted during the SRL active phase of a data frame.
DDH	[31:16]	rw	**Downstream Data for SRH Shift Register** Contains the data bits to be transmitted during the SRH active phase of a data frame.

The Downstream Command Register DC contains command information to be transmitted during command frames.

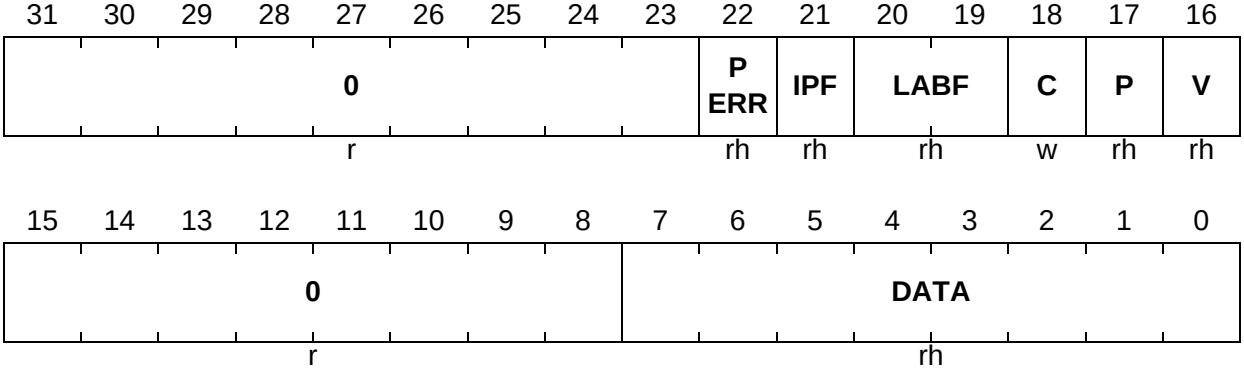
DC
Downstream Command Register (20$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
DCL	[15:0]	rw	**Downstream Command for SRL Shift Register** Contains the data bits to be transmitted during the SRL active phase of a command frame.
DCH	[31:16]	rw	**Downstream Command for SRH Shift Register** Contains the data bits to be transmitted during the SRH active phase of a command frame.

The four Upstream Data Registers UDx store the content (data, addresses, received and calculated parity bit, parity error bit) of a received upstream channel data frame.

UDx (x = 0-3)
Upstream Data Register x **(30_H+x*4_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
DATA	[7:0]	rh	**Received Data** This bit field contains the 8-bit receive data.
V	16	rh	**Valid Bit** This bit is set by hardware when the received data is written to UDx. Writing bit C = 1 clears V. If hardware setting and software clearing of the valid bit occur simultaneously, bit V will be cleared.
P	17	rh	**Parity Bit** This flag contains the parity bit that has been received with the data frame.
C	18	w	**Clear Bit** 0_B No operation. 1_B Bit V is cleared. C is always read as 0.
LABF	[20:19]	rh	**Lower Address Bit Field** This bit field contains the two address bits A[1:0] of the 4-bit address field (16-bit data frame). If 12-bit data frame is selected, LABF is always set to 00_B.
IPF	21	rh	**Internal Parity Flag** This bit contains the parity bit that has been calculated in the MSC during data frame reception.

Field	Bits	Type	Description
PERR	22	rh	**Parity Error** This bit indicates if a start bit error, parity error, or stop bit error occurred during frame reception. 0_B No error detected. 1_B Error detected.
0	[15:8], [31:23]	r	**Reserved** Read as 0; should be written with 0.

18.3 MSC Module Implementation

This section describes the MSC module interfaces as they are implemented in the TC1797. It especially covers clock control, port and on-chip connections, interrupt control, and address decoding.

18.3.1 Interface Connections of the MSC Module

Figure 18-29 shows the TC797-specific implementation details and interconnections of the MSC0 and MSC1 modules.

Each MSC module is supplied with a separate clock control, address decoding, and interrupt control logic. Two of the four modules' service request outputs are connected with interrupt nodes, and two with the DMA controller. Outputs of the GPTA module are connected to the alternate input buses ALTINL/ALTINH. The emergency stop output from the SCU controls the corresponding inputs of both MSC modules.

The serial data and clock outputs of the downstream channels of each MSC module are connected to combined GPIO/LVDS differential output drivers. Details about these eight Port 5 pins are defined in the ports chapter (see also **Table 18-10**).

Additionally, the positive serial clock (FCLP) and positive data output (SOP) are available at GPIO lines of Port 9. The device select outputs (ENxy) are wired to GPIO lines of Port 5 and Port 9. One Port 5 input line is connected to the upstream channel serial data input.

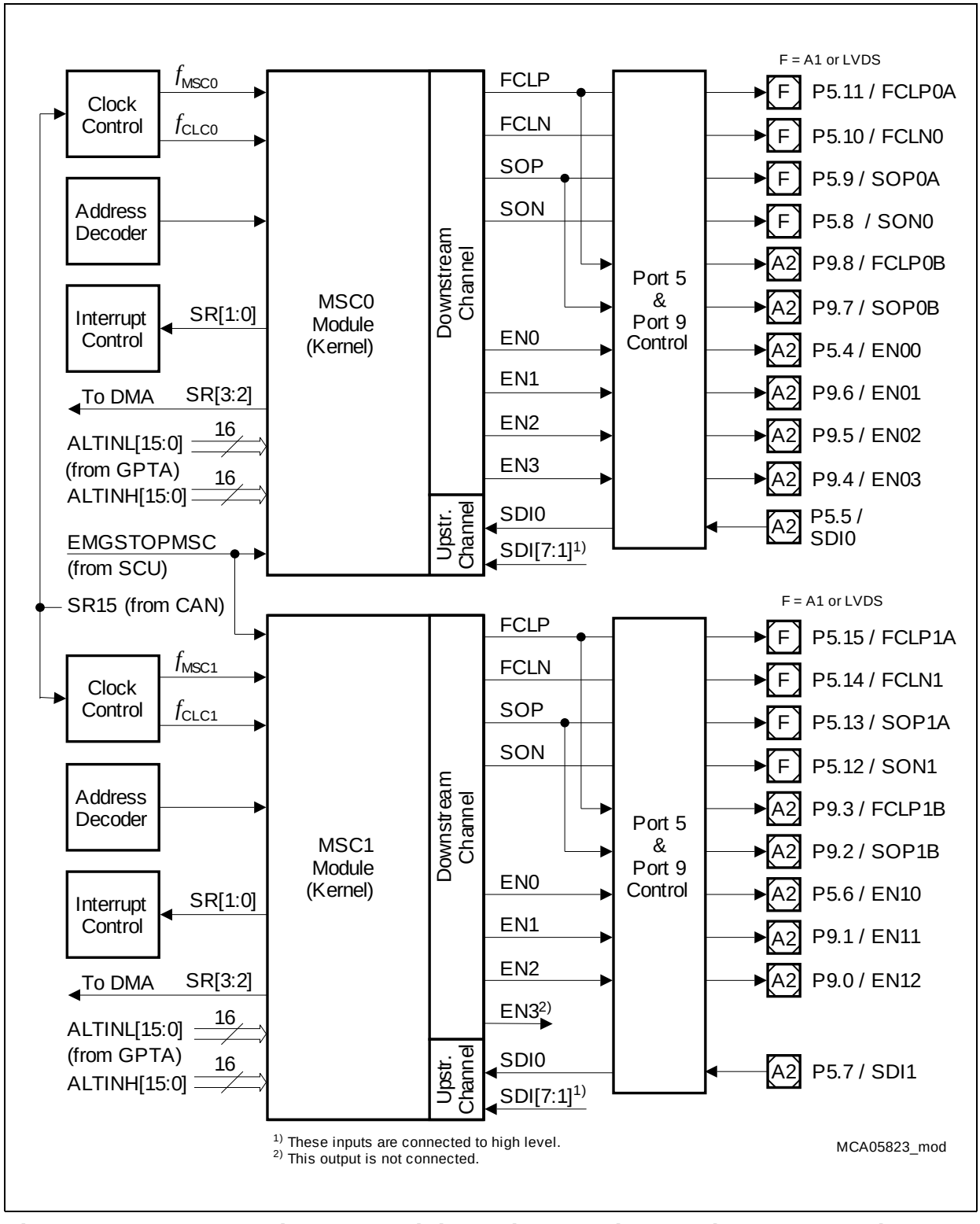


Figure 18-29 MSC0 and MSC1 Module Implementations and Interconnections

18.3.2 MSC0/MSC1 Module-Related External Registers

Figure 18-30 summarizes the module-related external registers which are required for MSC programming (see also **Figure 18-28** for the module kernel specific registers). These registers are described in the following sections.

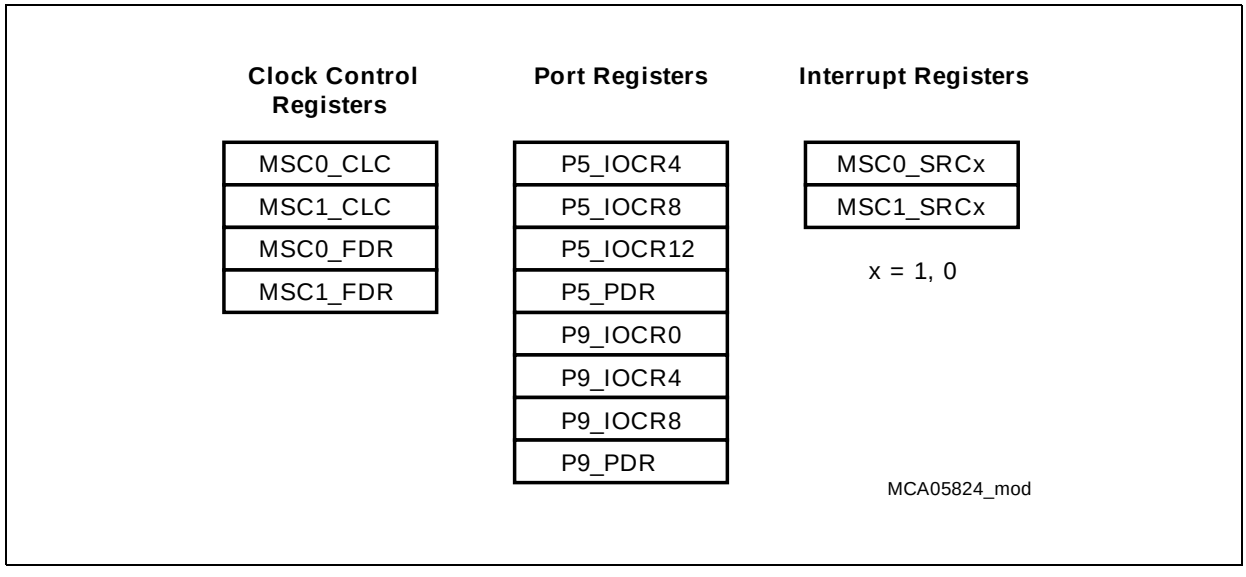


Figure 18-30 MSC Implementation-specific Special Function Registers

18.3.3 Clock Control

The MSC modules are provided with two independent clock signals (**Figure 18-31**):

- f_{CLC0} and f_{CLC1}
  These are the module clocks that are used inside the MSC kernel for control purposes such as clocking of control logic and register operations. The frequency of f_{CLC0} and f_{CLC1} is always identical to the system clock frequency f_{SYS}. The clock control registers MSC0_CLC and MSC1_CLC make it possible to enable/disable f_{CLC0} and f_{CLC1} under certain conditions.
- f_{MSC0} and f_{MSC1}
  These clocks are the module clocks that are used inside the MSC for baud rate generation of the serial upstream and downstream channel. The fractional divider registers MSC0_FDR and MSC1_FDR control the frequency of f_{MSC0} and f_{MSC1} and make it possible to enable/disable it independently of f_{CLC0} and f_{CLC1}.

For module test purposes only, the service request output SR15 of the MultiCAN controller makes it possible to synchronize the fractional divider clock generation of both MSC modules to external events. This feature should not be used for normal MSC operation.

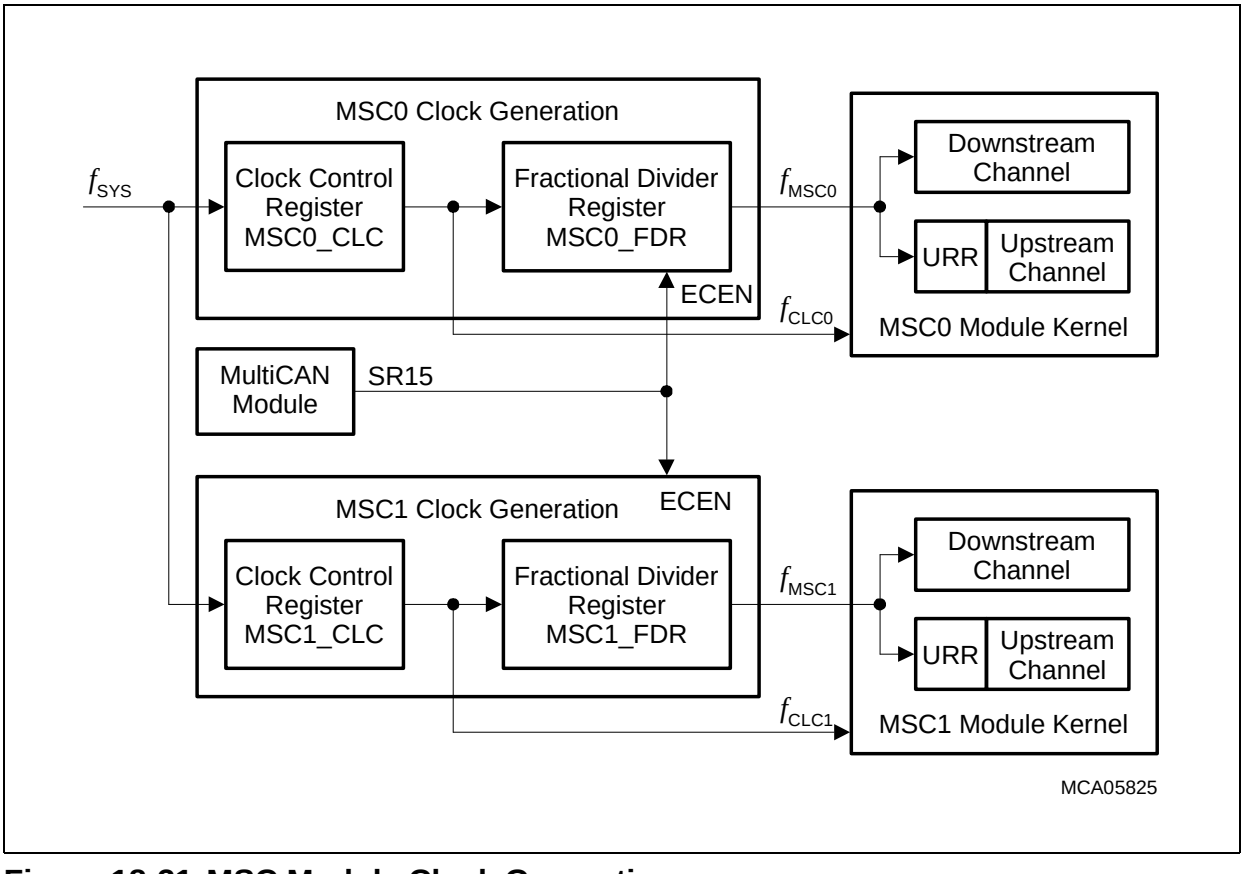


Figure 18-31 MSC Module Clock Generation

The following two formulas define the frequency of f_{MSC0}:

$$f_{MSC0} = f_{SYS} \times \frac{1}{n} \text{ with n = 1024 - MSC0.FDR.STEP} \quad (18.3)$$

$$f_{MSC0} = f_{SYS} \times \frac{n}{1024} \text{ with n = 0-1023} \quad (18.4)$$

Downstream Channel Baud Rate

As the clock signal FCL of the synchronous downstream channel is always half the frequency of f_{MSC0}, the resulting downstream channel baud rate is defined by:

$$\text{Baud rate}_{MSC0} = f_{SYS} \times \frac{1}{2 \times (1024 - \text{MSC0.FDR.STEP})} \quad (18.5)$$

$$\text{Baud rate}_{MSC0} = f_{SYS} \times \frac{\text{MSC0.FDR.STEP}}{2 \times 1024} \quad (18.6)$$

Upstream Channel Baud Rate

The baud rate of the asynchronous upstream channel is derived from the module clock f_{MSC0} by a programmable clock divider selected by bit field MSC0_USR.URR (see also **Equation (18.2)** on **Page 18-25**). The divide factor DF can be at minimum 4 and at maximum 256.

$$\text{Baud rate}_{MSC0} = f_{SYS} \times \frac{1}{DF \times (1024 - \text{MSC0.FDR.STEP})} \quad (18.7)$$

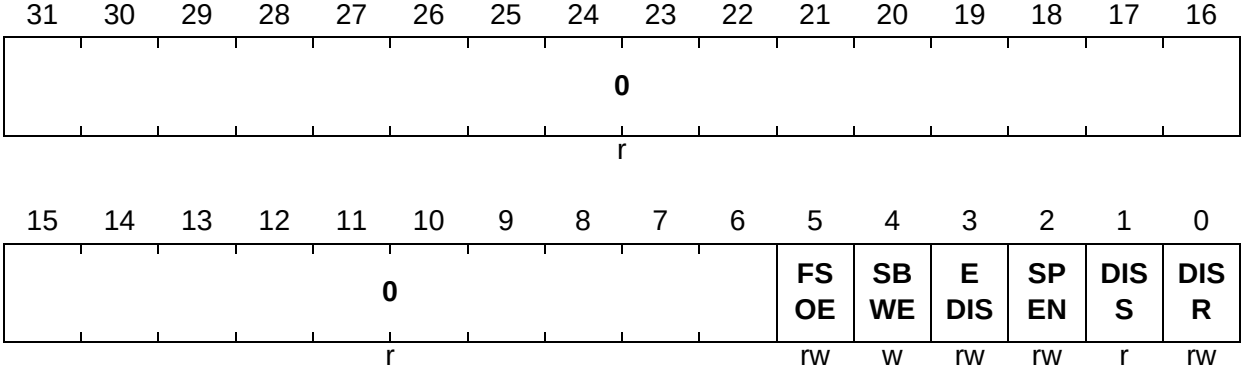
$$\text{Baud rate}_{MSC0} = f_{SYS} \times \frac{\text{MSC0.FDR.STEP}}{DF \times 1024} \quad (18.8)$$

Equation (18.3), **Equation (18.5)**, and **Equation (18.7)** are valid for normal divider mode (MSC0.FDR.DM = 01_B). **Equation (18.4)**, **Equation (18.6)**, and **Equation (18.8)** are valid for fractional divider mode (MSC0.FDR.DM = 10_B).

18.3.3.1 Clock Control Register

The Clock Control Register allows the programmer to control (enable/disable) the clock signals to the MSC0 module under certain conditions. The diagram below shows the clock control register functionality as is implemented for the MSC0 and MSC1 modules.

MSC0_CLC
MSC0 Clock Control Register (00_H) Reset Value: 0000 0003$_H$
MSC1_CLC
MSC1 Clock Control Register (00_H) Reset Value: 0000 0003$_H$



Field	Bits	Type	Description
DISR	0	rw	**Module Disable Request Bit** Used for enable/disable control of the module.
DISS	1	r	**Module Disable Status Bit** Bit indicates the current status of the module.
SPEN	2	rw	**Module Suspend Enable for OCDS** Used to enable the suspend mode
EDIS	3	rw	**Sleep Mode Enable Control** Used to control module's sleep mode.
SBWE	4	w	**Module Suspend Bit Write Enable for OCDS** Determines whether SPEN and FSOE are write-protected.
FSOE	5	rw	**Fast Switch Off Enable** Used to switch off fast clock in Suspend Mode.
0	[31:6]	r	**Reserved** Read as 0; should be written with 0.

Note: After a hardware reset operation, the f_{CLC0} and f_{MSC0} clocks are switched off and the MSC modules are disabled (DISS set).

18.3.3.2 Fractional Divider Register

The Fractional Divider Registers control the clock rate of the shift clock f_{MSC0} and f_{MSC1}. Each MSC module has its own fractional divider register.

MSC0_FDR
MSC0 Fractional Divider Register ($0C_H$) Reset Value: 0000 0000$_H$
MSC1_FDR
MSC1 Fractional Divider Register ($0C_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DISCLK	ENHW	SUSREQ	SUSACK	0		RESULT									
rwh	rw	rh	rh	r		rh									

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DM		SC		SM	0	STEP									
rw		rw		rw	r	rw									

Field	Bits	Type	Description
STEP	[9:0]	rw	**Step Value** Reload or addition value for RESULT.
SM	11	rw	**Suspend Mode** SM selects between granted or immediate suspend mode.
SC	[13:12]	rw	**Suspend Control** This bit field determines the behavior of the fractional divider in suspend mode.
DM	[15:14]	rw	**Divider Mode** DM selects normal or fractional divider mode.
RESULT	[25:16]	rh	**Result Value** Bit field for the addition result.
SUSACK	28	rh	**Suspend Mode Acknowledge** Indicates state of SPNDACK signal.
SUSREQ	29	rh	**Suspend Mode Request** Indicates state of SPND signal.
ENHW	30	rw	**Enable Hardware Clock Control** Controls operation of ECEN input and DISCLK bit.

Field	Bits	Type	Description
DISCLK	31	rwh	**Disable Clock** Hardware controlled disable for f_{OUT} signal.
0	10, [27:26]	r	**Reserved** Read as 0; should be written with 0.

18.3.4 Port Control

MSC0 and MSC1 clock and data output lines are connected to dedicated differential output drivers. Some of the MSC module I/O lines are connected to I/O ports and therefore controlled in the port logic (see also **Figure 18-29**). The following port control operations selections must be executed for these I/O lines:

- Input/output function selection (IOCR registers)
- Pad driver characteristics selection for the outputs (PDR registers)

18.3.4.1 Input/Output Function Selection

The port input/output control registers contain the bit fields that select the digital output and input driver characteristics such as port direction (input/output) with alternate output selection, pull-up/down devices, and open-drain selections. The I/O lines for the MSC modules are controlled by the Port 5 and Port 9 input/output control registers.

Table 18-10 shows in an overview how bits and bit fields must be programmed for the required I/O functionality of the MSC I/O lines.

Table 18-10 MSC0 and MSC1 I/O Line Selection and Setup

Module	Port Lines	Input/Output Control Register Bits	I/O
MSC0	P5.8 / SON0	P5_IOCR8.PC8 = $1X01_B$ P5_PDR.PD2 = $0XX_B$	CMOS Output[1)]
		P5_IOCR8.PC8 = $1XXX_B$ P5_PDR.PD2 = $1XX_B$	LVDS Output
	P5.9 / SOP0A	P5_IOCR8.PC9 = $1X01_B$ P5_PDR.PD2 = $0XX_B$	CMOS Output[1)]
		P5_IOCR8.PC9 = $1XXX1_B$ P5_PDR.PD2 = $1XX_B$	LVDS Output
	P5.10 / FCLN0	P5_IOCR8.PC10 = $1X01_B$ P5_PDR.PD2 = $0XX_B$	CMOS Output[1)]
		P5_IOCR8.PC10 = $1XXX_B$ P5_PDR.PD2 = $1XX_B$	LVDS Output
	P5.11 / FCLP0A	P5_IOCR8.PC11 = $1X01_B$ P5_PDR.PD2 = $0XX_B$	CMOS Output[1)]
		P5_IOCR8.PC11 = $1XXX_B$ P5_PDR.PD2 = $1XX_B$	LVDS Output
	P5.4 / EN00	P5_IOCR4.PC12 = $1X01_B$	Output
	P5.5 / SDI0[2)]	P5_IOCR4.PC5 = $0XXX_B$	Input
	P9.4 / EN03	P9_IOCR4.PC4 = $1X11_B$	Output
	P9.5 / EN02	P9_IOCR4.PC5 = $1X11_B$	Output
	P9.6 / EN01	P9_IOCR4.PC6 = $1X11_B$	Output
	P9.7 / SOP0B	P9_IOCR4.PC7 = $1X11_B$	Output
	P9.8 / FCLP0B	P9_IOCR8.PC8 = $1X01_B$	Output
		P9_IOCR8.PC8 = $1X10_B$	
		P9_IOCR8.PC8 = $1X11_B$	

Table 18-10 MSC0 and MSC1 I/O Line Selection and Setup (cont'd)

Module	Port Lines	Input/Output Control Register Bits	I/O
MSC1[3]	P5.12 / SON1	P5_IOCR12.PC12 = $1X01_B$ P5_PDR.PD3 = $0XX_B$	CMOS Output[1]
		P5_IOCR12.PC12 = $1XXX_B$ P5_PDR.PD3 = $1XX_B$	LVDS Output
	P5.13 / SOP1A	P5_IOCR12.PC13 = $1X01_B$ P5_PDR.PD3 = $0XX_B$	CMOS Output[1]
		P5_IOCR12.PC13 = $1XXX_B$ P5_PDR.PD3 = $1XX_B$	LVDS Output
	P5.14 / FCLN1	P5_IOCR12.PC14 = $1X01_B$ P5_PDR.PD3 = $0XX_B$	CMOS Output[1]
		P5_IOCR12.PC14 = $1XXX1_B$ P5_PDR.PD3 = $1XX_B$	LVDS Output
	P5.15 / FCLP1A	P5_IOCR12.PC15 = $1X01_B$ P5_PDR.PD3 = $0XX_B$	CMOS Output[1]
		P5_IOCR12.PC15 = $1XXX_B$ P5_PDR.PD3 = $1XX_B$	LVDS Output
	P5.6 / EN10	P5_IOCR4.PC6 = $1X01_B$	Output
	P5.7 / SDI1[2]	P5_IOCR4.PC7 = $0XXX_B$	Input
	P9.0 / EN12	P9_IOCR0.PC0 = $1X11_B$	Output
	P9.1 / EN11	P9_IOCR0.PC1 = $1X11_B$	Output
	P9.2 / SOP1B	P9_IOCR0.PC2 = $1X11_B$	Output
	P9.3 / FCLP1B	P9_IOCR0.PC3 = $1X11_B$	Output

1) Default after reset.

2) For the upstream channel serial data inputs, additionally bit fields MSC0_OCR.SDISEL (for SDI0) and MSC1_OCR.SDISEL (for MSC1) must be set to 000_B.

3) Chip enable output 3 (EN3) of the MSC1 module is not connected in the TC1797.

18.3.5 On-Chip Connections

This section describes the on-chip connections of the MSC0/MSC1 modules.

18.3.5.1 EMGSTOPMSC Signal (from SCU)

The emergency stop input signals EMGSTOPMSC of both MSC modules are connected to the output signal of the emergency stop input control logic. This logic is located in the SCU. Its functionality is controlled by the SCU emergency stop register.

18.3.5.2 ALTINH and ALTINL Connections

In the TC1797, output lines of GPTA0, GPTA1, and LTCA2 are connected to ALTINH and ALTINL inputs of the MSC0/MSC1 downstream channels as shown in **Figure 18-32**. The setting of the GPTA-to-MSC multiplexer determines which output of the three GPTA sub-modules is connected to a specific ALTINL/ALTINH line.

Please note that the upper eight MSC1 input lines ALTINH[15:8] are connected to logic 0 level.

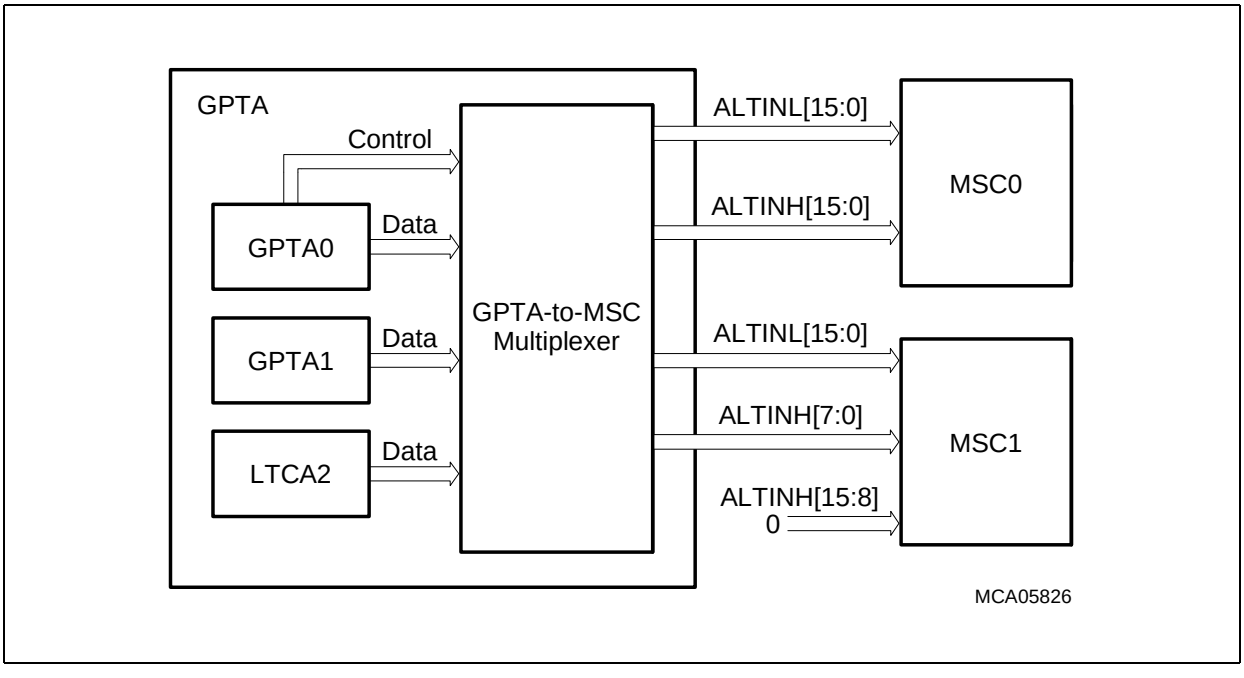


Figure 18-32 ALTINL/ALTINH Connections of the MSC0/MSC1

18.3.5.3 DMA Controller Service Requests

Two service request outputs (SR[3:2]) of each MSC module are connected as DMA request input to the DMA controller. The DMA request lines are connected to the DMA controller as shown in **Table 18-11**.

Table 18-11 Service Request Lines of MSC0/MSC1

Module	Service Request Line	Connected To	Description
MSC0	SR0	MSC0_SRC0	MSC0 Service Request Node 0
	SR1	MSC0_SRC1	MSC0 Service Request Node 1
	SR2	CH02_REQI6	DMA Channel 02 Request Input 6
		CH04_REQI6	DMA Channel 04 Request Input 6
	SR3	CH03_REQI6	DMA Channel 03 Request Input 6
		CH05_REQI6	DMA Channel 05 Request Input 6
MSC1	SR0	MSC1_SRC0	MSC1 Service Request Node 0
	SR1	MSC1_SRC1	MSC1 Service Request Node 1
	SR2	CH12_REQI14	DMA Channel 12 Request Input 14
		CH14_REQI14	DMA Channel 14 Request Input 14
	SR3	CH13_REQI14	DMA Channel 13 Request Input 14
		CH15_REQI14	DMA Channel 15 Request Input 14

18.3.6 Interrupt Control Registers

In the TC1797, the two service request outputs SR[1:0] of each MSC module are connected to one interrupt node. The upper two service request outputs SR[3:2] of each MSC module are not connected to interrupt nodes, but can be used as DMA requests (see **Table 18-11**).

SRC1
Service Request Control Register 1 ($F8_H$) Reset Value: 0000 0000$_H$
SRC0
Service Request Control Register 0 (FC_H) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SETR	CLRR	SRR	SRE	0	TOS	0		SRPN							
w	w	rh	rw	r	rw	r		rw							

Field	Bits	Type	Description
SRPN	[7:0]	rw	**Service Request Priority Number**
TOS	10	rw	**Type of Service Control**
SRE	12	rw	**Service Request Enable**
SRR	13	rh	**Service Request Flag**
CLRR	14	w	**Request Clear Bit**
SETR	15	w	**Request Set Bit**
0	[9:8], 11, [31:16]	r	**Reserved** Read as 0; should be written with 0.

18.3.7 MSC0/MSC1 Address Map

An absolute register address is given by the offset address of the register (given in **Table 18-9**) plus the module base address (given in **Table 18-8**).

Table 18-12 Address Map of MSC0/MSC1

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
MicroSecond Bus Controller 0 (MSC0)					
MSC0_ CLC	MSC0 Clock Control Register	F000 0800$_H$	U, SV	SV, E	0000 0003$_H$
–	Reserved	F000 0804$_H$	BE	BE	–
MSC0_ ID	MSC0 Module Identification Register	F000 0808$_H$	U, SV	BE	0028 C0XX$_H$
MSC0_ FDR	MSC0 Fractional Divider Register	F000 080C$_H$	U, SV	SV, E	0000 0000$_H$
MSC0_ USR	MSC0 Upstream Status Register	F000 0810$_H$	U, SV	U, SV	0000 0000$_H$
MSC0_ DSC	MSC0 Downstream Control Register	F000 0814$_H$	U, SV	U, SV	0000 0000$_H$
MSC0_ DSS	MSC0 Downstream Status Register	F000 0818$_H$	U, SV	U, SV	0000 0000$_H$
MSC0_ DD	MSC0 Downstream Data Register	F000 081C$_H$	U, SV	U, SV	0000 0000$_H$
MSC0_ DC	MSC0 Downstream Command Register	F000 0820$_H$	U, SV	U, SV	0000 0000$_H$
MSC0_ DSDSL	MSC0 Downstream Select Data Source Low Register	F000 0824$_H$	U, SV	U, SV	0000 0000$_H$
MSC0_ DSDSH	MSC0 Downstream Select Data Source High Register	F000 0828$_H$	U, SV	U, SV	0000 0000$_H$
MSC0_ ESR	MSC0 Emergency Stop Register	F000 082C$_H$	U, SV	U, SV	0000 0000$_H$
MSC0_ UD0	MSC0 Upstream Data Register 0	F000 0830$_H$	U, SV	U, SV	0000 0000$_H$
MSC0_ UD1	MSC0 Upstream Data Register 1	F000 0834$_H$	U, SV	U, SV	0000 0000$_H$

Table 18-12 Address Map of MSC0/MSC1 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
MSC0_ UD2	MSC0 Upstream Data Register 2	F000 0838$_H$	U, SV	U, SV	0000 0000$_H$
MSC0_ UD3	MSC0 Upstream Data Register 3	F000 083C$_H$	U, SV	U, SV	0000 0000$_H$
MSC0_ ICR	MSC0 Interrupt Control Register	F000 0840$_H$	U, SV	U, SV	0000 0000$_H$
MSC0_ ISR	MSC0 Interrupt Status Register	F000 0844$_H$	U, SV	U, SV	0000 0000$_H$
MSC0_ ISC	MSC0 Interrupt Set Clear Register	F000 0848$_H$	U, SV	U, SV	0000 0000$_H$
MSC0_ OCR	MSC0 Output Control Register	F000 084C$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 0850$_H$ - F000 0854$_H$	nBE	nBE	–
		F000 0858$_H$ - F000 08F4$_H$	BE	BE	–
MSC0_ SRC1	MSC0 Service Request Control Register 1	F000 08F8$_H$	U, SV	U, SV	0000 0000$_H$
MSC0_ SRC0	MSC0 Service Request Control Register 0	F000 08FC$_H$	U, SV	U, SV	0000 0000$_H$
Micro Second Channel 1 (MSC1)					
MSC1_ CLC	MSC1 Clock Control Register	F000 0900$_H$	U, SV	SV, E	0000 0003$_H$
–	Reserved	F000 0904$_H$	BE	BE	–
MSC1_ ID	MSC1 Module Identification Register	F000 0908$_H$	U, SV	BE	0000 44XX$_H$
MSC1_ FDR	MSC1 Fractional Divider Register	F000 090C$_H$	U, SV	SV, E	0000 0000$_H$
MSC1_ USR	MSC1 Upstream Status Register	F000 0910$_H$	U, SV	U, SV	0000 0000$_H$
MSC1_ DSC	MSC1 Downstream Control Register	F000 0914$_H$	U, SV	U, SV	0000 0000$_H$

Table 18-12 Address Map of MSC0/MSC1 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
MSC1_ DSS	MSC1 Downstream Status Register	F000 0918$_H$	U, SV	U, SV	0000 0000$_H$
MSC1_ DD	MSC1 Downstream Data Register	F000 091C$_H$	U, SV	U, SV	0000 0000$_H$
MSC1_ DC	MSC1 Downstream Command Register	F000 0920$_H$	U, SV	U, SV	0000 0000$_H$
MSC1_ DSDSL	MSC1 Downstream Select Data Source Low Register	F000 0924$_H$	U, SV	U, SV	0000 0000$_H$
MSC1_ DSDSH	MSC1 Downstream Select Data Source High Register	F000 0928$_H$	U, SV	U, SV	0000 0000$_H$
MSC1_ ESR	MSC1 Emergency Stop Register	F000 092C$_H$	U, SV	U, SV	0000 0000$_H$
MSC1_ UD0	MSC1 Upstream Data Register 0	F000 0930$_H$	U, SV	U, SV	0000 0000$_H$
MSC1_ UD1	MSC1 Upstream Data Register 1	F000 0934$_H$	U, SV	U, SV	0000 0000$_H$
MSC1_ UD2	MSC1 Upstream Data Register 2	F000 0938$_H$	U, SV	U, SV	0000 0000$_H$
MSC1_ UD3	MSC1 Upstream Data Register 3	F000 093C$_H$	U, SV	U, SV	0000 0000$_H$
MSC1_ ICR	MSC1 Interrupt Control Register	F000 0940$_H$	U, SV	U, SV	0000 0000$_H$
MSC1_ ISR	MSC1 Interrupt Status Register	F000 0944$_H$	U, SV	U, SV	0000 0000$_H$
MSC1_ ISC	MSC1 Interrupt Set Clear Register	F000 0948$_H$	U, SV	U, SV	0000 0000$_H$
MSC1_ OCR	MSC1 Output Control Register	F000 094C$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 0950$_H$ - F000 0954$_H$	nBE	nBE	–
		F000 0958$_H$ - F000 09F4$_H$	BE	BE	–

Table 18-12 Address Map of MSC0/MSC1 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
MSC1_ SRC1	MSC1 Service Request Control Register 1	F000 09F8$_H$	U, SV	U, SV	0000 0000$_H$
MSC1_ SRC0	MSC1 Service Request Control Register 0	F000 09FC$_H$	U, SV	U, SV	0000 0000$_H$

19 Controller Area Network Controller (MultiCAN)

This chapter describes the MultiCAN controller of the TC1797. It contains the following sections:

- CAN basics (see **Page 19-2**)
- Overview of the CAN Module in the TC1797 (see **Page 19-11**)
- Functional description of the MultiCAN Kernel (see **Page 19-14**)
- MultiCAN Kernel register description (see **Page 19-55**)
- TC1797 implementation-specific details (port connections and control, interrupt control, address decoding, clock control, see **Page 19-109**).

Note: The MultiCAN register names described in this chapter are referenced in the TC1797 User´s Manual by the module name prefix “CAN_”.

Table 19-1 Fixed Module Constants

Constant	Description
n_objects	**Number of Message Objects available.**
n_interrupts	**Number of Interrupt Output Lines available.**
n_pendings **n_pendingregs**	**Number of Message Pending Bits available.** There are n_pendings/32 message pending registers.
n_lists	**Number of Lists available for allocation of Message Objects.**
n_nodes	**Number of CAN Nodes available** As each CAN node has it’s own list in addition to the list of un-allocated elements, the relation n_nodes < n_lists is true.

19.1 CAN Basics

CAN is an asynchronous serial bus system with one logical bus line. It has an open, linear bus structure with equal bus participants called nodes. A CAN bus consists of two or more nodes.

The bus logic corresponds to a “wired-AND” mechanism. Recessive bits (equivalent to the logic 1 level) are overwritten by dominant bits (logic 0 level). As long as no bus node is sending a dominant bit, the bus is in the recessive state. In this state, a dominant bit from any bus node generates a dominant bus state. The maximum CAN bus speed is, by definition, 1 Mbit/s. This speed limits the CAN bus to a length of up to 40 m. For bus lengths longer than 40 m, the bus speed must be reduced.

The binary data of a CAN frame is coded in NRZ code (Non-Return-to-Zero). To ensure re-synchronization of all bus nodes, bit stuffing is used. This means that during the transmission of a message, a maximum of five consecutive bits can have the same polarity. Whenever five consecutive bits of the same polarity have been transmitted, the transmitter will insert one additional bit (stuff bit) of the opposite polarity into the bit stream before transmitting further bits. The receiver also checks the number of bits with the same polarity and removes the stuff bits from the bit stream (= destuffing).

19.1.1 Addressing and Bus Arbitration

In the CAN protocol, address information is defined in the identifier field of a message. The identifier indicates the contents of the message and its priority. The lower the binary value of the identifier, the higher is the priority of the message.

For bus arbitration, CSMA/CD with NDA (Carrier Sense Multiple Access/Collision Detection with Non-Destructive Arbitration) is used. If bus node A attempts to transmit a message across the network, it first checks that the bus is in the idle state (“Carrier Sense”) i.e. no node is currently transmitting. If this is the case (and no other node wishes to start a transmission at the same moment), node A becomes the bus master and sends its message. All other nodes switch to receive mode during the first transmitted bit (Start-Of-Frame bit). After correct reception of the message (acknowledged by each node), each bus node checks the message identifier and stores the message, if required. Otherwise, the message is discarded.

If two or more bus nodes start their transmission at the same time (“Multiple Access”), bus collision of the messages is avoided by bit-wise arbitration (“Collision Detection / Non-Destructive Arbitration” together with the “Wired-AND” mechanism, dominant bits override recessive bits). Each node that sends also reads back the bus level. When a recessive bit is sent but a dominant one is read back, bus arbitration is lost and the transmitting node switches to receive mode. This condition occurs for example when the message identifier of a competing node has a lower binary value and therefore sends a message with a higher priority. In this way, the bus node with the highest priority message wins arbitration without losing time by having to repeat the message. Other nodes that lost arbitration will automatically try to repeat their transmission once the bus

returns to idle state. Therefore, the same identifier can be sent in a Data Frame only by one node in the system. There must not be more than one node programmed to send Data Frames with the same identifier.

Standard message identifier has a length of 11 bits. CAN specification 2.0B extends the message identifier lengths to 29 bits, i.e. the extended identifier.

19.1.2 CAN Frame Formats

There are three types of CAN frames:

- Data Frames
- Remote Frames
- Error Frames

A Data Frame contains a Data Field of 0 to 8 bytes in length. A Remote Frame contains no Data Field and is typically generated as a request for data (e.g. from a sensor). Data and Remote Frames can use an 11-bit “Standard” identifier or a 29-bit “Extended” identifier. An Error Frame can be generated by any node that detects a CAN bus error.

19.1.2.1 Data Frames

There are two types of Data Frames defined (see **Figure 19-1**):

- Standard Data Frame
- Extended Data Frame

Standard Data Frame

A Data Frame begins with the Start-Of-Frame bit (SOF = dominant level) for hard synchronization of all nodes. The SOF is followed by the Arbitration Field consisting of 12 bits, the 11-bit identifier (reflecting the contents and priority of the message), and the RTR (Remote Transmission Request) bit. With RTR at dominant level, the frame is marked as Data Frame. With RTR at recessive level, the frame is defined as a Remote Frame.

The next field is the Control Field consisting of 6 bits. The first bit of this field is the IDE (Identifier Extension) bit and is at dominant level for the Standard Data Frame. The following bit is reserved and defined as a dominant bit. The remaining 4 bits of the Control Field are the Data Length Code (DLC) that specifies the number of bytes in the Data Field. The Data Field can be 0 to 8 bytes wide. The Cyclic Redundancy (CRC) Field that follows the data bytes is used to detect possible transmission errors. It consists of a 15-bit CRC sequence, completed by a recessive CRC delimiter bit.

The final field is the Acknowledge Field. During the ACK Slot, the transmitting node sends out a recessive bit. Any node that has received an error free frame acknowledges the correct reception of the frame by sending back a dominant bit, regardless of whether or not the node is configured to accept that specific message. This behavior assigns the

CAN protocol to the “in-bit-response” group of protocols. The recessive ACK delimiter bit, which must not be overwritten by a dominant bit, completes the Acknowledge Field.

Seven recessive End-of-Frame (EOF) bits finish the Data Frame. Between any two consecutive frames, the bus must remain in the recessive state for at least 3 bit times (called Inter Frame Space). If after the Inter Frame Space, no other nodes attempt to transmit the bus remains in idle state with a recessive level.

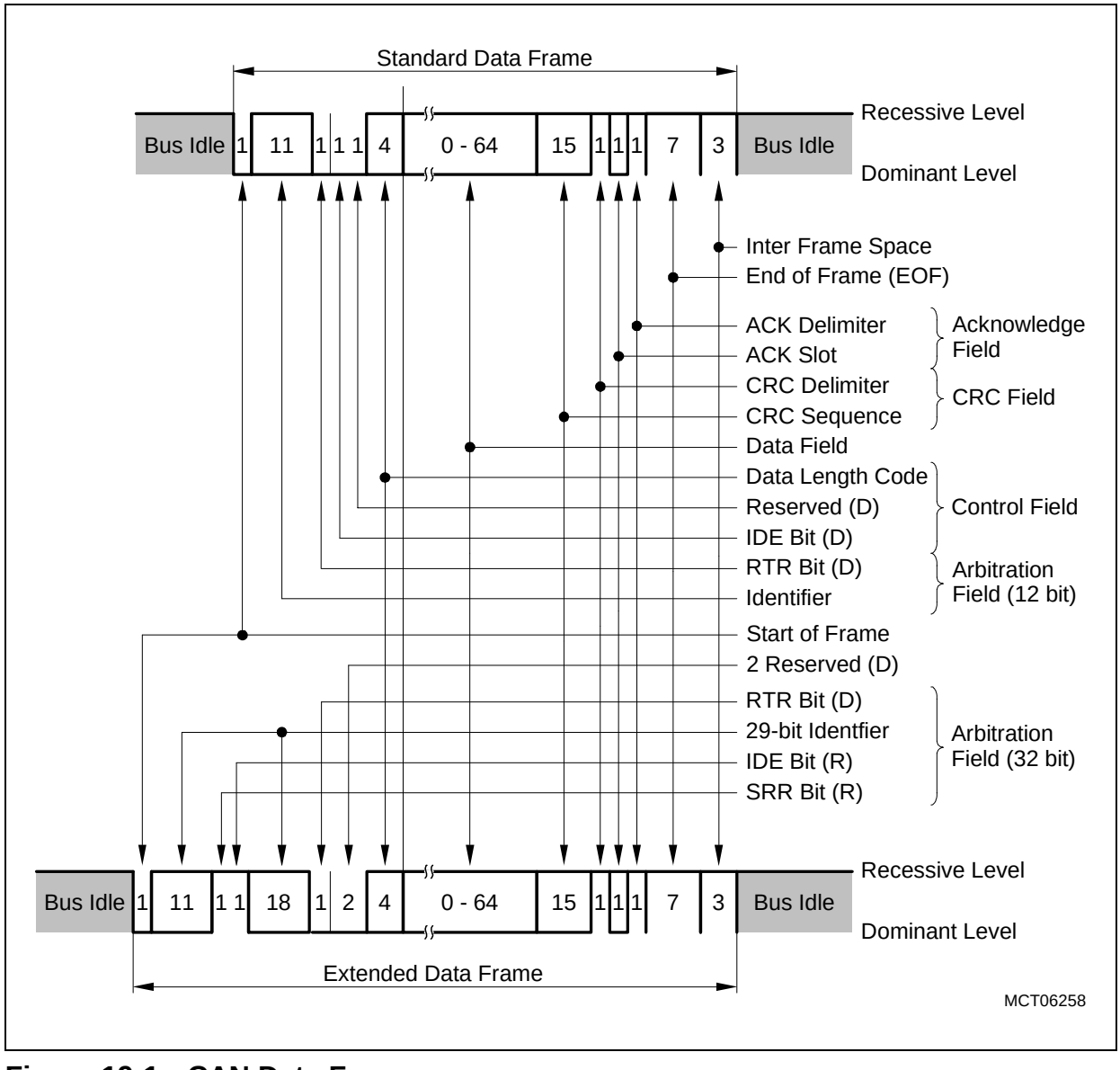


Figure 19-1 CAN Data Frame

Extended Data Frame

In the Extended CAN Data Frame, the message identifier of the standard frame has been extended to 29-bit. A split of the extended identifier into two parts, an 11-bit least

significant section (as in standard CAN frame) and an 18-bit most significant section, ensures that the Identifier Extension bit (IDE) can remain at the same bit position in both standard and extended frames.

In the Extended CAN Data Frame, the SOF bit is followed by the 32-bit Arbitration Field. The first 11 bits are the least significant bits of the 29-bit Identifier (“Base-ID”). These 11 bits are followed by the recessive Substitute Remote Request (SRR) bit. The SRR is further followed by the recessive IDE bit, which indicates the frame to be an Extended CAN frame. If arbitration remains unresolved after transmission of the first 11 bits of the identifier, and if one of the nodes involved in arbitration is sending a Standard CAN frame, then the Standard CAN frame will win arbitration due to the assertion of its dominant IDE bit. Therefore, the SRR bit in an Extended CAN frame is recessive to allow the assertion of a dominant RTR bit by a node that is sending a Standard CAN Remote Frame. The SRR and IDE bits are followed by the remaining 18 bits of the extended identifier and the RTR bit.

Control field and frame termination is identical to the Standard Data Frame.

19.1.2.2 Remote Frames

Normally, data transmission is performed on an autonomous basis with the data source node (e.g. a sensor) sending out a Data Frame. It is also possible, however, for a destination node (or nodes) to request the data from the source. For this purpose, the destination node sends a Remote Frame with an identifier that matches the identifier of the required Data Frame. The appropriate data source node will then send a Data Frame as a response to this remote request.

There are 2 differences between a Remote Frame and a Data Frame.

- The RTR bit is in the recessive state in a Remote Frame.
- There is no Data Field in a Remote Frame.

If a Data Frame and a Remote Frame with the same identifier are transmitted at the same time, the Data Frame wins arbitration due to the dominant RTR bit following the identifier. In this way, the node that transmitted the Remote Frame receives the requested data immediately. The format of a Standard and Extended Remote Frames is shown in **Figure 19-2**.

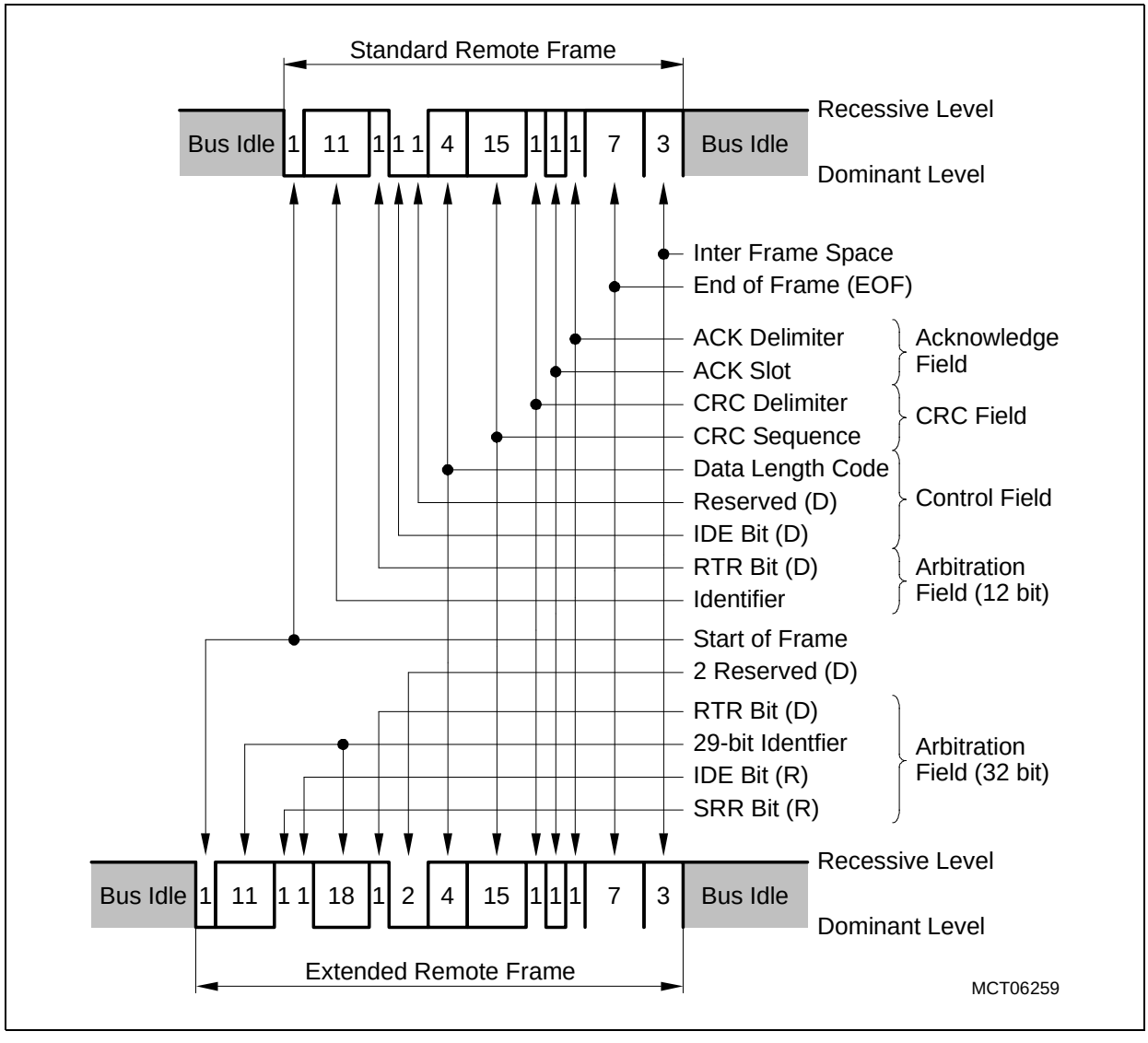


Figure 19-2 CAN Remote Frame

19.1.2.3 Error Frames

An Error Frame is generated by any node that detects a bus error. An Error Frame consists of two fields, an Error Flag field followed by an Error Delimiter field. The Error Delimiter Field consists of 8 recessive bits and allows the bus nodes to restart bus communications after an error. There are, however, two forms of Error Flag fields. The form of the Error Flag field depends on the error status of the node that detects the error.

When an error-active node detects a bus error, the node generates an Error Frame with an active-error flag. The error-active flag is composed of six consecutive dominant bits that actively violate the bit-stuffing rule. All other stations recognize a bit-stuffing error and generate Error Frames themselves. The resulting Error Flag field on the CAN bus therefore consists of six to twelve consecutive dominant bits (generated by one or more nodes). The Error Delimiter field completes the Error Frame. After completion of the Error Frame, bus activity returns to normal and the interrupted node attempts to re-send the aborted message.

If an error-passive node detects a bus error, the node transmits an error-passive flag followed, again, by the Error Delimiter field. The error-passive flag consists of six consecutive recessive bits, and therefore the Error Frame (for an error-passive node) consists of 14 recessive bits (i.e. no dominant bits). Therefore, the transmission of an Error Frame by an error-passive node will not affect any other node on the network, unless the bus error is detected by the node that is actually transmitting (i.e. the bus master). If the bus master node generates an error-passive flag, this may cause other nodes to generate Error Frames due to the resulting bit-stuffing violation. After transmission of an Error Frame an error-passive node must wait for 6 consecutive recessive bits on the bus before attempting to rejoin bus communications.

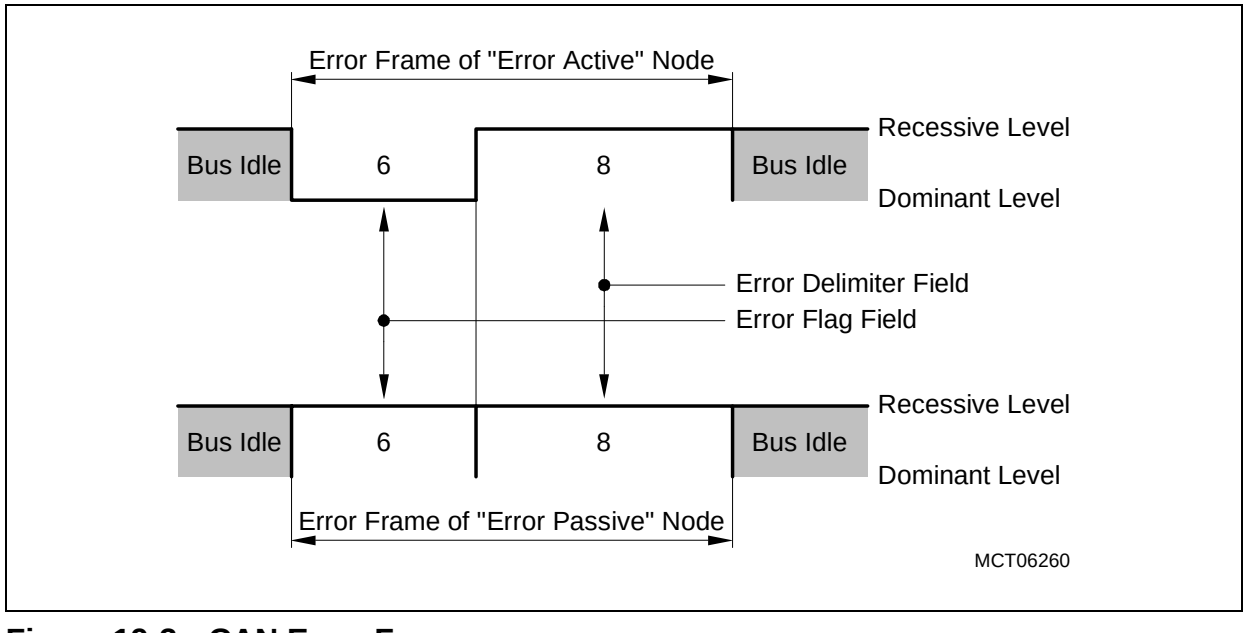


Figure 19-3 CAN Error Frames

19.1.3 The Nominal Bit Time

One bit cell (this means one high or low pulse of the NRZ code) is composed by four segments. Each segment is an integer multiple of Time Quanta t_Q. The Time Quanta is the smallest discrete timing resolution used by a CAN node. The nominal bit time definition with its segments is shown in **Figure 19-4**.

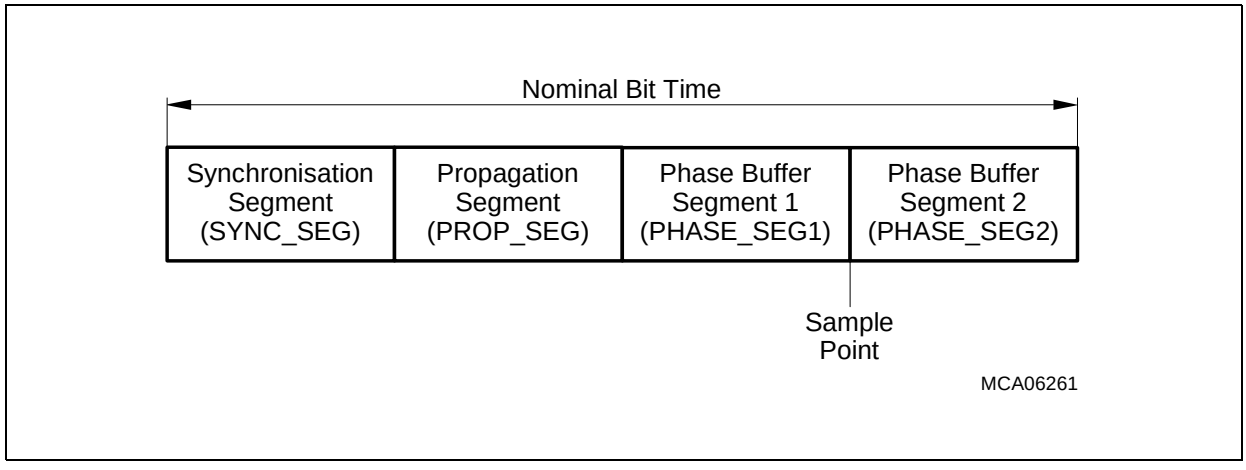


Figure 19-4 Partition of Nominal Bit Time

The Synchronization Segment (SYNC_SEG) is used to synchronize the various bus nodes. If there is a bit state change between the previous bit and the current bit, then the bus state change is expected to occur within this segment. The length of this segment is always 1 t_Q.

The Propagation Segment (PROP_SEG) is used to compensate for signal delays across the network. These delays are caused by signal propagation delay on the bus line and through the electronic interface circuits of the bus nodes.

The Phase Segments 1 and 2 (PHASE_SEG1, PHASE_SEG2) are used to compensate for edge phase errors. These segments can be lengthened or shortened by re-synchronization. PHASE_SEG2 is reserved for calculation of the subsequent bit level, and is ≥ 2 t_Q. At the sample point, the bus level is read and interpreted as the value of the bit cell. It occurs at the end of PHASE_SEG1.

The total number of t_Q in a bit time is between 8 and 25.

As a result of re-synchronization, PHASE_SEG1 can be lengthened or PHASE_SEG2 can be shortened. The amount of lengthening or shortening the phase buffer segments has an upper limit given by the re-synchronization jump width. The re-synchronization jump width may be between 1 and 4 t_Q, but it may not be longer than PHASE_SEG1.

19.1.4 Error Detection and Error Handling

The CAN protocol has sophisticated error detection mechanisms. The following errors can be detected:

- **Cyclic Redundancy Check (CRC) Error**
  With the Cyclic Redundancy Check, the transmitter calculates special check bits for the bit sequence from the start of a frame until the end of the Data Field. This CRC sequence is transmitted in the CRC Field. The receiving node also calculates the CRC sequence using the same formula, and performs a comparison to the received sequence. If a mismatch is detected, a CRC error has occurred and an Error Frame is generated. The message is repeated.
- **Acknowledge Error**
  In the Acknowledge Field of a message, the transmitter checks whether a dominant bit is read during the Acknowledge Slot (that is sent out as a recessive bit). If not, no other node has received the frame correctly, an Acknowledge Error has occurred, and the message must be repeated. No Error Frame is generated.
- **Form Error**
  If a transmitter detects a dominant bit in one of the four segments End of Frame, Interframe Space, Acknowledge Delimiter, or CRC Delimiter, a Form Error has occurred, and an Error Frame is generated. The message is repeated.
- **Bit Error**
  A Bit Error occurs if a) a transmitter sends a dominant bit and detects a recessive bit or b) if the transmitter sends a recessive bit and detects a dominant bit when monitoring the actual bus level and comparing it to the just transmitted bit. In case b), no error occurs during the Arbitration Field (ID, RTR, IDE) and the Acknowledge Slot.
- **Stuff Error**
  If between Start of Frame and CRC Delimiter, six consecutive bits with the same polarity are detected, the bit-stuffing rule has been violated. A stuff error occurs and an Error Frame is generated. The message is repeated.

Detected errors are made public to all other nodes via Error Frames (except Acknowledge Errors). The transmission of the erroneous message is aborted and the frame is repeated as soon as possible. Furthermore, each CAN node is in one of the three error states (error-active, error-passive or bus-off) according to the value of the internal error counters. The error-active state is the usual state where the bus node can transmit messages and active-error frames (made of dominant bits) without any restrictions. In the error-passive state, messages and passive-error frames (made of recessive bits) may be transmitted. The bus-off state makes it temporarily impossible for the node to participate in the bus communication. During this state, messages can be neither received nor transmitted.

Basic CAN, Full CAN

There is one more CAN characteristic that is related to the interface of a CAN module (controller) and the host CPU: Basic-CAN and Full-CAN functionality.

In Basic-CAN devices, only basic functions of the protocol are implemented in hardware, such as the generation and the check of the bit stream. The decision, whether a received message has to be stored or not (acceptance filtering), and the complete message management must be done by software. Normally, the CAN device also provides only one transmit buffer and one or two receive buffers. Therefore, the host CPU load is quite high when using Basic-CAN modules. The main advantage of Basic-CAN is a reduced chip size leading to low costs of these devices.

Full-CAN devices (this is the case for the MultiCAN controller as implemented in TC1797) manage the whole bus protocol in hardware, including the acceptance filtering and message management. Full-CAN devices contain message objects that handle autonomously the identifier, the data, the direction (receive or transmit) and the information of Standard CAN/Extended CAN operation. During the initialization of the device, the host CPU determines which messages are to be sent and which are to be received. The host CPU is informed by interrupt if the identifier of a received message matches with one of the programmed (receive-) message objects. The CPU load of Full-CAN devices is greatly reduced. When using Full-CAN devices, high baud rates and high bus loads with many messages can be handled.

19.2 Overview

This section describes the serial communication module called MultiCAN (CAN = Controller Area Network) of the TC1797. A MultiCAN module can contain between two and eight independent CAN nodes, depending on the device, each representing one serial communication interface.

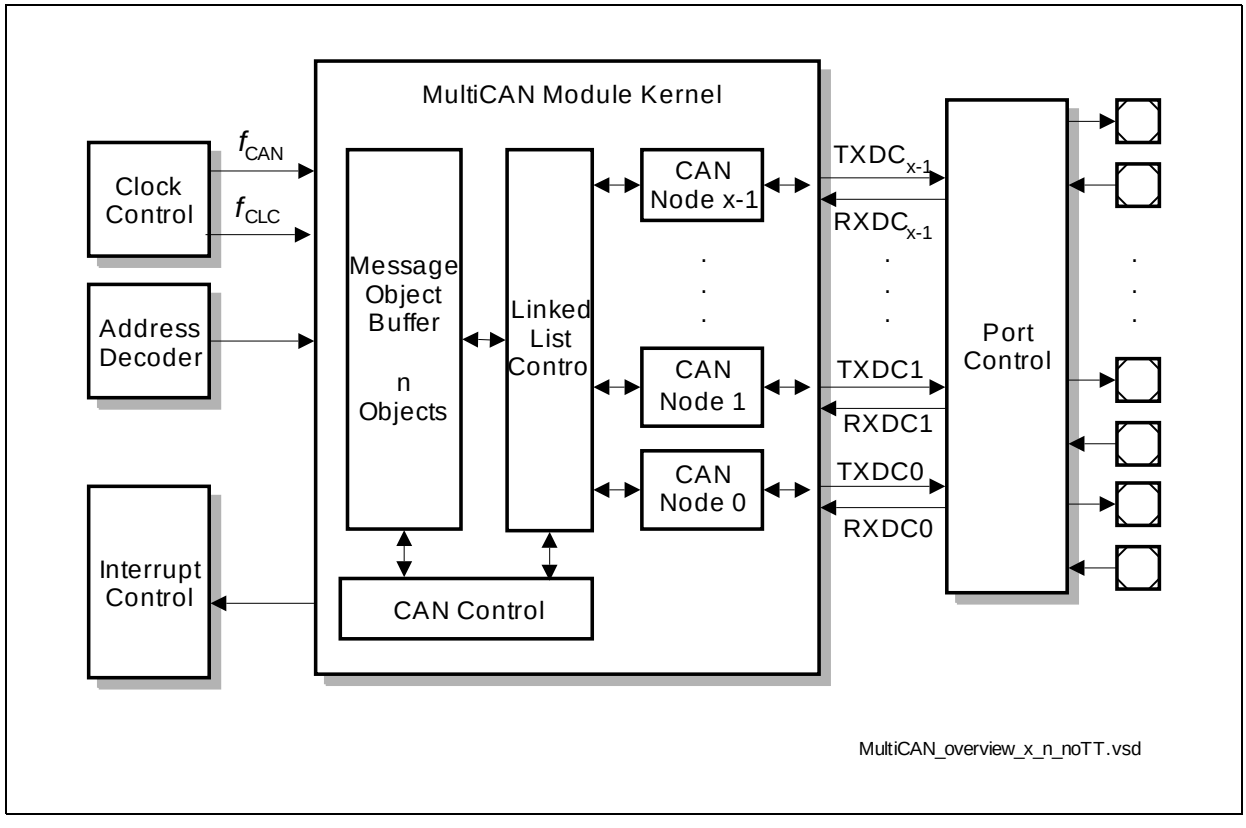


Figure 19-5 Overview of the MultiCAN Module

19.2.1 MultiCAN Module

The MultiCAN module contains independently operating CAN nodes with Full-CAN functionality that are able to exchange Data and Remote Frames via a gateway function. Transmission and reception of CAN frames is handled in accordance with CAN specification V2.0 B (active). Each CAN node can receive and transmit standard frames with 11-bit identifiers as well as extended frames with 29-bit identifiers.

Both CAN nodes share a common set of message objects. Each message object can be individually allocated to one of the CAN nodes. Besides serving as a storage container for incoming and outgoing frames, message objects can be combined to build gateways between the CAN nodes or to setup a FIFO buffer.

The message objects are organized in double-chained linked lists, where each CAN node has its own list of message objects. A CAN node stores frames only into message objects that are allocated to the message object list of the CAN node, and it transmits only messages belonging to this message object list. A powerful, command-driven list controller performs all message object list operations.

The bit timings for the CAN nodes are derived from the module timer clock (f_{CAN}), and are programmable up to a data rate of 1 Mbit/s. External bus transceivers are connected to a CAN node via a pair of receive and transmit pins.

Features

- Compliant with ISO 11898
- CAN functionality according to CAN specification V2.0 B active
- Dedicated control registers for each CAN node
- Data transfer rates up to 1 Mbit/s
- Flexible and powerful message transfer control and error handling capabilities
- Advanced CAN bus bit timing analysis and baud rate detection for each CAN node via a frame counter
- Full-CAN functionality: A set of message objects can be individually
  - Allocated (assigned) to any CAN node
  - Configured as transmit or receive object
  - Set up to handle frames with 11-bit or 29-bit identifier
  - Identified by a timestamp via a frame counter
  - Configured to remote monitoring mode
- Advanced acceptance filtering
  - Each message object provides an individual acceptance mask to filter incoming frames
  - A message object can be configured to accept standard or extended frames or to accept both standard and extended frames
  - Message objects can be grouped into four priority classes for transmission and reception

  - The selection of the message to be transmitted first can be based on frame identifier, IDE bit and RTR bit according to CAN arbitration rules, or according to its order in the list
- Advanced message object functionality
  - Message objects can be combined to build FIFO message buffers of arbitrary size, limited only by the total number of message objects
  - Message objects can be linked to form a gateway that automatically transfers frames between two different CAN buses. A single gateway can link any two CAN nodes. An arbitrary number of gateways can be defined.
- Advanced data management
  - The message objects are organized in double-chained lists
  - List reorganizations can be performed at any time, even during full operation of the CAN nodes
  - A powerful, command-driven list controller manages the organization of the list structure and ensures consistency of the list
  - Message FIFOs are based on the list structure and can easily be scaled in size during CAN operation
  - Static allocation commands offer compatibility with TwinCAN applications that are not list-based
- Advanced interrupt handling
  - Up to 16 interrupt output lines are available. Interrupt requests can be routed individually to one of the 16 interrupt output lines
  - Message post-processing notifications can be mapped flexibly using dedicated registers consisting of notification bits

19.3 MultiCAN Kernel Functional Description

This section describes the functionality of the MultiCAN module.

19.3.1 Module Structure

Figure 19-6 shows the general structure of the MultiCAN module.

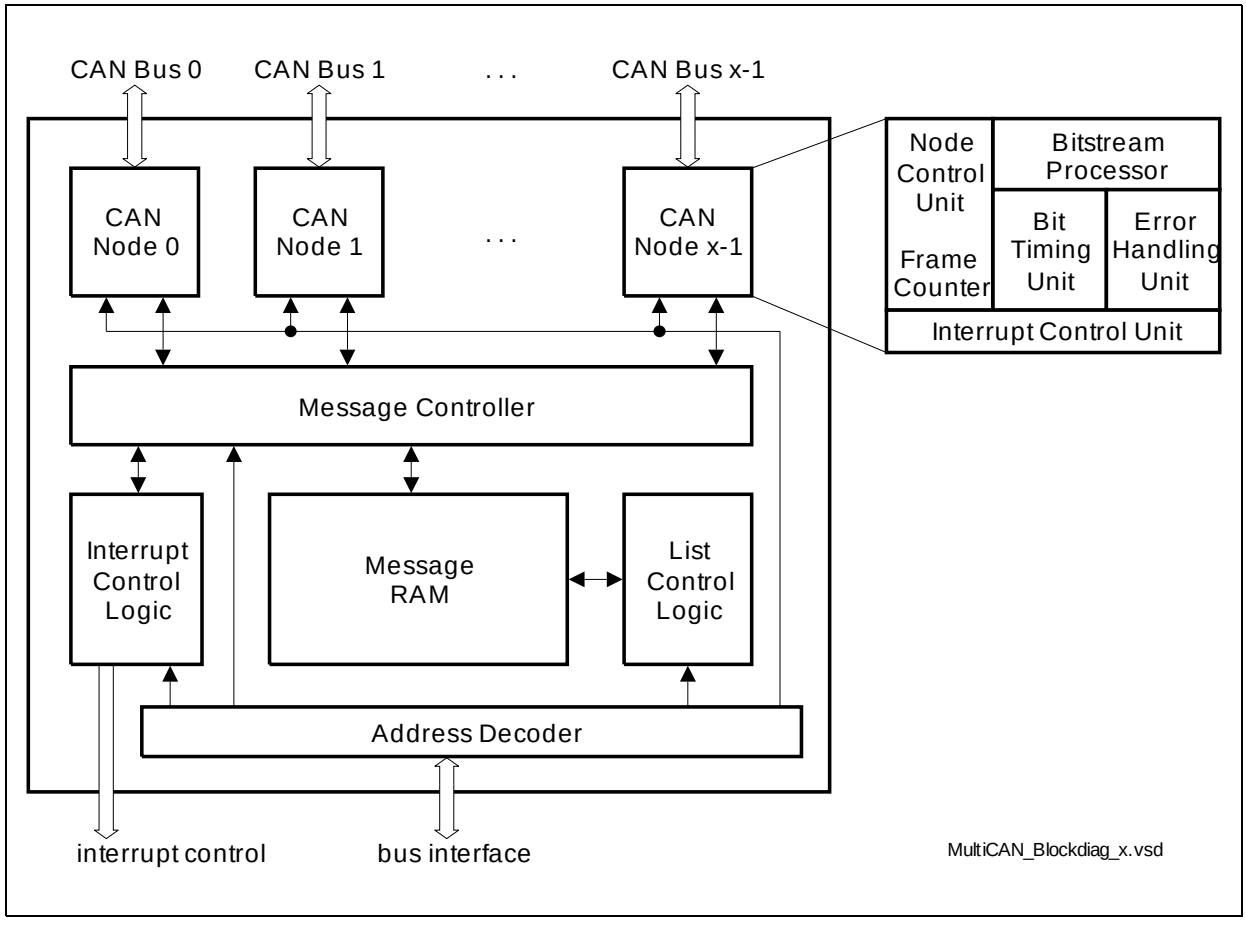


Figure 19-6 MultiCAN Block Diagram

CAN Nodes

Each CAN node consists of several sub-units.

- **Bitstream Processor**
  The Bitstream Processor performs data, remote, error and overload frame processing according to the ISO 11898 standard. This includes conversion between the serial data stream and the input/output registers.
- **Bit Timing Unit**
  The Bit Timing Unit determines the length of a bit time and the location of the sample point according to the user settings, taking into account propagation delays and phase shift errors. The Bit Timing Unit also performs resynchronization.

- **Error Handling Unit**
  The Error Handling Unit manages the receive and transmit error counter. Depending on the contents of both counters, the CAN node is set into an error-active, error passive or bus-off state.
- **Node Control Unit**
  The Node Control Unit coordinates the operation of the CAN node:
  – Enable/disable CAN transfer of the node
  – Enable/disable and generate node-specific events that lead to an interrupt request (CAN bus errors, successful frame transfers etc.)
  – Administration of the Frame Counter
- **Interrupt Control Unit**
  The Interrupt Control Unit in the CAN node controls the interrupt generation for the different conditions that can occur in the CAN node.

Message Controller

The Message Controller handles the exchange of CAN frames between the CAN nodes and the message objects that are stored in the Message RAM. The Message Controller performs several functions:

- Receive acceptance filtering to determine the correct message object for storing of a received CAN frame
- Transmit acceptance filtering to determine the message object to be transmitted first, individually for each CAN node
- Transfer contents between message objects and the CAN nodes, taking into account the status/control bits of the message objects
- Handling of the FIFO buffering and gateway functionality
- Aggregation of message-pending notification bits

List Controller

The List Controller performs all operations that lead to a modification of the double-chained message object lists. Only the list controller is allowed to modify the list structure. The allocation/deallocation or reallocation of a message object can be requested via a user command interface (command panel). The list controller state machine then performs the requested command autonomously.

Interrupt Control

The general interrupt structure is shown in **Figure 19-8**. The interrupt event can trigger the interrupt generation. The interrupt pulse is generated independently of the interrupt flag in the interrupt status register. The interrupt flag can be reset by software by writing a 0 to it.

If enabled by the related interrupt enable bit in the interrupt enable register, an interrupt pulse can be generated at one of the 16 interrupt output lines INT_Om of the MultiCAN

module. If more than one interrupt source is connected to the same interrupt node pointer (in the interrupt node pointer register), the requests are combined to one common line.

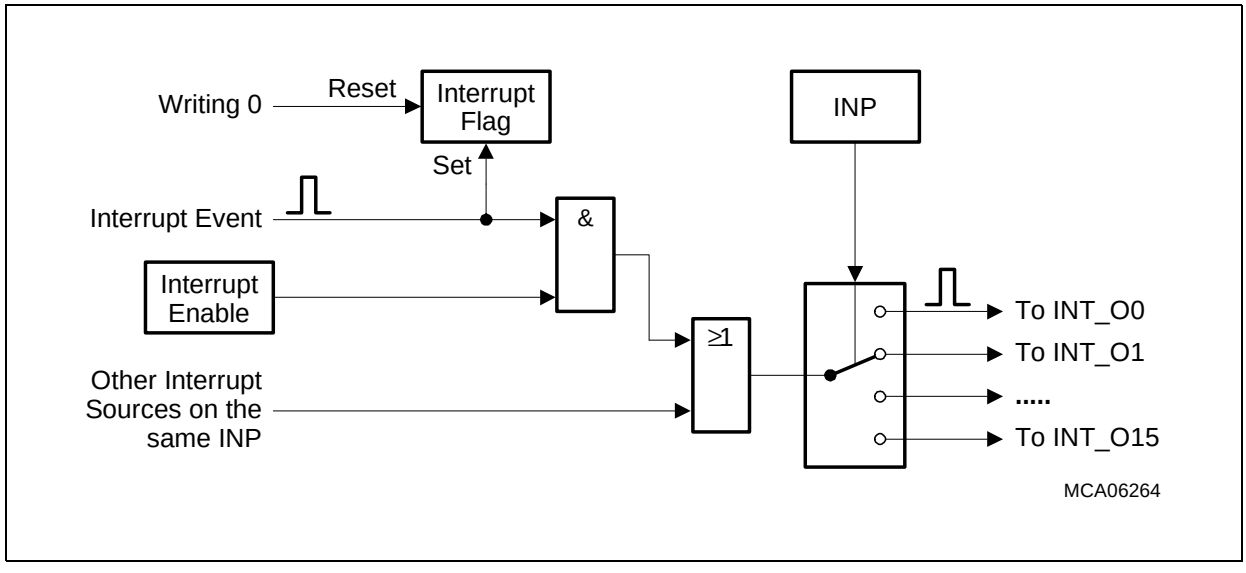


Figure 19-7 General Interrupt Structure

19.3.2 Clock Control

The CAN module timer clock f_{CAN} of the functional blocks of the MultiCAN module is derived from the module control clock f_{CLC}. The Fractional Divider is used to generate f_{CAN} used for bit timing calculation. The frequency of f_{CAN} is identical for all CAN nodes. The register file operate with the module control clock f_{CLC}. See also **"Module Clock Generation" on Page 19-111**.

The output clock f_{CAN} of the Fractional Divider is based on the system clock f_{CLC}, but only every n-th clock pulse is taken. The suspend signal (coming as acknowledge from the MultiCAN module in response to a OCDS suspend request) freezes or resets the Fractional Divider.

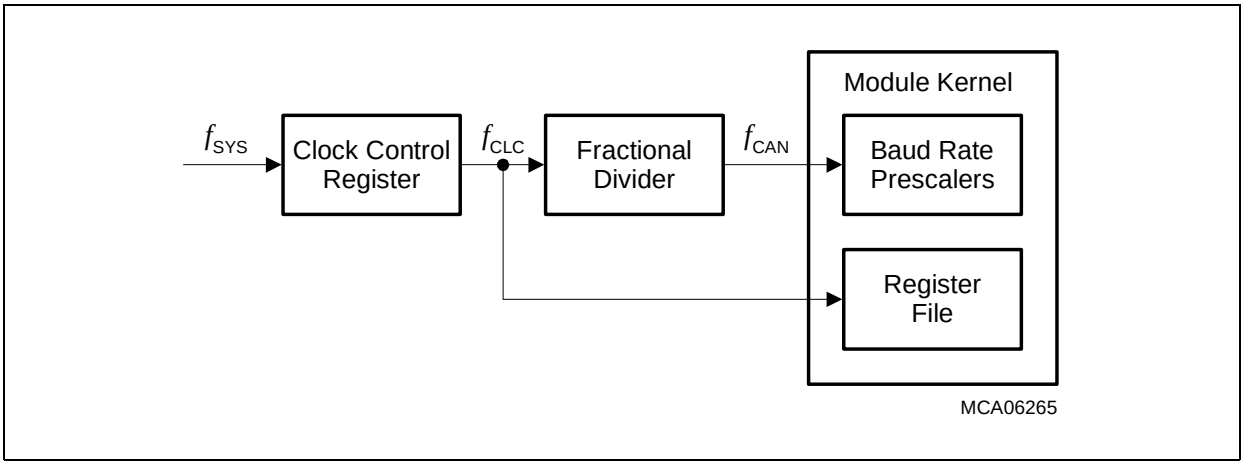


Figure 19-8 MultiCAN Clock Generation

Table 19-2 indicates the minimum operating frequencies in MHz for f_{CLC} that are required for a baud rate of 1 Mbit/s for the active CAN nodes. If a lower baud rate is desired, the values can be scaled linearly (e.g. for a maximum of 500 kbit/s, 50% of the indicated value are required).

The values imply that the CPU (or DMA) executes maximum accesses to the MultiCAN module. The values may contain rounding effects.

Table 19-2 Minimum Operating Frequencies [MHz]

Number of allocated message objects MO[1) 2)]	1 CAN node active	2 CAN nodes active	3 CAN nodes active	4 CAN nodes active	5 CAN nodes active
16 MO	12	19	26	33	40
32 MO	15	23	30	37	44
64 MO	21	28	37	46	53
128 MO	40	45	50	55	61

1) Only those message objects have to be taken into account that are allocated to a CAN node. The unallocated message objects have no influence on the minimum operating frequency.

2) In case of using CAN bootstrap loader, with one active node and two active message objects, the MultiCAN module needs minimum frequency of 10 MHz.

The baud rate generation of the MultiCAN being based on f_{SYS}, this frequency has to be chosen carefully to allow correct CAN bit timing. The required value of f_{SYS} is given by an integer multiple (n) of the CAN baud rate multiplied by the number of time quanta per CAN bit time. For example, to reach 1 Mbit/s with 20 tq per bit time, possible values of f_{SYS} are given by formula [n × 20] MHz, with n being an integer value, starting at 1.
In order to minimize jitter, it is not recommended to use the fractional divider mode for high baud rates.

19.3.3 Port Input Control

It is possible to select the input lines for the RXDCANx inputs for the CAN nodes. The selected input is connected to the CAN node and is also available to wake-up the system. More details are defined in **Section 19.5.4** on **Page 19-115**.

19.3.4 Suspend Mode

The Suspend Mode can be triggered by the OCDS in order to freeze the state of the module and to permit access to the registers (at least for read actions). The MultiCAN module provides two types of Suspend Modes:

- All actions are immediately stopped (Hard Suspend Mode):
  The module clocks f_{CLC} and f_{CAN} are switched off as soon as the suspend request becomes active. Read and write operations to the module are no longer possible. This means that the CAN registers cannot be accessed anymore. In this mode, there is a very high probability that the communication with other CAN devices is made impossible, and that the CAN bus is blocked (e.g. if the suspended CAN module just sends a dominant level). A reset operation must be executed to leave Hard Suspend Mode.

- The current action is finished (Soft Suspend Mode):
  The module clock f_{CLC} keeps running. Module functions are stopped automatically after internal actions have been finished (for example, after a CAN frame has been sent out). The end of the internal actions is indicated to the fractional divider by a suspend mode acknowledged signal. Due to this behavior, the communication network is not blocked. Furthermore, all registers are accessible for read and write actions. As a result, the debugger can stop the module actions and modify registers. These modifications are taken into account after the Suspend Mode is left.

The Hard Suspend Mode can be enabled/disabled only for the complete MultiCAN module. The Soft Suspend Mode can be individually enabled for each CAN node.

The fractional divider disables module clock f_{CAN} only if all CAN nodes signal that they can be suspended. A CAN node that is not active can always be suspended.

19.3.5 CAN Node Control

Each CAN node may be configured and run independently of the other CAN node. Each CAN node is equipped with its own node control logic to configure the global behavior and to provide status information.

Note: In the following descriptions, index “x” stands for the node number and index “n” represents the message object number.

Configuration Mode is activated when bit NCRx.CCE is set to 1. This mode allows CAN bit timing parameters and the error counter registers to be modified.

CAN Analyze Mode is activated when bit NCRx.CALM is set to 1. In this operation mode, Data And Remote Frames are monitored without active participation in any CAN transfer (CAN transmit pin is held on recessive level). Incoming Remote Frames are stored in a corresponding transmit message object, while arriving data frames are saved in a matching receive message object.

In CAN Analyze Mode, the entire configuration information of the received frame is stored in the corresponding message object, and can be evaluated by the CPU to determine their identifier, XTD bit information and data length code (ID and DLC optionally if the Remote Monitoring Mode is active, bit MOFCRn.RMM = 1). Incoming frames are not acknowledged, and no Error Frames are generated. If CAN Analyze Mode is enabled, Remote Frames are not responded to by the corresponding Data Frame, and Data Frames cannot be transmitted by setting the transmit request bit MOSTATn.TXRQ. Receive interrupts are generated in CAN Analyze Mode (if enabled) for all error free received frames.

The node-specific interrupt configuration is also defined by the Node Control Logic via the NCRx register bits TRIE, ALIE and LECIE:

- If control bit TRIE is set to 1, a transfer interrupt is generated when the NSRx register has been updated (after each successfully completed message transfer).
- If control bit ALIE is set to 1, an error interrupt is generated when a “bus-off” condition has been recognized or the Error Warning Level has been exceeded or under-run. Additionally, list or object errors lead to this type of interrupt.
- If control bit LECIE is set to 1, a last error code interrupt is generated when an error code > 0 is written into bit field NSRx.LEC by hardware.

The Node x Status Register NSRx provides an overview about the current state of the respective CAN node x, comprising information about CAN transfers, CAN node status, and error conditions.

The CAN frame counter can be used to check the transfer sequence of message objects or to obtain information about the instant a frame has been transmitted or received from the associated CAN bus. CAN frame counting is performed by a 16-bit counter, controlled by register NFCRx. Bit fields NFCRx.CFMOD and NFCRx.CFSEL determine the operation mode and the trigger event incrementing the frame counter.

19.3.5.1 Bit Timing Unit

According to the ISO 11898 standard, a CAN bit time is subdivided into different segments (**Figure 19-9**). Each segment consists of multiples of a time quantum t_q. The magnitude of t_q is adjusted by Node x Bit Timing Register bit fields NBTRx.BRP and NBTRx.DIV8, both controlling the baud rate prescaler (register NBTRx is described on **Page 19-81**). The baud rate prescaler is driven by the module timer clock f_{CAN} (generation and control of f_{CAN} is described on **Page 19-111**).

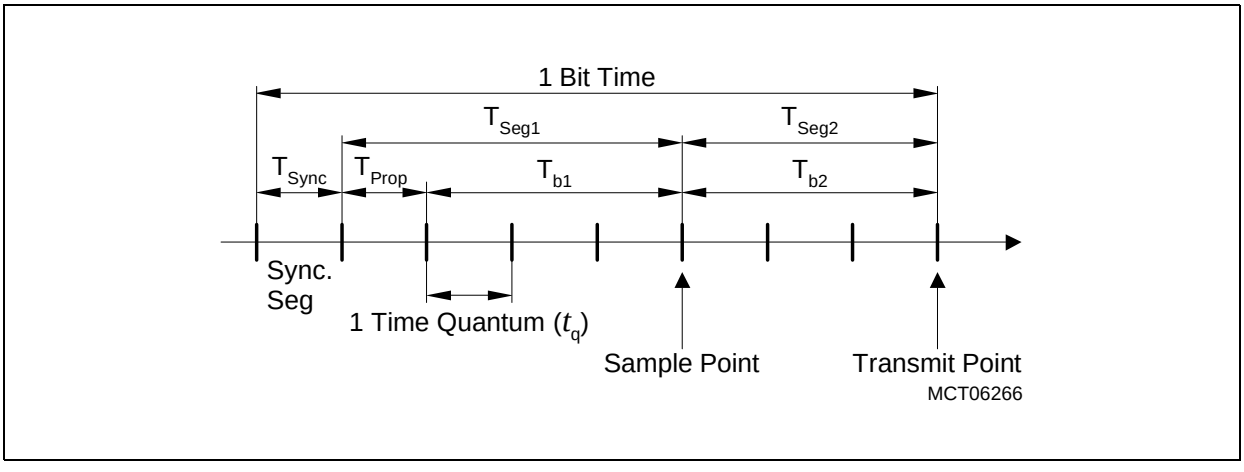


Figure 19-9 CAN Bus Bit Timing Standard

The Synchronization Segment (T_{Sync}) allows a phase synchronization between transmitter and receiver time base. The Synchronization Segment length is always one t_q. The Propagation Time Segment (T_{Prop}) takes into account the physical propagation delay in the transmitter output driver on the CAN bus line and in the transceiver circuit. For a working collision detection mechanism, T_{Prop} must be two times the sum of all propagation delay quantities rounded up to a multiple of t_q. The phase buffer segments 1 and 2 (T_{b1}, T_{b2}) before and after the signal sample point are used to compensate for a mismatch between transmitter and receiver clock phases detected in the synchronization segment.

The maximum number of time quanta allowed for re-synchronization is defined by bit field NBTRx.SJW. The Propagation Time Segment and the Phase Buffer Segment 1 are combined to parameter T_{Seg1}, which is defined by the value NBTRx.TSEG1. A minimum of 3 time quanta is demanded by the ISO standard. Parameter T_{Seg2}, which is defined by the value of NBTRx.TSEG2, covers the Phase Buffer Segment 2. A minimum of 2 time quanta is demanded by the ISO standard. According to ISO standard, a CAN bit time, calculated as the sum of T_{Sync}, T_{Seg1} and T_{Seg2}, must not fall below 8 time quanta.

Calculation of the bit time:

t_q	= (BRP + 1) / f_{CAN}	if DIV8 = 0
	= 8 × (BRP + 1) / f_{CAN}	if DIV8 = 1
T_{Sync}	= 1 × t_q	
T_{Seg1}	= (TSEG1 + 1) × t_q	(min. 3 t_q)
T_{Seg2}	= (TSEG2 + 1) × t_q	(min. 2 t_q)
bit time	= T_{Sync} + T_{Seg1} + T_{Seg2}	(min. 8 t_q)

To compensate phase shifts between clocks of different CAN controllers, the CAN controller must synchronize on any edge from the recessive to the dominant bus level. If the hard synchronization is enabled (at the start of frame), the bit time is restarted at the synchronization segment. Otherwise, the re-synchronization jump width T_{SJW} defines the maximum number of time quanta, a bit time may be shortened or lengthened by one re-synchronization. The value of SJW is defined by bit field NBTRx.SJW.

T_{SJW}	= (SJW + 1) × t_q
T_{Seg1}	≥ T_{SJW} + T_{prop}
T_{Seg2}	≥ T_{SJW}

The maximum relative tolerance for f_{CAN} depends on the Phase Buffer Segments and the re-synchronization jump width.

df_{CAN}	≤ min (T_{b1}, T_{b2}) / 2 × (13 × bit time - T_{b2})	AND
df_{CAN}	≤ T_{SJW} / 20 × bit time	

A valid CAN bit timing must be written to the CAN Node Bit Timing Register NBTR before resetting the INIT bit in the Node Control Register, i.e. before enabling the operation of the CAN node.

The Node Bit Timing Register may be written only if bit CCE (Configuration Change Enable) is set in the corresponding Node Control Register.

19.3.5.2 Bitstream Processor

Based on the message objects in the message buffer, the Bitstream Processor generates the remote and Data Frames to be transmitted via the CAN bus. It controls the CRC generator and adds the checksum information to the new remote or Data Frame. After including the SOF bit and the EOF field, the Bitstream Processor starts the CAN

bus arbitration procedure and continues with the frame transmission when the bus was found in idle state. While the data transmission is running, the Bitstream Processor continuously monitors the I/O line. If (outside the CAN bus arbitration phase or the acknowledge slot) a mismatch is detected between the voltage level on the I/O line and the logic state of the bit currently sent out by the transmit shift register, a CAN error interrupt request is generated, and the error code is indicated by the Node x Status Register bit field NSRx.LEC.

The data consistency of an incoming frame is verified by checking the associated CRC field. When an error has been detected, a CAN error interrupt request is generated and the associated error code is presented in the Node x Status Register NSRx. Furthermore, an Error Frame is generated and transmitted on the CAN bus. After decomposing a faultless frame into identifier and data portion, the received information is transferred to the message buffer executing remote and Data Frame handling, interrupt generation and status processing.

19.3.5.3 Error Handling Unit

The Error Handling Unit of a CAN node x is responsible for the fault confinement of the CAN device. Its two counters, the Receive Error Counter REC and the Transmit Error Counter TEC (bit fields of the Node x Error Counter Register NECNTx, see **Page 19-83**) are incremented and decremented by commands from the Bitstream Processor. If the Bitstream Processor itself detects an error while a transmit operation is running, the Transmit Error Counter is incremented by 8. An increment of 1 is used when the error condition was reported by an external CAN node via an Error Frame generation. For error analysis, the transfer direction of the disturbed message and the node that recognizes the transfer error are indicated for the respective CAN node x in register NECNTx. Depending on the values of the error counters, the CAN node is set into error-active, error-passive, or bus-off state.

The CAN node is in error-active state if both error counters are below the error-passive limit of 128. The CAN node is in error-passive state, if at least one of the error counters is equal to or greater than 128.

The bus-off state is activated if the Transmit Error Counter is equal to or greater than the bus-off limit of 256. This state is reported for CAN node x by the Node x Status Register flag NSRx.BOFF. The device remains in this state, until the “bus-off” recovery sequence is finished. Additionally, Node x Status Register flag NSRx.EWRN is set when at least one of the error counters is equal to or greater than the error warning limit defined by the Node x Error Count Register bit field NECNTx.EWRNLVL. Bit NSRx.EWRN is reset if both error counters fall below the error warning limit again (see **Page 19-71**).

19.3.5.4 CAN Frame Counter

Each CAN node is equipped with a frame counter that counts transmitted/received CAN frames or obtains information about the time when a frame has been started to transmit or be received by the CAN node. CAN frame counting/bit time counting is performed by a 16-bit counter that is controlled by Node x Frame Counter Register NFCRx (see **Page 19-84**). Bit field NFCRx.CFSEL determines the operation mode of the frame counter:

- **Frame Count Mode:**
  After the successful transmission and/or reception of a CAN frame, the frame counter is copied into the CFCVAL bit field of the MOIPRn register of the message object involved in the transfer. Afterwards, the frame counter is incremented.
- **Time Stamp Mode:**
  The frame counter is incremented with the beginning of a new bit time. When the transmission/reception of a frame starts, the value of the frame counter is captured and stored to the CFC bit field of the NFCRx register. After the successful transfer of the frame the captured value is copied to the CFCVAL bit field of the MOIPRn register of the message object involved in the transfer.
- **Bit Timing Mode:**
  Used for baud rate detection and analysis of the bit timing (**Chapter 19.3.7.3**).

19.3.5.5 CAN Node Interrupts

Each CAN node has four hardware triggered interrupt request types that are able to generate an interrupt request upon:

- The successful transmission or reception of a frame
- A CAN protocol error with a last error code
- An alert condition: Transmit/receive error counters reach the warning limit, bus-off state changes, a List Length Error occurs, or a List Object Error occurs
- An overflow of the frame counter

Besides the hardware generated interrupts, software initiated interrupts can be generated using the Module Interrupt Trigger Register MITR. Writing a 1 to bit n of bit field MITR.IT generates an interrupt request signal on the corresponding interrupt output line INT_On. When writing MITR.IT more than one bit can be set resulting in activation of multiple INT_On interrupt output lines at the same time. See also **“Interrupt Control” on Page 19-117** for further processing of the CAN node interrupts.

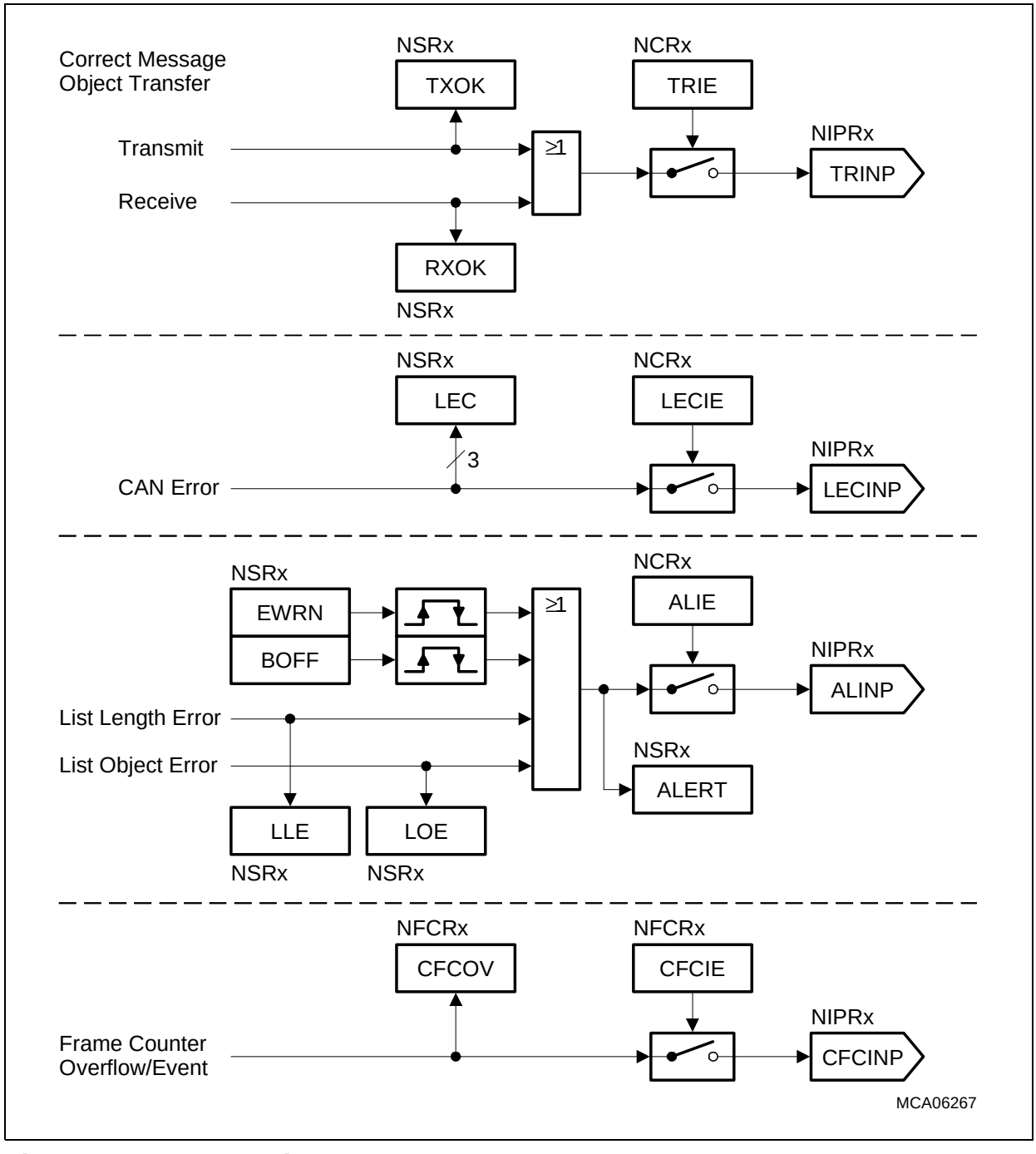


Figure 19-10 CAN Node Interrupts

19.3.6 Message Object List Structure

This section describes the structure of the message object lists in the MultiCAN module.

19.3.6.1 Basics

The message objects of the MultiCAN module are organized in double-chained lists, where each message object has a pointer to the previous message object in the list as well as a pointer to the next message object in the list. The MultiCAN module provides eight lists. Each message object is allocated to one of these lists. In the example in **Figure 19-11**, the three message objects (3, 5, and 16) are allocated to the list with index 2 (List Register LIST2).

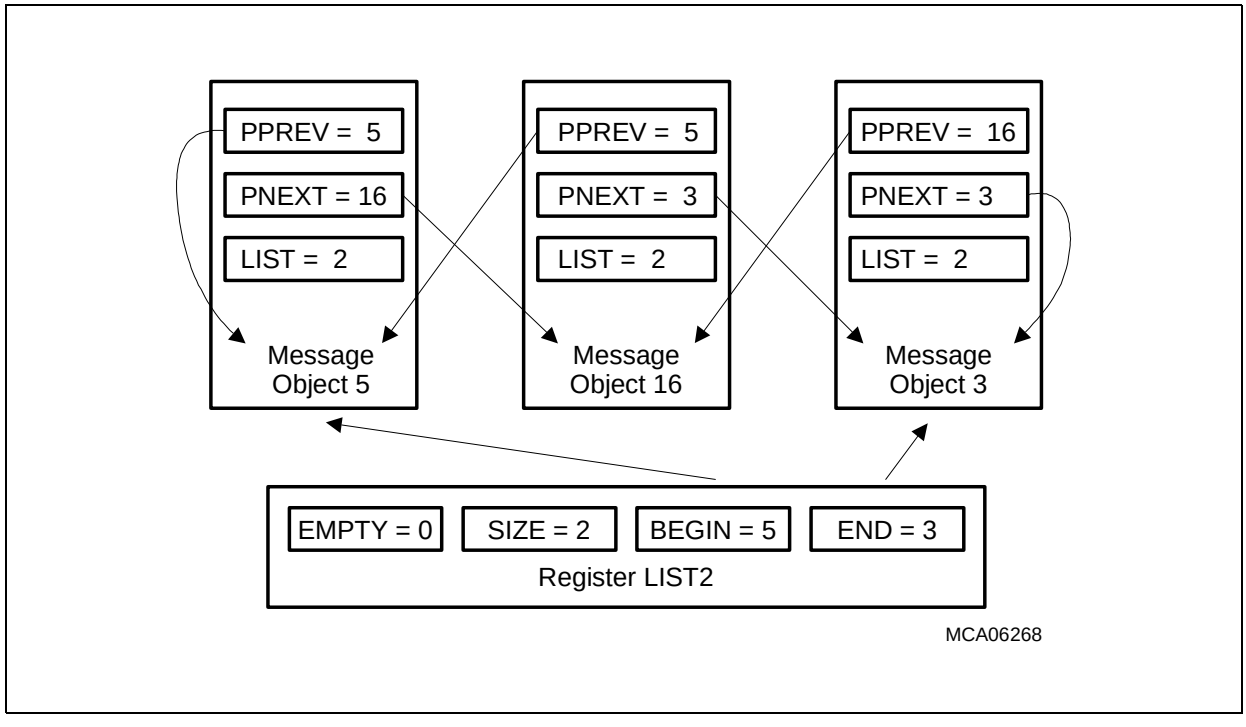


Figure 19-11 Example Allocation of Message Objects to a List

Bit field BEGIN in the List Register (for definition, see **Page 19-65**) points to the first element in the list (object 5 in the example), and bit field END points to the last element in the list (object 3 in the example). The number of elements in the list is indicated by bit field SIZE of the List Register (SIZE = number of list elements - 1, thus SIZE = 2 for the 3 elements in the example). The EMPTY bit of the List Register indicates whether or not a list is empty (EMPTY = 0 in the example, because list 2 is not empty).

Each message object n has a pointer PNEXT in its Message Object n Control Register MOCTRn (see **Page 19-88**) that points to the next message object in the list, and a pointer PPREV that points to the previous message object in the list. PPREV of the first message object points to the message object itself because the first message object has no predecessor (in the example message object 5 is the first message object in the list,

indicated by PPREV = 5). PNEXT of the last message object also points to the message object itself because the last message object has no successor (in the example, object 3 is the last message object in the list, indicated by PNEXT = 3).

Bit field MOCTRn.LIST indicates the list index number to which the message object is currently allocated. The message object of the example are allocated to list 2. Therefore, all LIST bit fields for the message objects assigned to list 2 are set to LIST = 2.

19.3.6.2 List of Unallocated Elements

The list with list index 0 has a special meaning: it is the list of all unallocated elements. An element is called unallocated if it belongs to list 0 (MOCTRn.LIST = 0). It is called allocated if it belongs to a list with an index not equal to 0 (MOCTRn.LIST > 0).

After reset, all message objects are unallocated. This means that they are assigned to the list of unallocated elements with MOCTRn.LIST = 0. After this initial allocation of the message objects caused by reset, the list of all unallocated message objects is ordered by message number (predecessor of message object n is object n-1, successor of object n is object n+1).

19.3.6.3 Connection to the CAN Nodes

Each CAN node is linked to one unique list of message objects. A CAN node performs message transfer only with the message objects that are allocated to the list of the CAN node. This is illustrated in **Figure 19-12**. Frames that are received on a CAN node may only be stored in one of the message objects that belongs to the CAN node; frames to be transmitted on a CAN node are selected only from the message objects that are allocated to that node, as indicated by the vertical arrows.

There are more lists (eight) than CAN nodes (two). This means that some lists are not linked to one of the CAN nodes. A message object that is allocated to one of these unlinked lists cannot receive messages directly from a CAN node and it may not transmit messages.

FIFO and gateway mechanisms refer to message numbers and not directly to a specific list. The user must take care that the message objects targeted by FIFO/gateway belong to the desired list. The mechanisms make it possible to work with lists that do not belong to the CAN node.

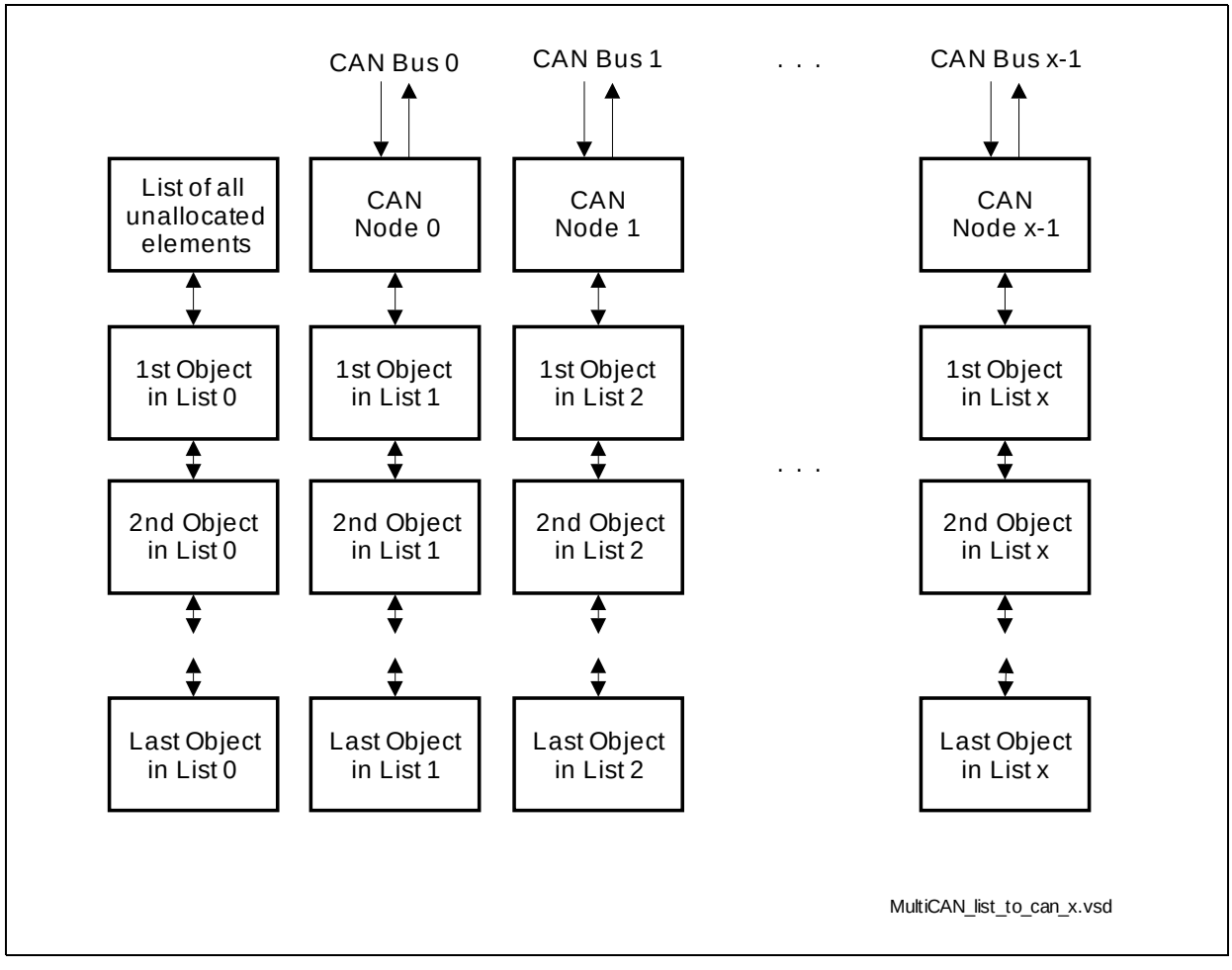


Figure 19-12 Message Objects Linked to CAN Nodes

19.3.6.4 List Command Panel

The list structure cannot be modified directly by write accesses to the LIST registers and the PPREV, PNEXT and LIST bit fields in the Message Object Control Registers, as they are read only. The list structure is managed by and limited to the list controller inside the MultiCAN module. The list controller is controlled via a command panel allowing the user to issue list allocation commands to the list controller. The list controller has two main purposes:

1. Ensure that all operations that modify the list structure result in a consistent list structure.
2. Present maximum ease of use and flexibility to the user.

The list controller and the associated command panel allows the programmer to concentrate on the final properties of the list, which are characterized by the allocation of message objects to a CAN node, and the ordering relation between objects that are allocated to the same list. The process of list (re-)building is done in the list controller.

Table 19-3 gives an overview on the available panel commands while Table 19-7 on Page 19-60 describes the panel commands in more detail.

Table 19-3 Panel Commands Overview

Command Name	Description
No Operation	No new command is started.
Initialize Lists	Run the initialization sequence to reset the CTRL and LIST field of all message objects.
Static Allocate	Allocate message object to a list.
Dynamic Allocate	Allocate the first message object of the list of unallocated objects to the selected list.
Static Insert Before	Remove a message object (source object) from the list that it currently belongs to, and insert it before a given destination object into the list structure of the destination object.
Dynamic Insert Before	Insert a new message object before a given destination object.
Static Insert Behind	Remove a message object (source object) from the list that it currently belongs to, and insert it behind a given destination object into the list structure of the destination object.
Dynamic Insert Behind	Insert a new message object behind a given destination object.

A panel command is started by writing the respective command code into the Panel Control Register bit field PANCTR.PANCMD (see Page 19-59). The corresponding command arguments must be written into bit fields PANCTR.PANAR1 and PANCTR.PANAR2 before writing the command code, or latest along with the command code in a single 32-bit write access to the Panel Control Register.

With the write operation of a valid command code, the PANCTR.BUSY flag is set and further write accesses to the Panel Control Register are ignored. The BUSY flag remains active and the control panel remains locked until the execution of the requested command has been completed. After a reset, the list controller builds up list 0. During this operation, BUSY is set and other accesses to the CAN RAM are forbidden. The CAN RAM can be accessed again when BUSY becomes inactive.

Note: The CAN RAM is automatically initialized after reset by the list controller in order to ensure correct list pointers in each message object. The end of this CAN RAM initialization is indicated by bit PANCTR.BUSY becoming inactive.

In case of a dynamic allocation command that takes an element from the list of unallocated objects, the PANCTR.RBUSY bit is also set along with the BUSY bit

(RBUSY = BUSY = 1). This indicates that bit fields PANAR1 and PANAR2 are going to be updated by the list controller in the following way:

1. The message number of the message object taken from the list of unallocated elements is written to PANAR1.
2. If ERR (bit 7 of PANAR2) is set to 1, the list of unallocated elements was empty and the command is aborted. If ERR is 0, the list was not empty and the command will be performed successfully.

The results of a dynamic allocation command are written before the list controller starts the actual allocation process. As soon as the results are available, RBUSY becomes inactive (RBUSY = 0) again, while BUSY still remains active until completion of the command. This allows the user to set up the new message object while it is still in the process of list allocation. The access to message objects is not limited during ongoing list operations. However, any access to a register resource located inside the RAM delays the ongoing allocation process by one access cycle.

As soon as the command is finished, the BUSY flag becomes inactive (BUSY = 0) and write accesses to the Panel Control Register are enabled again. Also, the “No Operation” command code is automatically written to the PANCTR.PANCMD field. A new command may be started any time when BUSY = 0.

All fields of the Panel Control Register PANCTR except BUSY and RBUSY may be written by the user. This makes it possible to save and restore the Panel Control Register if the Command Panel is used within independent (mutually interruptible) interrupt service routines. If this is the case, any task that uses the Command Panel and that may interrupt another task that also uses the Command Panel should poll the BUSY flag until it becomes inactive and save the whole PANCTR register to a memory location before issuing a command. At the end of the interrupt service routine, the task should restore PANCTR from the memory location.

Before a message object that is allocated to the list of an active CAN node shall be moved to another list or to another position within the same list, bit MOCTRn.MSGVAL (“Message Valid”) of message object n must be cleared.

19.3.7 CAN Node Analysis Features

The chapter describes the CAN node analysis capabilities of the MultiCAN module.

19.3.7.1 Analyze Mode

The CAN Analyze Mode makes it possible to monitor the CAN traffic for each CAN node individually without affecting the logical state of the CAN bus. The CAN Analyze Mode for CAN node x is selected by setting Node x Control Register bit NCRx.CALM.

In CAN Analyze Mode, the transmit pin of a CAN node is held at a recessive level permanently. The CAN node may receive frames (Data, Remote, and Error Frames) but is not allowed to transmit. Received Data/Remote Frames are not acknowledged (i.e. acknowledge slot is sent recessive) but will be received and stored in matching message objects as long as there is any other node that acknowledges the frame. The complete message object functionality is available, but no transmit request will be executed.

19.3.7.2 Loop-Back Mode

The MultiCAN module provides a Loop-Back Mode to enable an in-system test of the MultiCAN module as well as the development of CAN driver software without access to an external CAN bus.

The loop-back feature consists of an internal CAN bus (inside the MultiCAN module) and a bus select switch for each CAN node (see **Figure 19-13**). With the switch, each CAN node can be connected either to the internal CAN bus (Loop-Back Mode activated) or the external CAN bus, respectively to transmit and receive pins (normal operation). The CAN bus that is not currently selected is driven recessive; this means the transmit pin is held at 1, and the receive pin is ignored by the CAN nodes that are in Loop-Back Mode.

The Loop-Back Mode is selected for CAN node x by setting the Node x Port Control Register bit NPCRx.LBM. All CAN nodes that are in Loop-Back Mode may communicate together via the internal CAN bus without affecting the normal operation of the other CAN nodes that are not in Loop-Back Mode.

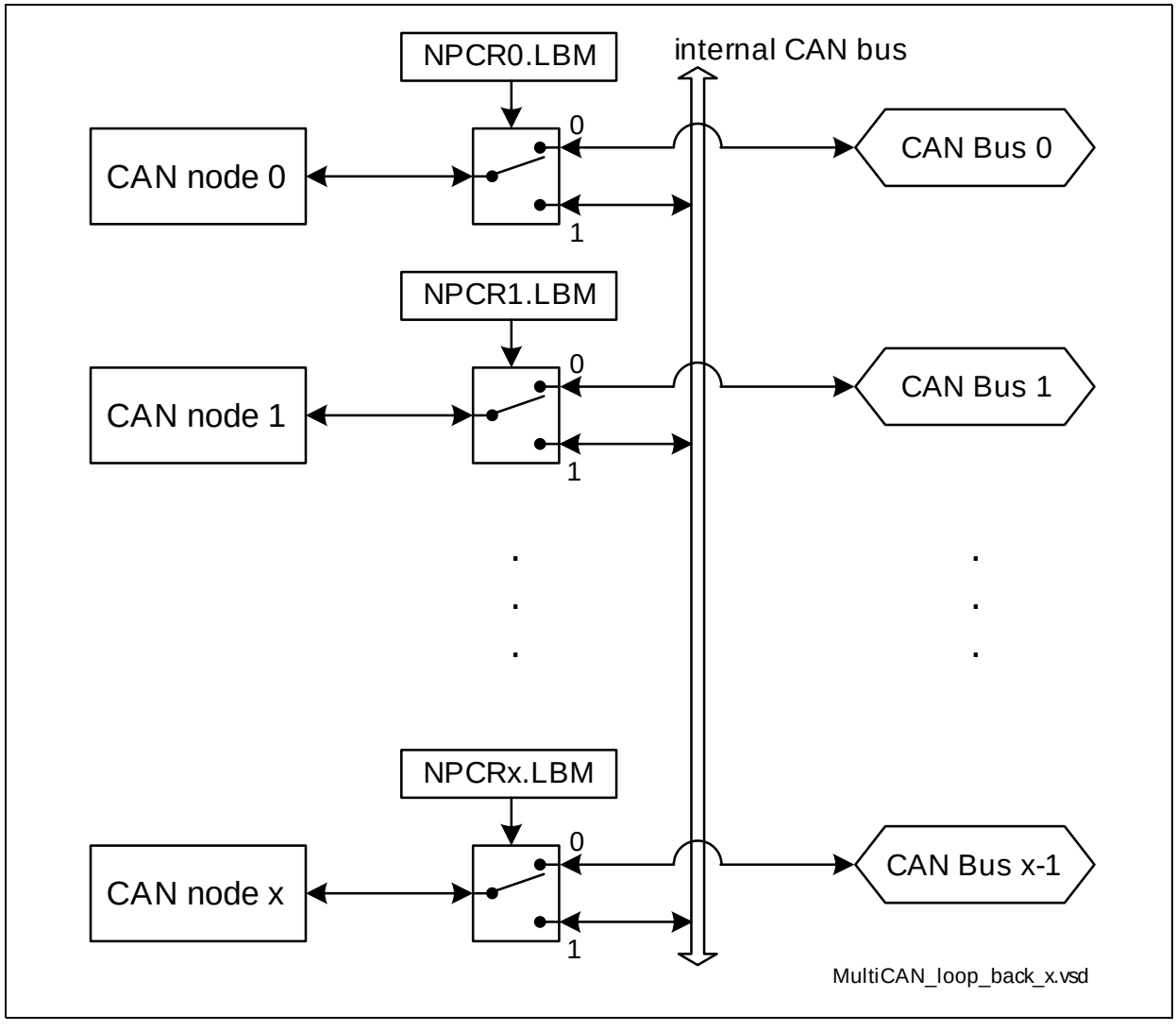


Figure 19-13 Loop-Back Mode

19.3.7.3 Bit Timing Analysis

Detailed analysis of the bit timing can be performed for each CAN node using the analysis modes of the CAN frame counter. The bit timing analysis functionality of the frame counter may be used for automatic detection of the CAN baud rate, as well as to analyze the timing of the CAN network.

Bit timing analysis for CAN node x is selected when bit field NFCRx.CFMOD = 10_B. Bit timing analysis does not affect the operation of the CAN node.

The bit timing measurement results are written into the NFCRx.CFC bit field. Whenever NFCRx.CFC is updated in bit timing analysis mode, bit NFCRx.CFCOV is also set to indicate the CFC update event. If NFCRx.CFCIE is set, an interrupt request can be generated (see **Figure 19-10**).

Automatic Baud Rate Detection

For automatic baud rate detection, the time between the observation of subsequent dominant edges on the CAN bus must be measured. This measurement is automatically performed if bit field NFCRx.CFSEL = 000_B. With each dominant edge monitored on the CAN receive input line, the time (measured in f_{CAN} clock cycles) between this edge and the most recent dominant edge is stored in the NFCRx.CFC bit field.

Synchronization Analysis

The bit time synchronization is monitored if NFCRx.CFSEL = 010_B. The time between the first dominant edge and the sample point is measured and stored in the NFCRx.CFC bit field. The bit timing synchronization offset may be derived from this time as the first edge after the sample point triggers synchronization and there is only one synchronization between consecutive sample points.

Synchronization analysis can be used, for example, for fine tuning of the baud rate during reception of the first CAN frame with the measured baud rate.

Driver Delay Measurement

The delay between a transmitted edge and the corresponding received edge is measured when NFCRx.CFSEL = 011_B (dominant to dominant) and NFCRx.CFSEL = 100_B (recessive to recessive). These delays indicate the time needed to represent a new bit value on the physical implementation of the CAN bus.

19.3.8 Message Acceptance Filtering

The chapter describes the Message Acceptance Filtering capabilities of the MultiCAN module.

19.3.8.1 Receive Acceptance Filtering

When a CAN frame is received by a CAN node, a unique message object is determined in which the received frame is stored after successful frame reception. A message object is qualified for reception of a frame if the following six conditions are met.

- The message object is allocated to the message object list of the CAN node by which the frame is received.
- Bit MOSTATn.MSGVAL in the Message Status Register (see **Page 19-91**) is set.
- Bit MOSTATn.RXEN is set.
- Bit MOSTATn.DIR is equal to bit RTR of the received frame.
  If bit MOSTATn.DIR = 1 (transmit object), the message object accepts only Remote Frames. If bit MOSTATn.DIR = 0 (receive object), the message object accepts only Data Frames.
- If bit MOAMRn.MIDE = 1, the IDE bit of the received frame becomes evaluated in the following way: If MOARn.IDE = 1, the IDE bit of the received frame must be set (indicates extended identifier). If MOARn.IDE = 0, the IDE bit of the received frame must be cleared (indicates standard identifier).
  If bit MOAMRn.MIDE = 0, the IDE bit of the received frame is “don’t care”. In this case, message objects with standard and extended frames are accepted.
- The identifier of the received frame matches the identifier stored in the Arbitration Register of the message object as qualified by the acceptance mask in the MOAMRn register. This means that each bit of the received message object identifier is equal to the bit field MOARn.ID, except those bits for which the corresponding acceptance mask bits in bit field MOAMRn.AM are cleared. These identifier bits are “don’t care” for reception. **Figure 19-14** illustrates this receive message identifier check.

Among all messages that fulfill all six qualifying criteria the message object with the highest receive priority wins receive acceptance filtering and becomes selected to store the received frame. All other message objects lose receive acceptance filtering.

The following priority scheme is defined for the message objects:

A message object a (MOa) has higher receive priority than a message object b (MOb) if the following two conditions are fulfilled (see **Page 19-105**):

1. MOa has a higher priority class than MOb. This means, the 2-bit priority bit field MOARa.PRI must be equal or less than bit field MOARb.PRI.
2. If both message objects have the same priority class (MOARa.PRI = MOARb.PRI), MOb is a list successor of MOa. This means that MOb can be reached by means of successively stepping forward in the list, starting from a.

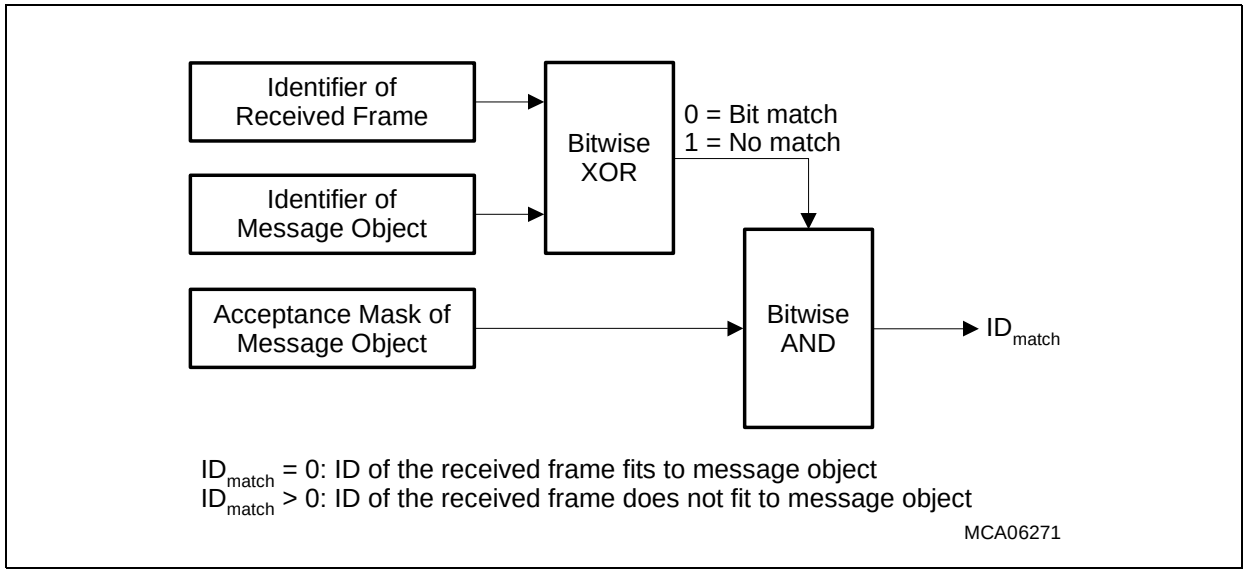


Figure 19-14 Received Message Identifier Acceptance Check

19.3.8.2 Transmit Acceptance Filtering

A message is requested for transmission by setting a transmit request in the message object that holds the message. If more than one message object have a valid transmit request for the same CAN node, one of these message objects is chosen for transmission, because only a single message object can be transmitted at one time on a CAN bus.

A message object is qualified for transmission on a CAN node if the following four conditions are met (see also **Figure 19-15**).

1. The message object is allocated to the message object list of the CAN node.
2. Bit MOSTATn.MSGVAL is set.
3. Bit MOSTATn.TXRQ is set.
4. Bit MOSTATn.TXEN0 and MOSTATn.TXEN1 are set.

A priority scheme determines which one of all qualifying message objects is transmitted first. It is assumed that message object a (MOa) and message object b (MOb) are two message objects qualified for transmission. MOb is a list successor of MOa. For both message objects, CAN messages CANa and CANb are defined (identifier, IDE, and RTR are taken from the message-specific bit fields and bits MOARn.ID, MOARn.IDE and MOCTRn.DIR).

If both message objects belong to the same priority class (identical PRI bit field in register MOARn), MOa has a higher transmit priority than MOb if one of the following conditions is fulfilled.

- PRI = 10_B and CAN message MOa has higher or equal priority than CAN message MOb with respect to CAN arbitration rules (see **Table 19-13** on **Page 19-106**).
- PRI = 01_B or PRI = 11_B (priority by list order).

The message object that is qualified for transmission and has highest transmit priority wins the transmit acceptance filtering, and will be transmitted first. All other message objects lose the current transmit acceptance filtering round. They get a new chance in subsequent acceptance filtering rounds.

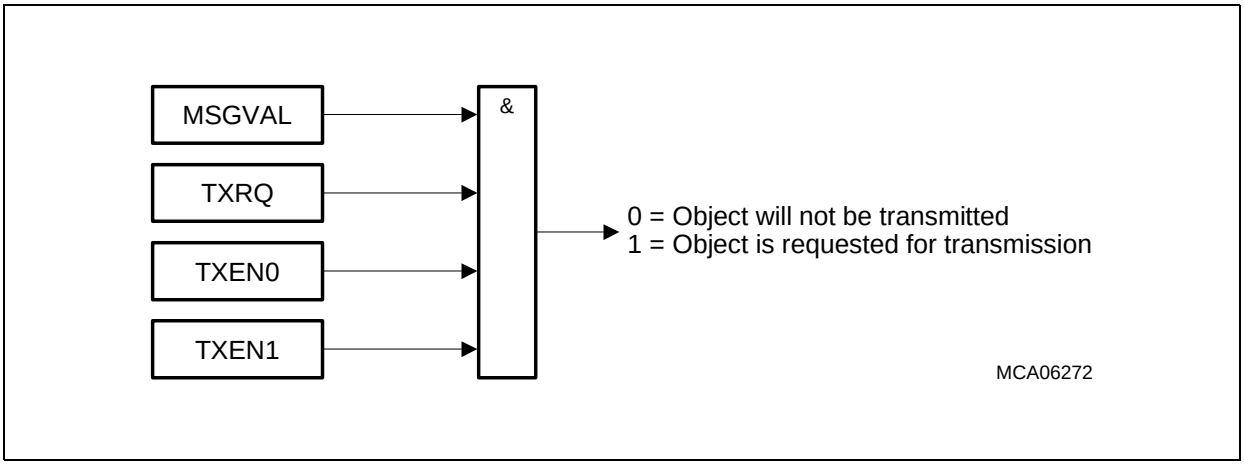


Figure 19-15 Effective Transmit Request of Message Object

19.3.9 Message Postprocessing

After a message object has successfully received or transmitted a frame, the CPU can be notified to perform a postprocessing on the message object. The postprocessing of the MultiCAN module consists of two elements:

1. Message interrupts to trigger postprocessing.
2. Message pending registers to collect pending message interrupts into a common structure for postprocessing.

19.3.9.1 Message Object Interrupts

When the storage of a received frame into a message object or the successful transmission of a frame is completed, a message interrupt can be issued. For each message object, a transmit and a receive interrupt can be generated and routed to one of the sixteen CAN interrupt output lines (see **Figure 19-16**). A receive interrupt occurs also after a frame storage event that has been induced by a FIFO or a gateway action. The status bits TXPND and RXPND in the Message Object n Status Register are always set after a successful transmission/reception, whether or not the respective message interrupt is enabled.

A third FIFO full interrupt condition of a message object is provided. If bit field MOFCRn.OVIE (Overflow Interrupt Enable) is set, the FIFO full interrupt will be activated depending on the actual message object type.

In case of a Receive FIFO Base Object (MOFCRn.MMC = 0001_B), the FIFO full interrupt is routed to the interrupt output line INT_Om as defined by the transmit interrupt node pointer MOIPRn.TXINP.

In case of a Transmit FIFO Base Object (MOFCRn.MMC = 0010_B), the FIFO full interrupt becomes routed to the interrupt output line INT_Om as defined by the receive interrupt node pointer MOIPRn.RXINP.

See also **“Interrupt Control” on Page 19-117** for further processing of the message object interrupts.

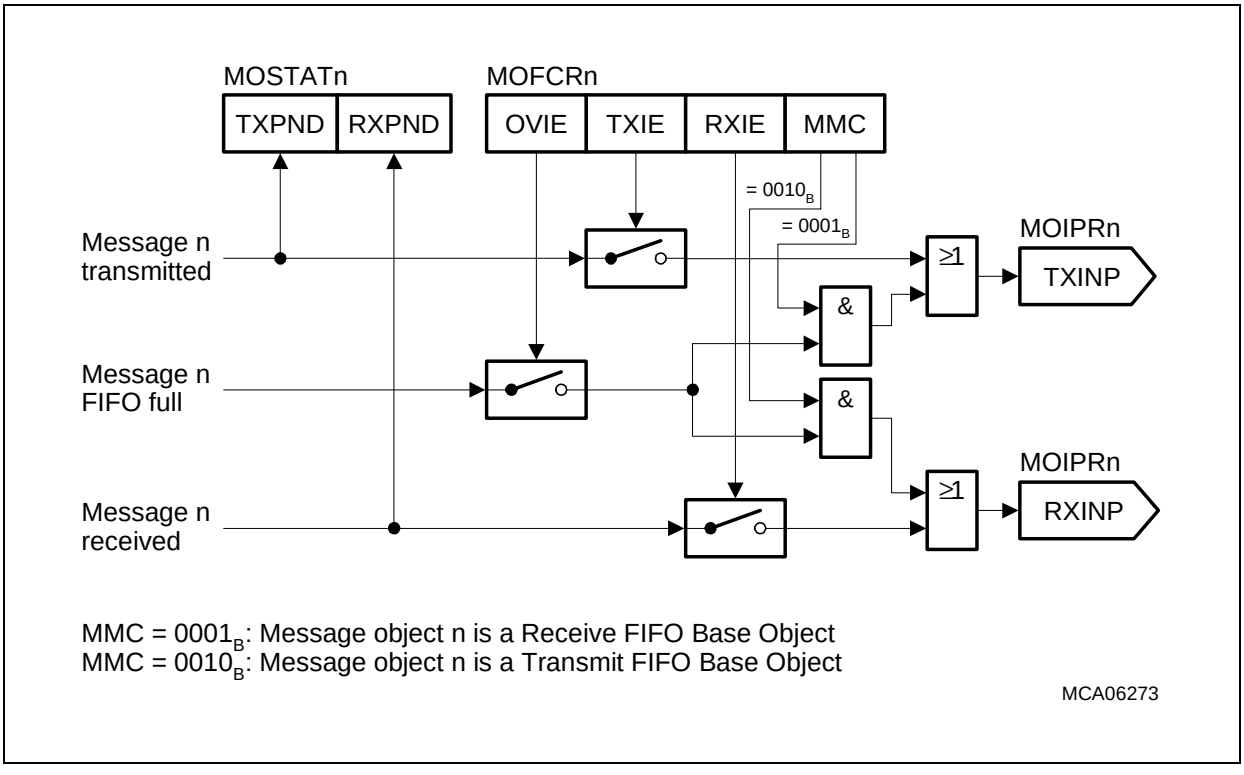


Figure 19-16 Message Interrupt Request Routing

19.3.9.2 Pending Messages

When a message interrupt request is generated, a message pending bit is set in one of the Message Pending Registers. There are 8 Message Pending Registers, MSPNDk (k = 0-7) with 32 pending bits available each. The general **Figure 19-17** shows the allocation of the message pending bits in case that the maximum possible number of eight Message Pending Registers are implemented and available on the chip.

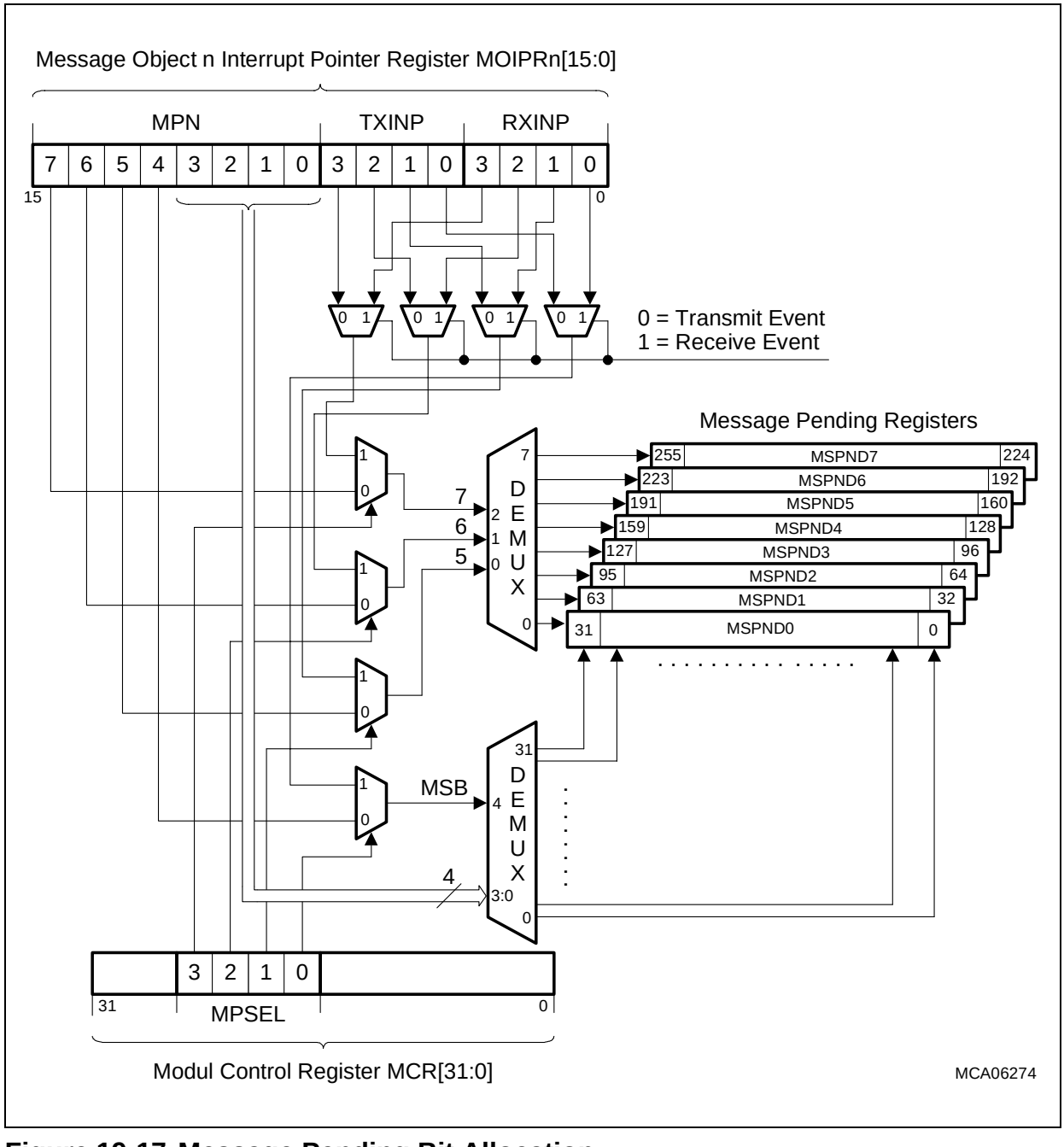


Figure 19-17 Message Pending Bit Allocation

The location of a pending bit is defined by two demultiplexers selecting the number k of the MSPNDk registers (3-bit demux), and the bit location within the corresponding MSPNDk register (5-bit demux).

Allocation Case 1

In this allocation case, bit field MCR.MPSEL = 0000_B (see **Page 19-63**). The location selection consists of 2 parts:

- The upper three bits of MOIPRn.MPN (MPN[7:5]) select the number k of a Message Pending Register MSPNDk in which the pending bit will be set.
- The lower five bits of MOIPRn.MPN (MPN[4:0]) select the bit position (0-31) in MSPNDk for the pending bit to be set.

Allocation Case 2

In this allocation case, bit field MCR.MPSEL is taken into account for pending bit allocation. Bit field MCR.MPSEL makes it possible to include the interrupt request node pointer for reception (MOIPRn.RXINP) or transmission (MOIPRn.TXINP) for pending bit allocation in such a way that different target locations for the pending bits are used in receive and transmit case. If MPSEL = 1111_B, the location selection operates in the following way:

- At a transmit event, the upper 3 bits of TXINP determine the number k of a Message Pending Register MSPNDk in which the pending bit will be set. At a receive event, the upper 3 bits of RXINP determine the number k.
- The bit position (0-31) in MSPNDk for the pending bit to be set is selected by the lowest bit of TXINP or RXINP (selects between low and high half-word of MSPNDk) and the four least significant bits of MPN.

General Hints

The Message Pending Registers MSPNDk can be written by software. Bits that are written with 1 are left unchanged, and bits which are written with 0 are cleared. This makes it possible to clear individual MSPNDk bits with a single register write access. Therefore, access conflicts are avoided when the MultiCAN module (hardware) sets another pending bit at the same time when software writes to the register.

Each Message Pending Register MSPNDk is associated with a Message Index Register MSIDk (see **Page 19-68**) which indicates the lowest bit position of all set (1) bits in Message Pending Register k. The MSIDk register is a read-only register that is updated immediately when a value in the corresponding Message Pending Register k is changed by software or hardware.

19.3.10 Message Object Data Handling

This chapter describes the handling capabilities for the Message Object Data of the MultiCAN module.

19.3.10.1 Frame Reception

After the reception of a message, it is stored in a message object according to the scheme shown in **Figure 19-18**. The MultiCAN module not only copies the received data into the message object, and it provides advanced features to enable consistent data exchange between MultiCAN and CPU.

MSGVAL

Bit MSGVAL (Message Valid) in the Message Object n Status Register MOSTATn is the main switch of the message object. During the frame reception, information is stored in the message object only when MSGVAL = 1. If bit MSGVAL is reset by the CPU, the MultiCAN module stops all ongoing write accesses to the message object. Now the message object can be re-configured by the CPU with subsequent write accesses to it without being disturbed by the MultiCAN.

RTSEL

When the CPU re-configures a message object during CAN operation (for example, clears MSGVAL, modifies the message object and sets MSGVAL again), the following scenario can occur:

1. The message object wins receive acceptance filtering.
2. The CPU clears MSGVAL to re-configure the message object.
3. The CPU sets MSGVAL again after re-configuration.
4. The end of the received frame is reached. As MSGVAL is set, the received data is stored in the message object, a message interrupt request is generated, gateway and FIFO actions are processed, etc.

After the re-configuration of the message object (after step 3 above) the storage of further received data may be undesirable. This can be achieved through bit MOCTRn.RTSEL (Receive/Transmit Selected) that makes it possible to disconnect a message object from an ongoing frame reception.

When a message object wins the receive acceptance filtering, its RTSEL bit is set by the MultiCAN module to indicate an upcoming frame delivery. The MultiCAN module checks RTSEL whether it is set on successful frame reception to verify that the object is still ready for receiving the frame. The received frame is then stored in the message object (along with all subsequent actions such as message interrupts, FIFO & gateway actions, flag updates) only if RTSEL = 1.

When a message object is invalidated during CAN operation (resetting bit MSGVAL), RTSEL should be cleared before setting MSGVAL again (latest with the same write

access that sets MSGVAL) to prevent the storage of a frame that belongs to the old context of the message object. Therefore, a message object re-configuration should consist of the following steps:

1. Clear MSGVAL bit
2. Re-configure the message object while MSGVAL = 0
3. Clear RTSEL bit and set MSGVAL again

RXEN

Bit MOSTATn.RXEN enables a message object for frame reception. A message object can receive CAN messages from the CAN bus only if RXEN = 1. The MultiCAN module evaluates RXEN only during receive acceptance filtering. After receive acceptance filtering, RXEN is ignored and has no further influence on the actual storage of a received message in a message object.

Bit RXEN enables the “soft phase out” of a message object: after clearing RXEN, a currently received CAN message for which the message object has won acceptance filtering is still stored in the message object but for subsequent messages the message object no longer wins receive acceptance filtering.

RXUPD, NEWDAT and MSGLST

An ongoing frame storage process is indicated by the RXUPD (Receive Updating) flag in the MOSTATn register. RXUPD is set with the start and cleared with the end of a message object update, which consists of frame storage as well as flag updates.

After storing the received frame (identifier, IDE bit, DLC; including the Data Field for Data Frames), the NEWDAT (New Data) bit of the message object is set. If NEWDAT was already set before it becomes set again, bit MSGLST (Message Lost) is set to indicate a data loss condition.

The RXUPD and NEWDAT flags can help to read consistent frame data from the message object during an ongoing CAN operation. The following steps are recommended to be executed:

1. Clear NEWDAT bit.
2. Read message content (identifier, data etc.) from the message object.
3. Check that both, NEWDAT and RXUPD, are cleared. If this is not the case, go back to step 1.
4. When step 3 was successful, the message object contents are consistent and has not been updated by the MultiCAN module while reading.

Bits RXUPD, NEWDAT and MSGLST have the same behavior for the reception of Data as well as Remote Frames.

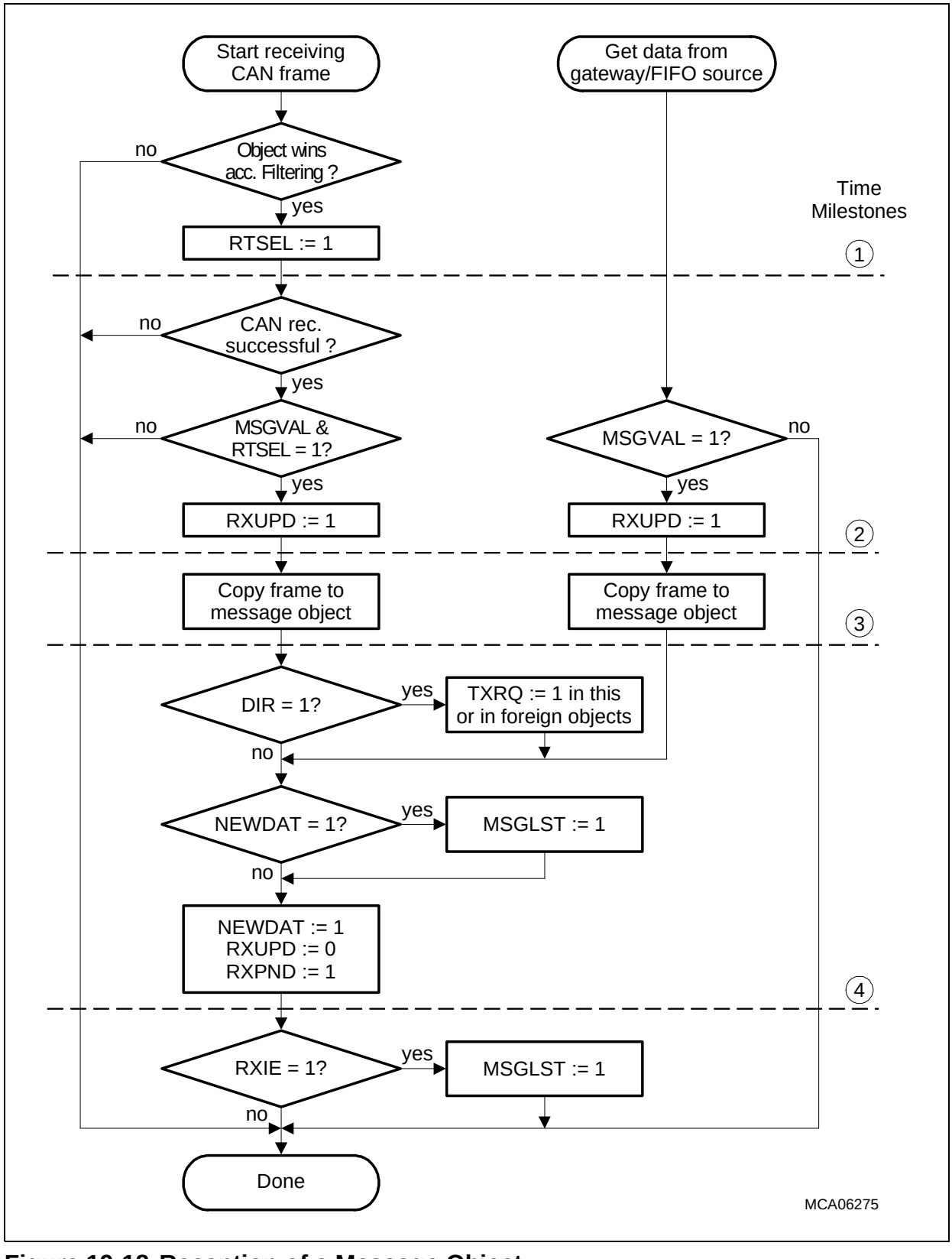


Figure 19-18 Reception of a Message Object

19.3.10.2 Frame Transmission

The process of a message object transmission is shown in **Figure 19-19**. Along with the copy of the message object content to be transmitted (identifier, IDE bit, RTR = DIR bit, DLC, including the Data Field for Data Frames) into the internal transmit buffer of the assigned CAN node, several status flags are also served and monitored to control consistent data handling.

The transmission process of a message object starting after the transmit acceptance filtering is identical for Remote and Data Frames.

MSGVAL, TXRQ, TXEN0, TXEN1

A message can only be transmitted if all four bits in registers MOSTATn, MSGVAL (Message Valid), TXRQ (Transmit Request), TXEN0 (Transmit Enable 0), TXEN1 (Transmit Enable 1) are set as shown in **Figure 19-15**. Although these bits are equivalent with respect to the transmission process, they have different semantics:

Table 19-4 Message Transmission Bit Definitions

Bit	Description
MSGVAL	**Message Valid** This is the main switch bit of the message object.
TXRQ	**Transmit Request** This is the standard transmit request bit. This bit must be set whenever a message object should be transmitted. TXRQ is cleared by hardware at the end of a successful transmission, except when there is new data (indicated by NEWDAT = 1) to be transmitted. When bit MOFCRn.STT (“Single Transmit Trial”) is set, TXRQ becomes already cleared when the contents of the message object are copied into the transmit frame buffer of the CAN node. A received remote request (after a Remote Frame reception) sets bit TXRQ to request the transmission of the requested data frame.
TXEN0	**Transmit Enable 0** This bit can be temporarily cleared by software to suppress the transmission of this message object when it writes new content to the Data Field. This avoids transmission of inconsistent frames that consist of a mixture of old and new data. Remote requests are still accepted when TXEN0 = 0, but transmission of the Data Frame is suspended until transmission is re-enabled by software (setting TXEN0).

Table 19-4 Message Transmission Bit Definitions (cont'd)

Bit	Description
TXEN1	**Transmit Enable 1** This bit is used in transmit FIFOs to select the message object that is transmit active within the FIFO structure. For message objects that are not transmit FIFO elements, TXEN1 can either be set permanently to 1 or can be used as a second independent transmission enable bit.

RTSEL

When a message object has been identified to be transmitted next after transmission acceptance filtering, bit MOCTRn.RTSEL (Receive/Transmit Selected) is set.

When the message object is copied into the internal transmit buffer, bit RTSEL is checked, and the message is transmitted only if RTSEL = 1. After the successful transmission of the message, bit RTSEL is checked again and the message postprocessing is only executed if RTSEL = 1.

For a complete re-configuration of a valid message object, the following steps should be executed:

1. Clear MSGVAL bit
2. Re-configure the message object while MSGVAL = 0
3. Clear RTSEL and set MSGVAL

Clearing of RTSEL ensures that the message object is disconnected from an ongoing/scheduled transmission and no message object processing (copying message to transmit buffer including clearing NEWDAT, clearing TXRQ, time stamp update, message interrupt, etc.) within the old context of the object can occur after the message object becomes valid again, but within a new context.

NEWDAT

When the contents of a message object have been transferred to the internal transmit buffer of the CAN node, bit MOSTATn.NEWDAT (New Data) is cleared by hardware to indicate that the transmit message object data is no longer new.

When the transmission of the frame is successful and NEWDAT is still cleared (if no new data has been copied into the message object meanwhile), TXRQ (Transmit Request) is cleared automatically by hardware.

If, however, the NEWDAT bit has been set again by the software (because a new frame should be transmitted), TXRQ is not cleared to enable the transmission of the new data.

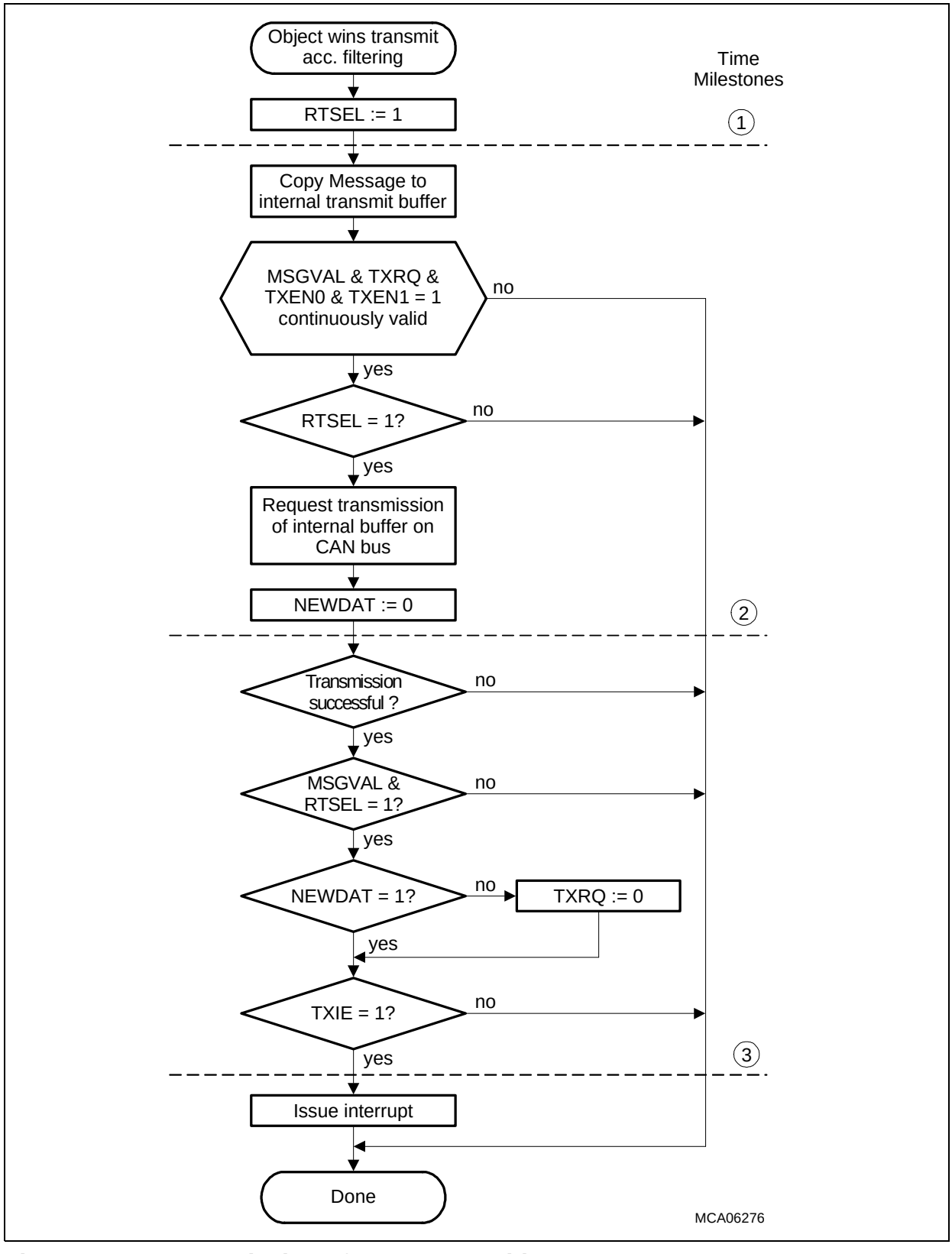


Figure 19-19 Transmission of a Message Object

19.3.11 Message Object Functionality

This chapter describes the functionality of the Message Objects in the MultiCAN module.

19.3.11.1 Standard Message Object

A message object is selected as standard message object when bit field MOFCRn.MMC = 0000_B (see **Page 19-84**). The standard message object can transmit and receive CAN frames according to the basic rules described in the previous sections. Additional services such as Single Data Transfer Mode or Single Transmit Trial (see following sections) are available and can be individually selected.

19.3.11.2 Single Data Transfer Mode

Single Data Transfer Mode is a useful feature in order to broadcast data over the CAN bus without unintended duplication of information. Single Data Transfer Mode is selected via bit MOFCRn.SDT.

Message Reception

When a received message stored in a message object is overwritten by a new received message, the contents of the first message are lost and replaced with the contents of the new received message (indicated by MSGLST = 1).

If SDT is set (Single Data Transfer Mode activated), bit MSGVAL of the message object is automatically cleared by hardware after the storage of a received Data Frame. This prevents the reception of further messages.

After the reception of a Remote Frame, bit MSGVAL is not automatically cleared.

Message Transmission

When a message object receives a series of multiple remote requests, it transmits several Data Frames in response to the remote requests. If the data within the message object has not been updated in the time between the transmissions, the same data can be sent more than once on the CAN bus.

In Single Data Transfer Mode (SDT = 1), this is avoided because MSGVAL is automatically cleared after the successful transmission of a Data Frame.

After the transmission of a Remote Frame, bit MSGVAL is not automatically cleared.

19.3.11.3 Single Transmit Trial

If the bit STT in the message object function register is set (STT = 1), the transmission request is cleared (TXRQ = 0) when the frame contents of the message object have been copied to the internal transmit buffer of the CAN node. Thus, the transmission of the message object is not tried again when it fails due to CAN bus errors.

19.3.11.4 Message Object FIFO Structure

In case of high CPU load it may be difficult to process a series of CAN frames in time. This may happen if multiple messages are received or must be transmitted in short time.

Therefore, a FIFO buffer structure is available to avoid loss of incoming messages and to minimize the setup time for outgoing messages. The FIFO structure can also be used to automate the reception or transmission of a series of CAN messages and to generate a single message interrupt when the whole CAN frame series is done.

There can be several FIFOs in parallel. The number of FIFOs and their size are limited only by the number of available message objects. A FIFO can be installed, resized and de-installed at any time, even during CAN operation.

The basic structure of a FIFO is shown in **Figure 19-20**. A FIFO consists of one base object and n slave objects. The slave objects are chained together in a list structure (similar as in message object lists). The base object may be allocated to any list. Although **Figure 19-20** shows the base object as a separate part beside the slave objects, it is also possible to integrate the base object at any place into the chain of slave objects. This means that the base object is slave object, too (not possible for gateways). The absolute object numbers of the message objects have no impact on the operation of the FIFO.

The base object does not need to be allocated to the same list as the slave objects. Only the slave object must be allocated to a common list (as they are chained together). Several pointers (BOT, CUR and TOP) that are located in the Message Object n FIFO/Gateway Pointer Register MOFGPRn link the base object to the slave objects, regardless whether the base object is allocated to the same or to another **list** than the slave objects.

The smallest FIFO would be a single message object which is both, FIFO base and FIFO slave (not very useful). The biggest possible FIFO structure would include all message objects of the MultiCAN module. Any FIFO sizes between these limits are possible.

In the FIFO base object, the FIFO boundaries are defined. Bit field MOFGPRn.BOT of the base object points to (includes the number of) the bottom slave object in the FIFO structure. The MOFGPRn.TOP bit field points to (includes the number of) the top slave object in the FIFO structure. The MOFGPRn.CUR bit field points to (includes the number of) the slave object that is actually selected by the MultiCAN module for message transfer. When a message transfer takes place with this object, CUR is set to the next message object in the list structure of the slave objects (CUR = PNEXT of current object). If CUR was equal to TOP (top of the FIFO reached), the next update of CUR will result in CUR = BOT (wrap-around from the top to the bottom of the FIFO). This scheme represents a circular FIFO structure where the bit fields BOT and TOP establish the link from the last to the first element.

Bit field MOFGPRn.SEL of the base object can be used for monitoring purposes. It makes it possible to define a slave object within the list at which a message interrupt is

generated whenever the CUR pointer reaches the value of the SEL pointer. Thus SEL makes it possible to detect the end of a predefined message transfer series or to issue a warning interrupt when the FIFO becomes full.

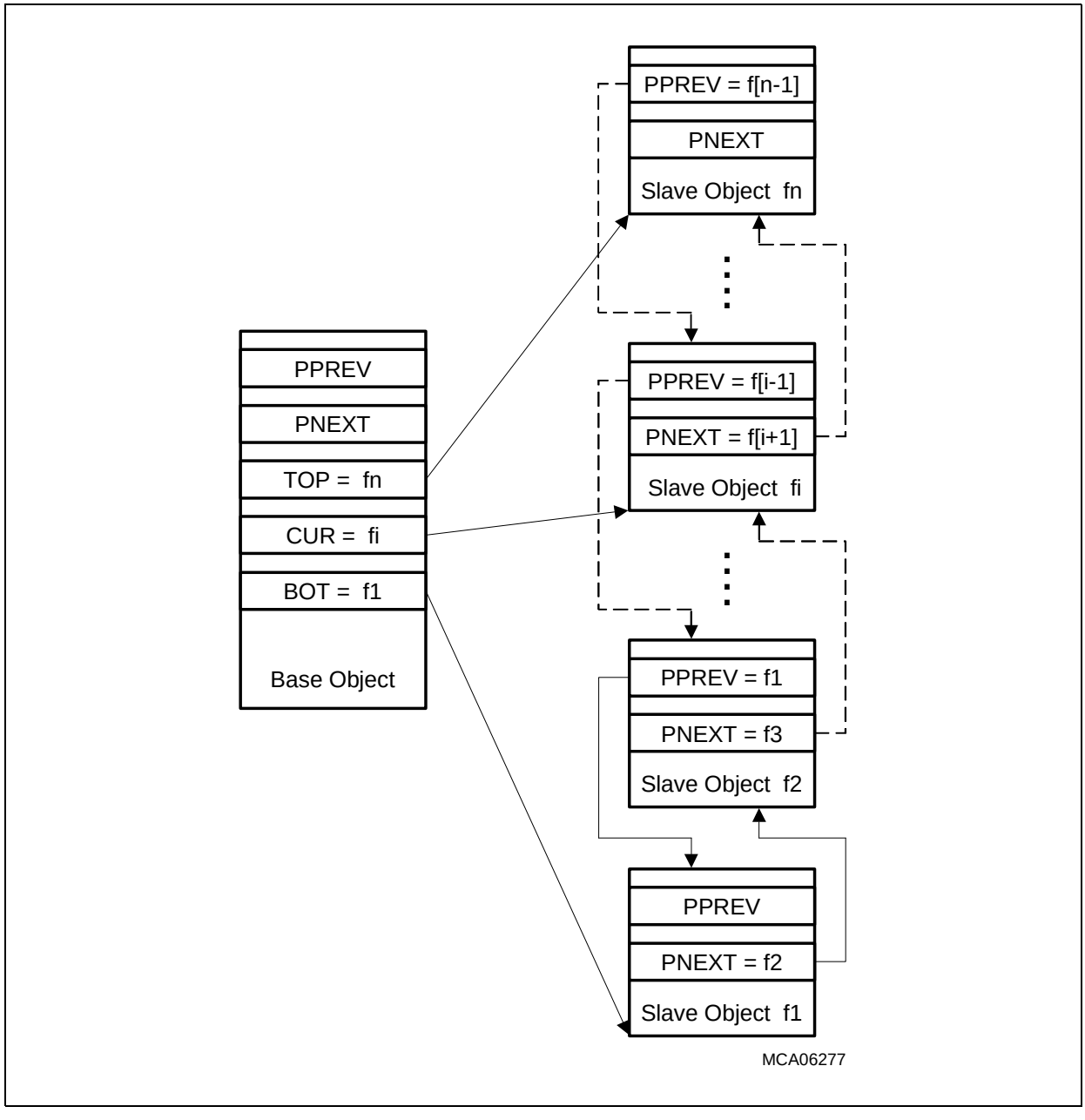


Figure 19-20 FIFO Structure with FIFO Base Object and n FIFO Slave Objects

19.3.11.5 Receive FIFO

The Receive FIFO structure is used to buffer incoming (received) Remote or Data Frames.

A Receive FIFO is selected by setting MOFCRn.MMC = 0001_B in the FIFO base object. This MMC code automatically designates a message object as FIFO base object. The message modes of the FIFO slave objects are not relevant for the operation of the Receive FIFO.

When the FIFO base object receives a frame from the CAN node it belongs to, the frame is not stored in the base object itself but in the message object that is selected by the base object's MOFGPRn.CUR pointer. This message object receives the CAN message as if it is the direct receiver of the message. However, MOFCRn.MMC = 0000_B is implicitly assumed for the FIFO slave object, and a standard message delivery is performed. The actual message mode (MMC setting) of the FIFO slave object is ignored. For the slave object, no acceptance filtering takes place that checks the received frame for a match with the identifier, IDE bit, and DIR bit.

With the reception of a CAN frame, the current pointer CUR of the base object is set to the number of the next message object in the FIFO structure. This message object will then be used to store the next incoming message.

If bit field MOFCRn.OVIE ("Overflow Interrupt Enable") of the FIFO base object is set and the current pointer MOFGPRn.CUR becomes equal to MOFGPRn.SEL, a FIFO overflow interrupt request is generated. This interrupt request is generated on interrupt node TXINP of the base object immediately after the storage of the received frame in the slave object. Transmit interrupts are still generated if TXIE is set.

A CAN message is stored in FIFO base and slave object only if MSGVAL = 1.

In order to avoid direct reception of a message by a slave message object, as if it was an independent message object and not a part of a FIFO, the bit RXEN of each slave object must be cleared. The setting of the bit RXEN is "don't care" only if the slave object is located in a list not assigned to a CAN node.

19.3.11.6 Transmit FIFO

The Transmit FIFO structure is used to buffer a series of Data or Remote Frames that must be transmitted. A transmit FIFO consists of one base message object and one or more slave message objects.

A Transmit FIFO is selected by setting MOFCRn.MMC = 0010_B in the FIFO base object. Unlike the Receive FIFO, slave objects assigned to the Transmit FIFO must explicitly set their bit fields MOFCRn.MMC = 0011_B. The CUR pointer in all slave objects must point back to the Transmit FIFO Base Object (to be initialized by software).

The MOSTATn.TXEN1 bits (Transmit Enable 1) of all message objects except the one which is selected by the CUR pointer of the base object must be cleared by software. TXEN1 of the message (slave) object selected by CUR must be set. CUR (of the base object) may be initialized to any FIFO slave object.

When tagging the message objects of the FIFO as valid to start the operation of the FIFO, then the base object must be tagged valid (MSGVAL = 1) first.

Before a Transmit FIFO becomes de-installed during operation, its slave objects must be tagged invalid (MSGVAL = 0).

The Transmit FIFO uses the TXEN1 bit in the Message Object Control Register of all FIFO elements to select the actual message for transmission. Transmit acceptance filtering evaluates TXEN1 for each message object and a message object can win transmit acceptance filtering only if its TXEN1 bit is set. When a FIFO object has transmitted a message, the hardware clears its TXEN1 bit in addition to standard transmit postprocessing (clear TXRQ, transmit interrupt etc.), and moves the CUR pointer in the next FIFO base object to be transmitted. TXEN1 is set automatically (by hardware) in the next message object. Thus, TXEN1 moves along the Transmit FIFO structure as a token that selects the active element.

If bit field MOFCRn.OVIE (“Overflow Interrupt Enable”) of the FIFO base object is set and the current pointer CUR becomes equal to MOFGPRn.SEL, a FIFO overflow interrupt request is generated. The interrupt request is generated on interrupt node RXINP of the base object after postprocessing of the received frame. Receive interrupts are still generated for the Transmit FIFO base object if bit RXIE is set.

19.3.11.7 Gateway Mode

The Gateway Mode makes it possible to establish an automatic information transfer between two independent CAN buses without CPU interaction.

The Gateway Mode operates on message object level. In Gateway mode, information is transferred between two message objects, resulting in an information transfer between the two CAN nodes to which the message objects are allocated. A gateway may be established with any pair of CAN nodes, and there can be as many gateways as there are message objects available to build the gateway structure.

Gateway Mode is selected by setting MOFCRs.MMC = 0100_B for the gateway source object s. The gateway destination object d is selected by the MOFGPRd.CUR pointer of the source object. The gateway destination object only needs to be valid (its MSGVAL = 1). All other settings are not relevant for the information transfer from the source object to the destination object.

Gateway source object behaves as a standard message object with the difference that some additional actions are performed by the MultiCAN module when a CAN frame has been received and stored in the source object (see **Figure 19-21**):

1. If bit MOFCRs.DLCC is set, the data length code MOFCRs.DLC is copied from the gateway source object to the gateway destination object.
2. If bit MOFCRs.IDC is set, the identifier MOARs.ID and the identifier extension MOARs.IDE are copied from the gateway source object to the gateway destination object.
3. If bit MOFCRs.DATC is set, the data bytes stored in the two data registers MODATALs and MODATAHs are copied from the gateway source object to the gateway destination object. All 8 data bytes are copied, even if MOFCRs.DLC indicates less than 8 data bytes.
4. If bit MOFCRs.GDFS is set, the transmit request flag MOSTATd.TXRQ is set in the gateway destination object.
5. The receive pending bit MOSTATd.RXPND and the new data bit MOSTATd.NEWDAT are set in the gateway destination object.
6. A message interrupt request is generated for the gateway destination object if its MOSTATd.RXIE is set.
7. The current object pointer MOFGPRs.CUR of the gateway source object is moved to the next destination object according to the FIFO rules as described on **Page 19-48**. A gateway with a single (static) destination object is obtained by setting MOFGPRs.TOP = MOFGPRs.BOT = MOFGPRs.CUR = destination object.

The link from the gateway source object to the gateway destination object works in the same way as the link from a FIFO base to a FIFO slave. This means that a gateway with an integrated destination FIFO may be created; in **Figure 19-20**, the object on the left is the gateway source object and the message object on the right side is the gateway destination objects.

The gateway operates equivalent for the reception of data frames (source object is receive object, i.e. DIR = 0) as well as for the reception of Remote Frames (source object is transmit object).

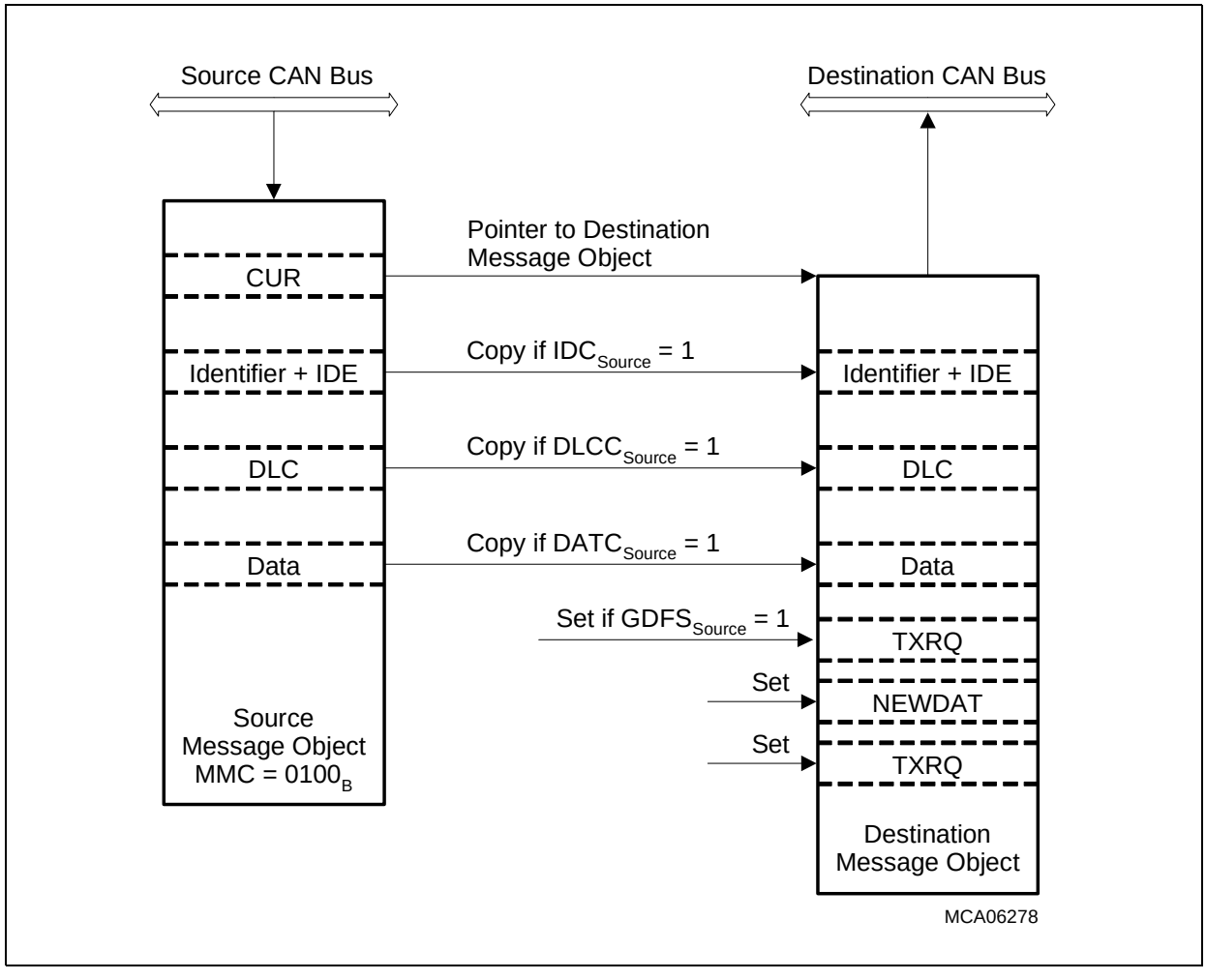


Figure 19-21 Gateway Transfer from Source to Destination

19.3.11.8 Foreign Remote Requests

When a Remote Frame has been received on a CAN node and is stored in a message object, a transmit request is set to trigger the answer (transmission of a Data Frame) to the request or to automatically issue a secondary request. If the Foreign Remote Request Enable bit MOFCRn.FRREN is cleared in the message object in which the remote request is stored, MOSTATn.TXRQ is set in the same message object.

If bit FRREN is set (FRREN = 1: foreign remote request enabled), TXRQ is set in the message object that is referenced by pointer MOFGPRn.CUR. The value of CUR is, however, not changed by this feature.

Although the foreign remote request feature works independently of the selected message mode, it is especially useful for gateways to issue a remote request on the source bus of a gateway after the reception of a remote request on the gateway destination bus. According to the setting of FRREN in the gateway destination object, there are two capabilities to handle remote requests that appear on the destination side (assuming that the source object is a receive object and the destination is a transmit object, i.e. DIR_{source} = 0 and $DIR_{destination}$ = 1):

FRREN = 0 in the Gateway Destination Object

1. A Remote Frame is received by gateway destination object.
2. TXRQ is set automatically in the gateway destination object.
3. A Data Frame with the current data stored in the destination object is transmitted on the destination bus.

FRREN = 1 in the Gateway Destination Object

1. A Remote Frame is received by gateway destination object.
2. TXRQ is set automatically in the gateway source object (must be referenced by CUR pointer of the destination object).
3. A remote request is transmitted by the source object (which is a receive object) on the source CAN bus.
4. The receiver of the remote request responds with a Data Frame on the source bus.
5. The Data Frame is stored in the source object.
6. The Data Frame is copied to the destination object (gateway action).
7. TXRQ is set in the destination object (assuming $GDFS_{source}$ = 1).
8. The new data stored in the destination object is transmitted on the destination bus, in response to the initial remote request on the destination bus.

19.4 MultiCAN Kernel Registers

This section describes the kernel registers of the MultiCAN module. All MultiCAN kernel register names described in this section are also referenced in other parts of the TC1797 User´s Manual by the module name prefix “CAN_”.

MultiCAN Kernel Register Overview

The MultiCAN Kernel include three blocks of registers:

- Global Module Registers
- Node Registers, for each CAN node x
- Message Object Registers, for each message object n

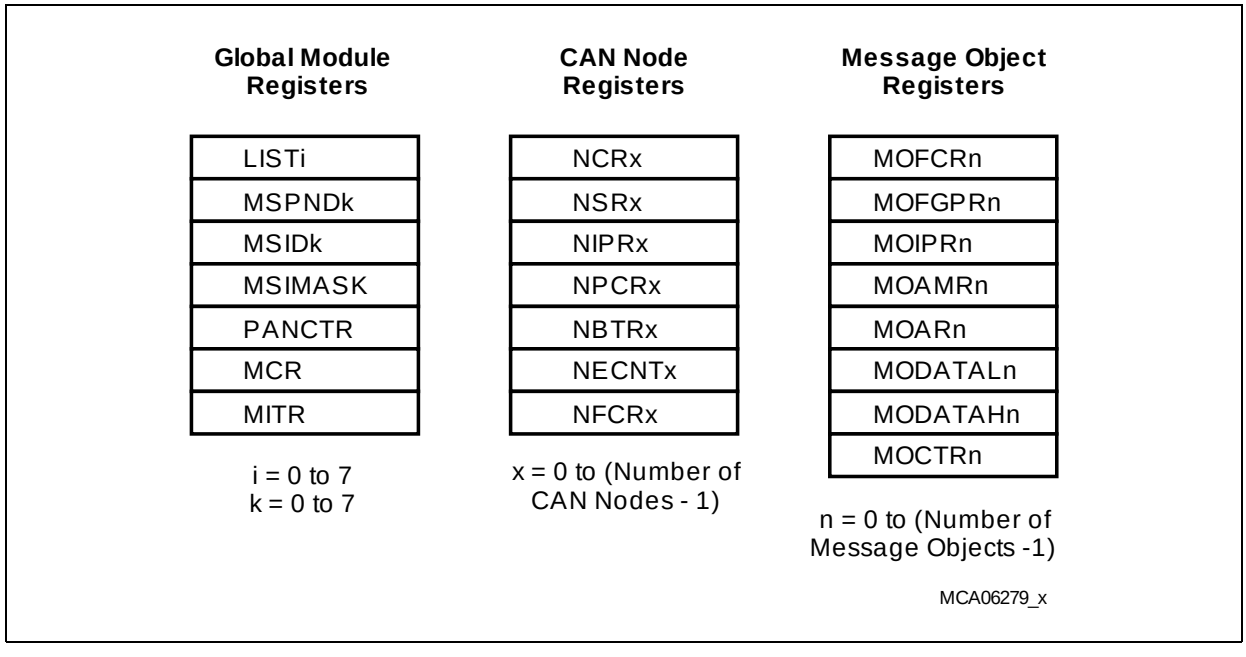


Figure 19-22 MultiCAN Kernel Registers

The registers of the MultiCAN module kernel are listed below.

Table 19-5 Registers Address Space - MultiCAN Kernel Registers

Module	Base Address	End Address	Note
CAN	$F000\ 4000_H$	$F000\ 7FFF_H$	-

Table 19-6 Registers Overview - MultiCAN Kernel Registers

Register Short Name	Register Long Name	Offset Address[1)]	Description see
LISTi	List Register i	$0100_H + i \times 4_H$	Page 19-65
MSPNDk	Message Pending Register k	$0140_H + k \times 4_H$	Page 19-67
MSIDk	Message Index Register k	$0180_H + k \times 4_H$	Page 19-68
MSIMASK	Message Index Mask Register	$01C0_H$	Page 19-69
ID	Module Identification Register	008_H	Page 19-58
PANCTR	Panel Control Register	$01C4_H$	Page 19-59
MCR	Module Control Register	$01C8_H$	Page 19-63
MITR	Module Interrupt Trigger Reg.	$01CC_H$	Page 19-64
NCRx	Node x Control Register	$0200_H + x \times 100_H$	Page 19-70
NSRx	Node x Status Register	$0204_H + x \times 100_H$	Page 19-74
NIPRx	Node x Interrupt Pointer Reg.	$0208_H + x \times 100_H$	Page 19-78
NPCRx	Node x Port Control Register	$020C_H + x \times 100_H$	Page 19-80
NBTRx	Node x Bit Timing Register	$0210_H + x \times 100_H$	Page 19-81
NECNTx	Node x Error Counter Register	$0214_H + x \times 100_H$	Page 19-83
NFCRx	Node x Frame Counter Register	$0218_H + x \times 100_H$	Page 19-84
MOFCRn	Message Object n Function Control Register	$1000_H + n \times 20_H$	Page 19-98
MOFGPRn	Message Object n FIFO/Gateway Pointer Register	$1004_H + n \times 20_H$	Page 19-102
MOIPRn	Message Object n Interrupt Pointer Register	$1008_H + n \times 20_H$	Page 19-96
MOAMRn	Message Object n Acceptance Mask Register	$100C_H + n \times 20_H$	Page 19-103
MODATALn	Message Object n Data Register Low	$1010_H + n \times 20_H$	Page 19-107
MODATAHn	Message Object n Data Register High	$1014_H + n \times 20_H$	Page 19-108
MOARn	Message Object n Arbitration Register	$1018_H + n \times 20_H$	Page 19-104
MOCTRn MOSTATn	Message Object n Control Reg. Message Object n Status Reg.	$101C_H + n \times 20_H$	Page 19-88 Page 19-91

1) The absolute register address is calculated as follows:
Module Base Address (**Table 19-5**) + Offset Address (shown in this column)
Further, the following ranges for parameters i, k, x, and n are valid: i = 0-7, k = 0-7, x = 0-3, n = 0-127.

Figure 19-23 shows the MultiCAN register address map.

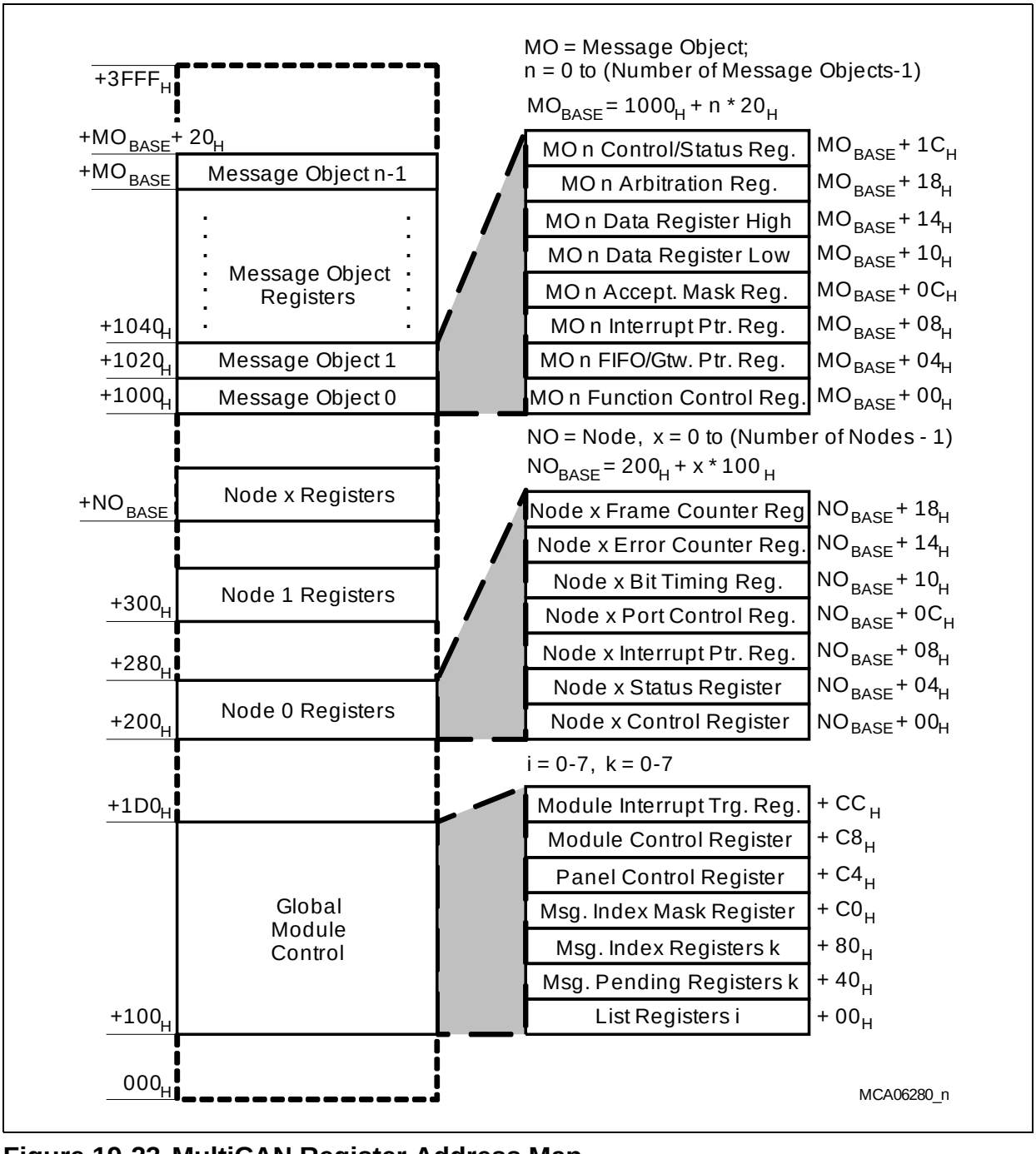


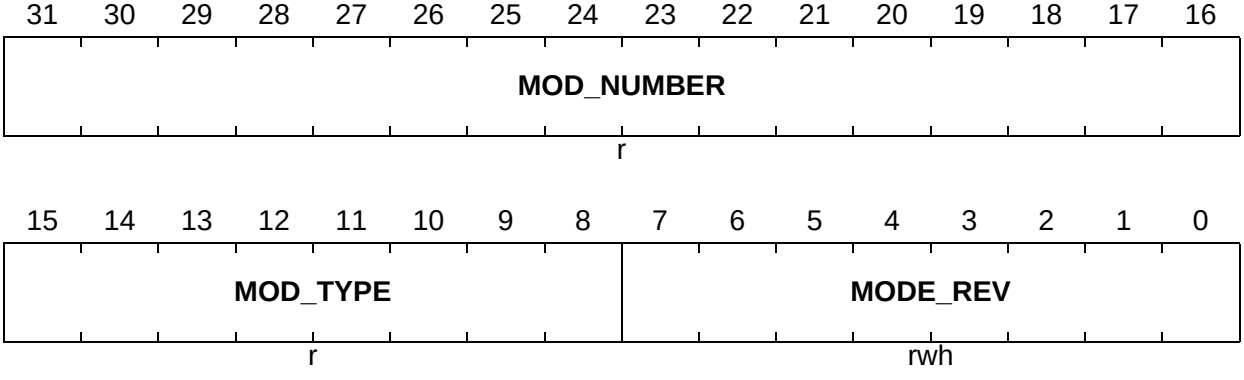
Figure 19-23 MultiCAN Register Address Map

19.4.1 Global Module Registers

All list operations such as allocation, de-allocation and relocation of message objects within the list structure are performed via the Command Panel. It is not possible to modify the list structure directly by software by writing to the message objects and the LIST registers.

Module Identification Register .

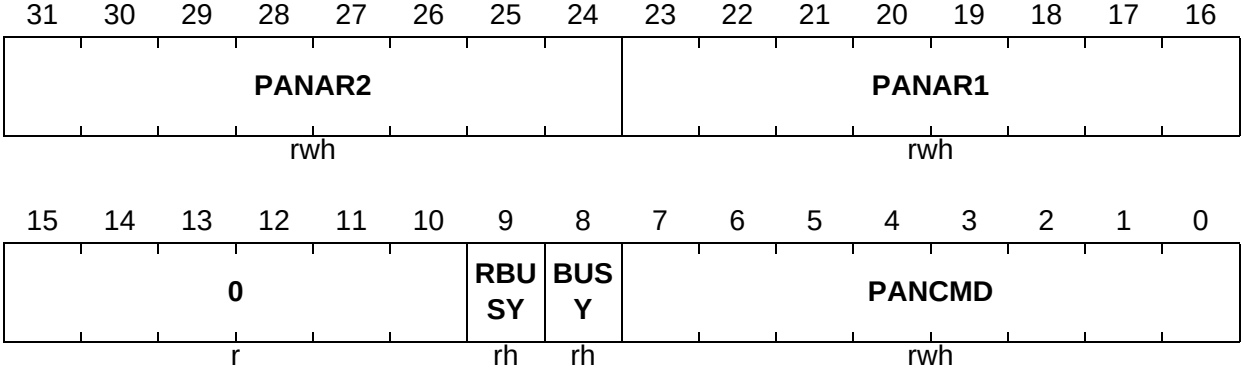
ID
Module Identification Register (008$_H$) Reset Value: 002B C0XX$_H$



Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision Number** MOD_REV defines the revision number. The value of a module revision starts with 01$_H$ (first revision).
MOD_TYPE	[15:8]	r	**Module Type** C0$_H$ Define the module as a 32-bit module.
MOD_NUMBER	[31:16]	r	**Module Number Value** This bit field defines the MultiCAN module identification number (=002BH)

The Panel Control Register PANCTR is used to start a new command by writing the command arguments and the command code into its bit fields.

PANCTR
Panel Control Register **($1C4_H$)** **Reset Value: $0000\ 0301_H$**



Field	Bits	Type	Description
PANCMD	[7:0]	rwh	**Panel Command** This bit field is used to start a new command by writing a panel command code into it. At the end of a panel command, the NOP (no operation) command code is automatically written into PANCMD. The coding of PANCMD is defined in **Table 19-7**.
BUSY	8	rh	**Panel Busy Flag** 0_B Panel has finished command and is ready to accept a new command. 1_B Panel operation is in progress.
RBUSY	9	rh	**Result Busy Flag** 0_B No update of PANAR1 and PANAR2 is scheduled by the list controller. 1_B A list command is running (BUSY = 1) that will write results to PANAR1 and PANAR2, but the results are not yet available.
PANAR1	[23:16]	rwh	**Panel Argument 1** See **Table 19-7**.
PANAR2	[31:24]	rwh	**Panel Argument 2** See **Table 19-7**.
0	[15:10]	r	**Reserved** Read as 0; should be written with 0.

Panel Commands

A panel operation consists of a command code (PANCMD) and up to two panel arguments (PANAR1, PANAR2). Commands that have a return value deliver it to the PANAR1 bit field. Commands that return an error flag deliver it to bit 31 of the Panel Control Register, this means bit 7 of PANAR2.

Table 19-7 Panel Commands

PANCMD	PANAR2	PANAR1	Command Description
00_H	–	–	**No Operation** Writing 00_H to PANCMD has no effect. No new command is started.
01_H	**Result:** Bit 7: ERR Bit 6-0: undefined	–	**Initialize Lists** Run the initialization sequence to reset the CTRL and LIST fields of all message objects. List registers LIST[7:0] are set to their reset values. This results in the de-allocation of all message objects. The initialization command requires that bits NCRx.INIT and NCRx.CCE are set for all CAN nodes. Bit 7 of PANAR2 (ERR) reports the success of the operation: 0_B Initialization was successful 1_B Not all NCRx.INIT and NCRx.CCE bits are set. Therefore, no initialization is performed. The initialize lists command is automatically performed with each reset of the MultiCAN module, but with the exception that all message object registers are reset, too.
02_H	**Argument:** List Index	**Argument:** Message Object Number	**Static Allocate** Allocate message object to a list. The message object is removed from the list that it currently belongs to, and appended to the end of the list, given by PANAR2. This command is also used to deallocate a message object. In this case, the target list is the list of unallocated elements (PANAR2 = 0).

Table 19-7 Panel Commands (cont'd)

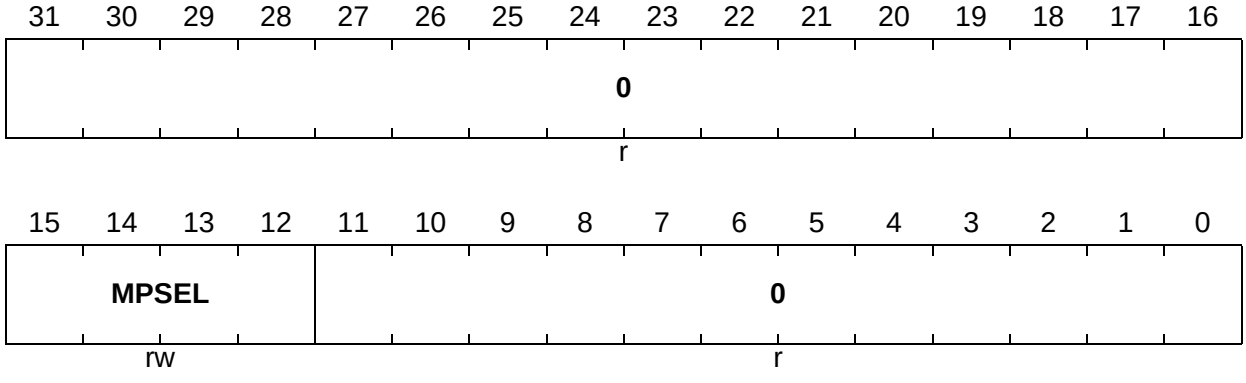
PANCMD	PANAR2	PANAR1	Command Description
03_H	**Argument:** List Index **Result:** Bit 7: ERR Bit 6-0: undefined	**Result:** Message Object Number	**Dynamic Allocate** Allocate the first message object of the list of unallocated objects to the selected list. The message object is appended to the end of the list. The message number of the message object is returned in PANAR1. An ERR bit (bit 7 of PANAR2) reports the success of the operation: 0_B Success. 1_B The operation has not been performed because the list of unallocated elements was empty.
04_H	**Argument:** Destination Object Number	**Argument:** Source Object Number	**Static Insert Before** Remove a message object (source object) from the list that it currently belongs to, and insert it before a given destination object into the list structure of the destination object. The source object thus becomes the predecessor of the destination object.
05_H	**Argument:** Destination Object Number **Result:** Bit 7: ERR Bit 6-0: undefined	**Result:** Object Number of inserted object	**Dynamic Insert Before** Insert a new message object before a given destination object. The new object is taken from the list of unallocated elements (the first element is chosen). The number of the new object is delivered as a result to PANAR1. An ERR bit (bit 7 of PANAR2) reports the success of the operation: 0_B Success. 1_B The operation has not been performed because the list of unallocated elements was empty.

Table 19-7 Panel Commands (cont'd)

PANCMD	PANAR2	PANAR1	Command Description
06_H	**Argument:** Destination Object Number	**Argument:** Source Object Number	**Static Insert Behind** Remove a message object (source object) from the list that it currently belongs to, and insert it behind a given destination object into the list structure of the destination object. The source object thus becomes the successor of the destination object.
07_H	**Argument:** Destination Object Number **Result:** Bit 7: ERR Bit 6-0: undefined	**Result:** Object Number of inserted object	**Dynamic Insert Behind** Insert a new message object behind a given destination object. The new object is taken from the list of unallocated elements (the first element is chosen). The number of the new object is delivered as result to PANAR1. An ERR bit (bit 7 of PANAR2) reports the success of the operation: 0_B Success. 1_B The operation has not been performed because the list of unallocated elements was empty.
08_H - FF_H	–	–	**Reserved**

The Module Control Register MCR contains basic settings that determine the operation of the MultiCAN module.

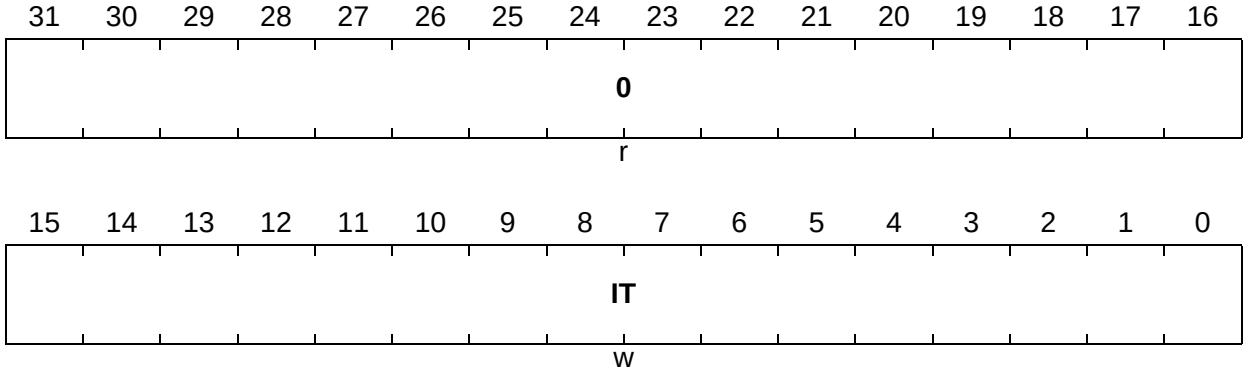
MCR
Module Control Register **($1C8_H$)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
MPSEL	[15:12]	rw	**Message Pending Selector** Bit field MPSEL makes it possible to select the bit position of the message pending bit after a message reception/transmission by a mixture of the MOIPRn register bit fields RXINP, TXINP, and MPN. Selection details are given in **Figure 19-17** on **Page 19-39**.
0	[31:16], [11:0]	r	**Reserved** Read as 0; should be written with 0.

The Interrupt Trigger Register ITR is used to trigger interrupt requests on each interrupt output line by software.

MITR
Module Interrupt Trigger Register (1CC$_H$) Reset Value: 0000 0000$_H$



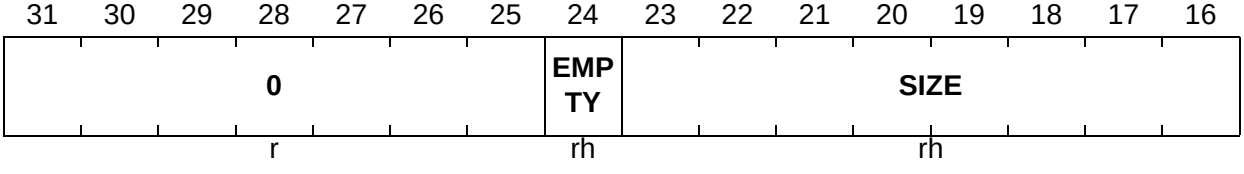
Field	Bits	Type	Description
IT	[15:0]	w	**Interrupt Trigger** Writing a 1 to IT[n] (n = 0-15) generates an interrupt request on interrupt output line INT_O[n]. Writing a 0 to IT[n] has no effect. Bit field IT is always read as 0. Multiple interrupt requests can be generated with a single write operation to MITR by writing a 1 to several bit positions of IT.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

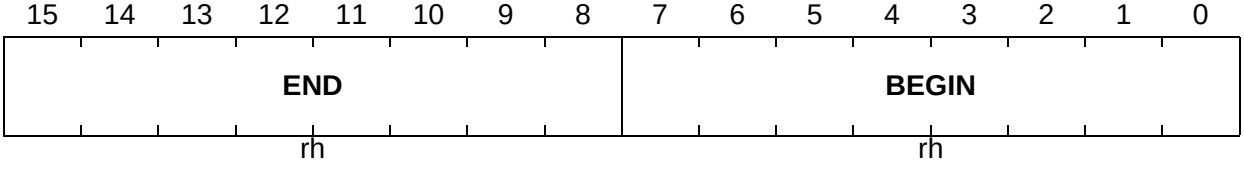
List Pointer and List Register

Each of the four CAN nodes has a list that determines the allocated message objects. Additionally, a list of all unallocated objects is available. Furthermore, general purpose lists are available which are not associated to a CAN node. The List Registers are assigned in the following way:

- LIST0 provides the list of all unallocated objects
- LIST1 provides the list for CAN node 0
- LIST2 provides the list for CAN node 1
- LIST3 provides the list for CAN node 2
- LIST4 provides the list for CAN node 3
- LIST[7:5] are not associated to a CAN node (free lists)

LIST0
List Register 0 **(100_H)** **Reset Value: $007F\ 7F00_H$**
LISTx (x = 1-7)
List Register x **(100_H+x*4_H)** **Reset Value: $0100\ 0000_H$**





Field	Bits	Type	Description
BEGIN	[7:0]	rh	**List Begin** BEGIN indicates the number of the first message object in list i.
END	[15:8]	rh	**List End** END indicates the number of the last message object in list i.
SIZE	[23:16]	rh	**List Size** SIZE indicates the number of elements in the list i. SIZE = number of list elements - 1 SIZE = 0 indicates that list i is empty.

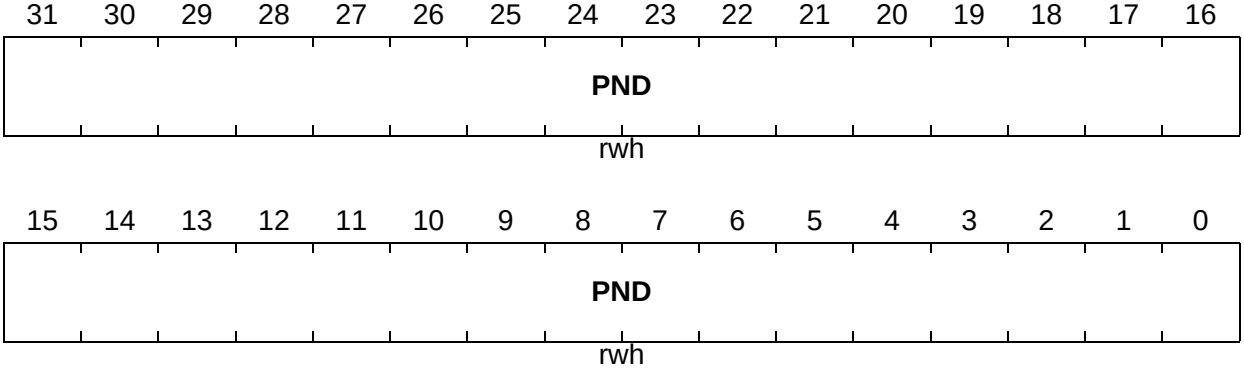
Field	Bits	Type	Description
EMPTY	24	rh	**List Empty Indication** 0_B At least one message object is allocated to list i. 1_B No message object is allocated to the list i. List i is empty.
0	[31:25]	r	**Reserved** Read as 0.

Message Notifications

When a message object n generates an interrupt request upon the transmission or reception of a message, then the request is routed to the interrupt output line selected by the bit field MOIPRn.TXINP or MOIPRn.RXINP of the message object n. As there are more message objects than interrupt output lines, an interrupt routine typically processes requests from more than one message object. Therefore, a priority selection mechanism is implemented in the MultiCAN module to select the highest priority object within a collection of message objects.

The Message Pending Register MSPNDk contains the pending interrupt notification of list i.

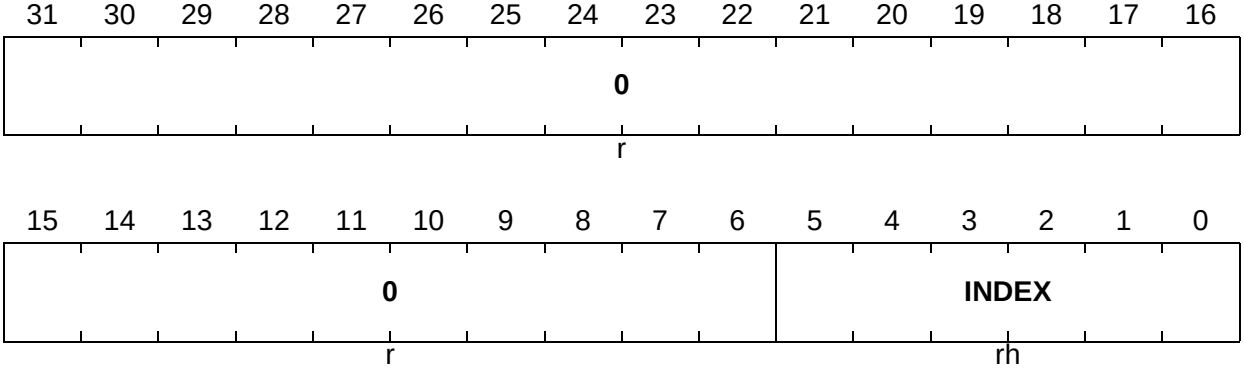
MSPNDk (k = 0-7)
Message Pending Register k **(140_H+k*4_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
PND	[31:0]	rwh	**Message Pending** When a message interrupt occurs, the message object sets a bit in one of the MSPND register, where the bit position is given by the MPN[4:0] field of the IPR register of the message object. The register selection n is given by the higher bits of MPN. The register bits can be cleared by software (write 0). Writing a 1 has no effect.

Each Message Pending Register has a Message Index Register MSIDk associated with it. The Message Index Register shows the active (set) pending bit with lowest bit position within groups of pending bits.

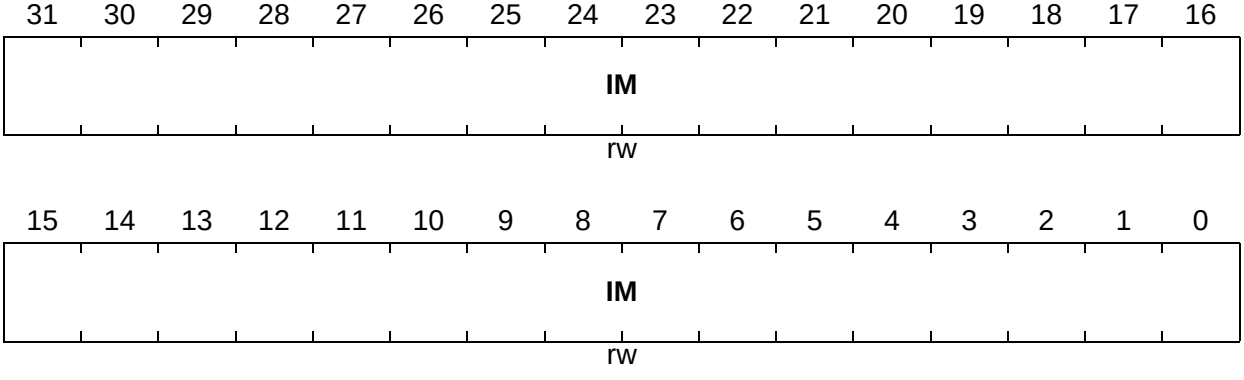
MSIDk (k = 0-7)
Message Index Register k **(180_H+k*4_H)** **Reset Value: 0000 0020_H**



Field	Bits	Type	Description
INDEX	[5:0]	rh	**Message Pending Index** The value of INDEX is given by the bit position i of the pending bit of MSPNDk with the following properties: 1. MSPNDk[i] & IM[i] = 1 2. i = 0 or MSPNDk[i-1:0] & IM[i-1:0] = 0 If no bit of MSPNDk satisfies these conditions then INDEX reads 100000_B. Thus INDEX shows the position of the first pending bit of MSPNDk, in which only those bits of MSPNDk that are selected in the Message Index Mask Register are taken into account.
0	[31:6]	r	**Reserved** Read as 0; should be written with 0.

The Message Index Mask Register MSIMASK selects individual bits for the calculation of the Message Pending Index. The Message Index Mask Register is used commonly for all Message Pending registers and their associated Message Index registers.

MSIMASK
Message Index Mask Register ($1C0_H$) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
IM	[31:0]	rw	**Message Index Mask** Only those bits in MSPNDk for which the corresponding Index Mask bits are set contribute to the calculation of the Message Index.

19.4.2 CAN Node Registers

The CAN node registers are built in for each CAN node of the MultiCAN module. They contain information that is directly related to the operation of the CAN nodes and are shared among the nodes.

The Node Control Register contains basic settings that determine the operation of the CAN node.

NCRx (x = 0-3)
Node x Control Register **(200_H+x*100_H)** **Reset Value: 0000 0001_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							SUSEN	CALM	CCE	0	CANDIS	ALIE	LECIE	TRIE	INIT
r							rw	rw	rw	r	rw	rw	rw	rw	rwh

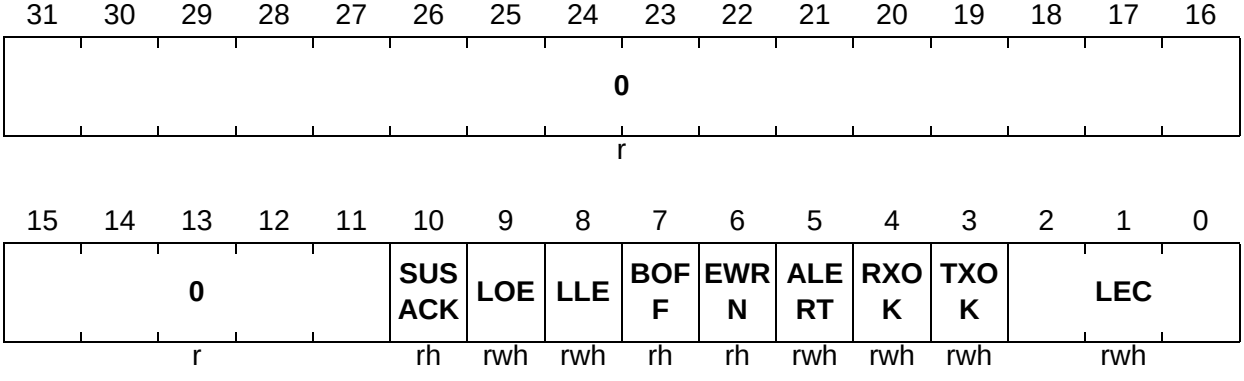
Field	Bits	Type	Description
INIT	0	rwh	**Node Initialization** 0_B Resetting bit INIT enables the participation of the node in the CAN traffic. If the CAN node is in the bus-off state, the ongoing bus-off recovery (which does not depend on the INIT bit) is continued. With the end of the bus-off recovery sequence the CAN node is allowed to take part in the CAN traffic. If the CAN node is not in the bus-off state, a sequence of 11 consecutive recessive bits must be detected before the node is allowed to take part in the CAN traffic. 1_B Setting this bit terminates the participation of this node in the CAN traffic. Any ongoing frame transfer is cancelled and the transmit line goes recessive. If the CAN node is in the bus-off state, then the running bus-off recovery sequence is continued. If the INIT bit is still set after the successful completion of the bus-off recovery sequence, i.e. after detecting 128 sequences of 11 consecutive recessive bits (11×1), then the CAN node leaves the bus-off state but remains inactive as long as INIT remains set. Bit INIT is automatically set when the CAN node enters the bus-off state (see **Page 19-23**).
TRIE	1	rw	**Transfer Interrupt Enable** TRIE enables the transfer interrupt of CAN node x. This interrupt is generated after the successful reception or transmission of a CAN frame in node x. 0_B Transfer interrupt is disabled. 1_B Transfer interrupt is enabled. Bit field NIPRx.TRINP selects the interrupt output line which becomes activated at this type of interrupt.

Field	Bits	Type	Description
LECIE	2	rw	**LEC Indicated Error Interrupt Enable** LECIE enables the last error code interrupt of CAN node x. This interrupt is generated with each update of bit field NSRx.LEC with LEC > 0 (CAN protocol error). 0_B Last error code interrupt is disabled. 1_B Last error code interrupt is enabled. Bit field NIPRx.LECINP selects the interrupt output line which becomes activated at this type of interrupt.
ALIE	3	rw	**Alert Interrupt Enable** ALIE enables the alert interrupt of CAN node x. This interrupt is generated by any one of the following events: • A change of bit NSRx.BOFF • A change of bit NSRx.EWRN • A List Length Error, which also sets bit NSRx.LLE • A List Object Error, which also sets bit NSRx.LOE • A Bit INIT is set by hardware 0_B Alert interrupt is disabled. 1_B Alert interrupt is enabled. Bit field NIPRx.ALINP selects the interrupt output line which becomes activated at this type of interrupt.
CANDIS	4	rw	**CAN Disable** Setting this bit disables the CAN node. The CAN node first waits until it is bus-idle or bus-off. Then bit INIT is automatically set, and an alert interrupt is generated if bit ALIE is set.
CCE	6	rw	**Configuration Change Enable** 0_B The Bit Timing Register, the Port Control Register, and the Error Counter Register may only be read. All attempts to modify them are ignored. 1_B The Bit Timing Register, the Port Control Register, and the Error Counter Register may be read and written.
CALM	7	rw	**CAN Analyze Mode** If this bit is set, then the CAN node operates in Analyze Mode. This means that messages may be received, but not transmitted. No acknowledge is sent on the CAN bus upon frame reception. Active-error flags are sent recessive instead of dominant. The transmit line is continuously held at recessive (1) level. Bit CALM can be written only while bit INIT is set.

Field	Bits	Type	Description
SUSEN	8	rw	**Suspend Enable** This bit makes it possible to set the CAN node into Suspend Mode via OCDS (on chip debug support): 0_B An OCDS suspend trigger is ignored by the CAN node. 1_B An OCDS suspend trigger disables the CAN node: As soon as the CAN node becomes bus-idle or bus-off, bit INIT is internally forced to 1 to disable the CAN node. The actual value of bit INIT remains unchanged. Bit SUSEN is reset via OCDS Reset.
0	[31:9], 5	r	**Reserved** Read as 0; should be written with 0.

The Node Status Register NSRx reports errors as well as successfully transferred CAN frames.

NSRx (x = 0-3)
Node x Status Register **(204_H+x*100_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
LEC	[2:0]	rwh	**Last Error Code** This bit field indicates the type of the last (most recent) CAN error. The encoding of this bit field is described in **Table 19-8**.
TXOK	3	rwh	**Message Transmitted Successfully** 0_B No successful transmission since last (most recent) flag reset. 1_B A message has been transmitted successfully (error-free and acknowledged by at least another node). TXOK must be reset by software (write 0). Writing 1 has no effect.
RXOK	4	rwh	**Message Received Successfully** 0_B No successful reception since last (most recent) flag reset. 1_B A message has been received successfully. RXOK must be reset by software (write 0). Writing 1 has no effect.

Field	Bits	Type	Description
ALERT	5	rwh	**Alert Warning** The ALERT bit is set upon the occurrence of one of the following events (the same events which also trigger an alert interrupt if ALIE is set): • A change of bit NSRx.BOFF • A change of bit NSRx.EWRN • A List Length Error, which also sets bit NSRx.LLE • A List Object Error, which also sets bit NSRx.LOE • Bit INIT has been set by hardware ALERT must be reset by software (write 0). Writing 1 has no effect.
EWRN	6	rh	**Error Warning Status** 0_B No warning limit exceeded. 1_B One of the error counters REC or TEC reached the warning limit EWRNLVL.
BOFF	7	rh	**Bus-off Status** 0_B CAN controller is not in the bus-off state. 1_B CAN controller is in the bus-off state.
LLE	8	rwh	**List Length Error** 0_B No List Length Error since last (most recent) flag reset. 1_B A List Length Error has been detected during message acceptance filtering. The number of elements in the list that belongs to this CAN node differs from the list SIZE given in the list termination pointer. LLE must be reset by software (write 0). Writing 1 has no effect.
LOE	9	rwh	**List Object Error** 0_B No List Object Error since last (most recent) flag reset. 1_B A List Object Error has been detected during message acceptance filtering. A message object with wrong LIST index entry in the Message Object Control Register has been detected. LOE must be reset by software (write 0). Writing 1 has no effect.

Field	Bits	Type	Description
SUSACK	10	rh	**Suspend Acknowledge** 0_B The CAN node is not in Suspend Mode or a suspend request is pending, but the CAN node has not yet reached bus-idle or bus-off. 1_B The CAN node is in Suspend Mode: The CAN node is inactive (bit NCR.INIT internally forced to 1) due to an OCDS suspend request.
0	[31:11]	r	**Reserved** Read as 0; should be written with 0.

Encoding of the LEC Bit Field

Table 19-8 Encoding of the LEC Bit Field

LEC Value	Signification
000_B	**No Error:** No error was detected for the last (most recent) message on the CAN bus.
001_B	**Stuff Error:** More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed.
010_B	**Form Error:** A fixed format part of a received frame has the wrong format.
011_B	**Ack Error:** The transmitted message was not acknowledged by another node.
100_B	**Bit1 Error:** During a message transmission, the CAN node tried to send a recessive level (1) outside the arbitration field and the acknowledge slot, but the monitored bus value was dominant.
101_B	**Bit0 Error:** Two different conditions are signaled by this code: 1. During transmission of a message (or acknowledge bit, active-error flag, overload flag), the CAN node tried to send a dominant level (0), but the monitored bus value was recessive. 2. During bus-off recovery, this code is set each time a sequence of 11 recessive bits has been monitored. The CPU may use this code as indication that the bus is not continuously disturbed.

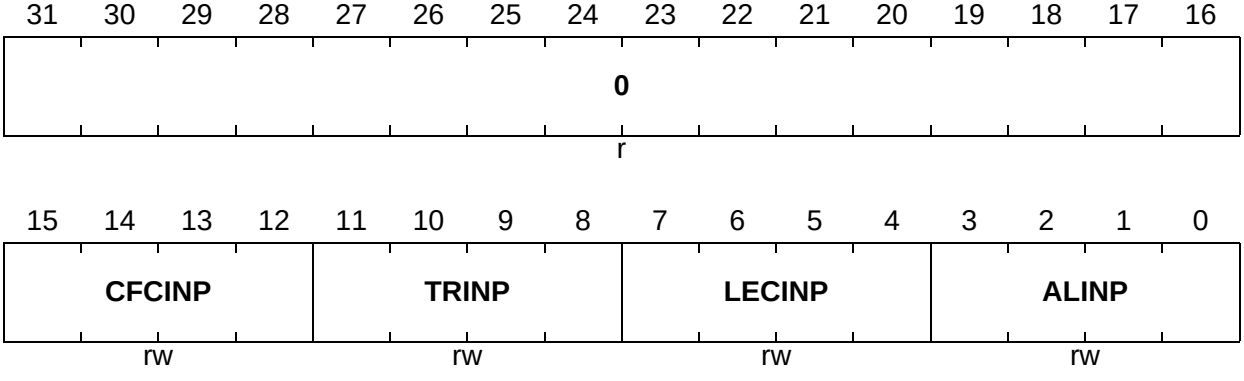
Table 19-8 Encoding of the LEC Bit Field (cont'd)

LEC Value	Signification
110_B	**CRC Error:** The CRC checksum of the received message was incorrect.
111_B	**CPU write to LEC:** Whenever the the CPU writes the value 111 to LEC, it takes the value 111. Whenever the CPU writes another value to LEC, the written LEC value is ignored.

The four interrupt pointers in the Node Interrupt Pointer Register NIPRx select one out of the sixteen interrupt outputs individually for each type of CAN node interrupt. See also **Page 19-24** for more CAN node interrupt details.

NIPRx (x = 0-3)
Node x Interrupt Pointer Register

(208_H+x*100_H) **Reset Value: 0000 0000_H**

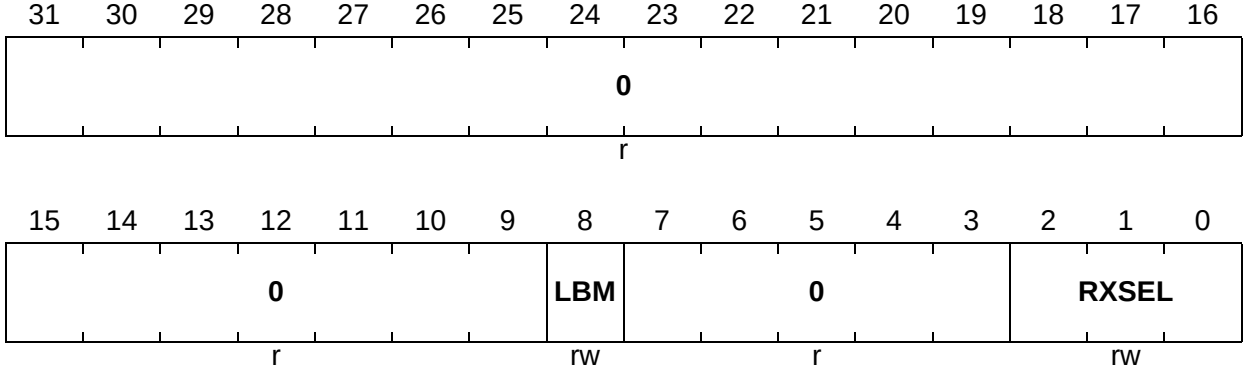


Field	Bits	Type	Description
ALINP	[3:0]	rw	**Alert Interrupt Node Pointer** ALINP selects the interrupt output line INT_Om (m = 0-15) for an alert interrupt of CAN Node x. 0000_B Interrupt output line INT_O0 is selected. 0001_B Interrupt output line INT_O1 is selected. $\cdots_B$ … 1110_B Interrupt output line INT_O14 is selected. 1111_B Interrupt output line INT_O15 is selected.
LECINP	[7:4]	rw	**Last Error Code Interrupt Node Pointer** LECINP selects the interrupt output line INT_Om (m = 0-15) for an LEC interrupt of CAN Node x. 0000_B Interrupt output line INT_O0 is selected. 0001_B Interrupt output line INT_O1 is selected. $\cdots_B$ … 1110_B Interrupt output line INT_O14 is selected. 1111_B Interrupt output line INT_O15 is selected.

Field	Bits	Type	Description
TRINP	[11:8]	rw	**Transfer OK Interrupt Node Pointer** TRINP selects the interrupt output line INT_Om (m = 0-15) for a transfer OK interrupt of CAN Node x. 0000_B Interrupt output line INT_O0 is selected. 0001_B Interrupt output line INT_O1 is selected. $\ldots_B$ … 1110_B Interrupt output line INT_O14 is selected. 1111_B Interrupt output line INT_O15 is selected.
CFCINP	[15:12]	rw	**Frame Counter Interrupt Node Pointer** CFCINP selects the interrupt output line INT_Om (m = 0-15) for a frame counter overflow interrupt of CAN Node x. 0000_B Interrupt output line INT_O0 is selected. 0001_B Interrupt output line INT_O1 is selected. $\ldots_B$ … 1110_B Interrupt output line INT_O14 is selected. 1111_B Interrupt output line INT_O15 is selected.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

The Node Port Control Register NPCRx configures the CAN bus transmit/receive ports. NPCRx can be written only if bit NCRx.CCE is set.

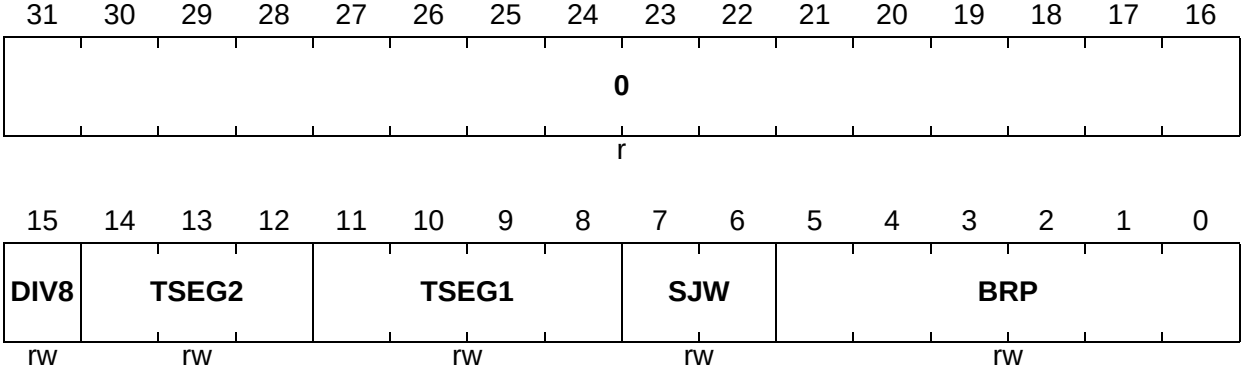
NPCRx (x = 0-3)
Node x Port Control Register **($20C_H+x*100_H$)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
RXSEL	[2:0]	rw	**Receive Select** RXSEL selects one out of 8 possible receive inputs. The CAN receive signal is performed only through the selected input. *Note: In TC1797, only specific combinations of RXSEL are available (see also **"Node Receive Input Selection" on Page 19-115**).*
LBM	8	rw	**Loop-Back Mode** 0_B Loop-Back Mode is disabled. 1_B Loop-Back Mode is enabled. This node is connected to an internal (virtual) loop-back CAN bus. All CAN nodes which are in Loop-Back Mode are connected to this virtual CAN bus so that they can communicate with each other internally. The external transmit line is forced recessive in Loop-Back Mode.
0	[7:3], [31:9]	r	**Reserved** Read as 0; should be written with 0.

The Node Bit Timing Register NBTRx contains all parameters to set up the bit timing for the CAN transfer. NBTRx can be written only if bit NCRx.CCE is set.

NBTRx (x = 0-3)
Node x Bit Timing Register **(210_H+x*100_H)** **Reset Value: 0000 0000_H**

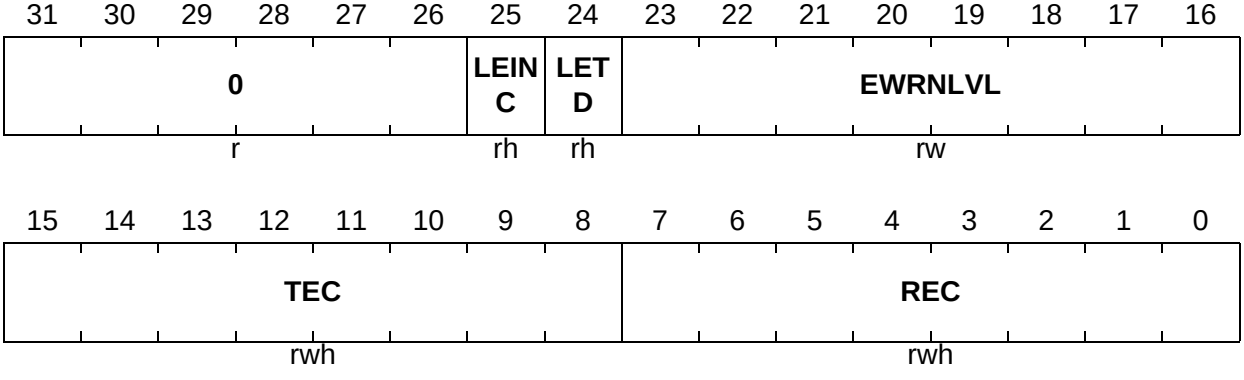


Field	Bits	Type	Description
BRP	[5:0]	rw	**Baud Rate Prescaler** The duration of one time quantum is given by (BRP + 1) clock cycles if DIV8 = 0. The duration of one time quantum is given by 8 × (BRP + 1) clock cycles if DIV8 = 1.
SJW	[7:6]	rw	**(Re) Synchronization Jump Width** (SJW + 1) time quanta are allowed for re-synchronization.
TSEG1	[11:8]	rw	**Time Segment Before Sample Point** (TSEG1 + 1) time quanta is the user-defined nominal time between the end of the synchronization segment and the sample point. It includes the propagation segment, which takes into account signal propagation delays. The time segment may be lengthened due to re-synchronization. Valid values for TSEG1 are 2 to 15.
TSEG2	[14:12]	rw	**Time Segment After Sample Point** (TSEG2 + 1) time quanta is the user-defined nominal time between the sample point and the start of the next synchronization segment. It may be shortened due to re-synchronization. Valid values for TSEG2 are 1 to 7.

Field	Bits	Type	Description
DIV8	15	rw	**Divide Prescaler Clock by 8** 0_B A time quantum lasts (BRP+1) clock cycles. 1_B A time quantum lasts 8 × (BRP+1) clock cycles.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

The Node Error Counter Register NECNTx contains the CAN receive and transmit error counter as well as some additional bits to ease error analysis. NECNTx can be written only if bit NCRx.CCE is set.

NECNTx (x = 0-3)
Node x Error Counter Register (214_H+x*100_H) Reset Value: 0060 0000_H

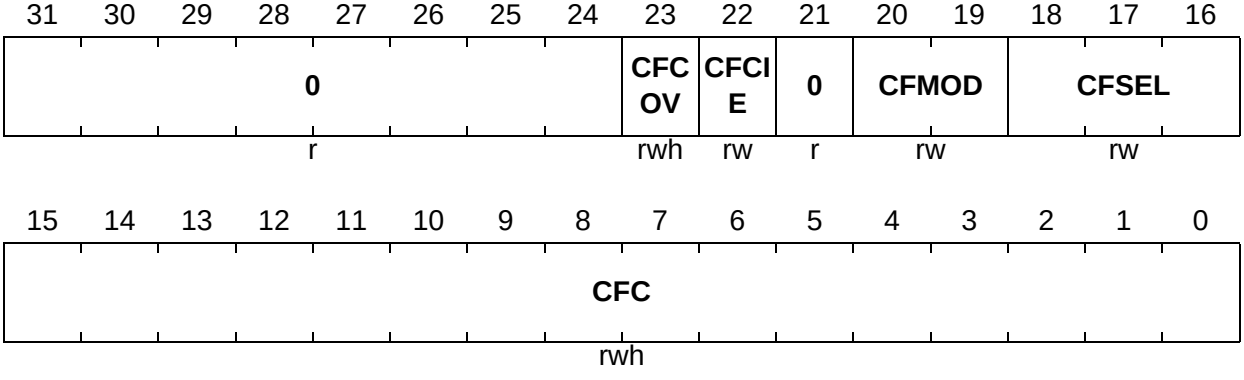


Field	Bits	Type	Description
REC	[7:0]	rwh	**Receive Error Counter** Bit field REC contains the value of the receive error counter of CAN node x.
TEC	[15:8]	rwh	**Transmit Error Counter** Bit field TEC contains the value of the transmit error counter of CAN node x.
EWRNLVL	[23:16]	rw	**Error Warning Level** Bit field EWRNLVL determines the threshold value (warning level, default 96) to be reached in order to set the corresponding error warning bit EWRN.
LETD	24	rh	**Last Error Transfer Direction** 0_B The last error occurred while the CAN node x was receiver (REC has been incremented). 1_B The last error occurred while the CAN node x was transmitter (TEC has been incremented).
LEINC	25	rh	**Last Error Increment** 0_B The last error led to an error counter increment of 1. 1_B The last error led to an error counter increment of 8.

Field	Bits	Type	Description
0	[31:26]	r	**Reserved** Read as 0; should be written with 0.

The Node Frame Counter Register NFCRx contains the actual value of the frame counter as well as control and status bits of the frame counter.

NFCRx (x = 0-3)
Node x Frame Counter Register (218_H+x*100_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
CFC	[15:0]	rwh	**CAN Frame Counter** In Frame Count Mode (CFMOD = 00_B), this bit field contains the frame count value. In Time Stamp Mode (CFMOD = 01_B), this bit field contains the captured bit time count value, captured with the start of a new frame. In all Bit Timing Analysis Modes (CFMOD = 10_B), CFC always displays the number of f_{CLC} clock cycles (measurement result) minus 1. Example: a CFC value of 34 in measurement mode CFSEL = 000_B means that 35 f_{CLC} clock cycles have been elapsed between the most recent two dominant edges on the receive input.

Field	Bits	Type	Description
CFSEL	[18:16]	rw	**CAN Frame Count Selection** This bit field selects the function of the frame counter for the chosen frame count mode. **Frame Count Mode** Bit 0 If Bit 0 of CFSEL is set, then CFC is incremented each time a foreign frame (i.e. a frame not matching to a message object) has been received on the CAN bus. Bit 1 If Bit 1 of CFSEL is set, then CFC is incremented each time a frame matching to a message object has been received on the CAN bus. Bit 2 If Bit 2 of CFSEL is set, then CFC is incremented each time a frame has been transmitted successfully by the node. **Time Stamp Mode** 000_B The frame counter is incremented (internally) at the beginning of a new bit time. The value is sampled during the SOF bit of a new frame. The sampled value is visible in the CFC field. **Bit Timing Mode** The available bit timing measurement modes are shown in **Table 19-9**. If CFCIE is set, then an interrupt on request node x (where x is the CAN node number) is generated with a CFC update.
CFMOD	[20:19]	rw	**CAN Frame Counter Mode** This bit field determines the operation mode of the frame counter. 00_B Frame Count Mode: The frame counter is incremented upon the reception and transmission of frames. 10_B 0Time Stamp Mode: The frame counter is used to count bit times. 10_B Bit Timing Mode: The frame counter is used for analysis of the bit timing.

Field	Bits	Type	Description
CFCIE	22	rw	**CAN Frame Count Interrupt Enable** CFCIE enables the CAN frame counter overflow interrupt of CAN node x. 0_B CAN frame counter overflow interrupt is disabled. 1_B CAN frame counter overflow interrupt is enabled. Bit field NIPRx.CFCINP selects the interrupt output line that is activated at this type of interrupt.
CFCOV	23	rwh	**CAN Frame Counter Overflow Flag** Flag CFCOV is set upon a frame counter overflow (transition from $FFFF_H$ to 0000_H). In bit timing analysis mode, CFCOV is set upon an update of CFC. An interrupt request is generated if CFCIE = 1. 0_B No overflow has occurred since last flag reset. 1_B An overflow has occurred since last flag reset. CFCOV must be reset by software.
0	21, [31:24]	r	**Reserved** Read as 0; should be written with 0.

Bit Timing Analysis Modes

Table 19-9 Bit Timing Analysis Modes (CFMOD = 10)

CFSEL	Measurement
000_B	Whenever a dominant edge (transition from 1 to 0) is monitored on the receive input, the time (measured in clock cycles) between this edge and the most recent dominant edge is stored in CFC.
001_B	Whenever a recessive edge (transition from 0 to 1) is monitored on the receive input the time (measured in clock cycles) between this edge and the most recent dominant edge is stored in CFC.
010_B	Whenever a dominant edge is received as a result of a transmitted dominant edge, the time (clock cycles) between both edges is stored in CFC.
011_B	Whenever a recessive edge is received as a result of a transmitted recessive edge, the time (clock cycles) between both edges is stored in CFC.
100_B	Whenever a dominant edge that qualifies for synchronization is monitored on the receive input, the time (measured in clock cycles) between this edge and the most recent sample point is stored in CFC.

Table 19-9 Bit Timing Analysis Modes (CFMOD = 10) (cont'd)

CFSEL	Measurement
101_B	With each sample point, the time (measured in clock cycles) between the start of the new bit time and the start of the previous bit time is stored in CFC[11:0]. Additional information is written to CFC[15:12] at each sample point: CFC[15]: Transmit value of actual bit time CFC[14]: Receive sample value of actual bit time CFC[13:12]: CAN bus information (see **Table 19-10**)
110_B	Reserved, do not use this combination.
111_B	Reserved, do not use this combination.

Table 19-10 CAN Bus State Information

CFC[13:12]	CAN Bus State
00_B	**NoBit** The CAN bus is idle, performs bit (de-) stuffing or is in one of the following frame segments: SOF, SRR, CRC, delimiters, first 6 EOF bits, IFS.
01_B	**NewBit** This code represents the first bit of a new frame segment. The current bit is the first bit in one of the following frame segments: Bit 10 (MSB) of standard ID (transmit only), RTR, reserved bits, IDE, DLC(MSB), bit 7 (MSB) in each data byte and the first bit of the ID extension.
10_B	**Bit** This code represents a bit inside a frame segment with a length of more than one bit (not the first bit of those frame segments that is indicated by NewBit). The current bit is processed within one of the following frame segments: ID bits (except first bit of standard ID for transmission and first bit of ID extension), DLC (3 LSB) and bits 6-0 in each data byte.
11_B	**Done** The current bit is in one of the following frame segments: Acknowledge slot, last bit of EOF, active/passive-error frame, overload frame. Two or more directly consecutive Done codes signal an Error Frame.

19.4.3 Message Object Registers

The Message Object Control Register MOCTRn and the Message Object Status Register MOSTATn are located at the same address offset within a message object address block (offset address $1C_H$). The MOCTRn is a write-only register that makes it possible to set/reset CAN transfer related control bits through software.

MOCTR0
Message Object 0 Control Register ($101C_H$) **Reset Value: $0100\ 0000_H$**
MOCTR127
Message Object 127 Control Register($1FFC_H$) **Reset Value: $7F7E\ 0000_H$**
MOCTRn (n = 1-126)
Message Object n Control Register
($101C_H+n*20_H$)
Reset Value: $((n+1)*01000000_H)+((n-1)*00010000_H)$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0				SETDIR	SETTXEN1	SETTXEN0	SETTXRQ	SETRXEN	SETRTSEL	SETMSGVAL	SETMSGLST	SETNEWDAT	SETRXUPD	SETTXPND	SETRXPND
w				w	w	w	w	w	w	w	w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				RESDIR	RESTXEN1	RESTXEN0	RESTXRQ	RESRXEN	RESRTSEL	RESMSGVAL	RESMSGLST	RESNEWDAT	RESRXUPD	RESTXPND	RESRXPND
w				w	w	w	w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
RESRXPND, SETRXPND	0, 16	w	**Reset/Set Receive Pending** These bits control the set/reset condition for RXPND (see **Table 19-11**).
RESTXPND, SETTXPND	1, 17	w	**Reset/Set Transmit Pending** These bits control the set/reset condition for TXPND (see **Table 19-11**).
RESRXUPD, SETRXUPD	2, 18	w	**Reset/Set Receive Updating** These bits control the set/reset condition for RXUPD (see **Table 19-11**).
RESNEWDAT, SETNEWDAT	3, 19	w	**Reset/Set New Data** These bits control the set/reset condition for NEWDAT (see **Table 19-11**).

Field	Bits	Type	Description
RESMSGLST, SETMSGLST	4, 20	w	**Reset/Set Message Lost** These bits control the set/reset condition for MSGLST (see **Table 19-11**).
RESMSGVAL, SETMSGVAL	5, 21	w	**Reset/Set Message Valid** These bits control the set/reset condition for MSGVAL (see **Table 19-11**).
RESRTSEL, SETRTSEL	6, 22	w	**Reset/Set Receive/Transmit Selected** These bits control the set/reset condition for RTSEL (see **Table 19-11**).
RESRXEN, SETRXEN	7, 23	w	**Reset/Set Receive Enable** These bits control the set/reset condition for RXEN (see **Table 19-11**).
RESTXRQ, SETTXRQ	8, 24	w	**Reset/Set Transmit Request** These bits control the set/reset condition for TXRQ (see **Table 19-11**).
RESTXEN0, SETTXEN0	9, 25	w	**Reset/Set Transmit Enable 0** These bits control the set/reset condition for TXEN0 (see **Table 19-11**).
RESTXEN1, SETTXEN1	10, 26	w	**Reset/Set Transmit Enable 1** These bits control the set/reset condition for TXEN1 (see **Table 19-11**).
RESDIR, SETDIR	11, 27	w	**Reset/Set Message Direction** These bits control the set/reset condition for DIR (see **Table 19-11**).
0	[15:12], [31:28]	w	**Reserved** Should be written with 0.

Table 19-11 Reset/Set Conditions for Bits in Register MOCTRn

RESy Bit[1)]	SETy Bit	Action on Write
Write 0	Write 0	Leave element unchanged
	No write	
No write	Write 0	
Write 1	Write 1	

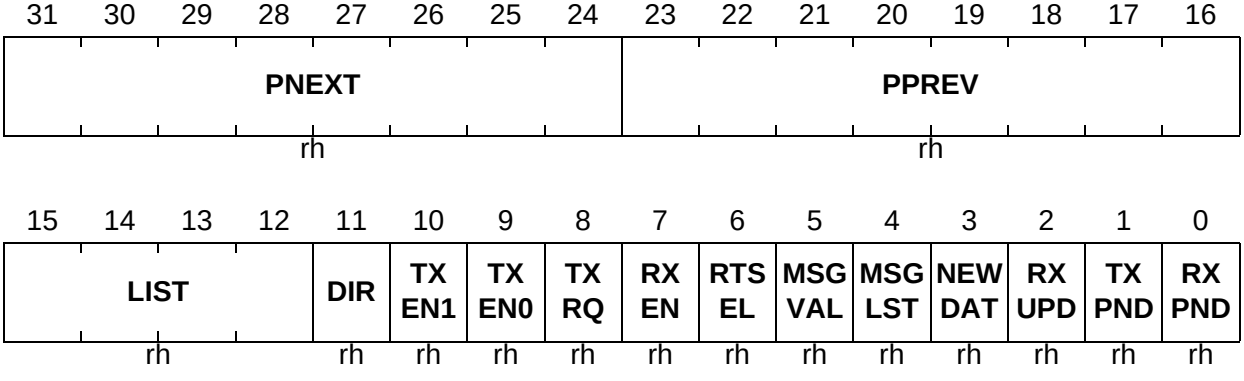
Table 19-11 Reset/Set Conditions for Bits in Register MOCTRn (cont'd)

RESy Bit[1)]	SETy Bit	Action on Write
Write 1	Write 0	Reset element
	No write	
Write 0	Write 1	Set element
No write		

1) The parameter "y" stands for the second part of the bit name ("RXPND", "TXPND", … up to "DIR").

The MOSTATn is a read-only register that indicates message object list status information such as the number of the current message object predecessor and successor message object, as well as the list number to which the message object is assigned.

MOSTAT0
Message Object 0 Status Register ($101C_H$) Reset Value: $0100\ 0000_H$
MOSTAT127
Message Object 127 Status Register ($1FFC_H$) Reset Value: $7F7E\ 0000_H$
MOSTATn (n = 1-126)
Message Object n Status Register
($101C_H+n*20_H$)
Reset Value: $((n+1)*01000000_H)+((n-1)*00010000_H)$



Field	Bits	Type	Description
RXPND	0	rh	**Receive Pending** 0_B No CAN message has been received. 1_B A CAN message has been received by the message object n, either directly or via gateway copy action. RXPND is not reset by hardware but must be reset by software.
TXPND	1	rh	**Transmit Pending** 0_B No CAN message has been transmitted. 1_B A CAN message from message object n has been transmitted successfully over the CAN bus. TXPND is reset by hardware but must be reset by software.

Field	Bits	Type	Description
RXUPD	2	rh	**Receive Updating** 0_B No receive update ongoing. 1_B Message identifier, DLC, and data of the message object are currently updated.
NEWDAT	3	rh	**New Data** 0_B No update of the message object n since last flag reset. 1_B Message object n has been updated. NEWDAT is set by hardware after a received CAN frame has been stored in message object n. NEWDAT is cleared by hardware when a CAN transmission of message object n has been started. NEWDAT should be set by software after the new transmit data has been stored in message object n to prevent the automatic reset of TXRQ at the end of an ongoing transmission.
MSGLST	4	rh	**Message Lost** 0_B No CAN message is lost. 1_B A CAN message is lost because NEWDAT has become set again when it has already been set.
MSGVAL	5	rh	**Message Valid** 0_B Message object n is not valid. 1_B Message object n is valid. Only a valid message object takes part in CAN transfers.

Field	Bits	Type	Description
RTSEL	6	rh	**Receive/Transmit Selected** 0_B Message object n is not selected for receive or transmit operation. 1_B Message object n is selected for receive or transmit operation. **Frame Reception:** RTSEL is set by hardware when message object n has been identified for storage of a CAN frame that is currently received. Before a received frame becomes finally stored in message object n, a check is performed to determine if RTSEL is set. Thus the CPU can suppress a scheduled frame delivery to this message object n by clearing RTSEL by software. **Frame Transmission:** RTSEL is set by hardware when message object n has been identified to be transmitted next. A check is performed to determine if RTSEL is still set before message object n is actually set up for transmission and bit NEWDAT is cleared. It is also checked that RTSEL is still set before its message object n is verified due to the successful transmission of a frame. RTSEL needs to be checked only when the context of message object n changes, and a conflict with an ongoing frame transfer shall be avoided. In all other cases, RTSEL can be ignored. RTSEL has no impact on message acceptance filtering. RTSEL is not cleared by hardware.
RXEN	7	rh	**Receive Enable** 0_B Message object n is not enabled for frame reception. 1_B Message object n is enabled for frame reception. RXEN is evaluated for receive acceptance filtering only.

Field	Bits	Type	Description
TXRQ	8	rh	**Transmit Request** 0_B No transmission of message object n is requested. 1_B Transmission of message object n on the CAN bus is requested. The transmit request becomes valid only if TXRQ, TXEN0, TXEN1 and MSGVAL are set. TXRQ is set by hardware if a matching Remote Frame has been received correctly. TXRQ is reset by hardware if message object n has been transmitted successfully and NEWDAT is not set again by software.
TXEN0	9	rh	**Transmit Enable 0** 0_B Message object n is not enabled for frame transmission. 1_B Message object n is enabled for frame transmission. Message object n can be transmitted only if both bits, TXEN0 and TXEN1, are set. The user may clear TXEN0 in order to inhibit the transmission of a message that is currently updated, or to disable automatic response of Remote Frames.
TXEN1	10	rh	**Transmit Enable 1** 0_B Message object n is not enabled for frame transmission. 1_B Message object n is enabled for frame transmission. Message object n can be transmitted only if both bits, TXEN0 and TXEN1, are set. TXEN1 is used by the MultiCAN module for selecting the active message object in the Transmit FIFOs.

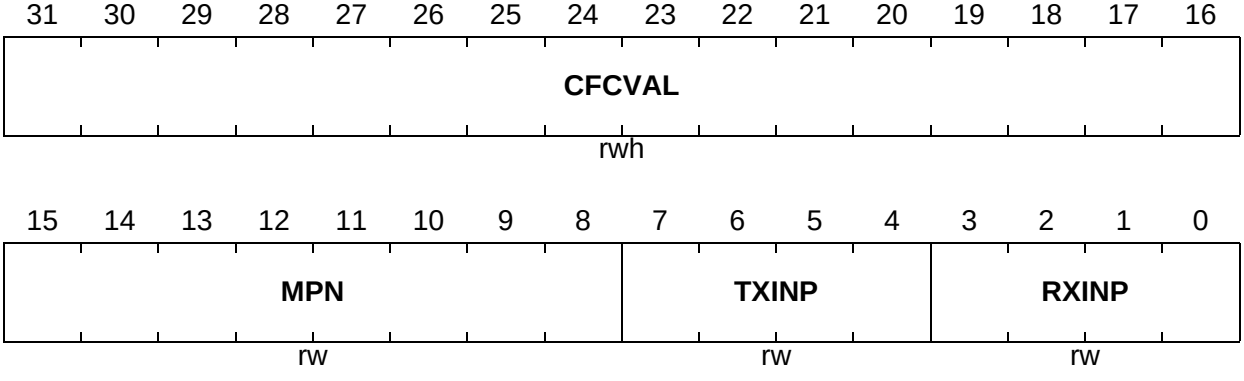
Field	Bits	Type	Description
DIR	11	rh	**Message Direction** 0_B Receive Object selected: With TXRQ = 1, a Remote Frame with the identifier of message object n is scheduled for transmission. On reception of a Data Frame with matching identifier, the message is stored in message object n. 1_B Transmit Object selected: If TXRQ = 1, message object n is scheduled for transmission of a Data Frame. On reception of a Remote Frame with matching identifier, bit TXRQ is set.
LIST	[15:12]	rh	**List Allocation** LIST indicates the number of the message list to which message object n is allocated. LIST is updated by hardware when the list allocation of the object is modified by a panel command.
PPREV	[23:16]	rh	**Pointer to Previous Message Object** PPREV holds the message object number of the previous message object in a message list structure.
PNEXT	[31:24]	rh	**Pointer to Next Message Object** PNEXT holds the message object number of the next message object in a message list structure.

Table 19-12 MOSTATn Reset Values

Message Object	PNEXT	PPREV	Reset Value
0	1	0	$0100\ 0000_H$
1	2	0	$0200\ 0000_H$
2	3	1	$0301\ 0000_H$
3	4	2	$0402\ 0000_H$
…	…	…	…
60	61	59	$3D3B\ 0000_H$
61	62	60	$3E3C\ 0000_H$
62	63	61	$3F3D\ 0000_H$
63	63	62	$3F3E\ 0000_H$

The Message Object Interrupt Pointer Register MOIPRn holds the message interrupt pointers, the message pending number, and the frame counter value of message object n.

MOIPRn (n = 0-127)
Message Object n Interrupt Pointer Register
(1008_H+n*20_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
RXINP	[3:0]	rw	**Receive Interrupt Node Pointer** RXINP selects the interrupt output line INT_Om (m = 0-15) for a receive interrupt event of message object n. RXINP can also be taken for message pending bit selection (see **Page 19-39**). 0000_B Interrupt output line INT_O0 is selected. 0001_B Interrupt output line INT_O1 is selected. $\ldots_B$ … 1110_B Interrupt output line INT_O14 is selected. 1111_B Interrupt output line INT_O15 is selected.
TXINP	[7:4]	rw	**Transmit Interrupt Node Pointer** TXINP selects the interrupt output line INT_Om (m = 0-15) for a transmit interrupt event of message object n. TXINP can also be taken for message pending bit selection (see **Page 19-39**). 0000_B Interrupt output line INT_O0 is selected. 0001_B Interrupt output line INT_O1 is selected. $\ldots_B$ … 1110_B Interrupt output line INT_O14 is selected. 1111_B Interrupt output line INT_O15 is selected.

Field	Bits	Type	Description
MPN	[15:8]	rw	**Message Pending Number** This bit field selects the bit position of the bit in the Message Pending Register that is set upon a message object n receive/transmit interrupt.
CFCVAL	[31:16]	rwh	**CAN Frame Counter Value** When a message is stored in message object n or message object n has been successfully transmitted, the CAN frame counter value NFCRx.CFC is then copied to CFCVAL.

The Message Object Function Control Register MOFCRn contains bits that select and configure the function of the message object. It also holds the CAN data length code.

MOFCRn (n = 0-127)
Message Object n Function Control Register
(1000_H+n*20_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0				DLC				STT	SDT	RMM	FRR EN	0	OVIE	TXIE	RXIE
rw				rwh				rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				DATC	DLCC	IDC	GDFS	0				MMC			
rw				rw	rw	rw	rw	rw				rw			

Field	Bits	Type	Description
MMC	[3:0]	rw	**Message Mode Control** MMC controls the message mode of message object n. 0000_B Standard Message Object 0001_B Receive FIFO Base Object 0010_B Transmit FIFO Base Object 0011_B Transmit FIFO Slave Object 0100_B Gateway Source Object $..._B$ Reserved
GDFS	8	rw	**Gateway Data Frame Send** 0_B TXRQ is unchanged in the destination object. 1_B TXRQ is set in the gateway destination object after the internal transfer from the gateway source to the gateway destination object. Applicable only to a gateway source object; ignored in other nodes.

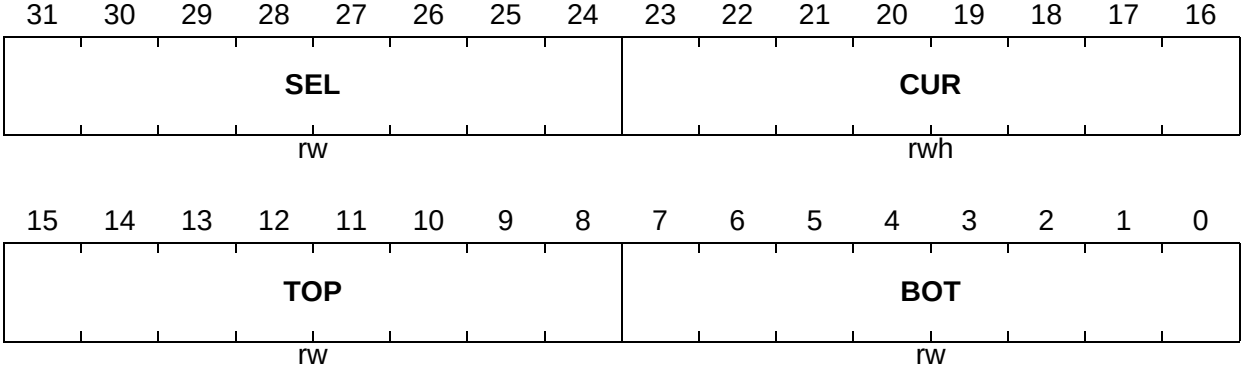
Field	Bits	Type	Description
IDC	9	rw	**Identifier Copy** 0_B The identifier of the gateway source object is not copied. 1_B The identifier of the gateway source object (after storing the received frame in the source) is copied to the gateway destination object. Applicable only to a gateway source object; ignored in other nodes.
DLCC	10	rw	**Data Length Code Copy** 0_B Data length code is not copied. 1_B Data length code of the gateway source object (after storing the received frame in the source) is copied to the gateway destination object. Applicable only to a gateway source object; ignored in other nodes.
DATC	11	rw	**Data Copy** 0_B Data fields are not copied. 1_B Data fields in registers MODATALn and MODATAHn of the gateway source object (after storing the received frame in the source) are copied to the gateway destination. Applicable only to a gateway source object; ignored in other nodes.
RXIE	16	rw	**Receive Interrupt Enable** RXIE enables the message receive interrupt of message object n. This interrupt is generated after reception of a CAN message (independent of whether the CAN message is received directly or indirectly via a gateway action). 0_B Message receive interrupt is disabled. 1_B Message receive interrupt is enabled. Bit field MOIPRn.RXINP selects the interrupt output line which becomes activated at this type of interrupt.

Field	Bits	Type	Description
TXIE	17	rw	**Transmit Interrupt Enable** TXIE enables the message transmit interrupt of message object n. This interrupt is generated after the transmission of a CAN message. 0_B Message transmit interrupt is disabled. 1_B Message transmit interrupt is enabled. Bit field MOIPRn.TXINP selects the interrupt output line which becomes activated at this type of interrupt.
OVIE	18	rw	**Overflow Interrupt Enable** OVIE enables the FIFO full interrupt of message object n. This interrupt is generated when the pointer to the current message object (CUR) reaches the value of SEL in the FIFO/Gateway Pointer Register. 0_B FIFO full interrupt is disabled. 1_B FIFO full interrupt is enabled. If message object n is a Receive FIFO base object, bit field MOIPRn.TXINP selects the interrupt output line which becomes activated at this type of interrupt. If message object n is a Transmit FIFO base object, bit field MOIPRn.RXINP selects the interrupt output line which becomes activated at this type of interrupt. For all other message object modes, bit OVIE has no effect.
FRREN	20	rw	**Foreign Remote Request Enable** Specifies whether the TXRQ bit is set in message object n or in a foreign message object referenced by the pointer CUR. 0_B TXRQ of message object n is set on reception of a matching Remote Frame. 1_B TXRQ of the message object referenced by the pointer CUR is set on reception of a matching Remote Frame.

Field	Bits	Type	Description
RMM	21	rw	**Transmit Object Remote Monitoring** 0_B Remote monitoring is disabled: Identifier, IDE bit, and DLC of message object n remain unchanged upon the reception of a matching Remote Frame. 1_B Remote monitoring is enabled: Identifier, IDE bit, and DLC of a matching Remote Frame are copied to transmit object n in order to monitor incoming Remote Frames. Bit RMM applies only to transmit objects and has no effect on receive objects.
SDT	22	rw	**Single Data Transfer** If SDT = 1 and message object n is not a FIFO base object, then MSGVAL is reset when this object has taken part in a successful data transfer (receive or transmit). If SDT = 1 and message object n is a FIFO base object, then MSGVAL is reset when the pointer to the current object CUR reaches the value of SEL in the FIFO/Gateway Pointer Register. With SDT = 0, bit MSGVAL is not affected.
STT	23	rw	**Single Transmit Trial** If this bit is set, then TXRQ is cleared on transmission start of message object n. Thus, no transmission retry is performed in case of transmission failure.
DLC	[27:24]	rwh	**Data Length Code** Bit field determines the number of data bytes for message object n. Valid values for DLC are 0 to 8. A value of DLC > 8 results in a data length of 8 data bytes. If a frame with DLC > 8 is received, the received value is stored in the message object.
0	[7:4], [15:12], 19, [31:28]	rw	**Reserved** Read as 0 after reset; value last written is read back; should be written with 0.

The Message Object FIFO/Gateway Pointer register MOFGPRn contains a set of message object link pointers that are used for FIFO and gateway operations.

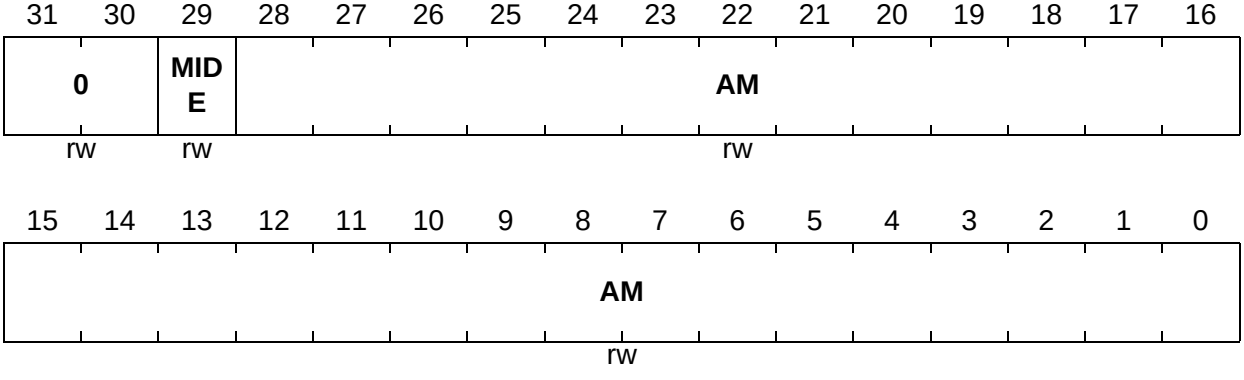
MOFGPRn (n = 0-127)
Message Object n FIFO/Gateway Pointer Register
(1004_H+n*20_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
BOT	[7:0]	rw	**Bottom Pointer** Bit field BOT points to the first element in a FIFO structure.
TOP	[15:8]	rw	**Top Pointer** Bit field TOP points to the last element in a FIFO structure.
CUR	[23:16]	rwh	**Current Object Pointer** Bit field CUR points to the actual target object within a FIFO/Gateway structure. After a FIFO/gateway operation CUR is updated with the message number of the next message object in the list structure (given by PNEXT of the message control register) until it reaches the FIFO top element (given by TOP) when it is reset to the bottom element (given by BOT).
SEL	[31:24]	rw	**Object Select Pointer** Bit field SEL is the second (software) pointer to complement the hardware pointer CUR in the FIFO structure. SEL is used for monitoring purposes (FIFO interrupt generation).

Message Object n Acceptance Mask Register MOAMRn contains the mask bits for the acceptance filtering of the message object n.

MOAMRn (n = 0-127)
Message Object n Acceptance Mask Register
($100C_H$+n*20_H) **Reset Value: $3FFF\ FFFF_H$**

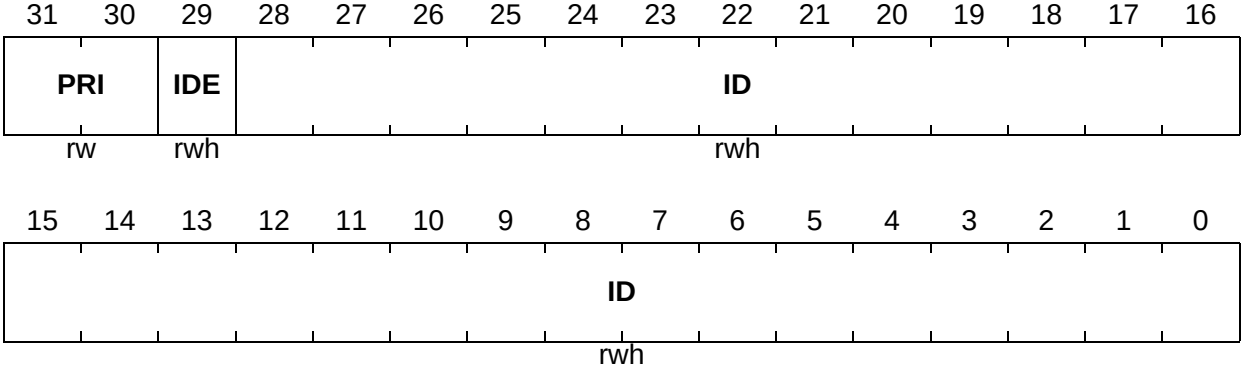


Field	Bits	Type	Description
AM	[28:0]	rw	**Acceptance Mask for Message Identifier** Bit field AM is the 29-bit mask for filtering incoming messages with standard identifiers (AM[28:18]) or extended identifiers (AM[28:0]). For standard identifiers, bits AM[17:0] are “don’t care”.
MIDE	29	rw	**Acceptance Mask Bit for Message IDE Bit** 0_B Message object n accepts the reception of both, standard and extended frames. 1_B Message object n receives frames only with matching IDE bit.
0	[31:30]	rw	**Reserved** Read as 0 after reset; value last written is read back; should be written with 0.

Message Object n Arbitration Register MOARn contains the CAN identifier of the message object.

MOARn (n = 0-127)
Message Object n Arbitration Register

(1018_H+n*20_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
ID	[28:0]	rwh	**CAN Identifier of Message Object n** Identifier of a standard message (ID[28:18]) or an extended message (ID[28:0]). For standard identifiers, bits ID[17:0] are “don’t care”.
IDE	29	rwh	**Identifier Extension Bit of Message Object n** 0_B Message object n handles standard frames with 11-bit identifier. 1_B Message object n handles extended frames with 29-bit identifier.

Field	Bits	Type	Description
PRI	[31:30]	rw	**Priority Class** PRI assigns one of the four priority classes 0, 1, 2, 3 to message object n. A lower PRI number defines a higher priority. Message objects with lower PRI value always win acceptance filtering for frame reception and transmission over message objects with higher PRI value. Acceptance filtering based on identifier/mask and list position is performed only between message objects of the same priority class. PRI also determines the acceptance filtering method for transmission: 00_B Reserved. 01_B Transmit acceptance filtering is based on the list order. This means that message object n is considered for transmission only if there is no other message object with valid transmit request (MSGVAL & TXEN0 & TXEN1 = 1) somewhere before this object in the list. 10_B Transmit acceptance filtering is based on the CAN identifier. This means, message object n is considered for transmission only if there is no other message object with higher priority identifier + IDE + DIR (with respect to CAN arbitration rules) somewhere in the list (see **Table 19-13**). 11_B Transmit acceptance filtering is based on the list order (as PRI = 01_B).

Transmit Priority of Msg. Objects based on CAN Arbitration Rules

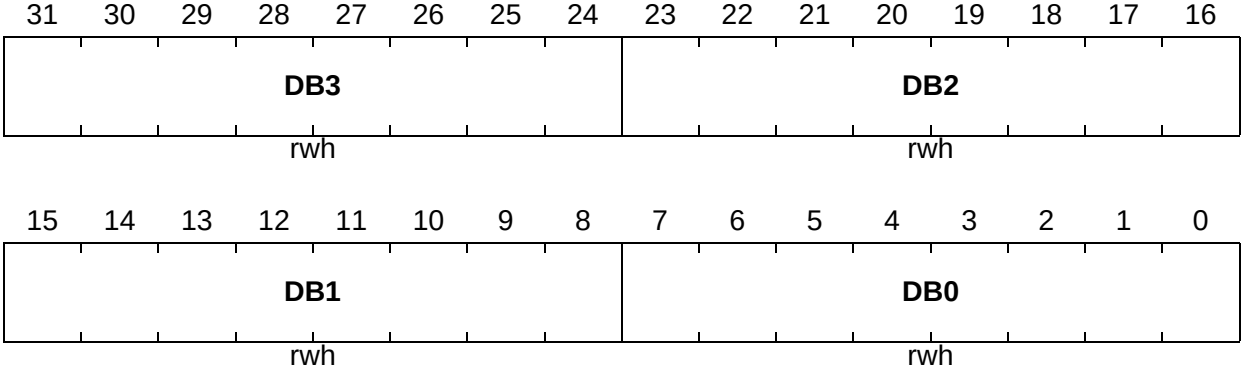
Table 19-13 Transmit Priority of Msg. Objects Based on CAN Arbitration Rules

Settings of Arbitrarily Chosen Message Objects A and B, (A has higher transmit priority than B)	Comment
A.MOAR[28:18] < B.MOAR[28:18] (11-bit standard identifier of A less than 11-bit standard identifier of B)	Messages with lower standard identifier have higher priority than messages with higher standard identifier. MOAR[28] is the most significant bit (MSB) of the standard identifier. MOAR[18] is the least significant bit of the standard identifier.
A.MOAR[28:18] = B.MOAR[28:18] A.MOAR.IDE = 0 (send Standard Frame) B.MOAR.IDE = 1 (send Extended Frame)	Standard Frames have higher transmit priority than Extended Frames with equal standard identifier.
A.MOAR[28:18] = B.MOAR[28:18] A.MOAR.IDE = B.MOAR.IDE = 0 A.MOCTR.DIR = 1 (send Data Frame) B.MOCTR.DIR = 0 (send Remote Fame)	Standard Data Frames have higher transmit priority than standard Remote Frames with equal identifier.
A.MOAR[28:0] = B.MOAR[28:0] A.MOAR.IDE = B.MOAR.IDE = 1 A.MOCTR.DIR = 1 (send Data Frame) B.MOCTR.DIR = 0 (send Remote Frame)	Extended Data Frames have higher transmit priority than Extended Remote Frames with equal identifier.
A.MOAR[28:0] < B.MOAR[28:0] A.MOAR.IDE = B.MOAR.IDE = 1 (29-bit identifier)	Extended Frames with lower identifier have higher transmit priority than Extended Frames with higher identifier. MOAR[28] is the most significant bit (MSB) of the overall identifier (standard identifier MOAR[28:18] and identifier extension MOAR[17:0]). MOAR[0] is the least significant bit (LSB) of the overall identifier.

Message Object n Data Register Low MODATALn contains the lowest four data bytes of message object n. Unused data bytes are set to zero upon reception and ignored for transmission.

MODATALn (n = 0-127)
Message Object n Data Register Low

(1010_H+n*20_H) **Reset Value: 0000 0000_H**

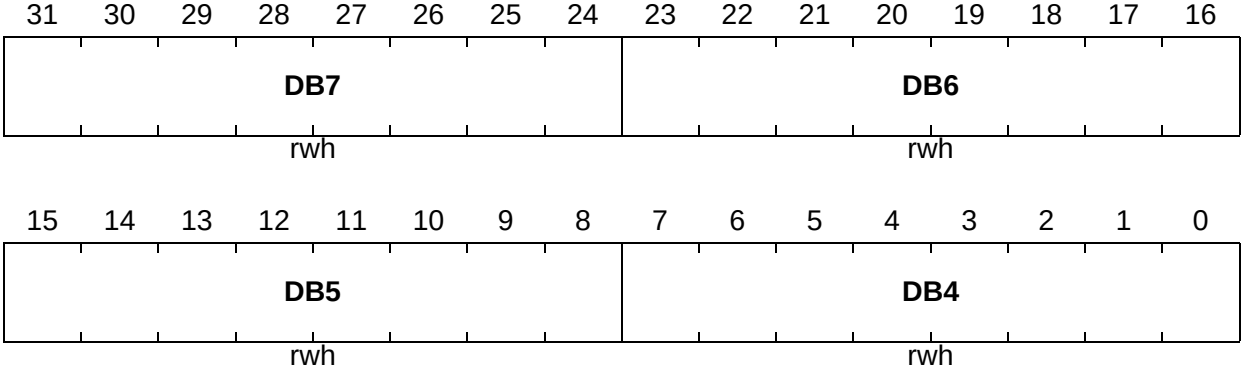


Field	Bits	Type	Description
DB0	[7:0]	rwh	**Data Byte 0 of Message Object n**
DB1	[15:8]	rwh	**Data Byte 1 of Message Object n**
DB2	[23:16]	rwh	**Data Byte 2 of Message Object n**
DB3	[31:24]	rwh	**Data Byte 3 of Message Object n**

Message Object n Data Register High MODATAH contains the highest four data bytes of message object n. Unused data bytes are set to zero upon reception and ignored for transmission.

MODATAHn (n = 0-127)
Message Object n Data Register High

(1014_H+n*20_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
DB4	[7:0]	rwh	**Data Byte 4 of Message Object n**
DB5	[15:8]	rwh	**Data Byte 5 of Message Object n**
DB6	[23:16]	rwh	**Data Byte 6 of Message Object n**
DB7	[31:24]	rwh	**Data Byte 7 of Message Object n**

19.5 MultiCAN Module Implementation

This section describes CAN module interfaces with the clock control, port connections, interrupt control, and address decoding.

19.5.1 Interfaces of the MultiCAN Module

Figure 19-24 shows the TC1797 specific implementation details and interconnections of the MultiCAN module. The eight I/O lines of the MultiCAN module (two I/O lines of each CAN node) are connected to I/O lines of Port 6. The MultiCAN module is also supplied by clock control, interrupt control, and address decoding logic. MultiCAN interrupts can be directed to the DMA controller and the GPTA modules.

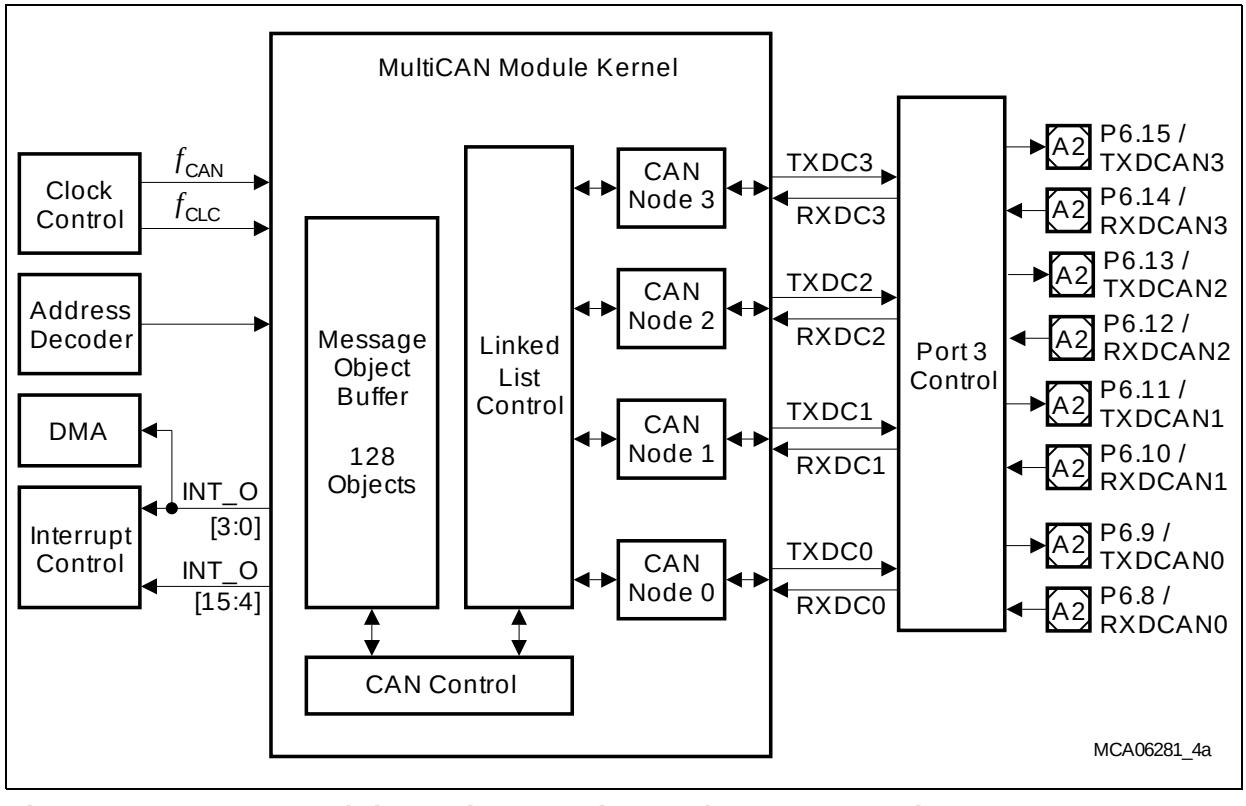


Figure 19-24 CAN Module Implementation and Interconnections

19.5.2 MultiCAN Module External Registers

The registers listed in **Figure 19-25** are not included in the MultiCAN module kernel but must be programmed for proper operation of the MultiCAN module.

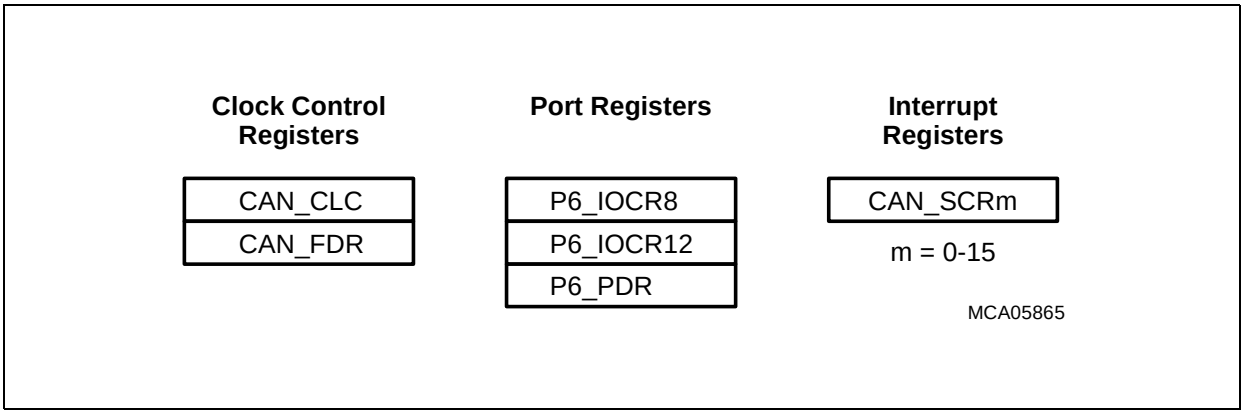


Figure 19-25 CAN Implementation-Specific Special Function Registers

19.5.3 Module Clock Generation

As shown in **Figure 19-26**, the clock signals for the MultiCAN module are generated and controlled by a clock control unit. This clock generation unit is responsible for the enable/disable control, the clock frequency adjustment, and the debug clock control. This unit includes two registers:

- CAN_CLC: generation of the module control clock f_{CLC}
- CAN_FDR: frequency control of the module timer clock f_{CAN}

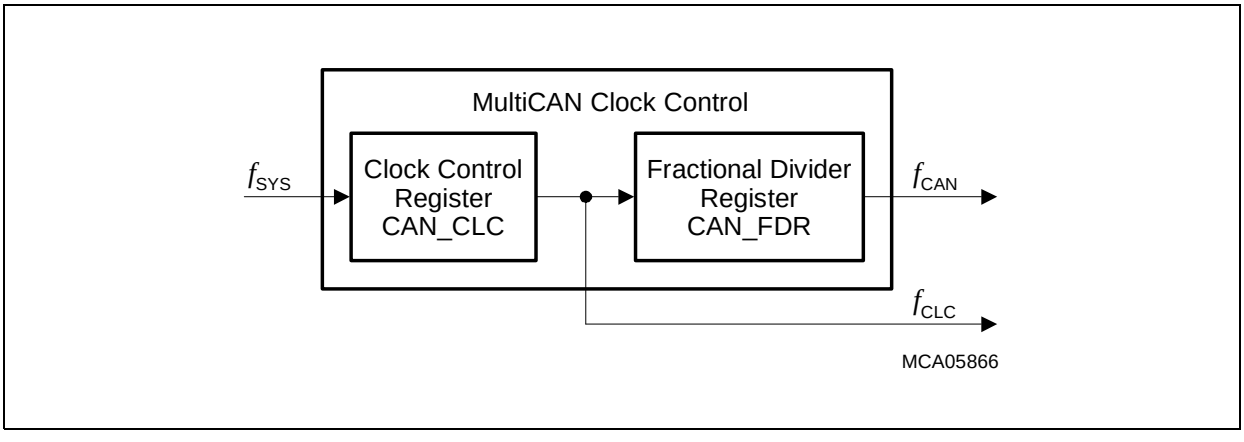


Figure 19-26 MultiCAN Module Clock Generation

The module control clock f_{CLC} is used inside the MultiCAN module for control purposes such as clocking of control logic and register operations. The frequency of f_{CLC} is identical to the system clock frequency f_{SYS}. The clock control register CAN_CLC makes it possible to enable/disable f_{CLC} under certain conditions.

The module timer clock f_{CAN} is used inside the MultiCAN module as input clock for all timing relevant operations (e.g. bit timing). The settings in the CAN_FDR register determine the frequency of the module timer clock f_{CAN} according the following two formulas:

$$f_{CAN} = f_{SYS} \times \frac{1}{n} \quad \text{with n = 1024 - CAN\_FDR.STEP} \qquad (19.1)$$

$$f_{CAN} = f_{SYS} \times \frac{n}{1024} \quad \text{with n = 0-1023} \qquad (19.2)$$

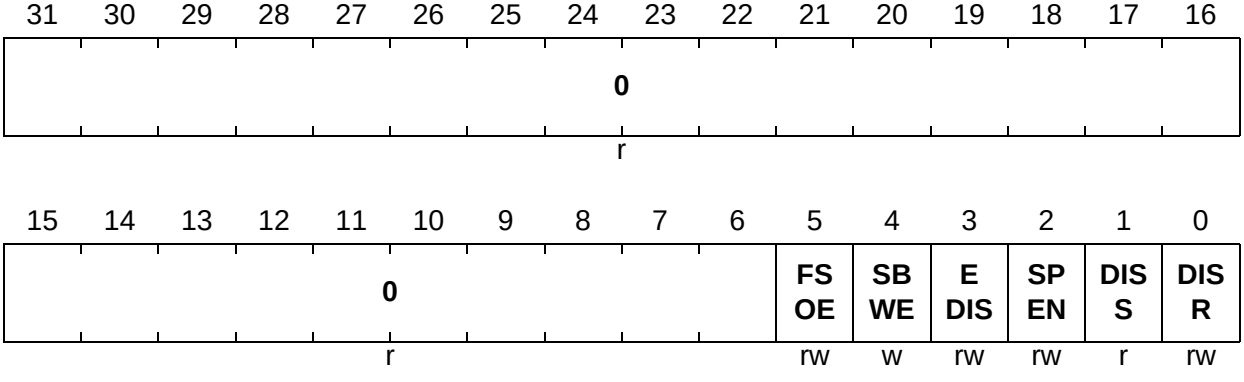
Equation (19.1) applies to normal divider mode (CAN_FDR.DM = 01_B) of the fractional divider. **Equation (19.2)** applies to fractional divider mode (CAN_FDR.DM = 10_B).

Note: The CAN module is disabled after reset. In general, after reset, the module control clock f_{CLC} must be switched on (writing to register CAN_CLC) before the frequency of the module timer clock f_{CAN} is defined (writing to register CAN_FDR).

19.5.3.1 Clock Control Registers

The clock control register makes it possible to control (enable/disable) the module control clock f_{CLC}.

CAN_CLC
CAN Clock Control Register (000_H) Reset Value: 0000 0003_H



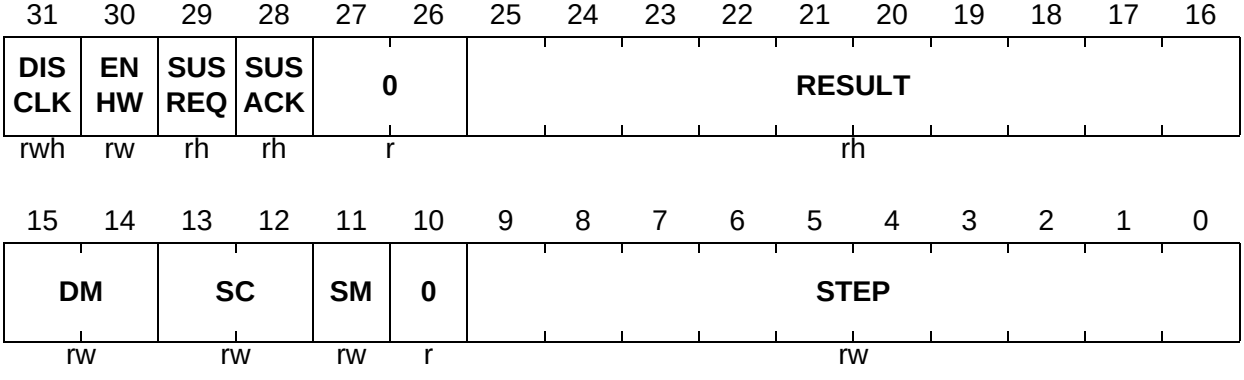
Field	Bits	Type	Description
DISR	0	rw	**Module Disable Request Bit** Used for enable/disable control of the module.
DISS	1	r	**Module Disable Status Bit** Bit indicates the current status of the module.
SPEN	2	rw	**Module Suspend Enable for OCDS** Used to enable the Suspend Mode.
EDIS	3	rw	**External Request Disable** Used to control external clock disable request.
SBWE	4	w	**Module Suspend Bit Write Enable for OCDS** Determines whether SPEN and FSOE are write-protected.
FSOE	5	rw	**Fast Switch Off Enable** Used for fast clock switch off in Suspend Mode.
0	[31:6]	r	**Reserved** Read as 0; should be written with 0.

*Note: Additional details about the clock control register functionality are described in section **"Clock Control Register CLC" on Page 3-47** of the TC1797 User´s Manual System Units part (Volume 1).*

Note: In disabled state, no registers of the CAN module can be read or written except the CAN_CLC register.

The fractional divider register allows the programmer to control the clock rate of the module timer clock f_{CAN}.

FDR
Fractional Divider Register ($00C_H$) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
STEP	[9:0]	rw	**Step Value** Reload or addition value for RESULT.
SM	11	rw	**Suspend Mode** SM selects between granted or immediate Suspend Mode.
SC	[13:12]	rw	**Suspend Control** This bit field determines the behavior of the fractional divider in Suspend Mode.
DM	[15:14]	rw	**Divider Mode** This bit field selects normal divider mode, fractional divider mode, and off-state.
RESULT	[25:16]	rh	**Result Value** Bit field for the addition result.
SUSACK	28	rh	**Suspend Mode Acknowledge** Indicates state of SPNDACK signal.
SUSREQ	29	rh	**Suspend Mode Request** Indicates state of SPND signal.
ENHW	30	rw	**Enable Hardware Clock Control** Controls operation of ECEN input and DISCLK bit.

Field	Bits	Type	Description
DISCLK	31	rwh	**Disable Clock** Hardware controlled disable for f_{OUT} signal.
0	10, [27:26]	r	**Reserved** Read as 0; should be written with 0.

*Note: Additional details about the fractional divider register functionality are described in section **"Fractional Divider Operating Modes" on Page 3-26** of the TC1797 User´s Manual System Units part (Volume 1).*

19.5.4 Port and I/O Line Control

The interconnections between the MultiCAN module and the port I/O lines are controlled in the port logic. Additionally to the port input selection, the following port control operations selections must be executed:

- Input/output function selection (IOCR registers)
- Pad driver characteristics selection for the outputs (PDR registers)

19.5.4.1 Input/Output Function Selection in Ports

The port input/output control registers contain the bit fields that select the digital output and input driver characteristics such as pull-up/down devices, port direction (input/output), open-drain, and alternate output selections. The I/O lines for the MultiCAN module are controlled by the port input/output control registers P6_IOCR8 and P6_OCR12.

Table 19-14 shows how bits and bit fields must be programmed for the required I/O functionality of the CAN I/O lines.

Table 19-14 MultiCAN I/O Control Selection and Setup

Module	Port Lines	Input/Output Control Register Bits	I/O
CAN	P6.8 / RXDCAN0	P6_IOCR8.PC8 = $0XXX_B$	Input
	P6.9 / TXDCAN0	P6_IOCR8.PC9 = $1X01_B$	Output
	P6.10 / RXDCAN1	P6_IOCR8.PC10 = $0XXX_B$	Input
	P6.11 / TXDCAN1	P6_IOCR8.PC11 = $1X01_B$	Output
	P6.12 / RXDCAN2	P6_IOCR12.PC12 = $0XXX_B$	Input
	P6.13 / TXDCAN2	P6_IOCR12.PC13 = $1X01_B$	Output
	P6.14 / RXDCAN3	P6_IOCR12.PC14 = $0XXX_B$	Input
	P6.15 / TXDCAN3	P6_IOCR12.PC15 = $1X01_B$	Output

19.5.4.2 Node Receive Input Selection

Additionally to the I/O control selection, as defined in **Table 19-14**, one of the four CAN node's eight receive input lines must be selected using bit field RXSEL in its node port control register NPCRx. Two of the four CAN node's receive input lines are connected with GPIO lines according to **Table 19-15**.

This feature allows, for example, a CAN node which operates in analyzer mode to monitor the receive operations of its neighbor CAN node. The default setting after reset of a node's NPCRx.RXSEL bit field connect node x with RXDCANx I/O line (x = 0-3).

Table 19-15 Receive Input Selection

Receive Input of	Connected to Port Pin	Selected by[1)]
CAN Node 0	P6.8 / RXDCAN0	NPCR0.RXSEL = 000_B
	P6.10 / RXDCAN1	NPCR0.RXSEL = 001_B
CAN Node 1	P6.10 / RXDCAN1	NPCR1.RXSEL = 000_B
	P6.12 / RXDCAN2	NPCR1.RXSEL = 001_B
CAN Node 2	P6.12 / RXDCAN2	NPCR2.RXSEL = 000_B
	P6.14 / RXDCAN3	NPCR2.RXSEL = 001_B
CAN Node 3	P6.14 / RXDCAN3	NPCR3.RXSEL = 000_B
	P6.8 / RXDCAN0	NPCR3.RXSEL = 001_B

1) Other values for NPCRx.RXSEL than 000_B and 001_B result in the following behavior for a CANx node:
Recessive receive input level for RXSEL = 010_B to 110_B.
Dominant receive input level for RXSEL = 111_B.

19.5.4.3 DMA Request Outputs

The interrupt output lines INT_O0 to INT_O2 of the MultiCAN module can be used as a DMA requestor and are able to trigger DMA transfers. INT_O[2:0] are connected to the DMA controller as shown in **Table 19-16**.

Table 19-16 CAN-to-DMA Request Connections

DMA Channel	Connected to CAN Interrupt Output	Selected in DMA Controller by Programming
04	INT_O0	CHCR04.PRSEL = 101_B
05	INT_O0	CHCR05.PRSEL = 101_B
06	INT_O1	CHCR06.PRSEL = 110_B
07	INT_O2	CHCR07.PRSEL = 110_B
15	INT_O3	CHCR15.PRSEL = 101_B

19.5.5 Interrupt Control

The interrupt control logic in the MultiCAN module uses an interrupt compressing scheme that allows high flexibility in interrupt processing. There are 274 hardware interrupt sources and one software interrupt source available:

- CAN node interrupts:
  - Four different interrupt sources for each of the four CAN nodes = 16 interrupt sources
- Message object interrupts:
  - Two interrupt sources for each message object = 256 interrupt sources
- One software initiated interrupt (register MITR)

Each hardware initiated interrupt source is controlled by a 4-bit interrupt pointer that directs the interrupt source to one of the sixteen interrupt outputs INT_Om (m = 0-15). This makes it possible to connect more than one interrupt source (between one and all) to one interrupt output line. The interrupt wiring matrix shown in **Figure 19-27** is built up according to the following rules:

- Each output of the 4-bit interrupt pointer demultiplexer is connected to exactly one OR-gate input of the INT_Om line. The number “m” of the corresponding selected INT_Om interrupt output line is defined by the interrupt pointer value.
- Each INT_Om output line has an input OR-gate which is connected to all interrupt pointer demultiplexer outputs which are selected by an identical 4-bit pointer value.

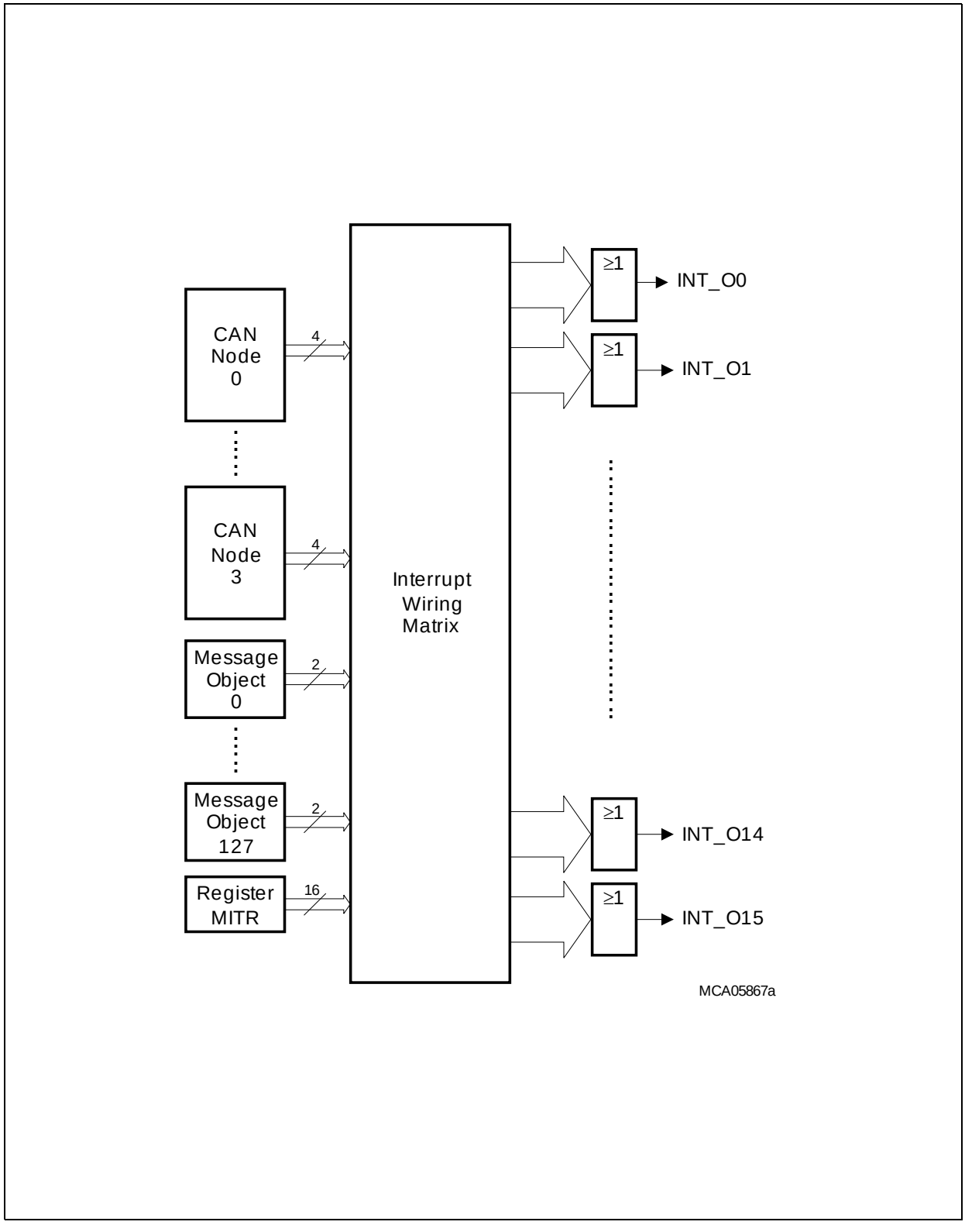
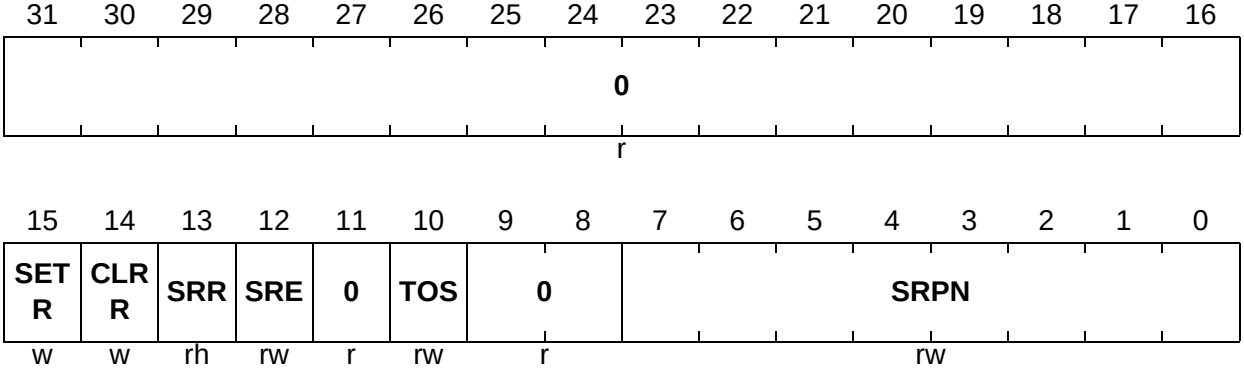


Figure 19-27 Interrupt Compressor

19.5.5.1 Service Request Control Registers

Each of the sixteen interrupt outputs INT_Om of the MultiCAN module is controlled by one service request control register.

CAN_SRCm (m = 0-15)
CAN Service Request Control Register m
($0FC_H$-m*4_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
SRPN	[7:0]	rw	**Service Request Priority Number**
TOS	10	rw	**Type of Service Control**
SRE	12	rw	**Service Request Enable**
SRR	13	rh	**Service Request Flag**
CLRR	14	w	**Request Clear Bit**
SETR	15	w	**Request Set Bit**
0	[9:8], 11, [31:16]	r	**Reserved** Read as 0; should be written with 0.

*Note: Additional details on service request nodes and the service request control registers are described in section **"Service Request Nodes" on Page 13-3** of the TC1797 User´s Manual System Units part (Volume 1).*

Some of the sixteen interrupt outputs of the MultiCAN module can be used to trigger operations in the DMA controller and the GPTA module.

19.5.6 Parity Protection for CAN Memories

In the TC1797, the static RAM memories in the MultiCAN module which contain message objects, are equipped with a parity error detection logic that makes it possible to detect parity errors. In case of a parity error a NMI is generated.

At a power-on reset operation the corresponding MultiCAN module memories are initialized automatically. Therefore, unlike other TC1797 on-chip memories the MultiCAN module memories must not be initialized by a user program before it can be enabled for parity error detection.

19.5.6.1 CAN Module Register Map

The complete MultiCAN module register address map is shown in **Figure 19-28**. It shows the general implementation-specific registers for clock control, module identification, and interrupt service request control and adds the absolute address information.

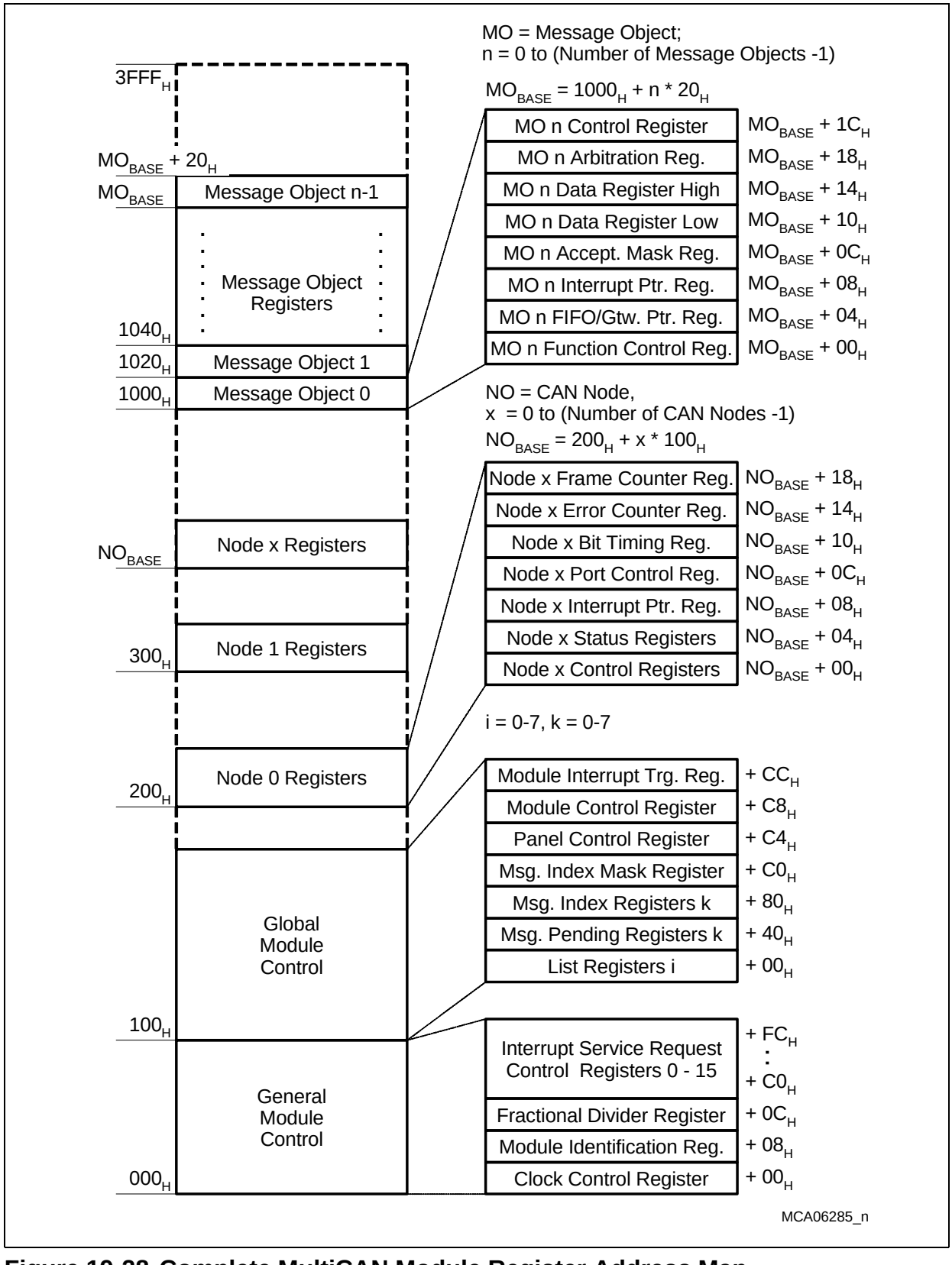


Figure 19-28 Complete MultiCAN Module Register Address Map

20 FlexRay™ Protocol Controller (E-Ray)

The E-Ray IP-module performs communication according to the FlexRay™ [1] protocol specification v2.1. With maximum specified clock the bitrate can be programmed to values up to 10 Mbit/s. Additional bus driver (BD) hardware is required for connection to the physical layer.

20.1 E-Ray Kernel Description

Figure 20.1 shows a global view of the E-Ray interface.

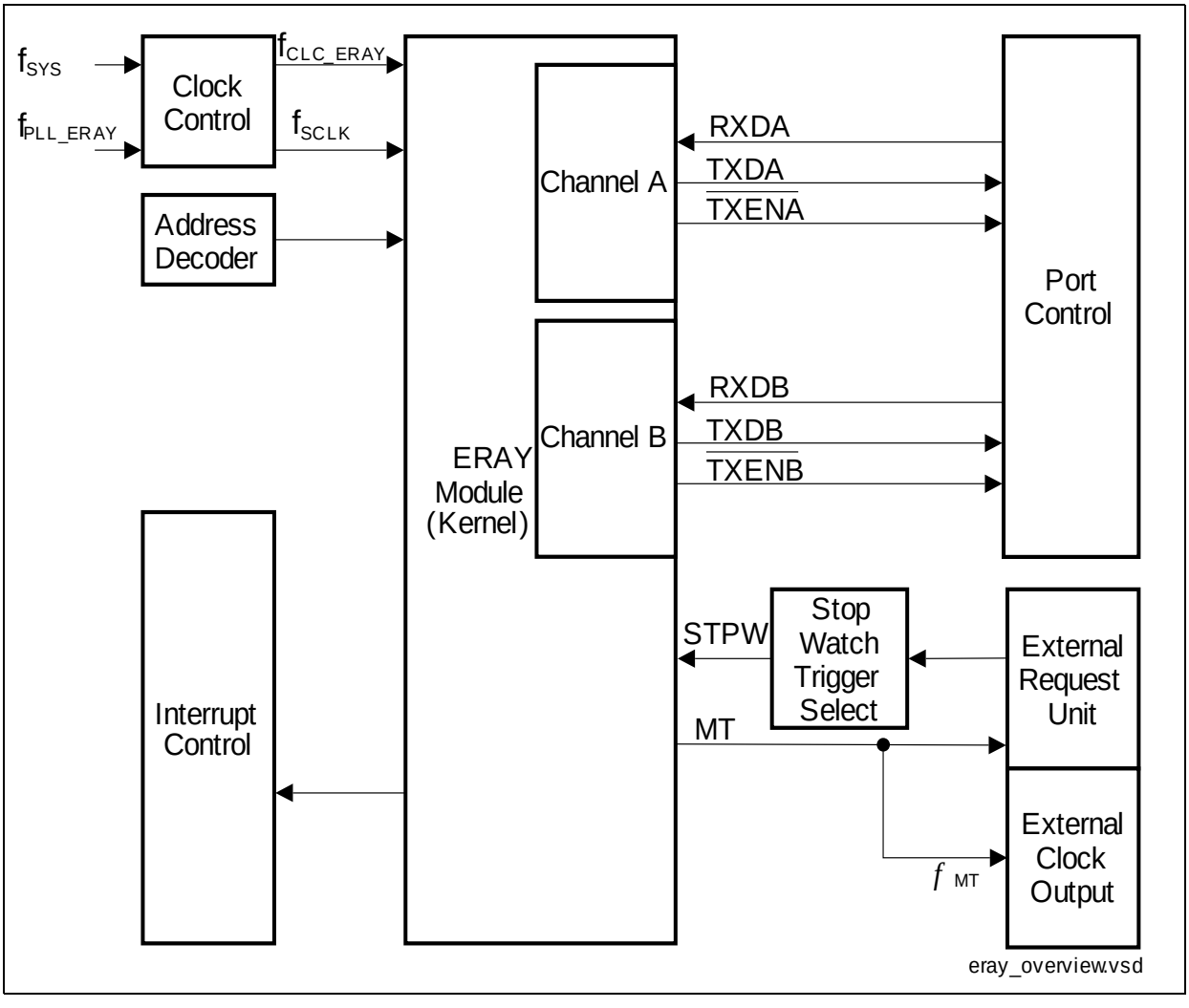


Figure 20-1 General Block Diagram of the E-Ray Interface

1) *Infineon®, Infineon Technologies®, are trademarks of Infineon Technologies AG. FlexRay™* is a trademark of FlexRay Consortium.

The E-Ray module communicates with the external world via three I/O lines each channel. The RXDAx and RXDBx lines are the receive data input signals, TXDA and TXDB lines are the transmit output signals, TXENA and TXENB the transmit enable signals.

Clock control, address decoding, and service request control are managed outside the E-Ray module kernel.

20.2 Overview

For communication on a FlexRay™ network, individual Message Buffers with up to 254 data byte are configurable. The message storage consists of a single-ported Message RAM that holds up to 128 Message Buffers. All functions concerning the handling of messages are implemented in the Message Handler. Those functions are the acceptance filtering, the transfer of messages between the two FlexRay™ Channel Protocol Controllers and the Message RAM, maintaining the transmission schedule as well as providing message status information.

The register set of the E-Ray IP-module can be accessed directly by an external Host via the module's Host interface. These registers are used to control/configure/monitor the FlexRay™ Channel Protocol Controllers, Message Handler, Global Time Unit, System Universal Control, Frame and Symbol Processing, Network Management, Service Request Control, and to access the Message RAM via Input / Output Buffer.

The E-Ray IP-module supports the following features:

- Conformance with FlexRay™ protocol specification v2.1
- Data rates of up to 10 Mbit/s on each channel
- Up to 128 Message Buffers configurable
- 8 Kbyte of Message RAM for storage of e.g. 128 Message Buffers with max. 48 byte data field or up to 30 Message Buffers with 254 byte Data Sections
- Configuration of Message Buffers with different payload lengths possible
- One configurable receive FIFO
- Each Message Buffer can be configured as receive buffer, as transmit buffer or as part of the receive FIFO
- Host access to Message Buffers via Input and Output Buffer.
  Input Buffer: Holds message to be transferred to the Message RAM
  Output Buffer: Holds message read from the Message RAM
- Filtering for slot counter, cycle counter, and channel
- Maskable module service requests
- Network Management supported
- Four service request lines
- Automatic delayed read access to Output Command Request Register (OBCR) if a data transfer from Message RAM to Output Shadow Buffer (initiated by a previous write access to the OBCR) is ongoing.

- Automatic delayed read access to Input Command Request Register (IBCR) if a data transfer from Input Shadow Buffer to Message RAM to (initiated by a previous write access to the IBCR) is ongoing.
- Four Input Buffer for building up transmission Frames in parallel.
- Flag indicating which Input Buffer is currently accessible by the host.

20.3 Definitions

FlexRay™ Frame: Header Segment + Payload Segment

Message Buffer: Header Section + Data Section

Message RAM: Header Partition + Data Partition

Data Frame: FlexRay™ Frame that is not a NULL Frame

20.4 Block Diagram

The E-Ray is built up by the following main submodules:

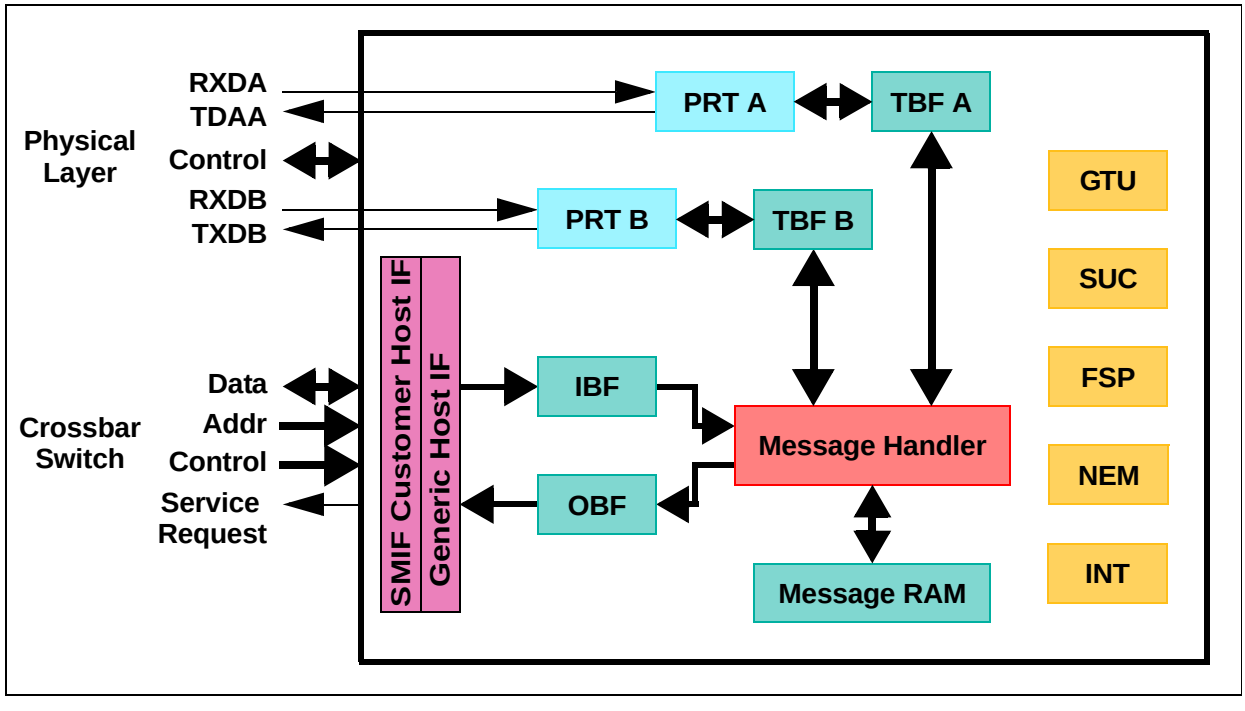


Figure 20-2 E-Ray Block Diagram

Customer Host Interface (CIF)

Connects the FPI Bus to the E-Ray IP-module via the Generic Host Interface.

Generic Host Interface (GIF)

The E-Ray IP-module is provided with an 8/16/32-bit Generic Host Interface prepared for the connection to a wide range of customer-specific Hosts. Configuration registers, status registers, and service request registers are attached to the respective blocks and can be accesssed via the Generic Host Interface.

Input Buffer (IBF)

For write access to the Message Buffers configured in the Message RAM, the Host can write the Header and Data Section for a specific Message Buffer to the Input Buffer. The Message Handler then transfers the data from the Input Buffer to the selected Message Buffer in the Message RAM.

Because the Input Buffer (IBF) Scheme does only allow to write the entire Message Frame, not only parts of it, the number of IBF has been increased from originally 2 to 4. This enables to fill the buffer partly and at the end request transfer into Message RAM. Therefore 2 extra bits allow to switch between the two banks of IBF and one status bit signals the IBF currently active for Host writes.

Output Buffer (OBF)

For read access to a Message Buffer configured in the Message RAM the Message Handler transfers the selected Message Buffer to the Output Buffer. After the transfer has completed, the Host can read the Header and Data Section of the transferred Message Buffer from the Output Buffer.

Message Handler (MHD)

The E-Ray Message Handler controls data transfers between the following components:

- Input / Output Buffer and Message RAM
- Transient Buffer RAMs of the two FlexRay™ Protocol Controllers and Message RAM

Message RAM (MRAM)

The Message RAM consists of a single-ported RAM that stores up to 128 FlexRay™ Message Buffers together with the related configuration data (Header and Data Partition).

Transient Buffer RAM (TBF 1/2)

Stores the Data Section of two complete messages.

FlexRay™ Channel Protocol Controller (PRT A/B)

The FlexRay™ Channel Protocol Controllers consist of shift register and FlexRay™ protocol FSM. They are connected to the Transient Buffer RAMs for intermediate message storage and to the physical layer via bus driver BD.

They perform the following functionality:

- Control and check of bit timing
- Reception and transmission of FlexRay™ Frames and symbols
- Check of Header CRC
- Generation / check of Frame CRC
- Interfacing to bus driver

The FlexRay™ Channel Protocol Controllers have interfaces to:

- Physical Layer (bus driver)
- Transient Buffer RAM
- Message Handler
- Global Time Unit
- System Universal Control
- Frame and Symbol Processing
- Network Management
- Service Request Control

Global Time Unit (GTU)

The Global Time Unit performs the following functions:

- Generation of Microtick
- Generation of Macrotick
- Fault tolerant clock synchronization by FTM algorithm
  - Rate correction
  - Offset correction
- Cycle counter
- Timing control of static segment
- Timing control of dynamic segment (minislotting)
- Support of external clock correction

System Universal Control (SUC)

The System Universal Control controls the following functions:

- Configuration
- Wakeup
- Startup
- Normal Operation
- Passive Operation
- Monitor Mode

Frame and Symbol Processing (FSP)

The Frame and Symbol Processing controls the following functions:

- Checks the correct timing of Frames and symbols
- Tests the syntactical and semantical correctness of received Frames
- Sets the slot status flags

Network Management (NEM)

Handles of the Network Management vector

Service Request Control (INT)

The Service Request Controller performs the following functions:

- Provides error and status service request flags
- Enables and disables service request sources
- Assignment of service request sources to one of the two module service request lines
- Enables and disables module service request lines
- Manages the two service request timers
- Stop watch time capturing

20.5 Programmer's Model

The programmer's model of the E-Ray module follows the principle of memory mapped peripheral. Some portion of the memory follows the principle of segmented/paged memory organization.

20.5.1 Register Map

The E-Ray module allocates an address space of 2 Kbyte (000_H to $7FF_H$). The registers are organized as 32-bit registers. 8/16-bit accesses are also supported. Host access to the Message RAM is done via the Input and Output Buffers. They buffer data to be transferred to and from the Message RAM under control of the Message Handler, avoiding conflicts between Host accesses and message reception / transmission. Addresses 0004_H - $000F_H$and $03C8_H$ - $03EC_H$ are reserved for customer specific purposes. All functions related to these addresses are located in the Customer Host Interface. The test registers located on address 0010_H and 0014_H are writeable only under the conditions described in **"Special Registers" on Page 20-24**.

The assignment of the Message Buffers is done according to the scheme shown in **Table 20-1** below. The number N of available Message Buffers depends on the payload length of the configured Message Buffers. The maximum number of Message Buffers is 128. The maximum payload length supported is 254 byte.

The Message Buffers are separated into three consecutive groups:

- Static Buffers: Transmit / Receive Buffers assigned to static segment

- Static and Dynamic Buffers: Transmit / Receive Buffers assigned to static or dynamic segment
- FIFO- Receive FIFO

The Message Buffer separation configuration can be changed only in “DEFAULT_CONFIG” or “CONFIG” state only by programming the Message RAM Configuration register (MRC).

The first group starts with Message Buffer 0 and consists of static Message Buffers only. Message Buffer 0 is dedicated to hold the startup / SYNC Frame or the single slot Frame, if node transmit one, as configured by SUCC1.TXST, SUCC1.TXSY, and SUCC1.TSM in the SUC Configuration Register 1 (SUCC1). In addition, Message Buffer 1 may be used for SYNC Frame transmission in case that SYNC Frames or single-slot Frames should have different payloads on the two channels. In this case bit MRC.SPLM has to be programmed to 1 and Message Buffers 0 and 1 have to be configured with the key slot ID and can be (re)configured in “DEFAULT_CONFIG” or “CONFIG” state only.

The second group consists of Message Buffers assigned to the static or to the dynamic segment. Message Buffers belonging to this group may be reconfigured during run time from dynamic to static or vice versa depending on the state of MRC.SEC.

The Message Buffers belonging to the third group are concatenated to a single receive FIFO.

Table 20-1 Assignment of Message Buffers

Message Buffer 0	⇓ Static Buffers	
Message Buffer 1		
…		
	⇓ Static + Dynamic Buffers	⇐ **FDB**
	⇓ FIFO	⇐ **FFB**
Message Buffer N-1		
Message Buffer N		⇐ **LCB**

20.5.2 E-Ray Kernel Registers

This chapter describes all registers of the E-Ray kernel.

Table 20-2 Registers Address Space E-Ray Kernel Register Address Space

Module	Base Address	End Address	Note
ERAY	$F0010000_H$	$F00107FF_H$	2Kbyte

Table 20-3 Registers OverviewE-Ray Kernel Registers

Register Short Name	Register Long Name	Offset Addr.[1)]	Access Mode		Reset Class	Description see
			Read	Write		
Customer Registers						
CLC	E-Ray Clock Control Register	0000_H	SV,U	SV,E	3	**Page 20-261**
CUST1	Busy and Input Buffer Control Register	0004_H	SV,U	SV,U	3	**Page 20-18**
ID	Module Identification Register	0008_H	SV,U	SV,U	3	**Page 20-17**
CUST3	Customer Interface Timeout Counter	$000C_H$	SV,U	SV,U	3	**Page 20-21**
Special Registers						
TEST1	Test Register 1	0010_H	SV,U	SV,U	3	**Page 20-24**
TEST2	Test Register 2	0014_H	SV,U	SV,U	3	**Page 20-29**
-	Reserved	0018_H	nBE	nBE	-	-
LCK	Lock Register	$001C_H$	SV,U	SV,U	3	**Page 20-32**
Service Request Registers						
EIR	Error Service Request Register	0020_H	SV,U	SV,U	3	**Page 20-34**
SIR	Status Service Request Register	0024_H	SV,U	SV,U	3	**Page 20-39**
EILS	Error Service Request Line Select	0028_H	SV,U	SV,U	3	**Page 20-44**
SILS	Status Service Request Line Select	$002C_H$	SV,U	SV,U	3	**Page 20-48**

Table 20-3 Registers OverviewE-Ray Kernel Registers (cont'd)

Register Short Name	Register Long Name	Offset Addr.[1)]	Access Mode		Reset Class	Description see
			Read	Write		
EIES	Error Service Request Enable Set	0030_H	SV,U	SV,U	3	Page 20-52
EIER	Error Service Request Enable Reset	0034_H	SV,U	SV,U	3	Page 20-57
SIES	Status Service Request Enable Set	0038_H	SV,U	SV,U	3	Page 20-62
SIER	Status Service Request Enable Reset	$003C_H$	SV,U	SV,U	3	Page 20-67
ILE	Service Request Line Enable	0040_H	SV,U	SV,U	3	Page 20-72
T0C	Timer 0 Configuration	0044_H	SV,U	SV,U	3	Page 20-73
T1C	Timer 1 Configuration	0048_H	SV,U	SV,U	3	Page 20-75
STPW1	Stop Watch Register 1	$004C_H$	SV,U	SV,U	3	Page 20-77
STPW2	Stop Watch Register 2	0050_H	SV,U	SV,U	3	Page 20-79
-	Reserved	0054_H - $007C_H$	nBE	nBE	-	-
Communication Controller Control Registers						
SUCC1	SUC Configuration Register 1	0080_H	SV,U	SV,U	3	Page 20-80
SUCC2	SUC Configuration Register 2	0084_H	SV,U	SV,U	3	Page 20-88
SUCC3	SUC Configuration Register 3	0088_H	SV,U	SV,U	3	Page 20-89
NEMC	NEM Configuration Register	$008C_H$	SV,U	SV,U	3	Page 20-90
PRTC1	PRT Configuration Register 1	0090_H	SV,U	SV,U	3	Page 20-91
PRTC2	PRT Configuration Register 2	0094_H	SV,U	SV,U	3	Page 20-93
MHDC	MHD Configuration Register	0098_H	SV,U	SV,U	3	Page 20-94
-	Reserved	$009C_H$	nBE	nBE	-	-

Table 20-3 Registers OverviewE-Ray Kernel Registers (cont'd)

Register Short Name	Register Long Name	Offset Addr.[1)]	Access Mode		Reset Class	Description see
			Read	Write		
GTUC01	GTU Configuration Register 1	$00A0_H$	SV,U	SV,U		Page 20-95
GTUC02	GTU Configuration Register 2	$00A4_H$	SV,U	SV,U	3	Page 20-96
GTUC03	GTU Configuration Register 3	$00A8_H$	SV,U	SV,U	3	Page 20-97
GTUC04	GTU Configuration Register 4	$00AC_H$	SV,U	SV,U	3	Page 20-98
GTUC05	GTU Configuration Register 5	$00B0_H$	SV,U	SV,U	3	Page 20-99
GTUC06	GTU Configuration Register 6	$00B4_H$	SV,U	SV,U	3	Page 20-100
GTUC07	GTU Configuration Register 7	$00B8_H$	SV,U	SV,U	3	Page 20-101
GTUC08	GTU Configuration Register 8	$00BC_H$	SV,U	SV,U	3	Page 20-102
GTUC09	GTU Configuration Register 9	$00C0_H$	SV,U	SV,U	3	Page 20-103
GTUC10	GTU Configuration Register 10	$00C4_H$	SV,U	SV,U	3	Page 20-104
GTUC11	GTU Configuration Register 11	$00C8_H$	SV,U	SV,U	3	Page 20-105
-	Reserved	$00CC_H$ - $00FC_H$	nBE	nBE	-	-
Communication Controller Status Registers						
CCSV	Communication Controller Status Vector	0100_H	SV,U	nBE	3	Page 20-107
CCEV	Communication Controller Error Vector	0104_H	SV,U	nBE	3	Page 20-112
-	Reserved	0108_H	nBE	nBE	-	-
-	Reserved	$010C_H$	nBE	nBE	-	-
SCV	Slot Counter Value	0110_H	SV,U	nBE	3	Page 20-113

Table 20-3 Registers OverviewE-Ray Kernel Registers (cont'd)

Register Short Name	Register Long Name	Offset Addr.[1)]	Access Mode		Reset Class	Description see
			Read	Write		
MTCCV	Macrotick and Cycle Counter Value	0114_H	SV,U	nBE	3	Page 20-114
RCV	Rate Correction Value	0118_H	SV,U	nBE	3	Page 20-115
OCV	Offset Correction Value	$011C_H$	SV,U	nBE	3	Page 20-116
SFS	SYNC Frame Status	0120_H	SV,U	nBE	3	Page 20-117
SWNIT	Symbol Window and Network Idle Time Status	0124_H	SV,U	nBE	3	Page 20-119
ACS	Aggregated Channel Status	0128_H	SV,U	SV,U	3	Page 20-122
-	Reserved	$012C_H$	nBE	nBE	-	-
ESIDnn	Even Sync ID Symbol Window nn	0130_H - 0168_H	SV,U	nBE	3	Page 20-125
-	Reserved	$016C_H$	SV,U	nBE	-	-
OSIDnn	Odd Sync ID Symbol Window nn	0170_H - $01A8_H$	SV,U	nBE	3	Page 20-127
-	Reserved	$01AC_H$	SV,U	nBE	-	-
NMVx	Network Management Vector [1…3]	$01B0_H$ - $01B8_H$	SV,U	nBE	3	Page 20-129
-	Reserved	$01BC_H$ - $02FC_H$	nBE	nBE	-	-
Message Buffer Control Registers						
MRC	Message RAM Configuration	0300_H	SV,U	SV,U	3	Page 20-130
FRF	FIFO Rejection Filter	0304_H	SV,U	SV,U	3	Page 20-133
FRFM	FIFO Rejection Filter Mask	0308_H	SV,U	SV,U	3	Page 20-135
FCL	FIFO Critical Level	$030C_H$	SV,U	SV,U	3	Page 20-136
Message Buffer Status Registers						
MHDS	Message Handler Status	0310_H	SV,U	SV,U	3	Page 20-137
LDTS	Last Dynamic Transmit Slot	0314_H	SV,U	SV,U	3	Page 20-140

Table 20-3 Registers OverviewE-Ray Kernel Registers (cont'd)

Register Short Name	Register Long Name	Offset Addr.[1)]	Access Mode		Reset Class	Description see
			Read	Write		
FSR	FIFO Status Register	0318_H	SV,U	SV,U	3	Page 20-141
MHDF	Message Handler Constraints Flags	$031C_H$	SV,U	nBE	3	Page 20-143
TXRQ1	Transmission Request Register 1	0320_H	SV,U	nBE	3	Page 20-145
TXRQ2	Transmission Request Register 2	0324_H	SV,U	nBE	3	Page 20-146
TXRQ3	Transmission Request Register 3	0328_H	SV,U	nBE	3	Page 20-147
TXRQ4	Transmission Request Register 4	$032C_H$	SV,U	nBE	3	Page 20-148
NDAT1	New Data Register 1	0330_H	SV,U	nBE	3	Page 20-149
NDAT2	New Data Register 2	0334_H	SV,U	nBE	3	Page 20-150
NDAT3	New Data Register 3	0338_H	SV,U	nBE	3	Page 20-151
NDAT4	New Data Register 4	$033C_H$	SV,U	nBE	3	Page 20-152
MBSC1	Message Buffer Status Changed 1	0340_H	SV,U	nBE	3	Page 20-153
MBSC2	Message Buffer Status Changed 2	0344_H	SV,U	nBE	3	Page 20-154
MBSC3	Message Buffer Status Changed 3	0348_H	SV,U	nBE	3	Page 20-155
MBSC4	Message Buffer Status Changed 4	$034C_H$	SV,U	nBE	3	Page 20-156
-	Reserved	0350_H - $03 A4_H$	nBE	nBE	-	-
NDIC1	New Data Interrupt Control 1	$03A8_H$	SV,U	SV,U	3	Page 20-264
NDIC2	New Data Interrupt Control 2	$03AC_H$	SV,U	SV,U	3	Page 20-265
NDIC3	New Data Interrupt Control 3	$03B0_H$	SV,U	SV,U	3	Page 20-266

Table 20-3 Registers OverviewE-Ray Kernel Registers (cont'd)

Register Short Name	Register Long Name	Offset Addr.[1)]	Access Mode		Reset Class	Description see
			Read	Write		
NDIC4	New Data Interrupt Control 4	$03B4_H$	SV,U	SV,U	3	Page 20-267
MSIC1	Message Buffer Status Changed Interrupt Control 1	$03B8_H$	SV,U	SV,U	3	Page 20-268
MSIC2	Message Buffer Status Changed Interrupt Control 2	$03BC_H$	SV,U	SV,U	3	Page 20-269
MSIC3	Message Buffer Status Changed Interrupt Control 3	$03C0_H$	SV,U	SV,U	3	Page 20-270
MSIC4	Message Buffer Status Changed Interrupt Control 4	$03C4_H$	SV,U	SV,U	3	Page 20-267
IBUSYSRC	Input Buffer Busy Service Request Control Register	$03C8_H$	SV,U	SV,U	3	Page 20-273
OBUSYSRC	Output Buffer Busy Service Request Control Register	$03CC_H$	SV,U	SV,U	3	Page 20-273
MBSC1SRC	Message Buffer Status Changed 1 Service Request Control Register	$03D0_H$	SV,U	SV,U	3	Page 20-273
MBSC0SRC	Message Buffer Status Changed 0 Service Request Control Register	$03D4_H$	SV,U	SV,U	3	Page 20-273
NDAT1SRC	New Data 1 Service Request Control Register	$03D8_H$	SV,U	SV,U	3	Page 20-273
NDAT0SRC	New Data 0 Service Request Control Register	$03DC_H$	SV,U	SV,U	3	Page 20-273
TINT1SRC	Timer Interrupt 1 Service Request Control Register	$03E0_H$	SV,U	SV,U	3	Page 20-273
TINT0SRC	Timer Interrupt 0 Service Request Control Register	$03E4_H$	SV,U	SV,U	3	Page 20-273

Table 20-3 Registers OverviewE-Ray Kernel Registers (cont'd)

Register Short Name	Register Long Name	Offset Addr.[1)]	Access Mode		Reset Class	Description see
			Read	Write		
INT1SRC	Interrupt 1 Service Request Control Register	$03E8_H$	SV,U	SV,U	3	Page 20-273
INT0SRC	Interrupt 0 Service Request Control Register	$03EC_H$	SV,U	SV,U	3	Page 20-273
Identification Registers						
CREL	Core Release Registers	$03F0_H$	SV,U	nBE	3	Page 20-157
ENDN	Endian Register	$03F4_H$	SV,U	nBE	3	Page 20-159
-	Reserved	$03F6_H$ - $03FC_H$	nBE	nBE	-	-
Input Buffer						
WRDSn	Write Data Section [1…64]	0400_H - $04FC_H$	SV,U	SV,U	3	Page 20-160
WRHS1	Write Header Section 1	0500_H	SV,U	SV,U	3	Page 20-161
WRHS2	Write Header Section 2	0504_H	SV,U	SV,U	3	Page 20-164
WRHS3	Write Header Section 3	0508_H	SV,U	SV,U	3	Page 20-165
	Reserved	$050C_H$	nBE	nBE	-	
IBCM	Input Buffer Command Mask	0510_H	SV,U	SV,U	3	Page 20-166
IBCR	Input Buffer Command Request	0514_H	SV,U	SV,U	3	Page 20-168
	Reserved	0518_H - $05FC_H$	nBE	nBE	-	
Output Buffer						
RDDSn	Read Data Section [1…64]	0600_H - $06FC_H$	SV,U	nBE	3	Page 20-170
RDHS1	Read Header Section 1	0700_H	SV,U	nBE	3	Page 20-171
RDHS2	Read Header Section 2	0704_H	SV,U	nBE	3	Page 20-173
RDHS3	Read Header Section 3	0708_H	SV,U	nBE	3	Page 20-175
MBS	Message Buffer Status	$070C_H$	SV,U	nBE	3	Page 20-177
OBCM	Output Buffer Command Mask	0710_H	SV,U	SV,U	3	Page 20-182

Table 20-3 Registers OverviewE-Ray Kernel Registers (cont'd)

Register Short Name	Register Long Name	Offset Addr.[1]	Access Mode		Reset Class	Description see
			Read	Write		
OBCR	Output Buffer Command Request	0714_H	SV,U	SV,U	3	Page 20-185
	Reserved	0718_H - $07FC_H$	nBE	nBE	-	-

1) The absolute register address is calculated as follows:
Module Base Address + Offset Address (shown in this column)

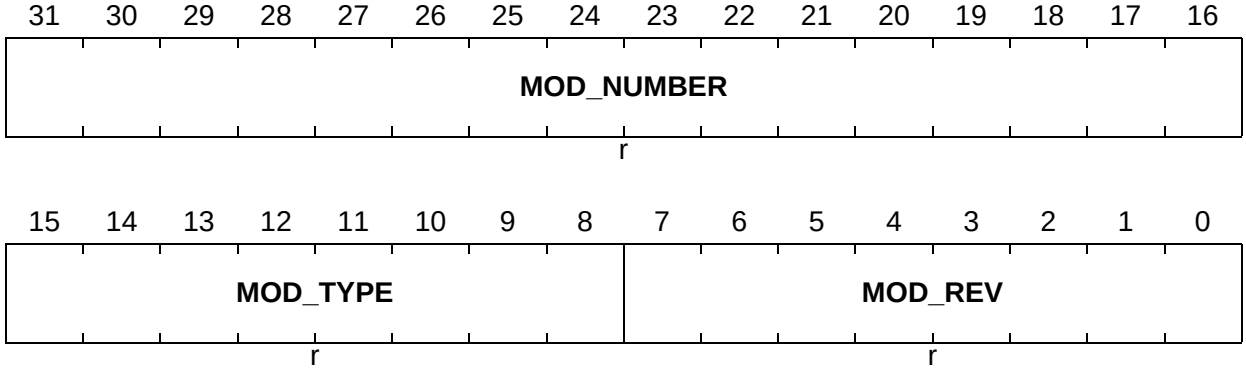
20.5.2.1 Customer Registers

The addresses 0004_H - $000F_H$and $03C8_H$ - $03EC_H$ are reserved for customer-specific registers.

Module Identification Register (ID)

This register contains bit fields identifying the E-Ray module in Infineons Module portfolio and is read only.

ID
Module Identification Register (0008$_H$) Reset Value: 0044 C0XX$_H$



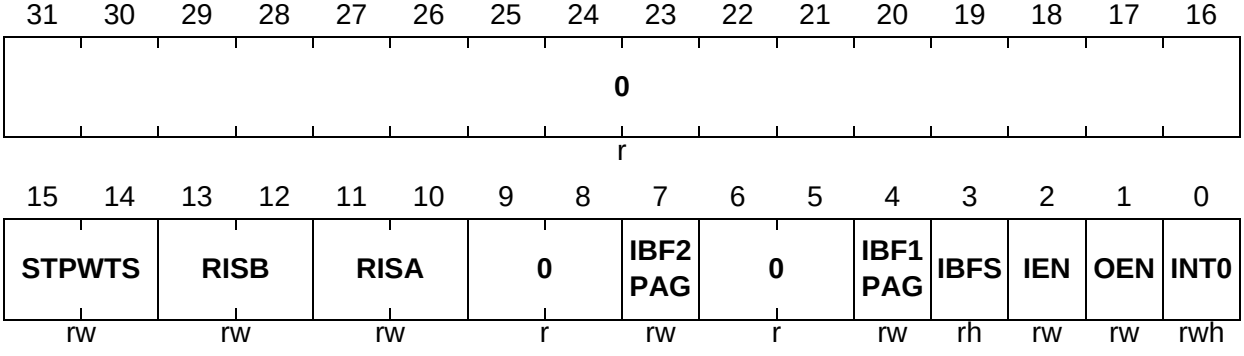
Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision Number** MOD_REV defines the module revision number. The value of a module revision starts with 01$_H$ (first revision).
MOD_TYPE	[15:8]	r	**Module Type** The value of this bit field is C0$_H$. It defines the module as a 32-bit module.
MOD_NUMBER	[31:16]	r	**Module Number Value** This bit field defines a module identification number. For the E-Ray module the module identification number is 44$_H$.

Busy Control Register (CUST1)

The Busy Control Register enables the automatic delay scheme. Furthermore it signals a timeout service request for the automatic delay scheme.

CUST1
Busy and Input Buffer Control Register

(0004_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
INT0	0	rwh	**CIF Timeout Service Request Status** INT0 will be set if a timeout has occurred during the auto delay scheme and must be reset by writing zero to INT0. *Note: In case hardware sets INT0 and at the same point of time software clears INT0, INT0 is cleared.*
OEN	1	rw	**Enable auto delay scheme for Output Buffer Control Register (OBCR)** This control bit controls the delay scheme for Output Buffer Control Register (OBCR) read accesses. 0_B Disable auto delay scheme for Output Buffer Control Register (OBCR) 1_B Enable auto delay scheme for Output Buffer Control Register (OBCR)
IEN	2	rw	**Enable auto delay scheme for Input Buffer Control Register (IBCR)** This control bit controls the auto delay scheme for Input Buffer Control Register (IBCR) read accesses. 0_B Disable auto delay scheme for Input Buffer Control Register (IBCR) 1_B Enable auto delay scheme for Input Buffer Control Register (IBCR)

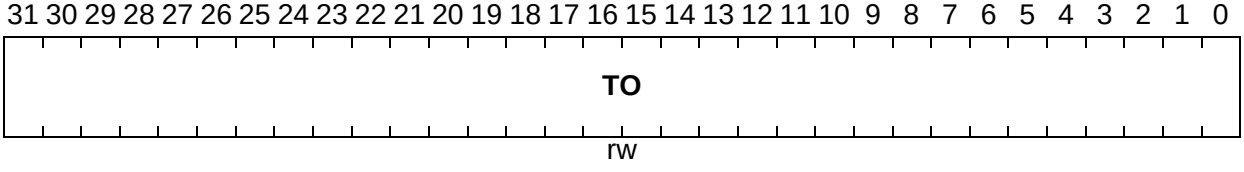
Field	Bits	Type	Description
IBFS	3	rh	**Input Buffer Status Register** This status bit indicates which of the two Input Buffer RAMs (IBF) is accessible by the host (via CIF) as Input Buffer. The other non accessible buffer RAM is currently used as shadow buffer RAM by the ERAY message handler and therefore not accessible by the host. 0_B Input Buffer RAM 2 (IBF2) is accessible as Input Buffer by the host (CIF) 1_B Input Buffer RAM 1 (IBF1) is accessible as Input Buffer by the host (CIF)
IBF1PAG	4	rw	**Input Buffer 1 Page Select Register** This control bit selects if the upper page or lower page of Input Buffer 1 (IBF1) currently active. Read: 0_B Lower Page (256 Bytes) of Input Buffer RAM 1 selected 1_B Upper Page (256 Bytes) of Input Buffer RAM 1 selected Write: 0_B Select Lower Page (256 Bytes) of Input Buffer RAM 1 1_B Select Upper Page (256 Bytes) of Input Buffer RAM 1 *Note: Write is only possible, if Input Buffer RAM 1 is currently accessible by the host (via CIF) and therefore IBFS set.*
IBF2PAG	7	rw	**Input Buffer 2 Page Select Register** This control bit selects if the upper page or lower page of Input Buffer 2 (IBF2) currently active. Read: 0_B Lower Page (256 Bytes) of Input Buffer RAM 2 selected 1_B Upper Page (256 Bytes) of Input Buffer RAM 2 selected Write: 0_B Select Lower Page (256 Bytes) of Input Buffer RAM 2 1_B Select Upper Page (256 Byte) of Input Buffer RAM 2 *Note: Write is only possible, if Input Buffer RAM 2 is currently accessible by the host (via CIF) and therefore IBFS cleared.*

Field	Bits	Type	Description
RISA	[11:10]	rw	**Receive Input Select Channel A** 00_B Channel A receiver input RXDA0 selected 01_B Channel A receiver input RXDA1 selected 10_B Channel A receiver input RXDA2 selected 11_B Channel A receiver input RXDA3 selected
RISB	[13:12]	rw	**Receive Input Select Channel B** 00_B Channel B receiver input RXDB0 selected 01_B Channel B receiver input RXDB1 selected 10_B Channel B receiver input RXDB2 selected 11_B Channel B receiver input RXDB3 selected
STPWTS	[15:14]	rw	**Stop Watch Trigger Input Select** 00_B Stop Watch Trigger input STPWT0 selected 01_B Stop Watch Trigger input STPWT1 selected 10_B Stop Watch Trigger input STPWT2 selected 11_B Stop Watch Trigger input STPWT3 selected
0	[6:5], [9:8], [31:16]	r	**Reserved** Returns 0 if read; should be written with 0.

Customer Interface Timeout Counter Register (CUST3)

The Timeout Counter Register is realizing the timeout counter reload (startup) value for the automatic delay scheme (not the timeout down counter itself).

CUST3
Customer Interface Timeout Counter ($000C_H$) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
TO	[31:0]	rw	**CIF Timeout Reload Value** The 32-bit down counter reload (start-up) value must be setup for the automatic delay scheme.

Automatic Delayed Write Access to OBCR and IBCR

Write and read accesses to the Output Buffer Control Register (OBCR) can be automatically stalled due to a ongoing transfer from the Message Buffer to the Output Buffer. Also write and read accesses to the Input Buffer Control Register (IBCR) may be automatically delayed due to a ongoing transfer from the Input Buffer to the Message Buffer.

This delay scheme can be controlled (enabled or disabled) by CUST1.IEN and CUST1.OEN. The maximum time to stall a write or read access is determined by a single timeout counter preloaded with the 32-bit value specified in the bit field CUST3.TO. If the timeout counter counts down to zero before the transfer to/from the Message Buffer is completed, the access (read or write) will be canceled and a service request will be generated. A canceled read access provides a 0 value. A canceled write access does not modify any bits in the OBCR or IBCR. In addition the bit CUST1.INT0 of the service request status register will be set and must be reset by the host to disable the service request line.

The read and write access to the Output Buffer Control Register (OBCR) may be configured without automatic delay by clearing CUST1.OEN. Setting OBCR.REQ and immediately afterwards reading or writing OBCR, e.g. to set OBCR.VIEW will lead to a canceled read or write operation, e.g. OBCR.VIEW remains cleared, and an error is signalled by a set EIR.IOBA. Besides canceling the erroneous read or write operation, and setting the error bit, no further state change happens. So full operation is granted. OBCR remains read and write inaccessible until the transfer of data from the Message Buffer to the Output Buffer (MBF⇒OBF) is completed. During this time span all read and

write accesses to the Output Buffer Control Register (OBCR) are canceled. The transfer is completed when OBCR.OBSYS is cleared. Additionally signal TOBC may be used, e.g. for service request triggering, DMA triggering, or driving a pin, to communicate the access status.

The read and write access to the Output Buffer Control Register (OBCR) may be configured to be automatic delayed by setting CUST1.OEN and configuring CUST3.TO to the maximum stall time acceptable to the system. If setting OBCR.REQ and immediately afterwards reading or writing to OBCR, e.g. to set the OBCR.VIEW bit, this read or write will be stalled until either the maximum delay time elapsed (in this case the read or write operation is cancelled after the stall time, e.g. OBCR.VIEW remains cleared, and an error is signalled by setting EIR.IOBA) or the read or write completes normally, e.g. set OBCR.VIEW after the transfer of data from the Message Buffer to the Output Buffer (MBF⇒OBF) is finalized. During this time the bus is locked and no further access to the E-Ray module is possible due to the ongoing stalled read or write operation. Because no access is possible to the E-Ray module, read or write stall may only be detected through the signal TOBC or due to other not processed read or write accesses to the E-Ray module.

The read and write access to the Input Buffer Control Register (IBCR) may also be configured without automatic delay by clearing CUST1.IEN. By writing to IBCR.IBRH the Input Buffers are swapped (shadow IBF changes to host IBF and host IBF to shadow IBF), the content of the shadow IBF is copied into the MBF (IBF⇒MBF), and IBCR.IBSYS is set. Writing to IBCR.IBRH a second time while IBCR.IBSYS remained set (previously initiated copy process IBF⇒MBF ongoing) will correctly update IBCR.IBRH and set IBCR.IBSYH. This will set the signal IBUSY. A third access, read or write, to IBCR while IBCR.IBSYH remains set will cancel this third access and an error is signalled by setting EIR.IIBA. Besides canceling this last access to IBCR and setting the error bit, no further state change happens. So full operation is granted. IBCR remains read and write inaccessible until the transfer of data from the Input Shadow Buffer to the Message Buffer (IBF⇒MBF) is completed and once more the Input Buffers are swapped (shadow IBF changes to host IBF and host IBF to shadow IBF). During this time span all read and write accesses to the Input Buffer Control Register (IBCR) are canceled. The transfer is completed when IBCR.IBSYH is cleared. Additionally signal TIBC may be used, e.g. for service request triggering, DMA triggering, or driving a pin, to communicate the access status.

The read and write access to the Input Buffer Control Register (IBCR) may be configured for being automatically delayed by setting CUST1.IEN and configuring CUST3.TO to the maximum stall time acceptable to the system. By writing to IBCR.IBRH the Input Buffers are swapped (shadow IBF changes to host IBF and host IBF to shadow IBF), the content of the shadow IBF copied into the MBF, and IBCR.IBSYS is set. Writing to IBCR.IBRH a second time while IBCR.IBSYS remains set (previously initiated copy process ongoing) will correctly update IBCR.IBRH and set IBCR.IBSYH. A third access to IBCR while IBCR.IBSYH remains set will stall this read or write until either the maximum delay

time elapsed (in this case the read or write operation is cancelled after the stall time and an error is signalled by setting EIR.IOBA) or the read or write completes normally, after the transfer of data from the Input Shadow Buffer to the Message Buffer (IBF⇒MBF) is finalized and once more the Input Buffers are swapped (shadow IBF changes to host IBF and host IBF to shadow IBF). During this time the bus is locked and no further access to E-Ray module is possible due to the ongoing stalled read or write operation. Because no access is possible to the E-Ray module, read or write stall may only be detected through the signal TIBC or due to other not processed read or write accesses to the E-Ray module.

So setting CUST3.TO = $FFFFFFFF_H$, CUST1.IEN = 1, and CUST1.OEN = 1 will always grant a consistent data access of the host to the Output and Input Buffers without the need of reading and taking into account the status of OBCR.OBSYS or IBCR.IBSYH. But this simplified access may cause system latencies and system performance loss.

20.5.2.2 Special Registers

Test Register 1 (TEST1)

The Test Register 1 holds the control bits to configure the test modes of the E-Ray module. Write access to these bits is only possible if bit TEST1.WRTEN is set.

The Test Register 1 bits therefore can be used to test the interface to the physical layer (connectivity test) by driving / reading the respective pins.

When the E-Ray IP is operated in one of its test modes that requires TEST1.WRTEN to be set (RAM Test Mode, I/O Test Mode, Asynchronous Transmit Mode, and Loop Back Mode) only the selected test mode functionality is available.

The test functions are not available in addition to the normal operational mode functions, they change the functions of parts of the E-Ray module. Therefore normal operation as specified outside this chapter and as required by the FlexRay™ protocol specification and the FlexRay™ conformance test is not possible. Test mode functions may not be combined with each other or with FlexRay™ protocol functions.

The test mode features are intended for hardware testing or for FlexRay™ bus analyzer tools. They are not intended to be used in FlexRay™ applications

TEST1
Test Register 1 **(0010_H)** **Reset Value: $0000\ 0300_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CERB				CERA				0		TXENB	TXENA	TXB	TXA	RXB	RXA
rh				rh				r		rwh	rwh	rwh	rwh	rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						AOB	AOA	0		TMC		0		ELBE	WRTEN
r						rh	rh	r		rw		r		rw	rw

Field	Bits	Type	Description
WRTEN	0	rw	**Write Test Register Enable** Enables write access to the test registers. To set the bit from 0 to 1 the test mode key has to be written as defined on **"Lock Register (LCK)" on Page 20-32**. The unlock sequence is not required when TEST1.WRTEN is kept at 1 while other bits of the register are changed. The bit can be reset to 0 at any time. 0_B Write access to test registers disabled. 1_B Write access to test registers enabled.

Field	Bits	Type	Description
ELBE	1	rw	**External Loop Back Enable** There are two possibilities to perform a loop back test. External loop back via physical layer or internal loop back for in-system self-test (default). In case of an internal loop back pins TXENA and TXENB are in their inactive state, pins TXDA and TXDB are set to HIGH, pins RXDA and RXDB are not evaluated. Bit ELBE is evaluated only when POC is in loop back mode and test multiplexer control is in non multiplexed mode TMC = 00. 0_B Internal loop back (default) 1_B External loop back
TMC	[5:4]	rw	**Test Multiplexer Control** 00_B Normal signal path (default). 01_B RAM Test Mode: Internal busses are multiplexed to make all RAM blocks of the E-Ray module directly accessible by the Host. This mode is intended to enable testing of the embedded RAM blocks during production testing. 10_B I/O Test Mode: Output pins are driven to the values defined by bits TXA, TXB, TXENA, TXENB. The values applied to the input pins can be read from register bits RXA and RXB. 11_B Reserved; should not be used.
AOA	8	rh	**Activity on A** The channel idle condition is specified in the FlexRay™ protocol spec v2.1, chapter 3, BITSTRB process (zChannelIdle). 0_B No activity detected, channel A idle 1_B Activity detected, channel A not idle
AOB	9	rh	**Activity on B** The channel idle condition is specified in the FlexRay™ protocol spec v2.1, chapter 3, BITSTRB process (zChannelIdle). 0_B No activity detected, channel B idle 1_B Activity detected, channel B not idle
RXA	16	rh	**Read Channel A Receive Pin** 0_B RXDA = 0 1_B RXDA = 1

Field	Bits	Type	Description
RXB	17	rh	**Read Channel B Receive Pin** 0_B RXDB = 0 1_B RXDB = 1
TXA	18	rwh	**Read or Write to Channel A Transmit Pin** 0_B TXDA = 0 1_B TXDA = 1
TXB	19	rwh	**Read or Write to Channel B Transmit Pin** 0_B TXDB = 0 1_B TXDB = 1
TXENA	20	rwh	**Read or Write to Channel A Transmit Enable Pin** 0_B $\overline{\text{TXENA}}$= 0 1_B $\overline{\text{TXENA}}$= 1
TXENB	21	rwh	**Read or Write to Channel B Transmit Enable Pin** 0_B $\overline{\text{TXENB}}$ = 0 1_B $\overline{\text{TXENB}}$= 1
CERA	[27:24]	rh	**Coding Error Report Channel A[1)]** Set when a coding error is detected on channel A. Reset to zero when register TEST1 is read or written. Once the CERA is set it will remain unchanged until the Host accesses the TEST1 register. 0000_B No coding error detected 0001_B Header CRC error detected 0010_B Frame CRC error detected 0011_B Frame Start Sequence FSS too long 0100_B First bit of Byte Start Sequence BSS seen LOW 0101_B Second bit of Byte Start Sequence BSS seen HIGH 0110_B First bit of Frame End Sequence FES seen HIGH 0111_B Second bit of Frame End Sequence FES seen LOW 1000_B CAS / MTS symbol seen too short 1001_B CAS / MTS symbol seen too long Other combinations are reserved.

Field	Bits	Type	Description
CERB	[31:28]	rh	**Coding Error Report Channel B**[1)] Set when a coding error is detected on channel B. Reset to zero when register TEST1 is read or written. Once the CERB is set it will remain unchanged until the Host accesses the TEST1 register. 0000_B No coding error detected 0001_B Header CRC error detected 0010_B Frame CRC error detected 0011_B Frame Start Sequence FSS too long 0100_B First bit of Byte Start Sequence BSS seen LOW 0101_B Second bit of Byte Start Sequence BSS seen HIGH 0110_B First bit of Frame End Sequence FES seen HIGH 0111_B Second bit of Frame End Sequence FES seen LOW 1000_B CAS / MTS symbol seen too short 1001_B CAS / MTS symbol seen too long Other combinations are reserved.
0	[3:2], [7:6], [15:10], [23:22]	r	**Reserved** Returns 0 if read; should be written with 0.

1) Coding errors are also signalled when the Communication Controller is in “MONITOR_MODE”. The error codes regarding CAS / MTS symbols concern only the monitored bit pattern, irrelevant whether those bit patterns are seen in the symbol window or elsewhere.

Asynchronous Transmit Mode (ATM)

The asynchronous transmit mode is entered by writing 1110_B to the CHI Command Vector SUCC1.CMD in the SUC Configuration Register 1 (CHI command: ATM) while the Communication Controller is in “CONFIG” state and bit TEST1.WRTEN in the Test Register 1 is set. This write operation has to be directly preceded by two consecutive write accesses to the Configuration Lock Key (unlock sequence). When called in any other state or when bit TEST1.WRTEN is not set, SUCC1.CMD will be reset to 0000_B = “COMMAND_NOT_ACCEPTED”. CCSV.POCS in the Communication Controller Status Vector will return 1110_B while the E-Ray module is in ATM mode. Asynchronous Transmit mode can be left by writing 0001_B (CHI command: “CONFIG”) to the CHI Command Vector SUCC1.CMD in the SUC Configuration Register 1.

In ATM mode transmission of a FlexRay™ Frame is triggered by writing the number of the respective Message Buffer to the Input Buffer Command Request register (IBCR.IBRH) while bit IBCM.STXRS in the Input Buffer Command Mask register is set to 1. In this mode wakeup, startup, and clock synchronization are bypassed. The CHI command SEND_MTS results in the immediate transmission of an MTS symbol.

The cycle counter value of Frames send in ATM mode can be programmed via MTCCV.CCV (writeable in ATM and loop back mode only).

Loop Back Mode

The loop back mode is entered by writing 1111_B to the CHI Command Vector SUCC1.CMD in the SUC Configuration Register 1 (CHI command: LOOP_BACK) while the Communication Controller is in “CONFIG” state and bit TEST1.WRTEN in the Test Register 1 is set. This write operation has to be directly preceded by two consecutive write accesses to the Configuration Lock Key (unlock sequence). When called in any other state or when bit TEST1.WRTEN is not set, SUCC1.CMD will be reset to 0000_B = “COMMAND_NOT_ACCEPTED”. CCSV.POCS in the Communication Controller Status Vector will show 0000 1101_H while the E-Ray module is in loop back mode.

Loop Back mode can be left by writing 0001_B (CHI command: “CONFIG”) to the CHI Command Vector SUCC1.CMD in the SUC Configuration Register 1.

The loop back test mode is intended to check the module’s internal data paths. Normal, time triggered operation is not possible in loop back mode.

There are two possibilities to perform a loop back test. External loop back via physical layer (TEST1.ELBE = 1) or internal loop back for in-system self-test (TEST1.ELBE = 0). In case of an internal loop back pins TXENA, TXENB are in their inactive state, pins TXDA and TXDB are set to HIGH, pins RXDAn and RXDBn are not evaluated.

When the Communication Controller is in loop back mode, a loop back test is started by the Host writing a message to the Input Buffer and requesting the transmission by writing to the Input Buffer Command Request register IBCR. The Message Handler will transfer the message into the Message RAM and then into the Transient Buffer of the selected channel. The Channel Protocol Controller (PRT) will read (in 32-bit words) the message from the transmit part of the Transient Buffer and load it into its Rx / Tx shift register. The serial transmission is looped back into the shift register; its content is written into the receive part of the channels’s Transient Buffer before the next word is loaded.

The PRT and the Message Handler will then treat this transmitted message like a received message, perform an acceptance filtering on Frame ID and receive channel, and store the message into the Message RAM if it passed acceptance filtering. The loop back test ends with the Host requesting this received message from the Message RAM and then checking the contents of the Output Buffer.

Each FlexRay™ channel is tested separately. The E-Ray cannot receive messages from the FlexRay™ bus while it is in the loop back mode.

The cycle counter value of Frames used in loop back mode can be programmed via MTCCV.CCV (writeable in ATM and loop back mode only).

Note that in case of an odd payload the last two bytes of the looped-back payload will be shifted by 16 bits to the right inside the last 32-bit data word.

Test Register 2 (TEST2)

The Test Register 2 holds all bits required for the RAM test of the seven embedded RAM blocks of the E-Ray module. Write access to this register is only possible when TEST1.WRTEN in the Test Register 1 is set to 1.

TEST2
Test Register 2 **(0014_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RDPB	WRPB	0							SSEL			0	RS		
rh	rw	r							rw			r	rw		

Field	Bits	Type	Description
RS	[2:0]	rw	**RAM Select** In RAM Test mode the RAM blocks selected by RS are mapped to module address $0000\ 0400_H$ to $0000\ 07FF_H$ (1024 byte addresses). 000_B Input Buffer RAM 1 (IBF1) 001_B Input Buffer RAM 2 (IBF2) 010_B Output Buffer RAM 1 (OBF1) 011_B Output Buffer RAM 2 (OBF2) 100_B Transient Buffer RAM A (TBF1) 101_B Transient Buffer RAM B (TBF2) 110_B Message RAM (MBF) 111_B Reserved; should not be used.

Field	Bits	Type	Description
SSEL	[6:4]	rw	**Segment Select** To enable access to the complete Message RAM (8192 byte addresses) the Message RAM is segmented. 000_B access to RAM byte 0000_H to $03FF_H$ enabled 001_B access to RAM byte 0400_H to $07FF_H$ enabled 010_B access to RAM byte 0800_H to $0BFF_H$ enabled 011_B access to RAM byte $0C00_H$ to $0FFF_H$ enabled 100_B access to RAM byte 1000_H to $11FF_H$ enabled 101_B access to RAM byte 1400_H to $17FF_H$ enabled 110_B access to RAM byte 1800_H to $1BFF_H$ enabled 111_B access to RAM byte $1C00_H$ to $1FFF_H$ enabled
WRPB	14	rw	**Write Parity Bit** Value of parity bit to be written to bit 32 of the addressed RAM word.
RDPB	15	rh	**Read Parity Bit** Value of parity bit read from bit 32 of the addressed RAM word.
0	3, [13:7], [31:16]	r	**Reserved** Returns 0 if read; should be written with 0.

RAM Test Mode

In RAM test mode (TEST1.TMC = 1), one of the seven RAM blocks can be selected for direct RD/WR access by programming TEST2.RS.

For external access the selected RAM block is mapped to address space 400_H to $7FF_H$ (1024 byte addresses or 256 word addresses).

Because the length of the Message RAM exceeds the available address space, the Message RAM is segmented into segments of 1024 byte. The segments can be selected by programming TEST2.SSEL in the Test Register 2.

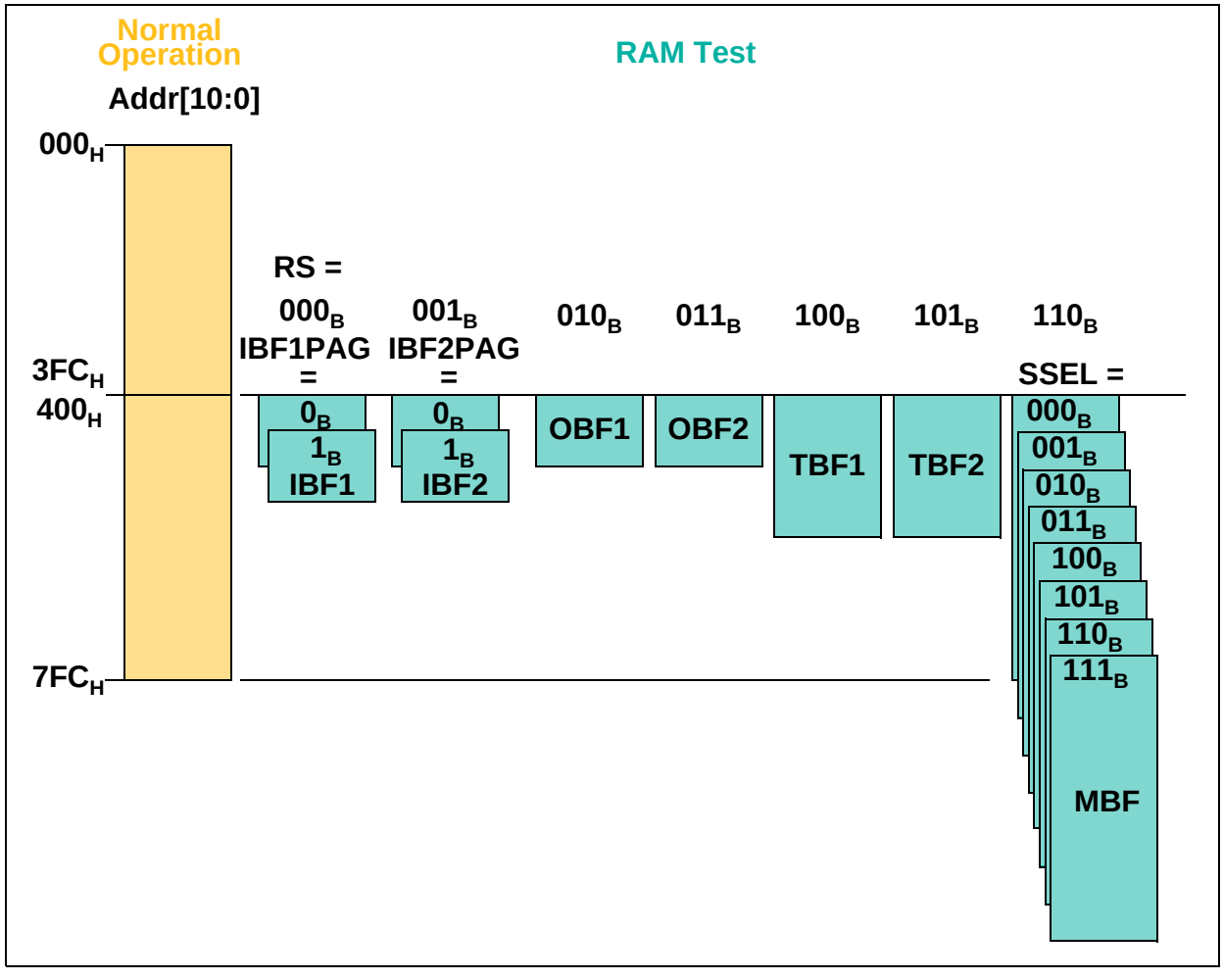
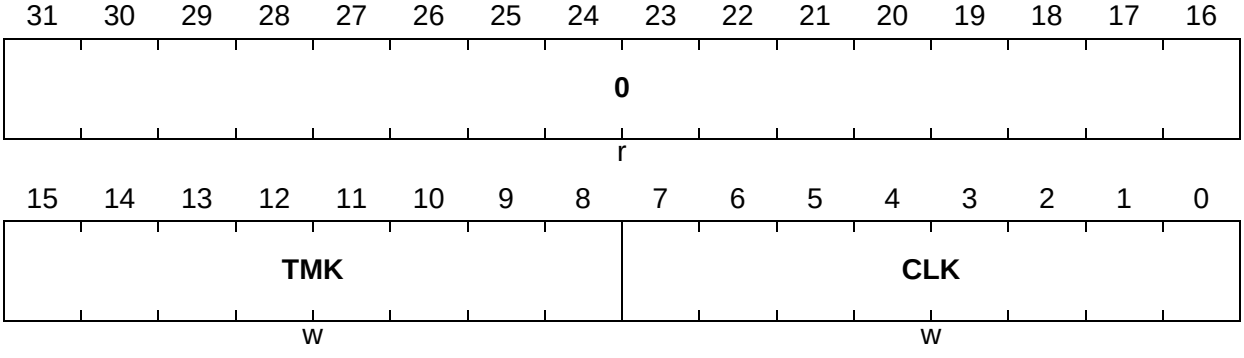


Figure 20-3 RAM test mode Access to E-Ray RAM Blocks

Lock Register (LCK)

The Lock Register is write-only. Reading the register will return 0000 0000$_H$.

LCK
Lock Register **(001C$_H$)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
CLK	[7:0]	w	**Configuration Lock Key** To leave “CONFIG” state by writing to SUCC1.CMD commands READY, MONITOR_MODE, ATM, LOOP_BACK) in the SUC Configuration Register 1, the write operation has to be directly preceded by two consecutive write accesses to the Configuration Lock Key (unlock sequence). If the write sequence below is interrupted by other write accesses between the second write to the Configuration Lock Key and the write access to the SUCC1 register, the Communication Controller remains in “CONFIG” state and the sequence has to be repeated. First write: LCK.CLK = CE$_H$ = 1100 1110$_B$ Second write: LCK.CLK = 31$_H$ = 0011 0001$_B$ Third write: SUCC1.CMD Returns 0 if read

Field	Bits	Type	Description
TMK	[15:8]	w	**Test Mode Key** To set bit TEST1.WRTEN the write operation has to be directly preceded by two consecutive write accesses to the Test Mode Key. If the write sequence is interrupted by other write accesses between the second write to the Test Mode Key and the write access to the Test1 register, bit TEST1.WRTEN is not set to 1 and the sequence has to be repeated. First write: LCK.TMK = 75_H = 0111 0101_B Second write: LCK.TMK = $8A_H$ = 1000 1010_B Second write: TEST1.WRTEN = 1 Returns 0 if read
0	[31:16]	r	**Reserved** Returns 0 if read; should be written with 0.

Note: In case the Host uses 8/16-bit accesses to write the listed bit fields, the programmer has to ensure that no “dummy accesses” e.g. to the remaining register bytes / words are inserted by the compiler.

To exit “CONFIG” state by writing to SUCC1.CMD in the SUC Configuration Register 1, the write operation has to be directly preceded by two consecutive write accesses to the Configuration Lock Key. If this write sequence is service requested by read accesses or write accesses to other locations, the Communication Controller remains in “CONFIG” state and the sequence has to be repeated.

First write: LCK.CLK = CE_H = 1100 1110_B

Second write: LCK.CLK = 31_H = 0011 0001_B

20.5.2.3 Service Request Registers

The address space from 0020_H to $007F_H$ is reserved for service request registers.

Error Service Request Select (EIR)

The flags are set when the Communication Controller detects one of the listed error conditions. They remain set until the Host clears them. A flag is cleared by writing a 1 to the corresponding bit position. Writing a 0 has no effect on the flag. An application reset will also clear the register.

EIR
Error Service Request Register (0020_H) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0					TABB	LTVB	EDB	0					TABA	LTVA	EDA
r					rwh	rwh	rwh	r					rwh	rwh	rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				MHF	IOBA	IIBA	EFA	RFO	PERR	CCL	CCF	SFO	SFBM	CNA	PEMC
r				rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
PEMC	0	rwh	**POC Error Mode Changed** This flag is set whenever the error mode signalled by CCEV.ERRM in the Communication Controller Error Vector register has changed. 0_B Error mode has not changed 1_B Error mode has changed This flag is cleared by writing a 1.
CNA	1	rwh	**Command Not Accepted** The flag signals that the write access to the CHI command vector SUCC1.CMD in the SUC Configuration Register 1 was not successful because the requested command was not valid in the actual POC state, or because the CHI command was locked (CCL = 1). 0_B CHI command accepted 1_B CHI command not accepted This flag is cleared by writing a 1.

Field	Bits	Type	Description
SFBM	2	rwh	**SYNC Frames Below Minimum** This flag signals that the number of SYNC Frames received during the last communication cycle was below the limit required by the FlexRay™ protocol. May be set during startup and therefore should be cleared by the Host after the Communication Controller entered “NORMAL_ACTIVE” state. 0_B Sync node: 1 or more SYNC Frames received Non-sync node: 2 or more SYNC Frames received 1_B Less than the required minimum of SYNC Frames received This flag is cleared by writing a 1.
SFO	3	rwh	**SYNC Frame Overflow** Set when either the number of SYNC Frames received during the last communication cycle or the total number of SYNC Frames received during the last double cycle exceeds the maximum number of SYNC Frames as defined by GTUC02.SNM in the GTU Configuration Register 2. 0_B Number of received SYNC Frames ≤ GTUC02.SNM 1_B More SYNC Frames received than configured by GTUC02.SNM This flag is cleared by writing a 1.
CCF	4	rwh	**Clock Correction Failure** This flag is set at the end of the cycle whenever one of the following errors occurred: • Missing offset and / or rate correction • Clock Correction limit reached The clock correction status is monitored in registers CCEV and SFS. A failure may occur during startup, therefore bit CCF should be cleared by the Host after the Communication Controller entered “NORMAL_ACTIVE” state. 0_B Clock correction successful so far 1_B Clock correction failed This flag is cleared by writing a 1.

Field	Bits	Type	Description
CCL	5	rwh	**CHI Command Locked** The flag signals that the write access to the CHI command vector SUCC1.CMD was not successful because the execution of the previous CHI command has not yet completed. In this case bit EIR.CNA is also set to 1. 0_B CHI command accepted 1_B CHI command not accepted This flag is cleared by writing a 1.
PERR	6	rh	**Parity Error** The flag signals a parity error to the Host. It is set whenever one of the flags MHDS.PIBF, MHDS.POBF, MHDS.PMR, MHDS.PTBF1, MHDS.PTBF2 changes from 0 to 1. See also **“Message Handler Status (MHDS)” on Page 20-137**. 0_B No parity error detected 1_B Parity error detected
RFO	7	rh	**Receive FIFO Overrun** The flag is set by the Communication Controller when a receive FIFO overrun is detected. When a receive FIFO overrun occurs, the oldest message is overwritten with the actual received message. The actual state of the FIFO is monitored in register FSR. 0_B No receive FIFO overrun detected 1_B A receive FIFO overrun has been detected
EFA	8	rwh	**Empty FIFO Access** This flag is set by the Communication Controller when the Host requests the transfer of a message from the receive FIFO via Output Buffer while the receive FIFO is empty. 0_B No Host access to empty FIFO occurred 1_B Host access to empty FIFO occurred

Field	Bits	Type	Description
IIBA	9	rwh	**Illegal Input Buffer Access** This flag is set by the Communication Controller when the Host wants to modify a Message Buffer via Input Buffer while the Communication Controller is not in “CONFIG” or “DEFAULT_CONFIG” state and one of the following conditions applies: 1. The Host writes to the Input Buffer Command Request register to modify the: a) Header Section of Message Buffer 0, 1 if configured for transmission in key slot b) Header Section of static Message Buffers with buffer number < MRC.FDB while MRC.SEC = 01_B c) Header Section of any static or dynamic Message Buffer while MRC.SEC = $1x_B$ d) Header and / or Data Section of any message buffer belonging to the receive FIFO 2. The Host writes to any register of the Input Buffer while IBCR.IBSYS is set. 0_B No illegal Host access to Input Buffer occurred 1_B Illegal Host access to Input Buffer occurred
IOBA	10	rwh	**Illegal Output Buffer Access** This flag is set by the Communication Controller when the Host requests the transfer of a Message Buffer from the Message RAM to the Output Buffer while OBCR.OBSYS is set to 1. 0_B No illegal Host access to Output Buffer occurred 1_B Illegal Host access to Output Buffer occurred
MHF	11	rwh	**Message Handler Constraints Flag** The flag signals a Message Handler constraints violation condition. It is set whenever one of the flags MHDF.SNUA, MHDF.SNUB, MHDF.FNFA, MHDF.FNFB, MHDF.TBFA, MHDF.TBFB, MHDF.WAHP changes from 0 to 1. 0_B No Message Handler failure detected 1_B Message Handler failure detected
EDA	16	rwh	**Error Detected on Channel A** This bit is set whenever one of the flags ACS.SEDA, ACS.CEDA, ACS.CIA, ACS.SBVA changes from 0 to 1. 0_B No error detected on channel A 1_B Error detected on channel A This flag is cleared by writing a 1.

Field	Bits	Type	Description
LTVA	17	rwh	**Latest Transmit Violation Channel A** The flag signals a latest transmit violation on channel A to the Host. 0_B No latest transmit violation detected on channel A 1_B Latest transmit violation detected on channel A This flag is cleared by writing a 1.
TABA	18	rwh	**Transmission Across Boundary Channel A** The flag signals to the Host that a transmission across a slot boundary occurred for channel A. 0_B No transmission across slot boundary detected on channel A 1_B Transmission across slot boundary detected on channel A This flag is cleared by writing a 1.
EDB	24	rwh	**Error Detected on Channel B** This bit is set whenever one of the flags ACS.SEDB, ACS.CEDB, ACS.CIB, ACS.SBVB changes from 0 to 1. 0_B No error detected on channel B 1_B Error detected on channel B This flag is cleared by writing a 1.
LTVB	25	rwh	**Latest Transmit Violation Channel B** The flag signals a latest transmit violation on channel B to the Host. 0_B No latest transmit violation detected on channel B 1_B Latest transmit violation detected on channel B This flag is cleared by writing a 1.
TABB	26	rwh	**Transmission Across Boundary Channel B** The flag signals to the Host that a transmission across a slot boundary occurred for channel B. 0_B No transmission across slot boundary detected on channel B 1_B Transmission across slot boundary detected on channel B This flag is cleared by writing a 1.
0	[15:12], [23:19], [31:27]	r	**Reserved** Returns 0 if read; should be written with 0.

Status Service Request Register (SIR)

The flags are set whenever the Communication Controller detects one of the listed events. The flags remain set until the Host clears them. A flag is cleared by writing a 1 to the corresponding bit position. Writing a 0 has no effect on the flag. An application reset will also clear the register.

SIR
Status Service Request Register (0024$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0						MTSB	WUPB	0						MTSA	WUPA
r						rwh	rwh	r						rwh	rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SDS	MBSI	SUCS	SWE	TOBC	TIBC	TI1	TI0	NMVC	RFCL	RFNE	RXI	TXI	CYCS	CAS	WST
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rh	rh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
WST	0	rwh	**Wakeup Status** This flag is set when the wakeup status vector CCSV.WSV in the Communication Controller Status Vector register changes. 0_B Wakeup status unmodified 1_B Wakeup status modified This flag is cleared by writing a 1.
CAS	1	rwh	**Collision Avoidance Symbol** This flag is set by the Communication Controller during STARTUP state when a CAS or potential CAS was received. 0_B No bit pattern matching the CAS symbol received 1_B Bit pattern matching the CAS symbol received This flag is cleared by writing a 1.
CYCS	2	rwh	**Cycle Start Service Request** This flag is set by the Communication Controller when a communication cycle starts 0_B No communication cycle started 1_B Communication cycle started This flag is cleared by writing a 1.

Field	Bits	Type	Description
TXI	3	rwh	**Transmit Service Request** This flag is set by the Communication Controller at the end of Frame transmission if bit WRHS1.MBI in the respective Message Buffer is set (see **Table 20-24**). 0_B No Frame transmitted from a transmit buffer with WRHS1.MBI = 1 1_B At least one Frame was transmitted from a transmit buffer with WRHS1.MBI = 1 This flag is cleared by writing a 1.
RXI	4	rwh	**Receive Service Request** This flag is set by the Communication Controller whenever the set condition of a Message Buffer ND flag is fulfilled and if bit WRHS1.MBI of that Message Buffer is set to 1(see **Table 20-24**). 0_B No ND flag of a receive buffer with WRHS1.MBI = 1 has been set to 1 1_B At least one ND flag of a receive buffer with WRHS1.MBI = 1 has been set to 1 This flag is cleared by writing a 1.
RFNE	5	rh	**Receive FIFO Not Empty** This flag is set by the Communication Controller when a received valid Frame was stored into the empty receive FIFO.m The actual state of the receive FIFO is monitored in register FSR 0_B Receive FIFO is empty 1_B Receive FIFO is not empty
RFCL	6	rh	**Receive FIFO Critical Level** This flag is set when a valid receive FIFO fill level FSR.RFFL is equal or greater than the critical level as configured by FCL.CL. 0_B Receive FIFO below critical level 1_B Receive FIFO critical level reached
NMVC	7	rwh	**Network Management Vector Changed** This service request flag signals a change in the Network Management Vector visible to the Host. 0_B No change in the Network Management vector 1_B Network Management vector changed This flag is cleared by writing a 1.

Field	Bits	Type	Description
TI0	8	rwh	**Timer Service Request 0** This flag is set whenever timer 0 matches the conditions configured in the Timer Service Request 0 Configuration Register T0C. A Timer Service Request 0 is also signalled by TINT0SRC. 0_B No Timer Service Request 0 1_B Timer Service Request 0 occurred This flag is cleared by writing a 1.
TI1	9	rwh	**Timer Service Request 1** This flag is set whenever the conditions programmed in the Timer Service Request 1 Configuration Register T1C are met. A Timer Service Request 1 is also signalled. 0_B No Timer Service Request 1 1_B Timer Service Request 1 occurred This flag is cleared by writing a 1.
TIBC	10	rwh	**Transfer Input Buffer Completed** This flag is set whenever a transfer from Input Buffer to the Message RAM has completed and bit IBCR.IBSYS in the Input Buffer Command Request register has been reset by the Message Handler. 0_B No transfer completed 1_B Transfer between Input Buffer and Message RAM completed This flag is cleared by writing a 1.
TOBC	11	rwh	**Transfer Output Buffer Completed** This flag is set whenever a transfer from Message RAM to the Output Buffer has completed and bit OBCR.OBSYS in the Output Buffer Command Request register has been reset by the Message Handler. 0_B No transfer completed 1_B Transfer between Message RAM and the Output Buffer completed This flag is cleared by writing a 1.
SWE	12	rwh	**Stop Watch Event** This flag is set after a stop watch activation when the current cycle counter and Macrotick value are stored in the Stop Watch Register 1 (STPW1). 0_B No Stop Watch Event 1_B Stop Watch Event occurred This flag is cleared by writing a 1.

Field	Bits	Type	Description
SUCS	13	rwh	**Startup Completed Successfully** This flag is set whenever a startup completed successfully and the Communication Controller entered “NORMAL_ACTIVE” state. 0_B No startup completed successfully 1_B Startup completed successfully This flag is cleared by writing a 1.
MBSI	14	rwh	**Message Buffer Status Service Request** This flag is set by the Communication Controller when the Message Buffer status MBS has changed and if bit RDHS1.MBI of that Message Buffer is set (see **Table 20-24**). 0_B No Message Buffer status change of Message Buffer with RDHS1.MBI= 1 has changed 1_B Message Buffer status of at least one Message Buffer with RDHS1.MBI= 1 has changed This flag is cleared by writing a 1.
SDS	15	rwh	**Start of Dynamic Segment** This flag is set by the Communication Controller when the dynamic segment starts. 0_B Dynamic segment not yet started 1_B Dynamic segment started
WUPA	16	rwh	**Wakeup Pattern Channel A** This flag is set by the Communication Controller when a wakeup pattern was received on channel A. Only set when the Communication Controller is in “WAKEUP”, “READY”, or “STARTUP” state, or when in Monitor mode. 0_B No wakeup pattern received on channel A 1_B Wakeup pattern received on channel A This flag is cleared by writing a 1.
MTSA	17	rwh	**MTS Received on Channel A** (vSS!ValidMTSA) Media Access Test symbol received on channel A during the proceeding symbol window. Updated by the Communication Controller for each channel at the end of the symbol window. 0_B No MTS symbol received on channel A 1_B MTS symbol received on channel A This flag is cleared by writing a 1.

Field	Bits	Type	Description
WUPB	24	rwh	**Wakeup Pattern Channel B** This flag is set by the Communication Controller when a wakeup pattern was received on channel B. Only set when the Communication Controller is in “WAKEUP”, “READY”, or “STARTUP” state, or when in Monitor mode. 0_B No wakeup pattern received on channel B 1_B Wakeup pattern received on channel B This flag is cleared by writing a 1.
MTSB	25	rwh	**MTS Received on Channel B** (vSS!ValidMTSB) Media Access Test symbol received on channel B during the proceeding symbol window. Updated by the Communication Controller for each channel at the end of the symbol window. 0_B No MTS symbol received on channel B 1_B MTS symbol received on channel B This flag is cleared by writing a 1.
0	[23:18], [31:26]	r	**Reserved** Returns 0 if read; should be written with 0.

Error Service Request Line Select (EILS)

The Error Service Request Line Select register assigns a service request generated by a specific error service request flag from register EIR to one of the two module service request lines INT0SRC or INT1SRC:

0 = Interrupt assigned to interrupt line (INT0SRC)

1 = Interrupt assigned to interrupt line (INT1SRC)

EILS
Error Service Request Line Select (0028$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0					TABBL	LTVBL	EDBL	0					TABAL	LTVAL	EDAL
r					rw	rw	rw	r					rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				MHFL	IOBAL	IIBAL	EFAL	RFOL	PERRL	CCLL	CCFL	SFOL	SFBML	CNAL	PEMCL
r				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
PEMCL	0	rw	**POC Error Mode Changed Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
CNAL	1	rw	**Command Not Accepted Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
SFBML	2	rw	**SYNC Frames Below Minimum Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC

Field	Bits	Type	Description
SFOL	3	rw	**SYNC Frame Overflow Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
CCFL	4	rw	**Clock Correction Failure Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
CCLL	5	rw	**CHI Command Locked Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
PERRL	6	rw	**Parity Error Service Request Line** 0_B Service Request assigned to service request line eray_int0 (INT0SRC) 1_B Service Request assigned to service request line eray_int1 (INT1SRC)
RFOL	7	rw	**Receive FIFO Overrun Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
EFAL	8	rw	**Empty FIFO Access Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
IIBAL	9	rw	**Illegal Input Buffer Access Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC

Field	Bits	Type	Description
IOBAL	10	rw	**Illegal Output Buffer Access Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
MHFL	11	rw	**Message Handler Constrains Flag Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
EDAL	16	rw	**Error Detected on Channel A Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
LTVAL	17	rw	**Latest Transmit Violation Channel A Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
TABAL	18	rw	**Transmission Across Boundary Channel A Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
EDBL	24	rw	**Error Detected on Channel B Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
LTVBL	25	rw	**Latest Transmit Violation Channel B Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC

Field	Bits	Type	Description
TABBL	26	rw	**Transmission Across Boundary Channel A Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
0	[15:12], [23:19], [31:27]	r	**Reserved** Returns 0 if read; should be written with 0.

Status Service Request Line Select (SILS)

The Status Service Request Line Select register assign an service request generated by a specific status service request flag from register SIR to one of the two module service request lines INT0SRC or INT1SRC:

0 = Interrupt assigned to interrupt line INT0SRC

1 = Interrupt assigned to interrupt line INT1SRC

SILS

Status Service Request Line Select (002C$_H$) **Reset Value: 0303 FFFF$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0						MTSBL	WUPBL	0						MTSAL	WUPAL
r						rw	rw	r						rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SDSL	MBSIL	SUCSL	SWEL	TOBCL	TIBCL	TI1L	TI0L	NMVCL	RFCLL	RFNEL	RXIL	TXIL	CYCSL	CASL	WSTL
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
WSTL	0	rw	**Wakeup Status Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
CASL	1	rw	**Collision Avoidance Symbol Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
CYCSL	2	rw	**Cycle Start Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC

Field	Bits	Type	Description
TXIL	3	rw	**Transmit Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
RXIL	4	rw	**Receive Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
RFNEL	5	rw	**Receive FIFO Not Empty Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
RFCLL	6	rw	**Receive FIFO Critical Level Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
NMVCL	7	rw	**Network Management Vector Changed Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
TI0L	8	rw	**Timer Service Request 0 Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
TI1L	9	rw	**Timer Service Request 1 Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC

Field	Bits	Type	Description
TIBCL	10	rw	**Transfer Input Buffer Completed Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
TOBCL	11	rw	**Transfer Output Buffer Completed Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
SWEL	12	rw	**Stop Watch Event Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
SUCSL	13	rw	**Startup Completed Successfully Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
MBSIL	14	rw	**Message Buffer Status Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
SDSL	15	rw	**Start of Dynamic Segment Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
WUPAL	16	rw	**Wakeup Pattern Channel A Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC

Field	Bits	Type	Description
MTSAL	17	rw	**Media Access Test Symbol Channel A Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
WUPBL	24	rw	**Wakeup Pattern Channel B Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
MTSBL	25	rw	**Media Access Test Symbol Channel B Service Request Line** 0_B Service Request assigned to service request line INT0SRC 1_B Service Request assigned to service request line INT1SRC
0	[23:18], [31:26]	r	**Reserved** Returns 0 if read; should be written with 0.

Error Service Request Enable Set (EIES)

The settings in the Error Service Request Enable register determine which status changes in the Error Service Request Register will result in an service request. The enable bits are set by writing to EIES and reset by writing to EIER. Writing a 1 sets the specific enable bit, a 0 has no effect.

EIES
Error Service Request Enable Set (0030$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0					TABBE	LTVBE	EDBE	0					TABAE	LTVAE	EDAE
r					rwh	rwh	rwh	r					rwh	rwh	rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				MHFE	IOBAE	IIBAE	EFAE	RFOE	PERRE	CCLE	CCFE	SFOE	SFBME	CNAE	PEMCE
r				rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
PEMCE	0	rwh	**POC Error Mode Changed Service Request Enable** Read: 0_B Protocol Error Mode Changed Service Request disabled 1_B Protocol Error Mode Changed Service Request enabled Write: 0_B Unchanged 1_B Enable Protocol Error Mode Changed Service Request
CNAE	1	rwh	**Command Not Accepted Service Request Enable** Read: 0_B Command Not Valid Service Request disabled 1_B Command Not Valid Service Request enabled Write: 0_B Unchanged 1_B Enable Command Not Valid Service Request

Field	Bits	Type	Description
SFBME	2	rwh	**SYNC Frames Below Minimum Service Request Enable** Read: 0_B SYNC Frames Below Minimum Service Request disabled 1_B SYNC Frames Below Minimum Service Request enabled Write: 0_B Unchanged 1_B Enable SYNC Frames Below Minimum Service Request
SFOE	3	rwh	**SYNC Frame Overflow Service Request Enable** Read: 0_B SYNC Frame Overflow Service Request disabled 1_B SYNC Frame Overflow Service Request enabled Write: 0_B Unchanged 1_B Enable Protocol Error Mode Changed Service Request
CCFE	4	rwh	**Clock Correction Failure Service Request Enable** Read: 0_B Clock Correction Failure Service Request disabled 1_B Clock Correction Failure Service Request enabled Write: 0_B Unchanged 1_B Enable Clock Correction Failure Service Request
CCLE	5	rwh	**CHI Command Locked Service Request Enable** Read: 0_B CHI Command Locked Service Request disabled 1_B CHI Command Locked Service Request enabled Write: 0_B Unchanged 1_B Enable CHI Command Locked Service Request
PERRE	6	rwh	**Parity Error Service Request Enable** Read: 0_B Parity Error Service Request disabled 1_B Parity Error Service Request enabled Write: 0_B Unchanged 1_B Enable Parity Error Service Request

Field	Bits	Type	Description
RFOE	7	rwh	**Receive FIFO Overrun Service Request Enable** Read: 0_B Receive FIFO Overrun Service Request disabled 1_B Receive FIFO Overrun Service Request enabled Write: 0_B Unchanged 1_B Enable Receive FIFO Overrun Service Request
EFAE	8	rwh	**Empty FIFO Access Service Request Enable** Read: 0_B Empty FIFO Access Service Request disabled 1_B Empty FIFO Access Service Request enabled Write: 0_B Unchanged 1_B Enable Empty FIFO Access Service Request
IIBAE	9	rwh	**Illegal Input Buffer Access Service Request Enable** Read: 0_B Illegal Input Buffer Access Service Request disabled 1_B Illegal Input Buffer Access Service Request enabled Write: 0_B Unchanged 1_B Enable Illegal Input Buffer Access Service Request
IOBAE	10	rwh	**Illegal Output Buffer Access Service Request Enable** Read: 0_B Illegal Output Buffer Access Service Request disabled 1_B Illegal Output Buffer Access Service Request enabled Write: 0_B Unchanged 1_B Enable Illegal Output Buffer Access Service Request
MHFE	11	rwh	**Message Handler Constraints Flag Service Request Enable** Read: 0_B Message Handler Constraints Flag Service Request disabled 1_B Message Handler Constraints Flag Service Request enabled Write: 0_B Unchanged 1_B Enable Message Handler Constraints Flag Service Request

Field	Bits	Type	Description
EDAE	16	rwh	**Error Detected on Channel A Service Request Enable** Read: 0_B Error Detected on Channel A Service Request disabled 1_B Error Detected on Channel A Service Request enabled Write: 0_B Unchanged 1_B Enable Error Detected on Channel A Service Request
LTVAE	17	rwh	**Latest Transmit Violation Channel A Service Request Enable** Read: 0_B Latest Transmit Violation Channel A Service Request disabled 1_B Latest Transmit Violation Channel A Service Request enabled Write: 0_B Unchanged 1_B Enable Latest Transmit Violation Channel A Service Request
TABAE	18	rwh	**Transmission Across Boundary Channel A Service Request Enable** Read: 0_B Transmission Across Boundary Channel A Service Request disabled 1_B Transmission Across Boundary Channel A Service Request enabled Write: 0_B Unchanged 1_B Enable Transmission Across Boundary Channel A Service Request
EDBE	24	rwh	**Error Detected on Channel B Service Request Enable** Read: 0_B Error Detected on Channel B Service Request disabled 1_B Error Detected on Channel B Service Request enabled Write: 0_B Unchanged 1_B Enable Error Detected on Channel B Service Request

Field	Bits	Type	Description
LTVBE	25	rwh	**Latest Transmit Violation Channel B Service Request Enable** Read: 0_B Latest Transmit Violation Channel B Service Request disabled 1_B Latest Transmit Violation Channel B Service Request enabled Write: 0_B Unchanged 1_B Enable Latest Transmit Violation Channel B Service Request
TABBE	26	rwh	**Transmission Across Boundary Channel B Service Request Enable** Read: 0_B Transmission Across Boundary Channel B Service Request disabled 1_B Transmission Across Boundary Channel B Service Request enabled Write: 0_B Unchanged 1_B Enable Transmission Across Boundary Channel B Service Request
0	[15:12], [23:19], [31:27]	r	**Reserved** Returns 0 if read; should be written with 0.

Error Service Request Enable Reset (EIER)

The settings in the Error Service Request Enable register determine which status changes in the Error Service Request Register will result in a service request. The enable bits are set by writing to EIES and reset by writing to EIER. Writing a 1 resets the specific enable bit, a 0 has no effect.

EIER
Error Service Request Enable Reset (0034_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0					TABBE	LTVBE	EDBE	0					TABAE	LTVAE	EDAE
r					rwh	rwh	rwh	r					rwh	rwh	rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				MHFE	IOBAE	IIBAE	EFAE	RFOE	PERRE	CCLE	CCFE	SFOE	SFBME	CNAE	PEMCE
r				rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
PEMCE	0	rwh	**POC Error Mode Changed Service Request Enable** Read: 0_B Protocol Error Mode Changed Service Request disabled 1_B Protocol Error Mode Changed Service Request enabled Write: 0_B Unchanged 1_B Disable Protocol Error Mode Changed Service Request
CNAE	1	rwh	**Command Not Accepted Service Request Enable** Read: 0_B Command Not Accepted Service Request disabled 1_B Command Not Accepted Service Request enabled Write: 0_B Unchanged 1_B Disable Command Not Accepted Service Request

Field	Bits	Type	Description
SFBME	2	rwh	**SYNC Frames Below Minimum Service Request Enable** Read: 0_B SYNC Frames Below Minimum Service Request disabled 1_B SYNC Frames Below Minimum Service Request enabled Write: 0_B Unchanged 1_B Disable SYNC Frames Below Minimum Service Request
SFOE	3	rwh	**SYNC Frame Overflow Service Request Enable** Read: 0_B SYNC Frame Overflow Service Request disabled 1_B SYNC Frame Overflow Service Request enabled Write: 0_B Unchanged 1_B Disable Protocol Error Mode Changed Service Request
CCFE	4	rwh	**Clock Correction Failure Service Request Enable** Read: 0_B Clock Correction Failure Service Request disabled 1_B Clock Correction Failure Service Request enabled Write: 0_B Unchanged 1_B Disable Clock Correction Failure Service Request
CCLE	5	rwh	**CHI Command Locked Service Request Enable** Read: 0_B CHI Command Locked Service Request disabled 1_B CHI Command Locked Service Request enabled Write: 0_B Unchanged 1_B Disable CHI Command Locked Service Request
PERRE	6	rwh	**Parity Error Service Request Enable** Read: 0_B Parity Error Service Request disabled 1_B Parity Error Service Request enabled Write: 0_B Unchanged 1_B Disable Parity Error Service Request

Field	Bits	Type	Description
RFOE	7	rwh	**Receive FIFO Overrun Service Request Enable** Read: 0_B Receive FIFO Overrun Service Request disabled 1_B Receive FIFO Overrun Service Request enabled Write: 0_B Unchanged 1_B Disable Receive FIFO Overrun Service Request
EFAE	8	rwh	**Empty FIFO Access Service Request Enable** Read: 0_B Empty FIFO Access Service Request disabled 1_B Empty FIFO Access Service Request enabled Write: 0_B Unchanged 1_B Disable Empty FIFO Access Service Request
IIBAE	9	rwh	**Illegal Input Buffer Access Service Request Enable** Read: 0_B Illegal Input Buffer Access Service Request disabled 1_B Illegal Input Buffer Access Service Request enabled Write: 0_B Unchanged 1_B Disable Illegal Input Buffer Access Service Request
IOBAE	10	rwh	**Illegal Output Buffer Access Service Request Enable** Read: 0_B Illegal Output Buffer Access Service Request disabled 1_B Illegal Output Buffer Access Service Request enabled Write: 0_B Unchanged 1_B Disable Illegal Output Buffer Access Service Request
MHFE	11	rwh	**Message Handler Constraints Flag Service Request Enable** Read: 0_B Message Handler Constraints Flag Service Request disabled 1_B Message Handler Constraints Flag Service Request enabled Write: 0_B Unchanged 1_B Disable Message Handler Constraints Flag Service Request

Field	Bits	Type	Description
EDAE	16	rwh	**Error Detected on Channel A Service Request Enable** Read: 0_B Error Detected on Channel A Service Request disabled 1_B Error Detected on Channel A Service Request enabled Write: 0_B Unchanged 1_B Disable Error Detected on Channel A Service Request
LTVAE	17	rwh	**Latest Transmit Violation Channel A Service Request Enable** Read: 0_B Latest Transmit Violation Channel A Service Request disabled 1_B Latest Transmit Violation Channel A Service Request enabled Write: 0_B Unchanged 1_B Disable Latest Transmit Violation Channel A Service Request
TABAE	18	rwh	**Transmission Across Boundary Channel A Service Request Enable** Read: 0_B Transmission Across Boundary Channel A Service Request disabled 1_B Transmission Across Boundary Channel A Service Request enabled Write: 0_B Unchanged 1_B Enable Transmission Across Boundary Channel A Service Request
EDBE	24	rwh	**Error Detected on Channel B Service Request Enable** Read: 0_B Error Detected on Channel B Service Request disabled 1_B Error Detected on Channel B Service Request enabled Write: 0_B Unchanged 1_B Disable Error Detected on Channel B Service Request

Field	Bits	Type	Description
LTVBE	25	rwh	**Latest Transmit Violation Channel B Service Request Enable** Read: 0_B Latest Transmit Violation Channel B Service Request disabled 1_B Latest Transmit Violation Channel B Service Request enabled Write: 0_B Unchanged 1_B Disable Latest Transmit Violation Channel B Service Request
TABBE	26	rwh	**Transmission Across Boundary Channel B Service Request Enable** Read: 0_B Transmission Across Boundary Channel B Service Request disabled 1_B Transmission Across Boundary Channel B Service Request enabled Write: 0_B Unchanged 1_B Disable Transmission Across Boundary Channel B Service Request
0	[15:12], [23:19], [31:27]	r	**Reserved** Returns 0 if read; should be written with 0.

Status Service Request Enable Set (SIES)

The settings in the Status Service Request Enable Set register determine which status changes in the Status Service Request Register will result in a service request. The enable bits are set by writing to SIES and reset by writing to SIER. Writing a 1 sets the specific enable bit, a 0 has no effect.

SIES

Status Service Request Enable Set (0038$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0						MTSBE	WUPBE	0						MTSAE	WUPAE
r						rwh	rwh	r						rwh	rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SDSE	MBSIE	SUCSE	SWEE	TOBCE	TIBCE	TI1E	TI0E	NMVCE	RFCLE	RFNEE	RXIE	TXIE	CYCSE	CASE	WSTE
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
WSTE	0	rwh	**Wakeup Status Service Request Enable** Read: 0_B Wakeup Status Service Request disabled 1_B Wakeup Status Service Request enabled Write: 0_B Unchanged 1_B Enable Wakeup Status Service Request
CASE	1	rwh	**Collision Avoidance Symbol Service Request Enable** Read: 0_B Collision Avoidance Symbol Service Request disabled 1_B Collision Avoidance Symbol Service Request enabled Write: 0_B Unchanged 1_B Enable Collision Avoidance Symbol Service Request
CYCSE	2	rwh	**Cycle Start Service Request Enable** Read: 0_B Cycle Start Service Request disabled 1_B Cycle Start Service Request enabled Write: 0_B Unchanged 1_B Enable Cycle Start Service Request

Field	Bits	Type	Description
TXIE	3	rwh	**Transmit Service Request Enable** Read: 0_B Transmit Service Request disabled 1_B Transmit Service Request enabled Write: 0_B Unchanged 1_B Enable Transmit Service Request
RXIE	4	rwh	**Receive Service Request Enable** Read: 0_B Receive Service Request disabled 1_B Receive Service Request enabled Write: 0_B Unchanged 1_B Enable Receive Service Request
RFNEE	5	rwh	**Receive FIFO Not Empty Service Request Enable** Read: 0_B Receive FIFO Not Empty Service Request disabled 1_B Receive FIFO Not Empty Service Request enabled Write: 0_B Unchanged 1_B Enable Receive FIFO Not Empty Service Request
RFCLE	6	rwh	**Receive FIFO Critical Level Service Request Enable** Read: 0_B Receive FIFO Critical Level Service Request disabled 1_B Receive FIFO Critical Level Service Request enabled Write: 0_B Unchanged 1_B Enable Receive FIFO Critical Level Service Request
NMVCE	7	rwh	**Network Management Vector Changed Service Request Enable** Read: 0_B Network Management Vector Changed Service Request disabled 1_B Network Management Vector Changed Service Request enabled Write: 0_B Unchanged 1_B Enable Network Management Vector Changed Service Request

Field	Bits	Type	Description
TI0E	8	rwh	**Timer Service Request 0 Enable** Read: 0_B Timer Service Request 0disabled 1_B Timer Service Request 0 enabled Write: 0_B Unchanged 1_B Enable Timer Service Request 0
TI1E	9	rwh	**Timer Service Request 1 Enable** Read: 0_B Timer Service Request 1disabled 1_B Timer Service Request 1 enabled Write: 0_B Unchanged 1_B Enable Timer Service Request 1
TIBCE	10	rwh	**Transfer Input Buffer Completed Service Request Enable** Read: 0_B Wakeup Status Service Request disabled 1_B Wakeup Status Service Request enabled Write: 0_B Unchanged 1_B Enable Wakeup Status Service Request
TOBCE	11	rwh	**Transfer Output Buffer Completed Service Request Enable** Read: 0_B Transfer Input Buffer Completed Service Request disabled 1_B Transfer Input Buffer Completed Service Request enabled Write: 0_B Unchanged 1_B Enable Transfer Input Buffer Completed Service Request
SWEE	12	rwh	**Stop Watch Event Service Request Enable** Read: 0_B Stop Watch Event Service Request disabled 1_B Stop Watch Event Service Request enabled Write: 0_B Unchanged 1_B Enable Stop Watch Event Service Request

Field	Bits	Type	Description
SUCSE	13	rwh	**Startup Completed Successfully Service Request Enable** Read: 0_B Startup Completed Successfully Service Request disabled 1_B Startup Completed Successfully Service Request enabled Write: 0_B Unchanged 1_B Enable Startup Completed Successfully Service Request
MBSIE	14	rwh	**Message Buffer Status Service Request Enable** Read: 0_B Message Buffer Status Service Request disabled 1_B Message Buffer Status Service Request enabled Write: 0_B Unchanged 1_B Enable Message Buffer Status Service Request
SDSE	15	rwh	**Start of Dynamic Segment Service Request Enable** Read: 0_B Start of Dynamic Service Request disabled 1_B Start of Dynamic Service Request enabled Write: 0_B Unchanged 1_B Enable Start of Dynamic Service Request
WUPAE	16	rwh	**Wakeup Pattern Channel A Service Request Enable** Read: 0_B Wakeup Pattern Channel A Service Request disabled 1_B Wakeup Pattern Channel A Service Request enabled Write: 0_B Unchanged 1_B Enable Wakeup Pattern Channel A Service Request

Field	Bits	Type	Description
MTSAE	17	rwh	**Media Access Test Symbol Channel A Service Request Enable** Read: 0_B Media Access Test Symbol Channel A Service Request disabled 1_B Media Access Test Symbol Channel A Service Request enabled Write: 0_B Unchanged 1_B Enable Media Access Test Symbol Channel A Service Request
WUPBE	24	rwh	**Wakeup Pattern Channel B Service Request Enable** Read: 0_B Wakeup Pattern Channel B Service Request disabled 1_B Wakeup Pattern Channel B Service Request enabled Write: 0_B Unchanged 1_B Enable Wakeup Pattern Channel A Service Request
MTSBE	25	rwh	**Media Access Test Symbol Channel B Service Request Enable** Read: 0_B Media Access Test Symbol Channel B Service Request disabled 1_B Media Access Test Symbol Channel B Service Request enabled Write: 0_B Unchanged 1_B Enable Media Access Test Symbol Channel B Service Request
0	[23:18], [31:26]	r	**Reserved** Returns 0 if read; should be written with 0.

Status Service Request Enable Reset (SIER)

The settings in the Status Service Request Enable Reset register determine which status changes in the Status Service Request Register will result in a service request. The enable bits are set by writing to SIES and reset by writing to SIER. Writing a 1 resets the specific enable bit, a 0 has no effect.

SIER
Status Service Request Enable Reset(003C$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0						MTSBE	WUPBE	0						MTSAE	WUPAE
r						rwh	rwh	r						rwh	rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SDSE	MBSIE	SUCSE	SWEE	TOBCE	TIBCE	TI1E	TI0E	NMVCE	RFCLE	RFNEE	RXIE	TXIE	CYCSE	CASE	WSTE
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
WSTE	0	rwh	**Wakeup Status Service Request Enable** Read: 0_B Wakeup Status Service Request disabled 1_B Wakeup Status Service Request enabled Write: 0_B Unchanged 1_B Disable Wakeup Status Service Request
CASE	1	rwh	**Collision Avoidance Symbol Service Request Enable** Read: 0_B Collision Avoidance Symbol Service Request disabled 1_B Collision Avoidance Symbol Service Request enabled Write: 0_B Unchanged 1_B Disable Collision Avoidance Symbol Service Request
CYCSE	2	rwh	**Cycle Start Service Request Enable** Read: 0_B Cycle Start Service Request disabled 1_B Cycle Start Service Request enabled Write: 0_B Unchanged 1_B Disable Cycle Start Service Request

Field	Bits	Type	Description
TXIE	3	rwh	**Transmit Service Request Enable** Read: 0_B Transmit Service Request disabled 1_B Transmit Service Request enabled Write: 0_B Unchanged 1_B Disable Transmit Service Request
RXIE	4	rwh	**Receive Service Request Enable** Read: 0_B Receive Service Request disabled 1_B Receive Service Request enabled Write: 0_B Unchanged 1_B Disable Receive Service Request
RFNEE	5	rwh	**Receive FIFO Not Empty Service Request Enable** Read: 0_B Receive FIFO Not Empty Service Request disabled 1_B Receive FIFO Not Empty Service Request enabled Write: 0_B Unchanged 1_B Disable Receive FIFO Not Empty Service Request
RFCLE	6	rwh	**Receive FIFO Critical Level Service Request Enable** Read: 0_B Service Request disabled 1_B Receive FIFO Critical Level Service Request enabled Write: 0_B Unchanged 1_B Disable Receive FIFO Critical Level Service Request
NMVCE	7	rwh	**Network Management Vector Changed Service Request Enable** Read: 0_B Network Management Vector Changed Service Request disabled 1_B Network Management Vector Changed Service Request enabled Write: 0_B Unchanged 1_B Disable Network Management Vector Changed Service Request

Field	Bits	Type	Description
TI0E	8	rwh	**Timer Service Request 0 Enable** Read: 0_B Timer Service Request 0 disabled 1_B Timer Service Request 0 enabled Write: 0_B Unchanged 1_B Disable Service Request 0
TI1E	9	rwh	**Timer Service Request 1 Enable** Read: 0_B Timer Service Request 1 disabled 1_B Timer Service Request 1 enabled Write: 0_B Unchanged 1_B Disable Timer Service Request 1
TIBCE	10	rwh	**Transfer Input Buffer Completed Service Request Enable** Read: 0_B Wakeup Status Service Request disabled 1_B Wakeup Status Service Request enabled Write: 0_B Unchanged 1_B Disable Wakeup Status Service Request
TOBCE	11	rwh	**Transfer Output Buffer Completed Service Request Enable** Read: 0_B Transfer Input Buffer Completed Service Request disabled 1_B Transfer Input Buffer Completed Service Request enabled Write: 0_B Unchanged 1_B Disable Transfer Input Buffer Completed Service Request
SWEE	12	rwh	**Stop Watch Event Service Request Enable** Read: 0_B Stop Watch Event Service Request disabled 1_B Stop Watch Event Service Request enabled Write: 0_B Unchanged 1_B Disable Stop Watch Event Service Request

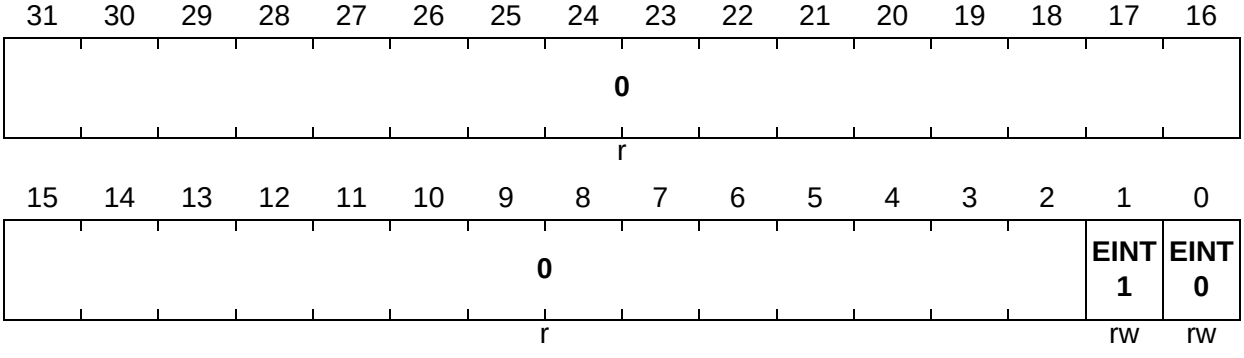
Field	Bits	Type	Description
SUCSE	13	rwh	**Startup Completed Successfully Service Request Enable** Read: 0_B Startup Completed Successfully Service Request disabled 1_B Startup Completed Successfully Service Request enabled Write: 0_B Unchanged 1_B Disable Startup Completed Successfully Service Request
MBSIE	14	rwh	**Message Buffer Status Service Request Enable** Read: 0_B Message Buffer Status Service Request disabled 1_B Message Buffer Status Service Request enabled Write: 0_B Unchanged 1_B Disable Message Buffer Status Service Request
SDSE	15	rwh	**Start of Dynamic Segment Service Request Enable** Read: 0_B Start of Dynamic Service Request disabled 1_B Start of Dynamic Service Request enabled Write: 0_B Unchanged 1_B Disable Start of Dynamic Service Request
WUPAE	16	rwh	**Wakeup Pattern Channel A Service Request Enable** Read: 0_B Wakeup Pattern Channel A Service Request disabled 1_B Wakeup Pattern Channel A Service Request enabled Write: 0_B Unchanged 1_B Disable Wakeup Pattern Channel A Service Request

Field	Bits	Type	Description
MTSAE	17	rwh	**Media Access Test Symbol Channel A Service Request Enable** Read: 0_B Media Access Test Symbol Channel A Service Request disabled 1_B Media Access Test Symbol Channel A Service Request enabled Write: 0_B Unchanged 1_B Disable Media Access Test Symbol Channel A Service Request
WUPBE	24	rwh	**Wakeup Pattern Channel B Service Request Enable** Read: 0_B Wakeup Pattern Channel B Service Request disabled 1_B Wakeup Pattern Channel B Service Request enabled Write: 0_B Unchanged 1_B Disable Wakeup Pattern Channel A Service Request
MTSBE	25	rwh	**Media Access Test Symbol Channel B Service Request Enable** Read: 0_B Media Access Test Symbol Channel B Service Request disabled 1_B Media Access Test Symbol Channel B Service Request enabled Write: 0_B Unchanged 1_B Disable Media Access Test Symbol Channel B Service Request
0	[23:18], [31:26]	r	**Reserved** Returns 0 if read; should be written with 0.

Service Request Line Enable (ILE)

Each of the two service request lines to the Host INT0SRC, INT1SRC can be enabled / disabled separately by programming bit EINT0 and EINT1.

ILE
Service Request Line Enable **(0040_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
EINT0	0	rw	**Enable Service Request Line 0** (INT0SRC) 0_B Service Request line disabled 1_B Service Request line enabled
EINT1	1	rw	**Enable Service Request Line 1** (INT1SRC) 0_B Service Request line disabled 1_B Service Request line enabled
0	[31:2]	r	**Reserved** Returns 0 if read; should be written with 0.

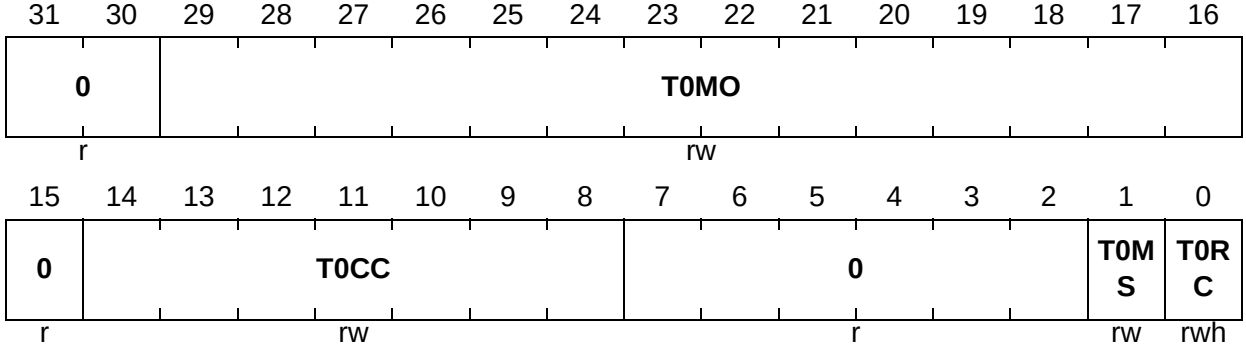
Timer 0 Configuration (T0C)

Absolute timer. Specifies in terms of cycle count and Macrotick the point in time when the timer 0 service request occurs. When the timer 0 service request is asserted, output signal TINT0SR is set to 1 for the duration of one Macrotick and SIR.TI0 is set to 1.

Timer 0 can be activated as long as the POC is either in “NORMAL_ACTIVE” state or in “NORMAL_PASSIVE” state. Timer 0 is deactivated when leaving “NORMAL_ACTIVE” state or “NORMAL_PASSIVE” state except for transitions between the two states.

Before reconfiguration of the timer, the timer has to be halted first by writing 0 to bit T0RC.

T0C
Timer 0 Configuration **(0044_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
T0RC	0	rwh	**Timer 0 Run Control** 0_B Timer 0 halted 1_B Timer 0 running
T0MS	1	rw	**Timer 0 Mode Select** 0_B Single-shot mode 1_B Continuous mode
T0CC	[14:8]	rw	**Timer 0 Cycle Code** The 7-bit timer 0 cycle code determines the cycle set used for generation of the timer 0 service request. For details about the configuration of the cycle code see **“Cycle Counter Filtering” on Page 20-215**.
T0MO	[29:16]	rw	**Timer 0 Macrotick Offset** Configures the Macrotick offset from the beginning of the cycle where the service request is to occur. The Timer 0 Service Request occurs at this offset for each cycle of the cycle set.

Field	Bits	Type	Description
0	[7:2], 15, [31:30]	r	**Reserved** Returns 0 if read; should be written with 0.

Note: The configuration of timer 0 is compared against the Macrotick counter value, there is no separate counter for timer 0. In case the Communication Controller leaves "NORMAL_ACTIVE" or "NORMAL_PASSIVE" state, or if timer 0 is halted by Host command, output signal TINT0SR *is reset to 0 immediately.*

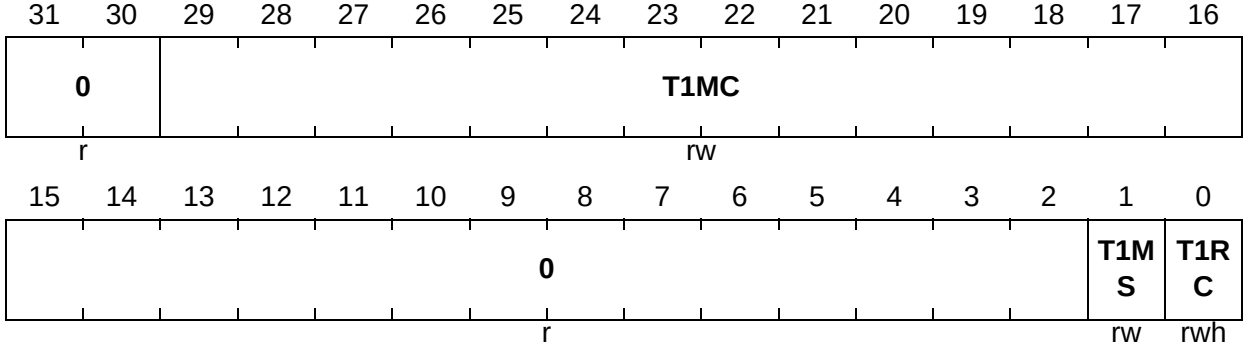
Timer 1 Configuration (T1C)

Relative timer. After the specified number of Macroticks has expired, the timer 1 service request is asserted, output signal TINT1SR is set to 1 for the duration of one Macrotick and SIR.TI1 is set to 1.

Timer 1 can be activated as long as the POC is either in “NORMAL_ACTIVE” state or in “NORMAL_PASSIVE” state. Timer 1 is deactivated when leaving “NORMAL_ACTIVE” state or “NORMAL_PASSIVE” state except for transitions between the two states.

Before reconfiguration of the timer, the timer has to be halted first by resetting bit T1RC to 0.

T1C
Timer 1 Configuration (0048$_H$) Reset Value: 0002 0000$_H$



Field	Bits	Type	Description
T1RC	0	rwh	**Timer 1 Run Control** 0_B Timer 1 halted 1_B Timer 1 running
T1MS	1	rw	**Timer 1 Mode Select** 0_B Single-shot mode 1_B Continuous mode
T1MC	[29:16]	rw	**Timer 1 Macrotick Count** When the configured Macrotick count is reached the timer 1 service request is generated. Valid values are: 2_H …3FFF$_H$ Macroticks in continuous mode 1_H …3FFF$_H$ Macroticks in single-shot mode
0	[15:2], [31:30]	r	**Reserved** Returns 0 if read; should be written with 0.

Note: In case the Communication Controller leaves "NORMAL_ACTIVE" or "NORMAL_PASSIVE" state, or if timer 1 is halted by Host command, output signal TINT1SR *is reset to 0 immediately.*

Stop Watch Register 1 (STPW1)

The stop watch is activated by a rising or falling edge on signal STPW, by a service request 0 or 1 event (rising edge on signal INT0SR or INT1SR) or by the Host by writing bit STPW1.SSWT to 1. With the Macrotick counter increment following next to the stop watch activation the actual cycle counter and Macrotick value are captured in the Stop Watch Register 1 STPW1 while the slot counter values for channel A and B are captured in the Stop Watch Register 2 STPW2.

STPW1
Stop Watch Register 1 **($004C_H$)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		SMTV													
r		rh													

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		SCCV						0	EINT1	EINT0	EETP	SSWT	EDGE	SWMS	ESWT
r		rh						r	rw	rw	rw	rwh	rw	rw	rwh

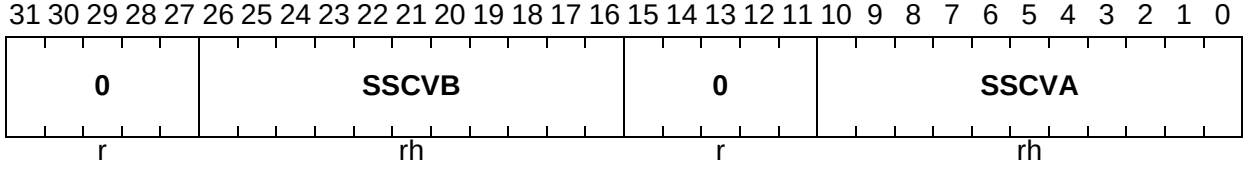
Field	Bits	Type	Description
ESWT	0	rwh	**Enable Stop Watch Trigger** If enabled an edge on input STPW or a service request 0 or 1 event (rising edge on signal INT0SR or INT1SR) activates the stop watch. In single-shot mode this bit is reset to 0 after the actual cycle counter and Macrotick value are stored in the Stop Watch register. 0_B Stop watch trigger disabled 1_B Stop watch trigger enabled
SWMS	1	rw	**Stop Watch Mode Select** It is not possible to change the Stop Watch Mode during enabled stop watch trigger (STPW1.ESWT) 0_B Single-shot mode 1_B Continuous mode
EDGE	2	rw	**Stop Watch Trigger Edge Select** 0_B Falling Edge 1_B Rising Edge

Field	Bits	Type	Description
SSWT	3	rwh	**Software Stop Watch Trigger** When the Host writes this bit to 1 the stop watch is activated. After the actual cycle counter and Macrotick value are stored in the Stop Watch register this bit is reset to 0. The bit is only writeable while ESWT = 0. 0_B Software trigger reset 1_B Stop watch activated by software trigger
EETP	4	rw	**Enable External Trigger Pin** Enables stop watch trigger event via signal STPW if ESWT = 1. 0_B Stop watch trigger via signal STPW disabled 1_B Edge on signal STPW triggers stop watch
EINT0	5	rw	**Enable Service Request 0 Trigger** Enables stop watch trigger by service request 0 event if ESWT = 1. 0_B Stop watch trigger by service request 0 disabled 1_B Service Request 0 event triggers stop watch
EINT1	6	rw	**Enable Service Request 1 Trigger** Enables stop watch trigger by service request 1 event if ESWT = 1. 0_B Stop watch trigger by service request 1 disabled 1_B Service Request 1 event triggers stop watch
SCCV	[13:8]	rh	**Stopped Cycle Counter Value** State of the cycle counter when the stop watch event occurred. Valid values are: $0…3F_H$ Valid Values
SMTV	[29:16]	rh	**Stopped Macrotick Value** State of the Macrotick counter when the stop watch event occurred. Valid values are: $0…3F_H$ Valid Values
0	7, [15:14], [31:30]	r	**Reserved** Returns 0 if read; should be written with 0.

Note: Bits ESWT and SSWT cannot be set to 1 simultaneously. In this case the write access is ignored, and both bits keep their previous values. Therefore either the external stop watch triggers or the software stop watch trigger may be used.

Stop Watch Register 2 (STPW2)

STPW2
Stop Watch Register 2 **(0050_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
SSCVA	[10:0]	rh	**Stop Watch Captured Slot Counter Value Channel A** State of the slot counter for channel A when the stop watch event occurred. Valid values are 0 to 2047 (0_H to $7FF_H$).
SSCVB	[26:16]	rh	**Stop Watch Captured Slot Counter Value Channel B** State of the slot counter for channel B when the stop watch event occurred. Valid values are 0 to 2047 (0_H to $7FF_H$).
0	[15:11], [31:27]	r	**Reserved** Returns 0 if read; should be written with 0.

20.5.2.4 Communication Controller Control Registers

This section describes the registers provided by the Communication Controller to allow the Host to control the operation of the Communication Controller. The FlexRay™ protocol specification requires the Host to write application configuration data in “CONFIG” state only. Please consider that the configuration registers are not locked for writing in “DEFAULT_CONFIG” state.

The configuration data is reset when “DEFAULT_CONFIG” state is entered from application reset. To change POC state from “DEFAULT_CONFIG” to “CONFIG” state the Host has to apply CHI command “CONFIG”. If the Host wants the Communication Controller to leave “CONFIG” state, the Host has to proceed as described on **“Lock Register (LCK)” on Page 20-32**.

SUC Configuration Register 1 (SUCC1)

SUCC1
SUC Configuration Register 1 (0080_H) Reset Value: $0C40\ 1080_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0				CCHB	CCHA	MTSB	MTSA	HCSE	TSM	WUCS	PTA				
r				rw	rw	rw	rw	rw	rw	rw	rw				

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSA					0	TXSY	TXST	PBSY	0			CMD			
rw					r	rw	rw	rh	r			rwh			

Field	Bits	Type	Description
CMD	[3:0]	rwh	**CHI Command Vector** The host may write any CHI command at any time, but certain commands are only enabled in specific POC states. A disabled command will not be executed, the CHI command vector CMD will be reset to 0000_B = “COMMAND_NOT_ACCEPTED”, and flag EIR.CNA in the Error Service Request register will be set to 1. In case the previous CHI command has not yet completed, EIR.CCL is set to 1 together with EIR.CNA; the CHI command needs to be repeated. Except for HALT state, POC state change command applied while the Communication Controller is already in the requested POC state will be ignored. 0000_B COMMAND_NOT_ACCEPTED” 0001_B CONFIG 0010_B READY 0011_B WAKEUP 0100_B RUN 0101_B ALL_SLOTS 0110_B HALT 0111_B FREEZE 1000_B SEND_MTS 1001_B ALLOW_COLDSTART 1010_B RESET_STATUS_INDICATORS 1011_B MONITOR_MODE 1100_B CLEAR_RAMS 1101_B Reserved 1110_B Reserved 1111_B Reserved Reading SUCC1.CMD shows whether the last CHI command was accepted. CCSV.POCS monitors the actual POC state. The reserved CHI commands code hardware test functions.
PBSY	7	rh	**POC Busy** Signals that the POC is busy and cannot accept a command from the Host. SUCC1.CMD is locked against write accesses. Set to 1 after hard reset during initialization of internal RAM blocks. 0_B POC not busy, SUCC1.CMD writeable 1_B POC is busy, SUCC1.CMD locked

Field	Bits	Type	Description
TXST	8	rw	**Transmit Startup Frame in Key Slot**[1) 2)] (pKeySlotUsedForStartup) Defines whether the key slot is used to transmit startup Frames. The bit can be modified in “DEFAULT_CONFIG” or “CONFIG” state only. 0_B No Startup Frame transmission in key slot, node is non-coldstarter 1_B Key slot used to transmit startup Frame, node is leading or following coldstarter
TXSY	9	rw	**Transmit SYNC Frame in Key Slot**[1)] [2)] (pKeySlotUsedForSync) Defines whether the key slot is used to transmit SYNC Frames. The bit can be modified in “DEFAULT_CONFIG” or “CONFIG” state only. 0_B No SYNC Frame transmission in key slot, node is neither sync nor coldstart node 1_B Key slot used to transmit SYNC Frames, node is sync node
CSA	[15:11]	rw	**Cold Start Attempts**[1)](gColdStartAttempts) Configures the maximum number of attempts that a cold starting node is permitted to try to start up the network without receiving any valid response from another node. It can be modified in “DEFAULT_CONFIG” or “CONFIG” state only. Must be identical in all nodes of a cluster. Valid values are 2 to 31.
PTA	[20:16]	rw	**Passive to Active**[1)] (pAllowPassiveToActive) Defines the number of consecutive even / odd cycle pairs that must have valid clock correction terms before the Communication Controller is allowed to transit from “NORMAL_PASSIVE” to “NORMAL_ACTIVE” state. If set to 00000_B the Communication Controller is not allowed to transit from “NORMAL_PASSIVE” to “NORMAL_ACTIVE” state. It can be modified in “DEFAULT_CONFIG” or “CONFIG” state only. Valid values are 0 to 31 even / odd cycle pairs.

Field	Bits	Type	Description
WUCS	21	rw	**Wakeup Channel Select**[1] (pWakeupChannel) With this bit the Host selects the channel on which the Communication Controller sends the Wakeup pattern. The Communication Controller ignores any attempt to change the status of this bit when not in “DEFAULT_CONFIG” or “CONFIG” state. 0_B Send wakeup pattern on channel A 1_B Send wakeup pattern on channel B
TSM	22	rw	**Transmission Slot Mode**[1] (pSingleSlotEnabled) Selects the initial transmission slot mode. In SINGLE slot mode the Communication Controller may only transmit in the preconfigured key slot. The key slot ID is configured in the Header Section of Message Buffer 0 respectively Message Buffers 0 and 1 depending on bit MRC.SPLM. In case SUCC1.TSM = 1, Message Buffer 0 respectively Message Buffers 0,1 can be (re)configured in “DEFAULT_CONFIG” or “CONFIG” state only. In ALL slot mode the Communication Controller may transmit in all slots. The bit can be written in “DEFAULT_CONFIG” or “CONFIG” state only. The communication controller changes to ALL slot mode when the Host successfully applied the ALL_SLOTS command by writing SUCC1.CMD = 0101_B in POC states “NORMAL_ACTIVE” or “NORMAL_PASSIVE”. The actual slot mode is monitored by CCSV.SLM. 0_B ALL Slot Mode 1_B SINGLE Slot Mode (default after application reset)
HCSE	23	rw	**Halt due to Clock Sync Error**[1] (pAllowHaltDueToClock) Controls the transition to “HALT” state due to a clock synchronization error. The bit can be modified in “DEFAULT_CONFIG” or “CONFIG” state only. 0_B Communication Controller will enter / remain in “NORMAL_PASSIVE” 1_B Communication Controller will enter “HALT” state
MTSA	24	rw	**Select Channel A for MTS Transmission**[1] [3] The bit selects channel A for MTS symbol transmission. The flag is reset by default and may be modified only in “DEFAULT_CONFIG” or “CONFIG” state. 0_B Channel A disabled for MTS transmission 1_B Channel A selected for MTS transmission

Field	Bits	Type	Description
MTSB	25	rw	**Select Channel B for MTS Transmission**[1) 3)] The bit selects channel B for MTS symbol transmission. The flag is reset by default and may be modified only in "DEFAULT_CONFIG" or "CONFIG" state. 0_B Channel B disabled for MTS transmission 1_B Channel B selected for MTS transmission
CCHA	26	rw	**Connected to Channel A**[1)] (pChannels) Configures whether the node is connected to channel A. 0_B Not connected to channel A 1_B Node connected to channel A (default after application reset)
CCHB	27	rw	**Connected to Channel B**[1)] (pChannels) Configures whether the node is connected to channel B. 0_B Not connected to channel B 1_B Node connected to channel B (default after application reset)
0	[6:4], 10, [31:28]	r	**Reserved** Returns 0 if read; should be written with 0.

1) This bit can be updated in "DEFAULT_CONFIG" or "CONFIG" state only!

2) The protocol requires that both bits TXST and TXSY are set for coldstart nodes.

3) MTSA and MTSB may also be changed outside "DEFAULT_CONFIG" or "CONFIG" state when the write to SUCC1 register is directly preceded by the unlock sequence as described in Lock Register (LCK). This may be combined with CHI command "SEND_MTS". If both bits MTSA and MTSB are set to 1 an MTS symbol will be transmitted on both channels when requested by writing SUCC1.CMD = 1000_B.

COMMAND_NOT_ACCEPTED

SUCC1.CMD is reset to 0000_B due to one of the following conditions:

- Illegal command applied by the Host
- Host writes command_not_accepted
- Host applied new command while execution of the previous Host command has not completed

When SUCC1.CMD is reset to 0000_B, bit EIR.CNA in the Error Service Request register is set, and - if enabled - an service request is generated. Commands which are not accepted are not executed.

CONFIG

Go to POC state "CONFIG" when called in POC states "DEFAULT_CONFIG", "READY", or in "MONITOR_MODE". When called in "HALT" state transits to POC state

“DEFAULT_CONFIG“. When called in any other state, SUCC1.CMD will be reset to 0000_B = “COMMAND_NOT_ACCEPTED”.

READY

Go to POC state “READY” when called in POC states “CONFIG”, “NORMAL_ACTIVE”, “NORMAL_PASSIVE”, “STARTUP”, or “WAKEUP”. When called in any other state, SUCC1.CMD will be reset to 0000_B = “COMMAND_NOT_ACCEPTED”.

WAKEUP

Go to POC state WAKEUP when called in POC state “READY”. When called in any other state, SUCC1.CMD will be reset to 0000_B = “COMMAND_NOT_ACCEPTED”.

RUN

Go to POC state “STARTUP” when called in POC state “READY”. When called in any other state, SUCC1.CMD will be reset to 0000_B = “COMMAND_NOT_ACCEPTED”.

ALL_SLOTS

Leave SINGLE slot mode after successful startup / integration at the next end of cycle when called in POC states “NORMAL_ACTIVE” or “NORMAL_PASSIVE”. When called in any other state, SUCC1.CMD will be reset to
0000_B = “COMMAND_NOT_ACCEPTED”.

HALT

Set the halt request CCSV.HRQ bit in the Communication Controller Status Vector register and go to POC state “HALT” at the next end of cycle when called in POC states “NORMAL_ACTIVE” or “NORMAL_PASSIVE“. When called in any other state, SUCC1.CMD will be reset to 0000_B = “COMMAND_NOT_ACCEPTED“.

FREEZE

Set the freeze status indicator CCSV.FSI and go to POC state “HALT” immediately. Can be called from any state.

SEND_MTS

Send single MTS symbol during the next following symbol window on the channel configured by SUCC1.MTSA, SUCC1.MTSB, when called in POC state “NORMAL_ACTIVE”. When called in any other state, SUCC1.CMD will be reset to 0000_B = “COMMAND_NOT_ACCEPTED”.

ALLOW_COLDSTART

The command resets bit CCSV.CSI to enable the node to become cold starter. When called in states “DEFAULT_CONFIG“, “CONFIG”, “HALT”, or “MONITOR_MODE“. SUCC1.CMD will be reset to 0000_B = “COMMAND_NOT_ACCEPTED”. To become leading coldstarter it is also required that both TXST and TXSY are set.

RESET_STATUS_INDICATORS

Resets status flags CCSV.CSNI, CCSV.CSAI, and CCSV.WSV to their default values. May be called in POC state READY. When called in any other state, SUCC1.CMD will be reset to 0000_B = “COMMAND_NOT_ACCEPTED”.

MONITOR_MODE

Enter MONITOR_MODE when called in POC state CONFIG. In this mode the Communication Controller is able to receive FlexRay™ Frames and wakeup pattern. It is also able to detect coding errors. The temporal integrity of received Frames is not checked. This mode can be used for debugging purposes, e.g. in case that the startup of a FlexRay™ network fails. When called in any other state, SUCC1.CMD will be reset to 0000_B = “COMMAND_NOT_ACCEPTED”. For details see **“MONITOR_MODE” on Page 20-199**.

CLEAR_RAMS

Sets bit MHDS.CRAM in the Message Handler Status register when called in “DEFAULT_CONFIG” or “CONFIG” state. When called in any other state, SUCC1.CMD will be reset to 0000_B = “COMMAND_NOT_ACCEPTED”. MHDS.CRAM is also set when the Communication Controller leaves application reset. By setting MHDS.CRAM all internal RAM blocks are initialized to zero. Note that only the currently active IBF bank is cleared. To clear the 2nd bank as well, CUST1.IBF1PAG and CUST1.IBF2PAG need to be set and command CLEAR_RAMS need to be issued again. This is required in particular after an application reset. If the 2nd bank of IBF is left unused, this procedure is not required. During the initialization of the RAMs, SUCC1.PBSY will show POC busy. Access to the configuration and status registers is possible during execution of CHI command CLEAR_RAMS.

The initialization of the E-Ray internal RAM blocks requires 2048 f_{CLC_ERAY} cycles. There should be no Host access to IBF or OBF during initialization of the internal RAM blocks after application reset or after assertion of CHI command CLEAR_RAMS. Before asserting CHI command CLEAR_RAMS the Host should make sure that no transfer between Message RAM and IBF / OBF or the Transient Buffer RAMs is ongoing. This command also resets the Message Buffer Status registers MHDS, LDTS, FSR, MHDF, TXRQ1, TXRQ2, TXRQ3, TXRQ4, NDAT1, NDAT2, NDAT3, NDAT4, MBSC1, MBSC2, MBSC3, and MBSC4.

Note: All accepted commands with exception of CLEAR_RAMS and SEND_MTS will cause a change of register CCSV after at most 8 cycles of the slower of the two clocks f_{CLC_ERAY} and f_{SCLK}, assumed that POC was not busy when the command was applied and that no POC state change was forced by bus activity in that time Frame. Reading register CCSV will show data that is delayed by synchronization from f_{SCLK} to f_{CLC_ERAY} domain and by the Host-specific CPU interface.

Table 20-4 below references the CHI commands from the FlexRay™ Protocol Specification v2.1 (section 2.2.1.1, Table 2-2) to the E-Ray CHI command vector CMD.]

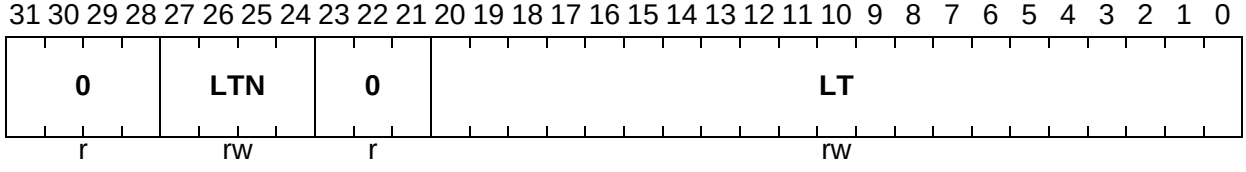
Table 20-4 Reference to CHI Host command summary from FlexRay™ protocol specification

CHI Command	Where processed (POC State)	CHI Command Vector CMD
ALL_SLOT	POC:NORMAL_ACTIVE, POC:NORMAL_PASSIVE	ALL_SLOTS
ALLOW_COLDSTART	All except POC:DEFAULT_CONFIG, POC:CONFIG, POC:HALT	ALLOW_COLDSTART
CONFIG	POC:DEFAULT_CONFIG, POC:READY	CONFIG
CONFIG_COMPLETE	POC:CONFIG	Unlock sequence & READY
DEFAULT_CONFIG	POC:HALT	CONFIG
FREEZE	All	FREEZE
HALT	POC:NORMAL_ACTIVE, POC:NORMAL_PASSIVE	HALT
READY	All except POC:DEFAULT_CONFIG, POC:CONFIG, POC:READY, POC:HALT	READY
RUN	POC:READY	RUN
WAKEUP	POC:READY	WAKEUP

SUC Configuration Register 2 (SUCC2)

The Communication Controller accepts modifications of the register in “DEFAULT_CONFIG” or “CONFIG” state only.

SUCC2
SUC Configuration Register 2 (0084_H) Reset Value: 0100 0504_H



Field	Bits	Type	Description
LT	[20:0]	rw	**Listen Timeout**[1] (pdListenTimeout) Configures wakeup / startup listen timeout in Microticks. The range for wakeup / startup listen timeout (pdListenTimeout) is 1284 to 1283846 (504_H to 139706_H) Microticks
LTN	[27:24]	rw	**Listen Timeout Noise**[1] (gListenNoise - 1) Configures the upper limit for startup and wakeup listen timeout in the presence of noise expressed as a multiple of the cluster constant pdListenTimeout. The range of pdListenTimeout 2 to 16. LTN must be configured identical in all nodes of a cluster. 1_H Listen Timeout Noise is equal 2 2_H Listen Timeout Noise is equal 3 $\cdots_H$ ... F_H Listen Timeout Noise is equal 16
0	[23:21], [31:28]	r	**Reserved** Returns 0 if read; should be written with 0.

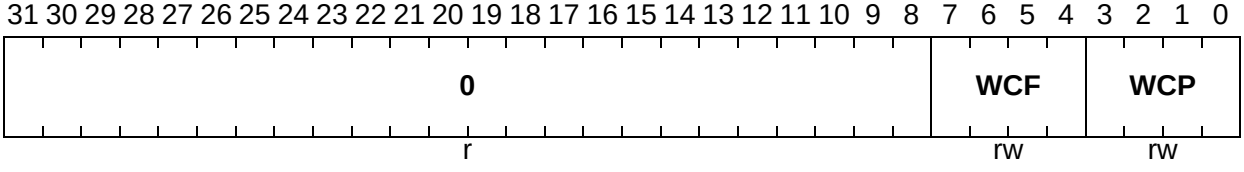
1) This bit can be updated in “DEFAULT_CONFIG” or “CONFIG” state only!

Note: The wakeup / startup noise timeout is calculated as follows:
The wakeup / startup noise timeout = pdListenTimeout • gListenNoise
= LT • (LTN+ 1)

SUC Configuration Register 3 (SUCC3)

The Communication Controller accepts modifications of the register in “DEFAULT_CONFIG” or “CONFIG” state only.

SUCC3
SUC Configuration Register 3 (0088_H) Reset Value: $0000\ 0011_H$



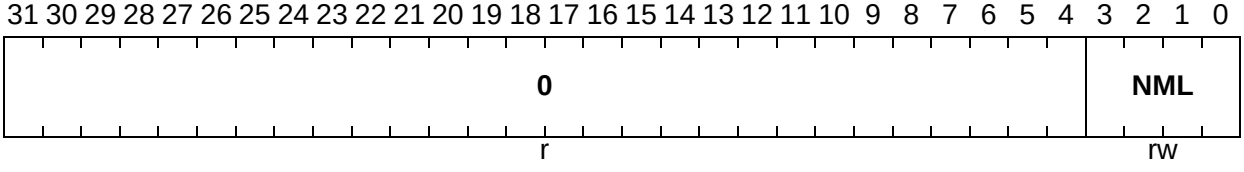
Field	Bits	Type	Description
WCP	[3:0]	rw	**Maximum Without Clock Correction Passive**[1)] (gMaxWithoutClockCorrectionPassive) Defines the number of consecutive even / odd cycle pairs with missing clock correction terms that will cause a transition from “NORMAL_ACTIVE” to “NORMAL_PASSIVE” state. Must be identical in all nodes of a cluster. Valid values are 1 to 15 (1_H to F_H) cycle pairs.
WCF	[7:4]	rw	**Maximum Without Clock Correction Fatal**[1)] (gMaxWithoutClockCorrectionFatal) Defines the number of consecutive even / odd cycle pairs with missing clock correction terms that will cause a transition from “NORMAL_ACTIVE” or “NORMAL_PASSIVE” to “HALT” state. Must be identical in all nodes of a cluster. Valid values are 1 to 15 (1_H to F_H)cycle pairs.
0	[31:8]	r	**Reserved** Returns 0 if read; should be written with 0.

1) This bit can be updated in “DEFAULT_CONFIG” or “CONFIG” state only!

NEM Configuration Register (NEMC)

The Communication Controller accepts modifications of the register in “DEFAULT_CONFIG” or “CONFIG” state only.

NEMC
NEM Configuration Register (008C$_H$) Reset Value: 0000 0000$_H$



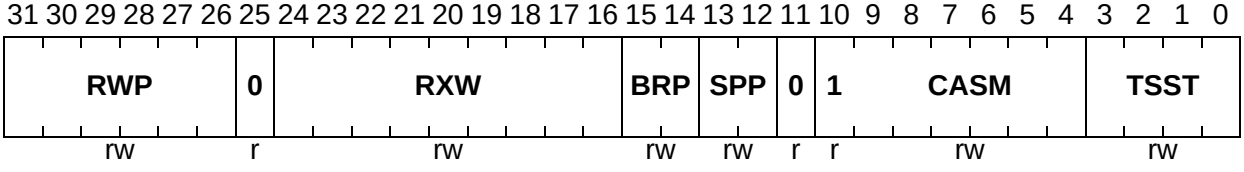
Field	Bits	Type	Description
NML	[3:0]	rw	**Network Management Vector Length**[1)] (gNetworkManagementVectorLength) These bits configure the length of the NM Vector. The configured length must be identical in all nodes of a cluster. Valid values are 0 to 12 (0_H to C_H) bytes.
0	[31:4]	r	**Reserved** Returns 0 if read; should be written with 0.

1) This bit can be updated in “DEFAULT_CONFIG” or “CONFIG” state only!

PRT Configuration Register 1 (PRTC1)

The Communication Controller accepts modifications of the register in “DEFAULT_CONFIG” or “CONFIG” state only.

PRTC1
PRT Configuration Register 1 (0090_H) Reset Value: $084C\ 0633_H$



Field	Bits	Type	Description
TSST	[3:0]	rw	**Transmission Start Sequence Transmitter**[1)] (gdTSSTransmitter) Configures the duration of the Transmission Start Sequence (TSS) in terms of Bit Times (1 bit time = 4 Microticks = 100ns at 10Mbps). Must be identical in all nodes of a cluster. Valid values are 3 to 15 (3_H to F_H) Bit Times.
CASM	[10:4]	rw	**Collision Avoidance Symbol Maximum**[1)] (gdCASRxLowMax) Configures the upper limit of the acceptance window for a collision avoidance symbol (CAS). Valid values are 67 to 99 (43_H to 63_H). Most significant bit of CASM is hard wired to 1 and can not be modified.
SPP	[13:12]	rw	**Strobe Point Position**[1)] Defines the sample count value for strobing. The strobed bit value is set to the voted value when the sample count is incremented to the value configured by SPP. 00_B Sample 5 (default) 01_B Sample 4 10_B Sample 6 11_B Reserved; should not be used. *Note: The current revision 2.1 of the FlexRay™ protocol requires that SPP = 00_B. The alternate strobe point positions could be used to compensate for asymmetries in the physical layer.*

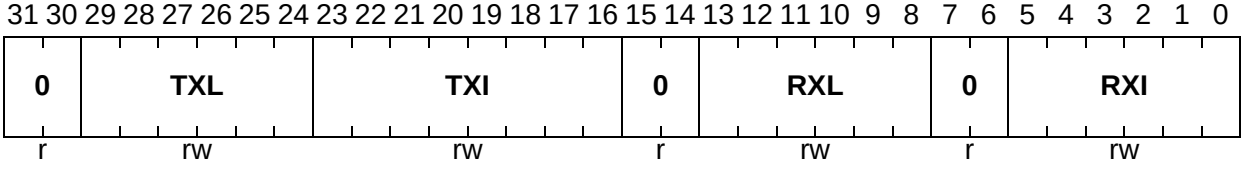
Field	Bits	Type	Description
BRP	[15:14]	rw	**Baud Rate Prescaler**[1] (gdSampleClockPeriod, pSamplePerMicrotick) The baud rate prescaler configures the baud rate on the FlexRay™ bus. The baud rates listed below are valid with a sample clock f_{SCLK}= 80 MHz. One bit time always consists of 8 samples independent of the configured baud rate. 00_B 10 Mbit/s (1 Microtick= 25 ns; twice sampled with f_{SCLK}) gdSampleClockPeriod = 12.5 ns = 1 / f_{SCLK} pSamplesPerMicrotick = 2 01_B 5 Mbit/s (1 Microtick= 25ns; single sampled with f_{SCLK} / 2) gdSampleClockPeriod = 25 ns = 2 / f_{SCLK} pSamplesPerMicrotick = 1 10_B 2.5 Mbit/s (1 Microtick = 50ns; single sampled with f_{SCLK} / 4) gdSampleClockPeriod = 50 ns = 4 / f_{SCLK} pSamplesPerMicrotick = 1 11_B Reserved; should not be used (2.5 Mbit/s (1 Microtick = 50 ns; single sampled with f_{SCLK} / 4) gdSampleClockPeriod = 50 ns = 4 / f_{SCLK} pSamplesPerMicrotick = 1
RXW	[24:16]	rw	**Wakeup Symbol Receive Window Length**[1] (gdWakeupSymbolRxWindow**)** Configures the number of Bit Times used by the node to test the duration of the received wakeup pattern. Must be identical in all nodes of a cluster. Valid values are 76 to 301 ($4C_H$ to $12D_H$) Bit Times.
RWP	[31:26]	rw	**Repetitions of Tx Wakeup Pattern**[1] (pWakeupPattern) Configures the number of repetitions (sequences) of the Tx wakeup symbol. Valid values are 2 to 63 (2_H to $3F_H$).
0	11, 25	r	**Reserved** Returns 0 if read; should be written with 0.

1) This bit can be updated in “DEFAULT_CONFIG” or “CONFIG” state only!

PRT Configuration Register 2 (PRTC2)

The Communication Controller accepts modifications of the register in “DEFAULT_CONFIG” or “CONFIG” state only.

PRTC2
PRT Configuration Register 2 (0094_H) Reset Value: $0F2D\ 0A0E_H$



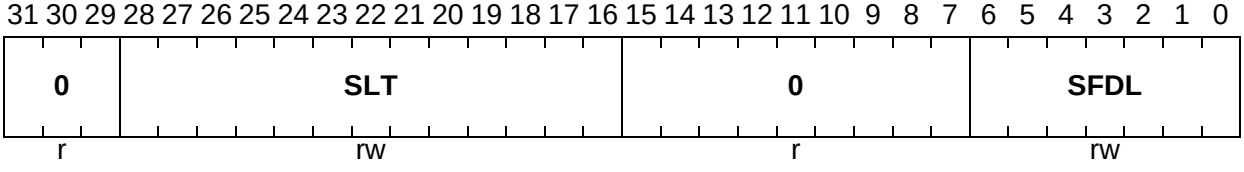
Field	Bits	Type	Description
RXI	[5:0]	rw	**Wakeup Symbol Receive Idle**[1] (gdWakeupSymbolRxIdle) Configures the number of Bit Times used by the node to test the duration of the idle phase of the received wakeup symbol. Must be identical in all nodes of a cluster. Valid values are 14 to 59 (E_H to $3B_H$) Bit Times.
RXL	[13:8]	rw	**Wakeup Symbol Receive Low**[1] (gdWakeupSymbolRxLow) Configures the number of Bit Times used by the node to test the duration of the low phase of the received wakeup symbol. Must be identical in all nodes of a cluster. Valid values are 10 to 55 (A_H to 37_H) Bit Times.
TXI	[23:16]	rw	**Wakeup Symbol Transmit Idle**[1] (gdWakeupSymbolTxIdle) Configures the number of Bit Times used by the node to transmit the idle phase of the wakeup symbol. Must be identical in all nodes of a cluster. Valid values are 45 to 180 ($2D_H$ to $B4_H$)Bit Times.
TXL	[29:24]	rw	**Wakeup Symbol Transmit Low**[1] (gdWakeupSymbolTxLow) Configures the number of Bit Times used by the node to transmit the low phase of the wakeup symbol. Must be identical in all nodes of a cluster. Valid values are 15 to 60 (F_H to $3C_H$) Bit Times.
0	[7:6], [15:14], [31:30]	r	**Reserved** Returns 0 if read; should be written with 0.

1) This bit can be updated in “DEFAULT_CONFIG” or “CONFIG” state only!

MHD Configuration Register (MHDC)

The Communication Controller accepts modifications of the register in “DEFAULT_CONFIG” or “CONFIG” state only.

MHDC
MHD Configuration Register (0098$_H$) Reset Value: 0000 0000$_H$



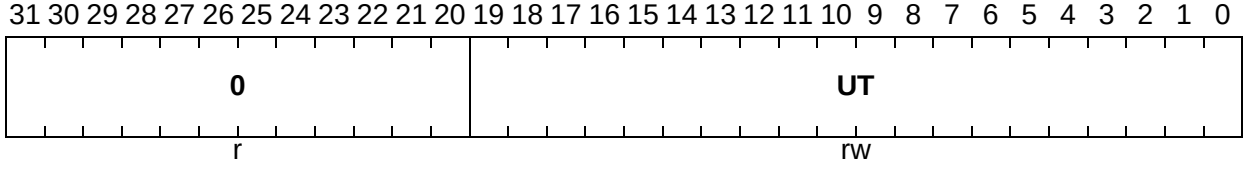
Field	Bits	Type	Description
SFDL	[6:0]	rw	**Static Frame Data Length** (gPayloadLengthStatic)[1)] Configures the cluster-wide payload length for all Frames sent in the static segment in double byte. The payload length must be identical in all nodes of a cluster. Valid values are 0 to 127 (0 to 7F$_H$).
SLT	[28:16]	rw	**Start of Latest Transmit** (pLatestTx)[1)] Configures the maximum minislot value allowed before inhibiting Frame transmission in the dynamic segment of the cycle. There is no transmission dynamic segment if SLT is reset to zero. Valid values are 0 to 7981 (0 to 1F2D$_H$) minislots.
0	[15:7], [31:29]	r	**Reserved** Returns 0 if read; should be written with 0.

1) This bit can be updated in “DEFAULT_CONFIG” or “CONFIG” state only!

GTU Configuration Register 1 (GTUC01)

The Communication Controller accepts modifications of the register in “DEFAULT_CONFIG” or “CONFIG” state only.

GTUC01
GTU Configuration Register 1 ($00A0_H$) Reset Value: $0000\ 0280_H$



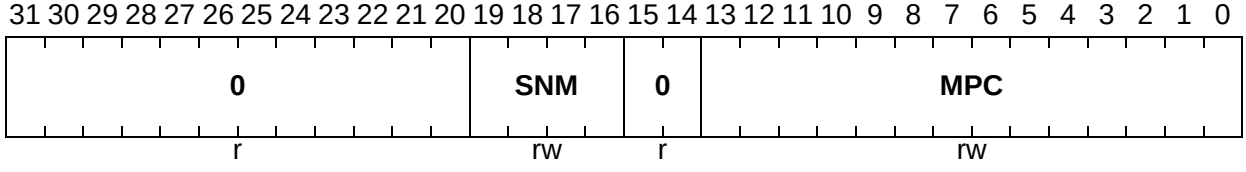
Field	Bits	Type	Description
UT	[19:0]	rw	**Microtick per Cycle** (pMicroPerCycle)[1)] Configures the duration of the communication cycle in Microticks. Valid values are 640 to 640000 (280_H to $9C400_H$) Microticks.
0	[31:20]	r	**Reserved** Returns 0 if read; should be written with 0.

1) This bit can be updated in “DEFAULT_CONFIG” or “CONFIG” state only!

GTU Configuration Register 2 (GTUC02)

The Communication Controller accepts modifications of the register in “DEFAULT_CONFIG” or “CONFIG” state only.

GTUC02
GTU Configuration Register 2 ($00A4_H$) Reset Value: $0002\ 000A_H$



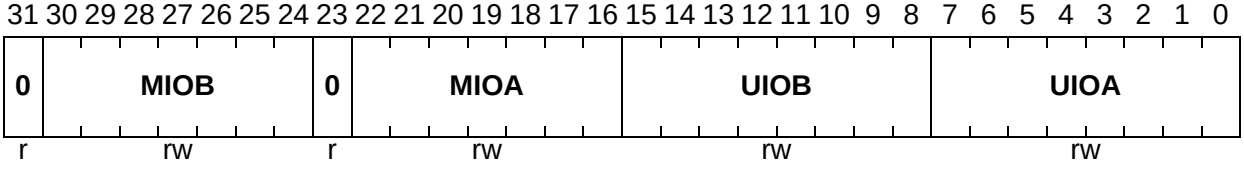
Field	Bits	Type	Description
MPC	[13:0]	rw	**Macrotick Per Cycle** (gMacroPerCycle)[1)] Configures the duration of one communication cycle in Macroticks. The cycle length must be identical in all nodes of a cluster. Valid values are 10 to 16000 (A_H to $3E80_H$) Macroticks.
SNM	[19:16]	rw	**Sync Node Max** (gSyncNodeMax)[1)] Maximum number of Frames within a cluster with SYNC Frame indicator bit SYN set to 1. Must be identical in all nodes of a cluster. Valid values are 2 to 15 (2_H to F_H).
0	[15:14], [31:20]	r	**Reserved** Returns 0 if read; should be written with 0.

1) This bit can be updated in “DEFAULT_CONFIG” or “CONFIG” state only!

GTU Configuration Register 3 (GTUC03)

The Communication Controller accepts modifications of the register in “DEFAULT_CONFIG” or “CONFIG” state only.

GTUC03
GTU Configuration Register 3 ($00A8_H$) Reset Value: $0202\ 0000_H$



Field	Bits	Type	Description
UIOA	[7:0]	rw	**Microtick Initial Offset Channel A**[1)] (pMicroInitialOffset[A]) Configures the number of Microticks between the actual time reference point on channel A and the subsequent Macrotick boundary of the secondary time reference point. The parameter depends on pDelayCompensation[A] and therefore has to be set for each channel independently. Valid values are 0 to 240 (0_H to $F0_H$) Microticks.
UIOB	[15:8]	rw	**Microtick Initial Offset Channel B**[1)] (pMicroInitialOffset[B]) Configures the number of Microticks between the actual time reference point on channel B and the subsequent Macrotick boundary of the secondary time reference point. The parameter depends on pDelayCompensation[B] and therefore has to be set for each channel independently. Valid values are 0 to 240 (0_H to $F0_H$) Microticks.
MIOA	[22:16]	rw	**Macrotick Initial Offset Channel A** (gMacroInitialOffset[A])[1)] Configures the number of Macroticks between the static slot boundary and the subsequent Macrotick boundary of the secondary time reference point based on the nominal Macrotick duration. Must be identical in all nodes of a cluster. Valid values are 2 to 72 (2_H to 48_H) Macroticks.
MIOB	[30:24]	rw	**Macrotick Initial Offset Channel B** (gMacroInitialOffset[B])[1)] Configures the number of Macroticks between the static slot boundary and the subsequent Macrotick boundary of the secondary time reference point based on the nominal Macrotick duration. Must be identical in all nodes of a cluster. Valid values are 2 to 72 (2_H to 48_H) Macroticks.

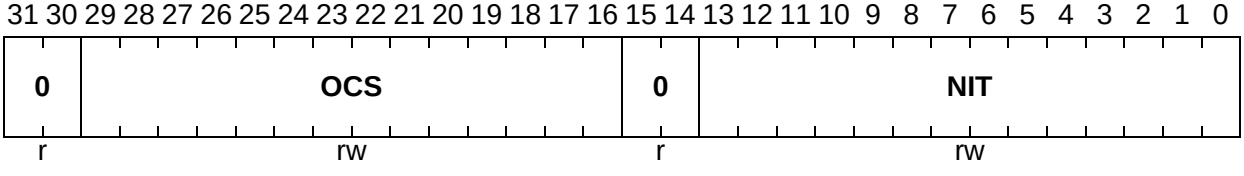
Field	Bits	Type	Description
0	23, 31	r	**Reserved** Returns 0 if read; should be written with 0.

1) This bit can be updated in “DEFAULT_CONFIG” or “CONFIG” state only!

GTU Configuration Register 4 (GTUC04)

The Communication Controller accepts modifications of the register in “DEFAULT_CONFIG” or “CONFIG” state only. For details about configuration of NIT and OCS see **“Configuration of Network Idle Time (NIT) Start and Offset Correction Start” on Page 20-188**.

GTUC04
GTU Configuration Register 4 (00AC$_H$) Reset Value: 0008 0007$_H$



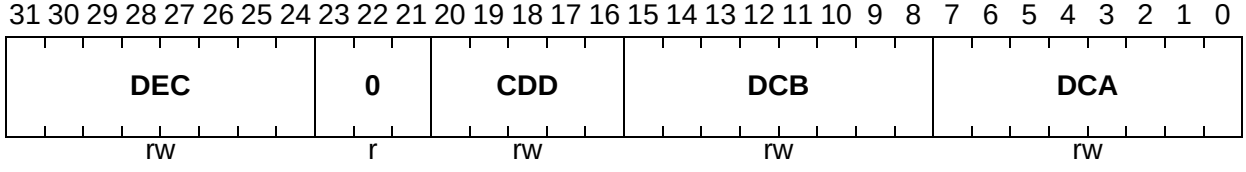
Field	Bits	Type	Description
NIT	[13:0]	rw	**Network Idle Time Start** [1)] (gMacroPerCycle - gdNIT - 1) Configures the starting point of the Network Idle Time (NIT) at the end of the communication cycle expressed in terms of Macroticks from the beginning of the cycle. The start of network idle time (NIT) is recognized if Macrotick = gMacroPerCycle - gdNIT -1 and the increment pulse of Macrotick is set. Must be identical in all nodes of a cluster. Valid values are 7 to 15997 (7_H to $3E7D_H$) Macroticks.
OCS	[29:16]	rw	**Offset Correction Start**[1)] (gOffsetCorrectionStart - 1) Determines the start of the offset correction within the network idle time (NIT) phase, calculated from start of cycle. Must be identical in all nodes of a cluster. For cluster consisting of E-Ray implementations only, it is sufficient to program OCS = NIT + 1. Valid values are 8 to 15998 (8_H to $3E7E_H$) Macroticks.
0	[15:14], [31:30]	r	**Reserved** Returns 0 if read; should be written with 0.

1) This bit can be updated in “DEFAULT_CONFIG” or “CONFIG” state only!

GTU Configuration Register 5 (GTUC05)

The Communication Controller accepts modifications of the register in “DEFAULT_CONFIG” or “CONFIG” state only.

GTUC05
GTU Configuration Register 5 ($00B0_H$) Reset Value: $0E00\ 0000_H$



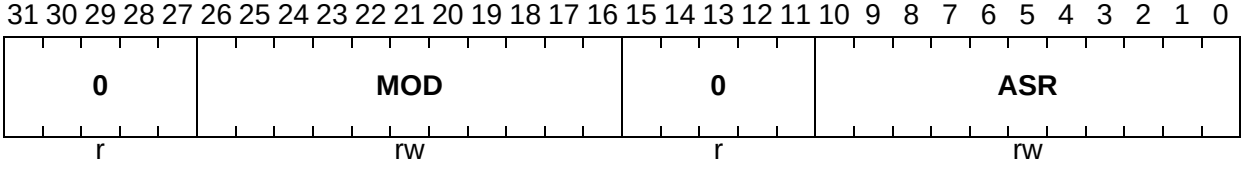
Field	Bits	Type	Description
DCA	[7:0]	rw	**Delay Compensation Channel A**[1] (pDelayCompensation[A]) Used to compensate for reception delays on channel A. This covers assumed propagation delay up to cPropagationDelayMax for Microticks in the range of 0.0125µs to 0.05µs. In practice, the minimum of the propagation delays of all sync nodes should be applied. Valid values are 0 to 200 (0_H to $C8_H$) Microticks.
DCB	[15:8]	rw	**Delay Compensation Channel B**[1] (pDelayCompensation[B]) Used to compensate for reception delays on channel B. This covers assumed propagation delay up to cPropagationDelayMax for Microticks in the range of 0.0125 to 0.05µs. In practice, the minimum of the propagation delays of all sync nodes should be applied. Valid values are 0 to 200 (0_H to $C8_H$) Microticks.
CDD	[20:16]	rw	**Cluster Drift Damping** (pClusterDriftDamping)[1] Configures the cluster drift damping value used in clock synchronization to minimize accumulation of rounding errors. Valid values are 0 to 20 (0_H to 14_H) Microticks.
DEC	[31:24]	rw	**Decoding Correction** (pDecodingCorrection)[1] Configures the decoding correction value used to determine the primary time reference point. Valid values are 14 to 143 (E_H to $8F_H$) Microticks.
0	[23:21]	r	**Reserved** Returns 0 if read; should be written with 0.

1) This bit can be updated in “DEFAULT_CONFIG” or “CONFIG” state only!

GTU Configuration Register 6 (GTUC06)

The Communication Controller accepts modifications of the register in “DEFAULT_CONFIG” or “CONFIG” state only.

GTUC06
GTU Configuration Register 6 ($00B4_H$) Reset Value: $0002\ 0000_H$



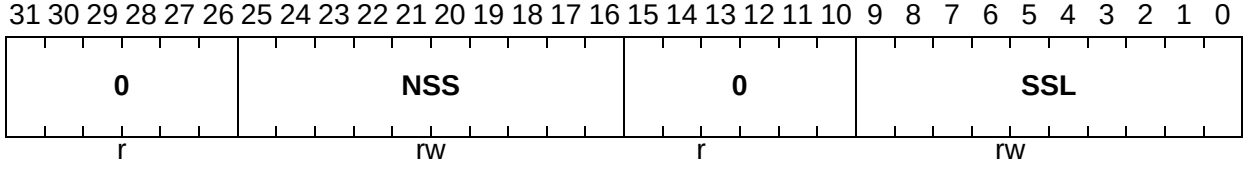
Field	Bits	Type	Description
ASR	[10:0]	rw	**Accepted Startup Range**[1)] (pdAcceptedStartupRange) Number of Microticks constituting the expanded range of measured deviation for startup Frames during integration. Valid values are 0 to 1875 (0_H to 753_H) Microticks.
MOD	[26:16]	rw	**Maximum Oscillator Drift** (pdMaxDrift)[1)] Maximum drift offset between two nodes that operate with unsynchronized clocks over one communication cycle in Microticks. Valid values are 2 to 1923 (2_H to 783_H) Microticks.
0	[15:11], [31:27]	r	**Reserved** Returns 0 if read; should be written with 0.

1) This bit can be updated in “DEFAULT_CONFIG” or “CONFIG” state only!

GTU Configuration Register 7 (GTUC07)

The Communication Controller accepts modifications of the register in “DEFAULT_CONFIG” or “CONFIG” state only.

GTUC07
GTU Configuration Register 7 (00B8$_H$) Reset Value: 0002 0004$_H$



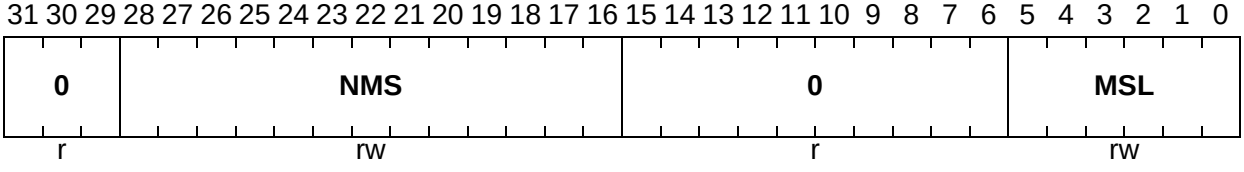
Field	Bits	Type	Description
SSL	[9:0]	rw	**Static Slot Length** (gdStaticSlot)[1)] Configures the duration of a static slot in Macroticks. The static slot length must be identical in all nodes of a cluster. Valid values are 4 to 659 (4_H to 293_H) Macroticks.
NSS	[25:16]	rw	**Number of Static Slots** (gNumberOfStaticSlots)[1)] Configures the number of static slots in a cycle. At least 2 coldstart nodes must be configured to startup a FlexRay™ network. The number of static slots must be identical in all nodes of a cluster. Valid values are 2 to 1023 (2_H to $3FF_H$).
0	[15:10], [31:26]	r	**Reserved** Returns 0 if read; should be written with 0.

1) This bit can be updated in “DEFAULT_CONFIG” or “CONFIG” state only!

GTU Configuration Register 8 (GTUC08)

The Communication Controller accepts modifications of the register in “DEFAULT_CONFIG” or “CONFIG” state only.

GTUC08
GTU Configuration Register 8 (00BC$_H$) Reset Value: 0000 0002$_H$



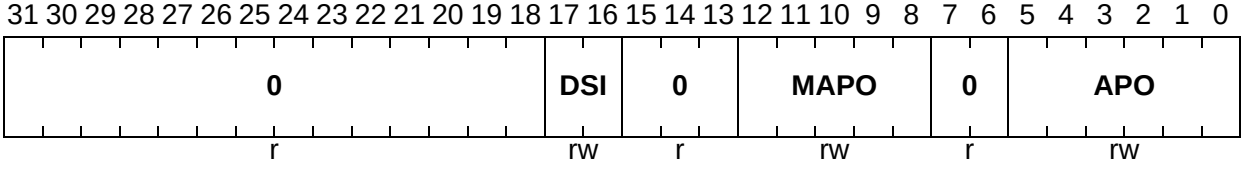
Field	Bits	Type	Description
MSL	[5:0]	rw	**Minislot Length** (gdMinislot)[1)] Configures the duration of a minislot in Macroticks. The minislot length must be identical in all nodes of a cluster. Valid values are 2 to 63 (2_H to $3F_H$) Macroticks.
NMS	[28:16]	rw	**Number of Minislots** (gNumberOfMinislots)[1)] Configures the number of minislots within the dynamic segment of a cycle. The number of minislots must be identical in all nodes of a cluster. Valid values are 0 to 7986 (0_H to $1F32_H$).
0	[15:6], [31:29]	r	**Reserved** Returns 0 if read; should be written with 0.

1) This bit can be updated in “DEFAULT_CONFIG” or “CONFIG” state only!

GTU Configuration Register 9 (GTUC09)

The Communication Controller accepts modifications of the register in “DEFAULT_CONFIG” or “CONFIG” state only.

GTUC09
GTU Configuration Register 9 ($00C0_H$) Reset Value: $0000\ 0101_H$



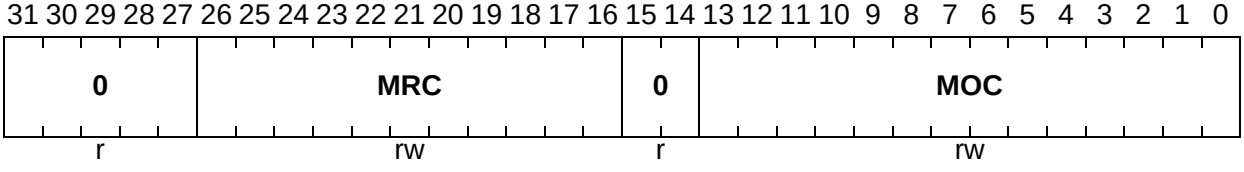
Field	Bits	Type	Description
APO	[5:0]	rw	**Action Point Offset** (gdActionPointOffset)[1)] Configures the action point offset in Macroticks within static slots and symbol window. Must be identical in all nodes of a cluster. Valid values are 1 to 63 (1_H to $3F_H$) Macroticks.
MAPO	[12:8]	rw	**Minislot Action Point Offset**[1)] (gdMinislotActionPointOffset) Configures the action point offset in Macroticks within the minislots of the dynamic segment. Must be identical in all nodes of a cluster. Valid values are 1 to 31 (1_H to $1F_H$) Macroticks.
DSI	[17:16]	rw	**Dynamic Slot Idle Phase**[1)] (gdDynamicSlotIdlePhase) The duration of the dynamic slot idle phase has to be greater or equal than the idle detection time. Must be identical in all nodes of a cluster. Valid values are 0 to 2 Minislot.
0	[7:6], [15:13], [31:18]	r	**Reserved** Returns 0 if read; should be written with 0.

1) This bit can be updated in “DEFAULT_CONFIG” or “CONFIG” state only!

GTU Configuration Register 10 (GTUC10)

The Communication Controller accepts modifications of the register in “DEFAULT_CONFIG” or “CONFIG” state only.

GTUC10
GTU Configuration Register 10 ($00C4_H$) Reset Value: $0002\ 0005_H$



Field	Bits	Type	Description
MOC	[13:0]	rw	**Maximum Offset Correction**[1] (pOffsetCorrectionOut) Holds the maximum permitted offset correction value to be applied by the internal clock synchronization algorithm (absolute value). The Communication Controller checks only the internal offset correction value against the maximum offset correction value. Valid values are 5 to 15266 (5_H to $3BA2_H$) Microticks.
MRC	[26:16]	rw	**Maximum Rate Correction**[1] (pRateCorrectionOut) Holds the maximum permitted rate correction value to be applied by the internal clock synchronization algorithm. The communication controller checks only the internal rate correction value against the maximum rate correction value (absolute value). Valid values are 2 to 1923 (2_H to 783_H) Microticks.
0	[15:14], [31:27]	r	**Reserved** Returns 0 if read; should be written with 0.

1) This bit can be updated in “DEFAULT_CONFIG” or “CONFIG” state only!

GTU Configuration Register 11 (GTUC11)

GTUC11
GTU Configuration Register 11 ($00C8_H$) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					ERC			0					EOC			0							ERCC								EOCC	
r					rw			r					rw			r							rw								rw	

Field	Bits	Type	Description
EOCC	[1:0]	rw	**External Offset Correction Control** (pExternOffsetControl) By writing to EOCC the external offset correction is enabled as specified below. Should be modified only outside network idle time (NIT). 00_B No external clock correction 01_B No external clock correction 10_B External offset correction value subtracted from calculated offset correction value 11_B External offset correction value added to calculated offset correction value
ERCC	[9:8]	rw	**External Rate Correction Control** (pExternRateControl) By writing to ERCC the external rate correction is enabled as specified below. Should be modified only outside network idle time (NIT). 00_B No external rate correction 01_B No external rate correction 10_B External rate correction value subtracted from calculated rate correction value 11_B External rate correction value added to calculated rate correction value
EOC	[18:16]	rw	**External Offset Correction**[1)] (pExternOffsetCorrection) Holds the external clock offset correction value in Microticks to be applied by the internal synchronization algorithm. The value is subtracted / added from / to the calculated offset correction value. The value is applied during network idle time (NIT). May be modified in “DEFAULT_CONFIG” or “CONFIG” state only. Valid values are 0 to 7 Microticks.

Field	Bits	Type	Description
ERC	[26:24]	rw	**External Rate Correction**[1] (pExternRateCorrection) Holds the external rate correction value in Microticks to be applied by the internal clock synchronization algorithm. The value is subtracted / added from / to the calculated rate correction value. The value is applied during network idle time (NIT). May be modified in “DEFAULT_CONFIG” or “CONFIG” state only. Valid values are 0 to 7 Microticks.
0	[7:2], [15:10], [23:19], [31:27]	r	**Reserved** Returns 0 if read; should be written with 0.

1) This bit can be updated in “DEFAULT_CONFIG” or “CONFIG” state only!

20.5.2.5 Communication Controller Status Registers

During 8/16-bit accesses to status variables coded with more than 8/16-bit, the variable might be updated by the Communication Controller between two accesses (non-atomic read accesses). The status vector may change faster than the Host can poll the status vector, depending on f_{CLC_ERAY} frequency.

Communication Controller Status Vector (CCSV)

CCSV

Communication Controller Status Vector(0100_H) **Reset Value: $0010\ 4000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		PSL						RCA					WSV		
r		rh						rh					rh		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	CSI	CSAI	CSNI	0		SLM		HRQ	FSI	POCS					
r	rh	rh	rh	r		rh		rh	rh	rh					

Field	Bits	Type	Description
POCS	[5:0]	rh	**Protocol Operation Control Status** Indicates the actual state of operation of the Communication Controller Protocol Operation Control 000000_B “DEFAULT_CONFIG” state 000001_B “READY” state 000010_B “NORMAL_ACTIVE” state 000011_B “NORMAL_PASSIVE” state 000100_B “HALT” state 000101_B “MONITOR_MODE” state 000110_B … 001110_B are reserved. 001111_B “CONFIG” state Indicates the actual state of operation of the POC in the wakeup path 010000_B WAKEUP_STANDBY state 010001_B “WAKEUP_LISTEN” state 010010_B “WAKEUP_SEND” state 010011_B “WAKEUP_DETECT” state 010100_B … 011111_B are reserved. Indicates the actual state of operation of the POC in the startup path 100000_B “STARTUP_PREPARE” state 100001_B “COLDSTART_LISTEN” state 100010_B “COLDSTART_COLLISION_RESOLUTION state 100011_B “COLDSTART_CONSISTENCY_CHECK” state 100100_B “COLDSTART_GAP state 100101_B “COLDSTART_JOIN” State 100110_B “INTEGRATION_COLDSTART_CHECK” state 100111_B “INTEGRATION_LISTEN” state 101000_B “INTEGRATION_CONSISTENCY_CHECK” state 101001_B “INITIALIZE_SCHEDULE” state 101010_B “ABORT_STARTUP” state 101011_B “STARTUP_SUCCESS” state 101100_B … 111111_B are reserved.

Field	Bits	Type	Description
FSI	6	rh	**Freeze Status Indicator** (vPOC!Freeze**)** Indicates that the POC has entered the “HALT” state due to CHI command “FREEZE” or due to an error condition requiring an immediate POC halt. Reset by transition from “HALT” to “DEFAULT_CONFIG” state.
HRQ	7	rh	**Halt Request** (vPOC!CHIHaltRequest) Indicates that a request from the Host has been received to halt the POC at the end of the communication cycle. Reset by transition from “HALT” to “DEFAULT_CONFIG” state or when entering “READY” state.
SLM	[9:8]	rh	**Slot Mode** (vPOC!SlotMode) Indicates the actual slot mode of the POC in states READY, STARTUP, NORMAL_ACTIVE, and NORMAL_PASSIVE. Default is “SINGLE”. Changes to “ALL”, depending on configuration bit SUCC1.TSM. In “NORMAL_ACTIVE” or “NORMAL_PASSIVE” state the CHI command “ALL_SLOTS” will change the slot mode from “SINGLE” over “ALL_PENDING” to “ALL”. Set to SINGLE in all other states. 00_B SINGLE 01_B Reserved 10_B ALL_PENDING 11_B ALL
CSNI	12	rh	**Coldstart Noise Indicator** (vPOC!ColdstartNoise) Indicates that the cold start procedure occurred under noisy conditions. Reset by CHI command “RESET_STATUS_INDICATORS” or by transition from “HALT” to “DEFAULT_CONFIG” state or from “READY” to “STARTUP” state.
CSAI	13	rh	**Coldstart Abort Indicator** Coldstart aborted. Reset by CHI command “RESET_STATUS_INDICATORS” or by transition from “HALT” to “DEFAULT_CONFIG” state or from “READY” to “STARTUP” state.

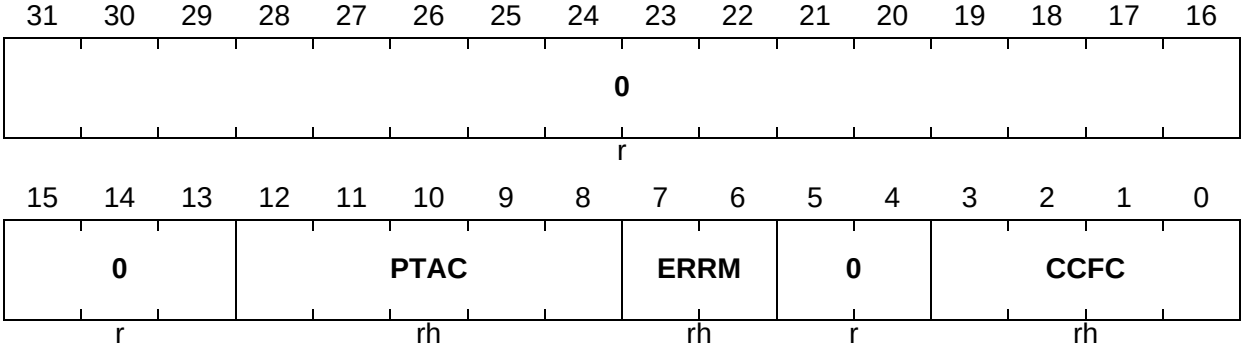
Field	Bits	Type	Description
CSI	14	rh	**Cold Start Inhibit** (vColdStartInhibit) Indicates that the node is disabled from cold starting. The flag is set whenever the POC enters “READY” state due to CHI command “READY”. The flag has to be reset under control of the Host by CHI command “ALLOW_COLDSTART” (SUCC1.CMD = 1001_B). 0_B Cold starting of node enabled 1_B Cold starting of node disabled
WSV	[18:16]	rh	**Wakeup Status** (vPOC!WakeupStatus) Indicates the status of the current wakeup attempt. Reset by CHI command “RESET_STATUS_INDICATORS” or by transition from “HALT” to “DEFAULT_CONFIG” state or from “READY” to “STARTUP” state. 000_B UNDEFINED. Wakeup not yet executed by the Communication Controller. 001_B RECEIVED_HEADER. Set when the Communication Controller finishes wakeup due to the reception of a Frame Header without coding violation on either channel in “WAKEUP_LISTEN” state. 010_B RECEIVED_WUP. Set when the Communication Controller finishes wakeup due to the reception of a valid wakeup pattern on the configured wakeup channel in “WAKEUP_LISTEN” state. 011_B COLLISION_HEADER. Set when the Communication Controller stops wakeup due to a detected collision during wakeup pattern transmission by receiving a valid Header on either channel. 100_B COLLISION_WUP. Set when the Communication Controller stops wakeup due to a detected collision during wakeup pattern transmission by receiving a valid wakeup pattern on the configured wakeup channel. 101_B COLLISION_UNKNOWN. Set when the Communication Controller stops wakeup by leaving “WAKEUP_DETECT” state after expiration of the wakeup timer without receiving a valid wakeup pattern or a valid Frame Header. 110_B TRANSMITTED. Set when the Communication Controller has successfully completed the transmission of the wakeup pattern. 111_B Reserved

Field	Bits	Type	Description
RCA	[23:19]	rh	**Remaining Coldstart Attempts** (vRemainingColdstartAttempts) Indicates the number of remaining coldstart attempts. The RUN command resets this counter to the maximum number of coldstart attempts as configured by SUCC1.CSA.
PSL	[29:24]	rh	**POC Status Log** Status of CCSV.POCS immediately before entering “HALT” state. Set when entering “HALT” state. Set to “HALT” when FREEZE command is applied during “HALT” state. Reset to 000000_B when leaving “HALT” state.
0	[11:10], 15, [31:30]	rh	**Reserved** Returns 0 if read; should be written with 0.

Communication Controller Error Vector (CCEV)

Reset by transition from “HALT” to “DEFAULT_CONFIG” state or when entering “READY” state.

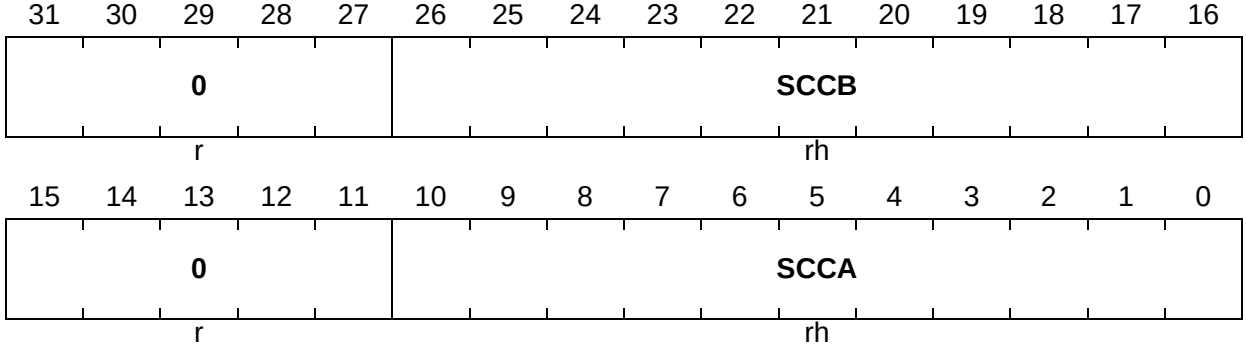
CCEV
Communication Controller Error Vector (0104_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
CCFC	[3:0]	rh	**Clock Correction Failed Counter** (vClockCorrectionFailed) The Clock Correction Failed Counter is incremented by one at the end of any odd communication cycle where either the missing offset correction error or missing rate correction error are active. The Clock Correction Failed Counter is reset to 0 at the end of an odd communication cycle if neither the offset correction failed nor the rate correction failed errors are active. The Clock Correction Failed Counter stops at 15.
ERRM	[7:6]	rh	**Error Mode** (vPOC!ErrorMode) Indicates the actual error mode of the POC. 00_B “ACTIVE” (green) 01_B “PASSIVE” (yellow) 10_B “COMM_HALT” (red) 11_B Reserved
PTAC	[12:8]	rh	**Passive to Active Count** (vAllowPassiveToActive) Indicates the number of consecutive even / odd cycle pairs that have passed with valid rate and offset correction terms, while the node is waiting to transit from “NORMAL_PASSIVE” state to “NORMAL_ACTIVE” state. The transition takes place when PTAC equals SUCC1.PTA.
0	[5:4], [31:13]	r	**Reserved** Returns 0 if read; should be written with 0.

Slot Counter Value (SCV)

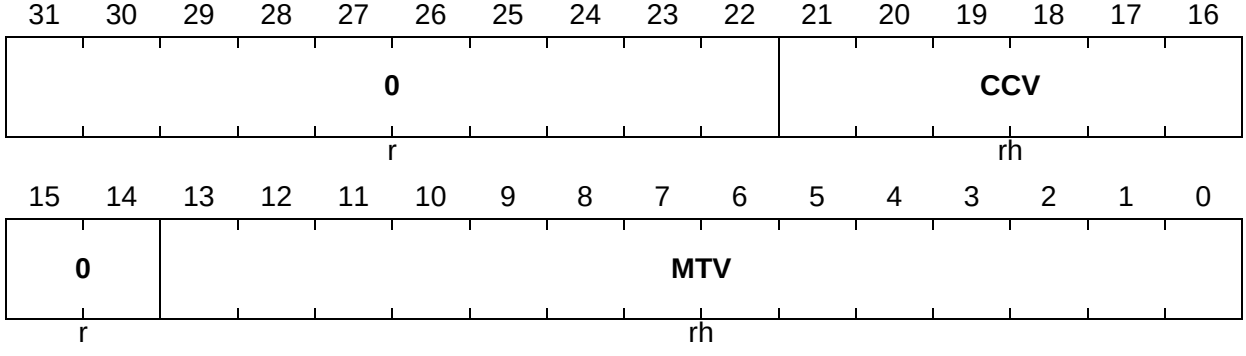
SCV
Slot Counter Value **(0110_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
SCCA	[10:0]	rh	**Slot Counter Channel A** (vSlotCounter[A]**)** Current slot counter value on channel A. The value is incremented by the Communication Controller and reset at the start of a communication cycle. Valid values are 0 to 2047 (0_H to $7FD_H$).
SCCB	[26:16]	rh	**Slot Counter Channel B** (vSlotCounter[B]**)** Current slot counter value on channel B. The value is incremented by the Communication Controller and reset at the start of a communication cycle. Valid values are 0 to 2047 (0_H to $7FD_H$).
0	[15:11], [31:27]	r	**Reserved** Returns 0 if read; should be written with 0.

Macrotick and Cycle Counter Value (MTCCV)

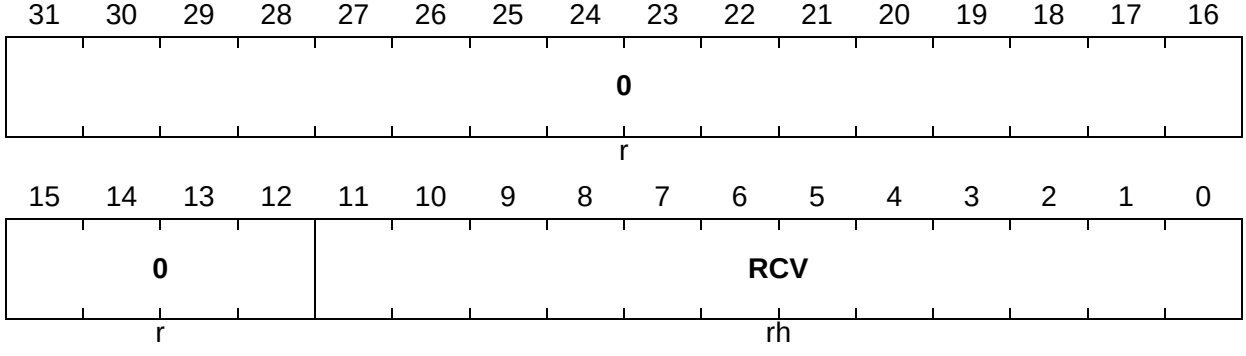
MTCCV
Macrotick and Cycle Counter Value (0114$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
MTV	[13:0]	rh	**Macrotick Value** (vMacrotick) Current Macrotick value. The value is incremented by the Communication Controller and reset at the start of a communication cycle. Valid values are 0 to 16000 (0_H to $3E80_H$).
CCV	[21:16]	rh	**Cycle Counter Value** (vCycleCounter) Current cycle counter value. The value is incremented by the Communication Controller at the start of a communication cycle. Valid values are 0 to 63 (0_H to $3F_H$).
0	[15:14], [31:22]	r	**Reserved** Returns 0 if read; should be written with 0.

Rate Correction Value (RCV)

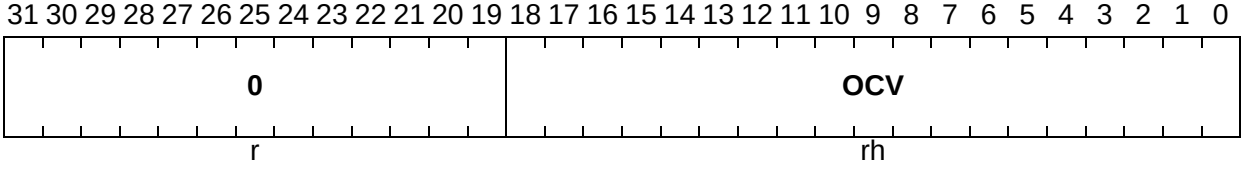
RCV
Rate Correction Value **(0118_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
RCV	[11:0]	rh	**Rate Correction Value** (vRateCorrection) Rate correction value (two's complement). Calculated internal rate correction value before limitation. If the RCV value exceeds the limits defined by GTUC10.MRC, flag SFS.RCLR is set to 1.
0	[31:12]	r	**Reserved** Returns 0 if read; should be written with 0.

Offset Correction Value (OCV)

OCV
Offset Correction Value **($011C_H$)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
OCV	[18:0]	rh	**Offset Correction Value** (vOffsetCorrection) Offset correction value (two's complement). Calculated internal offset correction value before limitation. If the OCV value exceeds the limits defined by GTUC10.MOC flag SFS.OCLR is set to 1.
0	[31:19]	r	**Reserved** Returns 0 if read; should be written with 0.

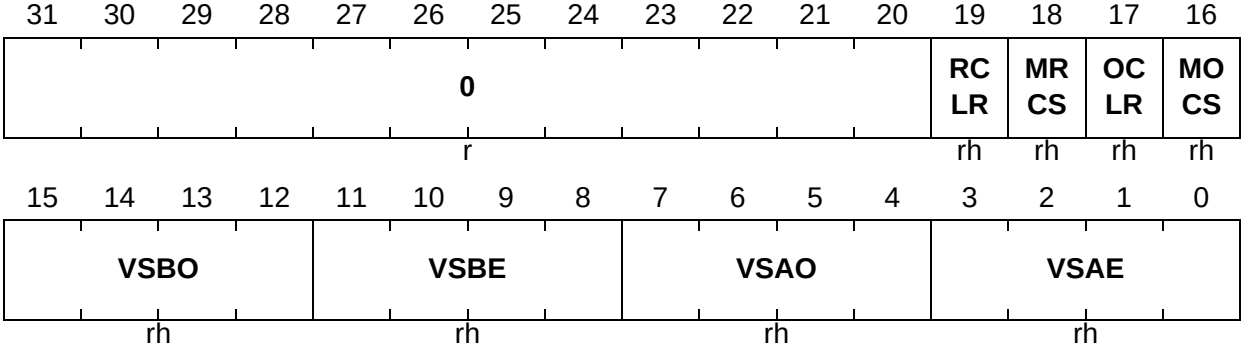
Note: The external rate / offset correction value is added to the limited rate / offset correction value.

SYNC Frame Status (SFS)

The maximum number of valid SYNC Frames in a communication cycle is 15.

SFS

SYNC Frame Status **(0120_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
VSAE	[3:0]	rh	**Valid SYNC Frames Channel A, even communication cycle** Holds the number of valid SYNC Frames received on channel A in the even communication cycle. If transmission of SYNC Frames is enabled by SUCC1.TXSY the value is incremented by one. The value is updated during the network idle time (NIT) of each even communication cycle. This bit field is only valid if the channel A is assigned to the Communication Controller by SUCC1.CCHA.
VSAO	[7:4]	rh	**Valid SYNC Frames Channel A, odd communication cycle** Holds the number of valid SYNC Frames received on channel A in the odd communication cycle. If transmission of SYNC Frames is enabled by SUCC1.TXSY the value is incremented by one. The value is updated during the network idle time (NIT) of each odd communication cycle. This bit field is only valid if the channel A is assigned to the Communication Controller by SUCC1.CCHA.
VSBE	[11:8]	rh	**Valid SYNC Frames Channel B, even communication cycle** Holds the number of valid SYNC Frames received on channel B in the even communication cycle. If transmission of SYNC Frames is enabled by SUCC1.TXSY the value is incremented by one. The value is updated during the network idle time (NIT) of each even communication cycle. This bit field is only valid if the channel B is assigned to the Communication Controller by SUCC1.CCHB.

Field	Bits	Type	Description
VSBO	[15:12]	rh	**Valid SYNC Frames Channel B, odd communication cycle** Holds the number of valid SYNC Frames received on channel B in the odd communication cycle. If transmission of SYNC Frames is enabled by SUCC1.TXSY the value is incremented by one. The value is updated during the network idle time (NIT) of each odd communication cycle. This bit field is only valid if the channel B is assigned to the Communication Controller by SUCC1.CCHB.
MOCS	16	rh	**Missing Offset Correction Signal** The Missing Offset Correction flag signals to the Host, that no offset correction calculation can be performed because no SYNC Frames were received. The flag is updated by the Communication Controller at start of offset correction phase. 0_B Offset correction signal valid 1_B Missing offset correction signal
OCLR	17	rh	**Offset Correction Limit Reached** The Offset Correction Limit Reached flag signals to the Host, that the offset correction value has exceeded its limit as defined by GTUC10.MOC. The flag is updated by the Communication Controller at start of offset correction phase. 0_B Offset correction below limit 1_B Offset correction limit reached
MRCS	18	rh	**Missing Rate Correction Signal** The Missing Rate Correction Flag signals to the Host, that no rate correction calculation can be performed because no pairs of even / odd SYNC Frames were received. The flag is updated by the Communication Controller at start of offset correction phase. 0_B Rate correction signal valid 1_B Missing rate correction signal
RCLR	19	rh	**Rate Correction Limit Reached** The Rate Correction Limit Reached flag signals to the Host, that the rate correction value has exceeded its limit.as defined by GTUC10.MRC. The flag is updated by the Communication Controller at start of offset correction phase. 0_B Rate correction below limit 1_B Rate correction limit reached
0	[31:20]	r	**Reserved** Returns 0 if read; should be written with 0.

Symbol Window and network idle time (NIT) Status (SWNIT)

Symbol window related status information. Updated by the Communication Controller at the end of the symbol window for each channel. During startup the status data is not updated.

Note: MTSA and MTSB may be changed outside "DEFAULT_CONFIG" or "CONFIG" state when the write to SUC Configuration Register 1 (SUCC1) register is directly preceded by the unlock sequence as described in ***"Lock Register (LCK)" on Page 20-32****. This may be combined with CHI command SEND_MTS. If both bits MTSA and MTSB are set to 1 an MTS symbol will be transmitted on both channels when requested by writing SUCC1.CMD = 1000_B*

SWNIT
Symbol Window and Network Idle Time Status
(0124_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				SBNB	SENB	SBNA	SENA	MTSB	MTSA	TCSB	SBSB	SESB	TCSA	SBSA	SESA
r				rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
SESA	0	rh	**Syntax Error in Symbol Window Channel A** (vSS!SyntaxErrorA) 0_B No syntax error detected 1_B Syntax error during symbol window detected on channel A
SBSA	1	rh	**Slot Boundary Violation in Symbol Window Channel A** (vSS!BViolationA) 0_B No slot boundary violation detected 1_B Slot boundary violation during symbol window detected on channel A

Field	Bits	Type	Description
TCSA	2	rh	**Transmission Conflict in Symbol Window Channel A** (vSS!TxConflictA) 0_B No transmission conflict detected 1_B Transmission conflict in symbol window detected on channel A
SESB	3	rh	**Syntax Error in Symbol Window Channel B** (vSS!SyntaxErrorB) 0_B No syntax error detected 1_B Syntax error during symbol window detected on channel B
SBSB	4	rh	**Slot Boundary Violation in Symbol Window Channel B** (vSS!BViolationB) 0_B No slot boundary violation detected 1_B Slot boundary violation during symbol window detected on channel B
TCSB	5	rh	**Transmission Conflict in Symbol Window Channel B** (vSS!TxConflictB) 0_B No transmission conflict detected 1_B Transmission conflict in symbol window detected on channel B
MTSA	6	rh	**MTS Received on Channel A** (vSS!ValidMTSA)[1)] Media Access Test symbol received on channel A during the proceeding symbol window. Updated by the Communication Controller for each channel at the end of the symbol window. When this bit is set to 1, also interrupt flag SIR.MTSA is set to 1. 0_B No MTS symbol received on channel A 1_B MTS symbol received on channel A
MTSB	7	rh	**MTS Received on Channel B** (vSS!ValidMTSB)[1)] Media Access Test symbol received on channel B during the proceeding symbol window. Updated by the Communication Controller for each channel at the end of the symbol window. When this bit is set to 1, also interrupt flag SIR.MTSB is set to 1. 0_B No MTS symbol received on channel B 1_B MTS symbol received on channel B

Field	Bits	Type	Description
SENA	8	rh	**Syntax Error during network idle time (NIT) Channel A** (vSS!SyntaxErrorA) Updated by the Communication Controller channel A at the end of the NIT. 0_B No syntax error detected 1_B Syntax error during network idle time (NIT) detected on channel A
SBNA	9	rh	**Slot Boundary Violation during network idle time (NIT) Channel A** (vSS!BViolationA) Updated by the Communication Controller channel A at the end of the NIT. 0_B No slot boundary violation detected 1_B Slot boundary violation during network idle time (NIT) detected on channel A
SENB	10	rh	**Syntax Error during network idle time (NIT) Channel B** (vSS!SyntaxErrorB) Updated by the Communication Controller channel B at the end of the NIT. 0_B No syntax error detected 1_B Syntax error during network idle time (NIT) detected on channel B
SBNB	11	rh	**Slot Boundary Violation during network idle time (NIT) Channel B** (vSS!BViolationB) Updated by the Communication Controller channel B at the end of the NIT. 0_B No slot boundary violation detected 1_B Slot boundary violation during network idle time (NIT) detected on channel B
0	[31:12]	r	**Reserved** Returns 0 if read; should be written with 0.

1) MTSA and MTSB may also be changed outside “DEFAULT_CONFIG” or “CONFIG” state when the write to SUCC1 register is directly preceded by the unlock sequence as described in “Lock Register (LCK)”. This may be combined with CHI command SEND_MTS. If both bits MTSA and MTSB are set to 1 an MTS symbol will be transmitted on both channels when requested by writing SUCC1.CMD = 1000_B.

Aggregated Channel Status (ACS)

The aggregated channel status provides the Host with an accrued status of channel activity for all communication slots regardless of whether they are assigned for transmission or subscribed for reception. The aggregated channel status also includes status data from the symbol window and the network idle time. The status data is updated (set) after each slot and aggregated until it is reset by the Host. During startup the status data is not updated. A flag is cleared by writing a 1 to the corresponding bit position. Writing a 0 has no effect on the flag. An application reset will also clear the register.

ACS

Aggregated Channel Status (0128$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0			SBVB	CIB	CEDB	SEDB	VFRB	0			SBVA	CIA	CEDA	SEDA	VFRA
r			rwh	rwh	rwh	rwh	rwh	r			rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
VFRA	0	rwh	**Valid Frame Received on Channel A** (vSS!ValidFrameA) One or more valid Frames were received on channel A in any static or dynamic slot during the observation period. 0_B No valid Frame received 1_B Valid Frame(s) received on channel A
SEDA	1	rwh	**Syntax Error Detected on Channel A** (vSS!SyntaxErrorA) One or more syntax errors in static or dynamic slots, symbol window, and network idle time (NIT) were observed on channel A. 0_B No syntax error observed 1_B Syntax error(s) observed on channel A
CEDA	2	rwh	**Content Error Detected on Channel A** (vSS!ContentErrorA) One or more Frames with a content error were received on channel A in any static or dynamic slot during the observation period. 0_B No Frame with content error received 1_B Frame(s) with content error received on channel A

Field	Bits	Type	Description
CIA	3	rwh	**Communication Indicator Channel A** One or more valid Frames were received on channel A in slots that also contained any additional communication during the observation period, i.e. one or more slots received a valid Frame AND had any combination of either syntax error OR content error OR slot boundary violation. 0_B No valid Frame(s) received in slots containing any additional communication 1_B Valid Frame(s) received on channel A in slots containing any additional communication
SBVA	4	rwh	**Slot Boundary Violation on Channel A** (vSS!BViolationA) One or more slot boundary violations were observed on channel A at any time during the observation period (static or dynamic slots, symbol window, and network idle time NIT). 0_B No slot boundary violation observed 1_B Slot boundary violation(s) observed on channel A
VFRB	8	rwh	**Valid Frame Received on Channel B** (vSS!ValidFrameB) One or more valid Frames were received on channel B in any static or dynamic slot during the observation period. 0_B No valid Frame received 1_B Valid Frame(s) received on channel B
SEDB	9	rwh	**Syntax Error Detected on Channel B** (vSS!SyntaxErrorB) One or more syntax errors in static or dynamic slots, symbol window, and network idle time (NIT) were observed on channel B. 0_B No syntax error observed 1_B Syntax error(s) observed on channel B
CEDB	10	rwh	**Content Error Detected on Channel B** (vSS!ContentErrorB) One or more Frames with a content error were received on channel B in any static or dynamic slot during the observation period. 0_B No Frame with content error received 1_B Frame(s) with content error received on channel B

Field	Bits	Type	Description
CIB	11	rwh	**Communication Indicator Channel B** One or more valid Frames were received on channel B in slots that also contained any additional communication during the observation period, i.e. one or more slots received a valid Frame AND had any combination of either syntax error OR content error OR slot boundary violation. 0_B No valid Frame(s) received in slots containing any additional communication 1_B Valid Frame(s) received on channel B in slots containing any additional communication
SBVB	12	rwh	**Slot Boundary Violation on Channel B** (vSS!BViolationB) One or more slot boundary violations were observed on channel B at any time during the observation period (static or dynamic slots, symbol window, and network idle time NIT). 0_B No slot boundary violation observed 1_B Slot boundary violation(s) observed on channel B
0	[7:5], [31:13]	r	**Reserved** Returns 0 if read; should be written with 0.

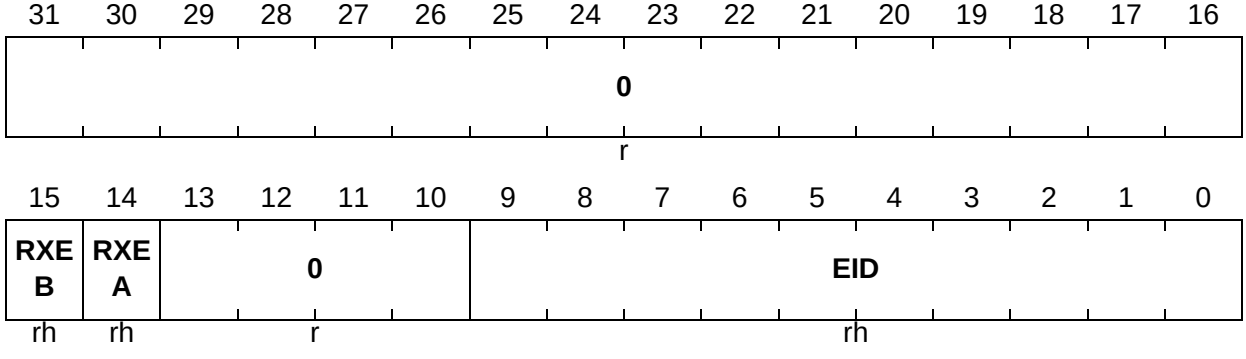
Note: The set condition of flags CIA and CIB is also fulfilled if there is only one single Frame in the slot and the slot boundary at the end of the slot is reached during the Frames channel idle recognition phase. When one of the flags SEDB, CEDB, CIB, SBVB changes from 0 to 1, service request flag EIR.EDB is set to 1.When one of the flags SEDA, CEDA, CIA, SBVA changes from 0 to 1, service request flag EIR.EDA is set to 1.

Even Sync ID [01…15] (ESIDnn)

Registers Even Sync ID nn (ESIDnn, nn=01-15) hold the Frame IDs of the SYNC Frames received in **even** communication cycles, sorted in ascending order, with register ESID01 holding the lowest received SYNC Frame ID. If the node itself transmits a SYNC Frame in an even communication cycle, register ESID01 holds the respective SYNC Frame ID as configured in Message Buffer 0 and the flags RXEA, RXEB are set. The value is updated during the network idle time (NIT) of each even communication cycle.

ESIDnn (nn = 01-15)
Even Sync ID Symbol Window nn

($012C_H$ + nn * 4) **Reset Value: 0000 0000_H**



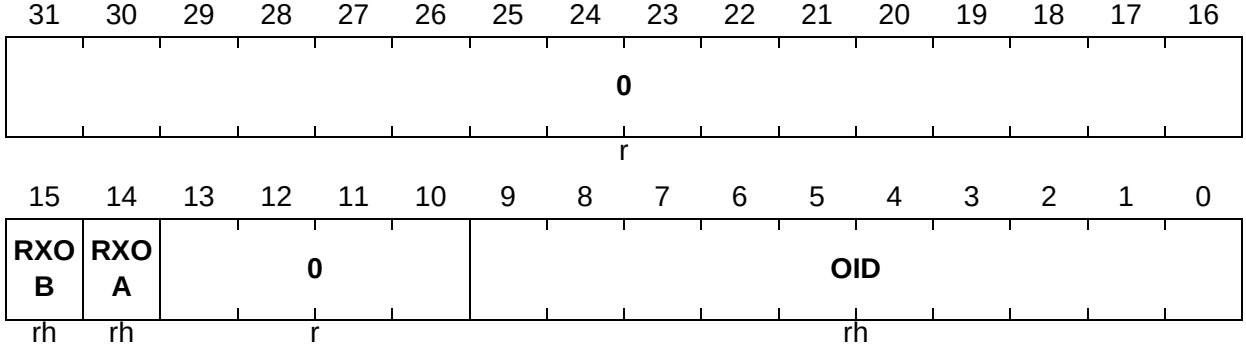
Field	Bits	Type	Description
EID	[9:0]	rh	**Even Sync ID** (vsSyncIDListA,B even) SYNC Frame ID even communication cycle.
RXEA	14	rh	**Received/Configured Even Sync ID on Channel A** Signals that a SYNC Frame corresponding to the stored even sync ID was received on channel A or that the node is configured to be a sync node with key slot = EID (ESID1 only). 0_B SYNC Frame not received on channel A / node configured to transmit SYNC Frames 1_B SYNC Frame received on channel A/ node not configured to transmit SYNC Frames
RXEB	15	rh	**Received/Configured Even Sync ID on Channel B** Signals that a SYNC Frame corresponding to the stored even sync ID was received on channel B or that the node is configured to be a sync node with key slot = EID (ESID1 only). 0_B SYNC Frame not received on channel B / node configured to transmit SYNC Frames 1_B SYNC Frame received on channel B / node not configured to transmit SYNC Frames

Field	Bits	Type	Description
0	[13:10], [31:16]	r	**Reserved** Returns 0 if read; should be written with 0.

Odd Sync ID [01…15] (OSIDnn)

he Odd Sync ID nn (OSIDnn, nn=01-15) hold the Frame IDs of the SYNC Frames received in **odd** communication cycles, sorted in ascending order, with register OSID01 holding the lowest received SYNC Frame ID. If the node itself transmits a SYNC Frame in an odd communication cycle, register OSID01 holds the respective SYNC Frame ID as configured in Message Buffer 0 and flags RXOA, RXOB are set. The value is updated during the network idle time (NIT) of each odd communication cycle.

OSIDnn (nn = 01-15)
Odd Sync ID Symbol Window nn ($016C_H$ + nn * 4) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
OID	[9:0]	rh	**Odd Sync ID** (vsSyncIDListA,B odd) SYNC Frame ID even communication cycle.
RXOA	14	rh	**Received Odd Sync ID on Channel A** Signals that a SYNC Frame corresponding to the stored odd sync ID was received on channel A or that the node is configured to be a sync node with key slot = OID (OSID1 only). 0_B SYNC Frame not received on channel A/ node configured to transmit SYNC Frames 1_B SYNC Frame received on channel A/ node not configured to transmit SYNC Frames
RXOB	15	rh	**Received Odd Sync ID on Channel B** Signals that a SYNC Frame corresponding to the stored odd sync ID was received on channel B or that the node is configured to be a sync node with key slot = OID (OSID1 only) 0_B SYNC Frame not received on channel B/ node configured to transmit SYNC Frames 1_B SYNC Frame received on channel B/ node not configured to transmit SYNC Frames

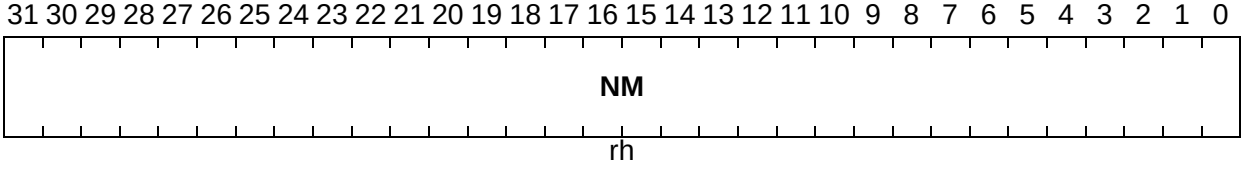
Field	Bits	Type	Description
0	[13:10], [31:16]	r	**Reserved** Returns 0 if read; should be written with 0.

Network Management Vector [1…3] (NMVx)

The three Network Management Vectors n (NMVx, x=1-3) registers hold the accrued Network Management (NM) vector (configurable 0 to 12 byte). The accrued Network Management (NM) vector is generated by the Communication Controller by bit-wise ORing each Network Management (NM) vector received (valid static Frames with PPI = 1) on each channel (see **“Network Management” on Page 20-213**). The Communication Controller updates the Network Management (NM) vector at the end of each communication cycle as long as the Communication Controller is either in “NORMAL_ACTIVE” or “NORMAL_PASSIVE” state. NMVx-bytes exceeding the configured Network Management (NM) vector length are not valid.

NMVx (x = 1-3)
Network Management Vector x

($01AC_H$ + x * 4) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
NM	[31:0]	rh	**Network Management Vector**

Table 20-5 below shows the assignment of the received payload’s data byte to the Network Management vector.

Table 20-5 Assignment of Data Byte to Network Management Vector

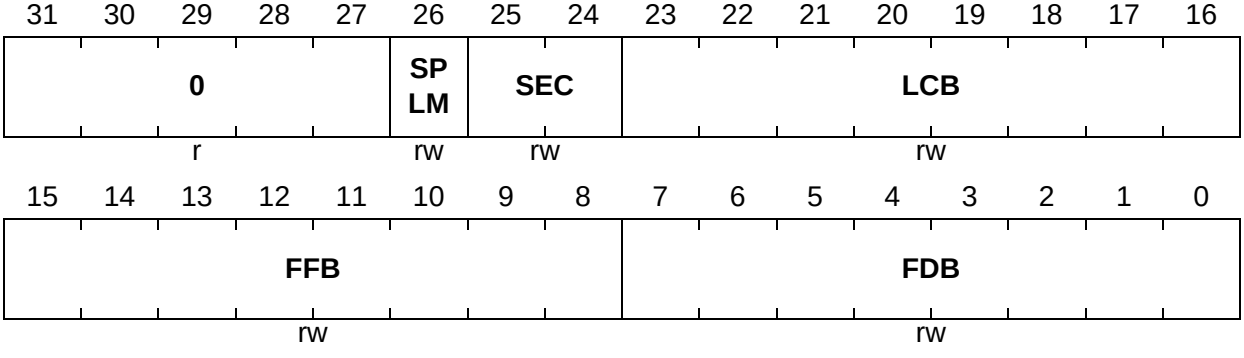
Bit / Word	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NM1	Data3								Data2								Data1								Data0							
NM2	Data7								Data6								Data5								Data4							
NM3	Data11								Data10								Data9								Data8							

20.5.2.6 Message Buffer Control Registers

Message RAM Configuration (MRC)

The Message RAM Configuration register defines the number of Message Buffers assigned to the static segment, dynamic segment, and FIFO. The register can be written during “DEFAULT_CONFIG” or “CONFIG” state only.

MRC
Message RAM Configuration (0300_H) Reset Value: $0180\ 0000_H$



Field	Bits	Type	Description
FDB	[7:0]	rw	**First Dynamic Buffer** May be modified in “DEFAULT_CONFIG” or “CONFIG” state only. 00_H No group of Message Buffers exclusively for the static segment configured 01_H …$7F_H$ Message Buffers 0 to FDB-1 reserved for static segment 80_H …FF_H No dynamic Message Buffers configured
FFB	[15:8]	rw	**First Buffer of FIFO** May be modified in “DEFAULT_CONFIG” or “CONFIG” state only. 00_H …$7E_H$ Message Buffers from FFB to LCB assigned to the FIFO $7F_H$ All Message Buffers assigned to the FIFO 80_H …FF_H No Message Buffers assigned to the FIFO

Field	Bits	Type	Description
LCB	[23:16]	rw	**Last Configured Buffer** May be only modified in “DEFAULT_CONFIG” or “CONFIG” state. 01_H …$7F_H$ Number of Message Buffers is LCB + 1 80_H …FF_H No Message Buffer configured
SEC	[25:24]	rw	**Secure Buffers** Not evaluated when the Communication Controller is in “DEFAULT_CONFIG” or “CONFIG” state. 00_B Reconfiguration of Message Buffers enabled with numbers < FFB enabled. *Note: In nodes configured for SYNC Frame transmission or for single slot mode operation Message Buffer 0 (and if SPLM = 1, also Message Buffer 1) Reconfiguration of all Message Buffers is always locked* 01_B Reconfiguration of Message Buffers with numbers < FDB and with numbers ≥ FFB locked and transmission of Message Buffers for static segment with numbers ≥ FDB disabled 10_B Reconfiguration of all Message Buffers locked 11_B Reconfiguration of all Message Buffers locked and transmission of Message Buffers for static segment with numbers ≥ FDB disabled
SPLM	26	rw	**SYNC Frame Payload Multiplex** This bit is only evaluated if the node is configured as sync node (SUCC1.TXSY = 1) or for single slot mode operation (SUCC1.TSM = 1). When this bit is set to 1 Message Buffers 0 and 1 are dedicated for SYNC Frame transmission with different payload data on channel A and B. When this bit is reset to 0, SYNC Frames are transmitted from Message Buffer 0 with the same payload data on both channels. Note that the channel filter configuration for Message Buffer 0 resp. Message Buffer 1 has to be chosen accordingly. 0_B Only Message Buffer 0 locked against reconfiguration 1_B Both Message Buffers 0 and 1 are locked against reconfiguration
0	[31:27]	r	**Reserved** Returns 0 if read; should be written with 0.

Note: In case the node is configured as sync node (SUCC1.TXSY = 1) or for single slot mode operation (SUCC1.TSM = 1), Message Buffer 0 resp. 1 is reserved for SYNC Frames or single slot Frames and have to be configured with the node-specific key slot ID. In case the node is neither configured as sync node nor for single slot operation Message Buffer 0 resp. 1 is treated like all other Message Buffers.

Table 20-6 Usage of the three Message Buffer Pointer

Message Buffer 0	⇓ Static Buffers		
Message Buffer 1			
	⇓ Static + Dynamic Buffers	⇐ **FDB**	
…			FIFO configured: **FFB** > **FDB**
	⇓ FIFO	⇐ **FFB**	No FIFO configured: **FFB** ≥ 128
Message Buffer N-1			**LCB** ≥ **FDB**, **LCB** ≥ **FFB**
Message Buffer N		⇐ **LCB**	

The programmer has to ensure that the configuration defined by FDB, FFB, and LCB is valid. **The Communication Controller does not check for erroneous configurations!**

Note: The maximum number of Header Sections is 128. This means a maximum of 128 Message Buffer can be configured. The maximum length of a Data Section is 254 byte. The length of the Data Section may be configured differently for each Message Buffer. For details see ***"Message RAM" on Page 20-239****.*

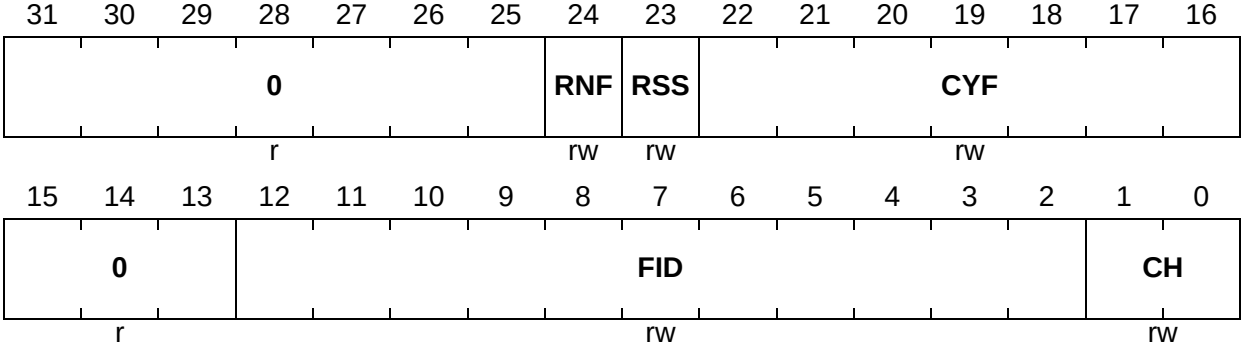
In case two or more Message Buffers are assigned to slot 1 by use of cycle filtering, all of them must be located either in the "Static Buffers" or at the beginning of the "Static + Dynamic Buffers" section.

The payload length configured and the length of the Data Section need to be configured identically for all Message Buffers belonging to the FIFO via WRHS2.PLC and WRHS3.DP. When the Communication Controller is not in "DEFAULT_CONFIG" or "CONFIG" state reconfiguration of Message Buffers belonging to the FIFO is locked.

FIFO Rejection Filter (FRF)

The FIFO Rejection Filter defines a user specified sequence of bits to which channel, Frame ID, and cycle count of the incoming Frames are compared. Together with the FIFO Rejection Filter Mask this register determines whether a message is rejected by the FIFO. The FRF register can be written during “DEFAULT_CONFIG” or “CONFIG” state only.

FRF
FIFO Rejection Filter **(0304_H)** **Reset Value: $0180\ 0000_H$**



Field	Bits	Type	Description
CH	[1:0]	rw	**Channel Filter** May be modified in “DEFAULT_CONFIG” or “CONFIG” state only. 00_B receive on both channels[1)] 01_B receive only on channel B 10_B receive only on channel A 11_B no reception
FID	[12:2]	rw	**Frame ID Filter** Determines the Frame ID to be rejected by the FIFO. With the additional configuration of register FRFM, the corresponding Frame ID filter bits are ignored, which results in further rejected Frame IDs. When FRFM.MFID is zero, a Frame ID filter value of zero means that no Frame ID is rejected. 000_H …$7FF_H$ Frame ID filter values

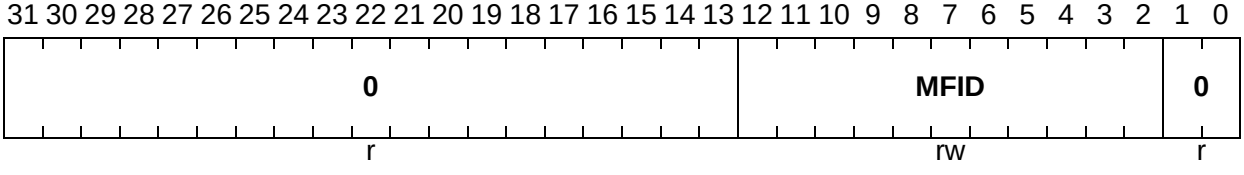
Field	Bits	Type	Description
CYF	[22:16]	rw	**Cycle Counter Filter** The 7-bit cycle counter filter determines the cycle set to which Frame ID and channel rejection filter are applied. In cycles **not** belonging to the cycle set specified by CYF, all Frames are rejected. For details about the configuration of the cycle counter filter see **“Cycle Counter Filtering” on Page 20-215**. May be modified in “DEFAULT_CONFIG” or “CONFIG” state only.
RSS	23	rw	**Reject in Static Segment** If this bit is set, the FIFO is used only be used in dynamic segment. May be modified in “DEFAULT_CONFIG” or “CONFIG” state only. 0_B FIFO also used in static segment 1_B Reject messages for static segment
RNF	24	rw	**Reject NULL Frames** If this bit is set, received NULL Frames are not stored in the FIFO. May be modified in “DEFAULT_CONFIG” or “CONFIG” state only. 0_B NULL Frames are stored in the FIFO 1_B Reject all NULL Frames
0	[15:13], [31:25]	r	**Reserved** Returns 0 if read; should be written with 0.

1) If reception on both channels is configured, also in static segment always both Frames (from channel A and B) are stored in the FIFO, even if they are identical.

FIFO Rejection Filter Mask (FRFM)

The FIFO Rejection Filter Mask specifies which of the corresponding Frame ID filter bits are relevant for rejection filtering. If a bit is set, it indicates that the corresponding bit in the FRF register will not be considered for rejection filtering. The FRFM register can be written during “DEFAULT_CONFIG” or “CONFIG” state only.

FRFM
FIFO Rejection Filter Mask **(0308_H)** **Reset Value: $0000\ 0000_H$**

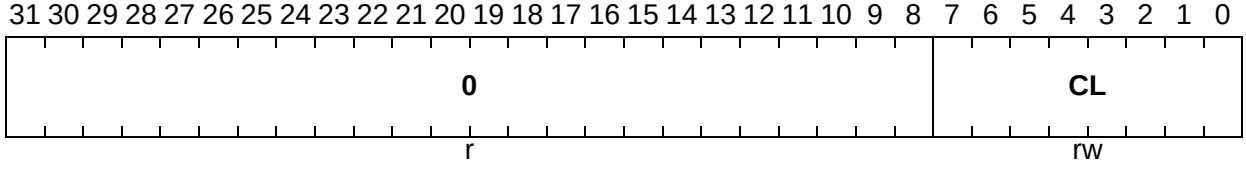


Field	Bits	Type	Description
MFID	[12:2]	rw	**Mask Frame ID Filter** May be modified in “DEFAULT_CONFIG” or “CONFIG” state only. 0_B Corresponding Frame ID filter bit is used for rejection filtering. 1_B Ignore corresponding Frame ID filter bit.
0	[1:0], [31:13]	r	**Reserved** Returns 0 if read; should be written with 0.

FIFO Critical Level (FCL)

The Communication Controller accepts modifications of the register in “DEFAULT_CONFIG” or “CONFIG” state only.

FCL
FIFO Critical Level **($030C_H$)** **Reset Value: $0000\ 0080_H$**



Field	Bits	Type	Description
CL	[7:0]	rw	**Critical Level** When the receive FIFO fill level FSR.RFFL is equal or greater than the critical level configured by CL, the receive FIFO critical level flag FSR.RFCL is set. If CL is programmed to values > 128, bit FSR.RFCL is never set. When FSR.RFCL changes from 0 to 1 bit **SIR.RFCL** is set to 1, and if enabled, a service request is generated.
0	[31:8]	r	**Reserved** Returns 0 if read; should be written with 0.

20.5.2.7 Message Buffer Status Registers

Message Handler Status (MHDS)

The Message Handler Status register gives the Host access to the current state of the Message Handler. A flag is cleared by writing a 1 to the corresponding bit position. Writing a 0 has no effect on the flag. An application reset will also clear the register. If one of the flags MHDS.PIBF, MHDS.POBF, MHDS.PMR, MHDS.PTBF1, MHDS.PTBF2changes from 0 to 1 EIR.EERR is set.

MHDS

Message Handler Status **(0310_H)** **Reset Value: $0000\ 0080_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	MBU							0	MBT						
r	rh							r	rh						

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	FMB							CRAM	MFMB	FMBD	PTBF2	PTBF1	PMR	POBF	PIBF
r	rh							rh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
PIBF	0	rwh	**Parity Error Input Buffer RAM 1,2** 0_B No parity error 1_B Parity error occurred when reading Input Buffer RAM 1 or Input Buffer RAM 2
POBF	1	rwh	**Parity Error Output Buffer RAM 1,2** 0_B No parity error 1_B Parity error occurred when reading Output Buffer RAM 1 or Output Buffer RAM 2
PMR	2	rwh	**Parity Error Message RAM** 0_B No parity error 1_B Parity error occurred when reading the Message RAM
PTBF1	3	rwh	**Parity Error Transient Buffer RAM A** 0_B No parity error 1_B Parity error occurred when reading Transient Buffer RAM A

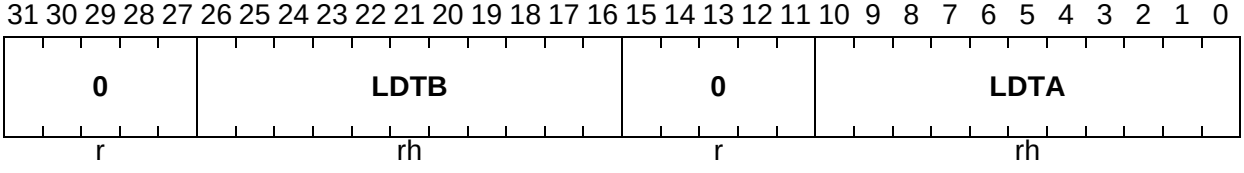
Field	Bits	Type	Description
PTBF2	4	rwh	**Parity Error Transient Buffer RAM B** 0_B No parity error 1_B Parity error occurred when reading Transient Buffer RAM B
FMBD	5	rwh	**Faulty Message Buffer Detected** 0_B No faulty Message Buffer 1_B Message Buffer referenced by MHDS.FMB holds faulty data due to a parity error
MFMB	6	rwh	**Multiple Faulty Message Buffers detected** 0_B No additional faulty Message Buffer 1_B Another faulty Message Buffer was detected while flag MHDS.FMBD is set
CRAM	7	rh	**Clear all internal RAM's** Signals that execution of the CHI command CLEAR_RAMS is ongoing (all bits of all internal RAM blocks are written to 0). The bit is set by application reset or by CHI command CLEAR_RAMS. 0_B No execution of the CHI command CLEAR_RAMS 1_B Execution of the CHI command CLEAR_RAMS ongoing
FMB	[14:8]	rh	**Faulty Message Buffer** Parity error occurred when reading from the Message Buffer or when transferring data from Input Buffer or Transient Buffer A or Transient Buffer B to the Message Buffer referenced by MHDS.FMB. Value only valid when one of the flags MHDS.PIBF, MHDS.PMR, MHDS.PTBF1, MHDS.PTBF2, and flag MHDS.FMBD is set. Updated only after the Host has reset flag MHDS.FMBD.
MBT	[22:16]	rh	**Message Buffer Transmitted** Number of last successfully transmitted Message Buffer. If the Message Buffer is configured for single-shot mode, the respective TXR flag in the Transmission Request Registers TXRQ1 to TXRQ4 was reset. MBT is reset when the Communication Controller leaves "CONFIG" state or enters "STARTUP" state.

Field	Bits	Type	Description
MBU	[30:24]	rh	**Message Buffer Updated** Number of Message Buffer that was updated last. For this Message Buffer the respective NDn (n = 0-31) to NDn (n = 96-127)and / or MBCn (n = 0-31) to MBCn (n = 96-127) flag in the New Data Registers NDAT1 to NDAT4 and the Message Buffer Status Changed MBSC1 to MBSC4 registers are also set. MBU is reset when the Communication Controller leaves “CONFIG” state or enters “STARTUP” state.
0	15, 23, 31	r	**Reserved** Returns 0 if read; should be written with 0.

Last Dynamic Transmit Slot (LDTS)

The Last Dynamic Transmit Slot Register stores the Slot Counter value at the time of the last Frame transmission in the dynamic segment. This register is reset when the Communication Controller leaves “CONFIG” state or enters “STARTUP” state.

LDTS
Last Dynamic Transmit Slot **(0314_H)** **Reset Value: $0000\ 0000_H$**

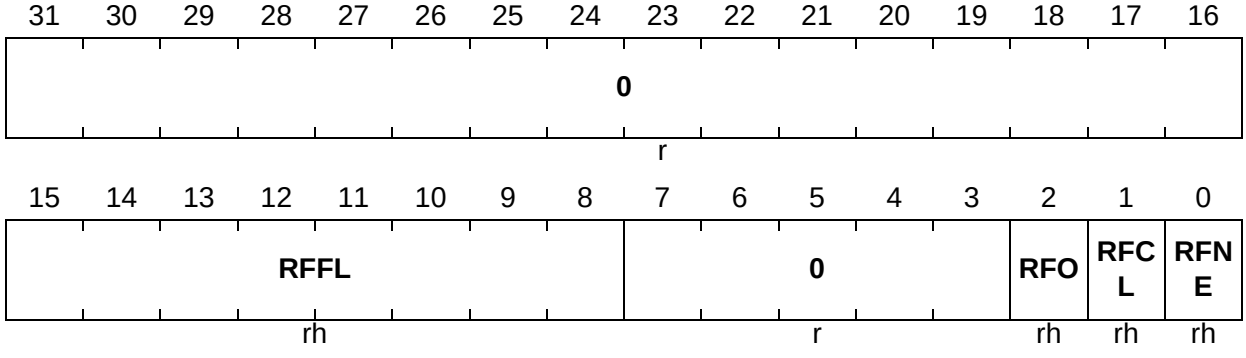


Field	Bits	Type	Description
LDTA	[10:0]	rh	**Last Dynamic Transmission Channel A** Value of (vSlotCounter[A]) at the time of the last Frame transmission on channel A in the dynamic segment of this node. It is updated at the end of the dynamic segment and is reset to zero if no Frame was transmitted during the dynamic segment.
LDTB	[26:16]	rh	**Last Dynamic Transmission Channel B** Value of (vSlotCounter[B]) at the time of the last Frame transmission on channel B in the dynamic segment of this node. It is updated at the end of the dynamic segment and is reset to zero if no Frame was transmitted during the dynamic segment.
0	[15:11], [31:27]	r	**Reserved** Returns 0 if read; should be written with 0.

FIFO Status Register (FSR)

The register is reset when the Communication Controller leaves “CONFIG” state or enters “STARTUP” state.

FSR
FIFO Status Register **(0318_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
RFNE	0	rh	**Receive FIFO Not Empty** This flag is set by the Communication Controller when a received valid Frame (data or NULL Frame depending on rejection mask) was stored in the FIFO. In addition, service request flag SIR.RFNE is set. The bit is reset after the Host has read all message from the FIFO. 0_B Receive FIFO is empty 1_B Receive FIFO is not empty
RFCL	1	rh	**Receive FIFO Critical Level** This flag is set when the receive FIFO fill level RFFL is equal or greater than the critical level as configured by FCL.CL. The flag is cleared by the Communication Controller as soon as RFFL drops below FCL.CL. When RFCL changes from 0 to 1 bit SIR.RFCL is set to 1, and if enabled, an service request is generated. 0_B Receive FIFO below critical level 1_B Receive FIFO critical level reached

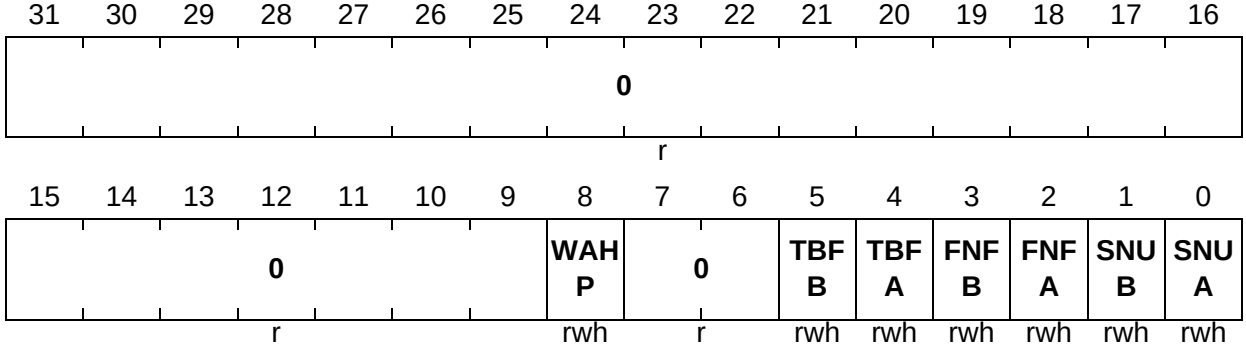
Field	Bits	Type	Description
RFO	2	rh	**Receive FIFO Overrun** The flag is set by the Communication Controller when a receive FIFO overrun is detected. When a receive FIFO overrun occurs, the oldest message is overwritten with the actual received message. In addition, service request flag EIR.RFO is set.The flag is cleared by the next FIFO read access issued by the Host. 0_B No receive FIFO overrun detected 1_B A receive FIFO overrun has been detected
RFFL	[15:8]	rh	**Receive FIFO Fill Level** Number of FIFO buffers filled up with new data not yet read by the Host. Maximum value is 128.
0	[7:3], [31:16]	r	**Reserved** Returns 0 if read; should be written with 0.

Message Handler Constraints Flags (MHDF)

Some constraints exist for the Message Handler regarding f_{CLC_ERAY} frequency, Message RAM configuration, and FlexRay™ bus traffic. To simplify software development, constraints violations are reported by setting flags in the MHDF. The register is reset when the Communication Controller leaves “CONFIG” state or enters “STARTUP” state. A flag is cleared by setting the corresponding bit position. Clearing has no effect on the flag. If any flag in MHDFL is set, interrupt flag EIR.MHF is set.

MHDF

Message Handler Constraints Flags ($031C_H$) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
SNUA	0	rwh	**Status Not Updated Channel A** This flag is set by the Communication Controller when the Message Handler, due to overload condition, was not able to update a Message Buffer’s status MBS with respect to channel A. 0_B No overload condition occurred when updating MBS for channel A 1_B MBS for channel A not updated
SNUB	1	rwh	**Status Not Updated Channel B** This flag is set by the Communication Controller when the Message Handler, due to overload condition, was not able to update a Message Buffer’s status MBS with respect to channel B. 0_B No overload condition occurred when updating MBS for channel B 1_B MBS for channel B not updated

Field	Bits	Type	Description
FNFA	2	rwh	**Find Sequence Not Finished Channel A** This flag is set by the Communication Controller when the Message Handler, due to overload condition, was not able to finish a find sequence (scan of Message RAM for matching Message Buffer) with respect to channel A. 0_B No find sequence not finished for channel A 1_B Find sequence not finished for channel A
FNFB	3	rwh	**Find Sequence Not Finished Channel B** This flag is set by the Communication Controller when the Message Handler, due to overload condition, was not able to finish a find sequence (scan of Message RAM for matching Message Buffer) with respect to channel B. 0_B No find sequence not finished for channel B 1_B Find sequence not finished for channel B
TBFA	4	rwh	**Transient Buffer Access Failure A** This flag is set by the Communication Controller when a read or write access to Transient Buffer A requested by PRT A could not complete within the available time. 0_B No TBF A access failure 1_B TBF A access failure
TBFB	5	rwh	**Transient Buffer Access Failure B** This flag is set by the Communication Controller when a read or write access to Transient Buffer B requested by PRT B could not complete within the available time. 0_B No Transient Buffer B access failure 1_B Transient Buffer B access failure
WAHP	8	rwh	**Write Attempt to Header Partition** Outside “DEFAULT_CONFIG” and “CONFIG” state this flag is set by the Communication Controller when the message handler tries to write message data into the Header Partition of the Message RAM due to faulty configuration of a Message Buffer. The write attempt is not executed, to protect the Header Partition from unintended write accesses. 0_B No write attempt to Header Partition 1_B Write attempt to Header Partition
0	[7:6], [31:9]	r	**Reserved** Returns 0 if read; should be written with 0.

Transmission Request 1 (TXRQ1)

This register reflect the state of the TXR flags of the configured Message Buffers 0 to 31. The flags are evaluated for transmit buffers only. If the number of configured Message Buffers is less than 31, the remaining TXRn flags have no meaning and are read as 0.

TXRQ1
Transmission Request Register 1 (0320_H) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TXR 31	**TXR 30**	**TXR 29**	**TXR 28**	**TXR 27**	**TXR 26**	**TXR 25**	**TXR 24**	**TXR 23**	**TXR 22**	**TXR 21**	**TXR 20**	**TXR 19**	**TXR 18**	**TXR 17**	**TXR 16**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TXR 15	**TXR 14**	**TXR 13**	**TXR 12**	**TXR 11**	**TXR 10**	**TXR 9**	**TXR 8**	**TXR 7**	**TXR 6**	**TXR 5**	**TXR 4**	**TXR 3**	**TXR 2**	**TXR 1**	**TXR 0**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
TXRn (n = 0-31)	n	rh	**Transmission Request n (n = 0-31)** If the flag is set, the respective Message Buffer 0 to 31 is ready for transmission respectively transmission of this Message Buffer is in progress. In single-shot mode the flags are reset after transmission has completed.

Transmission Request Register 2 (TXRQ2)

This register reflect the state of the TXR flags of the configured Message Buffers 31 to 63. The flags are evaluated for transmit buffers only. If the number of configured Message Buffers is less than 63, the remaining TXRn flags have no meaning and are read as 0.

TXRQ2
Transmission Request Register 2 (0324_H) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TXR 63	**TXR 62**	**TXR 61**	**TXR 60**	**TXR 59**	**TXR 58**	**TXR 57**	**TXR 56**	**TXR 55**	**TXR 54**	**TXR 53**	**TXR 52**	**TXR 51**	**TXR 50**	**TXR 49**	**TXR 48**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TXR 47	**TXR 46**	**TXR 45**	**TXR 44**	**TXR 43**	**TXR 42**	**TXR 41**	**TXR 40**	**TXR 39**	**TXR 38**	**TXR 37**	**TXR 36**	**TXR 35**	**TXR 34**	**TXR 33**	**TXR 32**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
TXRn (n = 32-63)	n - 32	rh	**Transmission Request n (n = 32-63)** If the flag is set, the respective Message Buffer 32 to 63 is ready for transmission respectively transmission of this Message Buffer is in progress. In single-shot mode the flags are reset after transmission has completed.

Transmission Request Register 3 (TXRQ3)

This register reflect the state of the TXR flags of the configured Message Buffers 64 to 95. The flags are evaluated for transmit buffers only. If the number of configured Message Buffers is less than 95, the remaining TXRn flags have no meaning and are read as 0.

TXRQ3
Transmission Request Register 3 (0328_H) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TXR 95	**TXR 94**	**TXR 93**	**TXR 92**	**TXR 91**	**TXR 90**	**TXR 89**	**TXR 88**	**TXR 87**	**TXR 86**	**TXR 85**	**TXR 84**	**TXR 83**	**TXR 82**	**TXR 81**	**TXR 80**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TXR 79	**TXR 78**	**TXR 77**	**TXR 76**	**TXR 75**	**TXR 74**	**TXR 73**	**TXR 72**	**TXR 71**	**TXR 70**	**TXR 69**	**TXR 68**	**TXR 67**	**TXR 66**	**TXR 65**	**TXR 64**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
TXRn (n = 64-95)	n - 64	rh	**Transmission Request n (n = 64-95)** If the flag is set, the respective Message Buffer 64 to 95 is ready for transmission respectively transmission of this Message Buffer is in progress. In single-shot mode the flags are reset after transmission has completed.

Transmission Request Register 4 (TXRQ4)

This register reflect the state of the TXR flags of the configured Message Buffers 96 to 127. The flags are evaluated for transmit buffers only. If the number of configured Message Buffers is less than 127, the remaining TXRn flags have no meaning and are read as 0.

TXRQ4
Transmission Request Register 4 ($032C_H$) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TXR 127	**TXR 126**	**TXR 125**	**TXR 124**	**TXR 123**	**TXR 122**	**TXR 121**	**TXR 120**	**TXR 119**	**TXR 118**	**TXR 117**	**TXR 116**	**TXR 115**	**TXR 114**	**TXR 113**	**TXR 112**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TXR 111	**TXR 110**	**TXR 109**	**TXR 108**	**TXR 107**	**TXR 106**	**TXR 105**	**TXR 104**	**TXR 103**	**TXR 102**	**TXR 101**	**TXR 100**	**TXR 99**	**TXR 98**	**TXR 97**	**TXR 96**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
TXRn (n = 96-127)	n - 96	rh	**Transmission Request n (n = 96-127)** If the flag is set, the respective Message Buffer 96 to 127 is ready for transmission respectively transmission of this Message Buffer is in progress. In single-shot mode the flags are reset after transmission has completed.

New Data Register 1 (NDAT1)

This register reflect the state of the ND flags of all configured Message Buffers 0 to 31. ND flags assigned to transmit buffers are meaningless. If the number of configured Message Buffers is less than 31, the remaining NDn flags have no meaning. The registers are reset when the Communication Controller leaves “CONFIG” state or enters “STARTUP” state.

NDAT1
New Data Register 1 **(0330_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ND 31	**ND 30**	**ND 29**	**ND 28**	**ND 27**	**ND 26**	**ND 25**	**ND 24**	**ND 23**	**ND 22**	**ND 21**	**ND 20**	**ND 19**	**ND 18**	**ND 17**	**ND 16**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ND 15	**ND 14**	**ND 13**	**ND 12**	**ND 11**	**ND 10**	**ND9**	**ND8**	**ND7**	**ND6**	**ND5**	**ND4**	**ND3**	**ND2**	**ND1**	**ND0**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
NDn (n = 0-31)	n	rh	**New Data n (n = 0-31)** The flags are set when a valid received Data Frame matches the Message Buffer’s filter configuration, independent of the payload length received or the payload length configured for that Message Buffer. The flags are not set after reception of NULL Frames except for Message Buffers belonging to the receive FIFO. An ND flag is reset when the Header Section of the corresponding Message Buffer is reconfigured or when the Data Section has been transferred to the Output Buffer.

New Data Register 2 (NDAT2)

This register reflect the state of the ND flags of all configured Message Buffers 32 to 63. ND flags assigned to transmit buffers are meaningless. If the number of configured Message Buffers is less than 63, the remaining NDn flags have no meaning. The registers are reset when the Communication Controller leaves “CONFIG” state or enters “STARTUP” state.

NDAT2
New Data Register 2 **(0334_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ND 63	**ND 62**	**ND 61**	**ND 60**	**ND 59**	**ND 58**	**ND 57**	**ND 56**	**ND 55**	**ND 54**	**ND 53**	**ND 52**	**ND 51**	**ND 50**	**ND 49**	**ND 48**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ND 47	**ND 46**	**ND 45**	**ND 44**	**ND 43**	**ND 42**	**ND 41**	**ND 40**	**ND 39**	**ND 38**	**ND 37**	**ND 36**	**ND 35**	**ND 34**	**ND 33**	**ND 32**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
NDn (n = 32-63)	n - 32	rh	**New Data n (n = 32-63)** The flags are set when a valid received Data Frame matches the Message Buffer’s filter configuration, independent of the payload length received or the payload length configured for that Message Buffer. The flags are not set after reception of NULL Frames except for Message Buffers belonging to the receive FIFO. An ND flag is reset when the Header Section of the corresponding Message Buffer is reconfigured or when the Data Section has been transferred to the Output Buffer.

New Data Register 3 (NDAT3)

This register reflect the state of the ND flags of all configured Message Buffers 64 to 95. ND flags assigned to transmit buffers are meaningless. If the number of configured Message Buffers is less than 95, the remaining NDn flags have no meaning. The registers are reset when the Communication Controller leaves “CONFIG” state or enters “STARTUP” state.

NDAT3
New Data Register 3 **(0338_H)** **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ND 95	**ND 94**	**ND 93**	**ND 92**	**ND 91**	**ND 90**	**ND 89**	**ND 88**	**ND 87**	**ND 86**	**ND 85**	**ND 84**	**ND 83**	**ND 82**	**ND 81**	**ND 80**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ND 79	**ND 78**	**ND 77**	**ND 76**	**ND 75**	**ND 74**	**ND 73**	**ND 72**	**ND 71**	**ND 70**	**ND 69**	**ND 68**	**ND 67**	**ND 66**	**ND 65**	**ND 64**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
NDn (n = 64-95)	n - 64	rh	**New Data n (n = 64-95)** The flags are set when a valid received Data Frame matches the Message Buffer’s filter configuration, independent of the payload length received or the payload length configured for that Message Buffer. The flags are not set after reception of NULL Frames except for Message Buffers belonging to the receive FIFO. An ND flag is reset when the Header Section of the corresponding Message Buffer is reconfigured or when the Data Section has been transferred to the Output Buffer.

New Data Register 4 (NDAT4)

This register reflect the state of the ND flags of all configured Message Buffers 96 to 127. ND flags assigned to transmit buffers are meaningless. If the number of configured Message Buffers is less than 127, the remaining NDn flags have no meaning. The registers are reset when the Communication Controller leaves “CONFIG” state or enters “STARTUP” state.

NDAT4
New Data Register 4 **($033C_H$)** **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ND 127	**ND 126**	**ND 125**	**ND 124**	**ND 123**	**ND 122**	**ND 121**	**ND 120**	**ND 119**	**ND 118**	**ND 117**	**ND 116**	**ND 115**	**ND 114**	**ND 113**	**ND 112**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ND 111	**ND 110**	**ND 109**	**ND 108**	**ND 107**	**ND 106**	**ND 105**	**ND 104**	**ND 103**	**ND 102**	**ND 101**	**ND 100**	**ND 99**	**ND 98**	**ND 97**	**ND 96**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
NDn (n = 96-127)	n - 96	rh	**New Data n (n = 96-127)** The flags are set when a valid received Data Frame matches the Message Buffer’s filter configuration, independent of the payload length received or the payload length configured for that Message Buffer. The flags are not set after reception of NULL Frames except for Message Buffers belonging to the receive FIFO. An ND flag is reset when the Header Section of the corresponding Message Buffer is reconfigured or when the Data Section has been transferred to the Output Buffer.

Message Buffer Status Changed 1 (MBSC1)

This register reflect the state of the MBC flags of all configured Message Buffers. If the number of configured Message Buffers is less than 31, the remaining MBCn flags have no meaning. The register is reset when the communication controller leaves “CONFIG” state or enters “STARTUP” state.

MBSC1
Message Buffer Status Changed 1 (0340_H) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MBC 31	**MBC 30**	**MBC 29**	**MBC 28**	**MBC 27**	**MBC 26**	**MBC 25**	**MBC 24**	**MBC 23**	**MBC 22**	**MBC 21**	**MBC 20**	**MBC 19**	**MBC 18**	**MBC 17**	**MBC 16**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MBC 15	**MBC 14**	**MBC 13**	**MBC 12**	**MBC 11**	**MBC 10**	**MBC 9**	**MBC 8**	**MBC 7**	**MBC 6**	**MBC 5**	**MBC 4**	**MBC 3**	**MBC 2**	**MBC 1**	**MBC 0**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
MBCn (n = 0-31)	n	rh	**Message Buffer Status Changed n (n = 0-31)** An MBC flags is set whenever the Message Handler changes on of the status flags VFRA, VFRB, SEOA, SEOB, CEOA, CEOB, SVOA, SVOB, TCIA, TCIB, ESA, ESB, MLST, FTA, FTB in the Header Section (see **“Message Buffer Status (MBS)” on Page 20-177**) of the respective Message Buffer 0 to Message Buffer 31. The flags are reset when the Header Section of the Message Buffer is reconfigured or when it has been transferred to the Output Buffer.

Message Buffer Status Changed 2 (MBSC2)

This register reflect the state of the MBC flags of all configured Message Buffers. If the number of configured Message Buffers is less than 63, the remaining MBCn flags have no meaning. The register is reset when the communication controller leaves “CONFIG” state or enters “STARTUP” state.

MBSC2
Message Buffer Status Changed 2 (0344_H) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MBC 63	**MBC 62**	**MBC 61**	**MBC 60**	**MBC 59**	**MBC 58**	**MBC 57**	**MBC 56**	**MBC 55**	**MBC 54**	**MBC 53**	**MBC 52**	**MBC 51**	**MBC 50**	**MBC 49**	**MBC 48**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MBC 47	**MBC 46**	**MBC 45**	**MBC 44**	**MBC 43**	**MBC 42**	**MBC 41**	**MBC 40**	**MBC 39**	**MBC 38**	**MBC 37**	**MBC 36**	**MBC 35**	**MBC 34**	**MBC 33**	**MBC 32**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
MBCn (n = 32-63)	n - 32	rh	**Message Buffer Status Changed n (n = 32-63)** An MBC flags is set whenever the Message Handler changes on of the status flags VFRA, VFRB, SEOA, SEOB, CEOA, CEOB, SVOA, SVOB, TCIA, TCIB, ESA, ESB, MLST, FTA, FTB in the Header Section (see **“Message Buffer Status (MBS)” on Page 20-177**) of the respective Message Buffer 32 to Message Buffer 63. The flags are reset when the Header Section of the Message Buffer is reconfigured or when it has been transferred to the Output Buffer.

Message Buffer Status Changed 3 (MBSC3)

This register reflect the state of the MBC flags of all configured Message Buffers. If the number of configured Message Buffers is less than 95, the remaining MBCn flags have no meaning. The register is reset when the communication controller leaves “CONFIG” state or enters “STARTUP” state.

MBSC3
Message Buffer Status Changed 3 (0348_H) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MBC 95	**MBC 94**	**MBC 93**	**MBC 92**	**MBC 91**	**MBC 90**	**MBC 89**	**MBC 88**	**MBC 87**	**MBC 86**	**MBC 85**	**MBC 84**	**MBC 83**	**MBC 82**	**MBC 81**	**MBC 80**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MBC 79	**MBC 78**	**MBC 77**	**MBC 76**	**MBC 75**	**MBC 74**	**MBC 73**	**MBC 72**	**MBC 71**	**MBC 70**	**MBC 69**	**MBC 68**	**MBC 67**	**MBC 66**	**MBC 65**	**MBC 64**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
MBCn (n = 64-95)	n - 64	rh	**Message Buffer Status Changed n (n = 64-95)** An MBC flags is set whenever the Message Handler changes on of the status flags VFRA, VFRB, SEOA, SEOB, CEOA, CEOB, SVOA, SVOB, TCIA, TCIB, ESA, ESB, MLST, FTA, FTB in the Header Section (see **“Message Buffer Status (MBS)” on Page 20-177**) of the respective Message Buffer 64 to Message Buffer 95. The flags are reset when the Header Section of the Message Buffer is reconfigured or when it has been transferred to the Output Buffer.

Message Buffer Status Changed 4 (MBSC4)

This register reflect the state of the MBC flags of all configured Message Buffers. If the number of configured Message Buffers is less than 127, the remaining MBCn flags have no meaning. The register is reset when the communication controller leaves “CONFIG” state or enters “STARTUP” state.

MBSC4
Message Buffer Status Changed 4 (034C$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MBC 127	**MBC 126**	**MBC 125**	**MBC 124**	**MBC 123**	**MBC 122**	**MBC 121**	**MBC 120**	**MBC 119**	**MBC 118**	**MBC 117**	**MBC 116**	**MBC 115**	**MBC 114**	**MBC 113**	**MBC 112**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MBC 111	**MBC 110**	**MBC 109**	**MBC 108**	**MBC 107**	**MBC 106**	**MBC 105**	**MBC 104**	**MBC 103**	**MBC 102**	**MBC 101**	**MBC 100**	**MBC 99**	**MBC 98**	**MBC 97**	**MBC 96**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

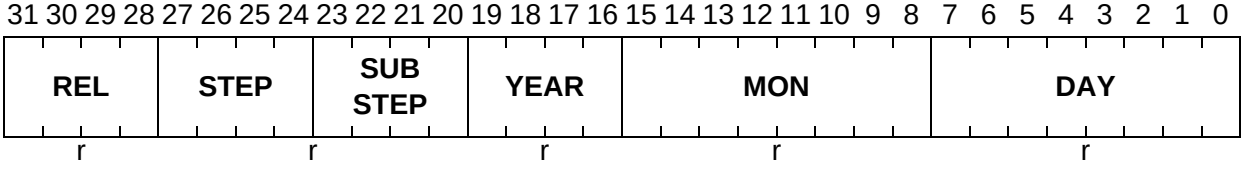
Field	Bits	Type	Description
MBCn (n = 96-127)	n - 96	rh	**Message Buffer Status Changed n (n = 96-127)** An MBC flags is set whenever the Message Handler changes on of the status flags VFRA, VFRB, SEOA, SEOB, CEOA, CEOB, SVOA, SVOB, TCIA, TCIB, ESA, ESB, MLST, FTA, FTB in the Header Section (see **“Message Buffer Status (MBS)” on Page 20-177**) of the respective Message Buffer 96 to Message Buffer 127. The flags are reset when the Header Section of the Message Buffer is reconfigured or when it has been transferred to the Output Buffer.

20.5.2.8 Identification Registers

Core Release Register (CREL)

This register contains bit fields about the ERAY module identification. It is read only.

CREL
Core Release Register **(03F0$_H$)** **Reset Value: XXXX XXXX$_H$**



Field	Bits	Type	Description
DAY	[7:0]	r	**Design Time Stamp, Day** Two digits, BCD-coded.
MON	[15:8]	r	**Design Time Stamp, Month** Two digits, BCD-coded.
YEAR	[19:16]	r	**Design Time Stamp, Year** One digit, BCD-coded.
SUBSTEP	[27:24]	r	**Sub-Step of Core Release** One digits, BCD-coded. 0_H Alpha, pre-Beta, pre-Beta-update, pre-Beta2, pre-Beta2-update, Beta, Beta2, Revision 1.0.0 1_H Beta_ct, Beta-ct-fix1, Revision 1.0.1 2_H Revision1.0RC1,Beta-ct-fix2, REVISION 1.0RC1
STEP	[27:24]	r	**Step of Core Release** One digits, BCD-coded. 0_H Revision 1.0.0 1_H Alpha 2_H pre-Beta 3_H pre-Beta-update 4_H pre-Beta2 5_H pre-Beta2-update 6_H Beta 7_H Beta2

Field	Bits	Type	Description
REL	[31:28]	r	**Core Release** One digit, BCD-coded. 0_B alpha…beta2ct 1_B Revision 1.0

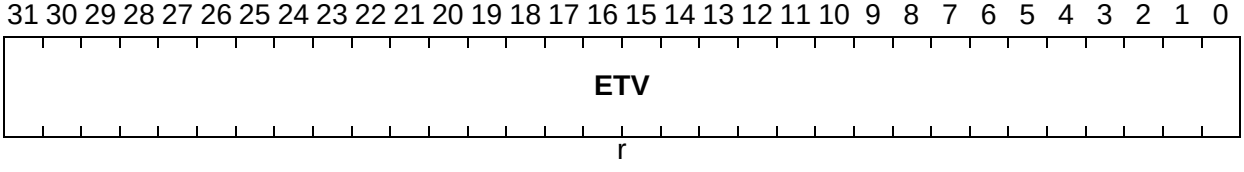
Table 20-7 Coding of releases

Release	Step	Sub-Step	Name	Release Date
0	1	0	Alpha	
0	2	0	pre-Beta	
0	3	0	pre-Beta-update	
0	4	0	pre-Beta2	
0	5	0	pre-Beta2-update	
0	6	0	Beta	
0	6	1	Beta-ct-fix1	14.10.2005
0	6	2	Beta-ct-fix2	14.12.2005
0	7	0	Beta2	03.02.2006
0	7	1	Beta2ct	24.03.2006
0	7	2	Revision 1.0RC1	07.04.2006
1	0	0	Release 1.0.0	19.05.2006
1	0	1	Release 1.0.1	2006
1	0	2	Release 1.0.2	31.10.2007

Endian Register (ENDN)

This register may be used to check, if the data of the E-Ray is handled by a host with the correct endian format. It is read only.

ENDN
Endian Register **(003F4$_H$)** **Reset Value: 8765 4321$_H$**



Field	Bits	Type	Description
ETV	[31:0]	r	**Endianness Test Value** The endianness test value.

20.5.2.9 Input Buffer

Double buffer structure consisting of Input Buffer Host and Input Buffer Shadow. While the Host can write to Input Buffer Host, the transfer to the Message RAM is done from Input Buffer Shadow. The Input Buffer holds the Header and Data Sections to be transferred to the selected Message Buffer in the Message RAM. It is used to configure the Message Buffers in the Message RAM and to update the Data Sections of transmit buffers.

When updating the Header Section of a Message Buffer in the Message RAM from the Input Buffer, the Message Buffer Status as described in **“Message Buffer Status (MBS)” on Page 20-177** is automatically reset to zero.

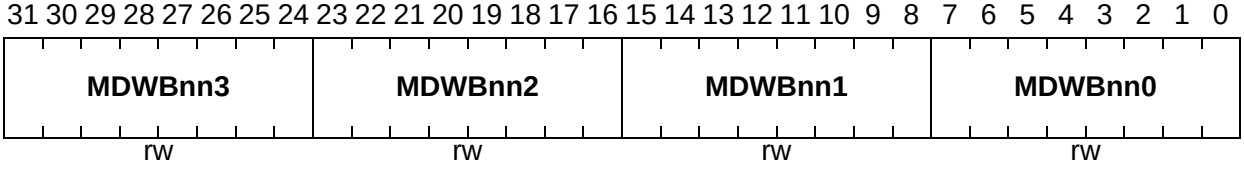
The Header Sections of Message Buffers belonging to the receive FIFO can only be (re)configured when the Communication Controller is in “DEFAULT_CONFIG” or “CONFIG” state. For those Message Buffers only the payload length configured and the data pointer need to be configured via WRHS2.PLC and WRHS2.DP. All information required for acceptance filtering is taken from the FIFO rejection filter and the FIFO rejection filter mask.

The data transfer between Input Buffer (IBF) and Message RAM is described in detail in **“Data Transfer from Input Buffer to Message RAM” on Page 20-224**.

Write Data Section [01 - 64] (WRDSnn (nn = 01-64))

The Write Data Section (WRDSnn, nn = 01-64) holds the data words to be transferred to the Data Section of the addressed Message Buffer. The data words (DW_n) are written to the Message RAM in transmission order from DW1 (byte0, byte1) to DW_{PL} (PL = number of data words as defined by the payload length configured by WRHS2.PLC).

WRDSnn (nn = 01-64)
Write Data Section nn **($03FC_H$ + nn * 4)** **Reset Value: $0000\ 0000_H$**



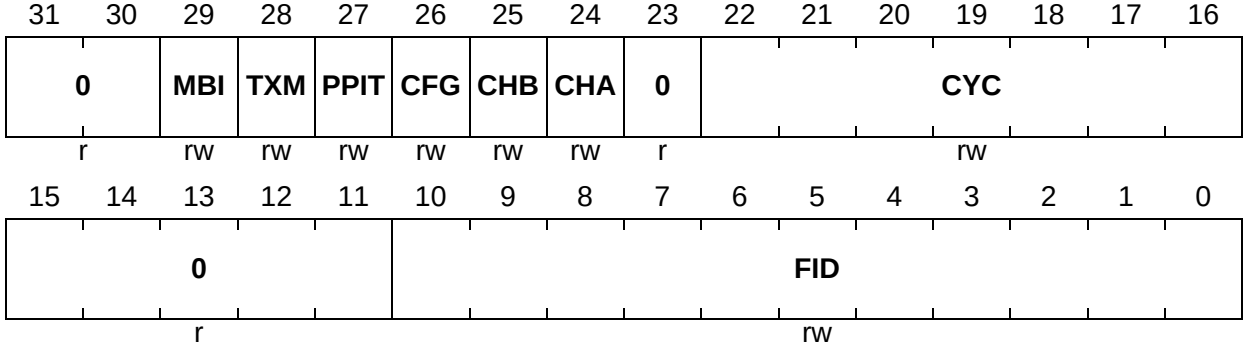
Field	Bits	Type	Description
MDWB0	[7:0]	rw	**32-Bit Word nn, Byte 0**
MDWB1	[15:8]	rw	**32-Bit Word nn, Byte 1**
MDWB2	[23:16]	rw	**32-Bit Word nn, Byte 2**
MDWB3	[31:24]	rw	**32-Bit Word nn, Byte 3**

Note: 16-bit Word 127 is located on WRDS64.MDW. In this case WRDS64.MDW is unused (no valid data). The Input Buffer RAMs are initialized to zero when leaving application reset or by CHI command CLEAR_RAMS.

Note: When writing to the WRDSnn (nn = 01-64), each 32-bit word has to be filled up by one 32-bit access OR two consecutive 16-bit accesses OR four consecutive 8-bit accesses before the transfer from the Input Buffer to the Message RAM is started by writing the number of the target Message Buffer in the Message RAM to the Input Buffer Command Request register. If a 32-bit word of the Input Buffer has been filled with less then two consecutive 16-bit accesses OR four consecutive 8-bit accesses (less then 32-bit), random data is transferred into the Input buffer for every not written 16-bit or 8-bit of a 32-bit word.

Write Header Section 1 (WRHS1)

WRHS1
Write Header Section 1 **(0500_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
FID	[10:0]	rw	**Frame ID** Frame ID of the selected Message Buffer. The Frame ID defines the slot number for transmission / reception of the respective message. Message Buffers with Frame ID = 0 are considered as not valid.
CYC	[22:16]	rw	**Cycle Code** The 7-bit cycle code determines the cycle set used for cycle counter filtering. For details about the configuration of the cycle code see **Section 20.6.7.3**.
CHA	24	rw	**Channel Filter Control A** The channel filtering field A associated with the buffer serves of channel A as a filter for receive buffers, and as a control field for transmit buffers
CHB	25	rw	**Channel Filter Control B** The channel filtering field B associated with the buffer serves of channel B as a filter for receive buffers, and as a control field for transmit buffers
CFG	26	rw	**Message Buffer Direction Configuration Bit** This bit is used to configure the corresponding buffer as a transmit buffer or as a receive buffer. For Message Buffers belonging to the receive FIFO the bit is not evaluated. 0_B The corresponding buffer is configured as Receive Buffer 1_B The corresponding buffer is configured as Transmit Buffer

Field	Bits	Type	Description
PPIT	27	rw	**Payload Preamble Indicator Transmit** This bit is used to control the state of the Payload Preamble Indicator in transmit Frames. If the bit is set in a static Message Buffer, the respective Message Buffer holds Network Management information. If the bit is set in a dynamic Message Buffer the first two byte of the Payload Segment may be used for message ID filtering by the receiver. Message ID filtering of received FlexRay™ Frames is not supported by the E-Ray module, but can be done by the Host. 0_B Payload Preamble Indicator not set 1_B Payload Preamble Indicator set
TXM	28	rw	**Transmission Mode** This bit is used to select the transmission mode (see **“Transmit Buffers” on Page 20-217**). 0_B Continuous mode 1_B Single-shot mode
MBI	29	rw	**Message Buffer Service Request** This bit enables the receive / transmit service request for the corresponding Message Buffer. After a dedicated receive buffer has been updated by the Message Handler, flag SIR.RXI and /or SIR.MBSI in the Status Service Request register are set. After a transmission has completed flag SIR.TXI is set. 0_B The corresponding Message Buffer service request is disabled 1_B The corresponding Message Buffer service request is enabled
0	[15:11], 23, [31:30]	r	**Reserved** Returns 0 if read; should be written with 0.

Note: The Input Buffer RAMs are initialized to zero when leaving application reset or by CHI command CLEAR_RAMS. Note that only the currently active IBF bank is cleared. To clear the 2nd bank as well, CUST1.IBF1PAG and CUST1.IBF2PAG need to be set and command CLEAR_RAMS need to be issued again. This is required in particular after an application reset. If the 2nd bank of IBF is left unused, this procedure is not required.

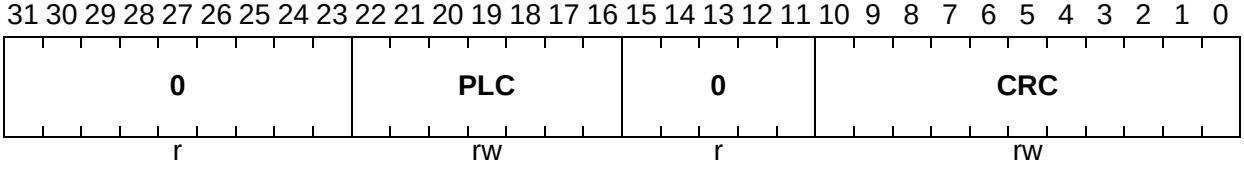
Table 20-8 Channel Filter Control Bits

CHA	CHB	Transmit Buffer transmit Frame on	Receive Buffer store Frame received from
1[1)]	1[1)]	Both Channels (static segment only)	Channel A or B (store first semantically valid Frame, static segment only)
1	0	Channel A	Channel A
0	1	Channel B	Channel B
0	0	No Transmission	Ignore Frame

1) If a Message Buffer is configured for the dynamic segment and both bits of the channel filtering field are set to 1, no Frames are transmitted resp. received Frames are ignored (same function as CHA = CHB = 0)

Write Header Section 2 (WRHS2)

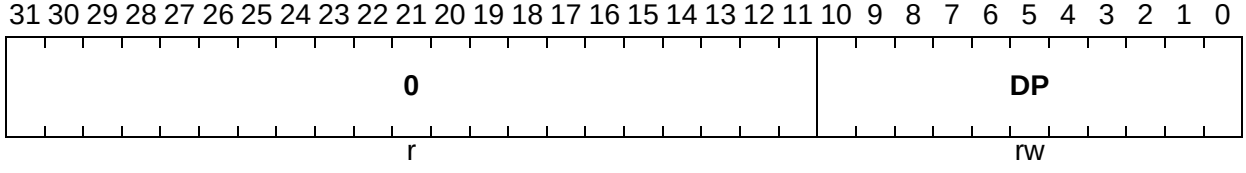
WRHS2
Write Header Section 2 **(0504_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
CRC	[10:0]	rw	**Header CRC** (vRF!Header!HeaderCRC) Receive Buffer: Configuration not required Transmit Buffer: Header CRC calculated and configured by the Host. For calculation of the Header CRC the payload length of the Frame send on the bus has to be considered. In static segment the payload length of all Frames is configured by MHDC.SFDL.
PLC	[22:16]	rw	**Payload Length Configured** Length of Data Section (number of 2-byte words) as configured by the Host. During static segment the static Frame payload length as configured by MHDC.SFDL in the MHD Configuration Register defines the payload length for all static Frames. If the payload length configured by PLC is shorter than this value padding byte are inserted to ensure that Frames have proper physical length. The padding pattern is logical zero.
0	[15:11], [31:23]	r	**Reserved** Returns 0 if read; should be written with 0.

Write Header Section 3 (WRHS3)

WRHS3
Write Header Section 3 **(0508_H)** **Reset Value: $0000\ 0000_H$**



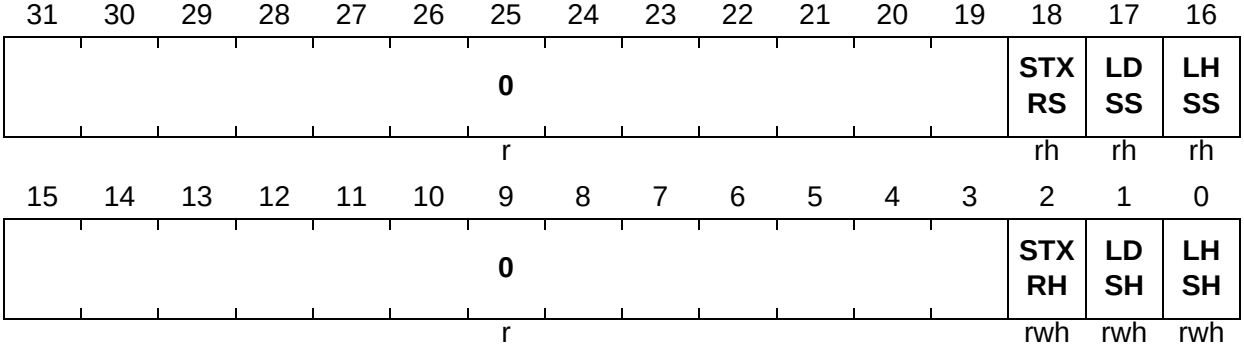
Field	Bits	Type	Description
DP	[10:0]	rw	**Data Pointer** Pointer to the first 32-bit word of the Data Section of the addressed Message Buffer in the Message RAM.
0	[31:11]	r	**Reserved** Returns 0 if read; should be written with 0.

Input Buffer Command Mask (IBCM)

Configures how the Message Buffer in the Message RAM selected by the Input Buffer Command Request register IBCR is updated. If IBF Host and IBF Shadow are swapped, also masked bits IBCM.LHSH, IBCM.LDSH, and IBCM.STXRH are swapped with bits IBCM.LHSS, IBCM.LDSS, and IBCM.STXRS to keep them attached to the respective Input Buffer transfer.

IBCM

Input Buffer Command Mask (0510_H) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
LHSH	0	rwh	**Load Header Section Host** 0_B Header Section is not updated 1_B Header Section selected for transfer from Input Buffer to the Message RAM
LDSH	1	rwh	**Load Data Section Host** 0_B Data Section is not updated 1_B Data Section selected for transfer from Input Buffer to the Message RAM
STXRH	2	rwh	**Set Transmission Request Host** If this bit is set to 1, the Transmission Request flag TXRQ1.TXRn (n = 0-31) to TXRQ4.TXRn (n = 0-31) for the selected Message Buffer is set in the Transmission Request Registers to release the Message Buffer for transmission. In single-shot mode the flag is cleared by the Communication Controller after transmission has completed. TXRQ1.TXRn (n = 0-31) to TXRQ4.TXRn (n = 0-31) are evaluated for transmit buffer only. 0_B Reset Transmission Request flag 1_B Set Transmission Request flag, transmit buffer released for transmission

Field	Bits	Type	Description
LHSS	16	rh	**Load Header Section Shadow** 0_B Header Section is not updated 1_B Header Section selected for transfer from Input Buffer to the Message RAM (transfer is ongoing of finalized)
LDSS	17	rh	**Load Data Section Shadow** 0_B Data Section is not updated 1_B Data Section selected for transfer from Input Buffer to the Message RAM (transfer is ongoing of finalized)
STXRS	18	rh	**Transmission Request Shadow** If this bit is set to 1, the Transmission Request flag TXRQ1.TXRn (n = 0-31) to TXRQ4.TXRn (n = 0-31) for the selected Message Buffer is set in the Transmission Request Registers to release the Message Buffer for transmission. In single-shot mode the flag is cleared by the Communication Controller after transmission has completed. TXRQ1.TXRn (n = 0-31) to TXRQ4.TXRn (n = 0-31) are evaluated for transmit buffer only. 0_B Reset Transmission Request flag 1_B Set Transmission Request flag, transmit buffer released for transmission (operation is ongoing of finalized)
0	[15:3], [31:19]	r	**Reserved** Returns 0 if read; should be written with 0.

Input Buffer Command Request (IBCR)

When the Host writes the number of the target Message Buffer in the Message RAM to IBRH in the Input Buffer Command Request register, IBF Host and IBF Shadow are swapped. In addition the Message Buffer numbers stored under IBRH and IBRS are also swapped (see also **“Data Transfer from Input Buffer to Message RAM” on Page 20-224**).

With this write operation the IBSYS bit in the Input Buffer Command Request register is set to 1. The Message Handler then starts to transfer the contents of IBF Shadow to the Message Buffer in the Message RAM selected by IBRS.

While the Message Handler transfers the data from IBF Shadow to the target Message Buffer in the Message RAM, the Host may write the next message into the IBF Host. After the transfer between IBF Shadow and the Message RAM has completed, the IBSYS bit is set back to 0 and the next transfer to the Message RAM may be started by the Host by writing the respective target Message Buffer number to IBRH.

If a write access to IBRH occurs while IBSYS is 1, IBSYH is set to 1. After completion of the ongoing data transfer from IBF Shadow to the Message RAM, IBF Host and IBF Shadow are swapped, IBSYH is reset to 0. IBSYS remains set to 1, and the next transfer to the Message RAM is started. In addition the Message Buffer numbers stored under IBRH and IBRS are also swapped. Any write access to an Input Buffer register while both IBSYS and IBSYH are set will cause the error flag EIR.IIBA to be set. In this case the Input Buffer will not be changed.

IBCR
Input Buffer Command Request (0514_H) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IBSYS	0								IBRS						
rh	r								rh						

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IBSYH	0								IBRH						
rh	r								rwh						

Field	Bits	Type	Description
IBRH	[6:0]	rwh	**Input Buffer Request Host** Selects the target Message Buffer in the Message RAM for data transfer from Input Buffer. Valid values are 00_H to $7F_H$ (0…127).

Field	Bits	Type	Description
IBSYH	15	rh	**Input Buffer Busy Host** Set to 1 by writing IBRH while IBSYS is still 1. After the ongoing transfer between IBF Shadow and the Message RAM has completed, the IBSYH is set back to 0. 0_B No request pending 1_B Request while transfer between IBF Shadow and Message RAM in progress
IBRS	[22:16]	rh	**Input Buffer Request Shadow** Number of the target Message Buffer actually updated/lately updated. Valid values are 00_H to $7F_H$ (0…127).
IBSYS	31	rh	**Input Buffer Busy Shadow** Set to 1 after writing IBRH. When the transfer between IBF Shadow and the Message RAM has completed, IBSYS is set back to 0. 0_B Transfer between IBF Shadow and Message RAM completed 1_B Transfer between IBF Shadow and Message RAM in progress
0	[14:7], [30:23]	r	**Reserved** Returns 0 if read; should be written with 0.

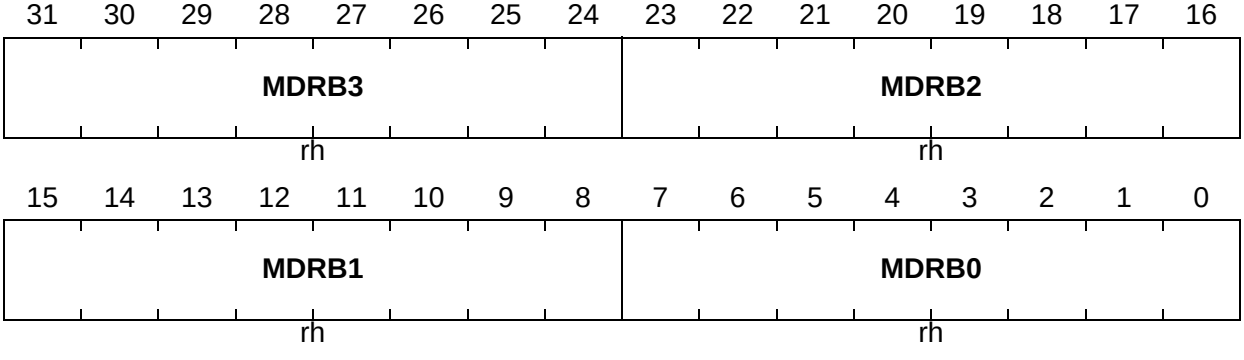
20.5.2.10 Output Buffer

Double buffer structure consisting of Output Buffer Host and Output Buffer Shadow. Used to read out Message Buffers from the Message RAM. While the Host can read from Output Buffer Host, the Message Handler transfers the selected Message Buffer from Message RAM to the respective Output Buffer Shadow. The data transfer between Message RAM and Output Buffer (OBF) is described in **"Data Transfer from Message RAM to Output Buffer" on Page 20-226**.

Read Data Section [1…64] (RDDSn)

The Read Data Section nn (RDDSnn, nn = 01-64) holds the data words read from the Data Section of the addressed Message Buffer. The data words are read from the Message RAM in reception order from DW1 (byte0, byte1) to DW_{PL} (PL = number of data words as defined by the Payload Length).

RDDSnn (nn = 01-64)
Read Data Section nn **($05FC_H$ + nn * 4)** **Reset Value: $0000\ 0000_H$**



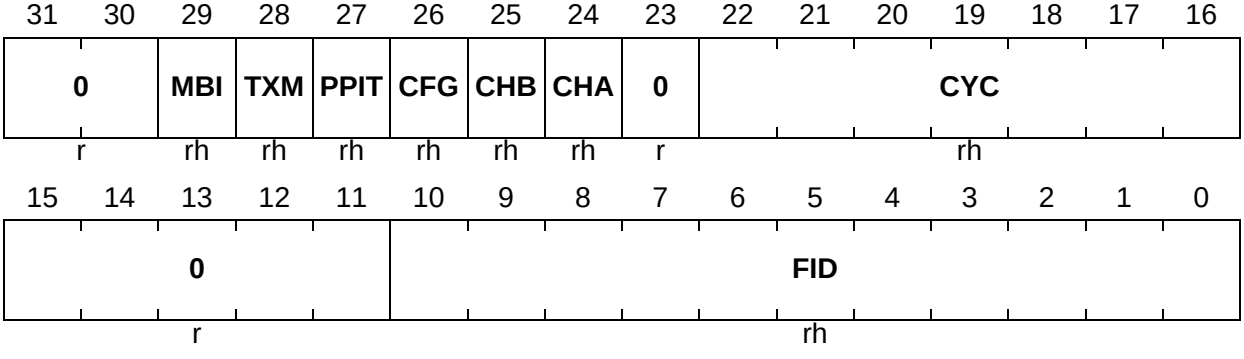
Field	Bits	Type	Description
MDRB0	[7:0]	rh	**32-Bit Word nn, Byte 0**
MDRB1	[15:8]	rh	**32-Bit Word nn, Byte 1**
MDRB2	[23:16]	rh	**32-Bit Word nn, Byte 2**
MDRB3	[31:24]	rh	**32-Bit Word nn, Byte 3**

Note: DW127 is located on RDDS64.MDW. In this case RDDS64.MDW is unused (no valid data). The Output Buffer RAMs are initialized to zero when leaving application reset or by CHI command CLEAR_RAMS.

Read Header Section 1 (RDHS1)

Values as configured by the Host via WRHS1 Register:

RDHS1
Read Header Section 1 **(0700_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
FID	[10:0]	rh	**Frame ID**
CYC	[22:16]	rh	**Cycle Code**
CHA	24	rh	**Channel Filter Control A**
CHB	25	rh	**Channel Filter Control B**
CFG	26	rh	**Message Buffer Direction Configuration Bit**
PPIT	27	rh	**Payload Preamble Indicator Transmit**
TXM	28	rh	**Transmission Mode**
MBI	29	rh	**Message Buffer Service Request**
0	[15:11], 23, [31:30]	r	**Reserved** Returns 0 if read; should be written with 0.

Note: In case that the Message Buffer read from the Message RAM belongs to the receive FIFO, FID holds the received Frame ID, while CYC, CHA, CHB, CFG, PPIT, TXM, and MBI are reset to zero.

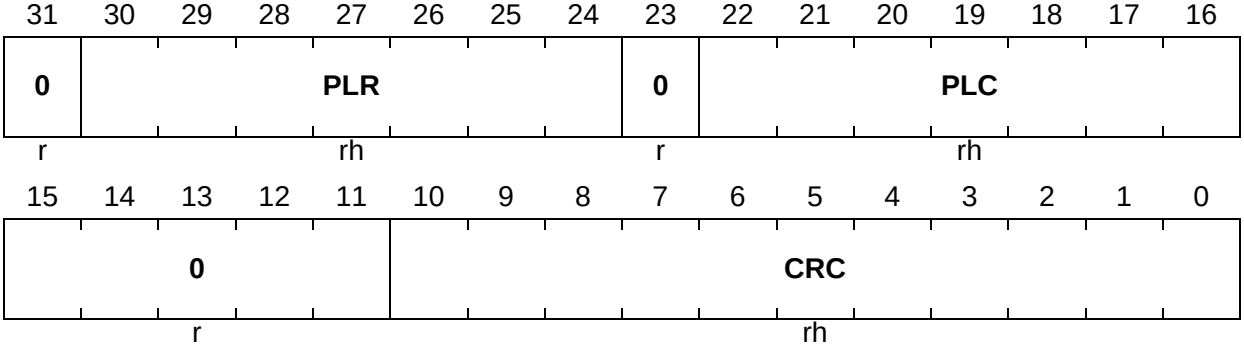
Table 20-9 Channel Filter Control Bits

CHA	CHB	Transmit Buffer transmit Frame on	Receive Buffer store Frame received from
1[1)]	1[1)]	Both Channels (static segment only)	Channel A or B (store first semantically valid Frame, static segment only)
1	0	Channel A	Channel A
0	1	Channel B	Channel B
0	0	No Transmission	Ignore Frame

1) If a Message Buffer is configured for the dynamic segment and both bits of the channel filtering field are set to 1, no Frames are transmitted resp. received Frames are ignored (same function as CHA = CHB = 0)

Read Header Section 2 (RDHS2)

RDHS2
Read Header Section 2 **(0704_H)** **Reset Value: $0000\ 0000_H$**



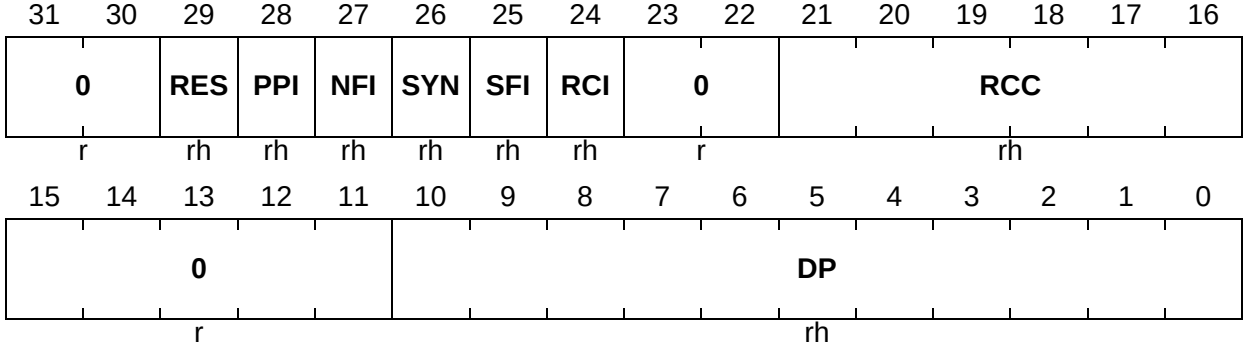
Field	Bits	Type	Description
CRC	[10:0]	rh	**Header CRC** (vRF!Header!HeaderCRC) Receive Buffer: Configuration not required. Header CRC updated from receive Data Frames. Transmit Buffer: Header CRC calculated and configured by the Host
PLC	[22:16]	rh	**Payload Length Configured** Length of Data Section (number of 2-byte words) as configured by the Host.

Field	Bits	Type	Description
PLR	[30:24]	rh	**Payload Length Received** (vRF!Header!Length) Payload length value updated from received Data Frame (exception: if Message Buffer belongs to the receive FIFO PLR is also updated from received NULL Frames). When a message is stored into a Message Buffer the following behavior with respect to payload length received and payload length configured is implemented: • **PLR > PLC:** The payload data stored in the Message Buffer is truncated to the payload length configured for even PLC or else truncated to PLC + 1. • **PLR ≤ PLC:** The received payload data is stored into the Message Buffers Data Section. The remaining data bytes of the Data Section as configured by PLC are filled with undefined data. • **PLR = 0:** The Message Buffer's Data Section is filled with undefined data. • **PLC = 0:** Message Buffer has no Data Section configured. No data is stored into the Message Buffer's Data Section.
0	[15:11], 23, 31	r	**Reserved** Returns 0 if read; should be written with 0.

Note: The Message RAM is organized in 4-byte words. When received data is stored into a Message Buffer's Data Section, the number of 2-byte data words written into the Message Buffer is PLC rounded to the next even value. PLC should be configured identical for all Message Buffers belonging to the receive FIFO. Header 2 is updated from Data Frames only.

Read Header Section 3 (RDHS3)

RDHS3
Read Header Section 3 **(0708_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
DP	[10:0]	rh	**Data Pointer** Pointer to the first 32-bit word of the Data Section of the addressed Message Buffer in the Message RAM.
RCC	[21:16]	rh	**Receive Cycle Count** (vRF!Header!CycleCount) Cycle counter value updated from received Data Frame.
RCI	24	rh	**Received on Channel Indicator** (vSS!Channel) Indicates the channel from which the received Data Frame was taken to update the respective receive buffer. 0_B Frame received on channel B 1_B Frame received on channel A
SFI	25	rh	**Startup Frame Indicator** (vRF!Header!SuFIndicator) A Startup Frame is marked by the Startup Frame indicator. 0_B The received Frame is not a startup Frame 1_B The received Frame is a startup Frame
SYN	26	rh	**SYNC Frame Indicator** (vRF!Header!SyFIndicator) A SYNC Frame is marked by the SYNC Frame indicator. 0_B The received Frame is not a SYNC Frame 1_B The received Frame is a SYNC Frame
NFI	27	rh	**NULL Frame Indicator** (vRF!Header!NFIndicator) Is set to 1 after storage of the first received Data Frame. 0_B Up to now no Data Frame has been stored into the respective Message Buffer 1_B At least one Data Frame has been stored into the respective Message Buffer

Field	Bits	Type	Description
PPI	28	rh	**Payload Preamble Indicator** (vRF!Header!PPIndicator) The payload preamble indicator defines whether a Network Management vector or message ID is contained within the Payload Segment of the received Frame. 0_B The Payload Segment of the received Frame does not contain a Network Management vector nor a message ID 1_B Static segment: Network Management vector in the first part of the payload Dynamic segment: Message ID in the first part of the payload
RES	29	rh	**Reserved Bit** (vRF!Header!Reserved) Reflects the state of the received reserved bit. The reserved bit is transmitted as 0.
0	[15:11], [23:22], [31:30]	r	**Reserved** Returns 0 if read; should be written with 0.

Note: Header 3 is updated from Data Frames only.

Message Buffer Status (MBS)

The Message Buffer status is updated by the Communication Controller with respect to the assigned channel(s) latest at the end of the slot following the slot assigned to the Message Buffer. The flags are updated only when the Communication Controller is in “NORMAL_ACTIVE” or “NORMAL_PASSIVE” state. If only one channel (A or B) is assigned to a Message Buffer, the channel-specific status flags of the other channel are written to zero. If both channels are assigned to a Message Buffer, the channel-specific status flags of both channels are updated. The Message Buffer status is updated only when the slot counter reached the configured Frame ID and when the cycle counter filter matched. When the Host updates a Message Buffer via Input Buffer, all MBS flags are reset to zero independent of which IBCM bits are set or not. For details about receive / transmit filtering see **“Filtering and Masking” on Page 20-213**, **“Transmit Process” on Page 20-217**, and **“Receive Process” on Page 20-220**.

Whenever the Message Handler changes one of the flags VFRA, VFRB, SEOA, SEOB, CEOA, CEOB, SVOA, SVOB, TCIA, TCIB, ESA, ESB, MLST, FTA, FTB the respective Message Buffer’s MBC flag in registers MBSC1 to MBSC4 is set

MBS

Message Buffer Status **($070C_H$)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		RESS	PPIS	NFIS	SYNS	SFIS	RCIS	0		CCS					
r		rh	rh	rh	rh	rh	rh	r		rh					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FTB	FTA	0	MLST	ESB	ESA	TCIB	TCIA	SVOB	SVOA	CEOB	CEOA	SEOB	SEOA	VRFB	VRFA
rh	rh	r	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
VFRA	0	rh	**Valid Frame Received on Channel A** (vSS!ValidFrameA) A valid Frame indication is set if a valid Frame was received on channel A. 0_B No valid Frame received on channel A 1_B Valid Frame received on channel A
VFRB	1	rh	**Valid Frame Received on Channel B** (vSS!ValidFrameB) A valid Frame indication is set if a valid Frame was received on channel B. 0_B No valid Frame received on channel B 1_B Valid Frame received on channel B

Field	Bits	Type	Description
SEOA	2	rh	**Syntax Error Observed on Channel A** (vSS!SyntaxErrorA) A syntax error was observed in the assigned slot on channel A. 0_B No syntax error observed on channel A 1_B Syntax error observed on channel A
SEOB	3	rh	**Syntax Error Observed on Channel B** (vSS!SyntaxErrorB) A syntax error was observed in the assigned slot on channel B. 0_B No syntax error observed on channel B 1_B Syntax error observed on channel B
CEOA	4	rh	**Content Error Observed on Channel A** (vSS!ContentErrorA) A content error was observed in the assigned slot on channel A. 0_B No content error observed on channel A 1_B Content error observed on channel A
CEOB	5	rh	**Content Error Observed on Channel B** (vSS!ContentErrorB) A content error was observed in the assigned slot on channel B. 0_B No content error observed on channel B 1_B Content error observed on channel B
SVOA	6	rh	**Slot Boundary Violation Observed on Channel A** (vSS!BViolationA) A slot boundary violation (channel active at the start or at the end of the assigned slot) was observed on channel A. 0_B No slot boundary violation observed on channel A 1_B Slot boundary violation observed on channel A
SVOB	7	rh	**Slot Boundary Violation Observed on Channel B** (vSS!BViolationB) A slot boundary violation (channel active at the start or at the end of the assigned slot) was observed on channel B. 0_B No slot boundary violation observed on channel B 1_B Slot boundary violation observed on channel B
TCIA	8	rh	**Transmission Conflict Indication Channel A** (vSS!TxConflictA) A transmission conflict indication is set if a transmission conflict has occurred on channel A. 0_B No transmission conflict occurred on channel A 1_B Transmission conflict occurred on channel A

Field	Bits	Type	Description
TCIB	9	rh	**Transmission Conflict Indication Channel B** (vSS!TxConflictB) A transmission conflict indication is set if a transmission conflict has occurred on channel B. 0_B No transmission conflict occurred on channel B 1_B Transmission conflict occurred on channel B
ESA	10	rh	**Empty Slot Channel A** In an empty slot there is no activity detected on the bus. The condition is checked in static and dynamic slots. 0_B Bus activity detected in the assigned slot on channel A 1_B No bus activity detected in the assigned slot on channel A
ESB	11	rh	**Empty Slot Channel B** In an empty slot there is no activity detected on the bus. The condition is checked in static and dynamic slots. 0_B Bus activity detected in the assigned slot on channel B 1_B No bus activity detected in the assigned slot on channel B
MLST	12	rh	**Message Lost** The flag is set in case the Host did not read the message before the Message Buffer was updated from a received Data Frame. Not affected by reception of NULL Frames except for Message Buffers belonging to the receive FIFO. The flag is reset by a Host write to the Message Buffer via IBF or when a new message is stored into the Message Buffer after the Message Buffers ND flag was reset by reading out the Message Buffer via OBF. 0_B No message lost 1_B Unprocessed message was overwritten
FTA	14	rh	**Frame Transmitted on Channel A** Indicates that this node has transmitted a Data Frame in the assigned slot on channel A. 0_B No transmission transmitted on channel A 1_B Data Frame transmitted on channel A in cycle defined by CCS bit field *Note: The FlexRay™ protocol specification requires that FTA can only be reset by the Host. Therefore the Cycle Count Status CCS for these bits is only valid for the cycle where the bits are set to 1*

Field	Bits	Type	Description
FTB	15	rh	**Frame Transmitted on Channel B** Indicates that this node has transmitted a Data Frame in the assigned slot on channel B. 0_B No transmission transmitted on channel B 1_B Data Frame transmitted on channel B in cycle defined by CCS bit field *Note: The FlexRay™ protocol specification requires that FTB can only be reset by the Host. Therefore the Cycle Count Status CCS for these bits is only valid for the cycle where the bits are set to 1*
CCS	[21:16]	rh	**Cycle Count Status** Cycle Count when status (MBS register) has been updated.
RCIS	24	rh	**Received on Channel Indicator Status** (vSS!Channel) Indicates the channel on which the Frame was received. 0_B Frame received on channel B 1_B Frame received on channel A *Note: For receive buffers (CFG = 0) the RCIS is updated from both valid data and NULL Frames. If no valid Frame was received, the previous value is maintained. For transmit buffers the flags have no meaning and should be ignored.*
SFIS	25	rh	**Startup Frame Indicator Status** (vRF!Header!SuFIndicator) A Startup Frame is marked by the Startup Frame indicator. 0_B No Startup Frame received 1_B The received Frame is a startup Frame *Note: For receive buffers (CFG = 0) the SFIS is updated from both valid data and NULL Frames. If no valid Frame was received, the previous value is maintained. For transmit buffers the flags have no meaning and should be ignored.*
SYNS	26	rh	**SYNC Frame Indicator Status** (vRF!Header!SyFIndicator) A Startup Frame is marked by the Startup Frame indicator. 0_B No SYNC Frame received 1_B The received Frame is a SYNC Frame *Note: For receive buffers (CFG = 0) the SYNS is updated from both valid data and NULL Frames. If no valid Frame was received, the previous value is maintained. For transmit buffers the flags have no meaning and should be ignored.*

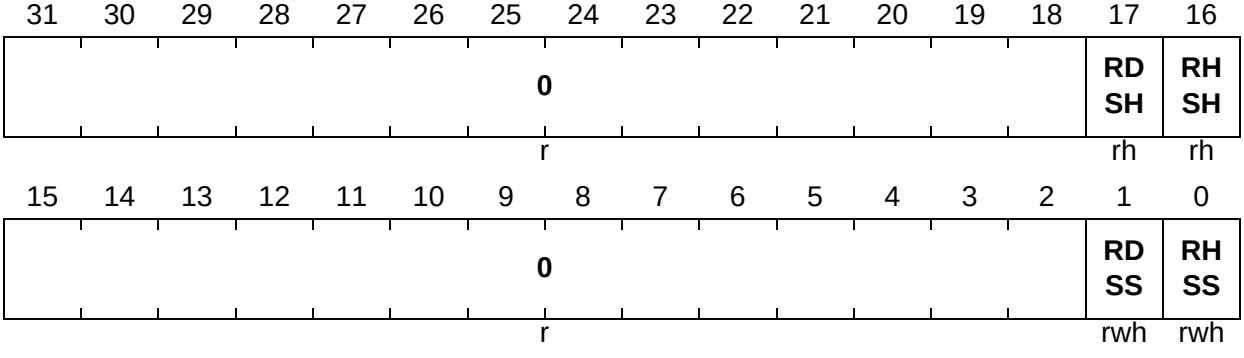
Field	Bits	Type	Description
NFIS	27	rh	**NULL Frame Indicator Status** (vRF!Header!NFIndicator) If reset to 0 the Payload Segment of the received Frame contains no usable data. 0_B Received Frame is a NULL Frame 1_B Received Frame is not a NULL Frame *Note: For receive buffers (CFG = 0) the NFIS is updated from both valid data and NULL Frames. If no valid Frame was received, the previous value is maintained. For transmit buffers the flags have no meaning and should be ignored.*
PPIS	28	rh	**Payload Preamble Indictor Status** (vRF!Header!PPIndicator) The payload preamble indicator defines whether a Network Management vector or message ID is contained within the Payload Segment of the received Frame. Static Segment: 0_B The Payload Segment of the received Frame does not contain a Network Management vector or a message ID 1_B Network Management vector at the beginning of the payload Dynamic Segment: 0_B The Payload Segment of the received Frame does not contain a Network Management vector or a message ID 1_B Message ID at the beginning of the payload *Note: For receive buffers (CFG = 0) the PPIS is updated from both valid data and NULL Frames. If no valid Frame was received, the previous value is maintained. For transmit buffers the flags have no meaning and should be ignored.*
RESS	29	rh	**Reserved Bit Status** (vRF!Header!Reserved) Reflects the state of the received reserved bit. The reserved bit is transmitted as 0. *Note: For receive buffers (CFG = 0) the RESS is updated from both valid data and NULL Frames. If no valid Frame was received, the previous value is maintained. For transmit buffers the flags have no meaning and should be ignored.*
0	13, [23:22], [31:30]	r	**Reserved** Returns 0 if read; should be written with 0.

Output Buffer Command Mask (OBCM)

Configures how the Output Buffer is updated from the Message Buffer in the Message RAM selected by the Output Buffer Command Request register. If OBF Host and OBF Shadow are swapped, also mask bits OBCM.RDSH and OBCM.RHSH are swapped with bits OBCM.RDSS and OBCM.RHSS to keep them attached to the respective Output Buffer transfer.

OBCM

Output Buffer Command Mask (0710_H) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
RHSS	0	rwh	**Read Header Section Shadow** 0_B Header Section is not read 1_B Header Section selected for transfer from Message RAM to Output Buffer
RDSS	1	rwh	**Read Data Section Shadow** 0_B Data Section is not read 1_B Data Section selected for transfer from Message RAM to Output Buffer
RHSH	16	rh	**Read Header Section Host** 0_B Header Section is not read 1_B Header Section selected for transfer from Message RAM to Output Buffer
RDSH	17	rh	**Read Data Section Host** 0_B Data Section is not read 1_B Data Section selected for transfer from Message RAM to Output Buffer
0	[15:2], [31:18]	r	**Reserved** Returns 0 if read; should be written with 0.

Note: After the transfer of the Header Section from the Message RAM to OBF Shadow has completed, the Message Buffer status changed flag MBCn (n = 0-31) to MBCn (n = 96-127) of the selected Message Buffer in the Message Buffer Changed MBSC1 to MBSC4 registers is cleared. After the transfer of the Data Section from the Message RAM to OBF Shadow has completed, the New Data flag NDn (n = 0-31) to NDn (n = 96-127) of the selected Message Buffer in the New Data NDAT1 to NDAT4 registers is cleared.

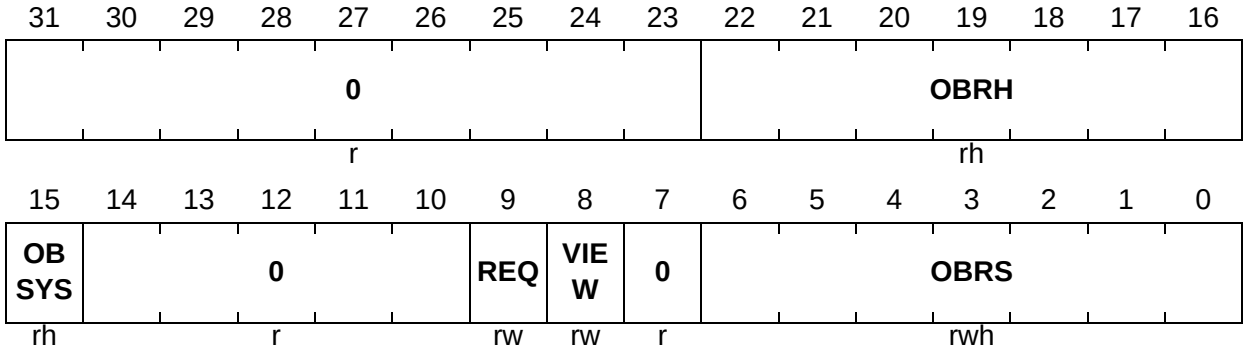
Output Buffer Command Request (OBCR)

The Message Buffer selected by OBCR.OBRS is transferred from the Message RAM to the Output Buffer as soon as the Host has set OBCR.REQ. Bit OBCR.REQ can only be set while OBCR.OBSYS is 0 (see also **“Data Transfer from Message RAM to Output Buffer” on Page 20-226**).

After setting OBCR.REQ, OBCR.OBSYS is automatically set, and the transfer of the Message Buffer selected by OBCR.OBRS from the Message RAM to Output Buffer Shadow is started. When the transfer between the Message RAM and OBF Shadow has completed, this is signalled by clearing OBCR.OBSYS. By setting OBCR.VIEW while OBCR.OBSYS is 0, OBF Host and OBF Shadow are swapped. When Output Buffer Host and Output Buffer Shadow are swapped, also mask bits OBCM.RDSH and OBCM.RHSH are swapped with bits OBCM.RDSS and OBCM.RHSS to keep them attached to the respective Output Buffer transfer. Now the Host can read the transferred Message Buffer from OBF Host. In parallel the Message Handler may transfer the next message from the Message RAM to OBF Shadow if OBCR.VIEW and OBCR.REQ are set at the same time.

Any write access to an Output Buffer register while OBCR.OBSYS is set will cause the error flag EIR.IOBA to be set. In this case the Output Buffer will not be changed.

OBCR
Output Buffer Command Request (0714$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
OBRS	[6:0]	rw	**Output Buffer Request Shadow** Number of source Message Buffer to be transferred from the Message RAM to OBF Shadow. Valid values are 00_H to $7F_H$ (0 to 127). If the number of the first Message Buffer of the receive FIFO is written to this register the Message Handler transfers the Message Buffer addressed by the GET Index Register (GIDX, **"FIFO Function" on Page 20-221**) to OBF Shadow.
VIEW	8	rw	**View Shadow Buffer** Toggles between OBF Shadow and OBF Host. Only writeable while OBCR.OBSYS = 0. 0_B No action 1_B Swap OBF Shadow and OBF Host
REQ	9	rw	**Request Message RAM Transfer** Requests transfer of Message Buffer addressed by OBCR.OBRS from Message RAM to OBF Shadow. Only writeable while OBCR.OBSYS = 0. 0_B No request 1_B Transfer to OBF Shadow requested
OBSYS	15	rh	**Output Buffer Busy Shadow** Set to 1 after setting bit OBCR.REQ. When the transfer between the Message RAM and OBF Shadow has completed, OBCR.OBSYS is cleared again. 0_B No transfer in progress 1_B Transfer between Message RAM and OBF Shadow in progress

Field	Bits	Type	Description
OBRH	[22:16]	rh	**Output Buffer Request Host** Number of Message Buffer currently accessible by the Host via RDHS1 to RDHS3, MBS, and RDDSnn (nn = 01-64). By setting OBCR.VIEW OBF Shadow and OBF Host are swapped and the transferred Message Buffer is accessible by the Host. Valid values are 00_H to $7F_H$ (01 to 27).
0	7, [14:10], [31:23]	r	**Reserved** Returns 0 if read; should be written with 0.

20.6 Functional Description

This chapter describes the E-Ray implementation together with the related FlexRay™ protocol features. More information about the FlexRay™ protocol itself can be found in the FlexRay™ protocol specification v2.1.

Communication on FlexRay™ networks is based on Frames and symbols. The wakeup symbol (WUS) and the collision avoidance symbol (CAS) are transmitted outside the communication cycle to setup the time schedule. Frames and media access test symbols (MTS) are transmitted inside the communication cycle.

20.6.1 Communication Cycle

A communication cycle in FlexRay™ consists of the following elements:

- Static Segment
- Dynamic Segment
- Symbol Window
- Network Idle Time (NIT)

Static segment, dynamic segment, and symbol window form the Network Communication Time (NCT). For each communication channel the slot counter starts at 1 and counts up until the end of the dynamic segment is reached. Both channels share the same arbitration grid which means that they use the same synchronized Macrotick.

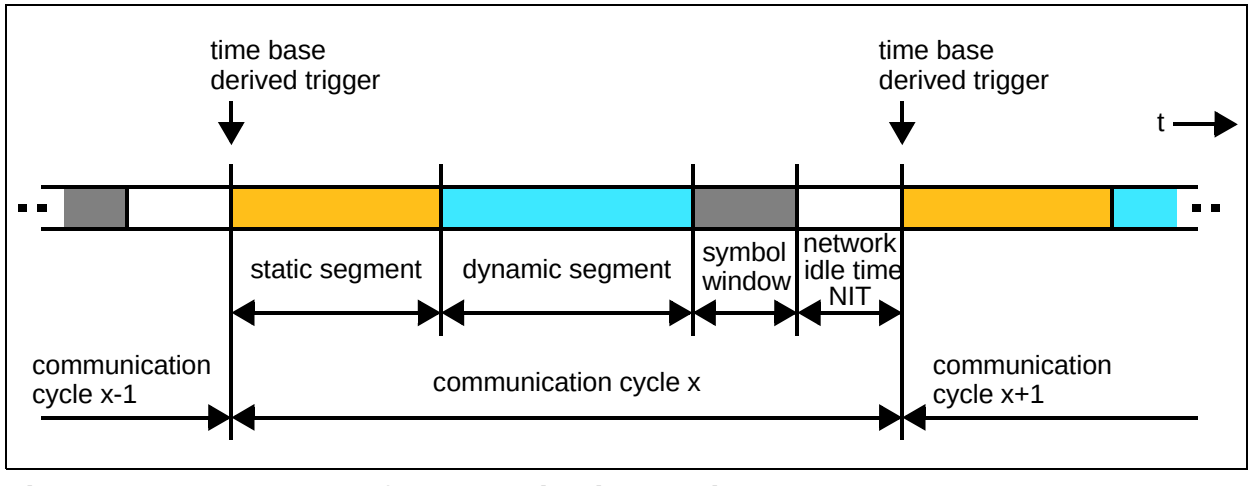


Figure 20-4 Structure of Communication Cycle

20.6.1.1 Static Segment

The Static Segment is characterized by the following features:

- Time slots of fixed length (optionally protected by bus guardian)
- Start of Frame transmission at action point of the respective static slot
- Payload length same for all Frames on both channel

Parameters: Number of Static Slots **GTUC07.NSS**, Static Slot Length **GTUC07.SSL**, Payload Length Static **MHDC.SFDL**, Action Point Offset **GTUC09.APO**.

20.6.1.2 Dynamic Segment

The Dynamic Segment is characterized by the following features:

- All controllers have bus access (no bus guardian protection possible)
- Variable payload length and duration of slots, different for both channels
- Start of transmission at minislot action point

Parameters: Number of Minislots **GTUC08.NMS**, Minislot Length **GTUC08.MSL** Minislot Action Point Offset **GTUC09.MAPO**, Start of Latest Transmit (last minislot) **MHDC.SLT**.

20.6.1.3 Symbol Window

During the symbol window only one media access test symbol (MTS) may be transmitted per channel. MTS symbols are send in “NORMAL_ACTIVE” state to test the bus guardian.

The symbol window is characterized by the following features:

- Send single symbol
- Transmission of the MTS symbol starts at the symbol windows action point

Parameters: Symbol Window Action Point Offset **GTUC09.APO** (same as for static slots), Network Idle Time Start **GTUC04.NIT**.

20.6.1.4 Network Idle Time (NIT)

During network idle time the Communication Controller has to perform the following tasks:

- Calculate clock correction terms (offset and rate)
- Distribute offset correction over multiple Macroticks
- Perform cluster cycle related tasks

Parameters: Network Idle Time Start **GTUC04.NIT**, Offset Correction Start **GTUC04.OCS**.

20.6.1.5 Configuration of Network Idle Time (NIT) Start and Offset Correction Start

The number of Macroticks per cycle (gMacroPerCycle) is assumed to be m. It is configured by programming **GTUC02.MPC** = m.

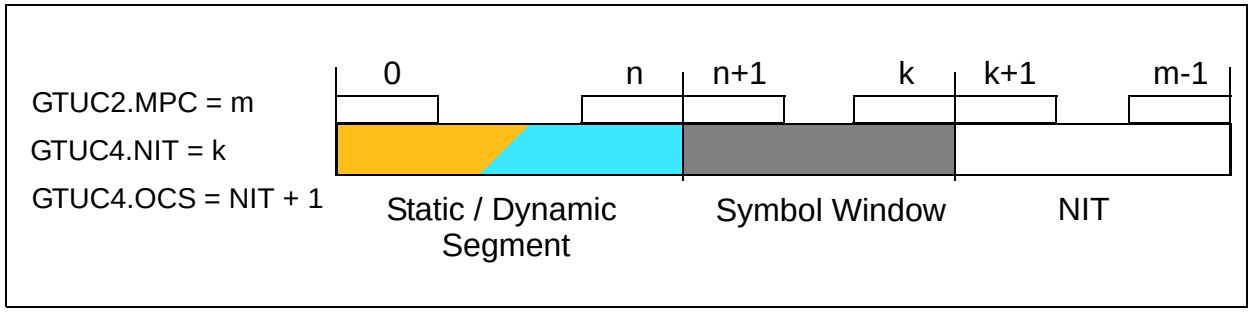


Figure 20-5 Configuration of network idle time (NIT) start and offset correction start

The static / dynamic segment starts with Macrotick 0 and ends with Macrotick n:

n = static segment length + dynamic segment offset + dynamic segment length - 1 Macrotick

n = gNumberOfStaticSlots • gdStaticSlot + dynamic segment offset + gNumberOfMinislots • gdMinislot - 1 Macroticks

The static segment length is configured by **GTUC07.SSL** and **GTUC07.NSS**.
The dynamic segment length is configured by **GTUC08.MSL** and **GTUC08.NMS**.

The dynamic segment offset is:

If gdActionPointOffset ≤ gdMinislotActionPointOffset:

dynamic segment offset = 0 MT

Else if gdActionPointOffset > gdMinislotActionPointOffset:

dynamic segment offset = gdActionPointOffset - gdMinislotActionPointOffset

The network idle time (NIT) starts with Macrotick k+1 and ends with the last Macrotick of cycle m-1. It has to be configured by setting **GTUC04.NIT** = k.

For the E-Ray the offset correction start is required to be

GTUC04.OCS ≥ **GTUC04.NIT** + 1 = k+1.

The length of symbol window results from the number of Macroticks between the end of the static / dynamic segment and the beginning of the NIT. It can be calculated by k - n.

20.6.2 Communication Modes

The FlexRay™ Protocol Specification v2.1 defines the Time-Triggered Distributed (TT-D) mode.

Time-triggered Distributed (TT-D)

In TT-D mode the following configurations are possible:

- **Pure static**: minimum 2 static slots + symbol window (optional)
- **Mixed static/dynamic**: minimum 2 static slots + dynamic segment + symbol window (optional)

A minimum of two coldstart nodes need to be configured for distributed time-triggered operation. Two fault-free coldstart nodes are necessary for the cluster startup. Each Startup Frame must be a SYNC Frame, therefore all coldstart nodes are sync nodes.

20.6.3 Clock Synchronization

In TT-D mode a distributed clock synchronization is used. Each node individually synchronizes itself to the cluster by observing the timing of received SYNC Frames from other nodes.

20.6.3.1 Global Time

Activities in a FlexRay™ node, including communication, are based on the concept of a global time, even though each individual node maintains its own view of it. It is the clock synchronization mechanism that differentiates the FlexRay™ cluster from other node collections with independent clock mechanisms. The global time is a vector of two values; the cycle (cycle counter) and the cycle time (Macrotick counter).

Cluster specific:

- Macrotick = basic unit of time measurement in a FlexRay™ network, a Macrotick consists of an integer number of Microticks
- Cycle length = duration of a communication cycle in units of Macroticks

20.6.3.2 Local Time

Internally, nodes time their behavior with Microtick resolution. Microticks are time units derived from the oscillator clock tick of the specific node. Therefore Microticks are controller-specific units. They may have different duration in different controllers. The precision of a node’s local time difference measurements is a Microtick.

Node specific:

- Oscillator clock → prescaler → Microtick
- Microtick = basic unit of time measurement in a Communication Controller, clock correction is done in units of Microticks
- Cycle counter + Macrotick counter = nodes local view of the global time

20.6.3.3 Synchronization Process

Clock synchronization is performed by means of SYNC Frames. Only preconfigured nodes (sync nodes) are allowed to send SYNC Frames. In a two-channel cluster a sync node has to send its SYNC Frame on both channels.

For synchronization in FlexRay™ the following constraints have to be considered:

- Max. one SYNC Frame per node in one communication cycle
- Max. 15 SYNC Frames per cluster in one communication cycle
- Every node has to use all available SYNC Frames for clock synchronization
- Minimum of two sync nodes required for clock synchronization and startup

For clock synchronization the time difference between expected and observed arrival time of SYNC Frames received during the static segment, valid on both channels (two-channel cluster), is measured. The calculation of correction terms is done during network idle time (NIT) (offset: every cycle, rate: odd cycle) by using a FTA / FTM algorithm. For details see FlexRay™ protocol specification v2.1, chapter 8.

Offset (phase) Correction

- Only deviation values measured and stored in the current cycle used
- For a two channel node the smaller value will be taken
- Calculation during network idle time (NIT) of **every** communication cycle, value may be negative
- Offset correction value calculated in even cycles used for error checking only
- Checked against limit values (violation: “NORMAL_ACTIVE” → “NORMAL_PASSIVE” → “HALT”)
- Correction value is an integer number of Microticks
- Correction done in **odd** numbered cycles, distributed over the Macroticks beginning at offset correction start up to cycle end (end of network idle time (NIT)) to shift nodes next start of cycle (Macroticks lengthened / shortened)

Rate (frequency) Correction

- Pairs of deviation values measured and stored in even / odd cycle pair used
- For a two channel node the average of the differences from the two channels is used
- Calculated during network idle time (NIT) of **odd** numbered cycles, value may be negative
- Cluster drift damping is performed using global damping value

- Checked against limit values
- Correction value is a signed integer number of Microticks
- Distributed over Macroticks comprising the next **even / odd** cycle pair (Macroticks lengthened / shortened)

Synchronization Process

Clock synchronization is performed by means of SYNC Frames. Only preconfigured nodes (sync nodes) are allowed to send SYNC Frames. In a two-channel cluster a sync node has to send its SYNC Frame on both channels.

For synchronization in FlexRay™ the following constraints have to be considered:

- Max. one SYNC Frame per node in one communication cycle
- Max. 15 SYNC Frames per cluster in one communication cycle
- Every node has to use all available SYNC Frames for clock synchronization
- Minimum of two sync nodes required for clock synchronization and startup

For clock synchronization the time difference between expected and observed arrival time of SYNC Frames received during the static segment, valid on both channels (two-channel cluster), is measured. The calculation of correction terms is done during network idle time (NIT) (offset: every cycle, rate: odd cycle) by using a FTA / FTM algorithm. For details see FlexRay™ protocol specification v2.1, chapter 8.

SYNC Frame Transmission

SYNC Frame transmission is only possible from buffer 0 and 1. Message Buffer 1 may be used for SYNC Frame transmission in case that SYNC Frames should have different payloads on the two channels. In this case bit **MRC.SPLM** has to be programmed to 1.

Message Buffers used for SYNC Frame transmission have to be configured with the key slot ID and can be (re)configured in “DEFAULT_CONFIG” or “CONFIG” state only. For nodes transmitting SYNC Frames **SUCC1.TXSY** must be set to 1.

20.6.3.4 External Clock Synchronization

During normal operation, independent clusters can drift significantly. If synchronous operation across independent clusters is desired, external synchronization is necessary; even though the nodes within each cluster are synchronized. This can be accomplished with synchronous application of host-deduced rate and offset correction terms to the clusters.

- External offset / rate correction value is a signed integer
- External offset / rate correction value is added to calculated offset / rate correction value
- Aggregated offset / rate correction term (external + internal) is not checked against configured limits

20.6.4 Error Handling

The implemented error handling concept is intended to ensure that in case of a lower layer protocol error in a single node communication between non-affected nodes can be maintained. In some cases, higher layer program command activity is required for the Communication Controller to resume normal operation. A change of the error handling state will set bit **EIR.PEMC** in the Error Service Request Register and may trigger an service request to the Host if enabled. The actual error mode is signalled by **CCEV.ERRM** in the Communication Controller Error Vector register.

Table 20-10 Error Modes of the POC (Degradation Model)

Error Mode	Activity
ACTIVE (green)	**Full operation**, State: “NORMAL_ACTIVE” The Communication Controller is fully synchronized and supports the cluster wide clock synchronization. The host is informed of any error condition(s) or status change by interrupt (if enabled) or by reading the error and status interrupt flags from registers **EIR** and **SIR**.
PASSIVE (yellow)	**Reduced operation**, State: “NORMAL_PASSIVE”, Communication Controller self rescue allowed The Communication Controller stops transmitting Frames and symbols, but received Frames are still processed. Clock synchronization mechanisms are continued based on received Frames. No active contribution to the cluster wide clock synchronization. The host is informed of any error condition(s) or status change by interrupt (if enabled) or by reading the error and status interrupt flags from registers **EIR** and **SIR**.
COMM_HALT (red)	**Operation halted**, State: “HALT”, Communication Controller self rescue not allowed The Communication Controller stops Frame and symbol processing, clock synchronization processing, and the Macrotick generation. The host has still access to error and status information by reading the error and status interrupt flags from registers **EIR** and **SIR**. The bus drivers are disabled.

20.6.4.1 Clock Correction Failed Counter

When the Clock Correction Failed Counter reaches the maximum “without clock correction passive” limit defined by **SUCC3.WCP**, the POC transits from “NORMAL_ACTIVE” to “NORMAL_PASSIVE” state. When it reaches the “maximum without clock correction fatal” limit defined by **SUCC3.WCF**, it transits “NORMAL_ACTIVE” or “NORMAL_PASSIVE” to the “HALT” state. Both limits are defined in the SUC Configuration Register 3.

The Clock Correction Failed Counter **CCEV.CCFC** allows the Host to monitor the duration of the inability of a node to compute clock correction terms after the Communication Controller passed protocol startup phase. It will be incremented by one at the end of any **odd** numbered communication cycle where either the Missing Offset Correction signal **SFS.MOCS** nor the Missing Rate Correction signal **SFS.MRCS** flag is set. The two flags are located in the SYNC Frame Status register, while the Clock Correction Failed Counter is located in the Communication Controller Error Vector register.

The Clock Correction Failed Counter is reset to zero at the end of an **odd** communication cycle if neither the Missing Offset Correction signal **SFS.MOCS** nor the Missing Rate Correction signal **SFS.MRCS** flag is set.

The Clock Correction Failed Counter stops incrementing when the “maximum without clock correction fatal” value **SUCC3.WCF** as defined in the SUC Configuration Register 3 is reached (i.e. incrementing the counter at its maximum value will not cause it to “wraparound” back to zero). The Clock Correction Failed Counter is initialized to zero when the Communication Controller enters “READY” state or when “NORMAL_ACTIVE” state is entered.

20.6.4.2 Passive to Active Counter

The passive to active counter controls the transition of the POC from “NORMAL_PASSIVE” to “NORMAL_ACTIVE” state. **SUCC1.PTA** in the SUC Configuration Register 1 defines the number of consecutive even / odd cycle pairs that must have valid clock correction terms before the Communication Controller is allowed to transit from “NORMAL_PASSIVE” to “NORMAL_ACTIVE” state. If **SUCC1.PTA** is reset to zero the Communication Controller is not allowed to transit from “NORMAL_PASSIVE” to “NORMAL_ACTIVE” state.

20.6.4.3 HALT Command

In case the Host wants to stop FlexRay™ communication of the local node it can bring the Communication Controller into “HALT” state by asserting the HALT command. This can be done by writing **SUCC1.CMD** = 0110_B in the SUC Configuration Register 1. When called in “NORMAL_ACTIVE” or “NORMAL_PASSIVE” state the POC transits to “HALT” state at the end of the current cycle. When called in any other state **SUCC1.CMD** will be reset to 0000_B = “COMMAND_NOT_ACCEPTED” and bit **EIR.CNA** in the Error Service Request Register is set to 1. If enabled an service request to the Host is generated.

20.6.4.4 FREEZE Command

In case the Host detects a severe error condition it can bring the Communication Controller into “HALT” state by asserting the FREEZE command. This can be done by writing **SUCC1.CMD** = 0111_B in the SUC Configuration Register 1. The FREEZE

command triggers the entry of the “HALT” state immediately regardless of the actual POC state.

The POC state from which the transition to HALT state took place can be read from **CCSV**.**PSL**.

20.6.5 Communication Controller States

This chapter introduces the states of the Communication Controller.

20.6.5.1 Communication Controller State Diagram

State transitions are controlled by externals the application reset or RXDA/B, by the POC state machine, and by the CHI Command Vector **SUCC1.CMD** located in the SUC Configuration Register 1.

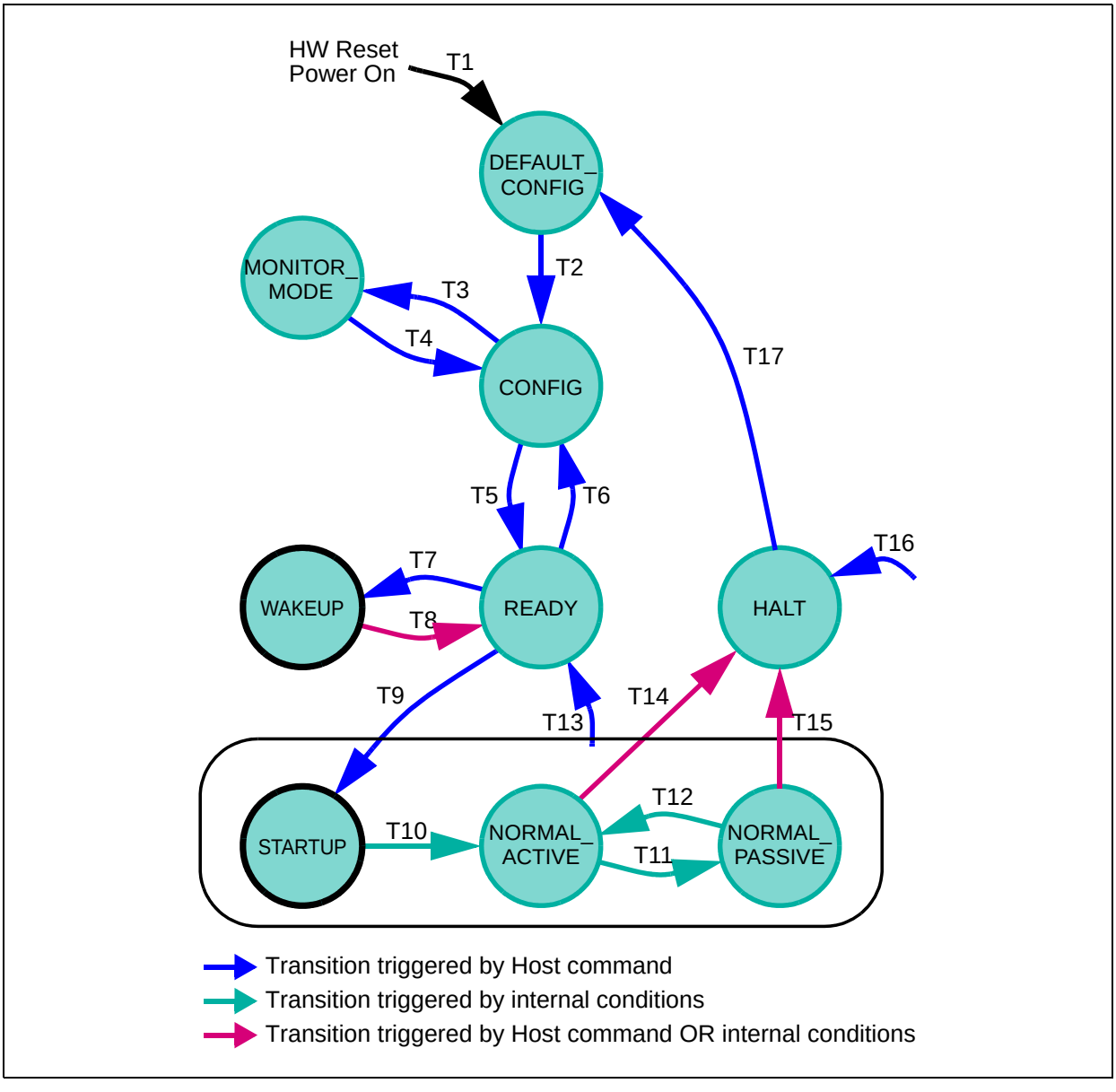


Figure 20-6 Overall State Diagram of E-Ray Communication Controller

The Communication Controller exits from all states to “HALT” state after application of the FREEZE command (SUCC1.CMD = 0111_B).

Table 20-11 State Transitions of E-Ray Overall State Machine

T#	Condition	From	To
1	application reset	HW Reset	DEFAULT_CONFIG
2	Command CONFIG, SUCC1.CMD = 0001_B	DEFAULT_CONFIG	CONFIG
3	Unlock sequence followed by command MONITOR_MODE, SUCC1.CMD = 1011_B	CONFIG	MONITOR_MODE
4	Command CONFIG, SUCC1.CMD = 0001_B	MONITOR_MODE	CONFIG
5	Unlock sequence followed by command READY, SUCC1.CMD = 0010_B	CONFIG	READY
6	Command CONFIG, SUCC1.CMD = 0001_B	READY	CONFIG
7	Command WAKEUP, SUCC1.CMD = 0011_B	READY	WAKEUP
8	Complete, non-aborted transmission of wakeup pattern OR received WUP OR received Frame Header OR command READY, SUCC1.CMD = 0010_B	WAKEUP	READY
9	Command RUN, SUCC1.CMD = 0100_B	READY	STARTUP
10	Successful startup	STARTUP	NORMAL_ACTIVE
11	Clock Correction Failed counter reached Maximum Without Clock Correction Passive limit configured by WCP in SUC Configuration Register 3	NORMAL_ACTIVE	NORMAL_PASSIVE
12	Number of valid correction terms reached the Passive to Active limit configured by PTA in SUC Configuration Register 1	NORMAL_PASSIVE	NORMAL_ACTIVE
13	Command READY, SUCC1.CMD = 0010_B	STARTUP, NORMAL_ACTIVE, NORMAL_PASSIVE	READY

Table 20-11 State Transitions of E-Ray Overall State Machine (cont'd)

T#	Condition	From	To
14	Clock Correction Failed counter reached Maximum Without Clock Correction Fatal limit configured by WCF in SUC Configuration Register 3 AND bit HCSE in the SUC Configuration Register 1 set to 1 OR command HALT, SUCC1.CMD = 0110_B	NORMAL_ACTIVE	HALT
15	Clock Correction Failed counter reached Maximum Without Clock Correction Fatal limit configured by WCF in SUC Configuration Register 3 AND bit HCSE in the SUC Configuration Register 1 set to 1 OR command HALT, SUCC1.CMD = 0110_B	NORMAL_PASSIVE	HALT
16	Command FREEZE, SUCC1.CMD = 0111_B	All States	HALT
17	Command CONFIG, SUCC1.CMD = 0001_B	HALT	DEFAULT_CONFIG

20.6.5.2 DEFAULT_CONFIG State

In "DEFAULT_CONFIG" state, the Communication Controller is stopped. All configuration registers are accessible and the pins to the physical layer are in their inactive state.

The Communication Controller enters this state

- When leaving application reset
- When exiting from "HALT" state

To leave "DEFAULT_CONFIG" state the Host has to write SUCC1.CMD = 0001_B in the SUC Configuration Register 1. The Communication Controller transits to "CONFIG" state.

CONFIG State

In "CONFIG" state, the Communication Controller is stopped. All configuration registers are accessible and the pins to the physical layer are in their inactive state. This state is used to initialize the Communication Controller configuration.

The Communication Controller enters this state

- When exiting from “DEFAULT_CONFIG” state
- When exiting from “MONITOR_MODE” or “READY” state

When the state has been entered via “HALT” and “DEFAULT_CONFIG” state, the Host can analyze status information and configuration. Before leaving “CONFIG” state the Host has to assure that the configuration is fault-free.

To leave “CONFIG” state, the Host has to perform the unlock sequence as described on **“LCK” on Page 20-32**. Directly after unlocking the “CONFIG” state the Host has to write **SUCC1.CMD** in the SUC Configuration Register 1 to enter the next state.

Internal counters and the Communication Controller status flags are reset when the Communication Controller leaves “CONFIG”.

*Note: The Message Buffer Status Registers (**MHDS**, **TXRQ1** to **TXRQ4**, **NDAT1** to **NDAT4**, **MBSC1** to **MBSC4**) and status data stored in the Message RAM and are not affected by the transition of the POC from “CONFIG” to “READY” state.*

When the Communication Controller is in “CONFIG” state it is also possible to bring the Communication Controller into a power saving mode by halting the module clocks (f_{SCLK}, f_{CLC_ERAY}). To do this the Host has to assure that all Message RAM transfers have finished before turning off the clocks.

20.6.5.3 MONITOR_MODE

After unlocking “CONFIG” state and writing **SUCC1.CMD** = 0011_B the Communication Controller enters “MONITOR_MODE”. In this mode the Communication Controller is able to receive FlexRay™ Frames and to detect wakeup pattern. The temporal integrity of received Frames is not checked, and therefore cycle counter filtering is not supported. It is not possible to distinguish between static and dynamic frames, because limited functions in Monitor Mode (FRF.RSS will be ignored, filtering not functional).This mode can be used for debugging purposes in case e.g. that startup of a FlexRay™ network fails. After writing **SUCC1.CMD** = 0001_B the Communication Controller transits back to “CONFIG” state.

In MONITOR_MODE the pick first valid mechanism is disabled. This means that a receive Message Buffer may only be configured to receive on one channel. Received Frames are stored into Message Buffers according to Frame ID and receive channel. NULL Frames are handled like Data Frames. After Frame reception only status bits **MBS.VFRA**, **MBS**, **MBS.MLST**, **MBS.RCIS**, **MBS.SFIS**, **MBS.SYNS**, **MBS.NFIS**, **MBS.PPIS**, **MBS.RESS** have valid value.

In “MONITOR_MODE” the Communication Controller is not able to distinguish between CAS and MTS symbols. In case one of these symbols is received on one or both of the two channels, the flags **SIR.MTSA** resp. **SIR.MTSB** are set. **SIR.CAS** has no function in “MONITOR_MODE”.

20.6.5.4 READY State

After unlocking “CONFIG” state and writing **SUCC1.CMD** = 0010_B the Communication Controller enters “READY” state. From this state the Communication Controller can transit to WAKEUP state and perform a cluster wakeup or to “STARTUP” state to perform a coldstart or to integrate into a running communication.

The Communication Controller enters this state

- When exiting from “CONFIG”, “WAKEUP”, “STARTUP”, “NORMAL_ACTIVE”, or “NORMAL_PASSIVE” state by writing **SUCC1.CMD** = 0010_B (READY command).

The Communication Controller exits from this state

- To “CONFIG” state by writing **SUCC1.CMD** = 0001_B (CONFIG command)
- To “WAKEUP” state by writing **SUCC1.CMD** = 0011_B (WAKEUP command)
- To “STARTUP” state by writing **SUCC1.CMD** = 0100_B (RUN command)

Internal counters and the Communication Controller status flags are reset when the Communication Controller enters “STARTUP” state.

*Note: Status bits **MHDS**, registers **TXRQ1** to **TXRQ4**, and status data stored in the Message RAM are not affected by the transition of the POC from “READY” to “STARTUP” state.*

20.6.5.5 WAKEUP State

The description below is intended to help configuring wakeup for the E-Ray IP-module. A detailed description of the wakeup procedure together with the respective SDL diagrams can be found in the FlexRay™ protocol specification v2.1, section 7.1.

The Communication Controller enters this state

- When exiting from “READY” state by writing **SUCC1.CMD** = 0011_B (WAKEUP command).

The Communication Controller exits from this state to “READY” state

- After complete non-aborted transmission of wakeup pattern
- After WUP reception
- After detecting a WUP collision
- After reception of a Frame Header
- By writing **SUCC1.CMD** = 0010_B (READY command)

The cluster wakeup must precede the communication startup in order to ensure that all mechanisms defined for the startup work properly. The minimum requirement for a cluster wakeup is that all bus drivers are supplied with power. A bus driver has the ability to wake up the other components of its node when it receives a wakeup pattern on its channel. At least one node in the cluster needs an **external** wakeup source.

The Host completely controls the wakeup procedure. It is informed about the state of the cluster by the bus driver and the Communication Controller and configures bus guardian

(if available) and Communication Controller to perform the cluster wakeup. The Communication Controller provides to the Host the ability to transmit a special wakeup pattern on each of its available channels separately. The Communication Controller needs to recognize the wakeup pattern only during “WAKEUP” state.

Wakeup may be performed on only one channel at a time. The Host has to configure the wakeup channel while the Communication Controller is in “CONFIG” state by writing bit **SUCC1**.**WUCS** in the SUC Configuration Register 1. The Communication Controller ensures that ongoing communication on this channel is not disturbed. The Communication Controller cannot guarantee that all nodes connected to the configured channel awake upon the transmission of the wakeup pattern, since these nodes cannot give feedback until the startup phase. The wakeup procedure enables single-channel devices in a two-channel system to trigger the wakeup, by only transmitting the wakeup pattern on the single channel to which they are connected. Any coldstart node that deems a system startup necessary will then wake the remaining channel before initiating communication startup.

The wakeup procedure tolerates any number of nodes simultaneously trying to wakeup a single channel and resolves this situation such that only one node transmits the pattern. Additionally the wakeup pattern is collision resilient, so even in the presence of a fault causing two nodes to simultaneously transmit a wakeup pattern, the resulting collided signal can still wake the other nodes.

After wakeup the Communication Controller returns to “READY” state and signals the change of the wakeup status to the Host by setting bit **SIR**.**WST** in the Status Service Request Register. The wakeup status vector can be read from the Communication Controller Status Vector register **CCSV**.**WSV**. If a valid wakeup pattern was received also either flag **SIR**.**WUPA** or flag **SIR**.**WUPB** in the Status Service Request Register is set.

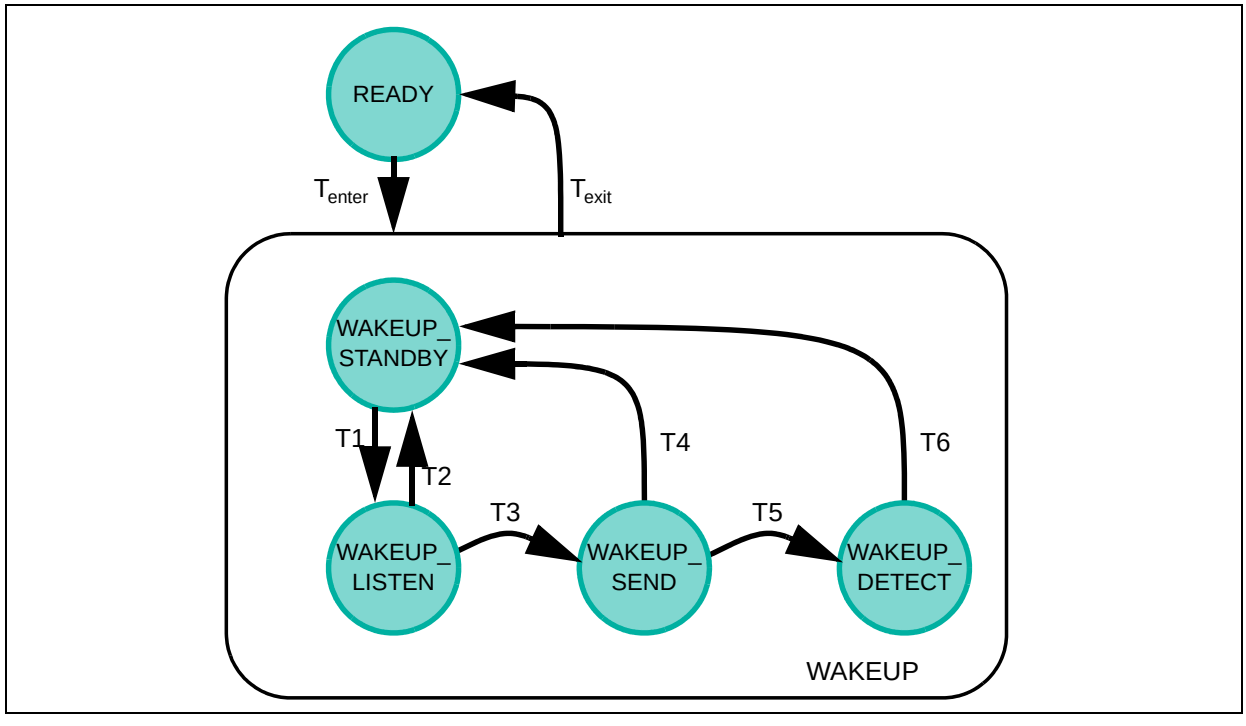


Figure 20-7 Structure of POC State WAKEUP

Table 20-12 State Transitions WAKEUP

T#	Condition	From	To
enter	Host commands change to “WAKEUP” state by writing **SUCC1.CMD** = 0011_B (WAKEUP command)	READY	WAKEUP
1	CHI command WAKEUP triggers wakeup FSM to transit to “WAKEUP_LISTEN” state	WAKEUP_ STANDBY	WAKEUP_ LISTEN
2	Received WUP on wakeup channel selected by flag **SUCC1.WUCS** in the SUC Configuration Register 1 OR Frame Header on either available channel	WAKEUP_ LISTEN	WAKEUP_ STANDBY
3	Timer event	WAKEUP_ LISTEN	WAKEUP_ SEND
4	Complete, non-aborted transmission of wakeup pattern	WAKEUP_ SEND	WAKEUP_ STANDBY
5	Collision detected	WAKEUP_ SEND	WAKEUP_ DETECT
6	Wakeup timer expired OR WUP detected on wakeup channel selected by flag **SUCC1.WUCS** in the SUC Configuration Register 1 OR Frame Header received on either available channel	WAKEUP_ DETECT	WAKEUP_ STANDBY
exit	Wakeup completed (after T2 or T4 or T6) OR Host commands change to “READY” state by writing **SUCC1.CMD** = 0010_B (READY command). This command also resets the wakeup FSM to “WAKEUP_STANDBY” state	WAKEUP	READY

The “WAKEUP_LISTEN” state is controlled by the wakeup timer and the wakeup noise timer. The two timers are controlled by the parameters listen timeout **SUCC2.LT** and listen timeout noise **SUCC2.LTN**. Both values can be configured in the SUC Configuration Register 2. listen timeout enables a fast cluster wakeup in case of a noise free environment, while listen timeout noise enables wakeup under more difficult conditions regarding noise interference.

In “WAKEUP_SEND” state the Communication Controller transmits the wakeup pattern on the configured channel and checks for collisions. After return from wakeup the Host has to bring the Communication Controller into “STARTUP” state by CHI command RUN.

In “WAKEUP_DETECT” state the Communication Controller attempts to identify the reason for the wakeup collision detected in “WAKEUP_SEND” state. The monitoring is

bounded by the expiration of listen timeout as configured by **SUCC2.LT** in the SUC Configuration Register 2. Either the detection of a wakeup pattern indicating a wakeup attempt by another node or the reception of a Frame Header indication existing communication, causes the direct transition to “READY” state. Otherwise WAKEUP_DETECT is left after expiration of listen timeout; in this case the reason for wakeup collision is unknown.

The Host has to be aware of possible failures of the wakeup and act accordingly. It is advisable to delay any potential startup attempt of the node having instigated the wakeup by the minimal time it takes another coldstart node to become awake and to be configured.

The FlexRay™ Protocol Specification v2.1 recommends that two different Communication Controllers shall awake the two channels.

Host activities

The host must coordinate the wakeup of the two channels and must decide whether, or not, to wake a specific channel. The sending of the wakeup pattern is initiated by the Host and generated by the Communication Controller. The wakeup pattern is detected by the remote BDs and signalled to their local Hosts.

Wakeup procedure controlled by Host (single-channel wakeup):

- Configure the Communication Controller in “CONFIG” state
  - Select wakeup channel by programming bit **SUCC1.WUCS**
- Check local BDs whether a WUP was received
- Activate BD of selected wakeup channel
- Command Communication Controller to start wakeup on the configured channel by writing **SUCC1.CMD** = 0011_B
  - Communication Controller enters “WAKEUP
  - Communication Controller returns to “READY” state and signals status of wakeup attempt to Host
- Wait predefined time to allow the other nodes to wakeup and configure themselves
- Coldstart node: wait for WUP on the other channel
  - In a dual channel cluster wait for WUP on the other channel
  - Reset coldstart inhibit flag **CCSV.CSI** by writing **SUCC1.CMD** = 1001_B (ALLOW_COLDSTART command))
- Reset Coldstart Inhibit flag **CCSV.CSI** in the CCSV register by writing **SUCC1.CMD** = 1001_B (ALLOW_COLDSTART command), coldstart node only
- Command Communication Controller to enter startup by writing **SUCC1.CMD** = 0100_B (RUN command)

Wakeup procedure triggered by BD:

- Wakeup recognized by BD
- BD triggers power-up of Host (if required)
- BD signals wakeup event to Host

- Host configures its local Communication Controller
- If necessary Host commands wakeup of second channel and waits predefined time to allow the other nodes to wakeup and configure themselves
- Host commands Communication Controller to enter “STARTUP” state by writing **SUCC1.CMD** = 0100_B (RUN command)

Wakeup pattern (WUP)

The wakeup pattern is composed of at least two wakeup symbols (WUS). Wakeup symbol and wakeup pattern are configured by the PRT Configuration Registers **PRTC1** and **PRTC2**.

- Single channel wakeup, wakeup symbol may not be sent on both channels at the same time
- Wakeup symbol collision resilient for up to two sending nodes (two overlapping wakeup symbols still recognizable)
- Wakeup symbol must be configured identical in all nodes of a cluster
- Wakeup symbol transmit low time configured by **PRTC2.TXL**
- Wakeup symbol idle time used to listen for activity on the bus, configured by **PRTC2.TXI**
- A wakeup pattern composed of at least two Tx-wakeup symbols needed for wakeup
- Number of repetitions configurable by **PRTC1.RWP** (2 to 63 repetitions)
- Wakeup symbol receive window length configured by **PRTC1.RXW**
- Wakeup symbol receive low time configured by **PRTC2.RXL**
- Wakeup symbol receive idle time configured by **PRTC2.RXI**

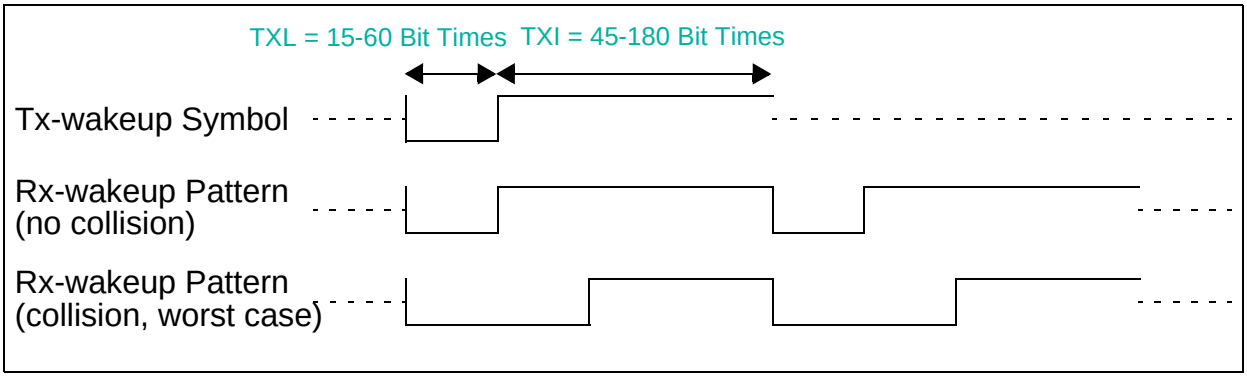


Figure 20-8 Timing of Wakeup Pattern

20.6.5.6 STARTUP State

The description below is intended to help configuring startup for the E-Ray IP-module. A detailed description of the startup procedure together with the respective SDL diagrams can be found in the FlexRay™ protocol specification v2.1, section 7.2.

Any node entering “STARTUP” state that has coldstart capability should assure that both channels attached have been awakened before initiating coldstart.

It cannot be assumed that all nodes and stars need the same amount of time to become completely awake and to be configured. Since at least two nodes are necessary to start up the cluster communication, it is advisable to delay any potential startup attempt of the node having instigated the wakeup by the minimal amount of time it takes another coldstart node to become awake, to be configured and to enter startup. It may require several hundred milliseconds (depending on the hardware used) before all nodes and stars are completely awakened and configured.

Startup is performed on all channels synchronously. During startup, a node only transmits startup Frames.

A fault-tolerant, distributed startup strategy is specified for initial synchronization of all nodes. In general, a node may enter “NORMAL_ACTIVE” state via (see **Figure 20-9**):

- Coldstart path initiating the schedule synchronization (leading coldstart node)
- Coldstart path joining other coldstart nodes (following coldstart node)
- Integration path integrating into an existing communication schedule (all other nodes)

A coldstart attempt begins with the transmission of a collision avoidance symbol (CAS). Only a coldstart node that had transmitted the CAS transmits Frames in the first four cycles after the CAS, it is then joined firstly by the other coldstart nodes and afterwards by all other nodes.

A coldstart node has the Transmit SYNC Frame in Key Slot bits **SUCC1.TXST** and **SUCC1.TXSY** in the SUC Configuration Register 1 set to 1. The Message Buffer 0 holds the key slot ID which defines the slot number where the Startup Frame is send. In the Frame Header of the Startup Frame the Startup Frame indicator bit is set.

In clusters consisting of three or more nodes, at least three nodes shall be configured to be coldstart nodes. In clusters consisting of two nodes, both nodes must be coldstart nodes. At least two fault-free coldstart nodes are necessary for the cluster to startup.

Each Startup Frame must also be a SYNC Frame; therefore each coldstart node will also be a sync node. The number of coldstart attempts is configured by **SUCC1.CSA** in the SUC Configuration Register 1.

A non-coldstart node requires at least two startup Frames from distinct nodes for integration. It may start integration before the coldstart nodes have finished their startup. It will not finish its startup until at least two coldstart nodes have finished their startup.

Both non-coldstart nodes and coldstart nodes start passive integration via the integration path as soon as they receive SYNC Frames from which to derive the TDMA schedule information. During integration the node has to adapt its own clock to the global clock (rate and offset) and has to make its cycle time consistent with the global schedule observable at the network. Afterwards, these settings are checked for consistency with all available network nodes. The node can only leave the integration phase and actively participate in communication when these checks are passed.

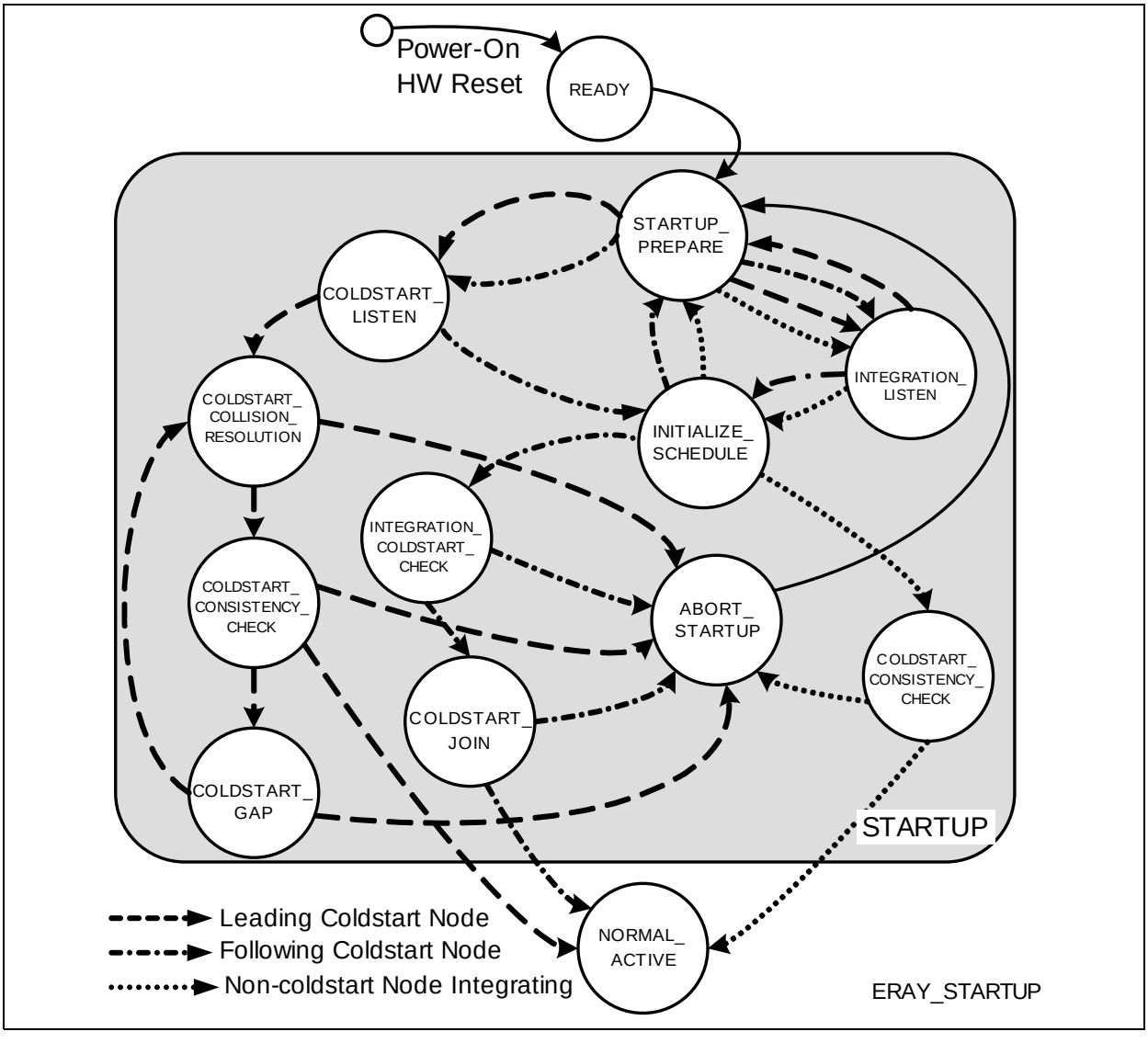


Figure 20-9 State Diagram Time-Triggered Startup

Coldstart Inhibit Mode

In coldstart inhibit mode the node is prevented from initializing the TDMA communication schedule. If bit **CCSV.CSI** in the Communication Controller Status Vector register is set, the node is not allowed to initialize the cluster communication, i.e. entering the coldstart path is prohibited. The node is allowed to integrate to a running cluster or to transmit startup Frames after another coldstart node started the initialization of the cluster communication.

The coldstart inhibit bit **CCSV.CSI** is set whenever the POC enters “READY” state. The bit has to be cleared under control of the Host by CHI command ALLOW_COLDSTART (**SUCC1.CMD** = 1001_B)

20.6.5.7 Startup Timeouts

The Communication Controller supplies two different Microtick timers supporting two timeout values, startup timeout and startup noise timeout. The two timers are reset when the Communication Controller enters the “COLDSTART_LISTEN” state. The expiration of either of these timers causes the node to leave the initial sensing phase (“COLDSTART_LISTEN” state) with the intention of starting up communication.

*Note: The startup and startup noise timers are identical with the wakeup and wakeup noise timers and use the same configuration values **SUCC2.LT** and **SUCC2.LTN** from the SUC Configuration Register 2.*

Startup Timeout

The startup timeout limits the listen time used by a node to determine if there is already communication between other nodes or at least one coldstart node actively requesting the integration of others.

The startup timer is configured by programming **SUCC2.LT** (pdListenTimeout) in the SUC Configuration Register 2.

The startup timer is restarted upon:

- Entering the “COLDSTART_LISTEN” state
- Both channels reaching idle state while in “COLDSTART_LISTEN” state

The startup timer is stopped:

- If communication channel activity is detected on one of the configured channels while the node is in the “COLDSTART_LISTEN” state
- When the “COLDSTART_LISTEN” state is left

Once the startup timeout expires, neither an overflow nor a cyclic restart of the timer is performed. The timer status is kept for further processing by the startup state machine.

Startup Noise Timeout

At the same time the startup timer is started for the first time (transition from “STARTUP_PREPARE” state to “COLDSTART_LISTEN” state), the startup noise timer is started. This additional timeout is used to improve reliability of the startup procedure in the presence of noise.

The startup noise timer is configured by programming **SUCC2.LTN** (gListenNoise - 1) in the SUC Configuration Register 2 (see **“SUC Configuration Register 2 (SUCC2)” on Page 20-88**).

The startup noise timeout is:
pdListenTimeout • ´gListenNoise = **SUCC2.LT** • (**SUCC2.LTN** + 1)

The startup noise timer is restarted upon:

- Entering the “COLDSTART_LISTEN” state
- Reception of correctly decoded Headers or CAS symbols while the node is in “COLDSTART_LISTEN” state

The startup noise timer is stopped when the “COLDSTART_LISTEN” state is left.

Once the startup noise timeout expires, neither an overflow nor a cyclic restart of the timer is performed. The status is kept for further processing by the startup state machine. Since the startup noise timer won’t be restarted when random channel activity is sensed, this timeout defines the fall-back solution that guarantees that a node will try to start up the communication cluster even in the presence of noise.

20.6.5.8 Path of leading Coldstart Node (initiating coldstart)

When a coldstart node enters “COLDSTART_LISTEN”, it listens to its attached channels.

If no communication is detected, the node enters the “COLDSTART_COLLISION_RESOLUTION” state and commences a coldstart attempt. The initial transmission of a CAS symbol is succeeded by the first regular cycle. This cycle has the number zero.

From cycle zero on, the node transmits its startup Frame. Since each coldstart node is allowed to perform a coldstart attempt, it may occur that several nodes simultaneously transmit the CAS symbol and enter the coldstart path. This situation is resolved during the first four cycles after CAS transmission.

As soon as a node that initiates a coldstart attempt receives a CAS symbol or a Frame Header during these four cycles, it re-enters the “COLDSTART_LISTEN” state. Thereby, only one node remains in this path. In cycle four, other coldstart nodes begin to transmit their startup Frames.

After four cycles in “COLDSTART_COLLISION_RESOLUTION” state, the node that initiated the coldstart enters the “COLDSTART_CONSISTENCY_CHECK” state. It collects all startup Frames from cycle four and five and performs the clock correction. If the clock correction does not deliver any errors and it has received at least one valid Startup Frame pair, the node leaves “COLDSTART_CONSISTENCY_CHECK” and enters “NORMAL_ACTIVE” state.

The number of coldstart attempts that a node is allowed to perform is configured by **SUCC1.CSA** in the SUC Configuration Register 1. The number of remaining coldstarts attempts **CCSV.RCA** can be read from Communication Controller Status Vector register. The number of remaining attempts is reduced by one for each attempted coldstart. A node may enter the “COLDSTART_LISTEN” state only if this value is larger than one and it may enter the “COLDSTART_COLLISION_RESOLUTION” state only if this value is larger than zero. If the number of coldstart attempts is one, coldstart is inhibited but integration is still possible.

Path of following Coldstart Node (responding to leading Coldstart Node)

When a coldstart node enters the “COLDSTART_LISTEN” state, it tries to receive a valid pair of startup Frames to derive its schedule and clock correction from the leading coldstart node.

As soon as a valid Startup Frame has been received the “INITIALIZE_SCHEDULE” state is entered. If the clock synchronization can successfully receive a matching second valid Startup Frame and can derive a schedule from this startup Frames, the “INTEGRATION_COLDSTART_CHECK” state is entered.

In “INTEGRATION_COLDSTART_CHECK” state it is assured that the clock correction can be performed correctly and that the coldstart node from which this node has initialized its schedule is still available. The node collects all SYNC Frames and performs clock correction in the following double-cycle. If clock correction does not signal any errors and if the node continues to receive sufficient Frames from the same node it has integrated on, the “COLDSTART_JOIN” state is entered.

In “COLDSTART_JOIN” state integrating coldstart nodes begin to transmit their own startup Frames. Thereby the node that initiated the coldstart and the nodes joining it can check if their schedules agree to each other. If for the following three cycles the clock correction does not signal errors and at least one other coldstart node is visible, the node leaves “COLDSTART_JOIN” state and enters “NORMAL_ACTIVE” state. Thereby it leaves “STARTUP” at least one cycle after the node that initiated the coldstart.

Path of Non-coldstart Node

When a non-coldstart node enters the INTEGRATION_LISTEN state, it listens to its attached channels and tries to receive FlexRay™ Frames.

As soon as a valid Startup Frame has been received the “INITIALIZE_SCHEDULE” state is entered. If the clock synchronization can successfully receive a matching second valid Startup Frame and derive a schedule from this, the INTEGRATION_CONSISTENCY_CHECK state is entered.

In “INTEGRATION_CONSISTENCY_CHECK” state it is verified that the clock correction can be performed correctly and that enough coldstart nodes (at least 2) send startup Frames that agree to the nodes own schedule. Clock correction is activated, and if any errors are signalled, the integration attempt is aborted.

During the first even cycle in this state, either two valid startup Frames or the Startup Frame of the node that this node has integrated on must be received; otherwise the node aborts the integration attempt.

During the first double-cycle in this state, either two valid Startup Frame pairs or the Startup Frame pair of the node that this node has integrated on must be received; otherwise the node aborts the integration attempt.

If after the first double-cycle less than two valid startup Frames are received within an even cycle, or less than two valid Startup Frame pairs are received within a double-cycle, the startup attempt is aborted.

Nodes in this state need to see two valid Startup Frame pairs for two consecutive double-cycles each to be allowed to leave STARTUP and enter NORMAL_OPERATION. Consequently, they leave startup at least one double-cycle after the node that initiated the coldstart and only at the end of a cycle with an odd cycle number.

20.6.5.9 NORMAL_ACTIVE State

As soon as the node that transmitted the first CAS symbol (resolving the potential access conflict and entering “STARTUP” via coldstart path) and one additional node have entered the “NORMAL_ACTIVE” state, the startup phase for the cluster has finished. In the “NORMAL_ACTIVE” state, all configured messages are scheduled for transmission. This includes all Data Frames as well as the SYNC Frames. Rate and offset measurement is started in all even cycles (even/odd cycle pairs required).

In “NORMAL_ACTIVE” state the Communication Controller supports regular communication functions

- The Communication Controller performs transmissions and reception on the FlexRay™ bus as configured
- Clock synchronization is running
- The Host interface is operational

The Communication Controller exits from that state to

- “HALT” state by writing **SUCC1.CMD** = 0110_B (HALT command, at the end of the current cycle)
- “HALT” state by writing **SUCC1.CMD** = 0111_B (FREEZE command, immediately)
- “HALT” state due to change of the error state from “ACTIVE” to “COMM_HALT”
- “NORMAL_PASSIVE” state due to change of the error state from “ACTIVE” to “PASSIVE”
- “READY” state by writing **SUCC1.CMD** = 0010_B (READY command)

20.6.5.10 NORMAL_PASSIVE State

“NORMAL_PASSIVE” state is entered from “NORMAL_ACTIVE” state when the error state changes from ACTIVE (green) to PASSIVE (yellow).

In “NORMAL_PASSIVE” state, the node is able to receive all Frames (node is fully synchronized and performs clock synchronization). In comparison to the “NORMAL_ACTIVE” state the node does not actively participate in communication, i.e. neither symbols nor Frames are transmitted.

In “NORMAL_PASSIVE” state

- The Communication Controller performs reception on the FlexRay™ bus
- The Communication Controller does not transmit any Frames or symbols on the FlexRay™ bus
- Clock synchronization is running
- The Host interface is operational

The Communication Controller exits from this state to

- “HALT” state by writing **SUCC1.CMD** = 0110_B (HALT command, at the end of the current cycle)
- “HALT” state by writing **SUCC1.CMD** = 0111_B (FREEZE command, immediately)
- “HALT” state due to change of the error state from “PASSIVE” to “COMM_HALT”
- “NORMAL_ACTIVE” state due to change of the error state from “PASSIVE” to “ACTIVE”. The transition takes place when **CCEV.PTAC** from the Communication Controller Error Vector register equals **SUCC1.PTA** - 1.
- “READY” state by writing **SUCC1.CMD** = 0010_B (READY command)

20.6.5.11 HALT State

In this state all communication (reception and transmission) is stopped.

The Communication Controller enters this state

- By writing **SUCC1.CMD** = 0110_B (HALT command) while the Communication Controller is in “NORMAL_ACTIVE” or “NORMAL_PASSIVE” state
- By writing **SUCC1.CMD** = 0111_B (FREEZE command) from all states
- When exiting from “NORMAL_ACTIVE” state because the clock correction failed counter reached the “maximum without clock correction fatal” limit
- When exiting from “NORMAL_PASSIVE” state because the clock correction failed counter reached the “maximum without clock correction fatal” limit

The Communication Controller exits from this state to “CONFIG” state

- By writing **SUCC1.CMD** = 0001_B (DEFAULT_CONFIG command)

When the Communication Controller enters “HALT” state, all configuration and status data is maintained for analyzing purposes.

When the Host writes **SUCC1.CMD** = 0110_B (HALT command) in the SUC Configuration Register 1 to 1, the Communication Controller sets bit **CCSV.HRQ** in the Communication Controller Status Vector register and enters “HALT” state after the current communication cycle has finished.

When the Host writes **SUCC1.CMD** = 0111_B (FREEZE command) in the SUC Configuration Register to 1, the Communication Controller enters “HALT” state immediately and sets the **CCSV.FSI** bit in the Communication Controller Status Vector register.

The POC state from which the transition to HALT state took place can be read from **CCSV.PSL**.

20.6.6 Network Management

The accrued Network Management (NM) vector is located in the Network Management Register 1 to Network Management Register 3 (**NMVx (x = 1-3)**). The Communication Controller performs a logical OR operation over all Network Management (NM) vectors out of all received valid Network Management (NM) Frames with the Payload Preamble Indicator (PPI) bit set. Only a static Frame may be configured to hold Network Management (NM) information. The Communication Controller updates the Network Management (NM) vector at the end of each cycle.

The length of the Network Management (NM) vector can be configured from 0 to 12 byte by NML in the NEM Configuration Register. The Network Management (NM) vector length must be configured identically in all nodes of a cluster.

To configure a transmit buffer to send FlexRay™ Frames with the PPI bit set, the PPIT bit in the Header Section of the respective transmit buffer has to be set via **WRHS1.PPIT**. In addition the Host has to write the Network Management (NM) information to the Data Section of the respective transmit buffer.

The evaluation of the Network Management (NM) vector has to be done by the application running on the Host.

*Note: In case a Message Buffer is configured for transmission / reception of Network Management Frames, the payload length configured in Header 2 of that Message Buffer should be equal or greater than the length of the NM Vector configured by **NEMC.NML**.*

When the Communication Controller transits to “HALT” state, the cycle count is not incremented and therefore the NM Vector is not updated. In this case NMV1 to NMV3 holds the value from the cycle before.

20.6.7 Filtering and Masking

Filtering is done by checking specific fields in a received Frame against the corresponding configuration constants of the valid Message Buffers and the actual slot and cycle counter values (acceptance filtering), or by comparing the configuration constants of the valid Message Buffers against the actual slot and cycle counter values (transmit filtering). A Message Buffer is only updated / transmitted if the required matches occur.

Filtering is done on the following fields:

- Channel ID
- Frame ID
- Cycle Counter

The following filter combinations for acceptance / transmit filtering are allowed:

- Frame ID + Channel ID
- Frame ID + Channel ID + Cycle Counter

In order to store a received message in a Message Buffer all configured filters must match.

Note: For the FIFO the acceptance filter is configured by the FIFO Rejection Filter and the FIFO Rejection Filter Mask.

A message will be transmitted in the time slot corresponding to the configured Frame ID on the configured channel(s). If cycle counter filtering is enabled the configured cycle filter value must also match.

20.6.7.1 Frame ID Filtering

Every transmit and receive buffer contains a Frame ID stored in the Header Section. This Frame ID is used differently for receive and transmit buffers.

Receive Buffers

A received message is stored in the first receive buffer where the received Frame ID matches the configured Frame ID, provided channel ID and cycle counter criteria are also met.

Transmit Buffers

For transmit buffers the configured Frame ID is used to determine the appropriate slot for message transmission. The Frame will be transmitted in the time slot corresponding to the configured Frame ID, provided channel ID and cycle counter criteria are also met.

20.6.7.2 Channel ID Filtering

There is a 2-bit channel filtering field (CHA, CHB) located in the Header Section of each Message Buffer in the Message RAM. It serves as a filter for receive buffers, and as a control field for transmit buffers (see **Table 20-13**).

Table 20-13 Channel Filtering Configuration

CHA	CHB	**Transmit Buffer** transmit Frame	**Receive Buffer** store valid receive Frame
1	1	on both channels (static segment only)	received on channel A or B (store first semantically valid Frame, static segment only)
1	0	on channel A	received on channel A
0	1	on channel B	received on channel B
0	0	no transmission	ignore Frame

Note: If a Message Buffer is configured for the dynamic segment and both bits of the channel filtering field are set to 1, no Frames are transmitted resp. received Frames are ignored (same function as CHA = CHB = 0)

Receive Buffers

Valid received Frames are stored if they are received on the channels specified in the channel filtering field. Only in static segment a receive buffer may be setup for reception on both channels (CHA and CHB set). Other filtering criteria must also be met.

If a valid Header Segment was stored, the respective MBC flag in the Message Buffer Status Changed register is set. If a valid Payload Segment was stored, the respective **NDn (n = 0-31)** to **NDn (n = 96-127)** flag in the New Data **NDAT1** to **NDAT4** register is set. In both cases, if bit **RDHS1.MBI** in the Header Section of the respective Message Buffer is set, the RXI flag in the Status Service Request Register is set to 1. If enabled an service request is generated.

Transmit Buffers

The content of the buffer is transmitted only on the channels specified in the channel filtering field when the Frame ID filtering and cycle counter filtering criteria are also met. Only in static segment a transmit buffer may be setup for transmission on both channels (CHA and CHB set). After transmission has completed, and if bit **WRHS1.MBI** in the Header Section of the respective Message Buffer is set, the TXI flag in the Status Service Request Register is set to 1. If enabled an service request is generated.

20.6.7.3 Cycle Counter Filtering

Cycle counter filtering is based on the notion of a cycle set. For filtering purposes, a match is detected if any one of the elements of the cycle set is matched. The cycle set is defined by the cycle code field in the Header Section of each Message Buffer.

If Message Buffer 0 is configured to hold the startup / SYNC Frame or the single slot Frame by bits TXST, TXSY, and TSM in the SUC Configuration Register 1, cycle counter filtering for Message Buffer 0 should be disabled.

*Note: Sharing of a static time slot via cycle counter filtering between different nodes of a FlexRay™ network is **not** allowed.*

The set of cycle numbers belonging to a cycle set is determined as described in **Table 20-14**.

Table 20-14 Definition of Cycle Set

Cycle Code	Matching Cycle Counter Values		
$000000x_B$	all Cycles		
$000001c_B$	every second Cycle	at (Cycle Count)mod2	= c
$00001cc_B$	every fourth Cycle	at (Cycle Count)mod4	= cc
$0001ccc_B$	every eighth Cycle	at (Cycle Count)mod8	= ccc
$001cccc_B$	every sixteenth Cycle	at (Cycle Count)mod16	= cccc
$01ccccc_B$	every thirty-second Cycle	at (Cycle Count)mod32	= ccccc
$1cccccc_B$	every sixty-fourth Cycle	at (Cycle Count)mod64	= cccccc

Table 20-15 below gives some examples for valid cycle sets to be used for cycle counter filtering:

Table 20-15 Examples for Valid Cycle Sets

Cycle Code	Matching Cycle Counter Values
0000011_B	1-3-5-7- ….-63 ↵
0000100_B	0-4-8-12- ….-60 ↵
0001110_B	6-14-22-30- ….-62 ↵
0011000_B	8-24-40-56 ↵
0100011_B	3-35 ↵
1001001_B	9 ↵

Receive Buffers

The received message is stored only if the received cycle counter matches an element of the receive buffer's cycle set. Channel ID and Frame ID criteria must also be met.

Transmit Buffers

The content of the buffer is transmitted on the configured channels when an element of the cycle set matches the current cycle counter value and the Frame ID matches the slot counter value.

20.6.7.4 FIFO Filtering

For FIFO filtering there is one rejection filter and one rejection filter mask available. The FIFO rejection filter consists of 20 bits for **Channel** (2 bits), **Frame ID** (11 bits), and **Cycle Code** (7 bits). Rejection filter and rejection filter mask can be configured in

DEFAIULT_CONGIF or “CONFIG” state only. The filter configuration in the Header Sections of Message Buffers belonging to the FIFO is ignored.

A valid received Frame is stored in the FIFO if channel ID, Frame ID, and cycle counter are not rejected by the configured rejection filter and rejection filter mask, and if there is no matching dedicated receive buffer.

20.6.8 Transmit Process

The transmit process is described in the following sections.

20.6.8.1 Static Segment

For the static segment, if there are several messages pending for transmission, the message with the Frame ID corresponding to the next sending slot is selected for transmission.

The Data Section of transmit buffers assigned to the static segment can be updated until the end of the preceding time slot. This means that a transfer from the Input Buffer has to be started by writing to the Input Buffer Command Request register latest at this time.

20.6.8.2 Dynamic Segment

In the dynamic segment, if several messages are pending, the message with the highest priority (lowest Frame ID) is selected next. Only Frame ID’s which are higher than the largest static Frame ID are allowed for the dynamic segment.

In the dynamic segment different slot counter sequences are possible (concurrent sending of different Frame ID’s on both channels). Therefore pending messages are selected according to their Frame ID and their channel configuration bit.

The Data Section of transmit buffers assigned to the dynamic segment can be updated until the end of the preceding slot. This means that a transfer from the Input Buffer has to be started by writing to the Input Buffer Command Request register latest at this time.

The start of latest transmit configured by SLT in the MHD Configuration Register 1 defines the maximum minislot value allowed before inhibiting new Frame transmission in the dynamic segment of the current cycle.

20.6.8.3 Transmit Buffers

A portion of the E-Ray Message Buffers can be configured as transmit buffers by programming bit CFG in the Header Section of the respective Message Buffer to 1. This can be done via the Write Header Section 1 register.

There exist the following possibilities to assign a transmit buffer to the Communication Controller channels:

- Static segment: channel A **or** channel B, channel A **and** channel B

- Dynamic segment: channel A **or** channel B

Message Buffer 0 is dedicated to hold the startup Frame, the SYNC Frame, or the designated single slot Frame as configured by TXST, TXSY, and TSM in the SUC Configuration Register 1. In this case it can be reconfigured in “DEFAULT_CONFIG” or “CONFIG” state only. This ensures that any node transmits at most one startup / SYNC Frame per communication cycle. Transmission of startup / SYNC Frames from other Message Buffers is not possible.

All other Message Buffers configured for transmission in static or dynamic segment are reconfigurable during runtime. Due to the organization of the Data Partition in the Message RAM (reference by data pointer), reconfiguration of the configured payload length and the data pointer in the Header Section of a Message Buffer may lead to erroneous configurations. If a Message Buffer is reconfigured during runtime it may happen that this Message Buffer is not send out in the respective communication cycle.

The Communication Controller does not have the capability to calculate the Header CRC. The Host is supposed to provide the Header CRCs for all transmit buffers. If Network Management is required the Host has to set the PPIT bit in the Header Section of the respective Message Buffer to 1 and write the Network Management information to the Data Section of the Message Buffer (see **Section 20.6.6**).

The payload length field configures the data payload length in 2-byte words. If the configured payload length of a static transmit buffer is shorter than the payload length configured for the static segment by SFDL in the Message Handler Configuration Register 1, the Communication Controller generates padding byte to ensure that Frames have proper physical length. The padding pattern is logical zero.

Each transmit buffer provides a transmission mode flag TXM that allows the Host to configure the transmission mode for the transmit buffer in the static segment. If this bit is set, the transmitter operates in the single-shot mode. If this bit is cleared, the transmitter operates in the continuous mode. In dynamic segment the transmitter always works in single-shot mode.

If a Message Buffer is configured in the continuous mode, the Communication Controller does not reset the transmission request flag TXR after successful transmission. In this case a Frame is sent out each time the Frame ID and cycle counter filter match. The TXR flag can be reset by the Host by writing the respective Message Buffer number to the Input Buffer Command Request register while bit **STXRH** in the Input Buffer Command Mask register is reset to 0.

If two or more transmit buffers are configured with the same Frame ID **and** cycle counter filter value, the transmit buffer with the lowest Message Buffer number will be transmitted in the respective slot.

20.6.8.4 Frame Transmission

To prepare a transmit buffer for transmission the following steps are required:

- Configure the Message Buffer as transmit buffer by writing bit CFG = 1 in the Write Header Section 1 register
- Write transmit message (Header and Data Section) to the Input Buffer.
- To transfer a transmit message from Input Buffer to the Message RAM proceed as described on **“Data Transfer from Input Buffer to Message RAM” on Page 20-224**.
- If configured in the Input Buffer Command Mask register the Transmission Request flag for the respective Message Buffer will be set as soon as the transfer has completed, and the Message Buffer is ready for transmission.
- Check whether the Message Buffer has been transmitted by checking the TXR bits (TXR = 0) in the Transmission Request 1,2 registers (single-shot mode only).

In single-shot mode the Communication Controller resets the TXR flag after transmission has been completed. Now the Host may update the transmit buffer with the next message. The Communication Controller does not transmit the message before the Host has indicated that the update is completed by setting the Transmission Request flag TXR again. The Host can check the actual state of the TXR flags of all Message Buffers by reading the Transmission Request registers. After successful transmission, if bit **WRHS1.MBI** in the Header Section of the respective Message Buffer is set, the transmit service request flag in the Status Service Request Register is set (TXI = 1). If enabled an service request is generated.

20.6.8.5 NULL Frame Transmission

If in static segment the Host does not set the transmission request flag before transmit time, and if there is no other transmit buffer with matching filter criteria (matching Frame ID and cycle counter filter), the Communication Controller transmits a NULL Frame with the NULL Frame indication bit reset to 0 and the payload data reset to zero.

In the following cases the Communication Controller transmits a NULL Frame with the NULL Frame indication bit reset to 0, and the rest of the Frame Header and the Frame length unchanged (payload data is reset to zero):

- All transmit buffers configured for the slot have cycle counter filters that do not match the current cycle
- There are matching Frame ID’s and cycle counter filters, but none of these transmit buffers has the transmission request flag TXR set

NULL Frames are not transmitted in the dynamic segment.

20.6.9 Receive Process

The receive process is described in the following sections.

20.6.9.1 Frame Reception

To prepare or change a Message Buffer for reception the following steps are required:

- Configure the Message Buffer as receive buffer by writing bit CFG = 0 in the Write Header Section 1 register
- Configure the receive buffer by writing the configuration data (Header Section) to the Input Buffer
- Transfer the configuration from Input Buffer to the Message RAM by writing the number of the target Message Buffer to the Input Buffer Command Request register.

Once these steps are performed, the Message Buffer functions as an active receive buffer and participates in the internal acceptance filtering process, which takes place every time the Communication Controller receives a message. The first matching receive buffer is updated from the received message. If the Message Buffer holds an unprocessed Data Section (ND = 1) it is overwritten with the new message and the MLST bit in the respective Message Buffer Status register is set.

If the payload length of a received Frame PLC is longer than the value programmed by PLC in the Header Section of the respective Message Buffer, the data field stored in the Message Buffer is truncated to that length.

If no Frame, a NULL Frame, or a corrupted Frame is received in a slot, the Data Section of the Message Buffer configured for this slot is not updated. In this case only the flags in the Message Buffer Status register are updated to signal the cause of the problem. In addition the respective MBC flag in the Message Buffer Status Changed 1,2,3,4 registers is set.

When the Data Section of a receive buffer has been updated from a received Frame, the respective New Data **NDn (n = 0-31)** to **NDn (n = 96-127)** flag in the New Data **NDAT1** to **NDAT4** registers is set. When the Message Handler has updated the Message Buffer status, the respective MBC flag in the Message Buffer Status Changed 1,2,3,4 registers is set. If bit **RDHS1.MBI** in the Header Section of the respective Message Buffer is set, the receive service request flag in the Status Service Request Register is set (RXI = 1). If enabled an service request is generated.

To read a receive buffer from the Message RAM via the Output Buffer proceed as described on **“Data Transfer from Message RAM to Output Buffer” on Page 20-226**.

Note: The ND and MBC flags are automatically cleared by the Message Handler when the received message has been transferred to the Output Buffer.

20.6.9.2 NULL Frame reception

The Payload Segment of a received NULL Frame is **not** copied into the matching receive buffer. If a NULL Frame has been received, the Header Section of the matching Message Buffer is updated from the received NULL Frame. The NULL Frame indication bit in the Header Section 3 of the respective Message Buffer is reset (NFI = 0) and the respective MBC flag in the Message Buffer Status Changed 1,2,3,4 registers is set.

In case that bit ND and / or MBC were already set before this event because the Host did not read the last received message, bit MLST in the Message Buffer Status register of the respective Message Buffer is also set.

20.6.10 FIFO Function

A group of the Message Buffers can be configured as a cyclic First-In-First-Out (FIFO). The group of Message Buffers belonging to the FIFO is contiguous in the register map starting with the Message Buffer referenced by FFB and ending with the Message Buffer referenced by LCB in the Message RAM Configuration register. Up to 128 Message Buffers can be assigned to the FIFO.

20.6.10.1 Description

Every valid incoming message not matching with any dedicated receive buffer but passing the programmable FIFO filter is stored into the FIFO. In this case Frame ID, payload length, receive cycle count, and the status bits of the addressed FIFO Message Buffer are overwritten with Frame ID, payload length, receive cycle count, and the status from the received message and can be read by the Host for message identification. Bit RFNE in the Status Service Request Register shows that the FIFO is not empty, bit RFF in the Status Service Request Register is set when the last available Message Buffer belonging to the FIFO is written, bit RFO in the Error Service Request Register shows that a FIFO overrun has been detected. If enabled, service requests are generated.

There are two index registers associated with the FIFO. The PUT Index Register (PIDX) is an index to the next available location in the FIFO. When a new message has been received it is written into the Message Buffer addressed by the PIDX register. The PIDX register is then incremented and addresses the next available Message Buffer. If the PIDX register is incremented past the highest numbered Message Buffer of the FIFO, the PIDX register is loaded with the number of the first (lowest numbered) Message Buffer in the FIFO chain. The GET Index Register (GIDX) is used to address the next Message Buffer of the FIFO to be read. The GIDX register is incremented after transfer of the contents of a Message Buffer belonging to the FIFO to the Output Buffer. The PUT Index Register and the GET Index Register are not accessible by the Host.

The FIFO is completely filled when the PUT index (PIDX) reaches the value of the GET index (GIDX). When the next message is written to the FIFO before the oldest message has been read, both PUT index and GET index are incremented and the new message

overwrites the oldest message in the FIFO. This will set FIFO overrun flag RFO in the Error Service Request Register.

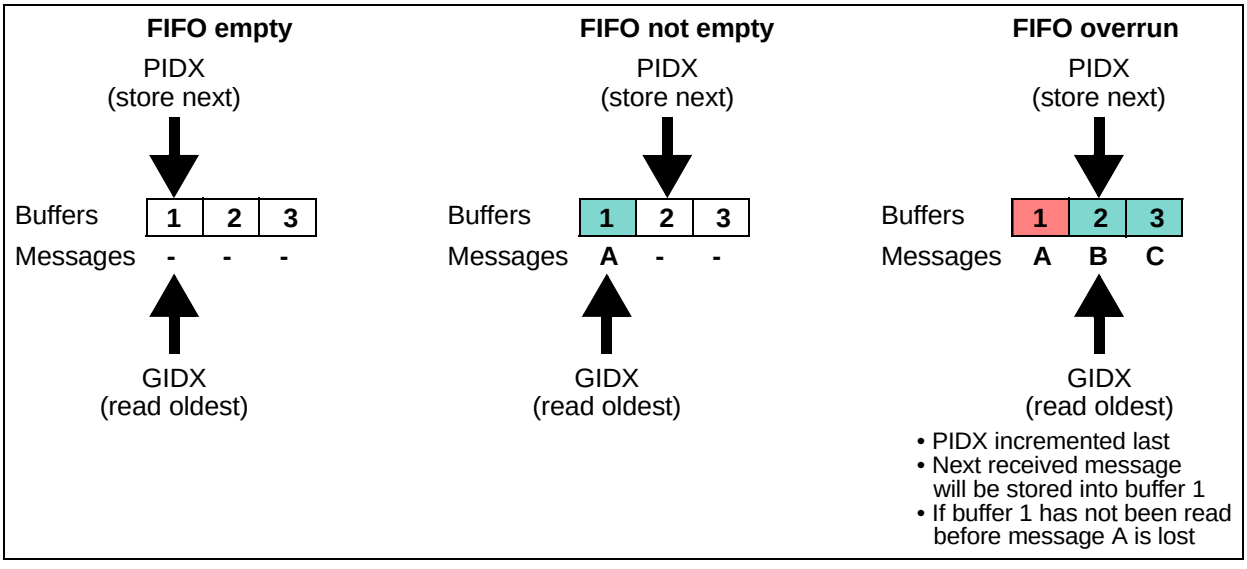


Figure 20-10 FIFO Status: Empty, Not Empty, Overrun

A FIFO non empty status is detected when the PUT index (PIDX) differs from the GET index (GIDX). In this case flag RFNE is set. This indicates that there is at least one received message in the FIFO. The FIFO empty, FIFO not empty, and the FIFO overrun states are explained in **Figure 20-10** for a three Message Buffer FIFO.

There is a programmable FIFO rejection filter for the FIFO. The FIFO Rejection Filter register (FRF) defines a filter pattern for messages to be rejected. The FIFO rejection filter consists of channel filter, Frame ID filter, and cycle counter filter. If bit RSS is set to 1 (default), all messages received in the static segment are rejected by the FIFO. If bit RNF is set to 1 (default), received NULL Frames are not stored in the FIFO.

The FIFO Rejection Filter Mask register (FRFM) specifies which bits of the Frame ID filter in the FIFO Rejection Filter register are marked “don’t care” for rejection filtering.

20.6.10.2 Configuration of the FIFO

For all Message Buffers belonging to the FIFO the data pointer to the first 32-bit word of the Data Section of the respective Message Buffer in the Message RAM has to be configured via the Write Header Section 3 register. All information required for acceptance filtering is taken from the FIFO rejection filter and the FIFO rejection filter mask and needs not be configured in the Header Sections of the Message Buffers belonging to the FIFO.

When programming the data pointers for the Message Buffers belonging to the FIFO, the payload length of all Message Buffers should be programmed to the same value.

*Note: It is recommended to program the MBI bits of the Message Buffers belonging to the FIFO to 0 via **WRHS1.MBI** to avoid generation of RX interrupts.*
*If the payload length of a received Frame is longer than the value programmed by **WRHS2.PLC** in the Header Section of the respective Message Buffer, the data field stored in a Message Buffer of the FIFO is truncated to that length.*

20.6.10.3 Access to the FIFO

To read from the FIFO the Host has to trigger a transfer from the Message RAM to the Output Buffer by writing the number of the first Message Buffer of the FIFO (referenced by FFB) to the Output Buffer Command Request register. The Message Handler then transfers the Message Buffer addressed by the GET Index Register (GIDX) to the Output Buffer. After this transfer the GET Index Register (GIDX) is incremented.

20.6.11 Message Handling

The Message Handler controls data transfers between the Input / Output Buffer and the Message RAM and between the Message RAM and the two Transient Buffer RAMs. All accesses to the internal RAM's are 32+1 bit accesses. The additional bit is used for parity checking.

Access to the Message Buffers stored in the Message RAM is done under control of the Message Handler state machine. This avoids conflicts between accesses of the two protocol controllers and the Host to the Message RAM.

Frame IDs of Message Buffers assigned to the static segment have to be in the range from 1 to NSS as configured in the GTU Configuration Register 7. Frame IDs of Message Buffers assigned to the dynamic segment have to be in the range from NSS + 1 to 2047.

Received messages with no matching dedicated receive buffer (static or dynamic segment) are stored in the receive FIFO (if configured) if they pass the FIFO rejection filter.

20.6.11.1 Host access to Message RAM

The message transfer between Input Buffer and Message RAM as well as between Message RAM and Output Buffer is triggered by the Host by writing the number of the target / source Message Buffer to be accessed to the Input or Output Buffer Command Request register.

The Input / Output Buffer Command Mask registers can be used to write / read Header and Data Section of the selected Message Buffer separately. If bit **STXRS** in the Input Buffer Command Mask register is set (**STXRS** = 1), the transmission request flag TXR of the selected Message Buffer is automatically set after the Message Buffer has been updated.

If bit **STXRS** in the Input Buffer Command Mask register is reset (**STXRS** = 0), the transmission request flag TXR of the selected Message Buffer is reset. This can be used to stop transmission from Message Buffers operated in continuous mode.

Input Buffer (IBF) and the Output Buffer (OBF) are build up as a double buffer structure. One half of this double buffer structure is accessible by the Host (IBF Host / OBF Host), while the other half (IBF Shadow / OBF Shadow) is accessed by the Message Handler for data transfers between IBF / OBF and Message RAM.

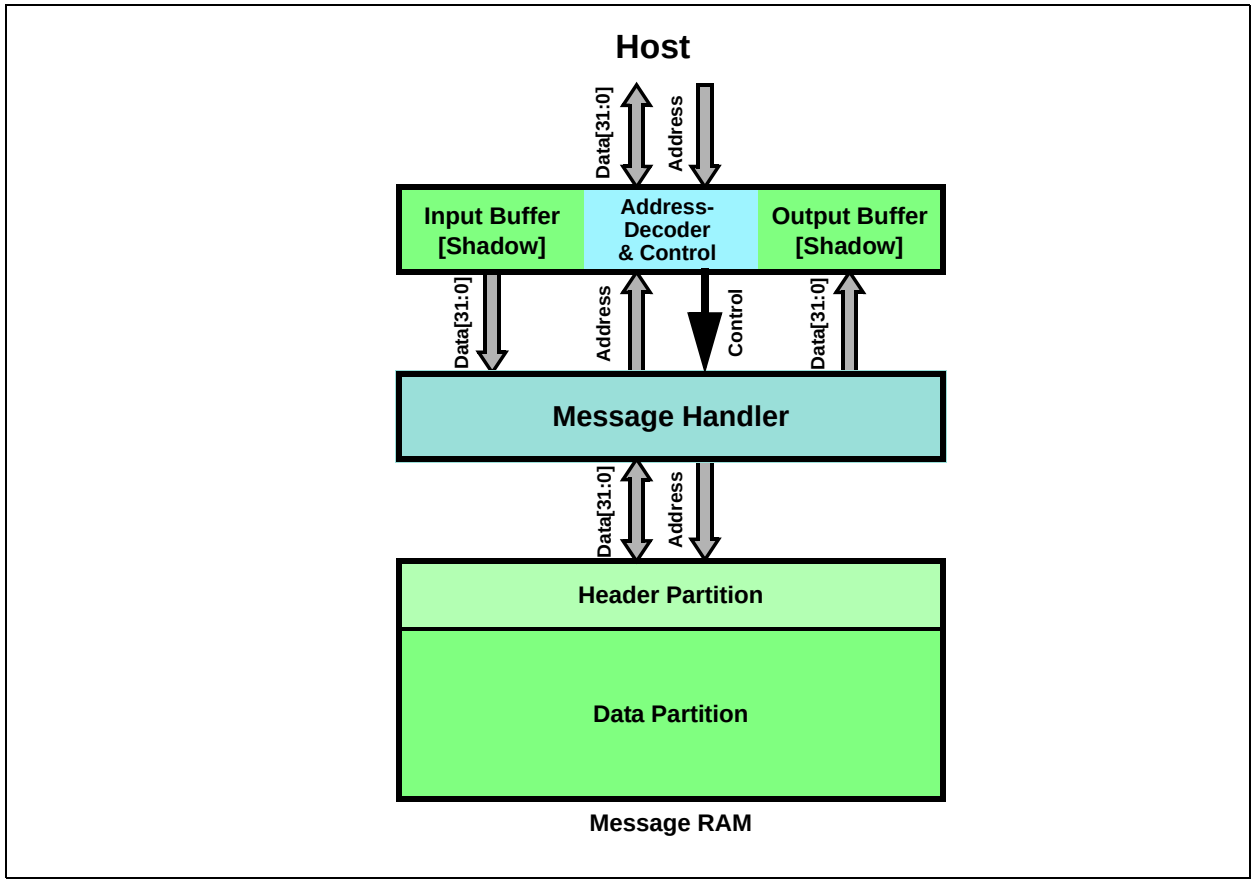


Figure 20-11 Host Access to Message RAM

Data Transfer from Input Buffer to Message RAM

To configure / update a Message Buffer in the Message RAM, the Host has to write the data to **WRDSnn (nn = 01-64)** and the Header to **WRHS1**, **WRHS2**, **WRHS3**. Two sets of **WRDSnn (nn = 01-64)** are available in parallel and selected by **CUST1**.**IBF1PAG**and **CUST1**.**IBF2PAG**. **CUST1**.**IBFS** shows which Input Buffer is currently used as Input Shadow Buffer and which as Input Host Buffer. **WRHS1**, **WRHS2**, and **WRHS2** does only exist once. The specific action is selected by configuring the Input Buffer Command Mask **IBCM**.

When the Host writes the number of the target Message Buffer in the Message RAM to IBRH in the Input Buffer Command Request register **IBCR**, IBF Host and IBF Shadow are swapped (see **Figure 20-12**).

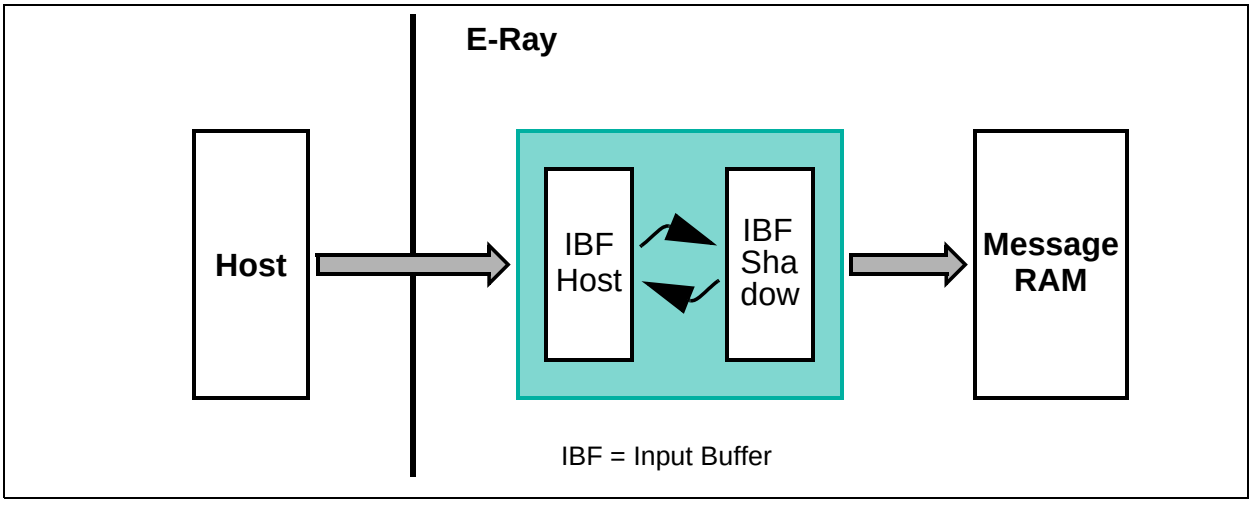


Figure 20-12 Double Buffer Structure Input Buffer

In addition the bits in the Input Buffer Command Mask and Input Buffer Command Request registers are also swapped to keep them attached to the respective IBF section (see **Figure 20-13**).

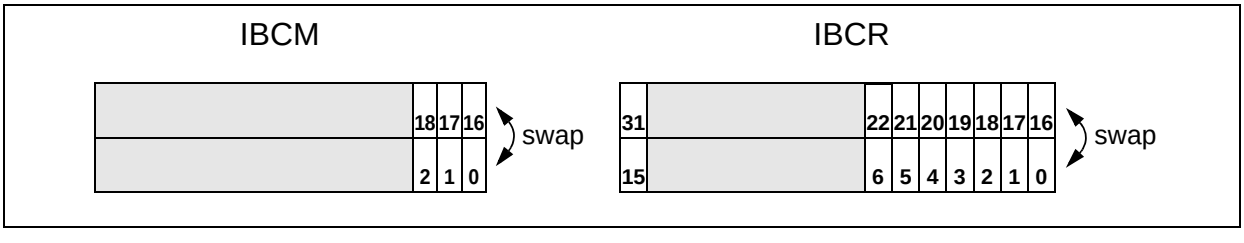


Figure 20-13 Swapping of IBCM and IBCR Bit

With this write operation the IBSYS bit in the Input Buffer Command Request register is set to 1. The Message Handler then starts to transfer the contents of IBF Shadow to the Message Buffer in the Message RAM selected by IBRS.

While the Message Handler transfers the data from IBF Shadow to the target Message Buffer in the Message RAM, the Host may write the next message to IBF Host. After the transfer between IBF Shadow and the Message RAM has completed, the IBSYS bit is set back to 0 and the next transfer to the Message RAM may be started by the Host by writing the respective target Message Buffer number to IBRH in the Input Buffer Command Request register.

If a write access to IBRH occurs while IBSYS is 1, IBSYH is set to 1. After completion of the ongoing data transfer from IBF Shadow to the Message RAM, IBF Host and IBF Shadow are swapped, IBSYH is reset to 0, IBSYS remains set to 1, and the next transfer

to the Message RAM is started. In addition the Message Buffer numbers stored under IBRH and IBRS and the Command Mask flags are also swapped.

Table 20-16 Assignment of Input Buffer Command Mask Bit

Pos.	Access	Bit	Function
18	rh	**STXRS**	Set Transmission Request Shadow
17	rh	**LDSS**	Load Data Section Shadow
16	rh	**LHSS**	Load Header Section Shadow
2	rw	**STXRH**	Set Transmission Request Host
1	rw	**LDSH**	Load Data Section Host
0	rw	**LHSH**	Load Header Section Host

Table 20-17 Assignment of Input Buffer Command Request Bit

Pos.	Access	Bit	Function
31	rh	IBSYS	**IBF Busy Shadow,** signals ongoing transfer from IBF Shadow to Message RAM
21–16	rh	IBRS	**IBF Request Shadow,** number of Message Buffer currently / last updated
15	rh	IBSYH	**IBF Busy Host,** transfer request pending for Message Buffer referenced by IBRH
5-0	rwh	IBRH	**IBF Request Host,** number of Message Buffer to be updated next

Data Transfer from Message RAM to Output Buffer

To read a Message Buffer from the Message RAM, the Host has to write to Command Request register **OBCR** to trigger the data transfer as configured in Output Buffer Command Mask **OBCM** register. After the transfer has completed, the Host can read the transferred data from **RDDSnn (nn = 01-64)**, **RDHS1**, **RDHS2**, **RDHS2**, and **MBS**.

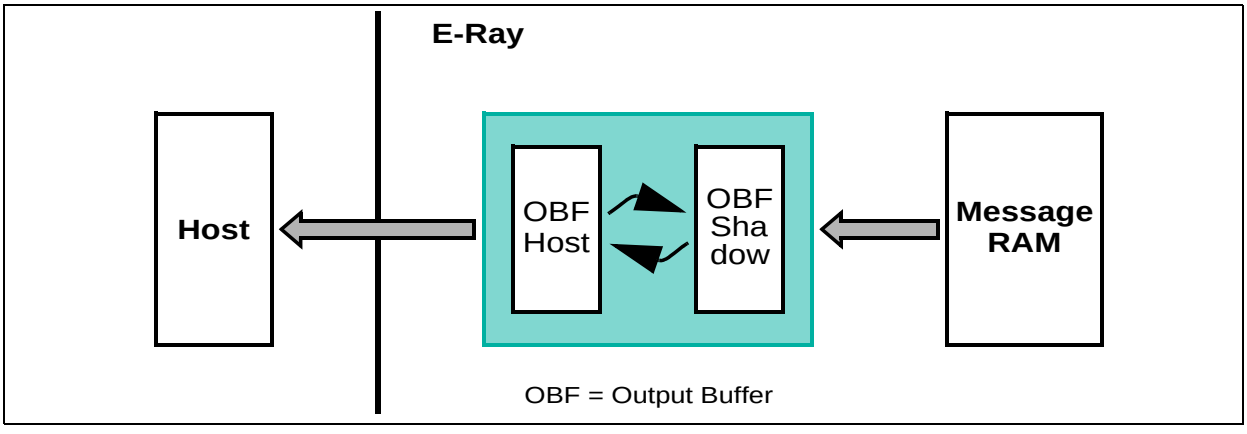


Figure 20-14 Double Buffer Structure Output Buffer

OBF Host and OBF Shadow as well as bits **OBCM.RHSS**, **OBCM.RDSS**, **OBCM.RHSH**, **OBCM.RDSH** and bits **OBCR.OBRS**, **OBCR.OBRH** are swapped under control of bits **OBCR.VIEW** and **OBCR.REQ**.

Writing bit **OBCR.REQ** to 1 copies bits **OBCM.RHSS**, **OBCM.RDSS** and bits **OBCR.OBRS** to an internal storage (see **Figure 20-15**).

After setting **OBCR.REQ** to 1, **OBCR.OBSYS** is set to 1, and the transfer of the Message Buffer selected by **OBCR.OBRS** from the Message RAM to OBF Shadow is started. After the transfer between the Message RAM and OBF Shadow has completed, the **OBCR.OBSYS** bit is set back to 0. Bits **OBCR.REQ** and **OBCR.VIEW** can only be set to 1 while **OBCR.OBSYS** is 0.

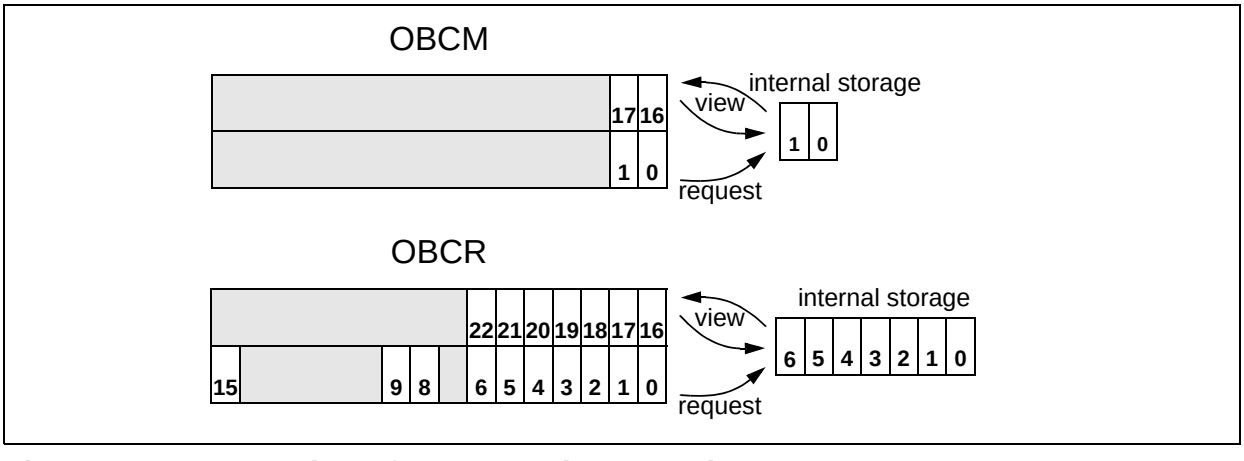


Figure 20-15 Swapping of OBCM and OBCR Bit

OBF Host and OBF Shadow are swapped by setting bit **OBCR.VIEW** to 1 while bit **OBCR.OBSYS** is 0 (see **Figure 20-14**).

In addition bits **OBCR.OBRH** and bits **OBCM.RHSH**, **OBCM.RDSH** are swapped with the registers internal storage thus assuring that the Message Buffer number stored in

OBCR.OBRH and the mask configuration stored in **OBCM.RHSH**, **OBCM.RDSH** matches the transferred data stored in OBF Host (see **Figure 20-15**).

Now the Host can read the transferred Message Buffer from OBF Host while the Message Handler may transfer the next message from the Message RAM to OBF Shadow.

Table 20-18 Assignment of Output Buffer Command Mask Bit

Pos.	Access	Bit	Function
17	rh	RDSH	Data Section available for Host access
16	rh	RHSH	Header Section available for Host access
1	rw	RDSS	Read Data Section Shadow
0	rw	RHSS	Read Header Section Shadow

Table 20-19 Assignment of Output Buffer Command Request Bit

Pos.	Access	Bit	Function
22–16	rh	OBRH	**OBF Request Host,** number of Message Buffer available for Host access
15	rh	OBSYS	**OBF Busy Shadow,** signals ongoing transfer from Message RAM to OBF Shadow
9	rw	REQ	**Request Transfer from Message RAM to OBF Shadow**
8	rwh	VIEW	**View OBF Shadow, swap OBF Shadow, and OBF Host**
6–0	rwh	OBRS	**OBF Request Shadow,** number of Message Buffer for next request

20.6.11.2 Data Transfers between IBF / OBF and Message RAM

This document uses the following terms and abbreviations:

Table 20-20 Terms and Abbreviations

Term	Meaning
MHD	Message Handler
IBF	Input Buffer 1 or 2 RAM
OBF	Output Buffer 1 or 2 RAM
MBF	Message Buffer RAM
TBF	Transient Buffer RAM Channel A (TBF1) or Channel B (TBF2)

Table 20-20 Terms and Abbreviations (cont'd)

IBF ⇒ MBF	Transfer from IBF to MBF
MBF ⇒ OBF	Transfer from MBF to OBF
MBF ⇒ TBF	Transfer from MBF to TBF
TBF ⇒ MBF	Transfer from TBF to MBF
SS	Slot Status
SS ⇒ MBF	Transfer SS to MBF

Message Handler functionality

The MHD controls the access to the MBF. It manages data-transfer between MBF and IBF, OBF, TBF1, TBF2. The data-path are shown in Figure 20-16.

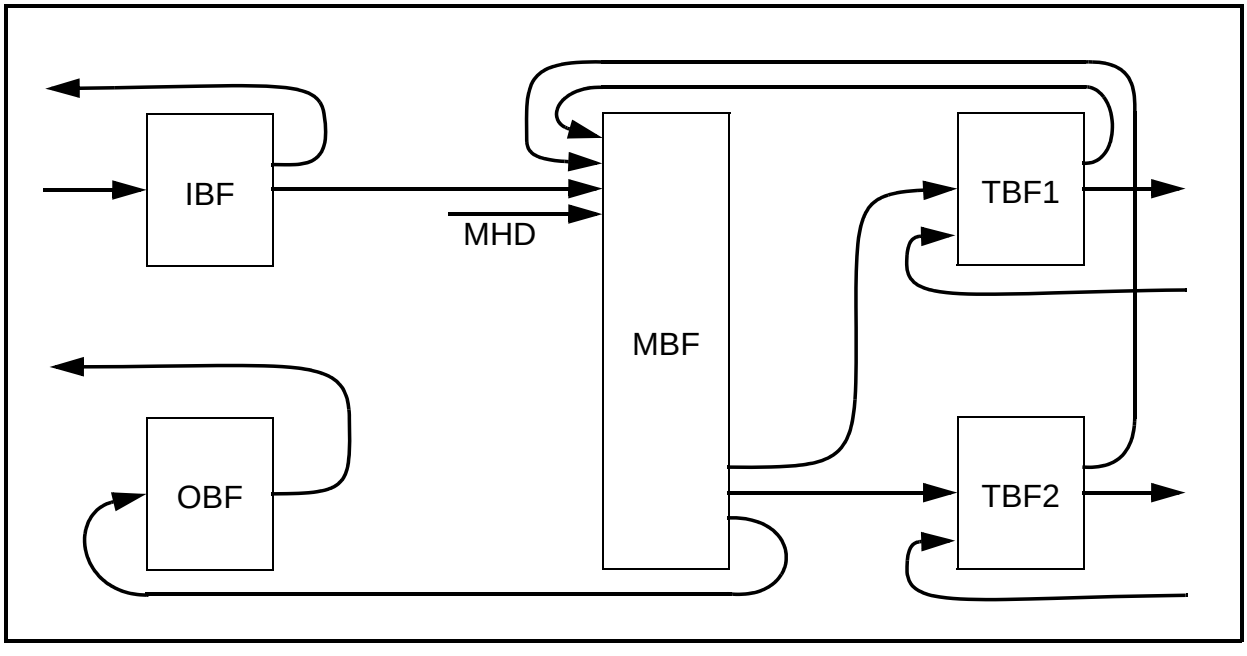


Figure 20-16 Interconnection of RAMs

Furthermore a search-algorithm allows to find the next valid message object in the MBF for transmission or reception.

Each transfer consists of a setup-time, four time steps to transfer the Header-section and a payload-length-dependent number of time steps to transfer the data-section. The internal data-busses have a width of 32 bits. Thereby it is possible to transfer two 2-byte words in one time step. If the payload consists of an odd number of 2-byte words the last time step of the data-section contains only 16 bit of valid data. If the Payload-Length (PL) is e.g. 7, the data-section consists of 4 time steps.

The maximum length for the data-section is 64 time steps, the minimum length is zero time steps.

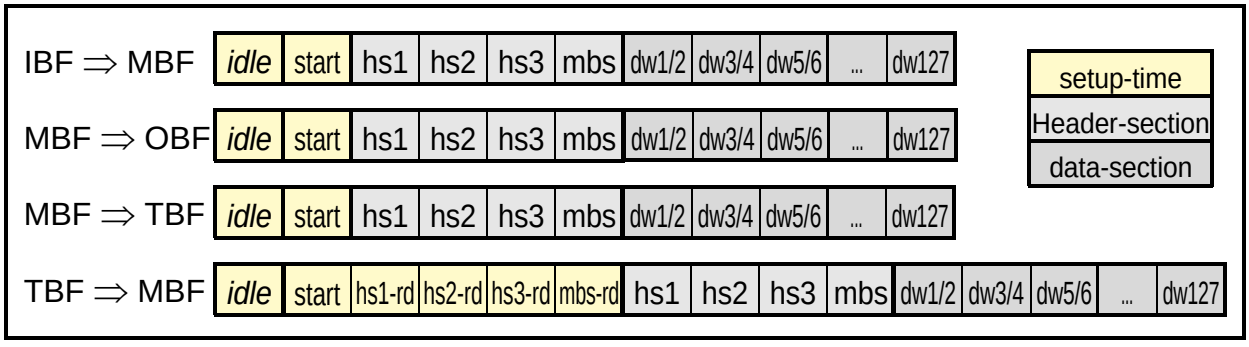


Figure 20-17 Different Possible Buffer Transfers

The update of the Slot-Status consists of a setup-time and one time-step to write the new Slot-Status.

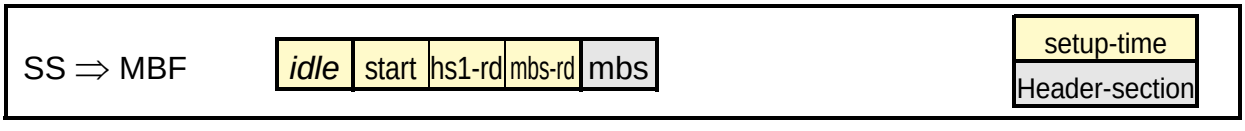


Figure 20-18 Update of Slot Status

The length of a time step depends on the number of concurrent tasks.

The following concurrent tasks are executed under control of the Message Handler:

- Data transfer between IBF or OBF and MBF
- Data transfer between TBF1 and MBF, search next TX / RX Message Buffer CHA
- Data transfer between TBF2 and MBF, search next TX / RX Message Buffer CHB

Thereby the time step length can vary between one and three f_{CLC_ERAY}periods.

Under certain conditions it is possible that a transfer is stopped or interrupted for a number of time steps until it is continued.

When a IB F⇒ MBF is started short after a TBF ⇒ MBF or SS ⇒ MBF the transfer from IBF has to wait until the setup-time of the internal transfer has finished (see Figure 20-19)

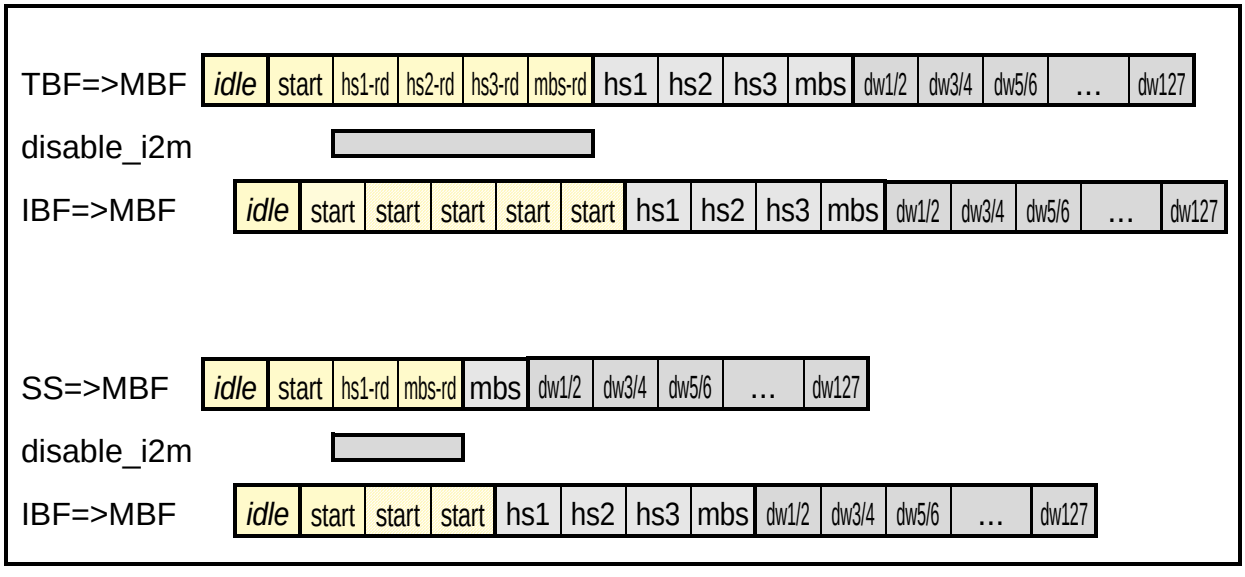


Figure 20-19 Delay start of IBF⇒MBF

The internal signal “disable_i2m” is always active when the TBF ⇒ MBF is in state “hs1-rd”, “hs2-rd”, “hs3-rd” or “mbs-rd” and when the SS ⇒ MBF is in state “hs1-rd” or “mbs-rd”.

The IBF ⇒ MBF is hold in state “start” until the internal signal “disable_i2m” gets inactive.

These additional time-steps are independent of any address-counter-values. This means, the IBF ⇒ MBF has to wait even if it writes to another buffer than the internal transfer.

Multiple requests of transfers between IBF/OBF and Message RAM

The time required to transfer the contents of a Message Buffer between IBF / OBF and Message RAM depends on the number of 4-byte words to be transferred, the number of concurrent tasks to be managed by the Message Handler, and in special cases the type and address range of the internal transfer. The number of 4-byte words varies from 4 (Header Section only) to 68 (Header + maximum Data Section) plus a short setup time to start the first transfer, while the number of concurrent task varies from one to three. The 4 Header words have to be included in calculation even if only the Data Section is requested for transfer.

The following concurrent tasks are executed under control of the Message Handler:

- Data transfer between IBF or OBF and MBF
- Data transfer between TBF1 and MBF, search next TX / RX Message Buffer CHA
- Data transfer between TBF2 and MBF, search next TX / RX Message Buffer CHB

Transfers between IBF and MBF respectively MBF and OBF can only be handled one after another. In case that e.g. a IBF ⇒ MBF has been started shortly before a

MBF ⇒ OBF is requested, the MBF ⇒ OBF has to wait until the IBF ⇒ MBF has completed.

In case that e.g. a second IBF⇒MBF is requested, a MBF⇒OBF is requested and a IBF⇒MBF is ongoing, the MBF⇒OBF has to wait until the first IBF⇒IBF has completed. The second IBF=MBF has to wait until the MBF⇒OBF has completed (see figure 20-20) independent whether MBF⇒OBF or second IBF⇒MBF is requested first.

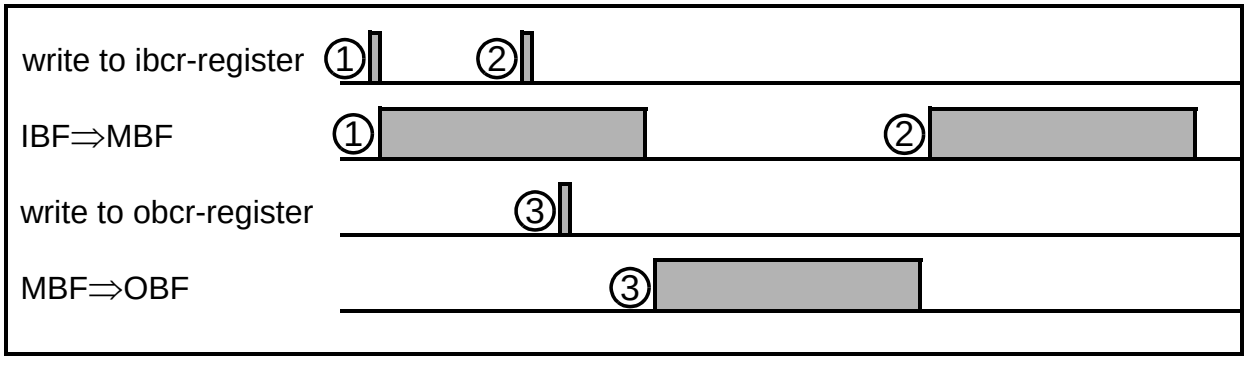


Figure 20-20 Multiple IBF/OBF Request

Worst case for single request

When a message with a large payload length is received the TBF⇒MBF is started at the begin of the next slot (n+1). If the next slot is a dynamic slot without transmission/reception (minislot), it may happen that the TBF⇒MBF has not finished until begin of the next but one slot (n+2). In this case the TBF⇒MBF will be service requested (break) to start a transmission in the next but one slot (MBF⇒TBF) and/or to update the slot status (SS⇒MBF) for the RX-buffer corresponding with next slot (n+1). After this interruption the TBF⇒MBF is continued.

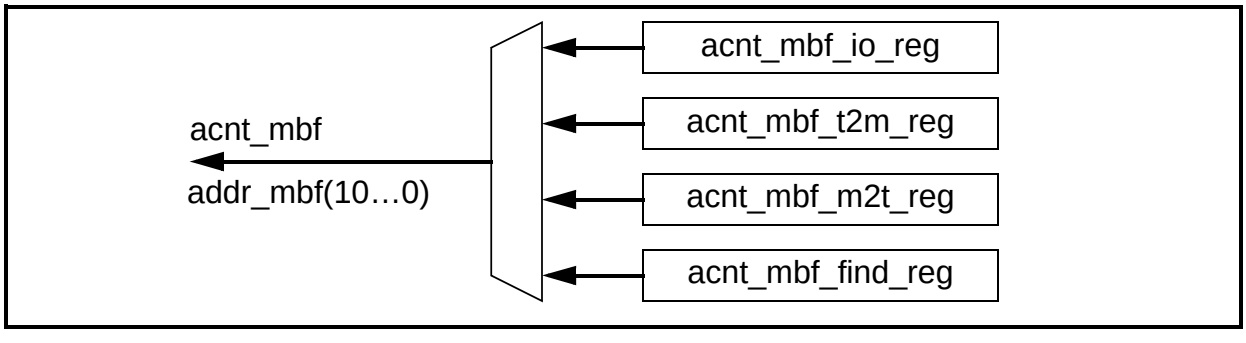


Figure 20-21 Address Counter Scheme of Message RAM (simplified)

For the transfers IBF⇒MBF / MBF⇒OBF, TBF⇒MBF and MBF⇒TBF separate address-counter are implemented (see Figure 20-21).

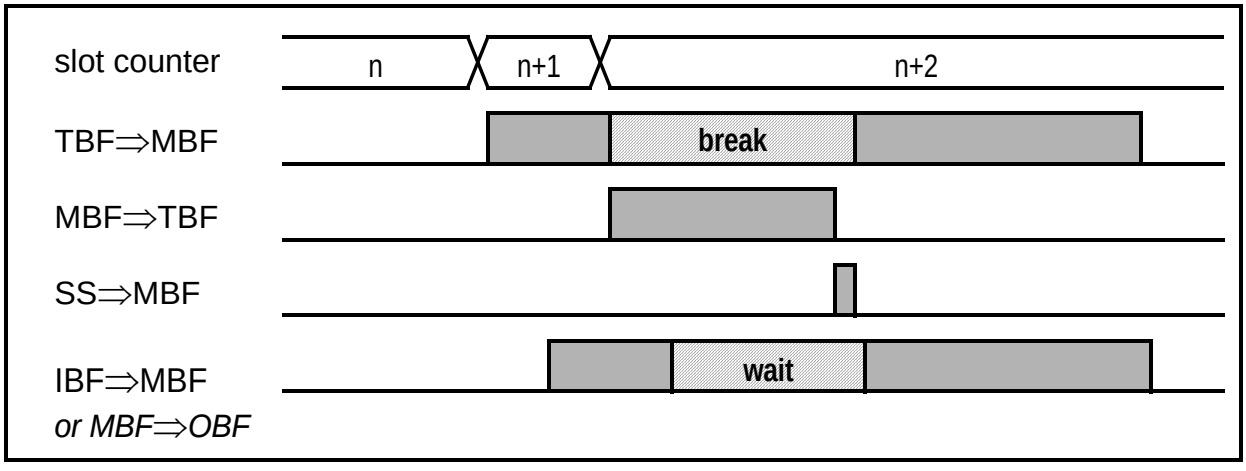


Figure 20-22 interruption of TBF⇒MBF

If the address-counter for IBF⇒MBF / MBF⇒OBF (acnt_mbf_io_reg) reaches the address of the interrupted TBF=>MBF (acnt_mbf_t2m_reg) the IBF⇒MBF / MBF⇒OBF has to wait until the TBF⇒MBF is continued (see Figure 20-22).

The relative time is measured in f_{CLC_ERAY} cycles. Absolute time depends on the actual f_{CLC_ERAY} cycle period.

$$\text{tbf\_to\_mbf\_break time}_{max} = (\text{setup time} + \text{mbf\_to\_tbf time}_{max}) + (\text{setup time} + \text{ss\_to\_mbf})$$

$$\text{cycles}_{req} = (\text{number of concurrent tasks}) \times ((\text{setup time} + (\text{number of 4-byte words})_{req}) + \text{tbf\_to\_mbf\_break time})$$

$$\text{setup time} = 2\ f_{CLC\_ERAY}\ \text{cycles}$$

Worst case for one IBF⇒MBF or MBF⇒OBF:

Max. break time: tbf_to_mbf_break time$_{max}$ = (2+68) + (4+1) = 75

Max. number of f_{CLC_ERAY} cycles: cycles$_{req}$ = 3 × (6 + 68 + 75) = 435

Worst case for multiple transfers

If a second IBF⇒MBF and a MBF⇒OBF (see Figure 20-20) is requested directly after the first IBF⇒MBF has started following worst case timing could appear:

cycles$_{trans}$ = (remaining cycles of transfer running)
+ (cycles of second requested transfer)
+ (cycles of third requested transfer)

cycles$_{trans}$ = cycles$_{rem}$ + cycles$_{req_2}$+ cycles$_{req_3}$

Max. number of f_{CLC_ERAY} cycles: cycles$_{trans}$ = 447 + 435 + 447 = 1329

20.6.11.3 Minimum f_{CLC_ERAY}

To calculate the minimum f_{CLC_ERAY} the worst case scenario has to be considered.

The worst case scenario depends on the following parameters

- maximum payload length
- minimum minislot length
- number of configured Message Buffers (excluding FIFO)
- used channels (single/dual channel)

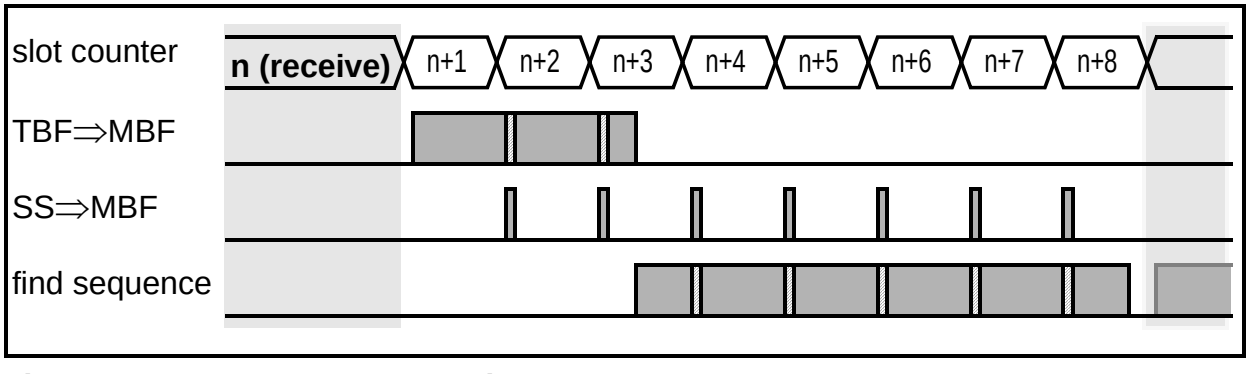


Figure 20-23 worst case scenario

Worst case scenario:

- reception of message with a maximum payload length in Slot n (n is 7,15,23,31,39,…)
- slot n+1 to n+7 are empty dynamic slots (minislot) and configured as receive buffer
- the find-sequence (usually started in slot 8,16,24,32,40,…) has to scan the maximum number of configured buffers
- the number of concurrent tasks has its maximum value of three

The find-sequence is executed each 8 Slots (slot 8,16,24,32,40,…). It has to be finished until the next find-sequence is requested.

The length of a TBF⇒MBF varies from 4 (Header Section only) to 68 (Header + maximum Data Section) time step plus a setup time of 6 time steps.

$$cycles_{t2m} = \frac{f_{CLC\_ERAY}}{\text{number of concurrent tasks}) \times (\text{setup time}_{t2m} + (\text{number of 4-byte words})_{t2m})}$$

A SS⇒MBF has a fixed length of 1 time steps plus a setup time of 4 time steps.

$$cycles_{ss2m} = \frac{f_{CLC\_ERAY}}{(\text{number of concurrent tasks}) \times 5}$$

The find sequence has a maximum length of 128 (maximum number of buffers) time steps plus a setup time of 2 time steps.

$$cycles_{find} = \frac{f_{CLC\_ERAY}}{(\text{number of concurrent tasks}) \times (\text{setup time}_{find} + (\text{number of configured buffers}))}$$

A minislot has a length of 2 to 63 Macrotick (gdMinislot). The minimum nominal Macrotick period (cdMinMTNom) is 1 µs. A sequence of 8 minislots has a length of

$$time_{8minislots} = 8 \times gdMinislot \times cdMinMTNom$$

The maximum period $T_{CLC\_ERAY} = 1/f_{CLC\_ERAY}$ can be calculated as followed:

$$time_{8minislots} \geq (f_{CLC\_ERAY}\ \text{period in}\ \mu s) \times (\ (f_{CLC\_ERAY}\ cycles_{t2m}) + 7 \times (f_{CLC\_ERAY}\ cycles_{ss2m}) + (f_{CLC\_ERAY}\ cycles_{find}))$$

$$f_{CLC\_ERAY}\ \text{period in ms} \leq \frac{time_{8mnislots}}{(\ cycles_{t2m}) + 7 \times (cycles_{ss2m}) + (cycles_{find})}$$

$$\text{minimum } time_{8minislots} = 8 \times 2 \times 1\ \mu s = 16\ \mu s$$

$$\text{maximum } f_{CLC\_ERAY}\ cycles_{t2m} = 3 \times (6 + 68) = 222$$

$$\text{maximum } f_{CLC\_ERAY}\ cycles_{ss2m} = 3 * 5 = 15$$

$$\text{maximum } f_{CLC\_ERAY}\ cycles_{find} = 3 * (2 + 128) = 390$$

$$f_{CLC\_ERAY}\ \text{period in ms} \leq \frac{16\mu s}{222 + 7 \times 15 + 390} = 22.315\ldots ns$$

The minimum f_{CLC_ERAY} frequency for this worst case scenario is 44.8125 MHz.

A too low f_{CLC_ERAY} frequency can cause a malfunction of the E-Ray.

The E-Ray can detect several malfunctions and reports this by setting the corresponding flag in the Message Handler Contraints Flags (**MHDF**) register.

Minimum f_{CLC_ERAY} for various maximum payload length

Table 20-21 summarizes the minimum required f_{CLC_ERAY} frequency for various maximum payload length assuming:

- a minimum minislot length of 2µs.
- a maximum of 128 configured Message Buffers.
- dual channels in use.

Table 20-21 Minimum f_{CLC_ERAY} for different maximum payload length

Maximum payload length of 32 bit words	4	8	16	32	64
minimum f_{CLC_ERAY}	32,82 MHz	33,57 MHz	35,07 MHz	38,07 MHz	44,82 MHz

Minimum f_{CLC_ERAY} for various minimum minislot length

Table 20-22 summarizes the minimum required f_{CLC_ERAY} frequency for various minimum minislot length assuming:

- a maximum payload length of 254 bytes / 64 four-byte-words.
- a maximum 128 configured Message Buffers.
- dual channels in use.

Table 20-22 Minimum f_{CLC_ERAY} for different minimum minislot length

gdMinislot at dMinMTNom = 1 µs	2 µs	3 µs	4 µs	7 µs	8 µs
minimum f_{CLC_ERAY}	44,82 MHz	29,88 MHz	22,412 MHz	12,8 MHz	9,96 MHz

Minimum f_{CLC_ERAY} for various amount of configured Message Buffers

Table 20-23 summarizes the minimum required f_{CLC_ERAY} frequency for various amount of configured Message Buffers assuming:

- a maximum payload length of 254 bytes / 64 four-byte-words.
- a minimum minislot length of 2 µs.
- dual channels in use.

Table 20-23 Minimum f_{CLC_ERAY} for different amount of configured Message Buffers

Configured maximum amount of Message Buffers	128	64	32
minimum f_{CLC_ERAY}	44,82 MHz	32,82 MHz	26,82 MHz

Minimum f_{CLC_ERAY} for a typical configuration

´The minimum required f_{CLC_ERAY} frequency for various assuming the following typical E-Ray configuration:

- a maximum payload length of 32 bytes / 8 four-byte-words.
- a minimum minislot length of 7 µs.
- a maximum 128 configured Message Buffers.
- dual channels in use

The minimum f_{CLC_ERAY} frequency for this typical example would be 10 MHz.

20.6.11.4 FlexRay™ Protocol Controller access to Message RAM

The two Transient Buffer RAMs (TBF 1, TBF 2) are used to buffer the data for transfer between the two FlexRay™ Protocol Controllers and the Message RAM.

Each Transient Buffer RAM is build up as a double buffer, able to store two complete FlexRay™ messages. There is always one buffer assigned to the corresponding Protocol Controller while the other one is accessible by the Message Handler.

If e.g. the Message Handler writes the next message to be send to Transient Buffer Tx, the FlexRay™ Channel Protocol Controller can access Transient Buffer Rx to store the message it is actually receiving. During transmission of the message stored in Transient Buffer Tx, the Message Handler transfers the last received message stored in Transient Buffer Rx to the Message RAM (if it passes acceptance filtering) and updates the respective Message Buffer.

Data transfers between the Transient Buffer RAMs and the shift registers of the FlexRay™ Channel Protocol Controllers are done in words of 32 bit. This enables the use of a 32 bit shift register independent of the length of the FlexRay™ messages.

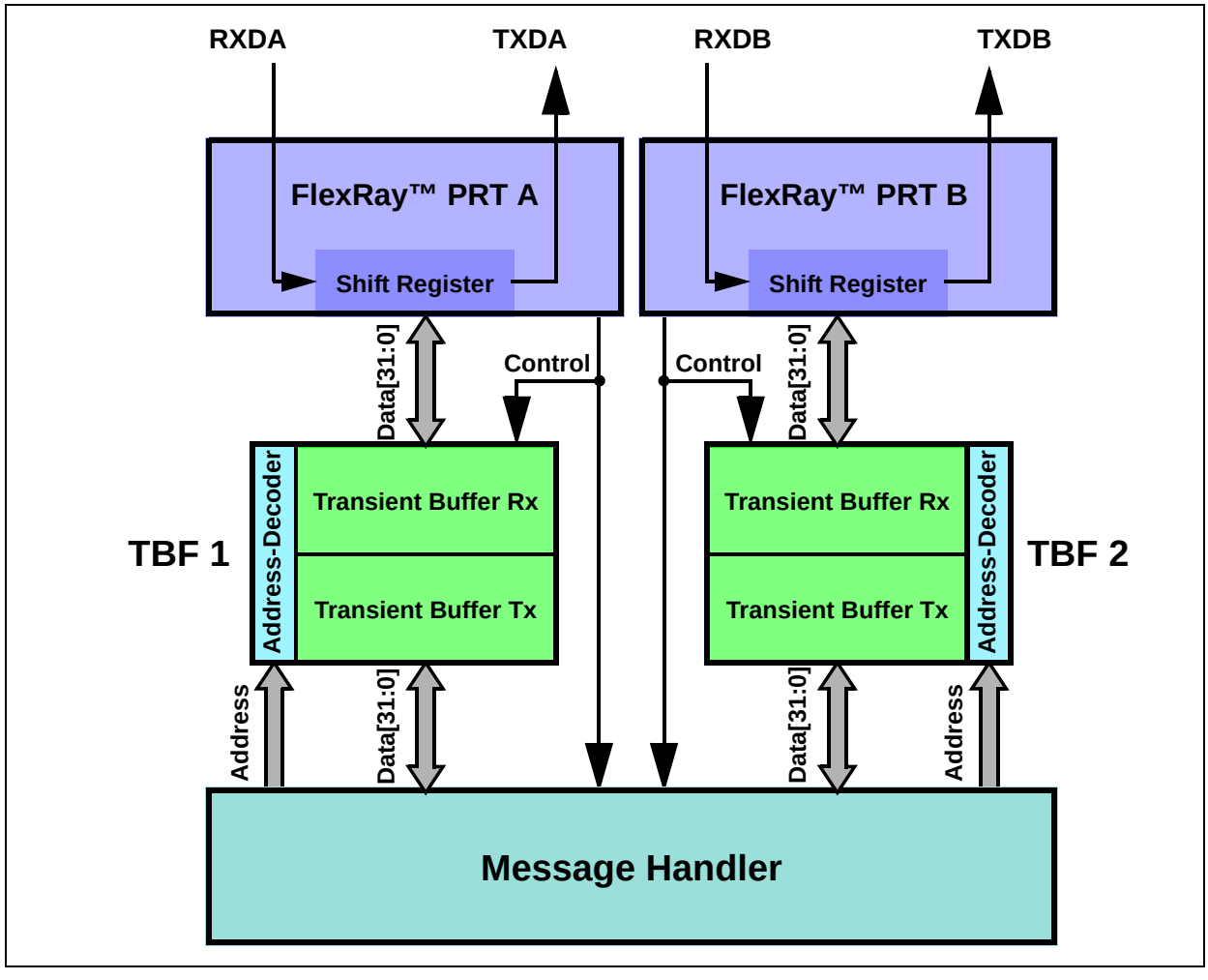


Figure 20-24 Access to Transient Buffer RAMs

20.6.12 Message RAM

To avoid conflicts between Host access to the Message RAM and FlexRay™ message reception / transmission, the Host cannot directly access the Message Buffers in the Message RAM. These accesses are handled via the Input and Output Buffers. The Message RAM is able to store up to 128 Message Buffers depending on the configured payload length.

The Message RAM is organized 2048 x 33. Each 32-bit word is protected by a parity bit. To achieve the required flexibility with respect to different numbers of data byte per FlexRay™ Frame (0 to 254), the Message RAM has a structure as shown in **Figure 20-25**.

The Data Partition is allowed to start at Message RAM word number: (MRC.LCB + 1) • 4

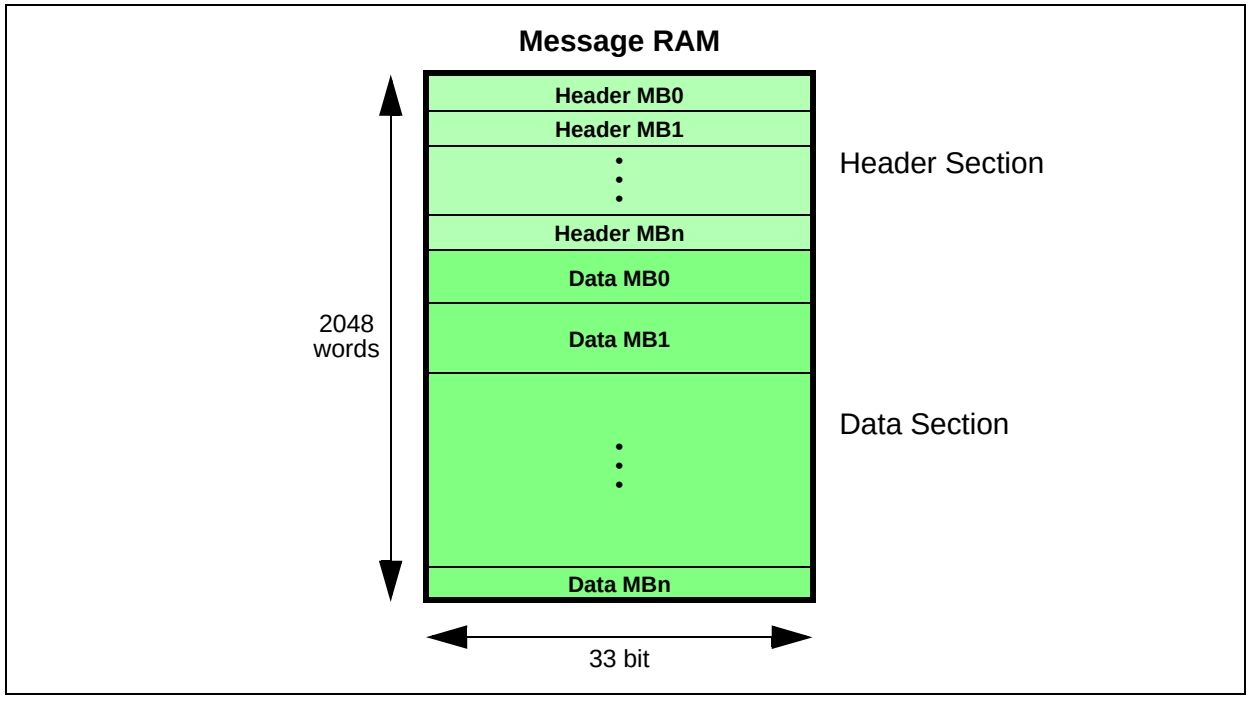


Figure 20-25 Structure of Message RAM

Header Partition

Stores Header Segments of FlexRay™ Frames:

- Supports a maximum of 128 Message Buffers
- Each Message Buffer has a Header of four 32+1 bit words
- Header 3 of each Message Buffer holds the11 bit pointer to the respective Data Section in the Data Partition

Data Partition

Flexible storage of Data Sections with different length. Some maximum values are:

- 30 Message Buffers with 254 byte Data Section each
- Or 56 Message Buffers with 128 byte Data Section each
- Or 128 Message Buffers with 48 byte Data Section each

Restriction: Header Partition + Data Partition may not occupy more than 2048 33-bit words.

20.6.12.1 Header Partition

The Header of each Message Buffer occupies four 33-bit words in the Header Partition of the Message RAM. The Header of Message Buffer 0 starts with the first word in the Message RAM.

For transmit buffers the Header CRC has to be calculated by the Host.

Payload Length Received **PLR**, Receive Cycle Count **RCC**, Received on Channel Indication **RCI**, Startup Frame Indication bit **SFI**, Sync bit **SYN**, NULL Frame Indication bit **NFI**, Payload Preamble Indication bit **PPI**, and Reserved bit **RES** are only updated from received valid Frames (including valid NULL Frames).

Header word 4 of each configured Message Buffer holds the respective Message Buffer Status **MBS** information.

Table 20-24 Header Section of a Message Buffer in the Message RAM

Bit Word	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	P			MBI	TXM	NME	CFG	CHB	CHA		Cycle Code												Frame ID										
2	P		Payload Length Received								Payload Length Configured												Tx Buffer: Header CRC Configured Rx Buffer: Header CRC Received										
3	P			RES	PPI	NFI	SYN	SFI	RCI			Receive Cycle Count											Data Pointer										
4	P			RESS	PPIS	NFIS	SYNS	SFIS	RCIS			Cycle Count Status						TFB	FTY		MLST	ESB	ESA	TCIB	TCIA	SVOB	SVOA	CEOB	CEOA	SEOB	SEOA	VFRB	VFRA

- Frame Configuration
- Filter Configuration
- Message Buffer Control
- Message RAM Configuration
- Updated from received Frame
- Message Buffer Status
- Parity Bit
- unused

Header 1 (word 0)

Write access via **WRHS1**, read access via **RDHS1**:

- Frame ID: Slot counter filtering configuration
- Cycle Code: Cycle counter filtering configuration
- CHA, CHB: Channel filtering configuration
- CFG: Message Buffer configuration: receive / transmit
- PPIT: Payload Preamble Indicator Transmit
- XMI: Transmit mode configuration: single-shot / continuous
- MBI: Message Buffer receive / transmit service request enable

Header 2 (word 1)

Write access via **WRHS2**, read access via **RDHS2**:

- Header CRC
  - Transmit Buffer: Configured by the Host (calculated from Frame Header Segment)
  - Receive Buffer: Updated from received Frame
- Payload Length Configured
  - Length of Data Section (2-byte words) as configured by the Host
- Payload Length Received
  - Length of Payload Segment (2-byte words) stored from received Frame

Header 3

Write access via **WRHS3**, read access via **RDHS3**:

- Data Pointer
  - Pointer to the beginning of the corresponding Data Section in the Data Partition

Read access via **RDHS3**, valid for receive buffers only, updated from received Frames:

- Receive Cycle Count: Cycle count from received Frame
- RCI: Received on Channel Indicator
- SFI: Startup Frame Indicator
- SYN: SYNC Frame Indicator
- NFI: NULL Frame Indicator
- PPI: Payload Preamble Indicator
- RES: Reserved bit

Message Buffer Status MBS (word 3)

Read access via MBS, updated by the Communication Controller at the end of the configured slot.

- VFRA: Valid Frame Received on channel A
- VFRB: Valid Frame Received on channel B
- SEOA: Syntax Error Observed on channel A
- SEOB: Syntax Error Observed on channel B
- CEOA: Content Error Observed on channel A
- CEOB: Content Error Observed on channel B
- SVOA: Slot boundary Violation Observed on channel A
- SVOB: Slot boundary Violation Observed on channel B
- TCIA: Transmission Conflict Indication channel A
- TCIB: Transmission Conflict Indication channel B
- ESA: Empty Slot Channel A
- ESB: Empty Slot Channel B
- MLST: Message LoST
- FTA: Frame Transmitted on Channel A
- FTA: Frame Transmitted on Channel B
- Cycle Count Status: Actual cycle count when status was updated
- RCIS: Received on CHannel Indicator Status
- SFIS: Startup Frame Indicator Status
- SYNS: SYNC Frame Indicator Status
- NFIS: NULL Frame Indicator Status
- PPIS: Payload Preamble Indicator Status
- RESS: Reserved Bit Status

20.6.12.2 Data Partition

The Data Partition of the Message RAM stores the Data Sections of the Message Buffers configured for reception / transmission as defined in the Header Partition. The number of data bytes for each Message Buffer can vary from 0 to 254. To optimize the data transfer between the shift registers of the two FlexRay™ Protocol Controllers and the Message RAM as well as between the Host interface and the Message RAM, the physical width of the Message RAM is set to 4 bytes plus one parity bit.

The Data Partition starts after the last word of the Header Partition. When configuring the Message Buffers in the Message RAM the programmer has to assure that the data pointers point to addresses within the Data Partition. **Table 20-25** below shows an example how the Data Sections of the configured Message Buffers can be stored in the Data Partition of the Message RAM.

The beginning and the end of a Message Buffer's Data Section is determined by the data pointer and the payload length configured in the Message Buffer's Header Section,

respectively. This enables a flexible usage of the available RAM space for storage of Message Buffers with different data length.

If the size of the Data Section is an odd number of 2-byte words, the remaining 16 bits in the last 32-bit word are unused (see **Table 20-25** below)

Table 20-25 Example for Structure of the Data Section in the Message RAM

Bit Word	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
…	P	unused								unused								unused								unused							
…	P	unused								unused								unused								unused							
…	P	**MB1** Data3								**MB1** Data2								**MB1** Data1								**MB1** Data0							
…	P	…								…								…								…							
…	P	…								…								…								…							
…	P	**MB1** Data(n)								**MB1** Data(n-1)								**MB1** Data(n-2)								**MB1** Data(n-3)							
…	P	…								…								…								…							
…	P	…								…								…								…							
…	P	…								…								…								…							
…	P	**MB1** Data3								**MB1** Data2								**MB1** Data1								**MB1** Data0							
…	P	…								…								…								…							
…	P	**MB1** Data(k)0								**MB1** Data(k-1)0								**MB1** Data(k-2)0								**MB1** Data(k-3)0							
2046	P	**MB80** Data3								**MB80** Data2								**MB80** Data2								**MB80** Data0							
2047	P	unused								unused								**MB80** Data5								**MB80** Data4							

20.6.12.3 Parity Check

There is a parity checking mechanism implemented in the E-Ray module to assure the integrity of the data stored in the seven RAM blocks of the module. The RAM blocks have a parity generator / checker attached as shown in **Figure 20-26**. When data is written to a RAM block, the local parity generator generates the parity bit. The E-Ray module uses an even parity (with an even number of one's in the 32-bit data word a zero parity bit is generated). The parity bit is stored together with the respective data word. The parity is checked each time a data word is read from any of the RAM blocks. The module internal data buses have a width of 32 bits.

If a parity error is detected, the respective error flag is set. The parity error flags **MHDS.PIBF**, **MHDS.POBF**, **MHDS.PMR**, **MHDS.PTBF1**, **MHDS.PTBF2**, and the faulty Message Buffer indicators **MHDS.FMBD**, **MHDS.MFMB**, **MHDS.FMB** are located in the

Message Handler Status register. These single error flags control the error interrupt flag **EIR.PERR**.

Figure 20-26 shows the data paths between the RAM blocks and the parity generators / checkers.

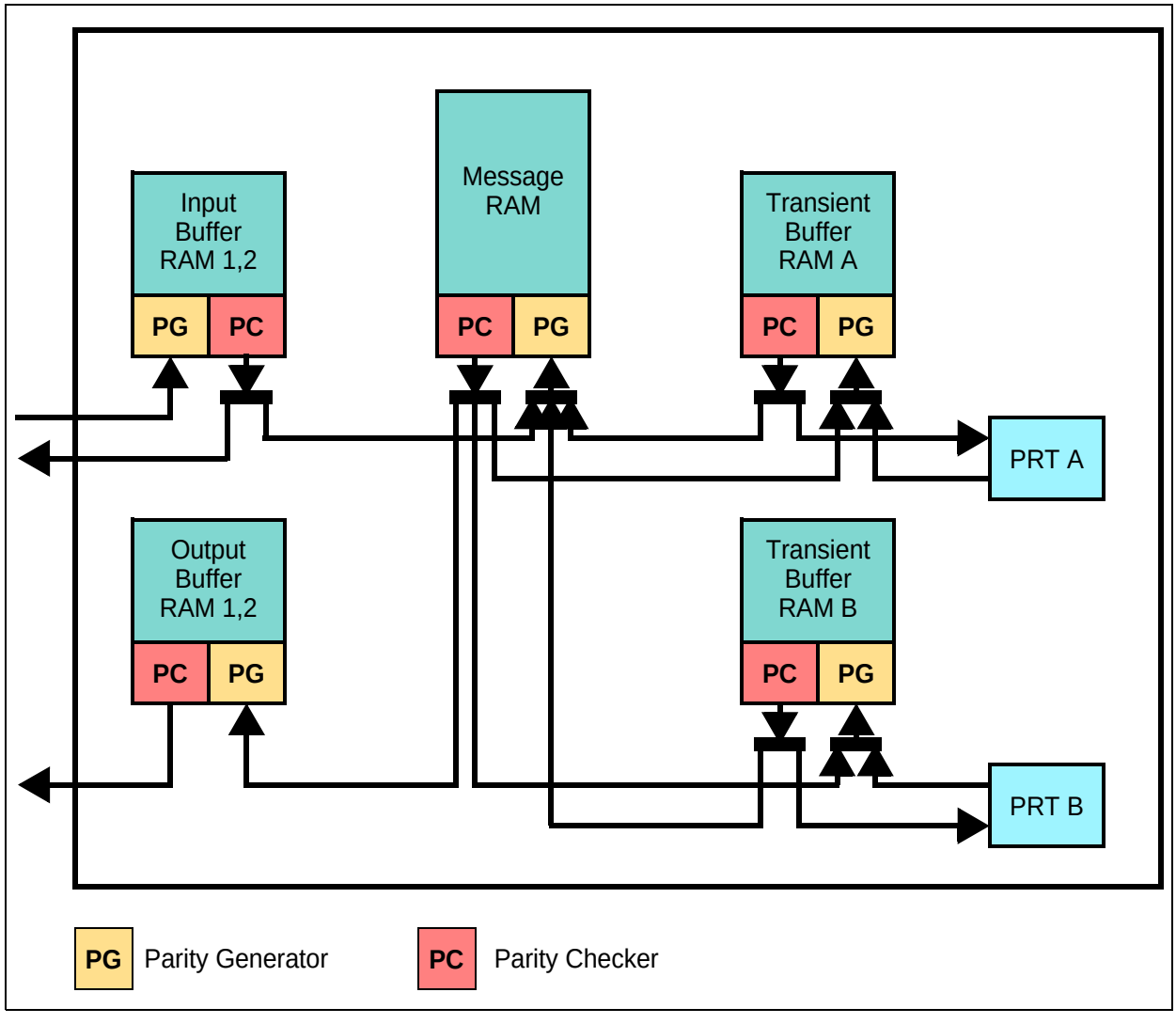


Figure 20-26 Parity Generation and Check

Note: Parity generator & checker are not part of the RAM blocks, but of the RAM access hardware which is part of the E-Ray core.

When a parity error has been detected the following actions will be performed:

In all cases

- The respective parity error flag in the Message Handler Status **MHDS** register is set
- The parity error flag **EIR.PERR** in the Error Service Request Register is set, and if enabled, a module service request to the Host will be generated.

Additionally in specific cases

1. Parity error in data transfer from Input Buffer RAM 1,2 ⇒ Message RAM (Transfer of Header and Data Section)
   a) **MHDS.PIBF** bit is set
   b) **MHDS.FMBD** bit is set to indicate that **MHDS.FMB** has been updated
   c) **MHDS.FMB** indicates the number of the faulty Message Buffer
   d) Transmit buffer: Transmission request for the respective Message Buffer is not set
2. Parity error in data transfer from Input Buffer RAM 1,2 ⇒ Message RAM (Transfer of Data Section only)
   a) **MHDS.PMR** bit is set
   b) **MHDS.FMBD** bit is set to indicate that **MHDS.FMB** points to a faulty Message Buffer
   c) **MHDS.FMB** indicates the number of the faulty Message Buffer
   d) The Data Section of the respective Message Buffer is not updated
   e) Transmit buffer: Transmission request for the respective Message Buffer is not set
3. Parity error during host reading Input Buffer RAM
   a) • **MHDS.PIBF** bit is set
4. Parity error during scan of Header Sections in Message RAM
   a) **MHDS.PMR** bit is set
   b) **MHDS.FMBD** bit is set to indicate that **MHDS.FMB** points to a faulty Message Buffer
   c) **MHDS.FMB** indicates the number of the faulty Message Buffer
   d) Ignore Message Buffer (Message Buffer is skipped)
5. Parity error during data transfer from Message RAM ⇒ Transient Buffer RAM A, B
   a) **MHDS.PMR** bit is set
   b) **MHDS.FMBD** bit is set to indicate that **MHDS.FMB** points to a faulty Message Buffer
   c) **MHDS.FMB** indicates the number of the faulty Message Buffer
   d) Frame not transmitted, Frames already in transmission are invalidated by setting the Frame CRC to zero
6. Parity error during data transfer from Transient Buffer RAM A, B ⇒ Protocol Controller 1, 2
   a) **MHDS.PTBF1**, **MHDS.PTBF2** bit is set
7. Parity error in data transfer from Transient Buffer RAM A, B ⇒ Message RAM (Parity error when reading Header Section of respective Message Buffer from Message RAM)
   a) **MHDS.PMR** bit is set
   b) **MHDS.FMBD** bit is set to indicate that **MHDS.FMB** points to a faulty Message Buffer
   c) **MHDS.FMB** indicates the number of the faulty Message Buffer
   d) The Data Section of the respective Message Buffer is not updated

8. Parity error in data transfer from Transient Buffer RAM A, B ⇒ Message RAM (Parity error when reading Transient Buffer RAM A, B)
   a) **MHDS.PTBF1**, **MHDS.PTBF2** bit is set
   b) **MHDS.FMBD** bit is set to indicate that **MHDS.FMB** points to a faulty Message Buffer
   c) **MHDS.FMB** indicates the number of the faulty Message Buffer
9. Parity error during data transfer from Message RAM ⇒ Output Buffer RAM
   a) **MHDS.PMR** bit is set
   b) **MHDS.FMBD** bit is set to indicate that **MHDS.FMB** points to a faulty Message Buffer
   c) **MHDS.FMB** indicates the number of the faulty Message Buffer
10. Parity error during Host reading Output Buffer RAM
    a) • **MHDS.POBF** bit is set
11. Parity error during data read of Transient Buffer RAM A, B

When a parity error occurs when the Message Handler reads a Frame with Network Management information (PPI = 1) from the Transient Buffer RAM A, B the corresponding Network Management vector register NMV1 to NMV3 are not updated from that Frame.

20.7 Module Service Request

In general, service requests provide a close link to the protocol timing as they are triggered almost immediately when an error or status change is detected by the controller, a Frame is received or transmitted, a configured timer service request is activated, or a stop watch event occurred. This enables the Host to react very quickly on specific error conditions, status changes, or timer events. On the other hand too many service requests can cause the Host to miss deadlines required for the application. Therefore the Communication Controller supports disable / enable controls for each individual service request source separately.

An service request may be triggered when

- An error was detected
- A status flag is set
- A timer reaches a preconfigured value
- A message transfer from Input Buffer to Message RAM or from Message RAM to Output Buffer has completed
- A stop watch event occurred

Tracking status and generating service requests when a status change or an error occurs are two independent tasks. Regardless of whether an service request is enabled or not, the corresponding status is tracked and indicated by the Communication Controller. The Host has access to the actual status and error information by reading the Error Service Request Register **EIR** and the Status Service Request **SIR** Register.

Table 20-26 Module Service Request Flags and Service Request Line Enable

Register	Bit	Function
SIR	WST	Wakeup Status
	CAS	Collision Avoidance Symbol
	CYCS	Cycle Start Service Request
	TXI	Transmit Service Request
	RXI	Receive Service Request
	RFNE	Receive FIFO not Empty
	RFF	Receive FIFO Full
	NMVC	Network Management Vector Changed
	TI0	Timer Service Request 0
	TI1	Timer Service Request 1
	TIBC	Transfer Input Buffer Completed
	TOBC	Transfer Output Buffer Completed
	SWE	Stop Watch Event
	SUCS	Startup Completed Successfully
	MBSI	Message Buffer Status Interrupt
	SDS	Start of Dynamic Segment
	WUPA	Wakeup Pattern Channel A
	MTSA	MTS Received on Channel A
	WUPB	Wakeup Pattern Channel B
	MTSB	MTS Received on Channel B
ILE	EINT0	Enable Service Request Line 0
	EINT1	Enable Service Request Line 1
EIR	PEMC	Protocol Error Mode Changed
	CNA	Command Not Valid
	SFBM	SYNC Frames Below Minimum
	SFO	SYNC Frame Overflow
	CCF	Clock Correction Failure
	CCL	CHI Command Locked
	PERR	Parity Error
	RFO	Receive FIFO Overrun

Table 20-26 Module Service Request Flags and Service Request Line Enable

Register	Bit	Function
EIR	EFA	Empty FIFO Access
	IIBA	Illegal Input Buffer Access
	IOBA	Illegal Output Buffer Access
	MHF	Message Handler Constraints Flag
	EDA	Error Detected on Channel A
	LTVA	Latest Transmit Violation Channel A
	TABA	Transmission Across Boundary Channel A
	EDB	Error Detected on Channel B
	LTVB	Latest Transmit Violation Channel B
	TABB	Transmission Across Boundary Channel B

The interrupt lines to the Host TINT0SR and TINT1SR are controlled by the enabled interrupts. In addition each of the two interrupt lines can be enabled / disabled separately by programming bit **ILE.EINT0**/**INT0SRC.SRE** and **ILE.EINT1**/ **INT1SRC.SRE**.

The interrupt lines to the Host NDAT0SR and NDAT1SR are controlled by the enabled new data interrupts (**NDIC1** to **NDIC4**). In addition each of the two interrupt lines can be enabled / disabled separately by programming bit **NDAT0SRC.SRE** and **NDAT1SRC.SRE**.

The interrupt lines to the Host MBSC0SR and MBSC1SR are controlled by the enabled new data interrupts (**MSIC1** to **MSIC4**). In addition each of the two interrupt lines can be enabled / disabled separately by programming bit **MBSC0SRC.SRE** and **MBSC1SRC.SRE**.

The two timer service requests generated by service request timer 0 and 1 are available on pins TINT0SR and TINT1SR. They can be configured via the Timer 0 and Timer 1 Configuration register. In addition each of the two interrupt lines can be enabled / disabled separately by programming bit **TINT0SRC.SRE** and **TINT1SRC.SRE**.

A stop watch event may be triggered via input pin STPWn.

The status of the data transfer between IBF / OBF and the Message RAM is signalled on signals IBUSY and OBUSY. When a transfer has completed bit **SIR.TIBC** or **SIR.TOBC** is set.

20.8 Restrictions

The following restrictions have to be considered when programming the E-Ray IP-module. A violation of these restrictions may lead to an erroneous behavior of the E-Ray IP-module.

20.8.1 Message Buffers with the same Frame ID

If two or more Message Buffers are configured with the same Frame ID, and if they have a matching cycle counter filter value for the same slot, then the Message Buffer with the lowest Message Buffer number is used.

Sharing of a static time slot via cycle counter filtering between different nodes of a FlexRay™ network is **not** allowed.

20.8.2 Data Transfers between IBF / OBF and Message RAM

The time required to transfer the contents of a Message Buffer between IBF / OBF and Message RAM depends on the setup time to start the first transfer, the number of 4-byte words to be transferred, and the number of concurrent tasks to be managed by the Message Handler. The number of 4-byte words varies from 4 (Header Section only) to 68 (Header + maximum Data Section) while the number of concurrent task varies from one to three.

The following concurrent tasks are executed under control of the Message Handler:

- Data transfer between IBF or OBF and Message RAM
- Data transfer between TBF1 and Message RAM, search next TX / RX Message Buffer CHA
- Data transfer between TBF2 and Message RAM, search next TX / RX Message Buffer CHB

Transfers between IBF and Message RAM respectively Message RAM and OBF can only be handled one after another. In case that e.g. a transfer between IBF and Message RAM has been started shortly before a transfer between Message RAM and OBF is requested, the OBF transfer has to wait until the IBF transfer has completed.

The relative time is measured in f_{CLC_ERAY} cycles. Absolute time depends on the actual f_{CLC_ERAY} cycle period.

cyclestrans = (remaining cycles of transfer running) + (cycles of requested transfer)

cyclestrans = cyclesrem + cyclesreq

cyclesrem = (number of concurrent tasks) * (setup time + (number of 4-byte words)rem)

cyclesreq = (number of concurrent tasks) * (setup time + (number of 4-byte words)req)

setup time = 2 f_{CLC_ERAY} cycles

Under worst case conditions a transfer is requested directly after the previous transfer started:

Max. number of f_{CLC_ERAY} cycles: cyclestrans = (3 * (2 + 68)) + (3 * (2 + 68)) = 420

Worst case timing: timetrans(40MHz) = 420 * 25ns = 10.5 ms

20.9 Known non functional Features of E-Ray Module Revision 1.0.1

The implemented E-Ray IP Block does not contain all function described throughout this document.

Reception of more than gSyncNodeMax different SYNC Frames per double cycle may lead to incorrect offset correction value.

In case of receiving gSyncNodeMax or more SYNC Frames in an even cycle, only Frames with the same SYNC Frame IDs, as received in the even cycle, may be used for offset correction term calculation in the following odd cycle.

The erratum is limited to the case where more than gSyncNodeMax nodes are configured to transmit SYNC Frames and where different sets of SYNC Frames are transmitted in even and odd cycle.

In the described case the offset correction term may base on a different set of SYNC Frames than the rate correction term. In this case registers ESIDn / OSIDn hold the IDs of the first received SYNC Frames up to gSyncNodeMax used for offset correction term calculation.

Workaround: Avoid faulty configurations with more than gSyncNodeMax nodes configured to be transmitter of SYNC Frames. But problem only appears in case of faulty configurations where the number of sync nodes configured in a cluster is greater than gSyncNodeMax (GTUC2.SNM).

Time stamp of the wrong channel may be used for offset correction term calculation if reception time stamps of a SYNC Frame are different for channel A and B.

In case the temporal deviation (= time between primary time reference point and action point offset) is different for channel A and B and the values are greater than or equal zero on one channel and negative on the other channel, the channel with a relative deviation value greater than or equal zero is chosen for offset correction term calculation instead of the negative value.

The erratum is limited to the case where both channels are used and if there is a large difference in the propagation delays on channel A and B.

In case of the described relation between measured deviation values on channel A and B, the calculated offset correction term may have an error of maximum the difference between the two deviation values. Thus, the error of the local time of the node is limited to the difference of the temporal deviation values of both channels.

Workaround: For dual channel FlexRay™ systems SYNC Frames have to be transmitted on both channels. In practice, the propagation delay between two nodes is expected to be nearly the same on both channels. If this is not the case, the channel depending parameter pDelayCompensation[A/B] (GTUC5.DCA,B) has to be used to compensate the different propagation delays. With a correct adjustment of the parameter the difference of the deviation values on both channels is expected to be very low. Therefore, the error of the local time caused by the implementation error is also minimized. But problem only appears with dual-channel topologies. Workaround helps to minimize the impact of the faulty behavior.

For sync nodes the error interrupt flag EIR.SFO may be set too late.

The E-Ray acting as sync node does not consider its own SYNC Frame for counting number of SYNC Frames. Thus it sets the error interrupt flag EIR.SFO if it receives more than gSyncNodeMax (GTUC2.SNM) SYNC Frames.

The erratum is limited to the case where a sync node receives exactly gSyncNodeMax SYNC Frames from other sync nodes during one communication cycle.

In the described case error interrupt flag EIR.SFO is not set.

Workaround: Avoid faulty configurations with more than gSyncNodeMax nodes configured to be transmitter of SYNC Frames. But problem only appears in case of faulty configurations where the number of sync nodes configured in a cluster is greater than gSyncNodeMax (GTUC2.SNM).

Valid Frame detection at slot boundary may lead to an incorrect temporal deviation value of a SYNC Frame in the following slot.

In case a non SYNC Frame is detected to be valid simultaneously with the slot boundary, and in the following slot a SYNC Frame is received, the temporal deviation value (= time between primary time reference point and action point) of the received SYNC Frame is erroneously set to the measured value of the previous non SYNC Frame.

The erratum is limited to the case where a non SYNC Frame is received in one slot and a SYNC Frame is received in the following slot. Additionally, the detection of “valid Frame” (fsp_val_Frame) for the non SYNC Frame has to be exactly at the slot boundary.

The temporal deviation value of the SYNC Frame is set to the deviation value of the previous non SYNC Frame. The positive shift of the non SYNC Frame in direction of slot end results in a relative large positive deviation value (order of magnitude of the parameter gdActionPointOffset). With more than two SYNC Frame senders within the cluster, this large value is most probably not taken into account for correction term calculation because of the nature of the fault-tolerant midpoint algorithm. If the faulty value is used for offset correction term calculation, the resulting offset shift will be corrected in the following cycles.

Workaround: There is no work around, but the resulting incorrect offset shift will be corrected in following cycles.

SYNC Frame reception after noise or aborted Frame before action point.

In case noise or an aborted Frame leads to the detection of a secondary time reference point (STRP), and after this, a valid SYNC Frame is detected in the same slot and the STRP of the valid SYNC Frame occurs simultaneously with the action point, the temporal deviation value of the first detected STRP is stored instead of the value of the correct STRP.

The erratum is limited to the case of noise before reception of a valid SYNC Frame or a Frame reception starting before action point is aborted and then a valid SYNC Frame is received in the same slot.

In the described case a wrong deviation value is used for correction term calculation. Depending on number of SYNC Frames and other measured temporal deviation values it may lead to an incorrect rate or offset correction value.

Workaround: There is no work around, but the problem occurs only in case of noise or Frame abortion between slot boundary and action point. The resulting incorrect rate and offset corrections will be corrected in following cycles. Very low probability for combination of both conditions.

20.10 E-Ray Module Implementation

This section describes the E-Ray interfaces as implemented in TC1797 with the clock control, port and DMA connections, interrupt control, and address decoding.

Figure 20-27 shows a detailed view of the E-Ray interface.

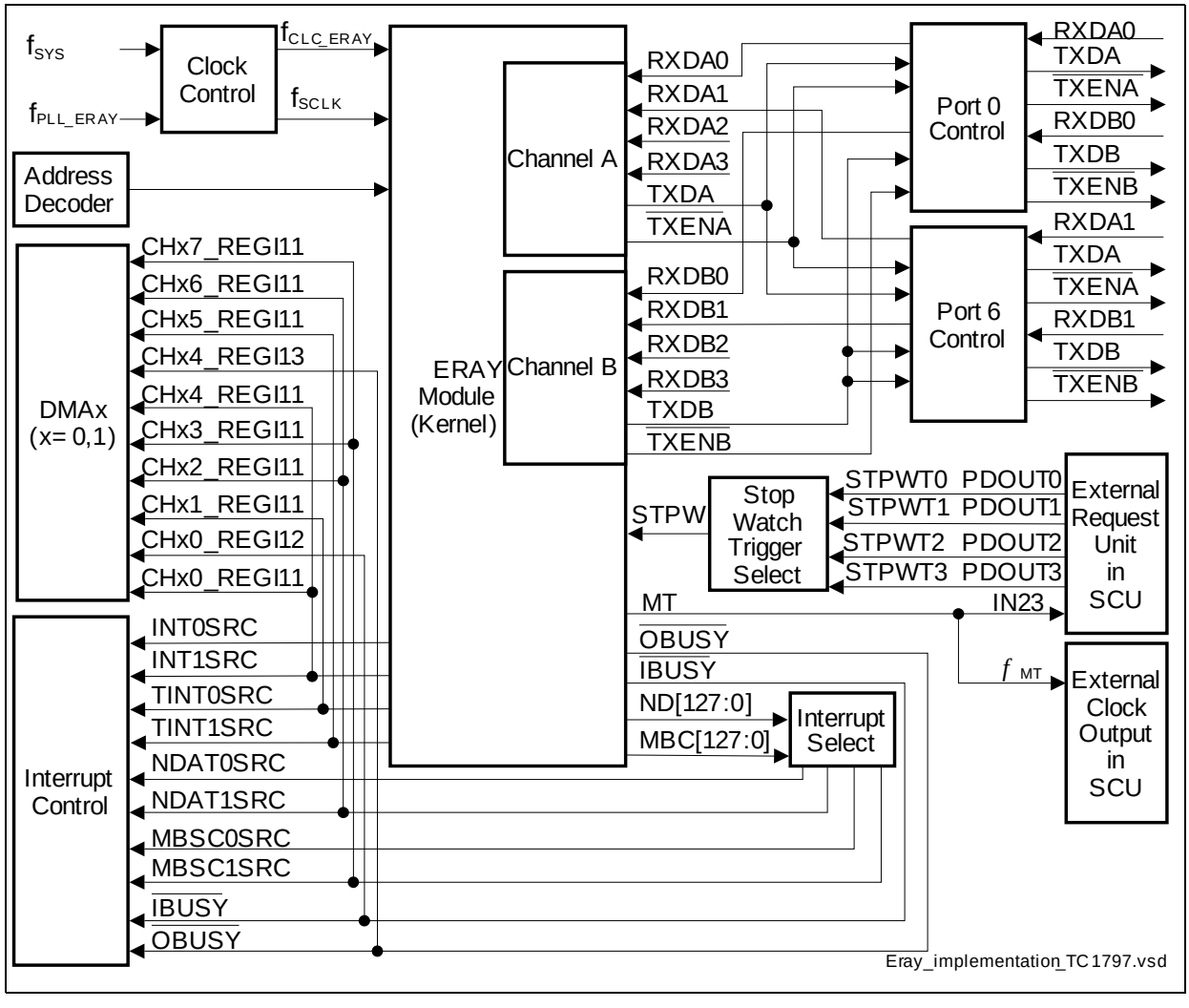


Figure 20-27 Detailed Block Diagram of the E-Ray Interface

20.10.1 Interconnections of the E-Ray Module

The E-Ray module has 2 FlexRay™ communication channels, channel A and channel B. Each channel provides a set of signals to drive a bus driver. The E-Ray module requires two different clocks, a sampling clock of the FlexRay™ bus f_{SCLK}. f_{SCLK} has to be 8 times the baud rate of the FlexRay™ communication. A second clock f_{CLC_ERAY} is used for the main protocol controller state machine and the customer interface logic. To enable deactivation of the E-Ray Module, f_{CLC_ERAY} and f_{SCLK} may be disabled (clock

gated) by the **CLC.DISR** Enable E-Ray (Clock Gating) bit.The following items are described in this section:

- E-Ray module (kernel) external registers
- Port control and connections
  - I/O port line assignment
  - I/O function selection
  - Pad driver characteristics selection
- On-chip connections
  - SCU Connections
  - DMA connections
- Module clock generation
- Interrupt registers
- E-Ray address map

20.10.2 Port Control and Connections

This section describes the I/O connections of the E-Ray module.

20.10.2.1 Input/Output Function Selection

Table 20-27 shows how bits and bit fields must be programmed for the required I/O functionality of the E-Ray I/O lines. This table also shows the values of the peripheral input select registers.

Table 20-27 XE-Ray1 I/O Control Selection and Setup

FlexRay™ Channel	Port Lines	Input Select Register	Input/Output Control Register Bits	I/O
A	RXDA0/ P0.9	ERAY_**CUST1**.**RISA** = 00_B	P0_IOCR8.PC9 = $0XXX_B$	Input
	RXDA1/ P6.12	ERAY_**CUST1**.**RISA** = 01_B	P6_IOCR12.PC12 = $0XXX_B$	Input
	RXDA2	ERAY_**CUST1**.**RISA** = 10_B		Input
	RXDA3	ERAY_**CUST1**.**RISA** = 11_B		Input
	TXDA/ P0.14	not applicable	P0_IOCR12.PC14 = $1X01_B$	Output
	TXDA/ P6.13	not applicable	P6_IOCR12.PC13 = $1X10_B$	Output
	$\overline{\text{TXENA}}$/ P0.10	not applicable	P0_IOCR8.PC10 = $1X01_B$	Output
	$\overline{\text{TXENA}}$/ P6.10	not applicable	P6_IOCR8.PC10 = $1X11_B$	Output
B	RXDB0/ P0.13	ERAY_**CUST1**.**RISB** = 00_B	P0_IOCR12.PC13 = $0XXX_B$	Input
	RXDB1/ P6.14	ERAY_**CUST1**.**RISB** = 01_B	P6_IOCR12.PC14 = $0XXX_B$	Input
	RXDB2	ERAY_**CUST1**.**RISB** = 10_B		Input
	RXDB3	ERAY_**CUST1**.**RISB** = 11_B		Input
	TXDB/ P0.12	not applicable	P0_IOCR12.PC12 = $1X01_B$	Output
	TXDB/ P6.15	not applicable	P6_IOCR12.PC15 = $1X10_B$	Output
	$\overline{\text{TXENB}}$/ P0.11	not applicable	P0_IOCR8.PC11 = $1X01_B$	Output
	$\overline{\text{TXENB}}$/ P6.11	not applicable	P6_IOCR8.PC11 = $1X11_B$	Output

20.10.3 On-Chip Connections

This section describes all on-chip interconnections of the E-Ray modules except the connections to I/O ports (see **Section 20.10.2**).

20.10.3.1 E-Ray Connections with DMA

The E-Ray module of the TC1797 has several on-chip interconnections to the DMA modules. **Table 20-28** shows these interconnections. These enable the DMA to handle different service request of E-Ray module via the DMA.

Table 20-28 DMA Request Assignment for DMA Sub-Block 0 and DMA Sub-Block 1

DMA Channel	ERAY Output Signal	DMA Request Input Line	Selected by
00	INT1SRC	CH00_REGI11	CHCR00.PRSEL = 1011_B
00	IBUSY	CH00_REGI12	CHCR00.PRSEL = 1100_B
01	TINT0SRC	CH01_REGI11	CHCR01.PRSEL = 1011_B
02	NDAT1SRC	CH02_REGI11	CHCR02.PRSEL = 1011_B
03	MBSC1SRC	CH03_REGI11	CHCR03.PRSEL = 1011_B
04	INT1SRC	CH04_REGI11	CHCR04.PRSEL = 1011_B
04	OBUSY	CH04_REGI13	CHCR04.PRSEL = 1101_B
05	TINT1SRC	CH05_REGI11	CHCR05.PRSEL = 1011_B
06	NDAT1SRC	CH06_REGI11	CHCR06.PRSEL = 1011_B
07	MBSC1SRC	CH07_REGI11	CHCR07.PRSEL = 1011_B
10	INT1SRC	CH10_REGI11	CHCR10.PRSEL = 1011_B
10	IBUSY	CH10_REGI12	CHCR10.PRSEL = 1100_B
11	TINT0SRC	CH11_REGI11	CHCR11.PRSEL = 1011_B
12	NDAT1SRC	CH12_REGI11	CHCR12.PRSEL = 1011_B
13	MBSC1SRC	CH13_REGI11	CHCR13.PRSEL = 1011_B
14	INT1SRC	CH14_REGI11	CHCR14.PRSEL = 1011_B
14	OBUSY	CH14_REGI13	CHCR14.PRSEL = 1101_B
15	TINT1SRC	CH15_REGI11	CHCR15.PRSEL = 1011_B
16	NDAT1SRC	CH16_REGI11	CHCR16.PRSEL = 1011_B
17	MBSC1SRC	CH17_REGI11	CHCR17.PRSEL = 1011_B

20.10.3.2 E-Ray Connections with the External Request Unit of SCU

The E-Ray module of the TC1797 has several on-chip interconnections to the External Request Unit (ERU) in the SCU to externally trigger stop watch events and to provide a global time e.g. to the on chip timers. **Table 20-29** and **Table 20-30** show these interconnections.

Table 20-29 External Stop Watch Request Assignment

ERAY Input Signal	ERU Request Output Line	Selected by
STPWT0	ERU_PDOUT0	**CUST1.STPWTS** = 00_B
STPWT1	ERU_PDOUT1	**CUST1.STPWTS** = 01_B
STPWT2	ERU_PDOUT2	**CUST1.STPWTS** = 10_B
STPWT3	ERU_PDOUT3	**CUST1.STPWTS** = 11_B

Table 20-30 Global Macrotick Connection to ERU

ERAY Output Signal	ERU Request Input Line	Selected by
MT	ERU_IN23	ERU_EICR1.EXIS2 = 11_B

20.10.3.3 E-Ray Connections with the Parity Error Handling Unit of SCU

The E-Ray module of the TC1797 has one on-chip interconnection to the Parity Error Handling Unit in the SCU to trigger a Parity Error Trap. **Table 20-31** shows this interconnection.

Table 20-31 Parity Error Signalling to SCU

ERAY Output Signal	SCU Input Line	Selected by
PERR	PERT	PETCR.PERT = 0_B

20.10.3.4 E-Ray Connections with the External Clock Output of SCU

The E-Ray module of the TC1797 has one on-chip interconnections to the External Clock Output Unit in the SCU to distribute externally as also internally the Macro Tick as time base for distributed system control e.g. to the GPTA® as global system timer or external devices. **Table 20-32** shows this interconnection.

Table 20-32 Global Macrotick Connection to External Clock Output

ERAY Output Signal	External Clock Output	Selected by
MT	f_{MT}	SCU_EXCTCON.SEL0 = 1111_B

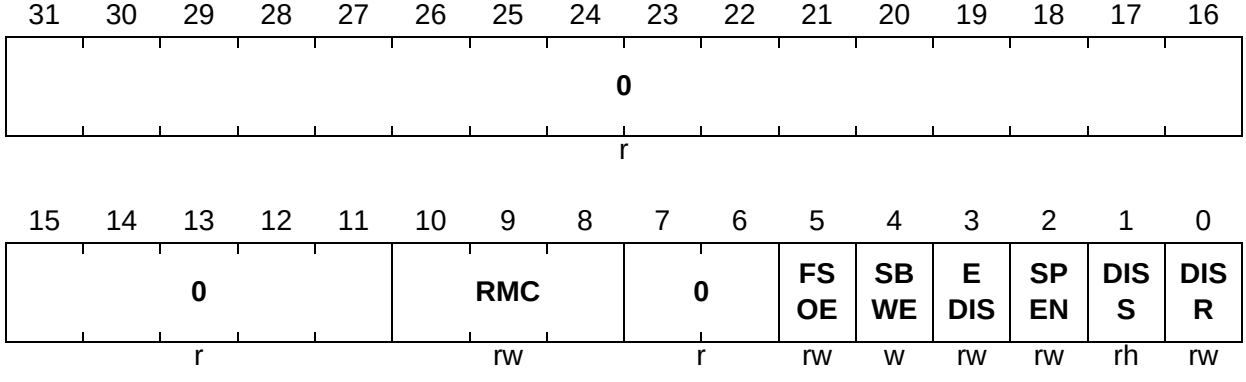
20.10.4 Clock Control Register

The clock control register makes it possible to control (enable/disable) the E-Ray module control clock f_{CLC_ERAY}. The clock signal f_{CLC_ERAY} is used by the E-Ray as a clock for internal control operations but not for FlexRay™ Bus Signal Sampling.

After an application reset, by default, the startup software (SSW) will disable the module by setting CLC.DISR and thereby requesting the HW to clear bit CLC.DISS.

Note: The application SW must make sure that the Transceiver Enable (TXENA, TXENB) pins are forced to their inactive state first, before the module clock is switched off.

CLC
ERAY Clock Control Register (0000_H) Reset Value: 0000 0100$_H$



Field	Bits	Type	Description
DISR	0	rw	**E-Ray Module Disable Request Bit** Used for enable/disable control of the E-Ray module. *Note: This bit disables the kernel clocks f_{CLC_ERAY} and the sampling clock f_{SCLK}.*
DISS	1	rh	**E-Ray Module Disable Status Bit** Bit indicates the current status of the E-Ray module.
SPEN	2	rw	**E-Ray Module Suspend Enable for OCDS** Used to enable the suspend mode.
EDIS	3	rw	**External Request Disable** Used to control the external clock disable request.
SBWE	4	w	**E-Ray Module Suspend Bit Write Enable for OCDS** Determines whether SPEN and FSOE are write-protected.
FSOE	5	rw	**Fast Switch Off Enable** Used for fast clock switch off in suspend mode.

Field	Bits	Type	Description
RMC	[10:8]	rw	**Clock Divider in Run Mode** 000_B No clock signal f_{CLC_ERAY} generated (default after reset) 001_B Clock $f_{CLC\_ERAY} = f_{SYS}$ selected 010_B Clock $f_{CLC\_ERAY} = f_{SYS}$ /2 selected 011_B Clock $f_{CLC\_ERAY} = f_{SYS}$ /3 selected 100_B Clock $f_{CLC\_ERAY} = f_{SYS}$ /4 selected 101_B Clock $f_{CLC\_ERAY} = f_{SYS}$ /5 selected 110_B Clock $f_{CLC\_ERAY} = f_{SYS}$ /6 selected 111_B Clock $f_{CLC\_ERAY} = f_{SYS}$ /7 selected *Note: This bit field is not affected by an application reset.* *Note: This bit field only controls the kernel clock f_{CLC_ERAY} and not the sampling clock f_{SCLK}.*
0	[7:6], [31:11]	r	**Reserved** Read as 0; should be written with 0.

Note: After an application reset, the f_{CLC_ERAY} clock is disabled (DISS set). Therefore, the E-Ray modules clock generation is completely disabled.

20.10.5 Interrupt Registers

Two different type of Interrupt Registers are described within this chapter.
The Interrupt Control register enable the selection of the Service Request used to signal an event. The Interrupt Control registers **NDIC1** to **NDIC4** select the service request node used for New Data Events.The Interrupt Control registers **MSIC1** to **MSIC4** select the service request node used for Message Buffer Status Changed Events.
The Interrupt Service Request Control Registers control the eight service request nodes.

New Data Interrupt Control 1 (NDIC1)

This New Data Interrupt Control register controls the interrupt that becomes active (**NDAT0SRC** or **NDAT1SRC**) on a ND flag turning active of all configured Message Buffers 0 to Message Buffers 31.

NDIC1
New Data Interrupt Control 1 (03A8$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
NDIP 31	**NDIP 30**	**NDIP 29**	**NDIP 28**	**NDIP 27**	**NDIP 26**	**NDIP 25**	**NDIP 24**	**NDIP 23**	**NDIP 22**	**NDIP 21**	**NDIP 20**	**NDIP 19**	**NDIP 18**	**NDIP 17**	**NDIP 16**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NDIP 15	**NDIP 14**	**NDIP 13**	**NDIP 12**	**NDIP 11**	**NDIP 10**	**NDIP 9**	**NDIP 8**	**NDIP 7**	**NDIP 6**	**NDIP 5**	**NDIP 4**	**NDIP 3**	**NDIP 2**	**NDIP 1**	**NDIP 0**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
NDIPn (n = 0-31)	n	rw	**New Data Interrupt Pointer n (n = 0-31)** NDIPn determines the interrupt (**NDAT0SRC** or **NDAT1SRC**) of the service request output that becomes active on a New Data Flag becoming active. 0_B **NDAT0SRC** selected for New Data Service Request 1_B **NDAT1SRC** selected for New Data Service Request

New Data Interrupt Control 2 (NDIC2)

This New Data Interrupt Control register controls the interrupt that becomes active (**NDAT0SRC** or **NDAT1SRC**) on a ND flag turning active of all configured Message Buffers 32 to Message Buffers 63.

NDIC2
New Data Interrupt Control 2 ($03AC_H$) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
NDIP 63	**NDIP 62**	**NDIP 61**	**NDIP 60**	**NDIP 59**	**NDIP 58**	**NDIP 57**	**NDIP 56**	**NDIP 55**	**NDIP 54**	**NDIP 53**	**NDIP 52**	**NDIP 51**	**NDIP 50**	**NDIP 49**	**NDIP 48**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NDIP 47	**NDIP 46**	**NDIP 45**	**NDIP 44**	**NDIP 43**	**NDIP 42**	**NDIP 41**	**NDIP 40**	**NDIP 39**	**NDIP 38**	**NDIP 37**	**NDIP 36**	**NDIP 35**	**NDIP 34**	**NDIP 33**	**NDIP 32**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
NDIPn (n = 32-63)	n - 32	rw	**New Data Interrupt Pointer n (n = 32-63)** NDIPn determines the interrupt (**NDAT0SRC** or **NDAT1SRC**) of the service request output that becomes active on a New Data Flag becoming active. 0_B **NDAT0SRC** selected for New Data Service Request 1_B **NDAT1SRC** selected for New Data Service Request

New Data Interrupt Control 3 (NDIC3)

This New Data Interrupt Control register controls the interrupt that becomes active (**NDAT0SRC** or **NDAT1SRC**) on a ND flag turning active of all configured Message Buffers 64 to Message Buffers 95.

NDIC3
New Data Interrupt Control 3 (03B0$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
NDIP 95	**NDIP 94**	**NDIP 93**	**NDIP 92**	**NDIP 91**	**NDIP 90**	**NDIP 89**	**NDIP 88**	**NDIP 87**	**NDIP 86**	**NDIP 85**	**NDIP 84**	**NDIP 83**	**NDIP 82**	**NDIP 81**	**NDIP 80**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NDIP 79	**NDIP 78**	**NDIP 77**	**NDIP 76**	**NDIP 75**	**NDIP 74**	**NDIP 73**	**NDIP 72**	**NDIP 71**	**NDIP 70**	**NDIP 69**	**NDIP 68**	**NDIP 67**	**NDIP 66**	**NDIP 65**	**NDIP 64**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
NDIPn (n = 64-95)	n - 64	rw	**New Data Interrupt Pointer n (n = 64-95)** NDIPn determines the interrupt (**NDAT0SRC** or **NDAT1SRC**) of the service request output that becomes active on a New Data Flag becoming active. 0_B **NDAT0SRC** selected for New Data Service Request 1_B **NDAT1SRC** selected for New Data Service Request

New Data Interrupt Control 4 (NDIC4)

This New Data Interrupt Control register controls the interrupt that becomes active (**NDAT0SRC** or **NDAT1SRC**) on a ND flag turning active of all configured Message Buffers 96 to Message Buffers 127.

NDIC4
New Data Interrupt Control 4 ($03B4_H$) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
NDIP 127	**NDIP 126**	**NDIP 125**	**NDIP 124**	**NDIP 123**	**NDIP 122**	**NDIP 121**	**NDIP 120**	**NDIP 119**	**NDIP 118**	**NDIP 117**	**NDIP 116**	**NDIP 115**	**NDIP 114**	**NDIP 113**	**NDIP 112**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NDIP 111	**NDIP 110**	**NDIP 109**	**NDIP 108**	**NDIP 107**	**NDIP 106**	**NDIP 105**	**NDIP 104**	**NDIP 103**	**NDIP 102**	**NDIP 101**	**NDIP 100**	**NDIP 99**	**NDIP 98**	**NDIP 97**	**NDIP 96**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
NDIPn (n = 96-127)	n - 96	rw	**New Data Interrupt Pointer n (n = 96-127)** NDIPn determines the interrupt (**NDAT0SRC** or **NDAT1SRC**) of the service request output that becomes active on a New Data Flag becoming active. 0_B **NDAT0SRC** selected for New Data Service Request 1_B **NDAT1SRC** selected for New Data Service Request

Message Buffer Status Changed Interrupt Control 1 (MSIC1)

This Message Buffer Status Change Interrupt Control register controls the interrupt that becomes active (**MBSC0SRC** or **MBSC1SRC**) on a MBC flag of all configured Message Buffer 0 to Message Buffer 31 turning active.

MSIC1
Message Buffer Status Changed Interrupt Control 1 **(03B8$_H$)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MSIP 31	**MSIP 30**	**MSIP 29**	**MSIP 28**	**MSIP 27**	**MSIP 26**	**MSIP 25**	**MSIP 24**	**MSIP 23**	**MSIP 22**	**MSIP 21**	**MSIP 20**	**MSIP 19**	**MSIP 18**	**MSIP 17**	**MSIP 16**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSIP 15	**MSIP 14**	**MSIP 13**	**MSIP 12**	**MSIP 11**	**MSIP 10**	**MSIP 9**	**MSIP 8**	**MSIP 7**	**MSIP 6**	**MSIP 5**	**MSIP 4**	**MSIP 3**	**MSIP 2**	**MSIP 1**	**MSIP 0**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
MSIPn (n = 0-31)	n	rw	**Message Buffer Status Changed Interrupt Pointer n (n = 0-31)** MSIPn determines the interrupt (**MBSC0SRC** or **MBSC1SRC**) of the service request output that becomes active on a Message Buffer Status Changed Flag becoming active. 0_B **MBSC0SRC** selected for Message Buffer Status Changed Service Request 1_B **MBSC1SRC** selected for Message Buffer Status Changed Service Request

Message Buffer Status Changed Interrupt Control 2 (MSIC2)

This Message Buffer Status Change Interrupt Control register controls the interrupt that becomes active (**MBSC0SRC** or **MBSC1SRC**) on a MBC flag of all configured Message Buffer 32 to Message Buffer 63 turning active.

MSIC2
Message Buffer Status Changed Interrupt Control 2 ($03BC_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MSIP 63	**MSIP 62**	**MSIP 61**	**MSIP 60**	**MSIP 59**	**MSIP 58**	**MSIP 57**	**MSIP 56**	**MSIP 55**	**MSIP 54**	**MSIP 53**	**MSIP 52**	**MSIP 51**	**MSIP 50**	**MSIP 49**	**MSIP 48**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSIP 47	**MSIP 46**	**MSIP 45**	**MSIP 44**	**MSIP 43**	**MSIP 42**	**MSIP 41**	**MSIP 40**	**MSIP 39**	**MSIP 38**	**MSIP 37**	**MSIP 36**	**MSIP 35**	**MSIP 34**	**MSIP 33**	**MSIP 32**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
MSIPn (n = 32-63)	n - 32	rh	**Message Buffer Status Changed Interrupt Pointer n (n = 32-63)** MSIPn determines the interrupt (**MBSC0SRC** or **MBSC1SRC**) of the service request output that becomes active on a Message Buffer Status Changed Flag becoming active. 0_B **MBSC0SRC** selected for Message Buffer Status Changed Service Request 1_B **MBSC1SRC** selected for Message Buffer Status Changed Service Request

Message Buffer Status Changed Interrupt Control 3 (MSIC3)

This Message Buffer Status Change Interrupt Control register controls the interrupt that becomes active (**MBSC0SRC** or **MBSC1SRC**) on a MBC flag of all configured Message Buffer 64 to Message Buffer 95 turning active.

MSIC3
Message Buffer Status Changed Interrupt Control 3
($03C0_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MSIP 95	**MSIP 94**	**MSIP 93**	**MSIP 92**	**MSIP 91**	**MSIP 90**	**MSIP 89**	**MSIP 88**	**MSIP 87**	**MSIP 86**	**MSIP 85**	**MSIP 84**	**MSIP 83**	**MSIP 82**	**MSIP 81**	**MSIP 80**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSIP 79	**MSIP 78**	**MSIP 77**	**MSIP 76**	**MSIP 75**	**MSIP 74**	**MSIP 73**	**MSIP 72**	**MSIP 71**	**MSIP 70**	**MSIP 69**	**MSIP 68**	**MSIP 67**	**MSIP 66**	**MSIP 65**	**MSIP 64**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
MSIPn (n = 64-95)	n - 64	rw	**Message Buffer Status Changed Interrupt Pointer n (n = 64-95)** MSIPn determines the interrupt (**MBSC0SRC** or **MBSC1SRC**) of the service request output that becomes active on a Message Buffer Status Changed Flag becoming active. 0_B **MBSC0SRC** selected for Message Buffer Status Changed Service Request 1_B **MBSC1SRC** selected for Message Buffer Status Changed Service Request

Message Buffer Status Changed Interrupt Control 4 (MSIC4)

This Message Buffer Status Change Interrupt Control register controls the interrupt that becomes active (**MBSC0SRC** or **MBSC1SRC**) on a MBC flag of all configured Message Buffer 96 to Message Buffer 127 turning active.

MSIC4
Message Buffer Status Changed Interrupt Control 4
($03C4_H$) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MSIP 127	**MSIP 126**	**MSIP 125**	**MSIP 124**	**MSIP 123**	**MSIP 122**	**MSIP 121**	**MSIP 120**	**MSIP 119**	**MSIP 118**	**MSIP 117**	**MSIP 116**	**MSIP 115**	**MSIP 114**	**MSIP 113**	**MSIP 112**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSIP 111	**MSIP 110**	**MSIP 109**	**MSIP 108**	**MSIP 107**	**MSIP 106**	**MSIP 105**	**MSIP 104**	**MSIP 103**	**MSIP 102**	**MSIP 101**	**MSIP 100**	**MSIP 99**	**MSIP 98**	**MSIP 97**	**MSIP 96**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
MSIPn (n = 96-127)	n - 96	rw	**Message Buffer Status Changed Interrupt Pointer n (n = 96-127)** MSIPn determines the interrupt (**MBSC0SRC** or **MBSC1SRC**) of the service request output that becomes active on a Message Buffer Status Changed Flag becoming active. 0_B **MBSC0SRC** selected for Message Buffer Status Changed Service Request 1_B **MBSC1SRC** selected for Message Buffer Status Changed Service Request

Service Request Control Registers

Each of the service request outputs of the E-Ray module kernels is able to generate an interrupt and is controlled by an interrupt service request control register **INT0SRC**, **INT1SRC**, **TINT0SRC**, **TINT1SRC**, **NDAT0SRC**, **NDAT1SRC**, **MBSC0SRC**, **MBSC1SRC**, **OBUSYSRC**, and **IBUSYSRC**.The service request **OBUSYSRC**, and **IBUSYSRC** is generated, when the Output Buffer or Input Buffer switches from busy state to the state being accessible by the host.

INT0SRC
Interrupt 0 Service Request Control Register
($3EC_H$) **Reset Value: 0000 0000$_H$**

INT1SRC
Interrupt 1 Service Request Control Register
($3E8_H$) **Reset Value: 0000 0000$_H$**

TINT0SRC
Timer Interrupt 0 Service Request Control Register
($3E4_H$) **Reset Value: 0000 0000$_H$**

TINT1SRC
Timer Interrupt 1 Service Request Control Register
($3E0_H$) **Reset Value: 0000 0000$_H$**

NDAT0SRC
New Data 0 Service Request Control Register
($3DC_H$) **Reset Value: 0000 0000$_H$**

NDAT1SRC
New Data 1 Service Request Control Register
($3D8_H$) **Reset Value: 0000 0000$_H$**

MBSC0SRC
Message Buffer Status Changed 0 Service Request Control Register
($3D4_H$) **Reset Value: 0000 0000$_H$**

MBSC1SRC
Message Buffer Status Changed 1 Service Request Control Register
($3D0_H$) **Reset Value: 0000 0000$_H$**

OBUSYSRC
Output Buffer Busy Service Request Control Register
($3CC_H$) **Reset Value: 0000 0000$_H$**

IBUSYSRC
Input Buffer Busy Service Request Control Register
($3C8_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SETR	CLRR	SRR	SRE	0	TOS	0		SRPN							
w	w	rh	rw	r	rw	r		rw							

Field	Bits	Type	Description
SRPN	[7:0]	rw	**Service Request Priority Number**
TOS	10	rw	**Type of Service Control**
SRE	12	rw	**Service Request Enable**
SRR	13	rh	**Service Request Flag**
CLRR	14	w	**Request Clear Bit**
SETR	15	w	**Request Set Bit**
0	[9:8], 11, [31:16]	r	**Reserved** Read as 0; should be written with 0.

Note: Additional details on service request nodes and the service request control registers are described in Chapter "Interrupt System" of the TC1797 User´s Manual System Units part (Volume 1).

20.10.6 E-Ray Access Delay

Due to a hierarchical design approach read and write accesses require FPI wait sates.

Table 20-33 E-Ray Access Time

Address Range	FPI Wait States for Reading	FPI Wait States for Writing
Customer Register (**ID**, **CUST1**, **CUST3**)	1 wait state	0 wait state
Input Buffer and Output Buffer	2 wait states	1 wait state
Bus Interface Register: **CLC**, **NDIC1** to **NDIC4**, **MSIC1** to **MSIC4**, **INT0SRC**, **INT1SRC**, **TINT0SRC**, **TINT0SRC**, **NDAT0SRC**, **NDAT1SRC**, **MBSC0SRC**, **MBSC1SRC**.	1 wait states	0 wait state
ERAY Kernel Register	2 wait states	1 wait state

20.10.7 E-Ray Register Address Map

The E-Ray register map shown in **Figure 20-28**.

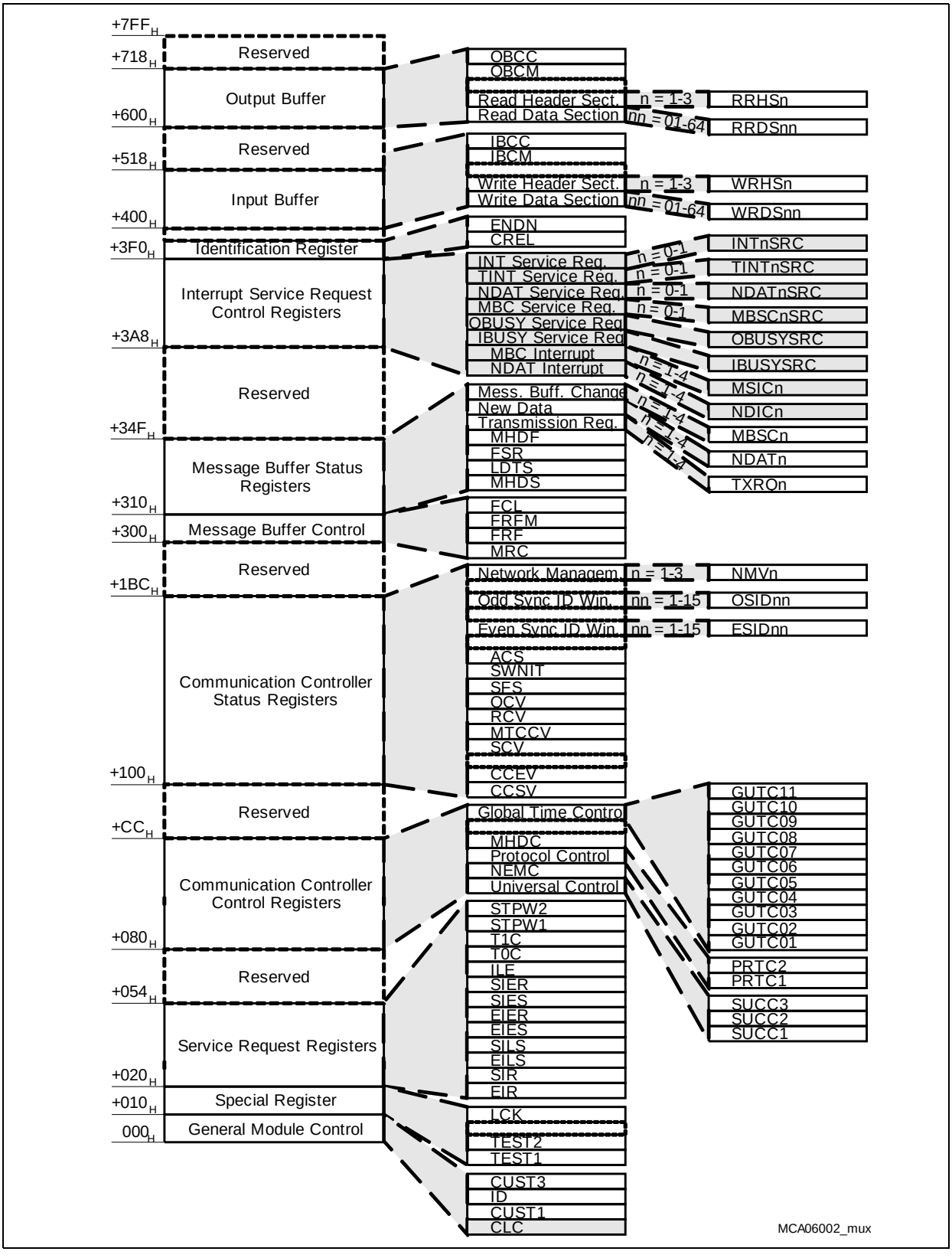


Figure 20-28 E-Ray Register Map

21 Micro Link Interface (MLI)

This chapter describes the Micro Link Interface module and the MLI protocol. It contains the following sections:

- Functional description of the MLI (see **Page 21-2**)
- Module kernel description (see **Page 21-27**)
- Operation the MLI module (see **Page 21-69**)
- MLI kernel register descriptions (see **Page 21-77**)
- Device implementation-specific descriptions and details (see **Page 21-127**)

*Note: The MLI kernel register names described in **Section 21.3** are referenced in the TC1797 User´s Manual by the module name prefix “MLI0_” for the MLI0 interface and “MLI1_” for the MLI1 interface.*

21.1 Functional Description

This chapter describes the functionality of the MLI interface.

- A general introduction to the interface (see **Page 21-2**)
- The MLI frame structure for data exchange (see **Page 21-10**)

21.1.1 General Introduction

The introduction comprises:

- An overview about the MLI (see **Page 21-2**)
- Naming conventions (see **Page 21-4**)
- A description of the MLI communication principles (see **Page 21-6**)

21.1.1.1 MLI Overview

The Micro Link Interface (MLI) is a fast synchronous serial interface to exchange data between microcontrollers or other devices, such as stand-alone peripheral components. **Figure 21-1** shows how two microcontrollers are typically connected together via their MLI interfaces.

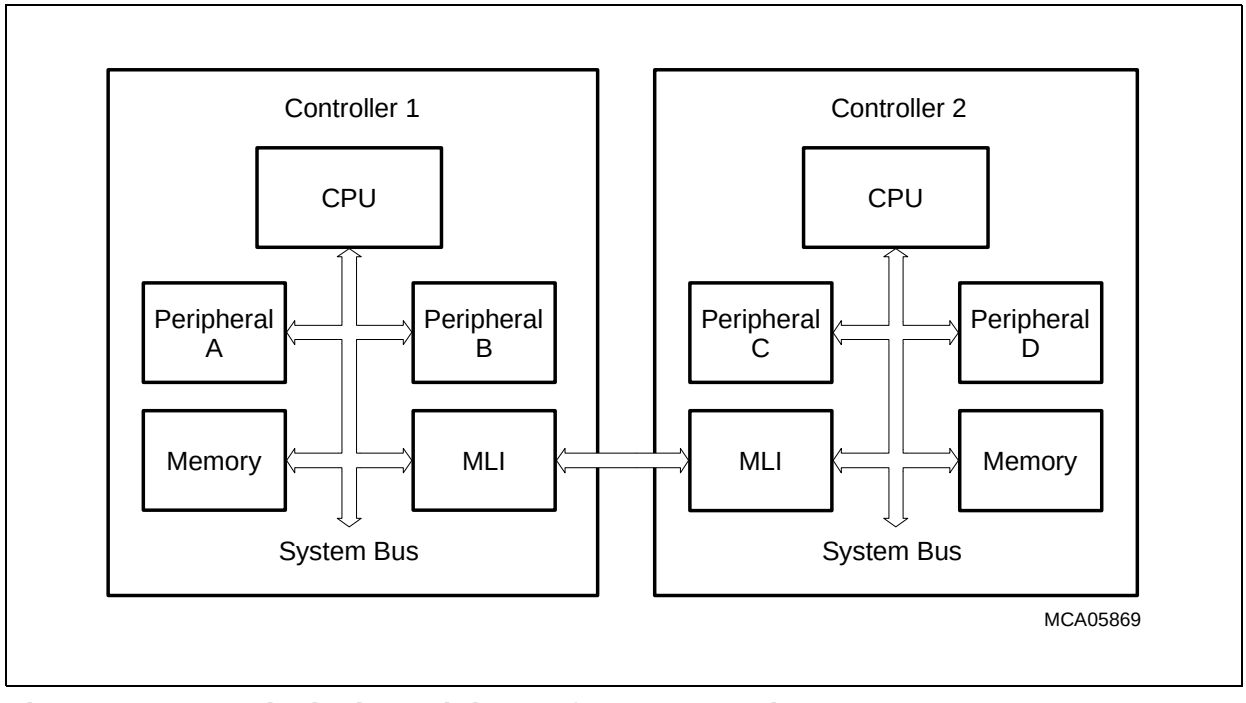


Figure 21-1 Typical Micro Link Interface Connection

Features

- Synchronous serial communication between an MLI transmitter and an MLI receiver
- Different system clock speeds supported in MLI transmitter and MLI receiver due to full handshake protocol (4 lines between a transmitter and a receiver)
- Fully transparent read/write access supported (= remote programming)
- Complete address range of target device available
- Specific frame protocol to transfer commands, addresses and data
- Error detection by parity bit
- 32-bit, 16-bit, or 8-bit data transfers supported
- Programmable baud rates
  - MLI transmitter baud rate: max. $f_{MLI}/2$ (= 45 Mbit/s @ 90 MHz module clock)
  - MLI receiver baud rate: max. f_{MLI}
- Address range protection scheme to block unauthorized accesses
- Multiple receiving devices supported

21.1.1.2 Naming Conventions

Local and Remote Controller

The terms “Local” and “Remote” Controller are assigned to the two partners (microcontrollers or other devices with MLI modules) of a serial MLI connection. The controller with an MLI module initiating a data exchange or a control task is defined as Local Controller. Each data exchange and control task starts with a frame transmission of the Local Controller. The controller with an MLI module reacting on received data exchange requests or executing control tasks is defined as Remote Controller. The terms “Local” and “Remote” are independent of the direction of the information flow (transmission or reception), except for Read Frames (always transmitted by the Local Controller) and Answer Frames (always transmitted by the Remote Controller). A Triggered Command Frame is transferred from the Local to the Remote Controller.

Due to the full duplex operation capability of an MLI module (independent transmitter and receiver), each microcontroller with an MLI module is able to operate as a Local Controller (e.g. for data transmission) as well as a Remote Controller (e.g. for data reception) at the same time.

Transmitting and Receiving Controller

The terms “transmitting” and “receiving” controller are referring to the direction of the information flow. These terms are independent from the terms “Local” and “Remote”. For example, the initialization of a bidirectional MLI connection between two controllers (or between a controller and a stand-alone device) is always controlled and initiated by one controller (named Local), although during this phase, both MLI participants can transmit and receive frames.

Due to the full duplex operation capability of the MLI module (independent transmitter and receiver), each microcontroller with an MLI module is able to operate as a transmitting controller as well as a receiving controller at the same time.

Transfer Window

A Transfer Window is an address space in the address map of the transmitting controller. Transfer Windows are typically assigned to a fixed address space (base address and size). The Transfer Windows are the logical data inputs for the MLI transmitter. Data write actions via MLI are initiated by a write access to a Transfer Window, whereas data read actions are started by a read access from a Transfer Window.

Each MLI module supports up to four independent Transfer Windows, one for each pipe. In the implementation of a specific device, a Transfer Window can appear at several locations in the address map. Here, each Transfer Window can be accessed at two different address ranges with two different window sizes (one 64 Kbyte and one 8 Kbyte area for each Transfer Window), leading to:

- Four Small Transfer Windows STW with 8 Kbyte address range each and

- Four Large Transfer Windows LTW with 64 Kbyte address range each

Remote Window

A Remote Window is an area in the address space of the receiving controller. Remote Window parameters (base address and size) of the receiving controller are programmable by the transmitting microcontroller by MLI transfers, independently for each pipe. Each Remote Window of a receiving controller is related to specific Transfer Window of the transmitting controller.

The Remote Windows are the logical data outputs of the MLI receiver. If enabled, the MLI module can automatically execute the requested data transfer to/from the defined address location in the Remote Window. If the automatic data handling is disabled, the offset and the data are available in the MLI receiver registers and have to be handled by software. Remote windows can not be accessed by read or write accesses by software of the Remote Controller (either the data is automatically transferred or it is located in receiver registers).

Pipe

A pipe defines the logical connection between a Transfer Window in the transmitting controller and the associated Remote Window in the receiving controller. The MLI protocol supports four independent pipes.

Frame

A frame is a contiguous set of bits forming a message sent by an MLI transmitter to an MLI receiver.

A **Normal Frame** is a frame used for data exchange between a transmitting and a receiving controller (read request and write data from a Local Controller to a Remote Controller, as well as the answer to a read request back to the Local Controller). Base address copy frames are also considered as Normal Frames.

A **Command Frame** contains information about the receiver setting or triggers actions in the MLI receiver.

A **Triggered Command Frame** is generated under hardware control and can be used to transfer interrupt or service requests between the MLI participants.

Offset

The offset is an address distance relative to the base address of the Transfer Window in the transmitting controller and the base address of the Remote Window in the receiving controller. For example, a write access to the 10th byte of the Transfer Window is transferred to a write to the 10th byte of the Remote Window.

The offset of a write access to a Transfer Window is also called write offset, whereas a read offset is related to a read access from a Transfer Window.

21.1.1.3 MLI Communication Principles

The communication principle of the MLI modules allows data to be transferred between a Local and a Remote Controller without intervention of a CPU in the Remote Controller. Data transfers are always triggered in the Local Controller by read or write operations to an address location in a Transfer Window. All control tasks, address and data transmissions that are required for the data transfer/request between Local and Remote Controller can be handled autonomously by the two connected MLI modules.

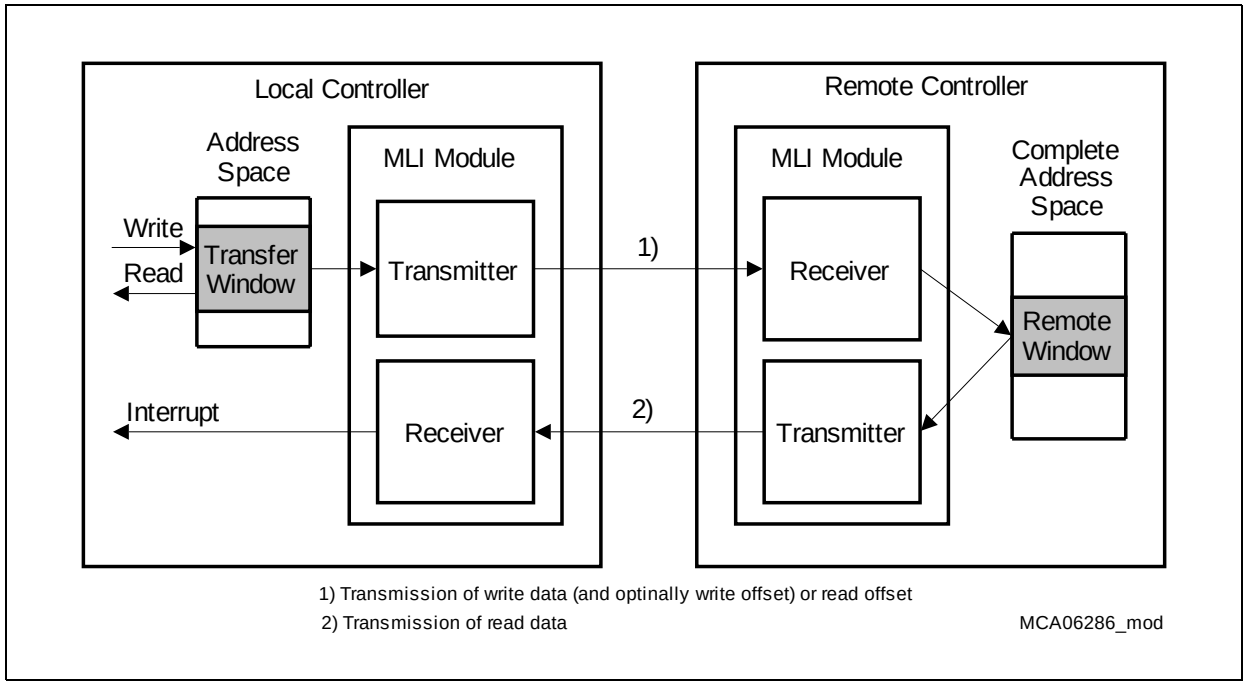


Figure 21-2 MLI Communication Principles

Write Access to a Transfer Window

A write access to a location within a Transfer Window of the transmitting (Local) controller is detected by the MLI transmitter. This detection initiates a transfer of the data that has been written to the Transfer Window together with the write offset to the MLI of the receiving controller. The receiving controller stores the data internally and can also automatically place the data in the Remote Window of the receiving controller (at the address location defined by the write offset plus the base address).

Read Access from a Transfer Window

A read access from a location of a Transfer Window in the Local Controller is detected by the MLI transmitter and delivers dummy data. This detection initiates a transfer of the read offset from the Local microcontroller to the MLI receiver to request data from the Remote Controller. This data can be automatically read or prepared by a CPU in the Remote Controller. When the requested data is available in the Remote Controller, it is

introduced into the data stream back to the Local Controller (Answer Frame). Then, the CPU in the Local Controller is informed by an MLI event that the requested data is now available and can be read.

Transfer Window Organization

Figure 21-3 shows an example of the organization of Transfer Windows and Remote Windows with a possible assignment in Local and Remote Controller. Each of the four pipes assigns one Transfer Window to one Remote Window with its base address and window size. For reasons of simplicity, a pipe to a Remote Window is only shown either from a LTW or from a STW, although each Transfer Window can be accessed at both address locations, its LTW and its STW.

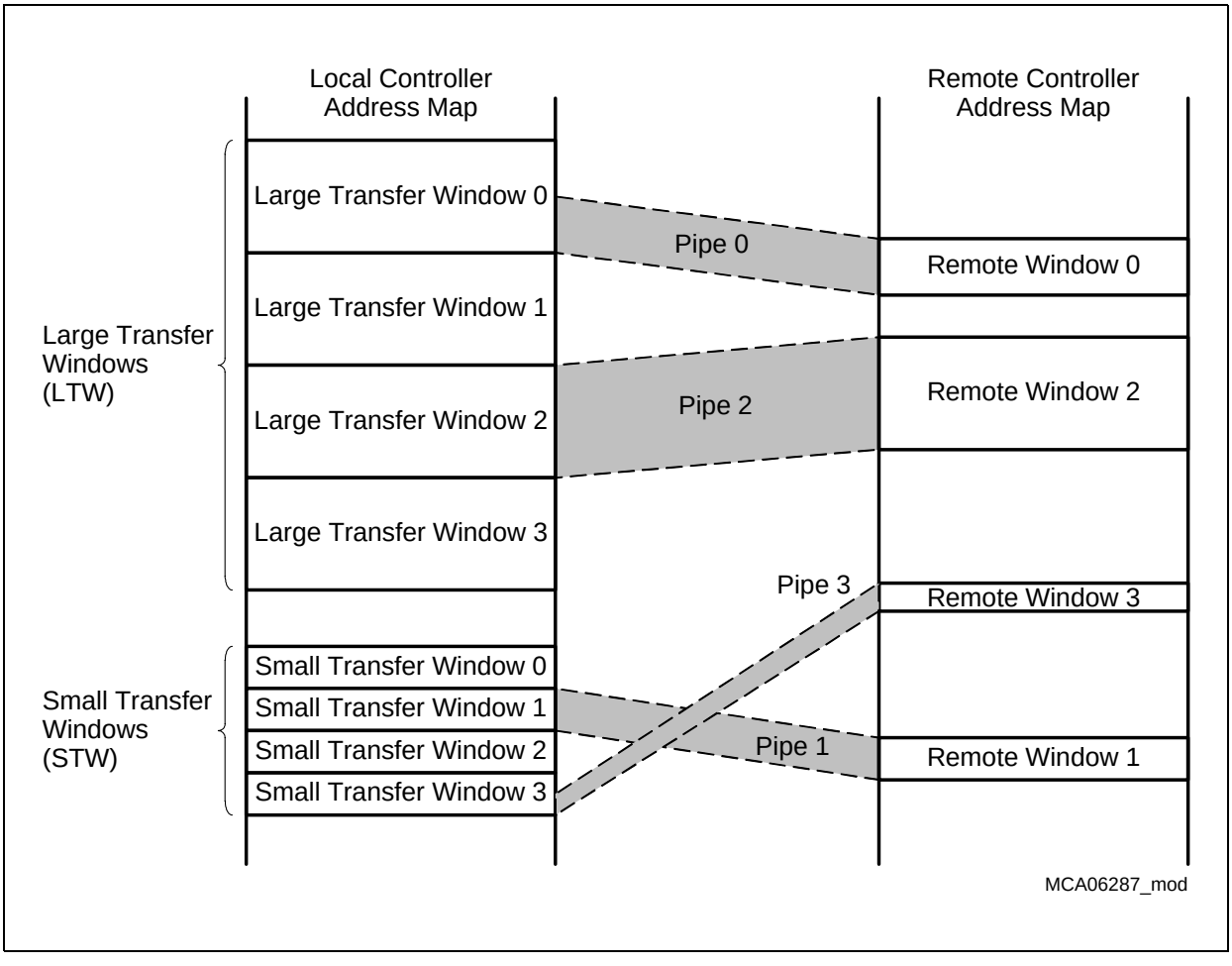


Figure 21-3 Transfer/Remote Window Assignment Example

During initialization of the pipes, base addresses and sizes of the Remote Windows are transmitted from the Local Controller to the Remote Controller. In the example of **Figure 21-3**, pipe 1 and pipe 2 cover the full range of their Transfer and Remote

Windows. The ranges of the Remote Windows of pipe 0 and pipe 3 are sub-ranges of the related Transfer Windows.

The location of a Transfer Window (base address and size) in the Local Controller is always fixed in a specific product device. Remote windows can be freely moved and located within the address space of the receiving controller. They are used to overlay address ranges of peripheral modules or internal memories.

Remote Window Address Generation

Figure 21-4 shows the generation of the Remote Window address ranges, with fixed base address part and additional variable address part. The variable address part is determined by the available address area for each Remote Window (also named buffer size, value of BSx = buffer size for Remote Window x indicates how many address bits are variable, defining the available address range).

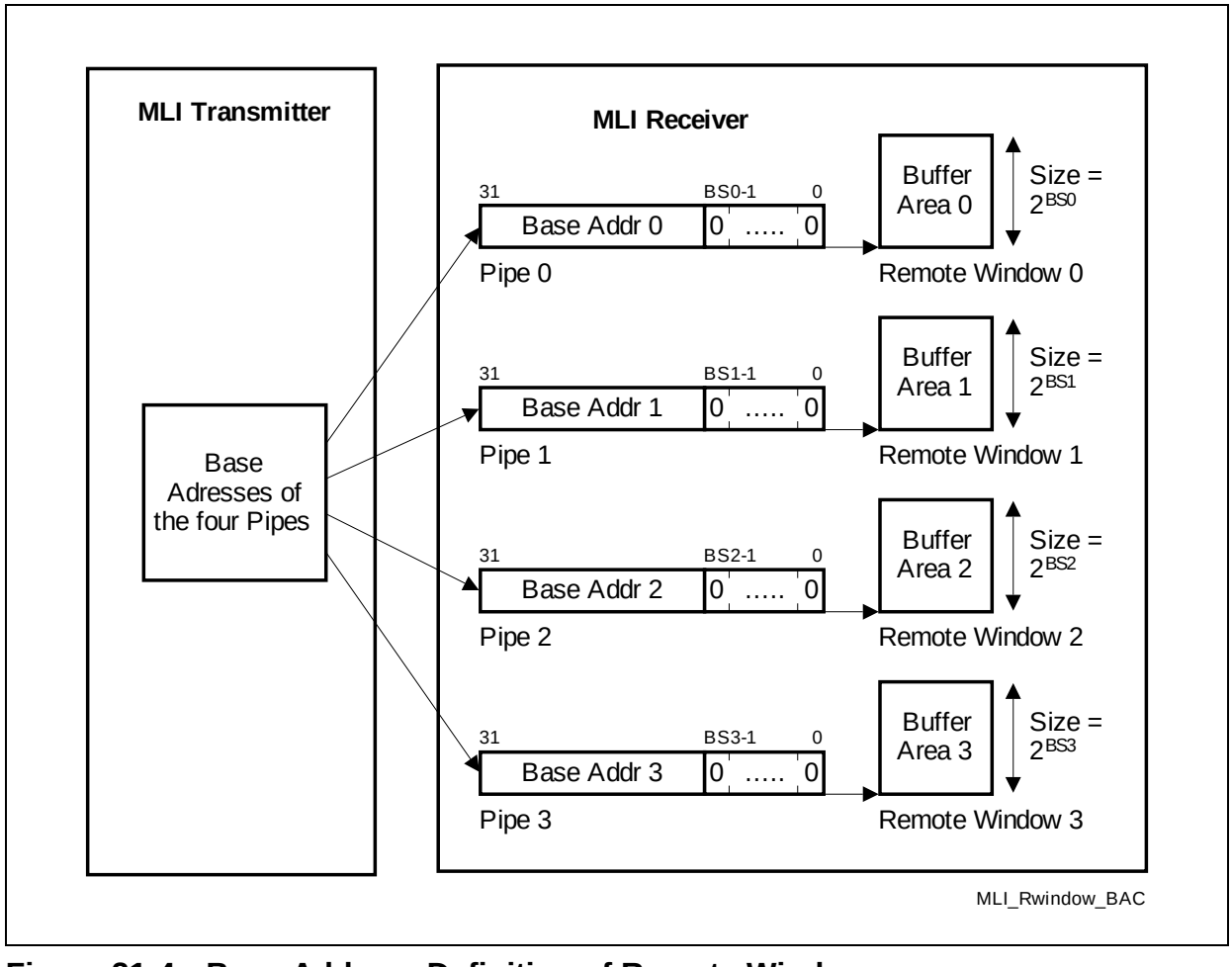


Figure 21-4 Base Address Definition of Remote Windows

Figure 21-5 shows the generation of the complete Remote Window address without address prediction. The variable address part can be transferred as offset by a write or a Read Frame, or it can be predicted in case of regular address modifications, whereas the fixed part of the address is defined by the upper bits of the base address.
In case of address prediction, the variable address part is internally calculated and taken as lower address bits of the target address (the upper address bits are given by the Remote Window's base address).

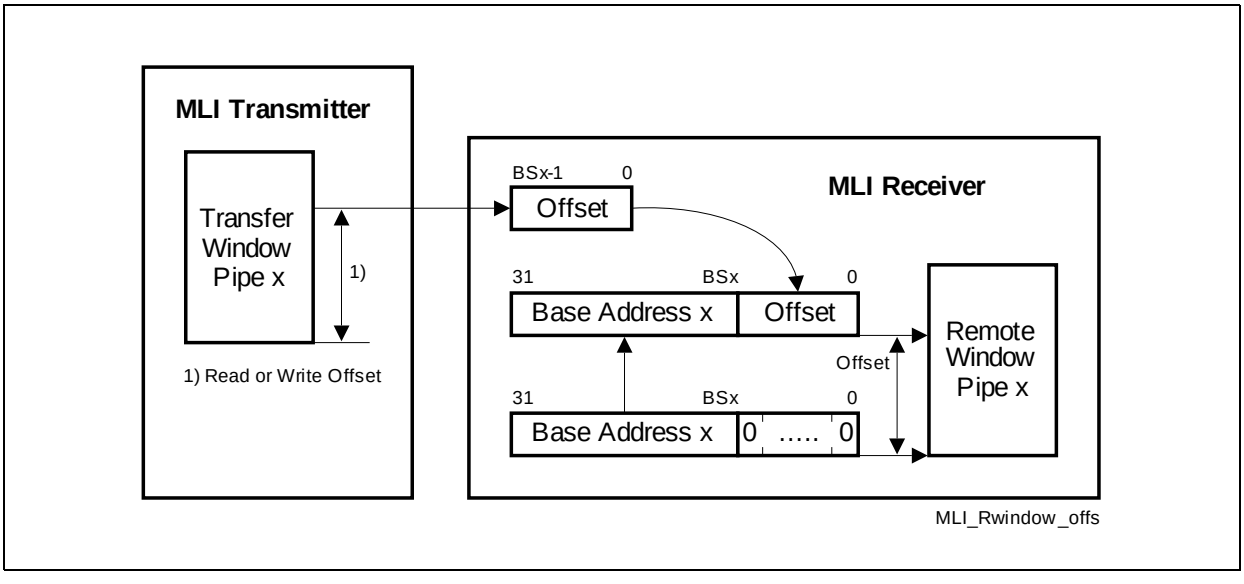


Figure 21-5 Remote Window Address Generation without Address Prediction

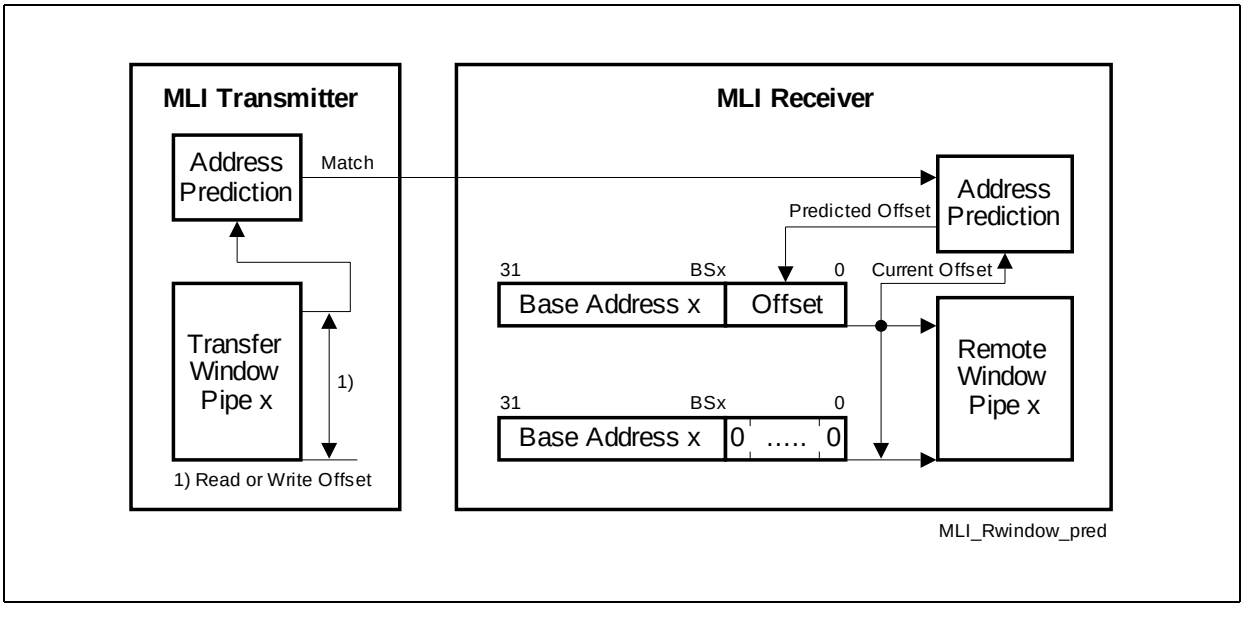


Figure 21-6 Remote Window Address Generation with Address Prediction

21.1.2 MLI Frame Structure

A frame is a message sent by an MLI transmitter to an MLI receiver. Depending on the desired behavior, different frame types exist:

- Copy Base Address Frame to define location and size of a Remote Window (see **Page 21-12**)
- Write Offset and Data Frame to transmit the write offset and the write data (see **Page 21-13**)
- Optimized Write Frame to transmit write data without write offset in case of an address prediction match (see **Page 21-14**)
- Discrete Read Frame to transmit read request with the read offset (see **Page 21-15**)
- Optimized Read Frame to transmit the read request without read offset in case of an address prediction match (see **Page 21-16**)
- Command Frame to transmit a command, e.g. setup information or MLI service request generation (see **Page 21-17**)
- Answer Frame to transmit the data previously requested by a Read Frame (see **Page 21-18**)

The local/remote structure of an MLI connection between two microcontrollers requires a transmitter unit and a receiver unit in both MLI modules (local and remote) for communication.

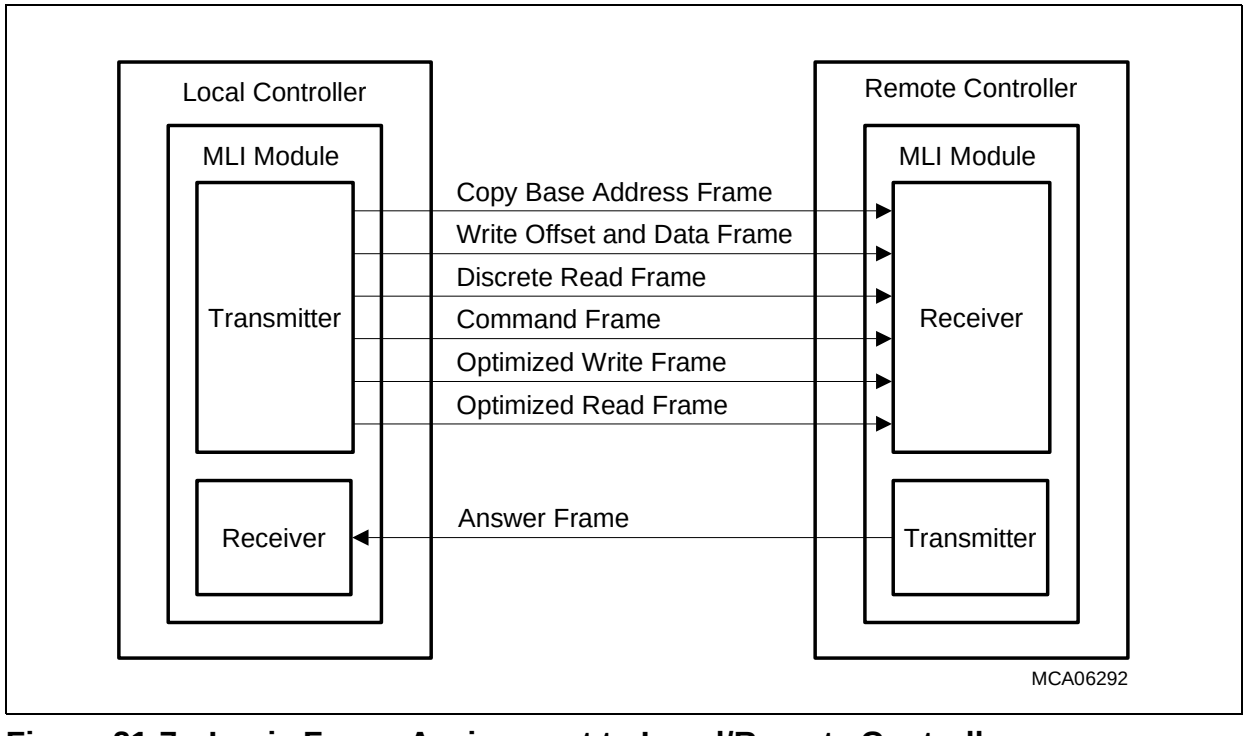


Figure 21-7 Logic Frame Assignment to Local/Remote Controller

21.1.2.1 General Frame Layout

The general layout of a frame is shown in **Figure 21-8**. It contains the following parts:

- A frame starts with a 4-bit header field that contains a 2-bit frame code (FC) and a 2-bit pipe number (PN).
- The data field can contain address, data, or control information. The width of the data field depends on the frame type.
- The frame is terminated by a parity bit (P) with even parity (see **Page 21-26**), calculated over header and data field bits.

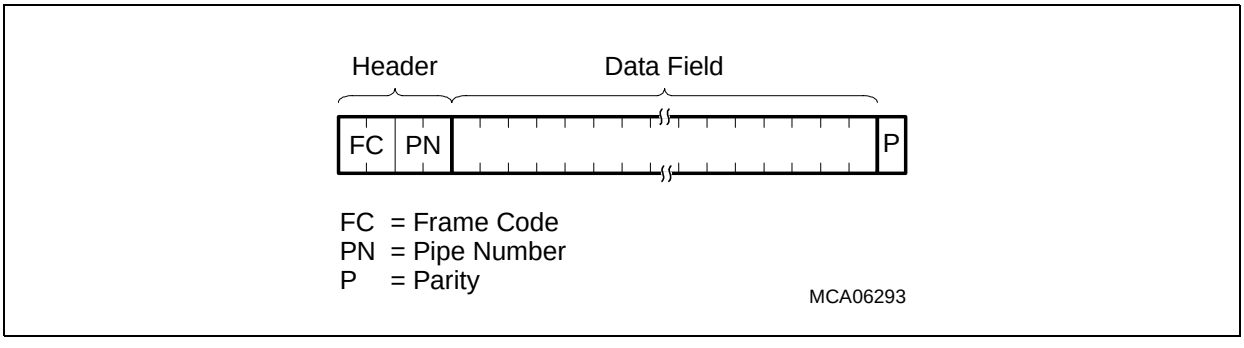


Figure 21-8 General Frame Layout

The frame code (FC) determines the frame type of the transmitted frame. The pipe number (PN) indicates the pipe that is related to the frame content (the value of PN is defined as 00_B for pipe 0, 01_B for pipe 1, 10_B for pipe 2, and 11_B for pipe 3).

The FC parameter is coded according to **Table 21-1**. If more than one frame type is defined with the same frame code value (see FC = 01_H, 10_H or 11_H), the width of the received frame defines the type. The value given by m in the table below represents the number of address bits transferred as offset (defined by the buffer size BSx of the Remote Window x).

Table 21-1 Frame Code Definition

Frame Code FC	Frame Type	Data Field Width [bits]	Description see
00_B	Copy Base Address Frame	32	**Page 21-12**
01_B	Write Offset and Data Frame	8+m, 16+m, or 32+m	**Page 21-13**
	Discrete Read Frame	2+m	**Page 21-15**
10_B	Command Frame	4	**Page 21-17**
	Answer Frame	8, 16, or 32	**Page 21-18**
11_B	Optimized Write Frame	8, 16, or 32	**Page 21-14**
	Optimized Read Frame	2	**Page 21-16**

21.1.2.2 Copy Base Address Frame

With a Copy Base Address Frame, the two parameters of a Remote Window are transferred from the transmitting controller to the receiving controller to initialize or to redirect the Remote Window.

The Copy Base Address Frame contains the following parts:

- Header:
  The header starts with frame code FC = 00_B followed by the pipe number PN of the pipe targeted by the transmitted base address bits and the size code.
- Remote Window address location:
  The 28 most significant bits of the 32-bit base address bits can be programmed by the transmitting controller (the 4 LSBs are considered as 0). The base address of a Remote Window has to be aligned to its size, e.g. a window of 1 Kbyte has to start at 1Kbyte address boundaries.
- Remote Window size:
  The size is defined by the 4-bit coded buffer size BS. The maximum size is 64 Kbytes.
- Parity bit P

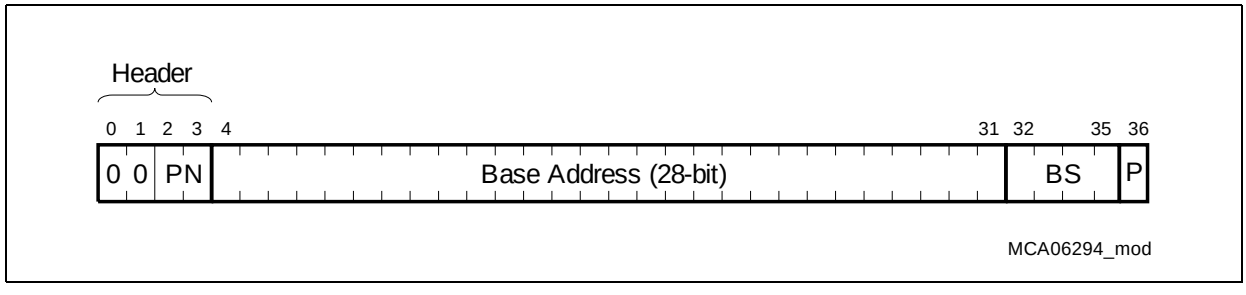


Figure 21-9 Copy Base Address Frame

Table 21-2 BS Coding

Buffer Size Code BS	Remote Window Size (also named Buffer Size)	Number m of Offset Bits
0000_B	2 bytes	m = 1
0001_B	4 bytes	m = 2
...	...	...
1110_B	32 Kbytes	m = 15
1111_B	64 Kbytes	m = 16

More details about the Copy Base Address Frame handling of the MLI module are described on **Page 21-28**.

21.1.2.3 Write Offset and Data Frame

A Write Offset and Data Frame is used by the transmitting controller to send an address offset and data to the receiving controller. This frame is initiated by a write operation to one of the Transfer Windows in the transmitting controller.

The Write Offset and Data Frame contains the following parts:

- Header:
  The header starts with frame code FC = 01_B followed by the pipe number PN of the Transfer Window that has been the target of the write operation.
- m-Bits of write offset:
  These bits define the write offset. The value of m depends on the size of the Remote Window, defined by the Copy Base Address Frame (m = 1-16).
- Write data field:
  The write data field can be 8-bit, 16-bit, or 32-bit wide, depending on the data width of the write access to the Transfer Window.
- Parity bit P

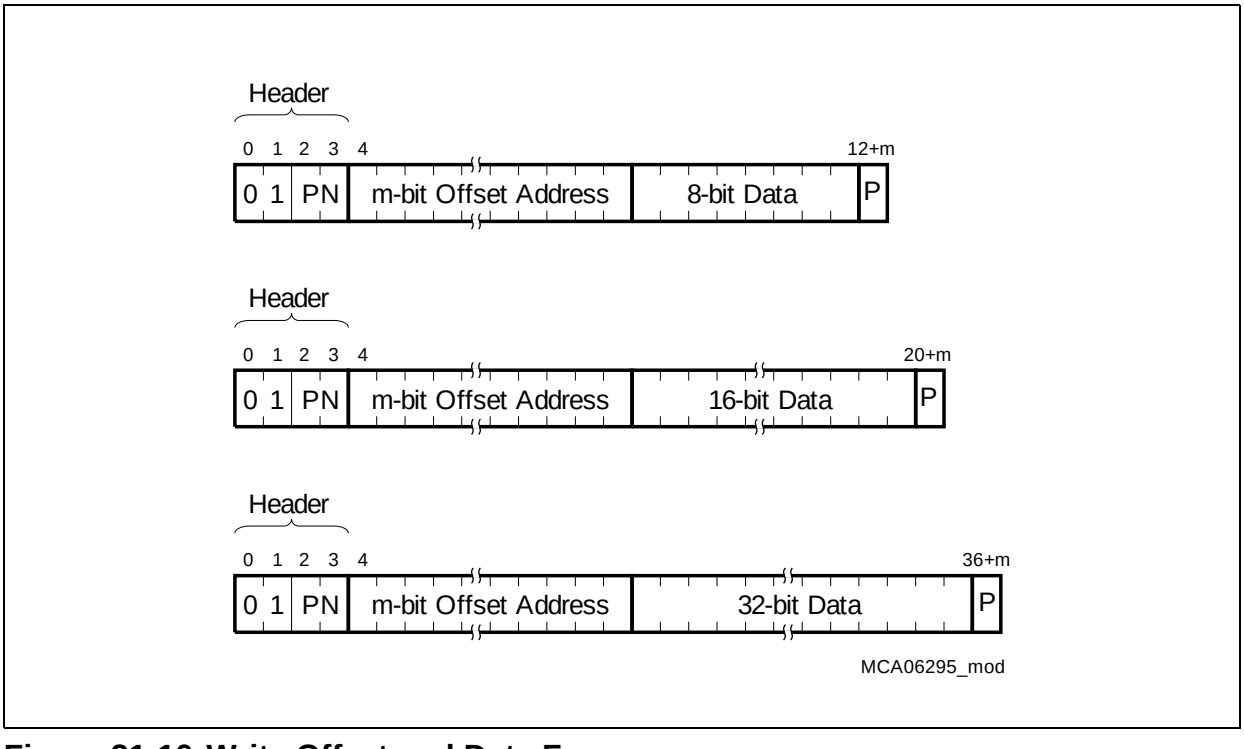


Figure 21-10 Write Offset and Data Frame

More details about the Write Offset and Data Frame handling of the MLI module are provided on **Page 21-30**.

21.1.2.4 Optimized Write Frame

An Optimized Write Frame is used by the transmitting controller to send 8-bit, 16-bit, or 32-bit wide data to the receiving controller. This frame is initiated by a write operation to one of the Transfer Windows in the transmitting controller. In contrast to a Write Offset and Data Frame, no write offset is transmitted because the offset address for the write data can be predicted and calculated by the receiving controller. An Optimized Write Frame allows a higher data bandwidth than Write Offset and Data Frames, because they are shorter. An optimized frame is only possible if the predicted address matches with the actually written one.

The Optimized Write Frame contains the following parts:

- Header:
  The header starts with frame code FC = 11_B followed by the pipe number PN of the Transfer Window that has been the target of the write operation.
- Write data field:
  The write data field can be 8-bit, 16-bit, or 32-bit wide, depending on the data width of the write access to the Transfer Window.
- Parity bit P

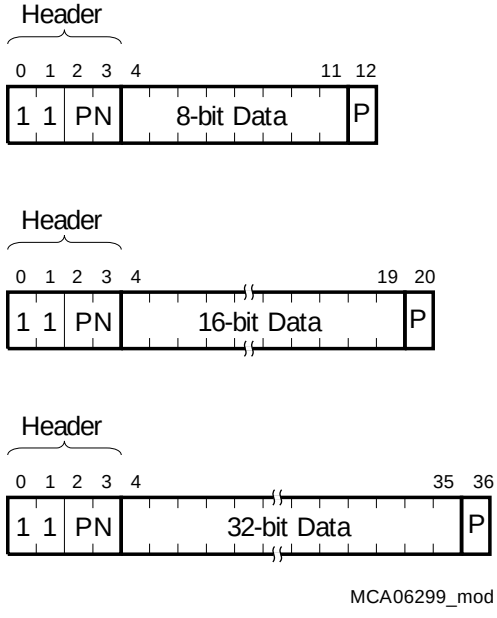


Figure 21-11 Optimized Write Frame

More details about the Optimized Write Frame handling of the MLI module are provided on **Page 21-30**.

21.1.2.5 Discrete Read Frame

A Discrete Read Frame is used by the Local Controller to request data to be read from the Remote Window in the Remote Controller. If the data is available, the Remote Controller typically responds to this request by sending an Answer Frame with the requested read data back to the Local Controller.

The Discrete Read Frame contains the following parts:

- Header:
  The header starts with frame code FC = 01_B followed by the pipe number PN of the Transfer Window that has been the target of the read operation.
- m-Bits of write offset:
  These bits define the read offset. The value of m depends on the size of the Remote Window, defined by the Copy Base Address Frame (m = 1-16).
- Data Width DW:
  The data width DW indicates if the read from the Transfer Window was a 8-bit, 16-bit, or 32-bit read action. It defines how many bytes have to be delivered to the Local Controller by the Answer Frame.
- Parity bit P

Figure 21-12 Discrete Read Frame

Table 21-3 Data Width DW Coding

Data Width DW	Number of Data Bits to be transferred
00_B	8-bit read access
01_B	16-bit read access
10_B	32-bit read access
11_B	reserved for future use

More details about the Discrete Read Frame handling of the MLI module are provided on **Page 21-34**.

21.1.2.6 Optimized Read Frame

An Optimized Read Frame is used by the Local Controller to request 8-bit, 16-bit, or 32-bit wide data from the Remote Controller without sending any offset address. The address for the requested data can be predicted and calculated by the MLI receiver of the Remote Controller.

The Optimized Read Frame contains the following parts:

- Header:
  The header starts with frame code FC = 11_B followed by the pipe number PN of the Transfer Window that has been the target of the read operation.
- Data Width DW:
  The data width DW indicates if the read from the Transfer Window was a 8-bit, 16-bit, or 32-bit read action. It defines how many bytes have to be delivered to the Local Controller by the Answer Frame. Same coding as for the Discrete Read Frame.
- Parity bit P

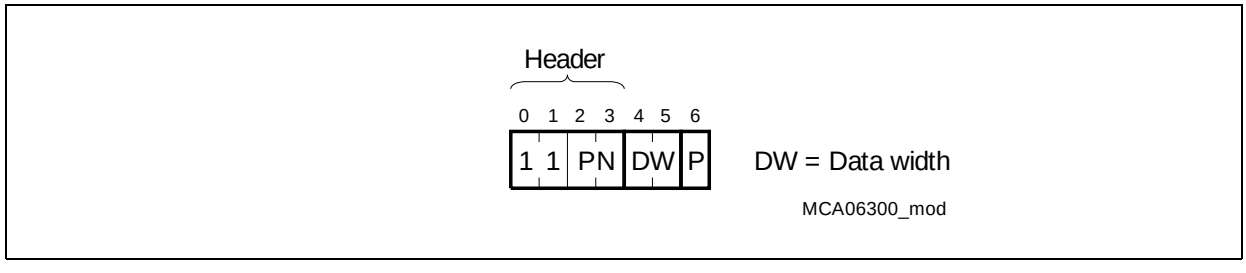


Figure 21-13 Optimized Read Frame

More details about the Optimized Read Frame handling of the MLI module are provided on **Page 21-30**.

21.1.2.7 Command Frame

The transmitting controller is able to initiate control actions to be executed by the receiving controller by sending a Command Frame.

The Command Frame contains the following parts:

- Header:
  The header starts with frame code FC = 10_B followed by the pipe number PN. The pipe number defines the type of command to be executed.
- Command Code CMD:
  Pipe number PN and a 4-bit CMD field are used for command coding. The command coding of some control actions is fixed, but free programmable software commands can also be defined (with PN = 11_B). The coding of the command bit field is pipe-specific and depends on the transmitted pipe number x.
- Parity bit P

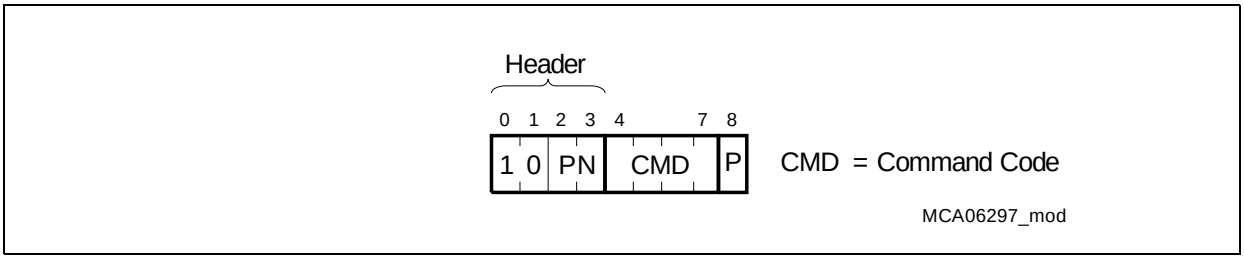


Figure 21-14 Command Frame

Table 21-4 PN for Command Coding

Pipe Number PN	Command Type
00_B	Activate MLI service request or other control signal(s) of the receiving controller. The definition which signal becomes activated is defined by CMD. The usage of these lines depends on the implementation.
01_B	Define delay for parity error indication in the receiving controller. The delay in RCLK cycles is defined by the value of CMD.
10_B	Control of internal functions of the receiving controller. The value of CMD indicates which function is controlled. The coding of CMD and the control mechanisms depend on the implementation.
11_B	Freely programmable software command.

More details about the Command Frame handling of the MLI module are provided on **Page 21-41**.

21.1.2.8 Answer Frame

An Answer Frame is used by the Remote Controller to send 8-bit, 16-bit, or 32-bit wide data to the Local Controller. The Answer Frame is the only frame that is transmitted within a logic Local/Remote Controller assignment from the Remote Controller to the Local Controller. It is the answer to a Discrete Read Frame or an Optimized Read Frame that has been sent by the Local Controller to request data from the Remote Controller.

The Answer Frame contains the following parts:

- Header:
  The header starts with frame code FC = 10_B followed by the pipe number PN. The value of PN is taken from the Read Frame that has triggered the Answer Frame.
- Read data field:
  The read data field can be 8-bit, 16-bit, or 32-bit wide, depending on the data width requested by the Read Frame that triggered the Answer Frame.
- Parity bit P

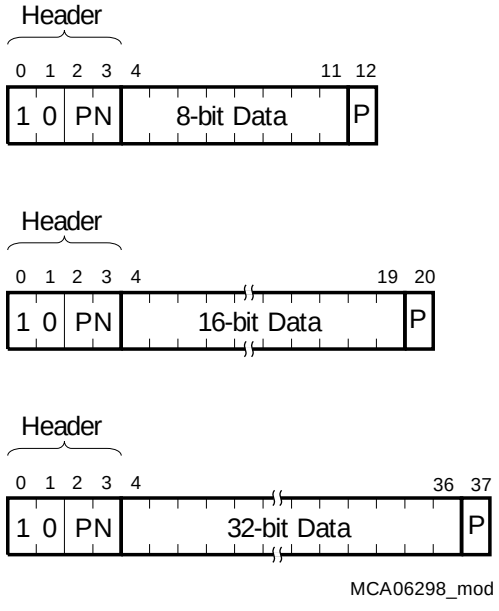


Figure 21-15 Answer Frame

More details about the Answer Frame handling of the MLI module are provided on **Page 21-39**.

21.1.3 Handshake Description

The description of the transmitter/receiver signal handshaking refers to an MLI connection between an MLI transmitter and an MLI receiver. MLI module transmitter I/O signals are indicated with prefix “T” and MLI receiver I/O signals are indicated with the prefix “R”. The 4-line MLI bus between a transmitter and a receiver outside the controllers uses signal names without any prefix.
In order to lay emphasis where a signal is generated or sampled, actions taken by the transmitter are described referring to signals with the prefix “T”, whereas receiver actions are referring to signals with the prefix “R”.

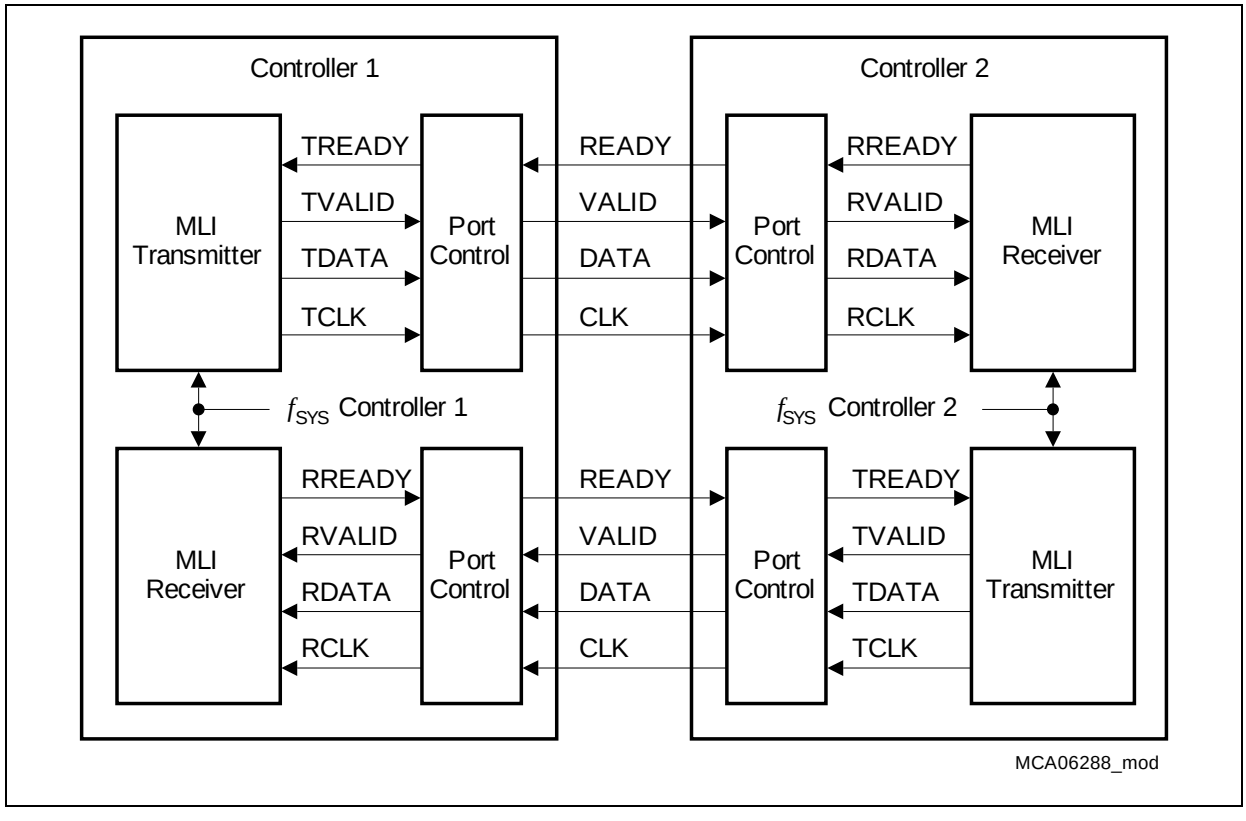


Figure 21-16 Transmitter/Receiver Signal Definitions

The MLI connection allows high data rates and, at the same time, supports significant signal propagation delays between the transmitter and the receiver. As shown in **Figure 21-16**, each output signal passes through the port stage, reaches the physical interface line between the MLI modules, enters via an input stage and can be finally evaluated. All these steps introduce an accumulating propagation delay. In standard synchronous serial connections (such as SPI), this delay limits the reachable baud rate to a few Mbit/s (closed-loop delay problem). In order to support higher baud rates than a standard SPI, the MLI protocol is based on a full handshake (READY-VALID) to deal with propagation delays in the range of some shift clock cycles and to avoid the closed-loop delay limitations of an SPI connection.

In a full handshake, each edge of the handshake signals has a defined meaning and the sequence of edges is clearly specified.
As a result, the propagation delays do not directly limit the MLI baud rate. Therefore, the points in time when a signal is generated, when it is visible on the physical interface line, or when it is evaluated have to be considered independently. This is done by defining 3 different names for a signal, referring to the 3 significant locations:

- The place where it is generated, also in relation to the generation clock edge
- The physical interface line where it can be observed
- The place where it is evaluated, also in relation to the evaluation clock edge

If a Local Controller should be connected to more than one Remote Controller, the transmitter signals CLK and DATA can be used as broadcast signals (parallel connection to the Remote Controllers), whereas the handshake signals VALID and READY have to be established as independent signal pairs for each device. As a result, a Local Controller only needs one CLK and one DATA output, but an individual set of READY and VALID handshake signals for each Remote Controller. Please note that Read Frames and Answer Frames are based on an established connection between a Local and a Remote Controller (because the Answer Frame is the only frame sent back to the Local Controller). Therefore, switching between several Remote Controllers can only be done while no read request is pending in the Local Controller. If no Read Frames are used by the Local Controller, frames can be sent out in parallel to all Remote Controllers if their READY signals are all respected.
If a Remote Controller should be connected to several Local Controllers, it may have several DATA and CLK inputs in addition to the READY-VALID signal sets. Please note that an active switching of a Remote Controller between several Local Controllers requires that all Local Controllers have the information which connection is active.
In any case, switching between Local and Remote Controllers is not allowed while frame transmission is in progress.

21.1.3.1 Handshake Signals

The synchronous serial frame transfer from an MLI transmitter to an MLI receiver is based on the following 4 signals (the MLI protocol only defines the signal transitions, but neither the signaling level nor the driver characteristics):

- Shift clock CLK:
  This signal is used as serial shift clock that is generated by the transmitter during the complete frame transfer (TVALID is active) and until the end of ready delay time. Signal TCLK can also be generated while no frame is transferred. In this case, the receiving controller can use the incoming RCLK receiver signal as base for its internal clock generation.
  The transmitter signals are always referring to the rising edge of TCLK, so TREADY is sampled and the output signals TDATA and TVALID are changing with the rising edge of TCLK.
  The MLI receiver actions refer to the falling edges of its RCLK input. The receiver samples the RVALID and RDATA signals and outputs its RREADY line with the falling edges of RCLK.
- Shift data DATA:
  This signal represents the transmit data TDATA transferred from the MLI transmitter to the MLI receiver input RDATA. Changes on transmitter side take place with rising edges of TCLK, whereas sampling on the receiver side takes place with falling edges of RCLK.
- Transmitter valid handshake VALID:
  This signal indicates the start and the end of each frame. It is active (1-level) during a frame transmission and passive (0-level) while no frame is transferred. Changes of TVALID on transmitter side take place with rising edges of TCLK, whereas sampling of RVALID on the receiver side takes place with falling edges of RCLK.
  An activation of TVALID to start a new frame can only take place if TREADY is 1.
- Receiver ready handshake READY:
  This signal indicates that the receiver is ready for a data transfer. Additionally, this line is used to indicate reception errors (parity error indication). Changes of RREADY on receiver side take place with falling edges of RCLK, whereas sampling of TREADY on the transmitter side takes place with rising edges of TCLK.

21.1.3.2 Error-free Handshake

A transmission can be started by an MLI transmitter when the MLI receiver is ready to receive data indicated by RREADY = 1 by the receiver. When the MLI transmitter detects TREADY = 1 and starts its transmission, TVALID is asserted to 1 level while a frame transfer is in progress. When the MLI receiver has detected the 0-to-1 transition of the RVALID signal it will de-assert RREADY back to 0 (transmission start acknowledged by receiver). At the end of the frame transmission, the MLI transmitter also de-asserts signal TVALID back to 0 and checks if the TREADY signal is at 0 level,

too. This check is used as life-sign of the receiver and the MLI transmitter can detect whether the receiver is able to react in-time to the transmitter actions (see also Page 21-23).

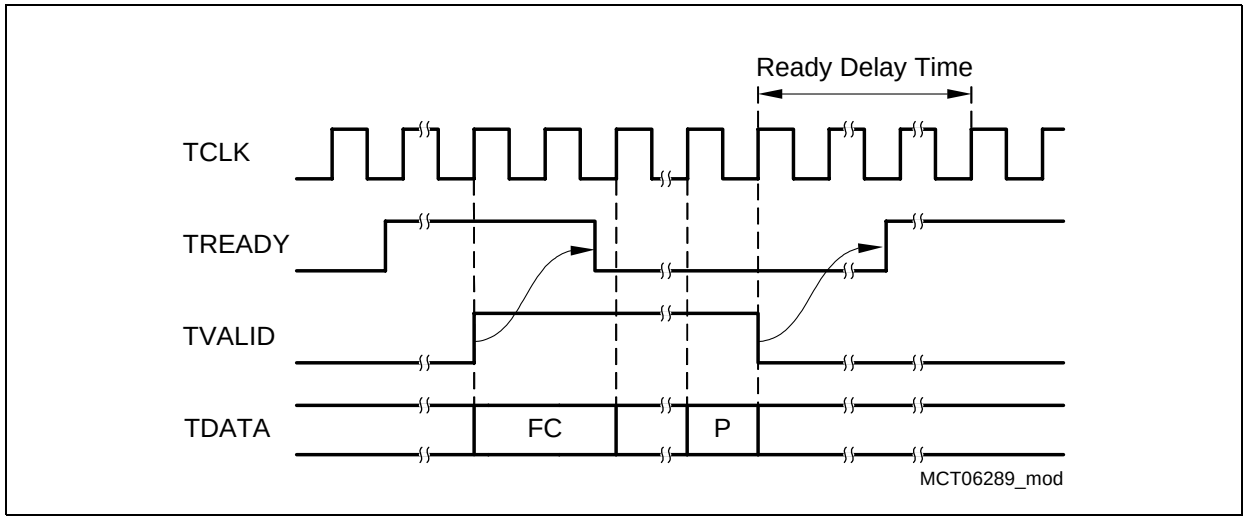


Figure 21-17 MLI Handshake without Error Indication

21.1.3.3 Ready Delay Time

In order to support significant propagation delays, the handshake signal TREADY is evaluated with respect to TVALID and TCLK in an time interval called Ready Delay Time after the end of the frame (see **Figure 21-17**). The length of the Ready Delay Time is programmable, defining the size of the time interval.

When a transmission is finished (RVALID becomes 0), the MLI receiver checks the received frame for correct reception (parity error). If no parity error has been detected, the MLI receiver asserts its RREADY signal again to 1 to indicate the correct reception with the next falling edge of RCLK. The MLI transmitter checks its TREADY input with each rising edge of TCLK after TVALID has become 0 and increments a counter. This counter is started from 0 at the end of a frame transmission (TVALID becomes 0) and counts TCLK periods (Ready Delay Time Counter). If the condition TREADY = 1 is detected before the programmed Ready Delay Time has elapsed, the MLI receiver has indicated a frame reception without parity error to the MLI transmitter. In this case, a new frame transmission can be started. The transfer handshake signalling without a parity error indication is shown in **Figure 21-17**.

Figure 21-18 shows the transfer handshake if a parity error condition has been detected by the MLI receiver and indicated to the MLI transmitter. In this case, the receiver waits a programmable number of RCLK clock cycles before setting RREADY to 1. If the TREADY = 1 condition is detected by the transmitter after the ready delay has elapsed, a parity error has been indicated by the MLI receiver. In this case, it is assumed that the MLI receiver has detected a frame with a parity error and has discarded the frame. The

transmitter automatically sends the last frame again after a parity error indication. Optionally, this MLI event can activate a service request output.

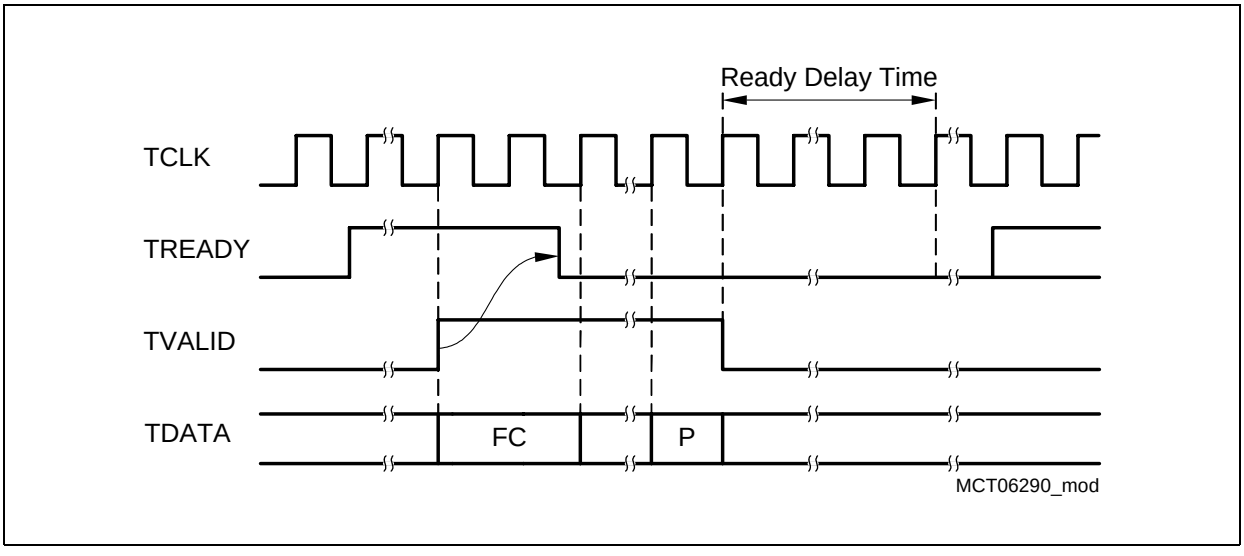


Figure 21-18 MLI Handshake with Parity Error Indication

21.1.3.4 Non-Acknowledge Error

A transmitter of an MLI module is able to detect an inoperable receiver by analyzing the handshake signal TREADY. After TVALID has been asserted to 1, the transmitter checks the receiver's acknowledge (TREADY becoming 0). A Non-Acknowledge error condition is detected by the transmitter when at the end of a frame transmission the TREADY signal is still at high level (TREADY = 1 when TVALID becomes 0). **Figure 21-19** shows the Non-Acknowledge error case. In this case, the transmitter automatically sends the last frame again. Optionally, this MLI event can activate a service request output.

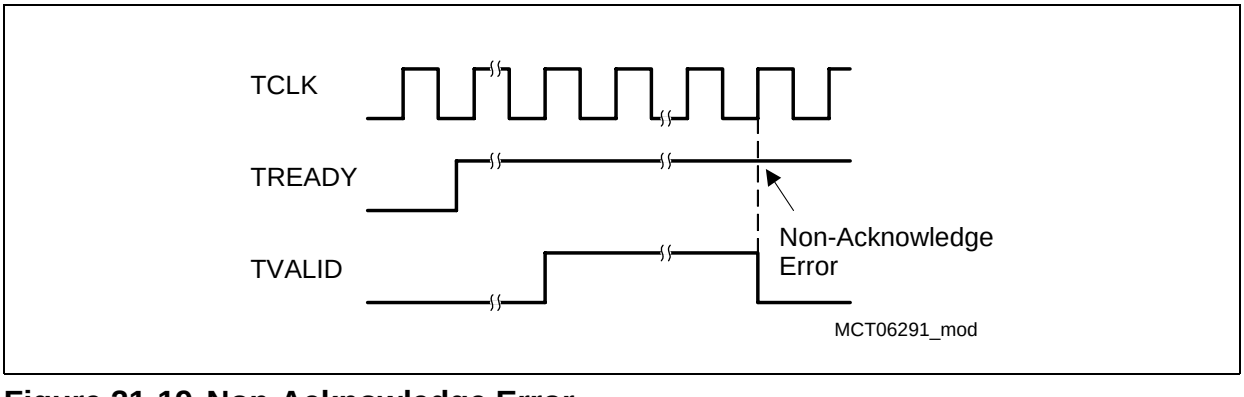


Figure 21-19 Non-Acknowledge Error

21.1.3.5 Signal Timing

Figure 21-20 shows the MLI timing requirements.

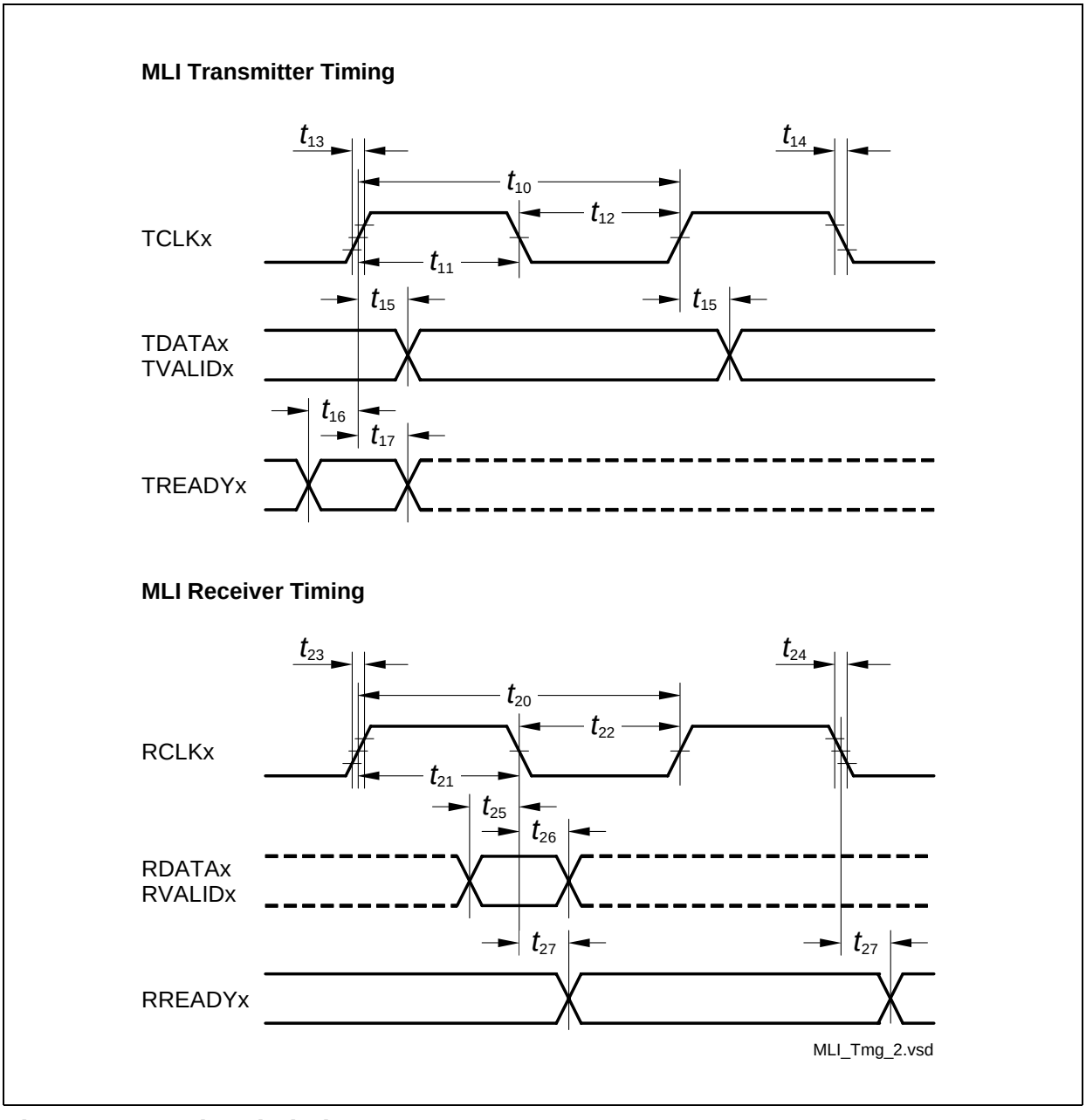


Figure 21-20 Signal Timing

The transmitter output signals TDATA and TVALIDx might have a certain delay to the transmitter clock output TCLK due to on-chip variation of the driver stages and differences in the propagation delays. The transmitter TREADY input can change at any point in time compared to TCLK. In order to ensure stability, it is internally synchronized to f_{MLI} of the transmitter before being evaluated with the rising TCLK edge when TVALID becomes 0. For the calculation of the signal propagation time, these 2 clock cycles have to be taken into account.

The transmitter input TREADYx has to be stable a certain time before TVALID becomes low, referring to the rising edge of TCLK when TVALID becomes low. If at this point in time, TREADYx is detected at a high level, a Non-Acknowledge error is signaled. The same timing relation has to be considered at the end of the ready delay time for the parity error detection.

The receiver input signals are handled asynchronously based on the RCLK signal. The synchronization to the receiver's system clock f_{SYS} is done in the receiver logic. The input signals RDATA and RVALID have to respect a certain setup and hold time at the falling edge of RCLK.

21.1.4 Parity Generation

For parity generation, the number of transmitted bits with the value of 1 is counted over the header and the complete data field of a frame. For even parity, the parity bit is set if the result of a modulo-2 division of the elaborated number is 1. For error-free MLI traffic, even parity generation and checking is defined.

More details about the parity handling of the MLI module are provided on **Page 21-44**.

21.1.5 Address Prediction

An address prediction method can be enabled to support communication between MLI transmitter and MLI receiver without sending address offset information in the frames. This feature reduces the required bandwidth for MLI communication. Both of the communication partners, MLI transmitter and receiver are able to detect regular offset differences of consecutive window accesses to the same window. The address prediction mechanism working independently for each pipe, different prediction values can be handled in parallel for the different pipes.

The MLI transmitter can compare the offset of each Transfer Window read or write access with the offset of the previous access to the same Transfer Window. Between the accesses to a specific window, other windows can be accessed without disturbing the prediction. Bigger offset differences than 512 bytes are not supported by the address prediction.

If the offset differences are identical in at least two accesses to the same Transfer Window, an address prediction is possible and Optimized Write Frames or Optimized Read Frames can be sent to the receiving controller for this pipe. If the offset difference of a next access to this Transfer Window does not match the former ones (predicted offset), address prediction is not possible. In this case, a Normal Frame for writing or reading (Write Offset and Data Frame or Discrete Read Frame) is started.
The identical address prediction mechanism is built in the receiver. As a result, the receiver can elaborate the original offset value in the transmitter when receiving an optimized frame for any pipe.

More details about the address prediction mechanism of the MLI module are provided on **Page 21-47**.

21.2 Module Kernel Description

This chapter describes how the MLI protocol is implemented in the MLI module and how frame handling can be done by software, comprising:

- The frame handling (see **Page 21-27**)
- The general MLI features (see **Page 21-44**)
- The interface signals (see **Page 21-51**)
- The general MLI service request structure (see **Page 21-57**)
- The MLI transmitter events (see **Page 21-59**)
- The MLI receiver events (see **Page 21-62**)
- The baud rate generation (see **Page 21-67**)

21.2.1 Frame Handling

The frame handling is based on receiver and transmitter registers and the Transfer Windows. Depending on the type of access to the Transfer Windows, different actions take place inside the MLI module. Please refer to the following pages for the handling of:

- Copy Base Address Frame (see **Page 21-28**)
- Data Frames (see **Page 21-30**)
- Read Frames (see **Page 21-34**)
- Answer Frame (see **Page 21-39**)
- Command Frame (see **Page 21-41**)

21.2.1.1 Copy Base Address Frame

A Copy Base Address Frame defines the location and the size of a Remote Window.

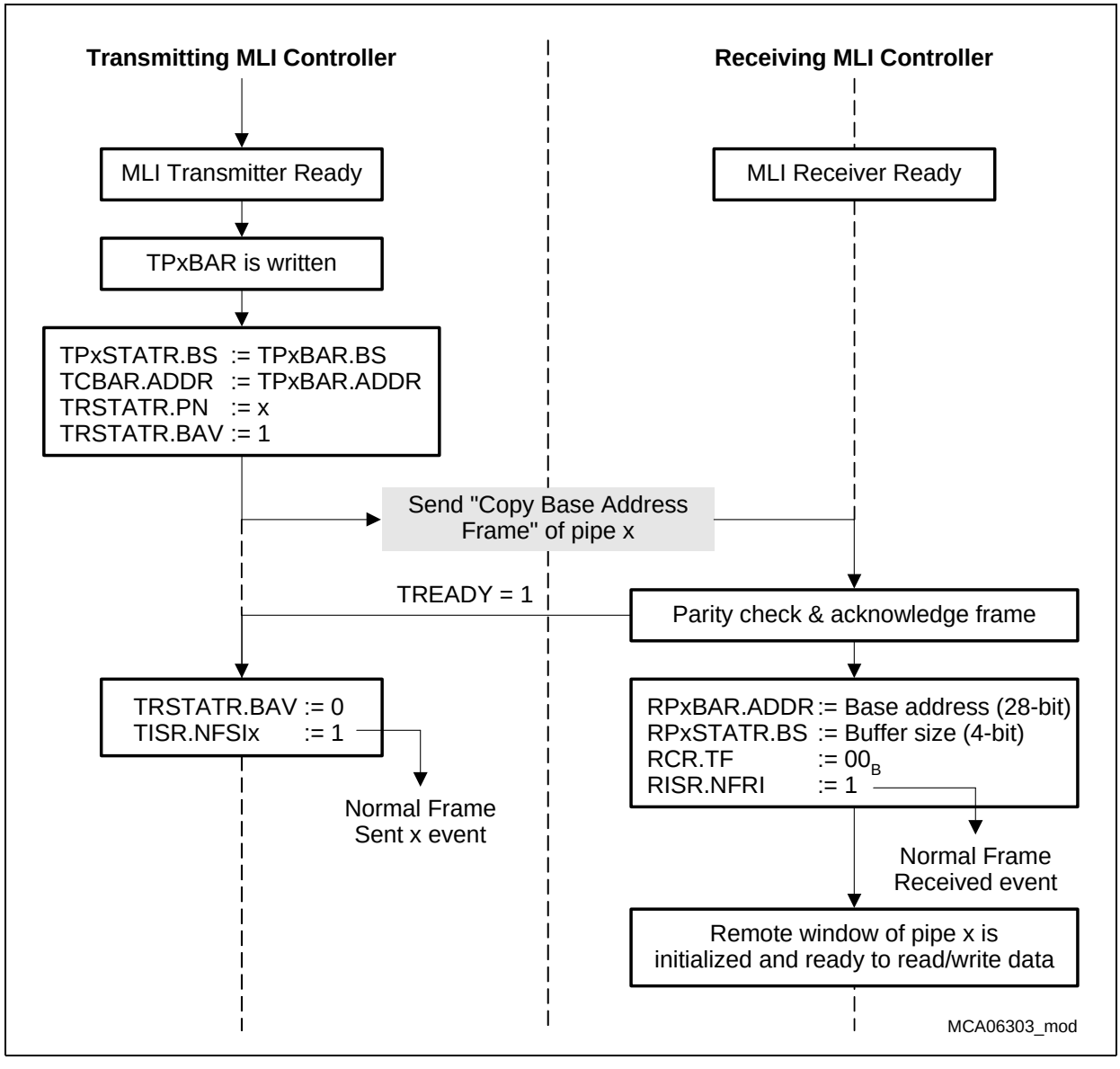


Figure 21-21 Copy Base Address Frame Flow

Transmitting Controller

The transmission of a Copy Base Address Frame is started after a transmitter pipe x base address registers TPxBAR has been written, triggering the following actions for pipe x.

- Bit field TPxBAR.BS (4-bit coded buffer size) is loaded into bit field TPxSTATR.BS
- Bit field TPxBAR.ADDR (28 most significant base address bits) is loaded into bit field TCBAR.ADDR.

- Status bit field TRSTATR.PN is updated with the pipe number x (for example x = 2 when TP2BAR has been written).
- Status flag TRSTATR.BAV (base address valid) becomes set.
- The transmission of a Copy Base Address Frame with the two buffered parameters TCBAR.ADDR and TPxSTATR.BS is started for pipe x (if the corresponding pipe is idle and TREADY = 1).
- Status flag TRSTATR.BAV (in the transmitting controller) is cleared after the Copy Base Address Frame has been finished and correctly acknowledged by the MLI receiver of the receiving controller.
- MLI event status flag TISR.NFSIx (Normal Frame Sent event in pipe x) is set and a service request output is activated if enabled by TIER.NFSIEx = 1.

Note: After the transfer of a Copy Base Address Frame the optimized mode will be suppressed automatically by hardware for the next two data frames. This ensures a correct offset prediction afterwards.

Receiving Controller

When a Copy Base Address Frame for pipe x has been received correctly and acknowledged, the following actions are executed in the MLI receiver.

- The received 28 most significant address bits are written into the receiver pipe x base address register bit field RPxBAR.ADDR. This bit field determines the base address of the pipe x Remote Window.
- The received 4-bit coded buffer size is stored in the receiver pipe x status register bit field RPxSTATR.BS. This bit field determines the number of variable address bits for the offset (determining the size) of the pipe x Remote Window.
- The information about the received frame type (= 00_B for Copy Base Address Frame) is written into the receiver control register bit field RCR.TF.
- MLI event status flag RISR.NFRI (Normal Frame Received event) is set and a service request output is activated if enabled by RIER.NFRIE = 01_B or 10_B.

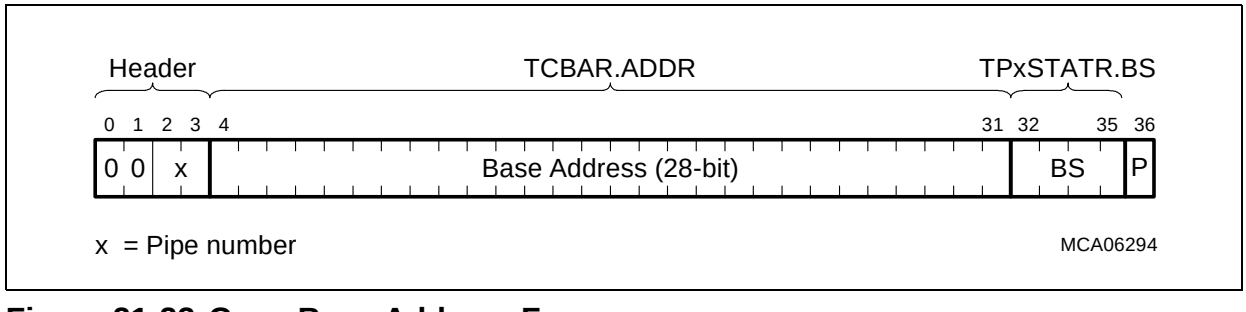


Figure 21-22 Copy Base Address Frame

21.2.1.2 Write/Data Frames

Write Frames (also named Data Frames) transmit the write data and optionally the write offset.

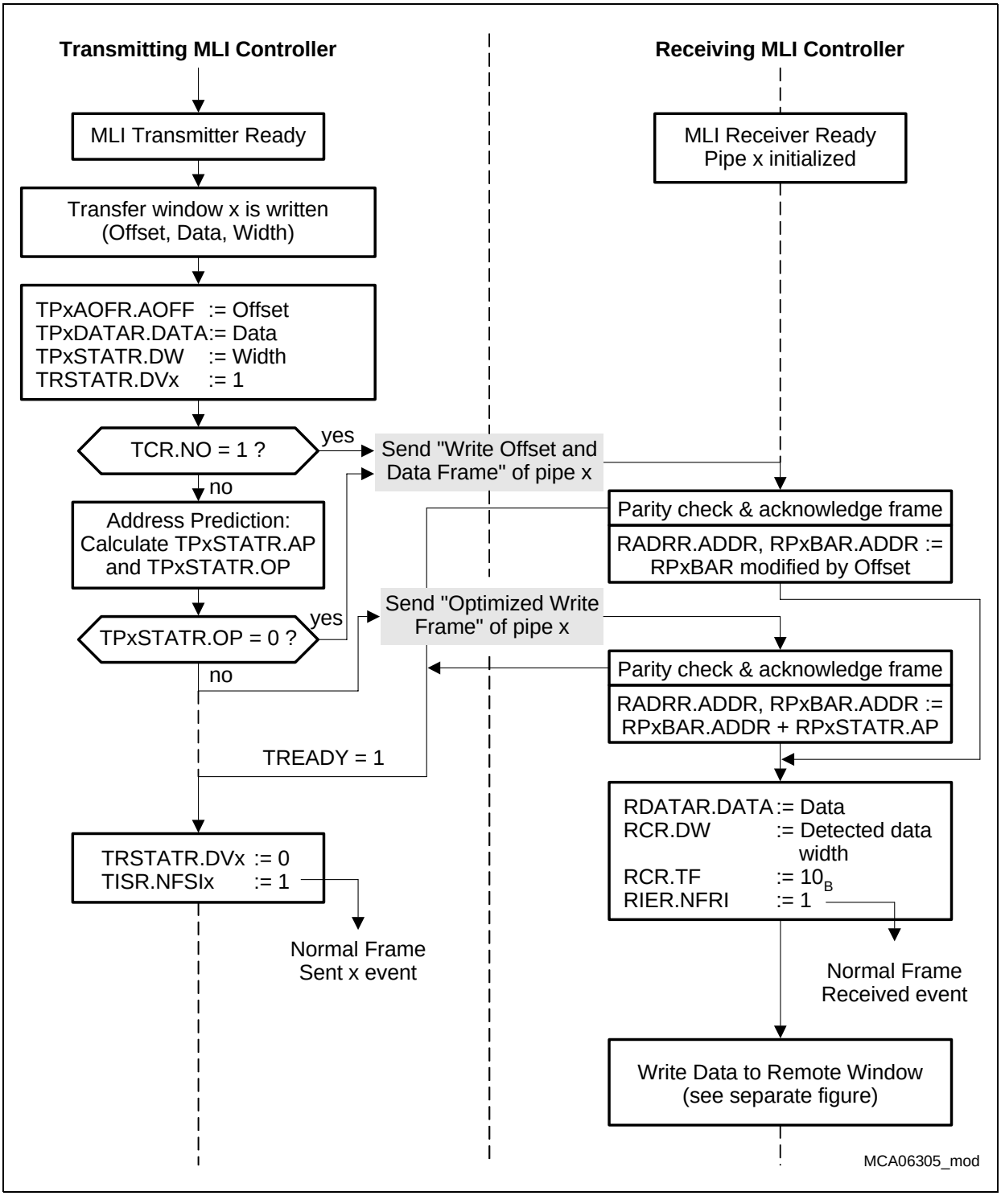


Figure 21-23 Write Frame Flow

Transmitting Controller

In the transmitting controller, a write operation to a location within a Transfer Window delivers the address, the data, and the data size to the transmitter and triggers the following actions in the MLI transmitter.

- The 16 least significant address bits of the Transfer Window write access are stored in TPxAOFR.AOFF as write offset address. In case of a an access to a Small Transfer Window, also 16 bits are stored, but the higher bits are not taken into account assuming the buffer size is configured correctly (see **Page 21-105**).
- The data of the write access to the Transfer Window is stored in TPxDATAR.DATA.
- The data width of the write access to the Transfer Window (8-bit, 16-bit, or 32-bit) is stored in bit field TPxSTATR.DW.
- Status flag TRSTATR.DVx (data valid) is set, indicating that the pipe contains valid data for transmission.
- If the address prediction method is disabled (TCR.NO = 1), the transmission of a Write Offset and Data Frame is started as soon as the MLI transmitter is idle, no higher priority frames are pending, and TREADY = 1.
  If the address prediction method is enabled (TCR.NO = 0), a Write Offset and Data Frame is started only if an address prediction is not possible (indicated by TPxSTATR.OP = 0). If TPxSTATR.OP = 1, an address prediction is possible in the MLI transmitter (and the MLI receiver) and an Optimized Write Frame can be started. The address prediction method used is described on **Page 21-47**.
- Status flag TRSTATR.DVx is cleared by hardware and MLI event status flag TISR.NFSIx (Normal Frame Sent event in pipe x) is set (and a service request output is activated if enabled by TIER.NFSIEx = 1) after the Write Frame has been finished and correctly acknowledged by the MLI receiver.

The number m of offset address bits that are transmitted at a Write Offset and Data Frame is determined by the size of the Remote Window in the receiving controller that has been previously initialized by the transmission of a Copy Base Address Frame. Parameter m is referring to bit field TPxSTATR.BS (and RPxSTATR.BS) and can be in the range of 1 to 16 bits.

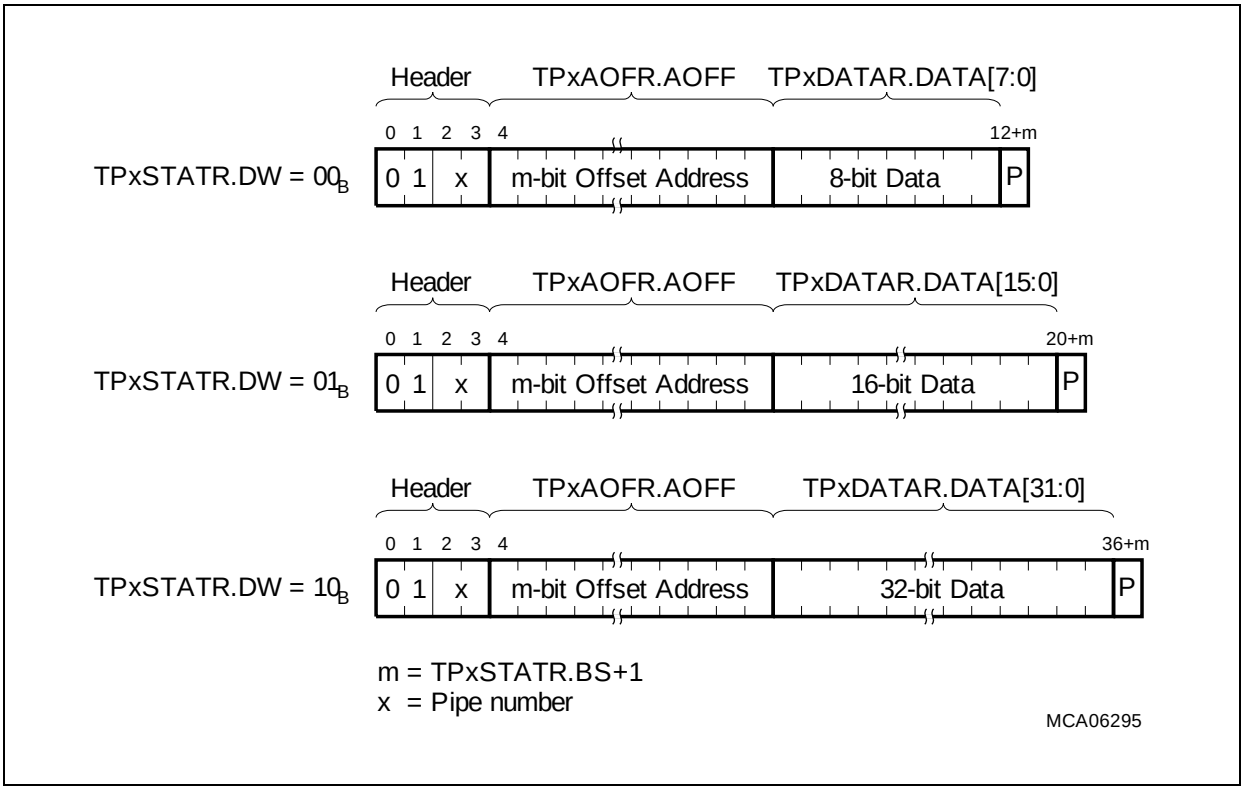


Figure 21-24 Write Offset and Data Frame

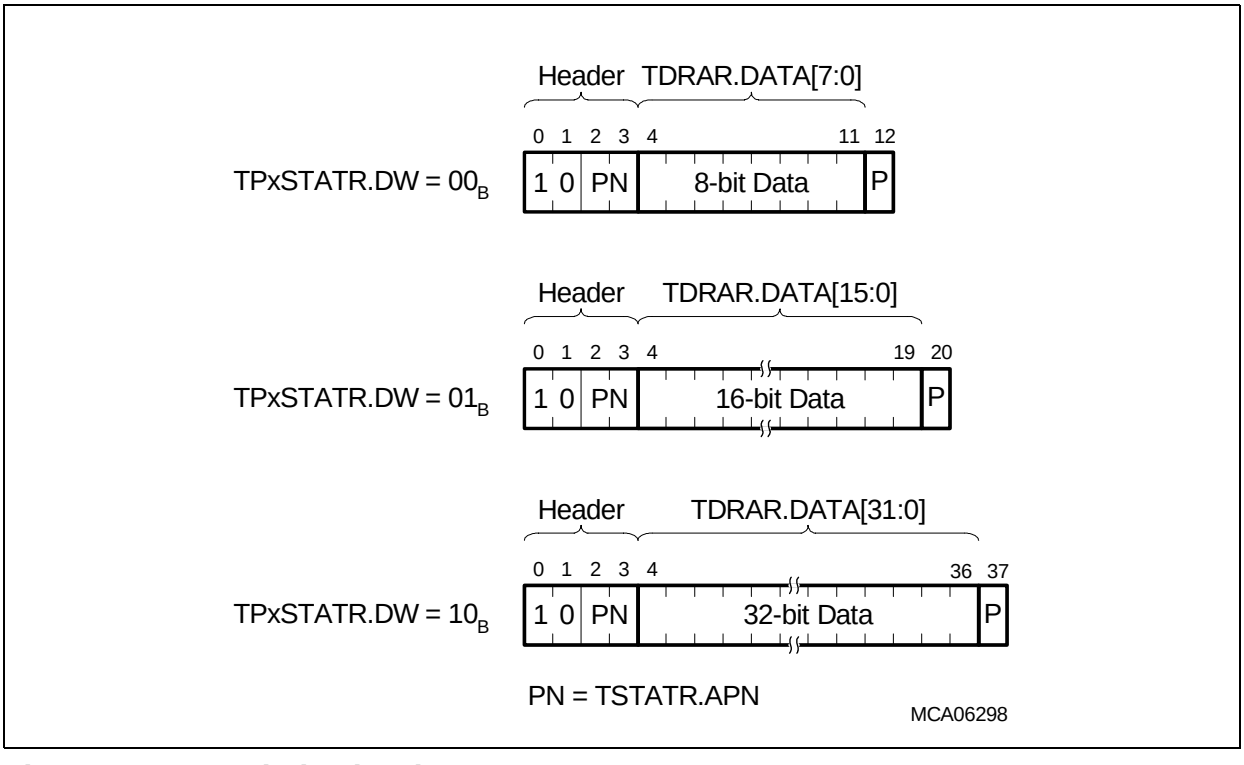


Figure 21-25 Optimized Write Frame

Receiving Controller

After a Write Frame has been received correctly and acknowledged, the following actions are automatically executed in the MLI receiver:

- In the case of a Write Offset and Data Frame:
  The result of the internal address prediction is not taken into account. The received offset address is added to the base address of the pipe x Remote Window and the result is stored in RPxBAR.ADDR. It is also stored in RADDR.ADDR and represents the destination address in the receiving controller where data should be written to.
  In the case of an Optimized Write Frame:
  The result of the internal address prediction is taken into account. The next address in the receiving controller where data should be written to is calculated by adding the detected receiver address prediction value RPxSTATR.AP to the actual address stored in RPxBAR.ADDR and the result is stored in RPxBAR.ADDR and in RADDR.ADDR.
- The received data is written into the receiver data register RDATAR (right aligned, unused bits are 0).
- The detected data width of the received data is written into bit field RCR.DW.
- The information about the received frame type (= 10_B for a Write Frame) is written into bit field RCR.TF.
- MLI event status flag RISR.NFRI (Normal Frame Received event) is set and an SR output line is activated if enabled by RIER.NFRIE = 01_B or 10_B.

After these actions related to the reception of a Write Frame by the receiving controller, the data that has been received from the transmitting controller is ready to be written into the Remote Window related to the receiving pipe.
This write operation can be executed in two ways:

- RCR.MOD = 0: Automatic Data Mode is disabled.
  In this mode, a bus master of the receiving controller, typically a CPU, is informed by a Normal Frame received event RISR.NFRI (a service request output is activated if RIER.NFRIE = 10_B) to transfer the received write data from the MLI receiver to the Remote Window. Therefore, it must read the data from RDATAR, together with width RCR.DW and the address stored in RADRR and write it to the indicated address location.
- RCR.MOD = 1: Automatic Data Mode is enabled.
  In this mode, the MLI module automatically writes the received write data to the Remote Window. This automatic action is controlled by a move engine block in the MLI receiver. It also sets event status flag RISR.MEI (move engine event when the access is terminated). A service request output is activated if enabled by RIER.MEIE = 1.
  The write operation to the Remote Window is executed only if the write address is within an enabled access protection range. If the address range is disabled for the write address, the automatic write action does not take place and event status flag RISR.MPEI (memory protection error) is set and a service request output is activated

if enabled by RIER.MPEIE = 1. In this case, the receiving controller software can analyze the values in RDATAR, together with width RCR.DW and the address stored in RADRR.

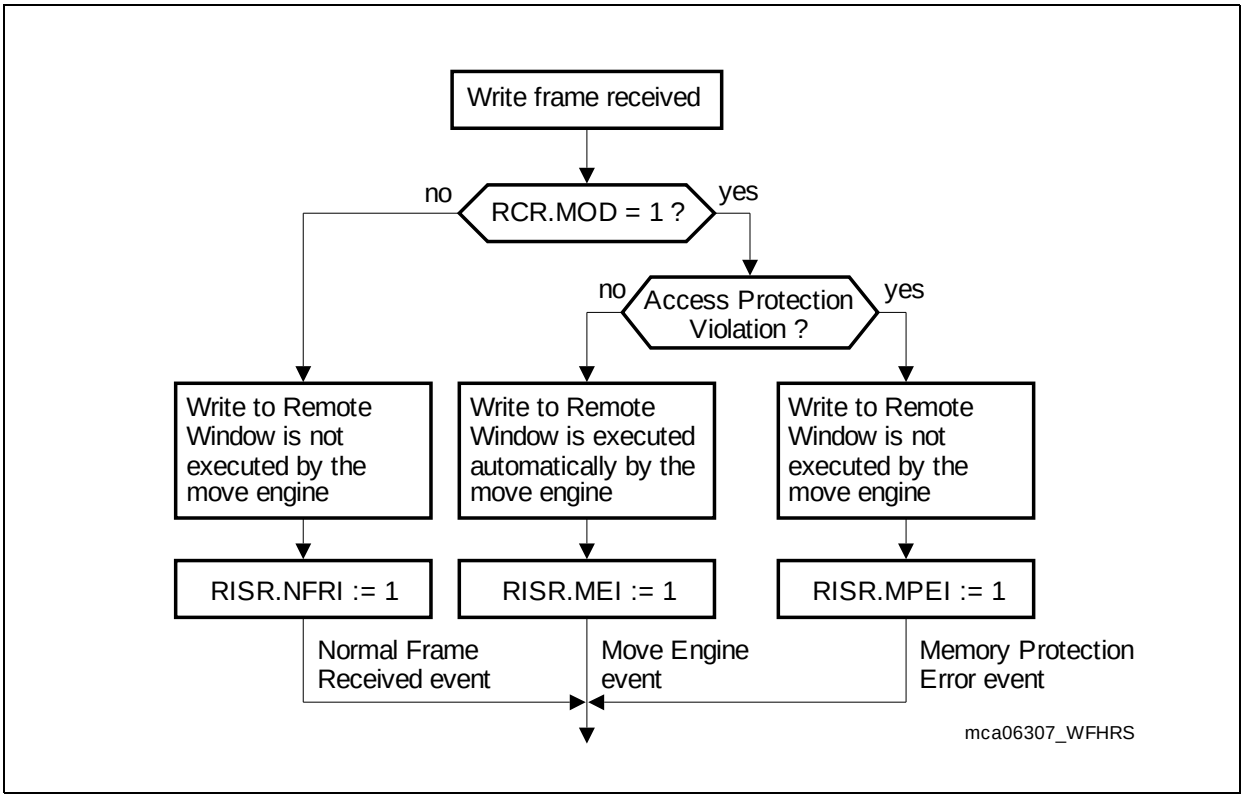


Figure 21-26 Write Frame Handling on Receiving Side

Note: In Automatic Data Mode, Write Frames are leading to a write action executed by the MLI move engine. During the move engine operation, only one more MLI frame can be received (stored in a waiting position to be executed). Then the reception of more frames is blocked by Non-Acknowledge handshake. If the move engine operation is finished, frame execution and reception continue normally.
If Automatic Data Mode is disabled, no blocking mechanism has been implemented. The receiving controller software has to take care to deal with the received data before it is overwritten by new incoming frames.

21.2.1.3 Read Frames

Read Frames transmit read request and optionally the read offset from the Local Controller to the Remote Controller.

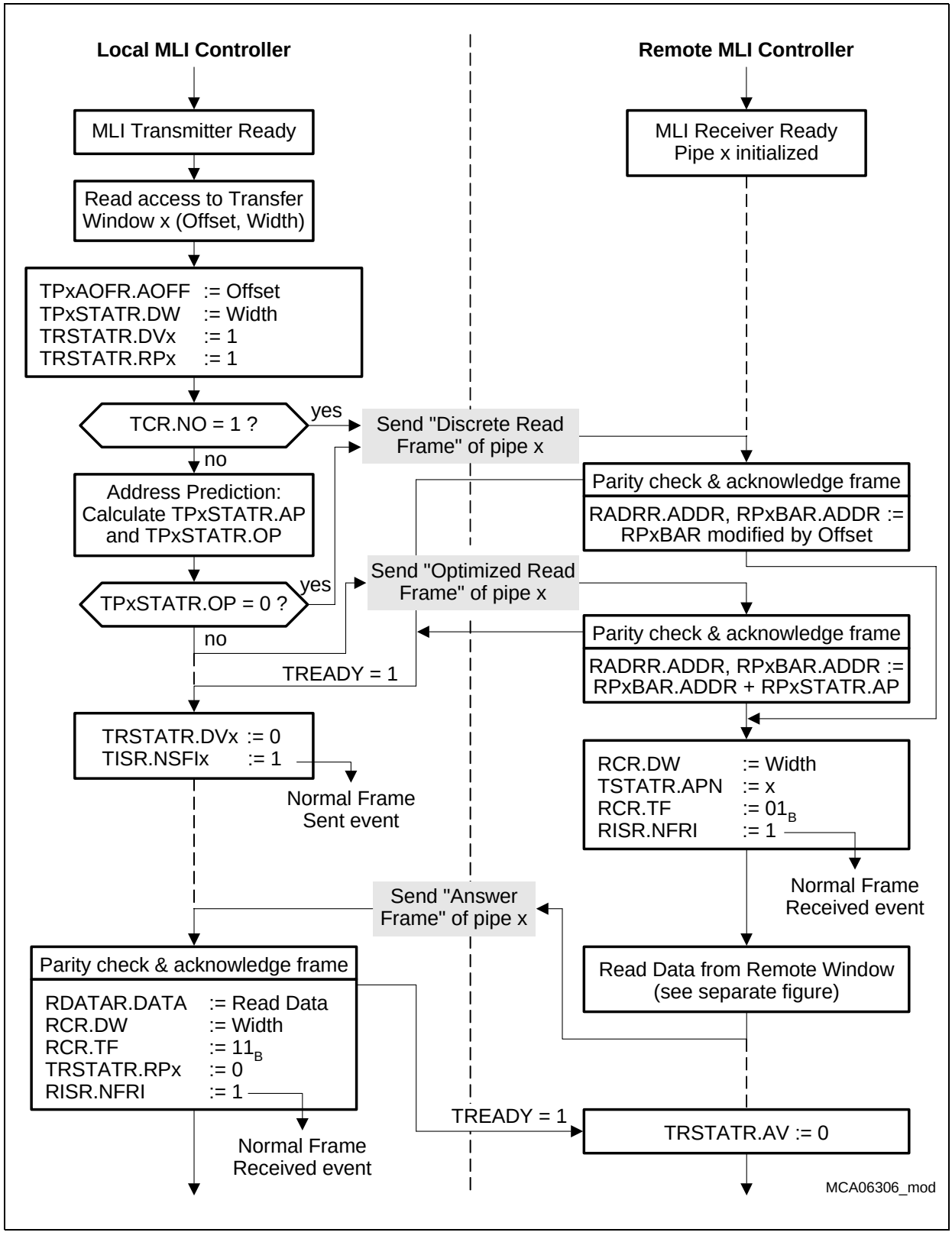


Figure 21-27 Read Frame and Answer Frame Flow

Local Controller

A read operation from a location within a Transfer Window x of the Local Controller delivers a dummy value as result of the read action and triggers the transmission of a Read Frame. The dummy value of the initial read action should be ignored and the software has to wait for the reception of the Answer Frame to get the desired data.

- The 16 least significant address bits of the Transfer Window read access are stored in TPxAOFR.AOFF as read offset address. In case of a an access to a Small Transfer Window, also 16 bits are stored, but the higher bits are not taken into account assuming the buffer size is configured correctly (see **Page 21-105**).
- The data width of the Transfer Window read access (8-bit, 16-bit, or 32-bit) is stored in bit field TPxSTATR.DW.
- Status flag TRSTATR.DVx (data valid) is set.
- Status flag TRSTATR.RPx (read pending) is set. This bit is cleared by hardware when an Answer Frame has been received correctly.
- If the address prediction method is not enabled (TCR.NO = 1), transmission of a Discrete Read Frame is started. If the address prediction method is enabled (TCR.NO = 0), a Discrete Read Frame is started only if an address prediction is not possible (indicated by TPxSTATR.OP = 0). If TPxSTATR.OP = 1, an address prediction is possible and an Optimized Read Frame is started.
- Status flag TRSTATR.DVx is cleared by hardware and MLI event status flag TISR.NFSIx (Normal Frame Sent event in pipe x) is set (and a service request output is activated if enabled by TIER.NFSIEx = 1) after the Read Frame has been finished and correctly acknowledged by the MLI receiver of the Remote Controller.

The number m of offset address bits that are transmitted at a Discrete Read Frame is determined by the (coded) size of the Remote Window in the Remote Controller that has been previously initialized by the transmission of a Copy Base Address Frame. Parameter m is stored in bit field TPxSTATR.BS (and RPxSTATR.BS) and can be in the range of 1 to 16 bits.

After a completed transmission of a Read Frame, the Local Controller expects the reception of an Answer Frame. The Answer Frame is introduced with the highest priority into the data flow of the transmitter of the Remote Controller.

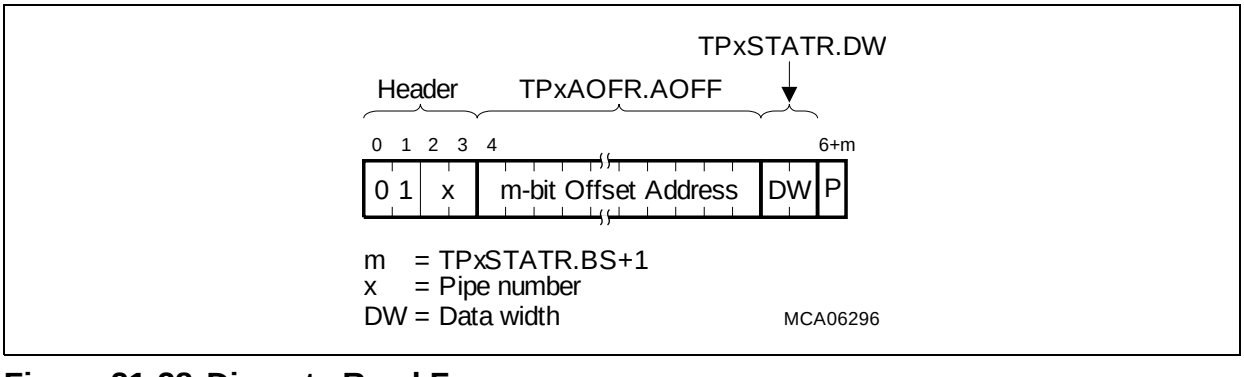


Figure 21-28 Discrete Read Frame

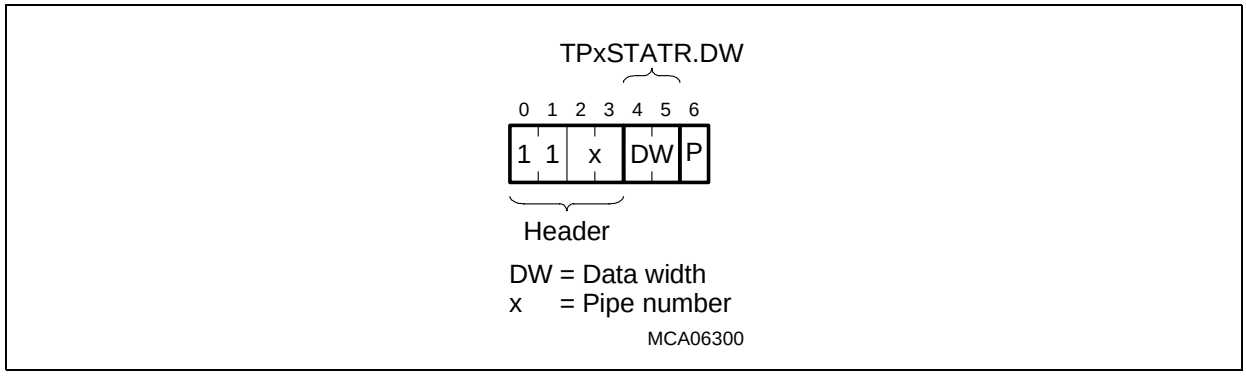


Figure 21-29 Optimized Read Frame

Remote Controller

After a Read Frame has been correctly received and acknowledged, the following actions are executed in the MLI receiver of the Remote Controller:

- In the case of a Discrete Read Frame:
  The result of the address prediction is not taken into account. The received offset address is added to the base address of the pipe x Transfer Window (stored in RPxBAR.ADDR). The result of this addition is stored in RADRR.ADDR and also in RPxBAR.ADDR and represents the destination address in the Remote Controller from where data should be read.
  In the case of an Optimized Read Frame:
  The result of the address prediction is taken into account. The next address in the Remote Controller where data should be read is calculated by adding the detected receiver address prediction value RPxSTATR.AP to the actual address stored in RPxBAR.ADDR. The result of this addition is stored in RADRR.ADDR and also in RPxBAR.ADDR and represents the destination address in the Remote Controller from where data should be read.
- The transmitted data width DW is written into bit field RCR.DW.
- The information about the received frame type (= 01_B for a Read Frame) is written into bit field RCR.TF.
- MLI event status flag RISR.NFRI (Normal Frame Received event) is set and a service request output is activated if enabled by RIER.NFRIE = 01_B or 10_B.

After correct reception of a Read Frame by the Remote Controller, the data requested by the Local Controller can be read by the Remote Controller and sent back to the Local Controller in form of an Answer Frame.

This read operation can be executed in two ways:

- RCR.MOD = 0:
  Automatic Data Mode is disabled. In this mode, a bus master of the Remote Controller, typically a CPU, is informed by a Normal Frame received event to read the requested read data and transfer it to the MLI receiver. Therefore, it must read data

with width RCR.DW from the address stored in RADRR and write the data into TDRAR.DATA.

- RCR.MOD = 1:
  Automatic Data Mode is enabled. In this mode, the move engine of the MLI automatically reads data from the Remote Window and sets event status flag RISR.MEI (move engine access terminated). A service request output is activated if enabled by RIER.MEIE = 1.
  The read operation from the Remote Window is executed only if the read address is within an enabled access protection range. If no address range is enabled for the actual read address, the automatic read action is not executed by the move engine, event status flag RISR.MPEI (memory protection error) is set and a service request output is activated if enabled by RIER.MPEIE = 1. In the interrupt handler routine, a bus master (e.g. CPU or PCP) must then take care of the remote window read operation and the data transfer to TDRAR.
- After TDRAR.DATA has been updated, status flag TRSTATR.AV of the Remote Controller is set and the transmission of an Answer Frame is started.

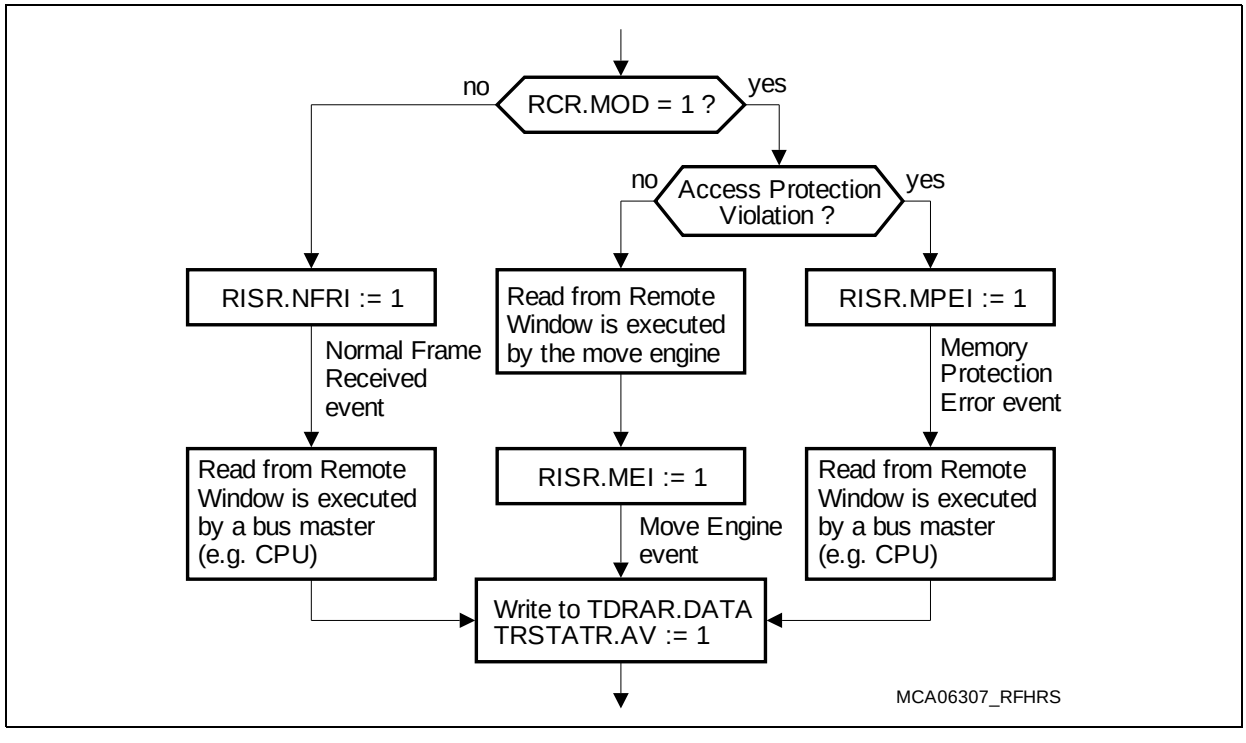


Figure 21-30 Read Frame Handling on Remote Side

Note: In Automatic Data Mode, Read Frames are leading to a read action executed by the MLI move engine. During the move engine operation, only one more MLI frame can be received (stored in a waiting position to be executed). Then the reception of more frames is blocked by a Non-Acknowledge handshake. If the move engine operation is finished, frame execution and reception can continue normally. If Automatic Data Mode is disabled, no blocking mechanism has been

implemented. The Remote Controller software has to take care to read the received data.

21.2.1.4 Answer Frame

Please note that only one Answer Frame can be handled by the system at a time (no Read Frame request while any TRSTATR.RPx is set). Make sure that not more than one Read Frame is pending at a time. If a Read Frame is not answered by an Answer Frame during a certain time interval, a time-out criterion should be handled in software. The Remote Controller has to take care that no Answer Frame is delivered after the time-out criterion has been detected (e.g. by a software-triggered Command Frame). Do not start a new Read Frame while waiting for an Answer Frame if the time-out criterion has not yet been detected and the Answer Frame has not yet been received. The length of the time-out interval depends on the application and has to be defined accordingly on a case by case base (e.g. the transfer rates between MLI modules, bus architecture, etc. have to be considered). In the case a time-out has been detected, the Local Controller software has to clear the TRSTATR.RPx bit by writing 1 to SCR.CDVx and can start a new Read Frame.

Remote Controller (Receiving the read request)

The Answer Frame is the only frame sent from the Remote Controller back to the Local Controller. The transmitter registers of the Remote Controller are used to generate the Answer Frame.

Every time the transmitter data read answer register TDRAR is written in the Remote Controller, the transmission of an Answer Frame is started and the following actions are triggered.

- Status flag TRSTATR.AV is set to trigger the transmission of an Answer Frame.

The following parameter is transmitted in the data field of the Answer Frame:

- Read data: stored in TDRAR.DATA; data width is determined by TRSTATR.DW.
- Status flag TRSTATR.AV is cleared after the Answer Frame has been finished and correctly acknowledged by the MLI receiver of the Local Controller.

An Answer Frame should be sent through the pipe that has received a read request but there must be only one MLI Transfer Window read access pending on any side of a MLI connection at any time, because the answer mechanism does not contain buffers for multiple Answer Frames.

Local Controller (Transmitting the read request)

If an Answer Frame has been received correctly and acknowledged, the following actions are executed in the MLI receiver of the Local Controller:

- The TRSTATR.RPx flags are cleared.

- The received data is written into the receiver data register RDATAR.
  If 8 data bits are received, they are duplicated to all 4 bytes in RDATAR.
  If 16 data bits are received, they are duplicated to both half-words in RDATAR.
- The detected data width of the received data is written into bit field RCR.DW.
- The received Pipe Number x represents the answer Pipe Number and is written into bit field TSTATR.APN.
- The information about the received frame type (= 11_B for an Answer Frame) is written into bit field RCR.TF.
- MLI event status flag RISR.NFRI (Normal Frame Received event) is set and a service request output is activated if enabled by RIER.NFRIE = 01_B or 10_B.
- The content of RADRR becomes invalid.
- The data that has been previously requested from the Remote Controller by a Read Frame is now available in RDATAR and can be read by a bus master (e.g. the CPU) of the Local Controller.
- If an Answer Frame is received while the corresponding TRSTATR.RPx bit is 0, the reception is declared as unintended and a Discarded Read Answer event is generated (see **Page 21-62**).

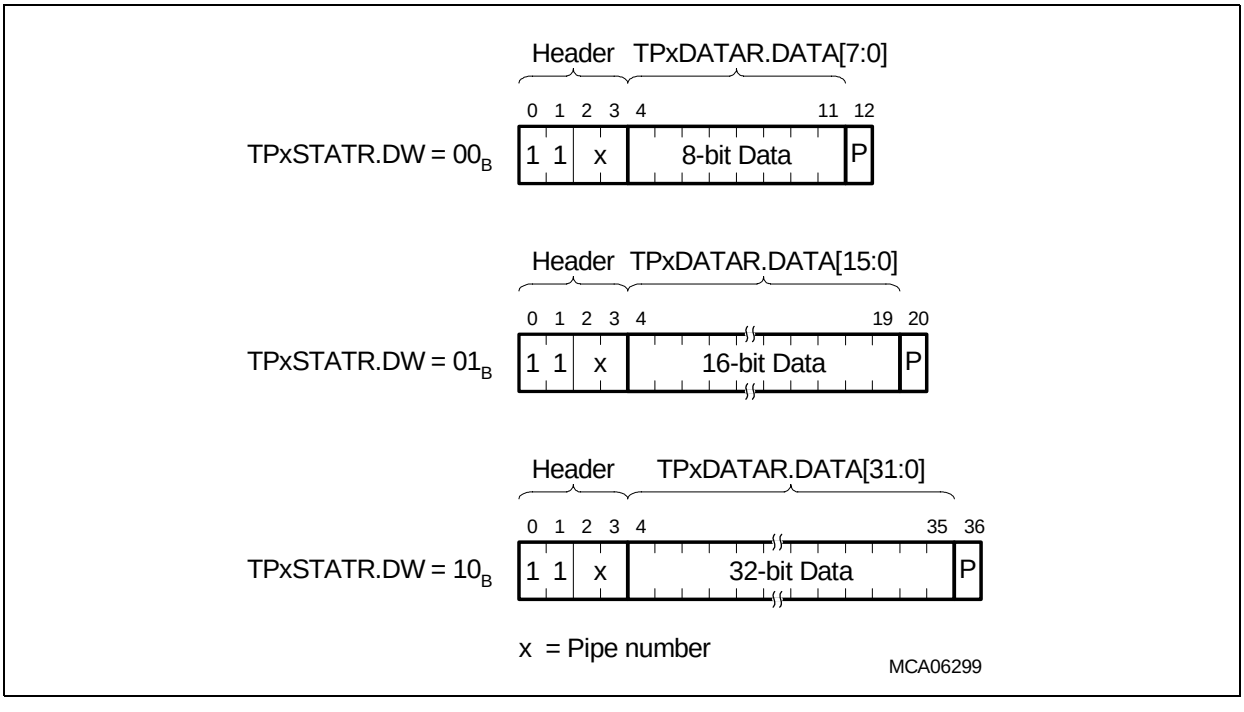


Figure 21-31 Answer Frame

Note: If an Answer Frame has been correctly received in the Local Controller, the Local Controller's software has to read it. As long as at least one byte of this data has not yet been read out, only one more MLI frame can be received (stored in a waiting position to be executed). Then the reception of more frames is blocked by Non-Acknowledge handshake. If the received data has been read out, frame execution and reception continue normally.

21.2.1.5 Command Frame

Command Frames transmit a command (e.g. setup information or service request) from a transmitting controller to a receiving controller.

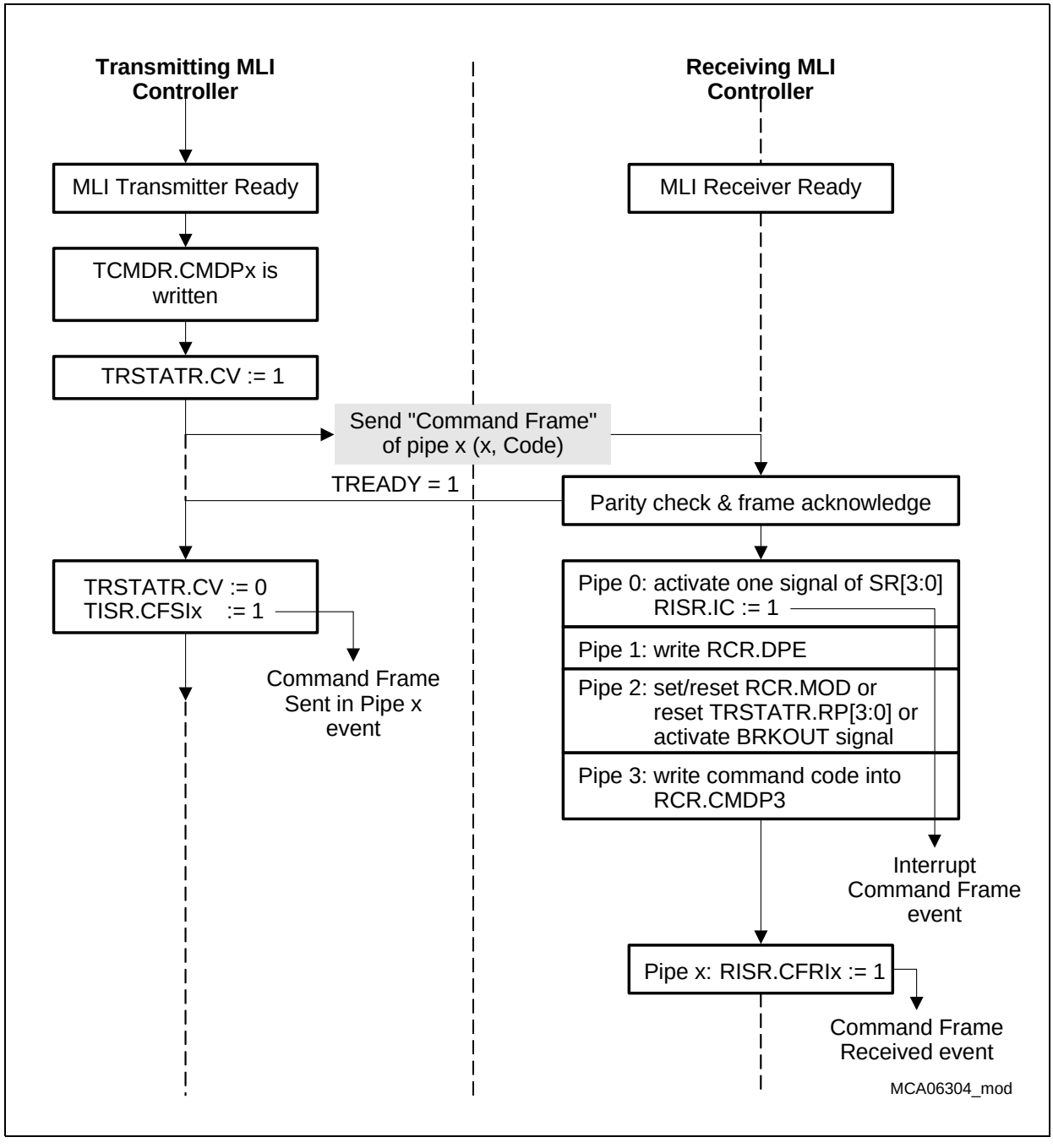


Figure 21-32 Command Frame Transaction Flow

Transmitting Controller

The transmission of a Command Frame is initiated by writing one of the four pipe x related command code bit fields in register TCMDR.CMDPx, triggering the following actions:

- Status flag TPxSTATR.CVx (command valid) is set and the Command Frame transmission is started using x as pipe number PN and the command code stored in TCMDR.CMDPx as parameters.
- TRSTATR.CVx is cleared after the Command Frame has been finished and correctly acknowledged by the MLI receiver of the Remote Controller.
- MLI event status flag TISR.CFSIx (Command Frame Sent event in pipe x) is set and a service request output is activated if enabled by TIER.CFSIEx = 1.

Receiving Controller

Depending on the pipe x related command code that is transmitted by a Command Frame, different actions are triggered in the receiving controller. **Table 21-5** describes the actions that are transmitted by a Command Frame and that cause a specific control task in the MLI receiver.

- The received PN value is checked and the corresponding control actions are executed according to **Table 21-5**.
- Independent of the received Pipe Number, event status flag RISR.CFRIx (Command Frame Received event in pipe x) is set and a service request output is activated if enabled by RIER.CFRIEx = 1.

If a Command Frame is received for pipe 2 with command code 1111_B, the $\overline{\text{BRKOUT}}$ output signal of the MLI module becomes activated if it is enabled by bit RCR.BEN = 1. If disabled by RCR.BEN = 0, signal $\overline{\text{BRKOUT}}$ will not be activated. The usage of $\overline{\text{BRKOUT}}$ is implementation-specific and can be used, for example, to generate a break condition in the on-chip debug support logic or trigger other functions.

Table 21-5 Command Frame Encoding

PN	CMD	Command Description
00_B	0001_B	Activate service request output SR0 of receiving MLI module
	0010_B	Activate service request output SR1 of receiving MLI module
	0011_B	Activate service request output SR2 of receiving MLI module
	0100_B	Activate service request output SR3 of receiving MLI module
	Others	no effect, reserved for future use

Table 21-5 Command Frame Encoding (cont'd)

PN	CMD	Command Description
01_B	0000_B	Set RCR.DPE (delay for parity error indication) in receiving MLI to 0000_B
	0001_B	Set RCR.DPE in receiving MLI to 0001_B
	0010_B	Set RCR.DPE in receiving MLI to 0010_B
	...	...
	1111_B	Set RCR.DPE in receiving MLI to 1111_B
10_B	0001_B	Enable Automatic Data Mode in receiving MLI (set RCR.MOD = 1)
	0010_B	Disable Automatic Data Mode in receiving MLI (set RCR.MOD = 0)
	0100_B	Clear bit TRSTATR.RP0 in receiving MLI
	0101_B	Clear bit TRSTATR.RP1 in receiving MLI
	0110_B	Clear bit TRSTATR.RP2 in receiving MLI
	0111_B	Clear bit TRSTATR.RP3 in receiving MLI
	1111_B	Generate break output signal $\overline{\text{BRKOUT}}$ in receiving MLI (if enabled by RCR.BEN = 1)
	others	no effect, reserved for future use
11_B	Any	Free programmable software command, written into bit field RCR.CMDP3 of receiving MLI

21.2.2 General MLI Features

The general MLI features comprise the:

- Parity generation and checking (see **Page 21-44**)
- Non-Acknowledge error (see **Page 21-47**)
- Address prediction (see **Page 21-47**)
- Automatic data transfers (see **Page 21-48**)
- Access protection (see **Page 21-49**)
- Triggered Command Frames (see **Page 21-49**)
- Transmit priority (see **Page 21-50**)
- Transmission delay (see **Page 21-50**)

21.2.2.1 Parity Check and Parity Error Indication

For parity generation, the number of transmitted bits with the value of 1 is counted over the header and the complete data field of a frame. For even parity, the parity bit is set if the result of a modulo-2 division of the elaborated number is 1. For odd parity, the parity bit is set if the result of a modulo-2 division of the elaborated number is 0.

For a parity error-free MLI connection, even parity must be selected in the transmitter because the receiver operates only with even parity detection. The capability to select odd parity can be used by the transmitter to force a parity error reply from the receiver during the startup procedure of the MLI connection. This can be used to measure the propagation delay and to optimize the ready delay time (see **Page 21-73**).

Note: There is no protection against frames where more than one bit is corrupted (e.g. shortened frames). In such a case, an unpredictable behavior of the MLI module may occur.

Transmitting Controller

The MLI transmitter counts the detected parity error conditions and generates a parity error event if a programmable number (max. 16) of parity error conditions has occurred. A parity error condition is indicated to the transmitter by the receiver after the transmission of a frame (see **Page 21-23**). The transmitter parity error condition is detected when the TREADY signal is sampled at low level within a programmable number (TCR.MDP = maximum delay for parity errors) of TCLK clock cycles after TVALID has been de-asserted to low.

If a transmitter parity error condition is detected, the MLI transmitter sets the parity error flag TSTATR.PE and also decreases the maximum parity error counter TCR.MPE by 1. The maximum parity error counter of the transmitter TCR.MPE determines the number of transmit parity error conditions that can be still detected until a transmitter parity error event is generated. If a transmitter parity error condition is detected and TCR.MPE is becoming 0 or while it is 0, a transmitter parity error event is generated by setting bit TISR.PEI (see **Figure 21-40** on **Page 21-60**) and an SRx output line is activated if

enabled by TIER.PEIE = 1. After a transmitter parity error event occurred, TCR.MPE can be set again by software to a value greater 0001_B. Otherwise, each additional transmitter parity error condition will generate a parity error event.

The transmitter parity error flag TSTATR.PE is cleared by hardware when a correct frame transmission and TREADY has been sampled with 1 within the ready delay time. It can be cleared by software by writing a 1 to bit SCR.CTPE. If for example, each transmitter parity error condition should generate a transmitter parity error event, TCR.MPE should be set to 0000_B. The software can check for accumulated parity error conditions by reading TCR.MPE or TISR.PEI, for the status of the latest received frame, it can check TSTATR.PE.

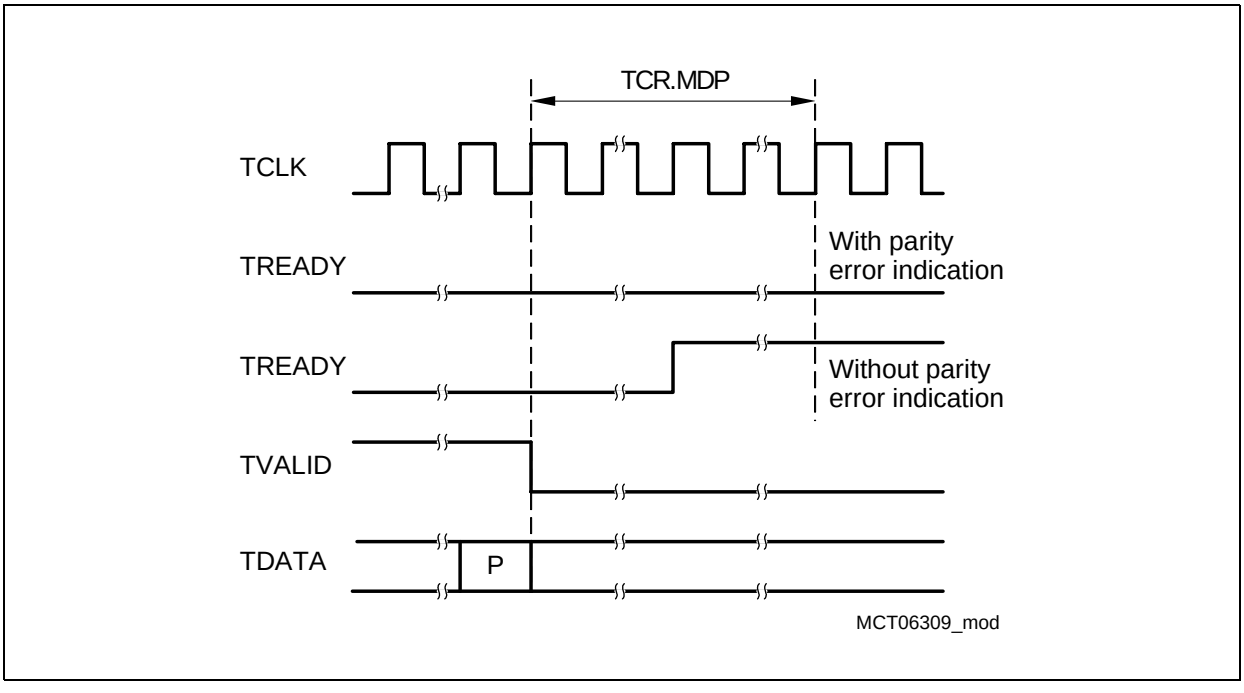


Figure 21-33 Parity Error Indication for the Transmitter

Receiving Controller

The receiver always checks the parity bit of a received frame for even parity. A receiver parity error condition is detected if the received parity bit does not match with the internally calculated one. If no receiver parity error condition is found after the reception of a frame, RREADY is immediately set to 1, otherwise RREADY is kept at 0 until a defined number of RCLK cycles (determined by bit field RCR.DPE = delay for parity error) has been elapsed. Then, RREADY is asserted high.

If a receiver parity error condition is found, the MLI receiver sets the parity error flag RCR.PE and additionally decreases the maximum parity error counter of the receiver RCR.MPE by 1. The maximum parity error counter RCR.MPE determines the number of receiver parity error conditions that can be still detected until a receiver parity error event is generated. If a receiver parity error condition is detected and RCR.MPE is becoming

0 or while it is already 0, a receiver parity error event is generated by setting bit RISR.PEI (see **Figure 21-44** on **Page 21-63**) and a service request output is activated if enabled by RIER.PEIE = 1. After a receiver parity error event has occurred, RCR.MPE can set again by software to a value greater 0001_B. If, for example, each receiver parity error condition should generate a receiver parity error event, RCR.MPE can be programmed to 0000_B or 0001_B.

The receiver parity error flag RCR.PE is cleared by hardware if a correct frame transmission has occurred. RCR.PE can be cleared by software by writing a 1 to bit SCR.CRPE.

The receiver parity error flag RCR.PE is cleared by hardware after a correct frame reception. It can be cleared by software by writing a 1 to bit SCR.CRPE. The software can check for accumulated parity error conditions by reading RCR.MPE or RISR.PEI, for the status of the latest received frame, it can check RCR.PE.

The delay for parity error bit field RCR.DPE is a read-only bit field in the receiver that updated by hardware if a Command Frame for pipe 1 is received. With this frame type, the transmitting controller transfers a value for RCR.DPE to the receiving controller during the setup phase of the MLI connection.

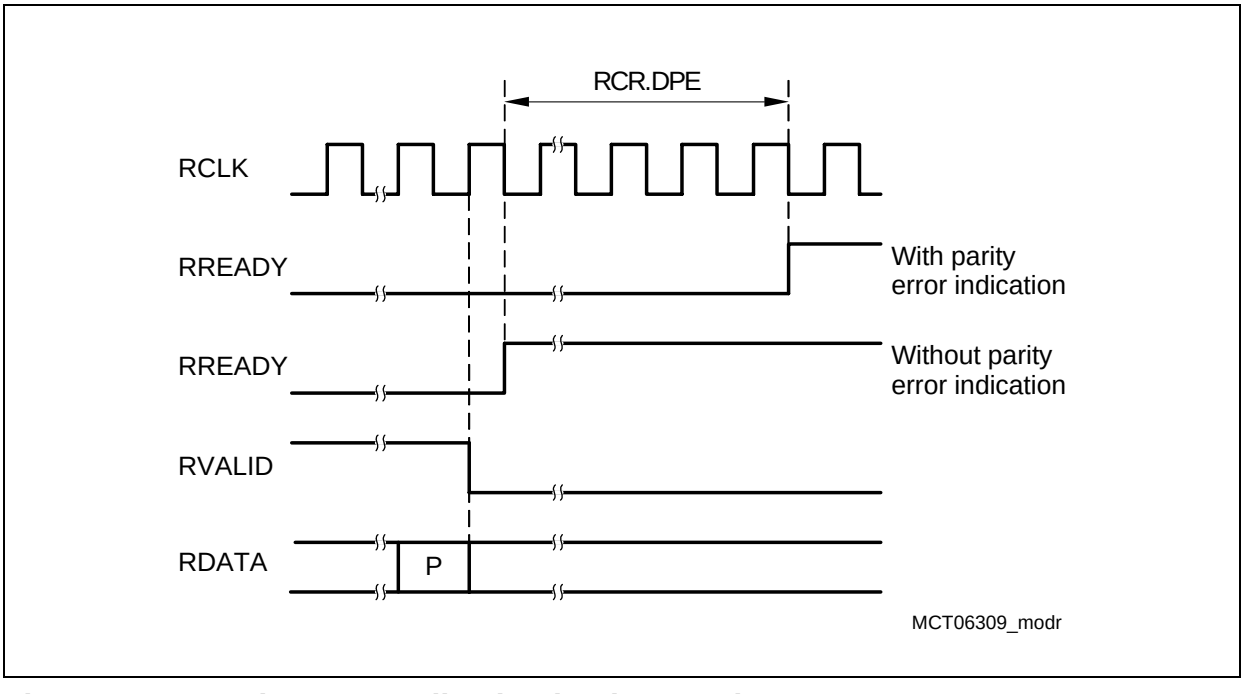


Figure 21-34 Parity Error Indication by the Receiver

21.2.2.2 Non-Acknowledge Error

A Non-Acknowledge error condition is detected by the transmitter when at the end of a frame transmission, the TREADY signal is still at high level (TREADY = 1 when TVALID becomes 0). In this case, the error flag TSTATR.NAE is set and the maximum Non-Acknowledge error counter TCR.MNAE is decremented by 1. If a Non-Acknowledge error condition is detected and TCR.MNAE is becoming 0 or while it is already 0, a time-out event is generated by setting bit TISR.TEI (see **Figure 21-40** on **Page 21-60**) and an MLI service request is generated if enabled by TIER.TEIE = 1. The Non-Acknowledge error flag TSTATR.NAE is cleared by hardware when a frame transmission has been acknowledged correctly. It can also be cleared by software when writing a 1 to bit SCR.CNAE.

The Non-Acknowledge error counter TCR.MNAE is automatically set to 11_B when a frame has been acknowledged correctly. It can be read and written by software, allowing a limited number of consecutive Non-Acknowledge errors to be defined that can be detected until a time-out error event is generated. If, for example, the first occurrence of a Non-Acknowledge error should lead to a time-out event, bit TCR.MNAE has to be written by software with 00_B or 01_B after each correctly received frame.

21.2.2.3 Address Prediction

An address prediction method can be enabled to support communication between MLI transmitter and MLI receiver without sending address offset information in the frames to optimize the required MLI bandwidth. This feature reduces the required bandwidth for MLI communication. Both communication partners, MLI transmitter and the MLI receiver are able to detect regular offset differences of consecutive window accesses to the same window. The address prediction mechanism working independently for each pipe, different prediction values can be handled in parallel for the different pipes.

Transmitting Controller

If the address prediction method is enabled (TCR.NO = 0), the MLI transmitter compares the offset of each Transfer Window read or write access with the offset of the previous access to the same Transfer Window (stored in TPxAOFR.AOFF). The result of this comparison is stored in two’s complement representation in TPxSTATR.AP (limited to 9 bits, otherwise prediction is not possible). Between the accesses to a specific window, other windows can be accessed without disturbing the prediction.

If the offset differences are identical in at least two consecutive accesses to the same Transfer Window, an address prediction is possible (flag TPxSTATR.OP becomes set) and optimized frames can be sent to the receiving controller for this pipe. If the offset difference of a next access to the same Transfer Window does not match the calculated value in TPxSTATR.AP, flag TPxSTATR.OP is cleared and address prediction is not possible. In this case, a Normal Frame for writing or reading (Write Offset and Data Frame or Discrete Read Frame) is started.

Receiving Controller

The MLI receiver operates with an address prediction method equivalent to the MLI transmitter. This means that after receiving at least two consecutive Write Offset and Data Frames and/or Discrete Read Frames that include address information, the MLI receiver is able to follow the address prediction method used by the MLI transmitter.
Each received offset is compared in the MLI receiver with the offset of the previously received frame of the same pipe. The result of this comparison is stored in two's complement representation in RPxSTATR.AP (limited 9 bits).

If an optimized frame is received by the MLI receiver, it calculates the next address by adding the value stored in RPxSTATR.AP to the contents of the receiver address register RADRR.

In case of a Write Offset and Data Frame or a Discrete Read Frame (m offset bits), the receiver address registers RADRR and RPxBAR are always loaded with an updated address. This address is calculated by replacing the lowest m bit positions in RPxBAR with the received offset value. In this case, the address delta value stored in RPxSTATR.AP is not taken into account. The programmed size of the Remote Window and the number m of offset bits are given by RPxSTATR.BS. The bit positions RPxBAR[31:m] are kept constant, whereas the bit positions RPxBAR[m-1:0] are replaced.

21.2.2.4 Automatic Data Mode

The MLI module supports automatic data transfers for read or Write Frames without any CPU load in the receiving controller. This features is based on a move engine block providing the data, the complete address and the data width to an associated bus master on the system bus (see **Figure 21-1**). Depending on the implementation, this bus master can be capable of executing the requested data move operations autonomously. The Automatic Data Mode in the receiving controller can be enabled (RCR.MOD = 1) or disabled (RCR.MOD = 0) by software on receiving side or a Command Frame sent by the transmitting controller.
If the Automatic Data Mode is disabled, the receiving controller software has to execute the requested data transfers.
Additionally to the global enable/disable of the automatic mode by **RCR**.MOD, it is possible to individually exclude address ranges from automatic data transfer by an access protection scheme. The definition of the address ranges depend on the product and has been introduced to support the protection of critical data or modules.

Note: If a device contains the MLI move engine block as the only bus master, automatic mode has to be selected to allow transfers. This could be the case for external peripheral devices without own CPU.

21.2.2.5 Memory Access Protection

The MLI receiver provides a memory access protection logic allowing to exclude read and write accesses of the MLI move engine to specific parts of the memory map from automatic mode. Each address of a data move (read or write) is always checked if it targets an address range that is enabled for read/write access. If a requested data move is targeting an excluded address range, a memory access protection error event is generated and the receiving controller's software can take care of the service request.

The memory access protection logic handles two levels of address range definitions:

- Fixed address ranges (for complete modules or memory areas)
- Programmable address sub-ranges (to limit accesses to specific parts of bigger memory areas)

There is a maximum of 32 fixed address ranges available that can be individually enabled/disabled by the address range enable bits AER.AENx (x = 0-31). If bit AER.AENx is set, read/write accesses to the associated address range x are supported in automatic mode. If bit AENx is cleared, read/write accesses to the associated address range x are not automatically executed, a memory protection error event is generated, and SRx output line is activated if enabled by RISR.MPEI.

The MLI module supports a definition of up to four programmable address sub-ranges (with index n) within fixed address ranges. The parameters for the sub-ranges are stored in the access range register ARR, comprising:

- The size of an address slice defined as sub-range (ARR.SIZEn)
- The location of an address slice defined as sub-range (ARR.SLICEn)

Note: The definition of the fixed address ranges and the sub-ranges is product-specific. Detailed values are given in the module implementation chapter.

21.2.2.6 Triggered Command Transfers

The MLI module supports the transmission of Command Frames triggered by hardware signals (up to 4 trigger inputs TR[3:0]). If a rising edge at a TRx input is detected, a corresponding bit TRSTATR.CIVx is set. The MLI transmitter sends out a Command Frame with PN = 00_B and CMD = x + 1 if bit CIVx = 1. This Command Frame can then trigger the activation of the corresponding SRx service request output of the Remote Controller. A Triggered Command Frame can be used monitor service request signals in the Local Controller and to transfer the requests to the Remote Controller, without intervention of any CPU.

Bit CIVx is automatically cleared after successful transmission of the related Command Frame or by writing 1 to SCR.CCIVx.

Note: The connection of the TR[3:0] input lines is product-specific. Detailed information is given in the module implementation chapter (see **Page 21-137***).*

21.2.2.7 Transmit Priority

In the case that several requests for frame transmission are pending at the same time in a MLI transmitter, the following priority scheme is applied, starting with the highest priority.

For the Answer Frame, only one frame can be pending at a time in the transmitter. So the user has to take care that an older Answer Frame is completely handled before requesting a new one. The same applies for the base address copy frame.

For the Triggered Command Frames, the software driven Command Frames and the read or Write Frames, one frame of each type can be pending per pipe at a time.

Note: The MLI has 4 inputs for Triggered Command Frames. They are not necessarily connected in all devices. Please refer to the device specific implementation chapter for details (see ***Page 21-137****).*

- Answer Frame (only one frame pending allowed at a time)
- Triggered Command Transfer (CIV0 before CIV1 before CIV2 before CIV3)
- Software driven Command Frames (CV0 before CV1 before CV2 before CV3)
- Read or Write Frames (DV0 before DV1 before DV2 before DV3)
- Base Address Copy Frame (only one frame pending allowed at a time)

21.2.2.8 Transmission Delay

A transmission delay can be introduced in the transmitter between the detection of the rising edge of the RREADY input signal and the next possible frame start. This delay represents the minimum time between the acknowledge of a former frame by RREADY and a new frame (if a request is pending). The delay is defined by bit field TCR.TDEL in cycles of the transmitter system clock f_{SYS}.

21.2.3 Interface Description

The MLI transmitter and MLI receiver communicate with other MLI receivers and MLI transmitters via a four-line serial connection each. Several I/O lines of these connections are available outside the MLI module kernel as a four-line output or input vector with index numbering A, B, C and D. The MLI module internal I/O control blocks define which signal of a vector is actually taken into account and also allow polarity inversions (to adapt to different physical interconnection means).

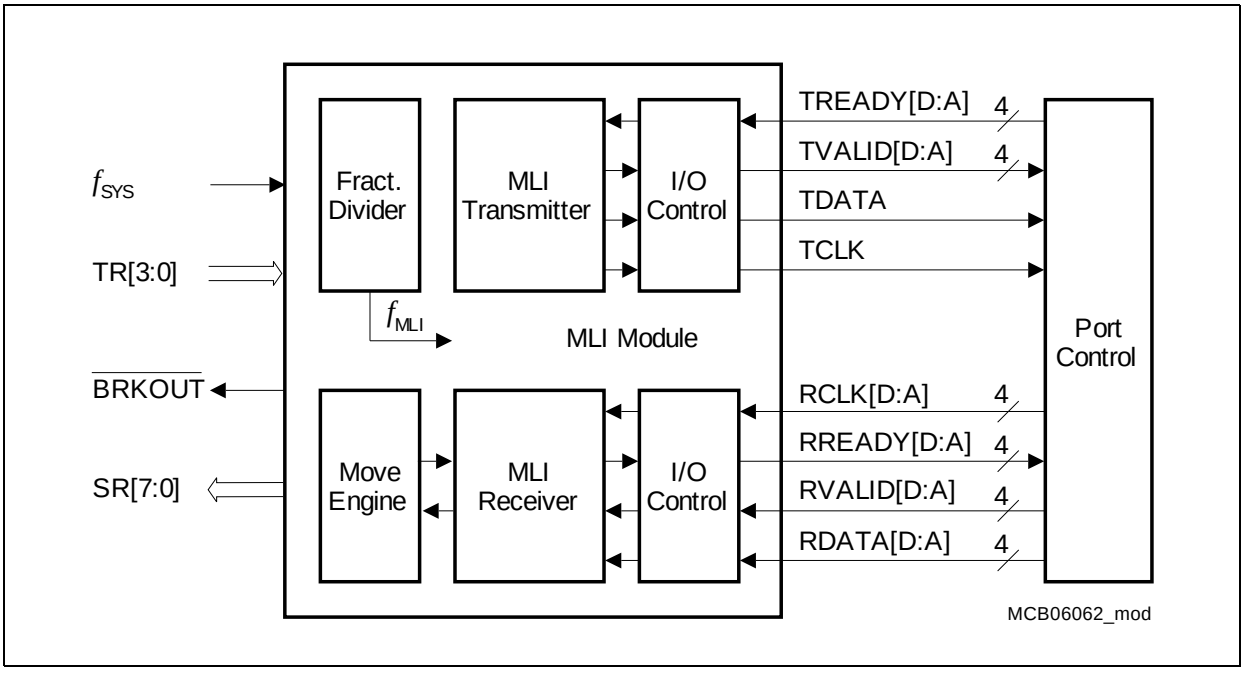


Figure 21-35 General Block Diagram of the MLI Module

Each input/output signal used for MLI communication between a transmitter and a receiver can be disabled and inverted in its polarity. Please note that all waveform diagrams in the MLI chapter refer to non-inverted signals. If polarity inversions are programmed, the waveform diagrams have to be interpreted accordingly. In order to avoid naming mismatches, the signals keep their names, although a polarity inversion might have been programmed. If desired, polarity inversions for the same signal have to be programmed in the transmitter and in the receiver to guaranty signal consistency (there has always to be an even number of inversions between an MLI transmitter and receiver). After reset, the following setting is applied, allowing MLI communication without modification of register OICR[1)]:

- The signal with the index A is selected from each input/output vector.
- TCLK generation is enabled and RCLK reception is enabled.

1) Other services (e.g. an automatic boot sequence or a boot routine) can change the OICR setting. Differing values are then indicated in the corresponding implementation chapter.

- Polarity inversion is disabled for all signals (no inversion).
- Not selected output signals are at low level.

The usage of signal BRKOUT is implementation-specific and can be used, for example, to generate a break condition in the on-chip debug support logic or trigger other functions. This signal is activated (as a pulse) by a Command Frame.

The service request outputs SR[7:0] of the MLI module can be activated (as a pulse) by transmitter or receiver events (for all SRx), as well as by Command Frames (only for SR[3:0]).

The MLI module also supports 4 trigger inputs TR[3:0]. A rising edge at input TRx sets bit TRSTATR.CIVx and requests the transfer of a Triggered Command Frame in pipe 0, with a CMD = x + 1.

21.2.3.1 Transmitter I/O Line Control

Figure 21-36 shows the MLI transmitter I/O control logic.

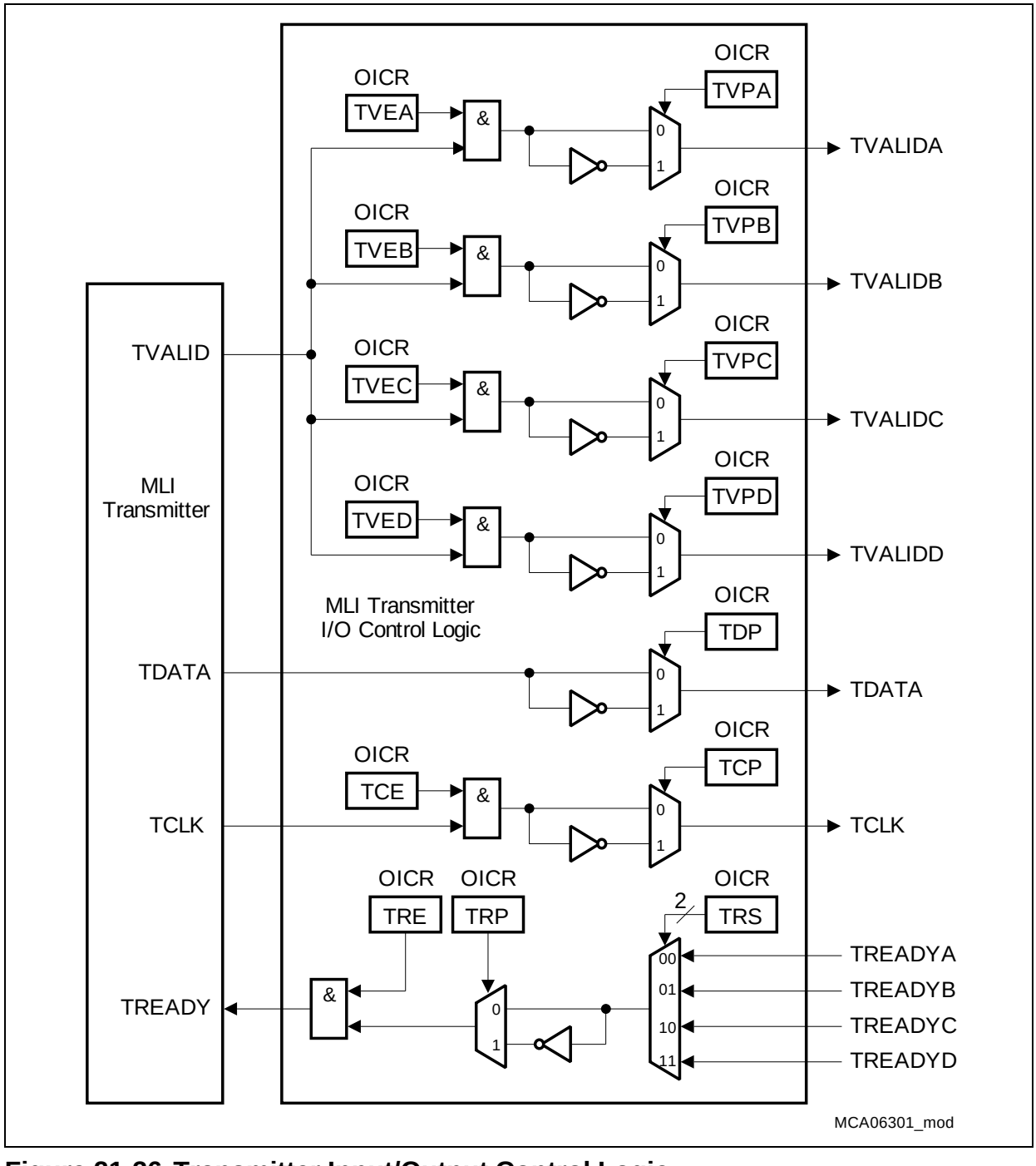


Figure 21-36 Transmitter Input/Output Control Logic

21.2.3.2 Receiver I/O Line Control

Figure 21-37 shows the MLI receiver I/O control logic.

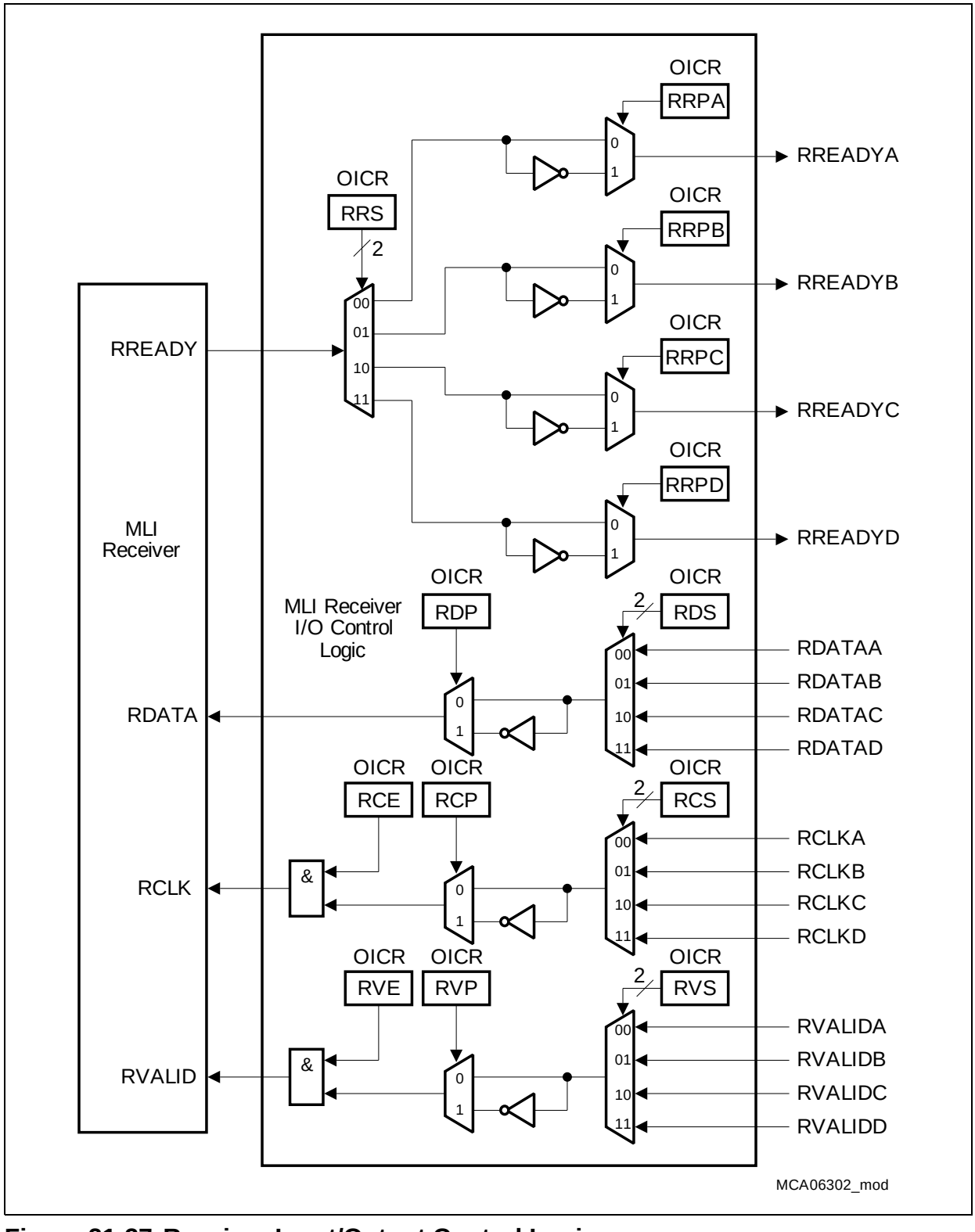


Figure 21-37 Receiver Input/Output Control Logic

21.2.3.3 Connecting Several MLI Modules

The MLI structure also allows to connect several MLI modules together, e.g. two Remote Controllers (X and Y) to one Local Controller. In this case, the Local Controller can send data to either one or the other or to both Remote Controllers in parallel. Each Remote Controller is connected via an own set of READY/VALID signals to the Local Controller, whereas the transmitter DATA and CLK are broadcast signals. The status of the VALID lines defines, which Remote Controller is accessed.

Only one receiver being available in the Local Controller, the reception of data can be handled only either from one or the other Remote Controller. The software has to ensure that only one Remote Controller sends data back to the Local Controller, e.g. by using Read Frames or by enabling/disabling the generation of Write Frames in the Remote Controllers.

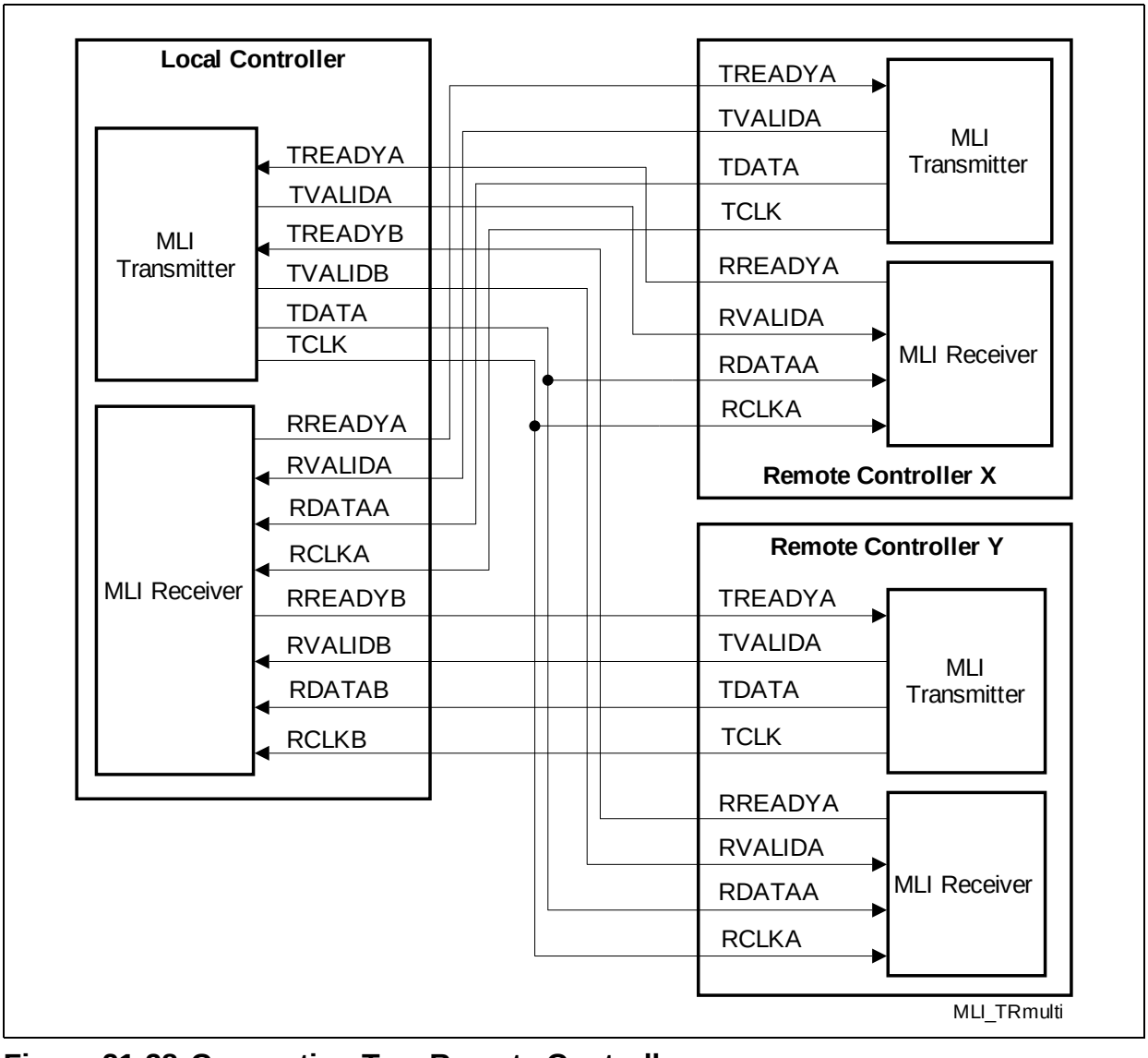


Figure 21-38 Connecting Two Remote Controllers

Another possibility to connect several MLI modules is a ring structure, with (at least) one dedicated pipe per device. This leads to a structure where the Local Controller's transmitter is connected to the receiver of Remote Controller X, the transmitter of Remote Controller X to the receiver of Remote Controller Y, and the transmitter of Remote Controller Y to the Local Controller's receiver.

This structure supports autonomous data generation and transfer in both Remote Controllers, for example to transfer data generated in a Remote Controller to the Local Controller without using Read Frames. In a ring structure, the Read Frame handling should be avoided. It is possible for the Local Controller to access both Remote Controllers independently. For example, the Remote Window of pipe x covers the address range of Remote Controller X, whereas pipe y targets the Transfer Window y of Remote Controller X. In Remote Controller Y, the pipe y targets the available address range. If the Local Controller issues a Write Frame on pipe x, the Remote Controller X is addressed. In case of a Write Frame on pipe y, the Remote Controller Y is targeted, passing through a Transfer Window of Remote Controller X. The two remaining pipes could be used for Write Frames issued by Remote Controller X (passing through a Transfer Window of Remote Controller Y) and by Remote Controller Y.

21.2.4 MLI Service Request Generation

The MLI module's service request outputs SRx are used to indicate module internal MLI events to other modules or devices outside the MLI module, depending on the device implementation. They can trigger interrupts of a CPU (if available), can be used as DMA request lines (if available), or for other trigger purposes. The MLI events being able to trigger interrupts or other service requests, names of some flags and control registers refer to interrupt generation.

MLI module events are generated by event sources in the transmitter and in the receiver. Each event source provides a status flag and an enable bit with software clear capability. In some cases, several event sources are combined to a common event. An MLI event, internally generated by an event source, is stored in a status flag that is located in the interrupt status registers TISR (for transmitter events) or RISR (for receiver events). All event flags can be cleared individually by software write actions to bits located in the interrupt enable registers TIER (for transmitter events) or RIER (for receiver events). These two registers also contain the enable control bits that allow each event source to be enabled/disabled individually for service request activation. Each event can be connected to exactly one of the eight service request outputs SR[7:0] by a 3-bit interrupt node pointer.

One additional register, the Global Interrupt Set Register GINTR, allows each service request output to be activated separately without setting the status flags of the event sources (see **Page 21-58**). This feature is sometimes helpful for software test purposes or to trigger MLI external actions.

Interrupt Registers

The MLI event sources are controlled by several registers (see **Table 21-6** and **Page 21-107**). The register name prefixes "T" and "R" indicate if a register is assigned to the MLI transmitter or to the MLI receiver.

Table 21-6 Interrupt Registers

Unit	Registers with		
	Request Flags	**Enable Bits/ Req. Flag Clear Bits**	**Node Pointer**
MLI Transmitter	TISR	TIER	TINPR
MLI Receiver	RISR	RIER	RINPR

Service Request Compressor

The MLI event logic uses a compressing scheme for flexible service request processing. Eleven MLI events (six transmitter events and four of the five receiver events) are directed via a 3-bit interrupt node pointer to one of the eight service request outputs

SR[7:0]. Each demultiplexer output selected by its Node Pointer = x (x = 0-7) is connected to one input of the SRx OR-Gate. This wiring scheme also supports the connection of more than one event source to an service request output SRx. One receiver event, the interrupt Command Frame event, has a special characteristic: its node pointer is controlled by the received CMD value directly and only SR[3:0] OR-Gates are selectable.

Figure 21-39 shows the service request compressing logic. For reasons of simplicity, not all MLI events, connections, and OR-Gates are explicitly shown. The OR-Gate inputs are connected to the demultiplexers of the MLI event specific lines. Furthermore, a service request output SRx can be triggered by software if the corresponding interrupt set bit in register GINTR is written with a 1.

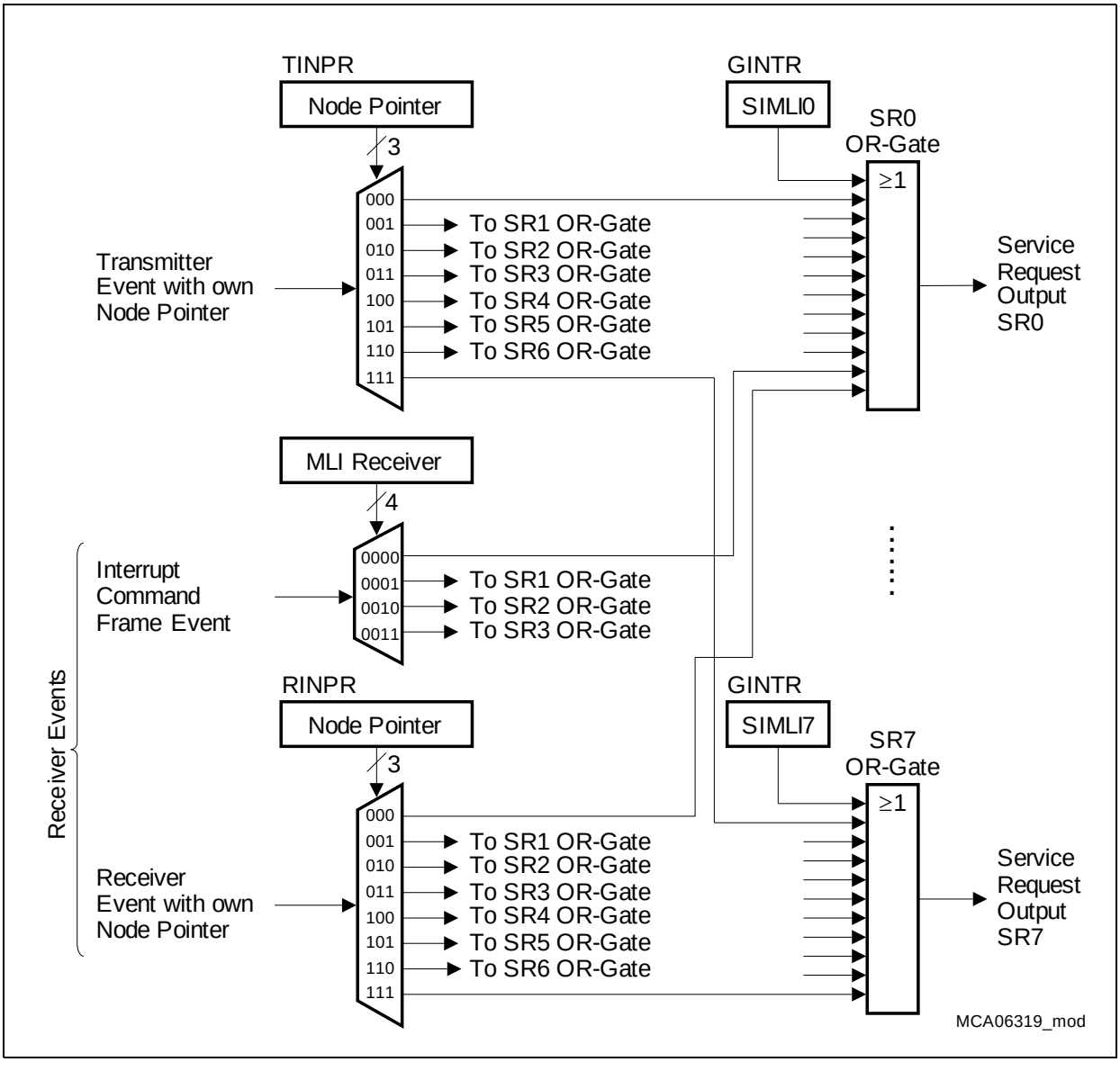


Figure 21-39 Service Request Compressor

Note: The number of SRx outputs of an MLI module and their connection to other modules depends on the implementation of the MLI module in the specific product.

21.2.5 Transmitter Events

The MLI transmitter can generate the following MLI events:

Table 21-7 MLI Transmitter Events

Events	Events combined to	See
Parity Error	Parity/Time-out Error	Page 21-60
Time-out Error		
Normal Frame Sent in Pipe 0	Normal Frame Sent in Pipe 0	Page 21-60
Normal Frame Sent in Pipe 1	Normal Frame Sent in Pipe 1	
Normal Frame Sent in Pipe 2	Normal Frame Sent in Pipe 2	
Normal Frame Sent in Pipe 3	Normal Frame Sent in Pipe 3	
Command Frame Sent in Pipe 0	Command Frame Sent	Page 21-61
Command Frame Sent in Pipe 1		
Command Frame Sent in Pipe 2		
Command Frame Sent in Pipe 3		

21.2.5.1 Parity/Time-out Error Event

A parity/time-out error event is generated when a programmable maximum number of parity errors or a programmable maximum number of Non-Acknowledge errors have been reached. Both events have separate status/control bits but are concatenated to one common error event.

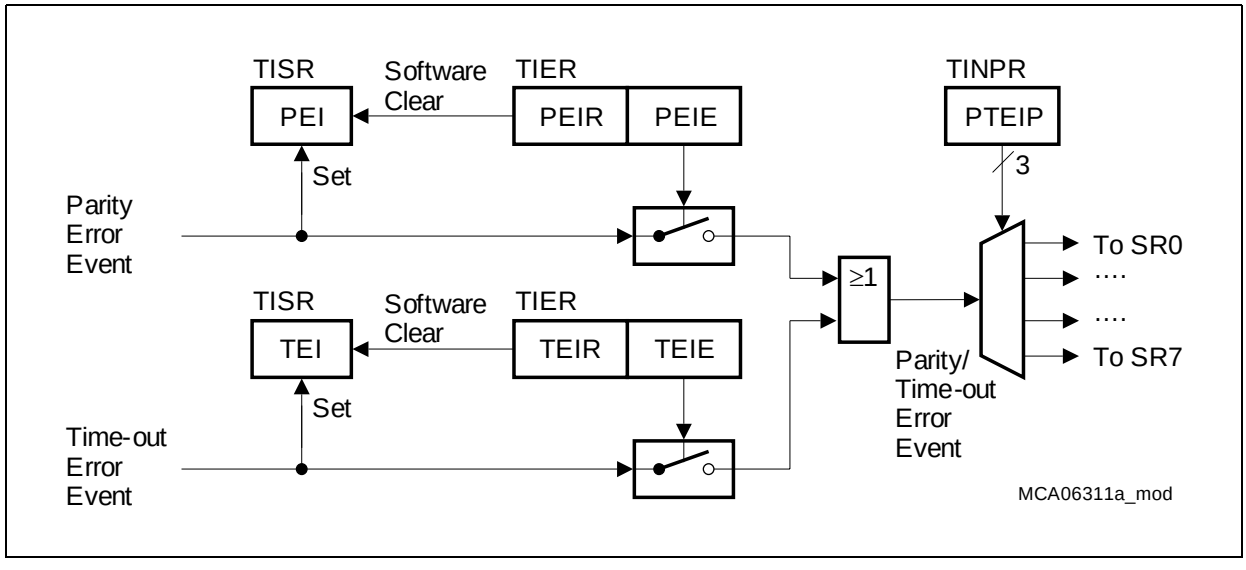


Figure 21-40 Parity/Time-out Error Event Logic

21.2.5.2 Normal Frame Sent x Event

A Normal Frame sent x (x = 0-3) event is generated when a Normal Frame has been sent and correctly received in pipe x.

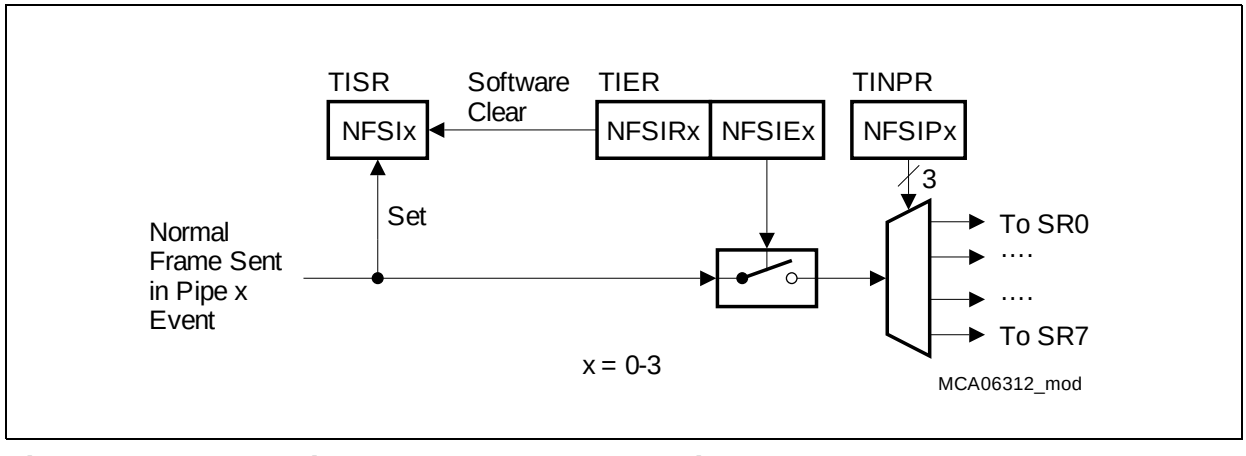


Figure 21-41 Normal Frame Sent x Event Logic

21.2.5.3 Command Frame Sent Events

A Command Frame sent event is generated when the MLI transmitter has sent a Command Frame through pipe x (x = 0-3) that has been correctly received. Separate status/control bits are assigned to each pipe. All four pipe related Command Frame sent events are concatenated to one common Command Frame sent event.

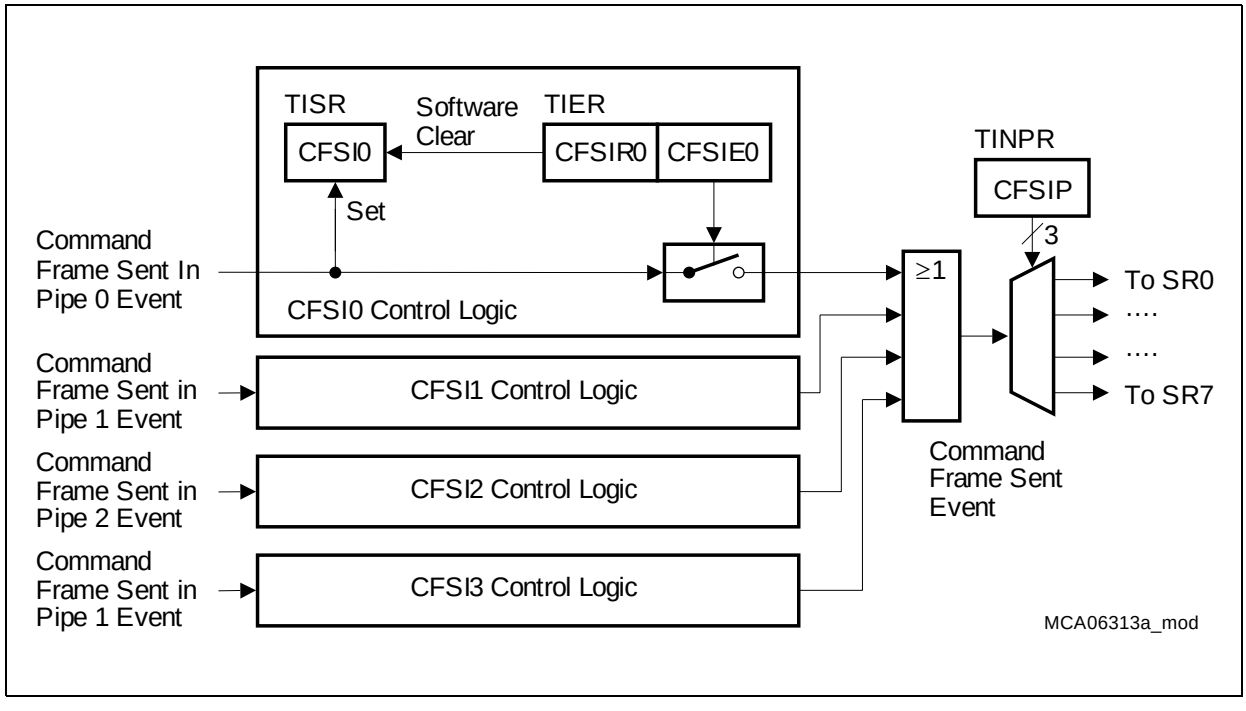


Figure 21-42 Command Frame Sent Event Logic

21.2.6 Receiver Events

The MLI receiver can generate the following MLI events:

Table 21-8 MLI Receiver Interrupts

Events	Events combined to	See
Discarded Read Answer	Discarded Read Answer	Page 21-62
Memory Access Protection Error	Memory Access Protection/ Parity Error	Page 21-63
Parity Error		
Normal Frame Correctly Received	Normal Frame Received	Page 21-64
Move Engine Access Terminated		
Interrupt Command Frame	Interrupt Command Frame	Page 21-65
Command Frame Received on Pipe 0	Command Frame Received	Page 21-66
Command Frame Received on Pipe 1		
Command Frame Received on Pipe 2		
Command Frame Received on Pipe 3		

21.2.6.1 Discarded Read Answer Event

A discarded read answer received event is generated if an Answer Frame has been received and the read pending flag TRSTATR.RPx of its correspondent pipe is 0. Although named “discarded”, the received data is available in the receiver data register until it is overwritten by the next incoming data.

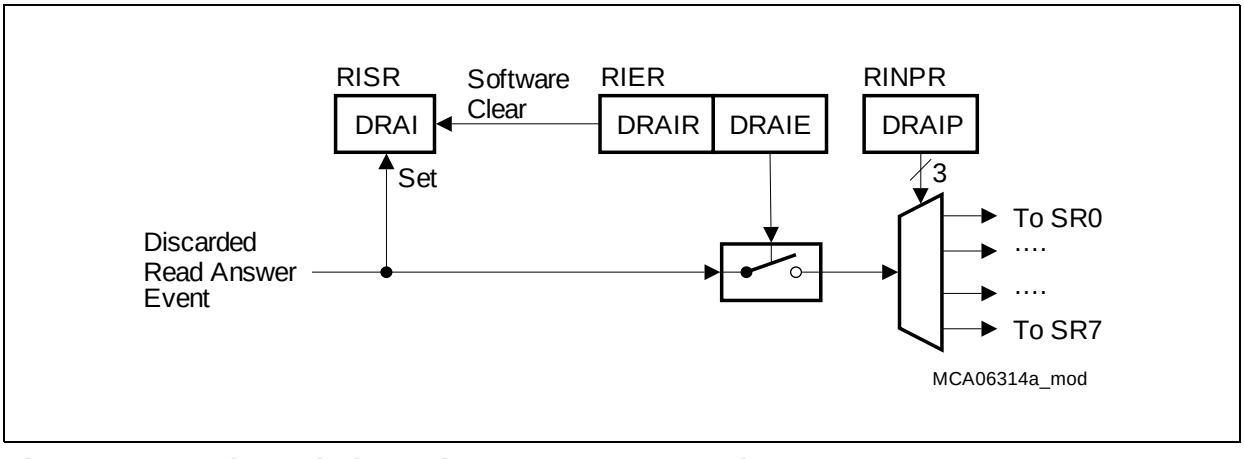


Figure 21-43 Discarded Read Answer Event Logic

21.2.6.2 Memory Access Protection/Parity Error Event

A memory access protection/parity error event is detected if a non allowed read or write access has been detected or if a programmable maximum number of receiver parity errors is reached. Both MLI events have separate status/control bits but are concatenated to one common error event.

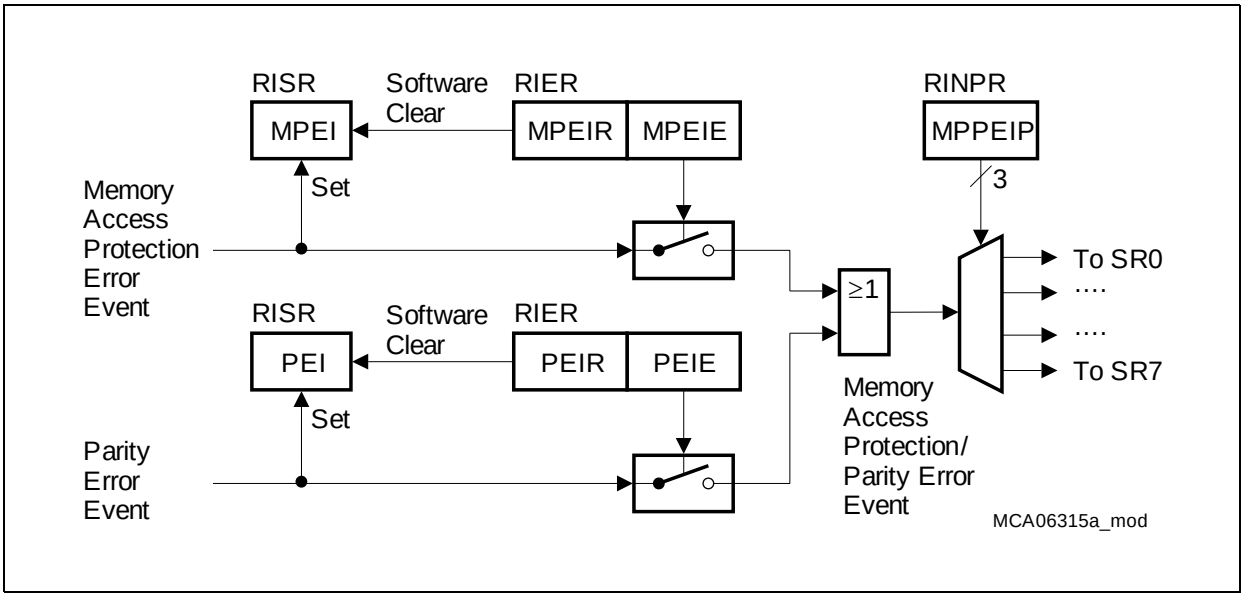


Figure 21-44 Memory Access Protection/Parity Error Event Logic

21.2.6.3 Normal Frame Received/Move Engine Terminated Event

A Normal Frame received event is generated if the MLI receiver has correctly received a Normal Frame (a Copy Base Address Frame, a Read or a Write Frame, an Answer Frame, but not a Command Frame) or if the move engine has terminated its read or write access. Both event sources have separate status/control bits but are concatenated to one common Normal Frame received event.

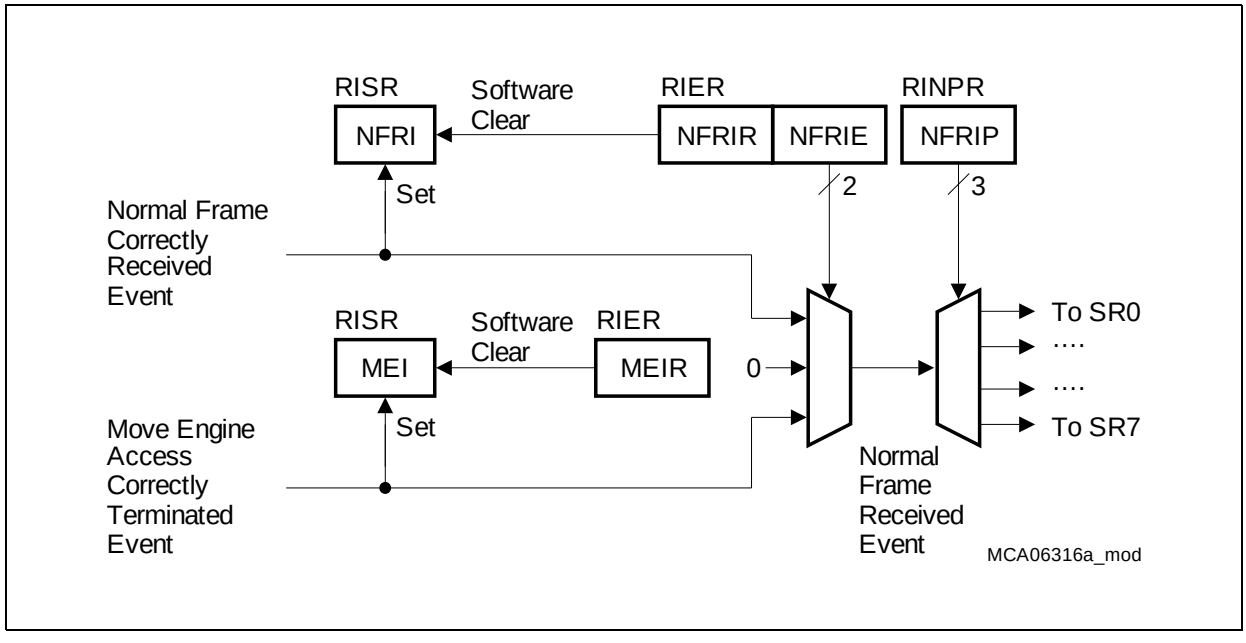


Figure 21-45 Normal Frame Received Event Logic

21.2.6.4 Interrupt Command Frame Event

An interrupt Command Frame event is generated if a Command Frame is received correctly on pipe 0 with a valid command code for service request output activation (CMD = 0000_B to 0011_B). The received command code determines which of the service request outputs SR[3:0] should be activated.

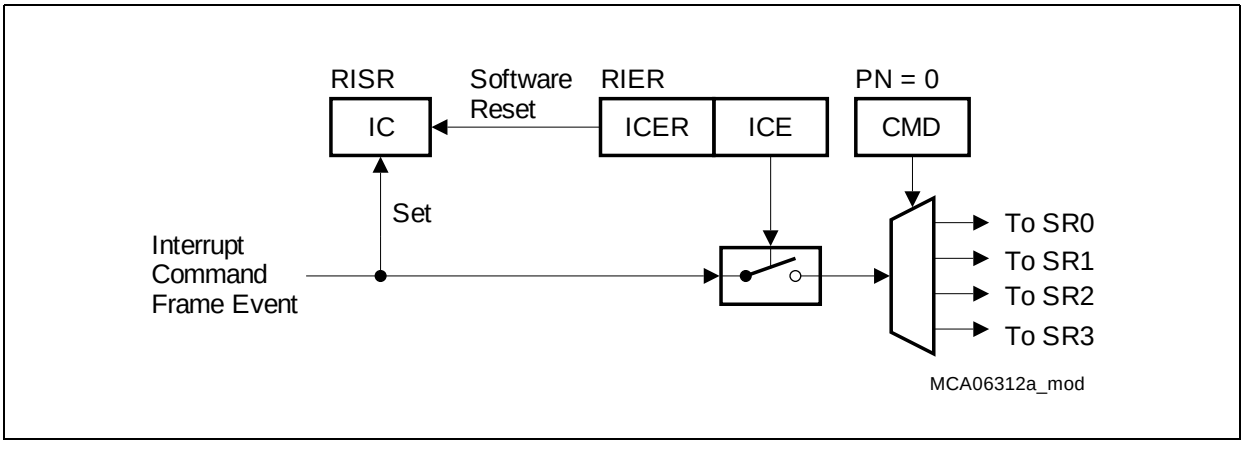


Figure 21-46 Interrupt Command Frame Event Logic

21.2.6.5 Command Frame Received Event

A Command Frame received event is generated if the MLI receiver has correctly received a Command Frame through Pipe Number x (x = 0-3). Separate status/control bits are assigned to each pipe. All four pipe related Command Frame received in pipe x events are concatenated to one common Command Frame received event.

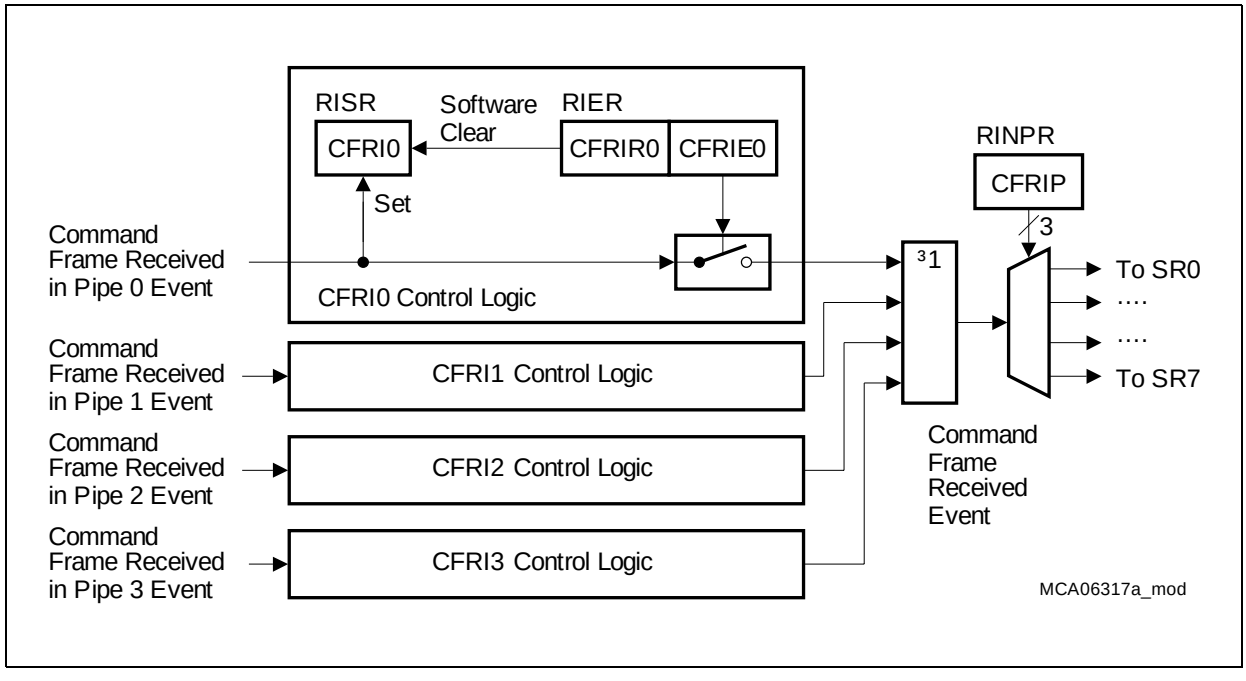


Figure 21-47 Command Frame Received Event Logic

21.2.7 Baud Rate Generation

The MLI transmitter baud rate is given by $f_{MLI}/2$. The MLI shift clock output signal TCLK of the transmitter toggles with each clock cycle of f_{MLI} in order to obtain a 50% duty cycle (the 50% duty cycle can vary up to one clock cycle of f_{SYS} in fractional divider mode). The MLI receiver automatically adapts to the incoming receive shift clock signal RCLK. The received baud rate is determined by the connected transmitter and has no direct relation to f_{SYS} except that it should not exceed f_{SYS}.
The frequency f_{MLI} is generated by the fractional divider FDIV.

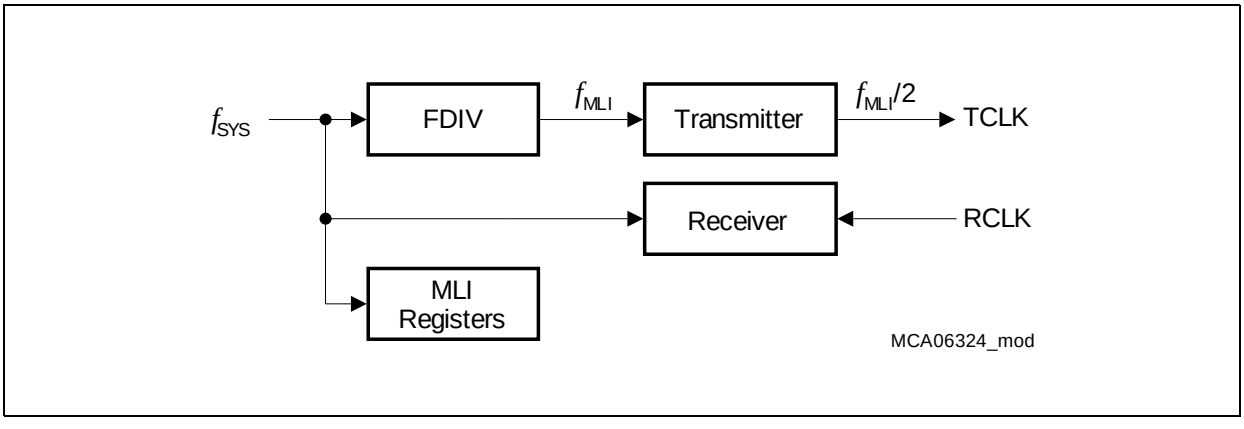


Figure 21-48 MLI Baud Rate Generation

Normal Divider Mode

In normal divider mode (FDR.DM = 01_B) the fractional divider behaves like a reload counter (addition of +1) that generates a clock f_{MLI} on the transition from $3FF_H$ to 000_H. FDR.RESULT represents the counter value and FDR.STEP defines the reload value. In order to achieve $f_{MLI} = f_{SYS}$, FDR.STEP must be programmed with $3FF_H$. The output frequency in normal divider mode is defined according the following equation:

$$f_{MLI} = f_{SYS} \times \frac{1}{1024 - FDR.STEP} \quad (21.1)$$

Fractional Divider Mode

If the fractional divider mode is selected (FDR.DM = 10_B), the clock f_{MLI} is derived from the input clock f_{SYS} by division of a fraction of STEP/1024 for any value of STEP from 0 to 1023. In general, the fractional divider mode allows to program the average clock frequency with a higher accuracy than in normal divider mode. In fractional divider mode a clock pulse f_{MLI} is generated depending on the result of the addition FDR.RESULT + FDR.STEP. The frequency f_{MLI} corresponds to the overflows over $3FF_H$. Note that in fractional divider mode the clock f_{MLI} can have a maximum period jitter of one f_{SYS} clock period. This jitter is not accumulated over several cycles and does not

exceed one cycle of f_{SYS}.
The frequency in fractional divider mode is defined according the following equation:

$$f_{MLI} = f_{SYS} \times \frac{STEP}{1024} \quad (21.2)$$

The baud rate of MLI transmissions equals f_{TCLK}, that is defined by the frequency of clock signal f_{MLI} divided by 2 to create the 50% duty cycle of the shift clock signal TCLK. The signal TCLK toggling with each period of f_{MLI}, a jitter due to fractional dividing is propagated to TCLK.

$$f_{TCLK} = \frac{f_{MLI}}{2} \quad (21.3)$$

21.2.8 Automatic Register Overwrite

The value of register OICR and bit RCR.RCVRST is overwritten by hardware in the next two clock cycles after a reset (first OICR, followed by RCR). The value applied during reset is given in the register description. This automatic overwrite allows adapting the module to different application requirements without changing the module itself. For example, during reset the receiver is set to a defined state and can be used afterwards for reception without the need to modify it by a write action (if the bit RCVRST is modified to 0).

The values applied after the overwrite can be identical to the indicated reset values. Please refer to the implementation chapter for the modified values (see **Page 21-130**).

21.3 Operating the MLI

Data transfer via MLI between a Local Controller and a Remote Controller is only possible if both are initialized correctly by following sequence of 4 steps. Steps 3 and 4 are necessary if the initialization sequence is exclusively controlled by the Local Controller. If both communication partners are able to run initialization software, steps 1 and 2 can be executed separately by both controllers to initialize both transmitters and both receivers.

1. The transmitter of the Local Controller has to be initialized by write actions to the transmitter registers.
2. The pipes from the Local Controller's transmitter to the Remote Controller's receiver and the Remote Controller's receiver have to be initialized.
3. The Remote Controller's transmitter has to be initialized by data write actions from the Local Controller via the Remote Controller's receiver to the Remote Controller's transmitter registers.
4. The pipes from the Remote Controller's transmitter back to the Local Controller's receiver and the Local Controller's receiver have to be initialized. This is done by frames from the Remote Controller's transmitter. These frames are the result of data write actions of the Local Controller to the Remote Controller's transmitter registers.

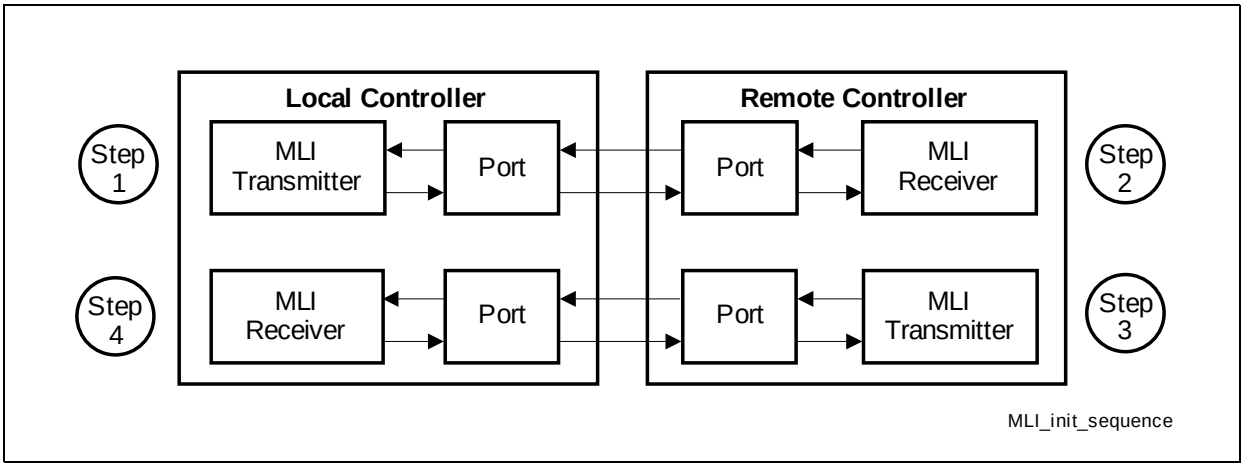


Figure 21-49 Initialization Sequence for an MLI Connection

To initialize and to operate the MLI, the following items should be taken into account:

- Connection setup (see **Page 21-70**)
- Local Controller transmitter and pipe setup (see **Page 21-71**)
- Remote Controller receiver setup (see **Page 21-71**)
- Remote Controller transmitter and Local Controller receiver setup (see **Page 21-72**)
- Delay adjustment (see **Page 21-73**)
- Connection to DMA mechanism (see **Page 21-75**)
- Connection of MLI to SPI (see **Page 21-75**)

21.3.1 Connection Setup

For the general setup of an MLI connection, several steps have to be respected.

- There is the possibility to change the signal routing to adapt to different applications. If another connection than the default one from an input/output vector of the MLI signals is desired, register OICR has to be programmed (see also **Section 21.2.8**).
- In some devices (mainly stand-alone peripheral devices without CPU, where the MLI module is a possible communication channel), the setting “A” can be modified by hardware to another setting (e.g. to setting “B”) during the boot phase. In this case, the initial setting “A” can correspond to an inactive setting (MLI not used for communication), whereas the setting “B” is used for MLI communication.
- In the case a memory access protection is implemented in the receiver and automatic handling of data is desired, the user has to enable the corresponding address range in registers AER and ARR. After a reset, in most microcontrollers, the access protection is generally disabled to avoid access to safety-critical data. Depending on the device, some specific address ranges can already be enabled for automatic access by default.
- In devices with explicit port control (such as microcontrollers), the port pins are generally set to input after a reset. In order to allow MLI communication, the MLI-related port pins have to be configured to make the MLI signals externally available and to adapt the driver setting (refer to port chapter).
  The MLI module should not be enabled for reception (RCR.RCVRST = 1) before programming the desired port setting, because changing the port setting can lead to unintended edges at the module inputs due to setting changes. If the MLI module is already enabled for reception, unintended edges are interpreted as communication signals, so the receiver might deliver wrong results. If this has happened unintentionally, the receiver can be reset by RCR.RCVRST=1.

21.3.2 Local Transmitter and Pipe Setup

The initialization of the transmitter of the Local Controller is done by writing to the transmitter registers. The Remote Controller's MLI receiver can then be initialized by the Local Controller's transmitter.

- After a hardware reset operation, the MLI transmitter is disabled (TCR.MOD = 0). In disabled mode, no frame transmission can take place. After writing TCR.MOD = 1, the transmitter is enabled to send frames.
- The desired transmitter baud rate can be adjusted by the fractional divider FDIV. It has to be ensured that the fractional divider is set to a value that is supported by the port structures of the Local and the Remote Controllers (rise/fall times) and the physical layer. For example, if a division by 1,5 is selected, the fractional divider will deliver count pulses for f_{MLI} with a sequence of 1-2-1-2-1-2- clock cycles of f_{SYS}. The shortest interval between two count pulses in a sequence (given by the truncated divider factor, so 1 cycle of in this example) has to be handled by the communicating devices.f_{SYS}
- Depending on the application requirements, a desired service request output SRx can be activated if a transmitter event is detected.
- The maximum delay for parity error detection in the transmitter has to be programmed. There are two possibilities to get the MLI communication started.
  First (easier) possibility is to write TCR.MDP to 14 and to set RCR.DPE to 15.
  The second possibility could be used to optimize the bandwidth of the MLI connection. It is described in **Section 21.3.5** on **Page 21-73**.

21.3.3 Remote Receiver Setup

The initialization of the Remote Controller's receiver is done by frames sent by the Local transmitter. Therefore, the Remote Controller's receiver has to be able to receive frames.

- In order to allow communication, the Remote Controller's MLI signals have to be connected to the Local Controller's transmitter signals (see register OICR and port settings).
- The Remote Controller's bit RCR.RCVRST has to be 0 to enable frame reception.
- The buffer area size and the base address of the Remote Window for pipe x are defined by the data written to registers TPxBAR. Bit TRSTATR.BAV has to be 0 before each write action to one of these registers. With this information, the buffer area sizes (defining the number of address bits in data frames or Read Frames) are known in the transmitter and in the receiver for each pipe.
  The base addresses for the Remote Windows have to be selected to cover the target address ranges in the Remote Controller. It is recommended to use the minimum buffer size required by the application in order to minimize the bandwidth taken by the transfer of the address bits. The base address of a Remote Window has to be set to a value aligned to its size, e.g. a Remote Window of 8 Kbytes must start at an 8 Kbyte address boundary.

- In devices with access protection mechanism against unauthorized accesses via MLI, the Remote Controller has to enable the desired address range(s) to support automatic mode. If automatic mode is not desired, the Remote Controller has to handle the complete data traffic by software.
- A possibility to test the setup in devices with the capability to run own test software is the local loop back (the transmitter is connected locally to the receiver of the same MLI module). In devices without this capability, the module loop back can be hardly used (or it is even not implemented, refer the connection table in the implementation chapter).
  If implemented, for local loop back, the signal connections have to be programmed to setting “D”, leading to the local receiver being connected directly to the local transmitter (without using a port structure). In this case, the local receiver seems to be the remote receiver. Data written to a local Transfer Window are received and handled by the local receiver. Test software in the Local Controller can check for correct setup, data consistency, MLI event handling, and correct address handling in the Local Controller.
- If automatic data handling is desired (necessary for devices without the capability to handle data traffic by its CPU), the Automatic Data Mode has to be enabled by sending a Command Frame in pipe 2 with CM = 0001_B to set RCR.MOD = 1 in the Remote Controller.

21.3.4 Remote Transmitter and Local Receiver Setup

The initialization of the Remote Controller’s transmitter and the Local Controller’s receiver can be done by data frames sent by the local transmitter. Therefore, the Remote Controller’s receiver has to be able to receive frames (the port structure has to be set up accordingly).

- The Remote Window of pipe x (x can be freely chosen) has to be set to the MLI register address range in the Remote Controller. The initialization by data frames is then done via pipe x.
- The automatic mode has to be enabled in the Remote Controller (Command Frame in pipe 2 with CM = 0001_B).
- The connections between the remote transmitter and the local receiver have to be established (if not already done by the default setting), similar to **Section 21.3.2**.
- The remote transmitter has to be enabled, similar procedure as for the local transmitter. The data word to be written to the Remote Controller’s MLI registers have to be written to the corresponding address in the local Transfer Window of pipe x.
- The local receiver can then be configured by writing the appropriate data (similar scenario as for the remote receiver) to the local Transfer Window of pipe x.
- A possibility to test the complete setup is the remote loop back. In this case another Remote Window is overlaid directly to a Transfer Window in the Remote Controller. Writing data to the corresponding Transfer Window in the Local Controller leads to a data frame sent to the Remote Controller. There, the received data is written to the

Transfer Window and a new data frame is sent back to the Local Controller. The MLI move engine in the Local Controller's receiver can be used to write the received data to a defined location, e.g. to a memory location. Test software in the Local Controller can check for correct setup, data consistency, MLI event handling, and correct address handling in the Local and the Remote Controllers.

21.3.5 Delay Adjustment

The local MLI transmitter is measuring the number of TCLK clock cycles between TVALID becoming 0 after a transmission and TREADY becoming 1 again. This time represents the overall loop delay of the MLI connection. The loop delay is the time used for signal propagation, input/output driver delay and remote receiver reaction. For example, with slow drivers and a high load (due to long wires, etc.), the signals take a longer time to propagate from the local transmitter to the remote receiver and back again (READY-VALID control handshake). This delay (also visible when TVALID becomes 1 at the beginning of a frame) limits the maximum baud rate of an MLI connection, because the answer of the receiver has to be detected by the transmitter with TREADY = 0 at the end of the frame. The value measured after the end of the frame is indicated in bit field TSTATR.RDC.

The receiver participates in the control handshake by changing its RREADY output as a reaction to an incoming RVALID signal. For the transmitter, the TREADY input delivers the information that a receiver is connected and that it is ready for reception (transfer only starts if TREADY = 1). If a receiver is not able to handle the data or is not connected, the TREADY line will not become low after TVALID becomes 1 (Non-Acknowledge).

In addition to this information, the MLI protocol offers the possibility to use the control handshake also to indicate that the receiver has detected a parity error in the received frame. If a correct frame has been received, the receiver immediately asserts RREADY = 1 after the reception of a frame when detecting RVALID = 0. If the receiver has detected a parity error, it waits for a programmable number of RCLK cycles before setting RREADY = 1 again. This additional delay is defined by bit field RCR.DPE.

The transmitter measuring the delay and comparing it to a programmed value, it can detect that the receiver has signaled a parity error by introducing the additional delay. The compare value for the transmitter is programmed by bit field TCR.MDP. A measured value of TSTATR.RDC above TCR.MDP is interpreted as parity error by the transmitter (for parity error handling refer to **Page 21-44**).

In the receiver, frames with parity error are ignored for data transfers and don't lead to internal move actions.

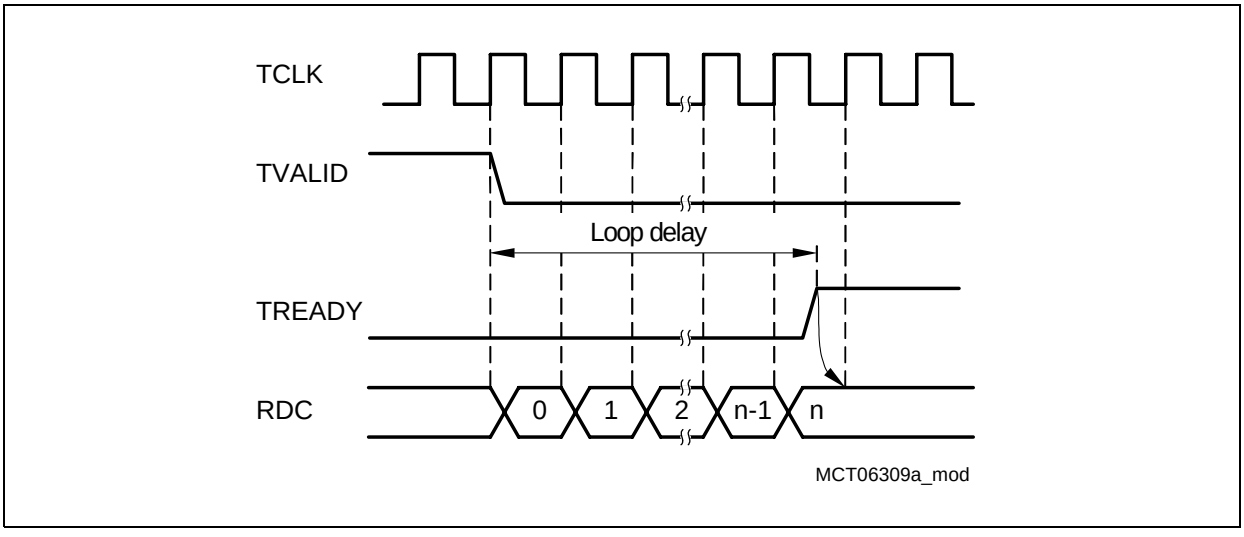


Figure 21-50 Loop Delay Measurement

To adjust the generated parity delay in the local transmitter and in the remote receiver, the following steps are necessary:

- Send a dummy frame to the receiver for measuring the loop delay. This frame should not lead to internal data move actions in the receiver, so a parity error can be simulated in the transmitter. The receiver has a fixed even parity scheme, whereas the transmitter can be programmed either for even or for odd parity. Programming odd parity before sending a frame will generate a (dummy) frame that will be discarded by the receiver (assuming a correct transfer). For a dummy frame, it is recommended to use a data frame with disabled Automatic Data Mode in the receiver (RCR.MOD = 0).
- The receiver delay RCR.DPE being 0 after a module reset, the transmitter can measure the loop delay and the receiver discards the frame (without modification of DPE, there is no difference in time between a frame with or without a parity error having been detected). The value given by TSTATR.RDC indicates how many TCLK cycles are necessary for a control handshake. This value should be incremented by a value DELTA (value see below) and written to TCR.MDP.
- The transmitter parity has to be programmed to even parity to be able to generate frames that are not discarded by the receiver.
- Programming the receiver delay for parity error (RCR.DPE) to a value bigger than DELTA will lead to a value of TSTATR.RDC bigger than TCR.MDP if the receiver detects a parity error. The value of DPE in the remote receiver is modified by the local transmitter by sending a Command Frame in pipe 1 with the desired value. The difference between TSTATR.RDC and TCR.MDP allows a certain timing tolerance between local transmitter and remote receiver.
- The value of DELTA depends on the possible variations of the propagation characteristics of the MLI connection. If the environment does not significantly change, DELTA can be 1. For systems with variations, DELTA could be bigger. The

user can check about changing propagation characteristics by reading TSTATR.RDC from time to time and to check if it is constant for correct transfers. If it changes, either a bigger DELTA value can be applied, or the delay adjustment can be repeated, adapting to the new circumstances.

21.3.6 Connection to DMA Mechanism

The MLI module supports the connection to a DMA (direct memory access) mechanism. This mechanism allows the transfer of blocks of data of programmable size via an MLI connection without CPU intervention. Therefore, a DMA mechanism can be used in the Local Controller to write the desired number of data words one after the other to the corresponding MLI Transfer Window. The address ranges of the data blocks and their length has to be handled by the DMA module.

An MLI pipe supporting only one pending Write Frame request at a time, the DMA has to wait until the pipe is capable to handle new data before writing another data word to the Transfer Window. Therefore, the Normal Frame sent events of the pipes can trigger DMA data transfers. Depending on the connection of the MLI module's service request outputs SRx to the DMA trigger inputs, the Normal Frame sent events have to be enabled for service request activation and directed to the desired SRx outputs. It is recommended to use only one type of MLI event per SRx output to trigger a data transfer by DMA. If the DMA mechanism needs a start trigger for the first data word transfer, register GINTR can be written with the appropriate pattern to activate an SRx output.

21.3.7 Connection of MLI to SPI

The handshake signals between a transmitter and a receiver are based on a synchronous transfer protocol. In the SPI protocol, the shift clock and the data signal are equivalent to CLK and DATA. In case of an 4-wire SPI, the slave select signal represents the VALID signal (the leading and the trailing delay have to be set up accordingly).

Contrary to the MLI, in the SPI protocol, a complete control handshake is not defined, so the READY signal does not exist in SPI modules. As a result, the SPI communication does not check by hardware for correct data transfer, but has to handle this on an upper software layer. If using an SPI module for communication with an MLI transmitter or an MLI receiver, the READY signal has to be handled by software or the handshake has to be given up. This can be done by connecting the TVALID signal of an MLI transmitter to one of its own TREADY inputs with polarity inversion. Like this, the TREADY input directly following the inverted TVALID signal, the parity error indication and the Non-Acknowledge error detection are not possible.

Furthermore, in the MLI protocol, the frames may have a different width, depending on their type and selected buffer size. The different numbers of data bits per frame have also to be handled by the SPI module. In order to minimize the number of different frames, it is recommended to restrict the possibility to program different buffer sizes, the use of Read Frames or Command Frames. In order to simplify the data handling by an

SPI module, the parity generation could be skipped for frames received by the SPI module and an error detection mechanism on an upper software layer could be implemented. For frames sent by an SPI module, the parity bit has to be calculated and sent correctly. Otherwise, the MLI receiver will discard the received frame.

21.4 MLI Kernel Registers

This section describes the kernel registers of the MLI module. All registers can be accessed with 8-bit, 16-bit or 32-bit write or read operations. Accesses to address locations inside the MLI address range not targeting the indicated registers are not allowed. The complete and detailed address map of the of the MLI module and its registers is described in **Table 21-14** on **Page 21-139**.

All registers in the MLI address spaces are reset with the application reset.

MLI Kernel Register Overview

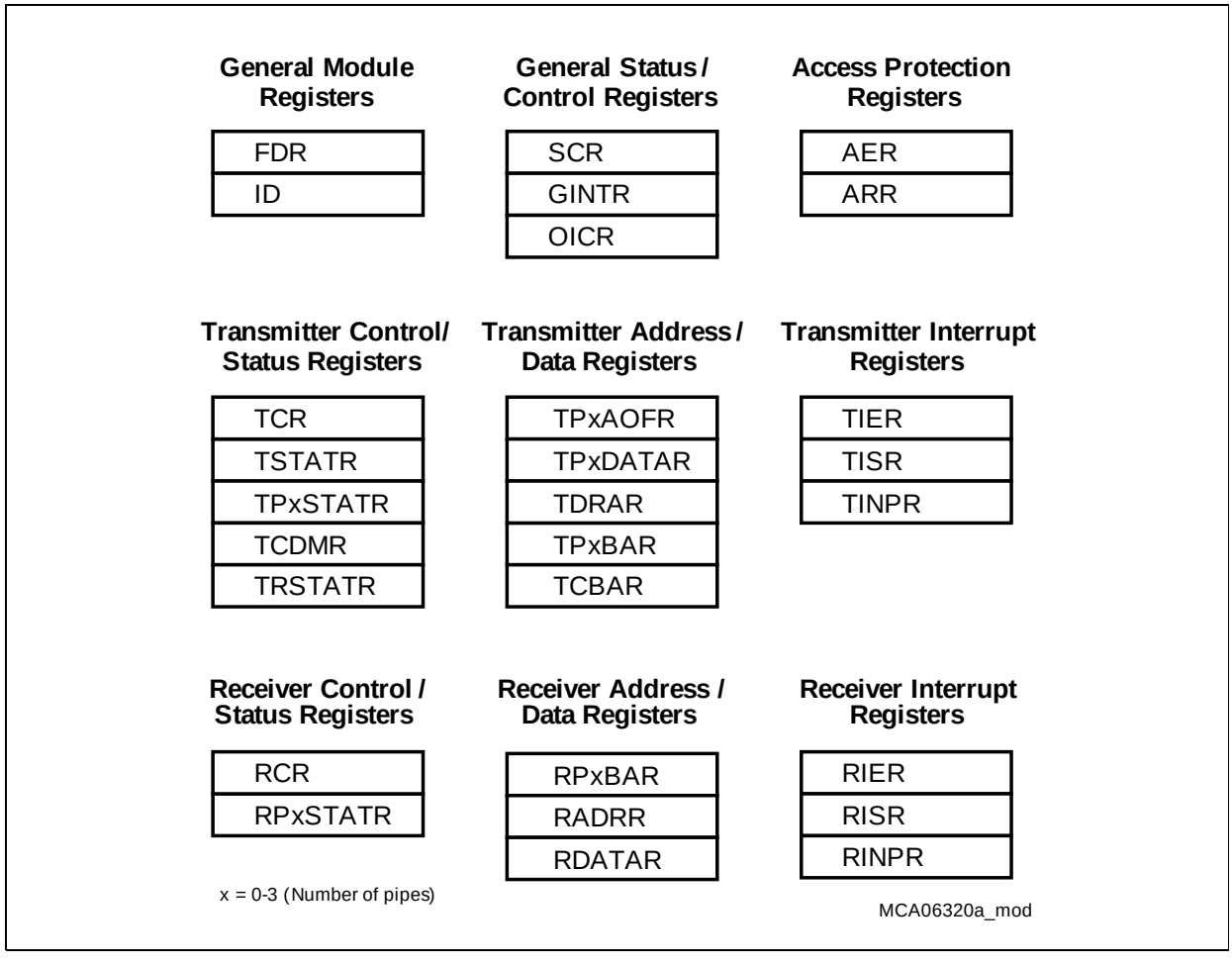


Figure 21-51 MLI Kernel Registers

Table 21-9 Registers Address Space - MLI Kernel Registers

Module	Base Address	End Address	Note
MLI0	$F010\ C000_H$	$F010\ C0FF_H$	–
MLI1	$F010\ C100_H$	$F010\ C1FF_H$	–

Table 21-10 Registers Overview - MLI Kernel Registers

Register Short Name	Register Long Name	Offset Address[1)]	Description see
ID	Module Identification Register	08_H	Page 21-82
FDR	Fractional Divider Register	$0C_H$	Page 21-79
TCR	Transmitter Control Register	10_H	Page 21-92
TSTATR	Transmitter Status Register	14_H	Page 21-95
TPxSTATR	Transmitter Pipe x Status Register	18_H + (x * 4)	Page 21-97
TCMDR	Transmitter Command Register	28_H	Page 21-99
TRSTATR	Transmitter Receiver Status Register	$2C_H$	Page 21-101
TPxAOFR	Transmitter Pipe x Address Offset Register	30_H + (x * 4)	Page 21-103
TPxDATAR	Transmitter Pipe x Data Register	40_H + (x * 4)	Page 21-104
TDRAR	Transmitter Data Read Answer Register	50_H	Page 21-104
TPxBAR	Transmitter Pipe x Base Address Register	54_H + (x * 4)	Page 21-105
TCBAR	Transmitter Copy Base Address Register	64_H	Page 21-106
RCR	Receiver Control Register	68_H	Page 21-113
RPxBAR	Receiver Pipe x Base Address Register	$6C_H$ + (x * 4)	Page 21-117
RPxSTATR	Receiver Pipe x Status Register	$7C_H$ + (x * 4)	Page 21-116
RADDR	Receiver Address Register	$8C_H$	Page 21-118
RDATAR	Receiver Data Register	90_H	Page 21-119
SCR	Set Clear Register	94_H	Page 21-84
TIER	Transmitter Interrupt Enable Register	98_H	Page 21-107
TISR	Transmitter Interrupt Status Register	$9C_H$	Page 21-109
TINPR	Transmitter Interrupt Node Pointer Register	$0A0_H$	Page 21-111
RIER	Receiver Interrupt Enable Register	$A4_H$	Page 21-120
RISR	Receiver Interrupt Status Register	$A8_H$	Page 21-123
RINPR	Receiver Interrupt Node Pointer Register	AC_H	Page 21-125
GINTR	Global Interrupt Set Register	$B0_H$	Page 21-83
OICR	Output Input Control Register	$B4_H$	Page 21-86
AER	Access Enable Register	$B8_H$	Page 21-90
ARR	Access Range Register	BC_H	Page 21-91

1) The absolute register address is calculated as follows:
Module Base Address (**Table 21-9**) + Offset Address (shown in this column)

21.4.1 General Module Registers

Fractional Divider Register

The fractional divider register allows to program the frequency f_{MLI} to generate the baud rate of the of the 50% duty cycle transmitter shift clock TCLK.

FDR
Fractional Divider Register ($0C_H$) Reset Value: 03FF 43FF$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
DIS CLK	EN HW	SUS REQ	SUS ACK	0		RESULT									
rwh	rw	rh	rh	r		rh									

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DM		SC		SM	0	STEP									
rw		rw		rw	r	rw									

Field	Bits	Type	Description
STEP	[9:0]	rw	**Step Value** In normal divider mode STEP contains the reload value for RESULT. In fractional divider mode this bit field defines the 10-bit value that is added to the RESULT with each input clock cycle.
SM	11	rw	**Suspend Mode** SM selects between granted or immediate suspend mode. This bit is only taken into account in devices supporting suspend mode. 0_B Granted suspend mode selected 1_B Immediate suspend mode selected

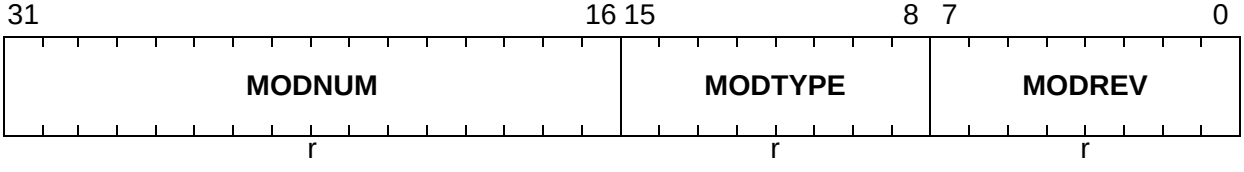
Field	Bits	Type	Description
SC	[13:12]	rw	**Suspend Control** This bit field defines the behavior of the fractional divider in suspend mode (bit SUSREQ and SUSACK set). This bit field is only taken into account in devices supporting suspend mode. 01_B Clock generation is stopped and the clock output signals are not generated. RESULT is not changed except when writing bit field DM with 01_B or 10_B. 00_B Clock generation continues. 10_B Clock generation is stopped and the clock output signals are not generated. RESULT is loaded with $3FF_H$. 11_B Same as SC = 10_B but RST_EXT_DIV is 1 (independently of bit field DM).
DM	[15:14]	rw	**Divider Mode** This bit fields defines the functionality of the fractional divider block. 00_B Fractional divider is switched off; no output clock is generated. RST_EXT_DIV is 1. RESULT is not updated (default after reset). 01_B Normal divider mode selected. 10_B Fractional divider mode selected. 11_B Fractional divider is switched off; no output clock is generated. RESULT is not updated.
RESULT	[25:16]	rh	**Result Value** In normal divider mode RESULT acts as reload counter (addition +1). In fractional divider mode this bit field contains the result of the addition RESULT+STEP. If DM is written with 01_B or 10_B, RESULT is loaded with $3FF_H$.
SUSACK	28	rh	**Suspend Mode Acknowledge** 0_B Suspend mode is not acknowledged. 1_B Suspend mode is acknowledged. Suspend mode is entered when SUSACK and SUSREQ are set.

Field	Bits	Type	Description
SUSREQ	29	rh	**Suspend Mode Request** 0_B Suspend mode is not requested. 1_B Suspend mode is requested. Suspend mode is entered when SUSACK and SUSREQ are set.
ENHW	30	rw	**Enable Hardware Clock Control** 0_B Bit DISCLK cannot be cleared by hardware by a high level at input signal ECEN. 1_B Bit DISCLK is cleared by hardware while input signal ECEN is at high level.
DISCLK	31	rwh	**Disable Clock** 0_B Clock generation of $f_{OUT} = f_{MLI}$ is enabled according to the setting of bit field DM. 1_B Fractional divider is stopped. Signal $f_{OUT} = f_{MLI}$ becomes inactive. No change except when writing bit field DM. In case of a conflict between hardware reset and software set of DISCLK, the software set wins. Any write or read-modify-write action leads to the described behavior. As a result read-modify-write operations should be avoided.
0	10, [27:26]	r	**Reserved** Read as 0; should be written with 0.

Module Identification Register

The MLI Module Identification Register ID contains read-only information about the module version.

ID
Module Identification Register (08$_H$) Reset Value: 0025 C007$_H$



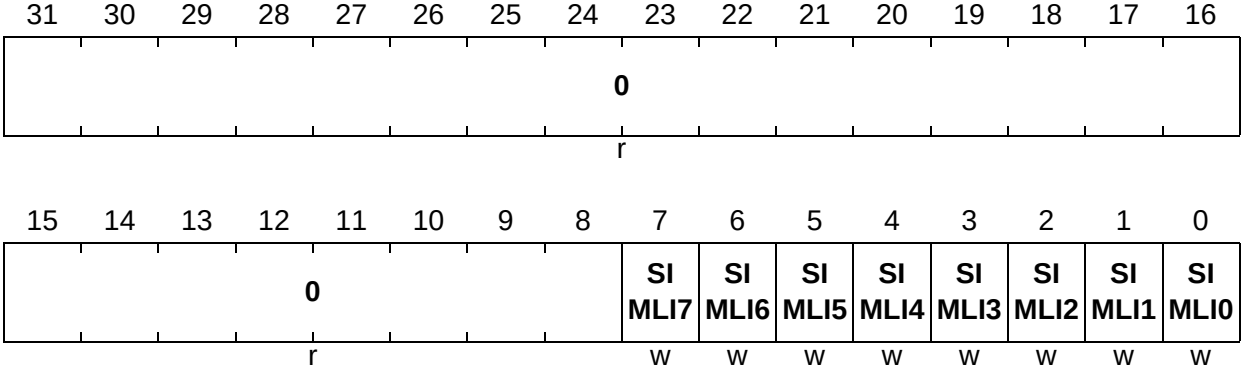
Field	Bits	Type	Description
MODREV	[7:0]	r	**Module Revision Number** This bit field defines the module revision number. The value of a module revision starts with 01$_H$ (first revision).
MODTYPE	[15:8]	r	**Module Type** This bit field defines the module as a 32-bit module: C0$_H$
MODNUM	[31:16]	r	**Module Number Value** This bit field defines the module identification number for the MLI: 0025$_H$

21.4.2 General Status/Control Registers

Global Interrupt Set Register

The Global Interrupt Set Register GINTR is a write only register (always reads 0) that allows each of the service request outputs SRx to be activated under software control (see **Page 21-58**).

GINTR
Global Interrupt Set Register (B0$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
SIMLIx (x = 0-7)	x	w	**Set MLI Service Request Output Line x** 0_B No action 1_B Service request output SRx is activated (pulse).
0	[31:8]	r	**Reserved** Read as 0; should be written with 0.

Set Clear Register

The Set Clear Register SCR is a write only register that makes it possible to set or clear by software several status flags located in registers TSTATR, TRSTATR and RCR. Reading register SCR always returns zeros at all bit locations. Bits that are not written with a 1 have no effect.

SCR
Set Clear Register **(94_H)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
C CIV3	C CIV2	C CIV1	C CIV0	C NAE	C TPE	C RPE	C AV	0						C BAV	C MOD
w	w	w	w	w	w	w	w	w						w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C CV3	C CV2	C CV1	C CV0	C DV3	C DV2	C DV1	C DV0	0			S MOD	S CV3	S CV2	S CV1	S CV0
w	w	w	w	w	w	w	w	w			w	w	w	w	w

Field	Bits	Type	Description
SCV0, **SCV1,** **SCV2,** **SCV3**	0, 1, 2, 3	w	**Set Command Valid** 0_B No effect 1_B Bit TRSTATR.CVx is set.
SMOD	4	w	**Set MOD Flag** 0_B No effect 1_B If CMOD = 0, RCR is set. If CMOD = 1, RCR.MOD is cleared.
CDV0, **CDV1,** **CDV2,** **CDV3**	8, 9, 10, 11	w	**Clear Data Valid x Flag** 0_B No effect 1_B Bits TRSTATR.DVx and TRSTATR.RPx are cleared.
CCV0, **CCV1,** **CCV2,** **CCV3**	12, 13, 14, 15	w	**Clear Command Valid x Flag** 0_B No effect. 1_B If SCVx = 0, bit TRSTATR.CVx is cleared. If SCVx = 1, bit TRSTATR.CVx is set.
CMOD	16	w	**Clear MOD Flag** 0_B No effect. 1_B Bit RCR.MOD is cleared.

Field	Bits	Type	Description
CBAV	17	w	**Clear BAV Flag** 0_B No effect. 1_B Bit TRSTATR.BAV is cleared.
CAV	24	w	**Clear AV Flag** 0_B No effect. 1_B Bit TRSTATR.AV is cleared.
CRPE	25	w	**Clear Receiver PE Flag** 0_B No effect. 1_B Bit RCR.PE is cleared.
CTPE	26	w	**Clear Transmitter PE Flag** 0_B No effect. 1_B Bit TSTATR.PE is cleared.
CNAE	27	w	**Clear NAE Flag** 0_B No effect. 1_B Bit TSTATR.NAE is cleared.
CCIV0, CCIV1, CCIV2, CCIV3	28, 29, 30, 31	w	**Clear Command Interrupt Valid x Flag** 0_B No effect. 1_B Bit TSTATR.CIVx is cleared.
0	[7:5], [23:18]	w	**Reserved** Read as 0; should be written with 0.

Output Input Control Register

The Output Input Control Register OICR determines the functionality of the MLI transmitter and MLI receiver I/O control logic. The bits in this register are automatically overwritten after a reset with a value given in the implementation chapter (see **Page 21-130**). Furthermore, the connection table of the MLI module signals is given there.

Note: The value of register OICR should not be modified while a data transfer (reception or transmission) is ongoing (bits in OICR directly control the I/O signal paths).

OICR
Output Input Control Register ($B4_H$) Reset Value: $1000\ 8000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RDP	**RDS**		**RCE**	**RCP**	**RCS**		**RVP**	**RVS**		**RRP D**	**RRP C**	**RRP B**	**RRP A**	**RRS**	
rw	rw		rw	rw	rw		rw	rw		rw	rw	rw	rw	rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RVE	**TDP**	**TCP**	**TCE**	**TRE**	**TRP**	**TRS**		**TVP D**	**TVP C**	**TVP B**	**TVP A**	**TVE D**	**TVE C**	**TVE B**	**TVE A**
rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
TVEA, TVEB, TVEC, TVED	0, 1, 2, 3	rw	**Transmitter Valid Enable** These bits enable the module kernel output signals TVALIDx (x = A, B, C, D) to be driven by MLI transmitter output signal TVALID. 0_B TVALIDx is disabled and remains at passive level (as selected by TVPx). 1_B Transmitter output signal TVALIDx is enabled and driven by TVALID.
TVPA, TVPB, TVPC, TVPD	4, 5, 6, 7	rw	**Transmitter Valid Polarity** These bits determine the polarity of the module kernel transmitter output signals TVALIDx (x = A, B, C, D). 0_B Non-inverted polarity for TVALIDx selected: TVALIDx is passive when driving a 0. TVALIDx is active when driving a 1. 1_B Inverted polarity for TVALIDx selected: TVALIDx is passive when driving a 1. TVALIDx is active when driving a 0.

Field	Bits	Type	Description
TRS	[9:8]	rw	**Transmitter Ready Selection** This bit field determines the module kernel input signal TREADYx (x = A, B, C, D) that is used as MLI transmitter input signal TREADY. 00_B TREADYA is selected. 01_B TREADYB is selected. 10_B TREADYC is selected. 11_B TREADYD is selected.
TRP	10	rw	**Transmitter Ready Polarity** This bit determines the polarity of TREADYx. 0_B Non-inverted polarity for TREADYx selected: TREADYx is passive if 0. TREADYx is active if 1. 1_B Inverted polarity for TREADYx selected: TREADYx is passive if 1. TREADY is active if 0.
TRE	11	rw	**Transmitter Ready Enable** This bit enables the MLI transmitter input signal TREADY. 0_B TREADY signal is disabled (always at 0 level). 1_B TREADY signal is enabled and driven by TREADYx according to the settings of TRS and TRP.
TCE	12	rw	**Transmitter Clock Enable** This bit enables the module kernel output signal TCLK. 0_B TCLK is disabled and remains at passive level (as selected by TCP). 1_B TCLK is enabled and driven according to the setting of TCP.
TCP	13	rw	**Transmitter Clock Polarity** This bit determines the polarity of the module kernel output clock signal TCLK. 0_B Non-inverted polarity for TCLK selected: TCLK is driving a 0 when it is passive. 1_B Inverted polarity for TCLK selected: TCLK is driving a 1 when it is passive.

Field	Bits	Type	Description
TDP	14	rw	**Transmitter Data Polarity** This bit determines the polarity of the module kernel output clock signal TDATA. 0_B TDATA is directly driven by MLI transmitter output signal TDATA (non-inverted). 1_B TDATA is directly driven by the inverted MLI transmitter output signal TDATA.
RVE	15	rw	**Receiver Valid Enable** This bit enables the MLI receiver input signal RVALID. 0_B RVALID signal is disabled (always at 0 level). 1_B RVALID signal is enabled and driven by RVALIDx according to the settings of RVS and RVP (default after reset).
RRS	[17:16]	rw	**Receiver Ready Selector** This bit field determines the module kernel output signal RREADYx (x = A, B, C, D) that is driven by the MLI receiver output signal RREADY. The RREADYx output signals that are not selected drives a passive level according to the setting of RRPx. 00_B RREADYA is selected. 01_B RREADYB is selected. 10_B RREADYC is selected. 11_B RREADYD is selected.
RRPA, RRPB, RRPC, RRPD	18, 19, 20, 21	rw	**Receiver Ready Polarity** These bits determine the polarity of the module kernel receiver output signals RREADYx (x = A, B, C, D). 0_B Non-inverted polarity for RREADYx selected: RREADYx is passive if 0. RREADYx is active if 1. 1_B Inverted polarity for RREADYx selected: RREADYx is passive if 1. RREADYx is active if 0.
RVS	[23:22]	rw	**Receiver Valid Selector** This bit field determines the module kernel input signal RVALIDx (x = A, B, C, D) that is used as MLI receiver input signal RVALID. 00_B RVALIDA is selected. 01_B RVALIDB is selected. 10_B RVALIDC is selected. 11_B RVALIDD is selected.

Field	Bits	Type	Description
RVP	24	rw	**Receiver Valid Polarity** This bit determines the polarity of RVALIDx. 0_B Non-inverted polarity for RVALIDx selected: RVALIDx is passive if 0. RVALIDx is active if 1. 1_B Inverted polarity for RVALIDx selected: RVALIDx is passive if 1. RVALIDx is active if 0.
RCS	[26:25]	rw	**Receiver Clock Selector** This bit field determines the module kernel input signal RCLKx (x = A, B, C, D) that is used as MLI receiver input clock CLK. 00_B RCLKA is selected. 01_B RCLKB is selected. 10_B RCLKC is selected. 11_B RCLKD is selected.
RCP	27	rw	**Receiver Clock Polarity** This bit determines the polarity of RCLKx. 0_B Non-inverted polarity for RCLKx selected: RCLKx is at 0 level in passive state. 1_B Inverted polarity for TCLK selected: RCLKx is at 1 level in passive state.
RCE	28	rw	**Receiver Clock Enable** This bit enables the MLI receiver input clock RCLK. 0_B RCLK signal is disabled (always at 0 level). 1_B RCLK signal is enabled and driven by RCLKx according to the settings of RCS and RCP.
RDS	[30:29]	rw	**Receiver Data Selector** This bit field determines the module kernel input signal RDATAx (x = A, B, C, D) that is used as MLI receiver data input line RDATA. 00_B RDATAA is selected. 01_B RDATAB is selected. 10_B RDATAC is selected. 11_B RDATAD is selected.
RDP	31	rw	**Receiver Data Polarity** This bit determines the polarity of RDATAx. 0_B Non-inverted polarity for RDATAx selected: RDATAx is passive if 0. RDATAx is active if 1. 1_B Inverted polarity for RDATAx selected: RDATAx is passive if 1. RDATAx is active if 0.

21.4.3 Access Protection Registers

Access Enable Register

The Access Enable Register AER enables write and read operations in the corresponding address ranges (x = 0 to 31) in addition to the global move engine enable RCR.MOD. Each address range can be individually enabled or excluded from automatic mode.

*Note: Please refer to the implementation chapter for the device specific access protection (see **Page 21-137**).*

AER
Access Enable Register **($B8_H$)** **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AEN 31	**AEN 30**	**AEN 29**	**AEN 28**	**AEN 27**	**AEN 26**	**AEN 25**	**AEN 24**	**AEN 23**	**AEN 22**	**AEN 21**	**AEN 20**	**AEN 19**	**AEN 18**	**AEN 17**	**AEN 16**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

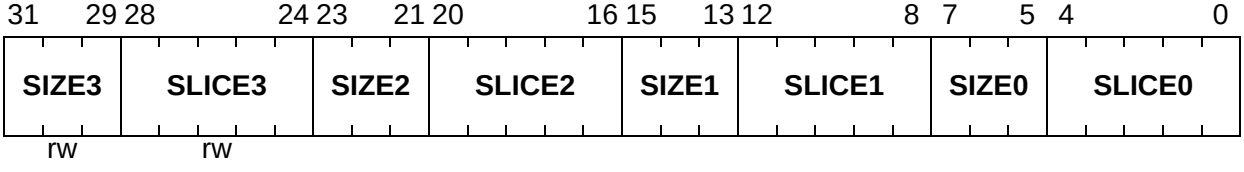
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AEN 15	**AEN 14**	**AEN 13**	**AEN 12**	**AEN 11**	**AEN 10**	**AEN 9**	**AEN 8**	**AEN 7**	**AEN 6**	**AEN 5**	**AEN 4**	**AEN 3**	**AEN 2**	**AEN 1**	**AEN 0**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
AENx (x = 0-31)	x	rw	**Address Range x Enable** This bit enables the read and write capability of the MLI move engine for address range x (x = 0-31). 0_B Automatic MLI read and write moves to address range x are disabled. Read/write moves to address range x are not executed automatically and an MLI service request can be generated. The receiving controller's software has to take care about the move. 1_B Automatic MLI read and write moves to address range x are enabled if RCR.MOD = 1.

Access Range Register

The Access Range Register ARR determines size and number of the address sub-range n (n = 0-3).

ARR
Access Range Register **(BC_H)** **Reset Value: 0000 0000$_H$**



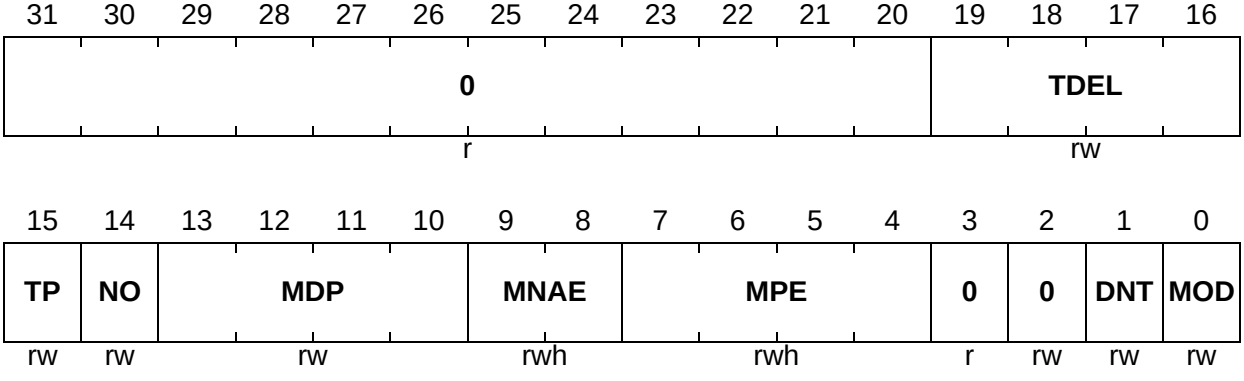
Field	Bits	Type	Description
SLICE0	[4:0]	rw	**Address Slice 0** SLICE0 selects a specific sub-range within address sub-range 0.
SIZE0	[7:5]	rw	**Address Size 0** SIZE0 determines the sub-range size within address sub-range 0.
SLICE1	[12:8]	rw	**Address Slice 1** SLICE1 selects a specific sub-range within address sub-range 1.
SIZE1	[15:13]	rw	**Address Size 1** SIZE1 determines the sub-range size within address sub-range 1.
SLICE2	[20:16]	rw	**Address Slice 2** SLICE2 selects a specific sub-range within address sub-range 2.
SIZE2	[23:21]	rw	**Address Size 2** SIZE2 determines the sub-range size within address sub-range 2.
SLICE3	[28:24]	rw	**Address Slice 3** SLICE3 selects a specific sub-range within address sub-range 3.
SIZE3	[31:29]	rw	**Address Size 3** SIZE3 determines the sub-range size within address sub-range 3.

21.4.4 Transmitter Control/Status Registers

Transmitter Control Register

The Transmitter Control Register TCR includes transmitter related control bits and bit fields that are used for parity/acknowledge, address optimization, TDATA idle polarity, retry, and transmitter enable/disable control.

TCR
Transmitter Control Register (10_H) Reset Value: 0000 0110$_H$



Field	Bits	Type	Description
MOD	0	rw	**Mode of Operation** This bit enables the MLI transmitter. 0_B The MLI transmitter is disabled. 1_B The MLI transmitter is enabled.
DNT	1	rw	**Data in Not Transmission** This bit determines the level of the transmitter data line TDATA when no transmission is in progress. 0_B TDATA is at low level if no transmission is running. 1_B TDATA is at high level if no transmission is running.
0	2	rw	**Reserved** Read as 0; should be written with 0.

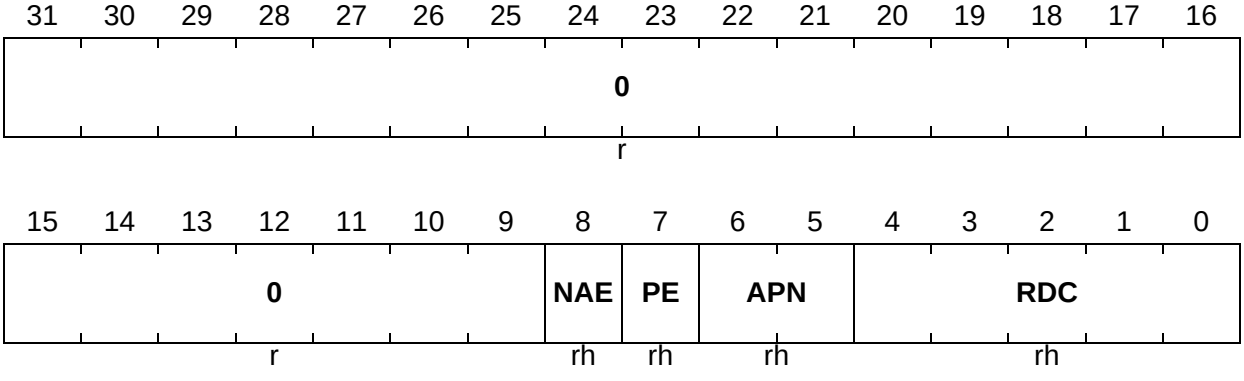
Field	Bits	Type	Description
MPE	[7:4]	rwh	**Maximum Parity Errors** This bit field determines the maximum number of transmitter parity error conditions that can be still detected until a transmitter parity error event is generated (see **Page 21-44**). With each condition detected, MPE is decremented down to 0. 0000_B A parity error event is generated if a transmitter parity error condition is detected. 0001_B A parity error event is generated if a transmitter parity error condition is detected. 0010_B A parity error event is generated if 2 transmitter parity error conditions are detected. 0011_B A parity error event is generated if 3 transmitter parity error conditions are detected. $\ldots_B$ … 1110_B A parity error event is generated if 14 transmitter parity error conditions are detected. 1111_B A parity error event is generated if 15 transmitter parity error conditions are detected.
MNAE	[9:8]	rwh	**Maximum Non Acknowledge Errors** This bit field determines the maximum number of consecutive Non-Acknowledge error conditions that can be still detected in the transmitter until a time-out event is generated. MNAE is decremented down to 0 at each Non-Acknowledge error condition. When MNAE = 0 or becoming 0, a time-out event is generated. MNAE is automatically set to 11_B after a successful frame transmission (see **Page 21-47**). 00_B A time-out event is generated if a non-ack condition is detected. 01_B A time-out event is generated if a non-ack condition is detected. 10_B A time-out event is generated if 2 consecutive non-ack conditions are detected. 11_B A time-out event is generated if 3 consecutive non-ack conditions are detected.

Field	Bits	Type	Description
MDP	[13:10]	rw	**Maximum Delay for Parity Error** This bit field determines a window for the transmitter in number of TCLK clock periods where a TREADY low-to-high signal transition signal is considered as “correctly received” condition (see **Page 21-22**). 0000_B Zero clock periods selected (not useful) 0001_B 1 clock period selected $\ldots_B$ … 1110_B 14 clock periods selected 1111_B 15 clock periods selected
NO	14	rw	**No Optimized Method** This bit field enables/disables the address prediction for read or Write Frames (see **Page 21-47**). 0_B Optimized method (address prediction) enabled. 1_B Optimized method (address prediction) disabled.
TP	15	rw	**Type of Parity** This bit will determines the type of parity used in frame transmissions. For correct data transfers, TP = 0 has to be programmed. The value TP = 1 can be selected to force parity errors to analyze the propagation delay (see **Page 21-26**). 0_B Even parity is selected. 1_B Odd parity selected.
TDEL	[19:16]	rw	**Transmission Delay** This bit field defines a delay in cycles of f_{SYS} of the transmitter between the reception of the rising edge of RREADY and the next possible frame start (see **Page 21-50**). 0000_B No transmission delay selected 0001_B One f_{SYS} cycle delay selected $\ldots_B$ … 1110_B Fourteen f_{SYS} cycles delay selected 1111_B Fifteen f_{SYS} cycles delay selected
0	3, [31:20]	r	**Reserved** Read as 0; should be written with 0.

Transmitter Status Register

The Transmitter Status Register TSTATR contains transmitter specific status information.

TSTATR
Transmitter Status Register (14_H) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
RDC	[4:0]	rh	**Ready Delay Counter** This bit field counts TCLK periods after the end of a frame transmission. When the TVALID signal goes to low level, RDC is cleared to zero and starts counting up the TCLK clock periods until a TREADY high level is detected (see **Page 21-22**).
APN	[6:5]	rh	**Answer Pipe Number** This bit field is written by the MLI receiver with the Pipe Number of a received Read Frame. APN is used by an Answer Frame that is transmitted as response to the Read Frame. 00_B Pipe 0 is used in Answer Frame. 01_B Pipe 1 is used in Answer Frame. 10_B Pipe 2 is used in Answer Frame. 11_B Pipe 3 is used in Answer Frame.
PE	7	rh	**Parity Error Flag** This bit is set if a transmitter parity error condition is detected by the transmitter after a frame transmission. PE is cleared by hardware when a frame has been transmitted without a parity error (see **Page 21-44**). Bit PE can be cleared by software via bit SCR.CTPE.

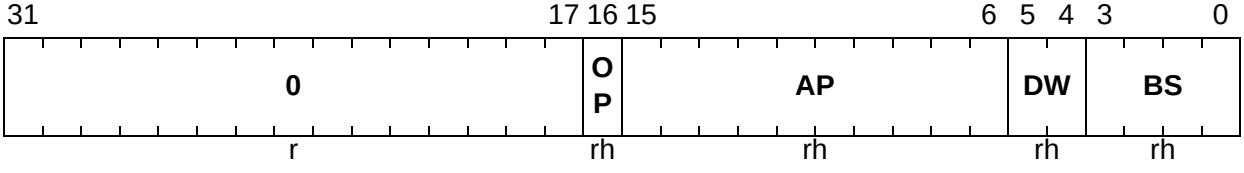
Field	Bits	Type	Description
NAE	8	rh	**Non Acknowledge Error Flag** This bit is set when a Non-Acknowledge error condition is detected by the MLI transmitter after a frame transmission (see **Page 21-47**). NAE is cleared by hardware if a transmitted frame has been acknowledged correctly. Bit NAE can be cleared by software via bit SCR.CNAE.
0	[31:9]	r	**Reserved** Read as 0; should be written with 0.

Transmitter Pipe x Status Registers

The Transmitter Pipe x Status Registers TPxSTATR contain pipe-specific status information related to address optimization and prediction, data width for transmit data, and Remote Window size.

TPxSTATR (x = 0-3)
Transmitter Pipe x Status Register

(18_H+4_H*x) **Reset Value: 0000 0000_H**



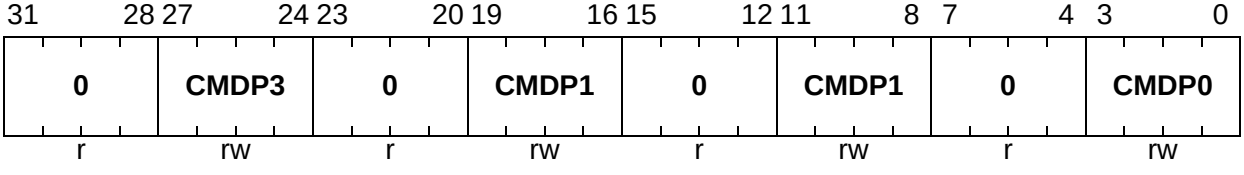
Field	Bits	Type	Description
BS	[3:0]	rh	**Buffer Size** This bit field indicates the coded buffer size of the pipe x Remote Window in the receiving controller. BS further determines how many address offset bits are transmitted in a Write Offset and Data Frame or in a Discrete Read Frame. When register TPxBAR is written for generation of a Copy Base Address Frame, BS is updated by the Copy Base Address Frame (see **Page 21-28**). 0000_B 1-bit offset address of Remote Window 0001_B 2-bit offset address of Remote Window 0010_B 3-bit offset address of Remote Window $\ldots_B$ … 1110_B 15-bit offset address of Remote Window 1111_B 16-bit offset address of Remote Window
DW	[5:4]	rh	**Data Width** This bit field indicates the data width that has been detected for a read or write access of a bus master to a Transfer Window of pipe x (see **Page 21-30** and **Page 21-34**). 00_B 8-bit data width detected 01_B 16-bit data width detected 10_B 32-bit data width detected 11_B Reserved

Field	Bits	Type	Description
AP	[15:6]	rh	**Address Prediction Factor** This bit field indicates the delta value (positive or negative number) of offset address used by the MLI transmitter for the next address prediction. AP is a signed 9-bit number (10th bit is the sign bit) that is written with each transmitter address prediction calculation (see **Page 21-26** and **Page 21-47**).
OP	16	rh	**Use Optimized Frame** When address optimization is enabled with TCR.NO = 0, this bit indicates if address prediction is possible in the transmitter. OP is written with each transmitter address prediction calculation (see **Page 21-26** and **Page 21-47**). 0_B No address prediction is possible. A Write Offset and Data Frame or a Discrete Read Frame are used for transmission. 1_B Address prediction is possible. An Optimized Write Frame or an Optimized Read Frame are used for transmission.
0	[31:17]	r	**Reserved** Read as 0; should be written with 0.

Transmitter Command Register

The Transmitter Command Register TCMDR contains the command codes that are used during Command Frame transmission (see **Page 21-41**). Each time one of the CMDPx bit fields is written, a Command Frame transmission is triggered. Independent of the transferred command code value, a Command Frame transmitted event can be generated in the transmitter for each pipe and a Command Frame received event for each pipe in the receiver, respectively.

TCMDR
Transmitter Command Register (28$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
CMDP0	[3:0]	rw	**Command Code for Pipe 0** This bit field contains the command code related to pipe 0. The pipe 0 command codes allow an activation (pulse) of one of the service request outputs SR[3:0] in the receiving controller. 0001_B Activate SR0 0010_B Activate SR1 0011_B Activate SR2 0100_B Activate SR3 Other bit combinations are reserved for future use; no further action in the receiver.
CMDP1	[11:8]	rw	**Command Code for Pipe 1** This bit field contains the command code related to pipe 1. The pipe 1 command codes allow to adjust the receiver delay for the parity error condition (see RCR.DPE) in the MLI receiver of the receiving controller. 0000_B Set RCR.DPE = 0000_B 0001_B Set RCR.DPE = 0001_B $\ldots_B$ … 1110_B Set RCR.DPE = 1110_B 1111_B Set RCR.DPE = 1111_B

Field	Bits	Type	Description
CMDP2	[19:16]	rw	**Command Code for Pipe 2** This bit field contains the command code related to pipe 2. The pipe 2 command codes allow to control the MLI receiver in the receiving controller. 0001_B Enable Automatic Data Mode (RCR.MOD = 1) 0010_B Disable Automatic Data Mode (RCR.MOD = 0) 0100_B Clear bit TRSTATR.RP0 0101_B Clear bit TRSTATR.RP1 0110_B Clear bit TRSTATR.RP2 0111_B Clear bit TRSTATR.RP3 1111_B Activate a pulse at $\overline{\text{BRKOUT}}$ Other bit combinations are reserved for future use; no further action in the receiver.
CMDP3	[27:24]	rw	**Command Code for Pipe 3** This bit field contains the command code related to pipe 3. The command codes for pipe 3 are free programmable by software.
0	[7:4], [15:12], [23:20], [31:28]	r	**Reserved** Read as 0; should be written with 0.

Transmitter-Receiver Status Register

The Transmitter Receiver Status Register TRSTATR contains read-only flags that indicate the status of MLI operations.

TRSTATR
Transmitter Receiver Status Register

($2C_H$) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0						PN		RP3	RP2	RP1	RP0	DV3	DV2	DV1	DV0
r						rh		rh	rh	rh	rh	rh	rh	rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						BAV	AV	CV3	CV2	CV1	CV0	CIV3	CIV2	CIV1	CIV0
r						rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
CIV0, **CIV1,** **CIV2,** **CIV3**	0, 1, 2, 3	rh	**Command Interrupt Valid** Bit is set to 1 by the MLI transmitter whenever it detects a rising edge at the corresponding TRx input line (for Triggered Command Frames in pipe 0). It is cleared by hardware when the Command Frame has been correctly transmitted. CIVx can be cleared by software via bit SCR.CCIVx.
CV0, **CV1,** **CV2,** **CV3**	4, 5, 6, 7	rh	**Command Valid** Bit is set by hardware when a TCMDR.CMDPx bit field is written. It is cleared by hardware when the Command Frame has been correctly transmitted. CVx can be set or cleared by software via bits SCR.SCVx or SCR.CCVx.
AV	8	rh	**Answer Valid** Bit is set by hardware when the TDRAR register in the the MLI transmitter (in the Remote Controller) is written. AV is cleared by hardware when the Answer Frame has been correctly sent. AV can be cleared by software via bit SCR.CAV.

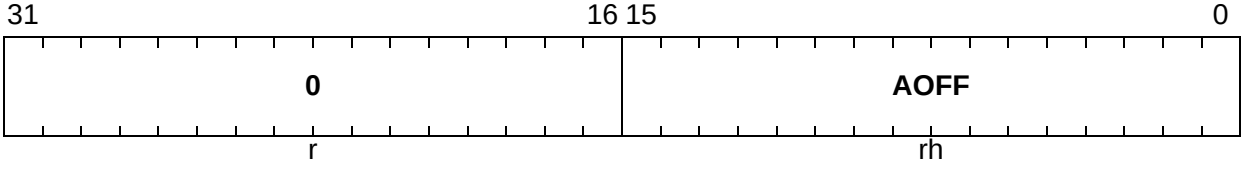
Field	Bits	Type	Description
BAV	9	rh	**Base Address Valid** Bit is set by hardware when the TCBAR register in the MLI transmitter is written. BAV is cleared by hardware when the Copy Base Address Frame has been correctly sent. BAV can be cleared by software via bit SCR.CBAV.
DV0, **DV1,** **DV2,** **DV3**	16, 17, 18, 19	rh	**Data Valid** Bit is set by hardware when the TPxDATAR and/or the TPxAOFR registers of the MLI transmitter are updated after a read or write access to a Transfer Window of pipe x. DVx is cleared again by hardware when the read or Write Frame has been correctly sent. DVx can be cleared by software via bit SCR.CDVx.
RP0, **RP1,** **RP2,** **RP3**	20, 21, 22, 23	rh	**Read Pending** Bit is set by hardware when the TPxAOFR register of the MLI transmitter is updated after a read access to a Transfer Window of pipe x. RPx is cleared by hardware when the MLI receiver in the Local Controller receives an Answer Frame for pipe x from the Remote Controller. RPx can be cleared by software via bit SCR.CDVx.
PN	[25:24]	rh	**Pipe Number** This bit field indicates the Pipe Number x of the base address that has been written into register TPxBAR. 00_B TP0BAR has been last written. 01_B TP1BAR has been last written. 10_B TP2BAR has been last written. 11_B TP3BAR has been last written.
0	[15:10], [31:26]	r	**Reserved** Read as 0; should be written with 0.

21.4.5 Transmitter Pipe x Address Offset Register

Transmitter Pipe x Address Offset Register

The Transmitter Pipe x Address Offset Register TPxAOFR is a read-only register that stores the offset address that has been used by the last read or write access to a Transfer Window of pipe x.

TPxAOFR (x = 0-3)
Transmitter Pipe x Address Offset Register
(30_H+4_H*x) **Reset Value: 0000 0000_H**

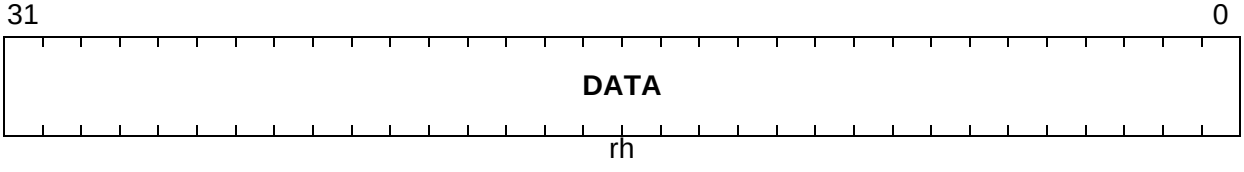


Field	Bits	Type	Description
AOFF	[15:0]	rh	**Address Offset** Whenever a location within a Transfer Window is accessed (read or written) AOFF is loaded with the lowest 16 address bits of the access. Also in the case of a small Transfer Window access, all AOFF bits are loaded, but AOFF[15:13] are not taken into account for further actions assuming the buffer size is configured correctly (see **Page 21-105**).
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

Transmitter Pipe x Data Register

The Transmitter Pipe x Data Register TPxDATAR is a read-only register that stores the data that has been written during the last write access to a Transfer Window of pipe x.

TPxDATAR (x = 0-3)
Transmitter Pipe x Data Register (40_H+4_H*x) Reset Value: 0000 0000_H

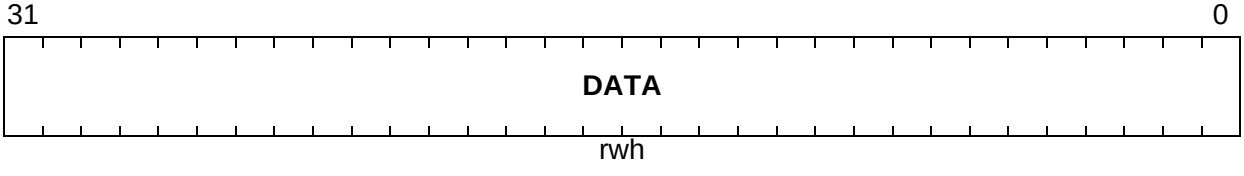


Field	Bits	Type	Description
DATA	[31:0]	rh	**Data** Whenever a location within a Transfer Window is written, the data is loaded in this bit field.

Transmitter Data Read Answer Register

The Transmitter Data Read Answer Register TDRAR contains the read data for the transmission of an Answer Frame.

TDRAR
Transmitter Data Read Answer Register
(50_H) Reset Value: 0000 0000_H

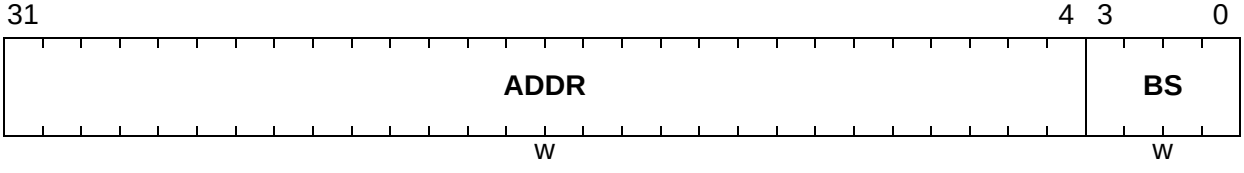


Field	Bits	Type	Description
DATA	[31:0]	rwh	**Data** This bit field is loaded with data that is read from the address requested by a Read Frame. An update of this bit field triggers the start of an Answer Frame with DATA used as content of the Answer Frame. This bit field can be updated either automatically by the move engine (if Automatic Data Mode is enabled) or by the CPU (if Automatic Data Mode is disabled).

Transmitter Pipe x Base Address Register

The write-only Transmitter Pipe x Base Address Register TPxBAR represents the 28-bit pipe x Remote Window base address and the Remote Window size that is transmitted to the receiving controller via a Copy Base Address Frame.

TPxBAR (x = 0-3)
Transmitter Pipe x Base Address Register
(54_H+4_H*x) **Reset Value: 0000 0000_H**

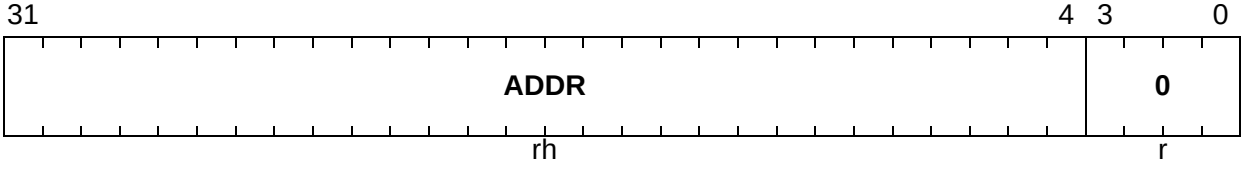


Field	Bits	Type	Description
BS	[3:0]	w	**Buffer Size** This bit field determines the coded buffer size of the pipe x Remote Window in the receiving controller. When writing TPxBAR, BS is copied into bit field TPxSTATR.BS. 0000_B 1-bit offset address of Remote Window 0001_B 2-bit offset address of Remote Window 0010_B 3-bit offset address of Remote Window $\ldots_B$ … 1101_B 14-bit offset address of Remote Window 1110_B 15-bit offset address of Remote Window 1111_B 16-bit offset address of Remote Window Do not use the coding values 1101_B, 1110_B, and 1111_B as buffer size for Small Transfer Windows.
ADDR	[31:4]	w	**Address** This bit field determines the most significant 28 bits of the pipe x Remote Window base address. When writing TPxBAR, ADDR is copied into bit field TCBAR.ADDR[31:4].

Transmitter Copy Base Address Register

The Transmitter Copy Base Address Register TCBAR contains the 28-bit pipe x Remote Window base address of the latest write access to TPxBAR.ADDR.

TCBAR
Transmitter Copy Base Address Register
(64_H) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
ADDR	[31:4]	rh	**Address** This bit field contains the 28 address bits written to TPxBAR.ADDR. This value will be transferred to the receiving controller to define the base address of the Remote Window for pipe x.
0	[3:0]	r	**Reserved** Read as 0; should be written with 0.

21.4.6 Transmitter Interrupt Registers

Transmitter Interrupt Enable Register

The Transmitter Interrupt Enable Register TIER contains the interrupt enable bits and the clear bits for all transmitter events. The bits marked w always read as 0.

TIER
Transmitter Interrupt Enable Register (98_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0						TEIR	PEIR	CFSIR3	CFSIR2	CFSIR1	CFSIR0	NFSIR3	NFSIR2	NFSIR1	NFSIR0
r						w	w	w	w	w	w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						TEIE	PEIE	CFSIE3	CFSIE2	CFSIE1	CFSIE0	NFSIE3	NFSIE2	NFSIE1	NFSIE0
r						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
NFSIE0, **NFSIE1,** **NFSIE2,** **NFSIE3**	0, 1, 2, 3	rw	**Normal Frame Sent in Pipe x Interrupt Enable** 0_B Normal frame sent in pipe x event is disabled for activation of an SRx line. 1_B Normal frame sent in pipe x event is enabled for activation of an SRx line.
CFSIE0, **CFSIE1,** **CFSIE2,** **CFSIE3**	4, 5, 6, 7	rw	**Command Frame Sent in Pipe x Interrupt Enable** 0_B Command frame sent in pipe x event is disabled for activation of an SRx line. 1_B Command frame sent in pipe x event is enabled for activation of an SRx line.
PEIE	8	rw	**Parity Error Interrupt Enable** 0_B Parity error event is disabled for activation of an SRx line. 1_B Parity error event is enabled for activation of an SRx line.

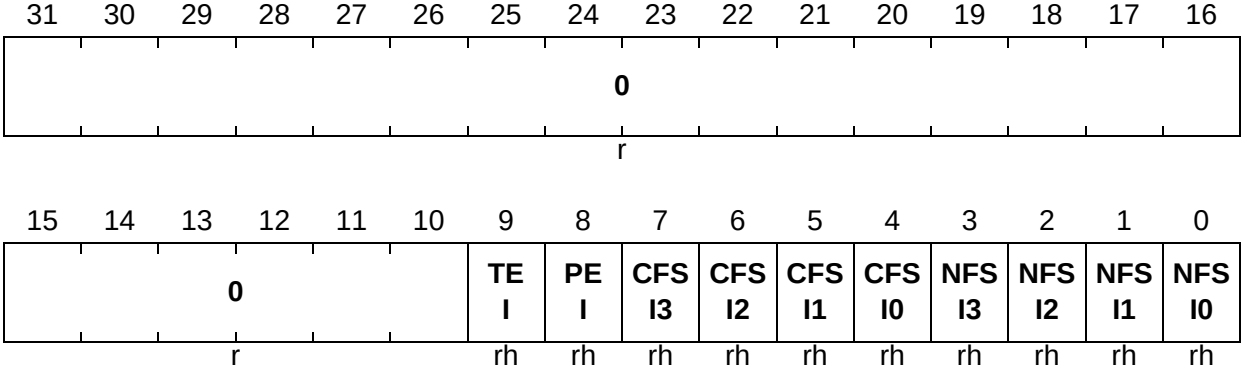
Field	Bits	Type	Description
TEIE	9	rw	**Time-Out Error Interrupt Enable** 0_B Time-out error event is disabled for activation of an SRx line. 1_B Time-out error event is enabled for activation of an SRx line.
NFSIR0, NFSIR1, NFSIR2, NFSIR3	16, 17, 18, 19	w	**Normal Frame Sent in Pipe x Flag Clear** 0_B No action. 1_B Clear TISR.NFSIx.
CFSIR0, CFSIR1, CFSIR2, CFSIR3	20, 21, 22, 23	w	**Command Frame Sent in Pipe x Flag Clear** 0_B No action. 1_B Clear TISR.CFSIx.
PEIR	24	w	**Parity Error Flag Clear** 0_B No action. 1_B Clear TISR.PEIx.
TEIR	25	w	**Time Out Error Flag Clear** 0_B No action. 1_B Clear TISR.TEIx.
0	[15:10], [31:26]	r	**Reserved** Read as 0; should be written with 0.

Transmitter Interrupt Register

The Transmitter Interrupt Status Register TISR contains all MLI event (or interrupt) flags of the MLI transmitter. These flags can be cleared by software when writing the appropriate bits in the TIER register; they are not cleared by hardware.

TISR

Transmitter Interrupt Status Register ($9C_H$) **Reset Value: $0000\ 0000_H$**



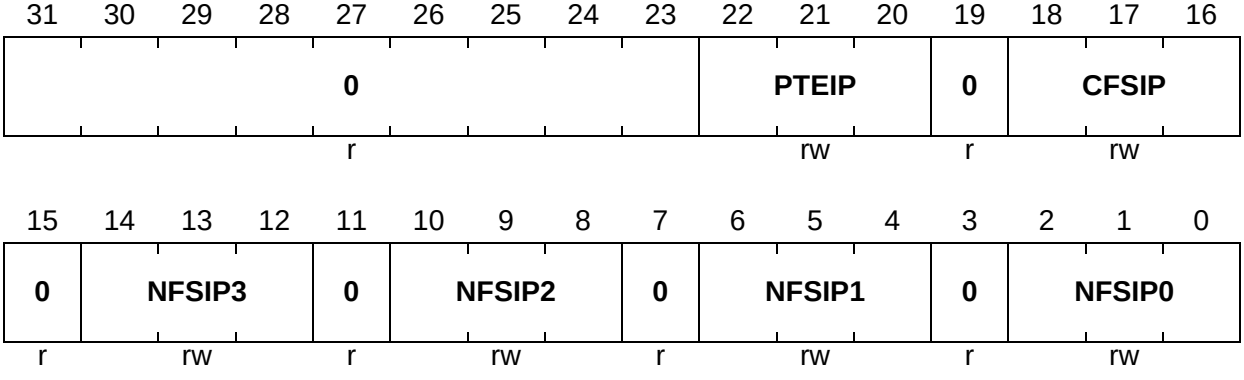
Field	Bits	Type	Description
NFSI0, **NFSI1,** **NFSI2,** **NFSI3**	0, 1, 2, 3	rh	**Normal Frame Sent in Pipe x Flag** The service request output that can be activated is defined by TINPR.NFSIPx. 0_B A Normal Frame has not yet been sent. 1_B A Write or Read Frame has been correctly sent and acknowledged for pipe x.
CFSI0, **CFSI1,** **CFSI2,** **CFSI3**	4, 5, 6, 7	rh	**Command Frame Sent in Pipe x Flag** The service request output that can be activated is defined by TINPR.CFSIP. 0_B A Command Frame has not yet been sent. 1_B A Command Frame has been correctly sent and acknowledged for pipe x.
PEI	8	rh	**Parity Error Flag** The service request output that can be activated is defined by TINPR.PTEIPx. 0_B A parity error event has not yet been detected. 1_B A parity error event has been detected.

Field	Bits	Type	Description
TEI	9	rh	**Time-Out Error Flag** The service request output that can be activated is defined by TINPR.PTEIPx. 0_B A time-out error event has not yet been detected. 1_B A time-out error event has been detected.
0	[31:10]	r	**Reserved** Read as 0; should be written with 0.

Transmitter Interrupt Node Pointer Register

The Transmitter Interrupt Node Pointer Register TINPR contains the node pointers for the MLI transmitter events.

TINPR
Transmitter Interrupt Node Pointer Register
($A0_H$) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
NFSIP0	[2:0]	rw	**Normal Frame Sent in Pipe 0 Interrupt Pointer** This bit field determines which service request output SRx becomes active when a Normal Frame sent in pipe 0 event occurs (if enabled). 000_B The service request output SR0 is selected. 001_B The service request output SR1 is selected. $\ldots_B$ … 110_B The service request output SR6 is selected. 111_B The service request output SR7 is selected.
NFSIP1	[6:4]	rw	**Normal Frame Sent in Pipe 1 Interrupt Pointer** This bit field determines which service request output SRx becomes active when a Normal Frame sent in pipe 1 event occurs (if enabled). Coding see NFSIP0.
NFSIP2	[10:8]	rw	**Normal Frame Sent in Pipe 2 Interrupt Pointer** This bit field determines which service request output SRx becomes active when a Normal Frame sent in pipe 2 event occurs (if enabled). Coding see NFSIP0.
NFSIP3	[14:12]	rw	**Normal Frame Sent in Pipe 3 Interrupt Pointer** This bit field determines which service request output SRx becomes active when a Normal Frame sent in pipe 3 event occurs (if enabled). Coding see NFSIP0.

Field	Bits	Type	Description
CFSIP	[18:16]	rw	**Command Frame Sent Interrupt Pointer** This bit field determines which service request output SRx becomes active when a Command Frame sent event occurs (if enabled). Coding see NFSIP0.
PTEIP	[22:20]	rw	**Parity or Time Out Interrupt Pointer** This bit field determines which service request output SRx becomes active when a parity/time-out event occurs (if enabled). Coding see NFSIP0.
0	3, 7, 11, 15, 19, [31:23]	r	**Reserved** Read as 0; should be written with 0.

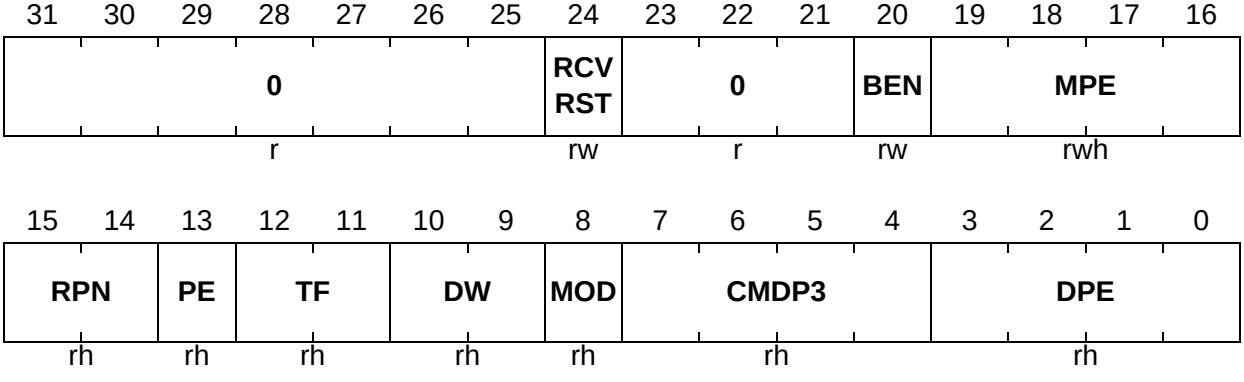
21.4.7 Receiver Control/Status Registers

Receiver Control Register

The Receiver Control Register RCR contains control and status bits/bit fields that are related to the MLI receiver operation.

Bit RCVRST is automatically overwritten after a reset (see **Page 21-68**) with a value given in the implementation chapter (see **Page 21-130**).

RCR
Receiver Control Register **(68_H)** **Reset Value: $0100\ 0000_H$**



Field	Bits	Type	Description
DPE	[3:0]	rh	**Delay for Parity Error** DPE determines the number of RCLK clock periods that the MLI receiver waits before the RREADY signal is raised again when it has detected a parity error (see **Page 21-22**). When a pipe 1 Command Frame is received by the MLI receiver, the command code is stored in this bit field (see **Page 21-41**). 0000_B Zero RCLK clock period delay is selected. 0001_B One RCLK clock period delay is selected. 0010_B Two RCLK clock periods delay is selected. $\ldots_B$ … 1110_B Fourteen RCLK clock periods delay is selected. 1111_B Fifteen RCLK clock periods delay is selected.
CMDP3	[7:4]	rh	**Command From Pipe 3** When a pipe 3 Command Frame is received by the MLI receiver, the command code is stored in this bit field. Pipe 3 commands are free for software use.

Field	Bits	Type	Description
MOD	8	rh	**Mode of Operation** This bit determines the data transfer operation mode of the MLI receiver. Bit MOD can be set by hardware with the reception of a pipe 2 Command Frame (see Page 21-100). It can be set or cleared by software via bits SCR.SMOD or SCR.CMOD. 0_B Automatic Data Mode is disabled. Data read/write operations from/to a Remote Window must be executed by a bus master (e.g. the CPU). 1_B Automatic Data Mode is enabled. Data read/write operations from/to a Remote Window are executed by the MLI's move engine.
DW	[10:9]	rh	**Data Width** This bit field is updated by the MLI receiver whenever new data is received in the RDATAR register. It indicates the relevant data width. 00_B 8-bit relevant data width in RDATAR 01_B 16-bit relevant data width in RDATAR 10_B 32-bit relevant data width in RDATAR 11_B Reserved
TF	[12:11]	rh	**Type of Frame** This bit field determines the frame type that has most recently been received by the MLI receiver. It is updated whenever the MLI receiver updates RDATAR, RADDR, or RPxBAR. The most recently received frame was a: 00_B Copy Base Address Frame 01_B Discrete Read Frame or Optimized Read Frame 10_B Write Offset and Data Frame or Optimized Write Frame 11_B Answer frame Note that the coding of TF is different from the frame coding as defined in **Table 21-1** on **Page 21-11**.
PE	13	rh	**Parity Error** PE is set when a parity error is detected in a received frame (see Page 21-44). PE is cleared by hardware when a frame has been received without parity error. PE can be cleared by software via bit SCR.CRPE.

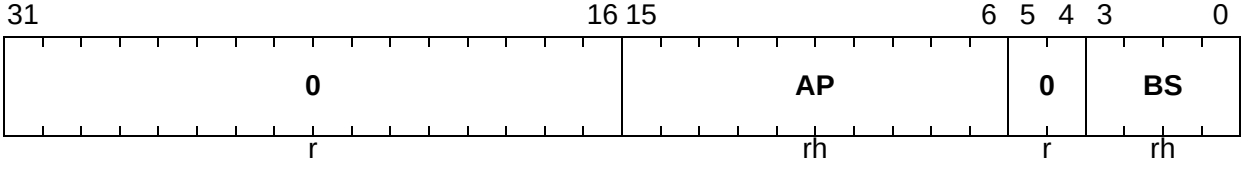
Field	Bits	Type	Description
RPN	[15:14]	rh	**Received Pipe Number** This bit field contains the Pipe Number that was indicated by the Pipe Number bit field of the latest received frame. It is updated by any received frame.
MPE	[19:16]	rwh	**Maximum Parity Errors** This bit field indicates the number of receive parity error conditions after which a receiver parity error event will be generated. It is set to a desired value by software and it is decremented down to 0 automatically by the MLI each time it detects a receiver parity error condition. If a receiver parity error condition is detected and MPE becomes 0 or is already 0, a receiver parity error event is generated (see **Page 21-44**). 0000_B A receiver parity event is generated if a receiver error condition is detected. 0001_B A receiver parity event is generated if a receiver error condition is detected. 0010_B A receiver parity event is generated if 2 receiver error conditions are detected. $\ldots_B$ … 1110_B A receiver parity event is generated if 14 receiver error conditions are detected. 1111_B A receiver parity event is generated if 15 receiver error conditions are detected.
BEN	20	rw	**Break Out Enable** When setting BEN = 1, the MLI receiver generates a pulse on its break output signal $\overline{\text{BRKOUT}}$ when a pipe 2 Command Frame with command code CMD = 1111_B is received. 0_B Break output signal generation is disabled. 1_B Break output signal is enabled.
RCVRST	24	rw	**Receiver Reset** This bit forces the receiver to be reset in order to be able to change OICR settings without affecting the receiver registers. 0_B The MLI receiver is in operating mode. 1_B The MLI receiver is held in reset state and OICR can be modified without unintentional actions in the receiver.

Field	Bits	Type	Description
0	[23:21], [31:25]	r	**Reserved** Read as 0; should be written with 0.

Receiver Pipe x Status Register

The Receiver Pipe x Status Register RPxSTATR indicates the coded buffer size which represents the Remote Window Size of 2 Bytes to 64 Kbytes and the address prediction factor that has been calculated for pipe x in the receiving controller.

RPxSTATR (x = 0-3)
Receiver Pipe x Status Register **($7C_H+4_H*x$)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
BS	[3:0]	rh	**Buffer Size** This bit field indicates the size of pipe x Remote Window in the receiving controller. It is updated by hardware when a Copy Base Address Frame has been received (see **Page 21-28**). 0000_B 1-bit offset address of Remote Window 0001_B 2-bit offset address of Remote Window 0010_B 3-bit offset address of Remote Window $\ldots_B$ … 1110_B 15-bit offset address of Remote Window 1111_B 16-bit offset address of Remote Window
AP	[15:6]	rh	**Address Prediction Factor** AP contains the address prediction factor that has been calculated for pipe x in the receiving controller. It is a signed 9-bit number with the sign in its most significant bit (see **Page 21-47**).
0	[5:4], [31:16]	r	**Reserved** Read as 0; should be written with 0.

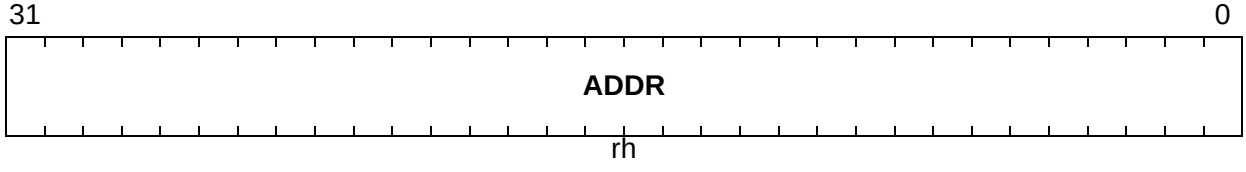
21.4.8 Receiver Address/Data Registers

Receiver Pipe x Base Address Register

The Receiver Pipe x Base Address Register RPxBAR is a read-only register that contains the complete target address in the Remote Window of pipe x.

RPxBAR (x = 0-3)
Receiver Pipe x Base Address Register

($6C_H$+4_H*x) **Reset Value: 0000 0000_H**

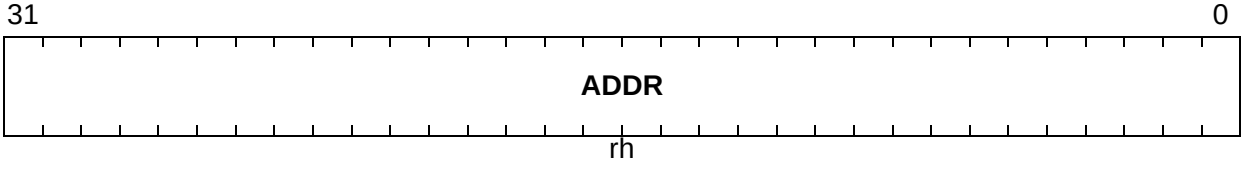


Field	Bits	Type	Description
ADDR	[31:0]	rh	**Address** ADDR indicates the complete target address for the pipe x Remote Window. If a pipe x Copy Base Address Frame is received, ADDR[31:4] becomes loaded with the transmitted 28-bit address and bits [3:0] are cleared. If a write or Read Frame with m bits of address offset is received, bits ADDR[31:m] are held constant and bits ADDR[m-1:0] are replaced by the received offset. If an optimized read or data frame is received, the address prediction mechanism adds the predicted address offset RPxSTATR.AP to ADDR and stores the result in ADDR. If an Answer Frame is received, ADDR is not changed.

Receiver Address Register

The Receiver Address Register RADRR is a read-only register storing the complete address of the most recently (or currently) targeted Remote Window.

RADRR
Receiver Address Register **($8C_H$)** **Reset Value: 0000 0000_H**

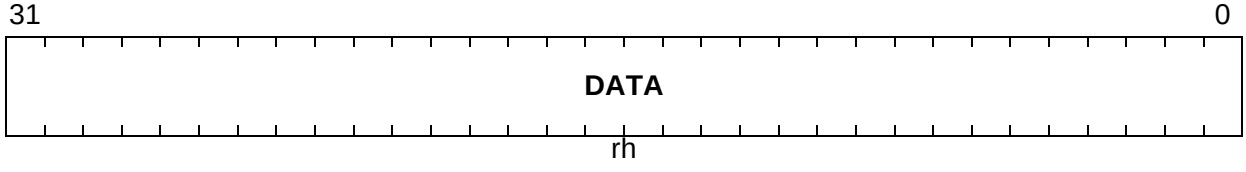


Field	Bits	Type	Description
ADDR	[31:0]	rh	**Address** ADDR indicates the complete target address for the most recently (or currently) targeted Remote Window (pipe x). If a Copy Base Address Frame is received, ADDR is unchanged. If a write or Read Frame with m bits of address offset is received, bits ADDR[31:m] replaced by the bits RPxBAR.ADDR[31:m] and bits ADDR[m-1:0] are replaced by the received offset. If an optimized read or data frame is received, the address prediction mechanism adds the predicted address offset RPxSTATR.AP to RPxBAR.ADDR and stores the result in ADDR. If an Answer Frame is received, ADDR becomes invalid.

Receiver Data Register

The Receiver Data Register RDATAR is a read-only register that stores data received by a Write Frame or an Answer Frame.

RDATAR
Receiver Data Register **(90_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
DATA	[31:0]	rh	**Data** In the receiving controller, DATA contains the data received by a Write Frame or an Answer Frame. Bit field RCR.DW determines the width of the relevant data that is stored in RDATAR. RCR.DW = 00_B: RDATAR[7:0] are relevant (8-bit) RCR.DW = 01_B: RDATAR[15:0] are relevant (16-bit) RCR.DW = 10_B: RDATAR[31:0] are relevant (32-bit)

21.4.9 Receiver Interrupt Registers

Receiver Interrupt Enable Register

The Receiver Interrupt Enable Register RIER contains the interrupt enable bits and the clear bits for all receiver events. The bits marked w are always read as 0.

RIER
Receiver Interrupt Enable Register (A4$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0						DRAIR	MPEIR	PEIR	ICER	CFRIR3	CFRIR2	CFRIR1	CFRIR0	MEIR	NFRIR
r						w	w	w	w	w	w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						DRAIE	MPEIE	PEIE	ICE	CFRIE3	CFRIE2	CFRIE1	CFRIE0	NFRIE	
r						rw	rw	rw	rw	rw	rw	rw	rw	rw	

Field	Bits	Type	Description
NFRIE	[1:0]	rw	**Normal Frame Received Interrupt Enable** This bit field defines if an SRx output is activated if a Normal Frame is correctly received. 00_B The SRx activation is disabled. 01_B The selected SRx line is activated each time a Normal Frame is correctly received. 10_B The selected SRx line is activated each time a Normal Frame is correctly received that is not handled automatically by the MLI move engine (e.g. an Answer Frame). 11_B Reserved
CFRIE0, CFRIE1, CFRIE2, CFRIE3	2, 3, 4, 5	rw	**Command Received in Pipe x Interrupt Enable** This bit determines if an SRx output is activated if a Command Frame for pipe x has been received correctly. 0_B Command received in pipe x event is disabled for activation of an SRx line. 1_B Command received in pipe x event is enabled for activation of an SRx line.

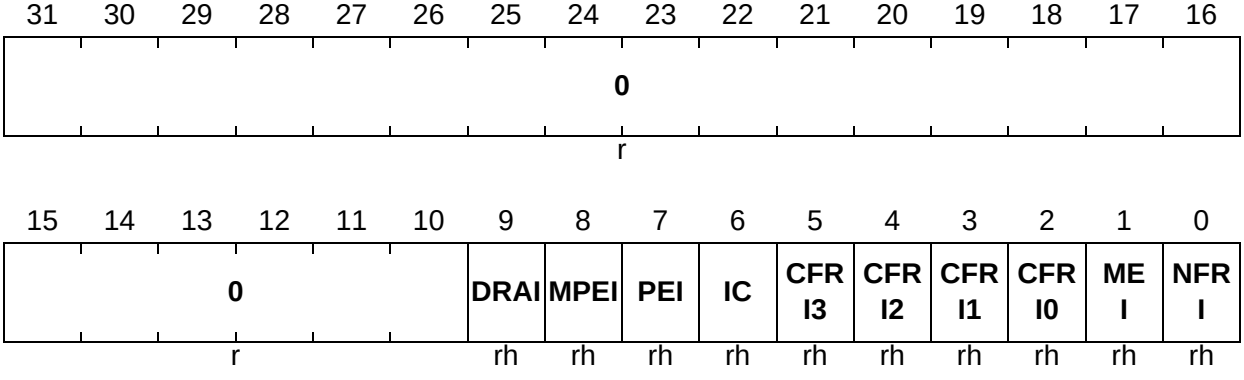
Field	Bits	Type	Description
ICE	6	rw	**Interrupt Command Enable** This bit determines if an SRx output line is activated if a Command Frame is received in pipe 0. 0_B Command frame received in pipe 0 event is disabled for activation of an SRx line. 1_B Command frame received in pipe 0 event is enabled for activation of an SRx line.
PEIE	7	rw	**Parity Error Interrupt Enable** This bit determines if an SRx output line is activated if receiver a parity error event is detected. 0_B Parity error event is disabled for activation of an SRx line. 1_B Parity error event is enabled for activation of an SRx line.
MPEIE	8	rw	**Memory Access Protection Interrupt Enable** This bit determines if an SRx output line is activated if a memory access protection error is detected. 0_B Memory access protection error event is disabled for activation of an SRx line. 1_B Memory access protection error event is enabled for activation of an SRx line.
DRAIE	9	rw	**Discarded Read Answer Interrupt Enable** This bit determines if an SRx output line is activated if a discarded read Answer Frame condition is detected. 0_B Discarded read answer event is disabled for activation of an SRx line. 1_B Discarded read answer event is enabled for activation of an SRx line.
NFRIR	16	w	**Normal Frame Received Interrupt Flag Clear** 0_B No action. 1_B Clear RISR.NFRI.
MEIR	17	w	**MLI Move Engine Interrupt Flag Clear** 0_B No action. 1_B Clear RISR.MEI.
CFRIR0, CFRIR1, CFRIR2, CFRIR3	18, 19, 20, 21	w	**Command Frame Received in Pipe x Interrupt Flag Clear** 0_B No action. 1_B Clear RISR.CFRIx.

Field	Bits	Type	Description
ICER	22	w	**Interrupt Command Flag Clear** 0_B No action. 1_B Clear RISR.ICE.
PEIR	23	w	**Parity Error Interrupt Flag Clear** 0_B No action. 1_B Clear RISR.PEI.
MPEIR	24	w	**Memory Protection Error Interrupt Flag Clear** 0_B No action. 1_B Clear RISR.MPEI.
DRAIR	25	w	**Discarded Read Answer Interrupt Flag Clear** 0_B No action. 1_B Clear RISR.DRAI.
0	[15:10], [31:26]	r	**Reserved** Read as 0; should be written with 0.

Receiver Interrupt Status Register

The Receiver Interrupt Status Register RISR contains all event (interrupt) flags of the MLI receiver. These flags can be cleared by software when writing the appropriate bits in the RIER register; they are not cleared by hardware.

RISR
Receiver Interrupt Status Register (A8$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
NFRI	0	rh	**Normal Frame Received Interrupt Flag** This flag is set when a write or a Read Frame has been received. The service request output that is activated is defined by RINPR.NFRIP.
MEI	1	rh	**MLI Move Engine Interrupt Flag** This flag is set when the move engine has finished an operation (read or write, depending on received frame). The service request output that is activated is defined by RINPR.MPPEIP.
CFRI0, CFRI1, CFRI2, CFRI3	2, 3, 4, 5	rh	**Command Frame Received in Pipe x Interrupt Flag** This flag is set when a Command Frame has been received in pipe x. The service request output that is activated is defined by RINPR.CFRIP.
IC	6	rh	**Interrupt Command Flag** This flag is set when a Command Frame has been received in pipe 0 leading to an activation of one of the service request outputs SR[3:0]. The service request output that is activated is defined by the received command CMD.

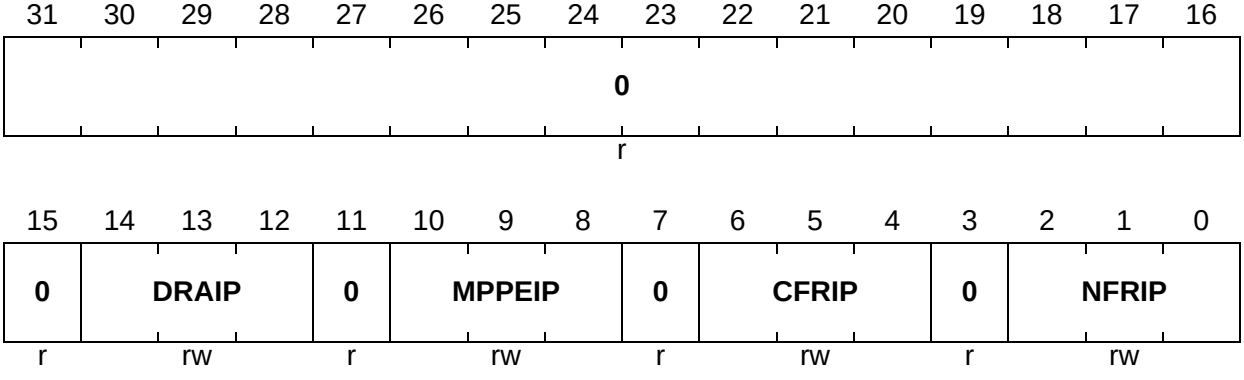
Field	Bits	Type	Description
PEI	7	rh	**Parity Error Interrupt Flag** This flag is set when a parity error event has occurred. The service request output that is activated is defined by RINPR.MPPEIP.
MPEI	8	rh	**Memory Protection Error Interrupt Flag** This flag is set when a memory protection event has occurred. The service request output that is activated is defined by RINPR.MPPEIP.
DRAI	9	rh	**Discarded Read Answer Interrupt Flag** This flag is set when the discarded read answer event has occurred. This condition occurs if an Answer Frame is received while none of the TRSTATR.RPx bits is set (the Answer Frame was not expected). The service request output that is activated is defined by RINPR.DRAIP.
0	[31:10]	r	**Reserved** Read as 0; should be written with 0.

Receiver Interrupt Node Pointer Register

The Receiver Interrupt Node Pointer Register RINPR contains the node pointers for the MLI receiver events.

RINPR
Receiver Interrupt Node Pointer Register

(AC_H) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
NFRIP	[2:0]	rw	**Normal Frame Received Interrupt Pointer** This bit field determines which service request output SRx becomes active when a Normal Frame received event occurs. 000_B The service request output SR0 is selected. 001_B The service request output SR1 is selected. $\ldots_B$ … 110_B The service request output SR6 is selected. 111_B The service request output SR7 is selected.
CFRIP	[6:4]	rw	**Command Frame Received Interrupt Pointer** This bit field determines which service request output SRx becomes active when a Command Frame received event occurs. Coding see NFRIP.
MPPEIP	[10:8]	rw	**Memory Protection or Parity Error Interrupt Pointer** This bit field determines which service request output SRx becomes active when a memory protection/parity error event occurs. Coding see NFRIP.
DRAIP	[14:12]	rw	**Discarded Read Answer Interrupt Pointer** This bit field determines which service request output SRx becomes active when a discarded read answer event occurs. Coding see NFRIP.

Field	Bits	Type	Description
0	3, 7, 11, [31:15]	r	**Reserved** Read as 0; should be written with 0.

21.5 Implementation of the MLI0/MLI1 in TC1797

This section describes the MLI0/MLI1 module related external functions such as port connections, interrupt and service request control, connections to other on-chip modules, clock control, and the address map.

21.5.1 Interfaces of the MLI Modules

Each MLI module is supplied with separate clock control, address decoding, and interrupt control logic. Four (for MLI0) and two (for MLI1) of the eight module service request outputs are connected to service request nodes. Four service request outputs of each MLI module are connected as DMA request to with the DMA controller.

The data, clock, and control lines of each MLI receiver and transmitter are connected to GPIO lines. Alternate functions of Port 1 and Port 5 lines are assigned to the MLI0 module I/O lines while alternate functions of Port 8 lines are assigned to the MLI0 module I/O lines. Additionally, within one MLI module transmitter and receiver signals can be dynamically connected among each other without using pins; this is useful for test purposes.

Figure 21-52 and **Figure 21-53** show how the MLI0 and MLI1 modules are interconnected to port lines and other on-chip functional blocks.

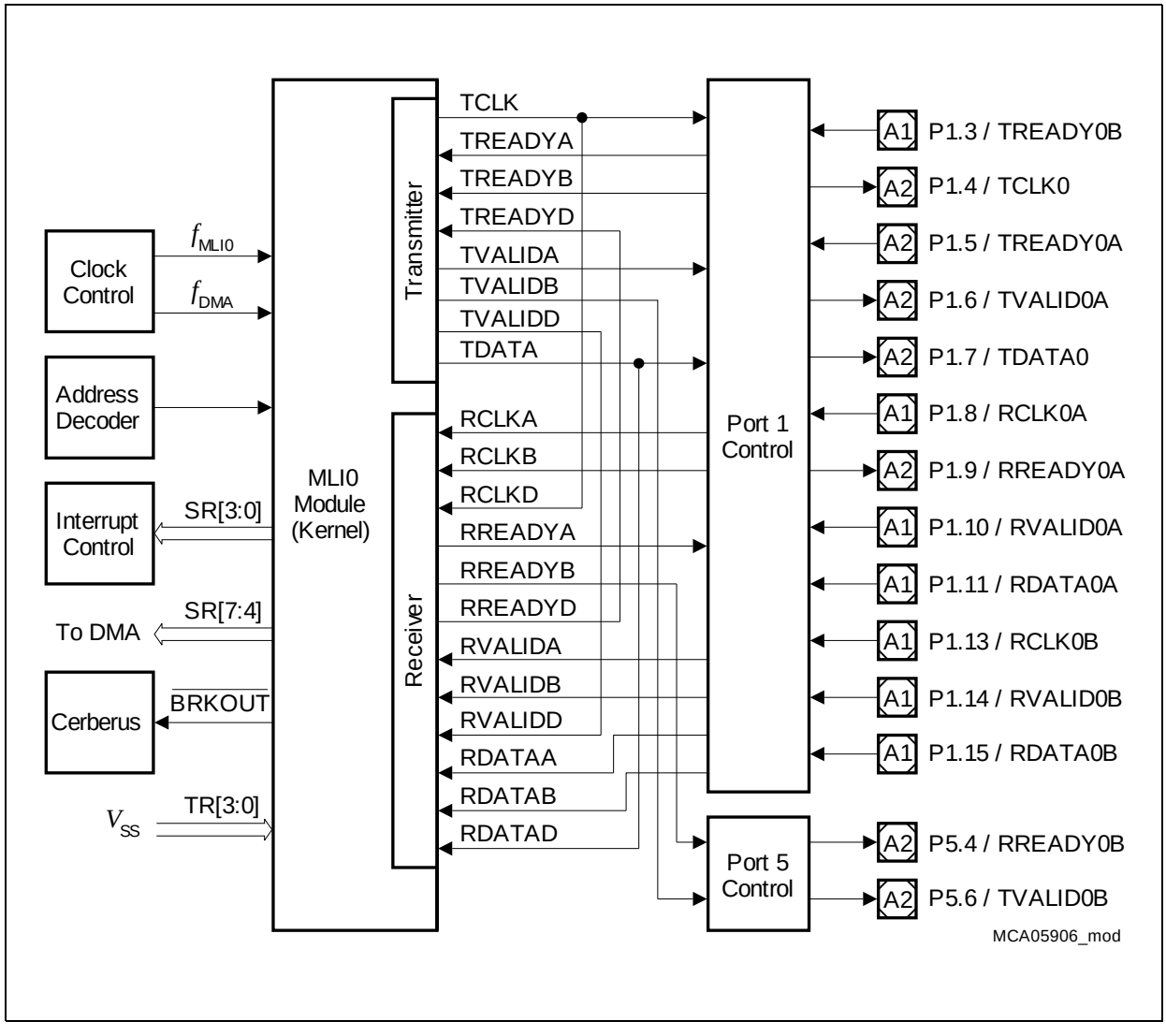


Figure 21-52 MLI0 Module Implementation and Interconnections

When programming the MLI0_OICR register, the following additional items must be considered:

- Unused transmitter/receiver output lines with index “C” (TVALIDC and RREADYC) are not connected.
- Unused transmitter/receiver input lines with index “C” (TREADYC, RCLKC, RVALIDC, and RDATAC) are connected to low level.

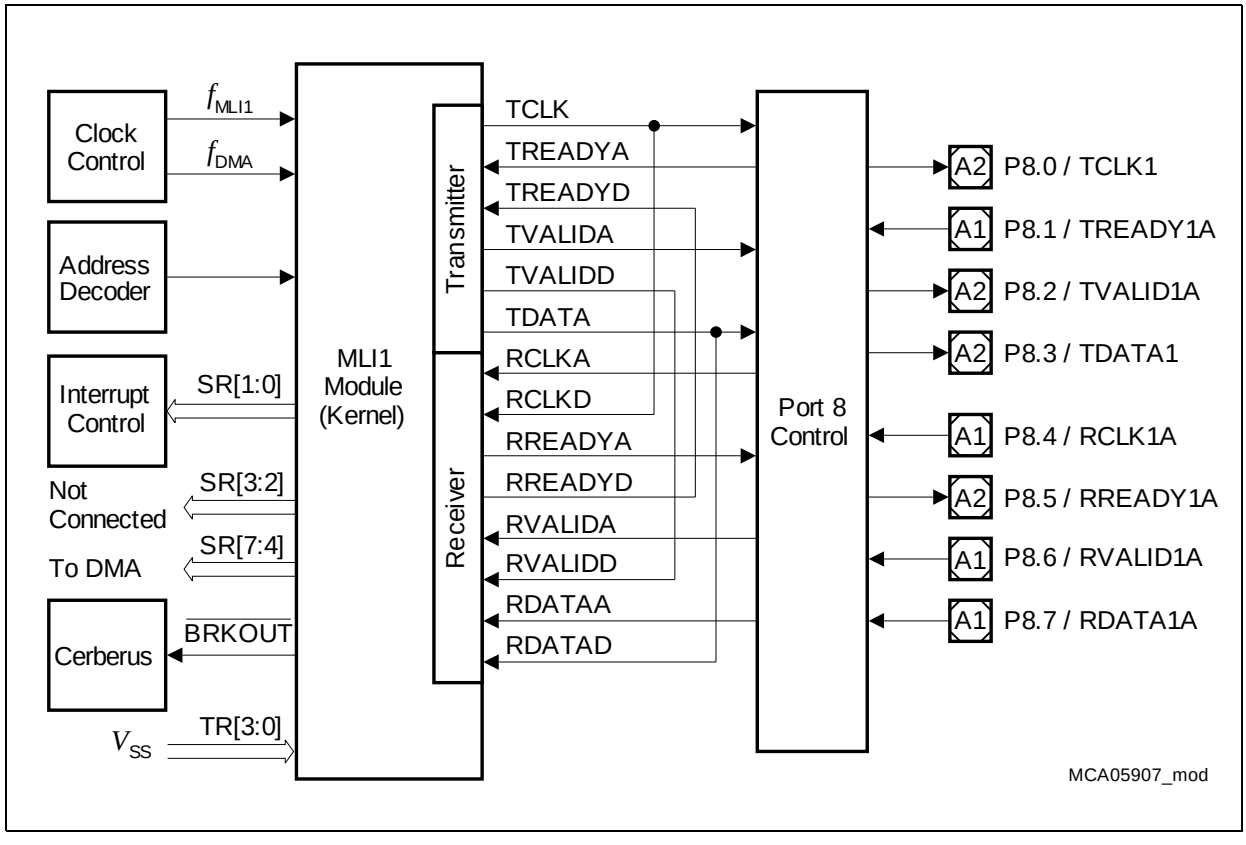


Figure 21-53 MLI1 Module Implementation and Interconnections

When programming the MLI1_OICR register, the following additional items must be considered:

- Lines with index “B” (not shown in the figure above)
  - Unused transmitter/receiver output lines TVALIDB and RREADYB are not connected.
  - Unused transmitter/receiver input lines TREADYB, RCLKB, RVALIDB, and RDATAB are connected to low level.
- Lines with index “C” (not shown in the figure above)
  - Unused transmitter/receiver output lines TVALIDC and RREADYC are reserved for emulation purposes.
  - Unused transmitter/receiver input lines TREADYC, RCLKC, RVALIDC, and RDATAC are reserved for emulation purposes and should not be selected during normal operation of the TC1797.

See also **Page 21-133** for additional details on I/O line control and function.

21.5.2 MLI Module External Registers

Figure 21-54 summarizes the module related external registers that are required for MLI0/MLI1 programming. Details on MLI0/MLI1 related register settings are shown in the following sections.

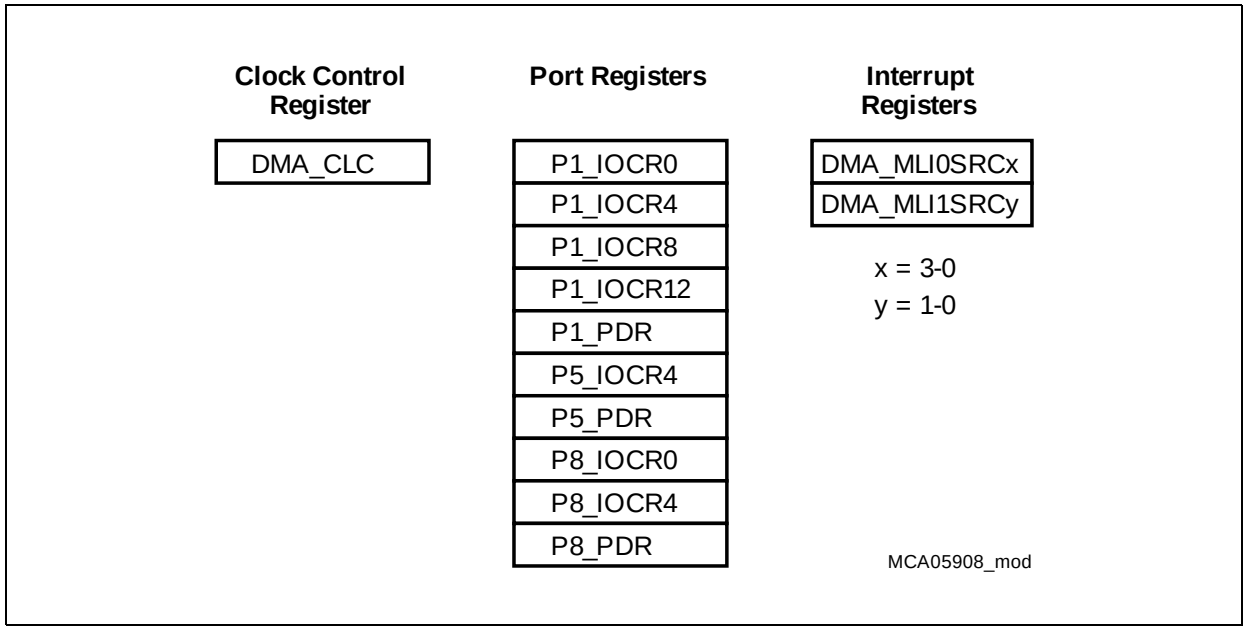


Figure 21-54 MLI0/MLI1 Implementation-Specific Special Function Registers

21.5.2.1 Automatic Register Overwrite

The following values are applied after reset (see **Page 21-68**).

- OICR = 1000 8000_H; Setting “A” is selected
- RCR.RCVRST = 0: the receiver is enabled for reception.

21.5.3 Module Clock Generation

The module clock generation configuration for the two MLI modules is shown in **Figure 21-55**.

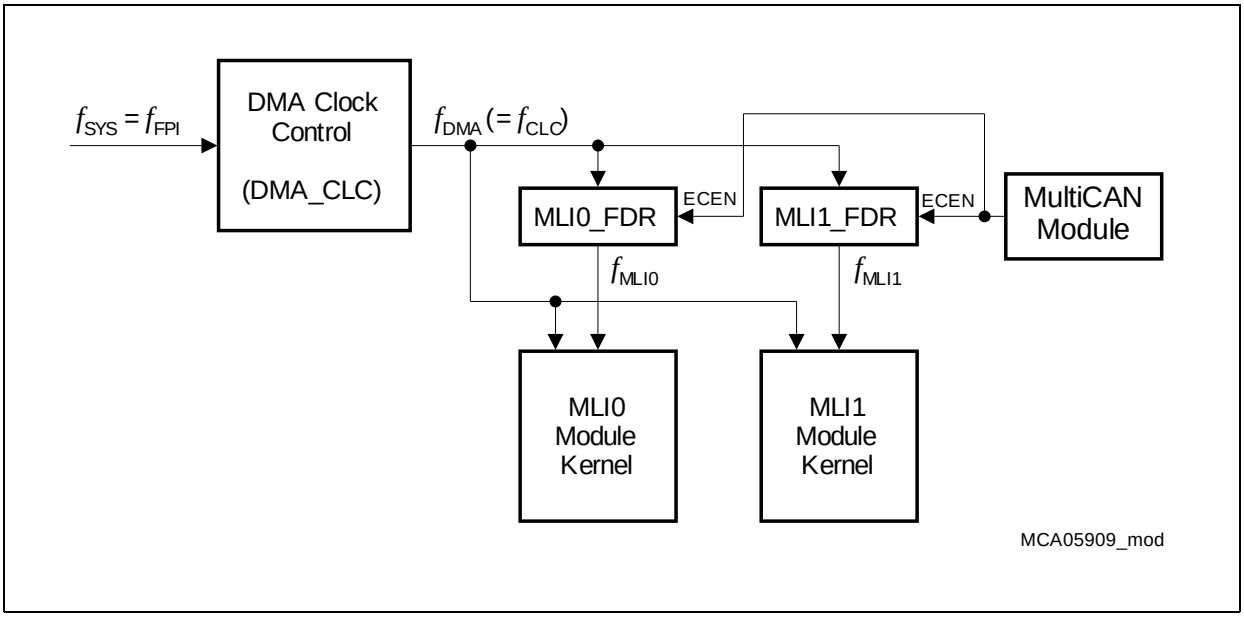


Figure 21-55 Clock Configuration of the MLI Modules

The DMA controller and the two MLI modules (MLI0 and MLI1) are supplied from a common module clock f_{DMA}, that has the frequency of the system clock f_{SYS} (= f_{FPI}) and is controlled via the DMA_CLC clock control register. The MLI modules do not have its own clock control registers. Its module clocks f_{MLI0} and f_{MLI1} are derived from f_{DMA} by two separate fractional divider registers, MLI0_FDR and MLI1_FDR (description see **Page 21-79**).

Output signal CAN_INT_O15 of the MultiCAN module can be used for external clock enable control of the fractional divider.

- f_{DMA}
  This is the module clock used inside the MLI kernels for control purposes such as for clocking of control logic and register operations. The clock control register DMA_CLC makes it possible to enable/disable f_{DMA} under certain conditions. DMA_CLC is described in the DMA chapter of this document.
- f_{MLI0} and f_{MLI1}
  This clock is the module clock used in the MLI kernels as base for the shift clock and therefore determines the baud rate of the synchronous serial data transmission. The fractional divider registers MLI0_FDR and MLI1_FDR control the frequencies of f_{MLI0} and f_{MLI1}. This configuration makes it possible to enable/disable the module clocks f_{MLI0} and f_{MLI1} independently of f_{DMA}.

Combined with the baud rate as derived in the MLI module (see **Equation (21.1)** on **Page 21-67**) and the MLIx_FDR fractional divider setup, the resulting MLI baud rate is defined by:

$$\text{Baud rate}_{MLIx} = \frac{f_{DMA}}{2} \times \frac{1}{n} \quad \text{with n = 1024 - FDR.STEP} \tag{21.4}$$

$$\text{Baud rate}_{MLIx} = \frac{f_{DMA}}{2} \times \frac{n}{1024} \quad \text{with n = 0-1023} \tag{21.5}$$

Equation (21.4) applies to normal divider mode of the fractional divider (FDR.DM = 01_B). **Equation (21.5)** applies to fractional divider mode (FDR.DM = 10_B).

After a reset operation, both MLI modules are enabled in normal divider mode. According the MLIx_FDR register's reset value of 03FF 43FF$_H$, the selected baud rate is $f_{DMA}/2$. Note that the DMA controller is also enabled after a reset operation with clock $f_{DMA} = f_{SYS}$.

21.5.4 Port Control and Connections

MLI0 and MLI1 clock and data output lines are connected to GPIO ports and are, therefore, controlled in the port logics (see also **Page 21-128** and **Page 21-129**). The following port control operations selections must be executed for these I/O lines:

- Input/output function selection (IOCR registers)
- Pad driver characteristics selection for the outputs (PDR registers)

21.5.4.1 Input/Output Function Selection

The port input/output control registers contain the bit fields that select the digital output and input driver characteristics such as port direction (input/output) with alternate output selection, pull-up/down devices, and open-drain selections. The I/O lines for the MLI modules are controlled by the Port 1, Port 5 and Port 8 input/output control registers. When the MLI modules are connected to the GPIO port lines, the correct settings of the enable/polarity control bits and bit fields in the output input control registers MLI0_IOCR and MLI1_IOCR must also be regarded (transmitter I/O line control see **Page 21-53**, receiver I/O line control see **Page 21-54**). Note that after a reset operation the MLI0 and MLI1 modules (although enabled) have no direct connections to the GPIO lines.

Table 21-11 shows how OICR register bits and bit fields must be programmed for the required GPIO functionality of the MLI I/O lines.

Table 21-11 MLI0 and MLI1 I/O Line Selection and Setup

Module	Port Lines	Input/Output Control Register Bits	I/O
MLI0	P1.3 / TREADY0B	P1_IOCR0.PC3 = $0XXX_B$ MLI0_OICR.TRE = 1 MLI0_OICR.TRP = X MLI0_OICR.TRS = 01_B	Input
	P1.4 / TCLK0	P1_IOCR4.PC4 = $1X01_B$ MLI0_OICR.TCE = 1 MLI0_OICR.TCP = X	Output
	P1.5 / TREADY0A	P1_IOCR4.PC5 = $0XXX_B$ MLI0_OICR.TRE = 1 MLI0_OICR.TRP = X MLI0_OICR.TRS = 00_B	Input
	P1.6 / TVALID0A	P1_IOCR4.PC6 = $1X01_B$ MLI0_OICR.TVEA = 1 MLI0_OICR.TVPA = X	Output
	P1.7 / TDATA0	P1_IOCR4.PC7 = $1X01_B$ MLI0_OICR.TDP = X	Output

Table 21-11 MLI0 and MLI1 I/O Line Selection and Setup (cont'd)

Module	Port Lines	Input/Output Control Register Bits	I/O
MLI0	P1.8 / RCLK0A	P1_IOCR8.PC8 = $0XXX_B$ MLI0_OICR.RCE = 1 MLI0_OICR.RCP = X MLI0_OICR.RCS = 00_B	Input
	P1.9 / RREADY0A	P1_IOCR8.PC9 = $1X01_B$ MLI0_OICR.RRS = 00_B MLI0_OICR.RRPA = X	Output
	P1.10 / RVALID0A	P1_IOCR8.PC10 = $0XXX_B$ MLI0_OICR.RVE = 1 MLI0_OICR.RVP = X MLI0_OICR.RVS = 00_B	Input
	P1.11 / RDATA0A	P1_IOCR8.PC11 = $0XXX_B$ MLI0_OICR.RDP = X MLI0_OICR.RDS = 00_B	Input
	P1.13 / RCLK0B	P1_IOCR12.PC13 = $0XXX_B$ MLI0_OICR.RCE = 1 MLI0_OICR.RCP = X MLI0_OICR.RCS = 01_B	Input
	P1.14 / RVALID0B	P1_IOCR12.PC14 = $0XXX_B$ MLI0_OICR.RVE = 1 MLI0_OICR.RVP = X MLI0_OICR.RVS = 01_B	Input
	P1.15 / RDATA0B	P1_IOCR12.PC15 = $0XXX_B$ MLI0_OICR.RDP = X MLI0_OICR.RDS = 01_B	Input
	P5.4 / RREADY0B	P5_IOCR4.PC4 = $1X10_B$ MLI0_OICR.RRS = 01_B MLI0_OICR.RRPB = X	Output
	P5.6 / TVALID0B	P5_IOCR4.PC6 = $1X10_B$ MLI0_OICR.TVEB = 1 MLI0_OICR.TVPB = X	Output

Table 21-11 MLI0 and MLI1 I/O Line Selection and Setup (cont'd)

Module	Port Lines	Input/Output Control Register Bits	I/O
MLI1	P8.0 / TCLK1	P8_IOCR0.PC0 = $1X11_B$ MLI1_OICR.TCE = 1 MLI1_OICR.TCP = X	Output
	P8.1 / TREADY1A	P8_IOCR0.PC1 = $0XXX_B$ MLI1_OICR.TRE = 1 MLI1_OICR.TRP = X MLI1_OICR.TRS = 00_B	Input
	P8.2 / TVALID1A	P8_IOCR0.PC2 = $1X11_B$ MLI1_OICR.TVEA = 1 MLI1_OICR.TVPA = X	Output
	P8.3 / TDATA1	P8_IOCR0.PC3 = $1X11_B$ MLI1_OICR.TDP = X	Output
	P8.4 / RCLK1A	P8_IOCR4.PC4 = $0XXX_B$ MLI1_OICR.RCE = 1 MLI1_OICR.RCP = X MLI1_OICR.RCS = 00_B	Input
	P8.5 / RREADY1A	P8_IOCR4.PC5 = $1X11_B$ MLI1_OICR.RRS = 00_B MLI1_OICR.RRPA = X	Output
	P8.6 / RVALID1A	P8_IOCR4.PC6 = $0XXX_B$ MLI1_OICR.RVE = 1 MLI1_OICR.RVP = X MLI1_OICR.RVS = 00_B	Input
	P8.7 / RDATA1A	P8_IOCR4.PC7 = $0XXX_B$ MLI1_OICR.RDP = X MLI1_OICR.RDS = 00_B	Input

21.5.5 On-Chip Connections

21.5.5.1 Service Request Output Connections

Each MLI module provides eight service request outputs SR[7:0] that can be used to generate interrupts or DMA requests. In the TC1797, four service request outputs SR[3:0] of the MLI0 module and two service request outputs SR[1:0] of the MLI1 module are connected to an interrupt node. Service request outputs SR[3:2] of the MLI1 module are not connected. Four service request outputs (SR[7:4]) of each MLI module are connected to DMA request inputs of the TC1797 DMA controller.

Each of the service request outputs used as interrupt requests are controlled by a service request control register. The service request control registers of the MLI modules are located inside the DMA address area. Therefore, all MLI0/MLI1 service request control registers are named as DMA_MLIxSRCy and described in the DMA chapter implementation part of the TC1797 User´s Manual.

All MLI service request output connections are listed in **Table 21-12**.

Table 21-12 Service Request Lines and Interconnections of MLI0/MLI1

Module	Service Req. Output Line	Connected to Node or DMA Request Input	Description
MLI0	SR0	DMA_MLI0SRC0	MLI0 Service Request Node 0 (in DMA)
	SR1	DMA_MLI0SRC1	MLI0 Service Request Node 1 (in DMA)
	SR2	DMA_MLI0SRC2	MLI0 Service Request Node 2 (in DMA)
	SR3	DMA_MLI0SRC3	MLI0 Service Request Node 3 (in DMA)
	SR4	CH00_REQI7	DMA Channel 00 Request Input 7
		CH04_REQI7	DMA Channel 04 Request Input 7
	SR5	CH01_REQI7	DMA Channel 01 Request Input 7
		CH05_REQI7	DMA Channel 05 Request Input 7
	SR6	CH02_REQI7	DMA Channel 02 Request Input 7
		CH06_REQI7	DMA Channel 06 Request Input 7
	SR7	CH03_REQI7	DMA Channel 03 Request Input 7
		CH07_REQI7	DMA Channel 07 Request Input 7

Table 21-12 Service Request Lines and Interconnections of MLI0/MLI1 (cont'd)

Module	Service Req. Output Line	Connected to Node or DMA Request Input	Description
MLI1	SR0	DMA_MLI1SRC0	MLI1 Service Request Node 0 (in DMA)
	SR1	DMA_MLI1SRC1	MLI1 Service Request Node 1 (in DMA)
	SR2	–	Not connected
	SR3	–	Not connected
	SR4	CH10_REQI14	DMA Channel 10 Request Input 15
		CH14_REQI14	DMA Channel 14 Request Input 15
	SR5	CH11_REQI6	DMA Channel 11 Request Input 15
		CH15_REQI5	DMA Channel 15 Request Input 15
	SR6	CH12_REQI5	DMA Channel 12 Request Input 15
		CH16_REQI6	DMA Channel 16 Request Input 15
	SR7	CH13_REQI5	DMA Channel 13 Request Input 15
		CH17_REQI6	DMA Channel 17 Request Input 15

21.5.5.2 Break Signals

The $\overline{\text{BRKOUT}}$ output signals of MLI0 and MLI1 are connected as break input signals to the Multi Core Break Switch (MCBS) that is a part of the Cerberus on-chip debug control module. These connections allow MLI0/MLI1 initiated break conditions to be generated in the Cerberus.

21.5.5.3 Trigger Input Signals

The five Trigger Input Signals TR[4:0] are connected to V_{SS}.

21.5.6 Access Protection

The access protection parameters for the MLI module in the TC1797 are identical with access protection parameters of the DMA Controller. Details of the access protection parameters are defined in the DMA chapter at "DMA Module Implementation" - "Access Protection Assignment".

The Table "DMA Access Protection Address Ranges" in the DMA chapter is also valid for MLI register bits AER.AENRx (x = 0-31).

The Tables "... Address Protection Sub-Range Definition" for PMI, OVRAM, DMI, and PCP PRAM in the DMA chapter are also valid for MLI register bits ARR.SLICEn and ARR.SIZEn (n = 0-3).

21.5.7 MLI0/MLI1 Transfer Window Address Maps

In the TC1797, the transfer windows for the MLI0 and MLI1 modules are located in the address ranges as identified in **Table 21-13**.

Table 21-13 MLI0/MLI1 Transfer Windows

Module	Window Type	Pipe	Address Range
MLI0	Small Transfer Window	Pipe 0	F01E 0000$_H$ to F01E 1FFF$_H$
		Pipe 1	F01E 2000$_H$ to F01E 3FFF$_H$
		Pipe 2	F01E 4000$_H$ to F01E 5FFF$_H$
		Pipe 3	F01E 6000$_H$ to F01E 7FFF$_H$
	Large Transfer Window	Pipe 0	F020 0000$_H$ to F020 FFFF$_H$
		Pipe 1	F021 0000$_H$ to F021 FFFF$_H$
		Pipe 2	F022 0000$_H$ to F022 FFFF$_H$
		Pipe 3	F023 0000$_H$ to F023 FFFF$_H$
MLI1	Small Transfer Window	Pipe 0	F01E 8000$_H$ to F01E 9FFF$_H$
		Pipe 1	F01E A000$_H$ to F01E BFFF$_H$
		Pipe 2	F01E C000$_H$ to F01E DFFF$_H$
		Pipe 3	F01E E000$_H$ to F01E FFFF$_H$
	Large Transfer Window	Pipe 0	F024 0000$_H$ to F024 FFFF$_H$
		Pipe 1	F025 0000$_H$ to F025 FFFF$_H$
		Pipe 2	F026 0000$_H$ to F026 FFFF$_H$
		Pipe 3	F027 0000$_H$ to F027 FFFF$_H$

21.5.8 MLI0/MLI1 Address Map

An absolute register address is given by the offset address of the register (given in **Table 21-10**) plus the module base address (given in **Table 21-9**).

Table 21-14 Address Map of MLI0/MLI1

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
Multi Link Interface 0 (MLI0)					
–	Reserved	$F010\ C000_H$	nBE	SV, E	–
–	Reserved	$F010\ C004_H$	nBE	nBE	–
MLI0_ ID	MLI0 Module Identification Register	$F010\ C008_H$	U, SV	BE	$0025\ C0XX_H$
MLI0_ FDR	MLI0 Fractional Divider Register	$F010\ C00C_H$	U, SV	SV, E	$03FF\ 43FF_H$
MLI0_ TCR	MLI0 Transmitter Control Register	$F010\ C010_H$	U, SV	U, SV	$0000\ 0110_H$
MLI0_ TSTATR	MLI0 Transmitter Status Register	$F010\ C014_H$	U, SV	BE	$0000\ 0000_H$
MLI0_ TP0STATR	MLI0 Transmitter Pipe 0 Status Register	$F010\ C018_H$	U, SV	BE	$0000\ 0000_H$
MLI0_ TP1STATR	MLI0 Transmitter Pipe 1 Status Register	$F010\ C01C_H$	U, SV	BE	$0000\ 0000_H$
MLI0_ TP2STATR	MLI0 Transmitter Pipe 2 Status Register	$F010\ C020_H$	U, SV	BE	$0000\ 0000_H$
MLI0_ TP3STATR	MLI0 Transmitter Pipe 3 Status Register	$F010\ C024_H$	U, SV	BE	$0000\ 0000_H$
MLI0_ TCMDR	MLI0 Transmitter Command Register	$F010\ C028_H$	U, SV	U, SV	$0000\ 0000_H$
MLI0_ TRSTATR	MLI0 Transmitter Registers Status Register	$F010\ C02C_H$	U, SV	BE	$0000\ 0000_H$
MLI0_ TP0AOFR	MLI0 Transmitter Pipe 0 Address Offset Register	$F010\ C030_H$	U, SV	BE	$0000\ 0000_H$
MLI0_ TP1AOFR	MLI0 Transmitter Pipe 1 Address Offset Register	$F010\ C034_H$	U, SV	BE	$0000\ 0000_H$
MLI0_ TP2AOFR	MLI0 Transmitter Pipe 2 Address Offset Register	$F010\ C038_H$	U, SV	BE	$0000\ 0000_H$

Table 21-14 Address Map of MLI0/MLI1 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
MLI0_ TP3AOFR	MLI0 Transmitter Pipe 3 Address Offset Register	F010 C03C$_H$	U, SV	BE	0000 0000$_H$
MLI0_ TP0DATAR	MLI0 Transmitter Pipe 0 Data Register	F010 C040$_H$	U, SV	BE	0000 0000$_H$
MLI0_ TP1DATAR	MLI0 Transmitter Pipe 1 Data Register	F010 C044$_H$	U, SV	BE	0000 0000$_H$
MLI0_ TP2DATAR	MLI0 Transmitter Pipe 2 Data Register	F010 C048$_H$	U, SV	BE	0000 0000$_H$
MLI0_ TP3DATAR	MLI0 Transmitter Pipe 3 Data Register	F010 C04C$_H$	U, SV	BE	0000 0000$_H$
MLI0_ TDRAR	MLI0 Transmitter Data Read Answer Register	F010 C050$_H$	U, SV	U, SV	0000 0000$_H$
MLI0_ TP0BAR	MLI0 Transmitter Pipe 0 Base Address Register	F010 C054$_H$	U, SV	U, SV	0000 0000$_H$
MLI0_ TP1BAR	MLI0 Transmitter Pipe 1 Base Address Register	F010 C058$_H$	U, SV	U, SV	0000 0000$_H$
MLI0_ TP2BAR	MLI0 Transmitter Pipe 2 Base Address Register	F010 C05C$_H$	U, SV	U, SV	0000 0000$_H$
MLI0_ TP3BAR	MLI0 Transmitter Pipe 3 Base Address Register	F010 C060$_H$	U, SV	U, SV	0000 0000$_H$
MLI0_ TCBAR	MLI0 Transmitter Copy Base Address Register	F010 C064$_H$	U, SV	BE	0000 0000$_H$
MLI0_ RCR	MLI0 Receiver Control Register	F010 C068$_H$	U, SV	U, SV	0100 0000$_H$
MLI0_ RP0BAR	MLI0 Receiver Pipe 0 Base Address Register	F010 C06C$_H$	U, SV	BE	0000 0000$_H$
MLI0_ RP1BAR	MLI0 Receiver Pipe 1 Base Address Register	F010 C070$_H$	U, SV	BE	0000 0000$_H$
MLI0_ RP2BAR	MLI0 Receiver Pipe 2 Base Address Register	F010 C074$_H$	U, SV	BE	0000 0000$_H$
MLI0_ RP3BAR	MLI0 Receiver Pipe 3 Base Address Register	F010 C078$_H$	U, SV	BE	0000 0000$_H$

Table 21-14 Address Map of MLI0/MLI1 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
MLI0_ RP0STATR	MLI0 Receiver Pipe 0 Status Register	F010 C07C$_H$	U, SV	BE	0000 0000$_H$
MLI0_ RP1STATR	MLI0 Receiver Pipe 1 Status Register	F010 C080$_H$	U, SV	BE	0000 0000$_H$
MLI0_ RP2STATR	MLI0 Receiver Pipe 2 Status Register	F010 C084$_H$	U, SV	BE	0000 0000$_H$
MLI0_ RP3STATR	MLI0 Receiver Pipe 3 Status Register	F010 C088$_H$	U, SV	BE	0000 0000$_H$
MLI0_ RADRR	MLI0 Receiver Address Register	F010 C08C$_H$	U, SV	BE	0000 0000$_H$
MLI0_ RDATAR	MLI0 Receiver Data Register	F010 C090$_H$	U, SV	BE	0000 0000$_H$
MLI0_ SCR	MLI0 Set Clear Register	F010 C094$_H$	U, SV	U, SV	0000 0000$_H$
MLI0_ TIER	MLI0 Transmitter Interrupt Enable Register	F010 C098$_H$	U, SV	U, SV	0000 0000$_H$
MLI0_ TISR	MLI0 Transmitter Interrupt Status Register	F010 C09C$_H$	U, SV	BE	0000 0000$_H$
MLI0_ TINPR	MLI0 Transmitter Interrupt Node Pointer Register	F010 C0A0$_H$	U, SV	U, SV	0000 0000$_H$
MLI0_ RIER	MLI0 Receiver Interrupt Enable Register	F010 C0A4$_H$	U, SV	U, SV	0000 0000$_H$
MLI0_ RISR	MLI0 Receiver Interrupt Status Register	F010 C0A8$_H$	U, SV	BE	0000 0000$_H$
MLI0_ RINPR	MLI0 Receiver Interrupt Node Pointer Register	F010 C0AC$_H$	U, SV	U, SV	0000 0000$_H$
MLI0_ GINTR	MLI0 Global Interrupt Set Register	F010 C0B0$_H$	U, SV	U, SV	0000 0000$_H$
MLI0_ OICR	MLI0 Output Input Control Register	F010 C0B4$_H$	U, SV	U, SV	1000 8000$_H$
MLI0_ AER	MLI0 Access Enable Register	F010 C0B8$_H$	U, SV	SV, E	0000 0000$_H$

Table 21-14 Address Map of MLI0/MLI1 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
MLI0_ ARR	MLI0 Access Range Register	F010 C0BC$_H$	U, SV	SV, E	0000 0000$_H$
–	Reserved	F010 C0C0$_H$ - F010 C0FC$_H$	BE	BE	–
Micro Link Interface 1 (MLI1)					
–	Reserved	F010 C100$_H$	nBE	SV, E	–
–	Reserved	F010 C104$_H$	nBE	nBE	–
MLI1_ ID	MLI1 Module Identification Register	F010 C108$_H$	U, SV	BE	0025 C0XX$_H$
MLI1_ FDR	MLI1 Fractional Divider Register	F010 C10C$_H$	U, SV	SV, E	03FF 43FF$_H$
MLI1_ TCR	MLI1 Transmitter Control Register	F010 C110$_H$	U, SV	U, SV	0000 0110$_H$
MLI1_ TSTATR	MLI1 Transmitter Status Register	F010 C114$_H$	U, SV	BE	0000 0000$_H$
MLI1_ TP0STATR	MLI1 Transmitter Pipe 0 Status Register	F010 C118$_H$	U, SV	BE	0000 0000$_H$
MLI1_ TP1STATR	MLI1 Transmitter Pipe 1 Status Register	F010 C11C$_H$	U, SV	BE	0000 0000$_H$
MLI1_ TP2STATR	MLI1 Transmitter Pipe 2 Status Register	F010 C120$_H$	U, SV	BE	0000 0000$_H$
MLI1_ TP3STATR	MLI1 Transmitter Pipe 3 Status Register	F010 C124$_H$	U, SV	BE	0000 0000$_H$
MLI1_ TCMDR	MLI1 Transmitter Command Register	F010 C128$_H$	U, SV	U, SV	0000 0000$_H$
MLI1_ TRSTATR	MLI1 Transmitter Registers Status Register	F010 C12C$_H$	U, SV	BE	0000 0000$_H$
MLI1_ TP0AOFR	MLI1 Transmitter Pipe 0 Address Offset Register	F010 C130$_H$	U, SV	BE	0000 0000$_H$
MLI1_ TP1AOFR	MLI1 Transmitter Pipe 1 Address Offset Register	F010 C134$_H$	U, SV	BE	0000 0000$_H$
MLI1_ TP2AOFR	MLI1 Transmitter Pipe 2 Address Offset Register	F010 C138$_H$	U, SV	BE	0000 0000$_H$

Table 21-14 Address Map of MLI0/MLI1 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
MLI1_ TP3AOFR	MLI1 Transmitter Pipe 3 Address Offset Register	F010 C13C$_H$	U, SV	BE	0000 0000$_H$
MLI1_ TP0DATAR	MLI1 Transmitter Pipe 0 Data Register	F010 C140$_H$	U, SV	BE	0000 0000$_H$
MLI1_ TP1DATAR	MLI1 Transmitter Pipe 1 Data Register	F010 C144$_H$	U, SV	BE	0000 0000$_H$
MLI1_ TP2DATAR	MLI1 Transmitter Pipe 2 Data Register	F010 C148$_H$	U, SV	BE	0000 0000$_H$
MLI1_ TP3DATAR	MLI1 Transmitter Pipe 3 Data Register	F010 C14C$_H$	U, SV	BE	0000 0000$_H$
MLI1_ TDRAR	MLI1 Transmitter Data Read Answer Register	F010 C150$_H$	U, SV	U, SV	0000 0000$_H$
MLI1_ TP0BAR	MLI1 Transmitter Pipe 0 Base Address Register	F010 C154$_H$	U, SV	U, SV	0000 0000$_H$
MLI1_ TP1BAR	MLI1 Transmitter Pipe 1 Base Address Register	F010 C158$_H$	U, SV	U, SV	0000 0000$_H$
MLI1_ TP2BAR	MLI1 Transmitter Pipe 2 Base Address Register	F010 C15C$_H$	U, SV	U, SV	0000 0000$_H$
MLI1_ TP3BAR	MLI1 Transmitter Pipe 3 Base Address Register	F010 C160$_H$	U, SV	U, SV	0000 0000$_H$
MLI1_ TCBAR	MLI1 Transmitter Copy Base Address Register	F010 C164$_H$	U, SV	BE	0000 0000$_H$
MLI1_ RCR	MLI1 Receiver Control Register	F010 C168$_H$	U, SV	U, SV	0100 0000$_H$
MLI1_ RP0BAR	MLI1 Receiver Pipe 0 Base Address Register	F010 C16C$_H$	U, SV	BE	0000 0000$_H$
MLI1_ RP1BAR	MLI1 Receiver Pipe 1 Base Address Register	F010 C170$_H$	U, SV	BE	0000 0000$_H$
MLI1_ RP2BAR	MLI1 Receiver Pipe 2 Base Address Register	F010 C174$_H$	U, SV	BE	0000 0000$_H$
MLI1_ RP3BAR	MLI1 Receiver Pipe 3 Base Address Register	F010 C178$_H$	U, SV	BE	0000 0000$_H$

Table 21-14 Address Map of MLI0/MLI1 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
MLI1_ RP0STATR	MLI1 Receiver Pipe 0 Status Register	F010 C17C$_H$	U, SV	BE	0000 0000$_H$
MLI1_ RP1STATR	MLI1 Receiver Pipe 1 Status Register	F010 C180$_H$	U, SV	BE	0000 0000$_H$
MLI1_ RP2STATR	MLI1 Receiver Pipe 2 Status Register	F010 C184$_H$	U, SV	BE	0000 0000$_H$
MLI1_ RP3STATR	MLI1 Receiver Pipe 3 Status Register	F010 C188$_H$	U, SV	BE	0000 0000$_H$
MLI1_ RADRR	MLI1 Receiver Address Register	F010 C18C$_H$	U, SV	BE	0000 0000$_H$
MLI1_ RDATAR	MLI1 Receiver Data Register	F010 C190$_H$	U, SV	BE	0000 0000$_H$
MLI1_ SCR	MLI1 Set Clear Register	F010 C194$_H$	U, SV	U, SV	0000 0000$_H$
MLI1_ TIER	MLI1 Transmitter Interrupt Enable Register	F010 C198$_H$	U, SV	SV	0000 0000$_H$
MLI1_ TISR	MLI1 Transmitter Interrupt Status Register	F010 C19C$_H$	U, SV	BE	0000 0000$_H$
MLI1_ TINPR	MLI1 Transmitter Interrupt Node Pointer Register	F010 C1A0$_H$	U, SV	U, SV	0000 0000$_H$
MLI1_ RIER	MLI1 Receiver Interrupt Enable Register	F010 C1A4$_H$	U, SV	U, SV	0000 0000$_H$
MLI1_ RISR	MLI1 Receiver Interrupt Status Register	F010 C1A8$_H$	U, SV	BE	0000 0000$_H$
MLI1_ RINPR	MLI1 Receiver Interrupt Node Pointer Register	F010 C1AC$_H$	U, SV	U, SV	0000 0000$_H$
MLI1_ GINTR	MLI1 Global Interrupt Set Register	F010 C1B0$_H$	U, SV	U, SV	0000 0000$_H$
MLI1_ OICR	MLI1 Output Input Control Register	F010 C1B4$_H$	U, SV	U, SV	1000 8000$_H$
MLI1_ AER	MLI1 Access Enable Register	F010 C1B8$_H$	U, SV	SV, E	0000 0000$_H$

Table 21-14 Address Map of MLI0/MLI1 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
MLI1_ ARR	MLI1 Access Range Register	F010 C1BC$_H$	U, SV	SV, E	0000 0000$_H$
–	Reserved	F010 C1C0$_H$ - F010 C1CC$_H$	BE	BE	–

22 General Purpose Timer Array (GPTA®v5)

This chapter describes the General Purpose Timer Array of the TC1797. The GPTA[1] consists of the following units: GPTA0 and GPTA1 with identical functionality; LTCA2 with reduced GPTA0 functionality.

This chapter contains the following sections:

- A summary on the structure and basic functionalities (see **Page 22-4**)
- Functional description of the GPTA®v5 kernel, applicable for GPTA0 and GPTA1 (see **Page 22-8**)
- Register descriptions of all GPTA®v5 kernel specific registers, applicable for GPTA0 and GPTA1 (see **Page 22-160**)
- Functional description of the LTCA2 kernel (see **Page 22-234**)
- Register descriptions of all LTCA2 kernel specific registers (see **Page 22-250**)
- TC1797 implementation-specific details and registers of the GPTA®v5 module, including port connections and control, interrupt control, address decoding, and clock control (see **Page 22-274**).

*Note: The GPTA®v5 kernel register names described in **Section 22.4**, **Section 22.6**, and **Section 22.7.2** will be referenced in the TC1797 User´s Manual by the unit name prefix “GPTA0_” for the GPTA0 unit, by “GPTA1_” for the GPTA1 unit, and by “LTCA2_” for the LTCA2 unit.*

22.1 What is new?

The major updates from GPTAv4 to GPTAv5 are:

- The flexibility to generate on-chip trigger and gating signals have been increased. The GPTAv5 provides 16 such signals. Each of the signals may be mapped to any output signal of a Local or Global Timer Cell. Therefore it is not limited as before to a single group of Global or Local Timer Cells (25% of the GTC or LTC). Limitation now is, that no more than 4 different on-chip trigger and gating signals may be mapped to one group of LTC or GTC. Details concerning this new on-chip trigger and gating signal multiplexer are described in **Section 22.3.4.3** (see **Page 22-108**). This new features is not fully upwards compatible to the GPTAv4. Additional output multiplexer registers have to be configured to achieve the same functionality (see **“Multiplexer Register Array Programming” on Page 22-121**). Some very minor issue may occur due to a minor reduction of on-chip signal and trigger signals compared to GPTAv4, but on the other hand the increased flexibility should nearly always compensated this. The following list summarizes the principle of mapping former GPTAv4 signals to the new GPTAv5 signals:
  – GPTAv4 Signal GPTA0_OUT0 is replaced by GPTAv5 Signal GPTA0_TRIG01
  – GPTAv4 Signal GPTA0_OUT1 is replaced by GPTAv5 Signal GPTA0_TRIG11

1) *TriCore®, C166®, Infineon®, Infineon Technologies®, and GPTA® are trademarks of Infineon Technologies AG.*

- GPTAv4 Signal GPTA0_OUT2 is replaced by GPTAv5 Signal GPTA0_TRIG00
- GPTAv4 Signal GPTA0_OUT3 is replaced by GPTAv5 Signal GPTA0_TRIG10
- GPTAv4 Signal GPTA0_OUT8 is replaced by GPTAv5 Signal GPTA0_TRIG03
- GPTAv4 Signal GPTA0_OUT9 is replaced by GPTAv5 Signal GPTA0_TRIG13
- GPTAv4 Signal GPTA0_OUT10 is replaced by GPTAv5 Signal GPTA0_TRIG02
- GPTAv4 Signal GPTA0_OUT11 is replaced by GPTAv5 Signal GPTA0_TRIG12
- GPTAv4 Signal GPTA0_OUT16 is replaced by GPTAv5 Signal GPTA0_TRIG05
- GPTAv4 Signal GPTA0_OUT18 is replaced by GPTAv5 Signal GPTA0_TRIG15
- GPTAv4 Signal GPTA0_OUT19 is replaced by GPTAv5 Signal GPTA0_TRIG04
- GPTAv4 Signal GPTA0_OUT24 is replaced by GPTAv5 Signal GPTA0_TRIG07
- GPTAv4 Signal GPTA0_OUT26 is replaced by GPTAv5 Signal GPTA0_TRIG17
- GPTAv4 Signal GPTA0_OUT27 is replaced by GPTAv5 Signal GPTA0_TRIG06
- GPTAv4 Signal GPTA0_OUT28 is replaced by GPTAv5 Signal GPTA0_TRIG07
- GPTAv4 Signal GPTA0_OUT4 is no longer available in GPTAv5. This signal was routed to the ERU (TC1766 only) to cover 75% of the GTC and LTC cells as input to Input channel 1. But Signal GPTA0_TRIG12 may be routed to all (100%) GTC and LTC cells due to the on-chip trigger and gating multiplexer and therefore fulfills this requirement already.
- GPTAv4 Signal GPTA0_OUT7 is no longer available in GPTAv5. This signal was routed to the ERU (TC1766 only) to cover 75% of the GTC and LTC cells as input to Input channel 2. But Signal GPTA0_TRIG14 may be routed to all (100%) GTC and LTC cells due to the on-chip trigger and gating multiplexer and therefore fulfills this requirement already.
- To be consistent to TC1797, the double connected input group of IOG3 is renamed to IOG6 and the Output Group OG1-7 are renamed to OG0-OG6 and the OG0 is renamed to IOG7.
- GPTAv4 Signal GPTA0_OUT17 is no longer available in GPTAv5. This signal was routed to the ERU to cover 50% of the GTC and LTC cells as input to Input channel 2. But Signal GPTA0_TRIG14 may be routed to all (100%) GTC and LTC cells due to the on-chip trigger and gating multiplexer and therefore fulfills this requirement already.
- GPTAv4 Signal GPTA0_OUT22 is no longer available in GPTAv5. This signal was routed to the ERU (TC1766 only) to cover 75% of the GTC and LTC cells as input to Input channel 3. But Signal GPTA0_TRIG16 may be routed to all (100%) GTC and LTC cells due to the on-chip trigger and gating multiplexer and therefore fulfills this requirement already.
- GPTAv4 Signal GPTA0_OUT25 is no longer available in GPTAv5. This signal was routed to the ERU to cover 50% of the GTC and LTC cells as input to Input channel 3. But Signal GPTA0_TRIG16 may be routed to all (100%) GTC and LTC cells due to the on-chip trigger and gating multiplexer and therefore fulfills this requirement already.
- GPTAv4 Signal GPTA0_OUT5 is no longer required (Time Trigger CAN) but GPTAv5 Signal GPTA0_TRIG05 is reserved for it.

- To improve effective usage of the Local Timer Cells, a new cell bypassing, so called global bypass, is introduced. This bypassing enables more flexible cell allocation and also reduces the number of LTC required for coherent update. Details on the two different Local Timer Cell Bypass mechanism may be found in the section **"Data Output Line Control" on Page 22-73**. Two different application examples using the global and local bypass may be found in **Section 22.3.3.5**(see **Page 22-85**). This new features is upwards compatible to the GPTAv4.
- Due to the new bypassing mechanism, a new coherent update mechanism has been introduced, the Local Coherent update described within **Section 22.3.3.5** (see **Page 22-85**). This new local coherent update or double action principle, is very useful to update single Local Timer Cells or a couple of Local Timer Cells within a Group sequentially (not simultaneously) without signal distortion (no other signal output beside the previously configured and the new configured). The new update principle allows to update a local timer cell within a group of local timer cells independent of other local timer cells and therefore also not synchronous/coherent to other local timer cells. This new mechanism upgrades the older mechanism of global coherent update. This older principle of global coherent is very useful to update a number of Local Timer Cells simultaneously. This new features is upwards compatible to the GPTAv4.
- The GPTA0/GPTA1, and LTCA2 OUTs are additionally assigned to new ports. Eight new outputs on Port 0, eight new outputs on Port 1, one new output on Port 2, two new outputs on Port 3, fourteen new outputs on Port 5, four new outputs on Port 14, sixteen new outputs on Port 13, and twelve new outputs on Port 14.
- Within the LTCA the number of LTC cells have been reduced from 64 down to 32. The input multiplexer matrix and the output multiplexer matrix has been adapted respectively, so only 4 IO Groups and 4 Output groups are implemented.
- To enable a family concept between TC1797 and TC1767, the GPTA to MSC Interconnection Assignment of MSC0 and MSC1 has been changed. Details can be found in **Section 22.7.4.2** (see **Page 22-287**).
- To fix a design bug for TC1797 and TC1767, the input line IN1 of the GPTA1 now switches the common input of GPTA0/GPTA1/LTCA2 units for connecting to the output of a 4-to-1 multiplexer. This multiplexer is controlled by bit field SCU_SYSCON.GPTAIS and allows the GPTA0/GPTA1/LTCA2 input IN1 to be connected to one out of four port input lines.
- GPTA1 provided the clock base for LTCA2 within the GPTAv4 version. This disables a family concept of products only having a GPTA0 and an LTCA2 (e.g. TC1767). Therefore GPTA0 is now used as clock source for the LTCA2 and GPTA1 is used as clock source for LTCA3.
- The common IN0 of GPTA0/GPTA1/LTCA2 is multiplexed within the SCU to connect either to a port pin or the EXTCLK0 (see **Page 22-295**).

22.2 GPTA®v5 Overview

The TC1797 contains the two General Purpose Timer Arrays (GPTA0 and GPTA1) with identical functionality, plus the additional Local Timer Cell Array (LTCA2). **Figure 22-1** shows a global view of the GPTA®v5 units.

The GPTA®v5 provides a set of timer, compare, and capture functionalities that can be flexibly combined to form signal measurement and signal generation cells. They are optimized for tasks typical of engine, gearbox, and electrical motor control applications, but can also be used to generate simple and complex signal waveforms required for other industrial applications.

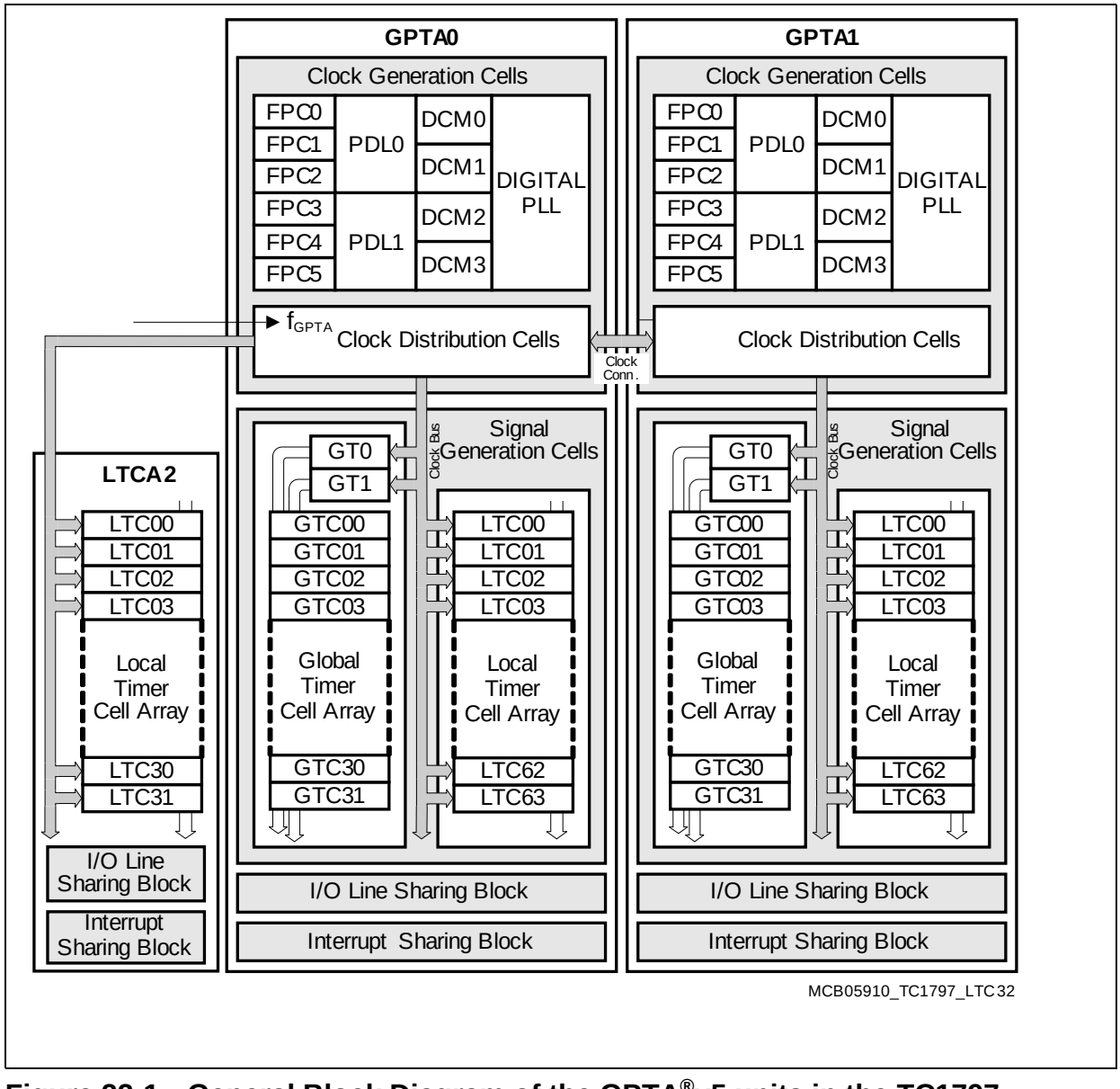


Figure 22-1 General Block Diagram of the GPTA®v5 units in the TC1797

22.2.1 Functionality of GPTA0 and GPTA1

The General Purpose Timer Arrays (GPTA0 and GPTA1) each provides a set of hardware cells required for high-speed digital signal processing:

- Filter and Prescaler Cells (FPC) support input noise filtering and prescaler operation.
- Phase Discrimination Logic cells (PDL) decode the direction information output by a rotation tracking system.
- Duty Cycle Measurement Cells (DCM) provide pulse-width measurement capabilities.
- A Digital Phase Locked Loop cell (PLL) generates a programmable number of GPTA®v5 unit ticks during an input signal's period.
- Global Timer cells (GT) driven by various clock sources are implemented to operate as a time base for the associated Global Timer Cells.
- Global Timer Cells (GTC) can be programmed to capture the contents of a Global Timer on an external or internal event. A GTC may also be used to control an external port pin depending on the result of an internal compare operation. GTCs can be logically concatenated to provide a common external port pin with a complex signal waveform.
- Local Timer Cells (LTC) operating in Timer, Capture, or Compare Mode may also be logically tied together to drive a common external port pin with a complex signal waveform. LTCs – enabled in Timer Mode or Capture Mode – can be clocked or triggered by various external or internal events.
- On-chip Trigger and Gating Signals (OTGS) can be configured to provide trigger or gating signals to integrated peripherals (GPTA0 only).

Input lines can be shared by an LTC and a GTC to trigger their programmed operation simultaneously.

The following list summarizes the specific features of the GPTA®v5 cells.

Clock Generation Cells

- Filter and Prescaler Cell (FPC)
  - Six independent cells
  - Three basic operating modes:
    Prescaler, Delayed Debounce Filter, Immediate Debounce Filter
  - Selectable input sources:
    Port lines, GPTA®v5 unit clock, FPC output of preceding FPC cell
  - Selectable input clocks:
    GPTA®v5 unit clock, prescaled GPTA®v5 unit clock, DCM clock, compensated or uncompensated PLL clock.
  - $f_{GPTA}/2$ maximum input signal frequency in Filter Modes
- Phase Discriminator Logic (PDL)
  - Two independent cells
  - Two operating modes (2- and 3- sensor signals)

  - f_{GPTA}/4 maximum input signal frequency in 2-sensor Mode, f_{GPTA}/6 maximum input signal frequency in 3-sensor Mode
- Duty Cycle Measurement (DCM)
  - Four independent cells
  - 0 - 100% margin and time-out handling
  - f_{GPTA} maximum resolution
  - f_{GPTA}/2 maximum input signal frequency
- Digital Phase Locked Loop (PLL)
  - One cell
  - Arbitrary multiplication factor between 1 and 65535
  - f_{GPTA} maximum resolution
  - f_{GPTA}/2 maximum input signal frequency
- Clock Distribution Cells (CDC)
  - One unit
  - Provides nine clock output signals:
    f_{GPTA}, divided f_{GPTA} clocks, FPC1/FPC4 outputs, DCM clock, LTC prescaler clock

Signal Generation Cells

- Global Timers (GT)
  - Two independent cells
  - Two operating modes (Free-Running Timer and Reload Timer)
  - 24-bit data width
  - f_{GPTA} maximum resolution
  - f_{GPTA}/2 maximum input signal frequency
- Global Timer Cell (GTC)
  - 32 cells related to the Global Timers
  - Two operating modes (Capture, Compare and Capture after Compare)
  - 24-bit data width
  - f_{GPTA} maximum resolution
  - f_{GPTA}/2 maximum input signal frequency
- Local Timer Cell (LTC)
  - 64 independent cells
  - Three basic operating modes (Timer, Capture and Compare) for 63 cells
  - Special compare modes for one cell
  - 16-bit data width
  - f_{GPTA} maximum resolution
  - f_{GPTA}/2 maximum input signal frequency

Interrupt Sharing Block

- 286 interrupt sources, generating up to 92 service requests

On-chip Trigger Block

- 16 on-chip trigger signals

I/O Sharing Block

- Interconnecting inputs and outputs from internal clocks, FPC, GTC, LTC, ports, and MSC interface

22.2.2 Functionality of LTCA2

The Local Timer Cell Array (LTCA2) provide a set of hardware cells required for high-speed digital signal processing:

- Local Timer Cells (LTC) operating in Timer, Capture, or Compare Mode may also be logically tied together to drive a common external port pin with a complex signal waveform. LTCs – enabled in Timer Mode or Capture Mode – can be clocked or triggered by various external or internal events.

The following list summarizes the specific features of the LTCA cells.

Signal Generation Cells

- Local Timer Cell (LTC)
  - 32 independent cells
  - Three basic operating modes (Timer, Capture and Compare) for 31 cells
  - Special compare modes for one cell
  - 16-bit data width
  - f_{GPTA} maximum resolution
  - $f_{GPTA}/2$ maximum input signal frequency

I/O Sharing Block

- Interconnecting inputs and outputs from internal clocks, LTC, ports, and MSC interface

22.3 GPTA0/GPTA1 Kernel Description

The functionality of the General Purpose Timer Arrays GPTA0/GPTA1 kernel is described in this section. Clock control, address decoding, and service (interrupt) request control are managed outside the GPTA0/GPTA1 unit kernel.

Figure 22-2 shows a global unit diagram of the GPTA®v5 unit kernel.

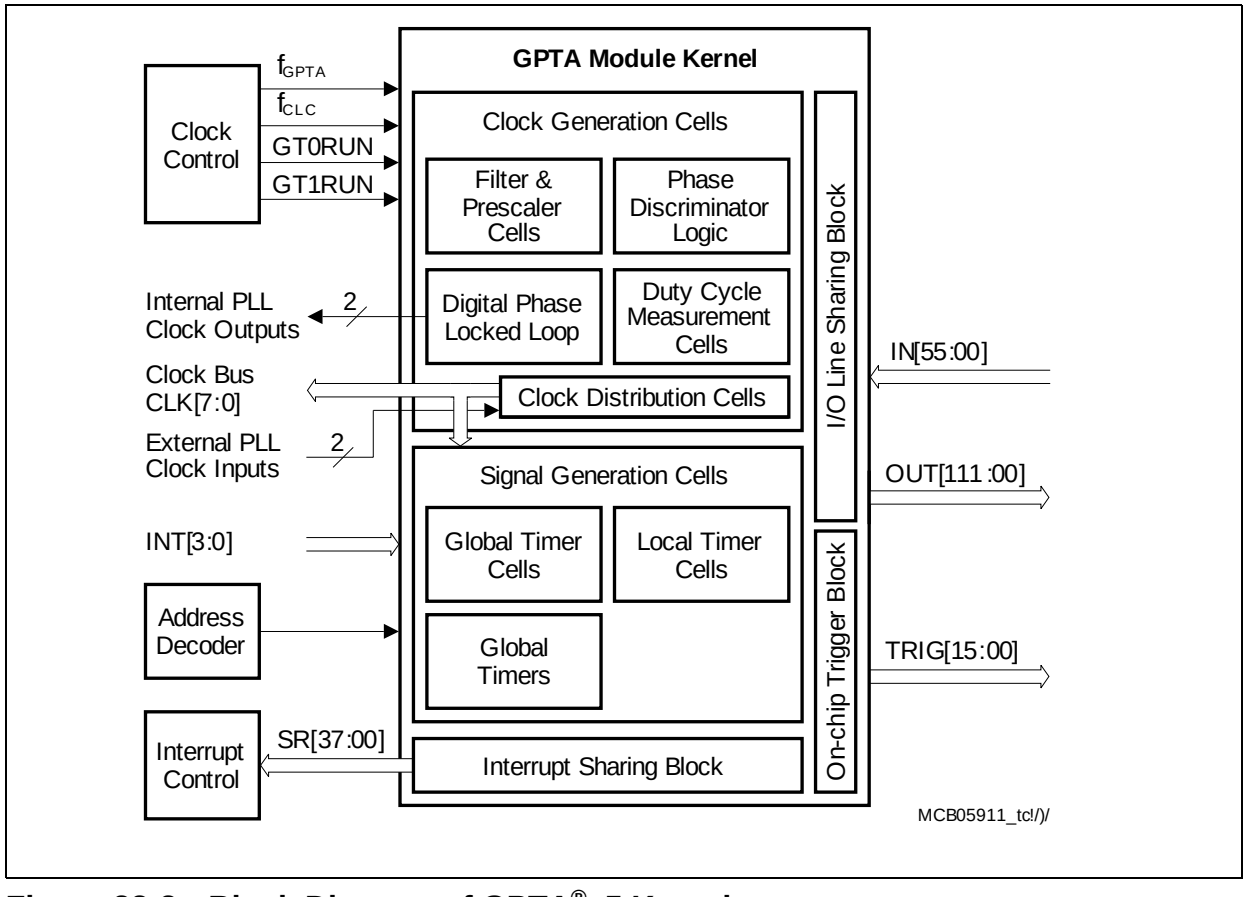


Figure 22-2 Block Diagram of GPTA®v5 Kernel

Each GPTA0/1 kernel has 56 input signals, 112 output signals, and four input signals, that can be connected to port pins or other on-chip logic modules (see **"GPTA®v5 Module Implementation" on Page 22-274** for the TC1797 specific interconnections). Further, several clock input and output signals are provided.

22.3.1 GTPA Units

Each of the General Purpose Timer Arrays GPTA0 and GPTA1 (**Figure 22-2**) is split into Clock Generation Cells (CGC) and a Signal Generation Cells (SGC):

- The **Clock Generation Cells** (see **Page 22-10**) allow a preprocessing of the input signals using filter, timer, capture, compare and enhanced digital PLL cells:
  - The **Filter and Prescaler Cells** (FPC) provide input noise filtering (Immediate Debounce and Delayed Debounce) and may also work as prescalers for the GPTA®v5 module clock and external signals.
  - The **Phase Discrimination Logic** (PDL) may take the outputs of the FPCs to decode phase encoded signals from a position and rotation direction sensor system.
  - The **Duty Cycle Measurement Cells** (DCM) provide signal measurement capabilities (timer plus capture register, single and double capture on rising and falling edges or both) as well as missing pulse detection/reconstruction functions.
  - The **Digital Phase Locked Loop** (Digital PLL) generates a clock with higher clock resolution (harmonic) out of the signal measured by DCM cells. Any arbitrary multiplication factor between 1 and 65535 is supported and may be changed each PLL clock period.
  - The **Clock Distribution Cells** (CDC) provide all LTCs and GTs with a variety of different clock signals. It is equipped with GPTA®v5 module clock prescalers and multiplexers supporting alternate clock sources.

The original signals and all outputs of the preprocessing cells are distributed to the Global Timers and LTCs via the clock bus.

- The **Signal Generation Cells** (see **Page 22-38**) provide a set of timers, capture and compare cells:
  - The two 24-bit **Global Timers** (GT) can be individually configured as free-running counters or as reload counters starting at a programmable value from 0_H to $FFFFFF_H$. Each GT is equipped with a scalable greater-or-equal comparator; the number of bits to be compared is selectable.
  - The **Global Timer Cell** registers (GTC) are 24-bit wide. GTCs may be used as comparators (modifying the logical state of a related output port pin), or as capture cells, storing the current GT0 or GT1 value on rising, falling or both signal edges detected on a related input port pin. Several adjacent GTCs may be connected to logical cells operating on the same pin, allowing complex functions to be implemented.
  - The **Local Timer Cell** registers (LTC) are 16-bit wide. 63 LTCs can be configured to operate in one of four different modes: free-running or resetable counter, capture or compare cell. Adjacent cells can be combined to operate on the same pin, thus generating complex waveforms. One LTC (LTC63) can be used for special compare modes.

22.3.2 Clock Generation Cells

As described in detail in the following sections, the Clock Generation Cells (CGC) provides the following signal pre-processing cells:

- Filter and Prescaler Cell (FPC)
- Phase Discrimination Logic (PDL)
- Duty Cycle Measurement cell (DCM)
- Digital Phase Locked Loop Cell (PLL)
- Clock Distribution Cells (CDC)

The **Filter and Prescaler Cells** (FPC) provide input noise filtering using a debounce filter. FPCs are also able to operate as a prescaler for the GPTA®v5 module clock and external signals. Each FPC can select among different data and clock input signals.

The **Phase Discrimination Logic** (PDL) is able to decode FPC debounce filtered and phase encoded signals coming from a position and rotation direction sensor system. In the PDL, phase encoding can be bypassed.

The **Duty Cycle Measurement Cells** (DCM) provide signal measurement capabilities (timer plus capture register, single and double capture on rising and falling edges or both) as well as missing pulse detection/reconstruction functions.

The **Digital Phase Locked Loop** (PLL) is intended to generate a higher resolution clock out of the values measured by DCM cells. Any arbitrary multiplication factor between 1 and 65535 is supported and may be changed from input clock period to input clock period.

The **Clock Distribution Cells** (CDC) provide all Local and Global Timer Cells with a variety of different clock signals. It is equipped with GPTA®v5 module clock prescalers and multiplexers supporting alternate clock sources.

Figure 22-3 shows how the cells of the CGC are interconnected. The external interface signals of the CGC are:

- GPTA®v5 module clock f_{GPTA}
- GPTA®v5 module input signals (connected to the FPCs)
- Clock bus outputs (generated by the CDC)
- PDL bus outputs
- External PLL clock inputs (fed into CDC)

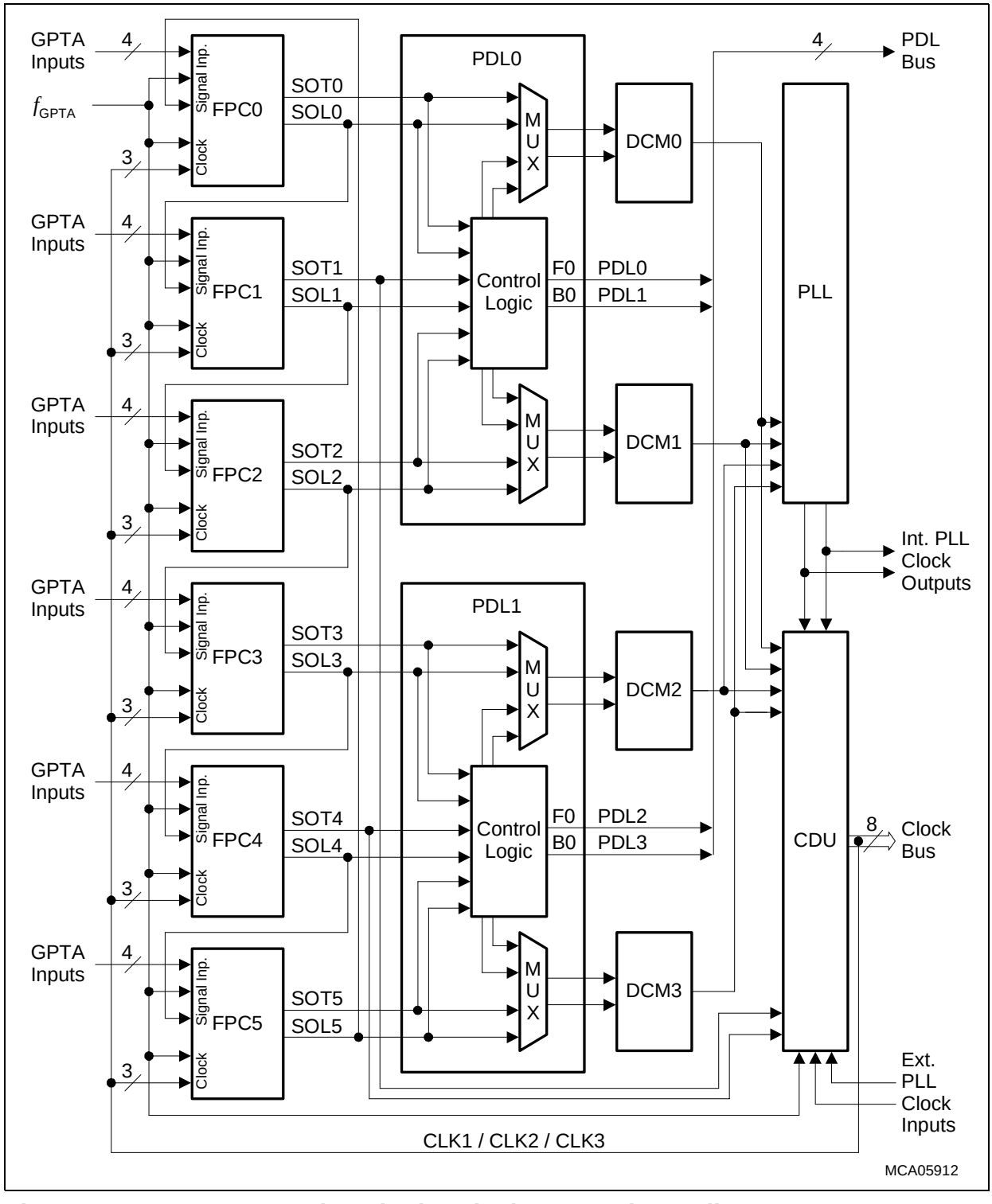


Figure 22-3 Interconnections in the Clock Generation Cells

22.3.2.1 Filter and Prescaler Cell (FPC)

Each GPTA®v5 contains six filter and prescaler cells, FPC0 to FPC5. As shown in **Figure 22-4**, each FPC is equipped with an signal input multiplexer, a clock multiplexer, an edge detection circuitry, a 16-bit timer, a 16-bit compare register, a 16-bit comparator, and a FPC control circuitry (see also **Page 22-126** for the FPC functional algorithm description). The edge detection circuitry detects respective edges for the prescaler modes and detects glitches in all other modes.

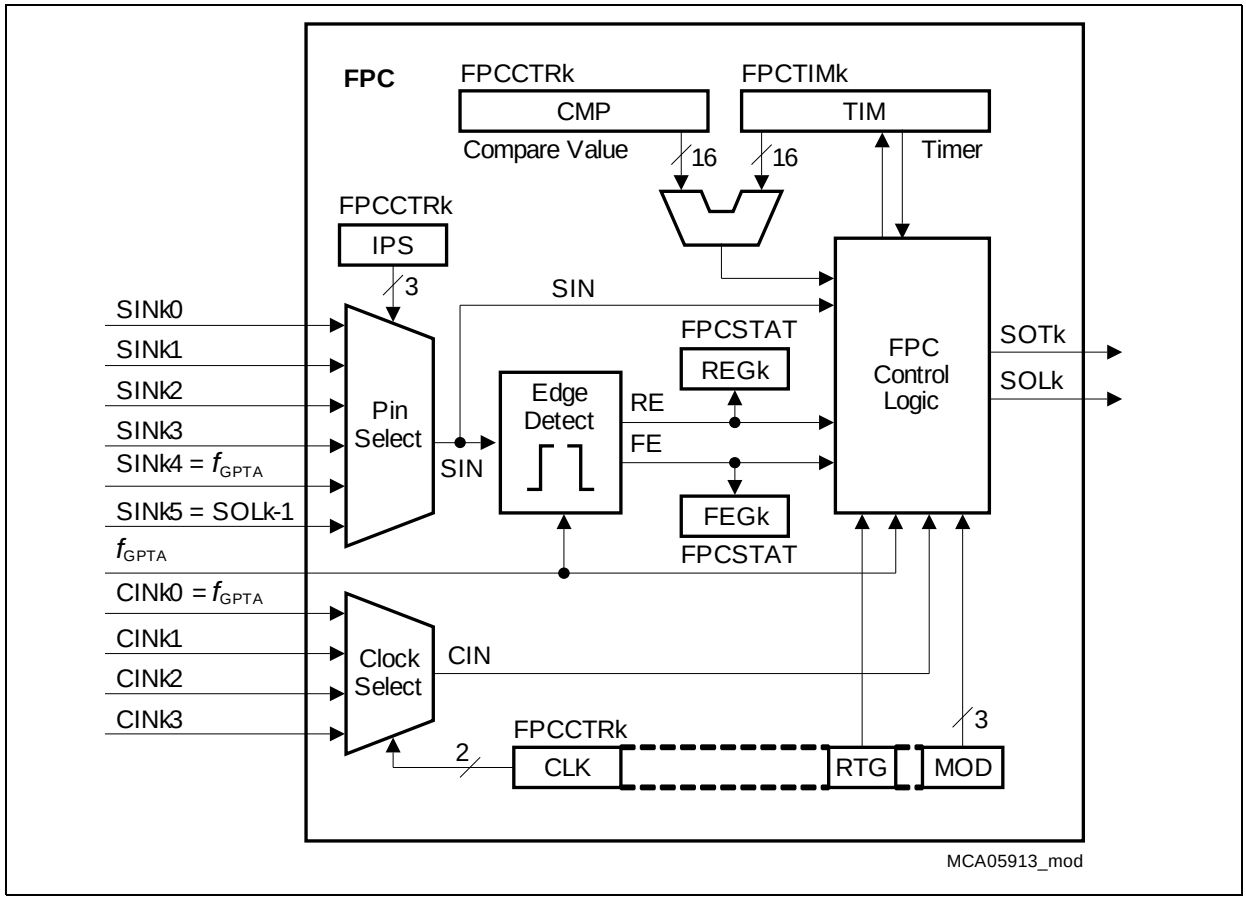


Figure 22-4 Filter and Prescaler Cell Architecture

FPC Registers

The following registers are assigned to the filter and prescaler cells FPCk (k = 0-5):

- FPCSTAT = Filter and Prescaler Cell Status Register (see **Page 22-167**)
- FPCCTRk = Filter and Prescaler Cell Control Register k (see **Page 22-168**)
- FPCTIMk = Filter and Prescaler Cell Timer Register k (see **Page 22-170**)

FPC Operating Modes

Each filter and prescaler cell can be individually configured to operate in one of the following operating modes:

- Delayed Debounce Filter Mode on both edges
- Immediate Debounce Filter Mode on both edges
- Rising edge: Immediate Debounce Filter Mode, falling edge: No filtering
- Rising edge: No filtering, falling edge: Immediate Debounce Filter Mode
- Rising edge: Delayed Debounce Filter Mode, falling edge: Immediate Debounce Filter Mode
- Rising edge: Immediate Debounce Filter Mode, falling edge: Delayed Debounce Filter Mode
- Prescaler Mode (triggered by edge detection circuitry on rising edge)
- Prescaler Mode (triggered by edge detection circuitry on falling edge)

The operation mode is selected by bit field FPCCTRk.MOD (**Page 22-168**).

FPC Input Signals

Bit field FPCCTRk.IPS (see **Page 22-168**) selects one of the following inputs for FPCk:

- Signal input 0 (SINk0)
- Signal input 1(SINk1)
- Signal input 2 (SINk2)
- Signal input 3 (SINk3)
- GPTA®v5 module clock f_{GPTA} (SINk4)
- Preceding FPC level output signal SOLk-1 (SIN05 is connected to SOL5)

When the preceding FPC level output signal is selected as input, two or more FPCs may be concatenated; for example, to combine a delayed debounce filter and an immediate debounce filter.

The maximum FPC input signal frequency must be less than or equal to the sampling rate ($f_{GPTA}/2$). The assignment of GPTA®v5 I/O line and FPC signal inputs SINk is defined in **“FPC Input Line Selection” on Page 22-102**.

FPC Filter Clocks

Bit field FPCCTRk.CLK (see **Page 22-169**) selects one of four filter clocks for FPCk:

- Clock input line 0 (CINk0) = GPTA®v5 module clock f_{GPTA}
- Clock input line 1 (CINk1) = local PLL clock,
- Clock input line 2 (CINk2) = (prescaled) GPTA®v5 module clock f_{GPTA} or PLL clock from other unit or DCM 3 clock
- Clock input line 3 (CINk3) = DCM 2 clock or PLL clock of other unit or uncompensated PLL clock or uncompensated PLL clock of other unit

When using a PLL clock for the FPC, no software is needed to adapt the FPC filter to changing speed for angle-based input signals. The standard PLL clock can be either the

compensated or uncompensated PLL clock or can be the PLL clock of the other GPTA®v5 units. The uncompensated PLL clock is useful in applications in which bursts (due to acceleration) might disturb the filter function.

With a prescaled GPTA®v5 module clock, very long time filter time periods can be achieved.

Note: All filter operation are always synchronously to f_{GPTA}. Therefore the further signal analysis (e.g. glitch detection) is not processed on the rising edge of the selected filter clock CIN, but on the next rising edge of f_{GPTA} following the rising edge of selected CIN (gated clock principle). Therefore CIN clock rates above f_{GPTA} will lead to non deterministic behavior.

Output Signal Splitting

Two output lines are provided by each FPC cell as follows:

- An trigger output signal SOTk, reporting a falling or rising signal edge on the FPC input by a single f_{GPTA} clock pulse,
- A level output signal SOLk, indicating the direction of the detected signal transition.

This signal-splitting scheme (pair of trigger and level output) provides subsequent PDL and DCM cells with the information about an input signal transition in the same f_{GPTA} clock cycle. This feature avoids cascading a one clock delay per edge detection circuitry implemented at the input of each subsequent cell. **Figure 22-5** shows the FPC output signal splitting scheme.

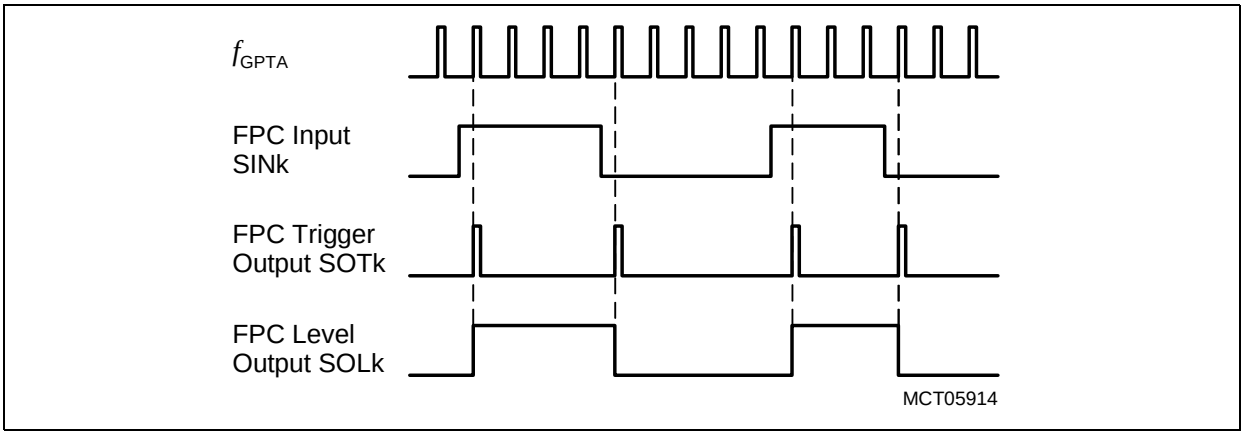


Figure 22-5 FPC Output Splitting into Trigger and Level Information

Delayed Debounce Filter Mode

In Delayed Debounce Filter Mode, the signal input SIN is filtered from all signal transitions and glitches with a width smaller than the selected clock period length multiplied by the compare register value.

The input signal SIN (sampled with f_{GPTA}) is analyzed at the selected filter clock rate of CIN. If the state of the input sample differs from the current output signal value, the 16-bit timer is incremented by one. When the timer register FPCTIMk is not in its idle state (0000_H) **and** the state of the input sample matches the current output signal value, the 16-bit timer is decremented by one (see **Figure 22-6**); if bit FPCCTRk.RTG is set, the timer will be set to idle state again (see **Figure 22-7**). A rising or falling edge, occurring on the signal input line SIN when the timer is greater than zero but less than the compare value, sets the corresponding glitch flag FPCSTAT.REG (on rising edge glitch) or FPCSTAT.FEG (on falling edge glitch). When the timer matches the 16-bit compare value stored in FPCCTRk.CMP (timer threshold), the level output signal line SOLk is inverted, a GPTA®v5 module clock pulse is generated at the trigger output signal SOTk, and the timer is reset to 0000_H. The rising/falling edge glitch flags must be reset by software.

The filter is by-passed if the compare value FPCCTRk.CMP is programmed to zero (0000_H). In this case, the input signal is directly copied to the output signal.

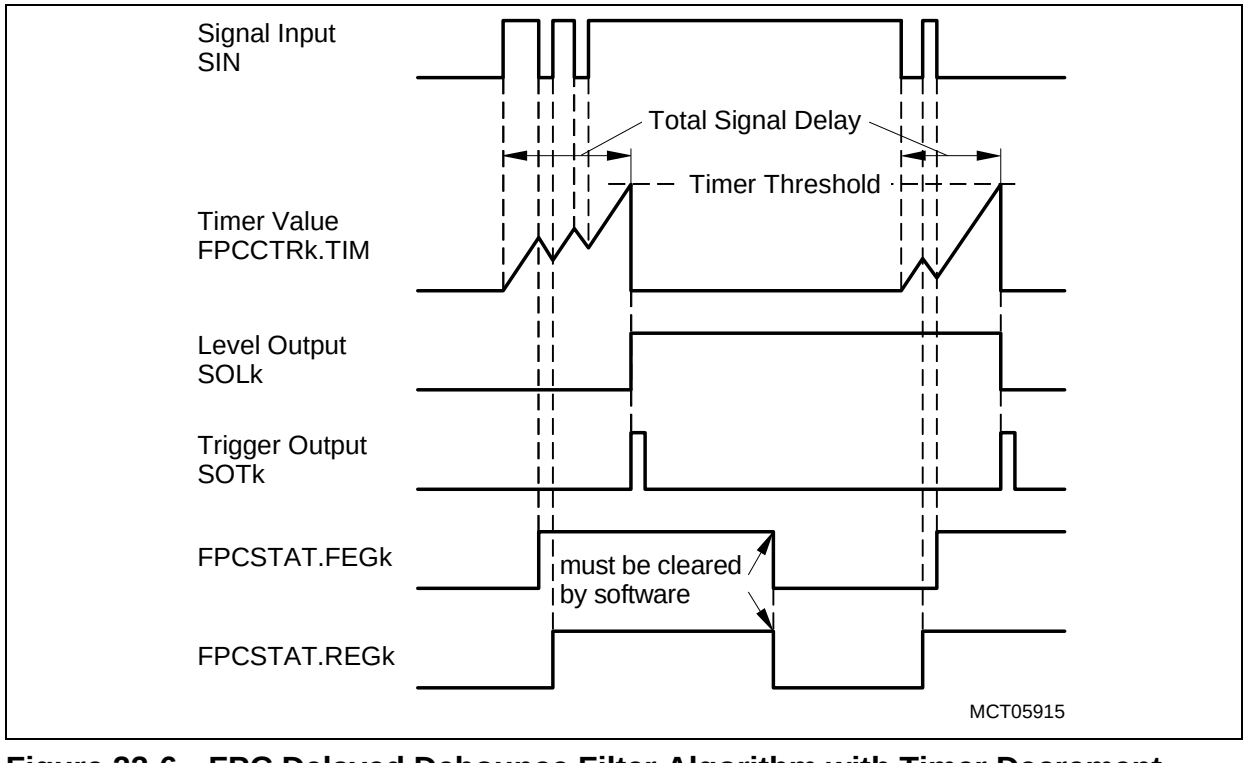


Figure 22-6 FPC Delayed Debounce Filter Algorithm with Timer Decrement

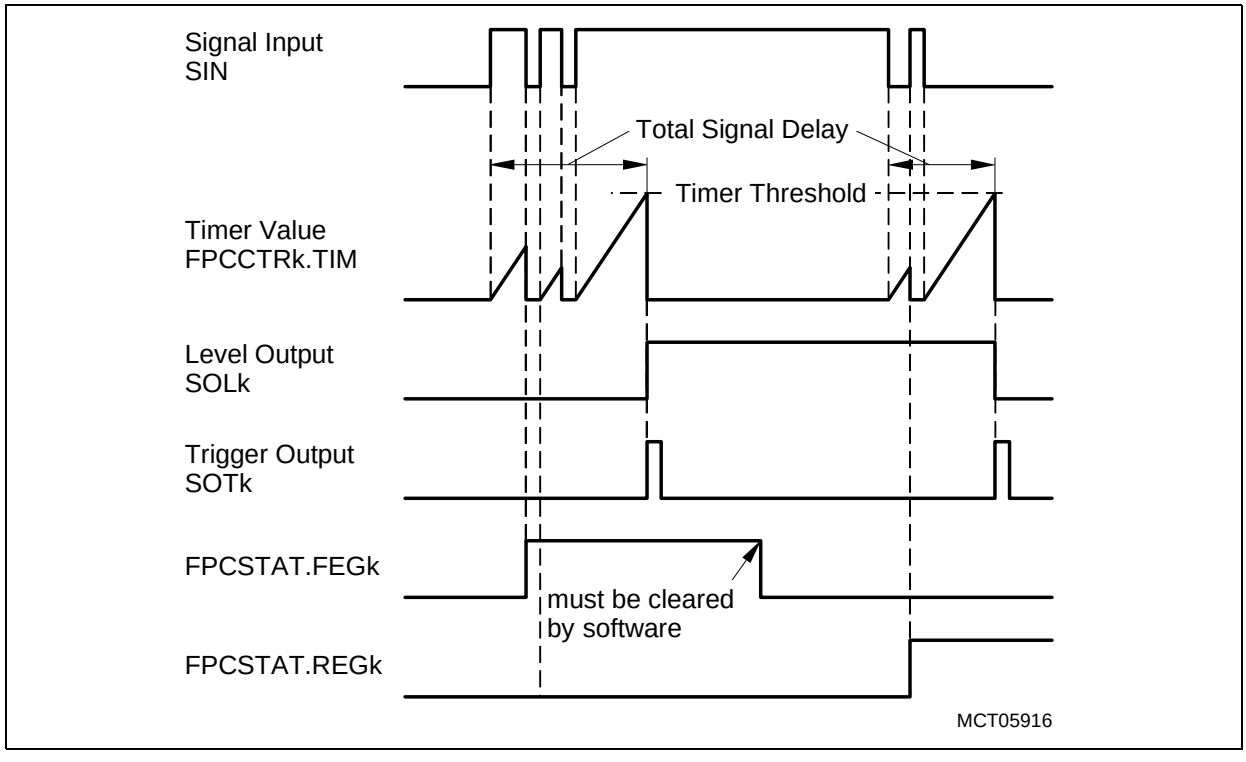


Figure 22-7 FPC Delayed Debounce Filter Algorithm with Timer Reset

The total signal delay from input to output depends on the programmed compare register value, the number of high-frequency pulses (glitches) during the filter operating time, and the timer behavior in case of a glitch (decrement or reset).

The FPC Delayed Debounce Filter Mode is selected by:

- FPCCTRk.MOD = 000_B

Immediate Debounce Filter Mode

In Immediate Debounce Filter Mode, the input signal is filtered from signal transitions and glitches arriving a programmable time after an input signal edge detection (see **Figure 22-8**).

The input signal SIN is sampled with f_{GPTA} and the input signal SIN edge detection is also performed with f_{GPTA}. The further analysis (e.g. filter timer increment, glitch detection) is done at the selected filter clock rate of CIN.

As long as the timer is reset, the FPC control circuitry copies the sampled input value directly to the level output signal line SOLk. When a rising or falling edge occurs on the signal input line SIN and the 16-bit compare value FPCCTRk.CMP is not zero, the timer is enabled to be incremented by the selected clock and the copy mechanism is disabled. When the timer value FPCTIMk.TIM matches the compare value FPCCTRk.CMP, the timer is reset and the copy mechanism is enabled again. A rising or falling edge, occurring on SIN while the timer is greater than zero but less than the compare value, sets the corresponding glitch flag FPCSTAT.REG (on rising edge glitch) or FPCSTAT.FEG (on falling edge glitch). The rising/falling edge glitch flags must be reset by software.

The filter is by-passed if the compare value FPCCTRk.CMP is programmed to zero (0000_H). In this case, the input signal is directly copied to the output signal without any disable periods.

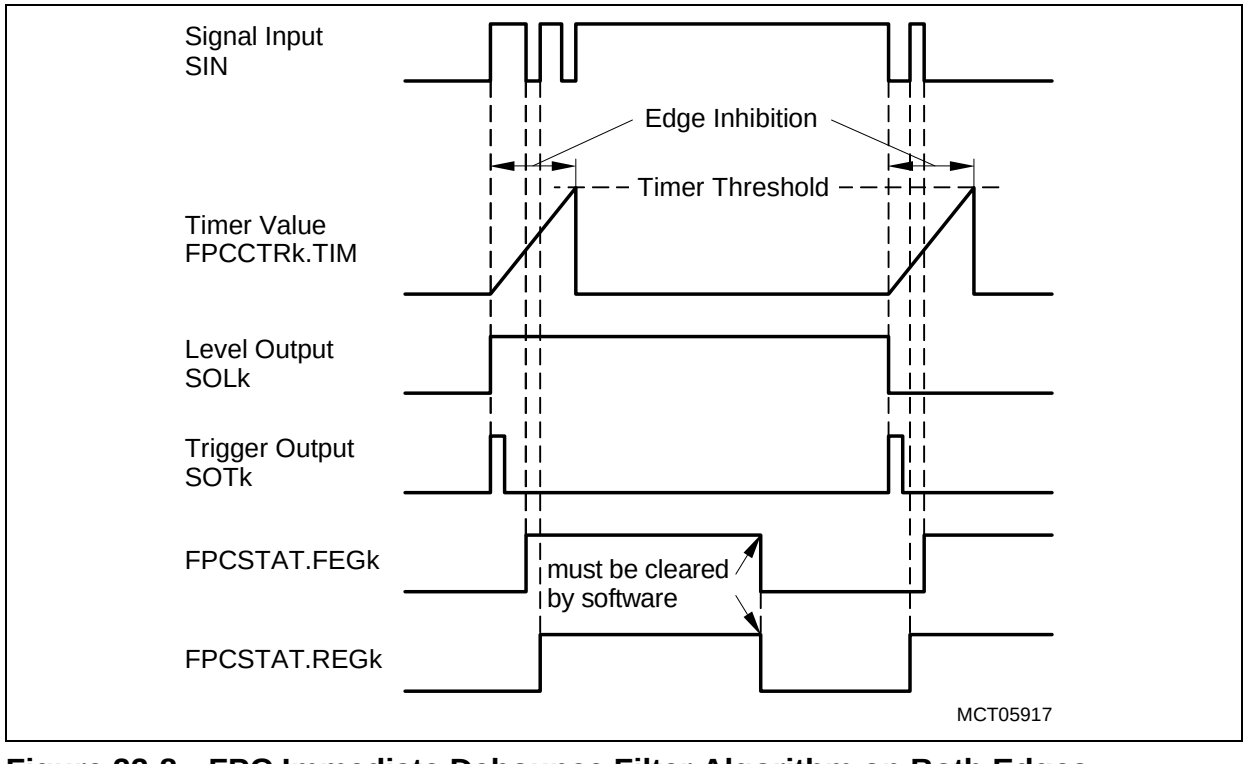


Figure 22-8 FPC Immediate Debounce Filter Algorithm on Both Edges

Note: During the last clock cycle of edge inhibition time (where timer value is equal to the compare value) an input signal glitch will be filtered but the corresponding glitch status flag in register FPCSTAT is not set.

The Immediate Debounce Filter can be enabled only for one edge, either rising or falling. In this case, the signal output follows the signal input value immediately after the timer threshold of the filtered edge is reached, without re-starting the timer (see **Figure 22-9**).

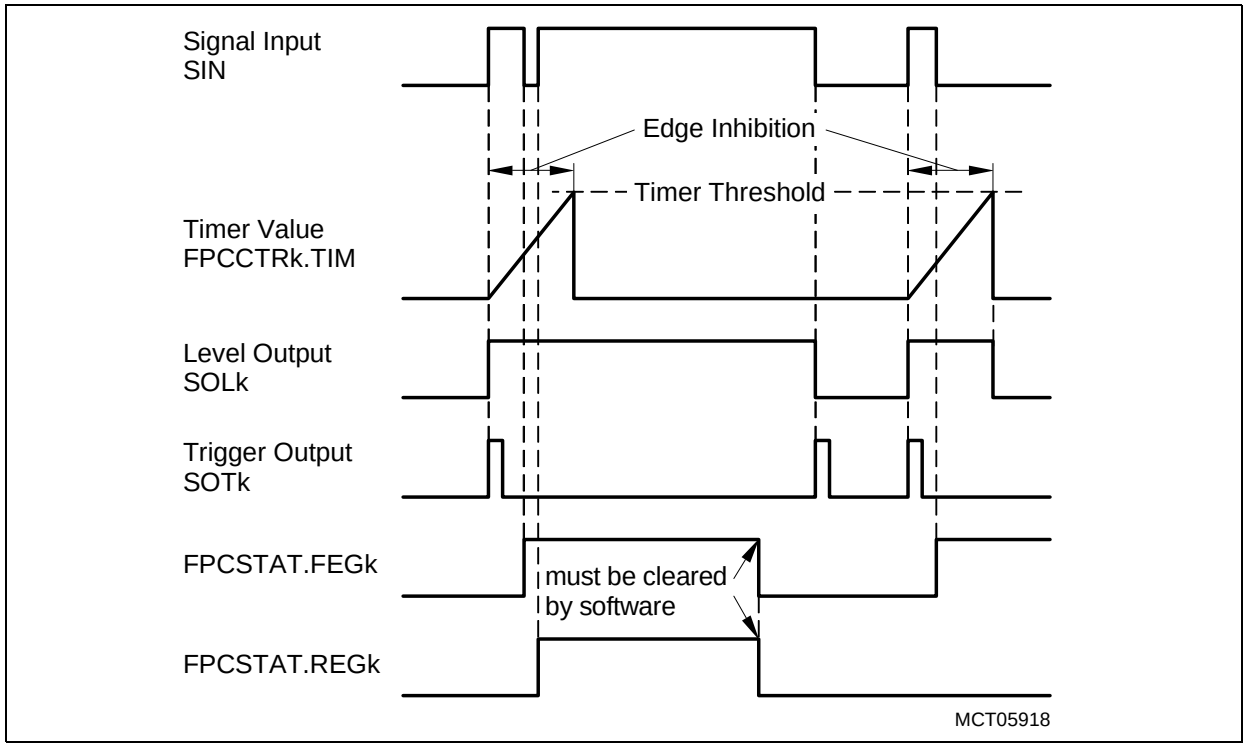


Figure 22-9 FPC Immediate Debounce Filter Algorithm on Rising Edge only

The FPC Immediate Debounce Filter Modes are selected by:

- FPCCTRk.MOD = 001_B: Immediate Debounce Filter Mode on both edges
- FPCCTRk.MOD = 010_B: Immediate Debounce Filter Mode on rising edge only, no filtering on falling edge.
- FPCCTRk.MOD = 011_B: Immediate Debounce Filter Mode on falling edge only, no filtering on rising edge.

Mixed Filter Modes

In the Mixed Filter Modes, one edge of a signal is filtered in the Delayed Debounce Mode, and the other edge is filtered in the Immediate Debounce Mode. The Debounce Mode is switched when the timer threshold is reached. Note that both filter modes use the same timer threshold in this case (see **Figure 22-10**, demonstrating Delayed Debounce Mode with Timer Decrement on Rising Edge and Immediate Debounce of on Falling Edge).

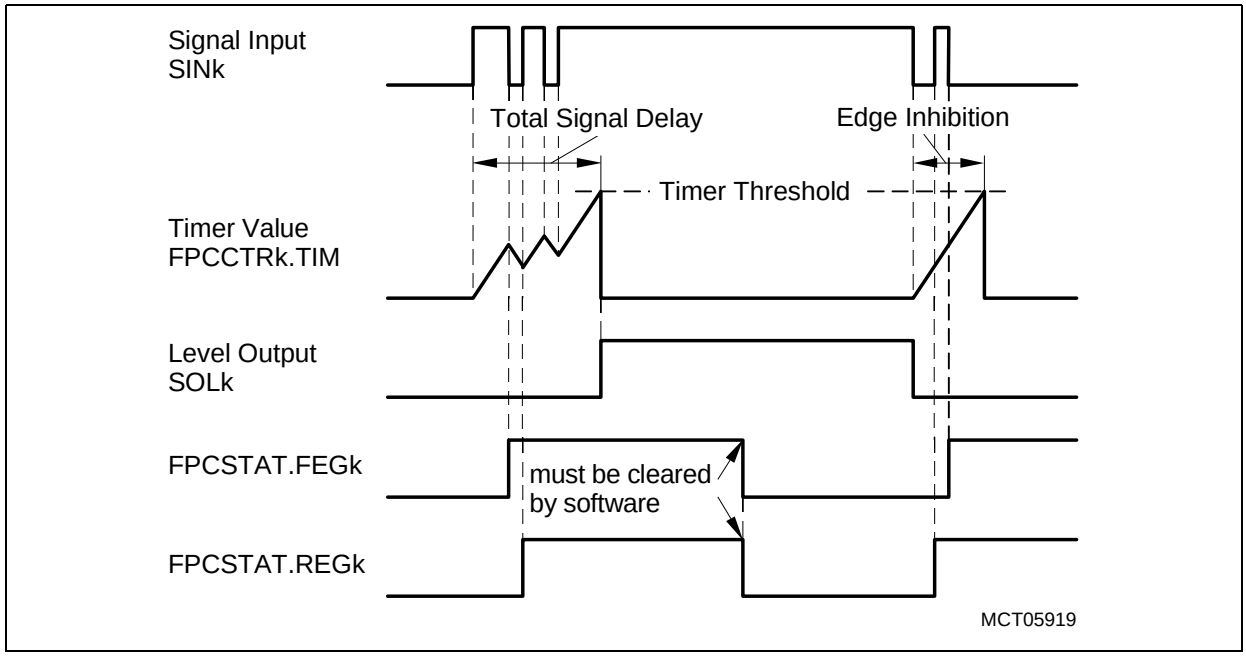


Figure 22-10 FPC Mixed Filter Algorithm

The FPC Mixed Filter Modes are selected by:

- FPCCTRk.MOD = 100_B: Delayed Debounce Filter Mode on rising edge
  Immediate Debounce Filter Mode on falling edge
- FPCCTRk.MOD = 101_B: Immediate Debounce Filter Mode on rising edge
  Delayed Debounce Filter Mode on falling edge

Prescaler Mode

In Prescaler Mode, the input signal is sampled and analyzed with f_{GPTA}. The FPC control circuitry counts each rising (or falling) edge of the input signal. When the timer value matches the compare value:

- one GPTA®v5 module clock pulse is generated at the trigger output signal SOTk and level output signal SOLk
- the timer FPCTIMk.TIM is reset to 0000_H

Figure 22-11 shows a divide-by-6 operation using the FPC in Prescaler Mode with trigger on rising edge selected.

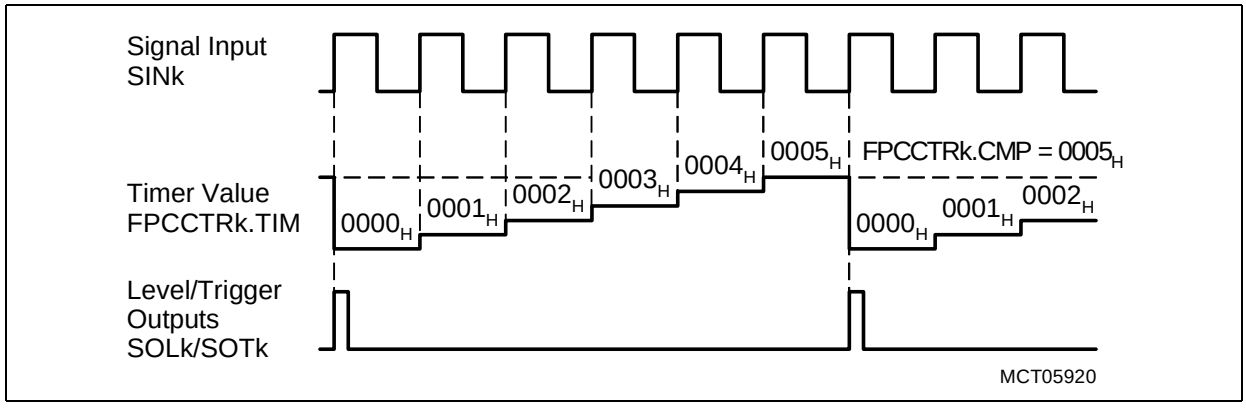


Figure 22-11 FPC Prescaler Mode

For a divide-by-n operation, the compare value FPCCTRk.CMP must be set to n - 1.

The FPC Prescaler Modes are selected by:

- FPCCTRk.MOD = 110_B: Prescaler Mode triggered by edge detection circuitry on rising edge
- FPCCTRk.MOD = 111_B: Prescaler Mode triggered by edge detection circuitry on falling edge

22.3.2.2 Phase Discrimination Logic (PDL)

The GPTA®v5 provides two Phase Discrimination Logic cells (PDL0, PDL1) driven by two signal lines coming from an FPC cell (for description, see **Page 22-14**):

- An event input signal
- A level input signal

Both Phase Discrimination Logic cells are controlled by the Phase Discrimination Logic Control Register PDLCTR (see **Page 22-171**).

Each PDL is equipped with an edge detection circuitry, a phase detection circuitry, a PDL control circuitry, and an output multiplexer. Six output lines are provided by each PDL Cell:

- A forward output signal (F0, F1) is driven by one f_{GPTA} clock pulse if an input signal edge is recognized as forward rotation. These signals can be connected to any Local Timer Cell via the PDL bus.
- A backward output signal (B0, B1) is driven by one f_{GPTA} clock pulse if an input signal edge is recognized as backward rotation. This signal can be connected to any Local Timer Cell via the PDL bus.
- Two pairs of output signals, carrying the bypassed input level and event information from the driving FPC cells or the angular velocity and error information provided by the PDL function. These output lines are directly connected to the adjacent Duty Cycle Measurement Cells, DCM0/DCM1(for PDL0) and DCM2/DCM3 (for PDL1).

The PDL processes the output signal of a 2-sensor or 3-sensor positioning system. With bit PDLCTR.TSEx = 1, a 3-sensor system execution is selected providing the DCM1 and/or DCM3 cell with information concerning erroneous states in the signal input. When PDLCTR.TSEx = 0, a 2-sensor system is selected and DCM1 and/or DCM3 are supplied with the input event and level information from the driving FPC2 and/or FPC5.

The rotation direction, monitored by the connected sensors, is automatically derived from the sequence in which the input signals change. Each edge detected on an input signal line generates a pulse on the F0, F1 forward output lines or on the B0, B1 backward output lines. Input jitter, which might occur if a sensor rests near to one of its switching points, is compensated.

If bit PDLCTR.MUXx = 1, the trigger output signal to DCM0/DCM2 (angular velocity information) is driven by a boolean ‘OR’ operation of the corresponding forward trigger and backward trigger signal while the level output signal at DCM0/DCM2 is at fixed high level. In this case, every pulse at F0/B0 and F1/B1 generates a rising edge at the DCM0/DCM trigger signal.

If bit PDLCTR.MUXx = 0, the associated DCM0/DCM2 signals are directly connected with the input event and level signals from the driving FPC0/FPC3.

To calculate the sensor’s current position, the associated LTCs should be clocked with the PDL forward and backward output pulses. A software operation, subtracting the backward counter contents from the forward counter contents, provides the absolute

position. Dynamic information (speed, acceleration, deceleration) may be obtained by analyzing the angular velocity signal periods with the associated DCM cell.

The maximum input frequency is $f_{GPTA}/4$ for a 2-sensor positioning system and $f_{GPTA}/6$ for a 3-sensor positioning system. To ensure that a transition of any input signal is correctly recognized, its level should be held high or low for at least twof_{GPTA} cycles before it changes (three f_{GPTA} cycles for a 3-sensor positioning system).

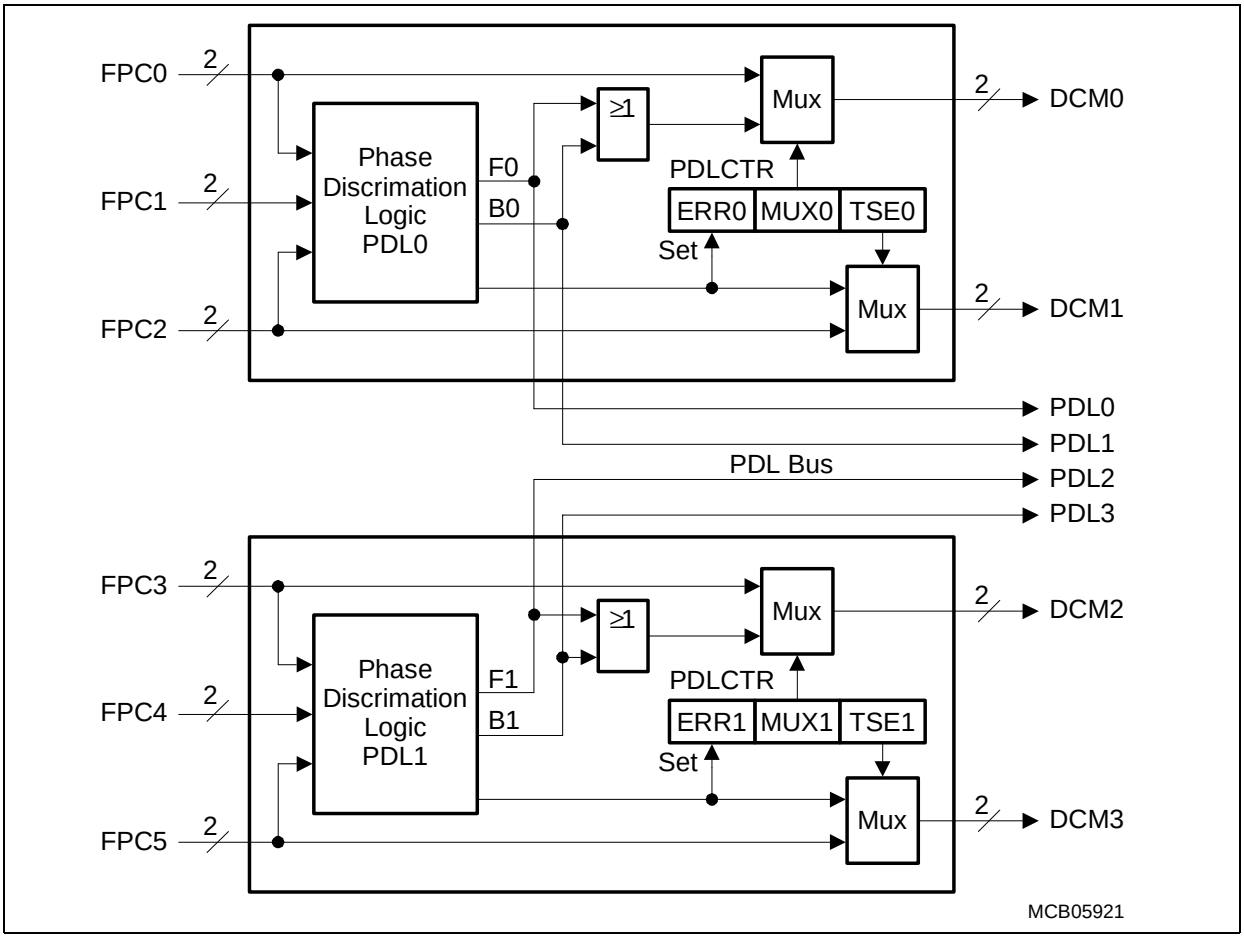


Figure 22-12 Block Diagram of Phase Discrimination Logic Cells

Positioning System With Two Sensors

The 2-sensor Mode is enabled when bit PDLCTR.TSEx is reset. The sensors are mounted at a 90° angle to each other (see **Figure 22-13**). The third sensor input of the PDL cell is internally disabled and DCM1/DCM3 cell inputs are driven by fed-through FPC2/FPC5 output lines.

This configuration can measure an absolute position with a resolution of 90°. No error conditions can be detected.

!	Means not
Re	Means rising edge
Fe	Means falling edge
Forward	ReS1*!S2 + S1*ReS2 + FeS1*S2 + !S1*FeS2
Backward	ReS1*S2 + !S1*ReS2 + FeS1*!S2 + S1*FeS2
Position	Forward_Counter - Backward_Counter

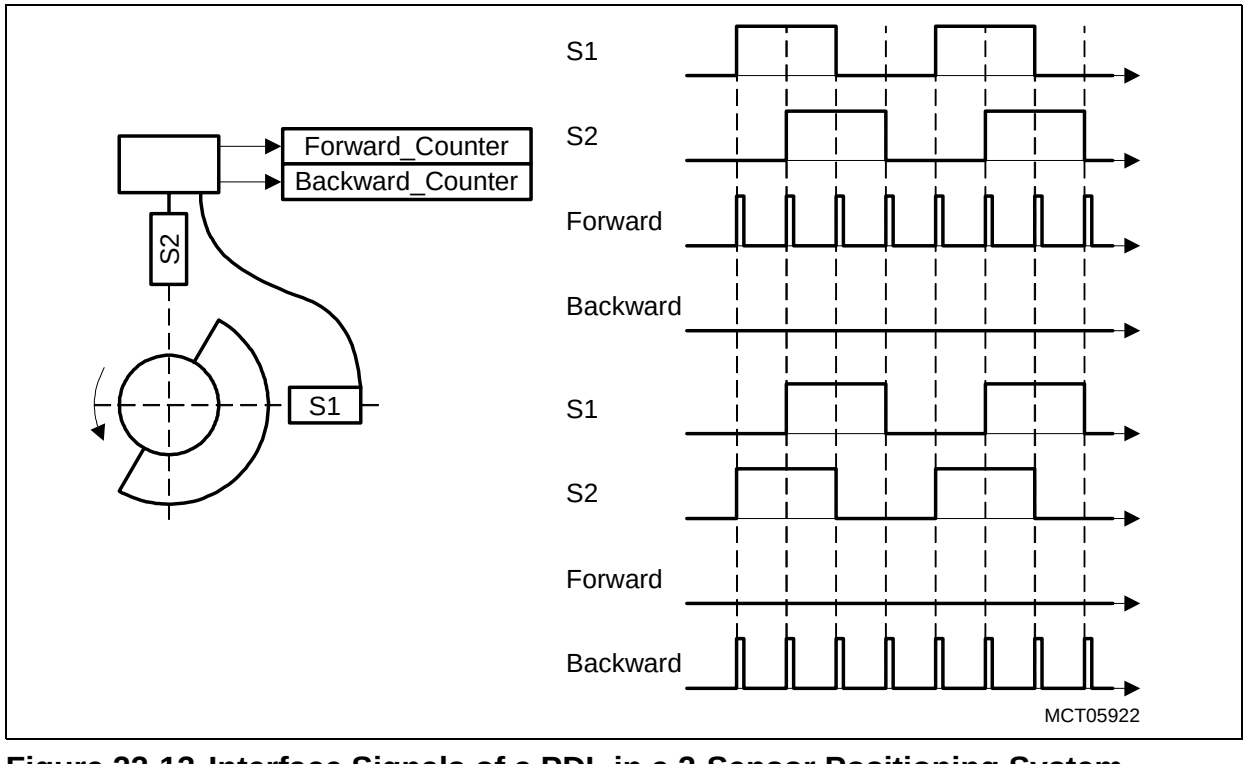


Figure 22-13 Interface Signals of a PDL in a 2-Sensor Positioning System

Figure 22-14 illustrates how the output signals of a 2-sensor system superimposed with noise are processed by the PDL cell. Jitter pulses are completely compensated if they do not occur on both signal lines simultaneously.

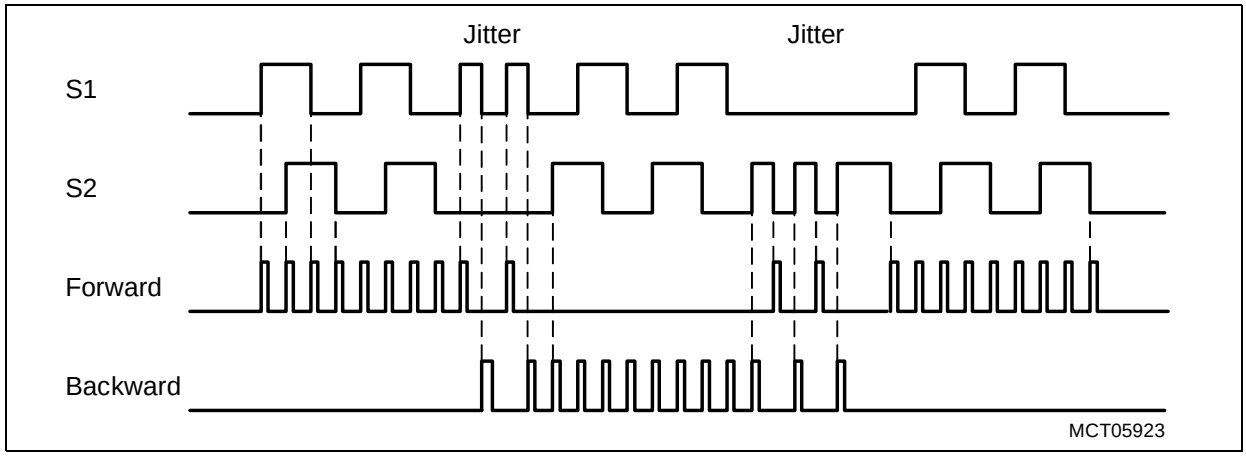


Figure 22-14 Compensation of Input Jitter

Positioning System with Three Sensors

The 3-sensor Mode is enabled when bit PDLCTR.TSEx is set to 1. The sensors are mounted at an 120° angle to each other (see **Figure 22-15**). This configuration can measure an absolute position with a resolution of 60°.

Input signal combinations that are not allowed in a properly-working positioning system (all inputs low or all inputs high) cause the following to occur:

- An error signal is generated, driving the Duty Cycle Measurement cells DCM1 and/or DCM3,
- The error flag PDLCTR.ERRx is set,
- No forward or backward pulses are generated.

When the error disappears, the error signal will be cleared. The error flag PDLCTR.ERRx must be reset by software.

!	Means not
Re	Means rising edge
Fe	Means falling edge
Forward	ReS1*!S2*S3 + FeS3*S1*!S2 + ReS2*S1*!S3 + FeS1*S2*!S3 + ReS3*!S1*S2 + FeS2*!S1*S3
Backward	ReS1*S2*!S3 + FeS3*!S1*S2 + ReS2*!S1*S3 + FeS1*!S2*S3 + ReS3*S1*!S2 + FeS2*S1*!S3
Error	The input signal states S1*S2*S3 and !S1*!S2*!S3 are not allowed
Position	Forward_Counter - Backward_Counter

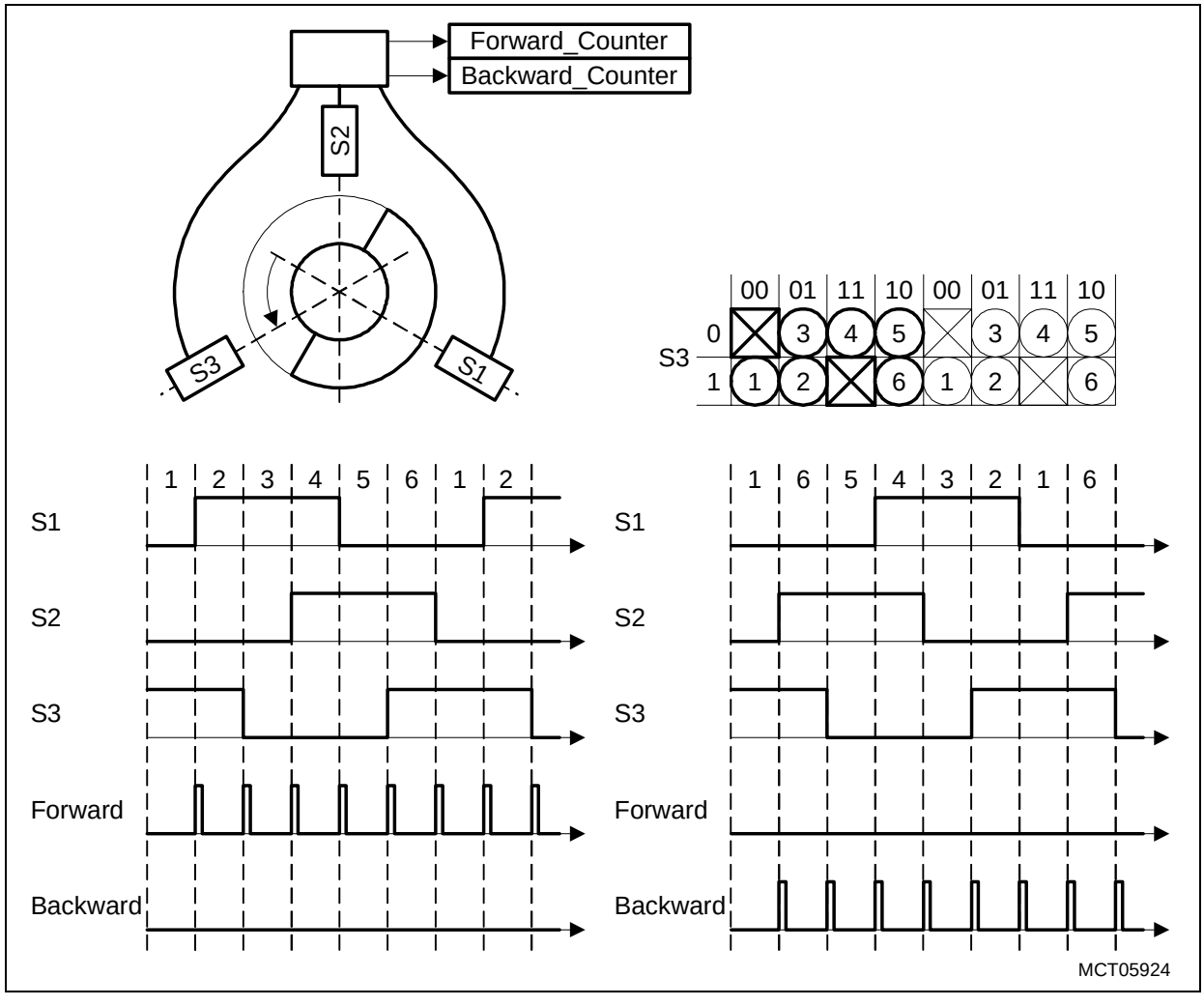


Figure 22-15 Interface Signals of a PDL in a 3-Sensor Positioning System

Jitter pulses are completely compensated as illustrated in **Figure 22-14**.

22.3.2.3 Duty Cycle Measurement Cell (DCM)

The GPTA®v5 contains four DCM cells (DCM0 to DCM3). The input signal to be analyzed is delivered as a 2-line signal input (see **Figure 22-5** for the event/level input signal splitting scheme). It is build by:

- An event input, and
- A signal level input.

Each DCM cell has four outputs:

- An event output line,
- An interrupt output that can become active at a signal input rising edge,
- An interrupt output that can become active at a signal input falling edge,
- An interrupt output that can become active at a compare event.

Each DCM cell is equipped with a 24-bit timer, a 24-bit capture register, a 24-bit capture/compare register, a 24-bit comparator and a DCM control circuitry (**Figure 22-16**).

The following registers are assigned to the DCM cells:

- DCMCTRk = Duty Cycle Measurement Control Register k (see **Page 22-173**)
- DCMTIMk = Duty Cycle Measurement Timer Register k (see **Page 22-175**)
- DCMCAVk = Duty Cycle Measurement Capture Register k (see **Page 22-175**)
- DCMCOVk = Duty Cycle Measurement Capture/Compare Register k (also referred as “CAPCOM”, see **Page 22-176**)
- SRSC0 = Service Request State Clear Register 0 (see **Page 22-223**)
- SRSS0 = Service Request State Set Register 0 (see **Page 22-225**)

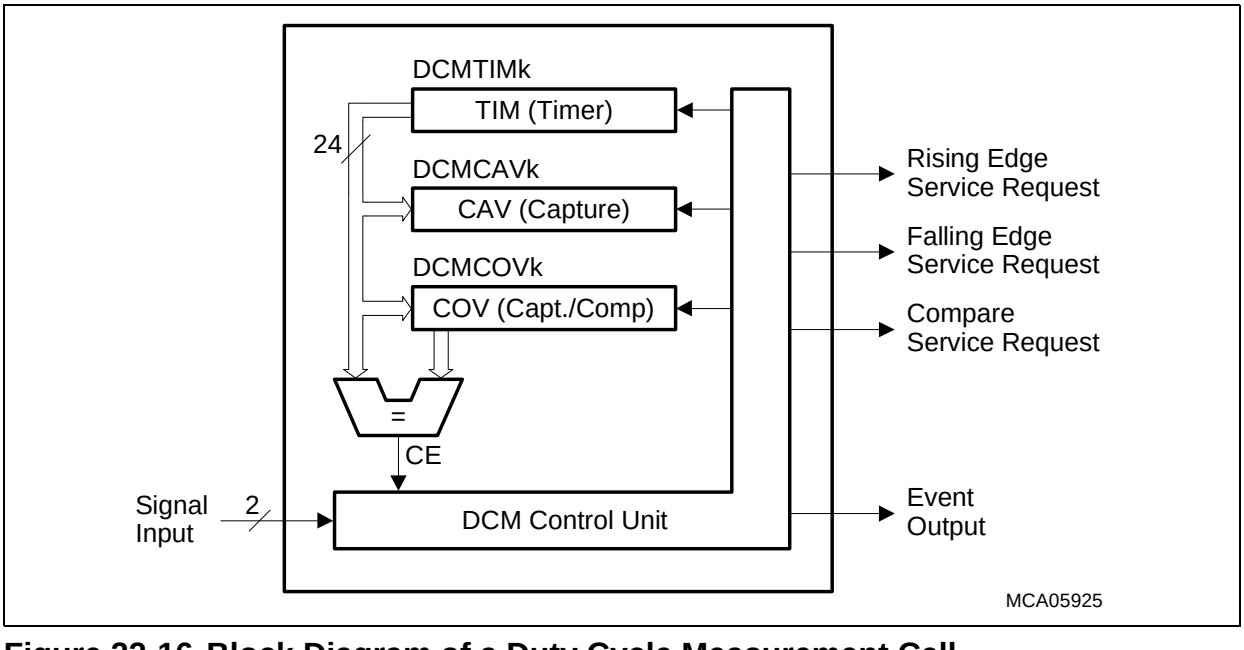


Figure 22-16 Block Diagram of a Duty Cycle Measurement Cell

The DCM cell inputs are connected to the PDL outputs. Depending on the configuration of the associated PDL cell, the DCM cells can also be driven by a FPC directly (as shown in **Figure 22-12**):

- DCM0 is driven by FPC0 or PDL0 angular velocity signal,
- DCM1 is driven by FPC2 or PDL0 error signal,
- DCM2 is driven by FPC3 or PDL1 angular velocity signal,
- DCM3 is driven by FPC5 or PDL1 error signal.

When the driving FPCs and PDL cells are programmed in feed-through mode, an external port pin signal as selected by the FPC input multiplexer can be directly processed by a DCM cell.

The duty cycle of the DCM cell signal input can be determined by measuring its period length and the width of its low or high state. For this purpose, several operations can be started on an signal input edge:

- **Reset Timer**
  The local timer can be reset on rising, falling, or both edges of the signal input line as selected via control bits DCMCTRk.RZE (for rising edge) and DCMCTRk.FZE (for falling edge). After a reset timer event, the timer is continuously incremented by the GPTA®v5 module clock f_{GPTA} until the next reset condition occurs. If no reset timer event is enabled, the timer operates in Free-Running Timer Mode, repeatedly counting from its lower limit (000000_H) to its upper limit ($FFFFFF_H$).
- **Capture**
  The current timer value is stored in the capture register DCMCAV on the rising edge (DCMCTR.RCA = 1) or falling edge (DCMCTRk.RCA = 0) of the signal input line.
  The current timer value is stored in the capture/compare register DCMCOV on the opposite signal edge as selected by DCMCTRk.RCA and if enabled by bit DCMCTRk.OCA = 1. With DCMCTRk.OCA = 0 the capture/compare register DCMCOV is not affected.
- **Edge Service Request and Interrupt Request**
  On a rising input signal edge of the DCMk cell (k = 0-3) the service request flag SRS0.DCM0kR is set. Additionally, a service request signal is triggered if bit DCMCTRk.RRE = 1. A falling input signal edge sets the service request flag SRS0.DCM0kF. An interrupt request generation on this edge is triggered if bit DCMCTRk.FRE = 1. Both edges of the signal input line initiate an interrupt request when both bits, DCMCTRk.FRE and DCMCTRk.RRE, are set. The interrupt on signal input edges is disabled if both bits are cleared.
- **Hardware Generated Output Pulse**
  A single f_{GPTA} clock pulse is generated on the DCM output line if enabled by control register bit DCMCTRk.RCK (rising edge at signal line) and/or DCMCTRk.FCK (falling edge at signal line) and an appropriate edge is detected at the input.
  The 0% or 100% duty cycle exception (no edge or only one edge detected) can be handled by a **limit checking** option. The expected input signal's maximum period length (measured in f_{GPTA} clock ticks) can be loaded into the capture/compare

register DCMCOV that is continuously compared with the timer value. When the timer is incremented up to the limit stored in capture/compare register, the service request flag SRS0.DCM0xC is set. If the compare service request is enabled (control register bit DCMCTRk.CRE = 1), an interrupt request is generated.

- **Software Generated Output Pulse**
  If the **software** intends to compensate an input pulse backlog, bit DCMCTRk.QCK should be set to 1. This immediately triggers a single **clock pulse generation** on the DCM output signal line.

DCM Interrupt Control

Each DCM cell is able to generate three service request output signals. The service request outputs of a DCMk cell are controlled as shown in **Figure 22-17**. When a service request condition occurs, the corresponding service request flag is always set. The service request output is activated only if it is enabled by the corresponding enable bit. Further details on service request and interrupt handling are provided in section **"Interrupt Sharing Block (IS)" on Page 22-123**.

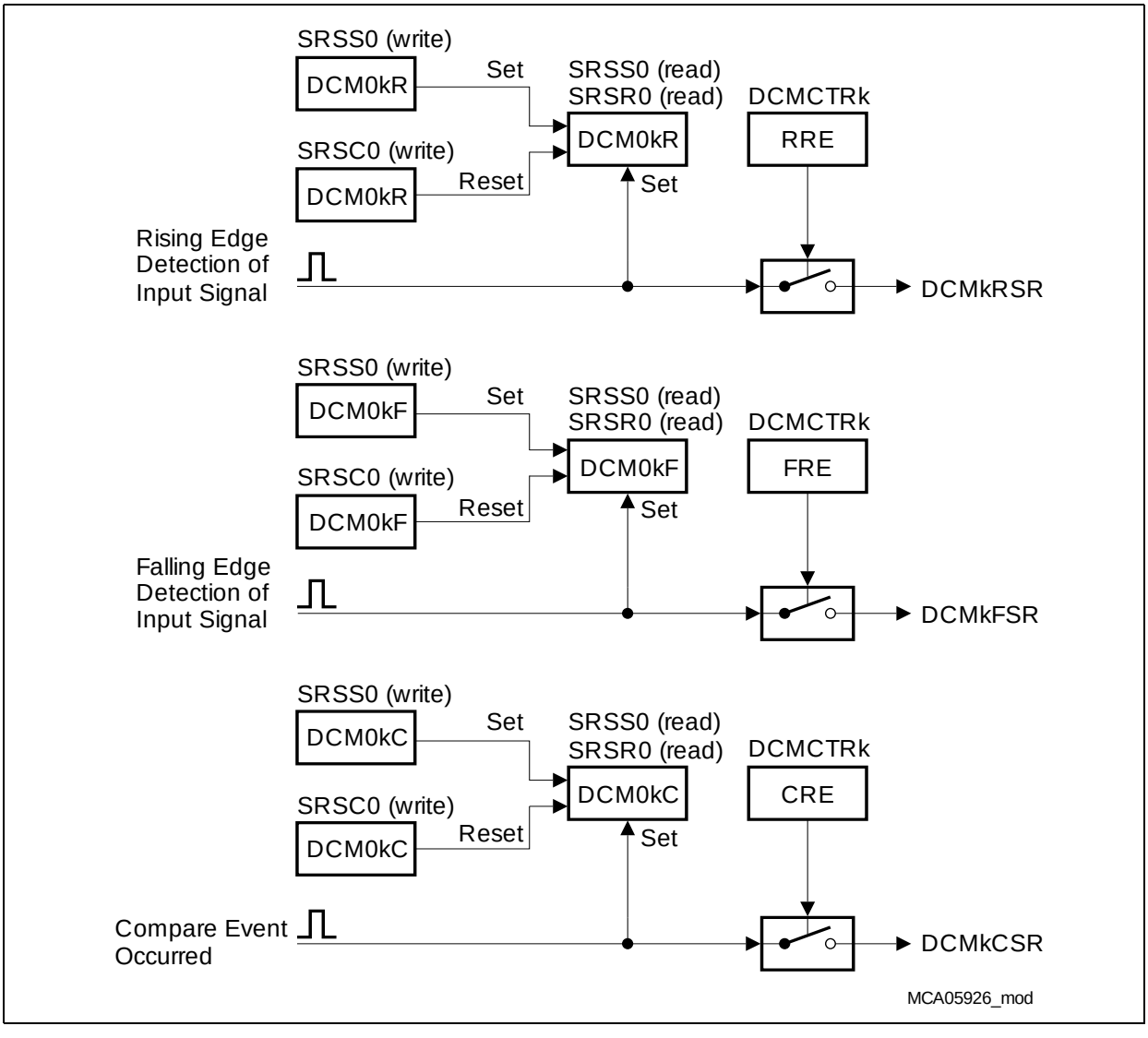


Figure 22-17 DCMk Service Request Generation

22.3.2.4 Digital Phase Locked Loop Cell (PLL)

The GPTA®v5 provides a digital Phase Locked Loop cell (PLL) with a frequency multiplier function. An input signal edge is used as a trigger to generate a programmable number of GPTA®v5 module clocks f_{GPTA} on the output signal line. The four signal output lines of the DCM cells can be used as PLL trigger input. The PLL control circuitry distributes the desired number of GPTA®v5 clocks in regular time intervals over the input signal period length. The PLL can automatically follow an acceleration or deceleration of the input signal. Alternatively, an external software routine may handle the input signal's period length variation.

The PLL includes a 4-channel input multiplexer, a 16-bit timer, a 16-bit step register, a 24-bit reload register, a 24-bit adder, a 24-bit multiplexer, a 25-bit delta register extended by one sign bit and a PLL control circuitry (see **Figure 22-18**).

The following registers are assigned to the Phase Locked Loop cell:

- PLLCTR = Phase Locked Loop Control Register (see **Page 22-177**)
- PLLMTI = Phase Locked Loop Microtick Register (see **Page 22-178**)
- PLLCNT = Phase Locked Loop Counter Register (see **Page 22-179**)
- PLLSTP = Phase Locked Loop Step Register (see **Page 22-179**)
- PLLREV = Phase Locked Loop Reload Register (see **Page 22-180**)
- PLLDTR = Phase Locked Loop Delta Register (see **Page 22-181**)
- SRSC0 = Service Request State Clear Register 0 (see **Page 22-223**)
- SRSS0 = Service Request State Set Register 0 (see **Page 22-225**)

Three output signals are available on the PLL cell:

- PLL signal output line
- Uncompensated PLL signal output line
- Service request line

The desired input signal is selected by programming bit field PLLCTR.MUX. The number of output pulses to be generated within one input signal period must be stored in the microtick register PLLMTI and (coded in 2-complement data format) in the step register PLLSTP. The PLLREV reload register must be programmed with a reload value. This reload value is calculated by subtracting the number of output pulses to be generated within one input signal period from the input signal's period length (measured in number of f_{GPTA} clocks). An automatic compensation of an input signal acceleration or deceleration is enabled by setting bit PLLCTR.AEN to 1 (Automatic End Mode). After disabling the Automatic End Mode, the PLL continuously generates output pulses without synchronization to an input signal edge.

When the counter for the number of remaining output signal pulses PLLCNT decrements to zero, the PLL service request flag is set. Additionally, a service request signal PLLSR will be generated if the control register bit PLLCTR.REN is set.

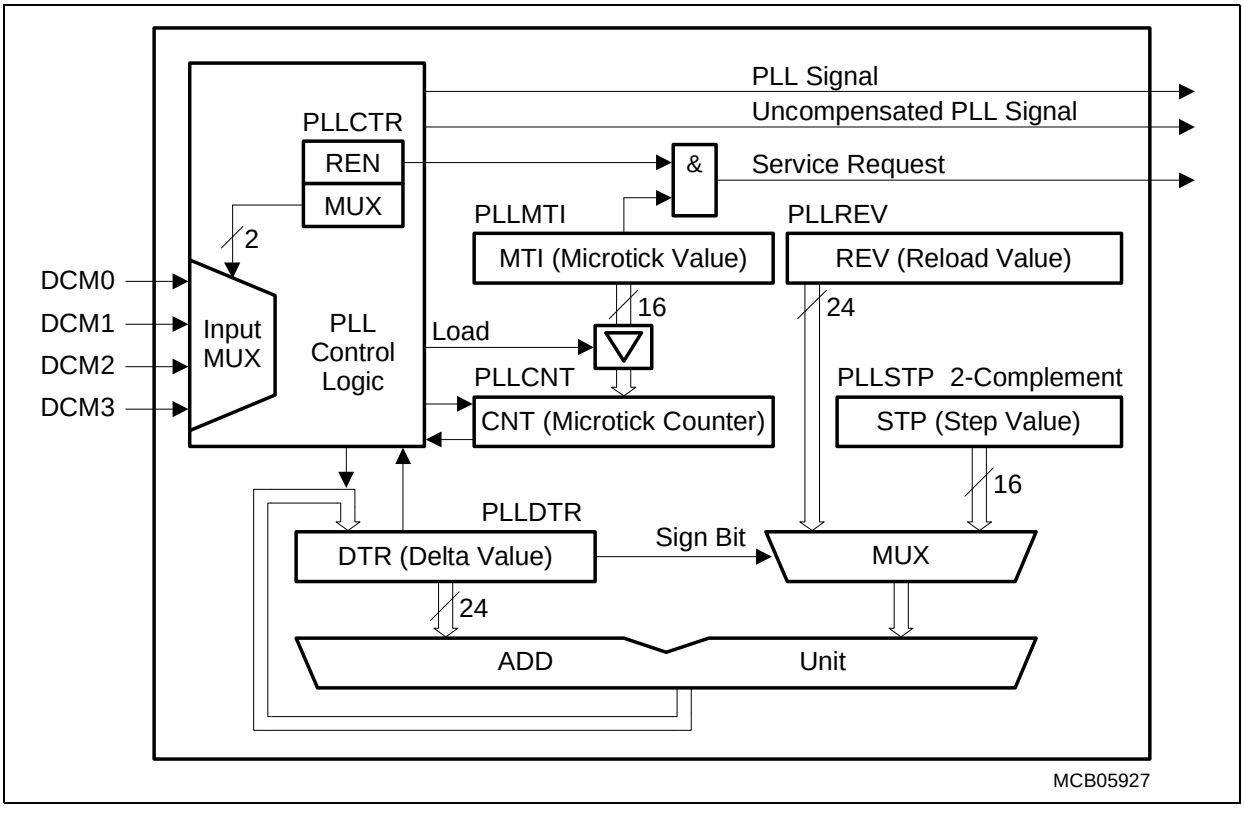


Figure 22-18 Block Diagram of Digital PLL Cell

PLL Interrupt Control

The PLL cell is able to generate a service request output signal PLLSR. This signal is controlled as shown in **Figure 22-19**. When the service request condition PLLCNT = 0 occurs, the service request flag is always set. The service request output PLLSR is activated only if it is enabled by the enable bit PLLCTR.REN. Additional information about service request and interrupt handling are given in section **"Interrupt Sharing Block (IS)" on Page 22-123**.

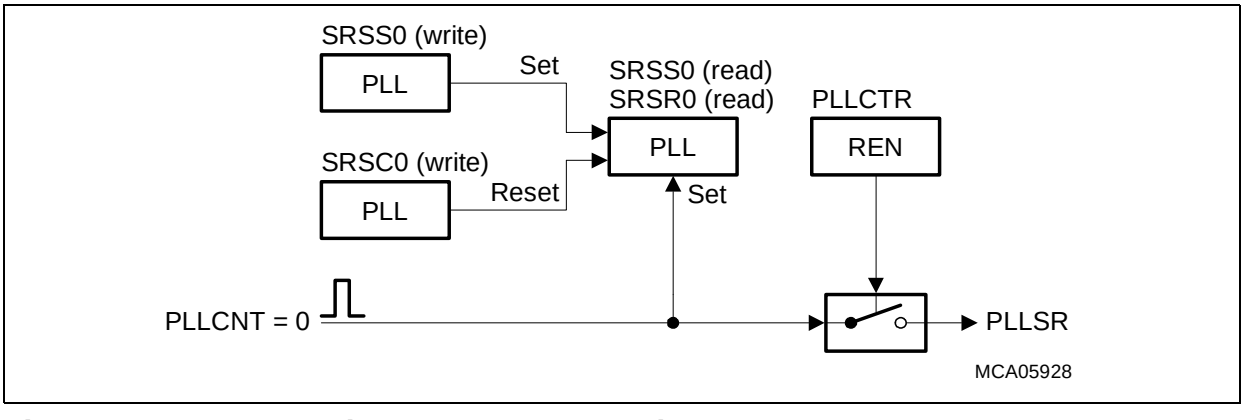


Figure 22-19 PLL Service Request Generation

Steady Input Signal Example

In the following example, the input signal's period length is 13 f_{GPTA} clock periods, which should be subdivided into three equally spaced sections. The reload value to be stored in PLLREV.REV register is calculated to $0A_H$ (10 = 13 - 3). PLLMTI.MTI is loaded with 03_H (number of output pulses) and its 2-complement representation ($FFFD_H$) is written into PLLSTP.STP.

After a reset, a state machine driven by the GPTA®v5 module clock, updates the delta register PLLDTR with the reload value. Afterwards, the PLLSTP register's contents are continuously added to the delta register value (**Figure 22-20**). In fact, the difference between both values is computed and stored in the PLLDTR register again, because the PLLSTP register has been loaded with a negative value (2-complement data format). When the PLLDTR register has been decremented to a negative value, the reload register contents are added to Delta register's current contents.

A rising edge detected on the selected input signal triggers the counter register PLLCNT to load the number of requested output pulses from PLLMTI. When a negative content of the PLLDTR register is detected, the microtick counter is decremented by one. In Automatic Mode (AEN = 1), the output pulse generation is stopped when the microtick counter reaches zero.

The period length of a single output pulse varies between four and five f_{GPTA} clocks; the maximum period length variation of output pulses is restricted to one f_{GPTA} clock. The total period length of all three output pulses, generated by one PLL loop corresponds to the input signal period width (5 + 4 + 4 = 13 f_{GPTA} clocks).

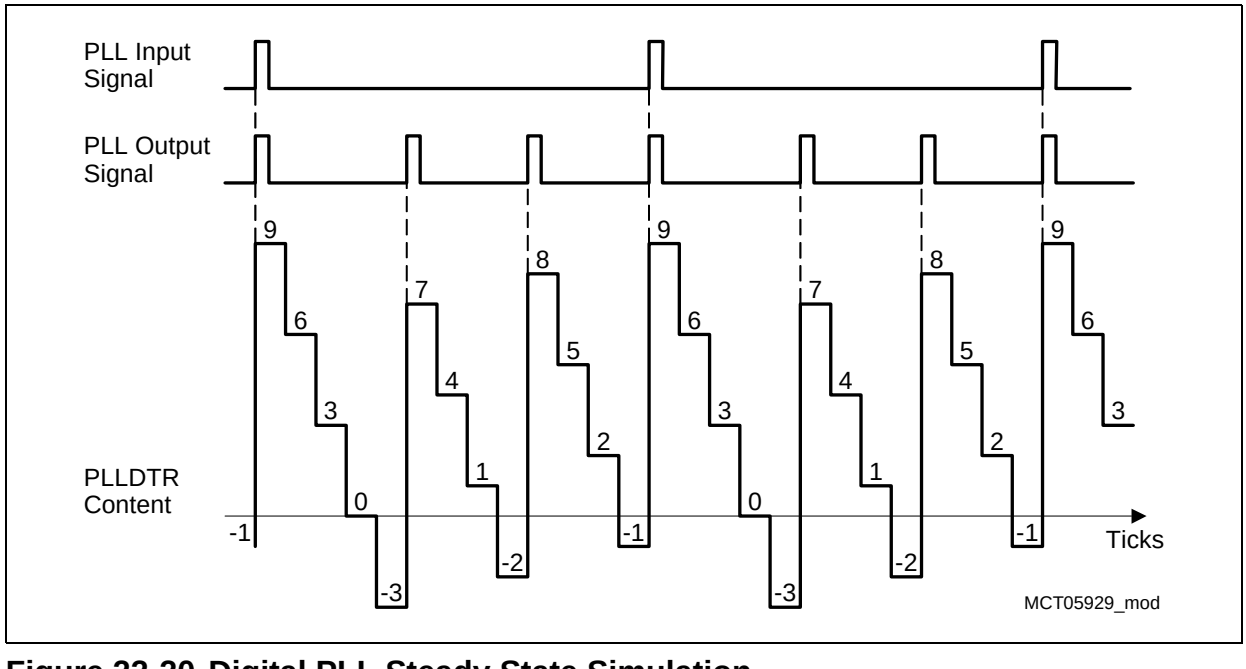


Figure 22-20 Digital PLL Steady State Simulation

This type of PLL implementation presents a valuable advantage compared to classic PLL implementation. Indeed, the generated microticks are equally distributed. The division reminder is distributed to several clocks instead of adding this reminder to the last pulse clock of the period.

Figure 22-21 illustrates this advantage. Considering a period of 15 clock pulses to be divided by a factor of 4, it gives a result of 3 with a reminder equal to 3. The reload value is calculated to $0B_H$ (11 = 15 - 4). The number of output pulses is equal to 4 and its 2-complement representation ($FFFC_H$) is written into the step register.

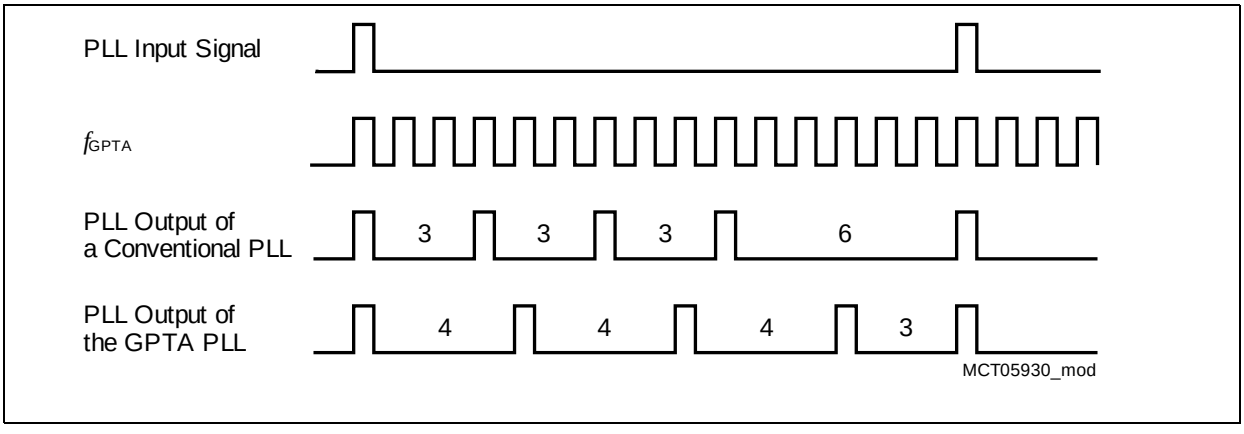


Figure 22-21 Advantage of the GPTA®v5 PLL

Input Signal Acceleration and Deceleration

The consequence of an input signal acceleration or deceleration can be compensated either automatically or by an external software routine. It detects an input signal's period length variation by comparing the current period length (measured in the associated DCM cell) with the expected period length used as calculation base for the PLLREV register contents.

- Compensation of input signal deceleration
  - Compensation by PLL Automatic End Mode
    If Automatic End Mode is enabled (PLLCTR.AEN = 1), the PLL stops at the calculated end of the current input signal period. Due to the deceleration, the rising edge of the following input signal period is delayed, starting the next PLL operation later than expected. A gap occurs between the last output pulse of the current input signal period and the first pulse of the following one (see **Figure 22-22**).
  - Compensation by Software
    After disabling the Automatic End Mode (PLLCTR.AEN = 0), the PLL generates output pulses without synchronization to an input signal edge. In case of a deceleration, more output pulses than calculated are generated during one input signal period. Several algorithms can be implemented to compensate the surplus of generated output pulses:
    The length of the current input signal period has been underestimated by a certain

number of f_{GPTA} clock periods. This deficit could be added to the calculated length of the next input signal period.
The PLL can continue to operate with the old input signal period length estimation, but the number of output pulses to be generated during the next input clock period may be decreased by the surplus of output pulses initiated during the last signal period.

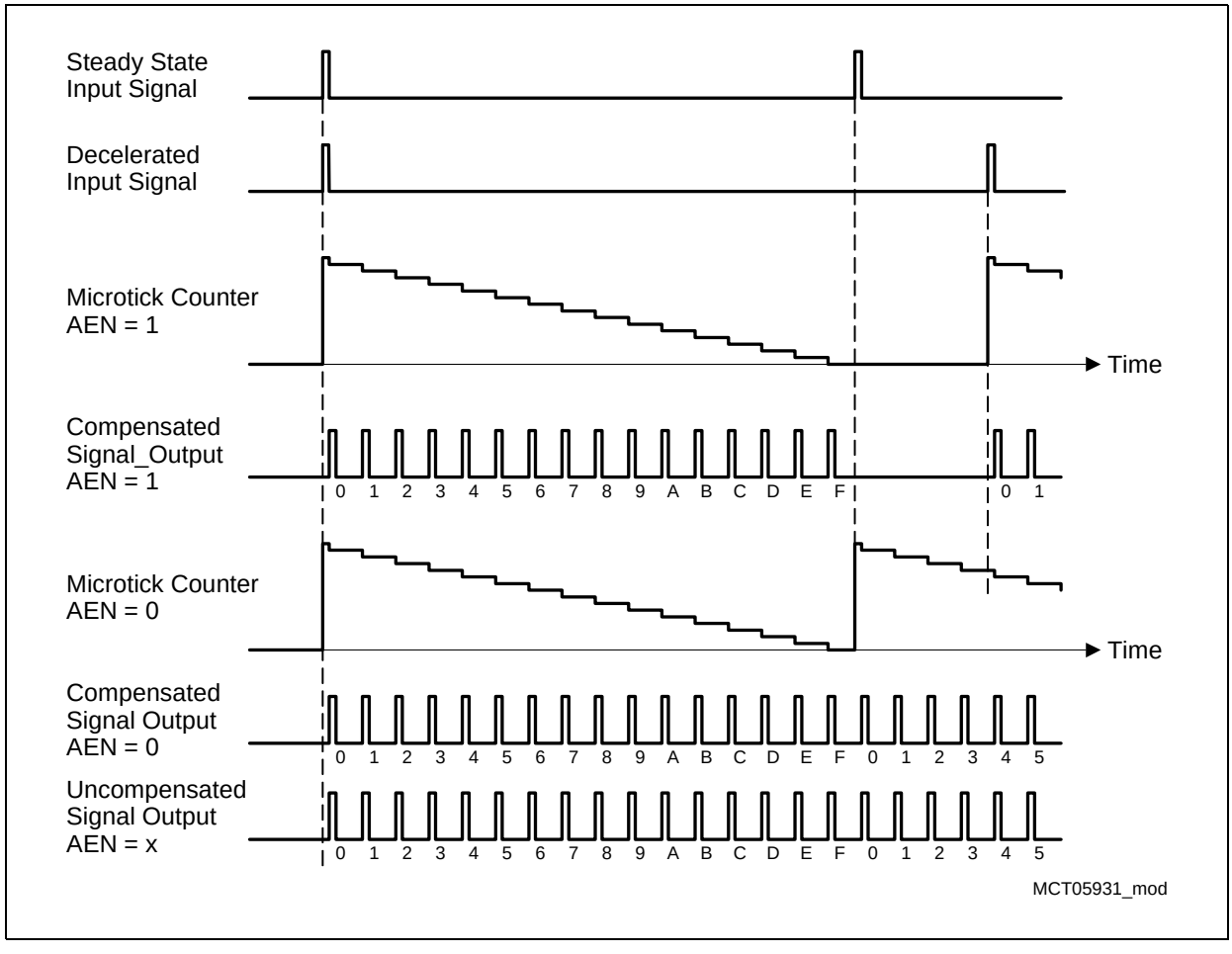


Figure 22-22 Compensation of Input Signal Deceleration

- Compensation of input signal acceleration
  - Compensation by PLL Automatic End Mode
    The next rising edge of the input signal arrives while the counter has not been decremented to zero. The PLL performs all remaining output signal pulses at full speed (f_{GPTA}), when control register bit AEN is set to 1. Afterwards, counter and Delta register are reloaded with their calculated values and the PLL operates at normal speed (see **Figure 22-23**).
  - Compensation by Software
    After disabling the Automatic End Mode, the PLL generates fewer output pulses than calculated during one input signal period. Several algorithm can be

implemented to compensate for the lack of generated output pulses:
The length of the current input signal period has been overestimated by a certain number of f_{GPTA} clock periods. This deficit should be subtracted from the calculated length of the next input signal period.
The PLL can continue to operate with the old input signal period length estimation, but the number of output pulses to be generated during the next input clock period may be increased by the lack of output pulses initiated during the last signal period.

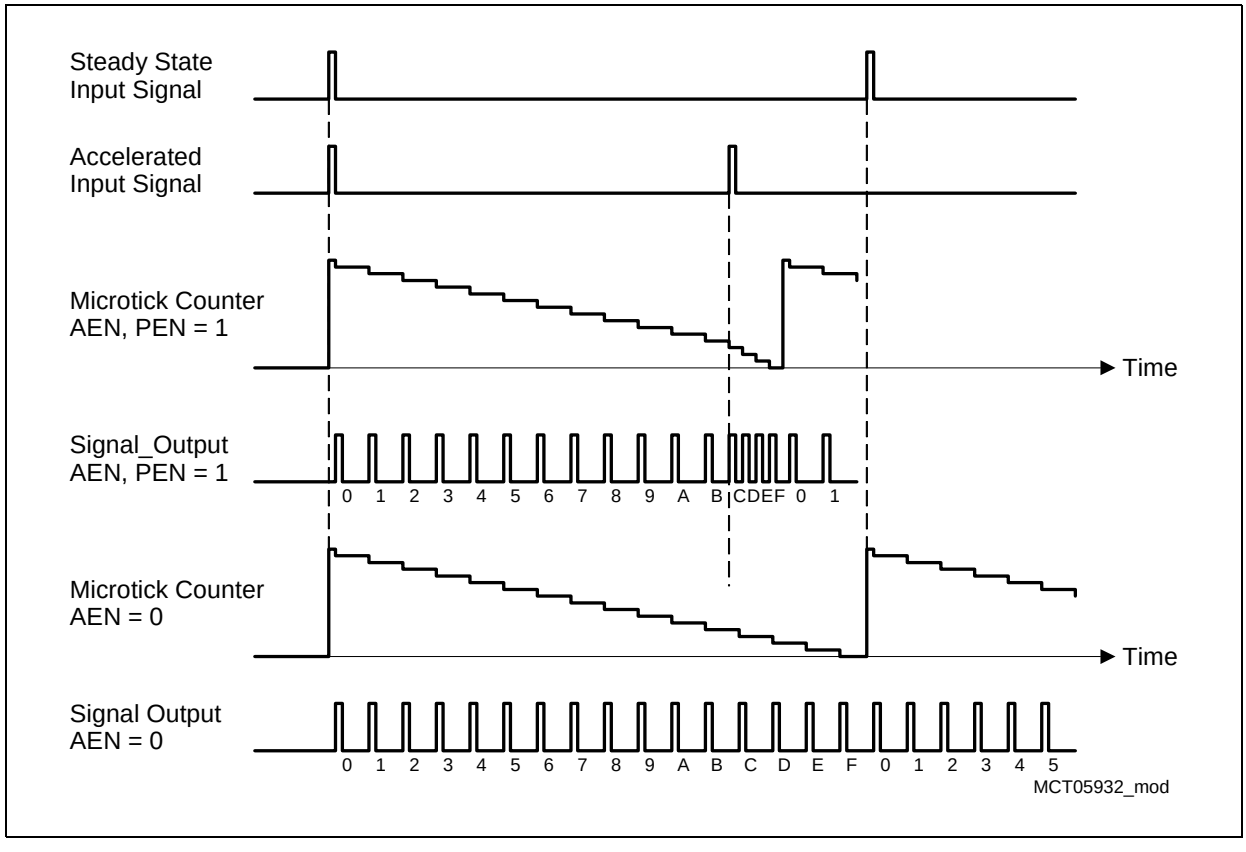


Figure 22-23 Compensation of Input Signal Acceleration

Additionally to the normal output signal, the PLL provides an uncompensated output signal. This signal has no gaps or acceleration bursts. However, the number of microticks during one signal period may be incorrect.

22.3.2.5 Clock Distribution Cell (CDC)

The Clock Distribution Cells (CDC) provides all Local and Global Timer Cells with a clock bus containing eight different clock output signals CLK[7:0] and a special LTC prescaler clock LTCPRE. These nine clock signals are generated out of eleven clock input signals coming from different clock sources (see **Figure 22-24**).

The prescalers divide the GPTA®v5 module clock f_{GPTA} by a programmable 2^n factor. Factor n is defined by bit fields DFA02, DFA04, DFA06 and DFA07 of control register

CKBCTR. A bit field value of 15 disables the related prescaler and selects alternate sources for clock bus lines 2, 4, 6 and 7. For clock bus line CLK2, a bit field value of 14 selects an alternate source.

For clock bus line CLK3, the 2-bit wide bit field DFA03 of control register CKBCTR selects one of the four available clocks.

The LTC prescaler clock LTCPRE is generated by dividing the f_{GPTA} module clock by a factor defined by the 3-bit wide bit field DFALTC of control register CKBCTR. Note that the LTCPRE clock is not a part of the clock bus but a clock signal that is distributed directly from the CDC to each LTC except LTC63.

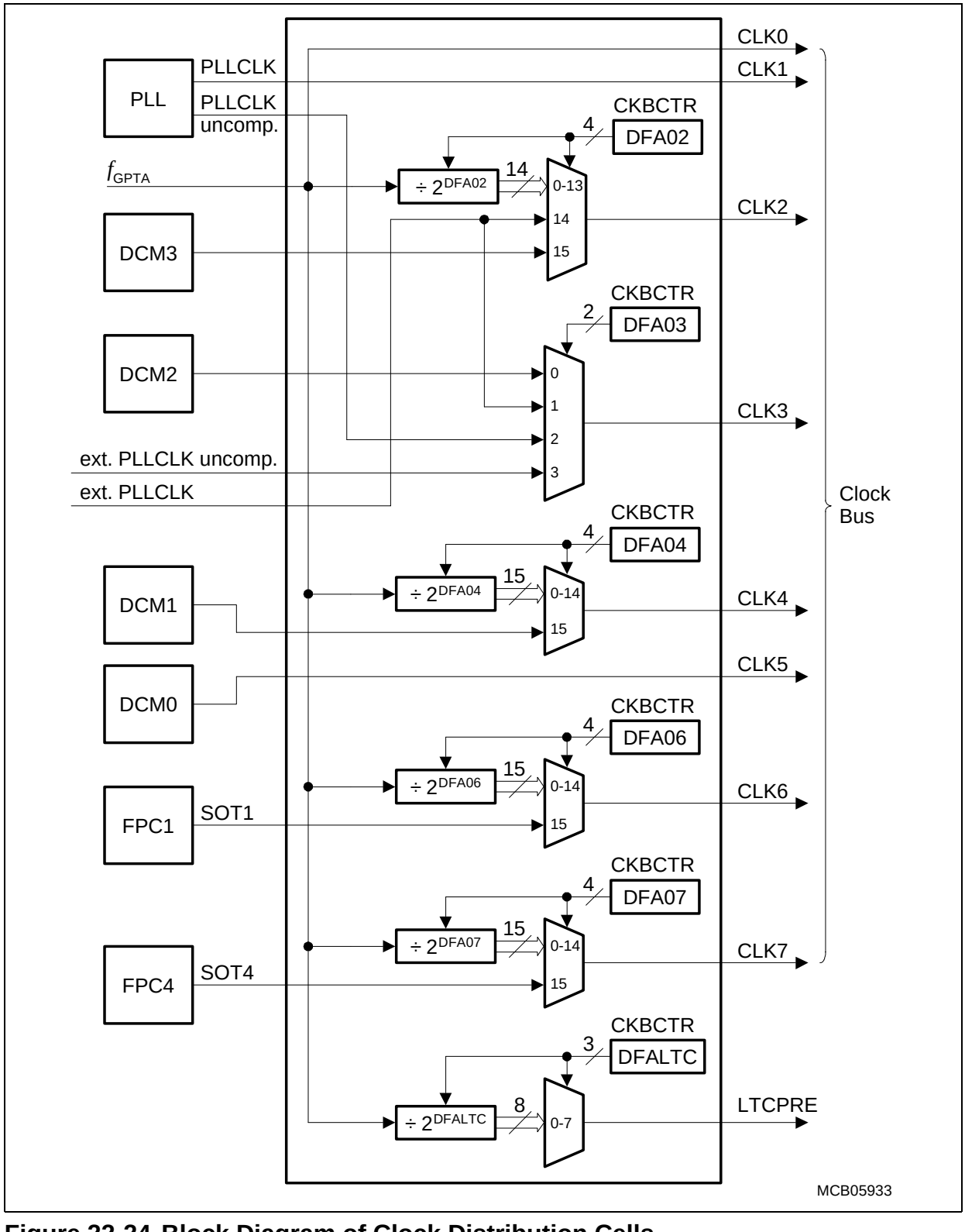


Figure 22-24 Block Diagram of Clock Distribution Cells

22.3.3 Signal Generation Cells

As described in detail in the following sections, the Signal Generation Cells contains the following types of cells:

- Global Timer (GT)
- Global Timer Cell (GTC)
- Local Timer Cell (LTC)

22.3.3.1 Global Timers (GT)

The GPTA®v5 provides two global 24-bit timers (GT) that are connected to the clock bus with its eight clock lines. Each GT is locally equipped with a clock source multiplexer, a 24-bit up-counter, a 24-bit reload register, and a 24-bit greater/equal comparator (see **Figure 22-25**).

Note: Index variable k (= 0, 1) determines the number of the Global Timer.

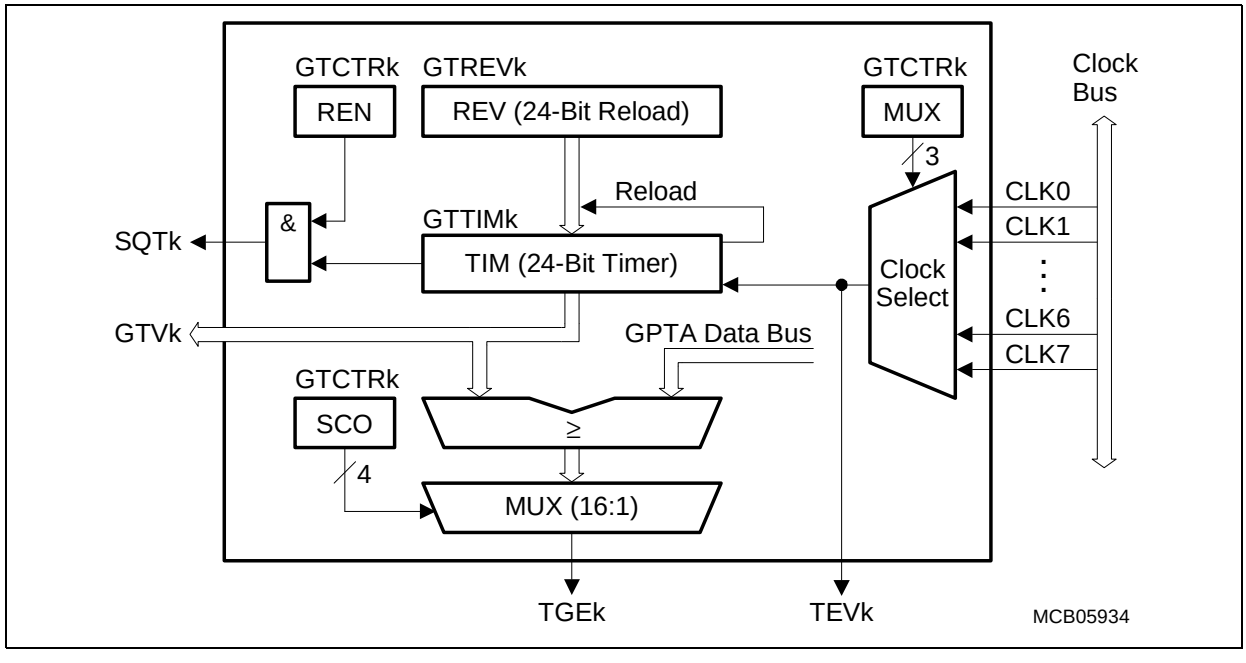


Figure 22-25 Block Diagram of Global Timer (GT)

The following registers are assigned to the Global Timers GTk (k = 0, 1):

- GTCTRk = Global Timer Control Register k (see **Page 22-182**)
- GTREVk = Global Timer Reload Value Register k (see **Page 22-184**)
- GTTIMk = Global Timer Register k (see **Page 22-183**)
- SRSC0 = Service Request State Clear Register 0 (see **Page 22-223**)
- SRSS0 = Service Request State Set Register 0 (see **Page 22-225**)

Each of the two GT cells provides the following input/output signals:

- Eight clock inputs, connected to the clock bus from the clock distribution cells (CDC)
- Global timer value bus GTVk (outputs), carrying the 24-bit GTk counter value
- TEVk output, indicating a GT counter update
- TGEk output, indicating the result of a compare operation
- SQTk service request output, triggered at a timer overflow.

The Global Timer output signals GTVk, TEVk, and TGEk are available as input signals at each GTC (see also **Page 22-56**).

Global timer k can be initialized with a start value, that is written by software into the GTTIMk register. The 24-bit Global Timer value GTTIMk.TIM is incremented by each rising edge of clock input signal TEVk that is selected from the 8-bit clock bus via bit field GTCTRk.MUX. On a Global Timer overflow (transition of $FFFFFF_H$ to 000000_H), the following events occur:

- The 24-bit reload value GTREVk.REV is copied into GTTIMk.TIM
- Bit SRSC0.GT0k is set
- The service request output SQTk is activated (if enabled by bit GTCTRk.REN)

A free-running timer is configured by programming GTREVk.REV with 000000_H.

The “Timer Event” (TEVk) output is activated if the GTk value changes because of a clock edge, a timer reload operation, or a software write access to GTCTRk. The TEVk output is connected to all GTCs. TEVk is used in the GTCs to trigger a compare operation, re-checking the equality of their compare register contents and the updated Global Timer value.

GT Interrupt Control

Each of the GTs is able to generate a service request output signal SQTk. This signal is controlled as shown in **Figure 22-26**. On a GTk timer overflow, the service request flag GT0k is always set. The service request output SQTk is activated only if it is enabled by the enable bit GTCTRk.REN. Additional information about service request and interrupt handling is given in section **“Interrupt Sharing Block (IS)” on Page 22-123**.

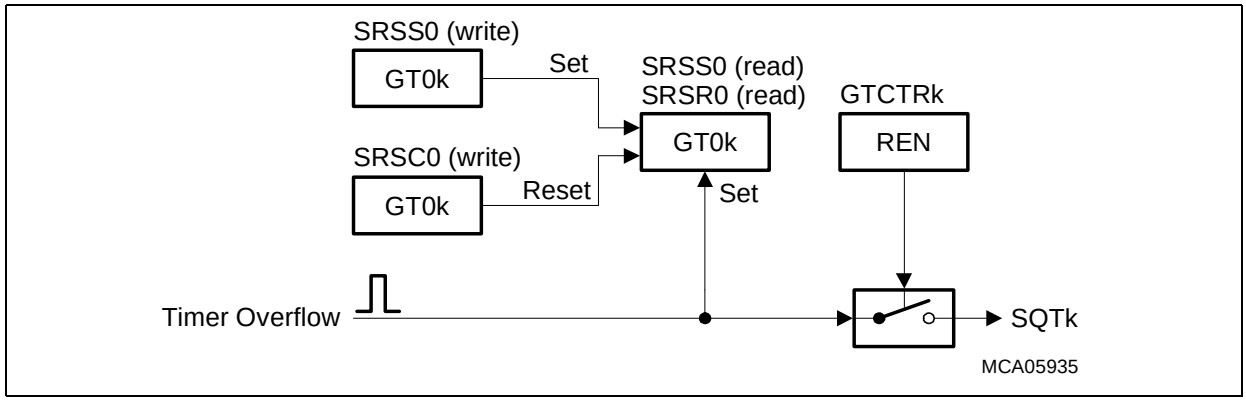


Figure 22-26 GTk Service Request Generation

Synchronization of Global Timers

Both Global Timers, GT0 and GT1, can be enabled and disabled individually. Each GT has its own run signal GTkRUN that is generated outside the GPTA®v5 kernel (see also **Page 22-8**). Signal GTkRUN is generated in a GPTA®v5 clock control circuitry. This external control capability allows the run signals GTkRUN to be controlled in a way that all Global Timers of one ore more GPTA®v5 units can be enabled/disabled synchronously.

The two Global Timers will run synchronously only if all of the following conditions are true:

- Timers use the same input signal
- Timers are started (and stopped, if required) synchronously
- Timers use identical start and reload values
- Timers are not written while they are running

Scalable Signed Greater or Equal Compare

This section (up to **Page 22-54**) explains the **classical timer update problem**, and the solutions supported by the GPTA®v5.

The two Global Timers embedded into the GPTA®v5 include a 24-bit greater/equal comparator. This comparator cell performs compare operations between the GT timer contents and the data value found on the GPTA®v5-internal data bus (coming from a GTC compare register update). The goal of this comparator is to be able to perform an action immediately if the compare cell is updated with a new threshold but the timer has already passed this value. **Figure 22-27** gives an example on this greater/equal concept.

Assumption: a timer is running and a new threshold (value T) is set.

The different points Px represent different cases of present time. When at P1 or P2, the moment represented by T lies in the future and no action is yet required. When at P3 or P4, the moment represented by T lies in the past, and an action is required immediately.

So, the problem is to determine if the threshold T has been passed or not.

Considering an **infinite counter**, the situation is simple. The evaluation consists in determining if point P is before or after T.

Considering a **reloaded counter**, as the timer rolls over at its maximum value, the situation is more complex.

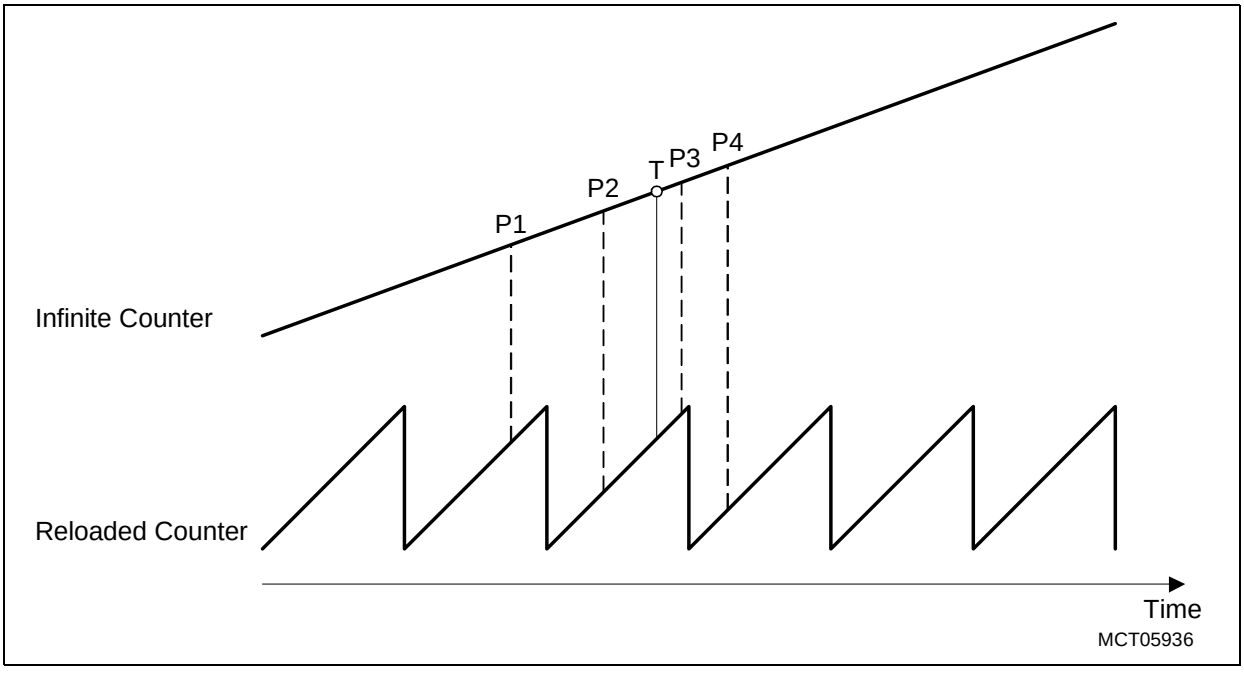


Figure 22-27 Greater/Equal Concept

The **observation window** determines the space in time where writing the value T to the comparator will lead to correct observation (meaning, there is an event if “After”; there is no event if “Before”). Considering an observation window, an event (threshold T) is

programmed and then the window is split into two windows, the “After” window and the “Before” window (**Figure 22-28**). If the timer lies in the “After” window at the time of programming the threshold, the event is performed immediately. If it lies in the “Before” window, the event will happen later when the timer reaches the threshold T. The “Before” window refers to a “prediction range”, and the “After” window refers to the “history buffer”.

From a practical point of view, once the value T is determined, it is necessary to calculate the observation window (position and width). Before updating the value T, the application must assure that the observation window was entered but has not yet been left.

The width of the **observation window** cannot exceed the timer period. To support reloaded counters where the overflow can occur within the observation window, a **signed** comparison is performed.

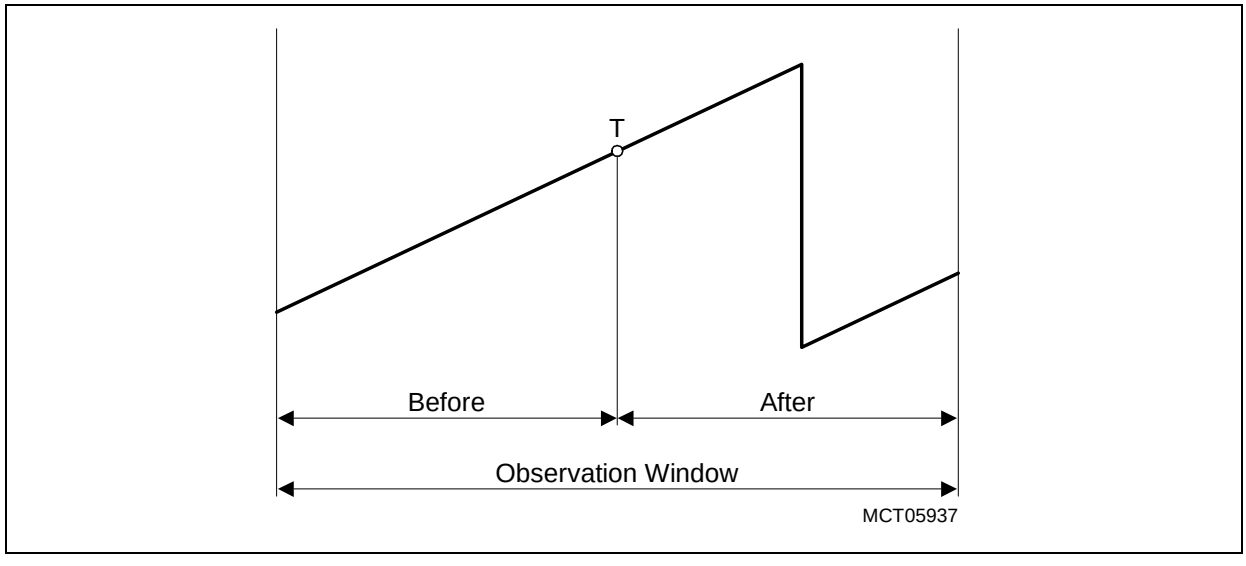


Figure 22-28 Before and After Windows

Comparison Between Unsigned and Signed Compare

To be able to support different timer periods and to support correct observation even beyond timer overflow, the GPTA®v5 embeds the **scalable** and **signed** greater/equal comparator. Using a signed comparison allows one overflow of the timer to occur within the observation window. This is illustrated in **Figure 22-29**.

Using a signed compare in order to take into account the timer overflow, the comparator window is introduced. The comparator window is centered to the point T and its width can be selected by the user.

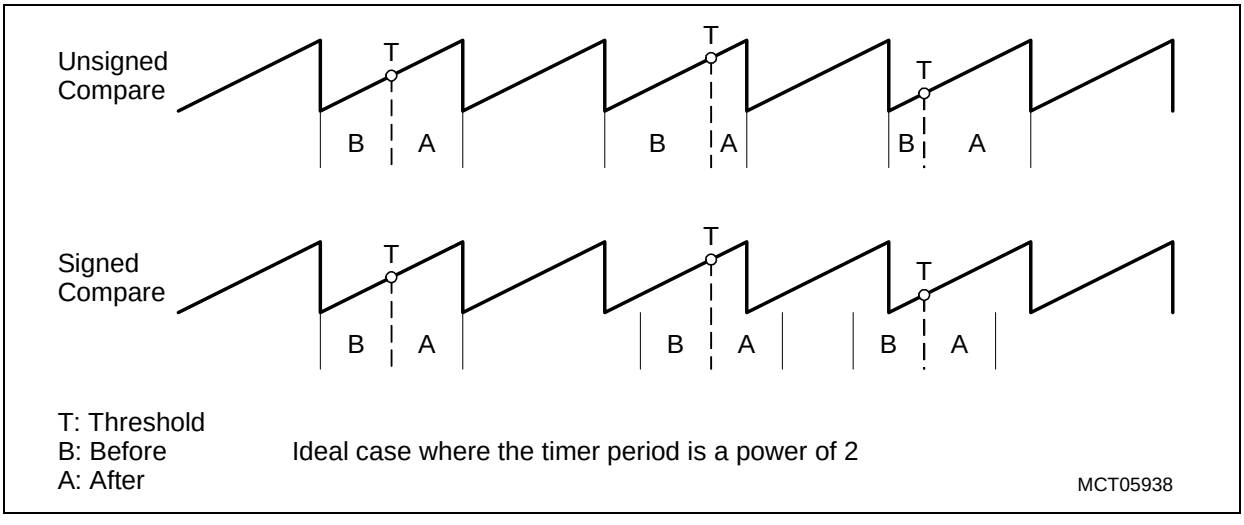


Figure 22-29 Unsigned Versus Signed Compare

When the timer range is a multiple of 2 and because the comparator is scalable, the observation window and the comparator window are identical. See **Figure 22-30**.

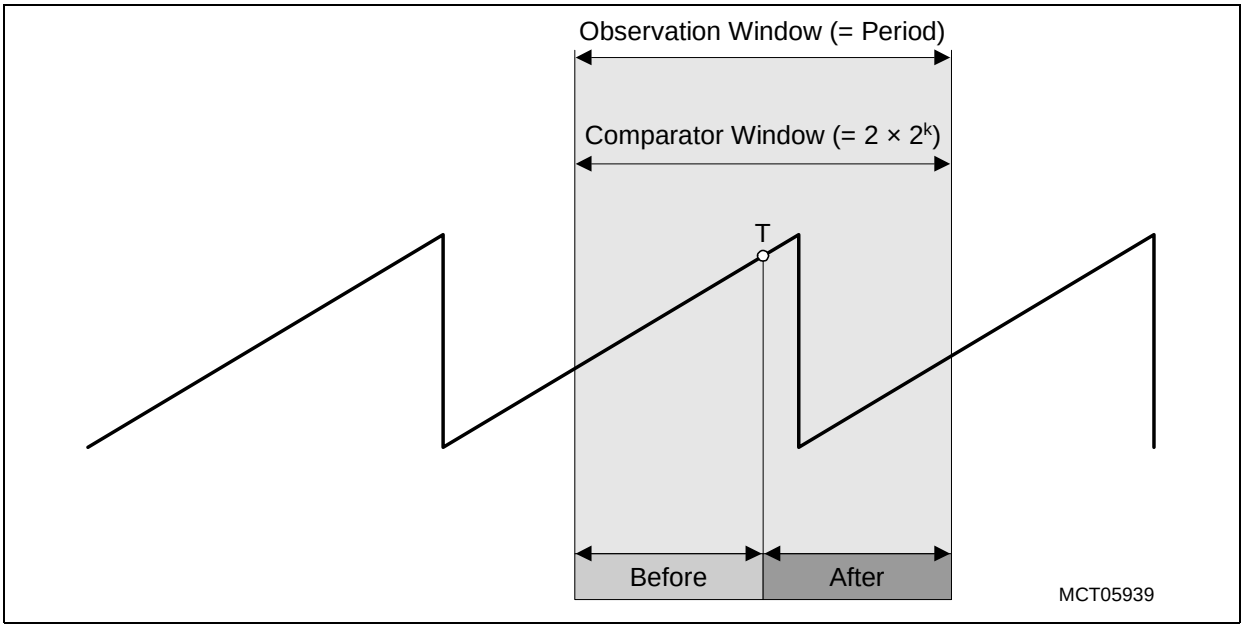


Figure 22-30 Observation and Comparator Windows (timer is a power of 2)

The scalable and signed greater/equal comparator scheme leads to a limitation that must be considered when programming the GPTA®v5 Module. If the timer range is not a power of 2, the comparator window (always a power of 2) will no longer match the timer period. This will impact the observation window as described in the following paragraph.

Observation window for reloaded timers (period is not a power of 2)

In that case, the comparator window must exceed the timer period. The user must find the comparator window (by selecting the scale factor k) which fits best the timer period.

The following equation must apply:

$$2^k < \text{Period} \leq 2 \times 2^k \quad (22.1)$$

Figure 22-31 and **Figure 22-32** show that one part of the comparator window must be discarded in order to avoid inconsistency, resulting in the observation window.

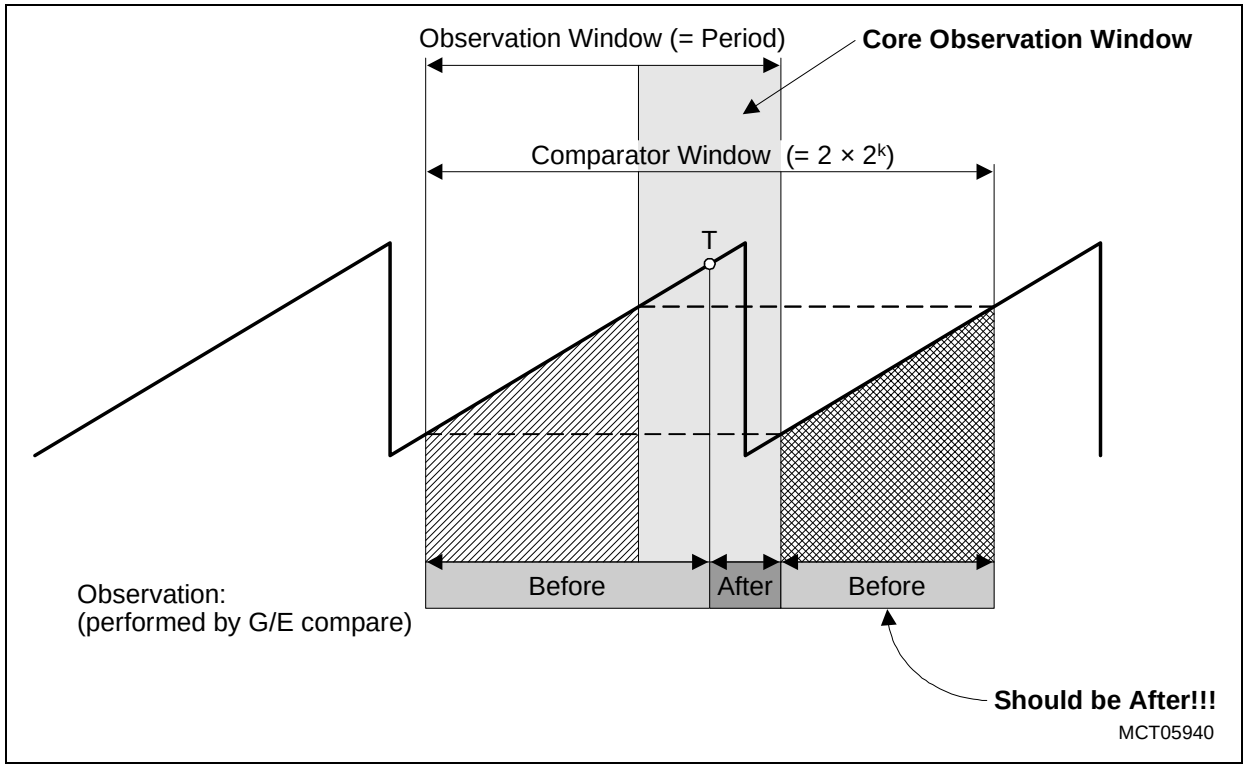


Figure 22-31 Observation Window when Threshold T is High

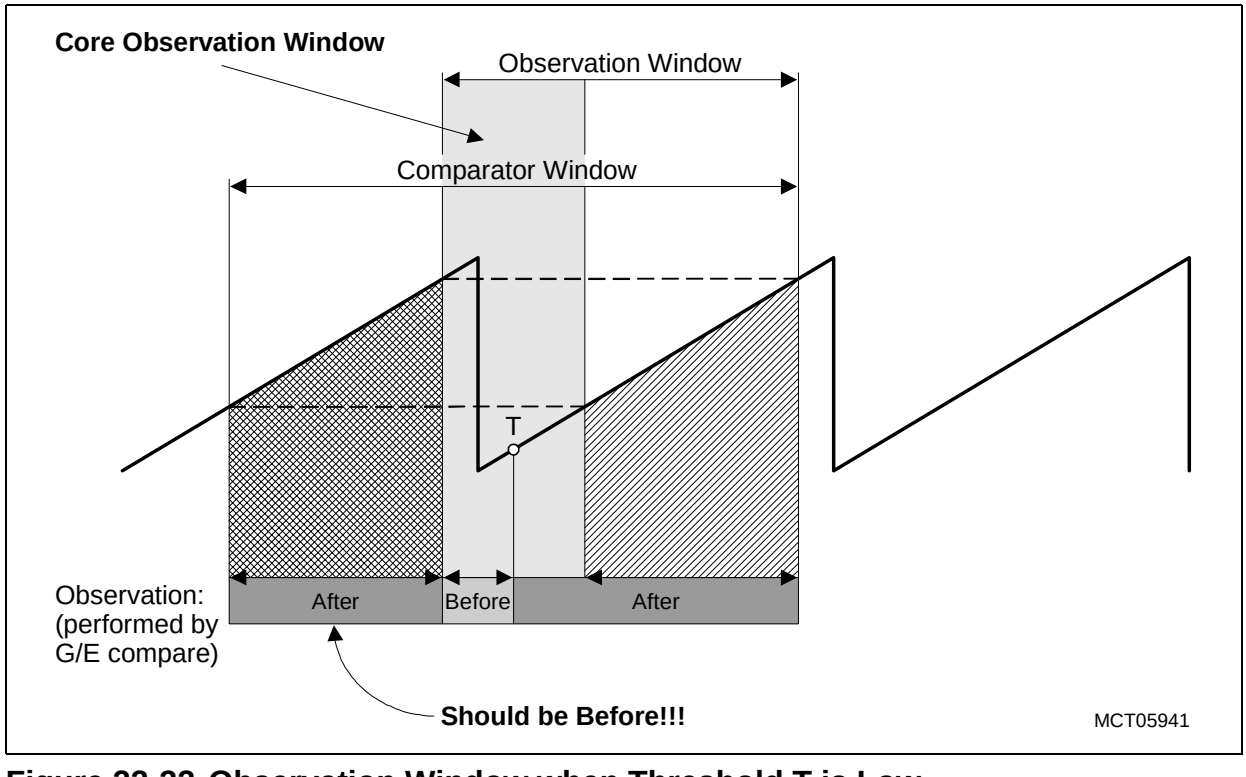


Figure 22-32 Observation Window when Threshold T is Low

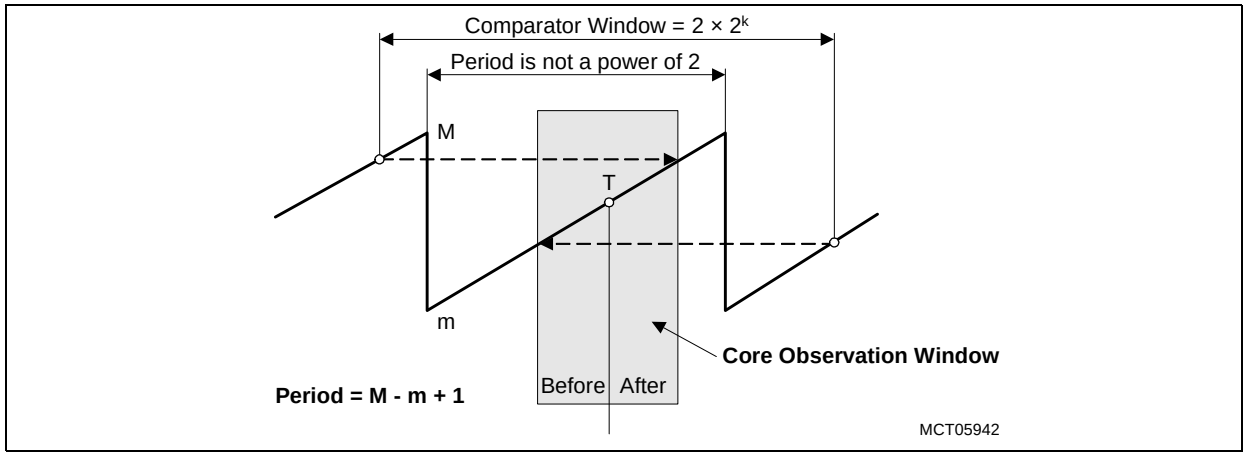


Figure 22-33 The Core Observation Window

A comparison of the previous figures shows that the position of the observation window with respect to T is dependent on the value of T itself. That means the user, before updating the comparator with T, needs to calculate the observation window as a function of T. To avoid this calculation, a **core observation window** can be defined that is independent of T. It will always be centered on T, whatever its value. However, one particularity exists when using the core observation window: the size of the core observation window varies depending on two static values: the timer period and the comparator window's sizes. In particular, the core observation window reduces as the value of the timer period is just after a power of 2. This is shown in **Figure 22-34**.

For any timer period (whatever the range) and any threshold position, a symmetrical core observation window of a statically defined size can be determined.

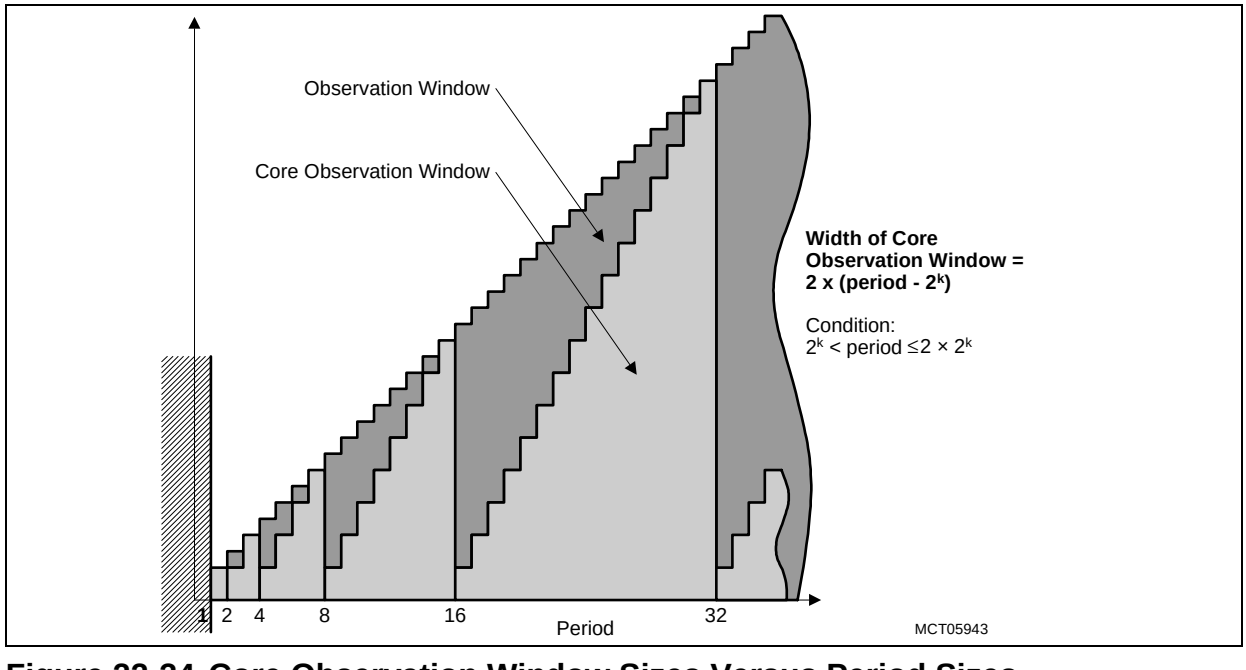


Figure 22-34 Core Observation Window Sizes Versus Period Sizes

Implementation

The hardware implementation of the **scalable and Signed/Unsigned** Greater/Equal compare is illustrated in **Figure 22-35**. The function consists of subtracting the threshold T from the GT timer value. The result is in 2s complement format. The result's sign bit and the 15 most significant bits are at available for observation. One of those bits is selected according to the mode of operation (Unsigned or Signed) and the period length (bit field GTCTRk.SCO). This bit drives the TGE (Timer Greater Equal) flag.

Unsigned compare: Select Sign bit (SCO = $0F_H$)

Signed compare: Select one of the 15 most significant result bits (SCO = 00_H to $0E_H$)

Note: How to choose one of the 15 bits is explained later.

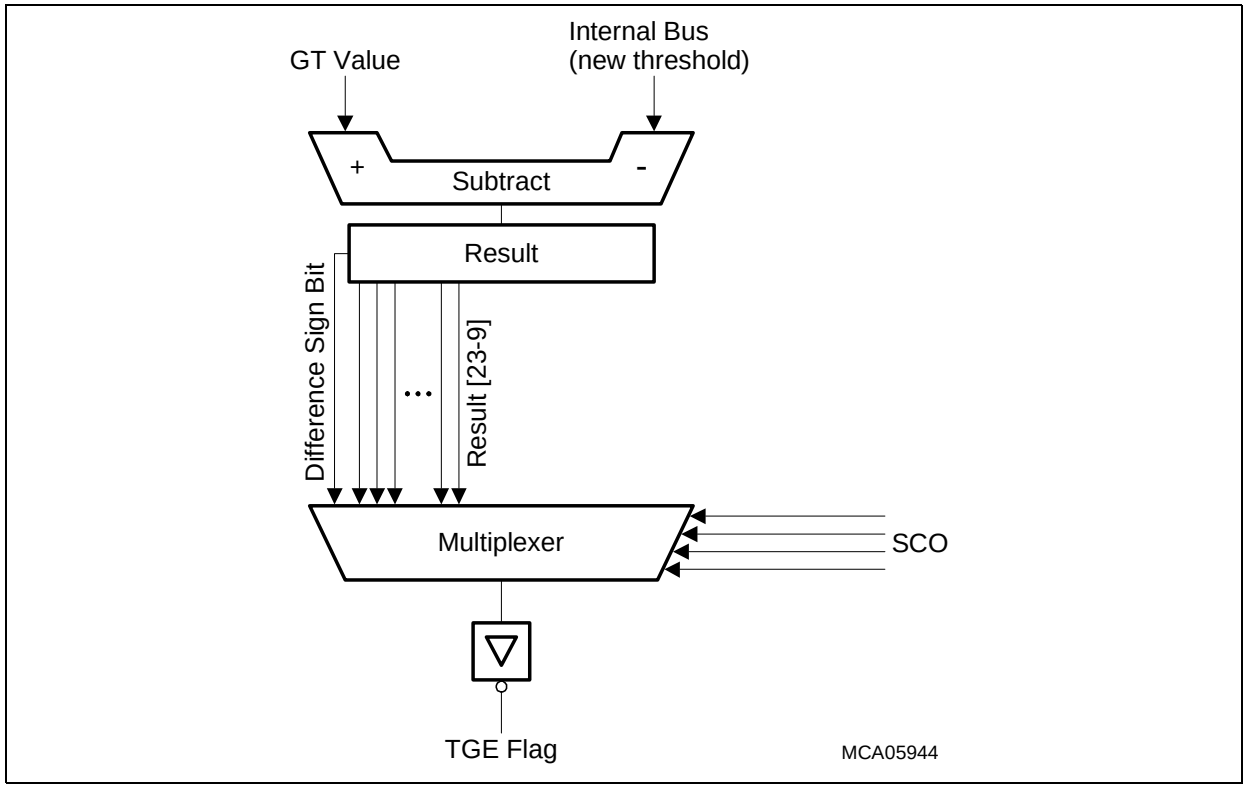


Figure 22-35 Comparator Implemented by a Subtraction Circuitry

The interpretation of the selected result bit is provided in the following simple example: For a 4-bit timer, the subtraction of the threshold T from the timer value, leads to a 4-bit signed result, as illustrated in **Figure 22-36**. This example is selected for simplicity although 4-bit periods are not covered by the implementation.

When using Unsigned compare, the sign bit S is selected. If it equals 0, the result is positive, indicating that the timer is greater or equal the threshold, and hence **After**. If it equals 1, the result is negative, and the observation indicates **Before**.

When using Signed compare, the result bit R_3 can be selected and interpreted, provided that the timer period is at least 9. Here, the range of the result can be split into four sub-

ranges. Because the result is in 2s complement format, a value of 0 for R_3 is interpreted as **After**, and a value of 1 is interpreted as **Before**. A comparison of **Figure 22-36** and **Figure 22-37** shows why this proceeding leads to correct interpretation within the observation window. **Figure 22-37** shows the case of a period equal a multiple of 2.

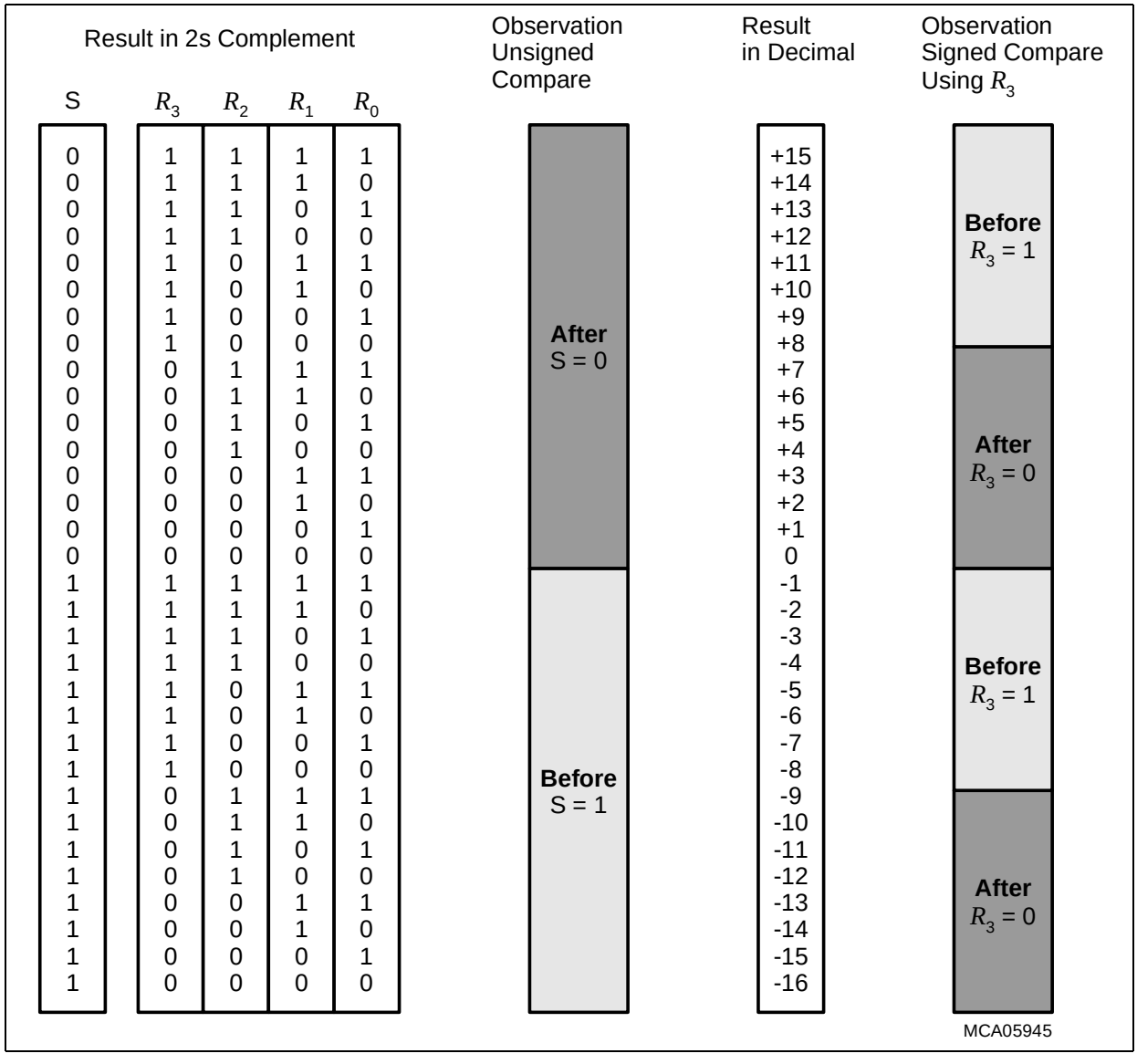


Figure 22-36 Result and Observation for a 4-Bit Timer

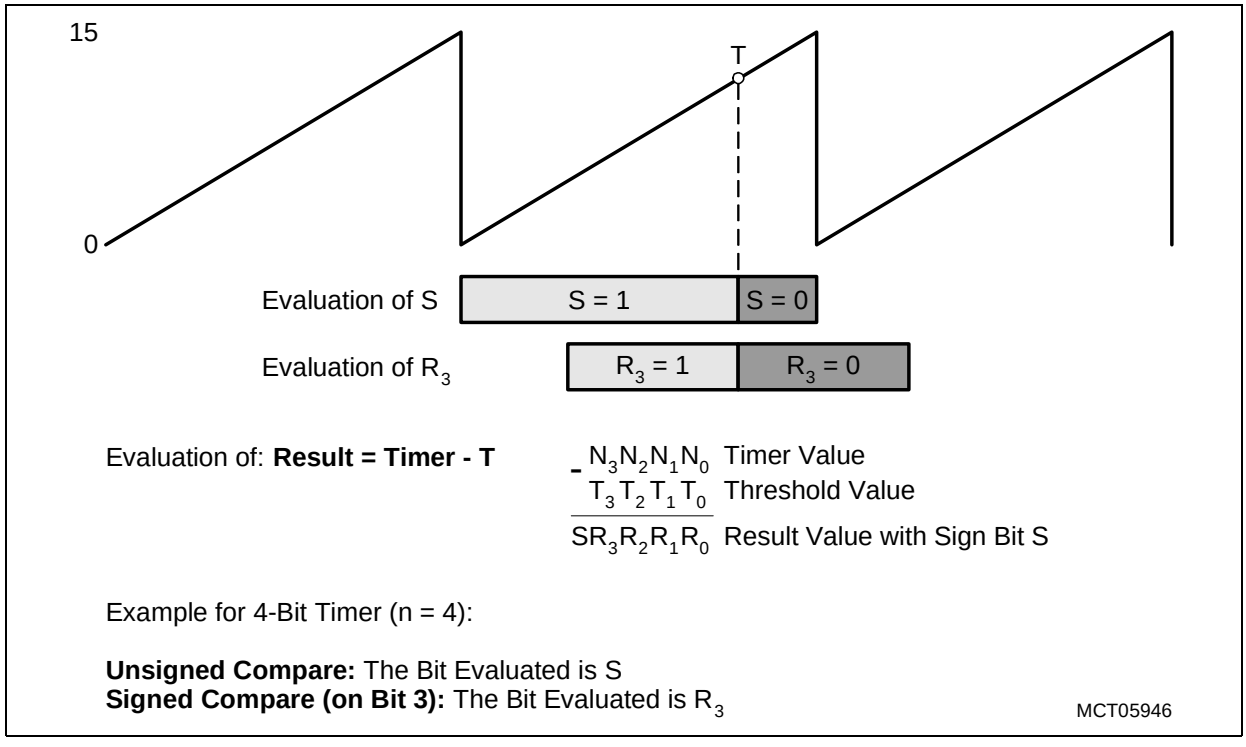


Figure 22-37 Result and Observation (Period = 16)

Figure 22-38 shows the case of a period of 12 which is not a power of 2. Here again, the Table in **Figure 22-36** applies.

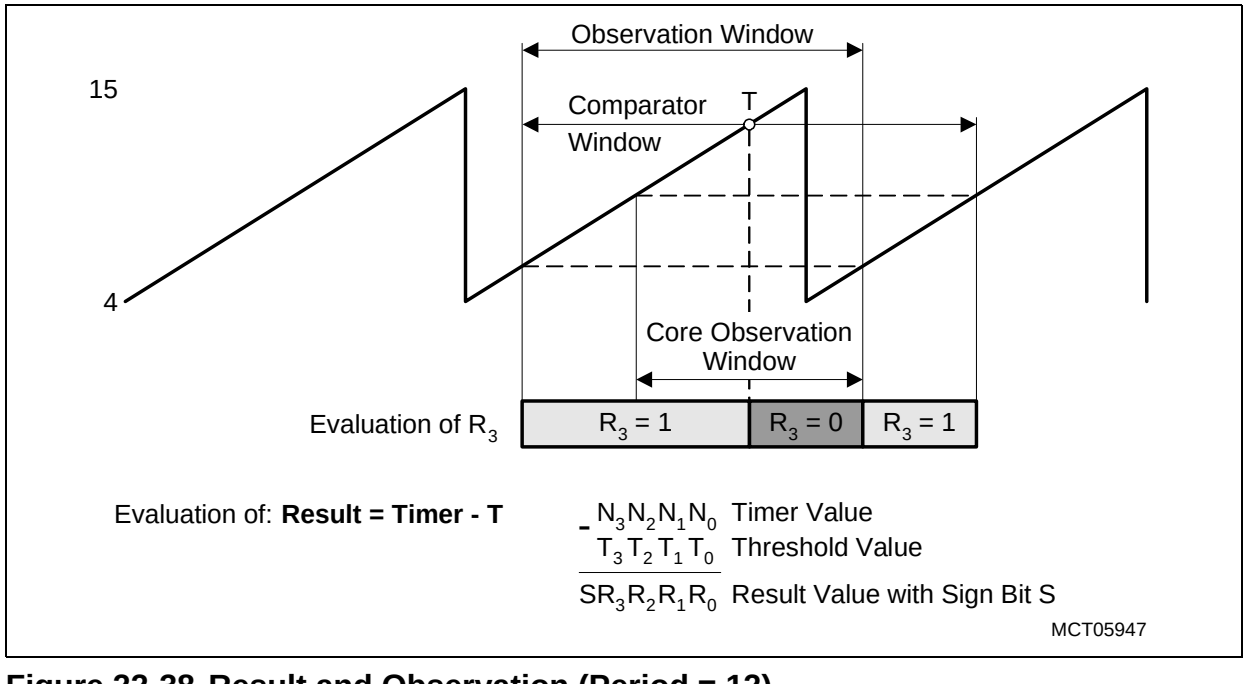


Figure 22-38 Result and Observation (Period = 12)

The previous examples show that the result bit to select for observation (R_3) corresponds to the comparator window's size (k = 3).

Considering the case in which the period is not a multiple of 2, choose a comparator window whose width is between 1 and 2 times the timer period:

$$2^k < \text{Period} \leq 2 \times 2^k \quad (22.2)$$

In no case may the comparator window be equal to or greater than twice the period.

k represents the Result bit to select.

How to Proceed

- Unsigned greater/equal compare:

SCO bit field = $0F_H$ (15_d)

Thereby, the sign bit of the result is selected to drive TGE flag.

This setting is valid for all possible periods. The observation window always matches the period.

- Signed greater/equal compare:

Depending on the period, the appropriate k is selected, so that:

$$\text{Period} = M - m + 1 \ (= \text{Max} - \text{Min} + 1) \quad (22.3)$$

$$2^k < \text{Period} \leq 2 \times 2^k \quad (22.4)$$

SCO bit field = 0 to $0E_H$ (0 to 14_d)

Thereby, the result bit R_k is selected to drive TGE flag.

This setting is possible for periods greater than 512.

Table 22-1 Period Range Depending on Selected k

$2^k < Period \leq 2 \times 2^k$	k	SCO Bit Field (decimal)
0 < period ≤ 512	Not covered by implementation	
512 < period ≤ 1024	9	0
1024 < period ≤ 2048	10	1
2048 < period ≤ 4096	11	2
4096 < period ≤ 8192	12	3
8192 < period ≤ 16384	13	4
16384 < period ≤ 32768	14	5
32768 < period ≤ 65536	15	6
65536 < period ≤ 131072	16	7
131072 < period ≤ 262144	17	8
262144 < period ≤ 524288	18	9
524288 < period ≤ 1048576	19	10
1048576 < period ≤ 2097152	20	11
2097152 < period ≤ 4194304	21	12
4194304 < period ≤ 8388608	22	13
8388608 < period ≤ 16777216	23	14

The width of the core observation window is defined by:

$$2 \times (period - 2^k) \quad (22.5)$$

As a consequence, the width of the “Before” window within the core observation window is (period - 2^k) and the width of the “After” window within the core observation window is (period - 2^k), including the value T.

Additional Information: Illustration on the General Case

The previous section illustrated the greater/equal compare for the particular case of a 4-bit timer. The purpose of this section is to describe the implementation from a general point of view, that is, for a timer period equal to M - m + 1.

In the following figures, the X axis indicates the timer value (elapsing time) and the Y axis indicates the threshold value T. The 45° line starting at (m, m) represents the position in time of T. The graphic shows the observation performed by the hardware for all cases of T (m ≤ T ≤ M).

Figure 22-39 illustrates the Unsigned compare. A particular case is shown in which, for a higher value of T, the observation indicates “Before” at the beginning of the period, and until the timer reaches the value T. Thereafter, the observation switches to “After” and remains there until the timer exits the period.

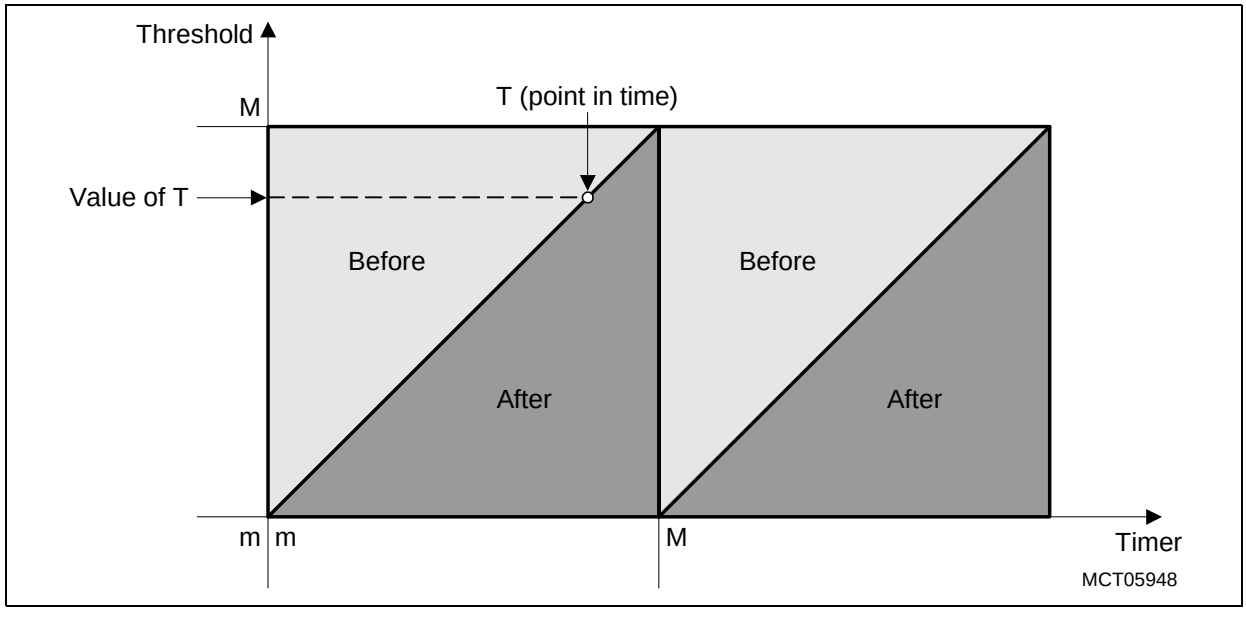


Figure 22-39 Graphical Representation of Unsigned Compare

Figure 22-40 illustrates the Signed compare where the period equals a multiple of 2 (that means M - m + 1 = 2×2^k). In this case, for a higher value of T, the observation indicates “After” at the beginning of the period (not yet inside the observation window). When entering the observation window, “Before” is indicated until the timer reaches the value T. Thereafter, the observation switches to “After” and remains there until the timer exits the observation window. This graphic can be related to **Table 22-37** where the comparator window equals the period, and the observation window is always centered on the threshold T.

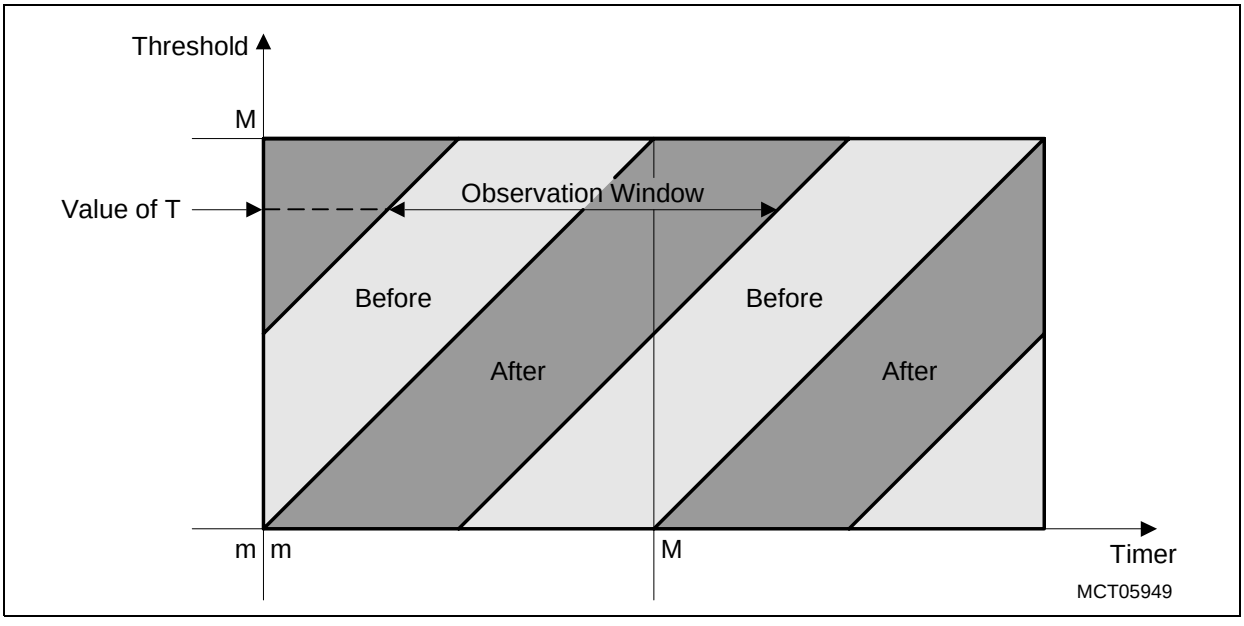


Figure 22-40 Graphical Representation of Signed Compare (Period = 2 × 2^k)

The **Figure 22-41** illustrates the Signed compare where the period may also be unequal a multiple of 2. The graphical representation of this general case is analogous to the one described in **Figure 22-31**.

If the period is not a multiple of 2, the graphical representation of the Signed compare shows a discontinuity in the “Before” and “After” ranges. Indeed, the widths of the “Before” and “After” windows are not constant, as they depend on the value T. As a consequence, the observation window is not centered on T. The result is that the position of the observation window would have to be re-evaluated for each value T (i.e. determining the widths of the “After” and the “Before” window). For this calculation, the principal characteristic is shown in **Table 22-41** (2×2^k - period = comparator window - period).

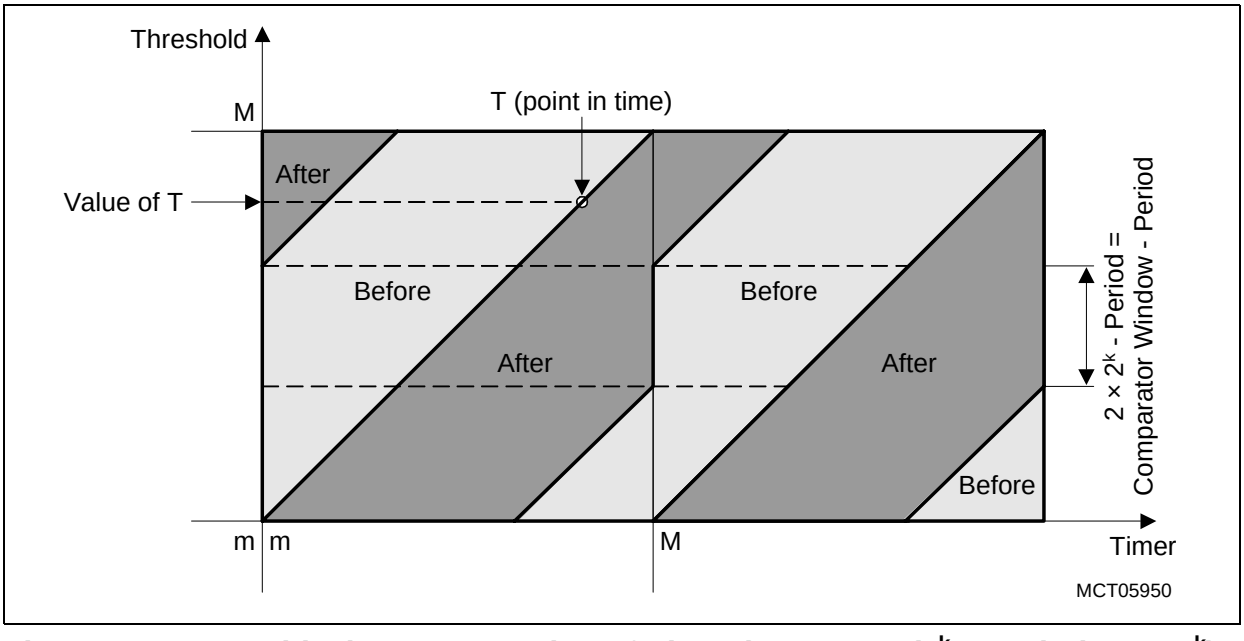


Figure 22-41 Graphical Representation of Signed Compare (2^k < Period ≤ 2 × 2^k)

Figure 22-42 shows how the observation window is positioned with respect to T. It also shows the **core observation window** that is always centered on T and which has a constant width.

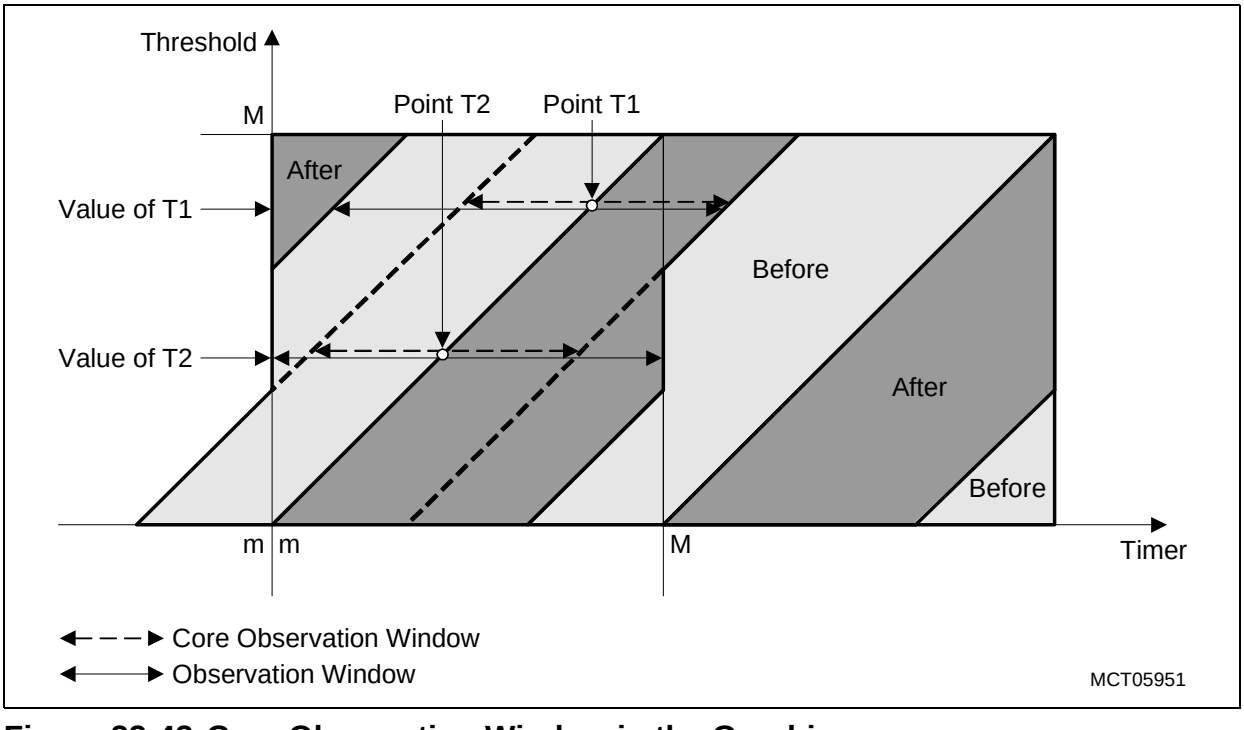


Figure 22-42 Core Observation Window in the Graphic

22.3.3.2 Global Timer Cell (GTC)

The GPTA®v5 provides 32 Global Timer Cells (GTC00 to GTC31) used for capture/compare operations.

Registers

The following registers are assigned to a GTCk (k = 00-31):

- GTCCTRk = Global Timer Cell Control Register k (see **Page 22-187**)
- GTCXRk = Global Timer Cell X Register k (see **Page 22-191**)
- SRSC1 = Service Request State Clear Register 1 (see **Page 22-226**)
- SRSS1 = Service Request State Set Register 1 (see **Page 22-227**)

Features

- **24-bit based timer cells** related to two Global Timers GT0 and GT1.
- **Capture Mode** on rising, falling or both edges with following actions:
  - Service request generation
  - Output signal transition generation (set, reset, toggle the output signal)
- **Compare Mode** on equal compare, or greater than, or equal to compare with following actions:
  - Service request generation
  - Output signal transition generation (set, reset, toggle the output signal)
  - Capture (after compare match) the value of the selected Global Timer or the opposite Global Timer
- **One Shot Mode** allows the selected (capture or compare) mode to be stopped after the first event.
- **Flexible mechanism** to link pin actions and allow complex combination of cells. (A cell has the ability to propagate actions over adjacent cells with higher number, in order to perform complex waveforms such as PWMs).

Architecture

The architecture of a GTC is shown in **Figure 22-43**. Each GTC has a multiplexer that allows selection of the GT0 or GT1 Global Timer value bus as data source, a 24-bit capture/compare register GTCXRk, and a 24-bit equal comparator.

The 32 Global Timer Cells (GTC00 to GTC31) have the following inputs:

- Two Global Timer value buses, GTV0 and GTV1, coming from the two Global Timers and carrying the GT0 and GT1 timer values
- Two inputs, TEV0 and TEV1, reporting GT0 and GT1 timer value updates
- Two inputs, TGE0 and TGE1, reporting the result of the GT0 and GT1 compare operations
- A trigger input (GTCkIN) that is connected via the GTC input multiplexer to one of the following signal sources:

  - External port lines
  - Local Timer Cell outputs
  - Filter and Prescaler Cell outputs
  - Internal input signals INTx
- Two action mode inputs (M0I, M1I) coming from the adjacent GTC with lower order number (M1I and M0I of GTC00 are 0)

Each GTC provides the following outputs:

- One data output (GTCkOUT) that can be connected to:
  - External port lines
  - Inputs of an MSC module
  - Outputs and/or inputs of Local Timer Cell inputs
- Two action mode outputs (M0O, M1O) going to the adjacent GTC with higher order number
- One service request line (SQSk) triggered by a capture/compare event.

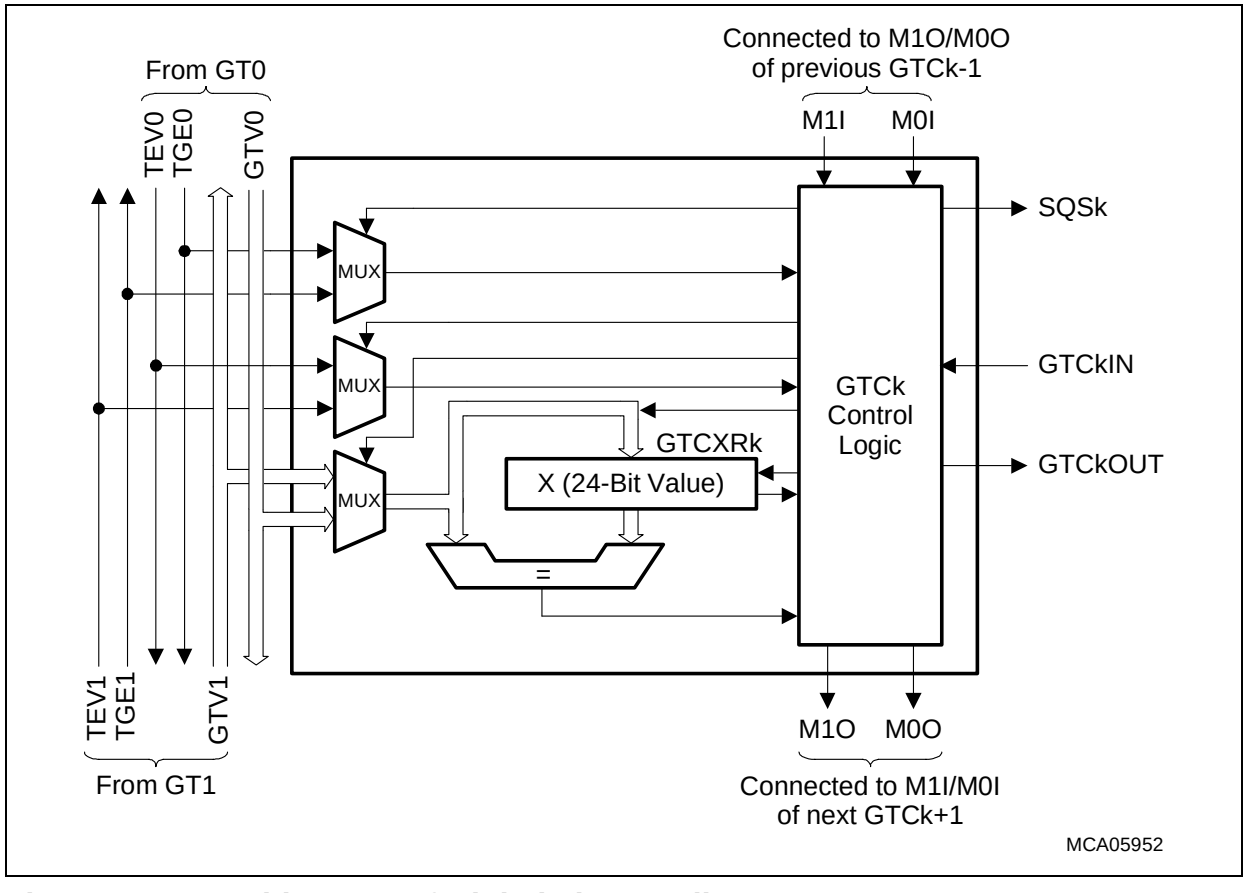


Figure 22-43 Architecture of Global Timer Cells

Figure 22-44 shows how the GTCs are arranged and connected to the adjacent GTCs and with the Global Timers GT0 and GT1.

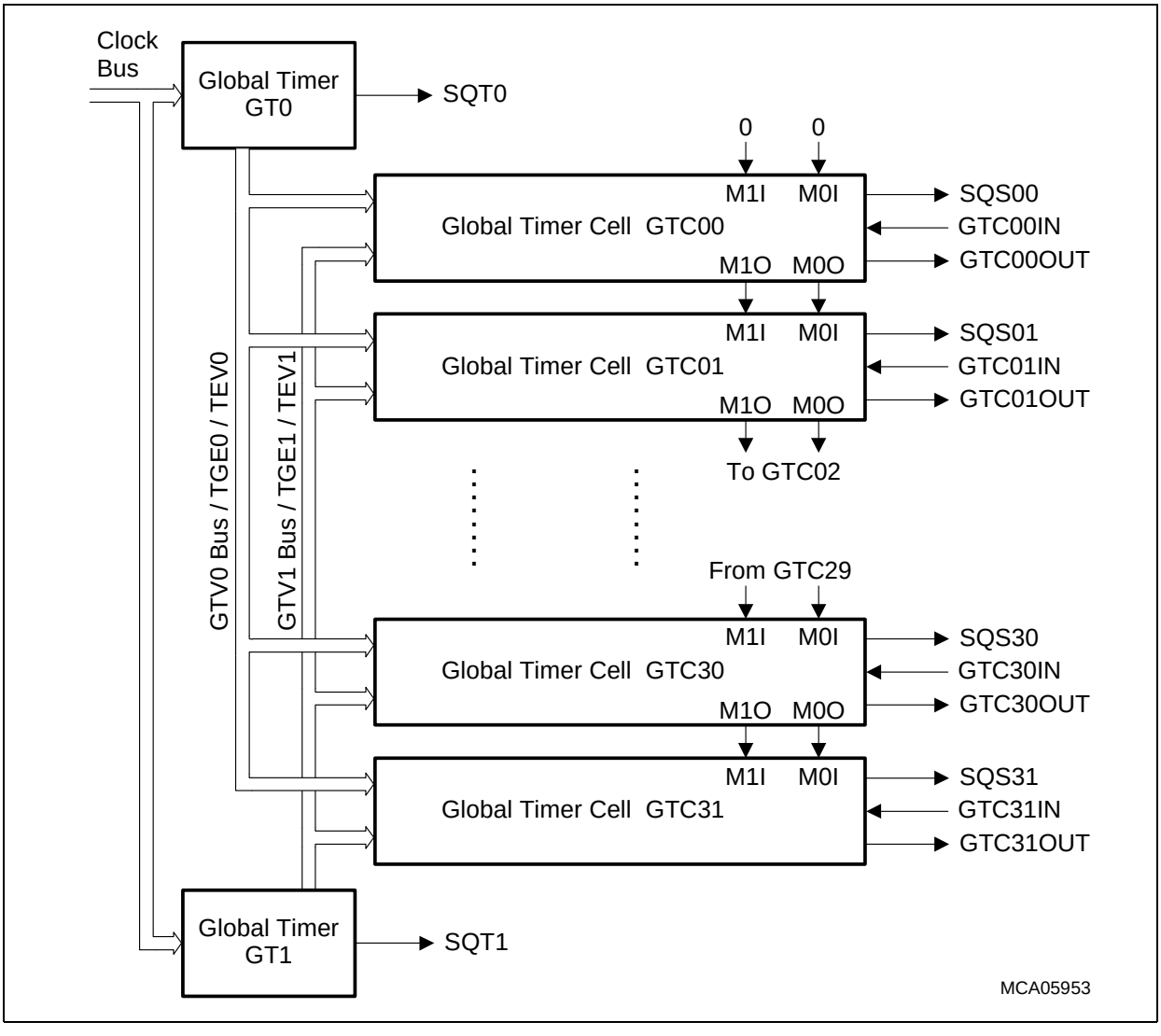


Figure 22-44 GTC Interconnections

Note: Cascading of GTCs is limited. TC1797 specific details are given on ***Page 22-313****.*

Capture Mode

The capture function of a GTCk cell is performed on a rising edge (GTCCTRk.RED = 1), a falling edge (GTCCTRk.FED = 1) or both edges of the selected GTCkIN input signal. On the requested event, the GTC:

- Copies the 24-bit value of the selected Global Timer into the 24-bit capture/compare register GTCXRk.X,
- Sets the GTCk service request flag in register SRSS1/SRSC1,
- Activates the service request output SQSk if control register bit GTCCTRk.REN = 1,
- Performs an GTCkOUT output signal line manipulation (set, reset, toggle, unchanged) as defined by bit field GTCCTRk.OCM,
- Transfers an action request, generated by an internal event or received on the M1I, M0I input lines, to the M1O, M0O output lines.

Compare Mode

In the Compare Code of a GTCk cell, several functions can be performed when the value of the selected Global Timer matches and/or exceeds the value stored in register GTCXR. With GTCCTRk.GES = 0 an “Equal Compare” match is selected while GTCCTRk.GES = 1 selects a “Greater Equal Compare” match. On the requested event, the GTC:

- Sets the GTCk service request flag in register SRSS1/SRSC1,
- Activates service request output SQSk if control register bit GTCCTRk.REN = 1,
- Performs an GTCkOUT output signal line manipulation (set, reset, toggle, unchanged) as defined by bit field GTCCTRk.OCM,
- Transfers an action request, generated by an internal event or received on the M1I, M0I input lines, to the M1O, M0O output lines.

If a greater or equal compare is selected, the condition is evaluated only when the compare value is written to the GTCXRk register. The user should then assure that the GTC is already enabled so that the evaluation can take place.

Capture after Compare Mode

When bit GTCCTRk.CAC = 1 and a compare event has occurred, register GTCXR is loaded with:

- The Global Timer value as selected by bit field GTCCTRk.MOD (GTCCTRk.CAT = 0),
- The alternate Global Timer value (GTCCTRk.CAT = 1). If a greater or equal compare match has been detected, the GTCk should be set into One Shot Mode in order to prevent double capturing.

One Shot Mode

In One Shot Mode (GTCCTRk.OSM = 1), a self-disable of GTCk is executed after each GTC event (GTCCTRk.CEN = 0). The current state of a GTCk can be evaluated by reading the control register flag bit GTCCTRk.CEN.

Note: The contents of the GTCk capture/compare register GTCXRk are write-protected for Capture_After_Compare in Single Shot Mode. Write protection is activated when the compare value is reached and released after a read access of register GTCXRk occurred.

Data Output Line Control

The data output GTCkOUT can be controlled by the GTCk itself and by adjacent GTCs with a lower order number. For this purpose, two communication signals between GTCs are available connecting all GTCs via their M1I/M0I inputs and their M1O/ M0O outputs respectively (see **Figure 22-45**).

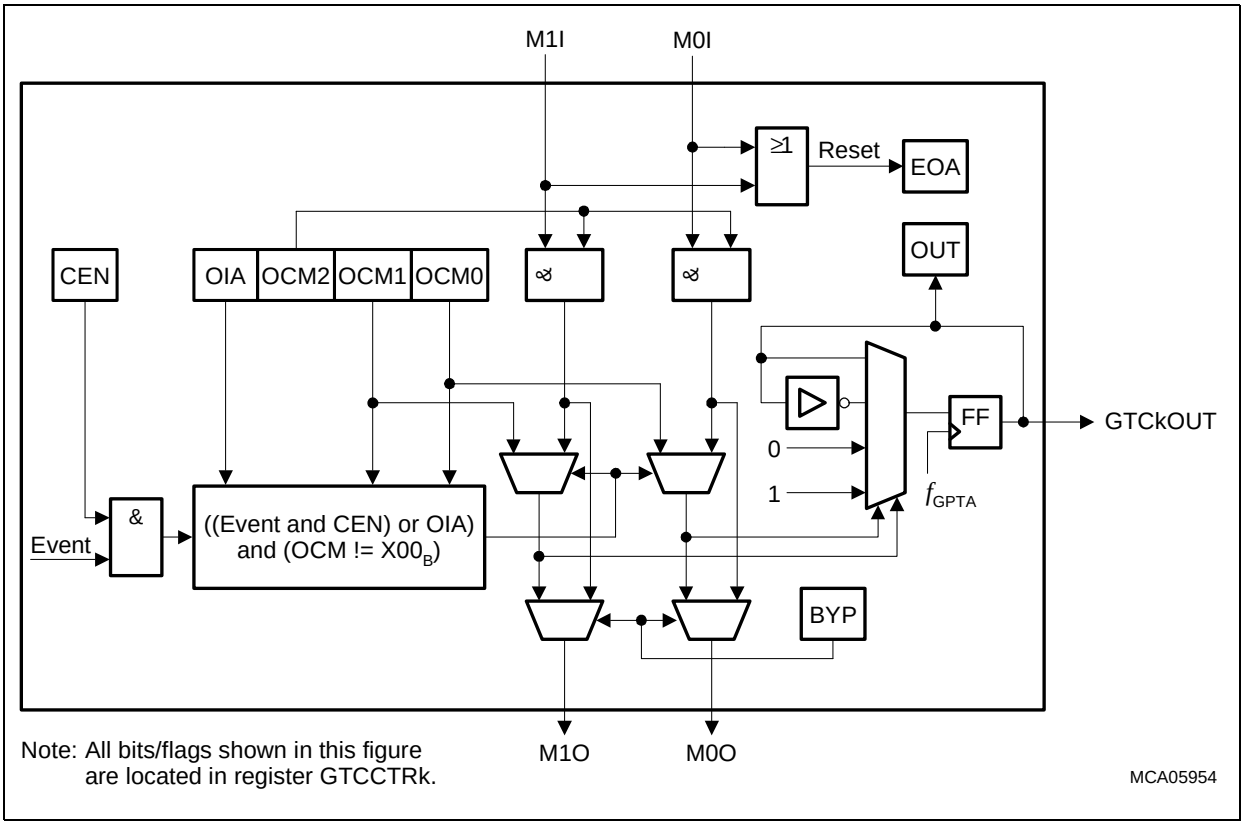


Figure 22-45 GTC Output Operation and Action Transfer

When bit GTCCTRk.OCM2 is reset, the data output GTCkOUT is only controlled by the local GTCk. A set, reset, toggle, or hold operation can be performed as selected by bits GTCCTRk.OCM1 and GTCCTRk.OCM0 (**Table 22-2**).

When bit GTCCTRk.OCM2 is set, the data output GTCkOUT is affected either by the local GTCCTRk.OCM1 and GTCCTRk.OCM0 bits or by the M1I/M0I input lines, which are connected to the adjacent GTCk-1 Global Timer output lines M1O/M0O. An enabled GTCk event superimposes an action request generated simultaneously by the M1I/M0I inputs.

When the bypass bit GTCCTRk.BYP is cleared, the M1O/M0O output lines logically OR together the local GTCk events and, if enabled by bit GTCCTRk.OCM2, the action requests received via the M1I/M0I input lines.

When bit GTCCTRk.BYP is set to 1, a local GTCk event will not modify the M1O/M0O output lines.

Table 22-2 Selection of GTC Output Operations and Action Transfer Modes

Bit Field OCM[2:0]	Local Capture or Compare Event	M1O/M0O BYP = 0	M1O/M0O BYP = 1	State of Local Data Output Line
0 0 0	not occurred occurred	0 0 0 0	0 0 0 0	not modified not modified
0 0 1	not occurred occurred	0 0 0 1	0 0 0 0	not modified inverted
0 1 0	not occurred occurred	0 0 1 0	0 0 0 0	not modified 0
0 1 1	not occurred occurred	0 0 1 1	0 0 0 0	not modified 1
1 0 0	not occurred occurred	M1I M0I M1I M0I	M1I M0I M1I M0I	modified according M1I/M0I modified according M1I/M0I
1 0 1	not occurred occurred	M1I M0I 0 1	M1I M0I M1I M0I	modified according M1I/M0I inverted
1 1 0	not occurred occurred	M1I M0I 1 0	M1I M0I M1I M0I	modified according M1I/M0I 0
1 1 1	not occurred occurred	M1I M0I 1 1	M1I M0I M1I M0I	modified according M1I/M0I 1

The GTCkOUT output line can be connected to output ports, on-chip peripheral inputs, and/or LTC inputs via the I/O Line Sharing Block (see **Page 22-98**). GTCkOUT can be updated directly by software (setting bit GTCCTRk.OIA = 1) or upon a timer, capture or compare event within the local GTCk or a preceding GTC. The current state of the data output line can be evaluated by reading status flag GTCCTRk.OUT.

Cell Enabling

After reset all GTCs are disabled. A GTC may be enabled by resetting GTCTRk.**EOA** (Enable-Of-Action) to 0 in Capture Mode or Compare Mode using a standard write assembler operation[1]. Because bit **EOA** is hardware protected, intrinsic read-modify-write assembler operations[2] only enable the GTC if bit **EOA** is modified from 1 to 0.

Cell Deactivation

By programming a GTC to Capture Mode with no edge selected (GTCCTRk.FED = GTCCTRk.RED = 0), an enabled cell becomes inactive and performs no action, but continues passing action commands via the communication link from M1I/M0I to M1O/M0O.

Cell Enabling on Event

A GTC can be enabled by an event in a GTC with lower index number. For this purpose, the local event function of an GTC must be temporary disabled by setting GTCCTRk.EOA (Enable-Of-Action) to 1. Because bit EOA is hardware protected, intrinsic read-modify-write assembler operations[2] only disable the GTC if bit EOA is modified from 0 to 1. Both operations will clear GTCCTRk.CEN and now a local event cannot affect the GTC. When a preceding GTC generates and communicates an event (or OIA) via its communication link M1O/M0O, at least one of the M1I/ M0I input lines changes its state to 1. This condition clears bit GTCCTRk.EOA of the disabled GTC via the OR gate as shown in **Figure 22-45**. Now GTCCTRk.CEN is set and the cell is enabled for local events.

It is also possible to enable the following GTC via the communication link for local events. For this purpose, the GTCCTRk.EOA bit of the following GTC must be set, too. If bit GTCCTRk.OCM2 of the preceding GTC is 1, the enable action will take place at the same time as in the preceding GTC. Otherwise, the GTC will be enabled later on a capture/compare event in the preceding GTC, provided OCM0 or OCM1 of this GTC is different from 0.

In this way, several GTCs can be enabled at the same time or one after the other. Normally, the cells will be used in One Shot Mode, and an interrupt will be generated after the last event to evaluate the data and to prepare the next enable sequence.

A disabled GTC (GTCCTRk.CEN = 0) behaves as an inactive cell.

1) Standard TriCore® write operations: ST.A, ST.B, ST.D, ST.DA, ST.DD, ST.HST.Q, ST.W
Standard PCP write operations: ST.F, ST.IF,BCOPY, COPY

2) Intrinsic TriCore® read-modify-write Operations: LDMST, ST.T, SWAP
Intrinsic PCP read-modify-write Operations: SET.F, XCH.F, CLR.F

Logical Operating Cells

The inter-cell communication architecture allows implementation of a complex waveform generation to be distributed over several GTCs, controlling a common port pin.

For example, one GTC may be configured in Capture Mode triggered by a rising edge detected on the associated input pin line. The related interrupt service routine can increment the captured timer value by a delay offset and store the result in the GTCXR register of the adjacent GTC configured in Compare Mode. Upon a compare event in the second GTC, the output port line of a third GTC can be set via M1O, M0O interface lines. When the GTCXR register of the third cell is loaded with another compare value by the interrupt service routine related to the second GTC, the output port line may be reset by the next compare event within GTC3.

This logical operating cell provides an output signal with programmable pulse width and configurable delay with minimal software overhead.

GTC Service Request

The service request output SQSk of a Global Timer Cell GTCk is controlled as shown in **Figure 22-46**. When the GTCk service request condition becomes active, the service request flag always becomes set. The service request output SQSk is only activated if it is enabled by the enable bit GTCCTRk.REN. Additional information about service request and interrupt handling is given on **Page 22-123**.

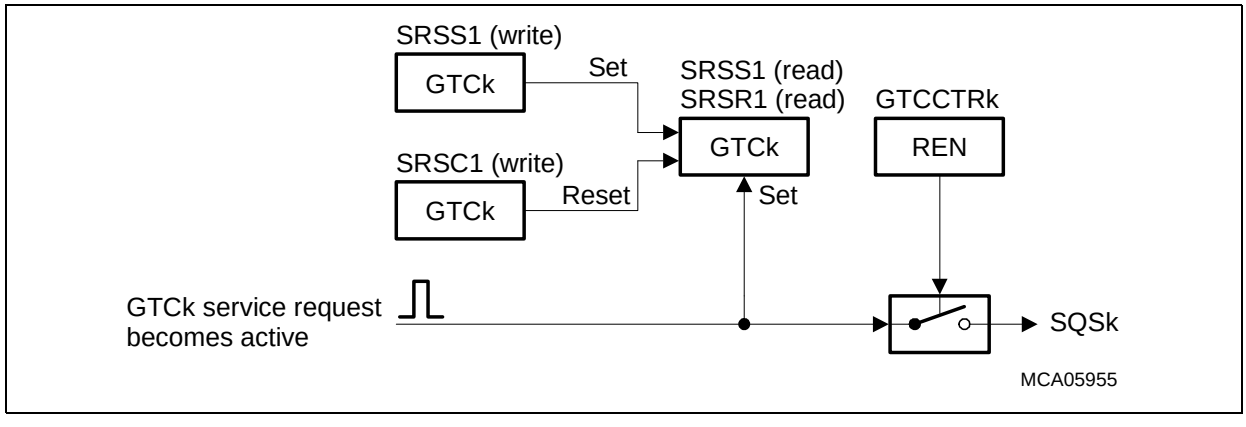


Figure 22-46 GTCk Service Request Generation

GTC Application Examples

The Global Timers together with GTCs can typically be used for input signal timing analysis of very complex input signals as well as for generation of complex output signals. **Figure 22-47** shows a configuration with Global Timer 0 and four GTCs, which is used in the following two examples:

- Example 1: Complex input signal capturing and analyzing
- Example 2: Complex periodical output signal generation

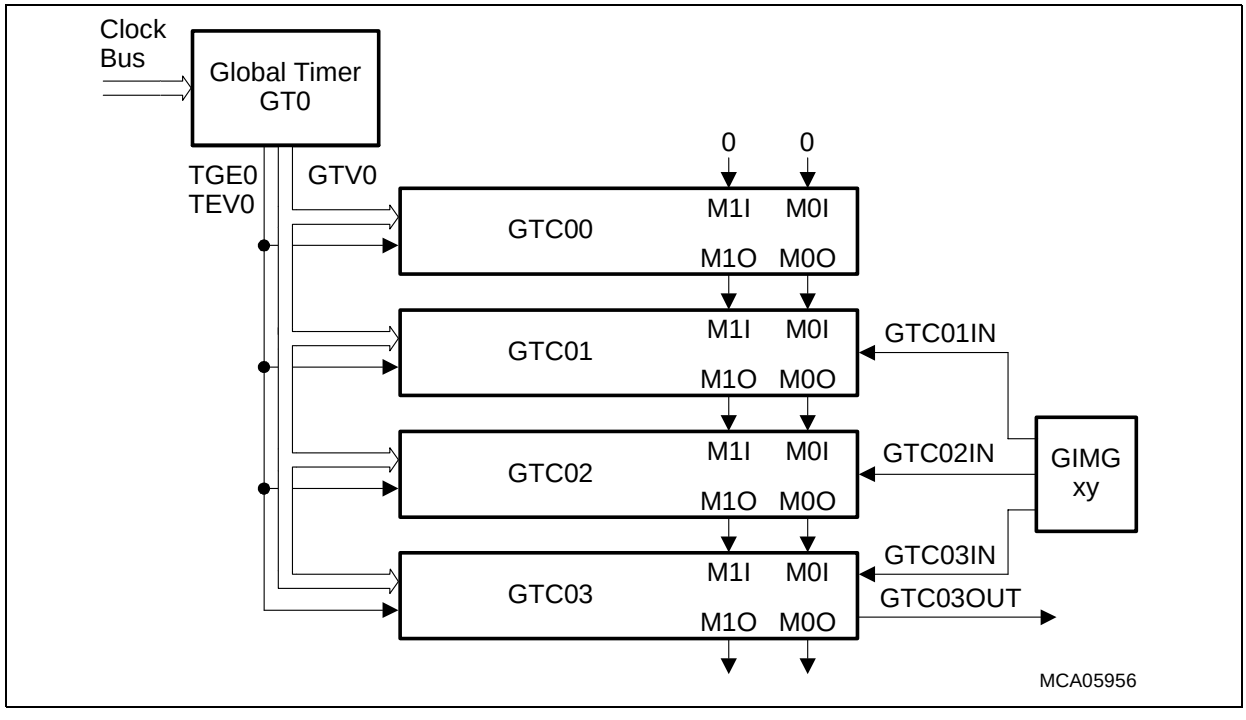


Figure 22-47 Complex Input/Output Signal Capturing/Generation with GTs and GTCs

Complex Input Signal Capturing and Analyzing

In this application example, one input signal from a GTC input multiplexer group becomes analyzed from a timing reference point for three consecutive signal transitions. This common input signal (e.g. a port line) is selected by a GTC input multiplexer group (GIMG) common for GTC01, GTC02, and GTC03 (see also **Page 22-111**). The GTCs are configured in the following way:

- GT0 operates as free-running up-counting 24-bit timer with reload to GTREV0.REV on overflow. It is clocked by one clock signal from the clock bus.
- GTC00 operates in Compare Mode with timer GT0. The compare match event is reported on the M0O/M1O output lines to the GTC01.
- GTC01 operates in Capture Mode at **rising** edge with enable-on-action set (EOA set) and One Shot Mode enabled (OSM set).

- GTC02 operates in Capture Mode at **falling** edge with enable-on-action set (EOA set) and One Shot Mode enabled (OSM set).
- GTC03 operates in Capture Mode at **rising** edge with enable-on-action set (EOA set) and One Shot Mode enabled (OSM set).

With the compare event of GTC00 (time stamp), GTC01 becomes active and waits for the next rising edge at its data input GTC01IN. While GTC01 is active, GTC02 and GTC03 are inactive.

When GTC01 detects a rising edge at its data input, it captures the current GT0 value into its GTCXR01 register, enables GTC02, and becomes disabled afterwards because it was operating in One Shot Mode. When GTC02 detects a falling edge at its data input, it captures the current GT0 value into its GTCXR02 register, enables GTC03, and becomes disabled afterwards because it was operating in One Shot Mode. When GTC03 detects a rising edge at its data input, it captures the current GT0 value into its GTCXR03 register and becomes disabled afterwards because it was operating in One Shot Mode. Optionally, the capture event at GTC03 may generate a service request to indicate that the three capture events have occurred and the captured values can be checked by software. Note that all of these capture events are executed by the GPTA®v5 hardware without any software interactions and with a resolution of the GT0 clock rate.

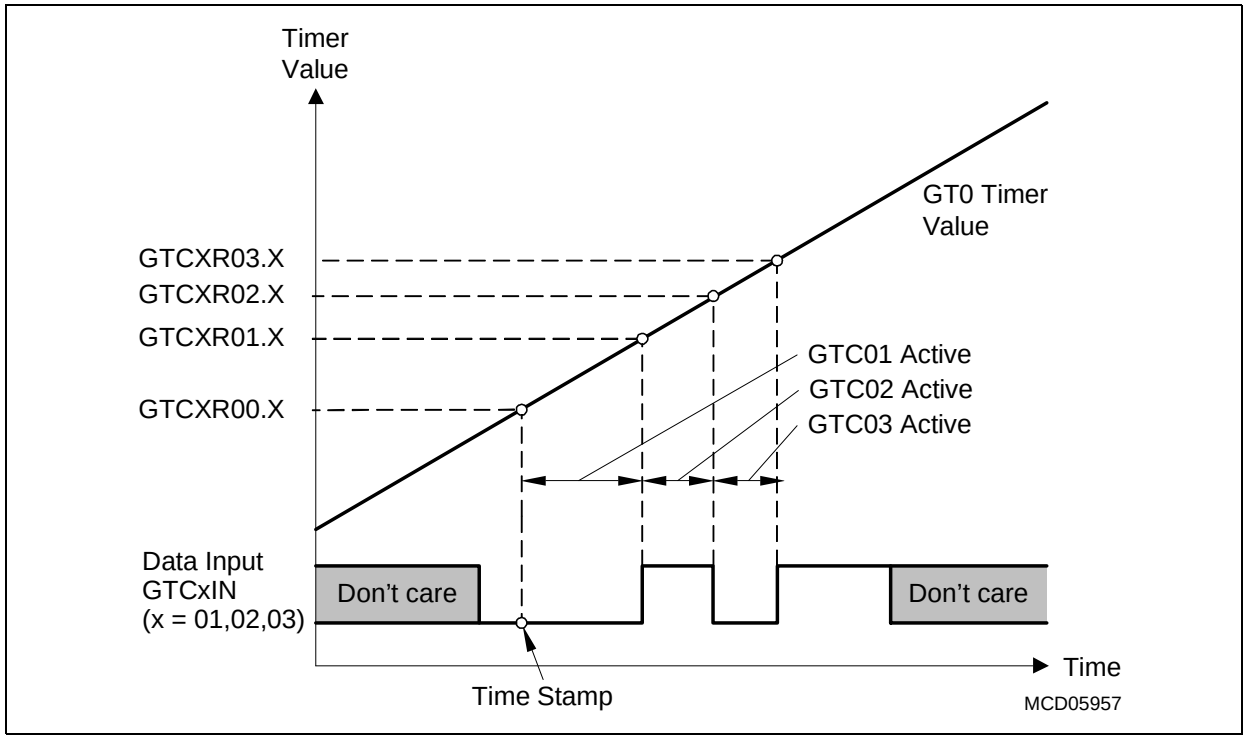


Figure 22-48 Complex Input Signal Analysis/Capturing with GTCs

Complex Periodic Output Signal Generation

This application example uses the GT/GTC configuration as shown in **Figure 22-47**. The generated output signal is available at GTC03OUT. The GTC input signals of the GT/GTC configuration are not used in this example.

- GT0 operates as free-running up-counting 24-bit timer with reload to GTREV0.REV on overflow. It is clocked by a clock signal from the clock bus. Its reload period determines the period of the generated PWM output signal.
- GTC00 operates in Compare Mode with timer GT0 with enable-on-action set (EOA set) and One Shot Mode enabled (OSM set). Furthermore, the output GTC03OUT becomes **set** on a local event (OCM = $X11_B$).
- GTC01 operates in Compare Mode with timer GT0 with enable-on-action set (EOA set) and One Shot Mode enabled (OSM set). Furthermore, the output GTC03OUT becomes **reset** on a local event (OCM = 110_B).
- GTC02 operates in Compare Mode with timer GT0 with enable-on-action set (EOA set) and One Shot Mode enabled (OSM set). Furthermore, the output GTC03OUT becomes **set** on a local event (OCM = 111_B).
- GTC03 operates in Compare Mode with timer GT0 with enable-on-action set (EOA set) and One Shot Mode enabled (OSM set). Furthermore, the output GTC03OUT becomes **reset** on a local event (OCM = 110_B).

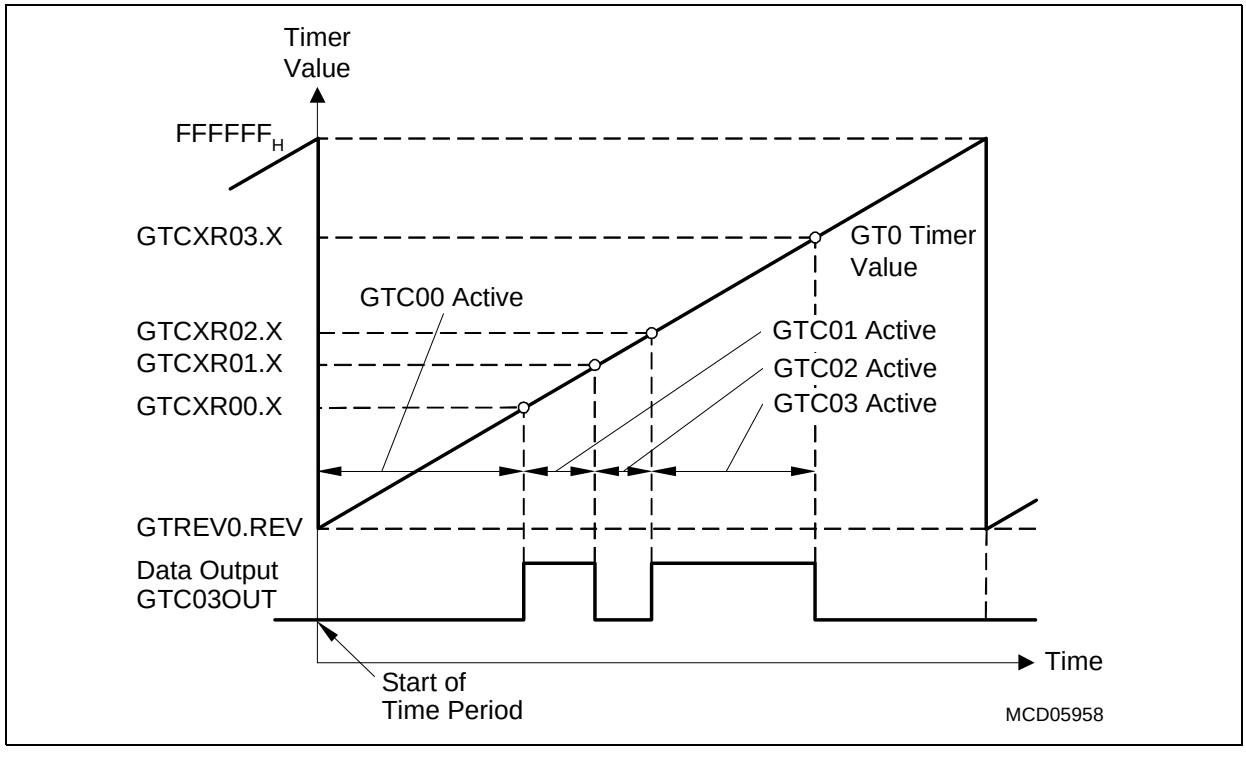


Figure 22-49 Complex Output Signal Generation with GTCs

At the start of the time period (reload of GT0), GTC00 becomes active and waits for the compare event. At this event, it sets the output signal GTC03OUT, enables GTC01 for

compare operation, and becomes disabled afterwards because it was operating in One Shot Mode. When the GTC01 compare event occurs, the output signal GTC03OUT is reset, GTC02 becomes enabled, and GTC01becomes disabled because it was operating in One Shot Mode. When the GTC02 compare event occurs, the output signal GTC03OUT is set, GTC03 becomes enabled, and GTC02 becomes disabled because it was operating in One Shot Mode. When the GTC03 compare event occurs, the output signal GTC03OUT is reset, and GTC02 becomes disabled because it was operating in One Shot Mode.

The capture event at GTC03 should generate a service request to indicate that the three compare events have occurred and that GTC01 can be enabled again (setting EOA and OSM). Note that all of the compare events are executed by the GPTA®v5 hardware without any software interactions and with a resolution of the GT0 clock rate.

22.3.3.3 Local Timer Cell (LTC00 to LTC62)

LTC00 to LTC62 are functionally identical. The functionality of LTC63 is different to LTC00 to LTC62 and therefore described separately at **Page 22-79**.

Registers

The following registers are assigned to a Local Timer Cell LTCk (k = 00-62):

- LTCCTRk = Local Timer Cell Control Register k (see **Page 22-192**)
- LTCXRk = Local Timer Cell X Register k (see **Page 22-205**)
- SRSC2 = Service Request State Clear Register 2 (see **Page 22-228**)
- SRSC3 = Service Request State Clear Register 3 (see **Page 22-230**)
- SRSS2 = Service Request State Set Register 2 (see **Page 22-229**)
- SRSS3 = Service Request State Set Register 3 (see **Page 22-231**)

Features

- **16-bit based timer cells** providing capture, compare, and timer functions.
- **Capture Mode** on rising, falling or both edges with following actions:
  - Service request generation
  - Output signal transition generation (set, reset, toggle the output signal).
- **Compare Mode** on equal compare of the corresponding (Reset-)Timer LTC with following actions:
  - Service request generation
  - Output signal transition generation (set, reset, toggle the output signal).
- **Timer Mode** incremented on hardware signal with following actions:
  - Event generation at overflow
  - Service request generation
  - Output signal transition generation (set, reset, toggle the output signal).
- **Reset Timer Mode** allows the selected LTC to be reset by an adjacent cell. Coherent update capability of adjacent LTCs for PWM management is provided.
- **One Shot Mode** allows the selected (capture, compare, timer or reset timer) mode to be stopped after the first event.
- **Flexible mechanism** to link pin actions and allow complex combination of cells. (A cell has the ability to propagate actions over adjacent cells with higher number, in order to perform complex waveforms such as multi channel PWMs).

Architecture

The architecture of an LTC is shown in **Figure 22-50**. Each LTC has a 16-bit capture/compare register and a 16-bit equal to comparator.

The first 63 Local Timer Cells (LTC00 to LTC62) have the following inputs:

- A local input data bus (YI) carrying the local timer value of the adjacent LTC with lower order number (YI of LTC00 is always 0000_H)

- A TI input reporting the occurrence of a local timer value update of the adjacent LTC with lower order number (TI of LTC00 is 0)
- A SI input used by the LTC in Compare Mode as enable line (SI of LTC00 is 0)
- Four action mode inputs (M3I, M2I, M1I, M0I) coming from the adjacent LTC cell with lower order number (M3I, M2I, M1I, and M0I of LTC00 are 0)
- An EI input reporting an event coming from the adjacent LTC with higher order number
- A trigger/clock/enable input LTCkIN hooked to one of the following signals sources:
  – External port lines
  – GTC00 to GTC31 outputs
  – Clock bus signals
  – PDL0 or PDL1 outputs
  – Internal GPTA®v5 kernel input signals INTx (x = 0-3)

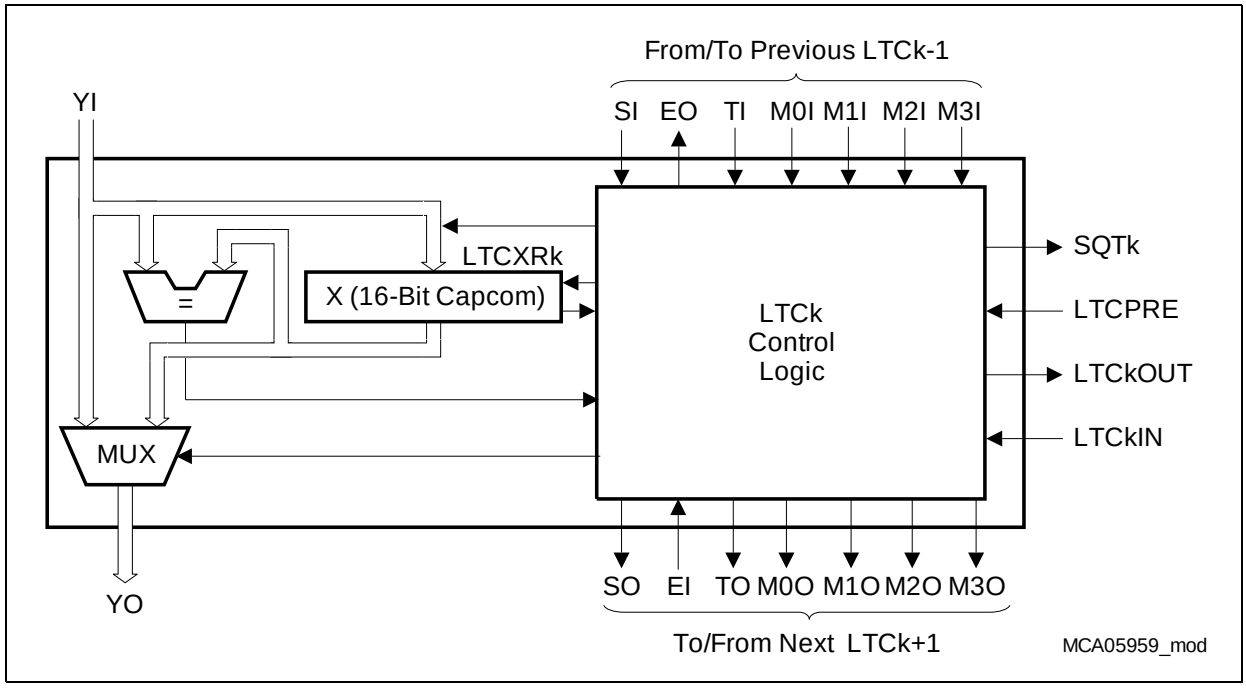


Figure 22-50 Architecture of Local Timer Cells

Each LTC provides the following input / output signals:

- One data output line (LTCkOUT) that can be connected to:
  – External port lines
  – Inputs of an MSC module
  – Outputs and/or inputs of Global Timer Cell inputs
- One LTC prescaler clock input (LTCPRE) for Timer Mode
- One service request line (SQT) triggered by a capture/compare event
- A local output data bus (YO) carrying the local timer value to the adjacent LTC with higher order number or the value on YI

- A TO output reporting the occurrence of a local timer value update to the adjacent LTC with higher order number
- An SO output used by the adjacent LTC with higher order number as enable signal for a compare function
- An EO output reporting the occurrence of a local event to the adjacent LTC with lower order number
- Four mode lines (M3O, M2O, M1O, M0O) going to the adjacent LTC with higher order number.

Figure 22-51 shows the arrangement of the LTCs and the connections with adjacent LTCs. LTC63 is a Local Timer Cell that differs from all other LTCs (LTC00 to LTC62). LTC63 is described in detail on **Page 22-79**.

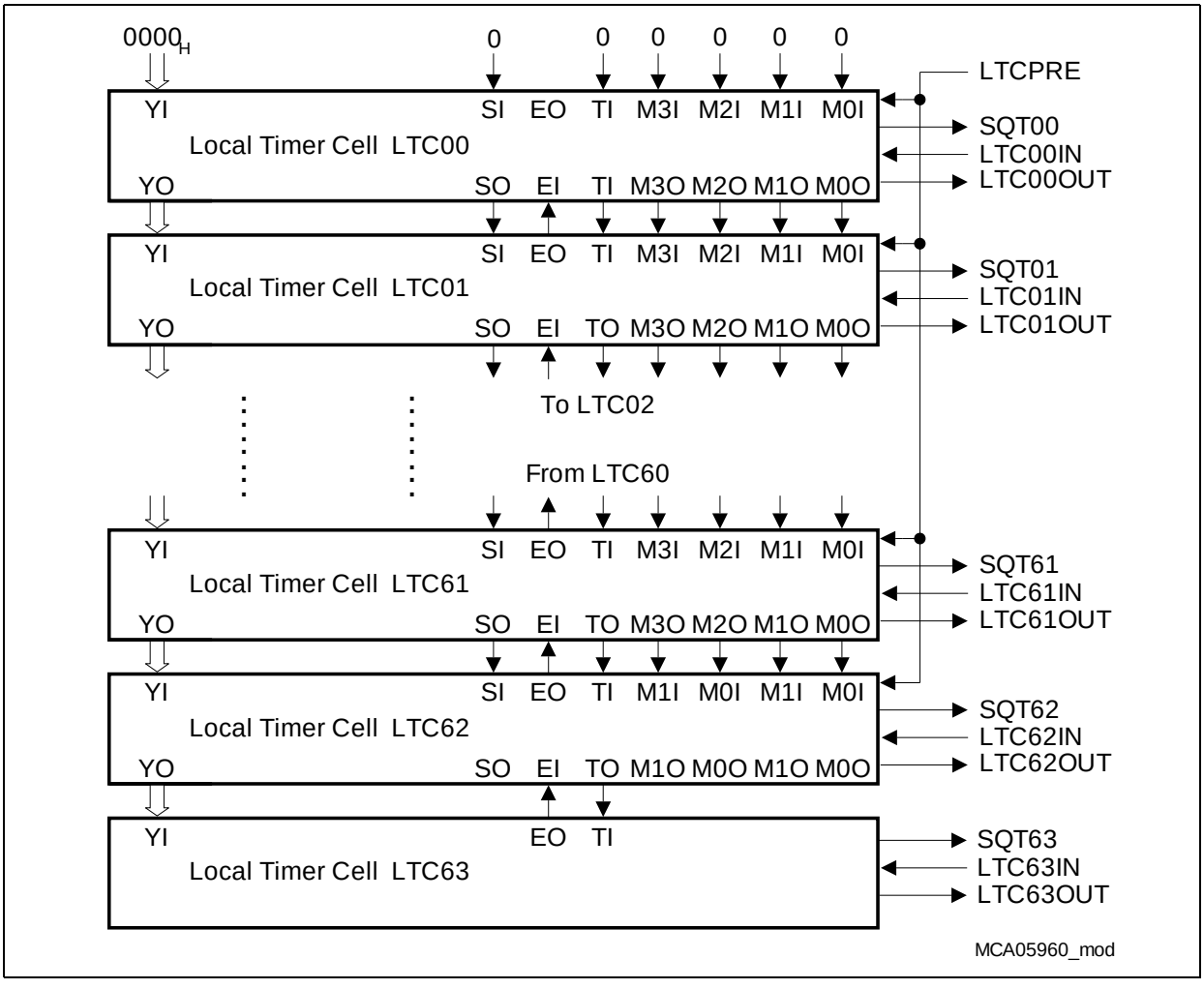


Figure 22-51 Interconnections between the LTCs

*Note: Cascading of LTCs is limited. TC1797 specific details are given on **Page 22-313**.*

Operating Mode Selection

The operating mode of an LTC – Free-Running Timer, Reset Timer, Capture, or Compare Mode – is defined by bit field LTCCTRk.MOD.

Free-Running Timer Mode

The content of the Local Timer Cell Register LTCXRk is initialized by a software write operation. LTCXRk is incremented by the selected LTCkIN input signal. Level or Edge Sensitive Mode can be selected for LTCkIN (see **Page 22-72**). In Level Sensitive Mode prescaler clock from the CDC (LTCPRE) can be used to reduce the timer frequency. Every change of the Local Timer Cell Register LTCXRk (increment, reset, or write access) is indicated by output signal TO = 1. When the timer reaches its overflow value ($FFFF_H$),

- the LTCk service request flag is set,
- the service request output SQTk is activated if control register bit LTCCTRk.REN = 1,
- the LTCk output line LTCkOUT can be altered (set, reset, toggle, unchanged),
- the LTCkOUT output line can be altered (set, reset, toggle, unchanged), depending on bit field LTCCTRk.OCM,
- an action request, generated by an LTCk internal event or received on the M1I/M0I input lines, is transferred via the M1O/M0O output lines to the LTC with higher order number (LTCk+1).

The event output line EO is also activated (set to high) by a software reset when writing $FFFF_H$ to register LTCXRk.

Reset Timer Mode

An LTC that is configured in Reset Timer Mode provides the same functionality as in Free-Running Timer Mode, but is extended by two additional features:

- The Local Timer Cell Register LTCXRk can be reset to $FFFF_H$ via the EI line, which can be activated by an event that occurred in the adjacent LTC with higher order number.
- If bit LTCCTRk.CUD is set to 1, the EI line reset event also toggles the logic state of the SO output line before it clears register bit LTCCTRk.CUD automatically. By accessing register bit LTCCTRk.SLO, the state of the timer's output line SO can be read or explicitly written.

Capture Mode

In Capture Mode, the LTCkIN input signal is used for capture function. Level or edge Sensitive Modes can be selected (see **Page 22-72**). On a capture event, the LTCk:

- copies the state of the local input data bus (YI) to the LTCXRk register (LTC00 always copies 0000_H),
- sets the LTCk service request flag,

- activates the service request line SQTk, if LTCCTRk.REN is set to 1,
- changes the LTCkOUT output line state (set, reset, toggle, unchanged), depending on bit field LTCCTRk.OCM and the M1I/ M0I input line state,
- generates and/or passes an action request via the M1O/M0O output lines to the LTC with higher order number (LTCk+1),
- sets the event output EO to high level for one f_{GPTA} clock cycle.

Compare Mode

The Compare Mode can be enabled on a low, high, or both levels of the select input line SI (LTCCTRk.SOL = 1, LTCCTRk.SOH = 1). The current state of SI is indicated by bit field LTCCTRk.SLL and can be read. When the value of the local input data bus (YI) matches the LTCXRk contents,

- The LTCk service request flag is set,
- The service request line SQTk is activated if LTCCTRk.REN is set to 1,
- The LTCkOUT output line state is changed (set, reset, toggle, unchanged), depending on bit field LTCCTRk.OCM,
- An action request is generated and/or passed via the M1O/M0O output lines to the LTC with higher order number (LTCk+1),
- The event output EO is set to high level for one f_{GPTA} clock cycle.

Note: To enable the compare function in all cases (on every timer or compare register update caused by a software write access, a reset event or a compare match), bits LTCCTRk.SOL and LTCCTRk.SOH must be set to 1.

An inactive cell (LTCCTRk.SOL = LTCCTRk.SOH = 0, or SI does not match the programmed value) will transfer the state of the event input line EI to the event output line EO.

One Shot Operation

When bit LTCCTRk.OSM is set to 1, a self-disable is executed after each LTC event. The current state of LTCk can be checked by reading the control register flag bit LTCCTRk.CEN.

Data Input Line Control

The data input line LTCkIN can operate in two modes (selected by bit LTCCTRk.ILM):

- Level Sensitive or
- Edge Sensitive.

In Edge Sensitive Mode, the active edges are selected by bits LTCCTRk.FED and LTCCTRk.RED. For the Level Sensitive Mode, the active level of the input signal can be selected by bit FED/AIL of register LTCCTRk.

Depending on which source is selected for the input line by the input multiplexer, different clocking modes of the LTC cell are possible (**Table 22-3**).

Table 22-3 LTC Data Input Line Operation (in Timer Mode)

Input Source	Level Sensitive Input Line LTCCTRk.ILM = 1	Edge Sensitive Input Line LTCCTRk.ILM = 0
External Signal (Port line)	The external signal operates as gating signal for the cell. The active input level can be selected with control register bit AIL. Additionally, the LTC prescaler mode can be enabled with LTCCTRk.PEN to reduce the timer frequency. The programmed function of the LTC is performed with the GPTA®v5 module clock frequency, or with the programmed prescaler clock LTCPRE (see **Page 22-37**).	The programmed function of the LTC cell is performed on selected edge(s).
Internal Clock Bus Line or PDL output or INT input	The programmed function is performed with the internal clock or PDL/INT signal. Note that all internal clock bus lines and PDL signals are active high pulses. The LTC Prescaler Mode and the input signal inversion must not be used.	The programmed function of the LTC cell is performed on selected edge(s). In case of full speed GPTA®v5 module clock selection as input clock, the Level Sensitive Mode must be selected. The Edge Sensitive Mode will not produce an event in this special case.

Table 22-3 LTC Data Input Line Operation (in Timer Mode) (cont'd)

Input Source	Level Sensitive Input Line LTCCTRk.ILM = 1	Edge Sensitive Input Line LTCCTRk.ILM = 0
GTC output	The GTC output signal operates as gating signal for the cell. The active input level can be selected with bit LTCCTRk.AIL. Additionally, the LTC prescaler clock LTCPRE can be enabled with bit LTCCTRk.PEN to reduce the timer frequency.	The programmed function of the LTC cell is performed on selected edge(s).

Note: If Capture Mode and level sensitive input is selected for an LTCk (bit LTCCTRk.ILM = 1), a capture event occurs on every LTC timer clock event if the corresponding LTC input signal is a high level.

Data Output Line Control

The data output LTCkOUT can be controlled by the LTCk itself and by adjacent LTCs with a lower order number. For this purpose, two communication signals between LTCs are available that make it possible to connect all LTCs via their M1I/M0I inputs and their M1O/ M0O outputs respectively (**Figure 22-52**).

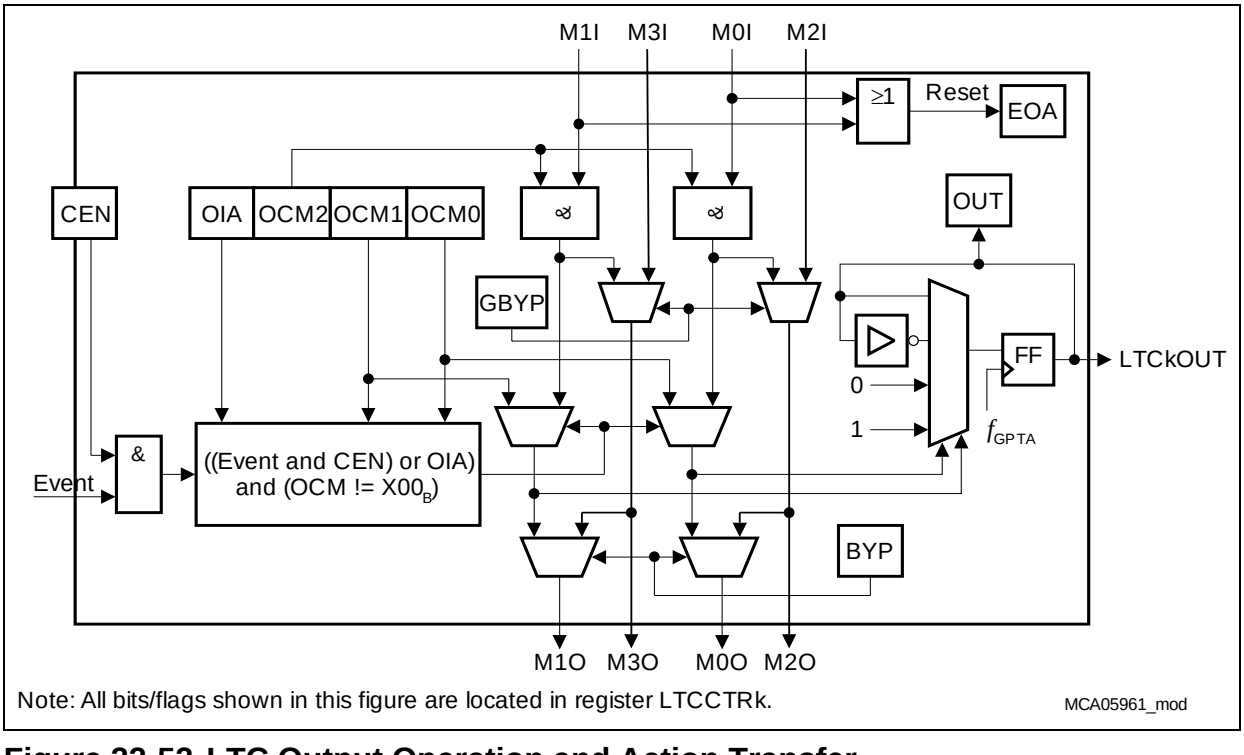


Figure 22-52 LTC Output Operation and Action Transfer

When bit LTCCTRk.OCM2 is reset, the data output LTCkOUT is controlled only by the local LTCk. A set, reset, toggle, or hold operation can be performed as selected by bits LTCCTRk.OCM1 and LTCCTRk.OCM0 (see **Table 22-4**).

When bit LTCCTRk.OCM2 is set, the data output LTCkOUT is affected either by the local LTCCTRk.OCM1 and LTCCTRk.OCM0 bits or by the M1I/M0I input lines, which are connected to the adjacent LTCk-1 Global Timer output lines M1O/M0O. An enabled LTCk event superimposes an action request generated simultaneously by the M1I/M0I inputs.

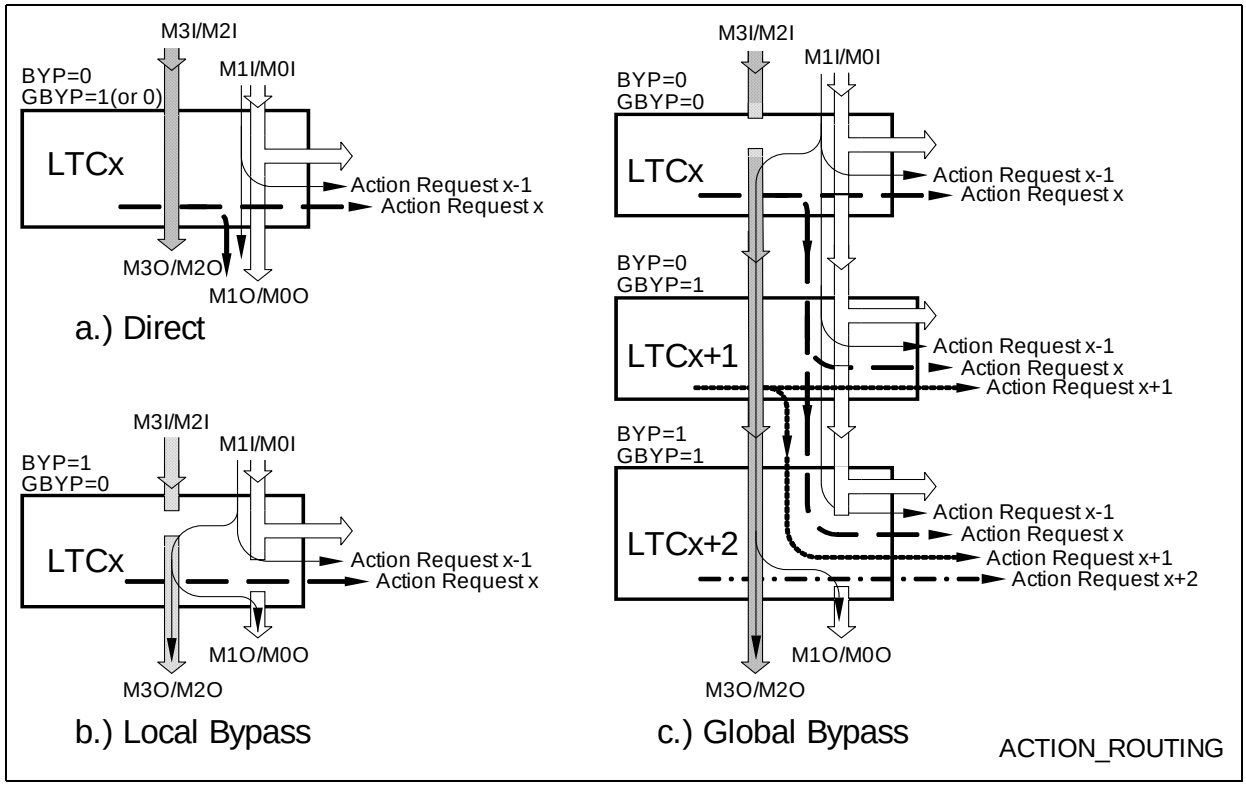


Figure 22-53 Direct, Local Bypass, and Global Bypass Action Request Routing

When the bypass bit LTCCTRk.BYP is cleared, the M1O/M0O output lines logically OR together the local LTCk events and, if enabled by bit LTCCTRk.OCM2, the action requests received via the M1I/M0I input lines.

When the bypass bit LTCCTRk.GBYP is cleared, the action requests received via M1I/M0I input lines, if enabled by bit LTCCTRk.OCM2, are forwarded to the subsequent LTCk+1 via the M3O/M2O output lines. If LTCCTRk.GBYP is set to 1, the action requests received via M3I/M2I input lines are forwarded to the subsequent LTCk+1 via the M3O/M2O output lines. Therefore the M3I/M2I may be used to pass the action requests of a reseted timer cross a group of LTC generating a complex signal or providing coherent update.

The two bypass bit LTCCTRk.BYP and LTCCTRk.GBYP enable three different types of action request routing, a direct routing, a local bypass routing and a global bypass

routing. These three different types of action request routing is sketched in **Figure 22-53**. The direct routing uses only the action request lines M1I/M0I. All action request run done the same path. The M3I/M2I are forwarded unchanged to M3O/M2O (next cells).

The local bypass copies all action request coming into a cell (M1I/M0I) to the outputs of the cell (M1O/M0O and M3O/M2O. The following cells do not see any action request generated within this locally bypassed cell.

The global bypass routing copies all action request coming into a group of sequential cells (M1I/M0I) to the outputs of this group of sequential cells (M1O/M0O and M3O/M2O. The following cells do not see any action request generated within this group of cells. Typically for edge aligned signals, the one action request for the edge is globally or locally bypassed, an all non edge aligned action request then locally generated within the group or local bypassed cell. Within a group complex signals may be generated or two cells used for local coherent update (double action principle).

Table 22-4 Selection of LTC Output Operations and Action Transfer Modes

Bit Field OCM [2:0]	Local Event: Overflow, Capture or Compare	M1O/M0O						State of Local Data Output Line
		BYP = 0		BYP = 1				
				GBYP = 0		GBYP = 1		
0 0 0	not occurred occurred	0 0	0 0	0 0	0 0	0 0	0 0	not modified not modified
0 0 1	not occurred occurred	0 0	0 1	0 0	0 0	0 0	0 0	not modified inverted
0 1 0	not occurred occurred	0 1	0 0	0 0	0 0	0 0	0 0	not modified 0
0 1 1	not occurred occurred	0 1	0 1	0 0	0 0	0 0	0 0	not modified 1
1 0 0	not occurred occurred	M1I M1I	M0I M0I	M1I M1I	M0I M0I	M3I M3I	M2I M2I	modified according M1I/M0I modified according M1I/M0I
1 0 1	not occurred occurred	M1I 0	M0I 1	M1I M1I	M0I M0I	M3I M1I	M2I M2I	modified according M1I/M0I inverted
1 1 0	not occurred occurred	M1I 1	M0I 0	M1I M1I	M0I M0I	M3I M3I	M2I M2I	modified according M1I/M0I 0
1 1 1	not occurred occurred	M1I 1	M0I 1	M1I M1I	M0I M0I	M3I M3I	M2I M2I	modified according M1I/M0I 1
		1)		M3O/M2O = M1I/M0I		M3O/M2O = M3I/M2I		

1) If GBYP = 0: M3O/M2O = M1I/M0I
If GBYP = 1: M3O/M2O = M3I/M2I

The LTCkOUT output line can be connected to output ports, on-chip peripheral inputs (OTGS) and/or LTC inputs via the I/O Line Sharing Block (see **Page 22-98**). LTCkOUT can be updated directly by software (setting bit LTCCTRk.OIA = 1) or upon a timer, capture, or compare event within the local LTCk or a preceding LTC. The current state of the data output line can be evaluated by reading status flag LTCCTRk.OUT.

Global bypass may also be used to move a group of Local Timer Cells (Consecutive Local Timer Cells using the same Local Timer) into the previous group of Local Timer to e.g. hit a specific output pin. The previous Local Timer Group will therefore have some Local Timer before and some after the to be moved Local Timer Group. Because the Local Time Bus YI and YO is driven by every LTC configured as timer regardless of the chosen bypass mechanism, special care has to be taken. An example is sketched in **Figure 22-54**. Three Local Timer for an edge aligned PWM, using the same time base as the local Timer Cell Group1 but different offsets (not edge aligned), is inserted into another group of Local Timer Cells implementing an independent other signal.

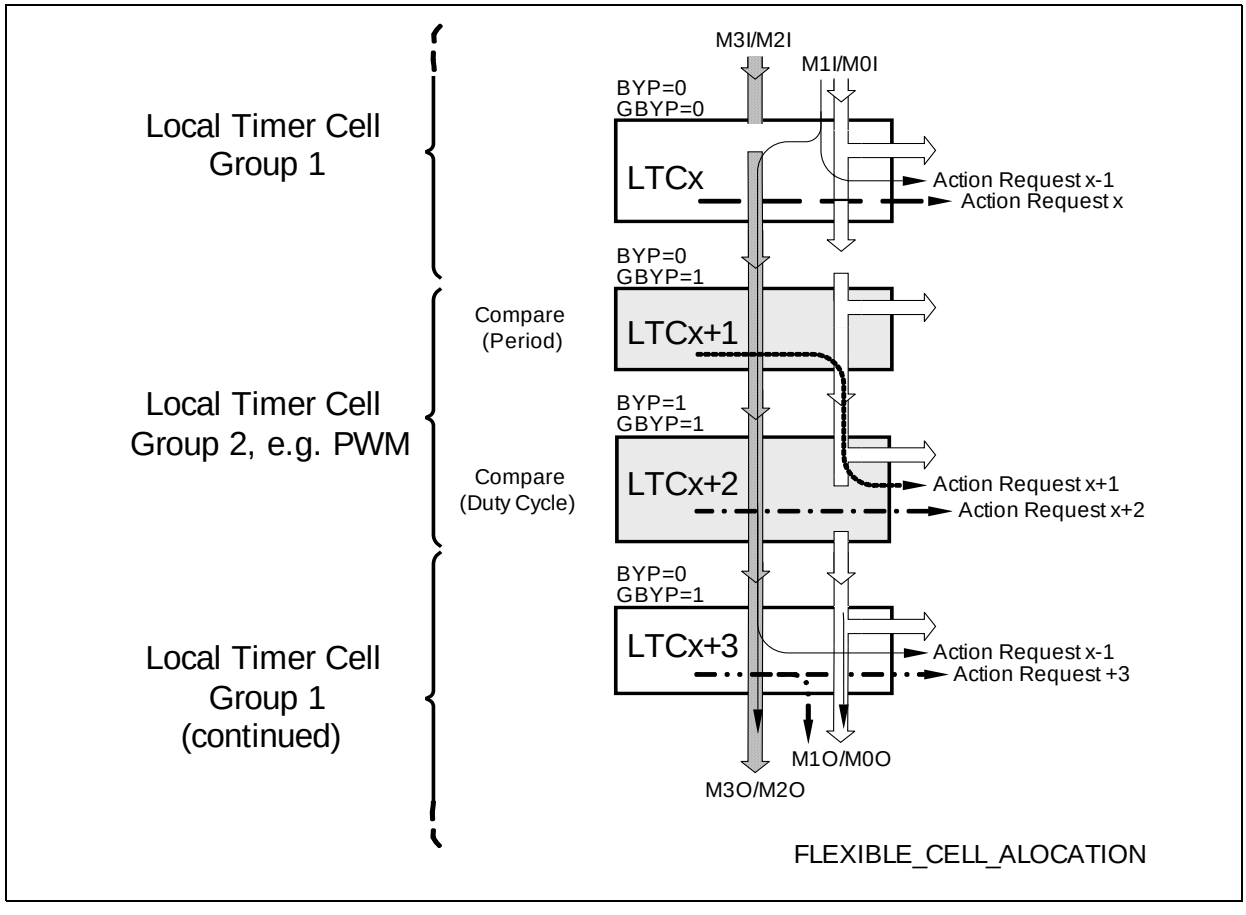


Figure 22-54 Global Bypass Action Request Routing for Flexible Cell Allocation

Cell Enabling

After reset all LTCs are disabled. An LTC may be enabled by resetting LTCCTRk.**EOA** (Enable-Of-Action) to 0 in Capture Mode or Compare Mode using a standard write assembler operation[1]. Because bit **EOA** is hardware protected, intrinsic read-modify-write assembler operations[2] only enable the LTC in Capture Mode or Compare Mode if bit **EOA** is modified from 1 to 0. If switching to Timer Mode, the LTC cell is enabled. In Timer Mode every write operation to bit 0…7 of LTCCTRk will enable the LTC.

Cell Deactivation

By programming an LTC to Capture Mode with no edge selected (LTCCTRk.ILM = LTCCTRk.FED = LTCCTRk.RED = 0), an enabled cell becomes inactive and performs no action, but continues passing action commands via the communication link from M1I/M0I to M1O/M0O. Output EO is inactive.

Alternatively, the LTC can be deactivated by setting it into Compare Mode with no active select line level (LTCCTRk.SOL = LTCCTRk.SOH = 0) but the communication link remains active. In this mode configuration, EI will be passed to EO.

Cell Enabling on Event

An LTC can be enabled in Capture Mode or Compare Mode by an event in an LTC with lower index number. For this purpose, the local event function of an LTC must be temporary disabled by setting LTCCTRk.EOA (Enable-Of-Action) to 1. Because bit EOA is hardware protected, intrinsic read-modify-write assembler operations[2] only disables the LTC if bit EOA is modified from 0 to 1. Both operations will clear LTCCTRk.CEN and now a local event cannot affect the LTC. When a preceding LTC generates and communicates an event (or OIA) via the communication link M1O/M0O, at least one of the M1I/M0I input lines changes its state to 1. This condition clears bit LTCCTRk.EOA of the disabled LTC via the OR gate as shown in **Figure 22-52**. Now LTCCTRk.CEN is set and the LTC is enabled for local events.

It is also possible to enable the following LTC via the communication link for local events. For this purpose, the bit LTCCTRk.EOA of this cell must be set, too. If bit LTCCTRk.OCM2 of the preceding cell is 1, the enable action will take place at the same time as in the preceding cell. Otherwise, the LTC will be enabled later on a capture/compare event in the preceding LTC, provided LTCCTRk.OCM0 or LTCCTRk.OCM1 of this cell is different from 0.

In this way, several LTCs can be enabled at the same time or one after the other. Normally, the LTCs will be used in One Shot Mode, and a service request will be

1) Standard TriCore® write operations: ST.A, ST.B, ST.D, ST.DA, ST.DD, ST.HST.Q, ST.W
Standard PCP write operations: ST.F, ST.IF,BCOPY, COPY

2) Intrinsic TriCore® read-modify-write Operations: LDMST, ST.T, SWAP
Intrinsic PCP read-modify-write Operations: SET.F, XCH.F, CLR.F

generated after the last event to evaluate the data and to prepare the next enable sequence. A disabled LTC (LTCCTRk.CEN = 0) behaves as an inactive capture LTC.

Logical Operating Cells

The inter-cell communication architecture allows concatenation of several LTCs to a logical cell. A logical cell contains any number of LTCs communicating via M1 and M0 lines and ends at an LTC disabled for action input or transfer (such as an LTC configured as timer, reset timer or LTC initiated with LTCCTRk.OCM2 = 0).

Therefore, the LTC with the lowest order number should be configured in Reset Timer Mode, thus providing all other LTCs of the logical cell with a time base (YO) and a compare enable signal (SO). Another LTC of the same logical cell can be initiated in Compare Mode to reset the LTC via its event output line EO, when a programmed threshold value is reached (register LTCXR) and the current state of its select line input SI matches the condition selected by the LTCCTRk bits SOH/SOL. Additional LTCs of the same logical cell can operate in Capture Mode triggered by a rising edge, falling edge, or both edges of a GPTA®v5 input line or a clock line of the clock bus. On the generated event, these LTCs capture the current contents of the timer cell, can generate a service request, can perform a manipulation of a GPTA®v5 output line (set, reset or toggle), and can also reset the LTC via the event output line EO.

LTC Service Request

The service request output SQTk of a Local Timer Cell LTCk is controlled as shown in **Figure 22-55**. When the LTCk service request condition becomes active, the service request flag becomes always set. The service request output SQTk is only activated if it is enabled by the enable bit LTCCTRk.REN. Additional information about service request and interrupt handling is given on **Page 22-123**.

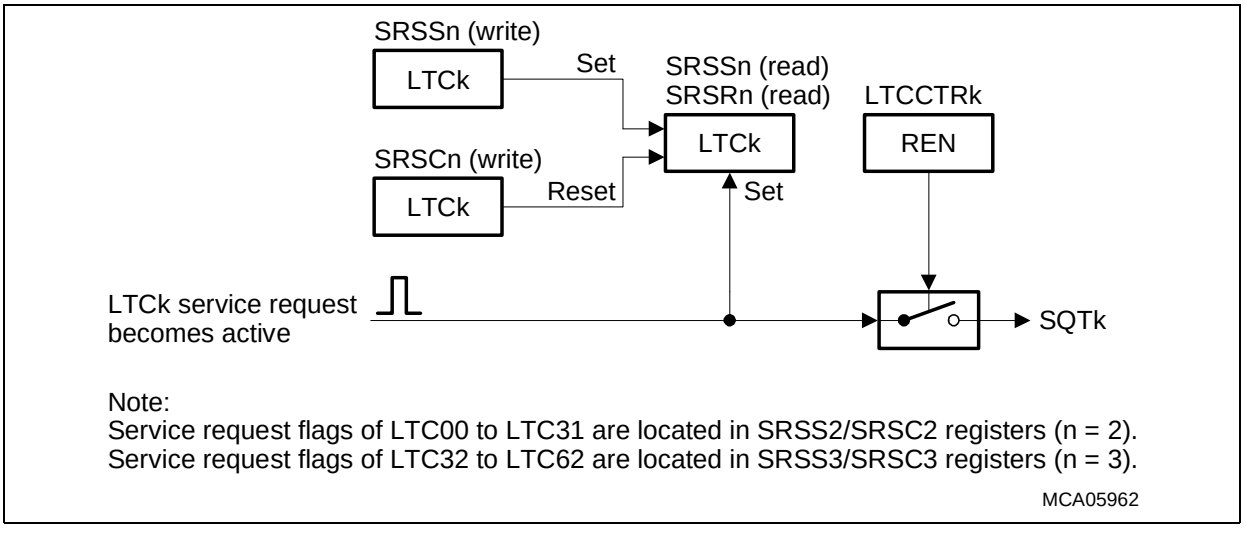


Figure 22-55 LTCk Service Request Generation

22.3.3.4 Local Timer Cell LTC63

The functionality of LTC63 is different to LTC00 to LTC62 and therefore described below.

Registers

The following registers are assigned to Local Timer Cell LTC63:

- LTCCTR63 = Local Timer Cell Control Register 63 (see **Page 22-204**)
- LTCXR63 = Local Timer Cell X Register 63 (see **Page 22-206**)
- SRSC3 = Service Request State Clear Register 3 (see **Page 22-230**)
- SRSS3 = Service Request State Set Register 3 (see **Page 22-231**)

Features

The GPTA®v5 Local Timer Cell array has one special cell, LTC63, which provides the following special features:

- **Compare Mode** on greater equal compare of the last timer, 16-bit based with following actions:
  - Service request generation
  - Output signal transition generation (set, reset, toggle the output signal).
- **Bit Reversal Mode:**
  - Timer can be selected to enable a special PWM Mode, called pulse count modulation (PCM)
- **Compare Value Switching** can be triggered by a hardware signal. This function can generate a service request. One Shot Mode makes it possible to stop the function after the first event.

Architecture

LTC63 is locally equipped with a 16-bit compare register, a 16-bit shadow register and a 16-bit greater comparator (**Figure 22-56**).

The LTC63 has the following inputs:

- A local input data bus (YI) carrying the local timer value of the adjacent LTC with lower order number
- A TI input reporting the occurrence of a local timer value update of the adjacent LTC with lower order number
- A trigger/enable input LTCkIN for compare value switching hooked to one of the following signals sources:
  - External port lines
  - GTC00 to GTC31 outputs
  - Clock bus signals
  - PDL0 or PDL1 outputs
  - Internal GPTA®v5 kernel input signals INTx (x = 0-3)

The LTC63 provides the following output signals:

- One data output line (LTCkOUT) that can be connected to:
  - External port lines
  - Inputs of an MSC module
  - Outputs and/or inputs of Global Timer Cell inputs
- One service request line (SQT) triggered by a compare or copy event
- An EO output reporting the occurrence of a local event to the adjacent LTC with lower order number

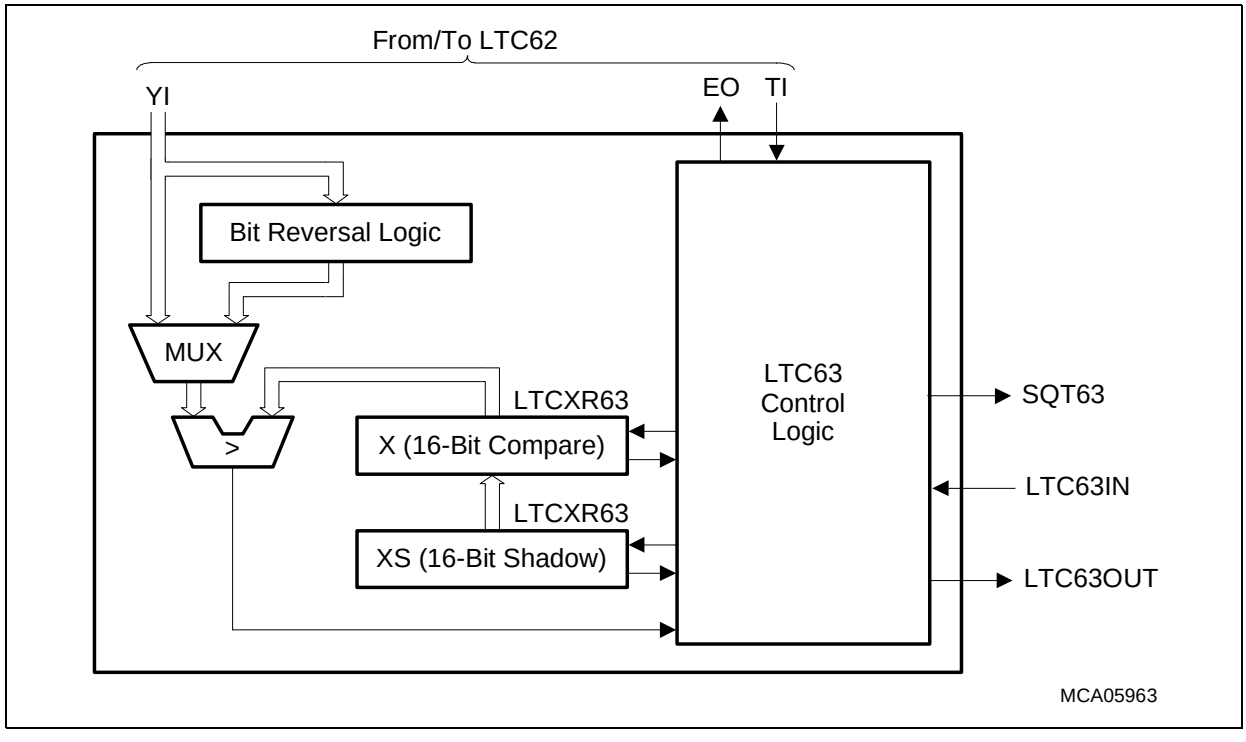


Figure 22-56 Architecture of Local Timer Cell 63

Compare

The compare function is always enabled. As long as the 16-bit compare value LTCXR63.X is greater than the timer value provided at YI, the comparator output signal is 1. The timer value at YI comes from the LTC62 either in original or in reversed order (Bit0 <-> Bit15, Bit1 <-> Bit14, etc.). The greater comparator output is connected directly to the output line LTC63OUT.

The 16-bit compare value LTCXR63.X is never greater than the LTC timer value ($FFFF_H$) coming from YI and which is used on LTC timer reset. Without special measures, a duty cycle of 100% cannot be achieved. There is always one LTC timer clock missing. Therefore, additional logic generates a permanent high signal whenever the 16-bit compare value LTCXR63.X is $FFFF_H$.

When the comparator output signal changes from 1 to 0,

- the service request flag LTC63 is set,
- an interrupt request will be activated if enabled by bit field LTCCTR63.REN,
- the event output line EO is set to high level for one f_{GPTA} clock cycle.

As well as the16-bit compare register LTCXR63.X, the LTC63 also contains a 16-bit shadow register LTCXR63.XS. Both 16-bit registers are combined in the 32-bit register LTCXR. On an LTC input signal selected via the LTC input multiplexer, the contents of the shadow register are copied to the compare register.

Standard PWM Mode

The LTC63 can be used for standard PWM duty cycle generation with enhanced update features. For this purpose, a pair of LTCs with lower index is configured as reset timer/period compare register. The user must set the period compare register to the desired period - 2 and LTC63 to the desired duty cycle. With LTCCTR63.BRM = 0 (Bit Reversal Mode), timer bit reversal is disabled. LTC63 is used for standard PWM Mode but with enhanced update features due to the “greater” comparator. The compare register LTCXR63.X can be written on-the-fly. If the duty cycle is changed at an arbitrary time, the actual duty cycle for the current period will reflect the old duty cycle, the new one, or a mixture of both. A duty cycle of 100% will be generated if the compare register is set to $FFFF_H$.

Pulse Count Modulation Mode (PCM)

With a period of 100 clocks and a duty cycle of 64%, standard PWM will produce an output signal that is ON for 64 clock cycles and OFF for the remaining 36 clock cycles. In contrast, pulse count modulation will generate 64 ON pulses and 36 OFF pulses distributed over the whole period as evenly as possible. PCM offers higher output frequency than standard PWM. This allows faster settling time e.g. when building a D/A converter in conjunction with an external low-pass filter. Only for very short or very long duty cycles does the method show no advantage or just little advantage compared to standard PWM.

As with standard PWM, a pair of LTCs with lower index is configured as reset timer/period compare register and LTC63 is used as duty cycle compare register. But now, Bit BRM (Bit Reversal Mode) in the LTCCTR63 register is set to 1 which enables the timer bit reversal to activate PCM.

The algorithm will also work if fewer than 16 timer bits are effectively used, even if the period is not a power of two. In any case, the user must write the duty cycle in unsigned 16-bit fractional format to the compare register.

Figure 22-57 shows an PCM example for an effective period of 6 clocks.

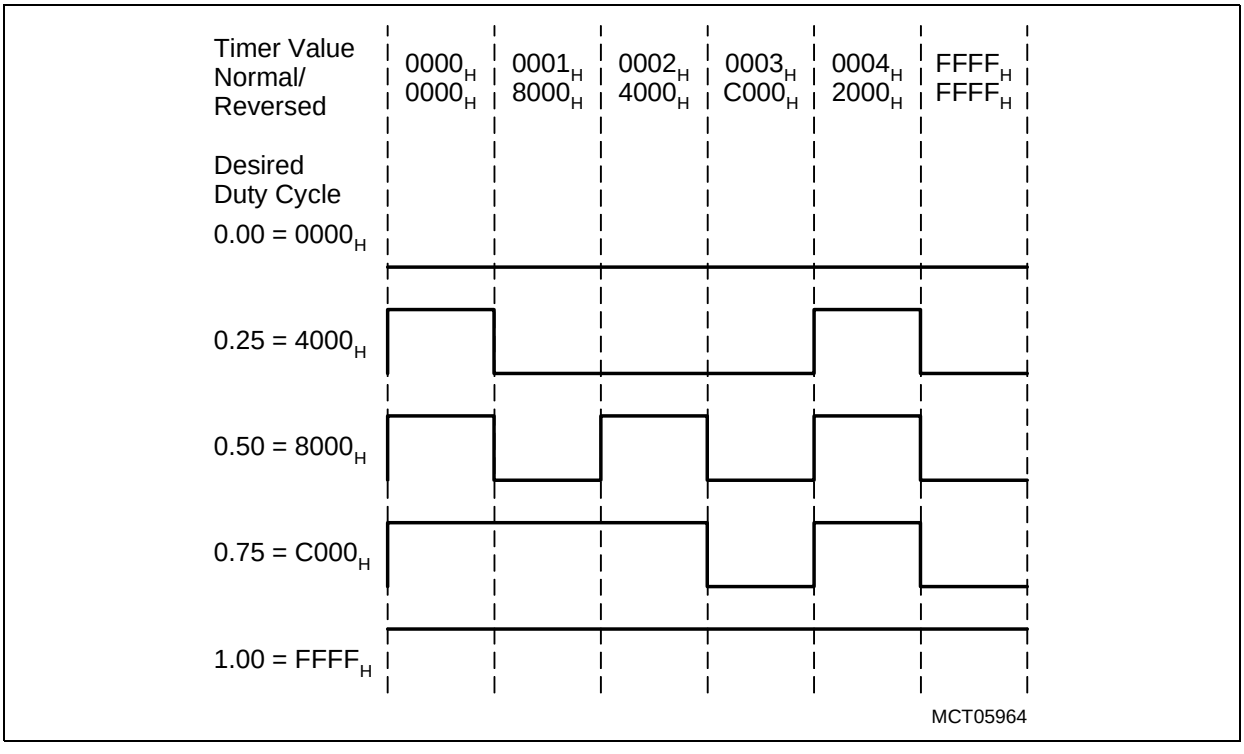


Figure 22-57 Pulse Count Modulation Example 1

Table 22-5 shows the rounding behavior for a period of 100 clocks.

Table 22-5 Implicit PCM Rounding

Desired Duty Cycle	Expected ON Pulses	Actual ON Pulses
0.000 = 0000	0	0
0.100 = $199A_H$	10	11
0.500 = 8000_H	50	50
0.800 = $CCCD_H$	80	82
0.900 = $E666_H$	90	90
0.999 = $FFBE_H$	100	99
1.000 = $FFFF_H$	100	100

The output is OFF for the remaining cycles of the period. The worst case error is approximately +2/-1 ON pulses. A subtraction performed via software may be used to reduce the worst case error.

If the duty cycle is changed at an arbitrary time, the actual duty cycle for the entire current period will reflect the old duty cycle, the new one, or a mixture of both.

Figure 22-58 shows another PCM example that demonstrates the difference between a standard PWM signal and the derived PCM signal. During one PWM period (128 clock cycles), the standard PWM signal is ON for 8 clock cycles and OFF for the remaining 120 clock cycles (duty cycle of 6.25%). The PCM signal operates with a PCM duty cycle of 1/8 = 0.125 resulting in 8 ON pulses of 1 clock cycle width within 1 PWM period.

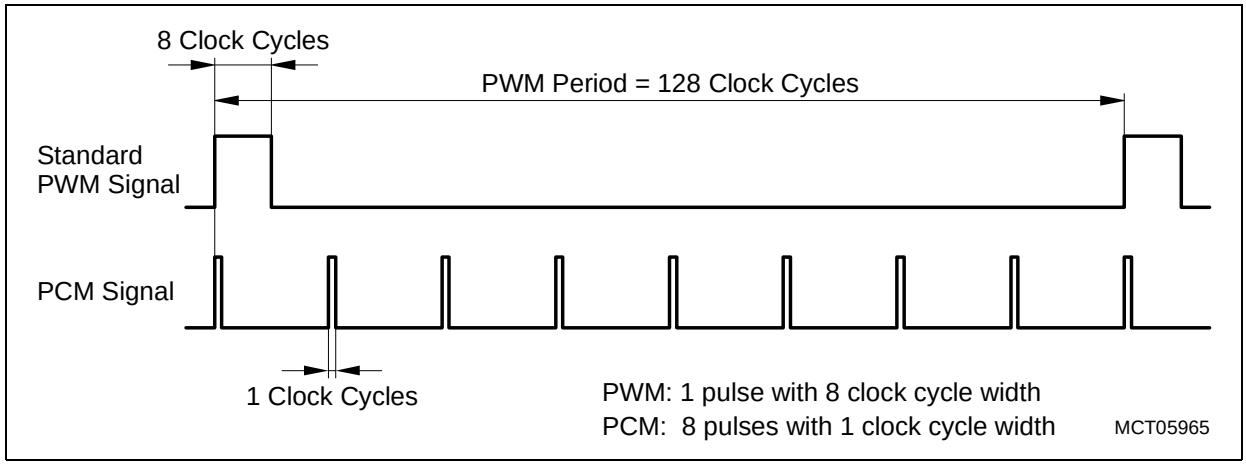


Figure 22-58 Pulse Count Modulation Example 2

Compare Value Switching

In both pulse modulation modes, it is possible to change the duty cycle either by software or on an LTC input signal. LTC63 contains two registers, the compare register and a shadow register. For software access, the compare register LTCXR63.X (= 16-bit low part of LTCXR63) is written directly.

For compare value switching triggered by hardware, the shadow register LTCXR63.XS (= 16-bit high part of LTCXR63) is pre-loaded with the desired duty cycle. On an LTC input signal selected via the LTC input multiplexer,

- The shadow register content LTCXR63.XS is copied to the compare register LTCXR63.X,
- The LTC63 service request flag is set,
- An interrupt request will be activated if enabled by bit field LTCCTR63.REN.

The data input line LTC63IN can operate in two modes (selected by bit LTCCTR63.ILM):

- Level Sensitive Mode or
- Edge Sensitive Mode

In Edge Sensitive Mode, the active edges are selected by bits LTCCTR63.FED and LTCCTR63.RED. In Level Sensitive Mode, the data input line LTCkIN is sensitive on a high level.

Various clocking modes of the LTC63 copy function are possible, depending on the source selected for the input line by the input multiplexer (see **Table 22-6**).

Table 22-6 LTC63 Data Input Line Operation

Input Source	Level Sensitive Input Line	Edge Sensitive Input Line
External Signal (Port line)	The external signal operates as gating signal for the cell. If the input is high the copy function of the LTC cell is performed with each rising edge of the GPTA®v5 module clock f_{GPTA}.	The copy function of the LTC cell is performed on selected edge(s).
Internal Clock Bus Line or PDL output or INT input	The copy function is performed with the internal clock or PDL/INT signal.	The copy function of the LTC cell is performed on selected edge(s). In case of full speed GPTA®v5 module clock selection, the Level Sensitive Mode must be selected. The Edge Sensitive Mode will not produce an event in this special case.
GTC output	The GTC output signal operates as gating signal for the cell. If the input is high the copy function of the LTC cell is performed with each rising edge of the GPTA®v5 module clock f_{GPTA}.	The copy function of the LTC cell is performed on selected edge(s).

When bit LTCCTR63.OSM is set to 1, a self-disable is executed after each copy event (PWM is not affected). The current state of the LTC copy enable may be evaluated by reading the control register flag bit LTCCTR63.CEN.

The output can be switched immediately to 0 or 1 in any pulse modulation mode by writing 0000_H or $FFFF_H$ to the duty cycle compare register LTCXR63.X.

LTC63 Service Request

The service request SQT63 can be generated by one of the following events:

- Comparator output changes from 1 to 0 (this makes sense mainly for standard PWM),
- Copy event.

Bit combinations 01_B and 10_B of bit field LTCCTR63.REN selects one of the two service request sources and enables it. Output SQT63 becomes active in these two cases. With the other two bit combinations of bit field LTCCTR63.REN (00_B, 11_B), the SQT63 output will not be activated. The LTC63 service request flag SRSS3.LTC63 will be set on a service request independently of LTCCTR63.REN. Additional information on service request and interrupt handling is provided on **Page 22-123**.

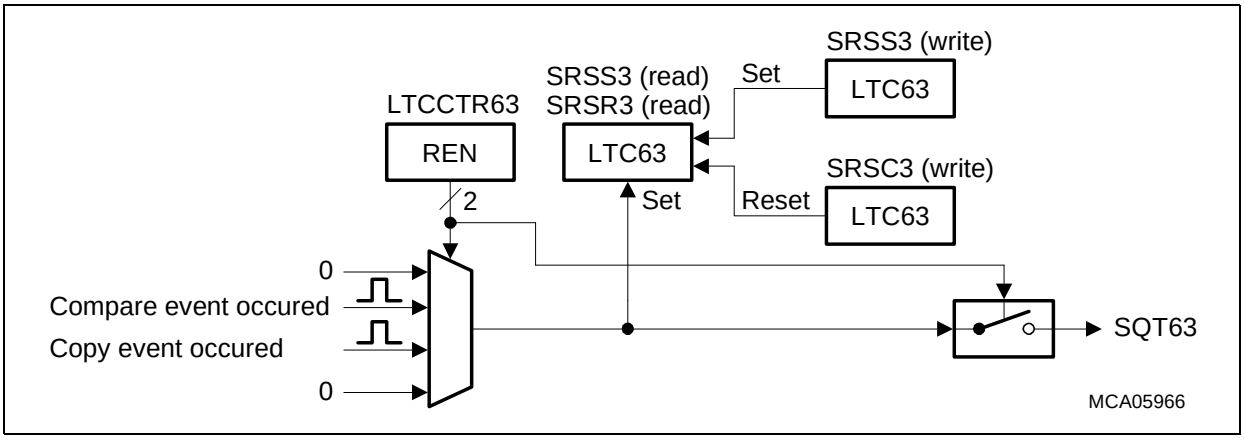


Figure 22-59 LTC63 Service Request Generation

22.3.3.5 Coherent Update

This section describes the two different mechanism to update signal features (e.g. period, duty cycle) if using Local Timer Cells. Both mechanism grant a coherent update only if a single update within a group of Local Timer Cells using a common Local Timer is performed within a timer period. So coherent update can only be granted if between coherent updating routine exit and coherent updating routine entry a time period of more then a period is maintained. If updating more frequently, software has to take care of coherency.

Global Coherent Update

The first mechanism, the so called global coherent update, is very useful to update a number of Local Timer Cells simultaneously. Furthermore this is the only way to grant a coherent update of a Local Timer Cell used as the period cell for a Reseted Timer. This global coherent update uses a common signal line (SI/SO) within a group of Local Timer Cells (all Local Timer Cells following an LTC configured as Timer). A pair of Local Timer Cells are configured so one cell being active on a high level of this SI/SO

(LTCCTRk.SOH = 1, LTCCTRk.SOL = 0) signal and the other cell being active on a low level of this SI/SO (LTCCTRk.SOH = 0, LTCCTRk.SOL = 1) signal. This pair of Local Timer Cells are configured to generate both action request for a single output signal (e.g. pin). If the global SI/SO is in a low state, all Local Timer Cells to be active on a high level of SI/SO can be configured (programmed with new values) without distortion of the output signal, because of being inactive. By setting the LTCCTRk.CUD bit within the Local TImer Cell used as Local Timer, the SI/SO bit will be toggled at the start of the next timer period, so all pair of Local Timer Cells simultaneously switch from active to passive (LTCCTRk.SOH = 0, LTCCTRk.SOL = 1) or passive to active state. Now the Local Timer Cells being inactive (LTCCTRk.SOH = 0, LTCCTRk.SOL = 1) may be configured (programmed) for the next update. No Local Timer Cell may be configured (programmed) while respective timer bit LTCCTRk.CUD =1, else wise a coherent update is no longer granted. Either the LTCCTRk.CUD has to be reset to 0 by software or a update of the Local Timer Registers have to be delayed to the next start of the period (LTCCTRk.CUD is reset by hardware to 0). The following example shows a PWM using global coherent update.

Fully Programmable PWM Signal Generation with 5 LTCs (global coherent update)

As shown in **Figure 22-59**, a logical cell of five LTCs can be used to generate a PWM signal with a programmable duty cycle, period length, and fully global coherent update of the period and duty cycle. In this example, LTC00 up to LTC04 are used to generate a PWM signal at the output of LTC04. To reduce complexity of this example, only a single duty cycle pair is described in the following text. More duty cycle cells may follow using the same reseted timer and pair of period cells. So if requiring a second duty cycle LTC pair, the first pair would be located to LTC2 and LTC6, the next pair on LTC 3 and LTC7 and the period cells would be assigned to LTC1 and LTC4.

LTC00 is configured in Reset Timer Mode thus providing all subsequent cells with a time base. LTC00 is clocked by a clock signal at the LTC00IN which has been selected by the LTC input multiplexer. LTC00 counts after reseted by LTC01 or LTC02 from $FFFF_H$, 0000_H, 0001_H … LTCXR01.X or LTCXR01.X. The period of the generated PWM is therefore LTCXR01.X +2 or LTCXR02.X + 2.

LTC01 and LTC02 are configured in Compare Mode. They are enabled if its SI inputs are at low level and responsible for the LTC04OUT signal generation in Phase 1. With the programmed values from **Table 22-7**, the LTC04OUT signal of Phase 1 has a period of 1000_D (= $3E8_H$) clocks of the LTC00IN clock signal and a duty cycle of 20% (= 200_D or $C8_H$).

LTC01 is configured in such a way (LTCCTR01.OCM = 011_B) that its output LTC01OUT is set to 1 whenever the LTC00 timer value LTCXR00.X is equal to the LTC01 compare value LTCXR01.X.

LTC02 is configured in such a way (LTCCTR02.OCM = 110_B) that its output LTC02OUT is reset whenever the LTC00 timer value LTCXR00.X is equal to the LTC02 compare value LTCXR02.X or it copies the action from the previous LTC01.

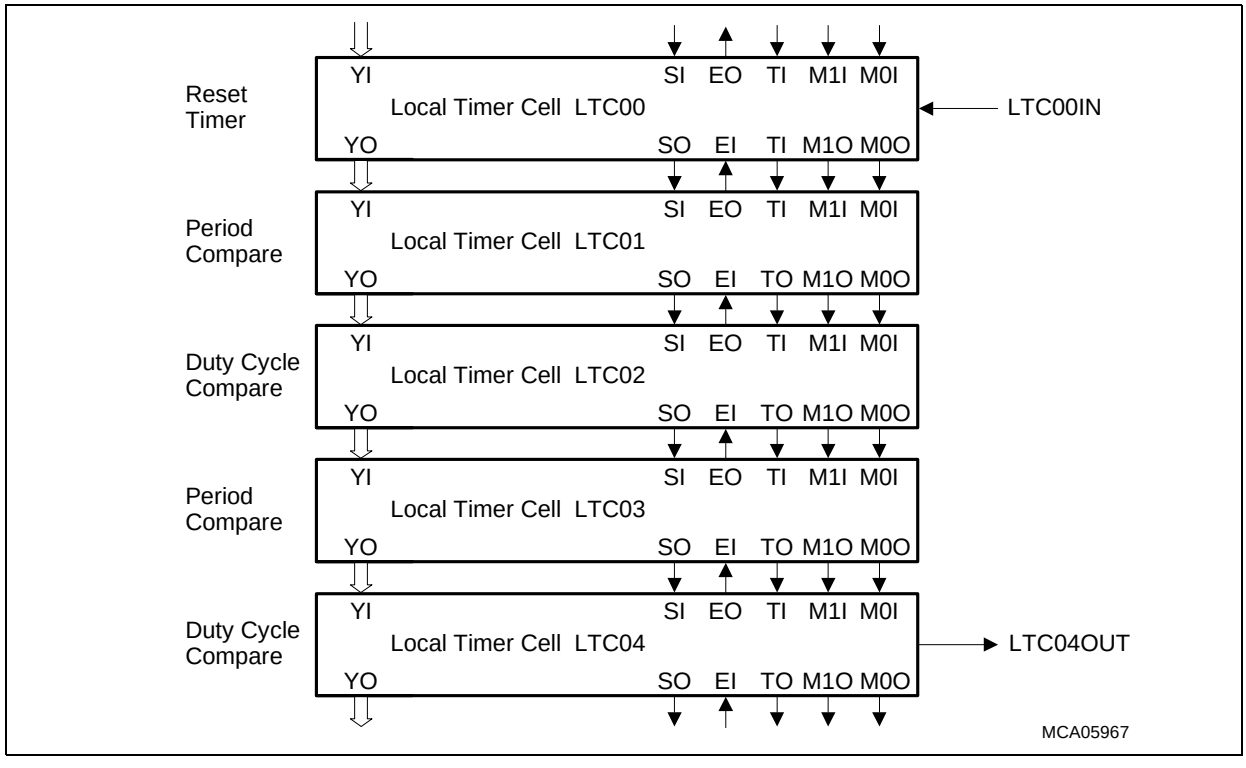


Figure 22-60 PWM Signal Generation with LTCs (Global Coherent Update)

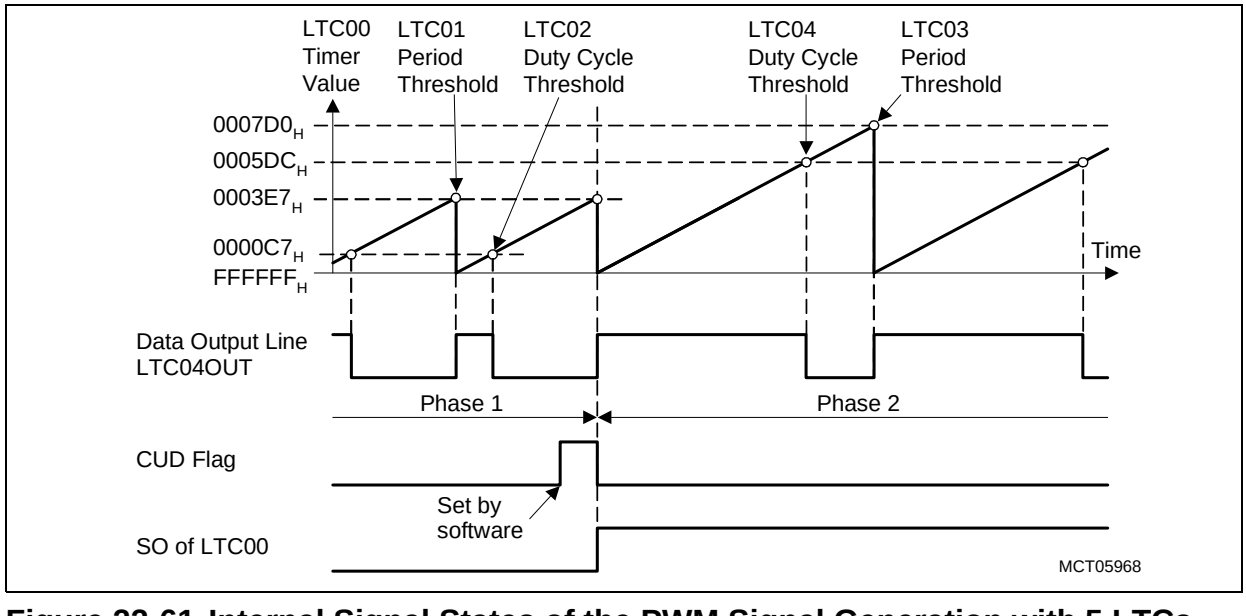


Figure 22-61 Internal Signal States of the PWM Signal Generation with 5 LTCs

LTC03 and LTC04 are configured in Compare Mode. They are enabled if its SI inputs are at high level and are responsible for the LTC04OUT signal generation in Phase 2. With the programmed values from **Table 22-7**, the LTC04OUT signal of Phase 2 has a period of 2000_D (= $7D0_H$) clocks of the LTC00IN clock signal and a duty cycle of 75% (= 1500_D or $5DC_H$).

LTC00 to LTC04 for the PWM example must be configured as defined in **Table 22-7**.

Note: Special care has to be taken not to reprogrammed the group of local timer cells (LTC) CAPCOM register before the previous global or local coherent update has been completed (end of current local timer period). Therefore maximum only one global coherent update within a timer period is possible! No Local coherent updates may be activated while a global coherent update modifying the period has not been completed.

Note: If several sequential coherent updates within a group of Local Timer Cells (LTC) is required, instead of using the global coherent update feature, the local coherent update mechanism (double action principle) is preferable. Mixing both principle, so updating the period using the coherent update and updating one or more duty cycle using local coherent update (double action principle) may result under specific condition in distorted signals (new duty cycle, old period or old duty cycle and new period). Therefore within a period either a global coherent update or multiple local coherent updates may be scheduled.

Note: To generate an output signal having 0% duty cycle (continuously low), the duty compare of the active cells must be set to $FFFF_H$. The timer sets the data output line by generating a respective signal on MO0 and MO1, but this signal is overruled by the dominating duty compare cell resetting the same data output line and therefore not passing the MI0 and MI1 signal from the timer to the data output line. This result in a data output line remaining continuously low.

Note: To generate an output signal having 100% duty cycle (continuously high), the duty cycle threshold must be set above the period threshold value. Therefore no reset event for the data line is generated and periodically the timer generates a set event. This result in a data output line remaining continuously high.

Table 22-7 Programming Values for PWM Signal Generation with 5 LTCs

Register	Value	Function
LTC00 Configuration Setup		
GPTA0_LTCXR00	0000 0000_H	LTC00 data register value = 0
GPTA0_LTCCTR00	0001 0413_H	MOD = 11_B: Reset Timer Mode selected OSM = 0: LTC00 continuously enabled ILM = 0, RED = 1, FED = 0: Input LTC00IN operates in Edge Sensitive Mode with rising edge; one clock bus signal is selected via the LTC input multiplexer SLO = 0: state of select line output SO is 0 CEN = 0: enable LTC00 for local events OCM = 000_B: hold LTC00OUT state
LTC01 Configuration Setup		
GPTA0_LTCXR01	0000 $03E7_H$	Load compare value = $3E7_H$ = 999_D
GPTA0_LTCCTR01	0001 $5C11_H$	MOD = 01_B: Compare Mode with LTC00 selected OSM = 0: LTC01 continuously enabled SOH = 0, SOL = 1: compare enabled by low level at SI BYP = 0: local bypass in LTC01 is disabled GBYP = 1: global bypass in LTC01 is disabled EOA = 0: LTC02 enabled for local events OCM = 011_B: set LTC01OUT by a local event only OIA = 1: output action defined by OCM must be performed immediately
LTC02 Configuration Setup		
GPTA0_LTCXR02	0000 $00C7_H$	Load compare value = $C7_H$ = 199_D
GPTA0_LTCCTR02	0001 3411_H	MOD = 01_B: Compare Mode with LTC00 selected OSM = 0: LTC02 continuously enabled SOH = 0, SOL = 1: compare enabled by low level at SI BYP = 0: local bypass in LTC02 is disabled GBYP = 1: global bypass in LTC02 is disabled EOA = 0: LTC02 enabled for local events OCM = 110_B: reset LTC02OUT by a local event or copy the previous cell action OIA = 0: no immediate output action required

Table 22-7 Programming Values for PWM Signal Generation with 5 LTCs (cont'd)

Register	Value	Function
LTC03 Configuration Setup		
GPTA0_LTCXR03	0000 $07CF_H$	Load compare value = $7CF_H$ = 1999_D
GPTA0_LTCCTR03	0001 $7C21_H$	MOD = 01_B: Compare Mode with LTC00 selected OSM = 0: LTC03 continuously enabled SOH = 1, SOL = 0: compare enabled by high level at SI BYP = 0: local bypass in LTC03 is disabled GBYP = 1: global bypass in LTC03 is disabled EOA = 0: LTC03 enabled for local events OCM = 111_B: set LTC04OUT by a local event or copy the previous cell action OIA = 1: output action defined by OCM must be performed immediately
LTC04 Configuration Setup		
GPTA0_LTCXR04	0000 $05DB_H$	Load compare value = $5DB_H$ = 1499_D
GPTA0_LTCCTR04	0001 3421_H	MOD = 01_B: Compare Mode with LTC00 selected OSM = 0: LTC04 continuously enabled SOH = 1, SOL = 0: compare enabled by high level at SI BYP = 0: local bypass in LTC04 is disabled GBYP = 1: global bypass in LTC04 is disabled EOA = 0: LTC04 enabled for local events OCM = 110_B: reset LTC04OUT by a local event or copy the previous cell action OIA = 0: no immediate output action required

Local Coherent Update

The second mechanism, the so called local coherent update or double action principle, is very useful to update single Local Timer Cells without signal distortion (no other signal output beside the previously configured and the new configured). This coherent update may not be used for Local Timer Cells generating toggle action requests (LTCCTRk.OCM = OCM = $x01_B$). The local coherent update is useful to configure (programme) several Local Timer Cells within a group of Local Timer Cells (all using the same Local Timer Cell as time base) sequentially (not simultaneously), e.g. within different routines of the application software. The only restriction to grant non distorted output signals by hardware is to update (configure or programme) every single pair of Local Timer Cells no often than once within a timer period (time measured from previous routine exit and current routine entry), else wise software has to take care of coherency (non distorted signals). A pair of Local Timer Cells are configured one cell being active (LTCCTRk.SOH = 1, LTCCTRk.SOL = 1) and the other being inactive (LTCCTRk.SOH = 0, LTCCTRk.SOL = 0, LTCCTRk.CEN = 0). This pair of Local Timer Cells are configured to generate both action requests for a single output signal (e.g. pin). A Local Timer Cell being inactive may be configured (programmed with a new value) without distortion of the output signal, because of being inactive. By activating the newly configured Local Timer Cell (LTCCTRk.SOH = 1, LTCCTRk.SOL = 1, LTCCTRk.OSM = 0), now two Local Timer Cells generate the same type of action request. The one being earlier within the period will now drive the output signal (either the newly configured or the previously configured one). Now the Local Timer Cell being active before configuration is coherently deactivated (LTCCTRk.OSM = 1) on its next action. So one period later this cell will be inactive (LTCCTRk.CEN = 0) and may be used for the next local coherent update. Local and coherent update may be used simultaneously within a group of Local Timer Cells. The following example shows a PWM using local coherent update. To reduce complexity of this example, only a single duty cycle pair is described in the following text. More duty cycle cells may follow using the same reseted timer and pair of period cells. So if requiring a second duty cycle LTC pair, the first pair would remain on LTC2 and LTC3 and the next pair on LTC 4and LTC5 using LTC05 as output.

Programmable PWM Signal Generation with 4 LTCs (local coherent update)

As shown in **Figure 22-62**, a logical cell of four LTCs can be used to generate a PWM signal with a programmable duty cycle and local coherent update of this duty cycle. In this example, LTC00 up to LTC03 are used to generate a PWM signal at the output of LTC03.

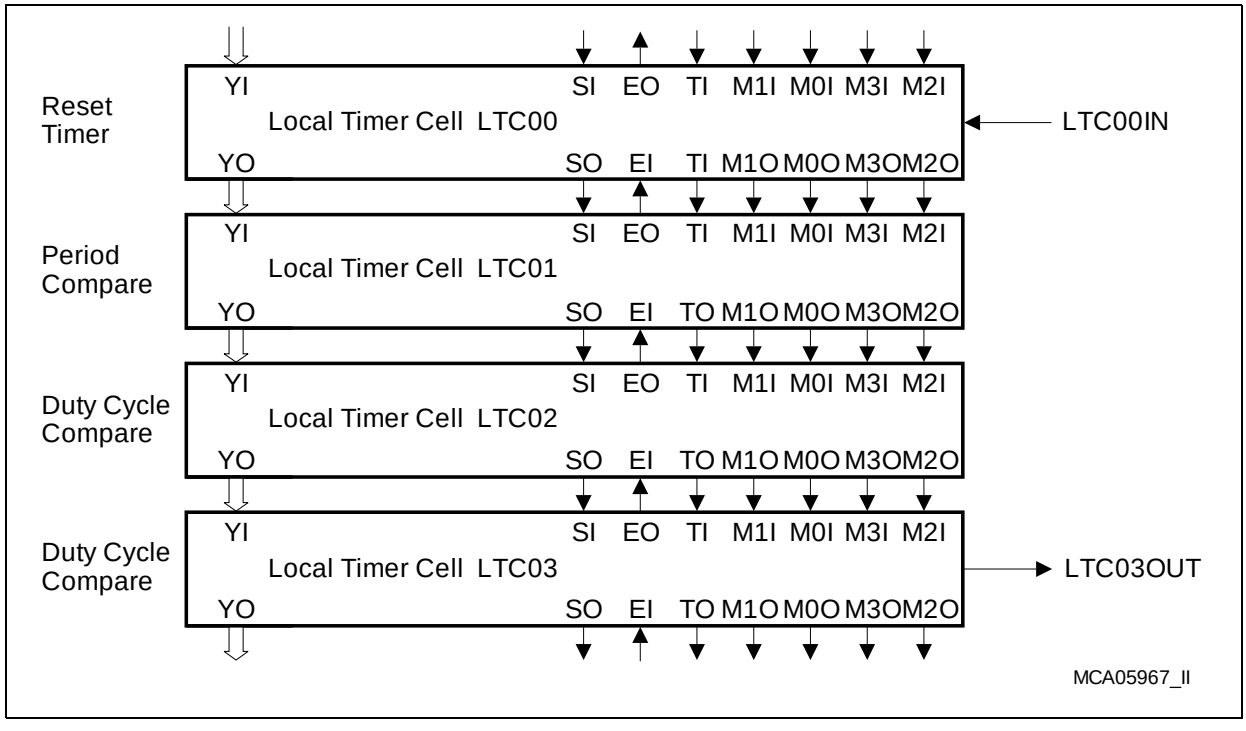


Figure 22-62 PWM Signal Generation with LTCs (Local Coherent Update)

LTC00 is configured in Reset Timer Mode thus providing all subsequent cells with a time base. LTC00 is clocked by a clock signal at the LTC00IN which has been selected by the LTC input multiplexer. LTC00 counts after reseted by LTC01 or LTC02 from $FFFF_H$, 0000_H, 0001_H … LTCXR01.X or LTCXR01.X. The period of the generated PWM is therefore LTCXR01.X +2 or LTCXR02.X + 2.

LTC01 is configured in Compare Mode. It is always active and responsible for the LTC03OUT signal generation in Phase 1. With the programmed value from **Table 22-8**, the LTC03OUT signal of Phase 1 has a period of 1000_D (= $3E8_H$) clocks of the LTC00IN clock signal and a duty cycle of 20% (= 200_D or $C8_H$).

LTC01 is configured in such a way (LTCCTR01.OCM = 011_B) that its output LTC01OUT is set to 1 whenever the LTC00 timer value LTCXR00.X is equal to the LTC01 compare value LTCXR01.X.

LTC02 and LTC03 are configured in Compare Mode. They are responsible for the LTC03OUT signal generation in Phase 2. With the programmed values from **Table 22-8**, the LTC03OUT signal of Phase 2 has a duty cycle of 77% (= 770_D or 302_H).

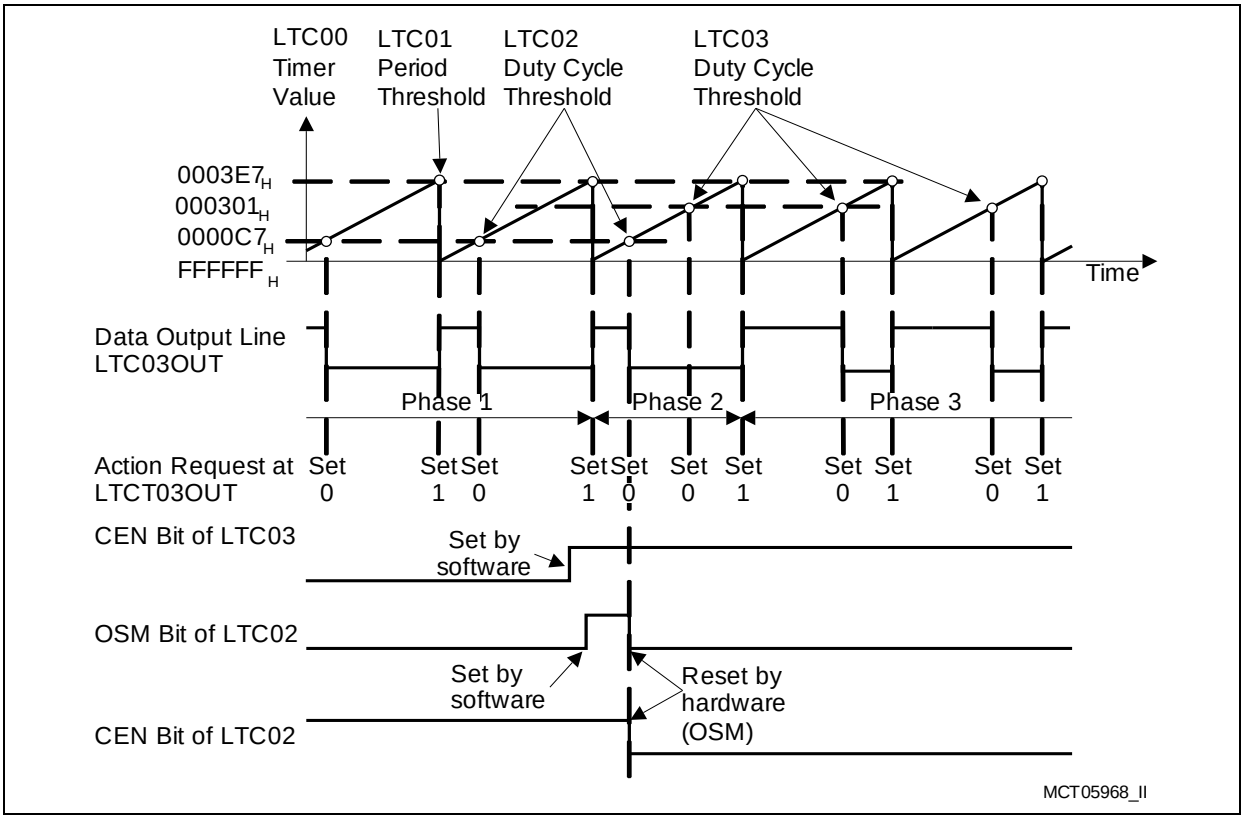


Figure 22-63 Internal Signal States of the PWM Signal Generation with 4 LTCs

LTC00 to LTC03 for the PWM example must be configured as defined in **Table 22-8**.

Note: Special care has to be taken not to reprogram LTCXR02.X or LTCXR03.X before the previous local coherent update has been completed (LTCXR02.CEN = 0 or LTCXR03.CEN = 0). Therefore maximum one coherent update within a timer period is possible (measured from previous routine exit to current routine entry)!

Note: If simultaneous coherent updates of several Local Timer Cells within a group of Local Timer Cells (LTC) is required, instead of using the local coherent update (double action principle), the global coherent update mechanism must be used.

Note: If coherent updating the period of a Timer in Reset Timer Mode is required, the global coherent update mechanism must be used.

Note: Global bypass may be used to route e.g. period action request (edge aligned PWM) around the pair of locally coherent updated Local Timer Cells to following Local Timer Cells. But special care has to be taken, because the timer bus is not routed over a LTC configured as timer.

Note: This scheme activates for one period two cells in parallel. Therefore, if enabled, also two interrupts for this one signal are generated, one for the old (previous) edge, one for the new (updated) edge.

Note: To generate an output signal having 0% duty cycle (continuously low), the duty compare of the active cells must be set to $FFFF_H$. The timer sets the data output line by generating a respective signal on MO0 and MO1, but this signal is overruled by the dominating duty compare cell resetting the same data output line and therefore not passing the MI0 and MI1 signal from the timer to the data output line. This result in a data output line remaining continuously low.

Note: To generate an output signal having 100% duty cycle (continuously high), the duty cycle threshold must be set above the period threshold value. Therefore no reset event for the data line is generated and periodically the timer generates a set event. This result in a data output line remaining continuously high.

Table 22-8 Programming Values for PWM Signal Generation with 4 LTCs

Register	Value	Function
LTC00 Configuration Setup		
GPTA0_LTCXR00	0000 0000_H	LTC00 data register value = 0
GPTA0_LTCCTR00	0000 0413_H	MOD = 11_B: Reset Timer Mode selected OSM = 0: LTC00 continuously enabled ILM = 0, RED = 1, FED = 0: Input LTC00IN operates in Edge Sensitive Mode with rising edge; one clock bus signal is selected via the LTC input multiplexer SLO = 0: state of select line output SO is 0 CEN = 0: enable LTC00 for local events OCM = 000_B: hold LTC00OUT state
LTC01 Configuration Setup		
GPTA0_LTCXR01	0000 $03E7_H$	Load compare value = $3E7_H$ = 999_D
GPTA0_LTCCTR01	0000 $5C11_H$	MOD = 01_B: Compare Mode with LTC00 selected OSM = 0: LTC01 continuously enabled SOH = 1, SOL = 1: enabled by both level at SI BYP = 0: bypass in LTC02 is disabled GBYP = 0: global bypass in LTC02 is disabled EOA = 0: LTC02 enabled for local events OCM = 011_B: set LTC01OUT by a local event only OIA = 1: output action defined by OCM must be performed immediately

Table 22-8 Programming Values for PWM Signal Generation with 4 LTCs (cont'd)

Register	Value	Function
LTC02 Configuration Setup		
GPTA0_LTCXR02	0000 00C7$_H$	Load compare value = C7$_H$ = 199$_D$
GPTA0_LTCCTR02	0000 3431$_H$	MOD = 01$_B$: Compare Mode with LTC00 selected OSM = 0: LTC02 continuously enabled SOH = 1, SOL = 1: compare enabled by low and high level at SI BYP = 0: local bypass in LTC02 is disabled GBYP = 0: begin of global bypass in LTC02 EOA = 0: LTC02 enabled for local events OCM = 110$_B$: reset LTC02OUT by a local event or copy the previous cell action OIA = 0: no immediate output action required
LTC03 Configuration Setup		
GPTA0_LTCXR03	0000 0301$_H$	Load compare value = 301$_H$ = 769$_D$
GPTA0_LTCCTR03	0001 3401$_H$	MOD = 01$_B$: Compare Mode with LTC00 selected OSM = 0: LTC03 continuously enabled SOH = 0, SOL = 0: compare disabled by low and high level at SI BYP = 1: local bypass in LTC03 is disabled GBYP = 1: end of global bypass in LTC03 EOA = 0: LTC04 enabled for local events OCM = 110$_B$: reset LTC03OUT by a local event or copy the previous cell action OIA = 0: no immediate output action required

The following code exemplifies the scheme to update the duty cycle of the PWM signal generation with 4 LTCs

```
Read LTCCTR02
If ( LTCCTR02.SOL=1 and LTCCTR02.SOH=1 and LTCCTR02.CEN=1 and
    LTCCTR02.OSM=0) Then
   Write New_Value into LTCXR03 of LTC03
   Set LTCCTR03.OSM=0 and LTCCTR03.SOL=LTCCTR03.SOH=1 for LTC03
    (Do not use LOAD/MODIFY/STORE, SWAP, or CLEAR_BIT assembler instructions)
   Set LTCCTR02.OSM=1 of LTC02
     (Use LOAD/MODIFY/STORE, SWAP, or CLEAR_BIT assembler instructions)
   Read LTCXR02
   Read LTCXR00
   If LTCXR00>Read_LTCXR02 then
      Clear LTCCTR02.SOL=LTCCTR02.SOH=0 of LTC02
   End If
Else
   Write New_Value into LTCXR02 of LTC02
   Set LTCCTR02.OSM=0 and LTCCTR02.SOL=LTCCTR02.SOH=1 for LTC02
    (Do not use LOAD/MODIFY/STORE, SWAP, or CLEAR_BIT assembler instructions)
   Set LTCCTR03.OSM=1 of LTC03
     (Use LOAD/MODIFY/STORE, SWAP, or CLEAR_BIT assembler instructions)
   Read LTCXR03
   Read LTCXR00
   If LTCXR00>Read_LTCXR03 then
      Clear LTCCTR03.SOL=LTCCTR03.SOH=0 of LTC03
   End If
End If
```

22.3.4 Input/Output Line Sharing Block (IOLS)

The I/O Line Sharing Block allows the 56 inputs and 112 outputs of the GPTA®v5 units to be routed with high flexibility between I/O lines, output lines, clock inputs, other on-chip peripherals and other GPTA®v5 cells. The GPTA®v5 module provides a total of 56 input lines and 112 output lines, assigned to seven I/O groups IOG[6:0], two on-chip trigger and gating signal groups OTG[1:0], and seven output groups OG[6:0].

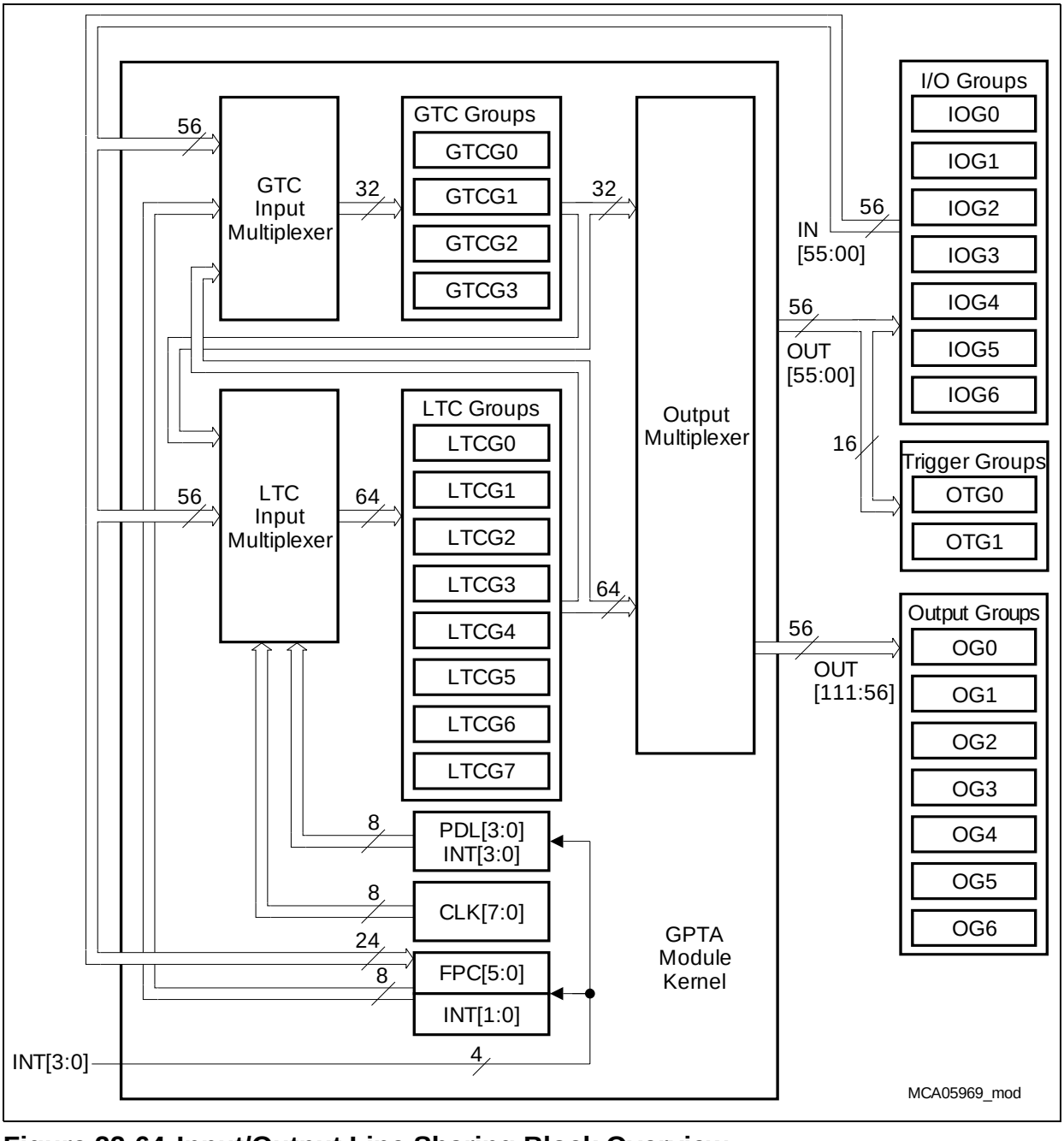


Figure 22-64 Input/Output Line Sharing Block Overview

The I/O Line Sharing Block does the following selections:

- FPC input line selection
- GTC and LTC output multiplexer selection
- On-chip trigger and gating signal selection
- GTC input multiplexer selection
- LTC input multiplexer selection

For choosing these selection, the input and output lines of the related cells are integrated into groups with eight parts each. Seven I/O groups, two on-chip and gating signal groups, seven output groups, four GTC groups, eight LTC groups, one clock group, one FPC/INT group, and one PDL/INT group are defined.

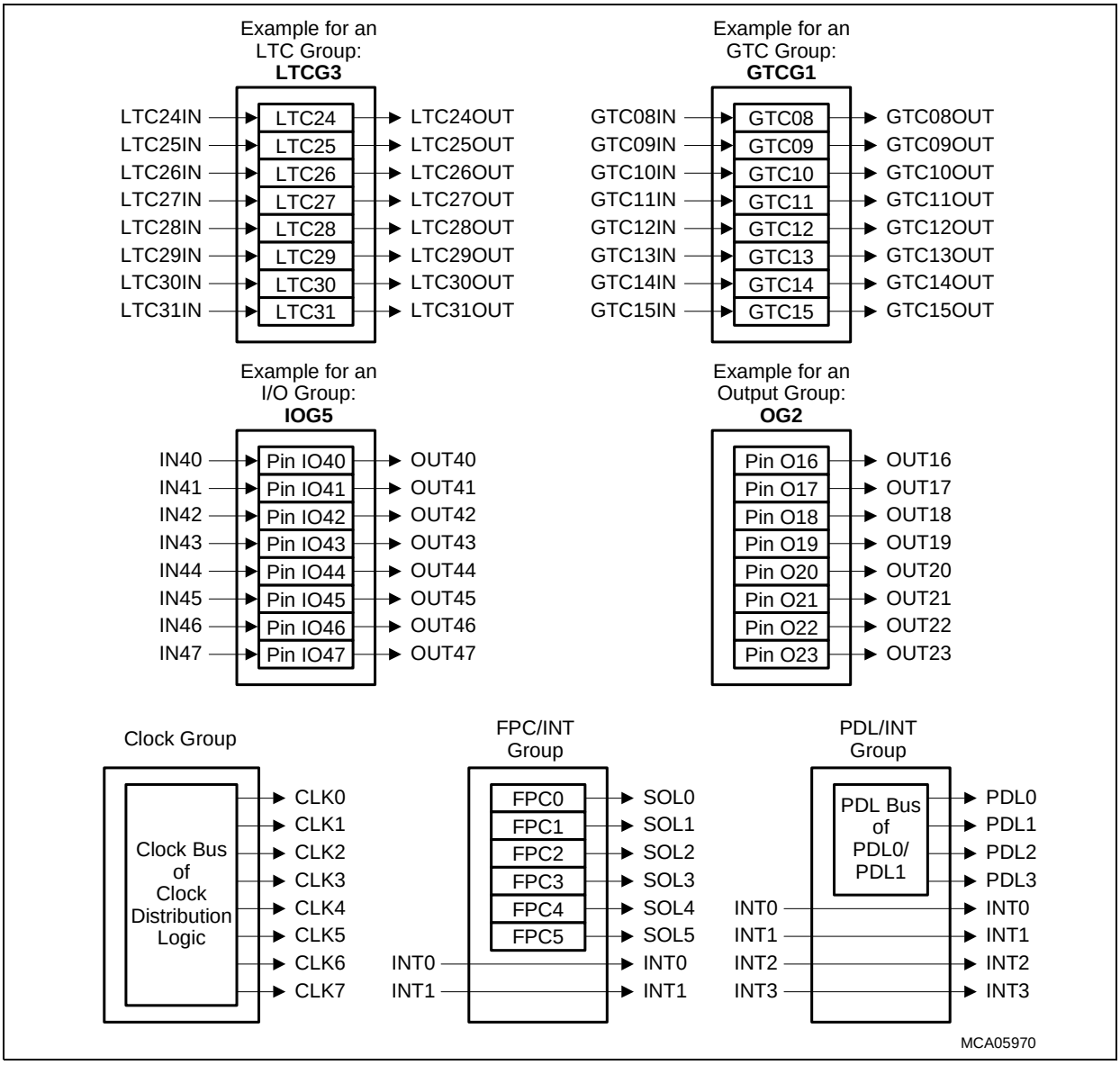


Figure 22-65 Groups Definitions for I/O Line Sharing Block

An **LTC group** combines eight LTC cells with its input and output lines. This results in eight LTC groups, LTCG0 to LTCG7.

A **GTC group** combines eight GTC cells with its input and output lines. This results in four GTC groups, GTCG0 to GTCG3.

An **I/O group** combines eight GPTA®v5 I/O lines connected to bi-directional device pins with its input and output lines. This results in seven I/O groups, IOG0 to IOG6, supporting 56 I/O lines.

An **Output group** combines eight GPTA®v5 output lines connected to device pins as an output. This results in seven output groups, OG0 to OG6, supporting 56 output lines.

The **Clock group** is a group that combines the eight clock bus output signals CLK[7:0] generated by the clock distribution cells.

The **FPC/INT group** is a group that combines the six level output signals SOL[5:0] of the FPCs with two external input lines INT[1:0] of the GPTA®v5 unit.

The **PDL/INT group** is a group that combines the four PDL output lines of the PDL bus with four external input lines INT[3:0] of the GPTA®v5 unit.

An **On-chip trigger and gating signal group** combines eight GPTA®v5 output lines connected to on-chip peripherals. This results in two on-chip trigger and gating signal groups, OTG0 to OG1, supporting 16 on-chip trigger and gating lines.

Table 22-9 Group to I/O Line/Cell Assignment

Group/Unit	Cell/Line	Input	Output
LTC Groups			
LTCG0	LTC[07:00]	LTC[07:00]IN	LTC[07:00]OUT
LTCG1	LTC[15:08]	LTC[15:08]IN	LTC[15:08]OUT
LTCG2	LTC[23:16]	LTC[23:16]IN	LTC[23:16]OUT
LTCG3	LTC[31:24]	LTC[31:24]IN	LTC[31:24]OUT
LTCG4	LTC[39:32]	LTC[39:32]IN	LTC[39:32]OUT
LTCG5	LTC[47:40]	LTC[47:40]IN	LTC[47:40]OUT
LTCG6	LTC[55:48]	LTC[55:48]IN	LTC[55:48]OUT
LTCG7	LTC[63:56]	LTC[63:56]IN	LTC[63:56]OUT
GTC Groups			
GTCG0	GTC[07:00]	GTC[07:00]IN	GTC[07:00]OUT
GTCG1	GTC[15:08]	GTC[15:08]IN	GTC[15:08]OUT
GTCG2	GTC[23:16]	GTC[23:16]IN	GTC[23:16]OUT
GTCG3	GTC[31:24]	GTC[31:24]IN	GTC[31:24]OUT

Table 22-9 Group to I/O Line/Cell Assignment (cont'd)

Group/Unit	Cell/Line	Input	Output
I/O Groups			
IOG0	–	IN[07:00]	OUT[07:00]
IOG1	–	IN[15:08]	OUT[15:08]
IOG2	–	IN[23:16]	OUT[23:16]
IOG3	–	IN[31:24]	OUT[31:24]
IOG4	–	IN[39:32]	OUT[39:32]
IOG5	–	IN[47:40]	OUT[47:40]
IOG6	–	IN[55:48]	OUT[55:48]
Output Groups			
OG0	–	–	OUT[63:56]
OG1	–	–	OUT[71:64]
OG2	–	–	OUT[79:72]
OG3	–	–	OUT[87:80]
OG4	–	–	OUT[95:88]
OG5	–	–	OUT[103:96]
OG6	–	–	OUT[111:104]
On-Chip Trigger and Gating Signals Groups			
OTG0	–	–	OTGS[07:00]
OTG1	–	–	OTGS[15:08]
Clock Group			
–	–	–	CLK[7:0]
FPC/INT Groups			
FPC[5:0]	–	–	SOL[5:0]
External Input [1:0]	–	–	INT[1:0]
PDL/INT Groups			
PDL[1:0] PDL Bus	–	–	PDL[3:0]
External Input [3:0]	–	–	INT[3:0]

22.3.4.1 FPC Input Line Selection

As shown on **Page 22-12**, each FPC cell can be connected to one out of four input lines SINk[3:0], to the GPTA®v5 module clock f_{GPTA}, or to the output of the preceding FPC. In total, 24 input lines out of the 56 input lines IN[55:00] from the I/O groups are connected (not programmable) with the FPCk inputs. The FPCk input line selection is controlled by the FPCCTRk.IPS bit fields. **Table 22-10** shows the FPC input line connections.

Table 22-10 FPC Input Line Assignments

FPC Control Register	Bit Field IPS	Selected Input Signal
FPCCTR0	000_B	IN0
	001_B	IN12
	010_B	IN24
	011_B	IN36
FPCCTR1	000_B	IN2
	001_B	IN14
	010_B	IN26
	011_B	IN38
FPCCTR2	000_B	IN4
	001_B	IN16
	010_B	IN28
	011_B	IN40
FPCCTR3	000_B	IN6
	001_B	IN18
	010_B	IN30
	011_B	IN42
FPCCTR4	000_B	IN8
	001_B	IN20
	010_B	IN32
	011_B	IN44
FPCCTR5	000_B	IN10
	001_B	IN22
	010_B	IN34
	011_B	IN46

22.3.4.2 GTC and LTC Output Multiplexer Selection

The output multiplexer shown in **Figure 22-64** and **Figure 22-66** below connects the 32 GTC output lines and the 64 LTC output lines with the I/O groups (7 × 8 = 56 output lines) and the output groups (7 × 8 = 56 output lines).

In case of low pin count packages, not all I/O groups may be routed to a pin.

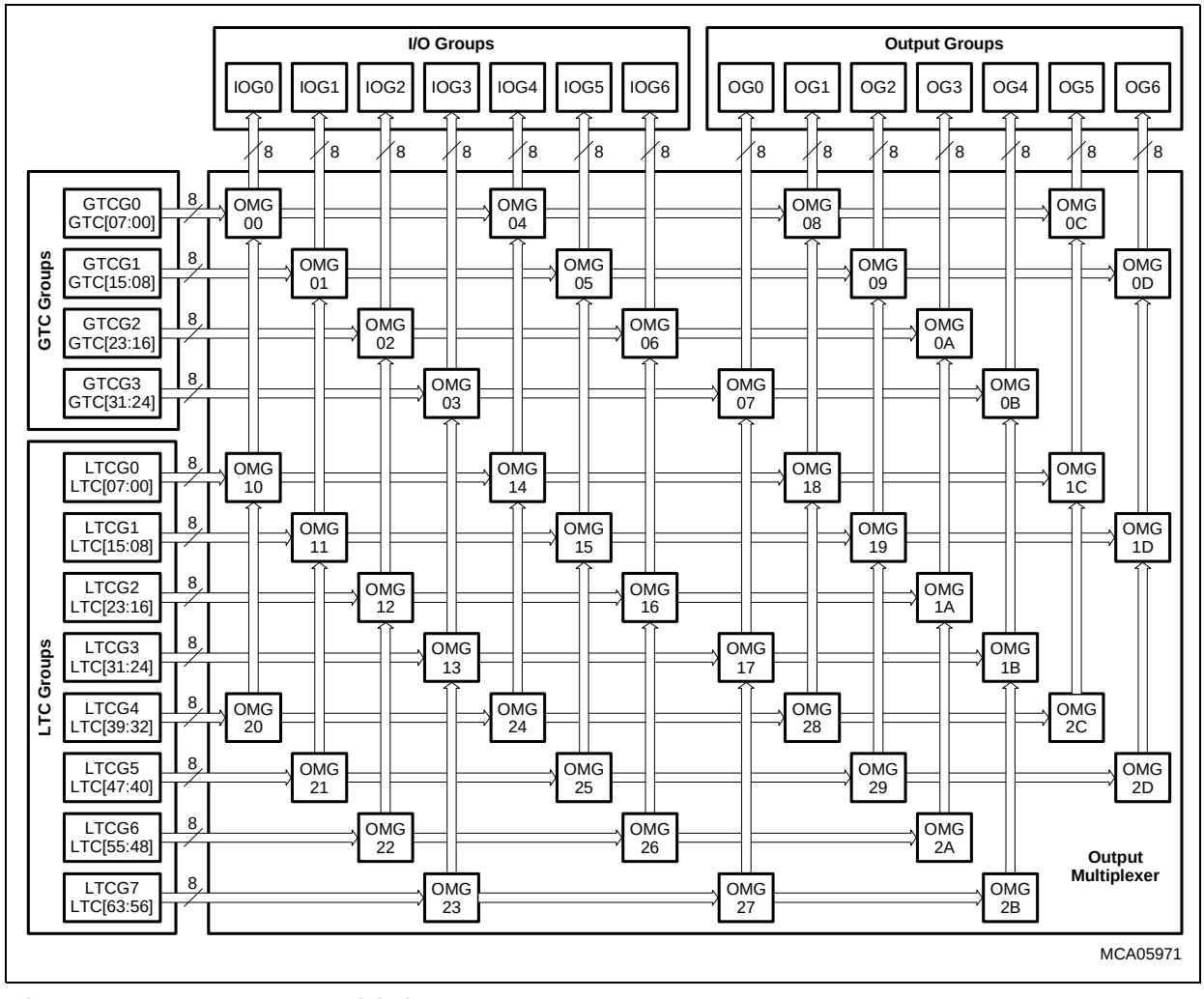


Figure 22-66 Output Multiplexer

The output multiplexer contains Output Multiplexer Groups (OMGs) that connect the Global Timer Cells or Local Timer Cells with the input lines of the I/O groups and output groups. GTCs and LTCs are grouped into four GTC groups (GTCG[3:0]) and eight LTC groups (LTCG[7:0]) with 8 cells each. In the same way, I/O groups and output groups are grouped into 14 groups (seven I/O groups and seven output groups) with 8 lines each.

Figure 22-67 shows the logical structure of an OMG.

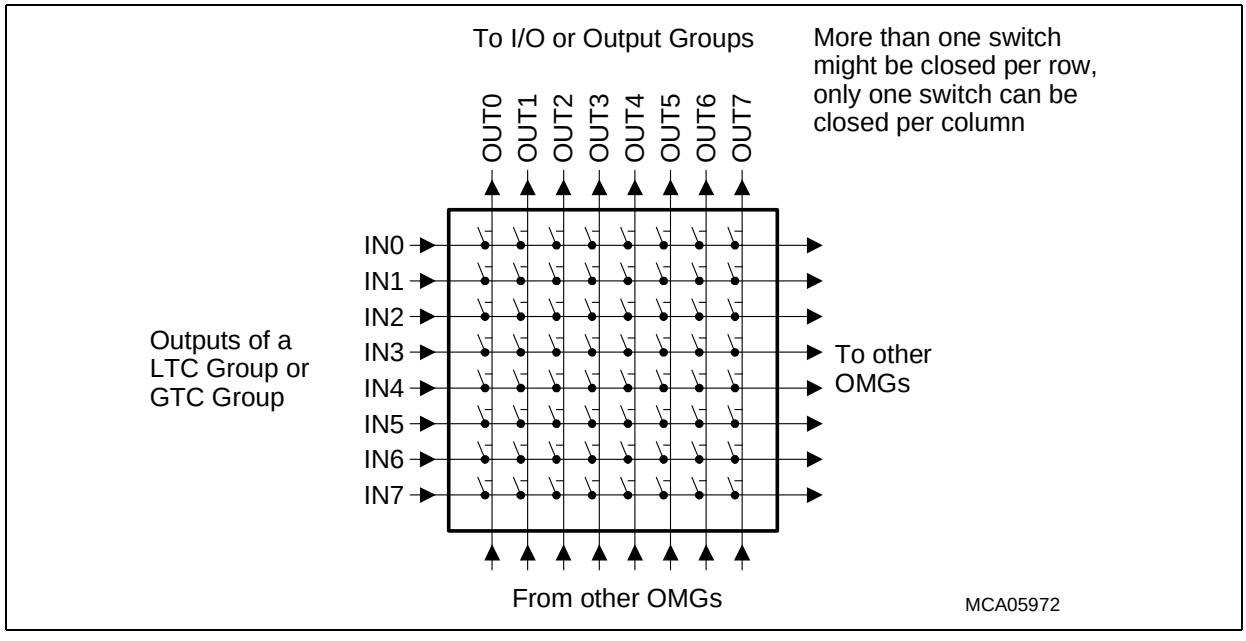


Figure 22-67 Output Multiplexer Group (OMG) Structure

Rules for connections to Output Multiplexer Group OMG:

- Within a GTC or LTC group, the output of the cell with the lowest index number is connected to OMG input line IN0. The remaining cells of a cell group are connected to OMG input lines IN1 to IN7 with ascending cell index numbers.
  Example: for OMG13 (see Figure 22-66), the cells LTC24 up to LTC31 are wired to the OMG13 input lines IN0 to line IN7.
- OMG output line OUT0 is always connected to the input of an I/O or output group with the lowest index. The remaining output lines OUT1 to OUT7 are connected to the I/O or Output lines with ascending index.
  Example: for OMG13 (see Figure 22-66), the outputs OUT0 to OUT7 are wired (via OMG03) to input lines 0 to 7 of I/O group 3 (IOG3).
- One input of an I/O or output group can be connected to the output of only one timer cell. This is guaranteed by the OMG control register layout. Otherwise, short circuits and unpredictable behavior would occur. On the other hand, it is permissible for the output of a GTC or LTC to be connected to more than one input of an I/O or output group.

The output multiplexer group configuration is based on the following principles:

- Each OMG is referenced with two index variables: n and g (OMGng)
- Index n is a group number. Global timer cell groups GTCG[3:0] have the group number 0, Local Timer Cell Groups LTCG[3:0] have the group number 1, and Local Timer Cell Groups LTCG[7:4] have the group number 2.

- Index g indicates the number of an I/O or output group g (g = 0-13_D) to which the outputs of the output multiplexer group OMGng are connected. I/O groups IOG0 to IOG6 are assigned to index variable g = 0 to 6 and output groups OG0 to OG6 are assigned to index variable g = 7 to 13.

The output multiplexer logic as seen for programming is shown in **Figure 22-68**. With this logic, always three GTC or LTC group signals are combined to one output line that leads to the input of an I/O or output group. For example, when looking at **Figure 22-66**, each of the eight output multiplexer output lines to I/O group IOG5 is connected via three OMGn5 (n = 0, 1, 2) with the eight outputs of one GTC group (GTCG1) and two LTC groups (LTCG1 and LTCG5).

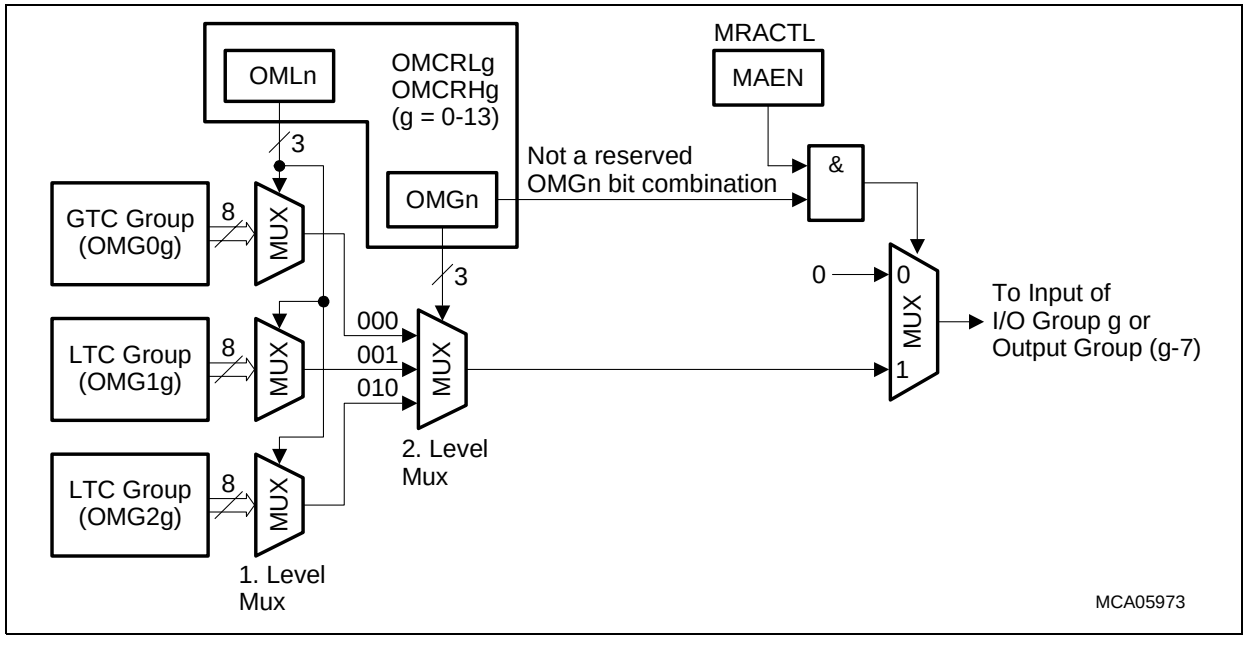


Figure 22-68 Output Multiplexer Group (Programmer's View)

The 1. level multiplexer is built up by three 8:1 multiplexers that are controlled in parallel by bit field OMLn. Bit field OMGn controls the 2. level multiplexer and connects one of the 1. level multiplexer outputs to output n. The output of the 2. level multiplexer is connected only to the input of an I/O group or output group if bit MRACTL.MAEN is set (multiplexer array enabled) and no reserved bit combination of OMGn is selected. If one of these conditions is not true, the corresponding OMG output will be held at a low level.

Two Output GPTA®v5, OMCRL and OMCRH (see also **Page 22-121**), are assigned to each of the I/O or output groups. Therefore, a total of 28 registers control the connections within the output multiplexer of the GPTA®v5 module.

The OMCRL registers control the OMG output lines 0 to 3. The OMCRH registers control the OMG output lines 4 to 7. **Table 22-11** lists all Output Multiplexer Control Registers with its control functions. Please note that the Output Multiplexer Control Registers are

not directly accessible but must be written or read using a FIFO array structure as described on Page 22-121.

Table 22-11 Output Multiplexer Control Register Assignments

I/O Group or Output Group		Controlled by Multiplexer Control Register	Selectable Groups via OMGng
IOG0	IN[03:00]/OUT[03:00]	OMCRL0	GTCG0, LTCG0, LTCG4
	IN[07:04]/OUT[07:04]	OMCRH0	
IOG1	IN[11:08]/OUT[11:08]	OMCRL1	GTCG1, LTCG1, LTCG5
	IN[15:12]/OUT[15:12]	OMCRH1	
IOG2	IN[19:16]/OUT[19:16]	OMCRL2	GTCG2, LTCG2, LTCG6
	IN[23:20]/OUT[23:20]	OMCRH2	
IOG3	IN[27:24]/OUT[27:24]	OMCRL3	GTCG3, LTCG3, LTCG7
	IN[31:28]/OUT[31:28]	OMCRH3	
IOG4	IN[35:32]/OUT[35:32]	OMCRL4	GTCG0, LTCG0, LTCG4
	IN[39:36]/OUT[39:36]	OMCRH4	
IOG5	IN[43:40]/OUT[43:40]	OMCRL5	GTCG1, LTCG1, LTCG5
	IN[47:44]/OUT[47:44]	OMCRH5	
IOG6	IN[51:48]/OUT[51:48]	OMCRL6	GTCG2, LTCG2, LTCG6
	IN[55:52]/OUT[55:52]	OMCRH6	
OG0	OUT[59:56]	OMCRL7	GTCG3, LTCG3, LTCG7
	OUT[63:60]	OMCRH7	
OG1	OUT[67:64]	OMCRL8	GTCG0, LTCG0, LTCG4
	OUT[71:68]	OMCRH8	
OG2	OUT[75:72]	OMCRL9	GTCG1, LTCG1, LTCG5
	OUT[79:76]	OMCRH9	
OG3	OUT[83:80]	OMCRL10	GTCG2, LTCG2, LTCG6
	OUT[87:84]	OMCRH10	
OG4	OUT[91:88]	OMCRL11	GTCG3, LTCG3, LTCG7
	OUT[95:92]	OMCRH11	
OG5	OUT[99:96]	OMCRL12	GTCG0, LTCG0, LTCG4
	OUT[103:100]	OMCRH12	

Table 22-11 Output Multiplexer Control Register Assignments (cont'd)

I/O Group or Output Group		Controlled by Multiplexer Control Register	Selectable Groups via OMGng
OG6	OUT[107:104]	OMCRL13	GTCG1, LTCG1, LTCG5
	OUT[111:108]	OMCRH13	

22.3.4.3 On-chip Trigger and Gating Output Multiplexer Selection

The On-chip Trigger and Gating Signal (OTGS) multiplexer shown in **Figure 22-64** and **Figure 22-69** below connects the 32 GTC output lines and the 64 LTC output lines with the on-chip trigger and gating signal groups (2 × 8 = 16 output lines).

In case of low pin count packages, not all I/O groups may be routed to a pin.

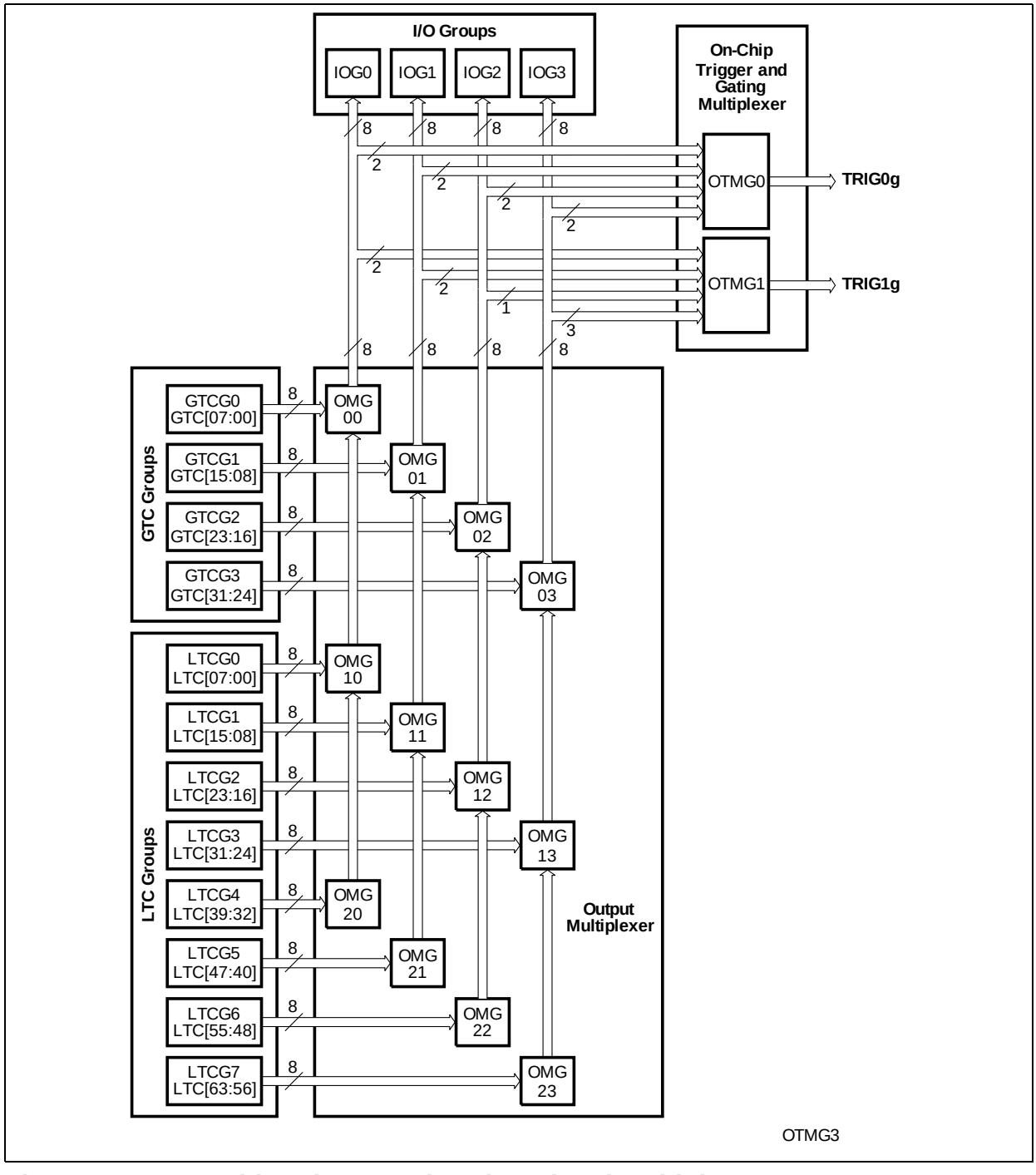


Figure 22-69 On-Chip Trigger and Gating Signal Multiplexer

The On-chip Trigger and Gating Signal multiplexer contains Output Multiplexer Groups (OMGs) that connect the Outputs of the I/O groups with the On-Chip Trigger Signals. These On-Chip Trigger Signals are grouped into two On-Chip Trigger groups (OTMG[1:0]) with 8 cells each.

Figure 22-67 shows the logical structure of an OTMG.

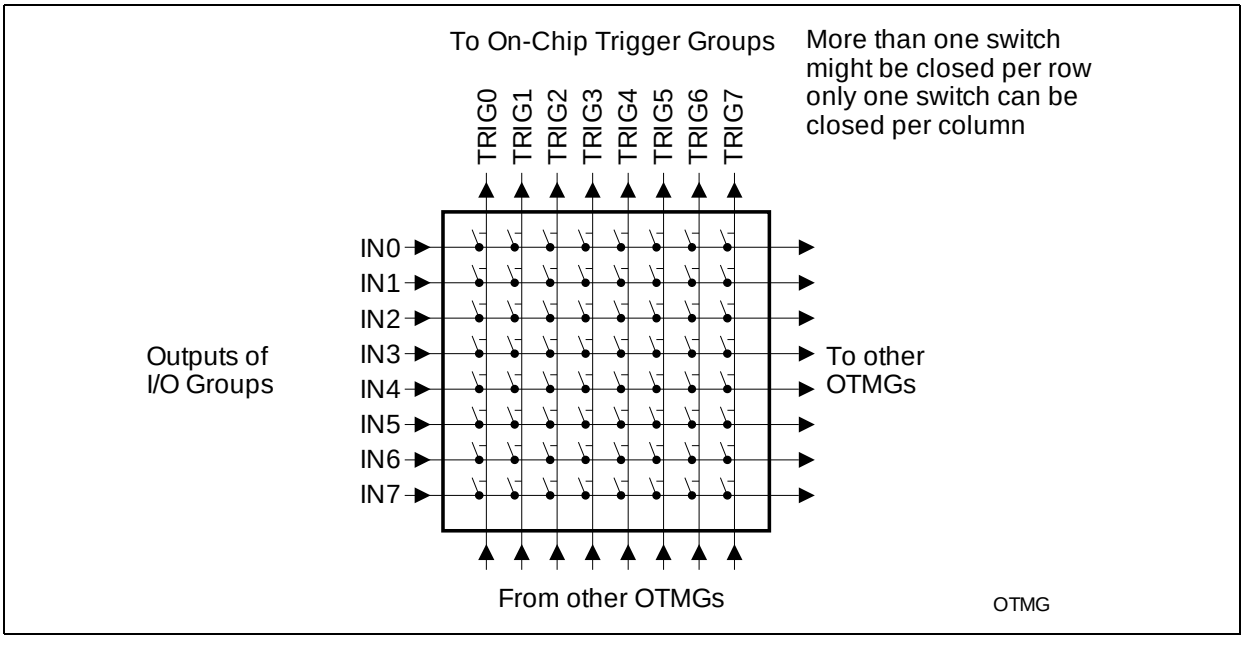


Figure 22-70 On-Chip Trigger and Gating Multiplexer Group (OTMG) Structure

Rules for connections to Output Multiplexer Group OTMG:

- Only one input of an On-chip Trigger and Gating Signal group can be connected to a On-chip Trigger and Gating Signal (TRIG). This is guaranteed by the OTMG control register layout. Otherwise, short circuits and unpredictable behavior would occur. On the other hand, it is permissible for the output of a I/O Group to be connected to more than one on-chip trigger and gating signal (TRIG).

The on-chip trigger and gating multiplexer group configuration is based on the following principles:

- Each OTMG is referenced with a single index variables: g (OTMGg)
- Index g indicates the number of an On-Chip Trigger and Gating Signal multiplexer group g ($g = 0\text{-}1_D$) to which the outputs of the On-Chip Trigger and Gating Signal multiplexer group OTMGg are connected. TRIG00 to TRIG07 are assigned to OTMG0 and TRIG10 to TRIG17 are assigned to OTMG1.

The on-chip trigger and gating signal multiplexer logic as seen for programming is shown in **Figure 22-71**.

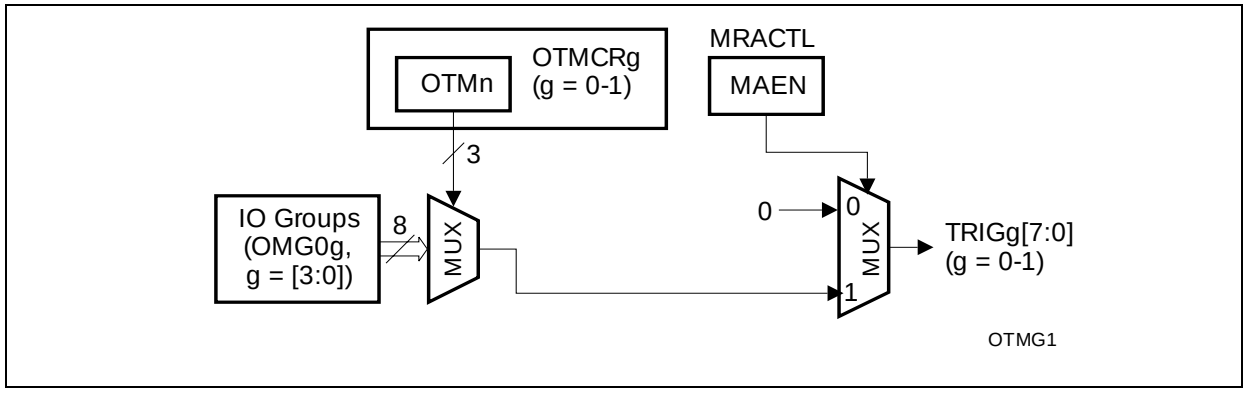


Figure 22-71 On-Chip Trigger and Gating Multiplexer Group (Programmer's View)

The multiplexer is built up by three 8:1 multiplexer that is controlled by bit field OTMn. Bit field The output of the multiplexer is connected only to the on-chip trigger and gating signal TRIGn if bit MRACTL.MAEN is set (multiplexer array enabled). If this condition is not true, the corresponding OTMG output will be held at a low level.

Sixteen on-chip trigger and gating signals are assigned to the GPTA0. Therefore, a total of 2 registers control the connections within the on-chip gating and trigger signal multiplexer of the GPTA0 unit.

Further sixteen on-chip trigger and gating signals are assigned to the GPTA1. Therefore, a total of 2 registers control the connections within the on-chip gating and trigger signal multiplexer of the GPTA1 unit.

The OTMCR0 register control the OTMG output lines TRIG00 to TRIG07. The OTMCR1 register control the OTMG output lines TRIG10 to TRIG17. **Table 22-12** lists all On-Chip Trigger and Gating Signal Multiplexer Control Registers with its control functions. Please note that the On-Chip Trigger and Gating Signal Multiplexer Control Registers are not directly accessible but must be written or read using a FIFO array structure as described on **Page 22-121**.

Table 22-12 On-Chip Trigger/Gating Multiplexer Control Register Assignments

I/O Group Signal		OTMG Input Signal	Controlled by Multiplexer Control Register	Trigger/Gating Signal (x=0-7)	Selectable Groups via OMGng
IOG0	OUT00	IN00	OTMCR0	TRIG0x	GTCG0, LTCG0, LTCG4
IOG0	OUT02	IN01	OTMCR0	TRIG0x	GTCG0, LTCG0, LTCG4
IOG1	OUT08	IN02	OTMCR0	TRIG0x	GTCG1, LTCG1, LTCG5
IOG1	OUT10	IN03	OTMCR0	TRIG0x	GTCG1, LTCG1, LTCG5
IOG2	OUT16	IN04	OTMCR0	TRIG0x	GTCG2, LTCG2, LTCG6
IOG2	OUT19	IN05	OTMCR0	TRIG0x	GTCG2, LTCG2, LTCG6
IOG3	OUT24	IN06	OTMCR0	TRIG0x	GTCG3, LTCG3, LTCG7
IOG3	OUT27	IN07	OTMCR0	TRIG0x	GTCG3, LTCG3, LTCG7
IOG0	OUT01	IN10	OTMCR1	TRIG1x	GTCG0, LTCG0, LTCG4
IOG0	OUT03	IN11	OTMCR1	TRIG1x	GTCG0, LTCG0, LTCG4
IOG1	OUT09	IN12	OTMCR1	TRIG1x	GTCG1, LTCG1, LTCG5
IOG1	OUT11	IN13	OTMCR1	TRIG1x	GTCG1, LTCG1, LTCG5
IOG2	OUT18	IN14	OTMCR1	TRIG1x	GTCG2, LTCG2, LTCG6
IOG3	OUT25	IN15	OTMCR1	TRIG1x	GTCG3, LTCG3, LTCG7
IOG3	OUT26	IN16	OTMCR1	TRIG1x	GTCG3, LTCG3, LTCG7
IOG3	OUT28	IN17	OTMCR1	TRIG1x	GTCG3, LTCG3, LTCG7

22.3.4.4 GTC Input Multiplexer Selection

The GTC input multiplexer as shown in **Figure 22-64** and **Figure 22-72** connects the 56 (= 7 × 8) input lines of the I/O groups, the 64 LTC output lines of the eight LTC groups, the six FPC output lines, and two internal input lines INT[1:0] with the 32 (= 4 × 8) LTC input lines, organized into eight LTC groups.

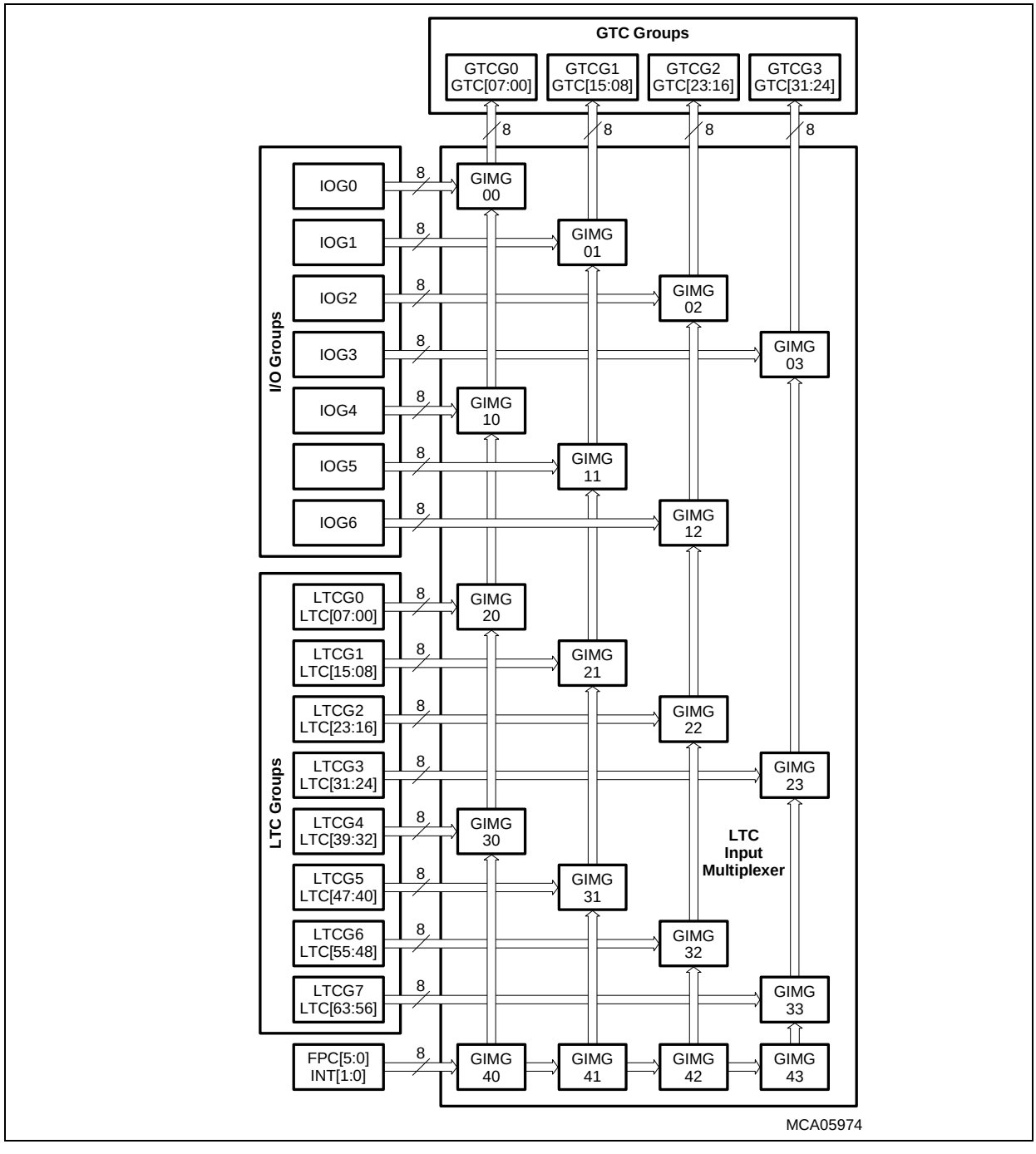


Figure 22-72 GTC Input Multiplexer

The GTC input multiplexer contains GTC input Multiplexer Groups (GIMGs) that connect the I/O groups or Local Timer Cells with the input lines of the GTC input lines, organized into four GTC groups with 8 cells each. GTC input Multiplexer Group are grouped into seven IOGs (IOG[6:0]) with eight lines each and eight LTC groups (LTCG[7:0]) with 8 cells each. One special FPC/INT group with eight outputs is established that combines the six FPC outputs and two internal input lines INT[1:0] as a group of GIMGs inputs.

Figure 22-73 shows the logical structure of a GIMG.

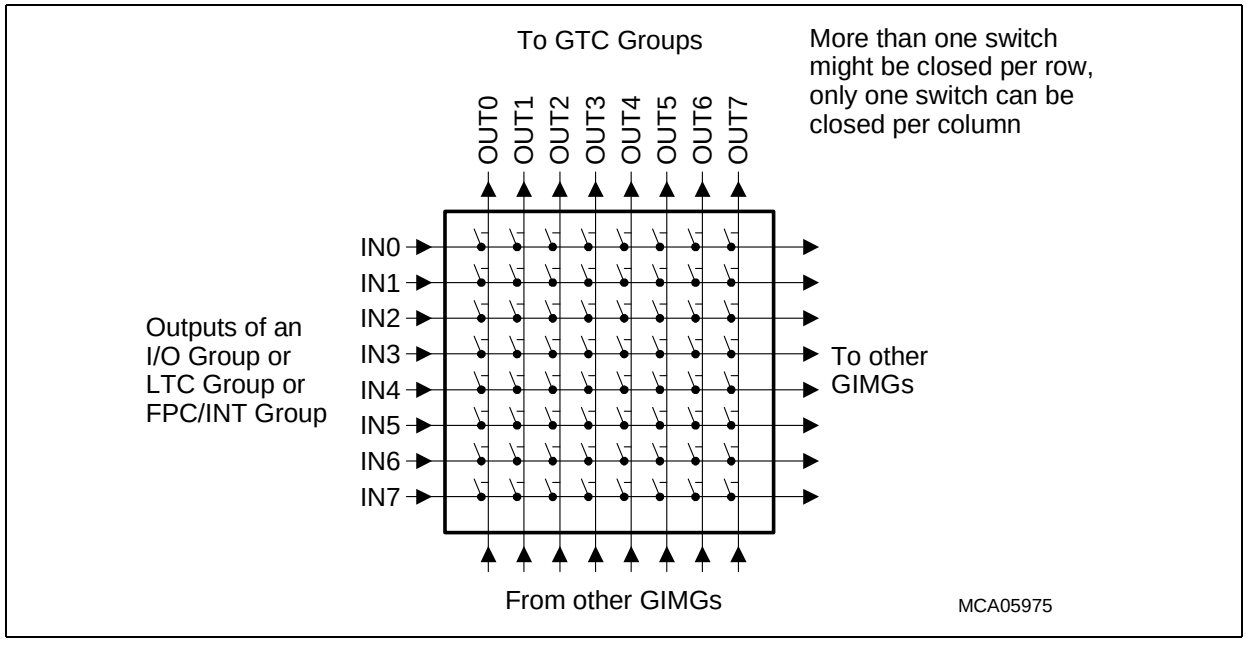


Figure 22-73 GTC Input Multiplexer Group (GIMG) Structure

Rules for connections to GTC Input Multiplexer Group GIMG:

- Within a I/O group or LTC group, the line or the output of the cell with the lowest index number is connected to GIMG input line IN0. The remaining lines, cells or lines of a group are connected to GIMG input lines IN1 to IN7 with ascending index numbers. At the FPC/INT group, FPC[5:0] is connected to IN[5:0] and INT[1:0] is connected to IN[7:6].
  Example: for GIMG23 (see Figure 22-72), the cells LTC24 up to LTC31 are wired to the GIMG23 input lines IN0 to line IN7.
- Multiplexer output OUT0 is always connected to the input of a GTC group with the lowest index. The remaining output lines OUT1 to OUT7 are connected to the GTC inputs with ascending index.
  Example: for GIMG23 (see Figure 22-72), the outputs OUT0 to OUT7 are wired to the inputs of GTC16 to GTC23.
- A GTC input can be connected either to an I/O group output, or to an LTC output, or to an FPC/INT output. This is guaranteed by the GIMG control register layout. Otherwise, short circuits and unpredictable behavior would occur. In contrast, it is permissible for an I/O group output, or an LTC output, or an FPC/INT output to be connected to more than one GTC input.

The GTC input multiplexer group configuration is based on the following principles:

- Each GIMG is referenced with two index variables: n and g (GIMGng)
- Index n is a group number. I/O groups IOG[3:0] have group number 0, I/O groups IOG[6:4] have group number 1, Local Timer Cell Groups LTCG[3:0] have group

number 2, Local Timer Cell Groups LTCG[7:4] have group number 3, and the FPC/INT group has group number 4.

- Index g indicates the number of the GTC group g (g = 0-3) to which the outputs of the input multiplexer group GIMGng are connected.

The GTC input multiplexer logic as seen for programming is shown in **Figure 22-74**. With this logic, five group signals (from an I/O group, LTC group, or FPC/INT group) are always combined to one output line that leads to the input of a GTC of GTC group g. For example, when looking at **Figure 22-73**, each of the eight GTC input multiplexer output lines to GTC group GTCG2 is connected via five OMGn2 (n = 0-4) with the eight outputs of two I/O group (IOG2 and IOG6), two LTC groups (LTCG2 and LTCG6), and the FPC/INT group.

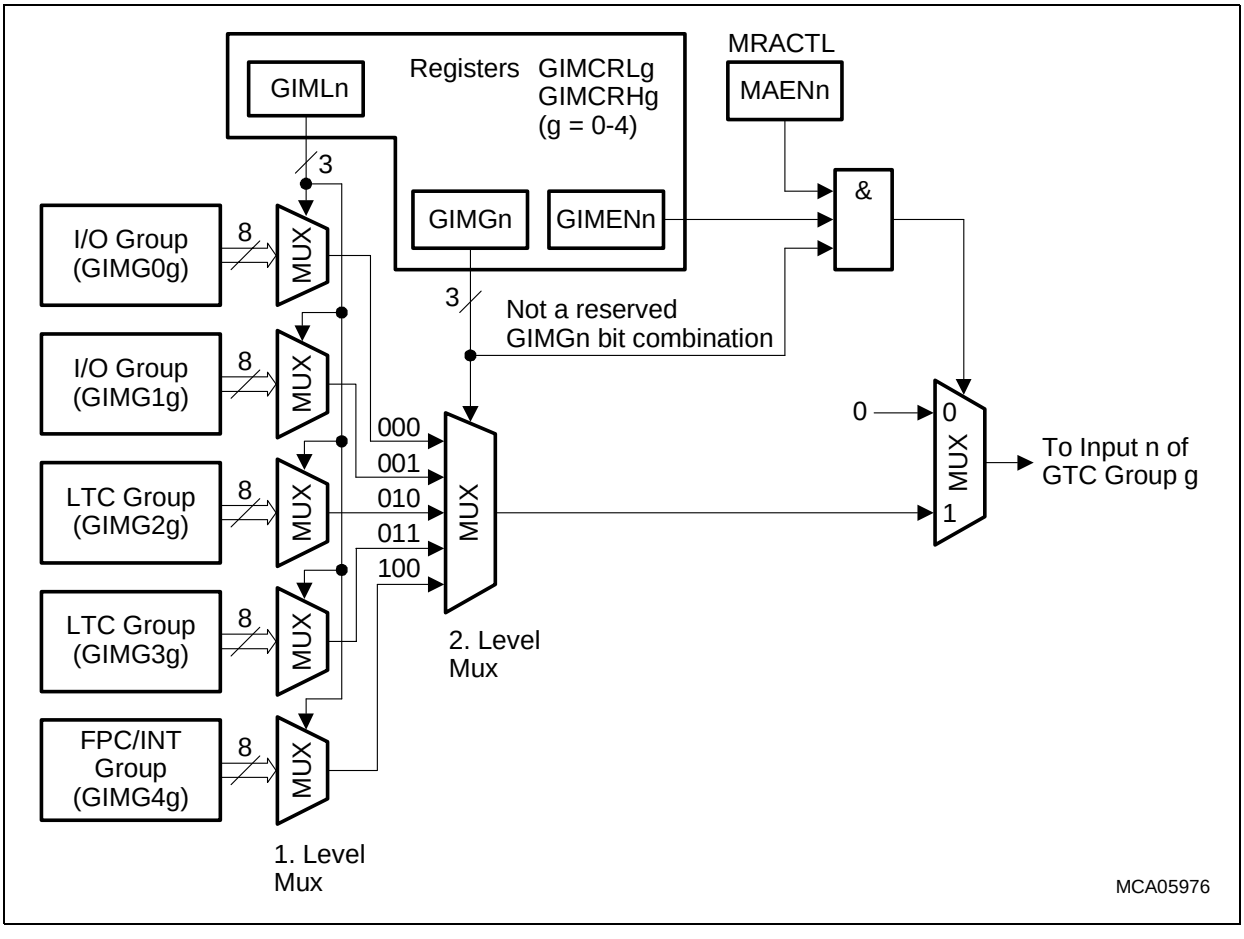


Figure 22-74 GTC Input Multiplexer Group (Programmer's View)

The 1. level multiplexer is built up by five 8:1 multiplexers that are controlled in parallel by bit field GIMLn. Bit field GIMGn controls the 2. level multiplexer and connects one of the 1. level multiplexer outputs to one of the GIMGng outputs. The output of the 2. level multiplexer is only connected to the input of an GTC if bit GIMENn (enable multiplexer connection) is set, and bit MRACTL.AEN is set (multiplexer array enabled), and no

reserved bit combination of GIMGn is selected. If one of these conditions is not true, the corresponding GIMG output will be held at a low level.

If one of these bit is not set, the corresponding GTC input will be held at a low level.

Two GTC Input Multiplexer Control Registers, GIMCRL and GIMCRH (see also **Page 22-215**), are assigned to each of the GTC groups. Therefore, a total of eight registers control the connections within the GTC input multiplexer of the GPTA®v5 module.

The GIMCRL registers control the GIMG output lines 0 to 3 and the GIMCRH registers control the GIMG output lines 4 to 7. **Table 22-13** lists all of the GTC Input Multiplexer Control Registers with its control functions. Please note that all GTC Input Multiplexer Control Registers are not directly accessible but must be written or read using a FIFO array structure as described on **Page 22-121**.

Table 22-13 GTC Input Multiplexer Control Register Assignments

GTC Group and GTCs		Controlled by Multiplexer Control Register	Selectable Groups via GIMGng
GTCG0	GTC[03:00]	GIMCRL0	IOG0, IOG4, LTCG0, LTCG4, FPC/INT
	GTC[07:04]	GIMCRH0	
GTCG1	GTC[11:08]	GIMCRL1	IOG1, IOG5, LTCG1, LTCG5, FPC/INT
	GTC[15:12]	GIMCRH1	
GTCG2	GTC[19:16]	GIMCRL2	IOG2, IOG6, LTCG2, LTCG6, FPC/INT
	GTC[23:20]	GIMCRH2	
GTCG3	GTC[27:24]	GIMCRL3	IOG3, LTCG3, LTCG7, FPC/INT
	GTC[31:28]	GIMCRH3	

22.3.4.5 LTC Input Multiplexer Selection

The LTC input multiplexer as shown in **Figure 22-64** and **Figure 22-75** connects the 56 (= 7 × 8) input lines of the I/O groups, the 32 (= 4 × 8) GTC output lines of the GTC groups, the eight clock bus lines, or the four PDL output lines with four internal input lines INT[3:0] with the 64 (= 8 × 8) LTC input lines, organized into eight LTC groups.

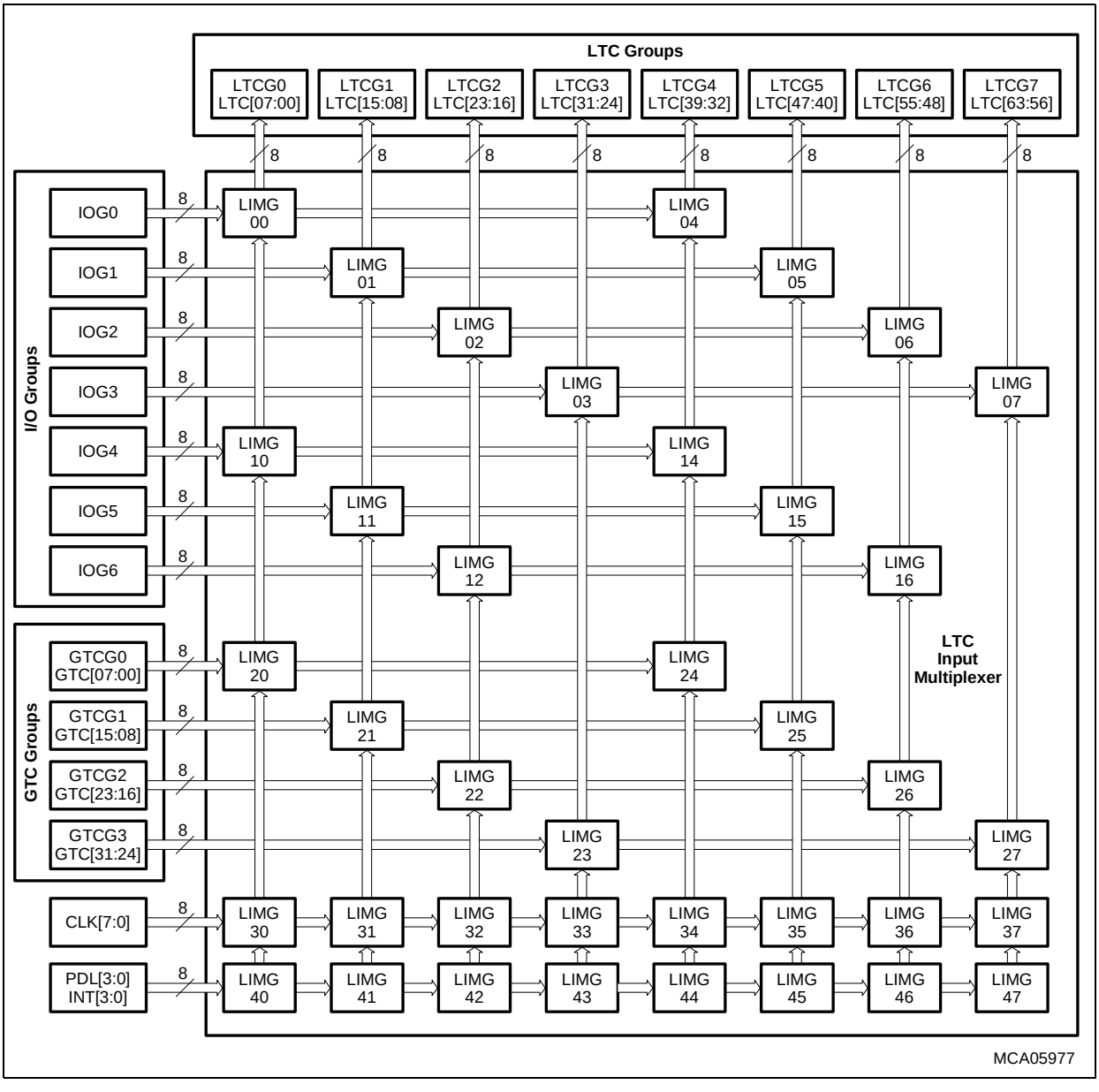


Figure 22-75 LTC Input Multiplexer

The LTC input multiplexer contains LTC input Multiplexer Groups (LIMGs) that connect the I/O groups or Global Timer Cells with the input lines of the LTCs, organized into eight LTC groups with 8 cells each. IOGs and GTCs are grouped into seven IOGs (IOG[6:0]) with eight lines each and four GTC groups (GTCG[3:0]) with 8 cells each. Two special groups are available: a clock group with eight lines representing the clock bus lines CLK[7:0] of the clock distribution cells and a PDL/INT group with eight outputs that combines the four PDL outputs and four internal input lines INT[3:0] as a group of LIMGs inputs.

Note: GPTA0 generates the clock bus lines CLK[7:0] and the four PDL outputs.

Figure 22-76 shows the logical structure of a LIMG.

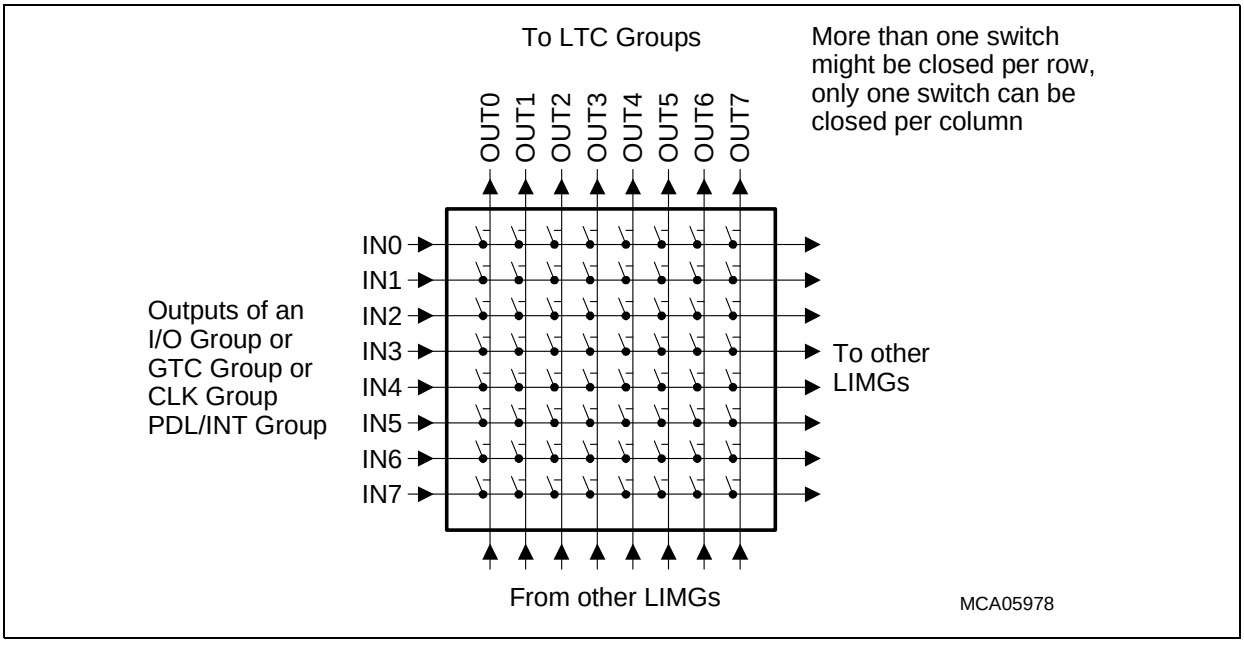


Figure 22-76 LTC Input Multiplexer Group (LIMG) Structure

Rules for connections to LTC Input Multiplexer Group LIMG:

- Within a I/O group or GTC group, the line or the output of the cell with the lowest index number is connected to LIMG input line IN0. The remaining lines, cells or lines of a group are connected to LIMG input lines IN1 to IN7 with ascending index numbers. At the clock group, CLK0 is connected to IN0 and the remaining clock lines are connected to LIMG input lines IN1 to IN7 with ascending index numbers. At the PDL/INT group, PDL[3:0] (see **Page 22-22**) is connected to IN[3:0] and INT[3:0] is connected to IN[7:4].
  Example: for LIMG23 (see **Figure 22-75**), the cells GTC24 up to GTC31 are wired to the LIMG23 input lines IN0 to line IN7.
- Multiplexer output OUT0 is always connected to the input of an LTC group with the lowest index. The remaining output lines OUT1 to OUT7 are connected to the LTC inputs with ascending index.

Example: for LIMG23 (see **Figure 22-75**), the outputs OUT0 to OUT7 are wired to the inputs of LTC24 to GTC31.

- An LTC input can be connected either to an I/O group output, or to an GTC output, or to a clock bus output, or to an PDL/INT output. This is guaranteed by the LIMG control register layout. Otherwise, short circuits and unpredictable behavior would occur. In contrast, it is permitted that an I/O group output, or an GTC output, or an PDL/INT output is connected to more than one LTC input.

The LTC input multiplexer group configuration is based on the following principles:

- Each LIMG is referenced with two index variables: n and g (LIMGng)
- Index n is a group number. I/O groups IOG[3:0] have group number 0, I/O groups IOG[6:4] have group number 1, Global Timer Cell Groups GTCG[3:0] have group number 2, clock bus lines CLK[7:0] have group number 3, and the PDL/INT group has group number 4.
- Index g indicates the number of the LTC group g (g = 0-7) to which the outputs of the input multiplexer group LIMGng are connected.

The LTC input multiplexer logic as seen for programming is shown in **Figure 22-77**. With this logic, five group signals (from an I/O group, GTC group, clock group, or PDL/INT group) are always combined to one output line that leads to the input of an LTC of LTC group g. For example, when looking at **Figure 22-75**, each of the eight LTC input multiplexer output lines to LTC group LTCG2 is connected via five LIMGn2 (n = 0-4) with the eight outputs of two I/O group (IOG2 and IOG6), one GTC group (GTCG2), the clock group, and the PDL/INT group.

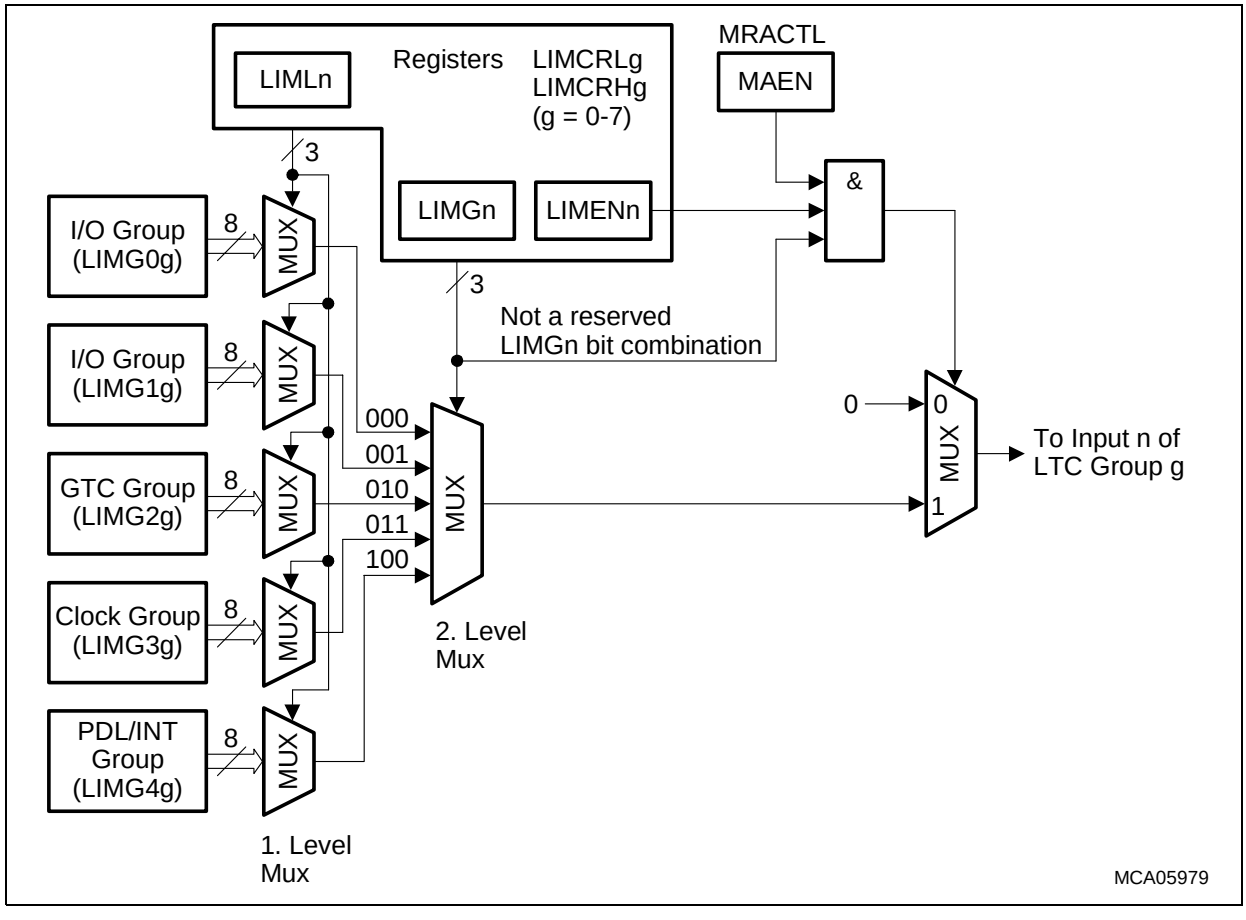


Figure 22-77 LTC Input Multiplexer Group (Programmer's View)

The 1. level multiplexer is built up by five 8:1 multiplexers that are controlled in parallel by bit field LIMLn. Bit field LIMGn controls the 2. level multiplexer and connects one of the 1. level multiplexer outputs to one of the LIMGng outputs. The output of the 2. level multiplexer is connected only to the input of an LTC if bit LIMENn is set (enable multiplexer connection), and bit MRACTL.AEN is set (multiplexer array enabled), and no reserved bit combination of LIMGn is selected. If one of these conditions is not true, the corresponding LTC input will be held at a low level.

Two LTC Input Multiplexer Control Registers, LIMCRL and LIMCRH (see also **Page 22-219**), are assigned to each of the LTC groups. Therefore, in total sixteen registers control the connections within the LTC input multiplexer of the GPTA®v5 module.

The LIMCRL registers control the LIMG output lines 0 to 3 and the GIMCRH registers control the LIMG output lines 4 to 7. **Table 22-14** lists all LTC Input Multiplexer Control Registers with its control functions. Please note that all LTC Input Multiplexer Control Registers are not directly accessible but must be written or read using a FIFO array structure as described on **Page 22-121**.

Table 22-14 LTC Input Multiplexer Control Register Assignments

LTC Group and LTCs		Controlled by Register	Selectable Groups via LIMGng
LTCG0	LTC[03:00]	LIMCRL0	IOG0, IOG4, GTCG0, CLOCK, PDL/INT
	LTC[07:04]	LIMCRH0	
LTCG1	LTC[11:08]	LIMCRL1	IOG1, IOG5, GTCG1, CLOCK, PDL/INT
	LTC[15:12]	LIMCRH1	
LTCG2	LTC[19:16]	LIMCRL2	IOG2, IOG6, GTCG2, CLOCK, PDL/INT
	LTC[23:20]	LIMCRH2	
LTCG3	LTC[27:24]	LIMCRL3	IOG3, GTCG3, CLOCK, PDL/INT
	LTC[31:28]	LIMCRH3	
LTCG4	LTC[35:32]	LIMCRL4	IOG0, IOG4, GTCG0, CLOCK, PDL/INT
	LTC[39:36]	LIMCRH4	
LTCG5	LTC[43:40]	LIMCRL5	IOG1, IOG5, GTCG1, CLOCK, PDL/INT
	LTC[47:44]	LIMCRH5	
LTCG6	LTC[51:48]	LIMCRL6	IOG2, IOG6, GTCG2, CLOCK, PDL/INT
	LTC[55:52]	LIMCRH6	
LTCG7	LTC[59:56]	LIMCRL7	IOG3, GTCG3, CLOCK, PDL/INT
	LTC[63:60]	LIMCRH7	

22.3.4.6 Multiplexer Register Array Programming

A total of 54 control registers are required to program the configuration of the output multiplexer, the On-chip trigger and gating multiplexer, and the two input multiplexers of the Input/Output Line Sharing Block. These IOLS control registers are combined into a Multiplexer Register Array FIFO that can only be read or written sequentially. Therefore, the control registers values cannot be accessed directly but must be accessed in a specific sequential order.

Three registers are available for controlling the Multiplexer Register Array:

- Multiplexer Register Array Control Register MRACTL
- Multiplexer Register Array Data In Register MRADIN
- Multiplexer Register Array Data Out Register MRADOUT

Figure 22-78 shows the structure of the multiplexer array FIFO with the arrangement of the multiplexer control registers.

For programming of the multiplexer array FIFO, the following steps must be executed:

1. Disable interconnections of the multiplexer array by writing MRACTL.MAEN = 0 (default after reset). The multiplexer array is disabled, all cell input lines are driven with 0, and device pins assigned to GPTA®v5 I/O lines or output lines are disconnected.
2. Reset the write cycle counter to 0 by writing MRACTL.WCRES = 1.
3. Write sequentially the multiplexer control register contents one after the other (54 values) into MRADIN, starting with the register values for OTMCR1, OTMCR0, … up to GIMCRH0, GIMCRL0 (see **Figure 22-78**). After the first MRADIN write operation, the contents for OTMCR1 is at FIFO position 1. With each following MRADIN write operation, it becomes shifted one FIFO position upwards. After the 54. MRADIN write operation, the OTMCR1 value is at its final position. The contents of FIFO position 54 can be read via register MRADOUT. With each MRADIN write operation the write cycle counter MRACTL.FIFOFILLCNT is incremented by 1. After all FIFO entries have been written, the FIFO is locked, bit MRACTL.FIFOFULL is set, and further MRADIN write operations are discarded until bit MRACTL.WCRES is written again with a 0.
4. Enable the multiplexer array by writing MRACTL.MAEN = 1. This establishes and enables all programmed interconnections.

To check the FIFO contents, the FIFO can be written a second time. At this check MRADIN is written before MRADOUT is read. This will return the FIFO contents of the first write sequence in the order of OTMCR1, OTMCR0, …, GIMCRH0, GIMCRL0.

Before disabling the multiplexer array FIFO, GPTA®v5 output pins that are already enabled as GPTA®v5 output should be switched to GPIO function to avoid output spikes. After enabling the multiplexer array FIFO again, the GPTA®v5 output can be switched again back to GPTA®v5 output function.

Shifting the write data through the FIFO requires a few clock cycles. When new data becomes written before the FIFO is ready to accept them, wait states will be inserted into the write access.

If the OMCRLg register bit field OMGn of the multiplexer array is programmed with an invalid (reserved) value, the related outputs will be forced to 0. When the array is disabled (MRACTL.MAEN = 0), all cell inputs and outputs are disconnected from the GPIO lines and are driven with 0.

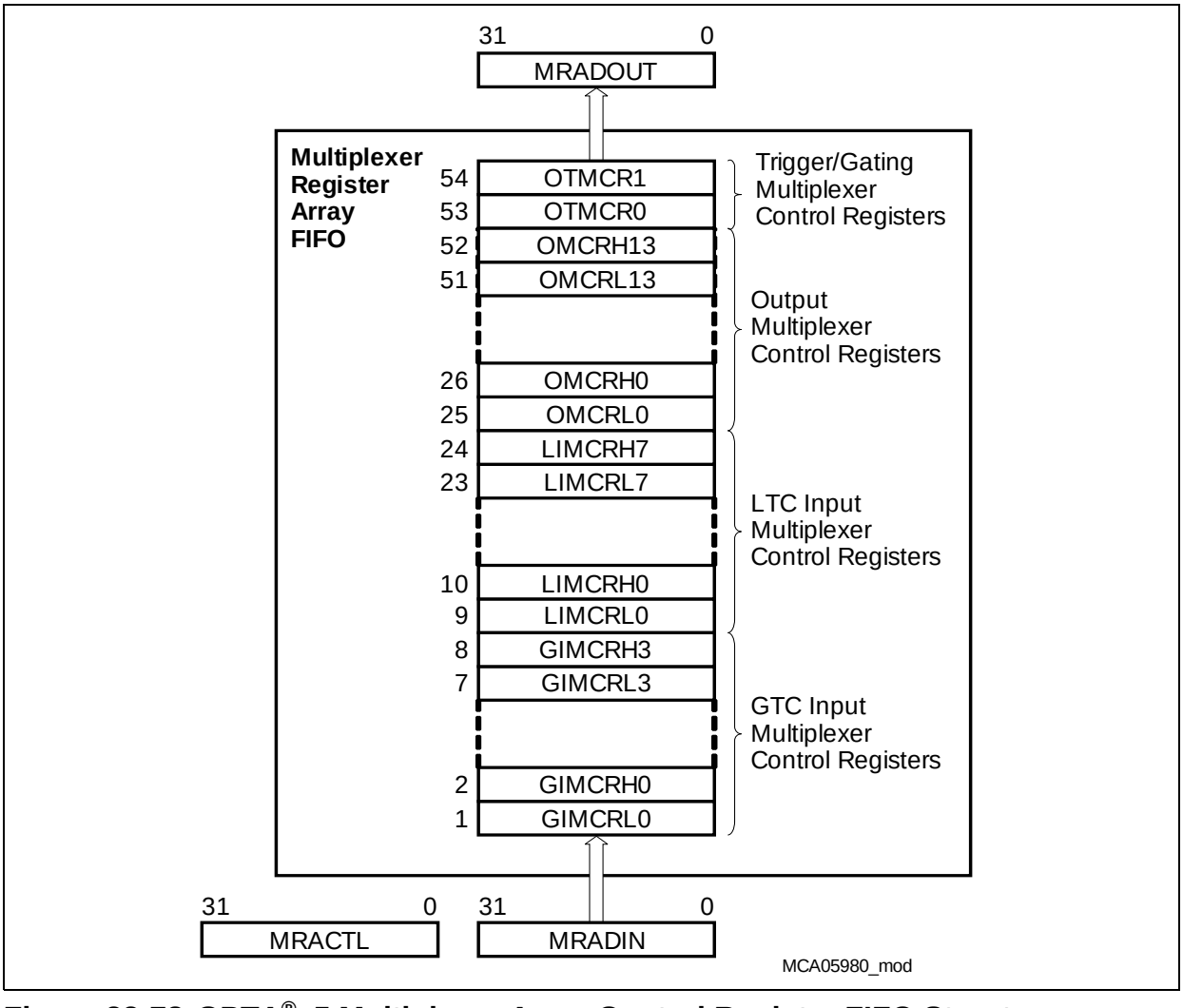


Figure 22-78 GPTA®v5 Multiplexer Array Control Register FIFO Structure

22.3.5 Interrupt Sharing Block (IS)

The GPTA®v5 provides 111 service request sources. These service request sources are generated by different cell types, as shown in **Table 22-15**.

Table 22-15 GPTA®v5 Number of Service Request Sources

Cell Type	Number of Cells	Number of Service Request Sources/Cell	Total Number of Request Sources
DCM	4	3	12
PLL	1	1	1
GT	2	1	2
GTC	32	1	32
LTC	64	1	64
			Sum: 111

To reduce hardware and software overhead, at maximum five request sources are combined together in service request groups. A service request group has up to five service request inputs and one service request output SRy which is typically connected outside the GPTA®v5 kernel with a standard interrupt node y and controlled by its SRCy register.

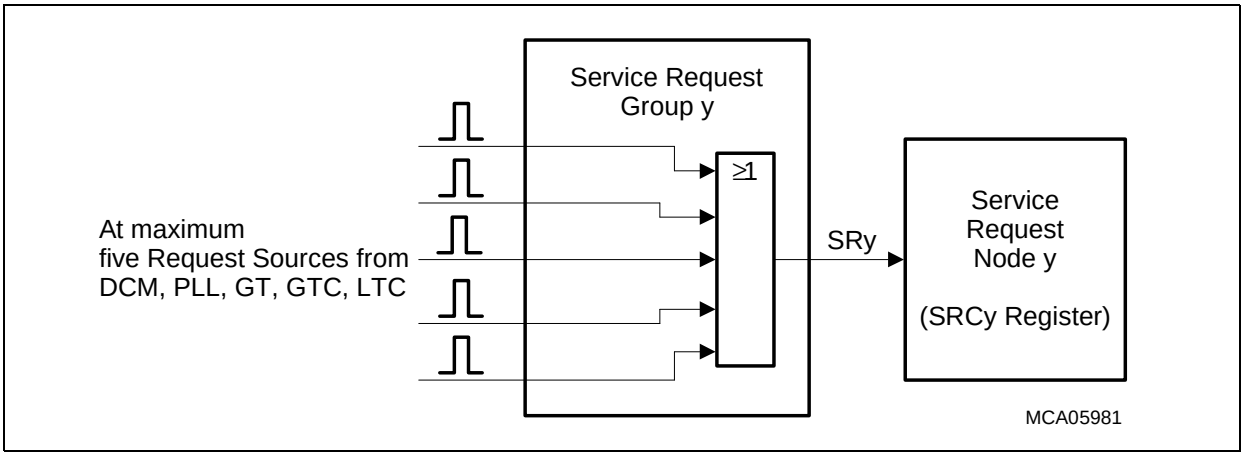


Figure 22-79 Service Request Groups

The bits in the Service Request State Registers (SRSSx and SRSCx) are service request status flags that are set by hardware (type “h”) when the related event occurs. Each GPTA®v5 service request source has its own service request flag. This flag is normally set by hardware but can be set and reset by software. Each service request status flag can be read twice, at the same bit location in the SRSCx register and in the SRSSx register, and cleared or set by software when writing to the corresponding request bit in SRSCx or SRSSx. When writing to SRSCx or SRSSx, several request flags

can be cleared at once by one write operation. Request flags of bit positions that are written with 0 are not changed. This feature allows fast, simple clearing or setting of request flags without affecting other bits in the same service request state register.

Note that service request flag is always set independently of whether it is enabled or disabled by the related cell; but the service request line to the corresponding service request group becomes only active if the corresponding service request is enabled by the related cell. Finally, each service request group y must be enabled by the enable flag that is located in SRC register y.

Table 22-16 lists all of the service requests groups with its request sources. Note that service requests of GTCs with an odd index number k can be individually redirected via register SRNR to a service request group that is assigned mainly to four LTCs.

Table 22-16 GPTA®v5 Service Request Groups

Service Request Group Number y	Request Source 1	Request Source 2	Request Source 3	Request Source 4	Request Source 5
00	DCM0 rising	DCM0 falling	DCM0 comp.	–	–
01	DCM1 rising	DCM1 falling	DCM1 comp.	–	–
02	DCM2 rising	DCM2 falling	DCM2 comp.	–	–
03	DCM3 rising	DCM3 falling	DCM3 comp.	–	–
04	PLL	–	–	–	–
05	GT0	GT1	–	–	–
06	GTC00	GTC01[1)]	–	–	–
07	GTC02	GTC03[1)]	–	–	–
08	GTC04	GTC05[1)]	–	–	–
09	GTC06	GTC07[1)]	–	–	–
10	GTC08	GTC09[1)]	–	–	–
11	GTC10	GTC11[1)]	–	–	–
12	GTC12	GTC13[1)]	–	–	–
13	GTC14	GTC15[1)]	–	–	–
14	GTC16	GTC17[1)]	–	–	–
15	GTC18	GTC19[1)]	–	–	–
16	GTC20	GTC21[1)]	–	–	–
17	GTC22	GTC23[1)]	–	–	–
18	GTC24	GTC25[1)]	–	–	–
19	GTC26	GTC27[1)]	–	–	–

Table 22-16 GPTA®v5 Service Request Groups (cont'd)

Service Request Group Number y	Request Source 1	Request Source 2	Request Source 3	Request Source 4	Request Source 5
20	GTC28	GTC29[1)]	–	–	–
21	GTC30	GTC31[1)]	–	–	–
22	LTC00	LTC01	LTC02	LTC03	GTC01[2)]
23	LTC04	LTC05	LTC06	LTC07	GTC03[2)]
24	LTC08	LTC09	LTC10	LTC11	GTC05[2)]
25	LTC12	LTC13	LTC14	LTC15	GTC07[2)]
26	LTC16	LTC17	LTC18	LTC19	GTC09[2)]
27	LTC20	LTC21	LTC22	LTC23	GTC11[2)]
28	LTC24	LTC25	LTC26	LTC27	GTC13[2)]
29	LTC28	LTC29	LTC30	LTC31	GTC15[2)]
30	LTC32	LTC33	LTC34	LTC35	GTC17[2)]
31	LTC36	LTC37	LTC38	LTC39	GTC19[2)]
32	LTC40	LTC41	LTC42	LTC43	GTC21[2)]
33	LTC44	LTC45	LTC46	LTC47	GTC23[2)]
34	LTC48	LTC49	LTC50	LTC51	GTC25[2)]
35	LTC52	LTC53	LTC54	LTC55	GTC27[2)]
36	LTC56	LTC57	LTC58	LTC59	GTC29[2)]
37	LTC60	LTC61	LTC62	LTC63	GTC31[2)]

1) Redirection bit SRNR.GTCkR = 0 (k = 01, 03, 05, … 27, 29, 31).

2) Redirection bit SRNR.GTCkR = 1 (k = 01, 03, 05, … 27, 29, 31).

22.3.6 Pseudo Code Description of GPTA®v5 Kernel Functionality

This section describes the functional algorithms of the GPTA®v5 cells in a pseudo code language.

22.3.6.1 FPC Algorithm

FPCk_Control_Logic() “to be performed every GPTA®v5 clock”

```
switch (FPCk.Mode)
  case PRESCALER_RISING:
    if (FPCk.Rising_Edge) then
      Prescaler()
    endif
    break
  case PRESCALER_FALLING:
    if (FPCk.Falling_Edge) then
      Prescaler()
    endif
    break
  case DELAYED_FILTER_BOTH:
    Delayed_Filter()
    break
  case IMMEDIATE_FILTER_BOTH:
  case IMMEDIATE_FILTER_RISING:
  case IMMEDIATE_FILTER_FALLING:
    Immediate_Filter()
    break
  case MIXED_FILTER_RISING_DELAYED:
    if (FPCk.Signal_Filtered == 0) then
      Delayed_Filter()
    else
      Immediate_Filter()
    endif
    break
  case MIXED_FILTER_RISING_IMMEDIATE:
    if (FPCk.Signal_Filtered == 0) then
      Immediate_Filter()
    else
      Delayed_Filter()
    endif
    break
endswitch
```

Delayed_Filter()

```
if (FPCk.Filter_Clock[n]) then
  if (FPCk.Timer >= FPCk.Compare_Value) then
    if (FPCk.Compare_Value == 0) then    //by-pass
      if (FPCk.Signal_Output.Level != FPCk.Signal_Input[m]) then
        generate pulse on FPCk.Signal_Output.Transition
        FPCk.Signal_Output.Level = FPCk.Signal_Input[m]
        FPCk.Signal_Filtered = FPCk.Signal_Output.Level
      endif
    else    //delay time is over
      generate pulse on FPCk.Signal_Output.Transition
      FPCk.Signal_Output.Level = !FPCk.Signal_Output.Level
      FPCk.Signal_Filtered = FPCk.Signal_Output.Level
    endif
    FPCk.Timer = 0
  else
    if (FPCk.Timer != 0) then      //delay time is running
      if (FPCk.Rising_Edge is detected) then      //edge detection done at clock input
        FPCk.Rising_Edge_Glitch = 1
      else
        if (FPCk.Falling_Edge is detected) then     //edge detection done at clock input
          FPCk.Falling_Edge_Glitch = 1
        endif
      endif
    endif
    if (FPCk.Signal_Output.Level != FPCk.Signal_Input[m])
    then    //expected level
      FPCk.Timer ++
    else    //unexpected level
      if (FPCk.Timer != 0) then
        if (FPCk.Reset_Timer) then
          FPCk.Timer = 0
        else
          FPCk.Timer --
        endif
      endif
    endif
  endif
endif
```

Prescaler()

```
if (FPCk.Timer >= FPCk.Compare_Value) then
  generate pulse on FPCk.Signal_Output.Transition
  generate pulse on FPCk.Signal_Output.Level
  FPCk.Timer = 0
else
  FPCk.Timer ++
endif
```

Immediate_Filter()

```
if (FPCk.Filter_Clock[n]) then
  if (FPCk.Timer == 0) then
    if (FPCk.Signal_Output.Level != FPCk.Signal_Input[m]) ) then    //change detected
      generate pulse on FPCk.Signal_Output.Transition
      FPCk.Signal_Output.Level = FPCk.Signal_Input[m]
      if ( (FPCk.Compare_Value == 0) or
      ((FPCk.Mode == IMMEDIATE_FILTER_RISING) and !FPCk.Signal_Input[m]) or
      ((FPCk.Mode == IMMEDIATE_FILTER_FALLING) and FPCk.Signal_Input[m]) )
      then    //by-pass
        FPCk.Signal_Filtered = FPCk.Signal_Output.Level
      else    //start delay time
        FPCk.Timer ++
      endif
    endif
  else
    if (FPCk.Timer >= FPCk.Compare_Value) then    //delay time is over
      FPCk.Timer = 0
      FPCk.Signal_Filtered = FPCk.Signal_Output.Level
    else    //delay time is running
      FPCk.Timer ++
      if (FPCk.Rising_Edge) then
        FPCk.Rising_Edge_Glitch = 1
      else
        if (FPCk.Falling_Edge) then
          FPCk.Falling_Edge_Glitch = 1
        endif
      endif
    endif
  endif
endif
```

Variables

Input, Local, Output variables of the cell (I, L, O)

Name k = [0 to 5] for FPC m = [0 to 5] for Signal n = [0 to 3] for Clock	Short Name (*)FPC	Used (ILO)	Comment
FPCk.Signal_Input[m]	*SINm	I	Signal input selected by FPCk.Input_Source
FPCk.Filter_Clock[n]	*CINn	I	Filter Clock selected by FPCk.Clock_Source
FPCk.Rising_Edge	*RE	L	Signal coming from the edge detect
FPCk.Falling_Edge	*FE	L	Signal coming from the edge detect
FPCk.Signal_Filtered	*SF	L	Filtered output signal (after delay time), initialized to 0 at reset
FPCk.Signal_Output.Transition FPCk.Signal_Output.Level	*SOTk *SOLk	O	Transition/Level of the output signal, initialized to 0 at reset

Global variables

Name k = [0 to 5] for FPC	Short Name (*)FPC	Size (bits)	Function
FPCk.Mode	*MODk	3	Selects one of these modes: DELAYED_FILTER_BOTH IMMEDIATE_FILTER_BOTH IMMEDIATE_FILTER_RISING IMMEDIATE_FILTER_FALLING MIXED_FILTER_RISING_DELAYED MIXED_FILTER_RISING_IMMEDIATE PRESCALER_RISING PRESCALER_FALLING
FPCk.Input_Source	*IPSk	3	Selects input signal
FPCk.Clock_Source	*CLKk	2	Selects FPC clock
FPCk.Rising_Edge_Glitch	*REGk	1	Bit is set when rising edge glitch occurs during filtering
FPCk.Falling_Edge_Glitch	*FEGk	1	Bit is set when falling edge glitch occurs during filtering
FPCk.Timer	*TIMk	16	Timer value
FPCk.Reset_Timer	*RTGk	1	Reset timer on glitch in Delayed Filter Mode
FPCk.Compare_Value	*CMPk	16	Compare value

22.3.6.2 PDL-Algorithm

PDLx_Control_Logic() “to be performed every GPTA®v5 clock”

```
if (x == 0) then
  S1.Level = FPC0.Signal_Output.Level
  S1.Transition = FPC0.Signal_Output.Transition
  S2.Level = FPC1.Signal_Output.Level
  S2.Transition = FPC1.Signal_Output.Transition
  S3.Level = FPC2.Signal_Output.Level
  S3.Transition = FPC2.Signal_Output.Transition
else    //x = 1
  S1.Level = FPC3.Signal_Output.Level
  S1.Transition = FPC3.Signal_Output.Transition
  S2.Level = FPC4.Signal_Output.Level
  S2.Transition = FPC4.Signal_Output.Transition
  S3.Level = FPC5.Signal_Output.Level
  S3.Transition = FPC5.Signal_Output.Transition
endif

if (PDLx.Three_Sensors_Enable) then
  Three_Sensors()
else
  Two_Sensors()
endif

if (PDLx.Mux) then
  PDLx.Signal_Output1.Level = 1
  if (PDLx.Signal_Forward or PDLx.Signal_Backward) then
    PDLx.Signal_Output1.Transition = 1
  else
    PDLx.Signal_Output1.Transition = 0
  endif
else
  PDLx.Signal_Output1.Transition = S1.Transition
  PDLx.Signal_Output1.Level = S1.Level
endif
```

Two_Sensors()

```
if ( ( S1.Level and !S2.Level and S1.Transition) or
     ( S1.Level and  S2.Level and S2.Transition) or
     (!S1.Level and  S2.Level and S1.Transition) or
     (!S1.Level and !S2.Level and S2.Transition) ) then
   generate pulse on PDLx.Signal_Forward
else
   if ( ( S1.Level and  S2.Level and S1.Transition) or
        (!S1.Level and  S2.Level and S2.Transition) or
        (!S1.Level and !S2.Level and S1.Transition) or
        ( S1.Level and !S2.Level and S2.Transition) ) then
      generate pulse on PDLx.Signal_Backward
   endif
endif

PDLx.Signal_Output2.Level = S3.Level
PDLx.Signal_Output2.Transition = S3.Transition
```

Three_Sensors()

```
if ( ( S1.Level and !S2.Level and  S3.Level and S1.Transition) or
     ( S1.Level and !S2.Level and !S3.Level and S3.Transition) or
     ( S1.Level and  S2.Level and !S3.Level and S2.Transition) or
     (!S1.Level and  S2.Level and !S3.Level and S1.Transition) or
     (!S1.Level and  S2.Level and  S3.Level and S3.Transition) or
     (!S1.Level and !S2.Level and  S3.Level and S2.Transition) ) then
   generate pulse on PDLx.Signal_Forward
else
   if ( ( S1.Level and  S2.Level and !S3.Level and S1.Transition) or
        (!S1.Level and  S2.Level and !S3.Level and S3.Transition) or
        (!S1.Level and  S2.Level and  S3.Level and S2.Transition) or
        (!S1.Level and !S2.Level and  S3.Level and S1.Transition) or
        ( S1.Level and !S2.Level and  S3.Level and S3.Transition) or
        ( S1.Level and !S2.Level and !S3.Level and S2.Transition) ) then
      generate pulse on PDLx.Signal_Backward
   endif
endif

if ( (S1.Level == S2.Level) and (S1.Level == S3.Level) ) then    //error
   if (!PDLx.Signal_Output2.Level) then    //rising edge
      generate pulse on PDLx.Signal_Output2.Transition
   endif
   PDLx.Signal_Output2.Level = 1
   PDLx.Error = 1
else    //no error
   if (PDLx.Signal_Output2.Level) then    //falling edge
      generate pulse on PDLx.Signal_Output2.Transition
   endif
   PDLx.Signal_Output2.Level = 0
endif
```

Variables

Input, Local, Output variables of the cell (I, L, O)

Name x = [0,1] for PDL k = [0 to 5] for FPC	Short Name (*)PDL	Used (ILO)	Comment
FPCk.Signal_Output.Transition FPCk.Signal_Output.Level	SOTk SOLk	I	Transition/Level of signals coming from FPC
S1.Transition, S1.Level S2.Transition, S2.Level S3.Transition, S3.Level	S1T, S1L S2T, S2L S3T, S3L	L	Transition/Level of Local FPC signals
PDLx.Signal_Output1.Transition PDLx.Signal_Output1.Level	SIT0, SIL0 SIT2, SIL2	O	Transition/Level of Output 1 signal going to DCM0/DCM2
PDLx.Signal_Output2.Transition PDLx.Signal_Output2.Level	SIT1, SIL1 SIT3, SIL3	O	Transition/Level of Output 2 signal going to DCM1/DCM3
PDLx.Signal_Forward	*F0 *F1	O	Forward signals to be counted by LTC
PDLx.Signal_Backward	*B0 *B1	O	Backward signals to be counted by LTC

Global variables

Name x = [0,1] for PDL	Short Name (*)PDL	Size (bits)	Function
PDLx.Mux	*MUXx	1	Selects PDL speed signal (instead of FPC feed-through signal) for output 1
PDLx.Three_Sensors_Enable	*TSEx	1	Selects 3-sensor option and PDL error signal (instead of FPC feed-through signal) for output 2
PDLx.Error	*ERRx	1	Allows the software to read PDL error

22.3.6.3 DCM-Algorithm

DCMk_Control_Logic() “to be performed every GPTA®v5 clock”

```
Compare()
Add_Clock()
Check_Input()
```

Compare()

```
if (DCMk.Timer == DCMk.Capcom_Value) then
  trig(DCMk.Service_Request_Compare)
endif
```

Add_Clock()

```
if (DCMk.Clock_Request) then
  Generate DCMk.Signal_Output
  DCMk.Clock_Request = 0
endif
```

Check_Input()

```
if (DCMk.Signal_Input.Transition) then
  if (DCMk.Signal_Input.Level) then    //rising edge
    trig(DCMk.Service_Request_Rising)
    if (DCMk.Capture_On_Rising_Edge) then
      DCMk.Capture_Value = DCMk.Timer
    else
      if (DCMk.Capcom_Opposite) then
        DCMk.Capcom_Value = DCMk.Timer
      endif
    endif
    if (DCMk.Clear_On_Rising_Edge) then
      DCMk.Timer = 0
    else DCMk.Timer ++
    endif
    if (DCMk.Clock_On_Rising_Edge) then
      Generate pulse on DCMk.Signal_Output
    endif
  else    //falling edge
    trig(DCMk.Service_Request_Falling)
    if (!DCMk.Capture_On_Rising_Edge) then
      DCMk.Capture_Value = DCMk.Timer
    else
      if (DCMk.Capcom_Opposite) then
        DCMk.Capcom_Value = DCMk.Timer
      endif
    endif
    if (DCMk.Clear_On_Falling_Edge) then
      DCMk.Timer = 0
    else DCMk.Timer ++
    endif
    if (DCMk.Clock_On_Falling_Edge) then
      Generate pulse on DCMk.Signal_Output
    endif
  endif
else DCMk.Timer ++
endif
```

Variables

Input, Local, Output variables of the cell (I, L, O)

Name k = [0 to 3] for DCM	Short Name	Used (ILO)	Comment
DCMk.Signal_Input.Transition DCMk.Signal_Input.Level	*SITk *SILk	I	Input of the cell
DCMk.Signal_Output	*SOk	O	Output of the cell
DCMk.Service_Request_Rising	*RTQk	O	Service request on rising edge
DCMk.Service_Request_Falling	*FTQk	O	Service request on falling edge
DCMk.Service_Request_Compare	*CTQk	O	Service request on compare event

Global variables

Name k = [0 to 3] for DCM	Short Name (*)DCM	Size (bits)	Function
DCMk.Capture_On_Rising_Edge	*RCAk	1	Capture into Capture_Value on rising edge
DCMk.Capcom_Opposite	*OCAk	1	Capture into Capcom_Value on opposite edge defined by RCAk
DCMk.Clear_On_Rising_Edge	*RZEk	1	Clear Timer on rising edge
DCMk.Clear_On_Falling_Edge	*FZEk	1	Clear Timer on falling edge
DCMk.Clock_On_Rising_Edge	*RCKk	1	Generate a single clock pulse on rising edge
DCMk.Clock_On_Falling_Edge	*FCKk	1	Generate a single clock pulse on falling edge
DCMk.Clock_Request	*QCKk	1	Generate a single clock pulse immediately
DCMk.Request_Enable_Rising	*RREk	1	Enable request on rising edge
DCMk.Request_Enable_Falling	*FREk	1	Enable request on falling edge
DCMk.Request_Enable_Compare	*CREk	1	Request enable on compare
DCMk.Timer	*TIMk	24	Timer value
DCMk.Capture_Value	*CAVk	24	Capture value
DCMk.Capcom_Value	*COVk	24	Capture/compare value

22.3.6.4 PLL-Algorithm

PLL_Control_Logic() “to be performed every GPTA®v5 clock”

```
if ( (Pll.Automatic_End) and (Pll.Event) ) then    //allow compensation
  Pll.Perform_End = 1
endif

if ( (Pll.Counter_Mtick == 0) and ((Pll.Perform_End) or (!Pll.Automatic_End)) )
then    //compensation finished or no automatic compensation
  Pll.Counter_Mtick = Pll.Number_Mtick
  Pll.Perfom_End = 0
endif

if ( (Pll.Counter_Mtick != 0) and ((Pll.Perform_End) or (Bit 24 of Pll.Delta)) )
then    //output pulse is necessary
  generate pulse on Pll.Signal_Output
  Pll.Counter_Mtick --
  if (Pll.Counter_Mtick == 0) then
    trig(Pll.Service_Request_Trigger)
  endif
endif

if (Bit 24 of Pll.Delta) then    //delta is < 0
  Pll.Delta = Pll.Delta + Pll.Reload_Value
  generate pulse on Pll.Signal_Uncomp
else    //delta is >= 0
  Pll.Delta = Pll.Delta + (0xFFFF0000 or (Pll.Step))
endif
```

Variables

Input, Local, Output variables of the cell (I, L, O)

Name k = [0 to 3] for DCM	Short Name (*)PLL	Used (ILO)	Comment
DCMk.Signal_Output	SOk	I	Input of the cell from DCM
Pll.Event	*EVE	L	Input selected by the multiplexer
Pll.Signal_Output	*SO	O	Output of the cell
Pll.Signal_Uncomp	*SU	O	Uncompensated output of the cell
Pll.Service_Request_Trigger	*SQT	O	Service request when Counter reaches zero

Global variables

Name	Short Name (*)PLL	Size (bits)	Function
Pll.Mux	*MUX	2	Selects the signal input for PLL
Pll.Automatic_End	*AEN	1	Performs the acceleration/ deceleration correction
Pll.Perform_End	*PEN	1	Makes it possible to decrement the Counter at full speed
Pll.Request_Enable	*REN	1	Allows a request when microtick counter reaches zero
Pll.Number_Mtick	*MTI	16	Number of microticks per input signal period
Pll.Counter_Mtick	*CNT	16	Microtick counter
Pll.Step	*STP	16	Step value, to be added to positive/zero delta register
Pll.Reload_Value	*REV	24	Reload value, to be added to negative delta register
Pll.Delta	*DTR	25	Delta register

22.3.6.5 GT-Algorithm

GTm_Control_Logic() “to be performed every GPTA®v5 clock”

```
if (GTm.Run) then
  if (Event on GTm.Clock_In[p] selected by GTm.Clock_Mux) then
    GTm.Timer ++
    if (Overflow of GTm.Timer) then
      GTm.Timer = GTm.Reload_Value
      trig(GTm.Service_Request_Trigger)
    endif
  endif
endif
```

Variables

Input, Local, Output variables of the cell (I, L, O)

Name m = [0, 1] for GT p = [0 to 7] for Clock Bus	Short Name (*)GT	Used (ILO)	Comment
GTm.Clock_In[p]	*CINmp	I	Input coming from clock bus
GTm.Timer_Greater_Equal_Comp	TGEm	O	Timer is greater or equal
GTm.Timer_Event	TEVm	O	Signal for timer change
GTm.Service_Request_Trigger	*SQTm	O	Service request line

Global variables

Name m = [0, 1] for GT	Short Name (*)GT	Size (bits)	Function
GTm.Run	*RUNm	1	Enables timer
GTm.Scale_Compare	*SCOm	4	Selects compare flag
GTm.Clock_Mux	*MUXm	3	Selects clock from clock bus
GTm.Request_Enable	*RENm	1	Allows a request when timer overflows
GTm.Timer	*TIMm	24	Timer value
GTm.Reload_Value	*REVm	24	Reload value when timer overflows

22.3.6.6 GTC-Algorithm

GTCk_Control_Logic() “to be performed every GPTA®v5 clock”

```
if (GTCk.Cell_Enable) then
  switch (GTCk.Mode)
    case CAPTURE_T0:
      Capture(0)
      break
    case CAPTURE_T1:
      Capture(1)
      break
    case COMPARE_T0:
      Compare(0)
      break
    case COMPARE_T1:
      Compare(1)
  endswitch

  if ( (GTCk.One_Shot_Mode) and (GTCk.Event) ) then
    GTCk.Cell_Enable = 0
  endif
endif

Manage_Mux()
```

Capture(m)

```
if (GTCk.Signal_Input) then
  trig(GTCk.Service_Request_Trigger)
  GTCk.X = GTm.Timer
  GTCk.Event = 1
else
  GTCk.Event = 0
endif
Ck.Event = 0
```

Compare(m)

```
if ( ((GTCk.X == GTm.Timer) and ((GTCk.X_Write_Access) or (GTm.Timer_Event))) or
   ((GTCk.Greater_Equal_Select) and (GTCk.X_Write_Access)
   and (GTm.Timer_Greater_Equal_Comp)) ) then
   if (GTCk.Capture_After_Compare) then
      if (GTCk.Capture_Alternate_Timer) then
         GTCk.X = GT(!m).Timer
      else
         GTCk.X = GTm.Timer
      endif
   endif
   trig(GTCk.Service_Request_Trigger)
   GTCk.Event = 1
else
   GTCk.Event = 0
endif
```

Set_Data_Out(mode)

```
switch (mode)
   case 00B:    //no change
      break
   case 01B:    //toggle
      GTCk.Data_Out = !GTCk.Data_Out
      break
   case 10B:    //clear
      GTCk.Data_Out = 0
      break
   case 11B:    //set
      GTCk.Data_Out = 1
      break
endswitch
GTCk.Output_State = GTCk.Data_Out
```

Manage_Mux()

```
if ((GTCk.Event or GTCk.OIA) and GTCk.OCM != x00) then    //local event
  Set_Data_Out(GTCk.Output_Control_Mode.[1:0])
  if (!GTCk.Bypass) then    //no bypass
    GTCk.Output_Mode_Out = GTCk.Output_Control_Mode.[1:0]
  else
    if (GTCk.Output_Control_Mode.2) then    //bypass, input link enabled
      GTCk.Output_Mode_Out = GTCk.Output_Mode_In
    else    //bypass, input link disabled
      GTCk.Output_Mode_Out = 00B
    endif
  endif
else    //no local event
  if (GTCk.Output_Control_Mode.2) then    //input link enabled
    Set_Data_Out(GTCk.Output_Mode_In)
    GTCk.Output_Mode_Out = GTCk.Output_Mode_In
  else    //input link disabled
    Set_Data_Out(00B)
    GTCk.Output_Mode_Out = 00B
  endif
endif
if ( (GTCk.Enable_Of_Action) and
    ((GTCk.Output_Mode_In.1) or (GTCk.Output_Mode_In.0)) ) then
  GTCk.Cell_Enable = 1
  GTCk.Enable_Of_Action = 0
endif
```

Variables

Input, Local, Output variables of the cell (I, L, O)

Name k = [0 to 31] for GTC m = [0, 1] for GT	Short Name (*)GTC	Used (ILO)	Comment
GTm.Timer_Greater_Equal_Comp	TGEm	I	Timer is greater or equal
GTm.Timer_Event	TEVm	I	Signal for timer change
GTm.Timer	*TIMm	I	Timer value
GTCk.Data_In	*DINk	I	Data input from input multiplexer
GTCk.Output_Mode_In	*M1Ik *M0Ik	I	Link signals from preceding cell
GTCk.X_Write_Access	*XWA	L	Indicates that GTCk.X was modified
GTCk.Event	*EVE	L	Local event
GTCk.Signal_Input	*INS	L	Qualified input signal
GTCk.Service_Request_Trigger	*SQSk	O	Service request line
GTCk.Data_Out	*DOUk	O	Data output for output multiplexer
GTCk.Output_Mode_Out	*M1Ok *M0Ok	O	Link signals to following cell

Global variables

Name k = [0 to 31] for GTC	Short Name (*)GTC	Size (bits)	Comment
GTCk.Mode	*MODk	2	Operation mode: CAPTURE_T0, CAPTURE_T1, COMPARE_T0, COMPARE_T1
GTCk.One_Shot_Mode	*OSMk	1	One shot mode
GTCk.Request_Enable	*RENk	1	Allows a request on event
GTCk.Input_Rising_Edge_Select (Capture Mode) GTCk.Greater_Equal_Select (Compare Mode)	*REDk *GESk	1 1	Selects rising edge of input pin Selects >= Compare Mode
GTCk.Input_Falling_Edge_Select (Capture Mode) GTCk.Capture_After_Compare (Compare Mode)	*FEDk *CACk	1 1	Selects falling edge of input pin Selects capture after compare
GTCk.Capture_Alternate_Timer (Compare Mode)	*CATk	1	Capture alternate global timer after compare
GTCk.Bypass	*BYPk	1	Local events bypassed for output link
GTCk.Enable_Of_Action	*EOAk	1	Enables cell on action communicated via link
GTCk.Cell_Enable	*CENk	1	Cell enable state
GTCk.Output_Control_Mode	*OCMk	3	Output control mode
GTCk.Output_Immediate_Action	*OIAk	1	Forces immediate action
GTCk.Output_State	*OUTk	1	Read value of Data_Out
GTCk.X	*Xk	24	Capture/Compare value

22.3.6.7 LTC-Algorithm for Cells 0 to 62

LTCk_Control_Logic() “to be performed every GPTA®v5 clock”

```
if (LTCk.Cell_Enable) then
  switch (LTCk.Mode)
    case TIMER_FREE_RUN:
      LTCk.Reset_Timer_Bit = 0
      Timer()
      break
    case TIMER_RESET:
      if (LTCk.Event_In) then
        LTCk.Reset_Timer_Bit = 1
      endif
      Timer()
      break;
    case CAPTURE:
      Capture()
      break
    case COMPARE:
      Compare()
      break
  endswitch
  if ((LTCk.One_Shot_Mode) and (LTCk.Event)) then
    LTCk.Cell_Enable = 0
  endif
endif

Manage_Mux()
```

Timer()

```
if ( (LTCk.X == 0xFFFF) and (LTCk.X_Write_Access) ) then
   //above condition is also true for timer overflow or software reset
   trig(LTCk.Service_Request_Trigger)
   LTCk.Event = 1
else
   LTCk.Event = 0
endif
if (LTCk.Signal_Input) then
   if (LTCk.Reset_Timer_Bit) then    //timer must be reset
      LTCk.Reset_Timer_Bit = 0
      LTCk.X = 0xFFFF
      if (LTCk.Coherent_Update_Enable) then
         LTCk.Select_Line_Value = !LTCk.Select_Line_Value
         LTCk.Coherent_Update_Enable = 0
      endif
   else    //timer runs normally
      LTCk.X ++
   endif
endif
LTCk.Event_Out = LTCk.Event
```

Capture()

```
if (LTCk.Signal_Input) then
  trig(LTCk.Service_Request_Trigger)
  LTCk.X = LTCk.Y_In
  LTCk.Event = 1
else
  LTCk.Event = 0
endif
LTCk.Event_Out = LTCk.Event
```

Compare()

```
if ( ((LTCk.Select_In) and (LTCk.Select_On_High_Level)) or
  ((!LTCk.Select_In) and (LTCk.Select_On_Low_Level)) ) then    //cell is active
  if ( (LTCk.X == LTCk.Y_In) and
    ((LTCk.X_Write_Access) or (LTCk.Timer_Event_In)) ) then    //event
    trig(LTCk.Service_Request_Trigger)
    LTCk.Event = 1
  else
    LTCk.Event = 0
  endif
  LTCk.Event_Out = LTCk.Event
else    //cell is inactive
  LTCk.Event_Out = LTCk.Event_In
endif
```

Manage_Mux()

```
if ( (LTCk.Mode == TIMER_FREE_RUN) or (LTCk.Mode == TIMER_RESET) ) then
  LTCk.Y_Out = LTCk.X
  if (the timer has been modified) then    //increment, reset, software overwrite
    LTCk.Timer_Event_Out = 1
  else
    LTCk.Timer_Event_Out = 0
  endif
  LTCk.Select_Out = LTCk.Select_Line_Value
else    //capture mode or compare mode
  LTCk.Y_Out = LTCk.Y_In
  LTCk.Timer_Event_Out = LTCk.Timer_Event_In
  LTCk.Select_Line_Value = LTCk.Select_In
  LTCk.Select_Out = LTCk.Select_In
endif
if (LTCk.Event) then    //local event
  Set_Data_Out(LTCk.Output_Control_Mode.[1:0])
  if (!LTCk.Bypass) then    //no bypass
    LTCk.Output_Mode_Out = LTCk.Output_Control_Mode.[1:0]
  endif
else    //no local event
  if (LTCk.Output_Control_Mode.2)    //input link enabled
    Set_Data_Out(LTCk.Output_Mode_In)
    if (!LTCk.Bypass) then    //no bypass
      LTCk.Output_Mode_Out = LTCk.Output_Mode_In
    endif
  else    //input link disabled
    Set_Data_Out(00B)
    if (!LTCk.Bypass) then    //no bypass
      LTCk.Output_Mode_Out = 00B
    endif
  endif
endif
```

Manage_Mux() - continued

```
if (LTCk.GlobalBypass) then    //global bypass
  LTCk.Output_Mode_Alternate_Out = LTCk.Output_Mode_Alternate_In
  if (LTCk.Bypass) then    // bypass
    LTCk.Output_Mode_Out = LTCk.Output_Mode_Alternate_In
  endif
else
  if (LTCk.Output_Control_Mode.2) then    //bypass, input link enabled
    LTCk.Output_Mode_Alternate_Out = LTCk.Output_Mode_In
    if (LTCk.Bypass) then    // bypass
      LTCk.Output_Mode_Out = LTCk.Output_Mode_In
    endif
  else    //bypass, input link disabled
    LTCk.Output_Mode_Alternate_Out = 00B
    if (LTCk.Bypass) then    // bypass
      LTCk.Output_Mode_Out = 00B
    endif
  endif
endif
if ( (LTCk.Enable_Of_Action) and
   ((LTCk.Output_Mode_In.1) or (LTCk.Output_Mode_In.0)) ) then    //enable condition
  LTCk.Cell_Enable = 1
  LTCk.Enable_Of_Action = 0
endif
```

Set_Data_Out(mode)

```
switch (mode)
  case 00B:    //no change
    break
  case 01B:    //toggle
    LTCk.Data_Out = !LTCk.Data_Out
    break
  case 10B:    //clear
    LTCk.Data_Out = 0
    break
  case 11B:    //set
    LTCk.Data_Out = 1
    break
endswitch
LTCk.Output_State = LTCk.Data_Out
```

Variables

Input, Local, Output variables of the cell (I, L, O)

Name	Short Name (*)LTC	Used (ILO)	Comment
LTCk.Data_In	*DINkp	I	Data input from input multiplexer
LTCk.Y_In	*YIk	I	Timer coming from preceding cell
LTCk.Output_Mode_In	*M1Ik *M0Ik	I	Link signals coming from preceding cell
LTCk.Output_Mode_Alternate_In	*M3Ik *M2Ik	I	Alternative Link signals coming from preceding cell
LTCk.Timer_Event_In	*TIk	I	Signal for timer change from preceding cell
LTCk.Event_In	*EIk	I	Signal for event from following cell
LTCk.Select_In	*SI	I	Select signal from preceding cell
LTCk.X_Write_Access	*XWA	L	Indicates that LTCk.X was modified
LTCk.Select_Line_Value	*SLV	L	Internal value for select line reset value: 0
LTCk.Signal_Input	*INS	L	Qualified input signal for Timer Mode and Capture Mode
LTCk.Reset_Timer_Bit	*RTM	L	Flip-flop to reset timer on next clock
LTCk.Event	*EVE	L	Local event
LTCk.Data_Out	*DOUk	O	Data output for output multiplexer
LTCk.Service_Request_Trigger	*SQTk	O	Service request line
LTCk.Y_Out	*YOk	O	Timer going to following cell
LTCk.Output_Mode_Out	*M1Ok *M0Ok	O	Link signals to following cell
LTCk.Output_Mode_Alternate_Out	*M3Ok *M2Ok	O	Link signals to following cell
LTCk.Timer_Event_Out	*TOk	O	Event output to following cell
LTCk.Select_Out	*SO	O	Select output to following cell
LTCk.Event_Out	*EOk	O	Event output to preceding cell

Global variables

Name k = [0 to 62] for LTC	Short Name (*)LTC	Size (bits)	Comment
LTCk.Mode	*MODk	2	Operation mode: TIMER, TIMER_RESET, CAPTURE, COMPARE
LTCk.One_Shot_Mode	*OSMk	1	One shot mode
LTCk.Request_Enable	*RENk	1	Allows a request on event
LTCk.Input_Rising_Edge_Select (Timer Mode, Capture Mode) LTCk.Select_On_Low_Level (Compare Mode)	*REDk *SOLk	1 1	Selects rising edge of input pin Enables compare on low level of select line
LTCk.Input_Falling_Edge_Select (Timer Mode, Capture Mode) LTCk.Select_On_High_Level (Compare Mode)	*FEDk *SOHk	1 1	Selects falling edge of input pin Enables compare on high level of select line
LTCk.Bypass (Capture Mode, Compare Mode)	*BYPk	1	Local events bypassed for output link
LTCk.GlobalBypass	*GBYPk	1	Alternative output links forwarded to alternative output link
LTCk.Enable_Of_Action (Capture Mode, Compare Mode)	*EOAk	1	Enables cell on action communicated via link
LTCk.Input_Line_Mode	*ILMk	1	Selects edge input line mode
LTCk.Coherent_Update_Enable (Timer Mode) LTCk.Select_Line_Level (Capture Mode, Compare Mode)	*CUDk *SLLk	1 1	Selects coherent update Select line level
LTCk.Cell_Enable	*CENk	1	Cell enable state
LTCk.Output_Control_Mode	*OCMk	3	Output control mode
LTCk.Output_Immediate_Action	*OIAk	1	Forces immediate action
LTCk.Output_State	*OUTk	1	Read value of Data_Out
LTCk.X	*Xk	16	Timer/Capture/Compare value

22.3.6.8 LTC Algorithm for Cell 63

LTC63_Control_Logic() “to be performed every GPTA®v5 clock”

```
Copy()
Compare()
```

Copy()

```
if (LTC63.Cell_Enable) then
  if (LTC63.Signal_Input) then
    LTC63.X = LTC63.X_Shadow
    trig(LTC63.Service_Request_Trigger)
    if (LTC63.One_Shot_Mode) then
      LTC63.Cell_Enable = 0
    endif
  endif
endif
```

Compare()

```
if ( (LTC63.X_Write_Access) or (LTC63.Timer_Event_In) ) then
  if (LTC63.Bit_Rev_Mode) then
    LTC63.Y_Comp = LTC63.Y_Rev
  else
    LTC63.Y_Comp = LTC63.Y_In
  endif
  if ( (LTC63.X > LTC63.Y_Comp) or (LTC63.X == FFFFH) ) then    //output must be 1
    LTC63.Data_Out = 1
    LTC63.Event_Out = 0
  else    //output must be 0
    if (LTC63.Data_Out == 1) then    //falling edge on output
      trig(LTC63.Service_Request_Trigger)
      LTC63.Event_Out = 1
    else
      LTC63.Event_Out = 0
    endif
    LTC63.Data_Out = 0
  endif
  LTC63.Output_State = LTC63.Data_Out
endif
```

Variables

Input, Local, Output variables of the cell (I, L, O)

Name	Short Name (*)LTC	Used (ILO)	Comment
LTC63.Data_In	*DIN63	I	Data input from input multiplexer
LTC63.Y_In	*YI63	I	Timer coming from preceding cell
LTC63.Timer_Event_In	*TI63	I	Signal for timer change from preceding cell
LTC63.Y_Rev	*YR	L	Timer coming from preceding cell, bit reversed
LTC63.Y_Comp	*YC	L	Timer actually used for compare
LTC63.X_Write_Access	*XWA	L	Indicates that LTC63.X was modified
LTC63.Signal_Input	*INS	L	Qualified input signal
LTC63.Data_Out	*DOU63	O	Data output for output multiplexer
LTC63.Service_Request_Trigger	*SQT63	O	Service request line
LTC63.Event_Out	*EO63	O	Event output to preceding cell

Global variables

Name	Short Name (*)LTC	Size (bits)	Comment
LTC63.Bit_Rev_Mode	*BRM63	1	Bit reverse mode
LTC63.One_Shot_Mode	*OSM63	1	One shot mode for copy
LTC63.Request_Enable	*REN63	2	Allows a request on compare or copy
LTC63.Input_Rising_Edge_Select	*RED63	1	Selects rising edge of input pin
LTC63.Input_Falling_Edge_Select	*FED63	1	Selects falling edge of input pin
LTC63.Input_Line_Mode	*ILM63	1	Selects edge input line mode
LTC63.Cell_Enable	*CEN63	1	Cell enable state for copy
LTC63.Output_State	*OUT63	1	Read value of Data_Out
LTC63.X	*X63	16	Compare value
LTC63.X_Shadow	*XS63	16	Shadow compare value

22.3.7 Programming of a GPTA®v5 Unit

A hierarchical top-down design approach may be used to implement a complex signal processing circuitry as follows:

- Partitioning the complex signal processing circuitry into simple function cells.
- Implementing each simple function cell by configuring the LTC and/or GTC cells which can be tied together for realizing a common signal operation.
- Implementing necessary signal pre-processing tasks by configuring the FPC, PDL, DCM and PLL cells accordingly.
- Defining and configuring all input/output port pins required as clock source, trigger input or signal output.

Table 22-17 summarizes all of the software tasks to be implemented for getting a GPTA®v5 unit into operation.

Table 22-17 Software Tasks Controlling a GPTA®v5 Unit

GPTA®v5 Shell Initialization	
GPTA®v5 Module Clock Enable	
Fractional Divider Setting	
Unit Enable	
Configuration of Interrupt Handling	
GPTA®v5 Kernel Initialization	
FPC:	**PDL:**
Selection of Operating Mode (Prescaler, Filter or Feed-Through)	Selection of Operating Mode (Phase Discriminator or Feed-Through)
Input Channel Selection	2- or 3-Sensor Mode Selection
Clock Selection	**PLL:**
Configuration of Prescaler Factor or Debounce Mode	Selection of Input Channel
DCM:	Estimation of Input Signal Period Width
Selection of Reset Event for Timer	Configuration of Output Signal Frequency
Selection of Trigger Source for Capture Event	Handling of Input Signal Period Length Variation
Selection of Trigger Source for Capture Compare Register Update	Interrupt Request Enable on End of Output Pulse Generation
Interrupt Request Enable on Input Edge or Compare Event	

Table 22-17 Software Tasks Controlling a GPTA®v5 Unit (cont'd)

Clock Bus Setup	
Selection and Configuration of 8 Clock Sources for GT, GTC and LTC Cells	
GT:	**GTC:**
Selection of Timer Clock Source	Selection of Operating Mode (Capture or Compare) and Time Base (GT0 or GT1)
Configuration of Timer Width (Reload Value, TGE Flag)	Configuration of Trigger Events for Capture Mode or Selection of a Relational Operator for Compare Mode
Interrupt Request Enable on Timer Overflow	Interrupt Request Enable on Capture or Compare Event
Start Global Timer(s)	Configuration of Data Output triggered by a GTC Event
LTC:	**IOLS:**
Selection of Operating Mode (Timer, Capture or Compare)	Configuration of the Multiplexer Array to link GTC and LTC data outputs/inputs to external Port Pins or other cells by writing the Multiplexer Register Array FIFO Configuration of the On-chip Trigger and Gating Signal Multiplexer Array to link GTC and LTC data outputs to on-chip modules by writing the Multiplexer Register Array FIFO
Selection of Trigger Source for Timer, Capture or Compare Mode	Configuration of Port Output Source
Configuration of Trigger Event for Timer, Capture or Compare Mode	
Interrupt Request Enable on Timer, Capture or Compare Event	
Configuration of Data Output triggered by an LTC Event	
Port Initialization	
Definition of Electrical Port Characteristic	
Configuration of Port Pin Direction (Input or Output)	

22.4 GPTA0/1 Kernel Registers

This section describes the kernel registers of the GPTA0 and GPTA1 unit.

GPTA0/1 Kernel Register Overview

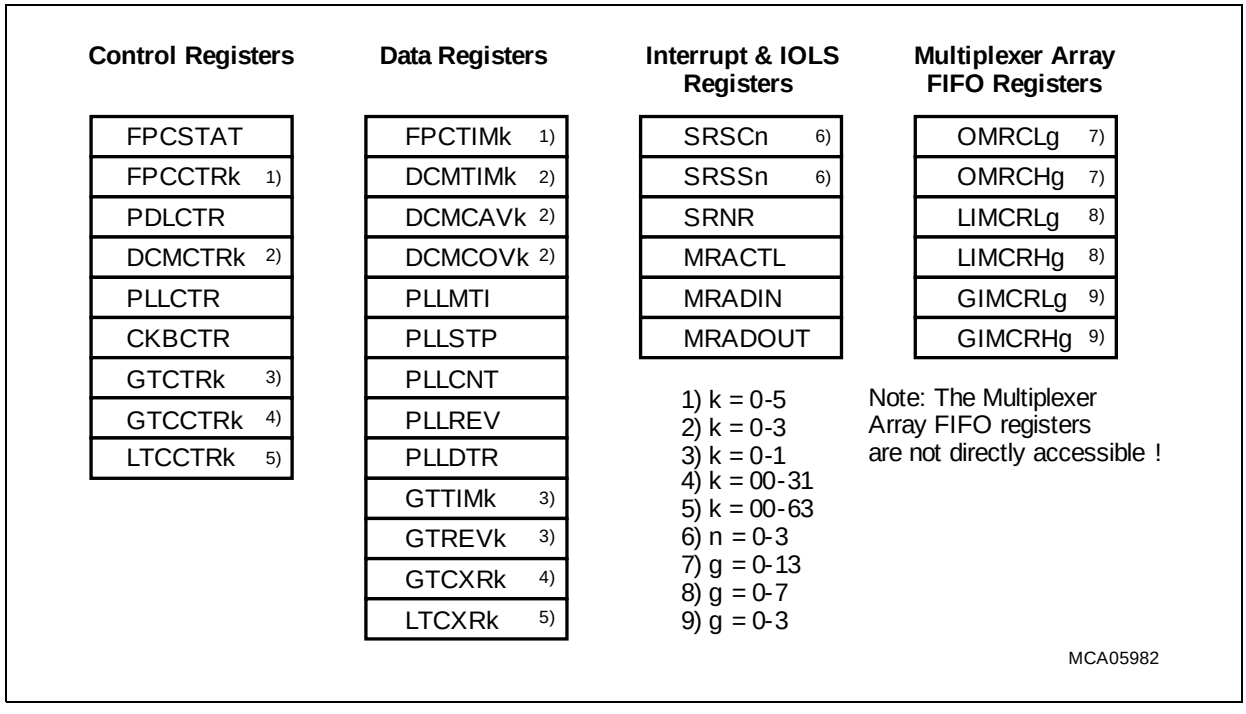


Figure 22-80 GPTA0 and GPTA1 Kernel Registers

In the TC1797, the registers of the GPTA®v5 units are located in the following address ranges.

Table 22-18 Registers Address Space

Module	Base Address	End Address	Note
GPTA0	F000 1800$_H$	F000 1FFF$_H$	-
GPTA1	F000 2000$_H$	F000 27FF$_H$	-
LTCA2	F000 2800$_H$	F000 2FFF$_H$	-

Table 22-19 Registers Overview - GPTA0 and GPTA1 Kernel Registers

Register Short Name	Register Long Name	Offset Addr.[1]	Access Mode		Reset Class	Description see
			Read	Write		
GPTA0_CLC[2]	GPTA Clock Control Register	0000_H	U, SV	SV, E	3	Page 22-300
GPTA0_DBGCTR[2]	GPTA Debug Clock Control Register	0004_H	U, SV	U, SV	3	Page 22-312
ID	GPTA Identification Register	0008_H	U, SV	nBE	3	Page 22-166
GPTA0_FDR[2]	GPTA Fractional Divider Register	$000C_H$	U, SV	SV, E	3	Page 22-309
SRSC0	Service Request State Clear Register 0	010_H	U, SV	U, SV	3	Page 22-223
SRSS0	Service Request State Set Register 0	014_H	U, SV	U, SV	3	Page 22-225
SRSC1	Service Request State Clear Register 1	018_H	U, SV	U, SV	3	Page 22-226
SRSS1	Service Request State Set Register 1	$01C_H$	U, SV	U, SV	3	Page 22-227
SRSC2	Service Request State Clear Register 2	020_H	U, SV	U, SV	3	Page 22-228
SRSS2	Service Request State Set Register 2	024_H	U, SV	U, SV	3	Page 22-229
SRSC3	Service Request State Clear Register 3	028_H	U, SV	U, SV	3	Page 22-230
SRSS3	Service Request State Set Register 3	$02C_H$	U, SV	U, SV	3	Page 22-231
SRNR	Service Request Node Redirection Register	030_H	U, SV	U, SV	3	Page 22-232
MRACTL	Multiplexer Register Array Control Register	0038_H	U, SV	U, SV	3	Page 22-207
MRADIN	Multiplexer Register Array Data In Register	$003C_H$	U, SV, 32	U, SV, 32	3	Page 22-208
MRADOUT	Multiplexer Register Array Data Out Register	0040_H	U, SV, 32	U, SV, 32	3	Page 22-209

Table 22-19 Registers Overview - GPTA0 and GPTA1 Kernel Registers (cont'd)

Register Short Name	Register Long Name	Offset Addr.[1)]	Access Mode		Reset Class	Description see
			Read	Write		
FPCSTAT	Filter and Prescaler Cell Status Register	0044_H	U, SV	U, SV	3	Page 22-167
FPCCTRk	Filter and Prescaler Cell Control Register k (k = 0-5)	$0048_H + k \times 8$	U, SV	U, SV	3	Page 22-168
FPCTIMk	Filter and Prescaler Cell Timer Register k (k = 0-5)	$004C_H + k \times 8$	U, SV	U, SV	3	Page 22-170
PDLCTR	Phase Discrimination Logic Control Register	0078_H	U, SV	U, SV	3	Page 22-171
DCMCTRk	Duty Cycle Measurement Control Register k (k = 0-3)	$0080_H + k \times 16$	U, SV	U, SV	3	Page 22-173
DCMTIMk	Duty Cycle Measurement Timer Register k (k = 0-3)	$0084_H + k \times 16$	U, SV	U, SV	3	Page 22-175
DCMCAVk	Duty Cycle Measurement Capture Register k (k = 0-3)	$0088_H + k \times 16$	U, SV	U, SV	3	Page 22-175
DCMCOVk	Duty Cycle Measurement Capture/Compare Register k (k = 0-3)	$008C_H + k \times 16$	U, SV	U, SV	3	Page 22-176
PLLCTR	Phase Locked Loop Control Register	$00C0_H$	U, SV	U, SV	3	Page 22-177
PLLMTI	Phase Locked Loop Micro Tick Register	$00C4_H$	U, SV	U, SV	3	Page 22-178
PLLCNT	Phase Locked Loop Counter Register	$00C8_H$	U, SV	U, SV	3	Page 22-179
PLLSTP	Phase Locked Loop Step Register	$00CC_H$	U, SV	U, SV	3	Page 22-179
PLLREV	Phase Locked Loop Reload Register	$00D0_H$	U, SV	U, SV	3	Page 22-180

Table 22-19 Registers Overview - GPTA0 and GPTA1 Kernel Registers (cont'd)

Register Short Name	Register Long Name	Offset Addr.[1]	Access Mode		Reset Class	Description see
			Read	Write		
PLLDTR	Phase Locked Loop Delta Register	$00D4_H$	U, SV	U, SV	3	Page 22-181
CKBCTR	Clock Bus Control Register	$00D8_H$	U, SV	U, SV	3	Page 22-185
GTCTRk	Global Timer Control Register k (k = 0, 1)	$00E0_H + k \times 16$	U, SV	U, SV	3	Page 22-182
GTREVk	Global Timer Reload Value Register k (k = 0, 1)	$00E4_H + k \times 16$	U, SV	U, SV	3	Page 22-184
GTTIMk	Global Timer Register k (k = 0, 1)	$00E8_H + k \times 16$	U, SV	U, SV	3	Page 22-183
GTCCTRk	Global Timer Cell Control Register k (k = 00-31)	$0100_H + k \times 8$	U, SV	U, SV	3	Page 22-187 Page 22-189
GTCXRk	Global Timer Cell X Register k (k = 00-31)	$0104_H + k \times 8$	U, SV	U, SV	3	Page 22-191
LTCCTRk	Local Timer Cell Control Register k (k = 00-62)	$0200_H + k \times 8$	U, SV	U, SV	3	Page 22-192 Page 22-198 Page 22-201
LTCXRk	Local Timer Cell X Register k (k = 00-62)	$0204_H + k \times 8$	U, SV	U, SV	3	Page 22-205
LTCCTR63	Local Timer Cell Control Register 63	$03F8_H$	U, SV	U, SV	3	Page 22-204
LTCXR63	Local Timer Cell X Register 63	$03FC_H$	U, SV	U, SV	3	Page 22-206
GPTA0_EDCTR[2]	GPTA Clock Enable/Disable Control Register	0400_H	U, SV	U, SV	3	Page 22-310

Table 22-19 Registers Overview - GPTA0 and GPTA1 Kernel Registers (cont'd)

Register Short Name	Register Long Name	Offset Addr.[1]	Access Mode		Reset Class	Description see
			Read	Write		
OTMCRg	On-Chip Trigger and Gating Signal Multiplexer Control Register of Group g (g = 0-1)	not directly address -able; see Page 22 -121	n.a.	n.a.	3	Page 22-214
OMCRLg	Output Multiplexer Control Register for Lower Half of Group g (g = 0-13)		n.a.	n.a.	3	Page 22-210
OMCRHg	Output Multiplexer Control Register for Upper Half of Group g (g = 0-13)		n.a.	n.a.	3	Page 22-212
GIMCRLg	Input Multiplexer Control Register for Lower Half of GTC Group g (g = 0-3)		n.a.	n.a.	3	Page 22-215
GIMCRHg	Input Multiplexer Control Register for Lower Half of GTC Group g (g = 0-3)		n.a.	n.a.	3	Page 22-217
LIMCRLg	Input Multiplexer Control Register for Upper Half of LTC Group g (g = 0-7)	not directly address -able; see Page 22 -121	n.a.	n.a.	3	Page 22-219
LIMCRLg	Input Multiplexer Control Register for Upper Half of LTC Group g (g = 0-7)		n.a.	n.a.	3	Page 22-221

1) The absolute register address is calculated as follows:
Unit Base Address + Offset Address (shown in this column)

2) Only implemented in GPTA0 kernel.

Bit Protection

Bits with bit protection (this is valid, for example, for all bits in the Service Request State Registers) are not changed during a read-modify-write instruction, for example when

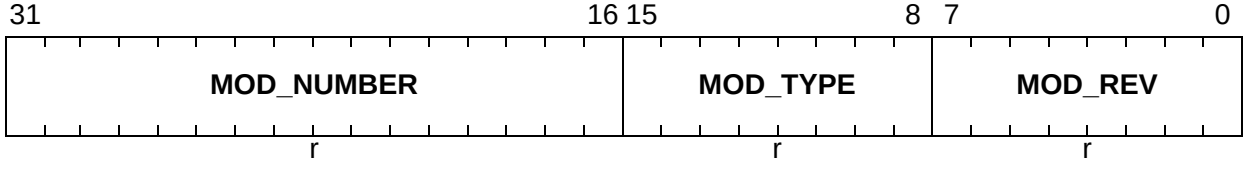
hardware sets a request state bit between the read and the write of the read-modify-write sequence. For bit protected bits it is guaranteed that a hardware setting operation always has priority. Thus, no hardware triggered events are lost.

Bits with bit protection are marked in the corresponding bit descriptions.

22.4.1 GPTA®v5 Identification Register

The GPTA®v5 Identification Register ID contains read-only information about the module version.

GPTA0_ID
GPTA0 Identification Register (08$_H$) Reset Value: 0029 C0XX$_H$
GPTA1_ID
GPTA1 Identification Register (08$_H$) Reset Value: 0029 C0XX$_H$
LTCA2_ID
LTCA2 Identification Register (08$_H$) Reset Value: 002A C0XX$_H$



Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision Number** MOD_REV defines the Module revision number. The value of a module revision starts with 01$_H$ (first revision). GPTAv5 will start with module revision 05$_H$.
MOD_TYPE	[15:8]	r	**Module Number Value** This bit field defines the module as a 32 bit module: C0$_H$
MOD_NUM	[31:16]	r	**Module Number Value** This bit field defines the identification number for the GPTA: 0029$_H$ and LTCA: 002A$_H$

22.4.2 FPC Registers

GPTA0_FPCSTAT
GPTA0 Filter and Prescaler Cell Status Register
(044_H) **Reset Value: 0000 0000_H**

GPTA1_FPCSTAT
GPTA1 Filter and Prescaler Cell Status Register
(044_H) **Reset Value: 0000 0000_H**

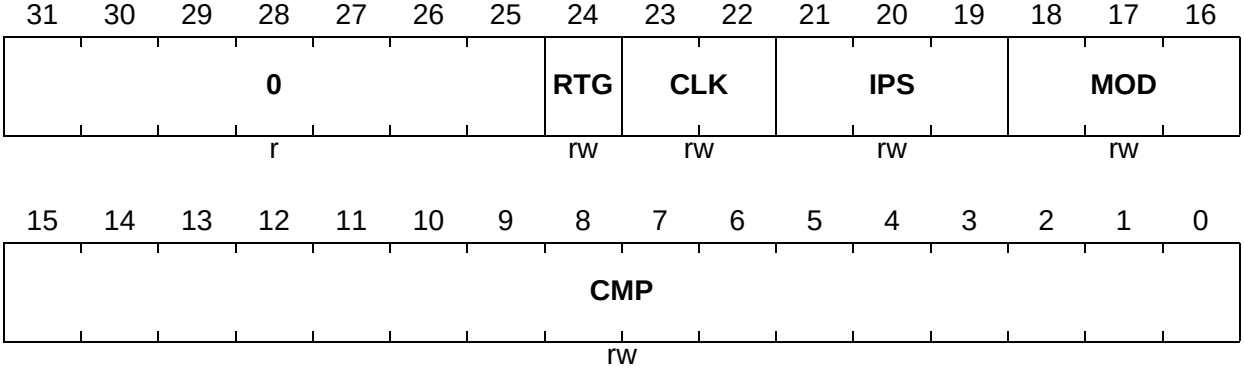
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		FEG5	FEG4	FEG3	FEG2	FEG1	FEG0	0		REG5	REG4	REG3	REG2	REG1	REG0
r		rwh	rwh	rwh	rwh	rwh	rwh	r		rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
REGk (k = 0-5)	k	rwh	**Rising Edge Glitch Flag for FPCk** 0_B No rising edge of glitch detected during filtering 1_B Rising edge of glitch detected during filtering Bits REGk are bit protected (see **Section 22.4.2**).
FEGk (k = 0-5)	k+8	rwh	**Falling Edge Glitch Flag for FPCk** 0_B No falling edge of glitch detected during filtering 1_B Falling edge of glitch detected during filtering Bits FEGk are bit protected (see **Section 22.4.2**).
0	[7:6], [31:14]	r	**Reserved** Read as 0; should be written with 0.

GPTA0_FPCCTRk (k = 0-5)
GPTA0 Filter and Prescaler Cell Control Register k
(048_H+k*8_H) **Reset Value: 0000 0000_H**

GPTA1_FPCCTRk (k = 0-5)
GPTA1 Filter and Prescaler Cell Control Register k
(048_H+k*8_H) **Reset Value: 0000 0000_H**

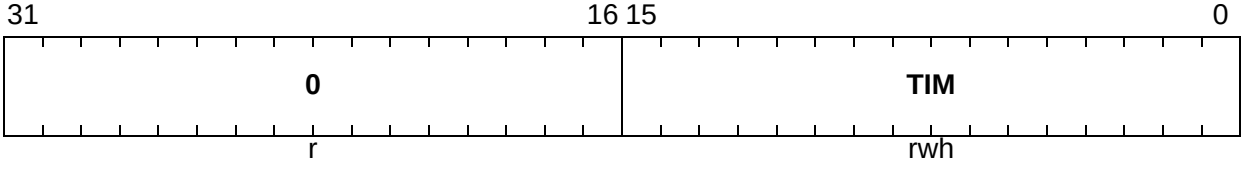


Field	Bits	Type	Description
CMP	[15:0]	rw	**Threshold Value of Filter and Prescaler Cell k** CMP is the 16-bit threshold value that is compared with the 16-bit timer value FPCTIMk.TIM.
MOD	[18:16]	rw	**Operation Mode Selection for FPCk** 000_B Delayed Debounce Filter Mode on both edges 001_B Immediate Debounce Filter Mode on both edges 010_B Rising edge: Immediate Debounce Filter Mode, falling edge: no filtering 011_B Rising edge: no filtering, falling edge: Immediate Debounce Filter Mode 100_B Rising edge: Delayed Debounce Filter Mode, falling edge: Immediate Debounce Filter Mode 101_B Rising edge: Immediate Debounce Filter Mode, falling edge: Delayed Debounce Filter Mode 110_B Prescaler Mode (triggered on rising edge) 111_B Prescaler Mode (triggered on falling edge)

Field	Bits	Type	Description
IPS	[21:19]	rw	**Input Line Selection for FPCk** IPS determines the signal input used for edge detection. 000_B Signal input SINk0 selected 001_B Signal input SINk1 selected 010_B Signal input SINk2 selected 011_B Signal input SINk3 selected 100_B Signal input SINk4 = GPTA®v5 module clock f_{GPTA} selected 101_B Signal input SINk5 = preceding FPC output SOLk-1 selected; SIN05 is connected to SOL5 $11X_B$ Reserved
CLK	[23:22]	rw	**Clock Selection for FPCk** CLK selects the clock signal used for edge detection. 00_B Clock input line 0 selected (GPTA®v5 module clock f_{GPTA}) 01_B Clock bus line 1 selected (local PLL clock) 10_B Clock bus line 2 selected (prescaled) GPTA®v5 module clock f_{GPTA} or PLL clock from other unit or DCM 3 clock 11_B Clock bus line 3 selected DCM 2 clock or PLL clock of other unit or uncompensated PLL clock or uncompensated PLL clock of other unit
RTG	24	rw	**Reset Timer for FPCk on Glitch** 0_B Timer for FPCk is decremented on glitch 1_B Timer for FPCk is cleared on glitch This bit is effective in Delayed Debounce Filter Mode only.
0	[31:25]	r	**Reserved** Read as 0; should be written with 0.

GPTA0_FPCTIMk (k = 0-5)
GPTA0 Filter and Prescaler Cell Timer Register k
(048_H+k*8_H+4_H) **Reset Value: 0000 0000_H**

GPTA1_FPCTIMk (k = 0-5)
GPTA1 Filter and Prescaler Cell Timer Register k
(048_H+k*8_H+4_H) **Reset Value: 0000 0000_H**

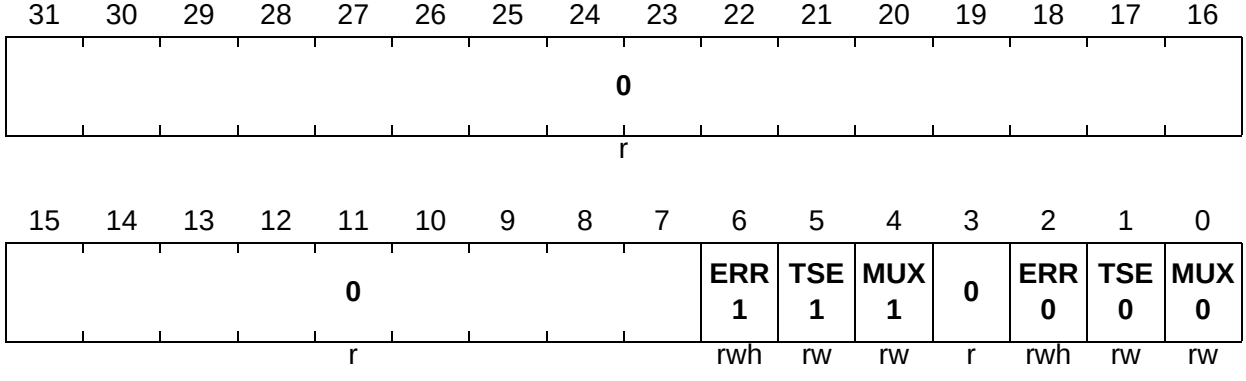


Field	Bits	Type	Description
TIM	[15:0]	rwh	**Timer Value of Filter and Prescaler Cell k**
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

22.4.3 Phase Discriminator Registers

GPTA0_PDLCTR
GPTA0 Phase Discrimination Logic Control Register
(078_H) **Reset Value: 0000 0000_H**

GPTA1_PDLCTR
GPTA1 Phase Discrimination Logic Control Register
(078_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
MUX0	0	rw	**Output Signal Source Selection for PDL0** 0_B DCM0 cell input is driven by fed-through FPC0 output lines 1_B DCM0 cell input is provided with PDL0 “Forward” and “Backward” pulses
TSE0	1	rw	**3-Sensor Mode Enable for PDL0** 0_B PDL0 operates in “2-Sensor Mode” and DCM1 cell input is driven by fed-through FPC2 output lines 1_B PDL0 operates in “3-Sensor Mode” and DCM1 cell input is provided with PDL0 error information
ERR0	2	rwh	**Error Flag for PDL0** 0_B No error has occurred 1_B Error detected in “3-Sensor Mode”: all PDL0 input signals are simultaneously provided with high or low level Bit ERR0 is bit protected (see **Page 22-167**).
MUX1	4	rw	**Output Signal Source Selection for PDL1** 0_B DCM2 cell input is driven by fed-through FPC3 output lines 1_B DCM2 cell input is provided with PDL1 “Forward” and “Backward” pulses

Field	Bits	Type	Description
TSE1	5	rw	**3-Sensor Mode Enable for PDL1** 0_B PDL1 operates in “2-Sensor Mode” and DCM3 cell input is driven by fed-through FPC5 output lines 1_B PDL1 operates in “3-Sensor Mode” and DCM3 cell input is provided with PDL1 error information
ERR1	6	rwh	**Error Flag for PDL1** 0_B No error has occurred 1_B Error detected in “3-Sensor Mode”: all PDL1 input signals are simultaneously provided with high or low level Bit ERR1 is bit protected (see **Page 22-167**).
0	3, [31:7]	r	**Reserved** Read as 0; should be written with 0.

22.4.4 Duty Cycle Measurement Registers

GPTA0_DCMCTRk (k = 0-3)
GPTA0 Duty Cycle Measurement Control Register k
(080_H+k*10_H) **Reset Value: 0000 0000_H**

GPTA1_DCMCTRk (k = 0-3)
GPTA1 Duty Cycle Measurement Control Register k
(080_H+k*10_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

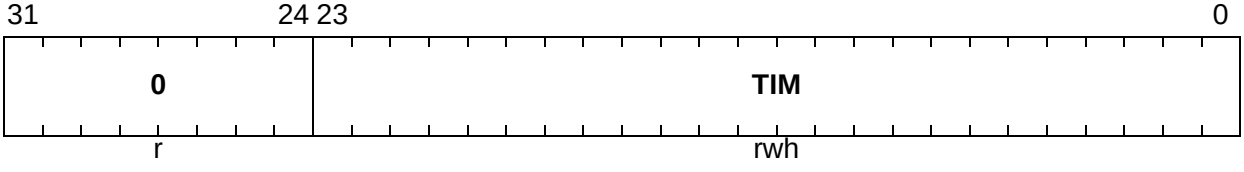
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						CRE	FRE	RRE	QCK	FCK	RCK	FZE	RZE	OCA	RCA
r						rw	rw	rw	w	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
RCA	0	rw	**Trigger Source Selection for Capture Event** 0_B Timer contents are copied to DCMCAVk capture register on a falling input signal edge 1_B Timer contents are copied to capture register on a rising input signal edge
OCA	1	rw	**Trigger Source for Capture/Compare Register Update** 0_B Capture/Compare register DCMCOVk is not affected. 1_B Timer contents are copied to DCMCOVk capture/compare register on the opposite edge selected by RCAk.
RZE	2	rw	**Timer Reset on Rising Edge** 0_B Timer is not affected 1_B Timer is reset on a rising input signal edge
FZE	3	rw	**Timer Reset on Falling Edge** 0_B Timer is not affected 1_B Timer is reset on a falling input signal edge
RCK	4	rw	**Output Pulse on Rising Edge** 0_B DCM output line is not affected 1_B DCM output line is provided with a single clock pulse generated on a rising input signal edge

Field	Bits	Type	Description
FCK	5	rw	**Output Pulse on Falling Edge** 0_B DCM output line is not affected 1_B DCM output line is provided with a single clock pulse generated on a falling input signal edge
QCK	6	w	**Additional Output Pulse Generation** 0_B DCM output line is not affected 1_B DCM output line is immediately provided with a single clock pulse QCK is always read as 0.
RRE	7	rw	**Interrupt Request on Rising Edge** 0_B Interrupt request is not affected 1_B Interrupt request is set on rising input signal edge
FRE	8	rw	**Interrupt Request on Falling Edge** 0_B Interrupt request is not affected 1_B Interrupt request is set on falling input signal edge
CRE	9	rw	**Interrupt Request on Compare Event** 0_B Interrupt request is not affected 1_B Interrupt request is set when the timer matches capture/compare register DCMCOVk
0	[31:10]	r	**Reserved** Read as 0; should be written with 0.

GPTA0_DCMTIMk (k = 0-3)
GPTA0 Duty Cycle Measurement Timer Register k
(080_H+k*10_H+4_H) **Reset Value: 0000 0000_H**

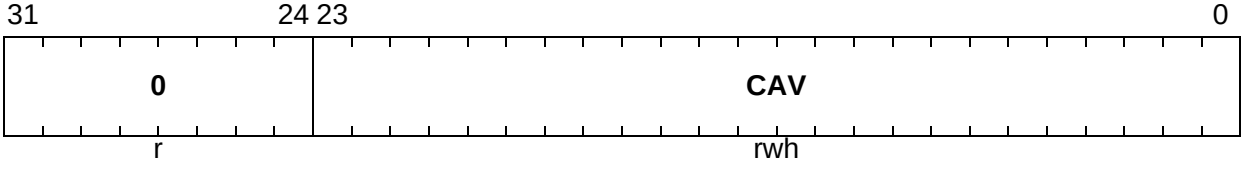
GPTA1_DCMTIMk (k = 0-3)
GPTA1 Duty Cycle Measurement Timer Register k
(080_H+k*10_H+4_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
TIM	[23:0]	rwh	**Timer Value of DCMk**
0	[31:24]	r	**Reserved** Read as 0; should be written with 0.

GPTA0_DCMCAVk (k = 0-3)
GPTA0 Duty Cycle Measurement Capture Register k
(088_H+k*10_H) **Reset Value: 0000 0000_H**

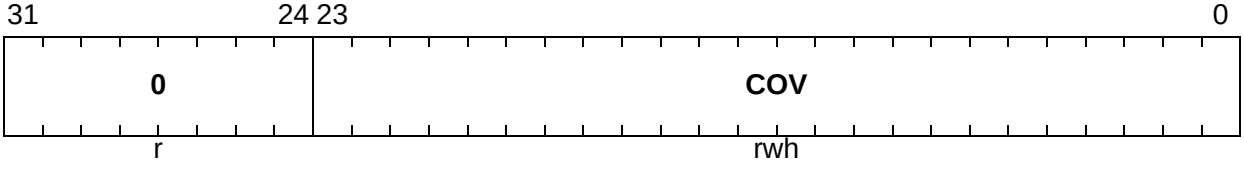
GPTA1_DCMCAVk (k = 0-3)
GPTA1 Duty Cycle Measurement Capture Register k
(088_H+k*10_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
CAV	[23:0]	rwh	**Capture Value of DCMk**
0	[31:24]	r	**Reserved** Read as 0; should be written with 0.

GPTA0_DCMCOVk (k = 0-3)
GPTA0 Duty Cycle Measurement Capture/Compare Register k
($08C_H$+k*10_H) **Reset Value: 0000 0000_H**

GPTA1_DCMCOVk (k = 0-3)
GPTA1 Duty Cycle Measurement Capture/Compare Register k
($08C_H$+k*10_H) **Reset Value: 0000 0000_H**

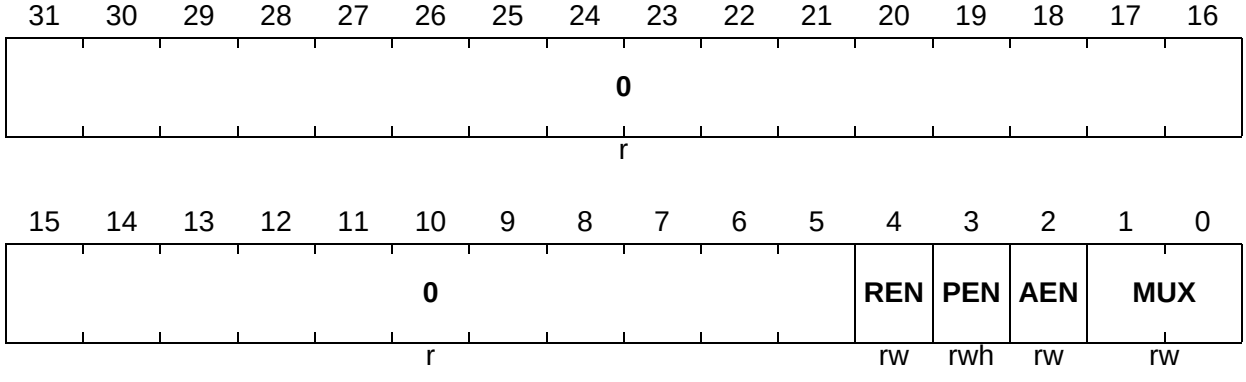


Field	Bits	Type	Description
COV	[23:0]	rwh	**Capture/Compare Register Value of DCMk**
0	[31:24]	r	**Reserved** Read as 0; should be written with 0.

22.4.5 Digital Phase Locked Loop Registers

GPTA0_PLLCTR
GPTA0 Phase Locked Loop Control Register
($0C0_H$) **Reset Value: 0000 0000_H**

GPTA1_PLLCTR
GPTA1 Phase Locked Loop Control Register
($0C0_H$) **Reset Value: 0000 0000_H**

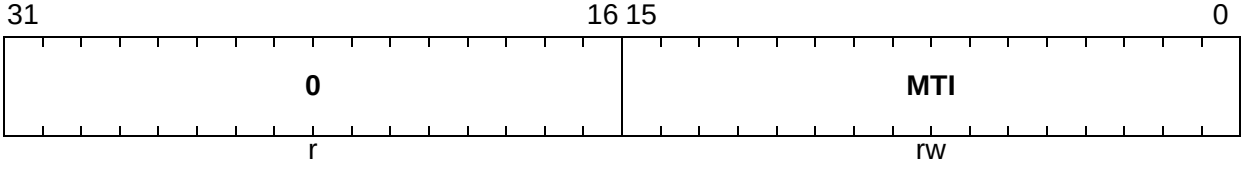


Field	Bits	Type	Description
MUX	[1:0]	rw	**Trigger Input Channel Selection** 00_B DCM0 output is selected as PLL input 01_B DCM1 output is selected as PLL input 10_B DCM2 output is selected as PLL input 11_B DCM3 output is selected as PLL input
AEN	2	rw	**Automatic End Mode Enable** With the Automatic End Mode compensation of input signal's period length variation (acceleration, deceleration) is requested 0_B Automatic End Mode is disabled 1_B Automatic End Mode is enabled
PEN	3	rwh	**Unexpected Period End Behavior** 0_B Counter decrements with constant frequency 1_B Counter is allowed to decrement with f_{GPTA} frequency in case of an input signal period length' reduction Programming PEN to 1 immediately changes the microtick counter to decrement with f_{GPTA} frequency. This bit is protected during read-modify-write operations (hardware will win).

Field	Bits	Type	Description
REN	4	rw	**Interrupt Service Request Enable** 0_B Interrupt request is disabled 1_B An interrupt request is set when the number of remaining output pulses to be generated reaches zero
0	[31:5]	r	**Reserved** Read as 0; should be written with 0.

GPTA0_PLLMTI
GPTA0 Phase Locked Loop Microtick Register
(0C4$_H$) **Reset Value: 0000 0000$_H$**

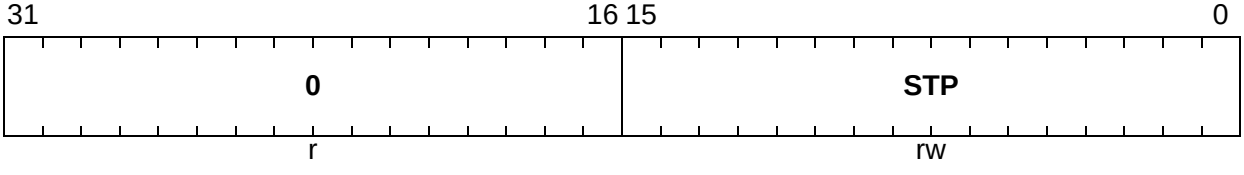
GPTA1_PLLMTI
GPTA1 Phase Locked Loop Microtick Register
(0C4$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
MTI	[15:0]	rw	**Microtick Value** Number of output pulses to be generated within one input signal period.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

GPTA0_PLLSTP
GPTA0 Phase Locked Loop Step Register
($0CC_H$) **Reset Value: 0000 0000_H**

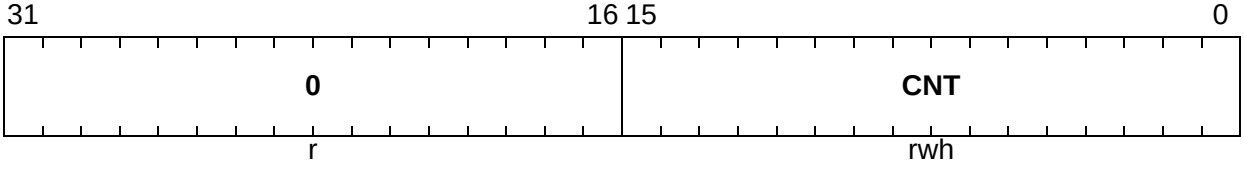
GPTA1_PLLSTP
GPTA1 Phase Locked Loop Step Register
($0CC_H$) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
STP	[15:0]	rw	**Step Value** Number of output pulses to be generated within one input signal period (2-complement data format).
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

GPTA0_PLLCNT
GPTA0 Phase Locked Loop Counter Register
($0C8_H$) **Reset Value: 0000 0000_H**

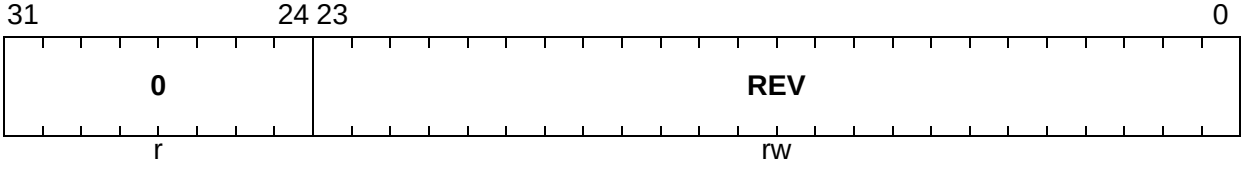
GPTA1_PLLCNT
GPTA1 Phase Locked Loop Counter Register
($0C8_H$) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
CNT	[15:0]	rwh	**Pulse Counter** Counter for the number of remaining output pulses to be generated.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

GPTA0_PLLREV
GPTA0 Phase Locked Loop Reload Register
($0D0_H$) **Reset Value: 0000 0000_H**

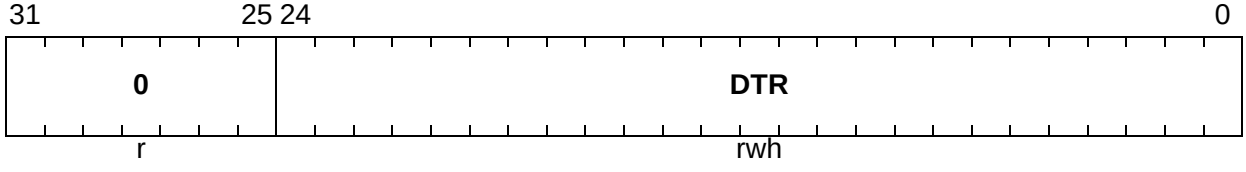
GPTA1_PLLREV
GPTA1 Phase Locked Loop Reload Register
($0D0_H$) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
REV	[23:0]	rw	**Reload Value** Reload value calculated by a subtraction of the number of output pulses to be generated within one input signal period from the input signal's period length (measured in number of GPTA®v5 module clocks).
0	[31:24]	r	**Reserved** Read as 0; should be written with 0.

GPTA0_PLLDTR
GPTA0 Phase Locked Loop Delta Register
($0D4_H$) **Reset Value: 0000 0000_H**

GPTA1_PLLDTR
GPTA1 Phase Locked Loop Delta Register
($0D4_H$) **Reset Value: 0000 0000_H**

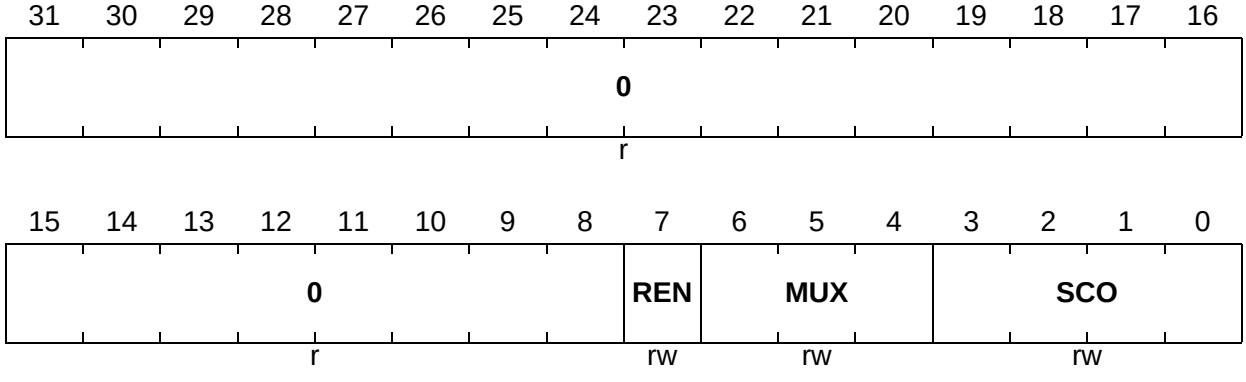


Field	Bits	Type	Description
DTR	[24:0]	rwh	**Delta Register Value** Internal register used to store intermediate results for output pulse generation. **Do not write to this register while PLL is running!**
0	[31:25]	r	**Reserved** Read as 0; should be written with 0.

22.4.6 Global Timer Registers

GPTA0_GTCTRk (k = 0-1)
GPTA0 Global Timer Control Register k
($0E0_H$+k*10_H) **Reset Value: 0000 0000_H**

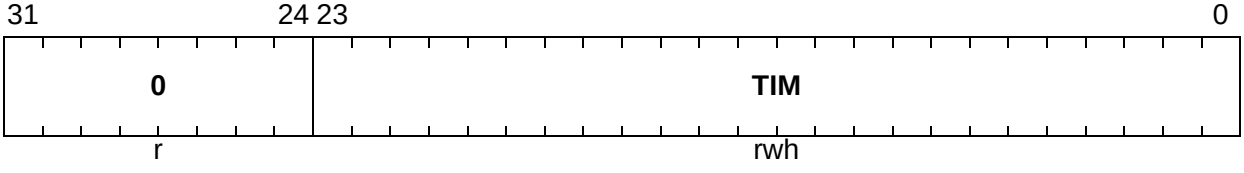
GPTA1_GTCTRk (k = 0-1)
GPTA1 Global Timer Control Register k
($0E0_H$+k*10_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
SCO	[3:0]	rw	**TGE Flag Source Selection** This bit field determines the bit of the operation result “GTk timer value - data bus value” which is used as TGE flag. 0000_B 10^{th} bit is used as TGE flag. 0001_B 11^{th} bit is used as TGE flag. $…_B$ … 1110_B 24^{th} bit is used as TGE flag. 1111_B 25^{th} bit is used as TGE flag.
MUX	[6:4]	rw	**Timer Clock Selection** One of eight available clock bus lines is selected as the timer GTk clock. 000_B Clock bus line CLK0 selected 001_B Clock bus line CLK1 selected 010_B Clock bus line CLK2 selected 011_B Clock bus line CLK3 selected 100_B Clock bus line CLK4 selected 101_B Clock bus line CLK5 selected 110_B Clock bus line CLK6 selected 111_B Clock bus line CLK7 selected

Field	Bits	Type	Description
REN	7	rw	**Interrupt Request Enable** 0_B The interrupt request is disabled 1_B An interrupt request is generated when timer GTk overflows
0	[31:8]	r	**Reserved** Read as 0; should be written with 0.

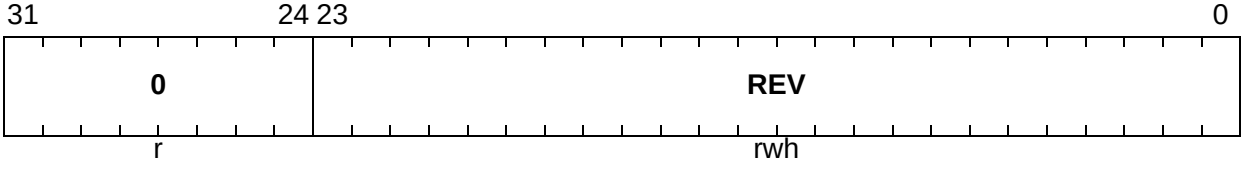
GPTA0_GTTIMk (k = 0-1)
GPTA0 Global Timer Register k **($0E8_H$+k*10_H)** **Reset Value: 0000 0000_H**
GPTA1_GTTIMk (k = 0-1)
GPTA1 Global Timer Register k **($0E8_H$+k*10_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
TIM	[23:0]	rwh	**Timer Value of Global Timer k**
0	[31:24]	r	**Reserved** Read as 0; should be written with 0.

GPTA0_GTREVk (k = 0-1)
GPTA0 Global Timer Reload Value Register k
($0E4_H$+k*10_H) **Reset Value: 0000 0000_H**

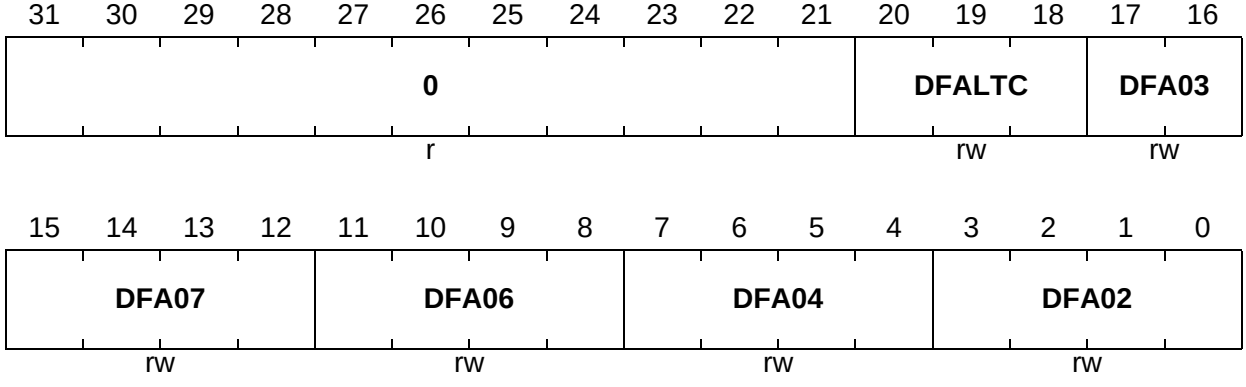
GPTA1_GTREVk (k = 0-1)
GPTA1 Global Timer Reload Value Register k
($0E4_H$+k*10_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
REV	[23:0]	rw	**Reload Value of Global Timer k** Reload value for timer GTk after an overflow
0	[31:24]	r	**Reserved** Read as 0; should be written with 0.

22.4.7 Clock Bus Register

GPTA0_CKBCTR
GPTA0 Clock Bus Control Register (0D8$_H$) Reset Value: 0000 FFFF$_H$
GPTA1_CKBCTR
GPTA1 Clock Bus Control Register (0D8$_H$) Reset Value: 0000 FFFF$_H$



Field	Bits	Type	Description
DFA02	[3:0]	rw	**Clock Line 2 Driving Source Selection** 0_D CLK2 is provided with the GPTA®v5 module clock f_{GPTA} divided by 2^{DFA02} 1_D CLK2 is provided with the GPTA®v5 module clock f_{GPTA} divided by 2^{DFA02} …$_D$ … 13_D CLK2 is provided with the GPTA®v5 module clock f_{GPTA} divided by 2^{DFA02} 14_D CLK2 is driven by PLL clock of other GPTA®v5 unit 15_D CLK2 is driven by DCM3 output
DFA04	[7:4]	rw	**Clock Line 4 Driving Source Selection** 0_D CLK4 is provided with the GPTA®v5 module clock f_{GPTA} divided by 2^{DFA04} 1_D CLK4 is provided with the GPTA®v5 module clock f_{GPTA} divided by 2^{DFA04} …$_D$ … 14_D CLK4 is provided with the GPTA®v5 module clock f_{GPTA} divided by 2^{DFA04} 15_D CLK4 is driven by DCM1 output

Field	Bits	Type	Description
DFA06	[11:8]	rw	**Clock Line 6 Driving Source Selection** 0_D CLK6 is provided with the GPTA®v5 module clock f_{GPTA} divided by 2^{DFA06} 1_D CLK6 is provided with the GPTA®v5 module clock f_{GPTA} divided by 2^{DFA06} …$_D$ … 14_D CLK6 is provided with the GPTA®v5 module clock f_{GPTA} divided by 2^{DFA06} 15_D CLK6 is driven by FPC1 output
DFA07	[15:12]	rw	**Clock Line 7 Driving Source Selection** 0_D CLK7 is provided with the GPTA®v5 module clock f_{GPTA} divided by 2^{DFA07} 1_D CLK7 is provided with the GPTA®v5 module clock f_{GPTA} divided by 2^{DFA07} …$_D$ … 14_D CLK7 is provided with the GPTA®v5 module clock f_{GPTA} divided by 2^{DFA07} 15_D CLK7 is driven by FPC4 output
DFA03	[17:16]	rw	**Clock Line 3 Driving Source Selection** 0_B CLK3 is driven by DCM2 output 1_B CLK3 is driven by PLL clock of other GPTA®v5 unit 2_B CLK3 is driven by uncompensated PLL clock 3_B CLK3 is driven by uncompensated PLL clock of other GPTA®v5 unit
DFALTC	[20:18]	rw	**Dividing Factor for LTC Prescaler Clock Selection** 0_D The LTCPRE clock is provided with the GPTA®v5 module clock f_{GPTA} divided by 2^{DFALTC}. 1_D The LTCPRE clock is provided with the GPTA®v5 module clock f_{GPTA} divided by 2^{DFALTC}. …$_D$ … 7_D The LTCPRE clock is provided with the GPTA®v5 module clock f_{GPTA} divided by 2^{DFALTC}.
0	[31:21]	r	**Reserved** Read as 0; should be written with 0.

22.4.8 Global Timer Cell Registers

GPTA0_GTCCTRk (k = 00-31)
GPTA0 Global Timer Cell Control Register k [Capture Mode]
(100_H+k*8_H) **Reset Value: 0000 0000_H**

GPTA1_GTCCTRk (k = 00-31)
GPTA1 Global Timer Cell Control Register k [Capture Mode]
(100_H+k*8_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OUT	OIA	OCM			CEN	0	EOA	BYP	NE	FED	RED	REN	OSM	MOD	
rh	rw	rw			rh	r	rwh	rw	rw	rw	rw	rw	rw	rw	

Field	Bits	Type	Description
MOD	[1:0]	rw	**Mode Control Bits** 00_B GTCk operates in Capture Mode hooked to GT0. 01_B GTCk operates in Capture Mode hooked to GT1. 10_B GTCk operates in Compare Mode hooked to GT0. 11_B GTCk operates in Compare Mode hooked to GT1.
OSM	2	rw	**One Shot Mode Enable** 0_B GTCk is continuously enabled. 1_B GTCk is enabled for one event only.
REN	3	rw	**Interrupt Request Enable** 0_B Service request is disabled. 1_B Service request line SQSk is activated when a capture or compare event has occurred.
RED	4	rw	**Input Rising Edge Select** 0_B Capture event is not triggered by a rising edge. 1_B Capture event is triggered by a rising edge on the GTCkIN input line.
FED	5	rw	**Input Falling Edge Select** 0_B Capture event is not triggered by a falling edge. 1_B Capture event is triggered by a falling edge on the GTCkIN input line.

Field	Bits	Type	Description
NE	6	rw	**Not Effective** Reserved
BYP	7	rw	**Bypass** 0_B M0O/M1O lines are affected either by M0I/M1I lines or by OCM0/OCM1 bits. 1_B M0O/M1O lines are affected only by M0I/M1I lines. *Note: OCM2 must be set in any case to enable reaction on M0I/M1I changes.*
EOA	8	rwh	**Enable On Action** 0_B GTCk is enabled for local events. 1_B GTCk is disabled for local events. On an event on the communication link via M0I/M1I lines, EOA will be cleared and local events will be enabled. EOA is bit protected (see **Section 22.4.2**). EOA is cleared if mode is switched to Timer Mode.
CEN	10	rh	**Cell Enable** 0_B GTCk is currently disabled for local events. 1_B GTCk is currently enabled for local events.
OCM	[13:11]	rw	**Output Control Mode Select** $X00_B$ Current state of GTCkOUT output line is hold. $X01_B$ Current state of GTCkOUT output line is toggled. $X10_B$ GTCkOUT output line is forced to 0. $X11_B$ GTCkOUT output line is forced to 1. $0XX_B$ GTCkOUT output line state is set by an internal GTCk event only. $1XX_B$ GTCkOUT output line state is affected by an internal GTCk event and/or by an operation occurred in an adjacent GTCn (n = less or equal k) and reported by the M1I, M0I interface lines.
OIA	14	rw	**Output Immediate Action** 0_B No immediate action required. 1_B Action defined by OCM must be performed immediately. Reading bit OIA always returns 0.
OUT	15	rh	**Output State** 0_B GTCkOUT output line is 0. 1_B GTCkOUT output line is 1.
0	9, [31:16]	r	**Reserved** Read as 0; should be written with 0.

GPTA0_GTCCTRk (k = 00-31)
GPTA0 Global Timer Cell Control Register k [Compare Mode]
(100_H+k*8_H) **Reset Value: 0000 0000_H**

GPTA1_GTCCTRk (k = 00-31)
GPTA1 Global Timer Cell Control Register k [Compare Mode]
(100_H+k*8_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OUT	OIA	OCM			CEN	0	EOA	BYP	CAT	CAC	GES	REN	OSM	MOD	
rh	rw	rw			rh	r	rwh	rw	rw	rw	rw	rw	rw	rw	

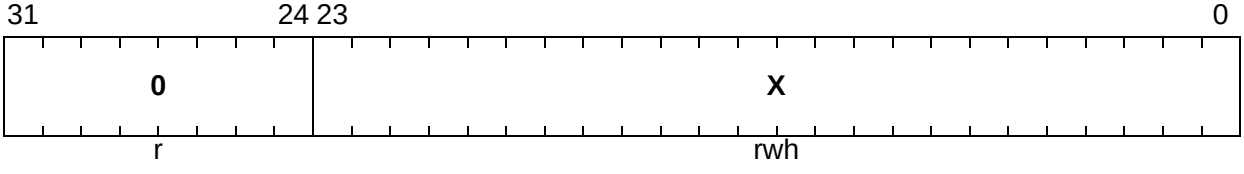
Field	Bits	Type	Description
MOD	[1:0]	rw	**Mode Control Bits** 00_B GTCk operates in Capture Mode hooked to GT0. 01_B GTCk operates in Capture Mode hooked to GT1. 10_B GTCk operates in Compare Mode hooked to GT0. 11_B GTCk operates in Compare Mode hooked to GT1.
OSM	2	rw	**One Shot Mode Enable** 0_B GTCk is continuously enabled. 1_B GTCk is enabled for one event only.
REN	3	rw	**Interrupt Request Enable** 0_B Service request is disabled. 1_B Service request line SQSk is activated when a capture or compare event has occurred.
GES	4	rw	**Greater Equal Select** 0_B An “equal” compare is selected. 1_B A “greater equal” compare is required.
CAC	5	rw	**Capture after Compare Select** 0_B Capture after compare is disabled. 1_B After a compare event, the contents of the associated Global Timer as selected by MOD or (depending on control bit CAT) the contents of the alternate Global Timer are copied to the capture/compare register GTCXRk.

Field	Bits	Type	Description
CAT	6	rw	**Capture Alternate Timer** 0_B The Global Timer as selected by MOD is captured, if enabled by control bit CAC = 1. 1_B The alternate Global Timer is captured.
BYP	7	rw	**Bypass** 0_B M1O/M0O lines are affected either by M1I/M0I lines or by OCM1/OCM0 bits. 1_B M0O/M1O lines are affected only by M0I/M1I lines. *Note: OCM2 must be set in any case to enable reaction on M0I/M1I changes.*
EOA	8	rwh	**Enable On Action** 0_B GTCk is enabled for local events. 1_B GTCk is disabled for local events. On an event on the communication link via M0I/M1I lines, EOA will be cleared and local events will be enabled. EOA is bit protected (see **Section 22.4.2**). EOA is cleared if mode is switched to Timer Mode.
CEN	10	rh	**Cell Enable** 0_B GTCk is currently disabled for local events. 1_B GTCk is currently enabled for local events.
OCM	[13:11]	rw	**Output Control Mode Select** $X00_B$ Current state of GTCkOUT output line is hold. $X01_B$ Current state of GTCkOUT output line is toggled. $X10_B$ GTCkOUT output line is forced to 0. $X11_B$ GTCkOUT output line is forced to 1. $0XX_B$ GTCkOUT output line state is set by an internal GTCk event only. $1XX_B$ GTCkOUT output line state is affected by an internal GTCk event and/or by an operation occurred in an adjacent GTCn (n = less or equal k) and reported by the M1I, M0I interface lines.
OIA	14	rw	**Output Immediate Action** 0_B No immediate action required. 1_B Action defined by OCM must be performed immediately. Reading bit OIA always returns 0.
OUT	15	rh	**Output State** 0_B GTCkOUT output line is 0. 1_B GTCkOUT output line is 1.

Field	Bits	Type	Description
0	9, [31:16]	r	**Reserved** Read as 0; should be written with 0.

GPTA0_GTCXRk (k = 00-31)
GPTA0 Global Timer Cell X Register k
(104_H+k*8_H) **Reset Value: 0000 0000_H**

GPTA1_GTCXRk (k = 00-31)
GPTA1 Global Timer Cell X Register k
(104_H+k*8_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
X	[23:0]	rwh	**Capture/Compare Register Contents of GTCk**
0	[31:24]	r	**Reserved** Read as 0; should be written with 0.

Note: GTCXRk is write-protected when control bits CAC and OSM are set to 1 (“capture after compare” in Single Shot Mode). Write protection is activated, when the value of the selected GT timer matches and/or exceeds the capture/compare register contents. Write protection is released after a software access to register GTCXRk.

22.4.9 Local Timer Cell Registers

GPTA0_LTCCTRk (k = 00-62)
GPTA0 Local Timer Cell Control Register k [Timer Mode; ILM = 0]
(200_H+k*8_H) Reset Value: 0000 0000_H

GPTA1_LTCCTRk (k = 00-62)
GPTA1 Local Timer Cell Control Register k [Timer Mode; ILM = 0]
(200_H+k*8_H) Reset Value: 0000 0000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															GBYP
r															rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OUT	OIA	OCM			CEN	CUD	ILM	CUD CLR	SLO	FED	RED	REN	OSM	MOD	
rh	rw	rw			rh	rwh	rw	w	rwh	rw	rw	rw	rw	rw	

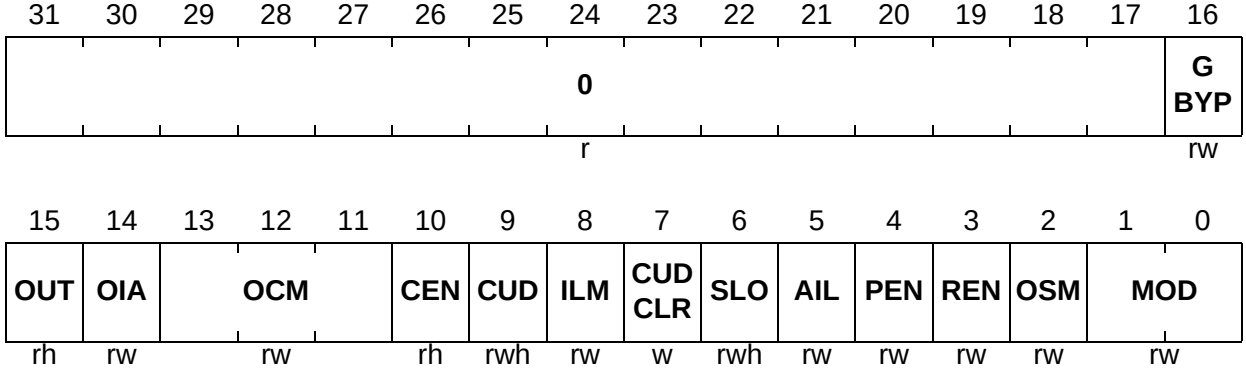
Field	Bits	Type	Description
MOD	[1:0]	rw	**Mode Control Bits** 00_B LTCk operates in Capture Mode. 01_B LTCk operates in Compare Mode. 10_B LTCk operates in Free-Running Timer Mode. 11_B LTCk operates in reset Timer Mode.
OSM	2	rw	**One Shot Mode Enable** 0_B LTCk is continuously enabled. 1_B LTCk is enabled for one event only.
REN	3	rw	**Request Enable** 0_B Service request is disabled. 1_B Service request SQSk is activated when a - capture event has occurred - compare event has occurred - timer overflow has happened depending on the operation mode selected by bit field MOD.
RED	4	rw	**Input Rising Edge Select** 0_B Timer is not updated by a rising edge. 1_B Timer is updated by a rising edge on the LTCkIN input line.

Field	Bits	Type	Description
FED	5	rw	**Input Falling Edge Select** 0_B Timer is not updated by a falling edge. 1_B Timer is updated by a falling edge on the LTCkIN input line.
SLO	6	rwh	**Select Line Output** 0_B State of select line output SO is 0. 1_B State of select line output SO is 1. SLO is bit protected (see **Page 22-167**).
CUDCLR	7	w	**Coherent Update Disable** 0_B No effect. 1_B Coherent update disabled (bit CUD is cleared). If bits CUD and CUDCLR are both written with 1, bit CUD will be set. CUDCLR is always read as 0.
ILM	8	rw	**Input Line Mode** 0_B Input line is operating in Edge Sensitive Mode. 1_B Input line is operating in Level Sensitive Mode. In case of full speed GPTA®v5 module clock selection as input clock, Level Sensitive Mode must be selected. In this case the Edge Sensitive Mode will not produce any event.
CUD	9	rwh	**Coherent Update Enable** 0_B Select output SO is not toggled on timer reset overflow. 1_B Select output SO is toggled on next timer reset overflow. When CUD is set by software, it remains set until the next timer reset overflow (LTCk reset event) occurs and is cleared by hardware afterwards. CUD can be reset by software by writing bit CUDCLR with 1 and CUD with 0. CUD is automatically cleared after LTCk reset event and when mode is switched to another mode than Reset Timer Mode. This bit can only be set in Reset Timer Mode. If bits CUD and CUDCLR are both written with 1, bit CUD will be set. CUDCLR is always read as 0.
CEN	10	rh	**Cell Enable** 0_B LTCk is currently disabled for local events. 1_B LTCk is currently enabled for local events.

Field	Bits	Type	Description
OCM	[13:11]	rw	**Output Control Mode Select** $X00_B$ Current state of LTCkOUT output line is hold. $X01_B$ Current state of LTCkOUT output line is toggled. $X10_B$ LTCkOUT output line is forced to 0. $X11_B$ LTCkOUT output line is forced to 1. $0XX_B$ LTCkOUT output line state is set by an internal LTCk event only. $1XX_B$ LTCkOUT output line state is affected by an internal LTCk event and/or by an operation occurred in an adjacent LTCk cell (reported by M1I/M0I interface lines).
OIA	14	rw	**Output Immediate Action** 0_B No immediate action required. 1_B Action defined by bit field OCM must be performed immediately. OIA is always read as 0.
OUT	15	rh	**Output State** 0_B LTCkOUT output line is 0. 1_B LTCkOUT output line is 1.
GBYP	16	rw	**Global Bypass** 0_B M3O/M2O lines are affected by M1I/M0I lines. 1_B M3O/M2O lines are affected by M3I/M2I lines.
0	[31:17]	r	**Reserved** Read as 0; should be written with 0.

GPTA0_LTCCTRk (k = 00-62)
GPTA0 Local Timer Cell Control Register k [Timer Mode; ILM = 1]
(200_H+k*8_H) **Reset Value: 0000 0000_H**

GPTA1_LTCCTRk (k = 00-62)
GPTA1 Local Timer Cell Control Register k [Timer Mode; ILM = 1]
(200_H+k*8_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
MOD	[1:0]	rw	**Mode Control Bits** 00_B LTCk operates in Capture Mode. 01_B LTCk operates in Compare Mode. 10_B LTCk operates in Free-Running Timer Mode. 11_B LTCk operates in reset Timer Mode.
OSM	2	rw	**One Shot Mode Enable** 0_B LTCk is continuously enabled. 1_B LTCk is enabled for one event only.
REN	3	rw	**Request Enable** 0_B Service request is disabled. 1_B Service request SQSk is activated when a - capture event has occurred - compare event has occurred - timer overflow has happened depending on the operation mode selected by bit field MOD.
PEN	4	rw	**LTC Prescaler Enable** 0_B LTC Prescaler Mode is disabled. 1_B LTC Prescaler Mode with LTC prescaler clock LTCPRE is enabled.

Field	Bits	Type	Description
AIL	5	rw	**Active Input Level Select** 0_B Input signal is active high. 1_B Input signal is active low.
SLO	6	rwh	**Select Line Output** 0_B State of select line output SO is 0. 1_B State of select line output SO is 1. SLO is bit protected (see **Page 22-167**).
CUDCLR	7	w	**Coherent Update Disable** 0_B No effect. 1_B Coherent update disabled (bit CUD is cleared). If bits CUD and CUDCLR are both written with 1, bit CUD will be set. CUDCLR is always read as 0.
ILM	8	rw	**Input Line Mode** 0_B Input line is operating in Edge Sensitive Mode. 1_B Input line is operating in Level Sensitive Mode. In case of full speed GPTA®v5 module clock selection as input clock, Level Sensitive Mode must be selected. In this case the Edge Sensitive Mode will not produce any event.
CUD	9	rwh	**Coherent Update Enable** 0_B Select output SO is not toggled on timer reset overflow. 1_B Select output SO is toggled on next timer reset overflow. When CUD is set by software, it remains set until the next timer reset overflow (LTCk reset event) occurs and is cleared by hardware afterwards. CUD can be reset by software by writing bit CUDCLR with 1 and CUD with 0. CUD is automatically cleared after LTCk reset event and when mode is switched to another mode than Reset Timer Mode. This bit can only be set in Reset Timer Mode. If bits CUD and CUDCLR are both written with 1, bit CUD will be set. CUDCLR is always read as 0.
CEN	10	rh	**Cell Enable** 0_B LTCk is currently disabled for local events. 1_B LTCk is currently enabled for local events.

Field	Bits	Type	Description
OCM	[13:11]	rw	**Output Control Mode Select** $X00_B$ Current state of LTCkOUT output line is hold. $X01_B$ Current state of LTCkOUT output line is toggled. $X10_B$ LTCkOUT output line is forced to 0. $X11_B$ LTCkOUT output line is forced to 1. $0XX_B$ LTCkOUT output line state is set by an internal LTCk event only. $1XX_B$ LTCkOUT output line state is affected by an internal LTCk event and/or by an operation occurred in an adjacent LTCk cell (reported by M1I/M0I interface lines).
OIA	14	rw	**Output Immediate Action** 0_B No immediate action required. 1_B Action defined by bit field OCM must be performed immediately. OIA is always read as 0.
OUT	15	rh	**Output State** 0_B LTCkOUT output line is 0. 1_B LTCkOUT output line is 1.
GBYP	16	rw	**Global Bypass** 0_B M3O/M2O lines are affected by M1I/M0I lines. 1_B M3O/M2O lines are affected by M3I/M2I lines.
0	[31:17]	r	**Reserved** Read as 0; should be written with 0.

GPTA0_LTCCTRk (k = 00-62)
GPTA0 Local Timer Cell Control Register k [Capture Mode]
(200_H+k*8_H) **Reset Value: 0000 0000_H**

GPTA1_LTCCTRk (k = 00-62)
GPTA1 Local Timer Cell Control Register k [Capture Mode]
(200_H+k*8_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															GBYP
r															rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OUT	OIA	OCM			CEN	SLL	ILM	EOA	BYP	FED	RED	REN	OSM	MOD	
rh	rw	rw			rh	rh	rw	rwh	rw	rw	rw	rw	rw	rw	

Field	Bits	Type	Description
MOD	[1:0]	rw	**Mode Control Bits** 00_B LTCk operates in Capture Mode. 01_B LTCk operates in Compare Mode. 10_B LTCk operates in Free-Running Timer Mode. 11_B LTCk operates in reset Timer Mode.
OSM	2	rw	**One Shot Mode Enable** 0_B LTCk is continuously enabled. 1_B LTCk is enabled for one event only.
REN	3	rw	**Request Enable** 0_B Service request is disabled. 1_B Service request SQSk is activated when a - capture event has occurred - compare event has occurred - timer overflow has happened depending on the operation mode selected by bit field MOD.
RED	4	rw	**Input Rising Edge Select** 0_B Capture event is not triggered by a rising edge. 1_B Capture event is triggered by a rising edge on the LTCkIN input line.

Field	Bits	Type	Description
FED	5	rw	**Input Falling Edge Select** 0_B Capture event is not triggered by a falling edge. 1_B Capture event is triggered by a falling edge on the LTCkIN input line.
BYP	6	rw	**Local Bypass** 0_B M1O/M0O lines are affected either by M1I/M0I lines or by OCM1/OCM0 bits. 1_B M1O/M0O lines are affected only by M1I/M0I (GBYP = 0) or M2I/M2I (GBYP = 1) lines. This bit is cleared if mode is switched to Timer Mode. OCM2 must be set in any case to enable reaction on M1I/M0I change.
EOA	7	rwh	**Enable On Action** 0_B LTCk is enabled for local events. 1_B LTCk is disabled for local events. On an event on the communication link via M0I/M1I lines, EOA will be cleared and local events will be enabled. EOA is bit protected (see **Section 22.4.2**). EOA is cleared if mode is switched to Timer Mode.
ILM	8	rw	**Input Line Mode** 0_B Input line is operating in Edge Sensitive Mode. 1_B Input line is operating in Level Sensitive Mode. In case of full speed GPTA®v5 module clock selection as input clock, Level Sensitive Mode must be selected. In this case the Edge Sensitive Mode will not produce any event.
SLL	9	rh	**Capture & Compare Mode: Select Line Level** 0_B Current state of select input SI is 0. 1_B Current state of select input SI is 1.
CEN	10	rh	**Cell Enable** 0_B LTCk is currently disabled for local events. 1_B LTCk is currently enabled for local events.

Field	Bits	Type	Description
OCM	[13:11]	rw	**Output Control Mode Select** $X00_B$ Current state of LTCkOUT output line is hold. $X01_B$ Current state of LTCkOUT output line is toggled. $X10_B$ LTCkOUT output line is forced to 0. $X11_B$ LTCkOUT output line is forced to 1. $0XX_B$ LTCkOUT output line state is set by an internal LTCk event only. $1XX_B$ LTCkOUT output line state is affected by an internal LTCk event and/or by an operation occurred in an adjacent LTCk cell (reported by M1I/M0I interface lines).
OIA	14	rw	**Output Immediate Action** 0_B No immediate action required. 1_B Action defined by bit field OCM must be performed immediately. OIA is always read as 0.
OUT	15	rh	**Output State** 0_B LTCkOUT output line is 0. 1_B LTCkOUT output line is 1.
GBYP	16	rw	**Global Bypass** 0_B M3O/M2O lines are affected by M1I/M0I lines. 1_B M3O/M2O lines are affected by M3I/M2I lines.
0	[31:17]	r	**Reserved** Read as 0; should be written with 0.

GPTA0_LTCCTRk (k = 00-62)
GPTA0 Local Timer Cell Control Register k [Compare Mode]
(200_H+k*8_H) **Reset Value: 0000 0000_H**

GPTA1_LTCCTRk (k = 00-62)
GPTA1 Local Timer Cell Control Register k [Compare Mode]
(200_H+k*8_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															G BYP
r															rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OUT	OIA	OCM			CEN	SLL	ILM	EOA	BYP	SOH	SOL	REN	OSM	MOD	
rh	rw	rw			rh	rh	rw	rwh	rw	rw	rw	rw	rw	rw	

Field	Bits	Type	Description
MOD	[1:0]	rw	**Mode Control Bits** 00_B LTCk operates in Capture Mode. 01_B LTCk operates in Compare Mode. 10_B LTCk operates in Free-Running Timer Mode. 11_B LTCk operates in reset Timer Mode.
OSM	2	rw	**One Shot Mode Enable** 0_B LTCk is continuously enabled. 1_B LTCk is enabled for one event only.
REN	3	rw	**Request Enable** 0_B Service request is disabled. 1_B Service request SQSk is activated when a - capture event has occurred - compare event has occurred - timer overflow has happened depending on the operation mode selected by bit field MOD.
SOL	4	rw	**Compare Mode: Select Output Low** 0_B Compare is deactivated or on high level. 1_B Compare operation is enabled by a low level on select input SI[1)].

Field	Bits	Type	Description
SOH	5	rw	**Compare Mode: Select Output High** 0_B Compare is deactivated or on high level. 1_B Compare operation is enabled by a high level on select input SI[1)].
BYP	6	rw	**Bypass** 0_B M1O/M0O lines are affected either by M1I/M0I lines or by OCM1/OCM0 bits. 1_B M1O/M0O lines are affected only by M1I/M0I lines. This bit is cleared if mode is switched to Timer Mode. OCM2 must be set in any case to enable reaction on M1I/M0I change.
EOA	7	rwh	**Enable On Action** 0_B LTCk is enabled for local events. 1_B LTCk is disabled for local events. On an event on the communication link via M0I/M1I lines, EOA will be cleared and local events will be enabled. EOA is bit protected (see **Section 22.4.2**). EOA is cleared if mode is switched to Timer Mode.
ILM	8	rw	**Input Line Mode** 0_B Input line is operating in Edge Sensitive Mode. 1_B Input line is operating in Level Sensitive Mode. In case of full speed GPTA®v5 module clock selection as input clock, Level Sensitive Mode must be selected. In this case the Edge Sensitive Mode will not produce any event.
SLL	9	rh	**Select Line Level** 0_B Current state of select input SI is 0. 1_B Current state of select input SI is 1.
CEN	10	rh	**Cell Enable** 0_B LTCk is currently disabled for local events. 1_B LTCk is currently enabled for local events.
OCM	[13:11]	rw	**Output Control Mode Select** $X00_B$ Current state of LTCkOUT output line is hold. $X01_B$ Current state of LTCkOUT output line is toggled. $X10_B$ LTCkOUT output line is forced to 0. $X11_B$ LTCkOUT output line is forced to 1. $0XX_B$ LTCkOUT output line state is set by an internal LTCk event only. $1XX_B$ LTCkOUT output line state is affected by an internal LTCk event and/or by an operation occurred in an adjacent LTCk cell (reported by M1I/M0I interface lines).

Field	Bits	Type	Description
OIA	14	rw	**Output Immediate Action** 0_B No immediate action required. 1_B Action defined by bit field OCM must be performed immediately. OIA is always read as 0.
OUT	15	rh	**Output State** 0_B LTCkOUT output line is 0. 1_B LTCkOUT output line is 1.
GBYP	16	rw	**Global Bypass** 0_B M3O/M2O lines are affected by M1I/M0I lines. 1_B M3O/M2O lines are affected by M3I/M2I lines.
0	[31:17]	r	**Reserved** Read as 0; should be written with 0.

1) To enable Compare Mode in all cases, SOL and SOH bits must be set to 1.

GPTA0_LTCCTR63
GPTA0 Local Timer Cell Control Register 63($3F8_H$) **Reset Value: 0000 0000$_H$**
GPTA1_LTCCTR63
GPTA1 Local Timer Cell Control Register 63($3F8_H$) **Reset Value: 0000 0000$_H$**

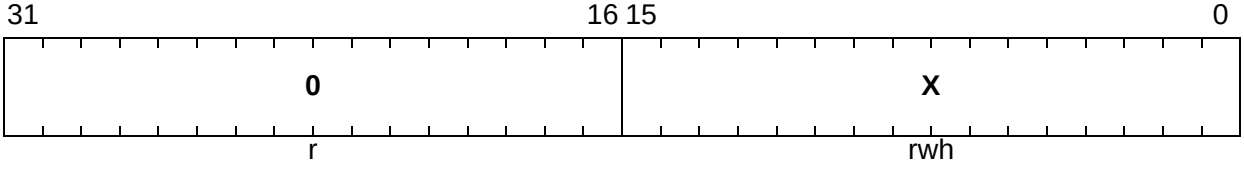
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OUT	0				CEN	0	ILM	0		FED	RED	REN		OSM	BRM
rh	r				rh	r	rw	r		rw	rw	rw		rw	rw

Field	Bits	Type	Description
BRM	0	rw	**Bit Reversal Mode Control** 0_B Compare uses normal sequence of local input data bus (YI) bits. 1_B Compare uses reversed sequence of local input data bus (YI) bits.
OSM	1	rw	**One Shot Mode Enable for Shadow Register Copy** 0_B Shadow register copy is continuously enabled. 1_B Shadow register copy is enabled for one event only.
REN	[3:2]	rw	**Request Enable** 00_B Service request SQT63 is disabled. 01_B Service request SQT63 is generated when a compare event has occurred. 10_B Service request SQT63 is generated when a shadow register copy event has occurred. 11_B Reserved.
RED	4	rw	**Rising Edge Select for Shadow Register Copy** 0_B Shadow register copy is not triggered by a rising edge on the LTC63IN input line. 1_B Shadow register copy is triggered by a rising edge on the LTC63IN input line.

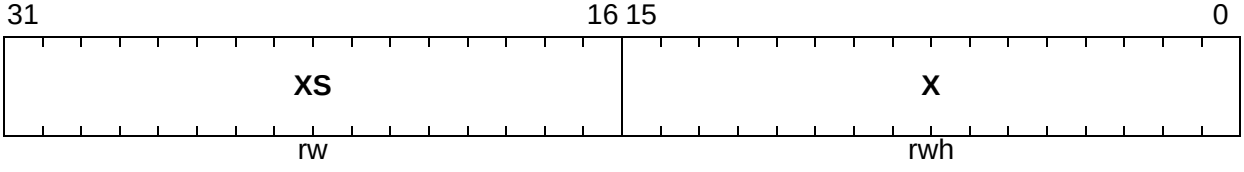
Field	Bits	Type	Description
FED	5	rw	**Falling Edge Select for Shadow Register Copy** 0_B Shadow register copy is not triggered by a falling edge on the LTC63IN input line. 1_B Shadow register copy is triggered by a falling edge on the LTC63IN input line.
ILM	8	rw	**Shadow Register Copy Input Line Mode** 0_B LTC63IN is operating in Edge Sensitive Mode. 1_B LTC63IN is operating in Level Sensitive Mode.
CEN	10	rh	**Enable for Shadow Register Copy** 0_B Shadow register copy is currently disabled. 1_B Shadow register copy is currently enabled.
OUT	15	rh	**Output State** 0_B LTC63OUT output line is 0. 1_B LTC63OUT output line is 1.
0	[7:6], 9, [14:11], [31:16]	r	**Reserved** Read as 0; should be written with 0.

GPTA0_LTCXRk (k = 00-62)
GPTA0 Local Timer Cell X Register k(204_H+k*8_H) **Reset Value: 0000 0000_H**
GPTA1_LTCXRk (k = 00-62)
GPTA1 Local Timer Cell X Register k(204_H+k*8_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
X	[15:0]	rwh	**Local Timer Data Register Value**
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

GPTA0_LTCXR63
GPTA0 Local Timer Cell X Register 63(3FC$_H$) **Reset Value: 0000 0000$_H$**
GPTA1_LTCXR63
GPTA1 Local Timer Cell X Register 63(3FC$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
X	[15:0]	rwh	**Compare Register Value** Software write operations has priority above a simultaneous hardware update.
XS	[31:16]	rw	**Shadow Register Value**

22.4.10 Multiplexer Control Registers

These registers are not directly accessible and can be written and read only via the multiplexer register array FIFO (see **Page 22-121**).

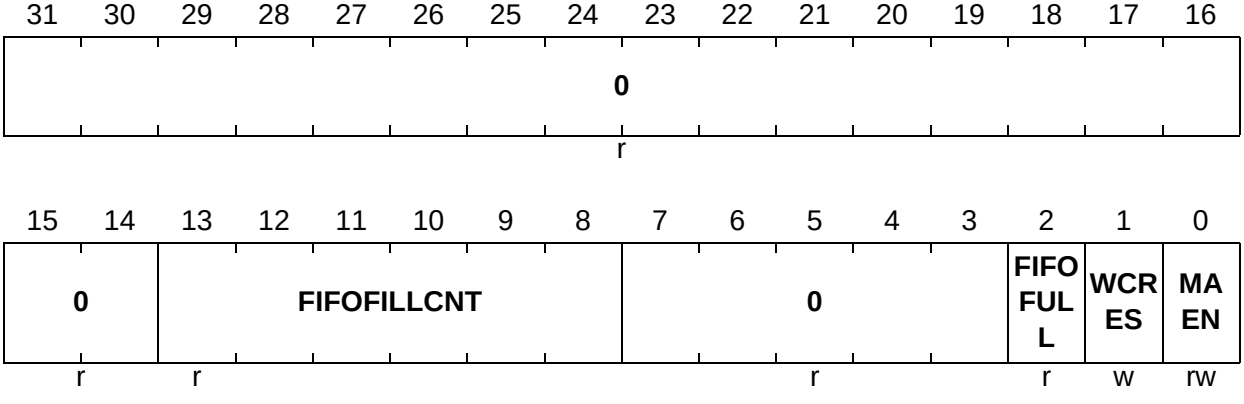
I/O Sharing Block Registers

The three registers MRACTL, MRADIN, and MRADOUT are used to write data to and read data from the GTCA Multiplexer Register Array FIFO. The Multiplexer Register Array FIFO controls the operation of the Input/Output Line Sharing Block (see **"Input/Output Line Sharing Block (IOLS)" on Page 22-235**).

The Multiplexer Register Array Control register controls the operation of the Multiplexer Register Array FIFO.

GPTA0_MRACTL
GPTA0 Multiplexer Register Array Control Register
(038_H) **Reset Value: 0000 0000$_H$**

GPTA1_MRACTL
GPTA1 Multiplexer Register Array Control Register
(038_H) **Reset Value: 0000 0000$_H$**



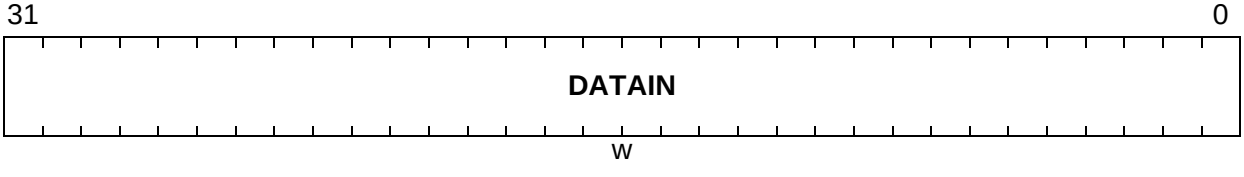
Field	Bits	Type	Description
MAEN	0	rw	**Multiplexer Array Enable** Bit field MAEN enables/disables the programming and the interconnections of the multiplexer array. 0_B Multiplexer array is disabled; all cell inputs are driven with 0, GPTA®v5 I/O lines (pins) are disconnected and FIFO writing is enabled. 1_B Multiplexer array is enabled; all cell and I/O line interconnections are established as previously programmed and FIFO writing is disabled.

Field	Bits	Type	Description
WCRES	1	w	**Write Count Reset** Writing WCRES with 1 while the array is disabled (MAEN = 0), resets the write cycle counter to zero and the FIFO written sequentially (initialized). WCRES is always read as 0.
FIFOFULL	2	r	**FIFO Full Status** 0_B FIFO not completely written (write access to MRADIN allowed). 1_B FIFO completely written (write access to MRADIN ignored). Must be re-enabled via WCRES before array can be re-initialized.
FIFOFILLCNT	[13:8]	r	**FIFO Fill Count** This bit field shows the current contents of the write cycle counter.
0	[7:3], [31:14]	r	**Reserved** Read as 0; should be written with 0.

The Multiplexer Register Array Data In register is used to **write** data to the Multiplexer Register Array FIFO. The Multiplexer Register Array Data Out register is used to **read** data from the Multiplexer Register Array FIFO.

GPTA0_MRADIN
GPTA0 Multiplexer Register Array Data In Register
($03C_H$) **Reset Value: 0000 0000$_H$**

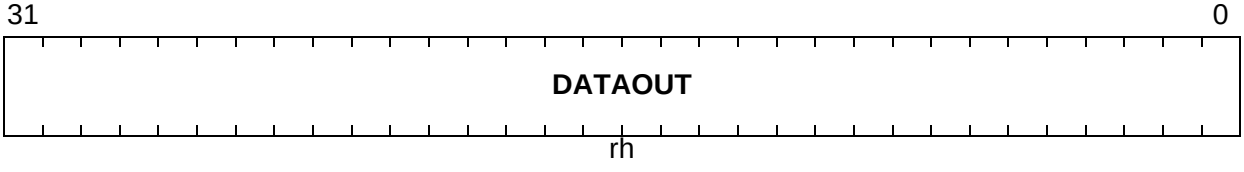
GPTA1_MRADIN
GPTA1 Multiplexer Register Array Data In Register
($03C_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
DATAIN	[31:0]	w	**FIFO Write Data** This register contains the FIFO write data as defined for the Output Multiplexer Control Registers and the Input Multiplexer Control Registers.

GPTA0_MRADOUT
GPTA0 Multiplexer Register Array Data Out Register
(040_H) **Reset Value: 0000 0000_H**

GPTA1_MRADOUT
GPTA1 Multiplexer Register Array Data Out Register
(040_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
DATAOUT	[31:0]	rh	**FIFO Read Data** This register contains the FIFO read data as assigned for the Output Multiplexer Control Registers and the Input Multiplexer Control Registers.

Note: For correct operation, the MRADIN and MRADIN registers must be always read or written 32-bit wide. 8-bit and 16-bit accesses are ignored without any bus error!

Output Multiplexer Control Registers

Two registers, OMCRL and OMCRH, are assigned to each I/O Group IOG[6:0] and each Output Group OG[6:0]. OMCRL[6:0]/OMCRH[6:0] are assigned to IOG[6:0] and OMCRL[13:7]/OMCRH[13:7] are assigned to OG[6:0].

OMCRL controls the connections of group pins 0 to 3. OMCRH controls the connections of group pins 4 to 7.

GPTA0_OMCRLg (g = 0-13)
GPTA0 Output Multiplexer Control Register for Lower Half of Group g
Reset Value: 0000 0000$_H$

GPTA1_OMCRLg (g = 0-13)
GPTA1 Output Multiplexer Control Register for Lower Half of Group g
Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**OMG3**			**0**	**OML3**			**0**	**OMG2**			**0**	**OML2**		
r	rw			r	rw			r	rw			r	rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**OMG1**			**0**	**OML1**			**0**	**OMG0**			**0**	**OML0**		
r	rw			r	rw			r	rw			r	rw		

Field	Bits	Type	Description
OML0, OML1, OML2, OML3	[2:0], [10:8], [18:16], [26:24]	rw	**Multiplexer Line Selection** This bit field selects the input line of a OMG that can be selected by bit field OMGn for OMG output n. 000_B OMG input IN0 selected 001_B OMG input IN1 selected 010_B OMG input IN2 selected 011_B OMG input IN3 selected 100_B OMG input IN4 selected 101_B OMG input IN5 selected 110_B OMG input IN6 selected 111_B OMG input IN7 selected
OMG0, OMG1, OMG2, OMG3	[6:4], [14:12], [22:20], [30:28]	rw	**Multiplexer Group Selection** This bit field determines the OMGng which is connected to input n of I/O Group g or Output group g-7. $X00_B$ OMG0g selected $X01_B$ OMG1g selected $X10_B$ OMG2g selected All other combinations are reserved. If a reserved combination of OMGn value is selected, the corresponding OMG output is forced to 0 level. For compatibility reasons, OMGn[2] = 0 should be used (as value for X) for OMGn bit field programming.
0	3, 7, 11, 15, 19, 23, 27, 31	r	**Reserved** Read as 0; should be written with 0.

GPTA0_OMCRHg (g = 0-13)
Output Multiplexer Control Register for Upper Half of Pin Group g

Reset Value: 0000 0000$_H$

GPTA1_OMCRHg (g = 0-13)
Output Multiplexer Control Register for Upper Half of Pin Group g

Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**OMG7**			**0**	**OML7**			**0**	**OMG6**			**0**	**OML6**		
r	rw			r	rw			r	rw			r	rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**OMG5**			**0**	**OML5**			**0**	**OMG4**			**0**	**OML4**		
r	rw			r	rw			r	rw			r	rw		

Field	Bits	Type	Description
OML4, OML5, OML6, OML7	[2:0], [10:8], [18:16], [26:24]	rw	**Multiplexer Line Selection** This bit field selects the input line of a OMG that can be selected by bit field OMGn for OMG output n. 000_B OMG input IN0 selected 001_B OMG input IN1 selected 010_B OMG input IN2 selected 011_B OMG input IN3 selected 100_B OMG input IN4 selected 101_B OMG input IN5 selected 110_B OMG input IN6 selected 111_B OMG input IN7 selected
OMG4, OMG5, OMG6, OMG7	[6:4], [14:12], [22:20], [30:28]	rw	**Multiplexer Group Selection** This bit field determines the OMGng which is connected to input n of I/O Group g or Output group g-7. $X00_B$ OMG0g selected $X01_B$ OMG1g selected $X10_B$ OMG2g selected All other combinations are reserved. If a reserved combination of OMGn value is selected, the corresponding OMG output is forced to 0 level. For compatibility reasons, OMGn[2] = 0 should be used (as value for X) for OMGn bit field programming.

Field	Bits	Type	Description
0	3, 7, 11, 15, 19, 23, 27, 31	r	**Reserved** Read as 0; should be written with 0.

On-Chip Trigger and Gating Signal Multiplexer Control Registers

OTMCR controls the connections of I/O output group signals to the Trigger and Gating Signals TRIGgn

GPTA0_OTMCRg (g = 0-1)
GPTA0 On-Chip Trigger and Gating Multiplexer Control Register of Group g
Reset Value: 0000 0000$_H$

GPTA1_OTMCRg (g = 0-1)
GPTA1 On-Chip Trigger and Gating Multiplexer Control Register of Group g
Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	OTM7			0	OTM6			0	OTM5			0	OTM4		
r	rw			r	rw			r	rw			r	rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	OTM3			0	OTM2			0	OTM1			0	OTM0		
r	rw			r	rw			r	rw			r	rw		

Field	Bits	Type	Description
OTMn, (n=0...7)	[4 x (n + 4):4 x n]	rw	**Multiplexer Line Selection** This bit field selects the input line of a OMG that can be selected by bit field OMGn for OMG output n. 000_B OTMG input IN0 selected 001_B OTMG input IN1 selected 010_B OTMG input IN2 selected 011_B OTMG input IN3 selected 100_B OTMG input IN4 selected 101_B OTMG input IN5 selected 110_B OTMG input IN6 selected 111_B OTMG input IN7 selected
0	3, 7, 11, 15, 19, 23, 27, 31	r	**Reserved** Read as 0; should be written with 0.

GTC Input Multiplexer Control Registers

Two registers, GIMCRL and GIMCRH, are assigned to each GTCG[3:0]. GIMCRL controls the connections of cells 0 to 3 in a GTC Group. GIMCRH controls the connections of cells 4 to 7 in a GTC Group.

*Note: These registers are not directly accessible and can be written and read only via the multiplexer register array FIFO (see **Section 22.3.4.6**).*

GPTA0_GIMCRLg (g = 0-3)
GPTA0 Input Multiplexer Control Register for Lower Half of GTC Group g
Reset Value: 0000 0000$_H$

GPTA1_GIMCRLg (g = 0-3)
GPTA1 Input Multiplexer Control Register for Lower Half of GTC Group g
Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
GIMEN3		GIMG3		0		GIML3		GIMEN2		GIMG2		0		GIML2	
r		rw		r		rw		r		rw		r		rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GIMEN1		GIMG1		0		GIML1		GIMEN0		GIMG0		0		GIML0	
rw		rw		r		rw		rw		rw		r		rw	

Field	Bits	Type	Description
GIML0, GIML1, GIML2, GIML3	[2:0], [10:8], [18:16], [26:24]	rw	**Multiplexer Line Selection** This bit field selects the input line of a GIMG that can be selected by bit field GIMGn for GIMG output n. 000_B GIMG input IN0 selected 001_B GIMG input IN1 selected 010_B GIMG input IN2 selected 011_B GIMG input IN3 selected 100_B GIMG input IN4 selected 101_B GIMG input IN5 selected 110_B GIMG input IN6 selected 111_B GIMG input IN7 selected

Field	Bits	Type	Description
GIMG0, GIMG1, GIMG2, GIMG3	[6:4], [14:12], [22:20], [30:28]	rw	**Multiplexer Group Selection** This bit field determines the GIMGng which is connected to input n of GTC group g. 000_B GIMG0g selected 001_B GIMG1g selected (reserved for g = 3) 010_B GIMG2g selected 011_B GIMG3g selected 100_B GIMG4g selected All other combinations are reserved.
GIMEN0, GIMEN1, GIMEN2, GIMEN3	7, 15, 23, 31	rw	**Enable Multiplexer Connection** 0_B Input n is not connected to any line. 1_B Input n is connected to the line defined by GIMLn and GIMGn.
0	3, 11, 19, 27	r	**Reserved** Read as 0; should be written with 0.

GPTA0_GIMCRHg (g = 0-3)
GPTA0 Input Multiplexer Control Register for Upper Half of GTC Group g
Reset Value: 0000 0000$_H$

GPTA1_GIMCRHg (g = 0-3)
GPTA1 Input Multiplexer Control Register for Upper Half of GTC Group g
Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
GIMEN7		GIMG7		0		GIML7		GIMEN6		GIMG6		0		GIML6	
r		rw		r		rw		r		rw		r		rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GIMEN5		GIMG5		0		GIML5		GIMEN4		GIMG4		0		GIML4	
r		rw		r		rw		r		rw		r		rw	

Field	Bits	Type	Description
GIML4, GIML5, GIML6, GIML7	[2:0], [10:8], [18:16], [26:24]	rw	**Multiplexer Line Selection** This bit field selects the input line of a GIMG that can be selected by bit field GIMGn for GIMG output n. 000_B GIMG input IN0 selected 001_B GIMG input IN1 selected 010_B GIMG input IN2 selected 011_B GIMG input IN3 selected 100_B GIMG input IN4 selected 101_B GIMG input IN5 selected 110_B GIMG input IN6 selected 111_B GIMG input IN7 selected
GIMG4, GIMG5, GIMG6, GIMG7	[6:4], [14:12], [22:20], [30:28]	rw	**Multiplexer Group Selection** This bit field determines the GIMGng which is connected to input n of GTC group g. 000_B GIMG0g selected 001_B GIMG1g selected (reserved for g = 3) 010_B GIMG2g selected 011_B GIMG3g selected 100_B GIMG4g selected All other combinations are reserved.

Field	Bits	Type	Description
GIMEN4, GIMEN5, GIMEN6, GIMEN7	7, 15, 23, 31	rw	**Enable Multiplexer Connection** 0_B Input n is not connected to any line. 1_B Input n is connected to the line defined by GIMLn and GIMGn.
0	3, 11, 19, 27	r	**Reserved** Read as 0; should be written with 0.

LTC Input Multiplexer Control Registers

Two registers, LIMCRL and LIMCRH, are assigned to each LTC group. LIMCRL controls the connections of LTC group cells with index 0 to 3. LIMCRH controls the connections of LTC group cells with index 4 to 7.

*Note: These registers are not directly accessible and can be written and read only via the multiplexer register array FIFO (see **Section 22.3.4.6**).*

GPTA0_LIMCRLg (g = 0-7)
GPTA0 Input Multiplexer Control Register for Lower Half of LTC Group g
Reset Value: 0000 0000$_H$

GPTA1_LIMCRLg (g = 0-7)
GPTA1 Input Multiplexer Control Register for Lower Half of LTC Group g
Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LIMEN3		LIMG3		0		LIML3		LIMEN2		LIMG2		0		LIML2	
r		rw		r		rw		r		rw		r		rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LIMEN1		LIMG1		0		LIML1		LIMEN0		LIMG0		0		LIML0	
r		rw		r		rw		r		rw		r		rw	

Field	Bits	Type	Description
LIML0, LIML1, LIML2, LIML3	[2:0], [10:8], [18:16], [26:24]	rw	**Multiplexer Line Selection** This bit field selects the input line of a LIMG that can be selected by bit field LIMGn for LIMG output n. 000_B LIMG input IN0 selected 001_B LIMG input IN1 selected 010_B LIMG input IN2 selected 011_B LIMG input IN3 selected 100_B LIMG input IN4 selected 101_B LIMG input IN5 selected 110_B LIMG input IN6 selected 111_B LIMG input IN7 selected

Field	Bits	Type	Description
LIMG0, LIMG1, LIMG2, LIMG3	[6:4], [14:12], [22:20], [30:28]	rw	**Multiplexer Group Selection** This bit field determines the LIMGng which is connected to input n of LTC group g. 000_B LIMG0g selected 001_B LIMG1g selected (reserved for g = 3) 010_B LIMG2g selected 011_B LIMG3g selected 100_B LIMG4g selected All other combinations are reserved.
LIMEN0, LIMEN1, LIMEN2, LIMEN3	7, 15, 23, 31	rw	**Enable Multiplexer Connection** 0_B Input n is not connected to any line. 1_B Input n is connected to the line defined by LIMLn and LIMGn.
0	3, 11, 19, 27	r	**Reserved** Read as 0; should be written with 0.

GPTA0_LIMCRHg (g = 0-7)
GPTA0 Input Multiplexer Control Register for Upper Half of LTC Group g
Reset Value: 0000 0000$_H$

GPTA1_LIMCRHg (g = 0-7)
GPTA1 Input Multiplexer Control Register for Upper Half of LTC Group g
Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LIM EN7		**LIMG7**		**0**		**LIML7**		**LIM EN6**		**LIMG6**		**0**		**LIML6**	
r		rw		r		rw		r		rw		r		rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LIM EN5		**LIMG5**		**0**		**LIML5**		**LIM EN4**		**LIMG4**		**0**		**LIML4**	
r		rw		r		rw		r		rw		r		rw	

Field	Bits	Type	Description
LIML4, LIML5, LIML6, LIML7	[2:0], [10:8], [18:16], [26:24]	rw	**Multiplexer Line Selection** This bit field selects the input line of a LIMG that can be selected by bit field LIMGn for LIMG output n. 000_B LIMG input IN0 selected 001_B LIMG input IN1 selected 010_B LIMG input IN2 selected 011_B LIMG input IN3 selected 100_B LIMG input IN4 selected 101_B LIMG input IN5 selected 110_B LIMG input IN6 selected 111_B LIMG input IN7 selected
LIMG4, LIMG5, LIMG6, LIMG7	[6:4], [14:12], [22:20], [30:28]	rw	**Multiplexer Group Selection** This bit field determines the LIMGng which is connected to input n of LTC group g. 000_B LIMG0g selected 001_B LIMG1g selected (reserved for g = 3) 010_B LIMG2g selected 011_B LIMG3g selected 100_B LIMG4g selected All other combinations are reserved.

Field	Bits	Type	Description
LIMEN4, LIMEN5, LIMEN6, LIMEN7	7, 15, 23, 31	rw	**Enable Multiplexer Connection** 0_B Input n is not connected to any line. 1_B Input n is connected to the line defined by LIMLn and LIMGn.
0	3, 11, 19, 27	r	**Reserved** Read as 0; should be written with 0.

22.4.11 Service Request Registers

The bits in the Service Request State Registers are service request status flags that are set by hardware (type “h”) when the related event occurs, regardless if a respective Interrupt Request is enabled. Each service request status flag can be read twice (in SRSCx register and in SRSSx register, x = 0-3) and cleared or set by software when writing to the specific request bit in SRSCx or SRSSx. If enabled, a interrupt request is generated regardless of the content of the SRSSx or SRSCx registers.

The service request status flags can be reset (cleared) by software when writing a 1 to the corresponding bit location in the SRSCx registers. Writing a 0 has no effect.

The service request status flags can be set by software when writing a 1 to the corresponding bit location in the SRSSx registers. Writing a 0 has no effect.

GPTA0_SRSC0
GPTA0 Service Request State Clear Register 0
(010_H) **Reset Value: 0000 0000_H**

GPTA1_SRSC0
GPTA1 Service Request State Clear Register 0
(010_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	GT01	GT00	PLL	DCM03C	DCM03F	DCM03R	DCM02C	DCM02F	DCM02R	DCM01C	DCM01F	DCM01R	DCM00C	DCM00F	DCM00R
r	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
DCM00R, **DCM01R,** **DCM02R,** **DCM03R**	0, 3, 6, 9	rwh[1)]	**DCMk[2)] Rising Edge Event Service Request State** 0_B No service is requested. 1_B Service is requested due to a rising edge detected on the DCMk input signal line.
DCM00F, **DCM01F,** **DCM02F,** **DCM03F**	1, 4, 7, 10	rwh[1)]	**DCMk[2)] Falling Edge Event Service Request State** 0_B No service is requested. 1_B Service is requested due to a falling edge detected on the DCMk input signal line.

Field	Bits	Type	Description
DCM00C, DCM01C, DCM02C, DCM03C	2, 5, 8, 11	rwh[1]	**DCMk[2] Compare Event Service Request State** 0_B No service is requested. 1_B Service is requested due to a compare event occurred in DCMk cell (k = 0-3).
PLL	12	rwh[1]	**Counter Service Request State for PLL** 0_B No service is requested. 1_B Service is requested because the counter for the number remaining output pulses decremented to 0.
GT00	13	rwh[1]	**GT0 Timer Service Request State** 0_B No service is requested. 1_B Service is requested due to a GT0 timer overflow.
GT01	14	rwh[1]	**GT1 Timer Service Request State** 0_B No service is requested. 1_B Service is requested due to a GT1 timer overflow.
0	[31:15]	r	**Reserved** Read as 0; should be written with 0.

1) Writing an one to a set bit clears the bit. All other write operations have no effect.

2) k = 0-3; k = 0 refers to DCM00R, DCM00F, or DCM00C; k = 1 refers to DCM01R, DCM01F, or DCM01C; k = 2 refers to DCM02R, DCM02F, or DCM02C; k = 3 refers to DCM03R, DCM03F, or DCM03C.

GPTA0_SRSS0
GPTA0 Service Request State Set Register 0
(014_H) **Reset Value: 0000 0000$_H$**

GPTA1_SRSS0
GPTA1 Service Request State Set Register 0
(014_H) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**GT 01**	**GT 00**	**PLL**	**DCM 03C**	**DCM 03F**	**DCM 03R**	**DCM 02C**	**DCM 02F**	**DCM 02R**	**DCM 01C**	**DCM 01F**	**DCM 01R**	**DCM 00C**	**DCM 00F**	**DCM 00R**
r	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
DCM00R, DCM01R, DCM02R, DCM03R	0, 3, 6, 9	rwh[1)]	**DCMk[2)] Rising Edge Event Service Request State** 0_B No service is requested. 1_B Service is requested due to a rising edge detected on DCMk input signal line.
DCM00F, DCM01F, DCM02F, DCM03F	1, 4, 7, 10	rwh[1)]	**DCMk[2)] Falling Edge Event Service Request State** 0_B No service is requested. 1_B Service is requested due to a falling edge detected on DCMk input signal line.
DCM00C, DCM01C, DCM02C, DCM03C	2, 5, 8, 11	rwh[1)]	**DCMk[2)] Compare Event Service Request State** 0_B No service is requested. 1_B Service is requested due to a compare event occurred in DCMk cell.
PLL	12	rwh[1)]	**Counter Service Request State for PLL** 0_B No service is requested 1_B Service is requested because the counter for the number remaining output pulses decremented to 0.
GT00	13	rwh[1)]	**GT0 Timer Service Request State** 0_B No service is requested. 1_B Service is requested due to a GT0 timer overflow.

Field	Bits	Type	Description
GT01	14	rwh[1)]	**GT1 Timer Service Request State** 0_B No service is requested. 1_B Service is requested due to a GT1 timer overflow.
0	[31:15]	r	**Reserved** Read as 0; should be written with 0.

1) Writing a one to a cleared bit sets the bit. All other write operations have no effect.

2) k = 0-3; k = 0 refers to DCM00R, DCM00F, or DCM00C; k = 1 refers to DCM01R, DCM01F, or DCM01C; k = 2 refers to DCM02R, DCM02F, or DCM02C; k = 3 refers to DCM03R, DCM03F, or DCM03C.

GPTA0_SRSC1
GPTA0 Service Request State Clear Register 1
(018_H) **Reset Value: 0000 0000_H**

GPTA1_SRSC1
GPTA1 Service Request State Clear Register 1
(018_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
GTC 31	**GTC 30**	**GTC 29**	**GTC 28**	**GTC 27**	**GTC 26**	**GTC 25**	**GTC 24**	**GTC 23**	**GTC 22**	**GTC 21**	**GTC 20**	**GTC 19**	**GTC 18**	**GTC 17**	**GTC 16**
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GTC 15	**GTC 14**	**GTC 13**	**GTC 12**	**GTC 11**	**GTC 10**	**GTC 09**	**GTC 08**	**GTC 07**	**GTC 06**	**GTC 05**	**GTC 04**	**GTC 03**	**GTC 02**	**GTC 01**	**GTC 00**
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
GTCk (k = 00-31)	k	rwh[1)]	**GTCk Capture/Compare Service Request State** 0_B No service is requested. 1_B Service is requested due to a capture or compare event occurred in GTCk.

1) Writing an one to a set bit clears the bit. All other write operations have no effect.

GPTA0_SRSS1
GPTA0 Service Request State Set Register 1
($01C_H$) **Reset Value: 0000 0000_H**

GPTA1_SRSS1
GPTA1 Service Request State Set Register 1
($01C_H$) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
GTC 31	**GTC 30**	**GTC 29**	**GTC 28**	**GTC 27**	**GTC 26**	**GTC 25**	**GTC 24**	**GTC 23**	**GTC 22**	**GTC 21**	**GTC 20**	**GTC 19**	**GTC 18**	**GTC 17**	**GTC 16**
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GTC 15	**GTC 14**	**GTC 13**	**GTC 12**	**GTC 11**	**GTC 10**	**GTC 09**	**GTC 08**	**GTC 07**	**GTC 06**	**GTC 05**	**GTC 04**	**GTC 03**	**GTC 02**	**GTC 01**	**GTC 00**
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
GTCk (k = 00-31)	k	rwh[1)]	**GTCk Capture/Compare Service Request State** 0_B No service is requested. 1_B Service is requested due to a capture or compare event occurred in GTCk.

1) Writing a one to a cleared bit sets the bit. All other write operations have no effect.

GPTA0_SRSC2
GPTA0 Service Request State Clear Register 2
(020_H) **Reset Value: 0000 0000$_H$**

GPTA1_SRSC2
GPTA1 Service Request State Clear Register 2
(020_H) **Reset Value: 0000 0000$_H$**

LTCA2_SRSC2
LTCA2 Service Request State Clear Register 2
(020_H) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LTC 31	**LTC 30**	**LTC 29**	**LTC 28**	**LTC 27**	**LTC 26**	**LTC 25**	**LTC 24**	**LTC 23**	**LTC 22**	**LTC 21**	**LTC 20**	**LTC 19**	**LTC 18**	**LTC 17**	**LTC 16**
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LTC 15	**LTC 14**	**LTC 13**	**LTC 12**	**LTC 11**	**LTC 10**	**LTC 09**	**LTC 08**	**LTC 07**	**LTC 06**	**LTC 05**	**LTC 04**	**LTC 03**	**LTC 02**	**LTC 01**	**LTC 00**
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
LTCk (k = 00-31)	k	rwh[1)]	**LTCk Timer/Capture/Compare Service Request State** 0_B No service is requested. 1_B Service is requested due to a timer overflow, capture, or compare event that occurred in LTCk.

1) Writing an one to a set bit clears the bit. All other write operations have no effect.

GPTA0_SRSS2
GPTA0 Service Request State Set Register 2
(024_H) **Reset Value: 0000 0000$_H$**

GPTA1_SRSS2
GPTA1 Service Request State Set Register 2
(024_H) **Reset Value: 0000 0000$_H$**

LTCA2_SRSS2
LTCA2 Service Request State Set Register 2
(024_H) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LTC 31	**LTC 30**	**LTC 29**	**LTC 28**	**LTC 27**	**LTC 26**	**LTC 25**	**LTC 24**	**LTC 23**	**LTC 22**	**LTC 21**	**LTC 20**	**LTC 19**	**LTC 18**	**LTC 17**	**LTC 16**
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LTC 15	**LTC 14**	**LTC 13**	**LTC 12**	**LTC 11**	**LTC 10**	**LTC 09**	**LTC 08**	**LTC 07**	**LTC 06**	**LTC 05**	**LTC 04**	**LTC 03**	**LTC 02**	**LTC 01**	**LTC 00**
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
LTCk (k = 00-31)	k	rwh[1)]	**LTCk Timer/Capture/Compare Service Request State** 0_B No service is requested. 1_B Service is requested due to a timer overflow, capture, or compare event that occurred in LTCk.

1) Writing a one to a cleared bit sets the bit. All other write operations have no effect.

GPTA0_SRSC3
GPTA0 Service Request State Clear Register 3
(028_H) **Reset Value: 0000 0000$_H$**

GPTA1_SRSC3
GPTA1 Service Request State Clear Register 3
(028_H) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LTC 63	**LTC 62**	**LTC 61**	**LTC 60**	**LTC 59**	**LTC 58**	**LTC 57**	**LTC 56**	**LTC 55**	**LTC 54**	**LTC 53**	**LTC 52**	**LTC 51**	**LTC 50**	**LTC 49**	**LTC 48**
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LTC 47	**LTC 46**	**LTC 45**	**LTC 44**	**LTC 43**	**LTC 42**	**LTC 41**	**LTC 40**	**LTC 39**	**LTC 38**	**LTC 37**	**LTC 36**	**LTC 35**	**LTC 34**	**LTC 33**	**LTC 32**
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
LTCk (k = 32-63)	k-32	rwh[1)]	**LTCk Timer/Capture/Compare Service Request State** 0_B No service is requested. 1_B Service is requested due to a timer overflow, capture, or compare event that occurred in LTCk.

1) Writing an one to a set bit clears the bit. All other write operations have no effect.

GPTA0_SRSS3
GPTA0 Service Request State Set Register 3
($02C_H$) **Reset Value: 0000 0000$_H$**

GPTA1_SRSS3
GPTA1 Service Request State Set Register 3
($02C_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LTC 63	**LTC 62**	**LTC 61**	**LTC 60**	**LTC 59**	**LTC 58**	**LTC 57**	**LTC 56**	**LTC 55**	**LTC 54**	**LTC 53**	**LTC 52**	**LTC 51**	**LTC 50**	**LTC 49**	**LTC 48**
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LTC 47	**LTC 46**	**LTC 45**	**LTC 44**	**LTC 43**	**LTC 42**	**LTC 41**	**LTC 40**	**LTC 39**	**LTC 38**	**LTC 37**	**LTC 36**	**LTC 35**	**LTC 34**	**LTC 33**	**LTC 32**
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
LTCk (k = 32-63)	k-32	rwh[1)]	**LTCk Timer/Capture/Compare Service Request State** 0_B No service is requested. 1_B Service is requested due to a timer overflow, capture, or compare event that occurred in LTCk.

1) Writing a one to a cleared bit sets the bit. All other write operations have no effect.

Node Redirection Register

The Service Request Node Redirection Register allows that GTC service requests of GTCs with an odd index number k can be individually redirected via register SRNR to a service request group that is assigned mainly to four LTCs. More details are provided on Page 22-124.

GPTA0_SRNR
GPTA0 Service Request Node Redirection Register
(030_H) **Reset Value: 0000 0000_H**

GPTA1_SRNR
GPTA1 Service Request Node Redirection Register
(030_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GTC 31R	GTC 29R	GTC 27R	GTC 25R	GTC 23R	GTC 21R	GTC 19R	GTC 17R	GTC 15R	GTC 13R	GTC 11R	GTC 09R	GTC 07R	GTC 05R	GTC 03R	GTC 01R
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
GTC01R, **GTC03R,** **GTC05R,** **GTC07R,** **GTC09R,** **GTC11R,** **GTC13R,** **GTC15R,** **GTC17R,** **GTC19R,** **GTC21R,** **GTC23R,** **GTC25R,** **GTC27R,** **GTC29R,** **GTC31R**	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15	rw	**Global Timer Cell k Redirection** 0_B No redirection of GTC service requests. 1_B Redirection of GTC service request to LTC service request groups (see Page 22-124).

Field	Bits	Type	Description
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

22.5 LTCA Kernel Description

The Local Timer Cell Array (LTCA2) is a unit that contains signal generation cells with 32 Local Timer Cells. These LTCs have identical functionality as the LTCs which are available in the signal generation cells of the GPTA®v5 module.

The Local Timer Cells (LTC00 to LTC30) can be configured to operate in different modes: Capture Mode, Compare Mode, Free-Running Timer Mode, Reset Timer Mode, and One Shot Mode. Adjacent cells may be combined to operate on the same pin, thus generating complex waveforms. One LTC (LTC31) can be used for special compare modes.

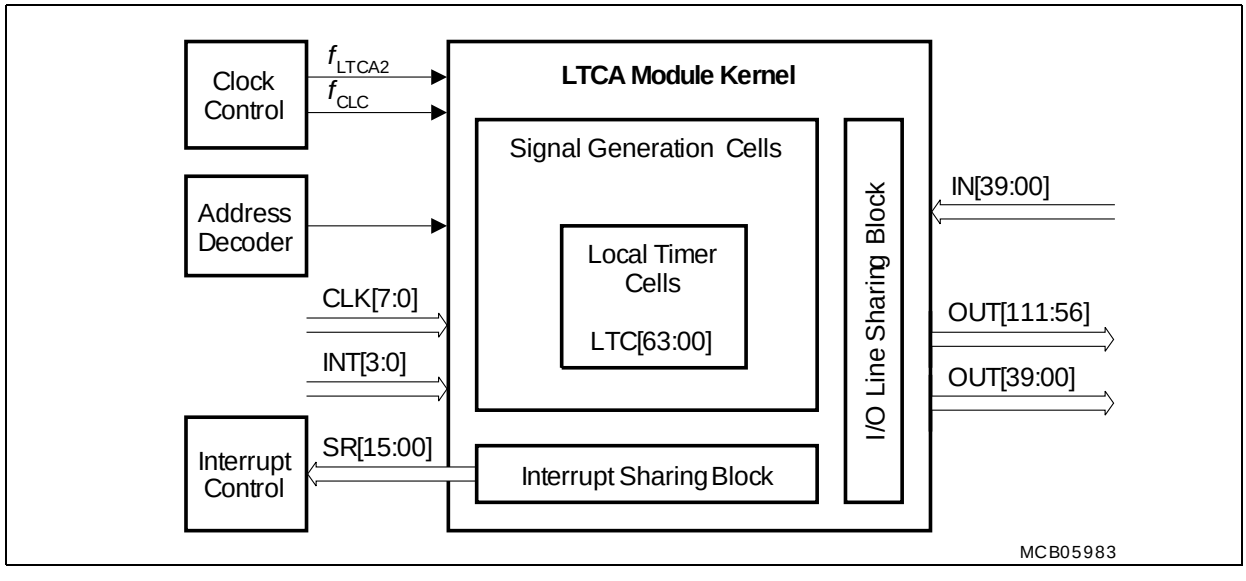


Figure 22-81 Block Diagram of LTCA Unit Kernel

The LTCA Unit Kernel contains Signal Generation Cells that contains all Local Timer Cells including an I/O Line Sharing Block that controls the LTC connections to the I/O lines and output lines. I/O lines are supposed to be connected to I/O port lines while the output lines are typically connected to a MSC interface that is especially able to control external power devices via a serial connection. The LTCs can be further connected to input signals of an external clock bus and input lines coming e.g. from other on-chip peripheral modules.

The clock control cells generates two modules clocks that are required for LTCA operation. An address decoder generates the select signals for the LTC registers. Service requests, coming from the LTCs, are able to generate interrupts via the Interrupt Sharing Block. The interrupts are handled by the external interrupt logic.

22.5.1 Local Timer Cell (LTC00 to LTC31)

LTC00 to LTC30 are fully identical with the Local Timer Cells in the GPTA0 and GPTA1 units. Its functionalities are described in GPTA®v5 section **“Local Timer Cell (LTC00 to LTC62)” on Page 22-67** and LTC31 functionalities are described in GPTA®v5 section **“Local Timer Cell LTC63” on Page 22-79**.

22.5.2 Input/Output Line Sharing Block (IOLS)

The I/O Line Sharing Block allows the inputs and outputs of the LTCA unit to be routed with high flexibility between I/O lines, output lines, clock inputs, and other on-chip peripherals. The LTCA unit provides a total of 32 input lines and 64 output lines, that are connected to four I/O groups IOG[3:0] and four output groups OG[6:3].

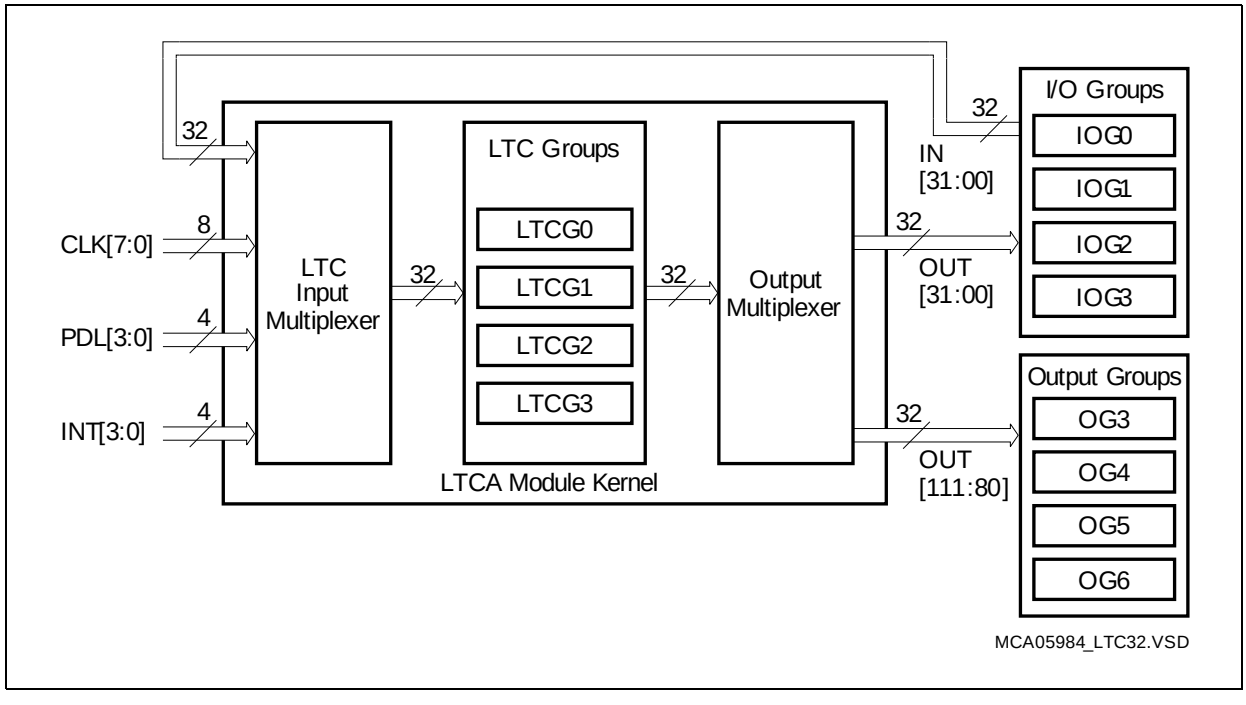


Figure 22-82 Input/Output Line Sharing Block Overview

The LTCA I/O Line Sharing Block makes the following two selections:

- LTC output multiplexer selection
- LTC input multiplexer selection

To choose these selection, the input and output lines of the related cells are integrated into groups with eight parts, each. There are I/O groups, output groups, LTC groups, and a PDL/INT group.

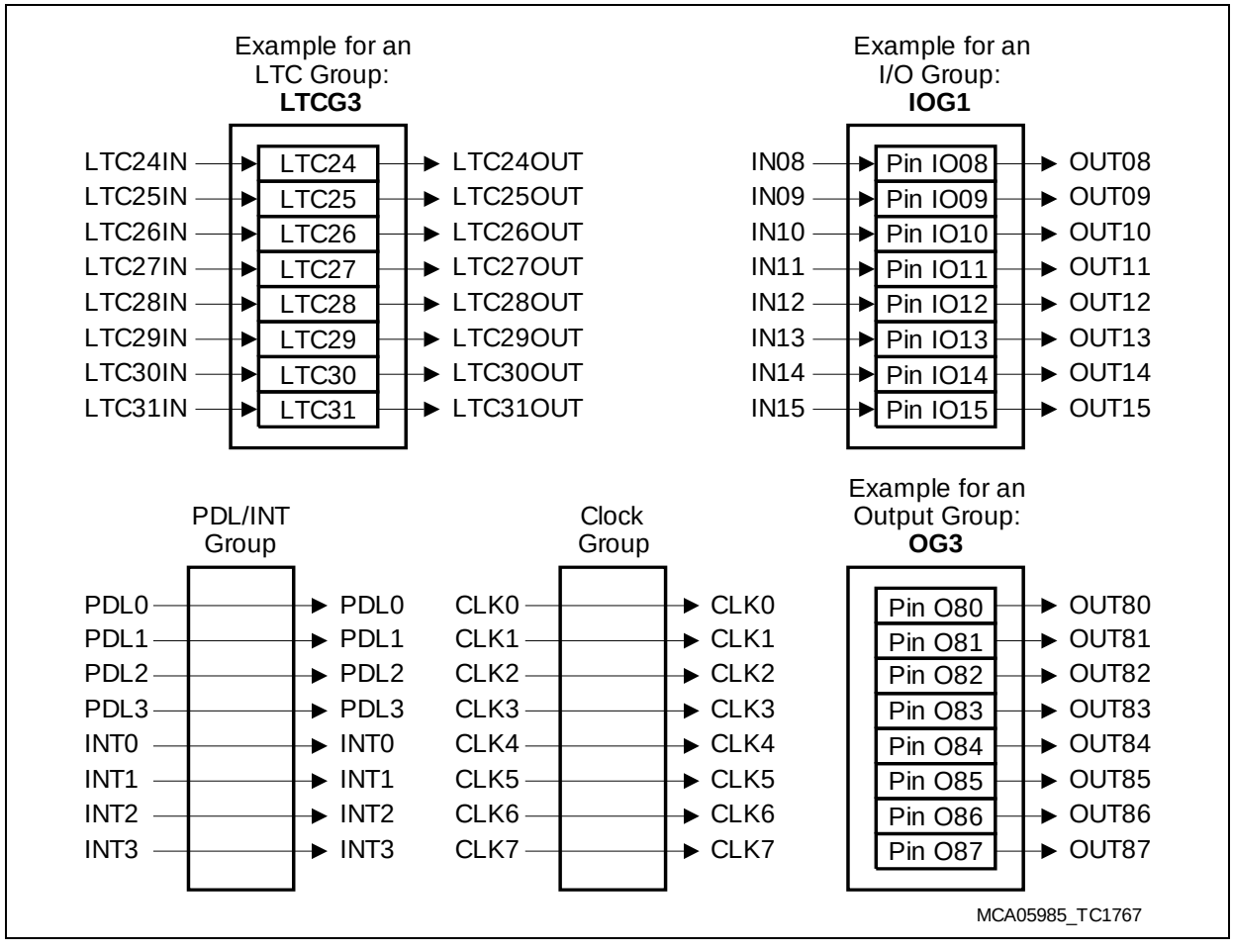


Figure 22-83 Groups Definitions for I/O Line Sharing Block

An **LTC group** combines eight LTC cells with its input and output lines. This results in four LTC groups, LTCG0 to LTCG3.

An **I/O group** combines eight LTCA I/O lines connected to bi-directional device pins with its input and output lines. This results in four I/O groups, IOG0 to IOG3, supporting 32 I/O lines.

An **output group** combines four LTCA output lines connected to device pins as an output. This results in four output groups, OG3 to OG6, supporting 32 output lines.

The **PDL/INT group** is a logical group that combines the LTCA unit inputs PDL[3:0] together with the inputs INT[3:0].

The **clock group** is a logical group that combines the eight LTCA unit clock inputs CLK[7:0].

Table 22-20 Group to I/O Line/Cell Assignment

Group/Unit	Cell/Line	Input	Output
LTC Groups			
LTCG0	LTC[07:00]	LTC[07:00]IN	LTC[07:00]OUT
LTCG1	LTC[15:08]	LTC[15:08]IN	LTC[15:08]OUT
LTCG2	LTC[23:16]	LTC[23:16]IN	LTC[23:16]OUT
LTCG3	LTC[31:24]	LTC[31:24]IN	LTC[31:24]OUT
I/O Groups			
IOG0	–	IN[07:00]	OUT[07:00]
IOG1	–	IN[15:08]	OUT[15:08]
IOG2	–	IN[23:16]	OUT[23:16]
IOG3	–	IN[31:24]	OUT[31:24]
Output Groups			
OG3	–	–	OUT[87:80]
OG4	–	–	OUT[95:88]
OG5	–	–	OUT[103:96]
OG6	–	–	OUT[111:104]
Clock Group			
–	–	CLK[7:0]	CLK[7:0]
PDL/INT Groups			
PDL[1:0] PDL Bus	–	PDL[3:0]	PDL[3:0]
External Input [3:0]	–	INT[3:0]	INT[3:0]

22.5.2.1 Output Multiplexer

The output multiplexer shown in **Figure 22-84** and **Figure 22-86** connects the 32 LTC output lines with the I/O groups ($4 \times 8 = 32$ output lines) and the output groups ($4 \times 8 = 32$ output lines).

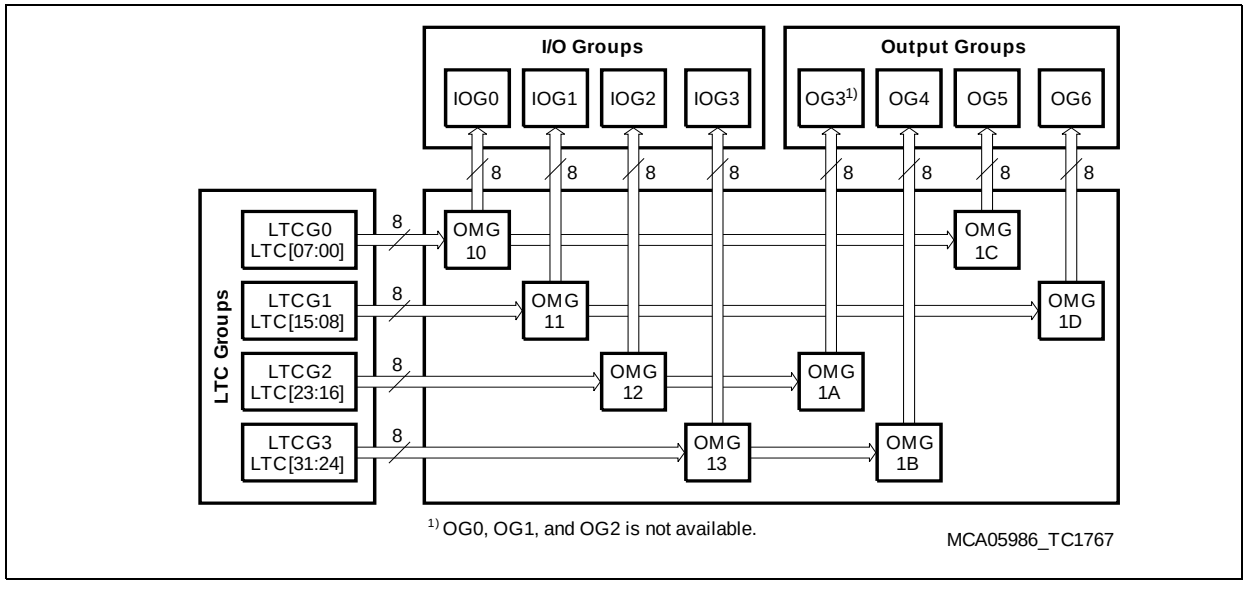


Figure 22-84 Output Multiplexer of LTCA

The output multiplexer contains Output Multiplexer Groups (OMGs) that connect the Local Timer Cells with the input lines of the I/O groups and output groups. The LTCs are grouped into four LTC groups (LTCG[3:0]) with 8 cells each. In the same way, I/O groups and output groups are grouped into 8 groups (four I/O groups and four output groups) with 8 lines each.

Figure 22-85 shows the logical structure of an OMG.

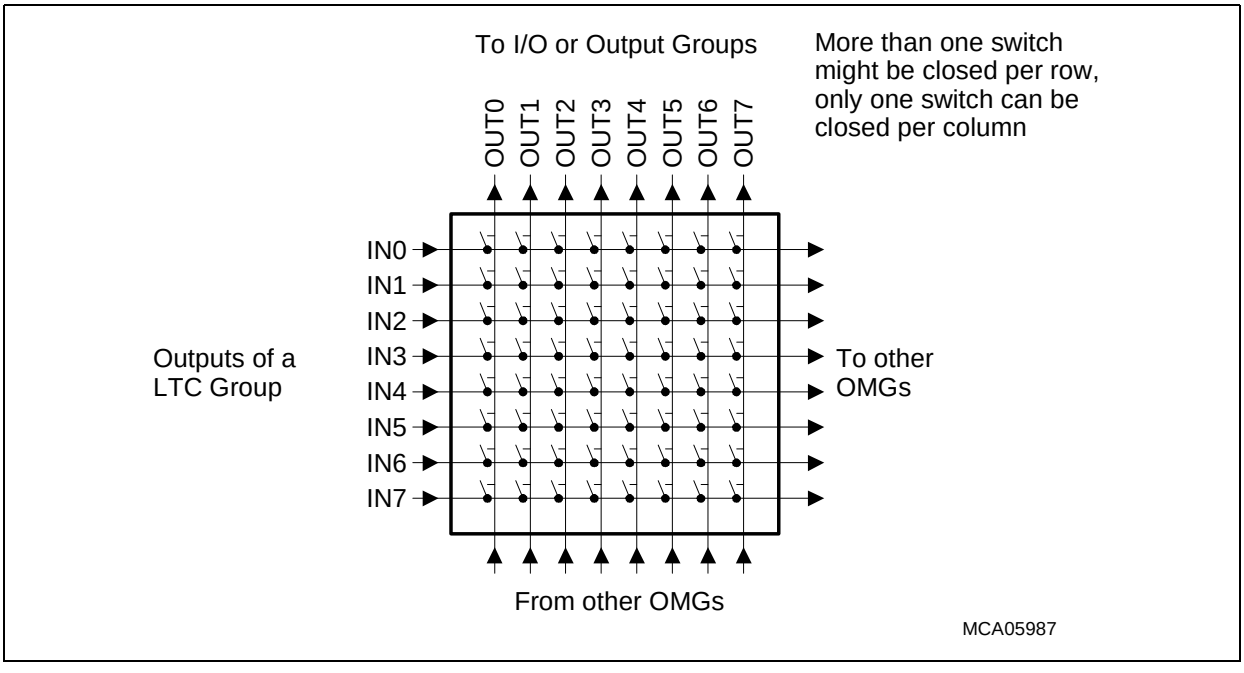


Figure 22-85 Output Multiplexer Group (OMG) Structure

Rules for connections to Output Multiplexer Group OMG:

- OMG output line OUT0 is always connected to the input of an I/O or output group with the lowest index. The remaining output lines OUT1 to OUT7 are connected to the I/O or Output lines with ascending index.
  Example: for OMG13 (see **Figure 22-84**), the outputs OUT0 to OUT7 are wired to input lines 0 to 7 of I/O group 3 (IOG3).
- One input of an I/O or output group can be connected only to the output of one timer cell. This is guaranteed by the OMG control register layout. Otherwise, short circuits and unpredictable behavior would occur. On the other hand, it is permitted that for the output of an LTC cell to be connected to more than one input of an I/O or output group.

The output multiplexer group configuration is based on the following principles:

- Each OMG is referenced with two index variables: n and g (OMGng)
- Index n is a group number. Local Timer Cell Groups LTCG[3:0] have the group number 1.
- Index g indicates the number of an I/O or output group g ($g = 0\text{-}7_D$) to which the outputs of the output multiplexer group OMGng are connected. I/O groups OG0 to OG3 are assigned to index variable g = 0 to 3 and output groups OG3 to OG6 are assigned to index variable g = 10 to 13. Index g = 4, 5, 6, 7, 8, 9 is not available.

The output multiplexer logic as seen for programming is shown in **Figure 22-86**. With this logic, one LTC group signal is always combined to one output line that leads to the

input of an I/O or output group. For example, when looking at **Figure 22-84**, each of the eight output multiplexer output lines to I/O group IOG4 is connected via one OMG04 with the eight outputs of two LTC groups (LTCG0 and LTCG4).

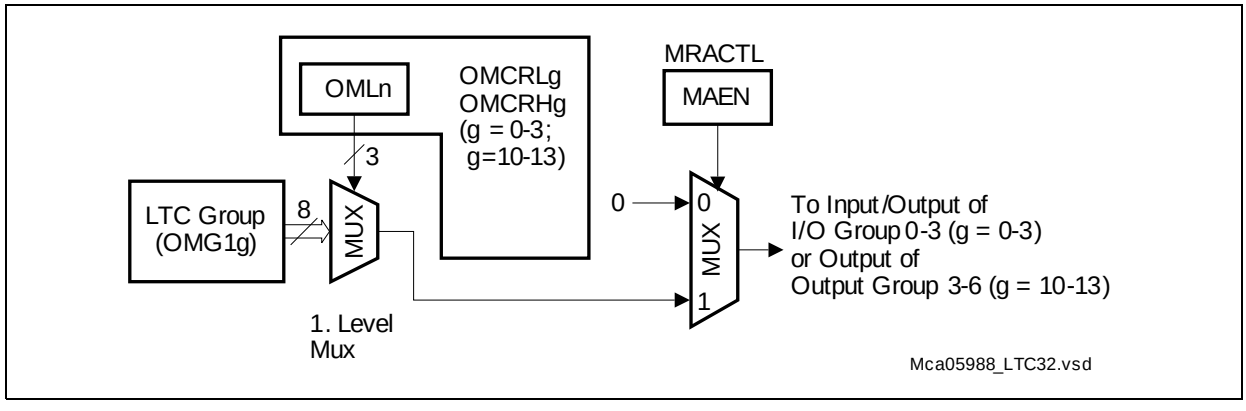


Figure 22-86 OMG Multiplexer (Programmer's View)

The 1. level multiplexer is built up by two 8:1 multiplexers that are controlled in parallel by bit field OMLn. The output of the 21. level multiplexer is only connected only to the input of an I/O group or output group if bit MRACTL.MAEN (multiplexer array enabled) is set. If MRACTL.MAEN = 0, the corresponding OMG output will be held at a low level.

Two Output Multiplexer Control Registers, OMCRL and OMCRH (see also **Page 22-210**), are assigned to each of the I/O or output groups. Therefore, in total 16 registers control the connections within the output multiplexer of the LTCA unit.

The OMCRL registers control the OMG output lines 0 to 3 and the OMCRH registers control the OMG output lines 4 to 7. **Table 22-21** lists all Output Multiplexer Control Registers with its control functions. Please note that all Output Multiplexer Control Registers are not directly accessible but must be written or read using the FIFO array structure as described on **Page 22-210**.

Table 22-21 Output Multiplexer Control Register Assignments

I/O Group or Output Group		Controlled by Multiplexer Control Register	Selectable Groups via OMGng
IOG0	IN[03:00]/OUT[03:00]	OMCRL0	LTCG0
	IN[07:04]/OUT[07:04]	OMCRH0	
IOG1	IN[11:08]/OUT[11:08]	OMCRL1	LTCG1
	IN[15:12]/OUT[15:12]	OMCRH1	
IOG2	IN[19:16]/OUT[19:16]	OMCRL2	LTCG2
	IN[23:20]/OUT[23:20]	OMCRH2	
IOG3	IN[27:24]/OUT[27:24]	OMCRL3	LTCG3
	IN[31:28]/OUT[31:28]	OMCRH3	
OG3	OUT[83:80][1)2)]	OMCRL10	LTCG2
	OUT[87:84]	OMCRH10	
OG4	OUT[91:88]	OMCRL11	LTCG3
	OUT[95:92]	OMCRH11	
OG5	OUT[99:96]	OMCRL12	LTCG0
	OUT[103:100]	OMCRH12	
OG6	OUT[107:104]	OMCRL13	LTCG1
	OUT[111:108]	OMCRH13	

1) OUT[79:32] is not available.

2) OG0 - OG2 is not available.

22.5.2.2 LTC Input Multiplexing Scheme

The LTC input multiplexer as shown in **Figure 22-87** and **Figure 22-89** connects the 32 (= 4 × 8) input lines of the I/O groups, the eight clock bus input lines, or the four PDL input lines PDL[3:0] and the four internal input lines INT[3:0] with the 32 (= 4 × 8) LTC inputs, organized in eight LTC groups.

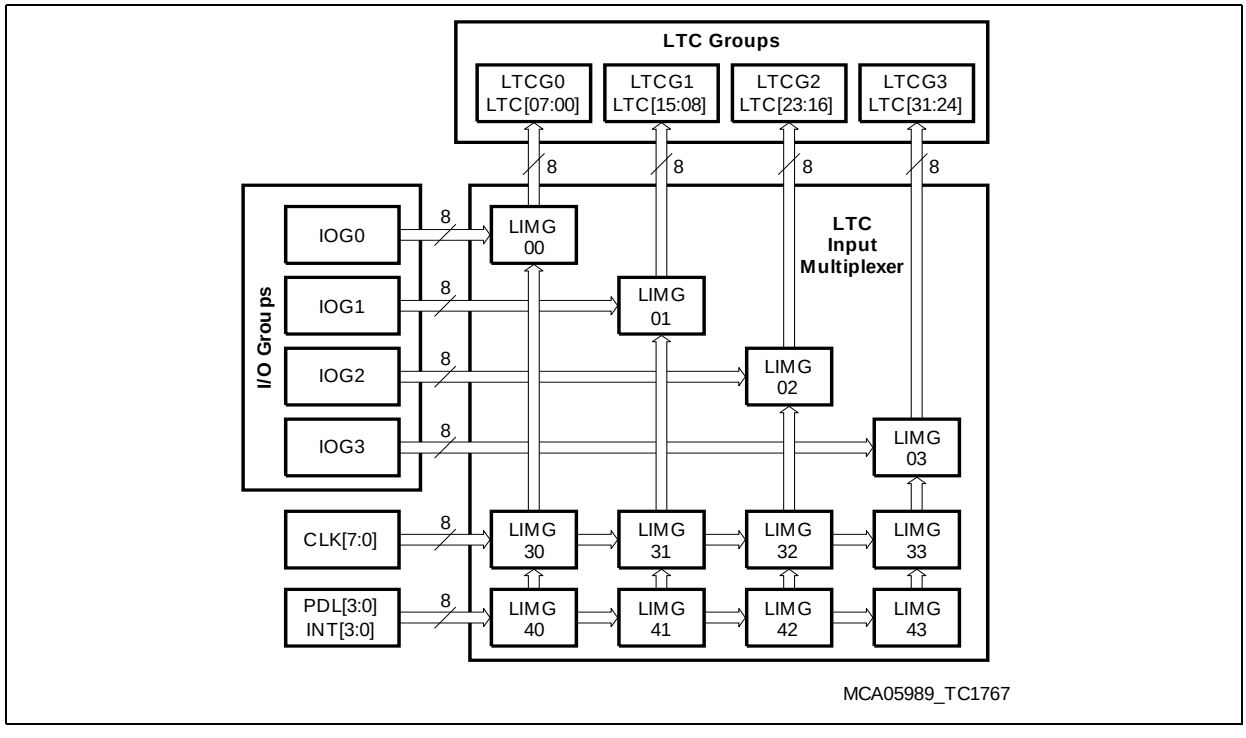


Figure 22-87 LTC Input Multiplexer of LTCA

The LTC input multiplexer contains LTC input Multiplexer Groups (LIMGs) that connect the I/O groups or the clock, PDL, or INT inputs to the input lines of the LTCs, organized in four LTC groups with 8 cells each. IOGs are grouped into four IOGs (IOG[3:0]) with eight lines each. Two special groups are available, a clock group with eight lines representing the clock bus inputs CLK[7:0] and a PDL/INT group with eight outputs that combines the four PDL inputs and the four inputs INT[3:0] as a group of LIMGs inputs.

Figure 22-88 shows the logical structure of a LIMG.

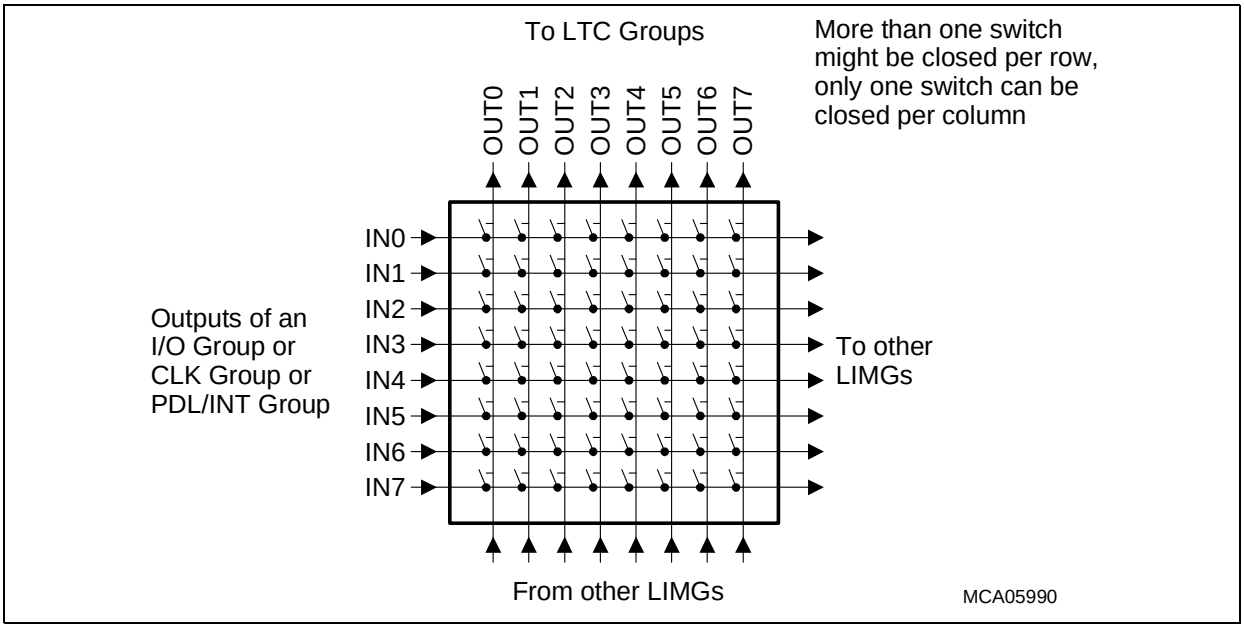


Figure 22-88 LTC Input Multiplexer Group (LIMG) Structure

Rules for connections to LTC Input Multiplexer Group LIMG:

- Within a I/O group, the line or the output of the cell with the lowest index number is connected to LIMG input line IN0. The remaining lines, cells or lines of a group are connected to LIMG input lines IN1 to IN7 with ascending index numbers. At the clock group, CLK0 is connected to IN0 and the remaining clock lines are connected to LIMG input lines IN1 to IN7 with ascending index numbers. At the PDL/INT group, PDL[3:0] (see **Page 22-22**) are connected to IN[3:0] and INT[3:0] are connected to IN[7:4].
  Example: for LIMG04 (see **Figure 22-87**), the I/O lines of IOG0 (IN00 up to IN07) are wired to its input lines IN0 to line IN7.
- Multiplexer output OUT0 of a LIMG is always connected to the input of an LTC group with the lowest index. The remaining output lines OUT1 to OUT7 are connected to the LTC inputs with ascending index.
  Example: for LIMG04 (see **Figure 22-87**), the outputs OUT0 to OUT7 are wired to the inputs of LTC32 to LTC39.
- An LTC input can be connected either to an I/O group output, or to a clock bus output, or to an PDL/INT output. This is guaranteed by the LIMG control register layout.

Otherwise, short circuits and unpredictable behavior would occur. In contrast, it is permitted that an I/O group output, or to a clock bus output, or a PDL/INT output is connected to more than one LTC input.

The LTC input multiplexer group configuration is based on the following principles:

- Each LIMG is referenced with two index variables: n and g (LIMGng)
- Index n is a group number. I/O groups IOG[3:0] have group number 0, I/O group IOG4 is not implemented, clock bus lines CLK[7:0] have group number 3, and the PDL/INT group has group number 4.
- Index g indicates the number of the LTC group g (g = 0-3) to which the outputs of the input multiplexer group LIMGng are connected.

The LTC input multiplexer logic as seen for programming is shown in **Figure 22-89**. With this logic, three group signals (from I/O groups, clock group, or PDL/INT group) are always combined to one output line that leads to an LTC input of LTC group g. For example, when looking at **Figure 22-87**, each of the eight LTC input multiplexer output lines to LTC group LTCG2 is connected via three LIMGn2 (n = 0, 2, 3) to the eight outputs of I/O group IOG2, the clock group, and the PDL/INT group.

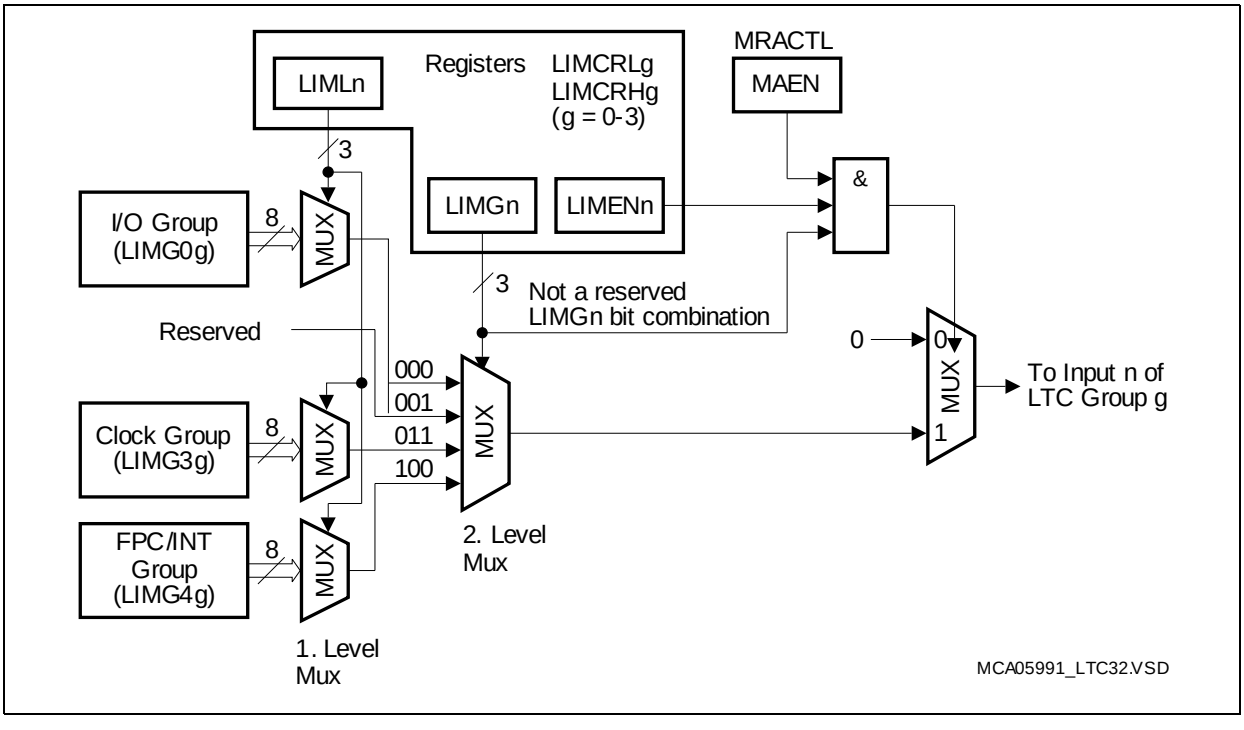


Figure 22-89 LTC Input Multiplexer (Programmer's View)

The 1. level multiplexer is built up by four three 8:1 multiplexers that are controlled in parallel by bit field LIMLn. The output of the multiplexer is connected only to the input of an LTC if bit LIMENn is set (enable multiplexer connection), and bit MRACTL.AEN is set (multiplexer array enabled), and no reserved bit combination is selected. If one of these conditions is not true, the corresponding LTC input will be held at a low level.

Two LTC Input Multiplexer Control Registers, LIMCRL (see also **Page 22-271** and **Page 22-272**), are assigned to each of the LTC groups. Therefore, in total eight registers control the connections within the LTC input multiplexer of the LTCA unit.

The LIMCRL registers control the LIMG output lines 0 to 3 and the LIMCRH registers control the LIMG output lines 4 to 7. **Table 22-22** lists all LTC Input Multiplexer Control Registers with its control functions. Please note that all LTC Input Multiplexer Control Registers are not directly accessible but must be written or read using a FIFO array structure as described on **Page 22-121**.

Table 22-22 LTC Input Multiplexer Control Register Assignments

LTC Group and LTCs		Controlled by Register	Selectable Groups via LIMGng
LTCG0	LTC[03:00]	LIMCRL0	IOG0, CLOCK, PDL/INT
	LTC[07:04]	LIMCRH0	
LTCG1	LTC[11:08]	LIMCRL1	IOG1, CLOCK, PDL/INT
	LTC[15:12]	LIMCRH1	
LTCG2	LTC[19:16]	LIMCRL2	IOG2, CLOCK, PDL/INT
	LTC[23:20]	LIMCRH2	
LTCG3	LTC[27:24]	LIMCRL3	IOG3, CLOCK, PDL/INT
	LTC[31:28]	LIMCRH3	

22.5.2.3 Multiplexer Register Array Programming

A total of 24 control registers are required to program the configuration of the output multiplexer and the LTC input multiplexer of the Input/Output Line Sharing Block. These IOLS control registers are combined into a Multiplexer Register Array FIFO that can only be read or written sequentially. Therefore, the control registers values cannot be accessed directly but must be accessed in a specific sequential order.

Three registers are available for controlling the Multiplexer Register Array:

- Multiplexer Register Array Control Register MRACTL
- Multiplexer Register Array Data Out Register MRADOUT
- Multiplexer Register Array Data In Register MRADIN

Figure 22-90 shows the structure of the multiplexer array FIFO with the arrangement of the multiplexer control registers.

For programming of the multiplexer array FIFO, the following steps must be executed:

1. Disable interconnections of the multiplexer array by writing MRACTL.MAEN = 0 (default after reset). The multiplexer array is disabled, all cell input lines are driven with 0, and device pins assigned to LTCA I/O lines or output lines are disconnected.
2. Reset the write cycle counter to 0 by writing MRACTL.WCRES = 1.

3. Write sequentially the multiplexer control register contents one after the other (24 values) into MRADIN, starting with the register values for OMCRH10, OMCRL10, … up to LIMCRH0, LIMCRL0 (see **Figure 22-90**). After the first MRADIN write operation, the contents for OMCRH10 is at FIFO position 1. With each following MRADIN write operation, it becomes shifted one FIFO position upwards. After the 24. MRADIN write operation, the OMCRH10 value is at its final position. The contents of FIFO position 24 can be read via register MRADOUT. With each MRADIN write operation the write cycle counter MRACTL.FIFOFILLCNT is incremented by 1. After all FIFO entries have been written, the FIFO is locked, bit MRACTL.FIFOFULL is set, and further MRADIN write operations are discarded until bit MRACTL.WCRES is written again with a 0.
4. Enable the multiplexer array by writing MRACTL.MAEN = 1. This establishes and enables all programmed interconnections.

To check the FIFO contents, the FIFO can be written a second time. At this check MRADIN is written before MRADOUT is read. This will return the FIFO contents of the first write sequence in the order of OMCRH10, OMCRL10, …, LIMCRL0.

Before disabling the multiplexer array FIFO, LTCA output pins that are already enabled as LTCA output should be switched to GPIO function to avoid output spikes. After enabling the multiplexer array FIFO again, the LTCA output can be switched again back to LTCA output function.

Shifting the write data through the FIFO requires a few clock cycles. When new data becomes written before the FIFO is ready to accept them, wait states will be inserted into the write access.

If the OMCRLg register bit field OMGn of the multiplexer array is programmed with an invalid (reserved) value, the related outputs will be forced to 0. When the array is disabled (MRACTL.MAEN = 0), all cell inputs and outputs are disconnected from the GPIO lines and are driven with 0.

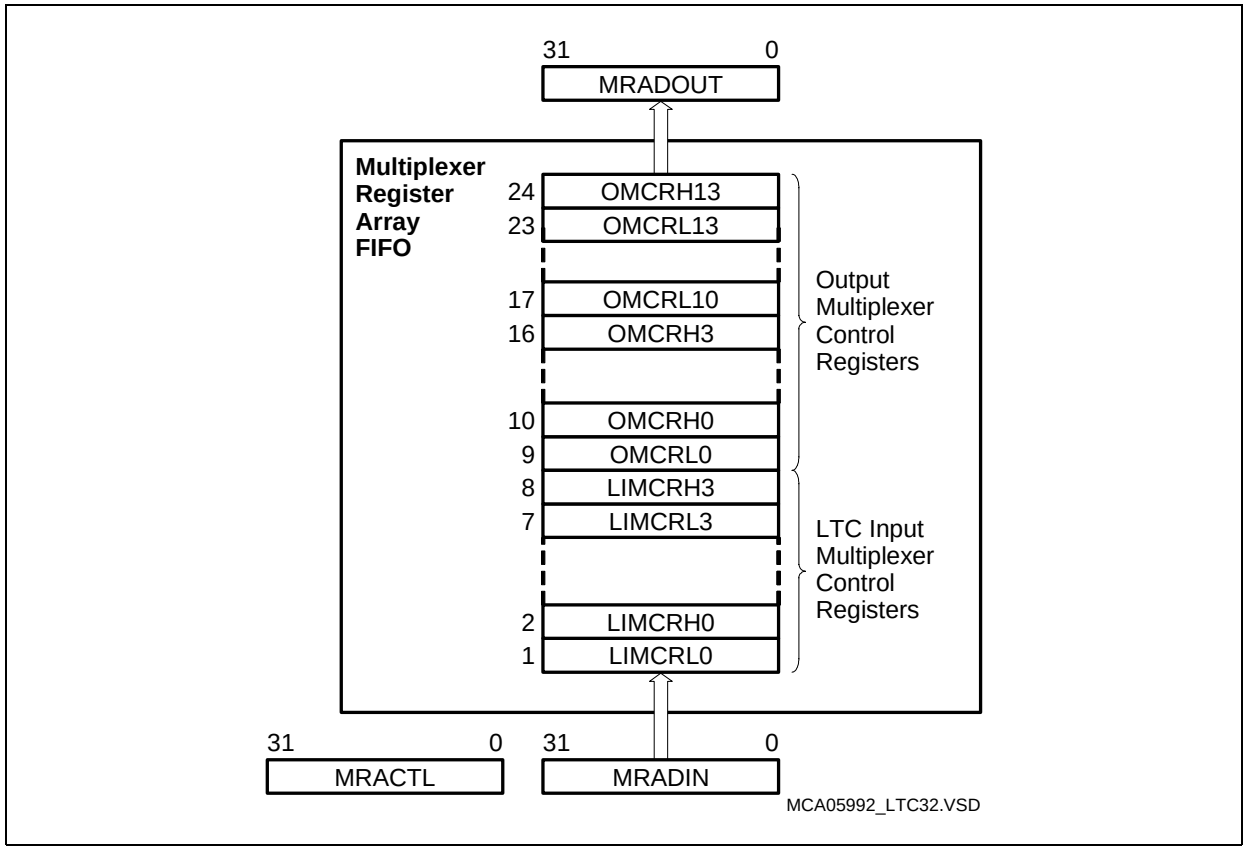


Figure 22-90 LTCA Multiplexer Array Control Register FIFO Structure

22.5.3 Interrupt Sharing Block (IS)

The LTCA provides 8 service request sources. These service request sources are generated by the LTCs.

Table 22-23 LTCA Number of Service Request Sources

Cell Type	Number of Cells	Number of Service Request Sources/Cell	Total Number of Request Sources
LTC	32	1	8

To reduce hardware and software overhead, four request sources are combined together in service request groups. A service request group y (y = 00-7) has four service request inputs and one service request output SRy which is typically connected outside the LTCA kernel to a standard interrupt node y and controlled by its SRCy register.

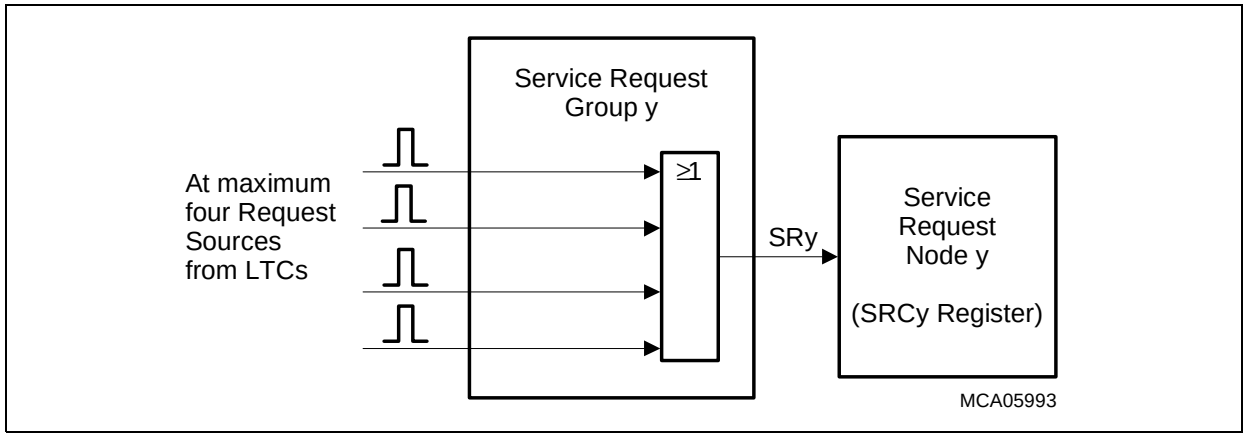


Figure 22-91 Service Request Groups

The bits in the Service Request State Registers (SRSSx and SRSCx, x = 3, 2) are service request status flags that are set by hardware (type “h”) when the related event occurs. Each LTCA service request source has its own service request flag. This flag is normally set by hardware but can be set and reset by software. Each service request status flag can be read twice, at the same bit location in SRSCx register and in SRSSx register, and cleared or set by software when writing to the corresponding request bit in SRSCx or SRSSx. When writing to SRSCx or SRSSx, several flags can be cleared at once by one write operation. Flags written with 0 are not influenced. This feature allows fast, simple clearing or setting of request flags without affecting other bits in the same register.

Note that the service request flag is always set by the service request event even if the corresponding service request is disabled in the interrupt node.

Table 22-24 lists the interrupt requests used by LTCA.

Table 22-24 LTCA Service Request Groups

Service Request Group Number	Source 1	Source 2	Source 3	Source 4
00	LTC00	LTC01	LTC02	LTC03
01	LTC04	LTC05	LTC06	LTC07
02	LTC08	LTC09	LTC10	LTC11
03	LTC12	LTC13	LTC14	LTC15
04	LTC16	LTC17	LTC18	LTC19
05	LTC20	LTC21	LTC22	LTC23
06	LTC24	LTC25	LTC26	LTC27
07	LTC28	LTC29	LTC30	LTC31

22.6 LTCA Kernel Registers

This section describes the LTCA kernel registers. Some of the kernel registers (SRSCn, SRSSn, LTCCTRk, and LTCXRk) are already described as GPTA®v5 kernel register at the pages as referenced in column “Description see” of **Table 22-25**. The multiplexer array FIFO registers are LTCA specific and therefore defined on the next pages.

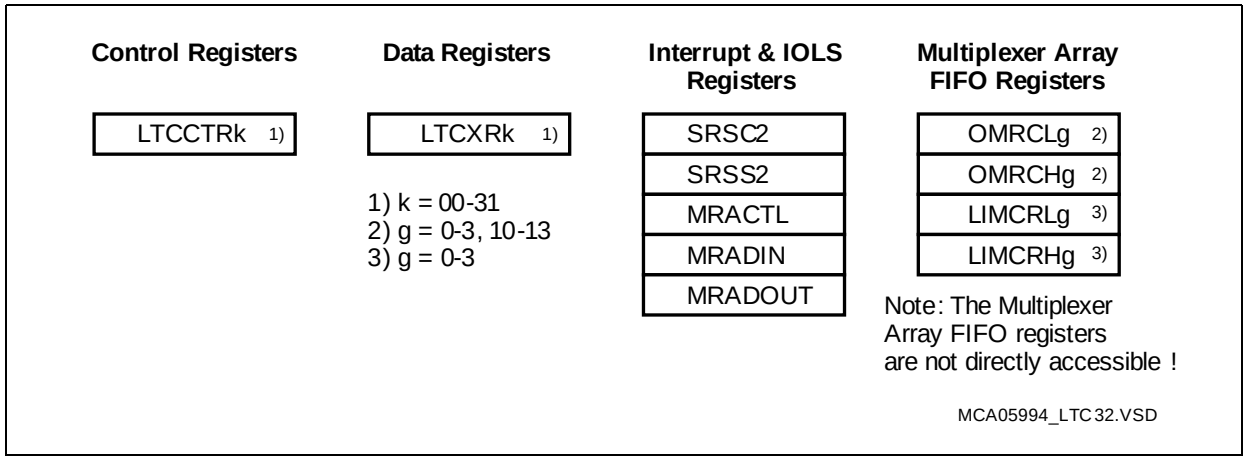


Figure 22-92 LTCA2 Kernel Registers

Table 22-25 LTCA2 Kernel Registers

Register Short Name	Register Long Name	Offset Addr.	Access Mode		Reset Class	Description see
			Read	Write		
ID	LTCA Identification Register	0008_H	U, SV	nBE	1	Page 22-166
SRSC2	Service Request State Clear Register 2	0020_H	U, SV	SV, E	3	Page 22-228
SRSS2	Service Request State Set Register 2	0024_H	U, SV	SV, E	3	Page 22-229
MRACTL	Multiplexer Register Array Control Register	0038_H	U, SV	U, SV	3	Page 22-264
MRADIN	Multiplexer Register Array Data In Register	$003C_H$	U, SV, 32	U, SV, 32	3	Page 22-266
MRADOUT	Multiplexer Register Array Data Out Register	0040_H	U, SV, 32	U, SV, 32	3	Page 22-266
LTCCTRk	Local Timer Cell Control Register k (k = 00-30)	$0200_H + k \times 8$	U, SV	U, SV	3	Page 22-252 Page 22-255 Page 22-258

Table 22-25 LTCA2 Kernel Registers (cont'd)

Register Short Name	Register Long Name	Offset Addr.	Access Mode		Reset Class	Description see
			Read	Write		
LTCXRk	Local Timer Cell X Register k (k = 00-30)	0204_H + k × 8	U, SV	U, SV	3	Page 22-263
LTCCTR31	Local Timer Cell Control Register 31	0200_H + 31 × 8	U, SV	U, SV	3	Page 22-261
LTCXR31	Local Timer Cell X Register 31	0204_H + 31 × 8	U, SV	U, SV	3	Page 22-263
OMCRLg	Output Multiplexer Control Register for Lower Half of Group g (g = 0-3, 10-13)	not directly address-able see Page 22-245	n.a.	n.a.	3	Page 22-268
OMCRHg	Output Multiplexer Control Register for Upper Half of Group g (g = 0-3, 10-13)		n.a.	n.a.	3	Page 22-269
LIMCRLg	Input Multiplexer Control Register for Lower Half of LTC Group g (g = 0-3)	not directly address-able see Page 22-245	n.a.	n.a.	3	Page 22-271
LIMCRHg	Input Multiplexer Control Register for Upper Half of LTC Group g (g = 0-3)		n.a.	n.a.	3	Page 22-272

22.6.1 Bit Protection

Bits with bit protection (this is valid, for example, for all bits in the Service Request State Registers) are not changed during a read-modify-write instruction, that is when hardware sets a request state bit between the read and the write of the read-modify-write sequence. For bit protected bits it is guaranteed that a hardware setting operation always has priority. Thus, no hardware triggered events are lost.

22.6.2 Service Request Registers

See **"GPTA0_SRSC2" on Page 22-228**.

22.6.3 Local Timer Cell Registers

LTCA2_LTCCTRk (k = 00-30)
Local Timer Cell Control Register k [Timer Mode]
(200_H+k*8_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															G BYP
r															rw

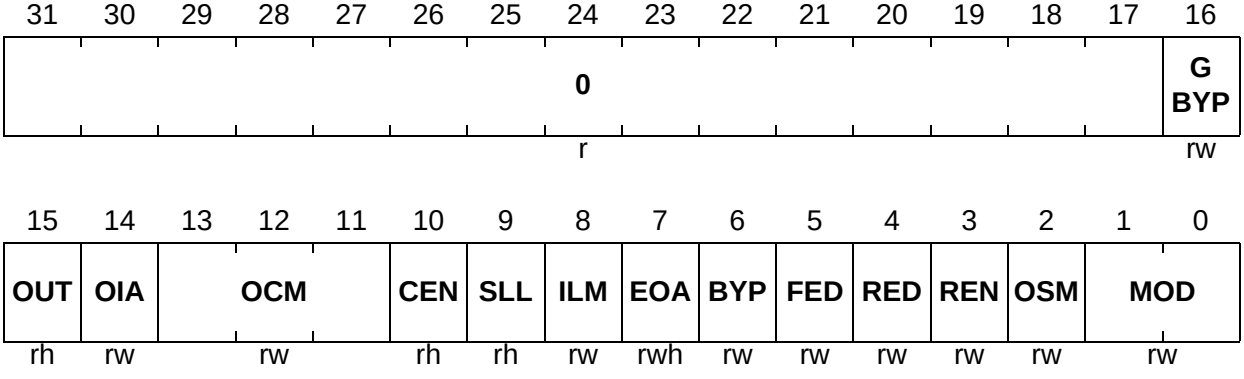
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OUT	OIA	OCM			CEN	CUD	ILM	CUD CLR	SLO	FED or AIL	RED or PEN	REN	OSM	MOD	
rh	rw	rw			rh	rwh	rw	w	rwh	rw	rw	rw	rw	rw	

Field	Bits	Type	Description
MOD	[1:0]	rw	**Mode Control Bits** 00_B LTCk operates in Capture Mode. 01_B LTCk operates in Compare Mode. 10_B LTCk operates in Free-Running Timer Mode. 11_B LTCk operates in reset Timer Mode.
OSM	2	rw	**One Shot Mode Enable** 0_B LTCk is continuously enabled. 1_B LTCk is enabled for one event only.
REN	3	rw	**Request Enable** 0_B Service request is disabled. 1_B Service request SQSk is activated when a - capture event has occurred - compare event has occurred - timer overflow has happened depending on the operation mode selected by bit field MOD.
RED	4	rw	**ILM = 0: Input Rising Edge Select** 0_B Timer is not updated by a rising edge. 1_B Timer is updated by a rising edge on the LTCkIN input line.

Field	Bits	Type	Description
PEN	4	rw	**ILM = 1: LTC Prescaler Enable** 0_B LTC Prescaler Mode is disabled. 1_B LTC Prescaler Mode with LTC prescaler clock LTCPRE is enabled.
FED	5	rw	**ILM = 0: Input Falling Edge Select** 0_B Timer is not updated by a falling edge. 1_B Timer is updated by a falling edge on the LTCkIN input line.
AIL	5	rw	**ILM = 1: Active Input Level Select** 0_B Input signal is active high. 1_B Input signal is active low.
SLO	6	rwh	**Select Line Output** 0_B State of select line output SO is 0. 1_B State of select line output SO is 1. SLO is bit protected (see **Page 22-167**).
CUDCLR	7	w	**Coherent Update Disable** 0_B No effect. 1_B Coherent update disabled (bit CUD is cleared). If bits CUD and CUDCLR are both written with 1, bit CUD will be set. CUDCLR is always read as 0.
ILM	8	rw	**Input Line Mode** 0_B Input line is operating in Edge Sensitive Mode. 1_B Input line is operating in Level Sensitive Mode. In case of full speed GPTA®v5 module clock selection as input clock, Level Sensitive Mode must be selected. In this case the Edge Sensitive Mode will not produce any event.
CUD	9	rwh	**Coherent Update Enable** 0_B Select output SO is not toggled on timer reset overflow. 1_B Select output SO is toggled on next timer reset overflow. When CUD is set by software (writing CUD and CUDCLR both with 1), it remains set until the next timer reset overflow (LTCk reset event) occurs and is cleared by hardware afterwards. CUD can be reset by software by writing bit CUDCLR with 1 and CUD with 0.
CEN	10	rh	**Cell Enable** 0_B LTCk is currently disabled for local events. 1_B LTCk is currently enabled for local events.

Field	Bits	Type	Description
OCM	[13:11]	rw	**Output Control Mode Select** $X00_B$ Current state of LTCkOUT output line is hold. $X01_B$ Current state of LTCkOUT output line is toggled. $X10_B$ LTCkOUT output line is forced to 0. $X11_B$ LTCkOUT output line is forced to 1. $0XX_B$ LTCkOUT output line state is set by an internal LTCk event only. $1XX_B$ LTCkOUT output line state is affected by an internal LTCk event and/or by an operation occurred in an adjacent LTCk cell (reported by M1I/M0I interface lines).
OIA	14	rw	**Output Immediate Action** 0_B No immediate action required. 1_B Action defined by bit field OCM must be performed immediately. OIA is always read as 0.
OUT	15	rh	**Output State** 0_B LTCkOUT output line is 0. 1_B LTCkOUT output line is 1.
GBYP	16	rw	**Global Bypass** 0_B M3O/M2O lines are affected by M1I/M0I lines. 1_B M3O/M2O lines are affected by M3I/M2I lines.
0	[31:17]	r	**Reserved** Read as 0; should be written with 0.

LTCA2_LTCCTRk (k = 00-30)
Local Timer Cell Control Register k [Capture Mode]
(200_H+k*8_H) **Reset Value: 0000 0000_H**

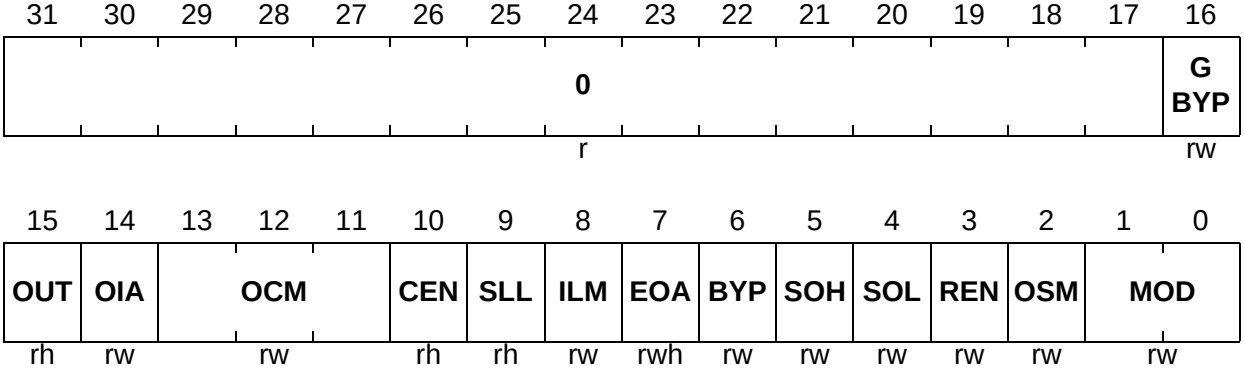


Field	Bits	Type	Description
MOD	[1:0]	rw	**Mode Control Bits** 00_B LTCk operates in Capture Mode. 01_B LTCk operates in Compare Mode. 10_B LTCk operates in Free-Running Timer Mode. 11_B LTCk operates in reset Timer Mode.
OSM	2	rw	**One Shot Mode Enable** 0_B LTCk is continuously enabled. 1_B LTCk is enabled for one event only.
REN	3	rw	**Request Enable** 0_B Service request is disabled. 1_B Service request SQSk is activated when a - capture event has occurred - compare event has occurred - timer overflow has happened depending on the operation mode selected by bit field MOD.
RED	4	rw	**Input Rising Edge Select** 0_B Capture event is not triggered by a rising edge. 1_B Capture event is triggered by a rising edge on the LTCkIN input line.
FED	5	rw	**Input Falling Edge Select** 0_B Capture event is not triggered by a falling edge. 1_B Capture event is triggered by a falling edge on the LTCkIN input line.

Field	Bits	Type	Description
BYP	6	rw	**Local Bypass** 0_B M1O/M0O lines are affected either by M1I/M0I lines or by OCM1/OCM0 bits. 1_B M1O/M0O lines are affected only by M1I/M0I (GBYP = 0) or M2I/M2I (GBYP = 1) lines. This bit is cleared if mode is switched to Timer Mode. OCM2 must be set in any case to enable reaction on M1I/M0I change.
EOA	7	rwh	**Enable On Action** 0_B LTCk is enabled for local events. 1_B LTCk is disabled for local events. On an event on the communication link via M0I/M1I lines, EOA will be cleared and local events will be enabled. EOA is bit protected (see **Section 22.4.2**). EOA is cleared if mode is switched to Timer Mode.
ILM	8	rw	**Input Line Mode** 0_B Input line is operating in Edge Sensitive Mode. 1_B Input line is operating in Level Sensitive Mode. In case of full speed GPTA®v5 module clock selection as input clock, Level Sensitive Mode must be selected. In this case the Edge Sensitive Mode will not produce any event.
SLL	9	rh	**Capture & Compare Mode: Select Line Level** 0_B Current state of select input SI is 0. 1_B Current state of select input SI is 1.
CEN	10	rh	**Cell Enable** 0_B LTCk is currently disabled for local events. 1_B LTCk is currently enabled for local events.
OCM	[13:11]	rw	**Output Control Mode Select** $X00_B$ Current state of LTCkOUT output line is hold. $X01_B$ Current state of LTCkOUT output line is toggled. $X10_B$ LTCkOUT output line is forced to 0. $X11_B$ LTCkOUT output line is forced to 1. $0XX_B$ LTCkOUT output line state is set by an internal LTCk event only. $1XX_B$ LTCkOUT output line state is affected by an internal LTCk event and/or by an operation occurred in an adjacent LTCk cell (reported by M1I/M0I interface lines).

Field	Bits	Type	Description
OIA	14	rw	**Output Immediate Action** 0_B No immediate action required. 1_B Action defined by bit field OCM must be performed immediately. OIA is always read as 0.
OUT	15	rh	**Output State** 0_B LTCkOUT output line is 0. 1_B LTCkOUT output line is 1.
GBYP	16	rw	**Global Bypass** 0_B M3O/M2O lines are affected by M1I/M0I lines. 1_B M3O/M2O lines are affected by M3I/M2I lines.
0	[31:17]	r	**Reserved** Read as 0; should be written with 0.

LTCA2_LTCCTRk (k = 00-30)
Local Timer Cell Control Register k [Compare Mode]
(200_H+k*8_H) **Reset Value: 0000 0000_H**



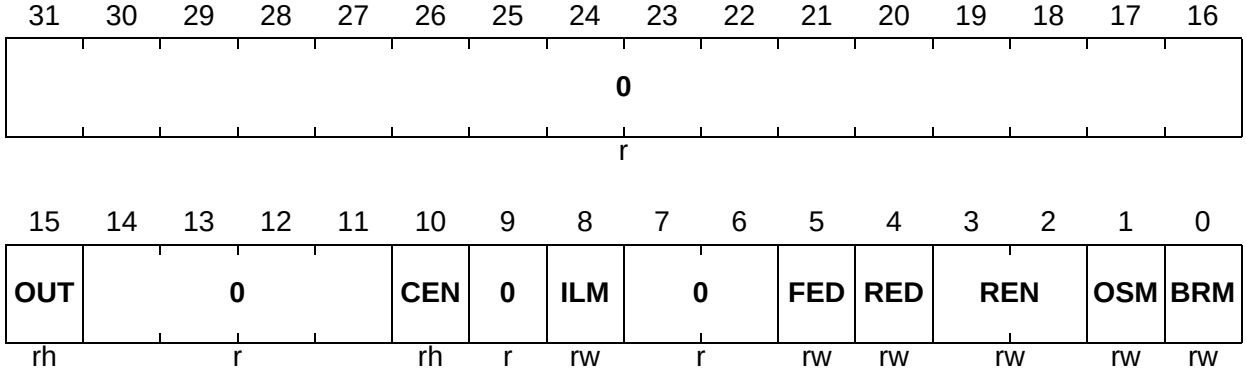
Field	Bits	Type	Description
MOD	[1:0]	rw	**Mode Control Bits** 00_B LTCk operates in Capture Mode. 01_B LTCk operates in Compare Mode. 10_B LTCk operates in Free-Running Timer Mode. 11_B LTCk operates in reset Timer Mode.
OSM	2	rw	**One Shot Mode Enable** 0_B LTCk is continuously enabled. 1_B LTCk is enabled for one event only.
REN	3	rw	**Request Enable** 0_B Service request is disabled. 1_B Service request SQSk is activated when a - capture event has occurred - compare event has occurred - timer overflow has happened depending on the operation mode selected by bit field MOD.
SOL	4	rw	**Compare Mode: Select Output Low** 0_B Compare is deactivated or on high level. 1_B Compare operation is enabled by a low level on select input SI[1)].
SOH	5	rw	**Compare Mode: Select Output High** 0_B Compare is deactivated or on high level. 1_B Compare operation is enabled by a high level on select input SI[1)].

Field	Bits	Type	Description
BYP	6	rw	**Bypass** 0_B M1O/M0O lines are affected either by M1I/M0I lines or by OCM1/OCM0 bits. 1_B M1O/M0O lines are affected only by M1I/M0I lines. This bit is cleared if mode is switched to Timer Mode. OCM2 must be set in any case to enable reaction on M1I/M0I change.
EOA	7	rwh	**Enable On Action** 0_B LTCk is enabled for local events. 1_B LTCk is disabled for local events. On an event on the communication link via M0I/M1I lines, EOA will be cleared and local events will be enabled. EOA is bit protected (see **Section 22.4.2**). EOA is cleared if mode is switched to Timer Mode.
ILM	8	rw	**Input Line Mode** 0_B Input line is operating in Edge Sensitive Mode. 1_B Input line is operating in Level Sensitive Mode. In case of full speed GPTA®v5 module clock selection as input clock, Level Sensitive Mode must be selected. In this case the Edge Sensitive Mode will not produce any event.
SLL	9	rh	**Select Line Level** 0_B Current state of select input SI is 0. 1_B Current state of select input SI is 1.
CEN	10	rh	**Cell Enable** 0_B LTCk is currently disabled for local events. 1_B LTCk is currently enabled for local events.
OCM	[13:11]	rw	**Output Control Mode Select** $X00_B$ Current state of LTCkOUT output line is hold. $X01_B$ Current state of LTCkOUT output line is toggled. $X10_B$ LTCkOUT output line is forced to 0. $X11_B$ LTCkOUT output line is forced to 1. $0XX_B$ LTCkOUT output line state is set by an internal LTCk event only. $1XX_B$ LTCkOUT output line state is affected by an internal LTCk event and/or by an operation occurred in an adjacent LTCk cell (reported by M1I/M0I interface lines).

Field	Bits	Type	Description
OIA	14	rw	**Output Immediate Action** 0_B No immediate action required. 1_B Action defined by bit field OCM must be performed immediately. OIA is always read as 0.
OUT	15	rh	**Output State** 0_B LTCkOUT output line is 0. 1_B LTCkOUT output line is 1.
GBYP	16	rw	**Global Bypass** 0_B M3O/M2O lines are affected by M1I/M0I lines. 1_B M3O/M2O lines are affected by M3I/M2I lines.
0	[31:17]	r	**Reserved** Read as 0; should be written with 0.

1) To enable Compare Mode in all cases, SOL and SOH bits must be set to 1.

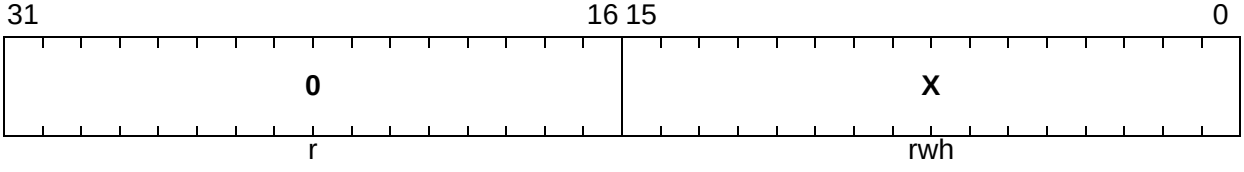
LTCA2_LTCCTR31
Local Timer Cell Control Register 31 ($2F8_H$) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
BRM	0	rw	**Bit Reversal Mode Control** 0_B Compare uses normal sequence of local input data bus (YI) bits. 1_B Compare uses reversed sequence of local input data bus (YI) bits.
OSM	1	rw	**One Shot Mode Enable for Shadow Register Copy** 0_B Shadow register copy is continuously enabled. 1_B Shadow register copy is enabled for one event only.
REN	[3:2]	rw	**Request Enable** 00_B Service request SQT63 is disabled. 01_B Service request SQT63 is generated when a compare event has occurred. 10_B Service request SQT63 is generated when a shadow register copy event has occurred. 11_B Reserved.
RED	4	rw	**Rising Edge Select for Shadow Register Copy** 0_B Shadow register copy is not triggered by a rising edge on the LTC63IN input line. 1_B Shadow register copy is triggered by a rising edge on the LTC63IN input line.
FED	5	rw	**Falling Edge Select for Shadow Register Copy** 0_B Shadow register copy is not triggered by a falling edge on the LTC63IN input line. 1_B Shadow register copy is triggered by a falling edge on the LTC63IN input line.

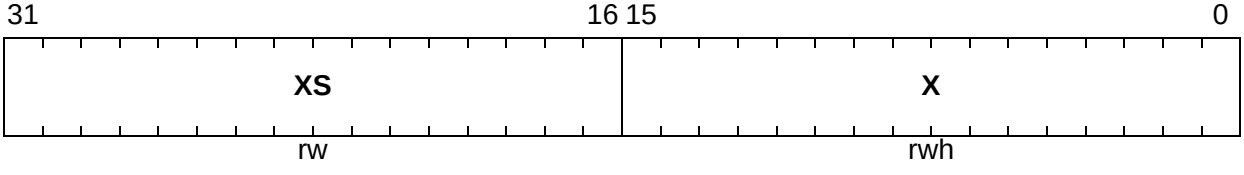
Field	Bits	Type	Description
ILM	8	rw	**Shadow Register Copy Input Line Mode** 0_B LTC63IN is operating in Edge Sensitive Mode. 1_B LTC63IN is operating in Level Sensitive Mode.
CEN	10	rh	**Enable for Shadow Register Copy** 0_B Shadow register copy is currently disabled. 1_B Shadow register copy is currently enabled.
OUT	15	rh	**Output State** 0_B LTC63OUT output line is 0. 1_B LTC63OUT output line is 1.
0	[7:6], 9, [14:11], [31:16]	r	**Reserved** Read as 0; should be written with 0.

LTCA2_LTCXRk (k = 00-30)
Local Timer Cell X Register k **(204_H+k*8_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
X	[15:0]	rwh	**Local Timer Data Register Value**
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

LTCA2_LTCXR31
Local Timer Cell X Register 31 **($2FC_H$)** **Reset Value: 0000 0000_H**



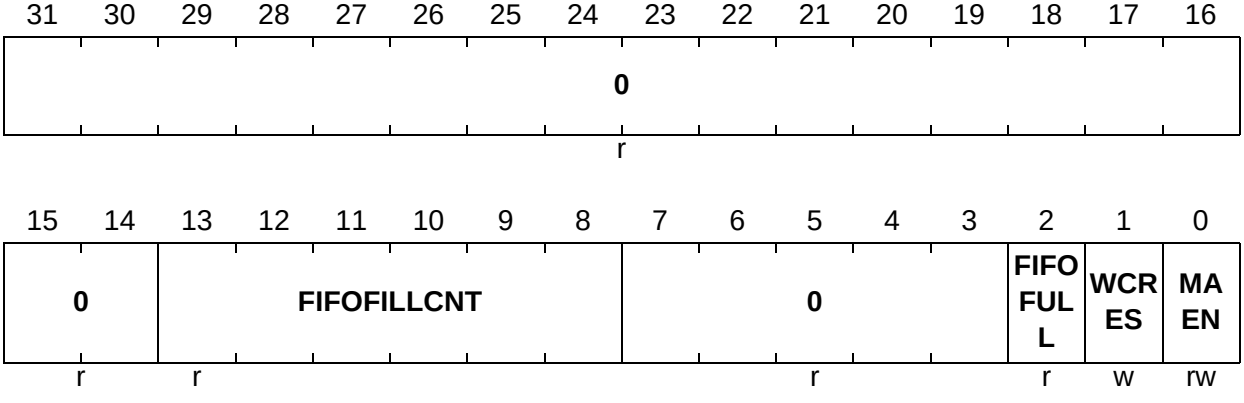
Field	Bits	Type	Description
X	[15:0]	rwh	**Compare Register Value** Software write operations has priority above a simultaneous hardware update.
XS	[31:16]	rw	**Shadow Register Value**

22.6.4 I/O Sharing Block Registers

The three registers MRACTL, MRADIN, and MRADOUT are used to write data to and read data from the LTCA Multiplexer Register Array FIFO. The Multiplexer Register Array FIFO controls the operation of the Input/Output Line Sharing Block (see **“Input/Output Line Sharing Block (IOLS)” on Page 22-98**).

The Multiplexer Register Array Control register controls the operation of the Multiplexer Register Array FIFO.

LTCA2_MRACTL
Multiplexer Register Array Control Register
(038_H) **Reset Value: 0000 0000_H**

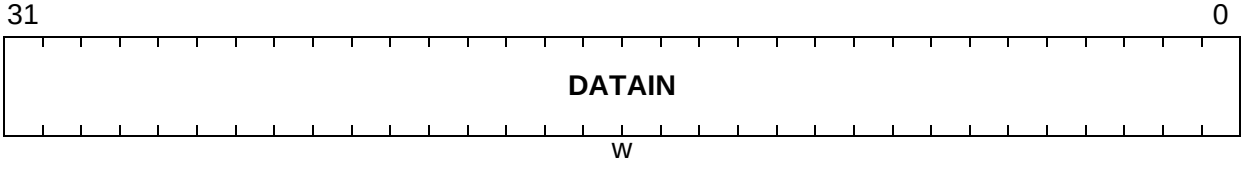


Field	Bits	Type	Description
MAEN	0	rw	**Multiplexer Array Enable** Bit field MAEN enables/disables the programming and the interconnections of the multiplexer array. 0_B Multiplexer array is disabled; all cell inputs are driven with 0, LTCA I/O lines (pins) are disconnected and FIFO writing is enabled. 1_B Multiplexer array is enabled; all cell and I/O line interconnections are established as previously programmed and FIFO writing is disabled.
WCRES	1	w	**Write Count Reset** Writing WCRES with 1 while the array is disabled (MAEN = 0), resets the write cycle counter to zero and the FIFO written sequentially (initialized). WCRES is always read as 0.
FIFOFULL	2	r	**FIFO Full Status** 0_B FIFO not completely written (write access to MRADIN allowed). 1_B FIFO completely written (write access to MRADIN ignored). Must be re-enabled via WCRES before array can be re-initialized.

Field	Bits	Type	Description
FIFOFILLCNT	[13:8]	r	**FIFO Fill Count** This bit field shows the current contents of the write cycle counter.
0	[7:3], [31:14]	r	**Reserved** Read as 0; should be written with 0.

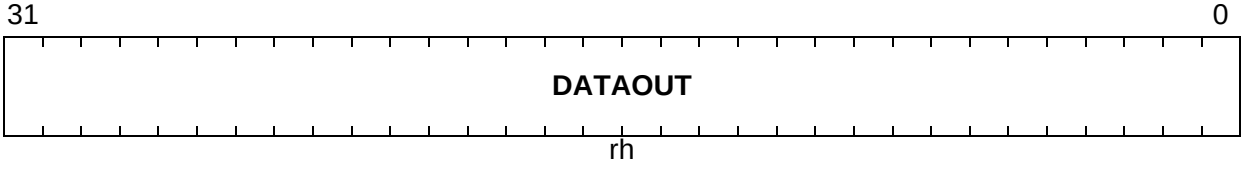
The Multiplexer Register Array Data In register is used to **write** data to the Multiplexer Register Array FIFO. The Multiplexer Register Array Data Out register is used to **read** data from the Multiplexer Register Array FIFO.

LTCA2_MRADIN
Multiplexer Register Array Data In Register
($03C_H$) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
DATAIN	[31:0]	w	**FIFO Write Data** This register contains the FIFO write data as defined for the LTC Output Multiplexer Control Registers and the LTC Input Multiplexer Control Registers.

LTCA2_MRADOUT
Multiplexer Register Array Data Out Register
(040_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
DATAOUT	[31:0]	rh	**FIFO Read Data** This register contains the FIFO read data as assigned for the LTC Output Multiplexer Control Registers and the LTC Input Multiplexer Control Registers.

Note: For correct operation, the MRADIN and MRADIN registers must be always read or written 32-bit wide. 8-bit and 16-bit accesses are ignored without any bus error!

22.6.5 Multiplexer Control Registers

These registers are not directly accessible and can be written and read only via the multiplexer register array FIFO (see **Page 22-121**).

22.6.5.1 Output Multiplexer Control Registers

Two registers, OMCRL and OMCRH, are assigned to each I/O Group IOG[3:0] and each Output Group OG[3:0]. OMCRL[3:0]/OMCRH[3:0] are assigned to IOG[3:0] and OMCRL[10:7]/OMCRH[10:7] are assigned to OG[3:0].

OMCRL controls the connections of group pins 0 to 3. OMCRH controls the connections of group pins 4 to 7.

LTCA2_OMCRLg (g = 0-3, 10-13)
Output Multiplexer Control Register for Lower Half of Pin Group g

Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		**OMG3**		**0**		**OML3**		**0**		**OMG2**		**0**		**OML2**	
r		rw		r		rw		r		rw		r		rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		**OMG1**		**0**		**OML1**		**0**		**OMG0**		**0**		**OML0**	
r		rw		r		rw		r		rw		r		rw	

Field	Bits	Type	Description
OML0, OML1, OML2, OML3	[2:0], [10:8], [18:16], [26:24]	rw	**Multiplexer Line Selection** This bit field selects the input line of a OMG that can be selected by bit field OMGn for OMG output n. 000_B OMG input IN0 selected 001_B OMG input IN1 selected 010_B OMG input IN2 selected 011_B OMG input IN3 selected 100_B OMG input IN4 selected 101_B OMG input IN5 selected 110_B OMG input IN6 selected 111_B OMG input IN7 selected
OMG0, OMG1, OMG2, OMG3	[6:4], [14:12], [22:20], [30:28]	rw	**Multiplexer Group Selection** This bit field determines the OMGng which is connected to input n of I/O Group g or Output group g-7. $XX1_B$ OMG1g selected All other combinations are reserved. If a reserved combination of OMGn value is selected, the corresponding OMG output is forced to 0 level. For compatibility reasons, the value 001_B (for $XX1_B$) should be used for OMGn bit field programming.
0	3, 7, 11, 15, 19, 23, 27, 31	r	**Reserved** Read as 0; should be written with 0.

LTCA2_OMCRHg (g = 0-3, 10-13)
Output Multiplexer Control Register for Upper Half of Pin Group g

Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		**OMG7**		**0**		**OML7**		**0**		**OMG6**		**0**		**OML6**	
r		rw		r		rw		r		rw		r		rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		**OMG5**		**0**		**OML5**		**0**		**OMG4**		**0**		**OML4**	
r		rw		r		rw		r		rw		r		rw	

Field	Bits	Type	Description
OML4, OML5, OML6, OML7	[2:0], [10:8], [18:16], [26:24]	rw	**Multiplexer Line Selection** This bit field selects the input line of a OMG that can be selected by bit field OMGn for OMG output n. 000_B OMG input IN0 selected 001_B OMG input IN1 selected 010_B OMG input IN2 selected 011_B OMG input IN3 selected 100_B OMG input IN4 selected 101_B OMG input IN5 selected 110_B OMG input IN6 selected 111_B OMG input IN7 selected
OMG4, OMG5, OMG6, OMG7	[6:4], [14:12], [22:20], [30:28]	rw	**Multiplexer Group Selection** This bit field determines the OMGng which is connected to input n of I/O Group g or Output group g-7. $XX1_B$ OMG1g selected All other combinations are reserved and if selected, the corresponding OMG output is forced to 0 level. For compatibility reasons, the value 001_B (for $XX1_B$) should be used for OMGn bit field programming.
0	3, 7, 11, 15, 19, 23, 27, 31	r	**Reserved** Read as 0; should be written with 0.

22.6.5.2 LTC Input Multiplexer Control Registers

Two registers, LIMCRL and LIMCRH, are assigned to each LTC group. LIMCRL controls the connections of LTC group cells with index 0 to 3. LIMCRH controls the connections of LTC group cells with index 4 to 7.

LTCA2_LIMCRLg (g = 0-3)
Input Multiplexer Control Register for Lower Half of LTC Group g

Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LIM EN3		LIMG3		0		LIML3		LIM EN2		LIMG2		0		LIML2	
r		rw		r		rw		r		rw		r		rw	

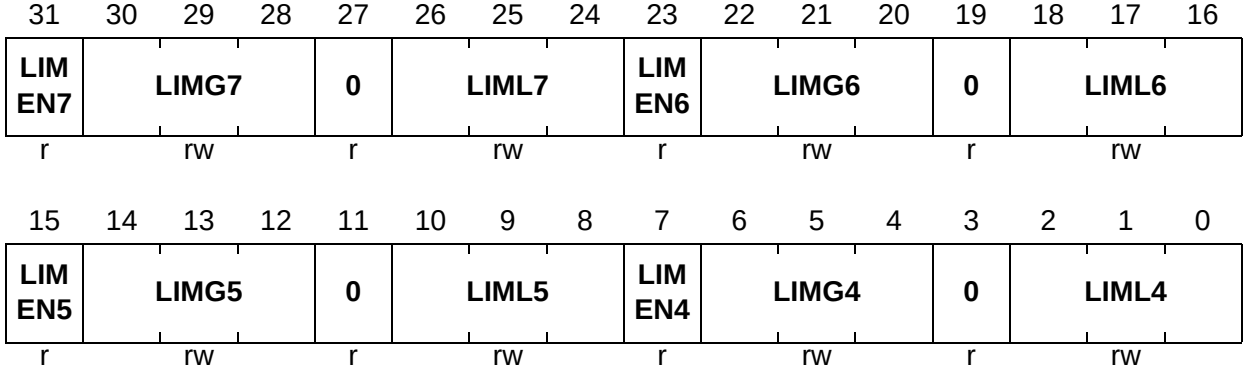
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LIM EN1		LIMG1		0		LIML1		LIM EN0		LIMG0		0		LIML0	
r		rw		r		rw		r		rw		r		rw	

Field	Bits	Type	Description
LIML0, LIML1, LIML2, LIML3	[2:0], [10:8], [18:16], [26:24]	rw	**Multiplexer Line Selection** This bit field selects the input line of a LIMG that can be selected by bit field LIMGn for LIMG output n. 000_B LIMG input IN0 selected 001_B LIMG input IN1 selected 010_B LIMG input IN2 selected 011_B LIMG input IN3 selected 100_B LIMG input IN4 selected 101_B LIMG input IN5 selected 110_B LIMG input IN6 selected 111_B LIMG input IN7 selected
LIMG0, LIMG1, LIMG2, LIMG3	[6:4], [14:12], [22:20], [30:28]	rw	**Multiplexer Group Selection** This bit field determines the LIMGng which is connected to input n of LTC group g. 000_B LIMG0g selected 011_B LIMG3g selected 100_B LIMG4g selected All other combinations are reserved. If a reserved combination of LIMGn is selected, or if LIMENn = 0, the corresponding LIMG output is forced to 0 level.
LIMEN0, LIMEN1, LIMEN2, LIMEN3	7, 15, 23, 31	rw	**Enable Multiplexer Connection** 0_B Input n is not connected to any line. 1_B Input n is connected to the line defined by LIMLn and LIMGn.

Field	Bits	Type	Description
0	3, 11, 19, 27	r	**Reserved** Read as 0; should be written with 0.

LTCA2_LIMCRHg (g = 0-3)
Input Multiplexer Control Register for Upper Half of LTC Group g

Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
LIML4, LIML5, LIML6, LIML7	[2:0], [10:8], [18:16], [26:24]	rw	**Multiplexer Line Selection** This bit field selects the input line of a LIMG that can be selected by bit field LIMGn for LIMG output n. 000_B LIMG input IN0 selected 001_B LIMG input IN1 selected 010_B LIMG input IN2 selected 011_B LIMG input IN3 selected 100_B LIMG input IN4 selected 101_B LIMG input IN5 selected 110_B LIMG input IN6 selected 111_B LIMG input IN7 selected
LIMG4, LIMG5, LIMG6, LIMG7	[6:4], [14:12], [22:20], [30:28]	rw	**Multiplexer Group Selection** This bit field determines the LIMGng which is connected to input n of LTC group g. 000_B LIMG0g selected 011_B LIMG3g selected 100_B LIMG4g selected All other combinations are reserved. If a reserved combination of LIMGn is selected, or if LIMENn = 0, the corresponding LIMG output is forced to 0 level.

Field	Bits	Type	Description
LIMEN4, LIMEN5, LIMEN6, LIMEN7	7, 15, 23, 31	rw	**Enable Multiplexer Connection** 0_B Input n is not connected to any line. 1_B Input n is connected to the line defined by LIMLn and LIMGn.
0	3, 11, 19, 27	r	**Reserved** Read as 0; should be written with 0.

22.7 GPTA®v5 Module Implementation

This section describes the GPTA®v5 interfaces as implemented in TC1797 with the clock control, port and Micro Second Channel connections, interrupt control, and address decoding.

22.7.1 Interconnections of GPTA0/GPTA1/LTCA2 Units

The following items are described in this section:

- GPTA®v5 module (kernel) external registers
- Port control and connections
  - I/O port line assignment
  - I/O function selection
  - Pad driver characteristics selection
  - Emergency control of GPTA®v5 outputs
- On-chip connections
  - Clock bus connections
  - MSC controller connections
  - FADC connections
  - MultiCAN, SCU, and DMA connections
  - SCU connections (ADC, DMA)
- Module clock generation
- Interrupt registers
- GPTA®v5 address map

Figure 22-93 shows the TC1797 specific implementation details and interconnections of the units GPTA0/GPTA1/LTCA2.The units are supplied by clock control and address decoding logic.

Each GPTA0/1 unit has 56 input signals and 112 output signals which can be connected to 56 port pins and 56 MSC interface lines. Additional four inputs for internal connections (coming from the SCU) and 38 service request outputs are provided.

The LTCA2 unit has 32 input signals and 64 output signals that can be connected to the same 56 port pins as GPTA0 and GPTA1 and MSC interface lines. Additional four inputs for internal connections (coming from the SCU) and 8 service request outputs are provided.

Additional connections are described in the following sections.

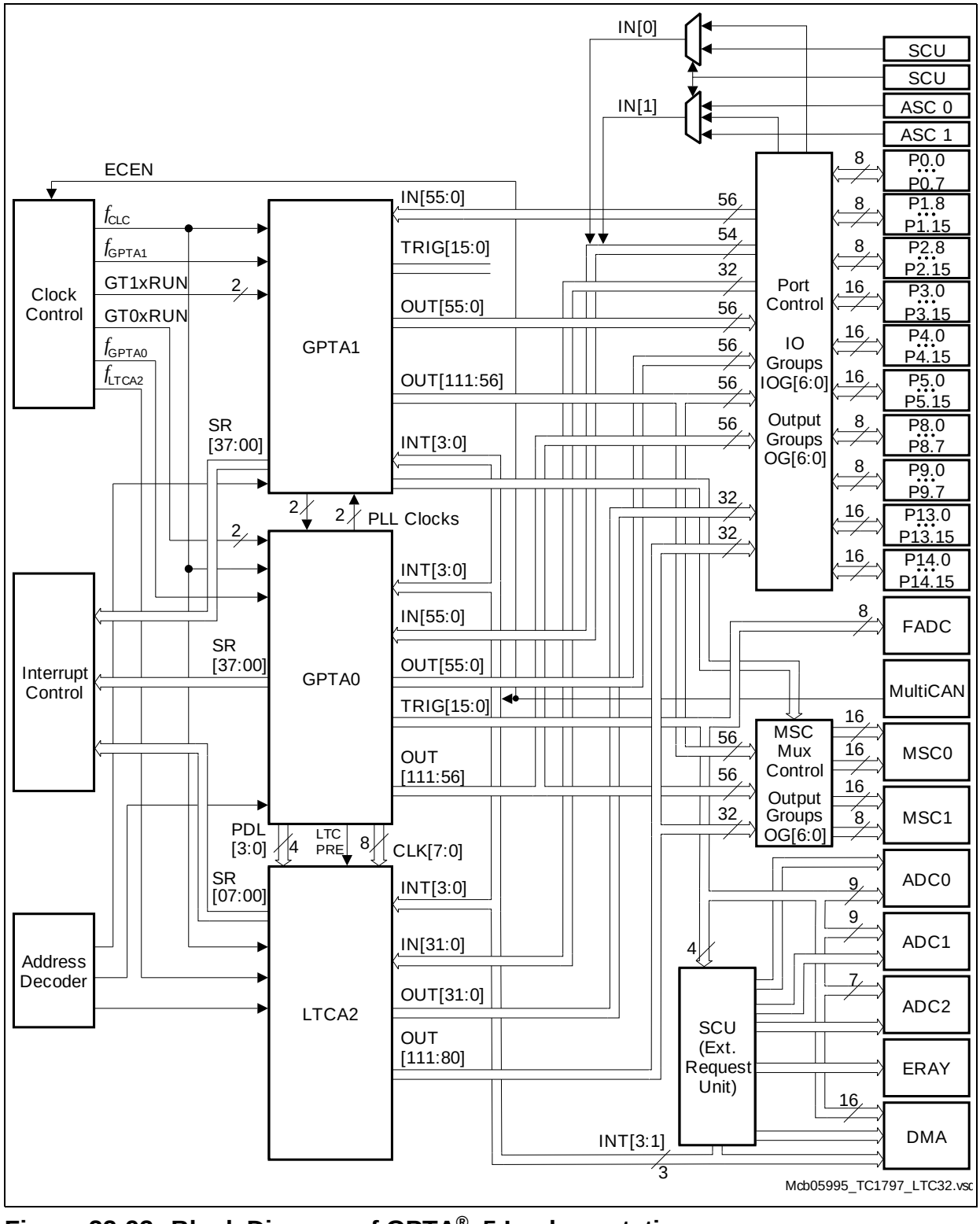


Figure 22-93 Block Diagram of GPTA®v5 Implementation

22.7.2 GPTA®v5 Module External Registers

Figure 22-94 summarizes the GPTA®v5 module related external registers that are required for GPTA0/GPTA1/LTCA2 programming. These registers are referenced and (some of it) described in detail in the following sub-sections.

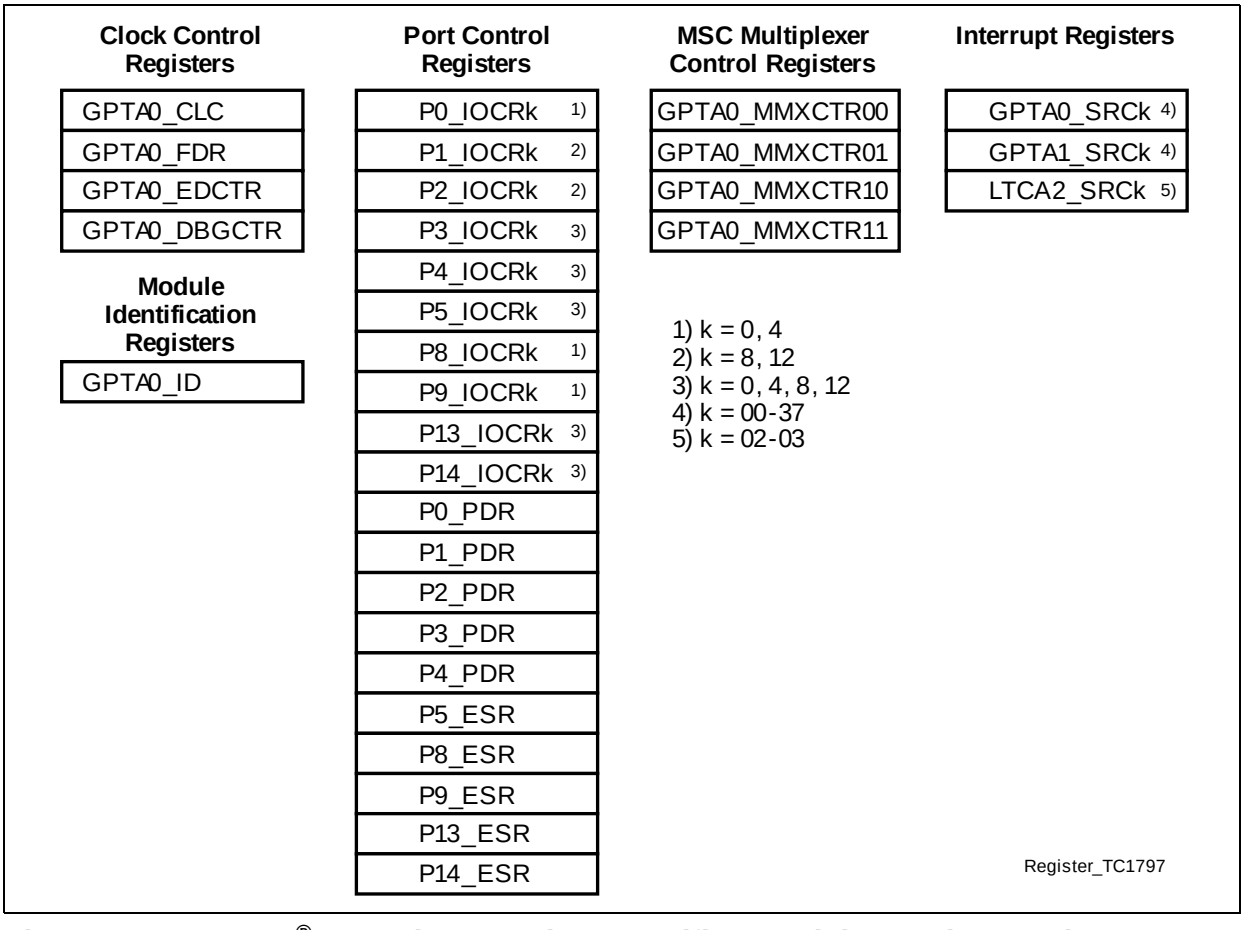


Figure 22-94 GPTA®v5 Implementation-Specific Special Function Registers

22.7.3 Port Control and Connections

This section describes the I/O connections of the GPTA0 unit.

22.7.3.1 I/O Port Line Assignment

In the TC1797, the seven I/O groups and seven output groups of GPTA0 and GPTA1 with their input lines IN[55:0] and output lines OUT[111:0] are assigned to seven 8-bit port groups as shown in **Figure 22-95**. Within an 8-bit I/O group, the IN/OUT line with lowest index number is assigned to the port line with the lowest index number. The remaining lines are assigned linearly with increasing index numbers. For example, P3.15 is assigned to IN13/OUT13. In the TC1797, the four I/O groups and four output groups of LTCA2 with their input lines IN[31:0] and output lines OUT[31:0] are assigned to five

8-bit port groups Therefore, the LTCA does not have IOG4, IOG5, and IOG6 and therefore it has no connection to Port 8 or Port 9 pins and only output connections to Port 4, pins 15 down to 8.

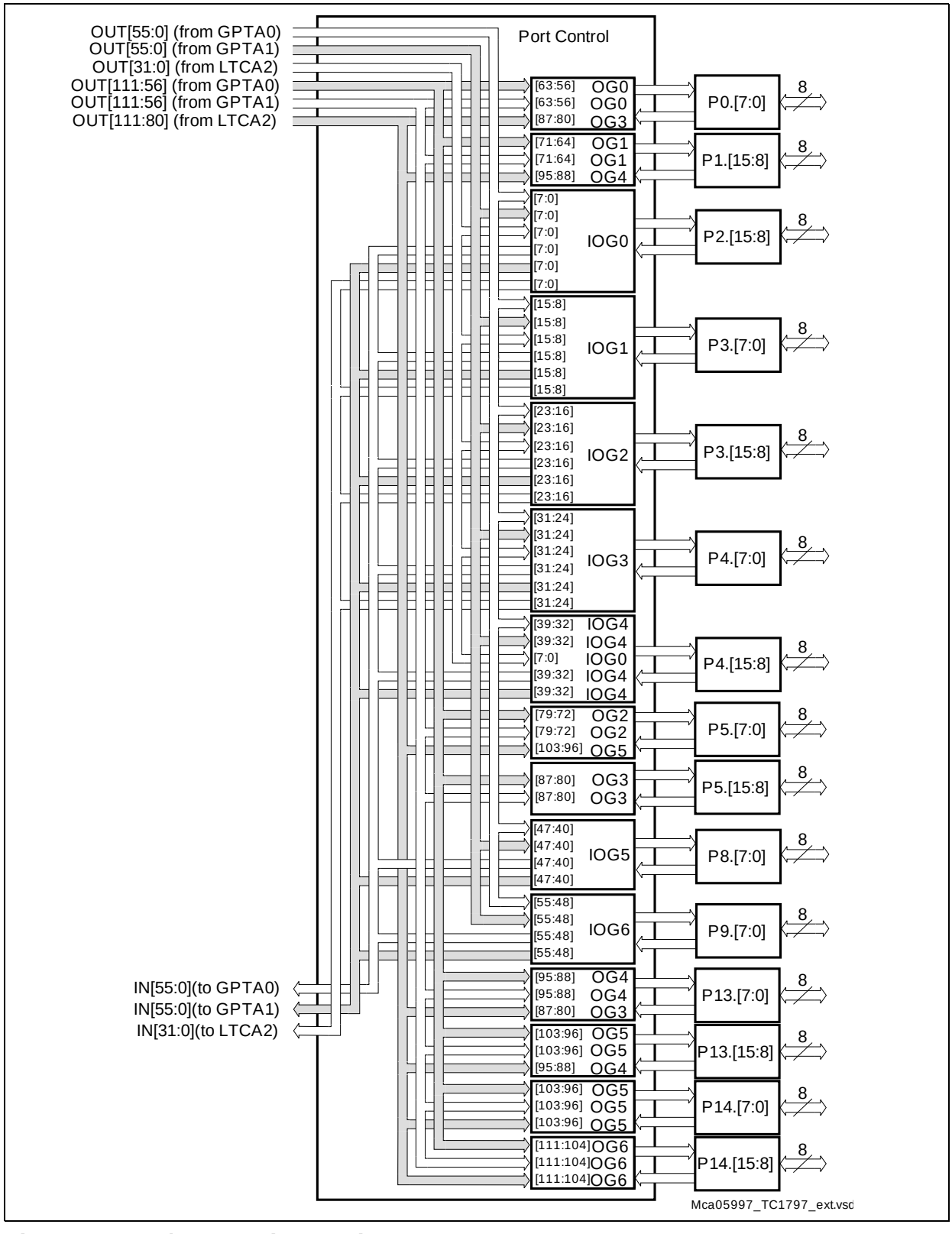


Figure 22-95 I/O Port Line Assignment

The interconnections between the GPTA0 unit and the port I/O lines are controlled in the port logics. The following port control operations selections must be executed:

- Input/output function selection (IOCR registers)
- Pad driver characteristics selection for outputs (PDR registers)

22.7.3.2 Input/Output Function Selection

The port input/output control registers contain bit fields that select the digital output and input driver characteristics such as port direction (input/output), pull-up/down device or open-drain selection for outputs, and alternate output selections. The I/O lines for the GPTA0/GPTA1/LTCA2 units are controlled by the port input/output control registers for Port 0, Port 1, Port 2, Port 3, Port 4,Port 5, Port 8, Port 9, Port13, and Port 14. Each of the input/output control registers controls four port lines using a 4-bit wide bit field PCx (definitions see **Table 22-27**). **Table 22-26** shows which of the input/output control register bit field is related to a specific GPTA0/GPTA1/LTCA2 unit I/O lines. Note that GPTA0/GPTA1/LTCA2 input P2.8/IN0 (see **Page 22-295**) and GPTA0/GPTA1/LTCA2 input P2.9/IN1 (see **Page 22-296**) has special connections.

Table 22-26 IOCR Assignment for GPTA®v5 Port Lines

Port	Port Lines for GPTA®v5	GPTA0/GPTA1 I/O Lines		LTCA2 I/O Lines		Controlled by IOCR Register
		Input	Output	Input	Output	
Port 0	P0.[3:0]		OUT[59:56]		OUT[83:80]	P0_IOCR0
	P0.[7:4]		OUT[63:60]		OUT[87:84]	P0_IOCR4
Port 1	P1.[11:8]		OUT[67:64][4)]		OUT[91:88][1)]	P1_IOCR8
	P1.[15:12]		OUT[71:68]		OUT[95:92][2)]	P1_IOCR12
Port 2	P2.[11:8]	IN[3:0][3)]	OUT[3:0]	IN[3:0][3)]	OUT[3:0]	P2_IOCR8
	P2.[15:12]	IN[7:4]	OUT[7:4]	IN[7:4]	OUT[7:4]	P2_IOCR12
Port 3	P3.[3:0]	IN[11:8]	OUT[11:8]	IN[11:8]	OUT[11:8]	P3_IOCR0
	P3.[7:4]	IN[15:12]	OUT[15:12]	IN[15:12]	OUT[15:12]	P3_IOCR4
	P3.[11:8]	IN[19:16]	OUT[19:16]	IN[19:16]	OUT[19:16]	P3_IOCR8
	P3.[15:12]	IN[23:20]	OUT[23:20]	IN[23:20]	OUT[23:20]	P3_IOCR12
Port 4	P4.[3:0]	IN[27:24]	OUT[27:24]	IN[27:24]	OUT[27:24]	P4_IOCR0
	P4.[7:4]	IN[31:28]	OUT[31:28]	IN[31:28]	OUT[31:28]	P4_IOCR4
	P4.[11:8]	IN[35:32]	OUT[35:32]		OUT[3:0]	P4_IOCR8
	P4.[15:12]	IN[39:36]	OUT[39:36]		OUT[7:4]	P4_IOCR12

Table 22-26 IOCR Assignment for GPTA®v5 Port Lines (cont'd)

Port	Port Lines for GPTA®v5	GPTA0/GPTA1 I/O Lines		LTCA2 I/O Lines		Controlled by IOCR Register
		Input	Output	Input	Output	
Port 5	P5.[3:0]		OUT[75:72]			P5_IOCR0
	P5.[7:4]		OUT[79:76][4]		OUT[103:100][5]	P5_IOCR4
	P5.[11:8]		OUT[83:80]			P5_IOCR8
	P5.[15:12]		OUT[87:84]			P5_IOCR12
Port 8	P8.[3:0]	IN[43:40]	OUT[43:40]			P8_IOCR0
	P8.[7:4]	IN[47:44]	OUT[47:44]			P8_IOCR4
Port 9	P9.[3:0]	IN[51:48]	OUT[51:48]			P9_IOCR0
	P9.[7:4]	IN[55:52]	OUT[55:52]			P9_IOCR4
Port 13	P13.[3:0]		OUT[91:88]		OUT[83:80]	P13_IOCR0
	P13.[7:4]		OUT[95:92]		OUT[87:84]	P13_IOCR4
	P13.[11:8]		OUT[99:96]		OUT[91:88]	P13_IOCR8
	P13.[15:12]		OUT[103:100]		OUT[95:92]	P13_IOCR12
Port 14	P14.[3:0]		OUT[99:96]		OUT[99:96]	P14_IOCR0
	P14.[7:4]		OUT[103:100]		OUT[103:100]	P14_IOCR4
	P14.[11:8]		OUT[107:104]		OUT[107:104]	P14_IOCR8
	P14.[15:12]		OUT[111:108]		OUT[111:108]	P14_IOCR12

1) Out[89] is not available as output on a pin.

2) Out[92] is not available as output on a pin.

3) There is a special connection provided for GPTA0/GPTA1 input line IN0 (see **Page 22-295**) and IN1 (see **Page 22-296**).

4) OUT[65], OUT[76], and OUT[78] are only as output of GPTA0 available on a pin.

5) OUT[100] and OUT[102] are not connected to this port.

Bit field PCx (x is the number of a port line) in the input/output control register IOCRy (y is the number of the first port line controlled by an IOCR) must be programmed according **Table 22-27**.

Note: Depending on the pad type (e.g. A2), not all PC codes are implemented

Table 22-27 PCx Coding

PCx[3:0]	I/O	Output Characteristics	Selected Pull-up/Pull-down/ Selected Output Function	Comment
$0X00_B$	Input	–	No pull device connected	Applicable for all Ports
$0X01_B$			Pull-down device connected	
$0X10_B$ 1)			Pull-up device connected	
$0X11_B$			No pull device connected	
1000_B	Output	Push-pull	General purpose output selected	All Ports
1001_B			GPTA®v5 (e.g. GPTA0) output (ALT1) selected	
1010_B			GPTA®v5 (e.g. GPTA1) output (ALT2) selected	
1011_B			GPTA®v5 (e.g. LTCA2) output (ALT3) selected	
1100_B		Open-drain	General purpose output selected	
1101_B			GPTA®v5 (e.g. GPTA0) output (ALT1) selected	
1110_B			GPTA®v5 (e.g. GPTA1) output (ALT2) selected	
1111_B			GPTA®v5 (e.g. LTCA2) output (ALT3) selected	

1) This bit field value is default after reset.

A port line that is programmed as input can be used by the GPTA0 or other units simultaneously as input.

Port lines selected as GPTA0/GPTA1/LTCA2 output are forced to a 0 level if the related multiplexer array in the I/O Line Sharing Block is disabled or if a reserved combination of an OMGn value is selected. Therefore, no glitches and spikes can occur during the programming of the related multiplexer array.

22.7.3.3 Pad Driver Characteristics Selection

The output driver strength and the slew rate of GPTA®v5 output lines is controlled by 3-bit wide PDX bit fields located in the pad driver mode registers PDR. These bit fields determine the driver strength and the slew rate for a group of port output lines.**Table 22-28** shows which of the pad driver register bit field is related to a specific GPTA0/GPTA1/LTCA2 unit I/O line group.

Table 22-28 PDR Assignment for GPTA®v5 Port Lines

Port	Pad Class	PDR Register PDx Bit Field	Controlled Port Lines	Related GPTA0 and GPTA1 Output Lines	Related LTCA2 Output Lines
Port 0	A1	P0_PDR.PD0	P0.[7:0]	OUT[63:56]	OUT[87:80]
Port 1	A1	P1_PDR.PD1	P1.[15:13]	OUT[71:69]	OUT[95:93]
	A1	P1_PDR.PD1	P1.[11:10]; P1.8	OUT[67:66]; OUT[64]	OUT[91:90], OUT[88]
	A2	P1_PDR.PDMLI0	P1.9	OUT[65]1)	
	A2	P1_PDR.PDEXTCLK	P1.12	OUT[68]	
Port 2	A1	P2_PDR.PD1	P2.[15:8]	OUT[7:0]	OUT[7:0]
Port 3	A1	P3_PDR.PD0	P3.[7:0]	OUT[15:8]	OUT[15:8]
	A1	P3_PDR.PD1	P3.[15:8]	OUT[23:16]	OUT[23:16]
Port 4	A2	P4_PDR.PD0	P4.[7:0]	OUT[31:24]	OUT[31:24]
	A1	P4_PDR.PD1	P4.[15:8]	OUT[39:32]	OUT[7:0]
Port 5	A2	P5_PDR.PDASC0	P5.[1:0]	OUT[73:72]	OUT[97:96]
	A2	P5_PDR.PDASC1	P5.[3:2]	OUT[75:74]	OUT[99:98]
	A2	P5_PDR.PDMSC0	P5.[5:4]	OUT[77:76]1)	OUT[101:100]
	A2	P5_PDR.PDMSC1	P5.[7:6]	OUT[79:78]1)	OUT[103:102]
	F2)	P5_PDR.PD2	P5.[9:8]	OUT[81:80]	OUT[105:104]
	F2)	P5_PDR.PD2	P5.[11:10]	OUT[83:82]	OUT[107:106]
	F2)	P5_PDR.PD3	P5.[13:12]	OUT[85:84]	OUT[109:108]
	F2)	P5_PDR.PD3	P5.[15:14]	OUT[87:86]	OUT[111:110]

Table 22-28 PDR Assignment for GPTA®v5 Port Lines (cont'd)

Port	Pad Class	PDR Register PDx Bit Field	Controlled Port Lines	Related GPTA0 and GPTA1 Output Lines	Related LTCA2 Output Lines
Port 8	A1	P8_PDR.PD0	P8.[7:6]; P8.4; P8.1	OUT[47:46]; OUT[44]; OUT[41]	
	A2	P8_PDR.PDMLI1	P8.5; P8.[3:2]; P8.0	OUT[45]; OUT[43:42]; OUT[40]	
Port 9	A2	P9_PDR.PDMSC0	P9.[7:4]	OUT[55:42]	
	A2	P9_PDR.PDMSC1	P9.[3:0]	OUT[51:48]	
Port 13	B1	P13_PDR.PD0	P13.[3:0]	OUT[91:88]	OUT[83:80]
	B1	P13_PDR.PD1	P13.[7:4]	OUT[95:92]	OUT[87:84]
	B1	P13_PDR.PD2	P13.[11:8]	OUT[99:96]	OUT[91:88]
	B1	P13_PDR.PD1	P13.[15:12]	OUT[103:100]	OUT[95:92]
Port 14	B1	P14_PDR.PD0	P14.[3:0]	OUT[99:96]	OUT[99:96]
	B1	P14_PDR.PD1	P14.[7:4]	OUT[103:100]	OUT[103:100]
	B1	P14_PDR.PD2	P14.[11:8]	OUT[107:104]	OUT[107:104]
	B1	P14_PDR.PD3	P14.[15:12]	OUT[111:108]	OUT[111:108]

1) OUT[65], OUT[76], and OUT[78] are only as output of GPTA0 available on a pin.

2) The following constraint applies to an LVDS pair used in CMOS mode (pad type F): only one pin of a pair should be used as output, the other should be used as input, or both pins should be used as inputs. Using both pins as outputs is not recommended because of the higher crosstalk between them.

The coding of the PDx bit field combinations is shown **Table 22-29**.

Table 22-29 Pad Driver Mode Selection

Pad Class	PDx.2	PDx.1	PDx.0	Functionality
A1	X	X	0	Medium driver selected
			1	Weak driver selected

Table 22-29 Pad Driver Mode Selection (cont'd)

Pad Class	PDx.2	PDx.1	PDx.0	Functionality
A2/B1/B2	0	0	0	Strong driver, sharp edge selected
	0	0	1	Strong driver, medium edge selected
	0	1	0	Strong driver, soft edge selected
	0	1	1	Weak driver selected
	1	0	0	Medium driver selected
	1	0	1	
	1	1	0	
	1	1	1	Weak driver selected
F	0	X	X	CMOS driver selected
	1			LVDS driver selected

22.7.3.4 Emergency Control of GPTA®v5 Output Ports Lines

Port lines connected to GPTA0/GPTA1/LTCA2 unit output pins can be selectively switched into an Emergency Mode. In this mode, GPTA0/GPTA1/LTCA2 unit output pins react immediately to an active input signal P10.1 (HWCFG1) and drive a logic level that has been programmed in the port output register. As a result, in Emergency Mode a GPTA0/GPTA1/LTCA2 unit output pin drives a predefined value instead of the corresponding logic level that is provided on the related GPTA0/GPTA1/LTCA2 unit output line.

All GPTA®v5 pins at Port 0, Port 1, Port 2, Port 3, Port 4, Port 8, Port 9, Port 13, and Port 14 are connected to one common emergency stop signal that is generated in the System Control Unit of the TC1797. More details about the generation of this emergency stop signal are described in the "System Control Unit" chapter of the TC1797 System Units User´s Manual.

The emergency stop signal always controls 8-bit groups of port lines. The enable function is controlled for each pin by bits ENy (y = number of port line) which are located in the Px_ESR (x = port number) registers. When the emergency stop signal generated in the SCU becomes active and bit Px_ESR.ENy set, output line Px.y is set to the value of register Px_OUT.Py (emergency enabled). Output Px.y is not affected by the emergency stop signal when bit Px_OUT.Py is reset (emergency disabled).

When the emergency stop signal is released, Pin x.y is switched back to the previously selected GPTA®v5 output function without reprogramming the related port registers.

The emergency stop enable bits ENy are only implemented for output pins at Port 0, Port 1, Port 2, Port 3, Port 4, Port 5, Port5, Port 8, Port 9, Port 13, and Port 14 that can be connected to the GPTA0/GPTA1/LTCA2 unit.

Table 22-30 Emergency Control for GPTA®v5 Port Output Lines

Port	ESR Register	ESR Enable Bits	GPTA®v5 Output Lines	LTCA Output Lines
Port 0	P0_ESR	EN[7:0]	OUT[63:56]	OUT[87:80]
Port 1	P1_ESR	EN[15:8]	OUT[71:64]	OUT[95:88]
Port 2	P2_ESR	EN[15:8]	OUT[7:0]	OUT[7:0]
Port 3	P3_ESR	EN[15:0]	OUT[23:8]	OUT[23:8]
Port 4	P4_ESR	EN[15:0]	OUT[39:24]	OUT[31:24], OUT[7:0]
Port 5	P5_ESR	EN[15:0]	OUT[87:72]	OUT[111:96]
Port 8	P8_ESR	EN[7:0]	OUT[47:40]	
Port 9	P9_ESR	EN[7:0]	OUT[55:48]	
Port 13	P13_ESR	EN[15:0]	OUT[103:88]	OUT[95:80]
Port 14	P14_ESR	EN[15:0]	OUT[111:96]	OUT[111:96]

22.7.4 On-Chip Connections

This section describes all on-chip interconnections of the GPTA0/GPTA1/LTCA2 units except the connections to I/O ports (see **Section 22.7.3**).

22.7.4.1 Clock Bus Connections

The clock bus signals generated in the GPTA0 and GPTA1 clock distribution cells and PLL clocks are interconnected between the GPTA0/GPTA1/LTCA2 units as shown in **Figure 22-96**.

The GPTA0 clock bus drives Global Timers (GTs), Global Timer Cells (GTCs), and Local Timer Cells (LTCs) of the GPTA0 unit **and** the LTCs of the LTCA2 unit. The GPTA1 clock bus drives its GTs, GTCs, and LTCs.

Each GPTA0 and GPTA1 clock distribution cells has two pairs for PLL clock inputs, an input pair for the local PLL clocks and an input pair for unit-external PLL clocks. The GPTA0 and GPTA1 units are capable to use the PLL clocks from each other.

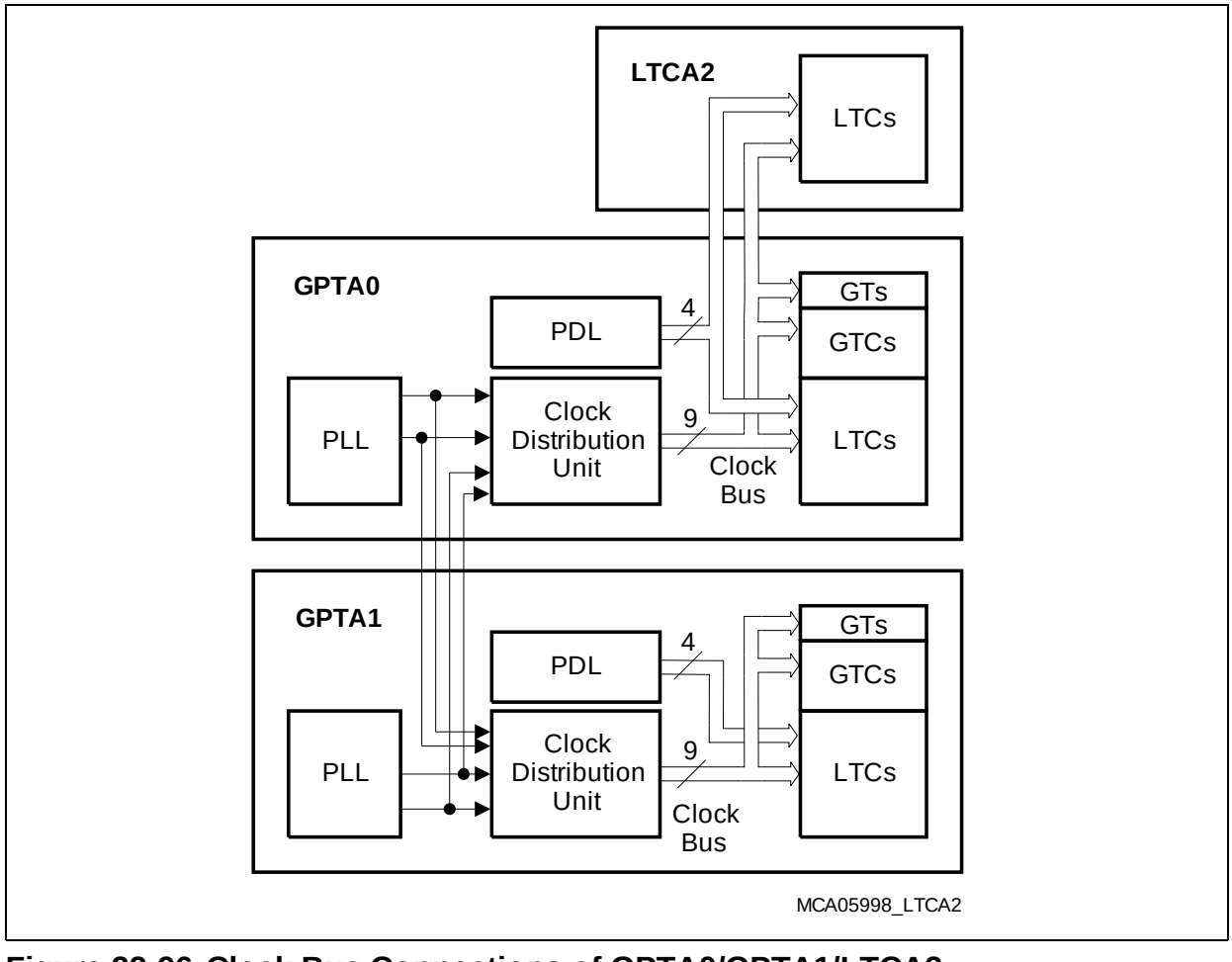


Figure 22-96 Clock Bus Connections of GPTA0/GPTA1/LTCA2

22.7.4.2 MSC Controller Connections

The MSC interfaces (MSC0 and MSC1) provide a serial communication link typically used to connect power switches or other peripheral devices. Each output multiplexers of GPTA0/GPTA1 generate 7 × 8 = 56 output lines OUT[111:56] grouped into seven output groups. Each output multiplexers of LTCA2 generate 4 × 8 = 32 output lines OUT[111:80] grouped into four output groups. All these output lines are wired to 32 MSC0 inputs. Up to 32 MSC0 and 24 MSC1 output extension lines (bits) can be connected to the GPTA®v5. **Figure 22-97** shows the interconnections among the MSC0/1 modules and the GPTA®v5 units. The source for each line of the ALTINL/ALTINH MSC input buses can be selected via a multiplexer connected either to GPTA0 output OUTx(x = 56-111), GPTA1 output OUTx(x = 56-111), LTCA2 output OUTx(x = 80-111).

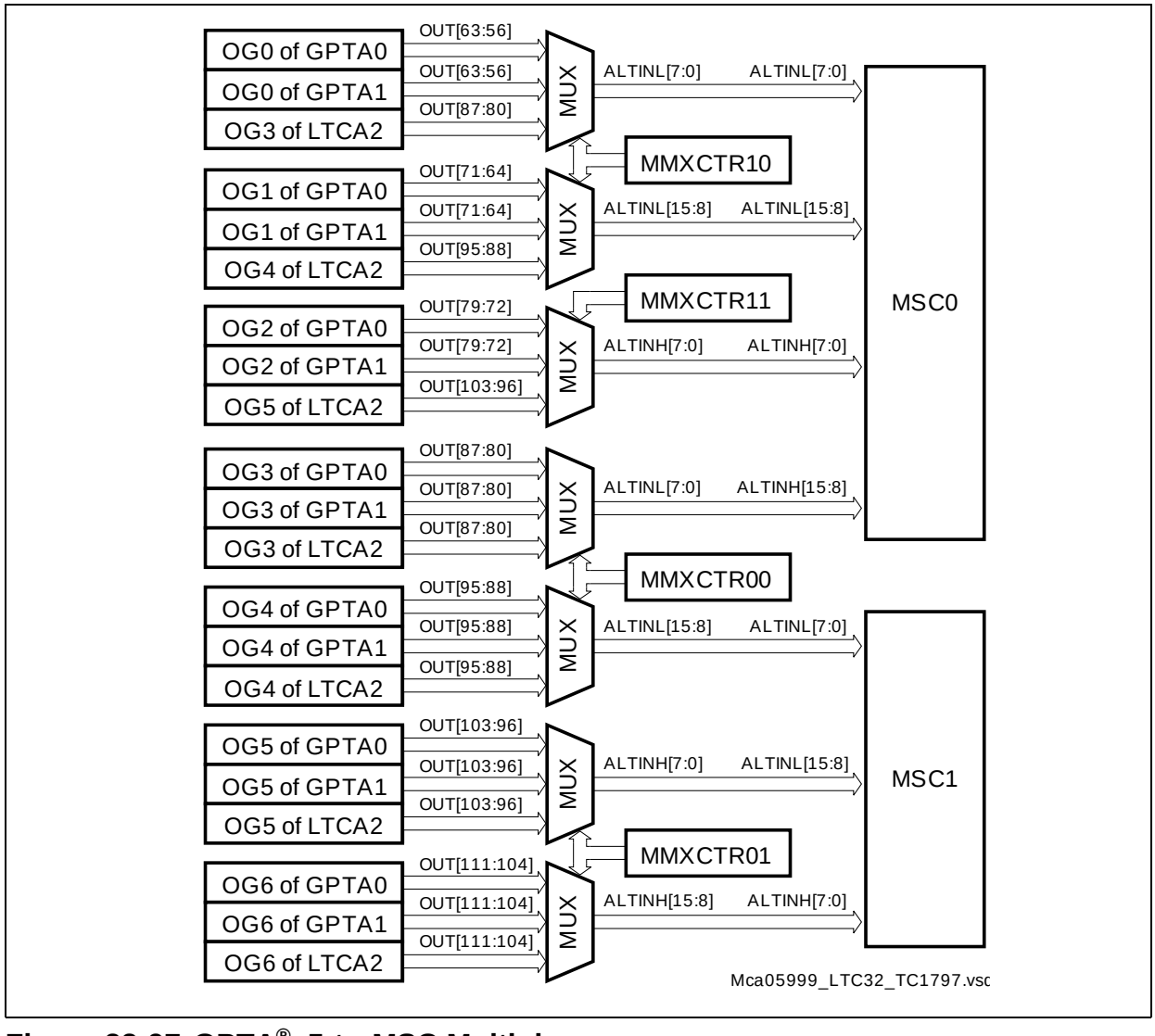


Figure 22-97 GPTA®v5-to-MSC Multiplexer

The multiplexer selection is controlled by four multiplexer control registers that are logically assigned to the GPTA0 unit address range (but described in this section).

Note: Note that eight ALTINH inputs (ALTINH[15:8]) of the MSC1 module are not connected.

Table 22-31 and **Table 22-32** shows the GPTA®v5-to-MSC interconnection assignment.

*Note: **Table 22-31** and **Table 22-32** also shows the assignment of the GPTA0/GPTA1/LTCA2 unit's seven OGx output group lines OGx.y to the output signals OUT[111:56].*

Table 22-31 GPTA0/GPTA1/LTCA2 to MSC0 Interconnection Assignment

MSC0 Input Line	Assigned GPTA0/ GPTA1 Output Line	Assigned LTCA2 Output Line	MSC0 Input Line	Assigned GPTA0/GPTA1 Output Line	Assigned LTCA2 Output Line
ALTINL.0	OUT56 / OG0.0	OUT80 / OG3.0	ALTINH.0	OUT72 / OG2.0	OUT96 / OG5.0
ALTINL.1	OUT57 / OG0.1	OUT81 / OG3.1	ALTINH.1	OUT73 / OG2.1	OUT97 / OG5.1
ALTINL.2	OUT58 / OG0.2	OUT82 / OG3.2	ALTINH.2	OUT74 / OG2.2	OUT98 / OG5.2
ALTINL.3	OUT59 / OG0.3	OUT83 / OG3.3	ALTINH.3	OUT75 / OG2.3	OUT99 / OG5.3
ALTINL.4	OUT60 / OG0.4	OUT84 / OG3.4	ALTINH.4	OUT76 / OG2.4	OUT100 / OG5.4
ALTINL.5	OUT61 / OG0.5	OUT85 / OG3.5	ALTINH.5	OUT77 / OG2.5	OUT101 / OG5.5
ALTINL.6	OUT62 / OG0.6	OUT86 / OG3.6	ALTINH.6	OUT78 / OG2.6	OUT102 / OG5.6
ALTINL.7	OUT63 / OG0.7	OUT87 / OG3.7	ALTINH.7	OUT79 / OG2.7	OUT103 / OG5.7
ALTINL.8	OUT64 / OG1.0	OUT88 / OG4.0	ALTINH.8	OUT80 / OG3.0	OUT104 / OG6.0
ALTINL.9	OUT65 / OG1.1	OUT89 / OG4.1	ALTINH.9	OUT81 / OG3.1	OUT105 / OG6.1
ALTINL.10	OUT66 / OG1.2	OUT90 / OG4.2	ALTINH.10	OUT82 / OG3.2	OUT106 / OG6.2

Table 22-31 GPTA0/GPTA1/LTCA2 to MSC0 Interconnection Assignment (cont'd)

MSC0 Input Line	Assigned GPTA0/ GPTA1 Output Line	Assigned LTCA2 Output Line	MSC0 Input Line	Assigned GPTA0/GPTA1 Output Line	Assigned LTCA2 Output Line
ALTINL.11	OUT67 / OG1.3	OUT91 / OG4.3	ALTINH.11	OUT83 / OG3.3	OUT107 / OG6.3
ALTINL.12	OUT68 / OG1.4	OUT92 / OG4.4	ALTINH.12	OUT84 / OG3.4	OUT108 / OG6.4
ALTINL.13	OUT69 / OG1.5	OUT93 / OG4.5	ALTINH.13	OUT85 / OG3.5	OUT109 / OG6.5
ALTINL.14	OUT56 / OG2.0	OUT94 / OG5.0	ALTINH.14	OUT86 / OG3.6	OUT110 / OG6.6
ALTINL.15	OUT57 / OG2.1	OUT95 / OG5.1	ALTINH.15	OUT87 / OG3.7	OUT111 / OG6.7

Table 22-32 GPTA0/GPTA1/LTCA2 to MSC1 Interconnection Assignment

MSC1 Input Line	Assigned GPTA0/ GPTA1/LTCA2 Output Line	MSC1 Input Line	Assigned GPTA0/ GPTA1/LTCA2 Output Line
ALTINL.0	OUT88 / OG4.0	ALTINH.0	OUT104 / OG6.0
ALTINL.1	OUT89 / OG4.1	ALTINH.1	OUT105 / OG6.1
ALTINL.2	OUT90 / OG4.2	ALTINH.2	OUT106 / OG6.2
ALTINL.3	OUT91 / OG4.3	ALTINH.3	OUT107 / OG6.3
ALTINL.4	OUT92 / OG4.4	ALTINH.4	OUT108 / OG6.4
ALTINL.5	OUT93 / OG4.5	ALTINH.5	OUT109 / OG6.5
ALTINL.6	OUT94 / OG4.6	ALTINH.6	OUT110 / OG6.6
ALTINL.7	OUT95 / OG4.7	ALTINH.7	OUT111 / OG6.7
ALTINL.8	OUT96 / OG5.0	ALTINH.8	
ALTINL.9	OUT97 / OG5.1	ALTINH.9	
ALTINL.10	OUT98 / OG5.2	ALTINH.10	
ALTINL.11	OUT99 / OG5.3	ALTINH.11	
ALTINL.12	OUT100 / OG5.4	ALTINH.12	
ALTINL.13	OUT101 / OG5.5	ALTINH.13	

Table 22-32 GPTA0/GPTA1/LTCA2 to MSC1 Interconnection Assignment (cont'd)

MSC1 Input Line	Assigned GPTA0/ GPTA1/LTCA2 Output Line	MSC1 Input Line	Assigned GPTA0/ GPTA1/LTCA2 Output Line
ALTINL.14	OUT102 / OG5.6	ALTINH.14	
ALTINL.15	OUT103 / OG5.7	ALTINH.15	

GPTA®v5-to-MSC Multiplexer Control Registers

The following registers are required for GPTA®v5-to-MSC multiplexer control:

- **GPTA0_MMXCTR00** controls the interconnections of GPTA®v5 Units to the MSC0 ALTINH[15:8] and MSC1 ALTINL[7:0] inputs.
- **GPTA0_MMXCTR01** controls the interconnections of GPTA®v5 Units to the MSC1 ALTINL[15:8] inputs and MSC1 ALTINH[7:0] inputs.
- **GPTA0_MMXCTR10** controls the interconnections of GPTA®v5 Units to the MSC0 ALTINL[15:0] inputs.
- **GPTA0_MMXCTR11** controls the interconnections of GPTA®v5 Units to the MSC0 ALTINH[7:0] inputs.

For each of the ALTINL/ALTINH inputs of the MSC, the 2-bit bit fields in these registers determine which unit output is selected.

GPTA0_MMXCTR00
GPTA-to-MSC Multiplexer Control Register 00

(700$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MUX 15		MUX 14		MUX 13		MUX 12		MUX 11		MUX 10		MUX 9		MUX 8		MUX 7		MUX 6		MUX 5		MUX 4		MUX 3		MUX 2		MUX 1		MUX 0	
rw		rw		rw		rw		rw		rw		rw		rw		rw		rw		rw		rw		rw		rw		rw		rw	

Field	Bits	Type	Description
MUXn (n = 0-7)	[2*n+1:2*n]	rw	**Multiplexer Control for MSC0 Inputs ALTINH.(n+8)** 00_B GPTA0 output OUT[80+n] selected and connected to MSC0 ALTINH.(n+8) 01_B GPTA1 output OUT[80+n] selected and connected to MSC0 ALTINH.(n+8) 10_B LTCA2 output OUT[80+n] selected and connected to MSC0 ALTINH.(n+8) 11_B Reserved
MUXn (n = 8-15)	[2*n+1:2*n]	rw	**Multiplexer Control for MSC1 Inputs ALTINL(n-8)** 00_B GPTA0 output OUT[80+n] selected and connected to ALTINL.(n-8) 01_B GPTA1 output OUT[80+n] selected and connected to MSC1 ALTINL.(n-8) 10_B LTCA2 output OUT[80+n] selected and connected to MSC1 ALTINL.(n-8) 11_B Reserved

GPTA0_MMXCTR01
GPTA-to-MSC Multiplexer Control Register 01

(704$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MUX 15		MUX 14		MUX 13		MUX 12		MUX 11		MUX 10		MUX 9		MUX 8		MUX 7		MUX 6		MUX 5		MUX 4		MUX 3		MUX 2		MUX 1		MUX 0	
rw		rw		rw		rw		rw		rw		rw		rw		rw		rw		rw		rw		rw		rw		rw		rw	

Field	Bits	Type	Description
MUXn (n = 0-7)	[2*n+1:2*n]	rw	**Multiplexer Control for MSC1 Inputs ALTINL.(n+8)** 00_B GPTA0 output OUT[96+n] selected and connected to MSC1 ALTINL.(n+8) 01_B GPTA1 output OUT[96+n] selected and connected to MSC1 ALTINL.(n+8) 10_B LTCA2 output OUT[96+n] selected and connected to MSC1 ALTINL.(n+8) 11_B Reserved
MUXn (n = 8-15)	[2*n+1:2*n]	rw	**Multiplexer Control for MSC1 Inputs ALTINH.(n-8)** 00_B GPTA0 output OUT[96+n] selected and connected to MSC1 ALTINH.(n-8) 01_B GPTA1 output OUT[96+n] selected and connected to MSC1 ALTINH.(n-8) 10_B LTCA2 output OUT[96+n] selected and connected to MSC1 ALTINH.(n-8) 11_B Reserved

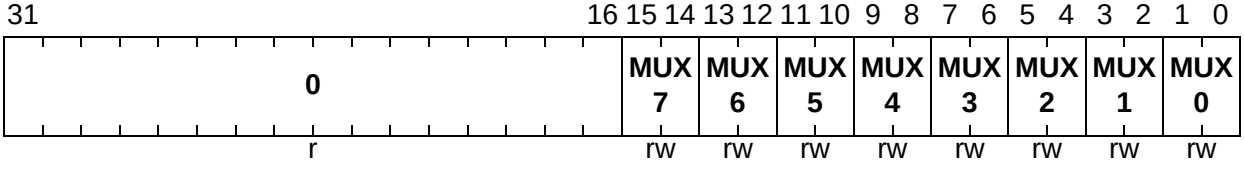
GPTA0_MMXCTR10
GPTA-to-MSC Multiplexer Control Register 10
(708_H) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MUX 15		**MUX 14**		**MUX 13**		**MUX 12**		**MUX 11**		**MUX 10**		**MUX 9**		**MUX 8**		**MUX 7**		**MUX 6**		**MUX 5**		**MUX 4**		**MUX 3**		**MUX 2**		**MUX 1**		**MUX 0**	
rw		rw		rw		rw		rw		rw		rw		rw		rw		rw		rw		rw		rw		rw		rw		rw	

Field	Bits	Type	Description
MUXn (n = 0-15)	[2*n+1:2*n]	rw	**Multiplexer Control for MSC0 Inputs ALTINL.n** 00_B GPTA0 output OUT[56+n] selected and connected to MSC0 ALTINL.n 01_B GPTA1 output OUT[56+n] selected and connected to MSC0 ALTINL.n 10_B LTCA2 output OUT[80+n] selected and connected to MSC0 ALTINL.n 11_B Reserved

GPTA0_MMXCTR11
GPTA-to-MSC Multiplexer Control Register 11

(70C$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
MUXn (n = 0-7)	[2*n+1:2*n]	rw	**Multiplexer Control for MSC0 Inputs ALTINH.n** 00_B GPTA0 output OUT[72+n] selected and connected to MSC0 ALTINH.n 01_B GPTA1 output OUT[72+n] selected and connected to MSC0 ALTINH.n 10_B LTCA2 output OUT[96+n] selected and connected to MSC0 ALTINH.n 11_B Reserved
0	[31:15]	r	**Reserved** Read as 0; should be written with 0.

22.7.4.3 Connections to SCU, MultiCAN, FADC, DMA, Ports

The GPTA0/GPTA1/LTCA2 units of the TC1797 GPTA®v5 module have several on-chip interconnections with the SCU, MultiCAN, FADC, DMA modules. **Figure 22-98** shows these interconnections.

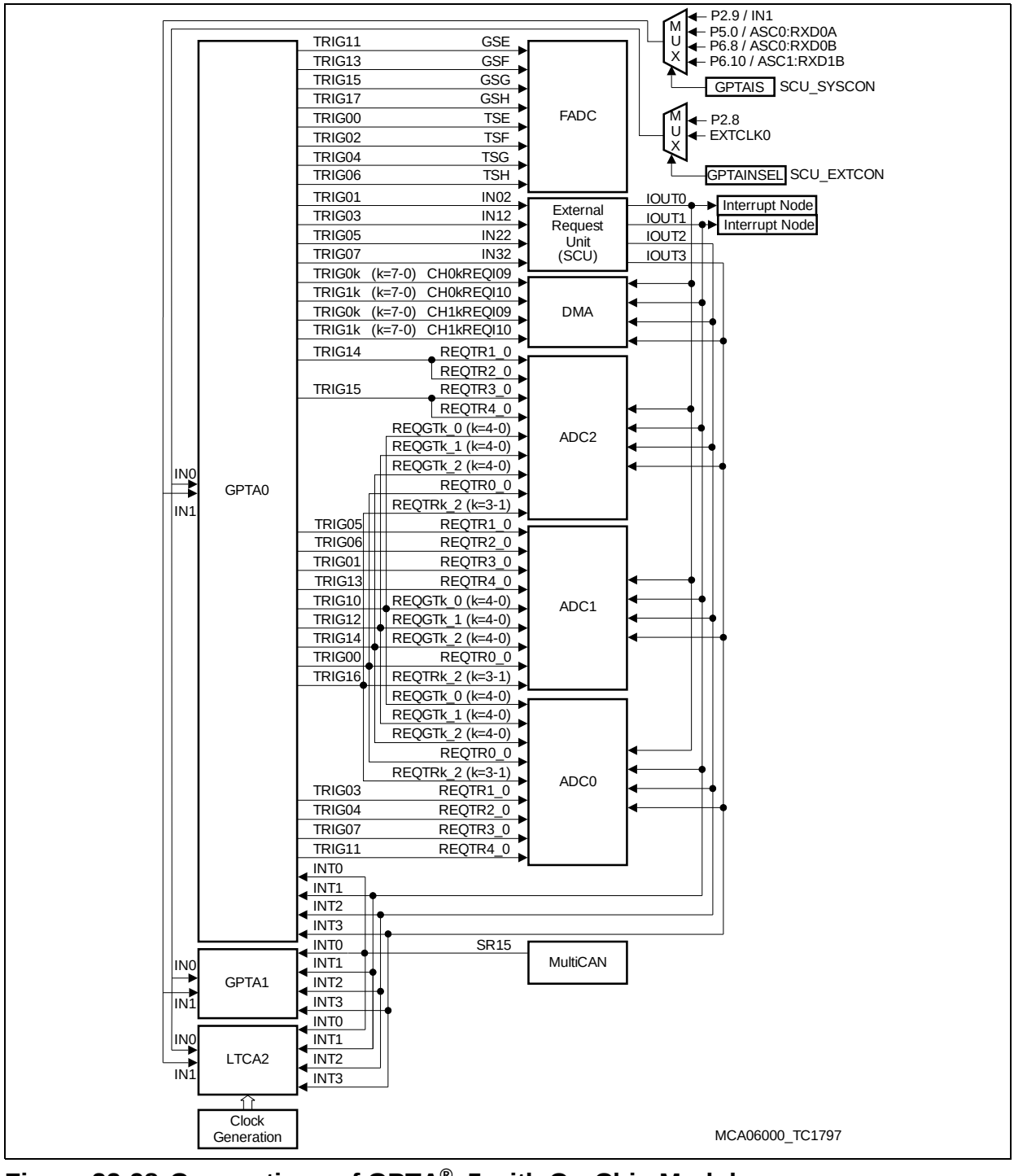


Figure 22-98 Connections of GPTA®v5 with On-Chip Modules

System Control Unit

The SCU contains the external request unit (ERU), which is especially responsible for controlling requests coming from the MSC modules, port pins, or from the GPTA0/GPTA1/LTCA2 unit and passing these request to AD converters, DMA controller, ERAY communication controller or interrupt nodes.

MultiCAN Connections

The MultiCAN controller has the following connections to the GPTA0/GPTA1/LTCA2 unit:

- MultiCAN service request output SR15 is connected to the INT0 input of GPTA0/GPTA1/LTCA2.

DMA Controller

As shown in **Figure 22-98**, eight GPTA0 on-chip trigger and gating output lines are connected as trigger input signals to the DMA request inputs. Furthermore the external request unit generates four DMA request output signals (IOUT[3:0]) that can be activated via port pins, the MSC clock outputs, or the four GPTA®v5 output lines. Three of these four DMA request output signals are connected to the GPTA0/GPTA1/LTCA2 internal inputs INT[3:1]. These connections allow, for example, GTC or LTC events in the GPTA®v5 units to be triggered by a request coming from a port pin or from the MSC clock.

ADC Connections

As shown in **Figure 22-98**, for each ADC nine GPTA0 on-chip trigger and gating output lines are connected as trigger input signals or gating input signals to the channel trigger logic of the ADC. Thus dedicated GPTA0 outputs can generate trigger events or act as gating signals for ADC channels. Furthermore the external request unit generates two ADC conversion trigger signals (IOUT[3:2]) and two ADC conversion gating signals (PDOUT[3:2]) that can be activated each via port pins, the MSC clock outputs, or two GPTA®v5 output lines.

FADC Connections

As shown in **Figure 22-98**, eight GPTA0 on-chip trigger and gating output lines are connected as trigger input signals or gating input signals to the channel trigger logic of the FADC. Thus dedicated GPTA0 outputs can generate trigger events or act as gating signals for FADC channels.

Port Connections of Input IN0

The common input line IN0 of the GPTA0/GPTA1/LTCA2 unit is connected to the output of a 2-to-1 multiplexer. This multiplexer is controlled by bit field

SCU_EXTCON.GPTAINSEL and allows the common GPTA0/GPTA1/LTCA2 input IN0 to be connected to one out of two input lines. This feature especially allows the number of clock of the PLL (to determine clock stability) to be measured by GTs (Global Timers) or FPC0 (Filter and Prescaler Cell) of the GPTA0/GPTA1.

Table 22-33 GPTA0 Input Line IN0 Connections

SCU_EXTCON. GPTAINSEL	GPTA0/GPTA1/LTCA2 Input IN0 Connected to
00_B	P2.9 / IN0 (default after reset)
01_B	SCU: EXTCLK0

Port Connections of Input IN1

The common input line IN1 of the GPTA0/GPTA1/LTCA2 unit is connected to the output of a 4-to-1 multiplexer. This multiplexer is controlled by bit field SCU_SYSCON.GPTAIS and allows the common GPTA0/GPTA1/LTCA2 input IN1 to be connected to one out of four port input lines. This feature especially allows the baud rates of an ASC0 or ASC1 receiver input signal to be measured by timers of the GPTA0/GPTA1/LTCA2.

Table 22-34 GPTA0 Input Line IN1 Connections

SCU_SYSCON. GPTAIS	GPTA0/GPTA1/LTCA2 Input IN1 Connected to
00_B	P2.9 / IN1 (default after reset)
01_B	P5.0 / ASC0: RXD0A
10_B	P6.8 ASC0: RXD0B
11_B	P6.10 / ASC1: RXD1B

22.7.5 Module Clock Generation

As shown in **Figure 22-99**, the clock signals for the GPTA0/GPTA1/LTCA2 units are generated and controlled by one clock generation circuitry. This clock generation circuitry is responsible for the enable/disable control, the clock frequency adjustment, and the debug clock control. The circuitry includes the following registers:

- **Clock Control Register GPTA0_CLC** (see **Page 22-300**), responsible for the generation of the control clock f_{CLC}that is used by each of the units.
- **Fractional Divider Register GPTA0_FDR** (see **Page 22-309**), responsible for the frequency control of the module timer clock f_{GPTA}.
- **Clock Enable/Disable Control Register GPTA0_EDCTR** (see **Page 22-310**), responsible for the enable/disable control of the different units as clocks f_{GPTA0},/f_{GPTA1}/f_{LTCA2}, and for the run control for the Global Timers in GPTA0 and GPTA1.

- **Debug Clock Control Register GPTA0_DBGCTR** (see **Page 22-312**), responsible for the module timer clock control in Debug Mode.

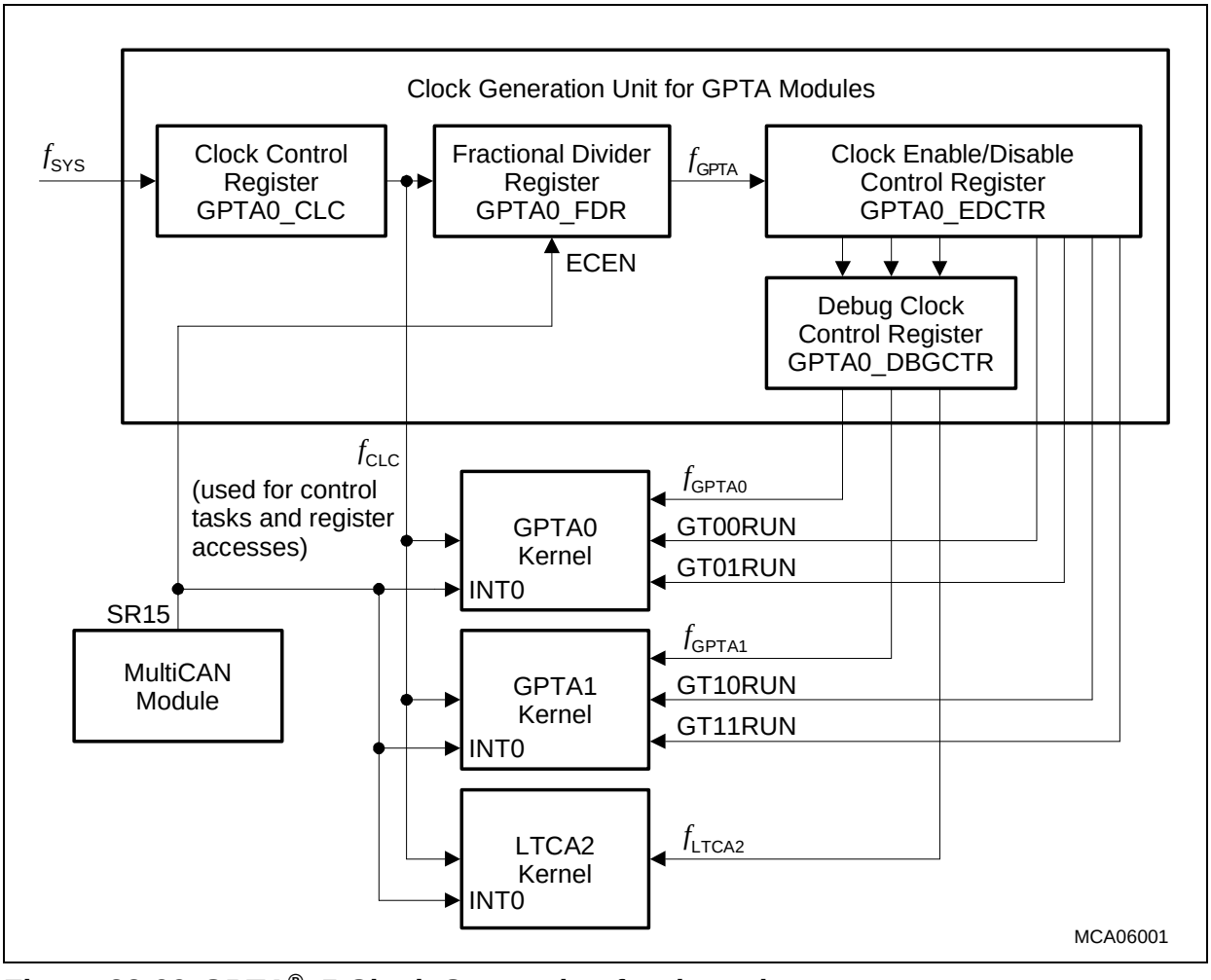


Figure 22-99 GPTA®v5 Clock Generation for the units

Note: Registers GPTA0_CLC, GPTA0_FDR, GPTA0_EDCTR and GPTA0_DBGCTR are located in the address space of GPTA0.

Module clock and CLC clock are both derived from the system clock f_{SYS}. The CLC register provides the f_{CLC} clock which acts as clock input for the fractional divider and control logic. The CLC clock f_{CLC} is typically used by a peripheral module for clocking its FPI Bus interface and registers, while the module clock f_{MOD} is dedicated for kernel operation or timer clocks. The output signal RST_EXT_DIV makes it possible to enable/disable external divider stages which are connected to the module clock f_{MOD}. The fractional divider divides the f_{CLC} either by the factor 1/n or by a fraction of n/1024 for any value of n from 0 to 1023.

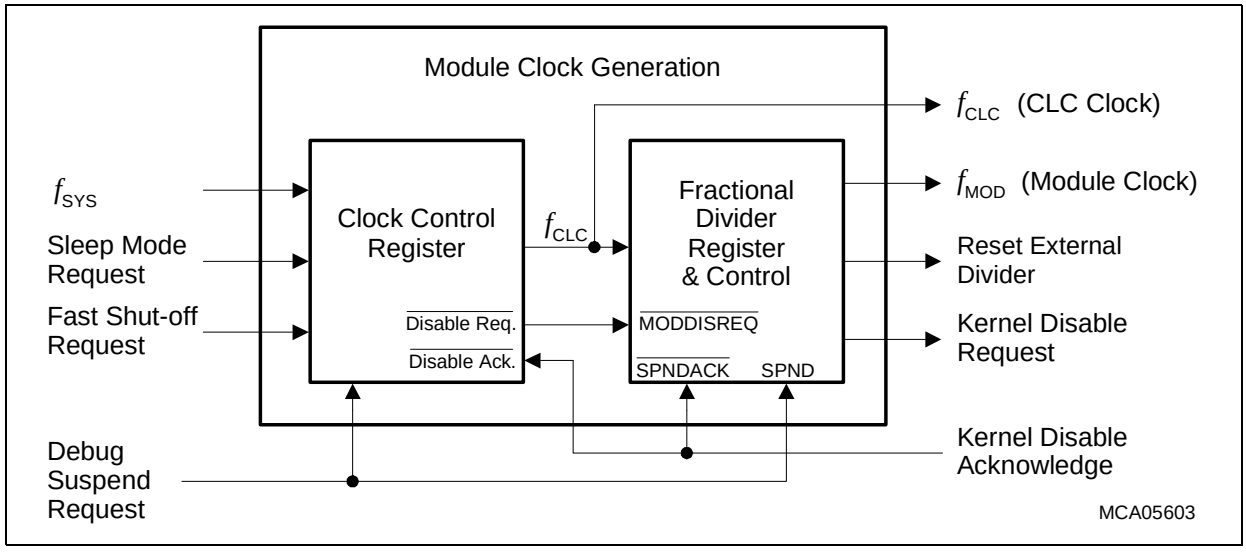


Figure 22-100Details on Module Clock Generation

Furthermore, the module clock generation circuitry handles the sleep mode request signal, the fast shut-off request signal, and the debug suspend request signal.

The GPTA®v5 module control clock f_{CLC} is used inside the GPTA®v5 module kernels for control purposes such as clocking of control logic and register operations. The frequency of f_{CLC} is identical to the system clock frequency f_{SYS}. The clock control registers GPTA0_CLC make it possible to enable/disable f_{CLC} under certain conditions.

The separate GPTA®v5 unit clocks f_{GPTA0}, f_{GPTA1}, f_{LTCA2} are used inside the GPTA®v5 units as input clocks for the timers. All unit clocks have the same frequency as f_{GPTA} (as selected through register GPTA0_FDR) and can be enabled/disabled separately each through register GPTA0_ECDTR.

Note: If f_{GPTA0}, f_{GPTA1}, f_{LTCA2} are disabled by the enable bits in register GPTA0_ECDTR, f_{CLC} keeps running. In this case, that means that register accesses to the GPTA®v5 units are possible.

The frequency of f_{GPTA} is defined by:

$$f_{GPTA} = f_{SYS} \times \frac{1}{n} \text{ with n = 1024 - FDR.STEP or} \quad (22.6)$$

$$f_{GPTA} = f_{SYS} \times \frac{n}{1024} \text{ with n = 0-1023} \quad (22.7)$$

Note: The upper formula applies to normal divider mode of the fractional divider (GPTA0_FDR.DM = 01_B). The lower formula applies to fractional divider mode (GPTA0_FDR.DM = 10_B).

The debug clock control register additionally makes it possible to control the timer clocks f_{GPTA0}, f_{GPTA1}, f_{LTCA2} for debug purposes on basis of a clock counter.

If the debug clock feature is enabled (GPTA0_DBGCTR.DBGCEN = 1) and bit GPTA0_DBGCTR.DBGCST is set, the timer clocks f_{GPTA0}, f_{GPTA1}, f_{LTCA2} will be activated in parallel for as many clock cycles as have been programmed into bit field GPTA0_DBGCTR.CLKCNT. When the debug clock feature becomes enabled, bit field CLKCNT counts down and stops counting at 0000_H. Bit DBGCST is again reset by hardware after the programmed number of clock pulses has been issued. This feature makes it possible to single step the GPTA®v5 units with a programmable timer clock granularity.

Note: The GPTA®v5 module is disabled after reset. In general, after reset, the GPTA®v5 module control clock f_{CLC} must be switched on (writing to register GPTA0_CLC) before the frequency of the GPTA®v5 module timer clock f_{GPTA} is defined (writing to register GPTA0_FDR).

22.7.5.1 Clock Control Registers

The clock control register makes it possible to control (enable/disable) the GPTA®v5 module control clock f_{CLC}. The clock signal f_{CLC} is used by the GPTA0/GPTA1/LTCA2 as a clock for internal control operations but not for timer purposes.

GPTA0_CLC
GPTA Clock Control Register (000_H) Reset Value: 0000 0003$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0										FSOE	SBWE	EDIS	SPEN	DISS	DISR
r										rw	w	rw	rw	r	rw

Field	Bits	Type	Description
DISR	0	rw	**GPTA®v5 Module Disable Request Bit** Used for enable/disable control of the GPTA®v5 module.
DISS	1	r	**GPTA®v5 Module Disable Status Bit** Bit indicates the current status of the GPTA®v5 module.
SPEN	2	rw	**GPTA®v5 Module Suspend Enable for OCDS** Used to enable the suspend mode.
EDIS	3	rw	**External Request Disable** Used to control the external clock disable request.
SBWE	4	w	**GPTA®v5 Module Suspend Bit Write Enable for OCDS** Determines whether SPEN and FSOE are write-protected.
FSOE	5	rw	**Fast Switch Off Enable** Used for fast clock switch off in suspend mode.
0	[31:6]	r	**Reserved** Read as 0; should be written with 0.

Note: After a hardware reset operation, the f_{CLC} clock is disabled (DISS set). Therefore, the GPTA®v5 module clock generation is completely disabled.

Note: In disabled state, no registers of GPTA®v5 module can be read or written except the GPTA_CLC register.

Module Enable/Disable Control

If the GPTA®v5 is not used at all by an application, it can be completely shut off by setting bit DISR in the GPTA0_CLC register.

The status bit GPTA0_CLC.DISS always indicates whether the GPTA®v5 is currently switched off (GPTA0_CLC.DISS = 1) or switched on (GPTA0_CLC.DISS = 0). The default state of a GPTA®v5 module after reset is “GPTA®v5 disabled” with GPTA0_CLC.DISS set.

Write operations to the registers of a disabled GPTA®v5 is not allowed. However, the GPTA0_CLC of a disabled GPTA®v5 can be written. An attempt to write to any of the other writable registers of a disabled GPTA®v5 module except GPTA0_CLC will cause the corresponding Bus Control Unit (BCU) to generate a bus error.

A read operation of registers of a disabled GPTA®v5 module is allowed and does not generate a bus error.

When a disabled GPTA®v5 module is switched on by writing an appropriate value to its GPTA0_CLC register (GPTA0_CLC.DISR = 0) the status bit GPTA0_CLC.DISS changes from 1 to 0. During the phase in which the GPTA®v5 module becomes active, any write access to corresponding GPTA®v5 module registers (when GPTA0_CLC.DISS is still set) will generate a bus error. Therefore, when enabling a disabled GPTA®v5 module, application software should check after activation of the GPTA®v5 module once (read back of the GPTA0_CLC.CLC register) to find out whether GPTA0_CLC.DISS is already reset, before a GPTA®v5 module register (including the GPTA0_CLC. register) will be written to.

Note: A read access occurring while a GPTA®v5 module is disabled is treated as a normal read access. This means, if a GPTA®v5 module register or a bit of it is cleared as a side-effect of a read access of an enabled GPTA®v5 module, it will not be cleared by this read access while the GPTA®v5 module is disabled.

Sleep Mode Control

The GPTA0_CLC.EDIS bit in the GPTA0_CLC register controls whether or not a GPTA®v5 module is stopped during sleep mode. If GPTA0_CLC.EDIS is 0 (default after reset), a sleep mode request can be recognized by the GPTA®v5 module and, when received, its clock is shut off.

If GPTA0_CLC.EDIS is set to 1, a sleep mode request is disregarded by the GPTA®v5 module and the GPTA®v5 module continues its operation.

Debug Suspend Mode Control

During emulation and debugging of TC1797 applications, the execution of an application program can be suspended. When an application is suspended, normal operation of the application’s program is halted, and the TC1797 begins (or resumes) executing a special debug monitor program. When the application is suspended, a suspend request signal

is generated by the TC1797 and sent to all modules. If bit GPTA0_CLC.SPEN is set to 1, the operation of the GPTA®v5 module is stopped when the suspend signal is asserted. If GPTA0_CLC.SPEN is set to 0, the module does not react to the suspend request signal but continues its normal operation. This feature allows each peripheral module to be adapted to the unique requirements of the application being debugged. Setting GPTA0_CLC.SPEN bits is usually performed by a debugger.

This feature is necessary because application requirements typically determine whether on-chip modules should be stopped or left running when an application is suspended for debugging. For example, a peripheral module that is controlling the motion of an external device through motors in most cases must not be stopped so as to prevent damage of the external device due to the loss of control through the peripheral. On the other hand, it makes sense to stop the system timer while the debugger is actively controlling the chip because it should only count the time when the user's application is running.

Note that it is never appropriate for application software to set the GPTA0_CLC.SPEN bit. The debug suspend mode should only be set by a debug software. To guard against application software accidently setting GPTA0_CLC.SPEN, bit GPTA0_CLC.SPEN is specially protected by the mask bit GPTA0_CLC.SBWE. The GPTA0_CLC.SPEN bit can only be written if, during the same write operation, GPTA0_CLC.SBWE is set, too. Application software should never set GPTA0_CLC.SBWE to 1. In this way, user software can not accidentally alter the value of the GPTA0_CLC.SPEN bit that has been set by a debugger.

Note: The operation of the Watchdog Timer is always automatically stopped in debug suspend mode.

Entering Disabled Mode

Software can request that a GPTA®v5 peripheral module be put into Disabled Mode by setting GPTA0_CLC.DISR. The GPTA®v5 will also be put into Disabled Mode if the sleep mode is requested and the GPTA®v5 module is configured to allow Sleep Mode.

In Secure Shut-off Mode, the GPTA®v5 module first finishes any operation in progress, then proceeds with an orderly shut down. When all sub-components of the GPTA®v5 module are ready to be shut down, the GPTA®v5 module signals its clock control cells, which turns off the clock to this peripheral module, that it is now ready for shut down. The status bit GPTA0_CLC.DISS is updated by the peripheral module accordingly.

The kernel logic of the GPTA®v5 and its FPI Bus interface must both perform shut-down operations before the clock can be shut off in Secure Shut-off Mode. This is performed as follows. The GPTA®v5's FPI Bus interface provides an internal acknowledge signal as soon as any current bus interface operation is finished. For example, if there is a DMA write access to the GPTA®v5 in progress when a disable request is detected, the access will be terminated correctly. Similarly, the GPTA®v5's kernel provides an internal acknowledge signal when it has entered a stable state. The clock control cells for the

GPTA®v5 module shuts off the GPTA®v5 unit clocks when it receives both acknowledge signals.

During emulation and debugging, it may be necessary to monitor the instantaneous state of the machine – including all or most of its modules – at the moment a software breakpoint is reached. In such cases, it may not be desired that the kernel of the GPTA®v5 finish whatever transaction is in progress before stopping, because that might cause important states in this GPTA®v5 module to be lost. Fast Shut-off Mode, controlled by bit GPTA0_CLC.FSOE, is available for this situation.

If GPTA0_CLC.FSOE = 0, the GPTA®v5 is stopped as described above. This is called Secure Shut-off Mode. The GPTA®v5 kernel is allowed to finish whatever operation is in progress. The clock to the module is then shut off if both the bus interface and the GPTA®v5 kernel have finished their current activity. If Fast Shut-off Mode is selected (GPTA0_CLC.FSOE = 1), clock generation to the module is stopped as soon as any outstanding bus interface operation is finished. The clock control cells does not wait until the GPTA®v5 kernel has finished its transaction. This option stops the GPTA®v5's clock as fast as possible, and the state of the GPTA®v5 will be the closest possible to the time of the occurrence of the software breakpoint.

Note: In all TC1797 modules the only shut down operating mode that is available is the Fast Shut-off Mode, regardless of the state of the FSOE bit.

Whether Secure Shut-off Mode or Fast Shut-off Mode is required depends on the application, the needs of the debugger, and the type of module.

Note that it is never appropriate for application software to set the GPTA0_CLC.FSOE bit. Fast Shut-off Mode should only be set by debug software. To guard against application software accidently setting GPTA0_CLC.FSOE, bit GPTA0_CLC.FSOE is specially protected by the mask bit GPTA0_CLC.SBWE. The GPTA0_CLC.SPEN bit can only be written if, during the same write operation, GPTA0_CLC.SBWE is set, too. Application software should never set GPTA0_CLC.SBWE to 1. In this way, user software can not accidentally alter the value of the GPTA0_CLC.FSOE bit. Note that this is the same guard mechanism used for the GPTA0_CLC.SPEN bit.

22.7.5.2 Fractional Divider Register

The fractional divider makes it possible to generate a GPTA®v5 module clock from an input clock using a programmable divider. The fractional divider divides the input clock f_{CLC} either by the factor 1/n or by a fraction of n/1024 for any value of n from 0 to 1023, and outputs the clock signal, f_{GPTA}. The fractional divider is controlled by the FDR register. **Figure 22-101** shows the fractional divider block diagram.

Overview

The adder logic of the fractional divider can be configured for two operating modes:

- Reload counter (addition of +1), generating an output clock pulse on counter overflow
- Adder that adds a STEP value to the RESULT value and generates an output clock pulse on counter overflow

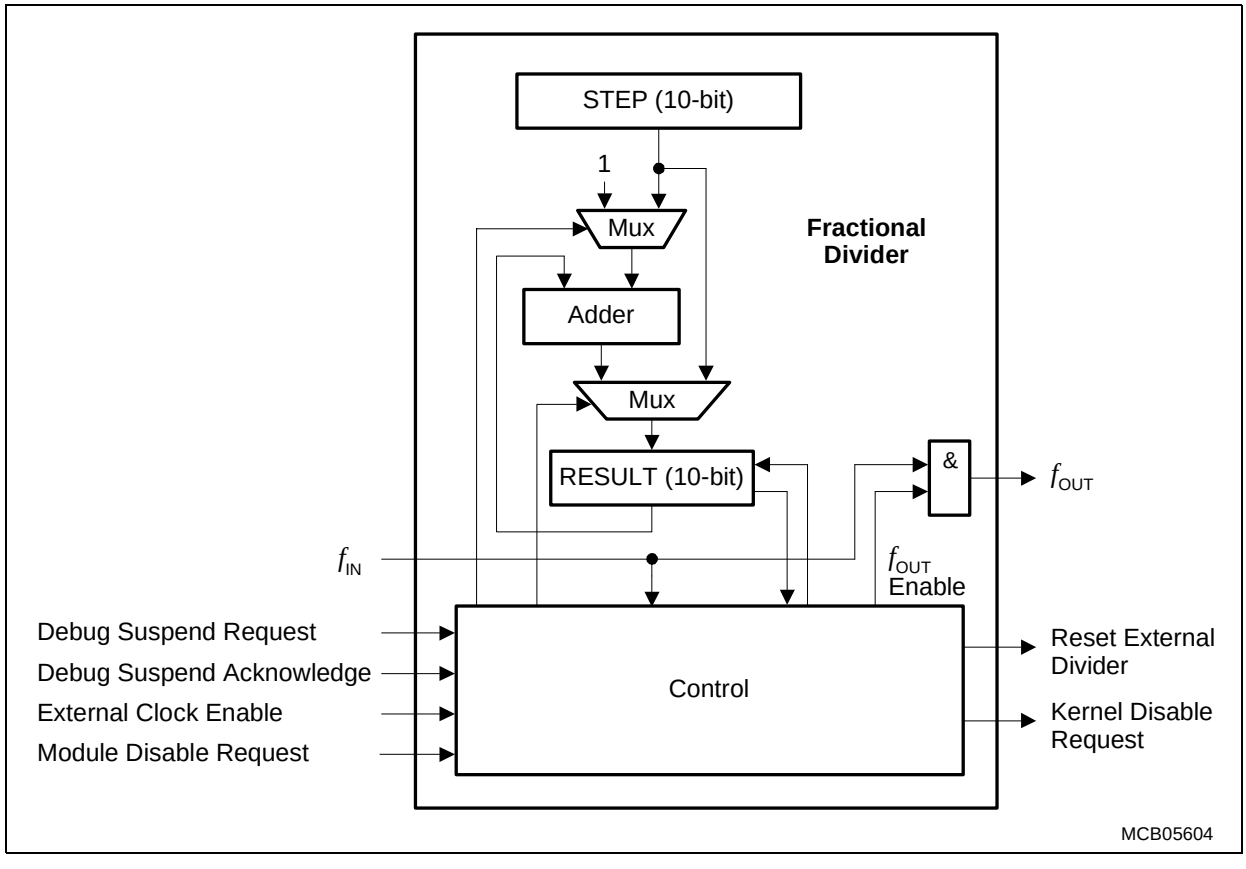


Figure 22-101Fractional Divider Block Diagram

The adder logic of the fractional divider can be configured for two operating modes:

- **Normal Mode:** Reload counter, generating an output clock pulse on counter overflow (GPTA0_FDR.RESULT = GPTA0_FDR.RESULT + 1).

- **Fractional Divider Mode:** Adder that adds a GPTA0_FDR.STEP value to the GPTA0_FDR.RESULT value and generates an output clock pulse on counter overflow.

The fractional divider is further controlled by several input and output signals. The purpose of these signals is described in **Table 22-35**.

Table 22-35 Fractional Divider Control I/O Lines

Signal	I/O	Description
Debug Suspend Request	Input	This input becomes active when a general suspend request is issued from the debug system to the GPTA®v5 module.
Debug Suspend Acknowledge		This input is driven with the disable acknowledge signal from the GPTA®v5 module kernel. This disable acknowledge signal is activated by the GPTA®v5 module kernel as a response to a suspend request that has been generated by the fractional divider via the GPTA®v5 Kernel Disable Request signal.
External Clock Enable		This input can be used to synchronize the fractional divider clock generation to external events.
GPTA®v5 Module Disable Request		This input is connected to the disable request output from the CLC logic (see **Figure 22-100**). An active signal at this input activates the GPTA®v5 Kernel Disable Request signal.
Kernel Disable Request	Output	This output signal becomes active when either the GPTA®v5 Module Disable Request input or the Debug Suspend Request input become active.
Reset External Divider		This output signal makes it possible to control (stop/reset) external divider stages which have f_{GPTA} as input.

*Note: In the TC1797, the fractional divider input clock f_{IN} is also referred to as f_{SYS} and the fractional divider input clock f_{OUT} is also referred to as f_{GPTA} (see **Figure 22-100**).*

Fractional Divider Operating Modes

The fractional divider has two operating modes:

- Normal divider mode
- Fractional divider mode

Normal Divider Mode

In normal divider mode (GPTA0_FDR.DM = 01_B), the fractional divider behaves as a reload counter (addition of +1) that generates an output clock pulse at f_{GPTA} on the transition from $3FF_H$ to 000_H. GPTA0_FDR.RESULT represents the counter value and GPTA0_FDR.STEP determines the reload value.

The output frequencies in normal divider mode are defined according to the following formulas:

$$f_{OUT} = f_{IN} \times \frac{1}{n} \quad \text{, with n = 1024 - STEP} \tag{22.8}$$

In order to get $f_{GPTA} = f_{SYS}$ STEP must be programmed with $3FF_H$. **Figure 22-102** shows the operation of the normal divider mode with a reload value of GPTA0_FDR.STEP = $3FD_H$. The clock signal f_{GPTA} is the AND combination of the f_{GPTA} Enable signal with f_{SYS}.

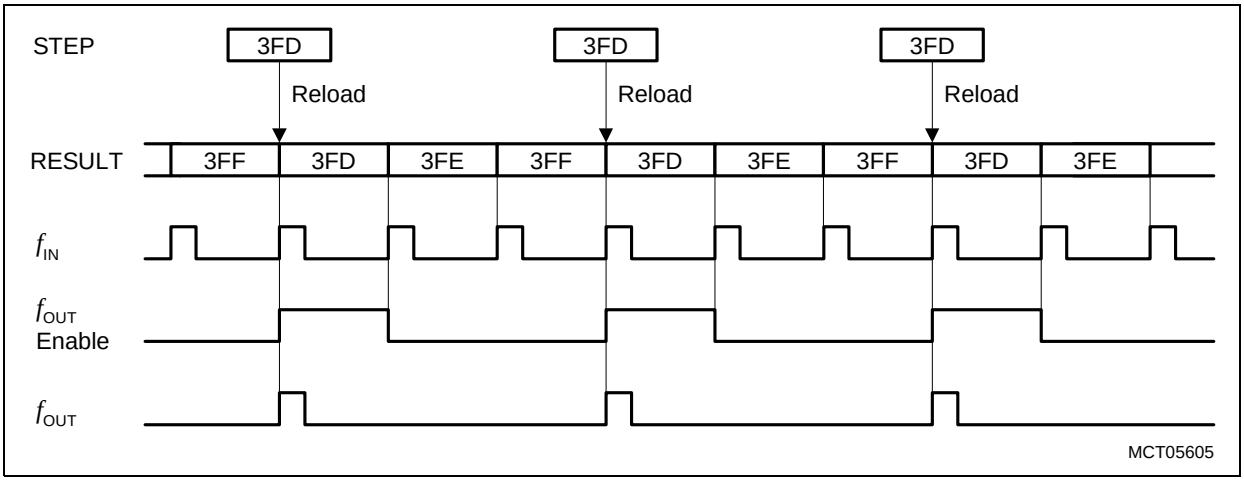


Figure 22-102Normal Mode Timing

Fractional Divider Mode

When the fractional divider mode is selected (GPTA0_FDR.DM = 10_B), the output clock f_{GPTA} is derived from the input clock f_{SYS} by division of a fraction of n/1024 for any value of n from 0 to 1023. In general, the fractional divider mode makes it possible to program the average output clock frequency with a higher accuracy than in normal divider mode.

In fractional divider mode, an output clock pulse at f_{OUT} is generated depending on the result of the addition GPTA0_FDR.RESULT + GPTA0_FDR.STEP. If the addition leads to an overflow over $3FF_H$, a pulse is generated at f_{OUT}. Note that in fractional divider mode the clock f_{GPTA} can have a maximum period jitter of one f_{IN} clock period.

The output frequencies in fractional divider mode are defined according to the following formulas:

$$f_{GPTA} = f_{SYS} \times \frac{n}{1024}, \text{ with n = 0-1023} \quad (22.9)$$

Figure 22-103 shows the operation of the fractional divider mode with a reload value of GPTA0_FDR.STEP = 234_H (= factor 564/1024 = 0.55). The clock signal f_{GPTA} is the AND combination of the f_{GPTA} Enable signal with f_{SYS}.

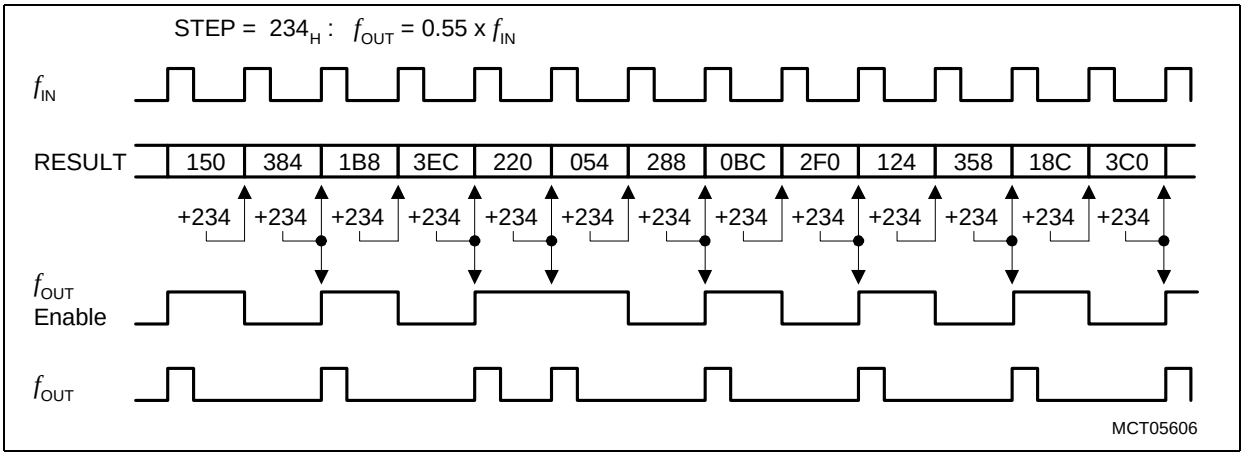


Figure 22-103Fractional Divider Mode Timing

Suspend Mode Control

The operation of the fractional divider can be controlled by the Debug Suspend Request input. This input is activated in suspend mode by the on-chip debug control logic. In suspend mode, GPTA®v5 module registers are accessible for read and write actions, but other GPTA®v5 module internal functions are frozen. Suspend mode is entered one f_{SYS} clock cycle after the Debug Suspend Request has been acknowledged by the Debug Suspend Acknowledge signal (granted suspend mode) **and** GPTA0_FDR.SC is not equal 00_B (clock output signal disabled). Suspend mode is immediately entered when bit SM is set to 1 **and** GPTA0_FDR.SC is not equal 00_B (immediate suspend mode).

The state of the Debug Suspend Request and Debug Suspend Acknowledge signal is latched in two status flags of register GPTA0_FDR, GPTA0_FDR.SUSREQ and GPTA0_FDR.SUSACK. Debug Suspend Request and (Debug Suspend Acknowledge or bit GPTA0_FDR.SM) must remain set both to maintain the suspend mode.

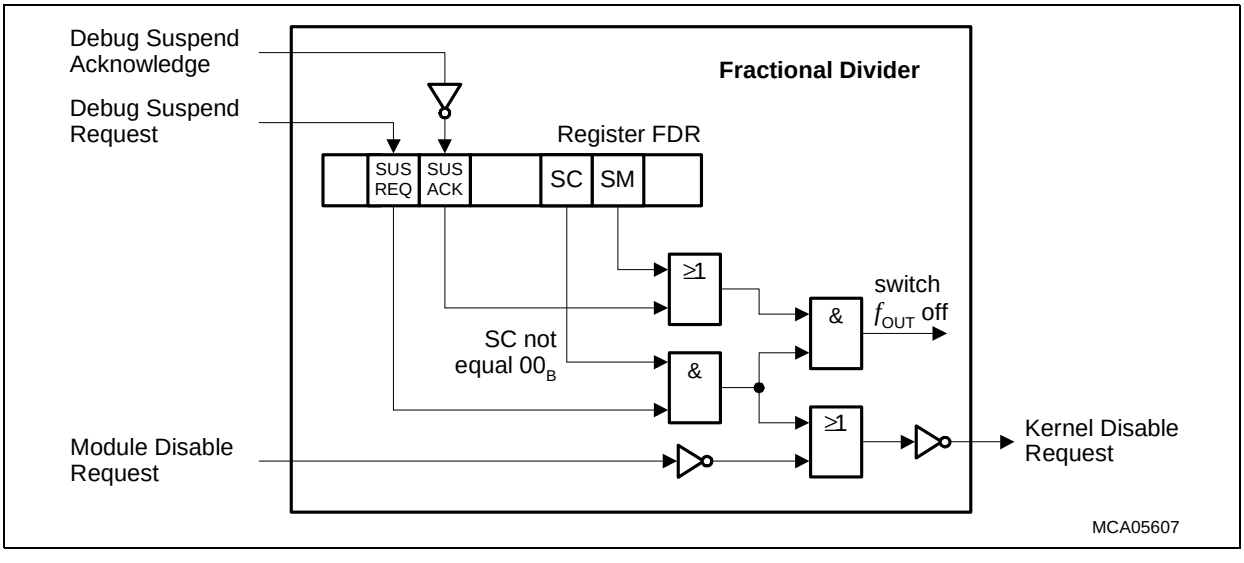


Figure 22-104Suspend Mode Configuration

The Kernel Disable Request signal becomes always active when the GPTA®v5 Module Disable Request signal is activated, independently of the suspend mode settings in the fractional divider logic.

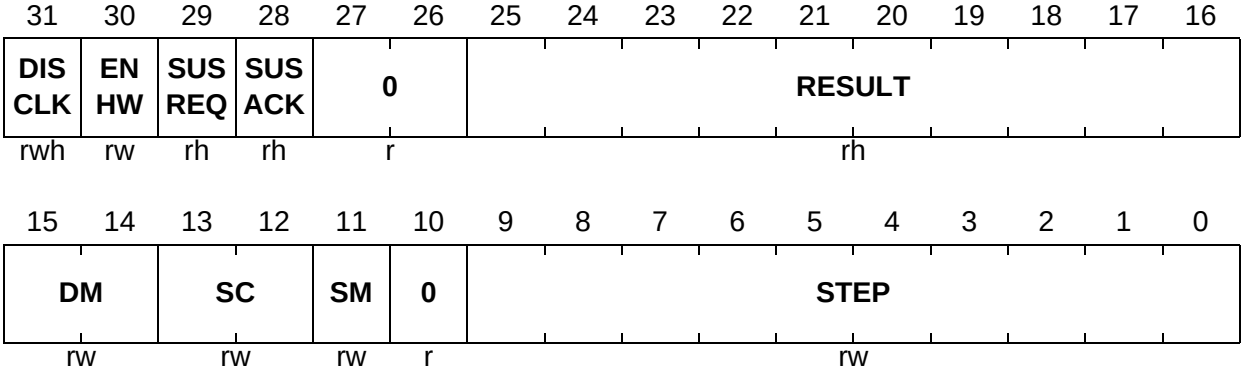
External Clock Enable

When the GPTA®v5 module clock generation has been disabled by software (setting GPTA0_FDR.DISCLK = 1), the disable state can be exited (hardware controlled) when the External Clock Enable input = 1. This feature is enabled when GPTA0_FDR.ENHW = 1.

The fractional divider register controls the clock frequency of the GPTA®v5 module timer clock f_{GPTA}. The clock frequency of f_{GPTA0}, f_{GPTA1}, f_{LTCA2} is identical to the one of f_{GPTA}.

GPTA0_FDR

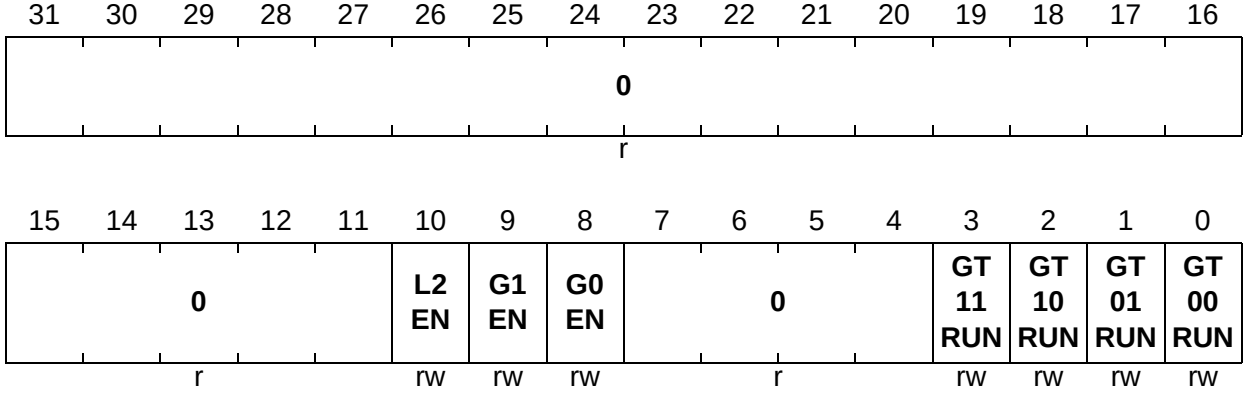
GPTA Fractional Divider Register (00C$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
STEP	[9:0]	rw	**Step Value** Reload or addition value for RESULT.
SM	11	rw	**Suspend Mode** SM selects between granted or immediate suspend mode.
SC	[13:12]	rw	**Suspend Control** This bit field determines the behavior of the fractional divider in suspend mode.
DM	[15:14]	rw	**Divider Mode** This bit field selects normal divider mode, fractional divider mode, and off-state.
RESULT	[25:16]	rh	**Result Value** Bit field for the addition result.
SUSACK	28	rh	**Suspend Mode Acknowledge** Indicates state of SPNDACK signal.
SUSREQ	29	rh	**Suspend Mode Request** Indicates state of SPND signal.
ENHW	30	rw	**Enable Hardware Clock Control** Controls operation of ECEN input and DISCLK bit.
DISCLK	31	rwh	**Disable Clock** Hardware controlled disable for f_{OUT} signal.
0	10, [27:26]	r	**Reserved** Read as 0; should be written with 0.

The clock enable/disable control register controls two functions: clock enable/disable control for each Global Timer in the GPTA0/GPTA1 units and enable/disable control for the GPTA®v5 unit clocks, separately for each clock f_{GPTA0}, f_{GPTA1}, f_{LTCA2}.

GPTA0_EDCTR
GPTA Clock Enable/Disable Control Register
(400_H) **Reset Value: 0000 0000_H**

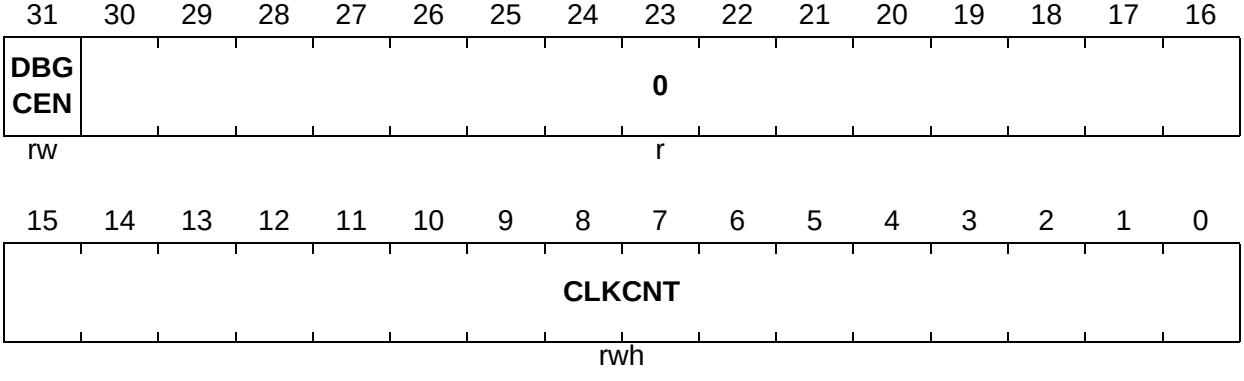


Field	Bits	Type	Description
GT00RUN	0	rw	**GPTA0 Global Timer 0 Run Control** 0_B GPTA0 Global Timer 0 clock is stopped. 1_B GPTA0 Global Timer 0 clock is started/running.
GT01RUN	1	rw	**GPTA0 Global Timer 1 Run Control** 0_B GPTA0 Global Timer 1 clock is stopped. 1_B GPTA0 Global Timer 1 clock is started/running.
GT10RUN	2	rw	**GPTA1 Global Timer 0 Run Control** 0_B GPTA1 Global Timer 0 clock is stopped. 1_B GPTA1 Global Timer 0 clock is started/running.
GT11RUN	3	rw	**GPTA1 Global Timer 1 Run Control** 0_B GPTA1 Global Timer 1 clock is stopped. 1_B GPTA1 Global Timer 1 clock is started/running.
G0EN	8	rw	**GPTA0 Timer Clock Enable** 0_B GPTA0 timer clock f_{GPTA0} is disabled. 1_B GPTA0 timer clock f_{GPTA0} is enabled.
G1EN	9	rw	**GPTA1 Timer Clock Enable** 0_B GPTA1 timer clock f_{GPTA1} is disabled. 1_B GPTA1 timer clock f_{GPTA1} is enabled.

Field	Bits	Type	Description
L2EN	10	rw	**LTCA2 Timer Clock Enable** 0_B LTCA2 timer clock f_{LTCA2} is disabled. 1_B LTCA2 timer clock f_{LTCA2} is enabled.
0	[7:4], [31:11]	r	**Reserved** Read as 0; should be written with 0.

The debug clock control register makes it possible to control the GPTA®v5 unit clocks f_{GPTA0}, f_{GPTA1}, f_{LTCA2} for debug purposes on the basis of a clock counter.

GPTA0_DBGCTR
GPTA Debug Clock Control Register (004_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
CLKCNT	[15:0]	rwh	**Debug Clock Count** This bit field determines the number of clock pulses to be issued when the debug clock feature is enabled (DBGCEN = 1). CLKCNT counts down to 0000_H and stops when the debug clock feature is enabled.
DBGCEN	31	rw	**Debug Clock Enable** 0_B The debug clock feature is disabled. The GPTA®v5 unit clocks are always enabled. 1_B The debug clock feature is enabled. If a non-zero value is written to bit field CLKCNT the related number of clock pulses is issued at f_{GPTA0}, f_{GPTA1}, f_{LTCA2}.
0	[30:16]	r	**Reserved** Read as 0; should be written with 0.

22.7.6 Limits of Cascading GTCs and LTCs

As shown on **Page 22-57** and **Page 22-69**, a maximum of 32 GTCs and a maximum of 64 LTCs can be cascaded. In the TC1797, however cascading of GTCs and LTCs is limited under certain conditions.

If the LTCs are running with the maximum GPTA®v5 unit clock of $f_{GPTA} = f_{SYS} = 90$ MHz, a maximum of 16 GTCs and 16 LTCs can be connected together. If the GPTA®v5 unit clock f_{GPTA} is reduced, the number of LTCs that can be cascaded increases accordingly. Only the integer part of the divider ratio as selected by the GPTA0_FDR fractional divider register determines the maximum number of cascaded GTCs and LTCs.

Table 22-36 Limits of Cascading GTCs and LTCs

f_{SYS}	Selected Clock Divider Ratio[1)]	Max. Number of Cascaded GTCs/LTCs
90 MHz	$1 <= f_{SYS}/f_{GPTA} < 2$	16 GTCs, 16 LTCs
	$2 <= f_{SYS}/f_{GPTA} < 3$	no limits for GTCs, 32 LTCs
	$3 <= f_{SYS}/f_{GPTA} < 4$	no limits for GTCs, 48 LTCs
	$4 <= f_{SYS}/f_{GPTA}$	no limits for GTCs and LTCs
45 MHz	$1 <= f_{SYS}/f_{GPTA} < 2$	no limits for GTCs, 32 LTCs
	$2 <= f_{SYS}/f_{GPTA}$	no limits for GTCs and LTCs

1) Selected by the GPTA0_FDR fractional divider register.

22.7.7 Interrupt Registers

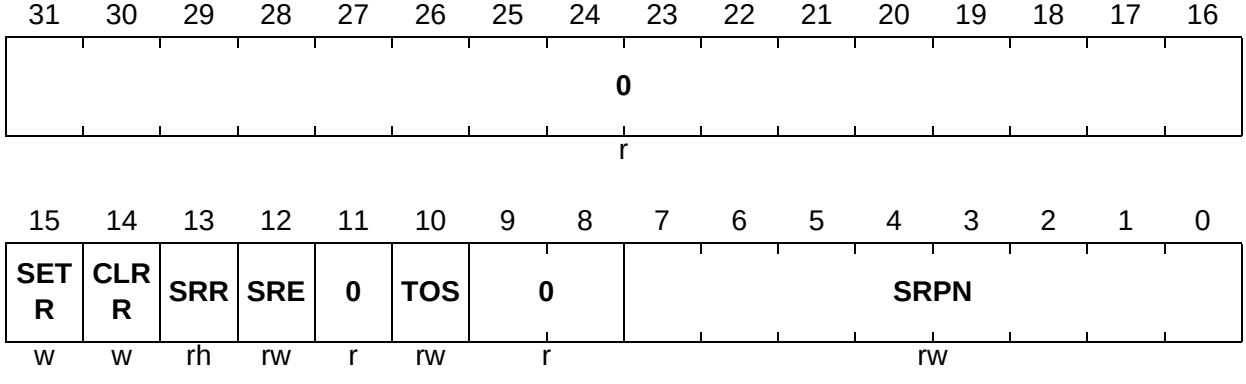
Each of the service request outputs of the GPTA0/GPTA1/LTCA2 units is able to generate an interrupt and is controlled by an interrupt service request control register GPTA_SRCk. Therefore, the following interrupt service request control registers are available:

- GPTA0: GPTA0_SRC[37:00]
- GPTA1: GPTA1_SRC[37:00]
- LTCA2: LTCA2_SRC[07:00]

GPTA0_SRCk (k = 00-37)
GPTA0 Interrupt Service Request Control Register k
($7FC_H$-k*4_H) **Reset Value: 0000 0000_H**

GPTA1_SRCk (k = 00-37)
GPTA1 Interrupt Service Request Control Register k
($7FC_H$-k*4_H) **Reset Value: 0000 0000_H**

LTCA2_SRCk (k = 00-07)
LTCA2 Interrupt Service Request Control Register k
($7FC_H$-k*4_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
SRPN	[7:0]	rw	**Service Request Priority Number**
TOS	10	rw	**Type of Service Control**
SRE	12	rw	**Service Request Enable**
SRR	13	rh	**Service Request Flag**
CLRR	14	w	**Request Clear Bit**
SETR	15	w	**Request Set Bit**
0	[9:8], 11, [31:16]	r	**Reserved** Read as 0; should be written with 0.

Note: Additional details on service request nodes and the service request control registers are described in the Interrupt chapter of the TC1797 User´s Manual System Units part (Volume 1).

22.7.8 GPTA Register Address Map

The GPTA0 and GPTA1 register map shown in **Figure 22-105**. The LTCA2 register map shown in **Figure 22-106**.

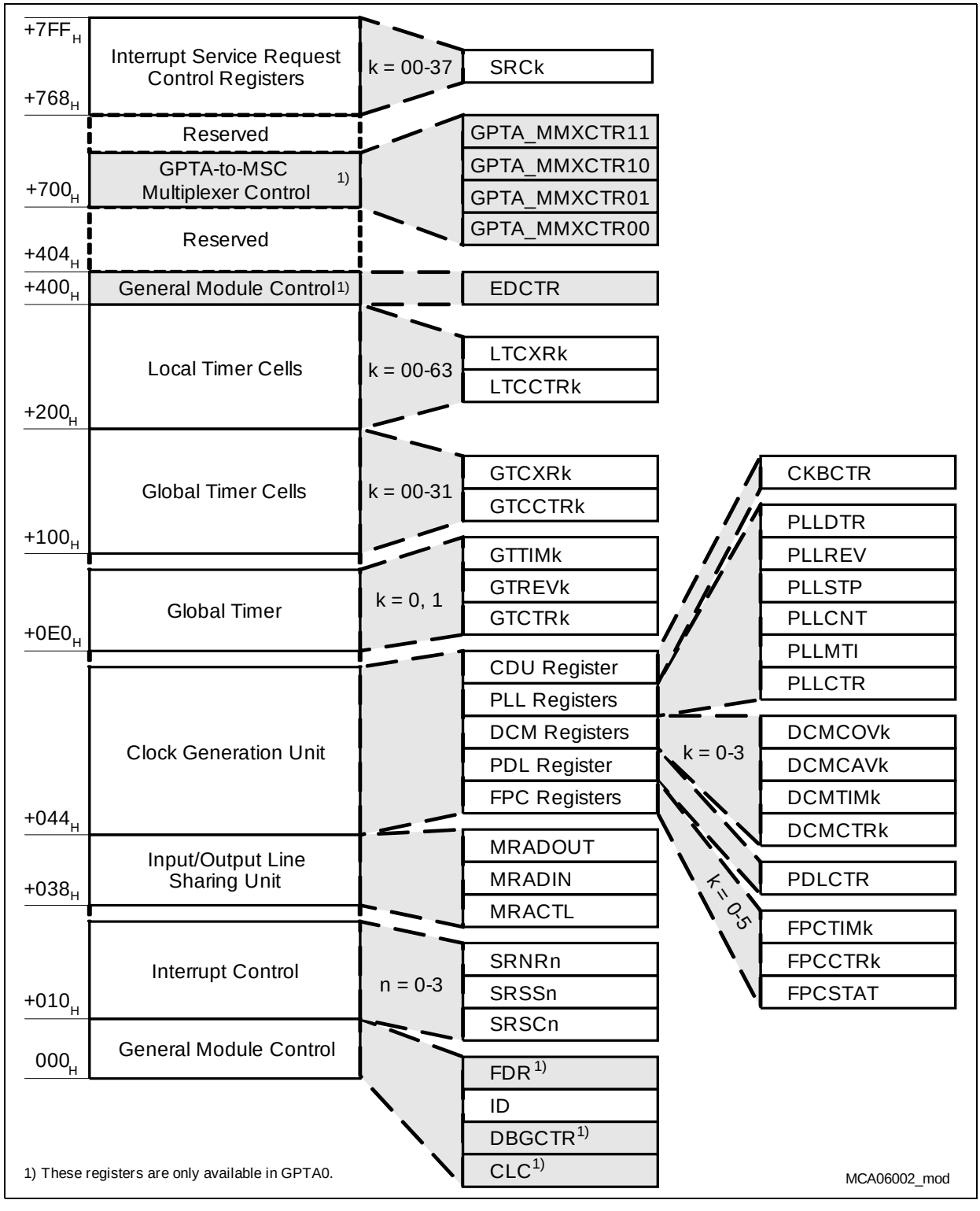


Figure 22-105GPTA0/GPTA1 Register Map

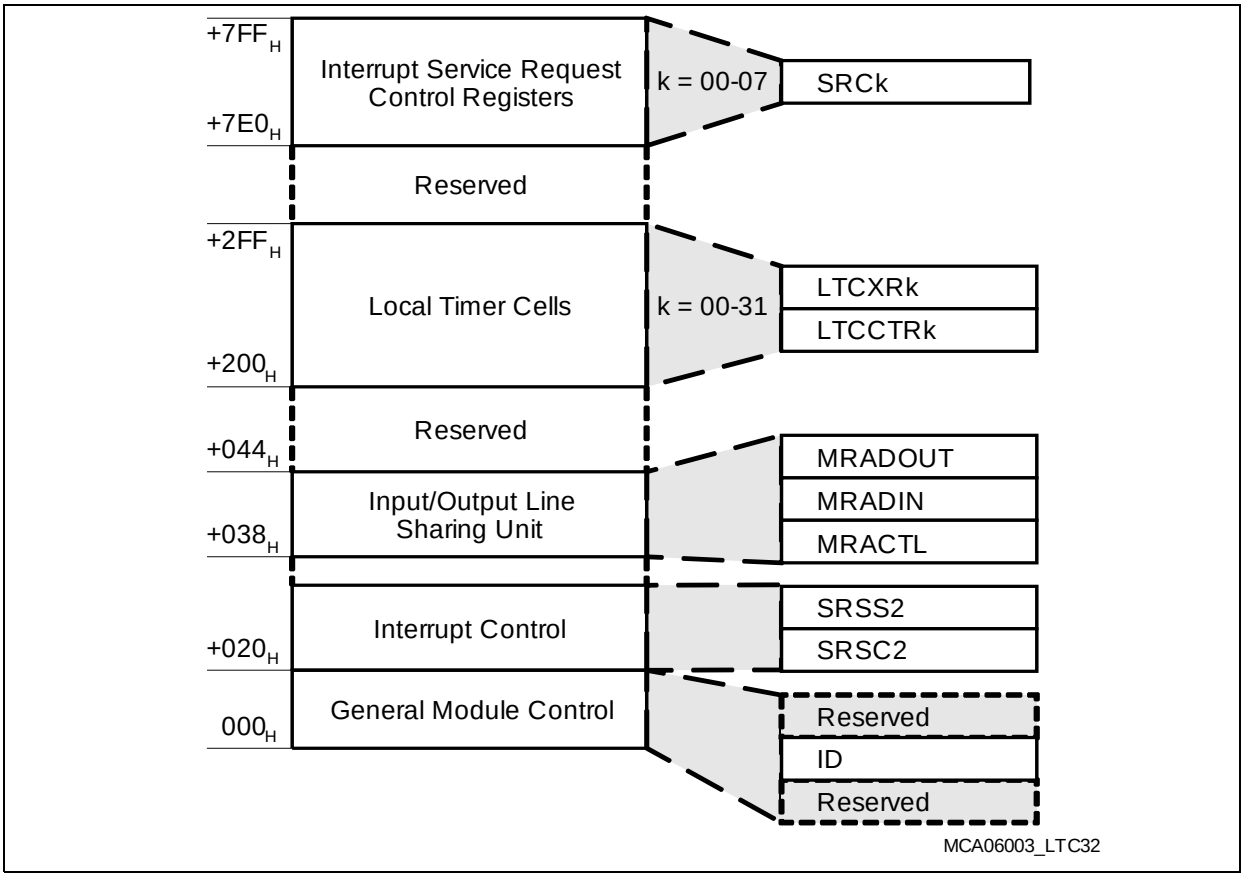


Figure 22-106LTCA2 Register Map

22.8 Revision History

This chapter gives a summary of recent changes within this specification.

Table 22-37 Revision History

Version Number	Changes to Previous Version
Rev_1.4	Changed the top level connection of the MSC and the output of the MSC multiplexer. Included a short description how to achieve 100% and 0% duty cycle if using LTC for PWM. This note has been in old Target Specifications and was not part of the user manual. Corrected a bug in the pseudo code of the FPC concerning external clocks. Corrected some attributes of register description to enable clean extraction (“No VHDL”, “Multiple View”. Included in the Connection figures missing port 3. Included Port 0, Port1, Port5, Port13 and Port14 as additional outputs for the GPTA0, GPTA1, and LTCA2. Changed attribute from VHDL = No to VHDL = Yes for GPTA0_MRACTL, GPTA0_MRADIN, GPTA0_MRADOUT, LTCA2_MRACTL, LTCA2_MRADIN, and LTCA2_MRADOUT. Changed address of GPTA0_LTCXR63 from 0xF0001BCF to 0xF0001BFC.
Rev_1.5	Cleaned-up the Register Index and the index entries. Included the Timer Overflow in the Output Action description for LTCs. Modified the Pseudo Code LTC MUX to include the reset of the EOA bit.
Rev_1.6	Modified the Block diagram of Implementation of TC1797. Included the Trigger Out signals and replaced the connection of GPTA1 to FADC/ADC/DMA by respective connection to GPTA0. Included the GPTA0 to ERAY connections and modified the number of LTCA2 interrupts from 9 to 8. LTCCTR Register in the Compare Modus description the GBYP Bit was missing and has been included. The description of the MSC multiplexer register has been reworked, because neither TC1797 nor TC1767 was consistent due to top level differences of these two devices concerning MSC connections. Modified in the MMXCTR10/11 description the out numbers of the LTCA2. Modified some conditional text settings (no TC1797).

Table 22-37 Revision History (cont'd)

Version Number	Changes to Previous Version
Rev_1.7	Included remark on how the ECEN signal is connected. Corrected some typos in the pin connection list. Included the GBYP Bit into the register figure of Local Timer Cell Control Register k [Compare Mode] of LTCA2, GPTA1, and GPTA0. Removed the edge selection for medium driver PD group. Include a remark ahead of the table of PD coding to emphasis that not all port support all driver strength. Removed for TC1767 LTCA2 output on Port 5Block Diagrams of TC1767 and TC1797. Included ECEN name and updated number of Inputs and Outputs (IN and OUT). Updated for LTCA2 i figures the correct output numbering (OUT80-OUT111). Updated MSC0
Rev_1.8	Clarified for LTC Reseted Timer Mode the Range (period) of the timer values. For using LTC for PWM generation, note was added to clarify how to achieve 100% and 0% duty cycle.
Rev_1.9	Corrected the LTCA2 IO Sharing Table for IOG0. Included for FPC detailed description of the edge detection cell usage. Corrected typo in table: "On-Chip Trigger/Gating Multiplexer Control Register Assignments", IOG3 OUT28 IN07 OTMCR1 TRIG17 GTCG0, LTCG0, LTCG4. Updated the Figure "Figure I/O Port Line Assignment". Two ports nibbles where assigned to LTCA, correct is GPTA. Some formatting in the table "Output Multiplexer Control Register Assignments "and table "IOCR Assignment for GPTA®v5 Port Lines"
Rev_1.10	Included TC1736 specifics. Modified the figures MCT05929, MCT05930, MCT05931, and MCT05932. The PLL Input within these figures was shown as square wave input with 50% duty cycle. But the Input of the PLL is connected only to the output of the DCM, which only generate an output with a single f_{GPTA} clock pulse. Therefore PLL Input signal has been modified to be a pulse train. Figure MCT05932 implies an action on the falling edge of the accelerated Input Signal. This is incorrect and has been shifted to the previous rising edge.

Table 22-37 Revision History (cont'd)

Version Number	Changes to Previous Version
Rev_1.11	Modified the tables "PDR Assignment for GPTA®v5 Port Lines" in TC7197, TC1767, and TC1736 specifications. Included the Pad Classes in PDR assignment table. Included PDx description for A1 and F pad class. Table on On-Chip Trigger/Gating Multiplexer Control Register Assignments modified. For LTCA2 the numbering of the Output Groups was done consistently as OG3-OG6. Further more the Output Multiplexer of LTCA2 has been made consistently to be 0-3/4 and 7/10-13. OUT[65], OUT[76] and OUT[78] are marked as signals not available as output for GPTA1.
Rev_1.12	

23 Analog to Digital Converter

The **A**nalog to **D**igital **C**onverter module (ADC) of the TC1797 allows the conversion of analog input values into discrete digital values based on the successive approximation method. With this method, the conversion result is elaborated bit by bit, starting with the most significant bit. As a consequence, an analog to digital conversion requires a certain number of clock cycles.

This chapter is structured as follows:

- Introduction (see **Section 23.1**)
- Operating the ADC (see **Section 23.2**)
- Module implementation in TC1797 (see **Section 23.3**)

23.1 Introduction

This section gives an overview about the feature set of the ADC module and introduces the general structure. It describes the:

- ADC block diagram (see **Section 23.1.1**)
- Feature set description (see **Section 23.1.2**)
- Abbreviations (see **Section 23.1.3**)
- Kernel overview (see **Section 23.1.4**)
- Conversion request handling (see **Section 23.1.5**)
- Conversion result handling (see **Section 23.1.6**)
- Interrupt structure (see **Section 23.1.7**)
- Electrical models (see **Section 23.1.8**)
- Transfer characteristics and error definitions (see **Section 23.1.9**)

23.1.1 ADC Block Diagram

The ADC module contains 3 independent kernels (ADC0, ADC1, ADC2) that can operate autonomously or can be synchronized to each other. An ADC kernel is a unit used to convert an analog input signal into a digital value and provides means for triggering conversions, data handling and storage.
With this structure, parallel conversion of up to three analog input channels is supported.

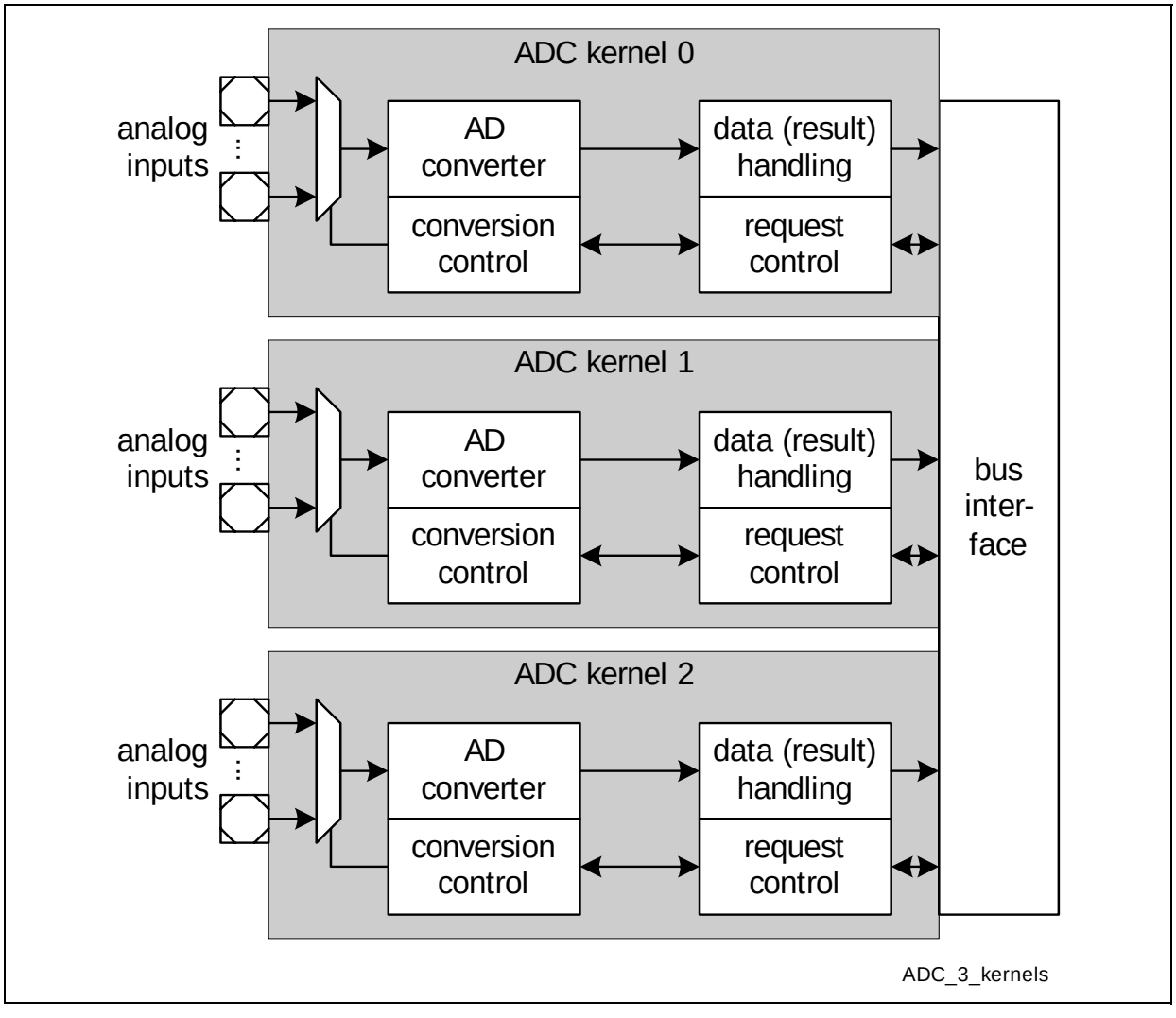


Figure 23-1 ADC Module Block Diagram

23.1.2 Feature Set

Features of each ADC kernel:

- Analog supply voltage range from 3.3 V (minimum) to 5 V (nominal) for V_{DDM}
- Input voltage range from 0 V to analog supply voltage V_{DDM}
- Input multiplexer for a maximum of 16 possible analog input channels
- Performance for 12 bit resolution (@f_{ADCI} = 10 MHz):
  - conversion time less than 2 µs, ±4 LSB_{12} TUE[1)] @ supply voltage V_{DDM} = 5 V
  - conversion time about 2.5 µs, **tbd** LSB_{12} TUE @ supply voltage V_{DDM} = 3.3 V
- One standard reference input (V_{AREF}) and one alternative reference input (CH0) available
- Multiplexer test support
- 5 conversion request sources for external or timer-driven events, auto-scan, programmable sequences, SW-driven conversions, etc.
- Synchronization of the ADC kernels for concurrent conversion starts and parallel sampling and measuring of analog input signals, e.g. for phase current measurements in AC drives
- Control capability for an external analog multiplexer, respecting the additional set up time
- Adjustable sampling times to accommodate output impedance of different analog signal sources (sensors, etc.)
- Possibility to cancel running conversions on demand with automatic restart
- Flexible interrupt generation (possibility of DMA support)
- Limit checking to reduce interrupt load (e.g. for temperature measurements or overload detection, only values exceeding a programmable level lead to an interrupt)
- Programmable data reduction filter, e.g. for digital anti-aliasing filtering, by adding a programmable number of conversion results
- Independent result registers (16 independent registers)
- Support of conversion result FIFO mechanism to allow a longer interrupt latency
- Support of suspend and power saving modes
- Individually programmable reference selection for each channel, e.g. to allow measurements of 3.3 V and 5 V signals in the full measurement range with the same ADC kernel (with exception of dedicated channels always referring to V_{AREF})

1) This value reflects the ADC module capability in an adapted electrical environment, e.g. characterized by “clean” routing of analog and digital signals and separation of analog and digital PCB areas, low noise on analog power supply (< 30mV), low switching activity of digital pins near to the ADC, etc.

23.1.3 Abbreviations

The following acronyms and terms are used in the ADC chapter:

Table 23-1 Abbreviations in ADC chapter

Abbreviation	Meaning
ADC	analog to digital converter
DMA	direct memory access mechanism
DNL	differential non-linearity error
FIFO	first-in-first-out data buffer mechanism
INL	integral non-linearity error
LSB_n	finest granularity of the analog value in digital format, represented by one least significant bit of the conversion result with n bits resolution (measurement range divided in 2^n equally distributed steps)
SCU	system control unit of the device
TUE	total unadjusted error

23.1.4 ADC Kernel Overview

Each ADC kernel comprises:

- An **analog to digital converter** with a maximum of 16 analog inputs (CH0 - CH15). This block selects an input signal and translates the analog voltage into a digital value.
  Not all analog input channels are necessarily available in all packages, please refer to the implementation description in **Section 23.3**.
- A **conversion control** unit defining the conversion parameters like the length of the sample phase, the resolution and the reference for each conversion. The length of the sample phase and the resolution depend on the type of sensor (or other analog sources) connected to the ADC. These values are similar for several channels and, therefore, are grouped together to form the so-called input classes. Each channel can be individually assigned to an input class to define these parameters.
  The conversion control also handles the start conditions for the conversions, such as the immediate start (cancel-inject-repeat), overwrite of former results (wait-for-read), or synchronization of the ADC kernels (parallel conversions).
  Additionally, an external analog multiplexer can be controlled by the output signals EMUX[2:0] of each ADC kernel.
- A **request control** unit defining which analog input channel has to be converted next. It contains 5 request sources that can trigger conversions depending on different events, such as edges of PWM or timer signals or events at port pins. Each request source can trigger either 1, up to 4, or up to 16 conversions in a sequence.
- A **result handling** unit providing 16 result registers for the conversion results. The conversion result of each analog input channel can be directed to one of the result registers to be stored there. The result handling block also supports data reduction (e.g. for digital anti-aliasing filtering) by automatically adding up to 4 conversion results before informing the CPU that new data is available.
  Additionally, the results registers can be concatenated to FIFO structures to provide storage capability for more than one conversion result without overwriting previous data. This feature also helps to handle CPU latency effects.
- An **interrupt generation** unit issuing interrupt requests to the CPU depending on ADC events. The interrupt generation in the ADC kernels support different mechanisms, e.g. some interrupts can be coupled to a value range of the conversion result (limit checking), some interrupts can be used to transport conversion data to locations in memory for further treatment, and other interrupts are generated after a complete sequence of conversions.

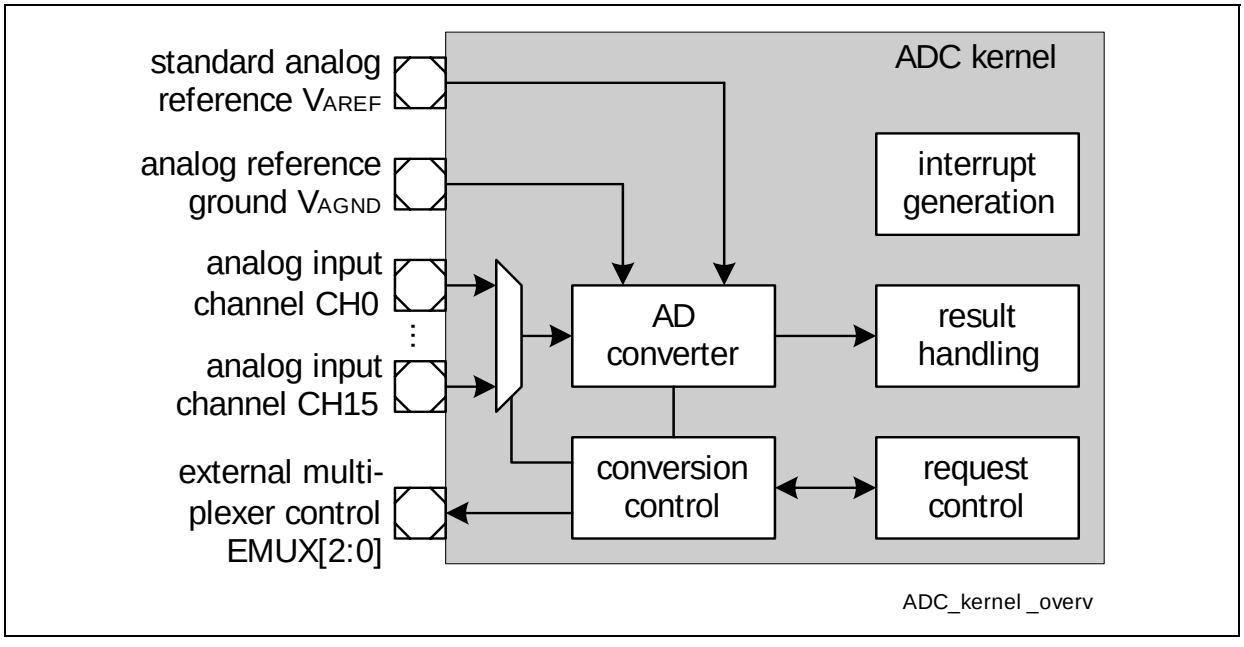


Figure 23-2 ADC Kernel Block Diagram

23.1.5 Conversion Request Unit

The conversion request unit of each ADC kernel autonomously handles the generation of conversion requests.
It contains 5 independent request sources that are connected to several modules to trigger the start of a conversion. A request source defines the analog input channel to be converted if a defined event occurs. For example, a trigger pulse from a timer unit generating a PWM signal can start the conversion of a single input channel or a programmed sequence of input channels.
Depending on the application, the request sources can be triggered by different events, either issued by other modules or under SW control. As a consequence, there can be two or more conversion requests pending at the same time. To allow the user to adapt the request source mechanism to the application needs, the trigger capability, the channel number(s) to be converted, and the priority can be individually programmed for each request source.
An arbiter block regularly scans the request sources for pending conversion requests and acts upon the conversion request with the highest priority. This conversion request is forwarded to the converter to start the conversion of the requested channel.

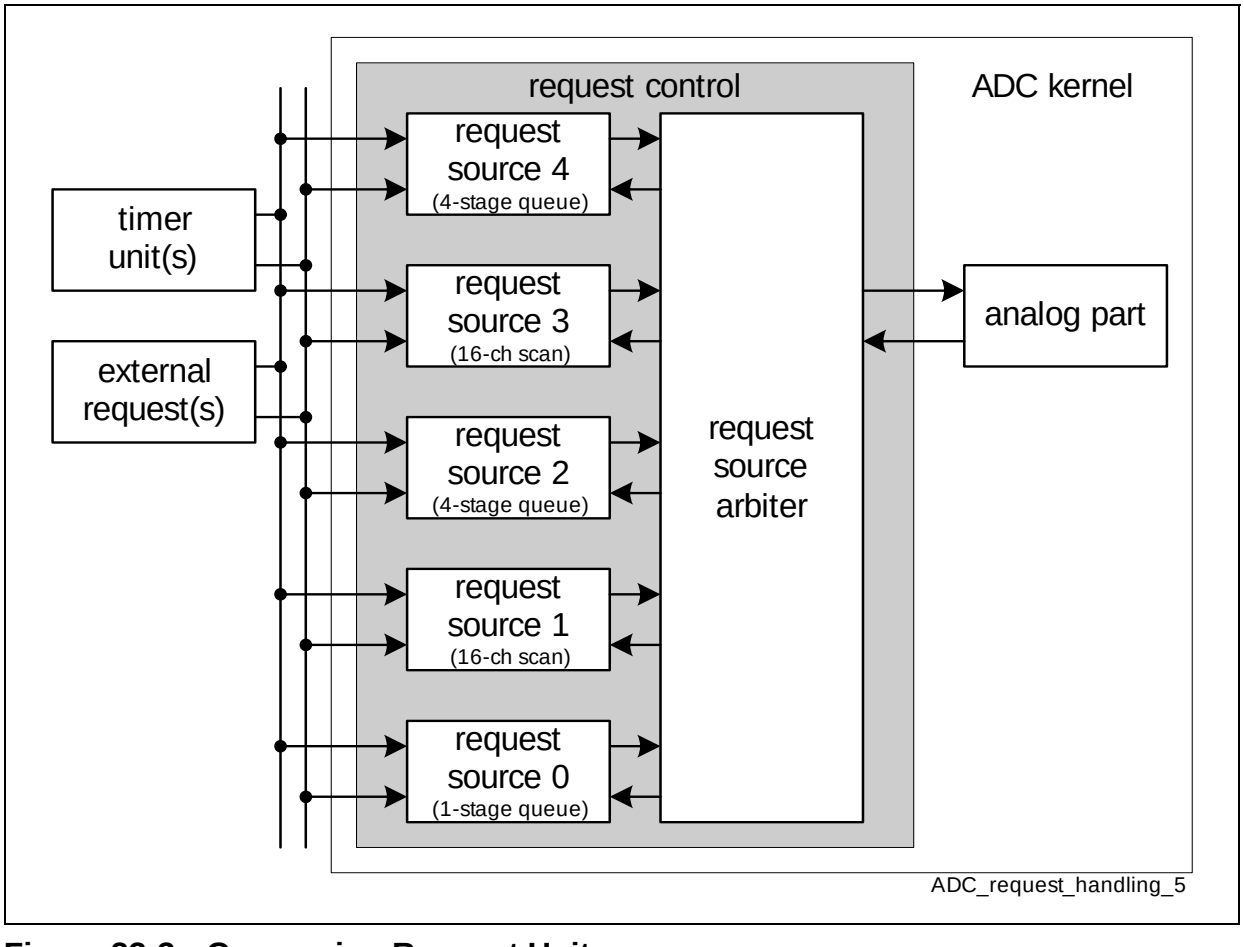


Figure 23-3 Conversion Request Unit

The functional characteristics of the request sources are adapted to the most common application requirements. In all request sources, a continuous operation or a single-shot operation can be selected. For continuous operation, the programmed sequence of conversions requests are continuously issued (once started), whereas in single-shot mode, each sequence of conversion requests has to be explicitly started. The trigger for a conversion request or a sequence can be handled under SW control or can be synchronized to ADC-external events, such as timer signals or port pins. For each request source, the user can select an input signal (from 8 possible signals REQTRx_[7:0]) as trigger input REQTRx and an input signal (from 8 possible signals REQGTx_[7:0]) as gating input REQGTx.

- **Request source 0** (1-stage sequential source) can issue a conversion request for a single input channel. The channel number can directly be programmed.
  This mechanism could be used for SW-controlled conversion requests or HW-triggered conversions of a single input channel. If programmed with a high priority, it can interrupt the sequences of the other request sources to inject a single conversion.
- **Request sources 1 and 3** (16-channel scan sources) can issue conversion requests for a sequence of up to 16 input channels. It can be programmed which channel takes part in this sequence. The sequence always starts with the highest enabled channel number and continues towards lower channel numbers (order defined by the channel number, each channel can be converted only once per sequence).
  This mechanism could be used to scan input channels permanently or on a regular time base. For example, if programmed with a low priority, some input channels can be scanned in a background task to update information that is not time-critical.
- **Request sources 2 and 4** (4-stage sequential sources) can issue a conversion request for a sequence of up to 4 input channels. The channel numbers can be freely programmed, especially multiple conversions of the same channel within the sequence are supported.
  This mechanism could be used to support application-specific conversion sequences that can not be covered by the scanning mechanism of request sources 1 or 3. Especially for timing-critical sequences containing multiple conversions of the same channel, one of these request sources should be used. For example, if programmed with a medium priority, some input channels can be converted when a specified event occurs (e.g. synchronized to a PWM) while the scan of other input channels of the background task (handled by request source 1) is interrupted.

23.1.6 Conversion Result Unit

The conversion result unit comprises for each ADC kernel:

- A set of **16 result registers** for storing the conversion results. A pointer mechanism for each analog input channel distributes the conversion results to the result registers. Especially for auto-scan applications, this feature simplifies DMA use (only one DMA channel needed to transfer a complete auto-scan sequence into the device memory).
- The result registers provide **valid flags** to indicate if new data has been stored since it has been read out (new data indication).
- A **result FIFO mechanism** for conversion results handling with a “relaxed” CPU timing. Result registers not directly used as target for a conversion result can be concatenated to form a result FIFO. This structure allows to store a sequence of conversion results before the CPU has to interact.
- A **digital anti-aliasing or data reduction filter**, accumulating a programmable number of conversion results before generating a result event interrupt.
  This feature can be used to avoid CPU intervention on each conversion result if a certain number of conversion results are added before further treatment, especially for fast conversions sequences and averaging of results.
- A **wait-for-read mechanism** can be enabled independently for each result register to delay conversions targeting a result register that has not yet been read out.
- A **flexible interrupt generation** based on result register events. A result register event occurs if a new valid data word becomes available in a result register and can be read out. Especially when using data reduction or digital anti-aliasing filtering, the result register event indicates that the final result is available.
- **Debugger support** for ADC result registers supporting read out of ADC conversion results without changing the result status (new data indication).

23.1.7 Interrupt Structure

Each ADC kernel provides 8 independent service request output signals (ADCx_SR[7:0]) used for interrupt handling. The interrupt generation inside the ADC kernel is based on three different types of events.

- **Channel events:**
  A channel event is detected if a conversion is finished and the conversion result is within a programmable value range.
  This type of event can be used to check if analog input values are inside or out of a nominal operating range, especially to reduce CPU load for background tasks. This allows the user to interrupt the CPU only if the specified conversion result range is met (or not met) instead of comparing each result by SW.
- **Result events:**
  A result event is detected if a new result is available in a result register and can be read out, e.g. to store the data in memory for further treatment by SW.
  This type of event can be used to trigger a read action by the CPU (or DMA). Especially when using data reduction or digital anti-aliasing filtering, not all finished conversion leads to a new result. Furthermore, when using a result FIFO, a result event decouples the CPU (DMA) read out from the channel events and tolerates a higher interrupt latency. The result register structure allows to use a single DMA channel for a complete auto-scan sequence by triggering the read out by a result event (if the conversion results of all channels taking part in the auto-scan sequence target the same result register, e.g. with FIFO mechanism or with a wait-for-read condition to avoid data loss).
- **Request source events:**
  A request source event is detected if a scan source has completely finished the requested conversion sequence. For a sequential source, the user can define where inside a conversion sequence a request source event is generated.
  This type of event can be used to inform the CPU that a conversion sequence has reached a defined state and SW can start the treatment of the related results in a block.

Each ADC event is indicated by a dedicated flag that can be cleared by SW. An interrupt can be generated (if enabled) for each event, independently from the status of the corresponding event indication flag. This structure ensures efficient DMA handling of ADC events (the ADC event can generate an interrupt without the need to clear the indication flag). A node pointer mechanism allows the user to group interrupts events by selecting which service request output signals SRx becomes activated by which event. Each ADC event can be individually directed to one of the service request output signals to adapt easily to application needs.

Note: A conversion can lead to three interrupts, one of each type. In this case, the ADC module first triggers the request source event interrupt, then the channel event interrupt, followed by the result event interrupt (all within a few f_{ADC} clock cycles).

23.1.8 Electrical Models

Each conversion of an analog input voltage into a digital value consists of two consecutive phases. During the sample phase, the input voltage is sampled and prepared for the following conversion phase. A simplified model for the sample phase describes the input signal path, whereas a second simplified model for the conversion phase is related to the reference voltage handling.

23.1.8.1 Input Signal Path

The ADC kernel in the TC1797 is based on one switched capacitor field for measurement with a total capacity represented by C_{AIN} and a small static capacitor at each input pin. During the sample phase, the capacitor field C_{AIN} is connected to one of the analog input CHx via an input multiplexer. The multiplexer is modeled by ideal switches and series resistors R_{AIN}. Only the switch to the selected analog input is closed during the sample phase. During the conversion phase or while no conversion is running (ADC is idle), all switches are open. The voltage at the analog input channel CHx is represented by V_{AINx}.

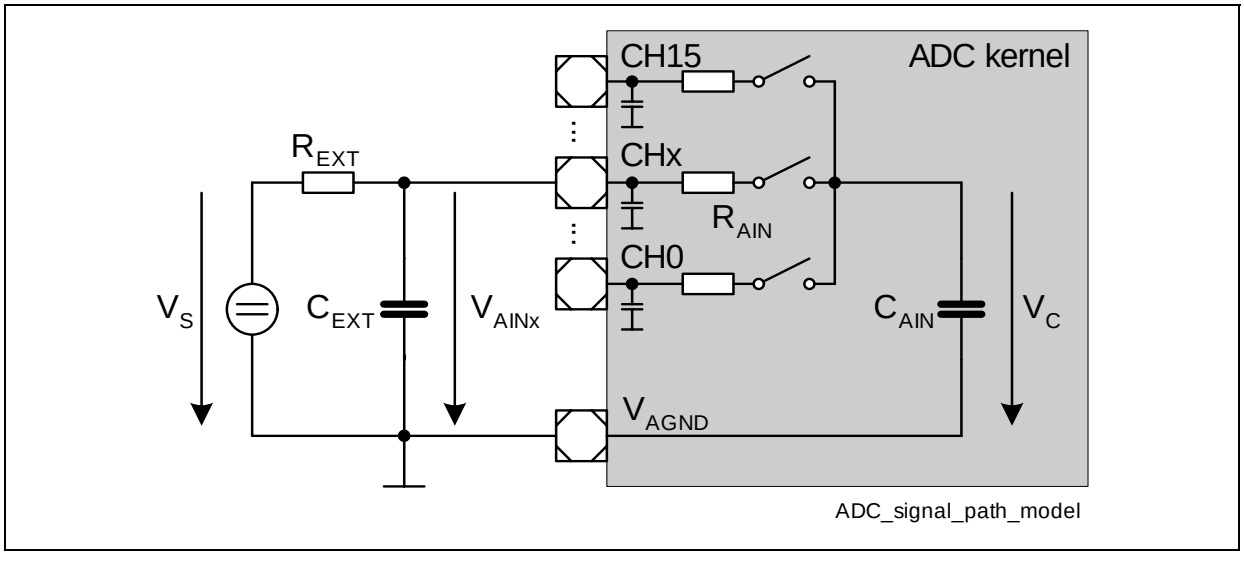


Figure 23-4 Signal Path Model

A simplified model for the analog input signal path is given in **Figure 23-4**. An analog voltage source (value V_S) with an internal impedance of R_{EXT} delivers the analog input that should be converted.

During the sample phase the corresponding switch is closed and the capacitor field C_{AIN} is charged. Due to the low-pass behavior of the resulting RC combination, the voltage V_C to be actually converted does not immediately follow V_S. The value R_{EXT} of the analog voltage source and the desired precision of the conversion strongly define the required length of the sample phase.

To reduce the influence of R_{EXT} and to filter input noise, it is recommended to introduce

a fast external blocking capacitor C_{EXT} at the analog input pin of the ADC. Like this, mainly C_{EXT} delivers the charge during the sample phase. This structure allows a significantly shorter sample phase than without a blocking capacitor, because the low-pass time constant defining the sample time is mainly given by the values of R_{AIN} and C_{AIN}.

Additionally, the capacitor C_{AIN} is automatically precharged to a voltage of approximately the half of the standard reference voltage V_{AREF} to minimize the average difference between V_{AINx} and V_C at the beginning of a sample phase. Due to varying parameters and parasitic effects, the precharge voltage of C_{AIN} is typically smaller than V_{AREF} / 2.

On the other hand, the charge redistribution between C_{EXT} and C_{AIN} leads to a voltage change of V_{AINx} during the sample phase. In order to keep this voltage change lower than 1 LSB_n, it is recommended to use an external blocking capacitor C_{EXT} in the range of at least $2^n \times C_{AIN}$.

The resulting low-pass filter of R_{EXT} and C_{EXT} should be dimensioned in a way to allow V_{AINx} to follow V_S between two sample phases of the same analog input channel.

Please note that, especially at high temperatures, the analog input structure of an ADC can lead to a leakage current and introduces an error due to a voltage drop over R_{EXT}. The ADC input leakage current increases if the input voltage level is close to the analog supply ground V_{SSM} or to the analog power supply V_{DDM}. It is recommended to use an operating range for the input voltage between approximately 3% and 97% of V_{DDM} to reduce input leakage values.

Furthermore, the leakage is influenced by an overload condition at adjacent analog inputs. During an overload condition, an input voltage exceeding the supply range is applied at an input and the built-in protection circuit limits the resulting input voltage. This leads to an overload current through the protection circuit that is translated (by a coupling factor) into an additional leakage at adjacent inputs.

23.1.8.2 Reference Path

During the conversion phase, parts of the capacitor field C_{AIN} are switched to a reference input or to V_{AGND}. The ADC kernel supports two possible reference inputs, V_{AREF} as standard reference and CH0 as alternative reference. The reference selection between both possibilities is handled individually for each analog input channel. For example, this structure allows conversions of 5 V and 3.3 V based analog input signals with the same ADC kernel.

A high accuracy of the conversion results requires a stable and noise-free reference voltage and analog supply voltages during the conversion phase. Instable voltages or noise on the supply or reference inputs lead to a reduced conversion accuracy. Please note that noise can also be introduced into the ADC module by other modules, e.g. by switching of neighboring pins. It is strongly recommended to carefully decouple analog from digital signal domains.

Due to the switching of parts of C_{AIN}, the ADC requires a dynamic current at the selected reference input. Thus, the impedance R_{AREF} of the reference voltage source V_R has to

be low enough to supply the reference current during the conversion phase. An external blocking capacitor C_{AREF} should be used to supply the peak currents and to minimize the current delivered by the reference source.

Due to the charge redistribution between C_{AREF} and parts of C_{AIN}, the voltage V_{AREF} decreases during the conversion phase. In order to limit the error introduced by this effect to 1/2 LSB_n, the external blocking capacitor C_{AREF} for the reference input should be at least $2^n \times C_{AIN}$.

The reference current I_{AREF} introduces a voltage drop at R_{AREF} that should not be neglected for the calculation of the overall accuracy. The average reference current during a conversion depends on the reference voltage level and the time t_{CONV} between two conversion starts.

$I_{AREF} = C_{AIN} \times V_{AREF} / t_{CONV}$

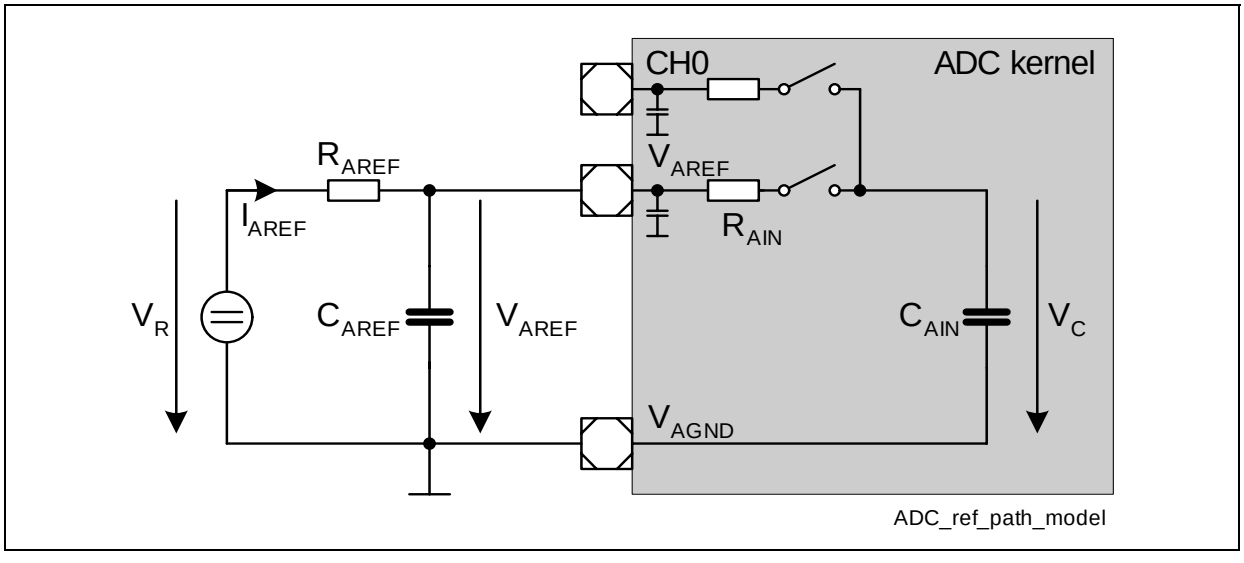


Figure 23-5 Reference Path Model

23.1.9 Transfer Characteristics and Error Definitions

The ideal transfer characteristic of the ADC translates a continuous analog input voltage into a discrete digital value out of a result range of 2^n steps for n bit resolution over a measurement range between 0 and a reference voltage. Each digital value in the available result range (from 0 to 2^n-1) represents an input voltage range that is defined by the reference voltage divided by 2^n. This range (called quantization step) represents the smallest granularity (called LSB_n) that can be handled by the ADC. Due to the discrete character of the digital result, each ADC conversion result has a system-inherent quantization uncertainty of ± 0.5 LSB_n. According to the ideal transfer characteristics, the first digital transition (between the digital values 0 and 1) takes place when the analog input reaches 0.5 LSB_n.
An analog input voltage above the reference voltage leads to a saturation of the digital result at 2^n-1.

Deviations of the conversion result from the ideal transfer characteristics can appear:

- An **offset error** is the deviation from the ideal transfer characteristics for an input voltage close to 0. It describes the difference between 0.5 LSB_n and the input voltage where the first digital transition (between the values of 0 and 1) occurs.
- A **gain error** is the deviation from the ideal transfer characteristics for an input voltage close to the reference voltage. It describes the difference between the reference voltage and the input voltage where the last digital transition (between the values of 2^n-2 and 2^n-1) occurs.
- A **differential non-linearity error** (DNL) describes the variations in the analog input voltage between two adjacent digital conversion results, over the full measurement range. If each step between the digital conversion results x and x+1 is exactly 1 LSB_n, the DNL value is zero. If the DNL value is lower than 1 LSB_n, the possibility of missing codes is excluded. A missing code occurs if not all values of the possible conversion result range can be reached.
- An **integral non-linearity error** (INL) describes the maximum difference between the transfer characteristics between the first and the last point of the measurement range and the real transfer characteristics (without quantization uncertainty, offset and gain errors).
- The **total unadjusted error** (TUE) describes the maximum deviation between a real conversion result and the ideal transfer characteristics over a given measurement range. Since some of these errors noted above can compensate each other, the TUE value generally is much less than the sum of the individual errors.
  The TUE also covers production process variations and internal noise effects (if switching noise is generated by the system, this generally leads to an increased TUE value).

23.2 Operating the ADC

This section describes the kernel functions and how to operate the kernel. It provides the functional descriptions and the associated register descriptions.

- Register overview (see **Section 23.2.1**)

General module, kernel and arbiter operation:

- Enabling the ADC module for configuration of the behavior for the different device operating modes (see mode control description in **Section 23.2.2**).
- Enabling the converter for operation or selecting the desired power saving mode (see **Section 23.2.3**)
- Selecting the appropriate frequency for the converter and for the request source arbiter (see **Section 23.2.4**).
- ADC module registers (see **Section 23.2.5**)
- General ADC kernel registers (see **Section 23.2.6**)
- Configuring the request source arbiter (see **Section 23.2.7**)
- Arbiter registers (see **Section 23.2.8**)

Request source operation:

- Scan request source handling (see **Section 23.2.9**)
- Scan request source registers (see **Section 23.2.10**)
- Sequential request source handling (see **Section 23.2.11**)
- Sequential request source registers (see **Section 23.2.12**)

Channel and result register operation:

- Configuring the channel-related functions (see **Section 23.2.13**)
- Channel-related registers (see **Section 23.2.14**)
- Conversion result handling (see **Section 23.2.15**)
- Conversion request handling (see **Section 23.2.16**)

Additional features:

- Multiplexer test support (see **Section 23.2.17**)
- External multiplexer control (see **Section 23.2.18**)
- Synchronization for parallel conversions (see **Section 23.2.19**)
- Equidistant sampling (see **Section 23.2.20**)
- Access protection (see **Section 23.2.21**)
- Additional feature registers (see **Section 23.2.22**)

23.2.1 Register Overview

Table 23-2 shows all registers required for programming the ADC module. It summarizes the ADC kernel registers and defines their relative addresses and the reset values. The relative addresses have to be added to the base addresses for the ADC kernels (see **Section 23.3**) to obtain the absolute address for each register. Each ADC kernel is located in an address window of 4 * 256 bytes.
All registers can be accessed with 8 bit, 16 bit, or 32 bit wide accesses.

The prefix "**ADCx**_" has to be added to the register names in this table for each ADC kernel to distinguish registers of the different kernels. In this naming convention, x indicates the kernel number (e.g. ADC0_ for the ADC0 kernel and "ADC1_" for the ADC1 kernel).
The registers that are implemented only once in the ADC module are located in the address range of ADC0.

All ADC registers (including KSCFG.NOMCFG and KSCFG.COMCFG) are reset by an application reset (class 3), whereas bit field KSCFG.SUMCFG is reset by a debug reset (class 1).

Note: Register bits marked "w" always deliver 0 when read.

Access rights within the address range of an ADC kernel:

- Read or write access to defined register addresses: U, SV
- Accesses to empty addresses: reserved, BE

Table 23-2 Register Overview of ADC

Short Name	Description	Offset Addr.[1)]	Access Mode		Reset	Description See
			Read	Write		
ADC Module Registers (only available in the address range of ADC0)						
CLC	Clock Control Register	000_H	U, SV	SV, E	Class 3	**Page 23-25**
KSCFG	Kernel State Configuration Register	$00C_H$	U, SV	SV, E	Class 3	**Page 23-26**
SRCx x = 0 - 8	Service Request Control Registers	$3FC_H$ - x * 4_H	U, SV	U, SV	Class 3	**Page 23-28**
General Kernel Registers (available in the address range of each kernel)						
ID	Module Identification Register	008_H	U, SV	U, SV	Class 3	**Page 23-32**
RSIRx (x = 0 - 4)	Request Source x Input Register	010_H + x * 4	U, SV	U, SV	Class 3	**Page 23-29**

Table 23-2 Register Overview of ADC

Short Name	Description	Offset Addr.[1)]	Access Mode		Reset	Description See
			Read	Write		
INTR	Interrupt Activation Register	204_H	U, SV	U, SV	Class 3	Page 23-33
GLOBCTR	Global Control Register	030_H	U, SV	U, SV	Class 3	Page 23-34
GLOBCFG	Global Configuration Register	034_H	U, SV	U, SV	Class 3	Page 23-37
GLOBSTR	Global Status Register	038_H	U, SV	U, SV	Class 3	Page 23-39
Arbiter Registers (available in the address range of each kernel)						
ASENR	Arbitration Slot Enable Register	$03C_H$	U, SV	U, SV	Class 3	Page 23-47
RSPR0	Request Source Priority Register 0	040_H	U, SV	U, SV	Class 3	Page 23-48
RSPR4	Request Source Priority Register 4	044_H	U, SV	U, SV	Class 3	Page 23-49
Request Source 0 Registers (available in the address range of each kernel)						
QMR0	Queue 0 Mode Register	080_H	U, SV	U, SV	Class 3	Page 23-65
QSR0	Queue 0 Status Register	084_H	U, SV	U, SV	Class 3	Page 23-68
Q0R0	Queue 0 Register 0	088_H	U, SV	U, SV	Class 3	Page 23-70
QBUR0	Queue 0 Backup Register	$08C_H$	U, SV	U, SV	Class 3	Page 23-72
QINR0	Queue 0 Input Register	$08C_H$	U, SV	U, SV	Class 3	Page 23-74
Request Source 1 Registers (available in the address range of each kernel)						
CRCR1	Conversion Request 1 Control Register	090_H	U, SV	U, SV	Class 3	Page 23-54

Table 23-2 Register Overview of ADC

Short Name	Description	Offset Addr.[1)]	Access Mode		Reset	Description See
			Read	Write		
CRPR1	Conversion Request 1 Pending Register	094_H	U, SV	U, SV	Class 3	Page 23-56
CRMR1	Conversion Request 1 Mode Register	098_H	U, SV	U, SV	Class 3	Page 23-57
Request Source 2 Registers (available in the address range of each kernel)						
QMR2	Queue 2 Mode Register	$0A0_H$	U, SV	U, SV	Class 3	Page 23-65
QSR2	Queue 2 Status Register	$0A4_H$	U, SV	U, SV	Class 3	Page 23-68
Q0R2	Queue 2 Register 0	$0A8_H$	U, SV	U, SV	Class 3	Page 23-70
QBUR2	Queue 2 Backup Register	$0AC_H$	U, SV	U, SV	Class 3	Page 23-72
QINR2	Queue 2 Input Register	$0AC_H$	U, SV	U, SV	Class 3	Page 23-74
Request Source 3 Registers (available in the address range of each kernel)						
CRCR3	Conversion Request 3 Control Register	$0B0_H$	U, SV	U, SV	Class 3	Page 23-54
CRPR3	Conversion Request 3 Pending Register	$0B4_H$	U, SV	U, SV	Class 3	Page 23-56
CRMR3	Conversion Request 3 Mode Register	$0B8_H$	U, SV	U, SV	Class 3	Page 23-57
Request Source 4 Registers (available in the address range of each kernel)						
QMR4	Queue 4 Mode Register	$0C0_H$	U, SV	U, SV	Class 3	Page 23-65
QSR4	Queue 4 Status Register	$0C4_H$	U, SV	U, SV	Class 3	Page 23-68
Q0R4	Queue 4 Register 0	$0C8_H$	U, SV	U, SV	Class 3	Page 23-70

Table 23-2 Register Overview of ADC

Short Name	Description	Offset Addr.[1)]	Access Mode		Reset	Description See
			Read	Write		
QBUR4	Queue 4 Backup Register	$0CC_H$	U, SV	U, SV	Class 3	Page 23-72
QINR4	Queue 4 Input Register	$0CC_H$	U, SV	U, SV	Class 3	Page 23-74
Channel Registers (available in the address range of each kernel)						
CHCTRx (x = 0 -15)	Channel x Control Register	100_H + x * 4	U, SV	U, SV	Class 3	Page 23-81
INPCRx (x = 0 -3)	Input Class x Register	050_H + x * 4	U, SV	U, SV	Class 3	Page 23-83
ALR0	Alias Register 0	210_H	U, SV	U, SV	Class 3	Page 23-84
LCBRx (x = 0 - 3)	Limit Check Boundary Register x	$0F0_H$ + x * 4	U, SV	U, SV	Class 3	Page 23-85
CHFR	Channel Flag Register	060_H	U, SV	U, SV	Class 3	Page 23-86
CHFCR	Channel Flag Clear Register	064_H	U, SV	U, SV	Class 3	Page 23-87
CHENPR0	Channel Event Node Pointer Register 0	068_H	U, SV	U, SV	Class 3	Page 23-88
CHENPR8	Channel Event Node Pointer Register 8	$06C_H$	U, SV	U, SV	Class 3	Page 23-89
Result Registers (available in the address range of each kernel)						
RESR0	Result Register 0	180_H	U, SV	U, SV	Class 3	Page 23-98
RESRx (x = 1 - 15)	Result Register x	180_H + x * 4	U, SV	U, SV	Class 3	Page 23-100
RESRD0	Result Register 0 for Debugging	$1C0_H$	U, SV	U, SV	Class 3	Page 23-98
RESRDx (x = 1 - 15)	Result Register x for Debugging	$1C0_H$ + x * 4	U, SV	U, SV	Class 3	Page 23-100

Table 23-2 Register Overview of ADC

Short Name	Description	Offset Addr.[1]	Access Mode		Reset	Description See
			Read	Write		
VFR	Valid Flag Register	200_H	U, SV	U, SV	Class 3	Page 23-102
RCRx (x = 0 - 15)	Result Control Register x	140_H + x * 4	U, SV	U, SV	Class 3	Page 23-103
EVFR	Event Flag Register	070_H	U, SV	U, SV	Class 3	Page 23-105
EVFCR	Event Flag Clear Register	074_H	U, SV	U, SV	Class 3	Page 23-107
EVNPR	Source Event Node Pointer Register	078_H	U, SV	U, SV	Class 3	Page 23-108
RNPR0	Result Node Pointer Register 0	208_H	U, SV	U, SV	Class 3	Page 23-109
RNPR8	Result Node Pointer Register 8	$20C_H$	U, SV	U, SV	Class 3	Page 23-110
Additional Feature Registers (available in the address range of each kernel)						
APR	Access Protection Register	218_H	U, SV	SV, E	Class 3	Page 23-121
EMCTR	External Multiplexer Control Register	220_H	U, SV	U, SV	Class 3	Page 23-122
SYNCTR	Synchronization Control Register	048_H	U, SV	U, SV	Class 3	Page 23-126

1) The absolute register address is calculated as follows:
Module Base Address + Offset Address (shown in this column)

23.2.2 Mode Control

The mode control concept for system control tasks, such as suspend request for debugging, allows to program the module behavior under different device operating conditions. The behavior of the ADC kernels can be programmed for each of the device operating modes. It is advantageous that the ADC kernels of an ADC module show an identical behavior regarding the device operating modes (e.g. to avoid that a non-suspended kernel waits for a suspended kernel to start a synchronized conversion). Therefore, the ADC module has a common associated register **ADC0_KSCFG** defining the behavior of all kernels of the module in the following device operating modes:

- **Normal operation:**
  This operating mode is the default operating mode when no suspend request is pending. The kernel behavior is defined by KSCFG.NOMCFG.
- **Suspend mode:**
  This operating mode is requested when a suspend request (issued by a debugger) is pending in the device. The kernel behavior is defined by KSCFG.SUMCFG.

For the ADC module, the following internal actions can be influenced by mode control:

- A current conversion of an analog value:
  If the request control unit has found a pending conversion request, the conversion can be started. This start has to be enabled by the mode control. If the current kernel mode allows the conversion start (run modes 0 and 1), it will be executed. If the kernel mode does not allow a start (stop modes 0 and 1), the conversion is not started. The start request is not cancelled, but frozen. A “frozen” conversion is started as programmed if the kernel mode is changed to a run mode again.
- An arbiter round:
  The start of a new arbiter round has to be enabled by the kernel modes. In stop mode 1, a new arbiter round will not start.

The behavior of the ADC kernels can be programmed for each of the device operating modes (normal operation, suspend mode). Therefore, the ADC kernels support four kernel modes, as shown in **Table 23-3**.

Table 23-3 ADC Kernel Behavior

Kernel Mode	Kernel Behavior	Code
run mode 0	kernel operation as specified, no impact on data transfer (same behavior for run mode 0 and run mode 1)	00_B
run mode 1		01_B

Table 23-3 ADC Kernel Behavior (cont'd)

Kernel Mode	Kernel Behavior	Code
stop mode 0	A currently running AD conversion is completely finished and the result is treated. Pending conversion request to start a new conversion are not taken into account (but not deleted). They start conversions after entering a run mode as programmed. The arbiter continues as programmed.	10_B
stop mode 1	Like stop mode 0, but the arbiter is stopped after it has finished its arbitration round.	11_B

Generally, bit field KSCFG.NOMCFG should be configured for run mode 0 as default setting for standard operation. If the ADC kernels should not react to a suspend request (and to continue operation as in normal mode), bit field KSCFG.SUMCFG has to be configured with the same value as KSCFG.NOMCFG. If the ADC kernels should show a different behavior and stop operation when a specific stop condition is reached, the code for stop mode 0 or stop mode 1 has to be written to KSCFG.SUMCFG.

Note: The stop mode selection strongly depends on the application needs and it is very unlikely that different stop modes are required in parallel in the same application. As a result, only one stop mode type (either 0 or 1) should be used in the bit fields in register KSCFG. Do not mix stop mode 0 and stop mode 1 and avoid transitions from stop mode 0 to stop mode 1 (or vice versa) for the ADC module.

23.2.3 Module Activation and Power Saving Modes

The converter of the ADC supports specific power down modes allowing an automatic reduction of the power consumption between two conversions. The following modes are determined by bit field **GLOBSTR**.ANON:

- ANON = 00_B: **Converter switched off** (default after reset)
  The complete converter is switched off and held in its reset state, conversions are not possible. To start a conversion, ANON has to be programmed to the desired mode. A maximum wake-up time of about 10 µs has to be respected before starting a conversion. Furthermore, digital logic blocks are set to their initial state.
- ANON = 01_B and 10_B: **Reserved**
  These modes are reserved and must not be selected.
- ANON = 11_B: **Normal operation**
  Conversions are always possible with the desired sample time. The converter stays active permanently.

23.2.4 Clocking Scheme

The different parts of an ADC kernel are driven by clock signals that are based on the clock f_{ADC} of the bus that is used to access the ADC module.

- The analog clock f_{ADCI} is used as internal clock for the converter and defines the conversion length and the sample time. It can be adjusted by programming bit field **GLOBCTR**.DIVA.
- The digital clock f_{ADCD} is used for the arbiter and defines the duration of an arbiter round. It can be adjusted by programming bit field **GLOBCTR**.DIVD.
- All other digital structures (such as interrupts, etc.) are directly driven by the module clock f_{ADC}.

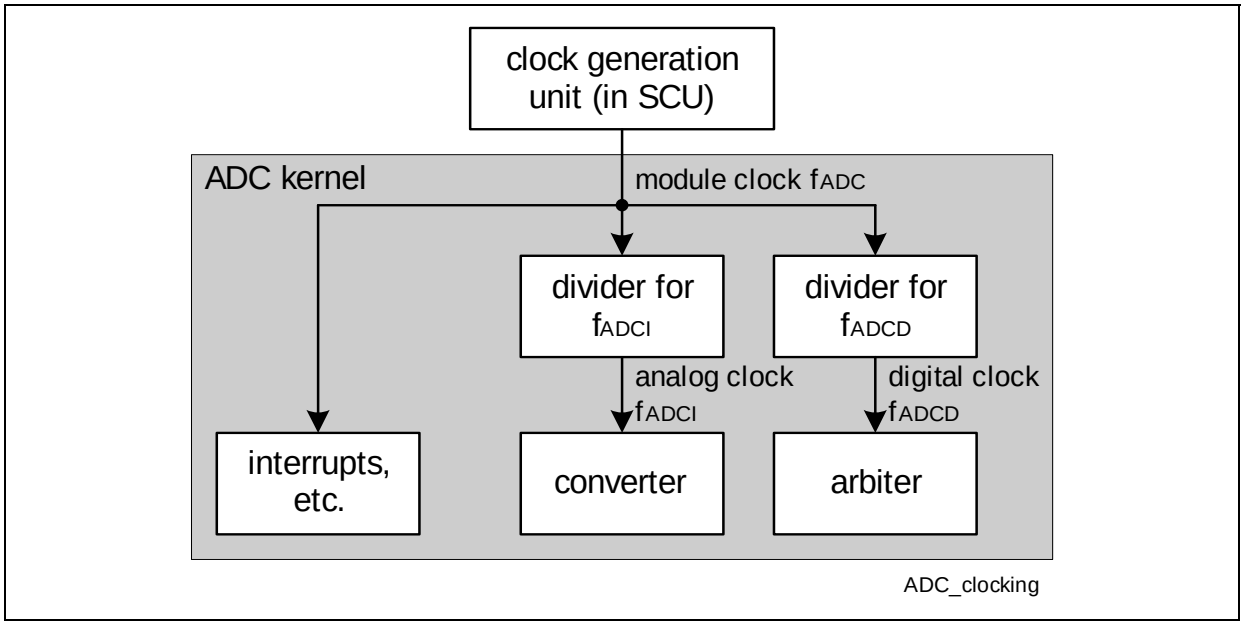


Figure 23-6 Clocking Scheme

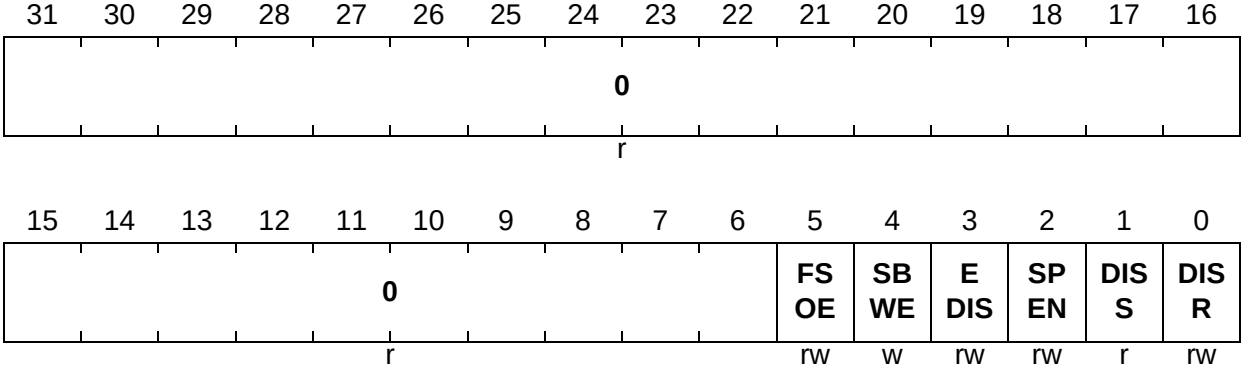
Note: If the clock generation for the converter of the ADC falls below a minimum value or is stopped during a running conversion, the conversion result can be corrupted. For correct ADC results, the frequency of f_{ADCI} must not exceed the range indicated in the electrical characteristics chapter.

23.2.5 ADC Module Registers

23.2.5.1 Clock Control Register

The clock control register allows the programmer to control (enable/disable) the clock signals to the ADC module.

ADC0_CLC
ADC Clock Control Register **(000_H)** **Reset Value: 0000 0003$_H$**



Field	Bits	Type	Description
DISR	0	rw	**Module Disable Request Bit** Used for enable/disable control of the module.
DISS	1	r	**Module Disable Status Bit** Bit indicates the current status of the module.
SPEN	2	rw	**Module Suspend Enable for OCDS** Used to enable the suspend mode
EDIS	3	rw	**Sleep Mode Enable Control** Used to control module's sleep mode.
SBWE	4	w	**Module Suspend Bit Write Enable for OCDS** Determines whether SPEN and FSOE are write-protected.
FSOE	5	rw	**Fast Switch Off Enable** Used to switch off fast clock in Suspend Mode.
0	[31:6]	r	**Reserved;** returns 0 if read; should be written with 0.

Note: After a hardware reset operation, the f_{CLC} and f_{ADC} clocks are switched off and the ADC module is disabled (DISS set).

23.2.5.2 Kernel State Configuration Register

The kernel state configuration register KSCFG allows the selection of the desired kernel modes. The ADC kernels are controlled by the same register ADC0_KSCFG.
All bits in KSCFG, except KSCFG.SUMCFG are reset by an application (class3) reset. Bit field KSCFG.SUMCFG is reset by the debug reset.
If a module should be switched off, SW can program a stop mode in bit field NOMCFG and check for ACK = 1 afterwards.

*Note: The coding of the bit fields NOMCFG, SUMCFG and COMCFG is described in **Table 23-3**.*

ADC0_KSCFG
Kernel State Configuration Register (00C$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				BP SUM	0	SUMCFG		BP NOM	0	NOMCFG		SUS REQ	ACK	0	
r				w	r	rw		w	r	rw		rh	rh	r	

Field	Bits	Type	Description
ACK	2	rh	**Module Acknowledge** This bit monitors the state of the ADC module's acknowledge on incoming requests. 0_B The acknowledge is not activated, because at least one of the module kernels is in a transition phase. 1_B The acknowledge is activated, because all module kernels have reached the requested state.
SUSREQ	3	rh	**Suspend Request** This bit monitors the state of the ADC module's suspend request input. 0_B A suspend mode is not requested and bit field NOMCFG defines the ADC kernel mode. 1_B A suspend mode is requested and bit field SUMCFG defines the ADC kernel mode.

Field	Bits	Type	Description
NOMCFG	[5:4]	rw	**Normal Operation Mode Configuration** This bit field defines the kernel mode applied in normal operation mode. 00_B Run mode 0 is selected. 01_B Run mode 1 is selected. 10_B Stop mode 0 is selected. 11_B Stop mode 1 is selected.
BPNOM	7	w	**Bit Protection for NOMCFG** This bit enables the write access to the bit field NOMCFG. It always reads 0. 0_B The bit field NOMCFG is not changed. 1_B The bit field NOMCFG is updated with the written value.
SUMCFG	[9:8]	rw	**Suspend Mode Configuration** This bit field defines the kernel mode applied in suspend mode. Coding like NOMCFG.
BPSUM	11	w	**Bit Protection for SUMCFG** This bit enables the write access to the bit field SUMCFG. It always reads 0. 0_B The bit field SUMCFG is not changed. 1_B The bit field SUMCFG is updated with the written value.
0	[1:0], 6, 10, [31:12]	r	**Reserved** Read as 0; should be written with 0.

23.2.5.3 Service Request Control Registers

The service request control registers of the ADC module are located in the address range of ADC0.

ADC0_SRCx (x = 0-8)
ADC Service Request Control Register x

($3FC_H$ - x*4) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SETR	CLRR	SRR	SRE	0	TOS	0		SRPN							
w	w	rh	rw	r	rw	r		rw							

Field	Bits	Type	Description
SRPN	[7:0]	rw	**Service Request Priority Number**
TOS	10	rw	**Type of Service Control**
SRE	12	rw	**Service Request Enable**
SRR	13	rh	**Service Request Flag**
CLRR	14	w	**Request Clear Bit**
SETR	15	w	**Request Set Bit**
0	[9:8], 11, [31:16]	r	**Reserved** Read as 0; should be written with 0.

Note: Additional details on service request nodes and the service request control registers are described in the TC1797 System Units part (Volume 1).

23.2.6 General ADC Kernel Registers

23.2.6.1 Request Source Input Registers

The setting of the request source input registers selects the desired input signal for the gating and trigger signals of the request sources. The status of the selected inputs is monitored. Additionally, the edge sensitivity for the trigger signal and the timer mode for equidistant sampling can be enabled/disabled.
The actual connections depending on the device implementation, please refer to the implementation description in **Section 23.3** for details.

Note: Signals from a synchronous domain can of course be connected to inputs with a synchronization stage. The additional synchronization delay of two ADC module clock cycles and an additional uncertainty of one ADC module clock cycle for asynchronous signals have to be taken into account when using a synchronization stage.

RSIRx (x = 0 - 4)
Request Source x Input Register (010_H + x * 4) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

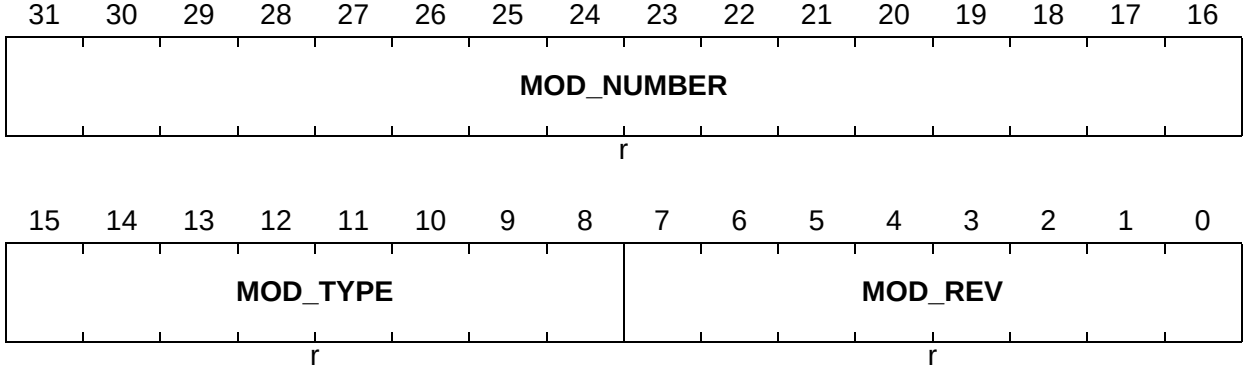
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRI	0	R EN	F EN	0	TRSEL			GTI	0		TM EN	0	GTSEL		
rh	r	rw	rw	r	rw			rh	r		rw	r	rw		

Field	Bits	Type	Description
GTSEL	[2:0]	rw	**Input Selection for REQGT of Source x** This bit field defines the input signal used for request gating in request source x. The inputs selected by codes $0XX_H$ are considered as synchronous to the ADC module. The inputs selected by codes $1XX_H$ are considered as asynchronous to the ADC module. 000_B The input signal REQGTx_0 is selected. 001_B The input signal REQGTx_1 is selected. 010_B The input signal REQGTx_2 is selected. 011_B The input signal REQGTx_3 is selected. 100_B The input signal REQGTx_4 is selected. 101_B The input signal REQGTx_5 is selected. 110_B The input signal REQGTx_6 is selected. 111_B The input signal REQGTx_7 is selected.
TMEN	4	rw	**Timer Mode Enable of Source x** This bit enables the timer mode for equidistant sampling for request source x. 0_B The timer mode is disabled. The standard gating mechanism can be used. 1_B The timer mode for equidistant sampling is enabled. The standard gating mechanism has to be disabled.
GTI	7	rh	**Gating Input of Source x** This flag monitors the status of the selected gating signal REQGTx for request source x. 0_B The selected gating signal is 0. 1_B The selected gating signal is 1.

Field	Bits	Type	Description
TRSEL	[10:8]	rw	**Input Selection for REQTR of Source x** This bit field defines the input signal used for request triggering in request source x. The inputs selected by codes $0XX_H$ are considered as synchronous to the ADC module. The inputs selected by codes $1XX_H$ are considered as asynchronous to the ADC module. 000_B The input signal REQTRx_0 is selected. 001_B The input signal REQTRx_1 is selected. 010_B The input signal REQTRx_2 is selected. 011_B The input signal REQTRx_3 is selected. 100_B The input signal REQTRx_4 is selected. 101_B The input signal REQTRx_5 is selected. 110_B The input signal REQTRx_6 is selected. 111_B The input signal REQTRx_7 is selected.
FEN	12	rw	**Falling Edge Enable of Source x** This bit enables the request trigger for falling edges of the selected REQTRx signal for request source x. 0_B The request trigger with a falling edge is disabled. 1_B The request trigger with a falling edge is enabled.
REN	13	rw	**Rising Edge Enable of Source x** This bit enables the request trigger for rising edges of the selected REQTRx signal for request source x. 0_B The request trigger with a rising edge is disabled. 1_B The request trigger with a rising edge is enabled.
TRI	15	rh	**Trigger Input of Source x** This flag monitors the status of the selected trigger signal REQTRx for request source x. 0_B The selected trigger signal is 0. 1_B The selected trigger signal is 1.
0	3, [6:5], 11, 14, [31:16]	r	**Reserved** Read as 0; should be written with 0.

23.2.6.2 Module Identification Register

ID
Module Identification Register (008$_H$) Reset Value: 0059 C000$_H$

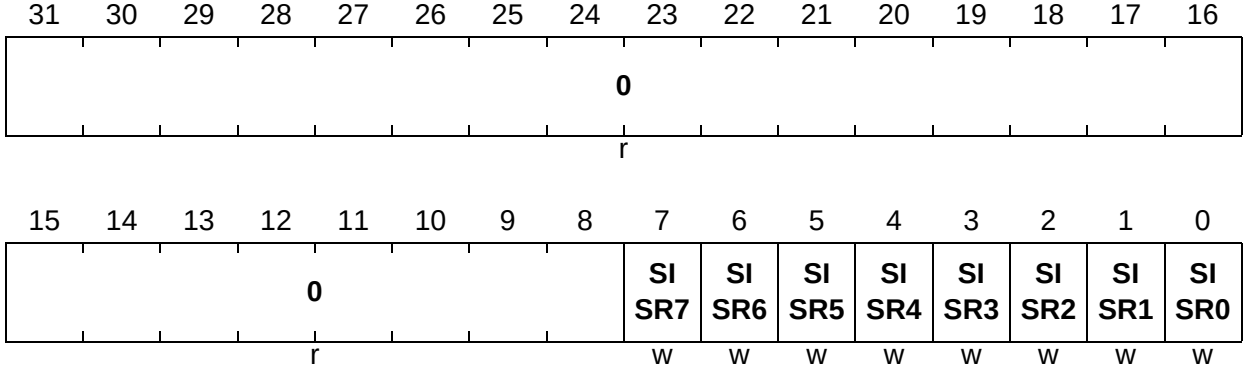


Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision** This bit field indicates the revision number of the module implementation (depending on the design step). The given value of 00$_H$ is a placeholder for the actual number. Bits [3:0] refer to the version of the digital part and bits [7:4] indicate the version of the analog part (anid).
MOD_TYPE	[15:8]	r	**Module Type**
MOD_NUMBER	[31:16]	r	**Module Number**

23.2.6.3 Interrupt Activation Register

The interrupt activation register contains bit locations allowing to activate one or more service request outputs SR[7:0] of the ADC kernel. Writing a 1 to a bit position x activates the corresponding SRx line. All bit positions read as 0.

INTR
Interrupt Activation Register (204$_H$) Reset Value: 0000 0000$_H$

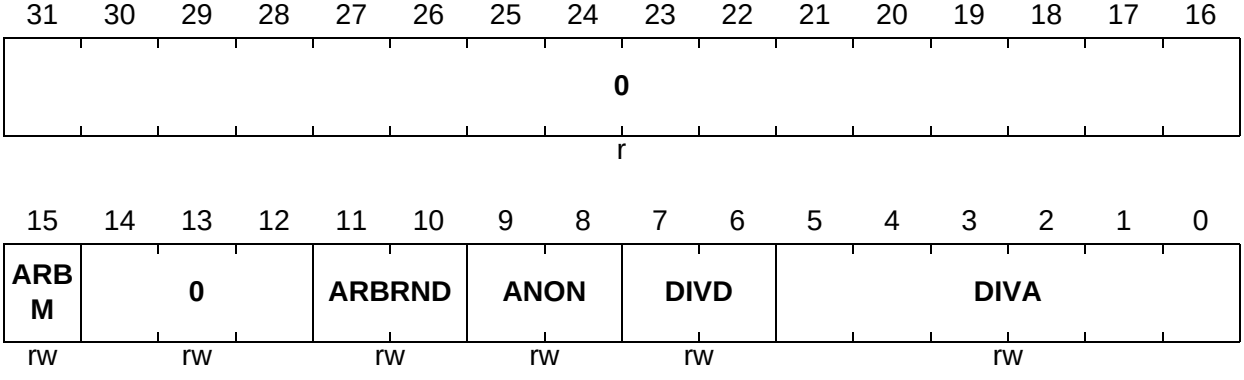


Field	Bits	Type	Description
SISRx (x = 0 - 7)	x	w	**Set Interrupt for SRx Line** Writing a 1 to a bit position sets an interrupt request at the SRx output of the ADC kernel (the activation is finished automatically). Writing a 0 has no effect. The read value is always 0. 0_B No action 1_B The service request output SRx of the ADC kernel becomes activated.
0	[31:8]	r	**Reserved** Read as 0; should be written with 0.

23.2.6.4 Global Control

The global control register contains bits to control the arbiter timing and the general enable function for the analog part.

GLOBCTR
Global Control Register **(030_H)** **Reset Value: 0000 00FF$_H$**



Field	Bits	Type	Description
DIVA	[5:0]	rw	**Divider Factor for Analog Internal Clock** This bit field defines the number of f_{ADC} clock cycles to generate the f_{ADCI} clock for the converter (used as internal base for the conversions and the sample time calculation). The minimum divider is 4. 00_H $f_{ADCI} = f_{ADC} / 4$ 01_H $f_{ADCI} = f_{ADC} / 4$ 02_H $f_{ADCI} = f_{ADC} / 4$ 03_H $f_{ADCI} = f_{ADC} / 4$ 04_H $f_{ADCI} = f_{ADC} / 4$ 05_H $f_{ADCI} = f_{ADC} / 5$ 06_H $f_{ADCI} = f_{ADC} / 6$ 07_H $f_{ADCI} = f_{ADC} / 7$ ... $3F_H$ $f_{ADCI} = f_{ADC} / 63$

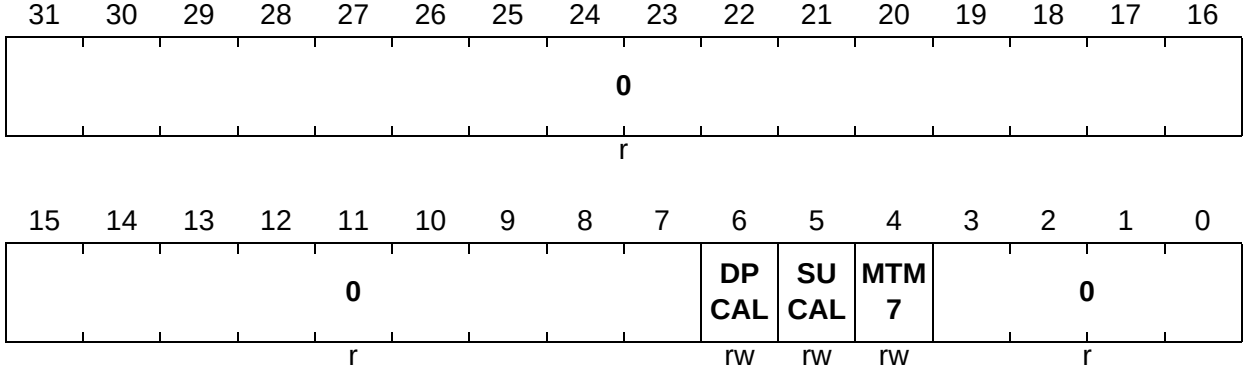
Field	Bits	Type	Description
DIVD	[7:6]	rw	**Divider Factor for Digital Arbiter Clock** This bit field defines the number of f_{ADC} clock cycles within each arbitration slot (each arbitration slots last one periods of f_{ADCD}). It is recommended to use the default setting 00_B to obtain the minimum arbiter reaction time. 00_B $f_{ADCD} = f_{ADC}$ 01_B $f_{ADCD} = f_{ADC} / 2$ 10_B $f_{ADCD} = f_{ADC} / 3$ 11_B $f_{ADCD} = f_{ADC} / 4$
ANON	[9:8]	rw	**Analog Part Switched On** This bit field defines the setting of bit field **GLOBSTR**.ANON (bit description see there) if this kernel is the synchronization master or without synchronization feature. For a synchronization slave, this bit field is not taken into account.
ARBRND	[11:10]	rw	**Arbitration Round Length** This bit field defines the number of arbitration slots per arbitration round (arbitration round length = t_{ARB}). 00_B An arbitration round contains 4 arbitration slots ($t_{ARB} = 4 / f_{ADCD}$). 01_B An arbitration round contains 8 arbitration slots ($t_{ARB} = 8 / f_{ADCD}$). 10_B An arbitration round contains 16 arbitration slots ($t_{ARB} = 16 / f_{ADCD}$). 11_B An arbitration round contains 20 arbitration slots ($t_{ARB} = 20 / f_{ADCD}$).
0	[14:12]	rw	**Reserved** This bit field is reserved for future use and has to be written with 000_B.

Field	Bits	Type	Description
ARBM	15	rw	**Arbitration Mode** This bit field defines whether the arbiter runs permanently or only while at least one conversion request is pending. 0_B The arbiter runs permanently. This setting has to be chosen in a synchronization slave (see **Section 23.2.19**) and for equidistant sampling using the signal ARBCNT (see **Section 23.2.20**). 1_B The arbiter only runs if at least one conversion request of an enabled request source is pending. This setting leads to a reproducible latency from an incoming request to the conversion start if the converter is idle. Synchronized conversions are not supported.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

23.2.6.5 Global Configuration

The global configuration register configures the general ADC kernel setting.

GLOBCFG
Global Configuration Register (034_H) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
MTM7	4	rw	**Multiplexer Test Mode for Channel 7** This bit enables/disables the multiplexer test mode for the input channel 7, see **Section 23.2.17**. This feature is independent of the current mode of the ADC (ANON, selected channel for conversion). 0_B The multiplexer test mode is disabled. The analog input CH7 can be used for normal measurements. 1_B The multiplexer test mode is enabled. The analog input CH7 is internally connected to ground via voltage divider based on an additional resistor.[1)]
SUCAL	5	rw	**Start-Up Calibration** The transition from 0 to 1 of this bit starts the start-up calibration phase of the analog part. This should be done after reset before starting the first conversion to reduce analog errors. During the calibration phase (indicated by **GLOBSTR**.CAL = 1), conversions must not be started. It can take a few f_{ADC} clock cycles before bit CAL is set after a rising edge of SUCAL. 0_B Start-up calibration can be started. 1_B Start-up calibration has been started.

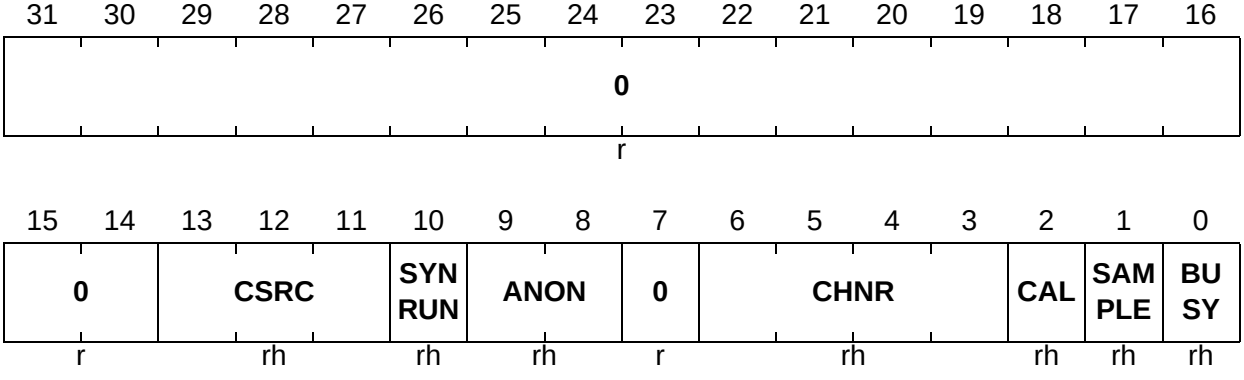
Field	Bits	Type	Description
DPCAL	6	rw	**Disable Post Calibration** This bit enables/disables the automatic post calibration of the analog part. 0_B The automatic post calibration is enabled. 1_B The automatic post calibration is disabled.
0	[31:7]	r	**Reserved** Read as 0; should be written with 0.

1) Please refer to the AC/DC chapter for the value of the grounding resistor and its current capability.

23.2.6.6 Global Status

The status control register contains bits indicating the current status of a conversion.

GLOBSTR
Global Status Register (038$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
BUSY	0	rh	**Analog Part Busy** This bit indicates that a conversion is currently active. 0_B The analog part is idle. 1_B A conversion is currently active.
SAMPLE	1	rh	**Sample Phase** This bit indicates that an analog input signal is currently sampled. 0_B The analog part is not in the sampling phase. 1_B The analog part is in the sampling phase.
CAL	2	rh	**Calibration Phase** This bit indicates that the analog part is in the startup calibration phase. 0_B The analog part is not in the calibration phase. 1_B The analog part is in the calibration phase.
CHNR	[6:3]	rh	**Channel Number** This bit field indicates which analog input channel is currently converted. This information is updated when a new conversion is started.

Field	Bits	Type	Description
ANON	[9:8]	rh	**Analog Part Switched On** This bit field defines the operation mode of the converter. It monitors either bit field GLOBCTR.ANON of the same ADC kernel (in master mode or without synchronization feature) or bit field GLOBCTR.ANON of the ADC kernel selected as synchronization master for this kernel (in slave mode). This ensures that all kernels of a synchronization group can be controlled with a single write operation to bit field GLOBCTR of the synchronization master. 00_B The analog part is switched off and conversions are not possible. To achieve a minimal power consumption, the internal analog circuitry is in its power-down state and the generation of f_{ADCI} and f_{ADCD} is stopped (counters set to an initial value). Furthermore, the arbiter finishes its current arbitration round (if running) and then remains in the idle state. 01_B reserved, do not use 10_B reserved, do not use 11_B The analog part of the ADC module is switched on and conversions are possible.
SYNRUN	10	rh	**Synchronous Conversion Running** This bit indicates that a synchronous (= parallel) conversion is currently running. 0_B There is no synchronous conversion running (either there is no conversion currently running or a parallel conversion has not been requested). A running conversion can be cancelled and repeated in case of a new incoming conversion request with higher priority. 1_B A synchronous conversion is running. This conversion can not be cancelled while running. Higher priority requests can trigger conversions only after the end of the currently running synchronous conversion.

Field	Bits	Type	Description
CSRC	[13:11]	rh	**Currently Converted Request Source** This bit field indicates the arbitration slot number of the current conversion (if BUSY = 1, a conversion is still running) or of the last conversion (if BUSY = 0, no conversion is running). This bit field is updated with each conversion start. 000_B The channel requested by the request source of arbitration slot 0 is (has been) converted. 001_B The channel requested by the request source of arbitration slot 1 is (has been) converted. ... 110_B The channel requested by the request source of arbitration slot 6 is (has been) converted. 111_B The channel requested by a synchronous injection is (has been) converted.
0	7, [31:14]	r	**Reserved** Read as 0; should be written with 0.

23.2.7 Request Source Arbiter

The request source arbiter evaluates which analog input channel has to be converted. Therefore, it regularly polls the request sources one after the other for pending conversion requests. The polling sequence is based on time slots with programmable length, called arbitration slots. If a request source is disabled or if no request source is available for an arbitration slot, the slot is considered as being empty and has no influence on the evaluation of the arbitration winner. After reset, all request sources are disabled and have to be enabled by bits in register **ASENR** to take part in the arbitration process.

The number of arbitration slots forming an arbitration round can be programmed to obtain a similar arbiter timing for different devices, even if the number of available request sources differs from one device to another. At the end of each arbitration round, the arbiter has determined the request source with the highest priority and a pending conversion request. This arbitration result is stored as arbitration winner for further actions. If a conversion is started in an arbitration round, this arbitration round does not deliver an arbitration winner.

In the TC1797, the following request sources are available:

- **Request source 0** in arbitration slot 0: **1-stage sequential source**
  This request source can issue a conversion request for a single input channel.
- **Request source 1** in arbitration slot 1: **16-channel scan source**
  This request source can issue a conversion request sequence of up to 16 input channels in a defined order.
- **Request source 2** in arbitration slot 2: **4-stage sequential source**
  This request source can issue a conversion request sequence of up to 4 input channels in a freely programmable order.
- **Request source 3** in arbitration slot 3: **16-channel scan source**
  This request source can issue a conversion request sequence of up to 16 input channels in a defined order.
- **Request source 4** in arbitration slot 4: **4-stage sequential source**
  This request source can issue a conversion request sequence of up to 4 input channels in a freely programmable order.
- Last arbitration slot of the arbitration round: **Synchronization source**
  In this slot, the arbiter checks for a synchronized request from another ADC kernel and does not evaluate any internal request source. A request for a synchronized conversion is always handled with the highest priority in a synchronization slave kernel (pending requests from other sources are not considered).

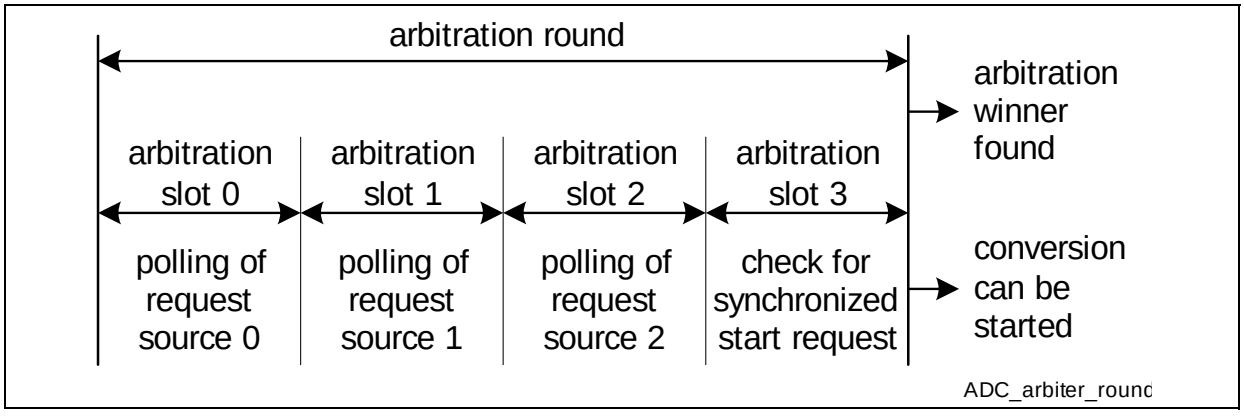


Figure 23-7 Arbitration Round with 4 Arbitration Slots

The period t_{ARB} of an arbitration round is given by:
$t_{ARB} = N \times (GLOBCTR.DIVD + 1) / f_{ADC}$
with N being 4, 8, 16, or 20 as defined by **GLOBCTR**.ARBRND

The period of the arbitration round introduces a timing granularity to detect an incoming conversion request signal and the earliest point to start the related conversion. This granularity can introduce a jitter of maximum one arbitration round. The jitter can be reduced by minimizing the period of an arbitration round (numbers of arbitration slots and their length).

To achieve a reproducible reaction time (constant delay without jitter) between the trigger event of a conversion request (e.g. by a timer unit or due to an external event) and the start of the related conversion, mainly the following two options exist. For both options, the converter has to be idle and other conversion requests must not be pending for at least one arbiter round before the trigger event occurs:

- If bit **GLOBCTR**.ARBM = 0, the **arbiter runs permanently**. In this mode, synchronized conversions of more than one ADC kernel are possible.
  The trigger for the conversion triggers has to be generated synchronously to the arbiter timing. Incoming triggers should have exactly n-times the granularity of the arbiter (n = 1, 2, 3, ...). In order to allow some flexibility, the duration of an arbitration slot can be programmed in cycles of f_{ADC}.
- If bit **GLOBCTR**.ARBM = 1, the **arbiter stops after an arbitration round** when no conversion request have been found pending any more. The arbiter is started again if at least one enabled request source indicates a pending conversion request. The trigger of a conversion request does need not to be synchronous to the arbiter timing. In this mode, parallel conversions are not possible for synchronization slave kernels.

23.2.7.1 Request Source Priority

Each request source has an individually programmable priority to be able to adapt to different applications (see registers **RSPR0**, **RSPR4**). The priorities define the order the request sources are handled by the arbiter if two or more request sources indicate

pending conversion requests at the same time.
Starting with request source 0, the arbiter checks if an enabled request source has a pending request for a conversion. The arbitration winner is the request source with a pending conversion request and the highest priority that has been found first in an arbitration round.

23.2.7.2 Conversion Start Modes

To start the requested conversion of the arbitration winner, the following aspects are automatically taken into consideration by the arbiter:

- If the converter is currently idle (no conversion running), the conversion of the arbitration winner is started immediately.
- If a conversion is currently running, the arbitration winner is compared to the priority of the currently running conversion. In the case that the current conversion has the same or a higher priority, it is completed. Then, the conversion of the arbitration winner is started.
- If a conversion is currently running, the arbitration winner is compared to the priority of the currently running conversion. In the case that the current conversion has the lower priority and the arbiter winner has been programmed for **wait-for-start mode**, the currently running conversion is completed. Then, the conversion of the arbitration winner is started.
  This mode can be used if the timing requirement for the higher priority conversions allow a jitter (between t3 and t4 in **Figure 23-8**) in the range of a running conversion.
- If a conversion is currently running, the arbitration winner is compared to the priority of the currently running conversion. In the case that the current conversion has the lower priority and the arbiter winner has been programmed for **cancel-inject-repeat mode**, the current conversion is aborted immediately if a new request with a higher priority has been found, unless both requests target the same result register with wait-for-read active (see **Section 23.2.15.2**). The conversion of the arbitration winner is started after the abort action. The aborted conversion request is restored in the request source that has requested the aborted conversion. As a consequence, it takes part again in the next arbitration round.
  Please note that the abort mechanism can take between 1 and 3 f_{ADCI} cycles, depending on the state of the current conversion.
  This mode can be used if higher priority conversions only tolerate a small jitter (between t8 and t9 in **Figure 23-8**).

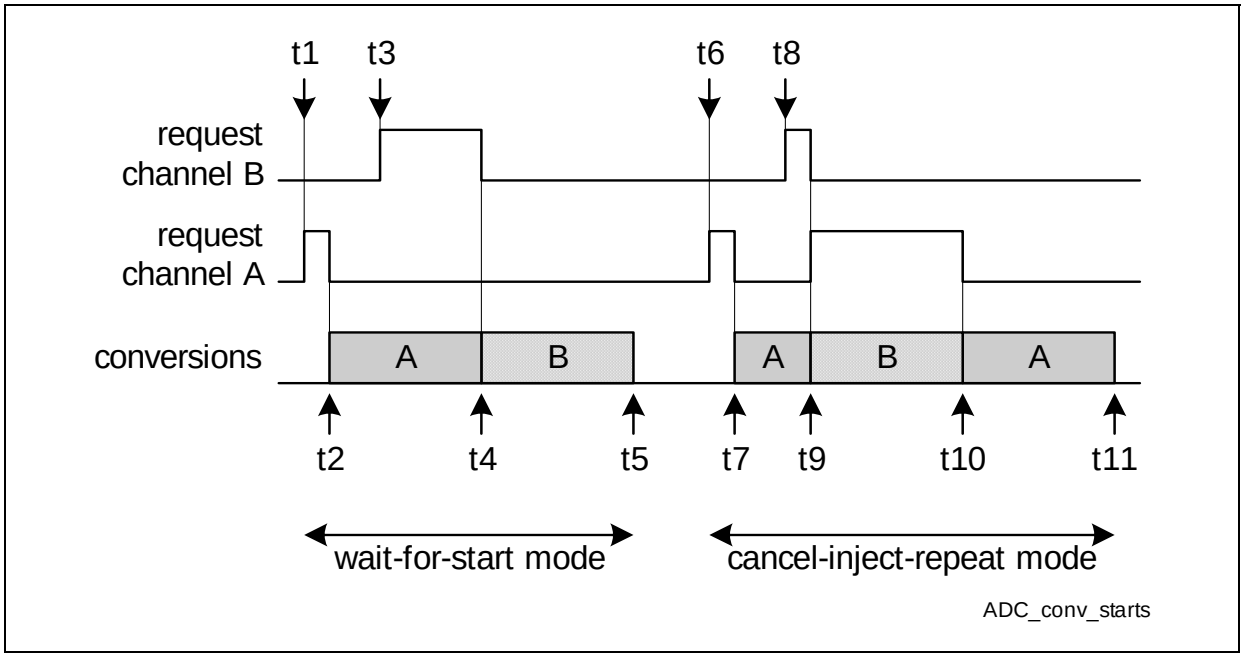


Figure 23-8 Conversion Start Modes

The conversion start mode can be individually programmed for each request source by bits in registers **RSPR0** and **RSPR4** and is applied to all channels requested by the source. **Figure 23-8** shows the influence of both conversion start modes on the conversion sequence if two request sources generate conversion requests. In this example, channel A is issued by a request source with a lower priority than the request source requesting the conversion of channel B.

- t1: The trigger event for channel A occurs and a conversion request is activated.
- t2: At the end of the arbitration round, channel A is determined as arbitration winner, the conversion of channel A is started. With the start of the conversion, the related conversion request is cleared.
- t3: The trigger event for channel B occurs and a conversion request is activated. In wait-for-read mode, the currently running conversion of channel A is finished normally.
- t4: After the conversion of channel A is finished, the conversion of channel B is started. With the start of the conversion, the related conversion request is cleared.
- t5: The conversion of channel B is finished.
- t6: The trigger event for channel A occurs and a conversion request is activated.
- t7: At the end of the arbitration round, channel A is determined as arbitration winner, the conversion of channel A is started. With the start of the conversion, the related conversion request is cleared.
- t8: The trigger event for channel B occurs and a conversion request is activated.
- t9: At the end of the arbitration round, channel B is determined as arbitration winner. In cancel-inject-repeat mode, the currently running conversion of channel A is aborted and the conversion of channel B is started. With the abort of the conversion,

the related conversion request is set again. With the start of the conversion, the related conversion request is cleared.

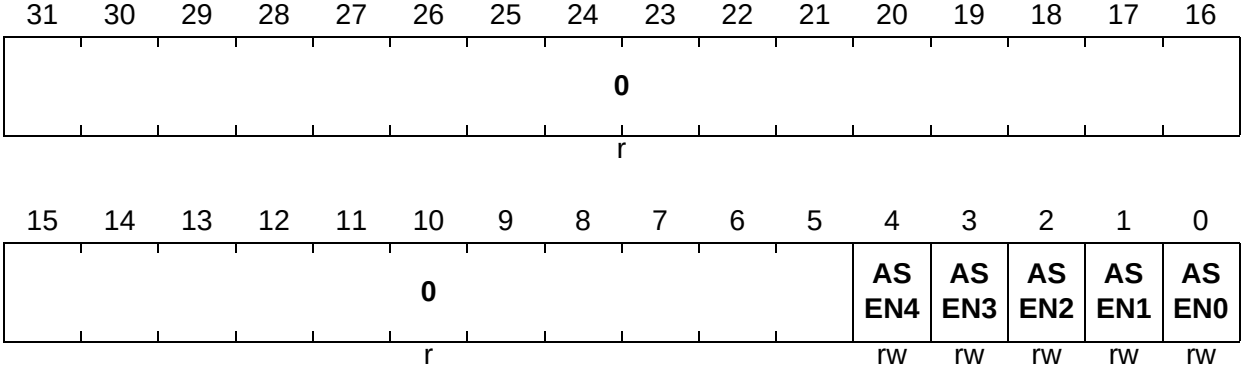
- t10: The conversion of channel B is finished. In the meantime, the pending request for channel A has been identified as arbitration winner and the conversion of channel A is started. With the start of the conversion, the related conversion request is cleared.
- t11: The conversion of channel A is finished.

23.2.8 Arbiter Registers

23.2.8.1 Arbitration Slot Enable Register

The arbitration slot enable register contains bits to enable/disable the conversion request treatment in the arbitration slots.

ASENR
Arbitration Slot Enable Register ($03C_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
ASENx (x = 0-4)	x	rw	**Arbitration Slot x Enable** Each bit enables an arbitration slot of the arbiter round. ASEN0 enables the arbitration slot 0, ASEN1 the slot 1, etc. If an arbitration slot is disabled, it is considered as being empty. The request bits of the request sources are not modified by write actions to ASENR. 0_B The corresponding arbitration slot is disabled and is not taken into account by the arbiter. Conversions are not requested, even for the request source(s) with pending request bit(s). 1_B The corresponding arbitration slot is enabled. Conversions are requested for the request source(s) with pending request bit(s).
0	[31:5]	r	**Reserved** Read as 0; should be written with 0.

23.2.8.2 Request Source Priority Register

The request source priority registers contain bits to define the request source priority and the conversion start mode.

RSPR0
Request Source Priority Register 0 (040_H) **Reset Value: 0000 0000_H**

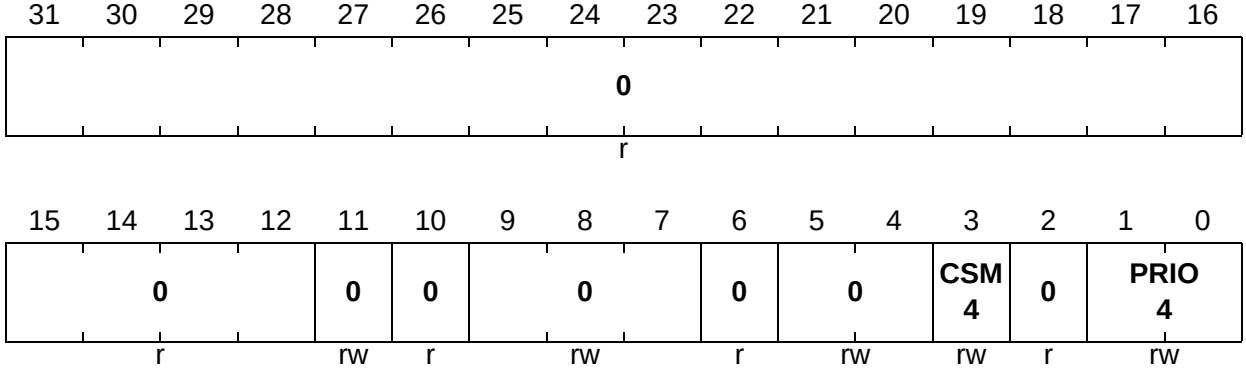
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSM 3	0	PRIO 3		CSM 2	0	PRIO 2		CSM 1	0	PRIO 1		CSM 0	0	PRIO 0	
rw	r	rw		rw	r	rw		rw	r	rw		rw	r	rw	

Field	Bits	Type	Description
PRIO0, PRIO1, PRIO2, PRIO3	[1:0], [5:4], [9:8], [13:12]	rw	**Priority of Request Source x** This bit field defines the priority of the conversion request source x, located in arbitration slot x. 00_B Lowest priority is selected. ... 11_B Highest priority is selected.
CSM0, CSM1, CSM2, CSM3	3, 7, 11, 15	rw	**Conversion Start Mode of Request Source x** This bit defines the conversion start mode of the conversion request source x, located in arbitration slot x. 0_B The wait-for-start mode is selected. 1_B The cancel-inject-repeat mode is selected.
0	2, 6, 10, 14, [31:16]	r	**Reserved** Read as 0; should be written with 0.

RSPR4

Request Source Priority Register 4 (044_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
PRIO4	[1:0]	rw	**Priority of Request Source 4** This bit field defines the priority of the conversion request source 4, located in arbitration slot 4. 00_B Lowest priority is selected. ... 11_B Highest priority is selected.
CSM4	3	rw	**Conversion Start Mode of Request Source 4** This bit defines the conversion start mode of the conversion request source 4, located in arbitration slot 4. 0_B The wait-for-start mode is selected. 1_B The cancel-inject-repeat mode is selected.
0	[5:4], [9:7], 11	rw	**Reserved** These bits are reserved for future use and have to be written with 000_B.
0	2, 6, 10, [31:12]	r	**Reserved** Read as 0; should be written with 0.

23.2.9 Scan Request Source Handling

A scan request source can issue conversion requests for a sequence of up to 16 input channels. It can be programmed individually for each input channel if it takes part in the scan sequence. The scan sequence always starts with the highest enabled channel number and continues towards lower channel numbers (order defined by the channel number, each channel can be converted only once per sequence).

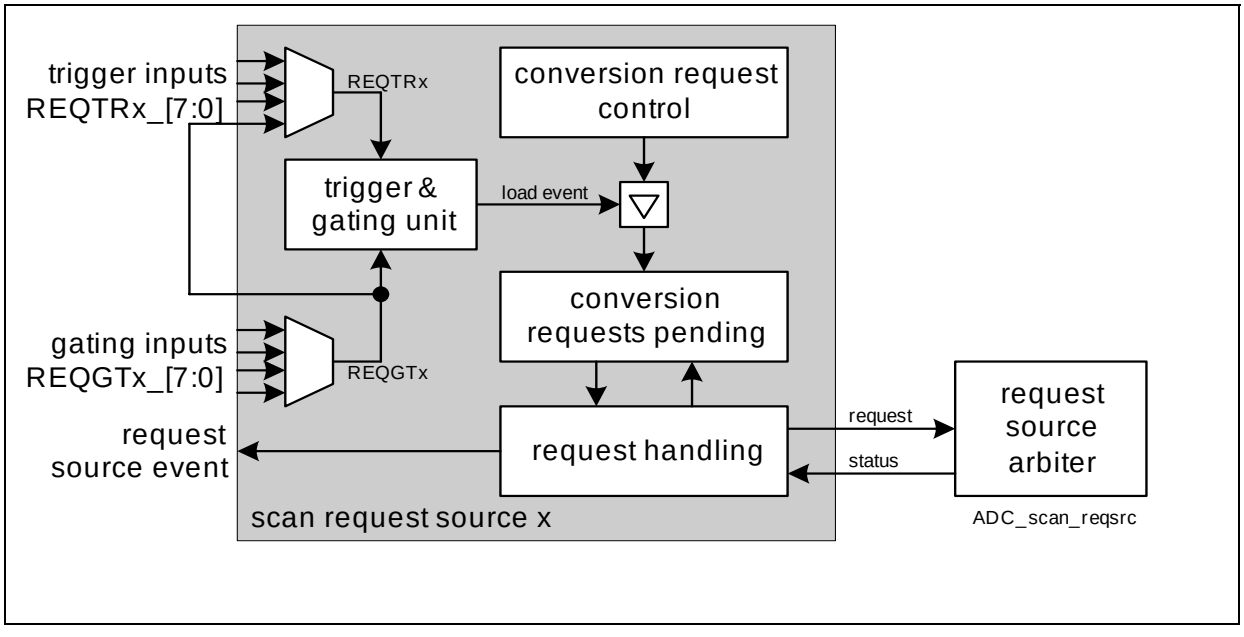


Figure 23-9 Scan Request Source

23.2.9.1 Overview

A scan request source performs the:

- Conversion request control:
  The conversion request control defines if an analog input channel takes part in the scan sequence (see bits in registers **CRCR1**, **CRCR3**). The programmed register value is kept unchanged by an ongoing scan sequence.
- Conversion request pending:
  The pending conversion requests indicate if an input channel has to be converted in an ongoing scan sequence (see bits in registers **CRPR1**, **CRPR3**). A conversion request can only be issued to the request source arbiter if at least one pending bit is set. With each conversion start that has been triggered by the scan request source, the corresponding pending bit is automatically cleared. The scan sequence is considered finished and a request source event is generated if the last conversion triggered by the scan source is finished and all pending bits have been cleared.
- Request handling:
  The request handling blocks interfaces with the request source arbiter. It requests conversion due to pending bits in the scan sequence and handles the conversion

status information. If a conversion triggered by the scan request source is aborted due to a conversion request from another request source with a higher priority, the corresponding pending bit is automatically set. This mechanism ensures that an aborted conversion takes part in the next arbitration round and does not get lost. The control of the scan sequence is done based on bits in registers **CRMR1**, **CRMR3**.

- Trigger and gating signal handling:
  The trigger and gating unit interfaces with signals and modules outside the ADC module that can request conversions. For example, a timer unit can issue a request signal to synchronize conversions to PWM events. A load event starts a scan sequence by modifying the request pending bits according to the request control bits.

23.2.9.2 Scan Sequence Operation

To **operate a scan request source**, the following aspects should be taken into account:

- The bits in register CRCRx have to be programmed to define the channels participating in the scan sequence.
- If a trigger or gating function by external signals is desired, the gating and trigger inputs have to be defined by bit fields in the related registers **RSIRx (x = 0 - 4)**, the value of x defines the number of the arbitration slot where the scan source is connected. Also the edge selection for the trigger event is done in these registers.
- The gating mechanism has to be defined by CRMRx.ENGT.
- The corresponding arbitration slot has to be enabled to accept conversion requests from the scan source (see register **ASENR**).
- The load event has to be defined by bits in CRMRx to start a scan sequence.
- If a load event occurs while CRMRx.LDM = 0, the content of CRCRx is copied to CRPRx (overwrite). This setting allows starting a new scan sequence and to “forget” remaining pending bits if a load event occurs while a scan sequence is running.
- If a load event occurs while CRMRx.LDM = 1, the content of CRCRx is bit-wisely logical OR-combined to CRPRx (no overwrite). This setting allows starting a new scan sequence without “forgetting” remaining pending bits if a load event occurs while a scan sequence is running.

To **start a scan sequence**, the following mechanisms are supported to generate a load event:

- An external trigger signal can be selected to start a scan sequence controlled by HW by an external module or signal, e.g. a timer unit or an input pin. The trigger feature is enabled by CRMRx.ENTR = 1. The load event is generated if the selected edge is detected at the selected trigger input REQTRx. The edge selection is done in register RSIRx.
- A load event is generated under SW control by writing CRMRx.LDEV = 1. This mechanism starts a scan sequence without modifying the bits in register CRCRx. A data write action to CRCRx does not lead to a load event (first prepare the channel control, then start the sequence).

- If SW writes data to register CRPRx, the written data is stored in register CRCRx and a load event is generated automatically. This mechanism starts a scan sequence with the channels defined by the written data (the sequence is defined and started with a single data write action, e.g. under DMA control).
- A load event is generated each time a scan sequence has finished and the request source event occurs if bit CRMRx.SCAN = 1. This setting leads to a permanent repetition of the scan sequence.

To **stop or abort an ongoing scan sequence**, the following mechanisms are supported:

- An external gating signal can be selected to stop and to continue a scan sequence at any point in time controlled by an external module or signal, e.g. a timer unit or an input pin. The gating feature can be enabled and the polarity of the gating signal REQGTx can be selected by CRMRx.ENGT. The gating mechanism does not modify the contents of the conversion pending bits, but only prevents the request handling block from issuing conversion requests to the arbiter.
- The arbiter can be disabled by SW for this arbiter slot by clearing the corresponding bit **ASENR**.ASENx. This mechanism does not modify the contents of the conversion pending bits, but only prevents the arbiter from accepting requests from the request handling block.
- The pending request bits can be cleared by writing bit CRMRx.CLRPND = 1. It is recommended to stop the scan sequence before clearing the pending bits.

23.2.9.3 Request Source Event and Interrupt

A request source event of a scan source occurs if the last conversion of a scan sequence is finished (all pending bits = 0). A request source event interrupt can be generated based on a request source event according to the structure shown in **Figure 23-10**. If a request source event is detected, it sets the corresponding indication flag in register **EVFR**. These flags can also be set by writing a 1 to the corresponding bit position, whereas writing 0 has no effect. Additionally, a gated event flag **EVFR**.GFSx indicates that a request source interrupt has been activated. The indication flags can be cleared by SW by writing a 1 to the corresponding bit position in register **EVFCR**.

The service request output ADCy_SRx that is selected by the request source event interrupt node pointer bit fields in register **EVNPR** becomes activated each time the related request source event is detected and the interrupt generation is enabled for this event in registers **CRCR1** (for request source 1) or **CRCR3** (for request source 3).
A service request output can be activated under SW control by writing **INTR**.SISRx.

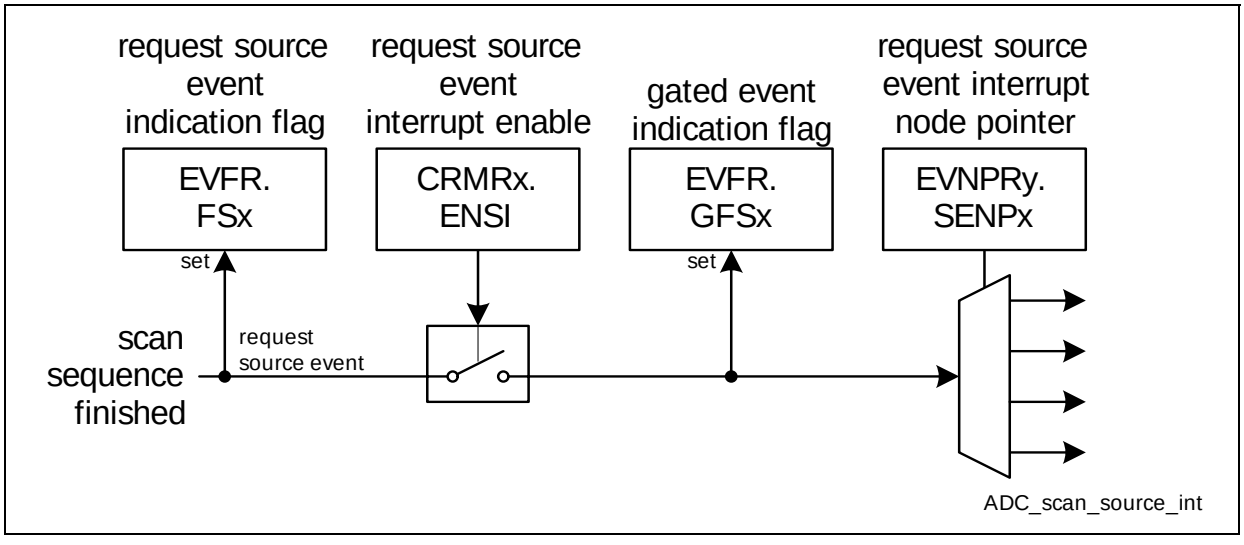


Figure 23-10 Interrupt Generation of a Scan Request Source

23.2.10 Scan Request Source Registers

23.2.10.1 Conversion Request Control Registers

These registers contain the control and status bits of a scan request source. In the TC1797, two scan sources are available (sources 1 and 3). The index describes the number of the arbitration slot where the request source is taking part in the arbitration.

The conversion request control register contains the bits that are copied to the pending register when the load event occurs. This register can be accessed at two different addresses. One address for read and write access is given for CRCRx (attribute “rw”), leading to a data write to the bits in CRCRx without an automatic load event. The second address only used for write actions is given for CRPRx (additional attribute “h”), leading to a data write to the bits in CRCRx with an automatic load event one clock cycle later.

CRCR1
Conversion Request 1 Control Register
(090_H) **Reset Value: 0000 0000_H**

CRCR3
Conversion Request 3 Control Register
($0B0_H$) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CH 15	CH 14	CH 13	CH 12	CH 11	CH 10	CH 9	CH 8	CH 7	CH 6	CH 5	CH 4	CH 3	CH 2	CH 1	CH 0
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

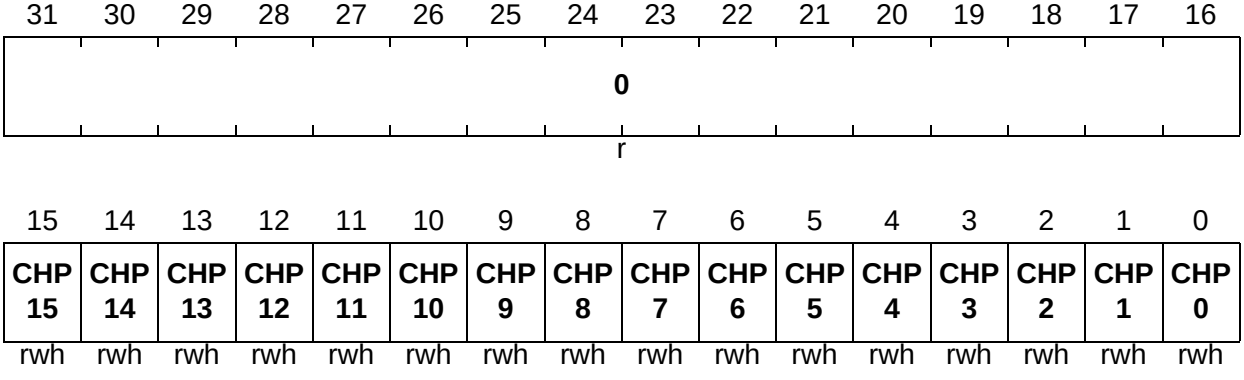
Field	Bits	Type	Description
CHx (x = 0-15)	x	rwh	**Channel Bit x** Each bit corresponds to one analog input channel, the channel number CHx is defined by the bit position x in this register. The corresponding bit x in the conversion request pending register will be overwritten by this bit (LDM = 0) or bit-wisely OR-combined with this bit (LDM = 1) when the load event occurs. 0_B The analog channel CHx will not be requested for conversion by this request source. 1_B The analog channel CHx will be requested for conversion by this request source.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

23.2.10.2 Conversion Request Pending Registers

The conversion request pending register contains the bits that are requesting a conversion of the corresponding analog channel.
A read operation to CRPRx delivers the pending bits (attribute “rh”). A write operation to CRPRx leads to a data write to the bits in CRCRx with an automatic load event generation (additional attribute “w”).

CRPR1
Conversion Request 1 Pending Register
(094_H) **Reset Value: 0000 0000_H**

CRPR3
Conversion Request 3 Pending Register
($0B4_H$) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
CHPx (x = 0-15)	x	rwh	**Channel Pending Bit x** Write view: A write to this address targets the bits in register **CRCR1** (for CRPR1) or **CRCR3** (for CRPR3). Read view: Each bit corresponds to one analog channel, the channel number CHx is defined by the bit position in the register. 0_B The analog channel CHx is not requested for conversion by this request source. 1_B The analog channel CHx is requested for conversion by this request source.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

23.2.10.3 Conversion Request Mode Registers

The conversion request mode registers contain bits to configure the desired operating mode of the scan request sources.

CRMR1
Conversion Request 1 Mode Register
(098_H) **Reset Value: 0000 0000_H**

CRMR3
Conversion Request 3 Mode Register
($0B8_H$) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						LDEV	CLRPND	REQGT	0	LDM	SCAN	ENSI	ENTR	ENGT	
r						w	w	rh	r	rw	rw	rw	rw	rw	

Field	Bits	Type	Description
ENGT	[1:0]	rw	**Enable Gate** This bit field enables the gating functionality for the request source. 00_B The request source does not issue conversion requests. 01_B The request source issues conversion requests if at least one pending bit is set. 10_B The request source issues conversion requests if at least one pending bit is set and the selected gating signal REQGTx = 1. 11_B The request source issues conversion requests if at least one pending bit is set and the selected gating signal REQGTx = 0.

Field	Bits	Type	Description
ENTR	2	rw	**Enable External Trigger** This bit enables the external trigger possibility. If enabled, the load event takes place if the selected edge is detected at the external trigger input REQTRx. 0_B The external trigger is disabled. 1_B The external trigger is enabled.
ENSI	3	rw	**Enable Source Interrupt** This bit enables the request source interrupt generation if a request source event occurs (last pending conversion is finished). 0_B The request source interrupt is disabled. 1_B The request source interrupt is enabled.
SCAN	4	rw	**Autoscan Enable** This bit enables a permanent scan functionality. If enabled, the load event is automatically generated if a request source event occurs. 0_B The permanent scan functionality is disabled. 1_B The permanent scan functionality is enabled.
LDM	5	rw	**Load Event Mode** This bit defines the transfer mechanism triggered by the load event. 0_B With the load event, the value of register CRCRx is copied to the pending register CRPRx (overwrite). 1_B With the load event, the value of register CRCRx is bit-wisely logical OR combined to the pending register CRPRx.
REQGT	7	rh	**Request Gate Level** This bit monitors the level at the REQGTx input. 0_B The level is 0. 1_B The level is 1.
CLRPND	8	w	**Clear Pending Bits** 0_B No action. 1_B The bits in register CRPRx are cleared.
LDEV	9	w	**Generate Load Event** 0_B No action. 1_B A load event is generated.

Field	Bits	Type	Description
0	6, [31:10]	r	**Reserved** Read as 0; should be written with 0.

23.2.11 Sequential Request Source Handling

Sequential request sources have been introduced to allow short conversion sequences with freely programmable channel numbers (contrary to a scan request source with a fixed conversion order for the enabled channels). Two versions of the sequential sources are available in each ADC kernel:

- Request sources in arbitration slots 2 and 4:
  These request sources can handle a sequence of up to 4 input channels (4-stage queue for 4 entries). This mechanism could be used to support application-specific conversion sequences, especially for timing-critical sequences containing multiple conversions of the same channel.
- Request source in arbitration slot 0:
  This request source can handle a single input channel (1-stage queue for 1 entry). This mechanism could be used for SW-controlled conversion requests or HW-triggered conversions of a single input channel (to “inject” a single conversion into a running sequence).

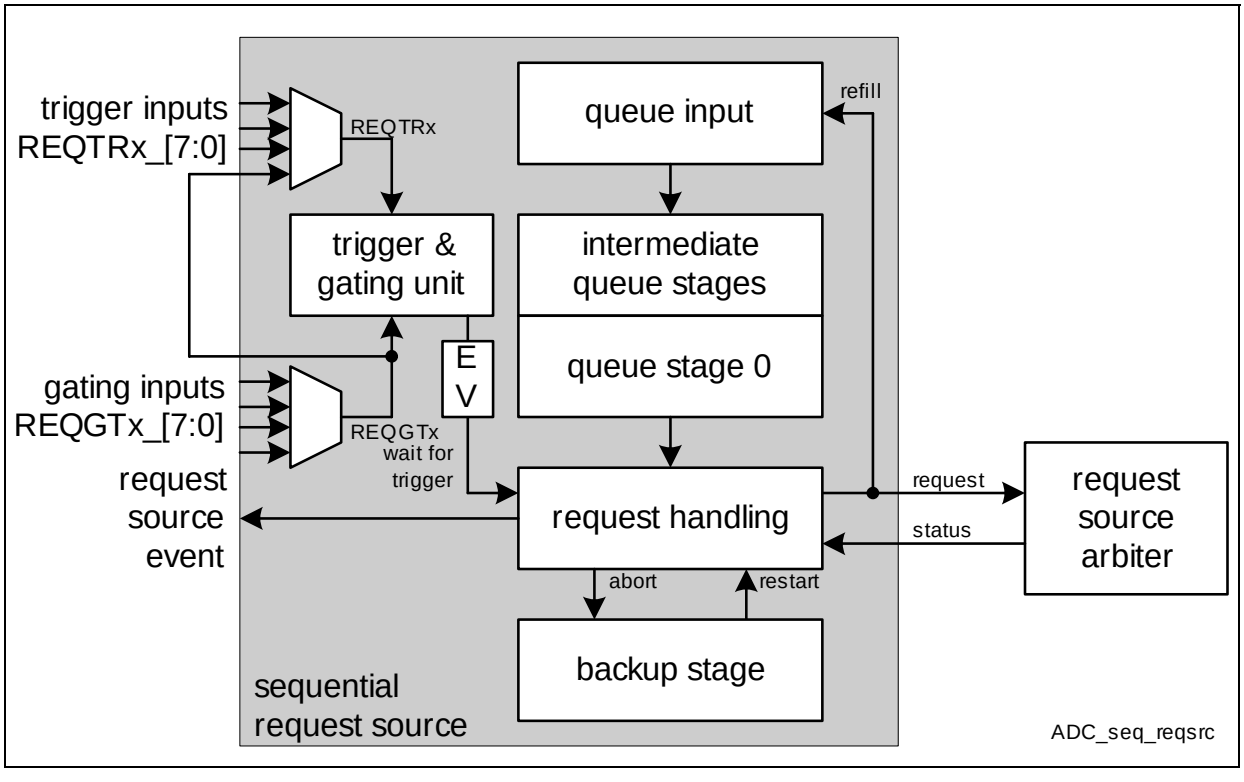


Figure 23-11 Sequential Request Source

The internal structure and the handling of the sequential sources is similar for both versions. The programmed sequence is stored in a queue buffer (based on a FIFO mechanism) with at least one queue stage (stage 0) and a backup stage for aborted conversions. The only difference between both versions is given by the number of intermediate queue stages for storing the sequence. The request source in arbitration

slot 0 does not provide intermediate queue stages (1-stage queue with only queue stage 0), whereas the ones in arbitration slots 2 and 4 provides 3 intermediate queue stages in additional to queue stage 0 (leading to a 4-stage queue each).

23.2.11.1 Overview

A sequential request source performs the:

- Queue input:
  The queue input represents the programming interface where the sequence is defined (see **QINR0**, **QINR2**, **QINR4**). It does not provide any buffer capability, but handles the filling of the queue buffer (queue stage 0 plus optional intermediate queue stages) by writing data to it. The contents of the queue stages can not be directly modified by program, except by the command for flushing the complete queue.
  The queue input also handles the refill mechanism, an automatic re-insertion of a started conversion from queue stage 0 (including the control parameters) as new queue input. This feature allows a single setup (by SW) of a conversion sequence and multiple repetitions of the same sequence without the need to re-program it each time. A conversion sequence is repeated if all queue entries of the sequence are setup for refill mode.
- Queue stage 0:
  The contents of this queue stage defines which channel will be requested next for a conversion (see **Q0R0**, **Q0R2**, **Q0R4**). It also defines if the request should be triggered by an external event or if the requested conversion should follow the previous one as soon as possible. It also enables the request source interrupt generation after the conversion.
  The contents of this queue stage is cleared when the requested conversion is started and the next queue entry can be handled (if available).
- Queue backup stage:
  The queue backup stage is used to store the request control parameters when a conversion requested by this request source is aborted. A validation bit indicates that the aborted conversion has to be requested next (before the current contents of queue stage 0) to maintain the original sequence (see **QBUR0**, **QBUR2**, **QBUR4**).
- Request handling:
  The request handling block interfaces with the request source arbiter. It requests a conversion due to a valid information in queue stage 0 and handles the conversion status information. The control of the queue sequence is done based on bits in registers **QMR0**, **QMR2**, and **QMR4** (for the arbitration slot x).
- Trigger and gating signal handling:
  The trigger and gating unit interfaces with signals and modules outside the ADC module that can request conversions. For example, a timer unit can issue a request signal to synchronize conversions to PWM events. A trigger event can start a conversion request for the entry in queue stage 0 (see **QMR0**, **QMR2**, **QMR4**). An

event flag QSRx.EV indicates that a trigger event has been detected (selected edge of selected trigger input signal REQTRx if enabled by QMRx.ENTR or write action with QMRx.TREV = 1). This bit is cleared with each conversion start requested by this source or by writing bits CEV = 1, FLUSH = 1, or CLRV = 1.

23.2.11.2 Sequential Source Operation

To **operate a sequential request source**, the following aspects should be taken into account:

- The sequence has to be initialized by writing to the queue input **QINR0** (for arbitration slot 0), **QINR2** (for arbitration slot 2), or **QINR4** (for arbitration slot 4) when using the refill mechanism. Each write access corresponds to one conversion request.
  The desired sequence should be completely initialized before enabling the request source, because with enabled refill feature, write accesses by SW to QINRx are not allowed.
- If a trigger or gating function by external signals is desired, the gating and trigger inputs have to be defined by bit fields in the related registers **RSIRx (x = 0 - 4)**, the value of x defines the number of the arbitration slot where the request source is connected. Also the edge selection for the trigger event is done in these registers.
- The gating mechanism has to be defined by QMRx.ENGT.
- The corresponding arbitration slot has to be enabled to accept conversion requests from the sequential source (see register **ASENR**).

To **start a sequence** of a sequential request source, the following mechanisms are supported:

- An external trigger signal can be selected to start a scan sequence controlled by HW by an external module or signal, e.g. a timer unit or an input pin. The trigger feature is enabled by QMRx.ENTR = 1. The trigger event is generated if the selected edge is detected at the selected trigger input.
- A trigger event is generated under SW control by writing QMRx.TREV = 1. This mechanism starts a request if queue stage 0 contains valid data (or the queue backup stage respectively).
- A write operation to a queue input leads to a (new) valid queue entry. If the queue is empty (no valid entry), the written data arrives in queue stage 0 and starts a conversion request (if enabled by QMRx.ENGT and without waiting for an external trigger). If the refill mechanism is used, the queue inputs must not be written while the queue is running. Write operations to a completely filled queue are ignored.

To **stop or abort an ongoing sequence** of a sequential request source, the following mechanisms are supported:

- An external gating signal can be selected to stop and to continue a sequence at any point in time controlled by an external module or signal, e.g. a timer unit or an input pin. The gating feature can be enabled and the polarity of the gating signal REQGTx can be selected by QMRx.ENGT. The gating mechanism does not modify the queue

entries, but only prevents the request handling block from issuing conversion requests to the arbiter.

- The arbiter can be disabled by SW for this arbiter slot by clearing the corresponding bit **ASENR**.ASENx. This mechanism does not modify the queue entries, but only prevents the arbiter from accepting requests from the request handling block.
- The next pending queue entry is cleared by writing bit QMRx.CLRV = 1. It is recommended to stop the sequence before clearing a queue entry (ENGT = 00_B). If the queue backup stage contains a valid entry, this one is cleared, otherwise a valid entry in queue register 0 is cleared.
- All queue entries are cleared by writing bit QMRx.FLUSH = 1. It is recommended to stop the sequence before clearing queue entries.

23.2.11.3 Request Source Event and Interrupt

A request source event occurs when a conversion that has been requested by this source is completely finished. The interrupt enable bits are located in the queue 0 register (if this has not been a repeated start after an abort) or in the queue backup register (if this has been a repeated start after an abort), e.g. see **Q0R0** for request source 0) or in the queue backup register (if this has been a repeated start after an abort, e.g. see **QBUR0** for request source 0).
A request source event interrupt can be generated based on a request source event according to the structure shown in **Figure 23-12**. If a request source event is detected, it sets the corresponding indication flag in register **EVFR**. These flags can also be set by writing a 1 to the corresponding bit position, whereas writing 0 has no effect. The indication flags can be cleared by SW by writing a 1 to the corresponding bit position in register **EVFCR**.

The service request output ADCy_SRx that is selected by the request source event interrupt node pointer bit fields in register **EVNPR** becomes activated each time the related request source event is detected.
A service request output can be activated under SW control by writing **INTR**.SISRx.

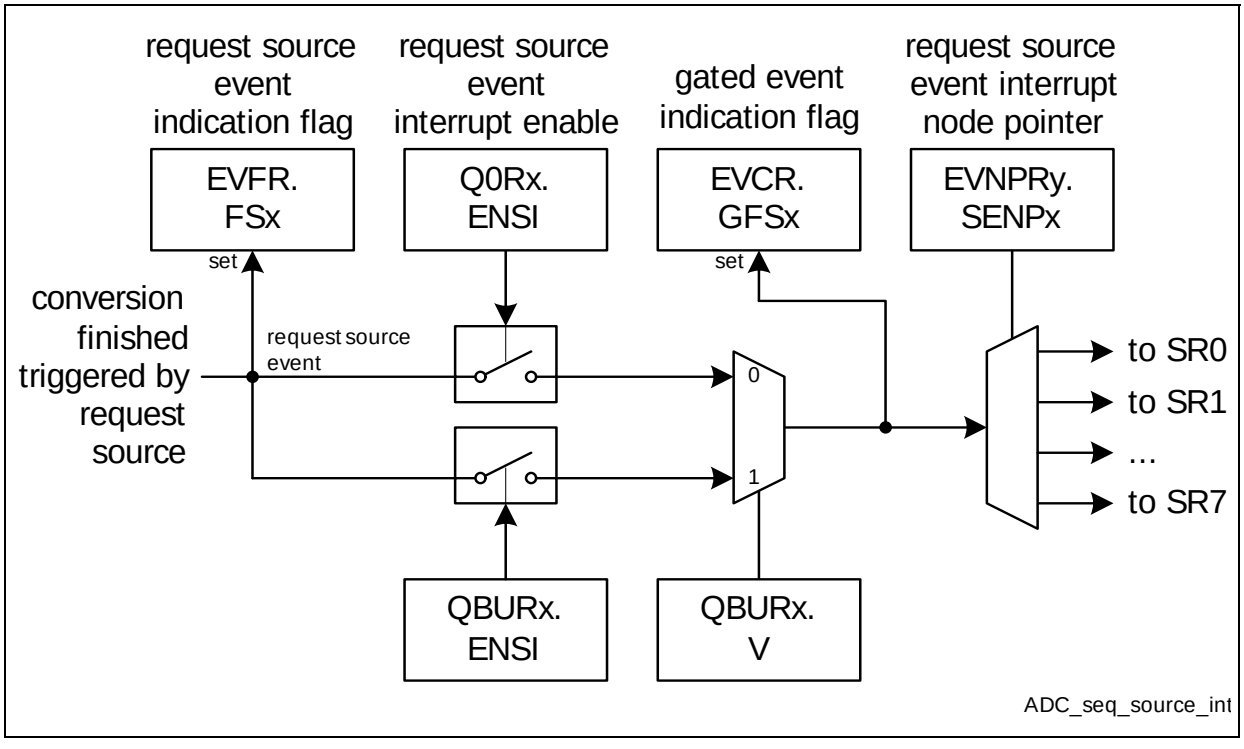


Figure 23-12 Interrupt Generation of a Sequential Request Source

23.2.12 Sequential Source Registers

23.2.12.1 Queue Mode Registers

These registers contain the control and status bits of a sequential source. The index describes the number of the arbitration slot where the request source is taking part in the arbitration.

*Note: Before SW modifies the queue content by QMRx.CLRV or QMRx.FLUSH, all HW actions related to this queue have to be finished. Therefore, the arbitration slot has to be disabled and SW has to wait for at least two arbitration rounds (to be sure that this request source can no longer be an arbitration winner). Then, it has to check **GLOBSTR**.CRSC and **GLOBSTR**.BUSY to be sure that a conversion triggered by this request source is no longer running. Then SW can read QBURx and Q0Rx and can start modification of the queue content.*

QMR0
Queue 0 Mode Register **(080_H)** **Reset Value: 0000 0000_H**
QMR2
Queue 2 Mode Register **($0A0_H$)** **Reset Value: 0000 0000_H**
QMR4
Queue 4 Mode Register **($0C0_H$)** **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				CEV	FLUSH	TREV	CLRV	0					ENTR	ENGT	
r				w	w	w	w	r					rw	rw	

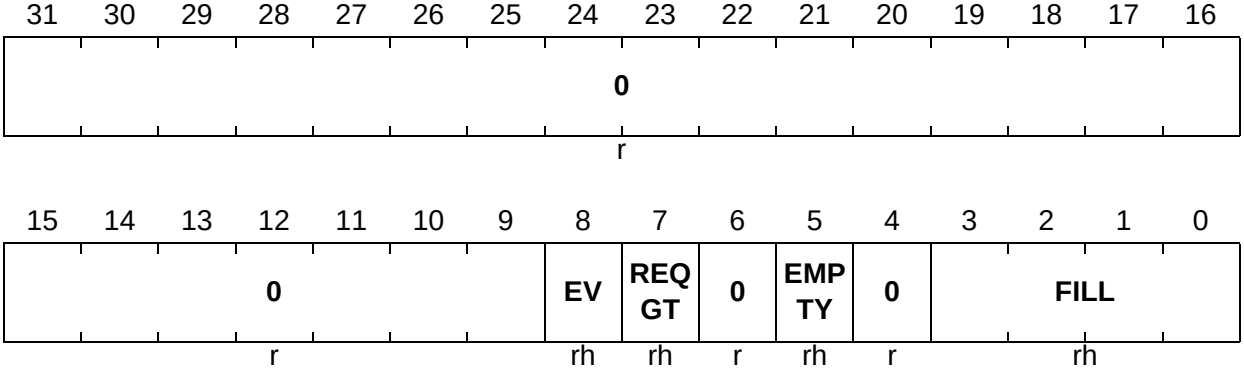
Field	Bits	Type	Description
ENGT	[1:0]	rw	**Enable Gate** This bit field enables the gating functionality for the request source. 00_B The request source does not issue conversion requests. 01_B The request source issues conversion requests if a valid conversion request is pending in the queue 0 register or in the backup register. 10_B The request source issues conversion requests if a valid conversion request is pending in the queue 0 register or in the backup register and the selected gating signal REQGTx = 1. 11_B The request source issues conversion requests if a valid conversion request is pending in the queue 0 register or in the backup register and the selected gating signal REQGTx = 0.
ENTR	2	rw	**Enable External Trigger** This bit enables the external trigger possibility. 0_B The external trigger is disabled and the trigger event is not generated. 1_B The external trigger is enabled and a trigger event is generated if the selected edge is detected at the selected trigger input signal for REQTRx.
CLRV	8	w	**Clear V Bit** 0_B No action. 1_B The next pending valid queue entry in the sequence and the event flag EV are cleared. If there is a valid entry in the queue backup register (QBURx.V = 1), this entry is cleared, otherwise the entry in queue register 0 is cleared.

Field	Bits	Type	Description
TREV	9	w	**Trigger Event** 0_B No action. 1_B A trigger event is generated by SW. If the a valid entry in the request source waits for a trigger event, a conversion request is started.
FLUSH	10	w	**Flush Queue** 0_B No action. 1_B All entries in the queue (including the backup stage) and the event flag EV are cleared. The queue contains no more valid entry.
CEV	11	w	**Clear Event Flag** 0_B No action. 1_B Bit EV is cleared.
0	[7:3], [31:12]	r	**Reserved** Read as 0; should be written with 0.

23.2.12.2 Queue Status Registers

The queue status registers contain bits indicating the status of the sequential source. The filling level and the empty information refer to the queue intermediate stages (if available) and to the queue register 0. An aborted conversion stored in the backup stage is not indicated by these bits (therefore, see QBURx.V).

QSR0
Queue 0 Status Register **(084_H)** **Reset Value: 0000 0020_H**
QSR2
Queue 2 Status Register **($0A4_H$)** **Reset Value: 0000 0020_H**
QSR4
Queue 4 Status Register **($0C4_H$)** **Reset Value: 0000 0020_H**



Field	Bits	Type	Description
FILL	[3:0]	rh	**Filling Level[1)]** This bit field indicates how many queue entries are valid in the sequential source. It is incremented each time a new entry is written to QINRx or by an enabled refill mechanism. It is decremented each time a requested conversion has been started. A new entry is ignored if the filling level has reached its maximum value. 00_B EMPTY = 1:There is no valid entry in the queue. EMPTY = 0:There is 1 valid entries in the queue. 01_B There are 2 valid entries in the queue. 10_B There are 3 valid entries in the queue. 11_B There are 4 valid entries in the queue.

Field	Bits	Type	Description
EMPTY	5	rh	**Queue Empty** This bit indicates if the sequential source contains valid entries. 0_B There are FILL+1 valid entries in the queue. 1_B There are no valid entries (queue is empty).
REQGT	7	rh	**Request Gate Level** This bit monitors the level at the REQGTx input. 0_B The level is 0. 1_B The level is 1.
EV	8	rh	**Event Detected** This bit indicates that an event has been detected while at least one valid entry has been in the queue (queue register 0 or backup stage). Once set, this bit is cleared automatically when the requested conversion is started. 0_B A trigger event has not been detected. 1_B A trigger event has been detected.
0	4, 6, [31:9]	r	**Reserved** Read as 0; should be written with 0.

1) This bit field is always 00_B for the 1-stage queue in arbitration slot 0.

23.2.12.3 Queue 0 Registers

The queue x registers 0 monitor the status of the current sequential request (queue stage 0).

Q0R0
Queue 0 Register 0 **(088_H)** **Reset Value: 0000 0000_H**
Q0R2
Queue 2 Register 0 **($0A8_H$)** **Reset Value: 0000 0000_H**
Q0R4
Queue 4 Register 0 **($0C8_H$)** **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							V	EXTR	ENSI	RF	0	REQCHNR			
r							rh	rh	rh	rh	r	rh			

Field	Bits	Type	Description
REQCHNR	[3:0]	rh	**Request Channel Number** This bit field indicates the channel number that will be or is currently requested.
RF	5	rh	**Refill** This bit indicates if the pending request is discarded after the conversion start or if it is automatically refilled into the queue input of the request queue. 0_B The request is discarded after the conversion start. 1_B The request is refilled into the queue after the conversion start.
ENSI	6	rh	**Enable Source Interrupt** This bit indicates if a request source interrupt is generated when the conversion is finished. 0_B The request source interrupt generation is disabled. 1_B The request source interrupt generation is enabled.

Field	Bits	Type	Description
EXTR	7	rh	**External Trigger** This bit indicates if a valid queue entry immediately leads to a conversion request or if the request handler waits for a trigger event. 0_B The request handler does not wait for a trigger event. 1_B The request handler waits for a trigger event.
V	8	rh	**Request Channel Number Valid** This bit indicates if the queue register 0 contains a valid queue entry. 0_B The queue entry is not valid and does not lead to a conversion request. 1_B The queue entry is valid and leads to a conversion request.
0	4, [31:9]	r	**Reserved** Read as 0; should be written with 0.

23.2.12.4 Queue Backup Registers

The queue backup registers monitor the status of an aborted sequential request.

The registers QBURx and QINRx share the same register address. A read operation at this register address will deliver the “rh” bits of register QBURx. A write operation to this address will target register QINRx.

QBUR0
Queue 0 Backup Register **($08C_H$)** **Reset Value: 0000 0000$_H$**
QBUR2
Queue 2 Backup Register **($0AC_H$)** **Reset Value: 0000 0000$_H$**
QBUR4
Queue 4 Backup Register **($0CC_H$)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							V	EXTR	ENSI	RF	0	REQCHNR			
r							rh	rh	rh	rh	r	rh			

Field	Bits	Type	Description
REQCHNR	[3:0]	rh	**Request Channel Number** This bit field contains the channel number of an aborted conversion that has been requested by this request source.
RF	5	rh	**Refill** This bit contains the refill bit of an aborted conversion that has been requested by this request source.
ENSI	6	rh	**Enable Source Interrupt** This bit contains the request source event interrupt enable bit of an aborted conversion that has been requested by this request source.

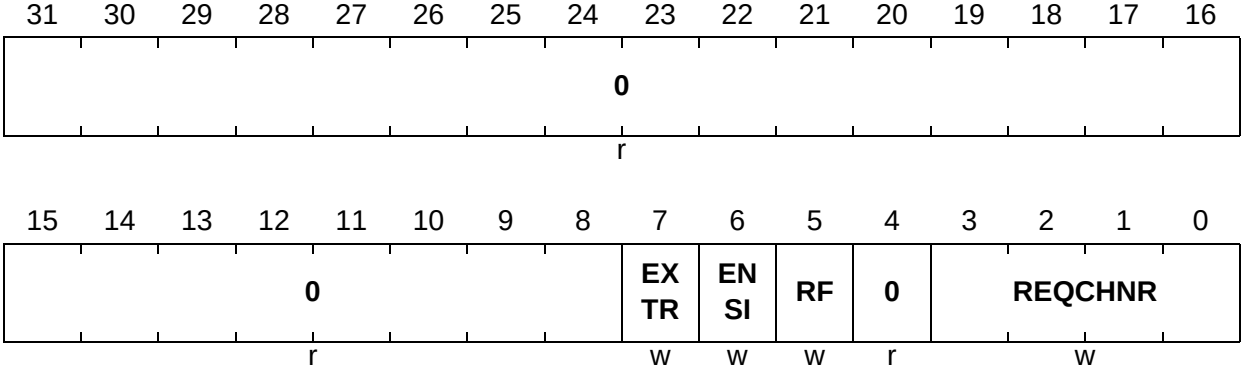
Field	Bits	Type	Description
EXTR	7	rh	**External Trigger** This bit contains the external trigger bit of an aborted conversion that has been requested by this request source.
V	8	rh	**Request Channel Number Valid** This bit indicates if the entry in the queue backup register is valid (REQCHNR, RF, TR and ENSI are valid). Bit V is set if a running conversion that has been requested by this request source is aborted. It is cleared when the repeated conversion is started. 0_B The backup register does not contain a valid entry. 1_B The backup register contains a valid entry. It will be requested before a valid entry in queue register 0 will be requested.
0	4, [31:9]	r	**Reserved** Read as 0; should be written with 0.

23.2.12.5 Queue Input Registers

The queue input registers are the entry registers for sequential requests for each sequential source (queue).

The registers QBURx and QINRx share the same register address. A read operation at this register address will deliver the “rh” bits of register QBURx. A write operation to this address will target the “w” bits in register QINRx.

QINR0
Queue 0 Input Register **($08C_H$)** **Reset Value: 0000 0000_H**
QINR2
Queue 2 Input Register **($0AC_H$)** **Reset Value: 0000 0000_H**
QINR4
Queue 4 Input Register **($0CC_H$)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
REQCHNR	[3:0]	w	**Request Channel Number** This bit field defines the requested channel number.
RF	5	w	**Refill** This bit defines the refill functionality for this queue entry. 0_B The content of this queue entry is not entered again in QINRx when the related conversion is started. 1_B The content of this queue entry is automatically entered again in QINRx when the related conversion is started.

Field	Bits	Type	Description
ENSI	6	w	**Enable Source Interrupt** This bit defines the request source event interrupt functionality. 0_B A request source event interrupt is not generated if the related conversion is finished. 1_B A request source event interrupt is generated if the related conversion is finished.
EXTR	7	w	**External Trigger** This bit defines the external trigger functionality. 0_B A valid queue entry immediately leads to a conversion request. 1_B A valid queue entry waits for a trigger event to occur before issuing a conversion request.
0	4, [31:8]	r	**Reserved** Read as 0; should be written with 0.

23.2.13 Channel-Related Functions

The channel control unit defines the conversion settings, that can be programmed individually for each analog input channel. Therefore, a channel control register **CHCTRx (x = 0 - 15)** is associated to each analog input channel CHx. After the arbiter has determined the channel to be converted, the defined settings are applied to the AD converter, comprising information about:

- **Conversion parameters:**
  Bit field ICLSEL defines which input class is taken into account for the conversion (see **Section 23.2.13.1**).
- **Reference selection:**
  Bit field REFSEL defines which reference input is used for the conversion (see **Section 23.2.13.2**)
- **Channel event handling:**
  Bit fields LCC, BNDASEL, and BNDBSEL define which boundaries are used for limit checking (see **Section 23.2.13.4**) and which channel event leads to a channel event interrupt (see **Section 23.2.13.5**).
- **Synchronous conversion request:**
  Bit SYNC defines if the channel triggers a synchronized conversion (see **Section 23.2.19**).
- **Alias feature:**
  In addition to the general channel control, the ADC kernel supports a mechanism (named alias feature, see **Section 23.2.13.3**) to redirect a conversion request to another channel number.

23.2.13.1 Input Classes

An input class defines the length of the sample phase and the resolution of the conversion. In most applications, the characteristics of the input circuitries (RC input low-pass filter and impedance of the signal source) are quite similar for several analog input signals, leading to similar timings for the sample phase of these channels. As a consequence, input channels with similar parameters can be grouped together to form an input class.
All channels with the same ICLSEL setting belong to the same input class and have the same sample phase length and resolution. In the TC1797, 4 input classes are supported. Registers **INPCRx (x = 0 - 3)** can be programmed to adjust the sample time and the resolution to the application requirements independently for each input class.
The default setting of these registers lead to the minimum sample phase length of 2 f_{ADCI} cycles and conversions with 10 bits resolution. If this default setting fits to the application requirements, bit fields **CHCTRx (x = 0 - 15)**.ICLSEL and registers **INPCRx (x = 0 - 3)** need not to be changed.

23.2.13.2 Reference Selection

The conversion result of the ADC is always referring to a reference voltage. The maximum digital result value (full scale) is obtained if the analog input voltage equals the reference voltage. In order to support more than one measurement range with full scale digital representation, the user can select between the standard reference input V_{AREF} and an alternative reference input at the analog input channel CH0 for each ADC kernel. The reference selection can be individually programmed for each input channel.
This feature can be used to connect 5 V based sensors and 3.3 V based sensors to the same ADC kernel. In this case, one set of input channels refers to the standard reference input, whereas the other one refers to the voltage level at input CH0.
Please note that the smallest granularity 1 LSB_n for n bit resolution refers to the selected reference voltage. The granularity becomes very small if a low reference voltage is applied, and as a consequence, the resulting TUE increases due to noise effects. Therefore it is recommended to avoid small reference voltages.

23.2.13.3 Alias Feature

The ADC kernel provides an alias feature, allowing a re-direction of conversion requests for channels CH0 or CH1 to other channel numbers. This feature can be used to measure the same input channel and to store the conversion results in two different result registers.

- The same signal can be measured twice without the need to read out the conversion result to avoid data loss. This allows triggering both conversions quickly one after the other and being independent from CPU interrupt latency.
- The sensor signal is connected to only one input channel (instead of two analog inputs). This saves input pins in low-cost applications and only the leakage of one input has to be considered in the error calculation.
- Even if the analog input CH0 is used as alternative reference (see **Figure 23-13**), the internal trigger and data handling features for channel CH0 can be used.
- The channel settings for both conversions can be different (boundary values, interrupts, etc.).
- If a sequential conversion request source has been set up, a conversion request for channels CH0 or CH1 can be easily directed to other input channels without flushing the queue.

In typical low-cost AC-drive applications, only one common current sensor is used to determine the phase currents. Depending on the applied PWM pattern, the measured value has different meanings and the sample points have to be precisely located in the PWM period. **Figure 23-13** shows an example where the sensor signal is connected to one input channel (CHx) but two conversions are triggered for two different channels (CHx and CH0). With the alias feature, a conversion request for CH0 leads to a conversion of the analog input CHx instead of CH0, but taking into account the settings for CH0. Although the same analog input (CHx) has been measured, the conversion

results can be stored and read out from the result registers RESRx (conversion triggered for CHx) and RESRy (conversion triggered for CH0). Additionally, different interrupts or limit boundaries can be selected, enabled or disabled.

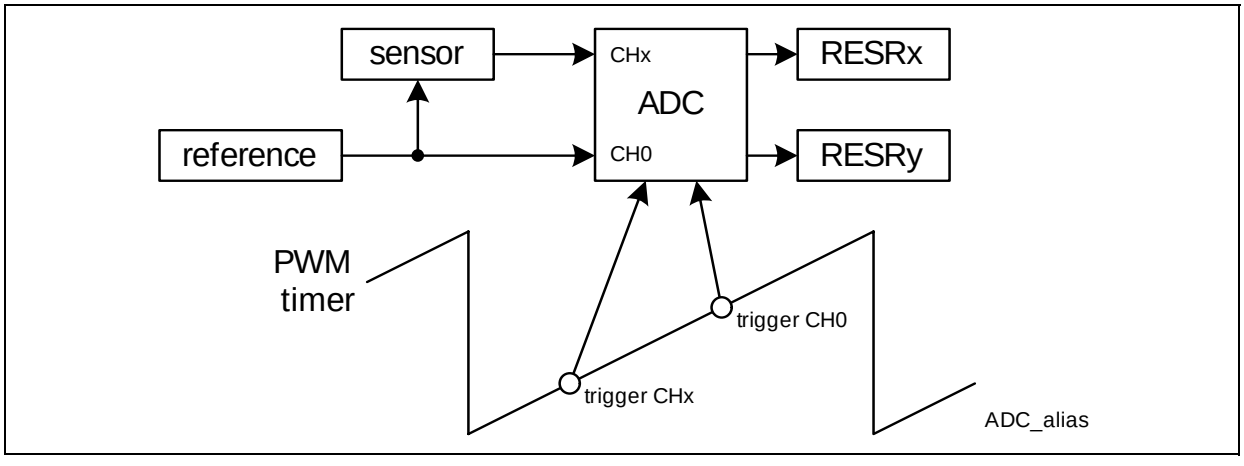


Figure 23-13 Alias Feature

23.2.13.4 Limit Checking

The limit checking mechanism automatically compares each conversion result to two boundary values (boundaries A and B). For each channel, the user can select these boundaries from a set of 4 programmable values (**LCBR0** to **LCBR3**).
With this structure, the conversion result range is split into three areas:

- Area I: The conversion result is below or equal to both boundaries.
- Area II: The conversion result is above one boundary and below or equal to the other boundary.
- Area III: The conversion result is above both boundaries.

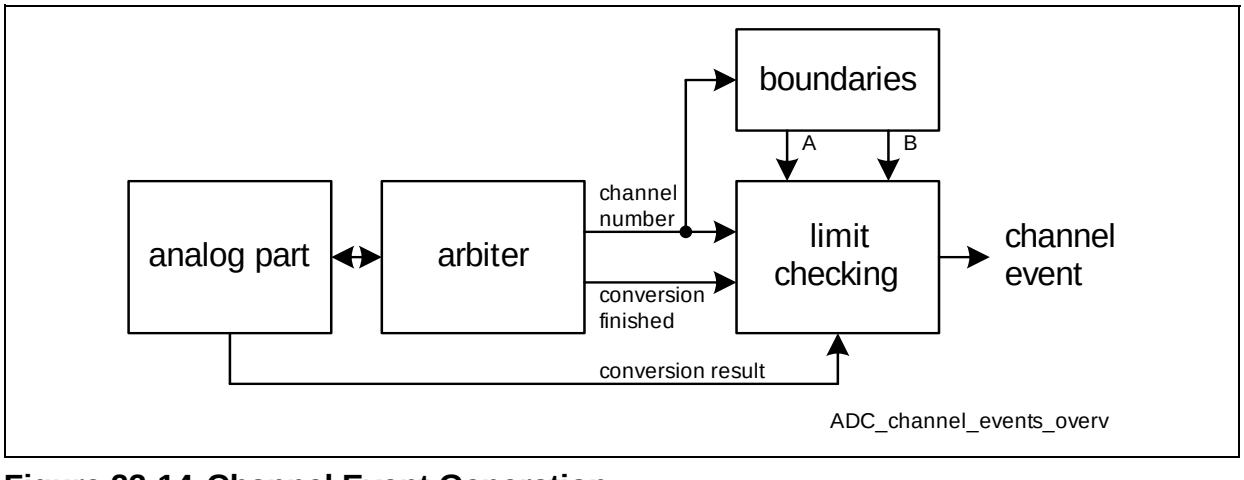


Figure 23-14 Channel Event Generation

Bit field LCC in the channel control register defines the condition to generate a channel event, leading to a channel event interrupt:

- LCC = 000_B: No trigger, the channel event generation is disabled.
- LCC = 001_B: A channel event is generated if the conversion result is not in area I.
- LCC = 010_B: A channel event is generated if the conversion result is not in area II.
- LCC = 011_B: A channel event is generated if the conversion result is not in area III.
- LCC = 100_B: A channel event is always generated (regardless of the boundaries).
- LCC = 101_B: A channel event is generated if the conversion result is in area I.
- LCC = 110_B: A channel event is generated if the conversion result is in area II.
- LCC = 111_B: A channel event is generated if the conversion result is in area III.

Figure 23-15 shows an example for limit checking where channel events are generated only if the conversion results are not in the normal operating range defined by area II (LCC = 010_B).
Typical applications for limit checking are temperature monitoring or overcurrent sensing. As long as the measured temperature value is below a boundary value, the CPU does not need to be informed. In this case, a channel event should be generated only if the conversion result is in area III (LCC = 111_B) to indicate an over-temperature condition. If the conversion of the analog temperature input signal is part of an auto-scan sequence autonomously triggered on a regular time base, the CPU load for the temperature monitoring is zero until the over-temperature condition is detected.

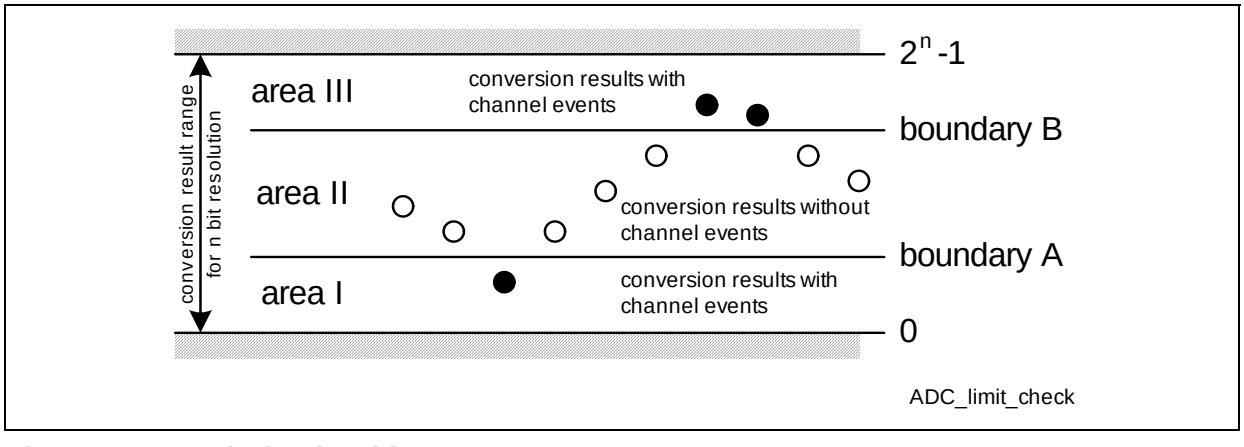


Figure 23-15 Limit Checking

Note: It is also possible to select the same boundary register for boundaries A and B. In this case, the conversion result range is split into two ranges (area II is empty).

23.2.13.5 Channel Event Interrupts

A channel event interrupt can be generated based on a channel event according to the structure shown in **Figure 23-16**. If a channel event is detected, it sets the corresponding indication flag FCx in register **CHFR**. These flags can also be set by writing a 1 to the corresponding bit position, whereas writing 0 has no effect. The indication flags can be cleared by SW by writing a 1 to the corresponding bit position in register **CHFCR**.

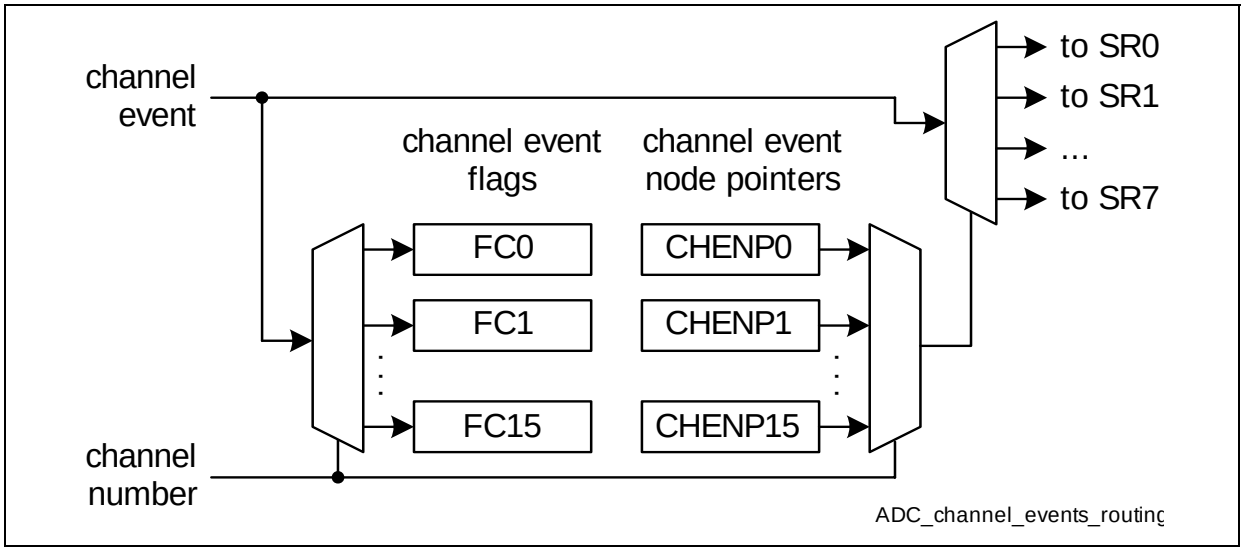


Figure 23-16 Channel Event Interrupt Generation

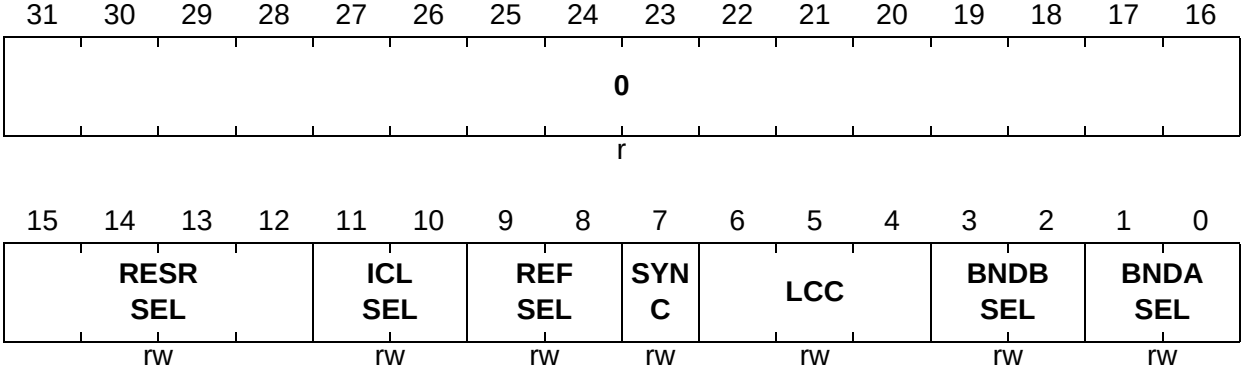
The service request output ADCy_SRx that is selected by the channel node pointer bit fields in registers **CHENPR0**, or **CHENPR8** is activated each time the related channel event is detected.
A service request output can be activated under SW control by writing **INTR**.SISRx.

23.2.14 Channel-Related Registers

23.2.14.1 Channel Control Registers

The channel control registers contain bits to select the targeted result register, to control the limit check mechanism and to select an input class.
The channel control register 0 defines the settings for the input channel 0, etc.

CHCTRx (x = 0 - 15)
Channel x Control Register **(100_H + x * 4)** **Reset Value: 0000 0000_H**



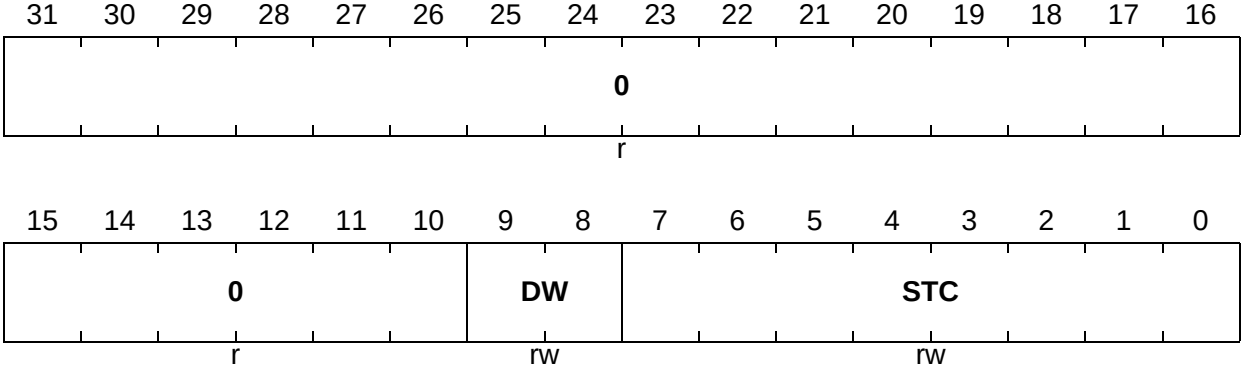
Field	Bits	Type	Description
BNDASEL	[1:0]	rw	**Boundary A Selection** This bit field defines which boundary will be taken as boundary A for the limit checking. 00_B The value given by LCBR0 is selected. 01_B The value given by LCBR1 is selected. 10_B The value given by LCBR2 is selected. 11_B The value given by LCBR3 is selected.
BNDBSEL	[3:2]	rw	**Boundary B Selection** This bit field defines which boundary will be taken as boundary B for the limit checking. 00_B The value given by LCBR0 is selected. 01_B The value given by LCBR1 is selected. 10_B The value given by LCBR2 is selected. 11_B The value given by LCBR3 is selected.
LCC	[6:4]	rw	**Limit Check Control** This bit field defines the behavior of the limit checking mechanism. Please refer to the coding in **Section 23.2.13.4** on **Page 23-78**.

Field	Bits	Type	Description
SYNC	7	rw	**Synchronization Request** This bit defines if a conversion request for this channel leads to a synchronized (parallel) conversion with other ADC kernels. This bit is only taken into account if the ADC kernel is a potential conversion master (**SYNCTR**.STSEL = 00), otherwise it is considered to be 0. 0_B This channel does not request a synchronized conversion. 1_B This channel requests a synchronized conversion if the ADC kernel is a potential synchronization master.
REFSEL	[9:8]	rw	**Reference Input Selection** This bit field defines the reference source for this channel. 00_B The standard reference input V_{AREF} is selected. 01_B The alternative reference input CH0 is selected. 10_B reserved, do not use 11_B reserved, do not use
ICLSEL	[11:10]	rw	**Input Class Selection** These bits are used to select the input class. 00_B The input class 0 is selected. 01_B The input class 1 is selected. 10_B The input class 2 is selected. 11_B The input class 3 is selected.
RESRSEL	[15:12]	rw	**Result Register Selection** This bit field defines which result register will be the target of a conversion of this channel. 0_H Result register 0 is selected. 1_H Result register 1 is selected. ... F_H Result register 15 is selected.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

23.2.14.2 Input Class Registers

The input class registers contain bits to control the sample time and the resolution for each input class.
The input class register 0 defines the settings for the input class 0, etc.

INPCRx (x = 0 - 3)
Input Class Register x **(050_H + x * 4)** **Reset Value: 0000 0000_H**

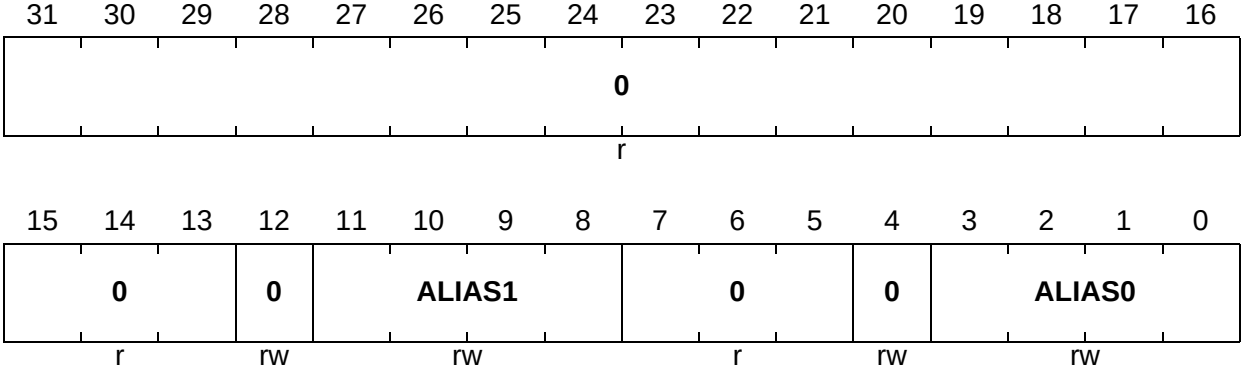


Field	Bits	Type	Description
STC	[7:0]	rw	**Sample Time Control** This bit field defines the additional length of the sample phase, given in analog clock cycles f_{ADCI}. A minimum sample phase of 2 analog clock cycles is extended by the programmed value. sample phase length = (2 + STC) / f_{ADCI}
DW	[9:8]	rw	**Data Width** This bit field defines how many bits are converted for the result. The MSBs of conversion results with different DW settings are left aligned in the result bit fields. Bit positions that are not converted are 0. 00_B The result is 10 bits wide. 01_B The result is 12 bits wide. 10_B The result is 8 bits wide. 11_B reserved
0	[31:10]	r	**Reserved** Read as 0; should be written with 0.

23.2.14.3 Alias Register

The alias register contains bits to change a requested channel number from CH0 and CH1 to another channel number, see also **Section 23.2.13.3**. The programmed alias channel number is replacing the internally requested number for analog input multiplexer (of the converter). The internally requested channel number is taken into account for all other internal actions and the synchronization request.

ALR0
Alias Register 0 **(210_H)** **Reset Value: $0000\ 0100_H$**

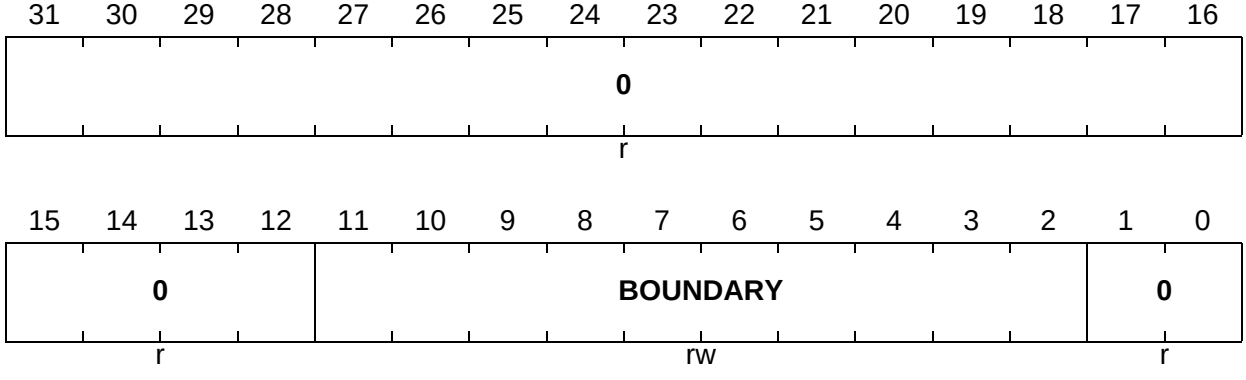


Field	Bits	Type	Description
ALIAS0	[3:0]	rw	**Alias Value for CH0 Conversion Requests** The channel indicated in this bit field is converted instead of channel CH0. The conversion is done with the settings defined for channel CH0.
ALIAS1	[11:8]	rw	**Alias Value for CH1 Conversion Requests** The channel indicated in this bit field is converted instead of channel CH1. The conversion is done with the settings defined for channel CH1.
0	4, 12	rw	**Reserved for Future Use** Bit is reserved for future use and has to be written with 0_B.
0	[7:5], [31:13]	r	**Reserved** Read as 0; should be written with 0.

23.2.14.4 Limit Check Boundary Registers

The bit fields in these registers define compare value (boundary) for the limit checking unit. The reset values of the boundaries are defined as 10%, 90%, 33% and 66% of the complete result range (the MSB located at bit position 11) of each conversion.

LCBR0
Limit Check Boundary Register 0 ($0F0_H$) Reset Value: 0000 0198_H
LCBR1
Limit Check Boundary Register 1 ($0F4_H$) Reset Value: 0000 $0E64_H$
LCBR2
Limit Check Boundary Register 2 ($0F8_H$) Reset Value: 0000 0554_H
LCBR3
Limit Check Boundary Register 3 ($0FC_H$) Reset Value: 0000 $0AA8_H$



Field	Bits	Type	Description
BOUNDARY	[11:2]	rw	**Boundary for Limit Checking** This bit field contains the value for the limit checking unit that is compared to the actual conversion result. The result of the limit check is used for the generation of the channel event, see **Section 23.2.13.4**.
0	[1:0], [31:12]	r	**Reserved** Read as 0; should be written with 0.

23.2.14.5 Channel Flag Register

The channel event indication flag register CHFR monitors the detected channel events.

CHFR
Channel Flag Register **(060_H)** **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FC15	FC14	FC13	FC12	FC11	FC10	FC9	FC8	FC7	FC6	FC5	FC4	FC3	FC2	FC1	FC0
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
FCx **(x = 0 - 15)**	x	rwh	**Event Flag for Channel x** Flag FCx indicates that a channel event for channel x has been detected. Writing a 0 has no effect, whereas writing a 1 sets the written bit position without generating an interrupt. Bit FCx is cleared by writing CHFCR.CFCx = 1. 0_B A channel x event has not occurred. 1_B A channel x event has occurred.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

23.2.14.6 Channel Flag Clear Register

Writing a 1 to a bit position in register CHFCR clears the corresponding channel flag in register CHFR. If a hardware event triggers the setting of bit CHFR.x and software writes CHFCR.x = 1, bit CHFR.x is cleared (software overrules hardware).

CHFCR
Channel Flag Clear Register (064$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CFC15	CFC14	CFC13	CFC12	CFC11	CFC10	CFC9	CFC8	CFC7	CFC6	CFC5	CFC4	CFC3	CFC2	CFC1	CFC0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
CFCx (x = 0 - 15)	x	w	**Clear Event Flag for Channel x** 0_B No action. 1_B Bit CHFR.FCx is cleared.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

23.2.14.7 Channel Event Node Pointer Registers

The bit fields in these registers define the service request output ADCy_SR[7:0] that is activated if a channel event occurs and the interrupt generation is enabled for this channel.

CHENPR0
Channel Event Node Pointer Register 0

(068_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**CHENP7**			**0**	**CHENP6**			**0**	**CHENP5**			**0**	**CHENP4**		
r	rw			r	rw			r	rw			r	rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**CHENP3**			**0**	**CHENP2**			**0**	**CHENP1**			**0**	**CHENP0**		
r	rw			r	rw			r	rw			r	rw		

Field	Bits	Type	Description
CHENP0, CHENP1, CHENP2, CHENP3, CHENP4, CHENP5, CHENP6, CHENP7	[2:0], [6:4], [10:8], [14:12], [18:16], [22:20], [26:24], [30:28]	rw	**Node Pointer for Channel x** This bit field defines which service request output becomes activated if the channel x event of kernel ADCy occurs while enabled by **CHCTRx (x = 0 - 15)**.LCC. 000_B ADCy_SR0 is selected. 001_B ADCy_SR1 is selected. ... 111_B ADCy_SR7 is selected.
0	3, 7, 11, 15, 19, 23, 27, 31	r	**Reserved** Read as 0; should be written with 0.

CHENPR8
Channel Event Node Pointer Register 8

(06C$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**CHENP15**			**0**	**CHENP14**			**0**	**CHENP13**			**0**	**CHENP12**		
r	rw			r	rw			r	rw			r	rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**CHENP11**			**0**	**CHENP10**			**0**	**CHENP9**			**0**	**CHENP8**		
r	rw			r	rw			r	rw			r	rw		

Field	Bits	Type	Description
CHENP8, CHENP9, CHENP10, CHENP11, CHENP12, CHENP13, CHENP14, CHENP15	[2:0], [6:4], [10:8], [14:12], [18:16], [22:20], [26:24], [30:28]	rw	**Node Pointer for Channel x** This bit field defines which service request output becomes activated if the channel x event of kernel ADCy occurs while enabled by **CHCTRx (x = 0 - 15)**.LCC. 000_B ADCy_SR0 is selected. 001_B ADCy_SR1 is selected. ... 111_B ADCy_SR7 is selected.
0	3, 7, 11, 15, 19, 23, 27, 31	r	**Reserved** Read as 0; should be written with 0.

23.2.15 Conversion Result Handling

The result generation part handles the:

- Storage of the conversion results (see **Section 23.2.15.1**)
- Wait-for-read mode (see **Section 23.2.15.2**)
- Result event interrupts (see **Section 23.2.15.3**)
- Result FIFO buffer (see **Section 23.2.15.4**)
- Data reduction or anti-aliasing filtering (see **Section 23.2.15.5**)

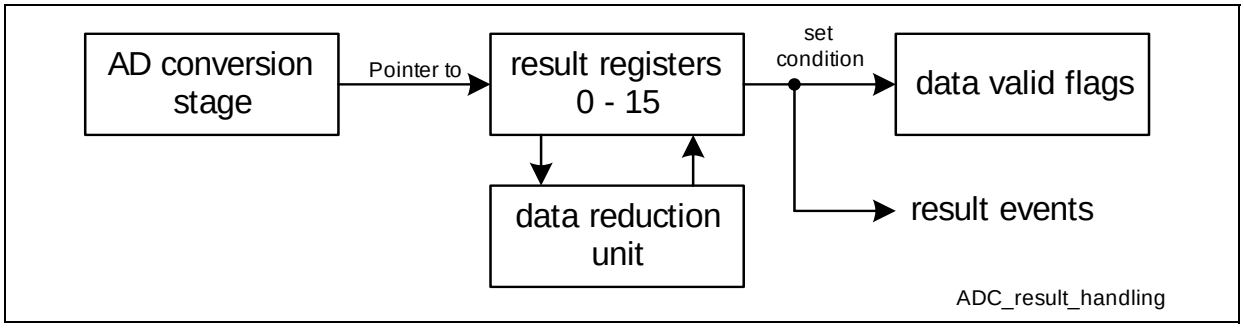


Figure 23-17 Conversion Result Handling

23.2.15.1 Storage of Conversion Results

For each analog input channel, the associated channel control register **CHCTRx (x = 0 - 15)** contains a pointer bit field (RESRSEL) defining the result register to store the conversion result of this channel. This structure allows the user to direct conversion results of different channels to one or more result registers. Depending on the application needs (data reduction, auto-scan, alias feature, result FIFO, etc.), the user can distribute the conversion results to minimize CPU load or to be more tolerant against interrupt latency.

An individual data valid flag **VFR**.VFx for each result register indicates that “new” valid data has been stored in the corresponding result register and can be read out.

Due to different result handling mechanisms, the conversion result can be represented in different ways:

- **Data reduction filter disabled:**
  The conversion result is maximum 12 bits wide with the MSB of the conversion result being always at bit position 11 and the remaining LSBs filled with 0.
  The data valid flag is set and a result event occurs each time a new conversion result is stored in the result register.
  It is possible to share a result register among several analog input channels.
- **Data reduction filter enabled:**
  The conversion result is maximum 12 bits wide with the MSB of the conversion result being always at bit position 11 and the remaining LSBs filled with 0. The additional bits [13:12] show the MSBs of the data accumulation.

The data valid flag is set and a result event occurs each time a data reduction sequence is finished and the final result is available in the result register.

In order to support a wait-for-read and FIFO buffer features, the valid flag has to be cleared automatically when SW does a read access or the result is transferred into another FIFO element (if result FIFO buffering is enabled).
This behavior is contradictory to debugging requirements. For debugging, it has to be possible to introduce read or write commands into the normal program flow, e.g. to monitor conversion results. If a debugger reads out a result register, it would change the status of the conversion result from valid = “new” (not yet read out) to “old” (already read out). This would have an undesired impact on the application.
Therefore, the read views with “D” deliver the same value as the read views without “D”, but without clearing the valid bit. As a result, a debugger using read views with “D” can monitor the conversion results without influencing their status for the application.

To allow debugger accesses without the risk of data sequence corruption, two different result register read views are supported. The read views refer to the same result register contents, but show a different behavior according to the address that has been read:

- Standard read view **RESR0** and **RESRx (x = 1 - 15)**:
  A read action clears the corresponding valid bit.
- Read view **RESRD0** and **RESRDx (x = 1 - 15)** for debugger:
  A read action does not clear the corresponding valid bit.

23.2.15.2 Wait-for-Read Mode

The wait-for-read mode is a feature of a result register allowing the CPU (or DMA) to treat each conversion result independently without the risk of data loss. Data loss could occur if the CPU does not read a conversion result from a result register before a new result overwrites the previous one.
Especially for auto-scan conversion sequences (or other sequences with “relaxed” timing requirements), the wait-for-read offers the possibility to request a conversion sequence according to an event (HW or SW), but to start a new conversion according to the CPU capability to read the formerly converted result.

If wait-for-read mode is enabled for a result register by setting bit WFR in register **RCRx (x = 0 - 15)**, a request source does not generate a conversion request while the targeted result register contains valid data (indicated by the valid flag VFx = 1) or if a currently running conversion targets the same result register.
A new conversion request is generated only after the targeted result register has been read out.
If two request sources target the same result register with wait-for-read selected, a lower priority request started before the higher priority source has requested its conversion can not be interrupted by the higher priority request. If a higher priority request targets a different result register, the lower priority conversion can be cancelled and repeated afterwards.

23.2.15.3 Result Event Interrupts

A result event interrupt can be generated based on a result event according to the structure shown in **Figure 23-18**. If a result event is detected, it sets the corresponding indication flag in register **EVFR**. These flags can also be set by writing a 1 to the corresponding bit position, whereas writing 0 has no effect. The indication flags can be cleared by SW by writing a 1 to the corresponding bit position in register **EVFCR**.

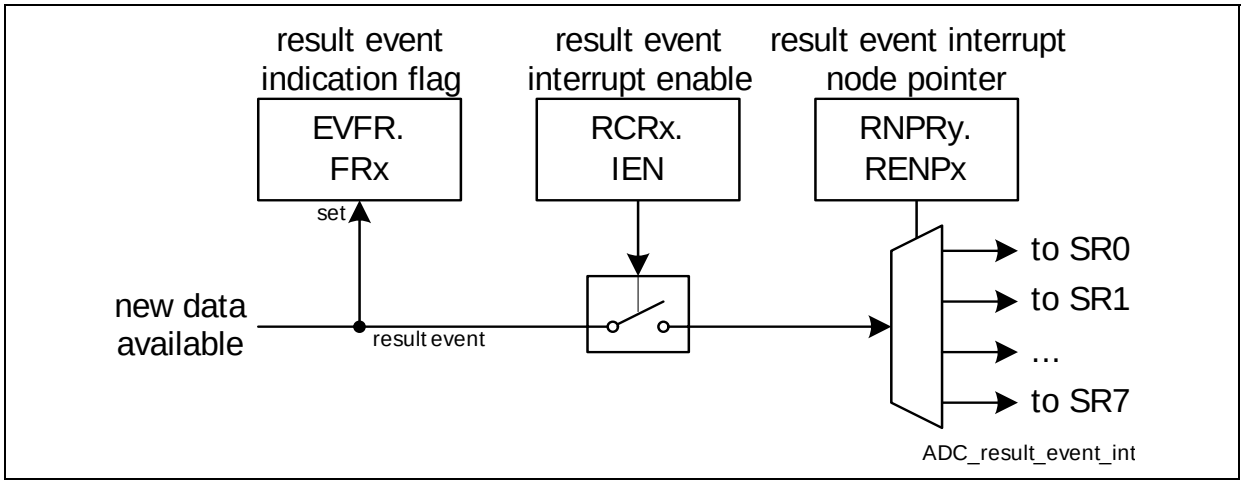


Figure 23-18 Result Event Interrupt Generation

The service request output ADCy_SRx that is selected by the result event interrupt node pointer bit fields in registers **RNPR0** or **RNPR8** issues an interrupt each time the related result event is detected.
A service request output can be activated under SW control by writing **INTR**.SISRx.

23.2.15.4 Result FIFO Buffer

If a result register is not used as direct target for a conversion result, it can be concatenated with other result registers of the same ADC kernel to form a result FIFO buffer (first-in-first-out buffer mechanism). This allows to store measurement results and to read them out later with a “relaxed” CPU access timing. It is possible to set up more than one FIFO buffer structure with the available result registers.
A FIFO structure can be built by at least two “neighbor” result registers with the indices x and z = x+1, where result register z represents the input and result register x represents the output of the FIFO buffer. The conversion result has to be delivered by the converter stage to the FIFO input, whereas the buffered data has to be read out from the FIFO output.
The FIFO buffer function can be enabled by setting bit FEN in registers **RCRx (x = 0 - 15)**.

In the example shown in **Figure 23-19**, the result registers have been configured to form two FIFO buffers with two buffer stages (result registers 0/1 and 6/7, respectively), one FIFO buffer with three buffer stages (result registers 2/3/4), whereas result register 5 is used as “normal” result register without additional FIFO buffer functionality.

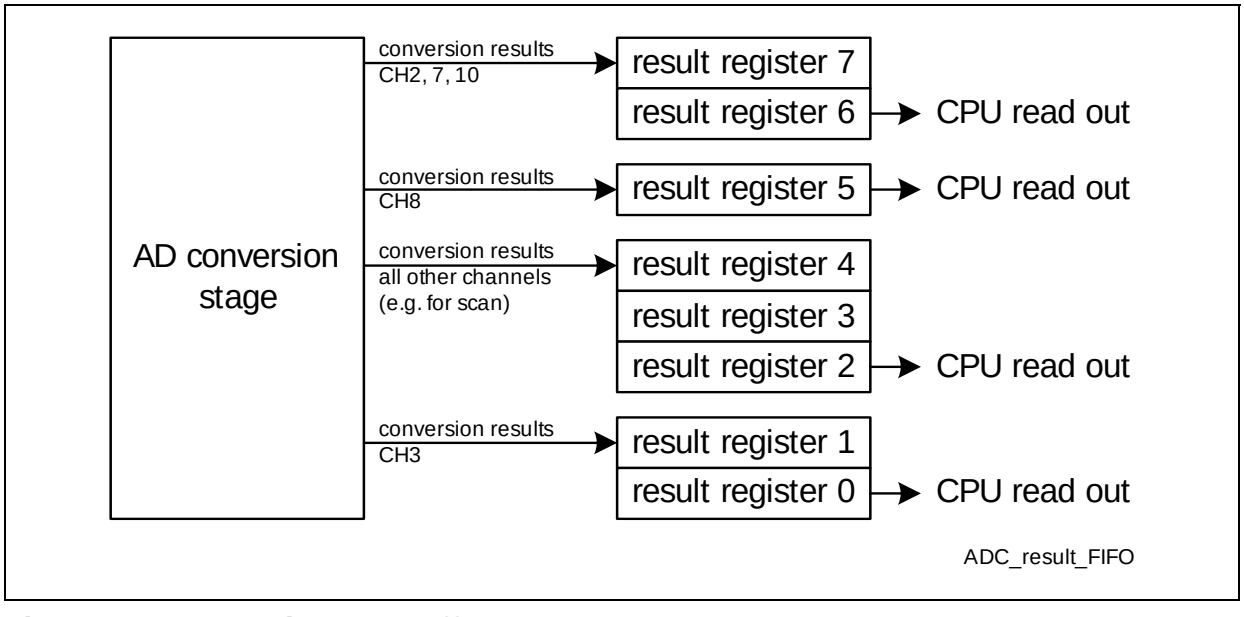


Figure 23-19 Result FIFO Buffers

If more than two result neighbor registers are concatenated to a FIFO buffer (from result register z to result register x, with z > x), the one with the highest index (z) is always the input and the one with the lowest index (x) is always the output. All intermediate result registers y (x < y < z) are used as intermediate FIFO stages without data input or data output functionality.

Result register features for each FIFO buffer:

- Result register z (**FIFO buffer input**):
  This result register can be enabled for data reduction. The wait-for-read mode is supported to avoid data loss if the FIFO is full. Result event interrupt generation is not supported. Must not be read at a read view modifying the valid bit.
- Result register y (**intermediate buffer stage**):
  This/these result register(s) must not be enabled neither for wait-for-read mode, nor for data reduction. Result event interrupt generation is not supported. Must not be read at a read view modifying the valid bit, nor be the target of a conversion result.
- Result register x (**FIFO buffer output**):
  This result register can be enabled for result event interrupt generation to inform the CPU that new data can be read out from this register location. Data reduction and wait-for-read are not supported and have to be disabled. Must not be the target of a conversion result.
  If enabled, a result interrupt is generated for each data word in the FIFO.

23.2.15.5 Data Reduction Filter

The data reduction filter can be used as digital filter for anti-aliasing or decimation purposes. It can accumulate a maximum of 4 conversion results to generate a final result.

Each result register can be individually enabled for data reduction. The feature is controlled by bit field DRCTR in registers **RCRx (x = 0 - 15)**. The actual status is given by bit field DRC (data reduction counter) in the related result register.
Conversion delivering results to other result registers do not influence the data reduction filter of result register x. As a consequence, other channels can be converted between two conversions targeting result register x.

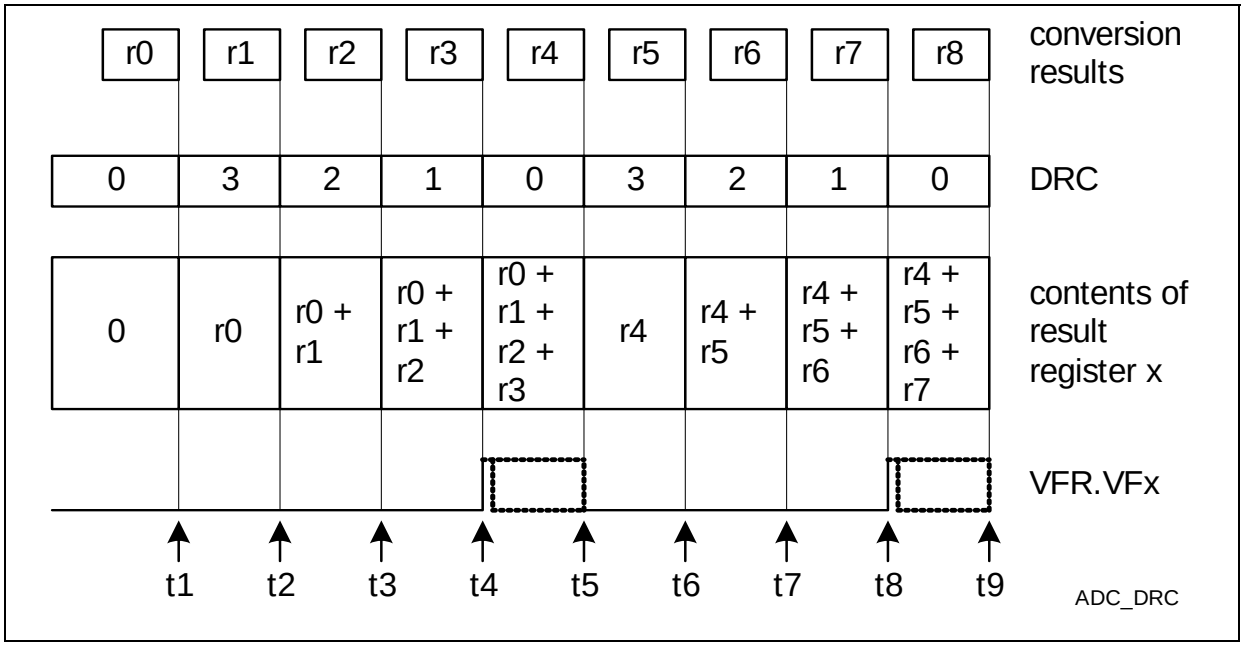


Figure 23-20 Data Reduction Filter

In the example given in **Figure 23-20**, a data reduction sequence of 4 accumulated conversion results is shown. The data reduction is based on three rules:

- Each time bit field DRC is 0 and a conversion targeting result register x is completed (t1, t5, t9), the contents of bit field RCRx.DRCTR is loaded into bit field DRC and the conversion result is stored in result register x.
- Each time bit field DRC is not 0 and a conversion targeting result register x is completed (t2, t3, t4 for the first final result and t6, t7, t8 for the next one), bit field DRC is decremented by 1 and the conversion result is added to the value already stored in result register x.
- Each time bit field DRC is 0 after decrementing or after loading it with RCRx.DRCTR = 0 (t4 for the first final result and t8 for the next one), the valid bit for the result register x becomes set and a result register event occurs.

The final result of a data reduction sequence has to be read out from result register x before the next data reduction sequence starts (interval between t4 and t5, or t8 and t9 respectively). With the read out of the final result from this register, the valid flag is automatically cleared.
If this interval is too short, it is recommended to associate a second result register z to result register x by enabling the result FIFO mechanism for result register x, see **Figure 23-21** (z = x + 1). In this case, result register x is loaded with the final result elaborated by result register z when a data reduction sequence is finished. The final result has to be read out from result register x before the next data reduction sequence is finished (interval between t4 and t8).

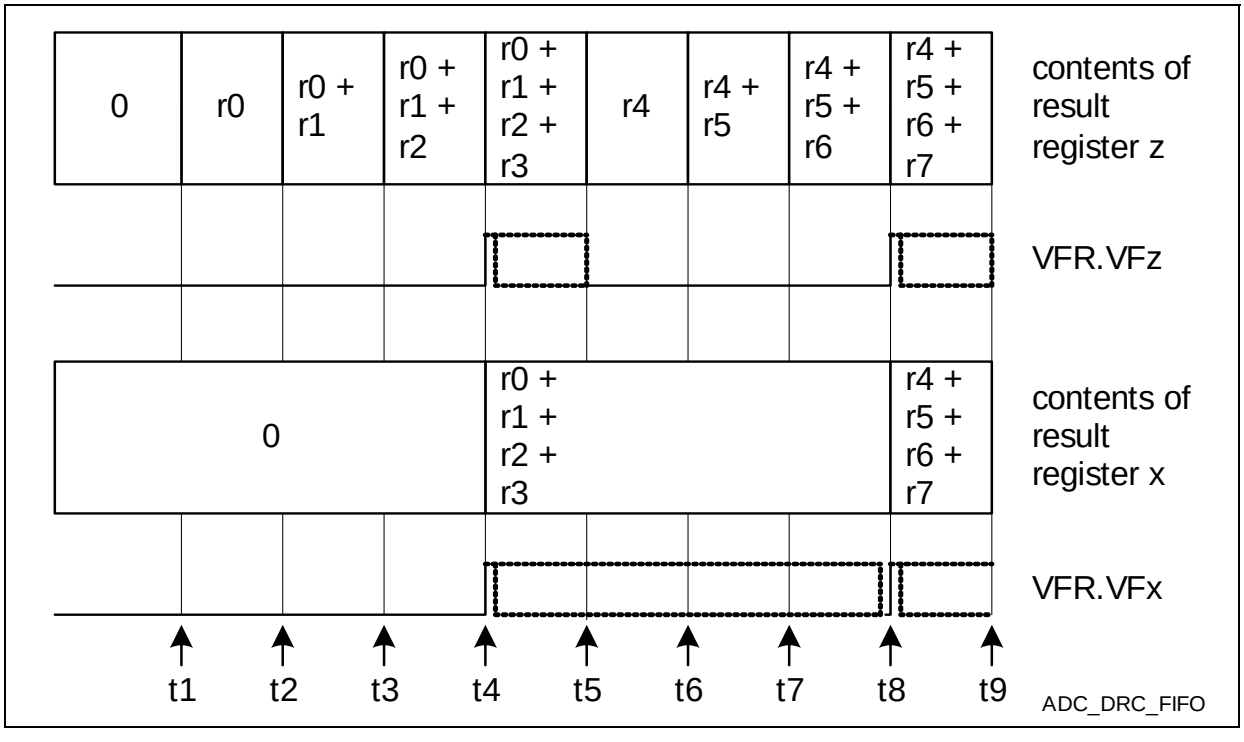


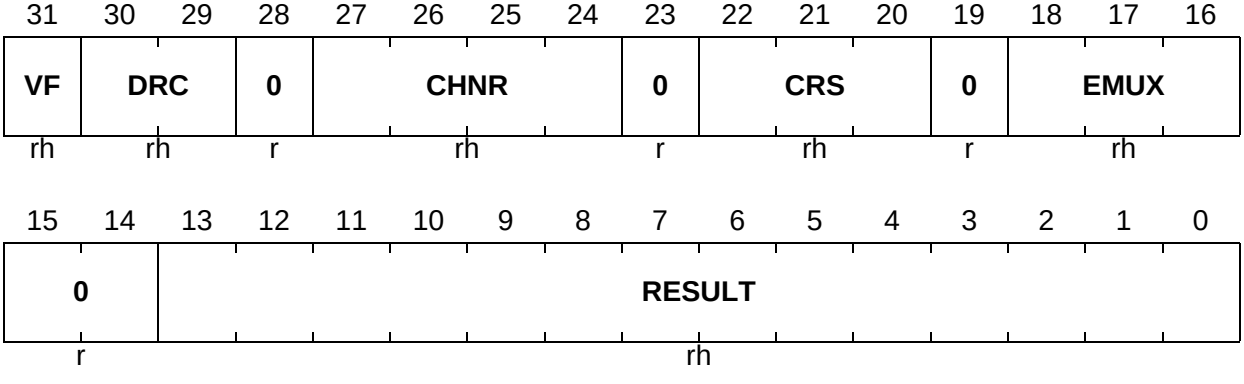
Figure 23-21 Data Reduction Filter with Result FIFO

23.2.16 Conversion Result-Related Registers

23.2.16.1 Result Register 0

The result registers deliver the conversion results and associated information.
Additionally to this information, result register RESR0 also indicates the setting of an external analog multiplexer. The conversion results of the channel used with an external multiplexer have to be directed to RESR0 in order to indicate the multiplexer setting used during the conversion. If this information is not necessary, the conversion result can be directed to any other result register.
The valid flag VF indicates that a result register contains updated data and can be used to poll for new data. The valid flag of a result register is cleared automatically if at least the low byte of register RESR0 is read, whereas it is left unchanged when reading RESRD0.

RESR0
Result Register 0 **(180_H)** **Reset Value: 0000 0000$_H$**
RESRD0
Result Register 0 for Debugging **($1C0_H$)** **Reset Value: 0000 0000$_H$**



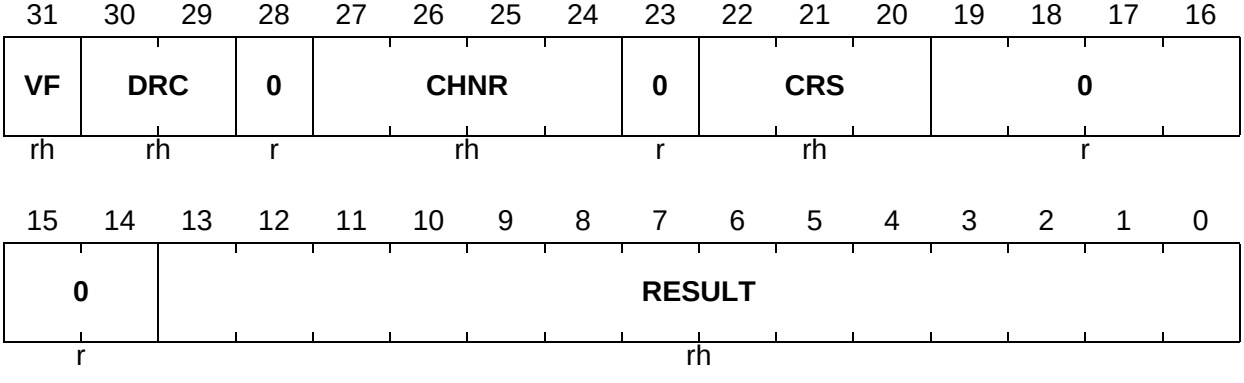
Field	Bits	Type	Description
RESULT	[13:0]	rh	**Conversion Result** This bit field contains the conversion result, or respectively, the result of the data reduction filter.
EMUX	[18:16]	rh	**External Multiplexer Setting** This bit field indicates the external multiplexer setting leading to the result stored in bit field RESULT.

Field	Bits	Type	Description
CRS	[22:20]	rh	**Converted Request Source** This bit field indicates the request source that has requested the conversion leading to the result stored in bit field RESULT. 000_B The conversion was requested by source 0. 001_B The conversion was requested by source 1. 010_B The conversion was requested by source 2. 011_B The conversion was requested by source 3. 100_B The conversion was requested by source 4. else reserved
CHNR	[27:24]	rh	**Channel Number** This bit field contains the channel number of the latest register update.
DRC	[30:29]	rh	**Data Reduction Counter** This bit field indicates how many conversion results have still to be accumulated to generate the final result for data reduction. The valid flag is automatically set when this bit field becomes 0. It can be cleared by SW by writing a 1 to the related bit position in register VFR. 00_B The final result is available in the result register. The valid flag is automatically set when this bit field is set to 0. 01_B 1 more conversion result has to be added to obtain the final result in the result register. 10_B 2 more conversion results have to be added to obtain the final result in the result register. 11_B 3 more conversion results have to be added to obtain the final result in the result register.
VF	31	rh	**Valid Flag** This bit indicates that bit field RESULT has been updated with valid data since it has been read out. It is another view of the corresponding bit in register VFR. 0_B The result register has not been updated. 1_B The result register has been updated.
0	[15:14], 19, 23, 28	r	**Reserved** Read as 0; should be written with 0.

23.2.16.2 Result Registers 1 to 15

The result registers deliver the conversion results and associated information.
The valid flag VF indicates that the result register contain updated data and can be used to poll for new data. The valid flag of a result register is cleared automatically if at least the low byte of register RESRx is read, whereas it is left unchanged when reading RESRDx.

RESRx (x = 1 - 15)
Result Register x **(180_H + x * 4)** **Reset Value: 0000 0000_H**
RESRDx (x = 1 - 15)
Result Register x for Debugging **($1C0_H$ + x * 4)** **Reset Value: 0000 0000_H**



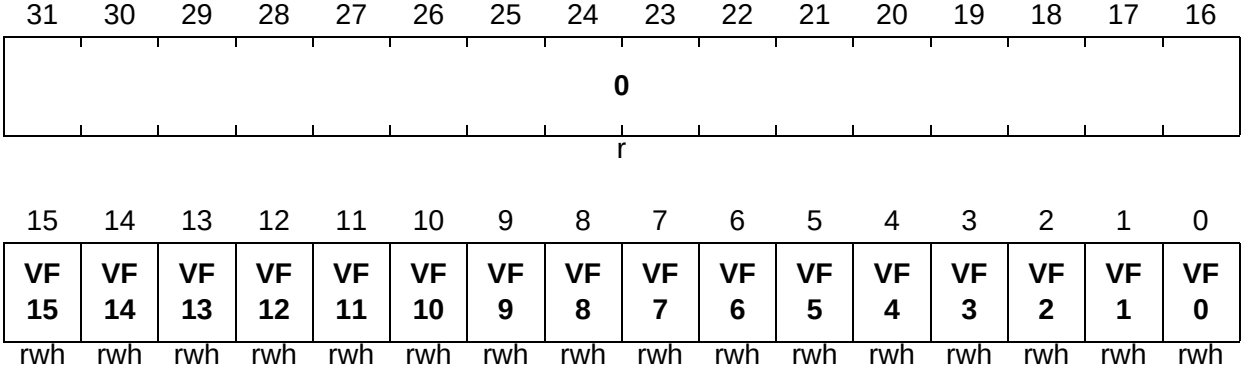
Field	Bits	Type	Description
RESULT	[13:0]	rh	**Conversion Result** This bit field contains the conversion result, or respectively, the result of the data reduction filter.
CRS	[22:20]	rh	**Converted Request Source** This bit field indicates the request source that has requested the conversion leading to the result stored in bit field RESULT. 000_B The conversion was requested by source 0. 001_B The conversion was requested by source 1. 010_B The conversion was requested by source 2. 011_B The conversion was requested by source 3. 100_B The conversion was requested by source 4. else reserved
CHNR	[27:24]	rh	**Channel Number** This bit field contains the channel number of the latest register update.

Field	Bits	Type	Description
DRC	[30:29]	rh	**Data Reduction Counter** This bit field indicates how many conversion results have still to be accumulated to generate the final result for data reduction. The valid flag is automatically set and a result event is generated when this bit field becomes 0 (by decrementing or by reload). Bit field DRC is cleared by writing the related **VFR**.VFx = 1. 00_B The final result is available in the result register. 01_B 1 more conversion result has to be added to obtain the final result in the result register. 10_B 2 more conversion results have to be added to obtain the final result in the result register. 11_B 3 more conversion results have to be added to obtain the final result in the result register.
VF	31	rh	**Valid Flag** This bit indicates that bit field RESULT has been updated with valid data since it has been read out. It is another view of the corresponding bit in register VFR. 0_B The result register has not been updated. 1_B The result register has been updated.
0	[19:14], 23, 28	r	**Reserved** Read as 0; should be written with 0.

23.2.16.3 Valid Flag Register

The valid flag register contains the flags indicating that the corresponding result register contents are valid (valid = “new” = not read out).
These bits are another (condensed) view of the valid flags in the result registers.

VFR
Valid Flag Register **(200_H)** **Reset Value: $0000\ 0000_H$**

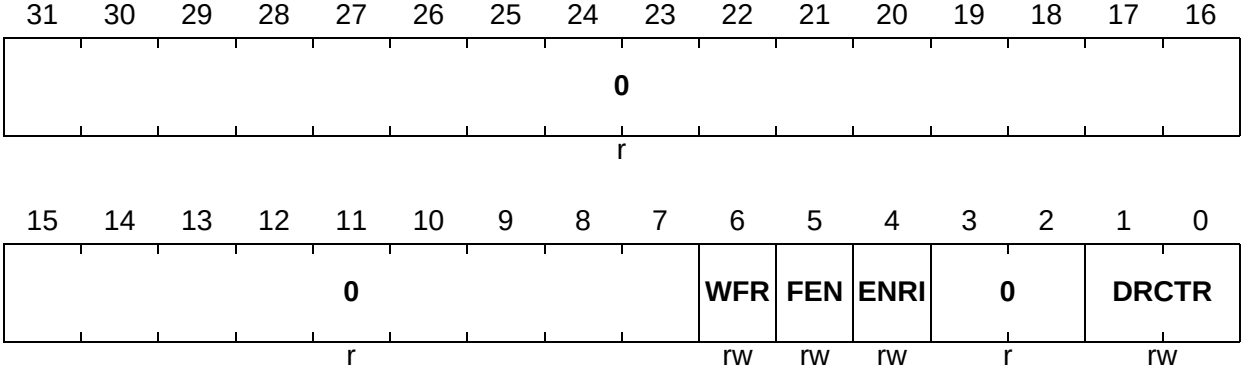


Field	Bits	Type	Description
VFx (x = 0 - 15)	x	rwh	**Valid Flag for Result Register x** This bit indicates that the contents of the result register x is valid. Writing a 0 has no effect, whereas writing a 1 clears the written bit position and the bit field DRC in the related result register. If a hardware event triggers the setting of a bit VFx and SW writes a 1 to the same bit position, the bit VFx is cleared (software overrules hardware). 0_B The result register x does not contain valid data. Either this register has been read out or no data has been moved to it. 1_B The result register x contains valid data that has not yet been read out.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

23.2.16.4 Result Control Registers

The result control registers contain bits to control the behavior of the result registers and to monitor their status. Result register x is controlled by result control register x.

RCRx (x = 0 - 15)
Result Control Register x **(140_H + x * 4)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
DRCTR	[1:0]	rw	**Data Reduction Control** This bit field defines how many conversion results are accumulated for data reduction (see **Section 23.2.15.5**). It defines the reload value for bit field DRC. 00_B The data reduction filter is disabled. The reload value for DRC is 0, so no accumulation is done. 01_B The data reduction filter is enabled. The reload value for DRC is 1, so the accumulation is done over 2 conversions. 10_B The data reduction filter is enabled. The reload value for DRC is 2, so the accumulation is done over 3 conversions. 11_B The data reduction filter is enabled. The reload value for DRC is 3, so the accumulation is done over 4 conversions.
ENRI	4	rw	**Enable Result Interrupt** This bit enables the result event interrupt if a result event is detected for result register x. 0_B The result event interrupt is disabled. 1_B The result event interrupt is enabled.

Field	Bits	Type	Description
FEN	5	rw	**FIFO Enable** This bit enables the FIFO functionality for result register x, see **Section 23.2.15.4**. 0_B The FIFO functionality is disabled. 1_B The FIFO functionality is enabled.
WFR	6	rw	**Wait-for-Read Mode** This bit enables the wait-for-read mode for result register x. 0_B The wait-for-read mode is disabled. 1_B The wait-for-read mode is enabled.
0	[3:2], [31:7]	r	**Reserved** Read as 0; should be written with 0.

23.2.16.5 Event Flag Register

The event flag register contains flags related to request source events and result register events.

EVFR
Event Flag Register **(070_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0			**GFS4**	**GFS3**	**GFS2**	**GFS1**	**GFS0**	**0**			**FS4**	**FS3**	**FS2**	**FS1**	**FS0**
r			rh	rh	rh	rh	rh	r			rwh	rwh	rwh	rwh	rwh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FR15	**FR14**	**FR13**	**FR12**	**FR11**	**FR10**	**FR9**	**FR8**	**FR7**	**FR6**	**FR5**	**FR4**	**FR3**	**FR2**	**FR1**	**FR0**
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
FRx (x = 0 - 15)	x	rwh	**Event Flag for Result Register x** Flag FRx indicates that a result event of result register x has been detected. Writing a 0 has no effect, whereas writing a 1 sets the written bit position without generating an interrupt. Bit FRx is cleared by writing EVFCR.CFRx = 1. 0_B An event of result register x has not yet been detected. 1_B An event of result register x has been detected.
FSx (x = 0 - 4)	x + 16	rwh	**Event Flag for Request Source x** Flag FSx indicates that a request source event of request source x has been detected. Writing a 0 has no effect, whereas writing a 1 sets the written bit position without generating an interrupt. Bit FSx is cleared by writing EVFCR.CFSx = 1. 0_B An event of request source x has not yet been detected. 1_B An event of request source x has been detected.

Field	Bits	Type	Description
GFSx (x = 0 - 4)	x + 24	rh	**Gated Event Flag for Request Source x** Flag GFSx indicates that a request source event of request source x has been detected while the related interrupt was enabled. Writing to this bit position has no effect. Bit GFSx is cleared by writing EVFCR.CFSx = 1. 0_B An event of request source x has not yet been detected or the related interrupt was not enabled. 1_B An event of request source x has been detected while the related event interrupt was enabled.
0	[23:21], [31:29]	r	**Reserved** Read as 0; should be written with 0.

23.2.16.6 Event Flag Clear Register

Writing a 1 to a bit position in register EVFCR clears the corresponding event flag in register EVFR. If a hardware event triggers the setting of bit EVFR.x and software writes EVFCR.x = 1, bit EVFR.x is cleared (software overrules hardware).

EVFCR
Event Flag Clear Register (074$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0											CFS4	CFS3	CFS2	CFS1	CFS0
r											w	w	w	w	w

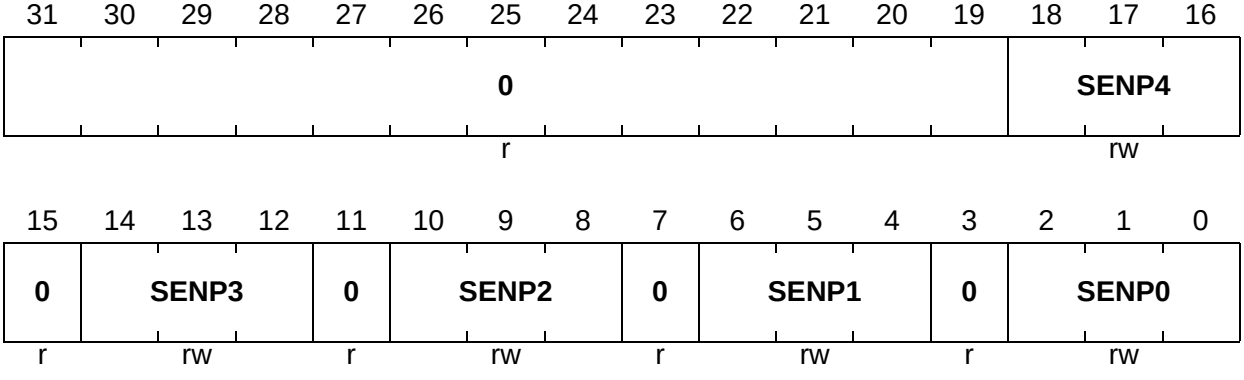
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CFR15	CFR14	CFR13	CFR12	CFR11	CFR10	CFR9	CFR8	CFR7	CFR6	CFR5	CFR4	CFR3	CFR2	CFR1	CFR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
CFRx (x = 0 - 15)	x	w	**Clear Event Flag for Result Register x** 0_B No action. 1_B Bit EVFR.FRx is cleared.
CFSx (x = 0 - 4)	x + 16	w	**Clear Event Flag for Source x** 0_B No action. 1_B Bits EVFR.FSx and EVFR.GFSx are cleared.
0	[31:21]	r	**Reserved** Read as 0; should be written with 0.

23.2.16.7 Event Node Pointer Registers

The bit fields in these registers define the service request output ADCy_SR[7:0] that is activated if a request source event or a result register event occurs and the interrupt generation is enabled for this event.

EVNPR
Event Node Pointer Register (078_H) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
SENP0, SENP1, SENP2, SENP3, SENP4	[2:0], [6:4], [10:8], [14:12], [18:16]	rw	**Node Pointer for Request Source x** This bit field defines which service request output becomes activated if the request source x event of kernel ADCy occurs while the interrupt generation is enabled for this event. 000_B ADCy_SR0 is selected. 001_B ADCy_SR1 is selected. ... 111_B ADCy_SR7 is selected.
0	3, 7, 11, 15, [31:19]	r	**Reserved** Read as 0; should be written with 0.

RNPR0
Result Node Pointer Register 0 (208_H) Reset Value: 0000 0000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**RENP7**			**0**	**RENP6**			**0**	**RENP5**			**0**	**RENP4**		
r	rw			r	rw			r	rw			r	rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**RENP3**			**0**	**RENP2**			**0**	**RENP1**			**0**	**RENP0**		
r	rw			r	rw			r	rw			r	rw		

Field	Bits	Type	Description
RENP0, RENP1, RENP2, RENP3, RENP4, RENP5, RENP6, RENP7	[2:0], [6:4], [10:8], [14:12], [18:16], [22:20], [26:24], [30:28]	rw	**Node Pointer for Result Register x** This bit field defines which service request output becomes activated if the result register x event of kernel ADCy occurs while the interrupt generation is enabled for this event. 000_B ADCy_SR0 is selected. 001_B ADCy_SR1 is selected. ... 111_B ADCy_SR7 is selected.
0	3, 7, 11, 15, 19, 23, 27, 31	r	**Reserved** Read as 0; should be written with 0.

RNPR8
Result Node Pointer Register 8 ($20C_H$) Reset Value: $0000\ 0000_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	**RENP15**			**0**	**RENP14**			**0**	**RENP13**			**0**	**RENP12**		
r	rw			r	rw			r	rw			r	rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**RENP11**			**0**	**RENP10**			**0**	**RENP9**			**0**	**RENP8**		
r	rw			r	rw			r	rw			r	rw		

Field	Bits	Type	Description
RENP8, RENP9, RENP10, RENP11, RENP12, RENP13, RENP14, RENP15	[2:0], [6:4], [10:8], [14:12], [18:16], [22:20], [26:24], [30:28]	rw	**Node Pointer for Result Register x** This bit field defines which service request output becomes activated if the result register x event of kernel ADCy occurs while the interrupt generation is enabled for this event. 000_B ADCy_SR0 is selected. 001_B ADCy_SR1 is selected. ... 111_B ADCy_SR7 is selected.
0	3, 7, 11, 15, 19, 23, 27, 31	r	**Reserved** Read as 0; should be written with 0.

23.2.17 Multiplexer Test Support

A specific multiplexer test mode has been implemented for the analog input CH7 that can be enabled during run time by the user to check the connection to the sensor.

- **Multiplexer test mode disabled** (**GLOBCFG**.MTM7 = 0):
  The switch for the voltage divider and static load R_{MTM7} is open. The analog input CH7 can be used for normal measurements.
- **Multiplexer test mode enabled** (**GLOBCFG**.MTM7 = 1):
  The switch for the voltage divider and static load R_{MTM7} is closed. The analog input CH7 is loaded by a resulting resistance and the measured voltage is reduced by a voltage divider.
  Please refer to the AC/DC chapter for the value of the resulting grounding resistor and its current capability.

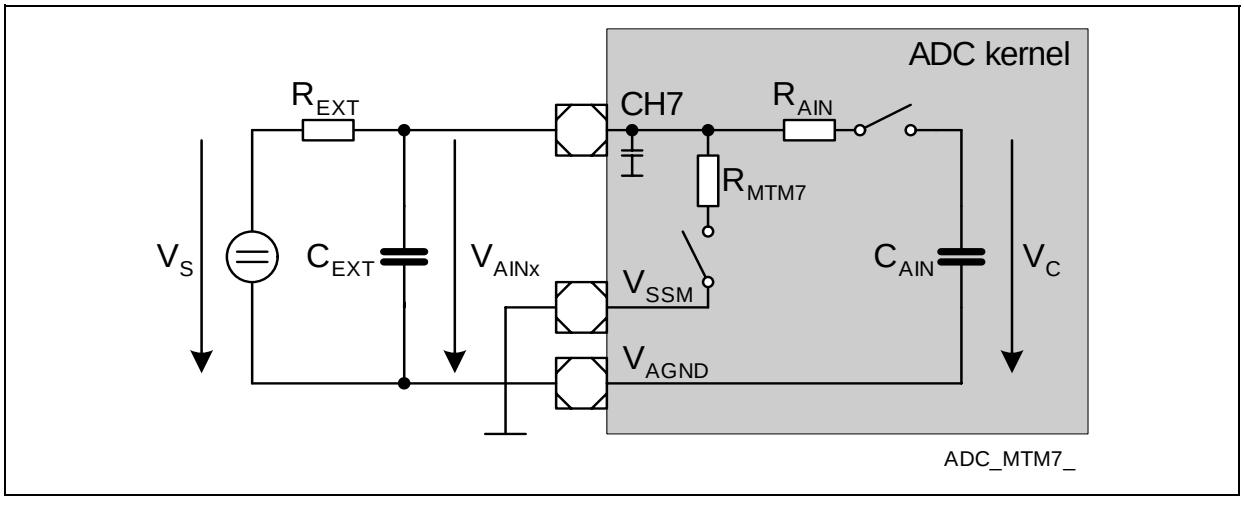


Figure 23-22 Multiplexer Test Mode for CH7

23.2.18 External Multiplexer Control

If an application requires more analog inputs channels than available on the TC1797, the ADC kernel supports an extension of analog channels by adding an external analog multiplexer. Three output signals EMUX[2:0] are delivered by each ADC kernel to control the settings of an external analog multiplexer. They can be used to extend the number of analog input channels by adding an external 1-out-of-8 multiplexer.
The external multiplexer control behavior is defined by the bits in register **EMCTR**.
The current setting of EMUX[2:0] is given by bit field EMUX. If another extended input channel should be converted, bit field SETEMUX has to be programmed to the desired value or the scan function has to be enabled. The SETEMUX value is automatically applied with the start of the next conversion of the related analog ADC input channel.

In the example shown in **Figure 23-23** and in the description below, the analog input CH7 has been extended, leading to additional analog inputs named CH70 to CH77. The channel number where the external multiplexer is connected to is defined by bit field EMUXCHNR.

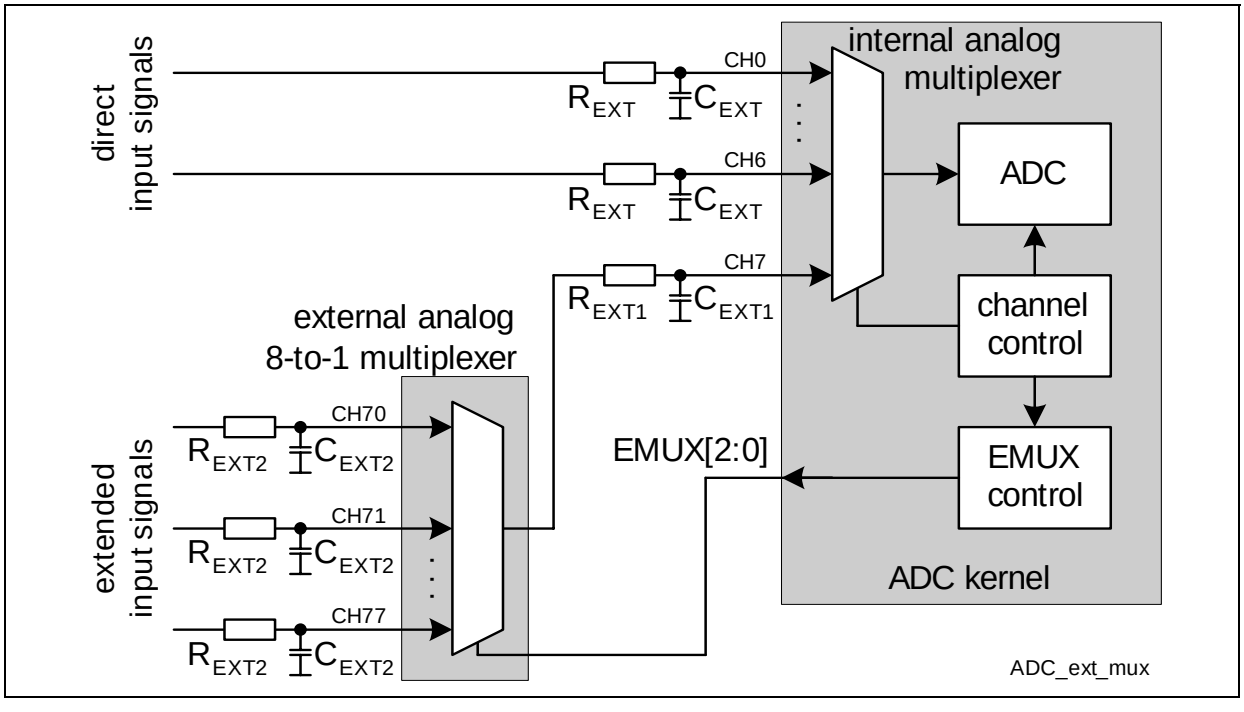


Figure 23-23 External Analog Multiplexer

If the external multiplexer is located far from the ADC analog input, it is recommended to introduce an RC filter R_{EXT1}-C_{EXT1} directly at the analog input CH7 of the ADC. If needed for signal filtering, local RC filters R_{EXT2}-C_{EXT2} can be optionally added at the inputs of the external analog multiplexer.
If the external multiplexer is located close to the analog ADC input, the components R_{EXT1} and C_{EXT1} are not necessarily needed. In this case it is strongly recommended to

introduce RC filters (R_{EXT2}, C_{EXT2}) at the multiplexer inputs.
Please note that each RC filter limits the bandwidth of the analog input signal.

The RC filters used with an external multiplexer may lead to another impedance “seen” by the ADC analog input CH7 than for the other (direct) analog inputs. The adaptation of the sample phase length can be done by using a different input class with a different value for the sample phase extension. This value can be adapted to execute conversions with an EMUX[2:0] setting that has changed a sufficiently long time before the conversion of CH7 starts. “A sufficiently long time before” signifies that signal transitions at the analog ADC input due to changing multiplexer setting are finished and the input signal is stable enough.
After changing the EMUX[2:0] setting of the external multiplexer, an additional settling time has to elapse before the switched analog signal is stable and can be measured. To compensate for this settling time, an alternative sample phase length (instead of the one given by the input class) is automatically applied for the first conversion of CH7 after EMUX[2:0] has changed. The alternative sample phase length can be programmed by bit field **EMCTR**.EMSAMPLE. If the first conversion of CH7 after the EMUX[2:0] setting has changed is aborted due to a higher priority request, the repeated conversion of CH7 also uses the value of EMSAMPLE. The settling time is considered to be finished after the complete conversion of CH7.

The external multiplexer control block supports different modes, that are programmed by the bits in register **EMCTR**:

- **SW control** without any HW interaction (EMUXEN = 0):
  The automatic control of the external multiplexer setting and of the sampling time is disabled. Bit field EMUX is permanently updated with the value of SETEMUX. The changes of EMUX are related to write actions to SETEMUX and not to conversion timing. The setting of EMSAMPLE is not taken into account. It is recommended to write the start value of the first scan sequence to SETEMUX while EMUXEN = 0.
- **HW control without scan** (EMUXEN = 1, SCANEN = 0):
  The update of EMUX with the value of SETEMUX happens with each conversion start of the channel selected by EMUXCHNR. For the first conversion with a new EMUX value, the setting of EMSAMPLE is applied.
- **HW control with single-input scan** (EMUXEN = 1, SCANEN = 1, TROEN = 0):
  The update of EMUX with a new value happens after each conversion of the channel selected by EMUXCHNR. For each update, EMUX is automatically decremented by 1. If EMUX = 0, it is reloaded with the value of SETEMUX for the next update. For each conversion of the selected channel, the setting of EMSAMPLE is applied.
  With this setting, an autoscan sequence requesting the conversion of the channel defined by EMUXCHNR leads to one conversion of the channel connected to the external multiplexer. As a result, for each completed auto scan sequence, another EMUX setting is applied.
  Assuming inputs 1, 2, 70, 71, and 72 being selected for scan, the following sequence will be executed: 1, 2, 72, 1, 2, 71, 1, 2, 70, 1, 2, 72, 1, 2, 71, 1, 2, 70, 1, 2, 72, ...

- **HW control with multi-input scan** (EMUXEN = 1, SCANEN = 1, TROEN = 1):
  The update of EMUX with a new value happens after each conversion of the channel selected by EMUXCHNR. For each update, EMUX is automatically decremented by 1. If EMUX = 0, it is reloaded with the value of SETEMUX for the next update. For each conversion of the selected channel, the setting of EMSAMPLE is applied.
  With enabled trigger option, the external multiplexer control block triggers a new conversion request each time a conversion is started of the channel defined by EMUXCHNR while EMUX > 0.
  In a scan request source, the corresponding pending bit becomes set, whereas in a sequential request source, the content of the backup stage becomes valid (V bit of backup stage becomes set).
  With this setting, all external multiplexer inputs are scanned during a single autoscan sequence, starting with the channel indicated by SETEMUX (same update rate of all channels of this sequence).
  Assuming inputs 1, 2, 70, 71, and 72 being selected for scan, the following sequence will be executed: 1, 2, 72, 71, 70, 1, 2, 72, 71, 70, 1, 2, 72, 71, 70, 1, 2, 72, ...

23.2.19 Synchronized Conversions for Parallel Sampling

The independent ADC kernels implemented in the TC1797 can be synchronized for simultaneous (parallel) measurements of analog input channels. While no parallel conversion is requested, the kernels can work independently.
The synchronization mechanism for parallel conversions ensures that the sample phases of the related channel start simultaneously. Different values for the resolution and the sample phase length of each kernel for a parallel conversion are supported.
A parallel conversion can be requested individually for each input channel (also several channels can be enabled for parallel conversions). In the example shown in the figure below, input channels CH3 of the ADC kernels ADC0 and ADC1 are converted synchronously, whereas other input channels do not lead to parallel conversions.

This leads to the following structure:

- A **synchronization master** ADC kernel can request a conversion of an analog channel. If this channel is selected for a synchronized conversion, it is also requested in the connected slave ADC kernel(s).
- A **synchronization slave** ADC kernel reacts to incoming synchronized conversion requests from its master. While no incoming master requests are active, the slave kernel can convert its own requests.
- All ADC kernels in an ADC module being similar, each kernel can be set up to be a synchronization master or a synchronization slave (depending on the application needs, such as trigger capability of request sources).
- A synchronization master can synchronize several slave kernels, whereas a slave kernel can only be synchronized to one master kernel.

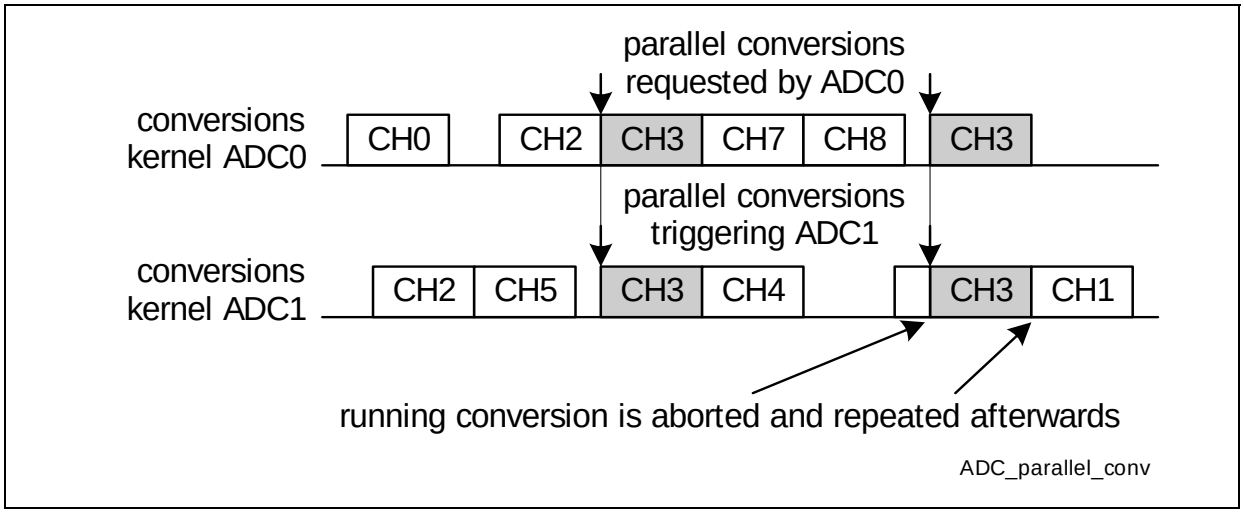


Figure 23-24 Parallel Conversions

The term “conversion group” has been introduced to define the kernel behavior allowing parallel sampling:

- Kernels in the same conversion group can execute parallel conversions.
- A conversion group contains at least 1 ADC kernel and can contain a maximum of all ADC kernels of the ADC module.
- Each conversion group contains exactly one synchronization master kernel that issues a parallel conversion request and defines the internal frequencies f_{ADCI} and f_{ADCD} and the channel number for a parallel conversion of the conversion group.
- All other kernels in a conversion group are synchronization slaves and have to be programmed with the same values of **GLOBCTR**.DIVA, DIVD and ARBRND as the synchronization master.
- If there is no need for parallel conversions, each kernel can be considered to form an own conversion group with only an ADC kernel as synchronization master, but without any synchronization slave.
- The channel number and the synchronization request are issued by the synchronization master to the kernels in the same synchronization group if a conversion is requested with **CHCTRx (x = 0 - 15)**.SYNC = 1 in the synchronization master kernel. Synchronization slaves can not issue synchronization requests.
- Once started, a parallel conversion can not be aborted.
- A parallel conversion request is always handled with highest priority and cancel-inject-repeat mode in a synchronization slave (see **Section 23.2.7.2**).
- Bit **GLOBCTR**.ARBM has to be 0 for synchronization slaves.
- The wait-for-read mode is supported for the master kernel, whereas the setting is ignored in the slave kernels (previous results may be overwritten).

The synchronization request issuing mechanism of the master to the slave kernels is based on bit field **GLOBSTR**.ANON. The information given by **GLOBCTR**.ANON is distributed by the synchronization master to all kernels in the conversion group (the bit fields **SYNCTR**.STSEL of all kernels must be programmed in a way that all kernels refer to the same information). In addition to the ANON information, the master delivers the requested channel number to the slave (not explicitly shown in **Figure 23-25**).
The start of the converters of all kernels of a conversion group is based on signals indicating when a kernel is ready and can start the sample phase of a parallel conversion. Bit **SYNCTR**.EVALRx defines if a kernel has to wait for the other kernel(s) (to allow parallel conversions) or can start without waiting (no parallel conversions possible). To support parallel conversions, all ready signals of the kernels of a conversion group have to be considered.

The alias feature is independent of synchronized conversions. All kernels of a conversion group request the same channel number (defined by the master), but can convert analog signals from different inputs. The requested channel number can be redirected by its alias setting. E.g., if the channel number requested in a conversion group is channel CH0, but for a kernel, an alternative reference is connected to this input,

the actually converted analog input can be changed to any value. This can be done by programming bit field ALIAS0 accordingly.

Note: A parallel conversion in a slave ADC should not target a result register that is already used for data reduction of other channels.

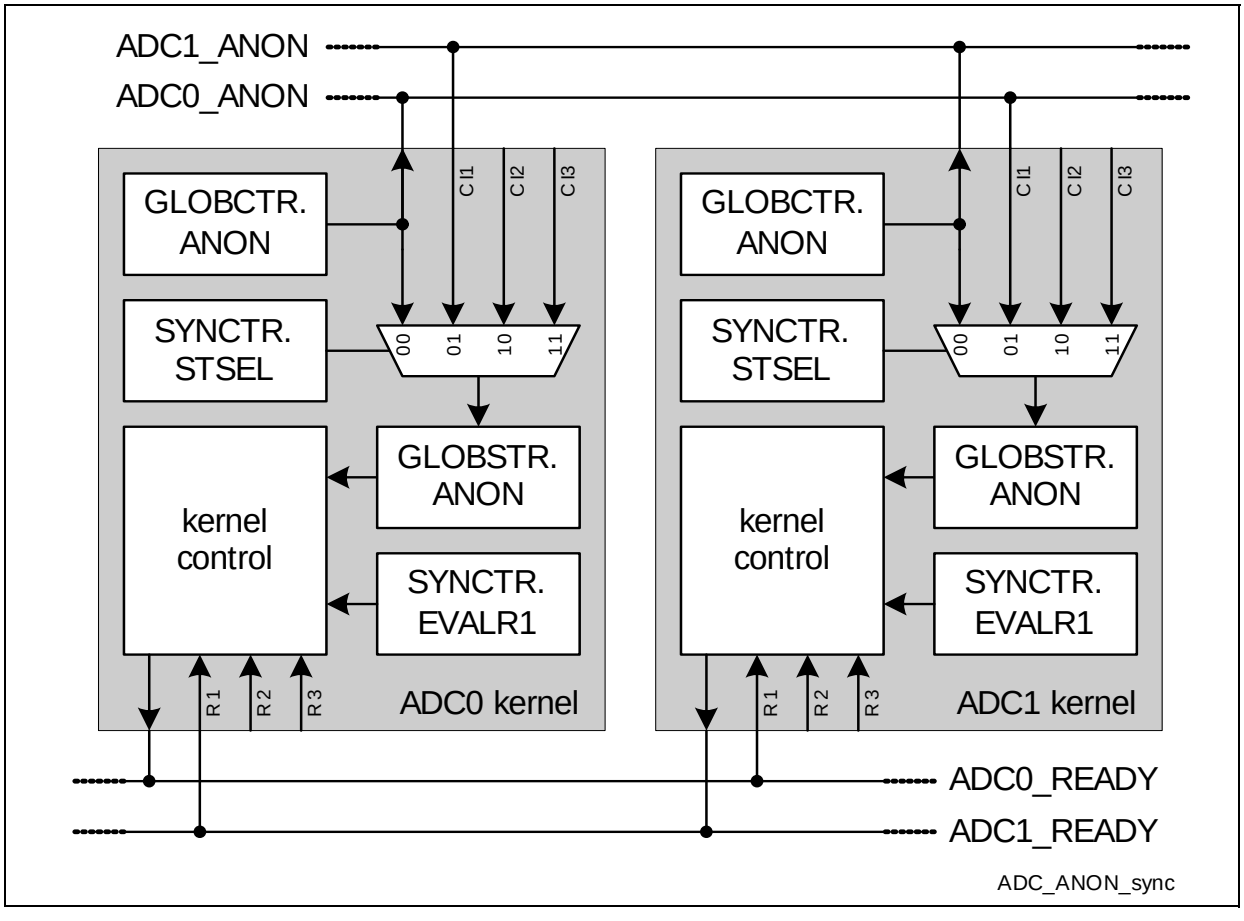


Figure 23-25 Synchronization via ANON and Ready Signals

The connections of the ADCx_ANON and ADCx_READY signals, as well as the programming hints for register EMCTR are described in the implementation chapter, see **Section 23.3**.

23.2.20 Equidistant Sampling

Each ADC kernel supports equidistant sampling of one (or more) analog input channels, e.g. for audio purposes or digital filters.
Therefore, each request source can be programmed to take part in the arbitration round and to win the arbitration (depending on the programmed priority levels), but without starting the conversion immediately. The exact start point of the conversion is given by a control signal (generated outside the ADC module, e.g. by a timer module) that is selected as trigger input REQTRx of request source x. Equidistant sampling is ensured if the REQTRx signal is generated synchronously to the arbiter timing, mainly for the arbiter. Each ADC kernel provides an output ARBCNT, that is activated once per arbitration round to count the arbiter cycles as timing base for the equidistant sampling by a timer located outside the ADC module.
A requested equidistant conversion can start its sampling phase if the converter is idle and the arbiter has decided which channel to convert. To ensure that the converter is idle, the arbiter decides which channel to convert (winner of the arbitration round), but it waits for the timer control signal to really start the measurement (preface time). If the request source selected for equidistant sampling has been programmed with the highest priority, no other request source can disturb the equidistant sampling.
The interpretation of the trigger signal REQTRx for equidistant sampling is enabled by selecting timer mode in the corresponding request source input register (RSIRx.TMEN = 1). The frequency of signal REQTRx defines the sampling rate and its high time defines the length of the preface time interval where the corresponding request source takes part in the arbitration. During the preface time, the currently running conversion can be finished. It has to be programmed to a value allowing the converter to become idle.
If signal ARBCNT is used as counting input signal for a timer, the arbiter has to be programmed to run permanently (GLOBCTR.ARBM = 0). If the timer has an independent time base, the arbiter can be stopped while no requests are pending. The preface time has to be longer than one arbitration round.
Depending on the request source requesting equidistant sampling, one or more channels can be converted one after the other. The order of the requested channels being fixed by the request source, the equidistant sampling is also supported for several channels. It is also possible to do equidistant sampling for more than one request source in parallel if the preface times and the equidistant conversions do not overlap.

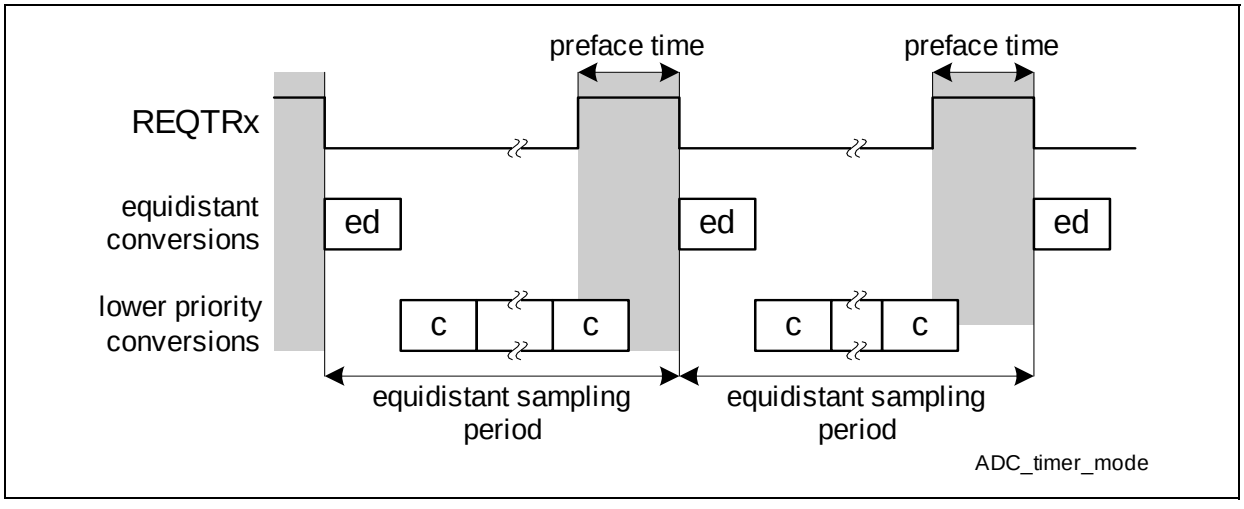


Figure 23-26 Timer Mode for Equidistant Sampling

23.2.21 Access Protection

An access protection scheme has been implemented to avoid unintended modification of some ADC control registers. It can be enabled by bits in register **APR** that itself is protected by the ENDINIT mechanism.
If the access protection is enabled for a group of registers and a write access occurs to one of them, the write access is discarded, the targeted register is not modified, the written data is ignored and the error flag ACCERR is set.
The protected ADC registers are located in one of the following register groups (registers not listed below can not be protected):

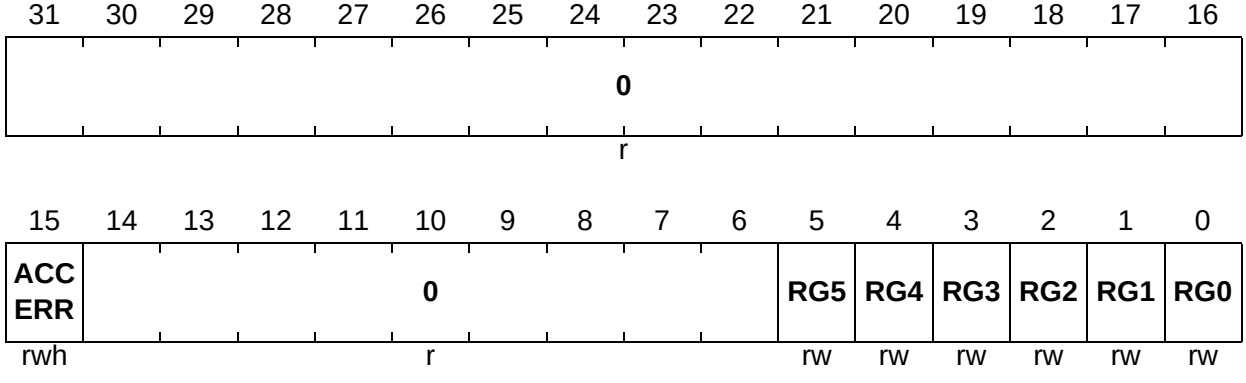
- Register group 0:
  GLOBCTR
- Register group 1:
  GLOBCFG, **EMCTR**, **RSIRx (x = 0 - 4)**, **ALR0**
- Register group 2:
  ASENR, **RSPR0**, **RSPR4**, **INPCRx (x = 0 - 3)**, **SYNCTR**
- Register group 3:
  CHCTRx (x = 0 - 15)
- Register group 4:
  RCRx (x = 0 - 15), **VFR**, **LCBR0**, **LCBR1**, **LCBR2**, **LCBR3**
- Register group 5:
  CHENPR0, **CHENPR8**, **EVNPR**, **RNPR0**, **RNPR8**, **INTR**, **CHFR**, **EVFR**

23.2.22 Additional Feature Registers

23.2.22.1 Access Protection Register

The access protection register APR contains bits to enable/disable modification of ADC control/configuration registers by write accesses. Register APR itself is protected by the ENDINIT mechanism.

APR
Access Protection Register **(218_H)** **Reset Value: $0000\ 0000_H$**

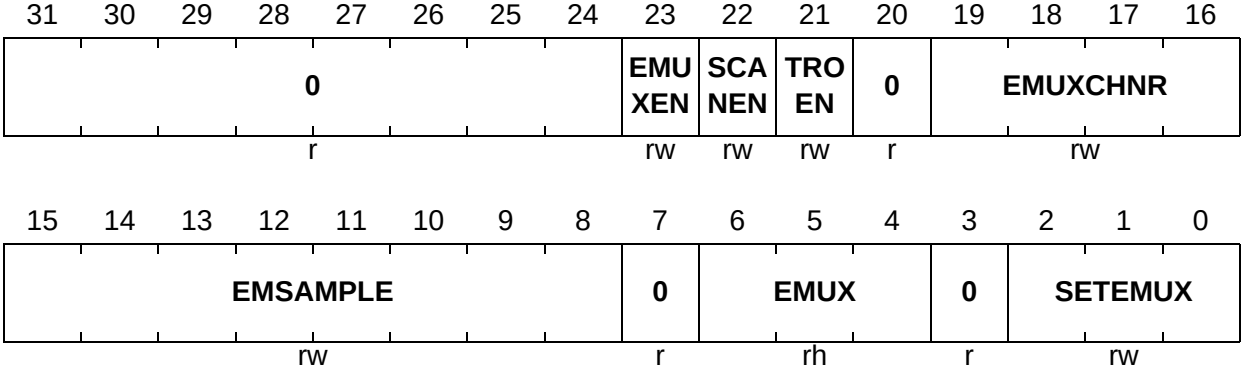


Field	Bits	Type	Description
RGx (x = 0 - 5)	x	rw	**Register Group x** This bit enables/disables write accesses to registers of register group x. 0_B Write actions to register group x are enabled and can modify the register contents. 1_B Write actions to register group x are disabled and do not modify the register contents.
ACCERR	15	rwh	**Access Error** This flag indicates a violation of the access protection mechanism for the ADC kernel. It can be cleared by writing 1 to this bit position. Writing 0 has no effect. 0_B A write access to a protected register group has not been detected. 1_B A write access to a protected register group has been detected and discarded.
0	[31:16], [14:6]	r	**Reserved** Read as 0; should be written with 0.

23.2.22.2 External Multiplexer Control

The external multiplexer control register defines the settings of the external analog multiplexer.

EMCTR
External Multiplexer Control Register (220_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
SETEMUX	[2:0]	rw	**Setting of External Multiplexer** If the external multiplexer control is disabled, EMUX is loaded with the SETEMUX value. If enabled, the following two options are available: Scan Mode disabled: This bit field defines the input of the external multiplexer that will be selected for the next conversion of the channel selected by EMUXCHNR. Bit field EMUX will be updated by SETEMUX at the beginning of the next conversion of this channel. Scan Mode enabled: This bit field defines the start value of the scan of the external multiplexer inputs. The scan starts with the programmed input down to input 0. Bit field EMUX is updated by SETEMUX at the end of the conversion of this channel if EMUX = 0.

Field	Bits	Type	Description
EMUX	[6:4]	rh	**Current Setting for External Multiplexer** This bit field defines the input of the external multiplexer selected for conversion. Its value is available at the output lines EMUX[2:0]. If the external multiplexer control is disabled, EMUX is loaded with the SETEMUX value. If enabled, the following two options are available: Scan Mode disabled: This bit field becomes updated by SETEMUX at the beginning of the conversion of the channel selected by EMUXCHNR. Scan Mode enabled: This bit field is decremented by 1 at the end of the conversion of the channel selected by EMUXCHNR. After reaching 0, it is reloaded with the value of bit field SETEMUX.
EMSAMPLE	[15:8]	rw	**External Multiplexer Sampling Time** This bit field defines the alternative sample phase length in the case the external multiplexer setting has changed with the start of a conversion with enabled external multiplexer (the value given by the selected input class is not taken into account). A minimum sample phase of 2 analog clock cycles is extended by the programmed value. sample phase length = (2 + EMSAMPLE) / f_{ADCI}
EMUXCHNR	[19:16]	rw	**Channel Number for External Multiplexer** If external multiplexer control is enabled (EMUXEN = 1), this bit field defines the analog ADC input channel connected to the external analog multiplexer.

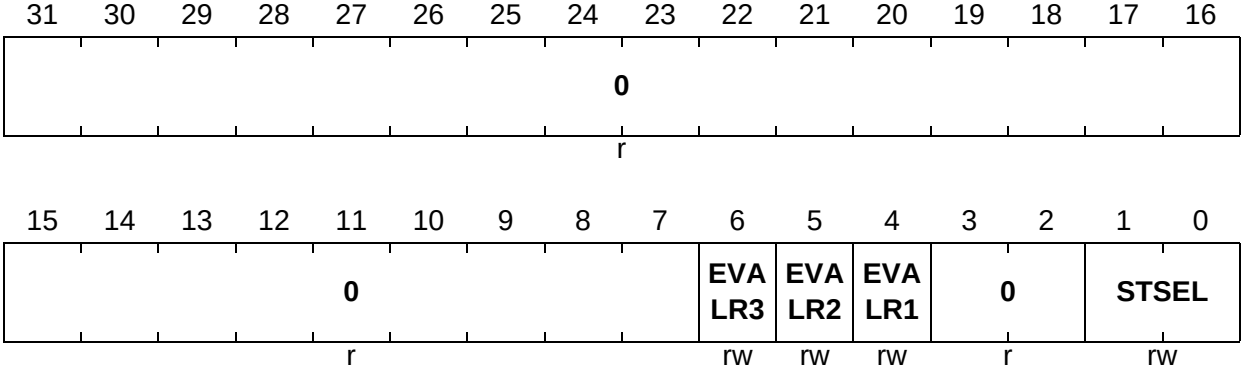
Field	Bits	Type	Description
TROEN	21	rw	**Trigger Option Enable** This bit selects the scan mode behavior of the external multiplexer (if enabled). Description see **Section 23.2.18**. 0_B Single-input scan is selected. The trigger option is disabled (no automatic trigger of more conversions of CHx). 1_B Multi-input scan is selected. The trigger option is enabled leading to an automatic scan through the externally connected multiplexer inputs by automatically triggering additional conversions of CHx until EMUX = 0.
SCANEN	22	rw	**Scan Enable** This bit enables/disables the automatic scan of the inputs of the external multiplexer for conversions of the channel selected by bit field EMUXCHNR (taken into account only if EMUXEN=1). 0_B The scan mode is disabled. Bit field EMUX is updated by bit field SETEMUX at the beginning of a conversion of the selected channel. If bit EMUX is changed, the value of EMSAMPLE is applied. 1_B The scan mode is enabled. Bit field EMUX is decremented by 1 for each conversion of the selected channel. After reaching 0, bit field EMUX is updated by bit field SETEMUX. The value of EMSAMPLE is always applied for the selected channel. It is recommended to write the start value of the first scan sequence to SETEMUX while EMUXEN=0.

Field	Bits	Type	Description
EMUXEN	23	rw	**External Multiplexer Control Enable** This bit enables/disables the automatic control of the external multiplexer. 0_B The external multiplexer control by HW is disabled. Bit field EMUX is immediately updated under SW control by writing to SETEMUX. The settings of SCANEN and TROEN are ignored. 1_B The external multiplexer control is enabled. The update of EMUX is under HW control respecting the conversion timings.
0	3, 7, 20, [31:24]	r	**Reserved** Read as 0; should be written with 0.

23.2.22.3 Synchronization Control Register

The synchronization control register contains bits controlling the synchronization between several kernels for parallel conversions. The programming of register SYNCTR in the kernels of a conversion group has to be done while the bit fields **GLOBCTR**.ANON = 00_B in all ADC kernels of the conversion group. Bit field **GLOBCTR**.ANON of the synchronization master can be set to 11_B afterwards.
The bits EVALRx are only taken into account if a synchronized, parallel conversion is requested by a master. This ensures that the conversions of the ADC kernels of the same synchronization group are started at the same time for parallel sampling (although a kernel might be idle, the master and all its connected slaves have to wait for all of them being ready).

SYNCTR
Synchronization Control Register (048_H) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
STSEL	[1:0]	rw	**Start Selection** This bit field controls the synchronization mechanism of the ADC kernel. 00_B The kernel is a synchronization master. The kernels own bit field GLOBCTR.ANON is taken into account. 01_B The kernel is a synchronization slave. The control information at input CI1 is taken into account instead. 10_B The kernel is a synchronization slave. The control information at input CI2 is taken into account instead. 11_B reserved, do not use (kernel is switched off)

Field	Bits	Type	Description
EVALR1	4	rw	**Evaluate Ready Input R1** This bit defines if a kernel is considered to be part of a synchronization group. Parallel conversions can only be started if the synchronization master and all slaves of the conversion group indicate that they are ready to start a parallel conversion. 0_B The ready input R1 is not considered for the start of a parallel conversion of this conversion group. 1_B The ready input R1 is considered for the start of a parallel conversion of this conversion group.
EVALR2	5	rw	**Evaluate Ready Input R2** This bit defines if a kernel is considered to be part of a synchronization group. Parallel conversions can only be started if the synchronization master and all slaves of the conversion group indicate that they are ready to start a parallel conversion. 0_B The ready input R2 is not considered for the start of a parallel conversion of this conversion group. 1_B The ready input R2 is considered for the start of a parallel conversion of this conversion group.
EVALR3	6	rw	**Evaluate Ready Input R3** This bit defines if a kernel is considered to be part of a synchronization group. Parallel conversions can only be started if the synchronization master and all slaves of the conversion group indicate that they are ready to start a parallel conversion. 0_B The ready input R3 is not considered for the start of a parallel conversion of this conversion group. 1_B The ready input R3 is considered for the start of a parallel conversion of this conversion group.
0	[3:2], [31:7]	r	**Reserved** Read as 0; should be written with 0.

23.3 Implementation

This chapter describes the implementation of the ADC kernels in the TC1797 device. It contains the following sections:

- Request sources (see **Section 23.3.1**)
- Address map (see **Section 23.3.2**)
- Connections to modules and pins (see **Section 23.3.3**)

23.3.1 Request Sources in TC1797

In each ADC kernel 5 request sources are implemented. They are numbered source 0 to source 4, with source x being evaluated in arbitration slot x. Each request source has the possibility to select one trigger input REQTRx out of a vector REQTRx_[7:0] and one gating input REQGTx out of a vector REQGTx_[7:0]. Each input vector contains 8 possible input signals, but not all of them are necessarily connected.

- Source 0: 1-stage sequential source
- Source 1: Parallel source for up to 16 channels
- Source 2: 4-stage sequential source
- Source 3: Parallel source for up to 16 channels
- Source 4: 4-stage sequential source

23.3.2 Address Map

The common KSCFG register of the ADC module can be accessed in the address range of kernel ADC0. The corresponding address in the range of kernel ADC1 is not used and delivers a dummy value when read.
The ADC kernels are available at the following base addresses:

Table 23-4 Registers Address Space

Module	Base Address	End Address	Note
ADC0	$F010\ 1000_H$	$F010\ 13FF_H$	
ADC1	$F010\ 1400_H$	$F010\ 17FF_H$	
ADC2	$F010\ 1800_H$	$F010\ 1BFF_H$	

Table 23-5 Registers Overview

Register Short Name	Register Long Name	Offset Address	Page Number
please refer to register table in **Section 23.2.1**		H	

23.3.3 ADC Module Connections

In addition to the standard signals of the interface between the analog and the digital parts, some side band signals have been introduced.
The ADC module consists of three ADC kernels. All kernels support the feature set listed above and share a common ADC0_KSCFG register. The kernels can be synchronized to each other for parallel sampling.

The channels CH4, CH5, CH6, and CH7 of ADC0 and the channels CH0 of all ADC kernels are always converted referring to the V_{AREF} input of the corresponding ADC kernel. For these channels, the programmed alternative reference selection is not taken into account. All other channels are converted with respect to the selected reference.

23.3.3.1 ADC0 Connections

Signals of the ADC module referring to the kernel of ADC0 are generally named with the prefix ADC0_.

The kernel ADC0 has its own reference inputs ADC0_V_{AGND} and ADC0_V_{AREF}. Depending on the package, these lines can be available as independent pins for high pin count packages or can be combined with the corresponding inputs of the other kernels for low pin count packages.
The respective voltage supply lines of all ADC analog parts are connected together.

Table 23-6 ADC0 Connections to Analog Part in TC1797

ADC0 Signal of Analog Part	from/to Module or Pin	Input or Output	Can be used to/as
V_{DDM}	V_{DDM}	I	analog power supply 3.3V - 5V
V_{DDA}	V_{DDMF}	I	analog power supply for comparator, connected to FADC 3.3V supply
V_{SSM}	V_{SSM}	I	analog power ground
V_{AREF}	V_{AREF}	I	positive analog reference
V_{AGND}	V_{AGND}	I	negative analog reference, combined with other kernels
CH0	AN0	I	analog input channel 0
CH1	AN1	I	analog input channel 1
CH2	AN2	I	analog input channel 2
CH3	AN3	I	analog input channel 3
CH4	AN4	I	analog input channel 4, overlaid with ADC2 channel 12
CH5	AN5	I	analog input channel 5, overlaid with ADC2 channel 13
CH6	AN6	I	analog input channel 6, overlaid with ADC2 channel 14
CH7	AN7	I	analog input channel 7, overlaid with ADC2 channel 15
CH8	AN8	I	analog input channel 8
CH9	AN9	I	analog input channel 9
CH10	AN10	I	analog input channel 10
CH11	AN11	I	analog input channel 11

Table 23-6 ADC0 Connections to Analog Part in TC1797 (cont'd)

ADC0 Signal of Analog Part	from/to Module or Pin	Input or Output	Can be used to/as
CH12	AN12	I	analog input channel 12
CH13	AN13	I	analog input channel 13
CH14	AN14	I	analog input channel 14
CH15	AN15	I	analog input channel 15

The following table shows the digital connections of the ADC0 kernel with other modules or pins in the TC1797 device. Signals of the ADC module referring to the kernel of ADC0 are named with the prefix ADC0_.

Table 23-7 ADC0 Connections of Digital Part in TC1797

ADC0 Signal of Digital Part	from/to Module or Pin	Input[1)] or Output	Can be used to/as
Kernel Signals			
ARBCNT	-	O	Counting signal for arbiter rounds
EMUXTR	-	O	Trigger output for scanning the external multiplexer inputs
EMUX0	see port chapter	O	control of external analog multiplexer(s)
EMUX1	see port chapter		
EMUX2	see port chapter		
Request Source 0			
REQGT0_0	TRIG10	I	GPTA
REQGT0_1	TRIG12	I	GPTA
REQGT0_2	TRIG14	I	GPTA
REQGT0_3	PDOUT2	I	ERU
REQGT0_4	REQ0	I (s)	P1.0
REQGT0_5	IRQ1	I (s)	STM (do not use for gating, but for triggering)
REQGT0_6	0	I (s)	not connected
REQGT0_7	0	I (s)	not connected
REQTR0_0	TRIG00	I	GPTA

Table 23-7 ADC0 Connections of Digital Part in TC1797 (cont'd)

ADC0 Signal of Digital Part	from/to Module or Pin	Input[1)] or Output	Can be used to/as
REQTR0_1	IOUT2	I	ERU
REQTR0_2	0	I	not connected
REQTR0_3	ADC_SR6	I	common service request output 6 of ADC module
REQTR0_4	REQ1	I (s)	P1.1
REQTR0_5	0	I (s)	not connected
REQTR0_6	ADC0_REQGT0	I (s)	extend input selection for triggering by using gating inputs (with ENGT = 0X)
REQTR0_7	ADC0_SR7	I (s)	service request output 7 of ADC0
REQTR0	-	O	selected trigger signal for source 0 (used as REQTRS for source 0)
REQGT0	ADC0_REQTR0_6	O	selected gating signal for source 0
Request Source 1			
REQGT1_0	TRIG10	I	GPTA
REQGT1_1	TRIG12	I	GPTA
REQGT1_2	TRIG14	I	GPTA
REQGT1_3	PDOUT3	I	ERU
REQGT1_4	REQ0	I (s)	P1.0
REQGT1_5	IRQ1	I (s)	STM (do not use for gating, but for triggering)
REQGT1_6	0	I (s)	not connected
REQGT1_7	0	I (s)	not connected
REQTR1_0	TRIG03	I	GPTA
REQTR1_1	IOUT3	I	ERU
REQTR1_2	TRIG16	I	GPTA
REQTR1_3	ADC_SR6	I	common service request output 6 of ADC module
REQTR1_4	REQ1	I (s)	P1.1
REQTR1_5	0	I (s)	not connected

Table 23-7 ADC0 Connections of Digital Part in TC1797 (cont'd)

ADC0 Signal of Digital Part	from/to Module or Pin	Input[1)] or Output	Can be used to/as
REQTR1_6	ADC0_REQGT1	I (s)	extend input selection for triggering by using gating inputs (with ENGT = 0X)
REQTR1_7	ADC0_SR7	I (s)	service request output 7 of ADC0
REQTR1	-	O	selected trigger signal for source 1 (used as REQTRS for source 1)
REQGT1	ADC0_REQTR1_6	O	selected gating signal for source 1
Request Source 2			
REQGT2_0	TRIG10	I	GPTA
REQGT2_1	TRIG12	I	GPTA
REQGT2_2	TRIG14	I	GPTA
REQGT2_3	PDOUT3	I	ERU
REQGT2_4	REQ4	I (s)	P7.0
REQGT2_5	IRQ1	I (s)	STM (do not use for gating, but for triggering)
REQGT2_6	0	I (s)	not connected
REQGT2_7	0	I (s)	not connected
REQTR2_0	TRIG04	I	GPTA
REQTR2_1	IOUT3	I	ERU
REQTR2_2	TRIG16	I	GPTA
REQTR2_3	ADC_SR6	I	common service request output 6 of ADC module
REQTR2_4	REQ5	I (s)	P7.1
REQTR2_5	0	I (s)	not connected
REQTR2_6	ADC0_REQGT2	I (s)	extend input selection for triggering by using gating inputs (with ENGT = 0X)
REQTR2_7	ADC0_SR7	I (s)	service request output 7 of ADC0
REQTR2	-	O	selected trigger signal for source 2 (used as REQTRS for source 2)
REQGT2	ADC0_REQTR2_6	O	selected gating signal for source 2
Request Source 3			

Table 23-7 ADC0 Connections of Digital Part in TC1797 (cont'd)

ADC0 Signal of Digital Part	from/to Module or Pin	Input[1)] or Output	Can be used to/as
REQGT3_0	TRIG10	I	GPTA
REQGT3_1	TRIG12	I	GPTA
REQGT3_2	TRIG14	I	GPTA
REQGT3_3	PDOUT2	I	ERU
REQGT3_4	REQ4	I (s)	P7.0
REQGT3_5	IRQ1	I (s)	STM (do not use for gating, but for triggering)
REQGT3_6	0	I (s)	not connected
REQGT3_7	0	I (s)	not connected
REQTR3_0	TRIG07	I	GPTA
REQTR3_1	IOUT2	I	ERU
REQTR3_2	TRIG16	I	GPTA
REQTR3_3	ADC_SR6	I	common service request output 6 of ADC module
REQTR3_4	REQ5	I (s)	P7.1
REQTR3_5	0	I (s)	not connected
REQTR3_6	ADC0_REQGT3	I (s)	extend input selection for triggering by using gating inputs (with ENGT = 0X)
REQTR3_7	ADC0_SR7	I (s)	service request output 7 of ADC0
REQTR3	-	O	selected trigger signal for source 3 (used as REQTRS for source 3)
REQGT3	ADC0_REQTR3_6	O	selected gating signal for source 3
Request Source 4			
REQGT4_0	TRIG10	I	GPTA
REQGT4_1	TRIG12	I	GPTA
REQGT4_2	TRIG14	I	GPTA
REQGT4_3	PDOUT2	I	ERU
REQGT4_4	REQ0	I (s)	P1.0
REQGT4_5	IRQ1	I (s)	STM (do not use for gating, but for triggering)

Table 23-7 ADC0 Connections of Digital Part in TC1797 (cont'd)

ADC0 Signal of Digital Part	from/to Module or Pin	Input[1] or Output	Can be used to/as
REQGT4_6	0	I (s)	not connected
REQGT4_7	0	I (s)	not connected
REQTR4_0	TRIG11	I	GPTA
REQTR4_1	IOUT2	I	ERU
REQTR4_2	0	I	not connected
REQTR4_3	ADC_SR6	I	common service request output 6 of ADC module
REQTR4_4	REQ6	I (s)	P7.4
REQTR4_5	0	I (s)	not connected
REQTR4_6	ADC0_REQGT4	I (s)	extend input selection for triggering by using gating inputs (with ENGT = 0X)
REQTR4_7	ADC0_SR7	I (s)	service request output 7 of ADC0
REQTR4	-	O	selected trigger signal for source 4 (used as REQTRS for source 4)
REQGT4	ADC0_REQTR4_6	O	selected gating signal for source 4

1) In case of input signals, the lines marked "I" don't contain synchronization stages, whereas the lines marked "I (s)" contain synchronization stages (so they can directly handle asynchronous input signals). A signal connected to an input line marked "I" has to be delivered from a block located in the same clock domain as the ADC (the signal has to be synchronous to the ADC clock domain).

23.3.3.2 ADC1 Connections

Signals of the ADC module referring to the kernel of ADC1 are named with the prefix ADC1_.

The kernel ADC1 has its own reference inputs ADC1_V_{AGND} and ADC1_V_{AREF}. Depending on the package, these lines can be available as independent pins for high pin count packages or can be combined with the corresponding inputs of the other kernels for low pin count packages.
The respective voltage supply lines of the ADC analog parts are connected together.

Table 23-8 ADC1 Connections of Analog Part in TC1797

ADC1 Signal of Analog Part	from/to Module or Pin	Input or Output	Can be used to/as
V_{DDM}	V_{DDM}	I	analog power supply 3.3V - 5V
V_{DDA}	V_{DDMF}	I	analog power supply for comparator, taken from FADC 3.3V supply
V_{SSM}	V_{SSM}	I	analog power ground
V_{AREF}	V_{AREF}	I	positive analog reference
V_{AGND}	V_{AGND}	I	negative analog reference, combined with other kernels
CH0	AN16	I	analog input channel 16
CH1	AN17	I	analog input channel 17
CH2	AN18	I	analog input channel 18
CH3	AN19	I	analog input channel 19
CH4	AN20	I	analog input channel 20
CH5	AN21	I	analog input channel 21
CH6	AN22	I	analog input channel 22
CH7	AN23	I	analog input channel 23
CH8	AN24	I	analog input channel 24, overlaid with FADC_FAIN0P
CH9	AN25	I	analog input channel 25, overlaid with FADC_FAIN0N
CH10	AN26	I	analog input channel 26, overlaid with FADC_FAIN1P
CH11	AN27	I	analog input channel 27, overlaid with FADC_FAIN1N

Table 23-8 ADC1 Connections of Analog Part in TC1797 (cont'd)

ADC1 Signal of Analog Part	from/to Module or Pin	Input or Output	Can be used to/as
CH12	AN28	I	analog input channel 28, overlaid with FADC_FAIN2P
CH13	AN29	I	analog input channel 29, overlaid with FADC_FAIN2N
CH14	AN30	I	analog input channel 30, overlaid with FADC_FAIN3P
CH15	AN31	I	analog input channel 31, overlaid with FADC_FAIN3N

The following table shows the digital connections of the ADC1 kernel with other modules or pins in the TC1797 device. Output signals of the ADC module referring to the kernel of ADC1 are named with the prefix ADC1_.

Table 23-9 ADC1 Connections of Digital Part in TC1797

ADC1 Signal of Digital Part	from/to Module or Pin	Input [1] or Output	Can be used to/as
Kernel Signals			
ARBCNT	-	O	Counting signal for arbiter rounds
EMUXTR	-	O	Trigger output for scanning the external multiplexer inputs
EMUX0	see port chapter	O	control of external analog multiplexer(s)
EMUX1	see port chapter		
EMUX2	non-connected		
Request Source 0			
REQGT0_0	TRIG10	I	GPTA
REQGT0_1	TRIG12	I	GPTA
REQGT0_2	TRIG14	I	GPTA
REQGT0_3	PDOUT2	I	ERU
REQGT0_4	REQ4	I (s)	P7.0
REQGT0_5	IRQ1	I (s)	STM (do not use for gating, but for triggering)

Table 23-9 ADC1 Connections of Digital Part in TC1797 (cont'd)

ADC1 Signal of Digital Part	from/to Module or Pin	Input [1] or Output	Can be used to/as
REQGT0_6	ADC0_SR7	I (s)	ADC1 trigger by ADC0
REQGT0_7	0	I (s)	not connected
REQTR0_0	TRIG00	I	GPTA
REQTR0_1	IOUT2	I	ERU
REQTR0_2	0	I	not connected
REQTR0_3	ADC_SR6	I	common service request output 6 of ADC module
REQTR0_4	REQ5	I (s)	P7.1
REQTR0_5	0	I (s)	not connected
REQTR0_6	ADC1_REQGT0	I (s)	extend input selection for triggering by using gating inputs (with ENGT = 0X)
REQTR0_7	ADC1_SR7	I (s)	service request output 7 of ADC1
REQTR0	-	O	selected trigger signal for source 0 (used as REQTRS for source 0)
REQGT0	ADC1_REQTR0_6	O	selected gating signal for source 0
Request Source 1			
REQGT1_0	TRIG10	I	GPTA
REQGT1_1	TRIG12	I	GPTA
REQGT1_2	TRIG14	I	GPTA
REQGT1_3	PDOUT3	I	ERU
REQGT1_4	REQ4	I (s)	P7.0
REQGT1_5	IRQ1	I (s)	STM (do not use for gating, but for triggering)
REQGT1_6	ADC0_SR7	I (s)	ADC1 trigger by ADC0
REQGT1_7	0	I (s)	not connected
REQTR1_0	TRIG05	I	GPTA
REQTR1_1	IOUT3	I	ERU
REQTR1_2	TRIG16	I	GPTA
REQTR1_3	ADC_SR6	I	common service request output 6 of ADC module

Table 23-9 ADC1 Connections of Digital Part in TC1797 (cont'd)

ADC1 Signal of Digital Part	from/to Module or Pin	Input [1] or Output	Can be used to/as
REQTR1_4	REQ5	I (s)	P7.1
REQTR1_5	0	I (s)	not connected
REQTR1_6	ADC1_REQGT1	I (s)	extend input selection for triggering by using gating inputs (with ENGT = 0X)
REQTR1_7	ADC1_SR7	I (s)	service request output 7 of ADC1
REQTR1	-	O	selected trigger signal for source 1 (used as REQTRS for source 1)
REQGT1	ADC1_REQTR1_6	O	selected gating signal for source 1
Request Source 2			
REQGT2_0	TRIG10	I	GPTA
REQGT2_1	TRIG12	I	GPTA
REQGT2_2	TRIG14	I	GPTA
REQGT2_3	PDOUT3	I	ERU
REQGT2_4	REQ0	I (s)	P1.0
REQGT2_5	IRQ1	I (s)	STM (do not use for gating, but for triggering)
REQGT2_6	ADC0_SR7	I (s)	ADC1 trigger by ADC0
REQGT2_7	0	I (s)	not connected
REQTR2_0	TRIG06	I	GPTA
REQTR2_1	IOUT3	I	ERU
REQTR2_2	TRIG16	I	GPTA
REQTR2_3	ADC_SR6	I	common service request output 6 of ADC module
REQTR2_4	REQ1	I (s)	P1.1
REQTR2_5	0	I (s)	not connected
REQTR2_6	ADC1_REQGT2	I (s)	extend input selection for triggering by using gating inputs (with ENGT = 0X)
REQTR2_7	ADC1_SR7	I (s)	service request output 7 of ADC1
REQTR2	-	O	selected trigger signal for source 2 (used as REQTRS for source 2)

Table 23-9 ADC1 Connections of Digital Part in TC1797 (cont'd)

ADC1 Signal of Digital Part	from/to Module or Pin	Input [1] or Output	Can be used to/as
REQGT2	ADC1_REQTR2_6	O	selected gating signal for source 2
Request Source 3			
REQGT3_0	TRIG10	I	GPTA
REQGT3_1	TRIG12	I	GPTA
REQGT3_2	TRIG14	I	GPTA
REQGT3_3	PDOUT2	I	ERU
REQGT3_4	REQ0	I (s)	P1.0
REQGT3_5	IRQ1	I (s)	STM (do not use for gating, but for triggering)
REQGT3_6	ADC0_SR7	I (s)	ADC1 trigger by ADC0
REQGT3_7	0	I (s)	not connected
REQTR3_0	TRIG01	I	GPTA
REQTR3_1	IOUT2	I	ERU
REQTR3_2	TRIG16	I	GPTA
REQTR3_3	ADC_SR6	I	common service request output 6 of ADC module
REQTR3_4	REQ1	I (s)	P1.1
REQTR3_5	0	I (s)	not connected
REQTR3_6	ADC1_REQGT3	I (s)	extend input selection for triggering by using gating inputs (with ENGT = 0X)
REQTR3_7	ADC1_SR7	I (s)	service request output 7 of ADC1
REQTR3	-	O	selected trigger signal for source 3 (used as REQTRS for source 3)
REQGT3	ADC1_REQTR3_6	O	selected gating signal for source 3
Request Source 4			
REQGT4_0	TRIG10	I	GPTA
REQGT4_1	TRIG12	I	GPTA
REQGT4_2	TRIG14	I	GPTA
REQGT4_3	PDOUT3	I	ERU
REQGT4_4	REQ0	I (s)	P1.0

Table 23-9 ADC1 Connections of Digital Part in TC1797 (cont'd)

ADC1 Signal of Digital Part	from/to Module or Pin	Input [1] or Output	Can be used to/as
REQGT4_5	IRQ1	I (s)	STM (do not use for gating, but for triggering)
REQGT4_6	ADC0_SR7	I (s)	ADC1 trigger by ADC0
REQGT4_7	0	I (s)	not connected
REQTR4_0	TRIG13	I	GPTA
REQTR4_1	IOUT3	I	ERU
REQTR4_2	0	I	not connected
REQTR4_3	ADC_SR6	I	common service request output 6 of ADC module
REQTR4_4	REQ7	I (s)	P7.5
REQTR4_5	0	I (s)	not connected
REQTR4_6	ADC1_REQGT4	I (s)	extend input selection for triggering by using gating inputs (with ENGT = 0X)
REQTR4_7	ADC1_SR7	I (s)	service request output 7 of ADC1
REQTR4	-	O	selected trigger signal for source 4 (used as REQTRS for source 4)
REQGT4	ADC1_REQTR4_6	O	selected gating signal for source 4

1) In case of input signals, the lines marked "I" don't contain synchronization stages, whereas the lines marked "I (s)" contain synchronization stages (so they can directly handle asynchronous input signals). A signal connected to an input line marked "I" has to be delivered from a block located in the same clock domain as the ADC (the signal has to be synchronous to the ADC clock domain).

23.3.3.3 ADC2 Connections

Signals of the ADC module referring to the kernel of ADC2 are generally named with the prefix ADC2_.

The kernel ADC2 has its own reference inputs ADC2_V_{AGND} and ADC2_V_{AREF}. Depending on the package, these lines can be available as independent pins for high pin count packages or can be combined with the corresponding inputs of the other kernels for low pin count packages.
The respective voltage supply lines of all ADC analog parts are connected together.

Table 23-10 ADC2 Connections to Analog Part in TC1797

ADC2 Signal of Analog Part	from/to Module or Pin	Input or Output	Can be used to/as
V_{DDM}	V_{DDM}	I	analog power supply 3.3V - 5V
V_{DDA}	V_{DDMF}	I	analog power supply for comparator, connected to FADC 3.3V supply
V_{SSM}	V_{SSM}	I	analog power ground
V_{AREF}	V_{AREF}	I	positive analog reference, independent from other kernels
V_{AGND}	V_{AGND}	I	negative analog reference, combined with other kernels
CH0	AN32	I	analog input channel 0
CH1	AN33	I	analog input channel 1
CH2	AN34	I	analog input channel 2
CH3	AN35	I	analog input channel 3
CH4	AN36	I	analog input channel 4
CH5	AN37	I	analog input channel 5
CH6	AN38	I	analog input channel 6
CH7	AN39	I	analog input channel 7
CH8	AN40	I	analog input channel 8
CH9	AN41	I	analog input channel 9
CH10	AN42	I	analog input channel 10
CH11	AN43	I	analog input channel 11
CH12	AN4	I	analog input channel 12, overlaid with ADC0 channel 4

Table 23-10 ADC2 Connections to Analog Part in TC1797 (cont'd)

ADC2 Signal of Analog Part	from/to Module or Pin	Input or Output	Can be used to/as
CH13	AN5	I	analog input channel 13, overlaid with ADC0 channel 5
CH14	AN6	I	analog input channel 14, overlaid with ADC0 channel 6
CH15	AN7	I	analog input channel 15, overlaid with ADC0 channel 7

The following table shows the digital connections of the ADC2 kernel with other modules or pins in the TC1797 device. Signals of the ADC module referring to the kernel of ADC2 are named with the prefix ADC2_.

Table 23-11 ADC2 Connections of Digital Part in TC1797

ADC2 Signal of Digital Part	from/to Module or Pin	Input[1)] or Output	Can be used to/as
Kernel Signals			
ARBCNT	-	O	Counting signal for arbiter rounds
EMUXTR	-	O	Trigger output for scanning the external multiplexer inputs
EMUX0	see port chapter	O	control of external analog multiplexer(s)
EMUX1	see port chapter		
EMUX2	see port chapter		
Request Source 0			
REQGT0_0	TRIG10	I	GPTA
REQGT0_1	TRIG12	I	GPTA
REQGT0_2	TRIG14	I	GPTA
REQGT0_3	PDOUT2	I	ERU
REQGT0_4	REQ6	I (s)	P7.4
REQGT0_5	IRQ1	I (s)	STM (do not use for gating, but for triggering)
REQGT0_6	ADC1_SR7	I (s)	ADC2 trigger by ADC1
REQGT0_7	0	I (s)	not connected

Table 23-11 ADC2 Connections of Digital Part in TC1797 (cont'd)

ADC2 Signal of Digital Part	from/to Module or Pin	Input[1)] or Output	Can be used to/as
REQTR0_0	TRIG00	I	GPTA
REQTR0_1	IOUT2	I	ERU
REQTR0_2	0	I	not connected
REQTR0_3	ADC_SR6	I	common service request output 6 of ADC module
REQTR0_4	REQ7	I (s)	P7.5
REQTR0_5	0	I (s)	not connected
REQTR0_6	ADC2_REQGT0	I (s)	extend input selection for triggering by using gating inputs (with ENGT = 0X)
REQTR0_7	ADC2_SR7	I (s)	service request output 7 of ADC2
REQTR0	-	O	selected trigger signal for source 0 (used as REQTRS for source 0)
REQGT0	ADC2_REQTR0_6	O	selected gating signal for source 0
Request Source 1			
REQGT1_0	TRIG10	I	GPTA
REQGT1_1	TRIG12	I	GPTA
REQGT1_2	TRIG14	I	GPTA
REQGT1_3	PDOUT3	I	ERU
REQGT1_4	REQ7	I (s)	P7.5
REQGT1_5	IRQ1	I (s)	STM (do not use for gating, but for triggering)
REQGT1_6	ADC1_SR7	I (s)	ADC2 trigger by ADC1
REQGT1_7	0	I (s)	not connected
REQTR1_0	TRIG14	I	GPTA
REQTR1_1	IOUT3	I	ERU
REQTR1_2	TRIG16	I	GPTA
REQTR1_3	ADC_SR6	I	common service request output 6 of ADC module
REQTR1_4	REQ6	I (s)	P7.4
REQTR1_5	0	I (s)	not connected

Table 23-11 ADC2 Connections of Digital Part in TC1797 (cont'd)

ADC2 Signal of Digital Part	from/to Module or Pin	Input[1)] or Output	Can be used to/as
REQTR1_6	ADC2_REQGT1	I (s)	extend input selection for triggering by using gating inputs (with ENGT = 0X)
REQTR1_7	ADC2_SR7	I (s)	service request output 7 of ADC2
REQTR1	-	O	selected trigger signal for source 1 (used as REQTRS for source 1)
REQGT1	ADC2_REQTR1_6	O	selected gating signal for source 1
Request Source 2			
REQGT2_0	TRIG10	I	GPTA
REQGT2_1	TRIG12	I	GPTA
REQGT2_2	TRIG14	I	GPTA
REQGT2_3	PDOUT3	I	ERU
REQGT2_4	REQ7	I (s)	P7.5
REQGT2_5	IRQ1	I (s)	STM (do not use for gating, but for triggering)
REQGT2_6	ADC1_SR7	I (s)	ADC2 trigger by ADC1
REQGT2_7	0	I (s)	not connected
REQTR2_0	TRIG14	I	GPTA
REQTR2_1	IOUT3	I	ERU
REQTR2_2	TRIG16	I	GPTA
REQTR2_3	ADC_SR6	I	common service request output 6 of ADC module
REQTR2_4	REQ6	I (s)	P7.4
REQTR2_5	0	I (s)	not connected
REQTR2_6	ADC2_REQGT2	I (s)	extend input selection for triggering by using gating inputs (with ENGT = 0X)
REQTR2_7	ADC2_SR7	I (s)	service request output 7 of ADC2
REQTR2	-	O	selected trigger signal for source 2
REQGT2	ADC2_REQTR2_6	O	selected gating signal for source 2
Request Source 3			
REQGT3_0	TRIG10	I	GPTA

Table 23-11 ADC2 Connections of Digital Part in TC1797 (cont'd)

ADC2 Signal of Digital Part	from/to Module or Pin	Input[1)] or Output	Can be used to/as
REQGT3_1	TRIG12	I	GPTA
REQGT3_2	TRIG14	I	GPTA
REQGT3_3	PDOUT2	I	ERU
REQGT3_4	REQ6	I (s)	P7.4
REQGT3_5	IRQ1	I (s)	STM (do not use for gating, but for triggering)
REQGT3_6	ADC1_SR7	I (s)	ADC2 trigger by ADC1
REQGT3_7	0	I (s)	not connected
REQTR3_0	TRIG15	I	GPTA
REQTR3_1	IOUT2	I	ERU
REQTR3_2	TRIG16	I	GPTA
REQTR3_3	ADC_SR6	I	common service request output 6 of ADC module
REQTR3_4	REQ7	I (s)	P7.5
REQTR3_5	0	I (s)	not connected
REQTR3_6	ADC2_REQGT3	I (s)	extend input selection for triggering by using gating inputs (with ENGT = 0X)
REQTR3_7	ADC2_SR7	I (s)	service request output 7 of ADC2
REQTR3	-	O	selected trigger signal for source 3 (used as REQTRS for source 3)
REQGT3	ADC2_REQTR3_6	O	selected gating signal for source 3
Request Source 4			
REQGT4_0	TRIG10	I	GPTA
REQGT4_1	TRIG12	I	GPTA
REQGT4_2	TRIG14	I	GPTA
REQGT4_3	PDOUT2	I	ERU
REQGT4_4	REQ6	I (s)	P7.4
REQGT4_5	IRQ1	I (s)	STM (do not use for gating, but for triggering)
REQGT4_6	ADC1_SR7	I (s)	ADC2 trigger by ADC1

Table 23-11 ADC2 Connections of Digital Part in TC1797 (cont'd)

ADC2 Signal of Digital Part	from/to Module or Pin	Input[1] or Output	Can be used to/as
REQGT4_7	0	I (s)	not connected
REQTR4_0	TRIG15	I	GPTA
REQTR4_1	IOUT2	I	ERU
REQTR4_2	0	I	not connected
REQTR4_3	ADC_SR6	I	common service request output 6 of ADC module
REQTR4_4	REQ7	I (s)	P7.5
REQTR4_5	0	I (s)	not connected
REQTR4_6	ADC2_REQGT4	I (s)	extend input selection for triggering by using gating inputs (with ENGT = 0X)
REQTR4_7	ADC2_SR7	I (s)	service request output 7 of ADC2
REQTR4	-	O	selected trigger signal for source 4 (used as REQTRS for source 4)
REQGT4	ADC2_REQTR4_6	O	selected gating signal for source 4

1) In case of input signals, the lines marked "I" don't contain synchronization stages, whereas the lines marked "I (s)" contain synchronization stages (so they can directly handle asynchronous input signals). A signal connected to an input line marked "I"has to be delivered from a block located in the same clock domain as the ADC (the signal has to be synchronous to the ADC clock domain).

23.3.3.4 Service Request Connections

Each ADC kernel provides the service request output lines ADCy_SR[7:0]. The ADC module's service request outputs ADC_SR[11:0] are generated based on these lines according to the following table. If a line ADCy_SRx can be activated by more than one ADC kernel, the corresponding request lines of the ADC kernels are logical OR-combinations of the kernel outputs.
The wiring of the ADC_SRx signals to the DMA channels is given in the DMA chapter.
The signal ADCy_SR7 of each ADC kernel is available as input for the request sources for this kernel. Additionally, ADC0_SR7 is connected to ADC1 inputs and ADC1_SR7 is connected to ADC2 inputs.
The common signal ADC_SR6 is available as input for the request sources of all kernels.

Table 23-12 ADC Module Service Request Generation

Module Service Request Output	from ADC0 Kernel	from ADC1 Kernel	from ADC2 Kernel	Service Request Node
ADC_SR0	ADC0_SR0	ADC1_SR0	-	ADC0_SRC0
ADC_SR1	ADC0_SR1	ADC1_SR1	-	ADC0_SRC1
ADC_SR2	ADC0_SR2	ADC1_SR2	-	ADC0_SRC2
ADC_SR3	ADC0_SR3	ADC1_SR3	-	ADC0_SRC3
ADC_SR4	ADC0_SR4	ADC1_SR4	ADC2_SR4	ADC0_SRC4
ADC_SR5	ADC0_SR5	ADC1_SR5	ADC2_SR5	ADC0_SRC5
ADC_SR6	ADC0_SR6	ADC1_SR6	ADC2_SR6	-
ADC_SR7	ADC0_SR7	ADC1_SR7	ADC2_SR7	-
ADC_SR8	-	-	ADC2_SR0	ADC0_SRC6
ADC_SR9	-	-	ADC2_SR1	ADC0_SRC7
ADC_SR10	-	-	ADC2_SR2	ADC0_SRC8
ADC_SR11	-	-	ADC2_SR3	-

23.3.3.5 Kernel Synchronization

The independent ADC kernels of the ADC module can be synchronized to each other by selecting the corresponding connections. The table below shows the setting of the bits in the synchronization control registers in the ADC to allow synchronization. A kernel can operate completely autonomously if it is configured as master ADC. A slave ADC can be synchronized by the selected master ADC.

Note: A master ADC can synchronize several slave ADCs, whereas a slave ADC can only be synchronized by one master ADC.

Table 23-13 SYNCTR Setting for Kernel Synchronization

Operating Mode	SYNCTR. EVALR3	SYNCTR. EVALR2	SYNCTR. EVALR1	SYNCTR. STSEL
ADC0 Kernel (values to be programmed to ADC0_SYNCTR)				
no sync	0	0	0	00_B
master of only ADC1	0	0	1	00_B
master of only ADC2	0	1	0	00_B
master of ADC1 and ADC2	0	1	1	00_B
single slave of ADC1	0	0	1	01_B
ADC0 and ADC2 are slaves of ADC1	0	1	1	01_B
single slave of ADC2	0	1	0	10_B
ADC0 and ADC1 are slaves of ADC2	0	1	1	10_B
ADC1 Kernel (values to be programmed to ADC1_SYNCTR)				
no sync	0	0	0	00_B
master of only ADC0	0	0	1	00_B

Table 23-13 SYNCTR Setting for Kernel Synchronization (cont'd)

Operating Mode	SYNCTR. EVALR3	SYNCTR. EVALR2	SYNCTR. EVALR1	SYNCTR. STSEL
master of only ADC2	0	1	0	00_B
master of ADC0 and ADC2	0	1	1	00_B
single slave of ADC0	0	0	1	01_B
ADC1 and ADC2 are slaves of ADC0	0	1	1	01_B
single slave of ADC2	0	1	0	10_B
ADC0 and ADC1 are slaves of ADC2	0	1	1	10_B
ADC2 Kernel (values to be programmed to ADC2_SYNCTR)				
no sync	0	0	0	00_B
master of only ADC0	0	0	1	00_B
master of only ADC1	0	1	0	00_B
master of ADC0 and ADC1	0	1	1	00_B
single slave of ADC0	0	0	1	01_B
ADC1 and ADC2 are slaves of ADC0	0	1	1	01_B
single slave of ADC1	0	1	0	10_B
ADC0 and ADC2 are slaves of ADC1	0	1	1	10_B

24 Fast Analog to Digital Converter (FADC)

The TC1797 contains a fast analog to digital converter (FADC) with differential input channels, thus allowing sampling of high frequency signals. For slow and mid-range frequency signals, an anti-aliasing filter by data reduction is implemented to avoid expensive external filters.

- Functional description of the FADC Kernel (see **Section 24.2**)
- FADC kernel register descriptions (see **Section 24.3**)
- TC1797 implementation-specific details and registers of the FADC module, including on-chip interconnections, service request control, address decoding, and clock control (see **Section 24.4**)

*Note: The FADC Kernel register names described in **Section 24.3** are referenced in the TC1797 User´s Manual by the module name prefix “FADC_” for the FADC interface.*

24.1 FADC Short Description

General Features

- Extreme fast conversion, 21 cycles of f_{FADC} clock (262.5 ns @ f_{FADC} = 80 MHz)
- 10-bit A/D conversion (higher resolution can be achieved by averaging of consecutive conversions in digital data reduction filter)
- Successive approximation conversion method
- All differential input channels with impedance control
- All FADC inputs are overlaid with ADC1 inputs
- Each differential input channel can also be used as single-ended input
- Offset calibration support for each channel
- Programmable gain of 1, 2, 4, or 8 for each channel
- Free-running (Channel Timers) or triggered conversion modes
- Trigger and gating control for external signals
- Built-in Channel Timers for internal triggering
- Channel timer request periods independently selectable for each channel
- Selectable, programmable digital anti-aliasing and data reduction filter block with four independent filter units

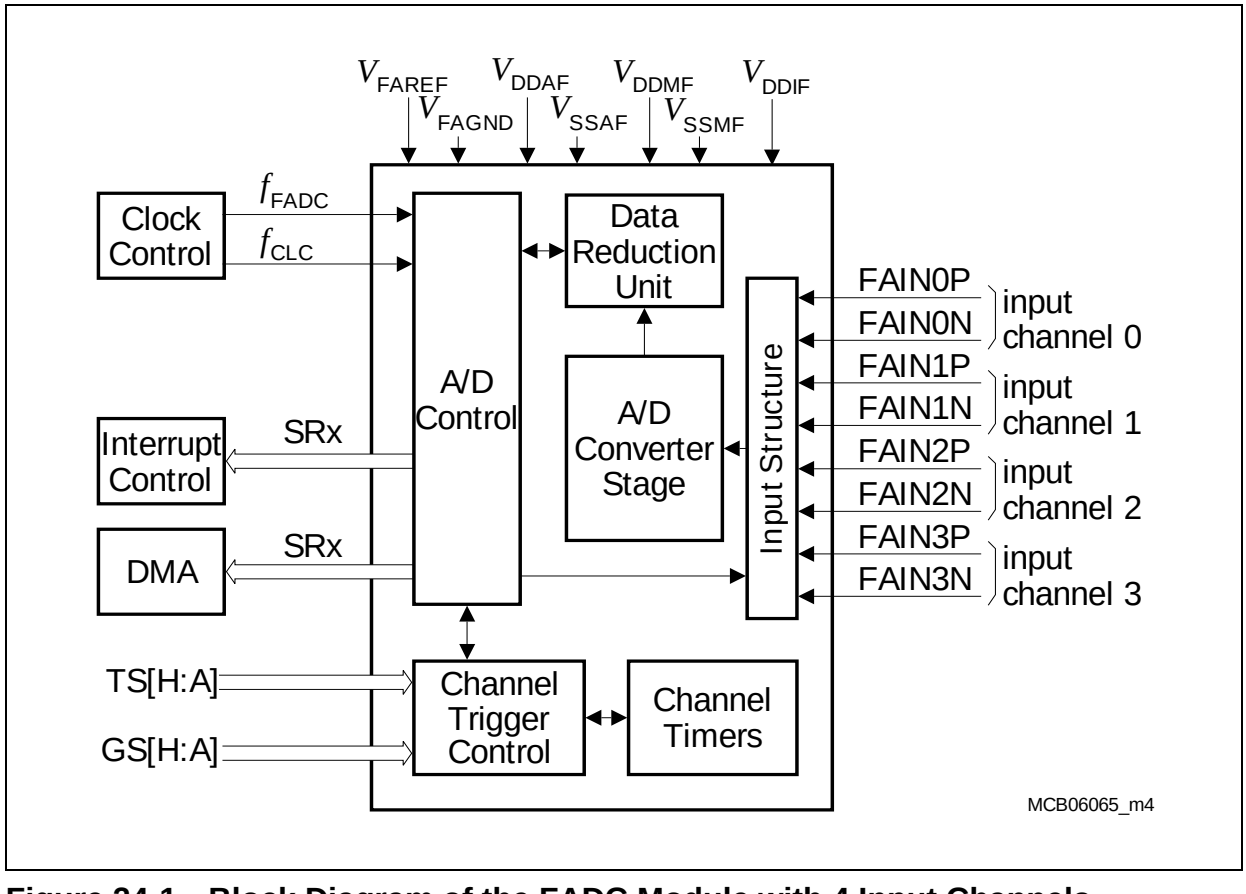


Figure 24-1 Block Diagram of the FADC Module with 4 Input Channels

As shown in **Figure 24-1**, the main FADC functional blocks are:

- An Input Structure containing the differential inputs and impedance control.
- An A/D Converter Stage responsible for the analog-to-digital conversion including an input multiplexer to select between the channel amplifiers
- A Data Reduction Unit containing programmable anti-aliasing and data reduction filters
- A Channel Trigger Control block determining the trigger and gating conditions for the FADC channels
- A Channel Timer for each channel to independently trigger the conversions
- An A/D Control block responsible for the overall FADC functionality

The analog block of the FADC in the TC1797 contains:

- A differential analog input stage for each input channel to select the input impedance (differential or single-ended measurement) and to decouple the FADC input signal from the pins. It is supplied by V_{DDIF} / V_{SSMF} (3.3 V - 5 V).
  The V_{DDIF} supply does not appear as supply pin in the pin list, because it is internally connected to the V_{DDM} supply of the ADC that is sharing the FADC input pins.
- A channel amplifier with a settling time of about 5µs if its configuration is changed by SW (changing between unused, differential, single-ended N, or single-ended P mode). It is supplied by V_{DDMF} / V_{SSMF} (3.3 V).
- All input channels are overlaid with ADC1 input signals (AN24 - AN31).
  Input levels on enabled FADC inputs must not exceed V_{DDMF}, whereas those of disabled FADC inputs must not exceed V_{DDIF}.
- A 10-bit analog converter stage supplied by V_{DDAF} / V_{SSAF} (1.5 V) to be delivered externally. The inputs for the FADC reference voltage (3.3 V max.) and the FADC reference ground V_{FAREF} / V_{FAGND} are connected to pins.

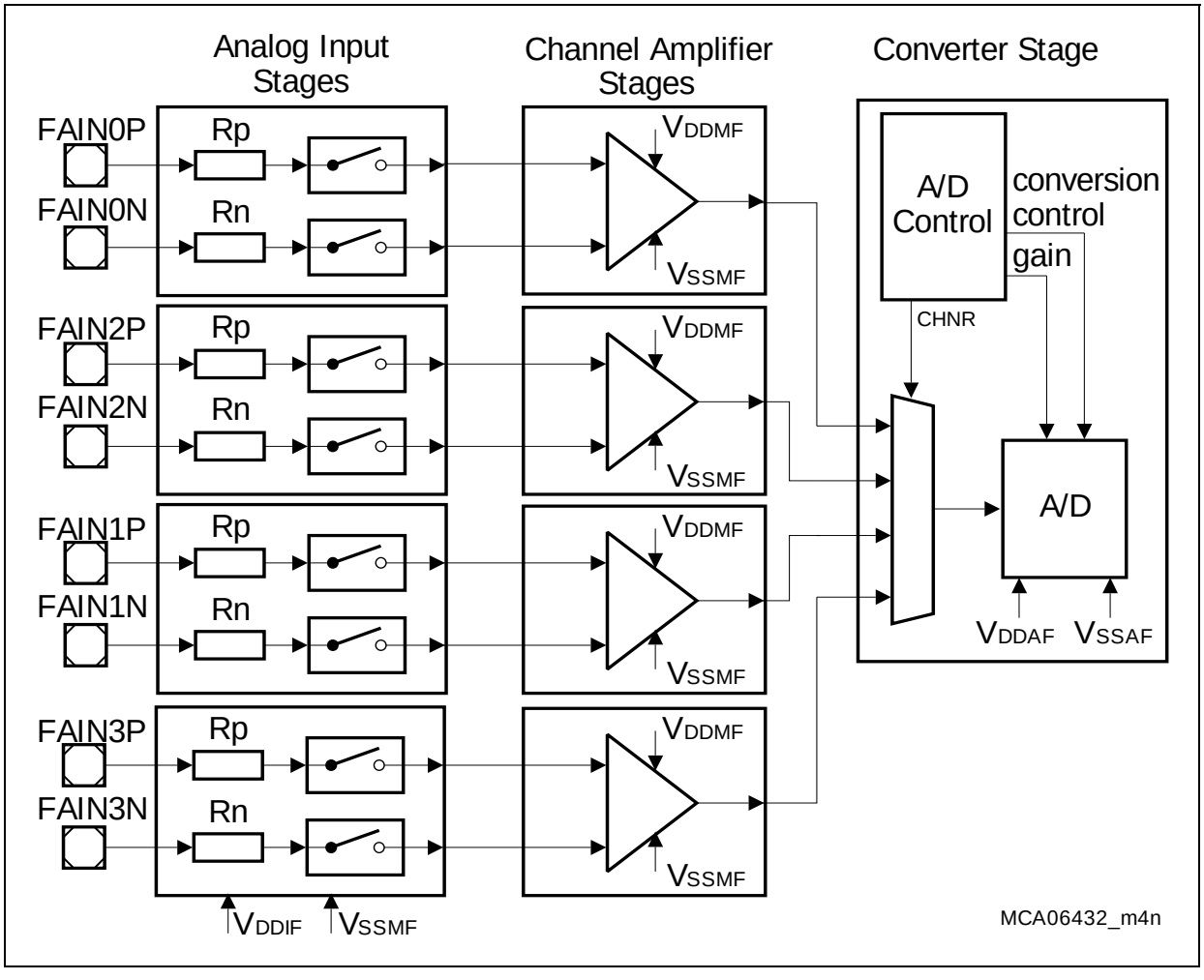


Figure 24-2 FADC Input Structure in TC1797

Note: The analog voltages of the FADC have to be stable and noise-free to ensure a high quality of the conversions.

24.2 FADC Kernel Description

The FADC kernel description covers the following items:

- Analog input stage configurations (see **Section 24.2.1**)
- Conversion timing (see **Section 24.2.3**)
- Channel triggers (see **Section 24.2.4**)
- Channel timers (see **Section 24.2.5**)
- Conversion control (see **Section 24.2.6**)
- Data reduction unit (see **Section 24.2.7**)
- Neighbor channel trigger (see **Section 24.2.8**)
- Offset calibration (see **Section 24.2.9**)
- Interrupt generation (see **Section 24.2.10**)

24.2.1 Analog Input Stage Configurations

The analog input stage makes it possible to control the input impedance by selecting different configurations independently for each FADC channel x. These combinations are shown in **Figure 24-3**.

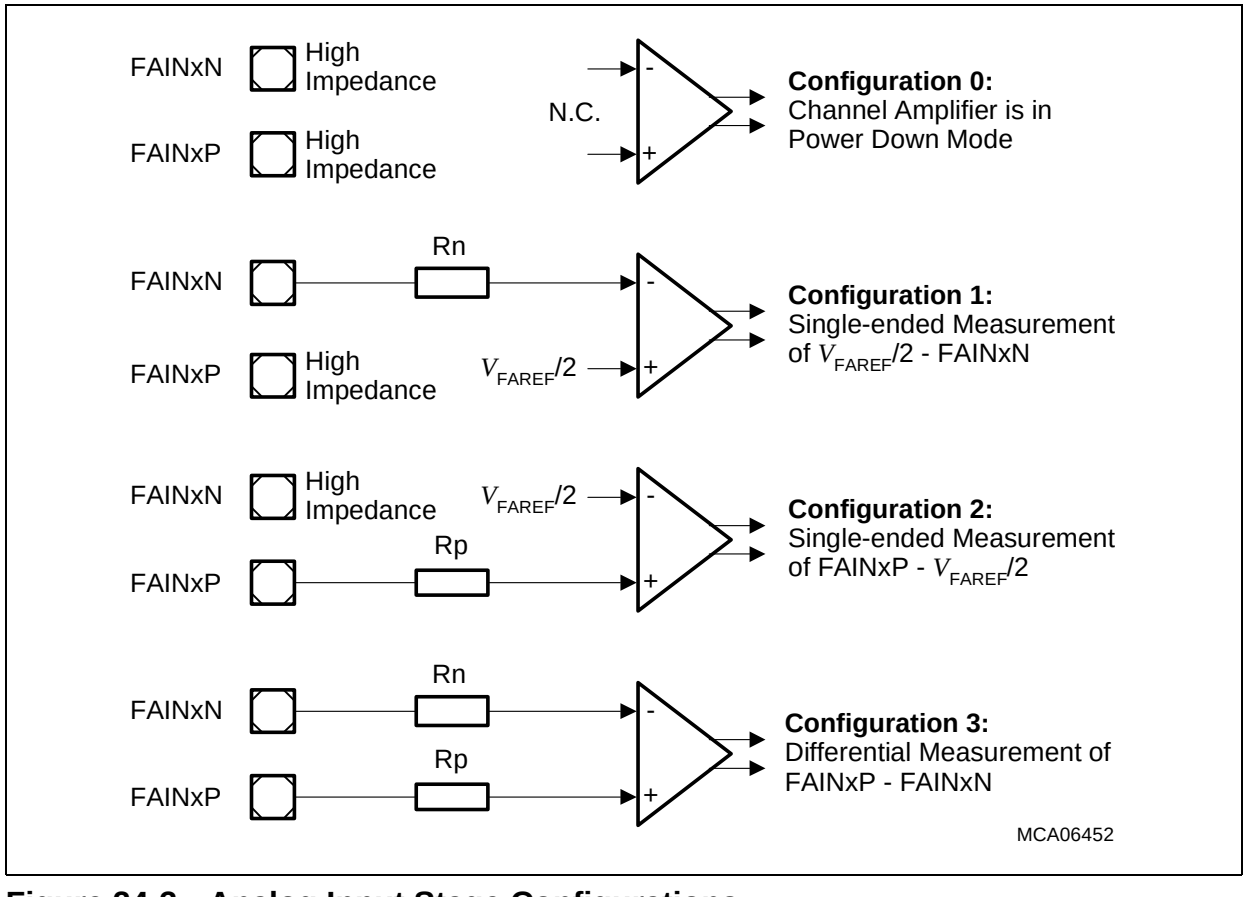


Figure 24-3 Analog Input Stage Configurations

Note: Due to the temperature characteristics of offset and gain of the internal amplifiers a TUE (total unadjusted error) cannot be specified. The input impedance for R_p and R_n is defined in the TC1797 Data Sheet.

Note: The analog input lines of the FADC can also be used as input lines for other ADCs. If both (regular ADC and FADC) are connected to the same pin at the same time, the input impedances of the analog inputs must be taken into account in order to minimize signal distortions and measurement errors.

Configuration 0: Channel Not Used (ACRx.ENP = ACRx.ENN = 0)

FADC channel x inputs FAINxP and FAINxN are in a high impedance state. The channel amplifier of the analog input stage is in power-down mode if both connected input channels are switched off for measurements.

Configuration 1: Single-ended of FAINxN (ACRx.ENP = 0 and ACRx.ENN = 1)

This configuration enables the single-ended measurement mode for $V_{VAREF}/2$ - FAINxN: The positive analog input FAINxP is disconnected is in a high impedance state. The negative analog input FAINxN is connected to the channel amplifier (input impedance is determined by R_n). The positive input of the channel amplifier is connected to $V_{FAREF}/2$ (1.65 V with V_{FAREF} = 3.3 V). If the voltage at the negative input FAINxN varies, the FADC will deliver conversion results as follows (gain = 1 selected by ACRx.GAIN = 00_B):

- FAINxN = 0 V: FADC conversion result is 768
- FAINxN = 3.3 V: FADC conversion result is 256

To cover the full range of the measurement result in this single-ended measurement mode, a gain of 2 must be selected (ACRx.GAIN = 01_B). With gain = 2, the FADC will deliver conversion results as follows:

- FAINxN = 0 V: FADC conversion result is 1023
- FAINxN = 3.3 V: FADC conversion result is 0

The voltage at the disconnected positive analog input FAINxP has no influence on the conversion result.

Configuration 2: Single-ended of FAINxP (ACRx.ENP = 1 and ACRx.ENN = 0)

This configuration enables the single-ended measurement mode for FAINxP - $V_{VAREF}/2$. The negative analog input FAINxN is disconnected and in a high impedance state. The positive analog input FAINxP is connected to the channel amplifier (input impedance is determined by R_p). The negative input of the channel amplifier is connected to $V_{FAREF}/2$ (1.65 V with V_{FAREF} = 3.3 V). If the voltage at the positive input FAINxP varies, the FADC will deliver conversion results as follows (gain = 1 selected by ACRx.GAIN = 00_B):

- FAINxP = 0 V: FADC conversion result is 256
- FAINxP = 3.3 V: FADC conversion result is 768

To cover the full range of the measurement result in this single-ended measurement mode, a gain of 2 must be selected (ACRx.GAIN = 01_B). With gain = 2, the FADC will deliver conversion results as follows:

- FAINxP = 0 V: FADC conversion result is 0
- FAINxP = 3.3 V: FADC conversion result is 1023

The voltage at the disconnected negative analog input FAINxN has no influence on the conversion result.

Configuration 3: Differential Measurement (ACRx.ENP = ACRx.ENN = 1)

This configuration enables the differential measurement mode for FAINxP - FAINxN. Both analog inputs, FAINx.N and FAINxP, are connected to the channel amplifier inputs. Their impedances are determined by R_n and R_p. The full measurement range is available.

24.2.2 Result Representation

The conversion result for FADC channel x is given by the following equations:

$$\text{Conversion Result } V_{Mx} = \text{GAIN}_x \times ((V_{FAINxP} - V_{FAREFM}) + (V_{FAREFM} - V_{FAINxN})) \quad (24.1)$$

$$\text{with } V_{FAREFM} = V_{FAGND} + (V_{FAREF} - V_{FAGND})/2$$

The absolute value of the result V_{Mx} is limited to V_{FAREF}. The mapping of the conversion result V_{Mx} to the result RCHx.ADRES is as follows:

$V_{Mx} = -V_{FAREF}$	leads to RCHx.ADRES = 0
$V_{Mx} = 0$	leads to RCHx.ADRES = 512
$V_{Mx} = +V_{FAREF}$	leads to RCHx.ADRES = 1023

For single-ended measurements, the following values are taken into account:

- if $ENP_x = 0$ then $V_{FAINxP} = V_{FAREFM}$
- if $ENN_x = 0$ then $V_{FAINxN} = V_{FAREFM}$

Note: In Multiplexer Test Mode (GCR.MUXTM = 1), the channel amplifiers are disconnected from the converter stage. The measured conversion result in multiplexer test mode is 512 plus/minus the offset of the converter stage.

24.2.3 Conversion Timing

The conversion time of the FADC is determined by the frequency of clock f_{FADC}. The conversion time including sampling, converting, and storing of the conversion result takes 21 periods of f_{FADC}.

24.2.4 Channel Triggers

As shown **Figure 24-4**, the trigger behavior of an FADC channel x is determined by its channel x trigger control logic. An FADC channel x can be triggered by three trigger sources:

- External trigger source input signal, selectable from an input vector TS[H:A]
- Internal channel x timer trigger signal
- Internal neighbor channel trigger signal

If one of these trigger sources is selected, becomes active, and the gating logic is programmed to enable trigger signals (signal ECHTIMx set), the conversion request flag CRFx becomes set indicating a valid request for FADC channel x.
The gating source input and gating mode selection logic generate an enable signal for channel x timer that determines whether any of the three conversion trigger signal sources is allowed to set the channel x conversion request flag CRFx. It can select an input signal from the input vector GS[H:A].
The control logic does the following tasks, independently in each of the FADC channels:

- Gating source input selection (CFGRx.GSEL)
- Gating mode selection (CFGRx.GM)
- Trigger source input selection (CFGRx.TSEL)
- Trigger mode selection (CFGRx.TM)
- Channel timer request generation
- Conversion request flag set/clear

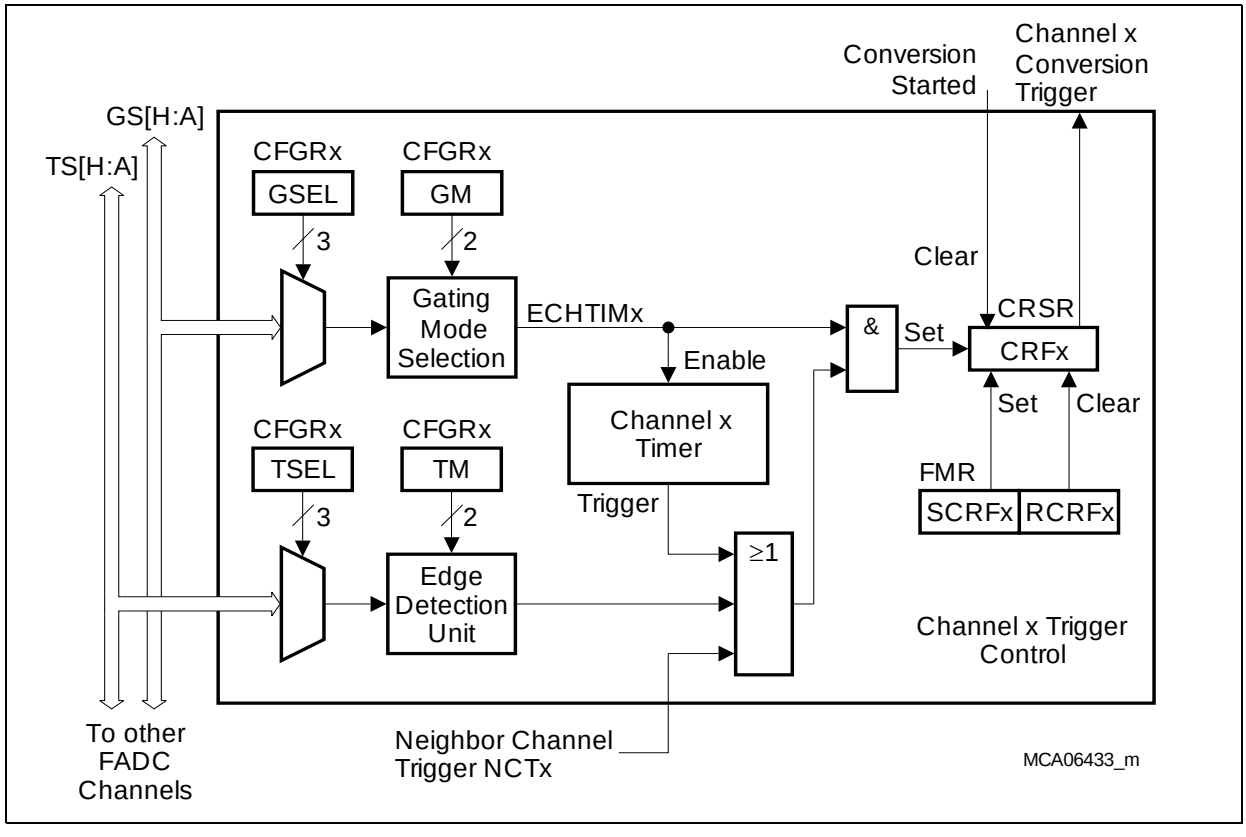


Figure 24-4 Channel Trigger Control Logic

Table 24-1 describes the possible gating modes (enabled, disabled, active gating source input polarity) of an FADC channel.

Table 24-1 Gating Modes

CFGRx.GM	Gating Mode
00_B	Conversion requests are disabled and Channel Timer is stopped. Bit CRSR.CRFx never becomes set (by hardware).
01_B	Conversion requests and Channel Timer are always enabled. Bit CRSR.CRFx becomes set by hardware with each active trigger signal.
10_B	When gating source input GSn = 1 (as selected by CFGRx.GSEL), the Channel Timer is enabled and the conversion request bit CRSR.CRFx becomes set by hardware with each active trigger signal.
11_B	When gating source input GSn = 0 (as selected by CFGRx.GSEL), the Channel Timer is enabled and the conversion request bit CRSR.CRFx becomes set by hardware with each active trigger signal.

An edge detection unit determines which edge of the trigger source input signal (as selected by CFGRx.TSEL) is generating a conversion request trigger signal. Rising, falling or both edges can be selected for trigger signal generation.

Table 24-2 Trigger Modes

CFGRx.TM	Trigger Mode
00_B	No trigger signal generated.
01_B	A conversion request trigger signal is generated on a rising edge of trigger source input TSn (selected by CFGRx.TSEL).
10_B	A conversion request trigger signal is generated on a falling edge of trigger source input TSn (selected by CFGRx.TSEL).
11_B	A conversion request trigger signal is generated on a rising or falling edge of trigger source input TSn (selected by CFGRx.TSEL).

The Conversion Request Flag CRSR.CRFx is cleared by hardware when the conversion of channel x is started. Bit CRFx can be also set or cleared by software via bits in the Flag Modification Register FMR. Writing a 1 to FMR.SCRF sets CRFx. Writing a 1 to FMR.RCRF clears CRFx (independently of FMR.SCRF).

24.2.5 Channel Timer

Each of the FADC channels contains an 8-bit Channel Timer that can be used to generate periodic conversion requests. The Channel Timer is built up by a decrementing counter that is reloaded with a programmable value. When the Channel Timer reaches zero while running, a Channel Timer trigger event is generated and the Channel Timer is reloaded with the reload value CFGRx.CTREL when the requested conversion is started. With the start of the A/D conversion, request bit CRSR.CRFx is cleared. Note that the request flag is set by a timer trigger event only if the gating condition is met (signal ECHTIMx in **Figure 24-4** set).

A clock divider, fed by the module clock f_{FADC} and common for all Channel Timers, generates several clock signals with different periods. Bit field CFGRx.CTF selects whether or not the channel x timer clock f_{CTx} is enabled and, if enabled, determines the frequency of channel x timer input clock f_{CTx}.

While the Channel Timer is disabled (CFGRx.CTM = 00_B) or if the gating condition is not met (gating line ECHTIMx delivers 0), the channel x timer value is set to 04_H. This value ensures a fast conversion trigger after the gating becomes enabled, but prevents unintended conversion starts in case of short pulses (bouncing) at the gating input.

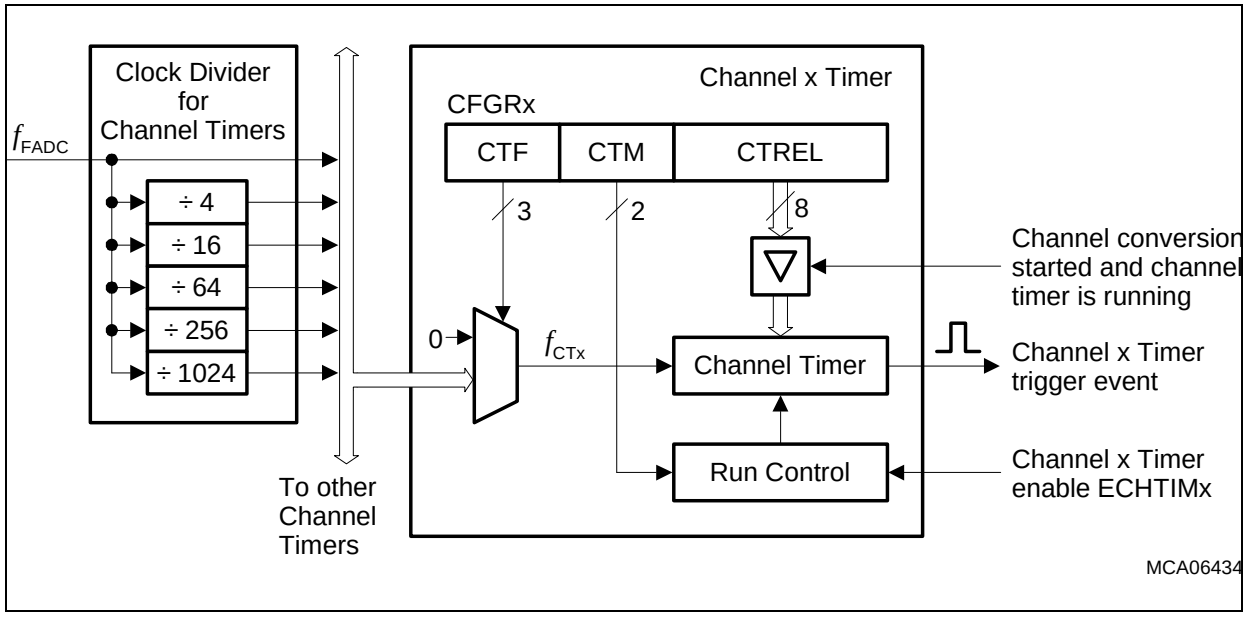


Figure 24-5 Channel Timer Block Diagram

Due to the common divider, the first event at the trigger output of CHTIMx after the start has a maximum jitter of one clock cycle of the selected channel x timer clock f_{CTx}.

A Channel x Timer input clock pulse at f_{CTx} is ignored if it occurs in the f_{FADC} clock cycle directly after the Channel x Timer has reached 0. If there is at least one f_{FADC} clock cycle between two Channel x Timer input clock pulses, all Channel x Timer input clock pulses

are taken into account. This leads to a Channel x Timer reload value (CFGRx.CTREL) whose definition depends on the ratio of f_{FADC} / f_{CTx}:

f_{FADC} / f_{CTx} = 1	Channel x timer divide factor = CTREL + 2
1 < f_{FADC} / f_{CTx} < 2	Not recommended to be used!
2 ≤ f_{FADC} / f_{CTx}	Channel x timer divide factor = CTREL + 1

In case of a f_{FADC} / f_{CTx} ratio between 1 and 2, a mixture of both divide factor definitions occurs depending on the divider ratio as programmed by the fractional divider value. Therefore, it is recommended that this ratio should not be used.

24.2.6 Conversion Control

A conversion is started when at least one of the CRSR.CRFx bits is set. A running conversion cannot be aborted and is indicated by the busy flag CRSR.BSYx set. The corresponding bit CRSR.CRFx is reset by hardware when the conversion starts.

24.2.6.1 Static Channel Priority

If more than one conversion request flag CRSR.CRFx (x = 0-3) is set, the channels are converted according to a priority scheme as defined by the bit field GCR.CRPRIO (without respecting the status of the current filter sequences).

Table 24-3 Static Channel Request Priority

Priority	GCR.CRPRIO			
	00_B	01_B	10_B	11_B
High	Channel 0	Channel 1	Channel 2	Channel 3
	Channel 1	Channel 2	Channel 3	Channel 0
	Channel 2	Channel 3	Channel 0	Channel 1
Low	Channel 3	Channel 0	Channel 1	Channel 2

24.2.6.2 Dynamic Priority Assignment

If dynamic priority assignment is enabled (GCR.DPAEN = 1), a channel that has the only active gate signal (signal ECHTIMx in **Figure 24-4**) among the four channels gets the highest priority (GCR.CRPRIO is set to the number of the channel). If more than one channel gating signal is active, GCR.CRPRIO is not changed automatically. In this case, it can be changed by software.

24.2.6.3 Clock Generation

The FADC module is provided with two clock signals: f_{CLC} and f_{FADC}. Clock f_{CLC} is used inside the FADC kernel for control purposes such as clocking of control logic, register operations, trigger detection, or filter calculation.
The clock rate of f_{FADC} is programmable. It is used inside the FADC kernel for the Channel Timers and other internal timings.

24.2.6.4 Suspend Mode Behavior

When a suspend/idle mode request is generated for the FADC module via the fractional divider, a currently running conversion is completely finished (not aborted) and, if selected, a filter calculation still takes place. Thereafter, no new conversion will be started and the state of the FADC module is frozen until the suspend/idle mode request is released again. If enabled, the related interrupts are signaled.
If suspend mode is needed, it is recommended to use to the suspend control by the fractional divider and not by the CLC register.

24.2.6.5 Alias Feature

The alias feature is mainly meant for emulation purposes and does not need to be used in standard applications. It allows a reassignment of the channel to be converted with respect to the channel number that has been requested for conversion. This feature can be used to redirect conversion requests to other input channels without changing the request SW. For example, a SW from TC1766 (with only the two input channels 0 and 1), can be adapted to either channels 0 and 1 (compatibility to older products) or to channels 2 and 3 instead. Especially when emulating several different devices with a common emulation chip, the channel reassignment allows compatible SW usage with different channel wiring to pins.

The setting of the input stages (incl. calibration information) is defined by register ACRx of the input channel that is requested (no alias-mapping). I.e. the ACRx register of the actually converted channel is used as it is permanently conntected to its channel. The result handling (incl. interrupts) also refers to the requested channel number. The alias feature is only taken into account as channel number for the conversion start.

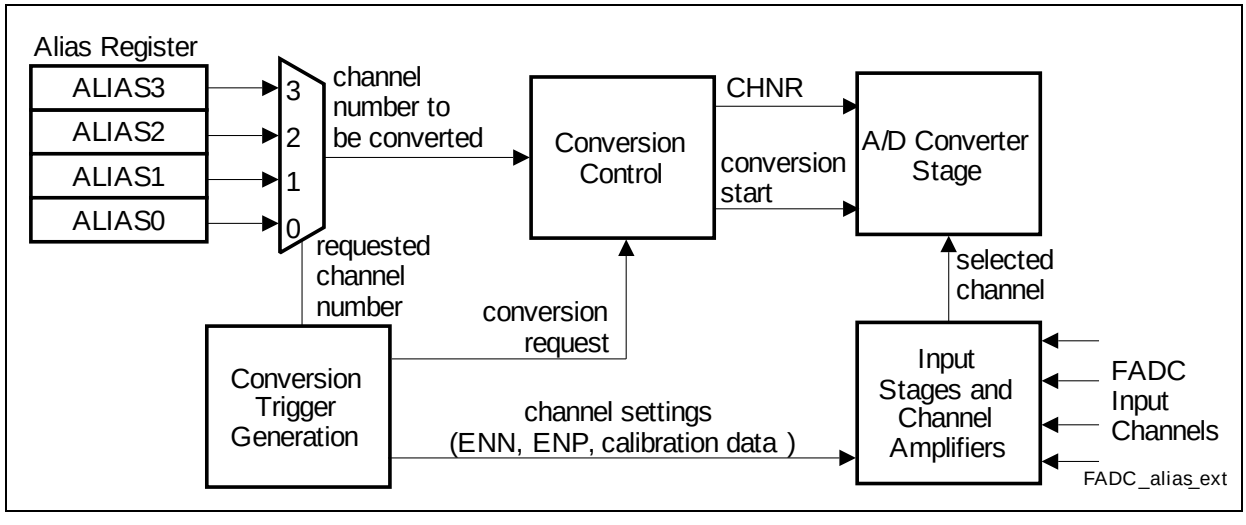


Figure 24-6 Alias Feature

24.2.7 Data Reduction Unit

A Data Reduction Unit is implemented in the FADC that operates as anti-aliasing filter. This unit allows the number of conversion data requests that are issued to the CPU or other bus masters to be reduced by adding multiple conversion results according to a certain algorithm and presenting it to the CPU or other bus masters with a reduced conversion request rate.

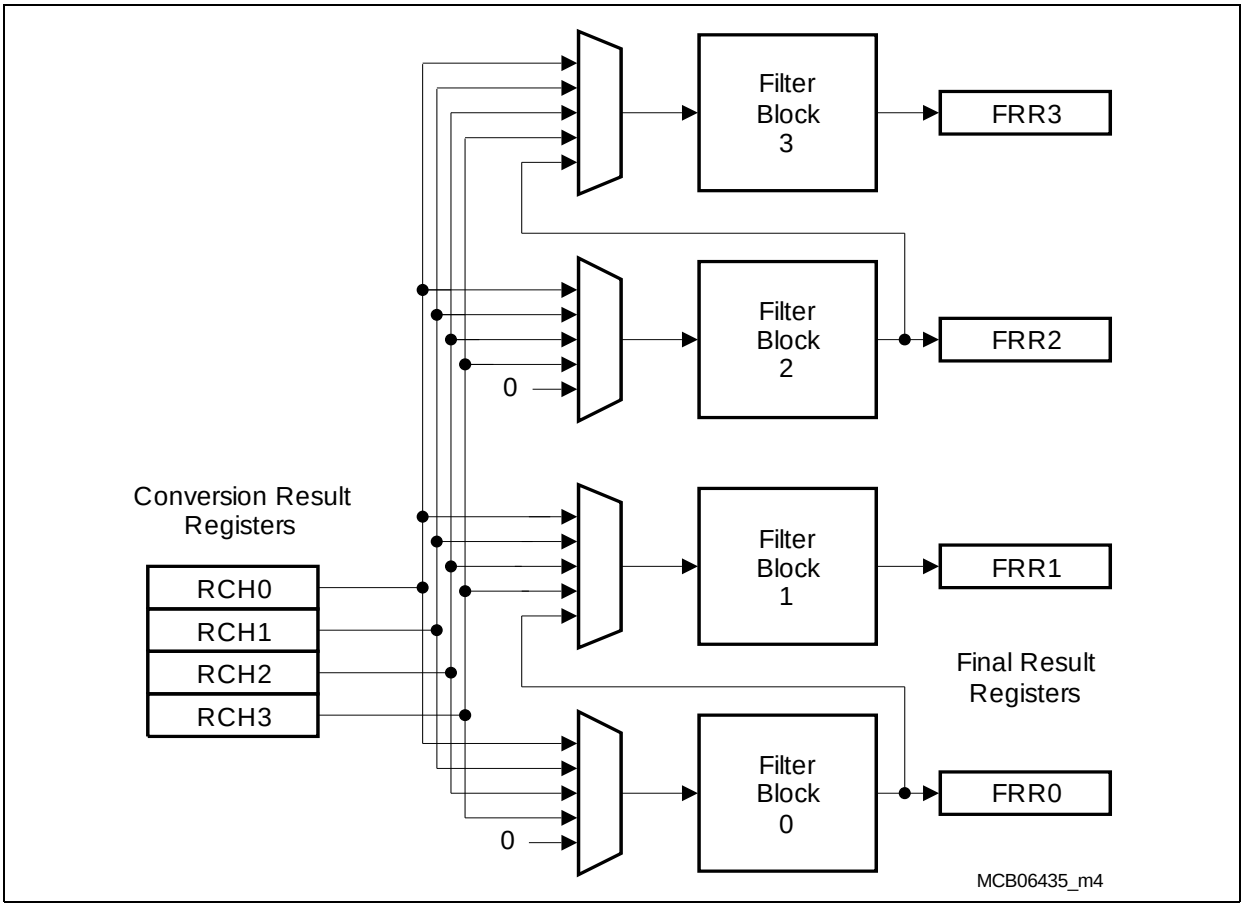


Figure 24-7 TC1797 FADC Filter Blocks

The Data Reduction Unit contains four filter blocks. Each filter block allows selection of its input data source. The input data sources are the conversion result registers of the four A/D converter channels.

Filter blocks can also be concatenated (block 0 and block 1 can be concatenated, same for block 2 and block 3). When the result of a filter operation is stored in one of the final result registers, a service request can be generated. Each filter block basically contains adder logic and intermediate storage registers that allow support for typical digital filter operations such as moving average calculations with intermediate results.

24.2.7.1 Filter Block Structure

The filter block consists of an adder and several result registers for calculating filter output data from the filter input data. For filter n, the Current Result Register CRRn is used for adding up conversion results. After a programmable number of conversion results have been added, the contents of CRRn are stored as intermediate result in the Intermediate Result Register IRR1n. The three intermediate result registers operate like a pipeline. Before IRR1n is overwritten, IRR2n is transferred to IRR3n, and IRR1n is transferred to IRR2n. The Final Result Register FRRn stores the sum that is built by the contents of the current result register and the intermediate result registers.

All result registers of the filter block can be read at any time. Please note that only one intermediate result register is available in filter block 1 (IRR11) and in filter block 3 (IRR13).

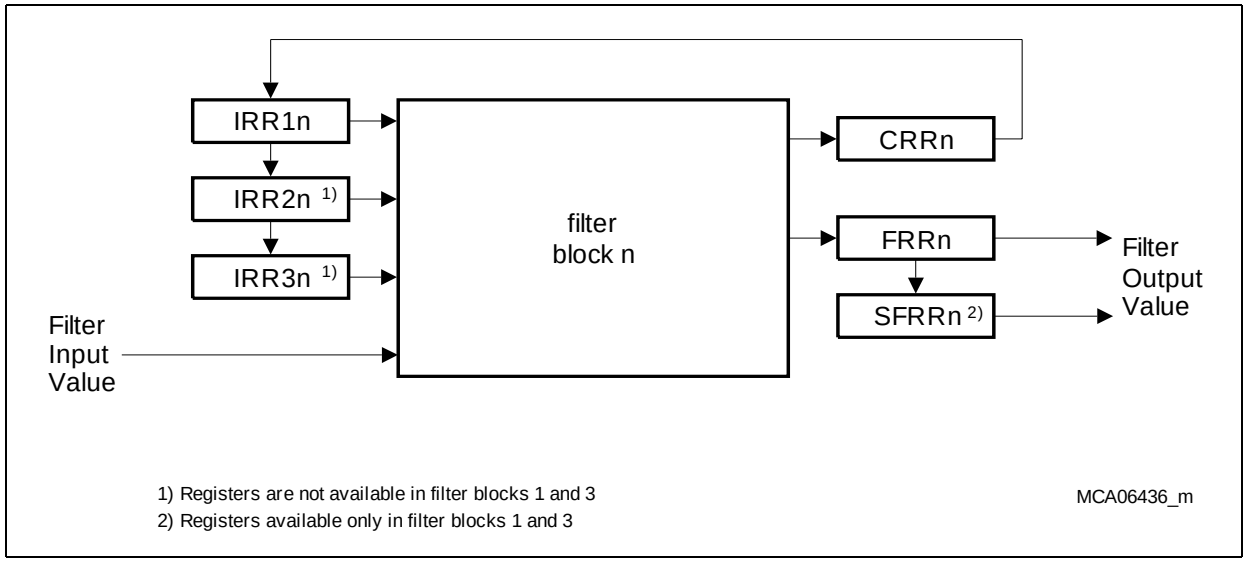


Figure 24-8 Filter Block Structure

24.2.7.2 Filter Block Operation

A filter block can be used for data-reduction or anti-aliasing filtering of the conversion results (n indicates the number of the filter block). It performs a combination of data reduction by adding and a moving average operation.

- A continuous A/D conversion is running on channel x.
- The filter input selection is set to channel x (FCRn.INSEL = 100_B + x).
- The addition length is controlled by FCRn.ADDL defining how many conversion results are added to build one intermediate result (intermediate cycle).

- The moving average length is controlled by FCRn.MAVL defining how many intermediate results are taken into account for a moving average to build the final result (final result cycle).

Intermediate Result Calculation

Each incoming conversion result is added to the content of CRRn until the programmed number of conversion results have been summed up in CRRn. At that point, CRRn contains a new intermediate result and the calculation of the next final result value by moving average is started. Then CRRn is cleared automatically after the final result cycle to be prepared for the first conversion result for the next intermediate cycle.
Before the filter operation of continuous conversion results of channel x is started, the filter block n has to be cleared (writing GCR.RSTFn = 1) after programming the filter control bit fields.

Final Result Calculation

The calculation of a final result is started when an intermediate cycle has been finished. The new intermediate result (stored in CRRn) and the contents of the intermediate registers IRRnx are added to build the final result in FRRn. The number of intermediate results taking part in the moving average operation to build the final result is programmable, the maximum is given by:

- Filter block 0: FRR0 := CRR0 + IRR10 + IRR20 + IRR30
- Filter block 1: FRR1 := CRR1 + IRR11
- Filter block 2: FRR2 := CRR2 + IRR12 + IRR22 + IRR32
- Filter block 3: FRR3 := CRR3 + IRR13

At the end of the final result cycle, the contents of IRR2n are transferred into IRR3n, then the contents of IRR1n into IRR2n, then the contents of CRRn into IRR10 (for filter blocks 0 and 2). The former contents of IRR3n are lost.
Bit field FCRn.MAVL determines the number of intermediate results that are used for the final result calculation. For filter blocks 1 and 3, only two bit combinations are valid and the intermediate result registers IRR2n and 3n are not available and handled as if they were 0.

Each update of a result register FRRn with a new final result value generates a filter block n service request.

24.2.7.3 Filter Concatenation

Filter block 1 and 3 allow filter concatenation to support more filter stages. Filter 1 can be programmed to use the result value of filter 0, similar for filter blocks 2 and 3.

Filter blocks 0 and 2 operate with the following parameters:

- Intermediate results are calculated based on the conversion results of one of the input channels (FCRn.INSEL = $1XX_B$).

- An intermediate cycle can contain a maximum of 8 conversion results.
- A final result cycle can contain a maximum of 4 intermediate results.

Filter blocks 1 and 3 operate with the following parameters:

- Intermediate results are based on the final results of filter block 0 (for filter block 1) and of filter block 2 (for filter block 3). Filter block concatenation is enabled by FCRn.INSEL = 010_B.
- An intermediate cycle can contain a maximum of 8 conversion results.
- A final result cycle can contain a maximum of 2 intermediate results.

24.2.7.4 Width of Result Registers

The additions executed in filter 0 and filter 1 together with the possible maximum values of the filter parameters determine the width of the current, intermediate, and final result registers (same for filter blocks 2 and 3).

In addition to the final result registers FRR1 and FRR3 with 20 bit width, another view is available that is shifted by 5 bit positions to the right, given by registers SFRR1 and SFRR3. This allows a representation of the data within a 16 bit word for further digital data handling.

Table 24-4 Data Width of Result Registers

Register Long Name	Register Short Name	Result Width
Filter 0 Current Result Register	CRR0	13-bit
Filter 0 Intermediate Result Register 1	IRR10	
Filter 0 Intermediate Result Register 2	IRR20	
Filter 0 Intermediate Result Register 3	IRR30	
Filter 0 Final Result Register	FRR0	15-bit
Filter 1 Current Result Register	CRR1	18-bit
Filter 1 Intermediate Result Register 1	IRR11	
Filter 1 Final Result Register	FRR1	20-bit
Filter 1 Shifted Final Result Register	SFRR1	15-bit
Filter 2 Current Result Register	CRR2	13-bit
Filter 2 Intermediate Result Register 1	IRR12	
Filter 2 Intermediate Result Register 2	IRR22	
Filter 2 Intermediate Result Register 3	IRR32	
Filter 2 Final Result Register	FRR2	15-bit
Filter 3 Current Result Register	CRR3	18-bit
Filter 3 Intermediate Result Register 1	IRR13	
Filter 3 Final Result Register	FRR3	20-bit
Filter 3 Shifted Final Result Register	SFRR3	15-bit

24.2.8 Neighbor Channel Trigger

The neighbor channel trigger feature allows the concatenation of channel conversions. This means that the start of a conversion for one channel can generate multiple channel trigger requests for the other three channels. A channel conversion request flag of a neighbor channel becomes only set by a neighbor channel trigger request if the gating condition (gating mode selection output at high level in **Figure 24-4**) in the corresponding neighbor channel is valid.

All neighbor channel trigger enable bits ENxy are located in the Neighbor Channel Trigger Register NCTR. Index “x” indicates the number of the channel that starts a neighbor channel trigger. Index “y” is the number of the neighbor channel to be triggered.

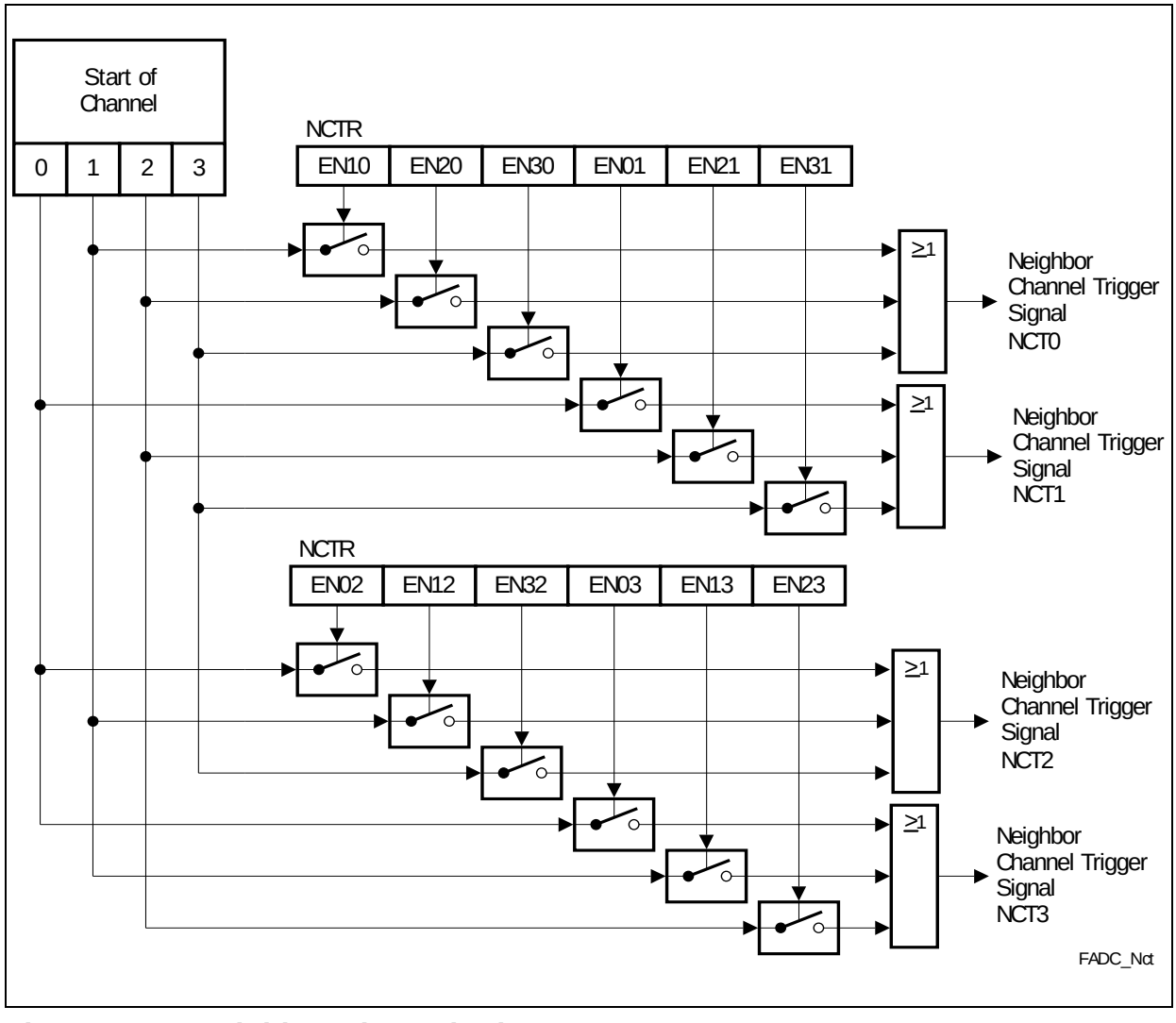


Figure 24-9 Neighbor Channel Trigger

24.2.9 Offset Calibration

The offset calibration is used to minimize the error of the FADC conversion results independently for each channel. The output of each channel amplifier can be adjusted to deliver a minimum offset value for zero input voltage difference. The channel which is calibrated is selected by GCR.CALCH. The calibration mode is selected by GCR.CALMODE. During the calibration process of a channel, the other channels can be used in normal conversion mode without restrictions.

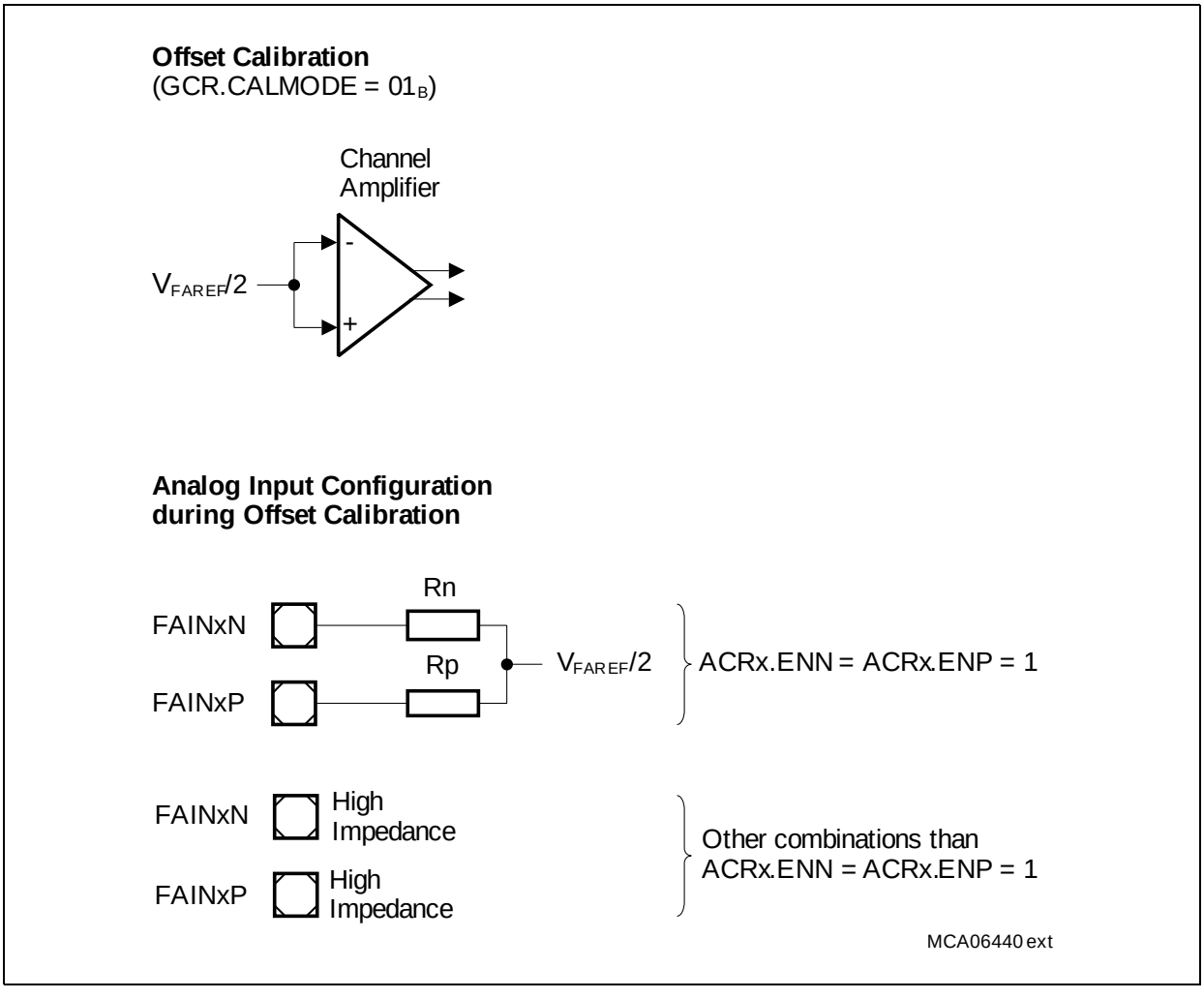


Figure 24-10 Analog Input and Channel Amplifier Configuration at Calibration

Figure 24-10 shows the channel amplifier configuration as well as the analog input pin configurations that are selected during offset calibration. Note that in the calibration modes, the impedance of the analog inputs depends on the settings of the ENN and ENP bits of the corresponding Channel x Analog Control Register ACRx.

24.2.9.1 Offset Calibration

When offset calibration is selected (GCR.CALMODE = 01_B), the channel amplifier inputs of the selected channel are both connected to $V_{FAREF}/2$. After enabling a channel amplifier for offset calibration, a delay of minimum 5 µs must be respected before starting a conversion for this channel. The conversion result must be compared by software with the tolerated offset value (a conversion result with zero offset is equal to 512). If the conversion result exceeds the tolerated range, bit field ACRx.CALOFF (with x specifying the calibrated channel) can be adjusted and a new conversion can be started. The calibration process is finished when the conversion result is in the tolerated offset range. After switching back to normal mode for channel x, a delay of minimum 5 µs must be respected before starting a new conversion for this channel.

24.2.10 Interrupt Generation

A flexible service request control structure is implemented in the FADC. The FADC provides the channel conversion request sources and the filter block request sources, that can be programmed to generate one of four service request output signals SR[3:0]. The service request compressor also makes it possible to assign more than one service request source to one service request output.

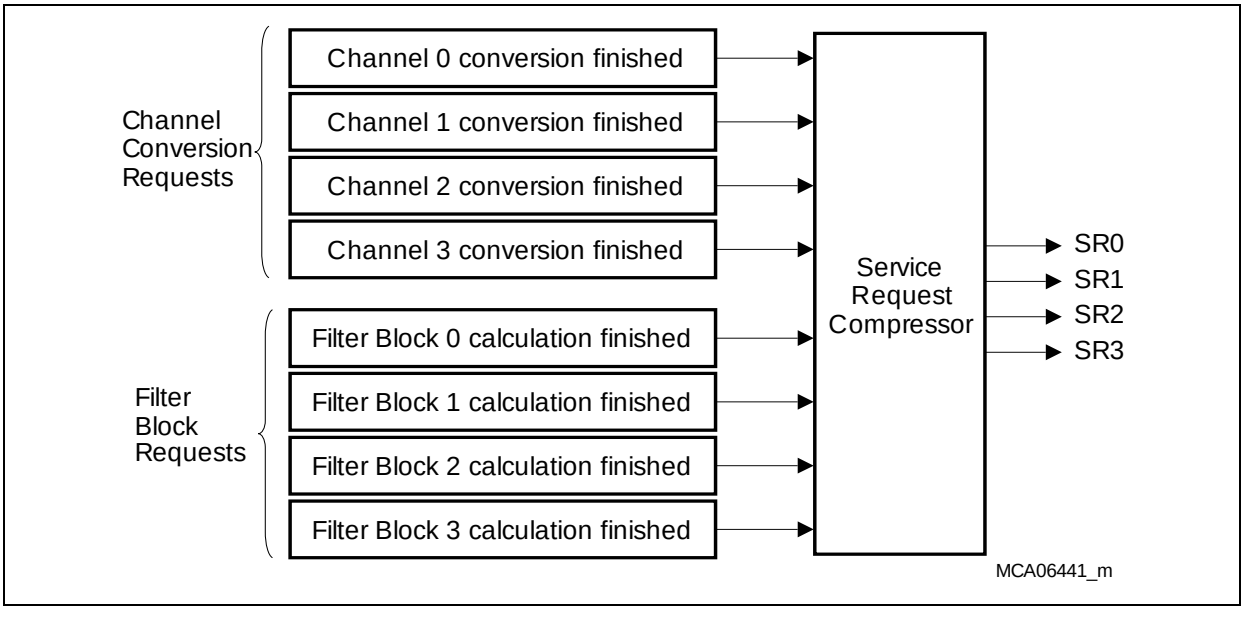


Figure 24-11 Service Request Configuration

All service requests are controlled by an identical control logic. This control logic as shown in **Figure 24-12** provides the following functionality:

- Service Request Flag
- Set/Clear Request Flag Control Bits
- Service Request Enable Bit
- Service Request Node Pointer

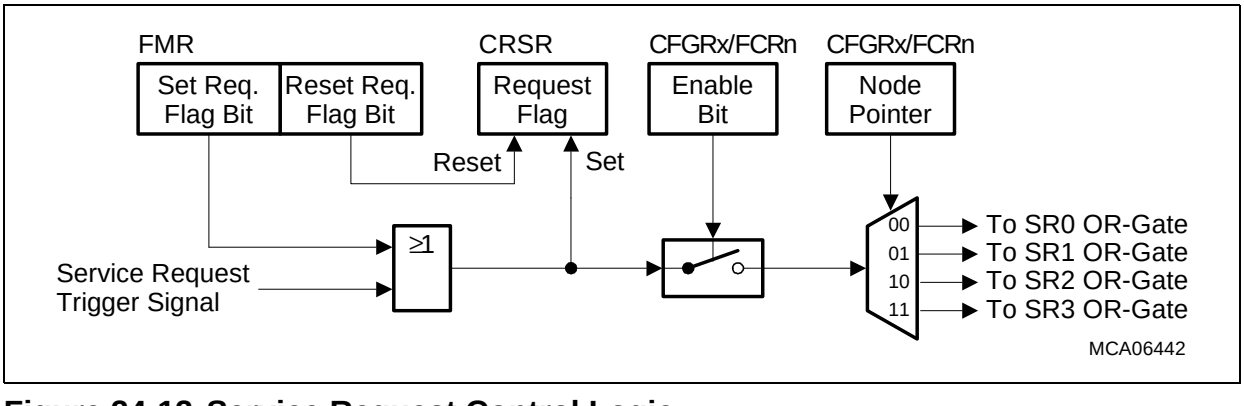


Figure 24-12 Service Request Control Logic

The request flag is always set by hardware when the corresponding request event occurs. It can also be set or cleared by software when writing a 1 to the corresponding set/clear request flag bit in the Flag Modification Register FMR. Finally, a service request event is directed to one of the four service request output lines SR[3:0] when the corresponding service request enable bit IEN is set. The service request node pointer determines which of the service request output lines SR[3:0] becomes activated.

Table 24-5 lists the six service request sources of the FADC Module with its related control and status flags/bits.

Table 24-5 Service Request Control/Status Bits/Flags

Service Request Source	Request Flag	Enable Bit	Set Request Bit / Clear Request Bit	Service Request Node Pointer
Channel x Conversion Request (x = 0-3)	CRSR.IRQx	CFGRx.IEN	FMR.SIRQx / FMR.RIRQx	CFGRx.INP
Filter Block n Request (n = 0-3)	CRSR.IRQFn	FCRn.IEN	FMR.SIRQFn / FMR.RIRQFn	FCRn.INP

In the service request compressor logic shown in **Figure 24-13**, the inputs of one SRx OR-gate are connected to all demultiplexer outputs with identical INP node pointer value. Therefore, one service request event can be only assigned to one of the four service request outputs, but one service request output can be used by multiple service request events.

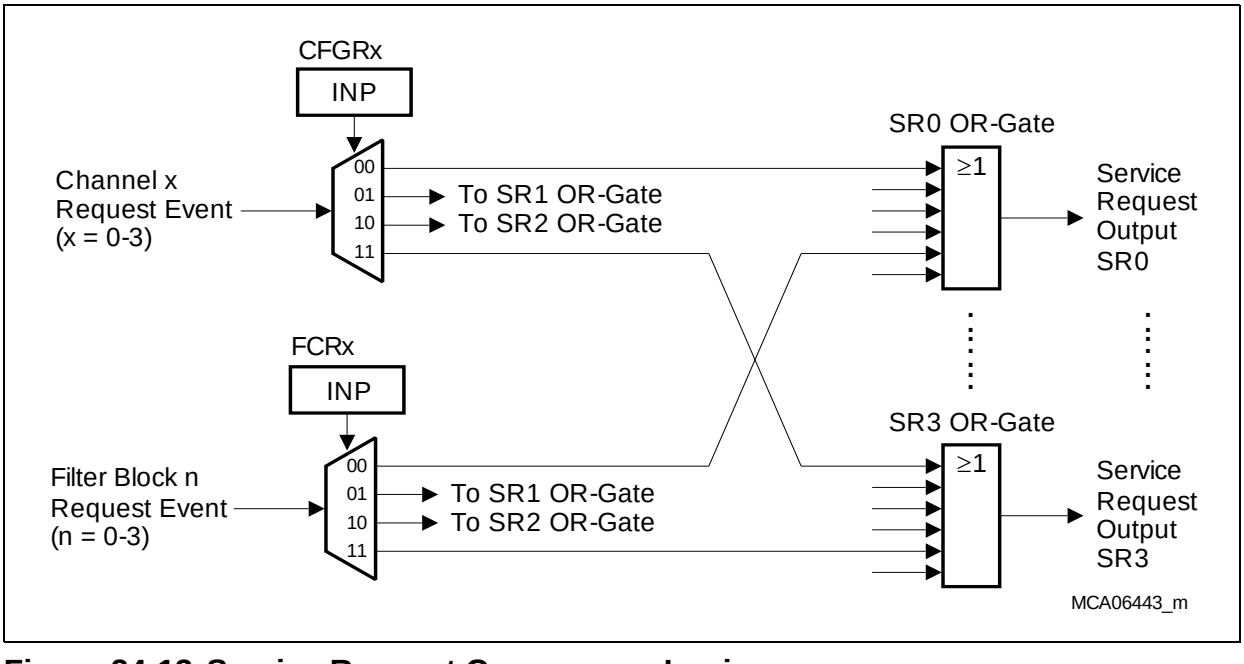


Figure 24-13 Service Request Compressor Logic

Depending on the implementation of the FADC Module in a specific micro controller, the service request output signals SR[3:0] can either be connected to an interrupt node (controlled by a service request control register) or can be used as DMA request input of a DMA controller unit. The TC1797 specific request output connections are described in the FADC implementation chapter.

24.3 FADC Register Description

This section describes the kernel registers of the FADC module. All FADC kernel register names described in this section will be referenced in other parts of the TC1797 User´s Manual by the module name prefix “FADC_”.

All registers can accessed with 8-bit, 16-bit, or 32-bit read or write operations.

Figure 24-14 FADC Kernel Registers

Access rights within the address range of an FADC kernel:

- Read access to defined register addresses: U, SV
- Write access to defined register addresses: U, SV
- Accesses to empty addresses: reserved, BE

Table 24-6 Register Overview of FADC

Short Name	Description	Offset Addr.[1)]	Access Mode		Reset	Description See
			Read	Write		
FADC Module Registers						
CLC	Clock Control Register	000_H	U, SV	SV, E	Class 3	Page 24-30
FDR	Fractional Divider Register	$00C_H$	U, SV	SV, E	Class 3	Page 24-31

Table 24-6 Register Overview of FADC

Short Name	Description	Offset Addr.[1)]	Access Mode		Reset	Description See
			Read	Write		
ID	Module Identification Register	008_H	U, SV	U, SV	Class 3	Page 24-33
SRCx x = 0 - 3	Service Request Control Registers	$0FC_H$ - x * 4_H	U, SV	U, SV	Class 3	Page 24-34
Global Registers						
CRSR	Conversion Request Status Register	010_H	U, SV	U, SV	Class 3	Page 24-35
FMR	Flag Modification Register	014_H	U, SV	U, SV	Class 3	Page 24-37
NCTR	Neighbor Channel Trigger Register	018_H	U, SV	U, SV	Class 3	Page 24-39
GCR	Global Control Register	$01C_H$	U, SV	U, SV	Class 3	Page 24-42
reserved	no BE, has to be written with 0	050_H				
ALR	Alias Register	054_H	U, SV	SV, E	Class 3	Page 24-46
Channel Registers						
CFGRx	Channel x Configuration Register (x = 0-3)	020_H + (x × 4)	U, SV	U, SV	Class 3	Page 24-48
ACRx	Channel x Analog Control Reg. (x = 0-3)	030_H + (x × 4)	U, SV	U, SV	Class 3	Page 24-HIDDEN
RCHx	Channel x Conversion Result Register (x = 0-3)	040_H + (x × 4)	U, SV	U, SV	Class 3	Page 24-54
Filter 0 Registers						
FCR0	Filter 0 Control Register	060_H	U, SV	U, SV	Class 3	Page 24-55
CRR0	Filter 0 Current Result Register	064_H	U, SV	U, SV	Class 3	Page 24-58

Table 24-6 Register Overview of FADC

Short Name	Description	Offset Addr.[1)]	Access Mode		Reset	Description See
			Read	Write		
IRR10	Filter 0 Intermediate Result Register 1	068_H	U, SV	U, SV	Class 3	Page 24-60
IRR20	Filter 0 Intermediate Result Register 2	$06C_H$	U, SV	U, SV	Class 3	Page 24-60
IRR30	Filter 0 Intermediate Result Register 3	070_H	U, SV	U, SV	Class 3	Page 24-60
FRR0	Filter 0 Final Result Register	074_H	U, SV	U, SV	Class 3	Page 24-62
Filter 1 Registers						
FCR1	Filter 1 Control Register	080_H	U, SV	U, SV	Class 3	Page 24-55
CRR1	Filter 1 Current Result Register	084_H	U, SV	U, SV	Class 3	Page 24-58
IRR11	Filter 1 Intermediate Result Register 1	088_H	U, SV	U, SV	Class 3	Page 24-60
FRR1	Filter 1 Final Result Register	094_H	U, SV	U, SV	Class 3	Page 24-62
SFRR1	Filter 1 Shifted Final Result Register	098_H	U, SV	U, SV	Class 3	Page 24-63
Filter 2 Registers						
FCR2	Filter 2 Control Register	$0A0_H$	U, SV	U, SV	Class 3	Page 24-55
CRR2	Filter 2 Current Result Register	$0A4_H$	U, SV	U, SV	Class 3	Page 24-58
IRR12	Filter 2 Intermediate Result Register 1	$0A8_H$	U, SV	U, SV	Class 3	Page 24-60
IRR22	Filter 2 Intermediate Result Register 2	$0AC_H$	U, SV	U, SV	Class 3	Page 24-60
IRR32	Filter 2 Intermediate Result Register 3	$0B0_H$	U, SV	U, SV	Class 3	Page 24-60

Table 24-6 Register Overview of FADC

Short Name	Description	Offset Addr.[1]	Access Mode		Reset	Description See
			Read	Write		
FRR2	Filter 2 Final Result Register	$0B4_H$	U, SV	U, SV	Class 3	Page 24-62
Filter 3 Registers						
FCR3	Filter 3 Control Register	$0C0_H$	U, SV	U, SV	Class 3	Page 24-55
CRR3	Filter 3 Current Result Register	$0C4_H$	U, SV	U, SV	Class 3	Page 24-58
IRR13	Filter 3 Intermediate Result Register 1	$0C8_H$	U, SV	U, SV	Class 3	Page 24-60
FRR3	Filter 3 Final Result Register	$0D4_H$	U, SV	U, SV	Class 3	Page 24-62
SFRR3	Filter 3 Shifted Final Result Register	$0D8_H$	U, SV	U, SV	Class 3	Page 24-63

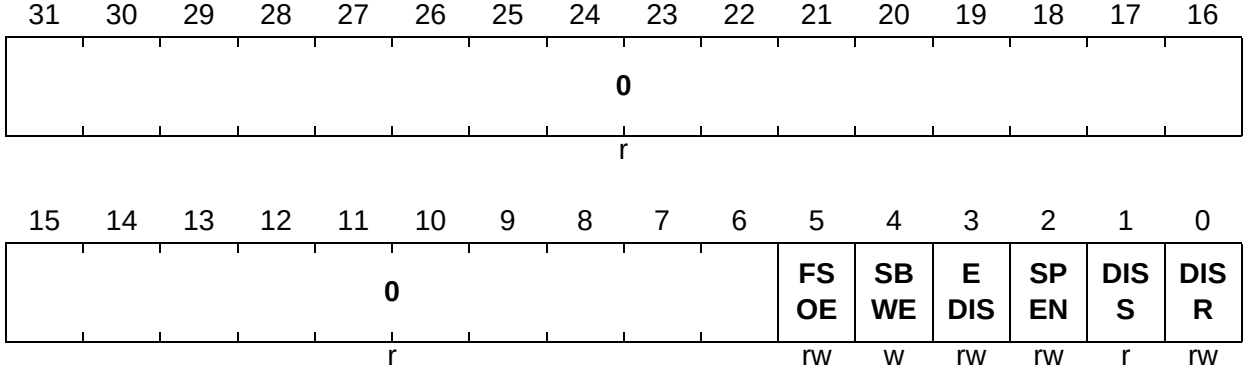
1) The absolute register address is calculated as follows:
Module Base Address + Offset Address (shown in this column)

24.3.1 System Registers

24.3.1.1 Clock Control Register

The Clock Control Register allows the programmer to control (enable/disable) the clock signal f_{CLC} under certain conditions. After a reset operation, the FADC module is disabled and its module clock signal f_{CLC} is switched off.

CLC
Clock Control Register **(000_H)** **Reset Value: 0000 0003$_H$**

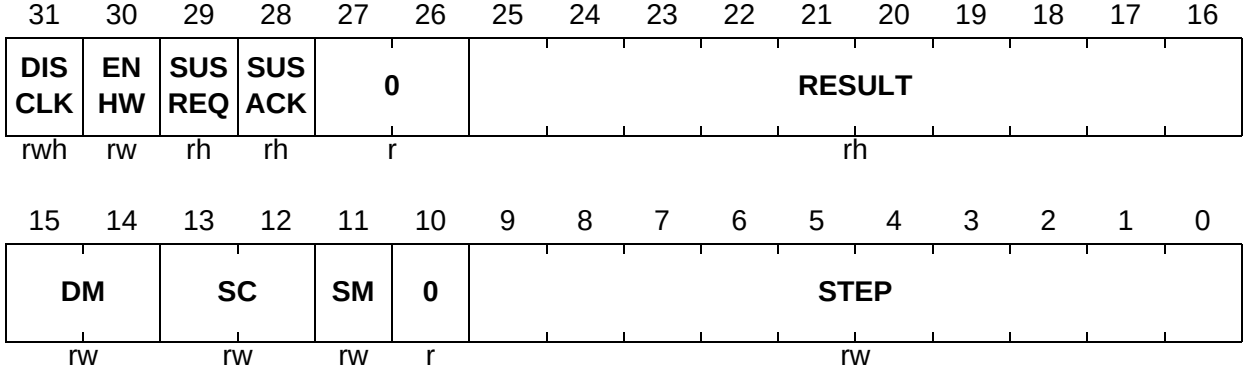


Field	Bits	Type	Description
DISR	0	rw	**Module Disable Request Bit** Used for enable/disable control of the module.
DISS	1	r	**Module Disable Status Bit** Bit indicates the current status of the module.
SPEN	2	rw	**Module Suspend Enable for OCDS** Used to enable the suspend mode.
EDIS	3	rw	**Sleep Mode Enable Control** Used to control module's sleep mode.
SBWE	4	w	**Module Suspend Bit Write Enable for OCDS** Determines whether SPEN and FSOE are write-protected.
FSOE	5	rw	**Fast Switch Off Enable** Used for fast clock switch off in Suspend Mode.
0	[31:6]	r	**Reserved** Read as 0. Should be written with 0.

24.3.1.2 Fractional Divider Register

The Fractional Divider Register allows the programmer to control the clock rate of the module clock f_{FADC}.

FDR
Fractional Divider Register **($00C_H$)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
STEP	[9:0]	rw	**Step Value** Reload or addition value for RESULT.
SM	11	rw	**Suspend Mode** SM selects between granted or immediate suspend mode.
SC	[13:12]	rw	**Suspend Control** This bit field determines the behavior of the fractional divider in suspend mode.
DM	[15:14]	rw	**Divider Mode** This bit field selects normal divider mode, fractional divider mode, and off-state.
RESULT	[25:16]	rh	**Result Value** Bit field for the addition result.
SUSACK	28	rh	**Suspend Mode Acknowledge** Indicates state of SPNDACK signal.
SUSREQ	29	rh	**Suspend Mode Request** Indicates state of SPND signal.
ENHW	30	rw	**Enable Hardware Clock Control** Controls operation of ECEN input and DISCLK bit. Should be always written with 0.

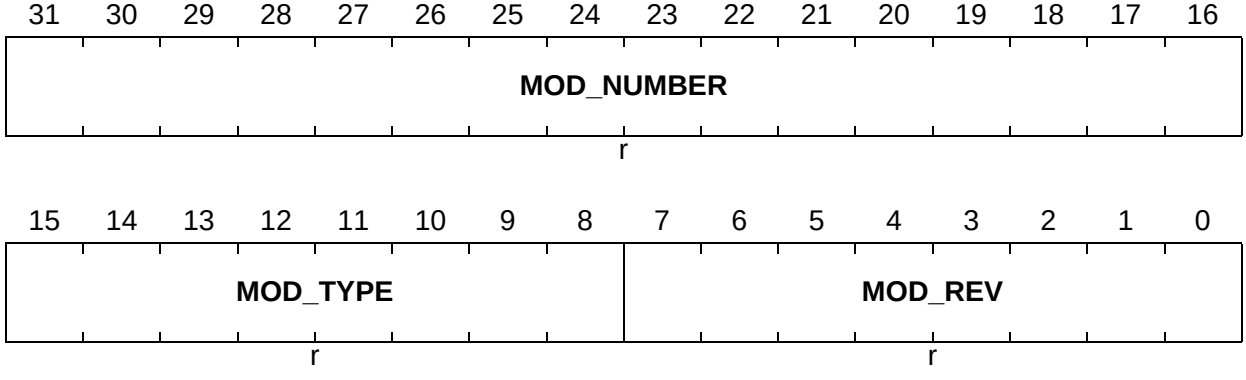
Field	Bits	Type	Description
DISCLK	31	rwh	**Disable Clock** Hardware controlled disable for f_{FADC} signal.
0	10, [27:26]	r	**Reserved** Read as 0. Should be written with 0.

24.3.1.3 Module Identification Register

The register table can be found on **Page 24-26**.

ID

Module Identification Register (008$_H$) Reset Value: 0027 C000$_H$

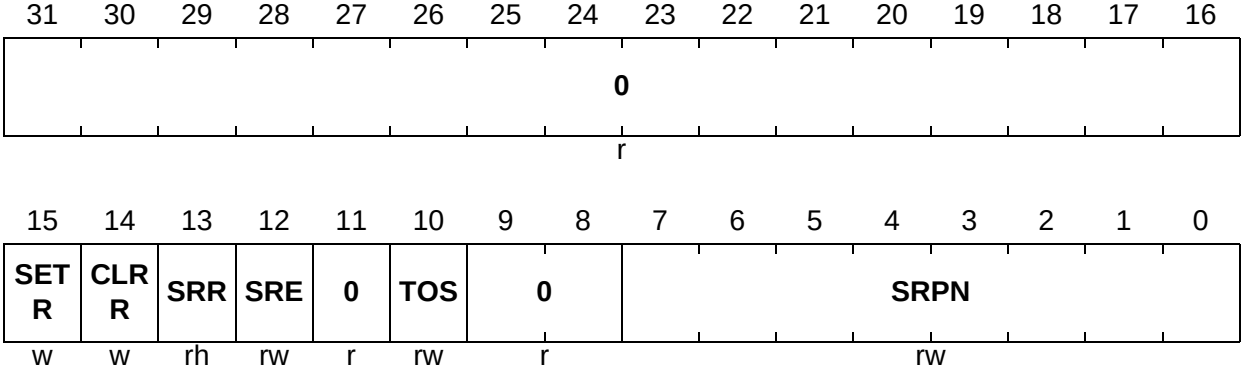


Field	Bits	Type	Description
MOD_REV	[7:0]	r	**Module Revision** This bit field indicates the revision number of the module implementation (depending on the design step). The given value of 00$_H$ is a placeholder for the actual number. Bits [3:0] refer to the version of the digital part and bits [7:4] indicate the version of the analog part (anid).
MOD_TYPE	[15:8]	r	**Module Type**
MOD_NUMBER	[31:16]	r	**Module Number**

24.3.1.4 Service Request Control Registers

Each of the interrupts of the FADC is controlled by a service request control register.

SRCx (x = 0-3)
Service Request Control Register x
($0FC_H$ - x*4_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
SRPN	[7:0]	rw	**Service Request Priority Number**
TOS	10	rw	**Type of Service Control**
SRE	12	rw	**Service Request Enable**
SRR	13	rh	**Service Request Flag**
CLRR	14	w	**Request Clear Bit**
SETR	15	w	**Request Set Bit**
0	[9:8], 11, [31:16]	r	**Reserved** Read as 0. Should be written with 0.

24.3.2 Global Registers

24.3.2.1 Conversion Request Status Register

The Conversion Request Status Register CRSR contains the flags for monitoring the state of pending conversions and the interrupt request flags.

CRSR

Conversion Request Status Register (010_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0								IRQF3	IRQF2	IRQF1	IRQF0	IRQ3	IRQ2	IRQ1	IRQ0
r								rh	rh	rh	rh	rh	rh	rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				BSY3	BSY2	BSY1	BSY0	0				CRF3	CRF2	CRF1	CRF0
r				rh	rh	rh	rh	r				rh	rh	rh	rh

Field	Bits	Type	Description
CRFx (x = 0-3)	x	rh	**Conversion Request Flag** This bit monitors whether a conversion request is pending for channel x. CRFx is set by hardware when a trigger event is detected while the gating condition delivers 1. CRFx is automatically cleared by hardware when a conversion of the channel x is started. 0_B A conversion of channel x has not been requested. 1_B A conversion of channel x has been requested. Bits CRFx can be set/cleared by software via bits FMR.RCRFx and FMR.SCRFx. If a set and a clear condition for CRFx occur simultaneously (generated by hardware and/or software), the clear condition always wins.

Field	Bits	Type	Description
BSYx (x = 0-3)	8 + x	rh	**Busy Flag** This bit indicates if a conversion is currently running for channel x. 0_B A conversion is not running. 1_B A conversion is running.
IRQx (x = 0-3)	16 + x	rh	**Interrupt Request Flag** This bit indicates that a conversion of channel x has been finished since it has been cleared by software. Interrupt requests can also be generated while IRQx is still set. An interrupt can only be generated when CFGRx.IEN = 1. 0_B A conversion has not been finished. 1_B A conversion has been finished. Bits IRQx can be set/cleared by software via bits FMR.SIRQx and FMR.RIRQx.
IRQFn (n = 0-3)	20 + n	rh	**Interrupt Request Flag for Filter n** This bit indicates that a filter sequence of filter n has been finished (new final result is available) since it has been cleared by software. Interrupt requests can also be generated while IRQ is still set. An interrupt can only be generated when FCRn.IEN = 1. 0_B A filter sequence has not been finished. 1_B A filter sequence has been finished. Bits IRQFn can be set/cleared by software via bits FMR.SIRQFn and FMR.RIRQFn.
0	[7:4], [15:12], [31:24]	r	**Reserved** Read as 0. Should be written with 0.

24.3.2.2 Flag Modification Register

The bits of the Flag Modification Register FMR allow the flags of the conversion request status register to be set/cleared by software.
If a clear and set request are issued at the same time, the target flag is cleared. It is recommended to avoid writing both bit positions (for set and for clear) of the same target bit with 1 within the same write operation.

FMR
Flag Modification Register **(014_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SIRQF3	**SIRQF2**	**SIRQF1**	**SIRQF0**	**SIRQ3**	**SIRQ2**	**SIRQ1**	**SIRQ0**	**RIRQF3**	**RIRQF2**	**RIRQF1**	**RIRQF0**	**RIRQ3**	**RIRQ2**	**RIRQ1**	**RIRQ0**
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				**SCRF3**	**SCRF2**	**SCRF1**	**SCRF0**	**0**				**RCRF3**	**RCRF2**	**RCRF1**	**RCRF0**
r				w	w	w	w	r				w	w	w	w

Field	Bits	Type	Description
RCRFx (x = 0-3)	x	w	**Clear Conversion Request Flag** This bit allows bit CRSR.CRFx to be cleared by software. 0_B No operation 1_B Bit CRSR.CRFx is cleared.
SCRFx (x = 0-3)	8 + x	w	**Set Conversion Request Flag** This bit allows bit CRSR.CRFx to be set by software. 0_B No operation 1_B Bit CRSR.CRFx is set.
RIRQx (x = 0-3)	16 + x	w	**Clear Interrupt Request Flag** This bit allows bit CRSR.IRQx to be cleared by software. 0_B No operation 1_B Bit CRSR.IRQx is cleared.

Field	Bits	Type	Description
RIRQFn (n = 0-3)	20 + n	w	**Clear Interrupt Request Flag for Filter n** This bit allows bit CRSR.IRQFn to be cleared by software. 0_B No operation 1_B Bit CRSR.IRQFn is cleared.
SIRQx (x = 0-3)	24 + x	w	**Set Interrupt Request Flag** This bit allows bit CRSR.IRQx to be set by software. 0_B No operation 1_B Bit CRSR.IRQx is set and an interrupt is generated if CFGRx.IEN = 1.
SIRQFn (n = 0-3)	28 + n	w	**Set Interrupt Request Flag for Filter n** This bit allows bit CRSR.IRQFn to be set by software. 0_B No operation 1_B Bit CRSR.IRQFn is set and an interrupt is generated if FCRn.IEN = 1.
0	[7:4], [15:12]	r	**Reserved** Read as 0. Should be written with 0.

24.3.2.3 Neighbor Channel Trigger Register

The Neighbor Channel Trigger Register NCTR contains the enable bits for the neighbor channel trigger signal (NCTx) generation (see **Page 24-20**).

NCTR
Neighbor Channel Trigger Register (018$_H$) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0					EN32	EN31	EN30	0				EN23	0	EN21	EN20
r					rw	rw	rw	r				rw	r	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				EN13	EN12	0	EN10	0				EN03	EN02	EN01	0
r				rw	rw	r	rw	r				rw	rw	rw	r

Field	Bits	Type	Description
EN01	1	rw	**Enable Neighbor Channel Trigger 01** This bit enables the neighbor channel trigger for channel 1 when a conversion of channel 0 is started. 0_B No action. 1_B A trigger will be generated.
EN02	2	rw	**Enable Neighbor Channel Trigger 02** This bit enables the neighbor channel trigger for channel 2 when a conversion of channel 0 is started. 0_B No action. 1_B A trigger will be generated.
EN03	3	rw	**Enable Neighbor Channel Trigger 03** This bit enables the neighbor channel trigger for channel 3 when a conversion of channel 0 is started. 0_B No action. 1_B A trigger will be generated.
EN10	8	rw	**Enable Neighbor Channel Trigger 10** This bit enables the neighbor channel trigger for channel 0 when a conversion of channel 1 is started. 0_B No action. 1_B A trigger will be generated.

Field	Bits	Type	Description
EN12	10	rw	**Enable Neighbor Channel Trigger 12** This bit enables the neighbor channel trigger for channel 2 when a conversion of channel 1 is started. 0_B No action. 1_B A trigger will be generated.
EN13	11	rw	**Enable Neighbor Channel Trigger 13** This bit enables the neighbor channel trigger for channel 3 when a conversion of channel 1 is started. 0_B No action. 1_B A trigger will be generated.
EN20	16	rw	**Enable Neighbor Channel Trigger 20** This bit enables the neighbor channel trigger for channel 0 when a conversion of channel 2 is started. 0_B No action. 1_B A trigger will be generated.
EN21	17	rw	**Enable Neighbor Channel Trigger 21** This bit enables the neighbor channel trigger for channel 1 when a conversion of channel 2 is started. 0_B No action. 1_B A trigger will be generated.
EN23	19	rw	**Enable Neighbor Channel Trigger 23** This bit enables the neighbor channel trigger for channel 3 when a conversion of channel 2 is started. 0_B No action. 1_B A trigger will be generated.
EN30	24	rw	**Enable Neighbor Channel Trigger 30** This bit enables the neighbor channel trigger for channel 0 when a conversion of channel 3 is started. 0_B No action. 1_B A trigger will be generated.
EN31	25	rw	**Enable Neighbor Channel Trigger 31** This bit enables the neighbor channel trigger for channel 1 when a conversion of channel 3 is started. 0_B No action. 1_B A trigger will be generated.

Field	Bits	Type	Description
EN32	26	rw	**Enable Neighbor Channel Trigger 32** This bit enables the neighbor channel trigger for channel 2 when a conversion of channel 3 is started. 0_B No action. 1_B A trigger will be generated.
0	0, [7:4], 9, [15:12], 18, [23:20], [31:27]	r	**Reserved** Read as 0. Should be written with 0.

Note: The hardware does not check whether the enable bits are set in such a way as to describe a loop of conversion requests (e.g. 0 triggers 2, 2 triggers 3 and 3 triggers 0, etc.). It is in the responsibility of the user to set these bits in an appropriate way.

24.3.2.4 Global Control Register

The Global Control Register GCR contains bits used to reset the Channel Timers, the filters and to control global FADC settings.

GCR
Global Control Register **($01C_H$)** **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0				CALCH		CALMODE		0		ANON	MUXTM	RESWEN	DPAEN	CRPRIO	
r				rw		rw		r		rw	rw	rw	rw	rwh	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0			RSTF3	RSTF2	RSTF1	RSTF0	RCD	0				RCT3	RCT2	RCT1	RCT0
r			w	w	w	w	w	r				w	w	w	w

Field	Bits	Type	Description
RCTx (x = 0-3)	x	w	**Reload Channel Timer** 0_B Channel x Timer will not be changed. 1_B Channel x Timer will be loaded with its reload value.
RCD	8	w	**Reset Common Divider** 0_B The common divider will not be changed. 1_B The common divider will be cleared.
RSTFn (n = 0-3)	9 + n	w	**Reset Filter n** 0_B The contents of filter n will not be changed. 1_B The contents of filter n will be cleared. The values of the bits in the filter registers will be cleared, except bit field CRRn.AC that is loaded with the value of FCRn.ADDL.

Field	Bits	Type	Description
CRPRIO	[17:16]	rwh	**Conversion Request Priority** This bit field determines the priority of the conversion requests if more than one channel is requested. If the dynamic priority assignment is enabled, the priority is automatically changed as a function of the gating inputs. The priority of the channels is: 00_B Channel 0 before channel 1 before channel 2 before channel 3 01_B Channel 1 before channel 2 before channel 3 before channel 0 10_B Channel 2 before channel 3 before channel 0 before channel 1 11_B Channel 3 before channel 0 before channel 1 before channel 2
DPAEN	18	rw	**Dynamic Priority Assignment Enable** If the dynamic priority assignment is enabled, the priority bit field CRPRIO is automatically changed as a function of the gating input signals. In this case, the channel that is active while the other three channels are not active gets the highest priority. 0_B The dynamic priority assignment is disabled. 1_B The dynamic priority assignment is enabled.
RESWEN	19	rw	**Result Write Enable** This bit enables a write action to the result registers RCHx (x = 0-3) of the FADC. 0_B Write accesses to the result registers are not taken into account. The written data is discarded. 1_B Write accesses to the result registers are taken into account. The former value of the written result register is overwritten by the write data. If a filter is sensitive to the written result register, the written value is taken as new filter input value.

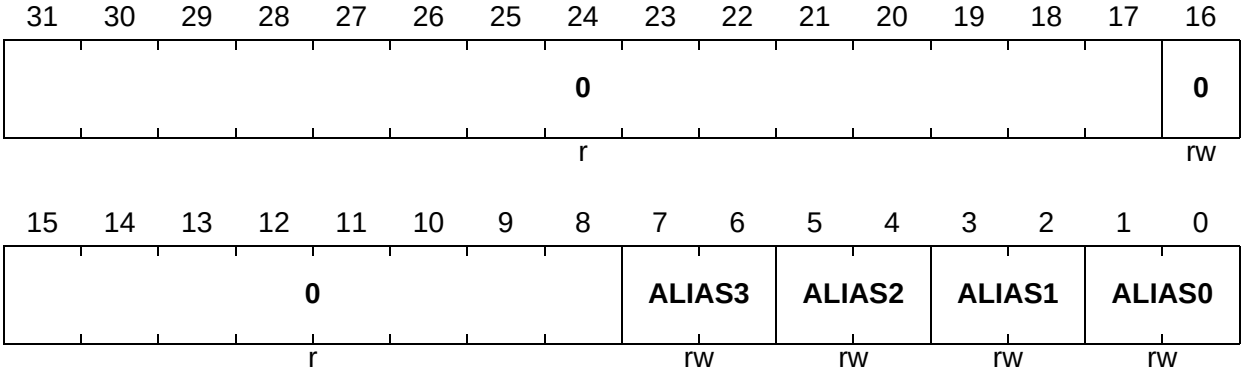
Field	Bits	Type	Description
MUXTM	20	rw	**Multiplexer Test Mode** The input multiplexer to select a channel for the conversion can be tested by opening all multiplexer inputs. In multiplexer test mode, the channel amplifiers are not connected to the converter stage. 0_B The multiplexer test mode is disabled. 1_B The multiplexer test mode is enabled.
ANON	21	rw	**Analog Part ON** This bit enables the analog part of the FADC. This bit must be set to convert the analog input signal to a digital value. 0_B The complete analog part is in power-down mode, the amplifiers and comparators are switched off. Conversions are not possible. 1_B The analog part is enabled.
CALMODE	[25:24]	rw	**Calibration Mode** This bit field enables the offset calibration for the channel selected by CALCH. 00_B No calibration process is running. All channels are in normal mode (default after reset). 01_B The analog channel selected by CALCH is in offset calibration mode. The other channels are in normal mode. 10_B All channels other than the analog channel selected by CALCH are in normal mode. 11_B Reserved
CALCH	[27:26]	rw	**Calibration Channel** This bit field selects the channel for the calibration process determined by CALMODE. The setting of CALCH is only taken into account while a calibration process is running. 00_B The analog input channel 0 is selected for a calibration process. 01_B The analog input channel 1 is selected for a calibration process. 10_B The analog input channel 2 is selected for a calibration process. 11_B The analog input channel 3 is selected for a calibration process.

Field	Bits	Type	Description
0	[7:4], [15:13], [23:22], [31:28]	r	**Reserved** Read as 0. Should be written with 0.

24.3.2.5 Alias Register

The Alias Register contains bit fields allowing a re-assignment of the requested channel number to the actually converted channel.

ALR
Alias Register **(054_H)** **Reset Value: 0000 00E4$_H$**



Field	Bits	Type	Description
ALIAS0	[1:0]	rw	**Alias of Channel 0** This bit field defines which channel is converted if a trigger for channel 0 occurs. 00_B Channel 0 will be converted (default). 01_B Channel 1 will be converted. 10_B Channel 2 will be converted. 11_B Channel 3 will be converted.
ALIAS1	[3:2]	rw	**Alias of Channel 1** This bit field defines which channel is converted if a trigger for channel 1 occurs. 00_B Channel 0 will be converted. 01_B Channel 1 will be converted (default). 10_B Channel 2 will be converted. 11_B Channel 3 will be converted.
ALIAS2	[5:4]	rw	**Alias of Channel 2** This bit field defines which channel is converted if a trigger for channel 2 occurs. 00_B Channel 0 will be converted. 01_B Channel 1 will be converted. 10_B Channel 2 will be converted (default). 11_B Channel 3 will be converted.

Field	Bits	Type	Description
ALIAS3	[7:6]	rw	**Alias of Channel 3** This bit field defines which channel is converted if a trigger for channel 3 occurs. 00_B Channel 0 will be converted. 01_B Channel 1 will be converted. 10_B Channel 2 will be converted. 11_B Channel 3 will be converted (default).
0	16	rw	**Placeholder Bit** This bit position is a placeholder for further extensions and should be written with 0.
0	[31:17], [15:8]	r	**Reserved** Read as 0. Should be written with 0.

24.3.3 Channel Registers

24.3.3.1 Channel Configuration Registers

The Channel x Configuration Register CFGRx contains the bits for the selection of the trigger source, the gating source, and other channel settings of channel x.

CFGRx (x = 0-3)

Channel x Configuration Register (020_H+x*4_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IEN	**0**	**INP**		**0**				**CTREL**							
rw	r	rw		r				rw							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**CTF**			**CTM**		**TM**		**GM**		**TSEL**			**GSEL**		
r	rw			rw		rw		rw		rw			rw		

Field	Bits	Type	Description
GSEL	[2:0]	rw	**Gating Selection** This bit field selects the gating source input signal for channel x. 000_B Gating source input signal GSA selected 001_B Gating source input signal GSB selected 010_B Gating source input signal GSC selected 011_B Gating source input signal GSD selected 100_B Gating source input signal GSE selected 101_B Gating source input signal GSF selected 110_B Gating source input signal GSG selected 111_B Gating source input signal GSH selected

Field	Bits	Type	Description
TSEL	[5:3]	rw	**Trigger Selection** This bit field selects the trigger source input signal for channel x. 000_B Trigger source input signal TSA selected 001_B Trigger source input signal TSB selected 010_B Trigger source input signal TSC selected 011_B Trigger source input signal TSD selected 100_B Trigger source input signal TSE selected 101_B Trigger source input signal TSF selected 110_B Trigger source input signal TSG selected 111_B Trigger source input signal TSH selected
GM	[7:6]	rw	**Gating Mode** This bit field determines the functionality of the gating (enable) signal. It determines whether and under which condition the generation of conversion requests by trigger signals is possible. 00_B Conversion requests are disabled and the Channel Timer is stopped. CRFx never becomes set (by hardware). 01_B Conversion requests and the Channel Timer are always enabled. CRFx becomes set by hardware with each active trigger signal. 10_B Conversion requests and the Channel Timer are enabled only if the gating source input (as selected by CFGRx.GSEL) is at high level. 11_B Conversion requests and the Channel Timer are enabled only if the gating source input (as selected by CFGRx.GSEL) is at low level.

Field	Bits	Type	Description
TM	[9:8]	rw	**Trigger Mode** This bit field enables the triggering and determines the edge of the trigger source input signal that generates a conversion trigger signal. 00_B No conversion trigger signals are generated. Edge detection unit is switched off. 01_B A conversion request is generated (if gating enabled) on a rising edge of a trigger source input (as selected by CFGRx.TSEL). 10_B A conversion request is generated (if gating enabled) on a falling edge of a trigger source input (as selected by CFGRx.TSEL). 11_B A conversion request is generated (if gating enabled) on both, rising and falling, edges of a trigger source input (as selected by CFGRx.TSEL).
CTM	[11:10]	rw	**Channel Timer Mode** This bit determines the operating mode of channel x timer. 00_B Channel x timer is switched off. 01_B Channel timer is permanently running. 10_B Channel timer is running only if ECHTIMx = 1. 11_B Reserved A Channel Timer trigger event is generated each time the channel x timer value reaches 00_H. While the Channel Timer is not running (CTM = 00_B or signal ECHTIMx = 0), the Channel Timer is loaded with 04_H.
CTF	[14:12]	rw	**Channel Timer Frequency** This bit field controls the channel x timer input clock f_{CT} (enable control and frequency selection). 000_B f_{CTx} is disabled. 001_B f_{CTx} is enabled with frequency f_{FADC}. 010_B f_{CTx} is enabled with frequency f_{FADC} / 4. 011_B f_{CTx} is enabled with frequency f_{FADC} / 16. 100_B f_{CTx} is enabled with frequency f_{FADC} / 64. 101_B f_{CTx} is enabled with frequency f_{FADC} / 256. 110_B f_{CTx} is enabled with frequency f_{FADC} / 1024. 111_B Reserved; do not use this combination.

Field	Bits	Type	Description
CTREL	[23:16]	rw	**Channel Timer Reload Value** This bit field determines the reload value of the Channel Timer CHTIMx, see **Section 24.2.5**. If CTREL = 0, no trigger event is generated.
INP	[29:28]	rw	**Interrupt Node Pointer** This bit field selects which service request output line will be activated when a conversion of channel x is finished while CFGRx.IEN is set. 00_B Service request output SR0 is selected. 01_B Service request output SR1 is selected. 10_B Service request output SR2 is selected. 11_B Service request output SR3 is selected.
IEN	31	rw	**Interrupt Enable** This bit enables the generation of a service request when a conversion of channel x is finished. 0_B Channel x conversion service request generation is disabled. 1_B Channel x conversion service request generation is enabled.
0	15, [27:24], 30	r	**Reserved** Read as 0. Should be written with 0.

24.3.3.2 Analog Control Registers

The Channel x Analog Control Register ACRx contains the bits that control the analog input stage.

ACRx (x = 0-3)
Channel x Analog Control Register

(030_H+x*4_H) **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0		CALOFF3	0	CALOFF[2:0]			0				ENN	ENP	GAIN	
r	rw		rw	r	rw			r				rw	rw	rw	

Field	Bits	Type	Description
GAIN	[1:0]	rw	**Amplifier Gain** This bit field determines the amplifier gain for channel x. 00_B The selected amplifier gain is 1. 01_B The selected amplifier gain is 2. 10_B The selected amplifier gain is 4. 11_B The selected amplifier gain is 8.
ENP	2	rw	**Enable Positive Input** This bit enables the voltage measurement on the FAINxP analog input. 0_B Analog input FAINxP is high-impedance. The upper half of the measuring range is not available. 1_B Analog input FAINxP line is connected to the channel amplifier. The upper half of the measuring range is available.

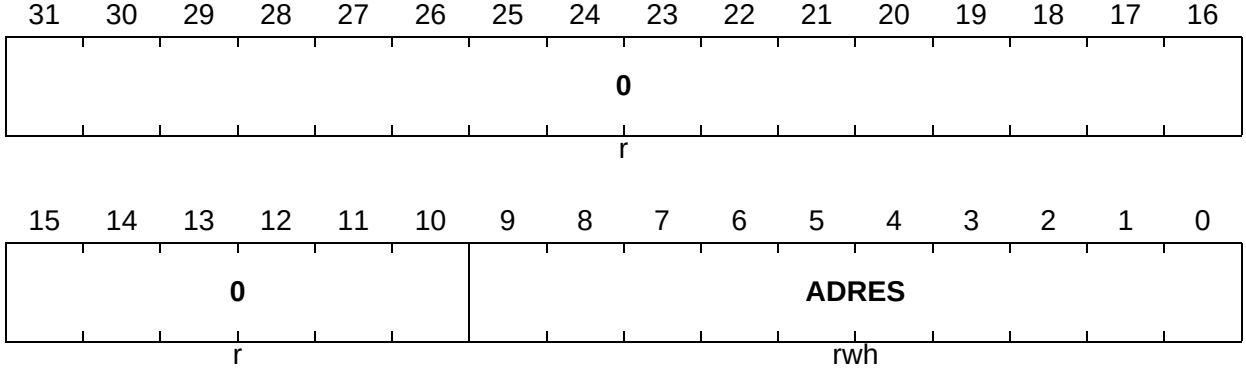
Field	Bits	Type	Description
ENN	3	rw	**Enable Negative Input** This bit enables the voltage measurement on the FAINxN analog input. 0_B Analog input FAINxN is high-impedance. The lower half of the measuring range is not available. 1_B Analog input FAINxN line is connected to the channel amplifier. The lower half of the measuring range is available.
CALOFF[2:0]	[10:8]	rw	**Calibrate Offset** This bit field determines the value applied for the offset calibration for channel x. The calibrate offset value is composed by the most significant bit CALOFF3 and bit field CALOFF[2:0], resulting in a 4-bit bit field CALOFF[3:0].
CALOFF3	12	rw	
0	[14:13]	rw	**Reserved** Read as 0. Should be written with 0.
0	[7:4], 11, [31:15]	r	**Reserved** Read as 0. Should be written with 0.

24.3.3.3 Conversion Result Registers

The Channel x Conversion Result Register RCHx contains the conversion result of channel x.

RCHx (x = 0-3)
Channel x Conversion Result Register

(040_H+x*4_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
ADRES	[9:0]	rwh	**AD Conversion Result** This bit field contains the conversion result of channel x. ADRES can only be overwritten by software if GCR.RESWEN = 1.
0	[31:10]	r	**Reserved** Read as 0. Should be written with 0.

24.3.4 Filter Registers

24.3.4.1 Filter Control Registers

Filter blocks are controlled by bits in the Filter n Control Registers FCRn.

FCRn (n = 0-3)
Filter n Control Register **(060_H+n*20_H)** **Reset Value: 0000 0000_H**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IEN	0	INP		0	INSEL			0		MAVL		0	ADDL		
rw	r	rw		r	rw			r		rw		r	rw		

Field	Bits	Type	Description
ADDL	[2:0]	rw	**Addition Length** This bit field determines the number of filter input values that are added to obtain one intermediate result. 000_B Each filter input value is considered as intermediate result. 001_B 2 filter input values are added up. 010_B 3 filter input values are added up. 011_B 4 filter input values are added up. 100_B 5 filter input values are added up. 101_B 6 filter input values are added up. 110_B 7 filter input values are added up. 111_B 8 filter input values are added up.

Field	Bits	Type	Description
MAVL	[5:4]	rw	**Moving Average Length** This bit field determines the number of intermediate results that are added up for a final result. 00_B No moving average is selected. Each intermediate result is considered as final result value: FRRn.FR = CRRn.CR 01_B A moving average of 2 values is selected. The final result is calculated by 2 values: FRRn.FR = CRRn.CR + IRR1n.IR 10_B A moving average of 3 values is selected. The final result is calculated by 3 values: FRRn.FR = CRRn.CR + IRR1n.IR + IRR2n.IR 11_B A moving average of 4 values is selected. The final result is calculated by 4 values: FRRn.FR = CRRn.CR + IRR1n.IR + IRR2n.IR + IRR3n.IR Bit combinations 10_B and 11_B are not available in filter blocks 1 and 3 and must not be selected there.

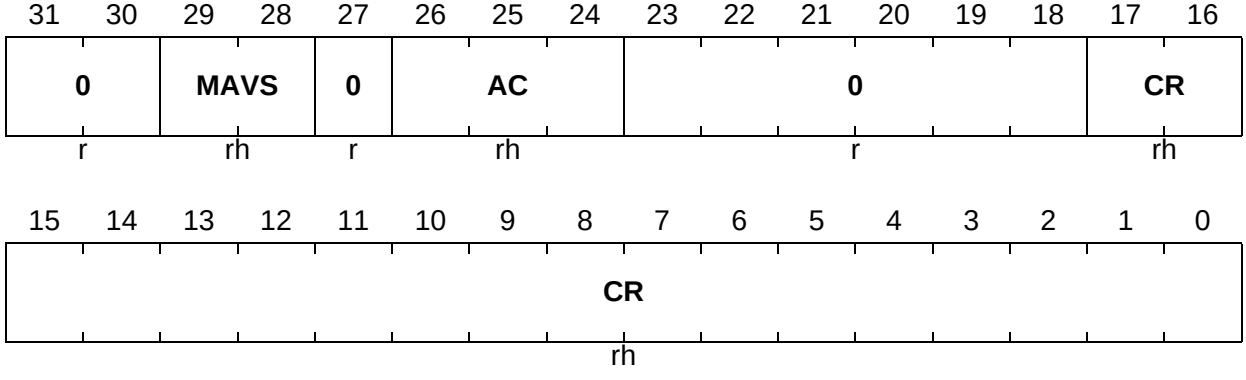
Field	Bits	Type	Description
INSEL	[10:8]	rw	**Input Selection** This bit field enables the filter block and determines which input value is taken for filter block n. For the settings 010_B, set FORM = 00_B. 000_B The filter block is disabled. Intermediate and final sum calculations are not executed. The filter register values are not changed (except by a filter block reset). 001_B Any conversion result of any channel is taken as new filter input value. 010_B Filter block 0: filter is stopped (as 000_B). Filter block 1: filter input value is the output value (final result) of filter block 0. Filter block 2: filter is stopped (as 000_B). Filter block 3: filter input value is the output value (final result) of filter block 2. 011_B Reserved 100_B Channel 0 conversion result is taken as filter input value. 101_B Channel 1 conversion result is taken as filter input value. 110_B Channel 2 conversion result is taken as filter input value. 111_B Channel 3 conversion result is taken as filter input value.
INP	[13:12]	rw	**Interrupt Node Pointer** This bit field selects which service request output line will be activated when a final result of filter block n is available while bit IEN is set. 00_B Service request output SR0 selected 01_B Service request output SR1 selected 10_B Service request output SR2 selected 11_B Service request output SR3 selected
IEN	15	rw	**Interrupt Enable** This bit enables the generation of a new final result service request of filter block n. 0_B Service request generation disabled 1_B Service request generation enabled

Field	Bits	Type	Description
0	3, [7:6], 11, 14, [31:16]	r	**Reserved** Read as 0. Should be written with 0.

24.3.4.2 Current Result Registers

The Current Result Registers CRRn store the current result of filter n. Further, status information of filter block n can be read from CRRn.

CRRn (n = 0-3)
Filter n Current Result Register (064_H + n * 20_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
CR	[17:0]	rh	**Current Result** This bit field (significant bits [12:0] for filters 0 and 2, [17:0] for filters 1 and 3) contains the right-aligned current result value of filter 0. CR is cleared when writing GCR.RSTFn = 1.
AC	[26:24]	rh	**Addition Count** This bit field indicates the number of additions of filter input values with remain to be executed before the next intermediate result register transfer occurs. AC is loaded with the value of FCRn.ADDL for a new addition sequence, also when writing GCR.RSTFn = 1.

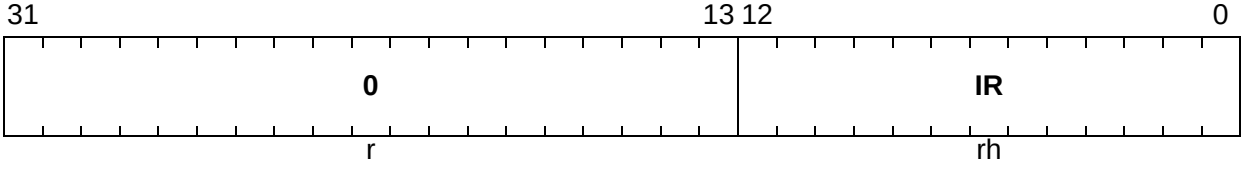
Field	Bits	Type	Description
MAVS	[29:28]	rh	**Moving Average State** This bit field indicates how many intermediate registers transfers remain to be executed for the generation of the next final result. MAVS = 0 indicates the end of a filter calculation operation. Since the filter calculation is executed very fast in comparison to a conversion, MAVS > 0 can be interpreted only as a kind of calculation busy flag. Therefore, it is recommended to read a valid filter result from register FRRn only when the corresponding interrupt request flag CRSR.IRQFn is set. MAVS is reset when writing GCR.RSTFn = 1.
0	[23:18], 27, [31:30]	r	**Reserved** Read as 0. Should be written with 0.

24.3.4.3 Intermediate Result Registers

The Intermediate Result Registers IRRmn hold the intermediate results y of filter n.

IRRy0 (y = 1-3)
Filter 0 Intermediate Result Register y
(064_H+y*4_H) **Reset Value: 0000 0000_H**

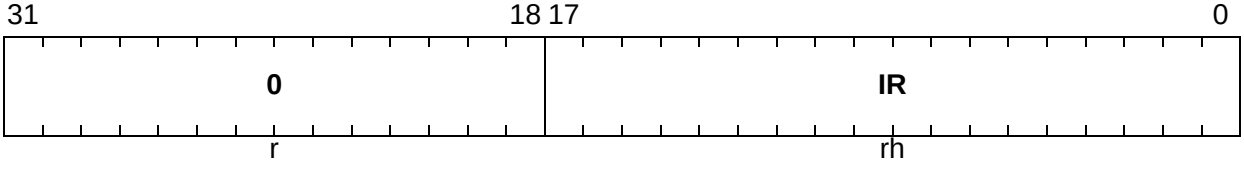
IRRy2 (y = 1-3)
Filter 2 Intermediate Result Register y
($0A4_H$+y*4_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
IR	[12:0]	rh	**Intermediate Result** This bit field contains the right-aligned intermediate result. IR is cleared when writing GCR.RSTFn = 1.
0	[31:13]	rh	**Reserved** Read as 0. Should be written with 0.

IRR11
Filter 1 Intermediate Result Register 1
(088_H) **Reset Value: 0000 0000_H**

IRR13
Filter 3 Intermediate Result Register 1
($0C8_H$) **Reset Value: 0000 0000_H**

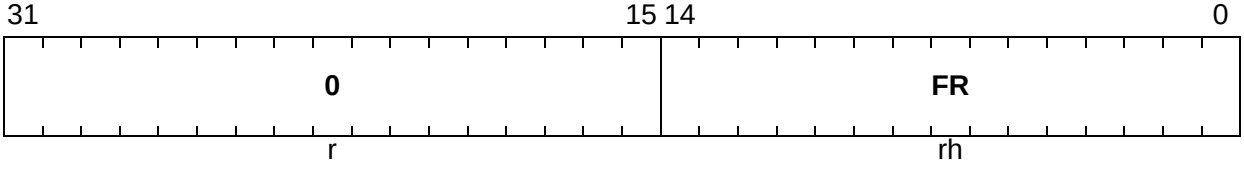


Field	Bits	Type	Description
IR	[17:0]	rh	**Intermediate Result** This bit field contains the right-aligned intermediate result. IR is reset when writing GCR.RSTFn = 1.
0	[31:18]	rh	**Reserved** Read as 0. Should be written with 0.

24.3.4.4 Final Result Registers

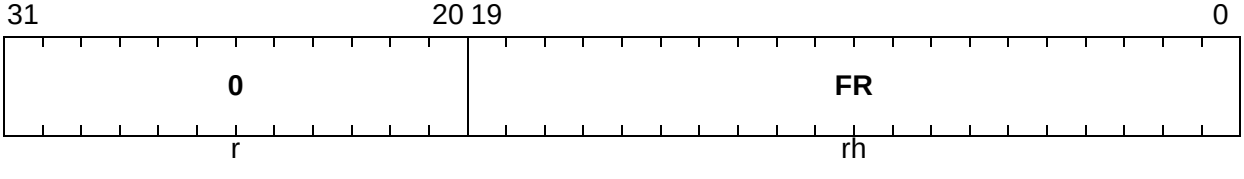
The Final Result Registers FRRn hold the final results of filter block n. The data width being different for the filter block 0 and 2 from the one from data blocks 1 and 3, two different register layouts are necessary.

FRR0
Filter 0 Final Result Register **(074_H)** **Reset Value: 0000 0000_H**
FRR2
Filter 2 Final Result Register **($0B4_H$)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
FR	[14:0]	rh	**Final Result** This bit field contains the right-aligned final result. FR is cleared when writing GCR.RSTFn = 1.
0	[31:15]	rh	**Reserved** Read as 0. Should be written with 0.

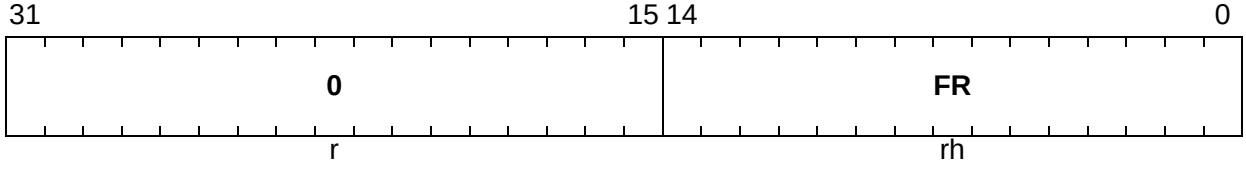
FRR1
Filter 1 Final Result Register **(094_H)** **Reset Value: 0000 0000_H**
FRR3
Filter 3 Final Result Register **($0D4_H$)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
FR	[19:0]	rh	**Final Result** This bit field contains the right-aligned final result. FR is cleared when writing GCR.RSTFn = 1.
0	[31:20]	rh	**Reserved** Read as 0. Should be written with 0.

The Shifted Final Result Registers SFRRn hold the final results of filter blocks 1 and 3 that are shifted right by 5 bit positions. The data representation allows the use of 16-bit data operations for further treatment.

SFRR1
Filter 1 Shifted Final Result Register (098_H) **Reset Value: $0000\ 0000_H$**
SFRR3
Filter 3 Shifted Final Result Register ($0D8_H$) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
FR	[14:0]	rh	**Final Result** This bit field contains the right-aligned final result from the corresponding final result register FRRn shifted right by 5 bit positions. FR is cleared when writing GCR.RSTFn = 1.
0	[31:15]	rh	**Reserved** Read as 0. Should be written with 0.

24.4 Implementation of FADC

This section describes the implementation of the FADC module in the TC1797.

24.4.1 Register Overview

All FADC kernel register names described in this section are referenced in other parts of the TC1797 User´s Manual by the module name prefix “FADC_”.

Table 24-7 Registers Address Space - FADC Module

Module	Base Address	End Address	Note
FADC	$F010\ 0400_H$	$F010\ 05FF_H$	-

Table 24-8 Registers Overview

Register Short Name	Register Long Name	Offset Address	Page Number
intentionally left blank, please refer to register table in **Section 24.3**		H	

24.4.2 Interfaces of the FADC Module

Figure 0-4 shows the TC1797 specific implementation details and interconnections of the FADC module.

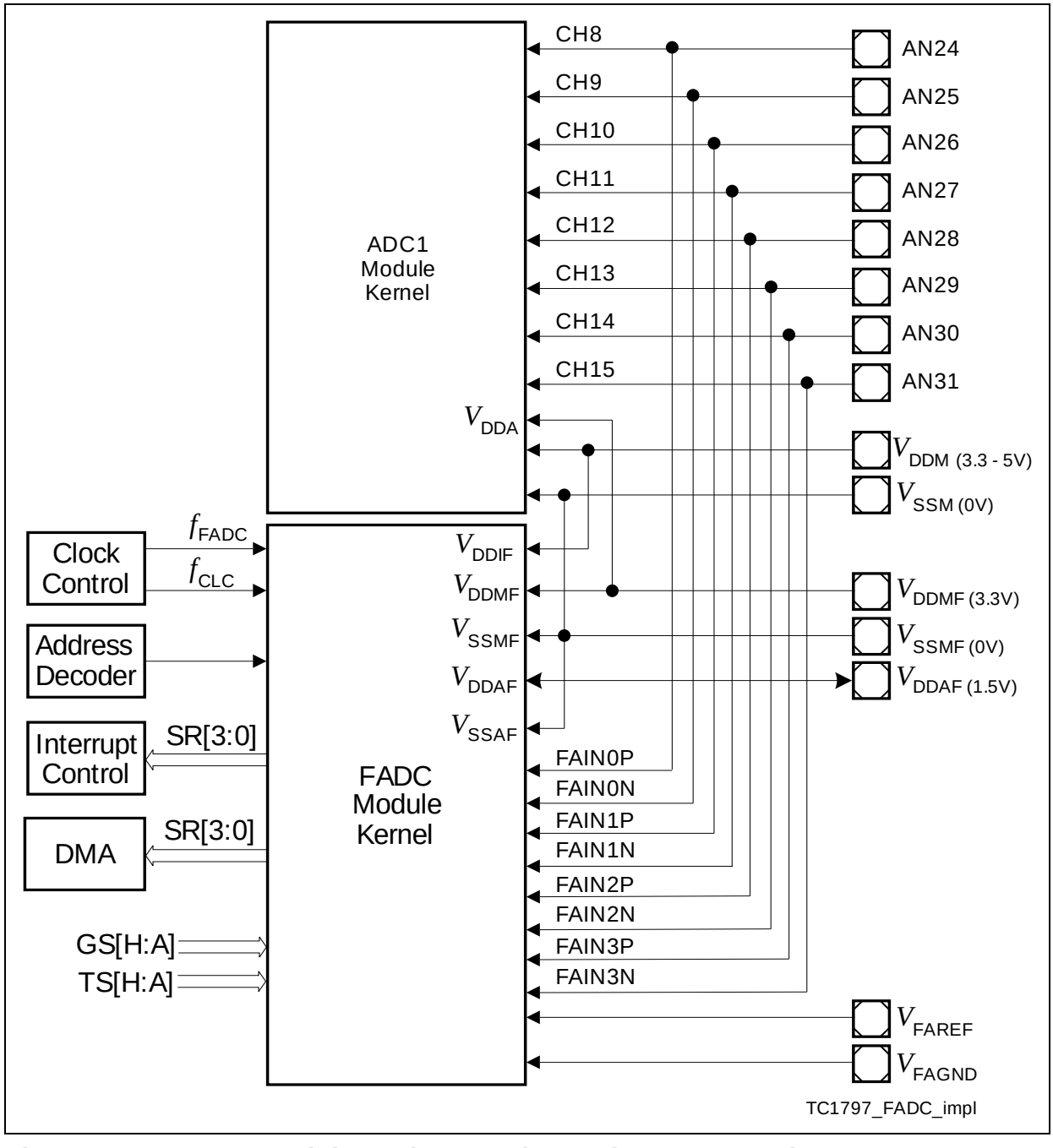


Figure 24-15 FADC Module Implementation and Interconnections

24.4.3 FADC Connections

The following table shows the analog connections of the FADC kernel with other modules or pins in the TC1797 device.

Table 24-9 Connections to FADC Analog Part in TC1797

FADC Signal of Analog Part	from/to Module or Pin	Input or Output	Can be used to/as
V_{DDIF}	V_{DDM}	I	analog power supply 3 V - 5.5 V of input stage, connected to the power supply of the ADC
V_{DDMF}	V_{DDMF}	I	analog power supply 3.3 V, connected also to ADCx_V_{DDA}
V_{SSMF}	V_{SSMF}	I	analog power ground
V_{DDAF}	V_{DDAF}	I	analog power supply 1.5V
V_{SSAF}	V_{SSMF}	I	analog power ground, connected to V_{SSMF} pin
V_{FAREF}	V_{FAREF}	I	positive analog reference
V_{FAGND}	V_{FAGND}	I	negative analog reference
FAIN0P	AN24	I	analog input P channel 0, overlaid with ADC1 channel 8
FAIN0N	AN25	I	analog input N channel 0, overlaid with ADC1 channel 9
FAIN1P	AN26	I	analog input P channel 1, overlaid with ADC1 channel 10
FAIN1N	AN27	I	analog input N channel 1, overlaid with ADC1 channel 11
FAIN2P	AN28	I	analog input P channel 2, overlaid with ADC1 channel 12
FAIN2N	AN29	I	analog input N channel 2 overlaid with ADC1 channel 13
FAIN3P	AN30	I	analog input P channel 3 overlaid with ADC1 channel 14
FAIN3N	AN31	I	analog input N channel 3 overlaid with ADC1 channel 15

The following table shows the digital connections of the FADC kernel with other modules or pins in the TC1797 device.

Table 24-10 Connections of FADC Digital Part in TC1797

FADC Signal of Digital Part	from/to Module or Pin	Input or Output	Can be used to/as
Gating Inputs			
GSA	REQ0	I	P1.0
GSB	REQ4	I	P7.0
GSC	PDOUT2	I	ERU
GSD	PDOUT3	I	ERU
GSE	TRIG11	I	GPTA
GSF	TRIG13	I	GPTA
GSG	TRIG15	I	GPTA
GSH	TRIG17	I	GPTA
Trigger Inputs			
TSA	REQ1	I	P1.1
TSB	REQ5	I	P7.1
TSC	IOUT2	I	ERU
TSD	IOUT3	I	ERU
TSE	TRIG00	I	GPTA
TSF	TRIG02	I	GPTA
TSG	TRIG04	I	GPTA
TSH	TRIG06	I	GPTA
Others			
FADC_SR[3:0]	interrupt controller, DMA	O	service request output lines of FADC (service request)

24.4.4 Service Request Connections

The FADC kernel provides 4 service request output lines FADC_SR[3:0]. All 4 lines are connected to the DMA (for more details, refer to DMA chapter).

Table 24-11 FADC Service Request Connections in TC1797

Service Request Signal	Connected to Service Request Node
FADC_SR[0]	FADC_SRC0
FADC_SR[1]	FADC_SRC1
FADC_SR[2]	FADC_SRC2
FADC_SR[3]	FADC_SRC3

24.4.5 Clock Control

The FADC module is provided with two clock signals:

- f_{CLC}
  This is the module clock that is used inside the FADC kernel for control purposes such as clocking of control logic and register operations. The frequency of f_{CLC} is always identical to the system clock frequency f_{SYS} (= f_{FPI}). The clock control register FADC_CLC makes it possible to enable/disable f_{CLC}.
- f_{FADC}
  This clock is the module clock that is used in the FADC as the clock for the channel timer and other internal timings, such as the conversion timing. The fractional divider registers FADC_FDR controls the frequency of f_{FADC} and allows it to be enabled/disabled independently of f_{CLC}.

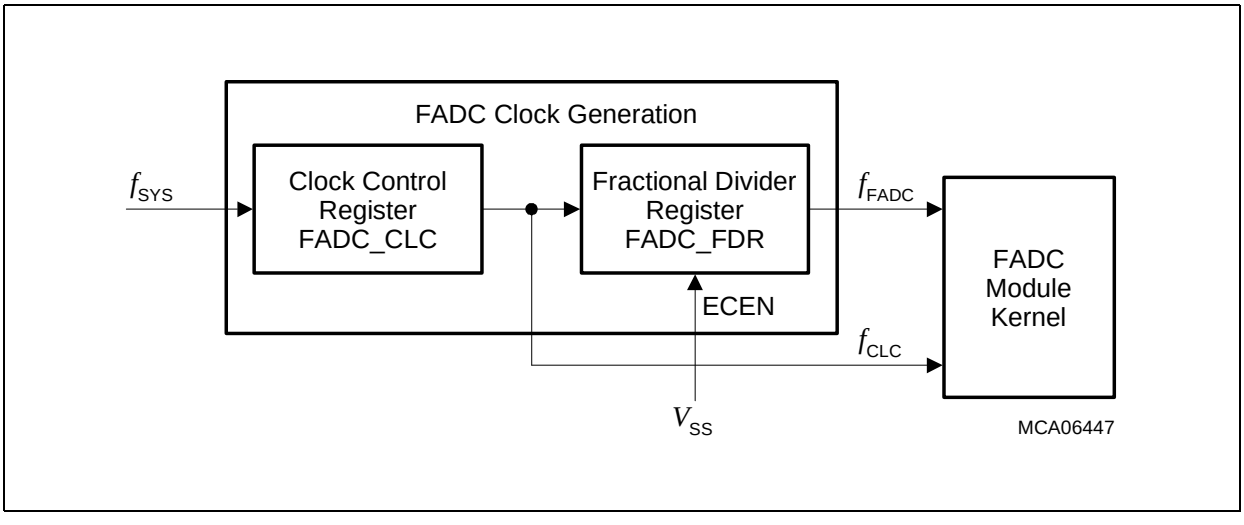


Figure 24-16 FADC Clock Generation

The following formulas define the frequency of f_{FADC}:

$$f_{FADC} = f_{SYS} \times \frac{1}{n} \quad \text{with n = 1024 - FDR.STEP} \tag{24.2}$$

$$f_{FADC} = f_{SYS} \times \frac{n}{1024} \quad \text{with n = 0-1023} \tag{24.3}$$

Equation (24.2) is valid for FADC_FDR.DM = 01_B (normal divider mode).

Equation (24.3) is valid for FADC_FDR.DM = 10_B (fractional divider mode).

Keyword Index

This section lists a number of keywords which refer to specific details of the TC1797 in terms of its architecture, its functional units, or functions. Bold page number entries identify the main definition material for a topic. The “Keyword Index” refers to page numbers in both parts of the TC1797 User´s Manual, the “System Units” (volume 1 with marking “[1]”) and the “Peripheral Units” (volume 2 with marking “[2]”) parts.

A

B

C

D

E

F

G

I

L

M

N

O

P

R

S

T

U

W

Register Index

This section lists the references to the Special Function Registers of the TC1797. It refers to page numbers in both parts of the TC1797 User´s Manual, the “System Units” (volume 1 with marking “[1]”) and the “Peripheral Units” (volume 2 with marking “[2]”) parts.

A

B

C

D

E

F

G

I

L

M

O

P

R

S

T

W

w w w . i n f i n e o n . c o m

Published by Infineon Technologies AG