TC1766

32-Bit Single-Chip Microcontroller

Volume 1 (of 2): System Units

Volume 2 (of 2): Peripheral Units

Microcontrollers

Never stop thinking

Edition 2007-07

Published by
Infineon Technologies AG
81726 München, Germany

Legal Disclaimer

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”). 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 your 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 your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems 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.

TC1766

32-Bit Single-Chip Microcontroller

Volume 1 (of 2): System Units

Volume 2 (of 2): Peripheral Units

Microcontrollers

Never stop thinking

TC1766 User's Manual
Volume 1 (of 2) System Units & Volume 2 (of 2) Peripheral Units
Revision History: V2.0, 2007-07

Previous Version: V1.1 2005-08

Page	Subjects (major changes since last revision)
TC1766 User's Manual Version 2.0 includes the contents of the TC1766 Documentation Addendum, V1.2, Apr. 2007. Furthermore, the complete document and especially the register description have been updated to allow document-automation process based on XML technologies.	
Volume 1: System Units	
1-10	Package name in **Section 1.2.2** is updated.
1-24	**Figure 1-7** is updated.
1-32	**Figure 1-11** is updated.
1-33	**Figure 1-12** is updated.
1-34	The pad driver class for $\overline{\text{HDRST}}$, $\overline{\text{NMI}}$ and $\overline{\text{PORST}}$ are updated.
1-47	**Table 1-5** is added.
2-10	CPU_ID is added.
2-12	MMU_CON long register name is updated
2-13	CPS_ID is added.
2-14	CPU_SRCn.TOS bit description is updated.
2-21	Offset addresses in **Table 2-6** are updated by absolute addresses.
2-24	**Figure 2-11** is updated.
2-28, **2-29**	PMI_ID is added.
2-35, **2-36**	DMI_ID is added.
3-19	Sequence of setting up the PLL after reset is updated.
4-25	**Section 4.4.3** is updated.
4-25	CRC generation and error checking is updated.
5-19 to **5-24**	Bit description of RENx is updated.
5-49	**Figure 5-11** is updated.
5-59, **5-65**	Bit description of **ENON** and **PARAV** are updated.
5-61, **5-66**	SCU_ID is added.
6-4	**Figure 6-2** is updated.
6-7, **6-8**	LBCU_ID are added.

TC1766 User's Manual
Volume 1 (of 2) System Units & Volume 2 (of 2) Peripheral Units
Revision History: V2.0, 2007-07

6-9	Bit description of **LEC** is updated.
6-16, 6-17	LFI_ID is added.
6-22	**Figure 6-7** is updated.
6-32	BCU breakpoint Trigger Combination Logic in **Figure 6-12** is updated
6-35, 6-37	SBCU_ID is added.
7-16	DFlash address of the Load Page Buffer Command is updated.
7-40, 7-41	FLASH_ID is added.
7-42	Bit description of **FABUSY** is updated.
7-50	Register description of **FLASH_FCON** is updated.
7-65, 7-66	PMU_ID is added.
8-5 to 8-13	Address range of some reserved areas in Segment 8 and 10 of **Table 8-2** and **Table 8-4** are updated.
8-17	Double-word access in PCP memories of **Table 8-5** is updated.
9-28	Description of reserved and user-configurable Port 0 pins is updated.
9-35, 9-44	Reset values for **P1_IOCR12** and **P2_IOCR12** are updated.
9-46	Alternate output function of Port 2 pins 1, 8 and 9; Port 3 pins 5, 6, 7 and 8 are updated.
10-52, 10-54	PCP_ID is added.
10-59	Bit 5 of **PCP_ES** is updated.
10-74	Counter Reload Value (COPY) is updated in **Table 10-13**.
10-78	Data Transfer block of **Figure 10-14** is updated.
10-103	The syntax description of **ST.PI** is updated.
11-9	**Figure 11-5** is updated.
11-10	Channel Request Control of **Figure 11-6** is updated.
11-16	The behaviour when CHRST.CH0n is set to 1 of **Section 11.1.4.5** is updated.
11-29	**Figure 11-20** is updated.
11-30	**Figure 11-21** is updated.
11-34	The description for pattern detection in **Section 11.1.9** is updated .

TC1766 User's Manual
Volume 1 (of 2) System Units & Volume 2 (of 2) Peripheral Units
Revision History: V2.0, 2007-07

11-43, **11-44**, **11-46**	DMA_ID is added.
11-50	Bit description **CH0x** and **HTRE0x** are updated.
11-88	Address range for **AEN21** is updated.
11-92	DMA interrupt registers in **Figure 11-31** is updated.
11-101, **11-102**	MCHK_ID is added.
12-26	NMI trap handler routine for parity error handling is updated.
13-7, **13-8**, **13-11**	STM_ID is added.
14-3	The description for handling of ENDINIT -protected registers is updated.
16-15	The access modes for some reserved areas in **Table 16-7** are updated.
16-20 to **16-25**	The write access mode for Px_OMR registers in **Table 16-9** are updated.
16-76	The short names, description and reset values for some reserved areas in **Table 16-26** are updated.
Volume 2: Peripheral Units	
17-19, **17-21**	ASC0_ID and ASC1_ID register are added.
17-31	Transmit DMA request in **Figure 17-12** is updated.
18-15	Configuration of bit CON.PO in **Figure 18-8** is updated.
18-15	TB write operation in **Section 18.1.2.8** is updated.
18-19 to **18-27**	SSC error interrupt control in **Section 18.1.2.9**, bit description **STIP**, **EN** are updated.
18-22, **18-24**	SSC0_ID and SSC1_ID registers are added.
18-31, **18-32**	Register description **SSOC** and **SSOTC** are updated.
19-20	**Section 19.1.2.6** is updated.
19-36, **19-38**	MSC0_ID register is added.

TC1766 User's Manual
Volume 1 (of 2) System Units & Volume 2 (of 2) Peripheral Units
Revision History: V2.0, 2007-07

19-42	Bit description NDBH is updated.
20-49, 20-50	Usage of bit RXEN in Section 20.3.11.5 and transmit FIFO in Section 20.3.11.6 are updated.
20-54, 20-55, 20-56, 20-57	CAN_ID register is added.
20-54 to 20-75	Offset address in Table 20-4 and CPU write to LEC in Table 20-7 are updated.
20-86	Bus state of NewBit in Table 20-9 is updated.
20-110	Configuration of CAN_FDR.DM for Equation (20.1) and Equation (20.2) are updated.
20-118	Interrupt output lines in Section 20.5.5 is updated.
Chapter 21	The functional description of the MLI , the module kernel description, and operation the MLI module is completely reworked.
21-2	Programmable baud rate in Section 21.1.1.1 is updated.
21-75, 21-76, 21-78	MLI0_ID and MLI1_ID registers are added.
21-115	Register description RPxBAR (x = 0-3) is updated.
21-129	Equation (21.6) is added.
21-134 to 21-136	P5_IOCR8, P5_IOCR12 and P5_PDR are updated.
21-141	Selected address range for AEN21 corrected.
22-30, 22-31, 22-32, 22-33	Figure 22-19 to Figure 22-22 are updated.
22-60 to 22-76	Cell Enabling on Event for GTC and LTC are updated.
22-75	Table 22-4 is updated.
22-89 to 22-221	Seven I/O and seven output groups with 56 I/O and 56 output lines in Section 22.2.4, Section 22.4.3 and Section 22.4.4 are updated.

TC1766 User's Manual
Volume 1 (of 2) System Units & Volume 2 (of 2) Peripheral Units
Revision History: V2.0, 2007-07

22-125, 22-131	"Check_Input()" in Section 22.2.6.3 and "Manage_Mux()" in Section 22.2.6.6 are updated.
22-150, 22-151, 22-154	GPTA0_ID register is added.
22-218	Table 22-20 is updated.
22-224	Module clock generation is updated.
23-9	Equation (23.1) is updated.
23-49, 23-50, 23-52	ADC0_ID register is added.
23-95	The reference voltage selection is updated.
24-8	Equation (24.1) is updated.
24-28, 24-29, 24-30	FADC_ID register is added.

Trademarks

TriCore® is a trademark of Infineon Technologies AG.

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 in both parts of the TC1766 User’s Manual, the “System Units” (volume 1 with marking “[1]”) and the “Peripheral Units” (volume 2 with marking “[2]”) parts.

Volume 1 (of 2): System Units

Volume 2 (of 2): Peripheral Units

1 Introduction

This User's Manual describes the Infineon TC1766, 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 TC1766 functional units, registers, instructions, and exceptions.

This TC1766 User's Manual describes the features of the TC1766 with respect to the TriCore Architecture. Where the TC1766 directly implements TriCore architectural functions, this manual simply refers to those functions as features of the TC1766. In all cases where this manual describes a TC1766 feature without referring to the TriCore Architecture, this means that the TC1766 is a direct implementation of the TriCore Architecture.

Where the TC1766 implements a subset of TriCore architectural features, this manual describes the TC1766 implementation, and then describes how it differs from the TriCore Architecture. For example, where the TriCore Architecture specifies up to four Memory Protection Register Sets, the TC1766 implements only two. Such differences between the TC1766 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 TC1766 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.

In addition to this "TC1766 System Units (Vol. 1 of 2) User's Manual", a second document, the "TC1766 Peripheral Units (Vol. 2 of 2) User's Manual", is available. These two User's Manuals together with the "TriCore Architecture Manual" are required to understand the complete TC1766 microcontroller functionality.

Implementation-specific details of the TC1766 such as electrical characteristics and timing parameters are defined in the "TC1766 Data Sheet".

1.1.2 Text Conventions

This document uses the following text conventions for named components of the TC1766:

- Functional units of the TC1766 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, $\overline{\text{HDRST}}$, has a dual function.”.
- Bit fields and bits in registers are in general referenced as “Register name.Bit field” or “Register name.Bit”. For example: “The Current CPU Priority Number bit field 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 of signals, 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** = 1024 bytes of memory
  - **Mbyte** = 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. • Writing these bit fields has no effect. These bit fields are reserved. When writing, software should always set such bit fields to 0 in order to preserve compatibility with future products.
Undefined, Reserved	Certain bit combinations in a bit field can be marked “Reserved”, indicating that the behavior of the TC1766 is undefined for that combination of bits. Setting the register to such undefined bit or bit field combinations may lead to unpredictable results. Such bit combinations are reserved. When writing, software must always set such bit fields to legal values as defined for it.
rw	The bit or bit field can be read and written.
rwh	As rw, but bit or bit field can be also set or cleared 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).
rh	The bit or bit field can only be read. It can be also set or cleared by hardware (typical example: status flags).
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.
X	Undefined value or bit.

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
DFLASH	Data Flash Memory
DGPR	Data General Purpose Register
DMA	Direct Memory Access
DMI	Data Memory 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

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
NMI	Non-Maskable Interrupt
OCDS	On-Chip Debug Support
OVRAM	Code Overlay Memory
PCP	Peripheral Control Processor
PMU	Program Memory Unit
PLL	Phase Locked Loop
PCODE	PCP Code Memory
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 TC1766

The TC1766 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 TC1766 include:

- Program Memory Unit – instruction memory and instruction cache
- 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 TC1766 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 TC1766 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 TC1766 offers several versatile on-chip peripheral units such as serial controllers, timer units, and Analog-to-Digital converters. Within the TC1766, all these peripheral units are connected to the TriCore CPU/system via the Flexible Peripheral Interconnect (FPI) Bus and two simplified bus interfaces. Several I/O lines on the TC1766 ports are reserved for these peripheral units to communicate with the external world.

1.2.1 TC1766 Block Diagram

Figure 1-1 shows the block diagram of the TC1766.

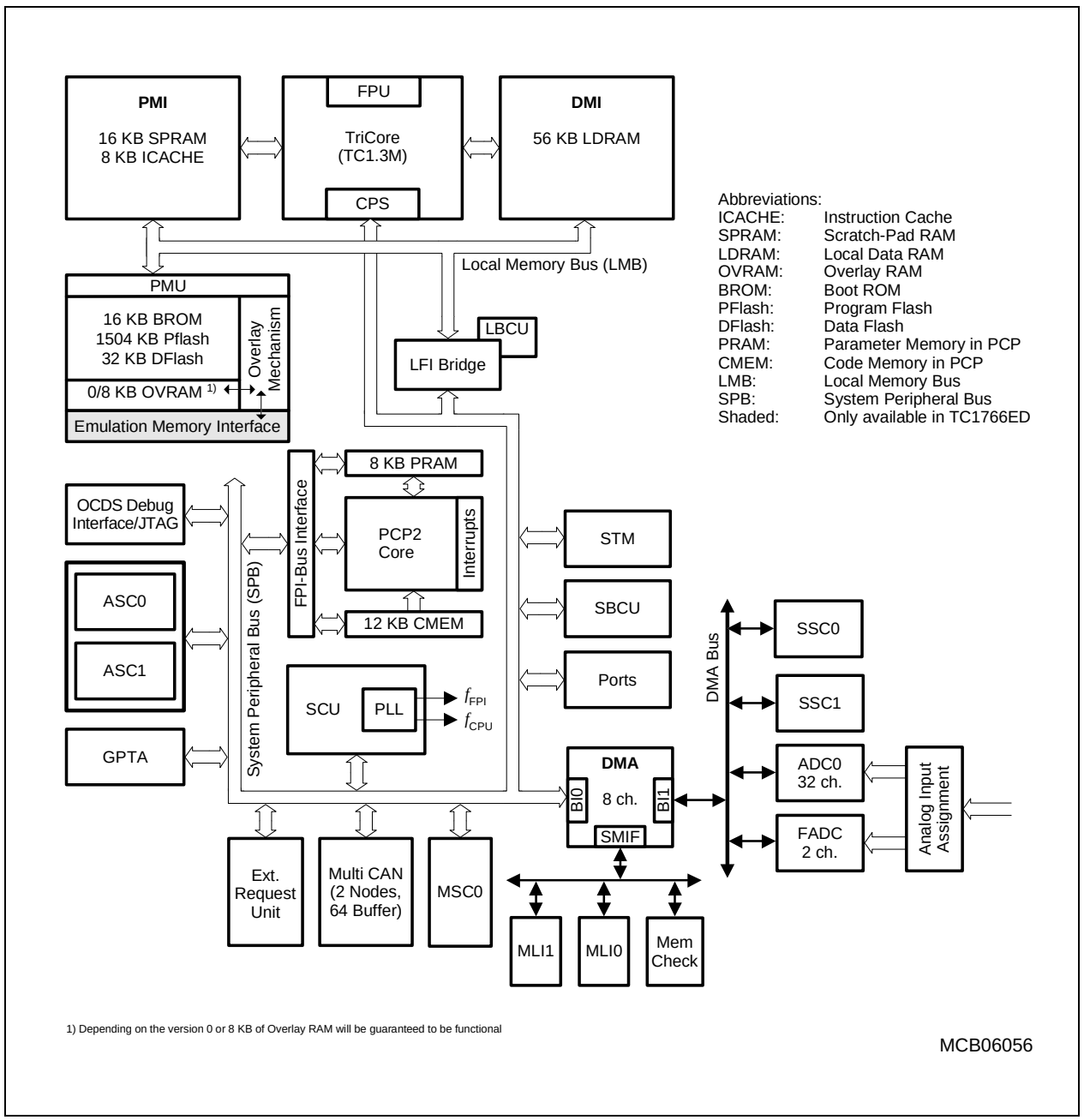


Figure 1-1 TC1766 Block Diagram

1.2.2 Features

The TC1766 has the following features:

High-performance 32-bit CPU

- 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 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 On-Chip Memories

- Code memory
  - 1504 Kbyte on-chip Program Flash (PFLASH)
  - 16 Kbyte Scratch-Pad RAM (SPRAM)
  - 8 Kbyte Instruction Cache (ICACHE)
  - 16 Kbyte Boot ROM (BROM)
- Data memory
  - 56 Kbyte Local Data RAM (LDRAM)
  - 32 Kbyte on-chip Data Flash (DFLASH)
- PCP
  - 12 Kbyte Code Memory (CMEM)
  - 8 Kbyte Parameter Memory (PRAM)
- On-chip SRAMs with parity error detection

Interrupt System

- Total of 103 Service Request Nodes (SRNs)
- Flexible interrupt-prioritizing scheme with 255 interrupt priority levels
- Fast interrupt response
- Service requests are serviced by CPU or PCP

Peripheral Control Processor (PCP)

- Data move between any two memory or I/O locations
- Data move until predefined limit reached 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)

DMA Controller

- 8 independent DMA channels
- Programmable priority of the DMA sub-blocks on the bus interfaces
- Buffer capability for move actions on the buses (minimum of 1 move per bus is buffered)
- Individually programmable operation modes for each DMA channel
- Full 32-bit addressing capability of each DMA channel
- Programmable data width of DMA transfer/transaction: 8-bit, 16-bit, or 32-bit
- Micro Link bus interface support
- One register set for each DMA channel
- Flexible interrupt generation
- Read/write requests of the System Bus side to the peripherals on DMA Bus are bridged to the DMA Bus, allowing easy access to these peripherals by PCP and CPU.

Parallel I/O Ports

- 81 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/cleared/toggled bit-wise

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
- One Micro Second Bus Interface (MSC) for serial communication
- One CAN Module with two 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
- One General Purpose Timer Array (GPTA) with a powerful set of digital signal filtering and timer functionality to accomplish autonomous and complex Input/Output management
- One medium-speed Analog-to-Digital Converter Unit (ADC) with 8-bit, 10-bit, or 12-bit resolution, possible to connect thirty two analog inputs to its sixteen channels
- One fast Analog-to-Digital Converter Unit (FADC)

Package

- PG-LQFP-176-2 package, 0.5 mm pitch

Temperature Ranges

- Ambient temperature: -40 °C to +125 °C
- Maximum junction temperature: +150 °C

Clock Frequencies

- Maximum CPU clock frequency: 80 MHz
- Maximum system clock frequency: 80 MHz

Complete Development Support

A variety of software and hardware development tools for the 32-bit microcontroller TC1766 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 TC1766 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 TC1766 includes a power management system, a watchdog timer as well as reset logic

1.3 On-Chip Peripheral Units of the TC1766

The TC1766 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 TC1766 ports are reserved for these peripheral units to communicate with the external world.

The peripherals mentioned in this overview section are all described in detail in the chapters of the “TC1766 Peripheral Units (Vol. 2 of 2) User’s Manual”:

1.3.1 Serial Interfaces

The TC1766 includes seven + two (CAN) serial peripheral interface units:

- Two Asynchronous/Synchronous Serial Interfaces (ASC0 and ASC1)
- Two high-speed Synchronous Serial Interfaces (SSC0 and SSC1)
- One Micro Second Channel Interface (MSC0)
- Two Micro Link Serial Bus Interfaces (MLI0 and MLI1) dedicated for the communication between two controllers of the TriCore family
- One MultiCAN Interface with 2 CAN nodes

1.3.1.1 Asynchronous/Synchronous Serial Interfaces

Figure 1-2 shows a global view of the Asynchronous/Synchronous Serial Interface (ASC).

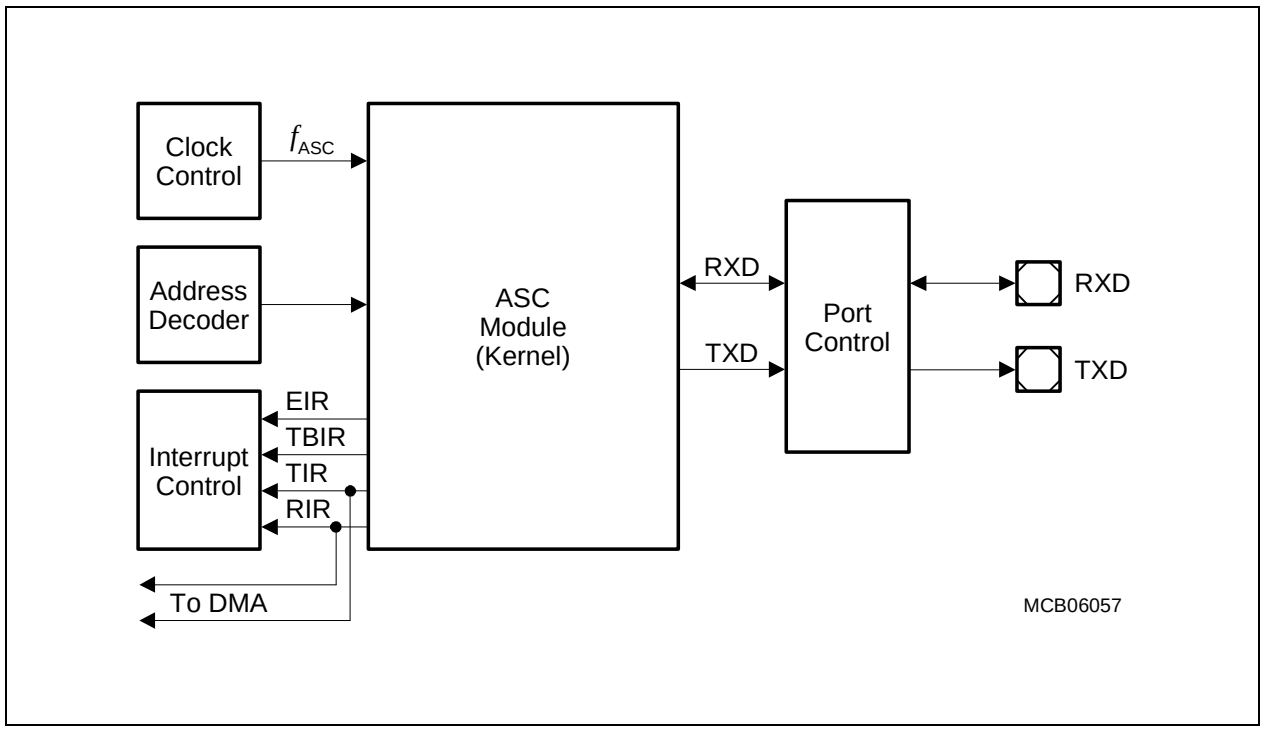


Figure 1-2 General Block Diagram of the ASC Interface

The ASC provides serial communication between the TC1766 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 a fractional divider.

Each ASC module, ASC0 and ASC1, communicates with the external world via two I/O lines. The RXD line is the receive data input signal (in Synchronous Mode also output). TXD is the transmit output signal. Clock control, address decoding, and interrupt service request control are managed outside the ASC module kernel.

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 5.0 Mbit/s to 1.19 bit/s (@ 80 MHz clock)
  - Multiprocessor mode for automatic address/data byte detection
  - Loop-back capability
- Half-duplex 8-bit synchronous operating mode
  - Baud rate from 10.0 Mbit/s to 813.8 bit/s (@ 80 MHz 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)

1.3.1.2 High-Speed Synchronous Serial Interfaces

Figure 1-3 shows a global view of the Synchronous Serial interface (SSC).

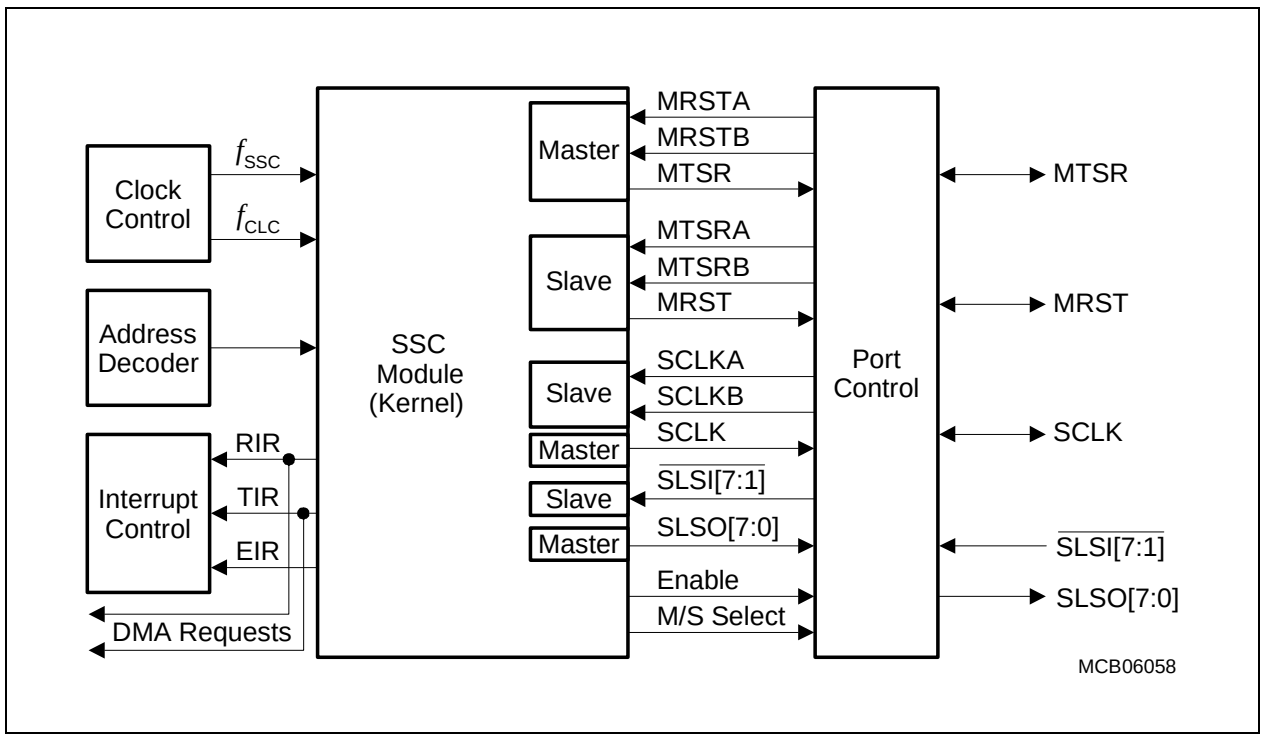


Figure 1-3 General Block Diagram of the SSC Interface

The SSC supports full-duplex and half-duplex serial synchronous communication up to 40 Mbit/s (@ 80 MHz module clock). 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 from 40 Mbit/s to 610.36 bit/s (@ 80 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 in Master Mode
  - Automatic SLSO generation with programmable timing
  - Programmable active level and enable control

1.3.1.3 Micro Second Channel Interface

The 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 1-4** shows a global view of the interface signals of the MSC interface.

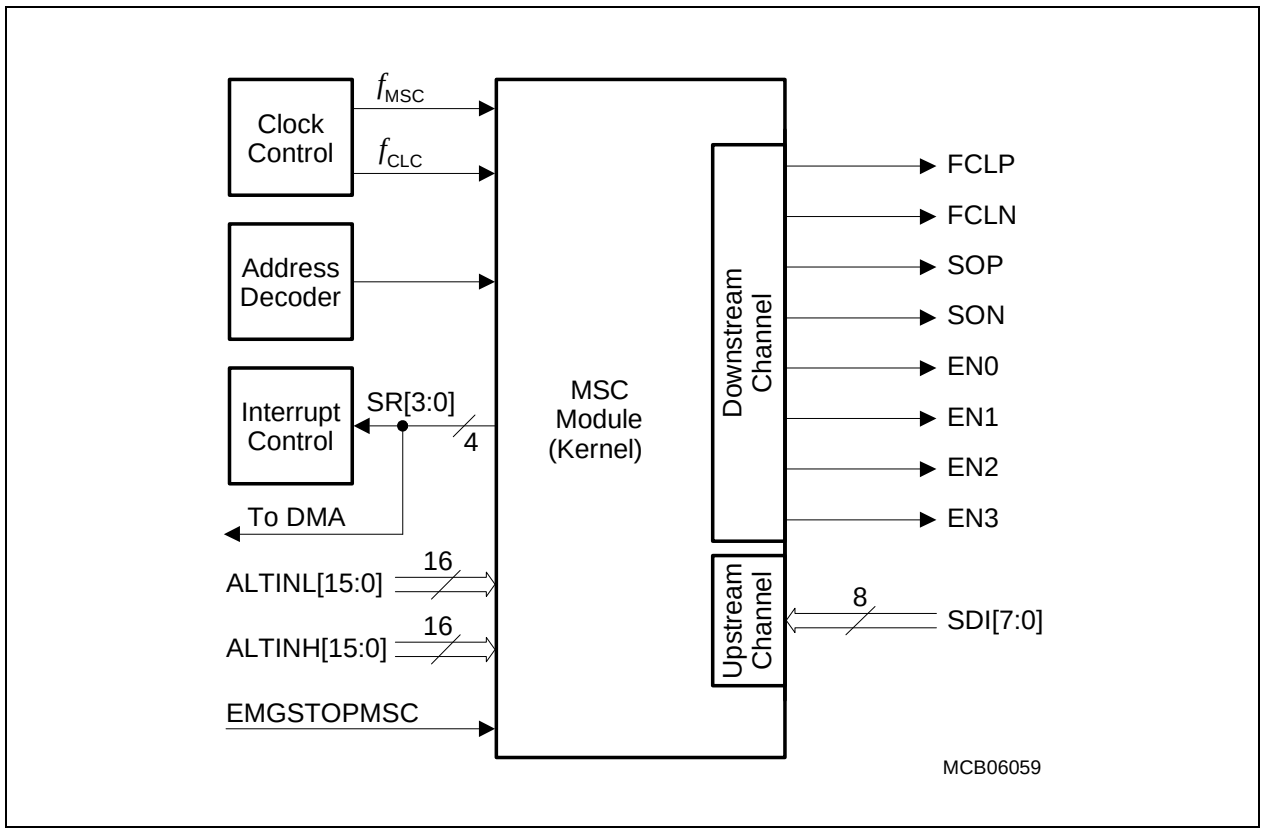


Figure 1-4 General Block Diagram of the MSC Interface

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 especially power switches in particular, or other peripheral devices via serial buses
- High-speed synchronous serial transmission on downstream channel
  - Serial output clock frequency: $f_{FCL} = f_{MSC}/2$
  - 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
  - Standard asynchronous serial frames
  - Parity error checker
  - 8-to-1 input multiplexer for SDI lines
  - Built-in spike filter on SDI lines

1.3.1.4 MultiCAN Controller

The MultiCAN module contains two independent CAN nodes, representing two serial communication interfaces.

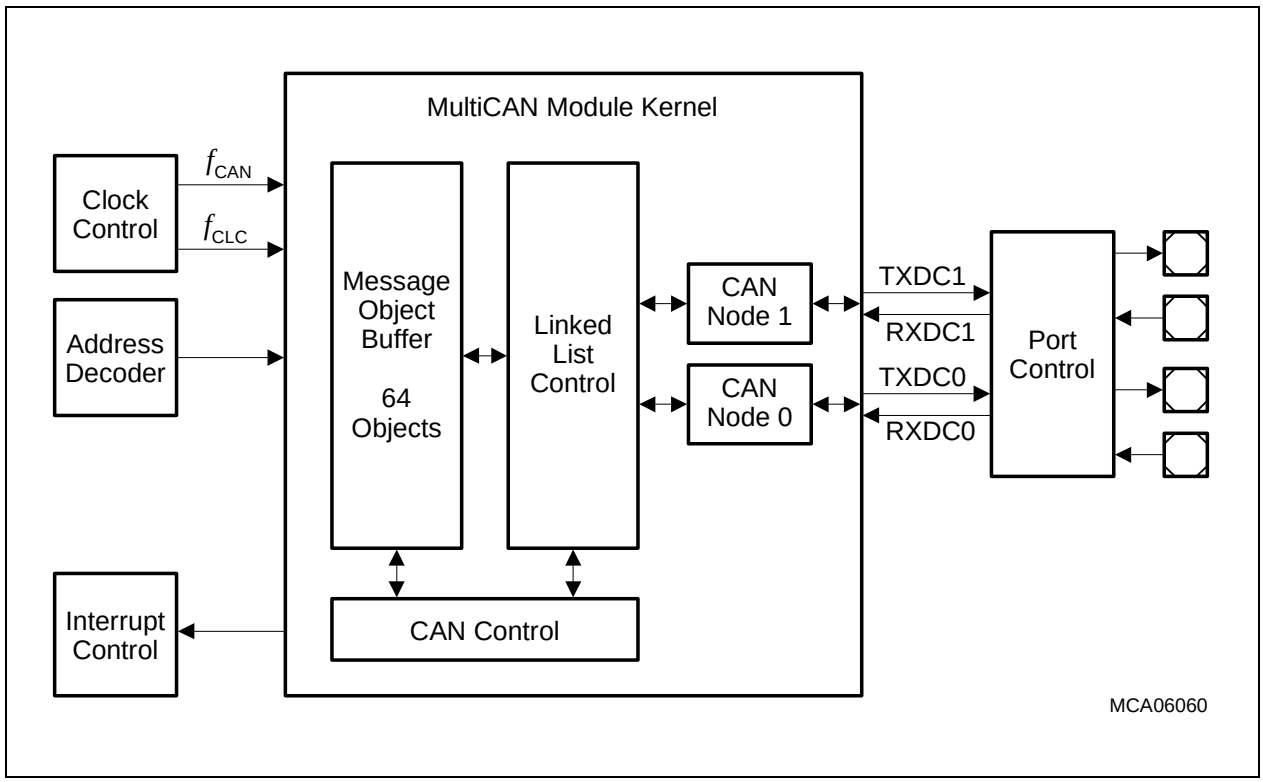


Figure 1-5 Overview of the MultiCAN Module

The MultiCAN module contains two 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 two 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.

CAN 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 64 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.

1.3.1.5 Micro Link Serial Bus Interface

The Micro Link Interface is a fast synchronous serial interface that makes it possible to exchange data between microcontrollers of the 32-bit Audo microcontroller family without intervention of a CPU or other bus masters. **Figure 1-6** shows how two microcontrollers are typically connected together via their MLI interfaces. The MLI operates in both microcontrollers as a bus master on the system bus.

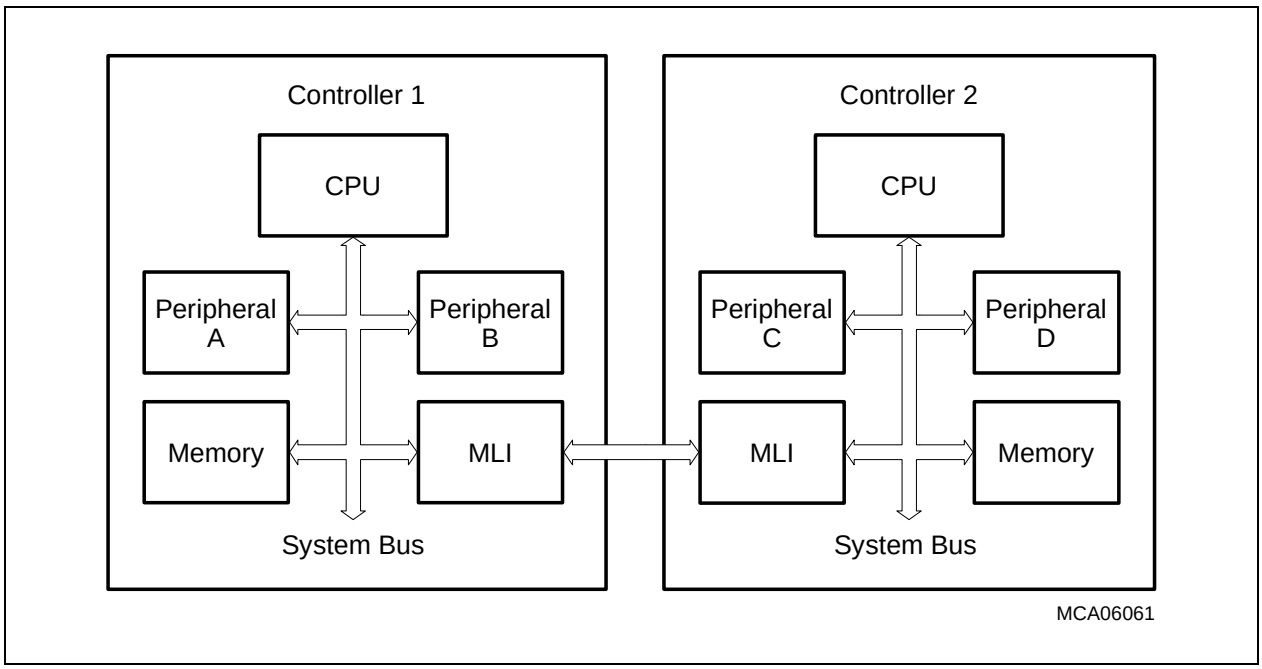


Figure 1-6 Typical Micro Link Interface Connection

Features

- Synchronous serial communication between MLI transmitters and MLI receivers located on the same or on different microcontroller devices
- Automatic data transfer/request transactions between local/remote controller
- Fully transparent read/write access supported (= remote programming)
- Complete address range of remote controller available
- Specific frame protocol to transfer commands, addresses and data
- Error control by parity bit
- 32-bit, 16-bit, and 8-bits data transfers
- Programmable baud rate: max. $f_{MLI}/2$ (= 40 Mbit/s @ 80 MHz module clock)
- Multiple remote (slave) controllers supported

Figure 1-7 shows a general block diagram of the MLI module.

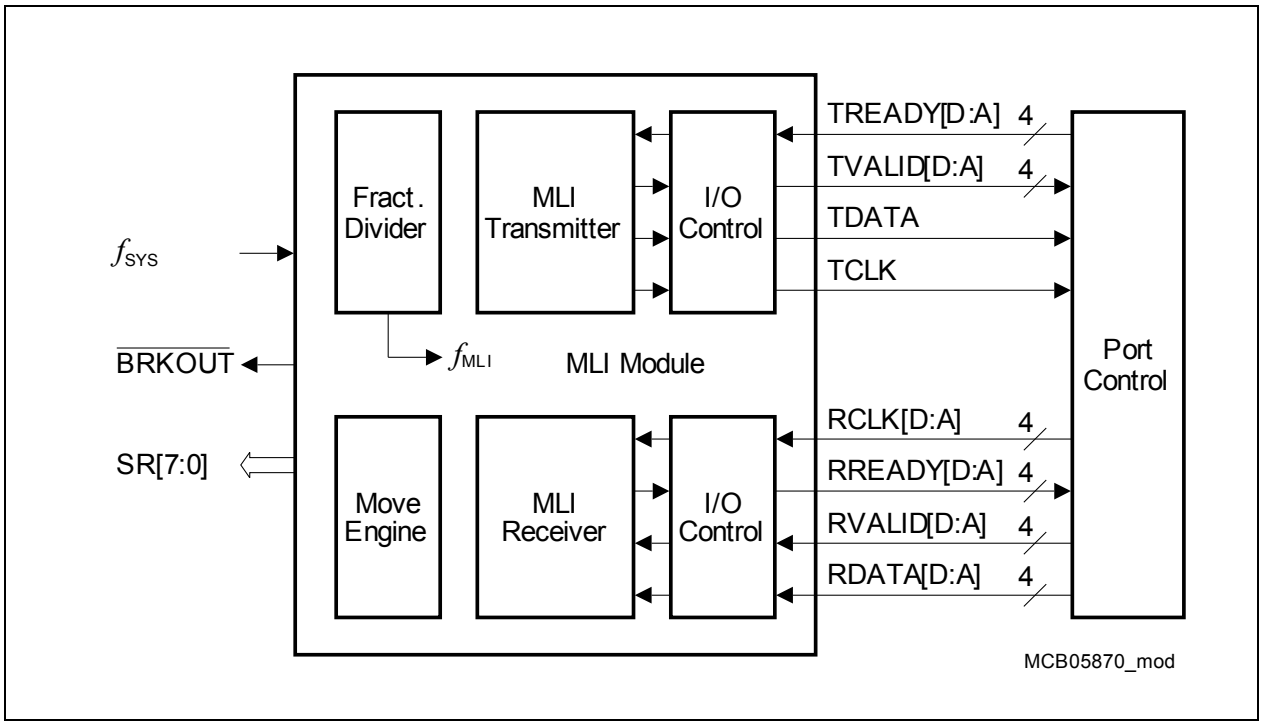


Figure 1-7 General Block Diagram of the MLI Modules

The MLI transmitter and MLI receiver communicate with other off-chip MLI receivers and MLI transmitters via a four-line serial I/O bus. Several I/O lines of these I/O buses are available outside the MLI module kernel as four-line output or input buses with index numbering A, B, C and D. The transmitter signals are named with the prefix “T” and the receiver signals are named with the prefix “R”.

Data read and write operations from/to remote window areas can be handled by a Move Engine that is able to operate as a bus master. Clock control, address decoding, and interrupt service request control are managed outside the MLI module kernel. Eight service request outputs can be used to trigger an interrupt or a DMA request.

1.3.2 General Purpose Timer Array

The TC1766 contains one General Purpose Timer Array (GPTA0). **Figure 1-8** shows a global view of the GPTA module.

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 typically for engine, gearbox, electrical motor control applications, but can be also be used to generate simple and complex signal waveforms needed in other industrial applications.

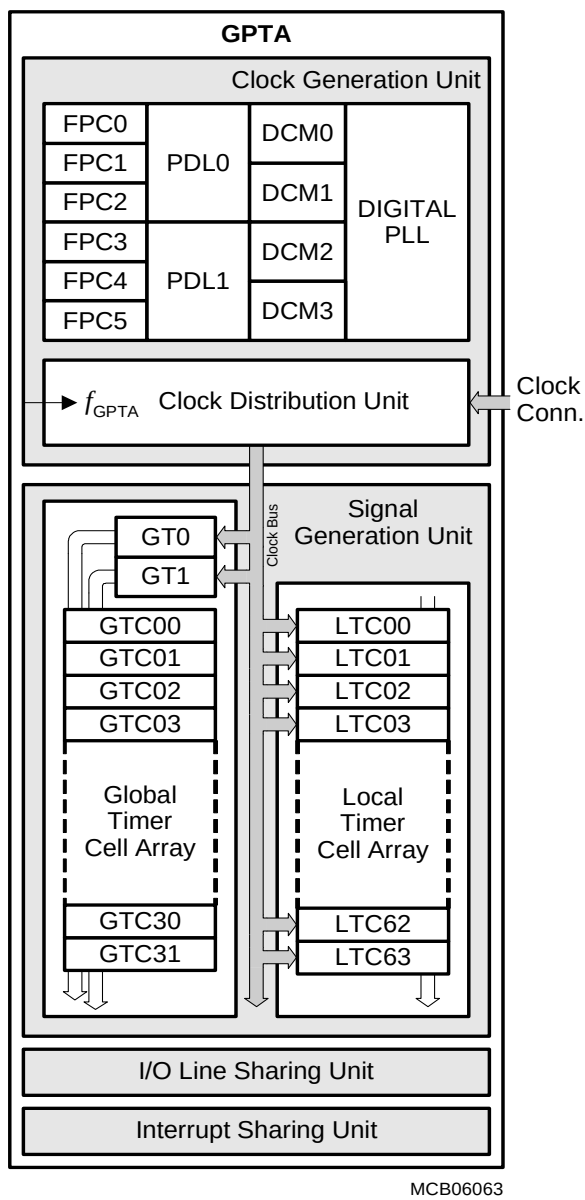


Figure 1-8 General Block Diagram of the GPTA Module

1.3.2.1 Functionality of GPTA0

The General Purpose Timer Arrays (GPTA0) 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.

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 eight clock output signals (clock bus): 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

- 111 interrupt sources, generating up to 38 service requests.

I/O Sharing Unit

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

1.3.3 Analog-to-Digital Converters

The TC1766 contains a medium-speed Analog-to-Digital Converter (ADC0) and a fast Analog-to-Digital Converter (FADC). ADC0 provides 2.5 µs conversion time @ 10-bit resolution and are intended mainly for single-ended signals. They offer a very flexible and comprehensive control for monitoring a large number of relatively slow signals, including synchronous conversion of the ADC.

The FADC offers very fast conversion rates (262.5 ns @ f_{FADC} = 80 MHz) thus allowing sampling of high-frequency signals. For slow and mid-range frequency signal heavy oversampling can be performed and thus expensive filters can be avoided.

1.3.3.1 Analog-to-Digital Converter (ADC)

The on-chip ADC module of the TC1766 is analog-to-digital converters with 8-bit, 10-bit, or 12-bit resolution including sample & hold functionality. The A/D converter operates by the method of successive approximation. A multiplexer selects up to 32 analog inputs that can be connected to the 16 conversion channels in the ADC module. An automatic self-calibration adjusts the ADC module to changing temperatures or process variations.

Features

- 8-bit, 10-bit, 12-bit A/D conversion
- Conversion time below 2.5 µs @ 10-bit resolution
- Extended channel status information on request source
- Successive approximation conversion method
- Total Unadjusted Error (TUE) of ±2 LSB @ 10-bit resolution
- Integrated sample & hold functionality
- Direct control of up to 16 (32) analog input channels per ADC
- Dedicated control and status registers for each analog channel
- Powerful conversion request sources
- Selectable reference voltages for each channel
- Programmable sample and conversion timing schemes
- Limit checking
- Flexible ADC module service request control unit
- Automatic control of external analog multiplexers
- Equidistant samples initiated by timer
- External trigger and gating inputs for conversion requests
- Power reduction and clock control feature

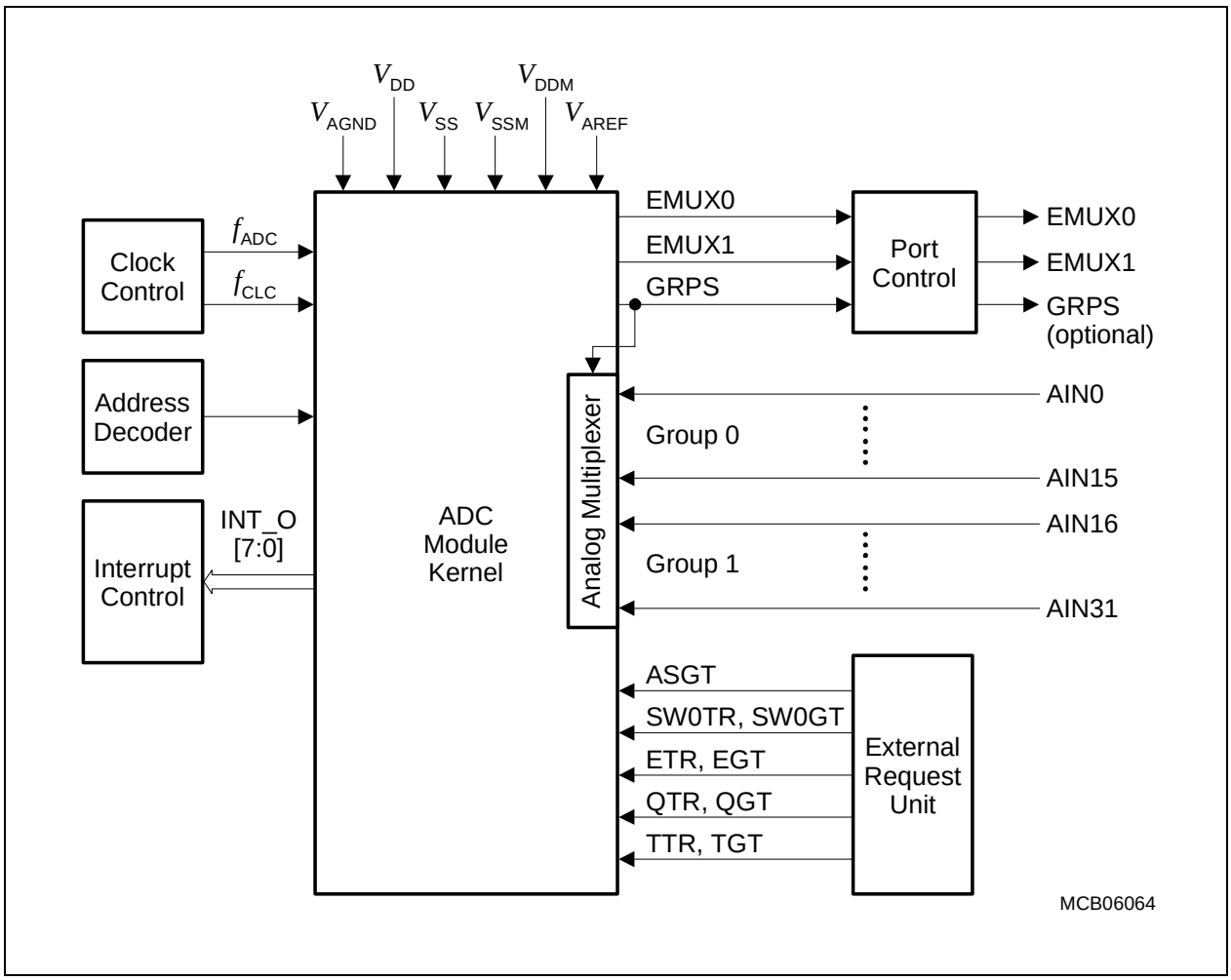


Figure 1-9 General Block Diagram of the ADC Module

The ADC module, ADC0 has 16 analog input channels. An analog multiplexer selects the input line for the analog input channels among the 32 analog inputs. Additionally, an external analog multiplexer can be used for analog input extension. External Clock control, address decoding, and service request (interrupt) control are managed outside the ADC module kernel. External trigger conditions are controlled by an External Request Unit. This unit generates the control signals for auto-scan control (ASGT), software trigger control (SW0TR, SW0GT), the event trigger control (ETR, EGT), queue control (QTR, QGT), and timer trigger control (TTR, TGT).

1.3.3.2 Fast Analog-to-Digital Converter Unit (FADC)

The on-chip FADC module of the TC1766 is primarily a two-channel A/D converter with 10-bit resolution that operates by the method of the successive approximation.

Features

- Extreme fast conversion, 21 cycles of f_{FADC} (262.5 ns @ f_{FADC} = 80 MHz)
- 10-bit A/D conversion
  – Higher resolution by averaging of consecutive conversions is supported
- Successive approximation conversion method
- Two differential input channels
- Offset and gain calibration support for each channel
- Differential input amplifier with programmable gain of 1, 2, 4 and 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 antialiasing and data reduction filter block

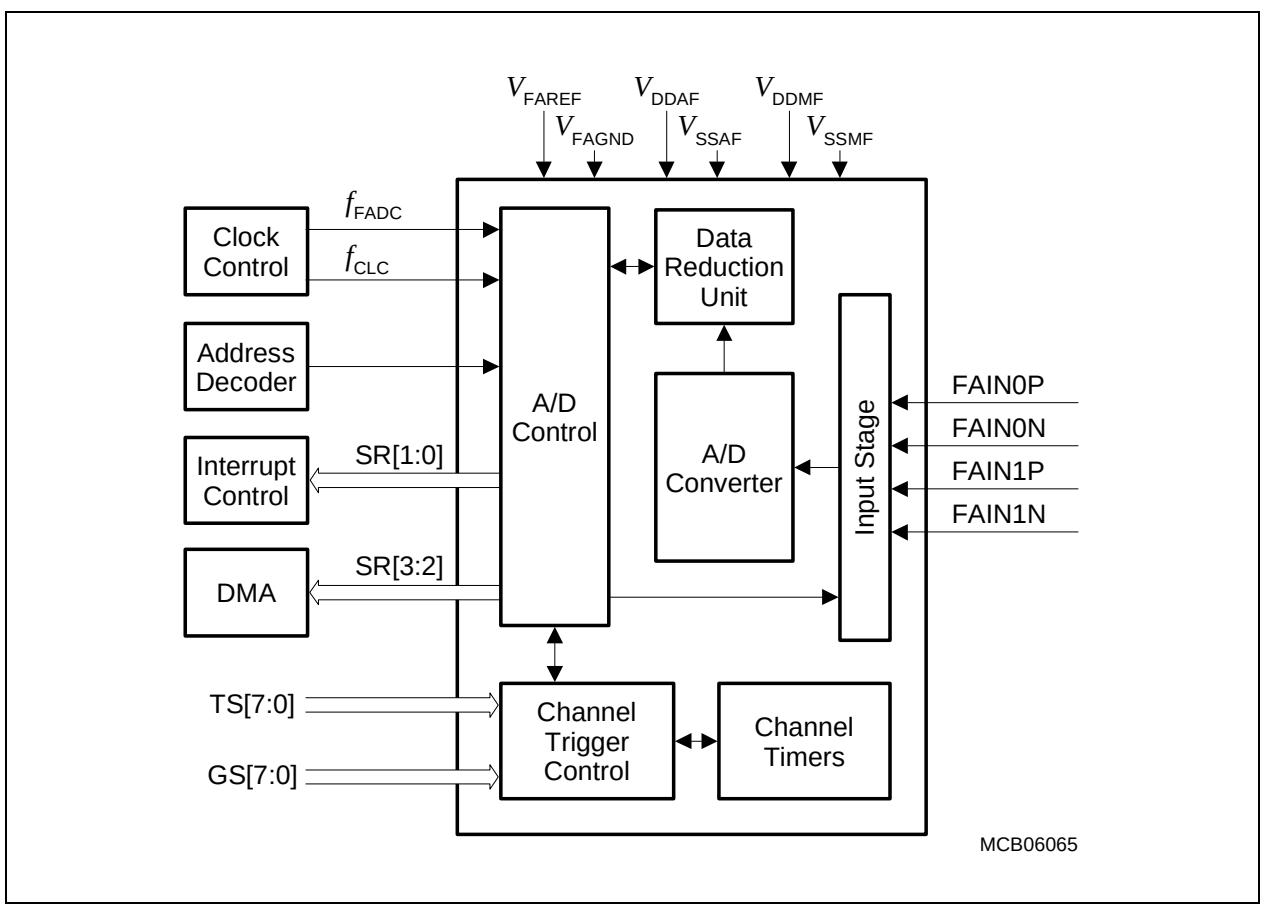


Figure 1-10 Block Diagram of the FADC Module

The main FADC functional blocks (see **Figure 1-10**) are:

- The Input Stage – contains the differential inputs and the programmable amplifier.
- The A/D Converter – is responsible for the analog-to-digital conversion.
- The Data Reduction Unit – contains programmable antialiasing and data reduction filters.
- The Channel Trigger Control block – determines the trigger and gating conditions for the two FADC channels.
- The Channel Timers – can independently trigger the conversion of each FADC channel.
- The A/D control block is responsible for the overall FADC functionality.

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

- V_{DDMF}/V_{DDMF}: FADC Analog Part Power Supply (3.3 V)
- V_{DDAF}/V_{DDAF}: FADC Analog Part Logic Power Supply (1.5 V)
- V_{FAREF}/V_{FAGND}: FADC Reference Voltage (3.3 V)/FADC Reference Ground

1.4 TC1766 Pin Definitions and Functions

Figure 1-11 shows the logic symbol of the TC1766.

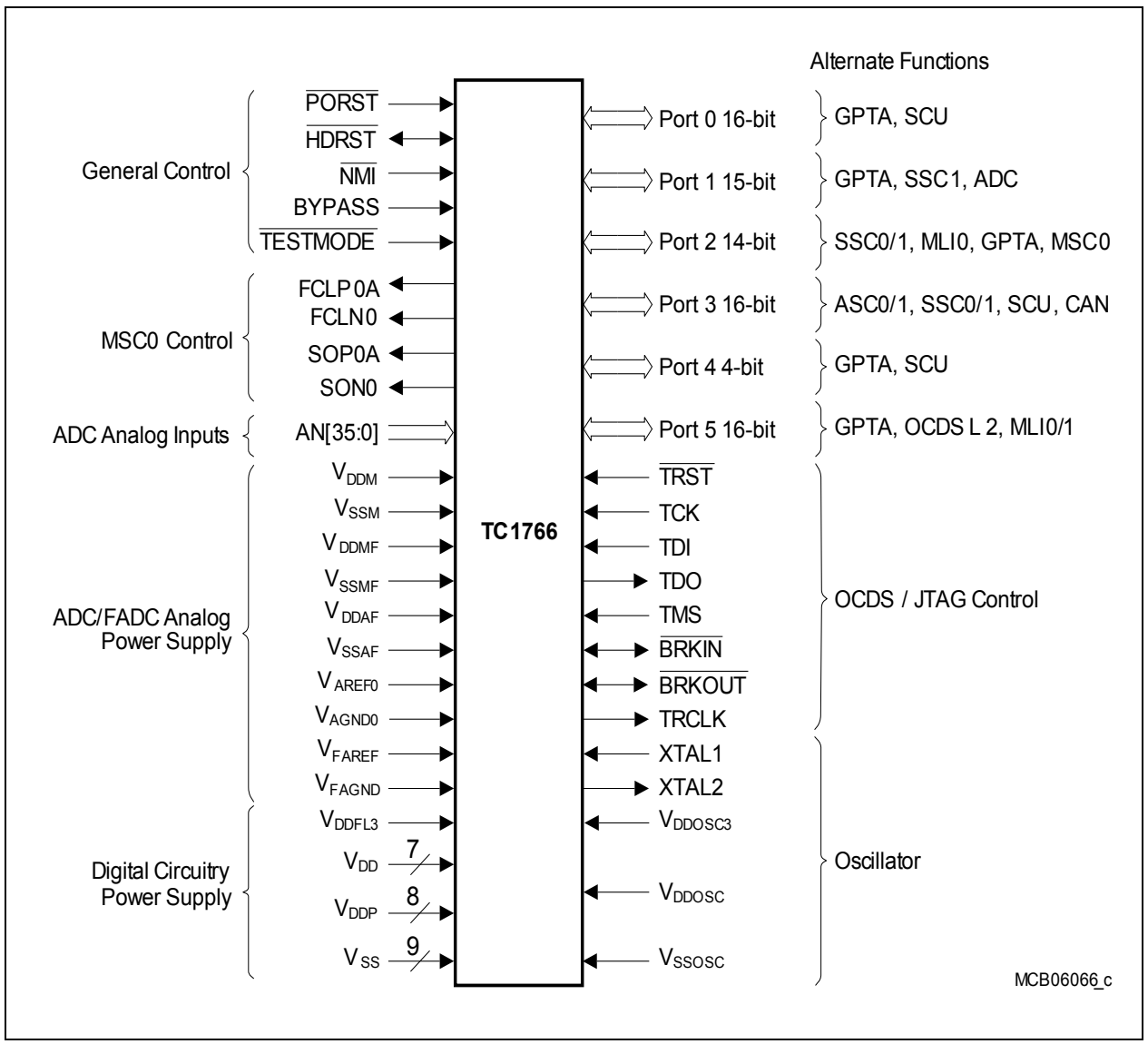


Figure 1-11 TC1766 Logic Symbol

1.4.1 TC1766 Pin Configuration

Figure 1-11 shows the TC1766 pin configuration.

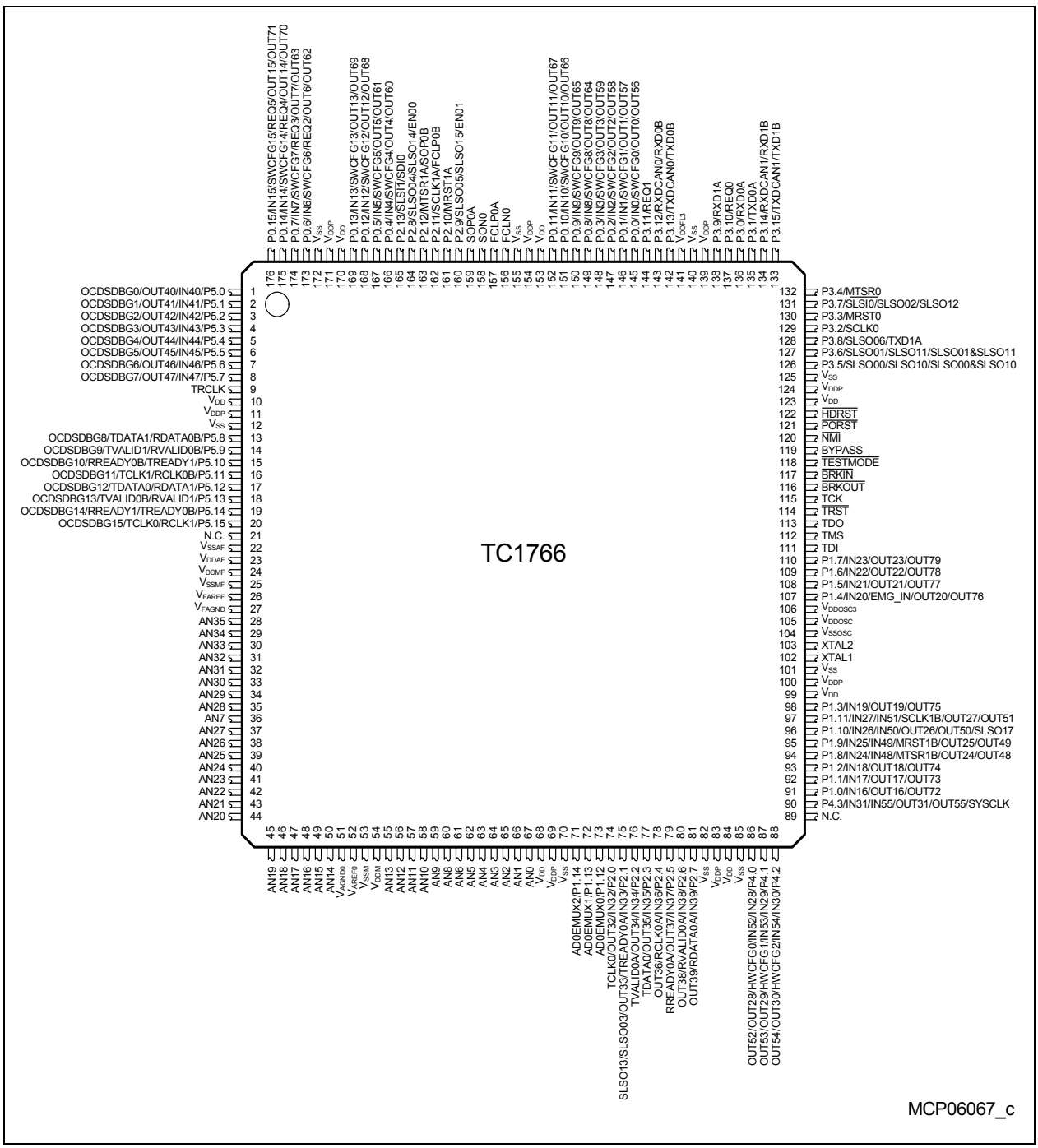


Figure 1-12 TC1766 Pinning for PG-LQFP-176-2 Package

Table 1-3 Pin Definitions and Functions

Symbol	Pins	I/O	Pad Driver Class	Power Supply	Functions
Parallel Ports					
P0		I/O	A1	V_{DDP}	**Port 0** Port 0 is a 16-bit bi-directional general-purpose I/O port which can be alternatively used for GPTA I/O lines or external trigger inputs.
P0.0	145				IN0 / OUT0 / OUT56 line of GPTA
P0.1	146				IN1 / OUT1 / OUT57 line of GPTA
P0.2	147				IN2 / OUT2 / OUT58 line of GPTA
P0.3	148				IN3 / OUT3 / OUT59 line of GPTA
P0.4	166				IN4 / OUT4 / OUT60 line of GPTA
P0.5	167				IN5 / OUT5 / OUT61 line of GPTA
P0.6	173				IN6 / OUT6 / OUT62 line of GPTA REQ2 External trigger input 2
P0.7	174				IN7 / OUT7 / OUT63 line of GPTA REQ3 External trigger input 3
P0.8	149				IN8 / OUT8 / OUT64 line of GPTA
P0.9	150				IN9 / OUT9 / OUT65 line of GPTA
P0.10	151				IN10 / OUT10 / OUT66 line of GPTA
P0.11	152				IN11 / OUT11 / OUT67 line of GPTA
P0.12	168				IN12 / OUT12 / OUT68 line of GPTA
P0.13	169				IN13 / OUT13 / OUT69 line of GPTA
P0.14	175				IN14 / OUT14 / OUT70 line of GPTA REQ4 External trigger input 4
P0.15	176				IN15 / OUT15 / OUT71 line of GPTA REQ5 External trigger input 5
					In addition, the state of the port pins are latched into the software configuration input register SCU_SCLIR at the rising edge of HDRST. In the different TC1766 device versions, several Port 0 pins are reserved for device configuration purposes (see **Page 9-28**).

Table 1-3 Pin Definitions and Functions (cont'd)

Symbol	Pins	I/O	Pad Driver Class	Power Supply	Functions
P1		I/O		V_{DDP}	**Port 1** Port 1 is a 15-bit bi-directional general-purpose I/O port which can be alternatively used for GPTA I/O lines, SSC1 and ADC0 interface.
P1.0	91		A1		IN16 / OUT16 / OUT72 line of GPTA
P1.1	92		A1		IN17 / OUT17 / OUT73 line of GPTA
P1.2	93		A1		IN18 / OUT18 / OUT74 line of GPTA
P1.3	98		A1		IN19 / OUT19 / OUT75 line of GPTA
P1.4	107		A1		IN20 / OUT20 / OUT76 line of GPTA
P1.5	108		A1		IN21 / OUT21 / OUT77 line of GPTA
P1.6	109		A1		IN22 / OUT22 / OUT78 line of GPTA
P1.7	110		A1		IN23 / OUT23 / OUT79 line of GPTA
P1.8	94		A2		IN24 / OUT24 / IN48 / OUT48 line of GPTA MTSR1B SSC1 master transmit output / slave rec. input B
P1.9	95		A2		IN25 / OUT25 / IN49 / OUT49 line of GPTA MRST1B SSC1 master receive input / slave transmit output B
P1.10	96		A2		IN26 / OUT26 / IN50 / OUT50 line of GPTA SLSO17 SSC1 slave select output 7
P1.11	97		A2		IN27 / OUT27 / IN51 / OUT51 line of GPTA SCLK1B SSC1 clock input / output B
P1.12	73		A1		AD0EMUX0 ADC0 external multiplexer control output 0
P1.13	72		A1		AD0EMUX1 ADC0 external multiplexer control output 1
P1.14	71		A1		AD0EMUX2 ADC0 external multiplexer control output 2 In addition, P1.4 also serves as **emergency shut-off input** for certain I/O lines (e.g. GPTA related outputs).

Table 1-3 Pin Definitions and Functions (cont'd)

Symbol	Pins	I/O	Pad Driver Class	Power Supply	Functions	
P2		I/O		V_{DDP}	**Port 2** Port 2 is a 14-bit bi-directional general-purpose I/O port which can be alternatively used for GPTA I/O, and interface for MLI0, MSC0 or SSC0/1.	
P2.0	74		A2		TCLK0	MLI0 transmit channel clock output
					IN32 / OUT32	line of GPTA
P2.1	75		A2		TREADY0A	MLI0 transmit channel ready input A
					IN33 / OUT33	line of GPTA
					SLSO03	SSC0 slave select output 3
					SLSO13	SSC1 slave select output 3
P2.2	76		A2		TVALID0A	MLI0 transmit channel valid output A
					IN34 / OUT34	line of GPTA
P2.3	77		A2		TDATA0	MLI0 transmit channel data output
					IN35 / OUT35	line of GPTA
P2.4	78		A1		RCLK0A	MLI0 receive channel clock input A
					IN36 / OUT36	line of GPTA
P2.5	79		A2		RREADY0A	MLI0 receive channel ready output A
					IN37 / OUT37	line of GPTA
P2.6	80		A1		RVALID0A	MLI0 receive channel valid input A
					IN38 / OUT38	line of GPTA
P2.7	81		A1		RDATA0A	MLI0 receive channel data input A
					IN39 / OUT39	line of GPTA

Table 1-3 Pin Definitions and Functions (cont'd)

Symbol	Pins	I/O	Pad Driver Class	Power Supply	Functions	
P2.8	164		A2		SLSO04	SSC0 Slave Select outp. 4
					SLSO14	SSC1 Slave Select outp. 4
					EN00	MSC0 enable output 0
P2.9	160		A2		SLSO05	SSC0 Slave Select outp. 5
					SLSO15	SSC1 Slave Select outp. 5
					EN01	MSC0 enable output 1
P2.10	161		A2		MRST1A	SSC1 master receive input / slave transmit output A
P2.11	162		A2		SCLK1A	SSC1 clock input/output A
					FCLP0B	MSC0 clock output B
P2.12	163		A2		MTRS1A	SSC1 master transmit out / slave receive input A
					$\overline{\text{SOP0B}}$	MSC0 serial data output B
P2.13	165		A1		SLSI1	SSC1 slave select input
					SDI0	MSC0 serial data input

Table 1-3 Pin Definitions and Functions (cont'd)

Symbol	Pins	I/O	Pad Driver Class	Power Supply	Functions	
P3		I/O		V_{DDP}	**Port 3** Port 3 is a 16-bit bi-directional general-purpose I/O port which can be alternatively used for ASC0/1, SSC0/1 and CAN lines.	
P3.0	136		A2		RXD0A	ASC0 receiver inp./outp. A
P3.1	135		A2		TXD0A	ASC0 transmitter output A
					This pin is sampled at the rising edge of $\overline{\text{PORST}}$. If this pin and the BYPASS input pin are both active, then oscillator bypass mode is entered.	
P3.2	129		A2		SCLK0	SSC0 clock input/output
P3.3	130		A2		MRST0	SSC0 master receive input/ slave transmit output
P3.4	132		A2		MTSR0	SSC0 master transmit output/slave receive input
P3.5	126		A2		SLSO00	SSC0 slave select output 0
					SLSO10	SSC1 slave select output 0 [1]
P3.6	127		A2		SLSO01	SSC0 slave select output 1
					SLSO11	SSC1 slave select output 1 [1]
P3.7	131		A2		$\overline{\text{SLSI0}}$	SSC0 slave select input
					SLSO02	SSC0 slave select output 2
					SLSO12	SSC1 slave select output 2
P3.8	128		A2		SLSO06	SSC0 slave select output 6
					TXD1A	ASC1 transmitter output A
P3.9	138		A2		RXD1A	ASC1 receiver inp./outp. A
P3.10	137		A1		REQ0	External trigger input 0
P3.11	144		A1		REQ1	External trigger input 1
P3.12	143		A2		RXDCAN0	CAN node 0 receiver input
					RXD0B	ASC0 receiver inp./outp. B
P3.13	142		A2		TXDCAN0	CAN node 0 transm. output
					TXD0B	ASC0 transmitter output B
P3.14	134		A2		RXDCAN1	CAN node 1 receiver input
					RXD1B	ASC1 receiver inp./outp. B
P3.15	133		A2		TXDCAN1	CAN node 1 transm. output
					TXD1B	ASC1 transmitter output B

Table 1-3 Pin Definitions and Functions (cont'd)

Symbol	Pins	I/O	Pad Driver Class	Power Supply	Functions
P4		I/O		V_{DDP}	**Port 4 / Hardware Configuration Inputs**
P4.[3:0]					HWCFG[3:0] Boot mode and boot location inputs; inputs are latched with the rising edge of $\overline{\text{HDRST}}$. During normal operation, Port 4 pins may be used as alternate functions for GPTA or system clock output.
P4.0	86		A1		IN28 / OUT28 / IN52 / OUT52 line of GPTA
P4.1	87		A1		IN29 / OUT29 / IN53 / OUT53 line of GPTA
P4.2	88		A2		IN30 / OUT30 / IN54 / OUT54 line of GPTA
P4.3	90		A2		IN31 / OUT31 / IN55 / OUT55 line of GPTA SYSCLK System Clock Output

Table 1-3 Pin Definitions and Functions (cont'd)

Symbol	Pins	I/O	Pad Driver Class	Power Supply	Functions
P5		I/O	A2	V_{DDP}	**Port 5** Port 5 is a 16-bit bi-directional general-purpose I/O port. In emulation, it is used as a trace port for OCDS Level 2 debug lines. In normal operation, it is used for GPTA I/O or the MLI0/MLI1 interface.
P5.0	1				OCDSDBG0 OCDS L2 Debug Line 0 (Pipeline Status Sig. PS0) IN40 / OUT40 line of GPTA
P5.1	2				OCDSDBG1 OCDS L2 Debug Line 1 (Pipeline Status Sig. PS1) IN41 / OUT41 line of GPTA
P5.2	3				OCDSDBG2 OCDS L2 Debug Line 2 (Pipeline Status Sig. PS2) IN42 / OUT42 line of GPTA
P5.3	4				OCDSDBG3 OCDS L2 Debug Line 3 (Pipeline Status Sig. PS3) IN43 / OUT43 line of GPTA
P5.4	5				OCDSDBG4 OCDS L2 Debug Line 4 (Pipeline Status Sig. PS4) IN44 / OUT44 line of GPTA
P5.5	6				OCDSDBG5 OCDS L2 Debug Line 5 (Break Qualification Line BRK0) IN45 / OUT45 line of GPTA
P5.6	7				OCDSDBG6 OCDS L2 Debug Line 6 (Break Qualification Line BRK1) IN46 / OUT46 line of GPTA
P5.7	8				OCDSDBG7 OCDS L2 Debug Line 7 (Break Qualification Line BRK2) IN47 / OUT47 line of GPTA

Table 1-3 Pin Definitions and Functions (cont'd)

Symbol	Pins	I/O	Pad Driver Class	Power Supply	Functions	
P5.8	13				OCDSDBG8	OCDS L2 Debug Line 8 (Indirect PC Addr. PC0)
					TDATA1	MLI1 transmit channel data output
					RDATA0B	MLI0 receive channel data input B
P5.9	14				OCDSDBG9	OCDS L2 Debug Line 9 (Indirect PC Addr. PC1)
					TVALID1	MLI1 transmit channel valid output
					RVALID0B	MLI0 receive channel valid input B
P5.10	15				OCDSDBG10	OCDS L2 Debug Line 10 (Indirect PC Addr. PC2)
					TREADY1	MLI1 transmit channel ready input
					RREADY0B	MLI0 receive channel ready output B
P5.11	16				OCDSDBG11	OCDS L2 Debug Line 11 (Indirect PC Addr. PC3)
					TCLK1	MLI1 transmit channel clock output
					RCLK0B	MLI0 receive channel clock input B
P5.12	17				OCDSDBG12	OCDS L2 Debug Line 12 (Indirect PC Addr. PC04)
					RDATA1	MLI1 receive channel data input
					TDATA0	MLI0 transmit channel data output
P5.13	18				OCDSDBG13	OCDS L2 Debug Line 13 (Indirect PC Addr. PC05)
					RVALID1	MLI1 receive channel valid input
					TVALID0B	MLI0 transmit channel valid output B

Table 1-3 Pin Definitions and Functions (cont'd)

Symbol	Pins	I/O	Pad Driver Class	Power Supply	Functions	
P5.14	19				OCDSDBG14	OCDS L2 Debug Line 14 (Indirect PC Address PC6)
					RREADY1	MLI1 receive channel ready output
					TREADY0B	MLI0 transmit channel ready input B
P5.15	20				OCDSDBG15	OCDS L2 Debug Line 15 (Indirect PC Address PC7)
					RCLK1	MLI1 receive channel clock input
					TCLK0	MLI0 transmit channel clock output
MSC0 Outputs						
			C	V_{DDP}	**LVDS MSC Clock and Data Outputs**[3)]	
FCLP0A	157	O			MSC0 Differential Driver Clock Output Positive A	
FCLN0	156	O			MSC0 Differential Driver Clock Output Negative	
SOP0A	159	O			MSC0 Differential Driver Serial Data Output Positive A	
SON0	158	O			MSC0 Differential Driver Serial Data Output Negative	

Table 1-3 Pin Definitions and Functions (cont'd)

Symbol	Pins	I/O	Pad Driver Class	Power Supply	Functions
Analog Inputs					
AN[35:0]		I	D	–	**Analog Input Port** The Analog Input Port provides altogether 36 analog input lines to ADC0 and FADC. AN[31:0]: ADC0 analog inputs [31:0] AN[35:32]: FADC analog differential inputs
AN0	67				Analog input 0
AN1	66				Analog input 1
AN2	65				Analog input 2
AN3	64				Analog input 3
AN4	63				Analog input 4
AN5	62				Analog input 5
AN6	61				Analog input 6
AN7	36				Analog input 7
AN8	60				Analog input 8
AN9	59				Analog input 9
AN10	58				Analog input 10
AN11	57				Analog input 11
AN12	56				Analog input 12
AN13	55				Analog input 13
AN14	50				Analog input 14
AN15	49				Analog input 15
AN16	48				Analog input 16
AN17	47				Analog input 17
AN18	46				Analog input 18
AN19	45				Analog input 19
AN20	44				Analog input 20
AN21	43				Analog input 21
AN22	42				Analog input 22
AN23	41				Analog input 23
AN24	40				Analog input 24
AN25	39				Analog input 25
AN26	38				Analog input 26
AN27	37				Analog input 27
AN28	35				Analog input 28
AN29	34				Analog input 29
AN30	33				Analog input 30

Table 1-3 Pin Definitions and Functions (cont'd)

Symbol	Pins	I/O	Pad Driver Class	Power Supply	Functions
AN31 AN32 AN33 AN34 AN35	32 31 30 29 28	I	D	–	Analog input 31 Analog input 32 Analog input 33 Analog input 34 Analog input 35
System I/O					
TRST	114	I	A2[2)]	V_{DDP}	**JTAG Module Reset/Enable Input**
TCK	115	I	A2[2)]	V_{DDP}	**JTAG Module Clock Input**
TDI	111	I	A1[2)]	V_{DDP}	**JTAG Module Serial Data Input**
TDO	113	O	A2	V_{DDP}	**JTAG Module Serial Data Output**
TMS	112	I	A2[2)]	V_{DDP}	**JTAG Module State Machine Control Input**
BRKIN	117	I/O	A3	V_{DDP}	**OCDS Break Input (Alternate Output)[3)4)]**
BRK OUT	116	I/O	A3	V_{DDP}	**OCDS Break Output (Alternate Input)[3)4)]**
TRCLK	9	O	A4	V_{DDP}	**Trace Clock for OCDS_L2 Lines[3)]**
NMI	120	I	A2[5)6)]	V_{DDP}	**Non-Maskable Interrupt Input**
HDRST	122	I/O	A2[7)]	V_{DDP}	**Hardware Reset Input / Reset Indication Output**
PORST[8)]	121	I	A2[5)6)]	V_{DDP}	**Power-on Reset Input**
BYPASS	119	I	A1[2)]	V_{DDP}	**PLL Clock Bypass Select Input** This input has to be held stable during power-on resets. With BYPASS = 1, the spike filters in the HDRST, PORST and NMI inputs are switched off.
TEST MODE	118	I	A2[5)9)]	V_{DDP}	**Test Mode Select Input** For normal operation of the TC1766, this pin should be connected to high level.
XTAL1 **XTAL2**	102 103	I O	n.a.	V_{DDOSC}	**Oscillator/PLL/Clock Generator Input/Output Pins**

Table 1-3 Pin Definitions and Functions (cont'd)

Symbol	Pins	I/O	Pad Driver Class	Power Supply	Functions
N.C.	21, 89	–	–	–	**Not Connected** These pins are reserved for future extension and must not be connected externally.
Power Supplies					
V_{DDM}	54	–	–	–	**ADC Analog Part Power Supply (3.3 V)**
V_{SSM}	53	–	–	–	**ADC Analog Part Ground for V_{DDM}**
V_{DDMF}	24	–	–	–	**FADC Analog Part Power Supply (3.3 V)**
V_{SSMF}	25	–	–	–	**FADC Analog Part Ground for V_{DDMF}**
V_{DDAF}	23	–	–	–	**FADC Analog Part Logic Power Supply (1.5 V)**
V_{SSAF}	22	–	–	–	**FADC Analog Part Logic Ground for V_{DDAF}**
V_{AREF0}	52	–	–	–	**ADC Reference Voltage**
V_{AGND0}	51	–	–	–	**ADC Reference Ground**
V_{FAREF}	26	–	–	–	**FADC Reference Voltage**
V_{FAGND}	27	–	–	–	**FADC Reference Ground**
V_{DDOSC}	105	–	–	–	**Main Oscillator and PLL Power Supply (1.5 V)**
V_{DDOSC3}	106	–	–	–	**Main Oscillator Power Supply (3.3 V)**
V_{SSOSC}	104	–	–	–	**Main Oscillator and PLL Ground**
V_{DDFL3}	141	–	–	–	**Power Supply for Flash (3.3 V)**
V_{DD}	10, 68, 84, 99, 123, 153, 170	–	–	–	**Core Power Supply (1.5 V)**

Table 1-3 Pin Definitions and Functions (cont'd)

Symbol	Pins	I/O	Pad Driver Class	Power Supply	Functions
V_{DDP}	11, 69, 83, 100, 124, 154, 171, 139	–	–	–	**Port Power Supply (3.3 V)**
V_{SS}	12, 70, 85, 101, 125, 155, 172, 140, 82	–	–	–	**Ground**

1) The logical AND function of the two slave select outputs is available as a third alternate output function.
2) These pads are I/O pads with input only function. Its input characteristics are identical with the input characteristics as defined for class A pads.
3) In case of a power-fail condition (one or more power supply voltages drop below the specified voltage range) an undefined output driving level may occur at these pins.
4) Programmed by software as either break input or break output.
5) These pads are input only pads with input characteristics.
6) Input only pads with input spike filter.
7) Open drain pad with input spike filter.
8) The dual input reset system of TC1766/TC1766ED, assumes that the $\overline{\text{PORST}}$ reset pin is used for power on reset only. If a system uses the $\overline{\text{PORST}}$ reset input for other system resets, it has to be taken into account, that the emulation part of the TC1766ED Emulation Device is reset also. It thus will force always a complete re-initialization of the emulator and will prevent the user debugging across these types of resets.
9) Input only pads without input spike filter.

1.4.2 Pad Driver Classes Overview

Table 1-4 gives a general overview of the pad driver classes of the TC1766. Further details can be found in the “TC1766 Data Sheet”.

Table 1-4 Pad Driver and Input Classes Overview

Class	Power Supply	Type	Sub Class	Speed Grade	Termination
A	3.3 V	LVTTL I/O, LVTTL outputs	**A1** (e.g. GPIO)	6 MHz	No
			A2 (e.g. serial I/Os)	40 MHz	Series termination recommended
			A3 (e.g. Trace outputs, serial I/Os)	80 MHz	Yes, series termination
			A4 (e.g. Trace Clock)	80 MHz	Yes, series termination
C	3.3 V	LVDS	–	50 MHz	Parallel termination
D	–	Analog input	–	–	–

1.4.3 Reset Behavior of the Pins

Table 1-5 describes the pull-up/pull-down behavior of the System I/O pins during power-on reset.

Table 1-5 List of Pull-up/Pull-down Reset Behavior of the Pins

Pins	Reset Behavior	
	$\overline{\text{PORST}}$ = 0	**$\overline{\text{PORST}}$ = 1**
All GPIOs,TDI, TMS, TDO	Pull-up	
$\overline{\text{HDRST}}$	Drive-low	Pull-up
BYPASS	Pull-up	High-impedance
$\overline{\text{TRST}}$, TCK	High-impedance	Pull-down
TRCLK	High-impedance	
$\overline{\text{BRKIN}}$, $\overline{\text{BRKOUT}}$, $\overline{\text{TESTMODE}}$	Pull-up	
$\overline{\text{NMI}}$, $\overline{\text{PORST}}$	Pull-down	

2 CPU Subsystem

The TC1766 processor contains a TriCore 1 V1.3 CPU. This chapter describes the implementation-specific options of the CPU, and should be read in conjunction with the TriCore 1 Architecture Manual, which describes the complete TriCore Architecture including the register and instruction set description.

2.1 TC1766 Processor Subsystem

The diagram below shows the block diagram of the TC1766 Processor subsystem. It also shows the on-chip bus systems.

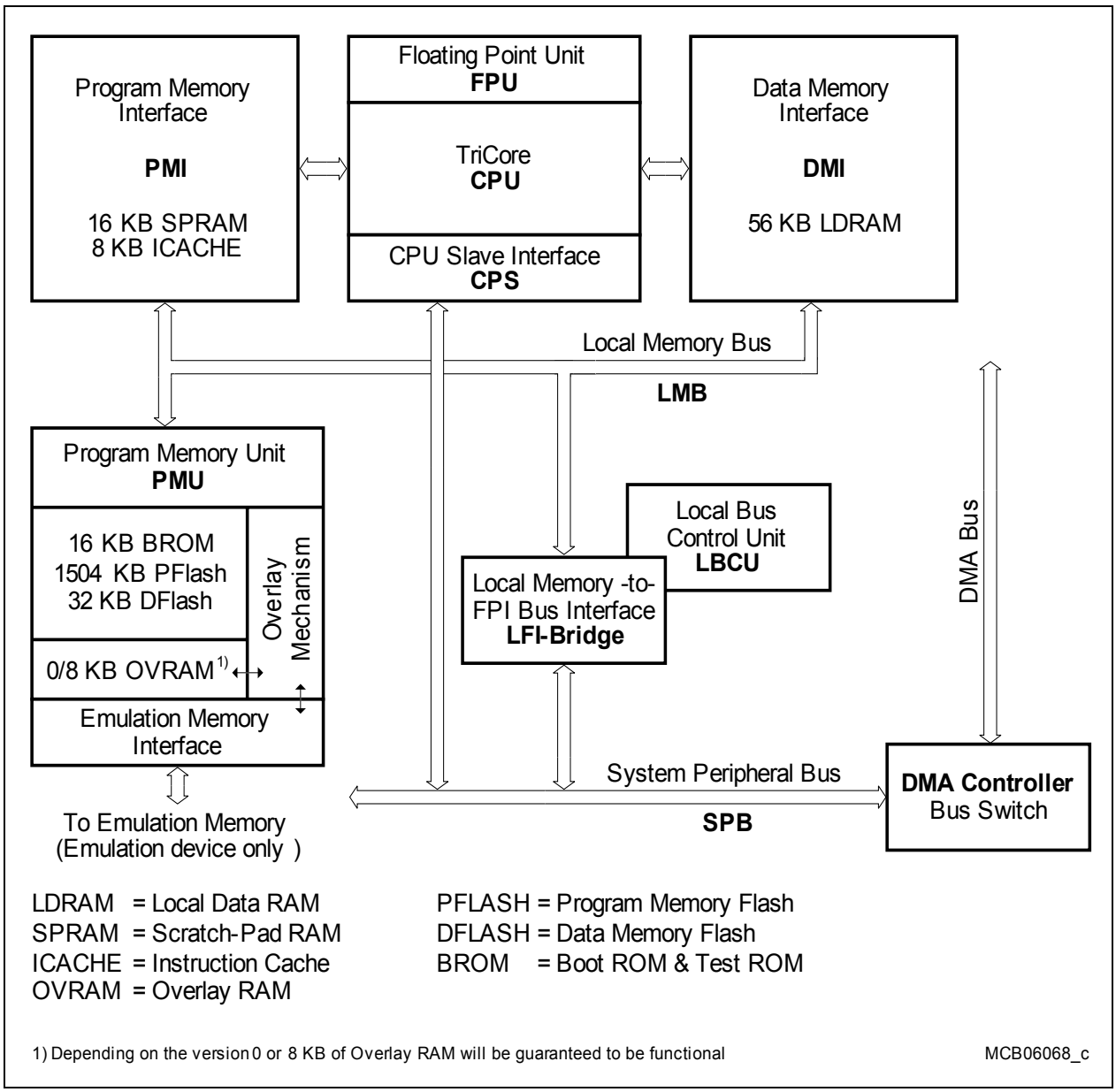


Figure 2-1 Processor Subsystem Block Diagram

2.2 Central Processing Unit Features

The 80 MHz TriCore TC1766 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 and 2, supported in conjunction with the CPS block
  – Level 3, supported

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
- Two 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 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 optional Floating Point Unit (FPU).

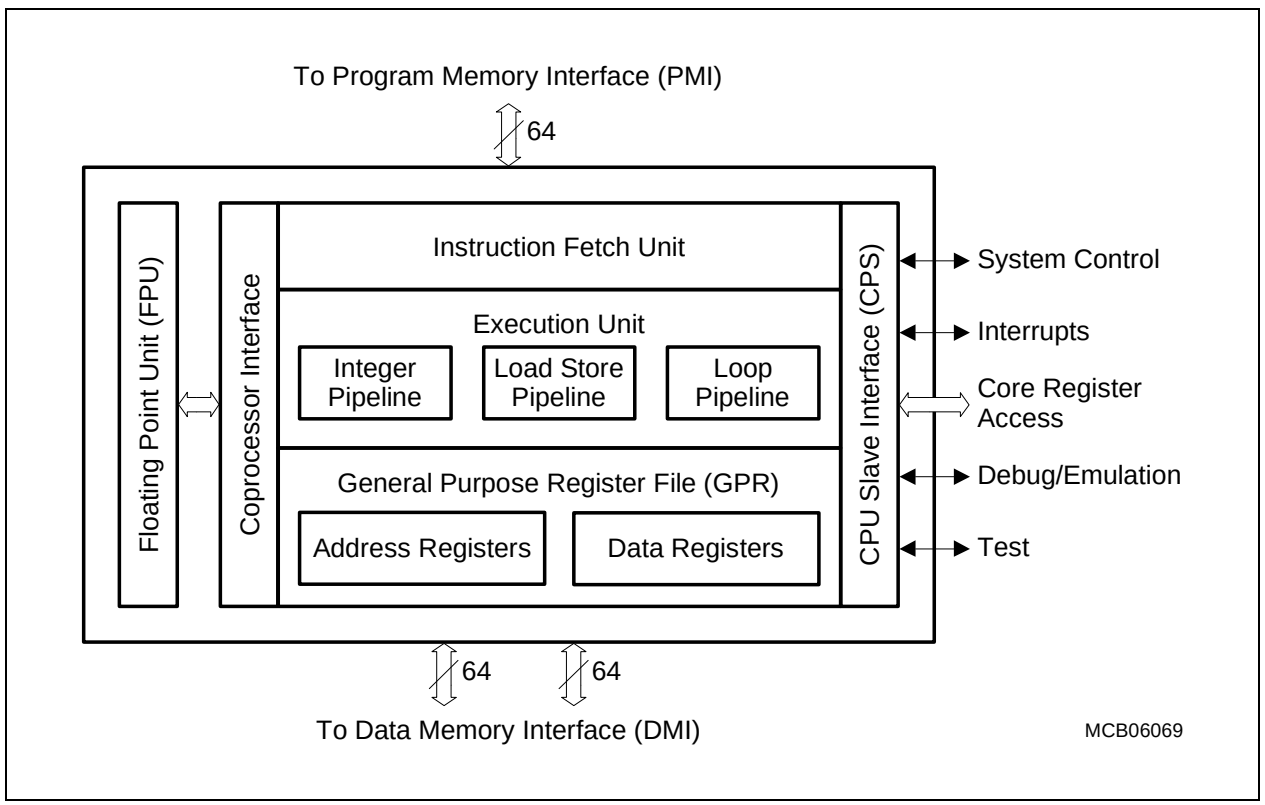


Figure 2-2 CPU Block Diagram

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). 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 PC Unit is responsible for updating the program counters.

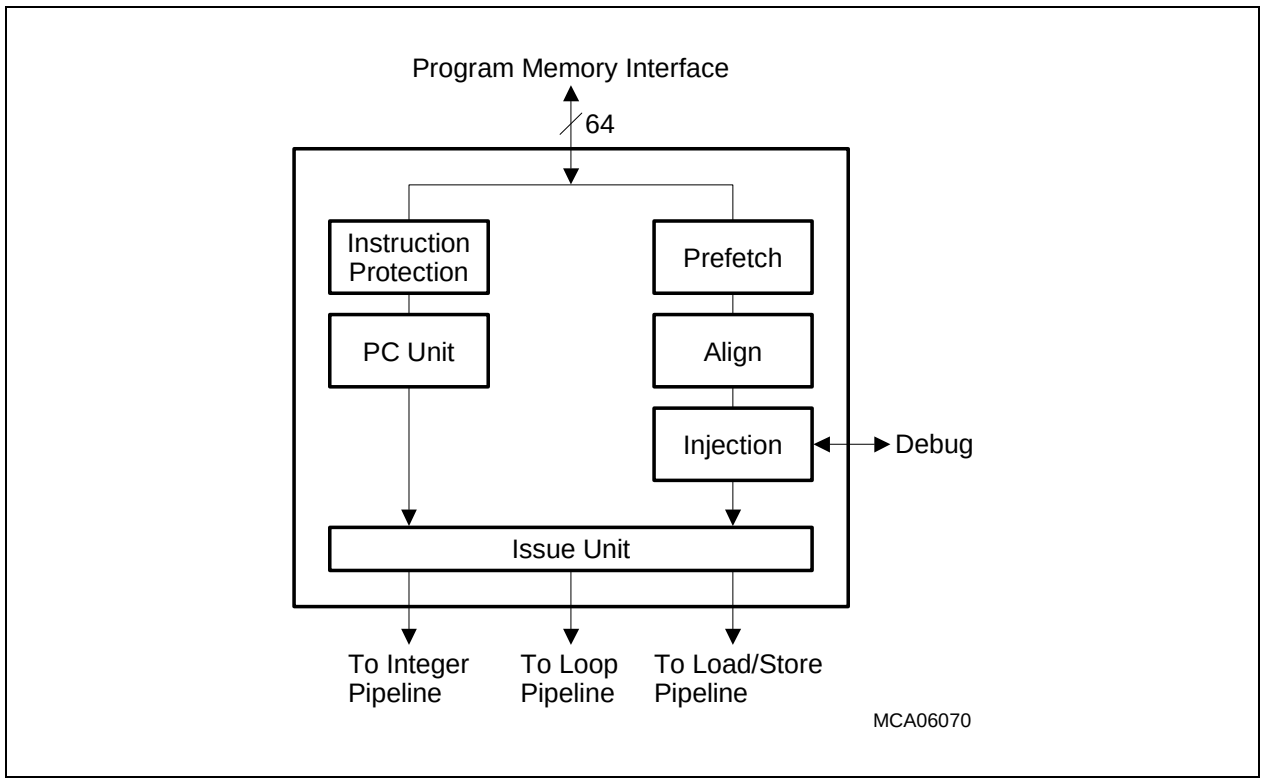


Figure 2-3 Instruction Fetch Unit

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.

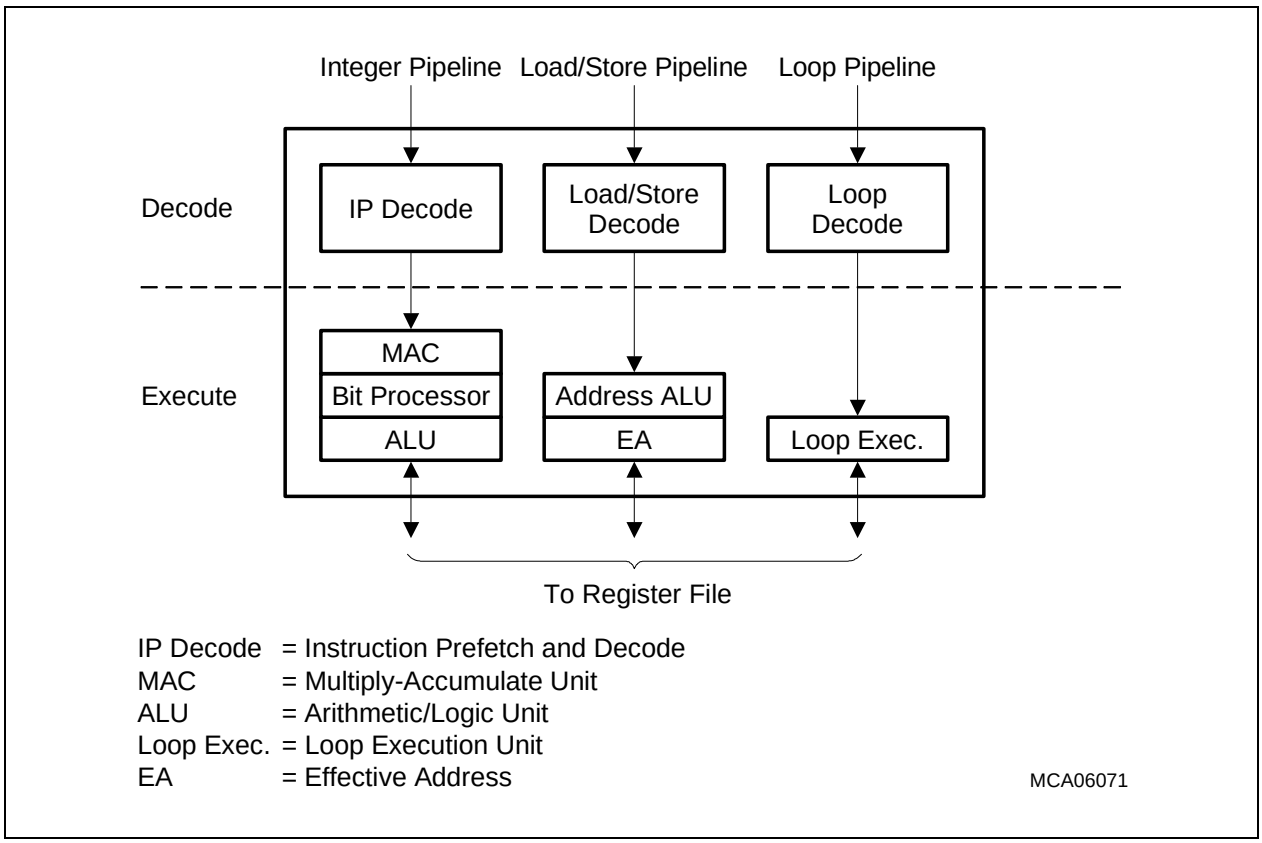


Figure 2-4 Execution Unit

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.

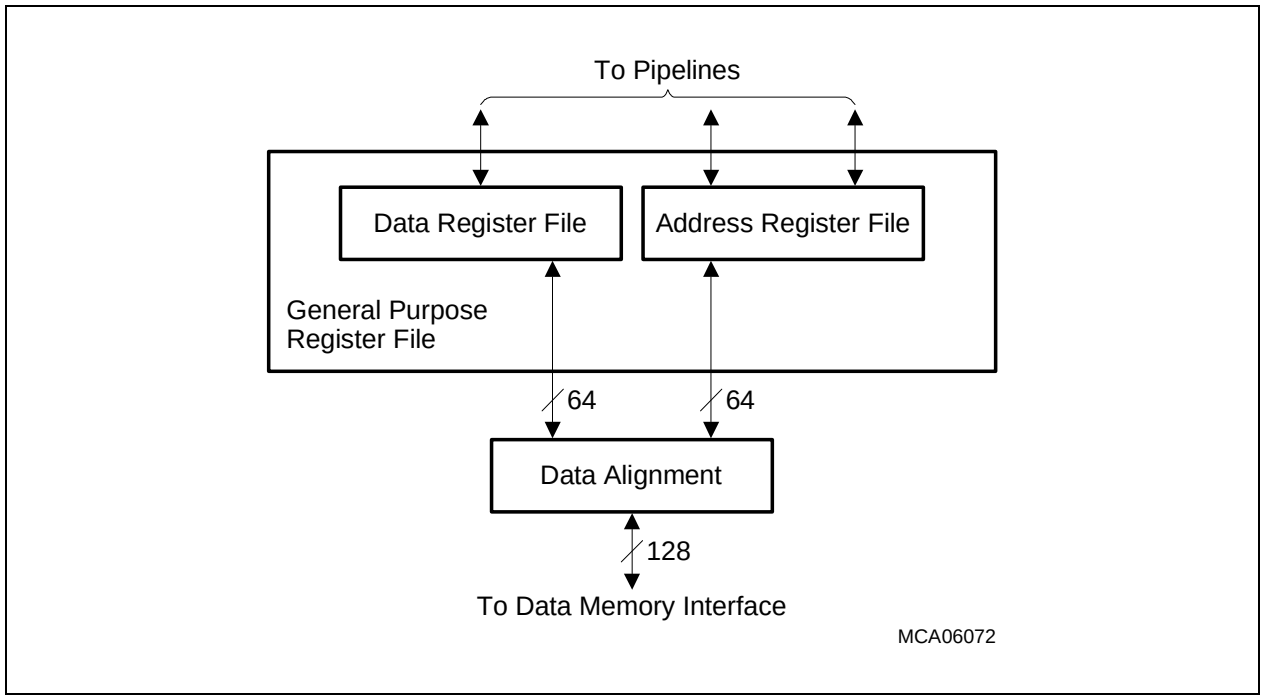


Figure 2-5 General Purpose Register File

2.3 Implementation-specific Features

This section describes the implementation-specific features of the TC1766 CPU. For a full description of the TriCore architecture refer to the TriCore 1 Architecture Manual.

2.3.1 Context Save Areas

In the TC1766, Context Save Areas (CSA) must be placed in LDRAM.

Refer to the TriCore 1 Architecture Manual Chapter 5 - Tasks and Functions.

2.3.2 Fast Context Switching

The TC1766 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. 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. Shadow registers are automatically restored from memory when required.

2.3.3 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.

Refer to the TriCore 1 Architecture Manual - Core Registers.

2.3.4 Interrupt System

An interrupt request can be generated by the TC1766 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 TC1766 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.

For more information, refer to **Chapter 12** of this User’s Manual.

2.3.5 Trap System

The following traps have implementation-specific properties. For a complete description of the trap system, refer to the TriCore 1 Architecture Manual - Trap System.

UOPC - Unimplemented Opcode (TIN 2)

The TC1766 UOPC trap is raised on optional MMU instructions, coprocessor two and coprocessor three instructions.

OPD - Invalid Operand (TIN 3)

The TC1766 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 2-34**.

DAE - Data Access Asynchronous Error (TIN 3)

The Data Access Asynchronous Error Trap (DAE) is generated by the DMI module when a store 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 2-34**.

2.4 TC1766 CPU Subsystem Registers

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

TC1766 implementation-specific CPU registers are referred directly in this section.

Table 2-1 CPU Subsystem Registers

Registers	Purpose	Description	Address Map
Core Special Function Registers (CSFRs)	Program state information, context and stack management, interrupt and trap control, system control	Architecture Manual	see **Page 16-69**
CPU Slave Interface Registers (CPSs)	Software break control and software service request control		see **Page 16-68**
Core General Purpose Registers (GPRs)	Address and data		see **Page 16-73**
Core Debug Registers	Debug control		see **Page 16-71**
Memory Protection Registers	Memory protection control and mode selection		see **Page 16-69**
Program Memory Interface Registers (PMI)	PMI instruction cache control and status	see **Page 2-28**	see **Page 16-82**
Data Memory Interface Registers (DMI)	DMI status and trap flags	see **Page 2-35**	see **Page 16-81**

The complete and detailed address map of the CPU and processor subsystem registers shown in **Table 2-1** is located in **Chapter 16**. The entries in column “Address Map” of **Table 2-1** directly point to the corresponding pages.

2.4.1 Core Special Function Registers (CSFR)

Figure 2-6 shows the CSFR registers of the TC1766.

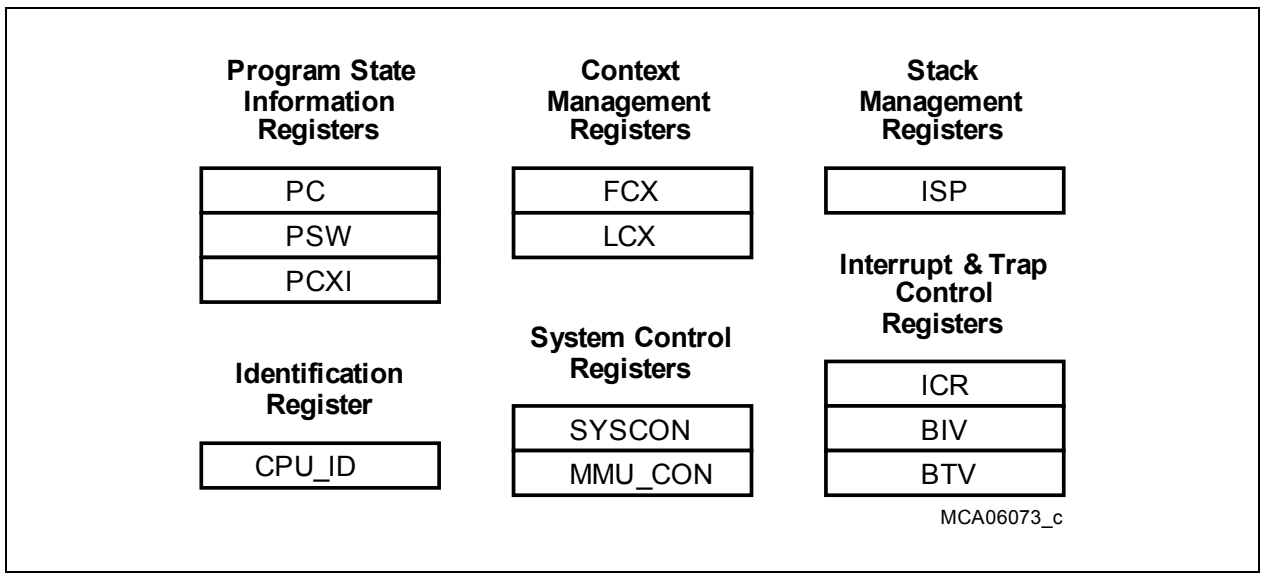


Figure 2-6 CSFR Registers

Table 2-2 Core Special Function Registers

Register Short Name	Register Long Name	Address
MMU_CON	MMU Configuration Register	F7E1 8000$_H$
PCXI	Previous Context Information Register	F7E1 FE00$_H$
PSW	Program Status Word Register	F7E1 FE04$_H$
PC	Program Counter Register	F7E1 FE08$_H$
SYSCON	System Configuration Register	F7E1 FE14$_H$
CPU_ID	CPU Identification Register	F7E1 FE18$_H$
BIV	Interrupt Vector Table Pointer Register	F7E1 FE20$_H$
BTV	Trap Vector Table Pointer Register	F7E1 FE24$_H$
ISP	Interrupt Stack Pointer Register	F7E1 FE28$_H$
ICR	ICU Interrupt Control Register	F7E1 FE2C$_H$
FCX	Free Context List Head Pointer Register	F7E1 FE38$_H$
LCX	Free Context List Limit Pointer Register	F7E1 FE3C$_H$

2.4.1.1 Implementation-specific Core Special Function Registers

This section describes the implementation-specific Program Status Word (PSW) which is an extension of the PSW description in the TriCore 1 Architecture Manual.

The PSW status flags used for FPU operations overlay the status flags used for Arithmetic Logic Unit (ALU) operations of the CPU. The non-shaded areas in the PSW register description define the implementation-specific bits/bit fields.

PSW
Program Status Word (F7E1FE04$_H$) Reset Value: 0000 0B80$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
C or FS	V or FI	SV or FV	AV or FZ	SAV or FU	FX	RM		0							
rwh	rwh	rwh	rwh	rwh	rwh	rw		r							

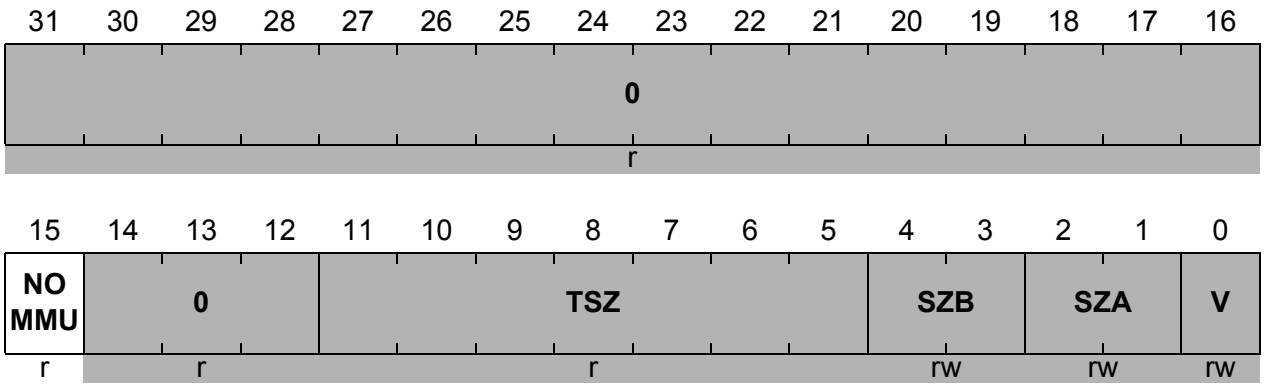
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		PRS		IO		IS	GW	CDE	CDC						
r		rwh		rwh		rwh	rwh	rwh	rwh						

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

The Interrupt Control Register is also an implementation-specific CFSR. Its Arbitration Cycle Control implementation-specific details are described on Page 12-8.

In the TC1766, the MMU_CON register indicates the non-availability of the TriCore 1's memory management unit (bit NO MMU is always set).

MMU_CON
MMU Configuration Register (F7E18000$_H$) 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.
0	[14:0], [31:16]	r	**Reserved** Read as 0; should be written with 0.

2.4.2 CPU Slave Interface (CPS) Registers

In the TC1766, the CPU Slave Interface (CPS) of the TriCore CPU directly accesses the interrupt service request registers in the CPU from the TC1766 System Peripheral Bus. The CPS registers are described in detail in the TriCore 1 Architecture Manual - Core Registers.

Figure 2-7 CPS Registers

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

Table 2-3 CPS Registers

Register Short Name	Register Long Name	Address
CPS_ID	CPS Module Identification Register	F7E0 FF08$_H$
CPU_SBSRC	CPU Software Breakpoint Service Request Control Register	F7E0 FFBC$_H$
CPU_SRC3	CPU Service Request Control 3 Register	F7E0 FFF0$_H$
CPU_SRC2	CPU Service Request Control 2 Register	F7E0 FFF4$_H$
CPU_SRC1	CPU Service Request Control 1 Register	F7E0 FFF8$_H$
CPU_SRC0	CPU Service Request Control 0 Register	F7E0 FFFC$_H$

2.4.2.1 Implementation-specific CPS Registers

All registers from **Table 2-3** have a TC1766-specific implementation detail, the Type of Service Control (TOS) bit/bit field. CPU_SBSRC is described on **Page 2-19**. The non-shaded areas in the CPU_SRCn register description defines the implementation-specific bits/bit fields.

CPU_SRCn (n = 0-3)
CPU Service Request Control Register n
($F7E0FFFC_H$-n*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
TOS	10	rw	**Type of Service Control** 0_B Service Provider = CPU 1_B Service Provider = PCP
0	11	r	**Reserved** Read as 0; should be written with 0.

2.4.3 CPU General Purpose Registers

Figure 2-8 shows the GPRs of the TC1766.

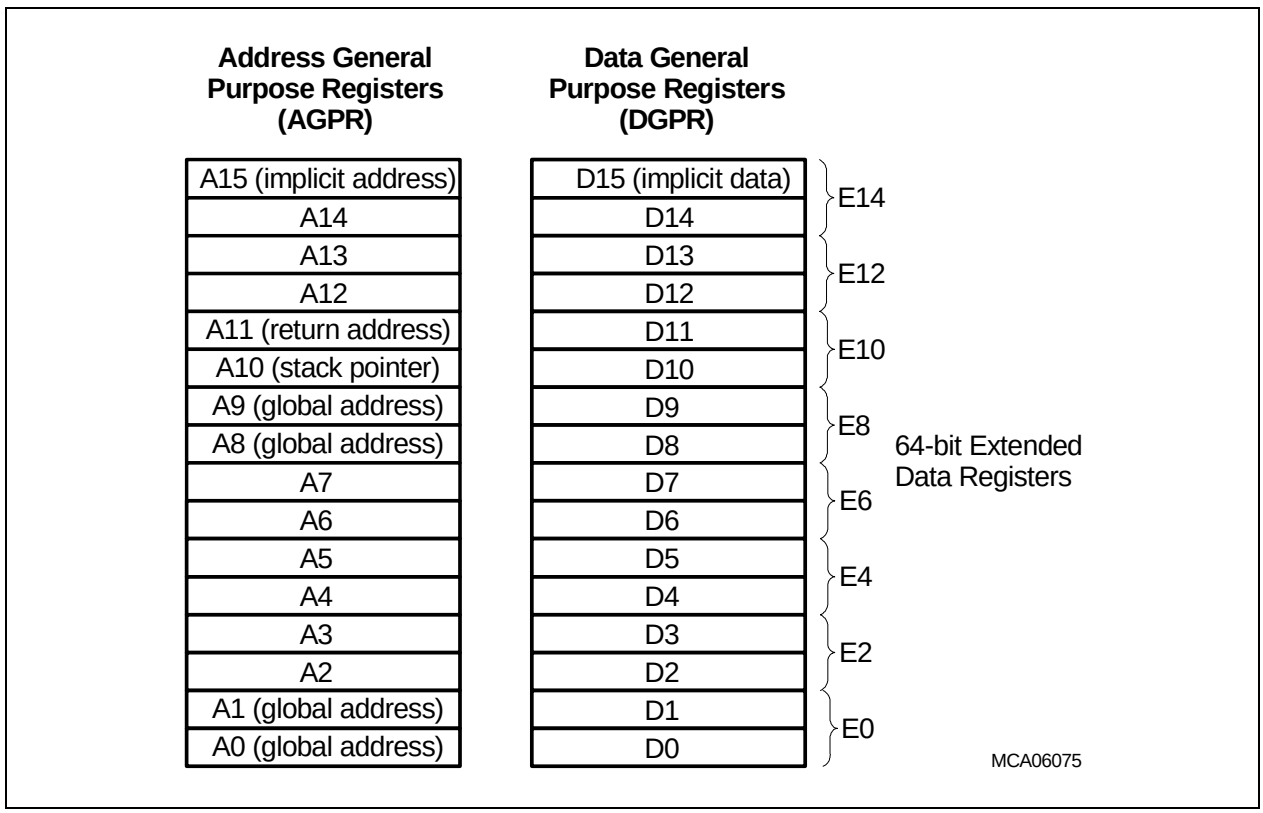


Figure 2-8 GPR Registers

Table 2-4 GPR Registers

Register Short Name	Register Long Name	Address
D0	Data Register 0	F7E1 FF00$_H$
D1	Data Register 1	F7E1 FF04$_H$
D2	Data Register 2	F7E1 FF08$_H$
D3	Data Register 3	F7E1 FF0C$_H$
D4	Data Register 4	F7E1 FF10$_H$
D5	Data Register 5	F7E1 FF14$_H$
D6	Data Register 6	F7E1 FF18$_H$
D7	Data Register 7	F7E1 FF1C$_H$
D8	Data Register 8	F7E1 FF20$_H$
D9	Data Register 9	F7E1 FF24$_H$

Table 2-4 GPR Registers (cont'd)

Register Short Name	Register Long Name	Address
D10	Data Register 10	F7E1 FF28$_H$
D11	Data Register 11	F7E1 FF2C$_H$
D12	Data Register 12	F7E1 FF30$_H$
D13	Data Register 13	F7E1 FF34$_H$
D14	Data Register 14	F7E1 FF38$_H$
D15	Data Register 15	F7E1 FF3C$_H$
A0	Address Register 0 (Global Address Register)	F7E1 FF80$_H$
A1	Address Register 1 (Global Address Register)	F7E1 FF84$_H$
A2	Address Register 2	F7E1 FF88$_H$
A3	Address Register 3	F7E1 FF8C$_H$
A4	Address Register 4	F7E1 FF90$_H$
A5	Address Register 5	F7E1 FF94$_H$
A6	Address Register 6	F7E1 FF98$_H$
A7	Address Register 7	F7E1 FF9C$_H$
A8	Address Register 8 (Global Address Register)	F7E1 FFA0$_H$
A9	Address Register 9 (Global Address Register)	F7E1 FFA4$_H$
A10	Address Register 10 (Stack Pointer)	F7E1 FFA8$_H$
A11	Address Register 11 (Return Address)	F7E1 FFAC$_H$
A12	Address Register 12	F7E1 FFB0$_H$
A13	Address Register 13	F7E1 FFB4$_H$
A14	Address Register 14	F7E1 FFB8$_H$
A15	Address Register 15	F7E1 FFBC$_H$

2.4.4 Core Debug Registers

In the TC1766, several Core Debug registers are available for debug purposes. These Core Debug registers are described in detail in the TriCore 1 Architecture Manual chapter - Core Debug Controller.

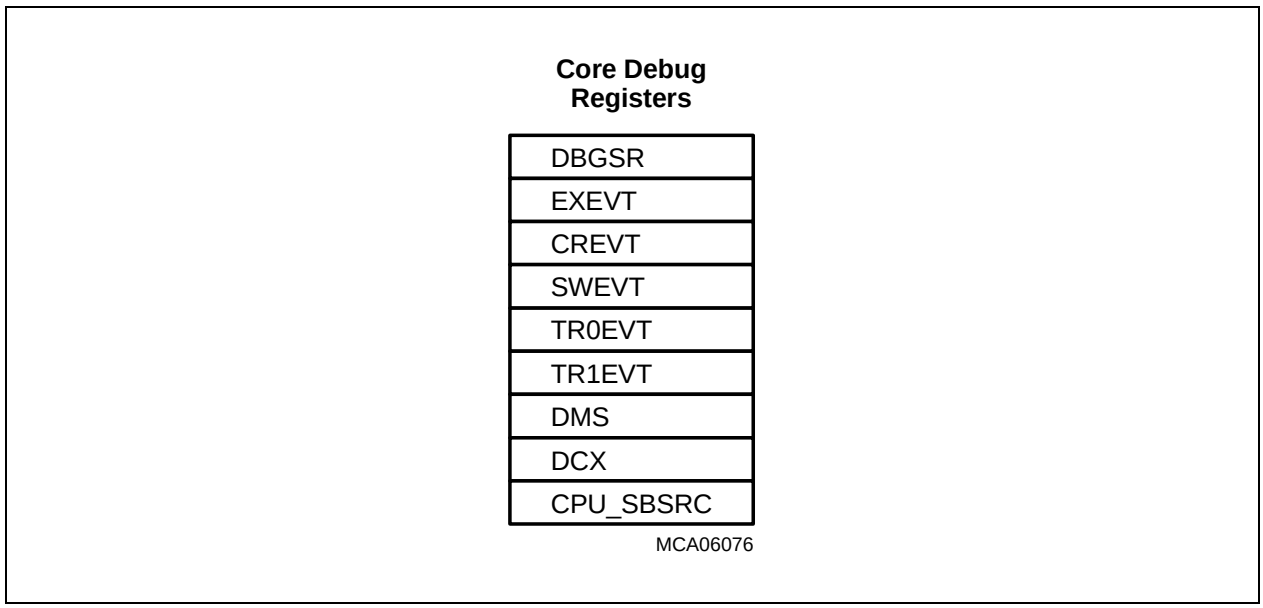


Figure 2-9 Core Debug Registers

Table 2-5 Core Debug 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 Specifier Register	F7E1 FD20$_H$
TR1EVT	Trigger Event 1 Specifier Register	F7E1 FD24$_H$
DMS	Debug Monitor Start Address Register	F7E1 FD40$_H$
DCX	Debug Context Save Area Pointer	F7E1 FD44$_H$
CPU_SBSRC	CPU Software Breakpoint Service Request Control Register	see **Table 2-3**[1)]

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

2.4.4.1 Implementation-specific Core Debug Registers

This section describes the implementation-specific Core Debug Registers which differ from the description in the TriCore 1 Architecture Manual. These are:

Trigger Event 0 Register
Trigger Event 1 Register
Software Breakpoint Service Request Control Register 0

The non-shaded areas in the register description define the implementation-specific bits/bit fields.

TR0EVT
Trigger Event 0 Specifier Register(F7E1 FD20$_H$) **Reset Value: 0000 0000$_H$**
TR1EVT
Trigger Event 1 Specifier Register(F7E1 FD24$_H$) **Reset Value: 0000 0000$_H$**

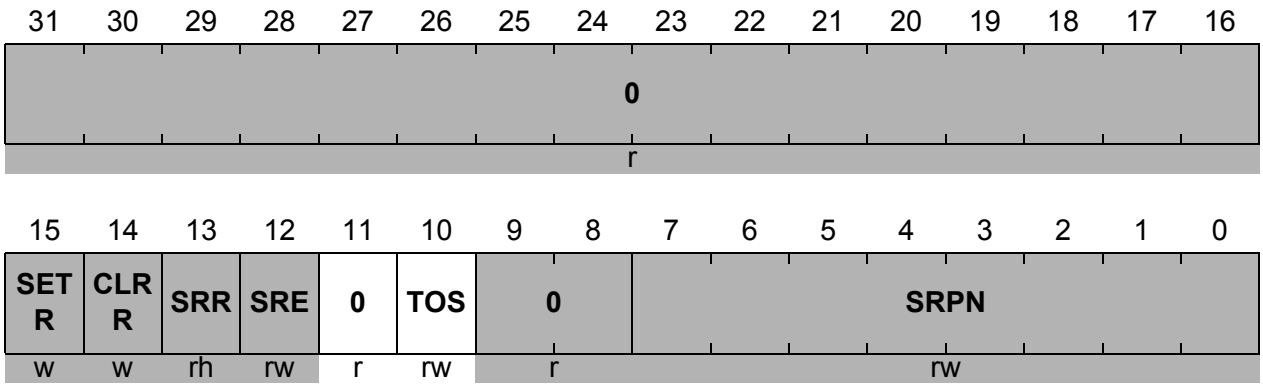
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0											0				
r											r				

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0			DU_U	DU_LR	DLR_U	DLR_LR	0		SUSP	0	BBM	EVTA		
r	r			rw	rw	rw	rw	r		rw	r	rw	rw		

Field	Bits	Type	Description
0	[20:15]	r	**Reserved** Read as 0; should be written with 0.

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.

2.4.5 Memory Protection Registers

As shown in **Figure 2-10**, there are two Memory Protection Register Sets in the TC1766, Set 0 and Set 1, which specify memory protection ranges and permissions for code and data. Set 2 and Set 3 are not implemented in the TC1766. The PSW.PRS bit field determines which of these sets is currently in use by the CPU.

The Memory Protection Registers are described in detail in the TriCore 1 Architecture Manual chapter - Memory Protection System.

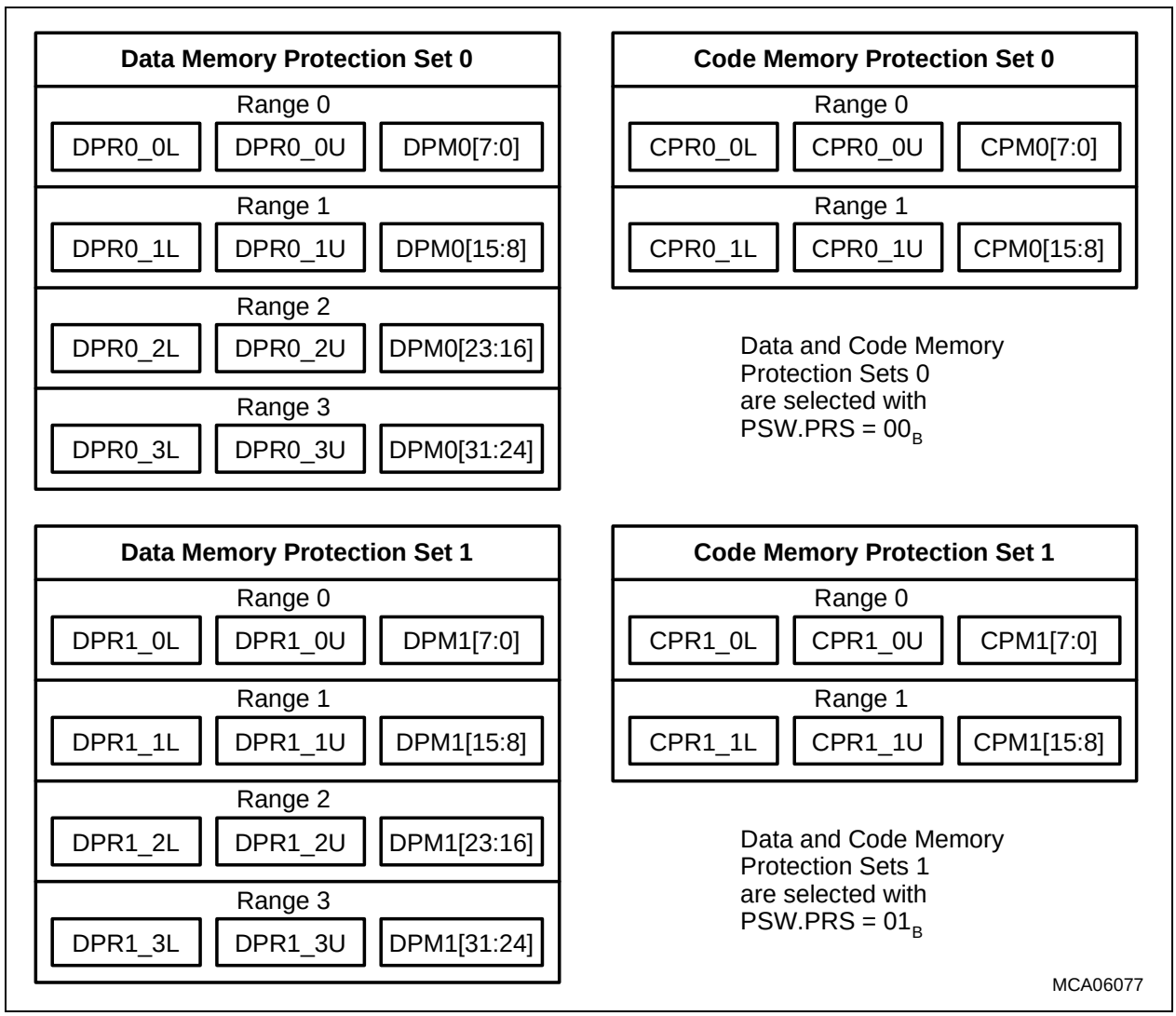


Figure 2-10 Memory Protection Register Sets of the TC1766

Table 2-6 Memory Protection Registers

Register Short Name	Register Long Name	Address
DPR0_0L	Data Segment Protection Register Set 0, Range 0, Lower Boundary	F7E1 C000$_H$
DPR0_0U	Data Segment Protection Register Set 0, Range 0, Upper Boundary	F7E1 C004$_H$
DPR0_1L	Data Segment Protection Register Set 0, Range 1, Lower Boundary	F7E1 C008$_H$
DPR0_1U	Data Segment Protection Register Set 0, Range 1, Upper Boundary	F7E1 C00C$_H$
DPR0_2L	Data Segment Protection Register Set 0, Range 2, Lower Boundary	F7E1 C010$_H$
DPR0_2U	Data Segment Protection Register Set 0, Range 2, Upper Boundary	F7E1 C014$_H$
DPR0_3L	Data Segment Protection Register Set 0, Range 3, Lower Boundary	F7E1 C018$_H$
DPR0_3U	Data Segment Protection Register Set 0, Range 3, Upper Boundary	F7E1 C01C$_H$
DPR1_0L	Data Segment Protection Register Set 1, Range 0, Lower Boundary	F7E1 C400$_H$
DPR1_0U	Data Segment Protection Register Set 1, Range 0, Upper Boundary	F7E1 C404$_H$
DPR1_1L	Data Segment Protection Register Set 1, Range 1, Lower Boundary	F7E1 C408$_H$
DPR1_1U	Data Segment Protection Register Set 1, Range 1, Upper Boundary	F7E1 C40C$_H$
DPR1_2L	Data Segment Protection Register Set 1, Range 2, Lower Boundary	F7E1 C410$_H$
DPR1_2U	Data Segment Protection Register Set 1, Range 2, Upper Boundary	F7E1 C414$_H$
DPR1_3L	Data Segment Protection Register Set 1, Range 3, Lower Boundary	F7E1 C418$_H$
DPR1_3U	Data Segment Protection Register Set 1, Range 3, Upper Boundary	F7E1 C41C$_H$

Table 2-6 Memory Protection Registers (cont'd)

Register Short Name	Register Long Name	Address
CPR0_0L	Code Segment Protection Register Set 0, Range 0, Lower Boundary	F7E1 D000$_H$
CPR0_0U	Code Segment Protection Register Set 0, Range 0, Upper Boundary	F7E1 D004$_H$
CPR0_1L	Code Segment Protection Register Set 0, Range 1, Lower Boundary	F7E1 D008$_H$
CPR0_1U	Code Segment Protection Register Set 0, Range 1, Upper Boundary	F7E1 D00C$_H$
CPR1_0L	Code Segment Protection Register Set 1, Range 0, Lower Boundary	F7E1 D400$_H$
CPR1_0U	Code Segment Protection Register Set 1, Range 0, Upper Boundary	F7E1 D404$_H$
CPR1_1L	Code Segment Protection Register Set 1, Range 1, Lower Boundary	F7E1 D408$_H$
CPR1_1U	Code Segment Protection Register Set 1, Range 1, Upper Boundary	F7E1 D40C$_H$
DPM0	Data Protection Mode Register Set 0	F7E1 E000$_H$
DPM1	Data Protection Mode Register Set 1	F7E1 E080$_H$
CPM0	Code Protection Mode Register Set 0	F7E1 E200$_H$
CPM1	Code Protection Mode Register Set 1	F7E1 E280$_H$

2.4.5.1 Implementation-specific Memory Protection Registers

This section describes the implementation-specific Data and Code Protection Mode Registers that differ from the description in the TriCore 1 Architecture Manual. The non-shaded areas in the CPMx register descriptions define the implementation-specific bits/bit fields. Therefore, the uppermost two bytes of CPMx are of type “Read as 0”.

CPMx (x = 0-1)
Code Protection Mode Register Set x **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
XE 1	0	XS 1	0	BL 1	0	0	BU 1	XE 0	0	XS 0	0	BL 0	0	0	BU 0
rw	r	rw	r	rw	r	r	rw	rw	r	rw	r	rw	r	r	rw

Field	Bits	Type	Description
0	[31:16]	r	**Reserved** Read as 0; should be written with 0. These bits refer to code memory ranges 2 and 3, which are are not available in the TC1766.

2.5 Program Memory Interface (PMI)

Figure 2-11 shows the block diagram of the Program Memory Interface (PMI) of the TC1766.

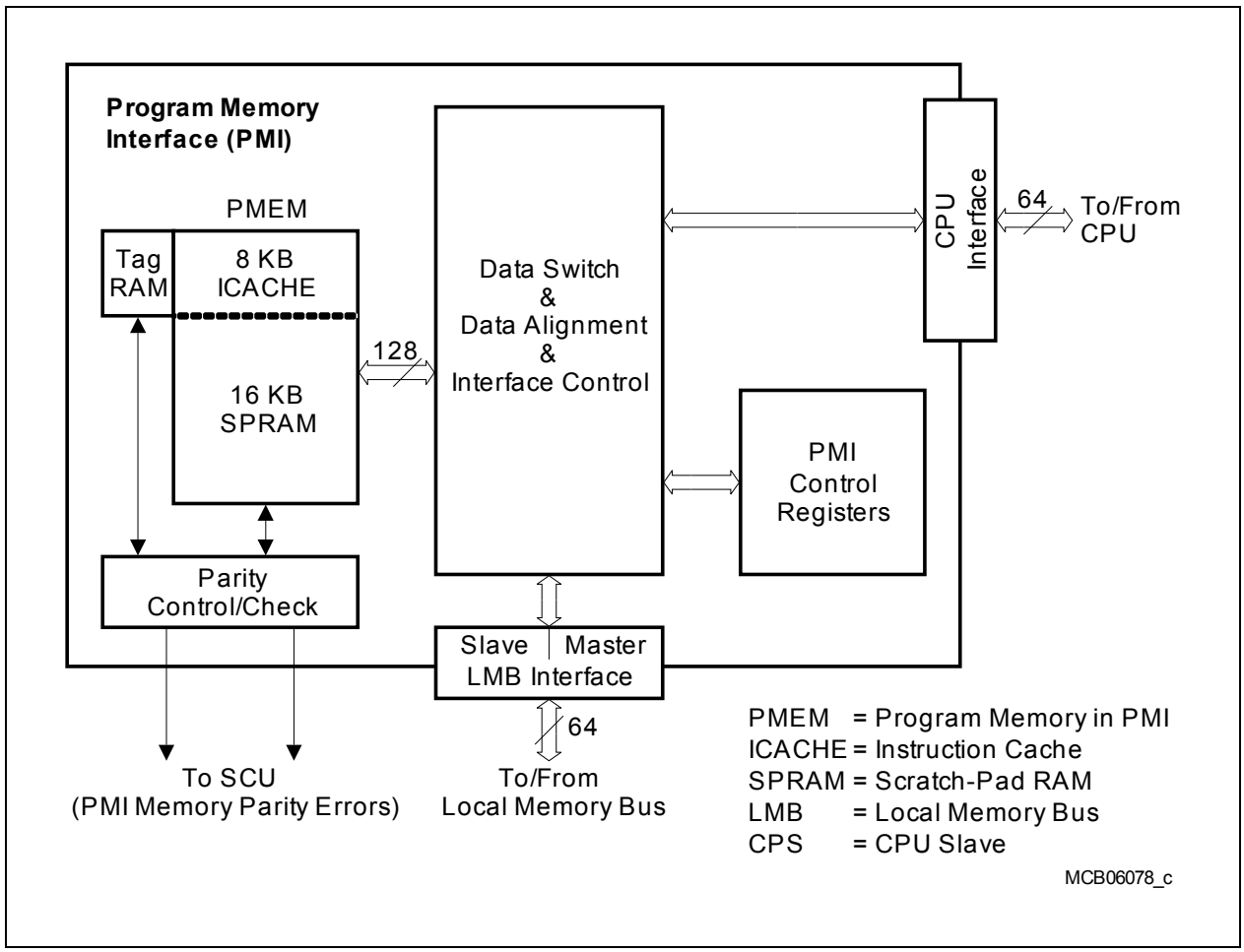


Figure 2-11 PMI Block Diagram

2.5.1 PMI Features

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

- 24 Kbyte memory as:
  - 8 Kbyte instruction cache (ICACHE)
  - 16 Kbyte scratch-pad RAM (SPRAM)
- ICACHE operation features:
  - Two-way set associative cache
  - LRU (Least-Recently Used) replacement algorithm
  - Cache line size = 256 bits (4 double-words)
  - Validity granularity = 4 double-words 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, no line wrap around, streaming to CPU
- CPU interface
  - Supporting unaligned accesses (16-bit aligned)
- Local Memory Bus (LMB) Master Interface to PMU, DMI and LFI-Bridge
- Local Memory Bus (LMB) Slave Interface to scratch-pad RAM
- PMI SRAMs (SPRAM, ICACHE, and cache tag SRAM) are parity protected

2.5.2 LMB Access Priorities

The TC1766 contains a common local memory bus, shared between the Program Memory Interface (PMI), Data Memory Interface (DMI) 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. LFI (Highest Priority)
2. DMI
3. PMI (Lowest Priority)

2.5.3 Scratch-pad RAM

The TC1766 contains a total of 16 Kbyte of scratch-pad RAM. Scratch-pad RAM provides a fast, deterministic program fetch access from the CPU for use by performance critical code sequences.

- CPU program fetch accesses to scratch-pad RAM are never cached in the instruction cache and are always directly targeted to the scratch-pad RAM.
- The scratch-pad RAM has the concept of a scratch-pad RAM “line” (similar to the instruction cache). Scratch-pad 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 scratch-pad 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 scratch-pad RAM line, such that 64-bits would span two scratch-pad RAM lines, then only the instruction half-words up to the end of the scratch-pad RAM line are returned to the CPU.

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

The scratch-pad RAM may also be accessed from the LMB Slave interface by another bus master, such as the Data Memory Interface (DMI). The scratch-pad RAM may be

both read and written from the LMB. In the TC1766, the PMI LMB Slave interface supports all LMB transaction types, with the exception of byte writes. A byte write access to scratch-pad RAM will result in the generation of an LMB Bus error by the PMI.

2.5.4 Instruction Cache

The TC1766 contains an 8 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 256 cache lines, with 256-bits per line. Each ICache line is divided into four double-words (64 bits) with a valid bit for each double-word.

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 an 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 an 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, until the end of the ICache line, without wrapping around, that is the refill size being 1, 2, 3 or 4 double-words. ICache refills are always performed in 64-bit quantities. If the requested half-word is contained within the last 64-bit double-word of a cache line, only this double-word will be refilled. Thus, depending on the location of the critical double-word, the refill sequence will always be from one up to four double-words without wrap-around (the instructions mapping to the refilled cache line which are on addresses lower than that of the requested half-word are not fetched, except for instructions half-words located within the double-word containing the requested half-word). A refill sequence will always affect only one cache line. There is no prefetching of the next cache line.

ICache refills are implemented using LMB transfer sizes, dependent on the number of double-words to be fetched. For the case where the refill size is 3 or 4 double-words, an LMB Burst Transfer 4 (BTR4) transaction is generated (for the case where the refill size is 1 or 2 double-words, single 64-bit or LMB Burst Transfer 2 (BTR2) transactions are generated respectively.

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.CCBYP, to provide a direct instruction fetch path for the CPU Fetch Unit. The default value of PMI_CON0.CCBYP 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.

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 setting the invalidate control bit PMI_CON1.CCINV, then clearing this bit.

2.5.5 PMI Registers

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

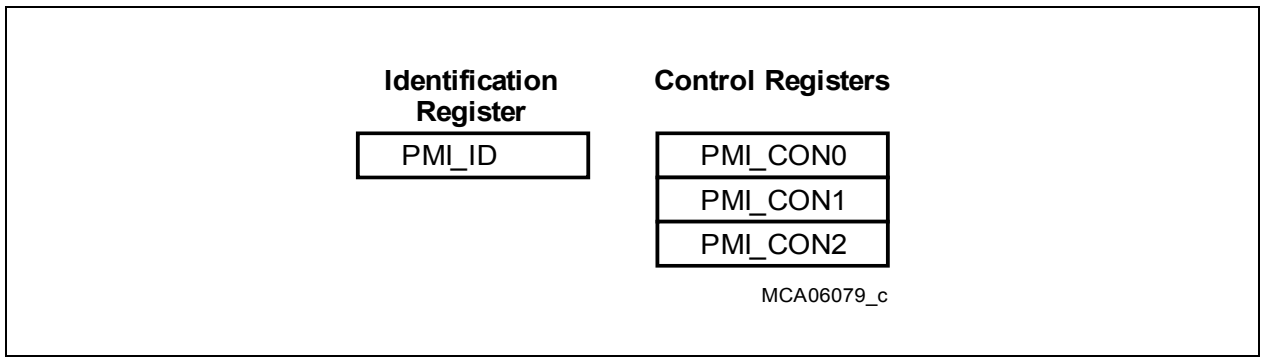


Figure 2-12 PMI Registers

Table 2-7 PMI Registers

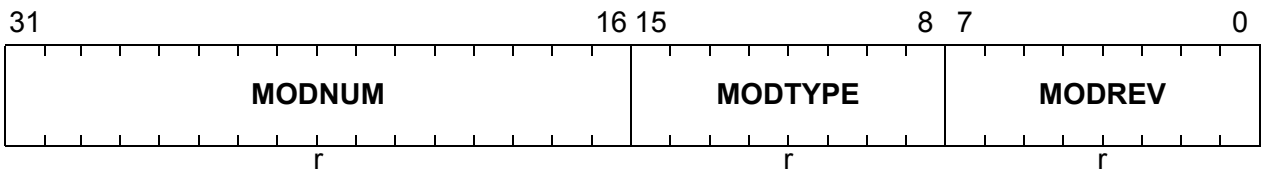
Register Short Name	Register Long Name	Address	Description see
PMI_ID	PMI Module Identification Register	F87F FD08$_H$	Page 2-29
PMI_CON0	PMI Control Register 0	F87F FD10$_H$	Page 2-30
PMI_CON1	PMI Control Register 1	F87F FD14$_H$	Page 2-31
PMI_CON2	PMI Control Register 2	F87F FD18$_H$	Page 2-32

2.5.5.1 PMI Module Identification Register

The PMI Module Identification Register ID contains read-only information about the PMI module version.

PMI_ID
PMI Module Identification Register

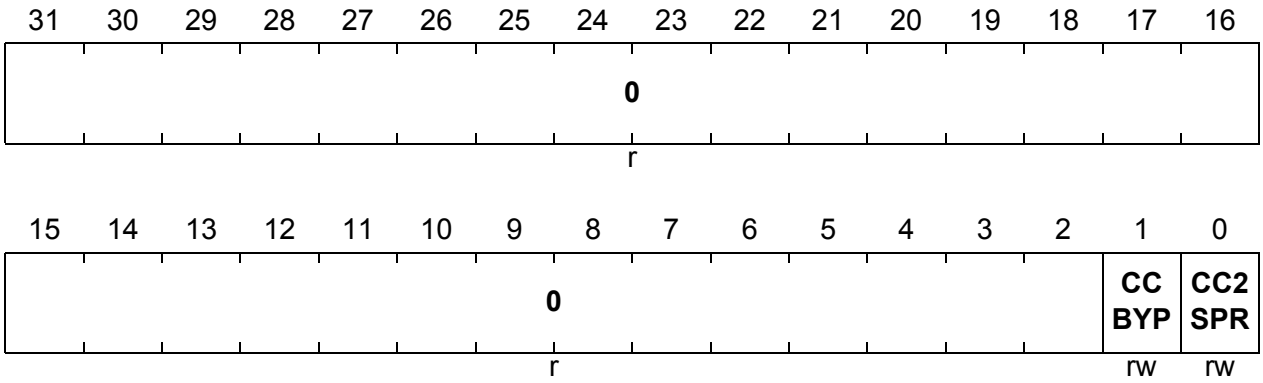
(F87FFD08$_H$) **Reset Value: 000B C0XX$_H$**



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 PMI: 000B$_H$

2.5.5.2 PMI Control Register 0

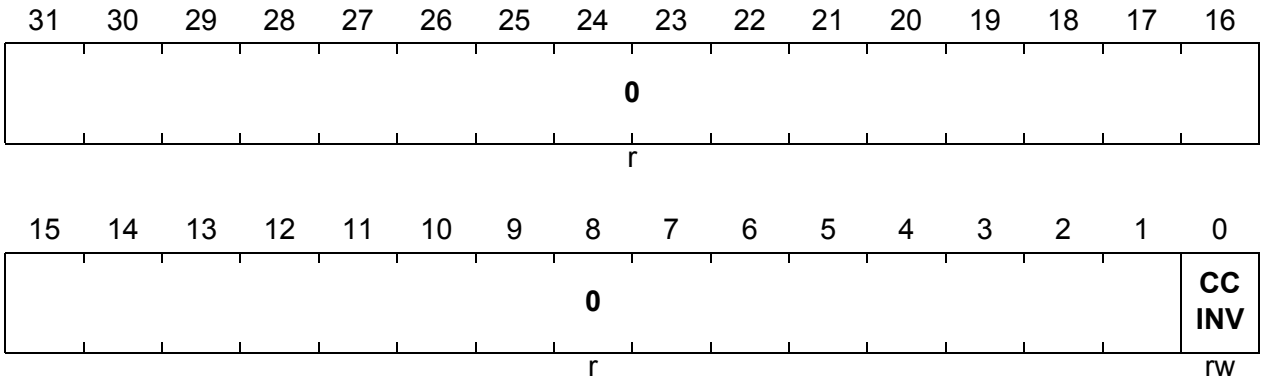
PMI_CON0
PMI Control Register 0 **(F87FFD10$_H$)** **Reset Value: 0000 0002$_H$**



Field	Bits	Type	Description
CC2SPR	0	rw	**Code Cache Memory to SPR** This bit is used for cache test mode purposes. CC2PR must be written with 0. Setting it to 1 may lead to unpredictable program behavior.
CCBYP	1	rw	**Code Cache Bypass** 0_B Cache enabled 1_B Cache bypassed (disabled)
0	[31:2]	r	**Reserved** Returns 0 when read; should be written with 0.

2.5.5.3 PMI Control Register 1

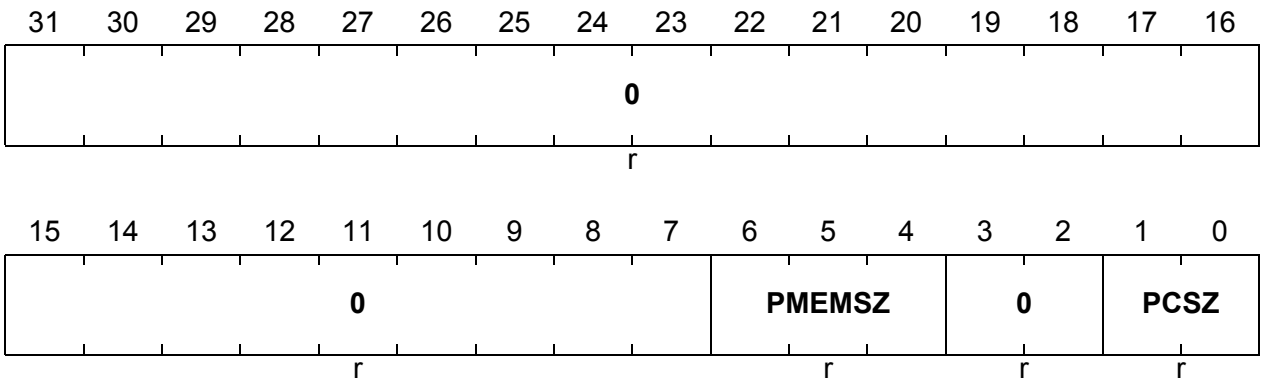
PMI_CON1
PMI Control Register 1 **(F87FFD14$_H$)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
CCINV	0	rw	**Code Cache Invalidate** 0_B Normal code cache (ICACHE) operation 1_B All cache lines are invalidated As long as CCINV is set, all instruction fetch accesses generate a cache refill. It is recommended that CCINV be kept set until ICACHE coherency is guaranteed.
0	[31:1]	r	**Reserved** Returns 0 when read; should be written with 0.

2.5.5.4 PMI Control Register 2

PMI_CON2
PMI Control Register 2 **(F87FFD18$_H$)** **Reset Value: 0000 0022$_H$**



Field	Bits	Type	Description
PCSZ	[1:0]	r	**Program Cache Size** This bit field indicates the ICACHE size and TAGRAM configuration of the PMI. The TC1766 has a fixed ICACHE size of 8 Kbyte. Therefore, PCSZ is always read as 10_B. 10_B 8 Kbyte cache
PMEMSZ	[6:4]	r	**Program Memory Size (ICACHE + SPRAM)** This bit field indicates the ICACHE plus SPRAM size of the PMI program memory. The TC1766 has a fixed ICACHE and SPRAM size of 24 Kbyte. PMEMSZ is always read as 010_B. 010_B 24 Kbyte program memory
0	[3:2], [31:7]	r	**Reserved** Returns 0 when read; should be written with 0.

2.6 Data Memory Interface (DMI)

Figure 2-13 shows the block diagram of the Data Memory Interface (DMI) of the TC1766.

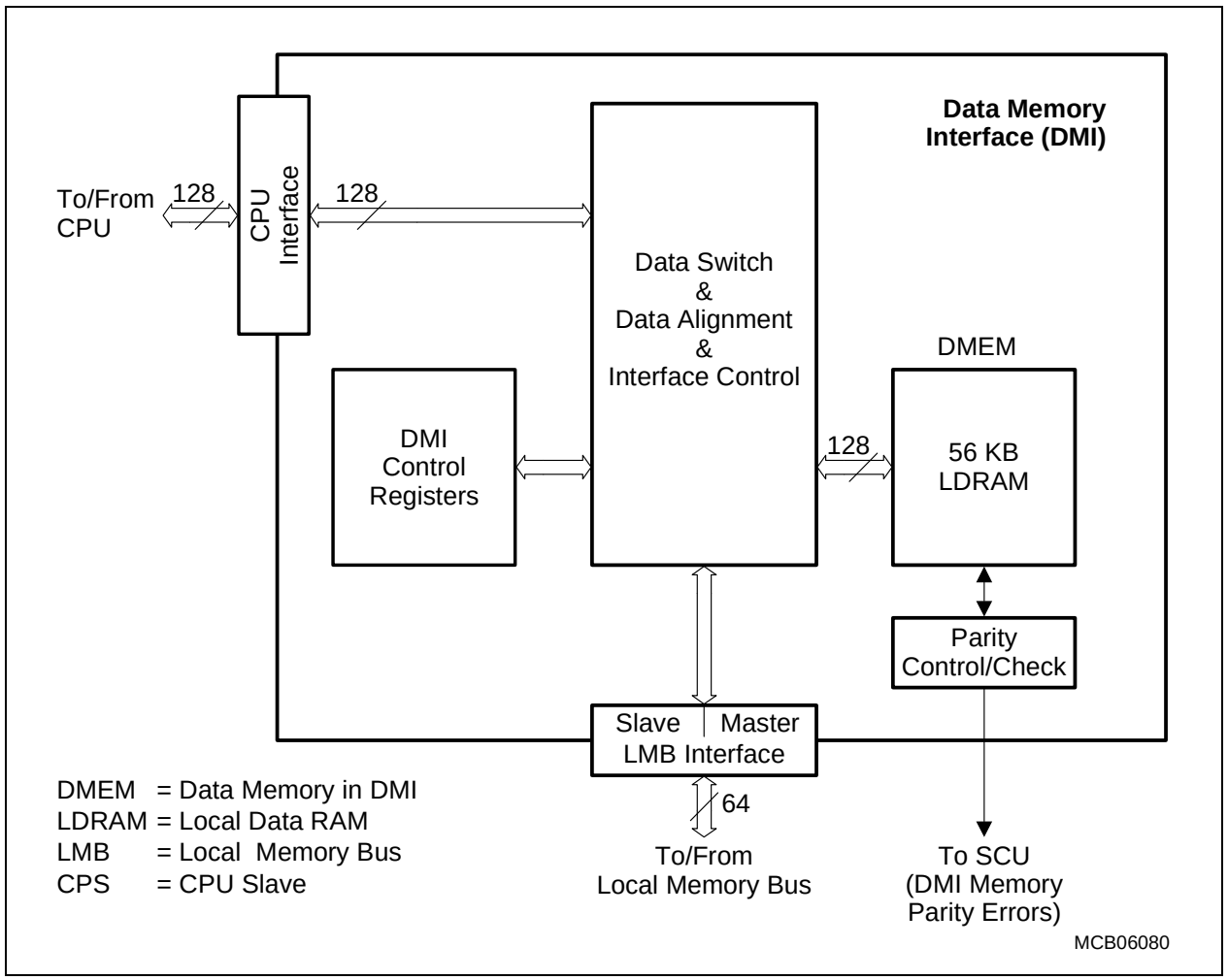


Figure 2-13 DMI Block Diagram

2.6.1 DMI Features

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

- 56 Kbyte data memory:
  - 56 Kbyte local data memory (LDRAM)
  - LDRAM is parity-protected
- CPU interface
  - supporting unaligned accesses (16-bit aligned) with a minimum penalty of one cycle for unaligned accesses crossing 2 lines
- Local Memory Bus (LMB) Master interface
- Local Memory Bus (LMB) Slave interface to LDRAM

2.6.2 LMB Access Priorities

See **“LMB Access Priorities” on Page 2-25**.

2.6.3 LDRAM

The TC1766 contains a total of 56 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.

2.6.4 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 implemented for a given device. 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.

2.6.5 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.

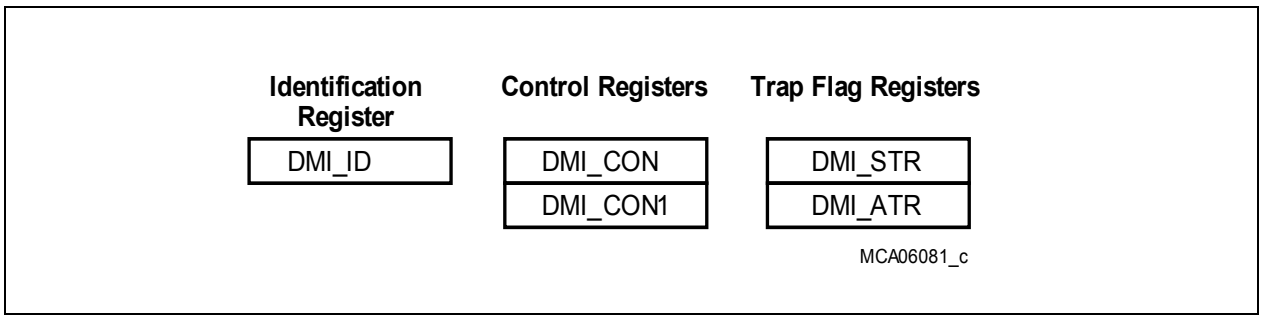


Figure 2-14 DMI Registers

Table 2-8 DMI Registers

Register Short Name	Register Long Name	Offset Address	Description
DMI_ID	DMI Module Identification Register	F87F FC08$_H$	Page 2-36
DMI_CON	DMI Control Register	F87F FC10$_H$	Page 2-36
DMI_STR	DMI Synchronous Trap Flag Register	F87F FC18$_H$	Page 2-39
DMI_ATR	DMI Asynchronous Trap Flag Register	F87F FC20$_H$	Page 2-40
DMI_CON1	DMI Control Register 1	F87F FC28$_H$	Page 2-38

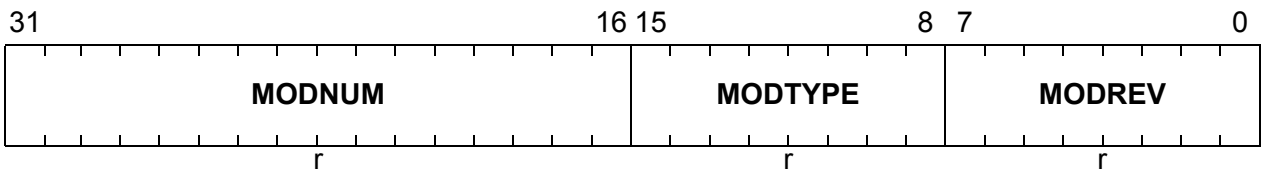
Access to DMI control registers must only be made with double-word aligned word accesses. An access not conforming to this rule, or an access that does not follow the specified privilege mode (Supervisor mode, Endinit-protection), or a write access to a read-only register, will lead to a bus error if 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.

2.6.5.1 DMI Register Description

The DMI Module Identification Register ID contains read-only information about the DMI module version.

DMI_ID
DMI Module Identification Register

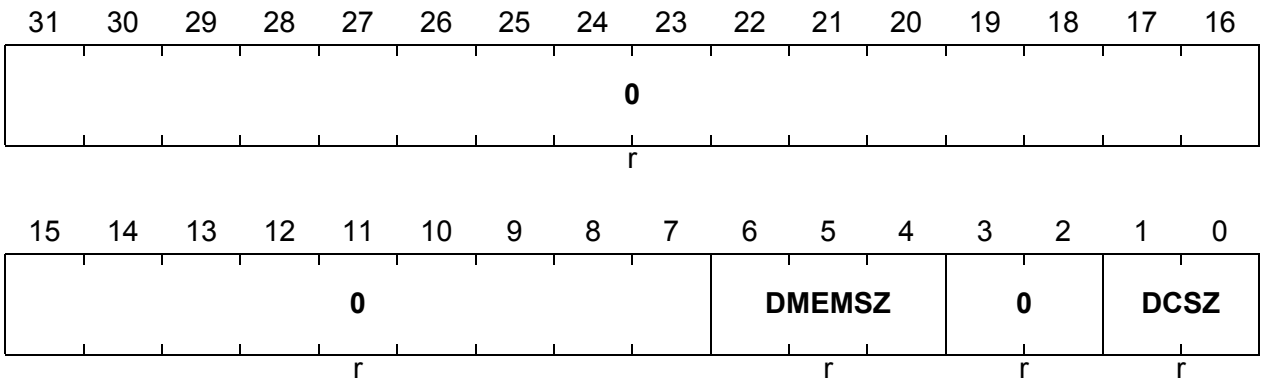
(F87FFC08$_H$) **Reset Value: 0008 C0XX$_H$**



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 DMI: 0008_H

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

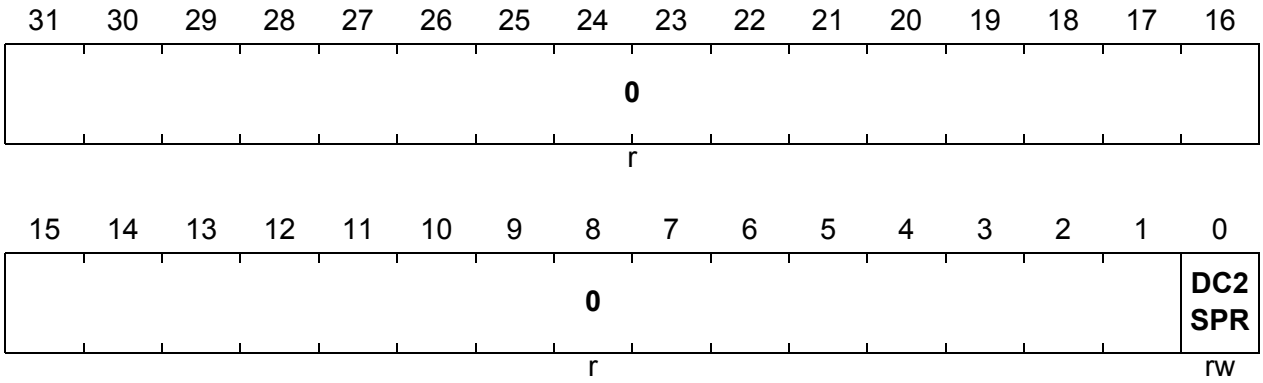
DMI_CON
DMI Control Register **(F87FFC10$_H$)** **Reset Value: 0000 0060$_H$**



Field	Bits	Type	Description
DCSZ	[1:0]	r	**Data Cache Size** This bit field indicates the DMI data cache configuration. In the TC1766 no data cache is available, therefore DCSZ is always read as 00_B. 00_B No cache available
DMEMSZ	[6:4]	r	**Data Memory Size** This bit field indicates the DMI data memory size. In the TC1766 DMEMSZ is always read as 110_B. 110_B 56 Kbyte DMI data memory
0	[3:2], [31:7]	r	**Reserved** Returns 0 when read.

The DMI control register 1 is required for data cache test purposes (where applicable).

DMI_CON1
DMI Control Register 1 **(F87FFC28$_H$)** **Reset Value: 0000 0000$_H$**

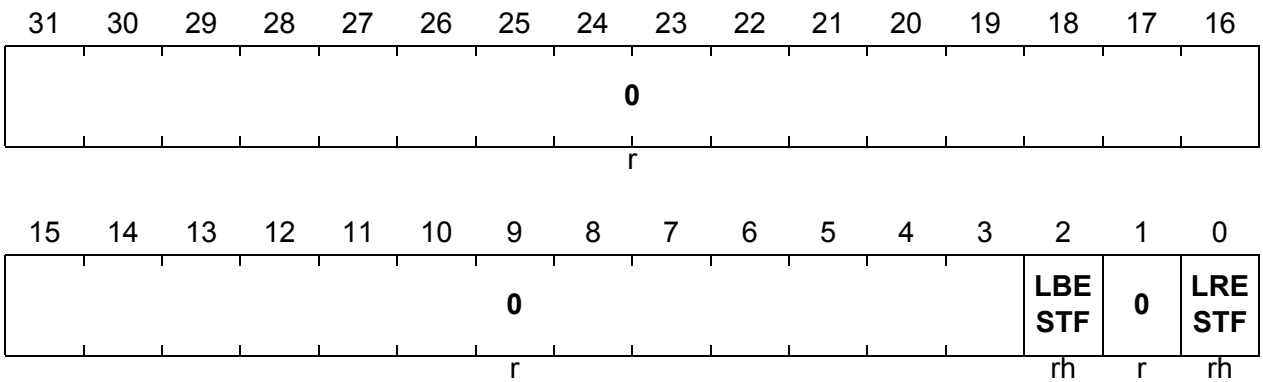


Field	Bits	Type	Description
DC2SPR	0	rw	**Cache Test Mode Enable** This bit must always be written with 0. Setting to 1 will have no effect in TC1766.
0	[31:1]	r	**Reserved** Returns 0 when read; should be written with 0.

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

(F87FFC18$_H$) **Reset Value: 0000 0000$_H$**

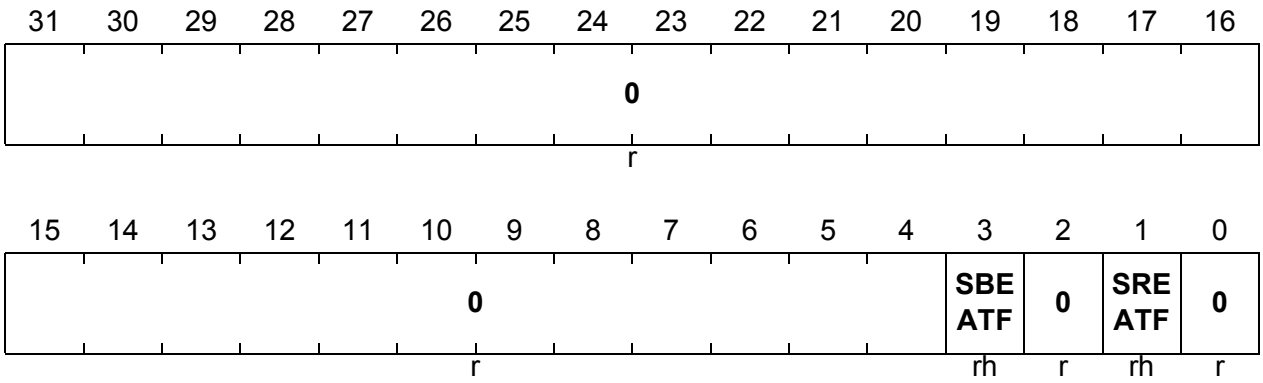


Field	Bits	Type	Description
LRESTF	0	rh	**Load Range Synchronous Error** 0_B No error 1_B Load range synchronous error has occurred
LBESTF	2	rh	**Bus Load Synchronous Error** 0_B No error 1_B Bus load synchronous error has occurred
0	1, [31:3]	r	**Reserved** Returns 0 when read.

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

($F87FFC20_H$) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
SREATF	1	rh	**Store Range Asynchronous Error** 0_B No error 1_B Store range asynchronous error has occurred
SBEATF	3	rh	**LMB Bus Store Asynchronous Error** 0_B No error 1_B Bus store asynchronous error has occurred
0	0, 2, [31:4]	r	**Reserved** Returns 0 when read.

2.7 Instruction Timing

This section gives information on 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.

2.7.1 Integer-Pipeline Instructions

0

2.7.1.1 Simple Arithmetic Instruction Timings

Each instruction is single issued

Table 2-9 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 2-9 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 2-9 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 2-9 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	**DVINIT.WS**	1	1
BSPLIT	1	1	**DVINIT.WU**	1	1
DVADJ	1	1	**DVSTEP.S**	4	4
DVINIT	1	1	**DVSTEP.U**	4	4
DVINIT.U	1	1	**IXMAX**	1	1
DVINIT.B	1	1	**IXMAX.U**	1	1
DVINIT.H	1	1	**IXMIN**	1	1
DVINIT.BS	1	1	**IXMIN.U**	1	1
DVINIT.BU	1	1	**PACK**	1	1
DVINIT.HS	1	1	**PARITY**	1	1
DVINIT.HU	1	1	**UNPACK**	1	1

2.7.1.2 Multiply Instruction Timings

Each instruction is single issued.

Table 2-10 Multiple Instruction Timing

Instruction	Result Latency	Repeat Rate	Instruction	Result Latency	Repeat Rate
IP Arithmetic Instructions					
MUL	3	2	**MUL.H**	2	1
MUL.U	3	2	**MUL.Q**	1/2/3	1/1/2
MULS	3	2	**MULM.H**	2	1
MULS.U	3	2	**MULR.H**	2	1
			MULR.Q	2	1

For MUL.Q Instruction:

	Result Latency	Repeat Rate
16 × 16	1	1
16 × 32	2	1
32 × 32	3	2

2.7.1.3 MAC Instruction Timings

Each instruction is single issued.

Table 2-11 MAC Instruction Timing

Instruction	Result Latency	Repeat Rate	Instruction	Result Latency	Repeat Rate
IP Arithmetic Instructions					
MADD	3	2	**MSUB**	3	2
MADD.U	3	2	**MSUB.U**	3	2
MADDS	3	2	**MSUBS**	3	2
MADDS.U	3	2	**MSUBS.U**	3	2
MADD.H	2	1	**MSUB.H**	2	1
MADD.Q	2/3	1/2	**MSUB.Q**	2/3	1/2
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	2/3	1/2	**MSUBS.Q**	2/3	1/2
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	3	2

2.7.1.4 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 2-12 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

2.7.2 Load-Store Pipeline Instructions

This section summarizes the Load-Store Pipeline instructions.

2.7.2.1 Address Arithmetic Timing

Each instruction is single issued.

Table 2-13 Address Arithmetic Instruction Timing

Instruction	Result Latency	Repeat Rate	Instruction	Result Latency	Repeat Rate
LS 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**[3)]	–	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.

2.7.2.2 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.

Table 2-14 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)]	2-5	2-5	**SYSCALL**[1)]	–	2-5
CALLA[1)]	2-5	2-5	**SVLCX**[1)]	–	4-9
CALLI[1)]	2-5	2-5	**RSLCX**[1)]	–	4
RET[1)]	–	2-5	**RFE**[1)]	–	2-5
BISR[1)]	–	4-9	**RFM**[2)]	–	–
Loop Instructions					
LOOP[3)]	2/1/3	2/1/3	**LOOPU**[3)]	2/1/3	2/1/3

1) Latency of CSA related instructions varies according to preceding instruction and status of the shadow register file. Average latency is ~2.7 cycles

2) Not strictly a CSA operation, but retrieves from memory a subset of context information and changes control flow in a similar manner.

3) 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

2.7.2.3 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 2-15 Load Instruction Timing

Instruction	Addr. Latency	Result Latency	Repeat Rate	Instruction	Addr. 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.

2.7.2.4 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 2-16 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			
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			
ST.Q	1	1	**LDMST**	2	2

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.

2.7.3 Floating Point Pipeline Timings

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

- Each instruction is single issued.

Table 2-17 Floating Point Instruction Timing

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

3 Clock System and Control

This chapter describes the TC1766 clock system. Topics covered include clock gating, clock domains, clock generation, the operation of clock circuitry, boot-time operation, fail-safe operation, and clock control registers.

3.1 Overview

The TC1766 clock system performs the following functions:

- Acquires and buffers incoming clock signals to create a master clock frequency
- Distributes in-phase synchronized clock signals throughout the TC1766's entire clock tree
- Divides a system master clock frequency into lower frequencies required by the different modules for operation
- Dynamically reduces power consumption during operation of functional units
- Statically reduces power consumption through programmable power-saving modes
- Reduces electromagnetic interference (EMI) by switching off unused modules

The clock system must be operational before the TC1766 can function, so it contains special logic to handle power-up and reset operations. Its services are fundamental to the operation of the entire system, so it contains special fail-safe logic.

Figure 3-1 shows the structure of the TC1766 clock system. The system clock f_{SYS} is generated by the oscillator circuit and the PLL (phase-locked loop) unit. Each peripheral module operates with its module clocks f_{CLC} and f_{MOD}. Suffix "MOD" is a place holder for the module name, e.g. f_{ASC0}.

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 unit on **Table 3-9** on **Page 3-39**.

All clock control registers CLC and the fractional divider registers FDR are Endinit-protected (details see **Page 14-3**).

Features of the TC1766 Clock System

- PLL operation for multiplying clock source by different factors
- Direct drive capability for direct clocking
- Comfortable state machine for secure switching between basic PLL, direct or prescaler operation
- Sleep and Power-Down Mode support

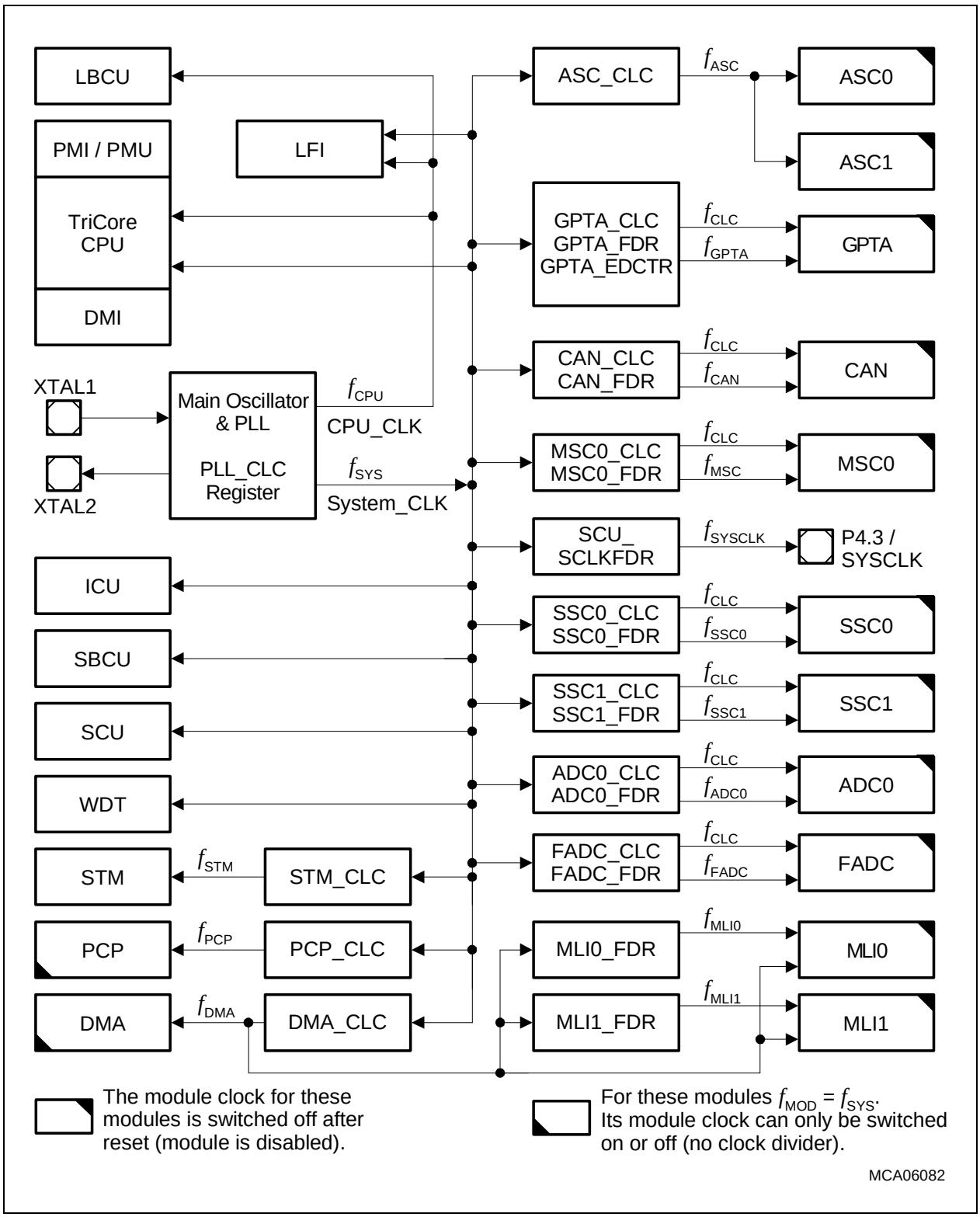


Figure 3-1 TC1766 Clocking System

3.2 Clock Generation Unit

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 degenerating 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.

The Clock Generation Unit (CGU) in the TC1766, shown in **Figure 3-2**, consists basically of an oscillator circuit and a Phase-Locked Loop (PLL). The operation of the CGU is controlled by two registers, OSC_CON and PLL_CLC, which are located in the System Control Unit (SCU).

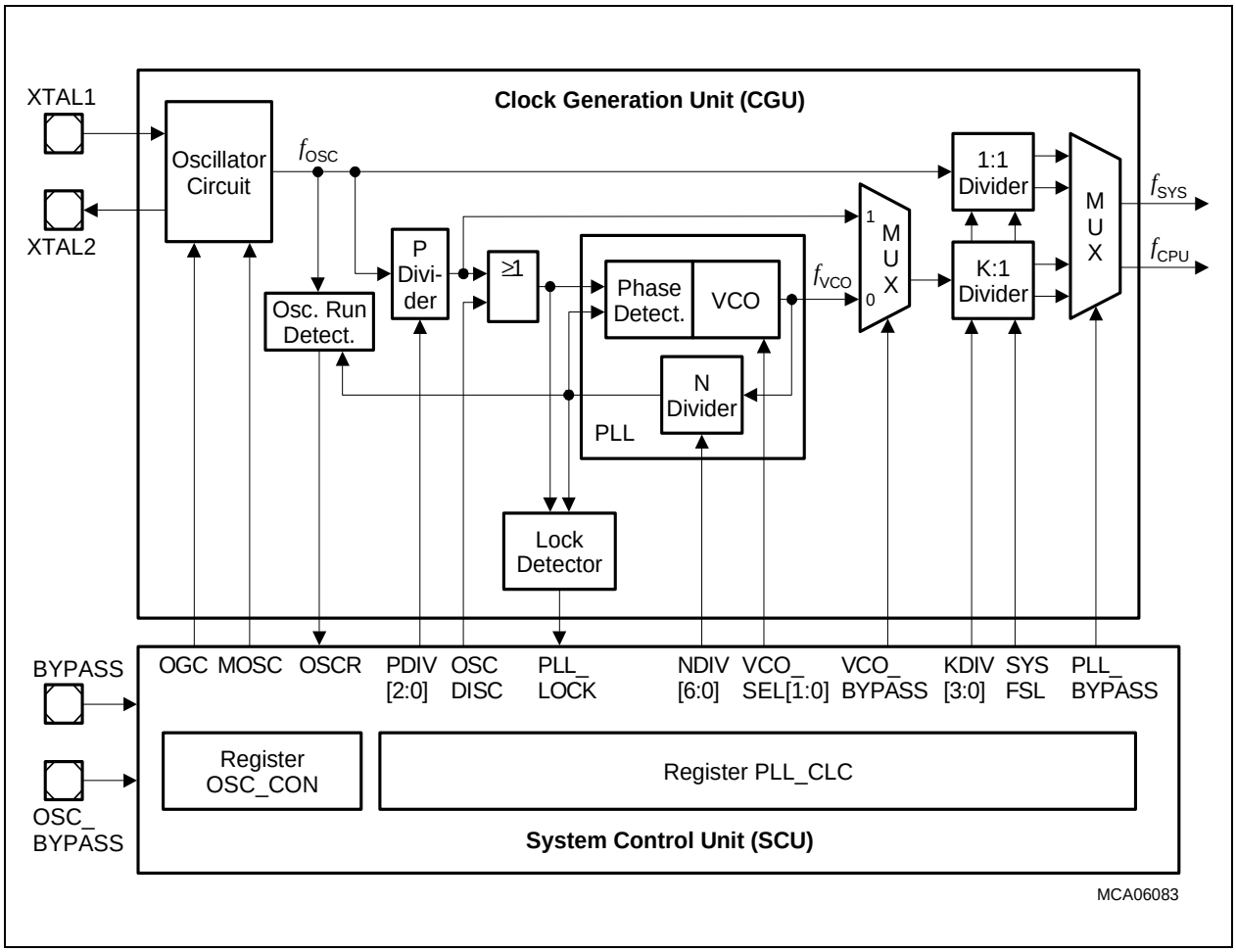


Figure 3-2 CGU Detailed Block Diagram

3.2.1 Main Oscillator Circuit

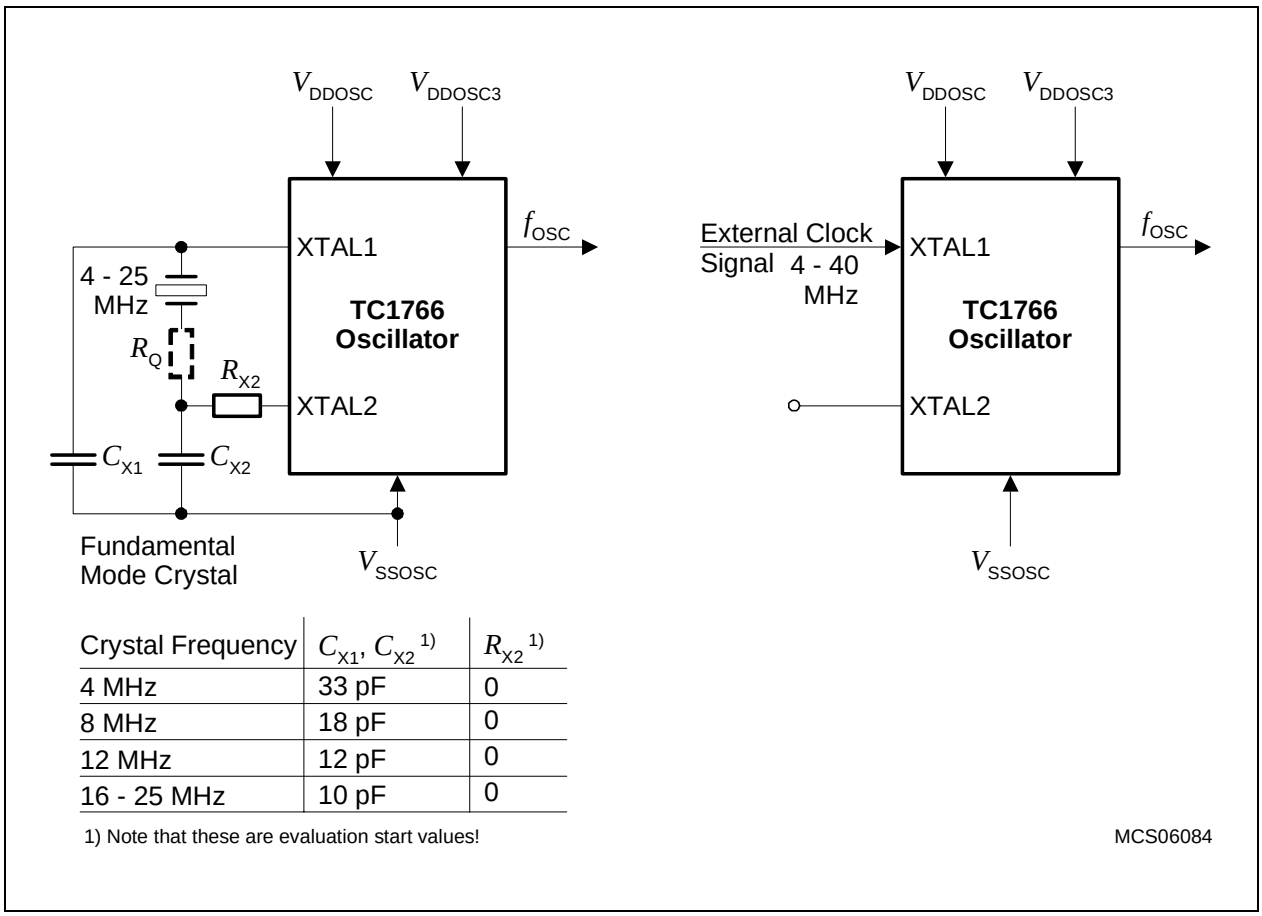
The oscillator circuit, a Pierce oscillator, is designed to work with both, an external crystal oscillator or an external stable clock source. It basically consists of an inverting amplifier and a feedback element with XTAL1 as input, and XTAL2 as output.

When using a crystal, a proper external oscillator circuitry must be connected to both pins, XTAL1 and XTAL2. The crystal frequency can be within the range of 4 MHz to 25 MHz. Additionally, it is necessary to have two load capacitances C_{X1} and C_{X2}, and depending on the crystal type, a series resistor R_{X2}, 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_{X1} and C_{X2} values shown in **Figure 3-3** 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.

When using an external clock signal, the signal must be connected to XTAL1. XTAL2 is left open (unconnected). The external clock frequency can be in the range of 0 - 40 MHz if the PLL is bypassed, and 4 - 40 MHz if the PLL is used.

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

Figure 3-3 shows the recommended external oscillator circuitries for both operating modes, external crystal mode and external input clock mode.



Crystal Frequency	C_{X1}, C_{X2} 1)	R_{X2} 1)
4 MHz	33 pF	0
8 MHz	18 pF	0
12 MHz	12 pF	0
16 - 25 MHz	10 pF	0

1) Note that these are evaluation start values!

Figure 3-3 Main Oscillator Circuitry

3.2.1.1 Oscillator Bypass Mode

Especially for test purposes, the oscillator circuitry can be bypassed (disabled). In this oscillator bypass mode, the clock line f_{OSC} is disconnected from the oscillator circuitry and directly connected to the XTAL1 pin. The oscillator bypass mode is controlled by bit OSC_CON.MOSC.

- **Normal Mode** (OSC_CON.MOSC = 0):
  f_{OSC} is derived from the crystal or from an external clock signal
- **Oscillator Bypass Mode** (OSC_CON.MOSC = 1):
  The oscillator is bypassed and f_{OSC} is directly derived from an external clock signal which is applied to XTAL1.

The MOSC bit can be set in two ways:

- By software: Writing a 1 to bit MOSC of register OSC_CON.
- By hardware at a power-on reset operation: If pin BYPASS = 1, the state of the pin P3.1/TXD1A is latched with the rising edge of PORST and determines the inverted state of the MOSC bit. If P3.1 = 0 at the rising edge of PORST, MOSC is set and the oscillator bypass mode is enabled (see also **Figure 3-1**).

3.2.1.2 Oscillator Run Detection

The oscillator run detection logic indicates during oscillator start-up after a power-on operation whether the oscillator is already running or if an emergency operation with PLL base frequency has to be started. When the oscillator run condition is met, bit OSC_CON.OSCR is set and the output clock f_{OSC} is enabled to supply the clock signal to the rest of the system. **Figure 3-4** shows the oscillator run detection logic.

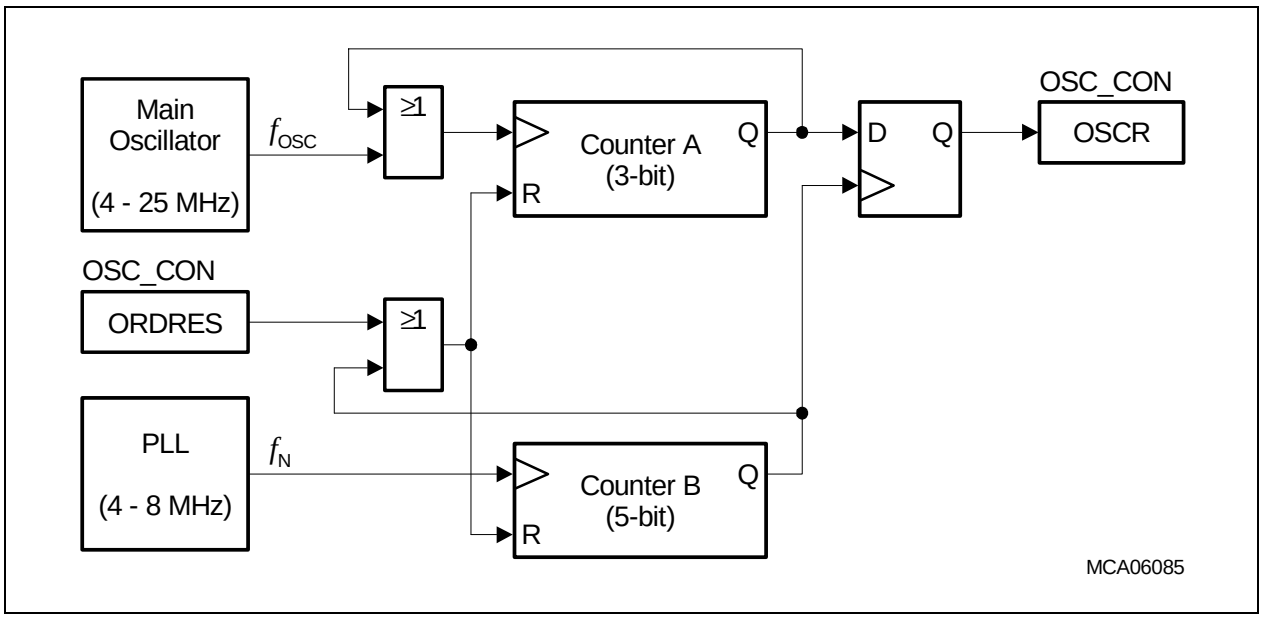


Figure 3-4 Oscillator Run Detection Circuitry

The oscillator run detection consists of two counters, Counter A and B. The 3-bit Counter A is running with the oscillator frequency and stops at its terminal count value. The 5-bit Counter B is running at f_N, the divided (N-Divider) VCO clock frequency. Always at the terminal count of Counter B the state of counter A is latched in a flip-flop, bit OSCR is updated, and both counters a reset. This means, if Counter A does not reaches its terminal count value (8 f_{OSC} clock periods) within a counter period of Counter B (32 f_N clock periods), the oscillator is designated as “not running” by OSCR = 0. If Counter A is at its terminal count value at the end of the counter period of Counter B, the “running” state (OSCR = 1) is detected.

The circuit can start without a reset and becomes defined after at least 32 pulses of counter B. Setting bit OSC_CON.ORDRES makes it possible to start the oscillator run detection during normal operation of the TC1766, e.g. in case of a PLL loss-of-lock condition.

3.2.1.3 Oscillator Gain Control

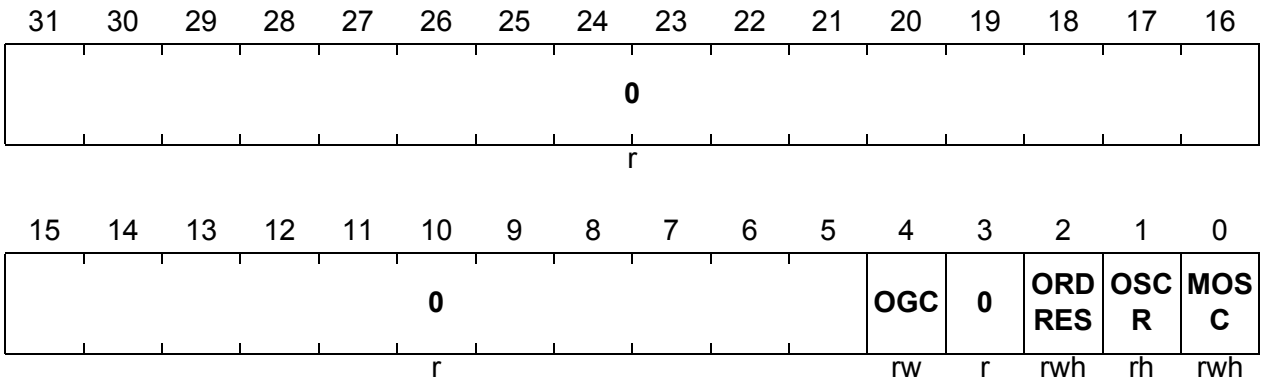
The oscillator starts with a high drive level (gain) during and after a power-on reset to ensure safe start-up behavior in the beginning (force the crystal oscillation). When a stable oscillation has been reached after oscillation start-up, the gain of the oscillator can be reduced. This reduces the oscillator's power consumption, which is especially important in the power saving modes. This gain reduction is selected by OSC_CON.OGC = 1.

Note: Oscillator measurement (margin or negative resistance and XTAL1 input amplitude) for the oscillator crystal system must be executed also with reduced gain.

3.2.1.4 Oscillator Control Register

Note: Register OSC_CON is Endinit-protected.

OSC_CON
Oscillator Control Register **($F0000018_H$)** **Reset Value: see Table 3-1**



Field	Bits	Type	Description
MOSC	0	rwh	**Main Oscillator Test Mode** This bit determines the mode of the main oscillator. 0_B The oscillator is running. The oscillator signal of the main oscillator or an external clock input signal is used as f_{OSC}. 1_B The oscillator circuitry is bypassed. An external clock input signal at XTAL1 must be provided and is used as f_{OSC}. It is 1 if the BYPASS pin is high and TXD0A is zero. Its state is latched with the rising edge of $\overline{PORST}$.
OSCR	1	rh	**Oscillator Run Status Bit** This bit shows the state of the oscillator run state. 0_B The oscillator is not running. 1_B The oscillator is running. The OSCR bit is valid always after a power-on reset operation. After an oscillator run detection reset (setting OSC_CON.ORDRES), OSCR is invalid up to a maximum of 64 f_N clock cycles.

Field	Bits	Type	Description
ORDRES	2	rwh	**Oscillator Run Detection Reset** This bit allows the oscillator run detection to start during normal operating mode. 0_B No operation 1_B The oscillator run detection logic is reset and restarted. After ORDRES has been set, it becomes automatically reset by hardware.
OGC	4	rw	**Oscillator Gain Control** This bit determines the main oscillator gain. 0_B High gain is selected (default after reset). 1_B Low gain is selected.
0	3, [31:5]	r	**Reserved** Read as 0; should be written with 0.

Table 3-1 Reset Values of Register OSC_CON

Condition		Function	Reset Values
BYPASS	**TXD0A**		
0	X	The system is driven by the PLL clock which is derived from the oscillator clock.	$0000\ 0000_H$
1	0	The system is driven directly by the external clock which is applied to XTAL1.	$0000\ 0001_H$
	1	The system is driven by the crystal oscillator clock or the external clock passed through the oscillator.	$0000\ 0000_H$

3.2.2 Phase Looked Loop (PLL) Circuitry

The PLL is a main component of the CGU that is dedicated to generate the CPU and system clock inside the TC1766. The PLL basically converts a low-frequency external clock signal into high-speed internal CPU and system clocks for maximum performance.

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 clock signals and accordingly controls the frequency of the VCO (f_{VCO}). During start-up, the VCO increases its frequency until the divided feedback clock matches the external clock frequency. A PLL lock detection unit monitors and signals this condition. The phase detection logic continues to monitor the two clock signals and adjusts the VCO clock if required. The CGU output clocks f_{CPU} and f_{SYS} are derived from the VCO clock by the K-Divider.

3.2.2.1 Clock Source Control

The CPU clock f_{CPU} and the system clock f_{SYS} are generated from f_{OSC} in one of four hardware/software selectable modes:

- Direct Drive Mode (PLL Bypass Mode)
- VCO Bypass Mode (Prescaler Mode)
- PLL Mode
- PLL Base Mode

Direct Drive Mode (PLL Bypass Operation)

In Direct Drive Mode, the PLL is bypassed and the CGU clock outputs are directly fed from the clock signal f_{OSC}, i.e. $f_{CPU} = f_{OSC}$ and $f_{SYS} = f_{OSC}$. This mode can be only selected by hardware when pin BYPASS = 1 during the rising edge of $\overline{\text{PORST}}$.

VCO Bypass Mode (Prescaler Mode)

In VCO Bypass Mode, f_{CPU} and f_{SYS} are derived from f_{OSC} by the two divider stages, P-Divider and K-Divider. This mode is selected by setting PLL_CLC.VCOBYP = 1. The system clock f_{SYS} equals to f_{CPU}.

$$f_{CPU} = \frac{1}{P \times K} \times f_{OSC}$$
$$f_{SYS} = f_{CPU} \quad (3.1)$$

PLL Mode

In PLL Mode, the PLL is running. The VCO clock f_{VCO} is derived from f_{OSC}, divided by the P factor, multiplied by the PLL (N-Divider). This mode is selected by setting PLL_CLC.VCOBYP = 0. Further, f_{CPU} and f_{SYS} are derived from f_{VCO} by the K-Divider. The system clock f_{SYS} equals to f_{CPU}.

$$f_{CPU} = \frac{N}{P \times K} \times f_{OSC}$$
$$f_{SYS} = f_{CPU} \quad (3.2)$$

PLL Base Mode

In PLL Base Mode, the PLL is running at its VCO base frequency and f_{CPU} and f_{SYS} are derived from f_{VCO} only by the K-Divider. In this mode, PLL_CLC.VCOBYP must be 0 and pin BYPASS must have been latched as 0 at the end of the last power-on reset operation. In this mode, the system clock f_{SYS} equals to f_{CPU}.

$$f_{CPU} = \frac{1}{K} \times f_{VCObase}$$
$$f_{SYS} = f_{CPU} \quad (3.3)$$

3.2.2.2 PLL Parameters

As shown in **Equation (3.1)** to **Equation (3.3)**, the PLL operation depends on the setup of up to four main PLL parameters: P-Divider, N-Divider, K-Divider, and VCO range selection.

P-Divider

The P-Divider divides the oscillator clock f_{OSC} by factor P for the PLL input clock f_P. **Table 3-2** shows the P factor values of the P-Divider which are selected by programming the PLL_CLC.PDIV bit field. It also lists the resulting f_P frequency for some dedicated values of f_{OSC} but the complete range of 4 to 40 MHz can be used for f_{OSC}. Note that the P-Divider factor is always PLL_CLC.PDIV+1.

Table 3-2 P-Divider Selections

PLL_CLC. PDIV	P-Divider: P = PDIV+1	Resulting f_P Frequency (in MHz) for				
		f_{osc} = 4 MHz	f_{osc} = 10 MHz	f_{osc} = 20 MHz	f_{osc} = 30 MHz	f_{osc} = 40 MHz
0	1	4	10	20	30	40
1	2	2	5	10	15	20
2	3	1.33	3.33	6.67	10	13.3
…	…	…	…	…	…	…
6	7	0.57	1.43	2.857	4.286	5.7
7	8	0.5	1.25	2.5	3.75	5

N-Divider

The N-Divider in the feedback path of the PLL divides the VCO clock f_{VCO} by factor N for the N-divider output clock f_N. This feedback clock is used as input clock for the PLL phase detection unit, which compares it with the PLL input clock f_P. The phase detector determines the difference between its two input clocks f_P and f_N and accordingly regulates the frequency of the VCO output clock f_{VCO}.

Table 3-3 shows the N factor values of the N-Divider which are selected by programming the PLL_CLC.NDIV bit field. It also lists the resulting N-divider output clock f_N depending on N and dedicated VCO frequencies. Note that the N-Divider factor is always PLL_CLC.NDIV+1. For proper operation of the PLL, only N-Divider values of 20 to 100 are allowed.

Table 3-3 N-Divider Selections

PLL_CLC.NDIV [1]	N-Divider: N = NDIV+1 [1]	Resulting f_N Frequency (in MHz) for			
		f_{VCO} = 400 MHz	f_{VCO} = 500 MHz	f_{VCO} = 600 MHz	f_{VCO} = 700 MHz
≤ 18	≤ 19	Not allowed			
19	20	20	25	30	35
20	21	19.05	23.81	28.57	33.33
21	22	18.18	22.73	27.27	31.82
…	…	…	…	…	…
97	98	4.08	5.10	6.12	7.14
98	99	4.04	5.05	6.06	7.07
99[2]	100	4	5	6	7
≥ 100	≥ 101	Not allowed			

1) These columns include decimal values.

2) This is the default value after a power-on reset.

K-Divider

The K-Divider divides the VCO clock f_{VCO} by factor K for the CGU output clocks f_{CPU} (and f_{SYS}). **Table 3-4** shows the K factor values of the K-Divider that are selected by programming the PLL_CLC.KDIV bit field. It also lists the resulting output clock f_{CPU} depending on K and dedicated VCO frequencies. Note that the K-Divider factor is always PLL_CLC.KDIV+1.

Table 3-4 K-Divider Selections

PLL_CLC. KDIV	K-Divider: K = KDIV+1	Resulting f_{CPU} Frequency (in MHz) for				f_{CPU} Duty Cycle [%]
		f_{VCO} = 400 MHz	f_{VCO} = 500 MHz	f_{VCO} = 600 MHz	f_{VCO} = 700 MHz	
0	1	400	500	600	700	1)
1	2	200	250	300	350	
2	3	133.33	166.67	200	233.33	
3	4	100	125	150	175	
4	5	80	100	120	140	40
5	6	66.67	83.33	100	116.67	50
6	7	57.14	71.43	85.71	100	42.86
7	8	50	62.5	75	87.5	50
8	9	44	55.56	66.67	77.78	44.44
9	10	40	50	60	70	50
10	11	36.36	45.45	54.55	63.64	45.45
11	12	33.33	41.67	50	58.33	50
12	13	30.77	38.46	46.15	53.85	46.15
13	14	28.57	35.71	42.86	50	50
14	15	26.67	33.33	40	46.67	46.67
15[2)]	16	25	31.25	37.5	43.75	50

1) These KDIV selections are not allowed in PLL Mode of the TC1766.

2) This is the default value after a power-on reset.

Note: The shaded selections cannot be used because the maximum TC1766 f_{CPU} frequency of 80 MHz is exceeded.

VCO Operating Range

The VCO can be selected for three operating ranges by programming the PLL_CLC.VCOSEL bit field. **Table 3-5** defines the min./max. f_{VCO} frequency as well as the VCO base frequency $f_{VCObase}$. This is the base VCO frequency when no PLL input clock f_P is connected.

Table 3-5 VCO Operating Range Selection

Bit PLL_CLC. VCOSEL	f_{VCOmin}	f_{VCOmax}	$f_{VCObase}$ [1]	f_{OSCmin} [2]	Unit
00_B	400	500	approx. 140 - 320	1.5	MHz
01_B	500	600	approx. 150 - 400	1.75	MHz
10_B [3]	600	700	approx. 200 - 480	2	MHz
11_B	Reserved, do not use this combination				

1) $f_{VCObase}$ is the free-running operation frequency of the PLL when no input clock is available.

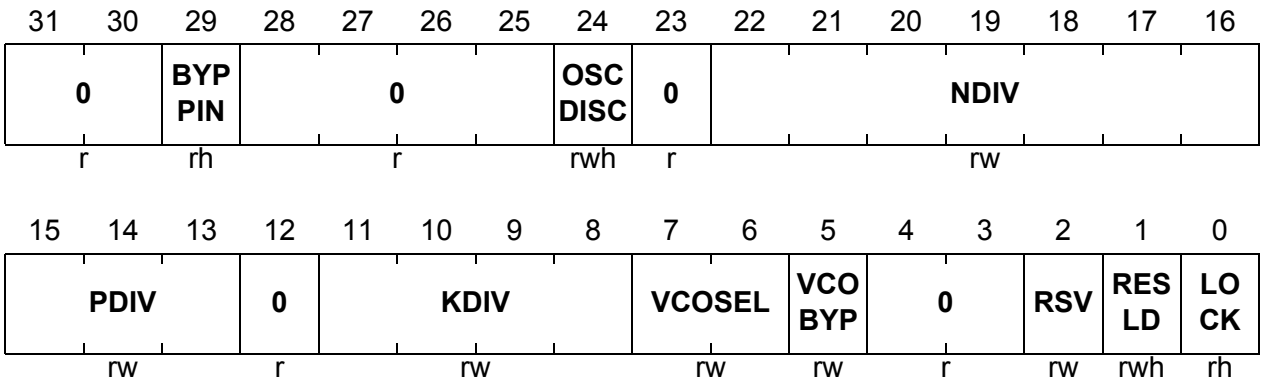
2) This is the minimum oscillator frequency to allow oscillator run detection to work properly.

3) This is the default value after a power-on reset.

3.2.2.3 PLL Clock Control and Status Register

The PLL Clock Control and Status Register PLL_CLC is located in the address range of the System Control Unit (see Page 5-70). It holds the hardware configuration bits of the PLL and provides the control for the N, P and K-Dividers as well as the PLL lock status bit. Register PLL_CLC is Endinit-protected.

PLL_CLC
PLL Clock Control Register **($F0000040_H$)** **Reset Value: see Table 3-6**



Field	Bits	Type	Description
LOCK	0	rh	**PLL Lock Status Flag** 0_B PLL is not locked (default after reset) 1_B PLL is locked
RESLD	1	rwh	**Restart Lock Detection** Writing a 1 to this bit will clear the LOCK flag and restart the PLL lock detection. After written with a 1, this bit is reset automatically to 0 and therefore always read back as 0. This bit becomes activated with a power-on reset operation. 0_B No operation 1_B PLL lock detection is restarted
RSV	2	rw	**Reserved** This bit field should be written with ‘1’ only
VCOBYP	5	rw	**VCO Bypass Mode Selection** 0_B Normal operation (default after reset) 1_B VCO Bypass Mode selected
VCOSEL	[7:6]	rw	**VCO Range Selection** This bit field selects operating range of the VCO. The coding is defined in Table 3-5.

Field	Bits	Type	Description
KDIV	[11:8]	rw	**PLL K-Divider Selection** This bit field selects the K-Divider value. The coding is defined in **Table 3-4**.
PDIV	[15:13]	rw	**PLL P-Divider Selection** This bit field selects the P-Divider value. The coding is defined in **Table 3-2**.
NDIV	[22:16]	rw	**PLL N-Divider** This bit field selects the N-Divider value. The coding is defined in **Table 3-3**.Note that only NDIV values between 19_D and 99_D (means N-Divider values of 20_D and 100_D) are allowed.
OSCDISC	24	rwh	**Oscillator Disconnect** This bit is used to disconnect the divided f_{OSC} clock from the PLL in order to avoid unstable operation due to noise or sporadic clock pulses coming from the oscillator circuit while the PLL is still trying to lock to invalid clock pulses. 0_B Oscillator clock f_{OSC} is connected to the PLL. 1_B Oscillator clock f_{OSC} is disconnected from the PLL (default after reset). This bit is set by hardware if a PLL loss-of-lock failure is detected.
BYPPIN	29	rh	**Bypass Pin Status Flag** This bit indicates the state of the BYPASS input pin as sampled with the last rising edge of PORST.
0	3, 4, 12, 23, [28:25], [31:30]	r	**Reserved** Read as 0; should be written with 0.

Table 3-6 Reset Values of Register PLL_CLC

Reset	BYPASS	Function	Reset Values
Power-on reset	0	The system is driven by the PLL clock based on the VCO base frequency.	0163 0F84$_H$
	1	The system is driven directly by the oscillator clock output.	2163 0F84$_H$
Other resets	X	Register content remains unmodified.	UUUU UUUU$_H$[1)]

1) U = unchanged to previously programmed values.

3.2.2.4 Changing PLL Parameters

There are some restrictions that must be regarded when PLL parameters in register PLL_CLC are modified.

- Only one parameter should be changed with a PLL_CLC register write operation
- VCOBYP can be changed without precautions
- PDIV and KDIV can be switched at any time in VCO Bypass Mode. However, the maximum operating frequency of the TC1766 must not be exceeded.
- Before changing VCOSEL, the VCO Bypass Mode must be selected.
- Before deselecting the VCO Bypass Mode, PLL lock detection must be restarted (RESLD bit set) and then the LOCK flag must be checked for the PLL lock condition. When LOCK is set, the VCO Bypass Mode can be deselected again.

Note: PDIV and NDIV can also be switched in PLL Mode. When changing NDIV, the VCO clock f_{VCO} may exceed the target frequency until the PLL becomes locked. After changing PDIV or NDIV, wait for the PLL lock condition. This procedure is typically used for increasing the VCO clock step-by-step.

3.2.2.5 Setting up the PLL after Reset

After reset, the system clock is running at the VCO base frequency $f_{VCObase}$ divided by factor K. The following actions must be executed:

1. Wait until the oscillator is running (OSC_CON.OSCR = 1)
2. Selection of the VCO Bypass Mode (PLL_CLC.VCOBYP = 1)
3. Selecting the VCO band by programming PLL_CON.VCOSEL
4. Program the desired P, N and K values (PDIV, NDIV, and KDIV bit fields of register PLL_CLC)
5. Connect the oscillator to the PLL, default after reset (PLL_CLC.OSCDISC = 0)
6. Wait until the PLL becomes locked (PLL_CLC.LOCK = 1)
7. Disable the VCO Bypass Mode (PLL_CLC.VCOBYP = 0)

After this procedure, the device is operating on the PLL target frequency. The note in **Section 3.2.2.4** is also valid for this procedure.

3.2.2.6 Lock Detection

The PLL has the capability to detect a failure of its input clock f_P clock and to bring the TC1766 into a safe state in such a case. This clock failure detection is done by the PLL lock detection unit. This unit indicates whether the PLL has reached its target frequency properly or not.

The lock detection unit operates as follows:

Two counters, A and B, count the clock pulses of the N-divider clock output f_N and the PLL reference clock f_P. When the counter values differ by more than 2 during counting, the counters are reset (meaning PLL is still unlocked). When the counter values reach

the end of a counting session (224 clock pulses) with a counting difference which is 2 or less than 2, the PLL is locked (PLL_CLC.LOCK = 1).

When the PLL is locked, the two counters proceed to count clock pulses. After every fourth clock pulse the counter values are checked, and when the counter values differ by more than 2, the unlocked state is entered (fast unlock check). If no unlock condition is detected (counter difference less than or equal 2), the counters are further incremented up to a maximum counter value of 232 clock pulses. After 232 clock pulses with no unlock condition of the counter values, the two counters are reset (slow unlock check).

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 by setting the NMISR.PLLNMI flag. Additionally, the PLL clock input f_P is disconnected from the PLL to avoid unstable operation due to noise or sporadic clock pulses coming from the oscillator circuit. Without a clock input f_P, the PLL gradually slows down to its VCO base frequency and remains there. The TC1766 remains in this state until the next power-on reset through pin $\overline{\text{PORST}}$, after that the PLL tries to restart and lock to the external clock again. No other reset cause can terminate this loss-of-clock state to avoid unstable operation due to the PLL trying to lock again. The TC1766 remains in the PLL unlocked state until the next power-on reset or a successful lock recovery occurs.

Note that the PLL unlock state can also be entered when the oscillator disconnect bit PLL_CLC.DISC becomes set by software.

3.2.2.7 Loss-of-Lock Recovery

If PLL has lost the lock condition, user software can try to re-lock the PLL again by executing the following sequence:

1. Restart the oscillator run detection by setting bit OSC_CON.ORDRES
2. Wait until OSCR is set
3. If bit OSCR is set, then
   a) Select the VCO Bypass Mode (PLL_CLC.VCOBYP = 1)
   b) Re-connect the oscillator to the PLL (PLL_CLC.OSCDISC = 0)
   c) Set the restart lock detection bit PLL_CLC.RESLD = 1
   d) Wait until the PLL becomes locked (PLL_CLC.LOCK = 1)
   e) When the PLL_CLC.LOCK is set again, the VCO Bypass Mode can be deselected (PLL_CLC.VCOBYP = 0) and normal PLL operation is resumed.

The note in **Section 3.2.2.4** is also valid for this procedure.

3.2.3 Power-on Start-up Operation

In order to support a wide range of input clock frequencies, the TC1766 requires a generic procedure to start-up the system clock.

When the TC1766 is powered up, a low level (0) must be applied to the power-on reset pin, $\overline{\text{PORST}}$. While $\overline{\text{PORST}}$ is active (at low level), the device is asynchronously held in

reset and the state of the BYPASS pin controls the operation of the clock circuitry. Therefore, BYPASS must be held at constant level during $\overline{\text{PORST}}$ active.

If the pin BYPASS is at a high level during power-on reset, Direct Drive Mode is selected (see **Page 3-10**). The low level at pin $\overline{\text{PORST}}$ has to be held long enough to make sure that a stable clock is provided to the TC1766. In case of an external crystal oscillator, it can take several ms until the oscillator has started up and is stable. If the clock input is provided by another clock source with faster start-up characteristics, the $\overline{\text{PORST}}$ low level can be released earlier.

If the pin BYPASS is at low level during power-on reset, PLL Mode is selected (see **Page 3-11**) and the procedure to start-up the PLL is as described in the following paragraph.

With $\overline{\text{PORST}}$ = 0, the oscillator is disconnected from the PLL. The PLL starts running at the VCO base frequency $f_{VCObase}$. After deactivation of the $\overline{\text{PORST}}$, the CPU and system clock are provided internally with a frequency that is equal to $f_{VCObase}$/K (K = 16 after reset). This means that in the TC1766, the Boot ROM code execution is started with the VCO base frequency. When the oscillator is running properly, bit OSCR becomes set and the user software can setup the PLL by programming its parameters as described in **Section 3.2.2.4**. If OSCR becomes not set, the user software has the possibility to run an emergency program using the base frequency of the PLL divided by K. In this case, K can be set to the minimum value, which results in the maximum possible CPU and system frequency.

*Note: See also **Page 4-6** for further details on the power-on reset operation.*

3.3 Module Power Management and Clock Gating

Because power dissipation is related to the frequency of gate transitions, the TC1766 performs power management principally by clock gating – that is, controlling whether the clock is supplied to its various functional units. Gating off the clock to unused functional modules also reduces electro-magnetic interference (EMI) since EMI is related to both the frequency and the number of gate transitions.

Clock gating is done either dynamically or statically. Dynamic clock gating in this context means that the TC1766 itself enables or disables clock signals within some functional modules to conserve power. Static gating means that software must enable or disable clock signals to functional modules. Clock gating is performed differently at different levels of system scope: Dynamic gating is generally performed at the lowest levels, either within a small region of logic, or at functional-unit boundaries for uncomplicated functions where hardware can dynamically determine whether that functionality is required, and can enable or disable it appropriately without software intervention. Static gating – which requires software intervention – is used to enable or disable clock delivery to individual high-level functional units, or to disable clock delivery globally at the clock's source. When the clock to individual functional units is gated off, they are said to be in Sleep Mode.

The TC1766 implements three levels of clock gating:

1. **Gated dynamically at the register:**
   The clock is shut off to a particular local resource in a functional module when this resource is not being used in that clock cycle. This operation is done primarily in the CPU and the PCP data paths, where unused resources are easily identified and controlled in each clock cycle.
2. **Gated dynamically at the functional unit (Idle Mode):**
   The clock is shut off at the functional unit boundary when the unit has nothing useful to do. This operation is done primarily in the CPU and the PCP. For the CPU, idle mode is controlled via software. The PCP disables its own clock when no program is running.
3. **Gated statically at each functional unit (Sleep Mode):**
   Software can send a global sleep request to individual functional units requesting that they enter Sleep Mode. Software must determine when conditions are such that entering Sleep Mode is appropriate. The individual units can be programmed to ignore or respond to this signal. If programmed to respond, units will first complete pending operations, then will shut off their own clocks according to their own criteria.

3.3.1 Module Clock Generation

As shown in **Figure 3-5** module clock generation of the TC1766 on-chip modules have two registers implemented:

- 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.

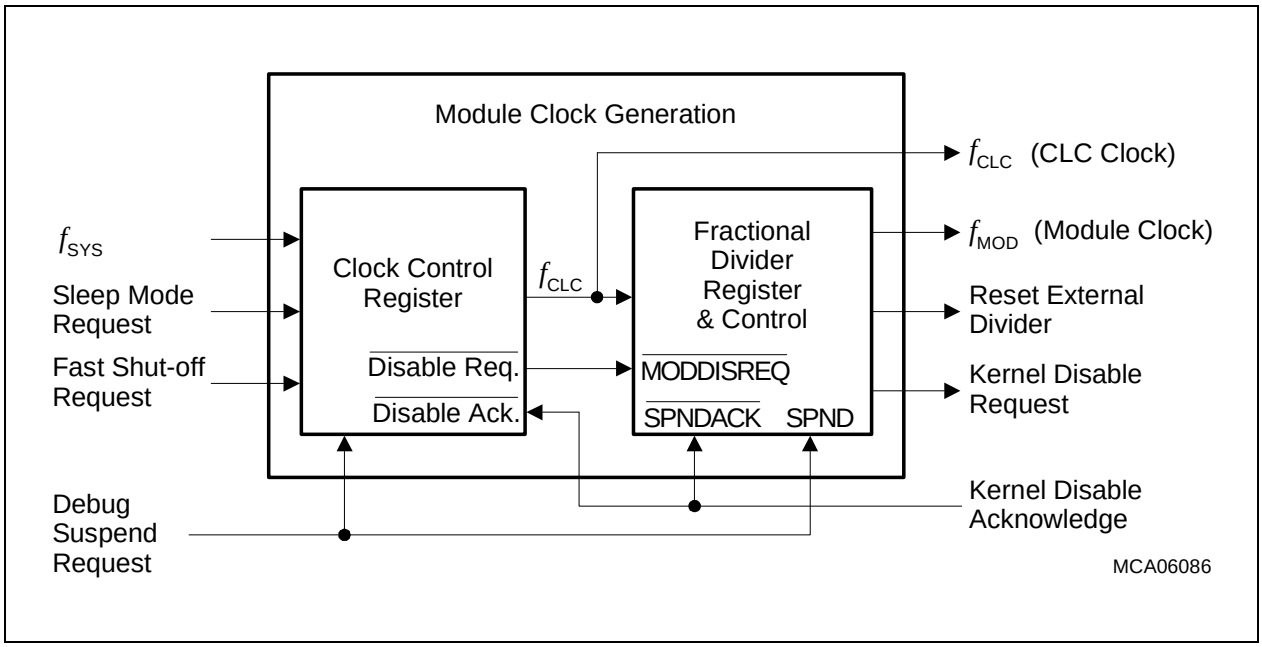


Figure 3-5 Module Clock Generation

Module clock and CLC clock are both derived from the system clock f_{SYS}. The CLC 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.

Furthermore, the module clock generation unit handles the sleep mode request signal, the fast shut-off request signal, and the debug suspend request signal.

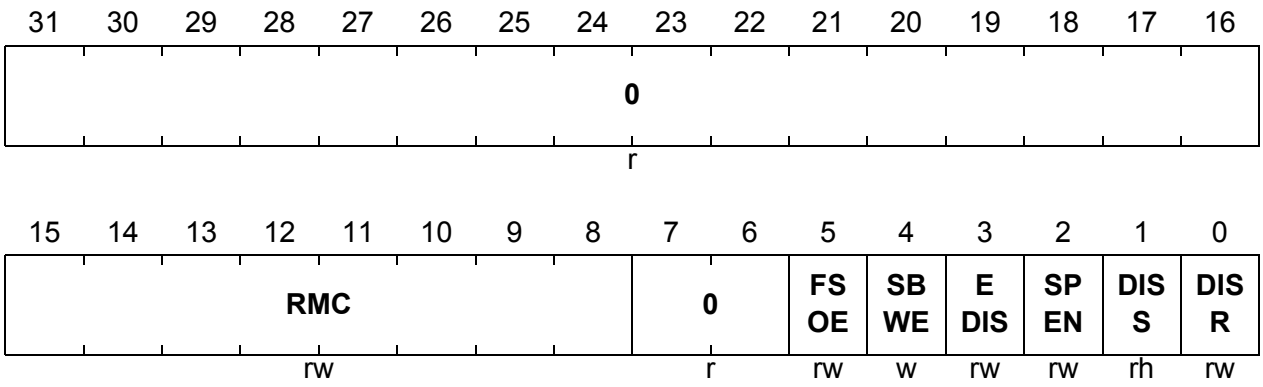
3.3.2 Clock Control Register CLC

All CLC registers have basically the same bit and bit field layout. However, not all CLC register functions are implemented for each peripheral unit. **Table 3-9** 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.

Associated Functions of the CLC Register

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

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 to 1 automatically.

Field	Bits	Type	Description
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 to 1 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.
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. This bit is a write-only bit. The value written to this bit is not stored. Reading this bit returns always 0.
FSOE	5	rw	**Fast Switch Off Enable** Used for fast clock switch-off in OCDS suspend mode. 0_B Clock switch-off in OCDS suspend mode via Disable Control Feature (Secure Clock Switch Off) selected. 1_B Fast clock switch off in OCDS suspend mode selected. This bit can be written only if SBWE is set to 1 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_{SYS}. 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.

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). With a few exceptions, the default state of a peripheral module after reset is "module disabled" with DISS set (see **Table 3-9**).

Write operations to the 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 (except CAN) 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 reset, before a module register (including the CLC register) will be written to.

Note: A read access occurring while a module is disabled is treated as a normal read access. This means, if a module register or a bit of it is cleared as a side-effect of a read access of an enabled module, it will not be cleared by this read access while the module is disabled.

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 (default after reset), 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.

Debug Suspend Mode Control

During emulation and debugging of TC1766 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 TC1766 begins (or resumes) executing a special debug monitor program. When the application is suspended, a suspend request signal is generated by the TC1766 and sent to all modules. If bit SPEN is set to 1, the operation of the peripheral module is stopped when the suspend signal is asserted. If 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 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 debug 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 to 1. In this way, user software cannot accidentally alter the value of the 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 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, which 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.

The kernel logic of the peripheral unit 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 peripheral module'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 a peripheral in progress when a disable request is detected, the access will be terminated correctly. Similarly, the peripheral's kernel provides an internal acknowledge signal when it has entered a stable state. The clock control unit for that peripheral module shuts off the module's clock 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 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 kernel is allowed to finish whatever operation is in progress. The clock to the unit is then shut off if both the bus interface and the module kernel have finished their current activity. If Fast Shut-off Mode is selected (FSOE = 1), clock generation to the unit is stopped as soon as any outstanding bus interface operation is finished. The clock control unit does not wait until the kernel 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 TC1766 modules except MultiCAN and DMA, the only shut down operating mode that is available is the Fast Shut-off ModeTC1766, 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 Debug 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 to 1. 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

Two peripheral modules of the TC1766, ASC0_CLC and STM_CLC, have an 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 CLC clock for a unit is generated as

$$f_{CLC} = f_{SYS} / RMC_{MOD} \quad (3.4)$$

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_{SYS}.

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 Buses for peripheral units with "destructive read" access (e.g. the ASC).

3.3.3 Fractional Divider Operation

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

3.3.3.1 Overview

The fractional divider makes it possible to generate a module clock from an input clock using a programmable divider. The fractional divider divides the input clock f_{IN} 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_{OUT}. The fractional divider is controlled by the FDR register. **Figure 3-6** 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

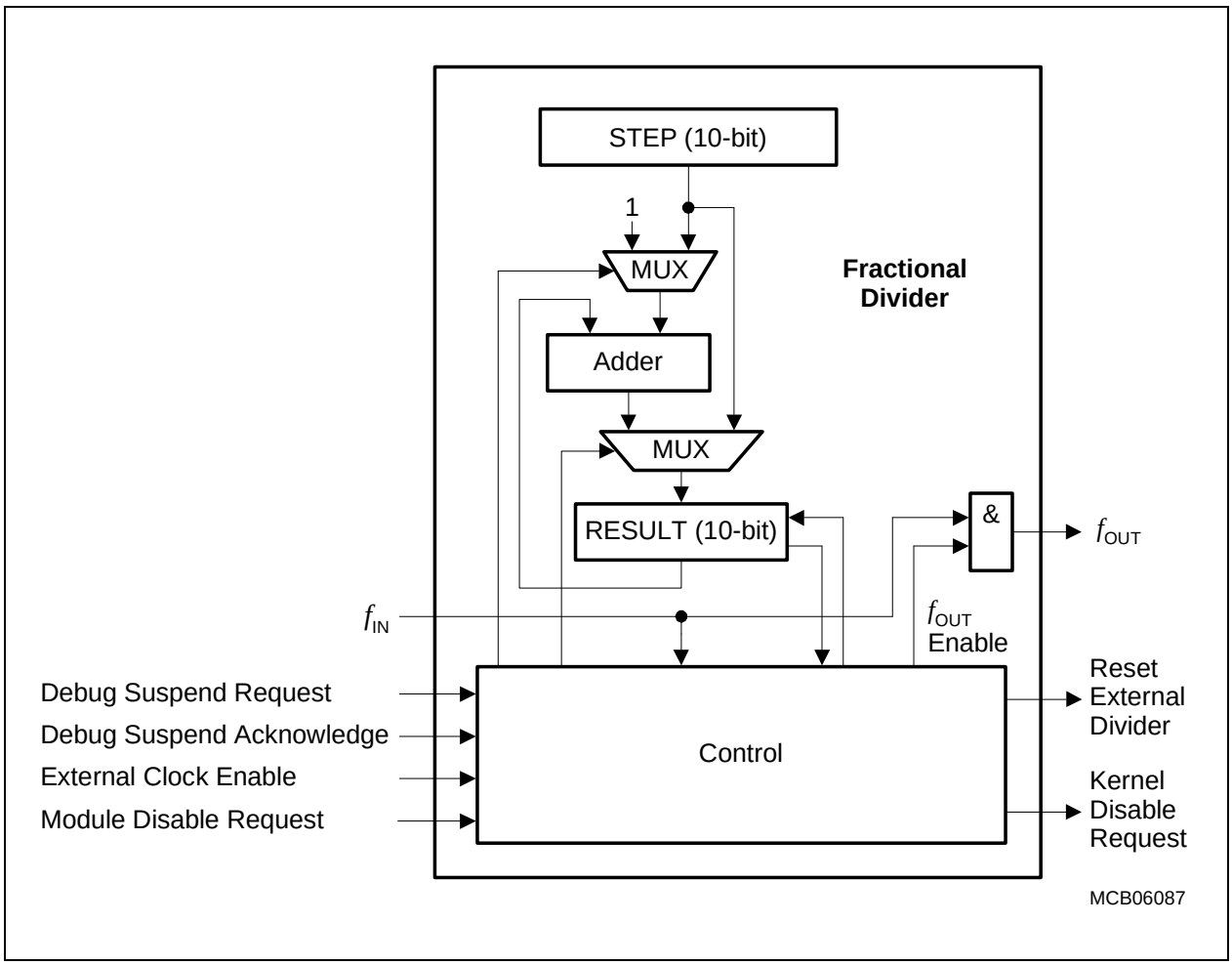


Figure 3-6 Fractional Divider Block Diagram

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 which adds a STEP value to the 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 3-7**.

Table 3-7 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 on-chip modules.
Debug Suspend Acknowledge		This input is driven with the disable acknowledge signal from the module kernel. This disable acknowledge signal is activated by the module kernel as a response to a suspend request that has been generated by the fractional divider via the Kernel Disable Request signal.
External Clock Enable		This input can be used to synchronize the fractional divider clock generation to external events.
Module Disable Request		This input is connected to the disable request output from the CLC logic (see **Figure 3-5**). An active signal at this input activates the Kernel Disable Request signal.
Kernel Disable Request	Output	This output signal becomes active when either the 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_{OUT} as input.

*Note: In the TC1766, the fractional divider input clock f_{IN} is also referred to as f_{CLC} and the fractional divider input clock f_{OUT} is also referred to as f_{MOD} (see **Figure 3-5**).*

3.3.3.2 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 at f_{OUT} 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:

$$f_{OUT} = f_{IN} \times \frac{1}{n} \quad \text{, with n = 1024 - STEP} \tag{3.5}$$

In order to get $f_{OUT} = f_{IN}$ STEP must be programmed with $3FF_H$. **Figure 3-7** shows the operation of the normal divider mode with a reload value of FDR.STEP = $3FD_H$. The clock signal f_{OUT} is the AND combination of the f_{OUT} Enable signal with f_{IN}.

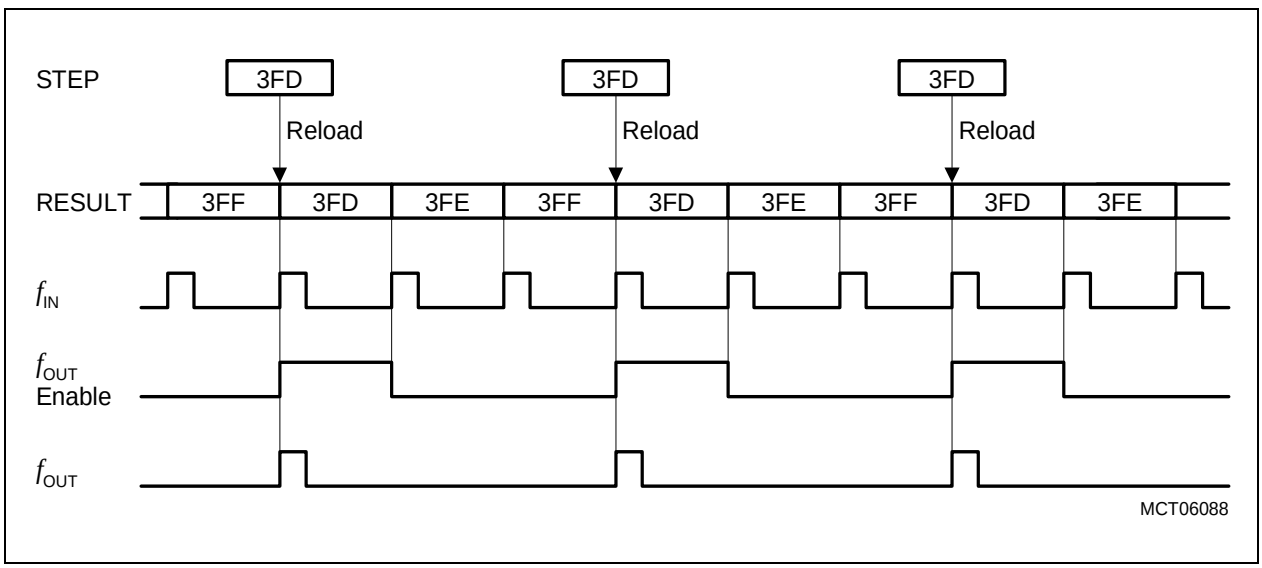


Figure 3-7 Normal Mode Timing

Fractional Divider Mode

When the fractional divider mode is selected (FDR.DM = 10_B), the output clock f_{OUT} is derived from the input clock f_{IN} 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 FDR.RESULT + 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_{OUT} 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:

$$f_{OUT} = f_{IN} \times \frac{n}{1024} \quad \text{, with n = 0-1023} \tag{3.6}$$

Figure 3-8 shows the operation of the fractional divider mode with a reload value of FDR.STEP = 234_H (= factor 564/1024 = 0.55). The clock signal f_{OUT} is the AND combination of the f_{OUT} Enable signal with f_{IN}.

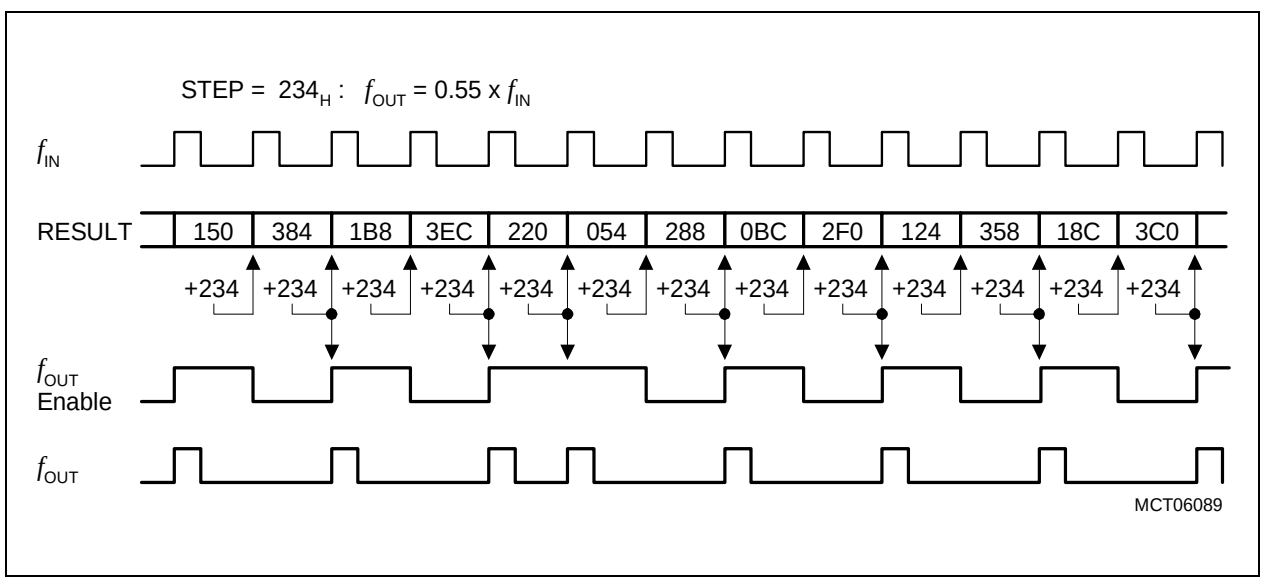


Figure 3-8 Fractional Divider Mode Timing

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, module registers are accessible for read and write actions, but other module internal functions are frozen. Suspend mode is entered one f_{IN} clock cycle after the Debug Suspend Request has been acknowledged by the Debug Suspend Acknowledge signal (granted suspend mode) **and** FDR.SC is not equal 00_B (clock output signal disabled). Suspend mode is immediately entered when bit SM is set to 1 **and** 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, SUSREQ and SUSACK of register FDR. Debug Suspend Request and (Debug Suspend Acknowledge or bit SM) both must remain set to maintain the suspend mode.

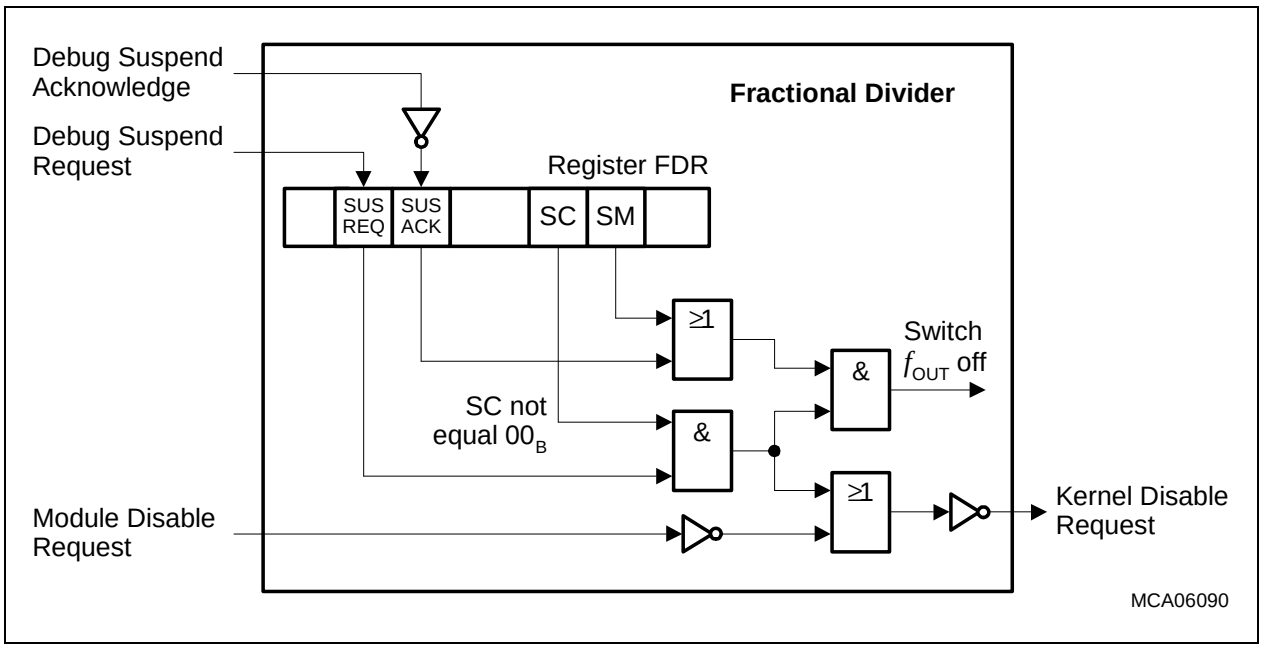


Figure 3-9 Suspend Mode Configuration

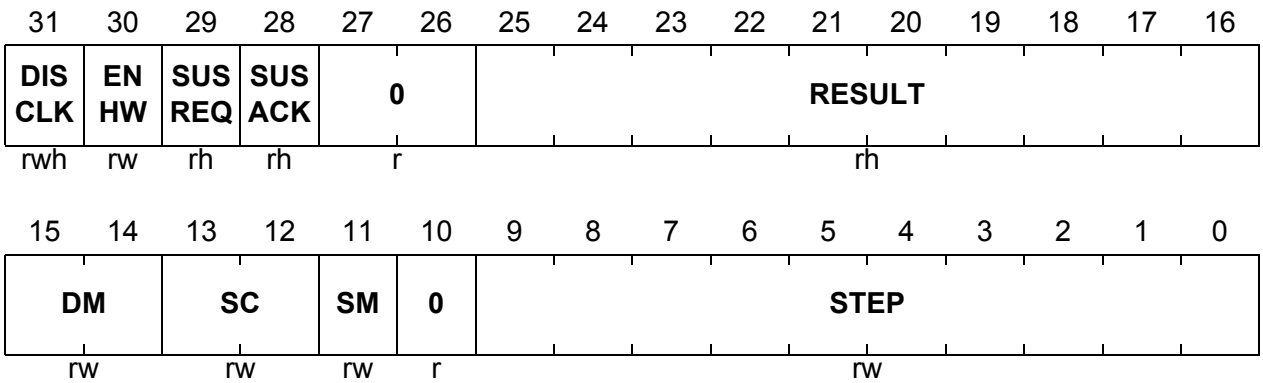
The Kernel Disable Request signal is always active when the Module Disable Request signal is activated, independently of the suspend mode settings in the fractional divider logic.

External Clock Enable

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

3.3.3.3 Fractional Divider Register FDR

FDR
Fractional Divider Register **($0C_H$)** **Reset Value: $0000\ 0000_H$**



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
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 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.

Field	Bits	Type	Description
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).
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 hardware by a high level of the External Clock Enable input signal. 1_B Bit DISCLK is cleared by hardware while the External Clock Enable input signal is at high level.

Field	Bits	Type	Description
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. Signal f_{OUT} becomes inactive. No change except when writing bit field DM. In case of a conflict between hardware-clearing and software-setting of DISCLK, the software-setting 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.

*Note: See also **Table 3-8** for further functional behavior of FDR bit fields and module operation.*

Note: The Fractional Divider Registers are Endinit-protected.

Table 3-8 Fractional Divider Function Table

Mode	SC	DM	Reset Ext. Divider Signal	Result	f_{OUT}	Operation of Fractional Divider
Normal Mode	–	00	1	unchanged	inactive	switched off
		01	0	continuously updated[1)]	active	normal divider mode
		10				fractional divider mode
		11		unchanged	inactive	switched off
Suspend Mode	00	00	1	unchanged	inactive	switched off
		01	0	continuously updated[1)]	active	normal divider mode
		10				fractional divider mode
		11		unchanged	inactive	switched off
	01	00	1	unchanged	inactive	switched off
		01	0	loaded with $3FF_H$		halted
		10				
		11		unchanged		switched off
	10	00	1	loaded with $3FF_H$	inactive	switched off
		01	0			halted
		10				
		11				switched off
	11	–	1	loaded with $3FF_H$	inactive	switched off

1) Each write operation to FDR with DM = 01_B or 10_B sets RESULT to $3FF_H$.

Implementation

FDR registers are implemented for several modules of the TC1766. The name of these FDR registers is always preceded by the module name (e.g. SSC0_FDR is the FDR register for the SSC0 module). **Table 3-9** defines which module is equipped with an FDR register.

In the implementation parts of the modules using a fractional divider (see **Table 3-9**), the signal f_{OUT} is described as a clock signal and not as clock enable signal.

3.3.4 Module Clock Register Implementations

Table 3-9 shows which of the CLC register bits/bit fields are implemented for each peripheral module in the TC1766 and which modules are equipped with a fractional divider.

Table 3-9 Clock Generation Implementation of the TC1766 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]
Name	State after Reset								
ADC0	off	✔	✔	✔	✔	✔	✔	–	✔
FADC	off	✔	✔	✔	✔	✔	✔	–	✔
ASC0	off	✔	✔	✔	✔	✔	✔	8-bit	–
ASC1	off	✔	✔	✔	✔	✔	✔	8-bit	–
SSC0	off	✔	✔	✔	✔	✔	✔	–	✔
SSC1	off	✔	✔	✔	✔	✔	✔	–	✔
MultiCAN	off	✔	✔	✔	✔	✔	✔	–	✔
DMA	on	✔	✔	✔	–	✔	✔	–	–
GPTA0	off	✔	✔	✔	✔	✔	✔	–	✔
MLI0	off	not implemented, MLI is connected to DMA_CLC							✔
MLI1	off	not implemented, MLI is connected to DMA_CLC							✔
MSC0	off	✔	✔	✔	✔	✔	✔	–	✔
PCP2	off	different bit definitions [2]							
PLL	on	different bit definitions						–	–
STM	on	✔	✔	✔	✔	✔	✔	3-bit	–

1) See Table 3-10 for more info on FDR implementation.

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

Note: The ports of the TC1766, SCU, and WDT do not provide CLC registers.

3.3.5 Fractional Divider Register Implementations

Table 3-10 shows the implementations specific differences of the fractional divider functionalities.

Table 3-10 FDR Register Implementations

FDR Register	Suspend Mode Acknowledge Operation[1)]	ENHW[2)]	Reset Ext. Divider[3)]
CAN_FDR	Acknowledge depends on module state	–	–
ADC0_FDR	Acknowledge depends on module states	–	resets analog parts of ADCs
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	–
SSC0_FDR	Always immediately acknowledged; independently from module state	from MultiCAN	–
SSC1_FDR	Always immediately acknowledged; independently from module state	from MultiCAN	–
SCU_SCLKFDR	Always immediately acknowledged; independently from module state	–	external divider-by-2

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 ("–").

3) This column shows whether the Reset External Divider output of the fractional divider is used (purpose see comment) or not ("–").

3.4 System Clock Output Control

The System Clock SYSCLK (alternate function of GPIO port line P4.3) is generated by a fractional divider block with a subsequent divide-by-2 stage (see **Figure 3-10**). This allows the SYSCLK frequency to be highly independent of the system frequency. Furthermore, the SYSCLK signal has duty cycle of 50% (with a small period jitter). After reset, SYSCLK is at low level when the Reset External Divider signal is at 1 (fractional divider stopped).

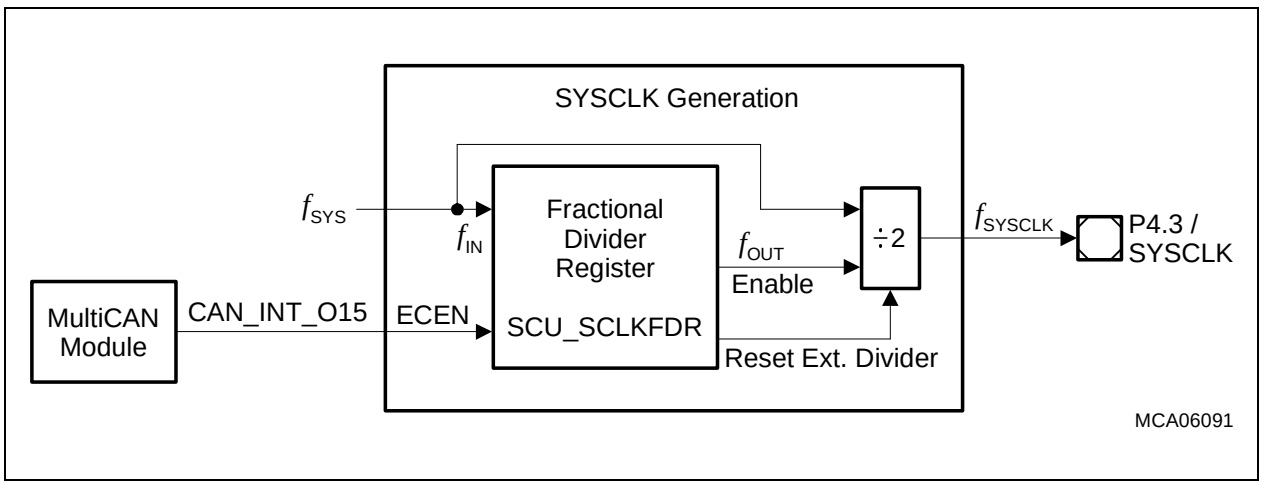


Figure 3-10 SYSCLK Generation

In normal divider mode, f_{SYSCLK} has a duty cycle of approximately 50%. The output frequency of f_{SYSCLK} in normal divider mode is defined according the following formula:

$$f_{SYSCLK} = f_{OUT}/2 = \frac{f_{SYS}}{2 \times n} \quad \text{, with n = 1024 - STEP} \tag{3.7}$$

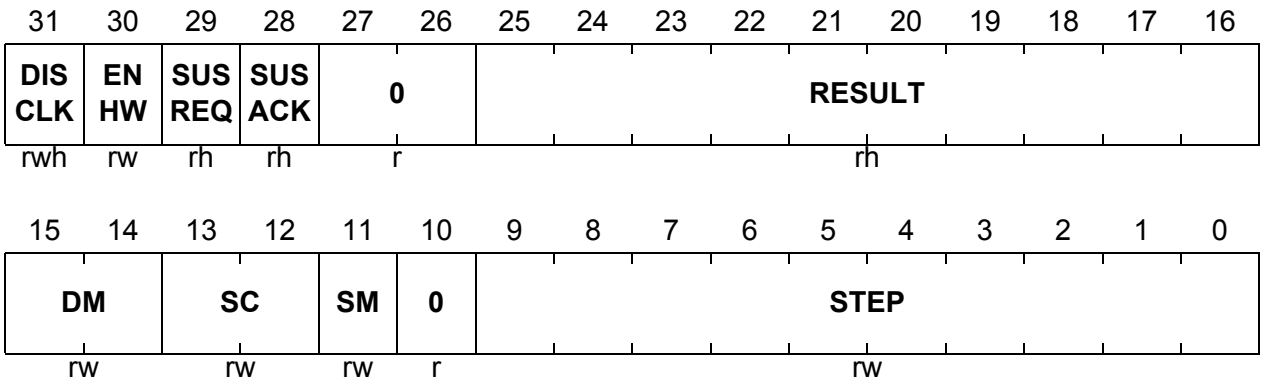
In fractional divider mode, f_{OUT} is derived from the input clock f_{IN} by division of a fraction of n/1024 for any value of n from 0 to 1023. In general, the fractional divider mode makes it possible to program the average output clock frequency with a higher accuracy than in normal divider mode. Note that in fractional divider mode, the clock f_{OUT} can have a maximum period jitter of one f_{IN} clock period (f_{SYSCLK} has a maximum period jitter of one half f_{IN} clock period).

The output frequency of f_{SYSCLK} in fractional divider mode is defined according to the following formula:

$$f_{SYSCLK} = f_{OUT}/2 = f_{SYS} \times \frac{n}{2 \times 1024} \quad \text{, with n = 0-1023} \tag{3.8}$$

3.4.1 System Clock Fractional Divider Register

SCU_SCLKFDR
SCU System Clock Fractional Divider Register
(F000000C$_H$) **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 granted or immediate suspend mode.
SC	[13:12]	rw	**Suspend Control** SC 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 $\overline{\text{SPNDACK}}$ signal.
SUSREQ	29	rh	**Suspend Mode Request** Indicates state of SPND signal.
ENHW	30	rw	**Enable Hardware Clock Control** No function for SYSCLK generation. Should be written with 0, will read back the last value written.
DISCLK	31	rwh	**Disable Clock** Makes it possible to disable the clock generation independently of the setting of the other functions.

Field	Bits	Type	Description
0	10, [27:26]	r	**Reserved** Read as 0; should be written with 0.

Note: This is only a short summary of the fractional divider behavior. The details on the fractional divider register functionality are described on ***Page 3-29****.*

4 Reset and Boot Operation

This chapter describes the conditions under which the TC1766 will be reset, the reset and boot operations, and the available boot options.

4.1 Reset and Boot Overview

When the TC1766 device is first powered up, several boot parameters such as the start location of the code have to be defined to enable proper start operation of the device. To accommodate this, the device has a separate Power-On Reset ($\overline{\text{PORST}}$) pin, and a number of configuration pins that are sampled during the power-on reset sequence or the hardware reset. At the end of this sequence, the sampled values are latched, and can not be modified until the next power-on reset. This guarantees stable conditions during the normal operation of the device.

To reset the device while it is operating, two options exist. For reset causes coming from the external world, a reset input pin, $\overline{\text{HDRST}}$ is provided. If software detects conditions that require resetting the device, it can perform a software reset by writing to a special register, the Reset Request (RST_REQ) register.

The Watchdog Timer (WDT) module is also capable of resetting the device if it detects a malfunction in the system. If the WDT is not serviced correctly and/or not serviced in time, it first generates a Non-Maskable Interrupt (NMI) request to the CPU (this allows the CPU to gather debug information), and then resets the device after a predefined time-out period.

After a reset has been executed, the Reset Status (RST_SR) register provides information on the type of the last reset and the selected boot configuration.

The external reset pin, $\overline{\text{HDRST}}$, has a double function. It serves as a reset input from the external world to reset the device, and it serves as a reset output to the external world to indicate that the device has executed a reset. For this purpose, pin $\overline{\text{HDRST}}$ is implemented as a bi-directional open-drain pin with an internal weak pull-up device.

Please note that any reset pulse generated on $\overline{\text{HDRST}}$ will be prolonged to reach a minimum pulse width. This ensures proper system reset, even it is caused by a very short pulse on $\overline{\text{HDRST}}$ coming from the system.

The boot configuration information required by the device to perform the desired start operation after a power-up reset includes the start location for the code execution, and the activation of special modes. This information is supplied to the chip via a number of dedicated input pins that are sampled and latched with $\overline{\text{HDRST}}$ or $\overline{\text{PORST}}$. However, the software reset provides the special option to alter these parameters to allow a different start configuration after the software reset has finished.

4.1.1 Reset Status and Control Registers

The Reset Status Register RST_SR indicates the cause of a reset and the selected boot configuration. The Reset Request Register RST_REQ is used to cause a software reset.

4.1.1.1 Reset Status Register

After a reset, the reset status register RST_SR indicates the type of reset that has occurred and which parts of the TC1766 were affected by the last reset operation. RST_SR also holds the state of the boot configuration pins HWCFG[3:0] (Port 4) that have been sampled during the HDRST inactive (low-to-high) transition. Register RST_SR is a read-only register.

RST_SR
Reset Status Register **($F0000014_H$)** **Reset Value: see Table 4-1**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	WDT RST	SFT RST	HD RST	PWO RST	0				TMP LS	HW BRK IN	0	HWCFG			
r	rh	rh	rh	rh	r				rh	rh	r	rh			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0													RS EXT	0	RS STM
r													rh	r	rh

Field	Bits	Type	Description
RSSTM	0	rh	**System Timer Reset Status** 0_B The system timer was not reset. 1_B The system timer was reset.
RSEXT	2	rh	**HDRST Input State during Last Reset** 0_B HDRST was not activated. 1_B HDRST was activated.
HWCFG	[19:16]	rh	**Boot Configuration Selection Status** This bit field indicates the status of the configuration pins HWCFG[3:0] at Port 4 that has been latched at the rising edge of HDRST. HWCFG[3:0] is assigned to P4.[3:0].

Field	Bits	Type	Description
HWBRKIN	21	rh	**Latched State of $\overline{\text{BRKIN}}$ Input** This bit indicates the logical state of the $\overline{\text{BRKIN}}$ input that has been latched at the end of a power-on reset.
TMPLS	22	rh	**Latched State of $\overline{\text{TESTMODE}}$ Input** This bit indicates the logical state of the $\overline{\text{TESTMODE}}$ input pin that has been latched at the end of a power-on reset.
PWORST	27	rh	**Power-On Reset Status Flag** 0_B The last reset was not a power-on reset. 1_B The last reset was a power-on reset.
HDRST	28	rh	**Hardware Reset Status Flag** 0_B The last reset was not a hardware reset. 1_B The last reset was a hardware reset.
SFTRST	29	rh	**Software Reset Status Flag** 0_B The last reset was not a software reset. 1_B The last reset was a software reset.
WDTRST	30	rh	**Watchdog Reset Status Flag** 0_B The last reset was not a watchdog reset. 1_B The last reset was a watchdog reset.
0	1, [15:3], 20, [26:23], 31	r	**Reserved** Read as 0.

Table 4-1 defines the reset values of the RST_SR register depending on the reset source.

Table 4-1 Reset Values of Register RST_SR

Reset Source	Reset Values
Power-on Reset	$0000\ 1000\ 0XX0\ XXXX\ 0000\ 0000\ 0000\ 0101_B$
Hardware Reset	$0001\ 0000\ 0XX0\ XXXX\ 0000\ 0000\ 0000\ 0000_B$
Software Reset	$0010\ 0000\ 0XX0\ XXXX\ 0000\ 0000\ 0000\ 0X0X_B$
Watchdog Timer Reset	$0100\ 0000\ 0XX0\ XXXX\ 0000\ 0000\ 0000\ 0101_B$

4.1.1.2 Reset Request Register

The reset request register RST_REQ is used to generate a software reset. Unlike the other reset causes, the software reset can exclude functions (System Timer reset and hardware reset ($\overline{\text{HDRST}}$) generation) from the reset. Additionally, the boot configuration information can be set by software.

RST_REQ is Endinit-protected, meaning that bit WDT_CON0.ENDINIT must be set to 0 first through the password-protected access scheme provided for the watchdog timer register WDT_CON0. Once access is gained through the Endinit protection scheme, RST_REQ can be written, causing the requested software reset.

RST_REQ
Reset Request Register (F0000010$_H$) Reset Value: 0000 0000$_H$

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0													RREXT	0	RRSTM
r													rw	r	rw

Field	Bits	Type	Description
RRSTM	0	rw	**Reset Request for the System Timer** 0_B Do not reset the System Timer 1_B Reset the System Timer
RREXT	2	rw	**Reset Request for External Devices** 0_B Do not activate hardware reset output $\overline{\text{HDRST}}$ 1_B Activate hardware reset output $\overline{\text{HDRST}}$
SWCFG	[19:16]	rw	**Software Boot Configuration** A software boot configuration different from the external applied hardware configuration can be specified with these bits. The encoding of this bit field is equal to the RST_SR.HWCFG line encoding.
SWBRKIN	21	rw	**Software Break Signal Boot Value** This bit determines the desired value for the $\overline{\text{BRKIN}}$ input signal to be used for software boot.

Field	Bits	Type	Description
SWBOOT	24	rw	**Software Boot Configuration Selection** 0_B The previously latched hardware configuration stored in RST_SR.HWCFG is used as boot selection. 1_B The software configuration as programmed in bit field SWCFG is used as boot selection.
0	1, 20, [15:3], [23:22], [31:25]	r	**Reserved** Read as 0; should be written with 0.

*Note: Please refer to the **Table 4-3** in this chapter for detailed value configuration for the SWCFG bit field as well as for SWBRKIN and SWBOOT bits.*

4.2 Reset Operations

This section describes the five reset sources of the TC1766.

4.2.1 Power-On Reset

The $\overline{\text{PORST}}$ pin performs a power-on reset, also called a cold reset. Driving the $\overline{\text{PORST}}$ pin low causes an asynchronous reset of the entire device. The device then enters its power-on reset sequence.

The PLL has its own power-on reset circuitry and is not affected by any other reset condition other than a low signal transition on the $\overline{\text{PORST}}$ pin. With an active power-on reset, the PLL is disconnected from the oscillator and will start running at its base frequency.

Simultaneously with $\overline{\text{PORST}}$ low, the reset circuitry drives the $\overline{\text{HDRST}}$ pin low and waits for the following two conditions to occur:

1. The system clock is active
2. Pin $\overline{\text{PORST}}$ is put to inactive level (driven high)

When both of these conditions are met and $\overline{\text{HDRST}}$ is not further pulled low externally, the power-on reset sequence is terminated synchronously with the next system clock transition.

The rising edge of $\overline{\text{PORST}}$ causes the state of some of the configuration pins for the PLL and the boot options to be latched into the appropriate registers. Others are latched with the rising edge of $\overline{\text{HDRST}}$. Fields in the Reset Status Register (RST_SR) are set to inform the user about this complete reset of the device. The power-on reset indication flag is set, while all other reset cause indication flags are cleared. Fields in this register that are set include the power-on reset indication flag (PWORST), as well as the reset status flags for the System Timer (RSSTM), and the reset output pin state (RSEXT).

The time $\overline{\text{PORST}}$ has to be active after the supply voltage is stable depends mainly on the settling time of the power supply lines. It does not directly depend on the oscillator start up time because a system clock signal is already provided by the PLL in PLL Base Mode as soon as the power supply lines are stable. This system clock signal is a slow clock signal based on the VCO base frequency. More details on the PLL Base Mode are provided on **Page 3-11**.

$\overline{\text{PORST}}$ is equipped with a noise-suppression filter that suppresses glitches below 10 ns pulse width. $\overline{\text{PORST}}$ pulses with a width above 100 ns are safely recognized as a valid signal. The noise-suppression filter is switched off when pin BYPASS = 1.

Attention: To ensure safe operation, the power-on reset operation must be completed prior to utilisation of the chip. With a completed power-on reset operation, a sequence of initialization of the chip is executed which includes the initialization of the ICACHE and TagRAM memories in the PMI. In the emulation device, the EEC is initialized upon the

completion of a power-on reset. Therefore, activation of the HDRST signal earlier than the maximum Power-on Reset Boot Time must be prevented. Please refer to the maximum Power-on Reset Boot Time defined in the Power, Pad and Reset Timing section of the TC1766 Data Sheet.

4.2.2 External Hardware Reset

The external hardware reset pin HDRST serves as an external reset input as well as a reset output. It is an active-low, bidirectional open-drain pin with an internal weak pull-up. An active-low signal at this pin causes the chip to enter its hardware reset sequence. The HDRST (output) pin is held low for 1024 f_{SYS} clock cycles by the reset circuitry until its internal reset sequence is terminated.

When the sequence is terminated, the reset circuitry then releases HDRST (that is, it does not actively drive HDRST anymore, so that the weak pull-up can try to drive the pin high). It then begins monitoring the level of the pin. If HDRST is still low (indicating that it is still being driven low externally), the reset circuitry holds the chip in hardware reset until a high level is detected. The hardware reset sequence is then terminated and flag RST_SR.HDRST is set.

The PLL is not affected by an external hardware reset and continues to operate.

In order to safely recognize a valid hardware reset, HDRST must be active for at least four f_{CPU} clock cycles.

HDRST is equipped with a noise-suppression filter that suppresses glitches below 10 ns pulse width. HDRST pulses with a width above 100 ns are safely recognized as a valid signal. The noise-suppression filter is switched off when pin BYPASS = 1.

4.2.3 Software Reset

A software reset is invoked by writing the appropriate bits in the reset request register RST_REQ. Unlike the other reset types, the software reset can optionally prevent the system timer reset and the external reset output HDRST generation from becoming active. To exclude one of these two system functions from software reset, the corresponding bits in RST_REQ (RRSTM or RREXT) must be set to 0. Additionally, a software reset can be executed with a programmable software boot configuration value (bit field RST_REQ.SWCFG) instead of the last hardware boot configuration value (as latched in bit field RST_SR.HWCFG).

To perform a software reset, the reset request register RST_REQ must be written. However, RST_REQ is Endinit-protected to avoid triggering of an unintentional software reset. Bit WDT_CON0.ENDINIT must be cleared via the password-protected access scheme. When this is done, a write access to RST_REQ can then be performed.

After the write access to RST_REQ, the entry to the software reset takes some time in which instructions can still be executed. Therefore, it is recommended not to execute

important instructions anymore after the instruction that writes to RST_REQ (for example, entering an endless loop).

4.2.4 Watchdog Timer Reset

A Watchdog Timer overflow or access error occurs only in response to severe and/or unknown malfunctions of the TC1766, caused either by software or hardware errors. Therefore, a Watchdog Timer reset occurs when an overflow of the Watchdog Timer takes place.

Before the Watchdog Timer generates its reset, it first indicates a Non-Maskable Interrupt (NMI) at the beginning of a prewarning phase and enters a time-out mode. The NMI invokes the trap service routine and can save critical states of the microcontroller for later examination to determine the cause of the Watchdog Timer failure. However, it is not possible to stop or terminate the Watchdog Timer's time-out mode or prevent the pending watchdog reset.

However, software can preempt the Watchdog Timer by issuing a software reset on its own. Since the cause of the system failure is presumably unknown at that time, and it is therefore uncertain which functions of the TC1766 are operating properly, it is recommended that the software reset be configured to reset all system functions including the System Timer and external reset output $\overline{\text{HDRST}}$, and to use the hardware boot configuration bit field as boot configuration source indicator.

If the NMI trap handler does not perform a software reset, or if the system is so compromised that the trap handler cannot be executed, the Watchdog Timer will cause a Watchdog Timer reset to occur at the end of its time-out mode period. The actions performed on a Watchdog Timer reset sequence are the same as those performed for an external hardware reset. At the end of the Watchdog Timer reset sequence, bits WDTRST, RSSTM, and RSEXT are set in register RST_SR. All other reset flags are cleared.

Watchdog Timer Reset Lock

When the system emerges from any reset condition, the Watchdog Timer becomes active, and, unless prevented by initialization software, will eventually time out. Ordinarily, initialization software will configure the Watchdog Timer and commence servicing it on a regular basis to indicate that it is functioning properly. Should the system be malfunctioning such that initialization and service are not performed in a timely fashion, the Watchdog Timer will time out, causing a Watchdog Timer reset.

If the TC1766 system is so corrupted that it is chronically unable to service the Watchdog Timer, the danger could arise that the system would be continuously reset every time the Watchdog Timer times out. This could lead to serious system instability, and to the loss of information about the original cause of the failure.

However, the reset circuitry of the TC1766 is designed to detect this condition. If a Watchdog Timer error occurs while one or both of the Watchdog Timer error flags (WDT_SR.WDTAE and WDT_SR.WDTOE) are already set to 1, the reset circuitry locks the TC1766 permanently in reset (Reset Lock, HDRST permanently active) until the next power-on reset occurs by activation of the PORST pin.

This situation could arise, for example, if the memory becomes corrupt, such that no valid code can be executed, including the initialization code. In this case, the initial time-out period of the Watchdog Timer cannot be properly terminated by software. The Watchdog Timer error flag WDTOE will be set when the Watchdog Timer overflows, and a Watchdog Timer reset will be triggered (after the watchdog reset pre-warning phase). The error flag WDTOE is not cleared by the Watchdog Timer reset that subsequently occurs. After finishing the Watchdog Timer reset sequence, the TC1766 will again attempt to execute the initialization code. If the code still cannot be executed because of connection problems, the WDTOE bit will not have been cleared by software. Again, the Watchdog Timer will time out and generate a Watchdog Timer reset. However, this time the reset circuitry detects that WDTOE is still set while a Watchdog Timer error has occurred, indicating danger of cyclic resets. The reset circuitry then puts the TC1766 in Reset Lock. This state can only be deactivated again by a power-on reset.

4.2.5 Debug System Reset

The debug system is not automatically reset by the regular resets except for the power-on reset. It is not affected by software resets or by a Watchdog Timer reset. A hardware reset becomes effective only if at the same time the OCDS reset is active as well.

Note: TriCore, PCP, DMA, and SBCU have integrated debug modules that are part of the debug system.

4.2.6 Module Reset Behavior

Table 4-2 lists the various functions of the TC1766 that are affected by a reset, depending on the reset type. A "■" means that this function/unit is reset to its default state.

Table 4-2 Effect of Reset on Device Functions

Module / Function		Watchdog Reset	Software Reset	Hardware Reset	Power-On Reset
Boot Configuration taken from		Bit field HWCFG	Bit fields HWCFG or SWCFG	Bit field HWCFG	Bit field HWCFG
CPU		■	■	■	■
SCU		■[1)]	■[1)]	■[1)]	■[1)]
BCUs, Bus System		■	■	■	■
Peripherals, PCP (except System Timer)		■	■	■	■
System Timer		■	Optional[2)]	Not affected	■
On-chip Static RAMs	**DMI, PMI**	Not affected, reliable	Not affected, reliable	Not affected, reliable	Affected, unreliable
	PMU	Not affected, unreliable	Not affected, unreliable	Not affected, unreliable	Affected, unreliable
	PCP Memory	Not affected[3)]	Not affected[2)]	Not affected[2)]	Affected, unreliable
On-chip Caches[4)]		■	■	■	■
Flash		■	■	■	■
Oscillator, PLL		Not affected	Not affected	Not affected	■
Port Pins		Tri-stated, weak pull-up active[5)]			
NMI Pin		Not affected	Disabled	Disabled	Disabled
Reset Out Pin $\overline{\text{HDRST}}$		■	Optional	■	■
OCDS L1 Debug System		Only if JTAG reset is also active[6)]			■

1) For two of the SCU registers (PLL_CLC, RST_SR) the reset value depends on the reset source.

2) “Affected” or “Not Affected” depends on bit RST_REQ.RRSTM.

3) If only the PCP accesses its memory, the contents are reliable. If an access from the FPI Bus is performed while the reset is activated, the contents are unreliable.

4) The actual data contents of the cache are not affected by a reset, however, the cache tag information is cleared, resulting in an “empty” cache.

5) While $\overline{\text{PORST}}$ is active it is guaranteed that the pins are in tri-state mode even if the core supply is not applied. Therefore the external power supply should activate $\overline{\text{PORST}}$ if any power failure is detected.

6) Default JTAG reset state (open JTAG pins) is active. A connected debugger tool controls the JTAG reset.

4.2.7 Booting Scheme

When the TC1766 is reset, it needs an indication on how it has to start after the reset sequence is finished. The TC1766 internal state is usually cleared by a reset, especially in the case of a power-up reset. Thus, boot configuration information needs to be applied by the external hardware through input pins. The boot configuration information is required for:

- The start location of the code execution, and
- Activation of special modes and conditions

For the start of code execution and activation of special mode, the TC1766 implements two basic booting schemes: A hardware booting scheme that is invoked through external pins; and a software booting scheme in which software can determine the boot options, overriding the externally-applied options.

Boot Options

Inputs HWCFG[3:0] (Port P4.[3:0]) and input $\overline{\text{BRKIN}}$, the $\overline{\text{TESTMODE}}$ pin, define the boot mode and boot location. **Table 4-3** shows the boot options/selections that are available in the TC1766. The source for $\overline{\text{BRKIN}}$ and HWCFG[3:0] can be either the corresponding bits HWBRKIN and HWCFG[3:0] in register RST_SR (sampled from configuration pins) or the software configuration bits SWBRKIN and SWCFG[3:0] in register RST_REQ.

The target boot address (program counter start address DFFF FFFC$_H$) is located at the end of the BootROM space. The BootROM code then determines how to proceed for the selected boot modes. When BootROM code is left, jump instructions to different dedicated addresses are executed. Note that pin $\overline{\text{TESTMODE}}$ must be at high level for all normal boot selection shown in **Table 4-3**, otherwise, undefined operational behavior of the chip may occur.

Table 4-3 TC1766 Boot Selections

BRKIN	HWCFG [3:0]	TESTMODE	Type of Boot	BootROM Exit Jump Address
Normal Boot Options				
1	0000_B	1	Enter bootstrap loader mode 1: Serial ASC0 boot via ASC0 pins	D400 0000_H
	0001_B		Enter bootstrap loader mode 2: Serial CAN boot via CAN pins	
	0010_B		Start from internal PFLASH	A000 0000_H
	0011_B		Alternate boot mode (ABM): Start from internal PFLASH after CRC check is correctly executed; enter a serial bootstrap loader mode[1] if CRC check fails	Defined in ABM header or D400 0000_H
	1000_B		Start from emulation memory if emulation device TC1766ED is available; in case of TC1766, execute stop loop	If TC1766ED: AFF2 0000_H
	1111_B		Enter bootstrap loader mode 3: Serial ASC0 boot via CAN pins	D400 0000_H
	others		Reserved; execute stop loop	–
Debug Boot Options				
0	0000_B	1	Tri-state chip	–
	others	irrel.	Reserved; execute stop loop	–

1) The type of bootstrap loader mode is selected by SCU_SCLIR.SWOPT[2:0] bit field.

4.2.7.1 Normal Boot Options

The normal boot options are invoked when $\overline{\text{BRKIN}}$ is inactive (at high level). The TC1766 has two options for booting into normal operation:

- Starting code execution from internal PFLASH
- Starting code execution from internal SPRAM after a bootstrap loader program has downloaded a program via a serial interface (ASC0 or CAN)

If a reserved boot selection is detected for HWCFG[3:0] with $\overline{\text{BRKIN}}$ = 1, the TC1766 stops further BootROM control operations and enters into an endless loop by executing a jump instruction to itself (can be aborted by a WDT reset).

Serial Boot via Bootstrap Loader

The bootstrap loader (BSL) is a software part of the BootROM which provides a mechanism to load a program code into the scratchpad RAM (SPRAM) of the PMI. The program code to be loaded is received serially either via the ASC0 or CAN interfaces. After loading of the code, the bootstrap loader program jumps directly to address D400 0000_H (start address of the PMI scratchpad RAM) and begins executing the program code that has been loaded. Further details about the BSL are described in Section 4.4.

Internal PFLASH Boot

After execution of the start-up code, two PFLASH boot options exist:

- PFLASH boot option 1 directly jumps to the internal flash.
- PFLASH boot option 2 (Alternate Boot Mode) first checks the flash via checksum calculation to determine whether one of two pre-defined entry locations contain valid code, and then jumps to a start address as defined by the ABM header.

4.2.7.2 Debug Boot Options

The debug boot options are invoked when $\overline{\text{BRKIN}}$ = 0 and $\overline{\text{TESTMODE}}$ = 1. The TC1766 has the option below for booting into debug operation:

- Tri-state chip

If a reserved boot selection is detected for HWCFG[3:0] with $\overline{\text{BRKIN}}$ = 0, the TC1766 stops further BootROM control operations and enters into an endless loop by executing a jump instruction to itself.

Tri-state Chip

If HWCFG[3:0] = 0000_B, all pins of the TC1766 are put into tri-state mode. In this mode, all pins are deactivated, including the oscillator, and internal circuitry is held in a low-power mode. This mode allows board-level test equipment or emulator probes, for example, to actively drive lines that are connected to TC1766 pins.

4.3 Boot ROM

The internal 16-Kbyte Boot ROM (BROM) has two parts:

- 8 Kbyte reserved for boot code (BootROM)
- 8 Kbyte reserved for factory test routines (TestROM)

Note: The expression "BootROM" in the text always refers to the 8 Kbyte boot code part of the BROM.

4.3.1 Addressing

As the original architectural position of the BROM is on the FPI Bus and the boot vector to the BootROM is hardwired in the CPU to this location, the internal BROM in the TC1766 is visible from the PMI side at two different locations, as can be seen in the memory map:

- In segment 8 (cached) starting at location 8FFF $C000_H$
- In segment 10 (non-cached) starting at location AFFF $C000_H$

The hardware-controlled start address after reset is DFFF $FFFC_H$. At this location (within the BROM), a jump to start address AFFF $F180_H$ is programmed to guarantee continuation of start program execution after reset.

Because the reset start address is fixed, the BootROM is mapped to the upper part of the internal BROM, at locations ..E000 - ..$FFFF_H$, and the TestROM is mapped to the lower part of internal BROM (..C000 - ..$DFFF_H$).

4.3.2 Program Structure

The different sections of the 8 Kbyte BootROM provide the following functionalities.

Startup Procedure

The startup procedure is the main control program in the BootROM which is always started after every reset operation. It initializes the chip, checks the hardware and software configuration selections, and branches to the corresponding boot routines. From a user's point of view, the startup procedure lengthens the reset time of the TC1766.

The startup procedure is responsible for the correct initialization of the on-chip resources. For example, the startup procedure monitors the Flash ramp-up phase and waits until the Flash memories are ready to be used. The startup procedure further controls the Flash read protection, initializes SRAM redundancy settings, and copies the unique chip identifier from Flash into the LDRAM memory.

During the startup procedure, the NMI, the watchdog timer, and the debug interfaces are disabled. The NMI has to be enabled by the user software (SCU_CON.NMIEN bit set) after proper setup of the interrupt vector table.

The reset configuration indicated in register RST_SR is evaluated by the startup procedure for selection of the configured operation and of program start memory. Additionally, some related bits of SWOPT configuration are evaluated in the SCU_SCLIR register.

Bootstrap Loader

The Bootstrap loader (BSL) is a software part of the BootROM that provides a mechanism to load a program code into the Scratchpad RAM (SPRAM) of the PMI. The program code to be loaded is received serially either via the ASC0 or CAN interfaces. After loading of the code, the bootstrap loader jumps directly to address D400 0000_H (start address of SPRAM) and begins executing the program code that has been loaded. Further details about the BSL are described in **Section 4.4**.

Alternate Boot Modes

The alternate boot modes will only branch to a user program in PFLASH memory if a CRC checksum test shows no error. If the CRC check fails, a bootstrap loader mode is entered instead of user program execution.

Emulation Support

If the device currently used is an emulation device (TC1766ED), a boot capability from emulation memory is supported. For the standard TC1766 this boot option cannot be activated.

Test and Stress Routines

The 8-Kbyte TestROM is reserved for special routines that are used for testing, stressing and qualification of the component. This boot option should never be activated by a user program.

BootROM Program Flow

Figure 4-1 shows the basic BootROM program flow. This flow diagram has one input (reset address DFFF $FFFC_H$) and several outputs that depend on the selected boot option (as described in **Table 4-3**).

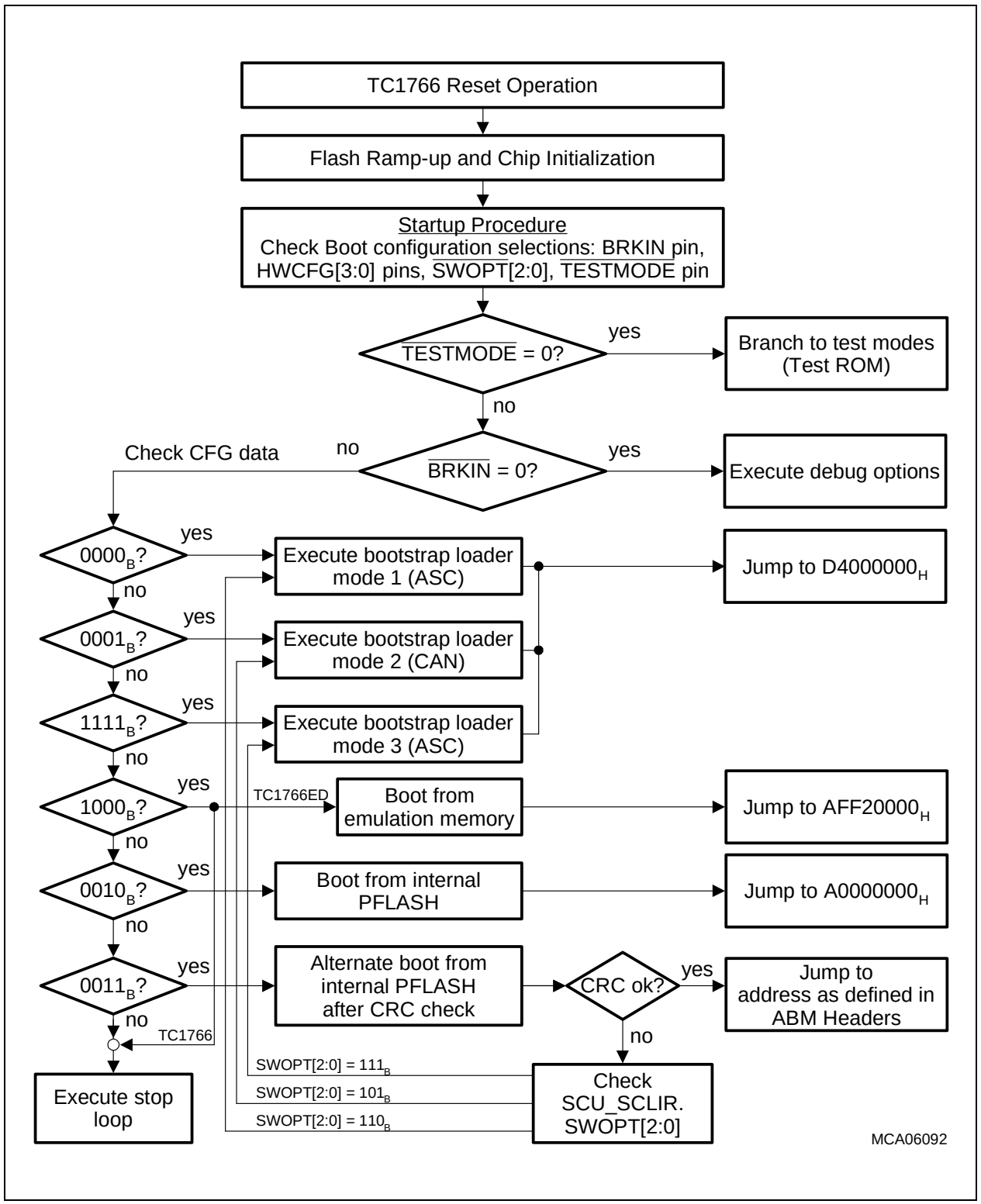


Figure 4-1 BootROM Flow Diagram

4.3.3 Initial State after BootROM Exit

The BootROM code is always executed after every reset operation. Depending on the program flow through the BootROM code and the BootROM exit path, several resources of the TC1766 on-chip hardware have been used and have been programmed. This means that the state of on-chip hardware resources that have been used by the BootROM code may differ from the device reset state as described by the register reset values. This section describes which initial state of the on-chip hardware resources is left after a specific BootROM exit.

Table 4-4 Hardware Status after BootROM Exit

BootROM Exit Path	State of On-Chip Hardware Resources
For all boot modes	CPU registers D[15:0] and A[15:0] are changed (note that the reset values are XXXX XXXX$_H$); Memory locations that are used in LDRAM: D000 0000$_H$ - D000 000F$_H$: updated with unique Chip ID D000 0010$_H$ - D000 0107$_H$: updated with BootROM data SCU_STAT[15:13] are set to 001$_B$. FLASH_MARP.TRAPDIS and FLASH_MARD.TRAPDIS are cleared
Branch to test modes	Only applicable for test purposes
Execute debug boot options	The following registers have been changed: CPR0_0L, CPM0, and TR0EVT
Exit of Bootstrap loader mode 1 (ASC0)	P3_IOCR0 has been changed (P3.1/TXD0A); ASC0 module is initialized and enabled; PLL_CLC has been changed
Exit of Bootstrap loader mode 2 (CAN)	P3_IOCR12 has been changed (P3.13/TXDCAN0); MultiCAN module is initialized and enabled; registers of CAN node 0, message object 0, and message object 1 have been used; PLL_CLC has been changed
Exit of Bootstrap loader mode 3 (ASC0)	P3_IOCR12 has been changed (P3.13/TXD0B); ASC0 module is initialized and enabled; PLL_CLC has been changed
Boot from emulation memory	EEA flag (bit 8) in the SCU_STAT is set
Boot from PFLASH	No further changes
Alternate boot from PFLASH	Memory checker module has been used
Execute Stop Loop	BTV has been changed

4.4 Bootstrap Loader (BSL)

The bootstrap loader (BSL) is a software part that is integrated in the TC1766 BootROM. The BSL provides a mechanism to load a program code via a serial interface (ASC or CAN) into the Scratchpad RAM (SPRAM) of the PMI. After loading of the code, the BSL jumps directly to address D400 0000_H (start address of the PMI SPRAM) and begins executing the program code that has been loaded. The BSL automatically calculates the baud rate of the serial data streams.

Table 4-5 shows the three BSL modes with their parameters.

Table 4-5 Bootstrap Loader Selections

Bootstrap Loader Mode	Selection[1)]		Associated I/O Lines	
	$\overline{\text{BRKIN}}$	HWCFG [3:0]	Receive	Transmit
Bootstrap Loader Mode 1 (BSL1): **ASC Boot via ASC0 Pins**	1	0000_B	P3.0 / RXD0A	P3.1 / TXD0A
Bootstrap Loader Mode 2 (BSL2): **CAN Boot**		0001_B	P3.12 / RXDCAN0	P3.13 / TXDCAN0
Bootstrap Loader Mode 3 (BSL3): **ASC Boot via CAN Pins**		1111_B	P3.12 / RXD0B	P3.13 / TXD0B

1) The bootstrap loader mode selections in alternate boot modes, see **Table 4-3**.

With the low-to-high signal transition of the hardware reset signal $\overline{\text{HDRST}}$ or the power-on reset signal $\overline{\text{PORST}}$, the input pins $\overline{\text{BRKIN}}$ and HWCFG[3:0] of the TC1766 are latched. If one of the latched $\overline{\text{BRKIN}}$/HWCFG[3:0] signal combinations in **Table 4-5** is detected, the BSL is started and the selected bootstrap loader mode is entered.

The BSL can also be started by a software reset. For this purpose, bit BRKIN and bit field SWCFG of the reset request register RST_SR must be loaded with the corresponding $\overline{\text{BRKIN}}$/HWCFG[3:0] code, and bit RST_REQ.SWBOOT in register RST_REQ must be set (see also RST_REQ register description on **Page 4-4**).

When a boot option for a BSL mode is detected, the TC1766 jumps to address DFFF $FFFC_H$ which is the last word address of the BootROM. During execution of the BSL, the WDT interrupts and NMI are disabled.

4.4.1 Bootstrap Loader Mode 1 - ASC Boot via ASC0 Pins

The ASC boot mode of the BSL moves 128 bytes of program code/data into the SPRAM and starts executing the loaded code from address D400 0000_H (address of the first transmitted byte). The initially-loaded program code will most probably load additional code or data. For that purpose, it can directly use the ASC0 interface that has already

been initialized during the BSL execution. **Figure 4-2** shows the actions that take place in BSL mode 1.

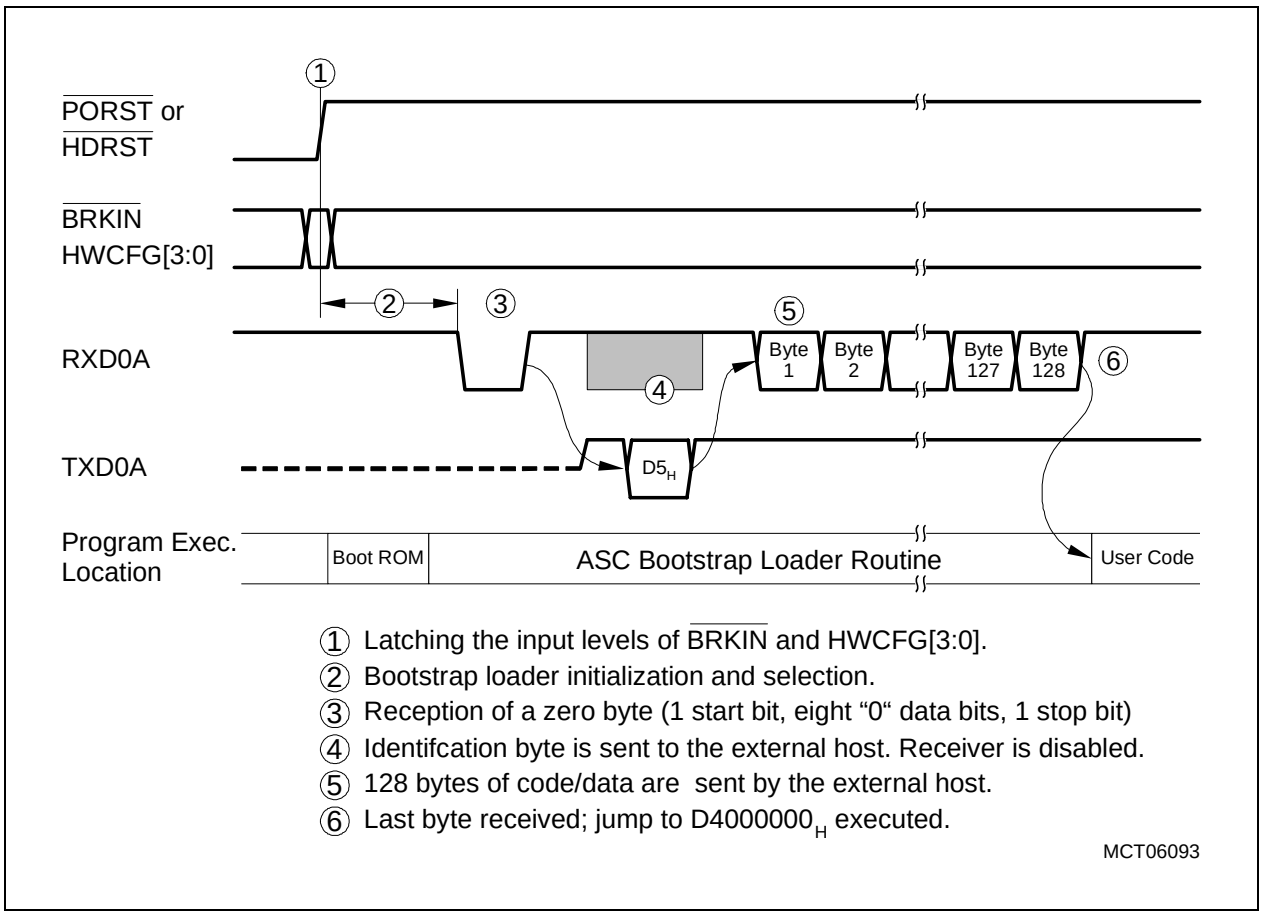


Figure 4-2 ASC0 Bootstrap Loader Sequence (Mode 1)

As the first task, the ASC0 determines the baud rate at which the external host is communicating. This task requires the external host to transmit an initialization byte to the TC1766 at the RXD0A input line of the ASC0. The initialization byte is built up by one low-level start bit, eight low-level data bits, and one stop bit (a low pulse with a width of nine serial bit cells). The BSL software measures the width of the initialization byte, calculates the baud rate, and writes the corresponding ASC0 registers with the values that select the detected baud rate.

After the baud rate calculation and initialization (receive pin remains disabled during this time), an identification byte ($D5_H$) is sent to the external host indicating that the TC1766 is ready to accept a data transfer from the host of exactly 128 bytes (32 words). After the identification byte has been transmitted, the ASC0 receive pin is enabled again. The BSL software now enters two receive loops. The inner loop waits until it has received four bytes. The outer loop writes one word (four bytes) to the SPRAM program memory of the PMI. These loops are running until 128 bytes have been received.

After the reception of the 128 bytes, the BSL software is finished and a jump to address D400 0000$_H$ is executed. This address is the 32-bit word location in the SPRAM at which the first four bytes of the transmitted 128 bytes have been written.

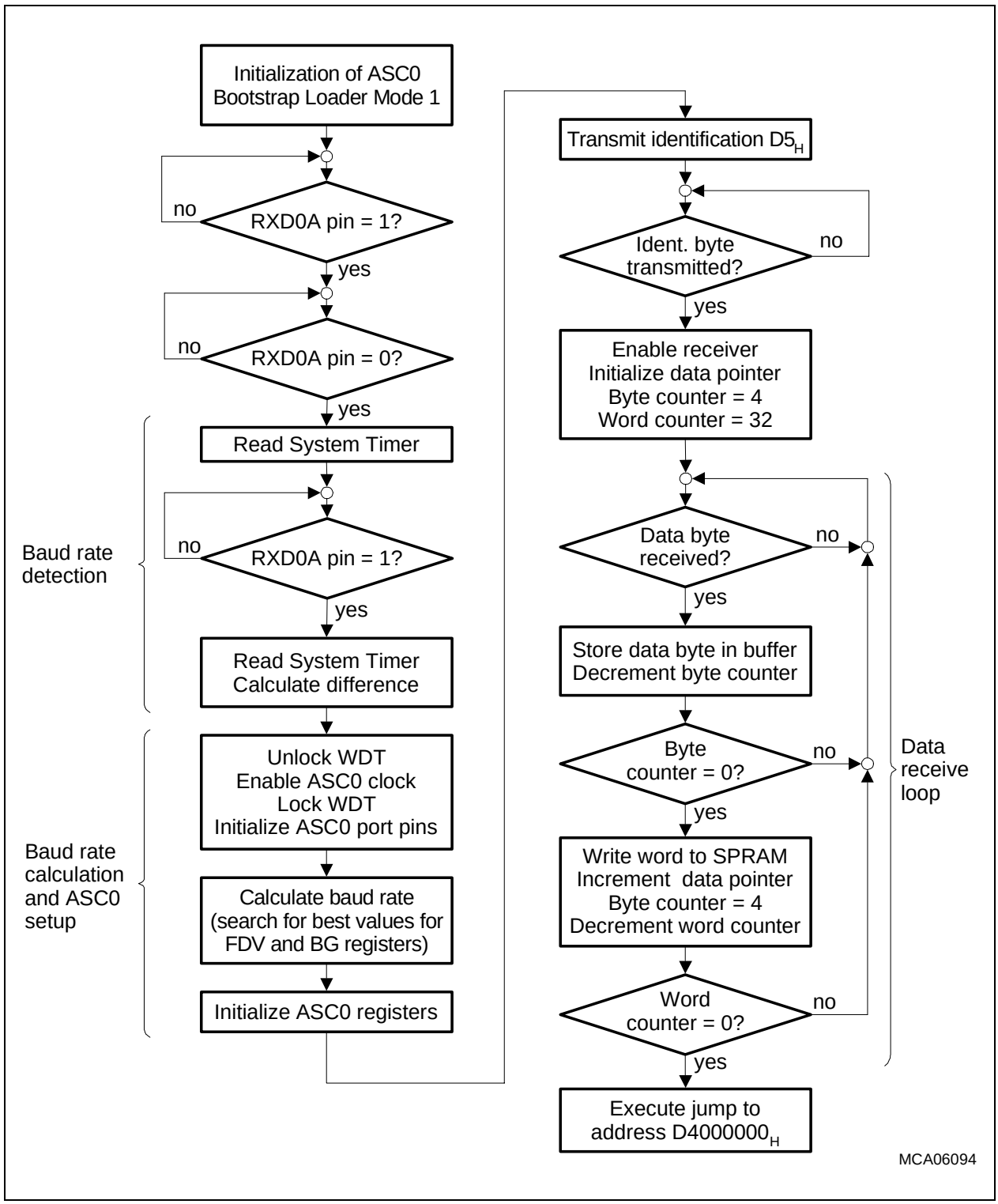


Figure 4-3 Flow Diagram of Bootstrap Loader ASC0 Boot Modes

4.4.2 Bootstrap Loader Mode 2 - CAN Boot via CAN Pins

The CAN boot mode of the BSL provides a mechanism to load program code/data via the MultiCAN module into the SPRAM, and start executing the loaded code from address D400 0000_H (address of the first transmitted byte). A single data frame always transmits eight code/data bytes from the external host to the TC1766. The number of data frames to be received in CAN boot mode is programmable and depends on the required amount of code/data to be transmitted. The 16-bit data message count value DMSGC defines the number of code/data bytes to be transmitted as 8 × MSGC with MSGC = 1 to 2^{16}. Therefore, the 16 Kbyte SPRAM of the TC1766 can be loaded completely during CAN boot mode.

The communication between TC1766 and external host is based on the following three CAN standard frames:

- Initialization frame - sent by the external host to the TC1766
- Acknowledge frame - sent by the TC1766 to the external host
- Data frame(s) - sent by the external host to the TC1766

The initialization frame is used in the TC1766 for baud-rate detection. After a successful baud-rate detection is reported to the external host by the acknowledge frame, data is transmitted by data frames. **Table 4-6** shows the parameters and settings of the three CAN standard frames used for CAN BSL.

Table 4-6 CAN Bootstrap Loader Frames

Frame Type	Parameter	Description
Initialization Frame	Identifier	11-bit, don’t care
	DLC = 8	Data length code, 8 bytes transmitted within CAN frame
	Data byte 0	55_H
	Data byte 1	55_H
	Data byte 2	Acknowledge message identifier ACKID, low byte
	Data byte 3	Acknowledge message identifier ACKID, high byte
	Data byte 4	Data message count DMSGC, low byte
	Data byte 5	Data message count DMSGC, high byte
	Data byte 6	Data message identifier DMSGID, low byte
	Data byte 7	Data message identifier DMSGID, high byte

Table 4-6 CAN Bootstrap Loader Frames (cont'd)

Frame Type	Parameter	Description
Acknowledge Frame	Identifier	Acknowledge message identifier ACKID as received by data bytes [3:2] of the initialization frame
	DLC = 4	Data length code, 4 bytes transmitted within CAN frame
	Data bytes 0/1	Contents of bit-timing register
	Data bytes 2/3	Copy of acknowledge identifier from initialization frame
Data frame	Identifier	Data message identifier DMSGID as sent by data bytes [7:6] of the initialization frame
	DLC = 8	Data length code, 8 bytes transmitted within CAN frame
	Data bytes 0 to 7	Data bytes, assigned to increasing destination (SPRAM) addresses

Initialization Phase

As in the ASC boot mode, as its first task the CAN BSL has 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 TC1766. The first two data bytes of the initialization frame include a 2-byte 5555_H baud-rate detection pattern, an 11-bit (2-byte) identifier ACKID[1)] for the acknowledge frame, a 16-bit data message count value DMSGC, and an 11-bit (2-byte) identifier DMSGID[1)] to be used by the data frame.

When CAN boot mode is entered, the CAN BSL program starts measuring signal pulses at the RXDCAN0 input. After reception and pulse measurements of a 5555_H bit pattern (data bytes 0 and 1 of the initialization frame), the CAN BSL program calculates the CAN baud rate and programs the baud rate registers of the MultiCAN module. The TC1766 is now ready to receive CAN frames at the baud rate of the external host.

Acknowledge Phase

In the acknowledge phase, the BSL 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. Following which, the BSL transmits an acknowledge frame back to the external host indicating that it is now ready to receive data frames. The

1)The CAN bootstrap loader copies the two identifier bytes received in the initialization frame directly to register MOAR. Therefore, the respective fields in the initialization frame must contain the intended identifier padded with two dummy bits at the lower end and extended with bitfields IDE ($=0_B$) and PRI ($=01_B$) at the upper end.

acknowledge frame uses the message identifier ACKID that has been received with the initialization frame. The eight data bytes of the acknowledge frame are a copy of the data bytes of the recognized initialization frame.

Data Transmission Phase

In the data transmission phase, data frames are sent by the external host and received by the TC1766. The data frame uses the 11-bit (2-byte) 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 SPRAM at 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 BSL software is finished and executes a jump to address D400 0000_H. This address is the first 32-bit word location in the SPRAM.

Timing Parameters

There are no general restrictions for CAN timings of the external host. After a reset of external host and TC1766, the external host can start transmitting initialization frames. If no acknowledge frame is sent back within a certain time as defined by the external host (e.g. after a dedicated number of initialization frame transmissions), the external host can determine that the TC1766 is not able to establish the CAN boot communication link.

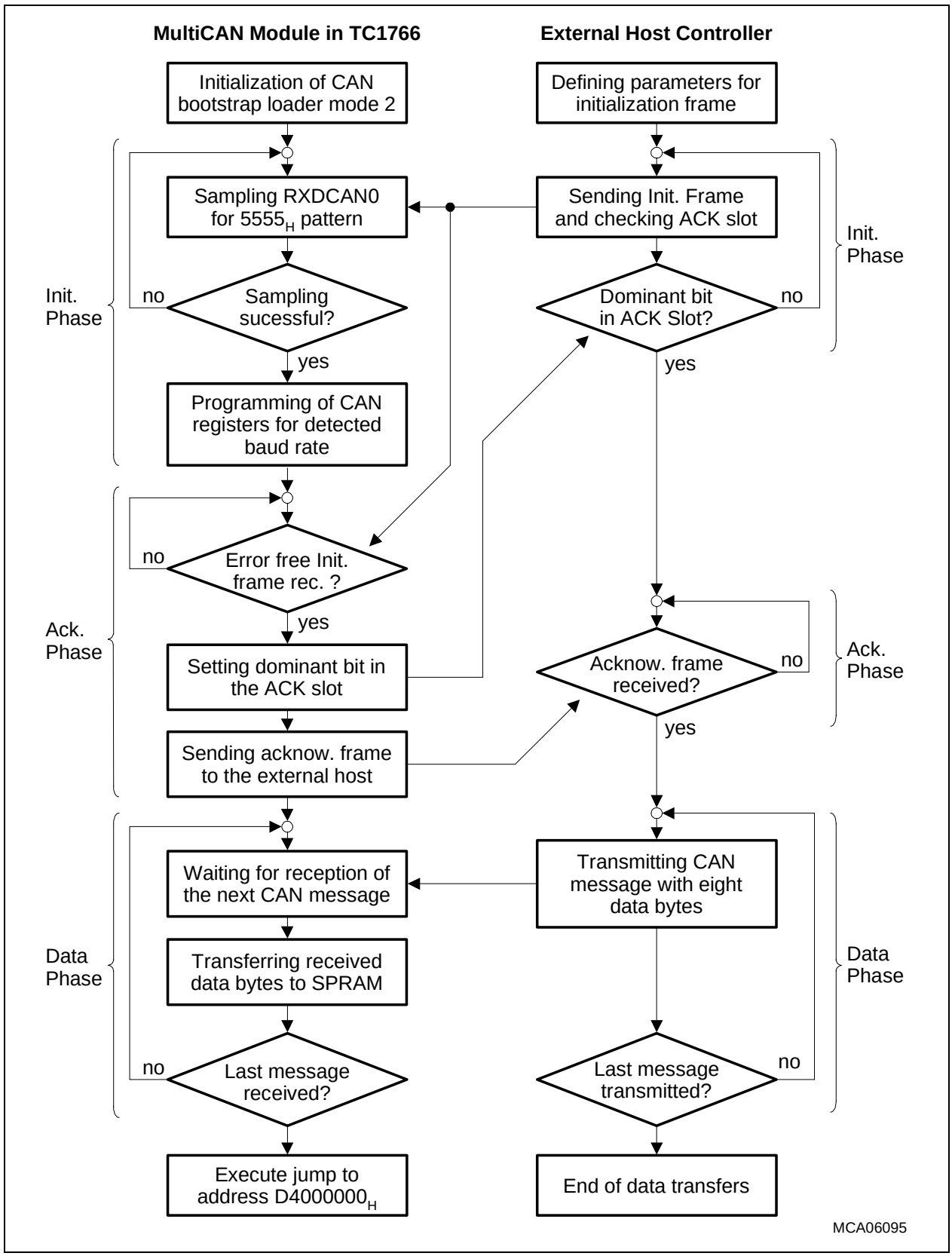


Figure 4-4 Flow Diagram of CAN Bootstrap Loader Mode

4.4.3 Bootstrap Loader Mode 3 - ASC Boot via CAN Pins

Except for different connections to serial port lines of ASC0, the bootstrap loader mode 3 is identical to bootstrap loader mode 1. The serial data input of the ASC0 is connected to RXD0B which is an alternate function of P3.12/RXDCAN0 and the serial data output of the ASC0 is connected to TXD0B which is an alternate function of P3.13/TXDCAN0.

4.4.4 Alternate Boot Modes

The alternate boot modes (ABM) make it possible to execute a CRC procedure before jumping to the program start addresses in internal PFLASH. In case of a CRC error in the checked code, the BSL is entered and loading of serial (ASC or CAN) code/data can be initiated.

To use ABM, the user has to define the parameters for the CRCs in the two ABM headers. These parameters are:

- The program start address in case of a positive CRC
- The start address of the memory range which has to be verified with CRC checksum
- The end address of the memory range which has to be verified with CRC checksum
- The checksums for the checked memory range and for the ABM header itself

Basically the CRC procedure of the BootROM program consists of three steps:

Step 1: The first CRC is executed with the parameters of the primary ABM header. If the first CRC passes, the user program is started at the address as defined in the first word of the primary ABM header.

Step 2: If the first CRC of the primary ABM header fails, a second CRC with the parameters of the secondary ABM header is performed. If this second CRC passes, the user program is started at an alternate start address as defined in the first word of the secondary ABM header.

Step 3: If the second CRC also fails, one of the three BSL modes is entered. The BSL mode to be used is defined by the content of bit field SCU_SCLIR.SWOPT[2:0]. This bit field contains the state of the Port 0 pins P0.[2:0] that was latched at the last rising edge of $\overline{\text{HDRST}}$ (see also **Page 9-27**).

Table 4-7 ABM Bootstrap Loader Selections

SWOPT[2:0]	Selected ABM Bootstrap Loader Mode
110_B	Bootstrap Loader Mode 1: ASC Boot via ASC0 Pins
101_B	Bootstrap Loader Mode 2: CAN Boot
111_B 1)	Bootstrap Loader Mode 3: ASC Boot via CAN Pins
others	Reserved; do not use these combinations

1) This value is default after reset without connecting P0.[2:0] to a low or high level. Port 0 pins are inputs after reset with a pull-up device connected.

Note: For CRC generation and error checking, the BootROM software uses the TC1766 on-chip memory checker module. For CRC generation and error checking, the BootROM software uses the TC1766 on-chip memory checker module with an initial value of FFFF FFFF$_H$ for the memory checker result register before the checksum is generated. Further details about this module are described in ***"Memory Checker Module" on Page 11-100****.*

Definition of the ABM Headers

There are two ABM headers defined, a primary and a secondary (alternate). The two ABM headers have a size of 32 bytes (8 words) and are located at fixed addresses in the internal Flash as shown in **Table 4-8**. Alternatively, cached addressing (segment 8) or non-cached addressing (segment 10) can be used.

Table 4-8 ABM Header Locations

Internal Flash Memory (PFLASH)	
Base Address	**End Address**
Primary ABM Header	
8001 FFE0$_H$	8001 FFFF$_H$
A001 FFE0$_H$	A001 FFFF$_H$
Secondary ABM Header	
8003 FFE0$_H$	8003 FFFF$_H$
A003 FFE0$_H$	A003 FFFF$_H$

Table 4-9 shows the structure of the ABM headers.

Table 4-9 Structure of ABM Headers

Address[1)]	Value	Function
XXXX XXE0$_H$	32-bit start address	Program/code start address
XXXX XXE4$_H$	DEADBEEF$_H$	Identifier string
XXXX XXE8$_H$	32-bit address (checksum start)	32-bit aligned start address of memory range to be checked
XXXX XXEC$_H$	32-bit address (checksum end)	32-bit aligned end address (last word address) of memory range to be checked
XXXX XXF0$_H$	32-bit CRC value CRC$_{RANGE}$	Expected 32-bit CRC result for memory range to be checked
XXXX XXF4$_H$	CRC$_{RANGE}$ inverted	Inverted expected 32-bit CRC result for memory range to be checked
XXXX XXF8$_H$	32-bit CRC value CRC$_{HEAD}$	CRC result of current ABM header from offset (byte) address E0$_H$ to F7$_H$
XXXX XXFC$_H$	CRC$_{HEAD}$ inverted	Inverted CRC result of current ABM header from offset (byte) address E0$_H$ to F7$_H$

1) XXXX XX is 8001 FF$_H$/A001 FF$_H$ for primary ABM header and 8003 FF$_H$/A003 FF$_H$ for secondary ABM header.

The time needed for CRC generation depends on the length of the memory range to be checked. In case of internal PFLASH check, the duration can be roughly calculated as follows:

- CRC of header: To be defined
- CRC of a Flash space of 400 bytes: To be defined

5 System Control Unit

The System Control Unit (SCU) of the TC1766 handles system control tasks. All the system functions described in this chapter are tightly coupled; thus, they are conveniently handled by one unit, the SCU.

The system tasks of the SCU that are covered in this chapter are:

5.1 Power Management

This section describes the power management system of the TC1766. Topics covered here include the internal system interfaces, external interfaces, state diagrams, and the operations of the CPU and peripherals. The power management state machine is also described.

5.1.1 Power Management Overview

The TC1766 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 5-1**, there are three power management modes available:

- Run Mode
- Idle Mode
- Sleep Mode

Table 5-1 Power Management Mode Summary

Mode	Description
Run	The system is fully operational. All clocks and peripherals are enabled, as determined by software.
Idle	The CPU clock 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. All peripherals remain powered and clocked. Processor memory is accessible to peripherals. A reset, Watchdog Timer event, a falling edge on the NMI pin, or any enabled interrupt event will return the system to Run Mode.
Sleep	The system clock signal is distributed only to those peripherals programmed to operate in Sleep Mode. The other peripheral module will be shut down by the suspend signal. Interrupts from operating peripherals, the Watchdog Timer, a falling edge on the NMI pin, or a reset event will return the system to Run Mode. Entering this state requires an orderly shut-down controlled by the Power Management State Machine.

The operation of each system component in each of these states can be configured by software. The power-management modes provide flexible reduction of power consumption through a combination of techniques, including:

- Stopping the CPU clock
- Stopping the clocks of other system components individually
- Clock-speed reduction of some peripheral components individually

The Power Management State Machine (PMSM) controls the power management mode of all system components during Run Mode, Idle Mode, and Sleep Mode. The PMSM continues to operate in Idle Mode and Sleep Mode, even if all other system components have been disabled, so that it can reawaken the system as needed. This flexibility in power management provides minimum power consumption for any application.

Besides these explicit software-controlled power-saving modes, in the TC1766 special attention has been paid to automatic power-saving in those operating units which are currently not required or idle. In that case they are shut off automatically until their operation is required again.

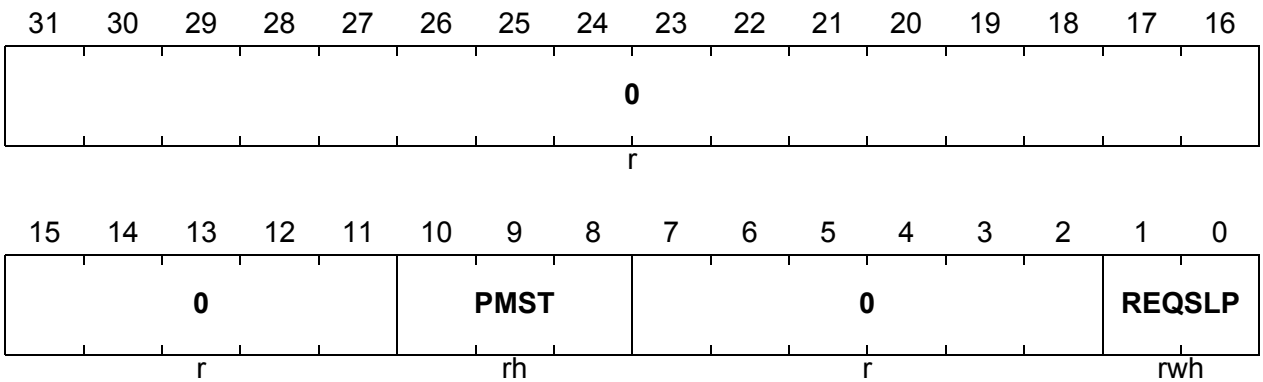
In typical operation, Idle Mode and Sleep Mode may be entered and exited frequently during the run time of an application. For example, system software will typically cause the CPU to enter Idle Mode each time it has to wait for an interrupt before continuing its tasks. In Sleep Mode and Idle Mode, wake-up is performed automatically when any enabled interrupt signal is detected, or the count value (WDT_SR.WDTTIM) changes from $7FFF_H$ to 8000_H.

5.1.2 Power Management Control and Status Register, PMG_CSR

The set of registers used for power management is divided between central TC1766 components and peripheral components.

The PMG_CSR register provides software control and status information for the PMSM. 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 PMSM directly controls TC1766 components such as the CPU, but indirectly controls peripheral components through their clock control registers.

PMG_CSR
Power Management Control and Status Register
($F0000034_H$) **Reset Value: $0000\ 0100_H$**



Field	Bits	Type	Description
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 enabled interrupt, or when bit 15 of the Watchdog Timer count register (the WDT_SR.TIM[15] bit) changes from 0 to 1.

Field	Bits	Type	Description
PMST	[10:8]	rh	**Power Management State Machine 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 Undefined, reserved 110_B Undefined, reserved 111_B Undefined, reserved
0	[7:2], [31:11]	r	**Reserved** Read as 0; should be written with 0.

5.1.3 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 TC1766 system components in all power management modes.

5.1.3.1 Idle Mode

The Idle Mode is requested by software when writing to register PMG_CSR with bit field REQSLP = 01_B. After requesting the Idle Mode, the power management state machine posts an idle request signal to the CPU. The CPU finishes its current operation, sends an acknowledge signal back to the PMSM, and then enters an inactive state in which the CPU clocks and the DMI and PMI memory units are shut off.

Other system components that are able to write to register PMG_CSR can also request the Idle Mode. For example, the PCP or DMA controller can request Idle Mode by writing to the PMG_CSR register.

During Idle Mode, memory accesses to the DMI and PMI via the local memory bus 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 signal is received from an enabled interrupt source.
- An NMI request is received either from an external source via the $\overline{\text{NMI}}$ pin.
- An external power-on reset signal ($\overline{\text{PORST}}$), or hardware reset signal ($\overline{\text{HDRST}}$) is received.
- A software reset is requested by an FPI Bus agent by writing to the reset request register RST_REQ.

If any of these conditions arise, the TC1766 immediately awakens and returns to Run Mode. If it is awakened by a hardware or software reset signal, the TC1766 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 signal or interrupt signal, the CPU will immediately vector to the appropriate handler.

5.1.3.2 Sleep Mode

The Idle Mode is requested by software when writing to register PMG_CSR with bit field 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 PMSM receives the Idle acknowledge signal back from the CPU, it proceeds with the second step.
2. A sleep signal is broadcasted on the FPI Bus. Each FPI Bus interface unit receives this signal. The response of each FPI Bus unit to the sleep signal is determined by its clock control register. These clock control registers must have been previously configured by software.

TC1766 State During Sleep Mode

Sleep Mode is disabled for a unit if its clock control register bit EDIS is set to 1. The sleep signal is ignored in this case and the corresponding unit continues normal operation. If EDIS = 0, Sleep Mode is enabled for this unit and the sleep signal will cause this unit to enter Sleep Mode. Two actions then occur:

- The unit's bus interface finishes whatever transaction was in progress when the signal was received.
- The unit's functions are suspended.

The TriCore architecture qualifies the actions in step 2 as follows. Depending on the module's clock control register Fast Shut-Off Enable bit, FSOE, the module's clocks are either immediately stopped (FSOE = 1), or the unit is allowed to finish ongoing operations (FSOE = 0) before the clocks are stopped. For example, setting FSOE to 1 for a serial port will stop all actions in the serial port immediately when the sleep signal is received. Ongoing transmissions or receptions will be aborted. If FSOE is 0, ongoing transmissions or receptions will be completed, and then the clock will be shut off. The purpose of setting 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.

5.1.3.3 States of TC1766 Units in Power Management Modes

Table 5-2 summarizes the state of the various units of the TC1766 during Run Mode, Idle Mode, and Sleep Mode.

Table 5-2 States of TC1766 Units in Power Management Modes

Unit	Run Mode	Idle Mode	Sleep Mode
Main Oscillator & PLL	On	On	On
CPU	Executing	Idle	Idle
DMI & PMI	Active	Idle, but accessible	Idle, but accessible
PMU	Active	Accessible	Accessible
Flash Module	Active	Active	Powered down
Watchdog Timer	Functioning as programmed	Functioning as programmed	Functioning as programmed
FPI Bus Peripherals	Functioning as programmed	Functioning as programmed	Peripherals with suspend enabled are shut down
Debug Units	Functioning	Functioning	Functioning
Ports	Functioning as programmed	Functioning as programmed	Functioning as programmed

5.2 Configuration Input Sampling

Several device pins of the TC1766 are latched either by a hardware reset or power-on reset operation. The latched states of such pins may directly determine the start-up configuration and conditions of the TC1766. The latched pins are divided into two types:

- Software configuration inputs, which are latched into a register and have no direct influence on the device start-up configuration.
- Hardware configuration inputs, which directly influence the start-up behavior of the TC1766.

Table 5-3 determines which pins are latched either by a hardware or power-on reset operation. The state of the corresponding signals is always latched at the rising edges of $\overline{\text{HDRST}}$ or $\overline{\text{PORST}}$.

Table 5-3 Configuration Input Sampling

Configuration Type	Reset Signal	Pins	Remark
Software Configuration	$\overline{\text{HDRST}}$	P0.[15:0]	The states of these pins are latched in register SCU_SCLIR (see also **Section 9.3.3.2** on **Page 9-27**).
Hardware Configuration	$\overline{\text{HDRST}}$	P4.[3:0] (HWFCG[3:0])	The states of these pins are used for selection of the Boot option (see **Table 4-3** on **Page 4-12**). Its state is latched in bit field RST_SR.HWCFG.
	$\overline{\text{PORST}}$	$\overline{\text{BRKIN}}$	The state of this pin is used for selection of the Boot option and for debug purposes.
		P3.1/TXD0A	The state of this pin determines whether oscillator bypass mode is selected or not (see also **Table 3-1** on **Page 3-9**).
		BYPASS	The state of this pin determines whether P3.1/TXD1A is evaluated or not. Additionally, the latched state of BYPASS controls the noise suppression filters of pins $\overline{\text{PORST}}$, $\overline{\text{HDRST}}$, and $\overline{\text{NMI}}$.
		$\overline{\text{TESTMODE}}$	The state of this pin is used for device test mode selection. For normal device operation, this pin should be always be set to 1 level.

5.3 External Request Unit (ERU)

In many cases, external events are used to trigger actions inside a microcontroller. This can be done by triggering external interrupts, which are then serviced by the CPU. Another possibility is to activate module functions that are related to external signals, such as the start of analog-to-digital conversions, counting control for a timer unit or to start a DMA transfer.

Due to the large variety of possible conditions of external signals, the simple generation of interrupt events after the detection of edges of the external signals might not be enough. Therefore, it can become necessary to check for patterns (gating of functions) or to reroute trigger events from one block to another.

In the TC1766, a flexible unit External Request Unit (ERU) makes it possible to generate trigger events that are able to generate interrupts, trigger a DMA transfers, or start analog-to-digital conversions.

Features

- Edge-detection of an input signal (rising, falling, or both edges)
- Event generation with combined conditions of input signals
- Pattern detection of input pins (e.g gating functionality)
- Flexible input signal selection
- Generation of multiple output trigger events possible

The block diagram of the ERU is shown **Figure 5-1**.

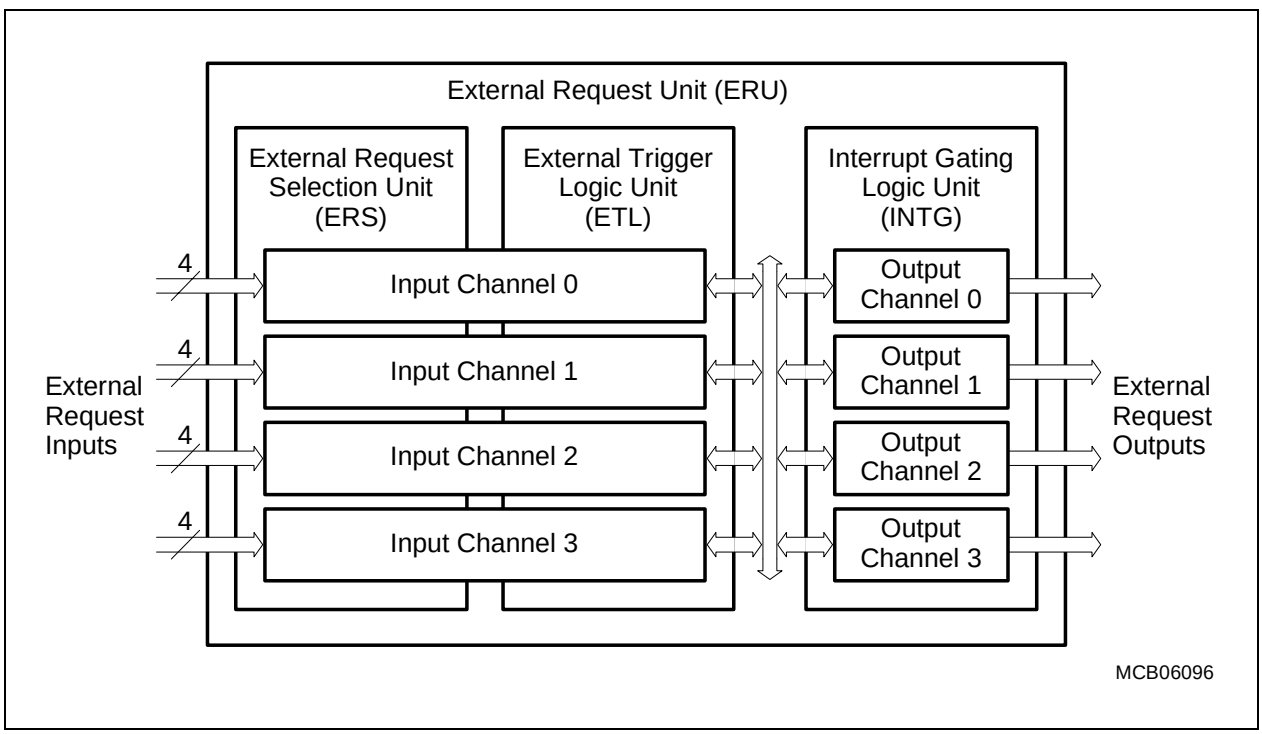


Figure 5-1 External Request Unit Block Diagram

The External Request Select unit (ERS) has three parts:

- The External Request Selection unit (ERS) selects one input signal for each input channel.
- The External Trigger Logic unit (ETL) selects the input channel's input signal trigger condition, and provides output signals to the output channel.
- The Interrupt Gating unit (INTG) combines the detected events of the ETL and provides pattern recognition for each output channel, as well as triggering and gating functions for A/D converters or DMA controller.

The functional blocks of the ERU are described in detail in the following sections.

5.3.1 Input Channel

Figure 5-2 shows the basic structure of an input channel. The input channel has two parts, one part located in the ERS and one part located in the ETL. Two of the four input channels are always controlled by one External Input Channel Register EICRn (n = 1, 0; n = 0 applies to input channel 0 and 1, n = 1 applies to input channels 2 and 3). In the following description, the index number "x" indicates the input channel number.

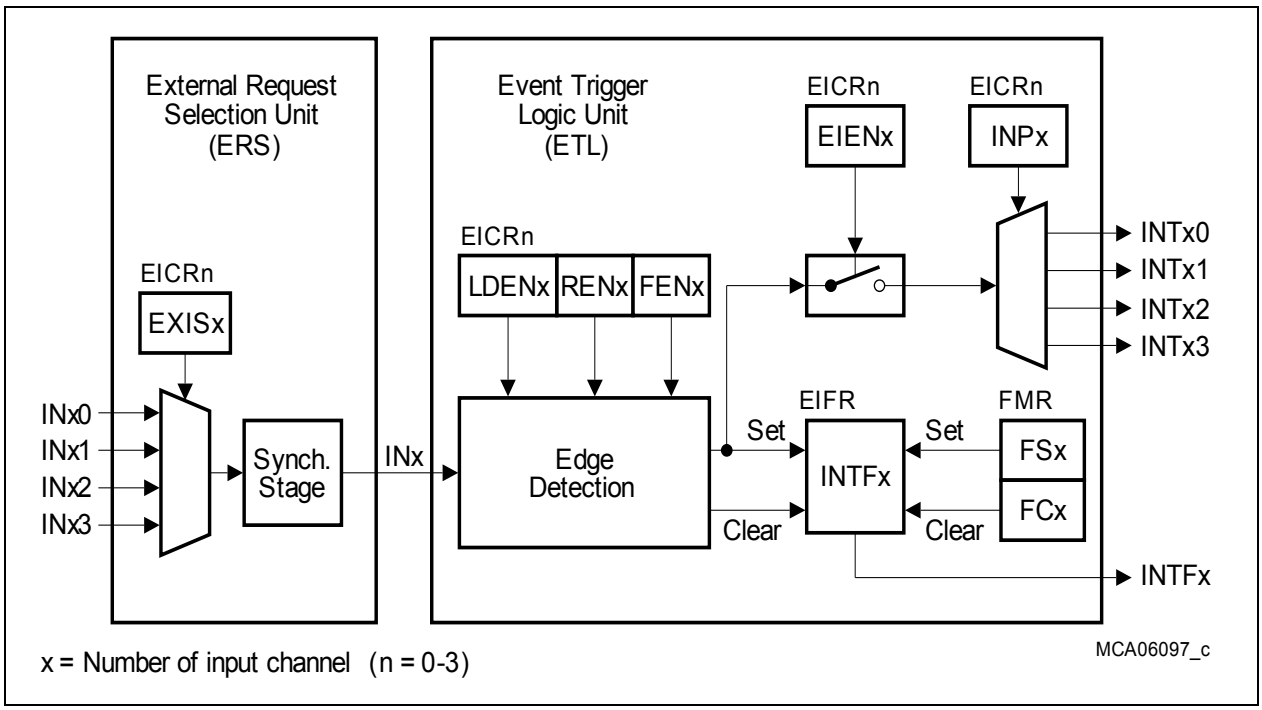


Figure 5-2 External Request Unit Input Channel

An input channel of the ERS contains a 4-to-1 input multiplexer (controlled by bit field EICRn.EXISx) that selects one of four possible request input lines INx[3:0]. The selected signal then becomes synchronized. This is required for further control purposes that are done in the Event Trigger Logic Unit (ETL) of the input channel.

The ETL of the input channel analyzes the synchronized output signal INx of the ERS by an edge-detection block. This edge detection block generates a pulse (one clock cycle long) when a signal transition is detected. The selection, which signal transition (edge) should generate a pulse (event) is done by two control bits, one for the rising edge (EICRn.RENx) and one for the falling edge (EICRn.FENx). Therefore, only a rising or falling edge, or both edges of the input signal can be detected. When a selected signal transition is detected, the external interrupt flag EIFRn.INTFx is always set by hardware, even if it has been set previously. The interrupt flag EIFRn.INTFx is available as output signal INTFx of the ETL.

The hardware reset condition for the EIFRn.INTFx can be controlled in two different ways. These two modes are selected by bit EIFRn.LDENx.

- EIFRn.LDENx = 0:
  Flag EIFRn.INTFx is used as a sticky bit, indicating that the selected signal transition has been detected by the edge-detection logic at least once. In this mode, bit EIFRn.INTFx can be cleared only by software.
- EIFRn.LDENx = 1:
  In this mode, flag EIFRn.INTFx is cleared by hardware if an edge is detected at the INx signal, that has not been selected. This means, EIFRn.INTFx is cleared by hardware at an INx rising edge when EICRn.FENx = 1 and EIFRn.INTFx is cleared by hardware at an INx falling edge when EICRn.RENx = 1. If both bits, EICRn.FENx and EICRn.RENx, are set, flag EIFRn.INTFx is never cleared by hardware. In this case, flag EIFRn.INTFx can only be cleared by software.

Flag EIFRn.INTFx can be set or cleared by software using the bits FSx or FCx in the flag modification register FMR. Writing to FMR with bit FMR.FSx sets flag EIFRn.INTFx. Writing to FMR with bit FMR.FCx clears flag EIFRn.INTFx. If both bits are set for one input channel x, a set operation of the flag will be executed.

Each output pulse of the edge-detection block that sets the EIFRn.INTFx flag by hardware can also be routed to one of the INTx[3:0] outputs of the ETL unit. The external interrupt enable bit EICRn.EIENx = 1 enables the interrupt output pulse at the INTx[3:0] outputs. Bit field EICRn.INPx controls which of the INTx[3:0] outputs is selected for the output pulse.

5.3.2 Output Channel

The output channel logic shown in **Figure 5-3** is built in for each of the four output channels.

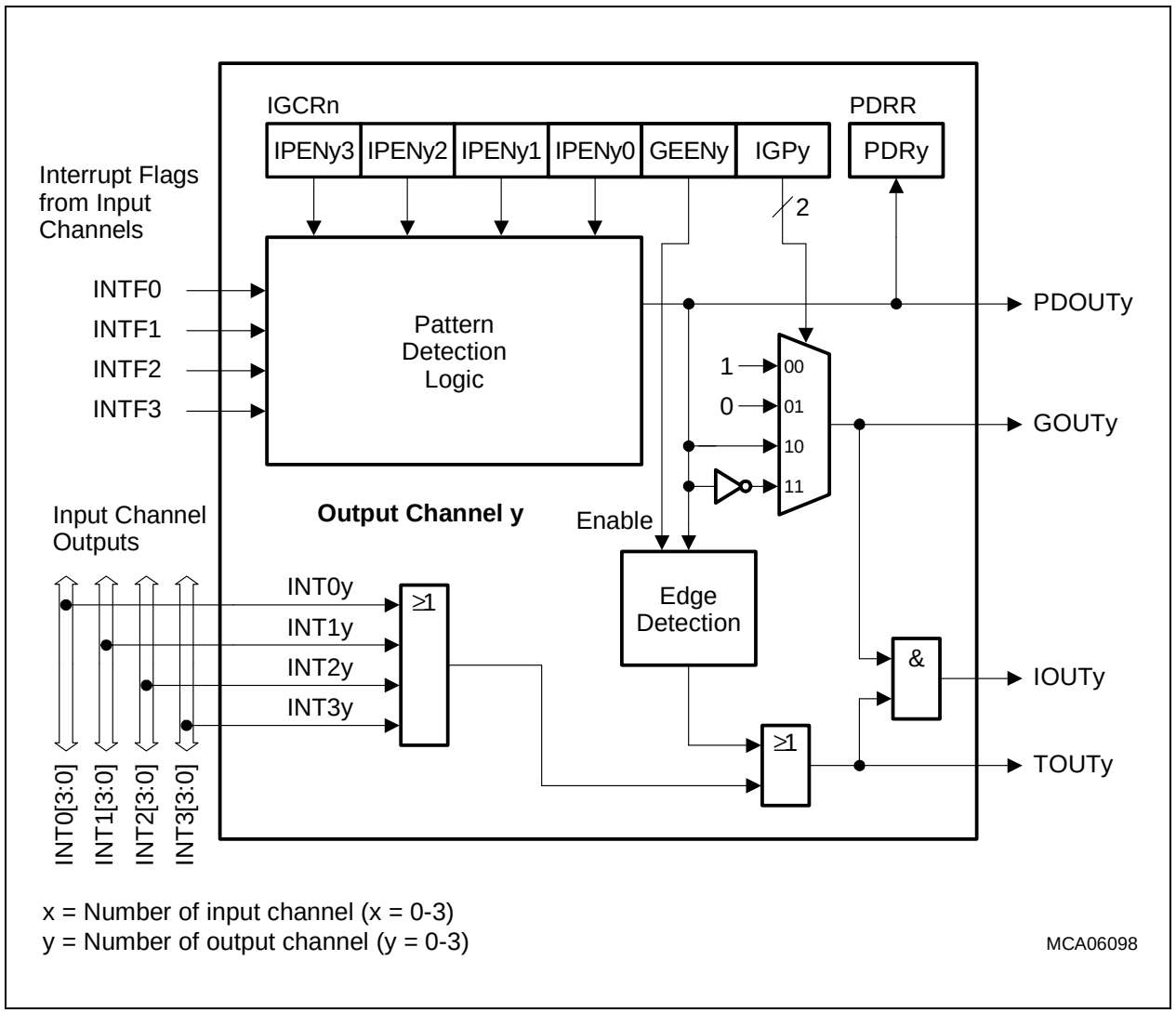


Figure 5-3 Interrupt Gating Logic (Output Channel y)

The output signals of the output channel y can be used to trigger peripherals actions, DMA requests, or interrupts.

PDOUTy Output Signal

The pattern detection output PDOUTy can be used to enable/disable peripheral functions while a certain condition (pattern) is detected (level controlled output). PDOUTy can be typically used to enable/disable ADC conversions. The output PDOUTy is active as long as the programmed condition of input signals (INTFx) is met.

Each INTFx output signal from input channel x is connected to each output channel y. Bits IPENy[3:0] determine whether flag INTFx of input channel x takes part in the pattern detection of output channel y. The output signal PDOUTy is an AND combination of all INTFx inputs that are enabled by IPENyx = 1. In other words, the PDOUTy signals becomes active (high) when the selected input channel interrupt flags are at active (high) level. The state of flag PDRR.PDRy indicates the actual state of the PDOUTy signal.

GOUTy Output Signal

The gating output GOUTy represents the programmable level of PDOUTy. This output signal is not connected/used in TC1766 but referred in the register descriptions.

TOUTy Output Signal

The trigger output TOUTy combines all related event trigger sources. This output can be used to request ADC conversions or to start other peripheral actions. TOUTy outputs pulses.

The incoming high-level active interrupt request pulses INT0y to INT3y from the event trigger logic are combined to one common interrupt request for the corresponding output channel. The incoming signals remain inactive when an event has been detected but another output channel has been selected by INPx in the channel event trigger logic. As a result, only those interrupt requests are taken into account, that are targeting this output channel (y).

IOUT Output Signal

The interrupt output IOUTy can be connected to an interrupt node, and combines the gating functionality GOUTy (programmed level of PDOUTy) and the trigger functionality of TOUTy. It can also be used to trigger module actions, such as DMA requests or GPTA actions. IOUTy outputs pulses.

Output Channel Control Logic

A useful additional feature is the possibility to gate the incoming interrupt requests with the indication flags INTFx (x = 0-3) coming from the Event Trigger Logic of the Input Channel x. This allows more gating capability for the interrupt generation. The output channel can be activated when a trigger event occurs while a certain pattern is detected (IGP = 10_B), or while this pattern is not detected (IGP = 11_B). If an indication flag should not be taken into account for the gating of the interrupt requests, the related IPENyx bit has to be 0.

Bit GEENy = 1 enables the generation of a trigger event for output channel y 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.

The output of the gating AND of all interrupts request are also delayed by one clock cycle to detect a change of the gating status (XOR), corresponding to the pattern detection. The bit GEENy (gating edge enable) makes it possible to generate a pulse whenever the gating status changes. Combined with the bit field IGPy, an event can be generated when a pattern is detected, when a pattern is no longer detected, or if both conditions are true.

5.3.3 External Request Unit Implementation

This section describes how the ERU is interconnected within other on-chip modules.

The connections of the ERU input lines and the connections between input and output channels are shown in **Figure 5-4**. **Figure 5-5** shows how the outputs of the output channels are connected to the A/D converters ADC0 and ADC1, to the DMA controller, and to the GPTA modules.

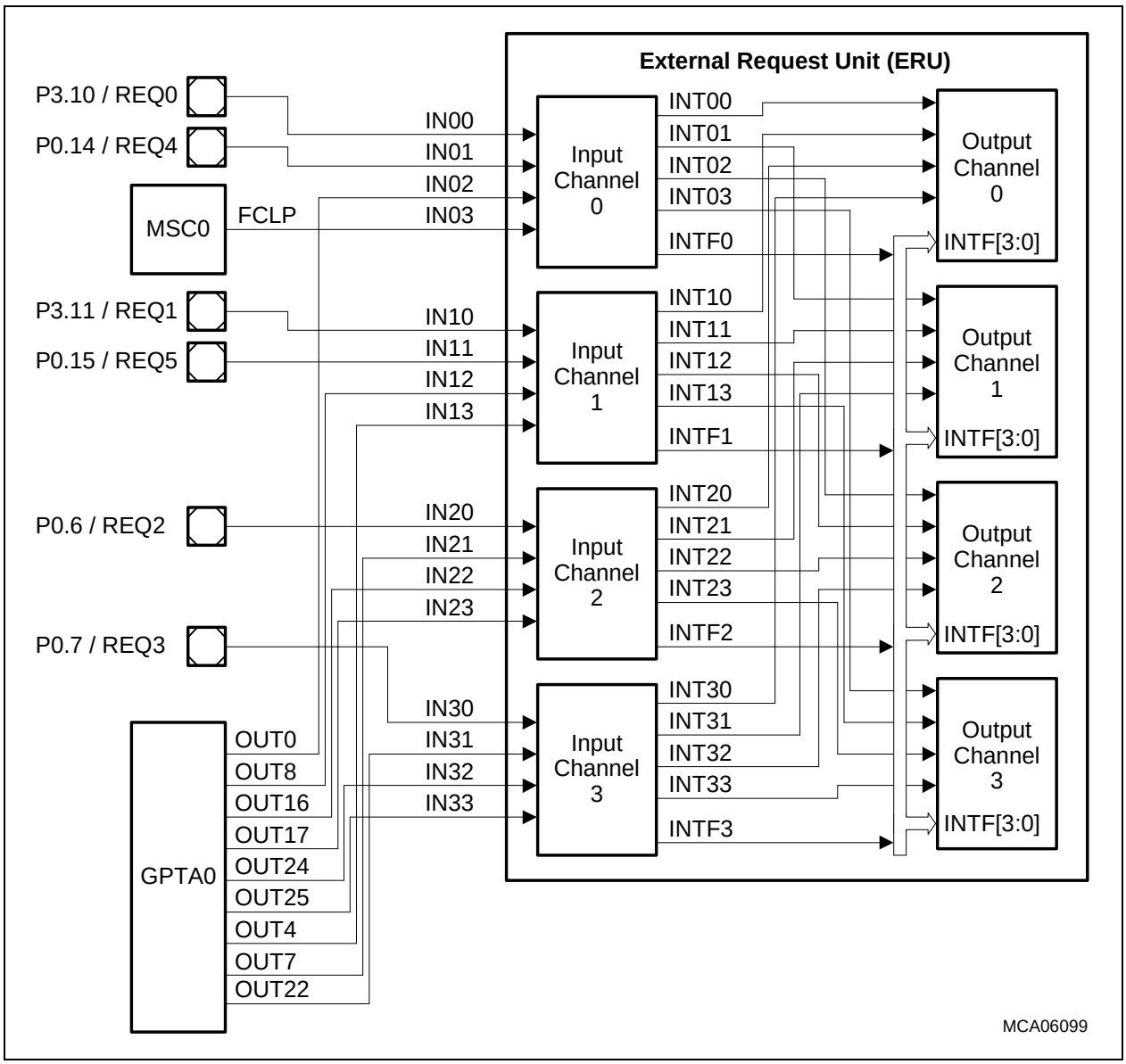


Figure 5-4 Input Connections of External Request Unit

As shown in **Figure 5-5**, additional trigger lines ROUT are generated by the GPTA0 module. These lines detect rising edges at three dedicated GPTA0 outputs and can be used to trigger ADC actions, for example. The GPTA0 outputs are connected to an edge-detection logic to obtain a trigger pulse each time a rising edge is detected.

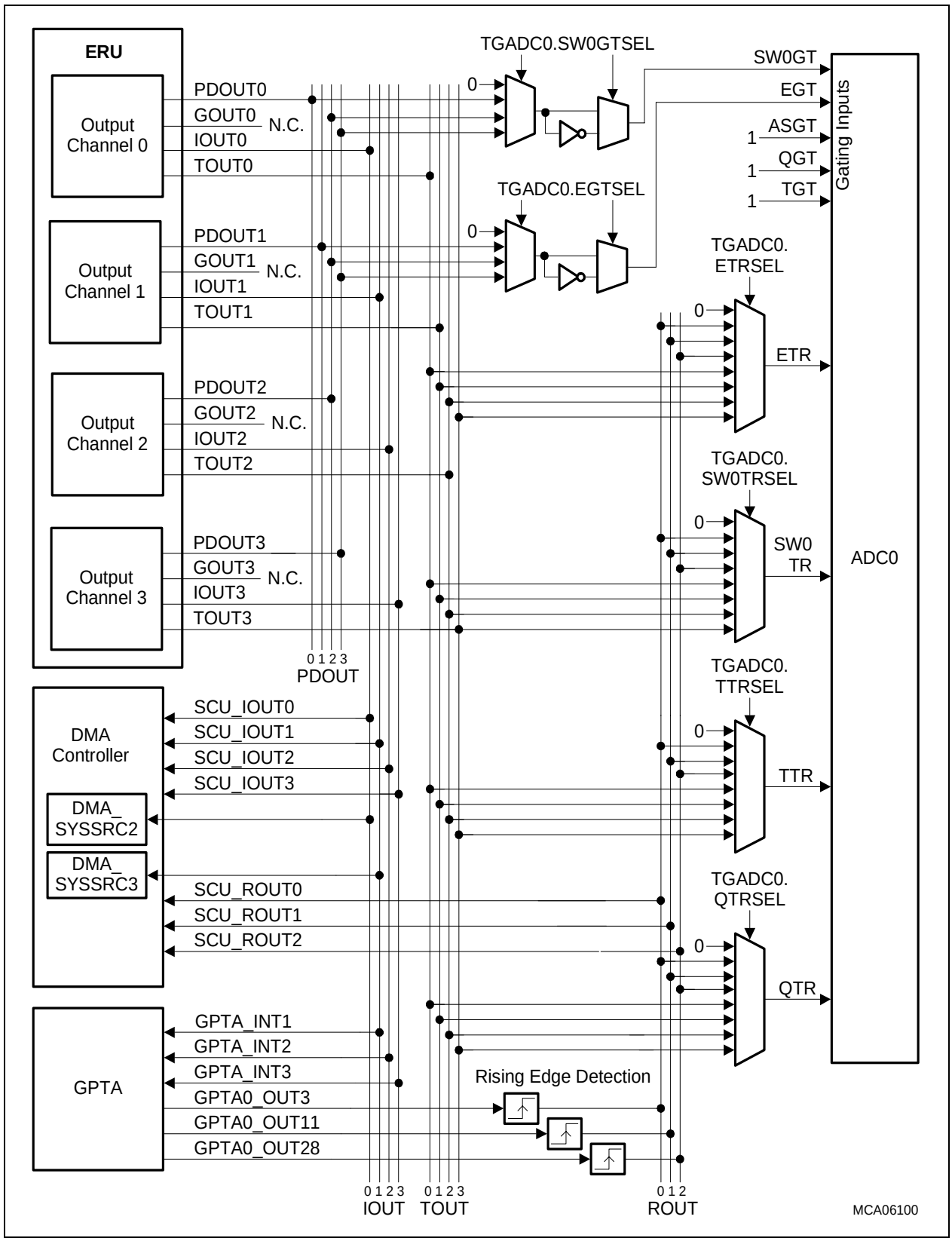


Figure 5-5 Output Connections of External Request Unit

5.3.4 External Request Unit Registers

Table 5-4 refers to the registers associated with the ERU kernel as well as the TC1766 implementation-specific ERU registers, which control the ERU connections with the A/D converter ADC0.

Table 5-4 : External Trigger Request Unit Kernel Registers

Register Short Name	Register Long Name	Description see
ERU Kernel Registers		
EICR0	External Input Channel Register 0	Page 5-19
EICR1	External Input Channel Register 1	Page 5-22
EIFR	External Input Flag Register	Page 5-25
FMR	Flag Modification Register	Page 5-26
PDRR	Pattern Detection Result Register	Page 5-27
IGCR0	Interrupt Gating Register 0	Page 5-28
IGCR1	Interrupt Gating Register 1	Page 5-30
Implementation-specific ERU Register		
TGADC0	Trigger Gating ADC0 Register	Page 5-33

The External Input Channel Registers EICR0 and EICR1 for the external input channels 0 to 3 contain bits to configure the input gating logic IGL and the event trigger logic ETL. A maximum of 12 input channels are supported by one input unit (defined by the maximum number of IGCRy.IPENx bits).

EICR0
External Input Channel Register 0

($F0000080_H$) **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 signal IN0. 00_B Input IN00 is selected. 01_B Input IN01 is selected. 10_B Input IN02 is selected. 11_B Input IN03 is selected.
FEN0	8	rw	**Falling Edge Enable 0** This bit determines if the falling edge of signal IN0 is used to set bit INTF0. 0_B The falling edge is not used. 1_B The detection of a falling edge of IN0 generates a trigger event (INTF0 becomes set).
REN0	9	rw	**Rising Edge Enable 0** This bit determines if the rising edge of signal IN0 is used to set bit INTF0. 0_B The rising edge is not used. 1_B The detection of a rising edge of IN0 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 signal IN0 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 Interrupt 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	**Interrupt Node Pointer** This bit field determines the destination (output channel) for trigger event 0 (if enabled by EIEN0). $X00_B$ The event of input channel 0 triggers output channel 0 (signal INT00). $X01_B$ The event of input channel 0 triggers output channel 1 (signal INT01). $X10_B$ The event of input channel 0 triggers output channel 2 (signal INT02). $X11_B$ The event of input channel 0 triggers output channel 3 (signal INT03).
EXIS1	[21:20]	rw	**External Input Selection 1** This bit field determines which input line is selected for signal IN1. 00_B Input IN10 is selected. 01_B Input IN11 is selected. 10_B Input IN12 is selected. 11_B Input IN13 is selected.
FEN1	24	rw	**Falling Edge Enable 1** This bit determines if the falling edge of signal IN1 is used to set bit INTF1. 0_B The falling edge is not used. 1_B The detection of a falling edge of IN1 generates a trigger event (INTF1 becomes set).

Field	Bits	Type	Description
REN1	25	rw	**Rising Edge Enable 1** This bit determines if the rising edge of signal IN1 is used to set bit INTF1. 0_B The rising edge is not used. 1_B The detection of a rising edge of IN1 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 signal IN1 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 Interrupt 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.
INP1	[30:28]	rw	**Interrupt Node Pointer** This bit field determines the destination (output channel) for trigger event 1 (if enabled by EIEN1). $X00_B$ The event of input channel 1 triggers output channel 0 (signal INT10). $X01_B$ The event of input channel 1 triggers output channel 1 (signal INT11). $X10_B$ The event of input channel 1 triggers output channel 2 (signal INT12). $X11_B$ The event of input channel 1 triggers output channel 3 (signal INT13).
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

(F0000084$_H$) 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			

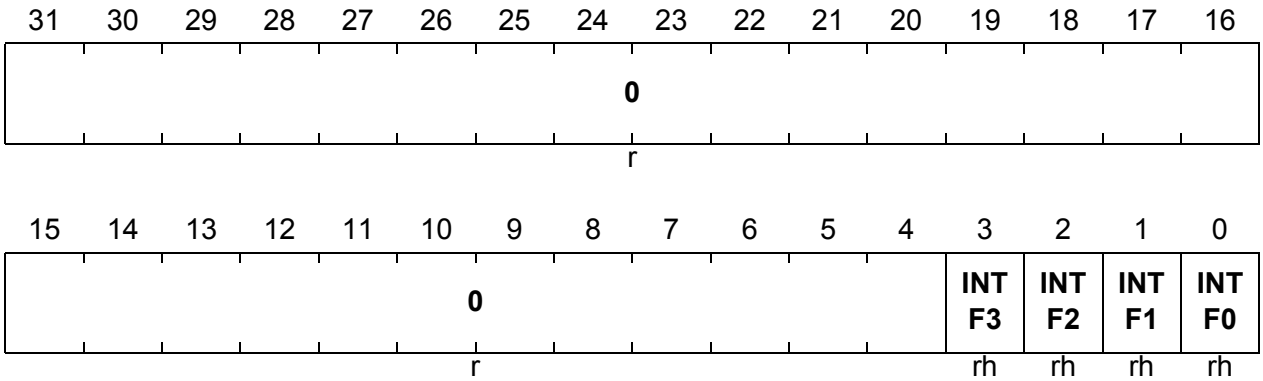
Field	Bits	Type	Description
EXIS2	[5:4]	rw	**External Input Selection 2** This bit field determines which input line is selected for signal IN2. 00_B Input IN20 is selected. 01_B Input IN21 is selected. 10_B Input IN22 is selected. 11_B Input IN23 is selected.
FEN2	8	rw	**Falling Edge Enable 2** This bit determines if the falling edge of signal IN2 is used to set bit INTF2. 0_B The falling edge is not used. 1_B The detection of a falling edge of IN2 generates a trigger event (INTF3 becomes set).
REN2	9	rw	**Rising Edge Enable 2** This bit determines if the rising edge of signal IN2 is used to set bit INTF2. 0_B The rising edge is not used. 1_B The detection of a rising edge of IN2 generates a trigger event (INTF2 becomes set).

Field	Bits	Type	Description
LDEN2	10	rw	**Level Detection Enable 2** This bit determines if bit INTF2 is cleared automatically if an edge of the input signal IN2 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 Interrupt 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.
INP2	[14:12]	rw	**Interrupt Node Pointer** This bit field determines the destination (output channel) for trigger event 2 (if enabled by EIEN2). $X00_B$ The event of input channel 2 triggers output channel 0 (signal INT20). $X01_B$ The event of input channel 2 triggers output channel 1 (signal INT21). $X10_B$ The event of input channel 2 triggers output channel 2 (signal INT22). $X11_B$ The event of input channel 2 triggers output channel 3 (signal INT23).
EXIS3	[21:20]	rw	**External Input Selection 3** This bit field determines which input line is selected for signal IN3. 00_B Input IN30 is selected. 01_B Input IN31 is selected. 10_B Input IN32 is selected. 11_B Input IN33 is selected.
FEN3	24	rw	**Falling Edge Enable 3** This bit determines if the falling edge of signal IN3 is used to set bit INTF3. 0_B The falling edge is not used. 1_B The detection of a falling edge of IN3 generates a trigger event (INTF3 becomes set).

Field	Bits	Type	Description
REN3	25	rw	**Rising Edge Enable 3** This bit determines if the rising edge of signal IN3 is used to set bit INTF3. 0_B The rising edge is not used. 1_B The detection of a rising edge of IN3 generates a trigger event (INTF3 becomes set).
LDEN3	26	rw	**Level Detection Enable 3** This bit determines if bit INTF3 is cleared automatically if an edge of the input signal IN3 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). $X00_B$ The event of input channel 3 triggers output channel 0 (signal INT30). $X01_B$ The event of input channel 3 triggers output channel 1 (signal INT31). $X10_B$ The event of input channel 3 triggers output channel 2 (signal INT32). $X11_B$ The event of input channel 3 triggers output channel 3 (signal INT33).
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 interrupt 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.

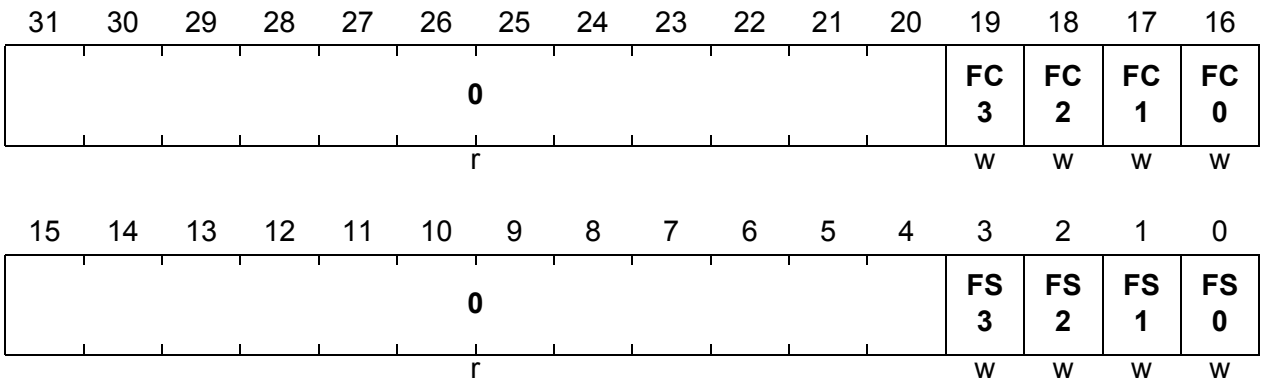
EIFR
External Input Flag Register (F0000088$_H$) Reset Value: 0000 0000$_H$



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 SW (if LDENx = 0).
0	[31:4]	r	**Reserved** Read as 0; should be written with 0.

The Flag Modification Register is a write-only register that is used to set and to clear the bits INTFx in register EIFR. If a set event and a clear event (hardware or software) for bit INTFx occur at the same time, the set event is taken into account.

FMR
Flag Modification Register (F000008C$_H$) Reset Value: 0000 0000$_H$

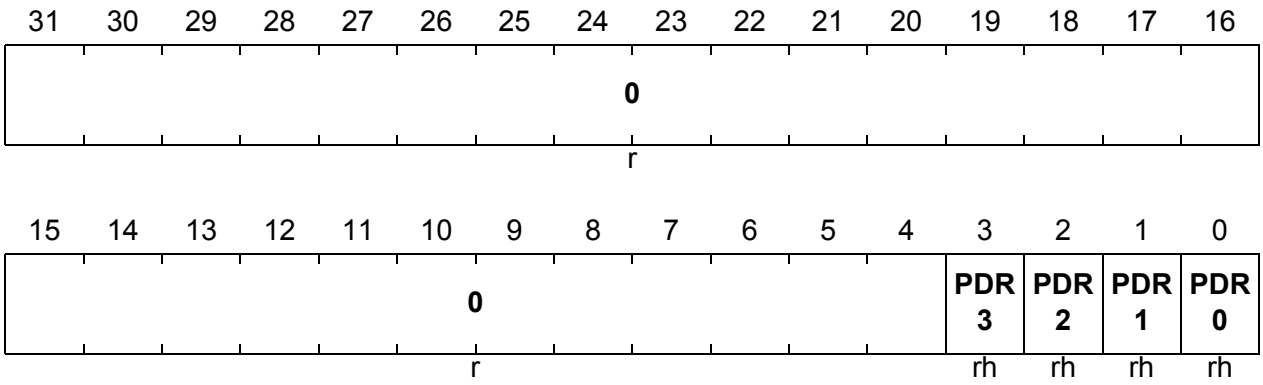


Field	Bits	Type	Description
FSx (x = 0-3)	x	w	**Set Flag INTFx for Channel x** Setting this bit will set the corresponding bit INTFx in register EIFR. Reading this bit always delivers a 0. 0_B The bit x in register EIFR is not modified. 1_B The bit x in register EIFR is set.
FCx (x = 0-3)	16+x	w	**Reset 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.

Note: This register is virtual and does not contain any flip-flop.

The Pattern Detection Result Register, PDRR monitors the combinatorial output status of the pattern detection units.

PDRR
Pattern Detection Result Register ($F0000090_H$) **Reset Value: $0000\ 000F_H$**



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 channels 0 to 3 (e.g. for interrupt nodes or peripherals).

IGCR0
Interrupt Gating Register 0 ($F0000094_H$) 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 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 **($F0000098_H$)** **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

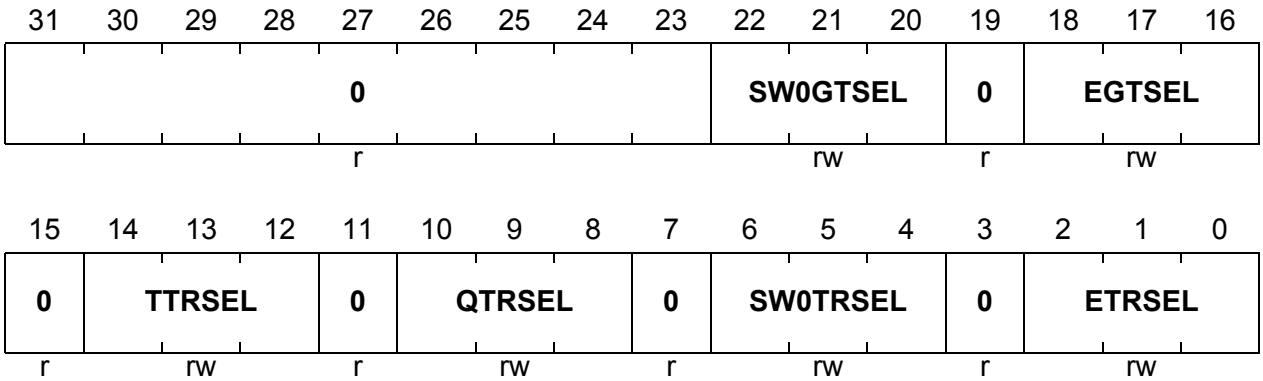
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IGP2		**GEEN2**	**0**									**IPEN23**	**IPEN22**	**IPEN21**	**IPEN20**
rw		rw	r									rw	rw	rw	rw

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.

The Trigger Gating ADC0 Register TGADC0 contains bit fields that determine the assignment of the output signals of the external request unit to the trigger and gating inputs of ADC0.

TGADC0
Trigger Gating ADC0 Register (F000009C$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
ETRSEL	[2:0]	rw	**External Trigger Request Selection** This bit determines which trigger source will be used for the external trigger request input ETR of ADC0. 000_B ETR is constant at 0 level (trigger function switched off). 001_B ROUT0 is connected to ETR. 010_B ROUT1 is connected to ETR. 011_B ROUT2 is connected to ETR. 100_B TOUT0 is connected to ETR. 101_B TOUT1 is connected to ETR. 110_B TOUT2 is connected to ETR. 111_B TOUT3 is connected to ETR.

Field	Bits	Type	Description
SW0TRSEL	[6:4]	rw	**SW0 Trigger Request Selection** This bit determines which trigger source will be used for the SW0 trigger request input SW0TR of ADC0. 000_B SW0TR is constant at 0 level (trigger function switched off). 001_B ROUT0 is connected to SW0TR. 010_B ROUT1 is connected to SW0TR. 011_B ROUT2 is connected to SW0TR. 100_B TOUT0 is connected to SW0TR. 101_B TOUT1 is connected to SW0TR. 110_B TOUT2 is connected to SW0TR. 111_B TOUT3 is connected to SW0TR.
QTRSEL	[10:8]	rw	**Queue Trigger Request Selection** This bit determines which trigger source will be used for the queue trigger request input QTR of ADC0. 000_B QTR is constant at 0 level (trigger function switched off). 001_B ROUT0 is connected to QTR. 010_B ROUT1 is connected to QTR. 011_B ROUT2 is connected to QTR. 100_B TOUT0 is connected to QTR. 101_B TOUT1 is connected to QTR. 110_B TOUT2 is connected to QTR. 111_B TOUT3 is connected to QTR.
TTRSEL	[14:12]	rw	**Timer Trigger Request Selection** This bit determines which trigger source will be used for the timer trigger request input TTR of ADC0. 000_B TTR is constant at 0 level (trigger function switched off). 001_B ROUT0 is connected to TTR. 010_B ROUT1 is connected to TTR. 011_B ROUT2 is connected to TTR. 100_B TOUT0 is connected to TTR. 101_B TOUT1 is connected to TTR. 110_B TOUT2 is connected to TTR. 111_B TOUT3 is connected to TTR.

Field	Bits	Type	Description
EGTSEL	[18:16]	rw	**External Gating Selection** This bit determines which trigger source will be used for the external gating input EGT of ADC0. 000_B EGT is constant at 0 level (externally triggered conversions permanently disabled). 001_B PDOUT1 is connected to EGT. 010_B PDOUT2 is connected to EGT. 011_B PDOUT3 is connected to EGT. 100_B EGT is constant at 1 level (externally triggered conversions permanently enabled). 101_B $\overline{\text{PDOUT1}}$ is connected to EGT. 110_B $\overline{\text{PDOUT2}}$ is connected to EGT. 111_B $\overline{\text{PDOUT3}}$ is connected to EGT.
SW0GTSEL	[22:20]	rw	**SW0 Gating Selection** This bit defines which trigger source will be used for the SW0 gating input SW0GT of ADC0. 000_B SW0GT is constant at 0 level (SW0 triggered conversions permanently disabled). 001_B PDOUT0 is connected to SW0GT. 010_B PDOUT2 is connected to SW0GT. 011_B PDOUT3 is connected to SW0GT. 100_B SW0GT is constant at 1 level (SW0 triggered conversions permanently enabled). 101_B $\overline{\text{PDOUT0}}$ is connected to SW0GT. 110_B $\overline{\text{PDOUT2}}$ is connected to SW0GT. 111_B $\overline{\text{PDOUT3}}$ is connected to SW0GT.
0	3, 7, 11, 15, 19, [31:23]	r	**Reserved** Read as 0; should be written with 0.

5.4 Special System Interrupts

For some of the possible interrupts in the system, the interrupt control logic is not directly controlled in the module, but via the SCU. For example, FPU interrupts are generated in the CPU, but these interrupts have to be processed outside the CPU.

5.4.1 Flash Interrupt

The flash module can generate an interrupt when the following conditions occur:

- End-of-busy state
- Protection error
- Sequence error
- Single-bit ECC error

Each source can be individually enabled and disabled. The detailed description of the flash interrupt generation can be found at **“Flash Interrupt Generation and Control” on Page 7-36**. The Flash interrupt uses the system interrupt node DMA_SYSSRC1 located in the DMA module.

5.4.2 FPU Interrupts

The FPU provides two interrupts outputs that are activated in case of error conditions:

- Signal FPU_Exception1, which is activated if any of the error flags **including** the inexact flag FX is set
- Signal FPU_Exception2, which is activated if any of the error flags **excluding** the inexact flag FX is set

Both interrupts are combined to one interrupt request output that is controlled by register DMA_SYSSCR0 in the DMA controller. Bit SCU_CON.FIEN determines whether an inexact condition will lead to an interrupt or not.

When an FPU interrupt is generated, the related FPU status flags are latched into the corresponding bits of the SCU Status Register SCU_STAT (see **Page 5-64**). Thus, the status of the last floating point instruction that caused an interrupt can be read from SCU_STAT.

The exception status information at the FPU output bus FPU_Exception_PSW is connected to the bits in the SCU_STAT register according to **Figure 5-6**.

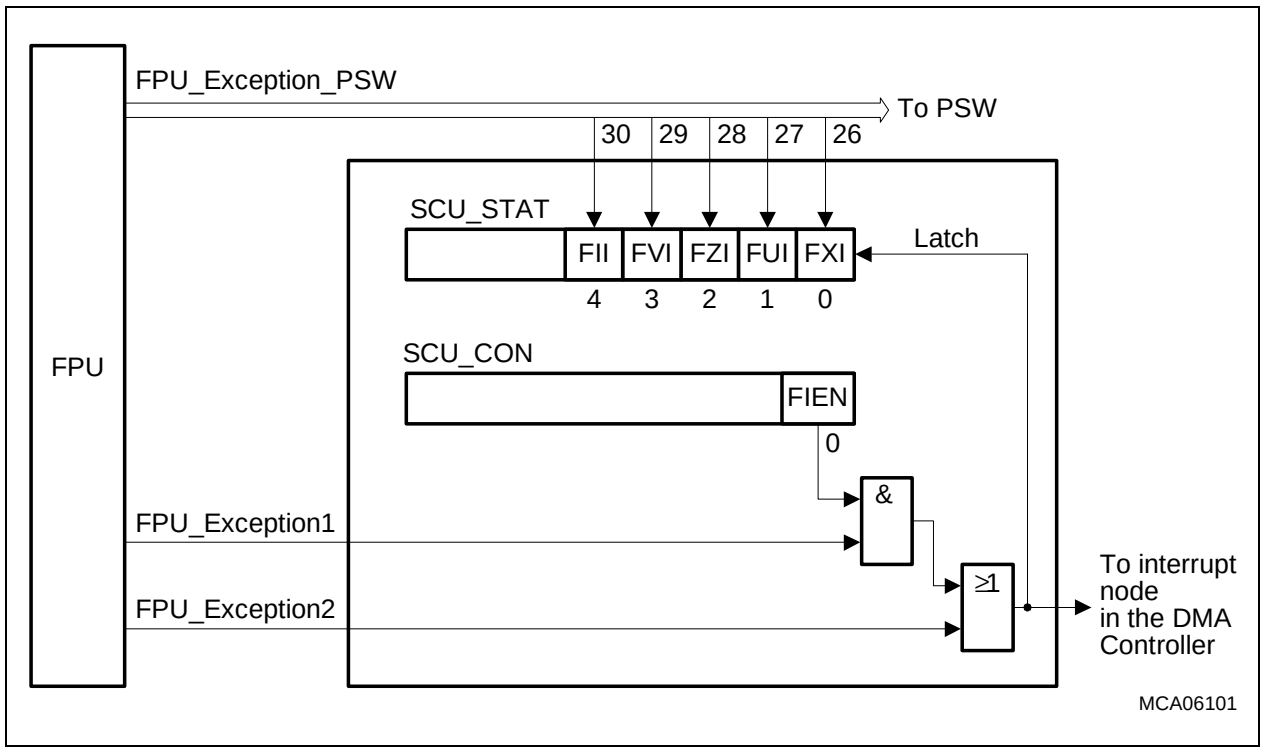


Figure 5-6 FPU Interrupt Control

The FPU interrupt uses the system interrupt node DMA_SYSSRC0 located in the DMA controller. A description of this register can be found on **Page 11-97**. Note that DMA_SYSSRC0.TOS should be written with 00_B. FPU interrupts should only be serviced by the CPU (and not by the PCP).

5.4.3 External Interrupts

As shown in **Figure 5-5** on **Page 5-17**, the External Request Unit provides two interrupt sources for events on external pins. The interrupt events can be defined by programming the corresponding registers in the ERU.

The two ERU interrupts are controlled by the system interrupt nodes DMA_SYSSRC2 and DMA_SYSSRC3 located in the DMA module. A description of this register can be found on **Page 11-97**.

5.5 SRAM Parity Control

In TC1766, several on-chip memory blocks are equipped with a parity error detection logic (see **Table 5-5**).

Table 5-5 On-chip SRAM with Parity Error Detection

Module	Memory Block	Parity Error Flag	Parity Enable Bit
DMI	Data Memory	PFL0	PEN0
PMI	Code Scratchpad RAM & Instruction Cache (SPRAM, ICACHE)	PFL1	PEN1
	Program Cache Tag RAM	PFL2	PEN2
PCP	Parameter RAM (PRAM)	PFL3	PEN3
	Code Memory (CMEM)	PFL4	PEN4
CAN	Module memory	PFL5	PEN5

Each memory block as defined in **Table 5-5** provides 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 (x = 0-5). If enabled by a parity enable control bit PENx, a non-maskable (NMI) is generated.

The bits PENx (parity error x enable) and PFLx (parity error flag) are located in the registers SCU_PETCR and SCU_PETSR, respectively.

Additional details about the NMI handling of SRAM parity errors are described in section **“SRAM Parity Error NMI” on Page 12-26**.

After a Boot ROM exit, the parity logic is enabled (initial value of SCU_STAT.PARAV = 1). During normal operation of the TC1766, the SRAM parity error logic can be completely disabled only once (PARAV clear) when the SCU_CON.RPARAV bit is written with 1. Note that if PARAV = 0, the SRAM parity error logic cannot be enabled anymore except by a power-on reset operation.

Before the parity logic of an SRAM memory block (except CAN memory) can be used the first time after a power-on reset operation (before setting its SCU_PETCR.PENx enable bit), the corresponding memories must be completely initialized by writing every memory location of it once (exceptions: Program Cache Tag RAM and ICACHE). Otherwise, unpredictable parity errors may occur after setting its SCU_PETCR.PENx enable bit.

Furthermore, before setting any SCU_PETCR.PENx enable bit after a power-on reset, register SCU_PETSR should be read once to clear parity error flags, that could have been set by parity logic tests during the Boot ROM code execution.

Figure 5-7 shows the functionality of the SRAM parity error control in the SCU.

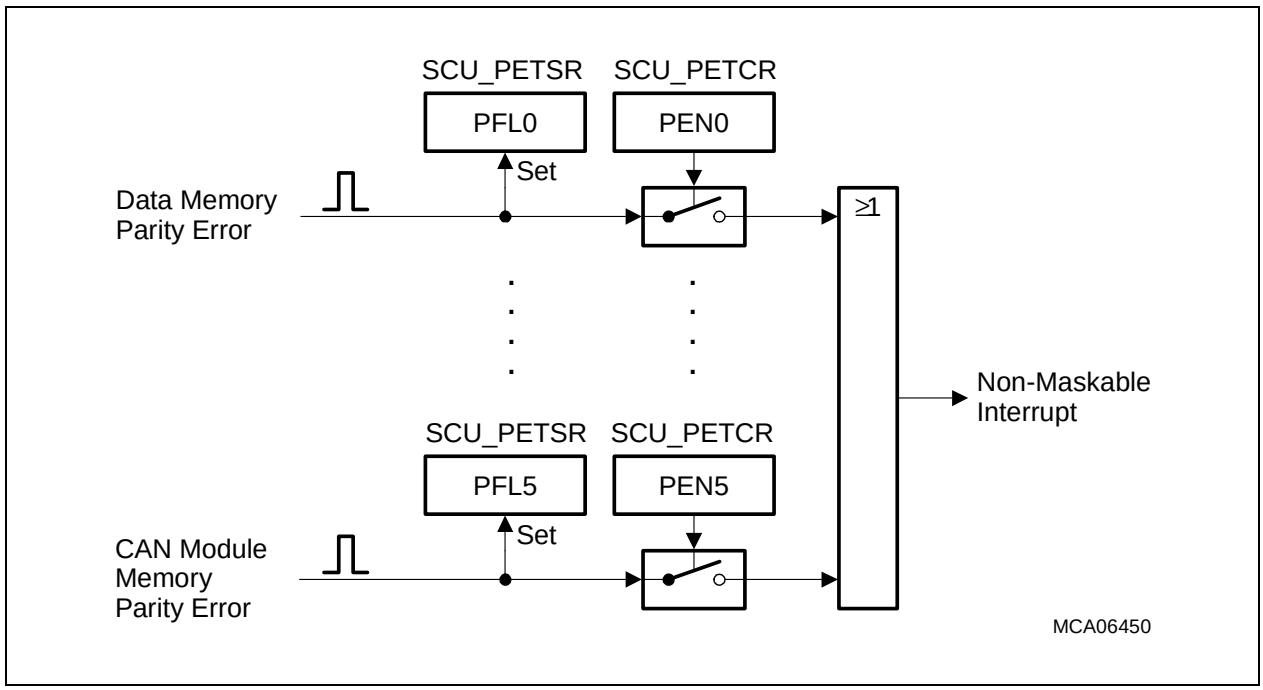
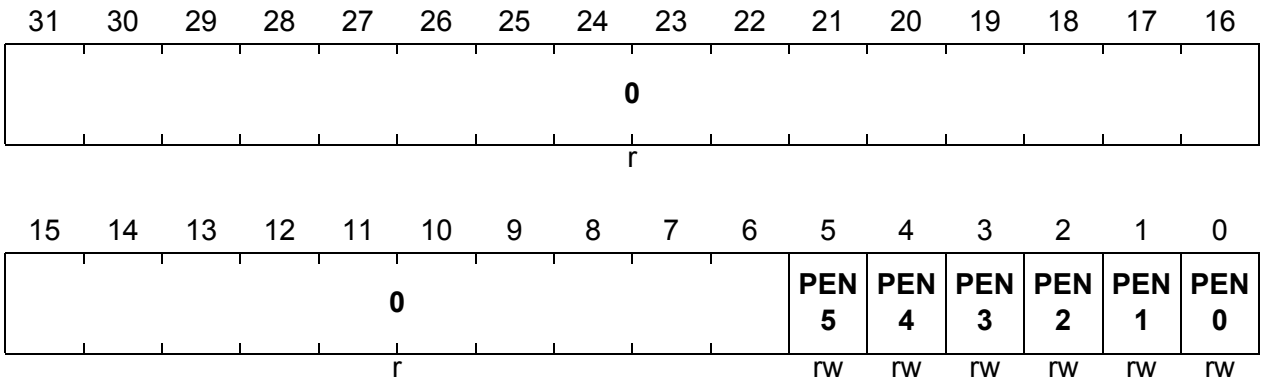


Figure 5-7 Control of SRAM Parity Error Detection in SCU

5.5.1 Parity Error Trap Control Register

SCU_PETCR
SCU Parity Error Trap Control Register

(F00000D0$_H$) **Reset Value: 0000 0000$_H$**

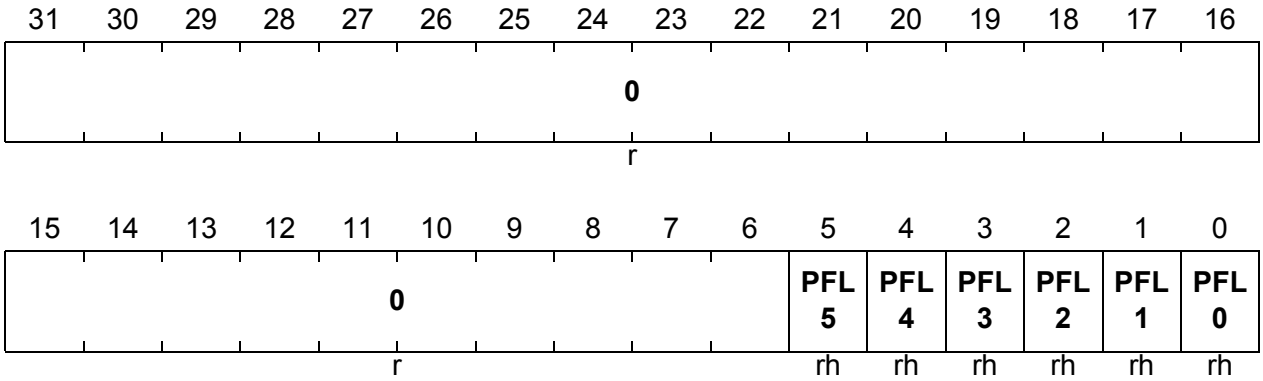


Field	Bits	Type	Description
PENx (x = 0-5)	x	rw	**Parity Error Trap Enable for SRAM Module x** These bits determine whether an NMI trap is generated if a parity error is detected in the associated SRAM memory module. The assignment of the enable bits to the SRAM modules is defined in Table 5-5. 0_B NMI parity error trap is disabled. 1_B NMI parity error trap is enabled.
0	[31:6]	r	**Reserved** Read as 0; should be written with 0.

Note: SFR SCU_PETCR is Endinit-protected for write operations.

SCU_PETSR
SCU Parity Error Trap Status Registe

(F00000D4$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
PFLx (x = 0-5)	x	rh	**Parity Error Flag for SRAM Module x** These bits indicate whether a parity error has been detected in the associated SRAM memory module. The assignment of the error flags to the SRAM modules is defined in **Table 5-5**. 0_B No parity error detected. 1_B Parity error is detected. The PFLx bits are cleared by hardware after a read access.
0	[31:6]	r	**Reserved** Read as 0.

Note: SCU_PETSR is a read-only register. Writing to SCU_PETSR results in a bus error.

5.6 Pad Driver Temperature Compensation Control

5.6.1 Functional Description

The temperature compensation for the pad output drivers makes it possible to get stable driver output characteristics within dedicated portions of the specified temperature range. As shown in **Figure 5-8**, the temperature compensation requires a 100 KHz reference input clock signal which is derived from the bus clock f_{SYS} by a programmable divider (TCDIV). This reference input clock f_{REF} is fed into a 12-bit free-running counter (SCOUNT). After each overflow of SCOUNT, the pad oscillator circuit (f_{POSC}), which is located in the pad area, and an 8-bit counter (THCOUNT) are enabled for one clock period of the reference clock. The oscillator circuit is temperature-sensitive and typically has a much higher frequency (8 - 16 MHz) than f_{REF}. The 8-bit counter counts the f_{POSC} pulses and its count value THCOUNT when stopped again is compared against three threshold values stored in THMAX, THMED, and THMIN. These thresholds must be set accordingly to the application needs for proper operation of the temperature compensation.

The temperature compensation can be controlled for Class A2 output drivers at GPIO ports (Port 0 to 5).

The clock divider TCDIV is programmed via bit field SCU_TCCON.TCDIV. TCDIV can be calculated using the following formula:

$$TCDIV = RoundDown\ (2.5 \times f_{SYS}) - 1 \quad (5.1)$$

The resulting divide factor is TCDIV + 1 and leads to an update of the temperature compensation approximately every 41 ms.

Example: for f_{SYS} = 80 MHz, TCDIV = RoundDown (2.5 × 80) - 1 = 199_D (= $C7_H$).

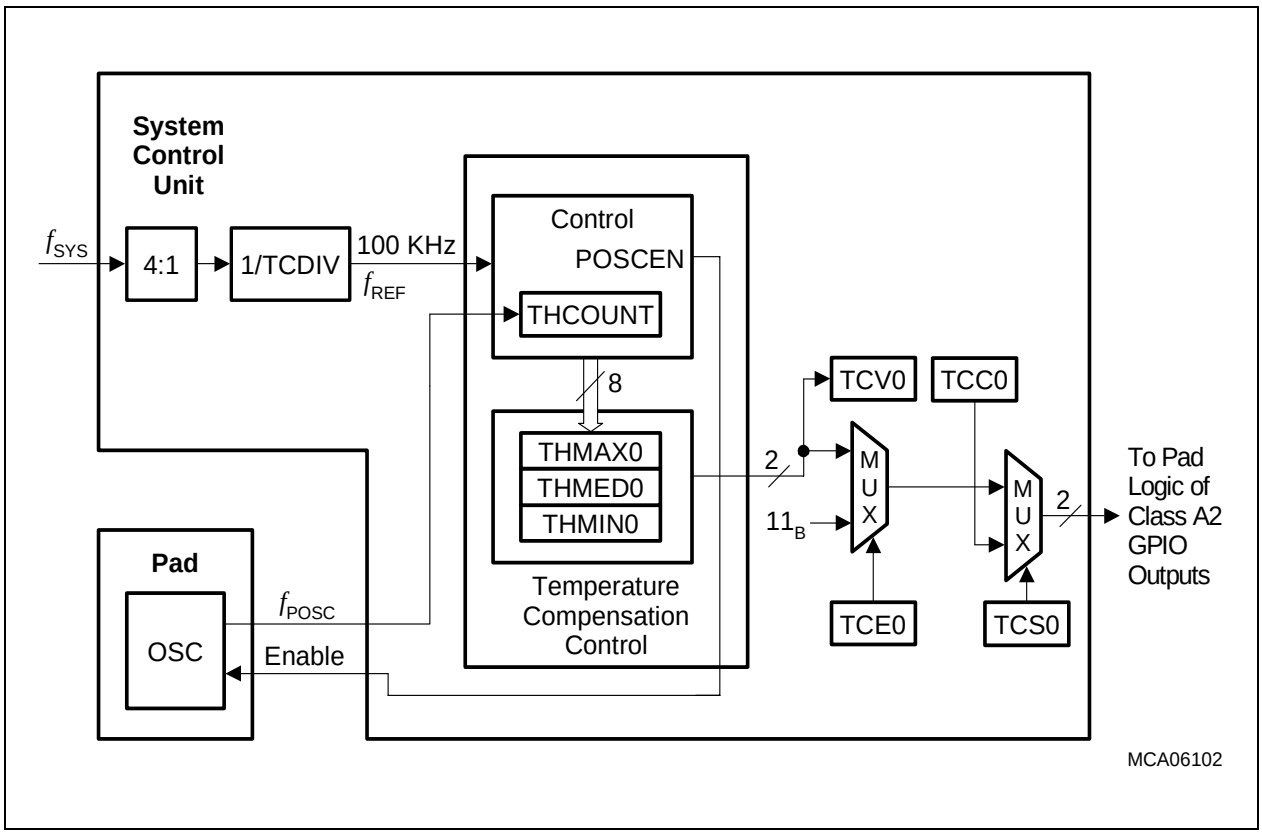


Figure 5-8 Pad Driver Temperature Compensation Block Diagram

Generally, temperature compensation is a transparent feature. Bit field THCOUNT of the port temperature compensation 0 level register SCU_TCLR0 provides direct access to the actual compensation (counter) value and allows software adjustment control of the port temperature compensation logic. This is useful for two situations:

1. **Device testing:** The function of the compensation mechanism can be verified during production testing or characterization.
2. **User control:** During operation the device can be controlled via externally provided compensation values rather than via the internal mechanism.

Temperature compensation is initialized by programming register SCU_TCCON (enable and prescaler for 100 kHz reference clock) and register SCU_TCLR0 (levels values).

The temperature-dependent counter value THCOUNT is compared against three threshold values (correspond to a certain temperature level) at which the 2-bit output signals to the pad logic are switched (see **Figure 5-9**). The three threshold values are defined by three bit fields in register SCU_TCLR0. The pads support different driving levels and are controlled via the 2-bit output signals.

The switching thresholds are evaluated hierarchically (see **Table 5-6**). For proper operation the relationship THMIN > THMED > THMAX must be true. With increasing temperature, the compensation value SCU_TCLRx.THCOUNT decreases.

Table 5-6 Switching Threshold Hierarchy

Output Driver Control	Relationship between THCOUNT and THMAX, THMED, THMIN
Maximum Level	THCOUNT < THMAX
High Level	THMED ≥ THCOUNT > THMAX
Low Level	THMIN ≥ THCOUNT > THMED
Minimum Level	THCOUNT > THMIN

Note: The reset value FF_H for the thresholds ensures that the drivers operate on maximum level if the threshold values have not been initialized.

Note: Reprogramming of the threshold levels in register SCU_TCLR0 should only be executed while the temperature compensation is disabled.

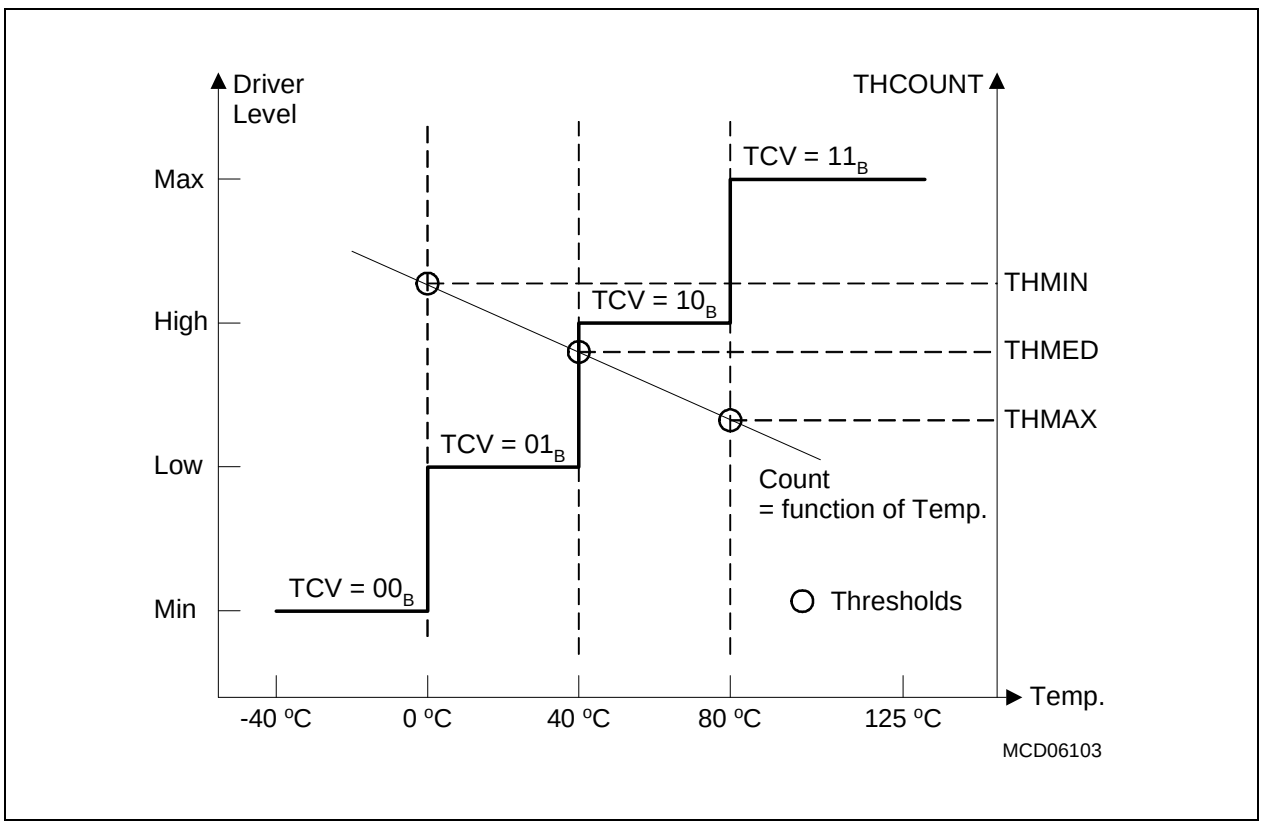
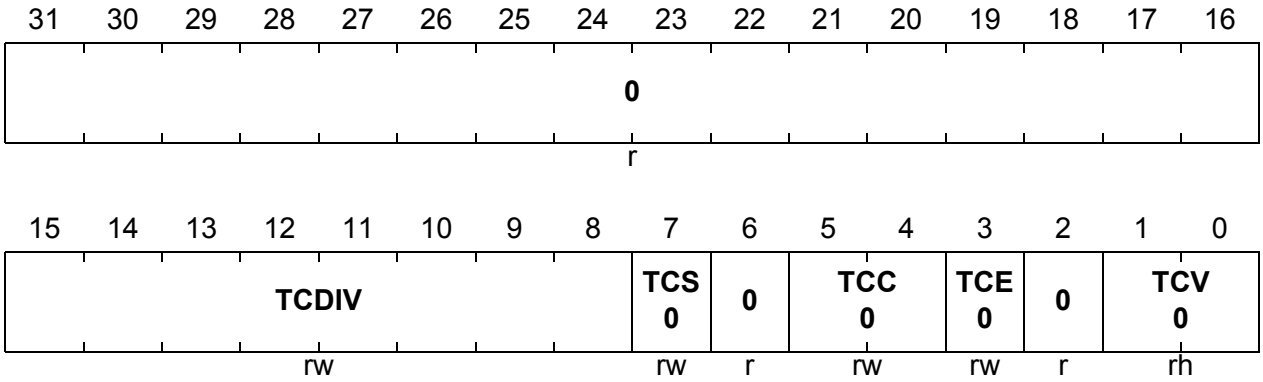


Figure 5-9 Temperature Compensation Switching Thresholds

5.6.2 Temperature Compensation Registers

SCU_TCCON
SCU Temperature Compensation Control Register
(F0000048$_H$) **Reset Value: 0000 0003$_H$**

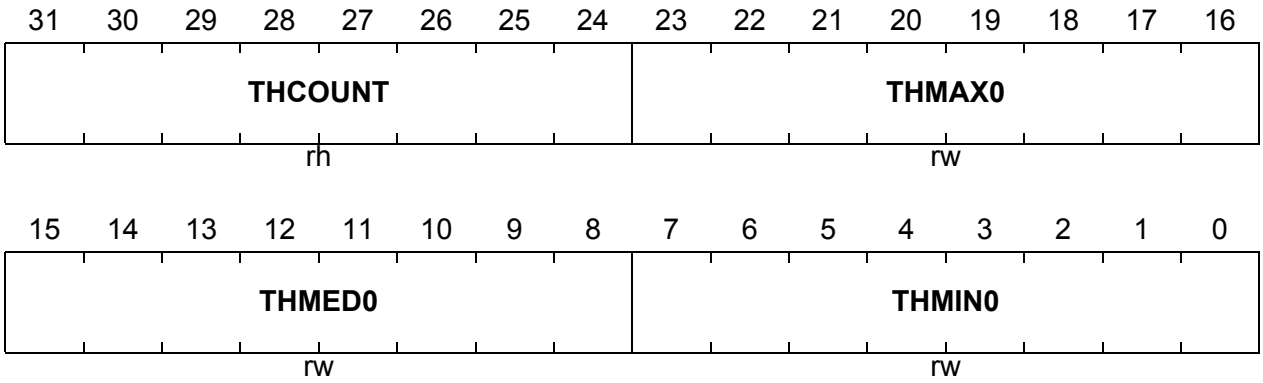


Field	Bits	Type	Description
TCV0	[1:0]	rh	**Temperature Compensation 0 Value** This bit field indicates the compensation value that is generated for the temperature compensation logic 0. TCV0 is fed to the temperature compensation logic 0 outputs while bit TCS0 = 0. 00_B Minimum driving strength required, i.e. very low temperature. ... ... 11_B Maximum driving strength required, i.e. very high temperature (default after reset).
TCE0	3	rw	**Temperature Compensation 0 Enable** 0_B Temperature compensation logic 0 is deactivated (default after reset). The port drivers are at maximum driver level. 1_B Temperature compensation logic 0 is active. THCOUNT is periodically updated.
TCC0	[5:4]	rw	**Temperature Compensation 0 Control** This 2-bit value is fed to the temperature compensation logic 0 outputs while bit TCS0 = 1. Encoding of TCC0 is equal to TCV0.

Field	Bits	Type	Description
TCS0	7	rw	**Temperature Compensation 0 Source** 0_B Temperature compensation logic 0 is controlled by the temperature compensation control hardware. 1_B Temperature compensation logic 0 is controlled by software via bit field TCC0.
TCDIV	[15:8]	rw	**Temperature Compensation Clock Divider** This value controls the input clock divider for the temperature compensation logic.
0	2, 6, [31:16]	r	**Reserved** Read as 0; should be written with 0.

SCU_TCLR0
SCU Temperature Compensation 0 Level Register

(F0000058$_H$) **Reset Value: 00FF FFFF$_H$**



Field	Bits	Type	Description
THMIN0	[7:0]	rw	**Minimum Threshold for Temp. Compensation 0** Driver level = Low or Min
THMED0	[15:8]	rw	**Medium Threshold for Temp. Compensation 0** Driver level = High or Low
THMAX0	[23:16]	rw	**Maximum Threshold for Temp. Compensation 0** Driver level = Max or High
THCOUNT	[31:24]	rh	**Threshold Counter** Returns the actual count value of the counter that counts the f_{POSC} clock pulses. THCOUNT is only updated when temperature compensation is enabled (TCE0 is set).

5.7 Die Temperature Sensor

The TC1766 provides an on-chip Die Temperature Sensor (DTS). The output voltage of the DTS can be measured using analog input AIN31 of the ADC0 module. For measuring the DTS output voltage, the following conditions must be met:

- The DTS circuitry has to be enabled by setting SCU_CON.DTSON = 1. After enabling the DTS, a settling time of about 10 µs has to be respected before starting a DTS measurement.
- Bit ADC0_CHCONx.EMUX[0] has to be set to 1 for AD0EMUX0 signal.
- The analog input AIN31 of ADC0 has to be requested for conversion (corresponding GRPS = 1). With this request, the reference voltages of the DTS (va_ref_aio, va_gnd_aio) and the sensor output signal (temp_ao) are connected to ADC0 for AD conversion.

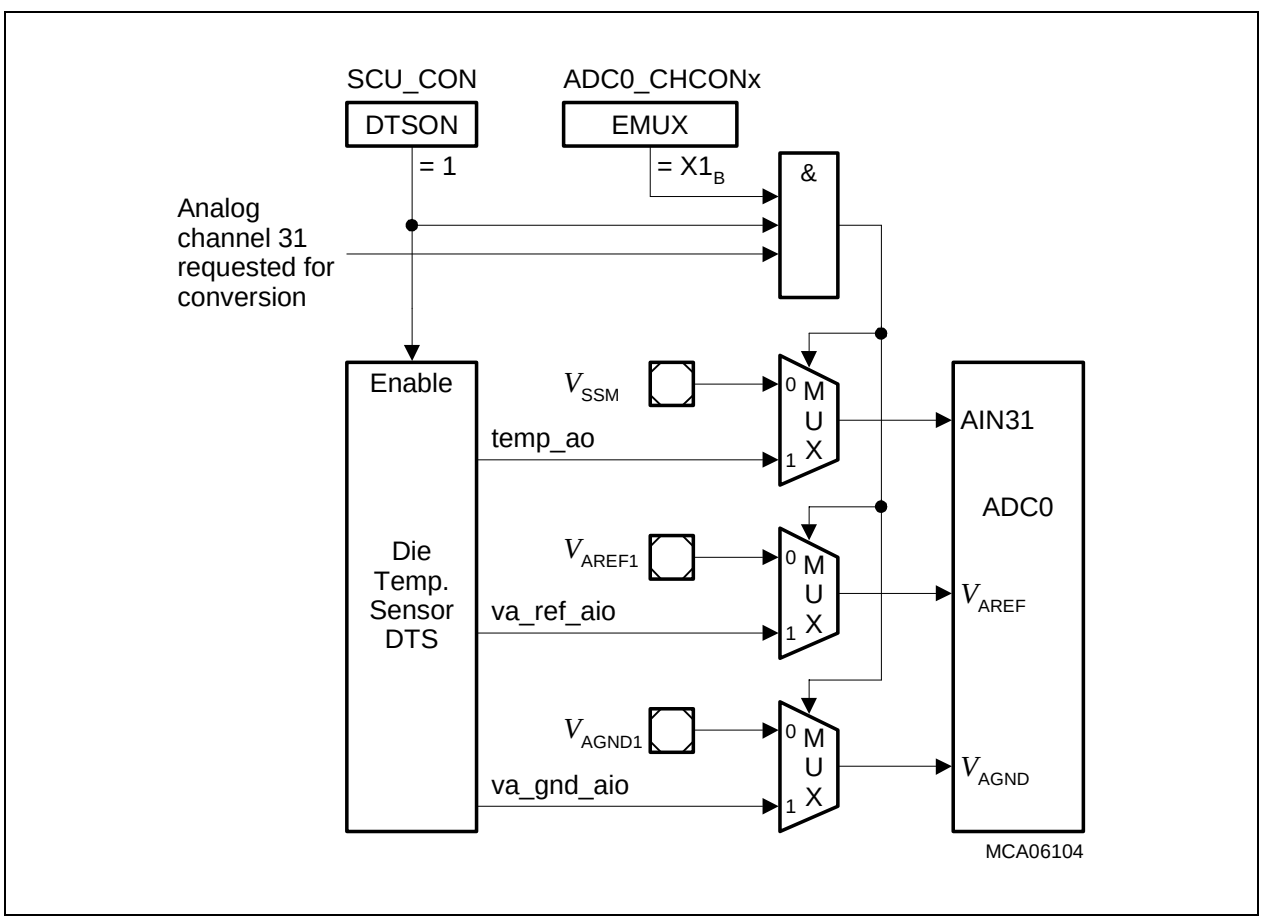


Figure 5-10 Die Temperature Sensor Selection

The die temperature can be determined according to the equation:

temp_ao[V] = 1.0 V + (die temperature + 40 °C) / 190 (5.2)

The DTS operates with an accuracy of ±10 °C.

5.8 DMA Request Signal Selection

The DMA controller supports a fixed number of input requests. For TC1766, the DMA controller has 8 input channels; each channel can be connected to a maximum of 8 requests signals.

In order to have more request signals connected to the DMA controller, additional multiplexing logic is introduced in the SCU. It consists of a register with bit fields to control the multiplexer. The basic scheme is shown in **Figure 5-11**.

Note: By default, TIR line is selected for both ASC and SSC modules. For RIR line to be selected, SEL0.DMARS to SEL3.DMARS should be set accordingly. See DMARS register.

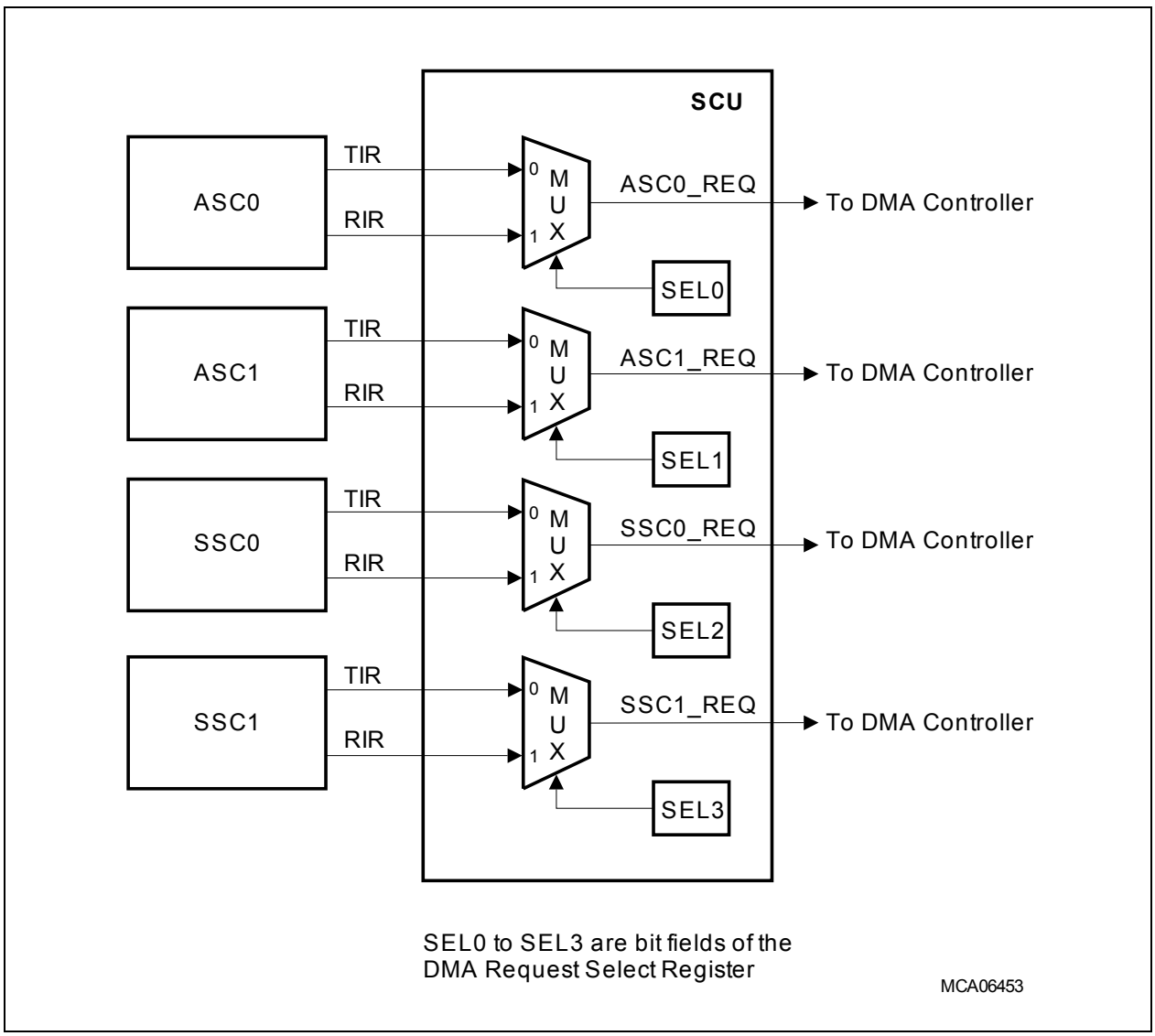
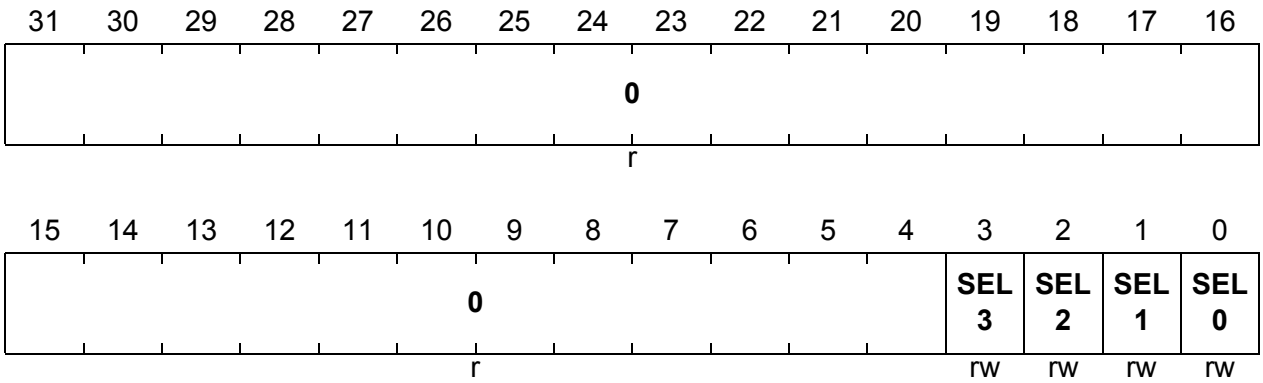


Figure 5-11 DMA Request Selection Logic

5.8.1 DMA Request Select Register

SCU_DMARS
SCU DMA Request Select Register

(F00000D8$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
SEL0	0	rw	**Request Select Bit 0** This bit is used to select the DMA request signals from ASC0. 0_B ASC0_TIR is selected. 1_B ASC0_RIR is selected.
SEL1	1	rw	**Request Select Bit 1** This bit is used to select the DMA request signals from ASC1. 0_B ASC1_TIR is selected. 1_B ASC1_RIR is selected.
SEL2	2	rw	**Request Select Bit 2** This bit is used to select the DMA request signals from SSC0. 0_B SSC0_TIR is selected. 1_B SSC0_RIR is selected.
SEL3	3	rw	**Request Select Bit 3** This bit is used to select the DMA request signals from SSC1. 0_B SSC1_TIR is selected. 1_B SSC1_RIR is selected.
0	[31:4]	r	**Reserved** Read as 0.

5.9 GPTA0 Input IN1 Control

In the TC1766, the input line IN1 of the GPTA0 module can be used to measure the baud rate of an ASC0 or ASC1 receiver input signal with GPTA0. This feature is controlled by a bit of the SCU, SCU_CON.GIN1S. Bit GIN1S controls a 4-to-1 multiplexer that makes it possible to switch several port lines to the IN1 input of the GPTA0 module. After reset, the nominal GPTA input IN1 (P0.1) is connected with IN1 of GPTA.

Table 5-7 GPTA0 Input Line IN1 Connections

SCU_CON.GIN1S	GPTA0 Input IN1 Connected to
00_B	P0.1 / IN1 (default after reset)
01_B	P3.0 / RXD0A
10_B	P3.12 / RXD0B
11_B	P3.14 / RXD1B

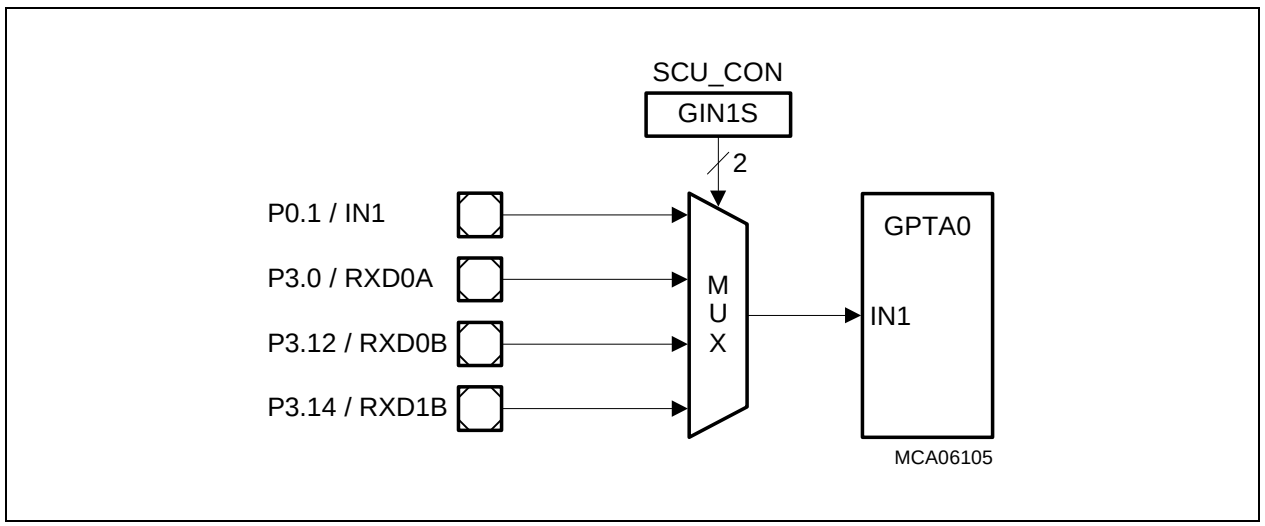


Figure 5-12 GPTA0 Input IN1 Control

5.10 Pad Test Mode Control

The pad test mode control logic in the SCU can be used for in-system tests of board connections for dedicated pins (pins without GPIO functionality). The pad test mode can be enabled in the normal operating mode of the TC1766. A special enable procedure (two-word write sequence) avoids unintentional enabling of the pad test mode.

The pad test mode control logic especially makes it possible to:

- output a value (low or high level) to dedicated pins
- read the logic level on dedicated pins

Figure 5-13 shows the pad test mode control logic for one pad/pin. Test data is read or written via the pad test data register PTDAT0.

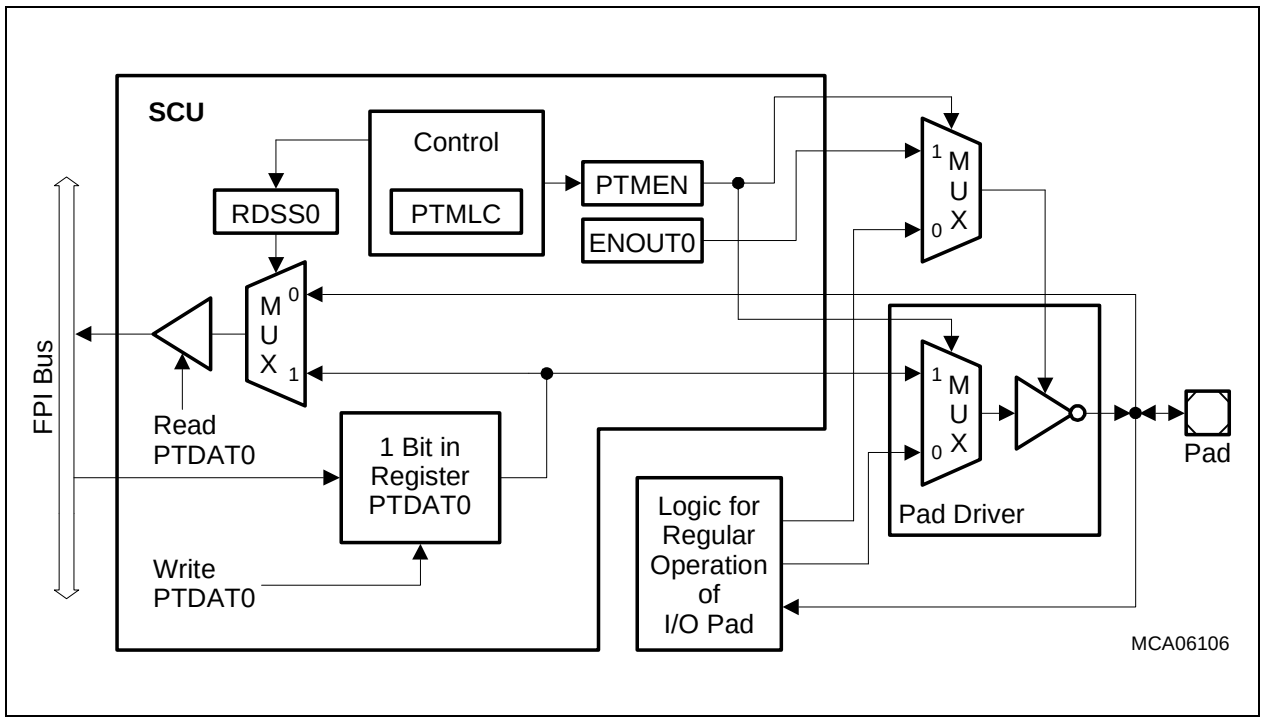


Figure 5-13 Pad Test Mode Control in the SCU

When the pad test mode is disabled (PTMEN = 0), the dedicated pads are in their normal operating mode. The pad output drivers are controlled by the hardware part that determines whether the dedicated pad/pin is used for input, output for I/O purposes.

When the pad test mode is enabled, PTMEM is set. A value written into register PTDAT0 is always output at the corresponding pad as inverted state. This means when writing a 1 (0) to a bit of register PTDAT0, a low (high) level will be available at the corresponding dedicated pin. When reading register PTDAT0, either the (non-inverted) logic level at the dedicated pad (RDSS0 = 0) or the value of the PTDAT0 register bit (RDSS0 = 1) can be read back. The ENOUT0 bits determine whether the logic level state as defined by the bits in the SCU_PTDAT0 register is output to the pad/pin or not.

5.10.1 Pad Test Mode Enabling

To enable the pad test mode, the following two-word write sequence must be executed:

1. Writing SCU_PTCON with lock code PTMLC = $5A_H$
2. Writing SCU_PTCON with lock code PTMLC = $A5_H$. After this write operation pad test mode is enabled, indicated by PTMEN = 1. Bits ENOUT0 and RDSS0 determine the requested pad test mode configuration (enable, input selection).

When pad test mode is enabled, the test mode configuration (as defined through ENOUT0 and RDSS0) can be changed without leaving the pad test mode by writing SCU_PTCON with new values for bits ENOUT0 and RDSS0 and PTMLC = $A5_H$. In pad test mode, any other write operation to SCU_PTCON with lock code PTMLC not equal to $A5_H$ terminates the pad test mode.

After pad test mode has been enabled via the two-word write operation to SCU_PTCON, it can be disabled again by any reset operation or a write operation to SCU_PTCON.

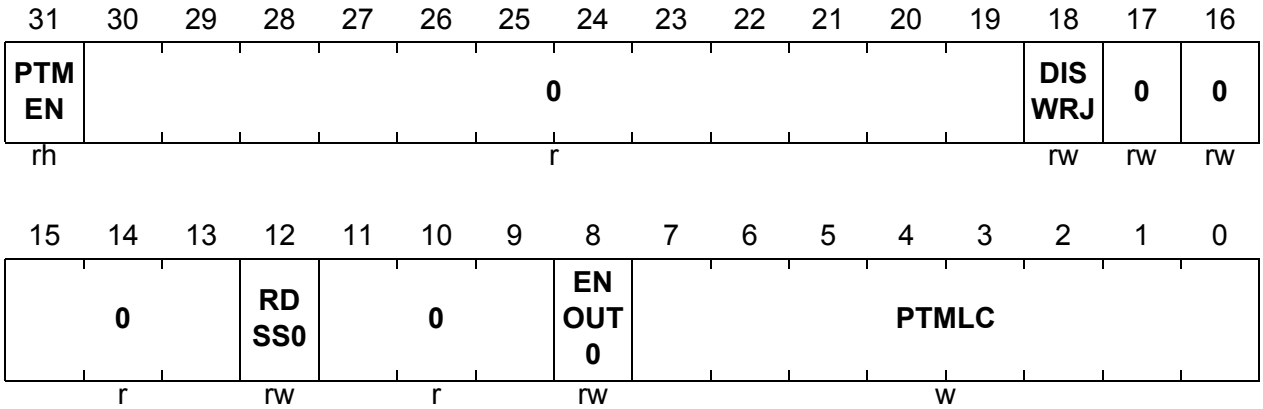
5.10.2 Pad Test Mode Registers

The pad test mode control logic contains two registers.

Table 5-8 Pad Test Mode Registers

Register Short Name	Register Long Name	Description see
SCU_PTCON	SCU Pad Test Control Register	Page 5-54
SCU_PTDAT0	SCU Pad Test Data Register 0	Page 5-56

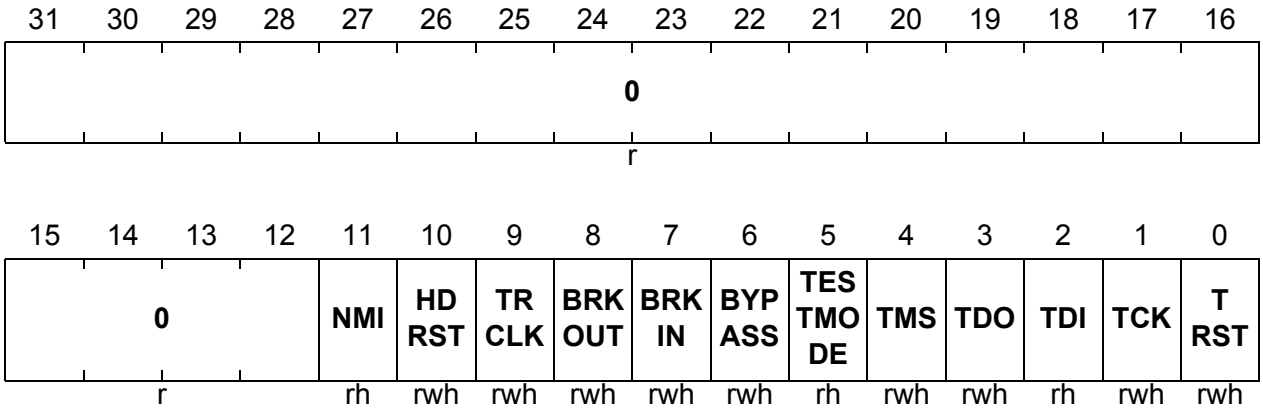
SCU_PTCON
SCU Pad Test Control Register (F00000B0$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
PTMLC	[7:0]	w	**Pad Test Mode Lock Code** This bit field must be written with a special two-word write sequence to enable the pad test mode. PTMLC is always read as 00$_H$.
ENOUT0	8	rw	**Enable Data Output for Pad Test Data Register 0** In pad test mode (PTMEN = 1), these bits determine whether or not the logic level state as defined by the bits in the SCU_PTDAT0 register is switched as inverted state to the pad/pin. 0_B Pad drivers of the pad test data register n related pads are disabled. 1_B Pad drivers of the pad test data register n related pads are enabled and the bits in the pad test data register n are output as inverted state at the related pins.
RDSS0	12	rw	**Read Source Selection for Pad Test Data Register 0** In pad test mode (PTMEN = 1), these bits determine the read source for the SCU_PTDAT0 register. 0_B Logic levels of the pad test data register 0 related pads/pins are read. 1_B Pad test data register n bits are read.
0	16	rw	**Reserved** Read as 0 after reset; returns the value that is written.

Field	Bits	Type	Description
0	17	rw	**Reserved** Read as 0; Bits must be written with 0.
DISWRJ	18	rw	**Disable Write to JTAG IO Lines** 0_B JTAG IO lines can be written through SCU_PTDAT0[4:0]. 1_B JTAG IO lines cannot be written through SCU_PTDAT0[4:0] when SCU_PTCON.ENOUT0 and SCU_PTCON.PTMEN are set to 1. *Note: With DISWRJ = 1 the JTAG output line TDO cannot be used (in the TC1766) for JTAG purposes.*
PTMEN	31	rh	**Pad Test Mode Enable Flag** 0_B Pad test mode disabled (default after reset) 1_B Pad test mode enabled
0	[11:9], [15:13], [30:19]	r	**Reserved** Read as 0; should be written with 0.

SCU_PTDAT0
SCU Pad Test Data Register 0 (F00000B4$_H$) Reset Value: XXXX XXXX$_H$



Field	Bits	Type	Description
TRST	0	rwh	**Pad Test Value for/of $\overline{\text{TRST}}$**
TCK	1	rwh	**Pad Test Value for/of TCK**
TDI	2	rh	**Pad Test Value for/of TDI**
TDO	3	rwh	**Pad Test Value for/of TDO**
TMS	4	rwh	**Pad Test Value for/of TMS**
TESTMODE	5	rh	**Pad Test Value of $\overline{\text{TESTMODE}}$**
BYPASS	6	rwh	**Pad Test Value for/of BYPASS**
BRKIN	7	rwh	**Pad Test Value for/of $\overline{\text{BRKIN}}$**
BRKOUT	8	rwh	**Pad Test Value for/of $\overline{\text{BRKOUT}}$**
TRCLK	9	rwh	**Pad Test Value for/of TRCLK**
HDRST	10	rwh	**Pad Test Value for/of $\overline{\text{HDRST}}$**
NMI	11	rh	**Pad Test Value for/of $\overline{\text{NMI}}$**
0	[31:12]	r	**Reserved** Read as 0; should be written with 0.

Note: In pad test mode, the bits in SCU_PTDAT0 are output to the pad/pin in inverted state: a 0 generates a high level and a 1 generates a low level at the pad/pin.

5.11 Emergency Stop Output Control for GPTA and MSC

The emergency stop feature of the TC1766 makes it possible for a fast emergency reaction on an external event for the GPTA and MSC modules without the intervention of software. In an emergency case, the GPTA outputs of the TC1766 can be selectively put immediately to a well-defined logic level. The MSC0 module is also able to handle an emergency case by selectively putting bits of the following serial downstream frame(s) to a programmable logic value instead of an actual data value.

The emergency case is indicated to the TC1766 by an emergency input signal with selectable polarity that has to be connected to input HWCFG1 (P1.4).

Figure 5-14 shows a diagram of the emergency stop input logic. This logic is controlled by the SCU Emergency Stop Register SCU_EMSR.

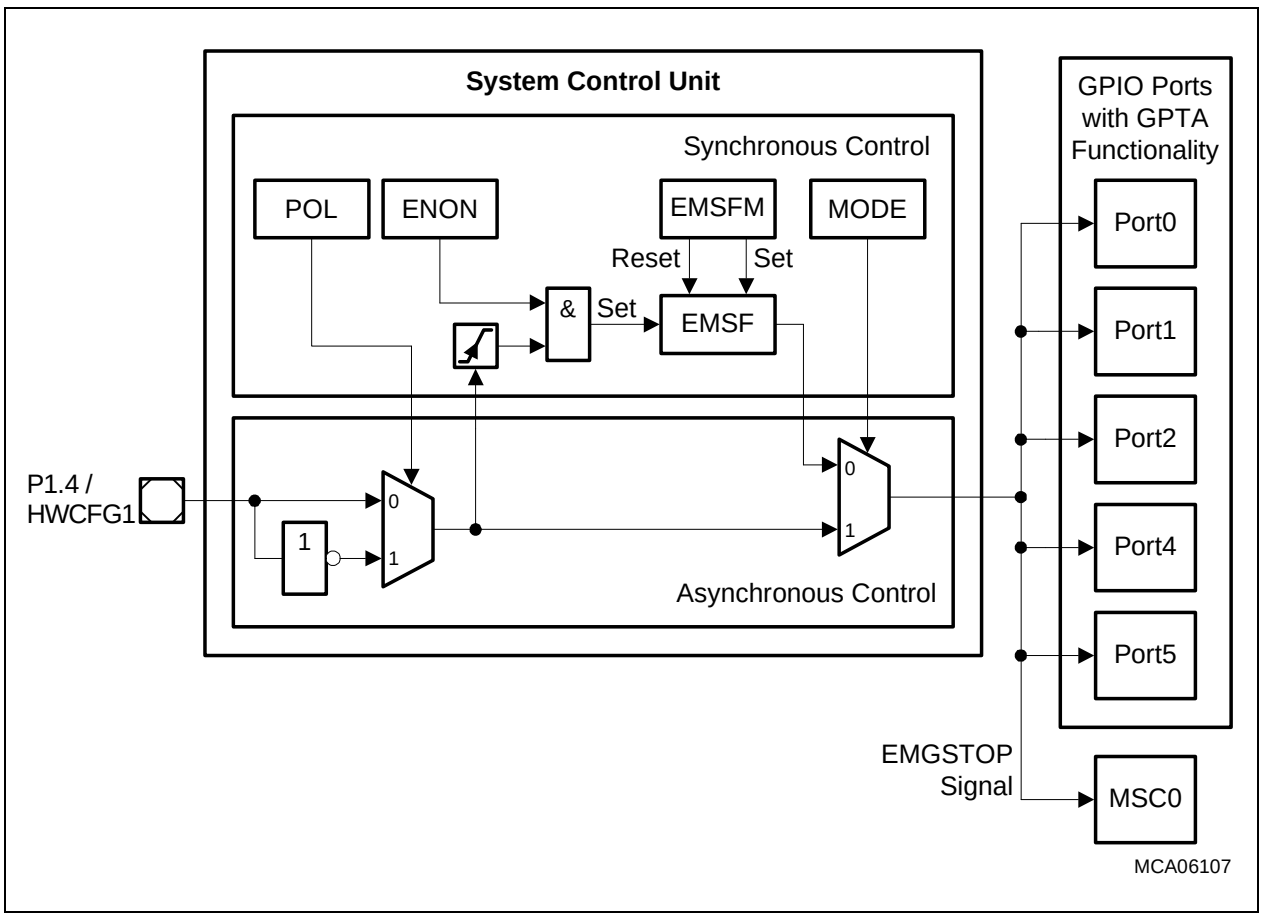


Figure 5-14 Emergency Stop Input Control

The emergency stop control logic 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 MODE = 0), the HWCFG1 signal is sampled for a inactive-to-active level transition, and an emergency stop flag EMSF is set if the transition is detected. The setting of EMSF activates the EMGSTOP signal. An emergency case can only be terminated by clearing EMSF via software. The synchronous control logic is clocked by the system clock f_{SYS}. This results in a small delay between the HWCFG1 signal and EMGSTOP signal generation. If the system clock is switched off (not applicable in TC1766), the synchronous mode control logic is frozen and not able to react to transitions at input HWCFG1. In this case, the asynchronous mode can be used to put GPTA outputs to dedicated logic levels.

In asynchronous mode (selected by MODE = 1), the occurrence of an active level at input HWCFG1 immediately activates the EMGSTOP signal to the GPTA outputs and the MSC modules even if the synchronous control part is inactive and not clocked by the system clock f_{SYS}. A valid-to-invalid transition of HWCFG1 (emergency case is released) also immediately deactivates the EMGSTOP signal.

The POL bit determines the active level of the emergency stop input signal HWCFG1. The MODE bit selects synchronous or asynchronous mode for emergency stop signal generation. The EMSF flag can be enabled/disabled for setting (control bit ENON) and it can be set or cleared by software too (bit field EMSFM).

5.11.1 GPTA Output Emergency Control in the GPIO Ports

The selection of which port line of a GPIO port is affected by an active EMGSTOP signal is done in the Emergency Stop Registers (Pn_ESR), which are located in the port logics. Each of the GPIO lines that can be assigned as GPTA output has its own emergency stop enable bit Pn_ESR.ENx. If the emergency stop signal EMGSTOP becomes active, one of two states can be selected:

- Emergency stop function disabled (Pn_ESR.ENx = 0):
  A GPTA output line remains connected to the GPTA module (alternate function).
- Emergency stop function enabled (Pn_ESR.ENx = 1):
  A GPTA output line is disconnected immediately from the GPTA module (alternate function) and connected to the corresponding bit of the Pn_OUT output register Pn_OUT.Px.

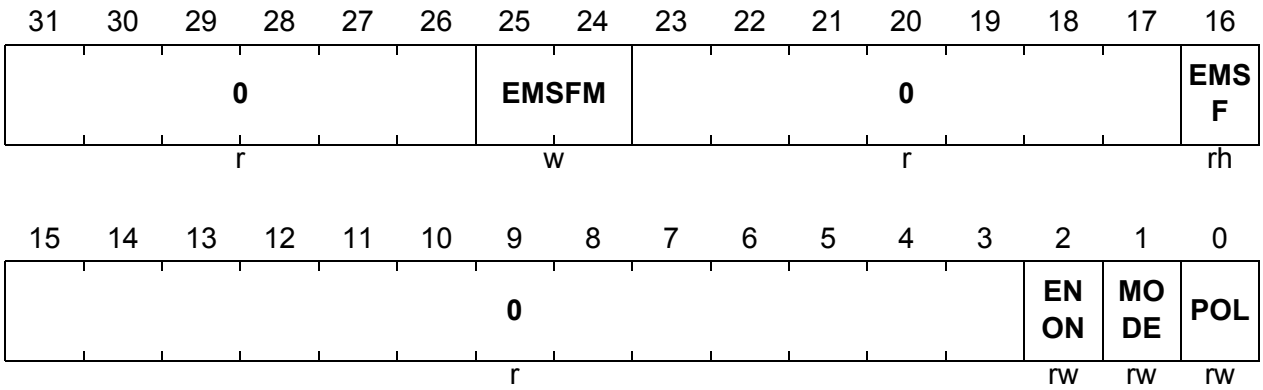
5.11.2 MSC Emergency Control Selection

With an active emergency stop signal (EMGSTOP = 1), the MSC0 module puts bits of the following serial downstream frame(s) to a programmable logic value instead of an actual data value. In an emergency case, the bits in the downstream shift registers SRL/SRH that are enabled for the emergency stop feature (corresponding bits in the emergency stop register ESR set) are loaded with the corresponding bit of the downstream data register DD instead with the bits from the ALTINL/ALTINH data source. Therefore, it takes a calculable time until the transmitted downstream frame with the emergency stop information reaches the receiver device.

5.11.3 Emergency Stop Register

The Emergency Stop Register SCU_EMSR contains control and status bits/flags of the emergency stop input logic.

SCU_EMSR
SCU Emergency Stop Register (F0000044$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
POL	0	rw	**Input Polarity** This bit determines the polarity of the emergency input line HWCFG1. 0_B HWCFG1 is high active. 1_B HWCFG1 is low active.
MODE	1	rw	**Mode Selection** This bit determines the operating mode of the EMGSTOP signal. 0_B Synchronous mode selected; EMGSTOP is derived from the state of flag EMSF. 1_B Asynchronous mode selected; EMGSTOP is directly derived from the state of the input signal HWCFG1.
ENON	2	rw	**Enable ON** This bit enables the (hardware) setting of flag EMSF by an inactive-to-active level transition of input signal HWCFG1. 0_B Setting of EMSF by hardware is disabled. 1_B Setting of EMSF by hardware 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 EMGSTOP becomes active (if MODE = 0).
EMSFM	[25:24]	w	**Emergency Stop Flag Modification** This bit field makes it possible to set or clear flag EMSF by software. In case of a simultaneous hardware and software modification request, the hardware operation will be executed. 00_B EMSF remains unchanged. 01_B EMSF is set. 10_B EMSF is 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.

5.12 Analog Input 7 Testmode

When setting bit SCU_CON.AN7TM, the analog input AN7 is pulled low via a resistor of max. 900 Ω. This feature makes it possible to check the correct operation of the ADC during operation. The conversion result should be near the minimum (depending on the source resistance of the connected sensor).

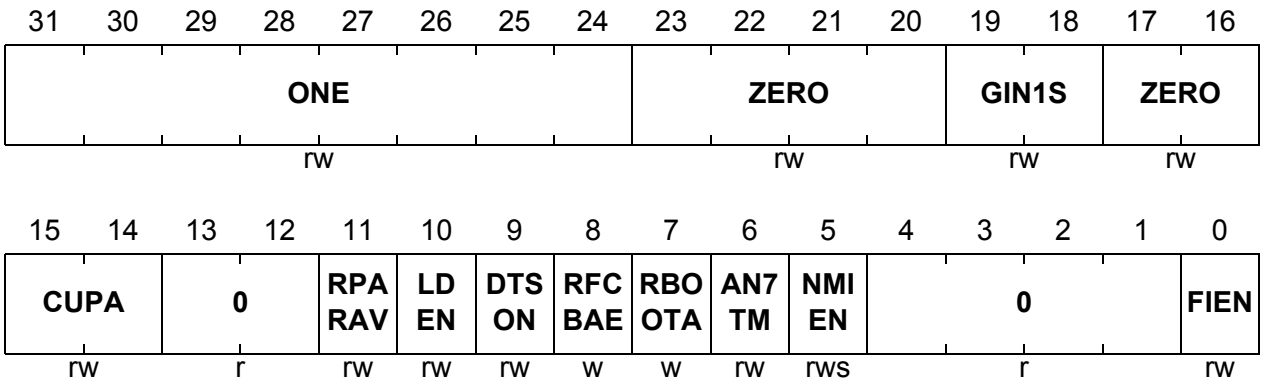
5.13 Miscellaneous SCU Registers

This section includes descriptions of the following registers:

- SCU Control Register SCU_CON
- SCU Status Register SCU_STAT
- SCU Module Identification Register SCU_ID
- Manufacturer Identification Register MANID
- Chip Identification Register CHIPID
- Redesign Tracing Identification Register RTID

5.13.1 SCU Control Register

SCU_CON
SCU Control Register **($F0000050_H$)** **Reset Value: $FF00\ 0000_H$**



Field	Bits	Type	Description
FIEN	0	rw	**FPU Inexact Interrupt Enable** 0_B Inexact error condition (setting FX flag) after a FPU calculation will not generate an interrupt. 1_B Inexact error condition (setting FX flag) after a FPU calculation will generate an interrupt.
NMIEN	5	rws	**NMI Enable** 0_B NMI is disabled (default). 1_B NMI is enabled. This bit is cleared with any reset. It can be only set by software and will remain in this state until the next reset. Writing a zero to this bit has no effect. The NMI is described in detail in **Section 12.10** on **Page 12-25**.

Field	Bits	Type	Description
AN7TM	6	rw	**Analog Input 7 Test Mode** 0_B Pull down of analog input 7 is disabled (default after reset). 1_B Pull down of analog input 7 is enabled.
RBOOTA	7	w	**Reset Boot Active** 0_B No action. 1_B Clear bit SCU_STAT.BOOTA. This bit will always be read as 0.
RFCBAE	8	w	**Reset Flash Config. Sector Access Enable Bit** 0_B No action. 1_B Clear bit SCU_STAT.FCBAE. This bit will always be read as 0.
DTSON	9	rw	**Die Temperature Sensor On** 0_B Die temperature sensor is switched off. 1_B Die temperature sensor is switched on.
LDEN	10	rw	**LVDS Driver Enable** 0_B The LVDS drivers are disabled and in power-down mode. 1_B The LVDS drivers are enabled. See also **Page 9-68**.
RPARAV	11	rw	**Reset Parity Available Bit** 0_B No action. 1_B Clear bit SCU_STAT.PARAV. This bit will always be read as 0. See also **“SRAM Parity Control” on Page 5-38**.

Field	Bits	Type	Description
CUPA	[15:14]	rw	**Control for USB Pin Assignment** This bit field controls the USB pin assignment for TC1766ED. 00_B No external USB interface is available. Pin assignment as in normal production chip is retained. (default) 01_B USB interface is assigned to JTAG/OCDSL1 port. 10_B USB interface is assigned to Port 2.[5:0]. 11_B No external USB interface is available. Pin assignment as in normal production chip is retained. A write operation to this bit field causes a hardware change to an internal shadow register; the value of which selects the desired assignment.
ZERO	[23:20], [17:16]	rw	**Spare 0 Control Bits** This bit field contains bits that are reserved for future SCU control tasks. Field ZERO is set to 00_H after reset. ZERO bits should be written with 00_H. Reading ZERO bits will return the value last written.
GIN1S	[19:18]	rw	**GPTA Input1 Source Select** 00_B IN1 of GPTA input array 01_B RxD0A 10_B RxD0B 11_B RxD1B Details of GIN1S control are described in Section 5.9 on Page 5-51.
ONE	[31:24]	rw	**Spare 1 Control Bits** This bit field contains bits that are reserved for future SCU control tasks. Field ONE is set to FF_H after reset. ONE bits should be written with FF_H. Reading ONE bits will return the value last written.
0	[4:1], [13:12]	r	**Reserved** Read as 0; should be written with 0.

5.13.2 SCU Status Register

The SCU Status Register SCU_STAT holds the status bits for the FPU interrupt (see **Section 5.4.2** on **Page 5-36**).

The Boot ROM code is always executed after every reset operation. Depending on the program flow through the Boot ROM code and the Boot ROM exit path, several resources of the TC1766 on-chip hardware have been used and are programmed. This means that the state of on-chip hardware resources that have been used by the Boot ROM code may differ from the device reset state as described by the register reset values. This section describes which initial state of the on-chip hardware resources is left after a specific Boot ROM exit.

SCU_STAT
SCU Status Register **($F0000054_H$)** **Reset Value: $0000\ D000_H$ 1)**

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
BOOTA	FCBAE	DTSRDY	PARAV	0			EEA	0			FII	FVI	FZI	FUI	FXI
rh	rh	rh	rh	r			rh	r			rh	rh	rh	rh	rh

1) The initial value of SCU_STAT after Boot ROM exit is $0000\ 1000_B$.

Field	Bits	Type	Description
FXI	0	rh	**FPU Inexact Result Indication Flag** Indicates the state of the FPU's FX status flag latched during the last FPU interrupt.
FUI	1	rh	**FPU Underflow Error Indication Flag** Indicates the state of the FPU's FU status flag latched during the last FPU interrupt.
FZI	2	rh	**FPU Divide by Zero Error Indication Flag** Indicates the state of the FPU's FZ status flag latched during the last FPU interrupt.
FVI	3	rh	**FPU Overflow Error Indication Flag** Indicates the state of the FPU's FV status flag latched during the last FPU interrupt.

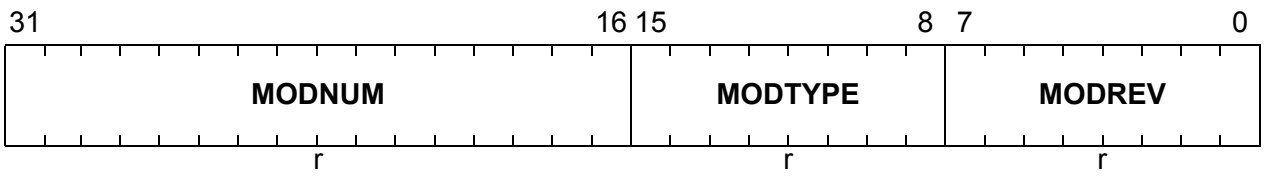
Field	Bits	Type	Description
FII	4	rh	**FPU Invalid Operation Error Indication Flag** Indicates the state of the FPU's FI status flag latched during the last FPU interrupt.
EEA	8	rh	**Emulation Extension Available** Indicates if the emulation extension (= 1766ED) is available or not (= production device). 0_B EEC is not available. 1_B EEC is available.
PARAV	12	rh	**Parity Available** 0_B Parity logic is not available in TC1766. 1_B Parity logic is available in TC1766. This bit is set after a power-on reset. It can be cleared by software by writing bit SCU_CON.RPAAV. It cannot be set by software. For further details, see text below **Table 5-5**.
DTSRDY	13	rh	**Die Temperature Sensor Ready** 0_B Die temperature sensor is not ready. 1_B Die temperature sensor is ready. A stable value can be read.
FCBAE	14	rh	**Flash Config. Sector Access Enable** 0_B Flash configuration sector is not accessible. 1_B Flash configuration sector is accessible. This bit is set with any reset. It can be cleared by software by writing bit SCU_CON.RFCBAE. It cannot be set by software. It will be zero after the boot software has been executed.
BOOTA	15	rh	**Boot Active** 0_B TC1766 is not in boot mode. 1_B TC1766 is in boot mode (default after reset). This bit is set with any reset. It can be cleared by software by writing bit SCU_CON.RBOOTA. It cannot be set by software. It will be zero after the boot software has been executed.
0	[7:5], [11:9], [31:16]	r	**Reserved** Read as 0.

5.13.3 Device Identification Registers

The SCU Module Identification Register ID contains read-only information about the SCU module version.

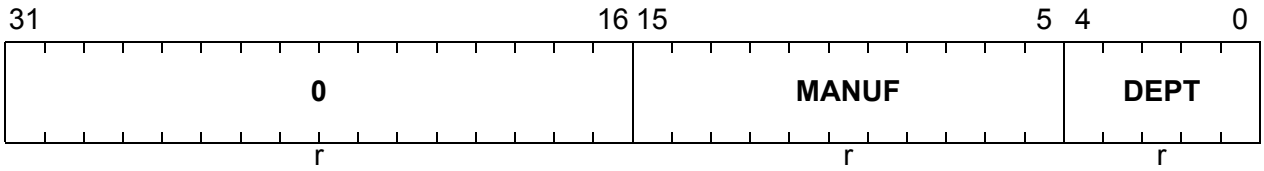
SCU_ID
SCU Module Identification Register

(F0000008$_H$) **Reset Value: 002C C0XX$_H$**



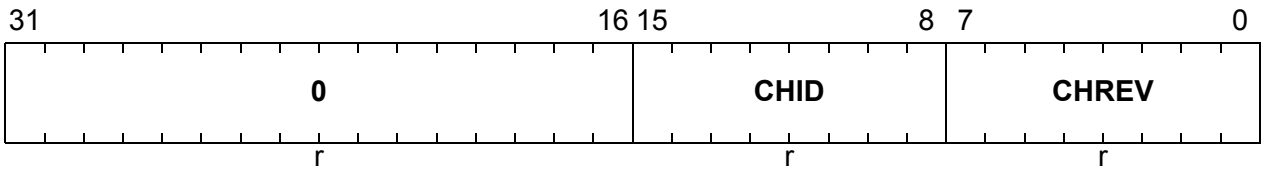
Field	Bits	Type	Description
MODREV	[7:0]	r	**Module Revision Number** MODREV defines the module revision number. The value of a module revision starts with 01_H (first revision).
MODTYPE	[15:8]	r	**Module Type** This bit field defines the module as a 32-bit module: $C0_H$
MODNUM	[31:16]	r	**Module Number Value** This bit field defines the module identification number for the SCU: $002C_H$

MANID
Manufacturer Identification Register(F000 0070$_H$) **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.

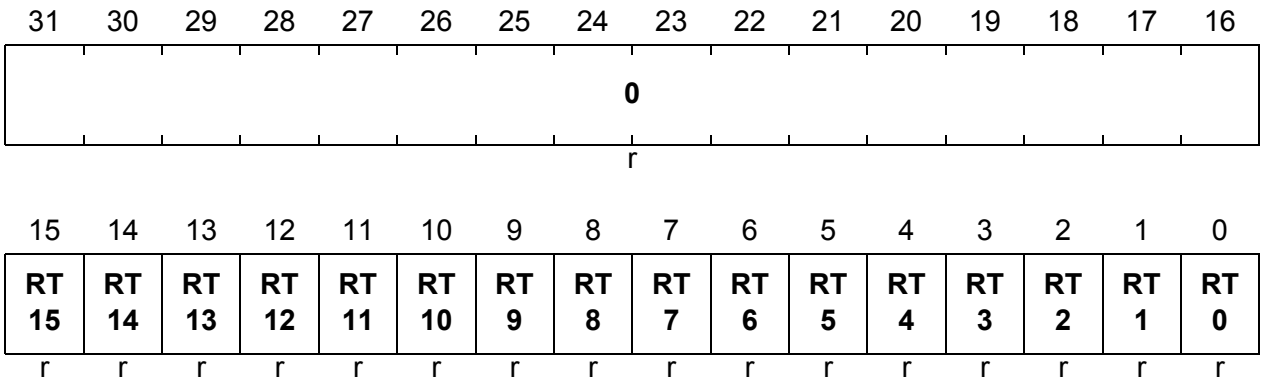
CHIPID
Chip Identification Register **(F000 0074$_H$)** **Reset Value: 0000 8B02$_H$**



Field	Bits	Type	Description
CHREV	[7:0]	r	**Chip Revision Number** This bit field indicates the revision number of the TC1766 device (01_H = first revision). For example, CHREV can be used for major step identification purposes. The value of this bit field is defined in the TC1766 Data Sheet.
CHID	[15:8]	r	**Chip Identification Number** $8B_H$ = TC1766
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 (F000 0078$_H$) Reset Value: 0000 XXXX$_H$



Field	Bits	Type	Description
RTn **(n = 0-15)**	n	r	**Redesign Trace Bit n** 0_B No change indicated. 1_B A change has been made (without changing bit field CHIPID.CHREV). For example, RTn can be used 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$_H$.

5.14 SCU Registers and Address Map

This section refers to all registers which are located in the SCU address range. Some of these registers are described in other chapters of this User's Manual. The entries in column entitled "Description see" of **Table 5-9** point to the pages on which such registers are described in detail.

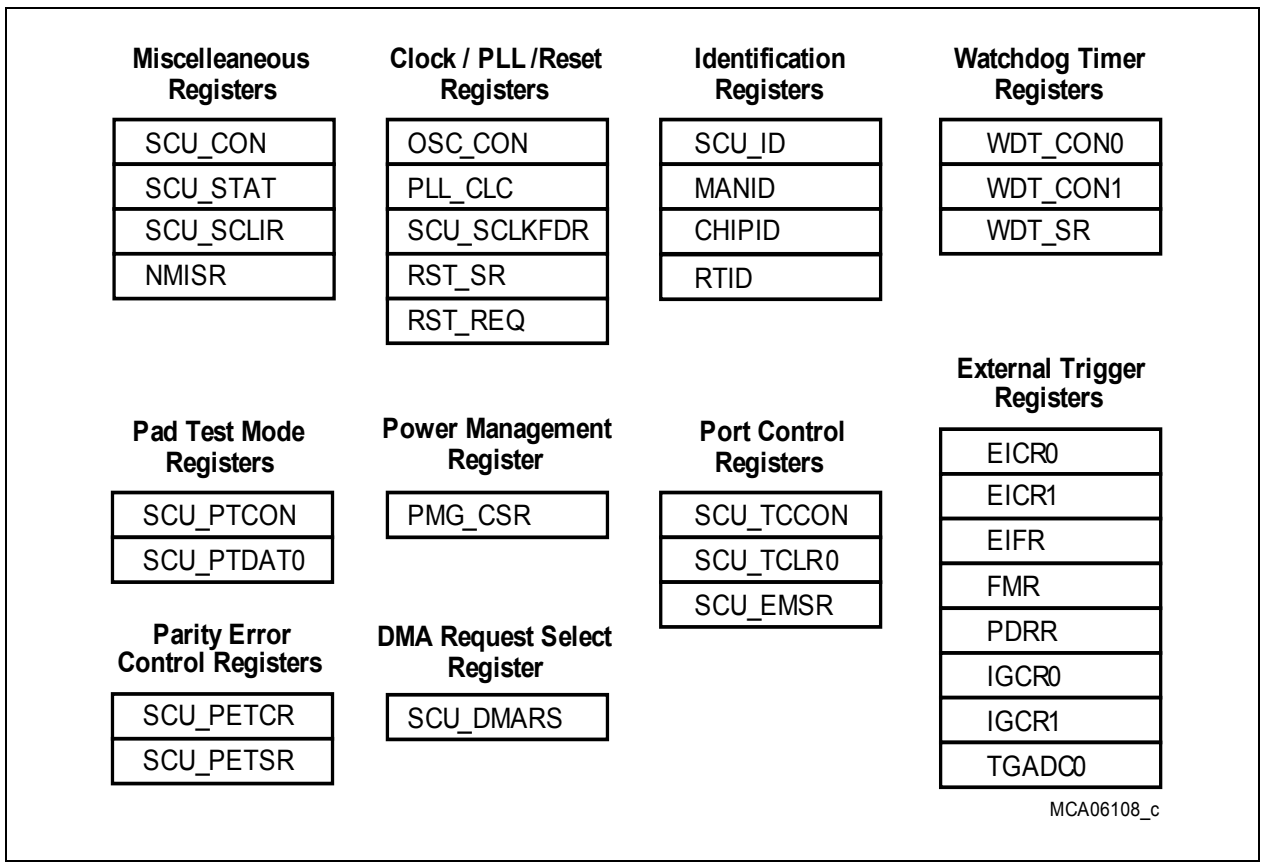


Figure 5-15 SCU Registers

Table 5-9 SCU Registers

Register Short Name	Register Long Name	Offset Address	Description see
SCU_ID	SCU Module IDentification Register	08_H	Page 5-66
SCU_SCLKFDR	SCU System Clock Fractional Divider Register	$0C_H$	Page 3-42
RST_REQ	Reset Request Register	10_H	Page 4-4
RST_SR	Reset Status Register	14_H	Page 4-2
OSC_CON	Oscillator Control Register	18_H	Page 3-8
WDT_CON0	Watchdog Timer Control Register 0	20_H	Page 14-29

Table 5-9 SCU Registers (cont'd)

Register Short Name	Register Long Name	Offset Address	Description see
WDT_CON1	Watchdog Timer Control Register 1	24_H	Page 14-31
WDT_SR	Watchdog Timer Status Register	28_H	Page 14-32
NMISR	NMI Status Register	$2C_H$	Page 12-27
PMG_CSR	Power Management Control and Status Register	34_H	Page 5-4
SCU_SCLIR	SCU Software Configuration Latched Inputs Register	38_H	Page 9-27
PLL_CLC	PLL Clock Control Register	40_H	Page 3-16
SCU_EMSR	SCU Emergency Stop Register	44_H	Page 5-59
SCU_TCCON	SCU Temperature Compensation Control Register	48_H	Page 5-45
SCU_CON	SCU Control Register	50_H	Page 5-61
SCU_STAT	SCU Status Register	54_H	Page 5-64
SCU_TCLR0	SCU Temperature Compensation 0 Level Register	58_H	Page 5-47
MANID	Manufacturer Identification Register	70_H	Page 5-67
CHIPID	Chip Identification Register	74_H	Page 5-67
RTID	Redesign Tracing Identification Register	78_H	Page 5-68
EICR0	External Input Channel Register 0	80_H	Page 5-19
EICR1	External Input Channel Register 1	84_H	Page 5-22
EIFR	External Input Flag Register	88_H	Page 5-25
FMR	Flag Modification Register	$8C_H$	Page 5-26
PDRR	Pattern Detection Result Register	90_H	Page 5-27
IGCR0	Interrupt Gating Register 0	94_H	Page 5-28
IGCR1	Interrupt Gating Register 1	98_H	Page 5-30
TGADC0	Trigger Gating ADC0 Register	$9C_H$	Page 5-33
SCU_PTCON	SCU Pad Test Control Register	$B0_H$	Page 5-54
SCU_PTDAT0	SCU Pad Test Data Register 0	$B4_H$	Page 5-56
SCU_PETCR	SCU Parity Error Trap Control Register	$D0_H$	Page 5-40
SCU_PETSR	SCU Parity Error Trap Status Register	$D4_H$	Page 5-41
SCU_DMARS	SCU DMA Request Select Register	$D8_H$	Page 5-50

5.14.1 Software Configuration Latched Inputs Register

SCU_SCLIR
SCU Software Configuration Latched Inputs Register
($F0000038_H$) **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
SWOPT15	SWOPT14	SWOPT13	SWOPT12	SWOPT11	SWOPT10	SWOPT9	SWOPT8	SWOPT7	SWOPT6	SWOPT5	SWOPT4	SWOPT3	SWOPT2	SWOPT1	SWOPT0
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Function
SWOPTx (x = 0-15)	x	rh	**Software Configuration Bits**
0	[31:16]	r	**Reserved**

6 On-Chip System Buses and Bus Bridges

The TC1766 has two independent on-chip buses:

- Local Memory Bus (LMB)
- System Peripheral Bus (SPB)

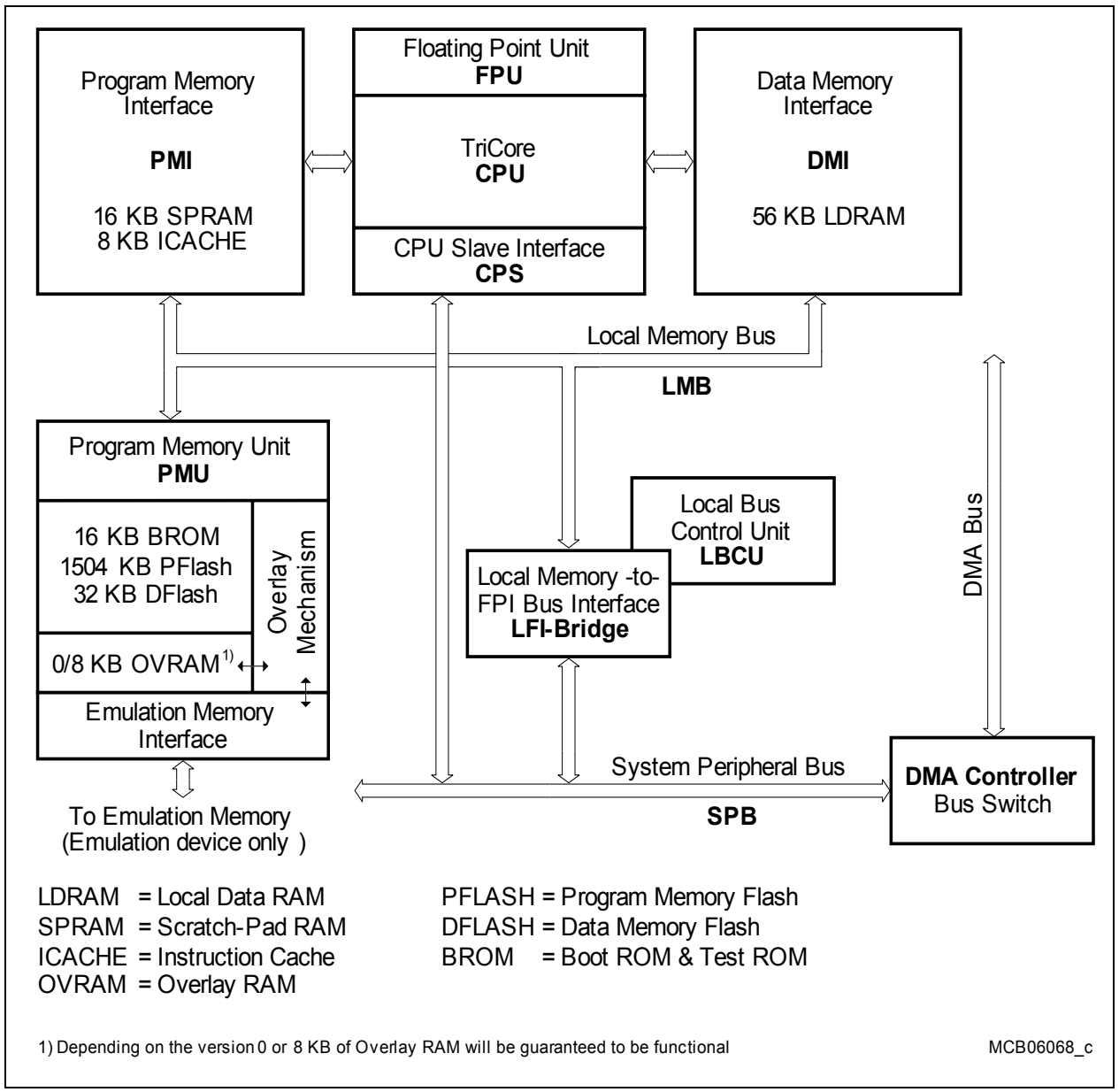


Figure 6-1 Buses in TC1766 Processor Subsystem

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.

The LMB runs at CPU clock speed f_{CPU}, while the SPB Bus runs at system clock speed f_{SYS}. Note that in the TC1766, f_{SYS} is equal to f_{CPU}, limited to maximum of 80 MHz.

6.1 Local Memory Bus

The following terminology is used for the bus:

Table 6-1 LMB BusTerms

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.

6.1.1 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)

6.1.2 Transaction Types

There are three transaction types of the LMB.

6.1.2.1 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

6.1.2.2 Block Transfers

Block transfers are only issued in two ways:

1. By the PMI 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.

6.1.2.3 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.

6.1.3 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.

6.1.4 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.

6.1.5 LMB Basic Operation

Figure 6-2 describes some basic bus operations of the LMB.

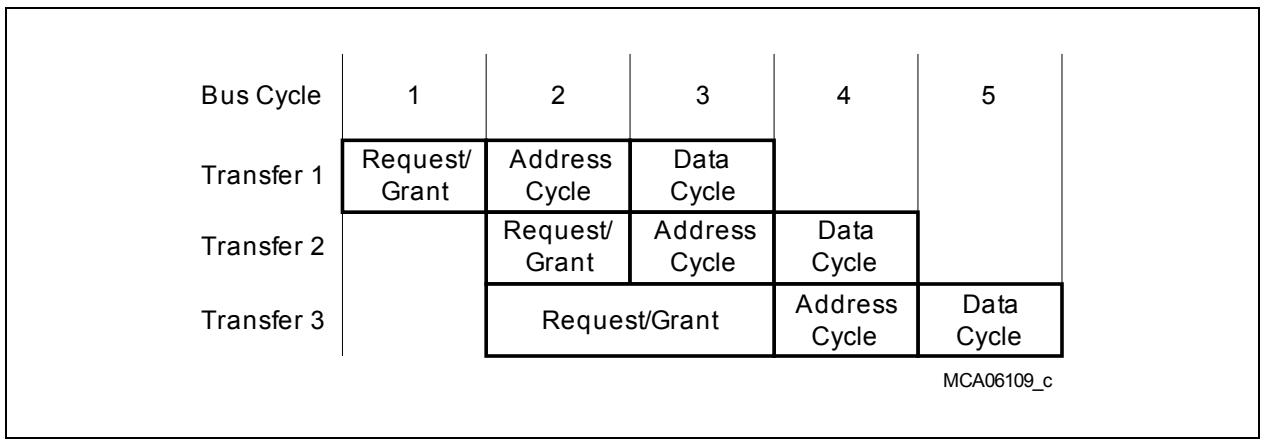


Figure 6-2 Basic LMB Transactions

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 6-3**, transfer 1 is a block transfer while transfer 2 is a single transfer.

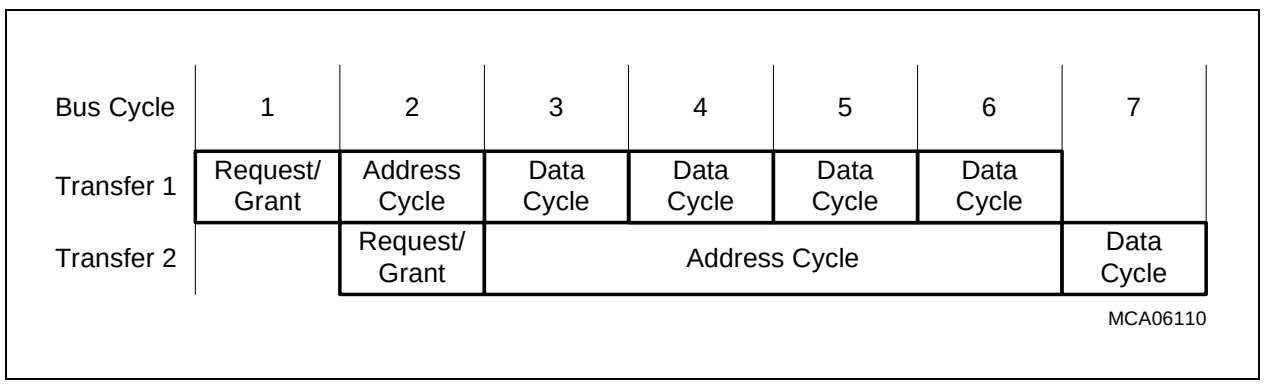


Figure 6-3 LMB Block Transactions

6.2 Local Memory Bus Controller Unit

The LMB in the TC1766 has an LMB Bus Control Unit (LBCU).

6.2.1 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.

6.2.2 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 6-2**.

Table 6-2 Priority of Master LMB Agents

Priority	LMB Master	Comment
Highest	LFI Bridge	–
Lowest	Data Memory Interface (DMI)	Default Master
	Program Memory Interface (PMI)	–

For all the masters requesting the LMB during any one cycle, the granted master is the one with the highest priority.

6.2.2.1 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.

6.2.3 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 transaction from the PCP, DMA or the Cerberus causes a bus error on the LMB Bus, the originating masters are not informed about this bus error because they are not an LMB master agent. 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.

6.2.4 LMB Bus Registers

This section describes the registers of the LBCU module. The complete and detailed address map of LBCU is described in**Table 16-30** on **Page 16-83** of this TC1766 System Units (Vol. 1 of 2) User's Manual.

LMB Bus Register Overview

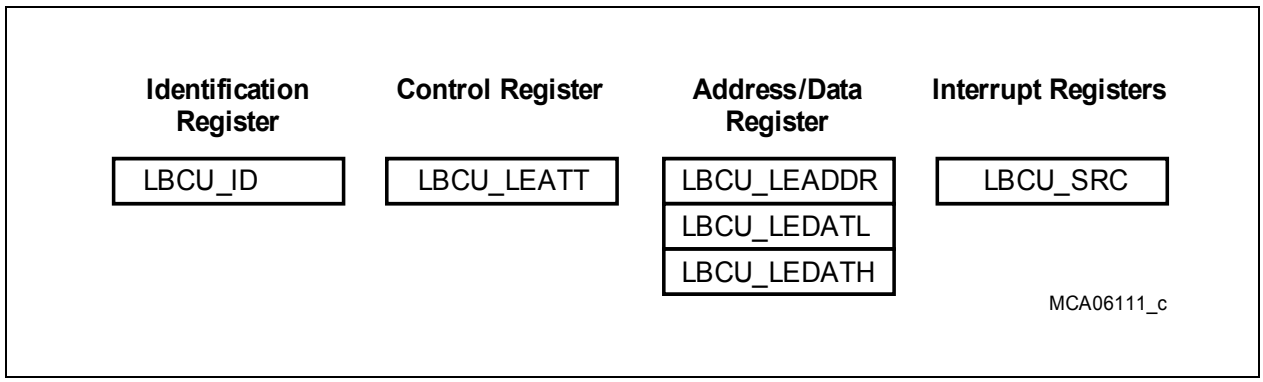


Figure 6-4 LMB Bus Control Unit Registers

Table 6-3 Registers Address Space

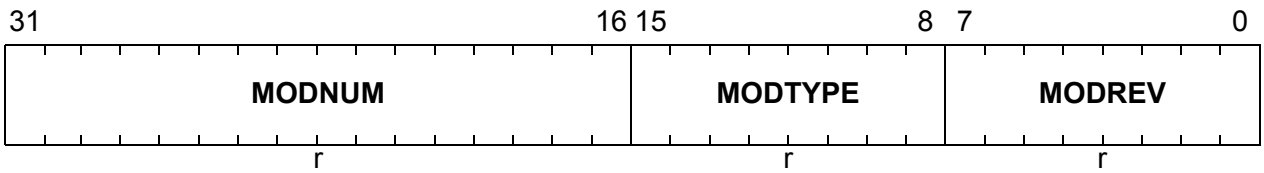
Module	Base Address	End Address	Note
LBCU	F87F FE00$_H$	F87F FEFF$_H$	-

Table 6-4 Registers Overview - LMB Bus Control Unit Registers

Register Short Name	Register Long Name	Offset Address	Description see
LBCU_ID	LBCU Module Identification Register	08$_H$	**Page 6-8**
LBCU_LEATT	LBCU LMB Error Attribute Register	20$_H$	**Page 6-8**
LBCU_LEADDR	LBCU LMB Error Address Register	24$_H$	**Page 6-11**
LBCU_LEDATL	LBCU LMB Error Data Low Register	28$_H$	**Page 6-12**
LBCU_LEDATH	LBCU LMB Error Data High Register	2C$_H$	**Page 6-12**
LBCU_SRC	LBCU Service Request Control Register	FC$_H$	**Page 6-13**

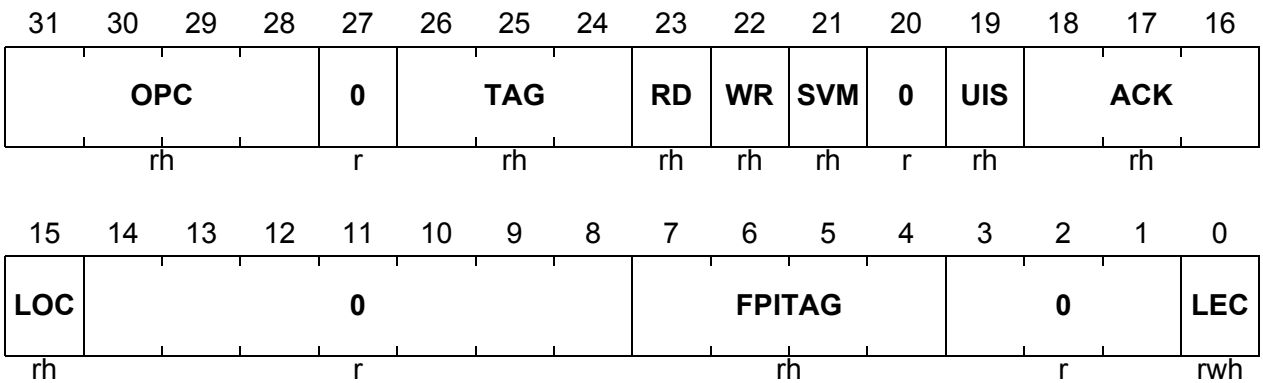
The LBCU Module Identification Register ID contains read-only information about the LBCU module version.

LBCU_ID
LBCU Module Identification Register (08_H) **Reset Value: $000F\ C0XX_H$**



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 LBCU: $000F_H$

LBCU_LEATT
LBCU LMB Error Attribute Register (20_H) **Reset Value: $XXXX\ XXX0_H$**



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, LEDATL, LEDATH, and bits [31:4] of LEATT contain valid data. 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 either by the PCP, DMA or by Cerberus.
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. Other bit combinations are reserved.
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.

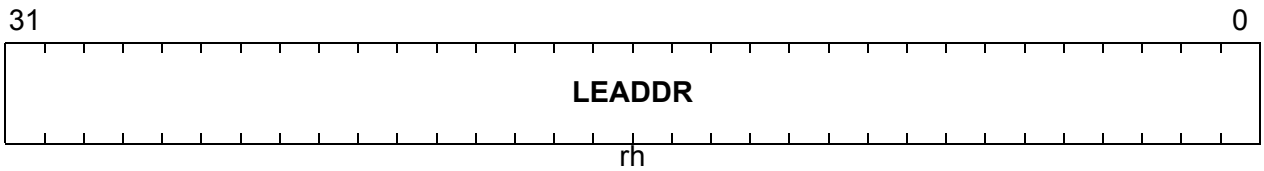
Field	Bits	Type	Description
SVM	21	rh	**LMB Bus Supervisor Mode** This bit indicates whether the LMB bus error occurred in supervisor mode or in user mode. 0_B Transfer was initiated in supervisor mode. 1_B Transfer was initiated in user mode.
WR	22	rh	**LMB Bus Write Error Indication** This bit indicates whether the LMB bus error occurred at a write cycle (see **Table 6-5**).
RD	23	rh	**LMB Bus Read Error Indication** This bit indicates whether the LMB bus error occurred at a read cycle (see **Table 6-5**).
TAG	[26:24]	rh	**LMB Master TAG** This bit field indicates the LMB master device in case of an LMB bus error. 000_B LFI 010_B PMI 100_B DMI Other bit combinations are reserved.
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 Other bit combinations are reserved.
0	[3:1], [14:8], 20, 27	r	**Reserved** Read as 0; should be written with 0.

Note: LEATT[31:4] contains valid read data only if its bit LEC is set.

Table 6-5 LMB Bus Read/Write Error Indication

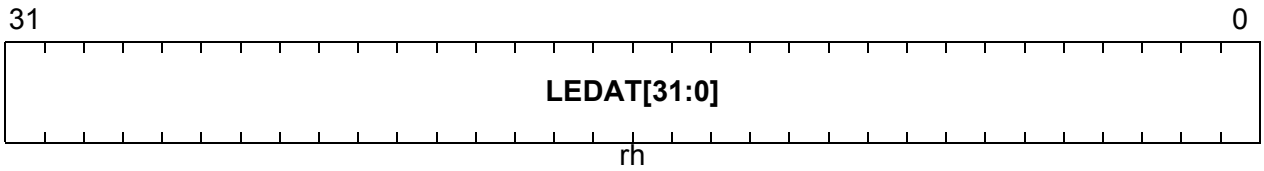
RD	WR	LMB Bus Cycle
0	0	LMB bus error occurred at the read cycle of an atomic transfer.
0	1	LMB bus error occurred at a read cycle of a single transfer.
1	0	LMB bus error occurred at a write cycle of a single transfer or at the write cycle of an atomic transfer.
1	1	Does not occur.

LBCU_LEADDR
LBCU LMB Error Address Register (24_H) Reset Value: XXXX XXXX$_H$



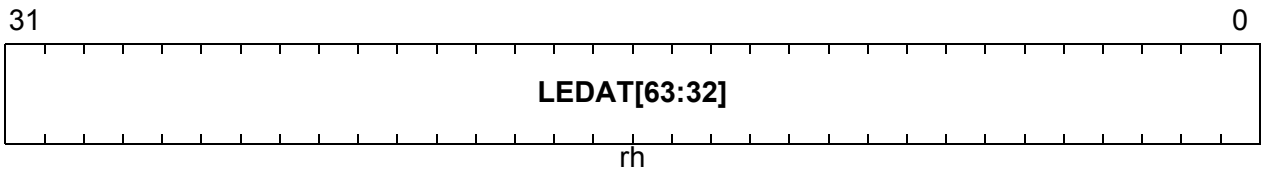
Field	Bits	Type	Description
LEADDR	[31:0]	rh	**LMB Address** This bit field holds the LMB address that has been captured at an LMB error. LEADDR only contains valid read data when bit LEC in the corresponding register LEATT is set.

LBCU_LEDATL
LBCU LMB Error Data Low Register (28_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 ($2C_H$) 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

(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	TOS = 00B		0		SRPN							
w	w	rh	rw	r		r		rw							

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.

*Note: Further details on interrupt handling and processing are described in **Chapter 12** of this TC1766 System Units (Vol. 1 of 2) User's Manual.*

6.3 Local Memory to FPI Bus Interface (LFI Bridge)

This section describes the basic functionality of the LFI Bridge.

6.3.1 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 either by PCP, DMA or by Cerberus.

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 either by the PCP, DMA controller or Cerberus) via the LFI Bridge that address an LMB slave device, are translated into an LMB address according to **Table 6-6**.

Table 6-6 SPB to LMB Bus Address Translation

Transaction Destination	SPB Access Range	Translated LMB Address
SRAM	$E800\ 0000_H$ - $E800\ FFFF_H$	$C000\ 0000_H$ - $C000\ FFFF_H$
Reserved	$E801\ 0000_H$ - $E83F\ FFFF_H$	$C001\ 0000_H$ - $C03F\ FFFF_H$
DMI LDRAM	$E840\ 0000_H$ - $E840\ FFFF_H$	$D000\ 0000_H$ - $D000\ FFFF_H$
Reserved	$E841\ 0000_H$ - $E84F\ FFFF_H$	$D001\ 0000_H$ - $D00F\ FFFF_H$
PMI SPRAM	$E850\ 0000_H$ - $E850\ BFFF_H$	$D400\ 0000_H$ - $D400\ BFFF_H$
Reserved	$E850\ C000_H$ - $E85F\ FFFF_H$	$D400\ C000_H$ - $D40F\ FFFF_H$

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.

6.3.2 LFI Register

This section describes the kernel register of the LFI Bridge.

LFI Register

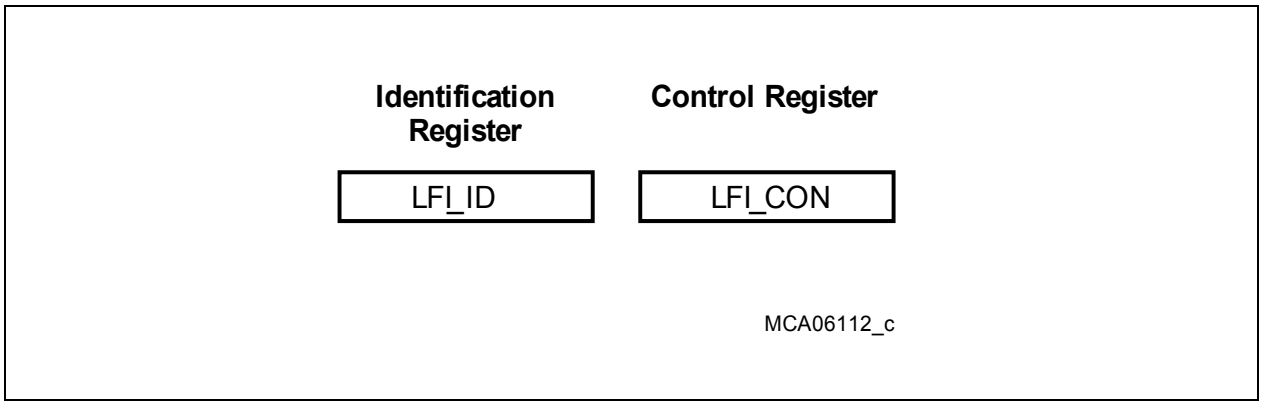


Figure 6-5 LFI Register

The complete and detailed LFI Bridge address map is shown in Table 16-31 on Page 16-84 of this TC1766 System Units (Vol. 1 of 2) User's Manual.

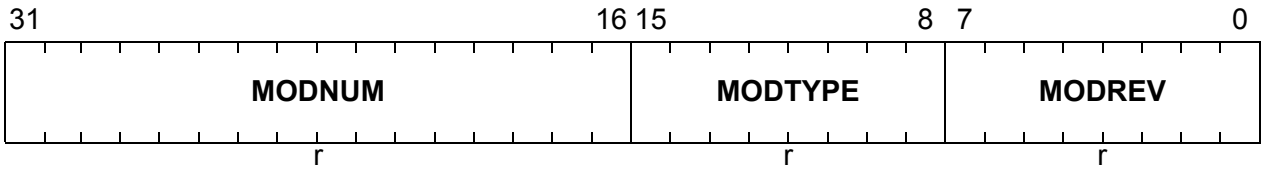
Table 6-7 Registers Address Space - LFI Bridge

Module	Base Address	End Address	Note
LFI	$F87F\ FF00_H$	$F87F\ FFFF_H$	-

Table 6-8 Registers Overview LFI Register

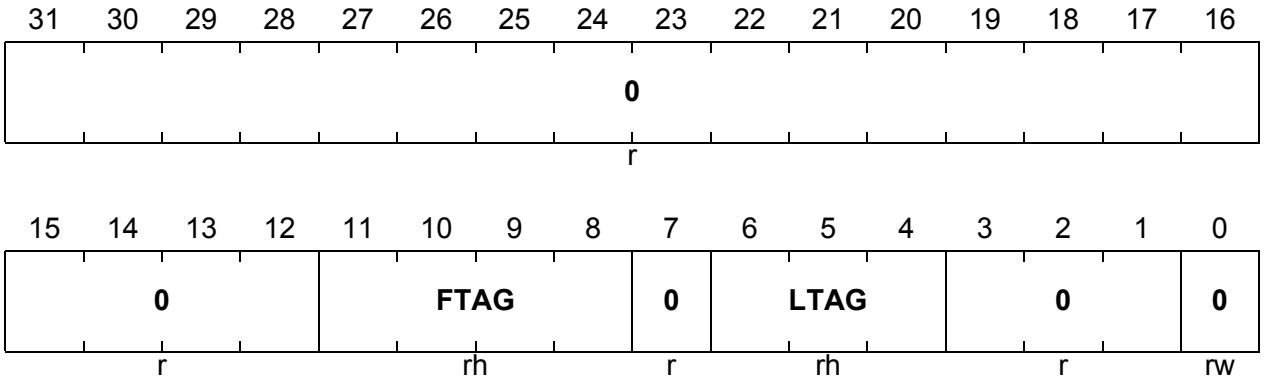
Register Short Name	Register Long Name	Offset Address	Description see
LFI_ID	LFI Module Identification Register	08_B	Page 6-17
LFI_CON	LFI Configuration Register	10_B	Page 6-18

LFI_ID
LFI Module Identification Register (08_H) Reset Value: 000C 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 LFI: $000C_H$

LFI_CON
LFI Configuration Register **(10_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 TC1766, the bit field LTAG = 000_B.
FTAG	[11:8]	rh	**FPI Bus (SPB) Tag ID** In the TC1766, 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.

6.4 System Peripheral Bus

The TC1766 has one on-chip FPI Bus:

- System Peripheral Bus (SPB)
  - System bus for on-chip peripherals (except for SSCs, ADC0 and FADC)

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.

6.4.1 Overview

The FPI Bus interconnects the on-chip peripheral functional units with the TC1766 processor subsystem. **Figure 6-6** gives an overview of the FPI Bus and the modules connected to it.

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
- 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
- Designed to minimize EMI and power consumption

The functional units of the TC1766 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. When bus ownership is granted by the

bus control unit, 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. In the TC1766, DMI and PMI (via the LFI Bridge), PCP, DMA and Cerberus operate as FPI Bus masters. On-chip peripheral units are typically FPI Bus slaves. **Figure 6-6** shows the FPI Bus and the interface types of the various modules.

FPI Bus arbitration is performed by the Bus Control Unit of the FPI Bus. In case of bus errors, the BCU generates an interrupt request to the CPU and provides debugging information about the actual bus error to the CPU.

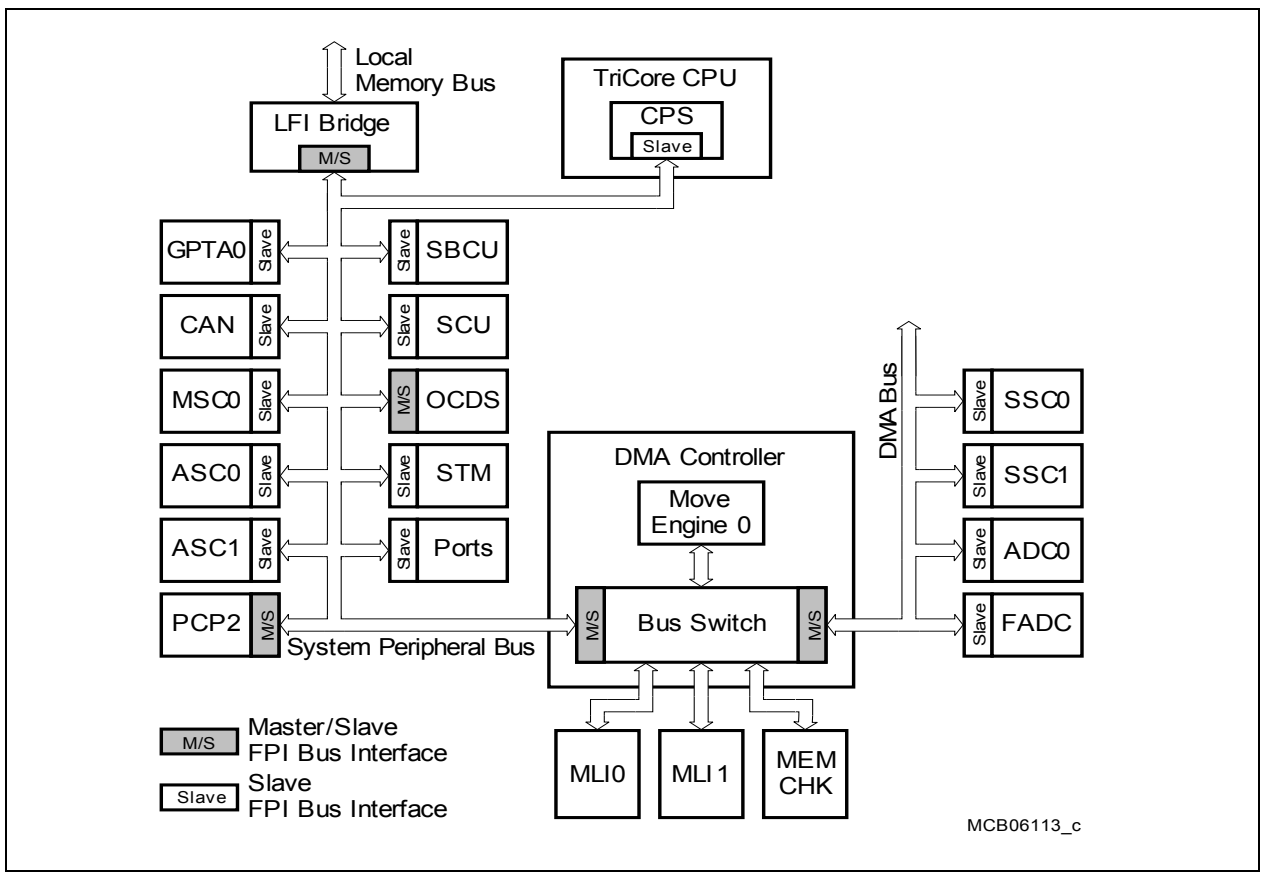


Figure 6-6 TC1766 FPI Bus Block Diagram

6.4.2 Bus Transaction Types

Thsi 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 TC1766 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 6-12** for available FPI Bus transfer types.*

6.4.3 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.

6.4.4 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.

6.4.5 FPI Bus Basic Operations

This section describes some basic transactions on the FPI Bus.

The example in **Figure 6-7** 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.

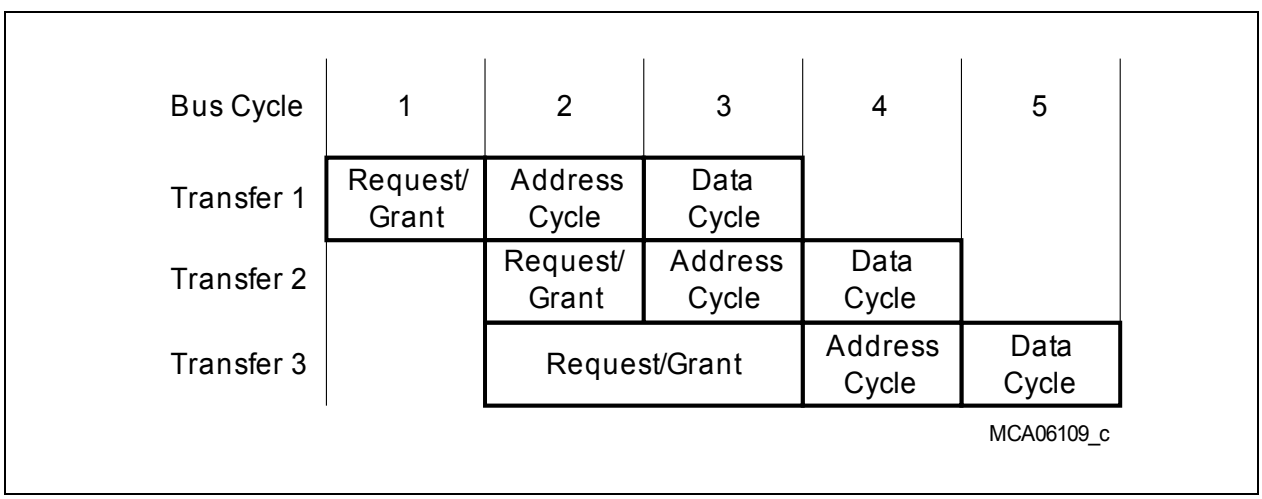


Figure 6-7 Basic FPI Bus Transactions

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 6-8**, transfer 1 is a block transfer, while transfer 2 is a single transfer.

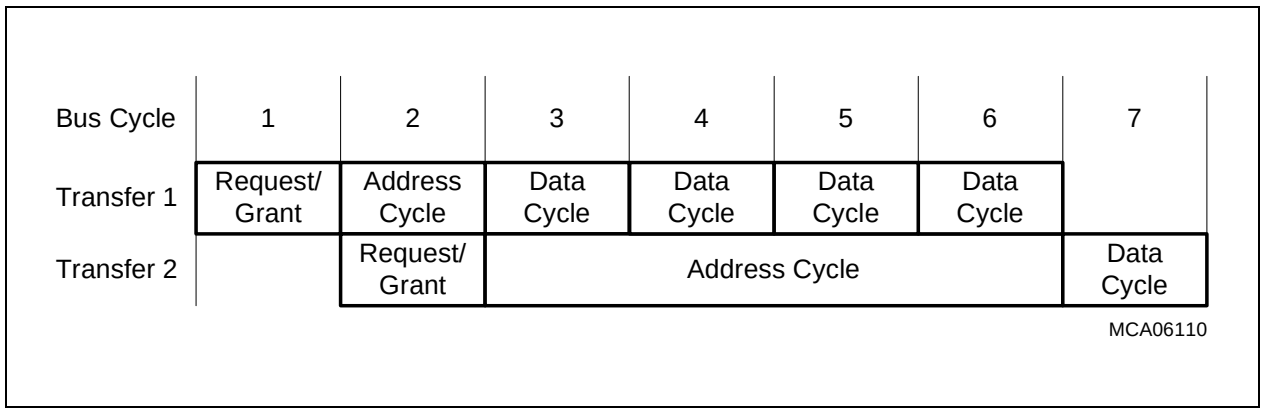


Figure 6-8 FPI Bus Block Transactions

6.5 FPI Bus Control Unit (SBCU)

The TC1766 incorporates one BCU for the SPB, called SBCU.

6.5.1 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.

6.5.1.1 Arbitration on the System Peripheral Bus

The TC1766 SPB has four bus agents that can become a SPB bus master. Each agent is supplied an arbitration priority as shown in **Table 6-9**. DMA controller and OCDS agents can be assigned to low or high priorities by software.

Table 6-9 Priority of TC1766 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
	On-Chip Debug System (Cerberus), high priority	Priority selection by software
	Peripheral Control Processor	Default master 0
	DMA Controller, high priority channels	Priority selection by software
	LFI Bridge	Default master 1
	DMA Controller, low priority channels	Priority selection by software
lowest	On-Chip Debug System (Cerberus), low priority	Priority selection by software

If there is no request from an SPB bus master, the SPB is granted to a default master (PCP or LFI Bridge) which has been at last the default master.

The DMA Bus in the TC1766 has only one bus master agent, which is the DMA controller. Therefore, the DMA controller is always the default master.

6.5.1.2 Starvation Prevention

Starvation prevention is a feature that is especially important for the SPB. 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, an optional feature of the TC1766 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 incremented each time an arbitration is performed in the BCU. When this counter reaches a user-programmable threshold value, 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 counter reaches the threshold value, it is automatically reset to zero and starts counting up again. 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.

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.

6.5.2 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 6-10**.

Table 6-10 FPI Bus Acknowledge Codes

Code (ACK)	Description
00_B	NSC: No Special Condition.
01_B	SPT: Split Transaction (not used in the TC1766).
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.

Each master on the FPI Bus is assigned a 4-bit identification number, the master TAG number (see **Table 6-11**). This makes it possible to distinguish which master has performed the current transaction.

Table 6-11 FPI Bus Master TAG Assignments

TAG-Number	FPI Bus	Description
1001_B	SPB	Peripheral Control Processor (PCP2)
1010_B		DMA Controller
1011_B		LFI Bridge
1100_B		OCDS (Cerberus)

Transactions on the FPI Bus are classified via a 4-bit operation code (see **Table 6-12**). Note that split transactions (OPC = 1000_B to 1110_B) are not used in the TC1766.

Table 6-12 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_B	No operation
0011_B, 0111_B, 1000_B - 1110_B	Reserved

6.5.3 Clock Management

The BCU can be configured so that it shuts down automatically when not needed by disabling its internal clock. When it is needed again, for instance when a bus request signal is received from a master, the BCU will enable its clock and perform the arbitration. If no further bus activity is required after the transfer has completed, the BCU will automatically shut off its clock and return to idle mode.

Automatic power management is controlled through the SBCU_CON.PSE bit. When cleared to 0, power management is disabled, and the BCU clock is always active. This might be required, for instance, to debug both the active and idle FPI Bus states of an application via an external emulator or other debugging hardware.

6.5.4 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

6.5.4.1 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.

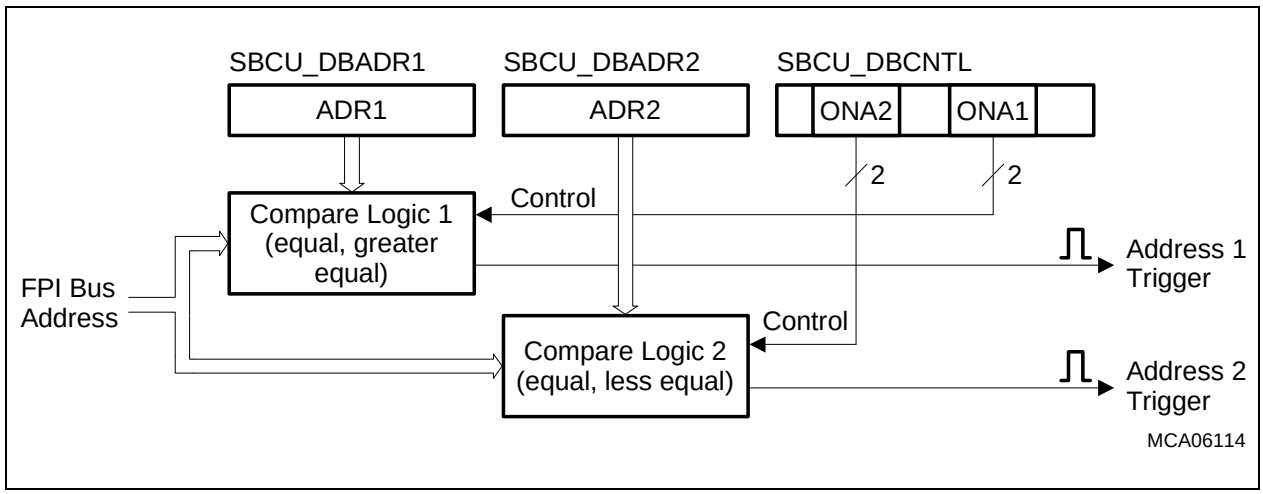


Figure 6-9 Address Trigger Generation

6.5.4.2 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.

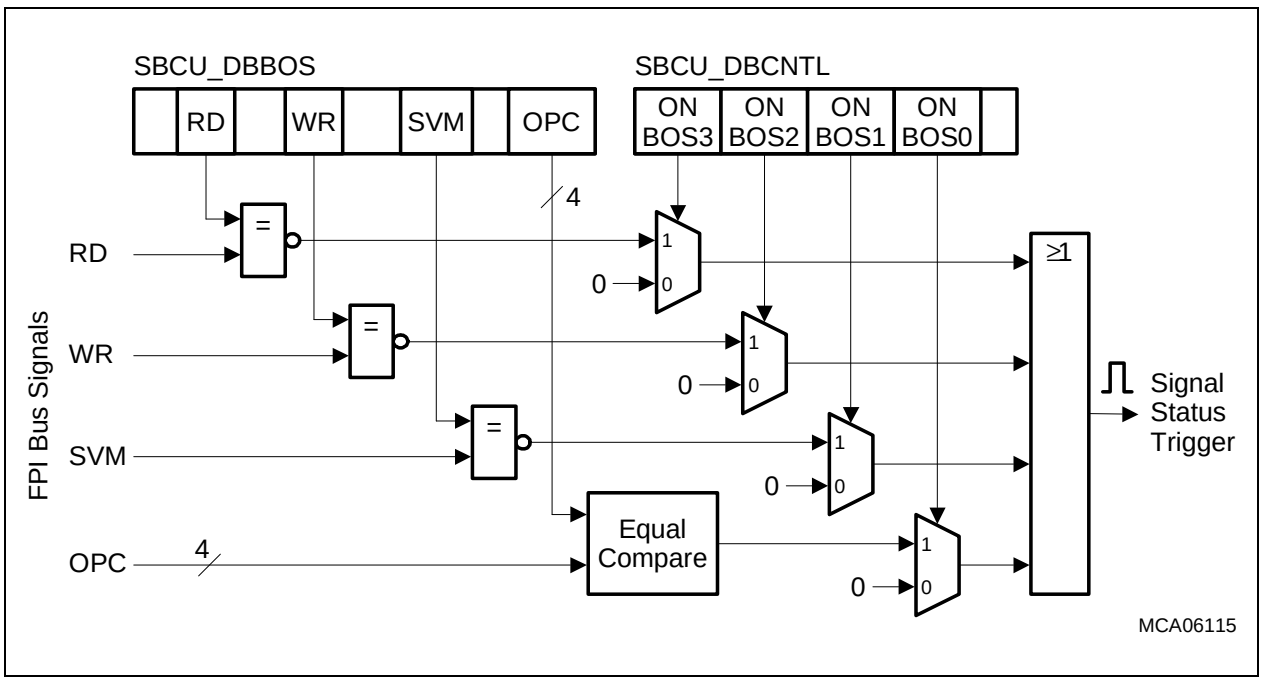


Figure 6-10 Signal Status Trigger Generation

6.5.4.3 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 6-11**).

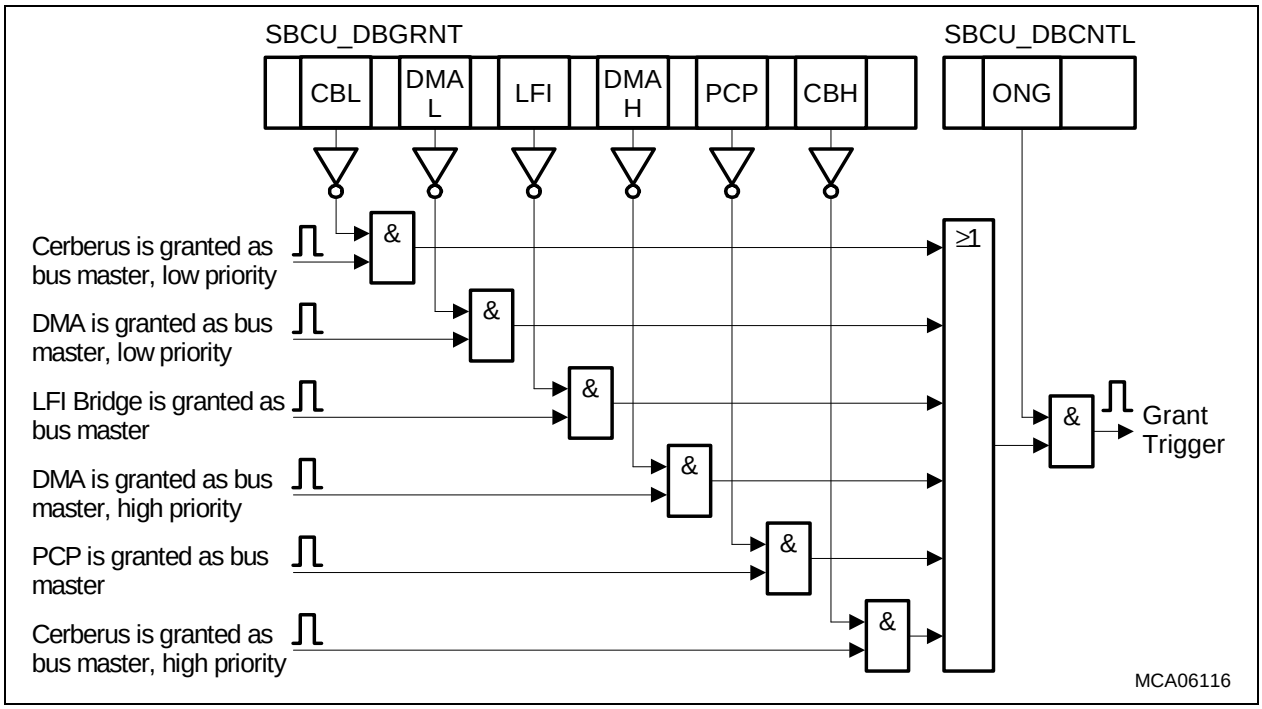


Figure 6-11 Grant Trigger Generation

6.5.4.4 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 6-12**). 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.

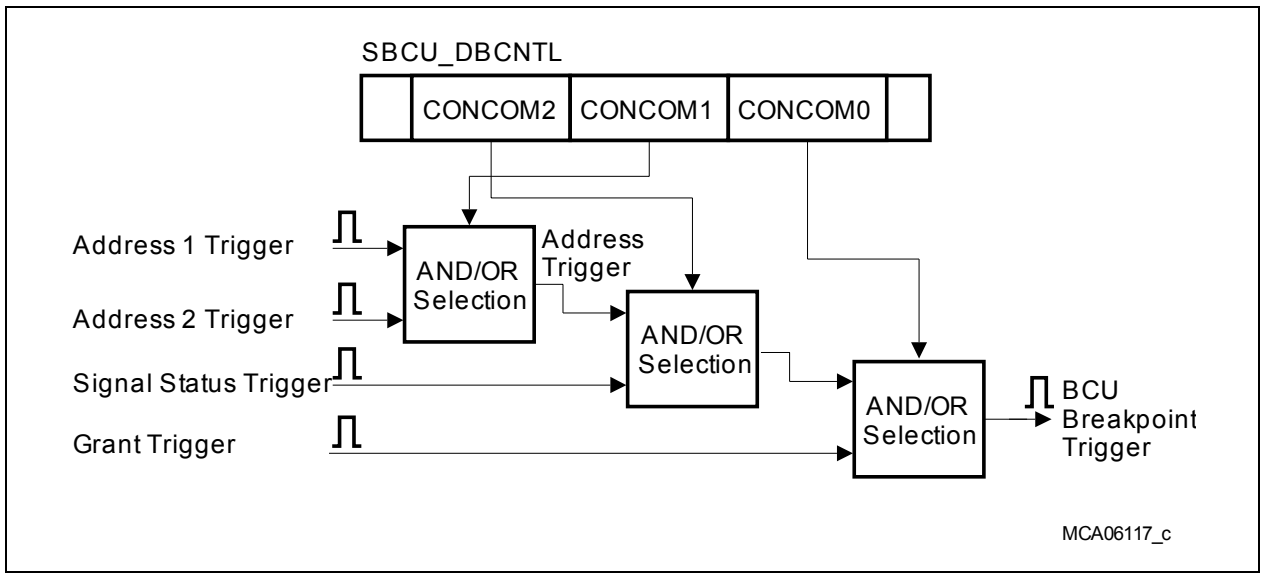


Figure 6-12 BCU Breakpoint Trigger Combination Logic

6.5.4.5 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 masters PCP or LFI Bridge.

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 masters 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 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 = $FFFFFFF7_H$:
   means that the grant trigger for the SPB bus master PCP is enabled.
5. Writing SBCU_DBBOS is "don't care". No signal trigger for SVM, WR, or RD is generated.

6.5.5 SBCU Registers

The registers shown in **Figure 6-13** are available for the SBCU.

SBCU Registers Overview

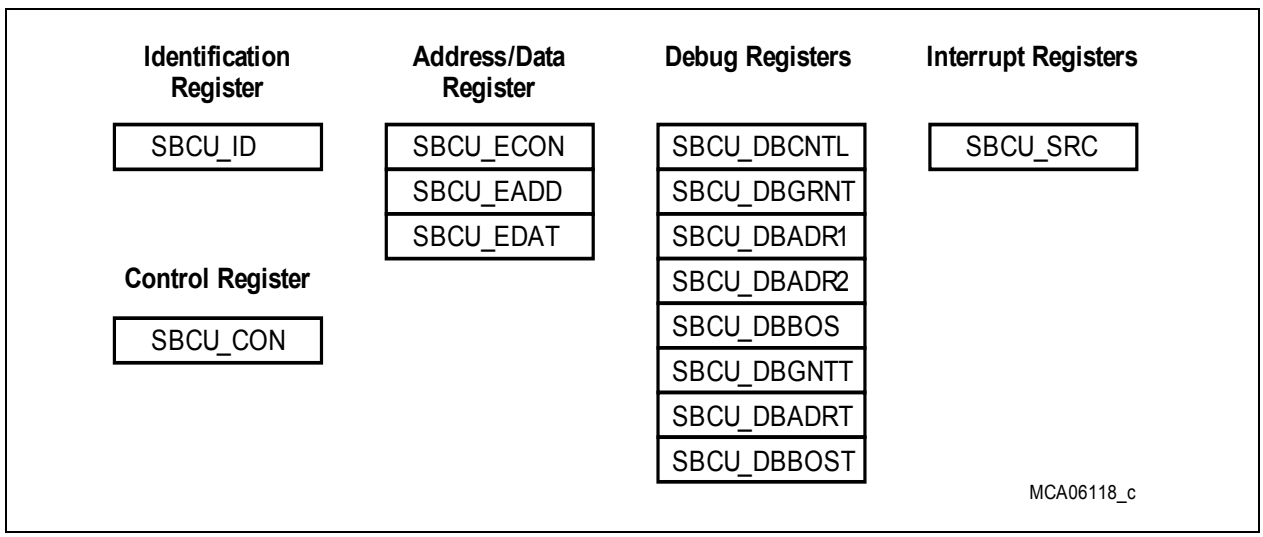


Figure 6-13 SBCU Registers

The complete and detailed address map of SBCU is described in **Table 16-4** on **Page 16-10** of this TC1766 System Units (Vol. 1 of 2) User's Manual.

Table 6-13 Registers Address Space

Module	Base Address	End Address	Note
SBCU	$F000\ 0100_H$	$F000\ 01FF_H$	-

Table 6-14 Registers Overview - SBCU Registers

Register Short Name	Register Long Name	Offset Address	Description see
SBCU_ID	SBCU Module Identification Register	08_H	**Page 6-37**
SBCU_CON	SBCU Control Register	10_H	**Page 6-37**
SBCU_ECON	SBCU Error Control Capture Register	20_H	**Page 6-39**
SBCU_EADD	SBCU Error Address Capture Register	24_H	**Page 6-41**
SBCU_EDAT	SBCU Error Data Capture Register	28_H	**Page 6-42**
SBCU_DBCNTL	SBCU Debug Control Register	30_H	**Page 6-43**
SBCU_DBGRNT	SBCU Debug Grant Mask Register	34_H	**Page 6-46**
SBCU_DBADR1	SBCU Debug Address 1 Register	38_H	**Page 6-47**

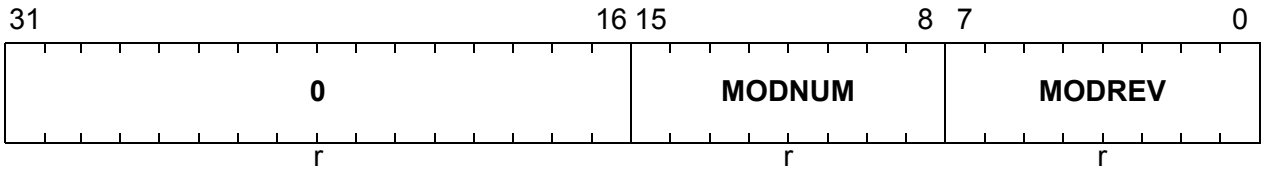
Table 6-14 Registers Overview - SBCU Registers (cont'd)

Register Short Name	Register Long Name	Offset Address	Description see
SBCU_DBADR2	SBCU Debug Address 2 Register	$3C_H$	Page 6-48
SBCU_DBBOS	SBCU Debug Bus Operation Signals Register	40_H	Page 6-49
SBCU_DBGNTT	SBCU Debug Trapped Master Register	44_H	Page 6-50
SBCU_DBADRT	SBCU Debug Trapped Address Register	48_H	Page 6-53
SBCU_DBBOST	SBCU Debug Trapped Bus Operation Signals Register	$4C_H$	Page 6-54
SBCU_SRC	SBCU Service Request Control Register	FC_H	Page 6-57

6.5.5.1 SBCU Control Registers

The SBCU Module Identification Register ID contains read-only information about the SBCU module version.

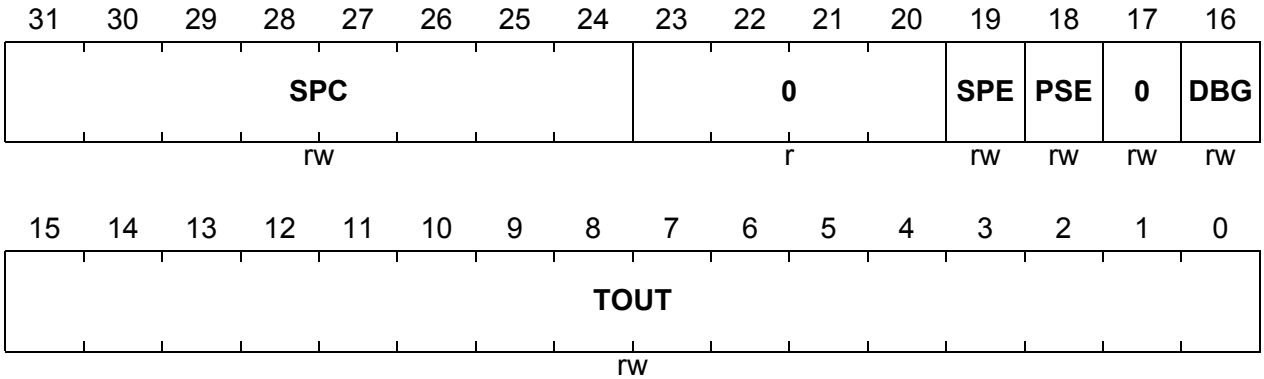
SBCU_ID
SBCU Module Identification Register (08_H) **Reset Value: 0000 6AXX$_H$**



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 SBCU: $6A_H$
0	[31:16]	r	**Reserved** Read as 0.

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 **(10_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)
PSE	18	rw	**SBCU Power Saving (Automatic Clock Control) Enable** 0_B SBCU power saving disabled (default after reset) 1_B SBCU power saving enabled
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	17	rw	**Reserved** Read as 0 after reset; should be written with 0.
0	[23:20]	r	**Reserved** Read as 0; should be written with 0.

6.5.5.2 SBCU Error Registers

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. A hardware 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 (20_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 6-10**.
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 6-15**).
RDN	23	rwh	**State of FPI Bus Read Signal** This bit indicates whether the FPI Bus error occurred at a read cycle (see**Table 6-15**).

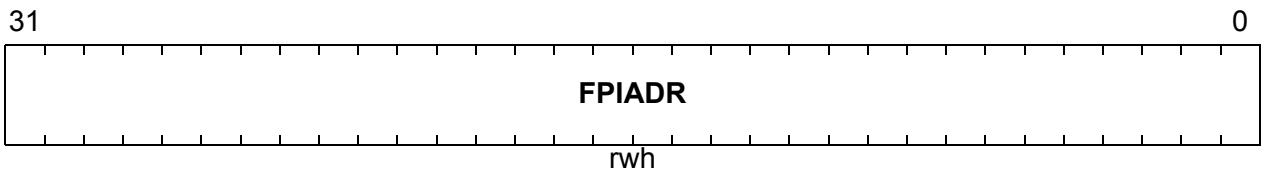
Field	Bits	Type	Description
TAG	[27:24]	rwh	**FPI Bus Master Tag Number Signals** This bit field indicates the FPI Bus master TAG number (definitions see **Table 6-11**).
OPC	[31:28]	rwh	**FPI Bus Operation Code Signals** The FPI Bus operation codes are defined in **Table 6-12**.

1) In the TC1766, aborted accesses to a 0 wait state SPB slave may also increment ERRCNT when the slave generates an error acknowledge.

Table 6-15 FPI Bus Read/Write Error Indication

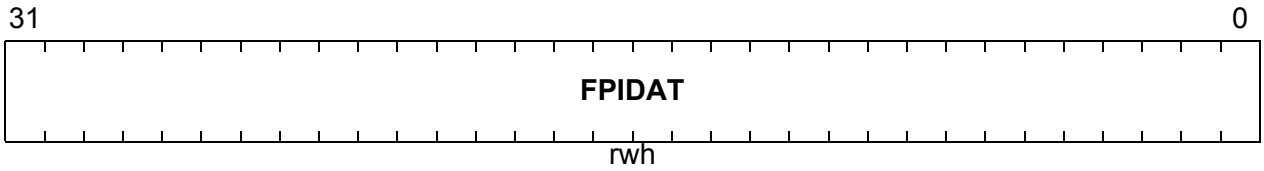
$\overline{RD}$	$\overline{WR}$	FPI Bus Cycle
0	0	FPI Bus error occurred at the read transfer of a read-modify-write transfer.
0	1	FPI Bus error occurred at a read cycle of a single transfer.
1	0	FPI Bus error occurred at a write cycle of a single transfer or at the write cycle of a read-modify-write transfer.
1	1	Does not occur.

SBCU_EADD
SBCU Error Address Capture Register (24_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 (28_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.

6.5.5.3 OCDS Registers

SBCU_DBCNTL
SBCU Debug Control Register (30_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

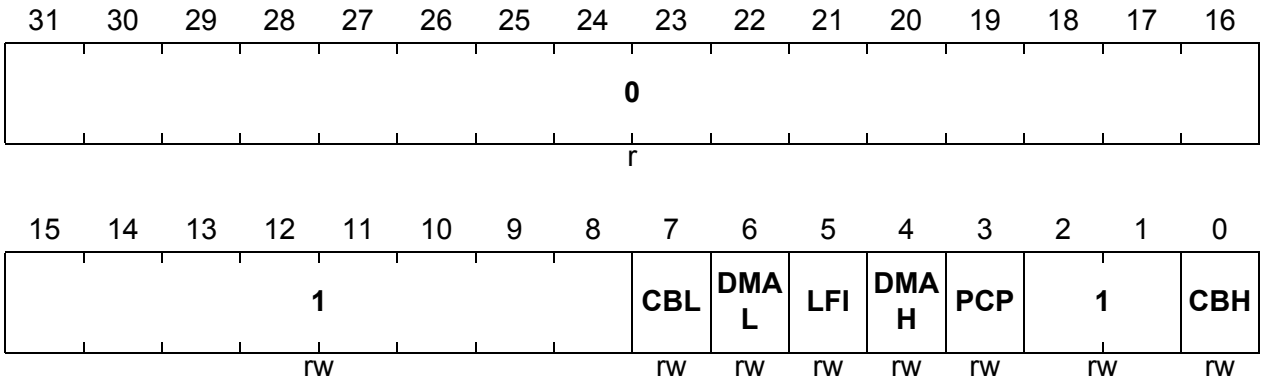
Field	Bits	Type	Description
EO	0	r	**Status of SBCU Debug Support Enable** This bit is controlled by the Cerberus and enables the SBCU debug support. 0_B SBCU debug support is disabled. 1_B SBCU debug support is enabled. (default after reset)
OA	1	r	**Status of SBCU Breakpoint Logic** 0_B The SBCU breakpoint 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.
CONCOM0	12	rw	**Grant and Address Trigger Relation** 0_B Grant trigger condition and the address trigger condition are combined with a logical OR for further control. 1_B The grant trigger condition and the address trigger condition are combined with a logical AND for further control (see **Figure 6-12**).

Field	Bits	Type	Description
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 for further control. 1_B Address 1 trigger condition and address 2 trigger condition are combined with a logical AND for further control (see **Figure 6-12**).
CONCOM2	14	rw	**Address and Signal Trigger Relation** 0_B Address trigger condition and signal trigger condition are combined with a logical OR for further control. 1_B Address trigger condition and the signal trigger condition are combined with a logical AND for further control (see **Figure 6-12**).
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 6-11**).
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 6-9**.
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 6-9**.

Field	Bits	Type	Description
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 6-10**).
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 6-10**).
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 6-10**).
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 6-10**).
0	[3:2], [11:5], 15, [19:17], [23:22], [27:26]	r	**Reserved** Read as 0; should be written with 0.

SBCU_DBGRNT

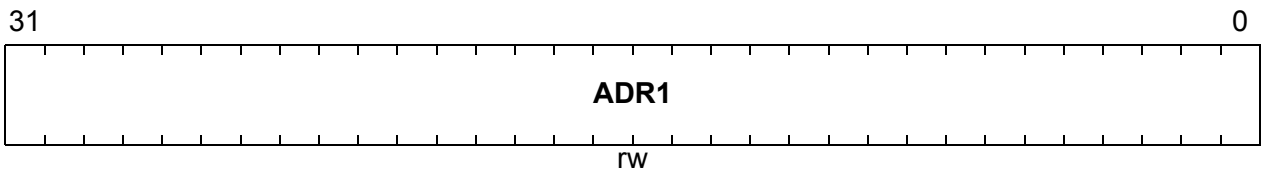
SBCU Debug Grant Mask Register (34_H) Reset Value: 0000 FFFF$_H$



Field	Bits	Type	Description
CBH	0	rw	**Cerberus Grant Trigger Enable, High Priority** 0_B FPI Bus transactions with high-priority Cerberus as bus master are enabled for grant trigger event generation. 1_B FPI Bus transactions with high-priority Cerberus as bus master are disabled for grant trigger event generation.
1	[2:1], [15:8]	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.
DMAH	4	rw	**DMA Grant Trigger Enable, High Priority** 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.

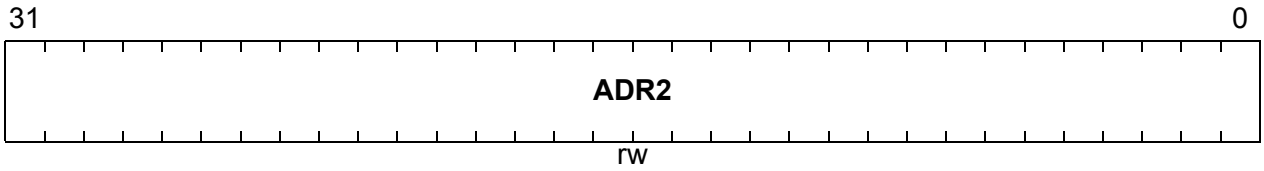
Field	Bits	Type	Description
LFI	5	rw	**LFI Bridge Grant Trigger Enable** 0_B FPI Bus transactions with LFI Bridge as bus master are enabled for grant trigger event generation. 1_B FPI Bus transactions with LFI Bridge as bus master are disabled for grant trigger event generation.
DMAL	6	rw	**DMA Grant Trigger Enable, Low Priority** 0_B FPI Bus transactions with high-priority DMA channels as bus master are enabled for grant trigger event generation. 1_B FPI Bus transactions with high-priority DMA channels as bus master are disabled for grant trigger event generation.
CBL	7	rw	**Cerberus Grant Trigger Enable, Low Priority** 0_B FPI Bus transactions with low-priority Cerberus as bus master are enabled for grant trigger event generation. 1_B FPI Bus transactions with low-priority Cerberus as bus master are disabled for grant trigger event generation.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

SBCU_DBADR1
SBCU Debug Address 1 Register (38_H) Reset Value: 0000 0000$_H$



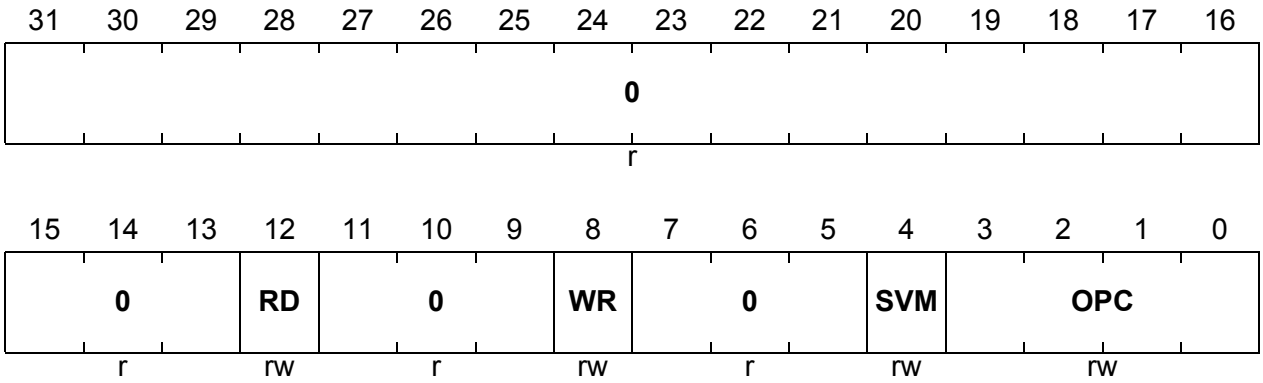
Field	Bits	Type	Description
ADR1	[31:0]	rw	**Debug Trigger Address 1** This register contains the address for the address 1 trigger event generation.

SBCU_DBADR2
SBCU Debug Address 2 Register ($3C_H$) 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 Registe
r **(40_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

Field	Bits	Type	Description
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
0	[7:5], [11:9], [31:13]	r	**Reserved** Read as 0; should be written with 0.

SBCU_DBGNTT
SBCU Debug Trapped Master Register

(44_H) **Reset Value: FFFF FFFF$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1								CHNR07	CHNR06	CHNR05	CHNR04	CHNR03	CHNR02	CHNR01	CHNR00
r								rh	rh	rh	rh	rh	rh	rh	rh

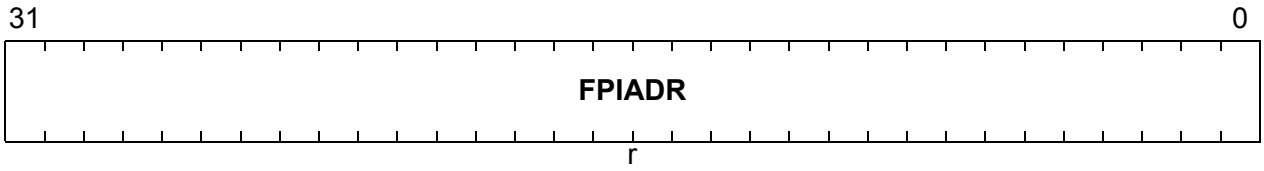
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1								CBL	DMAL	LFI	DMAH	PCP	1		CBH
r								rh	rh	rh	rh	rh	r		rh

Field	Bits	Type	Description
CBH	0	rh	**High-Priority Cerberus FPI Bus Master Status** This bit indicates whether the high-priority Cerberus was FPI Bus master when the break trigger event occurred. 0_B The high-priority Cerberus was not an FPI bus master. 1_B The high-priority Cerberus was 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.
DMAH	4	rh	**High-Priority DMA FPI Bus Master Status** This bit indicates whether the high-priority DMA channels were FPI Bus master when the break trigger event occurred. 0_B The high-priority DMA channels were not an FPI Bus master. 1_B The high-priority DMA channels were an FPI Bus master at the break trigger event. Bits CHNR0y determine the DMA channel number.
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.

Field	Bits	Type	Description
DMAL	6	rh	**Low-Priority DMA FPI Bus Master Status** This bit indicates whether the low-priority DMA channels were FPI Bus master when the break trigger event occurred. 0_B The low-priority DMA channels were not a FPI Bus master. 1_B The low-priority DMA channels were a FPI Bus master. Bits CHNR0y define the DMA channel number.
CBL	7	rh	**Low-Priority Cerberus FPI Bus Master Status** This bit indicates whether the low-priority Cerberus was FPI Bus master when the break trigger event occurred. 0_B The low-priority Cerberus was not an FPI Bus master. 1_B The low-priority Cerberus was 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.
1	[31:24], [15:8], [2:1]	r	**Reserved** Read as 1.

SBCU_DBADRT
SBCU Debug Trapped Address Register

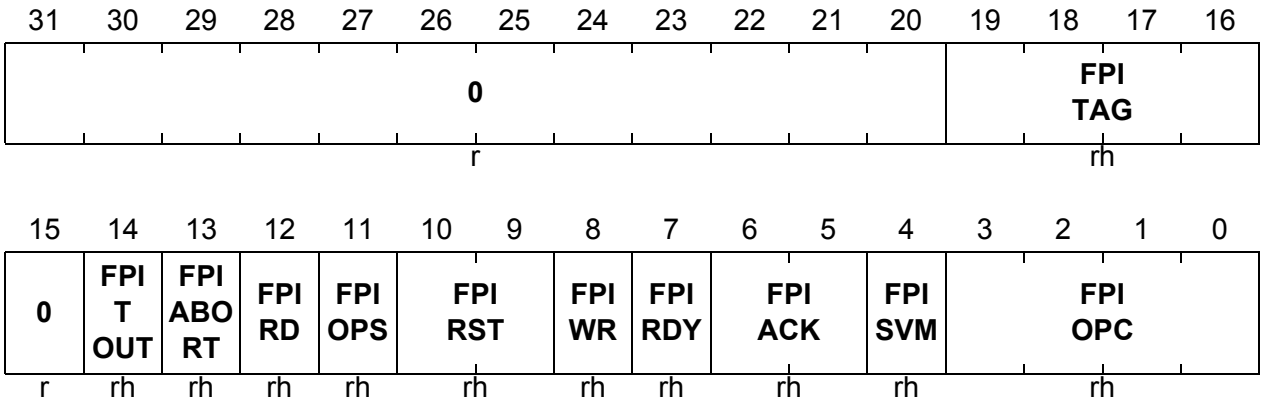
(48_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
FPIADR	[31:0]	r	**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

(4C$_H$) 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

Field	Bits	Type	Description
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
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 Other bit combinations are 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

Field	Bits	Type	Description
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
FPITAG	[19:16]	rh	**FPI Bus Master TAG Status** This bit indicates the master TAG captured from the FPI Bus signal lines when the BCU break trigger event occurred. The master TAG identifies the master of the transfer which generated BCU break trigger event. 0000_B Cerberus Interface (high-priority) 0001_B Peripheral Control Processor (PCP) 0010_B DMA Controller (high-priority channels) 0011_B LFI Bridge 0100_B DMA Controller (low-priority channels) 0101_B Cerberus Interface (low-priority) Other bit combinations are reserved.
0	15, [31:20]	rh	**Reserved** Read as 0; should be written with 0.

6.5.5.4 SBCU Service Request Control Register

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
(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	TOS		0		SRPN							
w	w	rh	rw	r		r		rw							

Field	Bits	Type	Description
SRPN	[7:0]	rw	**Service Request Priority Number**
TOS	[11:10]	r	**Type of Service Control** The SBCU can only be serviced by the CPU.
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], [31:16]	r	**Reserved;** read as 0; should be written with 0.

*Note: Further details on interrupt handling and processing are described in **Chapter 12** of this TC1766 System Units (Vol. 1 of 2) User's Manual.*

7 Program Memory Unit (PMU)

The Program Memory Unit (PMU) shown in **Figure 7-1** contains:

- 1504 Kbyte of Program Flash Memory (PFLASH)
- 32 Kbyte of Data Flash Memory (DFLASH)
- 16 Kbyte of Boot ROM (BROM)
  - 8 Kbyte Boot Code ROM
  - 8 Kbyte Factory Test ROM
- 8 Kbyte of Overlay RAM
- Local Memory Bus Interface
- Emulation Memory Interface

Note: Depending on the version (of the device ordered), 0 or 8 Kbyte of overlay RAM is guaranteed. In this chapter, 8 Kbyte of overlay RAM is described.

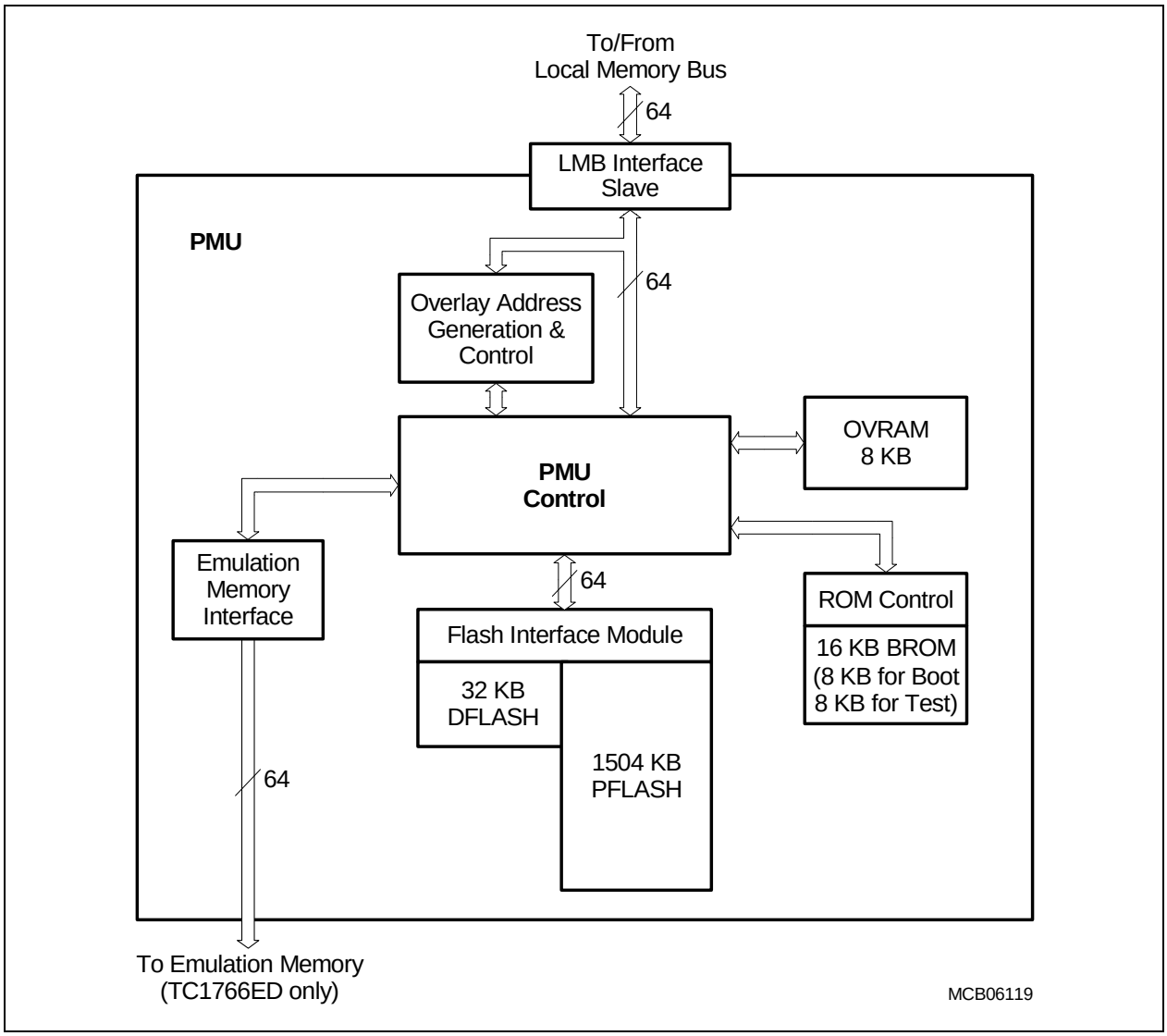


Figure 7-1 PMU Block Diagram

7.1 Boot ROM

The functionality of the 16 Kbyte Boot ROM (BROM) is described in **Chapter 4** (Reset and Boot Operation) at **Page 4-14**.

7.2 Program and Data Flash Memory

The embedded Flash module of TC1766 incorporates a 1504 Kbyte Flash memory for program code or constant data (called PFLASH or program Flash) and a 32 Kbyte Flash memory used for data storage (called DFLASH or data Flash).

Both PFLASH and DFLASH provide error correction of single-bit errors within a 64-bit read double-word, resulting in a very low failure rate. The programming quantity is one page, including 256 byte for the PFLASH and 128 byte for the DFLASH.

The PFLASH is implemented as one Flash bank. The DFLASH is built up by two Flash banks. This bank configuration allows combinations of concurrent Flash operations:

- Reading code or data from PFLASH while one bank of the DFLASH is busy with a program or erase operation.
- Reading data from one bank of the DFLASH while the other bank of the DFLASH is busy with a program or erase operation.
- Programming one bank of the DFLASH while the other bank of the DFLASH is busy with an erase operation and simultaneously reading from PFLASH.

Note: It is not possible to read data from DFLASH while the PFLASH is busy with a program or erase operation.

The embedded Flash module is divided into the following two sub-modules:

- The Flash Interface and Control Module (FIM)
  - Controls the execution of Flash commands (Flash Command State machine FCS)
  - Handles error correction and ECC generation
  - Provides an LMB bus interface to the PMI for instruction accesses and to the DMI module for data accesses.
- The Flash Array Module (FAM)
  - One PFLASH bank of 1504 Kbyte
  - Two DFLASH banks of 16 Kbyte each
  - Control logic that includes assembly buffers and voltage generators, for example.

The FIM and FAM are main parts of the Program Memory Unit (PMU). An overview of the PMU integration into the system architecture is shown in the TC1766 block and bus system diagrams (see **Page 1-9** and **Page 6-1**). A basic diagram of the Flash modules is shown in **Figure 7-2**.

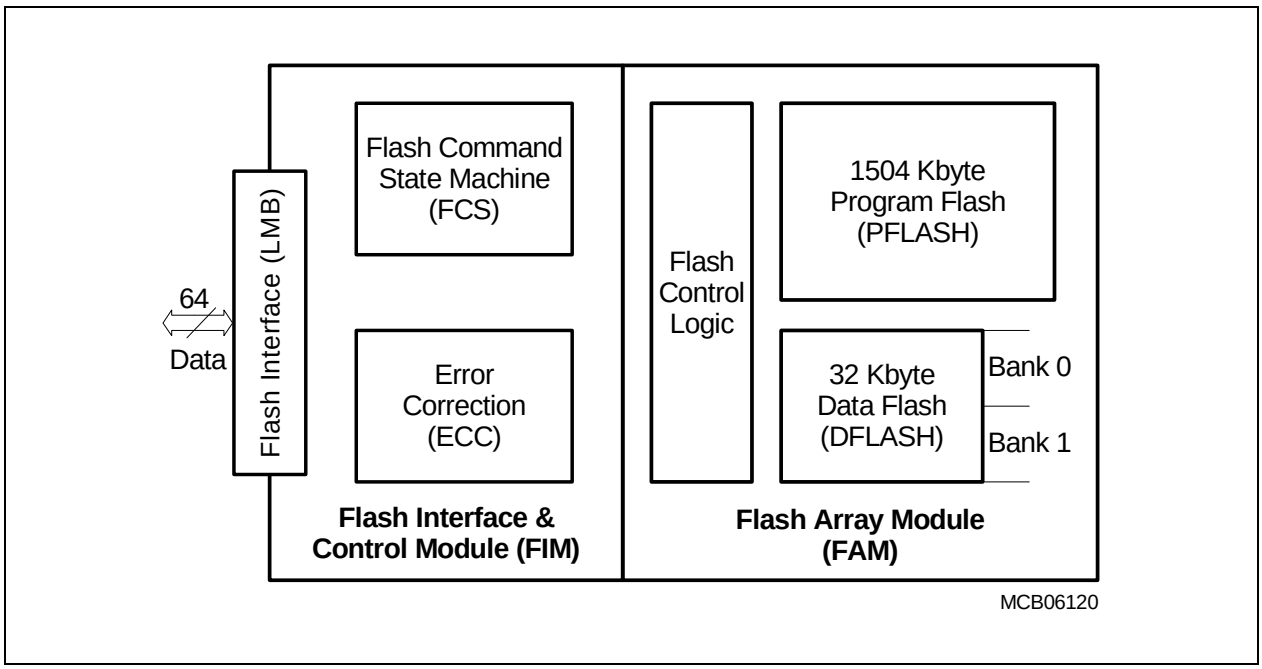


Figure 7-2 Basic Diagram of Flash Module

All Flash operations are initiated by transferring command sequences to the Flash based on the JEDEC standard. The command sequences to the Flash are high-level commands such as “Erase Sector”. State transitions during execution of the commands, such as termination of command execution or errors, are indicated by status flags and maskable interrupts. Command sequences are normally written to the PMU by the CPU but can also be issued by the PCP or the DMA controller.

7.2.1 Program Flash Overview

The on-chip PFLASH memory has a capacity of 1504 Kbyte. The internal structure of the PFLASH memory is based on a sector architecture. For flexible erase, programming, and protection capability, the 1504 Kbyte are divided into 12 sectors of eight 16 Kbyte sectors (one 128 Kbyte physical sector), one 128 Kbyte sector, one 256 Kbyte sector, one 512 Kbyte sector and one 480 Kbyte sector. PFLASH sectors are numbered as PSx with x = 0 to 11. Unless otherwise stated, all references to flash sectors refer to these standard sectors.

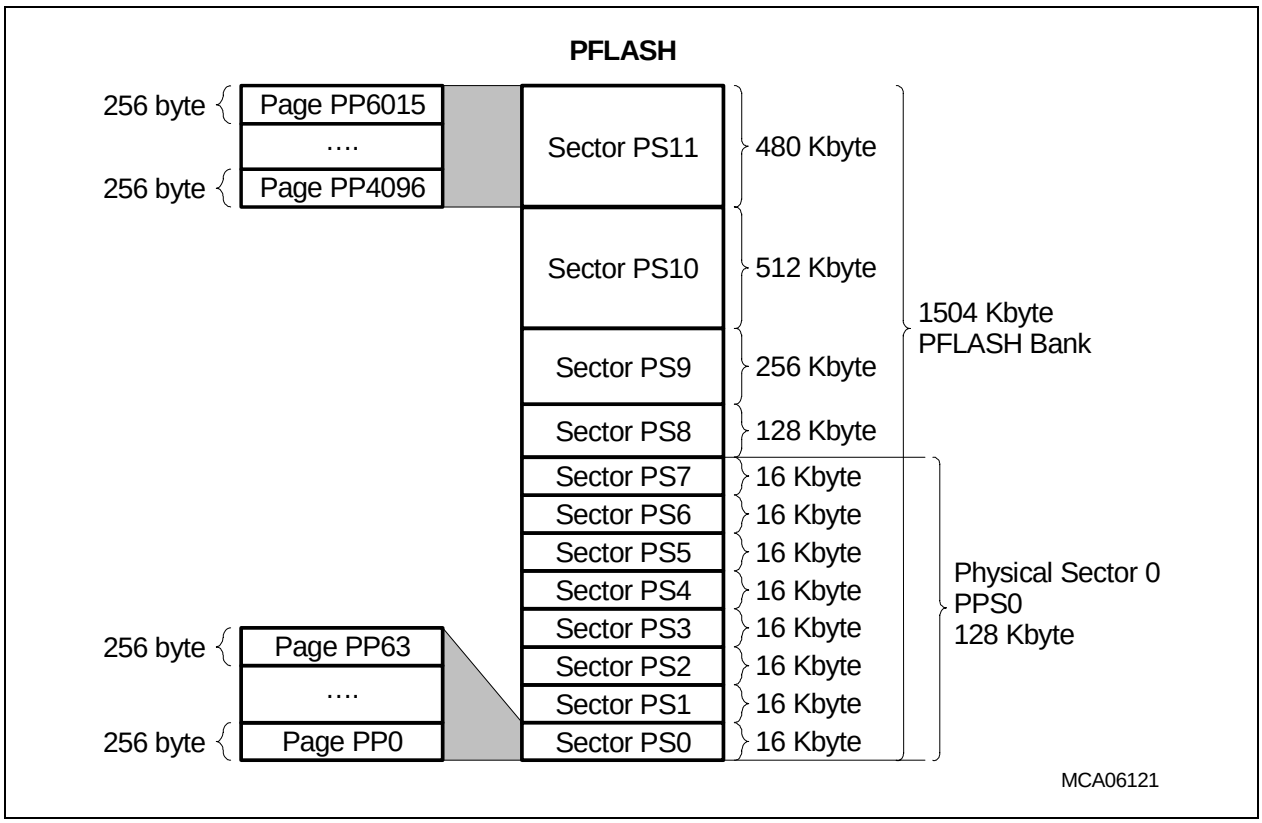


Figure 7-3 PFLASH Structure

The PFLASH operates in paging mode which makes it possible to load 64 words (= 256 byte) into a page assembly buffer with fast CPU accesses before this buffer is programmed into the Flash memory with one Write Page command. Thus, the programming width is always 256 byte. Programming of single words, bytes, or bits is not supported. The 256-byte wide PFLASH pages are numbered with PPy with y = 0 to 6015.

Only one complete sector can be erased. Program and erase operations are initiated and supervised by the Flash Command State Machine (FCS) but its execution is further controlled by control logic in the Flash Array Module (FAM).

The PFLASH array delivers 256-bit wide read data with one read access. In the TC1766, these 256 bits are transferred to the CPU and PMI via the 64-bit wide Local Memory Bus

(LMB) using single-cycle burst transfers with four 64-bit transfers. During such a burst transfer, the next sequential 256-bit read data is prefetched from the PFLASH. Therefore, except a delay of the first initial read data access, sequential burst transfers are provided, supporting the highest possible instruction throughput from PFLASH to the CPU with execution of two 32-bit instructions in one cycle. Also non-cached instruction accesses are burst accesses with up to four double-word transfers.

Each PFLASH sector can be separately locked against erasing and reprogramming. Write operations to a locked and protected sector are only possible if the sector is temporarily unlocked using a dedicated and comfortable password check sequence. Additionally, an OTP (One Time Programmable) function can also be selected for each sector. An OTP sector is locked forever and erasing and reprogramming is no longer possible.

Features of Program Flash

- 1504 Kbyte on-chip program Flash memory
- Usable for instruction code or constant data storage
- 256-byte wide program interface
  – 256 bytes are programmed into PFLASH page in one step/command.
- 256-bit wide read interface
  – Transfer from PFLASH to CPU/PMI by four 64-bit single-cycle burst transfers
- Dynamic correction of single-bit errors during read access
- Detection of double-bit errors
- Fixed sector architecture
  – Eight 16 Kbyte, one 128 Kbyte, one 256 Kbyte, one 512 Kbyte and one 480 Kbyte sectors
  – Each sector separately erasable
  – Each sector separately write-protectable
- Configurable read protection for complete PFLASH with sophisticated read access supervision, combined with write protection for complete PFLASH (protection against “Trojan horse” software)
- Configurable write protection for each sector
  – Each sector separately write-protectable
  – With capability to be re-programmed
  – With capability to be locked forever (OTP)
- Password mechanism for temporary disabling of write and read protection
- On-chip generation of programming voltage
- JEDEC standard based command sequences for PFLASH control
  – Write state machine controls programming and erase operations
  – Status and error reporting by status flags and interrupt
- Margin check for detection of problematic PFLASH bits

Table 7-1 defines the sectors with their sector numbers, sector sizes, and address ranges. The PFLASH contains twelve sectors PS[11:0] with different sizes. One 128 Kbyte physical sector PPS0 is defined for the PFLASH.

Table 7-1 PFLASH Bank, Sector and Page Definitions

Numbering		Size	Cached Address Range	Non-Cached Address Range
PFLASH Bank				
PB		1504 Kbyte	$8000\ 0000_H$ - $8017\ 7FFF_H$	$A000\ 0000_H$ - $A017\ 7FFF_H$
PFLASH Sectors				
PS0	PPS0[1)]	16 Kbyte	$8000\ 0000_H$ - $8000\ 3FFF_H$	$A000\ 0000_H$ - $A000\ 3FFF_H$
PS1		16 Kbyte	$8000\ 4000_H$ - $8000\ 7FFF_H$	$A000\ 4000_H$ - $A000\ 7FFF_H$
PS2		16 Kbyte	$8000\ 8000_H$ - $8000\ BFFF_H$	$A000\ 8000_H$ - $A000\ BFFF_H$
PS3		16 Kbyte	$8000\ C000_H$ - $8000\ FFFF_H$	$A000\ C000_H$ - $A000\ FFFF_H$
PS4		16 Kbyte	$8001\ 0000_H$ - $8001\ 3FFF_H$	$A001\ 0000_H$ - $A001\ 3FFF_H$
PS5		16 Kbyte	$8001\ 4000_H$ - $8001\ 7FFF_H$	$A001\ 4000_H$ - $A001\ 7FFF_H$
PS6		16 Kbyte	$8001\ 8000_H$ - $8001\ BFFF_H$	$A001\ 8000_H$ - $A001\ BFFF_H$
PS7		16 Kbyte	$8001\ C000_H$ - $8001\ FFFF_H$	$A001\ C000_H$ - $A001\ FFFF_H$
PS8		128 Kbyte	$8002\ 0000_H$ - $8003\ FFFF_H$	$A002\ 0000_H$ - $A003\ FFFF_H$
PS9		256 Kbyte	$8004\ 0000_H$ - $8007\ FFFF_H$	$A004\ 0000_H$ - $A007\ FFFF_H$
PS10		512 Kbyte	$8008\ 0000_H$ - $800F\ FFFF_H$	$A008\ 0000_H$ - $A00F\ FFFF_H$
PS11		480 Kbyte	$8010\ 0000_H$ - $8017\ 7FFF_H$	$A010\ 0000_H$ - $A017\ 7FFF_H$
PFLASH Pages				
PP0		256 byte	$8000\ 0000_H$ - $8000\ 00FF_H$	$A000\ 0000_H$ - $A000\ 00FF_H$
PPn[2)]		256 byte	($8000\ 0000_H$ + OFF) - ($8000\ 0000_H$ + OFF + FF_H) (with OFF = n × 100_H)	($A000\ 0000_H$ + OFF) - ($A000\ 0000_H$ + OFF + FF_H) (with OFF = n × 100_H)
PP6015		256 byte	$8017\ FF00_H$ - $8017\ FFFF_H$	$A017\ FF00_H$ - $A017\ FFFF_H$

1) PPS0 is a physical PFLASH sector.

2) n = 0-6015

Note: The sectors PS[7:0] are also called logical sectors. They have a reduced endurance (50) compared to the physical sectors (1000). It is possible to increase the endurance of a logical sector, if after every 50 erase/program cycles the whole

physical sector, which includes the logical sector is refreshed (erased and reprogrammed).

7.2.2 Data Flash Overview

The on-chip DFLASH has a capacity of 32 Kbyte, organized in two independent banks/sectors of 16 Kbyte each, DB0/DS0 and DB1/DS1. The structure with two independent Flash banks makes it possible to execute read accesses to one bank while erasing or programming the other array bank. Erase and programming operations can also be performed simultaneously in the two DFLASH banks, whereby an erase operation of one DFLASH bank is suspended by a programming operation of the other DFLASH bank, and automatically resumed afterwards. Each DFLASH bank can be erased only completely.

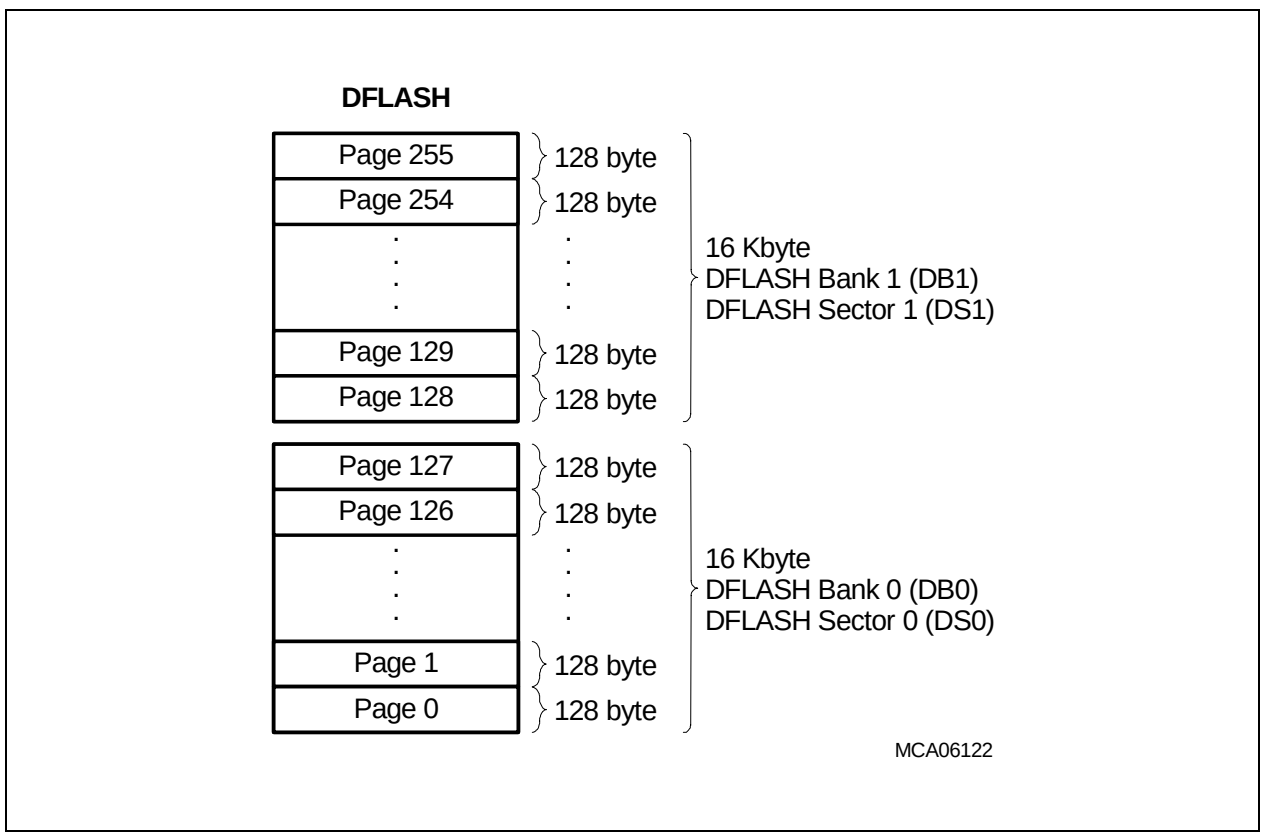


Figure 7-4 DFLASH Structure

For programming of the DFLASH, the data for one page (32 words = 128 bytes) must be loaded into an assembly buffer using fast CPU accesses before this assembly buffer is programmed into one page of the data Flash with one programming command operation. Thus, the programming width of the DFLASH is always 128 bytes. Programming of single words, bytes, or bits is not supported. The DFLASH uses the same JEDEC-standard based command sequences as the program Flash does.

Features of Data Flash

- 32 Kbyte on-chip data Flash memory, organized in two 16 Kbyte banks
- Usable for data storage with EEPROM functionality
- 128 byte of program interface
  – 128 bytes are programmed into one DFLASH page by one step/command
- 64-bit read interface (no burst transfers)
- Dynamic correction of single-bit errors during read access
- Detection of double-bit errors
- Fixed sector architecture
  – Two 16 Kbyte banks/sectors
  – Each sector separately erasable
- Configurable read protection (combined with write protection) for complete DFLASH together with PFLASH read protection
- Password mechanism for temporary disabling of write and read protection
- Erasing/programming of one bank possible while reading data from the other bank
- Programming of one bank possible while erasing the other bank
- On-chip generation of programming voltage
- JEDEC-standard based command sequences for DFLASH control
  – Write state machine controls programming and erase operations
  – Status and error reporting by status flags and interrupt
- Margin check for detection of problematic DFLASH bits

The DFLASH contains two 16 Kbyte sectors DS[1:0] (identical with the two data Flash banks DB[1:0]). DFLASH pages are always 128-byte wide.

Table 7-2 DFLASH Sector and Page Definitions

Numbering		Size	Cached Address Range	Non-Cached Address Range
DFLASH Sectors and Physical Sectors				
DS0	PDS0	16 Kbyte	$8FE0\ 0000_H$ - $8FE0\ 3FFF_H$	$AFE0\ 0000_H$ - $AFE0\ 3FFF_H$
DS1	PDS1	16 Kbyte	$8FE1\ 0000_H$ - $8FE1\ 3FFF_H$	$AFE1\ 0000_H$ - $AFE1\ 3FFF_H$
DFLASH Pages				
DS0	DP0	128 byte	$8FE0\ 0000_H$ - $8FE0\ 007F_H$	$AFE0\ 0000_H$ - $AFE0\ 007F_H$
	DPn[1)]	128 byte	($8FE0\ 0000_H$ + OFF) - ($8FE0\ 0000_H$ + OFF + $7F_H$) (with OFF = n × 80_H)	($AFE0\ 0000_H$ + OFF) - ($AFE0\ 0000_H$ + OFF + $7F_H$) (with OFF = n × 80_H)
	DP127	128 byte	$8FE0\ 3F80_H$ - $8FE0\ 3FFF_H$	$AFE0\ 3F80_H$ - $AFE0\ 3FFF_H$
DS1	DP128	128 byte	$8FE1\ 0000_H$ - $8FE1\ 007F_H$	$AFE1\ 0000_H$ - $AFE1\ 007F_H$
	DPn[1)]	128 byte	$8FE1\ 0000_H$ + OFF - $8FE1\ 0000_H$ + OFF + $7F_H$ (with OFF = n × 80_H)	$AFE1\ 0000_H$ + OFF - $AFE1\ 0000_H$ + OFF + $7F_H$ (with OFF = n × 80_H)
	DP255	128 byte	$8FE1\ 3F80_H$ - $8FE1\ 3FFF_H$	$AFE1\ 3F80_H$ - $AFE1\ 3FFF_H$

1) n = 0-255

7.2.3 User Configuration Blocks Overview

The contents of the three user configuration blocks (UCBm, m = 0-2) determine the user-specific Flash configuration and protection functions such as keywords and sector-specific lock bits for the PFLASH. Each 1 Kbyte configuration block contains four UCB pages.

The UCBs are implemented as a Flash memory. The addressing of the user configuration blocks overlays the non-cached addresses of the first 3 Kbyte of the PFLASH. The UCBs are not readable by the user. A user program can modify the contents of one UCB page only by two commands, the Write User Configuration Page and the Erase User Configuration Block command.

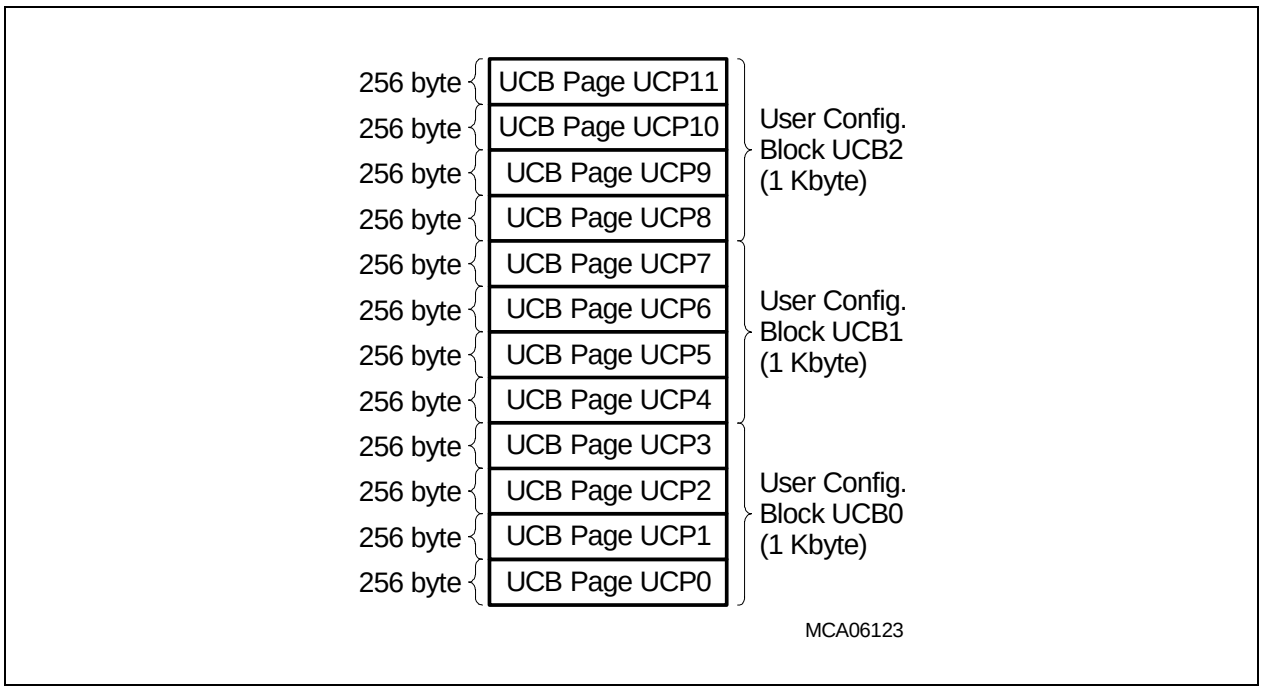


Figure 7-5 UCB Flash Structure

More details on UCBs are described on **Page 7-27**.

Table 7-3 User Configuration Block Definitions

Numbering		Size	Address Range
UCB0	UCP0	256 byte	$A000\ 0000_H$ - $A000\ 00FF_H$
	UCP1	256 byte	$A000\ 0100_H$ - $A000\ 01FF_H$
	UCP2	256 byte	$A000\ 0200_H$ - $A000\ 02FF_H$
	UCP3	256 byte	$A000\ 0300_H$ - $A000\ 03FF_H$
UCB1	UCP4	256 byte	$A000\ 0400_H$ - $A000\ 04FF_H$
	UCP5	256 byte	$A000\ 0500_H$ - $A000\ 05FF_H$
	UCP6	256 byte	$A000\ 0600_H$ - $A000\ 06FF_H$
	UCP7	256 byte	$A000\ 0700_H$ - $A000\ 07FF_H$
UCB2	UCP8	256 byte	$A000\ 0800_H$ - $A000\ 08FF_H$
	UCP9	256 byte	$A000\ 0900_H$ - $A000\ 09FF_H$
	UCP10	256 byte	$A000\ 0A00_H$ - $A000\ 0AFF_H$
	UCP11	256 byte	$A000\ 0B00_H$ - $A000\ 0BFF_H$

7.2.4 Basic Flash Operating Modes

The basic operating modes are Flash-bank specific. Generally, there are two basic operating modes of the Flash banks:

- Read Mode (optionally with Page Mode activated)
- Command Mode

Since the TC1766 Flash array has three autonomous Flash banks, one PFLASH bank and two DFLASH banks, parallel write command execution (programming one bank while erasing the other bank) is supported for the two DFLASH banks, but not for the PFLASH bank and one DFLASH bank.

Flash register accesses are independent of the operating modes and allowed in any state. After any reset operation, the Flash banks are in Read Mode.

The following sections describe the operating modes specifically for the PFLASH. If there is any difference for the DFLASH, additional hints are given.

7.2.4.1 Read Mode

A Flash bank enters or remains in the standard operating mode, the Read Mode.

- After a Reset-to-Read command, if no programming or erase operation is active
- After any completed Erase command
- After a completed Write Page (programming) command
- After every other completed command execution
- During ramp-up time after the deactivation of any reset
- After incorrect address/data values or wrong command sequence
- After incorrect requests (password failure) to program or erase a locked sector
- After an incorrect write access to a read-protected Flash memory

In Read Mode command sequences are allowed. The Read Mode remains active until the last cycle of a command sequence is executed. In case of a write or erase command, Read Mode is terminated at the end of the command sequence.

If Page Mode is active in Read Mode, the page assembly buffer can be loaded (written) with data for the next write page (program) command while Flash read operations occur in parallel. As a special case, it is even allowed to place instructions in the PFLASH, used to load the page assembly buffer or to write command sequences to the DFLASH.

Read accesses from a Flash array are always 64-bit aligned. During these 64-bit wide read accesses from the Flash array, automatic error detection and – if a single-bit error is detected – an error correction is also executed. Bit errors are reported by separate single-bit error flags and double-bit error flags that are located in the FLASH_FSR register. Setting of the single-bit or/and double-bit error flags can generate a Flash interrupt if enabled (see **Page 7-37**). In case of a double-bit error, the CPU is interrupted by a bus error trap per default.

For verify operations, the standard read can be combined with a margin check to find problematic bits (a 0 is read instead of a programmed 1) in advance. The change of margins is controlled via the margin registers (see Page 7-35).

7.2.4.2 Command Mode

Every write operation to the Flash memory space is interpreted as a command cycle, belonging to a command sequence. A command sequence is composed of one to a maximum of six command cycles (write operations) with well defined address and data values. After the last command cycle of a correct command sequence, the Command Mode is entered. The Command Mode remains active during the whole command execution. The state of a command execution is indicated by several status flags in the FSR status register.

Some command sequences that do not affect the Flash banks, for example, the Enter Page Mode command or the Clear Status command, are immediately executed and finished after the last command cycle of the command. For such type of command sequences, the Command Mode is terminated immediately after its execution. After all other command sequences which activate a Flash bank operation such as erasing or programming, the Command Mode and the related status flags remains active until the command is really finished.

Note that all write operations to the Flash memory space are handled by the FCS.

Writing incorrect addresses or data values within a command sequence, or writing them in a wrong sequence, generates a sequence error (FSR.SQER is set) and terminates Command Mode.

If one DFLASH bank is busy with erasing, a programming command sequence for the other DFLASH bank is accepted immediately and the execution of the erase operation is interrupted until the programming operation is terminated (automatic suspend/resume operation).

When accessing a busy PFLASH or DFLASH bank by a data read operation (e.g. by a CPU read in a user program), the read operation is blocked (halted) until the related Flash bank is no longer busy. Therefore, it is recommended to access a PFLASH or DFLASH bank by data read operations only when they are no longer busy (checking the corresponding busy flags first).

The cycle definitions of command sequences are based on the JEDEC standard. The different write cycles of command sequences are not only used for operation definitions such as a sector-erase command, but also as fail-safe and unlock cycles in order to protect the Flash against inadvertent changes.

Note: The write cycles to the Flash belonging to a command sequence are/may be buffered in the DMI store/write buffers. To maintain data coherency (defined sequence of command cycles is mandatory) and to guarantee immediate transfer of the command sequence 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 cycle to a PMU register after the last write cycle of a command sequence.

Note: User code (command sequences) that programs or erases Program Flash should not be executed from internal program flash, but from other internal or external memory (e.g. from SPRAM).

7.2.4.3 Page Mode

With the Enter Page Mode command, Page Mode is entered for one of the three Flash banks and status flag FSR.PFPAGE (for PFLASH) or FSR.DFPAGE (for DFLASH) is set. In Page Mode, the assembly buffer is ready to be filled with data in preparation of a subsequent PFLASH or DFLASH programming operation. The width of the page assembly buffer is 128-byte for DFLASH and 256-byte for PFLASH. Page Mode can be entered only if the related Flash bank is not busy (if it does not execute program or erase operations).

7.2.5 Command Sequence Definitions

Flash commands are executed by writing specific data to dedicated addresses in a well defined command specific sequence. The data to be transmitted within a command sequence must be transmitted right-aligned on data bus lines D[7:0] as byte (exception: 32-bit data, 64-bit data, 32-bit password). Data lines D[31:8] are ignored in a command sequence (indicated in the “Data” columns of the command tables by a “X”). Addresses in command sequence always refer to the non-cached address ranges. The available command sequences are shown in **Table 7-4**.

Table 7-4 Flash Command Overview

Command	Description	Details see
Reset-to-Read	Resetting Flash State Machine to Read Mode	**Page 7-15**
Enter Page Mode	Initiate Page Mode	**Page 7-15**
Load Page Buffer	Loading page assembly buffer with 32-bit or 64-bit data	**Page 7-16**
Write Page	Programming a Flash page with assembly buffer content	**Page 7-18**
Write User Configuration Page	Programming a user configuration page with assembly buffer content	**Page 7-19**
Erase Sector	Erasing a PFLASH or DFLASH sector	**Page 7-20**
Erase User Configuration Block	Erasing a UCB	**Page 7-22**
Disable Write Protection[1)]	Temporarily unlocking write protection	**Page 7-23**
Disable Read Protection	Temporarily unlocking read protection	**Page 7-24**
Resume Protection	Resumption of disabled read or write protection	**Page 7-24**
Clear Status	Resetting status register flags	**Page 7-25**

1) This command is not available for the DFLASH.

Note: During a programming or erasing operation, a minimum CPU clock (f_{CPU}) frequency of 1 MHz must be provided.

7.2.5.1 Reset-to-Read Command

With the one-cycle Reset-to-Read command, the internal command state machine is reset to its initial state. This command can be issued at any time during a command sequence.

Table 7-5 Reset-to-Read Command

Cycle No.	Address	Data
Cycle 1	A000 **5554**$_H$	XXXX XX**F0**$_H$

A running programming or erase operation of a Flash bank is not affected by a Reset-to-Read command and will be continued and finished. All error flags in the Flash Status Register FSR are cleared, and an active Page Mode is aborted. The busy state of the Flash bank (write operation or voltage ramp-up) is not aborted and the busy flags in the Flash status register FSR are not affected. A Reset-to-Read command during a command sequence does not generate a sequence error (FSR.SQER is not set).

7.2.5.2 Enter Page Mode Command

With the one-cycle Enter Page Mode command the Page Mode is entered for a Flash bank, indicating that the page assembly buffer is ready to be filled with data for the related Flash bank as preparation of a subsequent Flash programming operation. The Page Mode can only be assigned to one of the PFLASH/DFLASH banks by executing the address/data information shown in **Table 7-6**. The width of the page assembly buffer is 128-byte for DFLASH and 256-byte for PFLASH. The Page Mode can only be entered if the related Flash bank is not busy (if it does not execute program or erase operations). However, an erase or program operation can be active in another Flash bank than the one that is in the Page Mode.

Table 7-6 Enter Page Mode Command

Cycle No.	PFLASH		DFLASH		
	Address	Data	Bank	Address	Data
Cycle 1	A000 **5554**$_H$	XXXX XX**50**$_H$	DB0	AFE0 **5554**$_H$	XXXX XX**5D**$_H$
			DB1	AFE1 **5554**$_H$	XXXX XX**5D**$_H$

When Page Mode is entered, the pointer to the page assembly buffer is set to its first word location. Its base address has to point to the addressed Flash bank.

An active Page Mode is indicated when bit FSR.PFPAGE (for PFLASH) or FSR.DFPAGE (for DFLASH) is set.

The Page Mode and the Read Mode are allowed in parallel at the same time and in the same Flash memory bank. A new Enter Page Mode command during Page Mode aborts

the actual Page Mode, indicated by sequence error flag FSR.SQER set, and restarts a new page operation. The selected assembly register remains unchanged (not cleared) with a new Enter Page Mode command.

7.2.5.3 Load Page Buffer Command

There are basically two types of Load Page Buffer command:

- Load Page Buffer command with 32-bit data (two-cycle command)
- Load Page Buffer command with 64-bit data (one-cycle command)

Table 7-7 Load Page Buffer Command

Cycle No.	PFLASH		DFLASH		
	Address	**Data**	**Bank**	**Address**	**Data**
32-bit Load Page Buffer Command					
Cycle 1	A000 **55F0**$_H$	32-bit data	DB0	AFE0 **55F0**$_H$	32-bit data
			DB1	AFE1 **55F0**$_H$	
Cycle 2	A000 **55F4**$_H$	32-bit data	DB0	AFE0 **55F4**$_H$	32-bit data
			DB1	AFE1 **55F4**$_H$	
64-bit Load Page Buffer Command					
Cycle 1	A000 **55F0**$_H$	64-bit data	DB0	AFE0 **55F0**$_H$	64-bit data
			DB1	AFE1 **55F0**$_H$	

The Load Page Buffer command loads 32-bit words or 64-bit double-words into the assembly buffer, starting from the lowest to the highest assembly buffer entry. Only one type of the Load Page Buffer commands (either with 32-bit or 64-bit data width) is allowed during one assembly buffer loading sequence. Mixing 64-bit and 32-bit write data width within one assembly buffer filling sequence is not allowed.

The data width for the assembly buffer filling sequence is selected by using instruction(s) with the corresponding data format. A single Load Page Buffer command consists of a store instruction to a specific address. In case of 64-bit data width, the same address A000 55F0$_H$ always has to be used. In case of 32-bit data width, alternating addresses A000 55F0$_H$ and A000 55F4$_H$ must be used.

For loading of the 256-byte wide assembly buffer for PFLASH programming, thirty-two Load Page commands with 64-bit data or sixty-four Load Page Buffer commands with 32-bit data must be issued. For loading of the 128-byte wide assembly buffer for DFLASH programming, sixteen Load Page commands with 64-bit data or thirty-two Load Page Buffer commands with 32-bit data must be issued.

The following example shows a Load Page Buffer command sequence for the DFLASH bank 0 assembly buffer (128 bytes) with 32-bit data width:

1. Write AFE0 $55F0_H$ = word 0 (assembly buffer byte 3-0)
2. Write AFE0 $55F4_H$ = word 1 (assembly buffer byte 7-4)
3. Write AFE0 $55F0_H$ = word 2 (assembly buffer byte 11-8)
4. Write AFE0 $55F4_H$ = word 3 (assembly buffer byte 15-12)

…

1. Write AFE0 $55F0_H$ = word 30 (assembly buffer byte 123-120)
2. Write AFE0 $55F4_H$ = word 31 (assembly buffer byte 127-124)

Sequence errors (bit FSR.SQER set) are generated by a Load Page Buffer command in the following cases:

- A mix of 64-bit and 32-bit write data width within one assembly buffer filling sequence has been detected.
- An assembly buffer overrun condition is detected. The write data causing the overrun is lost.

7.2.5.4 Write Page Command

With the four-cycle Write Page command, the complete contents of the assembly buffer (256 bytes for PFLASH, 128 bytes for DFLASH) are transferred (programmed) into one page of PFLASH or DFLASH. The write address of cycle 4 is the page start address of the page to be programmed with the assembly buffer content. Parameter “n” is the page number PPn for PFLASH or DPn for DFLASH (see also **Table 7-1** and **Table 7-2**).

Table 7-8 Write Page Command

Cycle No.	PFLASH		DFLASH		
	Address	**Data**	**Bank**	**Address**	**Data**
Cycle 1	A000 **5554**$_H$	XXXX XX**AA**$_H$	DB0	AFE0 **5554**$_H$	XXXX XX**AA**$_H$
			DB1	AFE1 **5554**$_H$	
Cycle 2	A000 **AAA8**$_H$	XXXX XX**55**$_H$	DB0	AFE0 **AAA8**$_H$	XXXX XX**55**$_H$
			DB1	AFE1 **AAA8**$_H$	
Cycle 3	A000 **5554**$_H$	XXXX XX**A0**$_H$	DB0	AFE0 **5554**$_H$	XXXX XX**A0**$_H$
			DB1	AFE1 **5554**$_H$	
Cycle 4	A000 0000$_H$ + n × 100$_H$	XXXX XX**AA**$_H$	DB0	AFE0 0000$_H$ + n × 80$_H$	XXXX XX**AA**$_H$
			DB1	AFE1 0000$_H$ + n × 80$_H$	

Address bits [31:16] used in the command cycles of the Write Page command must be the same as those used by the previous Enter Page Mode command. Otherwise, a command sequence error is generated (bit FSR.SQER set) and the execution of the command is aborted.

The Write Page command programs a specific Flash page with the complete content of the assembly buffer. It also programs invalid data of assembler buffer locations that have not been loaded by previous Load Page Buffer commands.

With the last cycle of the Write Page command, Page Mode is terminated and the following status flags are updated:

- FSR.PFPAGE or FSR.DFPAGE is cleared, indicating that the related Flash is no more in Page Mode.
- FSR.PROG is set, indicating that a programming operation is running.
- Either FSR.PBUSY or FSR.D0BUSY or FSR.D1BUSY is set, indicating that the programming operation is running in PFLASH or in DFLASH bank 0 or in DFLASH bank 1.

The page programming algorithm includes a programming quality check that identifies and reprograms weak bits of a Flash page. If the reprogramming of weak bits is unsuccessful, the verify error flag FSR.VER is set.

If read protection is activated or if write protection is enabled for the sector that contains the page to be programmed, Page Mode is terminated, programming operation is not started, and the protection error flag FSR.PROER is set. Write protection must be disabled previously to a programming operation for a write-protected sector by issuing a Disable Read Protection or/and Disable Write Protection command.

7.2.5.5 Write User Configuration Page Command

The four-cycle Write User Configuration Page command is used to transfer (program) the contents of the 256-byte assembly buffer into one page of a user configuration block.

The Write User Configuration Page command sequence is identical with the Write Page command except cycle 3. Cycle 3 transmits different data than the Write Page command. The address transmitted in cycle 4 of the Write User Configuration Page command is the start address of the corresponding user configuration page UCP[11:0] (see **Table 7-3**).

Table 7-9 Write User Configuration Page Command

Cycle No.	Address	Data
Cycle 1	A000 **5554**$_H$	XXXX XX**AA**$_H$
Cycle 2	A000 **AAA8**$_H$	XXXX XX**55**$_H$
Cycle 3	A000 **5554**$_H$	XXXX XX**C0**$_H$
Cycle 4	Start address of UCP to be programmed (A000 0X00$_H$ with X = 0$_H$-B$_H$)	XXXX XX**AA**$_H$

Only if protection is not installed (e.g. for the very first installation of protection), read/write protection need not be disabled. After writing the correct protection confirmation code in the respective user configuration page, the protection is installed and becomes active after the next reset.

If a user configuration page needs to be reprogrammed (not possible for pages within UCB2) and protection is installed (FSR.PROIN = 1), the protection must be temporarily disabled, and the UCB must be erased afterwards by the Erase User Configuration Block command. When no protection is installed (FSR.PROIN = 0), the user configuration page can be programmed without any restriction.

If a Write User Configuration Page command is issued and the start address of the UCP (cycle 4) points to a UCB that is read- or write-protected, the protection error flag FSR.PROER is set and programming operation is not started.

7.2.5.6 Erase Sector Command

The six-cycle Erase Sector command is used to erase a sector in PFLASH or DFLASH. In PFLASH, either one of the 12 sectors PSx (x = 0-11) or the physical sector PPS0 that include the first eight logical sectors can be erased. In DFLASH, one of its two banks DB0 or DB1 (identical with its two DFLASH sectors DS0 and DS1) can be erased.

The write address transmitted in the last cycle of the Erase Sector command is the sector start address of the sector PSx that should be erased.

Table 7-10 Erase Sector Command

Cycle No.	PFLASH			DFLASH		
	Sector	Address	Data	Bank	Address	Data
Cycle 1	PSx[1)]	A000 **5554**$_H$	XXXX XX**AA**$_H$	DB0	AFE0 **5554**$_H$	XXXX XX**AA**$_H$
	PPS0			DB1	AFE1 **5554**$_H$	
Cycle 2	PSx[1)]	A000 **AAA8**$_H$	XXXX XX**55**$_H$	DB0	AFE0 **AAA8**$_H$	XXXX XX**55**$_H$
	PPS0			DB1	AFE1 **AAA8**$_H$	
Cycle 3	PSx[1)]	A000 **5554**$_H$	XXXX XX**80**$_H$	DB0	AFE0 **5554**$_H$	XXXX XX**80**$_H$
	PPS0			DB1	AFE1 **5554**$_H$	
Cycle 4	PSx[1)]	A000 **5554**$_H$	XXXX XX**AA**$_H$	DB0	AFE0 **5554**$_H$	XXXX XX**AA**$_H$
	PPS0			DB1	AFE1 **5554**$_H$	
Cycle 5	PSx[1)]	A000 **AAA8**$_H$	XXXX XX**55**$_H$	DB0	AFE0 **AAA8**$_H$	XXXX XX**55**$_H$
	PPS0			DB1	AFE1 **AAA8**$_H$	
Cycle 6	PSx[1)]	A0**XX XXXX**$_H$[2)]	XXXX XX**30**$_H$	DB0	AFE0 0000$_H$	XXXX XX**40**$_H$
	PPS0	A0**00 0000**$_H$	XXXX XX**40**$_H$	DB1	**AFE1 0000**$_H$	

1) "x" is the number of the PFLASH sector PSx that is defined in Table 7-1 (x = 0-11).

2) This is the complete 32-bit non-cached start address of the PFLASH sector PSx (x = 0-11) to be erased.

With the last cycle of the Erase Sector command, Command Mode is entered and the following status flags are updated:

- FSR.ERASE is set, indicating that an erase operation is running.
- Either FSR.PBUSY or FSR.D0BUSY or FSR.D1BUSY is set, indicating that an erase operation is running in PFLASH or in DFLASH bank 0 or in DFLASH bank 1.

The start of sector erase operation is delayed if the Erase Sector command was issued to one DFlash bank while the other DFlash bank is busy with a program operation.

Read accesses to a data Flash bank that is currently not erased or programmed are possible while the other DFlash bank is erased or programmed. A read access to a busy (erasing or programming) Flash bank is allowed but delayed until the corresponding Flash bank is no longer busy. Note that in this case the read path (LMB) is completely locked. Therefore, reading from a Flash bank in erase or program state should be avoided by polling the busy flags in the FSR register.

If the Erase Sector operation is used to erase a physical sector of the program Flash (PPS0), this operation is executed only if none of the eight sectors within the related physical sector is write-protected, and if no read protection is installed, or if Flash protection is disabled.

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 reprogramming of over-erased bits is unsuccessful, the verify error flag FSR.VER is set.

If read protection and/or write protection for the sector to be erased is enabled, or if one or more of the sectors within the physical sector to be erased is write-protected, the protection error flag FSR.PROER is set, Command Mode is not entered, and the erase operation is not started.

7.2.5.7 Erase User Configuration Block Command

This six-cycle Erase User Configuration Block command is used to erase a UCB.

The Erase User Configuration Block command sequence is identical with the Erase Sector command sequence except cycle 6. Cycle 6 of the Erase User Configuration Block command provides the base address of the UCB to be erased.

Table 7-11 Erase User Configuration Block Command

Cycle No.	Address		Data
Cycle 1	A000 **5554**$_H$		XXXX XX**AA**$_H$
Cycle 2	A000 **AAA8**$_H$		XXXX XX**55**$_H$
Cycle 3	A000 **5554**$_H$		XXXX XX**80**$_H$
Cycle 4	A000 **5554**$_H$		XXXX XX**AA**$_H$
Cycle 5	A000 **AAA8**$_H$		XXXX XX**55**$_H$
Cycle 6	UCB0	A000 0000$_H$	XXXX XX**C0**$_H$
	UCB1	A000 0400$_H$	
	UCB2	A000 0800$_H$	

With the last cycle of the Erase User Configuration command, Command Mode is entered and the following status flags are updated:

- FSR.ERASE is set, indicating that an erase operation is running.
- FSR.PBUSY is set, indicating that the erase operation is running in the UCB.

This command can only be executed after disabling of the user's write-protection (user 0 for UCB0 or user 1 for UCB1 erasing operation respectively) and/or after disabling of read protection (user 0 for UCB0 erasing operation). If protection is not disabled when the Erase User Configuration Block command is received, the protection error flag FSR.PROER is set, the Command Mode is not entered, and the erase operation is not started.

After an Erase User Configuration Block command, a new protection configuration (including keywords and protection confirmation code) can be written to the pages of the user configuration block. But note that the user configuration blocks UCB0 and UCB1 can be modified only up to 4 times during TC1766 device lifetime. UCB2 can be programmed only once.

Note: UCB2 is only One Time-Programmable (OTP). If sector write protection is installed and confirmed in UCB2, this block can never be erased again. UCB2 can only be erased before the confirmation of OTP write protection.

7.2.5.8 Disable Write Protection Command

The six-cycle Disable Write Protection command temporarily disables the write protection of all protected PFLASH sectors as defined for user 0 (indicated in register PROCON0) or user 1 (indicated in register PROCON1).
The Disable Write Protection command is not accepted for UCB2 because UCB2 is a OTP write-protected UCB.

Table 7-12 Disable Write Protection Command

Cycle No.	Address	Data
Cycle 1	A000 **5554**$_H$	XXXX XX**AA**$_H$
Cycle 2	A000 **AAA8**$_H$	XXXX XX**55**$_H$
Cycle 3	A000 **553C**$_H$	XXXX XX**00**$_H$ (user 0, UCB0) or XXXX XX**01**$_H$ (user 1, UCB1)
Cycle 4	A000 **AAA8**$_H$	First 32-bit password
Cycle 5	A000 **AAA8**$_H$	Second 32-bit password
Cycle 6	A000 **5558**$_H$	XXXX XX**05**$_H$

The Disable Write Protection command is a protected command sequence, meaning that two passwords (data in cycle 4 and 5) must be issued within the command. The first and second 32-bit passwords are internally compared with the first and second 32-bit keyword as defined in the related UCB0 or UCB1 (see **Table 7-16**). If one or both passwords are not identical to their related keywords, the sectors remain protected and the protection error flag FSR.PROER is set. In this case, a new Disable Write Protection command is only accepted after the next reset operation.

After the correct execution of the Disable Write Protection command, all protected sectors as defined for UCB0 or UCB1 are unlocked (if no read protection is additionally installed and active) and flag FSR.WPRODIS0 or FSR.WPRODIS1 is set. Erase and write operations to temporarily unlocked sectors are now possible, until

- A Resume Protection command is executed, or
- The next reset operation (hardware or software reset) is executed.

7.2.5.9 Disable Read Protection Command

The six-cycle Disable Read Protection command temporarily disables an installed Flash read protection.

Table 7-13 Disable Read Protection Command

Cycle No.	Address	Data
Cycle 1	A000 **5554**$_H$	XXXX XX**AA**$_H$
Cycle 2	A000 **AAA8**$_H$	XXXX XX**55**$_H$
Cycle 3	A000 **553C**$_H$	XXXX XX**00**$_H$
Cycle 4	A000 **AAA8**$_H$	First 32-bit password
Cycle 5	A000 **AAA8**$_H$	Second 32-bit password
Cycle 6	A000 **5558**$_H$	XXXX XX**08**$_H$

The Disable Read Protection command is a protected command sequence, meaning that two passwords (data in cycle 4 and 5) must be issued within the command. The first and second 32-bit passwords are internally compared to the first and second 32-bit keyword as defined in the related UCB0 (see **Table 7-16**).

If one or both passwords are not identical to their related keywords, the sectors remain protected and the protection error flag FSR.PROER is set. In this case, a new Disable Read Protection command is only accepted after the next reset operation.

After the correct execution of this command, the PFLASH and both DFLASH banks are temporarily unlocked and flag FSR.RPRODIS is set. Read, erase, and write operations to all unlocked (not separately write-protected) sectors are now possible, until

- A Resume Protection command is executed, or
- The next reset operation (hardware or software reset) is executed.

7.2.5.10 Resume Protection Command

With the one-cycle Resume Protection Command, a temporarily unlocked write protection and/or read protection is terminated and the protection state is resumed. This command resumes both kinds of temporarily disabled protection, write and read protection.

Table 7-14 Resume Protection Command

Cycle No.	Address	Data
Cycle 1	A000 **5554**$_H$	XXXX XX**5E**$_H$

7.2.5.11 Clear Status Command

The one-cycle Clear Status command clears the following flags of the Flash status register FSR:

- Error flags:
  PFOPER, DFOPER, SQER, PROER, PFSBER, DFSBER, PFDBER, DFDBER and VER
- Status flags:
  PROG, ERASE

The one-cycle Clear Status command is accepted only in Read Mode. Otherwise a command sequence error occurs (FSR.SQER is set). Read Mode is entered if Flash is not busy and after every error detection and indication change in FSR.

Table 7-15 Clear Status Command

Cycle No.	Address	Data
Cycle 1	A000 **5554**$_H$	XXXX XX**F5**$_H$

7.2.6 Data Flash and EEPROM Emulation

The 32 Kbyte DFLASH is able to support an emulation of an EEPROM. This EEPROM emulation is fully based on a software administration by a user program. The only hardware feature that supports EEPROM emulation in TC1766 is the ability to program a page in one DFLASH bank while the other DFLASH bank is erased, and to read data from one bank while the other bank is busy with programming or erasing operation.

The main difference between a Flash memory and an EEPROM is its endurance. For EEPROMs, the endurance is typically 120.000 write/erase cycles. The typical endurance of a Flash memory is in the range of 1.000 to 15.000 write/erase cycles. The following example discusses the basic principles of the EEPROM emulation as supported in the TC1766.

Example: 4 Kbyte EEPROM emulation, using the complete 32 Kbyte DFLASH:

The DFLASH is logically divided into eight 4 Kbyte regions that operate as a circular buffer memory. Each of these regions has 32 pages, the smallest DFLASH entity that can be programmed by one programming/write command. At a time, one of the eight 4 Kbyte regions is always regarded as the active EEPROM region. The active EEPROM region is simply identified by the used region with the highest address within active DFLASH bank.

The active EEPROM region is held as a mirror memory 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 4 Kbyte region within the DFLASH circular buffer becomes the active EEPROM region. The “old” DFLASH bank can be erased when the active EEPROM region is switched to the “new” DFLASH bank.

As a result of a continuously changing assignment of the active EEPROM region in a circular buffer, the DFLASH memory cells of one EEPROM region can be erased/programmed 8 times as often as one physical 4 Kbyte region, resulting in an 8-fold endurance for the EEPROM region. Additionally, a reduced retention is assumed for EEPROM data. Thus, for the above described example with 4 Kbyte regions and with a retention of 5 years, the endurance for an emulated EEPROM region is increased to 120.000 write/erase cycles.

For emulation of complexer 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.

For marking of invalid wordlines (two consecutive pages) it is allowed to overprogram a page with all-one data. This is the only kind of twofold programming of a page which is allowed and can only be performed in DFLASH.

7.2.7 Read and Write Protection

In the TC1766, three different types of protection are possible:

- Read protection for PFLASH and DFLASH
  (always implies a global write protection also for PFLASH and DFLASH)
- Sector write protection for PFLASH
- Sector OTP write protection for PFLASH

In general, three protection configurations, assigned to so-called “users”, are defined in UCBs.

- **User 0 = User Configuration Block 0**:
  This is the master user. It is able to install read protection for the whole PFLASH and DFLASH. It can also install sector write protection for the PFLASH sectors. User 0 controls its protection configuration and its keywords via UCB0.
- **User 1 = User Configuration Block 1**:
  User 1 is able to install sector write protection for the PFLASH sectors, with lower priority than user 0. User 1 controls its protection configuration and its keywords via UCB1.
- **User 2 = User Configuration Block 2**:
  User 2 is able to install sector OTP (one time programmable) write protection for the PFLASH sectors. OTP write-protected sectors are locked forever, and are never re-programmable.

If read or write protection is installed and activated once, changing the read/write protection configuration parameters in the UCBs is password-protected and can only be executed if the passwords are known.

Note: If any PFLASH sector is locked forever (OTP installed), PFLASH and DFLASH related testing investigations for Failure Analysis Requests (FARs) are no more possible.

7.2.7.1 User Configuration Block Definitions

The three 1 Kbyte user configuration blocks UCBx (x = 0-2) contain four 256 byte pages each. The contents of the three user configuration blocks determine the read, write, and OTP protection configuration as assigned for user 0, user 1, and user 2.

The UCBs are Flash memory locations that can be programmed page-wise with the Write User Configuration Page command, and erased block-wise by the Erase User Configuration Block command. User configuration blocks UCB0 and UCB1 can be programmed/erased during TC1766 device lifetime only up to 4 times. UCB2 can be programmed only once.

Table 7-16 defines the contents of the three UCBs.

Table 7-16 Layout of User Configuration Blocks

UCB	Page	Address	byte(s)	Content
UCB0	UCP0	A000 0000$_H$	[1:0]	Protection configuration bits (content as defined for register PROCON0[15:0])
		A000 0008$_H$	[9:8]	Copy of bytes [1:0]
		A000 0010$_H$	[19:16]	First 32-bit keyword of user 0
		A000 0014$_H$	[23:20]	Second 32-bit keyword of user 0
		A000 0018$_H$	[27:24]	Copy of first 32-bit keyword of user 0
		A000 001C$_H$	[31:28]	Copy of second 32-bit keyword of user 0
		–	others	Must be programmed to 00$_H$
	UCP1	–	all	This page is reserved for future purposes; must be programmed to 00$_H$
	UCP2	A000 0200$_H$	[3:0]	32-bit confirmation code: 8AFE15C3$_H$
		A000 0208$_H$	[11:8]	Copy of 32-bit confirmation code
		–	others	Must be programmed to 00$_H$
	UCP3	–	all	This page is reserved for future purposes; must be programmed to 00$_H$
UCB1	UCP4	A000 0400$_H$	[1:0]	Protection configuration bits (content as defined for register PROCON1[15:0])
		A000 0408$_H$	[9:8]	Copy of bytes [1:0]
		A000 0410$_H$	[19:16]	First 32-bit keyword of user 1
		A000 0414$_H$	[23:20]	Second 32-bit keyword of user 1
		A000 0418$_H$	[27:24]	Copy of first 32-bit keyword of user 1
		A000 041C$_H$	[31:28]	Copy of second 32-bit keyword of user 1
		–	others	Must be programmed to 00$_H$
	UCP5	–	all	This page is reserved for future purposes; must be programmed to 00$_H$
	UCP6	A000 0600$_H$	[3:0]	32-bit confirmation code: 8AFE15C3$_H$
		A000 0608$_H$	[11:8]	Copy of 32-bit confirmation code
		–	others	Must be programmed to 00$_H$
	UCP7	–	all	This page is reserved for future purposes; must be programmed to 00$_H$

Table 7-16 Layout of User Configuration Blocks (cont'd)

UCB	Page	Address	byte(s)	Content
UCB2	UCP8	A000 0800$_H$	[1:0]	Protection configuration bits (content as defined for register PROCON2[15:0])
		A000 0808$_H$	[9:8]	Copy of bytes [1:0]
		–	others	Must be programmed to 00$_H$
	UCP9	–	all	This page is reserved for future purposes; must be programmed to 00$_H$
	UCP10	A000 0600$_H$	[3:0]	32-bit confirmation code: 8AFE15C3$_H$
		A000 0608$_H$	[11:8]	Copy of 32-bit confirmation code
		–	others	Must be programmed to 00$_H$
	UCP11	–	all	This page is reserved for future purposes; must be programmed to 00$_H$

The protection configuration bits that are located in the first two bytes of the UCBs determine the requested protection configuration as it is defined for bits [15:0] of the Flash Protection Configuration registers PROCON0, PROCON1, and PROCON2. When a read/write protection has been installed for a user configuration block, the content of its first two bytes is copied into the corresponding Flash Protection Configuration register.

UCB Configuration, Confirmation and Activation

In order to set up a UCB correctly, several steps must be done to avoid incorrect and inoperable UCB contents and, as a result, irreparable read/write protection.

There are three main tasks to execute for UCB setup:

1. **Configuration of a UCB**
   This step includes the programming of the first page of a UCB by executing a User Configuration Page command. This first page determines the protection type and the two 32-bit keywords. Unused bytes in the first page of the UCB must be programmed with 00$_H$.
2. **Confirmation of the Keywords**
   The 32-bit confirmation code word, which is located in the third page of a UCB, should be programmed only after a check of the correct programming of the two 32-bit keywords. Reason: wrong keywords in a UCB can never be retrieved (because UCBs are not readable), and a confirmed read or write protection cannot be disabled and changed anymore when the password check (see **Page 7-33**) always fails.
   The check for correct keywords in a UCB requires the execution of a reset operation (e.g. software reset) after the configuration has been setup as described under point 1. After the reset, the protection is not fully activated because the confirmation

code in the UCB is still not valid but it is already configured. After issuing a Disable Read Protection or Disable Write Protection command (depending on the configured protection type) with the passwords, the status flag FSR.PROER indicates, whether password checking was successful.
If PROER = 0 after the Disable Read Protection or Disable Write Protection command, the password check was successful, i.e. that the keywords in the UCB are identical with the passwords that have been transmitted with the command, then the 32-bit confirmation code word must be programmed into the third page of the UCB by the Write User Configuration Page command. Unused bytes of the third page of the UCB should be programmed with 00_H. After that operation, the selected protection is defined to be “installed”.
If PROER = 1 after the Disable Read Protection or Disable Write Protection command, the password check was unsuccessful. In this case (and if the correct passwords are not known), the related UCB has to be erased again and the UCB setup must be repeated as described under point 1.

3. **Activation of an Installed Protection**
   When a read or write protection has been installed, it can only be activated by executing any reset operation. After this reset operation, an installed protection becomes “active”.

7.2.7.2 Write and OTP Protection for PFLASH

Write protection is a feature that must be installed by the user of the TC1766 device. In TC1766 delivery state, no write protection is installed meaning that the user configuration blocks are in erased state. If sector write protection is active for a PFLASH sector, erasing and programming of this sector is only possible if the corresponding UCB keywords are known.

OTP write protection can be installed and enabled for a PFLASH sector only once after the TC1766 delivery state. Write protection configuration for a PFLASH sector can be modified by erasing and re-programming of the related UCB.

The sector write protection configuration must be initially programmed into one of the three UCBs by using the Write User Configuration Page command. With this command, the user defines the PFLASH sector(s) to be write protected and two 32-bit keywords which are required to temporarily disable an active sector write protection and also required to change an already installed write protection configuration. Erasing and reprogramming of UCB0 or UCB1 can be performed up to 4 times during TC1766 device lifetime.

As described above on **Page 7-29**, sector write protection remains active as long as no Disable Write Protection command is issued. Within this command sequence, the user has to identify itself by its passwords and its user number (see command sequence definition). After the Disable Write Protection command, sector write protection is temporarily disabled for all sectors that belong to the user. Thereby, disabling of write

protection is hierarchically controlled. This means, if user 0 (assigned to UCB0) disables write protection for his sector(s), write protection for user 1 (assigned to UCB1) is also disabled but not vice versa (user 1 can only disable his own protected sectors).

Note: Sector specific write protection may be combined with read protection. In this case, after execution of the Disable Sector Write Protection command the protected sectors are only unlocked if read protection is also disabled.

Resumption of the temporarily disabled write protection (and read protection) is done by sending the Resume Protection command or by executing a reset operation. For UCB2, disabling write protection and thus re-programming is not possible.

The configuration of an installed write protection is indicated by:

- Three status flags FSR.WPROINx (x = 0-2) that indicate whether sector write protection is installed for UCBx or not
- Status flag FSR:PROIN = 1; this bit is set coincidently with FSR:WPROINx
- Status flags SnL (n = 0-11) in the three Protection Configuration registers PROCONx (x = 0-2) that indicate which Flash sectors are write-protected by UCBx
- The state of a write protection (enabled or temporarily disabled) is indicated by bits FSR.WPRODIS0 (for UCB0) and FSR.WPRODIS1 (for UCB1)

After the execution of an Erase User Configuration Block command, which requires the prior disabling of an active write protection by the Disable Write Protection command, all keywords and all protection parameters in the UCB are erased. Thus, the UCB is totally unprotected until it is re-programmed. The only exception is UCB2 which can never be erased after installation of OTP write protection.

If global write protection is additionally installed (implicitly with an installed read protection), a Disable Read Protection command must be issued before the write protection configuration parameters in UCB0 can be modified by user 0.

Note: All PFLASH sectors can be separately write-protected for all three users. DFLASH sectors cannot be separately write-protected (only generally via Read Protection).

7.2.7.3 Read Protection for PFLASH and DFLASH

When read protection is active, read accesses from PFLASH and DFLASH memory locations are generally disabled, if code execution is not started from internal Flash after reset. Additionally, a global write protection is always active for a read-protected PFLASH and DFLASH. This feature supports a protection against trojan horse programs. Note that read protection cannot be activated separately for PFLASH and DFLASH.

Read protection is installed only via UCB0. While programming the UCB0, the highest bit of its second byte (corresponds to bit PROCON0.RPRO, see **Page 7-58**) must be set.

After the configuration and confirmation of UCB0 (see **Page 7-29**), the read protection is installed but still not active. After the following reset operation, the installed read protection becomes really active. It remains active as long as no Disable Read Protection command is issued. This command is password-protected and the user must provide the two passwords for temporary disabling of the read protection. This mechanism ensures that an installed read protection configuration can only be changed (e.g. disabled) by a user who has knowledge about the two keywords that have been initially programmed into the UCB. After the Disable Read Protection command, the read protection configuration as defined in UCB0 can be changed (after erasing UCB0), if not coincidently sector write protection is installed by the user 0 (in this case, sector write protection must also be disabled). A temporarily disabled read protection can be re-enabled by sending the Resume Protection command, or by executing a reset operation.

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.

The status of a correctly installed read protection is indicated by:

- Status flag FSR.RPROIN = 1; this bit becomes updated after a reset operation (when the BootROM code has been left) and when read protection has been configured.
- Status flag PROCON0.RPRO = 1; indicates an installed (and confirmed) read protection as programmed in the protection configuration bits of UCB0.
- Read protection active flag FCON.RPA; indicates the state of an installed read protection (active or temporarily disabled).

There are also two Flash access disable bits in register FCON, Disable Code Fetch (DCF) and Disable Data Fetch (DDF), which control the Flash access during an active read protection. During execution of the Boot ROM program (see **Page 4-16**) with read protection, the following two locations for program code execution can be selected:

- Case 1: code execution from internal PFLASH
- Case 2: code execution from internal program RAM after execution of a bootstrap loader

With a reset operation, the FCON register bits DCF and DDF are set. When starting code execution from internal PFLASH, these flags are cleared by the BootROM program

before BootROM exit. This means that code execution from PFLASH and data read accesses from PFLASH or DFLASH are generally allowed. The program code, which executes the data read accesses from Flash, can be located in any program memory.

When starting from internal program RAM (case 2), flags DCF and DDF remain set at BootROM exit. This means that code execution from PFLASH and data read access from PFLASH or DFLASH is disabled. In this case, code or data accesses from PFLASH or DFLASH are only possible while read protection is temporarily disabled by the password protected Disable Read Protection command (FCON.RPA is cleared). In this disable state it is also possible to clear the DCF/DDF flags of register FCON.

Flash data accesses from dedicated bus masters other than the CPU/DMI can be disabled separately with FCON register bits DDFDBG, DDFDMA, and DDFPCP of register FCON. When such a bit is set, the corresponding bus master (Debug system, DMA controller, or PCP) is not allowed to access PFLASH or DFLASH memory. When these bits are set once, they can only be cleared again when read protection is not selected at all (inactive), or temporarily disabled.

Note: The debug interface is disabled after Boot ROM exit with read protection and program start in internal Flash.

7.2.7.4 Password Check Control

The Disable Write Protection command and the Disable Read Protection command provide a protected command sequence, meaning that two 32-bit passwords must be issued within the command. Both commands are executed only if the two passwords are identical with the two keywords that are stored in the corresponding user configuration block. If one or both passwords are not identical to their related keywords, the protected sectors remain in the locked state (read- and/or write-protected) and the protection error flag FSR.PROER is set. In this case, a new Disable Write Protection command or a Disable Read Protection command is only accepted after the next TC1766 reset operation.

Note that the Disable Write Protection command can be applied for user 0 (UCB0) or user 1 (UCB1). The Disable Read Protection command can be applied only for user 0 (UCB0).

7.2.8 Error Correction and Margin Control

Both PFLASH and DFLASH provide error correction of single-bit errors within a 64-bit read double-word, resulting in a very low failure rate. The margin control feature of the TC1766 Flash module makes it possible to change the sensing levels of the sense amplifiers of the Flash array bit lines for read operations.

7.2.8.1 Dynamic Error Correction

Error detection and correction are controlled in the Flash module by using a SECDED algorithm. This algorithm calculates an 8-bit error correction code (ECC) for every 64-bit data portion in PFLASH and DFLASH. This 8-bit ECC is generated during write operations to the assembly buffer and programmed into the Flash array together with the assembly buffer data during the execution of the Write Page command. With every 64-bit read access from PFLASH and DFLASH, the associated 8-bit ECC is fetched and checked whether it is correct or not.

The ECC is defined such that an 8-bit ECC of 00_H is generated for 64-bit data portions with all bits set at 0. Therefore, after an erase operation all Flash locations including the ECCs are at logic 0, meaning that the Flash is erased with correct ECC information. On the other hand, an ECC code of FF_H is generated when all bits of a 64-bit data portion are programmed with a 1.

In general, the TC1766 Flash module supports the following error detection and correction functionality for PFLASH and DFLASH read accesses:

- Single-bit error detection within 64-bit read data with on-the-fly correction
  - Status flag indication for PFLASH (FSR.PFSBER) and DFLASH (FSR.DFSBER) single-bit errors
  - Optional single-bit error Flash interrupt generation for PFLASH and DFLASH
- Double-bit error detection (no correction)
  - Status flag indication for PFLASH (FSR.PFDBER) and DFLASH (FSR.DFDBER) double-bit errors
  - Optional double-bit error Flash interrupt generation for PFLASH and DFLASH
  - In double-bit error case, a trap will occur. This trap can be disabled e.g. for margin check purposes.

A single-bit or a double-bit error may also be caused by a disturbed ECC with correct 64-bit data, or by a wrong selection of internal Flash access time with wait states.

Note: After the detection of a single-bit error it is generally recommended to execute a margin check procedure (described in the next section) and, as a result of this check, to erase the violated sector and reprogram it completely new (to avoid double-bit errors).

7.2.8.2 Margin Check Control

The margin control feature of the TC1766 Flash module makes it possible to change the sensing levels of the sense amplifiers of the Flash array bit lines for read operations. With this feature, problematic Flash array bits can be found in advance, before they convert to stable erratic bits. Margin control is supported separately for PFLASH and DFLASH.

Two Margin Control Registers, MARP for PFLASH and MARD for DFLASH, are provided to adapt the margin levels. Two margin levels, standard or high margin level, are selectable for 0 or 1 level detection. Standard margin levels are selected as default after reset.

Since problematic Flash array bits always change their value from a valid 1 to an invalid 0 level, these bits can be identified and corrected in advance by executing the following steps:

1. Checking the high-level margin by setting bit field MARGIN1 in the MARP or MARD register to 01_B (high margin selected) and MARGIN0 to 00_B (standard margin selected).
2. Reading related Flash memory locations while observing the single-bit and double-bit Flash interrupts; if an error occurs, an erase and reprogramming of the corresponding sector should be performed.

A zero check (analogous procedure as shown in step 1 and 2 for programmed cells) is not normally necessary, because the energetically stabilized state of Flash cells is close to the zero-state.

Only one high margin level (either for PFlash or DFlash) is allowed at a time. During erase or programming operations only the standard margin selections are allowed.

Attention: To increase the security and to inhibit unintended write accesses to the MARP register, the standard "ENDINIT" protection feature (including watchdog password access control) should be used for write accesses to the MARP register (for PFLASH). Using this mechanism, it is possible to change the read margins of the PFLASH only before End-of-initialization or after ENDINIT with valid password access to register WDT_CON0. The MARD register for Data Flash is not specially protected.

Note: After changing a margin level in the MARP or MARD registers, a wait time of 10 µs must elapse before Flash read operations with the modified margin can be executed.

7.2.9 Flash Interrupt Generation and Control

One interrupt request can be issued by the Flash module. The related interrupt control register is located in the DMA controller.

The Flash interrupt can be issued when any of the following events occur:

- End-of-busy: programming or erase operation of Flash bank finished
- Command sequence error
- Protection error
- Single-bit error in PFLASH
- Single-bit error in DFLASH
- Double-bit error in PFLASH
- Double-bit error in DFLASH

The source of the Flash Interrupt is indicated in the Flash Status Register FSR by the corresponding status (PROG and ERASE) and error flags. Every interrupt source is disabled after reset, and can be enabled individually via dedicated enable bits in the Flash Configuration Register FCON.

The end-of-busy interrupt becomes active whenever a programming or erasing operation is finished. At this event, one of the three FSR busy flags PBUSY, D0BUSY, or D1BUSY is cleared from 1 to 0. End-of-busy interrupts are enabled by setting the FCON.EOBM bit.

The error flags in the Flash Status Register are controlled independently of the interrupt configuration as defined in the FCON register. Thus they may be polled without interrupt support. When set, an error flag has to be cleared by the user with the Clear Status command. All error flags are also cleared with any reset.

Errors, which are caused by a reset during program or erase operation, are not indicated; but its detection is supported via the PROG and ERASE flags in the Flash Status Register FSR. These two flags are not cleared with a warm reset, but only cleared with a power-on reset. Therefore, it is possible to detect an aborted Flash operation, if after every terminated Flash operation these flags are immediately cleared by the user by the Clear Status command, and if these flags are checked with every user-boot after reset (all except PORST reset).

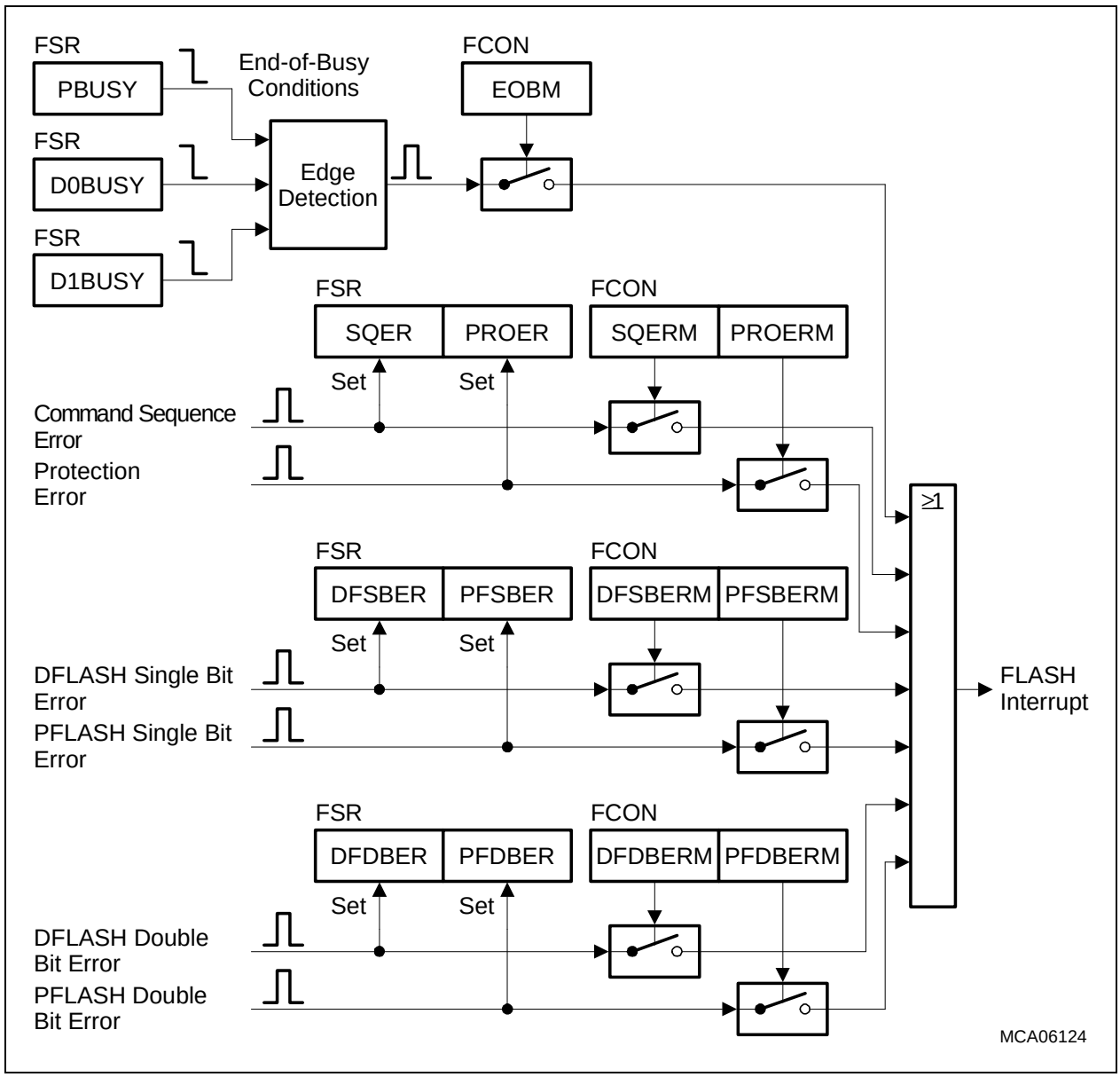


Figure 7-6 Flash Service Request Control

7.2.10 Flash Power Supply, Power Saving and Reset

This section describes some power supply, power saving mode, and reset related issues of the Flash modules.

7.2.10.1 Power Supply

The TC1766 Flash module has a separate 3.3 V power supply line V_{DDFL3}. It is used to generate on-chip regulated voltages for the program and erase operations, as well as for read operations.

If a power-fail occurs during a Flash erase or programming operation, a proper control of PORST reset input is required to avoid a possibly destructive operation (see Data Sheet for details on power-loss control).

7.2.10.2 Sleep Mode

The TC1766 Flash module provides a sleep mode which makes it possible to reduce the power consumption of the Flash module. Flash sleep mode is only entered if the Flash module is in idle state with all pending or active operations processed and terminated. Only in this case, the Flash module executes a ramp-down into Flash sleep mode. In Flash sleep mode, all sense amplifiers are switched off, the internal voltages are ramped down, and the Flash array oscillator is switched off. Next, status flag FSR.SLM is set indicating an active Flash sleep mode.

Flash sleep mode can be initiated by software (separately from the other modules in the TC1766) by setting bit FCON.SLEEP.

Flash sleep mode can also be initiated by hardware when the sleep mode is initiated by the power management system of the TC1766 (see **Section 5.1** on **Page 5-2**). A TC1766 device sleep mode request affects the Flash module only if the external sleep request disable bit FSR.ESLDIS is not set.

Wake-up from Flash sleep mode is initiated by clearing bit FCON.SLEEP or when the device sleep mode signal from the power management system is released. In both cases, the wake-up from sleep mode is controlled in the Flash module by starting the Flash array oscillator again, ramping-up the internal voltages, and returning to read mode. When it is again in read mode, the status flag FSR.SLM is cleared.

During ramp-down, active sleep mode, and wake-up from Flash sleep mode to active mode, the Flash reports to be busy also by setting the bits FSR.PBUSY, FSR.D0BUSY, and FSR.D1BUSY.

7.2.10.3 Shut-Down Mode

The Flash module can also be put into a shut-down mode in which it is switched off completely and cannot be used at all. This shut-down mode is not available for a user program but controlled by the BootROM startup procedure that is executed after each reset operation.

7.2.10.4 Reset Control

The Flash module uses

- the system reset, which is represented by the fast LMB reset; this reset includes all reset sources (power-on, hard, soft and watchdog reset), and
- the power-on reset.

The flash will be automatically reset to Read Mode within 1500 µs (including Flash ramp-up) after every reset represented by the system reset. When that happens, any erase or programming operation that is running is aborted, and the Flash may have erroneous data in the location being operated on (thus, the aborted operation must be repeated). Such an aborted state can be detected by careful handling of the PROG and ERASE flags in FSR.

Note: The startup time after reset until the first user instruction includes the Flash ramp-up and the BootROM startup, and it depends on the reset type (cold or warm) and on the clock frequency (PLL free-running or application frequency). But for all cases and clock > 12 MHz, it is < 2.5 ms (for normal start in internal Flash).

7.2.11 Flash Registers

This section includes the description of all Flash module related registers.

FLASH Register Overview

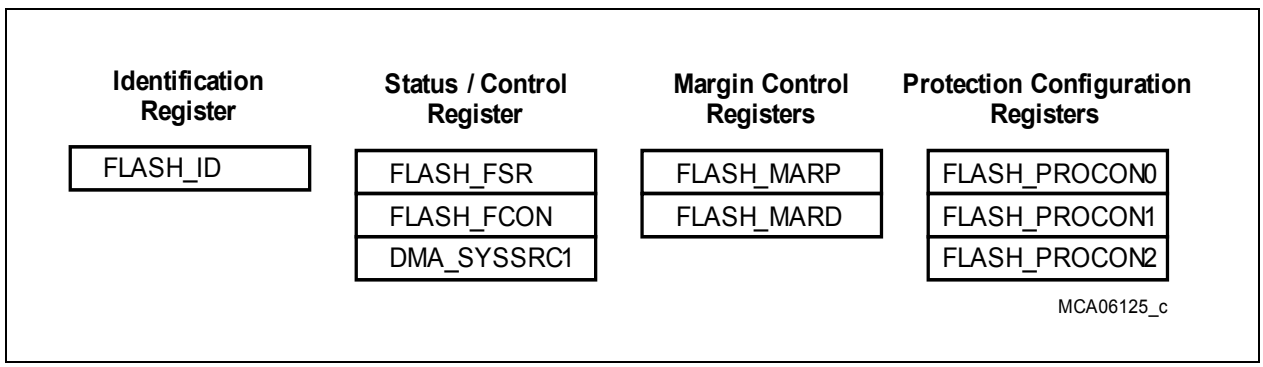


Figure 7-7 Flash Registers

The complete and detailed address map of the FLASH registers is described in Table 16-27 on Table 16-27 of the TC1766 User's Manual System Units part (Volume 1).

Table 7-17 Registers Address Space -Flash Registers

Module	Base Address	End Address	Note
FLASH	F800 1000_H	F800 $23FF_H$	-

Table 7-18 Registers Overview - Flash Registers

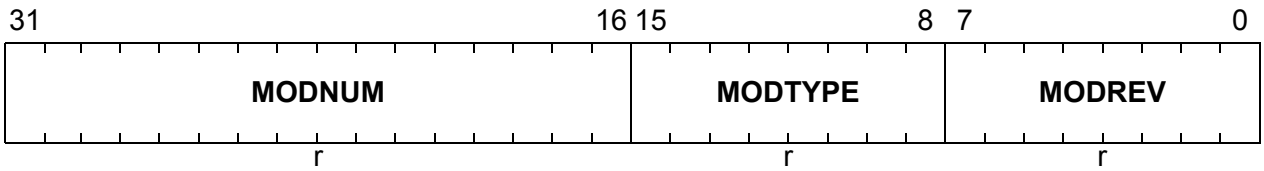
Register Short Name[1)]	Register Long Name	Offset Address	Description see
FLASH_ID	Flash ID Register	1008_H	Page 7-41
FLASH_FSR	Flash Status Register	1010_H	Page 7-42
FLASH_FCON	Flash Configuration Register	1014_H	Page 7-50
FLASH_MARP	Flash Margin Control Register PFLASH	1018_H	Page 7-56
FLASH_MARD	Flash Margin Control Register DFLASH	$101C_H$	Page 7-57
FLASH_PROCON0	Flash Protection Config. Reg. User 0	1020_H	Page 7-58
FLASH_PROCON1	Flash Protection Config. Reg. User 1	1024_H	Page 7-59
FLASH_PROCON2	Flash Protection Config. Reg. User 2	1028_H	Page 7-59
DMA_SYSSRC1	DMA System Interrupt Service Request Control Register 1	2)	Page 11-97

1) The Flash register short names are also referenced in other parts of this chapter without the module prefix name “FLASH_”.
2) This register is located in the address range for the DMA controller.

7.2.11.1 Flash Module Identification Registers

The Flash Module Identification Register ID contains read-only information about the Flash module version.

FLASH_ID
FLASH Module Identification Register(1008_H) **Reset Value: $0041\ C0XX_H$**



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 FLASH: 0041_H

7.2.11.2 Flash Status Register

The read-only Flash Status Register indicates the overall status of the Flash module. It includes busy, program, erase, error, protection, and sleep mode flags.

FLASH_FSR
Flash Status Register **(1010_H)** **Reset Value: X0XX X0X0$_H$**

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	0	rh	**PFLASH Busy** This flag indicates the busy state of the PFLASH during program or erase operations. It also indicates when the PFLASH is not in Read Mode, e.g. in ramp-up state. PBUSY is cleared by any reset operation. 0_B PFLASH is ready and in Read Mode. (default after reset) 1_B PFLASH is busy (program, erase, ramp-up or in sleep mode) and it is not in Read Mode.
FABUSY	1	rh	**Flash Array Busy** This status flag is a flag for test purposes that should not be used by software drivers. It indicates whether any of the Flash arrays is in busy state. FABUSY is cleared by any reset operation. 0_B Flash array banks are not busy and are in Read Mode (default after reset). 1_B At least one of the Flash array banks or the whole array is busy (program, erase, ramp-up or sleep mode). Flash read accesses are not possible.

Field	Bits	Type	Description
D0BUSY	2	rh	**DFLASH Bank 0 Busy** This flag indicates the busy state of DFLASH bank 0 during program or erase operations. It also indicates when the DFLASH bank 0 is not in Read Mode, e.g. in ramp-up state. D0BUSY is cleared by any reset operation. 0_B DFLASH bank 0 is ready and in Read Mode. (default after reset) 1_B DFLASH bank 0 is busy (program, erase, ramp-up or sleep mode) and it is not in Read Mode.
D1BUSY	3	rh	**DFLASH Bank 1 Busy** This flag indicates the busy state of DFLASH bank 1 during program or erase operations. It also indicates when the DFLASH bank 1 is not in Read Mode, e.g. in ramp-up state. D0BUSY is cleared by any reset operation. 0_B DFLASH bank 1 is ready and in Read Mode. (default after reset) 1_B DFLASH bank 1 is busy (program, erase, ramp-up or sleep mode) and it is not in Read Mode.
PROG	4	rh	**Programming State** This flag indicates whether PFLASH or DFLASH is currently programmed or have been programmed. PROG is set with the last cycle of a write page command. After a programming operation, this bit remains set. PROG is cleared by a Clear Status command or by a power-on reset. 0_B There is no programming operation running, requested or finished. 1_B A programming operation (write page) is running, requested or finished.

Field	Bits	Type	Description
ERASE	5	rh	**Erase State** This flag indicates whether PFLASH or DFLASH is currently erased or have been erased. ERASE is set with the last cycle of an erase command sequence. After an erase operation this bit remains set. ERASE is cleared by a Clear Status command or by a power-on reset. 0_B There is no erase operation running, requested, or finished. 1_B An erase operation is running, requested, or finished.
PFPAGE	6	rh	**PFLASH in Page Mode** This flag indicates whether the PFLASH is in Page Mode or not. It is set with the enter Page Mode command and cleared by a Write Page command, or by a Reset-to-Read command, or by any reset operation. 0_B PFLASH is not in Page Mode. (default after reset) 1_B PFLASH is in Page Mode. The page assembly buffer of the PFLASH is ready to be filled.
DFPAGE	7	rh	**DFLASH in Page Mode** This flag indicates whether the DFLASH is in Page Mode or not. It is set with the enter Page Mode command and cleared by a write Page Mode command, or by a Reset-to-Read command, or by any reset operation. 0_B DFLASH is not in Page Mode. (default after reset) 1_B DFLASH is in Page Mode. The page assembly buffer of the DFLASH is ready to be filled.
PFOPER	8	rh	**PFLASH Operation Error** This flag indicates a flash array error during PFLASH operation. PFOPER is cleared by a reset operation, by a Reset-to-Read command, or by a Clear Status command. 0_B PFLASH operation is finished or currently running without error. 1_B PFLASH operation is not correctly executed due to a PFLASH array failure.

Field	Bits	Type	Description
DFOPER	9	rh	**DFLASH Operation Error** This flag indicates a flash array error during DFLASH operation. DFOPER is cleared by a reset operation, by a Reset-to-Read command, or by a Clear Status command. 0_B DFLASH operation is finished or currently running without error. 1_B DFLASH operation is not correctly executed due to a DFLASH array failure.
SQER	10	rh	**Command Sequence Error** This flag indicates a command sequence error. SQER is cleared by a Reset-to-Read command, or by a Clear Status command, or by any reset operation. SQER is not set when a command sequence is aborted by a Reset-to-Read command. 0_B No command sequence error has occurred. (default after reset) 1_B An invalid command sequence has been detected (command is not executed).
PROER	11	rh	**Protection Error** This flag indicates a protection error. Such an error is a command that is not allowed, for example an erase sector or write page command addressing a locked sector. It also indicates when wrong passwords are used in disable read or disable write protection commands. PROER is cleared by a Reset-to-Read command, or by a Clear Status command, or by any reset operation. 0_B No protection error has occurred. (default after reset) 1_B A protection error has occurred.
PFSBER	12	rh	**PFLASH Single-Bit Error and Correction** This flag indicates the occurrence of a single-bit error in the PFLASH that has been corrected. PFSBER is cleared by a Reset-to-Read command, or by a Clear Status command, or by any reset operation. 0_B No error has been detected. (default after reset) 1_B A single-bit error in PFLASH has been detected and corrected.

Field	Bits	Type	Description
DFSBER	13	rh	**DFLASH Single-Bit Error and Correction** This flag indicates the occurrence of a single-bit error in the DFLASH that has been corrected. DFSBER is cleared by a Reset-to-Read command, or by a Clear Status command, or by any reset operation. 0_B No error has been detected. (default after reset) 1_B A single-bit error in DFLASH has been detected and corrected.
PFDBER	14	rh	**PFLASH Double-Bit Error** This flag indicates the occurrence of a double-bit error in the PFLASH. PFDBER is cleared by a Reset-to-Read command, or by a Clear Status command, or by any reset operation. 0_B No error has been detected. (default after reset) 1_B A double-bit error in PFLASH has been detected.
DFDBER	15	rh	**DFLASH Double-Bit Error** This flag indicates the occurrence of a double-bit error in the DFLASH. DFDBER is cleared by a Reset-to-Read command, or by a Clear Status command, or by any reset operation. 0_B No error has been detected. (default after reset) 1_B A double-bit error in DFLASH has been detected.
PROIN	16	rh	**Protection Installed** This flag indicates whether read and/or write protection is correctly installed. This bit is updated only if a modified protection installation is detected at a reset operation. 0_B No protection is installed. 1_B A read and/or write protection is correctly installed.

Field	Bits	Type	Description
RPROIN	18	rh	**Read Protection Installed** This flag indicates whether read protection is correctly installed. This bit is updated only if a modified read protection installation is detected at a reset operation. 0_B No read protection installed. 1_B Read protection is correctly installed.
RPRODIS	19	rh	**Read Protection Disable State** This flag indicates whether read protection is temporarily disabled. RPRODIS is cleared by any reset operation and by the Resume Protection command. 0_B The read protection is installed and enabled, or it is not installed (default after reset). 1_B The read and global write protection is temporarily disabled. External Flash read as well as programming or erase on not separately write-protected sectors is possible.
WPROIN0	21	rh	**UCB0 Write Protection Installed** This flag indicates whether sector write protection in user configuration block UCB0 is correctly installed and confirmed. This bit is updated only if a modified UCB0 protection installation is detected at a reset operation. 0_B No write protection installed in UCB0. 1_B Sector write protection is correctly installed in UCB0.
WPROIN1	22	rh	**UCB1 Write Protection Installed** This flag indicates whether sector write protection in UCB1 is correctly installed and confirmed. This bit is updated only if a modified UCB1 protection installation is detected at a reset operation. 0_B No write protection installed in UCB1. 1_B Sector write protection is correctly installed in UCB1.

Field	Bits	Type	Description
WPROIN2	23	rh	**UCB2 Write Protection (OTP) Installed** This flag indicates whether sector write protection (OTP) in user configuration block UCB2 is correctly installed and confirmed. 0_B No OTP write protection installed in UCB2. 1_B Sector OTP protection correctly installed in UCB2. The protection is permanently locked.
WPRODIS0	25	rh	**UCB0 Write Protection Disabled** This flag indicates whether PFLASH sectors that are write-protected by user configuration block UCB0 are temporarily unlocked. WPRODIS0 is cleared by any reset operation and by the Resume Protection command. 0_B All write-protected sectors of UCB0 are locked or write protection is not installed in UCB0. 1_B All write-protected sectors of UCB0 and UCB1 are temporarily unlocked. The programming or erasing of UCB0 is possible if no read protection is installed. Hierarchical protection levels: Unlock state of user-level zero also includes sectors that are protected by user 1.
WPRODIS1	26	rh	**UCB1 Write Protection Disabled** This flag indicates whether PFLASH sectors that are write-protected by UCB1 are temporarily unlocked. WPRODIS1 is cleared by any reset operation and by the Resume Protection command. 0_B All write-protected sectors of UCB1 are locked or write protection is not installed in UCB1. 1_B All write-protected sectors of UCB1 are temporarily unlocked if not globally (read protection) or concurrently (write protection) locked by UCB0/UCB2. The programming or erasing of UCB1 is possible.
SLM	28	rh	**Flash Sleep Mode** This flag indicates whether the Flash module is in sleep mode. SLM is cleared by any reset operation or when Flash sleep mode is left. 0_B Flash is not in sleep mode (default after reset). 1_B Flash is in sleep mode.

Field	Bits	Type	Description
X	30	rh	**Undefined after reset.**
VER	31	rh	**Verify Error** This flag indicates whether a Flash page has been correctly programmed or a Flash sector has been correctly erased. VER is cleared by any reset operation and by the Clear Status command. 0_B The page is correctly programmed or the sector correctly erased. All programmed or erased bits have full expected quality. 1_B A programming verify error or an erase verify error has been detected, and the correction of the weak bits was unsuccessful during the last program or erase operation. *Note: There is no interrupt generated when VER is set.*
0	17, 20, 24, 27, 29	r	**Reserved** Read as 0.

7.2.11.3 Flash Configuration Register

The Flash Configuration Register FCON controls general Flash configuration functions:

Number of internal wait states for Flash accesses
Indication of installed and active read protection
Instruction and data access control for read protection
Interrupt enable/mask bits
Power reduction and shut-down control

FCON is an Endinit-protected register. The reset value of register FCON as indicated below shows the value after ramp-up and after execution of the startup procedure out of Boot ROM.

FLASH_FCON
Flash Configuration Register (1014_H) Reset Value: 000X 0666_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EOBM	DFDBERM	PFDBERM	DFSBERM	PFSBERM	PROERM	SQERM	0			DDFPCP	DDFDMA	DDFDBG	DDF	DCF	RPA
rw	rw	rw	rw	rw	rw	rw	r			rw	rw	rw	rwh	rwh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SLEEP	ESLDIS	0		WSECDF	WSDFLASH			0	WSWLHIT			WSECPF	WSPFLASH		
rw	rw	r		rw	rw			r	rw			rw	rw		

Field	Bits	Type	Description
WSPFLASH	[2:0]	rw	**Wait States for PFLASH Read Access[1)]** This bit field determines the number of internal wait states that are used for an initial PFLASH read access. 000_B Not supported 001_B Not supported 010_B Not supported 011_B 3 clock cycles wait state selected 100_B 4 clock cycles wait state selected (support up to f_{CPU} of 80 MHz) 101_B 5 clock cycles wait state selected 110_B 6 clock cycles wait state selected (default after reset) 111_B 7 clock cycles wait state selected

Field	Bits	Type	Description
WSECPF	3	rw	**Wait State for PFLASH Error Correction[1)]** This bit determines whether an additional wait state is inserted for error correction during read accesses to PFLASH. 0_B No additional wait state inserted. 1_B One additional wait state inserted for PFLASH error correction. If enabled, this wait state is only used for the first transfer of a burst transfer.
WSWLHIT	[6:4]	rw	**Wait States for PFLASH Read Access with Wordline Hit[1)]** This bit field determines the number of internal wait states that are used for an initial read access to the same word-line (512 byte) of the PFLASH memory as the last PFLASH access. 000_B Reserved 001_B Reserved 010_B Reserved 011_B Flash access in 3 clock cycles 100_B Flash access in 4 clock cycles 101_B Flash access in 5 clock cycles 110_B Flash access in 6 clock cycles (default after reset) 111_B Reserved *Note: No access acceleration with WL hit. Therefore, the number of WSWLHIT waitstates must be identical to the WSPFLASH number.*
WSDFLASH	[10:8]	rw	**Wait States for DFLASH Read Access[1)]** This bit field determines the number of internal wait states that are used for a DFLASH read access. 000_B Not supported 001_B Not supported 010_B Not supported 011_B 3 clock cycles wait state selected 100_B 4 clock cycles wait state selected (support up to f_{CPU} of 80 MHz) 101_B 5 clock cycles wait state selected 110_B 6 clock cycles wait state selected (default after reset) 111_B 7 clock cycles wait state selected

Field	Bits	Type	Description
WSECDF	11	rw	**Wait State for DFLASH Error Correction**[1)] This bit determines whether an additional wait state is inserted for error correction during read accesses to DFLASH. 0_B No additional wait state inserted. 1_B One additional wait state inserted for DFLASH error correction.
ESLDIS	14	rw	**External Sleep Request Disable** 0_B External sleep mode request also requests the Flash sleep mode. 1_B External sleep mode request does not request the Flash sleep mode. The external sleep mode is controlled and requested by the Power Management System of the SCU.
SLEEP	15	rw	**Flash Sleep Mode Control** This bit controls the Flash sleep mode of the Flash module. 0_B Normal Flash mode or wake-up mode is active. 1_B Flash sleep mode is selected. The individual Flash sleep mode is left (wake-up from sleep) by clearing the SLEEP bit.
RPA	16	rh	**Read Protection Activated** This flag indicates the status of the read protection. 0_B The read protection is inactive or temporarily disabled. Bits DCF and DDF can be cleared or set by software. DCF and DDF are not evaluated. 1_B The read protection is active. Bits DCF and DDF are evaluated and cannot be cleared by software.

Field	Bits	Type	Description
DCF	17	rwh	**Disable Code Fetch from Flash Memory** This bit enables/disables the code fetches from the internal PFLASH memory when read protection is active. Once set, this bit can only be cleared when read protection is inactive (RPA = 0). DCF is automatically set after a reset operation. It is cleared by hardware in case of internal program start out of the PFLASH. 0_B Code fetches from PFLASH are allowed. 1_B Code fetches from PFLASH are not allowed. DCF is not evaluated when read protection is inactive (RPA = 0).
DDF	18	rwh	**Disable Data Fetch from Flash Memory** This bit enables/disables the data read access from the PFLASH and DFLASH memory when read protection is active. Once set, this bit can only be cleared when read protection is inactive (RPA = 0). DDF is automatically set after a reset operation. It is cleared by hardware in case of internal program start out of the PFLASH. 0_B Data read access to PFLASH and DFLASH is allowed. 1_B Data read access to PFLASH and DFLASH is not allowed. DDF is not evaluated when read protection is inactive (RPA = 0).
DDFDBG	19	rw	**Disable Data Fetch from Debug System** This bit enables/disables PFLASH and DFLASH data read accesses that are initiated by the on-chip debug system (Cerberus). Once set, this bit can only be cleared when RPA = 0. 0_B Data read accesses from PFLASH/DFLASH initiated by the debug system are enabled. 1_B Data read accesses from PFLASH/DFLASH initiated by the debug system are disabled.

Field	Bits	Type	Description
DDFDMA	20	rw	**Disable Data Fetch from DMA Controller** This bit enables/disables PFLASH and DFLASH data read accesses that are initiated by the DMA controller. Once set, this bit can only be cleared when RPA = 0. 0_B Data read accesses from PFLASH/DFLASH initiated by the DMA controller are enabled. 1_B Data read accesses from PFLASH/DFLASH initiated by the DMA controller are disabled.
DDFPCP	21	rw	**Disable Data Fetch from PCP** This bit enables/disables PFLASH and DFLASH data read accesses that are initiated by the PCP. Once set, this bit can only be cleared when RPA = 0. 0_B Data read accesses from PFLASH/DFLASH initiated by the PCP are enabled. 1_B Data read accesses from PFLASH/DFLASH initiated by the PCP are disabled.
SQERM	25	rw	**Command Sequence Error Interrupt Mask** This bit disables/enables the command sequence error interrupt. 0_B Command sequence error interrupt is disabled. 1_B Command sequence error interrupt is enabled.
PROERM	26	rw	**Protection Error Interrupt Mask** This bit disables/enables the protection error interrupt. 0_B Protection error interrupt is disabled. 1_B Protection error interrupt is enabled.
PFSBERM	27	rw	**PFLASH Single-Bit Error Interrupt Mask** This bit disables/enables the PFLASH single-bit error interrupt. 0_B PFLASH single-bit error interrupt is disabled. 1_B PFLASH single-bit error interrupt is enabled.
DFSBERM	28	rw	**DFLASH Single-Bit Error Interrupt Mask** This bit disables/enables the DFLASH single-bit error interrupt. 0_B DFLASH single-bit error interrupt is disabled. 1_B DFLASH single-bit error interrupt is enabled.

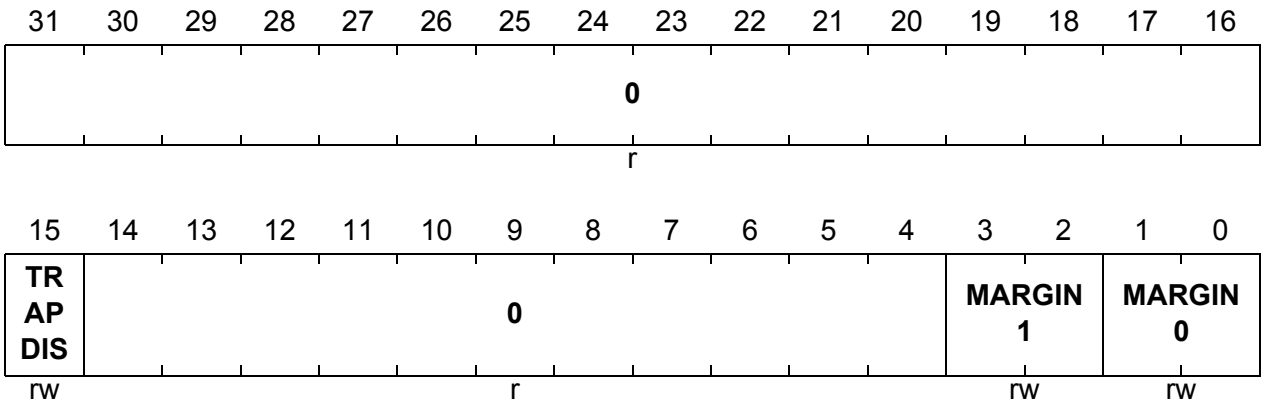
Field	Bits	Type	Description
PFDBERM	29	rw	**PFLASH Double-Bit Error Interrupt Mask** This bit disables/enables the PFLASH double-bit error interrupt. 0_B PFLASH double-bit error interrupt is disabled. 1_B PFLASH double-bit error interrupt is enabled.
DFDBERM	30	rw	**DFLASH Double-Bit Error Interrupt Mask** This bit disables/enables the DFLASH double-bit error interrupt. 0_B DFLASH single-bit error interrupt is disabled. 1_B DFLASH single-bit error interrupt is enabled.
EOBM	31	rw	**End-of-Busy Interrupt Mask** This bit enables the end-of-busy interrupt. 0_B End-of-busy interrupt is disabled. 1_B End-of-busy interrupt is enabled.
0	7, [13:12], [24:22]	r	**Reserved** Read as 0; should be written with 0.

1) These bits and bit fields can be changed at any time, also with code fetched from Program Flash. A modified wait state parameter will be taken into account with the next corresponding access.

7.2.11.4 Margin Control Registers

The Margin Control Registers for Program Flash (MARP) and for Data Flash (MARD) are defined as follows:

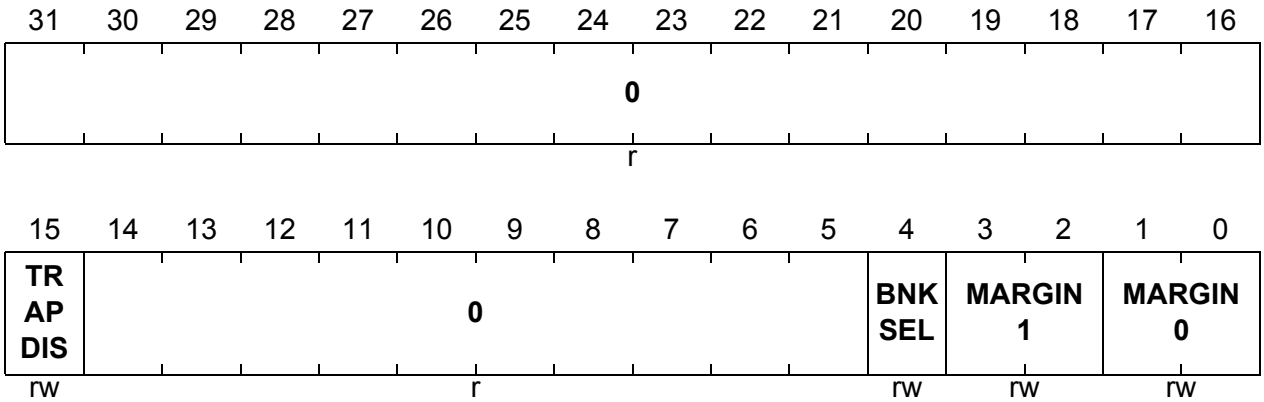
FLASH_MARP
Flash Margin Control Register PFLASH(1018_H) **Reset Value: $0000\ 8000_H$**



Field	Bits	Type	Description
MARGIN0	[1:0]	rw	**PFLASH Margin Selection for Low Level** 00_B Standard margin selected 01_B High margin for 0 (low) level selected 10_B Reserved 11_B Reserved
MARGIN1	[3:2]	rw	**PFLASH Margin Selection for High Level** 00_B Standard margin selected 01_B High margin for 1 (high) level selected 10_B Reserved 11_B Reserved
TRAPDIS	15	rw	**PFLASH Double-Bit Error LMB Bus Error Disable** 0_B If a double-bit error occurs in PFLASH, a trap is generated. 1_B If a double-bit error occurs in PFLASH, no trap is generated. After Boot ROM exit, double-bit error traps are enabled (TRAPDIS = 0).
0	[14:4], [31:16]	r	**Reserved** Read as 0; should be written with 0.

FLASH_MARD

Flash 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 margin selected 01_B High margin for 0 (low) level selected 10_B Reserved 11_B Reserved
MARGIN1	[3:2]	rw	**DFLASH Margin Selection for High Level** 00_B Standard margin selected 01_B High margin for 1 (high) level selected 10_B Reserved 11_B Reserved
BNKSEL	4	rw	**DFLASH Bank Selection** 0_B DFLASH bank 0 selected for margin control 1_B DFLASH bank 1 selected for margin control
TRAPDIS	15	rw	**DFLASH Double-Bit Error LMB Bus Error Disable** 0_B If a double-bit error occurs in DFLASH, a trap is generated. 1_B If a double-bit error occurs in DFLASH, no trap is generated. After Boot ROM exit, double-bit error traps are enabled (TRAPDIS = 0).
0	[14:5], [31:16]	r	**Reserved** Read as 0; should be written with 0.

7.2.11.5 Protection Configuration Registers

The configuration of read/write protection and OTP functionality is indicated by three Protection Configuration Registers:

- FLASH_PROCON0 indicates the read/write protection configuration for user 0 (UCB0)
- FLASH_PROCON1 indicates the write protection configuration for user 1 (UCB1)
- FLASH_PROCON2 indicates the OTP write protection for user 2 (UCB2)

FLASH_PROCON0
Flash Protection Configuration Register User 0
(1020_H) **Reset Value: 0000 XXXX$_H$**

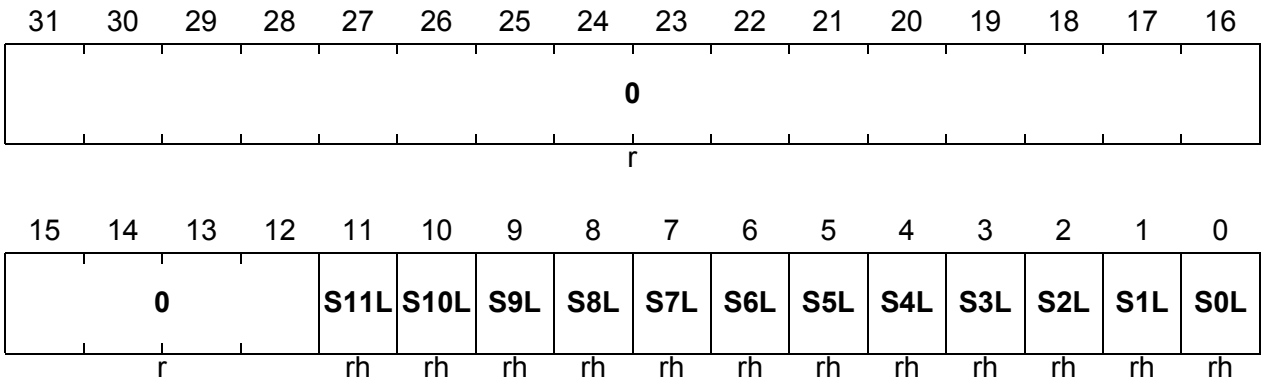
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RPRO	0			S11L	S10L	S9L	S8L	S7L	S6L	S5L	S4L	S3L	S2L	S1L	S0L
rh	r			rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
SnL (n = 0-11)	n	rh	**Sector Locked for Write Protection by User 0** These bits indicate whether PFLASH sector PSn is write-protected by user 0 or not. 0_B No write protection is installed for PFLASH sector PSn. 1_B Write protection is installed for PFLASH sector PSn.
RPRO	15	rh	**Read Protection Configuration** This bit indicates whether read protection is installed. 0_B No read protection installed 1_B Read protection and global write protection are installed
0	[14:12], [31:16]	r	**Reserved** Read as 0.

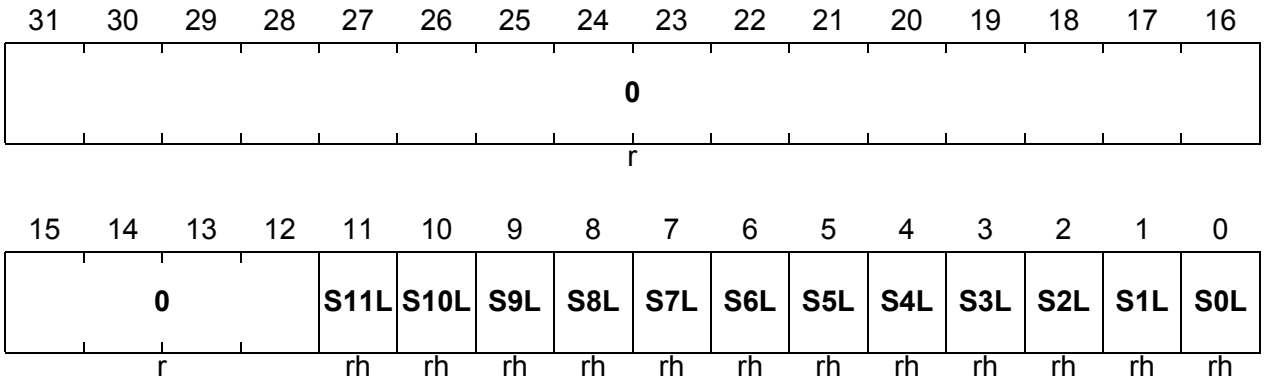
FLASH_PROCON1
Flash Protection Configuration Register User 1

(1024$_H$) **Reset Value: 0000 XXXX$_H$**



Field	Bits	Type	Description
SnL (n = 0-11)	n	rh	**Sector Locked for Write Protection by User 1** These bits indicate whether sector n is write-protected by user 1 or not. 0_B No write protection installed for sector n. 1_B Write protection installed for sector n.
0	[31:12]	r	**Reserved** Read as 0.

FLASH_PROCON2
Flash Protection Configuration Register User 2
(1028_H) **Reset Value: 0000 XXXX$_H$**



Field	Bits	Type	Description
SnL (n = 0-11)	n	rh	**Sector with OTP Protection (User 2)** These bits indicate whether an OTP protection is installed and locked for sector n. 0_B No OTP protection installed for sector n. 1_B OTP protection installed for sector n. Reprogramming of this sector is no longer possible.
0	[31:12]	r	**Reserved** Read as 0.

7.3 Emulation Interface

The emulation memory interface shown in **Figure 7-1** is always disabled in the TC1766 device. This interface is a 64-bit wide memory interface that controls an emulation memory in the TC1766 emulation device, called TC1766ED. Segments 8 and 10 reserve 256 Kbyte per segment for the emulation memory interface.

In the TC1766, 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 or DMA), additionally a SPB error is generated. Write accesses to the emulation memory region by any master causes an LMB bus error.

7.4 Data Access Overlay Functionality

The data overlay functionality provides the capability to redirect data read accesses from internal code memory to data accesses from the PMU SRAM of 8 Kbyte. This functionality makes it possible, for example, to modify program parameters (which are typically stored in the code memory) during run time of a program. Instruction fetches are not affected by the PMU data read access redirection capability. Note that read and write data accesses from/to code memory are redirected.

The basic overlay scheme is shown in **Figure 7-8**.

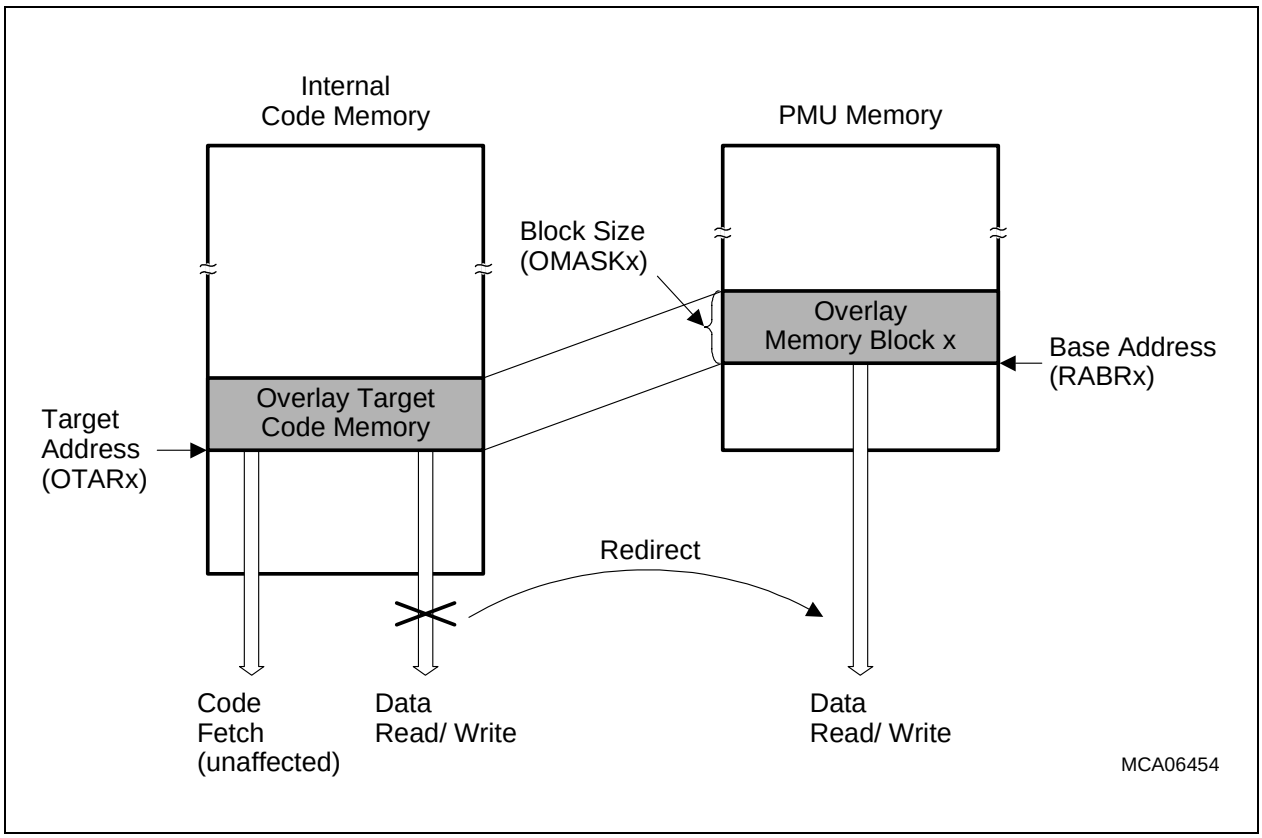


Figure 7-8 Redirection of Data Read Accesses from Code Memory to PMU SRAM (Internal Data) Memory

In the TC1766, the 8 Kbyte PMU SRAM memory can be divided into a maximum of sixteen overlay memory blocks, with programmable base address and block sizes, which can be individually enabled for overlay functionality. The block size of each overlay memory block can be in the range of 2 bytes up to 512 bytes.

Attention: In the TC1766ED emulation device, it is possible in addition to redirect code memory data accesses to the emulation memory of the TC1766ED.

The principal operation of the address translation process is shown in **Figure 7-9**.

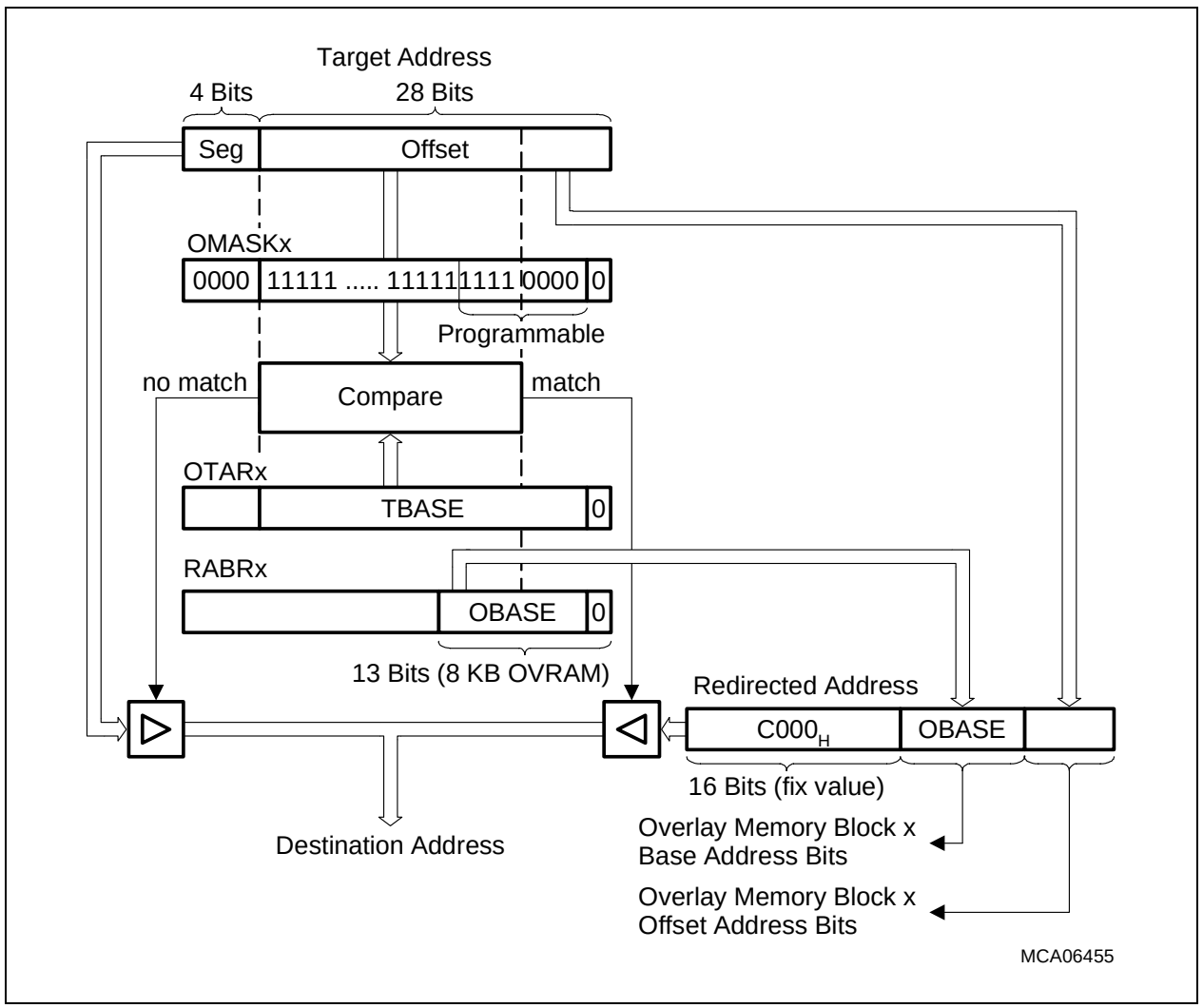


Figure 7-9 Address Translation Process

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 code memory data block x to be redirected
- The base address of the overlay memory block in the PMU SRAM in the Redirected Address Base Register RABRx
- A mask in the Overlay Mask Register OMASKx, defining the size of the block, the address bits to check for an address match, and which bits are used from the target address and which from the redirected address base

The size of the overlay memory blocks can be 2^n times the minimal block size (2 byte or 1 Kbyte for internal memory overlay and emulation memory overlay, respectively). The

start address of the block can be an integer multiple of the programmed block size (natural page boundary).

If the overlay block is enabled and the segment address is 8_H or A_H, the segment offset of the target address is compared with the target block address of the overlay target address. This bit-wise comparison is qualified by the content of the mask, ignoring the address bits that form the offset into the overlayed block.

If there is no match, the original target address is taken to perform the access.

If there is a match:

- The most significant part of the segment offset, that addresses memory sizes beyond the maximum overlay memory size, 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 and is masked, 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.

7.4.1 Internal Overlay Memory

For internal overlay, the size of the blocks can be 2^n, with n = 1 to 9 (2 to 512 byte). Thus the maximum PMU memory size needed for overlay memory (if all blocks are used with maximum size) is 8 Kbyte.

During address translation, the segment number is set to C_H.

7.4.2 Emulation Overlay Memory

For Emulation Memory Overlay (only available in the emulation device TC1766ED), the size of the blocks can be $2^n \times 1$ Kbyte, with n = 0 to 5 (1 Kbyte to 32 Kbyte). Thus, the maximum memory size needed for overlay memory (if all blocks are used with maximum size) is 512 Kbyte.

7.4.3 Switching between Internal and Emulation Overlay Memory

When switching a region between internal and external overlay, it must be ensured that the respective OVEN bit is cleared, all region parameters are set properly, and then the region is enabled again. Otherwise, unintended access re-directions may occur.

7.4.4 Region Priority

If concurrent matches in more than one region occur, the region with the lowest order number will win and perform the address translation.

7.4.5 Access Performance

Write accesses are performed with zero wait states.

If no overlay is enabled, the accesses to the PMU memories take two cycles (2 wait states).

The access to the Emulation Memory will not be more than 10 cycles. Details can be found in TC1766ED specific documents.

7.5 PMU Overlay Control Registers

Figure 7-10 shows all the overlay control registers associated with the overlay memory in the PMU.

PMU Registers Overview

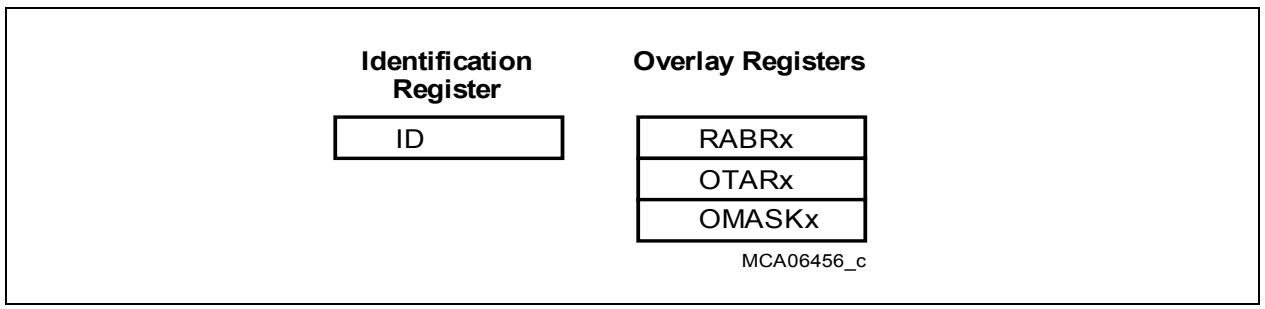


Figure 7-10 PMU Overlay Control Registers

The complete and detailed address map of the PMU module with its registers is shown in Table 16-26 on Page 16-76 of this TC1766 System Units (Vol. 1 of 2) User's Manual.

Table 7-19 Registers Address Space - PMU Overlay Registers

Module	Base Address	End Address	Note
PMU	$F800\ 0500_H$	$F800\ 05FF_H$	-

Table 7-20 Registers Overview PMU Overlay Registers

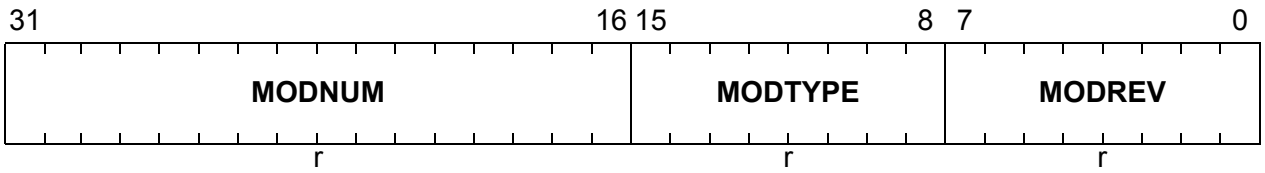
Register[1)] Short Name	Register Long Name	Offset Address	Description see
ID	Module Identification Register	08_H	Page 7-66
RABRx	Redirected Address Base Register x (x = 0-15)	20_H + x × C_H	Page 7-67
OTARx	Overlay Target Address Register x (x = 0-15)	20_H + x × C_H + 4	Page 7-69
OMASKx	Overlay Mask Register x (x = 0-15)	20_H + x × C_H + 8	Page 7-70

1) The PMU register short names are extended and referenced in the other parts of the TC1766 User's Manual with the module name prefix "PMU_".

The PMU Module Identification Register ID contains read-only information about the PMU module version.

PMU_ID
PMU Module Identification Register

(0008_H) **Reset Value: $002E\ C0XX_H$**



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 PMU: $002E_H$

For each of the 16 overlay sections (indicated by index x), three registers control the overlay operation:

- The Redirected Address Base Register RABRx, which holds the base address of the overlay memory block in the overlay memory, and some control bits
- The Overlay Target Address Register OTARx, which holds the base of the memory block being overlayed
- The Overlay Mask Register OMASKx, which determines which bits are part of the offset and which are part of the base address.

PMU_RABRx (x = 0-15)
PMU Redirected Address Base Register x

(20_H+x*C_H) **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OV EN	**IEMS = 0**	**RC1**	**RC0**	**0**											
rw	r	r	r	r											

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

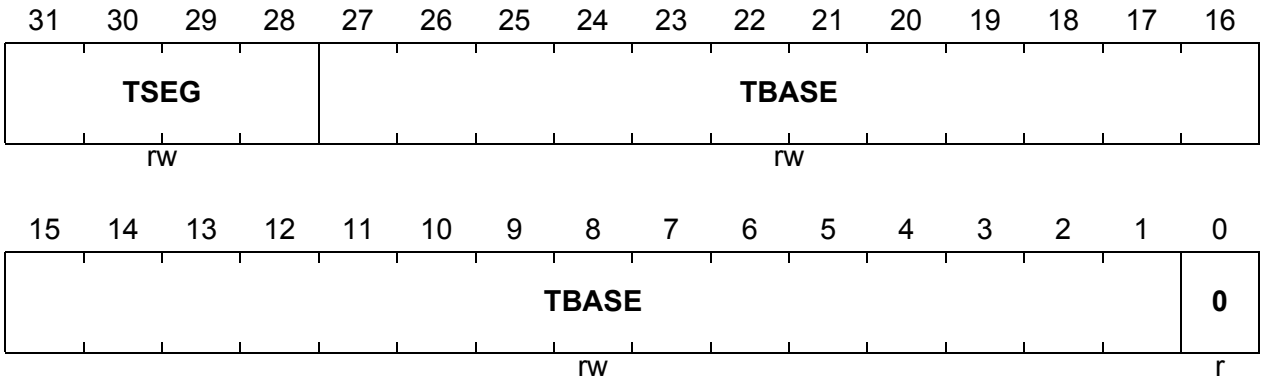
Field	Bits	Type	Description
OBASE	[12:1]	rw	**Overlay Base Address** This bit field holds the base address of the overlay memory block in the overlay memory. The largest block base address that is allowed, is ($OBASE_{max}$ - block size[1)])
RC0, RC1	28, 29	r	**Reserved Control Bits** These bits are reserved for future control expansions. Read returns 0. Bits should be written with 0.

Field	Bits	Type	Description
IEMS = 0	30	r	**Internal/Emulation Memory Select** This bit indicates the location of the overlay memory. In the TC1766, this bit is always read as 0, indicating that the internal memory (PMU SRAM) is used as overlay memory. 0_B Internal memory (PMU SRAM) is selected as overlay memory; Block sizes are 2^n bytes, n = 1-9; Segment address bits [31:28] are set to 1100_B. *Note: In a TC1766ED emulation device, this bit can be written, too. When set, an emulation memory is selected as overlay memory (not applicable in the TC1766).*
OVEN	31	rw	**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.
0	0, [27:13]	r	**Reserved** Read as 0; should be written with 0.

1) The block size is determined by the mask register OMASK.

PMU_OTARx (x = 0-15)
PMU Overlay Target Address Register x

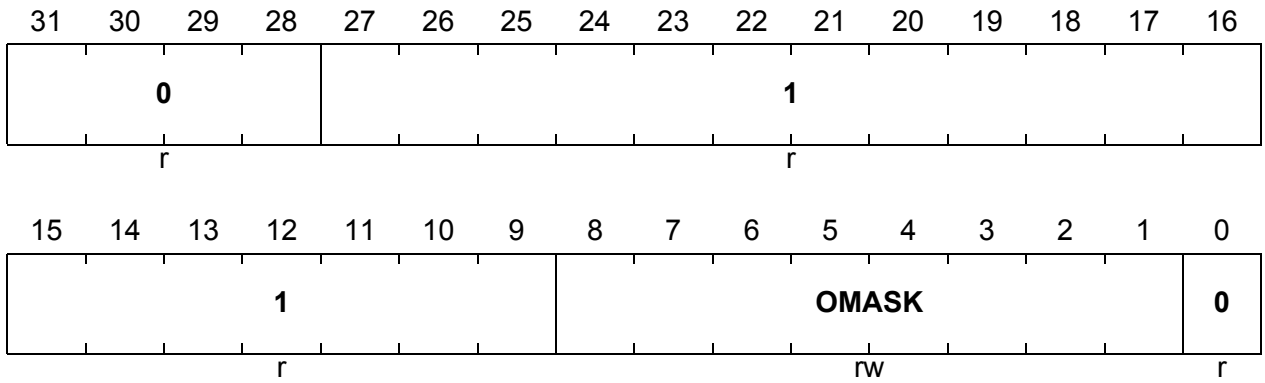
(24_H+x*C_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
TBASE	[27:1]	rw	**Target Base** Holds the base address of the overlay memory block in the target memory. If IEMS is set, bits [9:1] will be forced to 0 and cannot be modified.
TSEG	[31:28]	rw	**Target Segment (reserved)** This bit field is reserved for future use, to select a segment. In TC1766 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	0	r	**Reserved** Read as 0; should be written with 0.

The Overlay Mask Register x determines the size of the overlay memory block x. It also determines which address bits will participate in the address compare for a block base address and which bits are used from the original target address and which bits are taken from RABRx.OBASE.

PMU_OMASKx (x = 0-15)
PMU Overlay Mask Register x **(28_H+x*C_H)** **Reset Value: 0FFF FE00$_H$**



Field	Bits	Type	Description
OMASK	[8:1]	rw	**Overlay Address Mask** This bit field determines two parameters: - Block size of the overlay memory block x - Address bits taken for address generation The mask bits determine which address bits are taken to check if a target address is overlayed, and determines which address bits are taken from the original target address as offset, and which are taken from RABRx.OBASE. Thus the mask also determines the block size of the overlay block. 0_B Corresponding address bits are not used in the address comparison Corresponding final address bits are derived from the original address Block size is 2^n bytes, n = number of trailing zeros in the register 1_B Corresponding address bits are used in the address comparison Corresponding final address bits are derived from RABR_OBASE Block size is 2^n bytes, n = number of trailing zeros in the register 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. Selected overlay memory block size: 00000000_B 512 bytes 10000000_B256 bytes 11000000_B128 bytes 11100000_B64 bytes 11110000_B32 bytes 11111000_B16 bytes 11111100_B8 bytes 11111110_B4 bytes 11111111_B2 bytes

Field	Bits	Type	Description
0	0, [31:28]	r	**Fixed Values** Corresponding address bits are not used in the address comparison. Corresponding final address bits are taken from the original address.
1	[27:16], [15:9]	r	**Fixed Values** Corresponding address bits are participating in the address comparison. Corresponding final address bits are taken from RABR.

8 Memory Maps

This chapter gives an overview of the TC1766 memory map, and describes the address locations and access possibilities for the units, memories, and reserved areas as “seen” from the three different on-chip buses’ point of view.

The TC1766 has the following memories

- Program Memory Unit (PMU) with
  - 1504 Kbyte of Program Flash Memory (PFLASH)
  - 32 Kbyte of Data Flash Memory (DFLASH)
  - 16 Kbyte of Boot ROM (BROM)
- Program Memory Interface (PMI)
  - 16 Kbyte of Scratch-Pad RAM (SPRAM)
  - 8 Kbyte of Instruction Cache (ICACHE)
- Data Memory Interface (DMI)
  - 56 Kbyte of Local Data RAM (LDRAM)
- PCP memory
  - 12 Kbyte of PCP Code Memory (CMEM)
  - 8 Kbyte of PCP Data Memory (PRAM)

Furthermore, the TC1766 has two on-chip buses:

- System Peripheral Bus (SPB)
- Local Memory Bus (LMB)

8.1 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 OCDS, and DMA.

The LMB address map shows the system addresses from the point of view of the LMB master agents. LMB master agents are PMI and DMI.

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.

8.2 Contents of the Segments

This section summarizes the contents of the segments.

Segments 0-7

These segments are reserved segments in the TC1766.

Segment 8

From the SPB point of view (PCP, DMA, and Cerberus), this memory segment allows accesses to all PMU memories (PFLASH, DFLASH, BROM, and TROM).

From the CPU point of view (PMI and DMI), this memory segment allows cached accesses to all PMU memories (PFLASH, DFLASH, BROM, and TROM).

Segment 9

This memory segment is reserved in the TC1766.

Segment 10

From the SPB point of view (PCP, DMA, and Cerberus), this memory segment allows accesses to all PMU memories (PFLASH, DFLASH, BROM, and TROM).

From the CPU point of view (PMI and DMI), this memory segment allows non-cached accesses to all PMU memories (PFLASH, DFLASH, BROM, and TROM).

Segment 11

This memory segment is reserved in the TC1766.

Segment 12

From the SPB point of view (PCP, DMA, and Cerberus), this memory segment is reserved in the TC1766.

From the CPU point of view (PMI and DMI), this memory segment allows cached accesses to the PMU memory, OVRAM.

Segment 13

From the SPB point of view (PCP, DMA, and Cerberus), this memory segment is reserved in the TC1766.

From the CPU point of view (PMI and DMI), this memory segment allows non-cached accesses to the PMI scratch-pad RAM, read access to the boot ROM and test ROM (BROM and TROM) and the DMI memories (LDRAM).

Segment 14

From the SPB point of view (PCP, DMA, and Cerberus), this memory segment allows accesses to the PMU Overlay memory (OVRAM), the DMI Local Data RAM (LDRAM), and the PMI scratch-pad RAM (SPRAM).

From the CPU point of view (PMI and DMI), this memory segment is reserved in the TC1766.

Segment 15

From the SPB point of view (PCP, DMA, and Cerberus), this memory segment allows accesses to all SFRs and 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 and CSFRs, the PCP memories, and the MLI transfer windows.

8.3 Address Map of the FPI Bus System

8.3.1 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
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	$8000\ 0000_H$ - $8017\ 7FFF_H$	1.5 Mbyte	Program Flash (PFLASH)	access	access[1)]
	$8017\ 8000_H$ - $807F\ FFFF_H$	6.5 Mbyte	Reserved	LMBBE & SPBBE	LMBBE
	$8080\ 0000_H$ - $8FDF\ FFFF_H$	246 Mbyte	Reserved	LMBBE & SPBBE	LMBBE
	$8FE0\ 0000_H$ - $8FE0\ 3FFF_H$	16 Kbyte	Data Flash (DFLASH) Bank 0	access	access[1)]
	$8FE0\ 4000_H$ - $8FE0\ FFFF_H$	48 Kbyte	Reserved	LMBBE & SPBBE	LMBBE
	$8FE1\ 0000_H$ - $8FE1\ 3FFF_H$	16 Kbyte	Data Flash (DFLASH) Bank 1	access	access[1)]
	$8FE1\ 4000_H$ - $8FE1\ FFFF_H$	48 Kbyte	Reserved	LMBBE & SPBBE	LMBBE
	$8FE2\ 0000_H$ - $8FF1\ FFFF_H$	1 Mbyte	Reserved		
	$8FF2\ 0000_H$ - $8FF5\ FFFF_H$	256 Kbyte	Reserved for TC1766 emulation device memory		
	$8FF6\ 0000_H$ - $8FFF\ BFFF_H$	624 Kbyte	Reserved		
	$8FFF\ C000_H$ - $8FFF\ FFFF_H$	16 Kbyte	Boot ROM (BROM)	access	

Table 8-2 SPB Address Map of Segment 0 to 14 (cont'd)

Segment	Address Range	Size	Description	Access Type	
				Read	Write
9	$9000\ 0000_H$ - $9FFF\ FFFF_H$	256 Mbyte	Reserved	SPBBE	SPBBE
10	$A000\ 0000_H$ - $A017\ FFFF_H$	1.5 Mbyte	Program Flash (PFLASH)	access	access[1)]
	$A017\ 8000_H$ - $A07F\ FFFF_H$	6.5 Mbyte	Reserved	LMBBE & SPBBE	LMBBE
	$A080\ 0000_H$ - $AFDF\ FFFF_H$	246 Mbyte	Reserved	LMBBE & SPBBE	LMBBE
	$AFE0\ 0000_H$ - $AFE0\ 3FFF_H$	16 Kbyte	Data Flash (DFLASH) Bank 0	access	access[1)]
	$AFE0\ 4000_H$ - $AFE0\ FFFF_H$	48 Kbyte	Reserved	LMBBE & SPBBE	LMBBE
	$AFE1\ 0000_H$ - $AFF1\ 3FFF_H$	16 Kbyte	Data Flash (DFLASH) Bank 1	access	access[1)]
	$AFE1\ 4000_H$ - $AFE1\ FFFF_H$	48 Kbyte	Reserved	LMBBE & SPBBE	ignore
	$AFE2\ 0000_H$ - $AFF1\ FFFF_H$	1 Mbyte	Reserved		
	$AFF2\ 0000_H$ - $AFF5\ FFFF_H$	256 Kbyte	Reserved for TC1766 emulation device memory		
	$AFF6\ 0000_H$ - $AFFF\ BFFF_H$	624 Kbyte	Reserved		
	$AFFF\ C000_H$ - $AFFF\ FFFF_H$	16 Kbyte	Boot ROM (BROM)	access	
11	$B000\ 0000_H$ - $BFFF\ FFFF_H$	256 Mbyte	Reserved	SPBBE	SPBBE
12	$C000\ 0000_H$ - $C000\ 1FFF_H$	8 Kbyte	Overlay memory (OVRAM)	SPBBE	SPBBE
	$C000\ 2000_H$ - $CFFF\ FFFF_H$	256 Mbyte	Reserved	SPBBE	SPBBE

Table 8-2 SPB Address Map of Segment 0 to 14 (cont'd)

Segment	Address Range	Size	Description	Access Type	
				Read	Write
13	D000 0000$_H$ - D000 DFFF$_H$	56 Kbyte	DMI Local Data RAM (LDRAM)	SPBBE	SPBBE
	D000 E000$_H$ - D3FF FFFF$_H$	64 Mbyte	Reserved	SPBBE	SPBBE
	D400 0000$_H$ - D400 3FFF$_H$	16 Kbyte	PMI Scratch-Pad RAM (SPRAM)	SPBBE	SPBBE
	D400 4000$_H$ - D7FF FFFF$_H$	≈ 64 Mbyte	Reserved	SPBBE	SPBBE
	D800 0000$_H$ - DEFF FFFF$_H$	112 Mbyte	Reserved	SPBBE	SPBBE
	DF00 0000$_H$ - DFFF FFEF$_H$	16 Mbyte	Reserved (for Boot Rom)	SPBBE	SPBBE
	DFFF FFF0$_H$ - DFFF FFFF$_H$	16 byte	microROM	SPBBE	SPBBE
14	E000 0000$_H$ - E7FF FFFF$_H$	128 MB	Reserved	LMBBE	LMBBE
	E800 0000$_H$ - E800 1FFF$_H$	8 Kbyte	Overlay memory (OVRAM)	access	access
	E800 2000$_H$ - E83F FFFF$_H$	≈ 4 Mbyte	Reserved	LMBBE	LMBBE
	E840 0000$_H$ - E840 DFFF$_H$	56 Kbyte	DMI Local Data RAM (LDRAM)	access	access
	E840 E000$_H$ - E84F FFFF$_H$	≈ 1 Mbyte	Reserved	LMBBE	LMBBE
	E850 0000$_H$ - E850 3FFF$_H$	16 Kbyte	PMI Scratch-Pad RAM (SPRAM)	access	access
	E850 4000$_H$ - E85F FFFF$_H$	≈ 1 Mbyte	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) Only applicable when writing Flash command sequences.

8.3.2 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. If an access is really addressing a correct address, it can be found in the detailed tables in Chapter 16.

Table 8-3 SPB Address Map of Segment 15

Unit	Address Range	Size	Access Type	
			Read	Write
System Control Unit (SCU) and Watchdog Timer (WDT)	$F000\ 0000_H$ - $F000\ 00FF_H$	256 byte	access	access
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
Reserved	$F000\ 0500_H$ - $F000\ 07FF_H$	–	SPBBE	SPBBE
MicroSecond Bus Controller 0 (MSC0)	$F000\ 0800_H$ - $F000\ 08FF_H$	256 byte	access	access
Reserved	$F000\ 0900_H$ - $F000\ 09FF_H$	–	SPBBE	SPBBE
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
Reserved	$F000\ 1200_H$ - $F000\ 12FF_H$	–	SPBBE	SPBBE
Reserved	$F000\ 1300_H$ - $F000\ 13FF_H$	–	SPBBE	SPBBE
Reserved	$F000\ 1400_H$ - $F000\ 14FF_H$	–	SPBBE	SPBBE
Reserved	$F000\ 1500_H$ - $F000\ 15FF_H$	–	SPBBE	SPBBE
Reserved	$F000\ 1600_H$ - $F000\ 16FF_H$	–	SPBBE	SPBBE
Reserved	$F000\ 1700_H$ - $F000\ 17FF_H$	–	SPBBE	SPBBE
General Purpose Timer Array 0 (GPTA0)	$F000\ 1800_H$ - $F000\ 1FFF_H$	8 × 256 byte	access	access
Reserved	$F000\ 2000_H$ - $F000\ 27FF_H$	–	SPBBE	SPBBE
Reserved	$F000\ 2800_H$ - $F000\ 2FFF_H$	–	SPBBE	SPBBE
Reserved	$F000\ 3000_H$ - $F000\ 3BFF_H$	–	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)	$F000\ 4000_H$ - $F000\ 5FFF_H$	8 Kbyte	access	access

Table 8-3 SPB Address Map of Segment 15 (cont'd)

Unit		Address Range	Size	Access Type	
				Read	Write
Reserved		F000 6000$_H$ - 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 1FFF$_H$	8 Kbyte	nE, 32	nE, 32
	Reserved	F005 2000$_H$ - F005 FFFF$_H$	–	SPBBE	SPBBE
	PCP Code Memory (PCODE)	F006 0000$_H$ - F006 2FFF$_H$	12 Kbyte	nE, 32	nE, 32
	Reserved	F006 3000$_H$ - F007 FFFF$_H$	–	SPBBE	SPBBE
Reserved		F008 0000$_H$ - F00F FFFF$_H$	–	SPBBE	SPBBE
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
Fast Analog-to-Digital Converter (FADC)		F010 0300$_H$ - F010 03FF$_H$	256 byte	access	access
Analog-to-Digital Converter 0 (ADC0)		F010 0400$_H$ - F010 05FF$_H$	2 × 256 byte	access	access
Reserved		F010 0600$_H$ - F010 07FF$_H$	–	SPBBE	SPBBE
Reserved		F010 0800$_H$ - F010 9FFF$_H$	–	SPBBE	SPBBE

Table 8-3 SPB Address Map of Segment 15 (cont'd)

Unit		Address Range	Size	Access Type	
				Read	Write
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
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		$F028\ 0000_H$ - $F7E0\ FEFF_H$	–	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
Reserved		$F800\ 0000_H$ - $F800\ 03FF_H$	–	SPBBE	SPBBE
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 (PMU)		$F800\ 0500_H$ - $F800\ 05FF_H$	256 byte	access	access
Reserved		$F800\ 0600_H$ - $F800\ 0FFF_H$	–	LMBBE & SPBBE	LMBBE
Flash Register		$F800\ 1000_H$ - $F800\ 23FF_H$	5 Kbyte	access	access
Reserved		$F800\ 2400_H$ - $F801\ 00FF_H$	–	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
Reserved		$F87F\ FB00_H$ - $F87F\ FBFF_H$	–	LMBBE & SPBBE	LMBBE
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

8.4 Address Map of the Local Memory Bus (LMB)

Table 8-4 shows the address map as seen from the LMB bus masters (PMI and DMI).

Table 8-4 LMB Address Map

Segment	Address Range	Size	Description	Action	
				Read	Write
0-7[1)]	$0000\ 0000_H$ - $0000\ 0007_H$	8 byte	Reserved (virtual address space)	MPN trap	MPN trap
	$0000\ 0008_H$ - $7FFF\ FFFF_H$	8 × 256 Mbyte		SPBBET	SPBBE
8[1)]	$8000\ 0000_H$ - $8017\ 7FFF_H$	1.5 Mbyte	Program Flash (PFLASH)	access	access[2)]
	$8017\ 8000_H$ - $807F\ FFFF_H$	6.5 Mbyte	Reserved	LMBBET	LMBBET
	$8080\ 0000_H$ - $8FDF\ FFFF_H$	246 Mbyte	Reserved	LMBBET	LMBBET
	$8FE0\ 0000_H$ - $8FE0\ 3FFF_H$	16 Kbyte	Data Flash (DFLASH) Bank 0	access	access[2)]
	$8FE0\ 4000_H$ - $8FE0\ FFFF_H$	48 Kbyte	Reserved	LMBBET	LMBBET
	$8FE1\ 0000_H$ - $8FE1\ 3FFF_H$	16 Kbyte	Data Flash (DFLASH) Bank 2	access	access[2)]
	$8FE1\ 4000_H$ - $8FE1\ FFFF_H$	48 Kbyte	Reserved	LMBBET	LMBBET
	$8FE2\ 0000_H$ - $8FF1\ FFFF_H$	1 Mbyte	Reserved		
	$8FF2\ 0000_H$ - $8FF5\ FFFF_H$	256 Kbyte	Reserved for TC1766 emulation device memory		
	$8FF6\ 0000_H$ - $8FFF\ BFFF_H$	624 Kbyte	Reserved		
	$8FFF\ C000_H$ - $8FFF\ FFFF_H$	16 Kbyte	Boot ROM (BROM)	access	
9[1)]	$9000\ 0000_H$ - $9FFF\ FFFF_H$	256 Mbyte	Reserved	SPBBET	SPBBE

Table 8-4 LMB Address Map (cont'd)

Seg-ment	Address Range	Size	Description	Action	
				Read	Write
10[3]	A000 0000$_H$ - A017 FFFF$_H$	1.5 Mbyte	Program Flash (PFLASH)	access	access[2]
	A017 8000$_H$ - A07F FFFF$_H$	6.5 Mbyte	Reserved	LMBBET	LMBBET
	A080 0000$_H$ - AFDF FFFF$_H$	246 Mbyte	Reserved	LMBBET	LMBBET
	AFE0 0000$_H$ - AFE0 3FFF$_H$	16 Kbyte	Data Flash (DFLASH) Bank 0	access	access[2]
	AFE0 4000$_H$ - AFE0 FFFF$_H$	48 Kbyte	Reserved	LMBBET	LMBBET
	AFE1 0000$_H$ - AFE1 3FFF$_H$	16 Kbyte	Data Flash (DFLASH) Bank 1	access	access[2]
	AFE1 4000$_H$ - AFE1 FFFF$_H$	48 Kbyte	Reserved	LMBBET	LMBBET
	AFE2 0000$_H$ - AFF1 FFFF$_H$	1 Mbyte	Reserved		
	AFF2 0000$_H$ - AFF5 FFFF$_H$	256 Kbyte	Reserved for TC1766 emulation device memory		
	AFF6 0000$_H$ - AFFF BFFF$_H$	624 Kbyte	Reserved		
	AFFF C000$_H$ - AFFF FFFF$_H$	16 Kbyte	Boot ROM (BROM)	access	
11[3]	B000 0000$_H$ - BFFF FFFF$_H$	256 Mbyte	Reserved	SPBBET	SPBBE
12[1]	C000 0000$_H$ - C000 1FFF$_H$	8 Kbyte	Overlay memory (OVRAM)	access	access
	C000 2000$_H$ - CFFF FFFF$_H$	256 Mbyte	Reserved	LMBBET	LMBBET

Table 8-4 LMB Address Map (cont'd)

Segment	Address Range	Size	Description	Action	
				Read	Write
13[3)]	D000 0000$_H$ - D000 DFFF$_H$	56 Kbyte	DMI Local Data RAM (LDRAM)	access SPBBE	access SPBBE
	D000 E000$_H$ - D3FF FFFF$_H$	64 Mbyte	Reserved	LMBBET	LMBBET
	D400 0000$_H$ - D400 3FFF$_H$	16 Kbyte	PMI Scratch-Pad RAM (SPRAM)	access	access
	D400 4000$_H$ - D7FF FFFF$_H$	≈ 64 Mbyte	Reserved	LMBBET	LMBBET
	D800 0000$_H$ - DEFF FFFF$_H$	112 Mbyte	Reserved		
	DF00 0000$_H$ - DFFF FFEF$_H$	16 Mbyte	Reserved (for Boot Rom)		
14[3)]	E000 0000$_H$ - E7FF FFFF$_H$	128 Mbyte	Reserved	LMBBET	LMBBET
	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 8-4**) Reserved areas give an bus error.	SPBBET	SPBBE

Table 8-4 LMB Address Map (cont'd)

Seg-ment	Address Range	Size	Description	Action	
				Read	Write
15	$F800\ 0000_H$ - $F800\ 03FF_H$	1 Kbyte	Reserved	LMBBET	LMBBET
	$F800\ 0400_H$ - $F800\ 04FF_H$	256 byte	Reserved	LMBBET	LMBBET
	$F800\ 0500_H$ - $F800\ 05FF_H$	256 byte	Program Memory Unit (PMU)	access	access
	$F800\ 0600_H$ - $F800\ 0FFF_H$	≈ 2 Kbyte	Reserved	LMBBET	LMBBET
	$F800\ 1000_H$ - $F800\ 23FF_H$	5 Kbyte	Flash Registers	access	access
	$F800\ 2400_H$ - $F87F\ FBFF_H$	≈ 8 Mbyte	Reserved	LMBBET	LMBBET
	$F87F\ FC00_H$ - $F87F\ FCFF_H$	256 byte	Data Memory Interface Unit	access	access
	$F87F\ FD00_H$ - $F87F\ FDFF_H$	256 byte	Program Memory Interface Unit	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) Cached area

2) Only applicable when writing Flash command sequences

3) Non-cached area

8.5 Memory Module Access Restrictions

Table 8-5 describes which type of accesses are possible to the different memories in the TC1766.

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	–	✔	–	✔	–	✔	✔	✔	✔
PCP[2]	CRAM	–	–	–	–	–	✔	✔	✔	✔
	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.

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

The TC1766 has 81 digital general purpose input/output (GPIO) port lines which are connected to the on-chip peripheral units. They are divided into:

- three 16-bit GPIO ports (Ports 0, 3, and 5)
- one 15-bit GPIO port (Port 1)
- one 14-bit GPIO port (Port 2)
- one 4-bit GPIO port (Port 4)

Additionally, 40 dedicated input/output lines without GPIO functionality are provided. **Figure 9-1** shows an overview of the I/O lines with its general port-to-peripheral unit assignment. Further details are described in the port specific sections of this chapter.

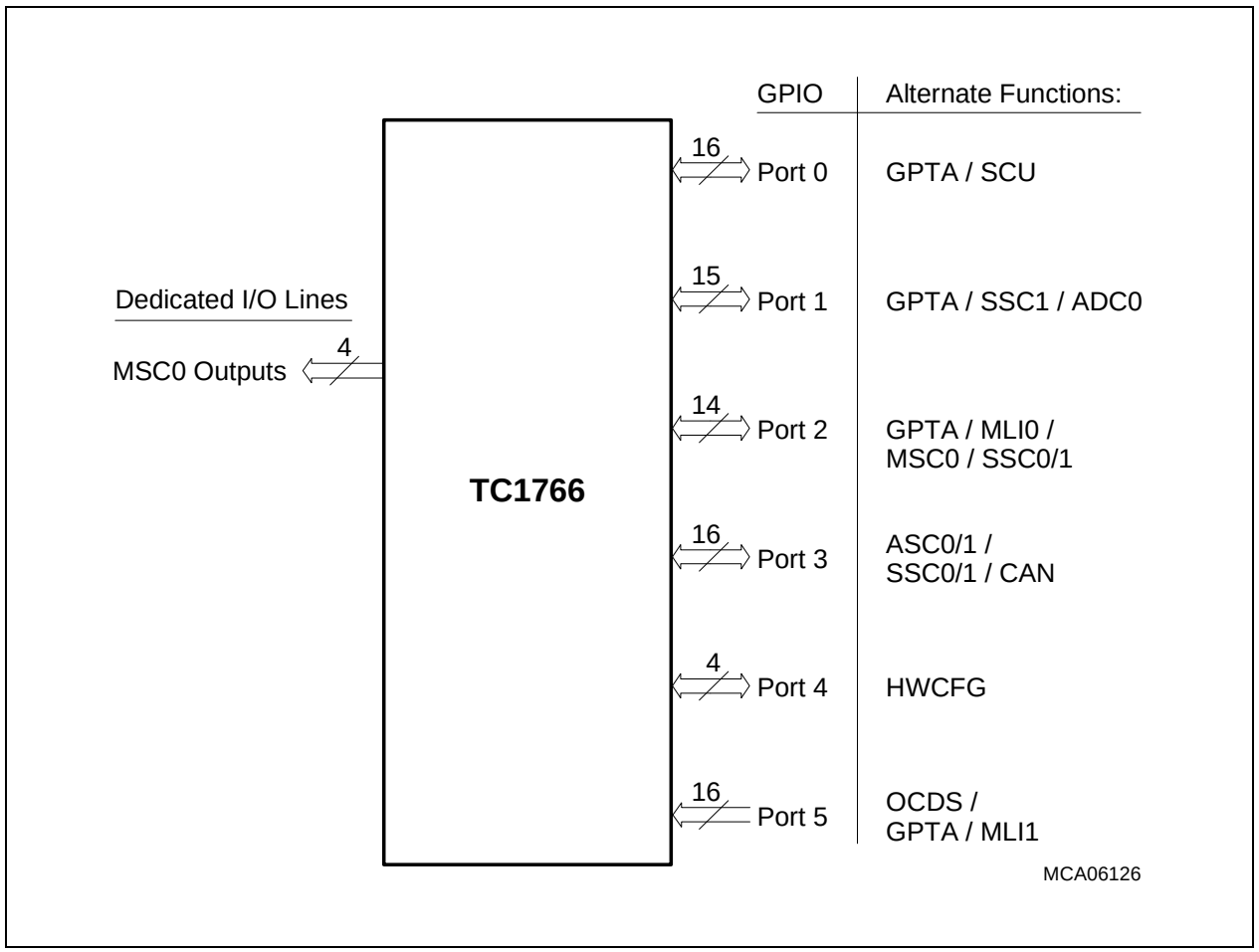


Figure 9-1 Peripheral/GPIO Lines of the TC1766

9.1 Basic Port Operation

Figure 9-2 shows a general block diagram of a TC1766 GPIO port line.

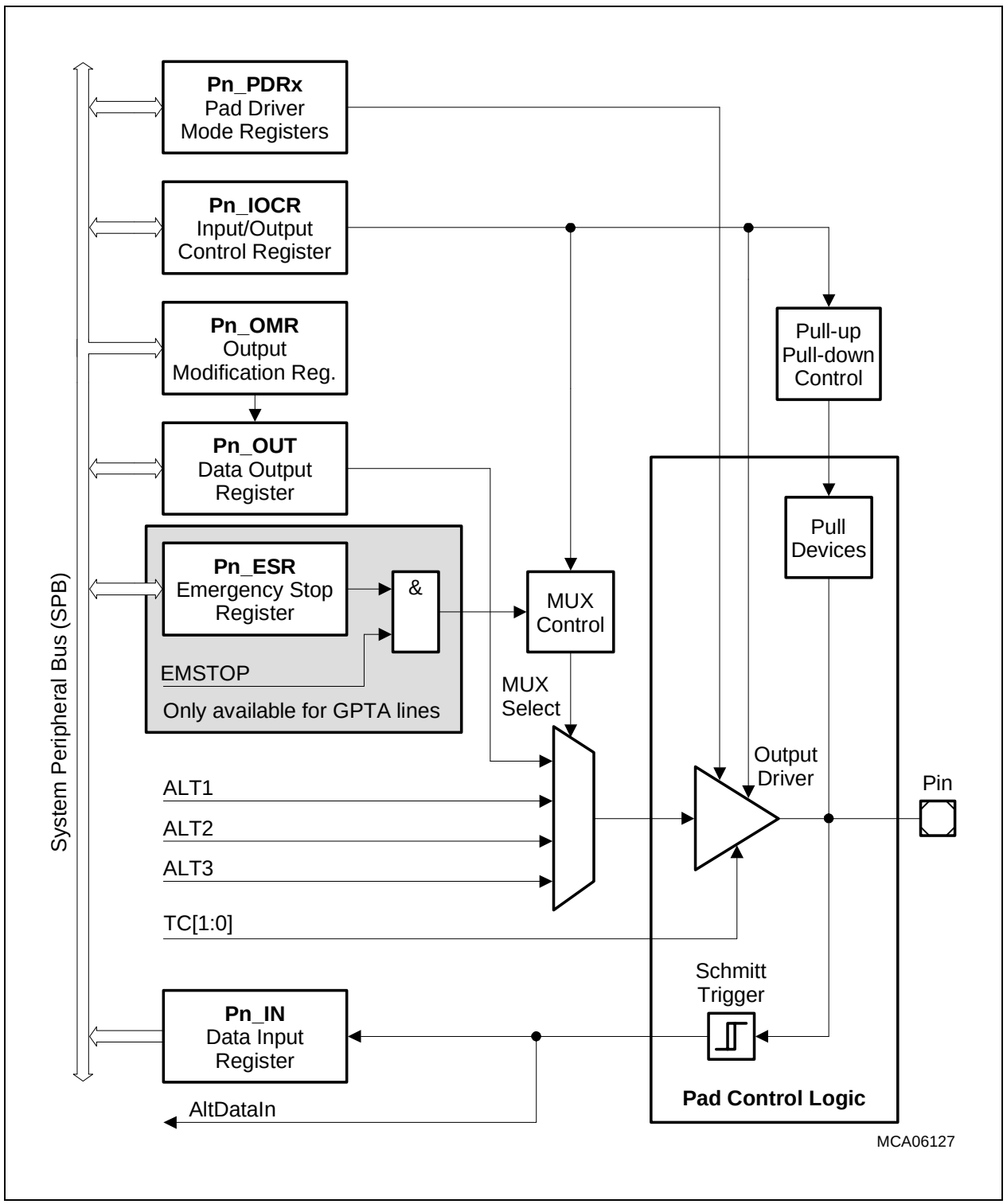


Figure 9-2 General Structure of a Port Pin

Each port line has a number of control and data bits, enabling very flexible usage of the line. Each port pin 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 logic 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 bi-directional I/O line, register Pn_IOCR has to be written for input or output selection. The Pn_IOCR register also 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 pin of a port 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, clear, 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 latch 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.

The temperature compensation feature for pad output drivers (two inputs lines TC[1:0]) allows the output driver characteristics to be controlled automatically within operating temperature range. The temperature compensation can be controlled for Class A2 output drivers at GPIO ports (Port 0 to Port 5).

All GPIO lines of the TC1766 that are used by the GPTA module 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 an 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 Port Register Description

The individual control and data bits of each GPIO port are implemented in a number of registers. Bits with the same meaning and function are assembled together in the same register. The registers are used to configure 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 9-6** on **Page 9-21** up to **Table 9-17** on **Page 9-61**.

Port Register Overview

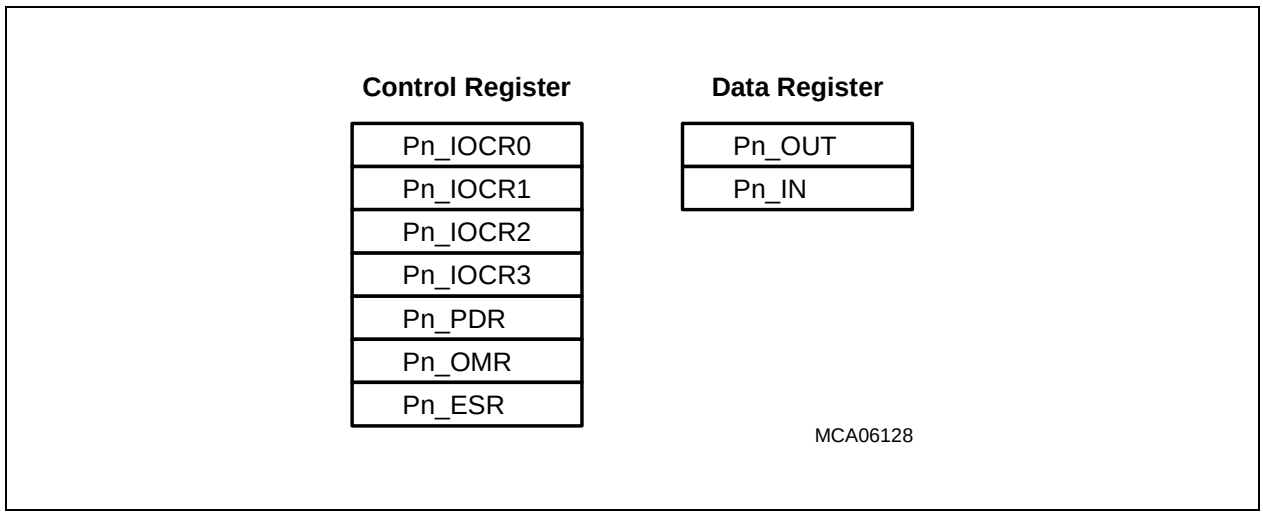


Figure 9-3 Port Registers

*Note: The complete address map of the GPIO ports is described in the **Chapter 16** "Register Overview" of this TC1766 System Units (Vol. 1 of 2) User's Manual.*

Table 9-1 Registers Address Space

Module	Base Address	End Address	Note
P0	$F000\ 0C00_H$	$F000\ 0CFF_H$	-
P1	$F000\ 0D00_H$	$F000\ 0DFF_H$	-
P2	$F000\ 0E00_H$	$F000\ 0EFF_H$	-
P3	$F000\ 0F00_H$	$F000\ 0FFF_H$	-
P4	$F000\ 1000_H$	$F000\ 10FF_H$	-
P5	$F000\ 1100_H$	$F000\ 11FF_H$	-

Table 9-2 Registers Overview

Register Short Name	Register Long Name	Offset Address	Description see
Pn_OUT	Port n Output Register	0000_H	Page 9-14
Pn_OMR	Port n Output Modification Register	0004_H	Page 9-15
Pn_IOCR0	Port n Input/Output Control Register 0	0010_H	Page 9-7
Pn_IOCR4	Port n Input/Output Control Register 4	0014_H	Page 9-8
Pn_IOCR8	Port n Input/Output Control Register 8	0018_H	Page 9-8
Pn_IOCR12	Port n Input/Output Control Register 12	$001C_H$	Page 9-9
Pn_IN	Port n Input Register	0024_H	Page 9-18
Pn_PDR	Port n Pad Driver Mode Register	0040_H	Page 9-11
Pn_ESR	Port n Emergency Stop Register	0050_H	Page 9-17

9.2.1 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
Port n Input/Output Control Register 0

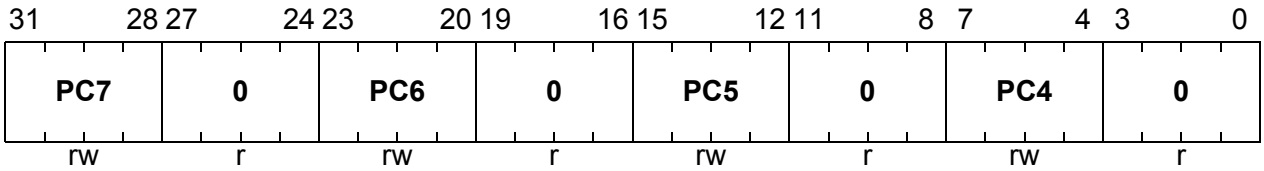
(10_H) **Reset Value: 2020 2020_H**

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
PCx (x = 0-3)	[7:4], [15:12], [23:20], [31:28]	rw	**Port Control for Port n Pin x** This bit field defines the Port n line x functionality according to the coding table (see **Table 9-3**).
0	[3:0], [11:8], [19:16], [27:24]	r	**Reserved** Read as 0; should be written with 0.

Pn_IOCR4
Port n Input/Output Control Register 4

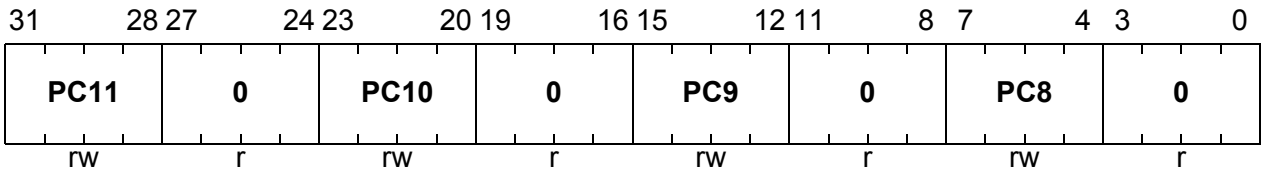
(14_H) Reset Value: 2020 2020_H



Field	Bits	Type	Description
PCx (x = 4-7)	[7:4], [15:12], [23:20], [31:28]	rw	**Port Control for Port n Pin x** This bit field defines the Port n line x functionality according to the coding table (see **Table 9-3**).
0	[3:0], [11:8], [19:16], [27:24]	r	**Reserved** Read as 0; should be written with 0.

Pn_IOCR8
Port n Input/Output Control Register 8

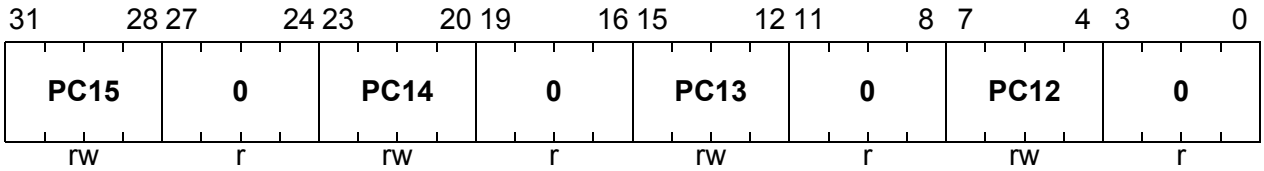
(18_H) Reset Value: 2020 2020_H



Field	Bits	Type	Description
PCx (x = 8-11)	[7:4], [15:12], [23:20], [31:28]	rw	**Port Control for Port n Pin x** This bit field defines the Port n line x functionality according to the coding table (see **Table 9-3**).
0	[3:0], [11:8], [19:16], [27:24]	r	**Reserved** Read as 0; should be written with 0.

Pn_IOCR12
Port n Input/Output Control Register 12

($1C_H$) **Reset Value: 2020 2020_H**



Field	Bits	Type	Description
PCx (x = 12-15)	[7:4], [15:12], [23:20], [31:28]	rw	**Port Control for Port n Pin x** This bit field defines the Port n line x functionality according to the coding table (see **Table 9-3**).
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 9-3 describes the coding of the PCx bit fields that determine the port line functionality.

Table 9-3 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
$0X10_B$ 1)			Input pull-up device connected
$0X11_B$			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 bit field value is default after reset.

9.2.2 Pad Driver Mode Register

Overview

The pad structure of the TC1766 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 of 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 consists of two classes of pads:

Class A1 pins (low speed 3.3 V LVTTL outputs)
Class A2 pins (high speed 3.3 V 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 (**Figure 9-4** to **Figure 9-9**). Further details about pad driver classes that are available in the TC1766 are summarized in **Table 1-4** on **Page 1-47**.

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 9-4**. 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 case of strong driver type, the signal transition edge can be additionally selected as sharp/medium/soft.

Table 9-4 Pad Driver Mode Selection

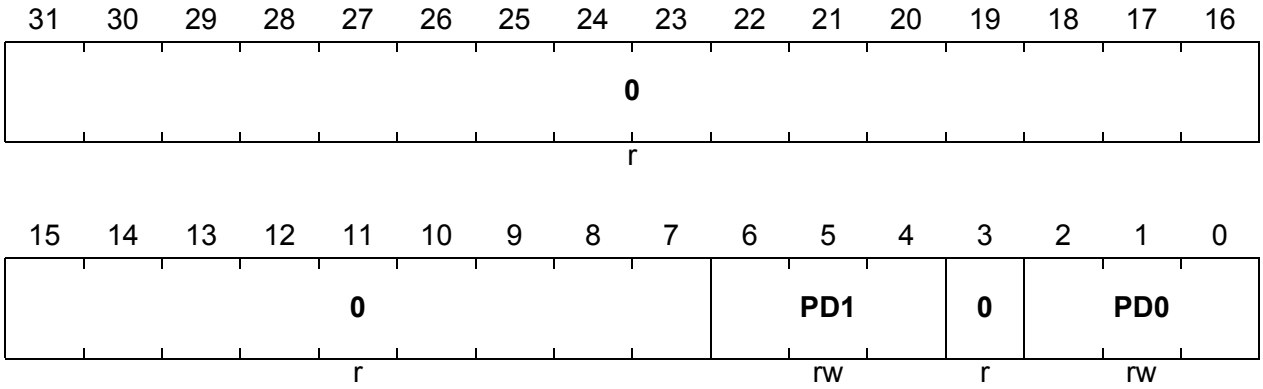
Pad Class	PDx.2	PDx.1	PDx.0	Functionality
A1	X	X	0	Medium driver selected
			1	Weak driver selected
A2	0	0	0	Strong driver, sharp edge selected[1)]
	0	0	1	Strong driver, medium edge selected[1)]
	0	1	0	Strong driver, soft edge selected[1)]
	0	1	1	Weak driver selected
	1	0	0	Medium driver selected
	1	0	1	
	1	1	0	
	1	1	1	Weak driver selected

1) In strong driver mode, the output driver characteristics of class A2 pads can be additionally controlled by a temperature compensation logic (see **Page 5-42**).

Note: Please refer to the TC1766 Data Sheet for detailed DC characteristics of class A1 and class A2 pads.

Pad Driver Mode Register Example: Port 0 Pad Driver Mode Register

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; coding see **Page 9-11**)
PD1	[6:4]	rw	**Pad Driver Mode for P0.[15:8]** (Class A1 pads; coding see **Page 9-11**)
0	3, [31:7]	r	**Reserved** Read as 0; should be written with 0.

In addition to the pad driver mode register control selections, a temperature compensation logic, which is a part of the system control unit (SCU), makes it possible to adjust the edges of the high-speed class A2 GPIO output lines depending on the die temperature. The temperature compensation logic affects all class A2 GPIO output lines that are used simultaneously as output.

The fully programmable temperature compensation logic provides four types of output driver levels:

- Maximum level (low temperature)
- High level
- Low level
- Minimum level (high temperature)

Further details of the temperature compensation logic are described in **Section 5.6** on **Page 5-42** (SCU) of this User's Manual.

9.2.3 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 also be set/cleared by writing appropriate values into the port output modification register Pn_OMR.

Pn_OUT
Port n Output Register (00_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 is Pn.x is 1. Pn.x can also be set/cleared by control bits of the Pn_OMR register.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

Note: Only Ports 0, 3, and 5 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).

9.2.4 Port Output Modification Register

The port output modification register contains control bits that make it possible to individually set, clear, or toggle the logic state of a single port line by manipulating the output register.

Pn_OMR
Port n Output Modification Register (04_H) Reset Value: 0000 XXXX$_H$

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 9-5**.
PRx (x = 0-15)	x + 16	w	**Port n Reset Bit x** Setting this bit will clear or toggle the corresponding bit in the port output register Pn_OUT. The function of this bit is shown in **Table 9-5**.

Note: Register Pn_OMR is virtual and does not contain any flip-flop. A read action delivers the value of the corresponding port input register Pn_IN. Therefore, reset values of Pn_OMR and Pn_IN registers are also identical.
The Pn_OMR registers should always be written using word accesses.

Table 9-5 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.

Table 9-5 Function of the Bits PRx and PSx (cont'd)

PRx	PSx	Function
1	0	Bit Pn_OUT.Px is cleared.
1	1	Bit Pn_OUT.Px is toggled.

9.2.5 Emergency Stop Register

All GPIO lines which are used by the GPTA module have an emergency stop logic (see **Figure 9-2**). These GPTA related GPIO lines are:

- P0.[15:0], P1.[11:0], P2.[7:0], P4.[3:0], and P5.[7:0]

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
Port n Emergency Stop Register (50_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.

Field	Bits	Type	Description
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

Note: Only Ports 0, 3, and 5 are 16-bit wide ports. The Pn_ESR registers of the other ports have a reduced number of bits (see Pn_ESR register descriptions in the corresponding port sections).

9.2.6 Port Input Register

The logic level of a GPIO pin can be read via the read-only port input register Pn_IN. When a read of Pn_IN occurs, the values at the GPIO pins are latched in Pn_IN independently of whether the GPIO pin is selected as input or output.

Pn_IN
Port n Input Register **(24_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 pin Pn.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 Ports 0, 3, and 5 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).

9.3 Port 0

This section describes the Port 0 functionality in detail.

9.3.1 Port 0 Configuration

Port 0 is a general-purpose 16-bit bi-directional port that can be alternatively used for the GPTA and SCU modules. The logic state of the sixteen Port 0 lines is stored in the Software Configuration Latched Input Register SCU_SCLIR at the rising edge of HDRST and can be read by software later. Note that some of the P0.[7:0] lines are used for configuration purposes too (see **Page 9-28**).

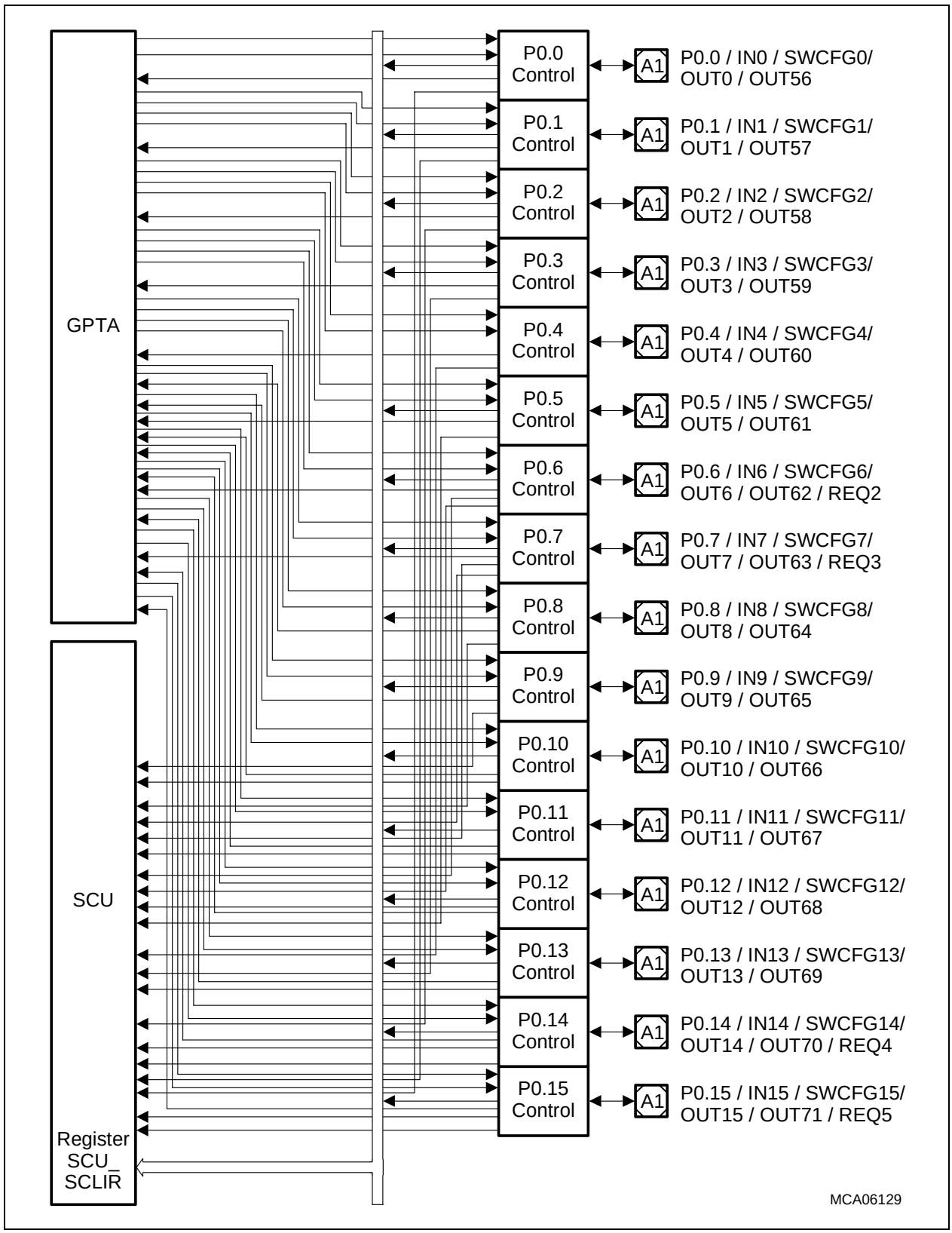


Figure 9-4 Port 0 Configuration Diagram

9.3.2 Port 0 Function Table

Table 9-6 summarizes the I/O control selection functions of each Port 0 line.

Table 9-6 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	$0XXX_B$
		GPTA input	IN0		
		SCU input	SWCFG0		
	O	General-purpose output	P0_OUT.P0		$1X00_B$
		GPTA output	OUT0		$1X01_B$
		GPTA output	OUT56		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P0.1	I	General-purpose input	P0_IN.P1	P0_IOCR0.PC1	$0XXX_B$
		GPTA input	IN1		
		SCU input	SWCFG1		
	O	General-purpose output	P0_OUT.P1		$1X00_B$
		GPTA output	OUT1		$1X01_B$
		GPTA output	OUT57		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P0.2	I	General-purpose input	P0_IN.P2	P0_IOCR0.PC2	$0XXX_B$
		GPTA input	IN2		
		SCU input	SWCFG2		
	O	General-purpose output	P0_OUT.P2		$1X00_B$
		GPTA output	OUT2		$1X01_B$
		GPTA output	OUT58		$1X10_B$
		Reserved[1)]	–		$1X11_B$

Table 9-6 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
P0.3	I	General-purpose input	P0_IN.P3	P0_IOCR0.PC3	$0XXX_B$
		GPTA input	IN3		
		SCU input	SWCFG3		
	O	General-purpose output	P0_OUT.P3		$1X00_B$
		GPTA output	OUT3		$1X01_B$
		GPTA output	OUT59		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P0.4	I	General-purpose input	P0_IN.P4	P0_IOCR4.PC4	$0XXX_B$
		GPTA input	IN4		
		SCU input	SWCFG4		
	O	General-purpose output	P0_OUT.P4		$1X00_B$
		GPTA output	OUT4		$1X01_B$
		GPTA output	OUT60		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P0.5	I	General-purpose input	P0_IN.P5	P0_IOCR4.PC5	$0XXX_B$
		GPTA input	IN5		
		SCU input	SWCFG5		
	O	General-purpose output	P0_OUT.P5		$1X00_B$
		GPTA output	OUT5		$1X01_B$
		GPTA output	OUT61		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P0.6	I	General-purpose input	P0_IN.P6	P0_IOCR4.PC6	$0XXX_B$
		GPTA input	IN6		
		SCU input	SWCFG6		
		SCU input	REQ2		
	O	General-purpose output	P0_OUT.P6		$1X00_B$
		GPTA output	OUT6		$1X01_B$
		GPTA output	OUT62		$1X10_B$
		Reserved[1)]	–		$1X11_B$

Table 9-6 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
P0.7	I	General-purpose input	P0_IN.P7	P0_IOCR4.PC7	$0XXX_B$
		GPTA input	IN7		
		SCU input	SWCFG7		
		SCU input	REQ3		
	O	General-purpose output	P0_OUT.P7		$1X00_B$
		GPTA output	OUT7		$1X01_B$
		GPTA output	OUT63		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P0.8	I	General-purpose input	P0_IN.P8	P0_IOCR8.PC8	$0XXX_B$
		GPTA input	IN8		
		SCU input	SWCFG8		
	O	General-purpose output	P0_OUT.P8		$1X00_B$
		GPTA output	OUT8		$1X01_B$
		GPTA output	OUT64		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P0.9	I	General-purpose input	P0_IN.P9	P0_IOCR8.PC9	$0XXX_B$
		GPTA input	IN9		
		SCU input	SWCFG9		
	O	General-purpose output	P0_OUT.P9		$1X00_B$
		GPTA output	OUT9		$1X01_B$
		GPTA output	OUT65		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P0.10	I	General-purpose input	P0_IN.P10	P0_IOCR8.PC10	$0XXX_B$
		GPTA input	IN10		
		SCU input	SWCFG10		
	O	General-purpose output	P0_OUT.P10		$1X00_B$
		GPTA output	OUT10		$1X01_B$
		GPTA output	OUT66		$1X10_B$
		Reserved[1)]	–		$1X11_B$

Table 9-6 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
P0.11	I	General-purpose input	P0_IN.P11	P0_IOCR8.PC11	$0XXX_B$
		GPTA input	IN11		
		SCU input	SWCFG11		
	O	General-purpose output	P0_OUT.P11		$1X00_B$
		GPTA output	OUT11		$1X01_B$
		GPTA output	OUT67		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P0.12	I	General-purpose input	P0_IN.P12	P0_IOCR12.PC12	$0XXX_B$
		GPTA input	IN12		
		SCU input	SWCFG12		
	O	General-purpose output	P0_OUT.P12		$1X00_B$
		GPTA output	OUT12		$1X01_B$
		GPTA output	OUT68		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P0.13	I	General-purpose input	P0_IN.P13	P0_IOCR12.PC13	$0XXX_B$
		GPTA input	IN13		
		SCU input	SWCFG13		
	O	General-purpose output	P0_OUT.P13		$1X00_B$
		GPTA output	OUT13		$1X01_B$
		GPTA output	OUT69		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P0.14	I	General-purpose input	P0_IN.P14	P0_IOCR12.PC14	$0XXX_B$
		GPTA input	IN14		
		SCU input	SWCFG14		
		SCU input	REQ4		
	O	General-purpose output	P0_OUT.P14		$1X00_B$
		GPTA output	OUT14		$1X01_B$
		GPTA output	OUT70		$1X10_B$
		Reserved[1)]	–		$1X11_B$

Table 9-6 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
P0.15	I	General-purpose input	P0_IN.P15	P0_IOCR12.PC15	$0XXX_B$
		GPTA input	IN15		
		SCU input	SWCFG15		
		SCU input	REQ5		
	O	General-purpose output	P0_OUT.P15		$1X00_B$
		GPTA output	OUT15		$1X01_B$
		GPTA output	OUT71		$1X10_B$
		Reserved[1)]	–		$1X11_B$

1) The port I/O control values P0_IOCRx.Py that are assigned to this reserved alternate output control selection should not be used. Otherwise, unpredictable output port line behavior may occur.

9.3.3 Port 0 Registers

The following registers are available on Port 0:

Table 9-7 Port 0 Registers

Register Short Name	Register Long Name	Offset Address	Description see
P0_OUT	Port 0 Output Register	0000_H	Page 9-14
P0_OMR	Port 0 Output Modification Register	0004_H	Page 9-15
P0_IOCR0	Port 0 Input/Output Control Register 0	0010_H	Page 9-7
P0_IOCR4	Port 0 Input/Output Control Register 4	0014_H	Page 9-8
P0_IOCR8	Port 0 Input/Output Control Register 8	0018_H	Page 9-8
P0_IOCR12	Port 0 Input/Output Control Register 12	$001C_H$	Page 9-9
P0_IN	Port 0 Input Register	0024_H	Page 9-18
P0_PDR	Port 0 Pad Driver Mode Register	0040_H	Page 9-26[1)]
P0_ESR	Port 0 Emergency Stop Register	0050_H	Page 9-28
SCU_SCLIR	SCU Software Configuration Latched Inputs Register	[2)]	Page 9-27

1) This register is listed here in the Port 0 section because they differ from the general port register description given in Section 9.2.

2) This register is located in the address range of the SCU.

*Note: The complete address map of Port 0 is described in **Table 16-9** on **Page 16-20** of this TC1766 System Units (Vol. 1 of 2) User's Manual.*

9.3.3.1 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. The Port 0 port lines are all class A1 pads (see also **Figure 9-4**).

P0_PDR
Port 0 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															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0									PD1			0	PD0		
r									rw			r	rw		

Field	Bits	Type	Description
PD0	[2:0]	rw	**Pad Driver Mode for P0.[7:0]** (Class A1 pads; coding see **Page 9-11**)
PD1	[6:4]	rw	**Pad Driver Mode for P0.[15:8]** (Class A1 pads; coding see **Page 9-11**)
0	3, [31:7]	r	**Reserved** Read as 0; should be written with 0.

9.3.3.2 Port 0 Software Configuration Selection

The logic levels of the sixteen Port 0 lines are latched into the register SCU_SCLIR (SCU Software Configuration Latched Inputs Register) when the hardware reset goes inactive (at the rising edge of HDRST). This feature makes it possible to use Port 0 lines for software selection purposes.

Depending on the TC1766 device used, several SWOPT bits (meaning several P0 lines) are reserved and may not be used by a user program for software configuration selection purposes. For details, see **Section 9.3.3.3**.

SCU_SCLIR
SCU Software Configuration Latched Inputs Register
($F0000038_H$) **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
SWOPT15	SWOPT14	SWOPT13	SWOPT12	SWOPT11	SWOPT10	SWOPT9	SWOPT8	SWOPT7	SWOPT6	SWOPT5	SWOPT4	SWOPT3	SWOPT2	SWOPT1	SWOPT0
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Function
SWOPTx (x = 0-15)	x	rh	**Software Configuration Bits** These bits show the state of pin P0.x that was latched with the last rising edge of HDRST. 0_B P0.x state/logic level latched by HDRST is 0. 1_B P0.x state/logic level latched by HDRST is 1.
0	[31:16]	r	**Reserved** Read as 0.

Note: The reset value (bits "X") of the register SCU_SCLIR is defined by the circuitry connected to Port 0 at the rising edge of HDRST. Port 0 lines are set to inputs with pull-up devices connected (reset values of port input/output control registers).

9.3.3.3 Reserved Port 0 Pins

Depending on the TC1766 device version used in an application, several Port 0 lines (meaning several SWOPT bits) are reserved and cannot be used for user system purposes during a HDRST reset operation. **Table 9-8** defines the reserved Port 0 lines (indicated by 0 or 1) as well as the Port 0 lines that can be used by a user program (indicated by “user”) for software configuration selection (or as GPIO pins) depending on the specific TC1766 device version.

Table 9-8 Reserved Port 0 Lines of TC1766 Devices

TC1766 Device Version	SWOPTx Bits (x = 0-15)					
	P0. [15:8]	P0. [7:6]	P0.5	P0.4	P0.3	P0. [2:0]
TC1766	user	user	user	1	user	1)
TC1766ED (Emulation Device)		XX_B 2)	0 or 1 3)			

1)The P0.[2:0] bits are only used in alternate boot modes (see Table 4-7). If alternate boot modes are not required or used in an application, P0.[2:0] can also be used for user program software configuration selection purposes during a hardware reset operation or as GPIO pins.

2) 00_B, 11_B : The USB interface of the TC1766ED is not connected to device pins.
01_B : The USB interface of the TC1766ED is connected to JTAG I/O lines.
10_B : The USB interface of the TC1766ED is connected to P2.[5:0] lines.

3) 0: Emulation device functionality is not available.
1: Emulation device functionality is fully supported.

9.3.3.4 Port 0 Emergency Stop Register

The basic P0_ESR register functionality is described on **Page 9-17**. At Port 0, port lines P0.[15:0] are connected to GPTA I/O lines. Therefore, all the port lines of Port 0 can be controlled for the emergency stop function.

9.4 Port 1

This section describes the Port 1 functionality in detail.

9.4.1 Port 1 Configuration

Port 1 is a 15-bit bi-directional general-purpose I/O port that can be alternatively used for the GPTA I/O lines, SSC1 and ADC0 interfaces.

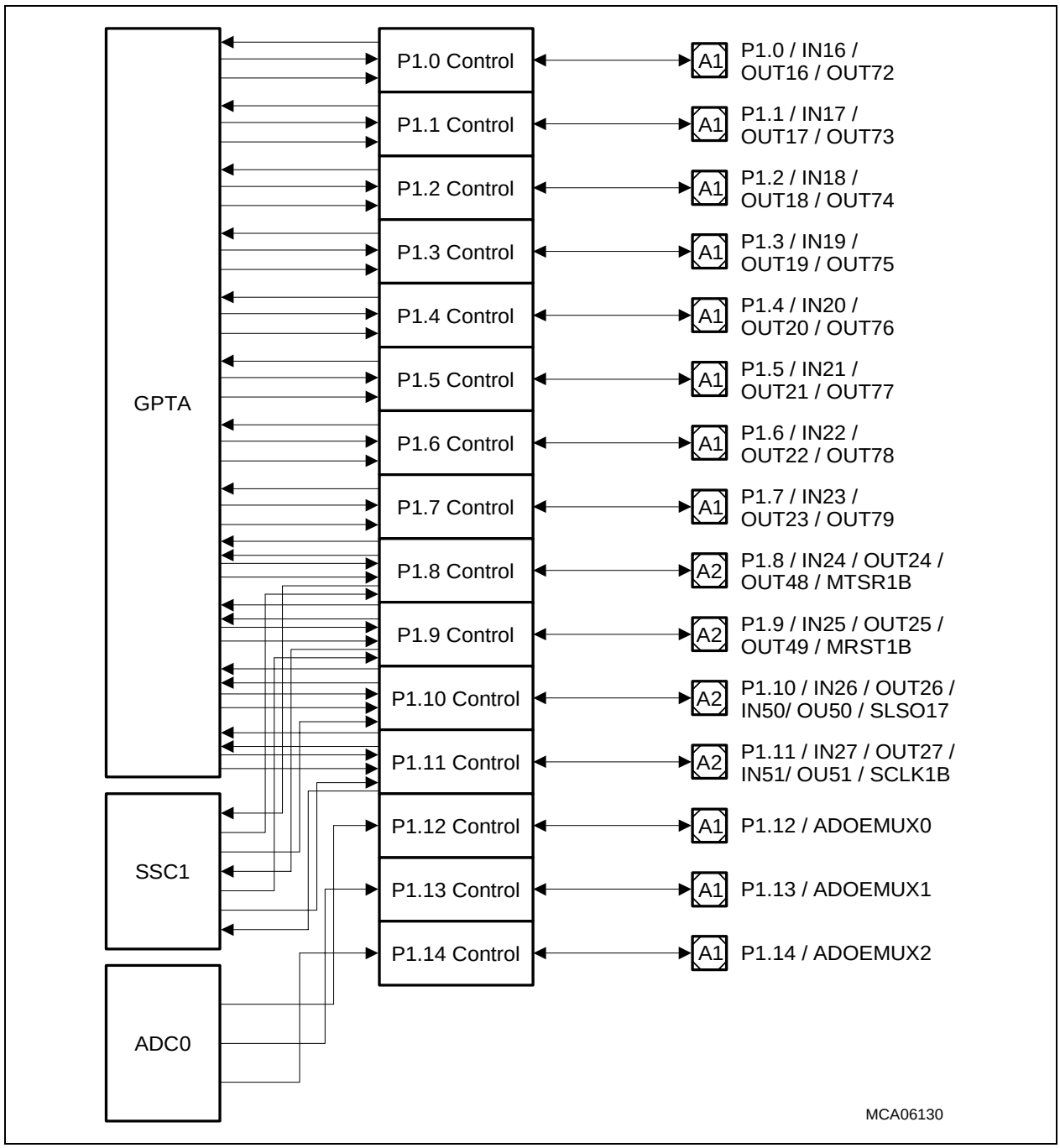


Figure 9-5 Port 1 Configuration Diagram

9.4.2 Port 1 Function Table

Table 9-9 summarizes the I/O control selection functions of each Port 1 line.

Table 9-9 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	$0XXX_B$
		GPTA input	IN16		
	O	General-purpose output	P1_OUT.P0		$1X00_B$
		GPTA output	OUT16		$1X01_B$
		GPTA output	OUT72		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P1.1	I	General-purpose input	P1_IN.P1	P1_IOCR0.PC1	$0XXX_B$
		GPTA input	IN17		
	O	General-purpose output	P1_OUT.P1		$1X00_B$
		GPTA output	OUT17		$1X01_B$
		GPTA output	OUT73		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P1.2	I	General-purpose input	P1_IN.P2	P1_IOCR0.PC2	$0XXX_B$
		GPTA input	IN18		
	O	General-purpose output	P1_OUT.P2		$1X00_B$
		GPTA output	OUT18		$1X01_B$
		GPTA output	OUT74		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P1.3	I	General-purpose input	P1_IN.P3	P1_IOCR0.PC3	$0XXX_B$
		GPTA input	IN19		
	O	General-purpose output	P1_OUT.P3		$1X00_B$
		GPTA output	OUT19		$1X01_B$
		GPTA output	OUT75		$1X10_B$
		Reserved[1)]	–		$1X11_B$

Table 9-9 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	$0XXX_B$
		GPTA input	IN20		
		SCU input	EMGSTOP		
	O	General-purpose output	P1_OUT.P4		$1X00_B$
		GPTA output	OUT20		$1X01_B$
		GPTA output	OUT76		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P1.5	I	General-purpose input	P1_IN.P5	P1_IOCR4.PC5	$0XXX_B$
		GPTA input	IN21		
	O	General-purpose output	P1_OUT.P5		$1X00_B$
		GPTA output	OUT21		$1X01_B$
		GPTA output	OUT77		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P1.6	I	General-purpose input	P1_IN.P6	P1_IOCR4.PC6	$0XXX_B$
		GPTA input	IN22		
	O	General-purpose output	P1_OUT.P6		$1X00_B$
		GPTA output	OUT22		$1X01_B$
		GPTA output	OUT78		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P1.7	I	General-purpose input	P1_IN.P7	P1_IOCR4.PC7	$0XXX_B$
		GPTA input	IN23		
	O	General-purpose output	P1_OUT.P7		$1X00_B$
		GPTA output	OUT23		$1X01_B$
		GPTA output	OUT79		$1X10_B$
		Reserved[1)]	–		$1X11_B$

Table 9-9 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
P1.8	I	General-purpose input	P1_IN.P8	P1_IOCR8.PC8	$0XXX_B$
		GPTA input	IN24		
		GPTA input	IN48		
		SSC1input (Slave Mode)	MTSR1B		
	O	General-purpose output	P1_OUT.P8		$1X00_B$
		GPTA output	OUT24		$1X01_B$
		GPTA output	OUT48		$1X10_B$
		SSC1 output (Master Mode)	MTSR1B		$1X11_B$
P1.9	I	General-purpose input	P1_IN.P9	P1_IOCR8.PC9	$0XXX_B$
		GPTA input	IN25		
		GPTA input	IN49		
		SSC1input (Master Mode)	MRST1B		
	O	General-purpose output	P1_OUT.P9		$1X00_B$
		GPTA output	OUT25		$1X01_B$
		GPTA output	OUT49		$1X10_B$
		SSC1 output (Slave Mode)	MRST1B		$1X11_B$
P1.10	I	General-purpose input	P1_IN.P10	P1_IOCR8.PC10	$0XXX_B$
		GPTA input	IN26		
		GPTA input	IN50		
	O	General-purpose output	P1_OUT.P10		$1X00_B$
		GPTA output	OUT26		$1X01_B$
		GPTA output	OUT50		$1X10_B$
		SSC1 output	SLSO17		$1X11_B$

Table 9-9 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
P1.11	I	General-purpose input	P1_IN.P11	P1_IOCR8.PC11	$0XXX_B$
		GPTA input	IN27		
		GPTA input	IN51		
		SSC1input	SCLK1B		
	O	General-purpose output	P1_OUT.P11		$1X00_B$
		GPTA output	OUT27		$1X01_B$
		GPTA output	OUT51		$1X10_B$
		SSC1 output	SCLK1B		$1X11_B$
P1.12	I	General-purpose input	P1_IN.P12	P1_IOCR12.PC12	$0XXX_B$
	O	General-purpose output	P1_OUT.P12		$1X00_B$
		ADC0[2)]	AD0EMUX0		$1X01_B$
		ADC0	AD0EMUX0		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P1.13	I	General-purpose input	P1_IN.P13	P1_IOCR12.PC13	$0XXX_B$
	O	General-purpose output	P1_OUT.P13		$1X00_B$
		ADC0[2)]	AD0EMUX1		$1X01_B$
		ADC0	AD0EMUX1		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P1.14	I	General-purpose input	P1_IN.P14	P1_IOCR12.PC14	$0XXX_B$
	O	General-purpose output	P1_OUT.P14		$1X00_B$
		ADC0[2)]	AD0EMUX2		$1X01_B$
		ADC0	AD0EMUX2		$1X10_B$
		Reserved[1)]	–		$1X11_B$

1) The port I/O control values P1_IOCRx.Py that are assigned to this reserved alternate output control selection should not be used. Otherwise, unpredictable output port line behavior may occur.

2) The ALT1 and ALT2 for this pin are connected together. There are no dependencies. Either one can be chosen.

9.4.3 Port 1 Registers

The following registers are available on Port 1:

Table 9-10 Port 1 Registers

Register Short Name	Register Long Name	Offset Address	Description see
P1_OUT	Port 1 Output Register	0000_H	Page 9-34 1)
P1_OMR	Port 1 Output Modification Register	0004_H	Page 9-34 1)
P1_IOCR0	Port 1 Input/Output Control Register 0	0010_H	Page 9-7
P1_IOCR4	Port 1 Input/Output Control Register 4	0014_H	Page 9-8
P1_IOCR8	Port 1 Input/Output Control Register 8	0018_H	Page 9-8
P1_IOCR12	Port 1 Input/Output Control Register 12	$001C_H$	Page 9-35 1)
P1_IN	Port 1 Input Register	0024_H	Page 9-35 1)
P1_PDR	Port 1 Pad Driver Mode Register	0040_H	Page 9-36 1)
P1_ESR	Port 1 Emergency Stop Register	0050_H	Page 9-35 1)

1) This register is listed here in the Port 1 section because they differ from the general port register description given in Section 9.2.

Note: The complete address map of Port 1 is described in Table 16-10 on Page 16-21 of this TC1766 System Units (Vol. 1 of 2) User's Manual.

9.4.3.1 Port 1 Output Register

The basic P1_OUT register functionality is described on Page 9-14. Port line P1.15 is not connected to port lines. Therefore, reading the P1_OUT bits P1.15 returns the value that was last written (0 after reset). These bits can also be set/cleared by the corresponding P1_OMR bits.

9.4.3.2 Port 1 Output Modification Register

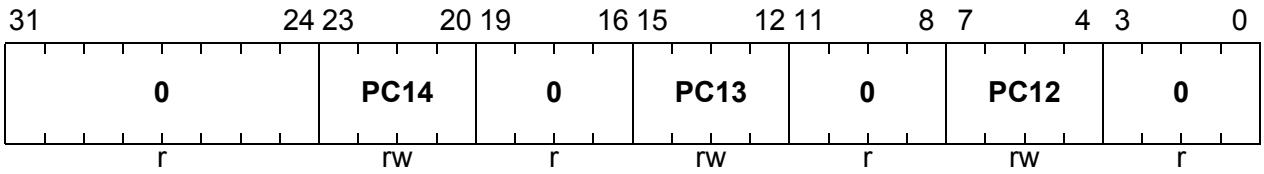
The basic P1_OMR register functionality is described on Page 9-15. However, Port line P1.15 is not available. Therefore, the P1_OMR bits PS.15 and PR.15 have no direct effect on port lines but only on register bit P1_OUT.P15.

9.4.3.3 Port 1 Input/Output Control Register 12

Port line P1.15 is not available. Therefore, the PC15 bit field in register P1_IOCR12 is not connected to any port lines.

P1_IOCR12
Port 1 Input/Output Control Register 12

($1C_H$) **Reset Value: 0020 2020_H**



Field	Bits	Type	Description
PC12	[7:4]	rw	**Port Control for Port 1.12** (coding see **Table 9-3** on **Page 9-10**)
PC13	[15:12]	rw	**Port Control for Port 1.13** (coding see **Table 9-3** on **Page 9-10**)
PC14	[23:20]	rw	**Port Control for Port 1.14** (coding see **Table 9-3** on **Page 9-10**)
0	[3:0], [11:8], [19:16], [31:24]	r	**Reserved** Read as 0; should be written with 0.

9.4.3.4 Port 1 Input Register

The basic P1_IN register functionality is described on **Page 9-18**. However, port line P1.15 is not available. Therefore, bits P15 in register P1_IN are always read as 0.

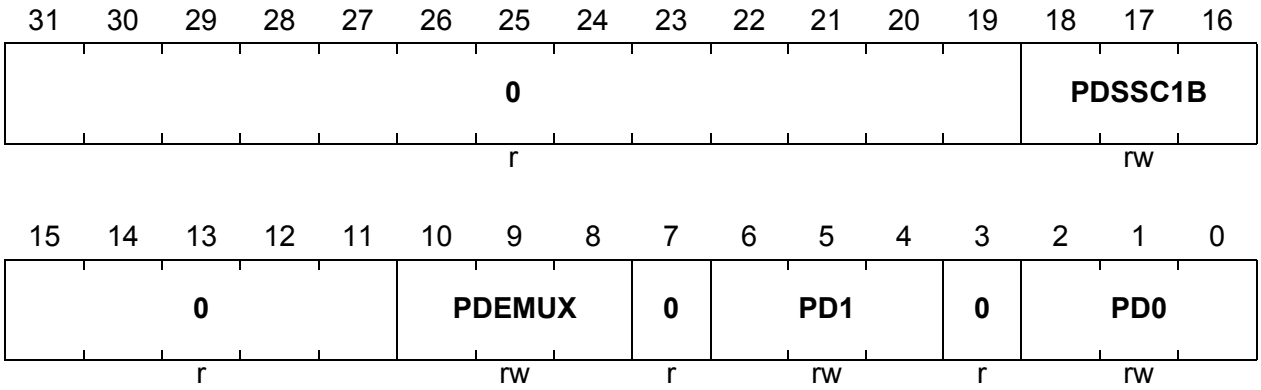
9.4.3.5 Port 1 Emergency Stop Register

The basic P1_ESR register functionality is described on **Page 9-17**. At Port 1, only port lines P1.[11:0] are connected to GPTA I/O lines. Therefore, only these port lines of Port 1 can be controlled for the emergency stop function. The P1_ESR bits EN[15:12] are not implemented. They are always read as 0 and should be written with 0.

9.4.3.6 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 (see also **Figure 9-5**).

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.[3:0]** (Class A1 pads; coding see **Page 9-11**)
PD1	[6:4]	rw	**Pad Driver Mode for P1.[7:4]** (Class A1 pads; coding see **Page 9-11**)
PDEMUX	[10:8]	rw	**Pad Driver Mode for P1.[14:12]** (Class A1 pads; coding see **Page 9-11**)
PDSSC1B	[18:16]	rw	**Pad Driver Mode for P1.[11:8]** (Class A2 pads; coding see **Page 9-11**)
0	3, 7, [15:11], [31:19]	r	**Reserved** Read as 0; should be written with 0.

9.5 Port 2

This section describes the Port 2 functionality in detail.

9.5.1 Port 2 Configuration

Port 2 is a 14-bit bi-directional general-purpose I/O port which can be alternatively used for GPTA I/O, and interface for MLI0, MSC0 or SSC0/1.

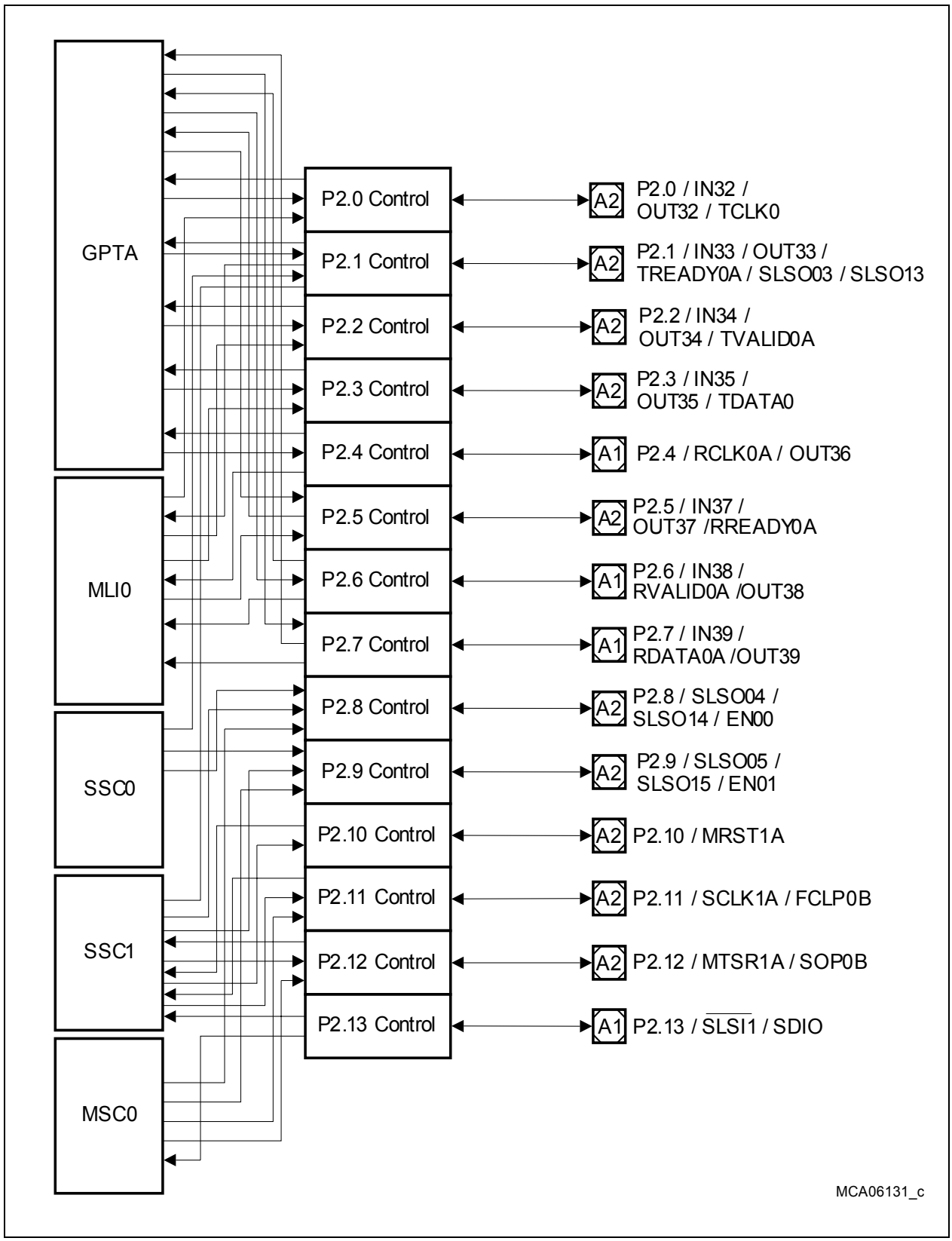


Figure 9-6 Port 2 Configuration Diagram

9.5.2 Port 2 Function Table

Table 9-11 summarizes the I/O control selection functions of each Port 2 line.

Table 9-11 Port 2 Functions

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P2.0	I	General-purpose input	P2_IN.P0	P2_IOCR0.PC0	$0XXX_B$
		GPTA input	IN32		
	O	General-purpose output	P2_OUT.P0		$1X00_B$
		GPTA output	OUT32		$1X01_B$
		MLI0 output	TCLK0		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P2.1	I	General-purpose input	P2_IN.P1	P2_IOCR0.PC1	$0XXX_B$
		GPTA input	IN33		
		MLI0 input	TREADY0A		
	O	General-purpose output	P2_OUT.P1		$1X00_B$
		GPTA output	OUT33		$1X01_B$
		SSC0 output	SLSO03		$1X10_B$
		SSC1 output	SLSO13		$1X11_B$
P2.2	I	General-purpose input	P2_IN.P2	P2_IOCR0.PC2	$0XXX_B$
		GPTA input	IN34		
	O	General-purpose output	P2_OUT.P2		$1X00_B$
		GPTA output	OUT34		$1X01_B$
		MLI0 output	TVALID0A		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P2.3	I	General-purpose input	P2_IN.P3	P2_IOCR0.PC3	$0XXX_B$
		GPTA input	IN35		
	O	General-purpose output	P2_OUT.P3		$1X00_B$
		GPTA output	OUT35		$1X01_B$
		MLI0 output	TDATA0		$1X10_B$
		Reserved[1)]	–		$1X11_B$

Table 9-11 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
P2.4	I	General-purpose input	P2_IN.P4	P2_IOCR4.PC4	$0XXX_B$
		GPTA input	IN36		
		MLI0 input	RCLK0A		
	O	General-purpose output	P2_OUT.P4		$1X00_B$
		GPTA output[1)]	OUT36		$1X01_B$
		GPTA output	OUT36		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P2.5	I	General-purpose input	P2_IN.P5	P2_IOCR4.PC5	$0XXX_B$
		GPTA input	IN37		
	O	General-purpose output	P2_OUT.P5		$1X00_B$
		GPTA output	OUT37		$1X01_B$
		MLI0 output	RREADY0A		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P2.6	I	General-purpose input	P2_IN.P6	P2_IOCR4.PC6	$0XXX_B$
		GPTA input	IN38		
		MLI0 input	RVALID0A		
	O	General-purpose output	P2_OUT.P6		$1X00_B$
		GPTA output[1)]	OUT38		$1X01_B$
		GPTA output[1)]	OUT38		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P2.7	I	General-purpose input	P2_IN.P7	P2_IOCR4.PC7	$0XXX_B$
		GPTA input	IN39		
		MLI0 input	RDATA0A		
	O	General-purpose output	P2_OUT.P7		$1X00_B$
		GPTA output[1)]	OUT39		$1X01_B$
		GPTA output[1)]	OUT39		$1X10_B$
		Reserved[1)]	–		$1X11_B$

Table 9-11 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
P2.8	I	General-purpose input	P2_IN.P8	P2_IOCR8.PC8	$0XXX_B$
	O	General-purpose output	P2_OUT.P8		$1X00_B$
		SSC0 output	SLSO04		$1X01_B$
		SSC1 output	SLSO14		$1X10_B$
		MSC0 output	EN00		$1X11_B$
P2.9	I	General-purpose input	P2_IN.P9	P2_IOCR8.PC9	$0XXX_B$
	O	General-purpose output	P2_OUT.P9		$1X00_B$
		SSC0 output	SLSO05		$1X01_B$
		SSC1 output	SLSO15		$1X10_B$
		MSC0 output	EN01		$1X11_B$
P2.10	I	General-purpose input	P2_IN.P10	P2_IOCR8.PC10	$0XXX_B$
		SSC1 input (Master Mode)[1)]	MRST1A		
	O	General-purpose output	P2_OUT.P10		$1X00_B$
		SSC1 output (Slave Mode)	MRST1A		$1X01_B$
		SSC1 output (Slave Mode)	MRST1A		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P2.11	I	General-purpose input	P2_IN.P11	P2_IOCR8.PC11	$0XXX_B$
		SSC1 input (Slave Mode)	SCLK1A		
	O	General-purpose output	P2_OUT.P11		$1X00_B$
		SSC1 output (Master Mode)[1)]	SCLK1A		$1X01_B$
		SSC1 output (Master Mode)	SCLK1A		$1X10_B$
		MSC0 output	FCLP0B		$1X11_B$

Table 9-11 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
P2.12	I	General-purpose input	P2_IN.P12	P2_IOCR12.PC12	$0XXX_B$
		SSC1 input (Slave Mode)	MTSR1A		
	O	General-purpose output	P2_OUT.P12		$1X00_B$
		SSC1 output (Master Mode)[2]	MTSR1A		$1X01_B$
		SSC1 output (Master Mode)	MTSR1A		$1X10_B$
		MSC0 output	SOP0B		$1X11_B$
P2.13	I	General-purpose input	P2_IN.P13	P2_IOCR12.PC13	$0XXX_B$
		SSC1 input	$\overline{\text{SLSI1}}$		
		MSC0 input	SDIO		
	O	General-purpose output	P2_OUT.P13		$1X00_B$
		Reserved[1]	–		$1X01_B$
		Reserved[1]	–		$1X10_B$
		Reserved[1]	–		$1X11_B$

1) The port I/O control values P2_IOCRx.Py that are assigned to this reserved alternate output control selection should not be used. Otherwise, unpredictable output port line behavior may occur

2) The ALT1 and ALT2 for this pin are connected together. There are no dependencies. Either one can be chosen.

9.5.3 Port 2 Registers

The following registers are available on Port 2:

Table 9-12 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 9-7
P2_IOCR4	Port 2 Input/Output Control Register 4	0014_H	Page 9-8
P2_IOCR8	Port 2 Input/Output Control Register 8	0018_H	Page 9-8
P2_IOCR12	Port 2 Input/Output Control Register 12	$001C_H$	Page 9-44[1]
P2_IN	Port 2 Input Register	0024_H	Page 9-44[1]
P2_PDR	Port 2 Pad Driver Mode Register	0040_H	Page 9-44[1]
P2_ESR	Port 2 Emergency Stop Register	0050_H	Page 9-44[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 9.2.

Note: The complete address map of Port 2 is described in Table 16-11 on Page 16-22 of this TC1766 System Units (Vol. 1 of 2) User's Manual.

9.5.3.1 Port 2 Output Register

The basic P2_OUT register functionality is described on Page 9-14. Port lines P2.[15:14] are not connected to port lines. Therefore, reading the P2_OUT bits P[15:14] returns the value that was last written (0 after reset). These bits can also be set/cleared by the corresponding P2_OMR bits.

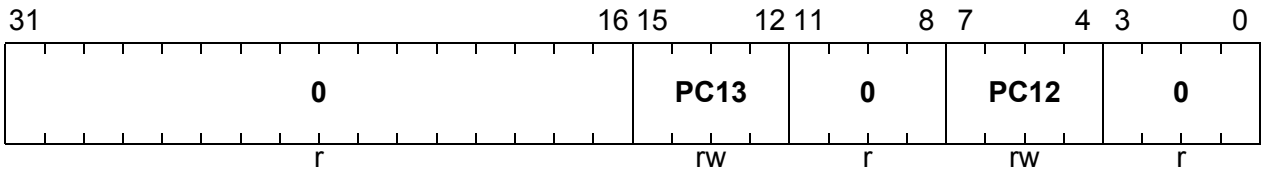
9.5.3.2 Port 2 Output Modification Register

The basic P2_OMR register functionality is described on Page 9-15. However, port lines P2.15 and P2.14 are not available. Therefore, the P2_OMR bits PS[15:14] and PR[15:14] have no direct effect on port lines but only on register bits P2_OUT.P[15:14].

9.5.3.3 Port 2 Input/Output Control Register 12

Port lines P2.14 and P2.15 are not available. Therefore, the PC14 and PC15 bit fields in register P2_IOCR12 are not connected.

P2_IOCR12
Port 2 Input/Output Control Register 12
($1C_H$) **Reset Value: $0000\ 2020_H$**



Field	Bits	Type	Description
PC12	[7:4]	rw	**Port Control for Port 2.12** (coding see **Table 9-3** on **Page 9-10**)
PC13	[15:12]	rw	**Port Control for Port 2.13** (coding see **Table 9-3** on **Page 9-10**)
0	[3:0], [11:8], [31:16]	r	**Reserved** Read as 0; should be written with 0.

9.5.3.4 Port 2 Input Register

The basic P2_IN register functionality is described on **Page 9-18**. However, port lines P2.14 and P2.15 are not available. Therefore, bits P14 and P15 in register P2_IN are always read as 0.

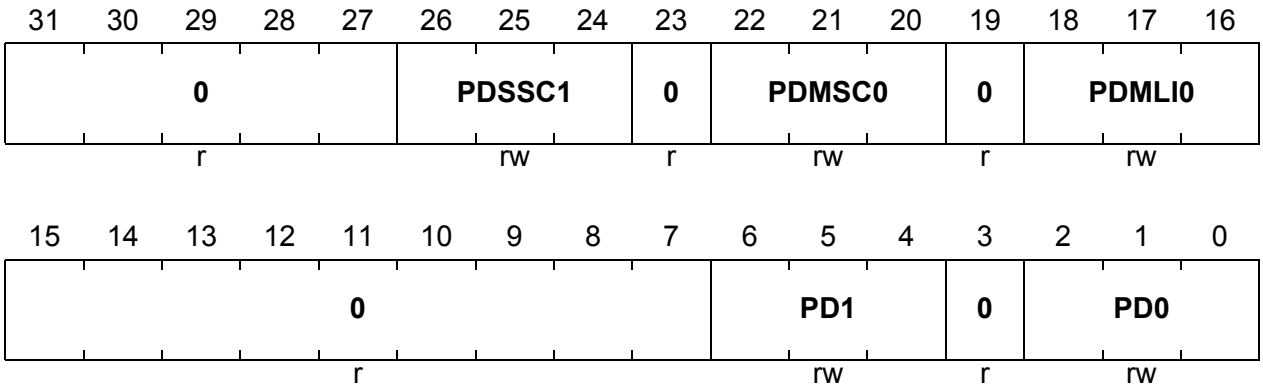
9.5.3.5 Port 2 Emergency Stop Register

The basic P2_ESR register functionality is described on **Page 9-17**. At Port 2, only port lines P2.[7:0] 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[15:8] are not implemented. They are always read as 0 and should be written with 0.

9.5.3.6 Port 2 Pad Driver Mode Register and Pad Classes

The Port 2 pad driver mode register contains five 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 (see also **Figure 9-6**).

P2_PDR
Port 2 Pad Driver Mode Register (40_H) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
PD0	[2:0]	rw	**Pad Driver Mode for P2.4, P2.[7:6]** (Class A1 pads; coding see **Page 9-11**)
PD1	[6:4]	rw	**Pad Driver Mode for P2.13** (Class A1 pads; coding see **Page 9-11**)
PDMLI0	[18:16]	rw	**Pad Driver Mode for P2.0, P2.[3:2], P2.5** (Class A2 pads; coding see **Page 9-11**)
PDMSC0	[22:20]	rw	**Pad Driver Mode for P2.1, P2.[9:8]** (Class A2 pads; coding see **Page 9-11**)
PDSSC1	[26:24]	rw	**Pad Driver Mode for P2.[12:10]** (Class A2 pads; coding see **Page 9-11**)
0	3, [15:7], 19, 23, [31:27]	r	**Reserved** Read as 0; should be written with 0.

9.6 Port 3

This section describes the Port 3 functionality in detail.

9.6.1 Port 3 Configuration

Port 3 is a 16-bit bi-directional general-purpose I/O port which can be alternatively used for ASC0/1, SSC0/1 and CAN lines.

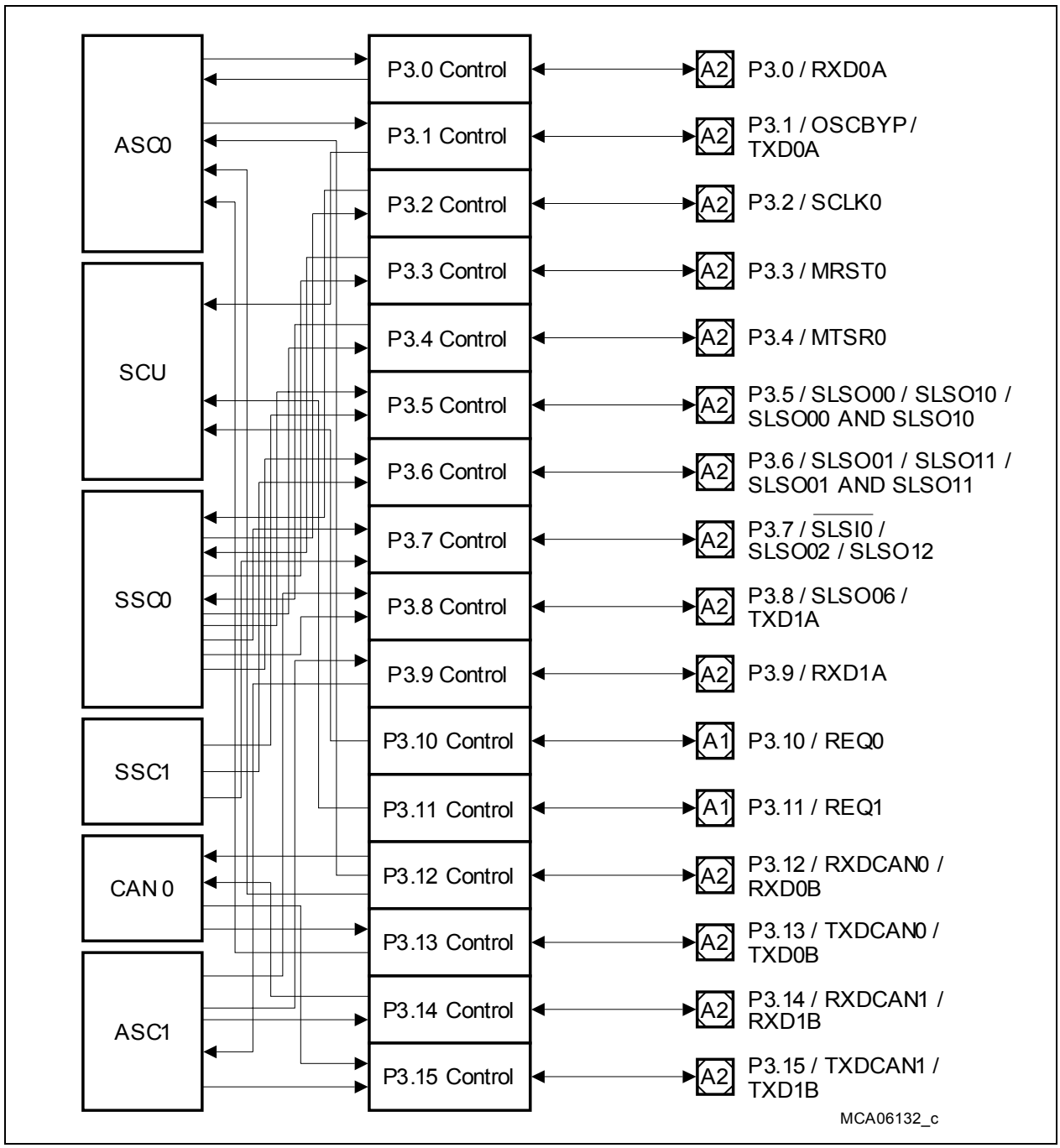


Figure 9-7 Port 3 Configuration Diagram

9.6.2 Port 3 Function Table

Table 9-13 summarizes the I/O control selection functions of each Port 3 line.

Table 9-13 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	$0XXX_B$
		ASC0 input	RXD0A (Async./Sync. Mode)		
	O	General-purpose output	P3_OUT.P0		$1X00_B$
		ASC0 output (Synchronous Mode)[1)]	RXD0A		$1X01_B$
		ASC0 output (Synchronous Mode)	RXD0A		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P3.1	I	General-purpose input	P3_IN.P1	P3_IOCR0.PC1	$0XXX_B$
		SCU input	OSCBYP		
	O	General-purpose output	P3_OUT.P1		$1X00_B$
		ASC0 output[1)]	TXD0A		$1X01_B$
		ASC0 output	TXD0A		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P3.2	I	General-purpose input	P3_IN.P2	P3_IOCR0.PC2	$0XXX_B$
		SSC0 input (Slave Mode)	SCLK0		
	O	General-purpose output	P3_OUT.P2		$1X00_B$
		SSC0 output (Master Mode)[1)]	SCLK0		$1X01_B$
		SSC0 output (Master Mode)	SCLK0		$1X10_B$
		Reserved[1)]	–		$1X11_B$

Table 9-13 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	$0XXX_B$
		SSC0 input (Master Mode)	MRST0		
	O	General-purpose output	P3_OUT.P3		$1X00_B$
		SSC0 output (Slave Mode)[1]	MRST0		$1X01_B$
		SSC0 output (Slave Mode)	MRST0		$1X10_B$
		Reserved[1]	–		$1X11_B$
P3.4	I	General-purpose input	P3_IN.P4	P3_IOCR4.PC4	$0XXX_B$
		SSC0 input (Slave Mode)	MTSR0		
	O	General-purpose output	P3_OUT.P4		$1X00_B$
		SSC0 output (Master Mode)[1]	MTSR0		$1X01_B$
		SSC0 output (Master Mode)	MTSR0		$1X10_B$
		Reserved[1]	–		$1X11_B$
P3.5	I	General-purpose input	P3_IN.P5	P3_IOCR4.PC5	$0XXX_B$
	O	General-purpose output	P3_OUT.P5		$1X00_B$
		SSC0 output	SLSO00		$1X01_B$
		SSC1 output	SLSO10		$1X10_B$
		SSC0 and SSC1 output	SLSO00 AND SLSO10[2]		$1X11_B$
P3.6	I	General-purpose input	P3_IN.P6	P3_IOCR4.PC11 x	$0XXX_B$
	O	General-purpose output	P3_OUT.P6		$1X00_B$
		SSC0 output	SLSO01		$1X01_B$
		SSC1 output	SLSO11		$1X10_B$
		SSC0 and SSC1 output	SLSO01 AND SLSO11[2]		$1X11_B$

Table 9-13 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	$0XXX_B$
		SSC0 input	$\overline{SLSI0}$		
	O	General-purpose output	P3_OUT.P7		$1X00_B$
		SSC0 output	SLSO02		$1X01_B$
		SSC1 output	SLSO12		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P3.8	I	General-purpose input	P3_IN.P8	P3_IOCR8.PC8	$0XXX_B$
	O	General-purpose output	P3_OUT.P8		$1X00_B$
		SSC0 output	SLSO06		$1X01_B$
		ASC1 output	TXD1A		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P3.9	I	General-purpose input	P3_IN.P9	P3_IOCR8.PC9	$0XXX_B$
		ASC1 input (Asynchronous Mode/Synchronous Mode)	RXD1A		
	O	General-purpose output	P3_OUT.P9		$1X00_B$
		ASC1 output (Synchronous Mode)[1)]	RXD1A		$1X01_B$
		ASC1 output (Synchronous Mode)	RXD1A		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P3.10	I	General-purpose input	P3_IN.P10	P3_IOCR8.PC10	$0XXX_B$
		SCU input	REQ0		
	O	General-purpose output	P3_OUT.P10		$1X00_B$
		Reserved[1)]	–		$1X01_B$
		Reserved[1)]	–		$1X10_B$
		Reserved[1)]	–		$1X11_B$

Table 9-13 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	$0XXX_B$
		SCU input	REQ1		
	O	General-purpose output	P3_OUT.P11		$1X00_B$
		Reserved[1)]	–		$1X01_B$
		Reserved[1)]	–		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P3.12	I	General-purpose input	P3_IN.P12	P3_IOCR12.PC12	$0XXX_B$
		CAN node 0 receive input 0 CAN node 1 receive input 1	RXDCAN0		
		ASC0 input (Asynchronous Mode /Synchronous Mode)	RXD0B		
	O	General-purpose output	P3_OUT.P12		$1X00_B$
		ASC0 output (Synchronous Mode)[1)]	RXD0B		$1X01_B$
		ASC0 output (Synchronous Mode)	RXD0B		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P3.13	I	General-purpose input	P3_IN.P13	P3_IOCR12.PC13	$0XXX_B$
	O	General-purpose output	P3_OUT.P13		$1X00_B$
		CAN node 0 output	TXDCAN0		$1X01_B$
		ASC0 output (Synchronous Mode)	TCD0B		$1X10_B$
		Reserved[1)]	–		$1X11_B$

Table 9-13 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
P3.14	I	General-purpose input	P3_IN.P14	P3_IOCR12.PC14	$0XXX_B$
		CAN node 1 receive input 0 CAN node 0 receive input 1	RXDCAN1		
		ASC1 output (Asynchronous Mode/Synchronous Mode)	RXD1B		
	O	General-purpose output	P3_OUT.P14		$1X00_B$
		ASC1 output (Synchronous Mode)[3]	RXD1B		$RXD0_B$
		ASC1 output (Synchronous Mode)	RXD1B		$RXD0_B$
		Reserved[1]	–		–
P3.15	I	General-purpose input	P3_IN.P15	P3_IOCR12.PC15	$0XXX_B$
	O	General-purpose output	P3_OUT.P15		$1X00_B$
		CAN node 1output	RXDCAN1		$1X01_B$
		ASC1 output (Synchronous Mode)	TXD1B		$1X10_B$
		Reserved[1]	–		$1X11_B$

1) The port I/O control values P3_IOCRx.Py that are assigned to this reserved alternate output control selection should not be used. Otherwise, unpredictable output port line behavior may occur.

2) The AND-gate of ALT3 is located in the GPIO module.

3) The ALT1 and ALT2 for this pin are connected together. There are no dependencies. Either one can be chosen.

9.6.3 Port 3 Registers

The following registers are available on Port 3:

Table 9-14 Port 3 Registers

Register Short Name	Register Long Name	Offset Address	Description see
P3_OUT	Port 3 Output Register	0000_H	Page 9-14
P3_OMR	Port 3 Output Modification Register	0004_H	Page 9-15
P3_IOCR0	Port 3 Input/Output Control Register 0	0010_H	Page 9-7
P3_IOCR4	Port 3 Input/Output Control Register 4	0014_H	Page 9-8
P3_IOCR8	Port 3 Input/Output Control Register 8	0018_H	Page 9-8
P3_IOCR12	Port 3 Input/Output Control Register 12	$001C_H$	Page 9-9
P3_IN	Port 3 Input Register	0024_H	Page 9-18
P3_PDR	Port 3 Pad Driver Mode Register	0040_H	Page 9-53[1)]

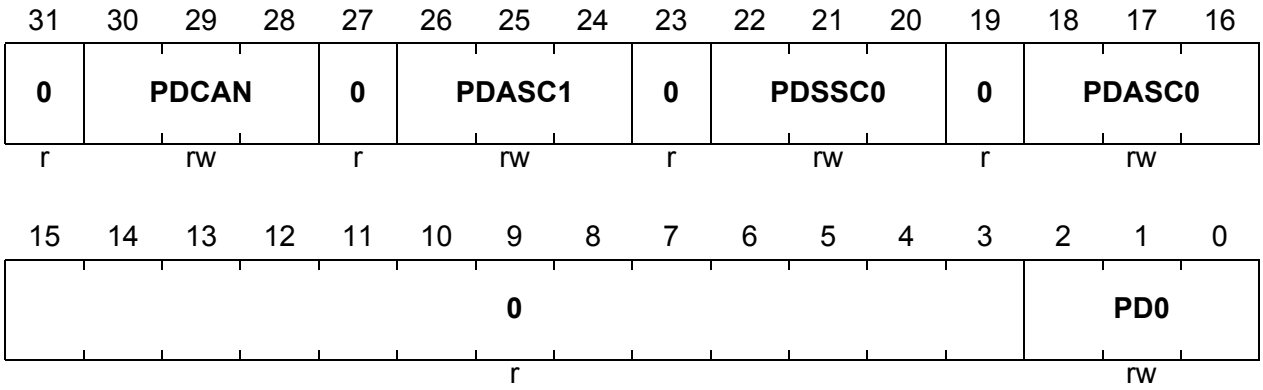
1) This register is listed here in the Port 3 section because they differ from the general port register description given in Section 9.2.

Note: The complete address map of Port 3 is described in Table 16-12 on Page 16-23 of this TC1766 System Units (Vol. 1 of 2) User's Manual.

9.6.4 Port 3 Pad Driver Mode Register and Pad Classes

The Port 3 pad driver mode register contains five 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 pads of Class A1 (see also **Figure 9-7**).

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.[11:10]** (Class A1 pads; coding see **Page 9-11**)
PDASC0	[18:16]	rw	**Pad Driver Mode for P3.[1:0]** (Class A2 pads; coding see **Page 9-11**)
PDSSC0	[22:20]	rw	**Pad Driver Mode for P3.[7:2]** (Class A2 pads; coding see **Page 9-11**)
PDASC1	[26:24]	rw	**Pad Driver Mode for P3.[9:8]** (Class A2 pads; coding see **Page 9-11**)
PDCAN	[30:28]	rw	**Pad Driver Mode for P3.[15:12]** (Class A2 pads; coding see **Page 9-11**)
0	[15:3], 19, 23, 27, 31	r	**Reserved** Read as 0; should be written with 0.

9.7 Port 4

This section describes the Port 4 functionality in detail.

9.7.1 Port 4 Configuration

Port 4 is a 4-bit bi-directional general-purpose I/O port. Its pins are used for hardware configuration as HWCFG[3:0] lines. The logic levels of the four Port 4 lines are latched when the hardware reset goes inactive (at the rising edge of $\overline{\text{HDRST}}$), and can be checked by software by reading bit field HWCFG in the reset status register RST_SR. Inside the TC1766, the HWCFG[3:0] lines are used for boot configuration selection. During normal operation, the pins can be used alternatively for GPTA interface or the system clock output.

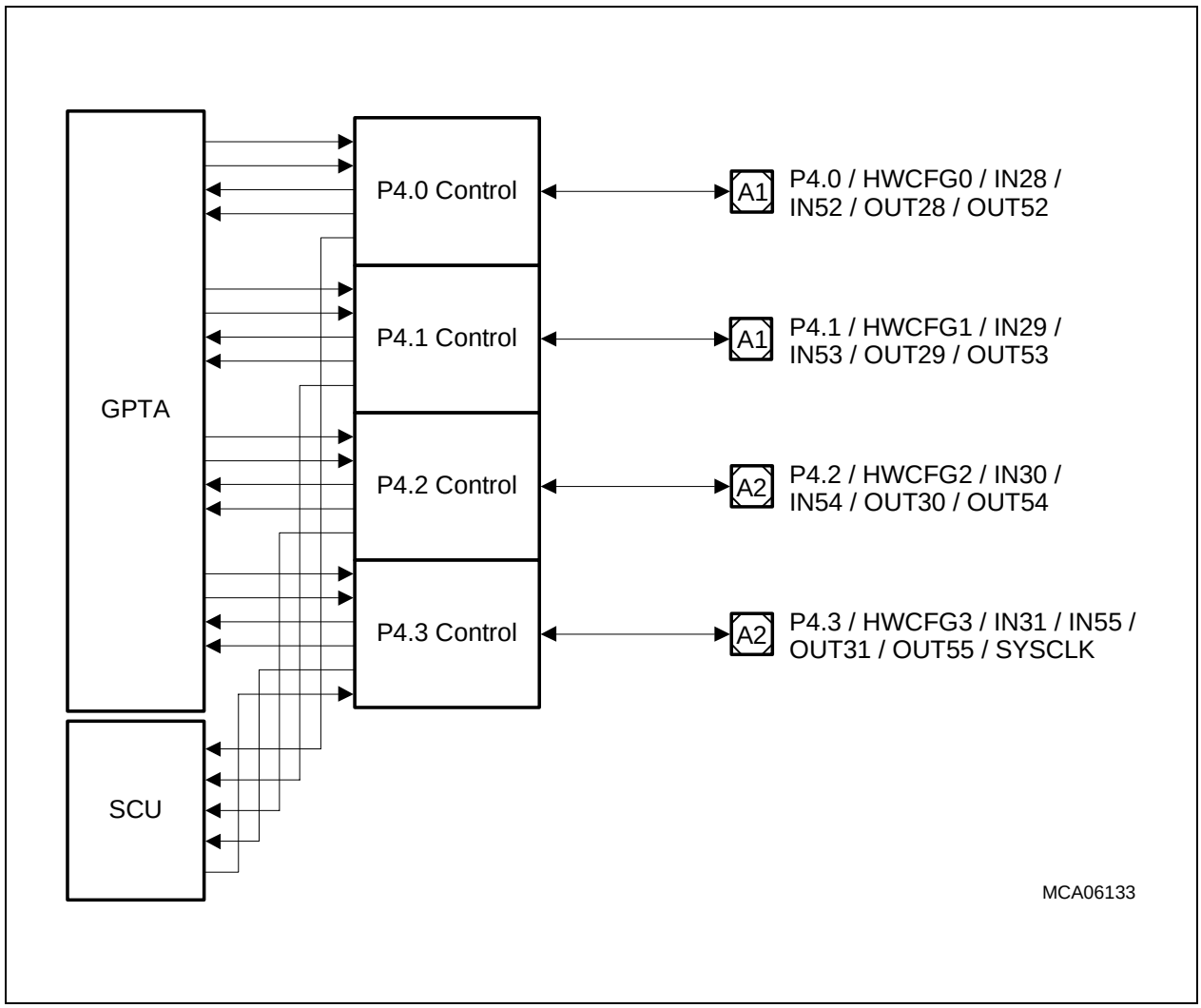


Figure 9-8 Port 4 Configuration Diagram

9.7.2 Port 4 Function Table

Table 9-15 summarizes the I/O control selection functions of each Port 4 line.

Table 9-15 Port 4 Functions

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.[1]	
				Reg./Bit Field	Value
P4.0	I	General-purpose input	P4_IN.P0	P4_IOCR0.PC0	$0XXX_B$
		SCU input	HWCFG0		
		GPTA input	IN28		
		GPTA input	IN52		
	O	General-purpose output	P4_OUT.P0		$1X00_B$
		GPTA output	OUT28		$1X01_B$
		GPTA output	OUT52		$1X10_B$
		Reserved[1]	–		$1X11_B$
P4.1	I	General-purpose input	P4_IN.P1	P4_IOCR0.PC1	$0XXX_B$
		SCU input	HWCFG1		
		GPTA input	IN29		
		GPTA input	IN53		
	O	General-purpose output	P4_OUT.P1		$1X00_B$
		GPTA output	OUT29		$1X01_B$
		GPTA output	OUT53		$1X10_B$
		Reserved[1]	–		$1X11_B$
P4.2	I	General-purpose input	P4_IN.P2	P4_IOCR0.PC2	$0XXX_B$
		SCU input	HWCFG2		
		GPTA input	IN30		
		GPTA input	IN54		
	O	General-purpose output	P4_OUT.P2		$1X00_B$
		GPTA output	OUT30		$1X01_B$
		GPTA output	OUT54		$1X10_B$
		Reserved[1]	–		$1X11_B$

Table 9-15 Port 4 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.[1]	
				Reg./Bit Field	Value
P4.3	I	General-purpose input	P4_IN.P3	P4_IOCR0.PC3	$0XXX_B$
		SCU input	HWCFG3		
		GPTA input	IN31		
		GPTA input	IN55		
	O	General-purpose output	P4_OUT.P3		$1X00_B$
		GPTA output	OUT31		$1X01_B$
		GPTA output	OUT55		$1X10_B$
		SCU output	SYSCLK		$1X11_B$

1) The port I/O control values P4_IOCRx.Py that are assigned to this reserved alternate output control selection should not be used. Otherwise, unpredictable output port line behavior may occur.

9.7.3 Port 4 Registers

The following registers are available on Port 4:

Table 9-16 Port 4 Registers

Register Short Name	Register Long Name	Offset Address	Description see
P4_OUT	Port 4 Output Register	0000_H	Page 9-57[1]
P4_OMR	Port 4 Output Modification Register	0004_H	Page 9-57[1]
P4_IOCR0	Port 4 Input/Output Control Register 0	0010_H	Page 9-7
P4_IN	Port 4 Input Register	0024_H	Page 9-57[1]
P4_PDR	Port 4 Pad Driver Mode Register	0040_H	Page 9-58[1]
P4_ESR	Port 4 Emergency Stop Register	0050_H	Page 9-57[1]
RST_SR	Reset Status Register	–	Page 4-2

1) This register is listed here in the Port 4 section because they differ from the general port register description given in Section 9.2.

Note: The complete address map of Port 4 is described in Table 16-13 on Page 16-24 of this TC1766 System Units (Vol. 1 of 2) User's Manual.

Note: Bit field HWCFG in register RST_SR contains the latched logic levels of the Port 4 inputs that have been detected at the last low-to-high transition of HDRST.

9.7.3.1 Port 4 Output Register

The basic P4_OUT register functionality is described on **Page 9-14**. Port lines P4.[15:4] are not connected to port lines. Therefore, reading the P1_OUT bits P4.[15:4] returns the value that was last written (0 after reset). These bits can also be set/cleared by the corresponding P4_OMR bits.

9.7.3.2 Port 4 Output Modification Register

The basic P4_OMR register functionality is described on **Page 9-15**. However, port line P4.[15:4] are not available. Therefore, the P4_OMR bits PS.[15:4] and PR.[15:4] have no direct effect on port lines but only on register bits P4_OUT.P[15:4].

9.7.3.3 Port 4 Input/Output Control Register x (x = 4, 8 and 12)

Port lines P4.[15:4] are not available. Therefore, the PC bit fields; PC[15:4] in registers P4_IOCR4, P4_IOCR8 and P4_IOCR12 are not connected.

9.7.3.4 Port 4 Input Register

The basic P4_IN register functionality is described on **Page 9-18**. However, port line P4.[15:4] are not available. Therefore, bits P[15:4] in register P4_IN are always read as 0.

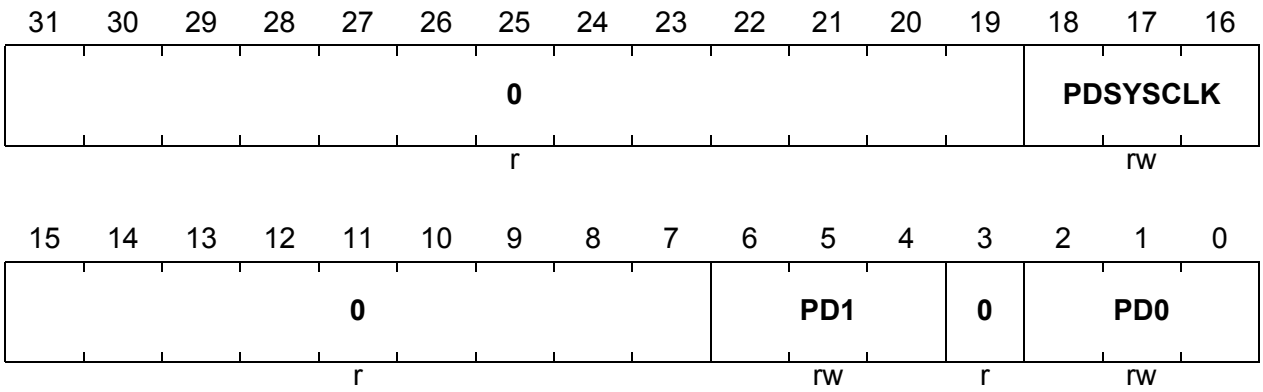
9.7.3.5 Port 4 Emergency Stop Register

The basic P4_ESR register functionality is described on **Page 9-17**. At Port 4, port lines P4.[3:0] are connected to GPTA I/O lines. Therefore, these port lines of Port 4 can be controlled for the emergency stop function. The P4_ESR bits EN[15:4] are not implemented. They are always read as 0 and should be written with 0.

9.7.3.6 Port 4 Pad Driver Mode Register and Pad Classes

The Port 4 pad driver mode register contains three 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 Class A1 and A2 pad classes (see also **Figure 9-8**).

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.[1:0]** (Class A1 pads; coding see **Page 9-11**)
PD1	[6:4]	rw	**Pad Driver Mode for P4.2** (Class A2 pads; coding see **Page 9-11**)
PDSYSCLK	[18:16]	rw	**Pad Driver Mode for P4.3** (Class A2 pads; coding see **Page 9-11**)
0	3, [15:7], [31:19],	r	**Reserved** Read as 0; should be written with 0.

9.8 Port 5

This section describes the Port 5 functionality in detail.

9.8.1 Port 5 Configuration

Port 5 is a 16-bit bi-directional general-purpose I/O port. In emulation, it is used as a trace port for OCDS Level 2 debug lines. In normal operation, it is used for the GPTA I/O or the MLI0/MLI1 interface.

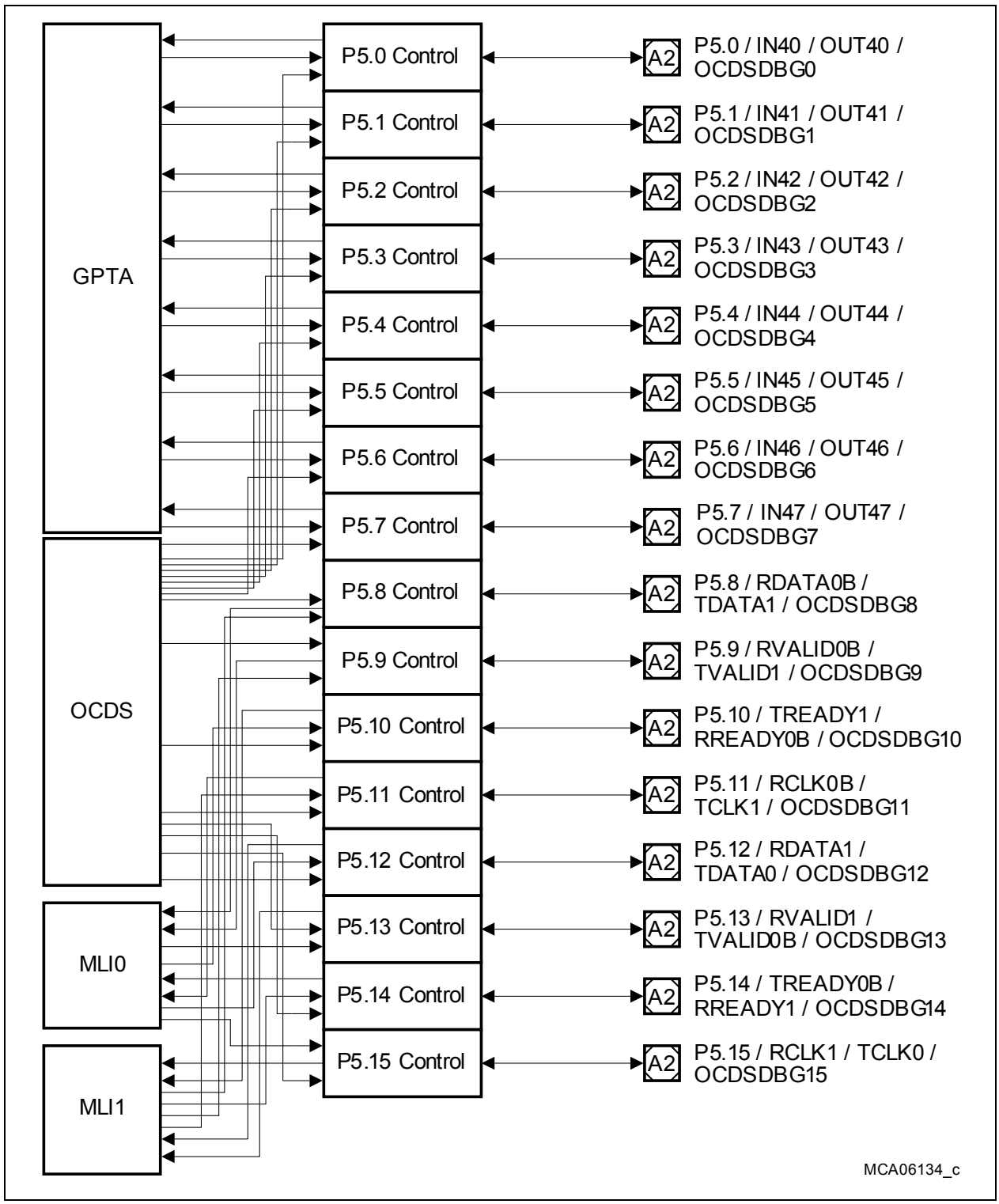


Figure 9-9 Port 5 Configuration Diagram

9.8.2 Port 5 Function Table

Table 9-17 summarizes the I/O control selection functions of each Port 5 line.

Table 9-17 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	$0XXX_B$
		GPTA input	IN40		
	O	General-purpose output	P5_OUT.P0		$1X00_B$
		GPTA output	OUT40		$1X01_B$
		OCDS output	OCDSDBG0 (Pipeline Status Signal PS0)		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P5.1	I	General-purpose input	P5_IN.P1	P5_IOCR0.PC1	$0XXX_B$
		GPTA input	IN41		
	O	General-purpose output	P5_OUT.P1		$1X00_B$
		GPTA output	OUT41		$1X01_B$
		OCDS output	OCDSDBG1 (Pipeline Status Signal PS1)		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P5.2	I	General-purpose input	P5_IN.P2	P5_IOCR0.PC2	$0XXX_B$
		GPTA input	IN42		
	O	General-purpose output	P5_OUT.P2		$1X00_B$
		GPTA output	OUT42		$1X01_B$
		OCDS output	OCDSDBG2 (Pipeline Status Signal PS2)		$1X10_B$
		Reserved[1)]	–		$1X11_B$

Table 9-17 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
P5.3	I	General-purpose input	P5_IN.P3	P5_IOCR0.PC3	$0XXX_B$
		GPTA input	IN43		
	O	General-purpose output	P5_OUT.P3		$1X00_B$
		GPTA output	OUT43		$1X01_B$
		OCDS output	OCDSDBG3 (Pipeline Status Signal PS3)		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P5.4	I	General-purpose input	P5_IN.P4	P5_IOCR4.PC4	$0XXX_B$
		GPTA input	IN44		
	O	General-purpose output	P5_OUT.P4		$1X00_B$
		GPTA output	OUT44		$1X01_B$
		OCDS output	OCDSDBG4 (Pipeline Status Signal PS4)		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P5.5	I	General-purpose input	P5_IN.P5	P5_IOCR4.PC5	$0XXX_B$
		GPTA input	IN45		
	O	General-purpose output	P5_OUT.P5		$1X00_B$
		GPTA output	OUT45		$1X01_B$
		OCDS output	OCDSDBG5 (Break Qualification Line BRK0)		$1X10_B$
		Reserved[1)]	–		$1X11_B$

Table 9-17 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
P5.6	I	General-purpose input	P5_IN.P6	P5_IOCR4.PC6	$0XXX_B$
		GPTA input	IN46		
	O	General-purpose output	P5_OUT.P6		$1X00_B$
		GPTA output	OUT46		$1X01_B$
		OCDS output	OCDSDBG6 (Break Qualification Line BRK1)		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P5.7	I	General-purpose input	P5_IN.P7	P5_IOCR4.PC7	$0XXX_B$
		GPTA input	IN47		
	O	General-purpose output	P5_OUT.P7		$1X00_B$
		GPTA output	OUT47		$1X01_B$
		OCDS output	OCDSDBG7 (Break Qualification Line BRK2)		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P5.8	I	General-purpose input	P5_IN.P8	P5_IOCR8.PC8	$0XXX_B$
		MLI0 input	RDATA0B		
	O	General-purpose output	P5_OUT.P8		$1X00_B$
		MLI1 output	TDATA1		$1X01_B$
		OCDS output	OCDSDBG8 (Indirect PC Address PC0)		$1X10_B$
		Reserved[1)]	–		$1X11_B$

Table 9-17 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	$0XXX_B$
		MLI0 input	RVALID0B		
	O	General-purpose output	P5_OUT.P9		$1X00_B$
		MLI1 output	TVALID1		$1X01_B$
		OCDS output	OCDSDBG9 (Indirect PC Address PC1)		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P5.10	I	General-purpose input	P5_IN.P10	P5_IOCR8.PC10	$0XXX_B$
		MLI1 input	TREADY1		
	O	General-purpose output	P5_OUT.P10		$1X00_B$
		MLI0 output	RREADY0B		$1X01_B$
		OCDS output	OCDSDBG10 (Indirect PC Address PC2)		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P5.11	I	General-purpose input	P5_IN.P11	P5_IOCR8.PC11	$0XXX_B$
		MLI0 input	RCLK0B		
	O	General-purpose output	P5_OUT.P11		$1X00_B$
		MLI1 output	TCLK1		$1X01_B$
		OCDS output	OCDSDBG11 (Indirect PC Address PC3)		$1X10_B$
		Reserved[1)]	–		$1X11_B$

Table 9-17 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
P5.12	I	General-purpose input	P5_IN.P12	P5_IOCR12.PC12	$0XXX_B$
		MLI1 input	RDATA1		
	O	General-purpose output	P5_OUT.P12		$1X00_B$
		MLI0 output	TDATA0		$1X01_B$
		OCDS output	OCDSDBG12 (Indirect PC Address PC4)		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P5.13	I	General-purpose input	P5_IN.P13	P5_IOCR12.PC13	$0XXX_B$
		MLI1 input	RVALID1		
	O	General-purpose output	P5_OUT.P13		$1X00_B$
		MLI0 output	TVALID0B		$1X01_B$
		OCDS output	OCDSDBG13 (Indirect PC Address PC5)		$1X10_B$
		Reserved[1)]	–		$1X11_B$
P5.14	I	General-purpose input	P5_IN.P14	P5_IOCR12.PC14	$0XXX_B$
		MLI0 input	TREADY0B		
	O	General-purpose output	P5_OUT.P14		$1X00_B$
		MLI1 output	RREADY1		$1X01_B$
		OCDS output	OCDSDBG14 (Indirect PC Address PC6)		$1X10_B$
		Reserved[1)]	–		$1X11_B$

Table 9-17 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
P5.15	I	General-purpose input	P5_IN.P15	P5_IOCR12.PC15	$0XXX_B$
		MLI1 input	RCLK1		
	O	General-purpose output	P5_OUT.P15		$1X00_B$
		MLI0 output	TCLK0		$1X01_B$
		OCDS output	OCDSDBG15 (Indirect PC Address PC7)		$1X10_B$
		Reserved[1)]	–		$1X11_B$

1) The port I/O control values P5_IOCRx.Py that are assigned to this reserved alternate output control selection should not be used. Otherwise, unpredictable output port line behavior may occur.

9.8.3 Port 5 Registers

The following registers are available on Port 5:

Table 9-18 Port 5 Registers

Register Short Name	Register Long Name	Offset Address	Description see
P5_OUT	Port 5 Output Register	0000_H	Page 9-14
P5_OMR	Port 5 Output Modification Register	0004_H	Page 9-15
P5_IOCR0	Port 5 Input/Output Control Register 0	0010_H	Page 9-7
P5_IOCR4	Port 5 Input/Output Control Register 4	0014_H	Page 9-8
P5_IOCR8	Port 5 Input/Output Control Register 8	0018_H	Page 9-8
P5_IOCR12	Port 5 Input/Output Control Register 12	$001C_H$	Page 9-9
P5_IN	Port 5 Input Register	0024_H	Page 9-18
P5_PDR	Port 5 Pad Driver Mode Register	0040_H	Page 9-67[1)]
P5_ESR	Port 5 Emergency Stop Register	0050_H	Page 9-67[1)]

1) These registers are noted here in the Port 5 section because they differ from the general port register description given in Section 9.2.

Note: The complete address map of Port 5 is described in Table 16-14 on Page 16-25 of this TC1766 System Units (Vol. 1 of 2) User's Manual.

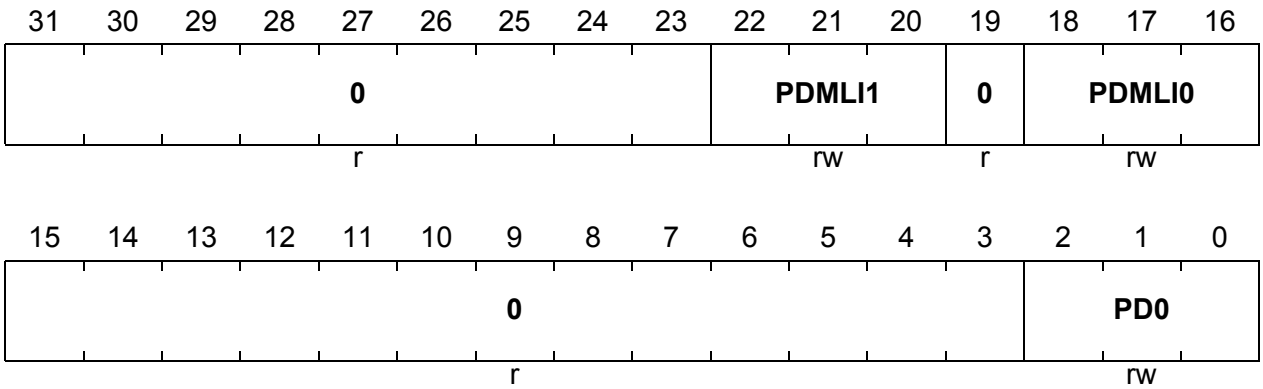
9.8.3.1 Port 5 Emergency Stop Register

The basic P5_ESR register functionality is described on Page 9-17. At Port 5, only port lines P5.[7:0] are connected to GPTA I/O lines. Therefore, only these port lines of Port 5 can be controlled for the emergency stop function. The P5_ESR bits EN[15:8] are not implemented. They are always read as 0 and should be written with 0.

9.8.3.2 Port 5 Pad Driver Mode Register and Pad Classes

The Port 5 pad driver mode register contains three bit fields that determine the pad driver mode (output driver strength and slew rate) of Port 5 line groups. The Port 5 port lines are of Class A2 pads (see also Figure 9-9).

P5_PDR
Port 5 Pad Driver Mode Register (40_H) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
PD0	[2:0]	rw	**Pad Driver Mode for P5.[7:0]** (Class A2 pads; coding see Page 9-11)
PDMLI0	[18:16]	rw	**Pad Driver Mode for P5.10, P5.[13:12], P5.15** (Class A2 pads; coding see Page 9-11)
PDMLI1	[22:20]	rw	**Pad Driver Mode for P5.[9:8], P5.11, P5.14** (Class A2 pads; coding see Page 9-11)
0	[15:3], 19, [31:23]	r	**Reserved** Read as 0; should be written with 0.

9.9 Dedicated Peripheral I/O Lines

9.9.1 LVDS Outputs of MSC0

The clock and data output lines of MSC0 are connected to dedicated differential output drivers, each of which has positive and negative signal polarity. These types of 3.3 V LVDS pads are assigned as Class C pads.

The LVDS outputs are controlled by a bit LDEN (LVDS Driver Enable) that is located in register SCU_CON. With LDEN = 0, all LVDS drivers are disabled and in its power-down mode. With LDEN = 1, all LVDS drivers are enabled for operation.

Table 9-19 LVDS Outputs of MSC0

Signal/Pin Short Name	Description
FCLP0A	MSC0 differential driver clock output positive A
FCLN0	MSC0 differential driver clock output negative
SOP0A	MSC0 differential driver serial data output positive A
SON0	MSC0 differential driver serial data output negative

10 Peripheral Control Processor (PCP)

This chapter describes the Peripheral Control Processor (PCP), its architecture, programming model, registers, and instructions. The TC1766's PCP is an enhanced version 2 (PCP2) of the TC1775's PCP peripheral control processor.

Section 10.1 to **Section 10.13** of this chapter describe the TC1766's PCP in general. TC1766 implementation-specific details are described in **Section 10.14**.

10.1 Peripheral Control Processor Overview

The PCP in the TC1766 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 TC1766 and also has reasonable stand-alone computational capabilities.

The TC1766 contains an improved version of the TC1775's PCP with the following enhancements:

- Optimized context switching
- Support for nested interrupts
- Enhanced instruction set
- Enhanced instruction execution speed
- Enhanced interrupt queueing

10.2 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)

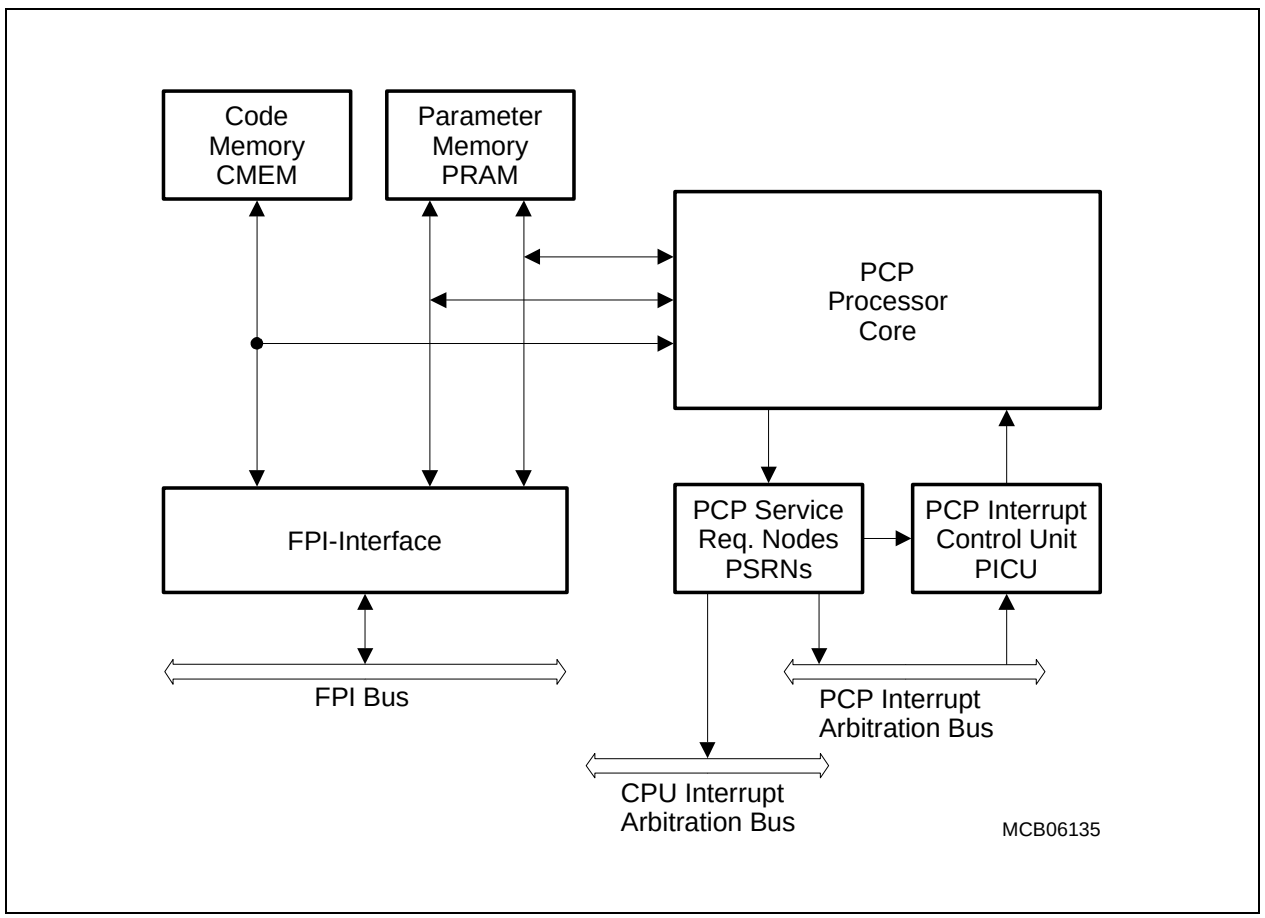


Figure 10-1 PCP Block Diagram

10.2.1 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 10-1** 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 10-6**) 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 10-1 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

10.2.2 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 10-124** for the implemented type and size of the code memory in the TC1766.

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 10-48** 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.

10.2.3 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, 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 10-124** 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 10-48** for details.

10.2.4 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.

10.2.5 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 10.5.3** for more detailed information on the operation of these nodes.

10.3 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).

10.3.1 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 10-2 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 COPY instruction
R5	DST (Destination)	Destination Pointer for 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.

10.3.1.1 Register R0

R0 is used as an implicit operand destination for some instructions. These are detailed in the individual instruction descriptions.

10.3.1.2 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.

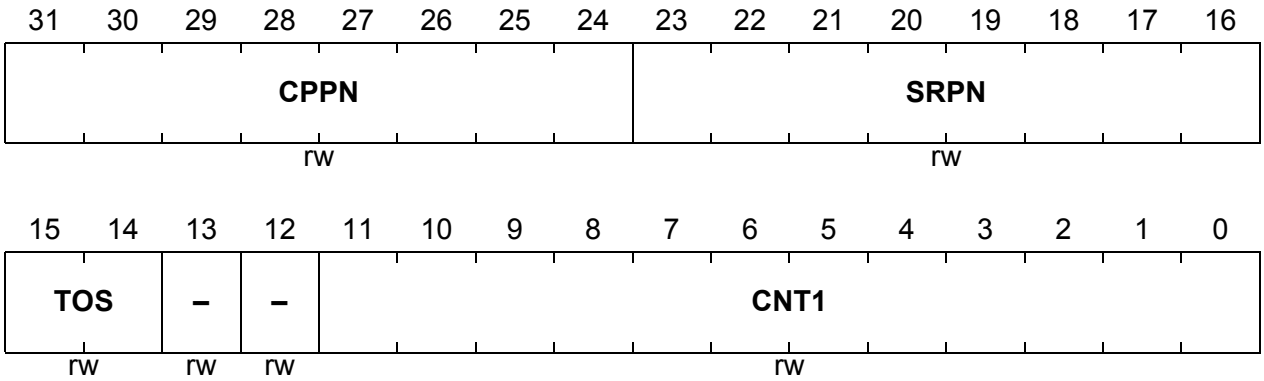
10.3.1.3 Registers R4 and R5

Registers R4 and R5 are also general-use registers. However, the COPY instruction implicitly uses R4 and R5 as full 32-bit address pointers (R4 is used as the source address and R5 as the destination address). As the COPY instruction uses these registers to maintain the address pointers, either or both R4 and R5 values may or may not be modified by the COPY instruction, depending on the options used in the instruction.

10.3.1.4 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
CNT1	[11:0]	rw	**General-use/Outer Loop count for COPY Instruction or EXIT Instruction**
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].
SRPN	[23:16]	rw	**General-use/Service Request Priority Number for EXIT Interrupt**
CPPN	[31:24]	rw	**General-use/PCP Priority Number Posted to PICU**
–	13, 12	rw	**General-use**

10.3.1.5 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 10-14** on **Page 10-107** 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.

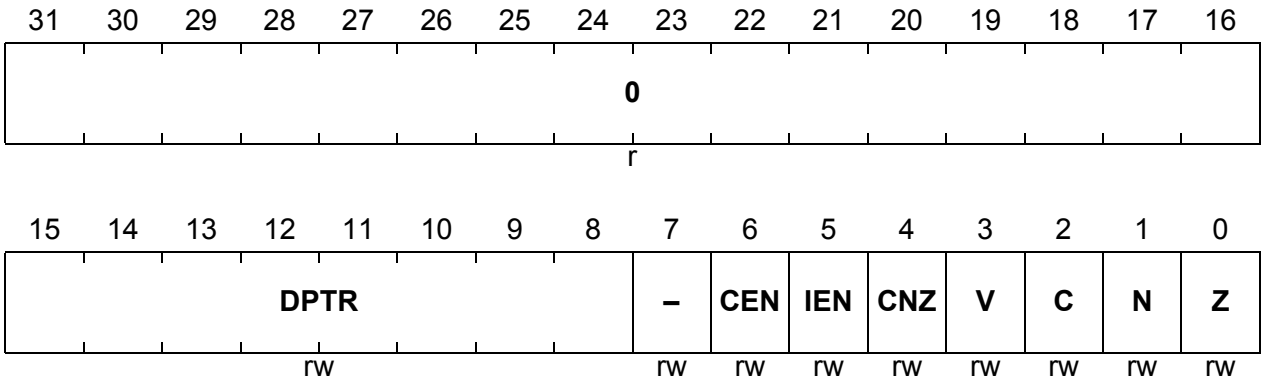
Note: Care must be taken when updating R7.DPTR to ensure that other bits within R7 (e.g. R7.CEN) are not inadvertently corrupted.

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 10.13.3.1** regarding the use of R7.IEN.*

PCP Register R7 **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
Z	0	rw	**Zero**
N	1	rw	**Negative**
C	2	rw	**Carry**
V	3	rw	**Overflow**
CNZ	4	rw	**Outer Loop Counter 1 Zero Flag**
IEN	5	rw	**Interrupt Enable** 0_B Channel is not interruptible 1_B Channel can be suspended in favor of a higher-priority service request
CEN	6	rw	**Channel Enable Control Bit**
DPTR	[15:8]	rw	**Data Pointer Segment Address for PRAM accesses**
–	7	rw	**Reserved;** should always be written with 0.
0	[31:16]	r	**Reserved;** read as 0; should be written with 0.

10.3.2 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.

10.3.2.1 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 10.3.2.3**.

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 unrestored 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 10.12.2** for more detail).

*Note: Special care must be taken when using Minimum or Small Context Model with nested interrupts (see **Page 10-118**).*

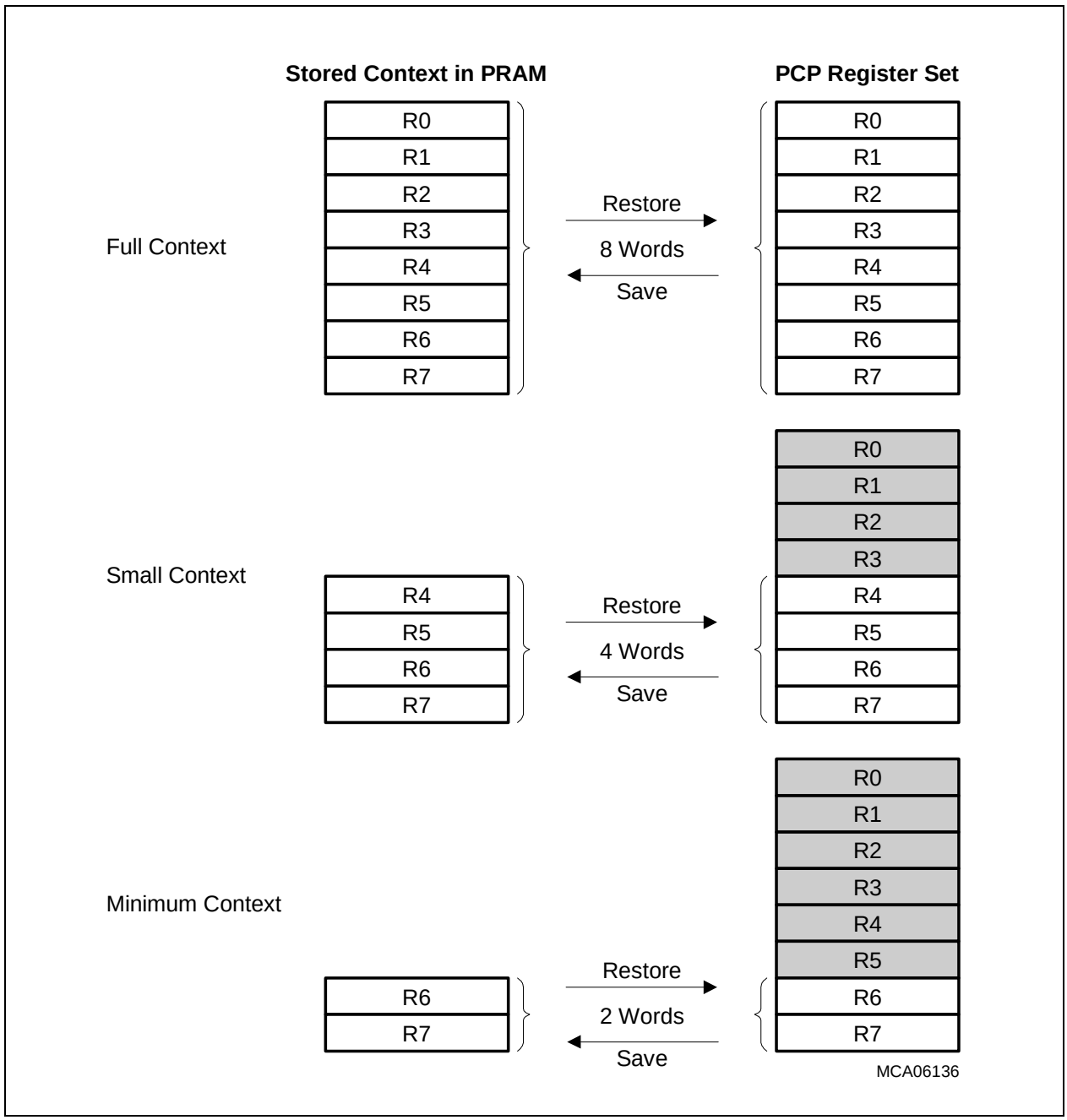


Figure 10-2 PCP Context Models

10.3.2.2 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 10-124** for the implemented size of the PRAM in the TC1766. 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 10-38**).

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.

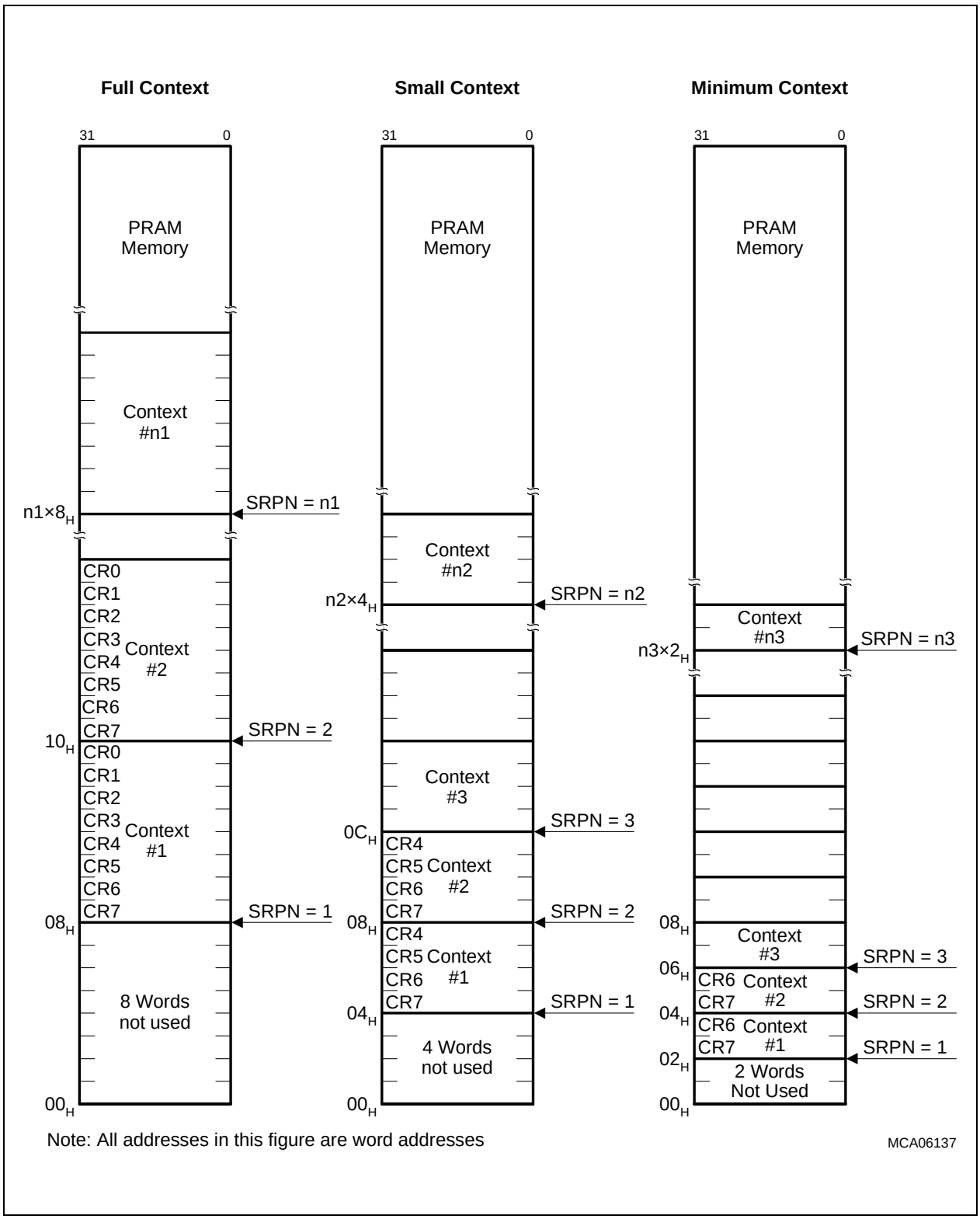


Figure 10-3 Context Storage in PRAM

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 10-118**).*

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 globals
- 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.

10.3.2.3 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 10-25**).

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 10-26**). 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.

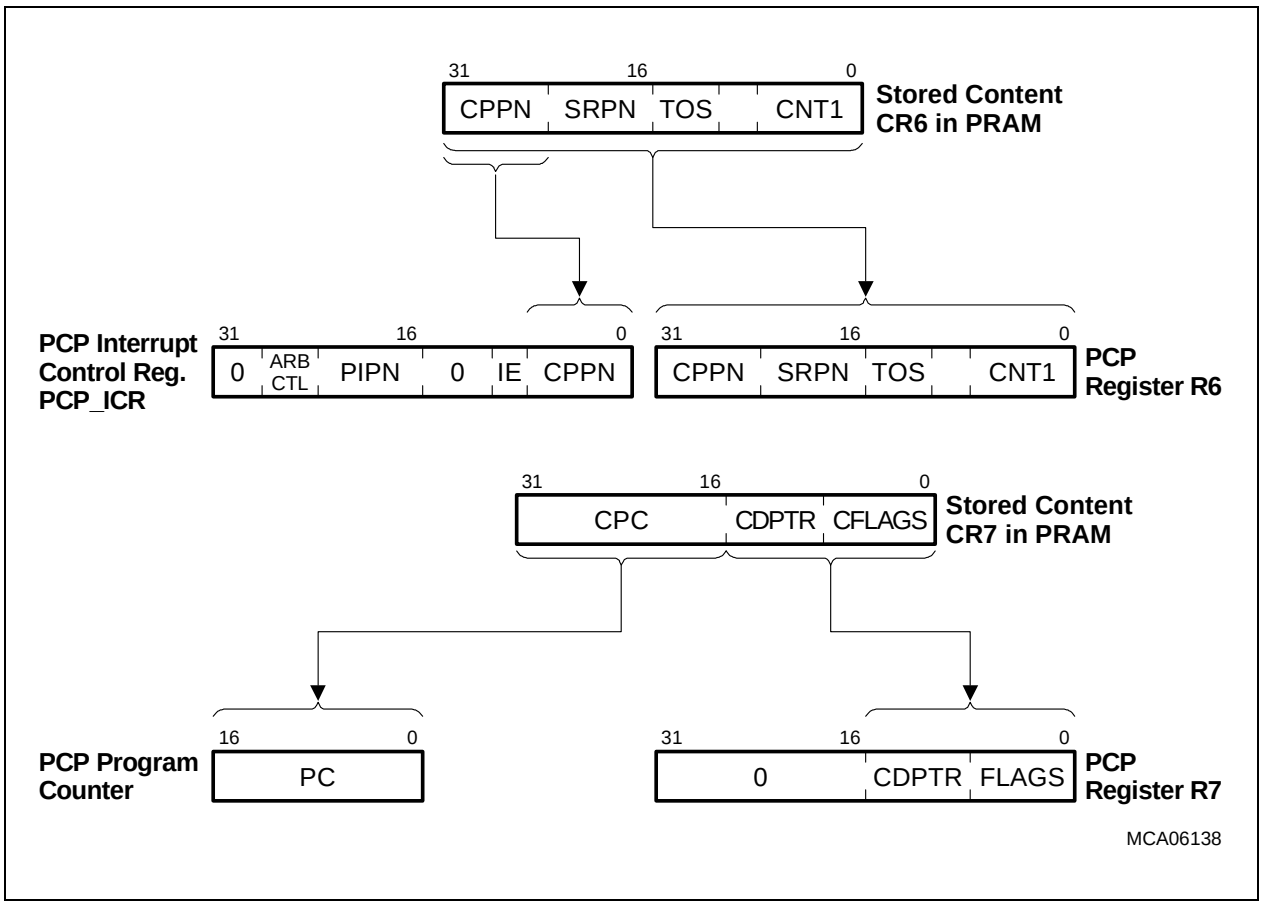


Figure 10-4 Context Restore: Channel Start in “Channel Resume Mode”

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 10-25**). 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.

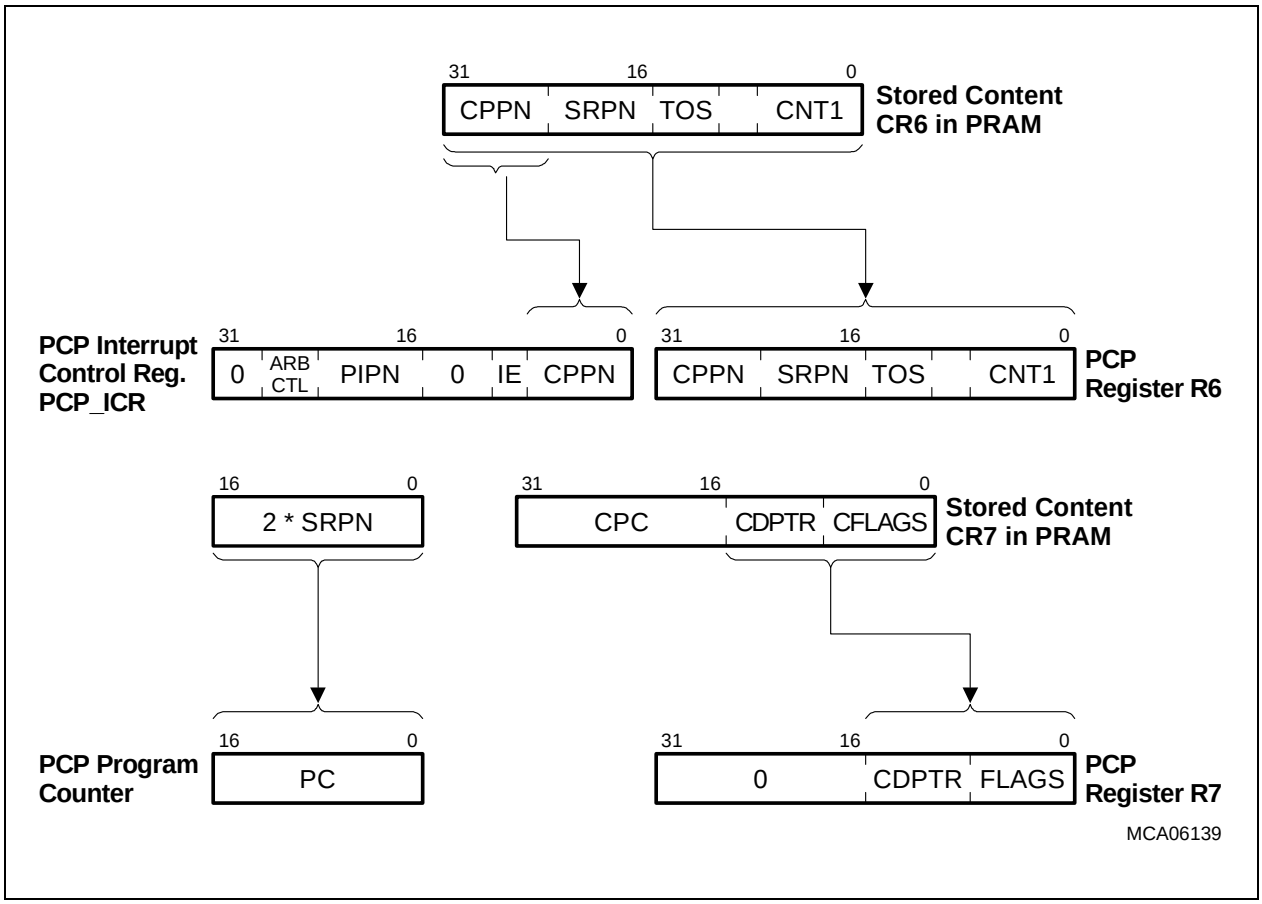
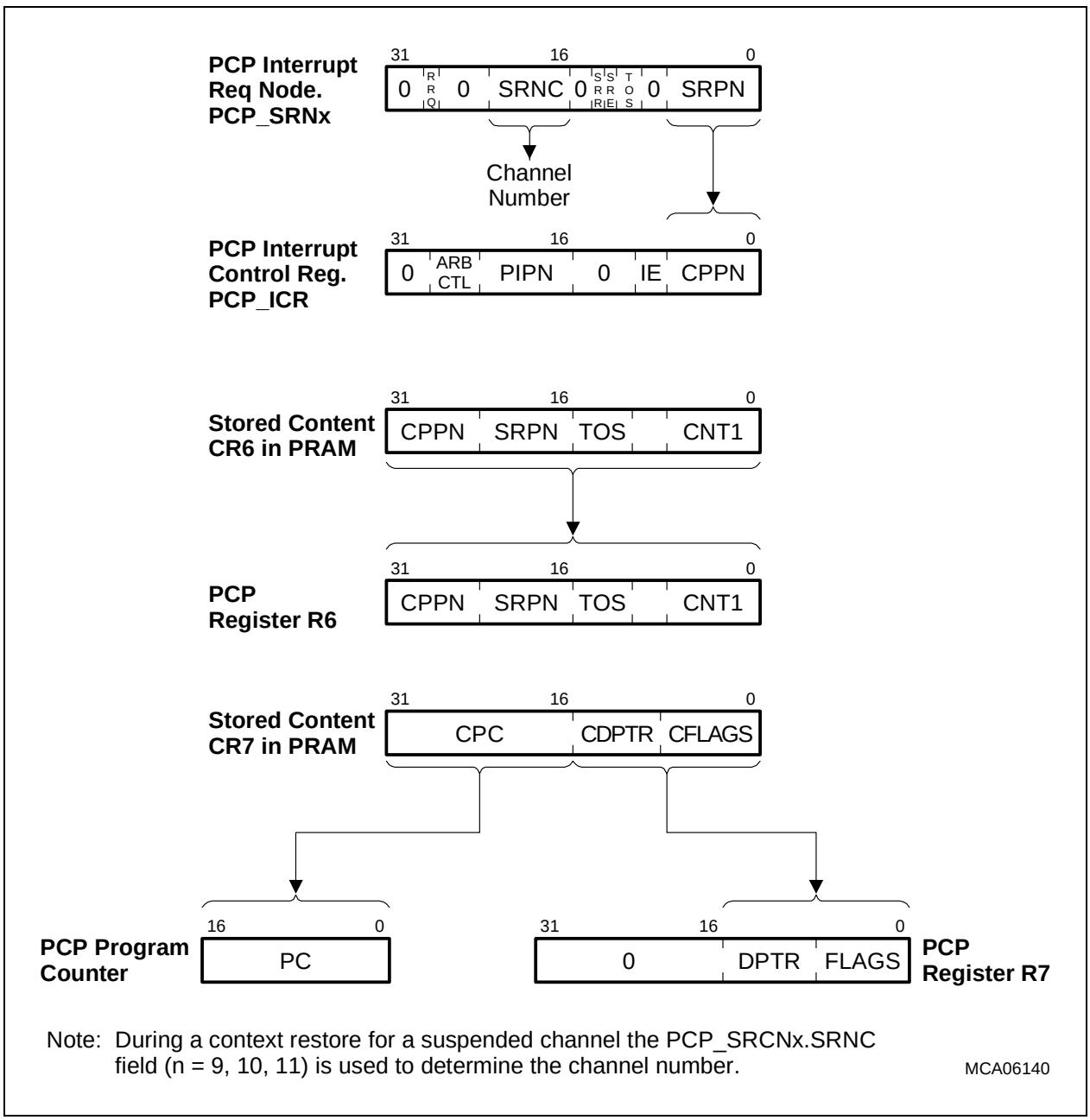


Figure 10-5 Context Restore: Channel Start in Channel Restart Mode

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 10-70**).



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

10.3.2.4 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.

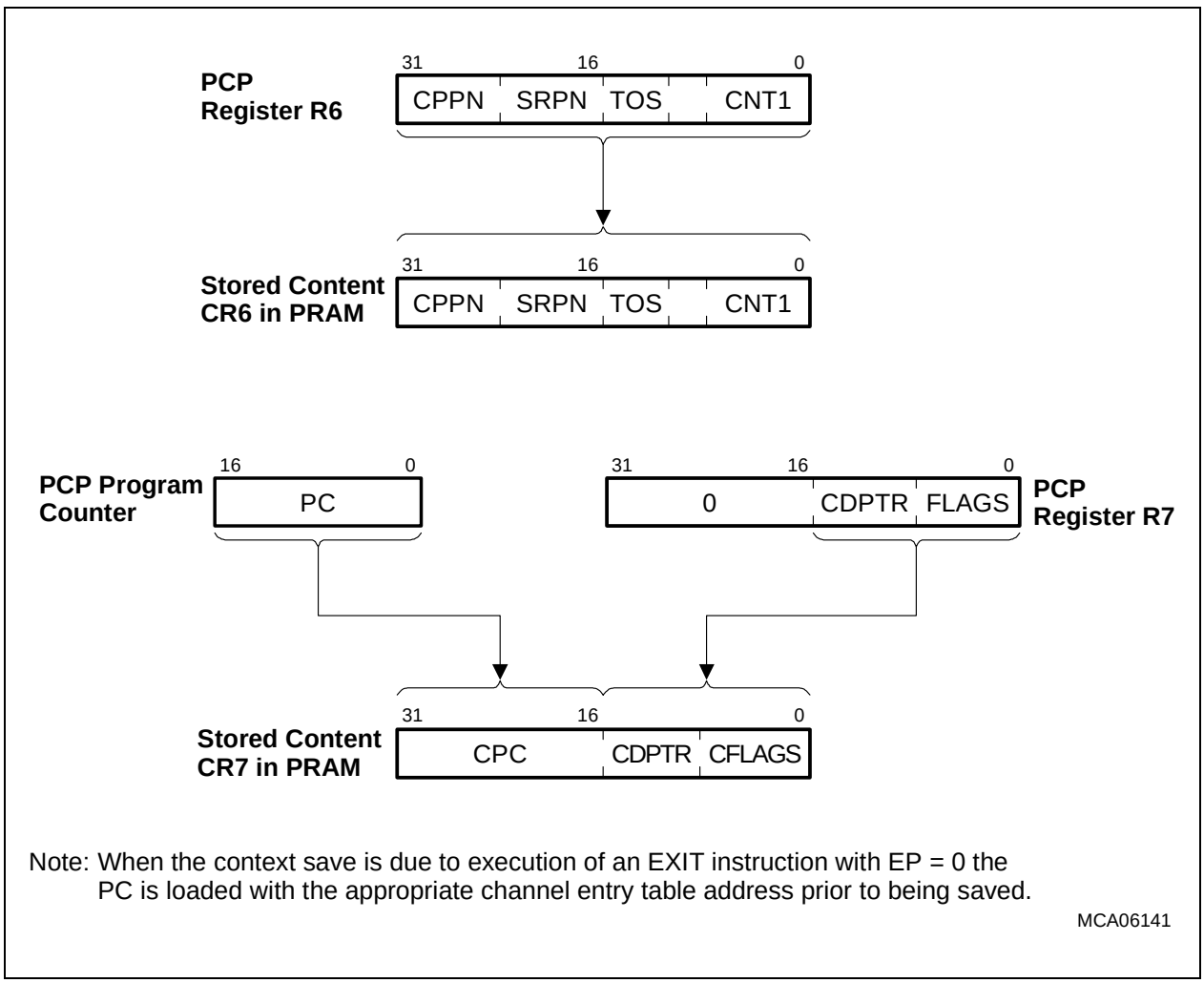


Figure 10-7 Context Save: Channel Exit in Channel Resume Mode

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].

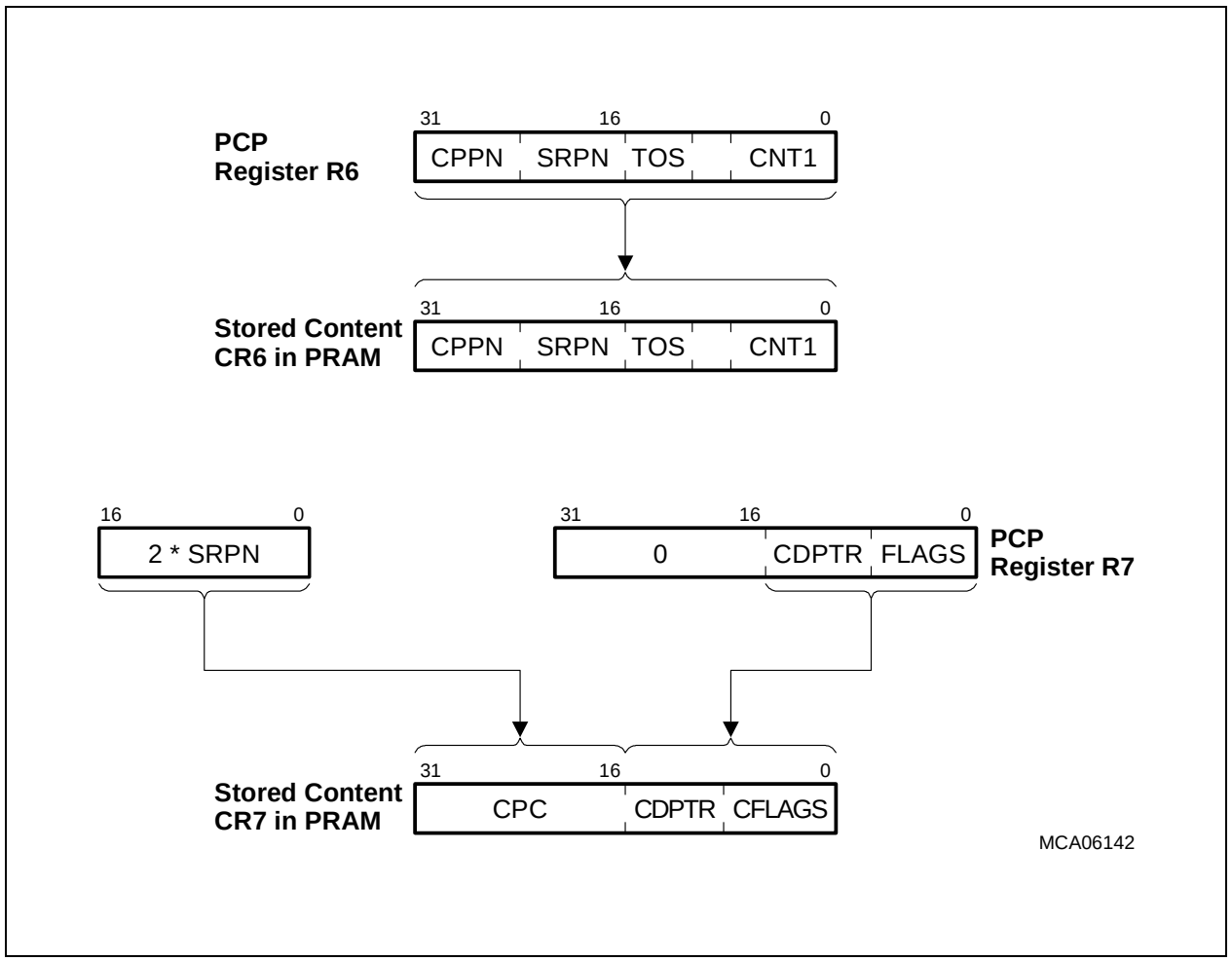


Figure 10-8 Context Save: Channel Exit in Channel Restart Mode

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 10-70**) 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.

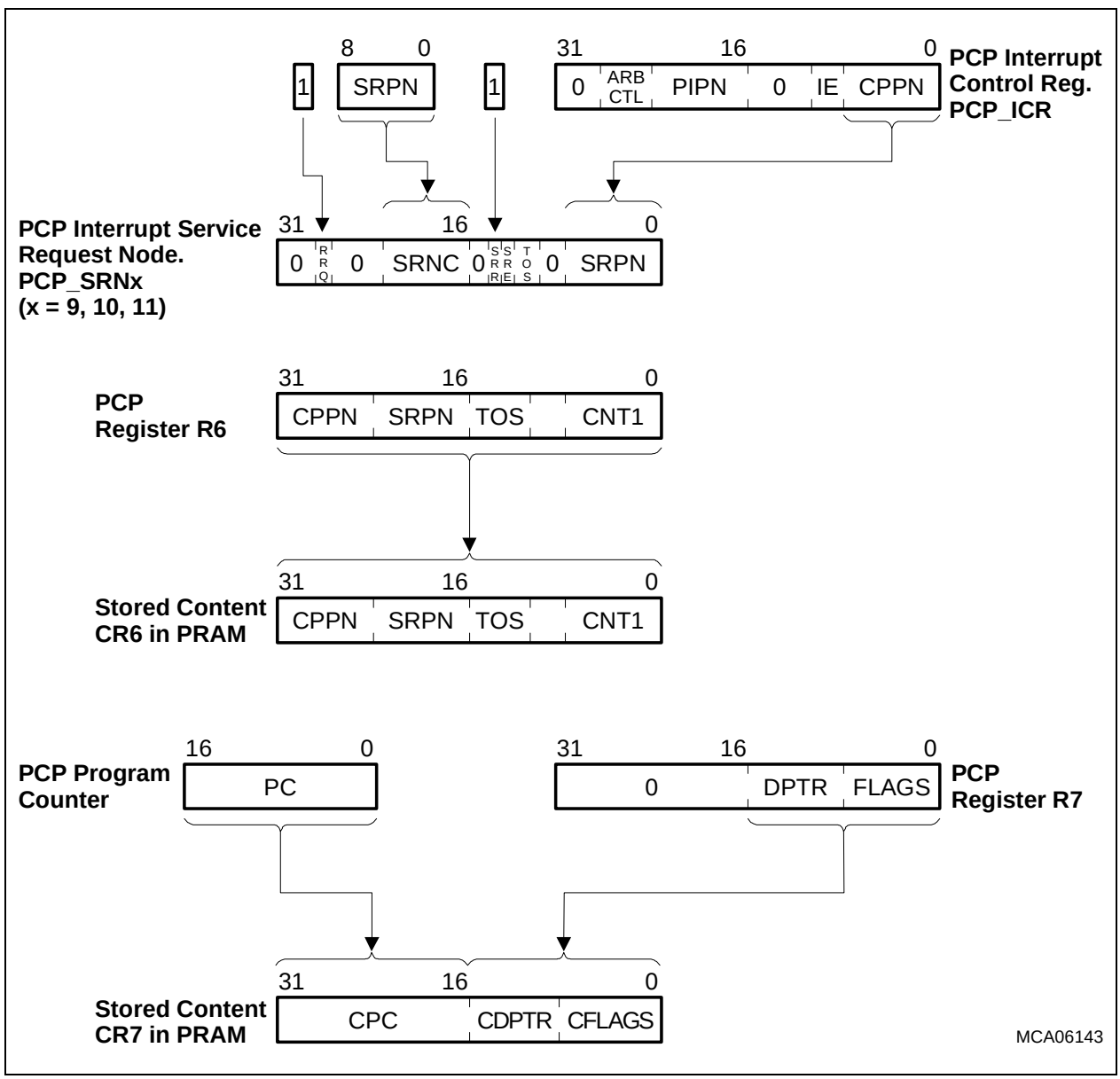


Figure 10-9 Context Save: Channel Suspend

10.3.2.5 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 10-48** for details).

10.3.2.6 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.

10.3.3 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 TC1766 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 10-56**).

10.3.3.1 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 10-89**).

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.

10.3.3.2 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 10-89**). 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.

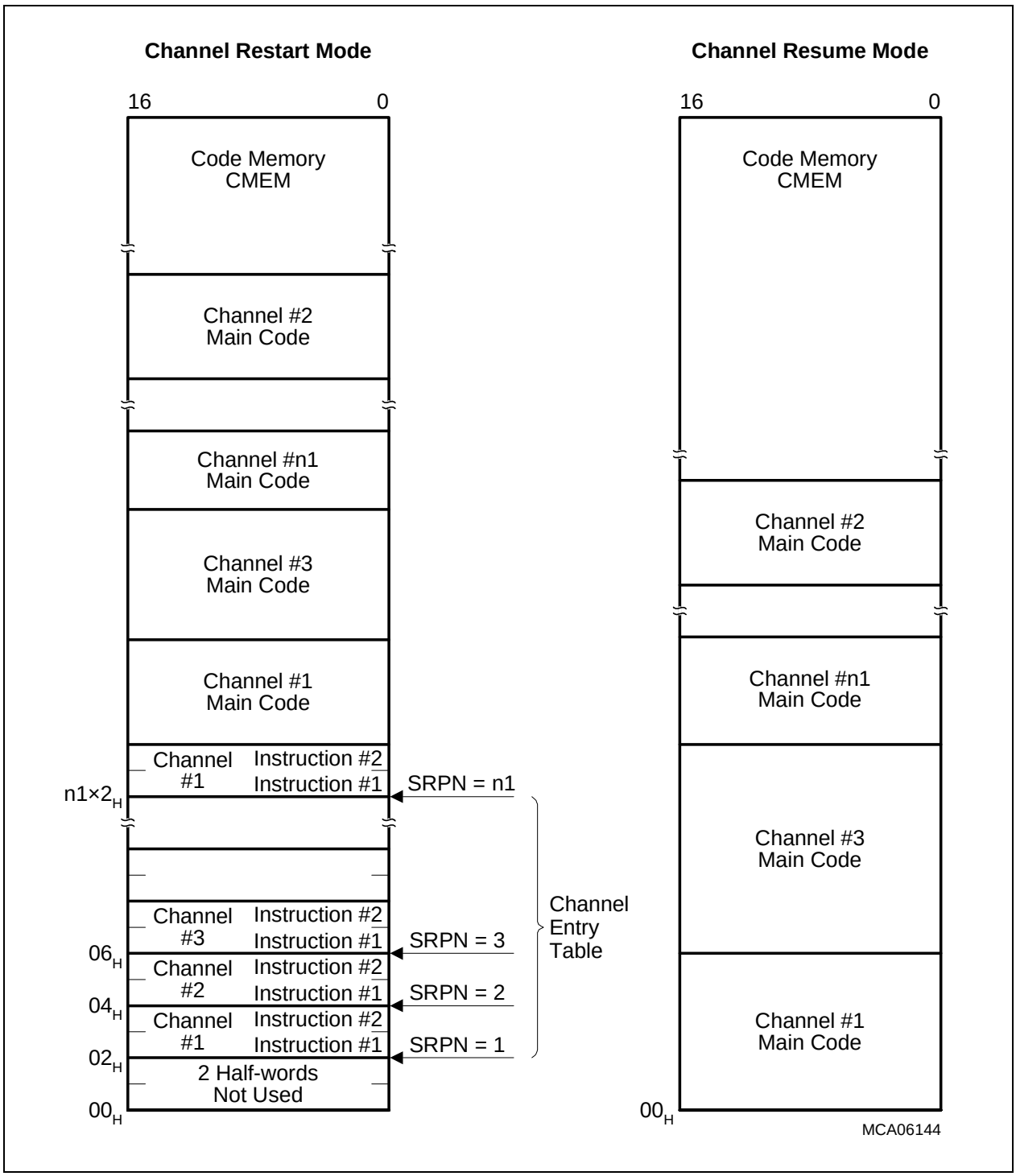


Figure 10-10 Examples of Code Memory Organization for Channel Restart and Channel Resume Modes

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.

10.4 PCP Operation

This section describes how to initialize the PCP, how to invoke a channel program, and the general operation of the PCP.

10.4.1 PCP Initialization

The PCP is placed in a quiescent state when the TC1766 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.

10.4.2 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 10-29).
- 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).

10.4.3 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.

10.4.3.1 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 10-89). 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.

10.4.3.2 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.

10.4.3.3 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.

10.5 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.

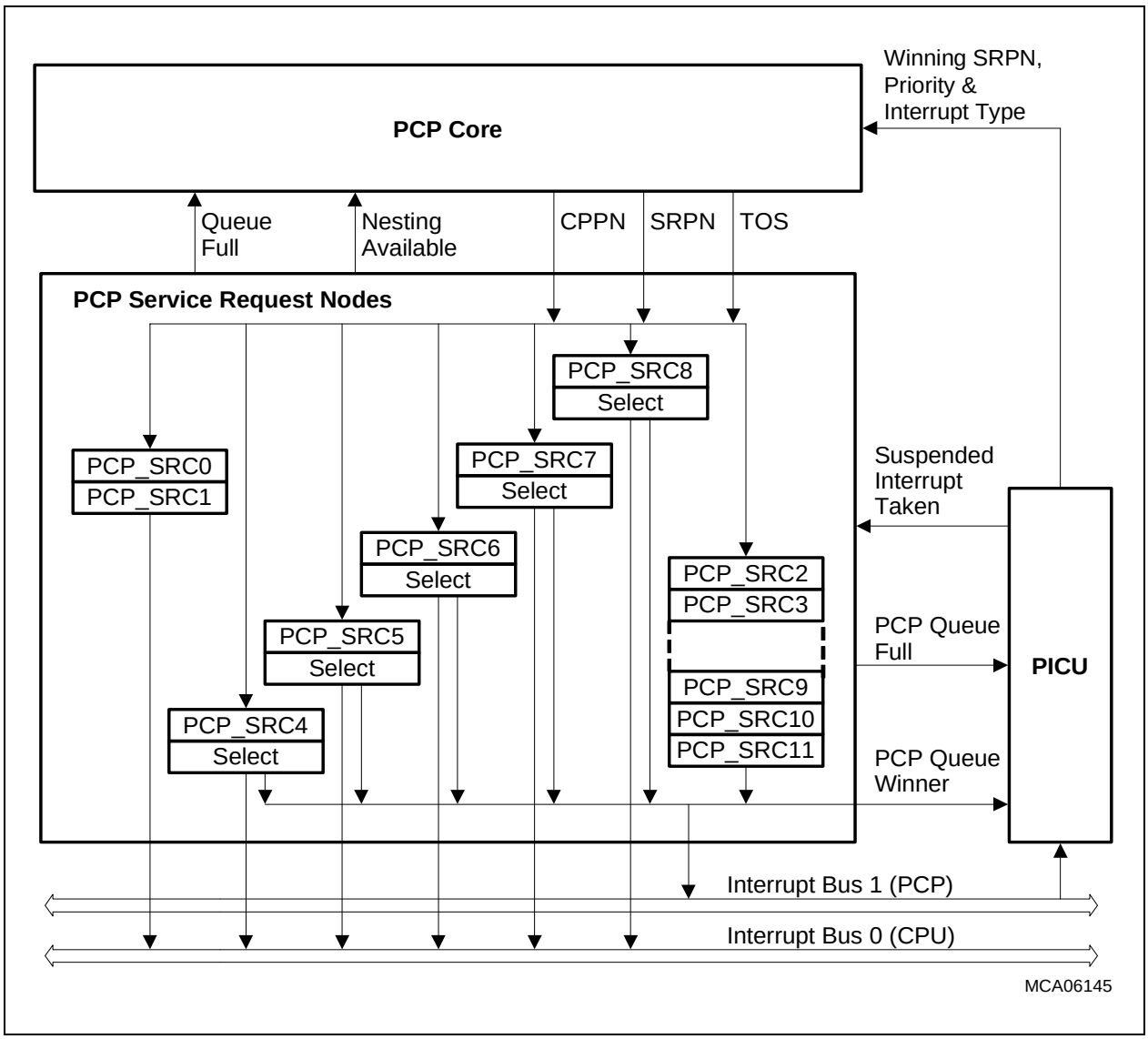


Figure 10-11 PCP Interrupt Block Diagram

10.5.1 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.

10.5.2 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 clears 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 10-36**.

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.

10.5.3 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 of two nodes each.

- 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 PCPSRC8, 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 TC1766) 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.

10.5.4 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.

10.5.4.1 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 10-36** 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.

10.5.4.2 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, PCPSRC11). 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.

10.5.4.3 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.

10.5.4.4 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.

10.6 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 10-30**).

10.6.1 Enforced PRAM Partitioning

As previously discussed, the PRAM can be considered as being split into two distinct areas. The lower of these two areas is the CSA (see **Page 10-14**) used for storing context information for each active channel while the channel program is not actually executing. The remainder of PRAM is available for general use and is typically used to hold variables and global data.

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 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.

To avoid spurious PCP operation as a result of either of these programming errors, the PCP can be optionally configured via the global PCP control and status register (PCP_CS) to enforce strict partitioning 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 10-56**). 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.

10.6.2 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.

10.6.3 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.

10.6.4 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 10-124** 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.

10.7 Instruction Set Overview

The following sections present an overview of the instruction set and the available addressing modes of the PCP in the TC1766.

10.7.1 DMA Primitives

Table 10-3 describes the two DMA instructions of the PCP.

Table 10-3 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.

10.7.2 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 10-4 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

10.7.3 Arithmetic and Logical Instructions

Arithmetic instructions that are fully register-based execute conditionally depending on the specified Condition Code A (see Page 10-73). 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 10-5 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 10-6 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
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.

10.7.4 Bit Manipulation

All bit manipulation instructions except INB are executed unconditionally. If conditional bit handling is required, INB should be used.

Table 10-7 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

10.7.5 Flow Control

Table 10-8 describes flow control instructions of the PCP in the TC1766.

Table 10-8 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)

10.7.6 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.

10.7.6.1 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.

10.7.6.2 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).

10.7.6.3 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.

10.7.6.4 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).

10.8 Accessing PCP Resources from the FPI Bus

Any FPI Bus master (on the TC1766's System Peripheral Bus) can access the three distinct PCP address ranges from the FPI Bus side. 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.

10.8.1 Access to the PCP Control Registers

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. Write operations are only possible to the PCP_CS register, all other register are read-only. Register PCP_CS can be optionally Endinit-protected via bit PCP_CS.EIE (see **Page 10-56**).

10.8.2 Access to the PRAM

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 10-9** shows these addresses.

Table 10-9 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)

10.8.3 Access to the CMEM

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

10.9 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 $\overline{\text{BRKOUT}}$ (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 10-30**). 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.

10.10 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. **Figure 10-12** gives an overview of the PCP registers.

PCP Register Overview

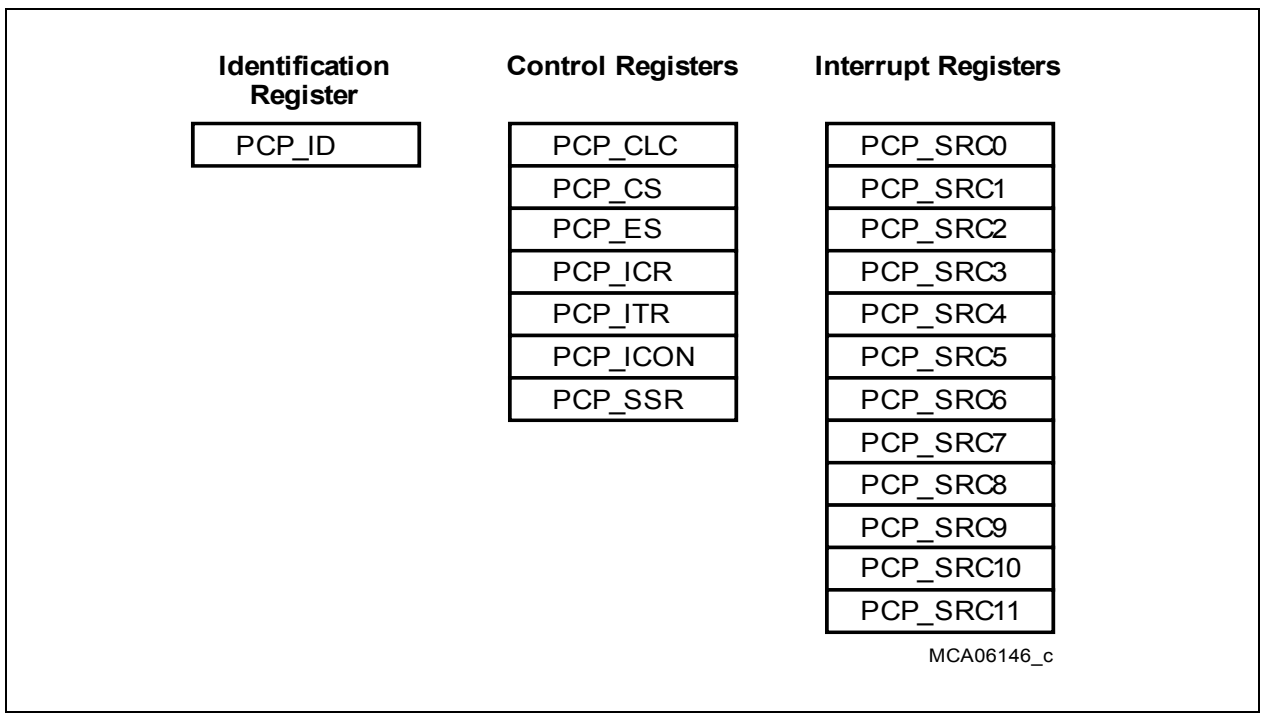


Figure 10-12 PCP Registers

The complete address map of the PCP is described in **Table 16-18** on **Page 16-48** of this TC1766 System Units (Vol. 1 of 2) User's Manual.

Table 10-10 Registers Address Space - PCP Registers

Module	Base Address	End Address	Note
PCP	F004 3F00$_H$	F004 3FFF$_H$	-

Table 10-11 Registers Overview - PCP Registers

Register Short Name	Register Long Name	Offset Address[1)]	Description see
PCP_CLC	PCP Clock Control Register	0000_H	Page 10-55
PCP_ID	PCP Module Identification Register	0008_H	Page 10-54
PCP_CS	PCP Control/Status Register	0010_H	Page 10-56
PCP_ES	PCP Error/Debug Status Register	0014_H	Page 10-59
PCP_ICR	PCP Interrupt Control Register	0020_H	Page 10-61
PCP_ITR	PCP Interrupt Threshold Control Register	0024_H	Page 10-63
PCP_ICON	PCP Interrupt Configuration Register	0028_H	Page 10-64
PCP_SSR	PCP Stall Status Register	$002C_H$	Page 10-66
PCP_SRC11	PCP Service Request Control Register 11	$00D0_H$	Page 10-70
PCP_SRC10	PCP Service Request Control Register 10	$00D4_H$	
PCP_SRC9	PCP Service Request Control Register 9	$00D8_H$	
PCP_SRC8	PCP Service Request Control Register 8	$00DC_H$	Page 10-69
PCP_SRC7	PCP Service Request Control Register 7	$00E0_H$	
PCP_SRC6	PCP Service Request Control Register 6	$00E4_H$	
PCP_SRC5	PCP Service Request Control Register 5	$00E8_H$	
PCP_SRC4	PCP Service Request Control Register 4	$00EC_H$	
PCP_SRC3	PCP Service Request Control Register 3	$00F0_H$	Page 10-68
PCP_SRC2	PCP Service Request Control Register 2	$00F4_H$	
PCP_SRC1	PCP Service Request Control Register 1	$00F8_H$	Page 10-67
PCP_SRC0	PCP Service Request Control Register 0	$00FC_H$	

1) The absolute register address is calculated as follows:
Module Base Address (Table 10-10) + Offset Address (shown in this column)

Accessing of Control Registers

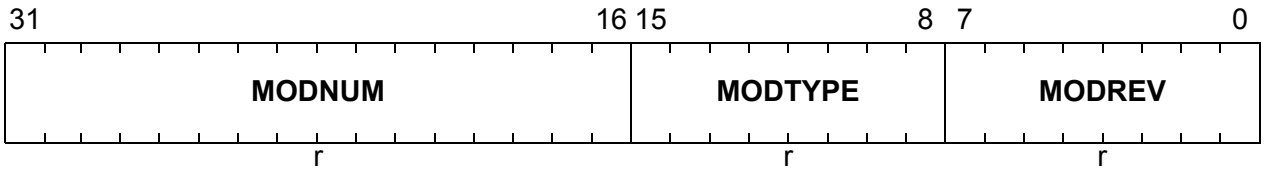
The control registers are accessible by any master via the FPI Bus. The control registers must be configured at initialization and then left unaltered. This is typically done by the CPU. The only setting that can be dynamically modified is the PCP_CS.EN bit. All other bits must only be modified when 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.

10.10.1 Module Identification Register, PCP_ID

The PCP Module Identification Register ID contains read-only information about the PCP module version.

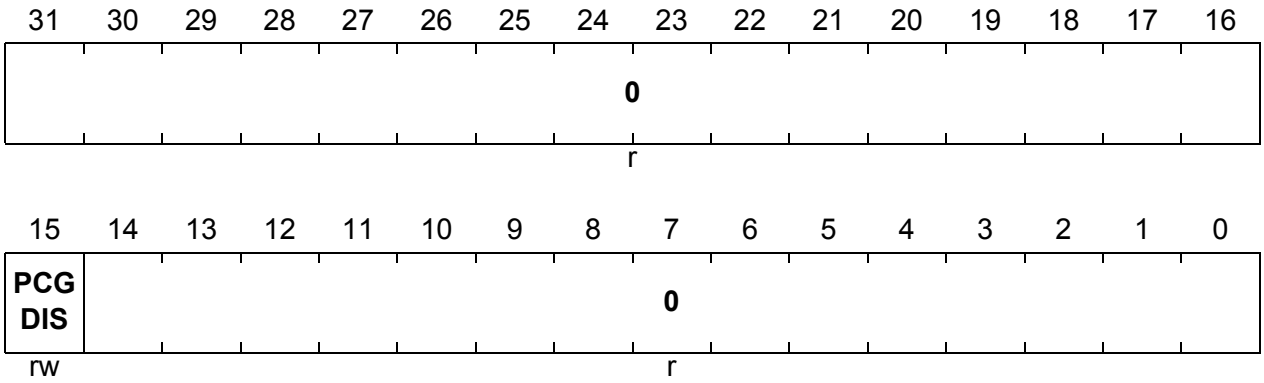
PCP_ID
PCP Module Identification Register (008$_H$) **Reset Value: 0020 C0XX$_H$**



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 PCP: 0020$_H$

10.10.2 PCP Clock Control Register, PCP_CLC

PCP_CLC
PCP Clock Control Register **(00_H)** **Reset Value: 0000 0000_H**

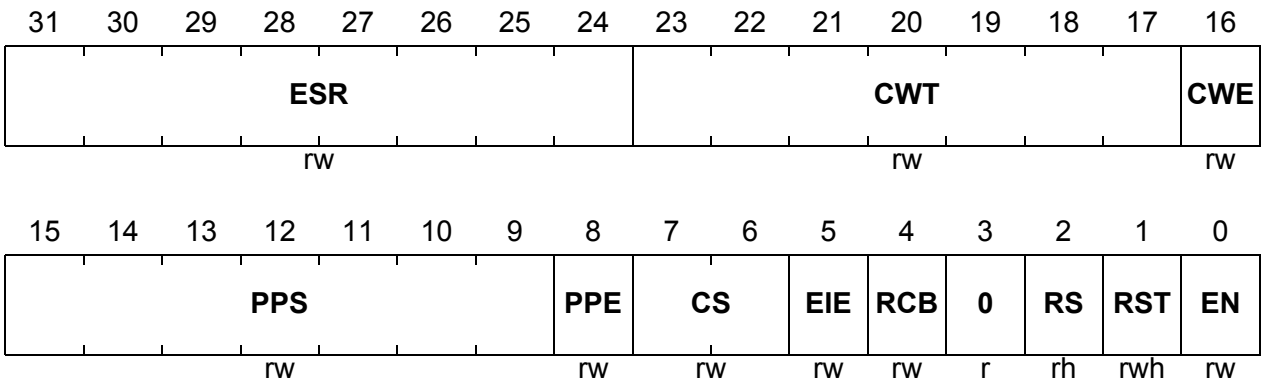


Field	Bits	Type	Description
PCGDIS	15	rw	**Clock Gating Disable Bit** Allows clock gating to be disabled. 0_B PCP Internal Clock stops when PCP is idle (default after reset) 1_B PCP Internal Clock always runs
0	[14:0], [31:16]	r	**Reserved** Read as 0; should be written with 0.

10.10.3 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$_H$



Field	Bits	Type	Description
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.*
RST	1	rwh	**PCP Reset Request** 0_B No PCP soft reset operation is requested 1_B A PCP soft reset is requested. Halt any operating channel. Reset all control registers to default values. Reset PCP state to default value. RST is always read as 0, but is written with 1 in order to initiate a reset. See also **Page 10-125**.
RS	2	rh	**PCP Run/Stop Status Flag** 0_B PCP is stopped or idle (default) 1_B PCP is currently running

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.*
EIE	5	rw	**Endinit Enable** 0_B Writes to PCP_CS are disabled if Endinit-protection is enabled. 1_B Writes to PCP_CS are enabled if Endinit-protection is enabled.
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
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).*

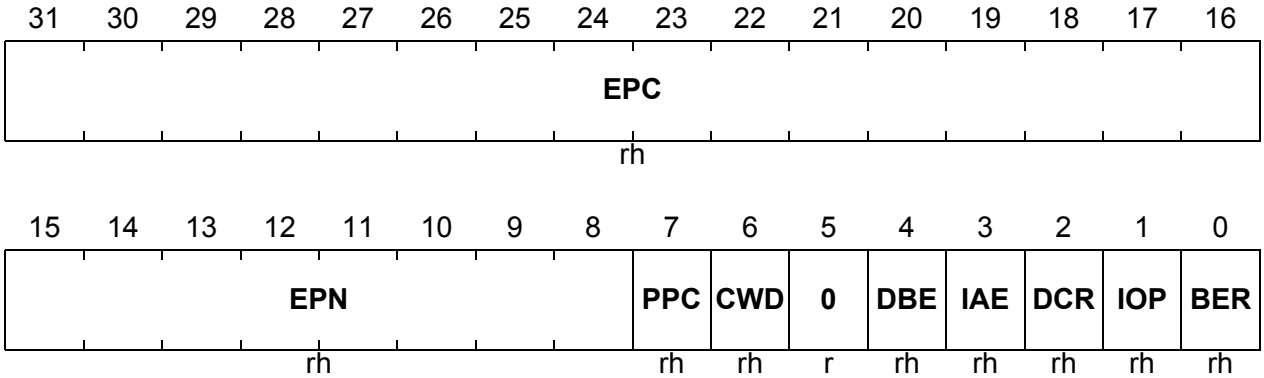
Field	Bits	Type	Description
PPS	[15:9]	rw	**PRAM Partition Size** 0_D Default, only allowed with PPE = 0 1_D CSA contains 3 context save regions … … n_D CSA contains 1 + 2 × n context save regions *Note: The actual size of the CSA (in words) is given by the formula (2 × n + 1) × m, where m is the number 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. If n = 1, 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.*
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.*
CWT	[23:17]	rw	**Channel Watchdog Threshold** 0_D Reserved 1_D Threshold = 16 instructions … … n_D Threshold = 16 × ‘n’ instructions
ESR	[31:24]	rw	**Error Service Request Number** SRPN for interrupt to CPU on an error condition. 00_H No interrupt request posted (default) value n_H Post n as SRNP interrupt to CPU on an error condition (n not equal 00_H)

Field	Bits	Type	Description
0	3	r	**Reserved** Read as 0; should be written with 0.

10.10.4 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
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.*
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.

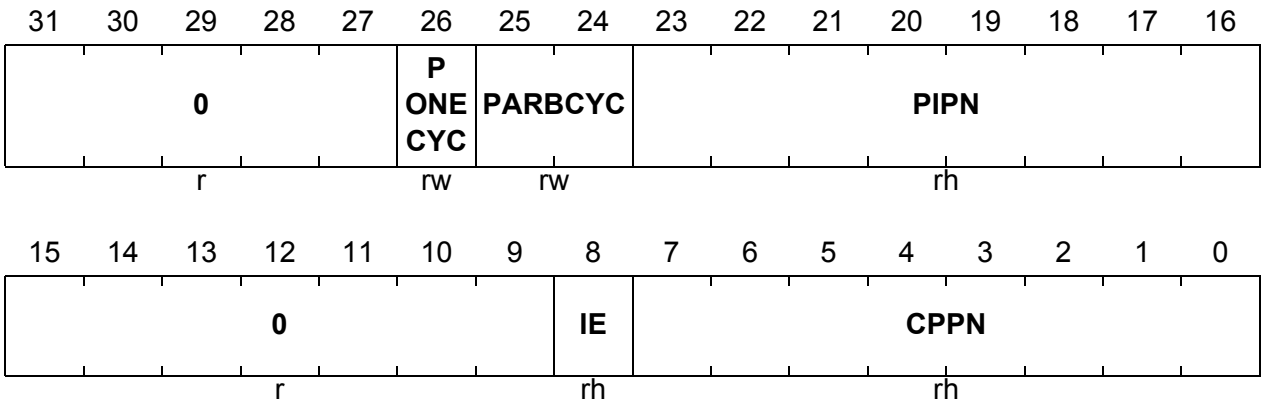
Field	Bits	Type	Description
DCR	2	rh	**Disabled Channel Request Flag** Set if the last error/debug event was an error generated by receipt of an interrupt request with an SRPN that attempted to start a disabled PCP channel; otherwise, clear.
IAE	3	rh	**Instruction Address Error** Set if the last error/debug 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.
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.*
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.
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.
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.
EPC	[31:16]	rh	**Error PC** PC value of the instruction that was executing when an error or debug event occurred. Default = 0000_H.
0	5	r	**Reserved** Read as 0.

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.

10.10.5 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: $0000\ 0000_H$



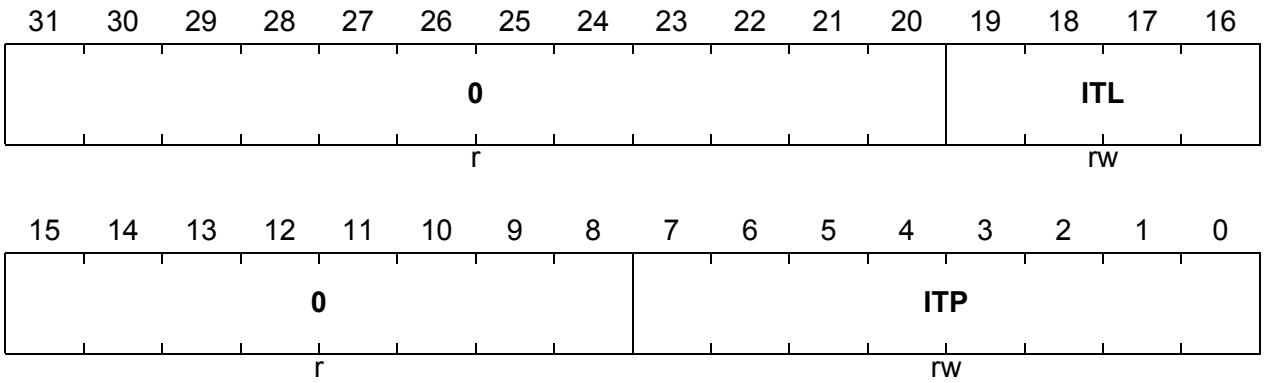
Field	Bits	Type	Description
CPPN	[7:0]	rh	**Current PCP Priority Number** This field indicates the current priority level of the PCP and is automatically updated by hardware on entry into an interrupt service routine.
IE	8	rh	**Reserved**
PIPN	[23:16]	rh	**Pending Interrupt Priority Number** This read-only 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.

Field	Bits	Type	Description
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
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
0	[15:9], [31:27]	r	**Reserved** Read as 0; should be written with 0.

10.10.6 PCP Interrupt Threshold Register, PCP_ITR

PCP_ITR
PCP Interrupt Threshold Control Register

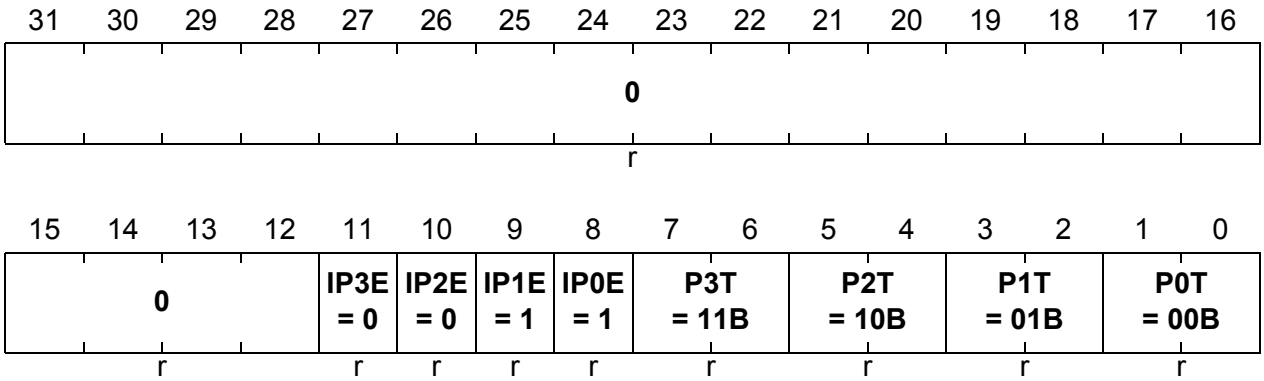
(24_H) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
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).
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], [31:20]	r	**Reserved** Read as 0; should be written with 0.

10.10.7 PCP Interrupt Configuration Register, PCP_ICON

PCP_ICON
PCP Interrupt Configuration Register (28_H) **Reset Value: 0000 03E4$_H$**

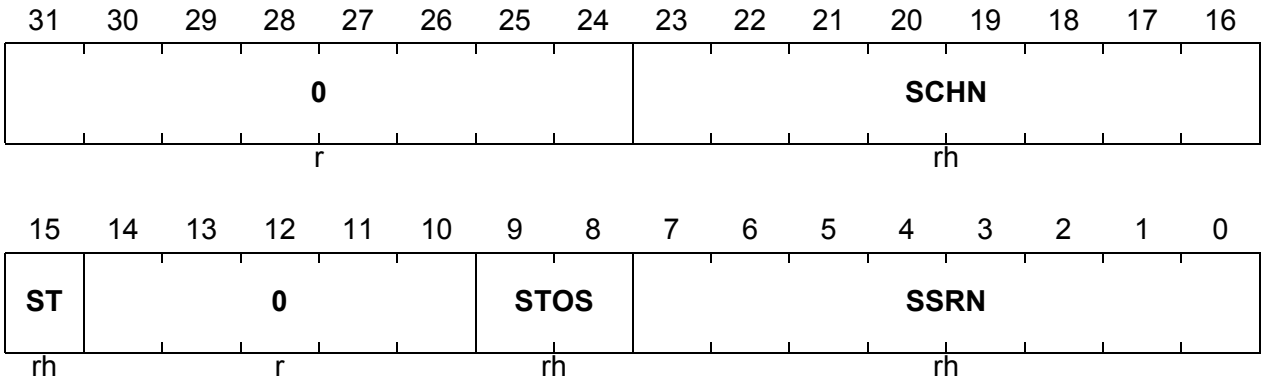


Field	Bits	Type	Description
P0T	[1:0]	r	**PCP Interrupt Bus 0 TOS Mapping** This field reflects the TOS associated with interrupt bus 0 (CPU interrupt arbitration bus). The PCP should use this value in R6.TOS when it wishes to raise an interrupt request via interrupt bus 0.
P1T	[3:2]	r	**PCP Interrupt 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).
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 TC1766.*
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 TC1766.*
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.

Field	Bits	Type	Description
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.
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 TC1766).
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 TC1766).
0	[31:12]	r	**Reserved** Read as 0.

10.10.8 PCP Stall Status Register, PCP_SSR

PCP_SSR
PCP Stall Status Register **($2C_H$)** **Reset Value: $0000\ 0000_H$**



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.
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.
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
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.
0	[14:10], [31:24]	r	**Reserved** Read as 0.

10.10.9 PCP Service Request Control Registers n, PCP_SRC[1:0]

Service request nodes for interrupt bus 0 (CPU interrupt arbitration bus).

PCP_SRCx (x = 0-1)
PCP Service Request Control Register x

(FC_H-x*4_H) **Reset Value: 0000 1000_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		SRR	SRE = 1	TOS = 00B		0		SRPN							
r		rh	r	r		r		rh							

Field	Bits	Type	Description
SRPN	[7:0]	rh	**PCP Node x 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.
TOS	[11:10]	r	**PCP Node x Type-of-Service State** Always read as 00_B. This means TOS is associated with interrupt bus 0 (CPU interrupt arbitration bus).
SRE	12	r	**PCP Node x Service Request Enable** Always read as 1 (enabled).
SRR	13	rh	**PCP Node x Service Request Flag** 0_B No service requested (default). 1_B Valid active service requested.
0	[9:8], [15:14], [31:16]	r	**Reserved** Read as 0.

10.10.10 PCP Service Request Control Registers n, PCP_SRC[3:2]

Service request nodes for interrupt bus 1 (PCP interrupt arbitration bus).

PCP_SRCx (x = 2-3)
PCP Service Request Control Register x

(FC_H-x*4_H) **Reset Value: 0000 1400$_H$**

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		SRR	SRE = 1	TOS = 01B		0		SRPN							
r		rh	r	r		r		rh							

Field	Bits	Type	Description
SRPN	[7:0]	rh	**PCP Node x 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.
TOS	[11:10]	r	**PCP Node x Type-of-Service State** Always read as 01_B. This means TOS is associated with interrupt bus 1 (PCP interrupt arbitration bus).
SRE	12	r	**PCP Node x Service Request Enable** Always read as 1 (enabled).
SRR	13	rh	**PCP Node x Service Request Flag** 0_B No service requested (default). 1_B Valid active service requested.
0	[9:8], [15:14], [31:16]	r	**Reserved** Read as 0.

10.10.11 PCP Service Request Control Registers n, PCP_SRC[8:4]

Service request nodes programmable for interrupt bus 0 (CPU interrupt arbitration bus) or 1 (PCP interrupt arbitration bus).

PCP_SRCx (x = 4-8)
PCP Service Request Control Register x
(FC_H-x*4_H) **Reset Value: 0000 1000_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		SRR	SRE = 1	TOS		0		SRPN							
r		rh	r	rw		r		rh							

Field	Bits	Type	Description
SRPN	[7:0]	rh	**PCP Node x 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.
TOS	[11:10]	rw	**PCP Node x Type-of-Service State/Control** TOS value depends on the interrupt mapping that has been selected (see **Page 10-118**). This bit field must be written at PCP configuration time and should not subsequently modified while the PCP is operating.
SRE	12	r	**PCP Node x Service Request Enable** Always read as 1 (enabled).
SRR	13	rh	**PCP Node x Service Request Flag** 0_B No service requested (default). 1_B Valid active service requested.
0	[9:8], [15:14], [31:16]	r	**Reserved** Read as 0; should be written with 0.

10.10.12 PCP Service Request Control Registers n, PCP_SRC[11:9]

Service request nodes for interrupt bus 1 (PCP interrupt arbitration bus) with suspended interrupt capability.

PCP_SRCx (x = 9-11)
PCP Service Request Control Register x
(FC_H-x*4_H) **Reset Value: 0000 1400$_H$**

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		SRR	SRE = 1	TOS = 01B		0		SRPN							
r		rh	r	r		r		rh							

Field	Bits	Type	Description
SRPN	[7:0]	rh	**PCP Node x 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.
TOS	[11:10]	r	**PCP Node x Type-of-Service State** Always read as 01_B. This means TOS is associated with interrupt bus 1 (PCP interrupt arbitration bus).
SRE	12	r	**PCP Node x Service Request Enable** Always read as 1 (enabled).

Field	Bits	Type	Description
SRR	13	rh	**PCP Node x Service Request Flag** 0_B No service requested (default) 1_B Valid active service requested
SRCN	[23:16]	rh	**PCP Node x 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.
RRQ	28	rh	**PCP Node x 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	[9:8], [15:14], [27:24], [31:29]	r	**Reserved** Read as 0.

10.11 PCP Instruction Set Details

This section describes the instruction set architecture of the PCP in detail.

10.11.1 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.

10.11.1.1 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 10-12**.

Table 10-12 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

10.11.1.2 Instruction Fields

Table 10-13 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 10-13 Instruction Field Definitions

Symbol	Syntax	Description
CNC		**Counter Control** This field is used by the COPY instruction to control the number of repetitions of the data transfer. See also Figure 10-13 at Page 10-77.
	$CNC = 00_B$	Decrement CNT0 after each transfer. Continue until CNT0 = 0, and proceed to next instruction.
	$CNC = 01_B$	Post Decrement CNT0 after each transfer. Continue until CNT0 = 0, then decrement CNT1 and proceed to next instruction.
	$CNC = 10_B$	Post Decrement CNT0 after each transfer. Continue until CNT0 = 0, then decrement CNT1. Reload CNT0 value and continue. Continue 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 10-13 at Page 10-77.
	$CNT0 = 001_B..111_B$	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.
	$CNT0 = 000_B$	Use block size of 8 words.
	$CNT0 = 010_B$	Use block size of 2 words.
	$CNT0 = 011_B$	Use block size of 4 words.
	Others	Reserved
cc_A, cc_B	see Table 10-12	**Condition Code** Specifies conditional execution of instruction according to CONDCA or CONDCB.

Table 10-13 Instruction Field Definitions (cont'd)

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)*
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

Table 10-13 Instruction Field Definitions (cont'd)

Symbol	Syntax	Description
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
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
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)

10.11.2 Counter Operation for COPY Instruction

Figure 10-13 shows the flow of a COPY instruction.

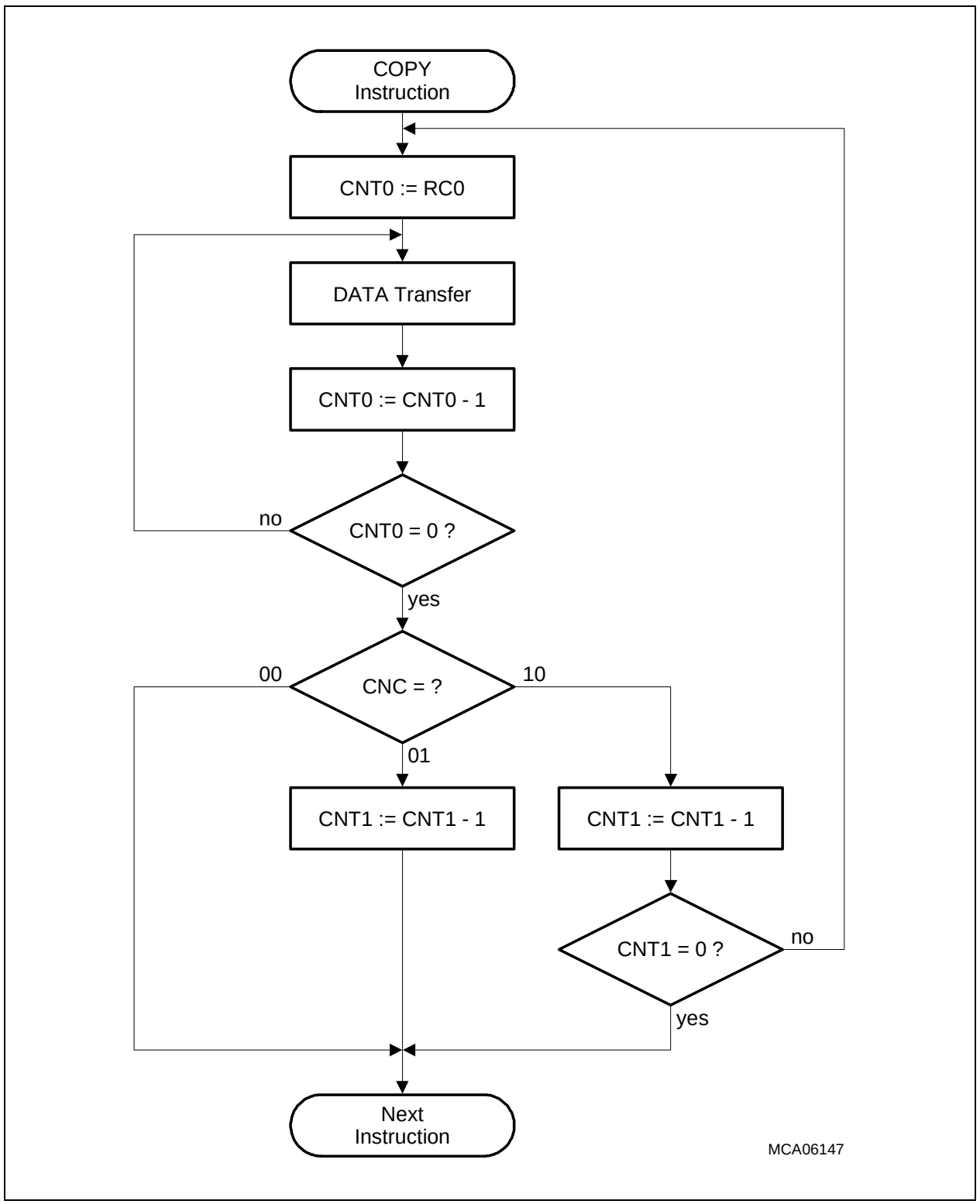


Figure 10-13 Counter Operation for COPY Instruction

10.11.3 Counter Operation for BCOPY Instruction

Figure 10-14 shows the flow of a BCOPY instruction.

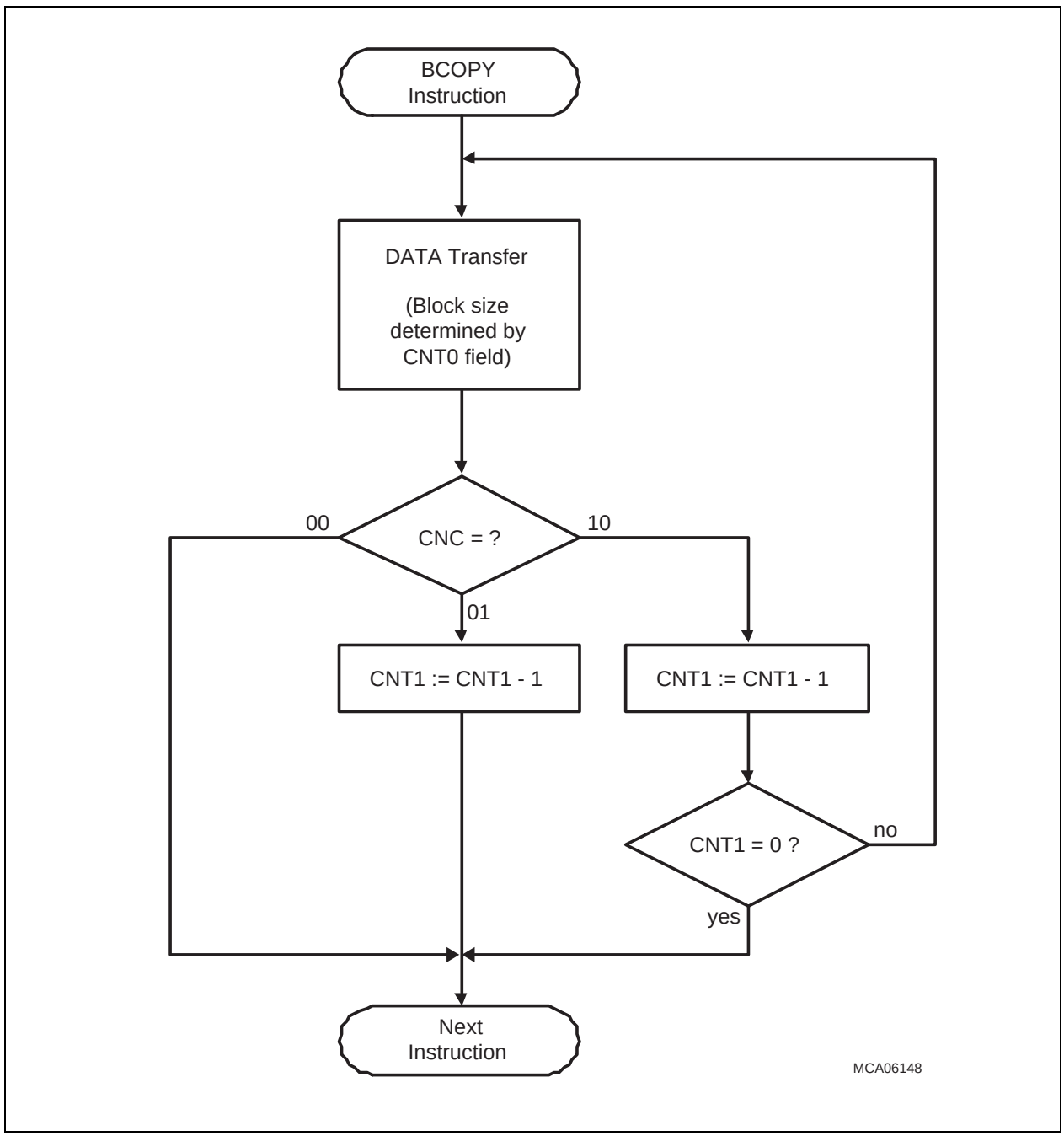


Figure 10-14 Counter Operation for BCOPY Instruction

10.11.4 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 10-121** 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 10-122** 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 10-120****).*

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.

10.11.5 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

10.11.6 BCOPY, DMA Operation

This section describes the BCOPY instruction of the PCP in the TC1766.

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 10-13** at **Page 10-74**): Source pointer (SRC+-): Increment, decrement or unchanged Destination pointer (SRC+-): Increment, decrement or unchanged Counter control (CNC): see **Table 10-13** Block size value (CNT0): see **Table 10-13**
	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 10-14** on **Page 10-78** and **Table 10-13** on **Page 10-74**.
	Flags	CN1Z

See also **Page 10-124** for TC1766 specific details of the BCOPY instruction.

10.11.7 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

10.11.8 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

10.11.9 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

10.11.10 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 sign-extended 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] - sign_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] - sign_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

10.11.11 COPY, DMA Instruction

This section describes the COMP instruction of the PCP.

COPY	Syntax	**COPY DST+-, SRC+-, CNC, RC0, 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 10-13** on **Page 10-74**): Source pointer (SRC+-): Increment, decrement or unchanged Destination pointer (SRC+-): Increment, decrement or unchanged Counter control (CNC): see **Table 10-13** Counter 0 reload value (CNT0): see **Table 10-13** 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 10-13** on **Page 10-77** and **Table 10-13** on **Page 10-74**.
	Flags	CN1Z

10.11.12 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.

10.11.13 DINIT, Divide Initialization Instruction

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

10.11.14 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 10-79** are followed, then the above description and operation are correct. Failure to adhere to these rules will yield undefined results.*

10.11.15 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**
A	Description	Insert the carry flag R7.C into register Ra at the bit position specified through the immediate value imm5.
A	Operation	R[a][imm5] = R7_C
A	Flags	None

10.11.16 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

10.11.17 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

10.11.18 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

10.11.19 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

10.11.20 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

10.11.21 Multiply Initialization Instruction

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

10.11.22 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

10.11.23 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.

10.11.24 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

10.11.25 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

10.11.26 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

10.11.27 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

10.11.28 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

10.11.29 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.

10.11.30 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

10.11.31 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

10.11.32 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

10.11.33 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

10.11.34 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

10.11.35 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

10.11.36 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

10.11.37 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

10.11.38 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

10.11.39 Flag Updates of Instructions

Most instructions update the state flags in R7. In **Table 10-14**, each instruction is shown with the flags that it updates.

Table 10-14 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 10-14 Flag Updates (cont'd)

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.

10.11.40 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 10-15** are PCP module clock cycles. In the TC1766, the PCP is connected to the System Peripheral Bus (which is an FPI Bus) and clocked with f_{SYS} as module clock (f_{SYSmax} = 80 MHz, resulting in a minimum clock cycle time of 12.5 ns).*

Table 10-15 Instruction Timing

Instruction	Number of Clock Cycles	Comments	Notes
Control			
NOP	1	–	–
COPY	–	–	1)
EXIT	f = 9, s = 7, m = 6	–	2)
BCOPY	–	–	1)
FPI Access			
ADD.F	8 min.	5 int. + 3 min. for FPI read	3)
SUB.F	8 min.	5 int. + 3 min. for FPI read	3)
COMP.F	8 min.	5 int. + 3 min. for FPI read	3)
AND.F	8 min.	5 int. + 3 min. for FPI read	3)

Table 10-15 Instruction Timing (cont'd)

Instruction	Number of Clock Cycles	Comments	Notes
OR.F	8 min.	5 int. + 3 min. for FPI read	3)
XOR.F	8 min.	5 int. + 3 min. for FPI read	3)
LD.F	8 min.	5 int. + 3 min. for FPI read	3)
ST.F	5 min.	2 int. + 3 min. for FPI write	3)
XCH.F	8 min.	4 int. + 4 min. for FPI read and write	3)
PRAM Access			
ADD.PI	2	–	–
SUB.PI	2	–	–
COMP.PI	2	–	–
MCLR.PI	4	–	–
AND.PI	2	–	–
MSET.PI	4	–	–
OR.PI	2	–	–
XOR.PI	2	–	–
LD.PI	2	–	–
ST.PI	2	–	–
XCH.PI	2	–	–
Arithmetic (Conditional)			
ADD	1	–	–
SUB	1	–	–
COMP	1	–	–
NEG	1	–	–
NOT	1	–	–
AND	1	–	–
OR	1	–	–
XOR	1	–	–
LD.P	2	–	–
ST.P	4	–	–
MOV	1	–	–
INB	1	–	–

Table 10-15 Instruction Timing (cont'd)

Instruction	Number of Clock Cycles	Comments	Notes
PRI	1	–	–
Immediate Access			
ADD.I	1	–	–
SUB.I	1	–	–
COMP.I	1	–	–
SHR	1	–	–
SHL	1	–	–
RR	1	–	–
RL	1	–	–
LDL.IU	1	–	–
LDL.IL	1	–	–
SET	1	–	–
CLR	1	–	–
LD.I	1	–	–
INB.I	1	–	–
CHKB	1	–	–
FPI + Immediate Access			
SET.F	8 min.	4 int. + 4 min. for locked FPI RMW	4)
CLR.F	8 min.	4 int. + 4 min. for locked FPI RMW	4)
LD.IF	8 min.	5 int. + 3 min. for FPI read	3)
ST.IF	5 min.	2 int. + 3 min. for FPI write	5)
Complex Maths			
DINIT	1	–	6)
DSTEP	10	–	6)
MINIT	1	–	7)
MSTEP.L	10	–	7)
MSTEP.U	10	–	7)
Jump			
JL	4	–	–

Table 10-15 Instruction Timing (cont'd)

Instruction	Number of Clock Cycles	Comments	Notes
JC	y = 4, n = 2	–	8)
JC.A	y = 4, n = 2	–	8)
JC.I	y = 4, n = 2	–	8)
JC.IA	y = 4, n = 2	–	8)
DEBUG	sdb - 0 = 2 sdb - 1 = exit_time	–	9)

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)
4) Cycles = 4 internal + 4 minimum for FPI read/modify/write (with 0 wait cycles + 1 cycle bus arbitration)
5) Cycles = 2 internal + 3 minimum for FPI write (with 0 wait cycles + 1 cycle for bus arbitration)
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 × 10 = 41 cycles
8) y = jump taken, n = jump not taken
9) 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

10.12 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.

10.12.1 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.

10.12.1.1 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.

10.12.1.2 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.IFbase #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.

10.12.2 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 10-118**).

10.12.3 Unused Registers as Globals 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.

10.12.4 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.

10.12.5 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
```

10.12.6 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
```

10.12.7 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.

10.12.7.1 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.

10.12.7.2 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 10-124** for TC1766 specific details of the BCOPY instruction.

10.13 PCP Programming Notes and Tips

This section discusses constraints on the use of the PCP and points out some non-obvious issues.

10.13.1 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).

10.13.2 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 10-9**.
- 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 10-114**.
- 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 10-120**).

10.13.3 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.

10.13.3.1 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
```

10.13.3.2 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.

10.13.4 Implementing Divide Algorithms

As discussed in **Section 10.11.4**, 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 10-88** only holds true under this condition. If the restrictions on the use of the DSTEP instruction (specified on **Page 10-79**) 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).

10.13.5 Implementing Multiply Algorithms

As discussed in **Section 10.11.4**, 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.

10.14 Implementation of the PCP in the TC1766

The addresses of the PCP registers and memories in the TC1766 are given in the following subsections:

10.14.1 PCP Memories

In the TC1766, the location of the registers and the memories sizes of the PRAM and the CMEM are given in **Table 10-16**.

Table 10-16 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 10-52**		256 byte
	Reserved	F004 4000$_H$ - F004 FFFF$_H$	BE	BE	–
	PCP Data Memory (PRAM)	F005 0000$_H$ - F005 1FFF$_H$	nE, 32	nE, 32	8 Kbyte
	Reserved	F005 2000$_H$ - F005 FFFF$_H$	BE	BE	–
	PCP Code Memory (CMEM)	F006 0000$_H$ - F006 2FFF$_H$	nE, 32	nE, 32	12 Kbyte

Note: "BE" means that in case of an access to this address region a bus error is generated.

10.14.2 Memory Error Status Flag

In the TC1766, bit PCP_ES.ME (see **Page 10-60**) is not connected. Therefore, PCP_ES.ME is always read as 0.

10.14.3 BCOPY Instruction

In the TC1766, 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).

10.14.4 PCP Reset Operation

The PCP module can be reset by two ways:

- By a system hardware signal (hard reset)
- By programming of a PCP register bit (soft reset)

PCP Hard Reset

A PCP hard reset is always triggered if at least one of these TC1766 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

PCP Soft Reset

A PCP soft reset is generated by writing a 1 to bit PCP_CS.RST (see also **Page 10-56**). A PCP soft reset differs from the PCP hard reset in that the interrupt system is not reset. Specifically, all PCP interrupt nodes with their service request control registers are not completely reset, and the PICU is also not affected.

The PCP soft reset is adopted to prevent loss of synchronization between any interrupt node and the Arbitration Unit (ICU) to which it is connected due to one, but not the other unit is reset. The effect of PCP soft reset is to:

- Halt any operating channel
- Reset all control registers (except PCP_SRCx registers) to their reset values
- PCP_SRCx registers: clear all SRR and RRQ bits and leave all other bits unchanged
- Reset the PCP Processor Core to its default state
- Reset the FPI Bus interface
- Clear all SRR and RRQ bits in all PCP_SRCx registers (all other bits are unchanged)

Note: The PCP soft reset is a useful hardware support feature in a PCP debugging environment. It has been included for backward compatibility with PCP version 1. However, generation of a PCP soft reset may be critical because it may cause an FPI Bus error as the PCP's interface to the FPI Bus becomes reset.
Therefore, it is generally not recommended to use the PCP soft reset capability in a PCP application program. A PCP hard reset should be generated instead.

11 Direct Memory Access Controller

This chapter describes the Direct Memory Access (DMA) Controller and the Memory Checker Module (MCHK) of the TC1766. It contains the following sections:

- Functional description of the DMA controller kernel (see **Section 11.1**)
- DMA controller kernel register description (see **Section 11.2**)
- TC1766 implementation-specific details of the DMA controller (interrupt control, address decoding, clock control, see **Section 11.3**)
- Functional description of the MCHK module (see **Section 11.4**)

*Note: The DMA kernel register names described in **Section 11.2** are referenced in the TC1766 User's Manual by the module name prefix "DMA_".*

11.1 DMA Controller Kernel Description

The DMA Controller of the TC1766 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-Block to the two FPI Bus interfaces and an MLI bus interface. In the TC1766, the FPI Bus interfaces are connected to the System Peripheral Bus and the DMA Bus. The third specific bus interface provides a connection to Micro Link Interface modules (two MLI modules in the TC1766) and other DMA-related devices (Memory Checker module in the TC1766). Clock control, address decoding, DMA request wiring, and DMA interrupt service request control are implementation-specific and managed outside the DMA controller kernel.

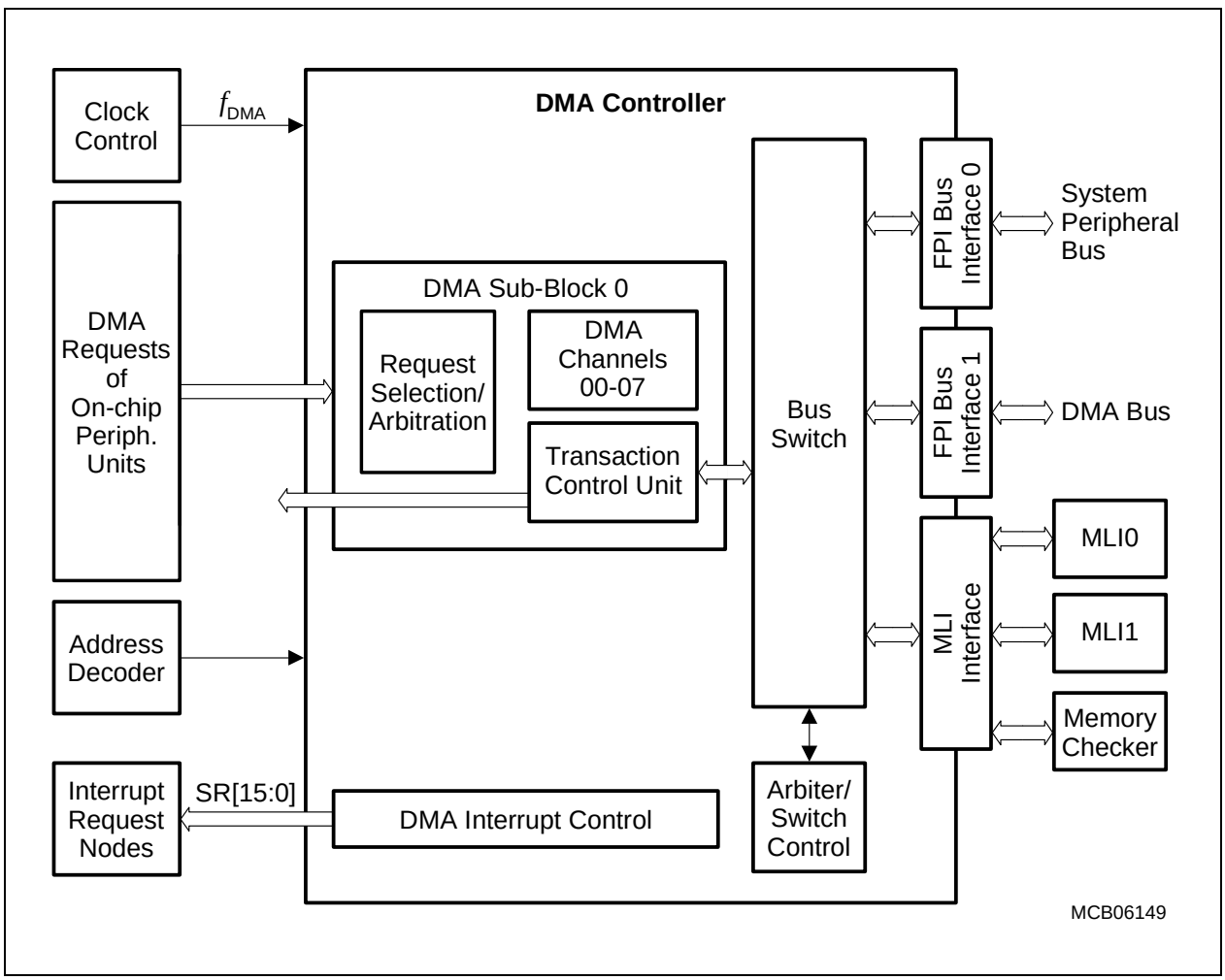


Figure 11-1 DMA Block Diagram

11.1.1 Features

The DMA controller has the following features:

- 8 independent DMA channels
  - 8 DMA channels in the DMA Sub-Block
  - Up to 8 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
- 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
- Full 32-bit addressing capability of each DMA channel
  - 4 Gbyte address range
  - Support of circular buffer addressing mode
- Programmable data width of DMA transfer/transaction: 8-bit, 16-bit, or 32-bit
- Micro Link bus interface support
- 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 module)
- All buses connected to the DMA module must work at the same frequency
- Read/write requests of the System Bus side to the peripherals on DMA Bus are bridged to the DMA Bus (only the DMA is the master on the DMA bus), allowing easy access to these peripherals by PCP and CPU

11.1.2 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.

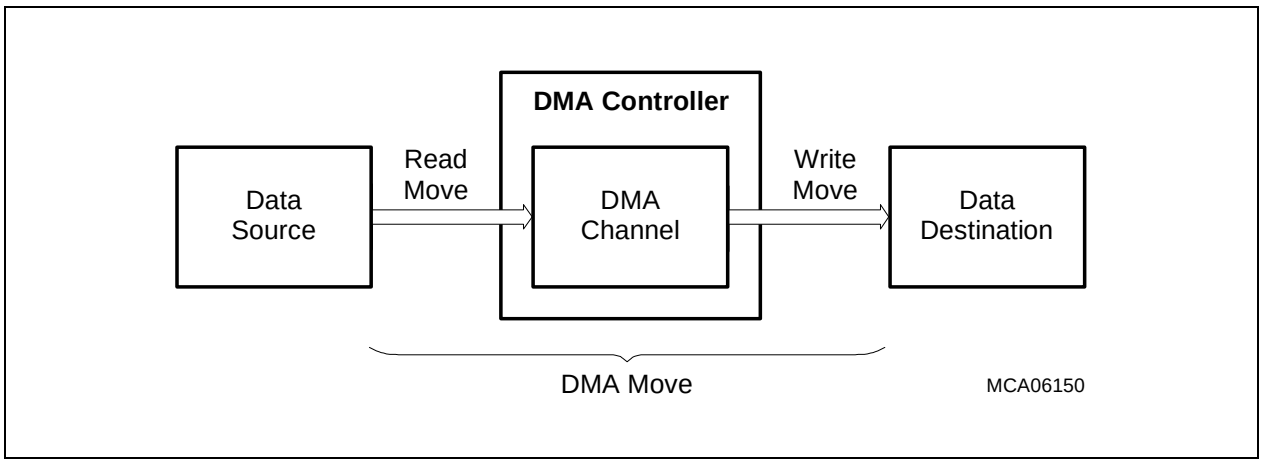


Figure 11-2 DMA Definition of Terms

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.

11.1.3 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 occurrence of a receive or transmit data interrupts in a peripheral module are able to generate a DMA request in parallel to an interrupt request. Therefore, the interrupt control unit and the DMA controller can react independently to interrupt and DMA requests that have been generated by one source.

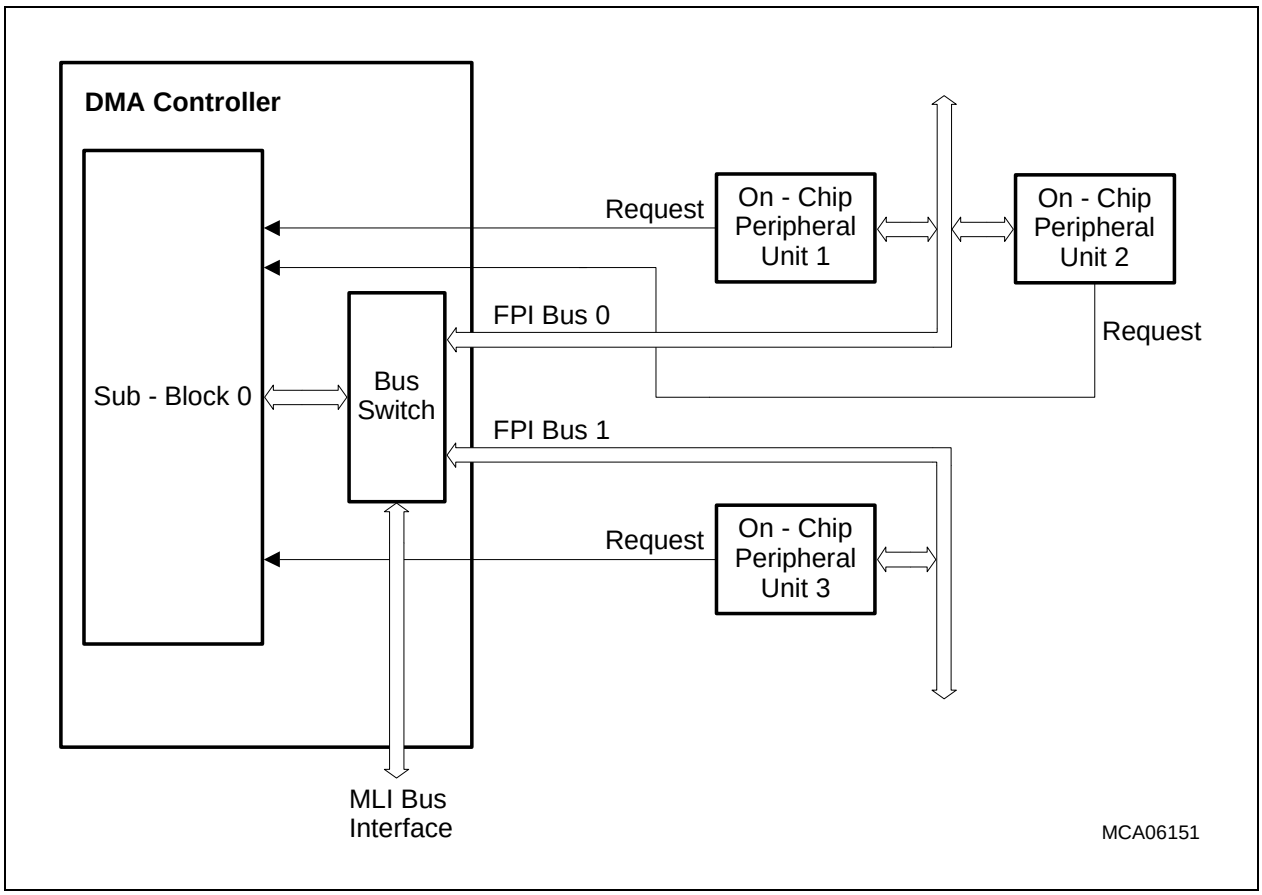


Figure 11-3 DMA Principle

The DMA controller mainly consists of one DMA Sub-Block and a Bus Switch. Once configured, the DMA sub-block is able to act as a master and on the FPI Bus.

11.1.4 DMA Channel Functionality

Each of the 8 DMA channels has one associated register set containing seven 32-bit registers. These registers are numbered by one index to indicate the related DMA channel: 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 0n Control Register CHCR0n (for details, see **Page 11-69**)
- Channel 0n Status Register CHSR0n (for details, see **Page 11-73**)
- Channel 0n Interrupt Control Register CHICR0n (for details, see **Page 11-74**)
- Channel 0n Address Control Register ADRCR0n (for details, see **Page 11-76**)
- Channel 0n Source Address Register SADR0n (for details, see **Page 11-80**)
- Channel 0n Destination Address Register DADR0n (for details, see **Page 11-81**)
- Channel 0n Shadow Address Register SHADR0n (for details, see **Page 11-82**)

11.1.4.1 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 SHADR0n holds the new address value that was written. For this operation, bit field ADRCR0n.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 SHADR0n is written either into SADR0n or DADR0n (ADRCR0n.SHCT must be set accordingly). After the shadow transfer, SHADR0n is set to 0000 0000_H. Therefore, 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 (ADRCR0n.SHCT = 00_B), SHADR0n is always read as 0000 0000_H.

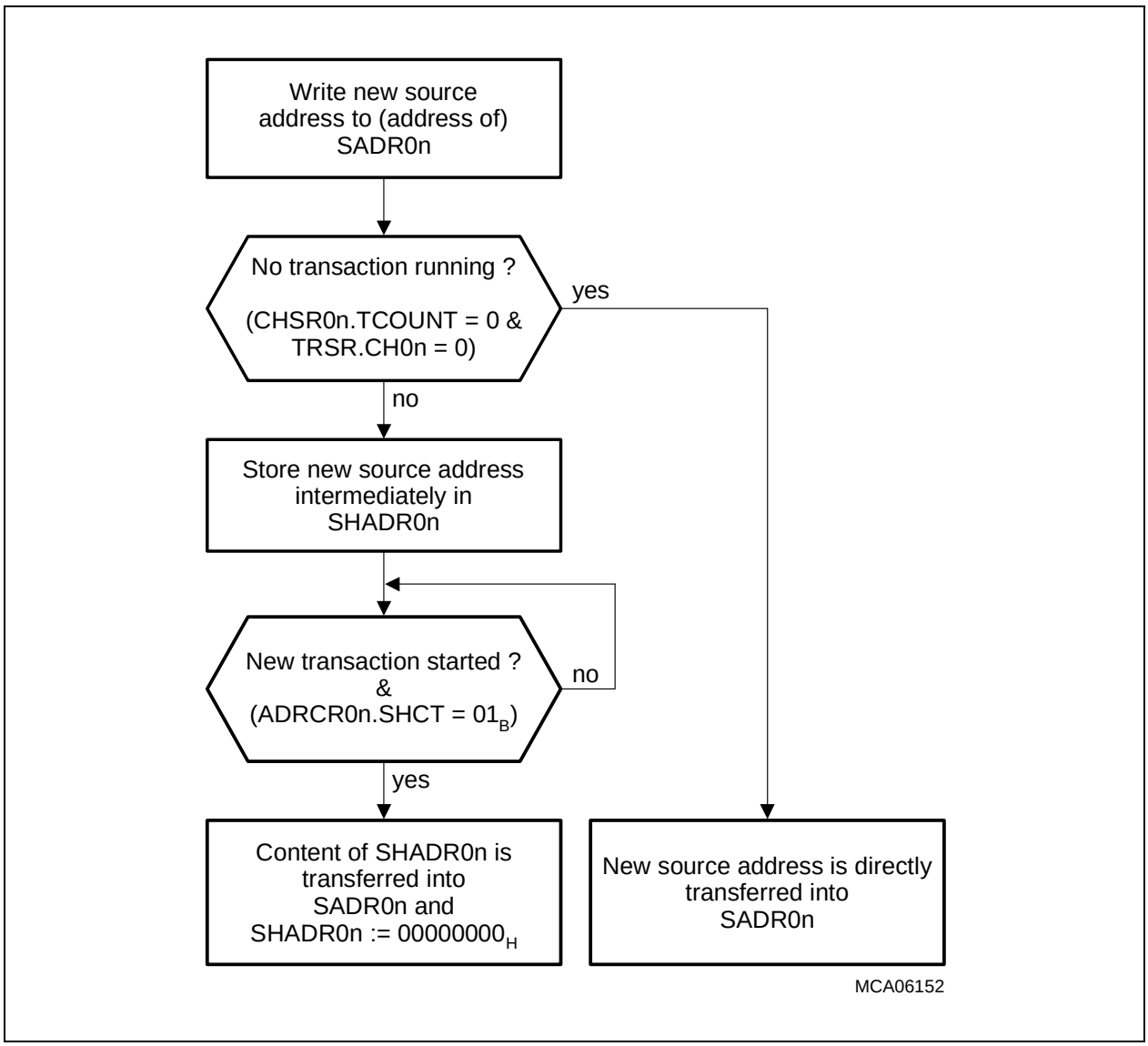


Figure 11-4 Source Address Update

The transfer count of a DMA transaction, stored in bit field CHCR0n.TREL, can also be programmed if the DMA transaction is running. At the start of a DMA transaction, TREL is transferred to bit field CHSR0n.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.

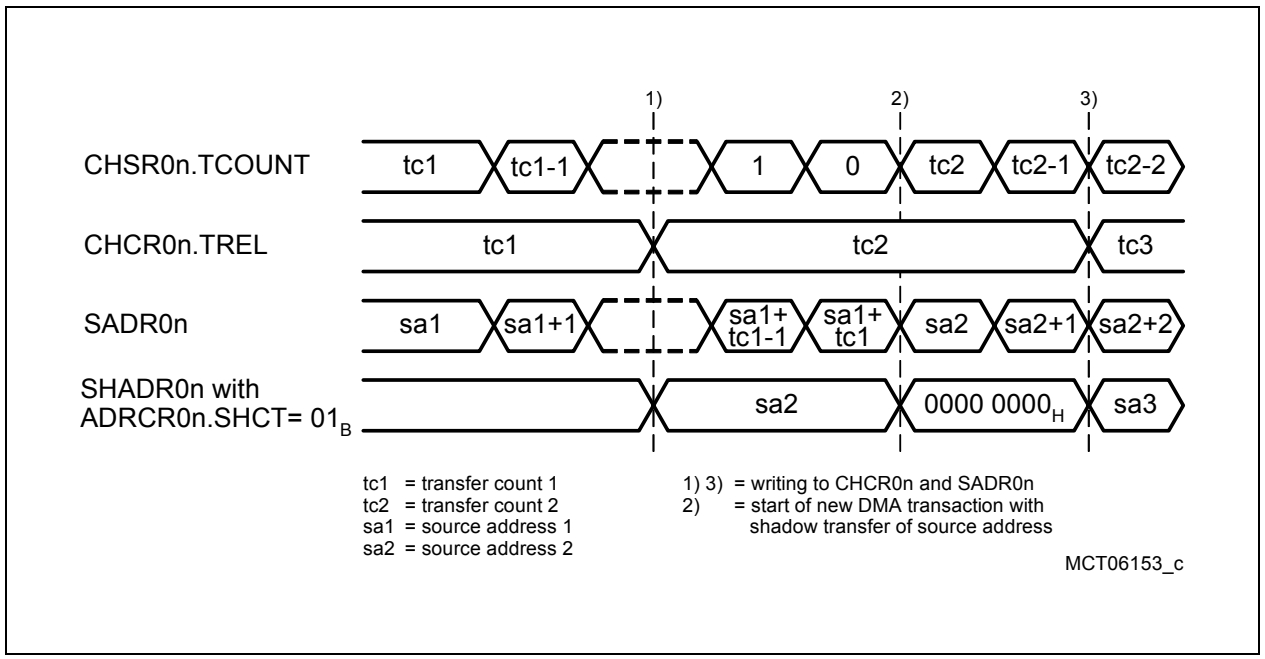


Figure 11-5 Shadow Source Address and Transfer Count Update

Figure 11-5 shows how the contents of the source address register SADR0n and the transfer count CHSR0n.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 SADR0n and transferred to SADR0n when the new DMA transaction is started at 2). At this time, transfer count tc2 is also transferred to CHSR0n.TCOUNT.

11.1.4.2 DMA Channel Request Control

Figure 11-6 shows the control logic for DMA requests that is implemented for each DMA channel.

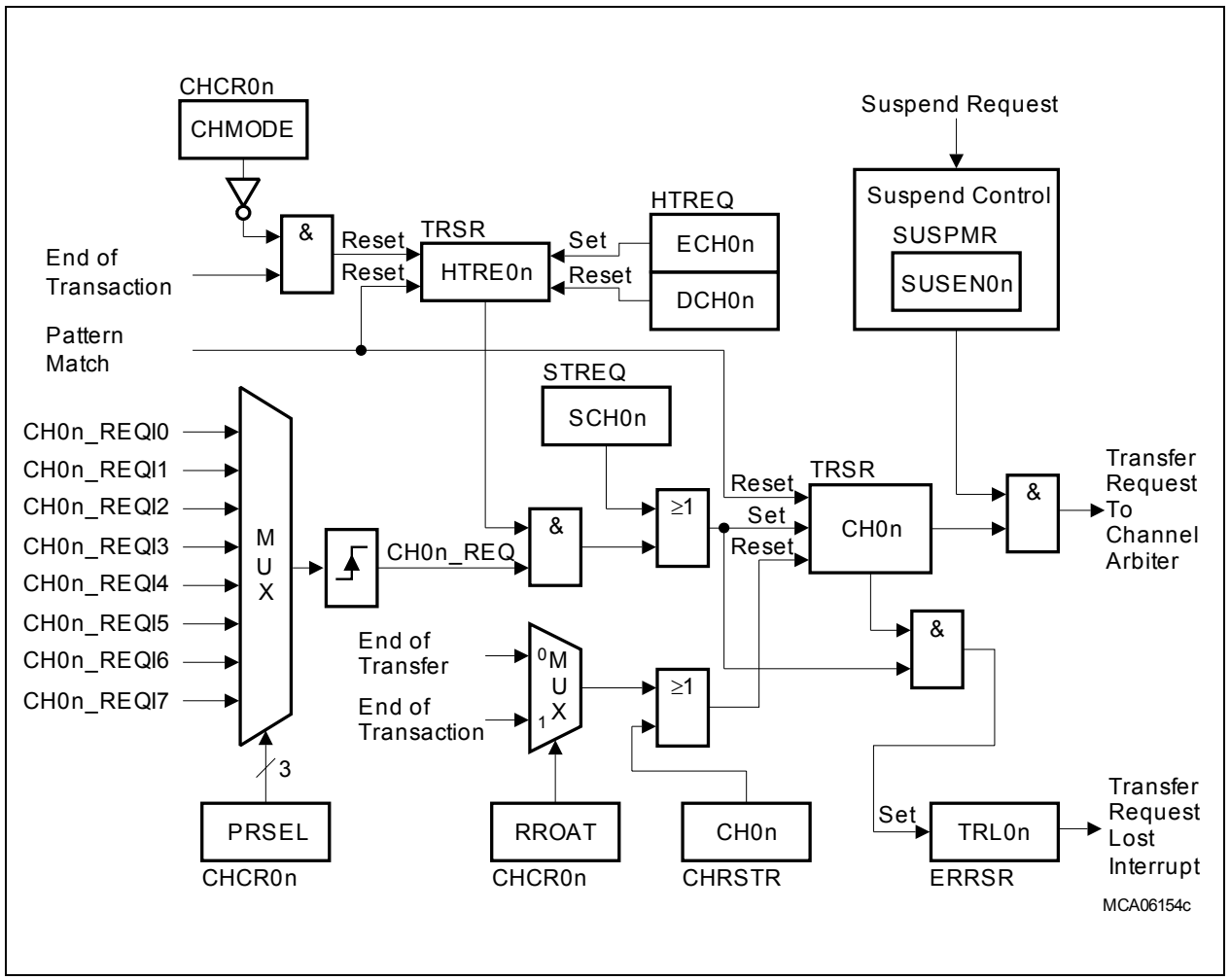


Figure 11-6 Channel Request Control

Two different types of DMA requests are possible:

- Hardware DMA requests
- Software DMA requests

The hardware request CH0n_REQ can be connected to one of eight possible hardware request input lines as selected by bit field CHCR0n.PRSEL. Hardware requests are enabled/disabled by status bit TRSR.HTRE0n. HTRE0n can be set/cleared 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.SCH0n.

Status flag TRSR.CH0n indicates whether or not a software or hardware generated DMA request for DMA channel 0n is pending. TRSR.CH0n can be cleared 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 0n while TRSR.CH0n 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.TRL0n will be set and a transfer lost interrupt can be generated.

11.1.4.3 DMA Channel Operation Modes

The operation mode of a DMA channel is individually programmable for each DMA channel 0n. 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 0n becomes disabled by hardware after the last DMA transfer of its DMA transaction. In hardware-controlled Continuous Mode, a DMA channel 0n 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.DCH0n = 1. This forces status flag TRSR.HTRE0n = 0 (hardware request of DMA channel 0n is disabled).

The software-controlled mode that initiates one complete DMA transaction to be executed is selected for DMA channel 0n by the following write operations:

- CHCR0n.RROAT = 1
- STREQ.SCH0n = 1

Setting STREQ.SCH0n to 1 (this is the software request) causes the DMA transaction of DMA channel 0n to be started and TRSR.CH0n to be set. At the start of the DMA transaction, the value of CHCR0n.TREL is loaded into CHSR0n.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 0n becomes disabled and status flag

TRSR.CH0n is cleared. Setting STREQ.SCH0n again starts a new DMA transaction of DMA channel 0n 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 0n by the following write operations:

- CHCR0n.RROAT = 0
- STREQ.SCH0n = 1, repeated for each DMA transfer

When CHCR0n.RROAT = 0, TRSR.CH0n is cleared after each DMA transfer of the DMA transaction and a new software request (writing STREQ.SCH0n = 1) must be generated for starting the next DMA transfer.

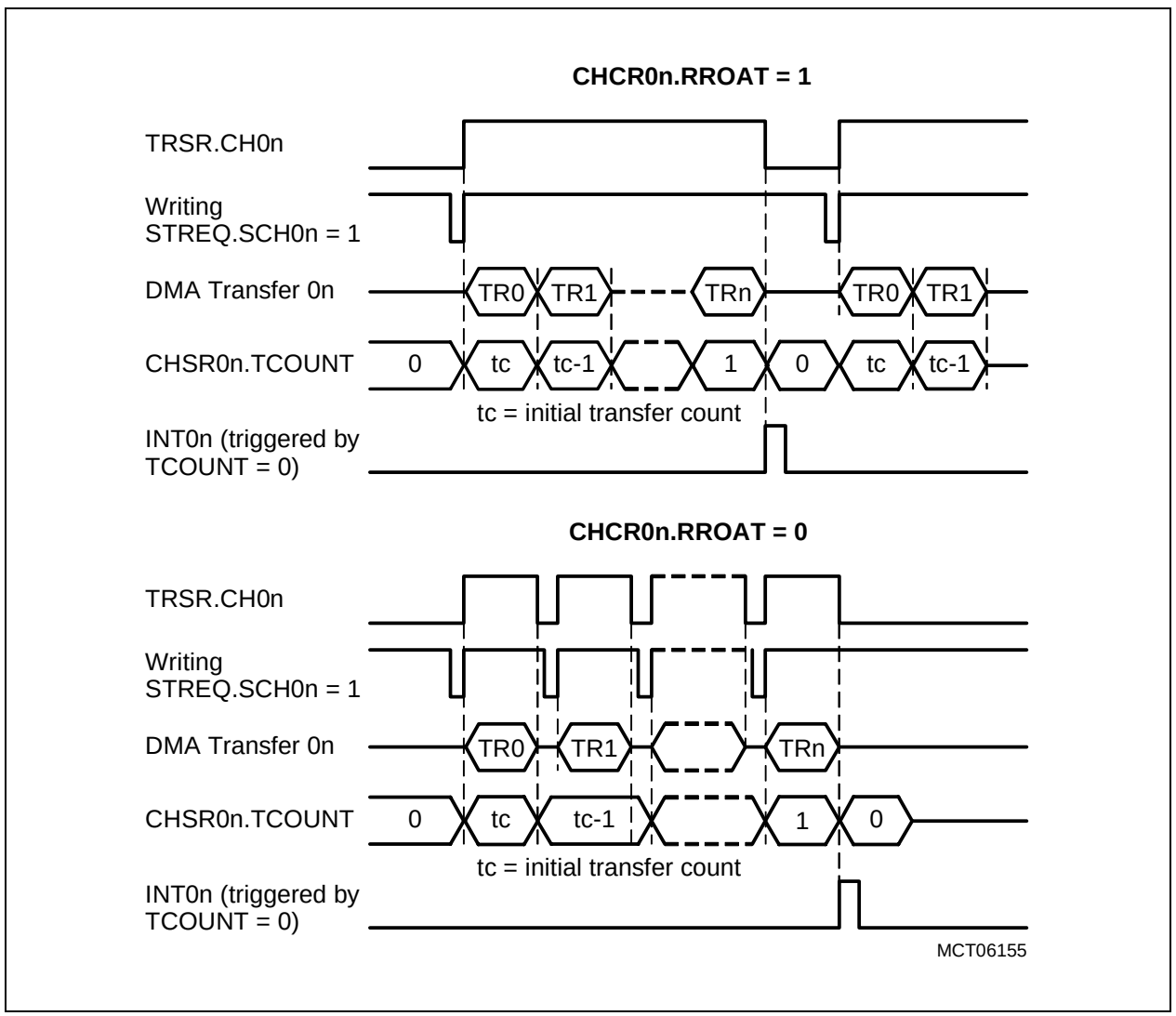


Figure 11-7 Software Controlled Mode Operation

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 0n is selected by the following operations:

- CHCR0n.CHMODE = 0
- CHCR0n.RROAT = 1
- Selecting one of the eight hardware request inputs via CHCR0n.PRSEL
- HTREQ.ECH0n = 1

Setting HTREQ.ECH0n to 1 causes the hardware request CH0n_REQ of channel 0n to be enabled (TRSR.HTRE0n = 1). Whenever the hardware request CH0n_REQ becomes active, the value of CHCR0n.TREL is loaded into CHSR0n.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.CH0n and TRSR.HTRE0n are cleared. In order to start a new hardware-controlled DMA transaction, hardware requests must be enabled again by setting TRSR.HTRE0n through HTREQ.ECH0n = 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:

- CHCR0n.RROAT = 0

In this operation mode, TRSR.CH0n becomes cleared after each DMA transfer of the DMA transaction, and a new hardware request at CH0n_REQ must be generated for starting the next DMA transfer.

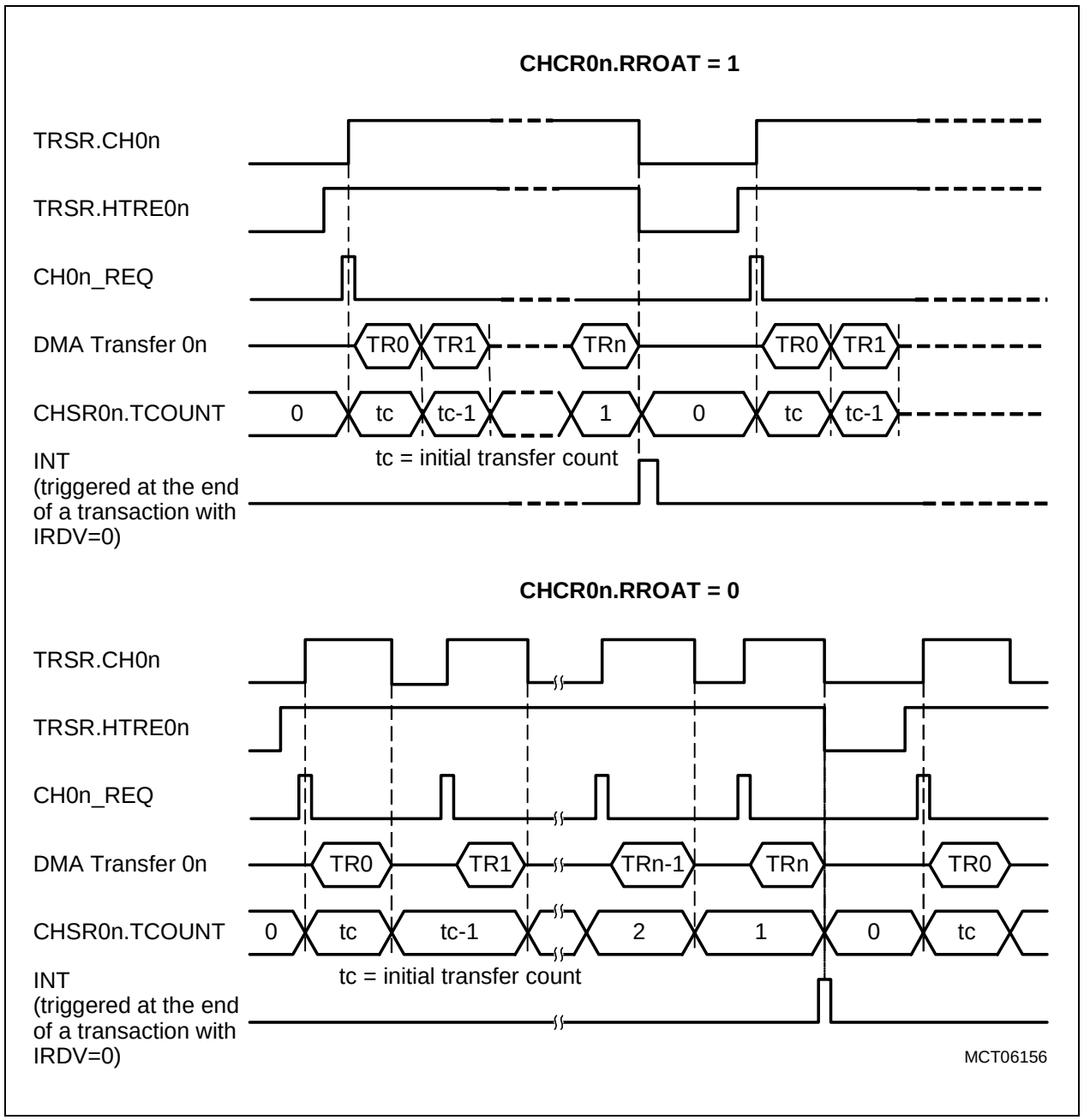


Figure 11-8 Hardware-controlled Single Mode Operation

Hardware-controlled Continuous Mode

In hardware-controlled Continuous Mode (CHCR0n.CHMODE = 1), the hardware transaction request enable bit HTRE0n is not cleared at the end of a DMA transaction. A new transaction of DMA channel 0n with the parameters actually stored in the channel register set of DMA channel 0n is started each time when CHSR0n.TCOUNT reaches 000_H. No software re-enable for a hardware request at CH0n_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.SCH0n. Hardware requests are still disabled. After hardware requests have been enabled by setting HTREQ.ECH0n, subsequent DMA transfers are triggered now by hardware request coming from the CH0n_REQ line.

In the example, DMA channel 0n operates in Single Mode (CHCR0n.CHMODE = 0). In this mode, TRSR.HTRE0n is cleared by hardware when CHSR0n.TCOUNT reaches 0 at the end of the DMA transaction.

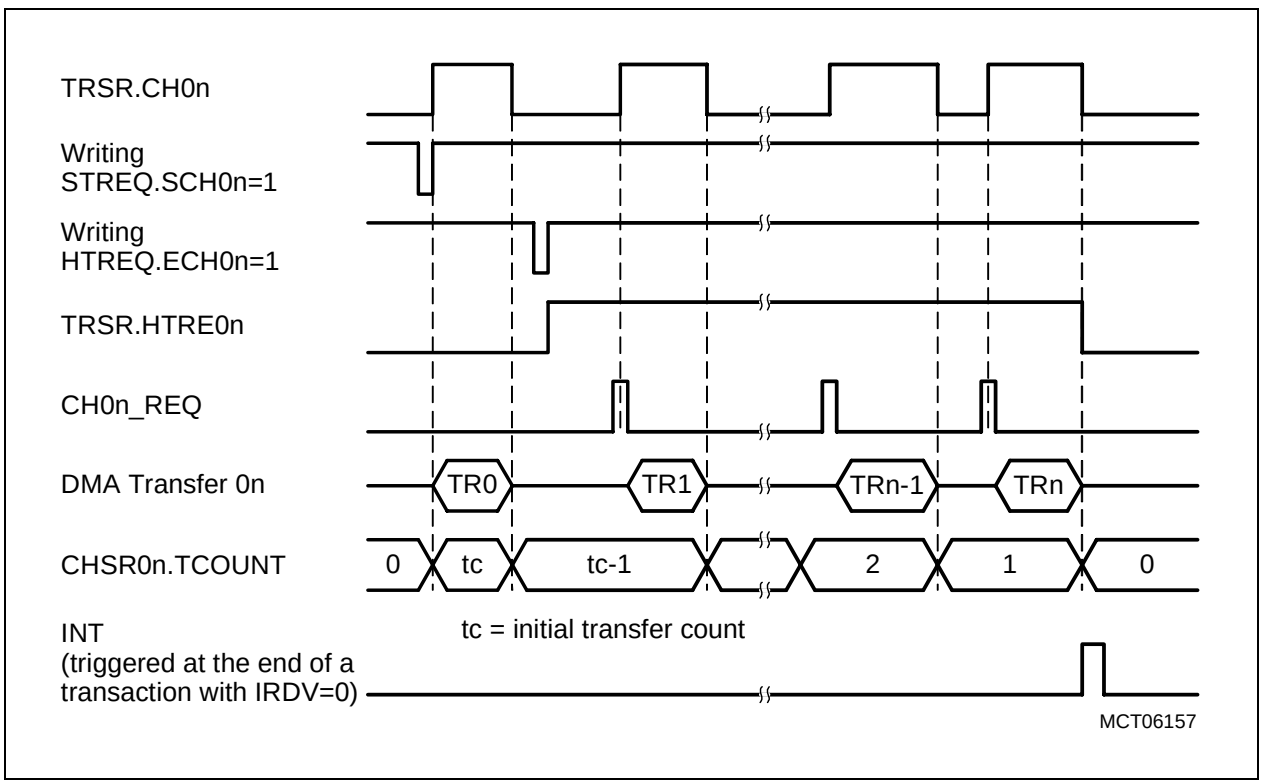


Figure 11-9 Transaction Start by Software, Continuation by Hardware

11.1.4.4 Error Conditions

The source move error flag ERRSR.FPI0SER indicates an FPI Bus error on bus 0 (SPB) that occurred during a source move (read) of a DMA transaction. The destination move error flag ERRSR.FPI0DER indicates an FPI Bus error on bus 0 (SPB) that occurred during a destination move (write) of a DMA transaction.
The transaction lost error flag ERRSR.TRL0n indicates if a DMA request for a DMA channel 0n has been lost.

In the case of a read error, the write action is not executed, but the destination address is updated.

11.1.4.5 Channel Reset Operation

A DMA transaction of DMA channel 0n can be stopped (channel is reset) by setting bit CHRSTR.CH0n. When a read or write FPI Bus access of DMA channel 0n is executed at the time when CHRSTR.CH0n is set, this FPI Bus access is finished normally. This behavior guarantees data consistency.

When CHRST.CH0n is set to 1:

- Bits TRSR.HTRE0n, TRSR.CH0n, ERRSR.TRL0n, INTSR.ICH0n, INTSR.IPM0n, WRPSR.WRPD0n, WRPSR.WRPS0n, CHSR0n.LXO, and bit field CHSR0n.TCOUNT are cleared.
- Source and destination address register will be set to the wrap boundary. SHADR0n will be cleared.
- All automatic functions are stopped for channel 0n.

A user program must execute the following steps for resetting a DMA channel:

1. If hardware requests are enabled for the DMA channel 0n, disable the DMA channel 0n hardware requests by setting HTREQ.ECH0n = 0.
2. Writing a 1 to CHRST.CH0n.
3. Waiting (polling) until CHRST.CH0n = 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 0n by setting HTREQ.ECH0n = 1 for hardware requests or STREQ.SCH0n = 1 for software requests.

The value of CHCR0n.TREL is copied to CHSR0n.TCOUNT when a new DMA transaction is requested and shadow address register contents is not equal 00000000_H.

11.1.4.6 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 (CHCR0n.BLKM).

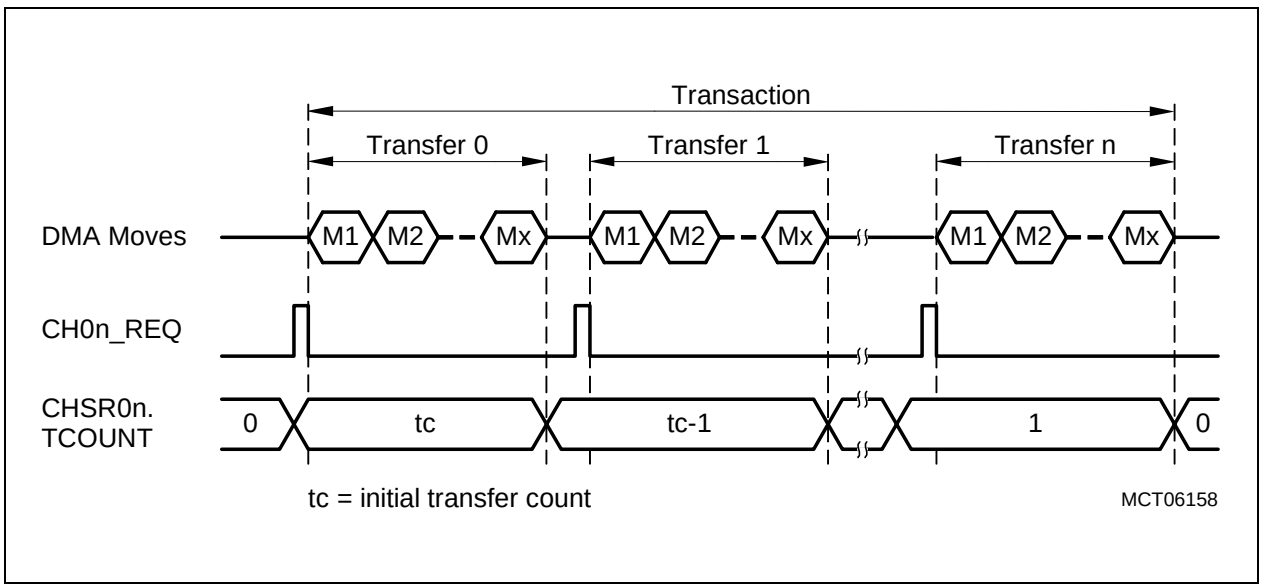


Figure 11-10 Transfer and Move Count

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 ADRCR0n determine how the addresses are incremented/decremented. Further, the data width as defined in CHCR0n.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 (CHCR0n.CHDW = 01_B) is assumed.

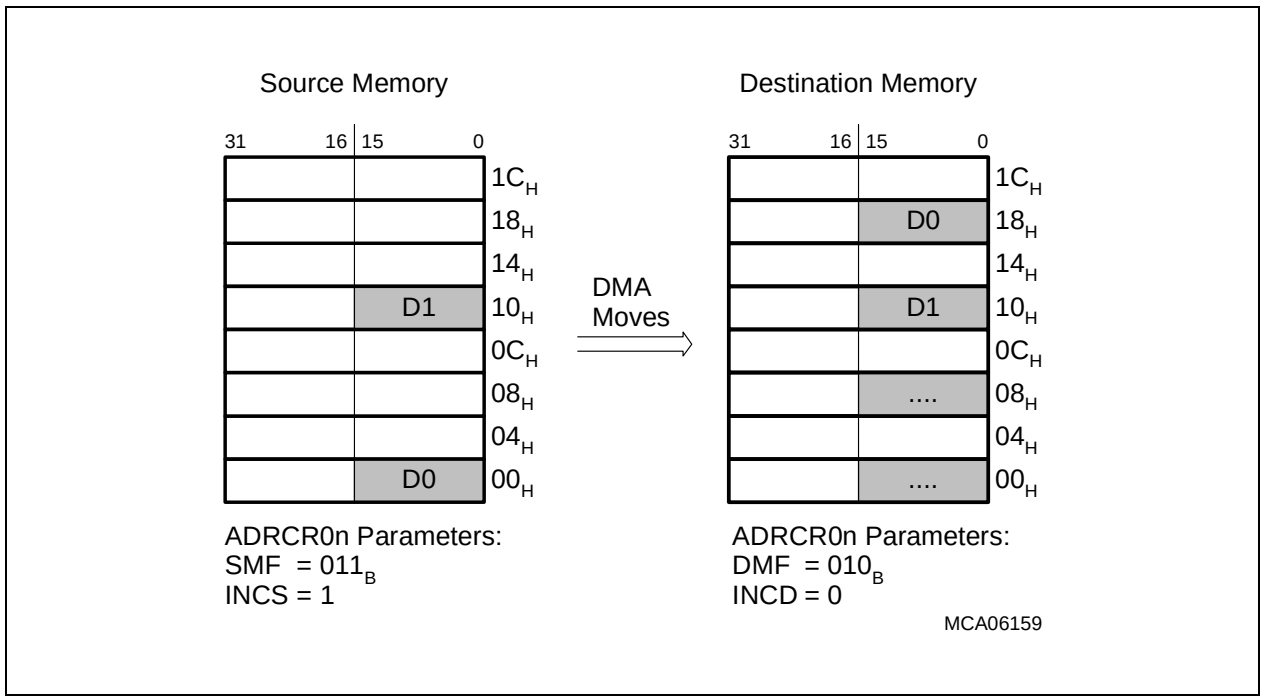


Figure 11-11 Programmable Address Modification - Example 1

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.

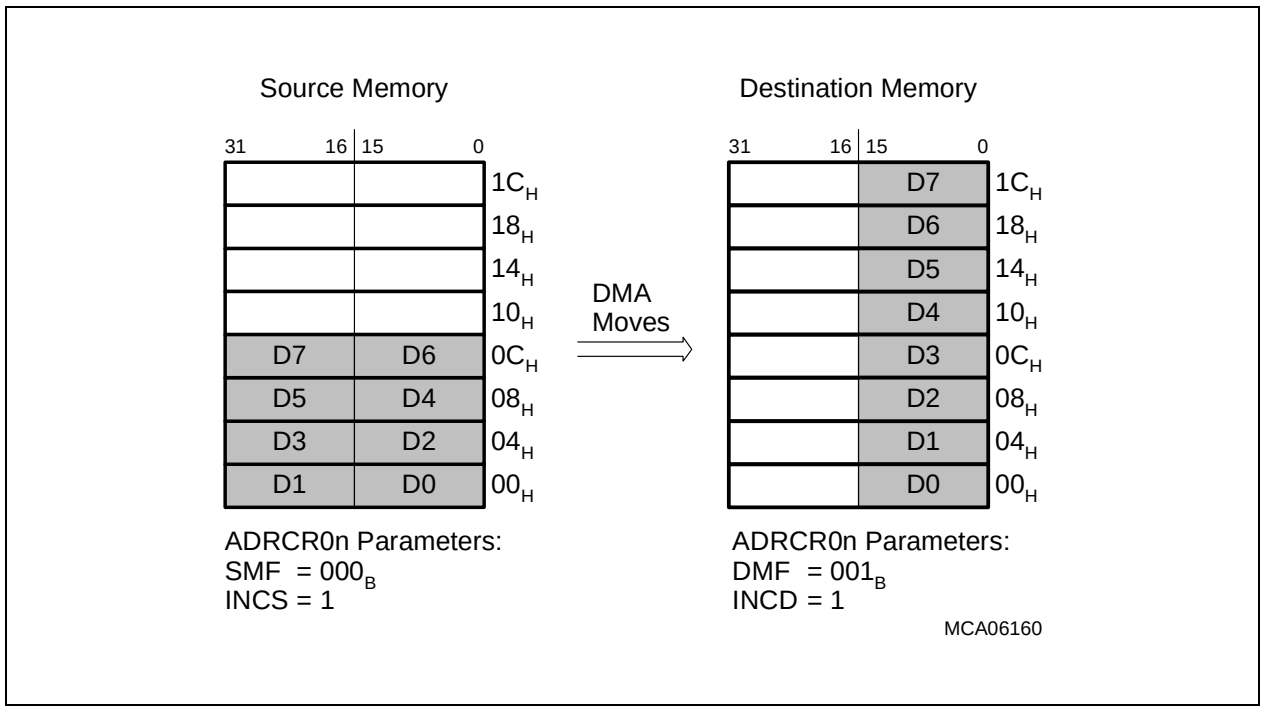


Figure 11-12 Programmable Address Modification - Example 2

11.1.4.7 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 ADRCR0n.CBLS (for the source buffer) and ADRCR0n.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 11-8**), the same circular buffer address will always be accessed.

11.1.5 Transaction Control Engine

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 CHCR0n.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.

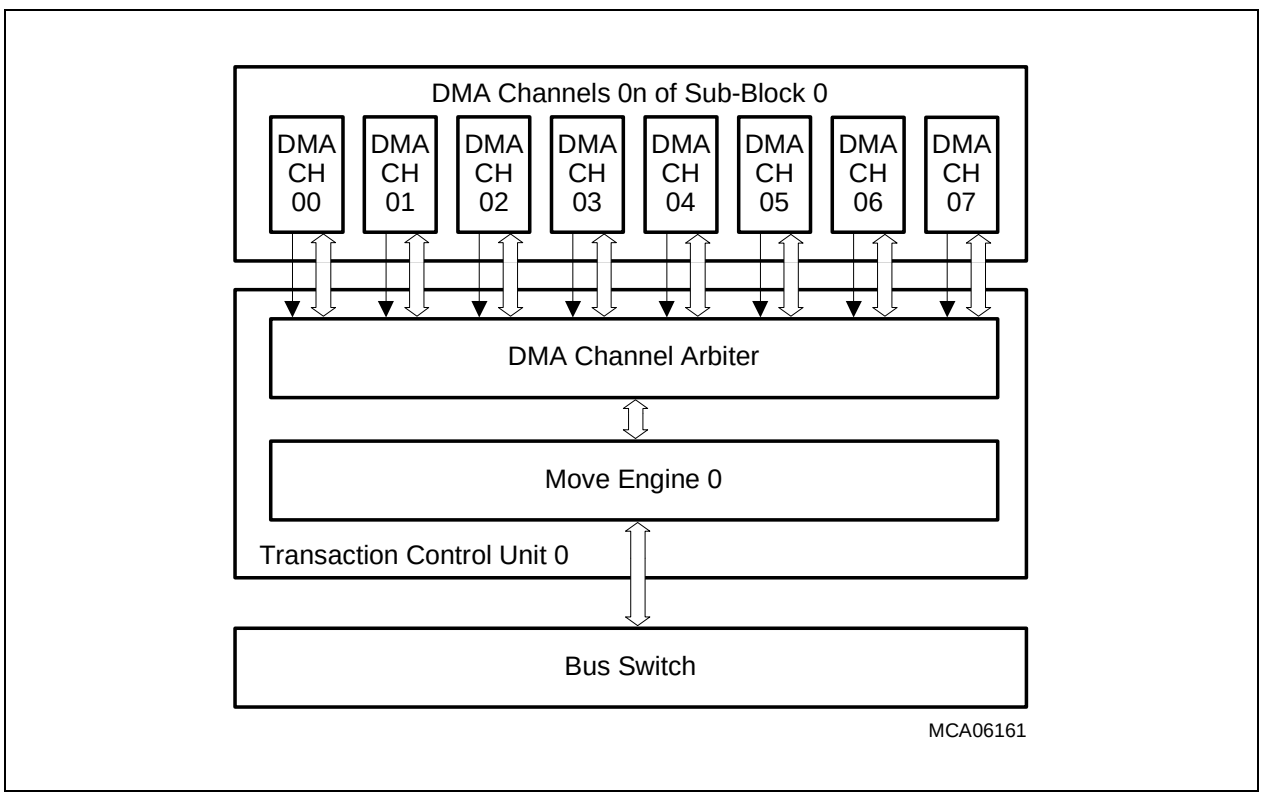


Figure 11-13 Transaction Control Engine

11.1.6 Bus Switch

The Bus Switch of the DMA controller provides the connection from the DMA Sub-Block to the two FPI Bus interfaces (connected to System Peripheral Bus and DMA Bus) and the MLI bus interface (see **Figure 11-14**).

A very simple bus bridge is implemented in the Bus Switch. It behaves as a feed-through for the slave interface on the FPI Bus interface 0 to the other bus interfaces (FPI Bus interface 1, MLI interface) because the working frequencies are identical on all these buses. The bridge is the only slave interface in the DMA because all the other interfaces are master on their buses. The slave interface also provides the access to the DMA and MLI registers.

The bridge functionality is covered independently from the DMA Move Engine. An access requesting from the FPI Bus interface 0 (SPB) to locations on the FPI Bus interface 1 (DMA) has priority over DMA actions. The DMA Move Engine is not involved in bridge transfers.

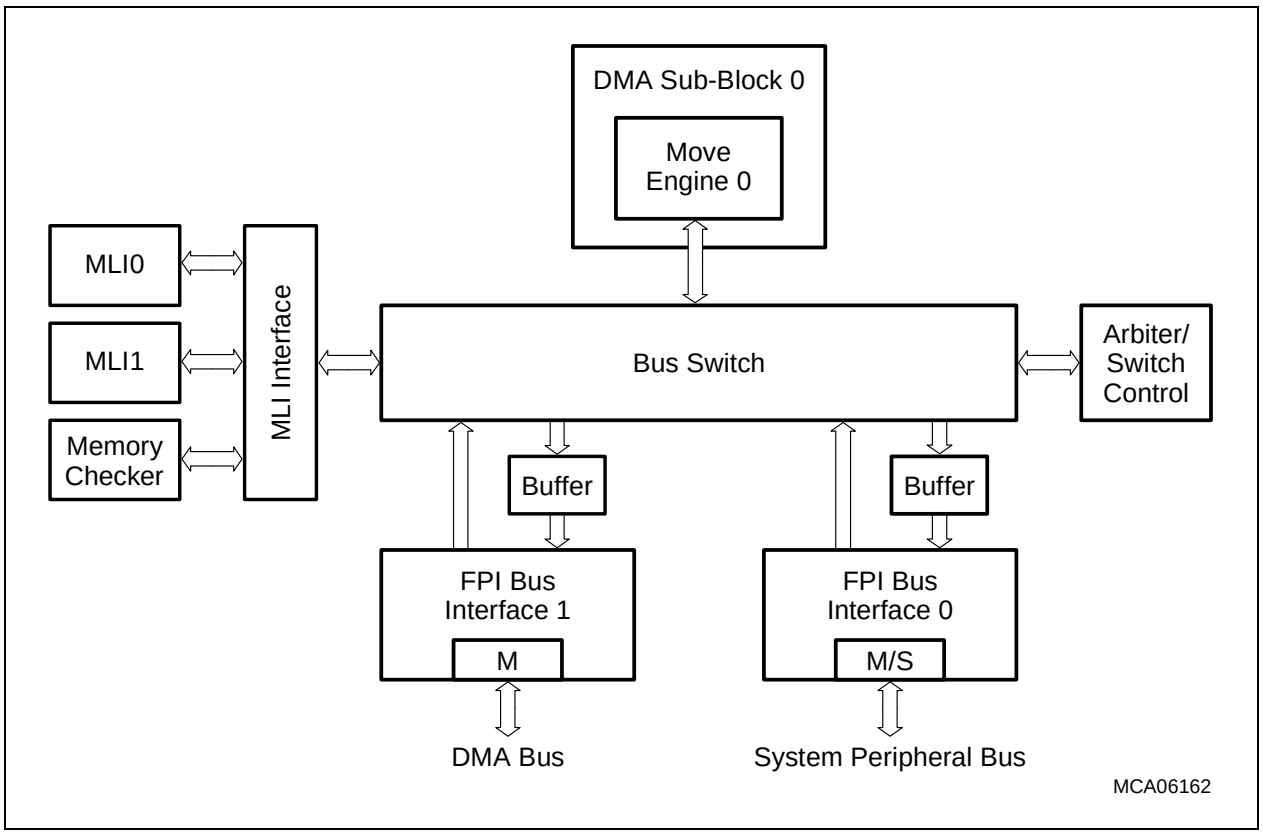


Figure 11-14 Bus Switch

One access can be buffered in the bus interfaces.

Note: The accesses of the DMA Move Engine's bus interfaces to FPI Bus interface 0 and FPI Bus interface 1 are always done in supervisor mode.

The arbiter/switch control unit arbitrates the bus requests for the switch and grants the buses connected to the switch for data transfers. **Table 11-1** defines the Bus Switch priorities.

Table 11-1 Bus Switch Priorities

Priority	Agent Requests	Comment
Highest	FPI Bus 0 to FPI Bus 1 FPI Bus 0 to MLI interface	The FPI Bus bridge functionality has higher priority than the DMA bus requests. Reason: minimizing wait states on the FPI Bus 0.
	DMA Move Engine write move	Priority is defined by CHCR0n.DMAPRIO of DMA channel 0n that is requesting the switch.
	DMA Move Engine read move	
Lowest	MLI	–

CHCR0n.DMAPRIO determines the priority that is used when a move operation related to this channel is targeting the FPI Bus 0 (SPB). In the TC1766, the DMA controller has two priorities on FPI Bus 0 (DMA0 and DMA1), where it competes against the other bus masters in the system to access the bus. The DMAPRIO bit field determines which priority is used by a DMA Move Engine to arbitrate the FPI Bus 0 access. DMIPRIO has no effect in the channel prioritization.

Access modes (U, SV) of an access from FPI Bus 0 to FPI Bus 1 are identically transferred to FPI Bus 1. All accesses triggered by the DMA Move Engine or the MLI modules are always done in SV mode.

The DMA bridge functionality from the FPI Bus 0 to FPI Bus 1, to the MLI modules or to the memory checker does not support read/modify/write instructions.

11.1.7 On-Chip Debug Capabilities

The DMA controller in the TC1766 provides some debugging capabilities. These debug features support:

- Hard-suspend Mode of the DMA controller (for test purposes only)
- Soft-suspend Mode of DMA channels
- Break signal generation
- Trace signal generation

In suspend modes, the operations of DMA channels or the complete DMA module are stopped. Under certain condition conditions also a break signal is generated for the on-chip debug support logic. Further, DMA trace information can be output.

11.1.7.1 Hard-suspend Mode

In Hard-suspend Mode, the DMA controller module clock f_{DMA} is switched off and all further module actions are disabled. Module registers cannot be written anymore in Hard-suspend Mode, but combinatorial read operations of the module registers are still possible. Hard-suspend Mode of the DMA controller can be entered either by an active module suspend request signal (generated by hardware) or by writing the clock control register DMA_CLC register with an appropriate value.

When switching the DMA controller module clock f_{DMA} off, the communication of the DMA controller via its Bus Switch is blocked. Hard-suspend Mode can only be left by a reset operation.

Attention: The Hard-suspend Mode is mainly applicable for test purposes only. It can only be exited by a reset operation, and should not be used during normal operation of the DMA controller.

11.1.7.2 Soft-suspend Mode

The TC1766 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 freezed, DMA requests are no longer forwarded, and the state of the DMA channel can be analyzed by reading the register contents.

Soft-suspend Mode of DMA channel 0n is entered if its suspend enable bit SUSEN0n in the Suspend Mode Register SUSPMR is set. When SUSREQ becomes active, the operation of all DMA channels 0n 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.SUSAC0n is set, indicating that DMA channel 0n is in Soft-suspend Mode. DMA channels that are disabled for Suspend Mode (SUSEN0n = 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 0n is left and its normal operation continues if either the SUSREQ signal becomes inactive, or if the enable bit SUSEN0n is cleared by software.

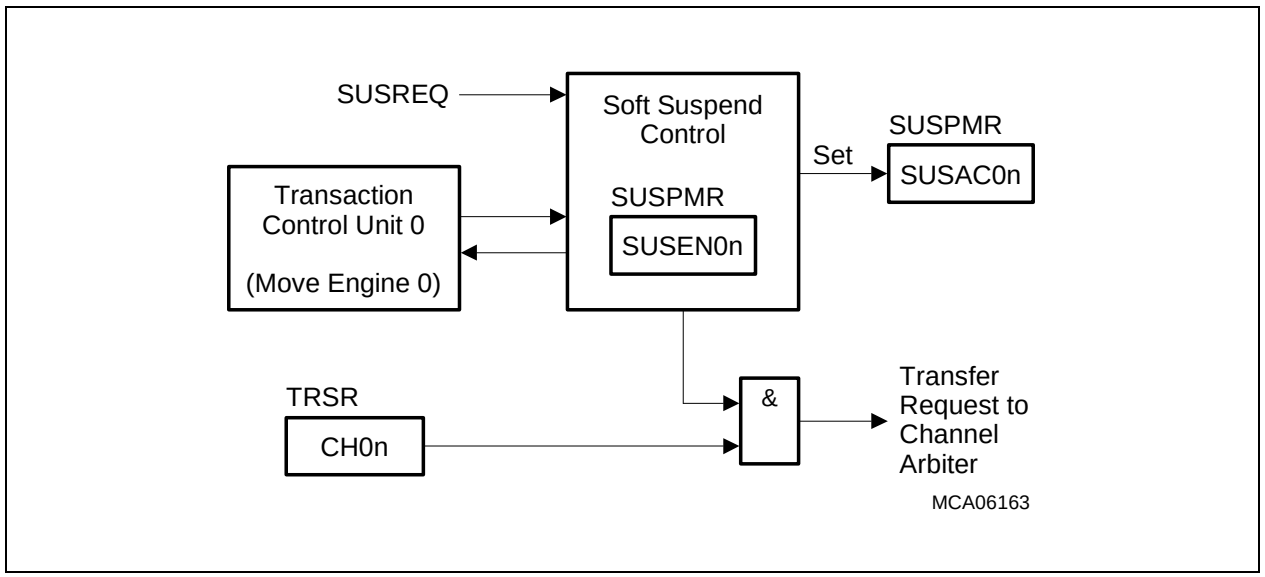


Figure 11-15 Soft-suspend Mode Control

11.1.7.3 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.CH0n

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 0 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 11.1.8.2**.

The second break condition of DMA Sub-Block 0 becomes active when the transaction request bit TRSR.CH0n of one of its eight DMA channels n (as selected by OCDSR.BCHSn) indicates a transition of its state. The CH0n transition type (set, cleared, or set and cleared) is selected by bit field OCDSR.BTCRn.

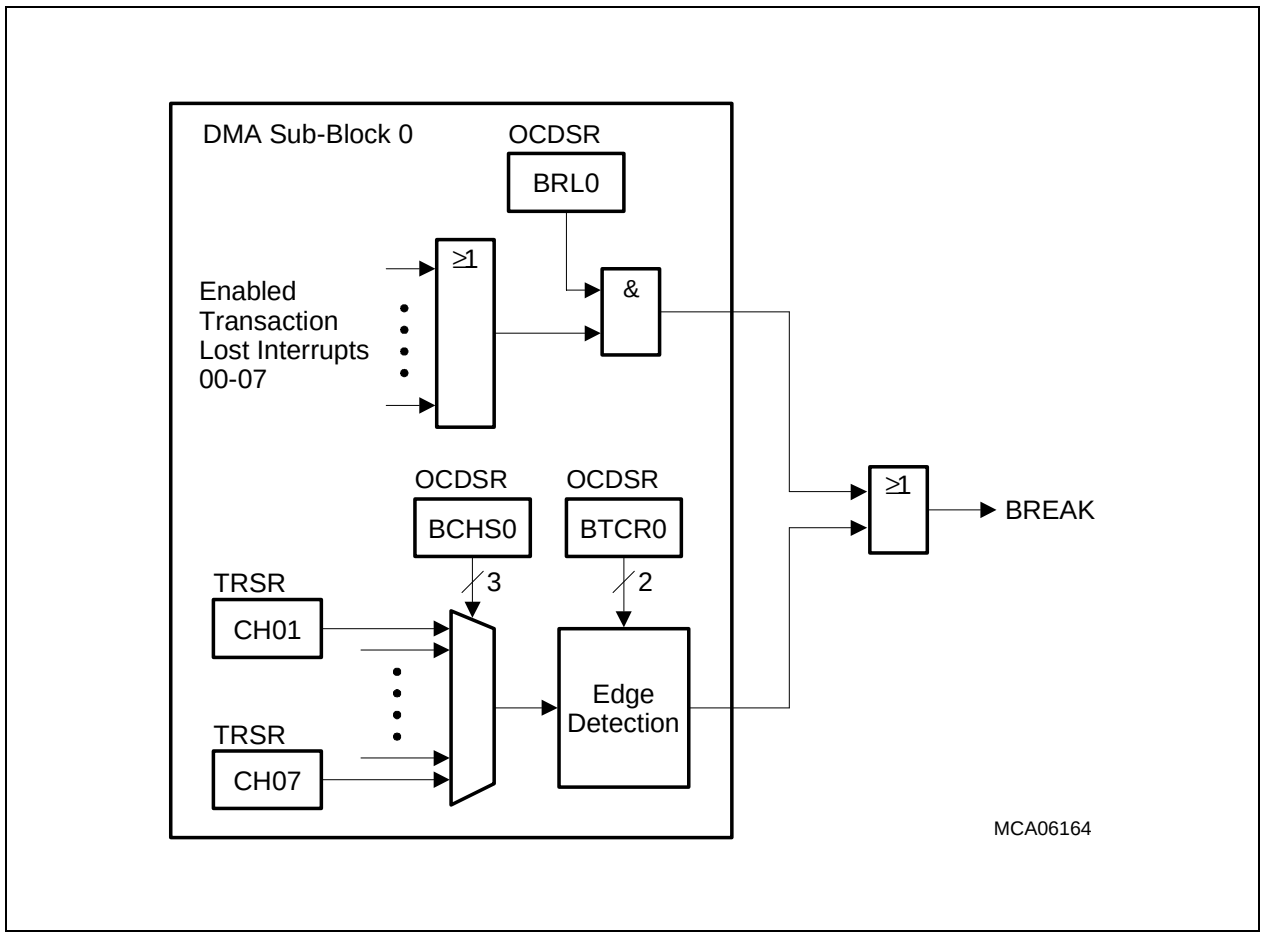


Figure 11-16 DMA Break Event Generation

11.1.7.4 Trace Signal Generation

The TC1766 provides sixteen OCDS Level 2 debug output lines OCDSL2[15:0]. These 16 output lines can be selected to output trace data of the DMA controller. Two trace data types are possible:

- Channel trace for monitoring the transaction request flags CH0n of register DMA_TRSR
- Move Engine trace for monitoring the status flags and bit fields of register DMA_MESR

Channel trace and Move Engine trace outputs are selected by bit TRCDS in the OSCU Configuration and Control Register OCNTRL. The selected DMA trace is further enabled to the trace output lines TR[15:0] by bit OCNTRL.TRCDEN.

The information that is output on the TR[15:0] trace port lines is shown in the following diagram.

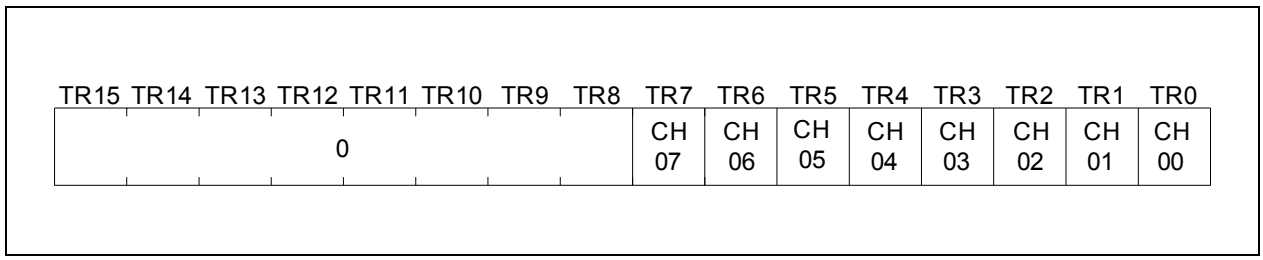


Figure 11-17 16-bit DMA Channel Transaction Request Trace

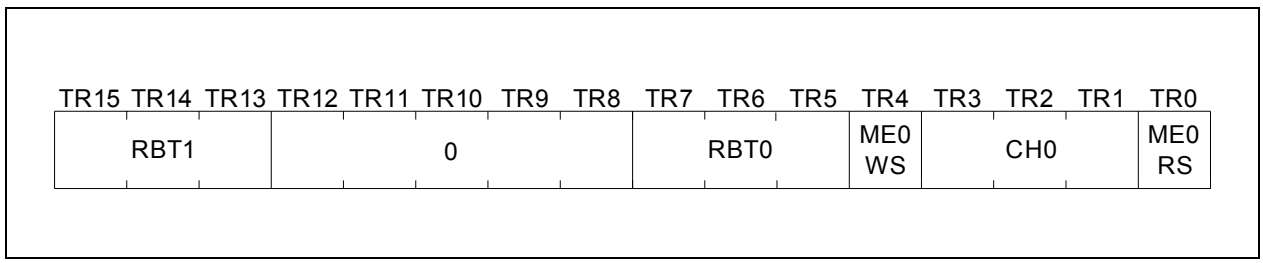


Figure 11-18 16-bit Move Engine Status Information Trace

11.1.8 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 cleared 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.

11.1.8.1 Channel Interrupts

Each DMA channel 0n has one associated channel interrupt. It can always be activated after a DMA transfer, or when CHSR0n.TCOUNT matches with the value of bit field CHSR0n.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 CHICR0n.INTP) are activated when the pattern detection interrupt of DMA channel 0n becomes active (when enabled by CHCR0n.PATSEL not equal 00_B).

A channel interrupt of DMA channel 0n is indicated when status flag INTSR.ICH0n is set. The status flags ICH0n and IPM0n can be cleared together by software when setting bit INTCR.CICH0n (or CHRSTR.CH0n). The channel interrupt of DMA channel 0n is enabled when bit CHICR0n.INTCT[1] is set. The channel interrupt pointer CHICR0n.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 ICH0n flag is available as CH0n_OUT signal at the DMA module boundary.

Bit CHICR0n.INTCT[0] selects these two types of interrupt sources. For the compare operation, bit field IRDV (4-bit) is zero-extended to 9-bit and then compared with the 9-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 IRDV = 0000_B, the match interrupt is generated at the end of a DMA transaction (after the last DMA transfer).

1) In the TC1766, only SR[3:0] are connected to interrupt nodes.

The pattern detection interrupt is indicated when status flag INTSR.IPM0n is set. The status flags IPM0n and ICH0n can be cleared together by software when setting bit INTCR.CICH0n (or CHRSTR.CH0n). The pattern detection interrupt of DMA channel 0n is enabled when bit CHICR0n.PATSEL is set to a value not equal to 00_B. The channel interrupt pointer CHICR0n.INTP defines which of the interrupt outputs SR[15:0] will be activated on a pattern detection interrupt or the channel interrupt pointer CHICR0n.INTP determines which of the interrupt outputs SR[15:0][1] will be activated on a pattern detection or channel interrupt.

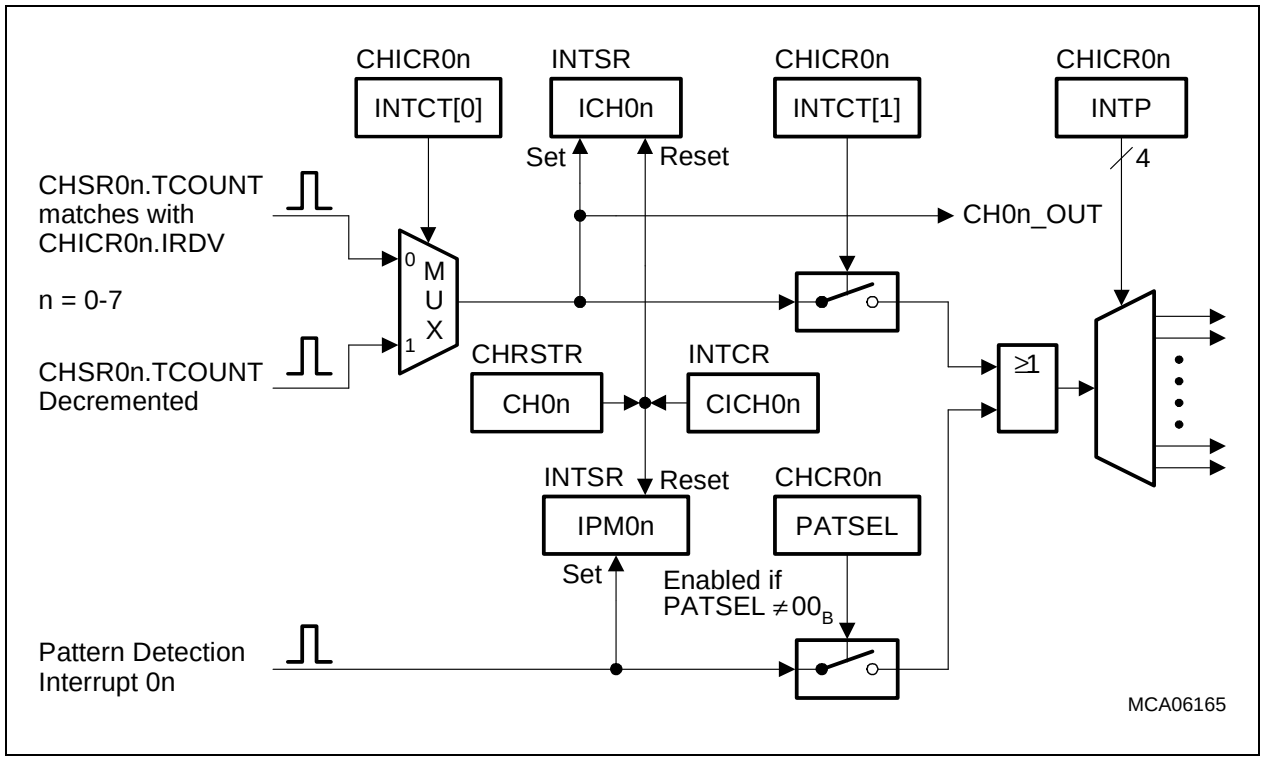


Figure 11-19 Channel Interrupts

1) In the TC1766, only SR[3:0] are connected to interrupt nodes.

11.1.8.2 Transaction Lost Interrupt

Each DMA channel 0n 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 0n is not finished, indicated by TRSR.CH0n still set. If such a transaction request lost condition occurs, bit ERRSR.TRL0n 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 0n is indicated by status flag ERRSR.TRL0n, which can be cleared by setting bit CLRE.CTL0n or CHRSTR.CH0n. The transaction lost interrupt for DMA channel 0n is enabled when bit EER.ETRL0n is set.

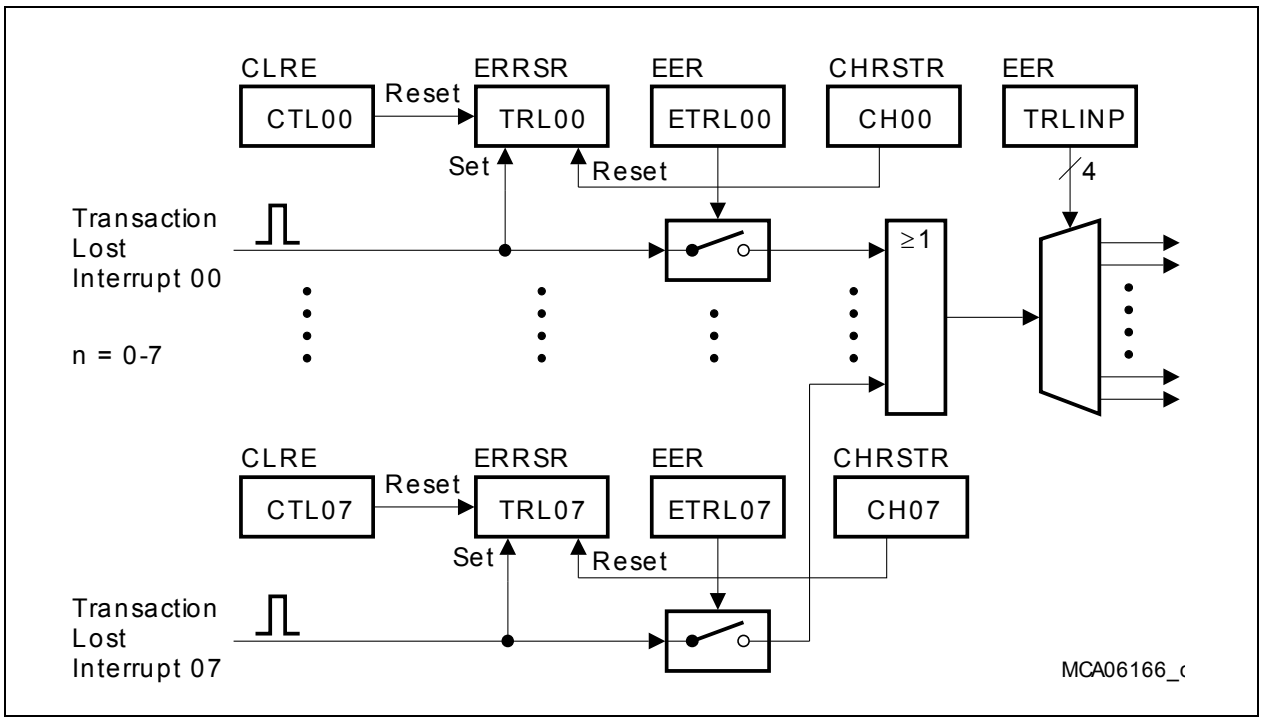


Figure 11-20 Transaction Lost Interrupt

1) In the TC1766, only SR[3:0] are connected to interrupt nodes.

11.1.8.3 Move Engine Interrupts

The Move Engine is able to detect error conditions that occur during accesses to the two FPI 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 error that occurred during a read move from the data source. A destination error indicates an FPI 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 cleared by software when setting bit CLRE.CME0SER. The source error interrupt of Move Engine 0 is enabled when bit EER.ME0SER is set. Separate clear, 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.

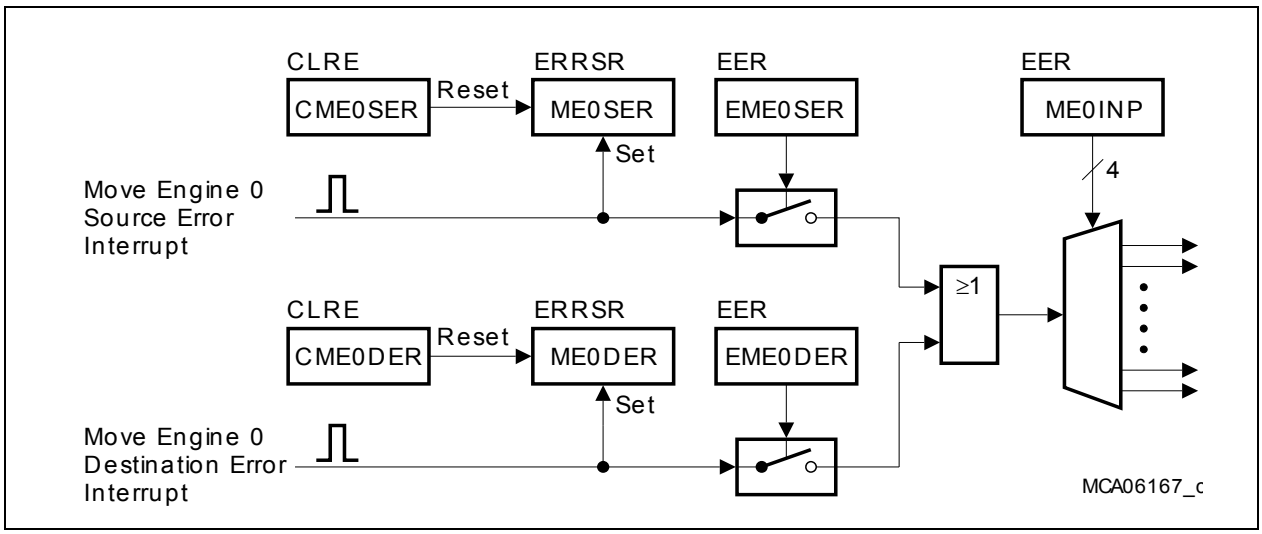


Figure 11-21 Move Engine Interrupts

1) In the TC1766, only SR[3:0] are connected to interrupt nodes.

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 FPI Bus interface a Move Engine 0 error occurred (FPI0ER and FPI1ER)
- 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 FPI Bus interface and DMA channel the Move Engine error has been generated. ERRSR.FPI0ER or ERRSR.FPI1ER is cleared when bits CLRE.CFPI0ER or CLRE.CFPI1ER is respectively set.

11.1.8.4 Wrap Buffer Interrupts

Each DMA channel 0n 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 11.1.9**.

A wrap source buffer interrupt of DMA channel 0n is indicated by status flag WRPSR.WRPS0n. A wrap destination buffer interrupt of DMA channel 0n is indicated by the status flag WRPSR.WRPD0n. Both interrupt status flags can be cleared by software when bit INTCR.CWRP0n (or CHRSTR.CH0n becomes set). The wrap source buffer interrupt is enabled when bit CHICR0n.WRPSE is set. The wrap destination buffer interrupt is enabled when bit CHICR0n.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 0n that can be directed to one of the interrupt outputs SR[15:0][1] by setting the wrap buffer interrupt pointer CHICR0n.WRPP with a corresponding value. Note that the pattern match should not be enabled while a wrap interrupt is enabled for the same channel.

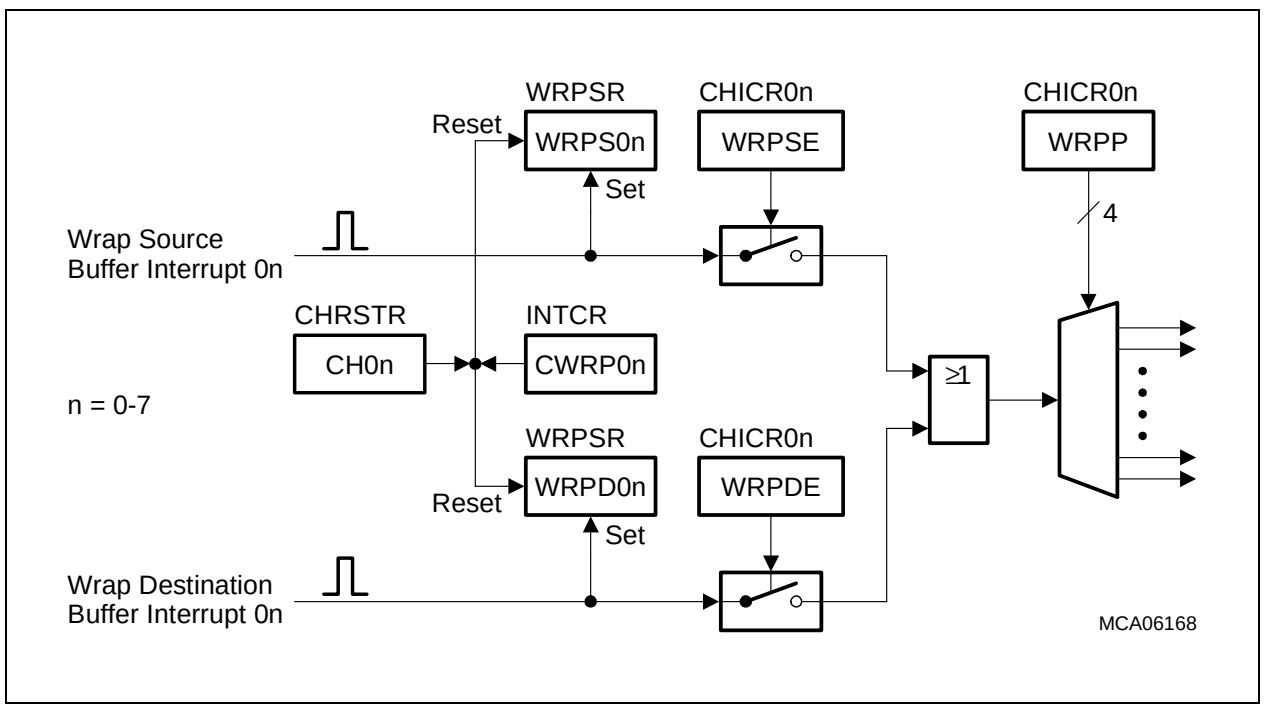


Figure 11-22 DMA Wrap Buffer Interrupts

1) In the TC1766, only SR[3:0] are connected to interrupt nodes.

11.1.8.5 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 11-23** 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 TC1766, only SR[3:0] are connected to interrupt nodes.

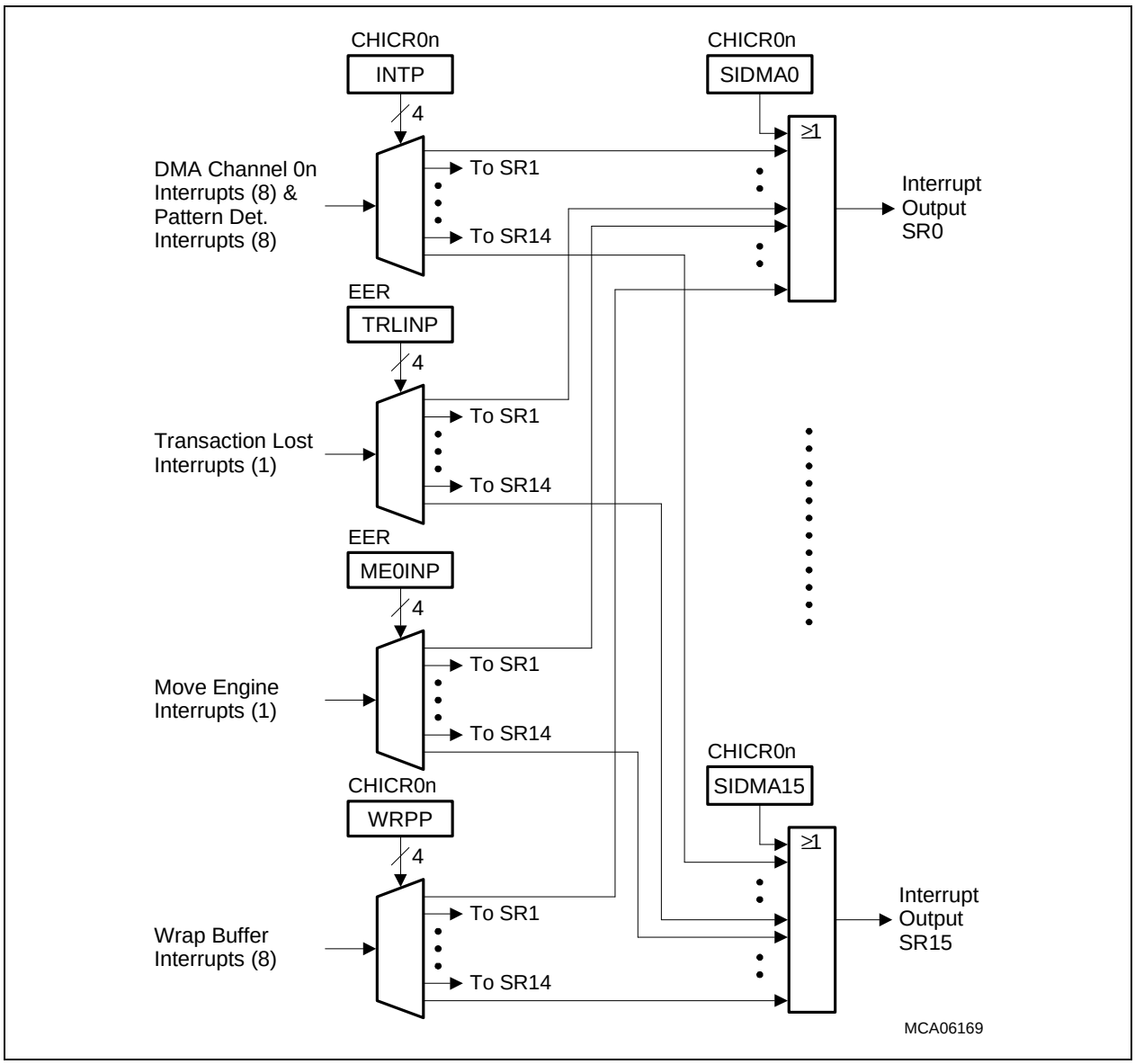


Figure 11-23 DMA Interrupt Request Compressor

11.1.9 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 0n 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 CHCR0n.CHDW. CHDW determines the data width for the read and write moves individually for each DMA channel 0n. Another control bit, CHCR0n.PATSEL, selects among the different operating modes for a specific value of CHDW.

Depending on CHCR0n.PATSEL and on the positive result of the comparison, two actions follow (if CHCR0n.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 0n using the Interrupt Pointer defined in CHICR0n.WRPP.
- Clear TRSR.HTRE0n and TRSR.CH0n in order to stop the current transaction (Hardware and Software request enable). The value of CHSR0n.TCOUNT can be read out by the interrupt SW.

The software will have to service the interrupt and to activate again the channel.

11.1.9.1 Pattern Compare Logic

Read move data and compare match patterns are compared on a bit-wise level. The logic as shown in **Figure 11-24** is implemented in each COMP block of **Figure 11-25**, **Figure 11-26**, and **Figure 11-27**. 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.

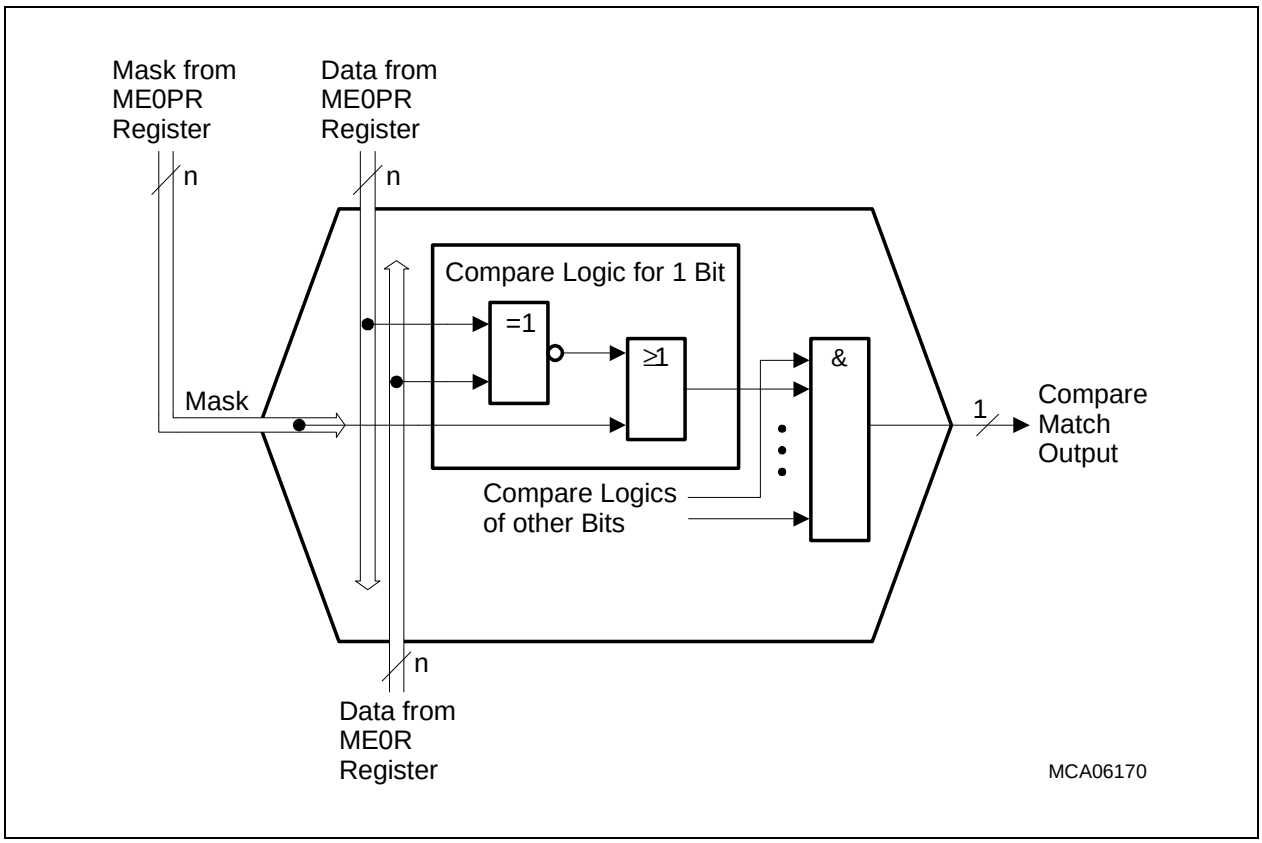


Figure 11-24 Pattern Compare Logic (COMP Block)

11.1.9.2 Pattern Detection for 8-bit Data Width

When 8-bit channel data width is selected (CHCR0n.CHDW = 00_B), the pattern detection logic is configured as shown in **Figure 11-25**. Three compare match configurations are possible.

Table 11-2 Pattern Detection for 8-bit Data Width

CHCR0n. 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 0n

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 CHCR0n.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.

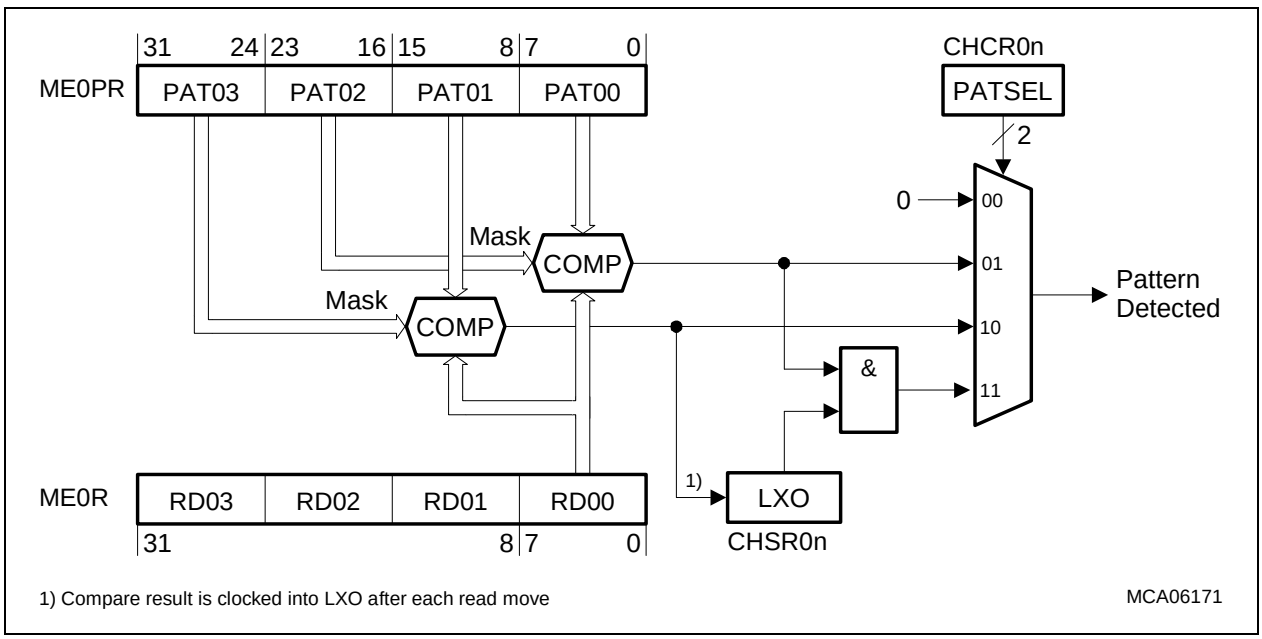


Figure 11-25 Pattern Detection for 8-bit Data Width (CHCR0n.CHDW = 00_B)

11.1.9.3 Pattern Detection for 16-bit Data Width

When 16-bit channel data width is selected (CHCR0n.CHDW = 01_B) the pattern detection logic can be configured as shown in **Figure 11-26**. Three compare match configurations are possible.

Table 11-3 Pattern Detection for 16-bit Data Width

CHCR0n. PATSEL	ADRCR0n. 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 0n
	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 0n
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.

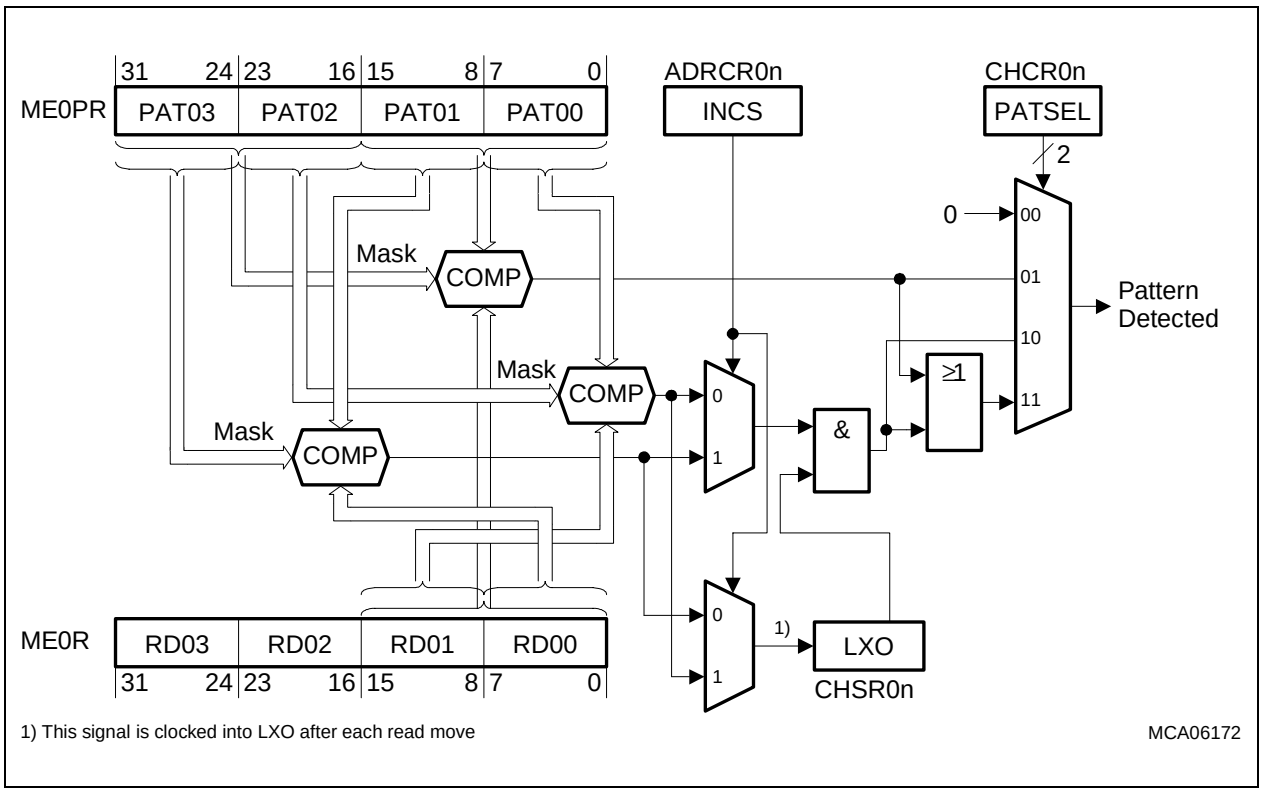


Figure 11-26 Pattern Detection for 16-bit Data Width (CHCR0n.CHDW = 01_B)

11.1.9.4 Pattern Detection for 32-bit Data Width

When 32-bit channel data width is selected (CHCR0n.CHDW = 10_B) the pattern detection logic is configured as shown in **Figure 11-27**. Three compare match configurations are possible.

Table 11-4 Pattern Detection for 32-bit Data Width

CHCR0n. 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.

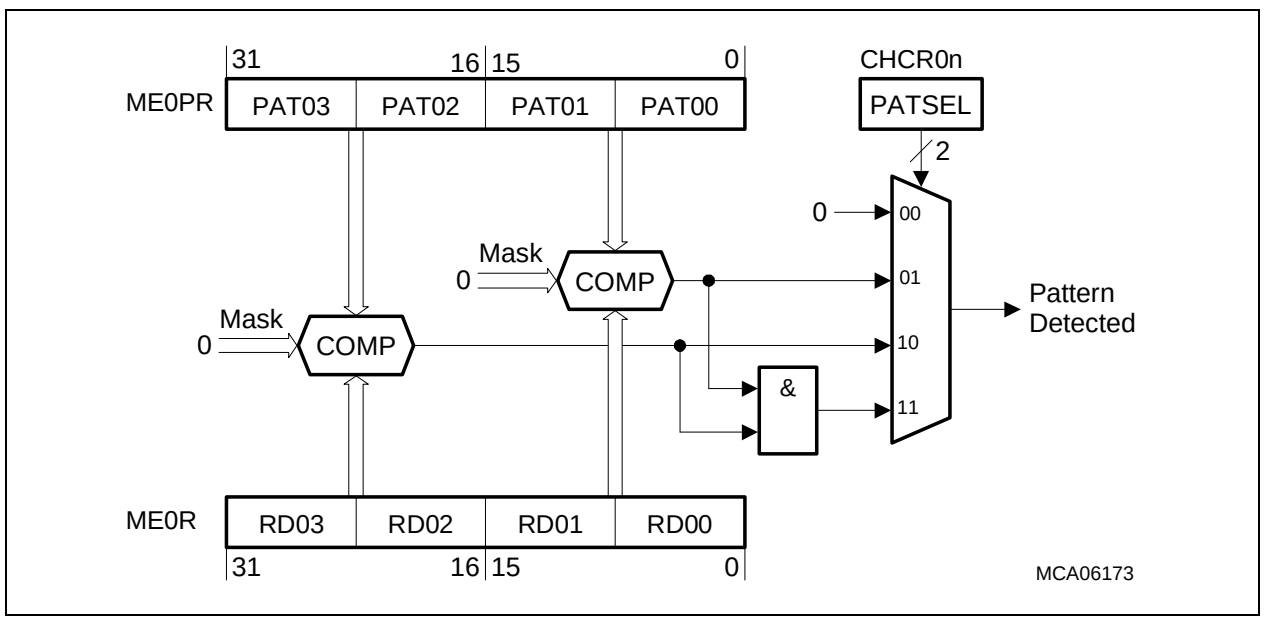


Figure 11-27 Pattern Detection for 32-bit Data Width (CHCR0n.CHDW = 10_B)

11.1.10 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 Access Enable Register ME0AENR. 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 11.1.8.3**).

There are four programmable address range extensions available for the Move Engine. These programmable address range extensions make it possible to define sub-ranges within four of the fixed address ranges. The parameters for the four sub-ranges are stored in the Move Engine 0 Access Range Register ME0ARR. 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 11-28 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 determines the sub-range size and bit field ME0ARR.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 are numbered by index “n” (n = 0-3).

Two sub-range examples (see **Figure 11-28**):

- 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 $3 \times 32 = 96$ 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 TC1766 are defined on ***Page 11-87****.*

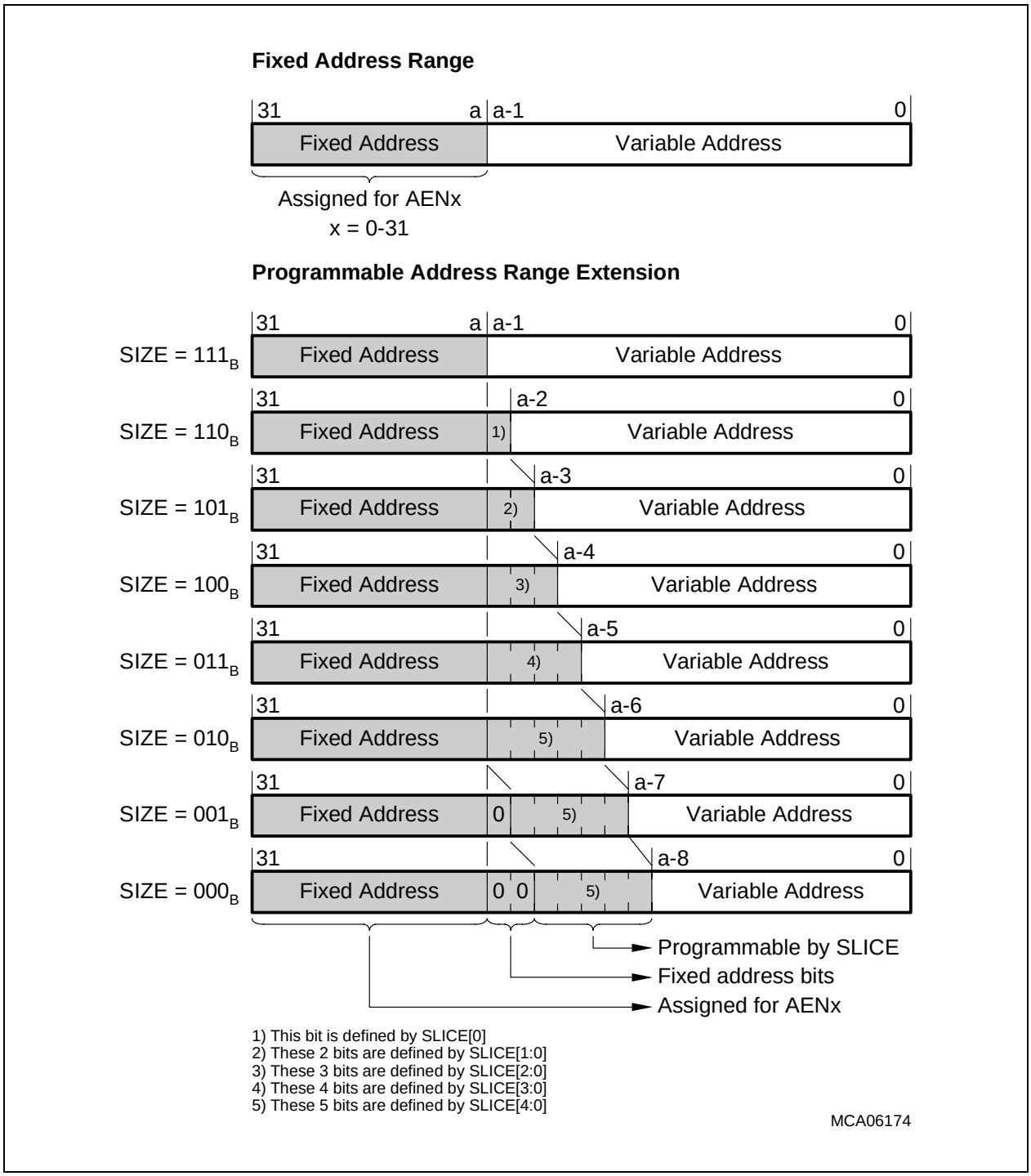


Figure 11-28 Access Protection Address Range Definitions

11.2 DMA Module Kernel Registers

Figure 11-29 and Table 11-5 show all registers associated with the DMA Controller Kernel. All DMA kernel register names described in this section are also referenced in other parts of the TC1766 User's Manual by the module name prefix "DMA_".

DMA Registers Overview

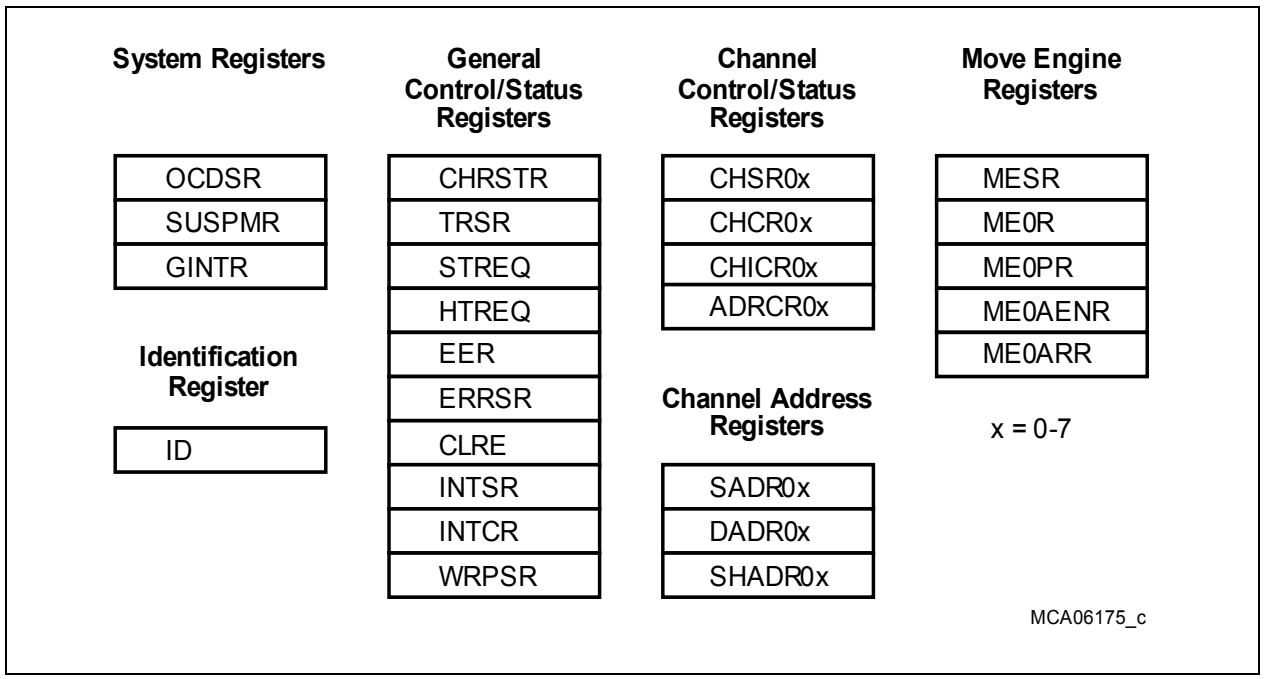


Figure 11-29 DMA Kernel Registers

The complete and detailed address map of the DMA module is described in Table 16-16 on Page 16-34 of the TC1766 User's Manual System Units part (Volume 1).

Attention: For documentation automation purposes, the DMA channel numbering index "x" is used in the DMA module kernel register descriptions. This index is represented by "n" for DMA channel numbering in the functional description. For example, an index reference "0x" is equivalent to index "0n". Indexes "n" and "x" (n = x = 0-7) refer to the DMA channels in the DMA Sub-Block 0.

Table 11-5 Registers Address Space - DMA Kernel Registers

Module	Base Address	End Address	Note
DMA	$F000\ 3C00_H$	$F000\ 3EFF_H$	-

Table 11-6 Registers Overview - DMA Kernel Registers

Register Short Name	Register Long Name	Offset Address[1)]	Description see
DMA_ID	DMA Module Identification Register	0008_H	Page 11-46
DMA_CHRSTR	DMA Channel Reset Request Register	0010_H	Page 11-50
DMA_TRSR	DMA Transaction Request State Register	0014_H	Page 11-51
DMA_STREQ	DMA Software Transaction Request Register	0018_H	Page 11-52
DMA_HTREQ	DMA Hardware Transaction Request Register	$001C_H$	Page 11-53
DMA_EER	DMA Enable Error Register	0020_H	Page 11-54
DMA_ERRSR	DMA Error Status Register	0024_H	Page 11-56
DMA_CLRE	DMA Clear Error Register	0028_H	Page 11-58
DMA_GINTR	DMA Global Interrupt Set Register	$002C_H$	Page 11-49
DMA_MESR	DMA Move Engine Status Register	0030_H	Page 11-63
DMA_ME0R	DMA Move Engine 0 Read Register	0034_H	Page 11-65
DMA_ME0PR	DMA Move Engine 0 Pattern Register	$003C_H$	Page 11-65
DMA_ ME0AENR	DMA Move Engine 0 Access Enable Register	0044_H	Page 11-66
DMA_ME0ARR	DMA Move Engine 0 Access Range Register	0048_H	Page 11-67
DMA_INTSR	DMA Interrupt Status Register	0054_H	Page 11-60
DMA_INTCR	DMA Interrupt Clear Register	0058_H	Page 11-62
DMA_WRPSR	DMA Wrap Status Register	$005C_H$	Page 11-61
DMA_OCDSR	DMA OCDS Register	0064_H	Page 11-46
DMA_SUSPMR	DMA Suspend Mode Register	0068_H	Page 11-48
DMA_CHSR0x	DMA Channel 0x Status Register (x = 0-7)	$x \times 20_H + 0080_H$	Page 11-73
DMA_CHCR0x	DMA Channel 0x Control Register (x = 0-7)	$x \times 20_H + 0084_H$	Page 11-69
DMA_CHICR0x	DMA Channel 0x Interrupt Control Register (x = 0-7)	$x \times 20_H + 0088_H$	Page 11-74

Table 11-6 Registers Overview - DMA Kernel Registers (cont'd)

Register Short Name	Register Long Name	Offset Address[1)]	Description see
DMA_ ADRCR0x	DMA Channel 0x Address Control Register (x = 0-7)	$x \times 20_H + 008C_H$	Page 11-76
DMA_SADR0x	DMA Channel 0x Source Address Register (x = 0-7)	$x \times 20_H + 0090_H$	Page 11-80
DMA_DADR0x	DMA Channel 0x Destination Address Register (x = 0-7)	$x \times 20_H + 0094_H$	Page 11-81
DMA_SHADR0x	DMA Channel 0x Shadow Address Register (x = 0-7)	$x \times 20_H + 0098_H$	Page 11-82

1) The absolute register address is calculated as follows:
Module Base Address (Table 11-5) + Offset Address (shown in this column)
Further, the following ranges for parameter x is valid: x = 0-7.

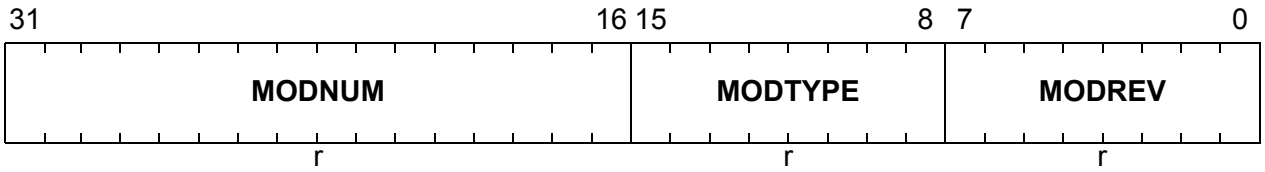
Note: Register bits marked "w" are virtual registers and do not contain flip-flops. They are always read as 0.

Note: The DMA registers can only be written in supervisor mode. Some of them are also Endinit-protected.

11.2.1 System Registers

The DMA Module Identification Register ID contains read-only information about the DMA module version.

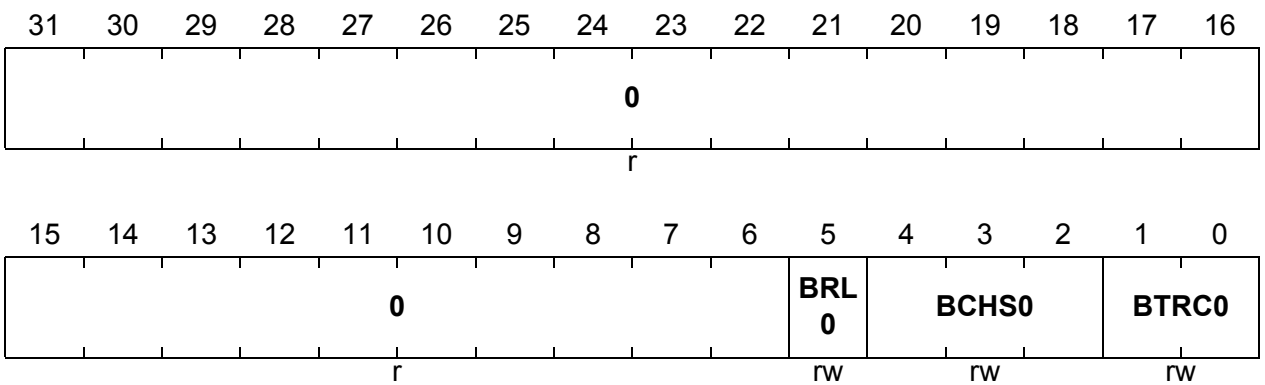
DMA_ID
DMA Module Identification Register (008_H) Reset Value: $001A\ C0XX_H$



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 DMA: $001A_H$

The OCDS Register describes the break capability of the DMA module. OCDSR is only reset with the OCDS Reset.

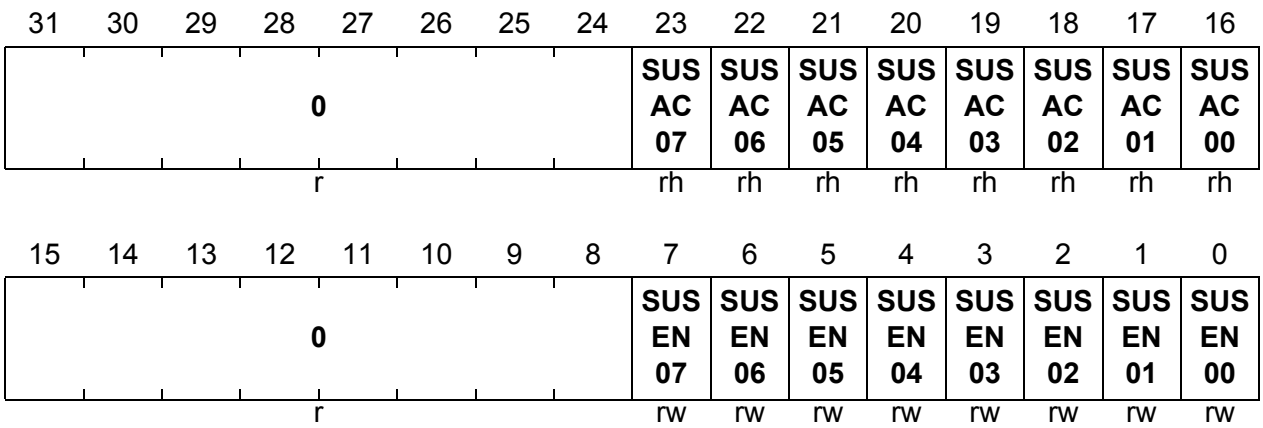
DMA_OCDSR
DMA OCDS Register (064_H) Reset Value: $0000\ 0000_H$



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.CH0x 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.CH0x changes from 0 to 1. 10_B A break condition is generated when TRSR.CH0x changes from 1 to 0. 11_B A break condition is generated when TRSR.CH0x 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.CH0x is observed for signal transitions as defined by BTRC0. 000_B DMA channel 00 selected 001_B DMA channel 01 selected … … 110_B DMA channel 06 selected 111_B DMA channel 07 selected
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.
0	[31:6]	r	**Reserved** Read as 0; should be written with 0.

The Suspend Mode Register contains bits for each DMA channel that allow the enabling/disabling of its soft suspend mode capability and to indicate its suspend status.

DMA_SUSPMR
DMA Suspend Mode Register (068$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
SUSEN0x (x = 0-7)	x	rw	**Suspend Enable for DMA Channel 0x** This bit enables the soft suspend capability individually for each DMA channel 0x. 0_B DMA channel 0x is disabled for Soft-suspend Mode. The DMA channel 0x does not react on an active suspend request signal SUSREQ. 1_B DMA channel 0x is enabled for Soft-suspend Mode. If the suspend request signal SUSREQ becomes active, a DMA transaction of DMA channel 0x is stopped after the current DMA transfer has been finished. Soft-suspend Mode can be terminated when SUSEN0x is written with 0.
SUSAC0x (x = 0-7)	x + 16	rh	**Suspend Active for DMA Channel 0x** This status bit indicates whether or not DMA channel 0x is in Soft-suspend Mode. 0_B DMA channel 0x is not in Soft-suspend Mode or internal actions are not yet finished after the Soft-suspend Mode was requested. 1_B DMA channel 0x is in Soft-suspend Mode.
0	[31:24], [15:8]	r	**Reserved** Read as 0; should be written with 0.

Note: Register SUSPMR is only reset by the OCDS 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
SI DMA 15	**SI DMA 14**	**SI DMA 13**	**SI DMA 12**	**SI DMA 11**	**SI DMA 10**	**SI DMA 9**	**SI DMA 8**	**SI DMA 7**	**SI DMA 6**	**SI DMA 5**	**SI DMA 4**	**SI DMA 3**	**SI DMA 2**	**SI DMA 1**	**SI DMA 0**
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.

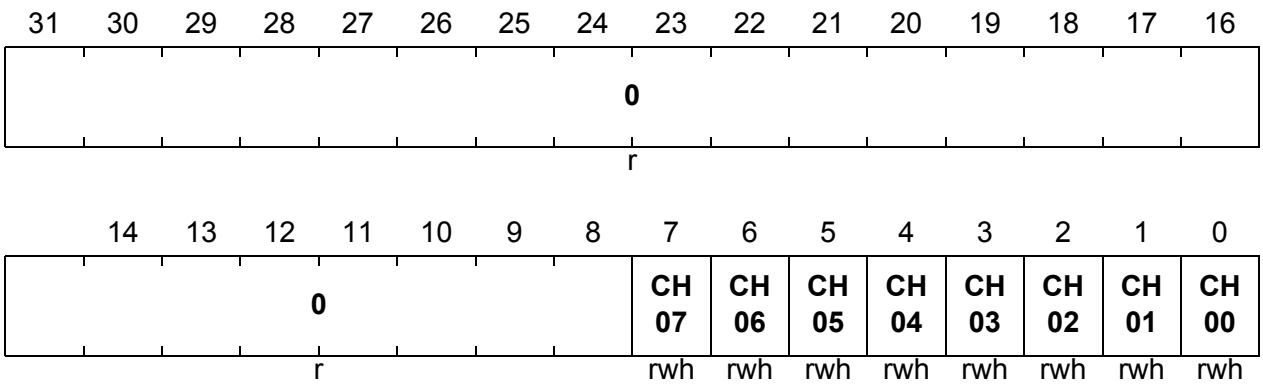
Note: In the TC1766, only interrupt output lines SR[3:0] are available. Therefore, only bits SIDMA[3:0] are effective.

11.2.2 General Control/Status Registers

The bits in the Channel Reset Request Register are used to reset DMA channel 0x.

DMA_CHRSTR
DMA Channel Reset Request Register

(010_H) **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
CH0x (x = 0-7)	x	rwh	**Channel 0x Reset** These bits force the DMA channel 0x 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 CH0x has no effect. 0_B No action (write) or the requested channel reset has been reset (read). 1_B DMA channel 0x is stopped. More details see Page 11-16.
0	[31:8]	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: $0000\ 0000_H$**

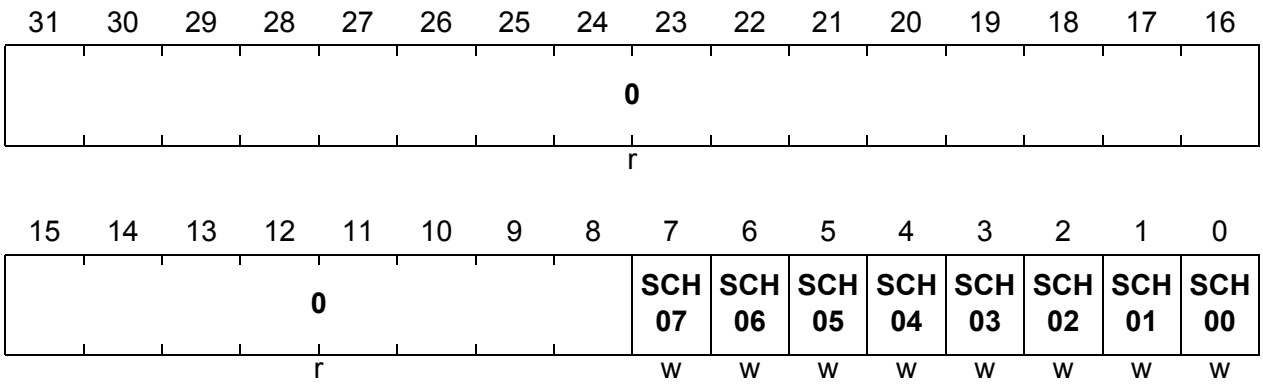
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0								HTRE07	HTRE06	HTRE05	HTRE04	HTRE03	HTRE02	HTRE01	HTRE00
r								rh	rh	rh	rh	rh	rh	rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								CH07	CH06	CH05	CH04	CH03	CH02	CH01	CH00
r								rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
CH0x (x = 0-7)	x	rh	**Transaction Request State of DMA Channel 0x** 0_B No DMA request is pending for channel 0x. 1_B A DMA request is pending for channel 0x. CH0x is cleared when a pattern match is detected.
HTRE0x (x = 0-7)	x +16	rh	**Hardware Transaction Request Enable State of DMA Channel 0x** 0_B Hardware transaction request for DMA Channel 0x is disabled. An input DMA request will not trigger the channel 0x. 1_B Hardware transaction request for DMA Channel 0x is enabled. The transfers of a DMA transaction are controlled by the corresponding channel request line of the DMA requesting source. HTRE0x is set to 0 when CHSR0x.TCOUNT is decremented and CHSR0x.TCOUNT = 0. HTRE0x can be enabled and disabled with HTREQ.ECH0x or HTREQ.DCH0x. HTRE0x is cleared when a pattern match is detected.
0	[31:24], [15:8]	r	**Reserved** Read as 0; should be written with 0.

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: $0000\ 0000_H$**



Field	Bits	Type	Description
SCH0x (x = 0-7)	x	w	**Set Transaction Request for DMA Channel 0x** 0_B No action. 1_B A transaction for DMA channel 0x is requested. When setting SCH0x, TRSR.CH0x becomes set to indicate that a DMA request is pending for DMA channel 0x.
0	[31:8]	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
0								DCH 07	DCH 06	DCH 05	DCH 04	DCH 03	DCH 02	DCH 01	DCH 00
r								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								ECH 07	ECH 06	ECH 05	ECH 04	ECH 03	ECH 02	ECH 01	ECH 00
r								w	w	w	w	w	w	w	w

Field	Bits	Type	Description
ECH0x (x = 0-7)	x	w	**Enable Hardware Transfer Request for DMA Channel 0x** see table below
DCH0x (x = 0-7)	x + 16	w	**Disable Hardware Transfer Request for DMA Channel 0x** see table below
0	[31:24], [15:8]	r	**Reserved** Read as 0; should be written with 0.

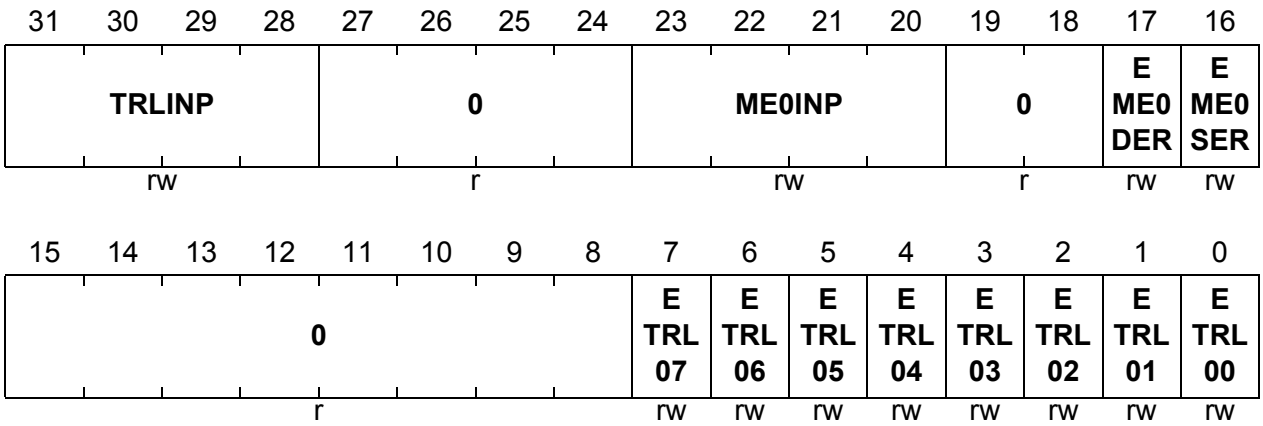
Table 11-7 Conditions to Set/Clear the Bits TRSR.HTRE0x

HTREQ.ECH0x	HTREQ.DCH0x	Transaction Finishes[1] for Channel 0x	Modification of TRSR.HTRE0x
0	0	0	Unchanged
1	0	0	Set
X	1	X	Clear
X	X	1	Clear

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: $0000\ 0000_H$**



Field	Bits	Type	Description
ETRL0x (x = 0-7)	x	rw	**Enable Transaction Request Lost for DMA Channel 0x** This bit enables the generation of an interrupt when the set condition for ERRSR.TRL0x is detected. 0_B The interrupt generation for a request lost event for channel 0x is disabled. 1_B The interrupt generation for a request lost event for channel 0x 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.
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.

Field	Bits	Type	Description
ME0INP	[23:20]	rw	**Move Engine 0 Error Interrupt Node Pointer** ME0INP determines the number x (x = 0-15) of the service request output SRx 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 … … 1111_B SR15 selected for Move Engine 0 interrupt *Note: In the TC1766, only SR[3:0] are connected to interrupt nodes.*
TRLINP	[31:28]	rw	**Transaction Lost Interrupt Node Pointer** TRLINP determines the number x (x = 0-15) of the service request output SRx that becomes active on a transaction lost interrupt. 0000_B SR0 selected for transaction lost interrupt 0001_B SR1 selected for transaction lost interrupt … … 1111_B SR15 selected for transaction lost interrupt *Note: In the TC1766, only SR[3:0] are connected to interrupt nodes.*
0	[15:8], [19:18], [27:24]	r	**Reserved** Read as 0; should be written with 0.

The Error Status Register indicates if the DMA controller could not answer to a request because the previous request was not terminated (see **Section 11.1.4.4**). 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
MLI1	**0**			**MLI0**	**LEC ME0**			**0**		**FPI1 ER**	**FPI0 ER**	**0**		**ME0 DER**	**ME0 SER**
rh	r			rh	rh			r		rh	rh	r		rh	rh

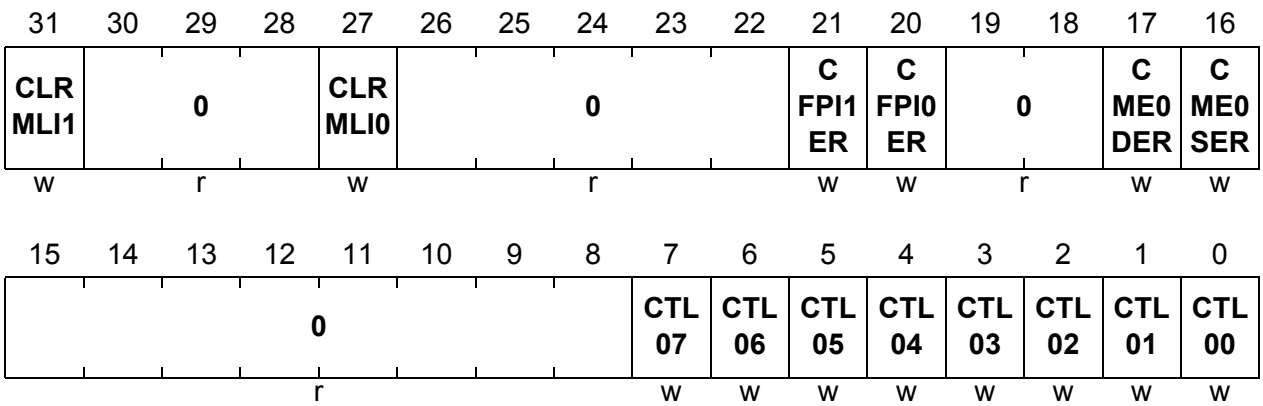
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								**TRL 07**	**TRL 06**	**TRL 05**	**TRL 04**	**TRL 03**	**TRL 02**	**TRL 01**	**TRL 00**
r								rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
TRL0x (x = 0-7)	x	rh	**Transaction/Transfer Request Lost of DMA Channel 0x** 0_B No request lost event has been detected for channel 0x. 1_B A new DMA request was detected while TRSR.CH0x=1 (request lost event). This bit is cleared by software when writing a 1 to CLRE.CTL0x, or by a channel reset (writing CHRSTR.CH0x = 1).
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.

Field	Bits	Type	Description
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.
FPI0ER	20	rh	**SPB Error** This bit is set whenever a move that has been started by the DMA/MLI master interface on FPI Bus interface 0 leads to an error. 0_B No error occurred. 1_B An error occurred on FPI Bus interface 0.
FPI1ER	21	rh	**RPB Error** This bit is set whenever a move that has been started by the DMA/MLI master interface on FPI Bus interface 1 leads to an error. 0_B No error occurred. 1_B An error occurred on FPI Bus interface 1.
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 FPI Bus error that has occurred.
MLI0	27	rh	**MLI0 Error Source** This bit is set whenever an FPI Bus error occurred due to an action of MLI0. 0_B No FPI error occurred due to MLI0. 1_B An FPI error occurred due to MLI0.
MLI1	31	rh	**MLI1 Error Source** This bit is set whenever an FPI Bus error occurred due to an action of MLI1. 0_B No FPI error occurred due to MLI1. 1_B An FPI error occurred due to MLI1.
0	[15:8], [19:18], [23:22], [30:28]	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$_H$**



Field	Bits	Type	Description
CTL0x (x = 0-7)	x	w	**Clear Transaction Request Lost for DMA Channel 0x** 0_B No action 1_B Clear DMA channel 0x transaction request lost flag ERRSR.TRL0x
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.
CFPI0ER	20	w	**Clear SPB Error** 0_B No action 1_B Clear error flag ERRSR.FPI0ER.
CFPI1ER	21	w	**Clear RPB Error** 0_B No action 1_B Clear error flag ERRSR.FPI1ER.
CLRMLI0	27	w	**Clear MLI0 Error** 0_B No action 1_B Clear error flag ERRSR.MLI0.

Field	Bits	Type	Description
CLRMLI1	31	w	**Clear MLI1 Error** 0_B No action 1_B Clear error flag ERRSR.MLI1.
0	[15:8], [19:18], [26:22], [30:28]	r	**Reserved** Read as 0; should be written with 0.

The Interrupt Status Register indicates if CHSR0x.TCOUNT matches with CHCR0x.IRDV, or if CHSR0x.TCOUNT has been decremented (depending on CHICR0x.INTCT[0]), or if a pattern has been detected. These conditions can also generate an interrupt if enabled (see **Figure 11-19** on **Page 11-28**).

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
0								IPM 07	IPM 06	IPM 05	IPM 04	IPM 03	IPM 02	IPM 01	IPM 00
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								ICH 07	ICH 06	ICH 05	ICH 04	ICH 03	ICH 02	ICH 01	ICH 00
r								rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
ICH0x (x = 0-7)	x	rh	**Interrupt from Channel 0x** This bit indicates that channel 0x has raised an interrupt for TCOUNT = IRDV or if TCOUNT has been decremented (depending on CHICR.INTCT[0]. This bit (and IP0x) is cleared by software when writing a 1 to INTCR.CICH0x or by a channel reset (writing CHRSTR.CH0x = 1). 0_B A channel interrupt has not been detected. 1_B A channel interrupt has been detected.
IPM0x (x = 0-7)	x + 16	rh	**Pattern Detection from Channel 0x** This bit indicates that a pattern has been detected for channel 0x while the pattern detection has been enabled. This bit (and ICH0x) is cleared by software when writing a 1 to INTCR.CICH0x or by a channel reset (writing CHRSTR.CH0x = 1). 0_B A pattern has not been detected. 1_B A pattern has been detected.
0	[15:8], [31:24]	r	**Reserved** Read as 0; should be written with 0.

The Wrap Status Register provides information on the channels that perform 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
0								WRPD07	WRPD06	WRPD05	WRPD04	WRPD03	WRPD02	WRPD01	WRPD00
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								WRPS07	WRPS06	WRPS05	WRPS04	WRPS03	WRPS02	WRPS01	WRPS00
r								rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
WRPS0x (x = 0-7)	x	rh	**Wrap Source Buffer for Channel 0x** These bits indicate which channels have done a wrap-around of their source buffer(s). 0_B No wrap-around occurred for channel 0x. 1_B A wrap-around occurred for channel 0x. This bit is cleared by software by writing a 1 to INTCR.CWRP0x or CHRSTR.CH0x.
WRPD0x (x = 0-7)	x +16	rh	**Wrap Destination Buffer for Channel 0x** These bits indicate which channels have done a wrap-around of their destination buffer(s). 0_B No wrap-around occurred for channel 0x. 1_B A wrap-around occurred for channel 0x. This bit is cleared by software by writing a 1 to INTCR.CWRP0x or CHRSTR.CH0x.
0	[15:8], [31:24]	r	**Reserved** Read as 0; should be written with 0.

The bits in the Interrupt Clear Register allow the channel interrupt flags and the wrap buffer interrupt flags for DMA Channels 0x to be cleared.

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
0								CWRP07	CWRP06	CWRP05	CWRP04	CWRP03	CWRP02	CWRP01	CWRP00
r								w	w	w	w	w	w	w	w

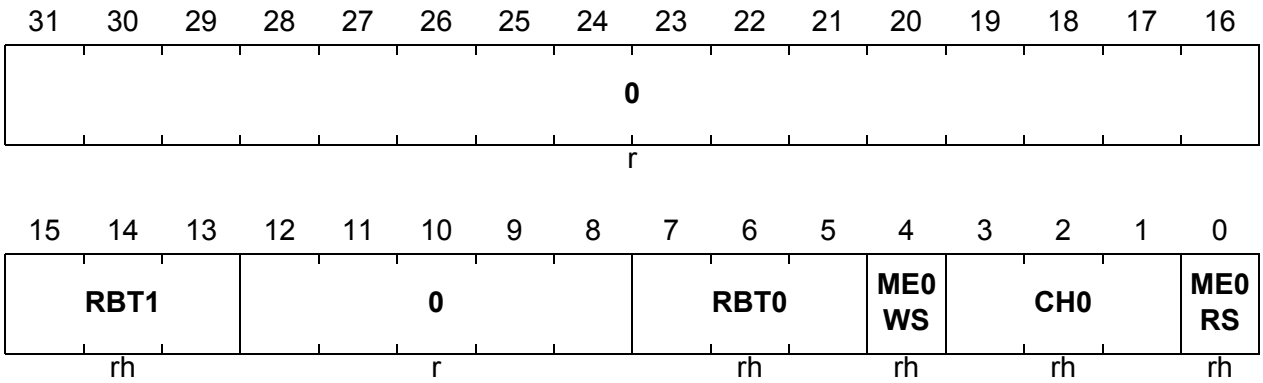
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								CICH07	CICH06	CICH05	CICH04	CICH03	CICH02	CICH01	CICH00
r								w	w	w	w	w	w	w	w

Field	Bits	Type	Description
CICH0x (x = 0-7)	x	w	**Clear Interrupt for DMA Channel 0x** These bits allow the channel interrupt flags INTSR.ICH0x and INTSR.IPM0x of DMA channel 0x to be cleared by software. 0_B No action. 1_B Bits INTSR.ICH0x and INTSR.IPM0x are cleared.
CWRP0x (x = 0-7)	x + 16	w	**Clear Wrap Buffer Interrupt for DMA Channel 0x** These bits allow the wrap source buffer interrupt flag WRPSR.WRPS0x and the wrap destination buffer interrupt flag WRPSR.WRPD0x (both together) of DMA channel 0x to be cleared by software. 0_B No action. 1_B Bits WRPSR.WRPS0x and WRPSR.WRPD0x are cleared.
0	[15:8], [31:24]	r	**Reserved** Read as 0; should be written with 0.

11.2.3 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: $0000\ 0000_H$

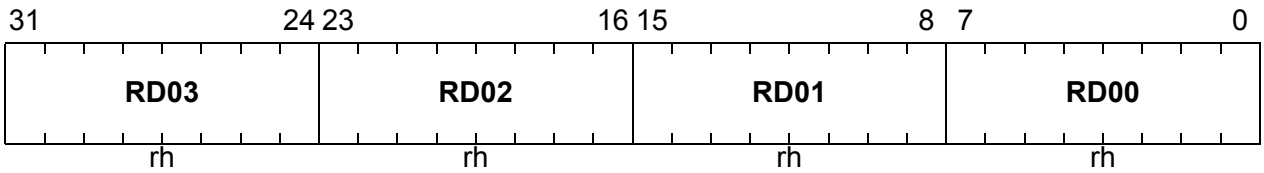


Field	Bits	Type	Description
ME0RS	0	rh	**Move Engine 0 Read Status** 0_B Move Engine 0 is not performing a read. 1_B Move Engine 0 is performing a read.
CH0	[3:1]	rh	**Reading Channel in Move Engine 0** This bit field indicates which channel number is currently being processed by the Move Engine 0.
ME0WS	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.
RBT0	[7:5]	rh	**Read Buffer Trace for FPI Bus Interface 0** This bit field contains trace information from the buffer in the FPI Bus Interface 0. In the TC1766 it indicates the source of a bus access to the DMA Bus. 000_B Reset value or bridge from FPI0 to FPI1 001_B DMA Move Engine 0 011_B MLI0 100_B MLI1 Other bit combinations are reserved. RBT0 is useful for emulation purposes. It is not recommended to evaluate this bit field during normal operation of the TC1766.

Field	Bits	Type	Description
RBT1	[15:13]	rh	**Read Buffer Trace for FPI Bus Interface 1** This bit field contains trace information from the buffer in the FPI Bus Interface 1. In the TC1766, it indicates the source of a bus access to the System Peripheral Bus. 000_B Reset value or bridge from FPI0 to FPI1 001_B DMA Move Engine 0 011_B MLI0 100_B MLI1 Other bit combinations are reserved. RBT1 is useful for emulation purposes. It is not recommended to evaluate this bit field during normal operation of the TC1766.
0	[31:16], [12:8]	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 CHCR0x.PATSEL.

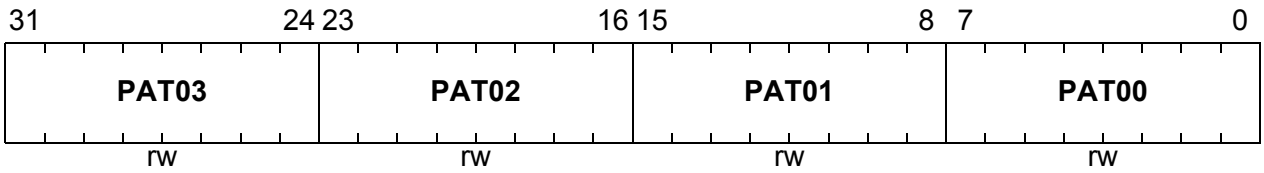
DMA_ME0R
DMA Move Engine 0 Read Register (034$_H$) Reset Value: 0000 0000$_H$



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$_H$



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 (CHCR0x.PATSEL) and channel data width (CHCR0x.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**

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

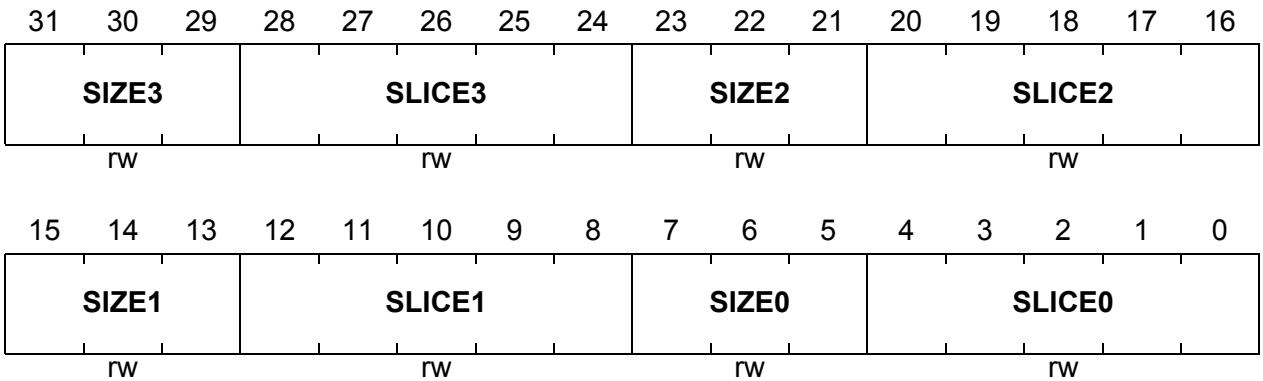
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AEN 15	**AEN 14**	**AEN 13**	**AEN 12**	**AEN 11**	**AEN 10**	**AEN 9**	**AEN 8**	**AEN 7**	**AEN 6**	**AEN 5**	**AEN 4**	**AEN 3**	**AEN 2**	**AEN 1**	**AEN 0**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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 11-10** on **Page 11-87** for the TC1766-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 11-28** for bit field definitions.

DMA_ME0ARR
DMA Move Engine 0 Access Range Register
(048_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.
SIZE2	[23:21]	rw	**Address Size 2** SIZE2 determines the sub-range size within address range extension 2.

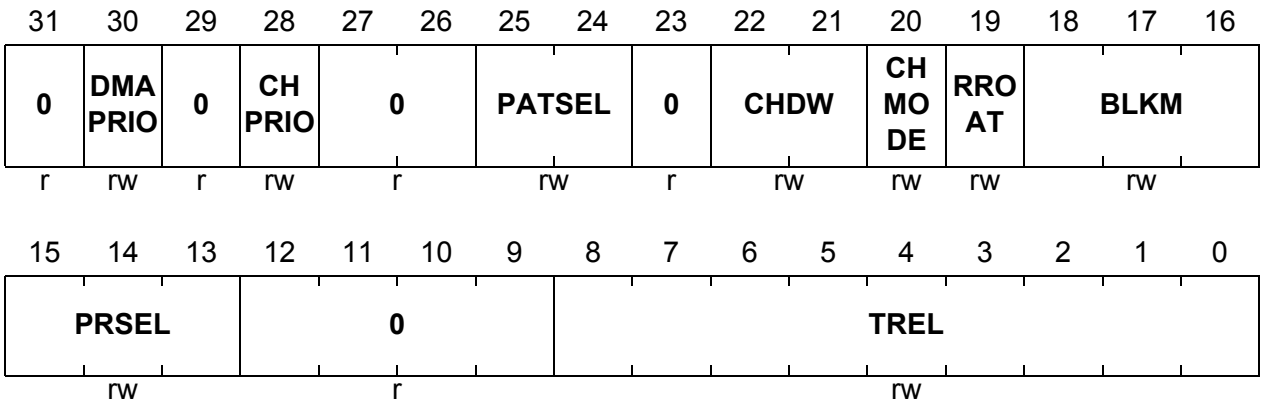
Field	Bits	Type	Description
SLICE3	[28:24]	rw	**Address Slice 3** SLICE3 selects a specific sub-range within address range extension 3.
SIZE3	[31:29]	rw	**Address Size 3** SIZE3 determines the sub-range size within address range extension 3.

*Note: See **Section 11.3.2** on **Page 11-87** for the TC1766-specific address range and address range extension definitions.*

11.2.4 Channel Control/Status Registers

The Channel Control Register for DMA channel 0x 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**



Field	Bits	Type	Description
TREL	[8:0]	rw	**Transfer Reload Value** This bit field contains the number of DMA transfers for s DMA transaction of DMA channel 0x. This 9-bit transfer count value is loaded into CHSR0x.TCOUNT at the start of a DMA transaction (when TRSR.CH0x becomes set and CHSR0x.TCOUNT = 0). TREL can be written during a running DMA transaction because TCOUNT will be updated (decremented) during the DMA transaction. If TREL = 0 or if TREL = 1, TCOUNT will be loaded with 1 when a new transaction is started (at least one DMA transfer must be executed per DMA transaction).

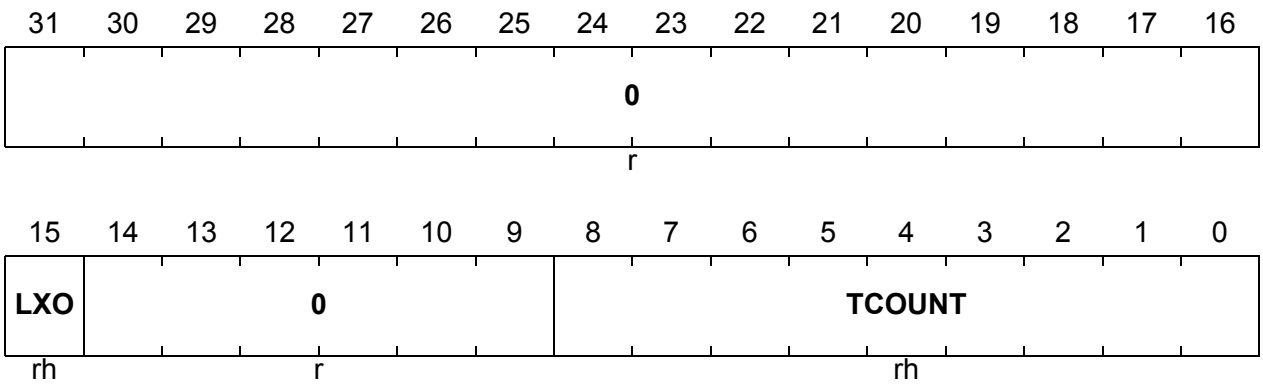
Field	Bits	Type	Description
PRSEL	[15:13]	rw	**Peripheral Request Select** This bit field controls the hardware request input multiplexer of DMA channel 0x (see **Figure 11-6** on **Page 11-10**). 000_B Input CH0x_REQI0 selected 001_B Input CH0x_REQI1 selected 010_B Input CH0x_REQI2 selected 011_B Input CH0x_REQI3 selected 100_B Input CH0x_REQI4 selected 101_B Input CH0x_REQI5 selected 110_B Input CH0x_REQI6 selected 111_B Input CH0xn_REQI7 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 moves 010_B One DMA transfer has 4 DMA moves 011_B One DMA transfer has 8 DMA moves 100_B One DMA transfer has 16 DMA moves Other bit combinations are reserved. See also **Figure 11-10** on **Page 11-17**.
RROAT	19	rw	**Reset Request Only After Transaction** RROAT determines whether or not the TRSR.CH0x transfer request state flag is cleared after each transfer. 0_B TRSR.CH0x is cleared after each transfer. A transfer request is required for each transfer. 1_B TRSR.CH0x is cleared when 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 clearing condition for control bit TRSR.HTRE0x of DMA channel 0x. 0_B Single Mode operation is selected for DMA channel 0x. After a transaction, DMA channel 0x is disabled for further hardware requests (TRSR.HTRE0x is cleared by hardware). TRSR.HTRE0x must be set again by software for starting a new transaction. 1_B Continuous Mode operation is selected for DMA channel 0x. After a transaction, bit TRSR.HTRE0x remains set.
CHDW	[22:21]	rw	**Channel Data Width** CHDW determines the data width for the read and write moves of DMA channel 0x. 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 11-2** on **Page 11-37**. **16-bit channel data width (CHDW = 01_B):** Selected pattern detection configuration see **Table 11-3** on **Page 11-38**. **32-bit channel data width (CHDW = 10_B):** Selected pattern detection configuration see **Table 11-4** on **Page 11-40**.
CHPRIO	28	rw	**Channel Priority** CHPRIO determines the priority of DMA channel 0x for the channel arbitration of Move Engine 0. 0_B DMA channel 0x has a low channel priority. 1_B DMA channel 0x has a high channel priority.

Field	Bits	Type	Description
DMAPRIO	30	rw	**DMA Priority** This bit determines the a bus request priority that is used when a move operation related to channel 0x is requesting FPI Bus 0. This bit has no effect in channel prioritization. 0_B Low priority selected 1_B High priority selected
0	[12:9], 23, [27:26], 29, 31	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**

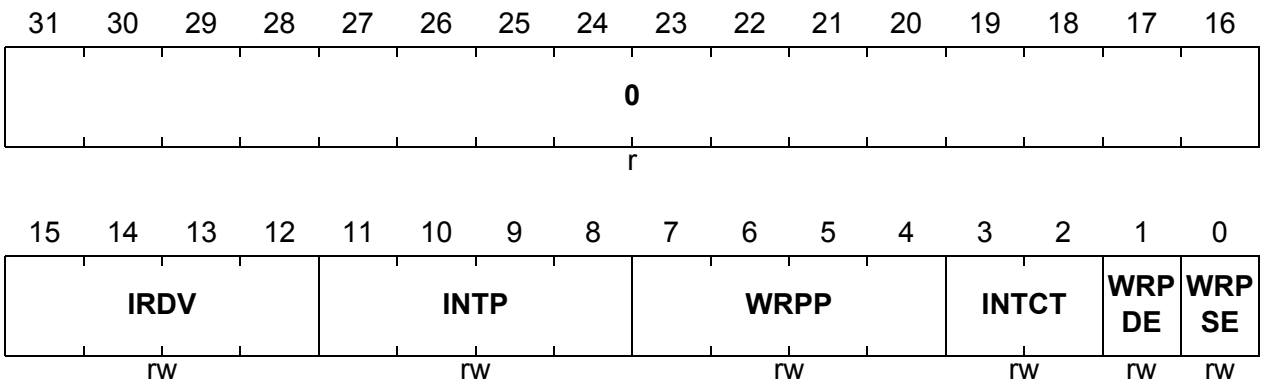


Field	Bits	Type	Description
TCOUNT	[8:0]	rh	**Transfer Count Status** TCOUNT holds the actual value of the DMA transfer count for DMA channel 0x. TCOUNT is loaded with the value of CHCR0x.TREL when TRSR.CH0x becomes set (and TCOUNT = 0). After each DMA transfer, TCOUNT is decremented by 1.
LXO	15	rh	**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 11-2** and **Figure 11-25** 16-bit data width: see **Table 11-3** and **Figure 11-26** 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	[14:9], [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**



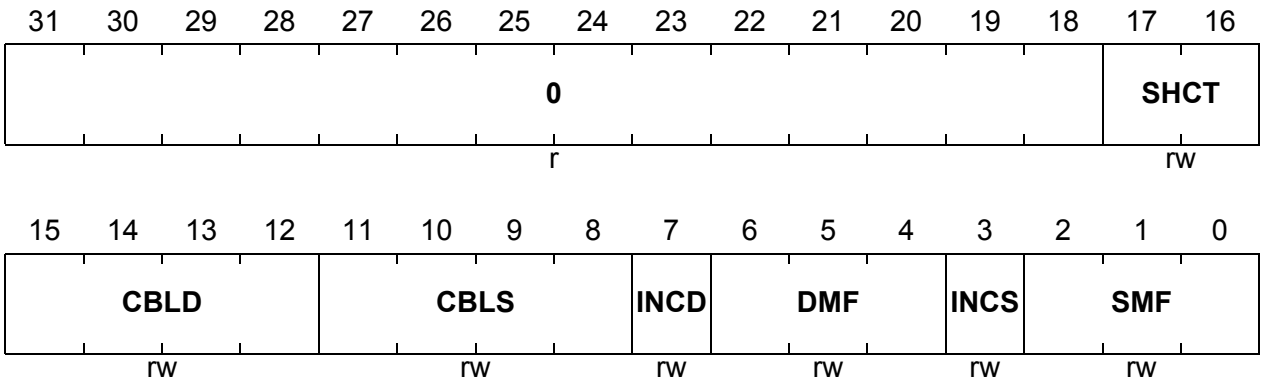
Field	Bits	Type	Description
WRPSE	0	rw	**Wrap Source Enable** 0_B Wrap source buffer interrupt disabled 1_B Wrap source buffer interrupt enabled
WRPDE	1	rw	**Wrap Destination Enable** 0_B Wrap destination buffer interrupt disabled 1_B Wrap destination buffer interrupt enabled
INTCT	[3:2]	rw	**Interrupt Control** 00_B No interrupt will be generated on changing the TCOUNT value. The bit INTSR.ICH0x is set when TCOUNT equals IRDV. 01_B No interrupt will be generated on changing the TCOUNT value. The bit INTSR.ICH0x is set when TCOUNT is decremented. 10_B An interrupt is generated and bit INTSR.ICH0x is set each time TCOUNT equals IRDV. 11_B Interrupt is generated and bit INTSR.ICH0x is set each time TCOUNT is decremented. (see **Figure 11-19**)

Field	Bits	Type	Description
WRPP	[7:4]	rw	**Wrap Pointer** WRPP determines the number x (x = 0-15) of the service request output SRx that becomes active on a wrap buffer interrupt. 0000_B SR0 selected for channel 0x wrap buffer interrupt 0001_B SR1 selected for channel 0x wrap buffer interrupt … … 1111_B SR15 selected for channel 0x wrap buffer interrupt *Note: In the TC1766, only SR[3:0] are connected to interrupt nodes.*
INTP	[11:8]	rw	**Interrupt Pointer** INTP determines the number x (x = 0-15) of the service request output SRx that becomes active on a channel interrupt. 0000_B SR0 selected for channel 0x interrupt 0001_B SR1 selected for channel 0x interrupt … … 1111_B SR15 selected for channel 0x interrupt *Note: In the TC1766, only SR[3:0] are connected to interrupt nodes.*
IRDV	[15:12]	rw	**Interrupt Raise Detect Value** These bits specify the value of CHSR0x.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**



Field	Bits	Type	Description
SMF	[2:0]	rw	**Source Address Modification Factor** This bit field and the data width as defined in CHCR0x.CHDW determine an address offset value by which the source address is modified after each DMA move. See also **Table 11-8**. 000_B Address offset is 1 × CHCR0x.CHDW. 001_B Address offset is 2 × CHCR0x.CHDW. 010_B Address offset is 4 × CHCR0x.CHDW. 011_B Address offset is 8 × CHCR0x.CHDW. 100_B Address offset is 16 × CHCR0x.CHDW. 101_B Address offset is 32 × CHCR0x.CHDW. 110_B Address offset is 64 × CHCR0x.CHDW. 111_B Address offset is 128 × CHCR0x.CHDW.
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.

Field	Bits	Type	Description
DMF	[6:4]	rw	**Destination Address Modification Factor** This bit field and the data width as defined in CHCR0x.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 11-8**. 000_B Address offset is 1 × CHDW. 001_B Address offset is 2 × CHDW. 010_B Address offset is 4 × CHDW. 011_B Address offset is 8 × CHDW. 100_B Address offset is 16 × CHDW. 101_B Address offset is 32 × CHDW. 110_B Address offset is 64 × CHDW. 111_B Address offset is 128 × 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.
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 11.1.4.7**). 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. … … 1110_B Source address SADR[31:14] is not updated. 1111_B Source address SADR[31:15] is not updated.

Field	Bits	Type	Description
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 11-19). 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. … … 1110_B Destination address DADR[31:14] is not updated. 1111_B Destination address DADR[31:15] is not updated.
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 SADR0x, the address is buffered in SHADR0x and transferred to SADR0x with the start of the next DMA transaction. 10_B Shadow address register used for destination address buffering. When writing to DADR0x, the address is buffered in SHADR0x and transferred to DADR0x 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.
0	[31:18]	r	**Reserved** Read as 0; should be written with 0.

Table 11-8 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 11-8 Address Offset Calculation Table

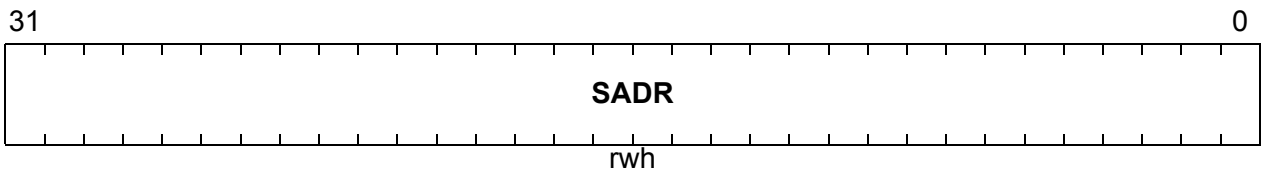
CHCR0x.CHDW = 00_B (8-bit Data Width)			CHCR0x.CHDW = 01_B (16-bit Data Width)			CHCR0x.CHDW = 10_B (32-bit Data Width)		
SMF DMF	**INCS INCD**	**Address Offset**	**SMF DMF**	**INCS INCD**	**Address Offset**	**SMF DMF**	**INCS INCD**	**Address Offset**
000_B	0	-1	000_B	0	-2	000_B	0	-4
	1	+1		1	+2		1	+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: CHCR0x.CHDW = 11_B is reserved and should not be used.

11.2.5 Channel Address Registers

The Source Address Register contains the 32-bit source address. If a DMA channel 0x is active, SADR0x 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**

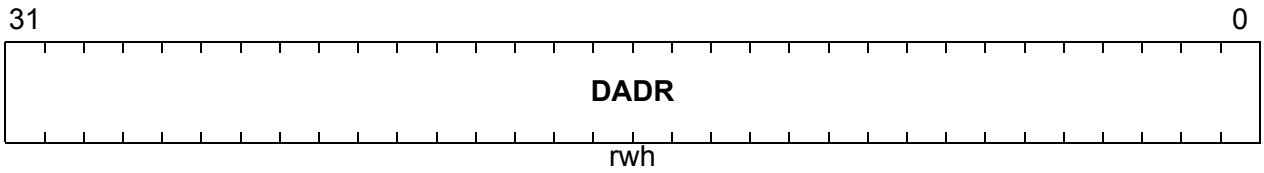


Field	Bits	Type	Description
SADR	[31:0]	rwh	**Source Start Address** This bit field holds the actual 32-bit source address of DMA channel 0x that is used for read moves.

A write to SADR0x is executed directly only when the DMA channel 0x is inactive (CHSR0x.TCOUNT = 0 and TRSR.CH0x = 0). If DMA channel 0x is active when writing to SADR0x, the source address will not be written into SADR0x directly but will be buffered in the shadow register SHADR0x until the start of the next DMA transaction. During this shadowed address register operation, bit field ADRCR0x.SHCT must be set to 01_B.

The Destination Address Register contains the 32-bit destination address. If a DMA channel is active, DADR0x 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**

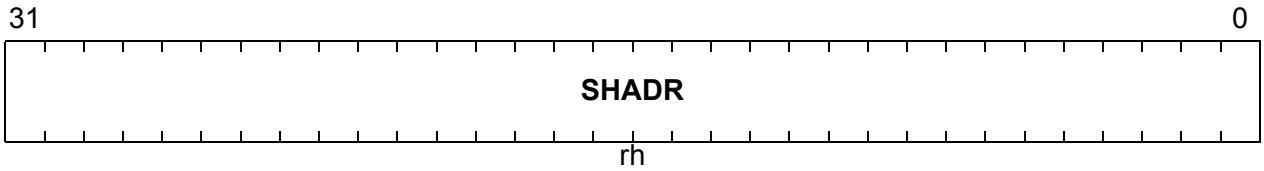


Field	Bits	Type	Description
DADR	[31:0]	rwh	**Destination Address** This bit field holds the actual 32-bit destination address of DMA channel 0x that is used for write moves.

A write to DADR0x is executed directly only when the DMA channel 0x is inactive (CHSR0x.TCOUNT = 0 and TRSR.CH0x = 0). If DMA channel 0x is active when writing to DADR0x, the source address will not be written into DADR0x directly but will be buffered in the shadow register SHADR0x until the start of the next DMA transaction. During this shadowed address register operation, bit field ADRCR0x.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. SHADR0x 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**



Field	Bits	Type	Description
SHADR	[31:0]	rh	**Shadowed Address** This bit field holds the shadowed 32-bit source or destination address of DMA channel 0x.

SHADR0x is written when source or destination address buffering is selected (ADRCR0x.SHCT = 01_B or ADRCR0x.SHCT = 10_B) and a transaction is running. While the shadow mechanism is disabled, SHADR is set to 0000 0000_H.

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.

11.3 DMA Module Implementation

This section describes the TC1766 DMA module interfaces with the clock control, interrupt control, and address decoding.

Figure 11-30 shows the TC1766-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.

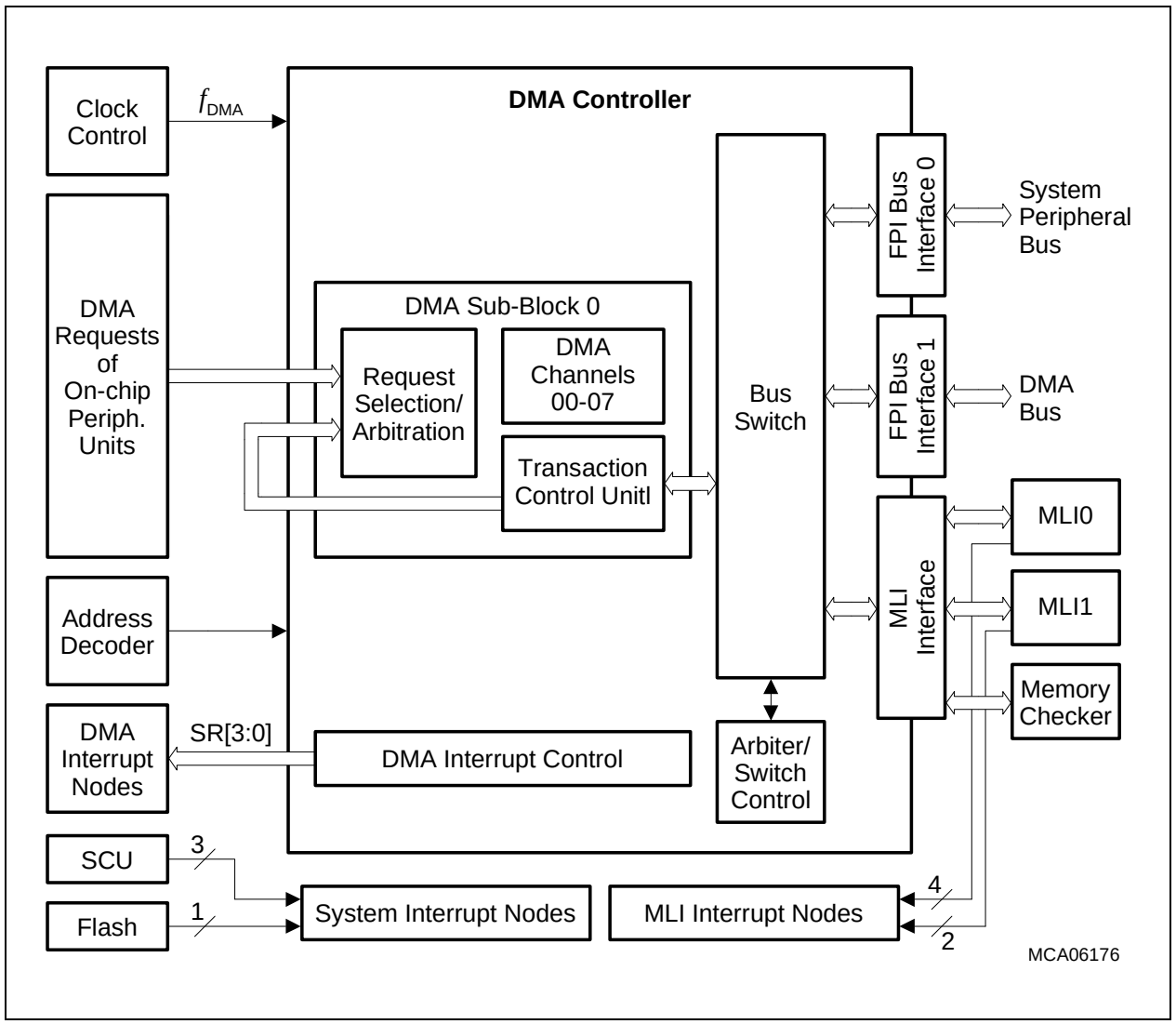


Figure 11-30 DMA Module Implementation and Interconnections

The request sources of the peripheral modules (ADC0, MSC0, MLI0/1, 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.

11.3.1 DMA Request Wiring Matrix

The DMA request input lines of each DMA channel within DMA Sub-Block 0 are connected to request output lines from the peripheral modules according to **Table 11-9**.

Table 11-9 DMA Request Assignment for DMA Sub-Block 0

DMA Channel	DMA Request Line	DMA Requesting Unit	Selected by
00	CH07_OUT	DMA channel 07	CHCR00.PRSEL = 000_B
	SCU_IOUT0	Ext. Request Unit[1)]	CHCR00.PRSEL = 001_B
	FADC_INT_02	FADC	CHCR00.PRSEL = 010_B
	ADC_INT_04	ADC	CHCR00.PRSEL = 011_B
	SSC0_REQ	SSC0	CHCR00.PRSEL = 100_B
	ASC0_REQ	ASC0	CHCR00.PRSEL = 101_B
	MLI0_INT_04	MLI0	CHCR00.PRSEL = 110_B
	MLI1_INT_04	MLI1	CHCR00.PRSEL = 111_B
01	CH00_OUT	DMA channel 00	CHCR01.PRSEL = 000_B
	DMA_SR12	DMA(INT_O12)	CHCR01.PRSEL = 001_B
	FADC_INT_03	FADC	CHCR01.PRSEL = 010_B
	ADC_INT_05	ADC	CHCR01.PRSEL = 011_B
	SSC1_REQ	SSC1	CHCR01.PRSEL = 100_B
	ASC1_REQ	ASC1	CHCR01.PRSEL = 101_B
	MLI0_INT_05	MLI0	CHCR01.PRSEL = 110_B
	MLI1_INT_05	MLI1	CHCR01.PRSEL = 111_B
02	CH01_OUT	DMA channel 01	CHCR02.PRSEL = 000_B
	SCU_IOUT1	Ext. Request Unit[1)]	CHCR02.PRSEL = 001_B
	FADC_INT_02	FADC	CHCR02.PRSEL = 010_B
	ADC_INT_06	ADC	CHCR02.PRSEL = 011_B
	SCU_ROUT0	Ext. Request Unit[1)]	CHCR02.PRSEL = 100_B
	SSC0_REQ	SSC0	CHCR02.PRSEL = 101_B
	MLI0_INT_06	MLI0	CHCR02.PRSEL = 110_B
	MLI1_INT_06	MLI1	CHCR02.PRSEL = 111_B

Table 11-9 DMA Request Assignment for DMA Sub-Block 0 (cont'd)

DMA Channel	DMA Request Line	DMA Requesting Unit	Selected by
03	CH02_OUT	DMA channel 02	CHCR03.PRSEL = 000_B
	DMA_SR13	DMA (INT_O13)	CHCR03.PRSEL = 001_B
	FADC_INT_03	FADC	CHCR03.PRSEL = 010_B
	ADC_INT_07	ADC	CHCR03.PRSEL = 011_B
	SCU_ROUT1	Ext. Request Unit[1)]	CHCR03.PRSEL = 100_B
	MSC_INT_03	MSC	CHCR03.PRSEL = 101_B
	MLI0_INT_07	MLI0	CHCR03.PRSEL = 110_B
	MLI1_INT_07	MLI1	CHCR03.PRSEL = 111_B
04	CH03_OUT	DMA channel 03	CHCR04.PRSEL = 000_B
	SCU_IOUT2	Ext. Request Unit[1)]	CHCR04.PRSEL = 001_B
	SCU_ROUT0	Ext. Request Unit[1)]	CHCR04.PRSEL = 010_B
	MSC_INT_02	MSC	CHCR04.PRSEL = 011_B
	SSC0_REQ	SSC0	CHCR04.PRSEL = 100_B
	ASC0_REQ	ASC0	CHCR04.PRSEL = 101_B
	MLI0_INT_04	MLI0	CHCR04.PRSEL = 110_B
	MLI1_INT_04	MLI1	CHCR04.PRSEL = 111_B
05	CH04_OUT	DMA channel 04	CHCR05.PRSEL = 000_B
	DMA_SR14	DMA (INT_O14)	CHCR05.PRSEL = 001_B
	SCU_ROUT1	Ext. Request Unit[1)]	CHCR05.PRSEL = 010_B
	MSC_INT_03	MSC	CHCR05.PRSEL = 011_B
	SSC1_REQ	SSC1	CHCR05.PRSEL = 100_B
	ASC1_REQ	ASC1	CHCR05.PRSEL = 101_B
	MLI0_INT_05	MLI0	CHCR05.PRSEL = 110_B
	MLI1_INT_05	MLI1	CHCR05.PRSEL = 111_B

Table 11-9 DMA Request Assignment for DMA Sub-Block 0 (cont'd)

DMA Channel	DMA Request Line	DMA Requesting Unit	Selected by
06	CH05_OUT	DMA channel 05	CHCR06.PRSEL = 000_B
	SCU_IOUT3	Ext. Request Unit[1)]	CHCR06.PRSEL = 001_B
	SCU_ROUT2	Ext. Request Unit[1)]	CHCR06.PRSEL = 010_B
	CAN_INT_O0	MultiCAN	CHCR06.PRSEL = 011_B
	SSC0_REQ	SSC0	CHCR06.PRSEL = 100_B
	ASC0_REQ	ASC0	CHCR06.PRSEL = 101_B
	MLI0_INT_06	MLI0	CHCR06.PRSEL = 110_B
	MLI1_INT_06	MLI1	CHCR06.PRSEL = 111_B
07	CH06_OUT	DMA channel 06	CHCR07.PRSEL = 000_B
	DMA_SR15	DMA(INT_O15)	CHCR07.PRSEL = 001_B
	CAN_INT_O1	MultiCAN	CHCR07.PRSEL = 010_B
	ASC1_REQ	ASC1	CHCR07.PRSEL = 011_B
	SSC0_REQ	SSC0	CHCR07.PRSEL = 100_B
	SSC1_REQ	SSC1	CHCR07.PRSEL = 101_B
	MLI0_INT_07	MLI0	CHCR07.PRSEL = 110_B
	MLI1_INT_07	MLI1	CHCR07.PRSEL = 111_B

1) The external request unit located in the System Control Unit handles DMA requests coming from GPTA and external inputs.

11.3.2 Access Protection Assignment

DMA access protection as described on **Page 11-41** requires the assignment of 32 fixed address range. **Table 11-10** shows this address range assignment as implemented in the TC1766.

Table 11-10 DMA Access Protection Address Ranges

Access Protection Range			Related Module(s)
No. n	**Enable Bit in MEmAENR**	**Selected Address Range**	
0	AEN0	$F000\ 0000_H$ - $F000\ 00FF_H$ $F010\ C200_H$ - $F010\ C2FF_H$	SCU, incl. WDT 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 $F000\ 08FF_H$	MSC0
5	AEN5	$F000\ 0A00_H$ to $F000\ 0AFF_H$	ASC0
6	AEN6	$F000\ 0B00_H$ to $F000\ 0BFF_H$	ASC1
7	AEN7	$F000\ 0C00_H$ - $F000\ 0FFF_H$	P0, P1, P2, P3
8	AEN8	$F000\ 1000_H$ - $F000\ 11FF_H$	P4, P5
9	AEN9	$F000\ 1800_H$ - $F000\ 1FFF_H$	GPTA
10	AEN10	$F000\ 3C00_H$ - $F000\ 3EFF_H$	DMA
11	AEN11	$F000\ 4000_H$ - $F000\ 5FFF_H$	MultiCAN
12	AEN12	$F004\ 0000_H$ - $F005\ FFFF_H$	PCP Registers, PCP Data Memory
13	AEN13	$F006\ 0000_H$ - $F006\ 7FFF_H$	PCP Code Memory
14	AEN14	$F010\ 0100_H$ - $F010\ 01FF_H$	SSC0
15	AEN15	$F010\ 0200_H$ - $F010\ 02FF_H$	SSC1
16	AEN16	$F010\ 0300_H$ - $F010\ 03FF_H$	FADC
17	AEN17	$F010\ 0400_H$ - $F010\ 05FF_H$	ADC0
18	AEN18	–	–
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 11-10 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$_H$	MLI1, MLI1 Small TWs, MLI1 Large TWs
21	AEN21	F7E0 FF00$_H$ - F7E0 FFFF$_H$ F7E1 0000$_H$ - F7E1 FFFF$_H$ F800 0500$_H$ - F87F FFFF$_H$	CPS, CPU SFRs & GPRs PMU, Flash Regs, LBCU, DMI, PMI, LFI
22	AEN22	–	–
23	AEN23	8000 0000$_H$ - 807F FFFF$_H$ A000 0000$_H$ - A07F FFFF$_H$	Program Flash Space
24	AEN24	–	–
25	AEN25	8FE0 0000$_H$ - 8FEF FFFF$_H$ AFE0 0000$_H$ - AFEF 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	–	–
29	AEN29	E800 0000$_H$ - E83F FFFF$_H$	LMU Image (E80x translated to C00x)
30	AEN30	E840 0000$_H$ - E84F FFFF$_H$	DMI Image (E84x translated to D00x)
31	AEN31	E850 0000$_H$ - E85F FFFF$_H$	PMI Image (E85x translated to D40x)

In the TC1766, two internal memory areas are assigned for access protection using programmable address sub-ranges:

- 64-Kbyte OVRAM, assigned as address range 29. The sub-ranges are controlled by bit fields ME0ARR.SIZE1 and ME0ARR.SLICE1.
- 64-Kbyte DMI RAM, assigned as address range 30. The sub-ranges are controlled by bit fields ME0ARR.SIZE2 and ME0ARR.SLICE2.

Bit fields SIZE0 and SLICE0 for the OVRAM sub-range access protection (with address translation from E800 to C000 in the LFI) are coded as shown in **Table 11-11**.

Table 11-11 OVRAM Address Protection Sub-Range Definitions

SIZE1	Sub-Ranges	SLICE1	Selected Address Range
000_B	32 sub-ranges of 512 bytes	00000_B 00001_B … 11111_B	$E800\ 0000_H$ - $E800\ 01FF_H$ $E800\ 0200_H$ - $E800\ 03FF_H$ … $E800\ 3E00_H$ - $E800\ 3FFF_H$ $E800\ 4000_H$ - $E800\ FFFF_H$ is not selectable
001_B	32 sub-ranges of 1 Kbyte	00000_B 00001_B … 11111_B	$E800\ 0000_H$ - $E800\ 03FF_H$ $E800\ 0400_H$ - $E800\ 07FF_H$ … $E800\ 7C00_H$ - $E800\ 7FFF_H$ $E800\ 8000_H$ - $E800\ FFFF_H$ is not selectable
010_B	32 sub-ranges of 2 Kbytes	00000_B 00001_B … 11111_B	$E800\ 0000_H$ - $E800\ 07FF_H$ $E800\ 0800_H$ - $E800\ 0FFF_H$ … $E800\ F800_H$ - $E800\ FFFF_H$
011_B	16 sub-ranges of 4 Kbytes	$X0000_B$ $X0001_B$ … $X1111_B$	$E800\ 0000_H$ - $E800\ 0FFF_H$ $E800\ 1000_H$ - $E800\ 1FFF_H$ … $E800\ F000_H$ - $E800\ FFFF_H$
100_B	8 sub-ranges of 8 Kbytes	$XX000_B$ $XX001_B$ … $XX111_B$	$E800\ 0000_H$ - $E800\ 1FFF_H$ $E800\ 2000_H$ - $E800\ 3FFF_H$ … $E800\ E000_H$ - $E800\ FFFF_H$
101_B	4 sub-ranges of 16 Kbytes	$XXX00_B$ $XXX01_B$ $XXX10_B$ $XXX11_B$	$E800\ 0000_H$ - $E800\ 3FFF_H$ $E800\ 4000_H$ - $E800\ 7FFF_H$ $E800\ 8000_H$ - $E800\ BFFF_H$ $E800\ C000_H$ - $E800\ FFFF_H$

Table 11-11 OVRAM Address Protection Sub-Range Definitions (cont'd)

SIZE1	Sub-Ranges	SLICE1	Selected Address Range
110_B	2 sub-ranges of 32 Kbytes	$XXXX0_B$ $XXXX1_B$	$E800\ 0000_H$ - $E800\ 7FFF_H$ $E800\ 8000_H$ - $E800\ FFFF_H$
111_B	64 Kbytes	$XXXXX_B$	$E800\ 0000_H$ - $E800\ FFFF_H$

Bit fields SIZE2 and SLICE2 for the DMI RAM sub-range access protection (with address translation from E840 to D000 in the LFI) are coded as shown in **Table 11-12**.

Table 11-12 DMI RAM Address Protection Sub-Range Definitions

SIZE2	Sub-Ranges	SLICE2	Selected Address Range
000_B	32 sub-ranges of 512 bytes	00000_B 00001_B … 11111_B	$E840\ 0000_H$ - $E840\ 01FF_H$ $E840\ 0200_H$ - $E840\ 03FF_H$ … $E840\ 3E00_H$ - $E840\ 3FFF_H$ $E840\ 4000_H$ - $E840\ FFFF_H$ is not selectable
001_B	32 sub-ranges of 1 Kbyte	00000_B 00001_B … 11111_B	$E840\ 0000_H$ - $E840\ 03FF_H$ $E840\ 0400_H$ - $E840\ 07FF_H$ … $E840\ 7C00_H$ - $E840\ 7FFF_H$ $E840\ 8000_H$ - $E840\ FFFF_H$ is not selectable
010_B	32 sub-ranges of 2 Kbytes	00000_B 00001_B … 11111_B	$E840\ 0000_H$ - $E840\ 07FF_H$ $E840\ 0800_H$ - $E840\ 0FFF_H$ … $E840\ F800_H$ - $E840\ FFFF_H$
011_B	16 sub-ranges of 4 Kbytes	$X0000_B$ $X0001_B$ … $X1111_B$	$E840\ 0000_H$ - $E840\ 0FFF_H$ $E840\ 1000_H$ - $E840\ 1FFF_H$ … $E840\ F000_H$ - $E840\ FFFF_H$
100_B	8 sub-ranges of 8 Kbytes	$XX000_B$ $XX001_B$ … $XX111_B$	$E840\ 0000_H$ - $E840\ 1FFF_H$ $E840\ 2000_H$ - $E840\ 3FFF_H$ … $E840\ E000_H$ - $E840\ FFFF_H$
101_B	4 sub-ranges of 16 Kbytes	$XXX00_B$ $XXX01_B$ $XXX10_B$ $XXX11_B$	$E840\ 0000_H$ - $E840\ 3FFF_H$ $E840\ 4000_H$ - $E840\ 7FFF_H$ $E840\ 8000_H$ - $E840\ BFFF_H$ $E840\ C000_H$ - $E840\ FFFF_H$

Table 11-12 DMI RAM Address Protection Sub-Range Definitions (cont'd)

SIZE2	Sub-Ranges	SLICE2	Selected Address Range
110_B	2 sub-ranges of 32 Kbytes	$XXXX0_B$ $XXXX1_B$	$E840\ 0000_H$ - $E840\ 7FFF_H$ $E840\ 8000_H$ - $E840\ FFFF_H$
111_B	64 Kbytes	$XXXXX_B$	$E840\ 0000_H$ - $E840\ FFFF_H$

Note: Sub-ranges 0 and 4 with bit fields ME0ARR.SIZE0, ME0ARR.SLICE0, ME0ARR.SIZE4, and ME0ARR.SLICE4 are not implemented.

11.3.3 Implementation-specific DMA Registers

The DMA controller as implemented in the TC1766 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_MLIySRC.x)
- Service request control registers for system related interrupts (DMA_SYSSRCx)

Figure 11-31 provides an overview of these registers.

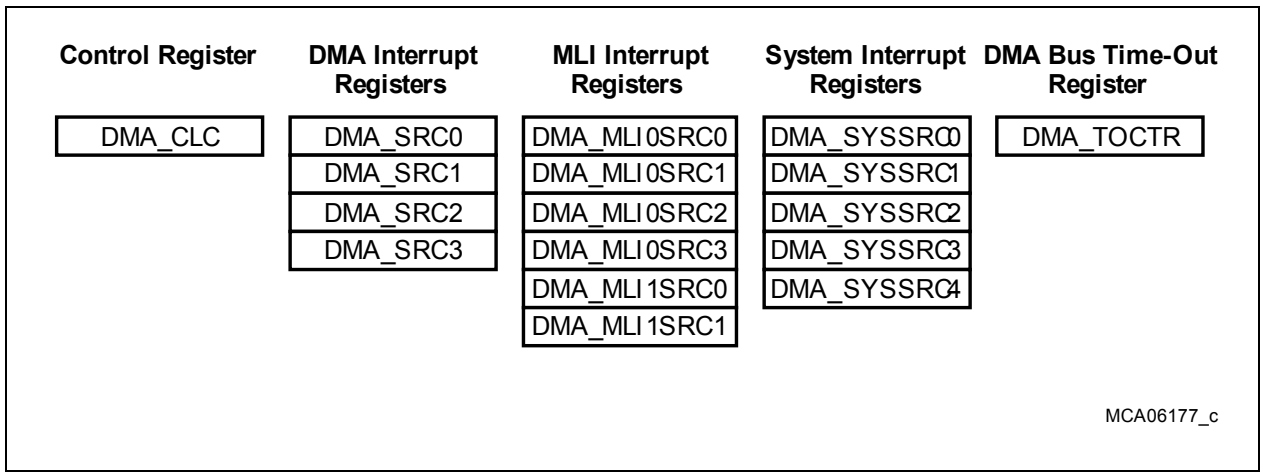


Figure 11-31 DMA Implementation-specific Registers

Note: Further details on interrupt handling and processing are described in the "Interrupt System" chapter of the TC1766 System Units User's Manual.

The clock generation and interrupt control configuration as implemented in the DMA controller module is shown in **Figure 11-32**.

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} and is controlled via the DMA_CLC clock control register. The MLI modules 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, except for the hard-suspend feature of the DMA (clock switch-off). This hard-suspend feature should only be used if the system is reset after the suspend state is left. In Hard-suspend Mode, the FPI Bus interface 1 (DMA) cannot be accessed, because the DMA Bus Switch is no longer clocked.

The DMA controller module contains in total 14 interrupt request nodes with its interrupt service request control registers:

- Four interrupt requests SR[3:0] = INT_O[3:0] from the DMA controller; upper twelve interrupt requests of the DMA controller INT_O[15:4] are not connected.
- 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.
- Two interrupt requests SR[1:0] = INT_O[1:0] from the MLI1 module; upper six interrupt requests of the MLI1 module INT_O[7:2] are not connected.
- Four system interrupts SR[3:0] from the SCU.

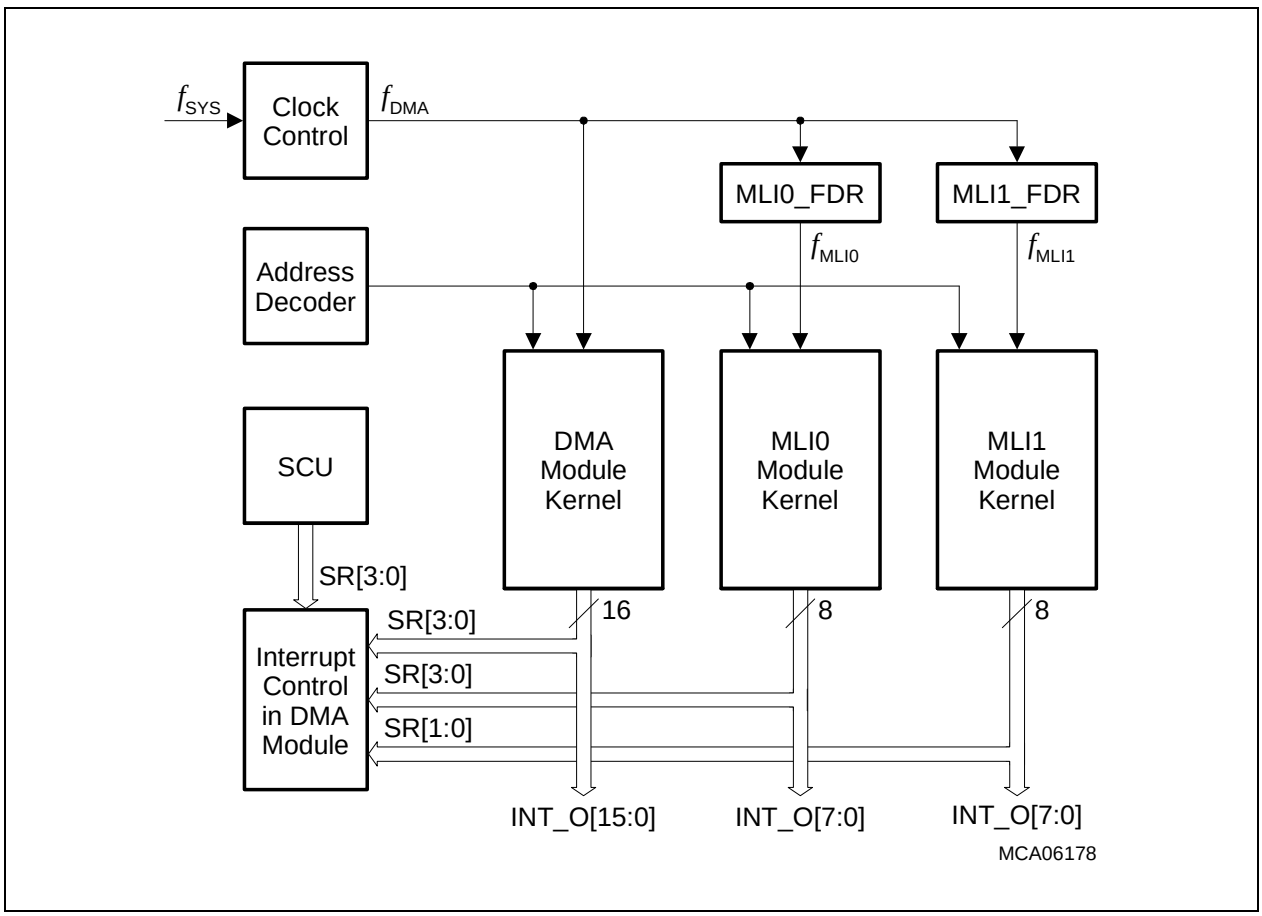
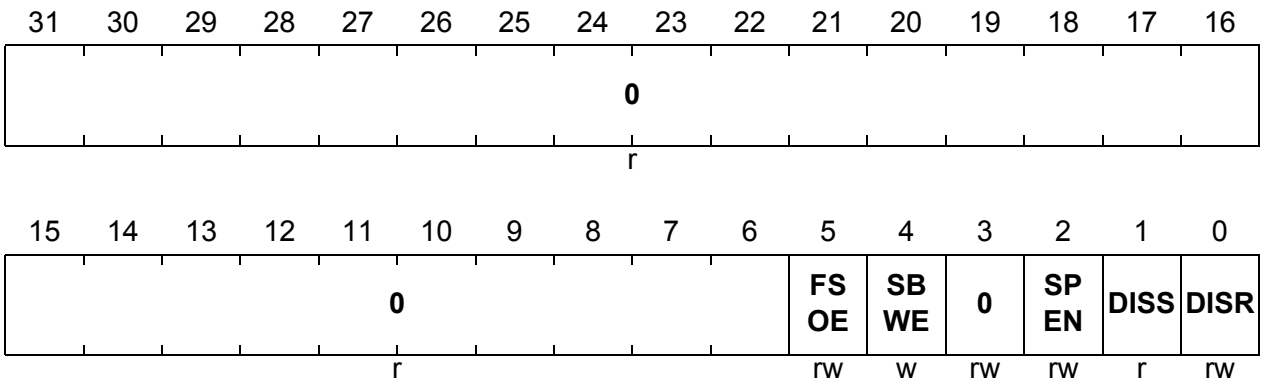


Figure 11-32 Implementation of the DMA Module and the MLI Modules

11.3.3.1 Clock Control Register

The Clock Control Register controls the f_{DMA} module 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 0000$_H$



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
0	3	rw	**Reserved;** returns 0 if read; must be written with 0.
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.
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.

11.3.3.2 DMA Interrupt Registers

In the TC1766, only the lower four DMA controller interrupts SR[3:0] are connected to service request control registers. The upper twelve DMA controller interrupt outputs SR[15:4] are not used and are not connected.

DMA_SRCx (x = 0-3)
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**
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.

11.3.3.3 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. They share one FPI Bus 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_MLI1SRCx (x = 0-1)
DMA MLI1 Service Request Control Register x
($2BC_H$-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**
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.

11.3.3.4 System Interrupt Registers

System interrupts of other on-chip modules are controlled by five system service request control registers which are located in the DMA address space. The five system service request registers are used according to **Table 11-13**.

System Interrupt Nodes

Table 11-13 Assignment of System Interrupt Nodes

System Service Request Node	Connected to System Interrupt Source
DMA_SYSSCR0	Floating Point Unit (FPU) interrupt (see also **Page 5-36**)
DMA_SYSSCR1	Flash module interrupt (see also **Page 7-36**)
DMA_SYSSCR2	External Interrupt 0 (see also **Page 5-37**)
DMA_SYSSCR3	External Interrupt 1 (see also **Page 5-37**)
DMA_SYSSCR4	DMA Bus Interrupt

DMA_SYSSRCx (x = 0-4)
DMA System Interrupt Service Request Control Register x
($29C_H$-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**
SRE	12	rw	**Service Request Enable**
SRR	13	rh	**Service Request Flag**
CLRR	14	w	**Request Clear Bit**

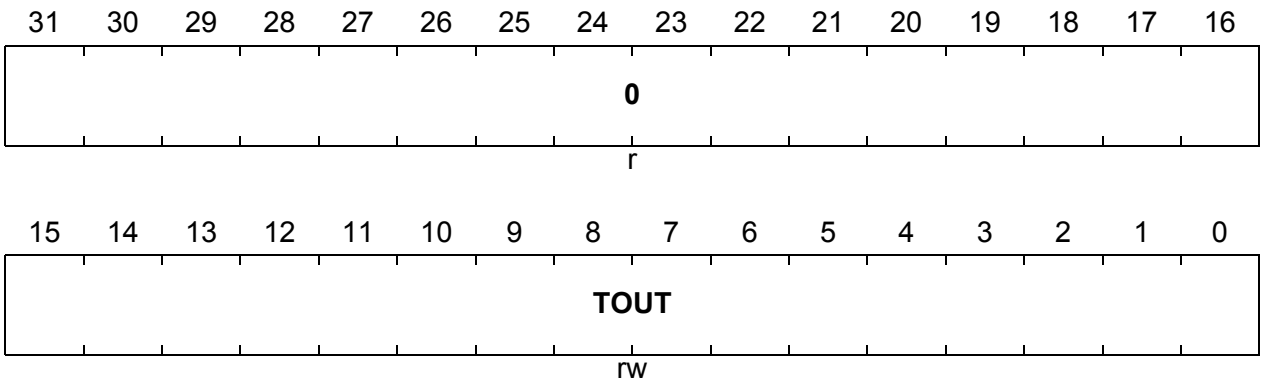
Field	Bits	Type	Description
SETR	15	w	**Request Set Bit**
0	[9:8], 11, [31:16]	r	**Reserved** Read as 0; should be written with 0.

Note: The bit coding of the system interrupt service request control registers is identical to that of the DMA service request control registers shown on ***Page 11-95****.*

11.3.3.5 DMA Bus Control Register

The TOCTR register contains the programmable time-out value for the DMA bus. In case of time out error, the system interrupt node DMA_SYSSRC4 will be flagged.

DMA_TOCTR
DMA Bus Time-Out Control Register (280_H) **Reset Value: 0000 FFFF$_H$**



Field	Bits	Type	Description
TOUT	[15:0]	rw	**Time-Out Value** This bit field defines the number of DMA Bus time-out cycles. Default after reset is FFFF$_H$ (= 65536 bus cycles).
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

11.3.4 Address Map

The DMA controller register block address map is shown in **Figure 11-33**. It shows how the different register blocks are arranged and adds the absolute address information. The complete address map of the DMA controller is described in **Table 16-16** on **Page 16-34**.

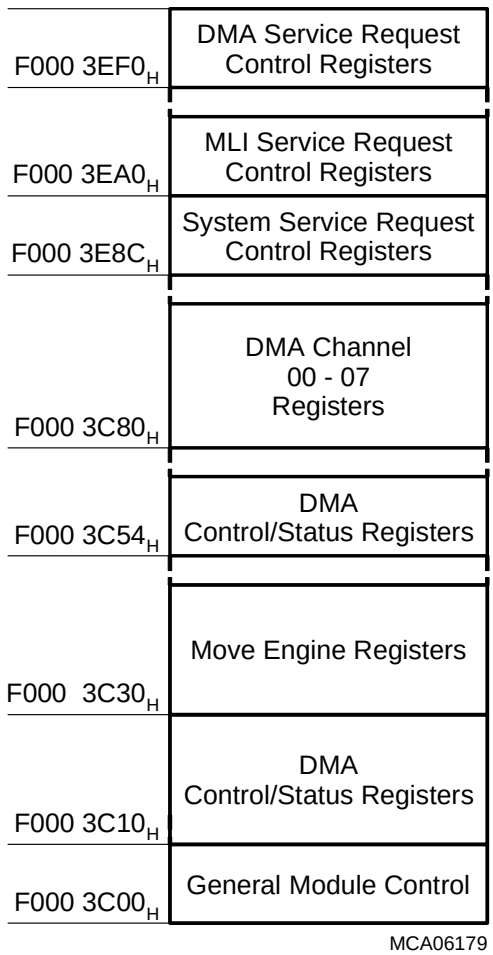


Figure 11-33 DMA Controller Register Block Address Map

11.4 Memory Checker Module

The Memory Checker Module (MCHK) makes it possible to check the data consistency of memories.

11.4.1 Functional Description

Any SPB bus master may access the memory checker; It is preferable that the DMA does it as described hereafter. It uses DMA 8-bit, 16-bit, or 32-bit moves to read from the selected address area and to write the value read in a 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 CHCR0x.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 (CHSR0x.TCOUNT = 0), an interrupt can be generated by the DMA channel (if CHCR0x.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.

11.4.2 Registers

This section describes the kernel registers of the Memory Checker module.

MCHK Register Overview

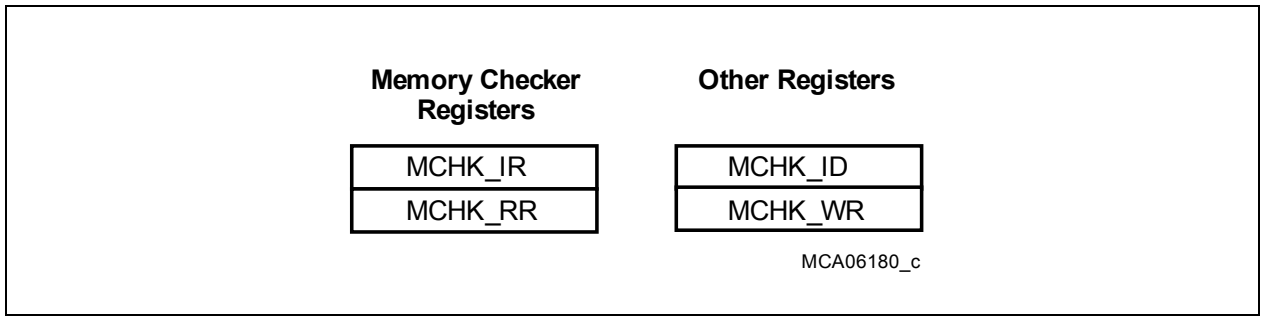


Figure 11-34 Memory Checker Registers

The complete and detailed address map of the of the ASC modules is described in **Table 16-23** on **Page 16-67** of the TC1766 User's Manual System Units part (Volume 1).

Table 11-14 Registers Address Space

Module	Base Address	End Address	Note
MCHK	F010 C200$_H$	F010 C2FF$_H$	-

Table 11-15 Registers Overview - Memory Checker egisters

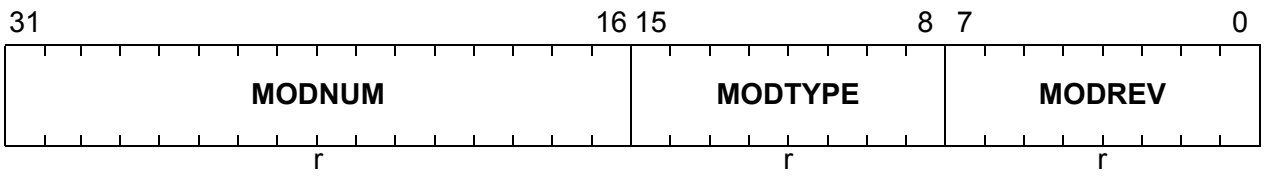
Register Short Name	Register Long Name	Offset Address[1)]	Description see
MCHK_ID	Memory Checker Module Identification Register	0008$_H$	**Page 11-102**
MCHK_IR	Memory Checker Input Register	0010$_H$	**Page 11-103**
MCHK_RR	Memory Checker Result Register	0014$_H$	**Page 11-103**
MCHK_WR	Memory Checker Write Register	0020$_H$	**Page 11-104**

1) The absolute register address is calculated as follows:
Module Base Address (**Table 11-5**) + Offset Address (shown in this column)

11.4.2.1 Memory Checker Registers

The MCHK Module Identification Register ID contains read-only information about the MCHK module version.

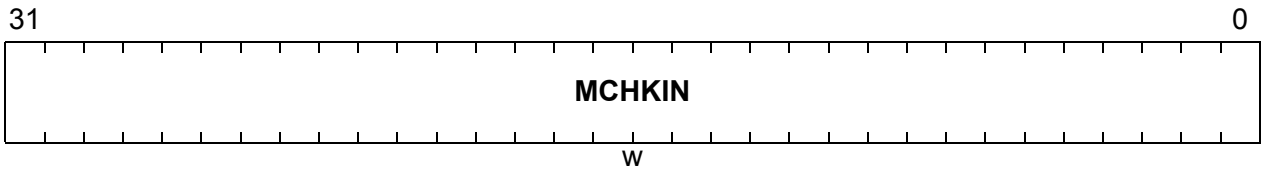
MCHK_ID
Memory Checker Module Identification Register
(008_H) **Reset Value: 001B C0XX$_H$**



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 MCHK: $001B_H$

The 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 register MCHK_IR 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 (10_H) Reset Value: 0000 0000_H

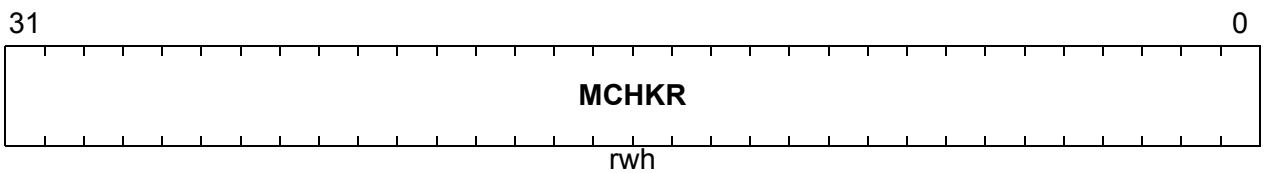


Field	Bits	Type	Description
MCHKIN	[31:0]	w	**Memory Checker Input** The value written to MCHKIN is used for the next checksum calculation. Any read action will deliver 0.

Note: MCHK_IR is a write-only register. Any read action will deliver 0000 0000_H.

The 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 (14_H) Reset Value: 0000 0000_H

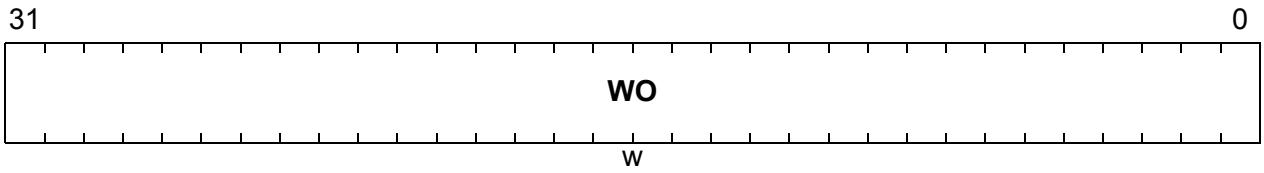


Field	Bits	Type	Description
MCHKR	[31:0]	rwh	**Memory Checker Result** This bit field contains the current result of the memory checksum calculation operation.

The Memory Checker Write Register is a dummy write-only register that is located within the memory checker address range. This register is nearby (above) the MLI address range of the TC1766. It has no meaning for memory checker operation.

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 TC1766, SPB and DMA, because it is near the MLI modules address ranges.

MCHK_WR
Memory Checker Write Register (20_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 Interrupt System

The TC1766 interrupt system provides a flexible and time-efficient means of processing interrupts. This chapter describes the interrupt system for the TC1766. Topics covered include the architecture of the interrupt system, interrupt system configuration, and the interrupt operations of the TC1766 peripherals and Central Processing Unit (CPU). General information is also given about the Peripheral Control Processor (PCP). For details about the PCP, see **Chapter 10**.

This chapter also discusses the Non-Maskable Interrupt (NMI) (see **Section 12.10** on **Page 12-25**).

12.1 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 TC1766 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 12-1**, each TC1766 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.

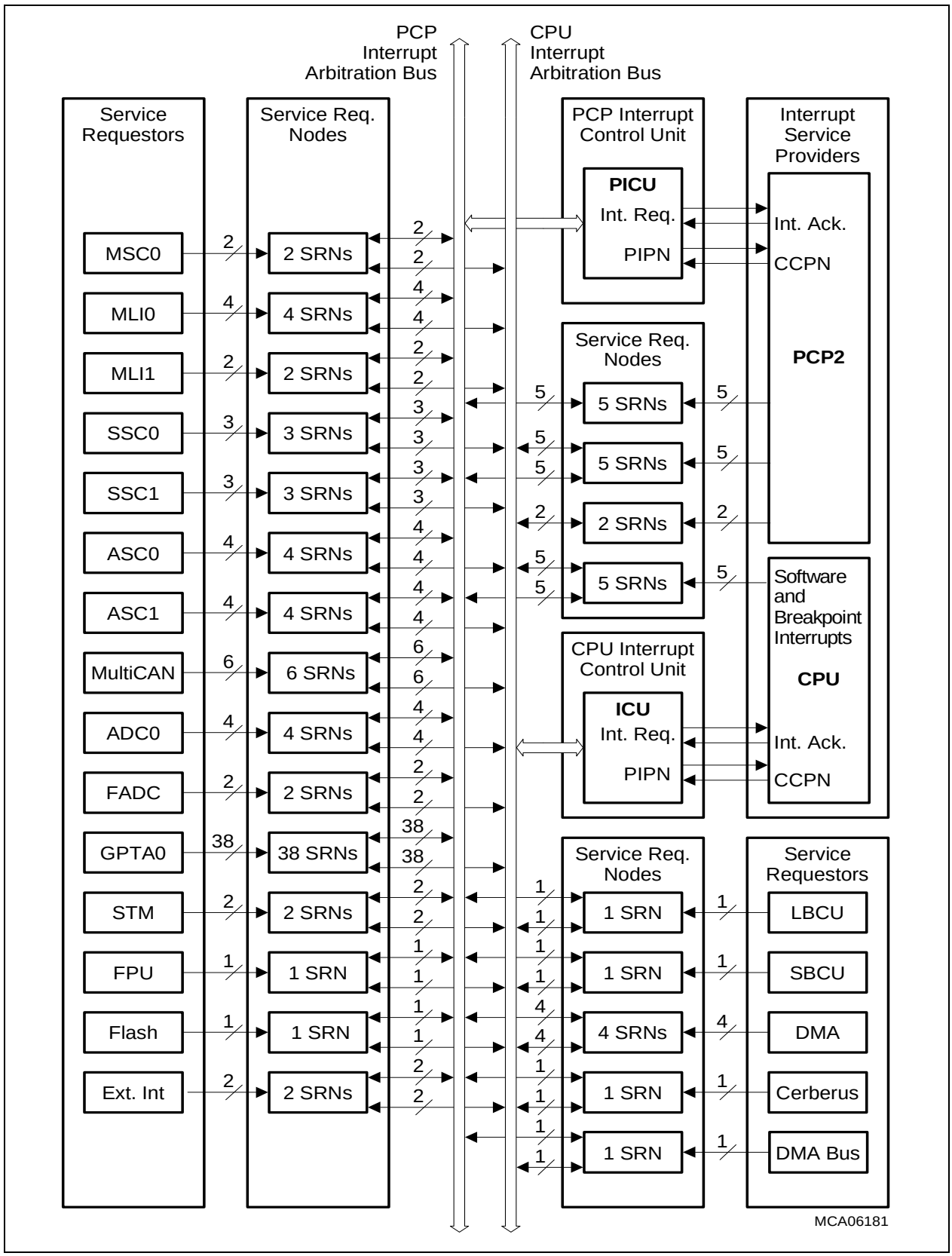


Figure 12-1 Block Diagram of the TC1766 Interrupt System

12.2 Service Request Nodes

In total, there are 181 Service Request Nodes available in the TC1766. 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 TC1766 have multiple SRNs.

12.2.1 Service Request Control Registers

All Service Request Control Registers in the TC1766 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 clear bits

Besides being activated by the associated triggering unit through hardware, each SRN can also be set or cleared by software via two software-initiated service request control bits.

The description given in this chapter characterizes all Service Request Control Registers of the TC1766. Information on peripheral module interrupt functions such as enable or request flags is provided in the corresponding sections of the module chapters.

12.2.1.1 General Service Request Control Register Format

The description given in this chapter characterizes all Service Request Control Registers of the TC1766. 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$_H$

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** 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
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.

12.2.1.2 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.

12.2.1.3 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 cleared 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.

12.2.1.4 Service Request Flag (SRR)

When set, the SRR flag indicates that a service request is pending. It can be set or cleared 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 cleared 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 cleared by hardware when the service request is acknowledged and serviced. Software can poll SRR to check for a pending service request. SRR must be cleared 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.

12.2.1.5 Type-Of-Service Control (TOS)

There are two service providers for service requests in the TC1766, 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).

12.2.1.6 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 TC1766 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 TC1766 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 TC1766

In the TC1766, 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 cleared 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.

12.3 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 TC1766 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 12-1**).

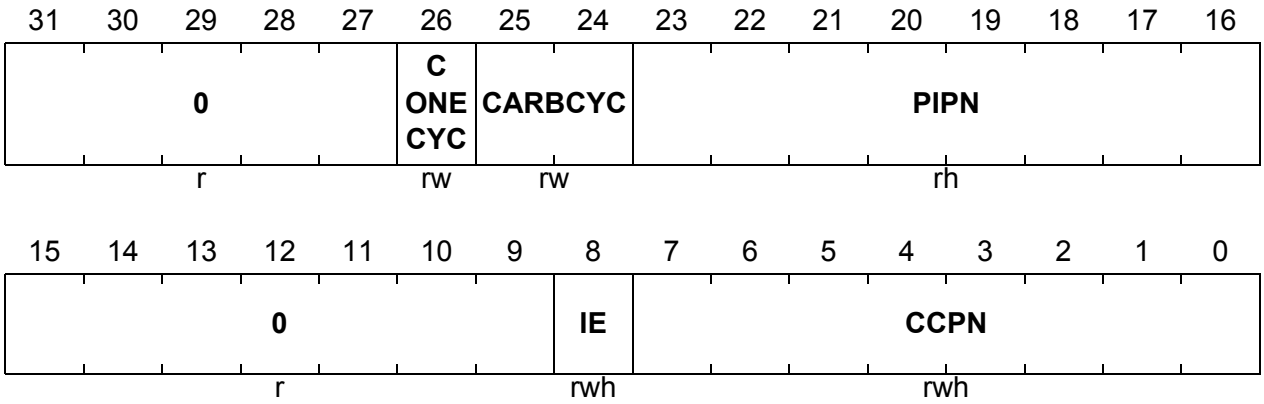
12.3.1 Interrupt Control Unit (ICU)

This section describes the interrupt control unit (ICU) for the CPU.

12.3.1.1 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_B Interrupt system is globally disabled 1_B 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_B 2 clocks per arbitration cycle (default) 1_B 1 clock per arbitration cycle
0	[15:9], [31:27]	r	**Reserved** Read as 0; should be written with 0.

12.3.1.2 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 cleared to 0.

After sending the acknowledgement, the ICU clears 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.

12.3.2 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. However, the PCP cannot handle nested interrupts; an interrupt request to the PCP cannot interrupt the service of another interrupt request. Thus, schemes such as interrupt priority grouping, are not feasible in the PCP.

Note: Details of the PCP_ICR register are described on ***Page 10-61****. The PICU is described on* ***Page 10-33****.*

12.4 Arbitration Process

The arbitration process implemented in the TC1766 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.

12.4.1 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 TC1766 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 12-1** gives the options for arbitration cycle control.

Table 12-1 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.

12.4.2 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 TC1766 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.

12.5 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.

12.6 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.

12.7 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 14**). 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 12-2**). 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).

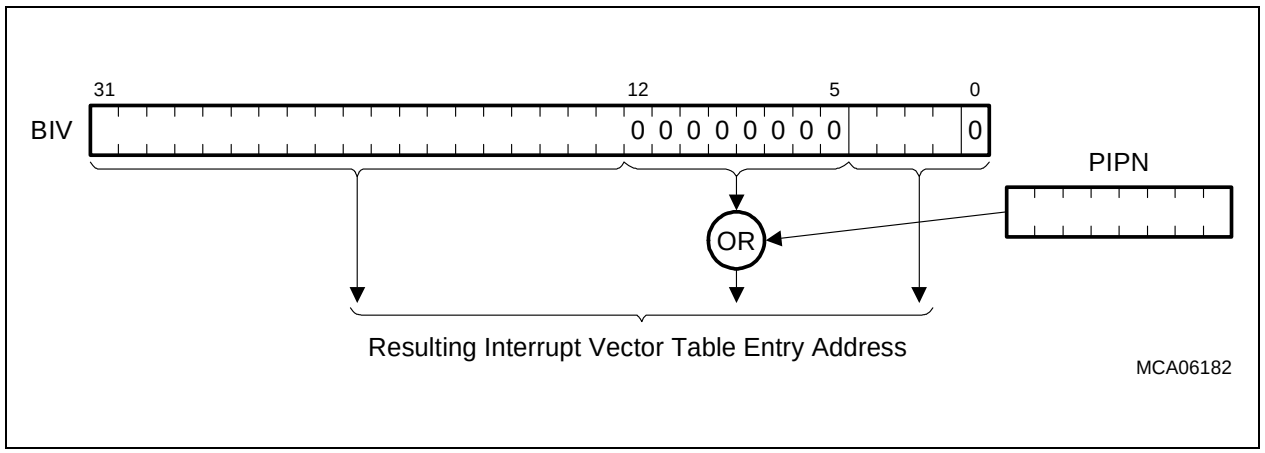


Figure 12-2 Interrupt Vector Table Entry Address Calculation

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 TC1766 offers an additional option by allowing spanning several Interrupt Vector Table entries as long as those entries are otherwise unused. **Figure 12-3** 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.

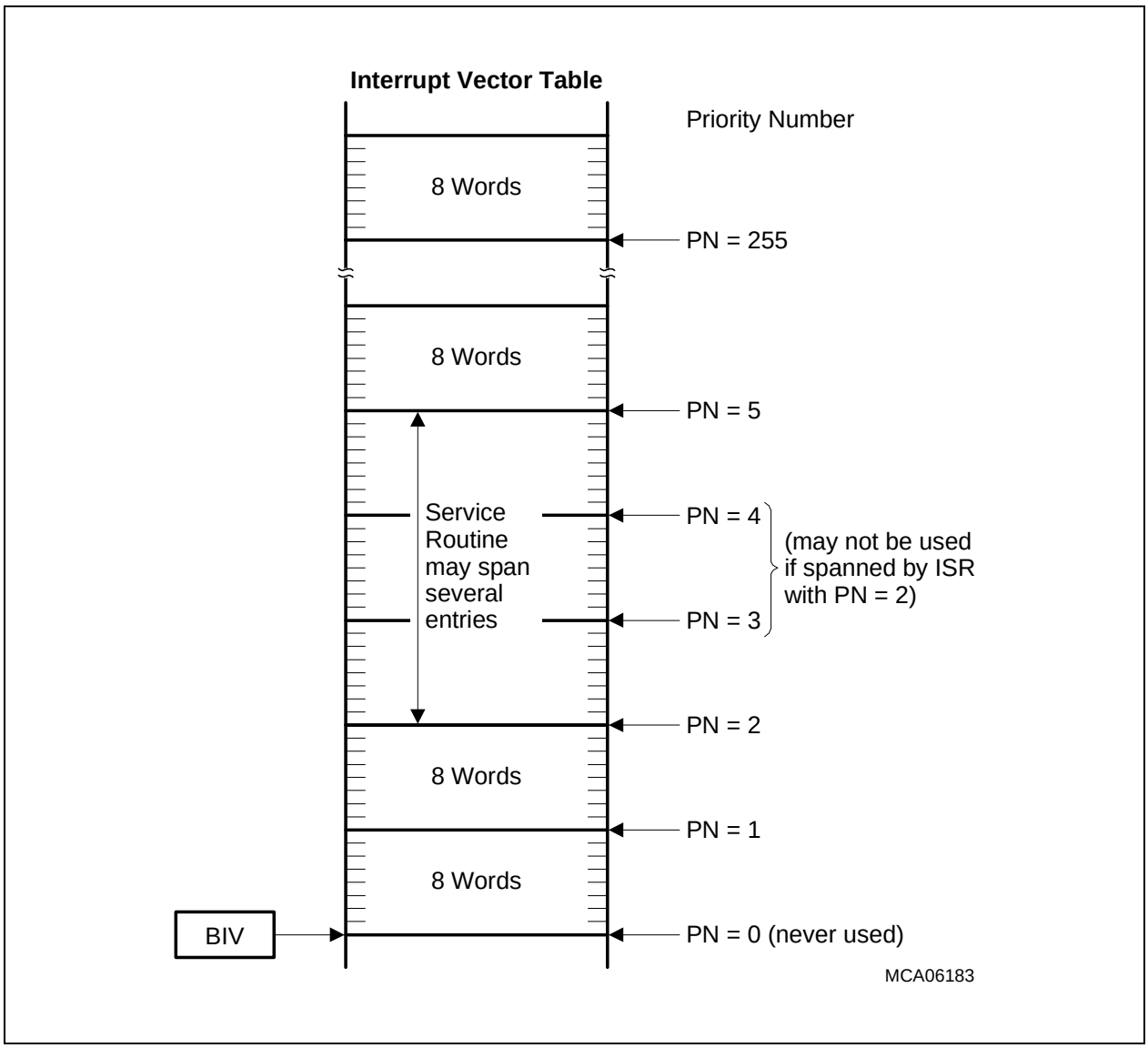


Figure 12-3 Interrupt Vector Table

12.8 Usage of the TC1766 Interrupt System

The following sections provide examples of using the TC1766 interrupt system to solve both typical and special application requirements.

12.8.1 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 12-3**, entry locations 3 and 4 are occupied by the ISR for entry 2. In **Figure 12-3**, 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 12-3**, 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.

12.8.2 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.

12.8.3 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 TC1766 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 12-4 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 12-4**, 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 TC1766 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.

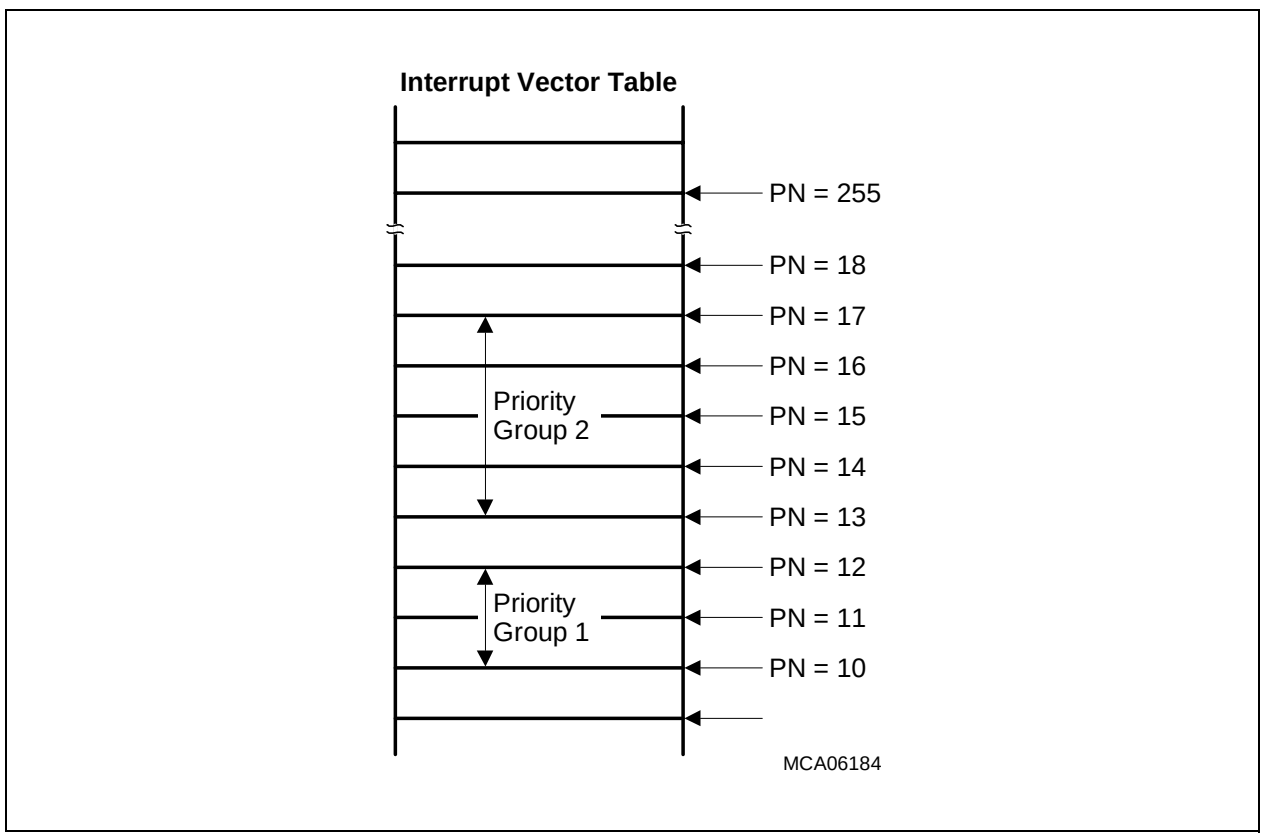


Figure 12-4 Interrupt Priority Groups

12.8.4 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.

12.8.5 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 TC1766 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 TC1766, 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.

12.8.6 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.

12.8.7 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 TC1766 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 **Page 2-14** for TC1766-specific implementation details of the four CPU Service Request Control Registers.

Additionally, any otherwise unused SRN can be employed to generate software interrupts.

12.8.8 External Interrupts

Two SRNs, DMA_SYSSRC2 and DMA_SYSSRC3, 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 **Section 5.3** on **Page 5-10**.

12.9 Service Request Node Table

Table 12-2 shows all TC1766 Service Request Nodes.

Table 12-2 Service Request Nodes in the TC1766

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	1	Cerberus/OCDS Request Node	CBS_SRC
LBCU	1	LBCU Request Node	LBCU_SRC1)
SBCU	1	SBCU Request Node	SBCU_SRC
DMA	4	DMA Service Request Nodes [3:0]	DMA_SRC[3:0]
	1	FPU Service Request Node	DMA_SYSSRC0
	1	FLASH Service Request Node	DMA_SYSSRC1
	2	External Interrupt Nodes [1:0]	DMA_SYSSRC2 DMA_SYSSRC3
	1	DMA Bus Error Interrupt Request Node	DMA_SYSSRC4
	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]
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 12-2 Service Request Nodes in the TC1766 (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]
CAN	6	CAN Service Request Nodes [5:0]	CAN_SRC[5:0][1)]
GPTA	38	GPTA0 Service Request Nodes [37:00]	GPTA0_SRC [37:00]
ADC0	4	ADC0 Service Request Nodes [3:0]	ADC0_SRC[3:0][1)]
FADC	2	FADC Service Request Nodes [1:0]	FADC_SRC[1:0][1)]
	181	= Total Number of Request Nodes	

1) These service request registers are not bit-addressable because its register address is outside the first 16 Kbyte of a segment.

12.10 Non-Maskable Interrupt

Although called an interrupt, the Non-Maskable Interrupt (NMI) is actually serviced as a trap, since it is not interruptible and does not follow the standards for regular interrupts.

In the TC1766, four different events can generate an NMI trap:

- A transition on the $\overline{\text{NMI}}$ input pin
- An error from the Watchdog Timer
- The PLL becomes unlocked
- The occurrence of an SRAM parity error in an on-chip memory block

The type of NMI trap is indicated in the NMI Status Register (NMISR).

12.10.1 External $\overline{\text{NMI}}$ Input

An external $\overline{\text{NMI}}$ event is generated when a one-to-zero transition is detected at the external $\overline{\text{NMI}}$ input pin. NMISR.NMIEXT is set in this case. The $\overline{\text{NMI}}$ pin is sampled at the system clock frequency. A transition is recognized when one sample shows a 1 and the next sample shows a 0. Subsequent 0-samples or a 0-to-1 transition do not trigger any action.

NMI is equipped with a noise suppression filter which suppresses glitches below 10 ns pulse width. NMI pulses with a width above 100 ns are safely recognized as a valid signal. The noise suppression filter is switched-off when pin BYPASS = 1.

12.10.2 Phase-Locked Loop NMI

The PLL clock generation unit sets the NMIPLL flag when it detects a loss in the synchronization with the external oscillator clock input. This condition means that the PLL clock frequency is no longer stable, and that the PLL will now decrease to its VCO clock base frequency.

12.10.3 Watchdog Timer NMI

The Watchdog Timer sets the NMIWDT flag when a Watchdog Timer error has occurred.

A Watchdog Timer error can produce an NMI event because

- Access to register WDT_CON0 was attempted improperly, or
- The Watchdog Timer overflowed either in Time-Out Mode or in Normal Watchdog Timer Mode.

12.10.4 SRAM Parity Error NMI

If an SRAM parity error is detected in an SRAM memory block that is enabled for parity error detection (corresponding enable bit SCU_PETCR.PENx is set), the NMIPER flag in register NMISR is set together with related parity error flag SCU_PETSR.PFLx. In contrast to the other NMI flags, the NMIPER flag remains set when reading register NMISR. NMIPER is cleared only with an NMISR read operation when no error flag PFLx is set during the NMISR read operation. This parity error flag mechanism has some effects on an NMI trap handling routine. **Figure 12-5** shows a trap handler flow diagram that especially handles a typical SRAM parity error NMI trap recognition.

Additional details about SRAM parity error control are described in section **"SRAM Parity Control" on Page 5-38**.

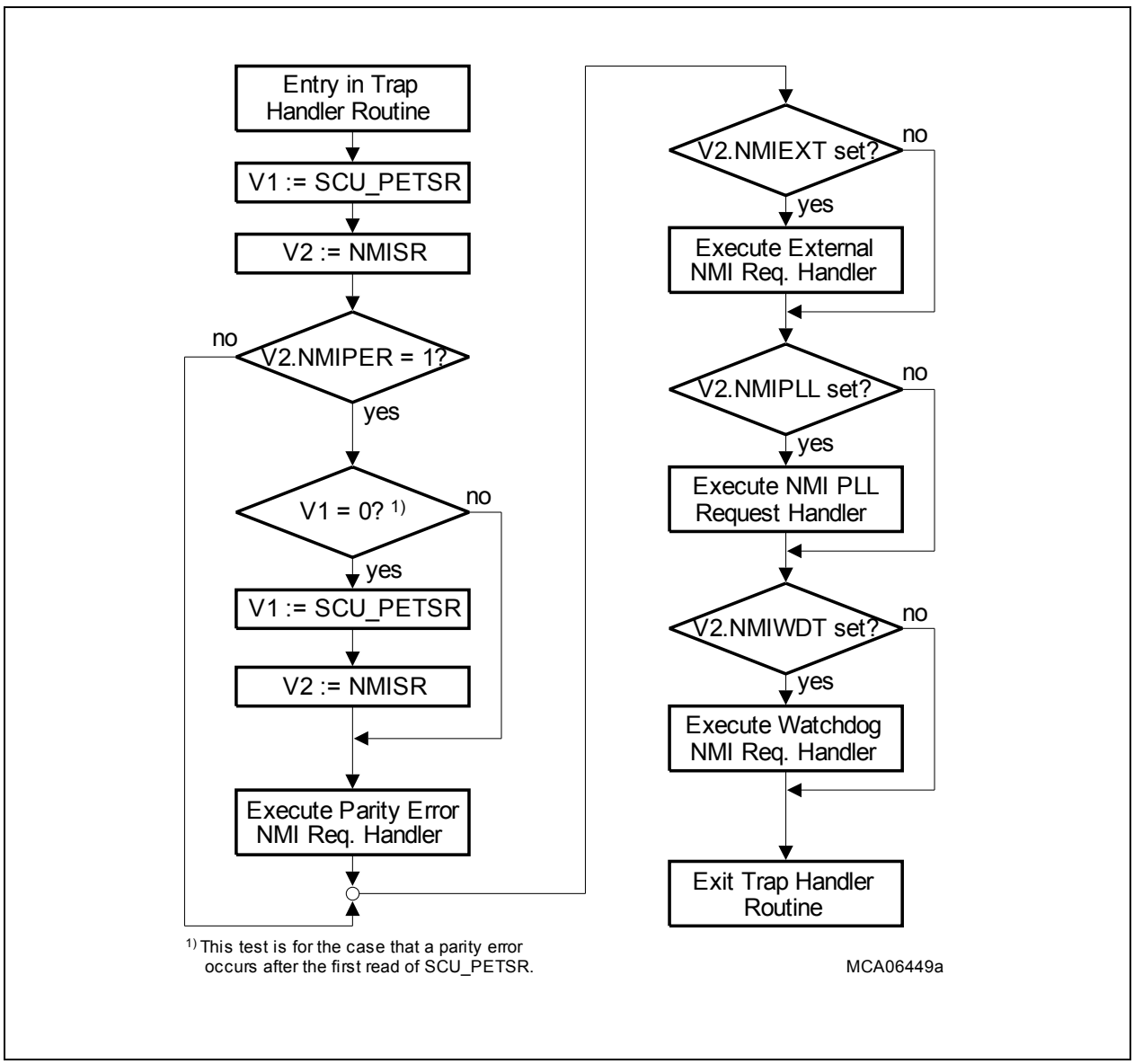


Figure 12-5 NMI Trap Handler Routine for Parity Error Handling

12.10.5 NMI Enable

After reset, the NMI is disabled. It must be enabled by the user program when the NMI handler routine has been setup. The NMI is enabled by setting bit SCU_CON.NMIEN. Once NMIEN has been set, it cannot be cleared again by software but only by a reset operation (except Watchdog reset).

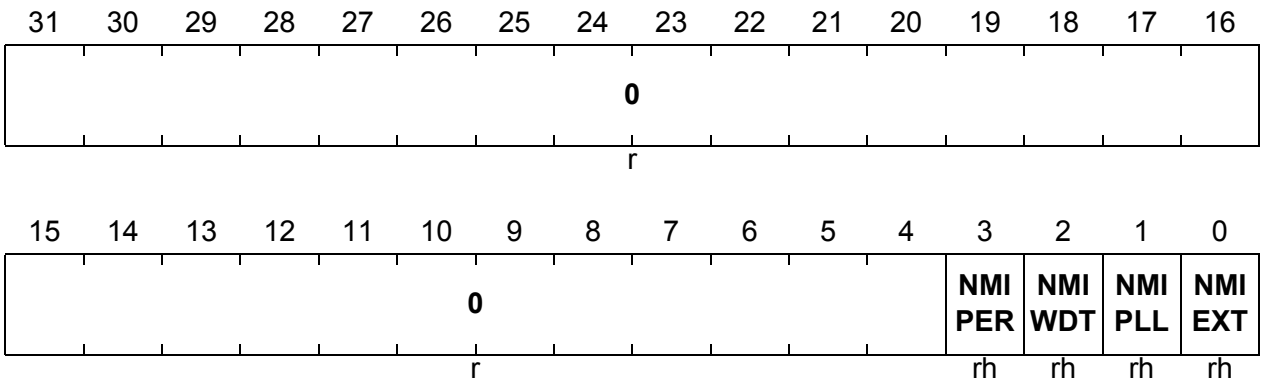
12.10.6 NMI Status Register

The source of an NMI trap can be identified through three status bits in NMISR. The bits in NMISR are read-only; writing to them has no effect.

The CPU detects a one-to-zero transition of the $\overline{\text{NMI}}$ input signal as indicating a NMI trap event. It then sets NMISR.NMIEXT. If the Watchdog Timer times out, it sets NMISR.NMIWDT. If the PLL loses its clock signal, it sets NMISR.PLL.

The bits in NMISR are OR-ed together to generate an NMI trap request to the CPU. If one of the NMISR bits is newly asserted while another bit is set, no new NMI trap request is generated. All flags are cleared automatically after a read of NMISR. Therefore, after reading NMISR, the NMI TSR must check all bits in NMISR to determine whether there have been multiple causes of an NMI trap.

NMISR
NMI Status Register **($F000002C_H$)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
NMIEXT	0	rh	**External NMI Flag** This flag indicates whether or not an external NMI request has occurred. 0_B No external NMI request has occurred. 1_B An external NMI request has been detected.

Field	Bits	Type	Description
NMIPLL	1	rh	**PLL NMI Flag** This flag indicates whether or not a PLL NMI request has occurred. 0_B No PLL NMI has occurred. 1_B The PLL has lost the lock to the external crystal (becomes unlocked).
NMIWDT	2	rh	**Watchdog Timer NMI Flag** This flag indicates whether or not a Watchdog Timer NMI request has occurred. 0_B No watchdog NMI occurred. 1_B The Watchdog Timer has entered the pre-warning phase due to a watchdog error.
NMIPER	3	rh	**Parity Error NMI Flag** This flag indicates whether or not SRAM parity error NMI request has occurred. 0_B No SRAM parity error NMI occurred. 1_B An SRAM parity error NMI has been detected. The SRAM module where the parity error occurred can be checked by reading SCU_PETSR. NMIPER is only cleared with an NMISR read operation when the bits in SCU_PETSR are no more set.
0	[31:4]	r	**Reserved** Read as 0.

*Note: The NMISR register is located in the address range of the System Control Unit (see **Page 5-70**).*

13 System Timer

This chapter describes the System Timer (STM). The TC1766's STM is designed for global system timing applications requiring both high precision and long period.

13.1 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 interrupt generation based on compare match with partial STM content
- Driven by maximum 80 MHz (= f_{SYS}, default after reset = $f_{SYS}/2$)
- Counting starts automatically after a reset operation
- STM is reset by:
  - Watchdog reset
  - Software reset (RST_REQ.RRSTM must be set)
  - Power-on reset
- STM (and clock divider STM_CLC.RMC) is not reset at a hardware reset ($\overline{\text{HDRST}}$ = 0)
- 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} = 80 MHz, for example, the STM counts 28.56 years before overflowing. Thus, it is capable of continuously timing the entire expected product life time of a system without overflowing.

13.2 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 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 TC1766. 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 TC1766 (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 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. Interrupts can be generated on a compare match of the STM with the STM_CMP0 or STM_CMP1 registers.

Figure 13-1 provides an overview on the STM module. It shows the options for reading parts of STM content.

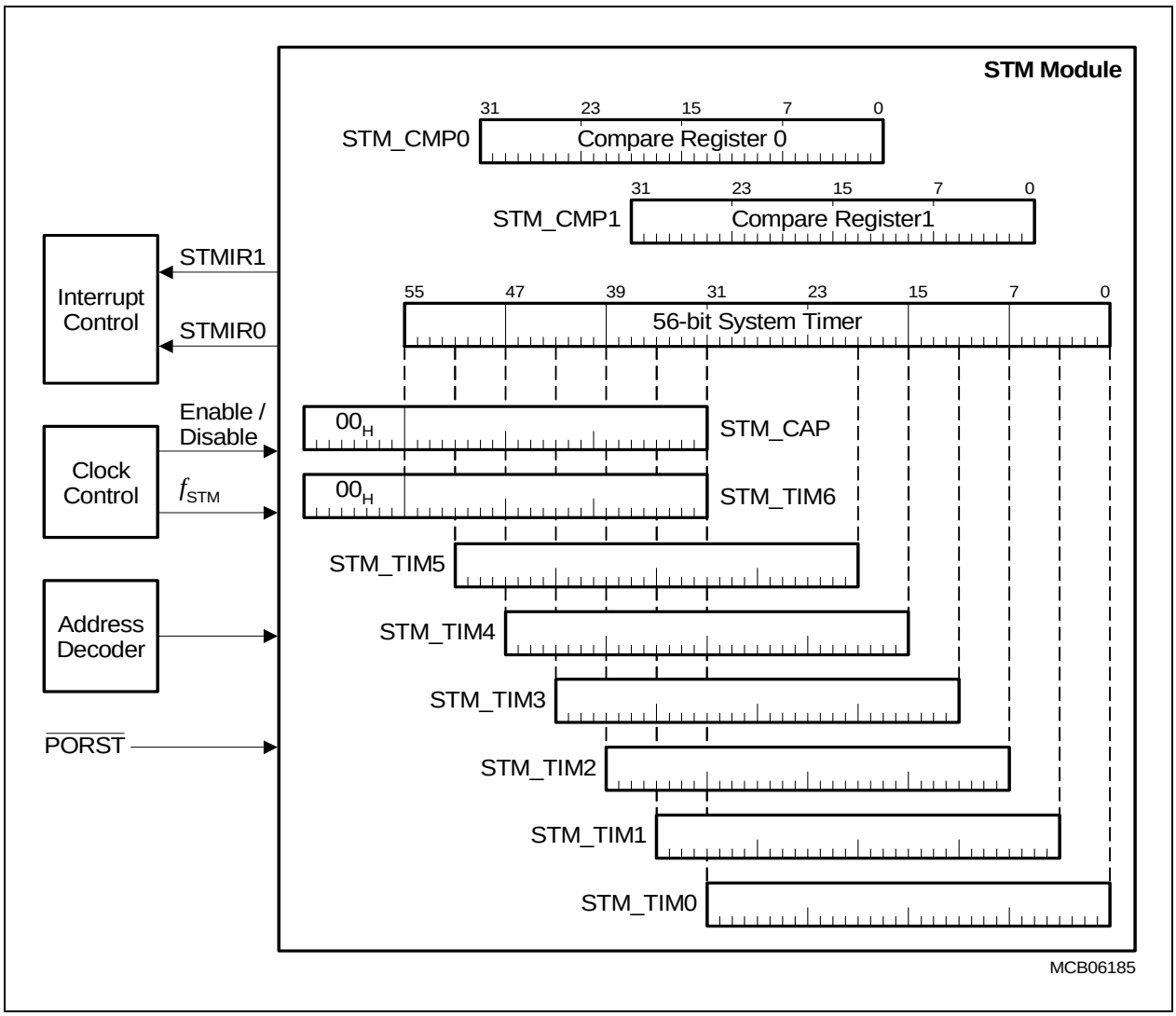


Figure 13-1 General Block Diagram of the STM Module Registers

13.2.1 Resolution and Ranges

Table 13-1 is an overview on the individual timer registers with their resolutions and timing ranges. As an example, the values for a 80 and 40 MHz STM input clock frequency are given.

Table 13-1 System Timer Resolutions and Ranges

Register	STM Bits	Resolution [s]	Range [s]	Resolution	Range	f_{STM} [MHz]
STM_TIM0	[31:0]	$1 / f_{STM}$	$2^{32} / f_{STM}$	12.5 ns	53.7 s	80
STM_TIM1	[35:4]	$16 / f_{STM}$	$2^{36} / f_{STM}$	200 ns	859.0 s	
STM_TIM2	[39:8]	$256 / f_{STM}$	$2^{40} / f_{STM}$	3.2 µs	229.1 min	
STM_TIM3	[43:12]	$4096 / f_{STM}$	$2^{44} / f_{STM}$	51.2 µs	61.1 h	
STM_TIM4	[47:16]	$65536 / f_{STM}$	$2^{48} / f_{STM}$	0.819 ms	40.72 days	
STM_TIM5	[51:20]	$2^{20} / f_{STM}$	$2^{52} / f_{STM}$	13.1 ms	1.79 yr	
STM_TIM6	[55:32]	$2^{32} / f_{STM}$	$2^{56} / f_{STM}$	53.7 s	28.56 yr	
STM_CAP	[55:32]	$2^{32} / f_{STM}$	$2^{56} / f_{STM}$	53.7 s	28.56 yr	
STM_TIM0	[31:0]	$1 / f_{STM}$	$2^{32} / f_{STM}$	25 ns	107.4 s	40
STM_TIM1	[35:4]	$16 / f_{STM}$	$2^{36} / f_{STM}$	400 ns	1718.0 s	
STM_TIM2	[39:8]	$256 / f_{STM}$	$2^{40} / f_{STM}$	6.4 µs	458.2 min	
STM_TIM3	[43:12]	$4096 / f_{STM}$	$2^{44} / f_{STM}$	102.4 µs	122.2 h	
STM_TIM4	[47:16]	$65536 / f_{STM}$	$2^{48} / f_{STM}$	1.64 ms	81.44 days	
STM_TIM5	[51:20]	$2^{20} / f_{STM}$	$2^{52} / f_{STM}$	26.2 ms	3.58 yr	
STM_TIM6	[55:32]	$2^{32} / f_{STM}$	$2^{56} / f_{STM}$	107.4 s	57.12 yr	
STM_CAP	[55:32]	$2^{32} / f_{STM}$	$2^{56} / f_{STM}$	107.4 s	57.12 yr	

Note: The maximum input clock f_{STM} is 80 MHz.

13.2.2 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. Interrupts 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.

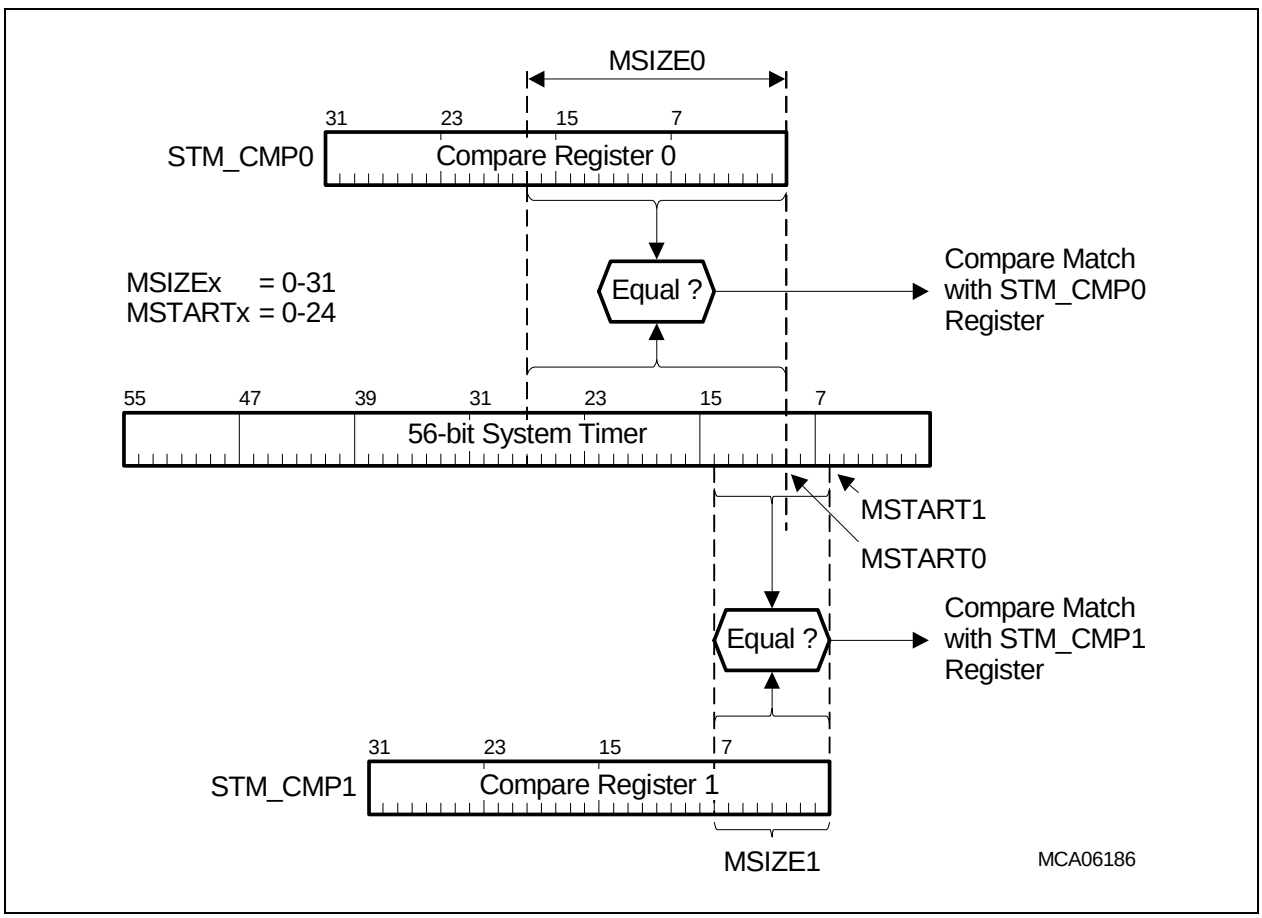


Figure 13-2 Compare Mode Operation

Figure 13-2 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 13-2**, 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.

13.2.3 Compare Match Interrupt Control

The compare match interrupt control logic is shown in **Figure 13-3**. 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 to 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 one of the 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. These registers control the general interrupt handling and processing as described in **Chapter 12** of this TC1766 System Units (Vol. 1 of 2) User's Manual.

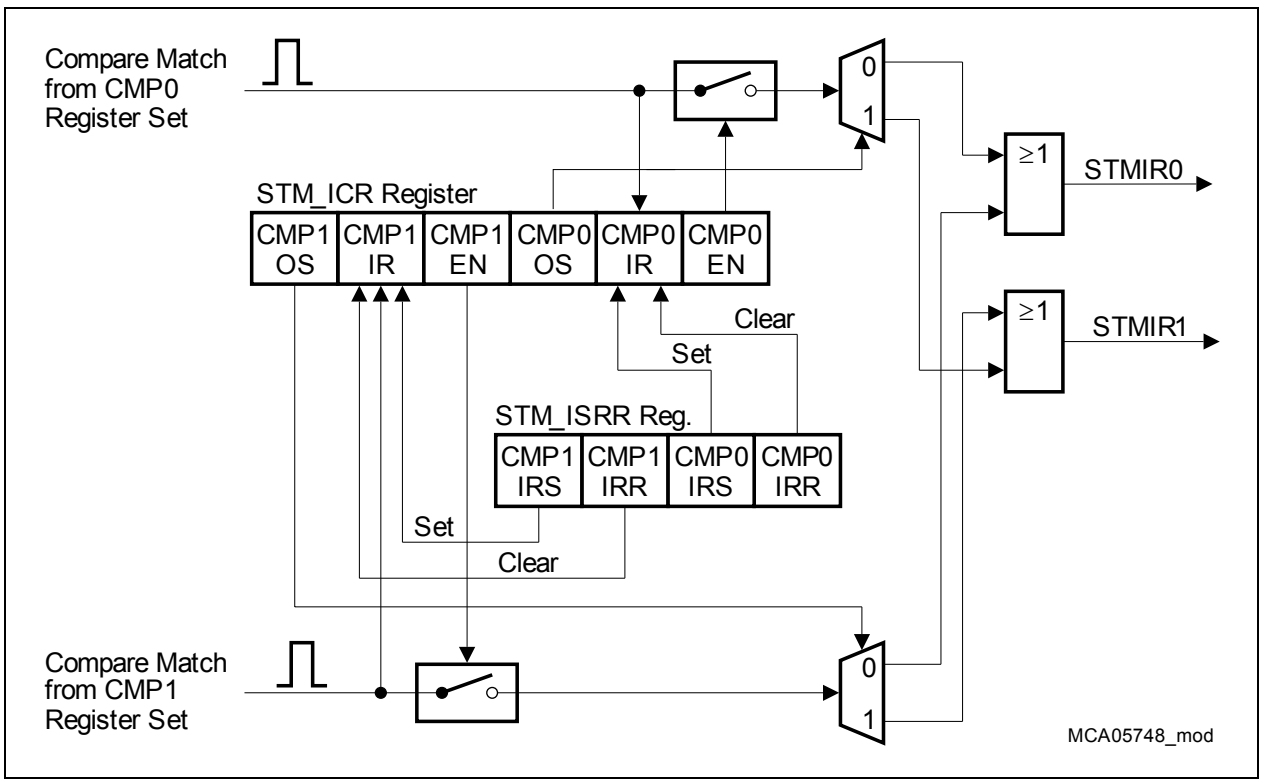


Figure 13-3 STM Interrupt Control

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 an STM reset operation (CMPxEN = 0). Therefore, before enabling a compare match interrupt after an 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.

13.3 Kernel Registers

This section describes the kernel registers of the STM. The STM registers can be divided into four types, as shown in **Figure 13-4**.

STM Registers Overview

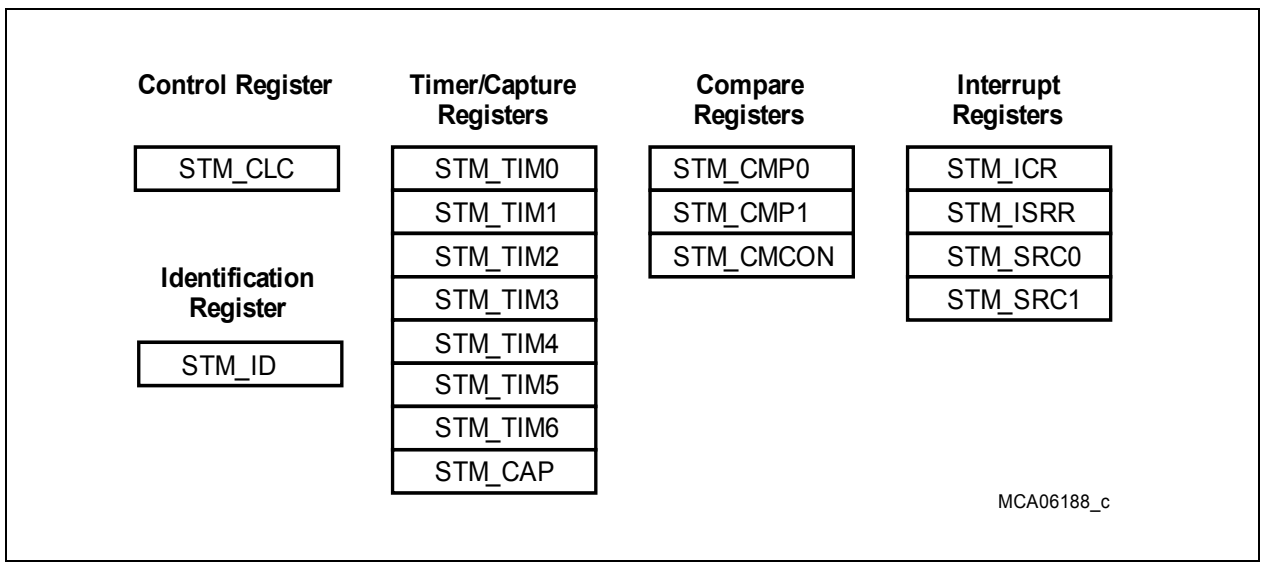


Figure 13-4 STM Registers

The complete and detailed address map of the STM module with its registers is shown in **Table 16-5** on **Page 16-12**.

Table 13-2 Registers Address Space

Module	Base Address	End Address	Note
STM	$F000\ 0200_H$	$F000\ 02FF_H$	-

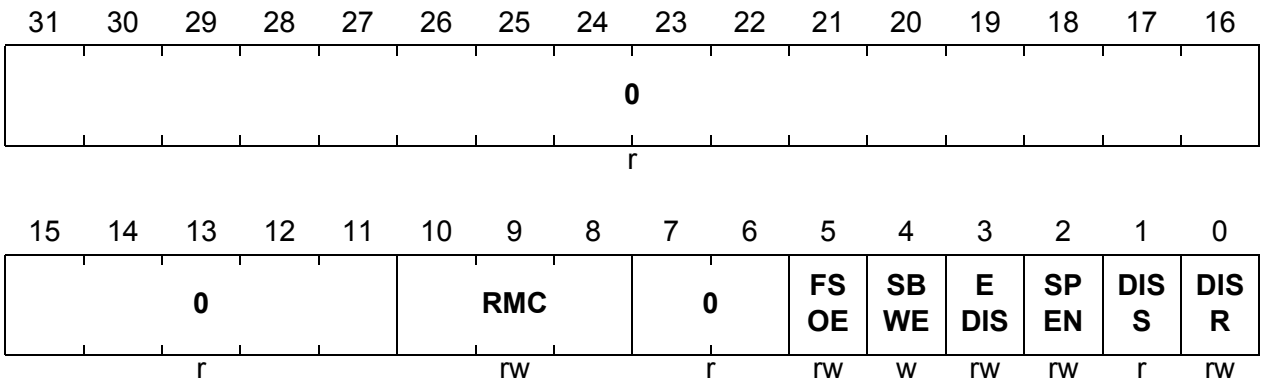
Table 13-3 Registers Overview - STM Registers

Register Short Name	Register Long Name	Offset Address	Description see
STM_CLC	STM Clock Control Register	00_H	Page 13-9
STM_ID	STM Module Identification Register	08_H	Page 13-11
STM_TIM0	STM Timer Register 0	10_H	Page 13-12
STM_TIM1	STM Timer Register 1	14_H	Page 13-12
STM_TIM2	STM Timer Register 2	18_H	Page 13-13
STM_TIM3	STM Timer Register 3	$1C_H$	Page 13-13
STM_TIM4	STM Timer Register 4	20_H	Page 13-13
STM_TIM5	STM Timer Register 5	24_H	Page 13-14
STM_TIM6	STM Timer Register 6	28_H	Page 13-14
STM_CAP	STM Timer Capture Register	$2C_H$	Page 13-15
STM_CMP0	STM Compare Register 0	30_H	Page 13-16
STM_CMP1	STM Compare Register 1	34_H	Page 13-16
STM_CMCON	STM Compare Match Control Register	38_H	Page 13-17
STM_ICR	STM Interrupt Control Register	$3C_H$	Page 13-19
STM_ISRR	STM Interrupt Set/Reset Register	40_H	Page 13-21
STM_SRC1	STM Interrupt Service Request Control Register 1	$F8_H$	Page 13-22
STM_SRC0	STM Interrupt Service Request Control Register 0	FC_H	Page 13-22

13.3.1 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
System Timer Clock Control Register (00_H) **Reset Value: $0000\ 0200_H$**



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.

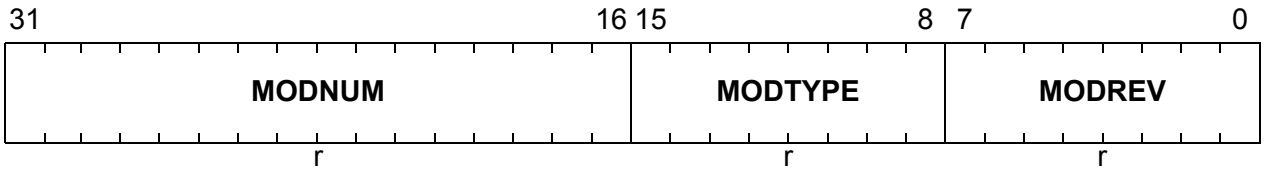
Field	Bits	Type	Description
RMC	[10:8]	rw	**Clock Divider in Run Mode** 000_B No clock signal f_{STM} generated 001_B Clock $f_{STM} = f_{SYS}$ selected 010_B Clock $f_{STM} = f_{SYS}$ / 2 selected (default after reset) 111_B Clock $f_{STM} = f_{SYS}$ / 7 selected *Note: This bit field is not affected by a hardware reset operation ($\overline{HDRST}$ = 0).*
0	[7:6], [31:11]	r	**Reserved** Read as 0; should be written with 0.

Note: In the TC1766, all registers are readable in suspend mode.

13.3.2 STM Module Identification Register

The STM Module Identification Register ID contains read-only information about the STM module version.

STM_ID
STM Module Identification Register (08_H) Reset Value: 0000 C0XX$_H$

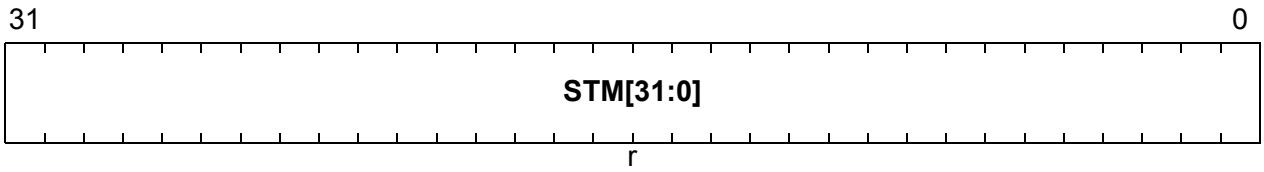


Field	Bits	Type	Description
MODREV	[7:0]	r	**Module Revision Number** MODREV defines the module revision number. The value of a module revision starts with 01_H (first revision).
MODTYPE	[15:8]	r	**Module Type** This bit field defines the module as a 32-bit module: $C0_H$
MODNUM	[31:16]	r	**Module Number Value** This bit field defines the module identification number for the STM: 0000_H

13.3.3 Timer/Capture Registers

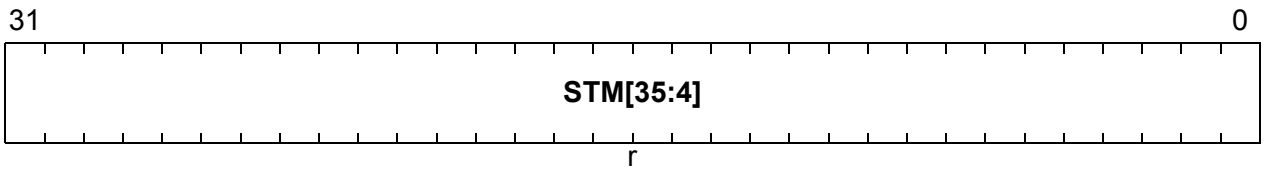
Registers STM_TIM1 to STM_TIM6 provide 32-bit views at varying resolutions of the underlying STM counter.

STM_TIM0
STM Timer Register 0 **(10_H)** **Reset Value: 0000 0000$_H$**



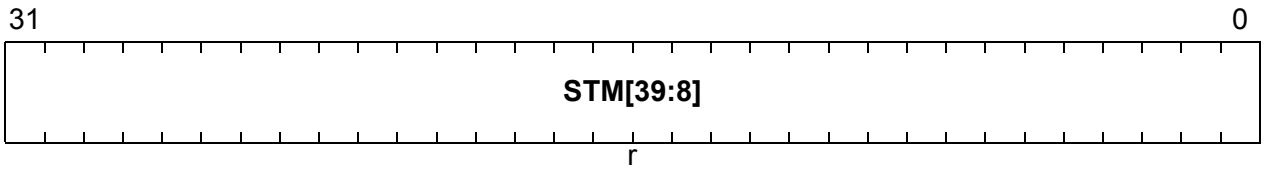
Field	Bits	Type	Description
STM[31:0]	[31:0]	r	**System Timer Bits [31:0]** This bit bield contains bits [31:0] of the 56-bit STM.

STM_TIM1
STM Timer Register 1 **(14_H)** **Reset Value: 0000 0000$_H$**



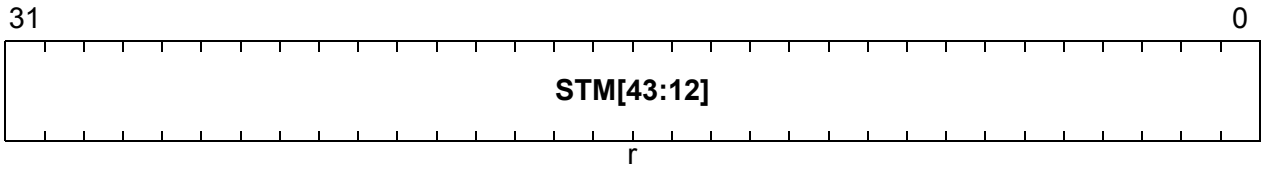
Field	Bits	Type	Description
STM[35:4]	[31:0]	r	**System Timer Bits [35:4]** This bit bield contains bits [35:4] of the 56-bit STM.

STM_TIM2
STM Timer Register 2 **(18_H)** **Reset Value: 0000 0000$_H$**



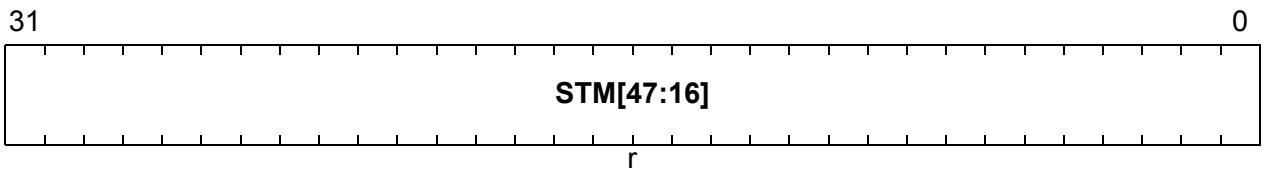
Field	Bits	Type	Description
STM[39:8]	[31:0]	r	**System Timer Bits [39:8]** This bit bield contains bits [39:8] of the 56-bit STM.

STM_TIM3
STM Timer Register 3 **($1C_H$)** **Reset Value: 0000 0000$_H$**



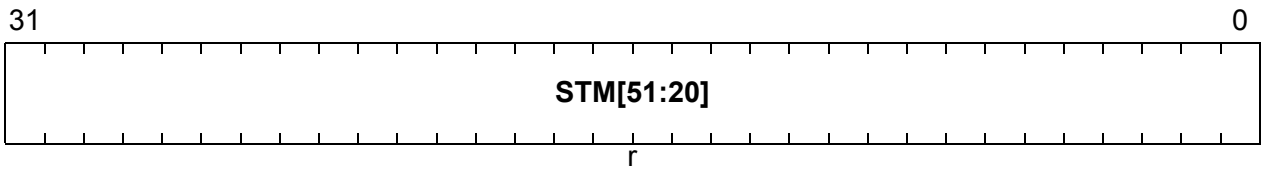
Field	Bits	Type	Description
STM[43:12]	[31:0]	r	**System Timer Bits [43:12]** This bit bield contains bits [43:12] of the 56-bit STM.

STM_TIM4
STM Timer Register 4 **(20_H)** **Reset Value: 0000 0000$_H$**



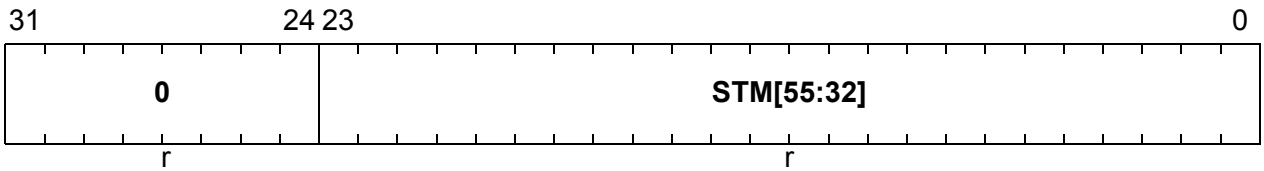
Field	Bits	Type	Description
STM[47:16]	[31:0]	r	**System Timer Bits [47:16]** This bit bield contains bits [47:16] of the 56-bit STM.

STM_TIM5
STM Timer Register 5 **(24_H)** **Reset Value: 0000 0000$_H$**



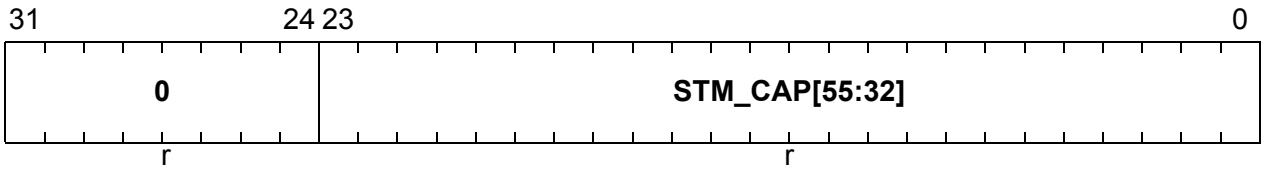
Field	Bits	Type	Description
STM[51:20]	[31:0]	r	**System Timer Bits [51:20]** This bit bield contains bits [51:20] of the 56-bit STM.

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 bield 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: $0000\ 0000_H$**



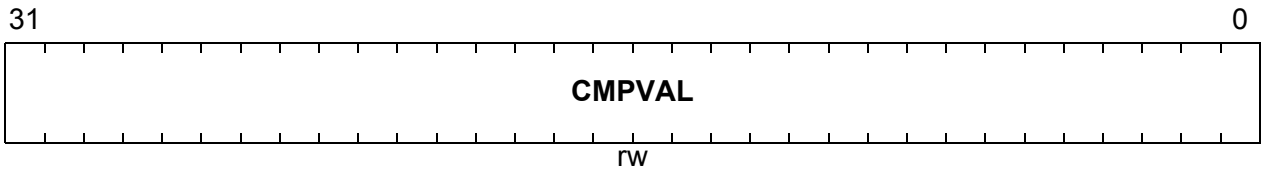
Field	Bits	Type	Description
STM[55:32]	[23:0]	r	**Captured System Timer Bits [55:32]** The capture register STM_CAP always captures the STM bits [55:32] when one of the registers STM_TIM0 to STM_TIM5 is read. This capture operation is performed in order to enable software to operate with a coherent value of all the 56 STM bits at one time stamp.This bit bield 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.

13.3.4 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**

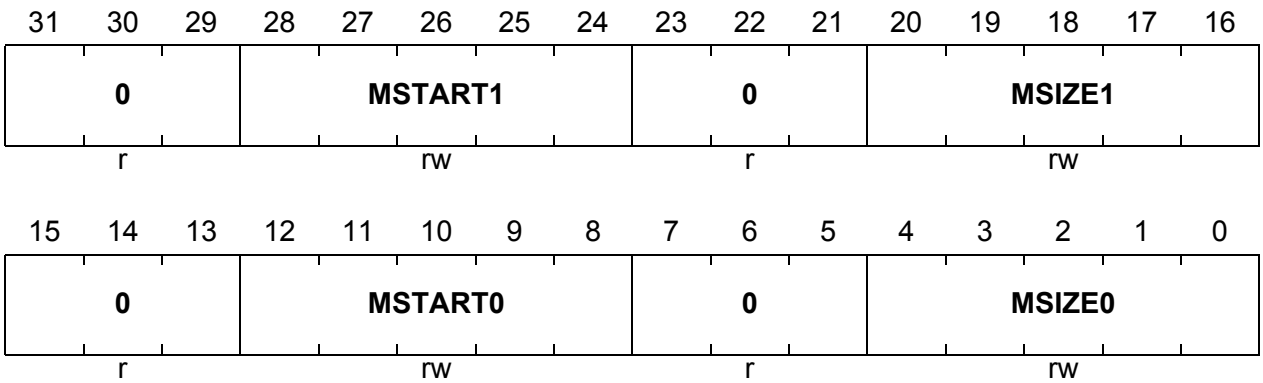


Field	Bits	Type	Description
CMPVAL	[31:0]	rw	**Compare Value of Compare Register x** This bit field holds up to 32 bits of the compare value (right-adjusted).

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 … 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 … 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 … 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 … 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.

13.3.5 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$_H$

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

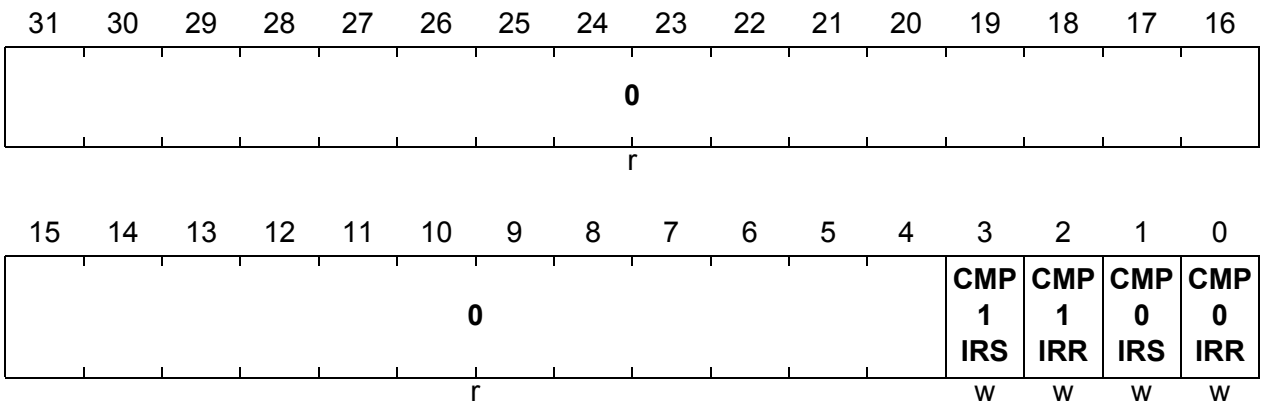
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0									CMP1OS	CMP1IR	CMP1EN	0	CMP0OS	CMP0IR	CMP0EN
r									rw	rh	rw	r	rw	rh	rw

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 an 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 an STM reset operation, 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 clear the compare match interrupt request status flags of register ICR.

STM_ISRR
STM Interrupt Set/Reset Register (40_H) Reset Value: 0000 0000_H



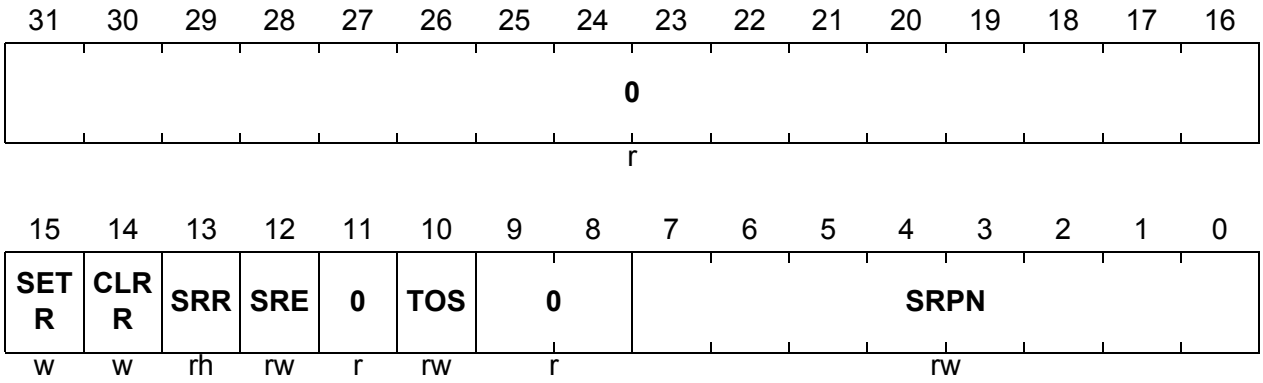
Field	Bits	Type	Description
CMP0IRR	0	w	**Reset Compare Register CMP0 Interrupt Flag** 0_B Bit ICR.CMP0IR is not changed 1_B Bit ICR.CMP0IR is cleared
CMP0IRS	1	w	**Set Compare Register CMP0 Interrupt Flag** 0_B Bit ICR.CMP0IR is not changed 1_B Bit ICR.CMP0IR is set. The state of bit CMP0IRR is “don’t 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 TC1766, the compare match interrupt output signals of the STM, STMIR0 and STMIR1 are controlled by the STM Interrupt 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.

*Note: Further details of interrupt handling and processing are described in **Chapter 12** of this TC1766 System Units (Vol. 1 of 2) User's Manual.*

14 Watchdog Timer

This chapter describes the TC1766 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.

14.1 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 TC1766 in a user-specified time period. When enabled, the WDT will cause the TC1766 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 TC1766 system 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. A system-wide line is connected to the WDT_CON0.ENDINIT bit, serving as an additional write-protection for critical registers (besides Supervisor Mode protection). Registers protected via this line can only be modified when Supervisor Mode is active and bit ENDINIT = 0.

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 triggered. 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 ENDINIT has not been properly set again before this limit expires, the system is assumed to have malfunctioned, and a Watchdog reset is triggered. These stringent requirements, although not guaranteed, nonetheless provide a high degree of assurance of the robustness of system operation.

A further enhancement in the TC1766’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. This gives the CPU a chance to save system state to memory for later examination of the cause of the malfunction, an important aid in debugging.

14.2 Features of the Watchdog Timer

The main features of the WDT are summarized here. The WDT is implemented in the System Control Unit (SCU) module of the TC1766. **Figure 14-1** gives an overview of its interface signals.

- 16-bit Watchdog counter
- Selectable input frequency: f_{SYS}/256 or f_{SYS}/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
- Proper access always requires two write accesses. The time between the two accesses is monitored by the WDT and is limited
- 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: If a Watchdog induced reset occurs twice, a severe system malfunction is assumed and the TC1766 is held in reset until a power-on or hardware 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
- Important debugging support is provided through the reset prewarning operation by first issuing an NMI to the CPU before finally resetting the device after a certain period of time

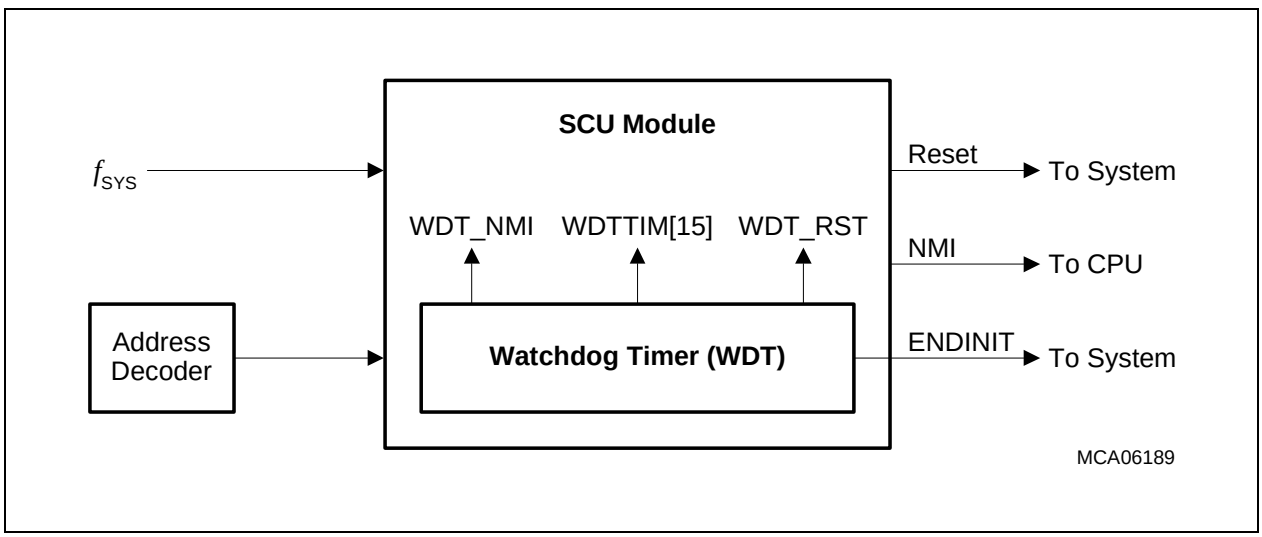


Figure 14-1 Interface of the WDT Inside and Outside the SCU Module

14.3 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 TC1766 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, which 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 TC1766 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. A system-wide line is connected to this bit. Registers protected via Endinit use the state of this line to determine whether or not writes are enabled. Writes are only enabled if 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. An interrupt in the corresponding bus control units of the TC1766 is generated as well instead of a bus error trap. There is an exception: If read-modify-write instructions are used for the write access, a bus error trap is generated instead of an interrupt.

An additional line, controlled through a separate bit, makes it possible to protect against unintentional writes, providing an extra level of security. However, 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. It is explained in the following sections. In addition, the WDT monitors ENDINIT modifications by starting a time-out sequence each time software opens access to the critical registers through clearing ENDINIT to 0. If the time out period ends before ENDINIT is set to 1 again, a malfunction of the software and/or the hardware is assumed and the device is reset.

The access-protection scheme and the Endinit time-out operation of the WDT is described in the following sections. **Table 14-1** lists the registers that are protected via the Endinit feature in the TC1766.

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 after ensuring that ENDINIT has been 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 ENDINIT has been cleared.

Table 14-1 TC1766 Registers Protected via the Endinit Feature

Normal Mode	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.
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.
RST_REQ **OSC_CON** **PLL_CLC** **SCU_SCLKFDR** **SCU_EMSR** **SCU_TCCON** **SCU_CON** **SCU_TCLR0** **SCU_PTCON** **Px_PDR** **Px_ESR** **PCP_CS** **DMA_OCDSR** **DMA_SUSPMR** **FLASH_FCON** **FLASH_MARP** **DMI_CON** **DMI_CON1** **PMI_CON0** **DMA_ME0AENR** **DMA_ME0ARR** **MLIx_AER** **MLIx_ARR**[1)]	All these registers are also Endinit-protected.

1) x = 0 or x = 1

14.4 Watchdog Timer Operation

The following sections describe the registers, the operation, and different modes of the WDT, as well as the Password Access mechanism. **Figure 14-2** gives an example of the operation of the WDT. A brief description of the sequence of events in this figure is provided. Refer to the following sections for a detailed explanation.

1. Time-Out Mode is automatically entered after reset. Timer counts with slowest input clock.
2. Time-Out Mode terminated and Normal Mode is entered by setting ENDINIT to 1.
3. Normal Mode is terminated and Time-Out Mode is entered through a Password Access to WDT_CON0. The reload value was set to REL_1.
4. Time-Out Mode is terminated and Normal Mode entered again by setting ENDINIT to 1. The reload value WDTREL has been changed to REL_2 and the timer input clock was set to the fast clock.
   Events 3) and 4) constitute a WDT service sequence.
5. The WDT was not serviced and continued to count until overflow. Reset Prewarning Mode is entered. Timer counts with selected fast input clock. Watchdog operation cannot be altered or stopped in this mode.
6. Timer continued to count until overflow, generating a WDT reset.
7. Time-Out Mode is automatically entered after reset. Timer counts with slowest input clock.
8. Time-Out Mode is terminated and Normal Mode is entered again.

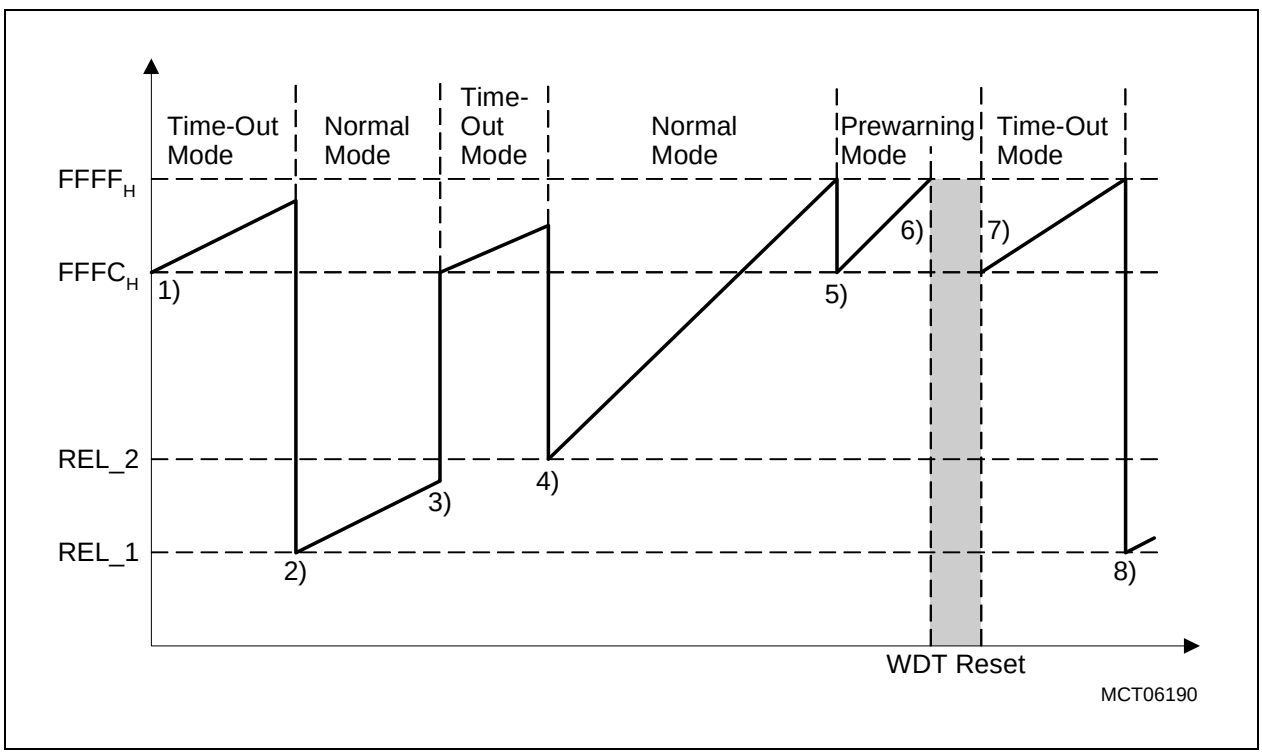


Figure 14-2 Example for an Operation Sequence of the WDT

14.4.1 WDT Register Overview

Two control registers, WDT_CON0 and WDT_CON1, and one status register, WDT_SR, serve the communication of the software with the WDT. This section provides a short overview and describes the access mechanisms of the WDT registers. Detailed layout and bit descriptions of the registers are given on Page 14-28.

Register WDT_CON0 holds the ENDINIT bit, a register lock status bit (WDTLCK), an 8-bit user-definable password field (WDTPW), and the user-definable reload (start) value (WDTREL) for the WDT in Normal Mode.

Register WDT_CON1 contains two bits. Bit WDTIR is a request bit for the WDT input frequency selection, while bit WDTDR is a request bit for the Disable Mode of the WDT. These two bits are only request bits in that they do not actually control the input frequency and disabling of the WDT. They can be modified only when the ENDINIT bit is 0, but they will have an effect only when ENDINIT is properly set to 1 again.

The status register WDT_SR holds information about the current conditions of the WDT. It contains the current timer count value (WDTTIM), three bits indicating the mode of operation (WDTTO for Time-Out Mode, WDTPR for Prewarning Mode, and WDTDS for Disable Mode), and the error indication bits for timer overflow (WDTOE) and access error (WDTAE).

While WDT_SR is a read-only register, the control registers can be read and written. Reading these registers is always possible; a write access, however, must follow certain protocols. Register WDT_CON1 is Supervisor Mode and Endinit-protected; thus, Supervisor Mode must be active and bit ENDINIT must be 0 for a successful write to this register. If one or both conditions are not met, a bus error will be generated, and the bits in WDT_CON1 will not be modified.

Register WDT_CON0 requires a much more complex write procedure as it has a special write-protection mechanism. Proper access to WDT_CON0 always requires two write accesses in order to modify its contents. The first write access requires a password to be written to the register to unlock it. This access is called Password Access. Then, the second access can modify the register's contents. It is called Modify Access. When the Modify Access completes, WDT_CON0 is automatically locked again. (Even if no parameters are changed in the second write access, it is still called a Modify Access.) If the Modify Access sets ENDINIT = 0, then other protected system registers, such as WDT_CON1, are unlocked and can be modified.

Note: WDT_CON0 is automatically re-locked after a Modify Access, so a new Password Access must be performed to modify it again. Note further that the WDT switches to Time-Out Mode as a side-effect of a successful Password Access, so that protected registers can remain unlocked at most for the duration of one Time-out Period. Otherwise, the system will be forced to reset.

14.4.2 Operating Modes of the Watchdog Timer

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. As well as these major operating modes, the WDT has special behavior during power-saving and OCDS suspend modes. Detailed discussions of each of the modes can be found on **Page 14-13**.

Figure 14-3 is a state diagram of the different modes of the WDT and the transition possibilities. Please refer to the description for the conditions required to change from one state to the other.

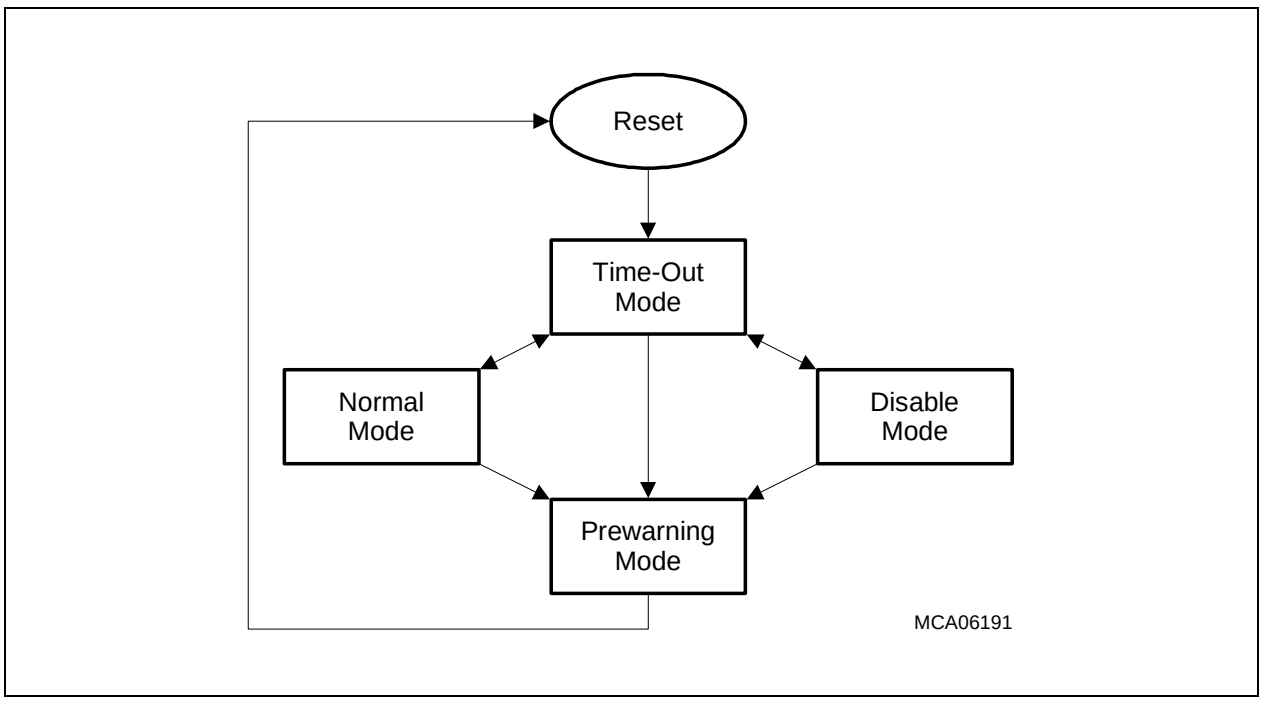


Figure 14-3 State Diagram of the WDT Operating Modes

14.4.2.1 Time-Out Mode

The Time-Out Mode is the default mode after a reset. It is also always entered when a Valid Password Access to register WDT_CON0 is performed (see **Section 14.4.3**). The timer is set to a predefined value and starts counting upwards. Time-Out Mode can only be exited properly by setting ENDINIT to 1 with a correct access sequence. If an improper access to the WDT is performed, or if the timer overflows before ENDINIT is set to 1, a WDT Non-Maskable Interrupt request (WDT_NMI) is requested, and Prewarning Mode is entered. A reset of the TC1766 is imminent and can no longer be stopped.

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, WDTDR, in register WDT_CON1.

14.4.2.2 Normal Mode

In Normal Mode (WDTDR = 0), the WDT operates in a standard Watchdog fashion. The timer is set to a user-defined start value, and begins counting up. It has to be serviced before the counter overflows. Servicing is performed through a proper access sequence to the WDT control register WDT_CON0. This reloads the timer with the start value, and normal operation continues.

If the WDT is not serviced before the timer overflows, or if an invalid access to the WDT is performed, a system malfunction is assumed. Normal Mode is terminated, a WDT NMI request (WDT_NMI) is requested, and Prewarning Mode is entered. A reset of the TC1766 is imminent and can no longer be stopped.

Because servicing the WDT is an access sequence, first requiring a Valid Password Access to register WDT_CON0, the WDT will enter Time-Out Mode until the second proper access is performed.

14.4.2.3 Disable Mode

Disable Mode is provided for applications which truly do not require the WDT function. It can be entered from Time-Out Mode when the disable request bit WDTDR is set to 1. The timer is stopped in this mode. However, disabling the WDT only stops it from performing the standard Watchdog function (Normal Mode), 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.

14.4.2.4 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 generating a reset of the device, as other WDTs do, the TC1766 WDT provides the system with a chance to save important state information before the reset occurs. This is done through first activating an NMI trap request to the CPU, warning it about the oncoming reset (reset prewarning). If the CPU is still able to do so (depending on the type and severity of the detected malfunction), it can react to the Watchdog NMI request and can save important system state to memory. This saved system state can then be examined during debugging to determine the cause of the malfunction. If the part would be immediately reset on the detection of a Watchdog error, this debugging information would never be available, and investigating the cause of the malfunction would be a very difficult task.

In Prewarning Mode, after having generated the NMI request, the WDT counts for a specified period of time, and then generates a Watchdog reset for the device. This reset generation cannot be avoided in this mode; the WDT does not react anymore to accesses to its registers, nor will it change its state. This is to prevent a malfunction from falsely terminating this mode, disabling the reset, and letting the device to continue to function improperly.

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 a soft reset to shorten the time. However, the state of the Watchdog Status Register should also be saved in this case, because the error flags contained in it will be cleared due to the soft reset (this is not the case if the Watchdog reset is awaited).

14.4.3 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 14-2** summarizes the requirements for the password.

Table 14-2 Password Access Bit Pattern Requirements

Bit Position	Required Value
0	Current state of the ENDINIT bit, WDT_CON0.ENDINIT
1	Fixed; must be written with 0
2	Current state of the input frequency request bit, WDT_CON1.WDTIR
3	Current state of the input frequency request bit, WDT_CON1.WDTDR
[7:4]	Fixed; must be written to 1111_B
[15:8]	Current value of user-definable password field, WDT_CON0.WDTPW
[31:16]	Current value of user-definable reload value, WDT_CON0.WDTREL

When reading register WDT_CON0, bit positions [7:4] always return 0s. As can be seen from **Table 14-2**, 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.WDTLCK = 0.

If WDT_CON0 is successfully unlocked, a subsequent write access can modify it, as described on **Page 14-11**.

If an improper password value is written to WDT_CON0 during the Password Access, a Watchdog Access Error condition exists. Bit WDTAE is set and the Prewarning Mode is entered.

The user-definable password, WDTPW, 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. See **Section 14.5.4**.

14.4.4 Modify Access to WDT_CON0

If WDT_CON0 is successfully unlocked as described on Page 14-10, 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 14-3 lists the required bit patterns. If the access does not follow these rules, a Watchdog Access Error condition is detected, bit WDTAE is set, and the Prewarning Mode is entered.

Table 14-3 Modify Access Bit Pattern Requirements

Bit Position	Value
0	User-definable; desired value for the ENDINIT 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.WDTPW.
[31:16]	User-definable; desired value of user-definable reload value, WDT_CON0.WDTREL.

After the Modify Access has completed, WDT_CON0.WDTLCK is set to 1 again by hardware, automatically re-locking WDT_CON0. Before the register can be modified again, a valid Password Access must be executed again.

14.4.5 Term Definitions for WDT_CON0 Accesses

To simplify the descriptions in the following sections, a number of terms are defined to indicate the type of access to register WDT_CON0:

Watchdog Access Sequence: Two accesses to register WDT_CON0 consisting of first a Password Access followed by a Modify Access. The two accesses do not have to be adjacent accesses, any number of accesses to other addresses can occur between these accesses as long as the Time-out Period is not exceeded.

Password Access: The first access of a Watchdog Access Sequence to register WDT_CON0 intended to open WDT_CON0 for modifications. This access needs to write a defined password value to WDT_CON0 in order to successfully open WDT_CON0.

Valid Password Access: A Password Access with the correct password value. A Valid Password Access opens register WDT_CON0 for one, and only one, Modify Access. Bit WDTLCK is set to 0 after this access. The WDT is placed into the Time-Out Mode after a Valid Password Access in Normal Mode or Disabled Mode.

Modify Access: The second access of an Watchdog Access Sequence to register WDT_CON0 intended to modify parameters in WDT_CON0. The parameters that can be modified are WDTREL, WDTPW and ENDINIT. Special guard bits in WDT_CON0 must be written with predefined values in order for this access to be accepted.

Valid Modify Access: A Modify Access with the correct guard bit values. The values written to WDTREL, WDTPW, and ENDINIT are in effect after completion of this access. Bit WDTLCK is automatically set to 1 after this access. Register WDT_CON0 is locked until it is re-opened with a Valid Password Access again.

14.4.6 Detailed Descriptions of the WDT Modes

The following subsections provide detailed descriptions of each of the modes of the WDT. The entry conditions and actions, operation of a mode, as well as exit conditions and the succeeding mode are provided for each mode.

14.4.6.1 Time-Out Mode Details

Time-Out Mode is the default after reset, and is entered each time a Valid Password Access to register WDT_CON0 is performed.

Table 14-4 WDT Time-Out Mode

State / Action	Description
Entry	• Automatically after any reset • If a valid password was written to WDT_CON0 in Normal or Disable Mode
Actions on Entry	• WDTTIM is set to $FFFC_H$; WDTTO is set to 1; WDTDS is set to 0. • ENDINIT = 0 if mode entered through reset; otherwise, it retains its previous value. • Bits WDTAE and WDTOE depend on their state before the reset if the reset was caused by the Watchdog. For any other reset (PORST, HDRST, SRST, PWDRST), they are 0. • WDTIS retains its previous value. • After reset, ENDINIT is 0. Thus, access to Endinit-protected registers is enabled. If Time-Out Mode was entered through other reasons, ENDINIT may or may not be 0.
Operation	• Timer starts counting up from $FFFC_H$; increments with clock rate determined through WDTIS (0 after reset, slowest clock). • Access to register WDT_CON0 is possible. Access to register WDT_CON1 is possible if ENDINIT = 0. • Restarting Time-Out Mode is not possible: A valid Password Access in this mode does not invoke another Time-out sequence (it does not reload the timer, etc.). A Modify Access to WDT_CON0 writing a 0 to ENDINIT does not terminate Time-Out Mode. • It is not possible to change the reload value or frequency in Time-Out Mode, as this would require setting ENDINIT to 1, which terminates Time-Out Mode. Reload value is not used until Normal mode is entered.

Table 14-4 WDT Time-Out Mode (cont'd)

State / Action	Description
Exit	1. Writing ENDINIT to 1 with a Valid Modify Access (a Valid Password Access must have been executed first). 2. Timer WDTTIM overflows from $FFFF_H$ to 0000_H. 3. An invalid access to WDT_CON0 (either during the password or the Modify Access).
Next Mode	Depending on the Exit condition: 1. If WDTDR = 0 (no disable request), the WDT enters the Normal Mode or if WDTDR = 1 (disable request), the WDT enters the Disable Mode. 2. Bit WDTOE is set to 1, and the WDT enters the Prewarning Mode. 3. Bit WDTAE is set to 1, and the WDT enters the Prewarning Mode.

14.4.6.2 Normal Mode Details

Normal Mode can be entered from Time-Out Mode only if bit WDT_CON1.WDTDR is set to 0 before proper termination of Time-Out Mode. The WDT operates as a standard Watchdog in this mode, requiring timely service to prevent a timer overflow.

Table 14-5 WDT Normal Mode

State / Action	Description
Entry	• Only from Time-Out Mode by writing ENDINIT to 1 with a Valid Modify Access (a Valid Password Access must have been executed first), while bit WDTDR = 0.
Actions on Entry	• WDTTIM is loaded with the value of WDTREL. • Bits WDTAE, WDTOE, WDTPR, WDTTO, and WDTDS are cleared to 0.
Operation	• WDTTIM starts counting up from reload value with frequency selected through WDTIS.
Exit	1. A Valid Password Access to register WDTCON. 2. Timer WDTTIM overflows from $FFFF_H$ to 0000_H. 3. An invalid access to WDT_CON0 (either during the password or the Modify Access).
Next Mode	Depending on Exit condition: 1. Time-Out Mode. 2. Prewarning Mode, bit WDTOE is set to 1 (overflow error). 3. Prewarning Mode, bit WDTAE is set to 1 (access error).

14.4.6.3 Disable Mode Details

Disable Mode is provided for applications that do not require the WDT function. It can only be entered from Time-Out Mode if bit WDT_CON1.WDTDR is set to 1 before proper termination of Time-Out Mode. The counter stops in this mode, eliminating the need for a WDT service. However, if an access to register WDT_CON0 is performed, the WDT will leave Disable Mode. Disable Mode does not stop the detection of access errors and the entry of Prewarning Mode nor the entry of Time-Out Mode on a Valid Password Access.

Table 14-6 WDT Disable Mode

State / Action	Description
Entry	• Only from Time-Out Mode by writing ENDINIT to 1 with a Valid Modify Access (a Valid Password Access must have been executed first), while bit WDTDR = 1.
Actions on Entry	• Bits WDTAE, WDTOE, WDTPR, and WDTTO are cleared. Bit WDTDS is set to 1. • Timer WDTTIM is stopped (it retains its current value).
Operation	–
Exit	1. Valid Password Access to register WDTCON. 2. Invalid access to WDT_CON0 (either during the password or the Modify Access).
Next Mode	Depending on Exit condition: 1. Time-Out Mode. 2. Prewarning Mode, bit WDTAE is set to 1 (access error).

14.4.6.4 Prewarning Mode Details

Prewarning Mode is always entered immediately after a Watchdog error condition is detected. This can be either an access error to register WDT_CON0 or an overflow of the counter in Normal or Time-Out Mode. This mode indicates that a reset of the device is imminent. Operation of the WDT in this mode can not be altered or stopped, except through a reset.

Table 14-7 WDT Prewarning Mode

State / Action	Description
Entry	Detection of a Watchdog error: • Overflow of timer WDTTIM. • Access error to register WDT_CON0 (either on a password or Modify Access) in Time-Out, Normal, or Disable modes.
Actions on Entry	• NMIWDT in register NMISR is set (this triggers an NMI request to the CPU). • WDTTIM is set to $FFFC_H$. • WDTPR is set to 1; WDTDS is set to 0; WDTIS retains its value. • WDTTO retains its previous value: If entry into Prewarning Mode was from Time-Out Mode, WDTTO is 1. In all other cases, WDTTO is 0. • Bits WDTAE and WDTOE indicate whether Prewarning Mode was entered due to an access or an overflow error. They have been set accordingly on exit of the previous mode.
Operation	• Timer WDT_TIM starts counting up from $FFFC_H$ with frequency selected through WDTIS. • Register WDT_CON0 can be accessed in this mode as usual. However, the WDT will not change its mode anymore, regardless whether valid or invalid accesses are made to WDT_CON0. For invalid accesses to WDT_CON0 (Password or Modify Access), however, bit WDTAE in WDT_SR will be set. • Register WDT_CON1 cannot be written to in Prewarning Mode, even if bit ENDINIT = 0. Write access to WDT_CON1 is totally prohibited.
Exit	• Prewarning Mode cannot be disabled, prolonged, or terminated (except through a reset). The timer will increment until it overflows from $FFFF_H$ to 0000_H, which then causes a system reset. Bit WDTRST in register RSTSR is set in this case.
Next Mode	Reset

Note: In Prewarning Mode, the device does not have to wait for the end of the Time-out Period and the reset. After having saved required state in the NMI routine,

software can execute a soft reset to shorten the time. However, the state of the Watchdog Status Register should also be saved in this case, since the error flags contained in it will be cleared due to the soft reset (this is not the case if the Watchdog reset is awaited).

14.4.6.5 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.WDTTIM) from $7FFF_H$ to 8000_H (when the most significant bit of the WDT counter changes its state from 0 to 1), the CPU is awakened and continues to execute the instruction that follows 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.WDTREL, 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.

14.4.6.6 WDT Operation in OCDS Suspend Mode

When the On-Chip Debugging System (OCDS) is enabled after reset, the WDT will automatically stop when OCDS Suspend Mode is activated. It will resume operation after the Suspend Mode is deactivated.

It is possible that severe system malfunctions may not be corrected even by a system reset. If application code cannot be executed properly due to a system fault, then the WDT initialization code itself may not be able to be executed in order to service the WDT, with the result that two WDT-initiated resets might occur back-to-back. A feature of the WDT detects such Double Watchdog Errors and suspends all system operations after the second reset occurs. This feature prevents the TC1766 from executing random wrong code for longer than the Time-out Period, and prevents the TC1766 from being repeatedly reset by the WDT.

The purpose of the Double Watchdog Error feature is to avoid loops such as: Reset - software does not start correctly - prewarning - watchdog reset - software does not start correctly - …

The WDT has an internal counter that generates a constant reset after a second watchdog error. This counter can only be cleared by external reset sources (power-on reset or hardware reset).

14.4.7 Determining WDT Periods

The WDT uses the system clock, f_{SYS}. A clock divider in front of the WDT provides two output frequencies, $f_{SYS}/256$ and $f_{SYS}/16384$. Bit WDT_SR.WDTIS selects between these options.

When the WDT is in Normal Mode, the duration of a WDT cycle is defined as a Normal Period, as described on **Page 14-20**.

When the WDT is in Time-out Mode or Prewarning Mode, the duration of a WDT cycle is defined as a Time-out Period, as described in the next section.

The general formula to calculate the Watchdog period is:

$$\text{period} = \frac{(2^{16} - \text{startvalue}) \times 256 \times 2^{(1-\text{WDTIS}) \times 6}}{f_{SYS}} \quad (14.1)$$

The parameter startvalue represents the fixed value $FFFC_H$ for the calculation of the Time-out Period, and the user-programmable reload value WDTREL for the calculation of the Normal Period. Note that the exponent (1 - WDTIS) × 6 results to 0 if WDTIS is 1, and to 6 if WDTIS is 0. This results in the value 256 being multiplied by either 1 (2^0 = 1) or by 64 (2^6), giving the two divider factors 256 and 16384.

Note: Because there is no synchronization of the clock divider to the mode transitions of the Watchdog, the next clock pulse, incrementing the counter, may come after one clock divider period, or immediately after the counter was reloaded. Thus, it is recommended that the reload value is programmed to a value that results in one clock pulse more than the required period.

14.4.7.1 Time-out Period

The duration of Time-Out Mode and Prewarning Mode is determined by the Time-out Period described here. The Time-out Period that occurs immediately after reset is governed entirely by system defaults, as no software is able to run at this point; it is described separately below.

Time-out Period After Reset

After reset, the initial count value for the timer is fixed at $FFFC_H$ when the WDT clock starts running. The WDT counts up at a rate determined by WDT_SR.WDTIS, which is 0 after any reset (f_{SYS}/16384). Counting up from $FFFC_H$, it takes four clocks for the counter to overflow, so the Time-out Period defaults to a period of $4 \times 16384/f_{SYS} = 65536/f_{SYS}$. This establishes the real-time deadline for software to initialize the Watchdog and critical system registers, and to then set ENDINIT. For example, the Time-out Period after reset would correspond to 1.6 ms (@ 40 MHz system frequency).

Changing the input frequency selection via WDT_CON1.WDTIR during this initial Time-out Period has no immediate effect, because frequency selection is actually determined by WDT_SR.WDTIS, but WDT_CON1.WDTIR is only copied into WDT_SR.WDTIS after WDT_CON0.ENDINIT has been set to 1, that is, after Time-Out Mode has been properly exited. Hence, the new input frequency will become effective only in a subsequent Time-out Period.

Time-out Period During Normal Operation

As after reset, the WDT counter is initially set to $FFFC_H$ when Time-Out Mode is entered, and Time-Out Mode expires when the counter overflows. However, there are two differences to the Time-out Period after reset. First, the input frequency can be either f_{SYS}/256 or f_{SYS}/16384, depending on the programmed state of bit WDT_SR.WDTIS before the Time-out Period was entered. Second, because there is no synchronization of the clock divider to the mode transitions of the Watchdog, the next clock pulse, incrementing the counter to $FFFD_H$, may come after one clock divider period, or immediately after the counter was initially set to $FFFC_H$. Thus, the minimum duration of the Time-out Period in the latter case will only be three counter clocks. The possible minimum and maximum periods are given in **Table 14-8**.

Table 14-8 Time-out Period During Normal Operation

WDTIS	Min./ Max.	Period	Example @ f_{SYS} = 80 MHz
0	min.	$3 \times 16384/f_{SYS} = 49152/f_{SYS}$	0.61 ms
	max.	$4 \times 16384/f_{SYS} = 65536/f_{SYS}$	0.82 ms
1	min.	$3 \times 256/f_{SYS} = 768/f_{SYS}$	9.6 µs
	max.	$4 \times 256/f_{SYS} = 1024/f_{SYS}$	12.8 µs

The WDT input clock rate cannot be changed during the Time-out Period. The control bit for the input clock rate, WDT_SR.WDTIS, is loaded from WDT_CON1.WDTIR when WDT_CON0.ENDINIT is set to 1, that is, after Time-Out Mode has been properly exited. Hence, the new input frequency will become effective only in the subsequent Time-out Period.

Note: In Prewarning Mode, the device does not need to wait for the end of the Time-out Period and the reset. After having saved the required state in the NMI routine, software can execute a soft reset to shorten the time. However, the state of the Watchdog Status Register should also be saved in this case, since the error flags contained in it will be cleared due to the soft reset (this is not the case if the Watchdog reset is awaited).

14.4.7.2 Normal Period

The duration of Normal Mode can be varied by two parameters, the input clock and the reload value.

The system clock, f_{SYS}, can be divided by either 256 or 16384. WDT_SR.WDTIS selects the input clock divider. The default value of WDTIS after reset is 0, corresponding to a frequency of f_{SYS}/16384.

When the WDT is serviced in Normal Mode, it is reloaded with the 16-bit reload value, WDT_CON0.WDTREL.

The WDT Period can be varied over a wide range with these two parameters. Again, since there is no synchronization of the clock divider to the mode transitions of the Watchdog, the next clock pulse, incrementing the counter, may come after one clock divider period, or immediately after the counter was reloaded. Thus, it is recommended that the reload value is programmed to a value that results to one clock pulse more than the required period. Using a reload value of $FFFF_H$ could, therefore, lead to an immediate overflow of the timer. Thus, the examples given in **Table 14-9** are only shown with a maximum reload value of $FFFE_H$.

Table 14-9 Timer Periods in Normal Mode

WDTIS	Reload Value	Min./ Max.	Period	Example @ f_{SYS} = 80 MHz
0	0000_H	min.	$65535 \times 16384/f_{SYS} = 1073725440/f_{SYS}$	13.4 s
		max.	$65536 \times 16384/f_{SYS} = 1073741824/f_{SYS}$	13.4 s
	$FFFE_H$	min.	$1 \times 16384/f_{SYS} = 16384/f_{SYS}$	205 µs
		max.	$2 \times 16384/f_{SYS} = 32768/f_{SYS}$	410 µs
1	0000_H	min.	$65535 \times 256/f_{SYS} = 16776960/f_{SYS}$	210 ms
		max.	$65536 \times 256/f_{SYS} = 16777216/f_{SYS}$	210 ms
	$FFFE_H$	min.	$1 \times 256/f_{SYS} = 256/f_{SYS}$	3.2 µs
		max.	$2 \times 256/f_{SYS} = 512/f_{SYS}$	6.4 µs

14.4.7.3 WDT Period During Power-Saving Modes

Care must be taken when programming the WDT reload value before going to Idle or Sleep Mode. As described on **Page 14-17**, the state of bit 15 of the Watchdog counter is used to wake-up from these modes. Thus, the reload value should be chosen such that it is less than $7FFE_H$ (bit 15 = 0); otherwise an immediate wake-up could occur. Only half of the maximum periods shown in **Table 14-9** can be used for the wake-up period.

14.5 Handling the Watchdog Timer

This section describes methods of handling the WDT function.

14.5.1 System Initialization

After any reset, the WDT is put in Time-Out Mode, and WDT_CON0.ENDINIT is 0, providing access to sensitive system registers. Changes to the operation of the WDT controlled by register WDT_CON1 become effective only after WDT_CON0.ENDINIT has been set to 1 again. Thus, changes to the WDT mode bits in WDT_CON1 do not interfere with the Time-out operation of the WDT after reset. **Table 14-10** shows the default contents of the WDT registers.

Table 14-10 Watchdog Timer Default Values After Reset

Register	Default Contents	Description
WDT_CON0	FFFC 0002$_H$	Reload value is FFFC$_H$, WDTPW is 0; WDT_CON0 is locked (WDTLCK = 1); ENDINIT is 0.
WDT_CON1	0000 0000$_H$	Watchdog Timer disable request is 0; input clock request set to f_{SYS}/16384.
WDT_SR	FFFC 001U$_H$	The Watchdog counter contains FFFC$_H$ (the initial Time-out value); WDT is operating in Time-Out Mode (WDTTO = 1); WDT is enabled (WDTDS = 0); input clock is f_{SYS}/16384. Bits WDTOE and WDTAE are set to 0 after a power-on, a hard or a soft reset. In case of a reset caused by the WDT, these two bits are set depending on the error condition that caused the Watchdog reset.

Because the WDT is in Time-Out Mode after reset, WDT_CON0.ENDINIT must be set to 1 before the Time-out Period expires. This means that initialization of ENDINIT-protected system registers must be complete before the expiration of the Time-out Period, defined on **Page 14-18**. To set WDT_CON0.ENDINIT to 1, a Valid Password Access to WDT_CON0 must be performed first. During the subsequent Valid Modify Access, WDT_CON0.ENDINIT must be set to 1, which will exit Time-Out Mode. The WDT is switched to the operation determined by the new values of WDTIS and WDTDS.

Note: The action described above must absolutely be performed during initialization of the device to properly terminate this mode. Even if the Watchdog function will not be used in an application and the WDT will be disabled, a valid access sequence to the WDT is mandatory. Otherwise, the Watchdog counter will overflow, Prewarning Mode will be entered, and a Watchdog reset will occur at the end of the Time-out Period.

Bit fields WDT_CON0.WDTREL and WDT_CON0.WDTPW can optionally be changed during the Valid Modify Access, but it is not required. WDT_CON0.ENDINIT can be set to 1 or 0; however, setting ENDINIT to 0 does not stop Time-Out Mode. Any values written to WDTREL, WDTPW, and ENDINIT are stored in WDT_CON0, and WDT_CON0 is automatically locked (WDTLCK = 1) after the Modify Access is finished.

14.5.2 Re-opening Access to Critical System 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 set to 0. 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 to 1 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.WDTDS = 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.

14.5.3 Servicing the Watchdog Timer

If the WDT is used in an application and is enabled (WDT_SR.WDTDS = 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 occur.

During the next Modify Access, it is strictly required that WDT_CON0.ENDINIT as well as bit 1 and bits [7:4] be written with 1, while bits [3:2] be 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 WDTREL or the user-definable password WDTPW are not required. However, changing WDTPW is recommended so that software can monitor WDT service operations throughout the duration of an application program (see next section).

If 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.

14.5.4 Handling the User-Definable Password Field

WDT_CON0.WDTPW is an 8-bit field that can be set by software to any arbitrary value during a Modify Access. Setting of this field has no effect on the operation of the WDT, other than the role it plays in forming the password bit pattern, as discussed on Page 14-10.

The purpose of this field is to support further enhancements to the password protection scheme. For the following description, it is assumed that software does at least not fully compute the value for the Password Access from the content of registers WDT_CON0 and WDT_CON1, but uses a predefined constant, embedded in the instruction stream, for the password (this is at least necessary for the user-definable password field WDTPW). For example, software can modify this bit field each time it executes a Watchdog service sequence. The next service sequence needs to take this new value into account for its Password Access, and it again changes the value during its Modify Access. Up to 256 different password values can be used. In this way, each service sequence is unique. If a malfunction occurs that, for instance, would result in the omission of one or more of these service sequences, the next service sequence would most probably not write the correct password. This service sequence would rely on the password value programmed during the normally preceding service sequence. However, if this one was skipped, the password value required by the contents of the Watchdog registers is the one programmed at the last service sequence executed before the malfunction had occurred. A Watchdog error condition would be detected in this case.

In the same manner, the Watchdog would detect the malfunction if a service sequence would be executed twice due to a falsely performed jump. **Figure 14-4** illustrates these examples.

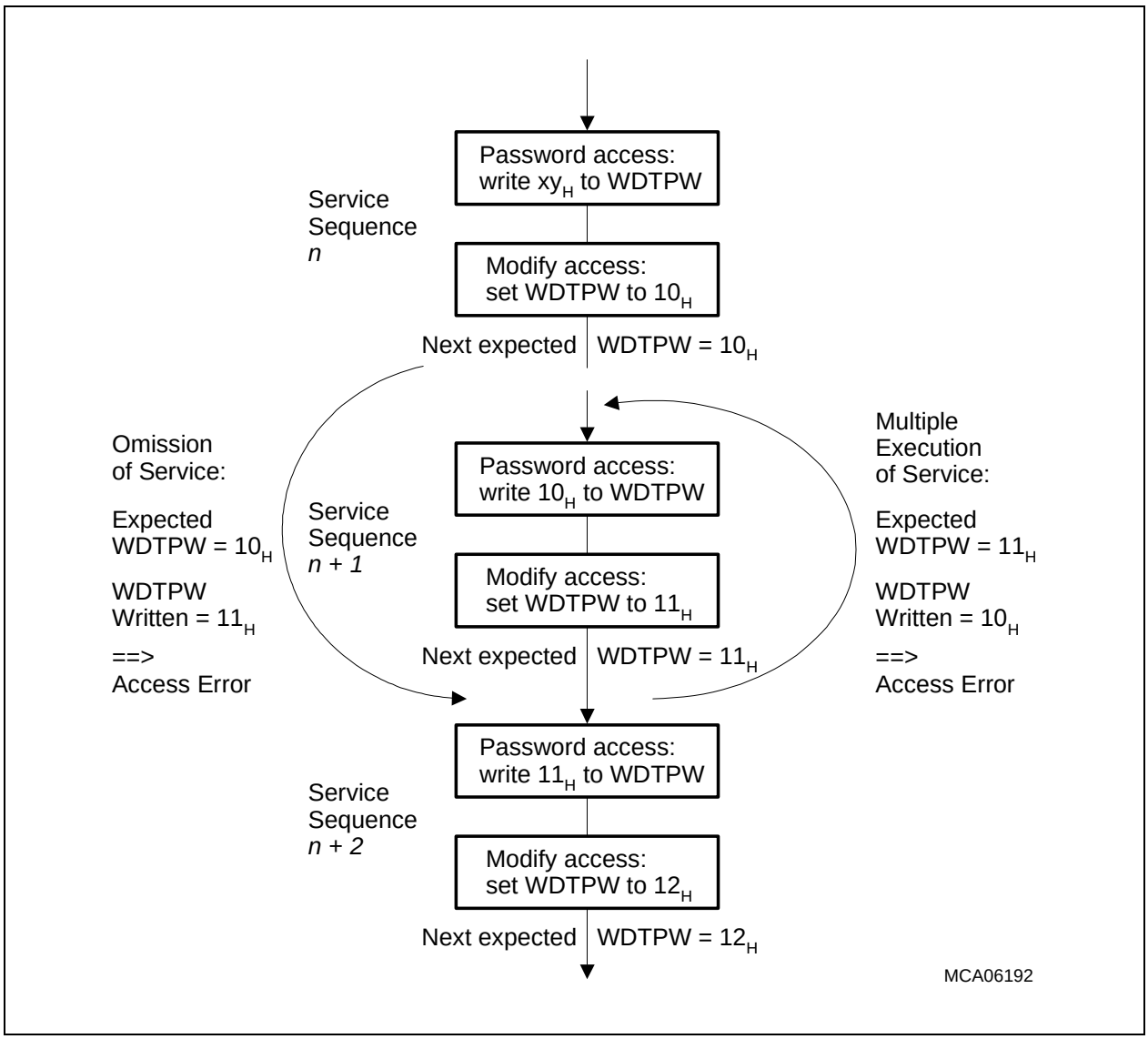


Figure 14-4 Detection of False Jumps and Loops

Other schemes are possible. Consider the case in which a routine determines some conditions that alter the program flow. One of two or more different paths will be executed next, depending on these conditions. Before branching to the appropriate routine(s), software performs a Watchdog service and sets the new password value for WDTPW such that it depends on these conditions, that is, some or all of these condition codes can be incorporated into WDTPW. The next service sequence is performed at the point where the different paths come together again. To determine the correct password, software uses a value returned from the path that was executed. This value must match the value in WDTPW, otherwise it means that the wrong path was executed. An example is shown in **Figure 14-5**.

It is also possible to have the different paths of a program compute the full or partial password to unlock register WDT_CON0. The password will only match at the next service sequence if all the expected paths and calculation routines have been executed properly. If one or more steps were omitted or a wrong path was executed due to a malfunction, the Watchdog failure mechanism will detect this and issue a reset of the device (after the prewarning phase).

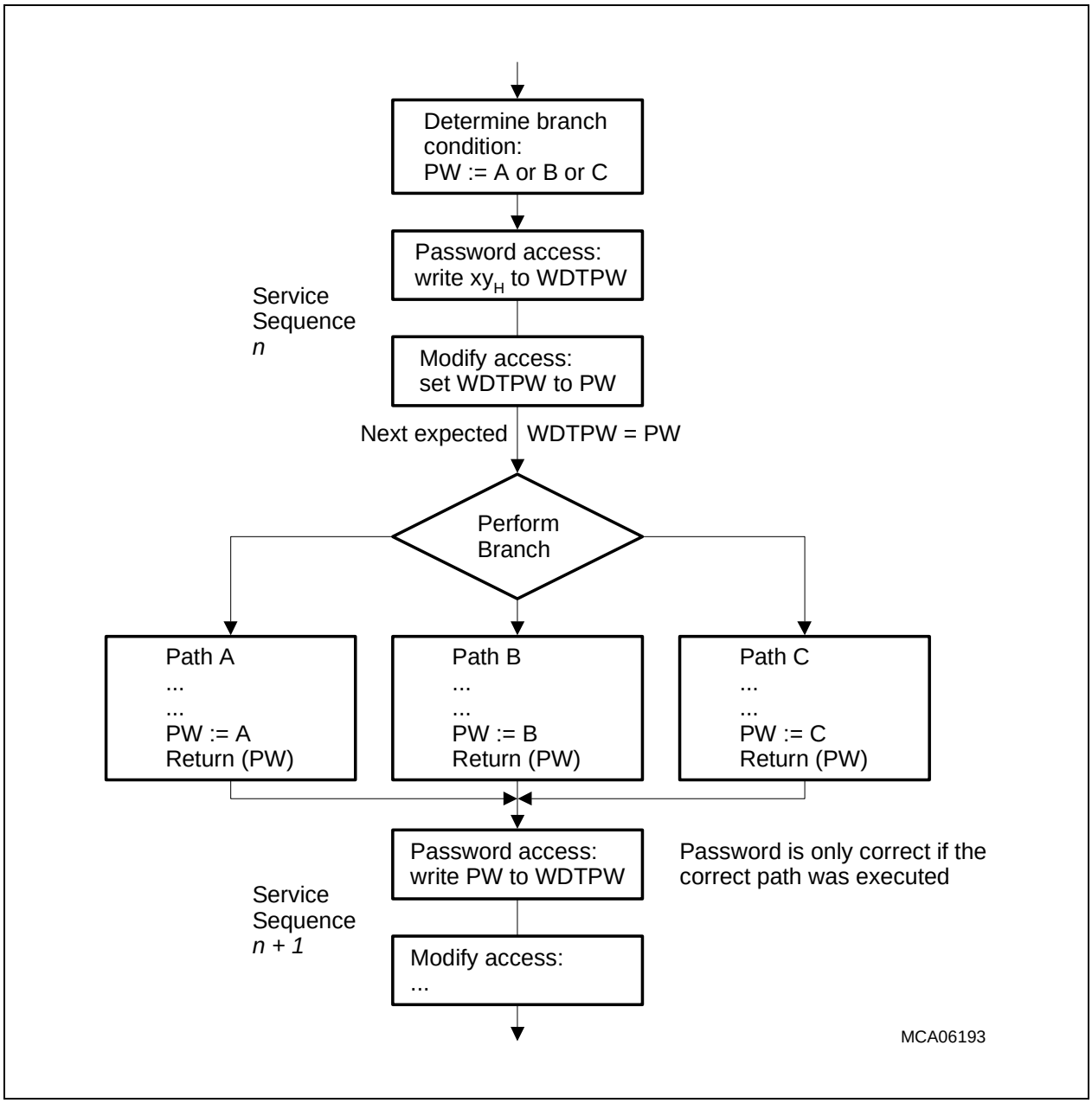


Figure 14-5 Monitoring Program Sequences

14.5.5 Determining the Required Values for a WDT Access

As described on **Page 14-10** and **Page 14-11**, the values required for the Password And Modify Accesses to register WDT_CON0 are designed such that they can be derived from the values read from registers WDT_CON0 and WDT_CON1. However, at least some bits have to be modified in order to get the correct write value. This makes it very unlikely that a false operation derives values from reading these registers that inadvertently affect the WDT operation when written back to WDT_CON0. Even if a false write operation would have written the correct password to WDT_CON0, one further, different correct value needs to be written to this register in order to have an effect. In addition, the WDT switches to Time-Out Mode after the Valid Password Access, providing only a time-limited window for the second access.

Although computing the required values from the current contents of the Watchdog registers is one option, using predetermined values that are set at compile-time of the program, may be the better approach in many cases. Usually, handling the WDT is performed by one and only one task. Thus, the problem will not occur that another task might have changed some of the parameters which must not be modified (which would require reading the contents, modifying the value appropriately, and then writing it back). The one task handling the Watchdog Timer function would always “know” how it has programmed the WDT last time, and would therefore also “know” the next password value for opening WDT_CON0. In fact, this method would actually detect the case if another task had illegally modified the Watchdog registers, since the predetermined password might not work anymore, and a Watchdog error condition would thus be generated.

In addition, accessing the WDT with predetermined values has the obvious benefit of shorter code, as no computing steps need to be performed.

14.6 Watchdog Timer Registers

This section describes the WDT registers. The WDT is provided with three registers: WDT_CON0, WDT_CON1, and WDT_SR, as shown in **Figure 14-6**. They are located inside the SCU module's address range.

WDT Registers Overview

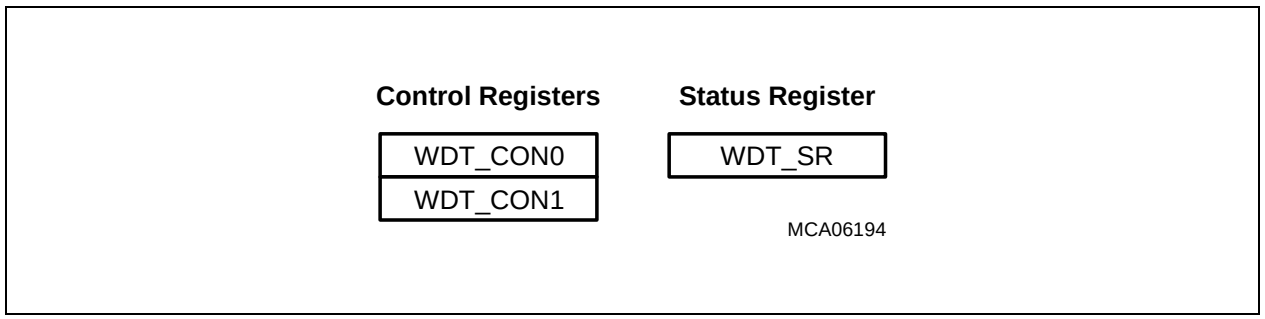


Figure 14-6 Watchdog Registers

The complete and detailed address map of the SCU which includes the WDT registers is shown in **Table 16-3** on **Page 16-7**.

Table 14-11 Registers Address Space

Module	Base Address	End Address	Note
WDT	F000 0000$_H$	F000 00FF$_H$	-

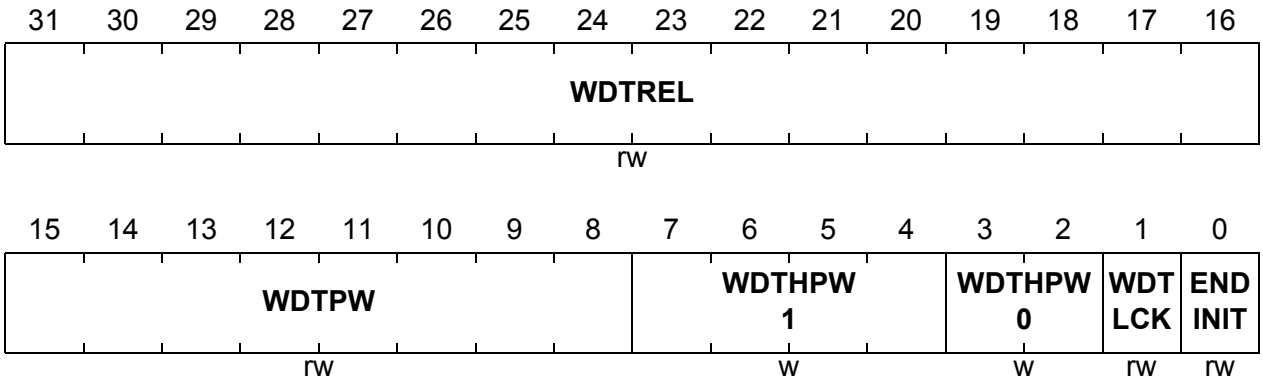
Table 14-12 Registers Overview - WDT Kernel Registers

Register Short Name	Register Long Name	Offset Address	Description see
WDT_CON0	Watchdog Timer Control Register 0	0020$_H$	**Page 14-29**
WDT_CON1	Watchdog Timer Control Register 1	0024$_H$	**Page 14-31**
WDT_SR	Watchdog Timer Status Register	0028$_H$	**Page 14-32**

14.6.1 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
Watchdog Timer Control Register 0 (20_H) Reset Value: FFFC 0002_H



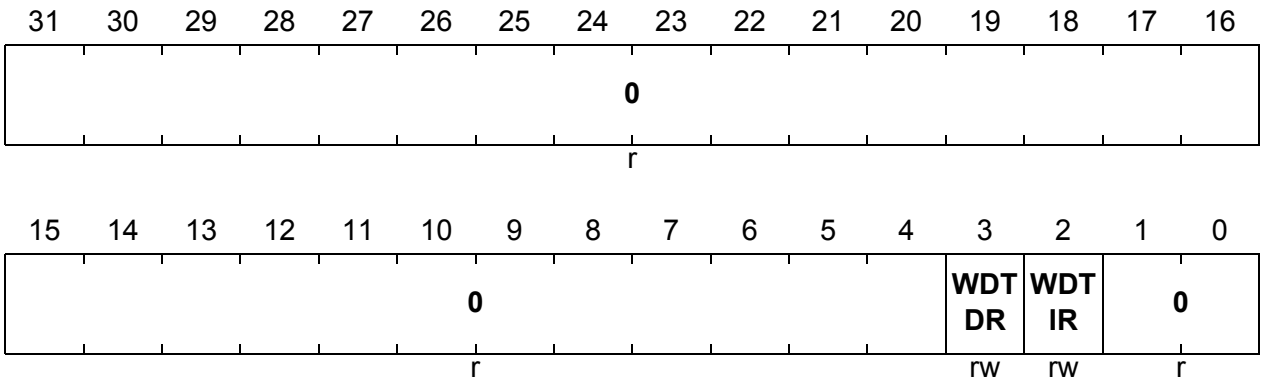
Field	Bits	Type	Description
ENDINIT	0	rw	**End-of-Initialization Control Bit** 0_B Access to Endinit-protected registers is permitted (default after reset). 1_B Access to Endinit-protected registers is not permitted. ENDINIT controls the access to critical system registers. During a Password Access it must be written with its current value. It can be changed during a Modify Access to WDT_CON0.

Field	Bits	Type	Description
WDTLCK	1	rw	**Lock Bit to Control Access to WDT_CON0** 0_B Register WDT_CON0 is unlocked. 1_B Register WDT_CON0 is locked (default after reset). The actual value of WDTLCK is controlled by hardware. It is set to 0 after a successful Password Access to WDT_CON0, and automatically set to 1 again after a successful 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 set to 0 during a Password Access to WDT_CON0, and set to 1 during a Modify Access to WDT_CON0. That is, the inverted value read from WDTLCK always must be written to itself.
WDTHPW0	[3:2]	w	**Hardware Password 0** This bit field must be written with the value of the bits WDT_CON1.WDTDR and WDT_CON1.WDTIR 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.
WDTHPW1	[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.
WDTPW	[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.
WDTREL	[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 reset).

14.6.2 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
Watchdog Timer Control Register 1 (24_H) Reset Value: 0000 0000_H



Field	Bits	Type	Description
WDTIR	2	rw	**WDT Input Frequency Request Control Bit** 0_B Request to set input frequency to f_{SYS}/16384. 1_B Request to set input frequency to f_{SYS}/256. This bit can only be modified if WDT_CON0.ENDINIT is set to 0. WDT_SR.WDTIS is updated by this bit only when ENDINIT is set to 1 again. As long as ENDINIT is left at 0, WDT_SR.WDTIS controls the current input frequency of the Watchdog Timer. When ENDINIT is set to 1 again, WDT_SR.WDTIS is updated with the state of WDTIR.
WDTDR	3	rw	**WDT 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 set to 0. WDT_SR.WDTDS is set to this bit's value when ENDINIT is set to 1 again. As long as ENDINIT is left at 0, bit WDT_SR.WDTDS controls the current enable/disable status of the WDT. When ENDINIT is set to 1 again with a Valid Modify Access, WDT_SR.WDTDS is updated with the state of WDTDR.

Field	Bits	Type	Description
0	[1:0], [31:4]	r	**Reserved** Read as 0; should be written with 0.

14.6.3 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.

The reset value for this register depends on the cause of the reset. For any reset other than a Watchdog reset, the reset value is FFFC 001U_H. After a Watchdog reset, bits WDTAE and WDTOE indicate the type of Watchdog error that occurred before the Watchdog reset. Either one or both bits can be set. These bits are not cleared on a Watchdog reset. Bits WDTDS and WDTIS are always 0 after any reset.

WDT_SR
Watchdog Timer Status Register (28_H) Reset Value: FFFC 0010$_H$

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0										WDTPR	WDTTO	WDTDS	WDTIS	WDTOE	WDTAE
r										rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
WDTAE	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 by hardware when an illegal Password Access or Modify Access to register WDT_CON0 was attempted. This bit is only cleared by: • A power-on, hardware, or software reset occurs. • 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, indicated by WDT_SR.WDTPR = 1.
WDTOE	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 by hardware when the WDT overflows from $FFFF_H$ to 0000_H. This bit is only cleared when: • A power-on, hardware, or software reset occurs; • 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, indicated by WDT_SR.WDTPR = 1.
WDTIS	2	rh	**Watchdog Input Clock Status Flag** 0_B WDT input clock is f_{SYS}/16384 (default after reset). 1_B WDT input clock is f_{SYS}/256. This bit is updated with the state of bit WDT_CON1.WDTIR after WDT_CON0.ENDINIT is written with 1 during a Valid Modify Access to register WDT_CON0.
WDTDS	3	rh	**Watchdog Enable/Disable Status Flag** 0_B WDT is enabled (default after reset). 1_B WDT is disabled. This bit is updated with the state of bit WDT_CON1.WDTDR after WDT_CON0.ENDINIT is written with 1 during a Valid Modify Access to register WDT_CON0.

Field	Bits	Type	Description
WDTTO	4	rh	**Watchdog Time-Out Mode Flag** 0_B Normal mode. 1_B The Watchdog is operating in Time-Out Mode (default after reset). This bit is set to 1 when Time-Out Mode is entered, automatically after a reset and after every Password Access to register WDT_CON0. It is automatically cleared by hardware when Time-Out Mode is properly terminated through a Valid Modify Access to WDT_CON0. It is left set when a Watchdog error occurs during Time-Out Mode, and Prewarning Mode is entered.
WDTPR	5	rh	**Watchdog Prewarning Mode Flag** 0_B Normal mode (default after reset) 1_B The Watchdog is operating in Prewarning Mode This bit is set to 1 when a Watchdog error is detected. The WDT has issued an NMI trap and is in Prewarning Mode. A reset of the chip occurs after the prewarning period has expired.
WDTTIM	[31:16]	rh	**WDT Value** Reflects the current content of the WDT.
0	[15:6]	r	**Reserved** Read as 0.

15 On-Chip Debug Support

This chapter gives an overview of the debug features of the TC1766 device. This chapter does not describe the TC1766 debug functionality and capabilities in detail. For detailed information about the On-Chip Debug Support (OCDS) functionality (for example, required by tool suppliers), please contact local Infineon representatives.

15.1 Overview

The TC1766 supports three levels of debug operation:

- OCDS Level 1
- OCDS Level 2
- OCDS Level 3

OCDS Level 1

The OCDS Level 1 is mainly assigned for real-time software debugging purposes which have a demand for low-cost standard debugger hardware.

The OCDS Level 1 is based on a JTAG interface that is used by the external debug hardware to communicate with the system. The on-chip Cerberus module controls the interactions between the JTAG interface and the on-chip modules. The external debug hardware may become master of the internal buses, and read or write the on-chip register/memory resources. The Cerberus also makes it possible to define breakpoint and trigger conditions as well as to control user program execution (run/stop, break, single-step).

OCDS Level 2

The OCDS Level 2 makes it possible to implement program tracing capabilities for enhanced debuggers by extending the OCDS Level 1 debug functionality with an additional 16-bit wide trace output port with trace clock. With the trace extension, the following four trace capabilities provided (only one of the four trace capabilities can be selected at a time):

- Trace of the CPU program flow
- Trace of the PCP program flow
- Trace of the DMA Controller transaction requests
- Trace of the DMA Controller Move Engine status information

OCDS Level 3

The OCDS Level 3 is based on a Multi Core Debug Solution (MCDS) using a special TC1766 emulation device, the TC1766ED. This device has additional features required for high-end emulation purposes. The TC1766ED includes the TC1766 product chip and additional emulation extension hardware in a package with the same footprint as the

TC1766. For detailed information about the TC1766ED functionality (e.g. required by high-end emulation manufacturers), please contact local Infineon representatives.

Figure 15-1 shows a block diagram of the TC1766 OCDS based system.

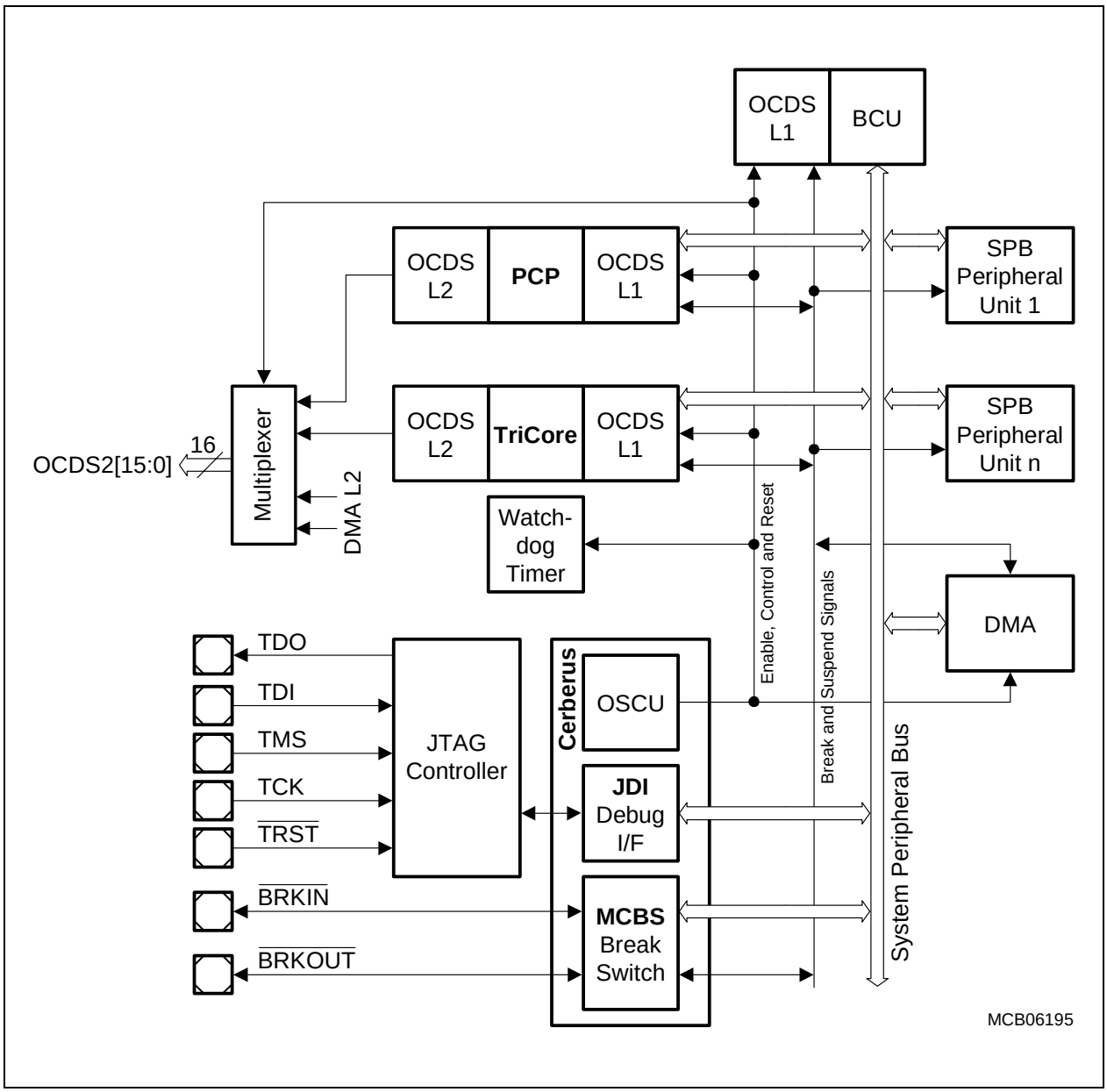


Figure 15-1 OCDS Based System Block Diagram

Components

The OCDS of the TC1766 consists of the following building blocks:

- OCDS Level 1 module of TriCore
- OCDS Level 2 interface of TriCore
- OCDS Level 1 module of PCP
- OCDS Level 2 interface of PCP
- OCDS Level 1 module in the BCU of the System Peripheral Bus (SBCU)
- OCDS Level 1 facilities within the DMA controller
- OCDS Level 2 interface of the DMA controller
- Cerberus - OCDS System Control Unit (OSCU)
- Cerberus - Multi-Core Break Switch (MCBS)
- Cerberus - JTAG Debug Interface (JDI)
- Suspend functionality of the peripherals

Summary of OCDS Features

- TriCore Level 1 OCDS
  - Hardware event generation
  - Break by DEBUG instruction 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 Level 1 OCDS:
  - Break by DEBUG instruction or Break signal from break switch
  - Concurrent access to memory and SFRs via Cerberus
- DMA Level 1 OCDS:
  - Break request on error
  - Event generation on specified channel activity
  - Suspending pre-selected channels
- BCU Level 1 OCDS:
  - Event generation on specified transactions
- 16-pin Level 2 trace port; outputs either TriCore, PCP or DMA trace
- OCDS System Control Unit (Cerberus OSCU):
  - Extensive control using only a minimum number of pins (no pins apart from JTAG)
  - Connecting a debugger to a running system allowed (hot attach)
  - Built in System security
  - Optional halt after reset
  - Debug resources not affected by system reset
- Multi-Core Break Switch (Cerberus MCBS):
  - TriCore, PCP, DMA, break pins and BCUs 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 main philosophy of the TriCore OCDS Level 1 is that the complete architecture and the status of a target system are visible from the FPI Bus. This means that every component of the system can be accessed through its mapping into the FPI address space, including on-chip memories, CPU core registers and register of the peripheral units.

A typical OCDS Level 1 debugging configuration is shown in **Figure 15-2**. It includes two parts:

1. The debugger software, supporting a standard JTAG protocol via a PC port
2. The debugger hardware interface adapter, connecting the TC1766 JTAG interface in the target system with the PC port (parallel, serial, or USB)

This configuration makes it possible to realize a cheap debugging environment that permits comprehensive real-time debugging tasks to be performed.

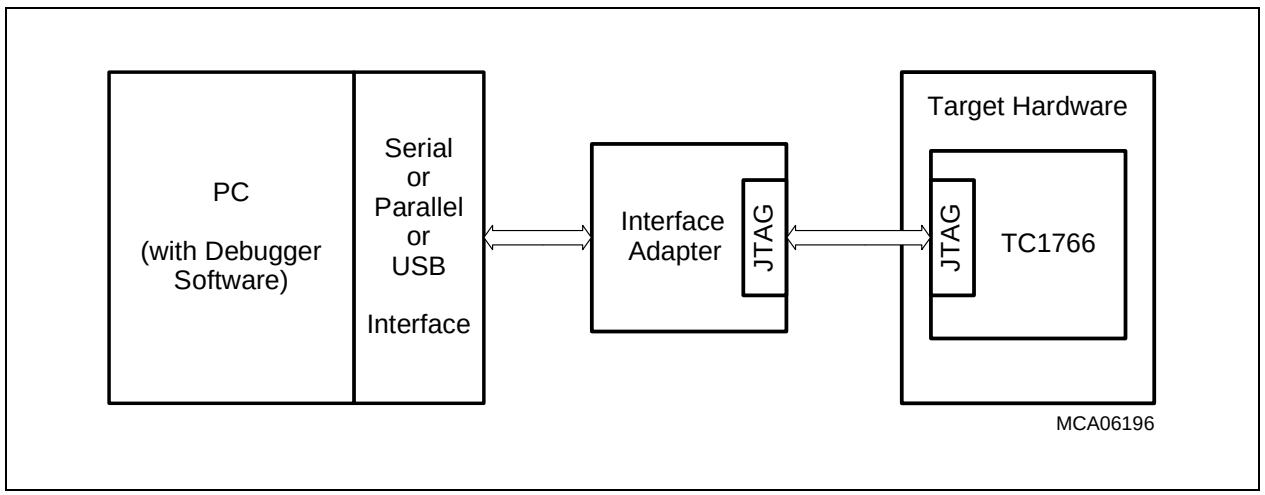


Figure 15-2 Typical OCDS Level 1 Hardware Connections

15.2.1 TriCore CPU Level 1 OCDS

The TriCore CPU provides the following OCDS Level 1 features:

- Full single-step support, by hardware as well as by software (code patching)
- Up to 4 programmable hardware breakpoints:
  Each one can be defined as a combination of instruction pointer value and memory/SFR address/value:
  – 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/address values or one data/address range
    No break before make possible (due to pipelined architecture)
  - Combinations of the above break conditions
- 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 TC1766 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.

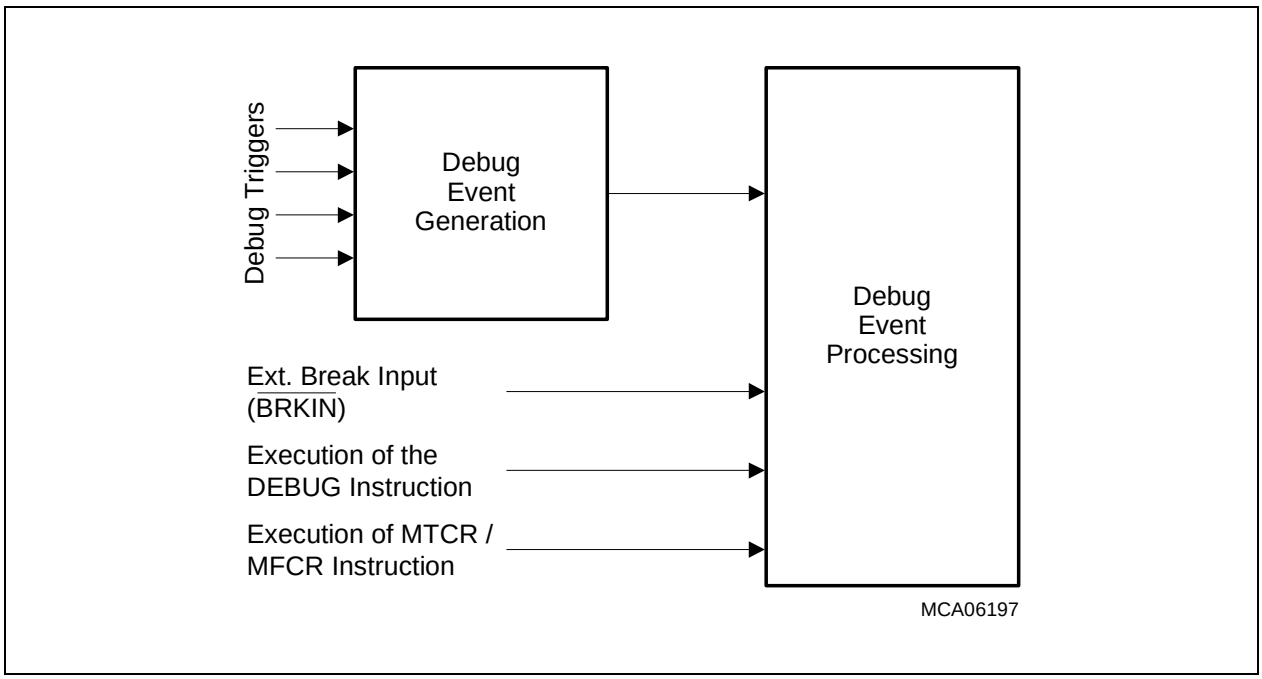


Figure 15-3 Basic TriCore Debug Concept

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 break input pin $\overline{\text{BRKIN}}$
- 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 Break Input Pin $\overline{\text{BRKIN}}$

When activating the TC1766 device pin $\overline{\text{BRKIN}}$ = 0, the MCBS unit induces a break event as specified in an External Break Input Event Specifier Register EXEVT.

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:

- DBGSR or
- CREVT or
- SWEVT or
- EXEVT or
- TR0EVT or
- TR1EVT

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 $\overline{\text{BRKOUT}}$ signals by the MCBS unit
- 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.

15.2.1.4 TriCore OCDS Registers

Figure 15-4 shows the TriCore OCDS registers in an overview.

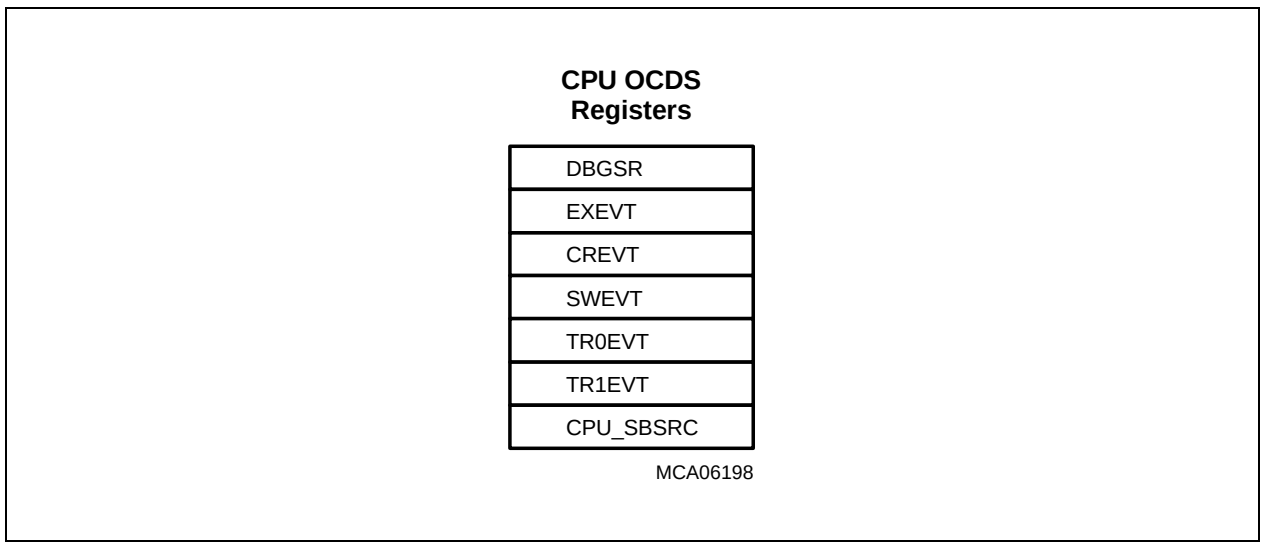


Figure 15-4 TriCore Core Debug Registers

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 Specifier Register	F7E1 $FD20_H$
TR1EVT	Trigger Event 1 Specifier Register	F7E1 $FD24_H$
CPU_SBSRC	CPU Software Break 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

The PCP has no 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. Additionally the Break Switch can send an external break request to the PCP, with the same consequence (Debug Exit or Error Exit).

15.2.3 BCU OCDS Level 1

The BCU of the FPI bus in the TC1766 support OCDS Level 1 offers very comfortable and powerful means for breakpoint generation.

The BCU contains one comparator 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 6** starting from section **“SBCU Registers” on Page 6-35**.

15.2.4 DMA OCDS Level 1

The DMA controller in the TC1766 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 11** in section **“On-Chip Debug Capabilities” on Page 11-23**.

15.3 OCDS Level 2 Debugging via Trace Port

The OCDS Level 2 debug support extends the OCDS Level 1 debug functionalities with an additional 16-bit wide trace port TR[15:0] with the trace clock TRCLK. The trace port extension makes it possible to output one out of four types of trace output signals:

- TriCore CPU OCDS Level 2 trace
- PCP OCDS Level 2 trace
- DMA Controller channel transaction request trace
- DMA Controller move engine trace

The trace output port is controlled by the OSCU. The trace data is always output at CPU clock speed ($f_{TRCLK} = f_{CPU}$).

15.3.1 TriCore CPU and PCP OCDS Level 2 Trace

Every trace clock cycle, 16 bits of CPU/PCP trace information are sent out, representing the current state of the CPU/PCP cores. The trace output lines are grouped into three parts:

- 5 bits of pipeline status information
- 8-bit indirect PC bus information
- 3 bits of breakpoint qualification information

With this information, an external emulator can reconstruct a cycle-by-cycle image of the instruction flow through the CPU or PCP. The trace information can be captured by the external debugger hardware and used to rebuild later on (off-line, using the source code) a cycle accurate disassembly of the code that has been executed. It is also possible to follow in real-time the current PC, facilitating advanced tools such as profilers, coverage analysis tools etc.

15.3.2 DMA OCDS Level 2 Trace

The DMA controller provides two sources for OCDS Level 2 tracing:

- DMA Controller channel transaction request trace
- DMA Controller move engine trace

More details on the OCDS Level 2 debug trace capabilities of the DMA controller are described in detail in section **“Trace Signal Generation” on Page 11-25**.

15.3.3 Concurrent Debugging

A limited concurrent debugging is possible for CPU and DMA.

It is not possible to trace CPU and DMA at the same time. But when the trace port is assigned to the DMA Controller, the break-in and break-out features of the CPU can be used at the same time. Selecting DMA trace and enabling the break-out lines of both units, makes it possible to distinguish between DMA trace events and CPU break-out activation:

- During DMA trace, the $\overline{\text{BRKOUT}}$ output becomes activated only together with one of the trace port pins being active (‘1’) as well.
- If $\overline{\text{BRKOUT}}$ output becomes activated with all trace pins being zero, this means that the CPU has activated its break-out signal.

15.4 Debug Interface (Cerberus)

The Cerberus module is the on-chip unit that controls all OCDS Levels 1 and 2 main debug functions. Generally, the Cerberus should 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
- JTAG Debug Interface - JDI
- Multi Core Break Switch - MCBS

A standard JTAG interface is connected via the JTAG controller with the JDI. Two pins are available to handle an external break condition. An external debug hardware can access the Cerberus registers and arbitrary memory locations across the System Peripheral (FPI) Bus.

Features

- 5-pin standard JTAG interface for OCDS Level 1 control
- Generation of external break condition via pins $\overline{\text{BRKIN}}$/$\overline{\text{BRKOUT}}$
- Full access to the complete SPB (FPI) Bus address space via JTAG
- No user resources (hardware/software) are required
- Minimum run-time impact
- 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 JTAG
- Control of the OCDS blocks
- Data acquisition

15.4.1 RW Mode

As the name implies, the RW mode is used by a JTAG host to read or write arbitrary memory locations via the JTAG interface. The RW mode needs the FPI 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 an FPI Bus error, a dedicated JTAG instruction is provided (IO_READ_HWORD).

- BYTE (8-bit): If the host wants to read a byte, it has to read the associated word or half-word. Then the JTAG host has to extract the part needed itself.

Writing bytes or half-words are supported with the IO_WRITE_BYTE and IO_WRITE_HWORD JTAG instructions. With these instructions, the JTAG host must again shift in the full 32-bit word, but only the selected byte or half-word is actually written. Its exact position is defined by the two lowest address bits in register IOADDR.

15.4.2 Communication Mode

In Communication Mode, the Cerberus has no access to the FPI Bus and communication is established between the external 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 JTAG host is master of all transactions. He 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 the Cerberus, but it sets request bits in the CPU accessible IOSR register to signal the monitor that the debugger wants to send (IO_WRITE_WORD) or receive (IO_READ_WORD) a value. The software monitor has to poll register IOSR. The IOADDR register is not used.

15.4.3 Triggered Transfers

Triggered transfers are an OCDS-specific feature of the Cerberus. They can be used to read from or write to a certain memory location when an OCDS trigger becomes active. Triggered Transfers behave like normal transfers with some exceptions.

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.

15.4.4 Multi Core Break Switch

In the TC1766, there are two main processor units, the CPU and the PCP2. For debugging purposes, the OCDS run control of one processor unit can break (interrupt) the other processor unit or vice versa. This run control tasks are handled by the MCBS unit which is a part of the Cerberus. **Figure 15-5** shows the break signal interfaces of this MCBS unit.

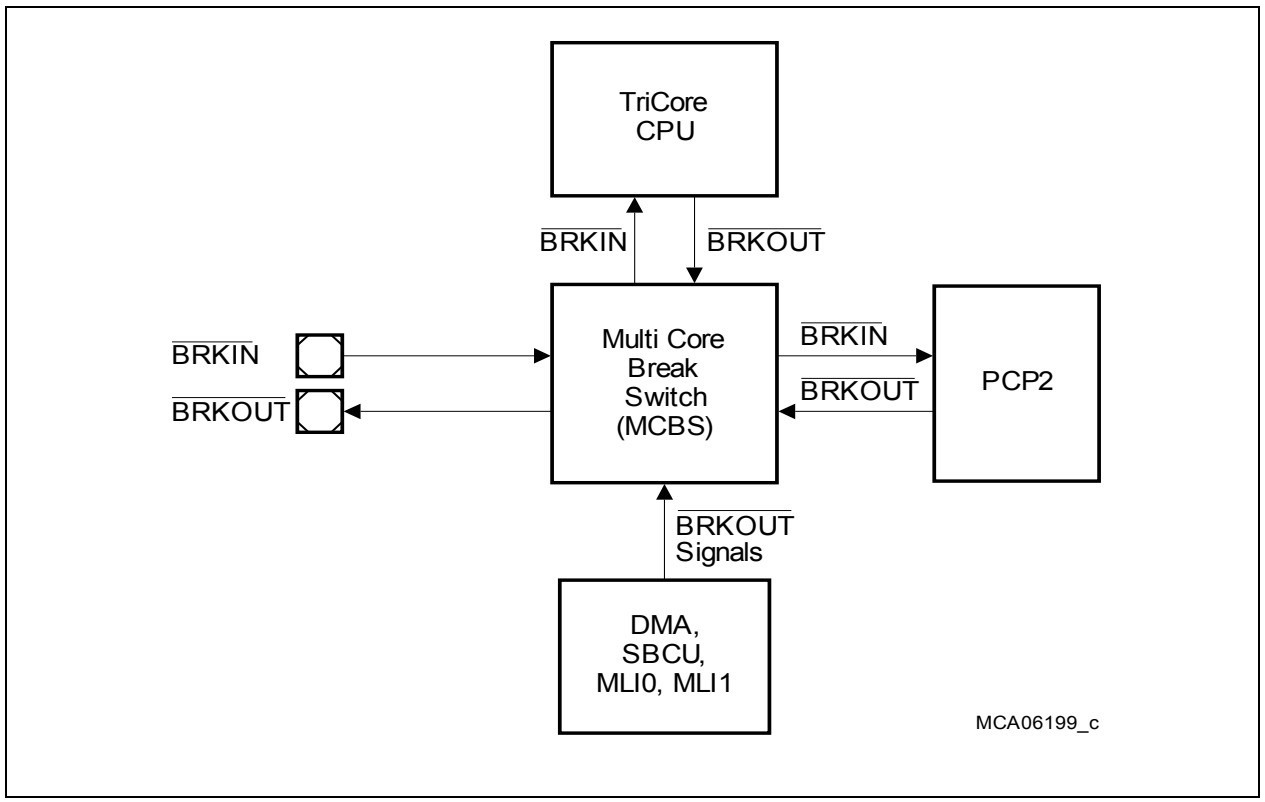


Figure 15-5 Break Switch Interfaces

The MCBS unit supports the following features:

- Two independent break-out master units (TriCore, PCP2)
- Six break-in sources (TriCore, PCP, DMA, SBCU, MLI0, MLI1)
- Two port pins, BRKIN and BRKOUT
- 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

15.5 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.

On the other hand, the TC1766 OCDS is designed to support complex multi-core/debugging environments. The challenge here is that several debugger applications may have to share a single resource, i.e. the same JTAG interface. This becomes even more complicated because the JTAG module contains the IEEE 1149.1 JTAG state machine, which must be handled in the correct manner.

The solution to this problem is the JTAG driver with its JTAG-API (Application Programming Interface). It allows several debugger applications to share the same JTAG interface. For example, it is possible to run a PCP debugger concurrently with a TriCore debugger on the same TC1766 device. In addition, the tool-specific PC interfaces like Ethernet, printer-port, or even USB can be hidden from the debugger software by the JTAG-API layer.

The JTAG-API enables the debugger vendor to ignore the complex task of understanding the JTAG module and supporting its functionality at low-level. All required information is provided as specifications and function references, ready for direct implementation into the tool-source, without the need to re-invent the wheel.

The maximum JTAG interface clock frequency (f_{TCK}) is 20 MHz. The following JTAG interface clock frequencies can be achieved:

Table 15-2 Cerberus Performance (Net Data Rates)

Operation	f_{TCK} = 200 kHz	f_{TCK} = 10 MHz	f_{TCK} = 20 MHz
Random Read	48 kbit/s	2.4 Mbit/s	4.6 Mbit/s
Random Write	50 kbit/s	2.5 Mbit/s	4.9 Mbit/s
Block Read	104 kbit/s	5.2 Mbit/s	10.0 Mbit/s
Block Write	114 kbit/s	5.7 Mbit/s	11.2 Mbit/s

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).

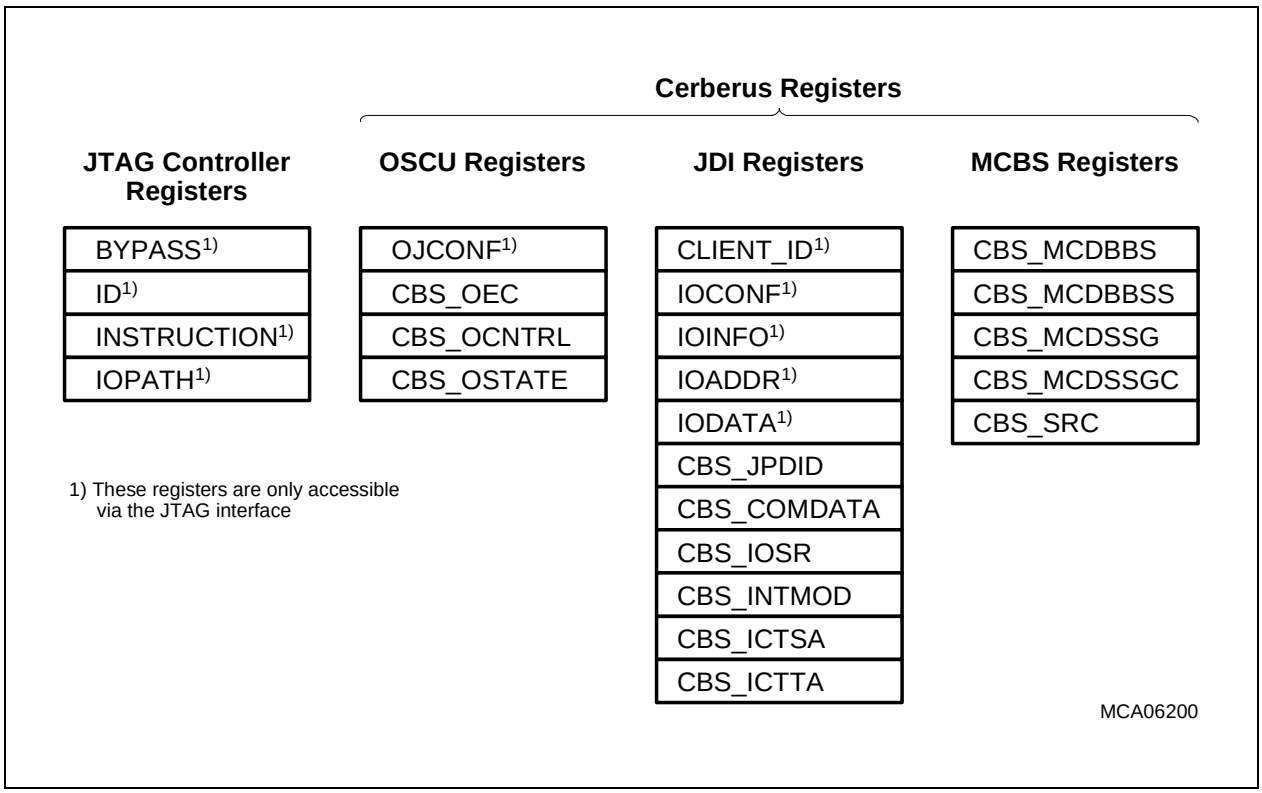


Figure 15-6 JTAG/Cerberus Register Overview

Table 15-3 JTAG/Cerberus Register Overview

Register Short Name	Register Long Name	Address
JTAG Controller Registers		
BYPASS	JTAG Bypass Register (1-bit)	1)
ID	JTAG Module 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)
CBS_OEC	Cerberus OCDS Enable Control Register	$F000\ 0478_H$
CBS_OCNTRL	Cerberus OSCU Configuration and Control Register	$F000\ 047C_H$

Table 15-3 JTAG/Cerberus Register Overview (cont'd)

Register Short Name	Register Long Name	Address
CBS_OSTATE	Cerberus OSCU Status Register	F000 0480$_H$
CLIENT_ID	Cerberus 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)
CBS_JDPID	Cerberus Module Identification Register	F000 0408$_H$
CBS_COMDATA	Cerberus Communication Mode Data Register	F000 0468$_H$
CBS_IOSR	Cerberus Status Register	F000 046C$_H$
CBS_INTMOD	Cerberus Internal Mode Status and Control Register	F000 0484$_H$
CBS_ICTSA	Cerberus Internal Controlled Trace Source Address Register	F000 0488$_H$
CBS_ICTTA	Cerberus Internal Controlled Trace Target Address Register	F000 048C$_H$
CBS_MCDBBS	Cerberus Break Bus Switch Configuration Register	F000 0470$_H$
CBS_MCDBBSS	Cerberus Break Bus Switch Status Register	F000 0490$_H$
CBS_MCDSSG	Cerberus Suspend Signal Generation Status and Control Register	F000 0474$_H$
CBS_MCDSSGC	Cerberus Suspend Signal Generation Configuration Register	F000 0494$_H$
CBS_SRC	Cerberus Service Request Control Register	F000 04FC$_H$

1) These registers are only accessible via the JTAG interface.

16 Register Overview

This chapter describes all registers of the TC1766 that are located in segment 15. It also describes the read/write access rights of the specific address ranges/registers.

Throughout the tables in this chapter, the “Access Mode” “Read” and “Write”, and “Reset Values” columns indicate access rights and values using symbols listed in **Table 16-1**.

Table 16-1 Address Map Symbols

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 that register/address range.
E	Endinit-protected register/address.
PW	Password protected register/address.
NC	No change, indicated register is not changed on a write operation.
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.
X	Undefined value or bit.

16.1 Address Map of Segment 15

Table 16-2 shows the block address map of Segment 15.

Table 16-2 Block Address Map of Segment 15

Unit	Address Range	Access Mode		Size
		Read	Write	
System Control Unit (SCU) and Watchdog Timer (WDT)	$F000\ 0000_H$ - $F000\ 00FF_H$	see Page 16-7		256 byte
System Peripheral Bus Control Unit (SBCU)	$F000\ 0100_H$ - $F000\ 01FF_H$	see Page 16-10		256 byte
System Timer (STM)	$F000\ 0200_H$ - $F000\ 02FF_H$	see Page 16-12		256 byte
Reserved	$F000\ 0300_H$ - $F000\ 03FF_H$	BE	BE	–
On-Chip Debug Support (Cerberus)	$F000\ 0400_H$ - $F000\ 04FF_H$	see Page 16-13		256 byte
Reserved	$F000\ 0500_H$ - $F000\ 07FF_H$	BE	BE	–
MicroSecond Channel 0 (MSC0)	$F000\ 0800_H$ - $F000\ 08FF_H$	see Page 16-15		256 byte
Reserved	$F000\ 0900_H$ - $F000\ 09FF_H$	BE	BE	–
Async./Sync. Serial Interface 0 (ASC0)	$F000\ 0A00_H$ - $F000\ 0AFF_H$	see Page 16-17		256 byte
Async./Sync. Serial Interface 1 (ASC1)	$F000\ 0B00_H$ - $F000\ 0BFF_H$	see Page 16-18		256 byte
Port 0	$F000\ 0C00_H$ - $F000\ 0CFF_H$	see Page 16-20		256 byte
Port 1	$F000\ 0D00_H$ - $F000\ 0DFF_H$	see Page 16-21		256 byte
Port 2	$F000\ 0E00_H$ - $F000\ 0EFF_H$	see Page 16-22		256 byte
Port 3	$F000\ 0F00_H$ - $F000\ 0FFF_H$	see Page 16-23		256 byte
Port 4	$F000\ 1000_H$ - $F000\ 10FF_H$	see Page 16-24		256 byte

Table 16-2 Block Address Map of Segment 15 (cont'd)

Unit		Address Range	Access Mode		Size
			Read	Write	
Port 5		$F000\ 1100_H$ - $F000\ 11FF_H$	see Page 16-25		256 byte
Reserved		$F000\ 1200_H$ - $F000\ 17FF_H$	BE	BE	–
General Purpose Timer Array 0 (GPTA0)		$F000\ 1800_H$ - $F000\ 1FFF_H$	see Page 16-26		8 × 256 byte
Reserved		$F000\ 2000_H$ - $F000\ 3BFF_H$	BE	BE	–
Direct Memory Access Controller (DMA)		$F000\ 3C00_H$ - $F000\ 3EFF_H$	see Page 16-34		3 × 256 byte
Reserved		$F000\ 3F00_H$ - $F000\ 3FFF_H$	BE	BE	–
MultiCAN Controller (CAN)		$F000\ 4000_H$ - $F000\ 5FFF_H$	see Page 16-41		8 Kbyte
Reserved		$F000\ 6000_H$ - $F003\ FFFF_H$	BE	BE	–
PCP	Reserved	$F004\ 0000_H$ - $F004\ 3EFF_H$	BE	BE	–
	PCP Registers	$F004\ 3F00_H$ - $F004\ 3FFF_H$	see Page 16-48		256 byte
	Reserved	$F004\ 4000_H$ - $F004\ FFFF_H$	BE	BE	–
	PCP Data Memory (PRAM)	$F005\ 0000_H$ - $F005\ 1FFF_H$	nE, 32	nE, 32	8 Kbyte
	Reserved	$F005\ 2000_H$ - $F005\ FFFF_H$	BE	BE	–
	PCP Code Memory (CMEM)	$F006\ 0000_H$ - $F006\ 2FFF_H$	nE, 32	nE, 32	12 Kbyte
	Reserved	$F006\ 3000_H$ - $F007\ FFFF_H$	BE	BE	–
Reserved		$F008\ 0000_H$ - $F00F\ FFFF_H$	BE	BE	–

Table 16-2 Block Address Map of Segment 15 (cont'd)

Unit	Address Range	Access Mode		Size
		Read	Write	
Reserved	$F010\ 0000_H$ - $F010\ 00FF_H$	BE	BE	–
Synchronous Serial Interface 0 (SSC0)	$F010\ 0100_H$ - $F010\ 01FF_H$	see Page 16-50		256 byte
Synchronous Serial Interface 1 (SSC1)	$F010\ 0200_H$ - $F010\ 02FF_H$	see Page 16-51		256 byte
Fast Analog-to-Digital Converter (FADC)	$F010\ 0300_H$ - $F010\ 03FF_H$	see Page 16-52		256 byte
Analog-to-Digital Converter 0 (ADC0)	$F010\ 0400_H$ - $F010\ 05FF_H$	see Page 16-55		2 × 256 byte
Reserved	$F010\ 0600_H$ - $F010\ 07FF_H$	BE	BE	–
Reserved	$F010\ 0800_H$ - $F010\ 9FFF_H$	BE	BE	–
Reserved	$F010\ A000_H$ - $F010\ BFFF_H$	BE	BE	–
Micro Link Interface 0 (MLI0)	$F010\ C000_H$ - $F010\ C0FF_H$	see Page 16-60		256 byte
Micro Link Interface 1 (MLI1)	$F010\ C100_H$ - $F010\ C1FF_H$	see Page 16-63		256 byte
Memory Checker (MCHK)	$F010\ C200_H$ - $F010\ C2FF_H$	see Page 16-67		256 byte
Reserved	$F010\ C300_H$ - $F01D\ FFFF_H$	BE	BE	–
MLI0 Small Transfer Windows	$F01E\ 0000_H$ - $F01E\ 7FFF_H$	nE	nE	4 × 8 Kbyte
MLI1 Small Transfer Windows	$F01E\ 8000_H$ - $F01E\ FFFF_H$	nE	nE	4 × 8 Kbyte
Reserved	$F01F\ 0000_H$ - $F01F\ FFFF_H$	BE	BE	–
MLI0 Large Transfer Windows	$F020\ 0000_H$ - $F023\ FFFF_H$	nE	nE	4 × 64 Kbyte

Table 16-2 Block Address Map of Segment 15 (cont'd)

Unit		Address Range	Access Mode		Size
			Read	Write	
MLI1 Large Transfer Windows		$F024\ 0000_H$ - $F027\ FFFF_H$	nE	nE	4 × 64 Kbyte
Reserved		$F028\ 0000_H$ - $F7E0\ FEFF_H$	BE	BE	–
CPU	CPU Slave Interface Registers (CPS)	$F7E0\ FF00_H$ - $F7E0\ FFFF_H$	see Page 16-68		256 byte
	CPU Core SFRs & GPRs	$F7E1\ 0000_H$ - $F7E1\ FFFF_H$	see Page 16-69		64 Kbyte
Reserved		$F7E2\ 0000_H$ - $F7FF\ FFFF_H$	BE	BE	–
Reserved		$F800\ 0000_H$ - $F800\ 03FF_H$	BE	BE	–
Reserved		$F800\ 0400_H$ - $F800\ 04FF_H$	BE	BE	–
Program Memory Unit (PMU)		$F800\ 0500_H$ - $F800\ 05FF_H$	see Page 16-76		256 byte
Reserved		$F800\ 0600_H$ - $F800\ 0FFF_H$	BE	BE	–
Flash Register		$F800\ 1000_H$ - $F800\ 23FF_H$	see Page 16-80		5 Kbyte
Reserved		$F800\ 2400_H$ - $F801\ 00FF_H$	BE	BE	–
Reserved		$F801\ 0100_H$ - $F801\ 01FF_H$	BE	BE	–
Reserved		$F801\ 0200_H$ - $F87F\ F9FF_H$	BE	BE	–
Reserved		$F87F\ FA00_H$ - $F87F\ FAFF_H$	BE	BE	–
Reserved		$F87F\ FB00_H$ - $F87F\ FBFF_H$	BE	BE	–

Table 16-2 Block Address Map of Segment 15 (cont'd)

Unit		Address Range	Access Mode		Size
			Read	Write	
CPU	DMI Registers	F87F FC00$_H$ - F87F FCFF$_H$	see Page 16-81		256 byte
	PMI Registers	F87F FD00$_H$ - F87F FDFF$_H$	see Page 16-82		256 byte
Local Memory Bus Control Unit (LBCU)		F87F FE00$_H$ - F87F FEFF$_H$	see Page 16-83		256 byte
LMB to SPB Bus Bridge (LFI)		F87F FF00$_H$ - F87F FFFF$_H$	see Page 16-84		256 byte
Reserved		F880 0000$_H$ - FFFF FFFF$_H$	BE	BE	–

16.2 Registers Tables

Table 16-3 to Table 16-31 shows the address maps with all registers of Segment 15.

Note: Addresses listed in column "Address" are word (32-bit) addresses.

Table 16-3 Address Map of SCU and WDT

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
System Control Unit (SCU) with Watchdog Timer (WDT)					
–	Reserved	$F000\ 0000_H$ - $F000\ 0004_H$	BE	BE	–
SCU_ ID	SCU Module Identification Register	$F000\ 0008_H$	U, SV	BE	$002C\ C0XX_H$
SCU_ SCLKFDR	SCU System Clock Fractional Divider Register	$F000\ 000C_H$	U, SV	SV, E	$0000\ 0000_H$
RST_ REQ	Reset Request Register	$F000\ 0010_H$	U, SV	U, SV, E	$0000\ 0000_H$
RST_ SR	Reset Status Register	$F000\ 0014_H$	U, SV	BE	depending on boot config.
OSC_ CON	Oscillator Control Register	$F000\ 0018_H$	U, SV	SV, E	$0000\ 000X_H$
–	Reserved	$F000\ 001C_H$	BE	BE	–
WDT_ CON0	Watchdog Timer Control Register 0	$F000\ 0020_H$	U, SV	U, SV, PW	$FFFC\ 0002_H$
WDT_ CON1	Watchdog Timer Control Register 1	$F000\ 0024_H$	U, SV	U, SV, E	$0000\ 0000_H$
WDT_ SR	Watchdog Timer Status Register	$F000\ 0028_H$	U, SV	U, SV, NC	$FFFC\ 0010_H$
NMISR	NMI Status Register	$F000\ 002C_H$	U, SV	U, SV	$0000\ 0000_H$
–	Reserved	$F000\ 0030_H$	BE	BE	–
PMG_ CSR	Power Management Control and Status Register	$F000\ 0034_H$	U, SV	U, SV	$0000\ 0100_H$
SCU_ SCLIR	SCU Software Configuration Latched Inputs Register	$F000\ 0038_H$	U, SV	BE	$0000\ XXXX_H$
–	Reserved	$F000\ 003C_H$	BE	BE	–

Table 16-3 Address Map of SCU and WDT (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
PLL_ CLC	PLL Clock Control Register	F000 0040$_H$	U, SV	U, SV, E	see Table 3-6
SCU_ EMSR	SCU Emergency Stop Register	F000 0044$_H$	U, SV	U, SV, E	0000 0000$_H$
SCU_ TCCON	SCU Temperature Compensation Control Register	F000 0048$_H$	U, SV	U, SV, E	0000 0003$_H$
–	Reserved	F000 004C$_H$	BE	BE	–
SCU_ CON	SCU Control Register	F000 0050$_H$	U, SV	U, SV, E	FF00 0000$_H$
SCU_ STAT	SCU Status Register	F000 0054$_H$	U, SV	BE	0000 D000$_H$
SCU_ TCLR0	SCU Temperature Compensation Level Register 0	F000 0058$_H$	U, SV	U, SV, E	00FF FFFF$_H$
–	Reserved	F000 005C$_H$	nBE	nBE	–
–	Reserved; these locations must not be read and written	F000 0060$_H$ - F000 0068$_H$	–	–	–
–	Reserved	F000 006C$_H$	BE	BE	–
MANID	Manufacturer Identification Register	F000 0070$_H$	U, SV	BE	0000 1820$_H$
CHIPID	Chip Identification Register	F000 0074$_H$	U, SV	BE	0000 8BXX$_H$
RTID	Redesign Tracing Identification Register	F000 0078$_H$	U, SV	BE	0000 XXXX$_H$
EICR0	External Input Channel Register 0	F000 0080$_H$	U, SV	U, SV	0000 0000$_H$
EICR1	External Input Channel Register 1	F000 0084$_H$	U, SV	U, SV	0000 0000$_H$
EIFR	External Input Flag Register	F000 0088$_H$	U, SV	U, SV	0000 0000$_H$

Table 16-3 Address Map of SCU and WDT (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
FMR	Flag Modification Register	F000 008C$_H$	U, SV	U, SV	0000 0000$_H$
PDRR	Pattern Detection Result Register	F000 0090$_H$	U, SV	U, SV	0000 000F$_H$
IGCR0	Interrupt Gating Register 0	F000 0094$_H$	U, SV	U, SV	0000 0000$_H$
IGCR1	Interrupt Gating Register 1	F000 0098$_H$	U, SV	U, SV	0000 0000$_H$
TGADC0	Trigger Gating ADC0 Register	F000 009C$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 00A0$_H$	nBE	nBE	–
–	Reserved	F000 00A4$_H$	BE	BE	–
SCU_ PTCON	SCU Pad Test Control Register	F000 00B0$_H$	U, SV	SV, E	0000 0000$_H$
SCU_ PTDAT0	SCU Pad Test Data Register 0	F000 00B4$_H$	U, SV	U, SV	XXXX XXXX$_H$
–	Reserved	F000 00B8$_H$ - F000 00CC$_H$	BE	BE	–
SCU_ PETCR	SCU Parity Error Trap Control Register	F000 00D0$_H$	U, SV	U, SV, E	0000 0000$_H$
SCU_ PETSR	SCU Parity Error Trap Status Register	F000 00D4$_H$	U, SV	BE	0000 0000$_H$
SCU_ DMARS	DMA Request Select Register	F000 00D8$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 00E0$_H$ - F000 00F4$_H$	BE	BE	–

1) Which Resets affect the register, see **Table 4-2**.

Table 16-4 Address Map of SBCU

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
System Peripheral Bus Control Unit (SBCU)					
–	Reserved	F000 0100$_H$ - F000 0104$_H$	BE	BE	–
SBCU_ID	SBCU Module Identification Register	F000 0108$_H$	U, SV	BE	0000 6AXX$_H$
–	Reserved	F000 010C$_H$	BE	BE	–
SBCU_ CON	SBCU Control Register	F000 0110$_H$	U, SV	SV	4009 FFFF$_H$
–	Reserved	F000 0114$_H$ - F000 011C$_H$	BE	BE	–
SBCU_ ECON	SBCU Error Control Capture Register	F000 0120$_H$	U, SV	SV	0000 0000$_H$
SBCU_ EADD	SBCU Error Address Capture Register	F000 0124$_H$	U, SV	SV	0000 0000$_H$
SBCU_ EDAT	SBCU Error Data Capture Register	F000 0128$_H$	U, SV	SV	0000 0000$_H$
–	Reserved	F000 012C$_H$	BE	BE	–
SBCU_ DBCNTL	SBCU Debug Control Register	F000 0130$_H$	U, SV	SV	0000 7003$_H$
SBCU_ DBGRNT	SBCU Debug Grant Mask Register	F000 0134$_H$	U, SV	SV	0000 FFFF$_H$
SBCU_ DBADR1	SBCU Debug Address Register 1	F000 0138$_H$	U, SV	SV	0000 0000$_H$
SBCU_ DBADR2	SBCU Debug Address Register 2	F000 013C$_H$	U, SV	SV	0000 0000$_H$
SBCU_ DBBOS	SBCU Debug Bus Operation Signals Register	F000 0140$_H$	U, SV	SV	0000 0000$_H$
SBCU_ DBGNTT	SBCU Debug Trapped Master Register	F000 0144$_H$	U, SV	BE	FFFF FFFF$_H$
SBCU_ DBADRT	SBCU Debug Trapped Address Register	F000 0148$_H$	U, SV	BE	0000 0000$_H$

Table 16-4 Address Map of SBCU (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
SBCU_ DBBOST	SBCU Debug Trapped Bus Operation Signals Register	F000 014C$_H$	U, SV	BE	0000 3180$_H$
–	Reserved	F000 0150$_H$ - F000 01F8$_H$	BE	BE	–
SBCU_ SRC	SBCU Service Request Control Register	F000 01FC$_H$	U, SV	SV	0000 0000$_H$

1) Which Resets affect the register, see **Table 4-2**.

Table 16-5 Address Map of STM

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
System Timer (STM)					
STM_CLC	STM Clock Control Reg.	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$_H$

1) Which Resets affect the register, see **Table 4-2**.

Table 16-6 Address Map of Cerberus

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
On-Chip Debug Support (Cerberus)					
–	Reserved	F000 0400$_H$ - F000 0404$_H$	BE	BE	–
CBS_ JDPID	Cerberus Module Identification Register	F000 0408$_H$	U, SV	BE	0000 63XX$_H$
–	Reserved	F000 040C$_H$ - F000 0464$_H$	BE	BE	–
CBS_ COMDATA	Cerberus Communication Mode Data Register	F000 0468$_H$	U, SV	SV	0000 0000$_H$
CBS_ IOSR	Cerberus Status Register	F000 046C$_H$	U, SV	SV	0000 0000$_H$
CBS_ MCDBBS	Cerberus Break Bus Switch Configuration Register	F000 0470$_H$	U, SV	SV	0000 0000$_H$
CBS_ MCDSSG	Cerberus Suspend Signal Generation Status and Control Register	F000 0474$_H$	U, SV	SV	0000 0000$_H$
CBS_ OEC	Cerberus OCDS Enable Control Register	F000 0478$_H$	U, SV	SV	0000 0000$_H$
CBS_ OCNTRL	Cerberus OSCU Configuration and Control Register	F000 047C$_H$	U, SV	SV	0000 0000$_H$
CBS_ OSTATE	Cerberus OSCU Status Register	F000 0480$_H$	U, SV	SV	0001 0000$_H$
CBS_ INTMOD	Cerberus Internal Mode Status and Control Register	F000 0484$_H$	U, SV	SV	0000 0000$_H$
CBS_ ICTSA	Cerberus Internal Controlled Trace Source Address Register	F000 0488$_H$	U, SV	SV	0000 0000$_H$
CBS_ ICTTA	Cerberus Internal Controlled Trace Target Address Register	F000 048C$_H$	U, SV	SV	0000 0000$_H$

Table 16-6 Address Map of Cerberus (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
CBS_ MCDBBSS	Cerberus Break Bus Switch Status Register	F000 0490$_H$	U, SV	SV	0000 0000$_H$
CBS_ MCDSSGC	Cerberus Suspend Signal Generation Configuration Register	F000 0494$_H$	U, SV	SV	0000 0000$_H$
–	Reserved	F000 0498$_H$ - F000 04F8$_H$	BE	BE	–
CBS_ SRC	Cerberus Service Request Control Register	F000 04FC$_H$	U, SV	SV	0000 0000$_H$

1) Which Resets affect the register, see **Table 4-2**.

Table 16-7 Address Map of MSC0

Short Name	Description	Address	Access Mode		Reset Value[1)]
			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$_H$
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$_H$

Table 16-7 Address Map of MSC0 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
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	–
–	Reserved	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$_H$

1) Which Resets affect the register, see **Table 4-2**.

Table 16-8 Address Map of ASC0/ASC1

Short Name	Description	Address	Access Mode		Reset Value[1)]
			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$_H$
ASC0_ ESRC	ASC0 Error Interrupt Service Req. Control Reg.	F000 0AF8$_H$	U, SV	U, SV	0000 0000$_H$

Table 16-8 Address Map of ASC0/ASC1 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
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 16-8 Address Map of ASC0/ASC1 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			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$_H$

1) Which Resets affect the register, see **Table 4-2**.

Table 16-9 Address Map of Port 0

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
Port 0					
P0_ OUT	Port 0 Output Register	F000 0C00$_H$	U, SV	U, SV	0000 0000$_H$
P0_ OMR	Port 0 Output Modification Register	F000 0C04$_H$	U, SV	U, SV,32	0000 XXXX$_H$
–	Reserved	F000 0C08$_H$ - F000 0C0C$_H$	U, SV	U, SV	–
P0_ IOCR0	Port 0 Input/Output Control Register 0	F000 0C10$_H$	U, SV	U, SV	2020 2020$_H$
P0_ IOCR4	Port 0 Input/Output Control Register 4	F000 0C14$_H$	U, SV	U, SV	2020 2020$_H$
P0_ IOCR8	Port 0 Input/Output Control Register 8	F000 0C18$_H$	U, SV	U, SV	2020 2020$_H$
P0_ IOCR12	Port 0 Input/Output Control Register 12	F000 0C1C$_H$	U, SV	U, SV	2020 2020$_H$
–	Reserved	F000 0C20$_H$	U, SV	U, SV	–
P0_ IN	Port 0 Input Register	F000 0C24$_H$	U, SV	U, SV	0000 XXXX$_H$
–	Reserved	F000 0C28$_H$ - F000 0C3C$_H$	U, SV	U, SV	–
P0_ PDR	Port 0 Pad Driver Mode Register	F000 0C40$_H$	U, SV	SV, E	0000 0000$_H$
–	Reserved	F000 0C44$_H$ - F000 0C4C$_H$	U, SV	U, SV	–
P0_ ESR	Port 0 Emergency Stop Register	F000 0C50$_H$	U, SV	SV, E	0000 0000$_H$
–	Reserved	F000 0C54$_H$ - F000 0CFC$_H$	U, SV	U, SV	–

1) Which Resets affect the register, see **Table 4-2**.

Table 16-10 Address Map of Port 1

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
Port 1					
P1_OUT	Port 1 Output Register	F000 0D00$_H$	U, SV	U, SV	0000 0000$_H$
P1_OMR	Port 1 Output Modification Register	F000 0D04$_H$	U, SV	U,SV, 32	0000 XXXX$_H$
–	Reserved	F000 0D08$_H$ - F000 0D0C$_H$	U, SV	U, SV	–
P1_IOCR0	Port 1 Input/Output Control Register 0	F000 0D10$_H$	U, SV	U, SV	0020 2020$_H$
P1_IOCR4	Port 1 Input/Output Control Register 4	F000 0D14$_H$	U, SV	U, SV	2020 2020$_H$
P1_IOCR8	Port 1 Input/Output Control Register 8	F000 0D18$_H$	U, SV	U, SV	2020 2020$_H$
P1_IOCR12	Port 1 Input/Output Control Register 12	F000 0D1C$_H$	U, SV	U, SV	0020 2020$_H$
–	Reserved	F000 0D20$_H$	U, SV	U, SV	–
P1_IN	Port 1 Input Register	F000 0D24$_H$	U, SV	U, SV	0000 XXXX$_H$
–	Reserved	F000 0D28$_H$ - F000 0D3C$_H$	U, SV	U, SV	–
P1_PDR	Port 1 Pad Driver Mode Register	F000 0D40$_H$	U, SV	SV, E	0000 0000$_H$
–	Reserved	F000 0D44$_H$ - F000 0D4C$_H$	U, SV	U, SV	–
P1_ESR	Port 1 Emergency Stop Register	F000 0D50$_H$	U, SV	SV, E	0000 0000$_H$
–	Reserved	F000 0D54$_H$ - F000 0DFC$_H$	U, SV	U, SV	–

1) Which Resets affect the register, see **Table 4-2**.

Table 16-11 Address Map of Port 2

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
Port 2					
P2_ OUT	Port 2 Output Register	F000 0E00$_H$	U, SV	U, SV	0000 0000$_H$
P2_ OMR	Port 2 Output Modification Register	F000 0E04$_H$	U, SV	U,SV, 32	0000 XXXX$_H$
–	Reserved	F000 0E08$_H$ - F000 0E0C$_H$	U, SV	U, SV	–
P2_ IOCR0	Port 2 Input/Output Control Register 0	F000 0E10$_H$	U, SV	U, SV	2020 2020$_H$
P2_ IOCR4	Port 2 Input/Output Control Register 4	F000 0E14$_H$	U, SV	U, SV	2020 2020$_H$
P2_ IOCR8	Port 2 Input/Output Control Register 8	F000 0E18$_H$	U, SV	U, SV	2020 2020$_H$
P2_ IOCR12	Port 2 Input/Output Control Register 12	F000 0E1C$_H$	U, SV	U, SV	0000 2020$_H$
–	Reserved	F000 0E20$_H$	U, SV	U, SV	–
P2_ IN	Port 2 Input Register	F000 0E24$_H$	U, SV	U, SV	0000 XXXX$_H$
–	Reserved	F000 0E28$_H$ - F000 0E3C$_H$	U, SV	U, SV	–
P2_ PDR	Port 2 Pad Driver Mode Register	F000 0E40$_H$	U, SV	SV, E	0000 0000$_H$
–	Reserved	F000 0E44$_H$ - F000 0E4C$_H$	U, SV	U, SV	–
P2_ ESR	Port 2 Emergency Stop Register	F000 0E50$_H$	U, SV	SV, E	0000 0000$_H$
–	Reserved	F000 0E54$_H$ - F000 0EFC$_H$	U, SV	U, SV	–

1) Which Resets affect the register, see **Table 4-2**.

Table 16-12 Address Map of Port 3

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
Port 3					
P3_ OUT	Port 3 Output Register	F000 0F00$_H$	U, SV	U, SV	0000 0000$_H$
P3_ OMR	Port 3 Output Modification Register	F000 0F04$_H$	U, SV	U,SV, 32	0000 XXXX$_H$
–	Reserved	F000 0F08$_H$ - F000 0F0C$_H$	U, SV	U, SV	–
P3_ IOCR0	Port 3 Input/Output Control Register 0	F000 0F10$_H$	U, SV	U, SV	2020 2020$_H$
P3_ IOCR4	Port 3 Input/Output Control Register 4	F000 0F14$_H$	U, SV	U, SV	2020 2020$_H$
P3_ IOCR8	Port 3 Input/Output Control Register 8	F000 0F18$_H$	U, SV	U, SV	2020 2020$_H$
P3_ IOCR12	Port 3 Input/Output Control Register 12	F000 0F1C$_H$	U, SV	U, SV	2020 2020$_H$
–	Reserved	F000 0F20$_H$	U, SV	U, SV	–
P3_ IN	Port 3 Input Register	F000 0F24$_H$	U, SV	U, SV	0000 XXXX$_H$
–	Reserved	F000 0F28$_H$ - F000 0F3C$_H$	U, SV	U, SV	–
P3_ PDR	Port 3 Pad Driver Mode Register	F000 0F40$_H$	U, SV	SV, E	0000 0000$_H$
–	Reserved	F000 0F44$_H$ - F000 0FFC$_H$	U, SV	U, SV	–

1) Which Resets affect the register, see **Table 4-2**.

Table 16-13 Address Map of Port 4

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
Port 4					
P4_OUT	Port 4 Output Register	F000 1000$_H$	U, SV	U, SV	0000 0000$_H$
P4_OMR	Port 4 Output Modification Register	F000 1004$_H$	U, SV	U, SV,32	0000 000X$_H$
–	Reserved	F000 1008$_H$ - F000 100C$_H$	U, SV	U, SV	–
P4_IOCR0	Port 4 Input/Output Control Register 0	F000 1010$_H$	U, SV	U, SV	2020 2020$_H$
–	Reserved	F000 1014$_H$ - F000 1020$_H$	U, SV	U, SV	–
P4_IN	Port 4 Input Register	F000 1024$_H$	U, SV	U, SV	0000 000X$_H$
–	Reserved	F000 1028$_H$ - F000 103C$_H$	U, SV	U, SV	–
P4_PDR	Port 4 Pad Driver Mode Register	F000 1040$_H$	U, SV	SV, E	0000 0000$_H$
–	Reserved	F000 1044$_H$ - F000 104C$_H$	U, SV	U, SV	–
P4_ESR	Port 4 Emergency Stop Register	F000 1050$_H$	U, SV	SV, E	0000 0000$_H$
–	Reserved	F000 1054$_H$ - F000 10FC$_H$	U, SV	U, SV	–

1) Which Resets affect the register, see **Table 4-2**.

Table 16-14 Address Map of Port 5

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
Port 5					
P5_ OUT	Port 5 Output Register	F000 1100$_H$	U, SV	U, SV	0000 0000$_H$
P5_ OMR	Port 5 Output Modification Register	F000 1104$_H$	U, SV	U,SV, 32	0000 XXXX$_H$
–	Reserved	F000 1108$_H$ - F000 110C$_H$	U, SV	U, SV	–
P5_ IOCR0	Port 5 Input/Output Control Register 0	F000 1110$_H$	U, SV	U, SV	2020 2020$_H$
P5_ IOCR4	Port 5 Input/Output Control Register 4	F000 1114$_H$	U, SV	U, SV	2020 2020$_H$
P5_ IOCR8	Port 5 Input/Output Control Register 8	F000 1118$_H$	U, SV	U, SV	2020 2020$_H$
P5_ IOCR12	Port 5 Input/Output Control Register 12	F000 111C$_H$	U, SV	U, SV	2020 2020$_H$
–	Reserved	F000 1120$_H$	U, SV	U, SV	–
P5_ IN	Port 5 Input Register	F000 1124$_H$	U, SV	U, SV	0000 XXXX$_H$
–	Reserved	F000 1128$_H$ - F000 113C$_H$	U, SV	U, SV	–
P5_ PDR	Port 5 Pad Driver Mode Register	F000 1140$_H$	U, SV	SV, E	0000 0000$_H$
–	Reserved	F000 1144$_H$ - F000 114C$_H$	U, SV	U, SV	–
P5_ ESR	Port 5 Emergency Stop Register	F000 1150$_H$	U, SV	SV, E	0000 0000$_H$
–	Reserved	F000 1154$_H$ - F000 11FC$_H$	U, SV	U, SV	–

1) Which Resets affect the register, see **Table 4-2**.

Table 16-15 Address Map of GPTA0

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
General Purpose Timer Array 0 (GPTA0)					
GPTA0_ CLC	GPTA Clock Control Register	F000 1800$_H$	U, SV	SV, E	0000 0003$_H$
GPTA0_ DBGCTR	GPTA Debug Clock Control Register	F000 1804$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ ID	GPTA0 Module Identification Register	F000 1808$_H$	U, SV	BE	0029 C0XX$_H$
GPTA0_ FDR	GPTA Fractional Divider Register	F000 180C$_H$	U, SV	SV, E	0000 0000$_H$
GPTA0_ SRSC0	GPTA0 Service Request State Clear Register 0	F000 1810$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRSS0	GPTA0 Service Request State Set Register 0	F000 1814$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRSC1	GPTA0 Service Request State Clear Register 1	F000 1818$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRSS1	GPTA0 Service Request State Set Register 1	F000 181C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRSC2	GPTA0 Service Request State Clear Register 2	F000 1820$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRSS2	GPTA0 Service Request State Set Register 2	F000 1824$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRSC3	GPTA0 Service Request State Clear Register 3	F000 1828$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRSS3	GPTA0 Service Request State Set Register 3	F000 182C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRNR	GPTA0 Service Request Node Redirection Register	F000 1830$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 1834$_H$	nBE	nBE	0000 0000$_H$
GPTA0_ MRACTL	GPTA0 Multiplexer Register Array Control Register	F000 1838$_H$	U, SV	U, SV	0000 0000$_H$

Table 16-15 Address Map of GPTA0 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
GPTA0_ MRADIN	GPTA0 Multiplexer Register Array Data In Register	F000 183C$_H$	U, SV, 32	U, SV, 32	0000 0000$_H$
GPTA0_ MRADOUT	GPTA0 Multiplexer Register Array Data Out Register	F000 1840$_H$	U, SV, 32	U, SV, 32	0000 0000$_H$
GPTA0_ FPCSTAT	GPTA0 Filter and Prescaler Cell Status Register	F000 1844$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ FPCCTR0	GPTA0 Filter and Prescaler Cell Control Register 0	F000 1848$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ FPCTIM0	GPTA0 Filter and Prescaler Cell Timer Register 0	F000 184C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ FPCCTR1	GPTA0 Filter and Prescaler Cell Control Register 1	F000 1850$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ FPCTIM1	GPTA0 Filter and Prescaler Cell Timer Register 1	F000 1854$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ FPCCTR2	GPTA0 Filter and Prescaler Cell Control Register 2	F000 1858$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ FPCTIM2	GPTA0 Filter and Prescaler Cell Timer Register 2	F000 185C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ FPCCTR3	GPTA0 Filter and Prescaler Cell Control Register 3	F000 1860$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ FPCTIM3	GPTA0 Filter and Prescaler Cell Timer Register 3	F000 1864$_H$	U, SV	U, SV	0000 0000$_H$

Table 16-15 Address Map of GPTA0 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
GPTA0_ FPCCTR4	GPTA0 Filter and Prescaler Cell Control Register 4	F000 1868$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ FPCTIM4	GPTA0 Filter and Prescaler Cell Timer Register 4	F000 186C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ FPCCTR5	GPTA0 Filter and Prescaler Cell Control Register 5	F000 1870$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ FPCTIM5	GPTA0 Filter and Prescaler Cell Timer Register 5	F000 1874$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ PDLCTR	GPTA0 Phase Discrimination Logic Control Register	F000 1878$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 187C$_H$	nBE	nBE	–
GPTA0_ DCMCTR0	GPTA0 Duty Cycle Measurement Control Register 0	F000 1880$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ DCMTIM0	GPTA0 Duty Cycle Measurement Timer Register 0	F000 1884$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ DCMCAV0	GPTA0 Duty Cycle Measurement Capture Register 0	F000 1888$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ DCMCOV0	GPTA0 Duty Cycle Measurement Capture/Compare Register 0	F000 188C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ DCMCTR1	GPTA0 Duty Cycle Measurement Control Register 1	F000 1890$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ DCMTIM1	GPTA0 Duty Cycle Measurement Timer Register 1	F000 1894$_H$	U, SV	U, SV	0000 0000$_H$

Table 16-15 Address Map of GPTA0 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
GPTA0_ DCMCAV1	GPTA0 Duty Cycle Measurement Capture Register 1	F000 1898$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ DCMCOV1	GPTA0 Duty Cycle Measurement Capture/Compare Register 1	F000 189C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ DCMCTR2	GPTA0 Duty Cycle Measurement Control Register 2	F000 18A0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ DCMTIM2	GPTA0 Duty Cycle Measurement Timer Register 2	F000 18A4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ DCMCAV2	GPTA0 Duty Cycle Measurement Capture Register 2	F000 18A8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ DCMCOV2	GPTA0 Duty Cycle Measurement Capture/Compare Register 2	F000 18AC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ DCMCTR3	GPTA0 Duty Cycle Measurement Control Register 3	F000 18B0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ DCMTIM3	GPTA0 Duty Cycle Measurement Timer Register 3	F000 18B4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ DCMCAV3	GPTA0 Duty Cycle Measurement Capture Register 3	F000 18B8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ DCMCOV3	GPTA0 Duty Cycle Measurement Capture/Compare Register 3	F000 18BC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ PLLCTR	GPTA0 Phase Locked Loop Control Register	F000 18C0$_H$	U, SV	U, SV	0000 0000$_H$

Table 16-15 Address Map of GPTA0 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
GPTA0_ PLLMTI	GPTA0 Phase Locked Loop Microtick Register	F000 18C4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ PLLCNT	GPTA0 Phase Locked Loop Counter Register	F000 18C8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ PLLSTP	GPTA0 Phase Locked Loop Step Register	F000 18CC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ PLLREV	GPTA0 Phase Locked Loop Reload Register	F000 18D0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ PLLDTR	GPTA0 Phase Locked Loop Delta Register	F000 18D4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ CKBCTR	GPTA0 Clock Bus Control Register	F000 18D8$_H$	U, SV	U, SV	0000 FFFF$_H$
–	Reserved	F000 18DC$_H$	nBE	nBE	–
GPTA0_ GTCTR0	GPTA0 Global Timer Control Register 0	F000 18E0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ GTREV0	GPTA0 Global Timer Reload Value Register 0	F000 18E4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ GTTIM0	GPTA0 Global Timer Register 0	F000 18E8$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 18EC$_H$	nBE	nBE	–
GPTA0_ GTCTR1	GPTA0 Global Timer Control Register 1	F000 18F0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ GTREV1	GPTA0 Global Timer Reload Value Register 1	F000 18F4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ GTTIM1	GPTA0 Global Timer Register 1	F000 18F8$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 18FC$_H$	nBE	nBE	–
GPTA0_ GTCCTRn	GPTA0 Global Timer Cell Control Register n (n = 00-31)	F000 1900$_H$ + n × 08$_H$ + 00$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ GTCXRn	GPTA0 Global Timer Cell X Register n (n = 00-31)	F000 1900$_H$ + n × 08$_H$ + 04$_H$	U, SV	U, SV	0000 0000$_H$

Table 16-15 Address Map of GPTA0 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
GPTA0_ LTCCTRn	GPTA0 Local Timer Cell Control Register n (n = 00-63)	F000 1A00$_H$ + n × 08$_H$ + 00$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ LTCXRn	GPTA0 Local Timer Cell X Register n (n = 00-63)	F000 1A00$_H$ + n × 08$_H$ + 04$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ EDCTR	GPTA Clock Enable/Disable Control Register	F000 1C00$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 1C04$_H$ - F000 1EFC$_H$	nBE	nBE	–
–	Reserved	F000 1F00$_H$ - F000 1F0C$_H$	nBE	nBE	–
–	Reserved	F000 1F10$_H$ - F000 1F64$_H$	nBE	nBE	–
GPTA0_ SRC37	GPTA0 Service Request Control Register 37	F000 1F68$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC36	GPTA0 Service Request Control Register 36	F000 1F6C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC35	GPTA0 Service Request Control Register 35	F000 1F70$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC34	GPTA0 Service Request Control Register 34	F000 1F74$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC33	GPTA0 Service Request Control Register 33	F000 1F78$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC32	GPTA0 Service Request Control Register 32	F000 1F7C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC31	GPTA0 Service Request Control Register 31	F000 1F80$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC30	GPTA0 Service Request Control Register 30	F000 1F84$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC29	GPTA0 Service Request Control Register 29	F000 1F88$_H$	U, SV	U, SV	0000 0000$_H$

Table 16-15 Address Map of GPTA0 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
GPTA0_ SRC28	GPTA0 Service Request Control Register 28	F000 1F8C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC27	GPTA0 Service Request Control Register 27	F000 1F90$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC26	GPTA0 Service Request Control Register 26	F000 1F94$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC25	GPTA0 Service Request Control Register 25	F000 1F98$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC24	GPTA0 Service Request Control Register 24	F000 1F9C$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC23	GPTA0 Service Request Control Register 23	F000 1FA0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC22	GPTA0 Service Request Control Register 22	F000 1FA4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC21	GPTA0 Service Request Control Register 21	F000 1FA8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC20	GPTA0 Service Request Control Register 20	F000 1FAC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC19	GPTA0 Service Request Control Register 19	F000 1FB0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC18	GPTA0 Service Request Control Register 18	F000 1FB4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC17	GPTA0 Service Request Control Register 17	F000 1FB8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC16	GPTA0 Service Request Control Register 16	F000 1FBC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC15	GPTA0 Service Request Control Register 15	F000 1FC0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC14	GPTA0 Service Request Control Register 14	F000 1FC4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC13	GPTA0 Service Request Control Register 13	F000 1FC8$_H$	U, SV	U, SV	0000 0000$_H$

Table 16-15 Address Map of GPTA0 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
GPTA0_ SRC12	GPTA0 Service Request Control Register 12	F000 1FCC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC11	GPTA0 Service Request Control Register 11	F000 1FD0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC10	GPTA0 Service Request Control Register 10	F000 1FD4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC09	GPTA0 Service Request Control Register 09	F000 1FD8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC08	GPTA0 Service Request Control Register 08	F000 1FDC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC07	GPTA0 Service Request Control Register 07	F000 1FE0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC06	GPTA0 Service Request Control Register 06	F000 1FE4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC05	GPTA0 Service Request Control Register 05	F000 1FE8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC04	GPTA0 Service Request Control Register 04	F000 1FEC$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC03	GPTA0 Service Request Control Register 03	F000 1FF0$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC02	GPTA0 Service Request Control Register 02	F000 1FF4$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC01	GPTA0 Service Request Control Register 01	F000 1FF8$_H$	U, SV	U, SV	0000 0000$_H$
GPTA0_ SRC00	GPTA0 Service Request Control Register 00	F000 1FFC$_H$	U, SV	U, SV	0000 0000$_H$

1) Which Resets affect the register, see **Table 4-2**.

Table 16-16 Address Map of DMA

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
Direct Memory Access Controller (DMA)					
DMA_CLC	DMA Clock Control Register	F000 3C00$_H$	U, SV	SV, E	0000 0000$_H$
–	Reserved	F000 3C04$_H$	nBE	SV	–
DMA_ID	DMA Module Identification Register	F000 3C08$_H$	U, SV	BE	001A C0XX$_H$
–	Reserved	F000 3C0C$_H$	BE	BE	–
DMA_ CHRSTR	DMA Channel Reset Request Register	F000 3C10$_H$	U, SV	SV	0000 0000$_H$
DMA_ TRSR	DMA Transaction Request State Register	F000 3C14$_H$	U, SV	BE	0000 0000$_H$
DMA_ STREQ	DMA Software Transaction Request Register	F000 3C18$_H$	U, SV	SV	0000 0000$_H$
DMA_ HTREQ	DMA Hardware Transaction Request Register	F000 3C1C$_H$	U, SV	SV	0000 0000$_H$
DMA_ EER	DMA Enable Error Register	F000 3C20$_H$	U, SV	SV	0000 0000$_H$
DMA_ ERRSR	DMA Error Status Register	F000 3C24$_H$	U, SV	BE	0000 0000$_H$
DMA_ CLRE	DMA Clear Error Register	F000 3C28$_H$	U, SV	SV	0000 0000$_H$
DMA_ GINTR	DMA Global Interrupt Set Register	F000 3C2C$_H$	U, SV	SV	0000 0000$_H$
DMA_ MESR	DMA Move Engine Status Register	F000 3C30$_H$	U, SV	BE	0000 0000$_H$
DMA_ ME0R	DMA Move Engine 0 Read Register	F000 3C34$_H$	U, SV	BE	0000 0000$_H$
–	Reserved	F000 3C38$_H$	U, SV	BE	0000 0000$_H$
DMA_ ME0PR	DMA Move Engine 0 Pattern Register	F000 3C3C$_H$	U, SV	SV	0000 0000$_H$

Table 16-16 Address Map of DMA (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
–	Reserved	F000 3C40$_H$	U, SV	SV	0000 0000$_H$
DMA_ ME0AENR	DMA Move Engine 0 Access Enable Register	F000 3C44$_H$	U, SV	SV, E	0000 0000$_H$
DMA_ ME0ARR	DMA Move Engine 0 Access Range Register	F000 3C48$_H$	U, SV	SV, E	0000 0000$_H$
–	Reserved	F000 3C4C$_H$	U, SV	SV, E	0000 0000$_H$
–	Reserved	F000 3C50$_H$	U, SV	SV, E	0000 0000$_H$
DMA_ INTSR	DMA Interrupt Status Register	F000 3C54$_H$	U, SV	BE	0000 0000$_H$
DMA_ INTCR	DMA Interrupt Clear Register	F000 3C58$_H$	U, SV	SV	0000 0000$_H$
DMA_ WRPSR	DMA Wrap Status Register	F000 3C5C$_H$	U, SV	BE	0000 0000$_H$
–	Reserved	F000 3C60$_H$	BE	BE	–
DMA_ OCDSR	DMA OCDS Register	F000 3C64$_H$	U, SV	SV, E	0000 0000$_H$
DMA_ SUSPMR	DMA Suspend Mode Register	F000 3C68$_H$	U, SV	SV, E	0000 0000$_H$
–	Reserved	F000 3C6C$_H$ - F000 3C7C$_H$	BE	BE	–
DMA_ CHSR00	DMA Channel 00 Status Register	F000 3C80$_H$	U, SV	BE	0000 0000$_H$
DMA_ CHCR00	DMA Channel 00 Control Register	F000 3C84$_H$	U, SV	SV	0000 0000$_H$
DMA_ CHICR00	DMA Channel 00 Interrupt Control Register	F000 3C88$_H$	U, SV	SV	0000 0000$_H$
DMA_ ADRCR00	DMA Channel 00 Address Control Register	F000 3C8C$_H$	U, SV	SV	0000 0000$_H$
DMA_ SADR00	DMA Channel 00 Source Address Register	F000 3C90$_H$	U, SV	SV	0000 0000$_H$
DMA_ DADR00	DMA Channel 00 Destination Address Reg.	F000 3C94$_H$	U, SV	SV	0000 0000$_H$

Table 16-16 Address Map of DMA (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
DMA_ SHADR00	DMA Channel 00 Shadowed Address Reg.	F000 3C98$_H$	U, SV	BE	0000 0000$_H$
–	Reserved	F000 3C9C$_H$	BE	BE	–
DMA_ CHSR01	DMA Channel 01 Status Register	F000 3CA0$_H$	U, SV	BE	0000 0000$_H$
DMA_ CHCR01	DMA Channel 01 Control Register	F000 3CA4$_H$	U, SV	SV	0000 0000$_H$
DMA_ CHICR01	DMA Channel 01 Interrupt Control Register	F000 3CA8$_H$	U, SV	SV	0000 0000$_H$
DMA_ ADRCR01	DMA Channel 01Address Control Register	F000 3CAC$_H$	U, SV	SV	0000 0000$_H$
DMA_ SADR01	DMA Channel 01 Source Address Register	F000 3CB0$_H$	U, SV	SV	0000 0000$_H$
DMA_ DADR01	DMA Channel 01 Destination Address Reg.	F000 3CB4$_H$	U, SV	SV	0000 0000$_H$
DMA_ SHADR01	DMA Channel 01 Shadowed Address Reg.	F000 3CB8$_H$	U, SV	BE	0000 0000$_H$
–	Reserved	F000 3CBC$_H$	BE	BE	–
DMA_ CHSR02	DMA Channel 02 Status Register	F000 3CC0$_H$	U, SV	BE	0000 0000$_H$
DMA_ CHCR02	DMA Channel 02 Control Register	F000 3CC4$_H$	U, SV	SV	0000 0000$_H$
DMA_ CHICR02	DMA Channel 02 Interrupt Control Register	F000 3CC8$_H$	U, SV	SV	0000 0000$_H$
DMA_ ADRCR02	DMA Channel 02 Address Control Register	F000 3CCC$_H$	U, SV	SV	0000 0000$_H$
DMA_ SADR02	DMA Channel 02 Source Address Register	F000 3CD0$_H$	U, SV	SV	0000 0000$_H$
DMA_ DADR02	DMA Channel 02 Destination Address Reg.	F000 3CD4$_H$	U, SV	SV	0000 0000$_H$
DMA_ SHADR02	DMA Channel 02 Shadowed Address Reg.	F000 3CD8$_H$	U, SV	BE	0000 0000$_H$
–	Reserved	F000 3CDC$_H$	BE	BE	–

Table 16-16 Address Map of DMA (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
DMA_ CHSR03	DMA Channel 03 Status Register	F000 3CE0$_H$	U, SV	BE	0000 0000$_H$
DMA_ CHCR03	DMA Channel 03 Control Register	F000 3CE4$_H$	U, SV	SV	0000 0000$_H$
DMA_ CHICR03	DMA Channel 03 Interrupt Control Register	F000 3CE8$_H$	U, SV	SV	0000 0000$_H$
DMA_ ADRCR03	DMA Channel 03 Address Control Register	F000 3CEC$_H$	U, SV	SV	0000 0000$_H$
DMA_ SADR03	DMA Channel 03 Source Address Register	F000 3CF0$_H$	U, SV	SV	0000 0000$_H$
DMA_ DADR03	DMA Channel 03 Destination Address Reg.	F000 3CF4$_H$	U, SV	SV	0000 0000$_H$
DMA_ SHADR03	DMA Channel 03 Shadowed Address Reg.	F000 3CF8$_H$	U, SV	BE	0000 0000$_H$
–	Reserved	F000 3CFC$_H$	BE	BE	–
DMA_ CHSR04	DMA Channel 04 Status Register	F000 3D00$_H$	U, SV	BE	0000 0000$_H$
DMA_ CHCR04	DMA Channel 04 Control Register	F000 3D04$_H$	U, SV	SV	0000 0000$_H$
DMA_ CHICR04	DMA Channel 04 Interrupt Control Register	F000 3D08$_H$	U, SV	SV	0000 0000$_H$
DMA_ ADRCR04	DMA Channel 04 Address Control Register	F000 3D0C$_H$	U, SV	SV	0000 0000$_H$
DMA_ SADR04	DMA Channel 04 Source Address Register	F000 3D10$_H$	U, SV	SV	0000 0000$_H$
DMA_ DADR04	DMA Channel 04 Destination Address Reg.	F000 3D14$_H$	U, SV	SV	0000 0000$_H$
DMA_ SHADR04	DMA Channel 04 Shadowed Address Reg.	F000 3D18$_H$	U, SV	BE	0000 0000$_H$
–	Reserved	F000 3D1C$_H$	BE	BE	–
DMA_ CHSR05	DMA Channel 05 Status Register	F000 3D20$_H$	U, SV	BE	0000 0000$_H$

Table 16-16 Address Map of DMA (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
DMA_ CHCR05	DMA Channel 05 Control Register	F000 3D24$_H$	U, SV	SV	0000 0000$_H$
DMA_ CHICR05	DMA Channel 05 Interrupt Control Register	F000 3D28$_H$	U, SV	SV	0000 0000$_H$
DMA_ ADRCR05	DMA Channel 05 Address Control Register	F000 3D2C$_H$	U, SV	SV	0000 0000$_H$
DMA_ SADR05	DMA Channel 05 Source Address Register	F000 3D30$_H$	U, SV	SV	0000 0000$_H$
DMA_ DADR05	DMA Channel 05 Destination Address Reg.	F000 3D34$_H$	U, SV	SV	0000 0000$_H$
DMA_ SHADR05	DMA Channel 05 Shadowed Address Reg.	F000 3D38$_H$	U, SV	BE	0000 0000$_H$
–	Reserved	F000 3D3C$_H$	BE	BE	–
DMA_ CHSR06	DMA Channel 06 Status Register	F000 3D40$_H$	U, SV	BE	0000 0000$_H$
DMA_ CHCR06	DMA Channel 06 Control Register	F000 3D44$_H$	U, SV	SV	0000 0000$_H$
DMA_ CHICR06	DMA Channel 06 Interrupt Control Register	F000 3D48$_H$	U, SV	SV	0000 0000$_H$
DMA_ ADRCR06	DMA Channel 06 Address Control Register	F000 3D4C$_H$	U, SV	SV	0000 0000$_H$
DMA_ SADR06	DMA Channel 06 Source Address Register	F000 3D50$_H$	U, SV	SV	0000 0000$_H$
DMA_ DADR06	DMA Channel 06 Destination Address Reg.	F000 3D54$_H$	U, SV	SV	0000 0000$_H$
DMA_ SHADR06	DMA Channel 06 Shadowed Address Reg.	F000 3D58$_H$	U, SV	BE	0000 0000$_H$
–	Reserved	F000 3D5C$_H$	BE	BE	–
DMA_ CHSR07	DMA Channel 07 Status Register	F000 3D60$_H$	U, SV	BE	0000 0000$_H$
DMA_ CHCR07	DMA Channel 07 Control Register	F000 3D64$_H$	U, SV	SV	0000 0000$_H$

Table 16-16 Address Map of DMA (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
DMA_ CHICR07	DMA Channel 07 Interrupt Control Register	F000 3D68$_H$	U, SV	SV	0000 0000$_H$
DMA_ ADRCR07	DMA Channel 07 Address Control Register	F000 3D6C$_H$	U, SV	SV	0000 0000$_H$
DMA_ SADR07	DMA Channel 07 Source Address Register	F000 3D70$_H$	U, SV	SV	0000 0000$_H$
DMA_ DADR07	DMA Channel 07 Destination Address Reg.	F000 3D74$_H$	U, SV	SV	0000 0000$_H$
DMA_ SHADR07	DMA Channel 07 Shadowed Address Reg.	F000 3D78$_H$	U, SV	BE	0000 0000$_H$
–	Reserved	F000 3D7C$_H$	BE	BE	–
–	Reserved	F000 3D80$_H$ - F000 3E7C$_H$	BE	BE	–
DMA_ TOCTR	DMA Bus Time Out Control Register	F000 3E80$_H$	U, SV	SV	0000 0000$_H$
–	Reserved	F000 3E84$_H$ - F000 3E88$_H$	BE	BE	–
DMA_ SYSSRC4	DMA System Interrupt Service Request Control Register 4	F000 3E8C$_H$	U, SV	SV	0000 0000$_H$
DMA_ SYSSRC3	DMA System Interrupt Service Request Control Register 3	F000 3E90$_H$	U, SV	SV	0000 0000$_H$
DMA_ SYSSRC2	DMA System Interrupt Service Request Control Register 2	F000 3E94$_H$	U, SV	SV	0000 0000$_H$
DMA_ SYSSRC1	DMA System Interrupt Service Request Control Register 1	F000 3E98$_H$	U, SV	SV	0000 0000$_H$
DMA_ SYSSRC0	DMA System Interrupt Service Request Control Register 0	F000 3E9C$_H$	U, SV	SV	0000 0000$_H$

Table 16-16 Address Map of DMA (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
DMA_ MLI0SRC3	DMA MLI0 Service Request Control Register 3	F000 3EA0$_H$	U, SV	SV	0000 0000$_H$
DMA_ MLI0SRC2	DMA MLI0 Service Request Control Register 2	F000 3EA4$_H$	U, SV	SV	0000 0000$_H$
DMA_ MLI0SRC1	DMA MLI0 Service Request Control Register 1	F000 3EA8$_H$	U, SV	SV	0000 0000$_H$
DMA_ MLI0SRC0	DMA MLI0 Service Request Control Register 0	F000 3EAC$_H$	U, SV	SV	0000 0000$_H$
–	Reserved	F000 3EB0$_H$ - F000 3EB4$_H$	BE	BE	–
DMA_ MLI1SRC1	DMA MLI1 Service Request Control Register 1	F000 3EB8$_H$	U, SV	SV	0000 0000$_H$
DMA_ MLI1SRC0	DMA MLI1 Service Request Control Register 0	F000 3EBC$_H$	U, SV	SV	0000 0000$_H$
–	Reserved	F000 3EC0$_H$ - F000 3EDC$_H$	BE	BE	–
–	Reserved	F000 3EE0$_H$ - F000 3EEC$_H$	BE	BE	–
DMA_ SRC3	DMA Service Request Control Register 3	F000 3EF0$_H$	U, SV	SV	0000 0000$_H$
DMA_ SRC2	DMA Service Request Control Register 2	F000 3EF4$_H$	U, SV	SV	0000 0000$_H$
DMA_ SRC1	DMA Service Request Control Register 1	F000 3EF8$_H$	U, SV	SV	0000 0000$_H$
DMA_ SRC0	DMA Service Request Control Register 0	F000 3EFC$_H$	U, SV	SV	0000 0000$_H$

1) Which Resets affect the register, see **Table 4-2**.

Table 16-17 Address Map of CAN

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
MultiCAN Controller (CAN)					
CAN_ CLC	CAN Clock Control Register	F000 4000$_H$	U, SV	SV, E	0000 0003$_H$
–	Reserved	F000 4004$_H$	nBE	nBE	–
CAN_ID	CAN Module Identification Register	F000 4008$_H$	U, SV	nBE	002B C0XX$_H$
CAN_ FDR	CAN Fractional Divider Register	F000 400C$_H$	U, SV	SV, E	0000 0000$_H$
–	Reserved	F000 4010$_H$ - F000 40E4$_H$	nBE	nBE	–
CAN_ SRC5	CAN Service Request Control Register 5	F000 40E8$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ SRC4	CAN Service Request Control Register 4	F000 40EC$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ SRC3	CAN Service Request Control Register 3	F000 40F0$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ SRC2	CAN Service Request Control Register 2	F000 40F4$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ SRC1	CAN Service Request Control Register 1	F000 40F8$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ SRC0	CAN Service Request Control Register 0	F000 40FC$_H$	U, SV	U, SV	0000 0000$_H$
Global Module Control Registers					
CAN_ LIST0	CAN List Register 0	F000 4100$_H$	U, SV	U, SV	007F 7F00$_H$
CAN_ LIST1	CAN List Register 1	F000 4104$_H$	U, SV	U, SV	0100 0000$_H$
CAN_ LIST2	CAN List Register 2	F000 4108$_H$	U, SV	U, SV	0100 0000$_H$
CAN_ LIST3	CAN List Register 3	F000 410C$_H$	U, SV	U, SV	0100 0000$_H$

Table 16-17 Address Map of CAN (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
CAN_ LIST4	CAN List Register 4	F000 4110$_H$	U, SV	U, SV	0100 0000$_H$
CAN_ LIST5	CAN List Register 5	F000 4114$_H$	U, SV	U, SV	0100 0000$_H$
CAN_ LIST6	CAN List Register 6	F000 4118$_H$	U, SV	U, SV	0100 0000$_H$
CAN_ LIST7	CAN List Register 7	F000 411C$_H$	U, SV	U, SV	0100 0000$_H$
CAN_ MSPND0	CAN Message Pending Register 0	F000 4120$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSPND1	CAN Message Pending Register 1	F000 4124$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSPND2	CAN Message Pending Register 2	F000 4128$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSPND3	CAN Message Pending Register 3	F000 412C$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSPND4	CAN Message Pending Register 4	F000 4130$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSPND5	CAN Message Pending Register 5	F000 4134$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSPND6	CAN Message Pending Register 6	F000 4138$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSPND7	CAN Message Pending Register 7	F000 413C$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MSID0	CAN Message Index Register 0	F000 4140$_H$	U, SV	U, SV	0000 0020$_H$
CAN_ MSID1	CAN Message Index Register 1	F000 4144$_H$	U, SV	U, SV	0000 0020$_H$
CAN_ MSID2	CAN Message Index Register 2	F000 4148$_H$	U, SV	U, SV	0000 0020$_H$
CAN_ MSID3	CAN Message Index Register 3	F000 414C$_H$	U, SV	U, SV	0000 0020$_H$

Table 16-17 Address Map of CAN (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
CAN_ MSID4	CAN Message Index Register 4	F000 4150$_H$	U, SV	U, SV	0000 0020$_H$
CAN_ MSID5	CAN Message Index Register 5	F000 4154$_H$	U, SV	U, SV	0000 0020$_H$
CAN_ MSID6	CAN Message Index Register 6	F000 4158$_H$	U, SV	U, SV	0000 0020$_H$
CAN_ MSID7	CAN Message Index Register 7	F000 415C$_H$	U, SV	U, SV	0000 0020$_H$
–	Reserved	F000 4160$_H$ - F000 41BC$_H$	nBE	nBE	–
CAN_ MSIMASK	CAN Message Index Mask Register	F000 41C0$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ PANCTR	CAN Panel Control Register	F000 41C4$_H$	U, SV	U, SV	0000 0301$_H$
CAN_ MCR	CAN Module Control Register	F000 41C8$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MITR	CAN Module Interrupt Trigger Register	F000 41CC$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 41D0$_H$ - F000 41FC$_H$	nBE	nBE	–
CAN Node 0 Registers					
CAN_ NCR0	CAN Node 0 Control Register	F000 4200$_H$	U, SV	U, SV	0000 0001$_H$
CAN_ NSR0	CAN Node 0 Status Register	F000 4204$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ NIPR0	CAN Node 0 Interrupt Pointer Register	F000 4208$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ NPCR0	CAN Node 0 Port Control Register	F000 420C$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ NBTR0	CAN Node 0 Bit Timing Register	F000 4210$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ NECNT0	CAN Node 0 Error Counter Register	F000 4214$_H$	U, SV	U, SV	0060 0000$_H$

Table 16-17 Address Map of CAN (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
CAN_ NFCR0	CAN Node 0 Frame Counter Register	F000 4218$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 421C$_H$ - F000 427C$_H$	nBE	nBE	–
–	Reserved	F000 4280$_H$ - F000 42F8$_H$	nBE	nBE	–
CAN Node 1 Registers					
CAN_ NCR1	CAN Node 1 Control Register	F000 4300$_H$	U, SV	U, SV	0000 0001$_H$
CAN_ NSR1	CAN Node 1 Status Register	F000 4304$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ NIPR1	CAN Node 1 Interrupt Pointer Register	F000 4308$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ NPCR1	CAN Node 1 Port Control Register	F000 430C$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ NBTR1	CAN Node 1 Bit Timing Register	F000 4310$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ NECNT1	CAN Node 1 Error Counter Register	F000 4314$_H$	U, SV	U, SV	0060 0000$_H$
CAN_ NFCR1	CAN Node 1 Frame Counter Register	F000 4318$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F000 431C$_H$ - F000 43FC$_H$	nBE	nBE	–
CAN Message Object 0					
CAN_ MOFCR0	CAN Message Object 0 Function Control Register	F000 4400$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MOFGPR0	CAN Message Object 0 FIFO Gateway Pointer Register	F000 4404$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MOIPR0	CAN Message Object 0 Interrupt Pointer Register	F000 4408$_H$	U, SV	U, SV	0000 0000$_H$

Table 16-17 Address Map of CAN (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
CAN_ MOAMR0	CAN Message Object 0 Acceptance Mask Register	F000 440C$_H$	U, SV	U, SV	3FFF FFFF$_H$
CAN_ MODATAL0	CAN Message Object 0 Data Register Low	F000 4410$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MODATAH0	CAN Message Object 0 Data Register High	F000 4414$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MOAR0	CAN Message Object 0 Arbitration Register	F000 4418$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MOSTAT0	CAN Message Object 0 Status Register (Read)	F000 441C$_H$	U, SV	–	0100 0000$_H$
CAN_ MOCTR0	CAN Message Object 0 Control Register (Write)		–	U, SV	–
CAN Message Object 1					
CAN_ MOFCR1	CAN Message Object 1 Function Control Register	F000 4420$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MOFGPR1	CAN Message Object 1 FIFO Gateway Pointer Register	F000 4424$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MOIPR1	CAN Message Object 1 Interrupt Pointer Register	F000 4428$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MOAMR1	CAN Message Object 1 Acceptance Mask Register	F000 442C$_H$	U, SV	U, SV	3FFF FFFF$_H$
CAN_ MODATAL1	CAN Message Object 1 Data Register Low	F000 4430$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MODATAH1	CAN Message Object 1 Data Register High	F000 4434$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MOAR1	CAN Message Object 1 Arbitration Register	F000 4438$_H$	U, SV	U, SV	0000 0000$_H$
CAN_ MOSTAT1	CAN Message Object 1 Status Register (Read)	F000 443C$_H$	U, SV	–	0200 0000$_H$
CAN_ MOCTR1	CAN Message Object 1 Control Register (Write)		–	U, SV	–

Table 16-17 Address Map of CAN (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
CAN Message Object 2					
CAN_ MOFCR2	CAN Message Object 2 Function Control Register	$F000\ 4440_H$	U, SV	U, SV	$0000\ 0000_H$
CAN_ MOFGPR2	CAN Message Object 2 FIFO Gateway Pointer Register	$F000\ 4444_H$	U, SV	U, SV	$0000\ 0000_H$
CAN_ MOIPR2	CAN Message Object 2 Interrupt Pointer Register	$F000\ 4448_H$	U, SV	U, SV	$0000\ 0000_H$
CAN_ MOAMR2	CAN Message Object 2 Acceptance Mask Register	$F000\ 444C_H$	U, SV	U, SV	$3FFF\ FFFF_H$
CAN_ MODATAL2	CAN Message Object 2 Data Register Low	$F000\ 4450_H$	U, SV	U, SV	$0000\ 0000_H$
CAN_ MODATAH2	CAN Message Object 2 Data Register High	$F000\ 4454_H$	U, SV	U, SV	$0000\ 0000_H$
CAN_ MOAR2	CAN Message Object 2 Arbitration Register	$F000\ 4458_H$	U, SV	U, SV	$0000\ 0000_H$
CAN_ MOSTAT2	CAN Message Object 2 Status Register (Read)	$F000\ 445C_H$	U, SV	–	$0301\ 0000_H$
CAN_ MOCTR2	CAN Message Object 2 Control Register (Write)		–	U, SV	–
CAN Message Object n (n = 3-63)					
CAN_ MOFCRn	CAN Message Object n Function Control Register	$F000\ 4400_H + n \times 20_H + 00_H$	U, SV	U, SV	$0000\ 0000_H$
CAN_ MOFGPRn	CAN Message Object n FIFO Gateway Pointer Register	$F000\ 4400_H + n \times 20_H + 04_H$	U, SV	U, SV	$0000\ 0000_H$
CAN_ MOIPRn	CAN Message Object n Interrupt Pointer Register	$F000\ 4400_H + n \times 20_H + 08_H$	U, SV	U, SV	$0000\ 0000_H$
CAN_ MOAMRn	CAN Message Object n Acceptance Mask Register	$F000\ 4400_H + n \times 20_H + 0C_H$	U, SV	U, SV	$3FFF\ FFFF_H$

Table 16-17 Address Map of CAN (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
CAN_ MOARn	CAN Message Object n Arbitration Register	$F000\,4400_H$ + n × 20_H + 10_H	U, SV	U, SV	$0000\,0000_H$
CAN_ MODATALn	CAN Message Object n Data Register Low	$F000\,4400_H$ + n × 20_H + 14_H	U, SV	U, SV	$0000\,0000_H$
CAN_ MODATAHn	CAN Message Object n Data Register High	$F000\,4400_H$ + n × 20_H + 18_H	U, SV	U, SV	$0000\,0000_H$
CAN_ MOCTRn	CAN Message Object n Control Register	$F000\,4400_H$ + n × 20_H + $1C_H$	U, SV	U, SV	(n+1) << 24 + (n-1) << 16
–	Reserved	$F000\,4500_H$ - $F000\,45FC_H$	nBE	nBE	–

1) Which Resets affect the register, see **Table 4-2**.

Table 16-18 Address Map of PCP

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
Peripheral Control Processor (PCP)					
PCP_CLC	PCP Clock Control Register	F004 3F00$_H$	U, SV	SV, E, 32	0000 0000$_H$
–	Reserved	F004 3F04$_H$	BE	BE	–
PCP_ID	PCP Module Identification Register	F004 3F08$_H$	U, SV	SV, 32	0020 C0XX$_H$
–	Reserved	F004 3F0C$_H$	BE	BE	–
PCP_CS	PCP Control/Status Register	F004 3F10$_H$	U, SV, 32	SV, E, 32	0000 0000$_H$
PCP_ES	PCP Error/Debug Status Register	F004 3F14$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	F004 3F18$_H$ - F004 3F1C$_H$	BE	BE	–
PCP_ ICR	PCP Interrupt Control Register	F004 3F20$_H$	U, SV, 32	SV, 32	0000 0000$_H$
PCP_ ITR	PCP Interrupt Threshold Register	F004 3F24$_H$	U, SV, 32	SV, 32	0000 0000$_H$
PCP_ ICON	PCP Interrupt Configuration Register	F004 3F28$_H$	U, SV, 32	SV, 32	0000 03E4$_H$
PCP_ SSR	PCP Stall Status Register	F004 3F2C$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved; this location must not be written	F004 3F30$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	F004 3F34$_H$ - F004 3FCC$_H$	BE	BE	–
PCP_ SRC11	PCP Service Request Control Register 11	F004 3FD0$_H$	U, SV, 32	SV, 32	0000 1400$_H$
PCP_ SRC10	PCP Service Request Control Register 10	F004 3FD4$_H$	U, SV, 32	SV, 32	0000 1400$_H$
PCP_ SRC9	PCP Service Request Control Register 9	F004 3FD8$_H$	U, SV, 32	SV, 32	0000 1400$_H$

Table 16-18 Address Map of PCP (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
PCP_ SRC8	PCP Service Request Control Register 8	F004 3FDC$_H$	U, SV, 32	SV, 32	0000 1000$_H$
PCP_ SRC7	PCP Service Request Control Register 7	F004 3FE0$_H$	U, SV, 32	SV, 32	0000 1000$_H$
PCP_ SRC6	PCP Service Request Control Register 6	F004 3FE4$_H$	U, SV, 32	SV, 32	0000 1000$_H$
PCP_ SRC5	PCP Service Request Control Register 5	F004 3FE8$_H$	U, SV, 32	SV, 32	0000 1000$_H$
PCP_ SRC4	PCP Service Request Control Register 4	F004 3FEC$_H$	U, SV, 32	SV, 32	0000 1000$_H$
PCP_ SRC3	PCP Service Request Control Register 3	F004 3FF0$_H$	U, SV, 32	SV, 32	0000 1400$_H$
PCP_ SRC2	PCP Service Request Control Register 2	F004 3FF4$_H$	U, SV, 32	SV, 32	0000 1400$_H$
PCP_ SRC1	PCP Service Request Control Register 1	F004 3FF8$_H$	U, SV, 32	SV, 32	0000 1000$_H$
PCP_ SRC0	PCP Service Request Control Register 0	F004 3FFC$_H$	U, SV, 32	SV, 32	0000 1000$_H$

1) Which Resets affect the register, see **Table 4-2**.

Table 16-19 Address Map of SSC0/SSC1

Short Name	Description	Address	Access Mode		Reset Value[1)]
			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	0000 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$_H$
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$_H$

Table 16-19 Address Map of SSC0/SSC1 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
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	0000 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$_H$
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$_H$

1) Which Resets affect the register, see **Table 4-2**.

Table 16-20 Address Map of FADC

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
Fast Analog-to-Digital Converter (FADC)					
FADC_ CLC	FADC Clock Control Register	$F010\ 0300_H$	U, SV	SV, E	$0000\ 0003_H$
–	Reserved	$F010\ 0304_H$	BE	BE	–
FADC_ ID	FADC Module Identification Register	$F010\ 0308_H$	U, SV	BE	$0027\ C0XX_H$
FADC_ FDR	FADC Fractional Divider Register	$F010\ 030C_H$	U, SV	SV, E	$0000\ 0000_H$
FADC_ CRSR	FADC Conversion Request Status Register	$F010\ 0310_H$	U, SV	U, SV	$0000\ 0000_H$
FADC_ FMR	FADC Flag Modification Register	$F010\ 0314_H$	U, SV	U, SV	$0000\ 0000_H$
FADC_ NCTR	FADC Neighbor Channel Trigger Register	$F010\ 0318_H$	U, SV	U, SV	$0000\ 0000_H$
FADC_ GCR	FADC Global Control Register	$F010\ 031C_H$	U, SV	U, SV	$0000\ 0000_H$
FADC_ CFGR0	FADC Configuration Register Channel 0	$F010\ 0320_H$	U, SV	U, SV	$0000\ 0000_H$
FADC_ CFGR1	FADC Configuration Register Channel 1	$F010\ 0324_H$	U, SV	U, SV	$0000\ 0000_H$
–	Reserved	$F010\ 0328_H$ - $F010\ 032C_H$	BE	BE	–
FADC_ ACR0	FADC Analog Control Register Channel 0	$F010\ 0330_H$	U, SV	U, SV	$0000\ 0000_H$
FADC_ ACR1	FADC Analog Control Register Channel 1	$F010\ 0334_H$	U, SV	U, SV	$0000\ 0000_H$
–	Reserved	$F010\ 0338_H$ - $F010\ 033C_H$	BE	BE	–
FADC_ RCH0	FADC Conversion Result Register Channel 0	$F010\ 0340_H$	U, SV	U, SV	$0000\ 0000_H$
FADC_ RCH1	FADC Conversion Result Register Channel 1	$F010\ 0344_H$	U, SV	U, SV	$0000\ 0000_H$

Table 16-20 Address Map of FADC (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
–	Reserved	F010 0348$_H$ - F010 035C$_H$	BE	BE	–
FADC_ FCR0	FADC Filter 0 Control Register	F010 0360$_H$	U, SV	U, SV	0000 0000$_H$
FADC_ CRR0	FADC Filter 0 Current Result Register	F010 0364$_H$	U, SV	U, SV	0000 0000$_H$
FADC_ IRR10	FADC Filter 0 Intermediate Result Register 1	F010 0368$_H$	U, SV	U, SV	0000 0000$_H$
FADC_ IRR20	FADC Filter 0 Intermediate Result Register 2	F010 036C$_H$	U, SV	U, SV	0000 0000$_H$
FADC_ IRR30	FADC Filter 0 Intermediate FADC Result Register 3	F010 0370$_H$	U, SV	U, SV	0000 0000$_H$
FADC_ FRR0	FADC Filter 0 Final Result Register	F010 0374$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 0378$_H$ - F010 037C$_H$	BE	BE	–
FADC_ FCR1	FADC Filter 1 Control Register	F010 0380$_H$	U, SV	U, SV	0000 0000$_H$
FADC_ CRR1	FADC Filter 1 Current Result Register	F010 0384$_H$	U, SV	U, SV	0000 0000$_H$
FADC_ IRR11	FADC Filter 1 Intermediate Result Register 1	F010 0388$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 038C$_H$ - F010 0390$_H$	nBE	nBE	–
FADC_ FRR1	FADC Filter 1 Final Result Register	F010 0394$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 0398$_H$ - F010 03EC$_H$	BE	BE	–
–	Reserved	F010 03F0$_H$ - F010 03EC$_H$	nBE	nBE	–

Table 16-20 Address Map of FADC (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
FADC_ SRC1	FADC Service Request Control Register 1	F010 03F8$_H$	U, SV	U, SV	0000 0000$_H$
FADC_ SRC0	FADC Service Request Control Register 0	F010 03FC$_H$	U, SV	U, SV	0000 0000$_H$

1) Which Resets affect the register, see **Table 4-2**.

Table 16-21 Address Map of ADC0

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
Analog-to-Digital Converter 0 (ADC0)					
ADC0_CLC	ADC Clock Control Register	$F010\ 0400_H$	U, SV	SV, E	$0000\ 0003_H$
–	Reserved	$F010\ 0404_H$	BE	BE	–
ADC0_ID	ADC Module Identification Register	$F010\ 0408_H$	U, SV	U, SV	$0030\ C0XX_H$
ADC0_FDR	ADC Fractional Divider Register	$F010\ 040C_H$	U, SV	SV, E	$0000\ 0000_H$
ADC0_CHCON0	ADC0 Channel Control Register 0	$F010\ 0410_H$	U, SV	U, SV	$0000\ 0000_H$
ADC0_CHCON1	ADC0 Channel Control Register 1	$F010\ 0414_H$	U, SV	U, SV	$0000\ 0000_H$
ADC0_CHCON2	ADC0 Channel Control Register 2	$F010\ 0418_H$	U, SV	U, SV	$0000\ 0000_H$
ADC0_CHCON3	ADC0 Channel Control Register 3	$F010\ 041C_H$	U, SV	U, SV	$0000\ 0000_H$
ADC0_CHCON4	ADC0 Channel Control Register 4	$F010\ 0420_H$	U, SV	U, SV	$0000\ 0000_H$
ADC0_CHCON5	ADC0 Channel Control Register 5	$F010\ 0424_H$	U, SV	U, SV	$0000\ 0000_H$
ADC0_CHCON6	ADC0 Channel Control Register 6	$F010\ 0428_H$	U, SV	U, SV	$0000\ 0000_H$
ADC0_CHCON7	ADC0 Channel Control Register 7	$F010\ 042C_H$	U, SV	U, SV	$0000\ 0000_H$
ADC0_CHCON8	ADC0 Channel Control Register 8	$F010\ 0430_H$	U, SV	U, SV	$0000\ 0000_H$
ADC0_CHCON9	ADC0 Channel Control Register 9	$F010\ 0434_H$	U, SV	U, SV	$0000\ 0000_H$
ADC0_CHCON10	ADC0 Channel Control Register 10	$F010\ 0438_H$	U, SV	U, SV	$0000\ 0000_H$
ADC0_CHCON11	ADC0 Channel Control Register 11	$F010\ 043C_H$	U, SV	U, SV	$0000\ 0000_H$

Table 16-21 Address Map of ADC0 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
ADC0_ CHCON12	ADC0 Channel Control Register 12	F010 0440$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHCON13	ADC0 Channel Control Register 13	F010 0444$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHCON14	ADC0 Channel Control Register 14	F010 0448$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHCON15	ADC0 Channel Control Register 15	F010 044C$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 0450$_H$ - F010 0470$_H$	BE	BE	–
–	Reserved	F010 0474$_H$	BE	BE	–
–	Reserved	F010 0478$_H$	BE	BE	–
–	Reserved	F010 047C$_H$	BE	BE	–
–	Reserved	F010 0480$_H$	BE	BE	–
ADC0_AP	ADC0 Arbitration Participation Register	F010 0484$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_SAL	ADC0 Source Arbitration Level Register	F010 0488$_H$	U, SV	U, SV	0103 4067$_H$
ADC0_TTC	ADC0 Timer Trigger Control Register	F010 048C$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ EXTC	ADC0 External Trigger Control Register	F010 0490$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 0494$_H$	BE	BE	–
ADC0_ SCON	ADC0 Source Control Register	F010 0498$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 049C$_H$	BE	BE	–
–	Reserved	F010 04E0$_H$ - F010 04FC$_H$	BE	BE	–
ADC0_ LCCON0	ADC0 Limit Check Control Register 0	F010 0500$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ LCCON1	ADC0 Limit Check Control Register 1	F010 0504$_H$	U, SV	U, SV	0000 0000$_H$

Table 16-21 Address Map of ADC0 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
ADC0_ LCCON2	ADC0 Limit Check Control Register 2	F010 0508$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ LCCON3	ADC0 Limit Check Control Register 3	F010 050C$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 0510$_H$	BE	BE	–
ADC0_ TCON	ADC0 Timer Control Register	F010 0514$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHIN	ADC0 Channel Injection Register	F010 0518$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_QR	ADC0 Queue Register	F010 051C$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_CON	ADC0 Converter Control Register	F010 0520$_H$	U, SV	U, SV	0000 0001$_H$
ADC0_ SCN	ADC0 Auto Scan Control Register	F010 0524$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ REQ0	ADC0 Conversion Request Register SW0	F010 0528$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 052C$_H$	BE	BE	–
ADC0_ CHSTAT0	ADC0 Channel Status Register 0	F010 0530$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT1	ADC0 Channel Status Register 1	F010 0534$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT2	ADC0 Channel Status Register 2	F010 0538$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT3	ADC0 Channel Status Register 3	F010 053C$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT4	ADC0 Channel Status Register 4	F010 0540$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT5	ADC0 Channel Status Register 5	F010 0544$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT6	ADC0 Channel Status Register 6	F010 0548$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT7	ADC0 Channel Status Register 7	F010 054C$_H$	U, SV	U, SV	0000 0000$_H$

Table 16-21 Address Map of ADC0 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
ADC0_ CHSTAT8	ADC0 Channel Status Register 8	F010 0550$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT9	ADC0 Channel Status Register 9	F010 0554$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT10	ADC0 Channel Status Register 10	F010 0558$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT11	ADC0 Channel Status Register 11	F010 055C$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT12	ADC0 Channel Status Register 12	F010 0560$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT13	ADC0 Channel Status Register 13	F010 0564$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT14	ADC0 Channel Status Register 14	F010 0568$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ CHSTAT15	ADC0 Channel Status Register 15	F010 056C$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ QUEUE0	ADC0 Queue Status Register	F010 0570$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 0574$_H$ - F010 057C$_H$	BE	BE	–
ADC0_ SW0CRP	ADC0 Software SW0 Conversion Request Pending Register	F010 0580$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 0584$_H$	BE	BE	–
ADC0_ ASCRP	ADC0 Auto Scan Conversion Request Pending Register	F010 0588$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 058C$_H$ - F010 059C$_H$	BE	BE	–
–	Reserved	F010 05A8$_H$ - F010 05AC$_H$	BE	BE	–
ADC0_ TSTAT	ADC0 Timer Status Register	F010 05B0$_H$	U, SV	U, SV	0000 0000$_H$

Table 16-21 Address Map of ADC0 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
ADC0_ STAT	ADC0 Converter Status Register	F010 05B4$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ TCRP	ADC0 Timer Conversion Request Pending Register	F010 05B8$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ EXCRP	ADC0 External Conversion Request Pending Register	F010 05BC$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 05C4$_H$ F010 05CC$_H$	BE	BE	–
ADC0_ MSS0	ADC0 Service Request Status Register 0	F010 05D0$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ MSS1	ADC0 Service Request Status Register 1	F010 05D4$_H$	U, SV	U, SV	0000 0000$_H$
	Reserved	F010 05D8$_H$	BE	BE	–
ADC0_ SRNP	ADC0 Service Request Node Pointer Register	F010 05DC$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 05E0$_H$ - F010 05EC$_H$	BE	BE	–
ADC0_ SRC3	ADC0 Service Request Control Register 3	F010 05F0$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ SRC2	ADC0 Service Request Control Register 2	F010 05F4$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ SRC1	ADC0 Service Request Control Register 1	F010 05F8$_H$	U, SV	U, SV	0000 0000$_H$
ADC0_ SRC0	ADC0 Service Request Control Register 0	F010 05FC$_H$	U, SV	U, SV	0000 0000$_H$

1) Which Resets affect the register, see **Table 4-2**.

Table 16-22 Address Map of MLI0/MLI1

Short Name	Description	Address	Access Mode		Reset Value[1)]
			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$_H$
MLI0_ TP3AOFR	MLI0 Transmitter Pipe 3 Address Offset Register	F010 C03C$_H$	U, SV	BE	0000 0000$_H$

Table 16-22 Address Map of MLI0/MLI1 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
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$_H$
MLI0_ RP0STATR	MLI0 Receiver Pipe 0 Status Register	F010 C07C$_H$	U, SV	BE	0000 0000$_H$

Table 16-22 Address Map of MLI0/MLI1 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
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$_H$
MLI0_ ARR	MLI0 Access Range Register	F010 C0BC$_H$	U, SV	SV, E	0000 0000$_H$

Table 16-22 Address Map of MLI0/MLI1 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
–	Reserved	F010 C0C0$_H$ - F010 C0FC$_H$	BE	BE	–
Multi 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$_H$
MLI1_ TP3AOFR	MLI1 Transmitter Pipe 3 Address Offset Register	F010 C13C$_H$	U, SV	BE	0000 0000$_H$

Table 16-22 Address Map of MLI0/MLI1 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
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$_H$
MLI1_ RP0STATR	MLI1 Receiver Pipe 0 Status Register	F010 C17C$_H$	U, SV	BE	0000 0000$_H$

Table 16-22 Address Map of MLI0/MLI1 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
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	U, 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 16-22 Address Map of MLI0/MLI1 (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
MLI1_ ARR	MLI1 Access Range Register	F010 C1BC$_H$	U, SV	SV, E	0000 0000$_H$
–	Reserved	F010 C1C0$_H$ - F010 C1FC$_H$	BE	BE	–

1) Which Resets affect the register, see **Table 4-2**.

Table 16-23 Address Map of MCHK[1)]

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
Memory Checker (MCHK)					
–	Reserved	F010 C200$_H$ - F010 C204$_H$	BE	BE	–
MCHK_ ID	Memory Checker Module Identification Register	F010 C208$_H$	U, SV	BE	001B C0XX$_H$
–	Reserved	F010 C20C$_H$	BE	BE	–
MCHK_ IR	Memory Checker Input Register	F010 C210$_H$	U, SV	U, SV	0000 0000$_H$
MCHK_ RR	Memory Checker Result Register	F010 C214$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 C218$_H$ - F010 C21C$_H$	BE	BE	–
MCHK_ WR	Write Register	F010 C220$_H$	U, SV	U, SV	0000 0000$_H$
–	Reserved	F010 C224$_H$ - F010 C2FF$_H$	BE	BE	–

1) The reset values of the memory checker registers can be modified into initial values after Boot ROM exit. See **"Hardware Status after BootROM Exit" on Page 4-17** for more details.

Table 16-24 Address Map of CPS

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
CPU Slave Interface Registers (CPS)					
–	Reserved	F7E0 FF00$_H$ - F7E0 FF04$_H$	BE	BE	–
CPS_ID	CPS Module Identification Register	F7E0 FF08$_H$	U, SV	U, SV, NC	0015 C0XX$_H$
–	Reserved	F7E0 FF0C$_H$ - FFFE FFB8$_H$	BE	BE	–
CPU_ SBSRC	CPU Software Break Service Request Control Register	F7E0 FFBC$_H$	U, SV	SV	0000 0000$_H$
–	Reserved	F7E0 FFC0$_H$ - F7E0 FFEC$_H$	BE	BE	–
CPU_SRC3	CPU Service Request Control Register 3	F7E0 FFF0$_H$	U, SV	SV	0000 0000$_H$
CPU_SRC2	CPU Service Request Control Register 2	F7E0 FFF4$_H$	U, SV	SV	0000 0000$_H$
CPU_SRC1	CPU Service Request Control Register 1	F7E0 FFF8$_H$	U, SV	SV	0000 0000$_H$
CPU_SRC0	CPU Service Request Control Register 0	F7E0 FFFC$_H$	U, SV	SV	0000 0000$_H$

1) Which Resets affect the register, see **Table 4-2**.

Table 16-25 Address Map of CPU Core SFRs & GPRs

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
CPU Core SFRs & GPRs					
–	Reserved	F7E1 0000$_H$ - F7E1 7FFC$_H$	BE	BE	–
MMU_CON	MMU Configuration Register	F7E1 8000$_H$	U, SV, 32	SV, 32	0000 8000$_H$
–	Reserved	F7E1 8004$_H$ - F7E1 803C$_H$	U, SV, 32	SV, 32[1)]	–
–	Reserved	F7E1 8400$_H$ - F7E1 BFFC$_H$	BE	BE	–
Memory Protection Registers					
DPR0_0L	Data Seg. Protect. Reg. 0, Set 0, Lower Bound	F7E1 C000$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR0_0U	Data Seg. Protect. Reg. 0, Set 0, Upper Bound	F7E1 C004$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR0_1L	Data Seg. Protect. Reg. 1, Set 0, Lower Bound	F7E1 C008$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR0_1U	Data Seg. Protect. Reg. 1, Set 0, Upper Bound	F7E1 C00C$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR0_2L	Data Seg. Protect. Reg. 2, Set 0, Lower Bound	F7E1 C010$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR0_2U	Data Seg. Protect. Reg. 2, Set 0, Upper Bound	F7E1 C014$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR0_3L	Data Seg. Protect. Reg. 3, Set 0, Lower Bound	F7E1 C018$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR0_3U	Data Seg. Protect. Reg. 3, Set 0, Upper Bound	F7E1 C01C$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	F7E1 C020$_H$ - F7E1 C3FC$_H$	nE	nE	–
DPR1_0L	Data Seg. Protect. Reg. 0, Set 1, Lower Bound	F7E1 C400$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR1_0U	Data Seg. Protect. Reg. 0, Set 1, Upper Bound	F7E1 C404$_H$	U, SV, 32	SV, 32	0000 0000$_H$

Table 16-25 Address Map of CPU Core SFRs & GPRs (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
DPR1_1L	Data Seg. Protect. Reg. 1, Set 1, Lower Bound	F7E1 C408$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR1_1U	Data Seg. Protect. Reg. 1, Set 1, Upper Bound	F7E1 C40C$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR1_2L	Data Seg. Protect. Reg. 2, Set 1, Lower Bound	F7E1 C410$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR1_2U	Data Seg. Protect. Reg. 2, Set 1, Upper Bound	F7E1 C414$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR1_3L	Data Seg. Protect. Reg. 3, Set 1, Lower Bound	F7E1 C418$_H$	U, SV, 32	SV, 32	0000 0000$_H$
DPR1_3U	Data Seg. Protect. Reg. 3, Set 1, Upper Bound	F7E1 C41C$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	F7E1 C420$_H$ - 7E1 CFFC$_H$	nE	nE	–
CPR0_0L	Code Seg. Prot. Reg. 0, Set 0, Lower Bound	F7E1 D000$_H$	U, SV, 32	SV, 32	0000 0000$_H$
CPR0_0U	Code Seg. Prot. Reg. 0, Set 0, Upper Bound	F7E1 D004$_H$	U, SV, 32	SV, 32	0000 0000$_H$
CPR0_1L	Code Seg. Prot. Reg. 1, Set 0, Lower Bound	F7E1 D008$_H$	U, SV, 32	SV, 32	0000 0000$_H$
CPR0_1U	Code Seg. Prot. Reg. 1, Set 0, Upper Bound	F7E1 D00C$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	F7E1 D010$_H$ - F7E1 D3FC$_H$	nE	nE	–
CPR1_0L	Code Seg. Prot. Reg. 0, Set 1, Lower Bound	F7E1 D400$_H$	U, SV, 32	SV, 32	0000 0000$_H$
CPR1_0U	Code Seg. Prot. Reg. 0, Set 1, Upper Bound	F7E1 D404$_H$	U, SV, 32	SV, 32	0000 0000$_H$
CPR1_1L	Code Seg. Prot. Reg. 1, Set 1, Lower Bound	F7E1 D408$_H$	U, SV, 32	SV, 32	0000 0000$_H$
CPR1_1U	Code Seg. Prot. Reg. 1, Set 1, Upper Bound	F7E1 D40C$_H$	U, SV, 32	SV, 32	0000 0000$_H$

Table 16-25 Address Map of CPU Core SFRs & GPRs (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
–	Reserved	F7E1 D410$_H$ - F7E1 DFFC$_H$	nE	nE	–
DPM0	Data Protection Mode Register 0	F7E1 E000$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	F7E1 E004$_H$ - F7E1 E07C$_H$	nE	nE	–
DPM1	Data Protection Mode Register 1	F7E1 E080$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	F7E1 E084$_H$ - F7E1 E1FC$_H$	nE	nE	–
CPM0	Code Protection Mode Register 0	F7E1 E200$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	F7E1 E204$_H$ - F7E1 E27C$_H$	nE	nE	–
CPM1	Code Protection Mode Register 1	F7E1 E280$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	F7E1 E284$_H$ - F7E1 EFFC$_H$	nE	nE	–
Core Debug Register (OCDS)					
DBGSR	Debug Status Register	F7E1 FD00$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	F7E1 FD04$_H$	nE	nE	–
EXEVT	External Break Input Event Specifier Register	F7E1 FD08$_H$	U, SV, 32	SV, 32	0000 0000$_H$
CREVT	Core SFR Access Break Event Specifier Register	F7E1 FD0C$_H$	U, SV, 32	SV, 32	0000 0000$_H$
SWEVT	Debug Instruction Break Event Specifier Register	F7E1 FD10$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	F7E1 FD14$_H$ - F7E1 FD1C$_H$	nE	nE	–
TR0EVT	Trigger Event 0 Specifier Register	F7E1 FD20$_H$	U, SV, 32	SV, 32	0000 0000$_H$

Table 16-25 Address Map of CPU Core SFRs & GPRs (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
TR1EVT	Trigger Event 1 Specifier Register	F7E1 FD24$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	F7E1 FD28$_H$ - F7E1 FD3C$_H$	nE	nE	–
DMS	Debug Monitor Start Address Register	F7E1 FD40$_H$	U, SV, 32	U, SV, 32, NC	DE00 0000$_H$
DCX	Debug Context Save Area Pointer	F7E1 FD44$_H$	U, SV, 32	U, SV, 32	DE80 0000$_H$
–	Reserved	F7E1 FD48$_H$ - F7E1 FDFC$_H$	nE	nE	–
Core Special Function Registers (CSFR)					
PCXI	Previous Context Information Register	F7E1 FE00$_H$	U, SV, 32	SV, 32	0000 0000$_H$
PSW	Program Status Word	F7E1 FE04$_H$	U, SV, 32	SV, 32	0000 0B80$_H$
PC	Program Counter	F7E1 FE08$_H$	U, SV, 32	SV, 32	DFFF FFFC$_H$ (Boot ROM boot)
–	Reserved	F7E1 FE0C$_H$ - F7E1 FE10$_H$	nE	nE	–
SYSCON	System Configuration Register	F7E1 FE14$_H$	U, SV, 32	SV, 32	0000 0000$_H$
CPU_ID	CPU Identification Register	F7E1 FE18$_H$	U, SV, 32	U, SV, NC, 32	000A C0XX$_H$
–	Reserved	F7E1 FE1C$_H$	nE	nE	–
BIV	Interrupt Vector Table Pointer	F7E1 FE20$_H$	U, SV, 32	SV, E, 32	0000 0000$_H$
BTV	Trap Vector Table Pointer	F7E1 FE24$_H$	U, SV, 32	SV, E, 32	A000 0100$_H$
ISP	Interrupt Stack Pointer	F7E1 FE28$_H$	U, SV, 32	SV, E, 32	0000 0100$_H$

Table 16-25 Address Map of CPU Core SFRs & GPRs (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
ICR	ICU Interrupt Control Register	F7E1 FE2C$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	Reserved	F7E1 FE30$_H$ - F7E1 FE34$_H$	nE	nE	–
FCX	Free Context List Head Pointer	F7E1 FE38$_H$	U, SV, 32	SV, 32	0000 0000$_H$
LCX	Free Context List Limit Pointer	F7E1 FE3C$_H$	U, SV, 32	SV, 32	0000 0000$_H$
–	–	F7E1 FE40$_H$ - F7E1 FEFC$_H$	nE	nE	–
CPU Core General Purpose Register (GPRs)[2)]					
D0	Data Register D0 (DGPR)	F7E1 FF00$_H$	–	–	XXXX XXXX$_H$
D1	Data Register D1 (DGPR)	F7E1 FF04$_H$	–	–	XXXX XXXX$_H$
D2	Data Register D2 (DGPR)	F7E1 FF08$_H$	–	–	XXXX XXXX$_H$
D3	Data Register D3 (DGPR)	F7E1 FF0C$_H$	–	–	XXXX XXXX$_H$
D4	Data Register D4 (DGPR)	F7E1 FF10$_H$	–	–	XXXX XXXX$_H$
D5	Data Register D5 (DGPR)	F7E1 FF14$_H$	–	–	XXXX XXXX$_H$
D6	Data Register D6 (DGPR)	F7E1 FF18$_H$	–	–	XXXX XXXX$_H$
D7	Data Register D7 (DGPR)	F7E1 FF1C$_H$	–	–	XXXX XXXX$_H$
D8	Data Register D8 (DGPR)	F7E1 FF20$_H$	–	–	XXXX XXXX$_H$
D9	Data Register D9 (DGPR)	F7E1 FF24$_H$	–	–	XXXX XXXX$_H$
D10	Data Register 10 (DGPR)	F7E1 FF28$_H$	–	–	XXXX XXXX$_H$

Table 16-25 Address Map of CPU Core SFRs & GPRs (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
D11	Data Register 11 (DGPR)	F7E1 FF2C$_H$	–	–	XXXX XXXX$_H$
D12	Data Register 12 (DGPR)	F7E1 FF30$_H$	–	–	XXXX XXXX$_H$
D13	Data Register 13 (DGPR)	F7E1 FF34$_H$	–	–	XXXX XXXX$_H$
D14	Data Register 14 (DGPR)	F7E1 FF38$_H$	–	–	XXXX XXXX$_H$
D15	Data Register 15 (DGPR)	F7E1 FF3C$_H$	–	–	XXXX XXXX$_H$
–	Reserved	F7E1 FF40$_H$ - F7E1 FF7C$_H$	nE	nE	–
A0	Address Reg. 0 (AGPR) Global Address Register	F7E1 FF80$_H$	–	–	XXXX XXXX$_H$
A1	Address Reg. 1 (AGPR) Global Address Register	F7E1 FF84$_H$	–	–	XXXX XXXX$_H$
A2	Address Register 2 (AGPR)	F7E1 FF88$_H$	–	–	XXXX XXXX$_H$
A3	Address Register 3 (AGPR)	F7E1 FF8C$_H$	–	–	XXXX XXXX$_H$
A4	Address Register 4 (AGPR)	F7E1 FF90$_H$	–	–	XXXX XXXX$_H$
A5	Address Register 5 (AGPR)	F7E1 FF94$_H$	–	–	XXXX XXXX$_H$
A6	Address Register 6 (AGPR)	F7E1 FF98$_H$	–	–	XXXXXXXX$_H$
A7	Address Register 7 (AGPR)	F7E1 FF9C$_H$	–	–	XXXXXXXX$_H$
A8	Address Reg. 8 (AGPR) Global Address Register	F7E1 FFA0$_H$	–	–	XXXXXXXX$_H$
A9	Address Reg. 9 (AGPR) Global Address Register	F7E1 FFA4$_H$	–	–	XXXXXXXX$_H$

Table 16-25 Address Map of CPU Core SFRs & GPRs (cont'd)

Short Name	Description	Address	Access Mode		Reset Value
			Read	Write	
A10 (SP)	Address Reg. 10 (AGPR) Stack Pointer	F7E1 FFA8$_H$	–	–	XXXX XXXX$_H$
A11 (RA)	Address Reg. 11 (AGPR) Return Address	F7E1 FFAC$_H$	–	–	XXXX XXXX$_H$
A12	Address Reg. 12 (AGPR)	F7E1 FFB0$_H$	–	–	XXXX XXXX$_H$
A13	Address Reg. 13 (AGPR)	F7E1 FFB4$_H$	–	–	XXXX XXXX$_H$
A14	Address Reg. 14 (AGPR)	F7E1 FFB8$_H$	–	–	XXXX XXXX$_H$
A15	Address Reg. 15 (AGPR)	F7E1 FFBC$_H$	–	–	XXXX XXXX$_H$
–	Reserved	F7E1 FFC0$_H$ - F7E1 FFFC$_H$	nE	nE	–

1) Read/write operations from/to these locations have no effect.

2) Note that all GPRs are undefined (XXXX XXXX$_B$) after a reset operation.

Table 16-26 Address Map of PMU

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
Program Memory Unit (PMU)					
–	Reserved	F800 0500$_H$ - F800 0504$_H$	BE	BE	–
PMU_ID	PMU Module Identification Register	F800 0508$_H$	U, SV	BE	002E C0XX$_H$
–	Reserved	F800 050C$_H$	BE	BE	–
–	Reserved	F800 0510$_H$	nBE	nBE	–
–	Reserved	F800 0514$_H$ - F800 051C$_H$	BE	BE	–
PMU_ RABR0	PMU Redirected Address Base Register 0	F800 0520$_H$	U, SV	SV	0000 0000$_H$
PMU_ OTAR0	PMU Overlay Target Address Register 0	F800 0524$_H$	U, SV	SV	0000 0000$_H$
PMU_ OMASK0	PMU Overlay Mask Register 0	F800 0528$_H$	U, SV	SV	0FFF FE00$_H$
PMU_ RABR1	PMU Redirected Address Base Register 1	F800 052C$_H$	U, SV	SV	0000 0000$_H$
PMU_ OTAR1	PMU Overlay Target Address Register 1	F800 0530$_H$	U, SV	SV	0000 0000$_H$
PMU_ OMASK1	PMU Overlay Mask Register 1	F800 0534$_H$	U, SV	SV	0FFF FE00$_H$
PMU_ RABR2	PMU Redirected Address Base Register 2	F800 0538$_H$	U, SV	SV	0000 0000$_H$
PMU_ OTAR2	PMU Overlay Target Address Register 2	F800 053C$_H$	U, SV	SV	0000 0000$_H$
PMU_ OMASK2	PMU Overlay Mask Register 2	F800 0540$_H$	U, SV	SV	0FFF FE00$_H$
PMU_ RABR3	PMU Redirected Address Base Register 3	F800 0544$_H$	U, SV	SV	0000 0000$_H$
PMU_ OTAR3	PMU Overlay Target Address Register 3	F800 0548$_H$	U, SV	SV	0000 0000$_H$

Table 16-26 Address Map of PMU (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
PMU_ OMASK3	PMU Overlay Mask Register 3	F800 054C$_H$	U, SV	SV	0FFF FE00$_H$
PMU_ RABR4	PMU Redirected Address Base Register 4	F800 0550$_H$	U, SV	SV	0000 0000$_H$
PMU_ OTAR4	PMU Overlay Target Address Register 4	F800 0554$_H$	U, SV	SV	0000 0000$_H$
PMU_ OMASK4	PMU Overlay Mask Register 4	F800 0558$_H$	U, SV	SV	0FFF FE00$_H$
PMU_ RABR5	PMU Redirected Address Base Register 5	F800 055C$_H$	U, SV	SV	0000 0000$_H$
PMU_ OTAR5	PMU Overlay Target Address Register 5	F800 0560$_H$	U, SV	SV	0000 0000$_H$
PMU_ OMASK5	PMU Overlay Mask Register 5	F800 0564$_H$	U, SV	SV	0FFF FE00$_H$
PMU_ RABR6	PMU Redirected Address Base Register 6	F800 0568$_H$	U, SV	SV	0000 0000$_H$
PMU_ OTAR6	PMU Overlay Target Address Register 6	F800 056C$_H$	U, SV	SV	0000 0000$_H$
PMU_ OMASK6	PMU Overlay Mask Register 6	F800 0570$_H$	U, SV	SV	0FFF FE00$_H$
PMU_ RABR7	PMU Redirected Address Base Register 7	F800 0574$_H$	U, SV	SV	0000 0000$_H$
PMU_ OTAR7	PMU Overlay Target Address Register 7	F800 0578$_H$	U, SV	SV	0000 0000$_H$
PMU_ OMASK7	PMU Overlay Mask Register 7	F800 057C$_H$	U, SV	SV	0FFF FE00$_H$
PMU_ RABR8	PMU Redirected Address Base Register 8	F800 0580$_H$	U, SV	SV	0000 0000$_H$
PMU_ OTAR8	PMU Overlay Target Address Register 8	F800 0584$_H$	U, SV	SV	0000 0000$_H$
PMU_ OMASK8	PMU Overlay Mask Register 8	F800 0588$_H$	U, SV	SV	0FFF FE00$_H$

Table 16-26 Address Map of PMU (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
PMU_ RABR9	PMU Redirected Address Base Register 9	F800 058C$_H$	U, SV	SV	0000 0000$_H$
PMU_ OTAR9	PMU Overlay Target Address Register 9	F800 0590$_H$	U, SV	SV	0000 0000$_H$
PMU_ OMASK9	PMU Overlay Mask Register 9	F800 0594$_H$	U, SV	SV	0FFF FE00$_H$
PMU_ RABR10	PMU Redirected Address Base Register 10	F800 0598$_H$	U, SV	SV	0000 0000$_H$
PMU_ OTAR10	PMU Overlay Target Address Register 10	F800 059C$_H$	U, SV	SV	0000 0000$_H$
PMU_ OMASK10	PMU Overlay Mask Register 10	F800 05A0$_H$	U, SV	SV	0FFF FE00$_H$
PMU_ RABR11	PMU Redirected Address Base Register 11	F800 05A4$_H$	U, SV	SV	0000 0000$_H$
PMU_ OTAR11	PMU Overlay Target Address Register 11	F800 05A8$_H$	U, SV	SV	0000 0000$_H$
PMU_ OMASK11	PMU Overlay Mask Register 11	F800 05AC$_H$	U, SV	SV	0FFF FE00$_H$
PMU_ RABR12	PMU Redirected Address Base Register 12	F800 05B0$_H$	U, SV	SV	0000 0000$_H$
PMU_ OTAR12	PMU Overlay Target Address Register 12	F800 05B4$_H$	U, SV	SV	0000 0000$_H$
PMU_ OMASK12	PMU Overlay Mask Register 12	F800 05B8$_H$	U, SV	SV	0FFF FE00$_H$
PMU_ RABR13	PMU Redirected Address Base Register 13	F800 05BC$_H$	U, SV	SV	0000 0000$_H$
PMU_ OTAR13	PMU Overlay Target Address Register 13	F800 05C0$_H$	U, SV	SV	0000 0000$_H$
PMU_ OMASK13	PMU Overlay Mask Register 13	F800 05C4$_H$	U, SV	SV	0FFF FE00$_H$
PMU_ RABR14	PMU Redirected Address Base Register 14	F800 05C8$_H$	U, SV	SV	0000 0000$_H$

Table 16-26 Address Map of PMU (cont'd)

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
PMU_ OTAR14	PMU Overlay Target Address Register 14	F800 05CC$_H$	U, SV	SV	0000 0000$_H$
PMU_ OMASK14	PMU Overlay Mask Register 14	F800 05D0$_H$	U, SV	SV	0FFF FE00$_H$
PMU_ RABR15	PMU Redirected Address Base Register 15	F800 05D4$_H$	U, SV	SV	0000 0000$_H$
PMU_ OTAR15	PMU Overlay Target Address Register 15	F800 05D8$_H$	U, SV	SV	0000 0000$_H$
PMU_ OMASK15	PMU Overlay Mask Register 15	F800 05DC$_H$	U, SV	SV	0FFF FE00$_H$
–	Reserved	F800 05E0$_H$ - F800 05EC$_H$	BE	BE	–
–	Reserved[2)]	F800 05F0$_H$	U, SV	E, U, SV	–
–	Reserved	F800 05F4$_H$	BE	BE	–
–	Reserved[2)]	F800 05F8$_H$	BE	E, U, SV, 32	–
–	Reserved[2)]	F800 05FC$_H$	U, SV	BE	–

1) Which Resets affect the register, see **Table 4-2**.

2) Do not read from or write to these address locations. Otherwise, unpredictable results may occur.

Table 16-27 Address Map of Flash

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
Flash Register					
–	Reserved	F800 1000_H - F800 $1FFC_H$	BE	BE	–
–	Reserved	F800 2000_H - F800 2004_H	BE	BE	–
FLASH_ID	Flash Module Identification Register	F800 2008_H	U, SV	BE	0041 $C0XX_H$
–	Reserved	F800 $200C_H$	BE	BE	–
FLASH_ FCON	Flash Configuration Register	F800 2014_H	U, SV	SV, E	000X 0666_H
FLASH_ MARP	Flash Margin Control Register PFlash	F800 2018_H	U, SV	SV, E	0000 8000_H
FLASH_ MARD	Flash Margin Control Register DFlash	F800 $201C_H$	U, SV	U, SV	0000 8000_H
FLASH_ PROCON0	Flash Protection Configuration User 0	F800 2020_H	U, SV	BE	0000 $XXXX_H$
FLASH_ PROCON1	Flash Protection Configuration User 1	F800 2024_H	U, SV	BE	0000 $XXXX_H$
FLASH_ PROCON2	Flash Protection Configuration User 2	F800 2028_H	U, SV	BE	0000 $XXXX_H$
–	Reserved	F800 $202C_H$ - F800 $20FC_H$	BE	BE	–
–	Reserved	F800 $210C_H$ - F800 $23FC_H$	BE	BE	–

1) Which Resets affect the register, see **Table 4-2**.

Table 16-28 Address Map of DMI

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
DMI Registers					
–	Reserved	F87F FC00$_H$ - F87F FC04$_H$	2)3)		–
DMI_ID	DMI Module Identification Register	F87F FC08$_H$	U, SV	2)	0008 C0XX$_H$
–	Reserved	F87F FC0C$_H$	2)3)		–
DMI_CON	DMI Control Register	F87F FC10$_H$	U, SV	SV, E	0000 0060$_H$
–	Reserved	F87F FC14$_H$	2)3)		–
DMI_STR	DMI Synchronous Trap Flag Register	F87F FC18$_H$	U, SV 2)4)	1)	0000 0000$_H$
–	Reserved	F87F FC1C$_H$	2)3)		–
DMI_ATR	DMI Asynchronous Trap Flag Register	F87F FC20$_H$	U, SV 2)4)	1)	0000 0000$_H$
–	Reserved	F87F FC24$_H$	2)3)		–
DMI_CON1	DMI Control Register 1	F87F FC28$_H$	U, SV	SV, E	0000 0000$_H$
–	Reserved	F87F FC2C$_H$ - F87F FCFC$_H$	2)3)		–

1) **Which Resets affect the register, see Table 4-2.**

2) Access to the 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 FPI Bus, or to a trap, flagged with a DMI Control Register Error Flag (see DMI_STR/DMI_ATR registers) in case of a CPU load/store access.

3) An access to these reserved locations will not be flagged with an error. A read will return all zeros, a write will have no effect.

4) Reading this register 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.

Table 16-29 Address Map of PMI

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
PMI Registers					
–	Reserved	F87F FD00$_H$ - F87F FD04$_H$	BE	BE	–
PMI_ID	PMI Module Identification Register	F87F FD08$_H$	U, SV, 32	U, SV, NC, 32	000B C0XX$_H$
–	Reserved	F87F FD0C$_H$	BE	BE	–
PMI_CON0	PMI Control Register 0	F87F FD10$_H$	U, SV, 32	SV, E, 32	0000 0002$_H$
PMI_CON1	PMI Control Register 1	F87F FD14$_H$	U, SV, 32	SV, 32	0000 0000$_H$
PMI_CON2	PMI Control Register 2	F87F FD18$_H$	U, SV, 32	SV, NC, 32	0000 0022$_H$
–	Reserved	F87F FD1C$_H$ - F87F FDFC$_H$	BE	BE	–

1) Which Resets affect the register, see **Table 4-2**.

Table 16-30 Address Map of LBCU

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
Local Memory Bus Control Unit (LBCU)					
–	Reserved	F87F FE00$_H$ - F87F FE04$_H$	BE	BE	–
LBCU_ID	LBCU Module Identification Register	F87F FE08$_H$	U, SV	BE	000F C0XX$_H$
–	Reserved	F87F FE0C$_H$ - F87F FE18$_H$	BE	BE	–
LBCU_ LEATT	LBCU LMB Error Attributes Register	F87F FE20$_H$	U, SV, 32	SV, 32	XXXX XXX0$_H$
LBCU_ LEADDR	LBCU LMB Error Address Register	F87F FE24$_H$	U, SV	BE	XXXX XXXX$_H$
LBCU_ LEDATL	LBCU LMB Error Data Low Register	F87F FE28$_H$	U, SV, 64	BE	XXXX XXXX$_H$
LBCU_ LEDATH	LBCU LMB Error Data High Register	F87F FE2C$_H$	U, SV		XXXX XXXX$_H$
–	Reserved	F87F FE30$_H$ - F87F FEF8$_H$	BE	BE	–
LBCU_ SRC	LBCU Service Request Control Register	F87F FEFC$_H$	U, SV, 32	SV, 32	0000 0000$_H$

1) Which Resets affect the register, see **Table 4-2**.

Table 16-31 Address Map of LFI-Bridge

Short Name	Description	Address	Access Mode		Reset Value[1)]
			Read	Write	
LMB to SPB Bus Bridge (LFI)					
–	Reserved	$F87F\ FF00_H$ - $F87F\ FF04_H$	BE	BE	–
LFI_ID	LFI Module Identification Register	$F87F\ FF08_H$	U, SV	BE	$000C\ C0XX_H$
–	Reserved	$F87F\ FF0C_H$	BE	BE	–
LFI_CON	LFI Configuration Register	$F87F\ FF10_H$	U, SV	SV	$0000\ 0B00_H$
–	Reserved	$F87F\ FF14_H$ - $F87F\ FFFF_H$	BE	BE	–

1) Which Resets affect the register, see **Table 4-2**.

17 Asynchronous/Synchronous Serial Interface (ASC)

This chapter describes the two ASC Asynchronous/Synchronous Serial Interfaces, ASC0 and ASC1, of the TC1766. It contains the following sections:

- Functional description of the ASC kernel, valid for ASC0 and ASC1 (see **Page 17-1**)
- ASC kernel register description, describes all ASC kernel specific registers (see **Page 17-19**)
- TC1766 implementation-specific details and registers of the ASC0/ASC1 modules (see **Page 17-31**).

*Note: The ASC kernel register names described in **Section 17.2** are referenced in the TC1766 User's Manual by the module name prefix "ASC0_" for the ASC0 interface and by "ASC1_" for the ASC1 interface.*

17.1 ASC Kernel Description

Figure 17-1 shows a global view of the ASC interface.

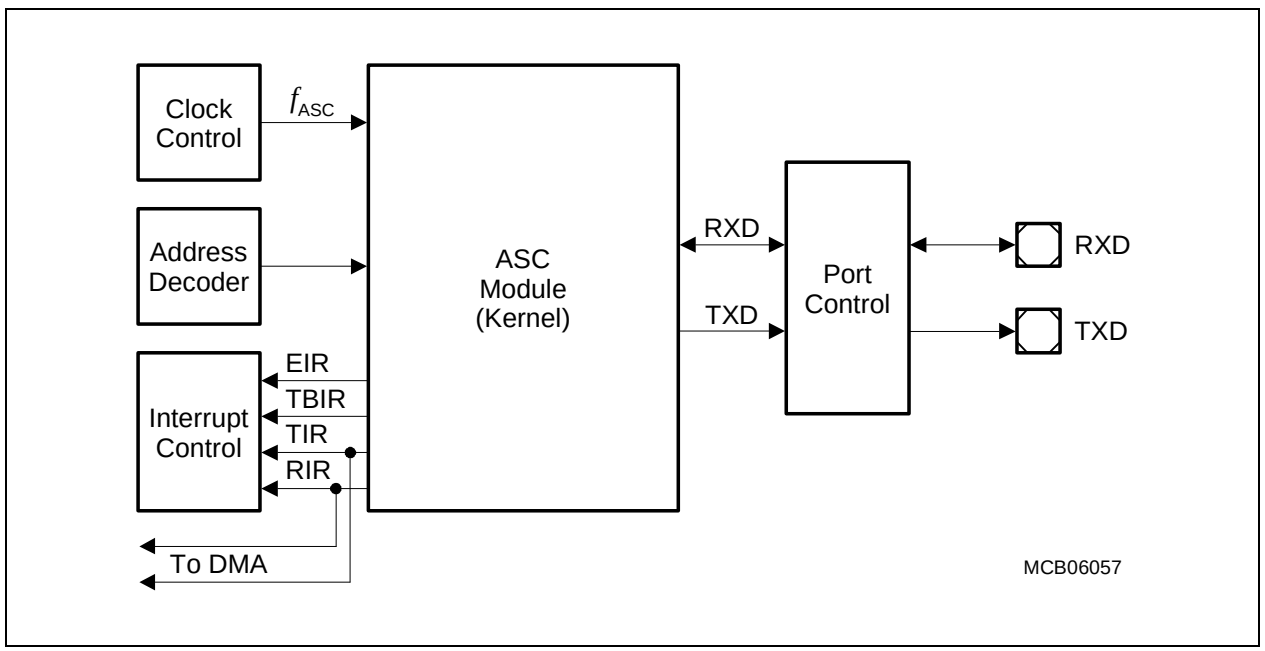


Figure 17-1 General Block Diagram of the ASC Interface

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.

17.1.1 Overview

The ASC provides serial communication between the TC1766 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 5.0 Mbit/s to 1.19 bit/s (@ 80 MHz module clock)
  - Multiprocessor mode for automatic address/data byte detection
  - Loop-back capability
- Half-duplex 8-bit synchronous operating mode
  - Baud rate from 10.0 Mbit/s to 813.8 bit/s (@ 80 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)

17.1.2 General Operation

The ASC supports full-duplex asynchronous communication up to 5.0 Mbit/s and half-duplex synchronous communication up to 10.0 Mbit/s (@ 80 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.

17.1.3 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 17-2** shows the block diagram of the ASC when operating in Asynchronous Mode.

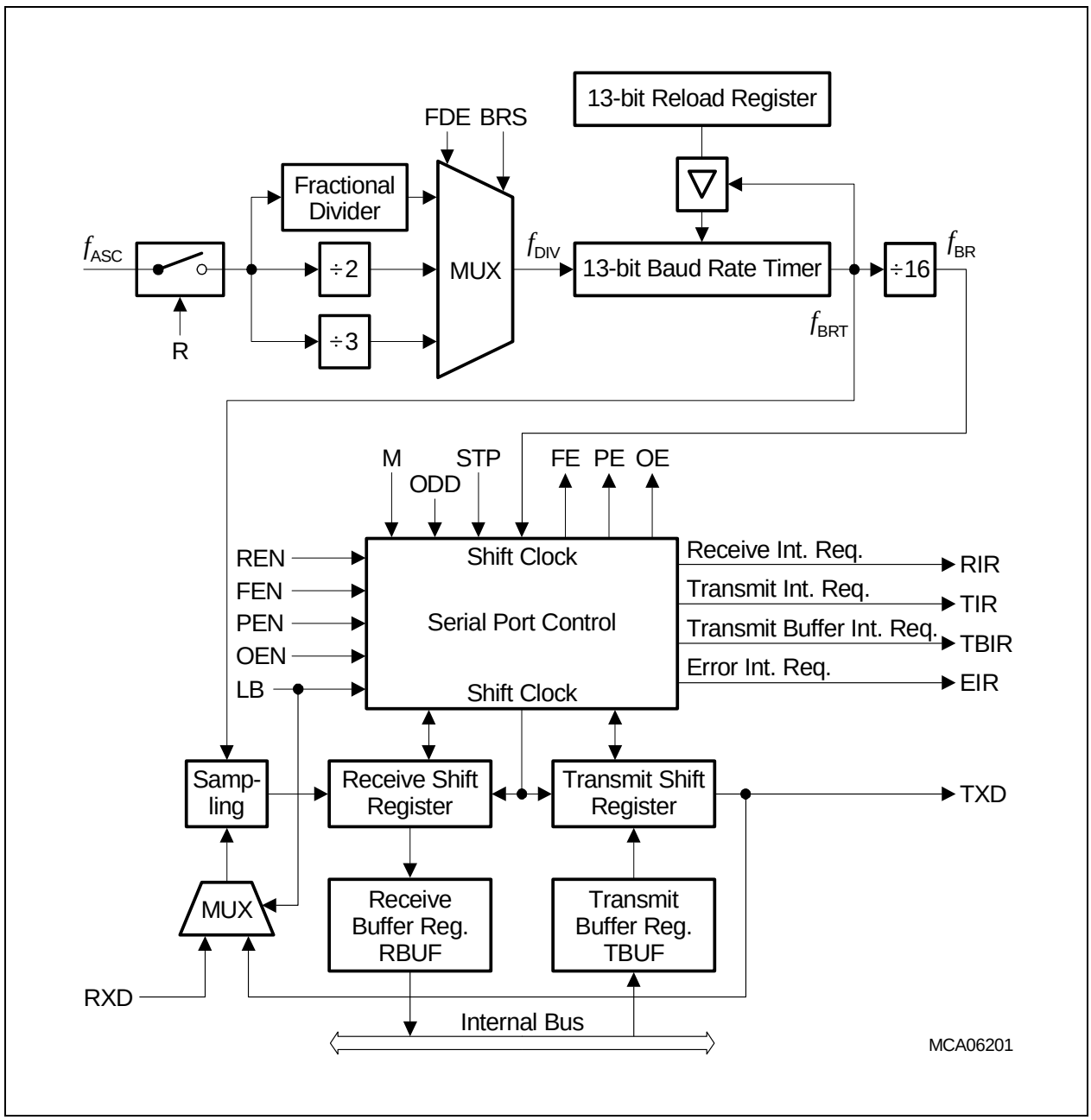


Figure 17-2 Asynchronous Mode of the ASC

17.1.3.1 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.

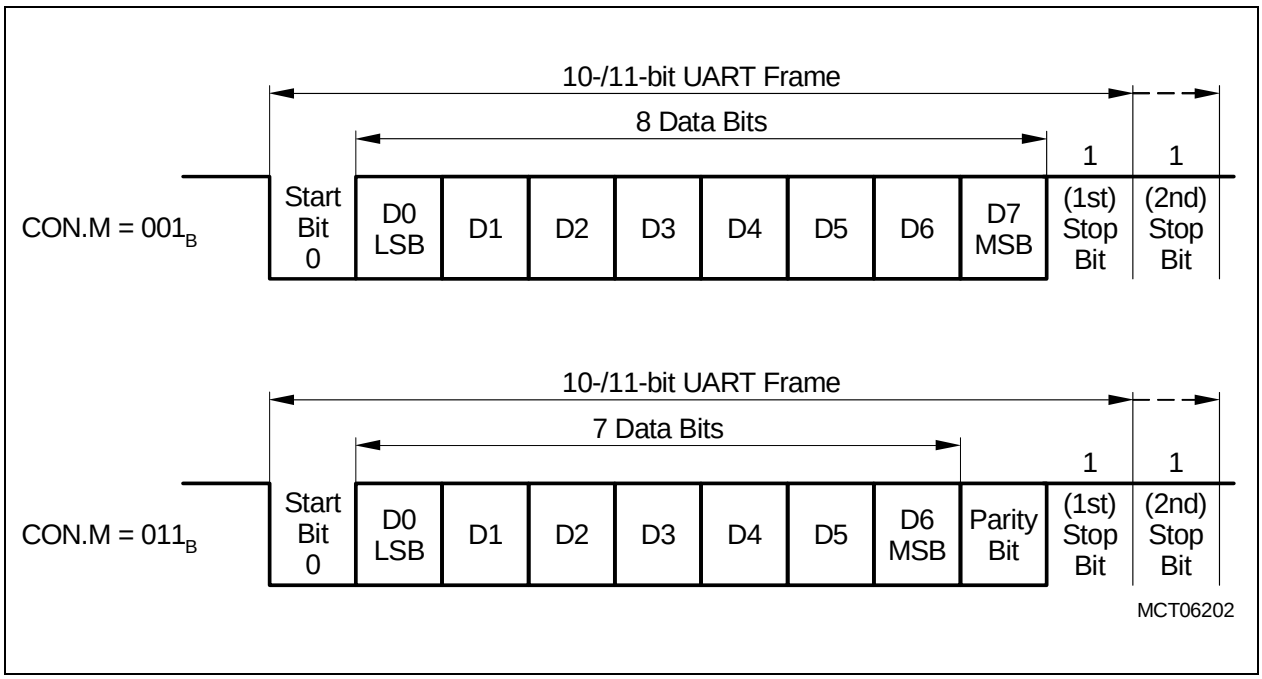


Figure 17-3 Asynchronous 8-bit Frames

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.

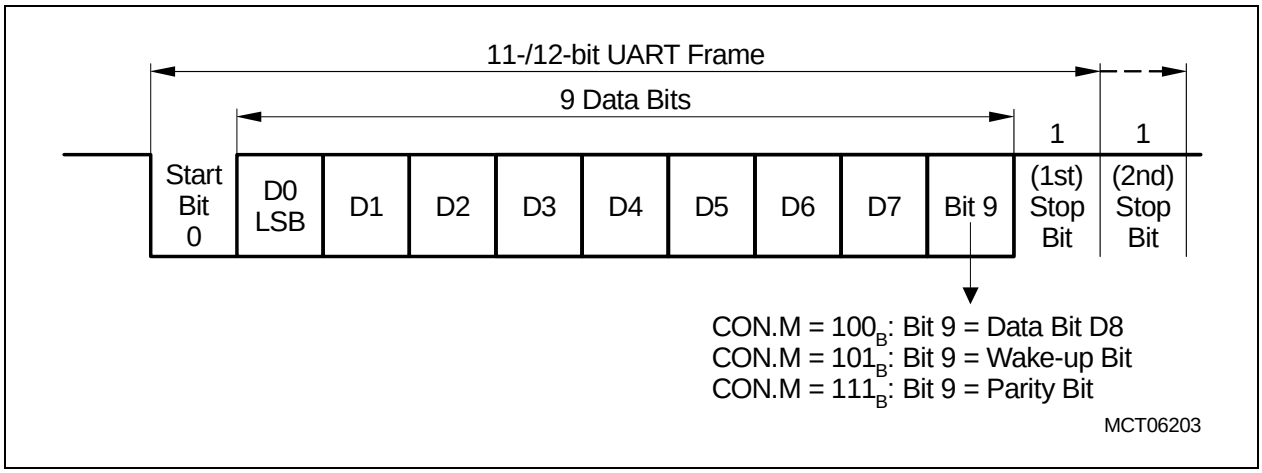


Figure 17-4 Asynchronous 9-bit Frames

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’.

17.1.3.2 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.

17.1.3.3 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 timeslot (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.

17.1.3.4 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 (loopback) as shown in **Figure 17-5**.

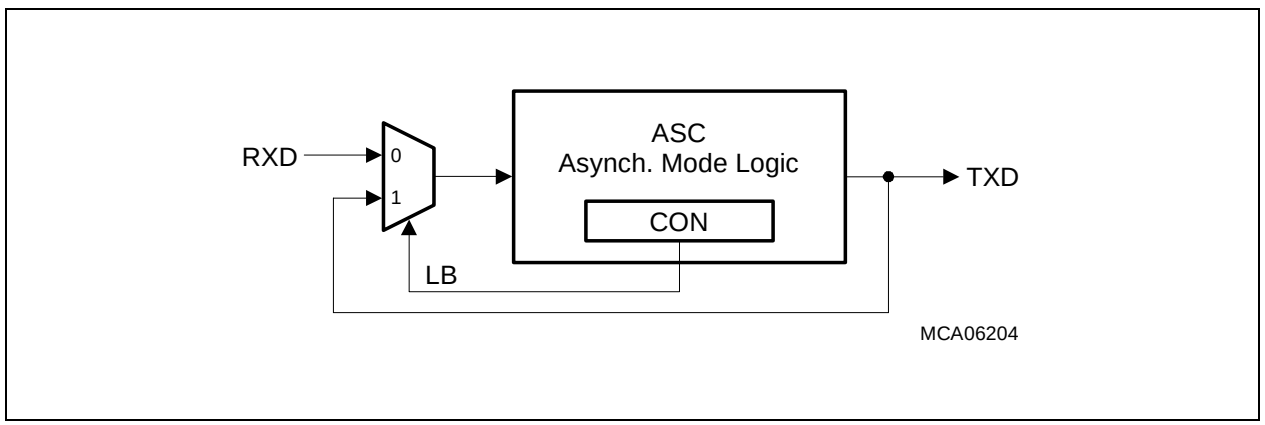


Figure 17-5 RXD/TXD Data Path Selection in Asynchronous Modes

17.1.4 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.

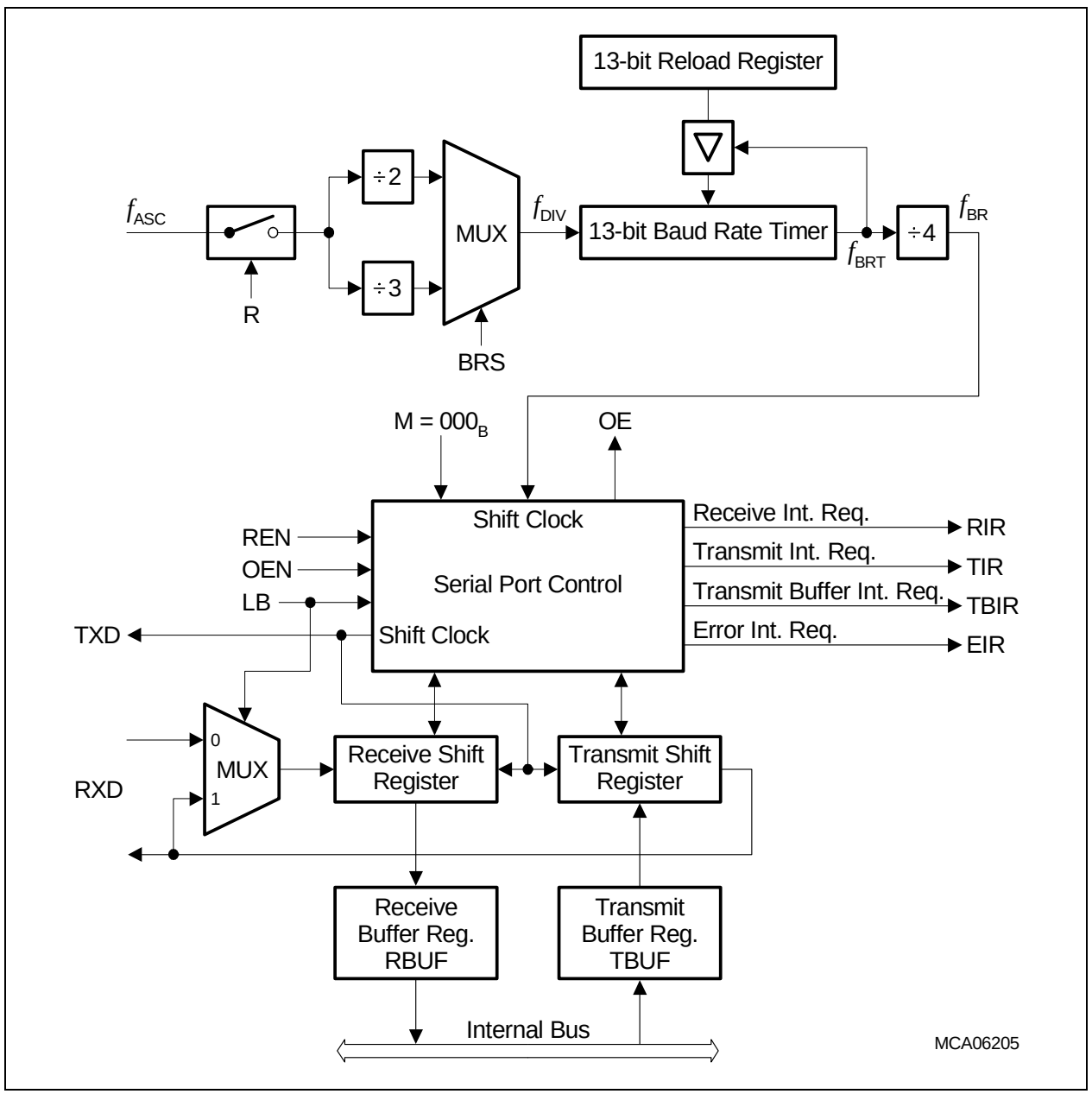


Figure 17-6 Synchronous Mode of Serial Channel ASC

17.1.4.1 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.

17.1.4.2 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 cleared, 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.

17.1.4.3 Synchronous Timing

Figure 17-7 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.

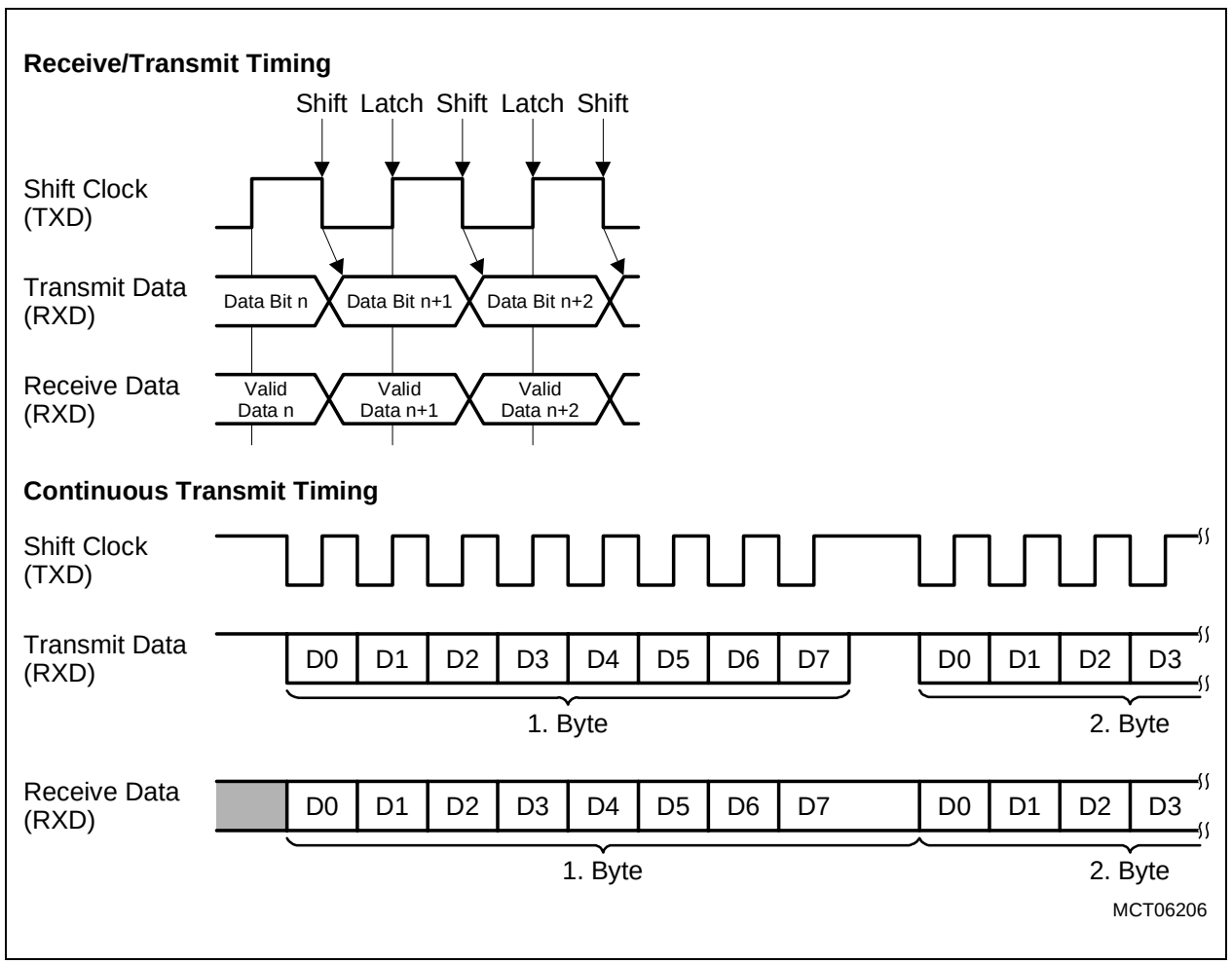


Figure 17-7 ASC Synchronous Mode Waveforms

17.1.5 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 unpredicted behavior of the ASC may occur during running transmit or receive operations.

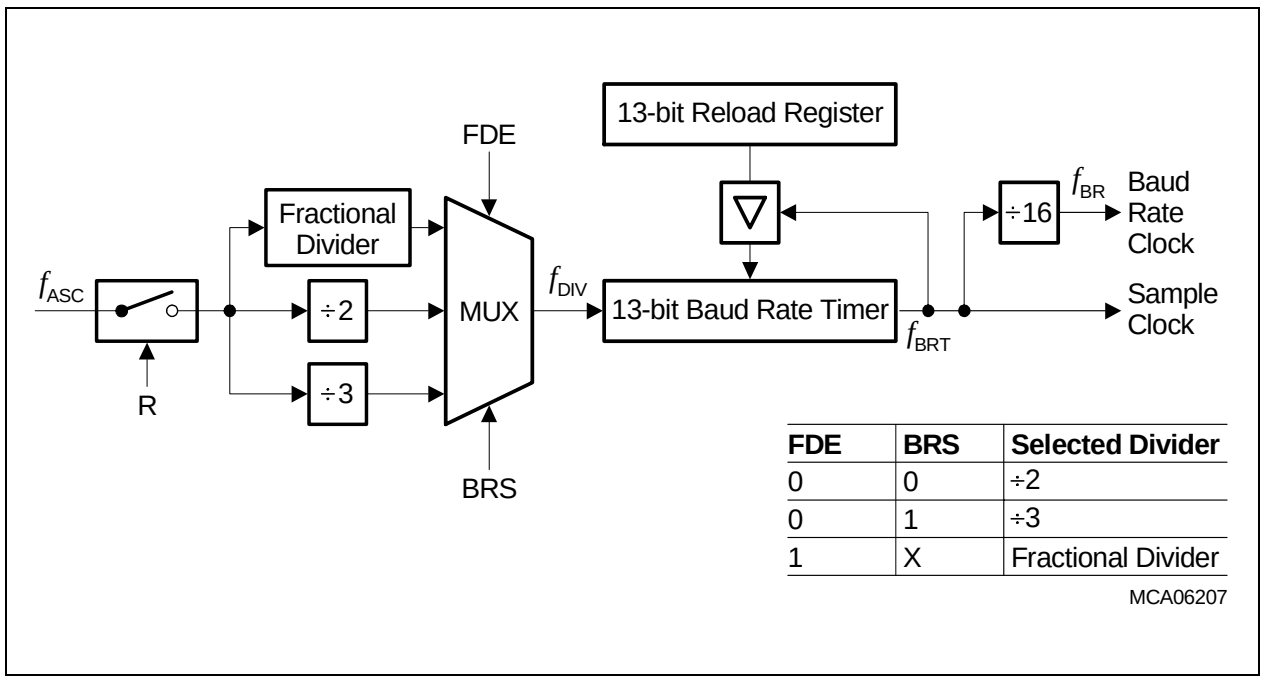
17.1.5.1 Baud Rates in Asynchronous Mode

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 17-8 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 17-1 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 80 MHz is 2.5 Mbit/s. Table 17-2 lists various commonly used baud rates together with the required reload values and the deviation errors compared to the intended baud rate.

Table 17-2 Typical Asynchronous Baud Rates using Fixed Input Clock Dividers

Baud Rate	CON.BRS = 0, f_{ASC} = 80 MHz		CON.BRS = 1, f_{ASC} = 80 MHz	
	Deviation Error	Reload Value	Deviation Error	Reload Value
2.5 Mbit/s	–	0000_H	–	–
19.2 kbit/s	+0.2% / -0.6%	0081_H / 0082_H	+0.9% / -0.2%	0055_H / 0056_H
9600 bit/s	+0.2% / -0.2%	0103_H / 0104_H	+0.4% / -0.2%	$00AC_H$ / $00AD_H$
4800 bit/s	+0.2% / -0.0%	0207_H / 0209_H	+0.1% / -0.1%	$015A_H$ / $015B_H$

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 17-3 Asynchronous Baud Rate Formulas using the Fractional Input Clock Divider

FDE	BRS	BG	FDV	Formula
1	–	0 … 8191	1 … 511	$\text{Baud rate} = \frac{\text{FDV}}{512} \times \frac{f_{ASC}}{16 \times (\text{BG} + 1)}$
			0	$\text{Baud rate} = \frac{f_{ASC}}{16 \times (\text{BG} + 1)}$

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 17-4 Typical Asynchronous Baud Rates using the Fractional Input Clock Divider

f_{ASC}	Desired Baud Rate	BG	FDV	Resulting Baud Rate	Deviation
80 MHz	115.2 kbit/s	0022_H	191_H	115.177 kbit/s	< 0.02%
	57.6 kbit/s	$004D_H$	$1CC_H$	57.592 kbit/s	< 0.01%
	38.4 kbit/s	0049_H	123_H	38.403 kbit/s	< 0.01%
	19.2 kbit/s	0075_H	$0E8_H$	19.200 kbit/s	0%

17.1.5.2 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 17-9**).

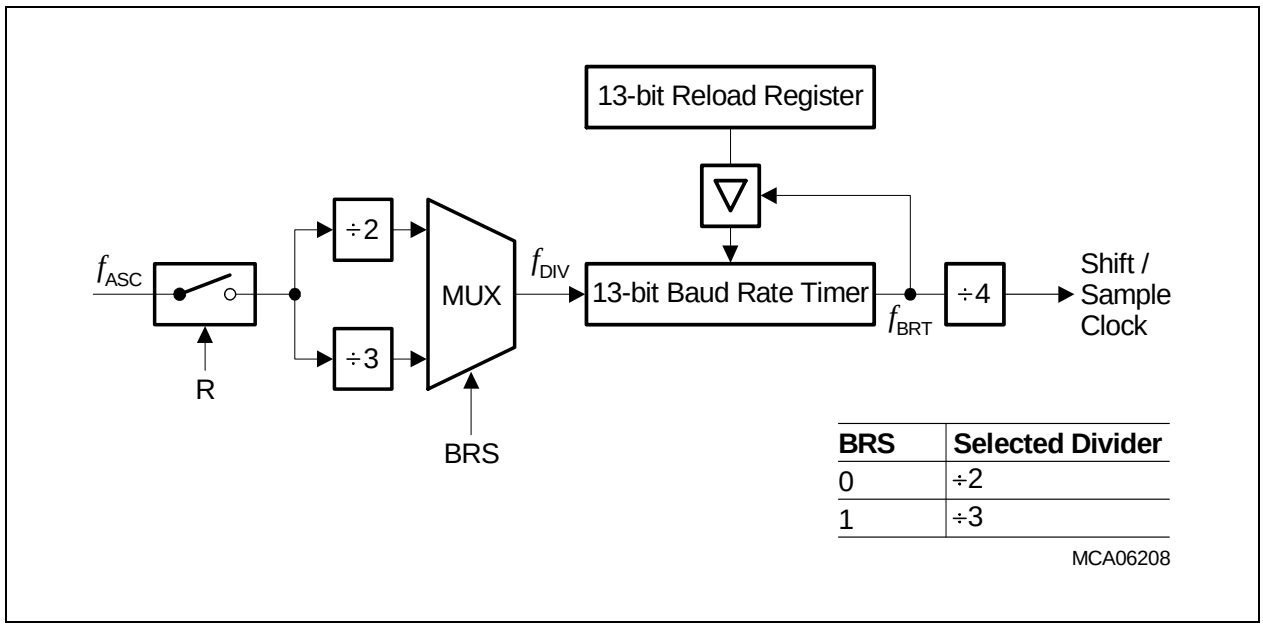


Figure 17-9 ASC Baud Rate Generator Circuitry in Synchronous Mode

The baud rate for synchronous operation of the serial channel ASC can be determined by the formulas as shown in **Table 17-5**.

Table 17-5 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 80 MHz is 10.0 Mbit/s.

17.1.6 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).

17.1.7 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 cleared 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.

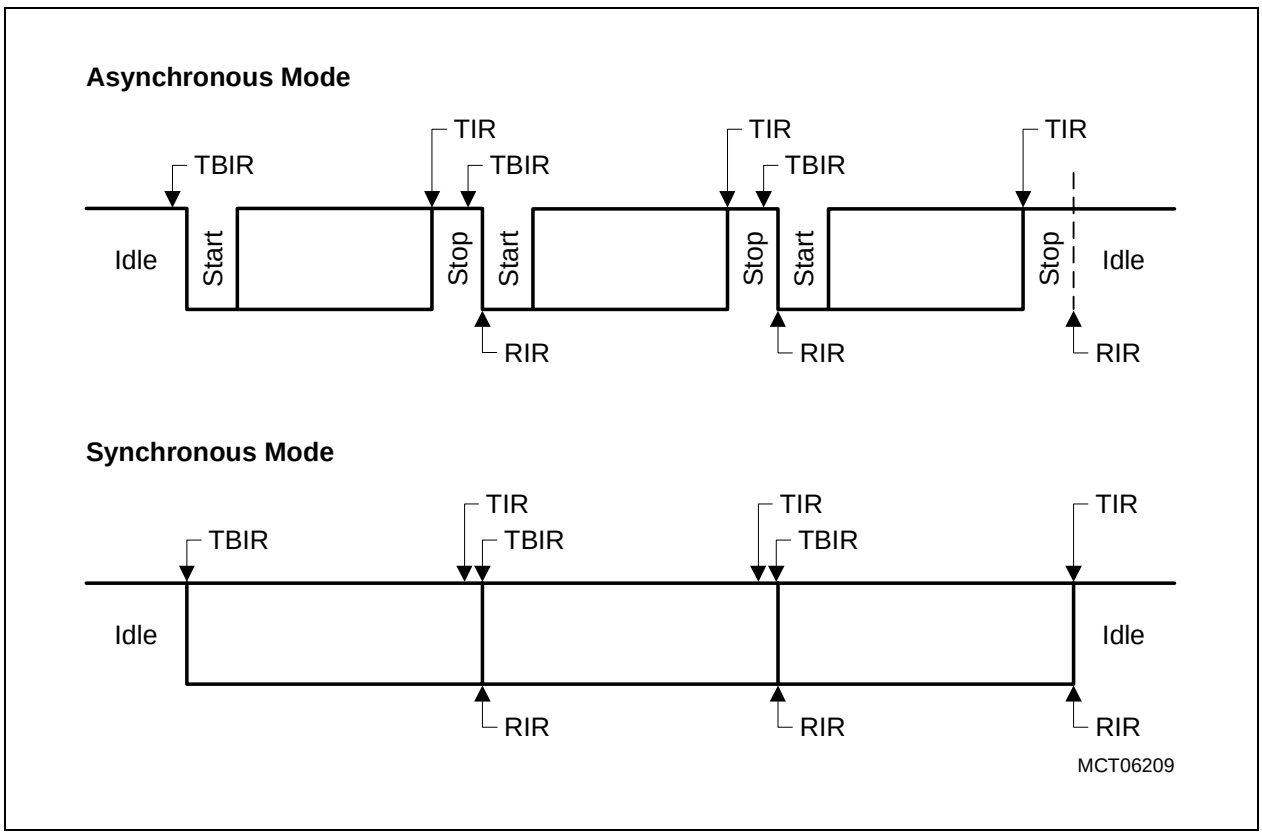


Figure 17-10 ASC Interrupt Generation

As shown in **Figure 17-10**, 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.

17.2 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 TC1766 User's Manual by the module name prefix "ASC0_" for the ASC0 interface and "ASC1_" for the ASC1 interface.

ASC Kernel Register Overview

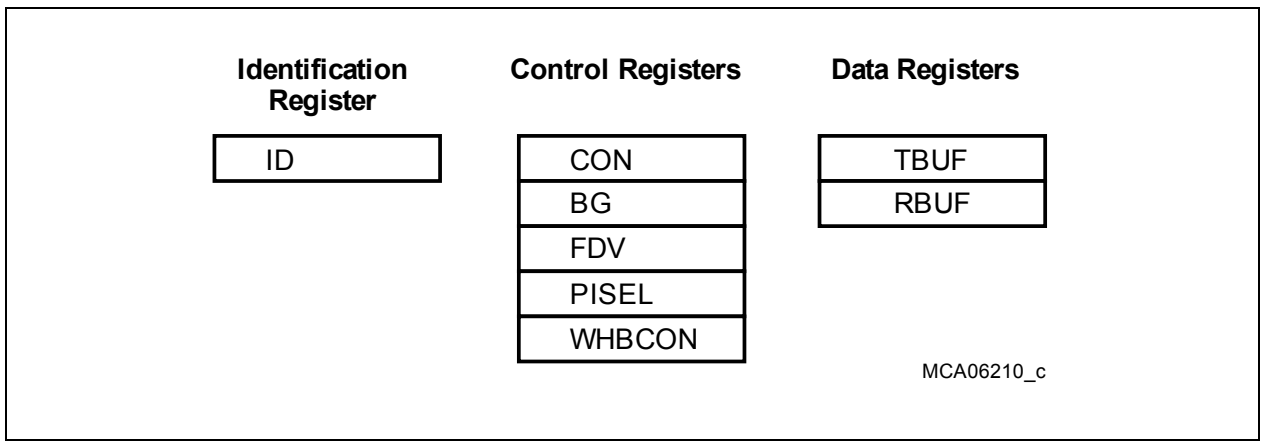


Figure 17-11 ASC Kernel Registers

The complete and detailed address map of the of the ASC modules is described in **Table 16-8** on **Page 16-17** of the TC1766 User's Manual System Units part (Volume 1).

Table 17-6 Registers Address Space

Module	Base Address	End Address	Note
ASC0	F000 0A00$_H$	F000 0AFF$_H$	-
ASC1	F000 0B00$_H$	F000 0BFF$_H$	-

Table 17-7 Registers Overview - ASC Kernel Registers

Register Short Name	Register Long Name	Offset Address[1)]	Description see
PISEL	Peripheral Input Select Register	04$_H$	**Page 17-22**
ID	Module Identification Register	08$_H$	**Page 17-21**
CON	Control Register	10$_H$	**Page 17-23**
BG	Baud Rate Timer Reload Register	14$_H$	**Page 17-28**
FDV	Fractional Divider Register	18$_H$	**Page 17-28**
TBUF	Transmit Buffer Register	20$_H$	**Page 17-29**

Table 17-7 Registers Overview - ASC Kernel Registers (cont'd)

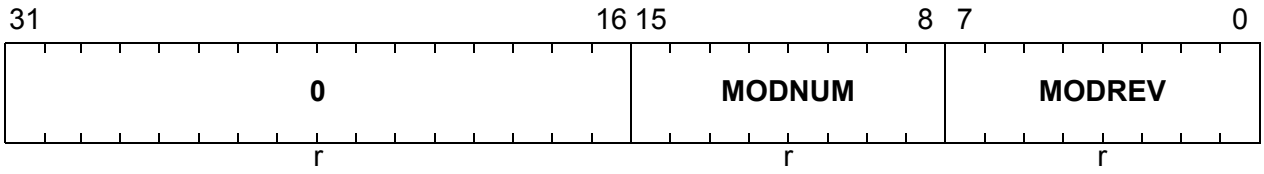
Register Short Name	Register Long Name	Offset Address[1)]	Description see
RBUF	Receive Buffer Register	24_H	Page 17-30
WHBCON	Write Hardware Bits Control Register	50_H	Page 17-26

1) The absolute register address is calculated as follows:
Module Base Address (Table 17-6) + Offset Address (shown in this column)

17.2.1 ASC Module Identification Register

The ASC Module Identification Register ID contains read-only information about the ASC module version.

ID
Module Identification Register **(08_H)** **Reset Value: 0000 44XX$_H$**

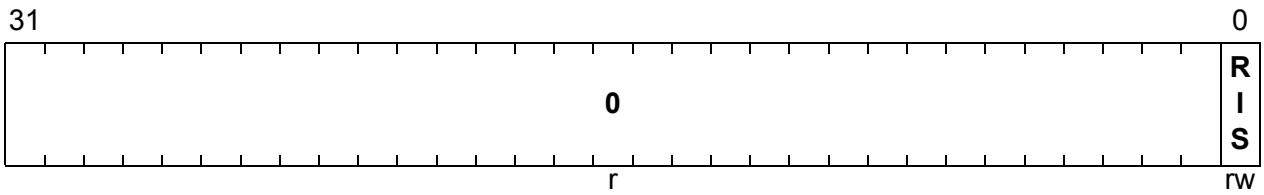


Field	Bits	Type	Description
MODREV	[7:0]	r	**Module Revision Number** MODREV defines the module revision number. The value of a module revision starts with 01_H (first revision).
MODNUM	[15:8]	r	**Module Number Value** This bit field defines the module identification number for the ASC: 44_H
0	[31:16]	r	**Reserved** Read as 0.

17.2.2 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.

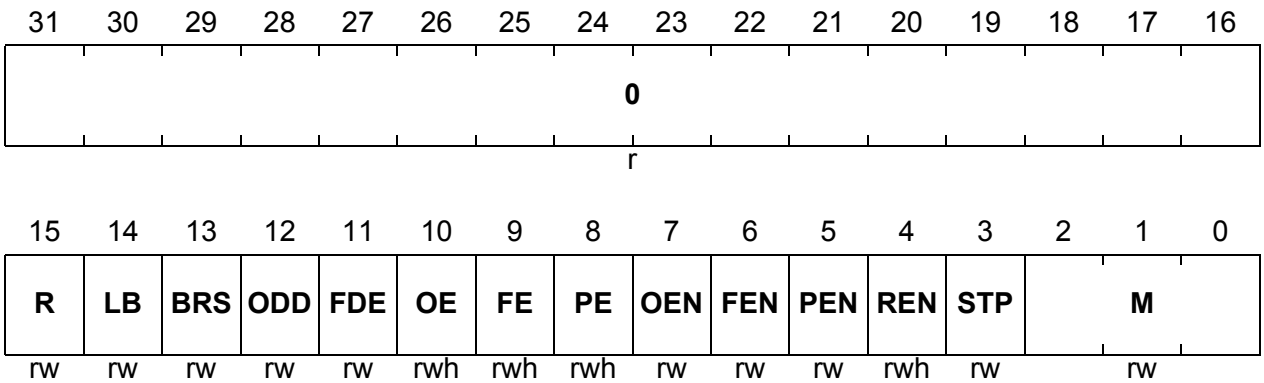
PISEL
Peripheral Input Select Register (04H) Reset Value: 0000 0000$_H$



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]	0	**Reserved** Read as 0; should be written with 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 cleared 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 cleared by software.
FE	9	rwh	**Framing Error Flag** Set by hardware on a framing error (FEN = 1). Must be cleared by software.
OE	10	rwh	**Overrun Error Flag** Set by hardware on an overrun error (OEN = 1). Must be cleared 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	**Loopback 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.

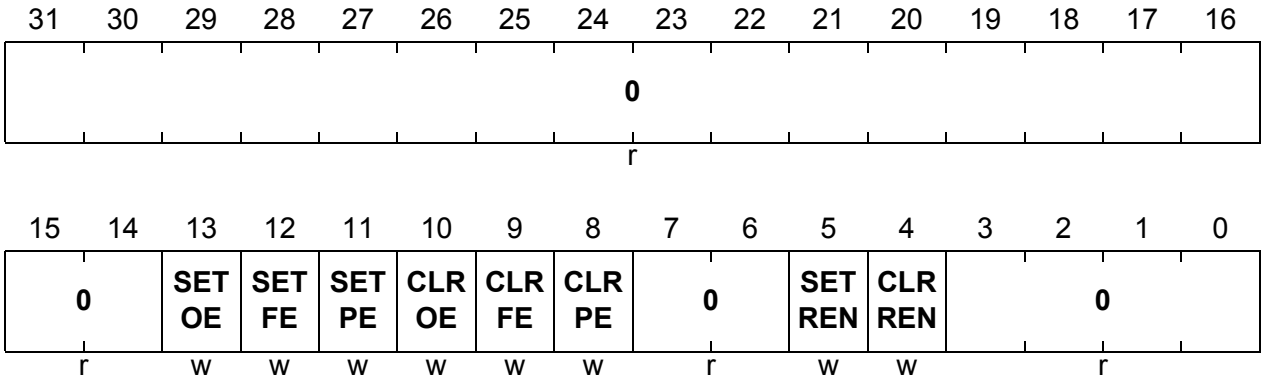
Critical “rwh” Bits

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 cleared 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: $0000\ 0000_H$**



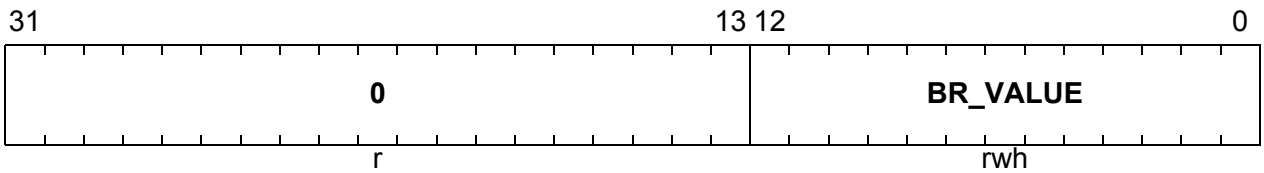
Field	Bits	Type	Description
CLRREN	4	w	**Clear Receiver Enable Bit** 0_B No effect 1_B Bit CON.REN is cleared. Bit is always read as 0.
SETREN	5	w	**Set Receiver Enable Bit** 0_B No effect 1_B Bit CON.REN is set. Bit is always read as 0.
CLRPE	8	w	**Clear Parity Error Flag** 0_B No effect 1_B Bit CON.PE 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.

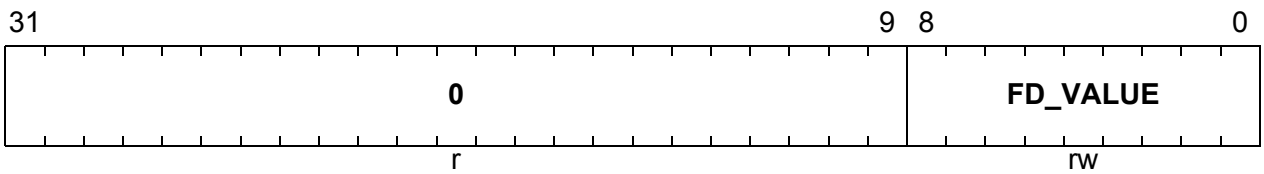
BG
Baud Rate Timer/Reload Register (14_H) Reset Value: 0000 0000$_H$



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$_H$

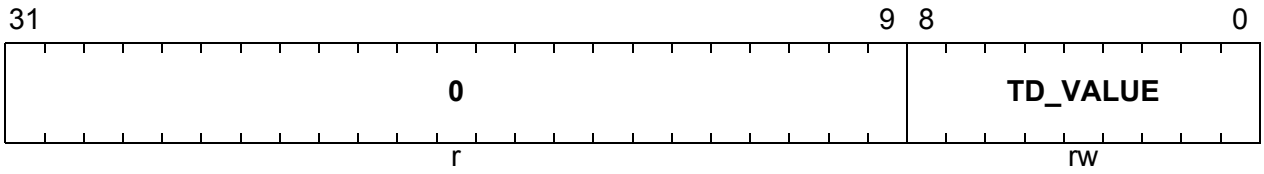


Field	Bits	Type	Description
FD_VALUE	[8:0]	rw	**Fractional Divider Register Value** FD_VALUE contains the 9-bit value n of the fractional divider which determines the fractional divider ratio n/512 (n = 0-511). With n = 0, the fractional divider is switched off (divider ratio = 1).
0	[31:9]	r	**Reserved** Read as 0; should be written with 0.

17.2.3 Data Registers

The Transmit Buffer Register TBUF of the ASC module contains the transmit data value in Asynchronous And Synchronous Modes.

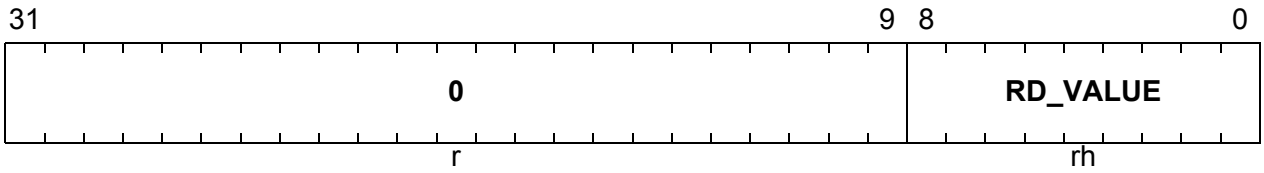
TBUF
Transmit Buffer Register **(20_H)** **Reset Value: 0000 0000_H**



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.

17.3 ASC0/ASC1 Module Implementation

This section describes ASC0/ASC1 module interfaces with the clock control, port connections, interrupt control, and address decoding.

17.3.1 Interfaces of the ASC Modules

The serial I/O lines of both modules are connected to Port 3. 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.

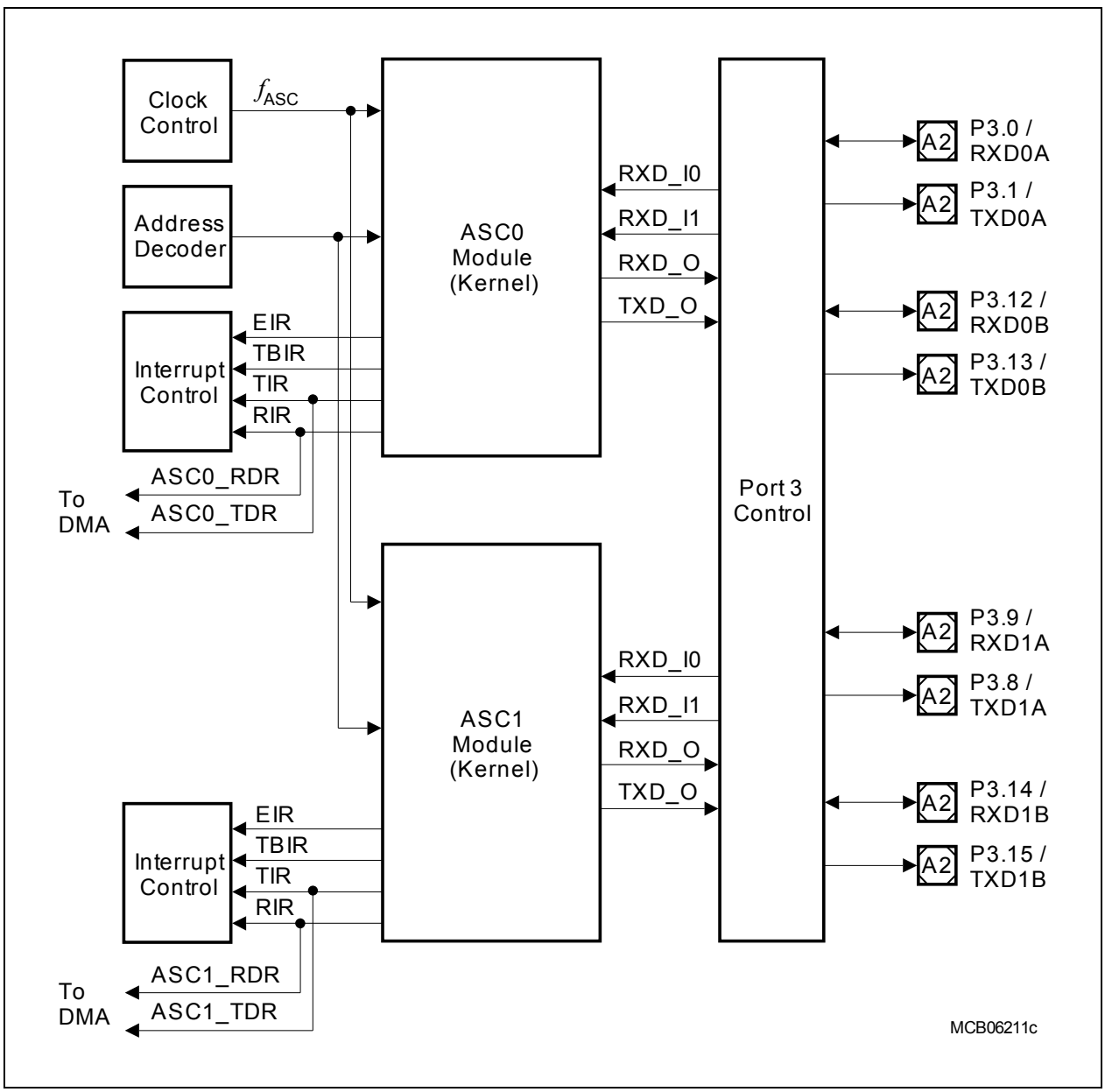


Figure 17-12 ASC0/ASC1 Module Implementation and Interconnections

17.3.2 ASC0/ASC1 Module Related External Registers

Figure 17-13 summarizes the module-related external registers which are required for ASC0/ASC1 programming (see also **Figure 17-11** for the module kernel-specific registers).

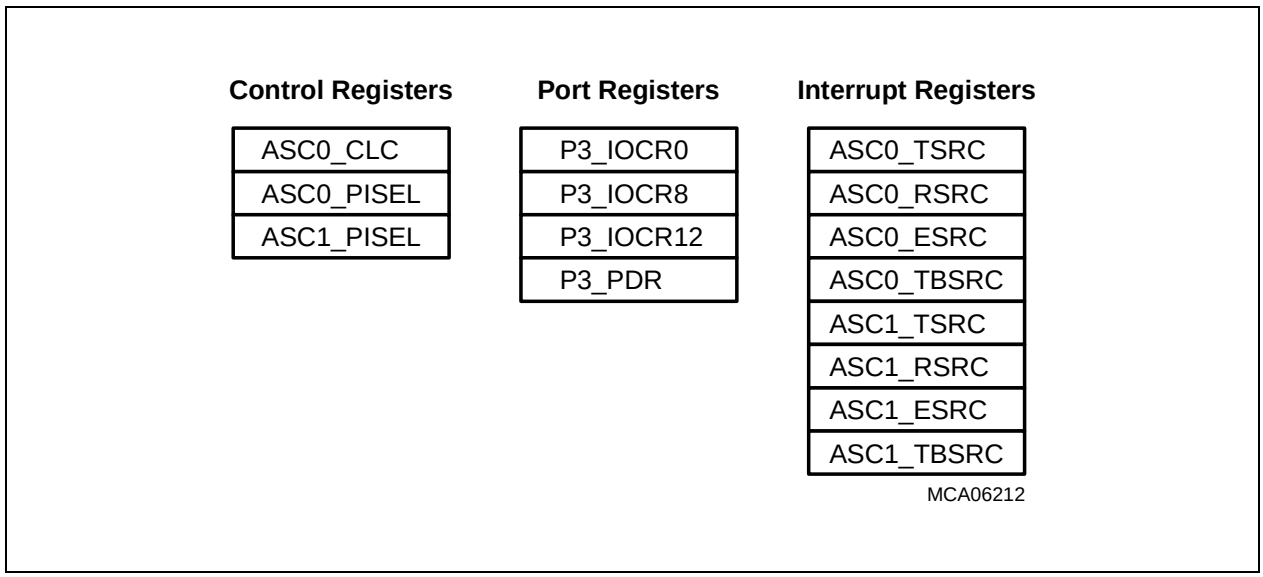
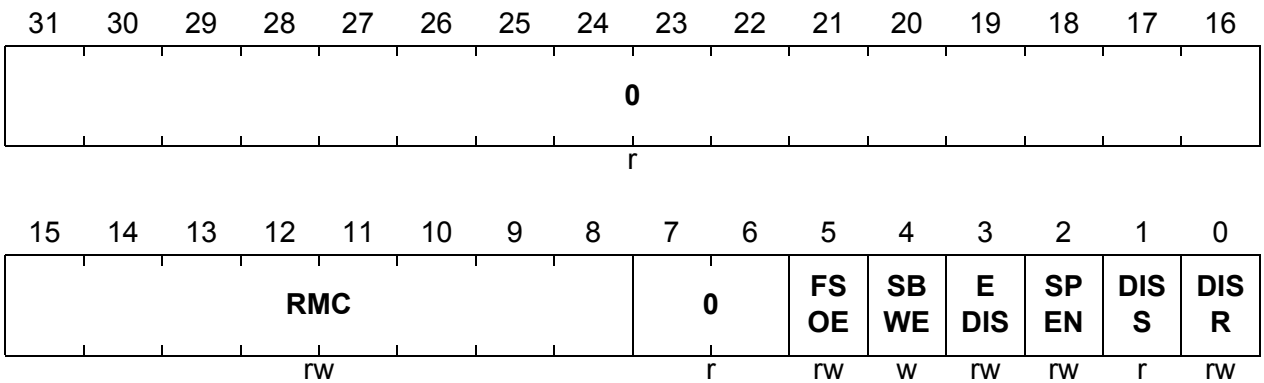


Figure 17-13 ASC0/ASC1 Implementation-specific Special Function Registers

17.3.2.1 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/cleared 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.

*Note: Additional details on the clock control register functionality are described in section **"Clock Control Register CLC" on Page 3-24** of the TC1766 User's Manual System Units part (Volume 1).*

17.3.2.2 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 pair of pins of Port 3 as shown in **Figure 17-14**. Register ASC0_PISEL controls the RXD input selection for ASC0, and ASC1_PISEL controls the RXD input selection for ASC1.

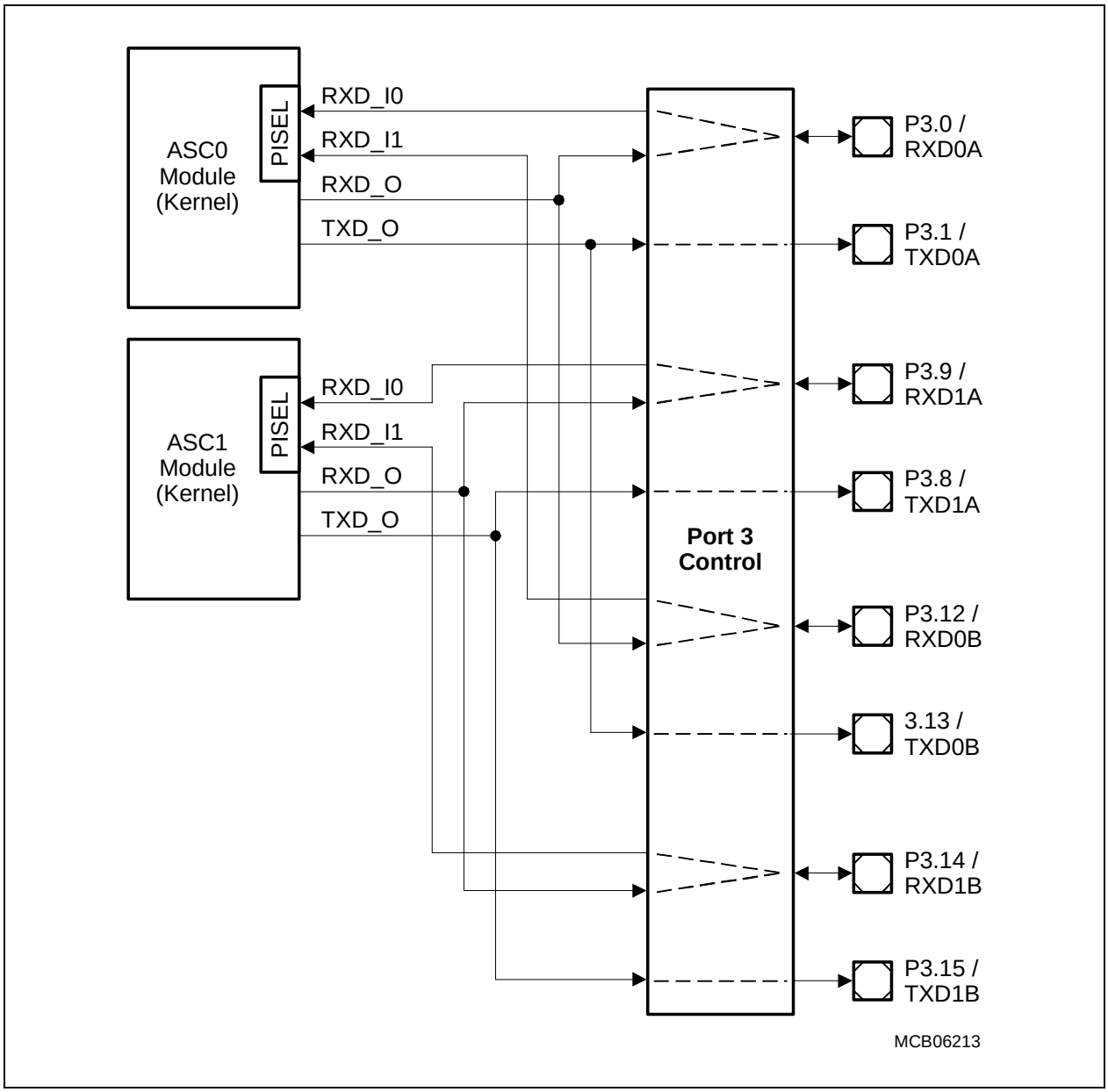
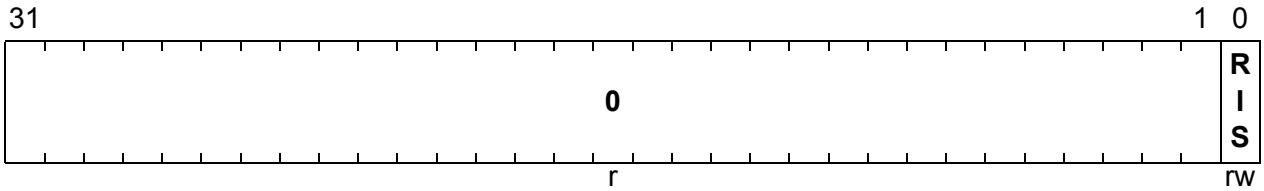


Figure 17-14 RXD Input Line Selection of the ASC Modules

ASC0_PISEL
ASC0 Peripheral Input Select Register

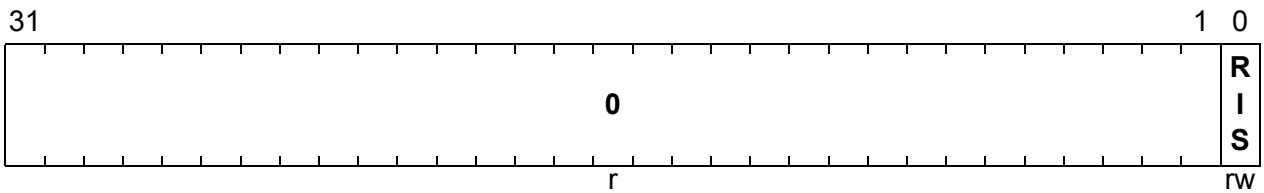
(04_H) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
RIS	0	rw	**Receive Input Select** 0_B ASC0 receiver input RXD0A (P3.0) selected 1_B ASC0 receiver input RXD0B (P3.12) 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 (P3.9) selected 1_B ASC1 receiver input RXD1B (P3.14) selected
0	[31:1]	0	**Reserved** Read as 0; should be written with 0.

17.3.2.3 Port Control Registers

As shown in **Figure 17-14**, 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 P3_IOCR0, P3_IOCR8 and P3_IOCR12.

Table 17-8 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 17-8 ASC0/ASC1 I/O Control Selection and Setup

Module	Port Lines	PISEL Register	Input/Output Control Register Bits[1)]	I/O
ASC0	P3.0/RXD0A (Async./Sync. Mode)	ASC0_PISEL.RIS = 0	P3_IOCR0.PC0 = $0XXX_B$	Input
	P3.0/RXD0A (Sync. Mode)	–	P3_IOCR0.PC0 = $1X01_B$ or $1X10_B$	Output [2)]
	P3.12/RXD0B (Async./Sync. Mode)	ASC0_PISEL.RIS = 1	P3_IOCR12.PC12 = $0XXX_B$	Input
	P3.12/RXD0B (Sync. Mode)	–	P3_IOCR12.PC12 = $1X01_B$ or $1X10_B$	Output [2)]
	P3.1/TXD0A	–	P3_IOCR0.PC1 = $1X01_B$or $1X10_B$	Output
	P3.13/TXD0B (Sync. Mode)	–	P3_IOCR12.PC13 = $1X10_B$	Output

Table 17-8 ASC0/ASC1 I/O Control Selection and Setup (cont'd)

Module	Port Lines	PISEL Register	Input/Output Control Register Bits[1)]	I/O
ASC1	P3.9/RXD1A (Async./Sync. Mode)	ASC1_PISEL.RIS = 0	P3_IOCR8.PC9 = $0XXX_B$	Input
	P3.9/RXD1A (Sync. Mode)	–	P3_IOCR8.PC9 = $1X01_B$ or $1X10_B$	Output [2)]
	P3.14/RXD1B (Async./Sync. Mode)	ASC1_PISEL.RIS = 1	P3_IOCR12.PC14 = $0XXX_B$	Input
	P3.14/RXD1B (Sync. Mode)	–	P3_IOCR12.PC14 = $1X01_B$ or $1X10_B$	Output [2)]
	P3.8/TXD1A	–	P3_IOCR8.PC8 = $1X10_B$	Output
	P3.15/TXD1B (Sync. Mode)	–	P3_IOCR12.PC15 = $1X10_B$	Output

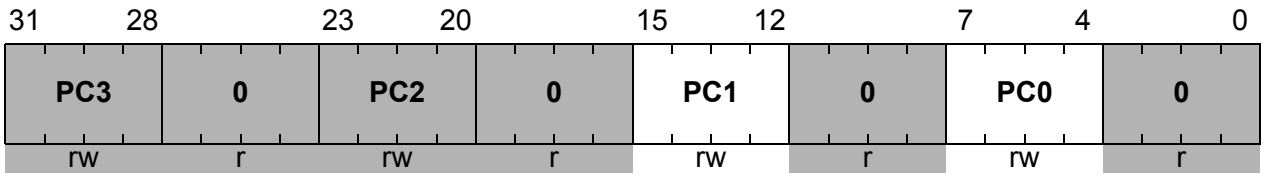
1) For possible PCx bit field combinations, see **Table 17-9**.

2) 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.*

Input/Output Control Registers

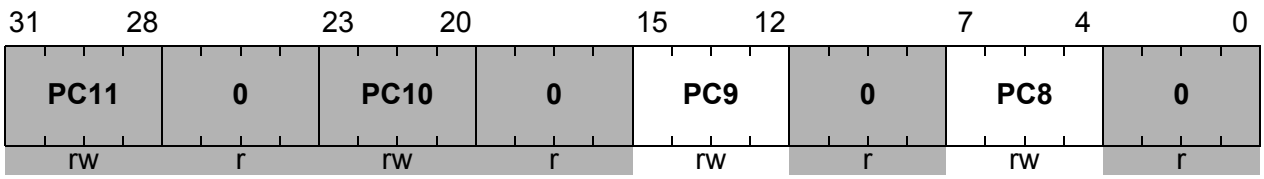
P3_IOCR0
Port 3 Input/Output Control Register 0 **Reset Value: 2020 2020$_H$**



Field	Bits	Type	Description
		rw	**Port Output Control for Port 3.0 and Port 3.1[1)]** These bit fields determine the output port functionality:
PC0	[7:4],		Port input/output control for P3.0/RXD0A
PC1	[15:12]		Port input/output control for P3.1/TXD0A

1) For coding of bit field, see **Table 17-9**. Shaded bits and bit fields are “don’t care” for ASC I/O port control.

P3_IOCR8
Port 3 Input/Output Control Register 8 **Reset Value: 2020 2020$_H$**

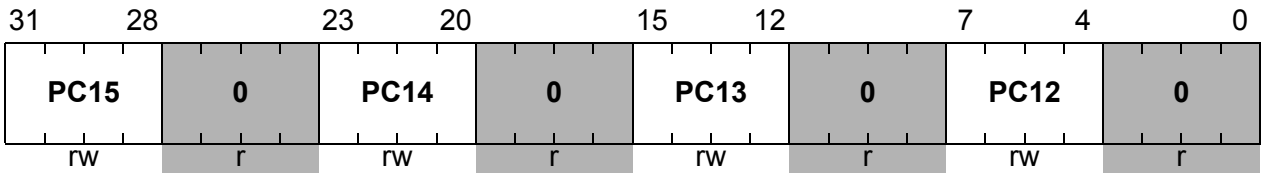


Field	Bits	Type	Description
		rw	**Port Output Control for Port 3.8 and Port 3.9[1)]** These bit fields determine the output port functionality:
PC8	[7:4],		Port input/output control for P3.8/TXD1A
PC9	[15:12]		Port input/output control for P3.9/RXD1A

1) For coding of bit field, see **Table 17-9**. Shaded bits and bit fields are “don’t care” for ASC I/O port control.

P3_IOCR12
Port 3 Input/Output Control Register 12

Reset Value: 2020 2020$_H$



Field	Bits	Type	Description
		rw	**Port Output Control for Port 3.[15:12]**[1)] These bit fields determine the output port functionality:
PC12	[7:4]		Port input/output control for P3.12/RXD0B
PC13	[15:12]		Port input/output control for P3.13/TXD0B
PC14	[23:20]		Port input/output control for P3.14/RXD1B
PC15	[31:28]		Port input/output control for P3.15/TXD1B

1) For coding of bit field, see **Table 17-9**. Shaded bits and bit fields are “don’t care” for ASC I/O port control.

PCx Coding Table

Table 17-9 PCx Coding

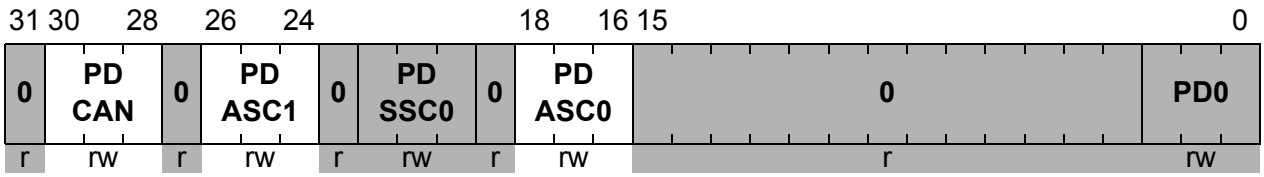
PCx[3:0]	I/O	Output Characteristics	Selected Pull-up/Pull-down/ Selected Output Function
$0X00_B$	Input	–	No pull device connected
$0X01_B$			Pull-down device connected
$0X10_B$[1)]			Pull-up device connected
$0X11_B$			No pull device connected
1001_B	Output	Push-pull	Output function ALT1
1101_B		Open-drain	Output function ALT1
1010_B		Push-pull	Output function ALT2
1110_B		Open-drain	Output function ALT2
1011_B		Push-pull	Output function ALT3
1111_B		Open-drain	Output function ALT3

1) This bit field value is default after reset.

Pad Output Driver Characteristics Selection

The Port 3 Pad Driver Mode Register contains bit fields that determine the output driver's strength and the slew rate of ASC output lines. The coding of the PDx bit field combinations is shown **Table 17-10**.

P3_PDR
Port 3 Pad Driver Mode Register **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
PDASC0	[18:16]	rw	**Pad Driver Mode for P3.0/RXD0A and P3.1/TXD0A[1)]**
PDASC1	[26:24]	rw	**Pad Driver Mode for P3.9/RXD1A and P3.8/TXD1A[1)]**
PDCAN	[30:28]	rw	**Pad Driver Mode for P3.12/RXD0B, P3.13/TXD0B, P3.14/RXD1B, and P3.15/TXD1B[1)]**

1) For coding of bit fields, see **Table 17-10**. Shaded bits and bit fields are "don't care" for ASC I/O port control.

PDx Selection Table

Table 17-10 Pad Output Driver Characteristic Selection (Class A2 pads)

PDx Bit Field	Driver Strength	Signal Transitions
000_B	Strong driver	Sharp edge[1)]
001_B		Medium edge[1)]
010_B		Soft edge[1)]
011_B	Weak driver	–
100_B	Medium driver	–
101_B		
110_B		
111_B	Weak driver	–

1) In strong driver mode, the output driver characteristics of class A2 pads can be additionally controlled by the temperature compensation logic.

17.3.2.4 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$_H$**

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**

Field	Bits	Type	Description
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 on **Page 12-3** of the TC1766 User's Manual System Units part (Volume 1).*

17.3.3 DMA Requests

The DMA request output lines of the ASC0/ASC1 modules become active whenever the related interrupt line is activated. The DMA request lines are connected to the DMA controller as shown in **Table 17-11**.

Table 17-11 DMA Request Lines of ASC0/ASC1

Module	Related ASC Interrupt	DMA Request Line	Description
ASC0	RIR	ASC0_RDR	ASC0 Receive DMA Request
	TIR	ASC0_TDR	ASC0 Transmit DMA Request
ASC1	RIR	ASC1_RDR	ASC1 Receive DMA Request
	TIR	ASC1_TDR	ASC1 Transmit DMA Request

Note: Further details on DMA handling and processing are described in the chapter "DMA Controller" of the TC1766 System Units User's Manual.

18 Synchronous Serial Interface (SSC)

This chapter describes the two SSC Synchronous Serial Interfaces, SSC0 and SSC1, of the TC1766. It contains the following sections:

- Functional description of the SSC kernel, valid for SSC0 and SSC1 (see **Page 18-1**).
- SSC kernel register description, describes all SSC kernel specific registers (see **Page 18-22**).
- TC1766 implementation-specific details and registers of the SSC0/SSC1 modules (port connections and control, interrupt control, address decoding, clock control, see **Page 18-35**).

*Note: The SSC kernel register names described in **Section 18.2** are referenced in the TC1766 User's Manual by the module name prefix "SSC0_" for the SSC0 interface and by "SSC1_" for the SSC1 interface.*

18.1 SSC Kernel Description

Figure 18-1 shows a global view of the SSC interface.

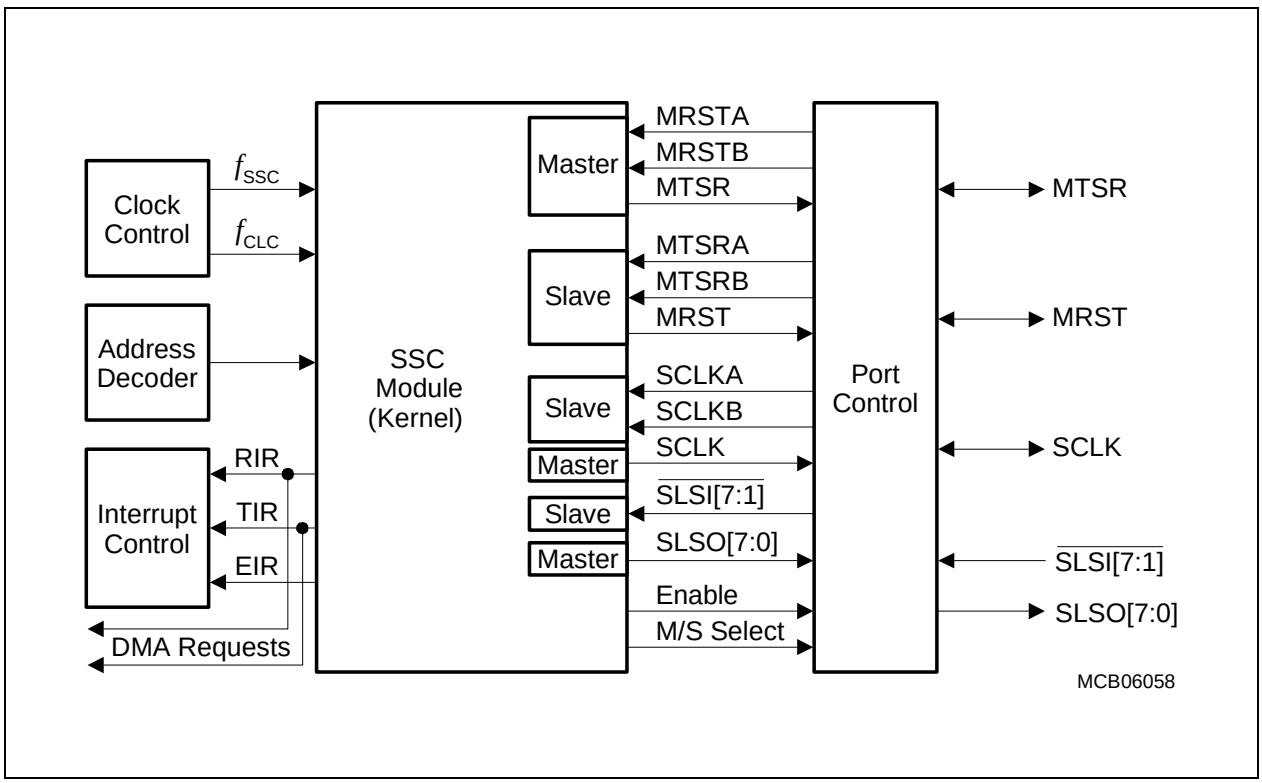


Figure 18-1 General Block Diagram of the SSC Interface

18.1.1 Overview

The SSC supports full-duplex and half-duplex serial synchronous communication up to 40.0 Mbit/s (@ 80 MHz module clock). 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 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 from 40.0 Mbit/s to 610.36 bit/s (@ 80 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

18.1.2 General Operation

The SSC supports full-duplex and half-duplex synchronous communication up to 40.0 Mbit/s (@ 80 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.

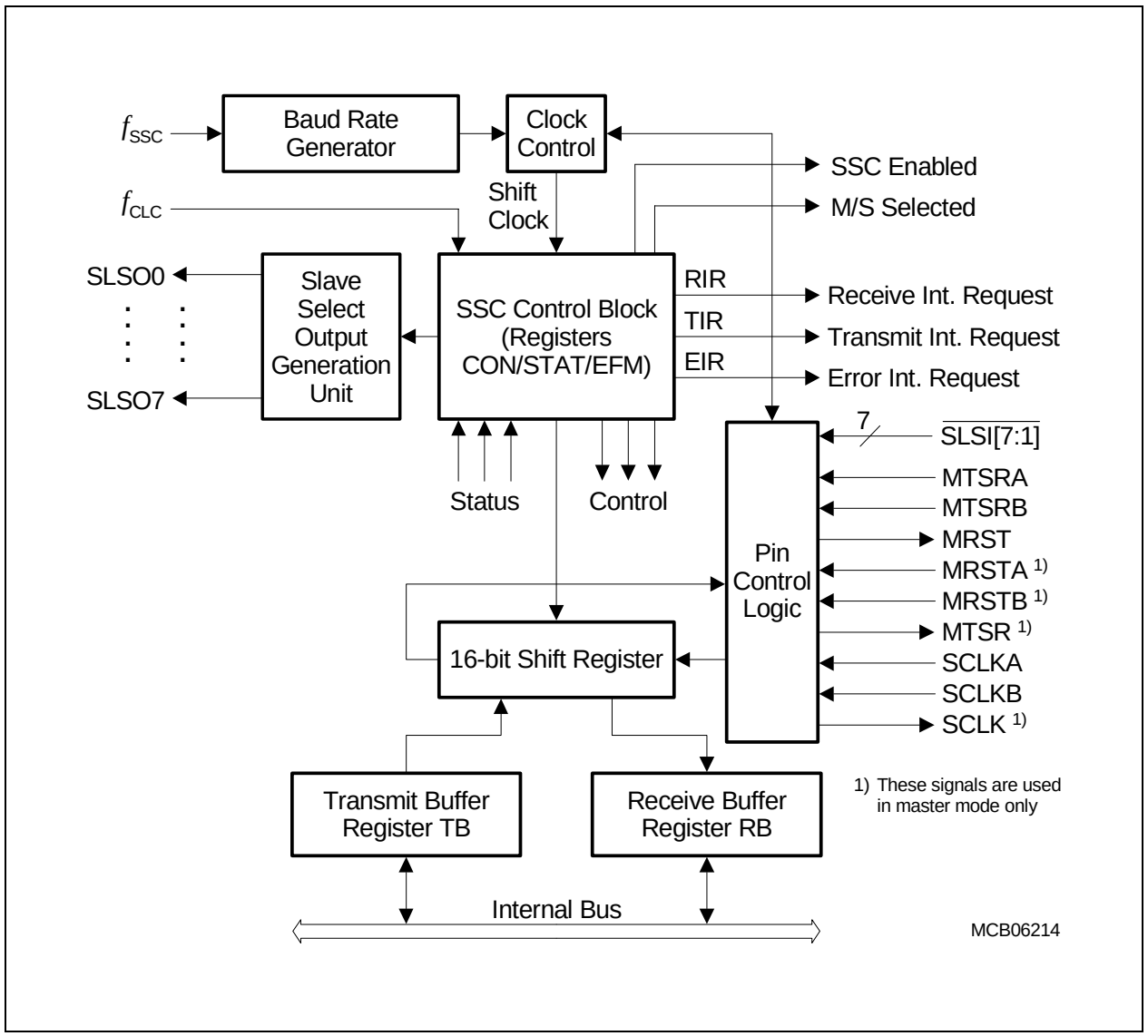


Figure 18-2 Synchronous Serial Channel SSC Block Diagram

18.1.2.1 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 18-2**). 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 (2 to 16, as programmed) have been transferred, the contents of the shift register are moved to the Receive Buffer Register (RB), 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 bits to 16 bits
- A transfer may start with the LSB or the MSB
- 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 610.36 bit/s up to 40.0 Mbit/s (@ 80 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 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 data width selected 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 18-3**).

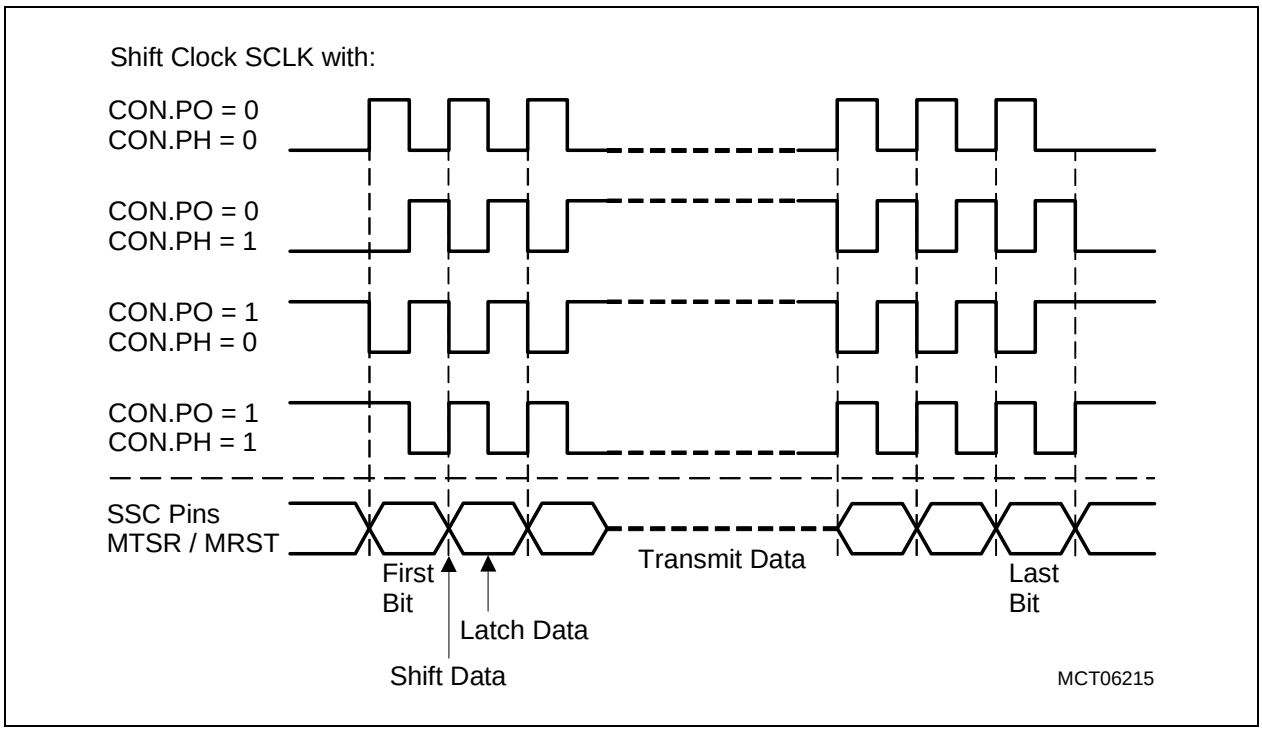


Figure 18-3 Serial Clock SCLK Phase and Polarity Options

18.1.2.2 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 18.1.2.5**).

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 18-4** 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.

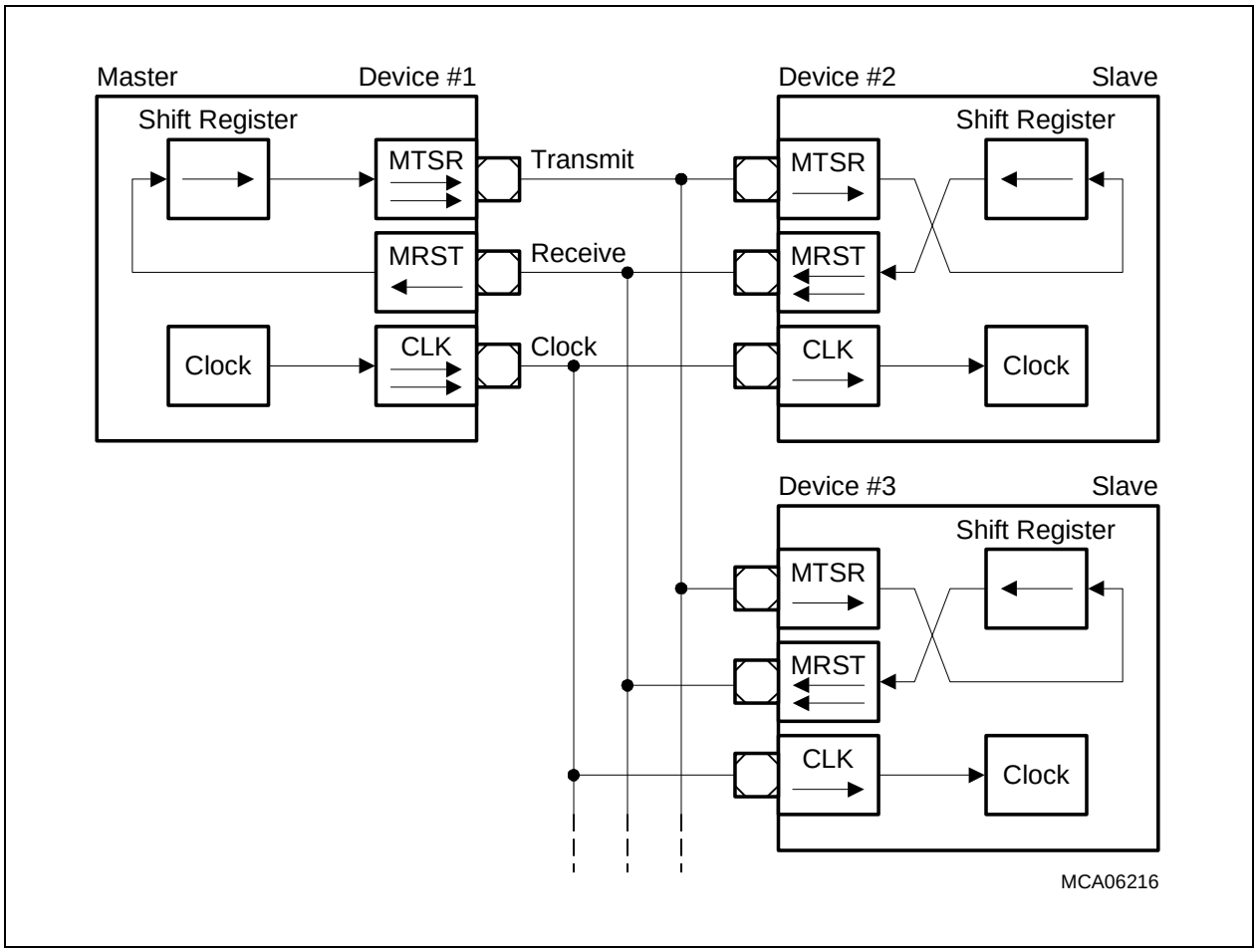


Figure 18-4 SSC Full-Duplex Configuration

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.*

18.1.2.3 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 18.1.2.5**).

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.

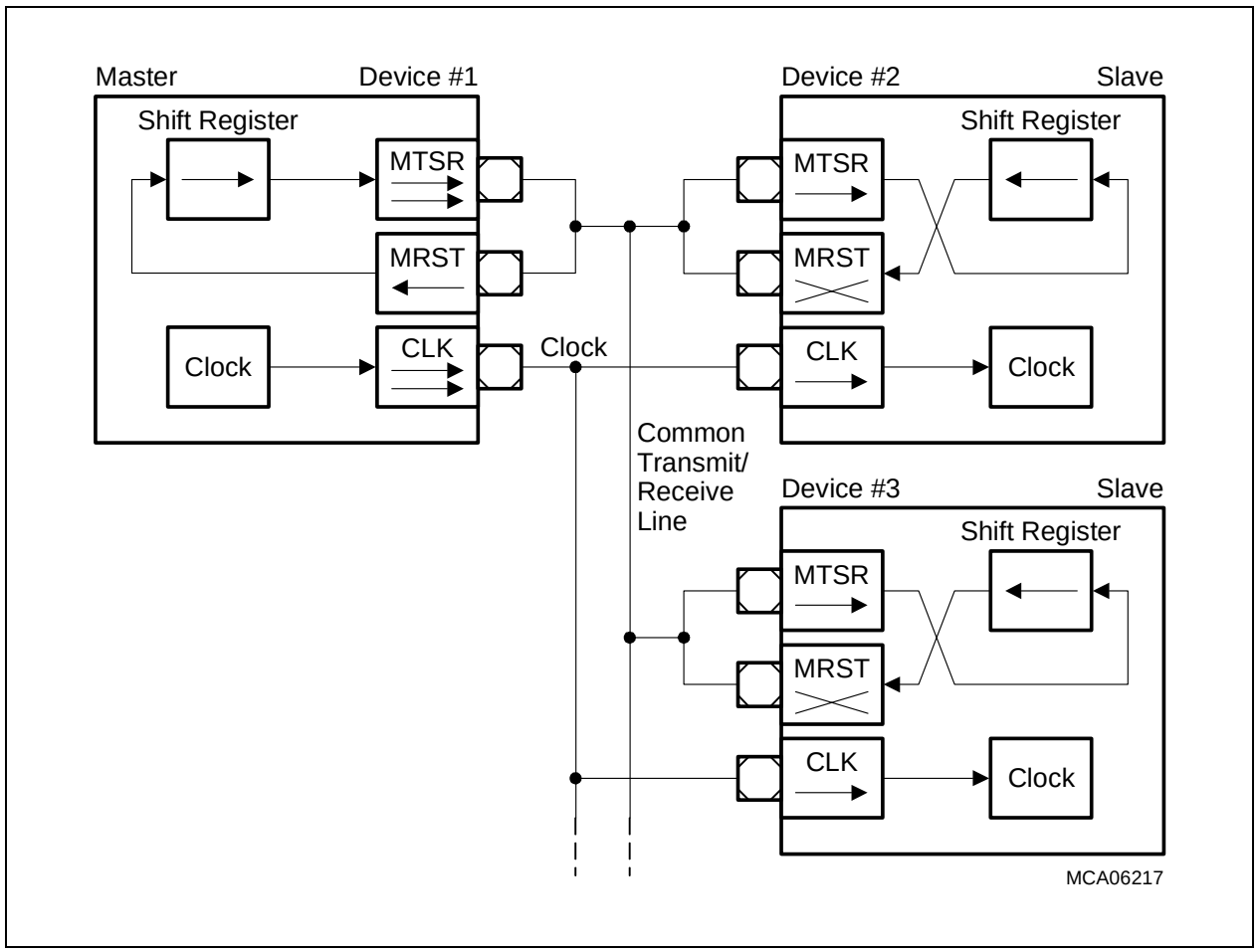


Figure 18-5 SSC Half-Duplex Configuration

18.1.2.4 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.

18.1.2.5 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 18-1** 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 **Section 18.3.1**.*

18.1.2.6 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 18-2**, **Figure 18-6** shows the baud rate generator of the SSC in more detail.

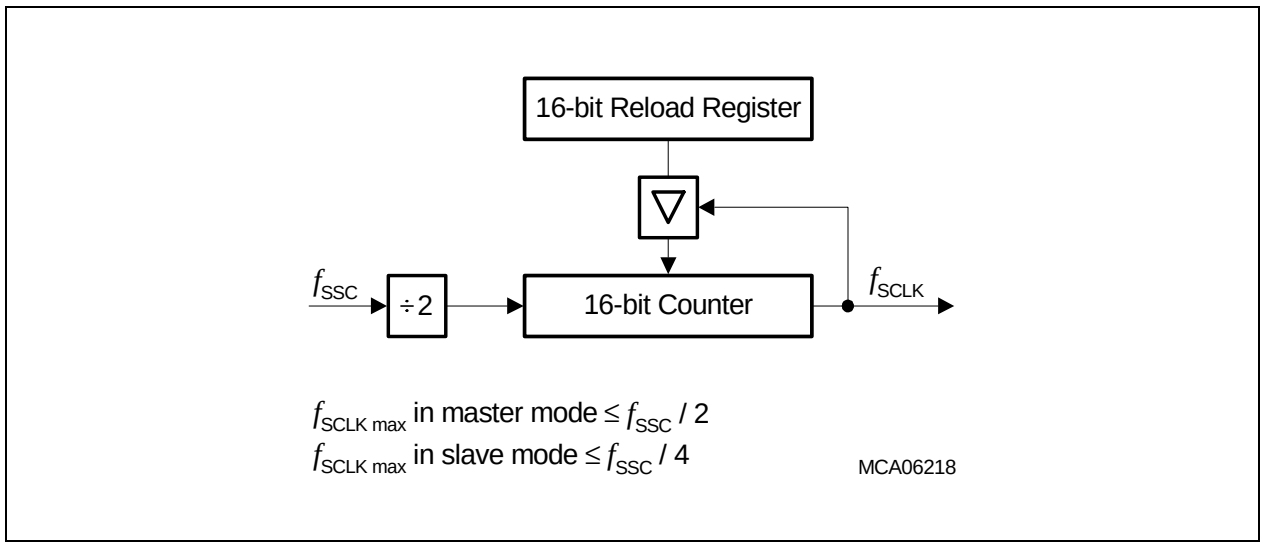


Figure 18-6 SSC Baud Rate Generator

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:

$$\text{Baud rate}_{SSC} = \frac{f_{SSC}}{2 \times (\text{BR\_VALUE} + 1)} \qquad \text{BR\_VALUE} = \frac{f_{SSC}}{2 \times \text{Baud rate}_{SSC}} - 1 \qquad (18.1)$$

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 18-6**.

The maximum baud rate that can be achieved with f_{SSC} = 80 MHz is 40.0 Mbit/s in Master Mode (with BR_VALUE = 0000_H) and 20.0 Mbit/s in Slave Mode (with BR_VALUE = 0001_H).

Table 18-1 lists some possible baud rates together with the required reload values and the resulting bit times, assuming a module clock f_{SSC} of 80 MHz and 40 MHz.

Table 18-1 Typical Baud Rates of the SSC (f_{SSC} = 80 MHz)

Reload Value	Baud Rate (= f_{SCLK})	Deviation
0000_H	40 Mbit/s (only in Master Mode)	0.0%
0001_H	20 Mbit/s	0.0%
0003_H	10 Mbit/s	0.0%
0027_H	1 Mbit/s	0.0%
0063_H	400 kbit/s	0.0%
$018F_H$	100 kbit/s	0.0%
$0F9F_H$	10 kbit/s	0.0%
$9C3F_H$	1 kbit/s	0.0%
$FFFF_H$	610.36 bit/s	0.0%

In the TC1766, 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 18.3.2.1** on **Page 18-38** describes these dependencies in detail.

18.1.2.7 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 18-7** 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.

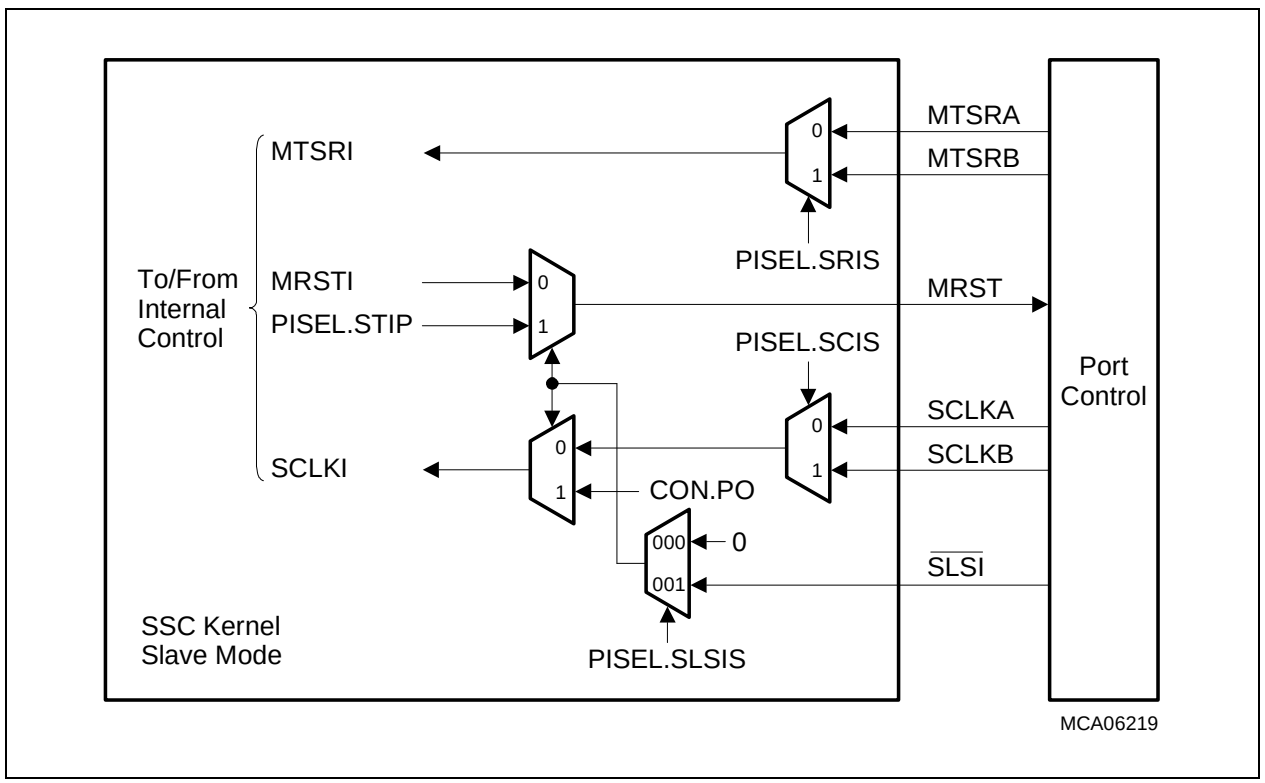


Figure 18-7 Slave Select Input Logic

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).
- $\overline{SLSI}$ = 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).

18.1.2.8 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 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 18-9**). The basic slave select output timing is shown in **Figure 18-8**, assuming a low active level of the SLSOn lines.

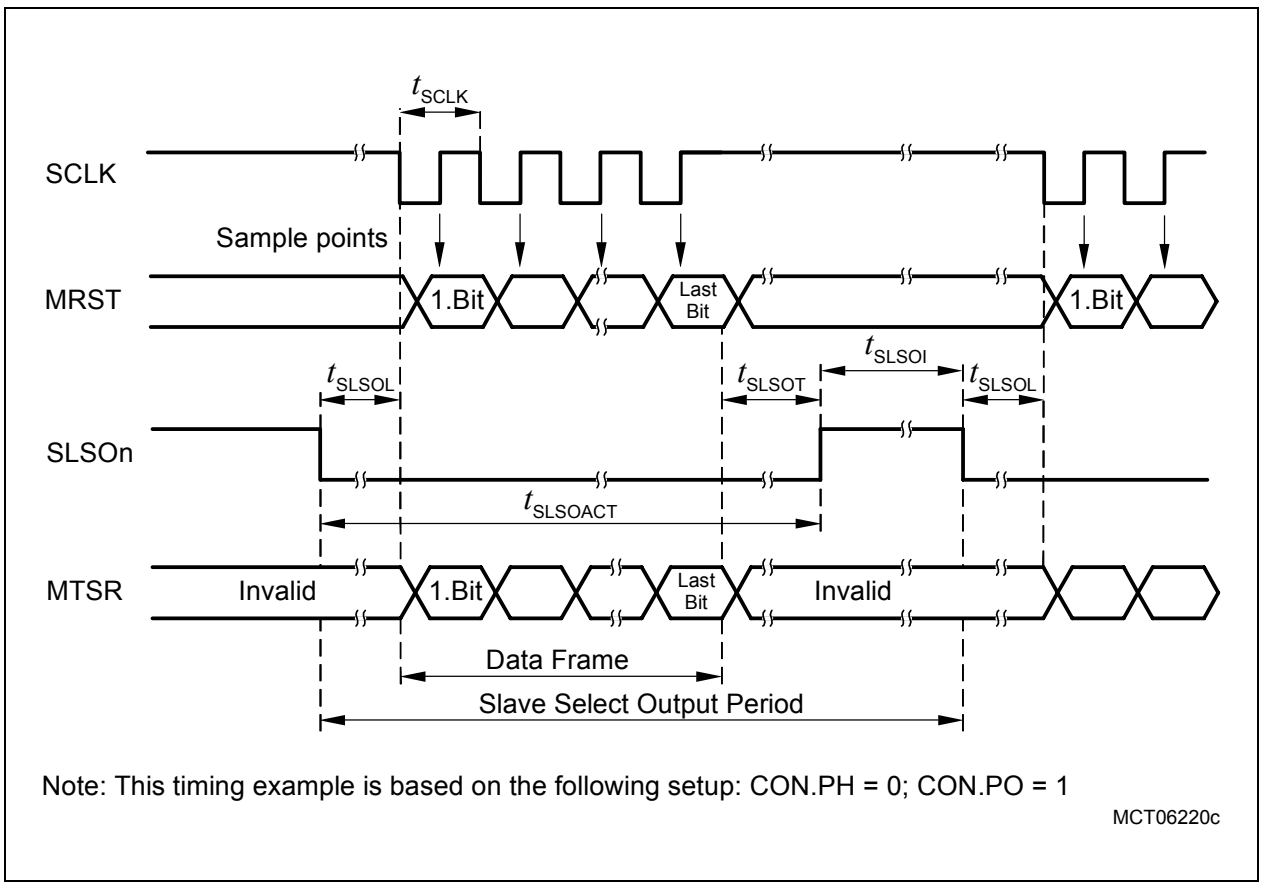


Figure 18-8 SSC Slave Select Output Timing

A slave select output period always starts after a serial write operation to register TB. With a TB write operation, all timing parameters stored in register SSOTC as well as the SSOC register 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, SLSOx 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 SLSOx 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} = SSSOTC.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, SLSOx 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.

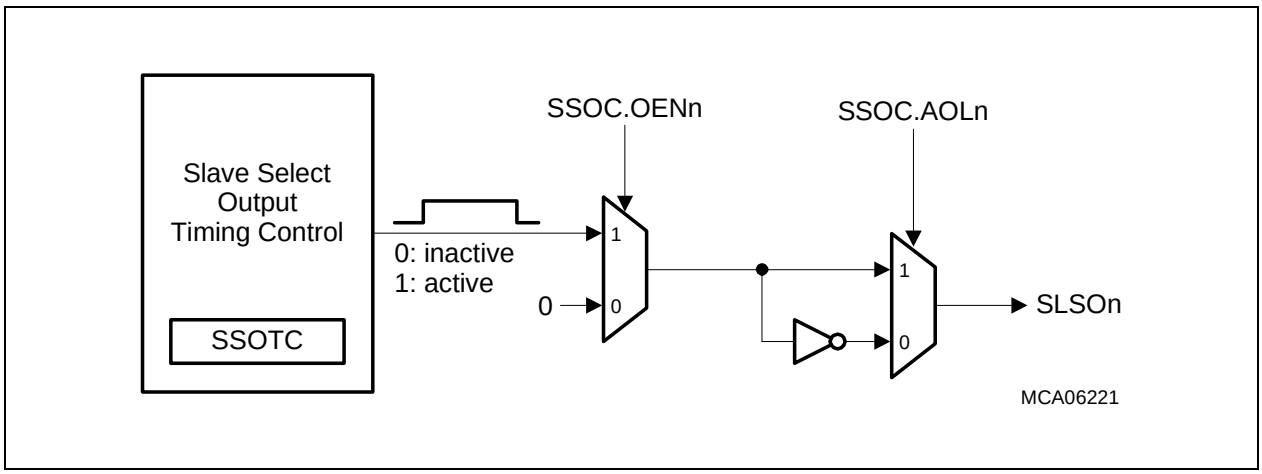


Figure 18-9 Slave Select Output Control Logic

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 18-10**. The bold lines show the timing of SLSO7 in normal operating mode, and the dotted lines show the timing of SLSO7 in delayed mode.

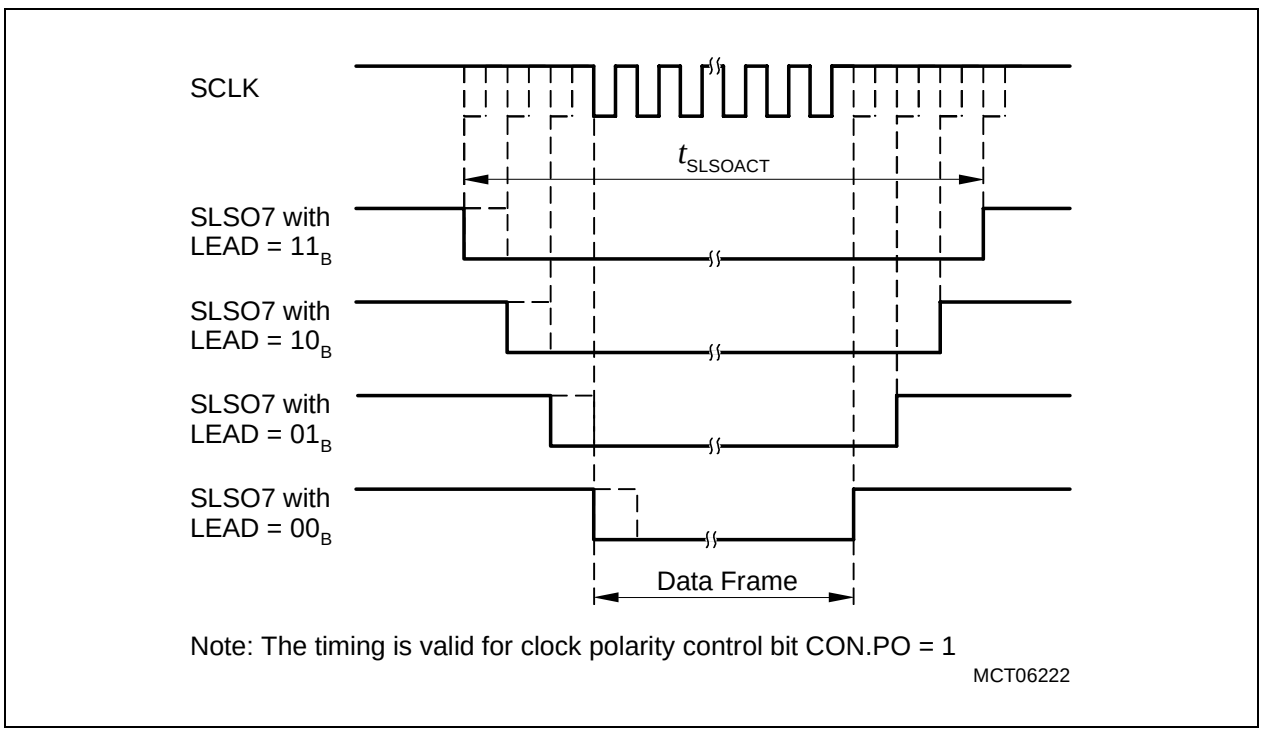


Figure 18-10 SLSO7 Delayed Mode

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 buffered. 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.

18.1.2.9 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. When an error is detected, the respective error flag is always set and the error interrupt request will be generated by activating the EIR line if the corresponding error enable bit is set (see **Figure 18-11**). 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.

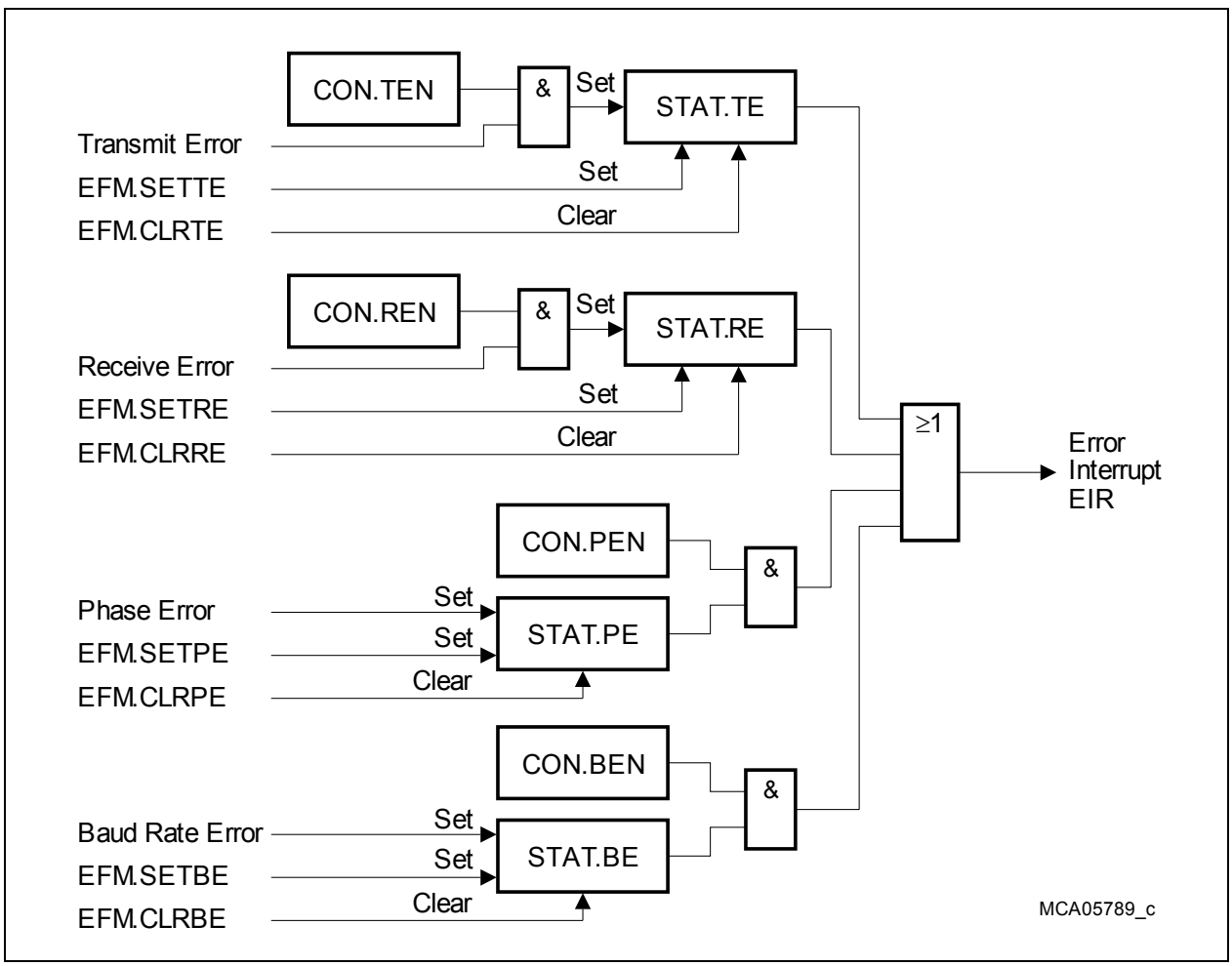


Figure 18-11 SSC Error Interrupt Control

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. 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.

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: This error can occur after any transfer if the communication is stopped. This is due to the fact that SSC supports back-to-back transfers for multiple transfers. In order to handle this, the baud rate detection logic expects a next clock cycle immediately for a new transfer after a finished 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, the slave SSC expects continuation of the clock pulses for the next data frame transmission immediately after finishing the current data frame. Therefore, if the master (shift) clock is not continued, 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.

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. 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 cleared automatically upon entry into the error interrupt service routine, but must be cleared by software.

18.2 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 TC1766 User's Manual by the module name prefix "SSC0_" for the SSC0 interface and "SSC1_" for the SSC1 interface.

SSC Kernel Register Overview

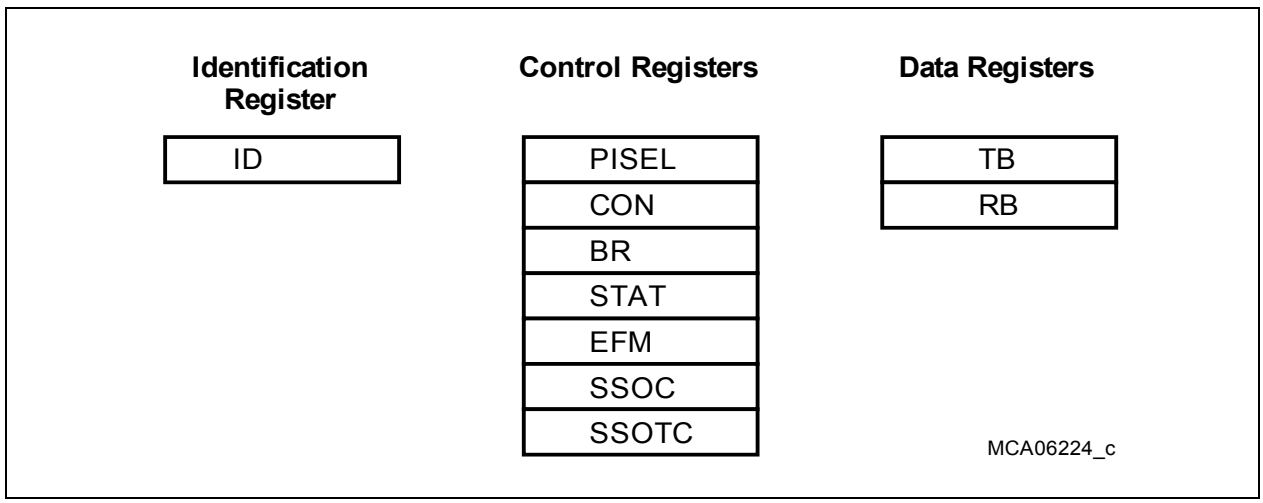


Figure 18-12 SSC Kernel Registers

The complete and detailed address map of the SSC modules is described in **Table 16-19** on **Page 16-50** of the TC1766 User's Manual System Units part (Volume 1).

Table 18-2 Registers Address Space - SSC Kernel Registers

Module	Base Address	End Address	Note
SSC0	$F010\ 0100_H$	$F010\ 02FF_H$	-
SSC1	$F010\ 0200_H$	$F010\ 02FF_H$	-

Table 18-3 Registers Overview - SSC Kernel Registers

Register Short Name	Register Long Name	Offset Address[1)]	Description see
PISEL	Port Input Select Register	04_H	**Page 18-24**
ID	Module Identification Register	08_H	**Page 18-24**
CON	Control Register	10_H	**Page 18-26**
BR	Baud Rate Timer Reload Register	14_H	**Page 18-33**
STAT	Status Register	28_H	**Page 18-28**

Table 18-3 Registers Overview - SSC Kernel Registers (cont'd)

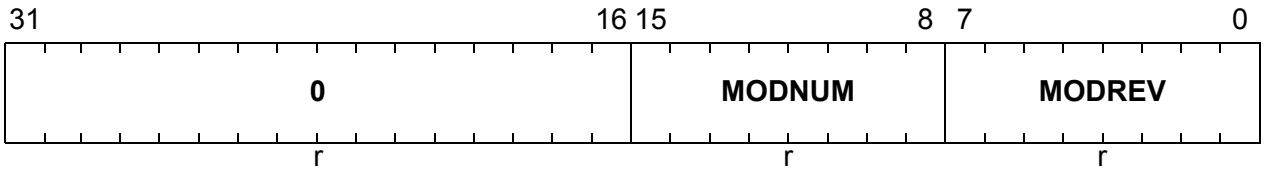
Register Short Name	Register Long Name	Offset Address[1)]	Description see
EFM	Error Flag Modification Register	$2C_H$	Page 18-29
SSOC	Slave Select Output Control Register	18_H	Page 18-31
SSOTC	Slave Select Output Timing Control Register	$1C_H$	Page 18-32
TB	Transmit Buffer Register	20_H	Page 18-34
RB	Receive Buffer Register	24_H	Page 18-34

1) The absolute register address is calculated as follows:
Module Base Address (Table 18-2) + Offset Address (shown in this column)

18.2.1 SSC Module Identification Register

The SSC Module Identification Register ID contains read-only information about the SSC module version.

ID
Module Identification Register **(08_H)** **Reset Value: 0000 45XX$_H$**

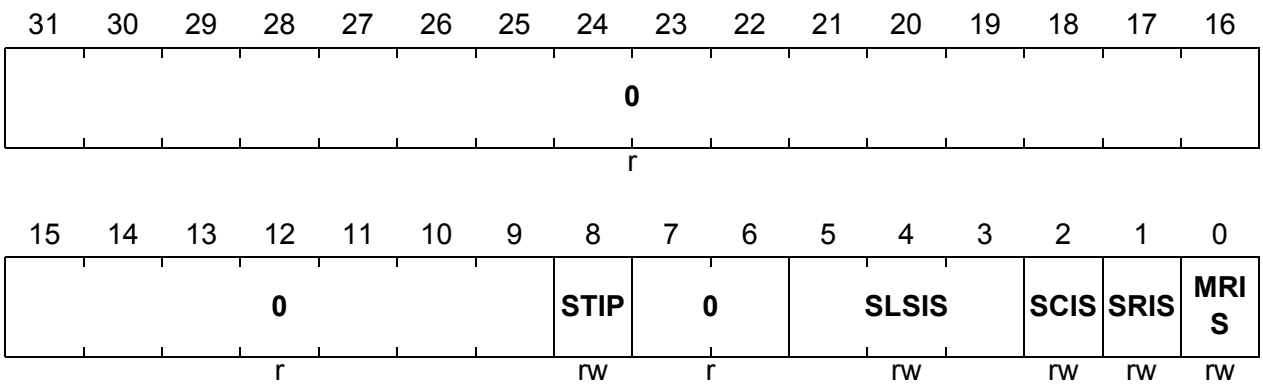


Field	Bits	Type	Description
MODREV	[7:0]	r	**Module Revision Number** MODREV defines the module revision number. The value of a module revision starts with 01_H (first revision).
MODNUM	[15:8]	r	**Module Number Value** This bit field defines the module identification number for the SSC: 45_H
0	[31:16]	r	**Reserved** Read as 0.

18.2.2 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 (port A and port B).

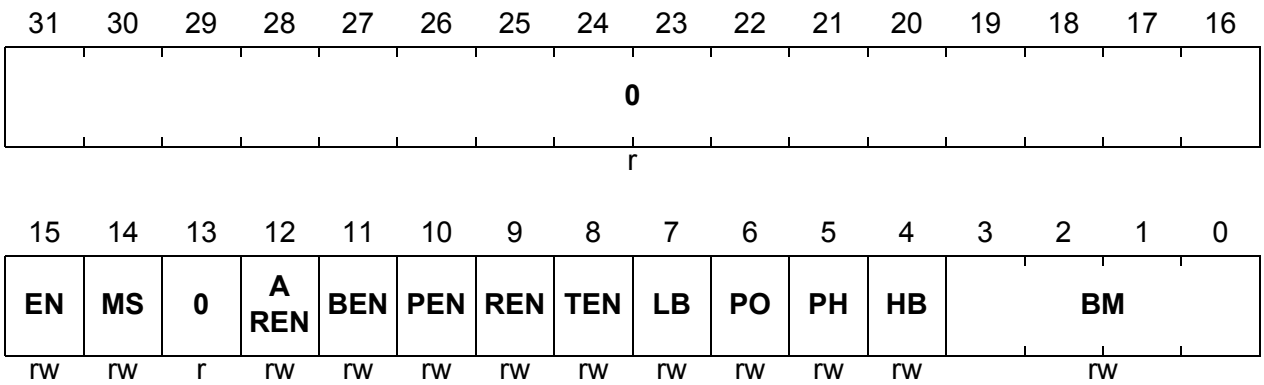
PISEL
Port Input Select Register **(04_H)** **Reset Value: 0000 0000$_H$**



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 that 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
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 is selected for operation. In the TC1766, other combinations of SLSIS 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 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.

CON
Control Register **(10_H)** **Reset Value: $0000\ 0000_H$**

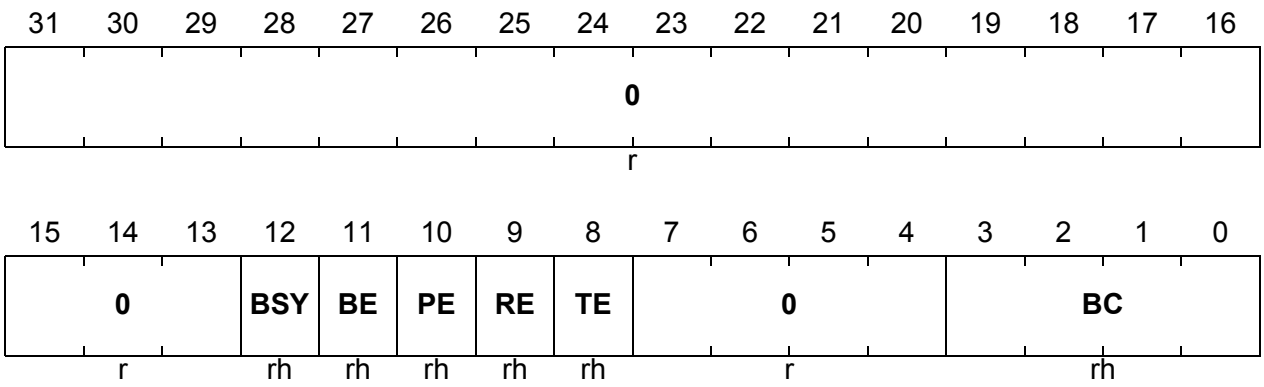


Field	Bits	Type	Description
BM	[3:0]	rw	**Data Width Selection** BM determines the number of data bits of the serial frame. 0000_B Reserved; do not use this combination. 0001_B Transfer Data Width is 2 bit. 0010_B Transfer Data Width is 3 bit. $\ldots_B$ … 1110_B Transfer Data Width is 15 bit. 1111_B 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
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

Field	Bits	Type	Description
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 18-2**).
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 18-2**). Note that EN should only be cleared by software while no transfer is in progress (STAT.BSY = 0)
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$_H$**

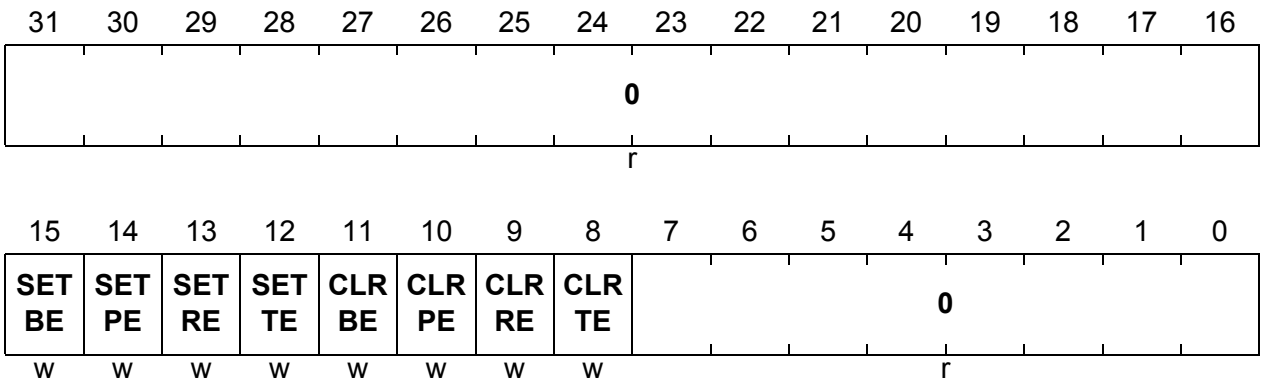


Field	Bits	Type	Description
BC	[3:0]	rh	**Bit Count Status** BC indicates the current status of the shift counter. The shift counter is updated with every shifted bit.
TE	8	rh	**Transmit Error Flag** 0_B No error 1_B Transfer starts with the slave's transmit buffer not being updated
RE	9	rh	**Receive 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's 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.

Field	Bits	Type	Description
SETTE	12	w	**Set Transmit Error Flag** 0_B No effect. 1_B Bit STAT.TE is set. Bit is always read as 0.
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															

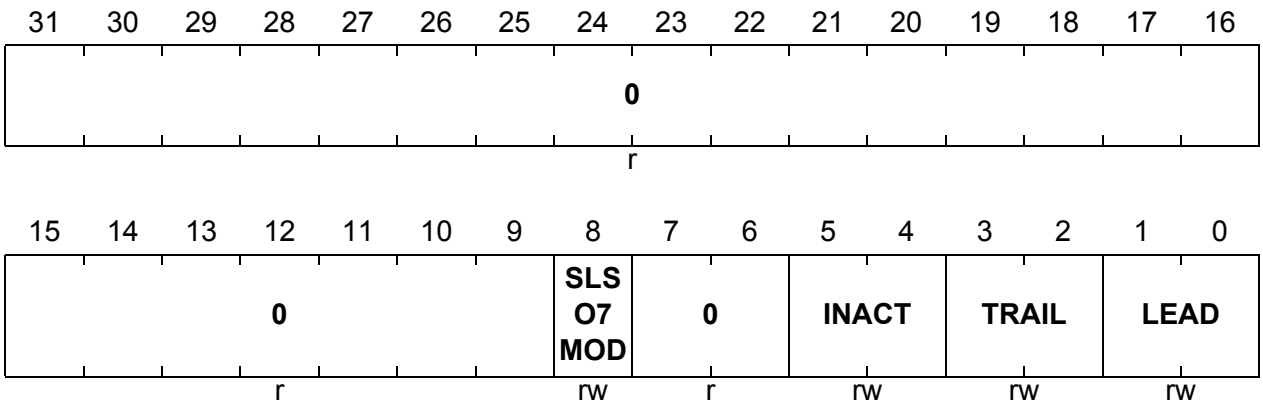
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OEN 7	OEN 6	OEN 5	OEN 4	OEN 3	OEN 2	OEN 1	OEN 0	AOL 7	AOL 6	AOL 5	AOL 4	AOL 3	AOL 2	AOL 1	AOL 0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

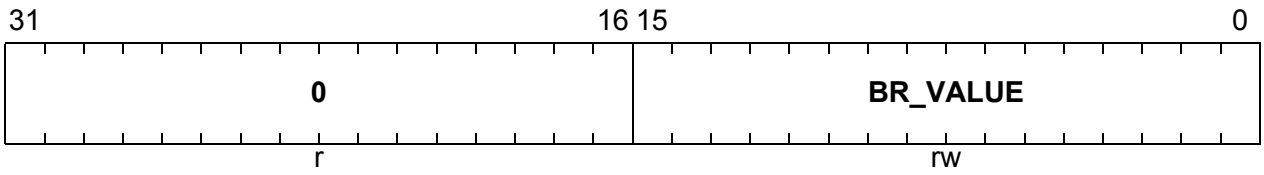
Field	Bits	Type	Description
INACT	[5:4]	rw	**Slave Output Select Inactive Delay** This bit field determines the number of inactive delay clock cycles. An inactive delay clock cycle is always a multiple of an SCLK shift clock period. 00_B Zero inactive delay clock cycle selected[1)] 01_B One inactive 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 18-8), 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 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

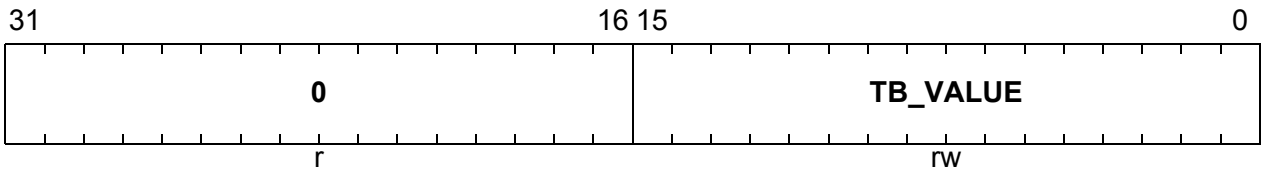


Field	Bits	Type	Description
BR_VALUE	[15:0]	rw	**Baud Rate Timer/Reload Register Value** Reading BR returns the 16-bit content of the baud rate timer. Writing BR loads the baud rate timer reload register with BR_VALUE.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

18.2.3 Data Registers

The Transmit Buffer Register TB contains the transmit data value.

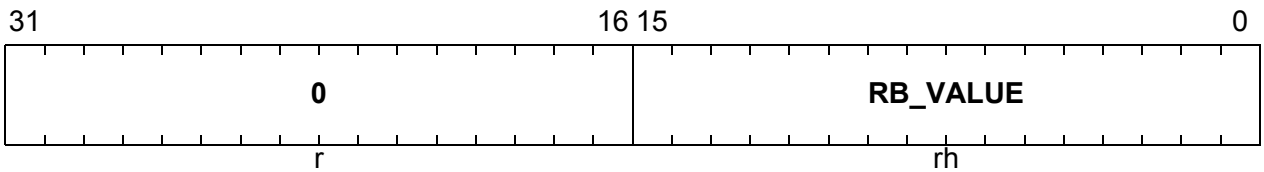
TB
Transmit Buffer Register **(20_H)** **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
TB_VALUE	[15:0]	rw	**Transmit Data Register Value** Register TB_VALUE 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. 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.

18.3 SSC0/SSC1 Module Implementation

This section describes SSC0/SSC1 module interfaces with the clock control, port connections, interrupt control, and address decoding.

18.3.1 Interfaces of the SSC Modules

Figure 18-13 shows the TC1766-specific implementation details and interconnections of the SSC0/SSC1 modules.

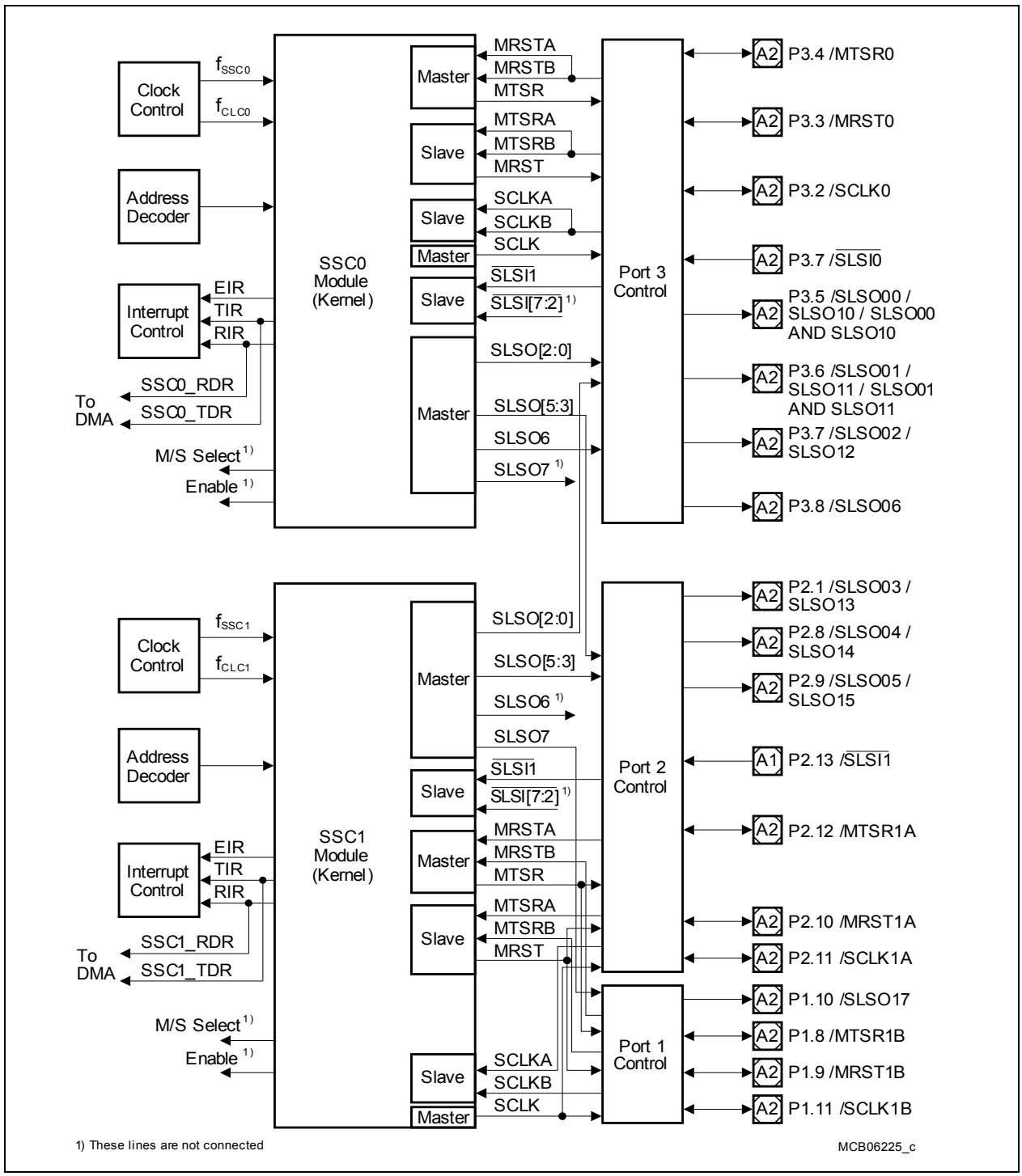


Figure 18-13 SSC0/SSC1 Module Implementation and Interconnections

Each of the SSC modules is supplied with a separate clock control, interrupt control, and address decoding logic. Two interrupt outputs can be used to generate DMA requests. The SSC0/SSC1 I/O lines are connected to Port 1, Port 2 and Port 3.

18.3.2 SSC0/SSC1 Module Related External Registers

Figure 18-14 summarizes the module-related external registers which are required for SSC0/SSC1 programming (see also **Figure 18-12** for the module kernel specific registers).

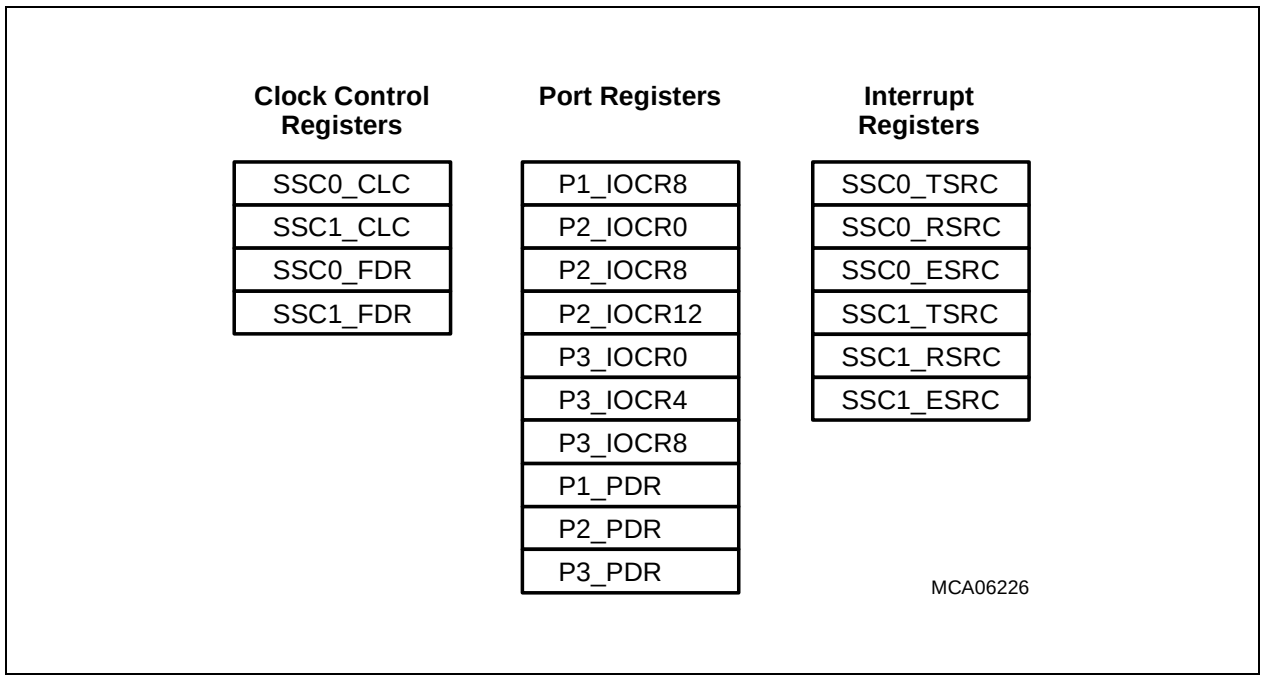


Figure 18-14 SSC0/SSC1 Implementation-specific Special Function Registers

18.3.2.1 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}. 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}.

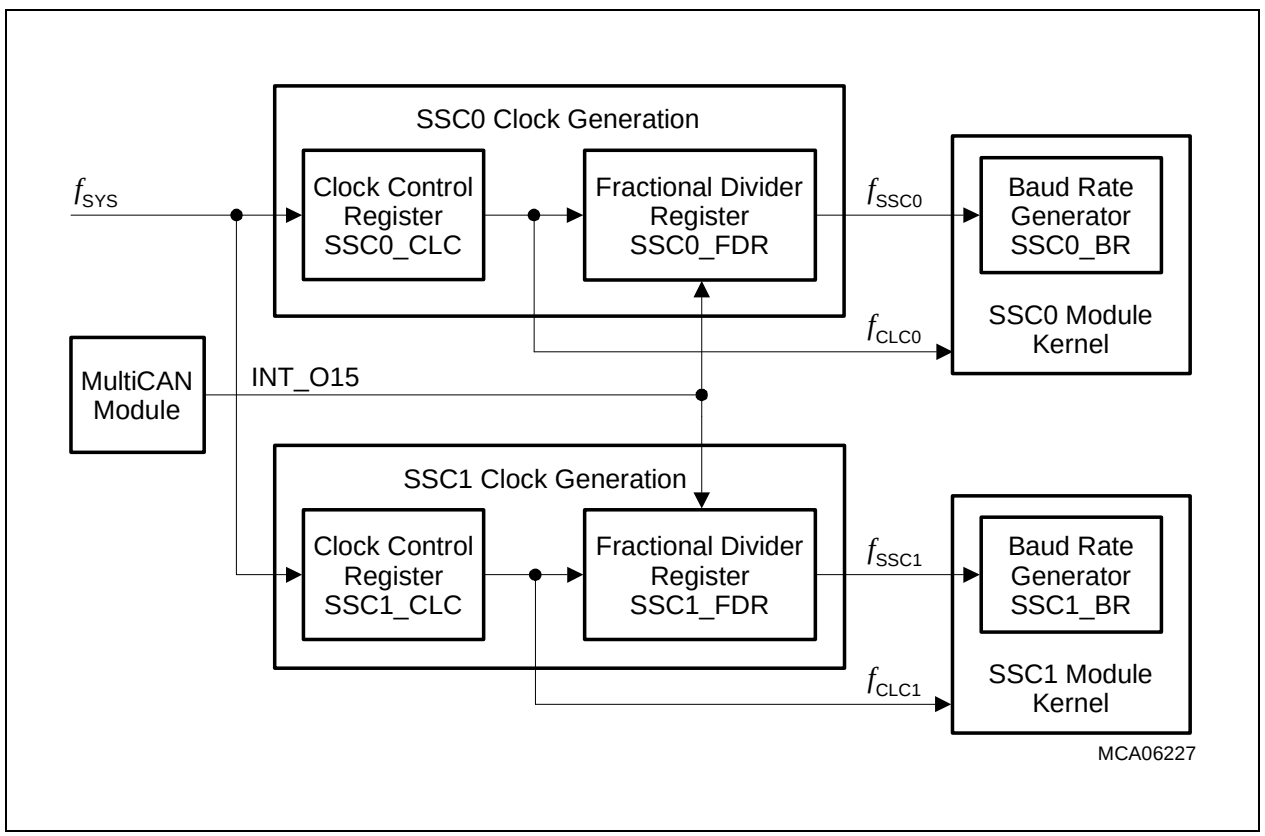


Figure 18-15 SSC Clock Generation

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}

$$f_{SSCx} = f_{SYS} \times \frac{1}{n} \text{ with n = 1024 - FDR.STEP} \quad (18.2)$$

$$f_{SSCx} = f_{SYS} \times \frac{n}{1024} \text{ with n = 0-1023} \quad (18.3)$$

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 18-12**) and the fractional divider (see chapter “System Control Unit” of the TC1766 System Units User’s Manual), the resulting serial data baud rate is defined by:

$$\text{Baud rate}_{SSC} = \frac{f_{SYS}}{2 \times (\text{BR.BR\_VALUE} + 1) \times (1024 - \text{FDR.STEP})} \quad (18.4)$$

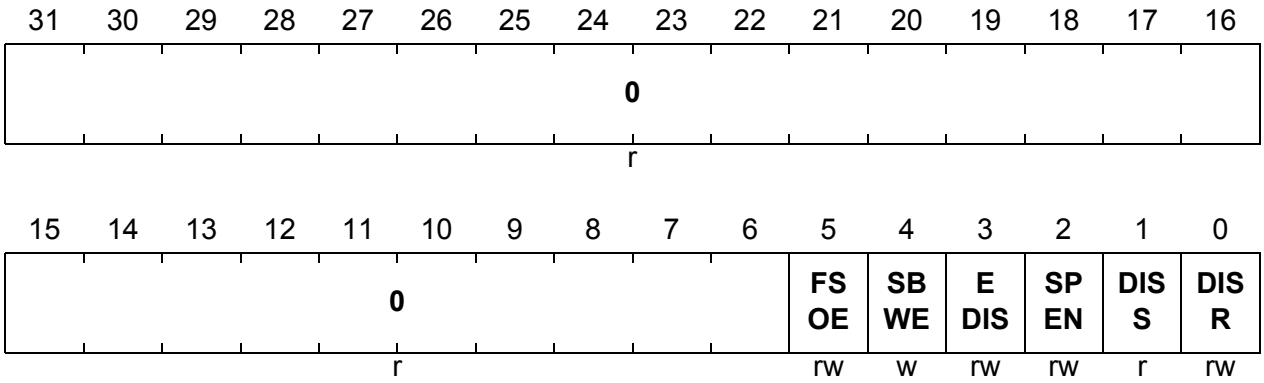
$$\text{Baud rate}_{SSC} = \frac{f_{SYS} \times \text{FDR.STEP}}{2 \times (\text{BR.BR\_VALUE} + 1) \times 1024} \text{ with FDR.STEP = 0-1023} \quad (18.5)$$

*Note: **Equation (18.2)** and **Equation (18.4)** apply to normal divider mode of the fractional divider (FDR.DM = 01_B). **Equation (18.3)** and **Equation (18.5)** 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$_H$**



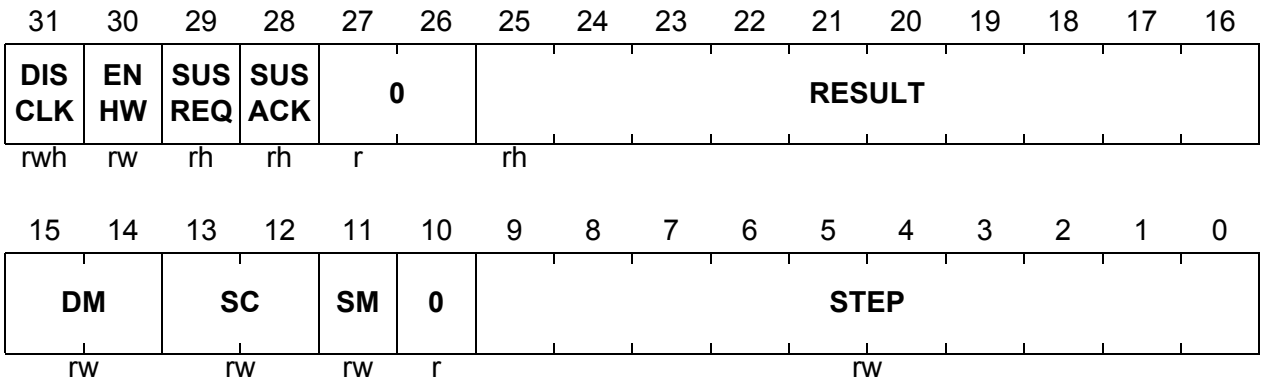
Field	Bits	Type	Description
DISR	0	rw	**Module Disable Request Bit** Used for enable/disable control of the module.
DISS	1	r	**Module Disable Status Bit** Bit indicates the current status of the module.
SPEN	2	rw	**Module Suspend Enable for OCDS** Used to enable the suspend mode.
EDIS	3	rw	**Sleep Mode 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.

*Note: Additional details on the clock control register functionality are described in section **"Clock Control Register CLC" on Page 3-24** of the TC1766 User's Manual System Units part (Volume 1).*

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
Fractional Divider Register ($0C_H$) Reset Value: 0000 0000$_H$
SSC1_FDR
SSC1 Fractional Divider Register ($0C_H$) 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 $\overline{\text{SPNDACK}}$ 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]	rw	**Reserved;** read as 0; should be written with 0.

*Note: Further details on the fractional divider register functionality are described in section **"Fractional Divider Operation" on Page 3-29** of the TC1766 User's Manual System Units part (Volume 1).*

18.3.2.2 Port Control

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 SSC0/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 inputs for the SSC0/SSC1 modules are class A1/A2 GPIO pins that are controlled by the port input/output control registers of Port 1, Port 2 and Port 3.

Table 18-4 shows how bits and bit fields must be programmed for the required I/O functionality of the SSC I/O lines.

Table 18-4 SSC0 and SSC1 I/O Line Selection and Setup

Module	Port Lines	Input/Output Control Register Bits[1)]	I/O
SSC0	P3.4/MTSR0 (Slave Mode)	P3_IOCR4.PC4 = $0XXX_B$	Input
	P3.4/MTSR0 (Master Mode)	P3_IOCR4.PC4 = $1X01_B$ or $1X10_B$	Output
	P3.3/MRST0 (Master Mode)	P3_IOCR0.PC3 = $0XXX_B$	Input
	P3.3/MRST0 (Slave Mode)	P3_IOCR0.PC3 = $1X01_B$ or $1X10_B$	Output
	P3.2/SCLK0 (Slave Mode)	P3_IOCR0.PC2 = $0XXX_B$	Input
	P3.2/SCLK0 (Master Mode)	P3_IOCR0.PC2 = $1X01_B$ or $1X10_B$	Output
	P3.7/$\overline{\text{SLSI0}}$	P3_IOCR4.PC7 = $0XXX_B$	Input

Table 18-4 SSC0 and SSC1 I/O Line Selection and Setup (cont'd)

Module	Port Lines	Input/Output Control Register Bits[1)]	I/O
SSC1	P2.12/MTSR1A (Slave Mode)	P2_IOCR12.PC12 = $0XXX_B$	Input
	P2.12/MTSR1A (Master Mode)	P2_IOCR12.PC12 = $1X01_B$ or $1X10_B$	Output
	P2.10/MRST1A (Master Mode)	P2_IOCR8.PC10 = $0XXX_B$	Input
	P2.10/MRST1A (Slave Mode)	P2_IOCR8.PC10 = $1X01_B$ or $1X10_B$	Output
	P2.11/SCLK1A (Slave Mode)	P2_IOCR8.PC11 = $0XXX_B$	Input
	P2.11/SCLK1A (Master Mode)	P2_IOCR8.PC11 = $1X01_B$ or $1X10_B$	Output
	P1.8/MTSR1B (Slave Mode)	P1_IOCR8.PC8 = $0XXX_B$	Input
	P1.8/MTSR1B (Master Mode)	P1_IOCR8.PC8 = $1X11_B$	Output
	P1.9/MRST1B (Master Mode)	P1_IOCR8.PC9 = $0XXX_B$	Input
	P1.9/MRST1B (Slave Mode)	P1_IOCR8.PC9 = $1X11_B$	Output
	P1.11/SCLK1B (Slave Mode)	P1_IOCR8.PC11 = $0XXX_B$	Input
	P1.11/SCLK1B (Master Mode)	P1_IOCR8.PC11 = $1X11_B$	Output
	P2.13/$\overline{\text{SLSI1}}$	P2_IOCR12.PC13 = $0XXX_B$	Input
Slave Select Outputs			
SSC0	P3.5/SLSO00	P3_IOCR4.PC5 = $1X01_B$	Output
	P3.6/SLSO01	P3_IOCR4.PC6 = $1X01_B$	Output
	P3.7/SLSO02	P3_IOCR4.PC7 = $1X01_B$	Output
	P2.1/SLSO03	P2_IOCR0.PC1 = $1X10_B$	Output
	P2.8/SLSO04	P2_IOCR8.PC8 = $1X01_B$	Output
	P2.9/SLSO05	P2_IOCR8.PC9 = $1X01_B$	Output
	P3.8/SLSO06	P3_IOCR8.PC8 = $1X01_B$	Output

Table 18-4 SSC0 and SSC1 I/O Line Selection and Setup (cont'd)

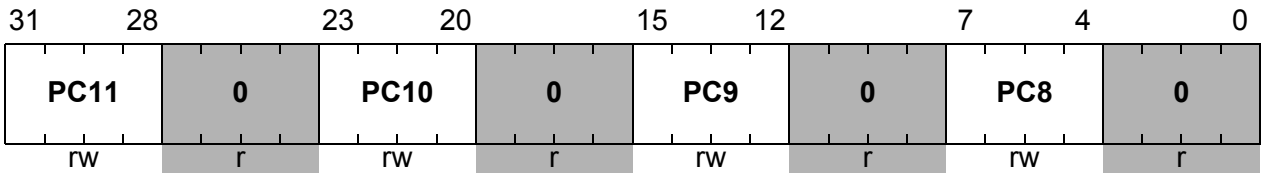
Module	Port Lines	Input/Output Control Register Bits[1)]	I/O
SSC1	P3.5/SLSO10	P3_IOCR4.PC5 = $1X10_B$	Output
	P3.6/SLSO11	P3_IOCR4.PC6 = $1X10_B$	Output
	P3.7/SLSO12	P3_IOCR4.PC7 = $1X10_B$	Output
	P2.1/SLSO13	P2_IOCR0.PC1 = $1X11_B$	Output
	P2.8/SLSO14	P2_IOCR8.PC8 = $1X10_B$	Output
	P2.9/SLSO15	P2_IOCR8.PC9 = $1X10_B$	Output
	P1.10/SLSO17	P1_IOCR8.PC10 = $1X11_B$	Output
SSC0 & SSC1	P3.5/SLSO00 AND SLSO10	P3_IOCR4.PC5 = $1X11_B$	Output
	P3.6/SLSO01 AND SLSO11	P3_IOCR4.PC6 = $1X11_B$	Output

1) For possible PCx bit field combinations, see **Table 18-5**.

Input/Output Control Registers

P1_IOCR8

Port 1 Input/Output Control Register 8(18_H) **Reset Value: 2020 2020_H**

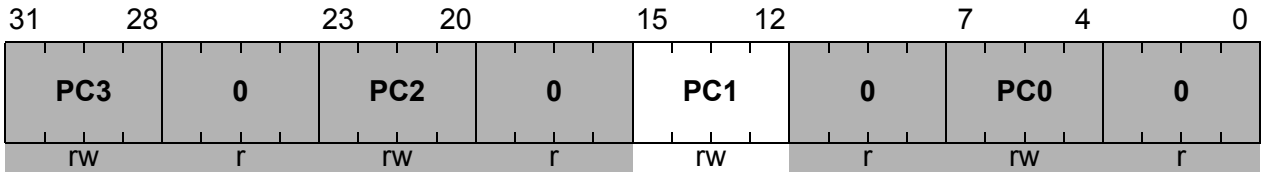


Field	Bits	Type	Description
		rw	**Port Output Control for Port 1.[11:8]**[1)] These bit fields determine the output port functionality:
PC8	[7:4]		Port input/output control for P1.8/MTSR1B
PC9	[15:12]		Port input/output control for P1.9/MRST1B
PC10	[23:20]		Port output control for P1.10/SLSO17
PC11	[31:28]		Port input/output control for P1.11/SCLK1B

1) For coding of bit field, see **Table 18-5**. Shaded bits and bit fields are “don’t care” for SSC I/O port control.

P2_IOCR0

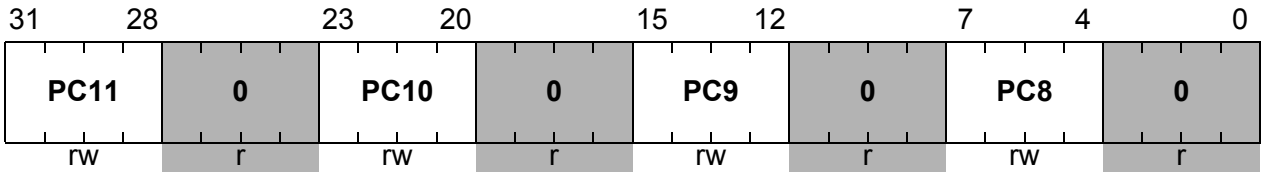
Port 2 Input/Output Control Register 0(10_H) **Reset Value: 2020 2020_H**



Field	Bits	Type	Description
PC1	[15:12]	rw	**Port Output Control for Port 2.1**[1)] These bit fields determine the output port functionality: Port output control for P2.1/SLSO03/SLSO13

1) For coding of bit fields, see **Table 18-5**. Shaded bits and bit fields are “don’t care” for SSC I/O port control.

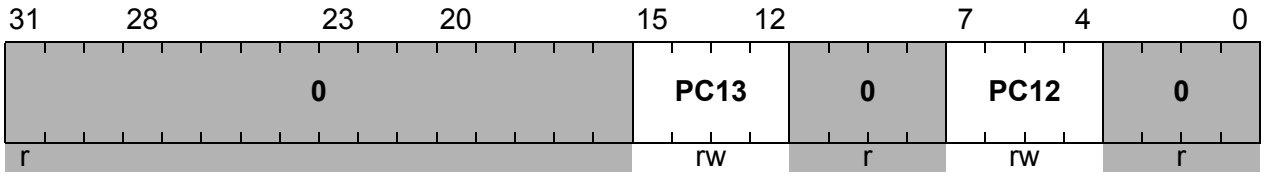
P2_IOCR8
Port 2 Input/Output Control Register 8(18_H) **Reset Value: 2020 2020_H**



Field	Bits	Type	Description
		rw	**Port Output Control for Port 2.[11:8]**[1)] These bit fields determine the output port functionality:
PC8	[7:4]		Port output control for P2.8/SLSO04/SLSO14
PC9	[15:12]		Port output control for P2.9/SLSO05/SLSO15
PC10	[23:20]		Port input/output control for P2.10/MRST1A
PC11	[31:28]		Port input/output control for P2.11/SCLK1A

1) For coding of bit field, see **Table 18-5**. Shaded bits and bit fields are “don’t care” for SSC I/O port control.

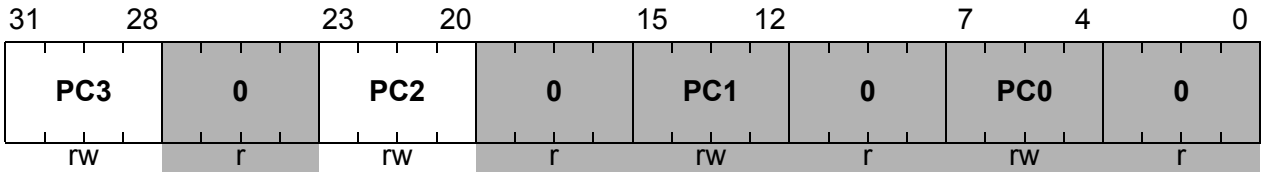
P2_IOCR12
Port 2 Input/Output Control Register 12($1C_H$) **Reset Value: 2020 2020_H**



Field	Bits	Type	Description
		rw	**Port Output Control for Port 2.12 and Port 2.13**[1)] These bit fields determine the output port functionality:
PC12	[7:4]		Port input/output control for P2.12/MTSR1A
PC13	[15:12]		Port input control for P2.13/SLSI1

1) For coding of bit field, see **Table 18-5**. Shaded bits and bit fields are “don’t care” for SSC I/O port control.

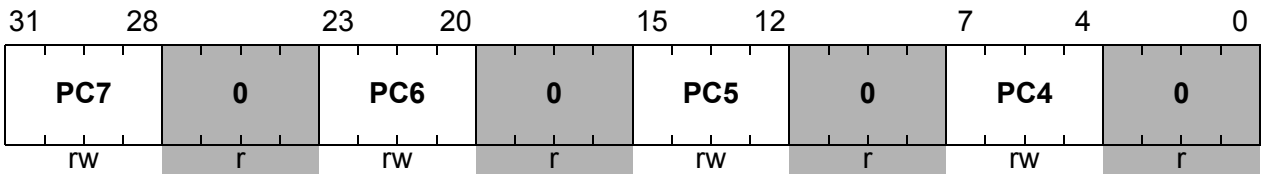
P3_IOCR0
Port 3 Input/Output Control Register 0(10_H) **Reset Value: 2020 2020_H**



Field	Bits	Type	Description
		rw	**Port Output Control for Port 3.2 and Port 3.3[1)]** These bit fields determine the output port functionality:
PC2	[23:20]		Port input/output control for P3.2/SCLK0
PC3	[31:28]		Port input/output control for P3.3/MRST0

1) For coding of bit field, see **Table 18-5**. Shaded bits and bit fields are “don’t care” for SSC I/O port control.

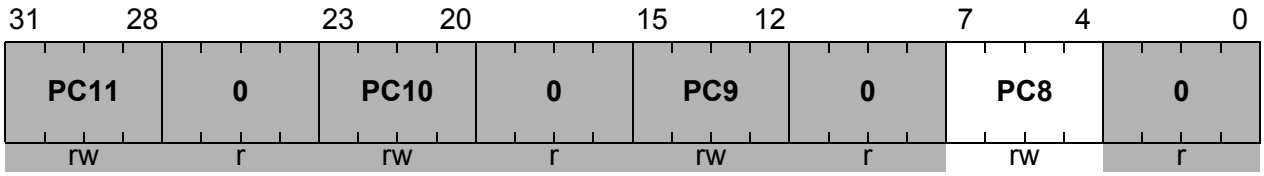
P3_IOCR4
Port 3 Input/Output Control Register 4(14_H) **Reset Value: 2020 2020_H**



Field	Bits	Type	Description
		rw	**Port Output Control for Port 3.[7:4][1)]** These bit fields determine the output port functionality:
PC4	[7:4]		Port input/output control for P3.4/MTSR0
PC5	[15:12]		Port output control for P3.5/SLSO00/SLSO10/SLSO00 AND SLSO10
PC6	[23:20]		Port output control for P3.6/SLSO01/SLSO11/SLSO01 AND SLSO11
PC7	[31:28]		Port input control for P3.7/ $\overline{\text{SLSI0}}$ output control for P3.7/SLSO02/SLSO12

1) For coding of bit field, see **Table 18-5**. Shaded bits and bit fields are “don’t care” for SSC I/O port control.

P3_IOCR8
Port 3 Input/Output Control Register 8(18_H) **Reset Value: 2020 2020_H**



Field	Bits	Type	Description
PC8	[7:4]	rw	**Port Output Control for Port 3.8[1)]** These bit fields determine the output port functionality: Port output control for P3.8/SLSO06

1) For coding of bit field, see **Table 18-5**. Shaded bits and bit fields are “don’t care” for SSC I/O port control.

Table 18-5 PCx Coding

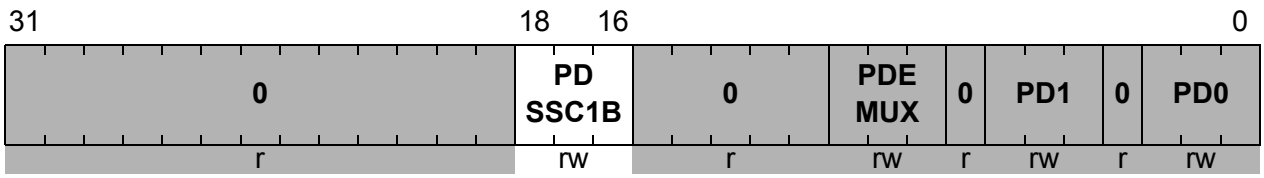
PCx[3:0]	I/O	Output Characteristics	Selected Pull-up/Pull-down/ Selected Output Function
$0X00_B$	Input	–	No pull device connected
$0X01_B$			Pull-down device connected
$0X10_B$[1)]			Pull-up device connected
$0X11_B$			No pull device connected
1001_B	Output	Push-pull	Output function ALT1
1101_B		Open-drain	
1010_B		Push-pull	Output function ALT2
1110_B		Open-drain	
1011_B		Push-pull	Output function ALT3
1111_B		Open-drain	

1) This bit field value is default after reset.

Pad Driver Mode Registers

The Port 1, Port 2 and Port 3 Pad Driver Mode Registers contain bit fields that determine the output driver strength, and the slew rate of SSC output lines. A detailed description of all available PDx bit field combinations is given in Chapter “GPIO Ports and Peripheral I/O” in the TC1766 System Units User’s Manual.

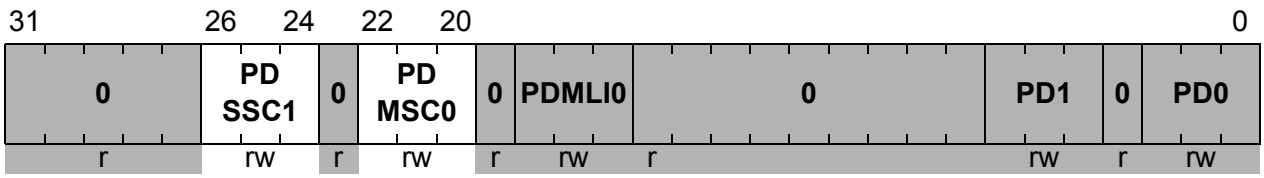
P1_PDR
Port 1 Pad Driver Mode Register (40_H) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
PDSSC1B	[18:16]	rw	**Pad Driver Mode for P1.8/MTSR1B, P1.9/MRST1B, P1.10/SLSO17 and P1.11/SCLK1B[1)]**

1) For coding of bit field, see **Table 18-6**. Shaded bits and bit fields “don’t care” for SSC I/O port control.

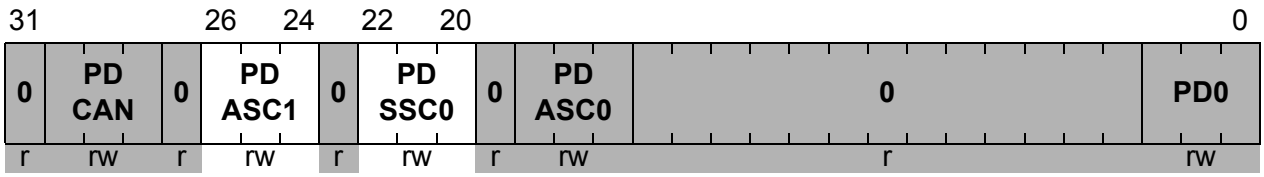
P2_PDR
Port 2 Pad Driver Mode Register (40_H) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
PDMSC0	[22:20]	rw	**Pad Driver Mode for P2.1/ SLSO03/SLSO13, P2.[9:8]/SLSO0[5:4]/SLSO1[5:4][1)]**
PDSSC1	[26:24]	rw	**Pad Driver Mode for P2.10/MRST1A, P2.11/SCLK1A, P2.12/MTSR1A[1)]**

1) For coding of bit field, see **Table 18-6**. Shaded bits and bit fields “don’t care” for SSC I/O port control.

P3_PDR
Port 3 Pad Driver Mode Register (40_H) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
PDSSC0	[22:20]	rw	**Pad Driver Mode for P3.2/SCLK0, P3.3/MRST0, P3.4/MTSR0 and P3.[7:5]/SLSO0[2:0]/SLSO1[2:0]1)**
PDASC1	[26:24]	rw	**Pad Driver Mode for P3.8/SLSO061)**

1) For coding of bit field, see **Table 18-6**. Shaded bits and bit fields “don’t care” for SSC I/O port control.

PDx Selection Table

Table 18-6 Pad Driver Mode Mode Selection (Class A2 Pads)

PDx Bit Field	Driver Strength	Signal Transitions
000_B	Strong driver	Sharp edge[1)]
001_B		Medium edge[1)]
010_B		Soft edge[1)]
011_B	Weak driver	–
100_B	Medium driver	Sharp edge
101_B		Medium edge
110_B		Soft edge
111_B	Weak driver	–

1) In strong driver mode, the output driver characteristics of class A2 pads can be additionally controlled by the temperature compensation logic.

18.3.2.3 Interrupt Control Registers

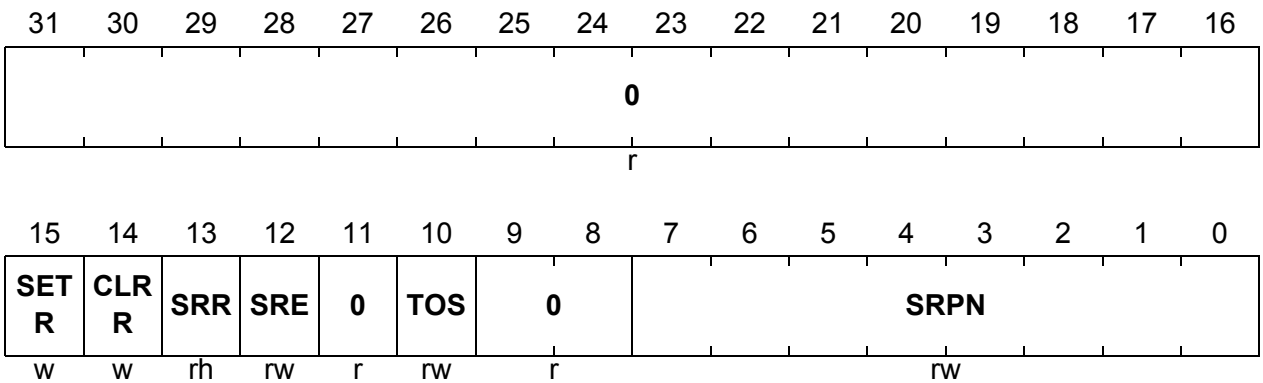
The 2 × 3 interrupts of the SSC0 and SSC1 modules are controlled by the following service request control registers:

- 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.

*Note: Additional details on service request nodes and the service request control registers are described on **Page 12-3** of the TC1766 User's Manual System Units part (Volume 1).*

18.3.3 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 18-7**.

Table 18-7 DMA Request Lines of SSC0/SSC1

Module	SSC Interrupt Request Line	DMA Request Line	Description
SSC0	RIR	SSC0_RDR	SSC0 Receive DMA Request
	TIR	SSC0_TDR	SSC0 Transmit DMA Request
SSC1	RIR	SSC1_RDR	SSC1 Receive DMA Request
	TIR	SSC1_TDR	SSC1 Transmit DMA Request

*Note: Additional details on DMA handling and processing are described in **Chapter 11** of the TC1766 System Units User's Manual.*

19 Micro Second Channel (MSC)

This chapter describes the Micro Second Channel Interface, MSC0 of the TC1766. It contains the following sections:

- Functional description of the MSC kernel (see **Page 19-3**)
- MSC kernel register descriptions (see **Page 19-36**)
- TC1766 implementation-specific details and registers of the MSC module (port connections and control, interrupt control, address decoding, and clock control, see **Page 19-62**)

*Note: The MSC kernel register names described in **Section 19.2** are referenced in the TC1766 User's Manual by the module name prefix "MSC0_" for the MSC0 module.*

MSC Applications

The MSC is a serial interface that is especially designed to connect external power devices to the TC1766. 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 19-1 shows a typical TC1766 application in which an MSC interface controls two power devices. Output data is provided by the GPTA module.

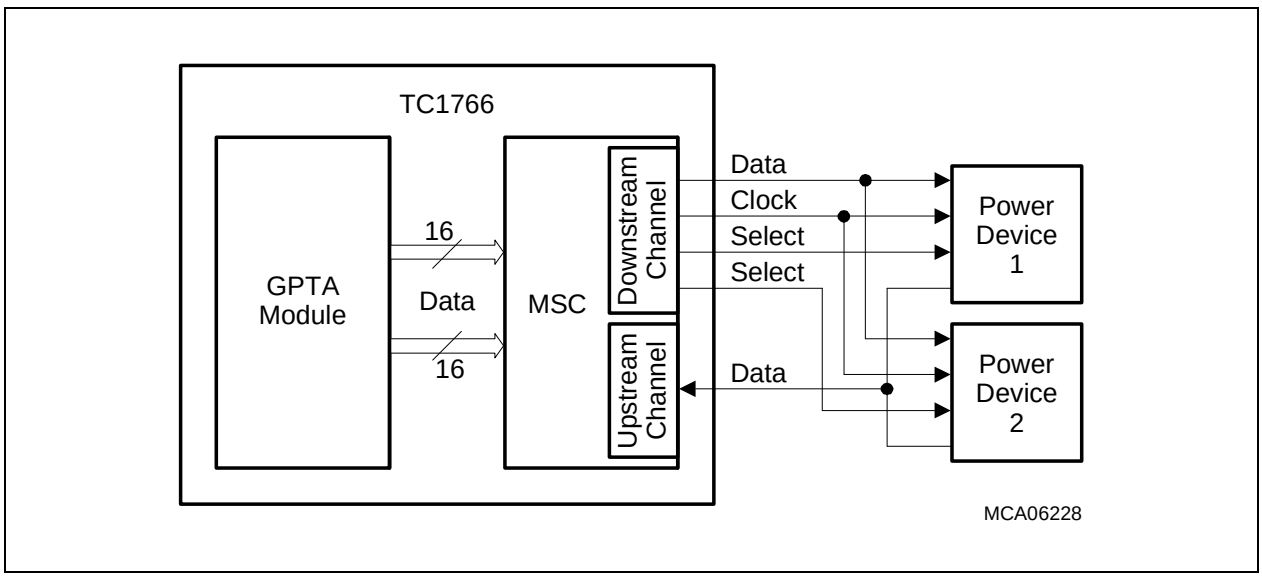


Figure 19-1 MSC to External Power Device Connection

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 TC1766 to other peripheral devices

19.1 MSC Kernel Description

This section describes the functionality of the MSC kernel.

19.1.1 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 19-2** shows a global view of the MSC interface signals.

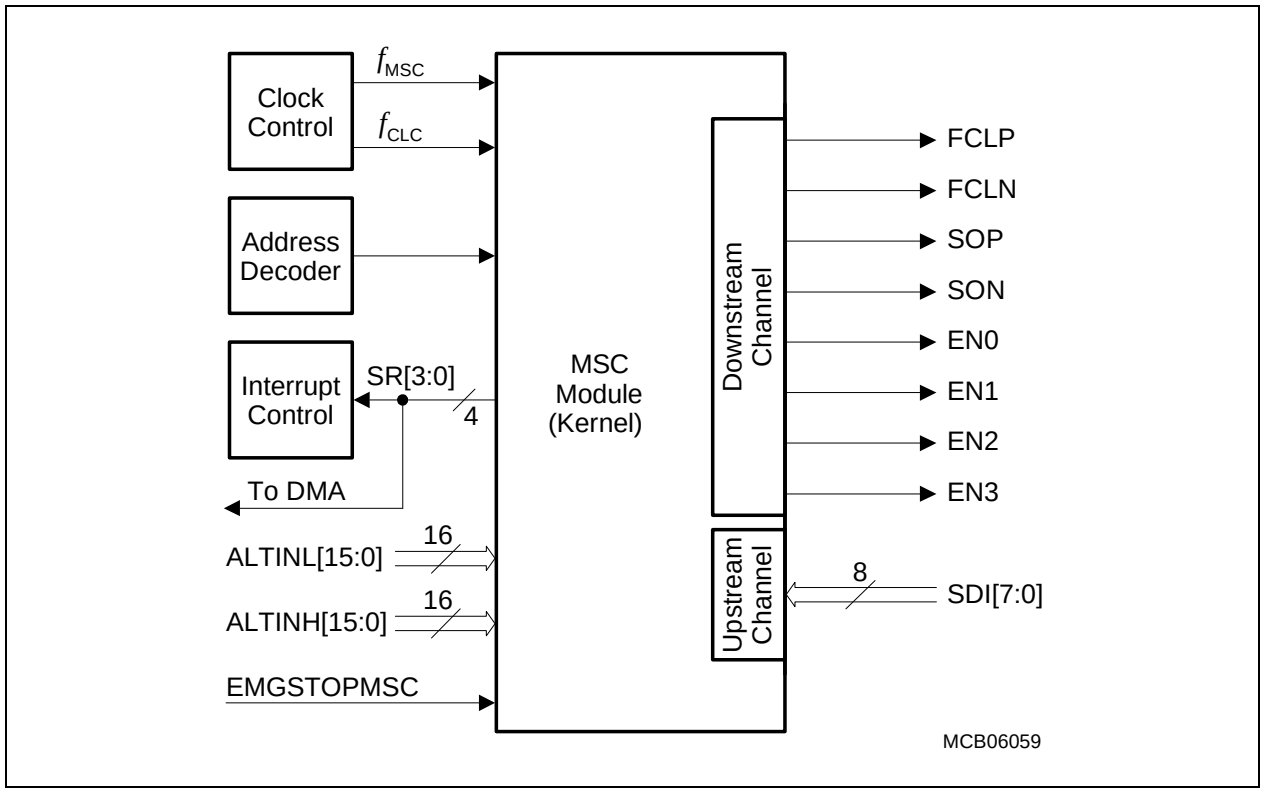


Figure 19-2 General Block Diagram of the MSC Interface

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: $f_{FCL} = f_{MSC}/2$
  - 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
  - Standard asynchronous serial frames
  - Parity error checker
  - 8-to-1 input multiplexer for SDI lines
  - Built-in spike filter on SDI lines

19.1.2 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 19-3 is a diagram of the MSC downstream channel.

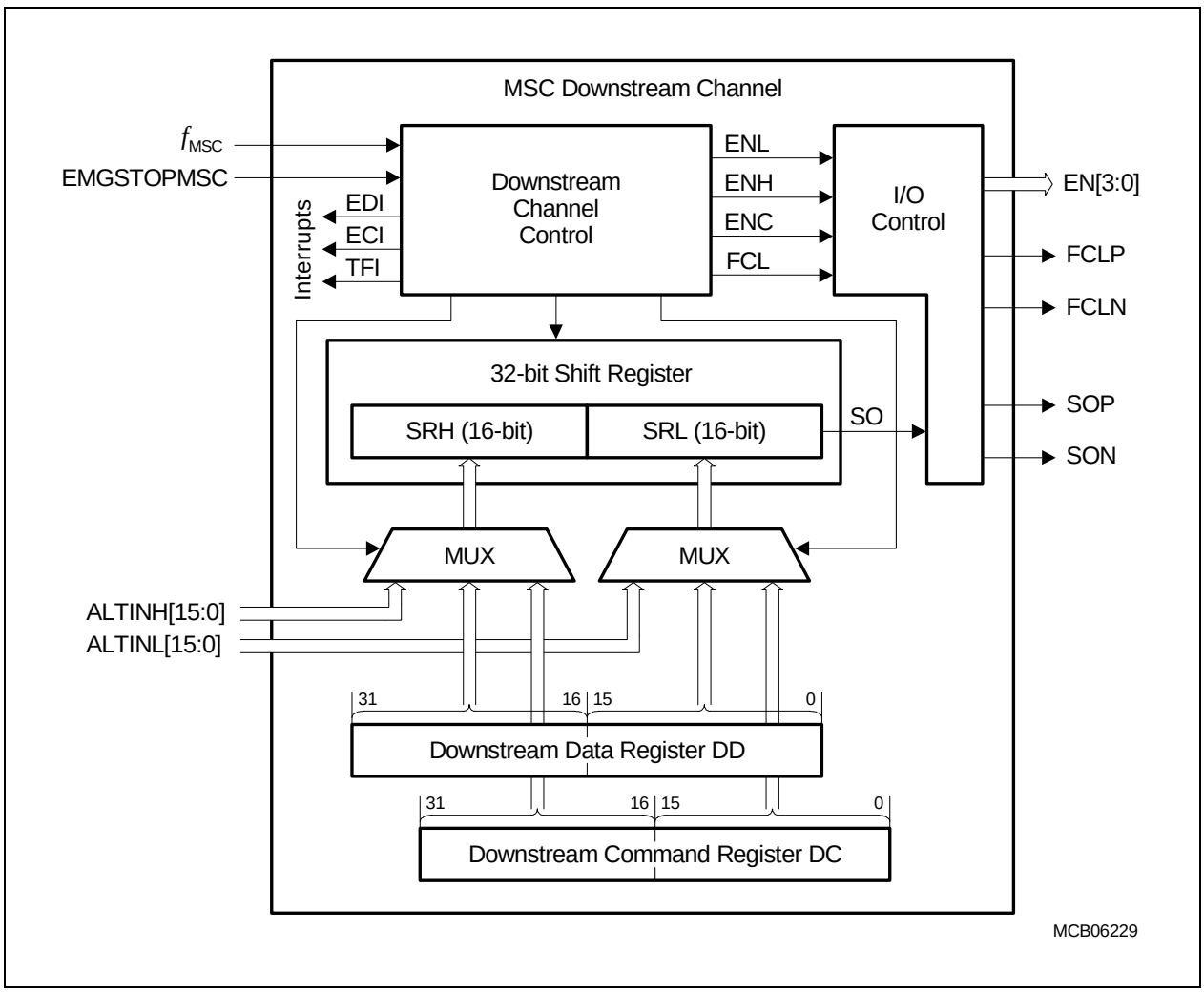


Figure 19-3 Downstream Channel Block Diagram

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.

19.1.2.1 Frame Formats and Definitions

This section describes the frame formats and definitions of the MSC.

Basic Definitions

Figure 19-4 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 19-4**, 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.

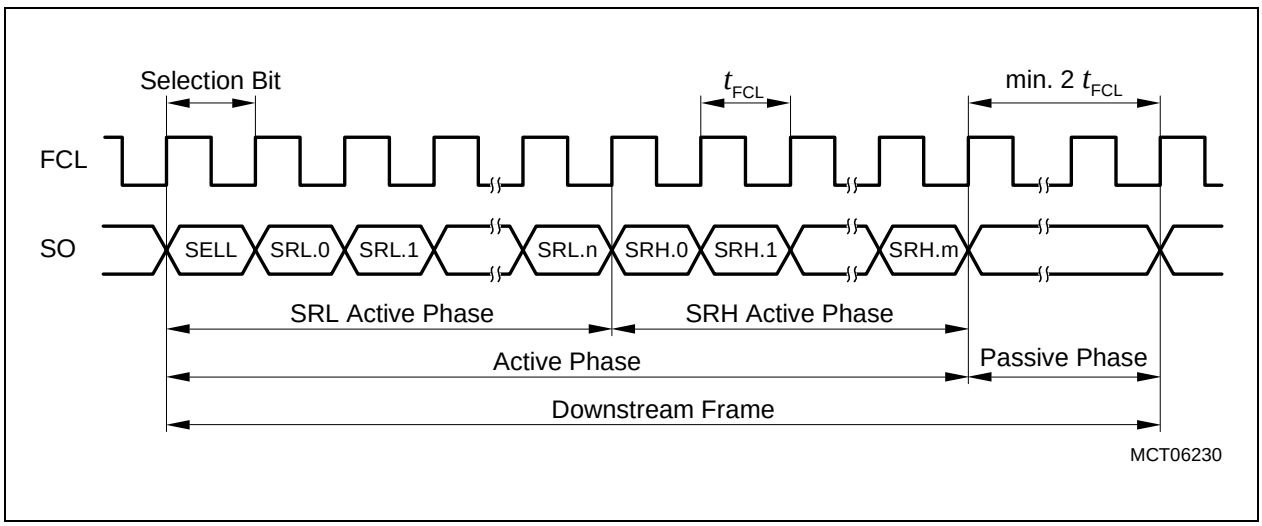


Figure 19-4 Downstream Channel Frame

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 19-5** assumes that the FCL clock is only generated during the active phase of the command frame (OCR.CLKCTRL = 0).

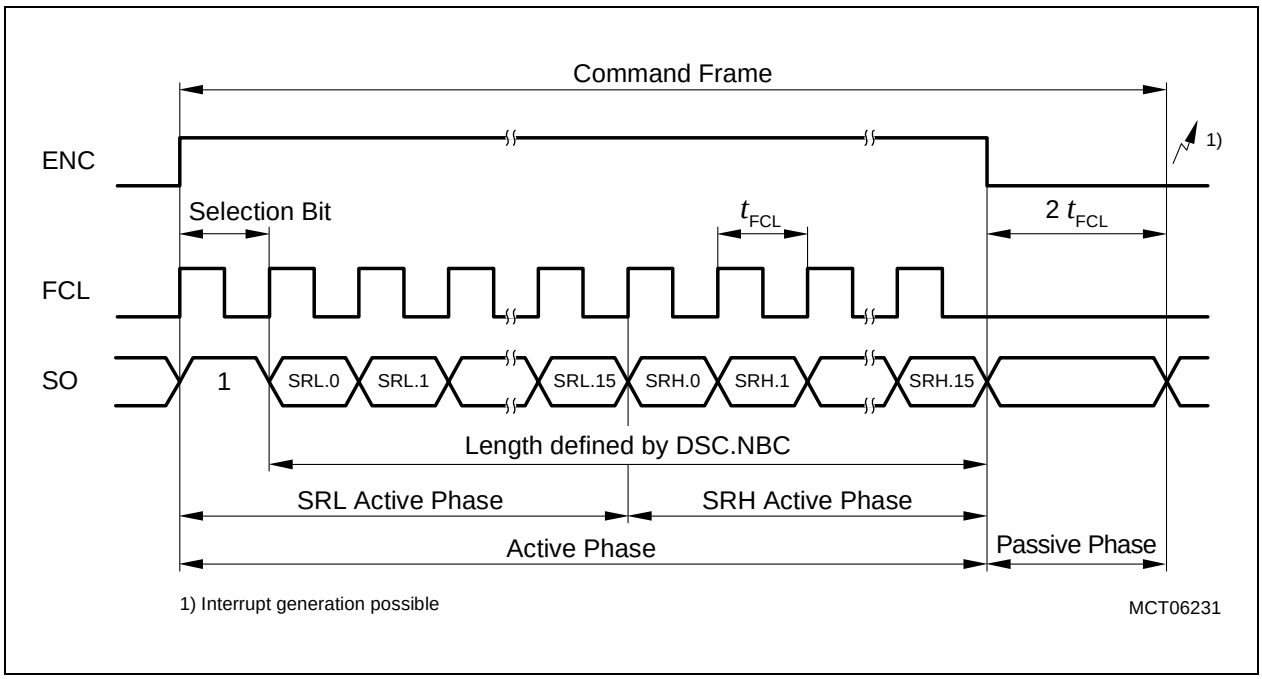


Figure 19-5 Command Frame Layout

Table 19-1 shows the programming of the bits to be transmitted and the resulting length of the complete command frame.

Table 19-1 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 + 19 + 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 19-5** assumes that the FCL clock is only generated during the active phase of the data frame (OCR.CLKCTRL = 0).

Table 19-2, **Table 19-3**, and **Table 19-4** show the definitions of the five data frame parameters that determine the layout of the data frame.

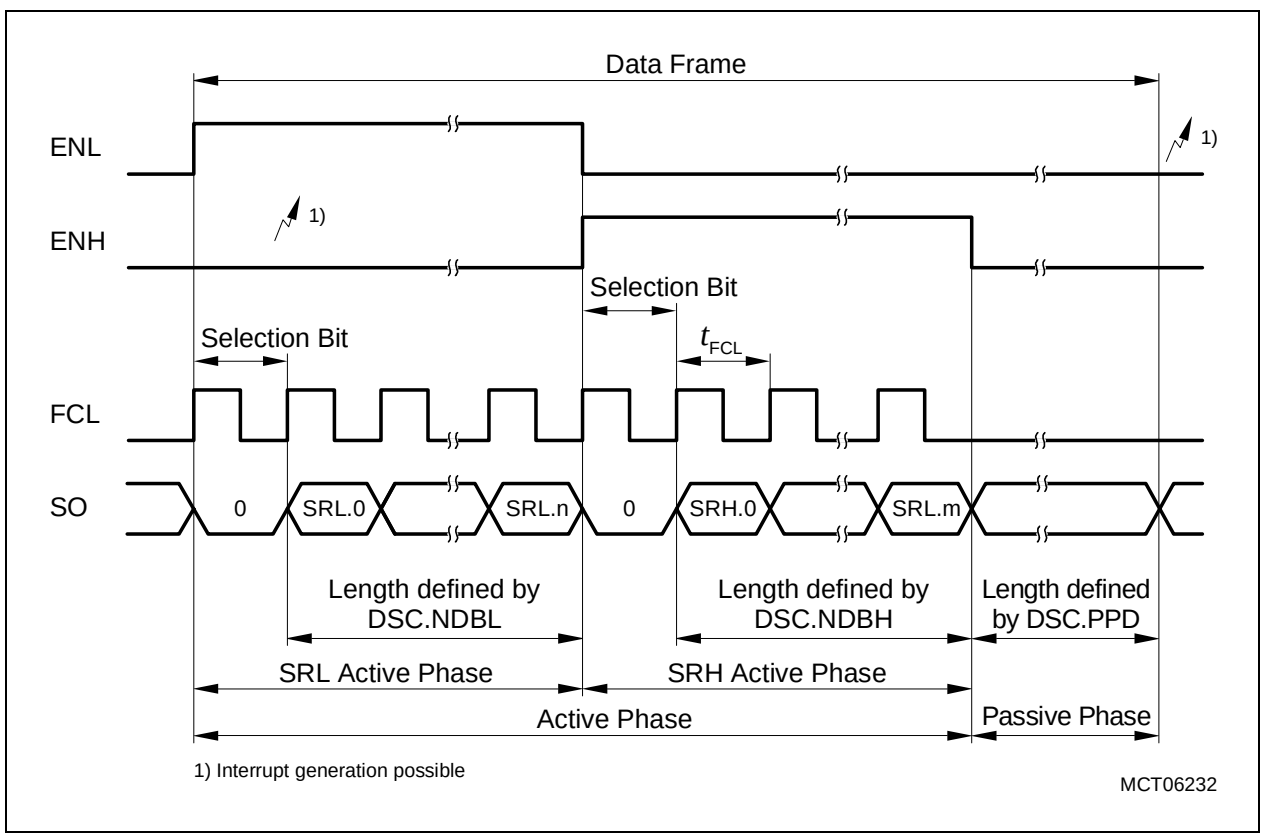


Figure 19-6 Data Frame Layout

Table 19-2 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 19-3 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 19-4 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.

$$\text{Number of cycles} = \text{ENSELL} + \text{NDBL} + \text{ENSELH} + \text{NDBH} + \text{PPD} \quad (19.1)$$

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 (19.1)**. They are generated only in Data Repetition Mode. Under special conditions (command frame insertion), passive time frames can be shortened (see **Figure 19-9**).

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.

19.1.2.2 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 19-7** 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.

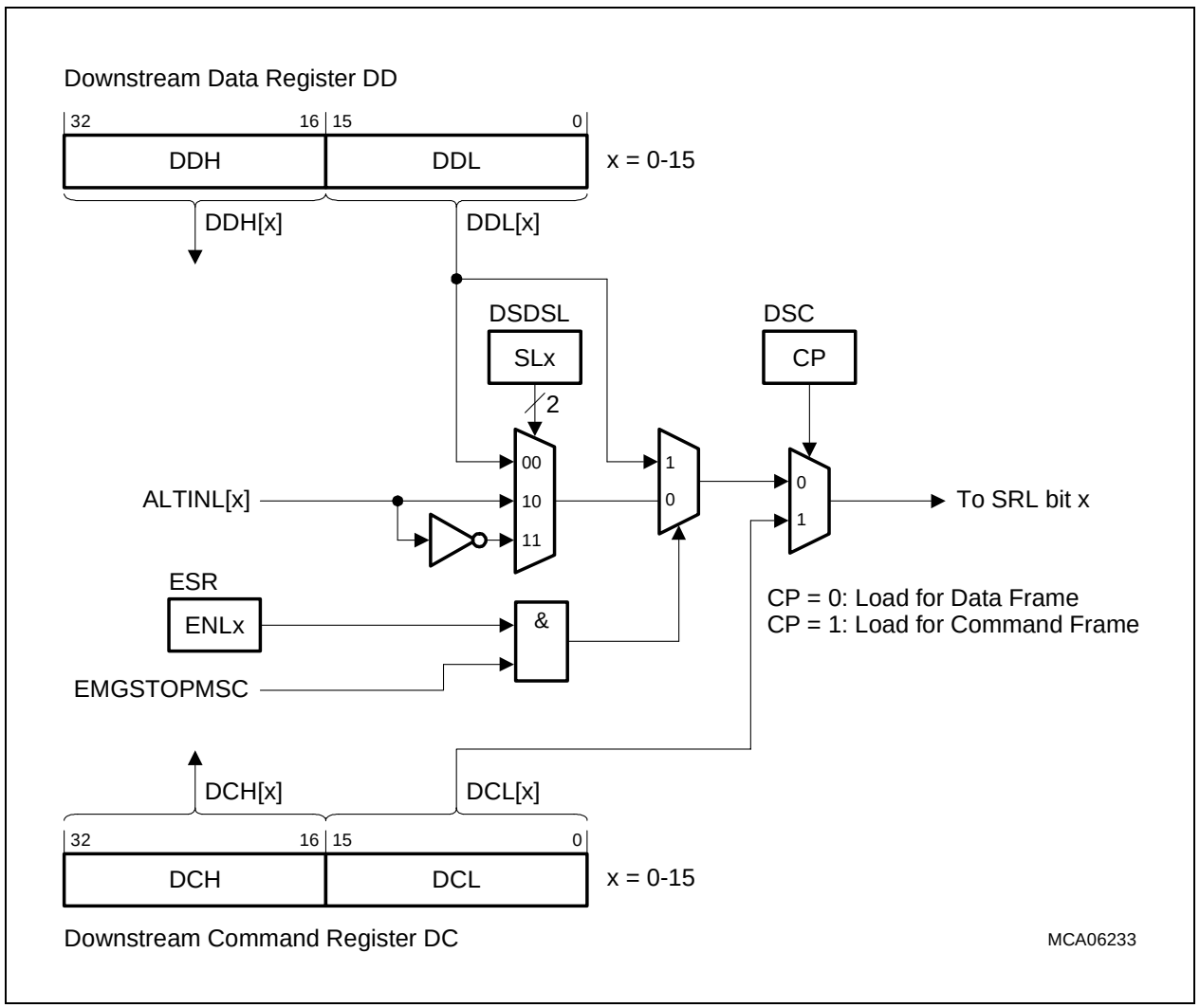


Figure 19-7 SRL Shift Register Data Loading Control

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 19-5 summarizes all SRL data source selection capabilities (x = 0-15).

Table 19-5 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 DDL.

19.1.2.3 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 19-8 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 19.1.2.2** occurs just before a command or data frame transmission is started.

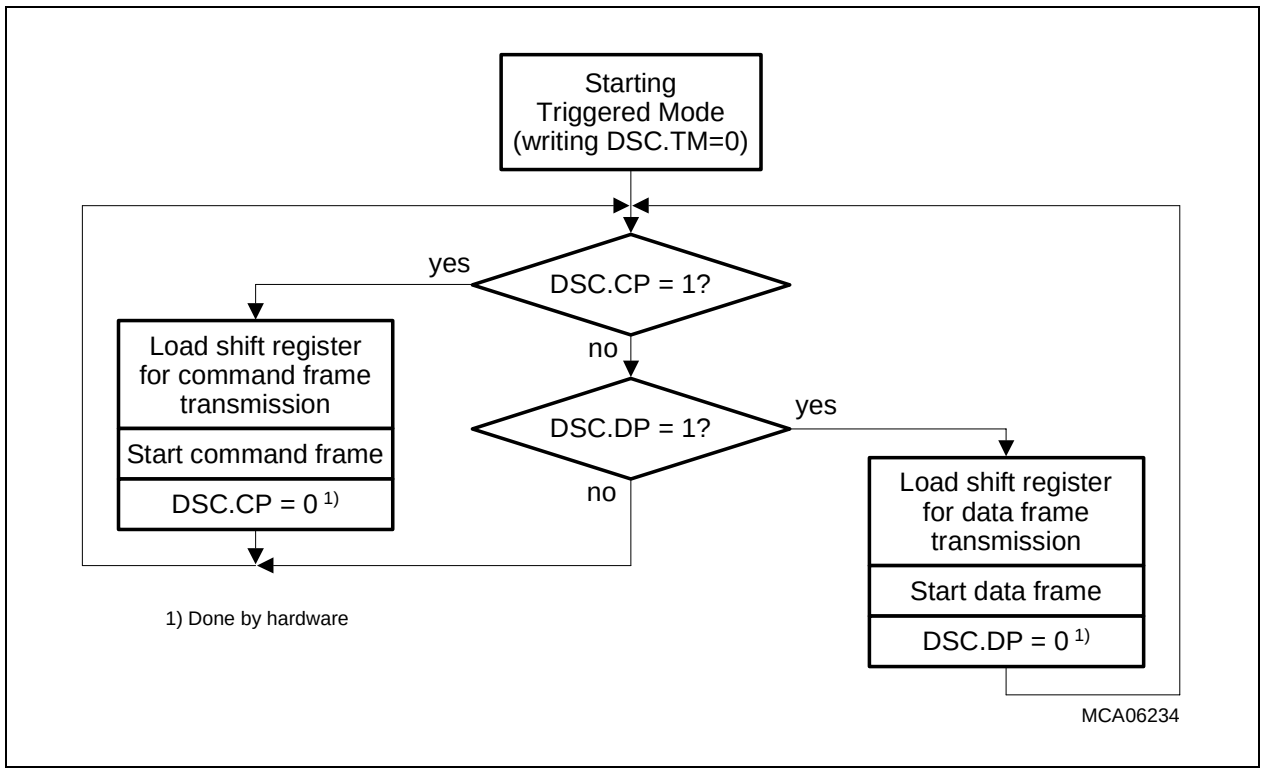


Figure 19-8 Triggered Mode Flow Diagram

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 (19.1)**). These parameters determine time reference points (TRP) at which a data or passive time frames is started (see diagram A in **Figure 19-9**).

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.

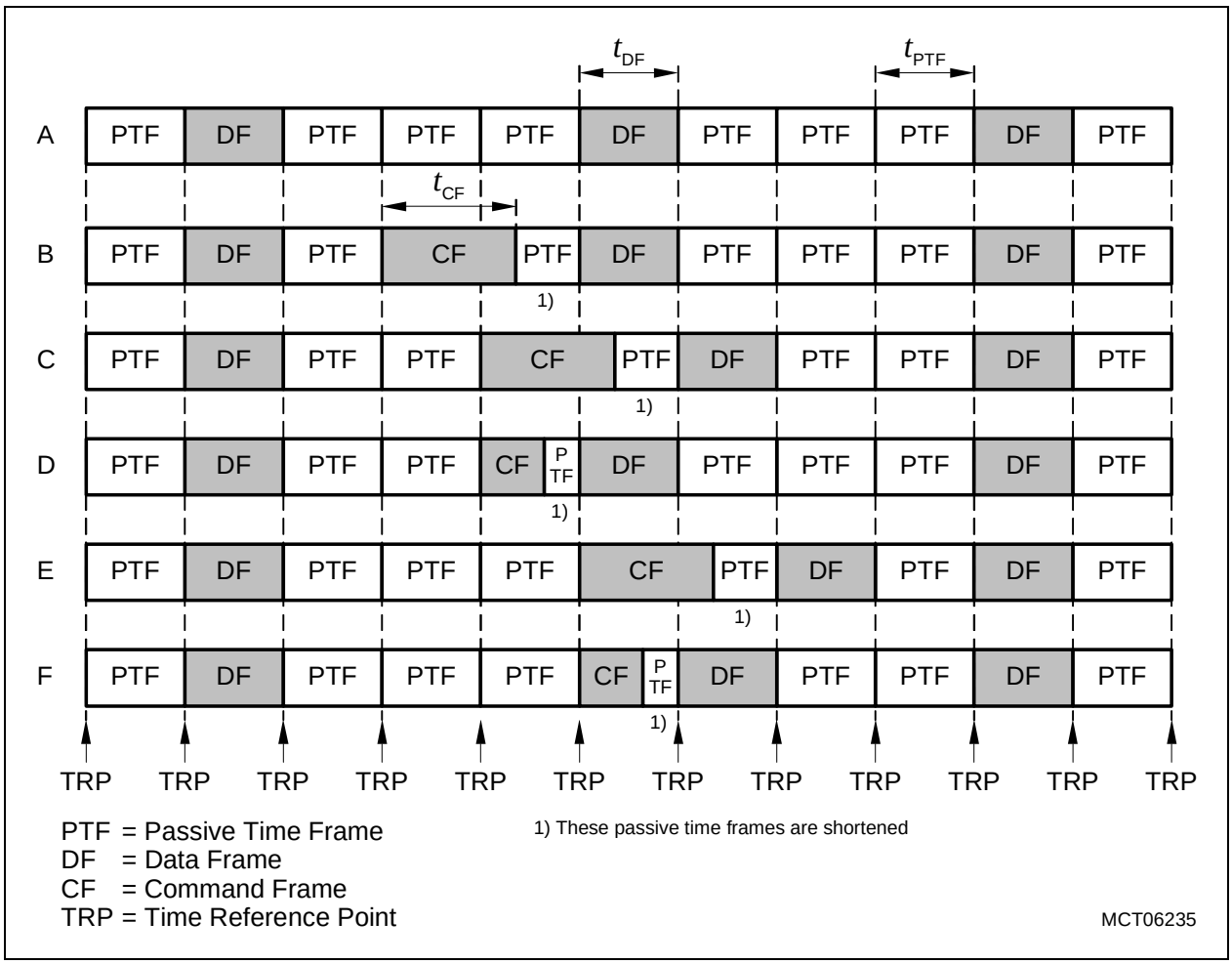


Figure 19-9 Data Repetition Mode Frame Examples with DSS.NPTF = 0011_B

Diagrams B to F in **Figure 19-9** 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 19-10 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.

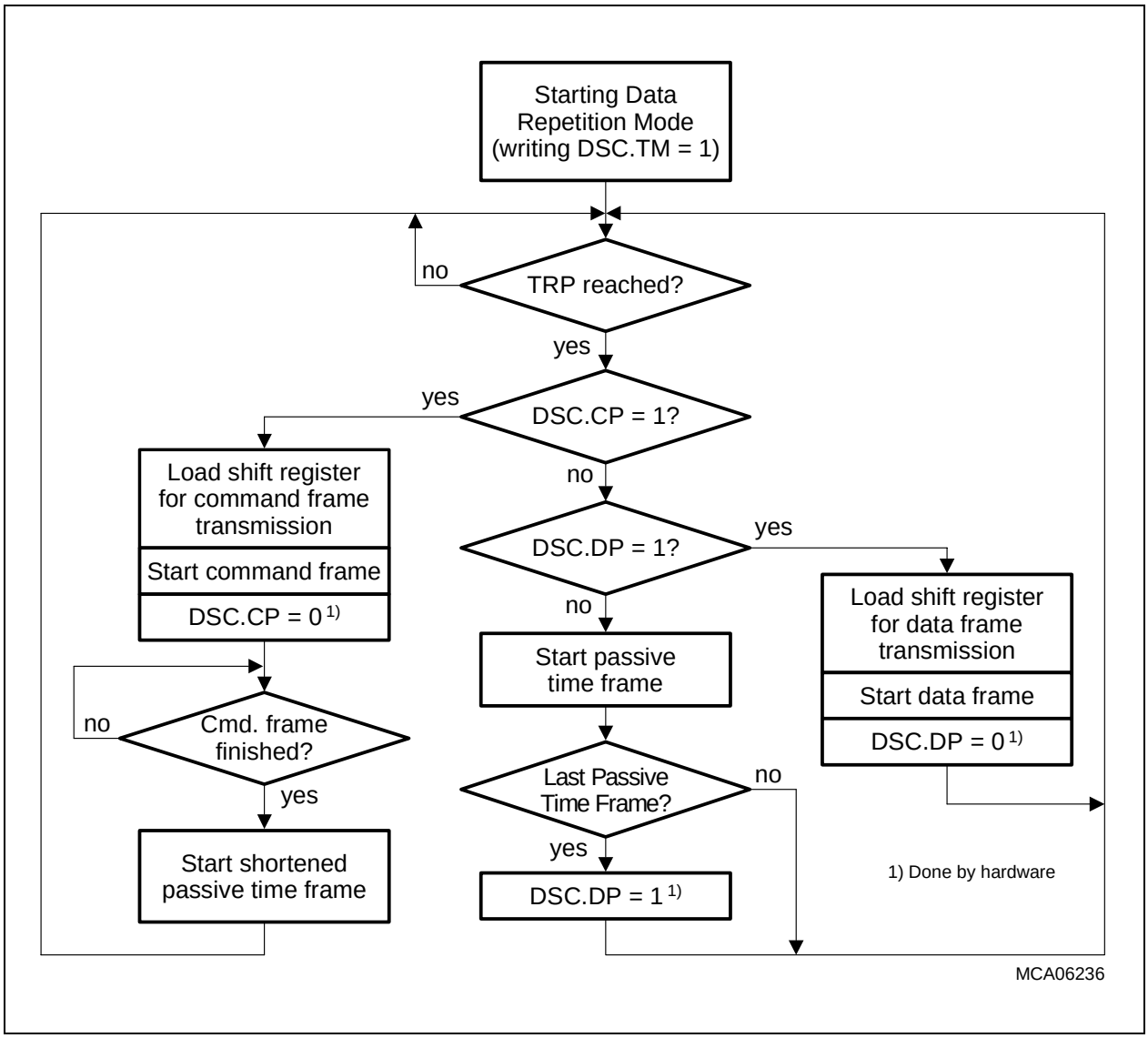


Figure 19-10 Data Repetition Mode Flow Diagram

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 19.1.2.2** 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.DC 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.

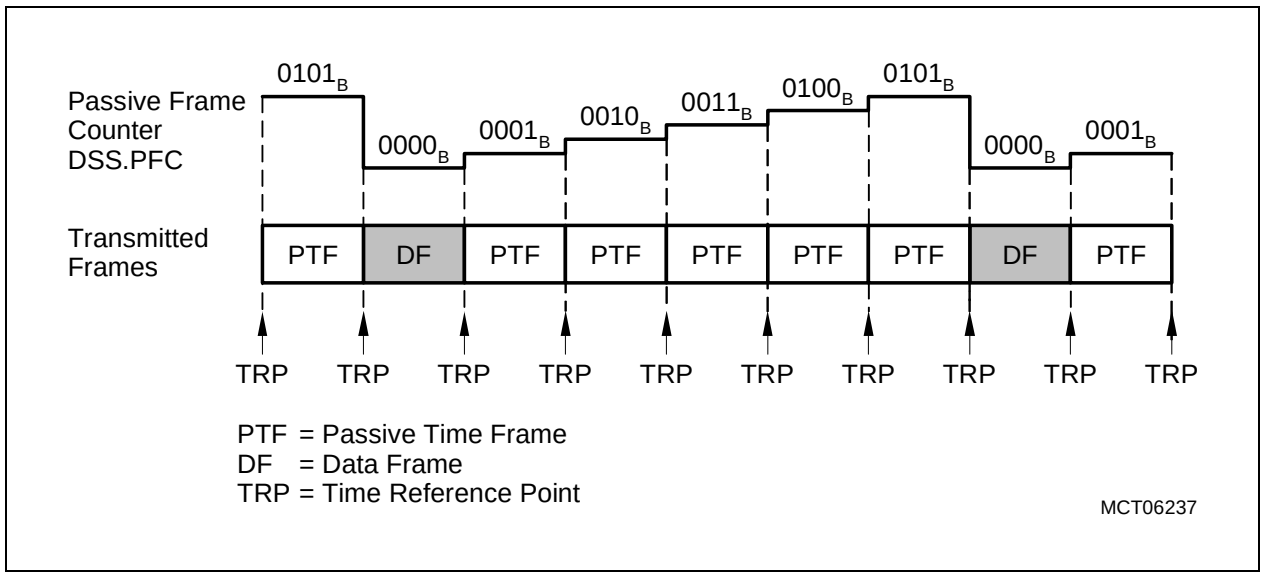


Figure 19-11 Passive Frame Counter Operation (with DSS.NPTF = 0101_B)

19.1.2.4 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 19-12 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 (ENSELL = 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.

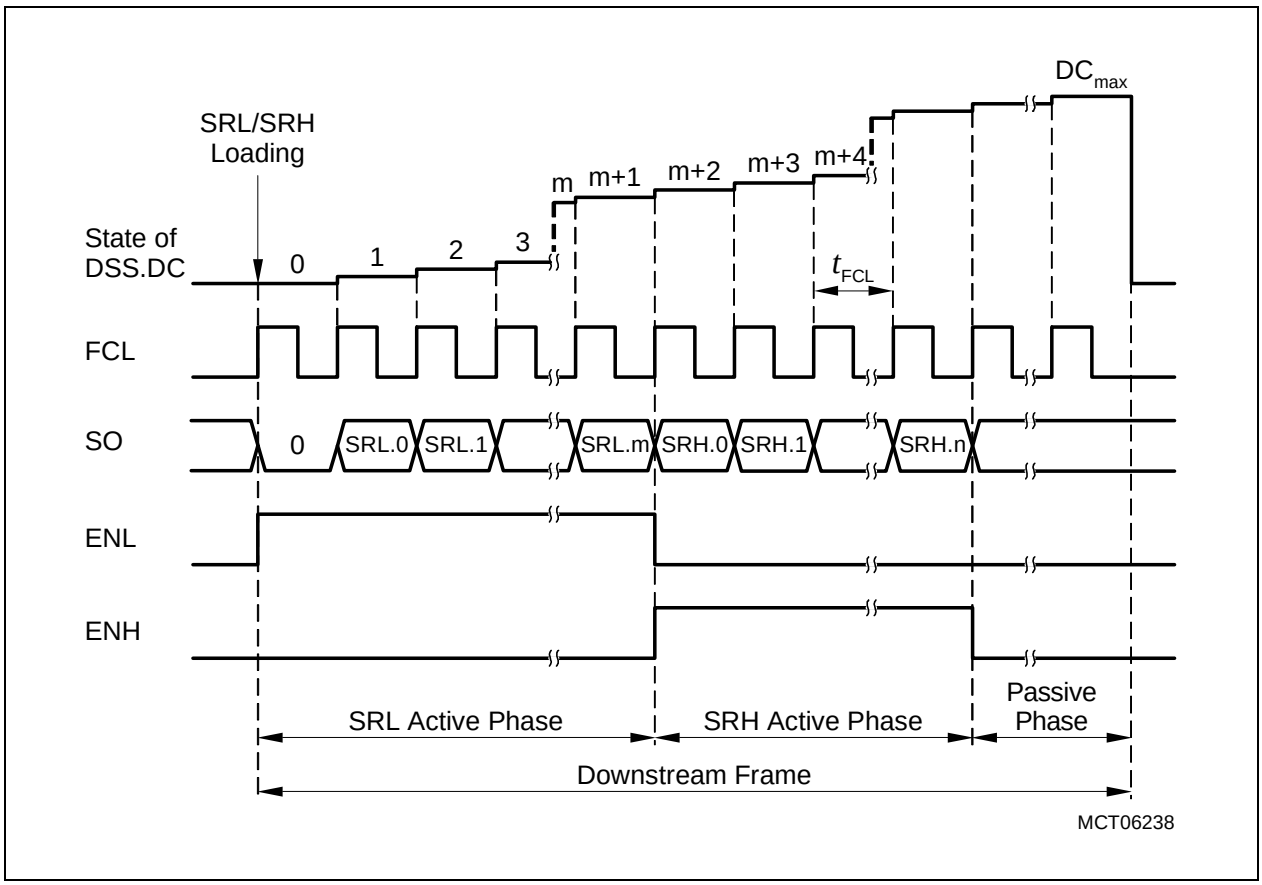


Figure 19-12 Shift Clock Counting: Data Frame with ENSELL = 1 and ENSELH = 0

When the selection bit for the SRL active frame is disabled (ENSELL = 0, see **Figure 19-13**), 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.

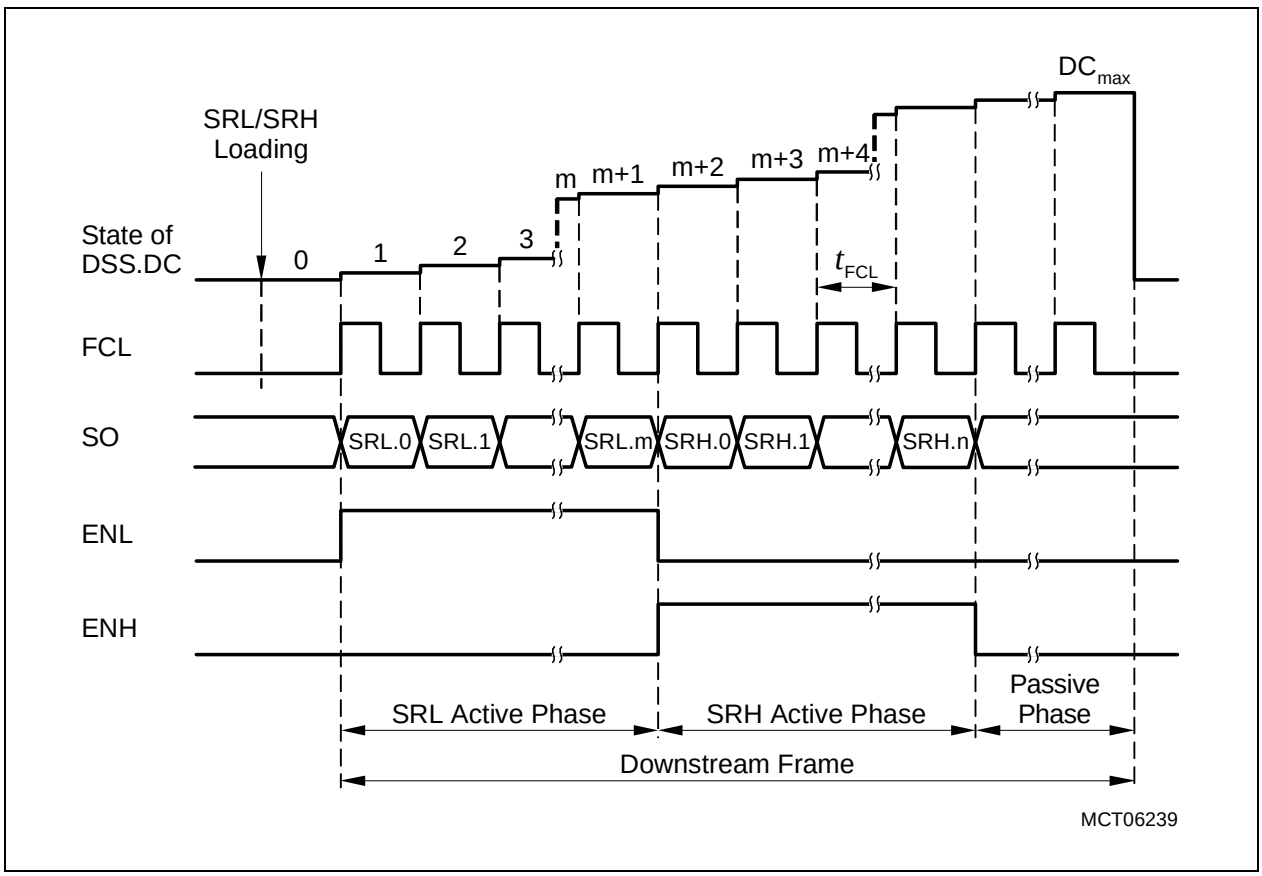


Figure 19-13 Shift Clock Counting: Data Frame with ENSELL = 0 and ENSELH = 0

19.1.2.5 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 TC1766 specific clock generation is described on **Page 19-65**.

19.1.2.6 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.

19.1.3 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 19-14 is a block diagram of the MSC upstream channel.

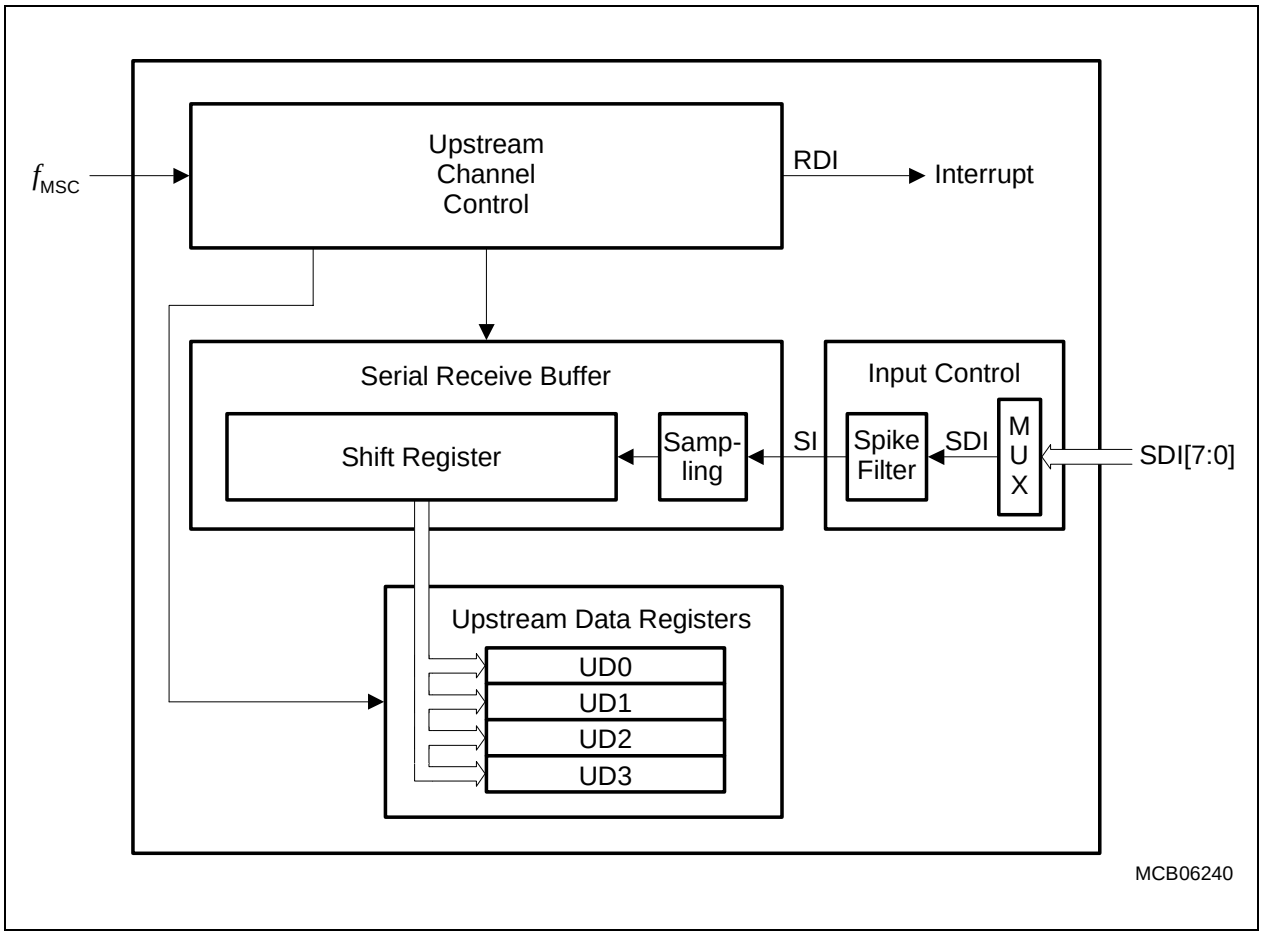


Figure 19-14 Upstream Channel Block Diagram

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 $f_{MSC}/256$.

19.1.3.1 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 19-15**, 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

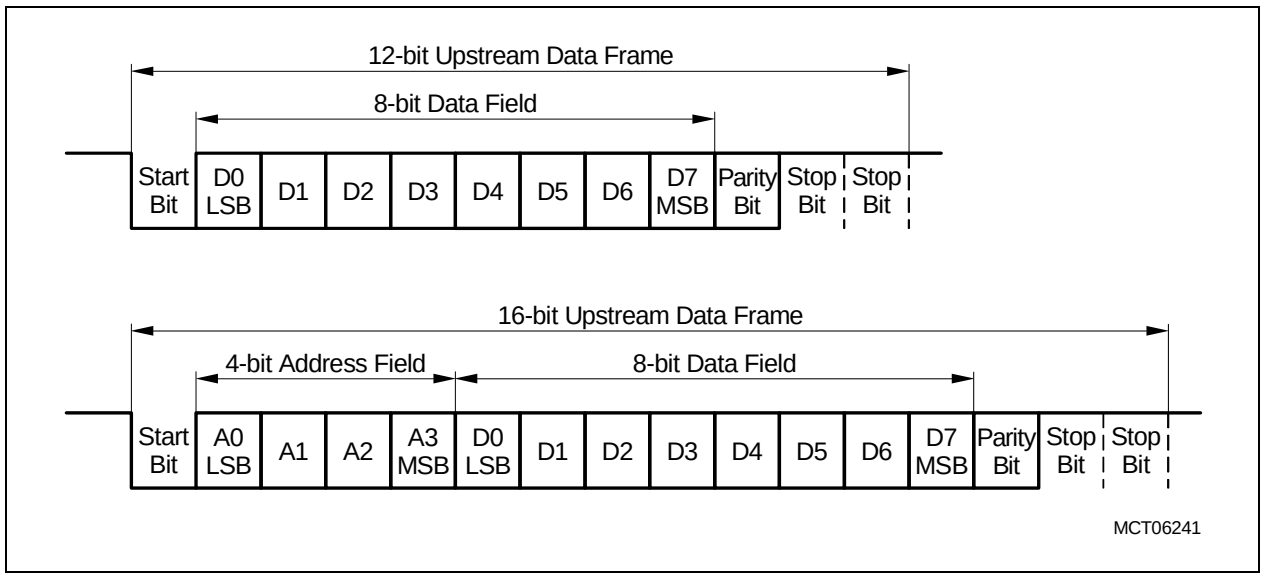


Figure 19-15 Upstream Channel Frame Types

19.1.3.2 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.

19.1.3.3 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 19-30****).*

Frame Reception with Address Field

Frame reception for a 16-bit data frame (see **Figure 19-16**) 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 19-16**, 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 19-25**).

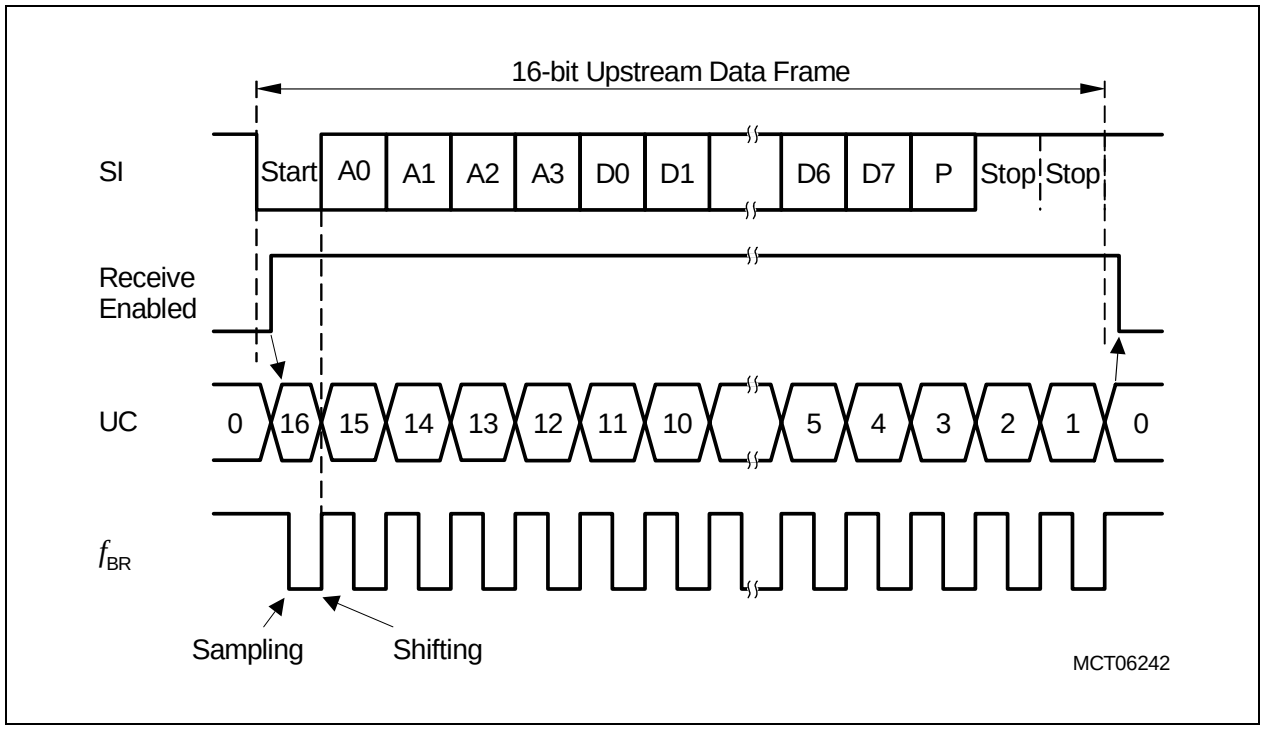


Figure 19-16 16-bit Upstream Reception

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.

19.1.3.4 Baud Rate

The baud rate of the upstream channel is derived from the MSC module clock f_{MSC}. **Figure 19-17** shows the configuration of the upstream channel clock circuitry.

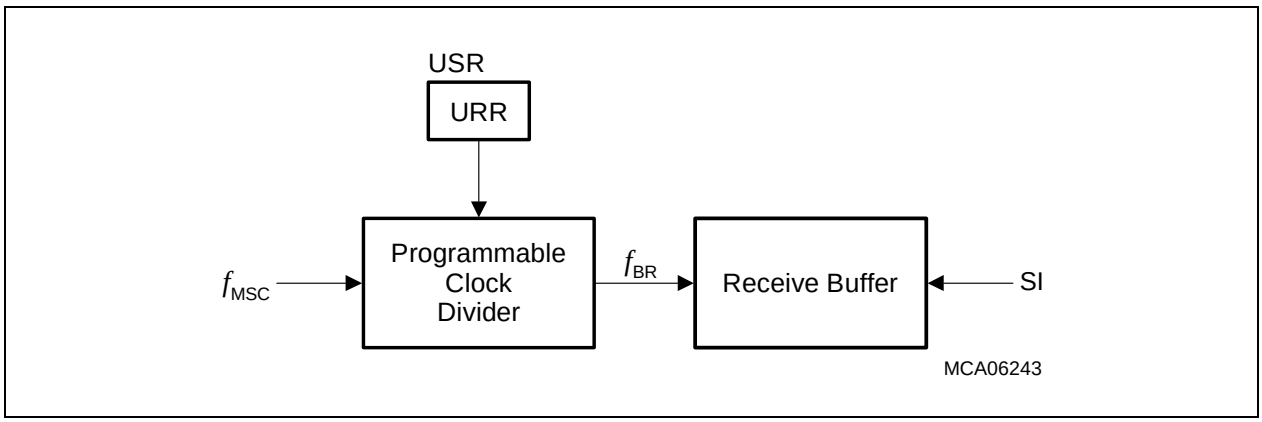


Figure 19-17 Upstream Channel Clock Circuitry

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 19-6**. The resulting baud rate formula is:

$$\text{Baud rate}_{\text{MSC Upstream Channel}} = \frac{f_{MSC}}{DF} \quad (19.2)$$

Table 19-6 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	$f_{MSC}/8$
010_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 19-18** shows the operation of the sampling counter at the beginning of an upstream frame with a divide factor DF of 8 (OCSR.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 19-6** 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.

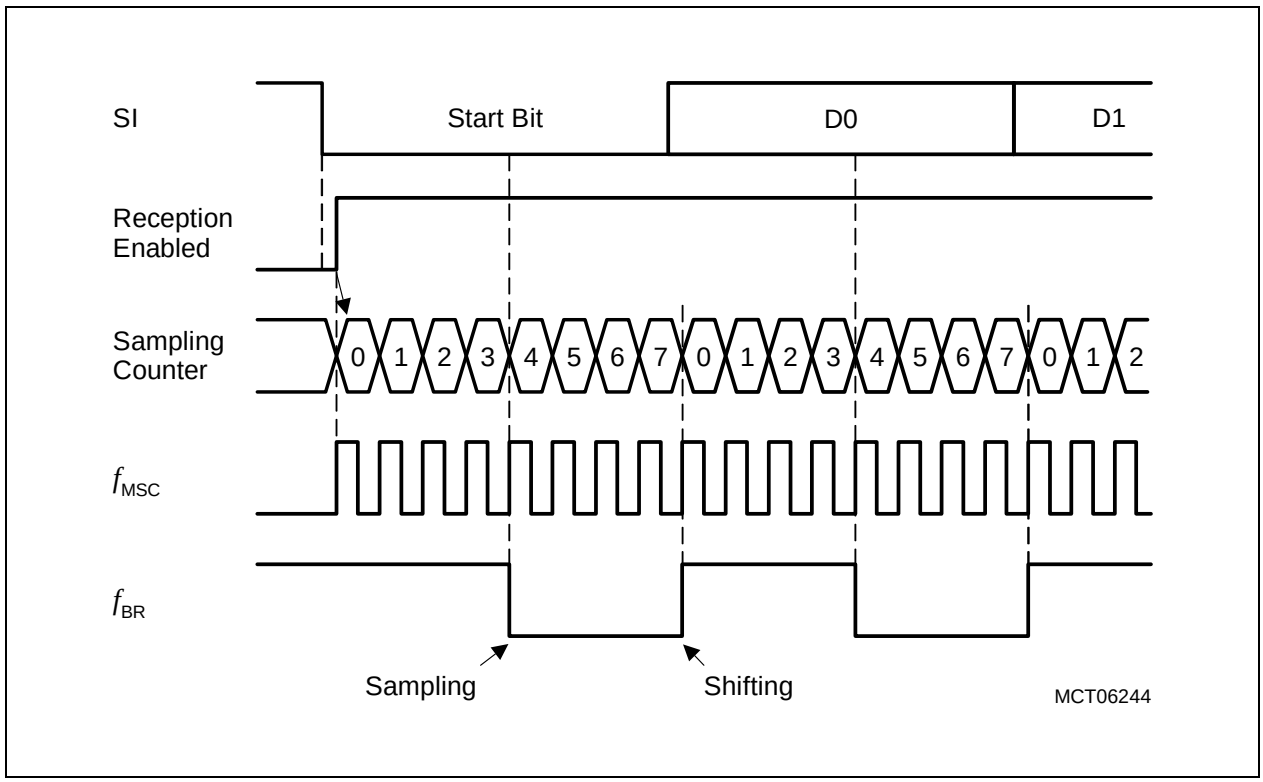


Figure 19-18 Upstream Channel Sampling with URR = 010_B

19.1.3.5 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.

19.1.4 I/O Control

The types of I/O control logic for the MSC module I/O lines are shown in **Figure 19-19**. 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.

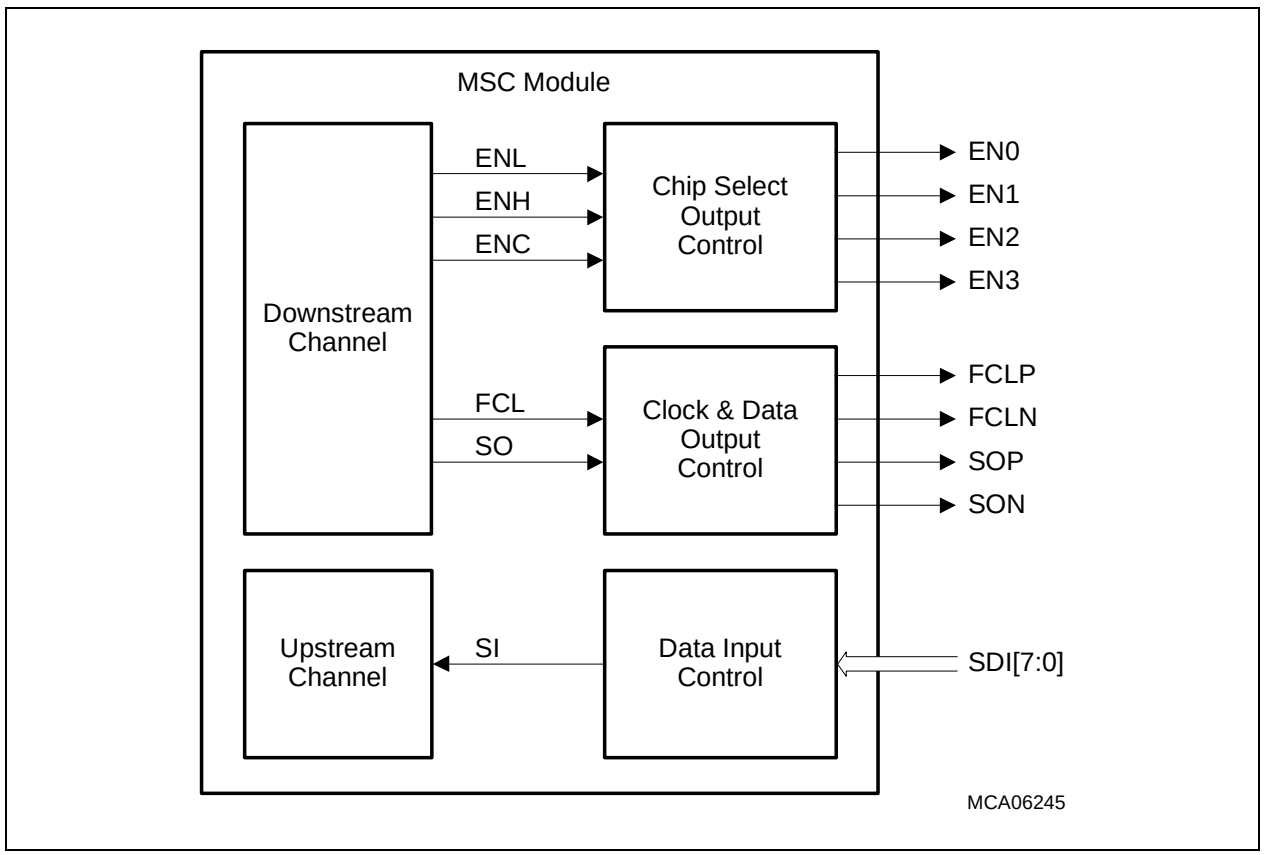


Figure 19-19 I/O Control

The MSC module I/O signals is controlled by bit fields that are located in the Output Control Register OCR.

19.1.4.1 Downstream Channel Output Control

As shown in **Figure 19-5** and **Figure 19-6**, 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 19-27**) 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.

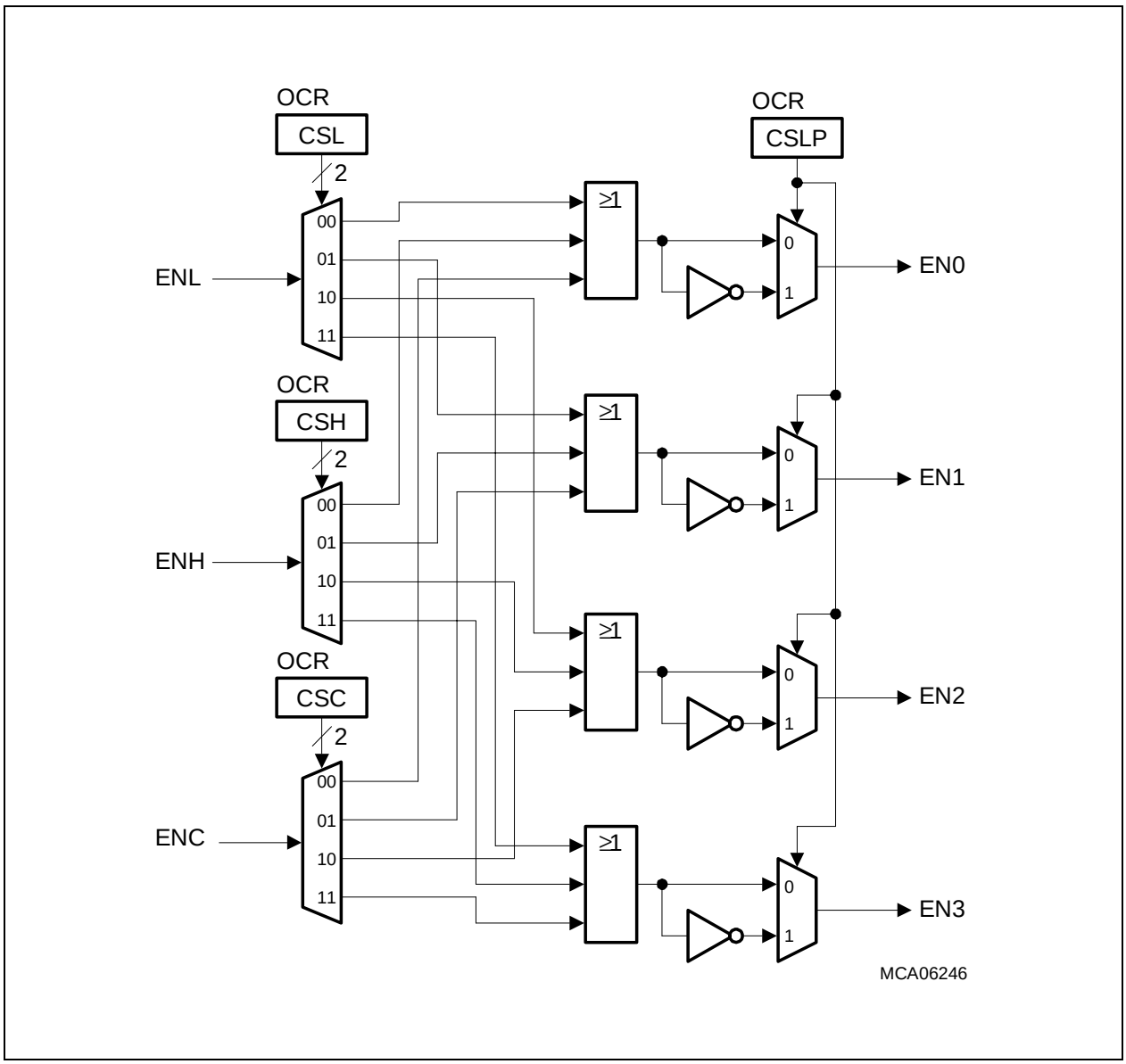


Figure 19-20 Downstream Channel: Chip Enable Output Control

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 19-21**.

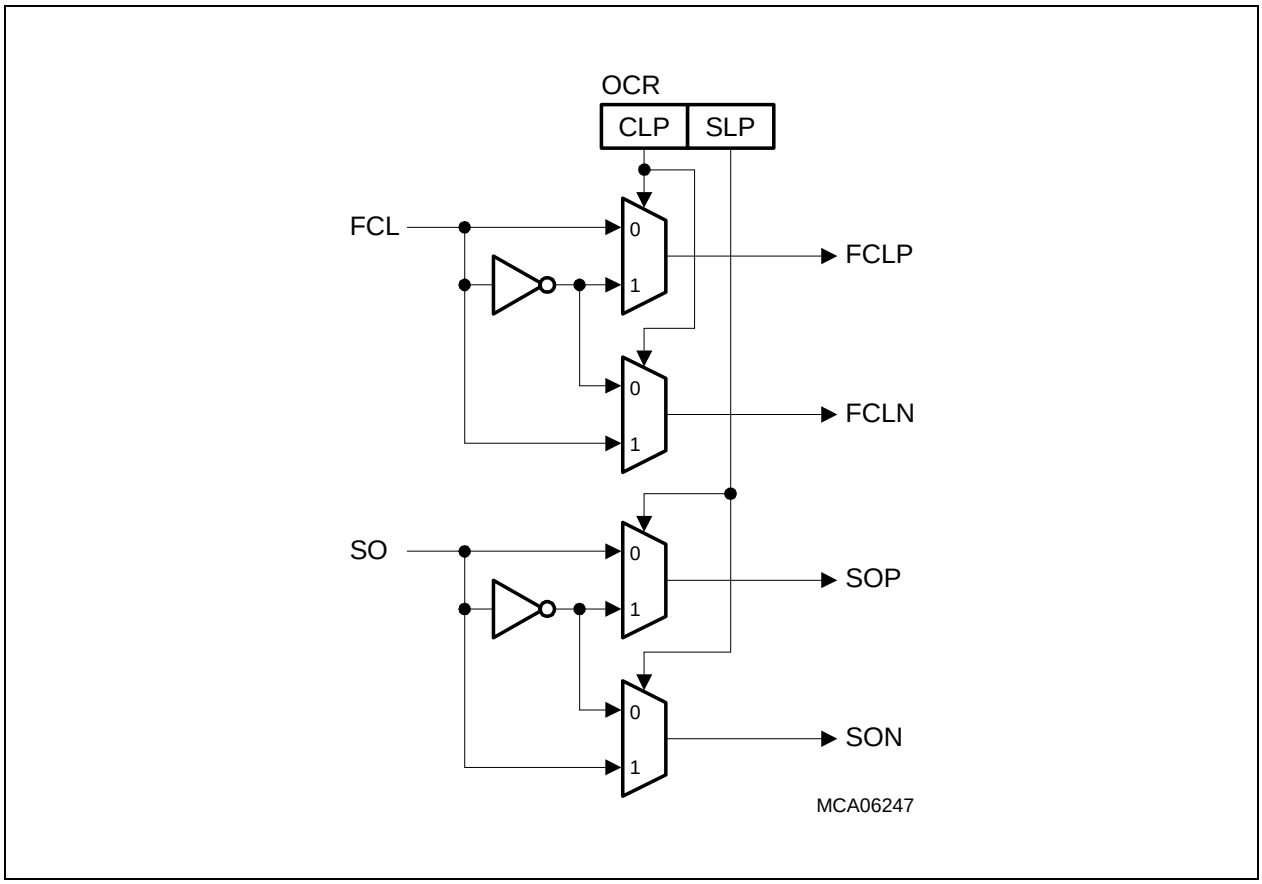


Figure 19-21 Downstream Channel: Clock and Data Output Control

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 19-21** 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).

19.1.4.2 Upstream Channel

As shown in **Figure 19-22**, 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.

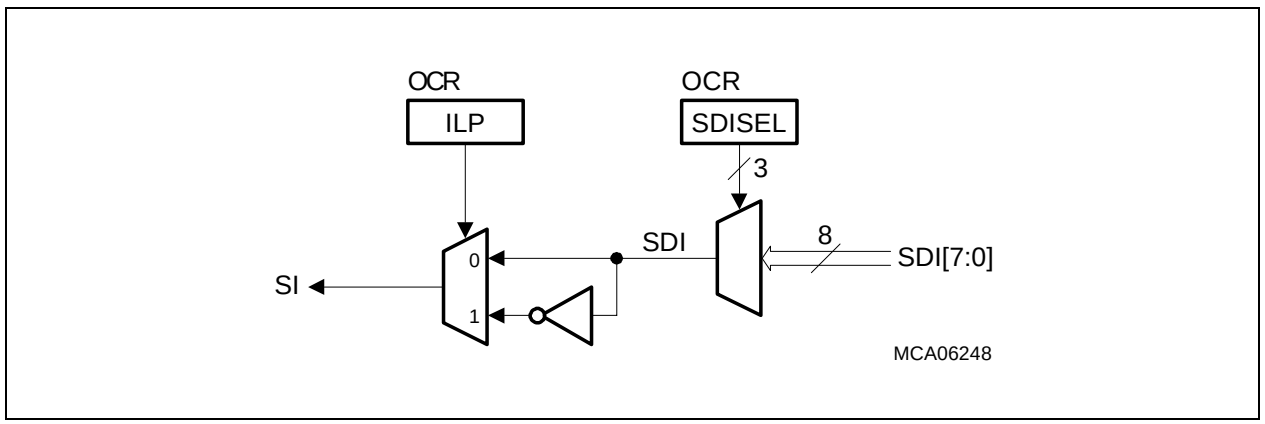


Figure 19-22 Upstream Channel Serial Data Input Control

19.1.5 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 (see also **Section 19.3.5.2** and **Section**). 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 19-7 shows the four MSC interrupt sources.

Table 19-7 MSC Interrupts

Interrupt Type	Generated by
Data frame interrupt	Downstream Channel
Command frame interrupt	
Time frame finished interrupt	
Receive data interrupt	Upstream Channel

19.1.5.1 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 19-6**). Bit ICR.EDIEI selects which case is selected.

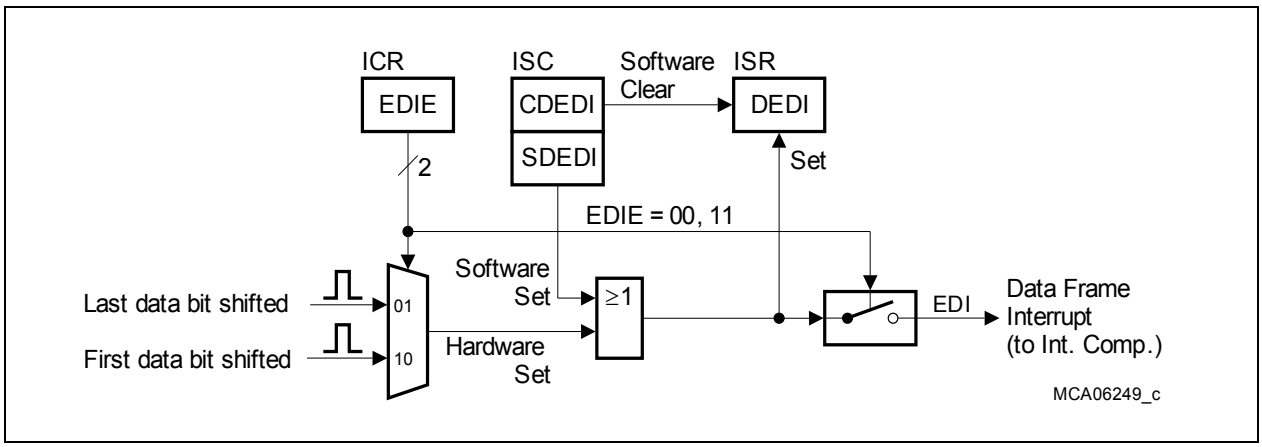


Figure 19-23 Data Frame Interrupt Control

19.1.5.2 Command Frame Interrupt

A command frame interrupt can be generated at the end of a downstream channel command frame (see also **Figure 19-5**).

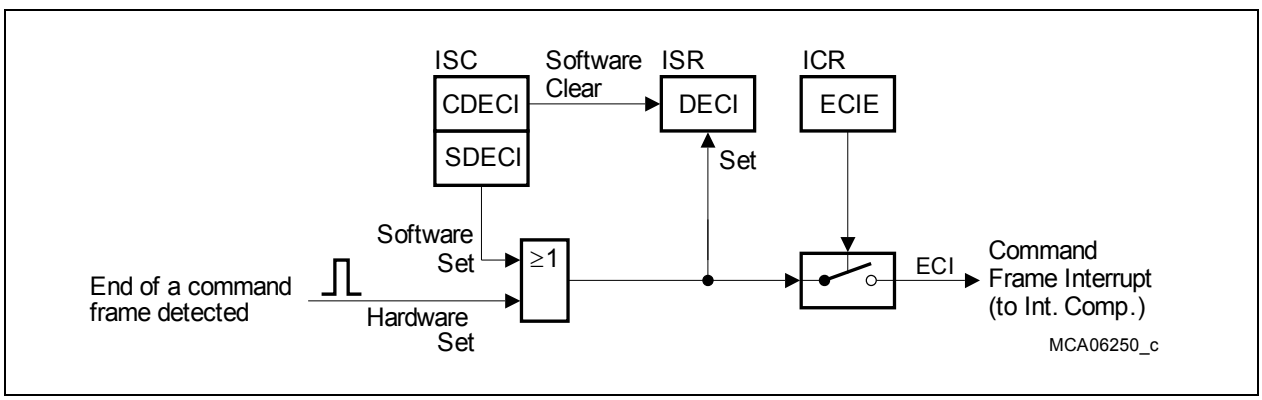


Figure 19-24 Command Frame Interrupt Control

19.1.5.3 Time Frame Finished Interrupt

A time frame finished interrupt can be generated at the end of a downstream channel passive time phase.

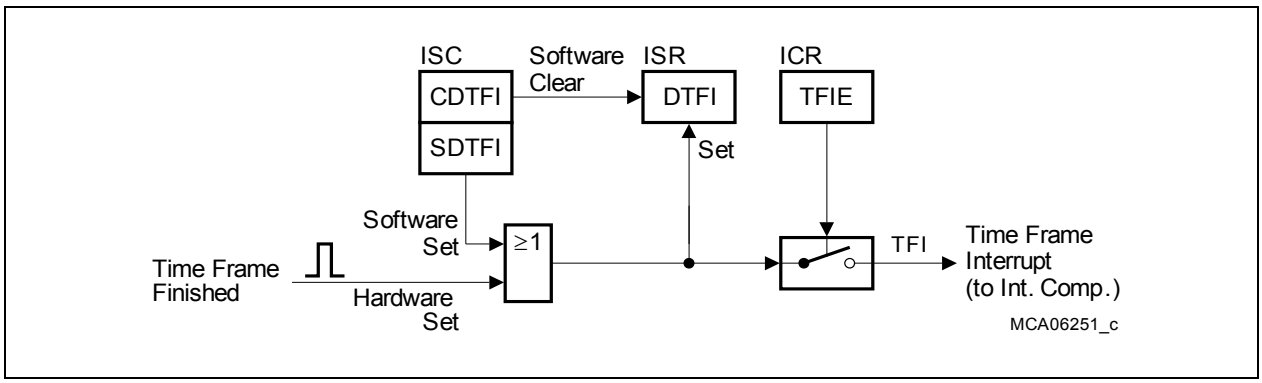


Figure 19-25 Time Frame Interrupt Control

19.1.5.4 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 cleared when writing bits ISC.SRDI or ISC.CRDI with a 1.

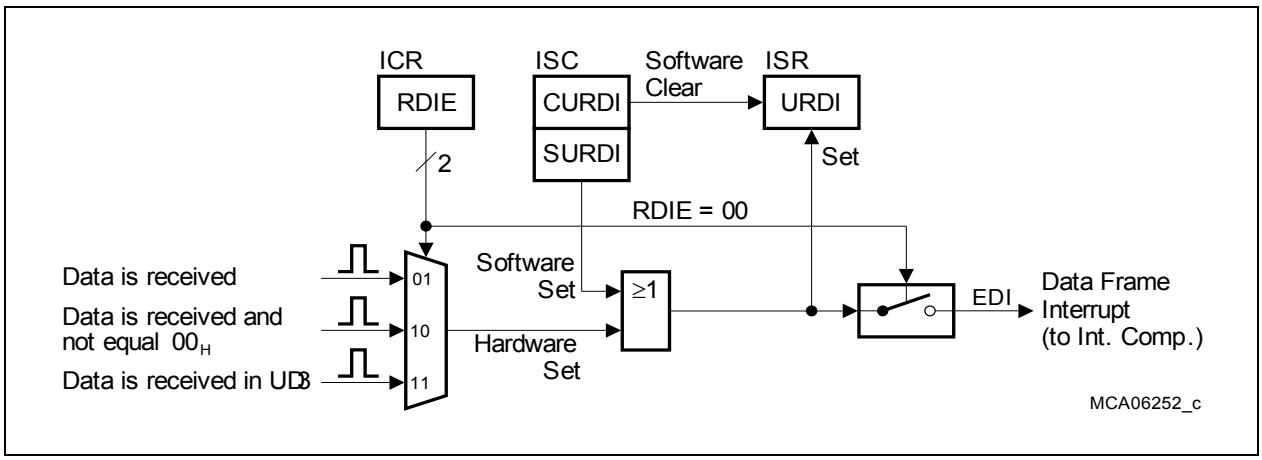


Figure 19-26 Receive Data Interrupt Control

19.1.5.5 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.

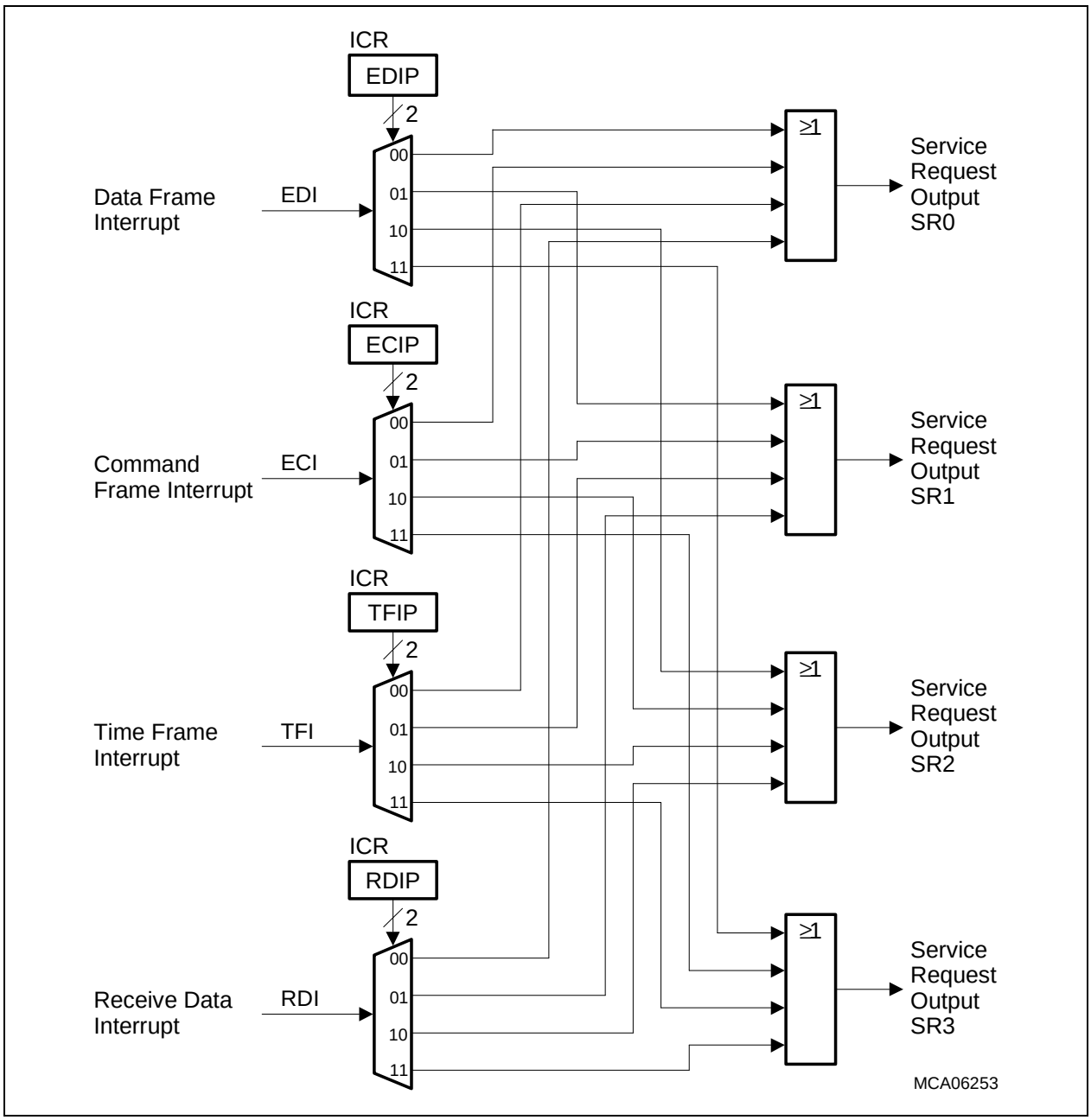


Figure 19-27 MSC Interrupt Request Compressor

*Note: The number of available MSC interrupt outputs depends on the implementation of the MSC module(s) in the specific product (see **Section** for TC1766 details).*

19.2 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 TC1766 User's Manual by the module name prefix "MSC0_" for the MSC0 interface.

MSC Kernel Register Overview

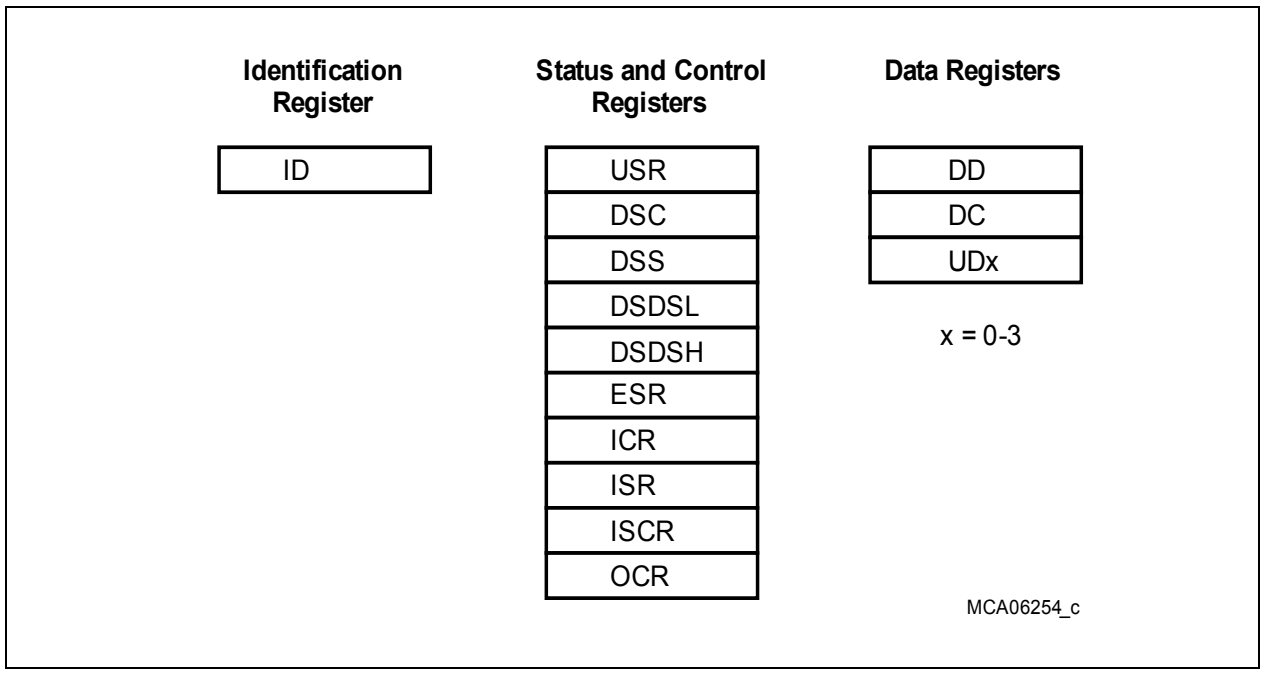


Figure 19-28 MSC Kernel Registers

The complete and detailed address map of the MSC0 module is described in **Table 16-7** on **Page 16-15** of the TC1766 User's Manual System Units part (Volume 1).

Table 19-8 Registers Address Space - MSC0 Kernel Registers

Module	Base Address	End Address	Note
MSC0	F000 0800$_H$	F000 08FF$_H$	-

Table 19-9 Registers Overview - MSC Kernel Registers

Register Short Name	Register Long Name	Offset Address[1)]	Description see
ID	Module Identification Register	08$_H$	**Page 19-38**
USR	Upstream Status Register	10$_H$	**Page 19-39**
DSC	Downstream Control Register	14$_H$	**Page 19-41**
DSS	Downstream Status Register	18$_H$	**Page 19-44**

Table 19-9 Registers Overview - MSC Kernel Registers (cont'd)

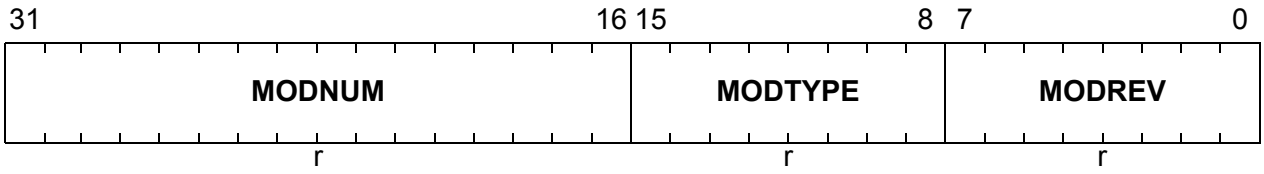
Register Short Name	Register Long Name	Offset Address[1)]	Description see
DD	Downstream Data Register	$1C_H$	Page 19-59
DC	Downstream Command Register	20_H	Page 19-59
DSDSL	Downstream Select Data Source Low Register	24_H	Page 19-46
DSDSH	Downstream Select Data Source High Register	28_H	Page 19-47
ESR	Emergency Stop Register	$2C_H$	Page 19-48
UD0	Upstream Data Register 0	30_H	Page 19-60
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 19-49
ISR	Interrupt Status Register	44_H	Page 19-52
ISC	Interrupt Set Clear Register	48_H	Page 19-54
OCR	Output Control Register	$4C_H$	Page 19-56

1) The absolute register address is calculated as follows:
Module Base Address (Table 19-8) + Offset Address (shown in this column)

19.2.1 MSC Module Identification Register

The MSC Module Identification Register ID contains read-only information about the MSC module version.

ID
Module Identification Register (008_H) Reset Value: $0028\ C0XX_H$

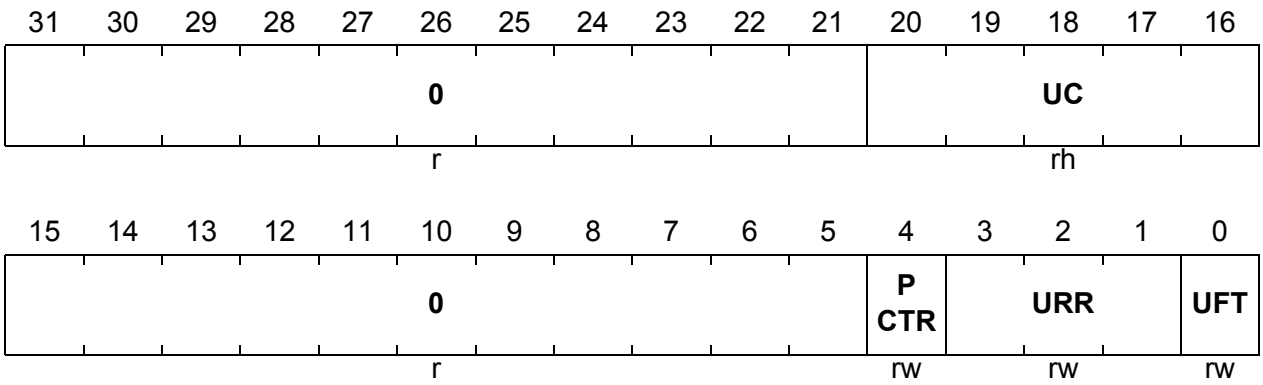


Field	Bits	Type	Description
MODREV	[7:0]	r	**Module Revision Number** MODREV defines the module revision number. The value of a module revision starts with 01_H (first revision).
MODTYPE	[15:8]	r	**Module Type** This bit field defines the module as a 32-bit module: $C0_H$
MODNUM	[31:16]	r	**Module Number Value** This bit field defines the module identification number for the MSC: 0028_H

19.2.2 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$_H$

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DS DIS	EN SEL H	EN SEL L	NDBH					NDBL					DP	CP	TM
rh	rw	rw	rw					rw					rh	rh	rw

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.

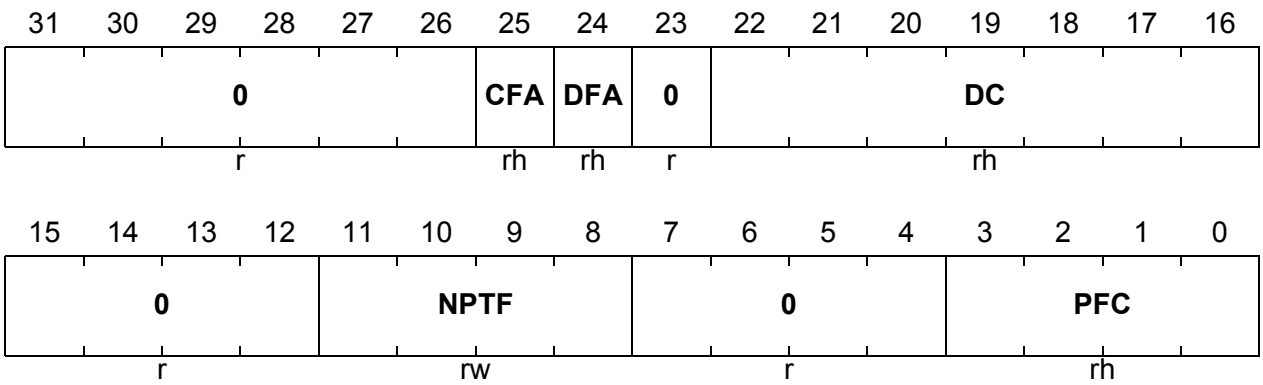
Field	Bits	Type	Description
NDBL	[7:3]	rw	**Number of SRL Bits Shifted at Data Frames** NDBL determines the number of shift register low part (SRL) bits that are shifted out on SO during a data frame. 00000_B No SRL bit shifted 00001_B SRL[0] shifted 00010_B SRL[1:0] shifted …$_B$ … 01111_B SRL[14:0] 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: $0000\ 0000_H$

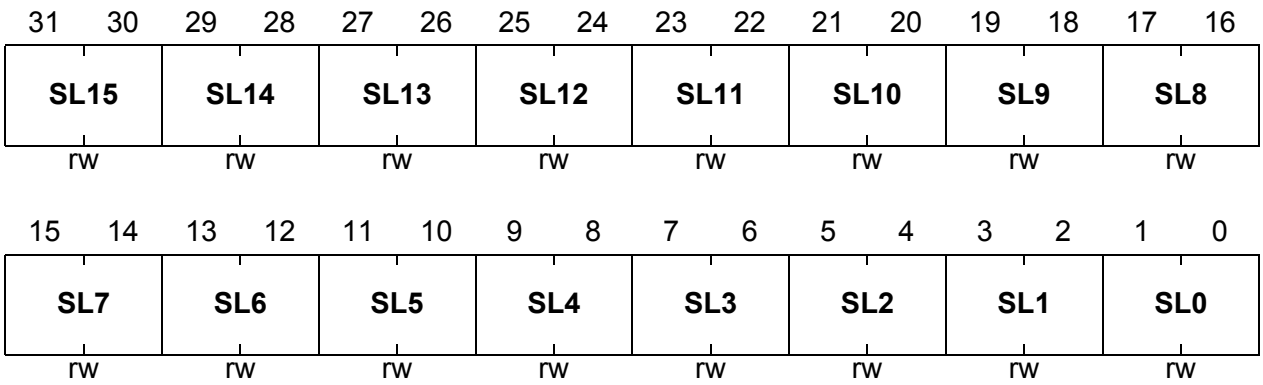


Field	Bits	Type	Description
PFC	[3:0]	rh	**Passive Time Frame Counter** In Data Repetition Mode, this bit field indicates the count of passive time frames that are currently transmitted. In Triggered Mode PFC remains at 0000_B. 0000_B Data frame is transmitted. 0001_B First passive time frame is transmitted. 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. $\ldots_B$ … $7F_H$ 127 shift clocks elapsed. The 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.

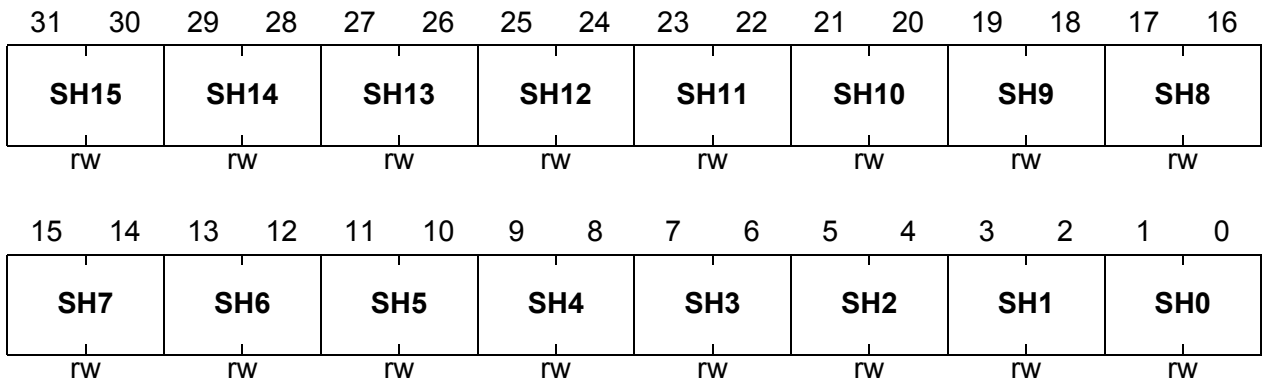
DSDSL
Downstream Select Data Source Low Register
(24_H) **Reset Value: $0000\ 0000_H$**



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 DSDSL determine the data source for each bit in shift register SRL.

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 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 ALTINH input line x. 11_B SRL[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$_H$

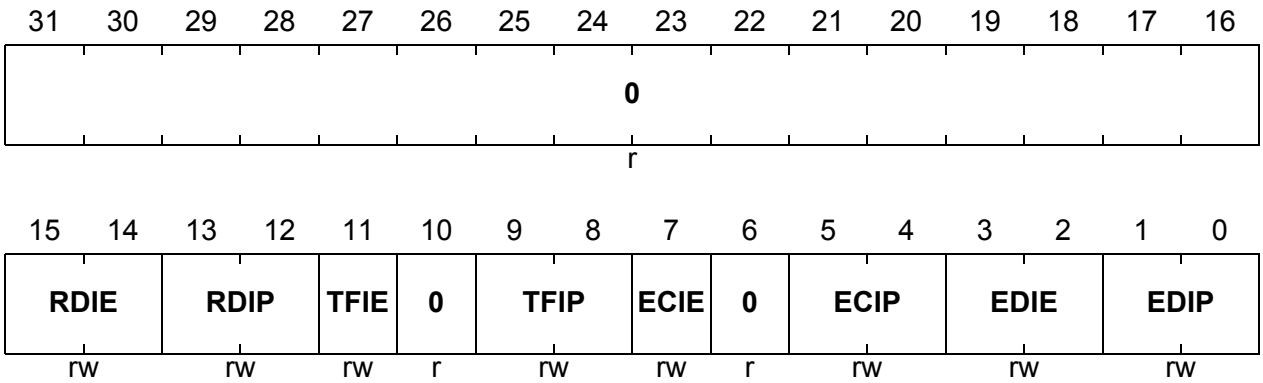
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ENH 15	**ENH 14**	**ENH 13**	**ENH 12**	**ENH 11**	**ENH 10**	**ENH 9**	**ENH 8**	**ENH 7**	**ENH 6**	**ENH 5**	**ENH 4**	**ENH 3**	**ENH 2**	**ENH 1**	**ENH 0**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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: $0000\ 0000_H$**



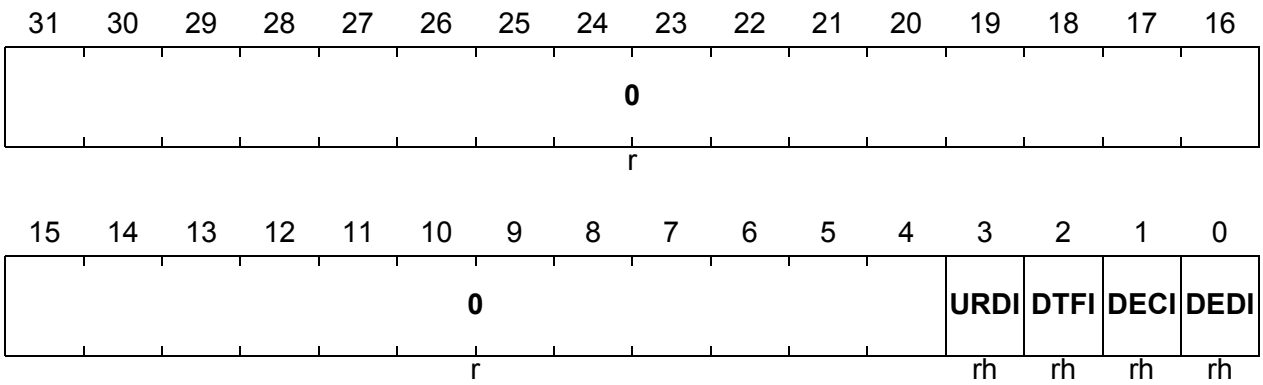
Field	Bits	Type	Description
EDIP	[1:0]	rw	**Data Frame Interrupt Node Pointer** EDIP selects the service request output line SRn (n = 3-0) for the data frame interrupt. 00_B Service request output SR0 selected 01_B Service request output SR1 selected 10_B Service request output SR2 selected 11_B Service 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 = 3-0) 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 ISR 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

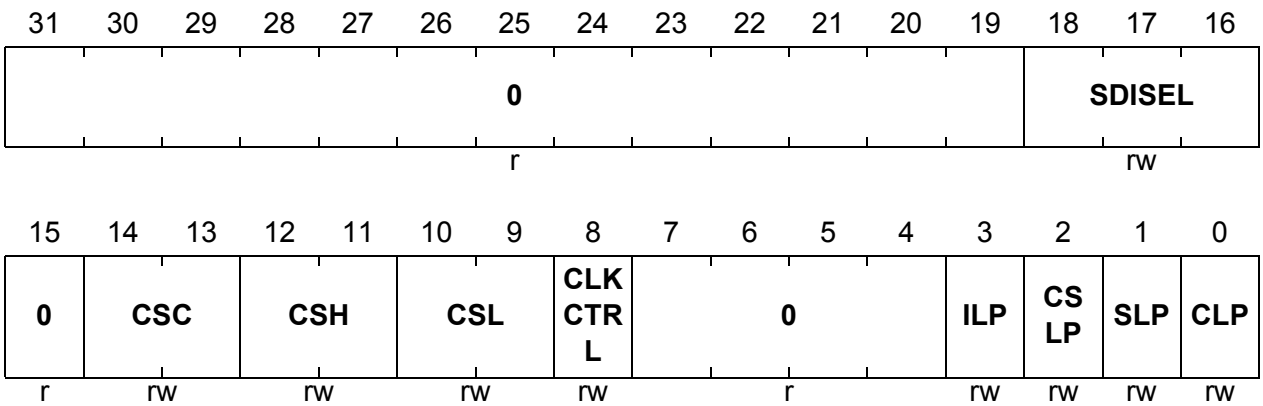
Field	Bits	Type	Description
CDEDI	0	w	**Clear DEDI Flag** 0_B No operation. 1_B Bit ISR.DEDI is cleared.
CDECI	1	w	**Clear DECI Flag** 0_B No operation. 1_B Bit ISR.DECI is cleared.
CDTFI	2	w	**Clear DTFI Flag** 0_B No operation. 1_B Bit ISR.DTFI is cleared.
CURDI	3	w	**Clear URDI Flag** 0_B No operation. 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 (clear and set bits) 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.

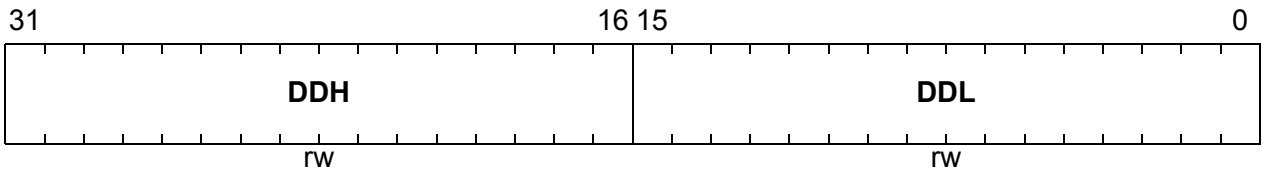
Field	Bits	Type	Description
CLKCTRL	8	rw	**Clock Control** This bit determines the activation of clock output FCL. 0_B FCL is activated only during the active phases of data or command frames (not during passive time frames). 1_B FCL is always active whether or not a downstream frame is currently transmitted.
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 SRL 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 as SDI. 001_B SDI[1] input is selected as SDI. 010_B SDI[2] input is selected as SDI. 011_B SDI[3] input is selected as SDI. 100_B SDI[4] input is selected as SDI. 101_B SDI[5] input is selected as SDI. 110_B SDI[6] input is selected as SDI. 111_B SDI[7] input is selected as SDI.
0	[7:4], [31:19]	r	**Reserved** Read as 0; should be written with 0.

19.2.3 Data Registers

The Downstream Data Register DD contains data to be transmitted during data frames.

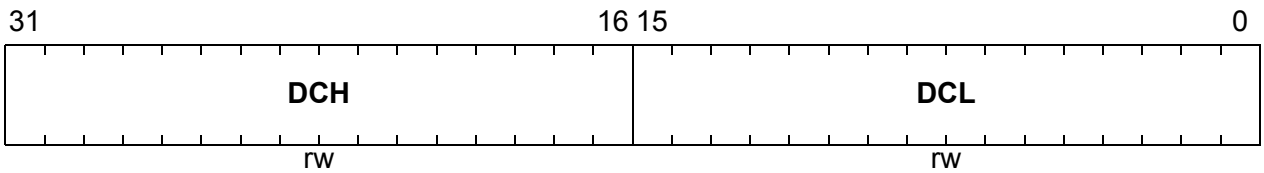
DD
Downstream Data Register **($1C_H$)** **Reset Value: $0000\ 0000_H$**



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.

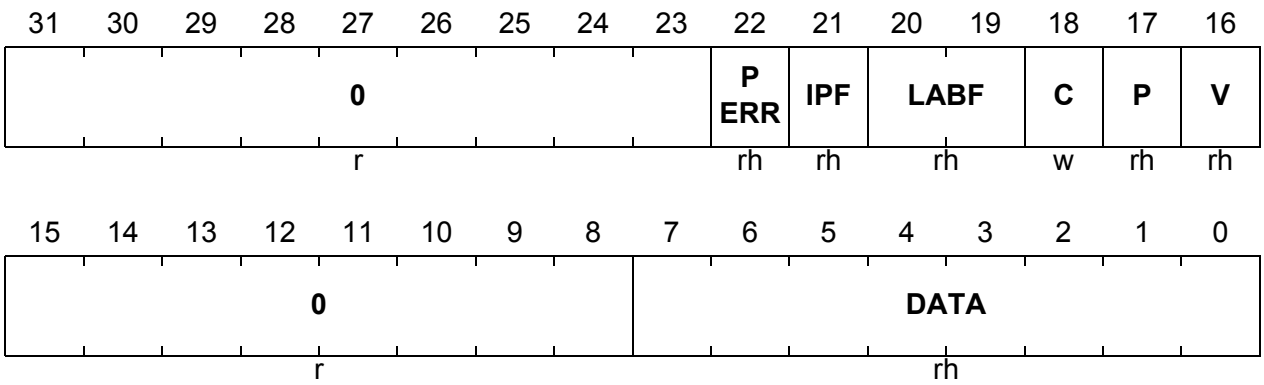
DC
Downstream Command Register **(20_H)** **Reset Value: $0000\ 0000_H$**



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 DDx 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. Bit 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.

19.3 MSC Module Implementation

This section describes the MSC module interfaces as they are implemented in the TC1766. It especially covers clock control, port and on-chip connections, interrupt control, and address decoding.

19.3.1 Interface Connections of the MSC Module

Figure 19-29 shows the TC1766-specific implementation details and interconnections of the MSC0 module.

The MSC0 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 MSC0 module.

The serial data and clock outputs of the downstream channels of the MSC0 module are connected to dedicated LVDS differential output drivers. After a reset operation, all LVDS outputs are disabled and in power-down mode. They must be enabled before usage by setting SCU_CON.LDEN = 1.

Additionally, the positive serial clock (FCLP) and positive data output (SOP) are available at GPIO lines of Port 2 and also as dedicated lines. The device select outputs (EN0x) are wired to GPIO lines of Port 2. One Port 2 input line is connected to the upstream channel serial data input.

The MSC0 module's downstream channel serial data input is connected to one port line of Port 2.

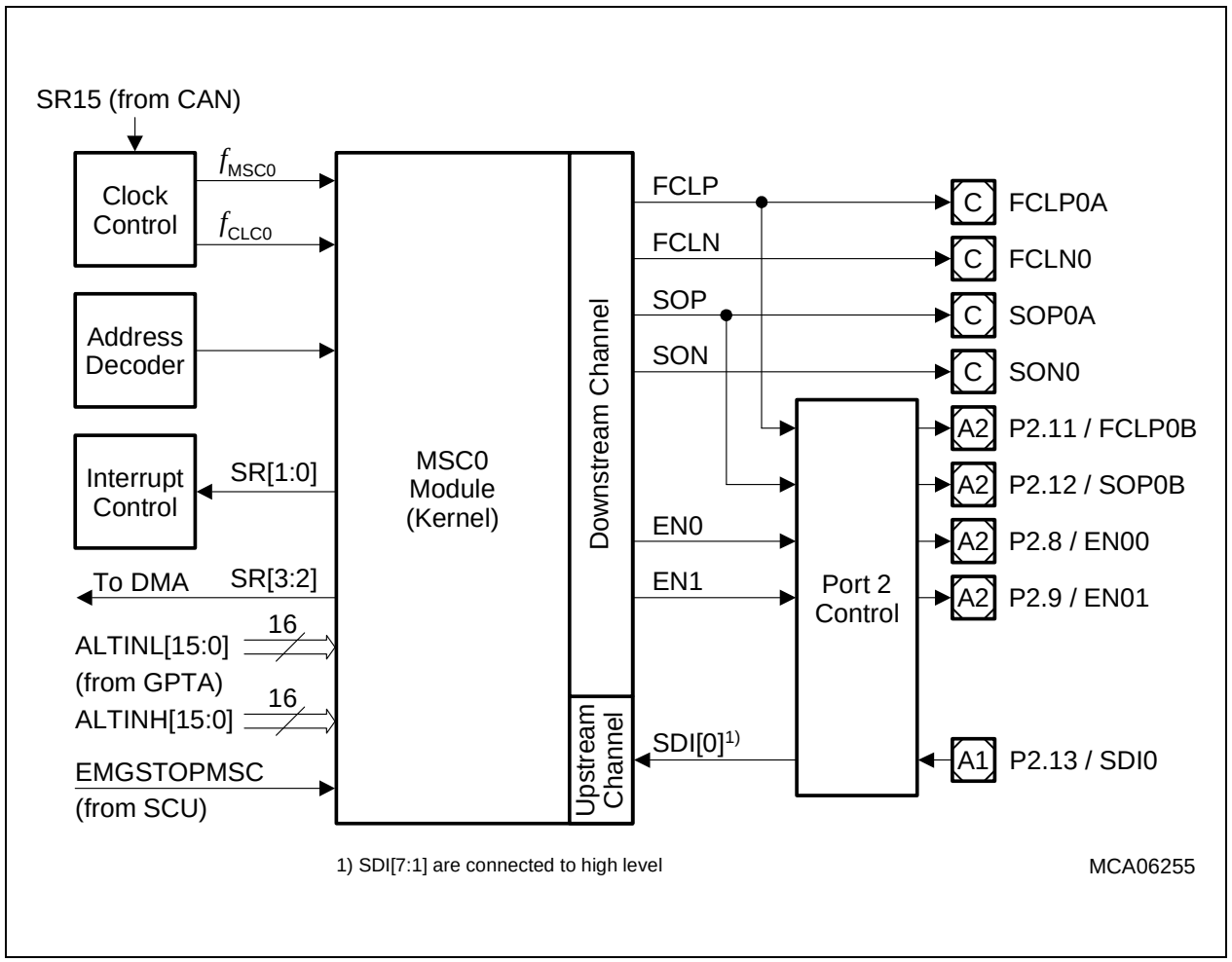


Figure 19-29 MSC0 Module Implementation and Interconnections

19.3.2 MSC0 Module-Related External Registers

Figure 19-30 summarizes the module-related external registers which are required for MSC programming (see also **Figure 19-28** for the module kernel specific registers). These registers are described in the following sections.

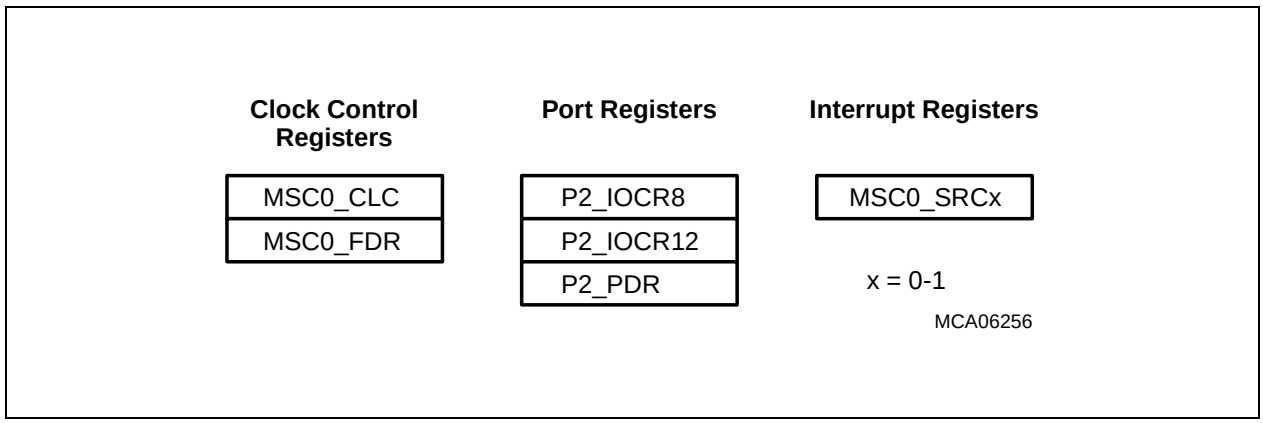


Figure 19-30 MSC Implementation-specific Special Function Registers

19.3.3 Clock Control

The MSC0 module is provided with two independent clock signals (**Figure 19-31**):

- f_{CLC0}
  This is the module clock that is used inside the MSC kernel for control purposes such as clocking of control logic and register operations. The frequency of f_{CLC0} is always identical to the system clock frequency f_{SYS}. The clock control register MSC0_CLC makes it possible to enable/disable f_{CLC0} under certain conditions.
- f_{MSC0}
  This clock is the module clock that is used inside the MSC for baud rate generation of the serial upstream and downstream channel. The fractional divider register MSC0_FDR controls the frequency of f_{MSC0} and makes it possible to enable/disable it independent of f_{CLC0}.

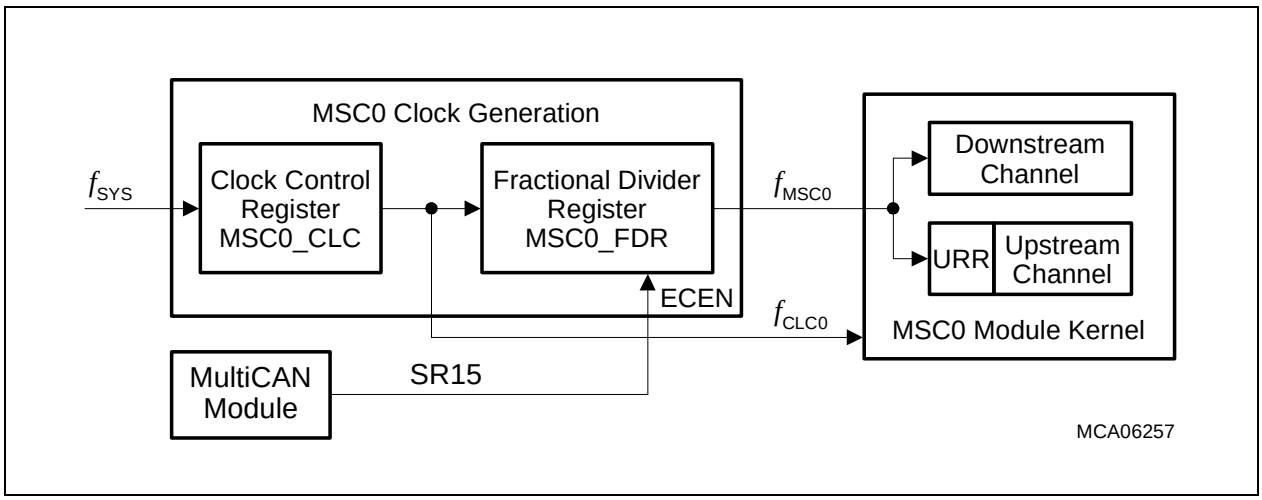


Figure 19-31 MSC Module Clock Generation

The following two formulas define the frequency of f_{MSC0}:

$$f_{MSC0} = f_{SYS} \times \frac{1}{n} \quad \text{with n = 1024 - MSC0.FDR.STEP} \tag{19.3}$$

$$f_{MSC0} = f_{SYS} \times \frac{n}{1024} \quad \text{with n = 0-1023} \tag{19.4}$$

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:

$$\text{Baud rate}_{MSC0} = f_{SYS} \times \frac{1}{2 \times (1024 - \text{MSC0.FDR.STEP})} \quad (19.5)$$

$$\text{Baud rate}_{MSC0} = f_{SYS} \times \frac{\text{MSC0.FDR.STEP}}{2 \times 1024} \quad (19.6)$$

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 (19.2)** on **Page 19-25**). The divide factor DF can be at minimum 4 and at maximum 256.

$$\text{Baud rate}_{MSC0} = f_{SYS} \times \frac{1}{DF \times (1024 - \text{MSC0.FDR.STEP})} \quad (19.7)$$

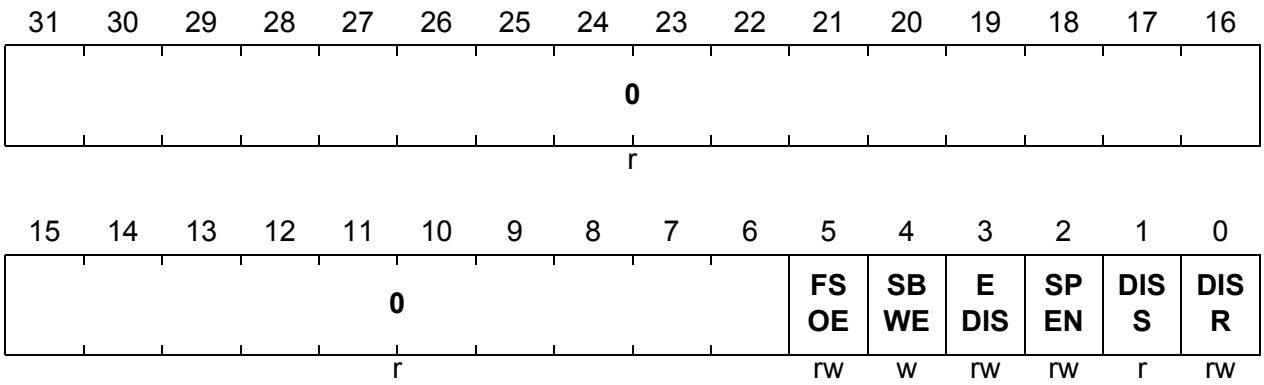
$$\text{Baud rate}_{MSC0} = f_{SYS} \times \frac{\text{MSC0.FDR.STEP}}{DF \times 1024} \quad (19.8)$$

Equation (19.3), **Equation (19.5)**, and **Equation (19.7)** are valid for normal divider mode (MSC0.FDR.DM = 01_B). **Equation (19.4)**, **Equation (19.6)**, and **Equation (19.8)** are valid for fractional divider mode (MSC0.FDR.DM = 10_B).

19.3.3.1 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 module.

MSC0_CLC
MSC0 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;** returns 0 if read; should be written with 0.

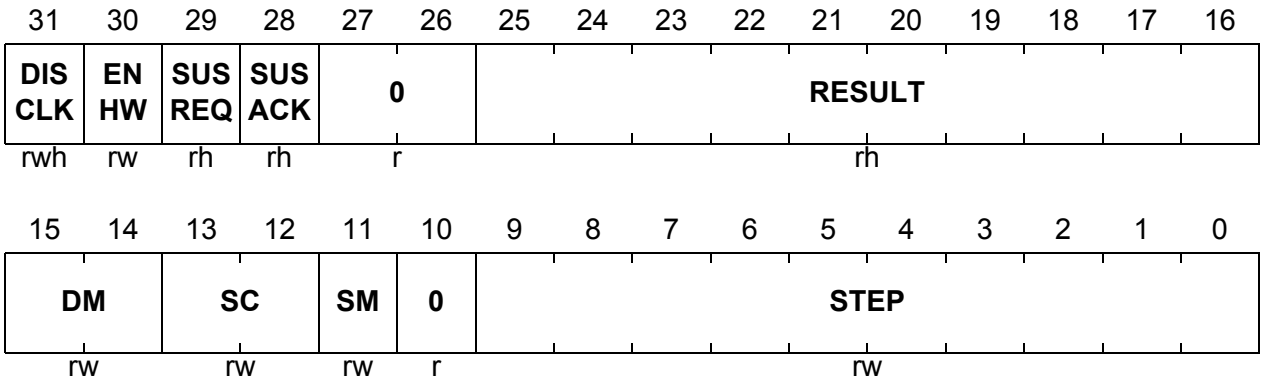
Note: After a hardware reset operation, the f_{CLC0} and f_{MSC0} clocks are switched off and the MSC0 module is disabled (DISS set).

*Note: Additional details on the Clock Control Register functionality are described in section **"Clock Control Register CLC" on Page 3-24** of the TC1766 User's Manual System Units part (Volume 1).*

19.3.3.2 Fractional Divider Register

The Fractional Divider Register controls the clock rate of the shift clock f_{MSC0}.

MSC0_FDR
MSC0 Fractional Divider Register ($0C_H$) 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** 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.

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]	rw	**Reserved;** read as 0; should be written with 0.

*Note: Additional details on the fractional divider register functionality are described in section **"Fractional Divider Operation" on Page 3-29** of the TC1766 User's Manual System Units part (Volume 1).*

19.3.4 Port Control

MSC0 clock and data output lines are connected to dedicated differential output drivers. Some of the MSC0 module I/O lines are connected to I/O ports and therefore controlled in the port logic (see also **Figure 19-29**). 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)

19.3.4.1 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 MSC0 module are controlled by the Port 2 input/output control registers.

Table 19-10 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 19-10 MSC0 I/O Line Selection and Setup

Module	Port Lines	Input/Output Control Register Bits[1)]	I/O
MSC0	P2.8 / EN00	P2_IOCR8.PC8 = $1X11_B$	Output
	P2.9 / EN01	P2_IOCR8.PC9 = $1X11_B$	Output
	P2.11 / FCLP0B	P2_IOCR8.PC11 = $1X11_B$	Output
	P2.12 / SOP0B	P2_IOCR12.PC12 = $1X11_B$	Output
	P2.13 / SDI0[2)]	P2_IOCR12.PC13 = $0XXX_B$	Input

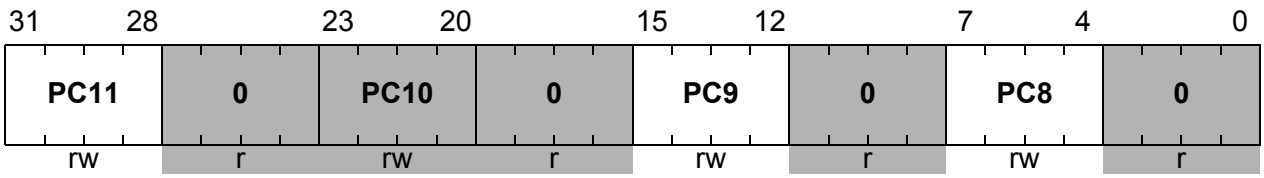
1) For possible PCx bit field combinations, see **Table 19-11**.

2) For the upstream channel serial data inputs, additionally bit fields MSC0_OCR.SDISEL must be set to 000_B.

Input/Output Control Registers

P2_IOCR8
Port 2 Input/Output Control Register 8

(18$_H$) Reset Value: 2020 2020$_H$

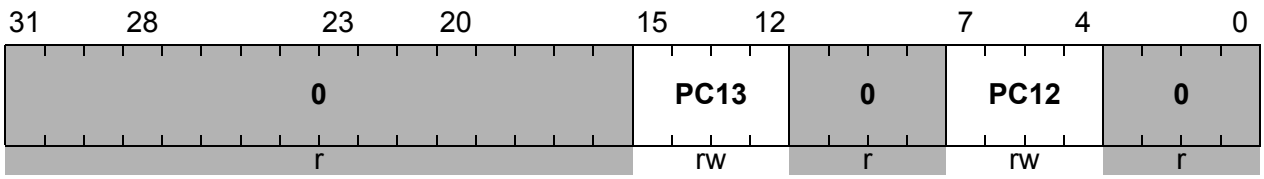


Field	Bits	Type	Description
		rw	**Port Output Control for Port 2.[9:8] and Port 2.11**[1)] These bit field determines the output port functionality:
PC8,	[7:4],		Port output control for P2.8 / EN00
PC9,	[15:12],		Port output control for P2.9 / EN01
PC11	[31:28]		Port output control for P2.11 / FCLP0B

1) For coding of bit field, see **Table 19-11**. Shaded bits and bit fields are “don’t care” for MSC0 I/O port control.

P2_IOCR12
Port 2 Input/Output Control Register 12

(1C$_H$) Reset Value: 2020 2020$_H$



Field	Bits	Type	Description
		rw	**Port Output Control for Port 2.[13:12]**[1)] These bit field determines the output port functionality:
PC12,	[7:4]		Port output control for P2.12 / SOP0B
PC13	[15:12]		Port input control for P2.13 / SDI0

1) For coding of bit field, see **Table 19-11**. Shaded bits and bit fields are “don’t care” for MSC0 I/O port control.

PCx Coding Table

Table 19-11 PCx Coding

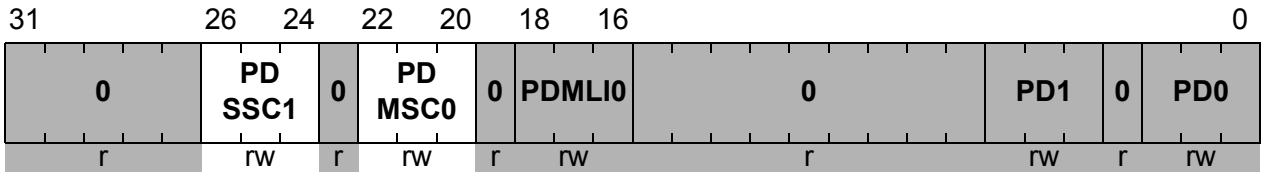
PCx[3:0]	I/O	Output Characteristics	Selected Pull-up/Pull-down/ Selected Output Function
$0X00_B$	Input	–	No pull device connected
$0X01_B$			Pull-down device connected
$0X10_B$ 1)			Pull-up device connected
$0X11_B$			No pull device connected
1011_B	Output	Push-pull	Output function ALT3
1111_B		Open-drain	Output function ALT3

1) This bit field value is default after reset.

19.3.4.2 Pad Driver Mode Register

The Port 2 pad driver mode register contains bit fields that determine the output driver strength and the slew rate of MSC0 output lines.

P2_PDR
Port 2 Pad Driver Mode Register (40_H) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
PDMSC0	[22:20]	rw	**Pad Driver Mode for P2.8/EN00, P2.9/EN01[1)]**
PDSSC1	[26:24]	rw	**Pad Driver Mode for P2.11/FCLP0B, P2.12/SOP0B[1)]**

1) For coding of bit field, see **Table 19-12**. Shaded bits and bit fields “don’t care” for MSC I/O port control.

PDx Selection Table

Table 19-12 Pad Driver Mode Mode Selection (Class A2 Pads)

PDx Bit Field	Driver Strength	Signal Transitions
000_B	Strong driver	Sharp edge[1)]
001_B		Medium edge[1)]
010_B		Soft edge[1)]
011_B	Weak driver	–
100_B	Medium driver	–
101_B		
110_B		
111_B	Weak driver	–

1) In strong driver mode, the output driver characteristics of class A2 pads can be additionally controlled by the temperature compensation logic.

19.3.5 On-Chip Connections

This section describes the on-chip connections of the MSC0 module.

19.3.5.1 EMGSTOPMSC Signal (from SCU)

The emergency stop input signal EMGSTOPMSC of the MSC0 module is 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.

*Note: Additional details on the emergency stop capabilities are described in **Section 5.11** on **Page 5-57** of the TC1766 System Units User's Manual.*

19.3.5.2 DMA Controller Service Requests

Two service request outputs (SR[3:2]) of the MSC0 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 19-13**.

Table 19-13 Service Request Lines of MSC0

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	CH04_REQI3	DMA Channel 04 Request Input 3
	SR3	CH03_REQI5 CH05_REQI3	DMA Channel 03 Request Input 5 DMA Channel 05 Request Input 3

19.3.6 Interrupt Control Registers

In the TC1766, the two service request outputs SR[1:0] of the MSC0 module are connected to one interrupt node. The upper two service request outputs SR[3:2] of the MSC0 module are not connected to interrupt nodes, but can be used as DMA requests (see **Table 19-13**).

MSC0_SRCx (x = 0-1)
MSC0 Service Request Control Register x
(FC_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**
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;** returns 0 if read; should be written with 0.

*Note: Additional details on service request nodes and the service request control registers are described on **Page 12-3** of the TC1766 User's Manual System Units part (Volume 1).*

20 Controller Area Network (MultiCAN) Controller

This chapter describes the MultiCAN controller of the TC1766. It contains the following sections:

- CAN basics (see **Page 20-2**)
- Overview of the CAN Module in the TC1766 (see **Page 20-11**)
- Functional description of the MultiCAN Kernel (see **Page 20-14**)
- MultiCAN Kernel register description (see **Page 20-54**)
- TC1766 implementation-specific details (port connections and control, interrupt control, address decoding, clock control, see **Page 20-108**).

Note: The MultiCAN register names described in this chapter are referenced in the TC1766 User's Manual by the module name prefix "CAN_".

20.1 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).

20.1.1 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.

20.1.2 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.

20.1.2.1 Data Frames

There are two types of Data Frames defined (see **Figure 20-1**):

- 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.

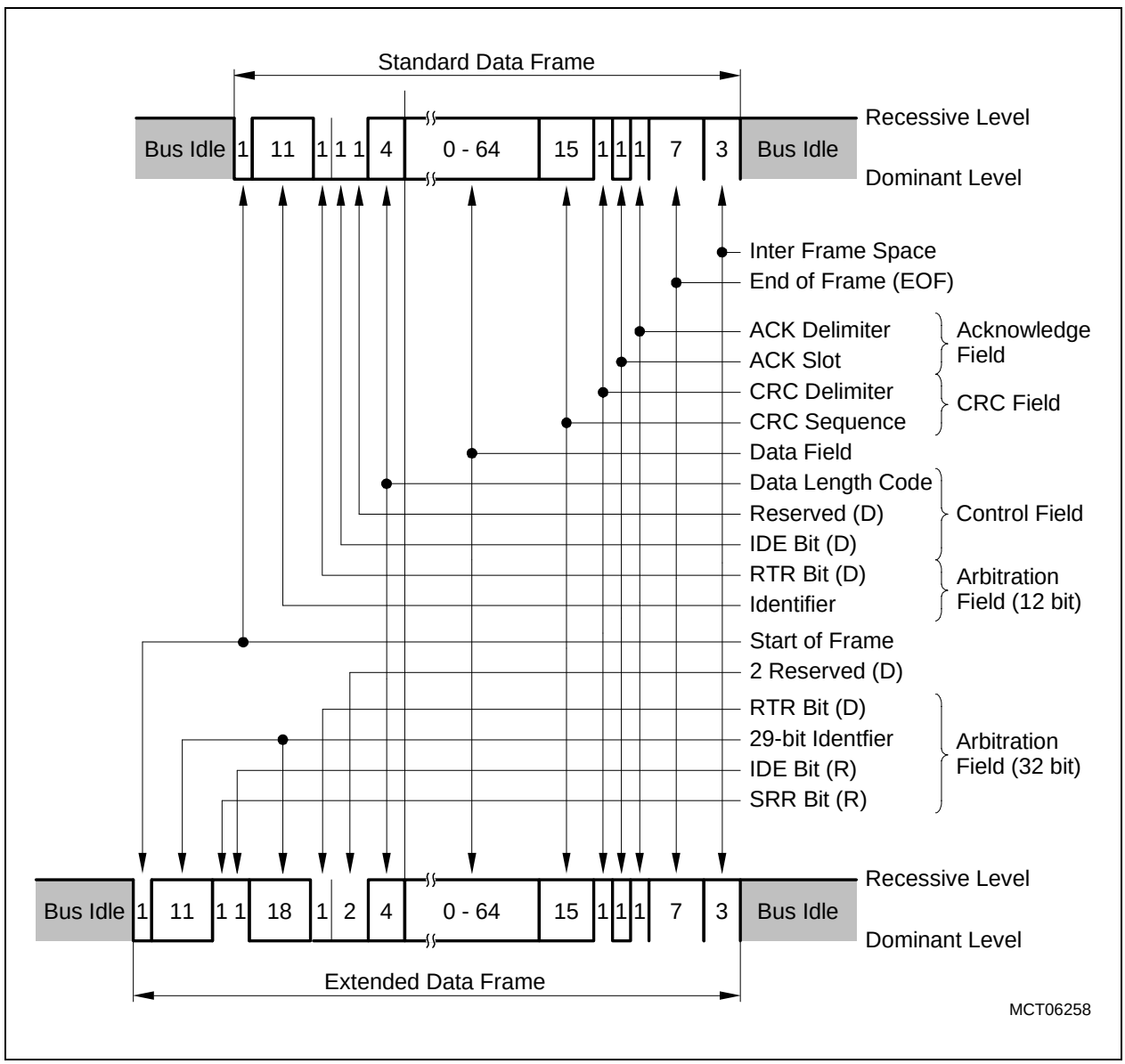


Figure 20-1 CAN Data Frame

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.

20.1.2.2 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 20-2**.

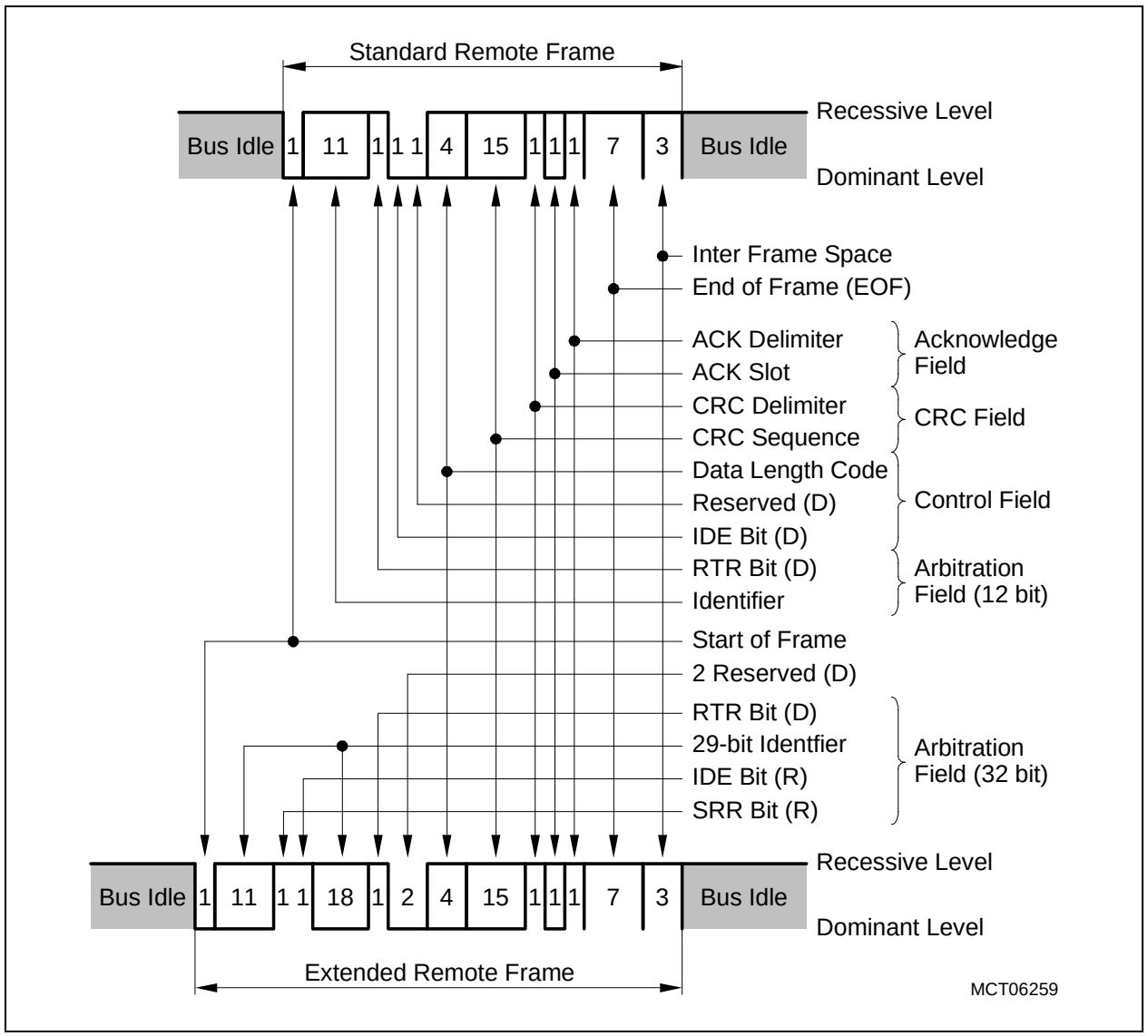


Figure 20-2 CAN Remote Frame

20.1.2.3 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.

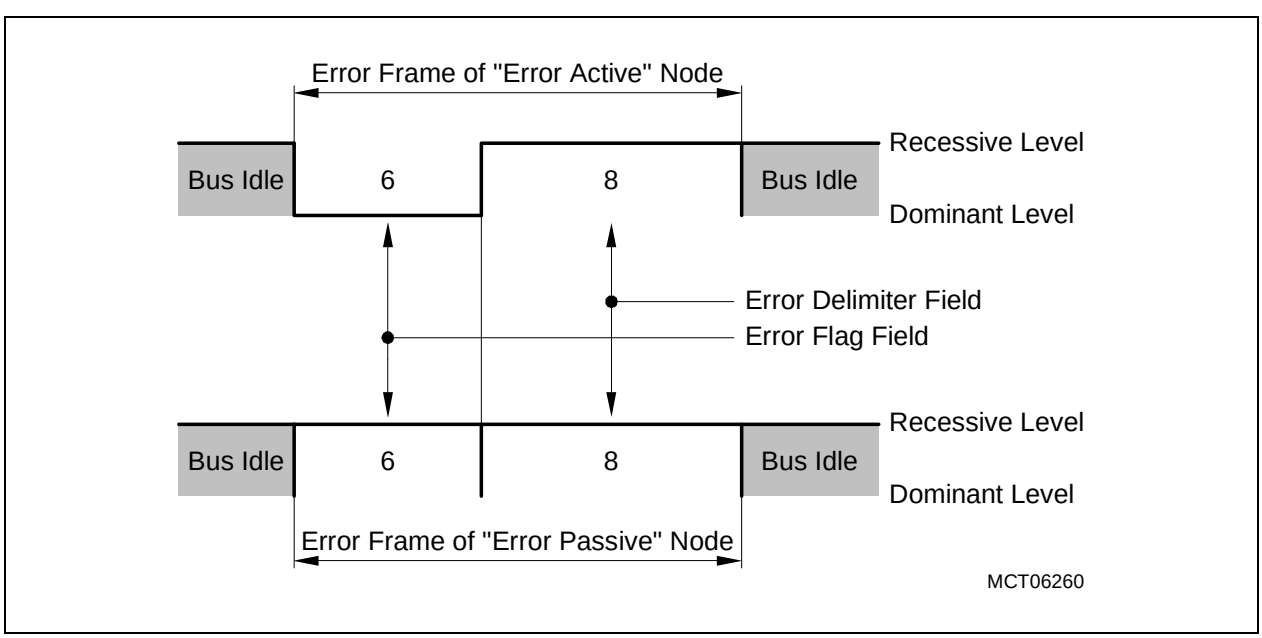


Figure 20-3 CAN Error Frames

20.1.3 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 20-4**.

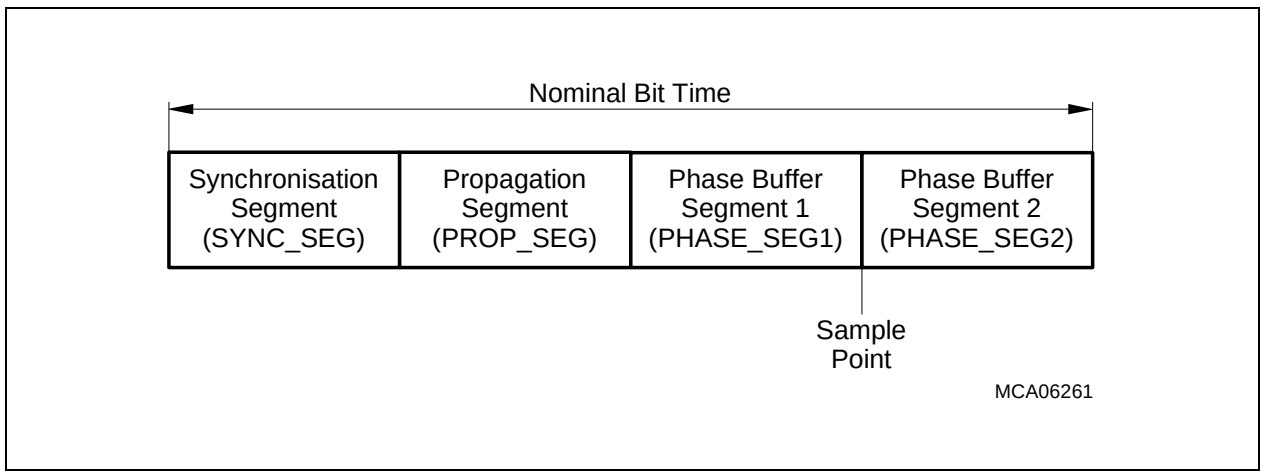


Figure 20-4 Partition of Nominal Bit Time

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.

20.1.4 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 TC1766) 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.

20.2 Overview

This section describes the serial communication module called MultiCAN (CAN = Controller Area Network) of the TC1766. The MultiCAN module contains two independent CAN nodes, representing two serial communication interfaces.

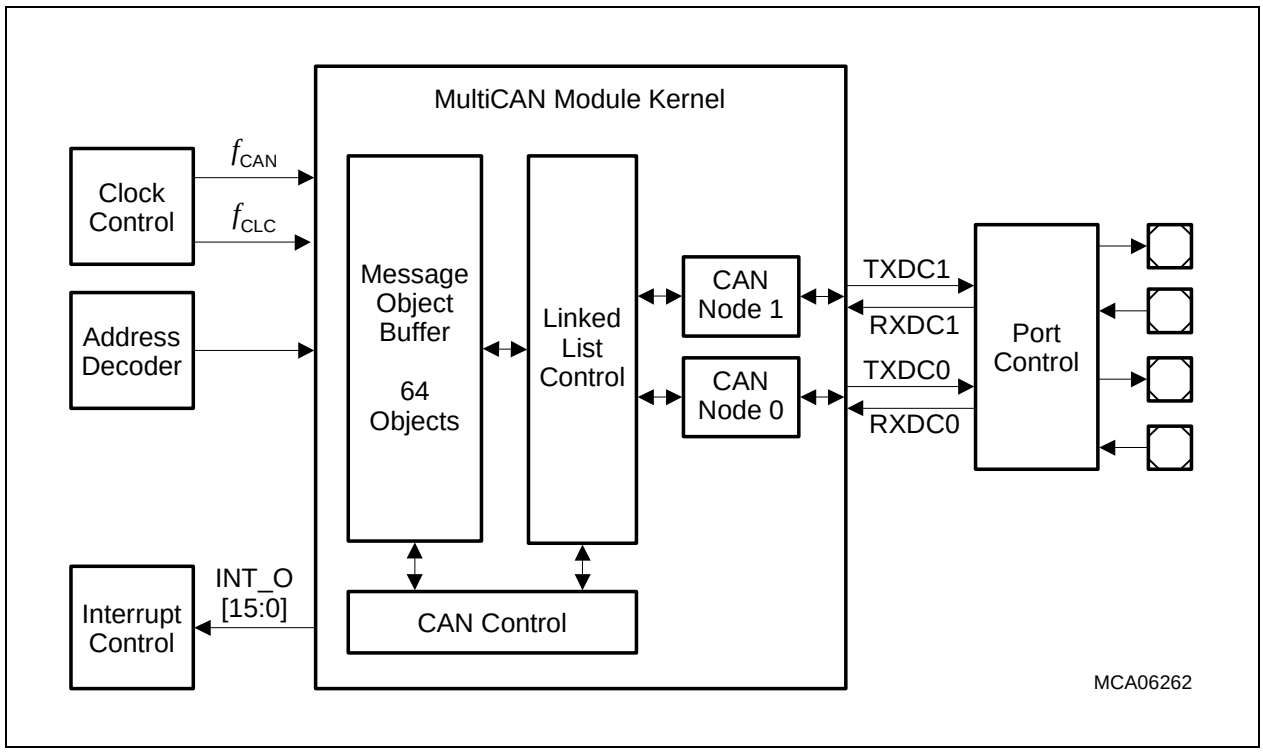


Figure 20-5 Overview of the MultiCAN Module

20.2.1 MultiCAN Module

The MultiCAN module contains two 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 64 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 combined flexibly into a dedicated register field of 128 notification bits

20.3 MultiCAN Kernel Functional Description

This section describes the functionality of the MultiCAN module.

20.3.1 Module Structure

Figure 20-6 shows the general structure of the MultiCAN module.

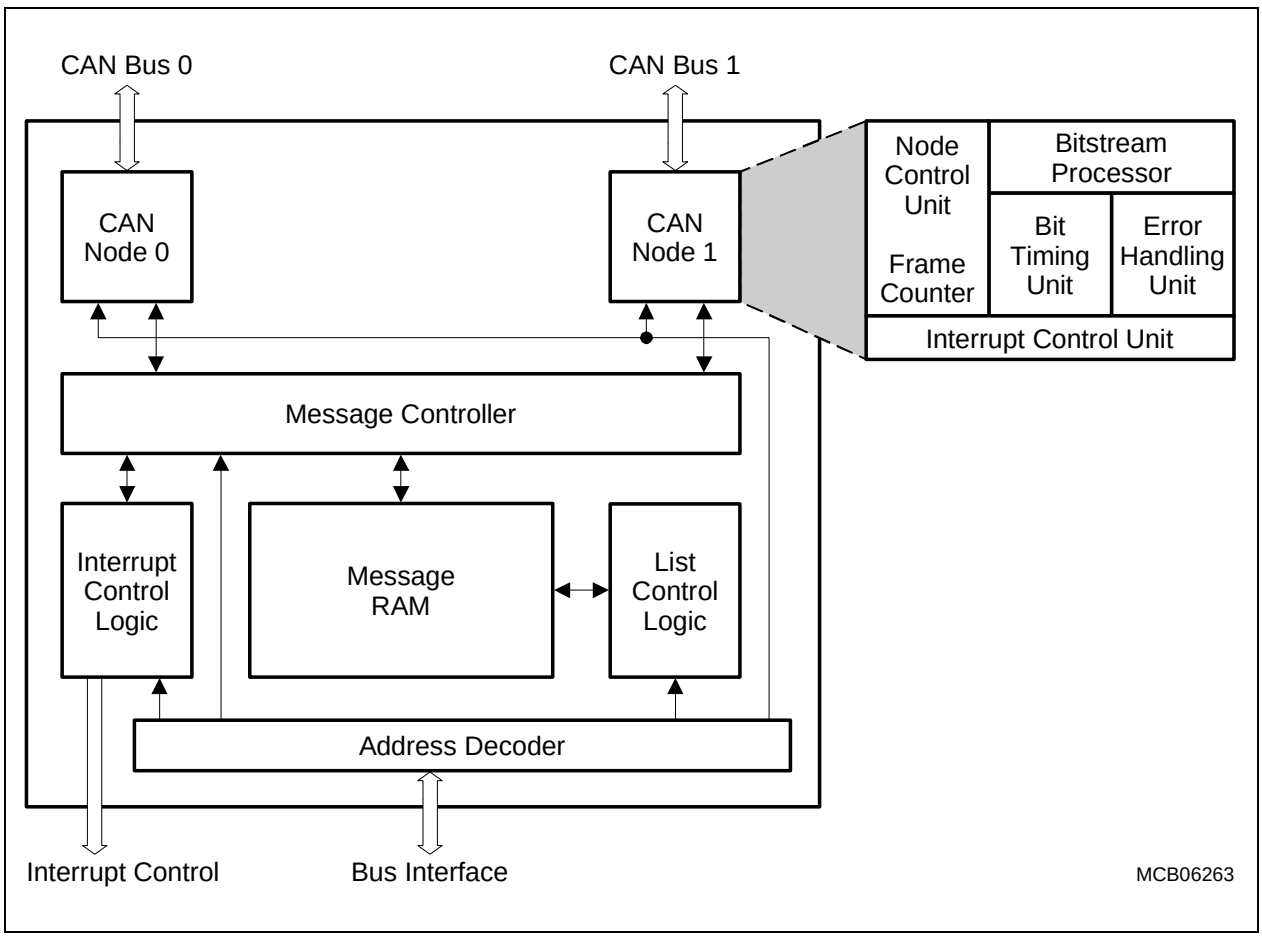


Figure 20-6 MultiCAN Block Diagram

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 20-8**. 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 cleared 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.

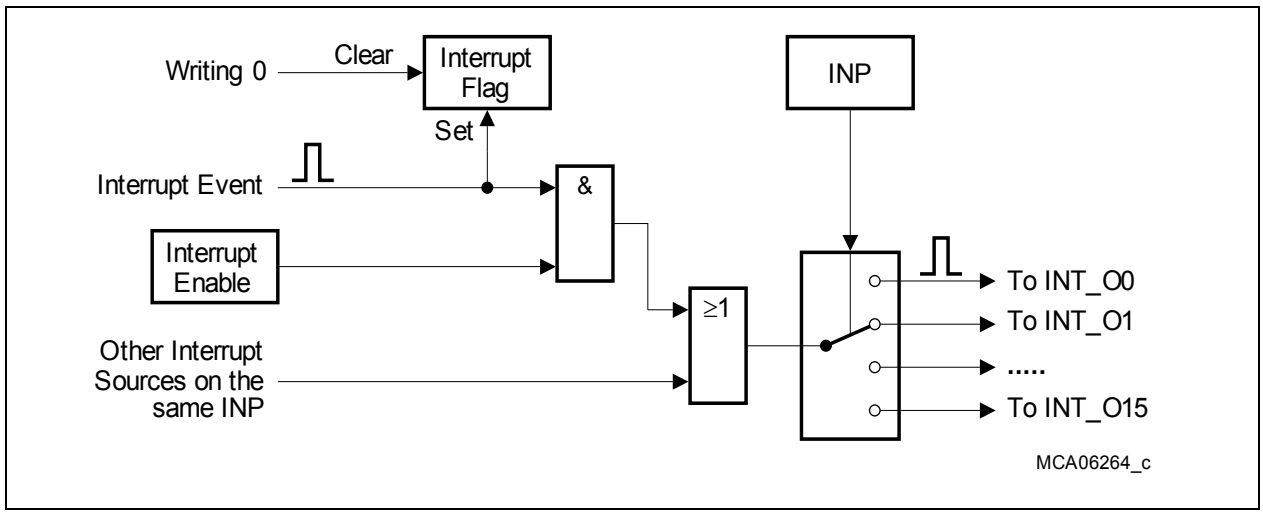


Figure 20-7 General Interrupt Structure

20.3.2 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 20-110**.

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.

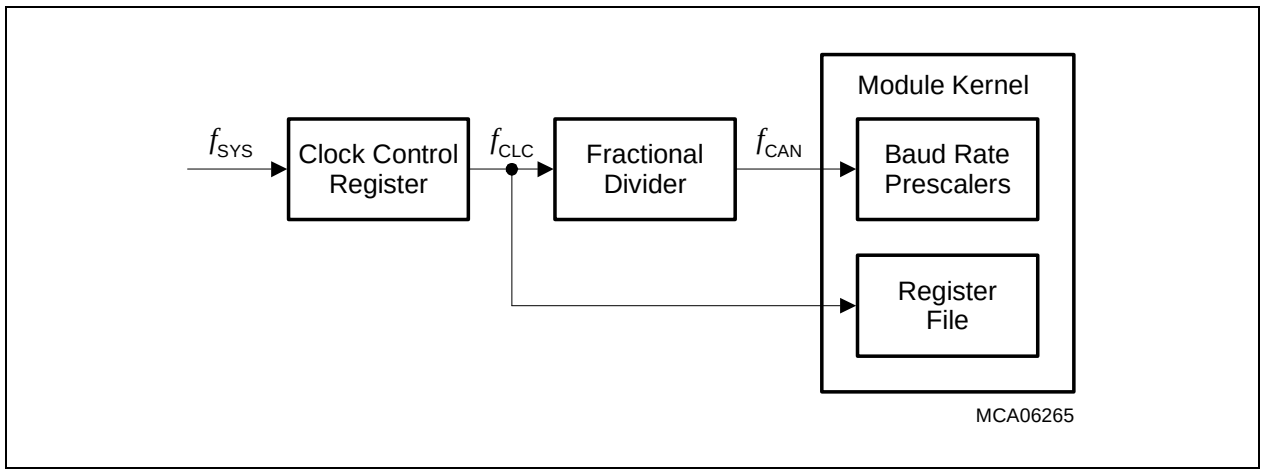


Figure 20-8 MultiCAN Clock Generation

Table 20-1 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 20-1 Minimum Operating Frequencies [MHz]

Number of Allocated Message Objects[1)]	with 1 CAN Node Active	with 2 CAN Nodes Active
16 Message Objects	12	19
32 Message Objects	15	23
64 Message Objects	21	28

1) Only those message objects that are allocated to a CAN node must be taken into account. The unallocated message objects have no influence on the minimum operating frequency.

20.3.3 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 20.5.4.2** on **Page 20-116**.

20.3.4 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.

20.3.5 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.

20.3.5.1 Bit Timing Unit

According to the ISO 11898 standard, a CAN bit time is subdivided into different segments (**Figure 20-9**). 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 20-80**). The baud rate prescaler is driven by the module timer clock f_{CAN} (generation and control of f_{CAN} is described on **Page 20-110**).

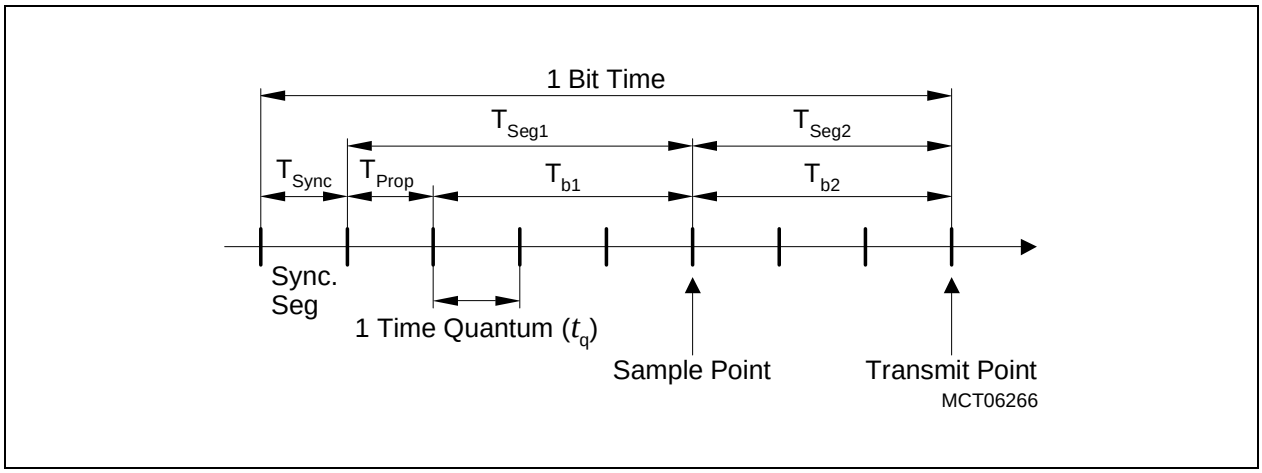


Figure 20-9 CAN Bus Bit Timing Standard

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
= (BRP + 1) / 8 × 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 clearing 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.

20.3.5.2 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.

20.3.5.3 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 20-82**) 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 cleared if both error counters fall below the error warning limit again.

20.3.5.4 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 20-83**). Bit field NFCRx.CFSEL determines the operation mode of the frame counter:

- **Frame Count Mode:**
  The frame counter is incremented after the successful transmission and/or reception of a CAN frame. The incremented value is copied into the CFCVAL bit field of the MOIPRn register of the message object involved in the transfer.
- **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 20.3.7.3**).

20.3.5.5 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 20-118** for further processing of the CAN node interrupts.

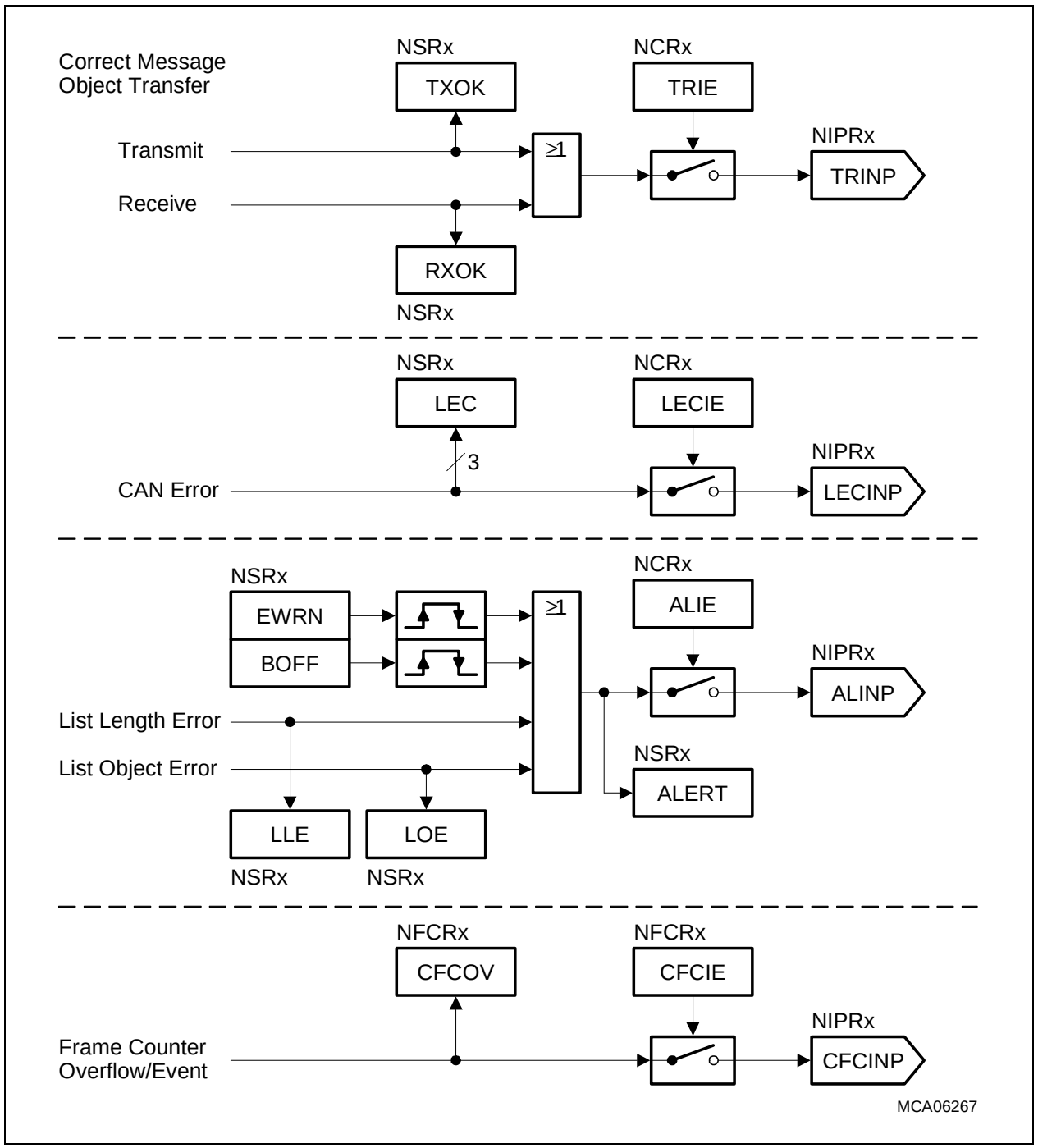


Figure 20-10 CAN Node Interrupts

20.3.6 Message Object List Structure

This section describes the structure of the message object lists in the MultiCAN module.

20.3.6.1 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 20-11**, the three message objects (3, 5, and 16) are allocated to the list with index 2 (List Register LIST2).

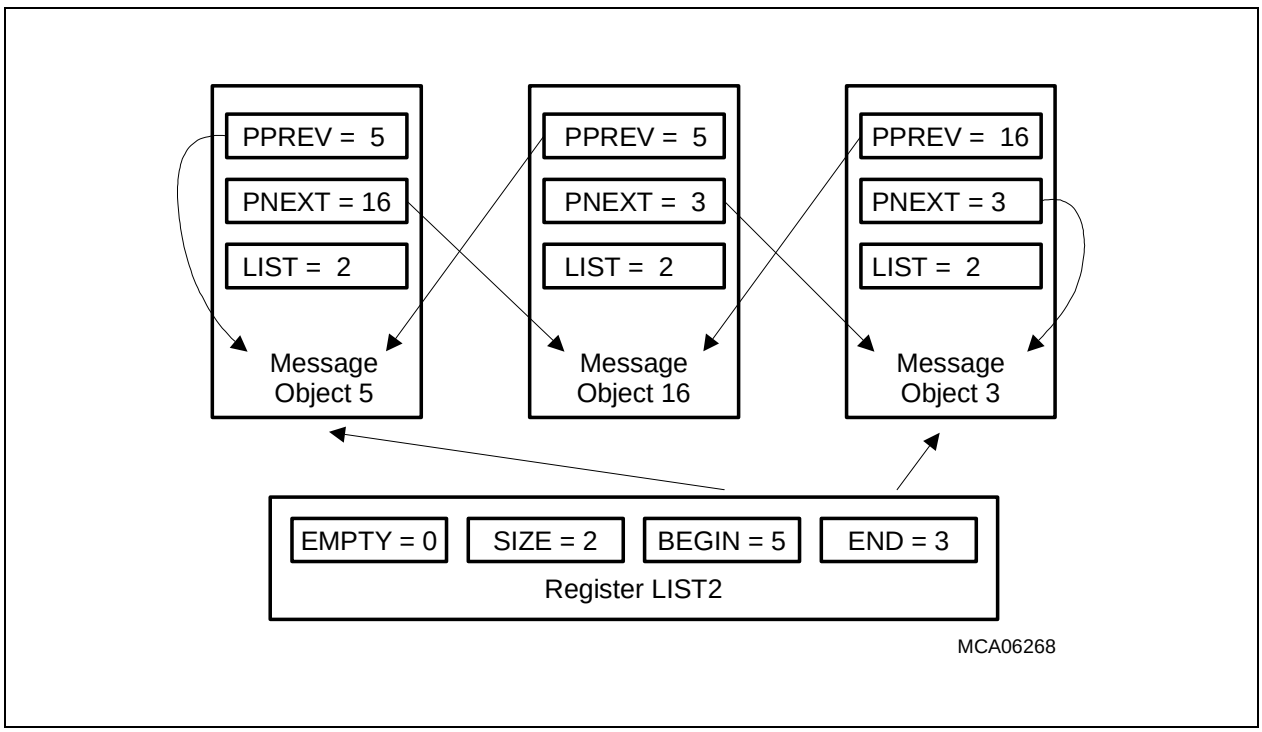


Figure 20-11 Example Allocation of Message Objects to a List

Bit field BEGIN in the List Register (for definition, see **Page 20-64**) 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 20-87**) 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.

20.3.6.2 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).

20.3.6.3 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 20-12**. 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.

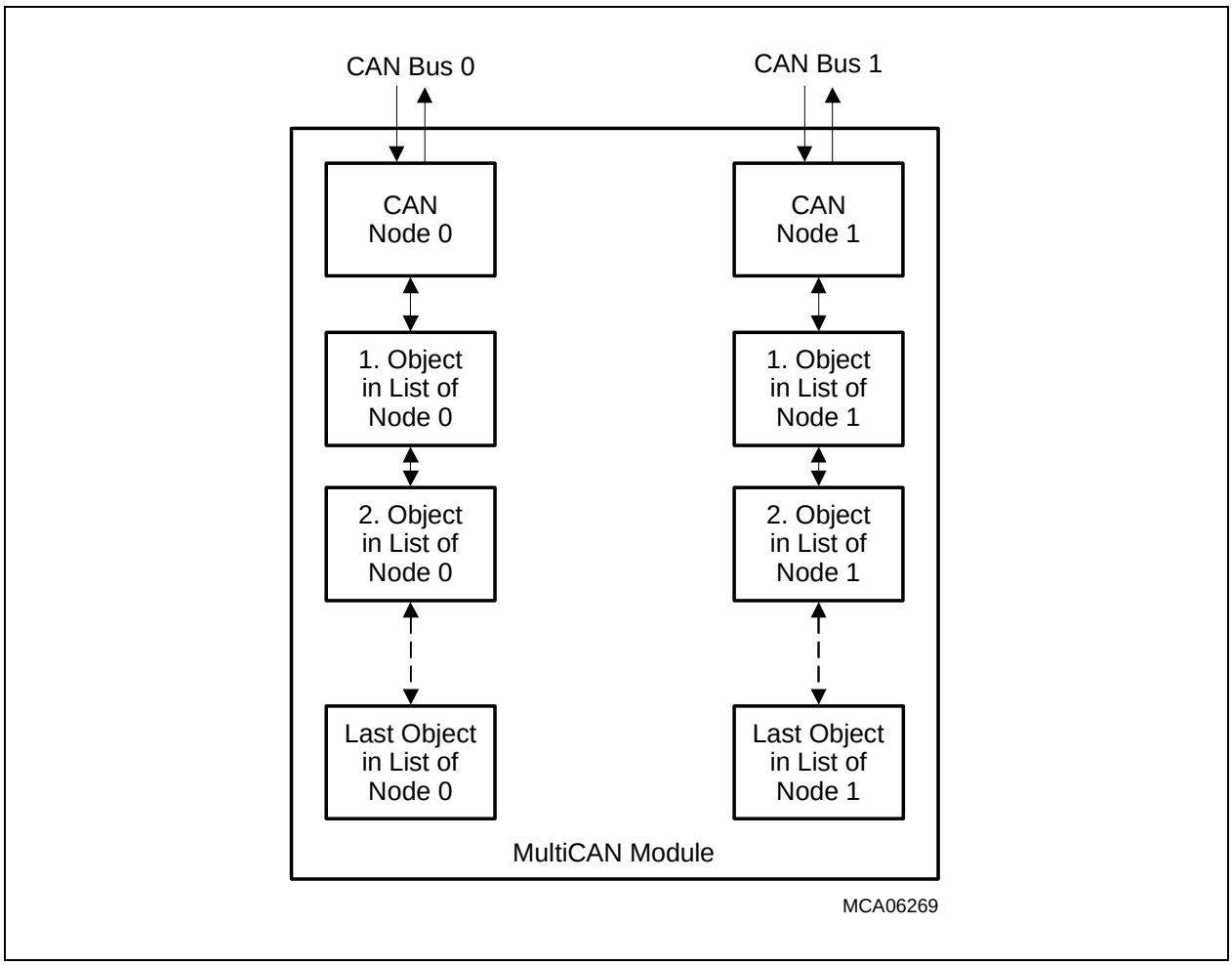


Figure 20-12 Message Objects Linked to CAN Nodes

20.3.6.4 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 20-2 gives an overview on the available panel commands while Table 20-6 on Page 20-59 describes the panel commands in more detail.

Table 20-2 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 20-58). 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.

20.3.7 CAN Node Analysis Features

The chapter describes the CAN node analysis capabilities of the MultiCAN module.

20.3.7.1 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.

20.3.7.2 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 20-13**). 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.

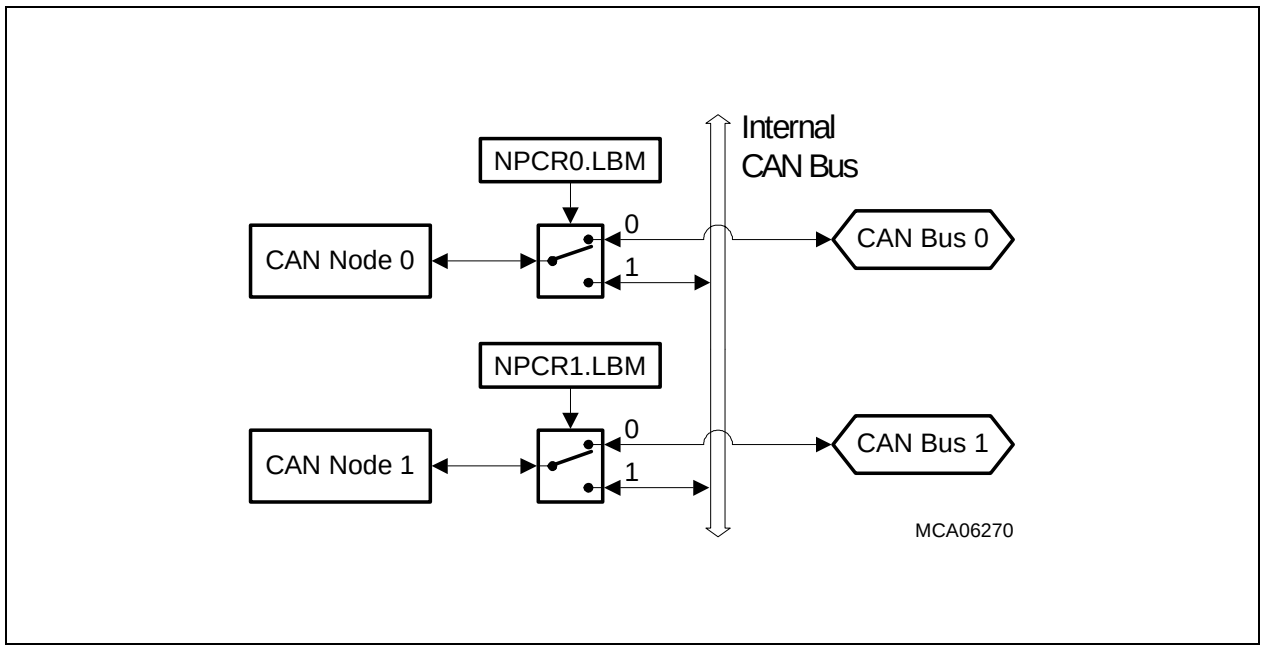


Figure 20-13 Loop-Back Mode

20.3.7.3 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 20-10**).

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.

20.3.8 Message Acceptance Filtering

The chapter describes the Message Acceptance Filtering capabilities of the MultiCAN module.

20.3.8.1 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 20-90**) 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 20-14** 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 20-104**):

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.

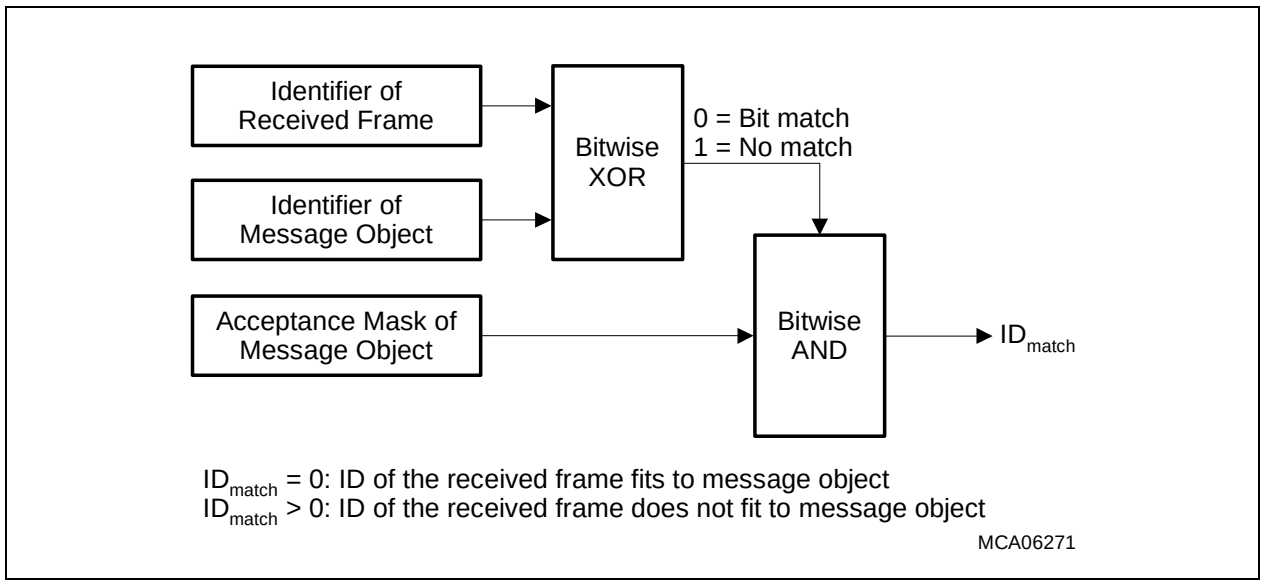


Figure 20-14 Received Message Identifier Acceptance Check

20.3.8.2 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 20-15**).

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. MOa is a list successor of MOb. 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 20-12** on **Page 20-105**).
- 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.

The three priority rules are valid for normal CAN operation.

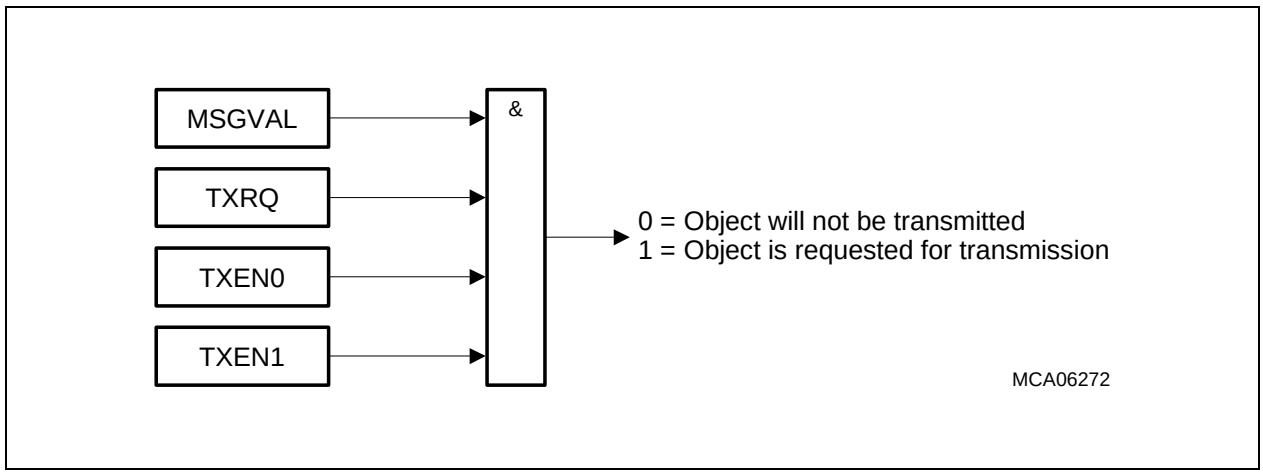


Figure 20-15 Effective Transmit Request of Message Object

20.3.9 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.

20.3.9.1 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 20-16**). 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 20-118** for further processing of the message object interrupts.

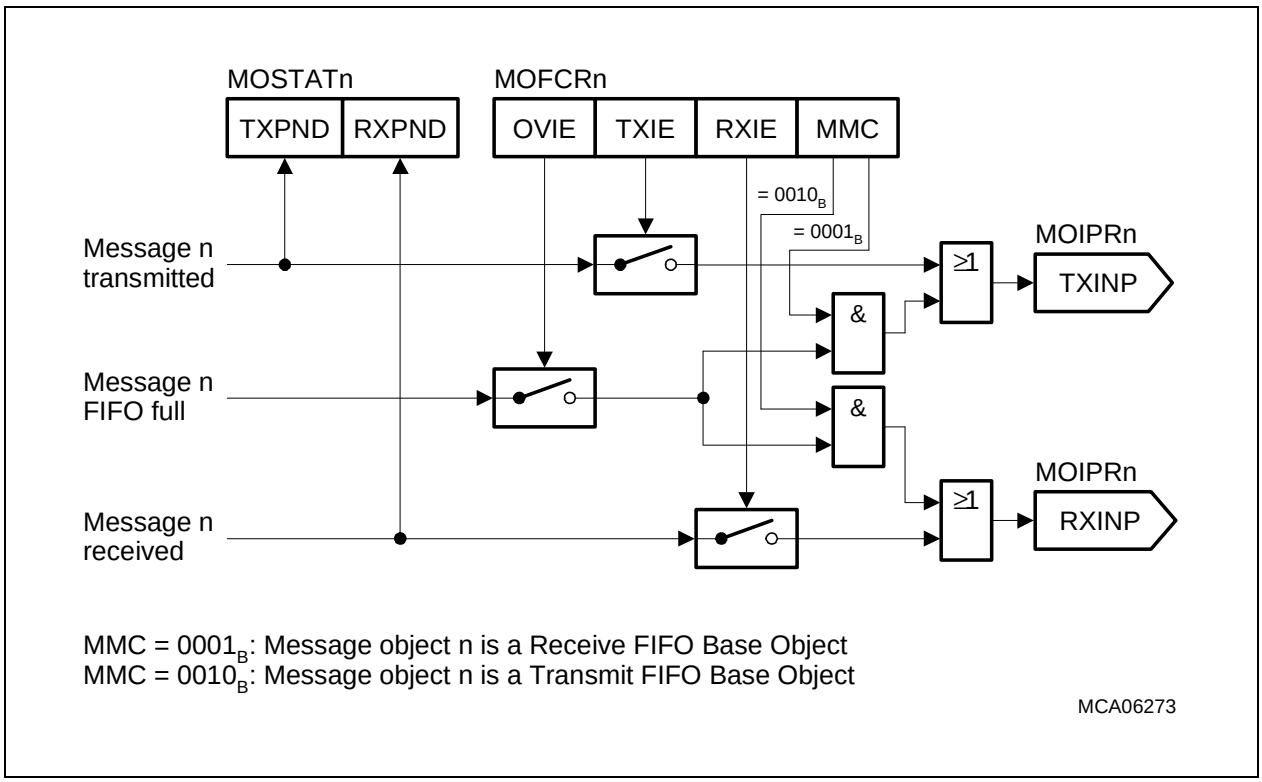


Figure 20-16 Message Interrupt Request Routing

20.3.9.2 Pending Messages

When a message interrupt request is generated, a message pending bit is set in one of the Message Pending Registers. There are eight Message Pending Registers, MSPNDk (k = 0-7) with 32 pending bits available each, resulting in 256 pending bits. **Figure 20-17** shows the allocation of the message pending bits.

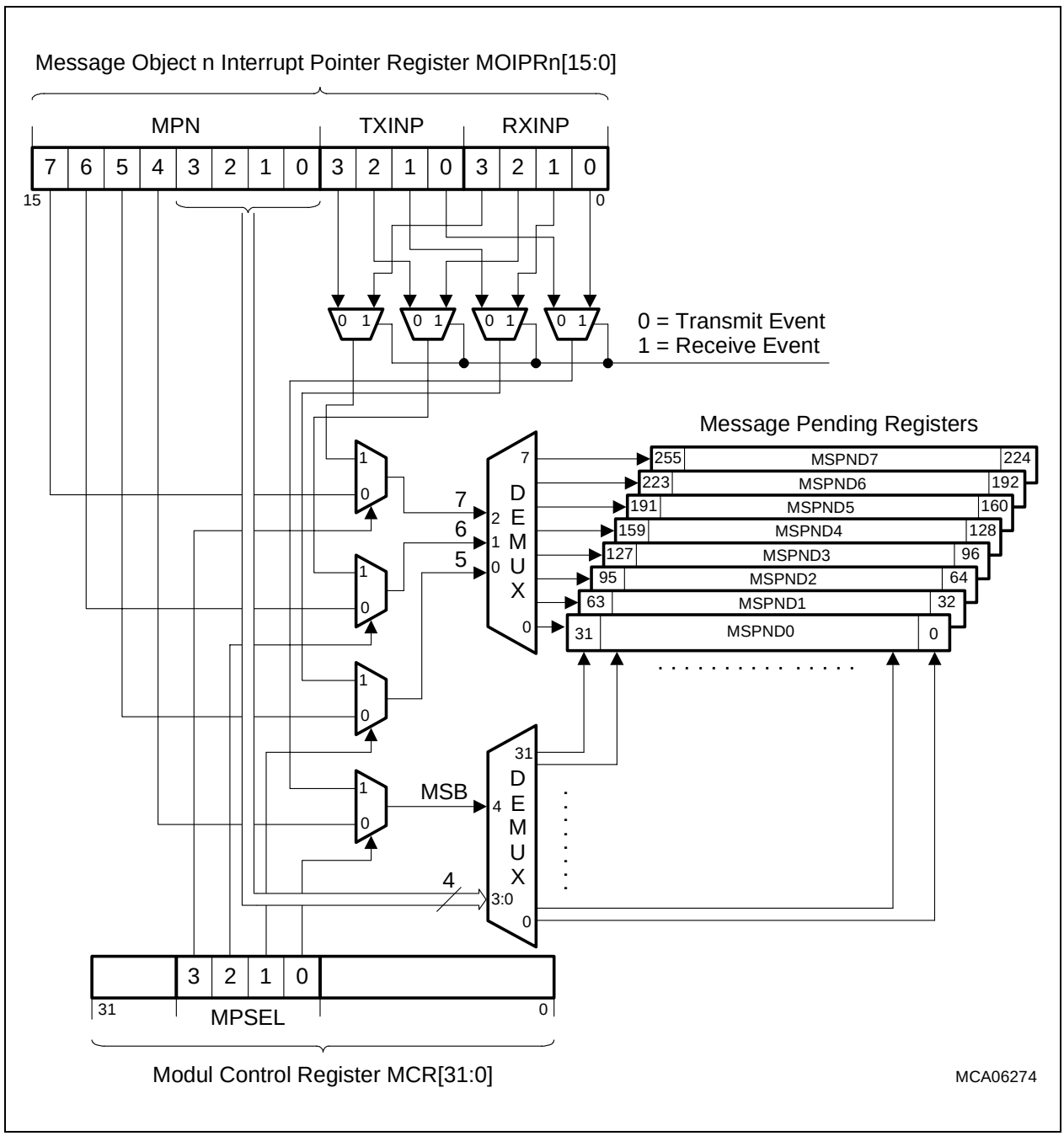


Figure 20-17 Message Pending Bit Allocation

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 20-62**). The location selection consists of 2 parts:

- The upper three bits of MOINPRn.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 MOINPRn.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 20-67**) 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.

20.3.10 Message Object Data Handling

This chapter describes the handling capabilities for the Message Object Data of the MultiCAN module.

20.3.10.1 Frame Reception

After the reception of a message, it is stored in a message object according to the scheme shown in **Figure 20-18**. 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 cleared 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 (clearing 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.

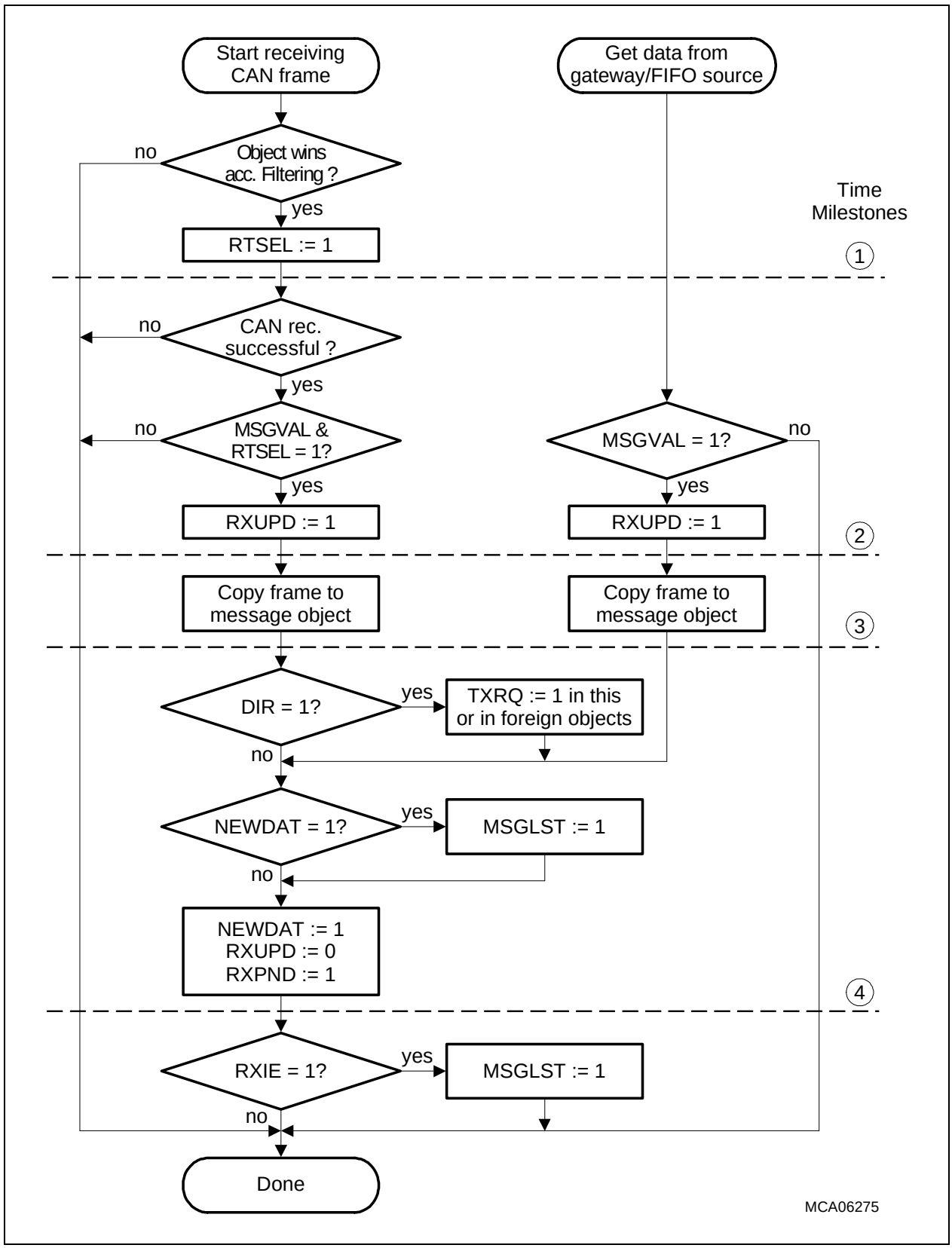


Figure 20-18 Reception of a Message Object

20.3.10.2 Frame Transmission

The process of a message object transmission is shown in **Figure 20-19**. 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 20-15**. Although these bits are equivalent with respect to the transmission process, they have different semantics:

Table 20-3 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 20-3 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.

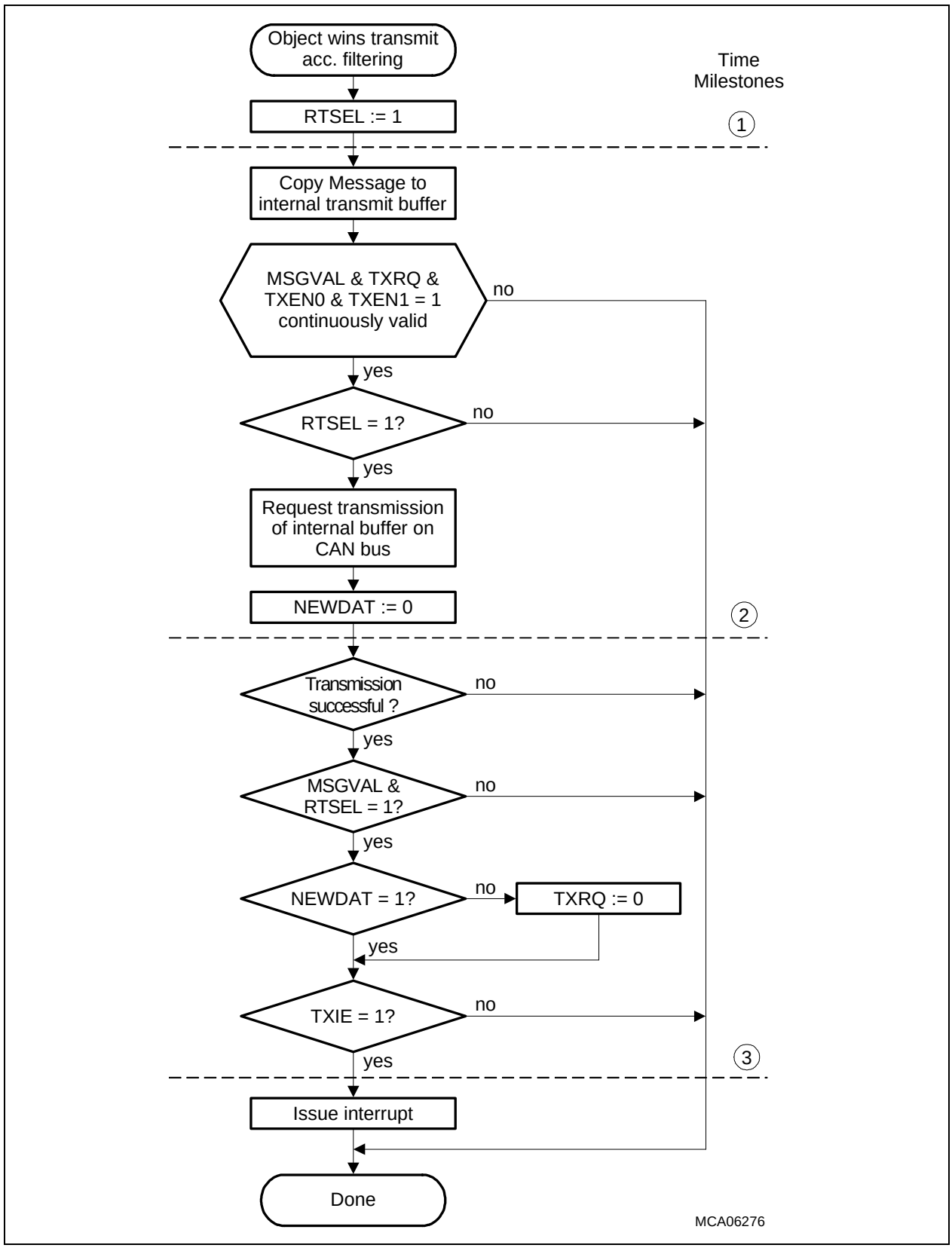


Figure 20-19 Transmission of a Message Object

20.3.11 Message Object Functionality

This chapter describes the functionality of the Message Objects in the MultiCAN module.

20.3.11.1 Standard Message Object

A message object is selected as standard message object when bit field MOFCRn.MMC = 0000_B (see **Page 20-83**). 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.

20.3.11.2 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.

20.3.11.3 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.

20.3.11.4 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 20-20**. 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 20-20** 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.

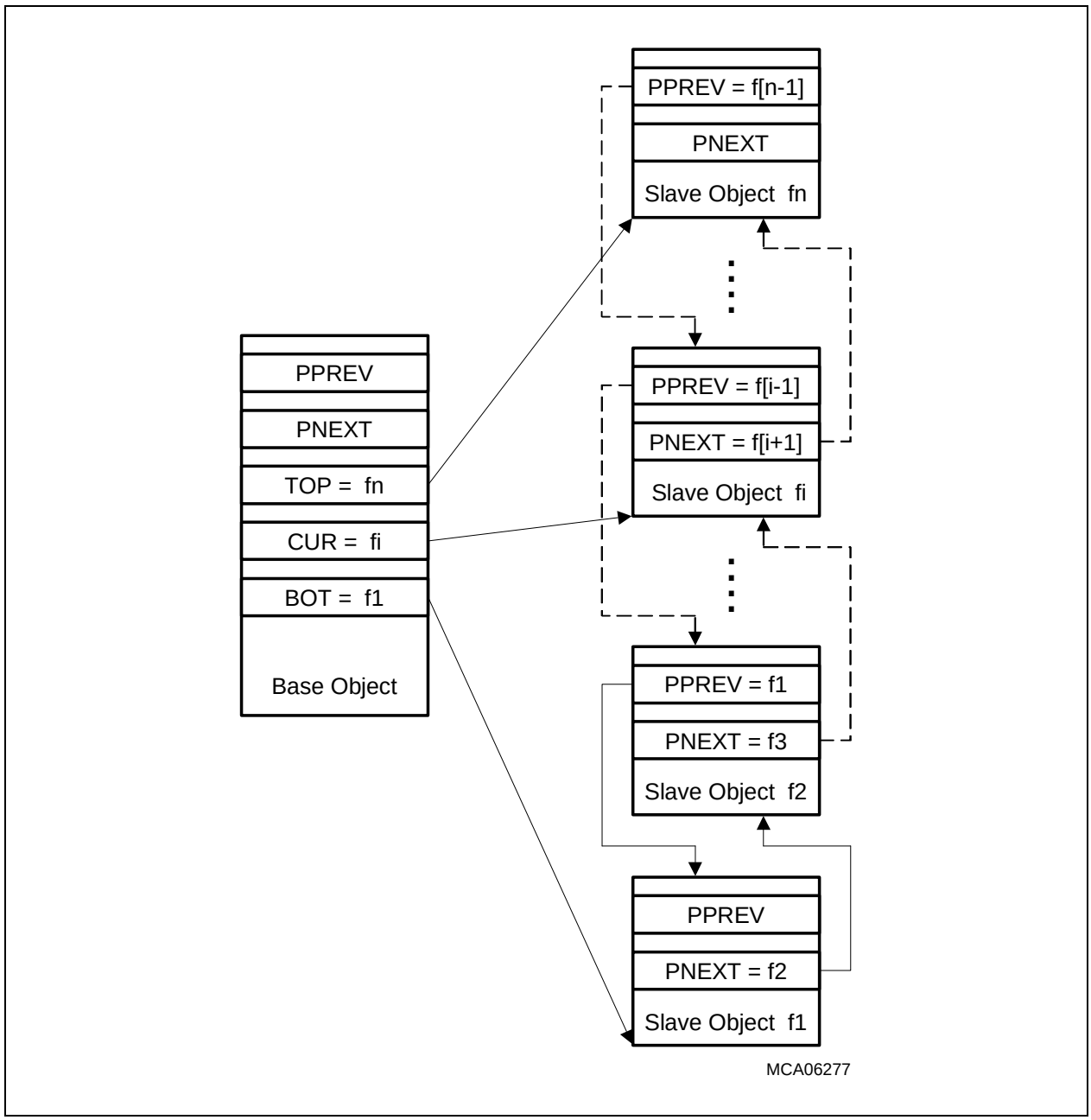


Figure 20-20 FIFO Structure with FIFO Base Object and n FIFO Slave Objects

20.3.11.5 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.

20.3.11.6 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 TXEN 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.

20.3.11.7 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 20-21**):

1. If bit MOFCR.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 20-47**. 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 20-20**, 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).

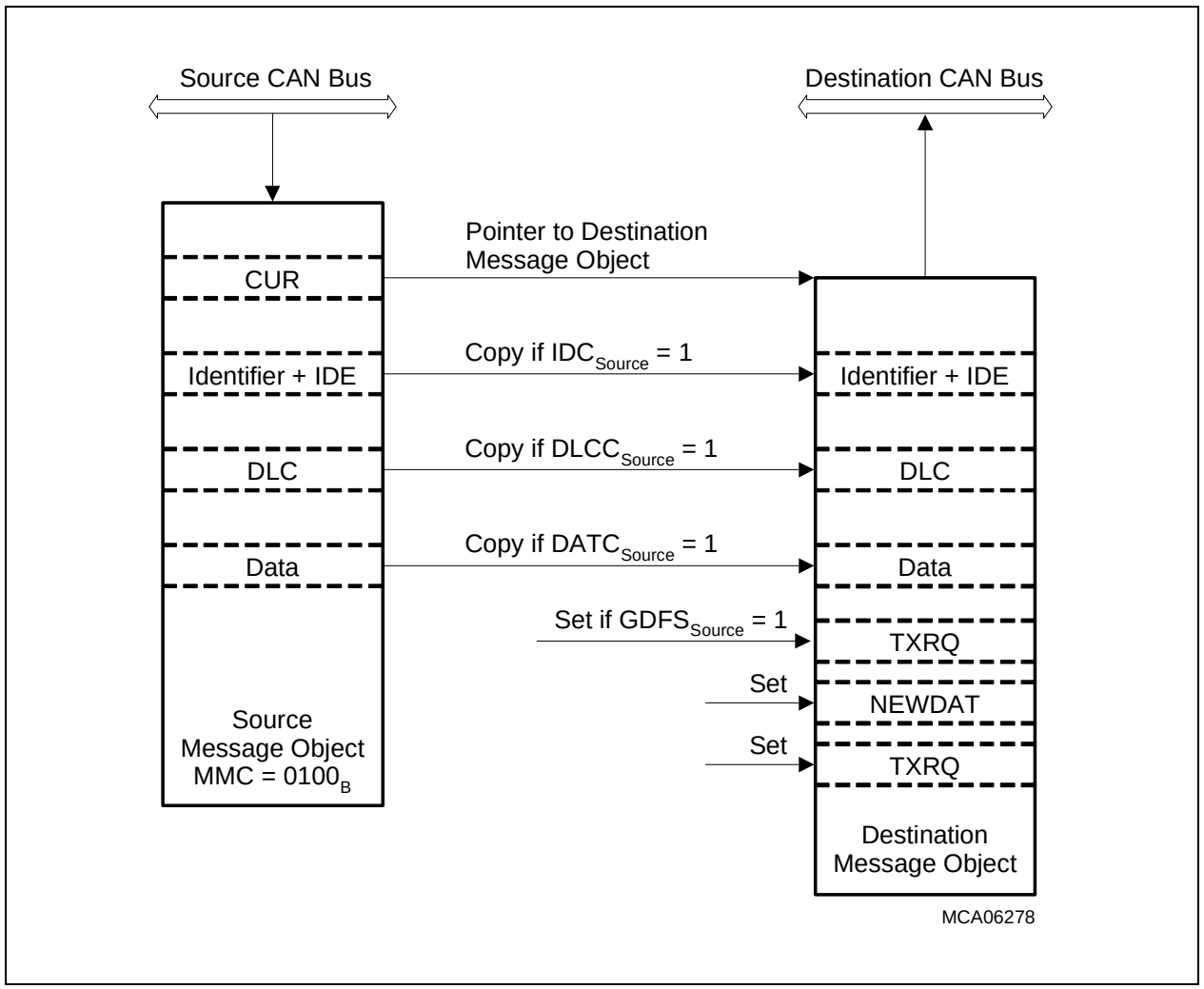


Figure 20-21 Gateway Transfer from Source to Destination

20.3.11.8 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.

20.4 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 TC1766 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

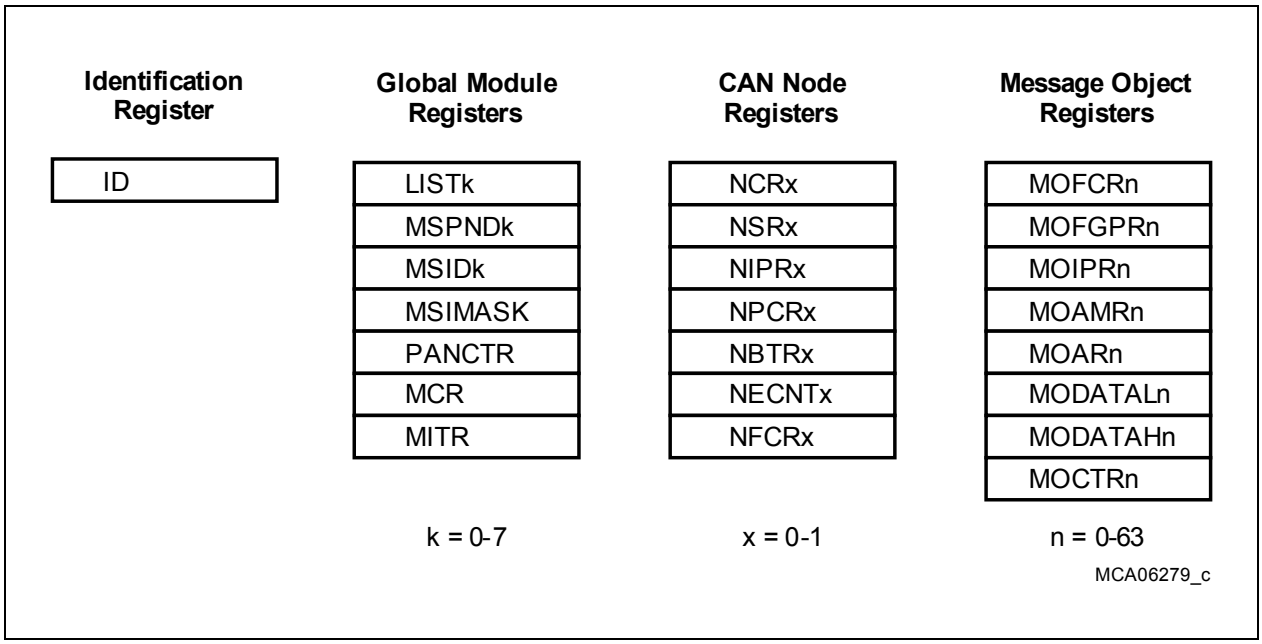


Figure 20-22 MultiCAN Kernel Registers

The complete and detailed address map of the MultiCAN module is described in **Table 16-17** on **Page 16-41** of the TC1766 User's Manual System Units part (Volume 1).

Table 20-4 Registers Address Space - MultiCAN Kernel Registers

Module	Base Address	End Address	Note
CAN	F000 4000_H	F000 $5FFF_H$	-

Table 20-5 Registers Overview - MultiCAN Kernel Registers

Register Short Name	Register Long Name	Offset Address[1)]	Description see
ID	Module Identification Register i	008_H	Page 20-57
LISTk	List Register k	$100_H + k \times 4_H$	Page 20-64
MSPNDk	Message Pending Register k	$120_H + k \times 4_H$	Page 20-66
MSIDk	Message Index Register k	$140_H + k \times 4_H$	Page 20-67
MSIMASK	Message Index Mask Register	$1C0_H$	Page 20-68
PANCTR	Panel Control Register	$1C4_H$	Page 20-58
MCR	Module Control Register	$1C8_H$	Page 20-62
MITR	Module Interrupt Trigger Reg.	$1CC_H$	Page 20-63
NCRx	Node x Control Register	$200_H + x \times 100_H$	Page 20-69
NSRx	Node x Status Register	$204_H + x \times 100_H$	Page 20-73
NIPRx	Node x Interrupt Pointer Reg.	$208_H + x \times 100_H$	Page 20-77
NPCRx	Node x Port Control Register	$20C_H + x \times 100_H$	Page 20-79
NBTRx	Node x Bit Timing Register	$210_H + x \times 100_H$	Page 20-80
NECNTx	Node x Error Counter Register	$214_H + x \times 100_H$	Page 20-82
NFCRx	Node x Frame Counter Register	$218_H + x \times 100_H$	Page 20-83
MOFCRn	Message Object n Function Control Register	$400_H + n \times 20_H$	Page 20-97
MOFGPRn	Message Object n FIFO/Gateway Pointer Register	$404_H + n \times 20_H$	Page 20-101
MOIPRn	Message Object n Interrupt Pointer Register	$408_H + n \times 20_H$	Page 20-95
MOAMRn	Message Object n Acceptance Mask Register	$40C_H + n \times 20_H$	Page 20-102
MODATALn	Message Object n Data Register Low	$410_H + n \times 20_H$	Page 20-106
MODATAHn	Message Object n Data Register High	$414_H + n \times 20_H$	Page 20-107
MOARn	Message Object n Arbitration Register	$418_H + n \times 20_H$	Page 20-103
MOCTRn MOSTATn	Message Object n Control Reg. Message Object n Status Reg.	$41C_H + n \times 20_H$	Page 20-87 Page 20-90

1) The absolute register address is calculated as follows:
Module Base Address (**Table 20-4**) + 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.

Figure 20-23 shows the MultiCAN register address map.

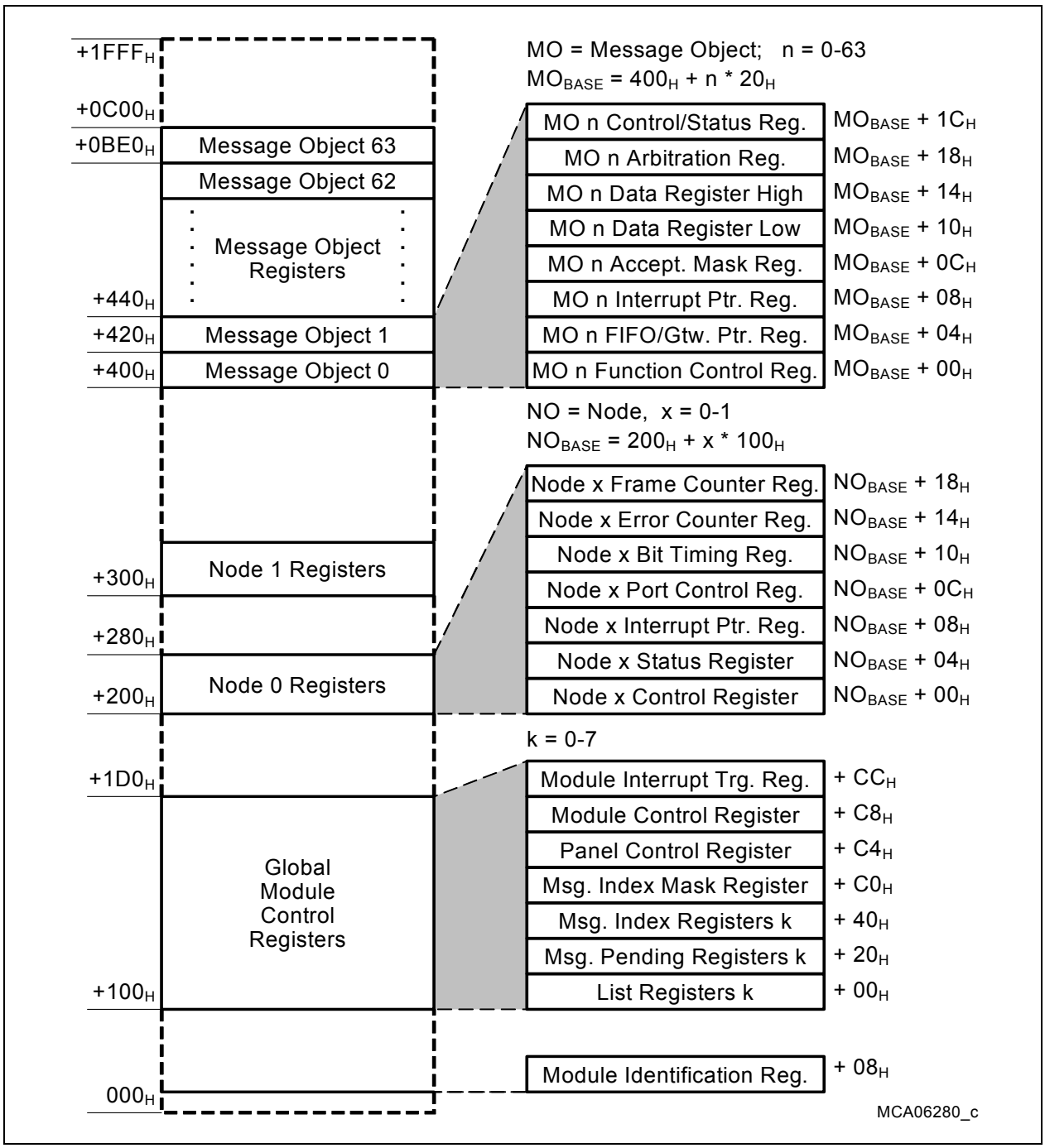
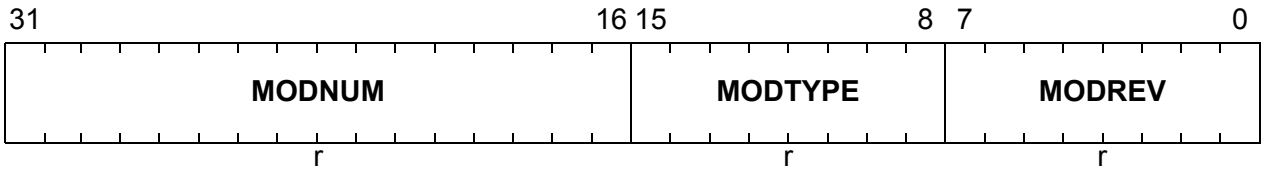


Figure 20-23 MultiCAN Kernel Register Address Map

20.4.1 MultiCAN Module Identification Register

The MultiCAN Module Identification Register ID contains read-only information about the MultiCAN module version.

ID
Module Identification Register **(008_H)** **Reset Value: $002B\ C0XX_H$**



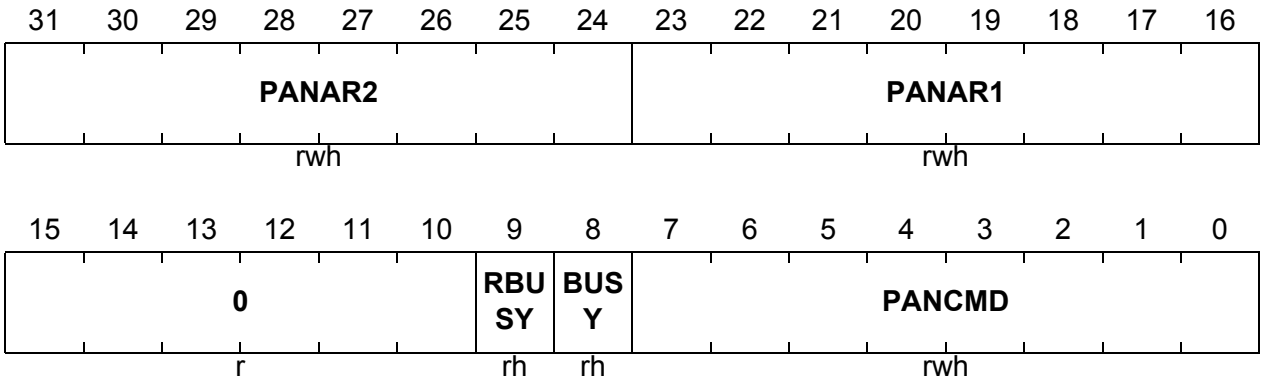
Field	Bits	Type	Description
MODREV	[7:0]	r	**Module Revision Number** MODREV defines the module revision number. The value of a module revision starts with 01_H (first revision).
MODTYPE	[15:8]	r	**Module Type** This bit field defines the module as a 32-bit module: $C0_H$
MODNUM	[31:16]	r	**Module Number Value** This bit field defines the module identification number for the MultiCAN : $002B_H$

20.4.2 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.

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 20-6**.
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 20-6**.
PANAR2	[31:24]	rwh	**Panel Argument 2** See **Table 20-6**.
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 20-6 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 (x = 0-1). Bit 7 of PANAR2 (ERR) reports the success of the operation: 0 Initialization was successful 1 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 20-6 Panel Commands (cont'd)

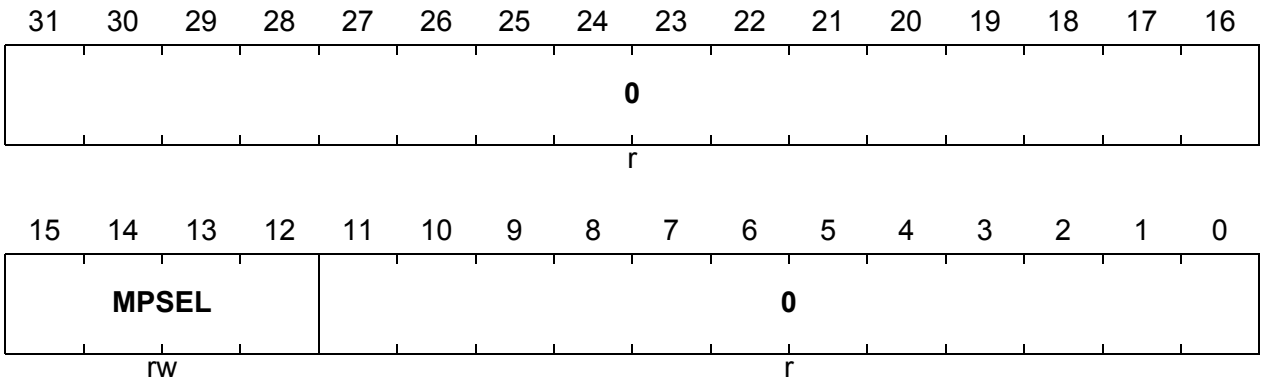
PANCMD	PANAR2	PANAR1	Command Description
03_H	**Argument:** List Index **Result:** Bit 7: ERR Bit 6-0: undefined	**Result:** Message Object Number	**Dynamic Allocate** Allocate the first message object of the list of unallocated objects to the selected list. The message object is appended to the end 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 20-6 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.

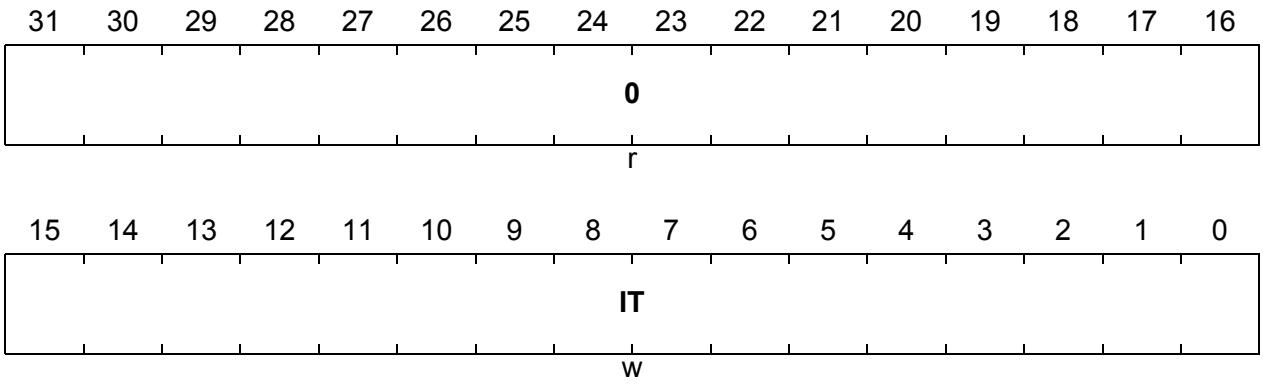
MCR
Module Control Register **($1C8_H$)** **Reset Value: $0000\ 0000_H$**



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 20-17** on **Page 20-38**.
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$_H$



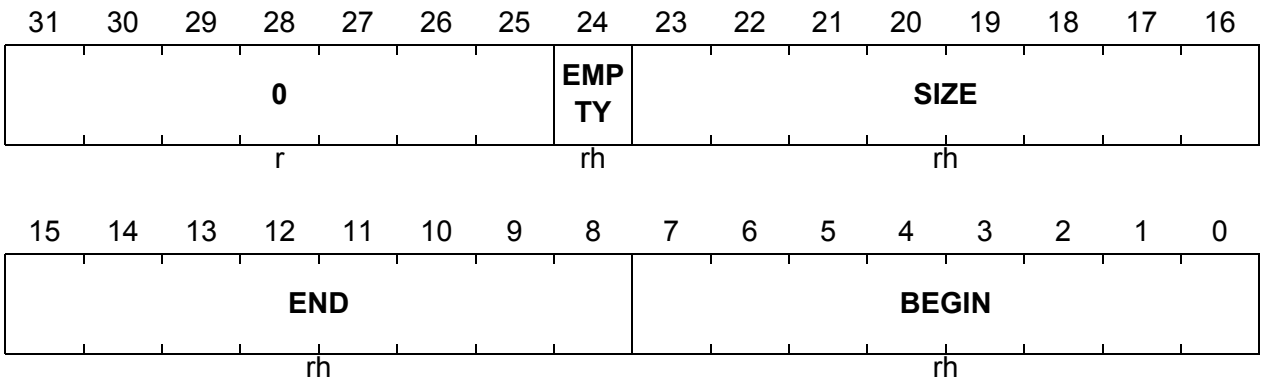
Field	Bits	Type	Description
IT	[15:0]	w	**Interrupt Trigger** Writing a 1 to IT[n] (n = 0-15) generates an interrupt request on interrupt output line INT_O[n]. Writing a 0 to IT[n] has no effect. Bit field IT is always read as 0. Multiple interrupt requests can be generated with a single write operation 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
- LIST[7:3] are not associated to a CAN node (free lists)

LIST0
List Register 0 **(100_H)** **Reset Value: 007F 7F00$_H$**
LISTk (k = 1-7)
List Register k **(100_H+k*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 k.
END	[15:8]	rh	**List End** END indicates the number of the last message object in list k.
SIZE	[23:16]	rh	**List Size** SIZE indicates the number of elements in the list k. SIZE = number of list elements - 1 SIZE = 0 indicates that list k is empty.

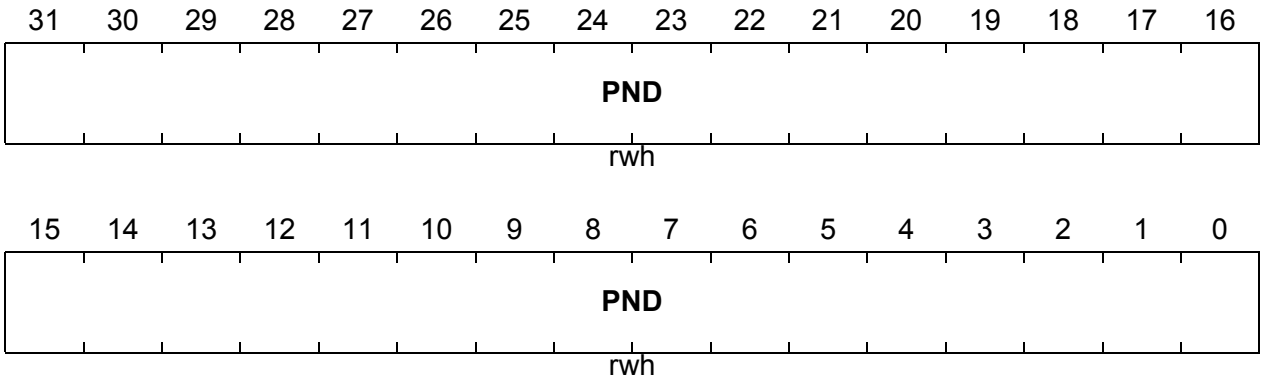
Field	Bits	Type	Description
EMPTY	24	rh	**List Empty Indication** 0_B At least one message object is allocated to list k. 1_B No message object is allocated to the list k. List k is empty.
0	[31:25]	r	**Reserved** Read as 0.

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.TXIPND or MOIPRn.RXIPND 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 k.

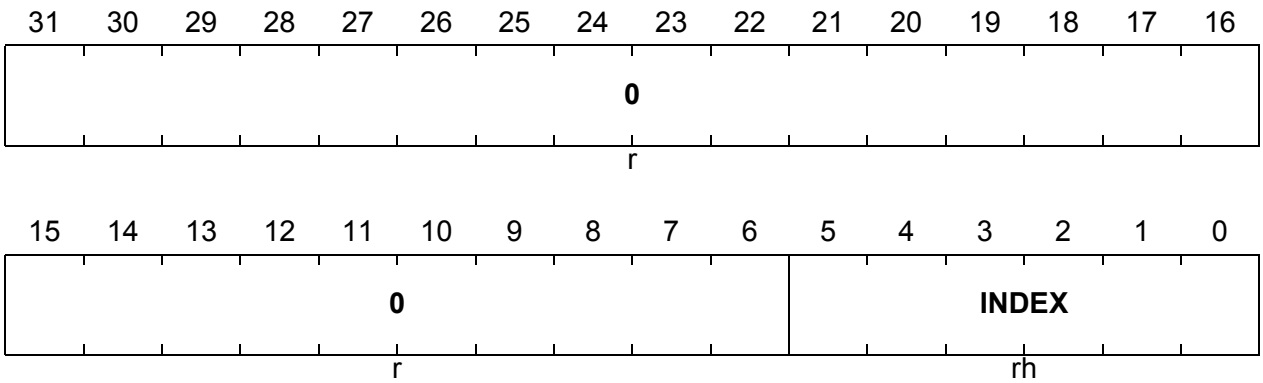
MSPNDk (k = 0-7)
Message Pending Register k **(120_H+k*4_H)** **Reset Value: 0000 0000_H**



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 MSGPND 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.

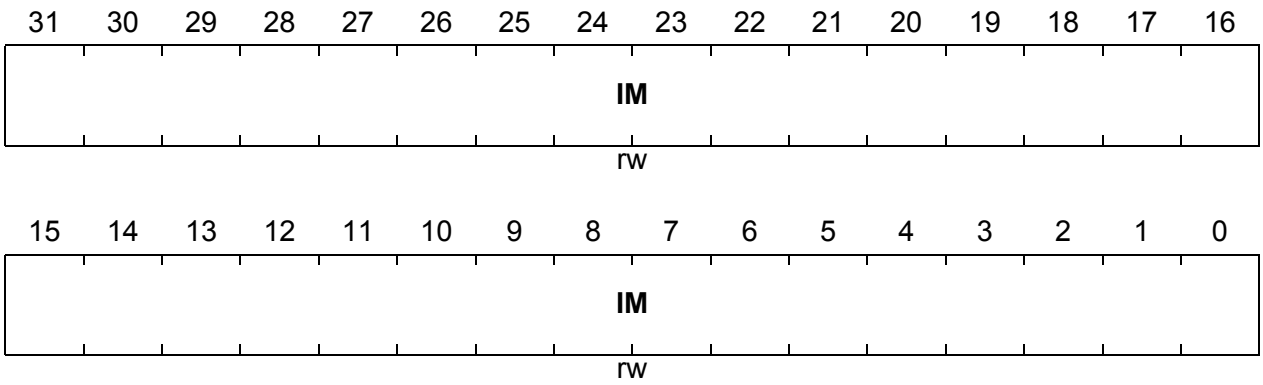
MSIDk (k = 0-7)
Message Index Register k **(140_H+k*4_H)** **Reset Value: 0000 0020_H**



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 MSPNDn for which the corresponding Index Mask bits are set contribute to the calculation of the Message Index.

20.4.3 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 = 0-1)
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

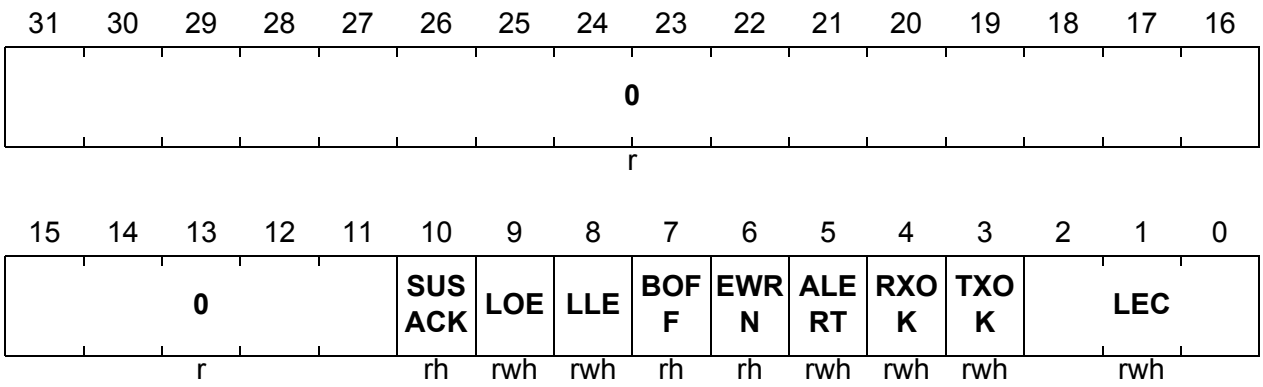
Field	Bits	Type	Description
INIT	0	rwh	**Node Initialization** 0_B Clearing bit INIT enables the participation of the node in the CAN traffic. If the CAN node is in the bus-off state, the ongoing bus-off recovery (which does not depend on the INIT bit) is continued. With the end of the bus-off recovery 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.
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	**Last Error Code 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 can 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 cleared 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 = 0-1)
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 20-7**.
TXOK	3	rwh	**Message Transmitted Successfully** 0_B No successful transmission since the last (most recent) clearance of the flag. 1_B A message has been transmitted successfully (error-free and acknowledged by at least another node). TXOK must be cleared by software (write 0). Writing 1 has no effect.
RXOK	4	rwh	**Message Received Successfully** 0_B No successful reception since the last (most recent) clearance of the flag. 1_B A message has been received successfully. RXOK must be cleared 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 cleared 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 the last (most recent) clearance of the flag. 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 cleared by software (write 0). Writing 1 has no effect.
LOE	9	rwh	**List Object Error** 0_B No List Object Error since the last (most recent) clearance of the flag 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 cleared 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 20-7 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.

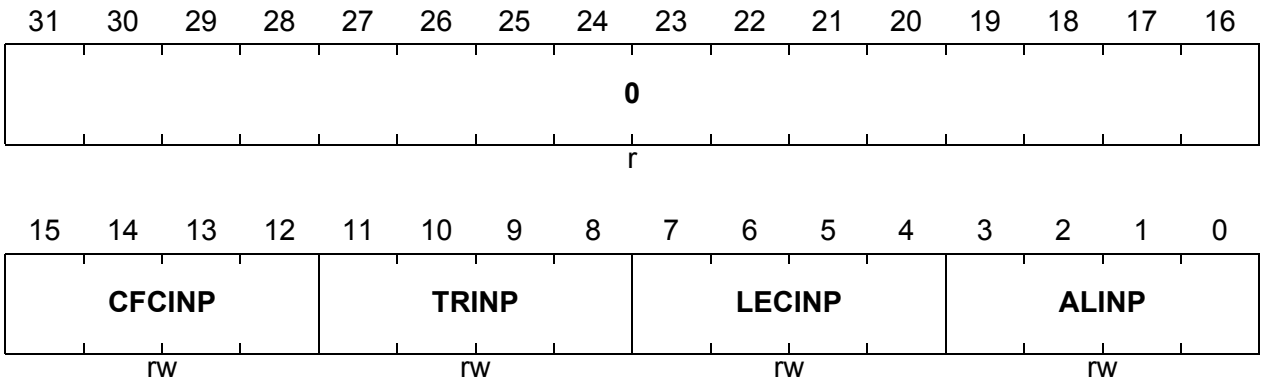
Table 20-7 Encoding of the LEC Bit Field (cont'd)

LEC Value	Signification
110_B	**CRC Error:** The CRC checksum of the received message was incorrect.
111_B	**CPU write to LEC:** Whenever the CPU writes the value 111B to LEC, it takes the value 111B. 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 20-23 for more CAN node interrupt details.

NIPRx (x = 0-1)
Node x Interrupt Pointer Register

(208_H+x*100_H) **Reset Value: 0000 0000_H**

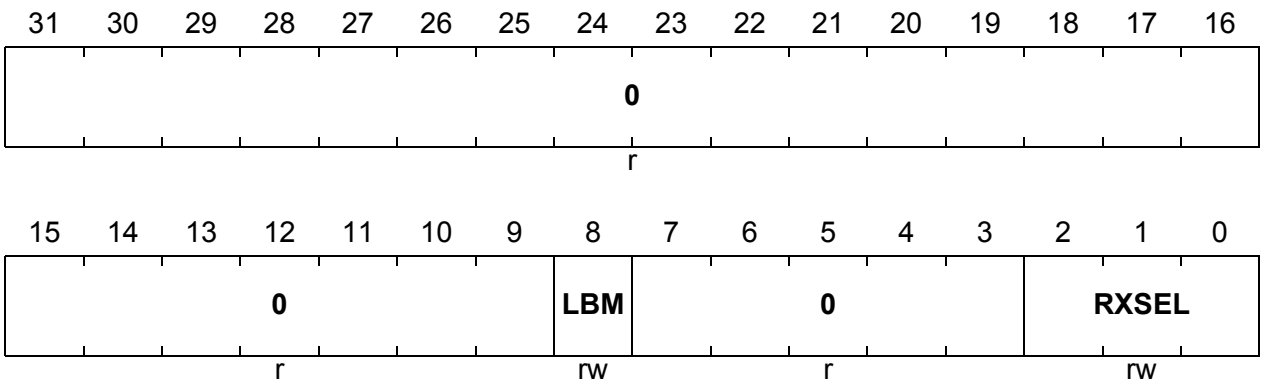


Field	Bits	Type	Description
ALINP	[3:0]	rw	**Alert Interrupt Node Pointer** ALINP selects the interrupt output line INT_Om (m = 0-15) for an alert interrupt of CAN Node x. 0000_B Interrupt output line INT_O0 is selected. 0001_B Interrupt output line INT_O1 is selected. $\ldots_B$ … 1110_B Interrupt output 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 a LEC 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.

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.

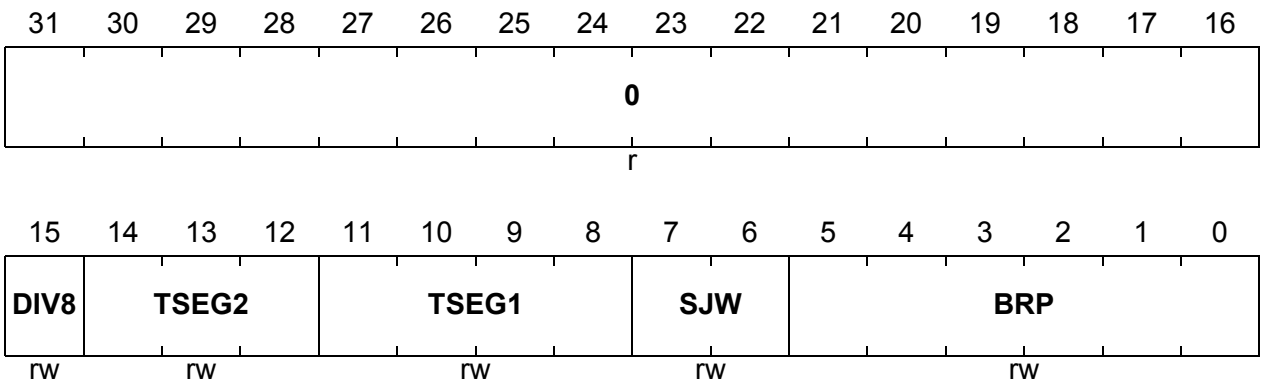
NPCRx (x = 0-1)
Node x Port Control Register **($20C_H$+x*100_H)** **Reset Value: 0000 0000_H**



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 TC1766, only specific combinations of RXSEL are available (see also **"Receive Input Selection" on Page 20-116**).*
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 = 0-1)
Node x Bit Timing Register **(210_H+x*100_H)** **Reset Value: 0000 0000_H**

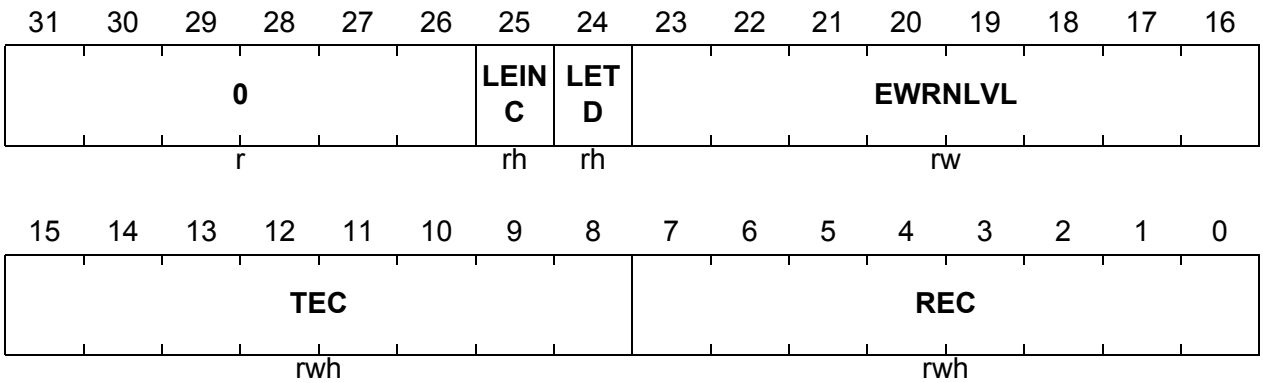


Field	Bits	Type	Description
BRP	[5:0]	rw	**Baud Rate Prescaler** The duration of one time quantum is given by (BRP + 1) clock cycles if DIV8 = 0. The duration of one time quantum is given by 8 × (BRP + 1) clock cycles if DIV8 = 1.
SJW	[7:6]	rw	**(Re) Synchronization Jump Width** (SJW + 1) time quanta 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 = 0-1)
Node x Error Counter Register (214_H+x*100_H) Reset Value: 0060 0000_H

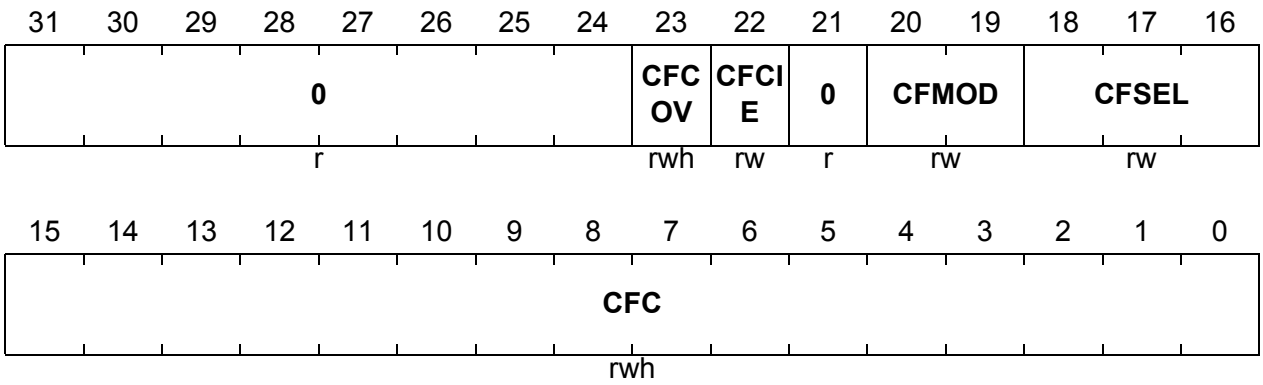


Field	Bits	Type	Description
REC	[7:0]	rwh	**Receive Error Counter** Bit field REC contains the value of the receive error counter of CAN node x.
TEC	[15:8]	rwh	**Transmit Error Counter** Bit field TEC contains the value of the transmit error counter of CAN node x.
EWRNLVL	[23:16]	rw	**Error Warning Level** 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 = 0-1)
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 20-8. 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. 01_B Time 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 the last (most recent) clearance of the flag. 1_B An overflow has occurred since the last (most recent) clearance of the flag. CFCOV must be cleared by software.
0	21, [31:24]	r	**Reserved** Read as 0; should be written with 0.

Bit Timing Analysis Modes

Table 20-8 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 20-8 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 20-9**)
111_B	Reserved, do not use this combination.

Table 20-9 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.

20.4.4 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/clear CAN transfer related control bits through software.

When reading from the MOCTRn address after reset, the reset value of register MOSTATn will always be delivered (see also **Table 20-11**).

MOCTR0
Message Object 0 Control Register ($041C_H$) **Reset Value: $0100\ 0000_H$**
MOCTR63
Message Object 63 Control Register ($0BFC_H$) **Reset Value: $3F3E\ 0000_H$**
MOCTRn (n = 1-62)
Message Object n Control Register
($41C_H$+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 clear/set condition for RXPND (see **Table 20-10**).
RESTXPND, SETTXPND	1, 17	w	**Reset/Set Transmit Pending** These bits control the clear/set condition for TXPND (see **Table 20-10**).
RESRXUPD, SETRXUPD	2, 18	w	**Reset/Set Receive Updating** These bits control the clear/set condition for RXUPD (see **Table 20-10**).

Field	Bits	Type	Description
RESNEWDAT, SETNEWDAT	3, 19	w	**Reset/Set New Data** These bits control the clear/set condition for NEWDAT (see **Table 20-10**).
RESMSGLST, SETMSGLST	4, 20	w	**Reset/Set Message Lost** These bits control the clear/set condition for MSGLST (see **Table 20-10**).
RESMSGVAL, SETMSGVAL	5, 21	w	**Reset/Set Message Valid** These bits control the clear/set condition for MSGVAL (see **Table 20-10**).
RESRTSEL, SETRTSEL	6, 22	w	**Reset/Set Receive/Transmit Selected** These bits control the clear/set condition for RTSEL (see **Table 20-10**).
RESRXEN, SETRXEN	7, 23	w	**Reset/Set Receive Enable** These bits control the clear/set condition for RXEN (see **Table 20-10**).
RESTXRQ, SETTXRQ	8, 24	w	**Reset/Set Transmit Request** These bits control the clear/set condition for TXRQ (see **Table 20-10**).
RESTXEN0, SETTXEN0	9, 25	w	**Reset/Set Transmit Enable 0** These bits control the clear/set condition for TXEN0 (see **Table 20-10**).
RESTXEN1, SETTXEN1	10, 26	w	**Reset/Set Transmit Enable 1** These bits control the clear/set condition for TXEN1 (see **Table 20-10**).
RESDIR, SETDIR	11, 27	w	**Reset/Set Message Direction** These bits control the clear/set condition for DIR (see **Table 20-10**).
0	[15:12], [31:28]	w	**Reserved** Should be written with 0.

Table 20-10 Clear/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	
Write 1	Write 0	Clear 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 ($41C_H$) Reset Value: $0100\ 0000_H$
MOSTAT63
Message Object 63 Status Register ($0BFC_H$) Reset Value: $3F3E\ 0000_H$
MOSTATn (n = 1-62)
Message Object n Status Register
($41C_H+n*20_H$)
Rest 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
PNEXT								PPREV							
rh								rh							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LIST				DIR	TX EN1	TX EN0	TX RQ	RX EN	RTS EL	MSG VAL	MSG LST	NEW DAT	RX UPD	TX PND	RX PND
rh				rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

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 cleared by hardware but must be cleared 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 cleared by hardware but must be cleared 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 the last clearance of the flag. 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 clearing 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 it 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 cleared 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.

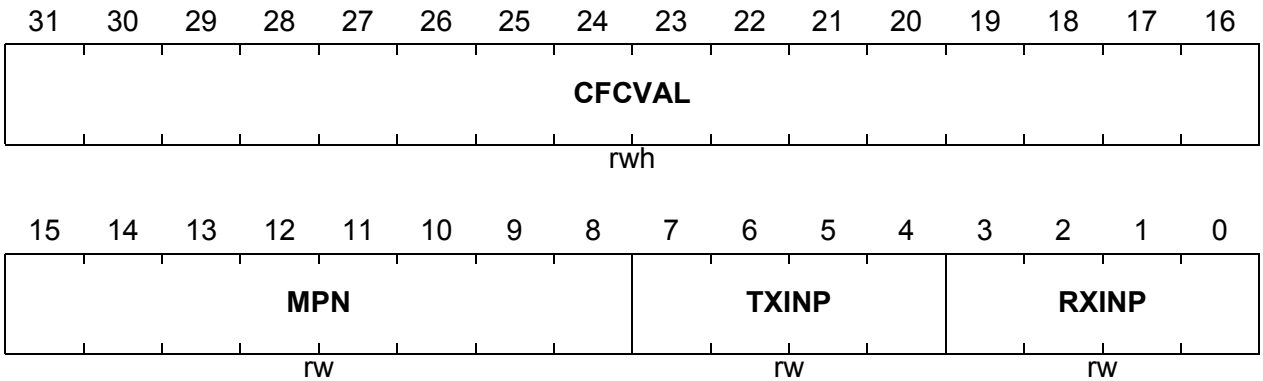
Field	Bits	Type	Description
DIR	11	rh	**Message Direction** 0_B Receive Object selected: With TXRQ = 1, a Remote Frame with the identifier of message object n is scheduled for transmission. On reception of a Data Frame with matching identifier, the message is stored in message object n. 1_B Transmit 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 20-11 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 = 0-63)
Message Object n Interrupt Pointer Register
(408_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 20-38**). 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 20-38**). 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 = 0-63)
Message Object n Function Control Register
(400_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 Other bit combinations are 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.

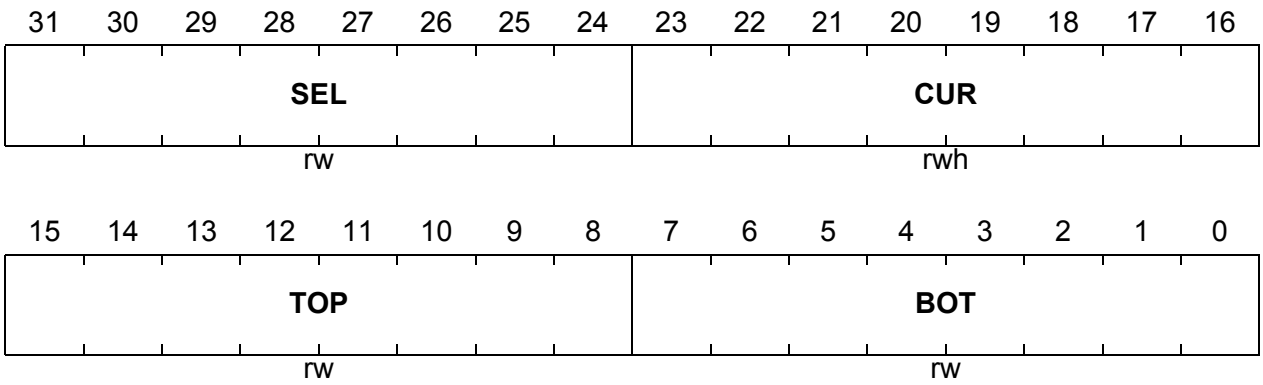
Field	Bits	Type	Description
IDC	9	rw	**Identifier Copy** 0_B The identifier of the gateway source object is not copied. 1_B The identifier of the gateway source object (after storing the received frame in the source) is copied to the gateway destination object. Applicable only to a gateway source object; 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 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 cleared 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 cleared 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.

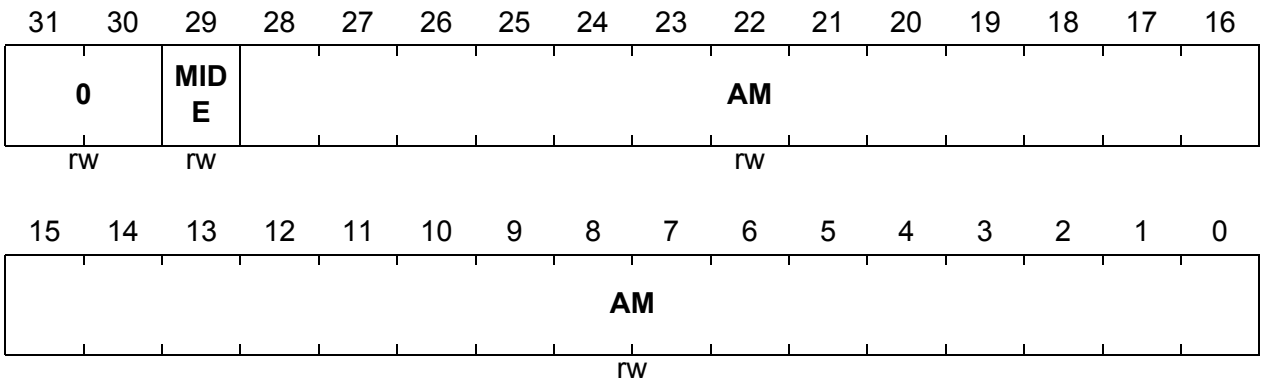
MOFGPRn (n = 0-63)
Message Object n FIFO/Gateway Pointer Register
(404_H+n*20_H) **Reset Value: 0000 0000_H**



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 = 0-63)
Message Object n Acceptance Mask Register
($40C_H$+n*20_H) **Reset Value: 3FFF FFFF$_H$**

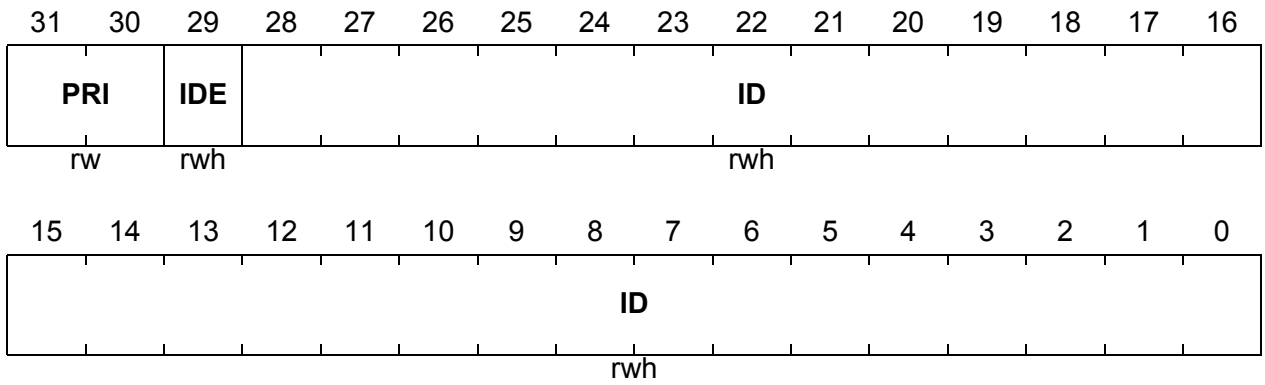


Field	Bits	Type	Description
AM	[28:0]	rw	**Acceptance Mask for Message Identifier** Bit field AM is the 29-bit mask for filtering incoming messages with standard identifiers (AM[28:18]) or extended identifiers (AM[28:0]). For standard identifiers, bits AM[17:0] are “don’t care”.
MIDE	29	rw	**Acceptance 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 = 0-63)
Message Object n Arbitration Register

(418_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 20-12**). 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

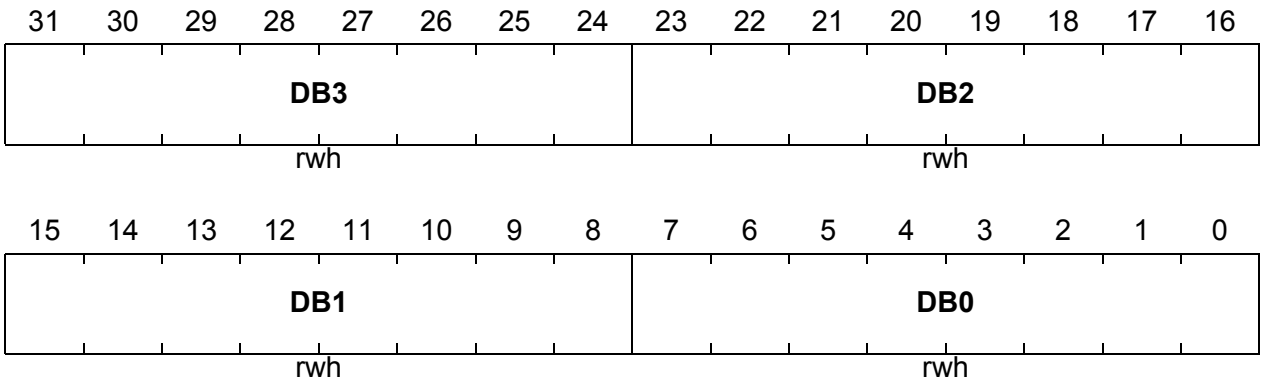
Table 20-12 Transmit Priority of Msg. Objects Based on CAN Arbitration Rules

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 = 0-63)
Message Object n Data Register Low

(410_H+n*20_H) **Reset Value: 0000 0000_H**

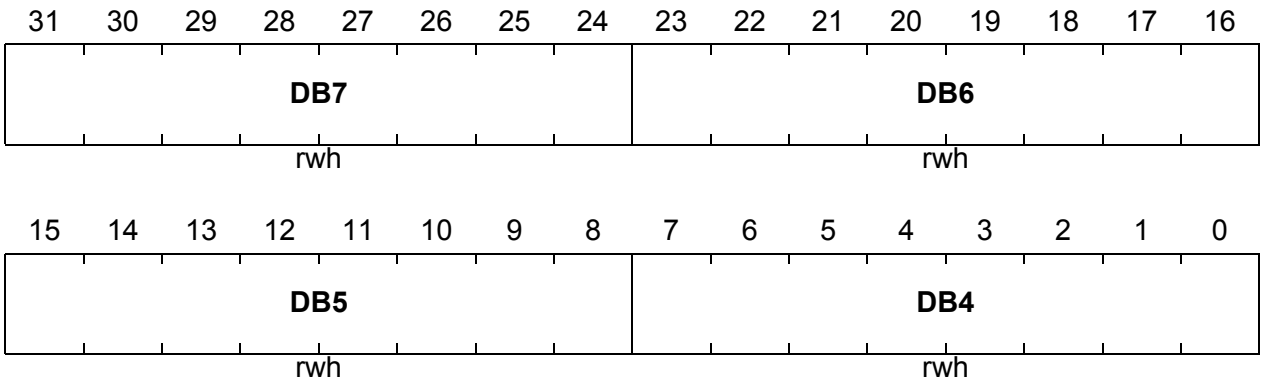


Field	Bits	Type	Description
DB0	[7:0]	rwh	**Data Byte 0 of Message Object n**
DB1	[15:8]	rwh	**Data Byte 1 of Message Object n**
DB2	[23:16]	rwh	**Data Byte 2 of Message Object n**
DB3	[31:24]	rwh	**Data Byte 3 of Message Object n**

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 = 0-63)
Message Object n Data Register High

(414_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**

20.5 MultiCAN Module Implementation

This section describes CAN module interfaces with the clock control, port connections, interrupt control, and address decoding.

20.5.1 Interfaces of the MultiCAN Module

Figure 20-24 shows the TC1766 specific implementation details and interconnections of the MultiCAN module. The four I/O lines of the MultiCAN module (two I/O lines of each CAN node) are connected to I/O lines of Port 3. 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. CAN interrupts are able to trigger DMA transfers and GPTA operations.

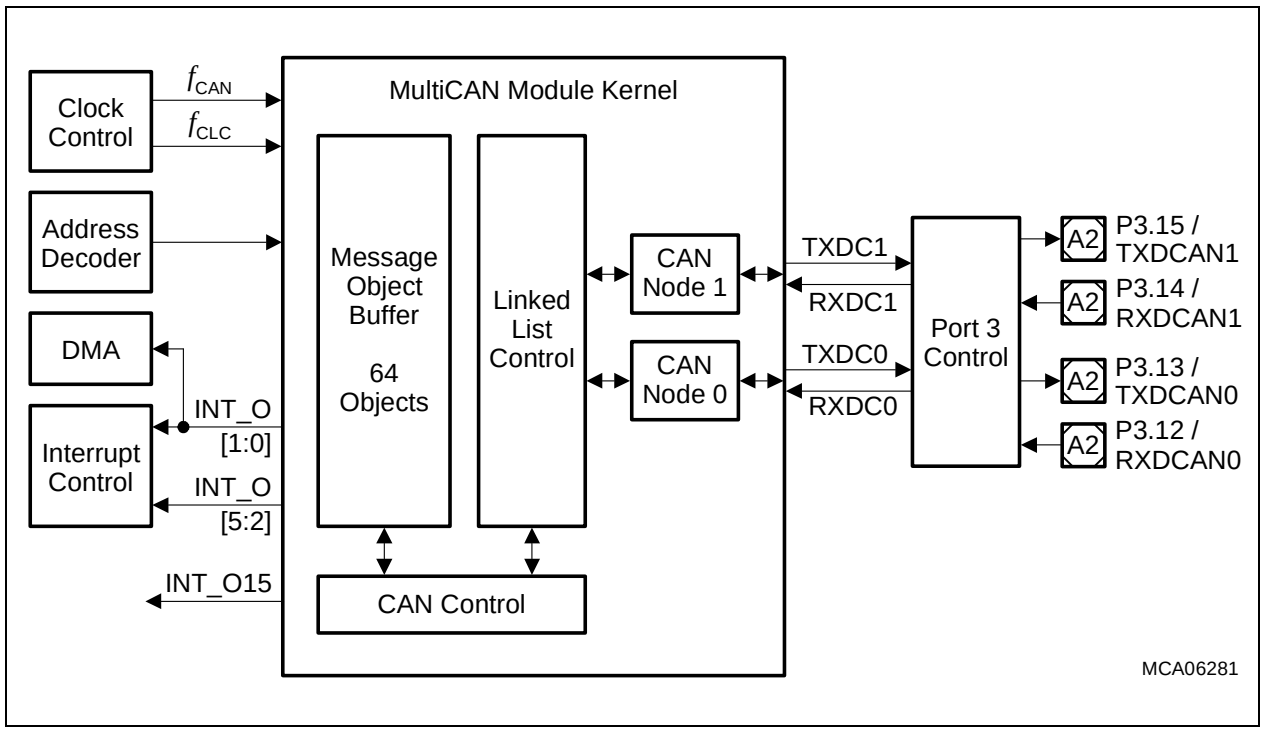


Figure 20-24 CAN Module Implementation and Interconnections

20.5.2 MultiCAN Module External Registers

The registers listed in **Figure 20-25** are not included in the MultiCAN module kernel but must be programmed for proper operation of the MultiCAN module.

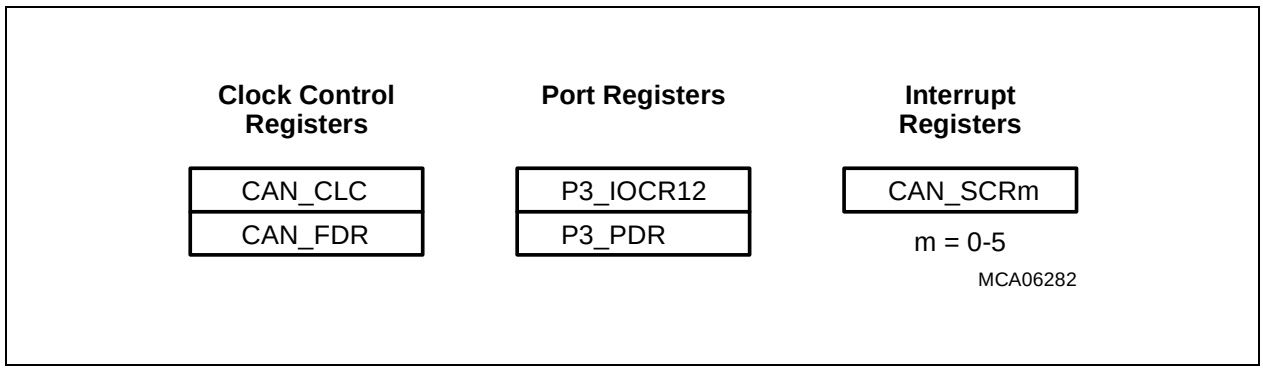


Figure 20-25 CAN Implementation-specific Special Function Registers

20.5.3 Module Clock Generation

As shown in **Figure 20-26**, 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}

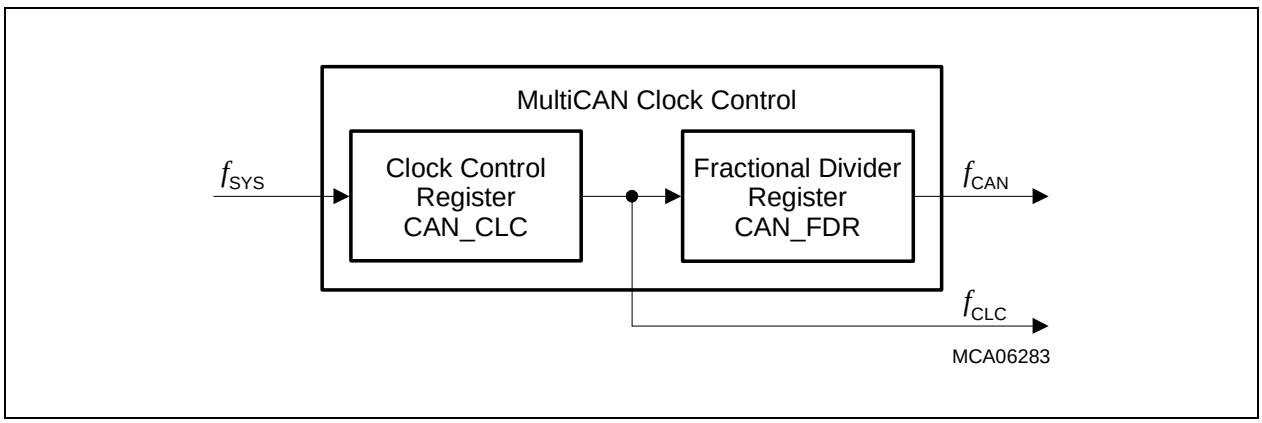


Figure 20-26 MultiCAN Module Clock Generation

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:

$$f_{CAN} = f_{SYS} \times \frac{1}{n} \quad \text{with n = 1024 - CAN\_FDR.STEP} \qquad (20.1)$$

$$f_{CAN} = f_{SYS} \times \frac{n}{1024} \quad \text{with n = 0-1023} \qquad (20.2)$$

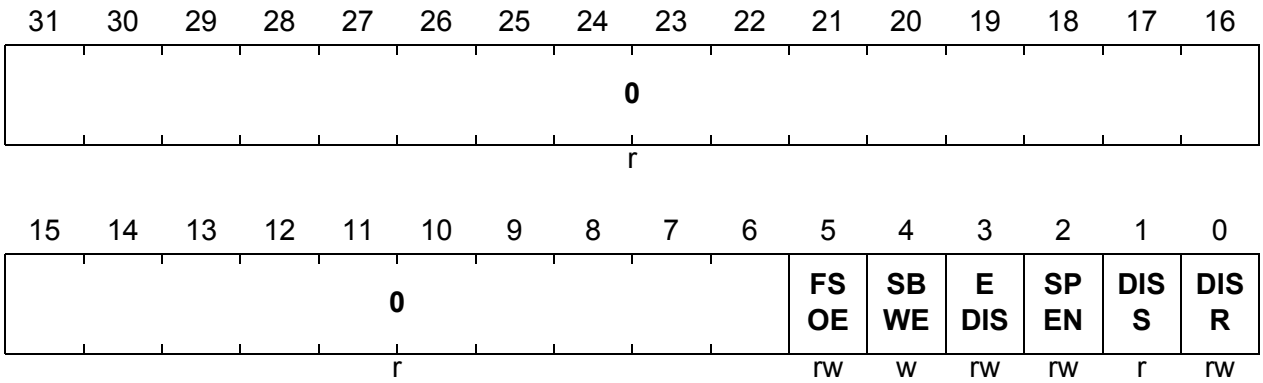
Equation (20.1) applies to normal divider mode (CAN_FDR.DM = 01_B) of the fractional divider. **Equation (20.2)** applies to fractional divider mode (CAN_FDR.DM = 10_B).

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).

20.5.3.1 CAN Clock Control Register

The clock control registers 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	**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: Additional details on the clock control register functionality are described in section **"Clock Control Register CLC" on Page 3-24** of the TC1766 User's Manual System Units part (Volume 1).*

Note: In disabled state, no registers of 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}.

CAN_FDR
CAN Fractional Divider Register ($00C_H$) 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** 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]	rw	**Reserved** Read as 0; should be written with 0.

*Note: Additional details on the fractional divider register functionality are described in section **"Fractional Divider Operation" on Page 3-29** of the TC1766 User's Manual System Units part (Volume 1).*

20.5.4 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 must be executed:

- Input/output function selection (IOCR registers)
- Pad driver characteristics selection for the outputs (PDR registers)

20.5.4.1 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 register P3_OCR12.

Table 20-13 shows how bits and bit fields must be programmed for the required I/O functionality of the CAN I/O lines.

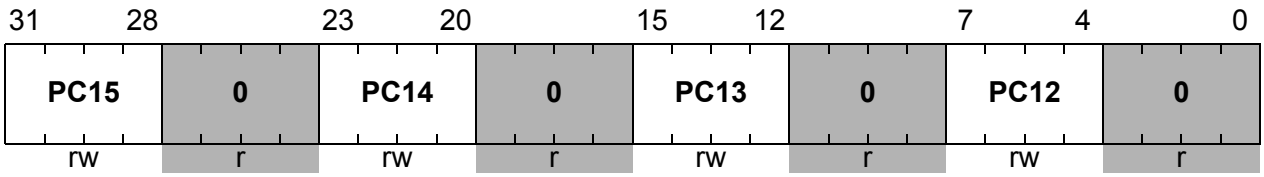
Table 20-13 MultiCAN I/O Control Selection and Setup

Module	Port Lines	Input/Output Control Register Bits[1)]	I/O
CAN	P3.12 / RXDCAN0	P3_IOCR12.PC12 = $0XXX_B$	Input
	P3.13 / TXDCAN0	P3_IOCR12.PC13 = $1X01_B$	Output
	P3.14 / RXDCAN1	P3_IOCR12.PC14 = $0XXX_B$	Input
	P3.15 / TXDCAN1	P3_IOCR12.PC15 = $1X01_B$	Output

1) For possible PCx bit field combinations, see **Table 20-14**.

CAN Related Input/Output Control Registers

P3_IOCR12
Port 3 Input/Output Control Register 12 **Reset Value: 2020 2020$_H$**



Field	Bits	Type	Description
		rw	**Port Output Control for Port 3.[15:12]**[1)] These bit field determines the output port functionality:
PC12	[7:4]		Port input/output control for P3.12 / RXDCAN0
PC13	[15:12]		Port input/output control for P3.13 / TXDCAN0
PC14	[23:20]		Port input/output control for P3.14 / RXDCAN1
PC15	[31:28]		Port input/output control for P3.15 / TXDCAN1

1) For coding of bit field, see **Table 20-14**. Shaded bits and bit fields are “don’t care” for MultiCAN I/O port control.

PCx Coding

Table 20-14 PCx Coding for MultiCAN I/O Lines

PCx[3:0]	I/O	Output Characteristics	Selected Pull-up/Pull-down/ Selected Output Function
$0X00_B$	Input	–	No pull device connected
$0X01_B$			Pull-down device connected
$0X10_B$[1)]			Pull-up device connected
$0X11_B$			No pull device connected
1001_B	Output	Push-pull	Output function ALT1
1101_B		Open-drain	Output function ALT1
1010_B		Push-pull	Output function ALT2
1110_B		Open-drain	Output function ALT2
1011_B		Push-pull	Output function ALT3
1111_B		Open-drain	Output function ALT3

1) This bit field value is the default after reset.

20.5.4.2 Node Receive Input Selection

Additionally to the I/O control selection, as defined in **Table 20-13**, the selection of a CAN node's receive input line requires that bit field RXSEL in its node port control register NPCRx must be set according to **Table 20-15**. Values for NPCRx.RXSEL other than those of **Table 20-15** result in a recessive receive input for node x.

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 = 0-1).

Table 20-15 Receive Input Selection

Receive Input of	Connected to	Selected by
CAN Node 0	P3.12 / RXDCAN0	NPCR0.RXSEL = 000_B
	P3.14 / RXDCAN1	NPCR0.RXSEL = 001_B
CAN Node 1	P3.14 / RXDCAN1	NPCR1.RXSEL = 000_B
	P3.12 / RXDCAN0	NPCR1.RXSEL = 001_B

20.5.4.3 Port 3 Pad Driver Mode Register

The Port 3 pad driver mode register contains bit fields that determine the output driver strength and the slew rate of MultiCAN output lines. The coding of the PDx bit field combinations is shown in **Table 20-16**.

P3_PDR
Port 3 Pad Driver Mode Register **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	0
0	PDCAN			0	PDASC1			0	PDSSC0			0	PDASC0			0	
r	rw			r	rw			r	rw			r	rw			r	

Field	Bits	Type	Description
PDCAN	[30:28]	rw	**Pad Driver Mode for P3.13 and P3.15**[1)] Applies to CAN node 0 and 1 outputs P3.13 / TXDCAN0 and P3.15 / TXDCAN1.

1) For coding of bit field, see **Table 20-16**. Shaded bits and bit fields are "don't care" for MultiCAN I/O port control.

PDx Selection Table

Table 20-16 Pad Output Driver Characteristic Selection (Class A2 Pads)

PDx Bit Field	Driver Strength	Signal Transitions
000_B	Strong driver	Sharp edge[1)]
001_B		Medium edge[1)]
010_B		Soft edge[1)]
011_B	Weak driver	–
100_B	Medium driver	Sharp edge
101_B		Medium edge
110_B		Soft edge
111_B	Weak driver	–

1) In strong driver mode, the output driver characteristics of class A2 pads can be additionally controlled by the temperature compensation logic.

20.5.4.4 DMA Request Outputs

The interrupt output lines INT_O0 to INT_O1 of the MultiCAN module can be used as a DMA requestor and are able to trigger DMA transfers. INT_O[1:0] are connected to the DMA controller as shown in **Table 20-17**.

Table 20-17 CAN-to-DMA Request Connections

DMA Channel	Connected to CAN Interrupt Output	Selected in DMA Controller by programming
06	INT_O0	CHCR06.PRSEL = 011_B
07	INT_O1	CHCR07.PRSEL = 010_B

20.5.5 Interrupt Control

The interrupt control logic in the MultiCAN module uses an interrupt compressing scheme that allows high flexibility in interrupt processing. There are 136 hardware interrupt sources and one software interrupt source available:

- CAN node interrupts:
  – Four different interrupt sources for each of the two CAN nodes = 8 interrupt sources
- Message object interrupts:
  – Two interrupt source for each message object = 128 interrupt sources
- One software initiated interrupt (register MITR)

Each of the 136 hardware initiated interrupt sources is controlled by a 4-bit interrupt pointer that directs the interrupt source to one of the six interrupt outputs INT_Om (m = 0-5). This makes it possible to connect more than one interrupt source to one interrupt output line. The interrupt wiring matrix shown in **Figure 20-27** 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 a 72-input OR gate which is connected to all interrupt pointer demultiplexer outputs which are selected by an identical 4-bit pointer value.

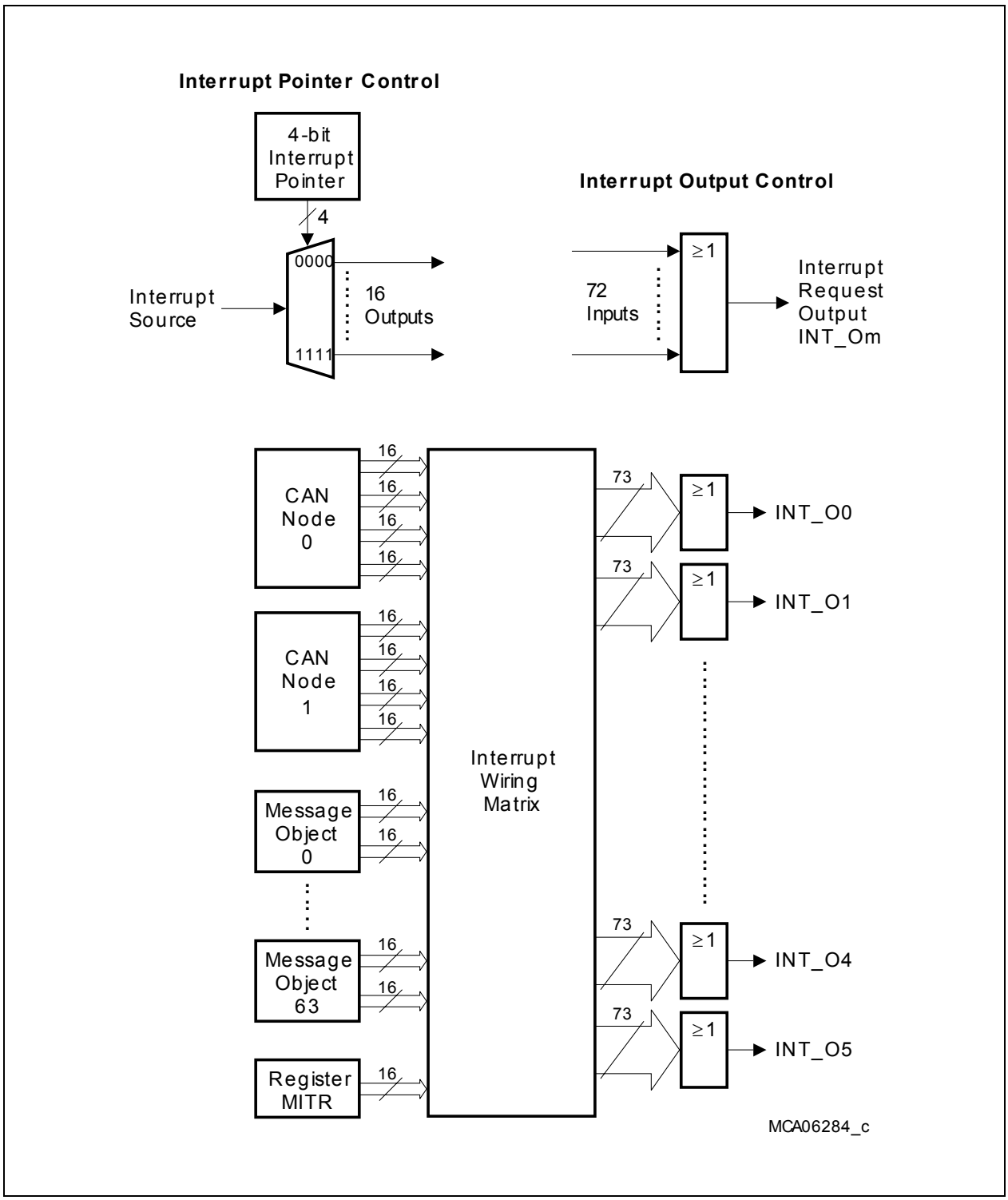
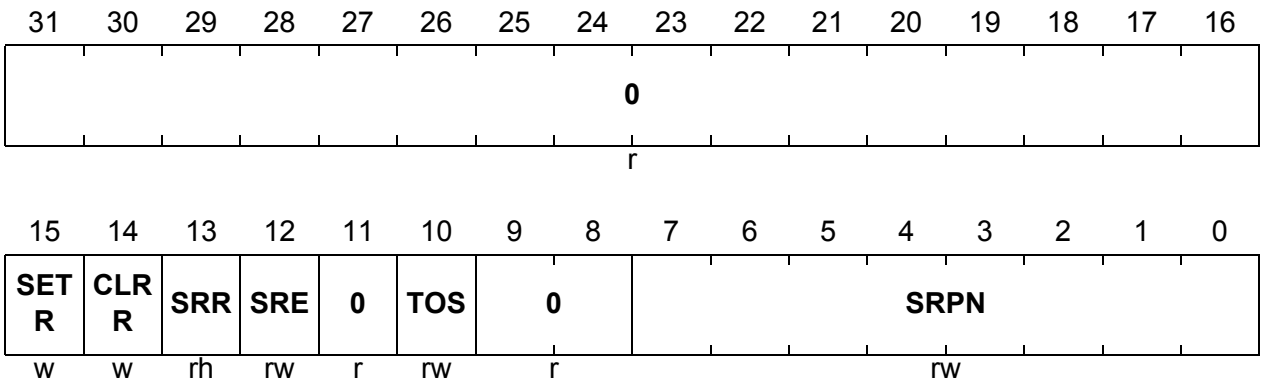


Figure 20-27 Interrupt Compressor

20.5.5.1 CAN Service Request Control Register

Each of the six interrupt outputs INT_Om of the MultiCAN module is controlled by its service request control registers.

CAN_SRCm (m = 0-5)
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 12-3** of the TC1766 User's Manual System Units part (Volume 1).*

Some of the six interrupt outputs of the MultiCAN module can be used to trigger operations in the DMA controller.

20.5.6 MultiCAN Module Register Address Map

In addition to the MultiCAN register address map from Page 20-56, the complete MultiCAN module register address map of Figure 20-28 also shows the general implementation-specific registers for clock control, module identification, and interrupt service request control and adds the absolute address information.

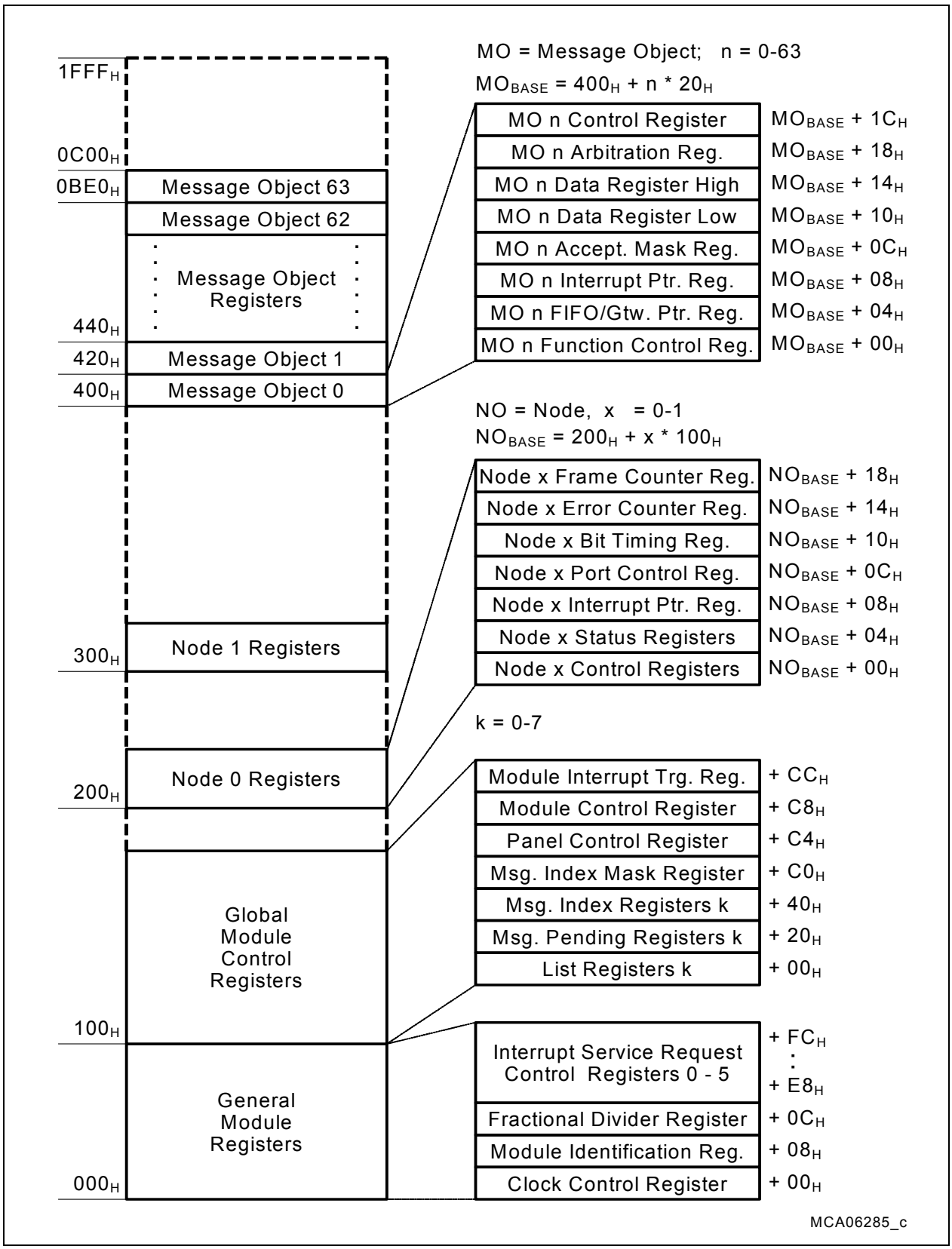


Figure 20-28 MultiCAN Module Register Map

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 21-25**)
- Operation the MLI module (see **Page 21-67**)
- MLI kernel register descriptions (see **Page 21-75**)
- Device implementation-specific descriptions and details (see **Page 21-124**)

*Note: The MLI kernel register names described in **Section 21.3** are referenced in the TC1766 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 21-3**)
- 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.

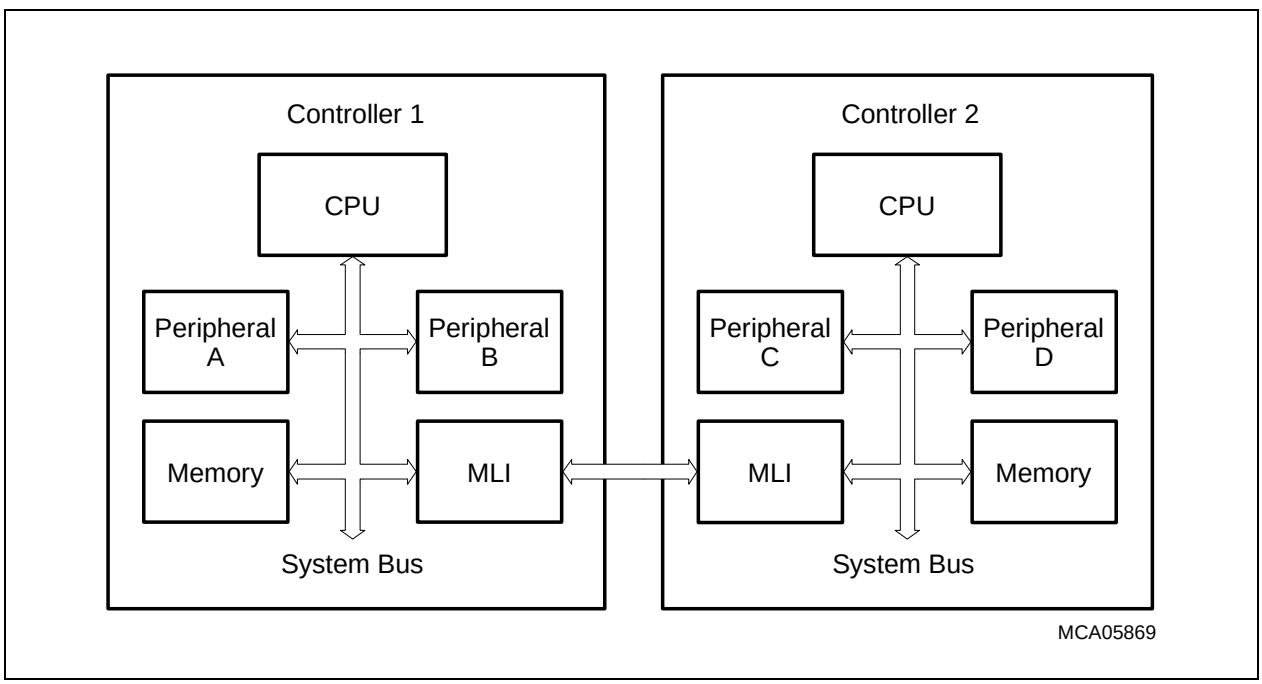


Figure 21-1 Typical Micro Link Interface Connection

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. $f_{MLI}/2$ (= 40 Mbit/s @ 80 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).

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 as well as a Remote Controller 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 bi-directional 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

If the address areas of the four small transfer windows together form a 64 Kbyte address range (aligned to its size), then a single MLI pipe can pass through a device with several MLI modules before being split up in a target device into 4 independent Remote Windows.

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.

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.

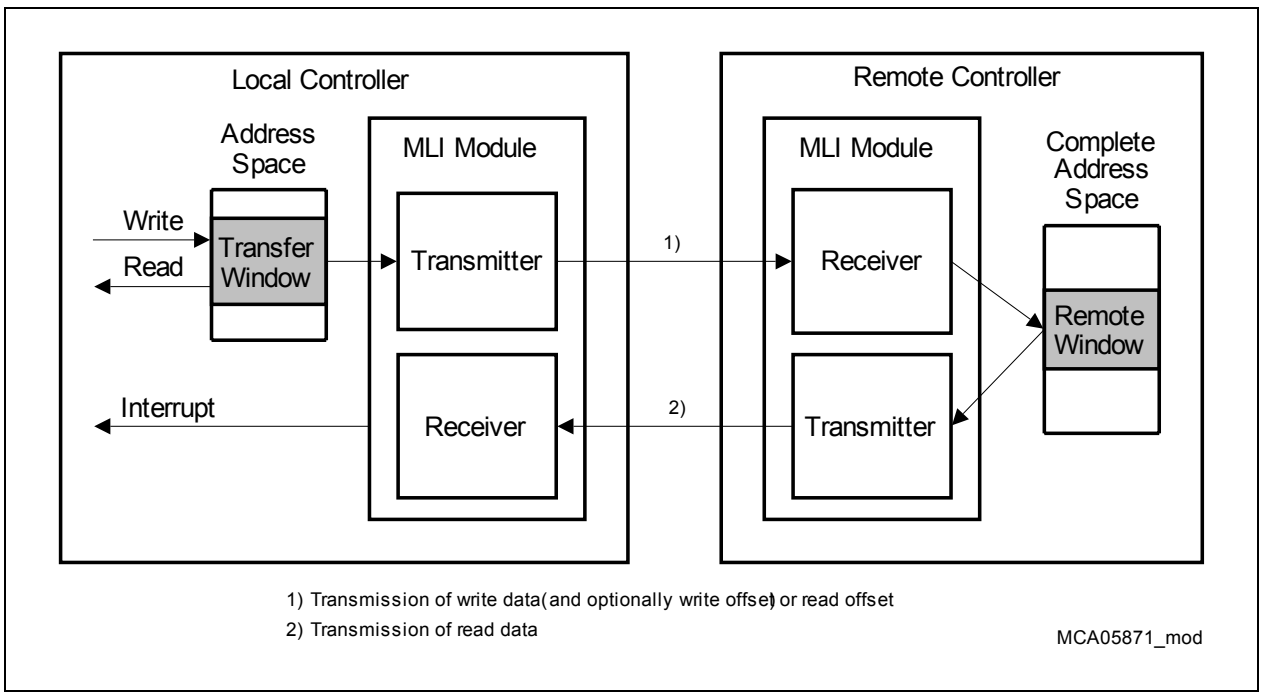


Figure 21-2 MLI Communication Principles

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.

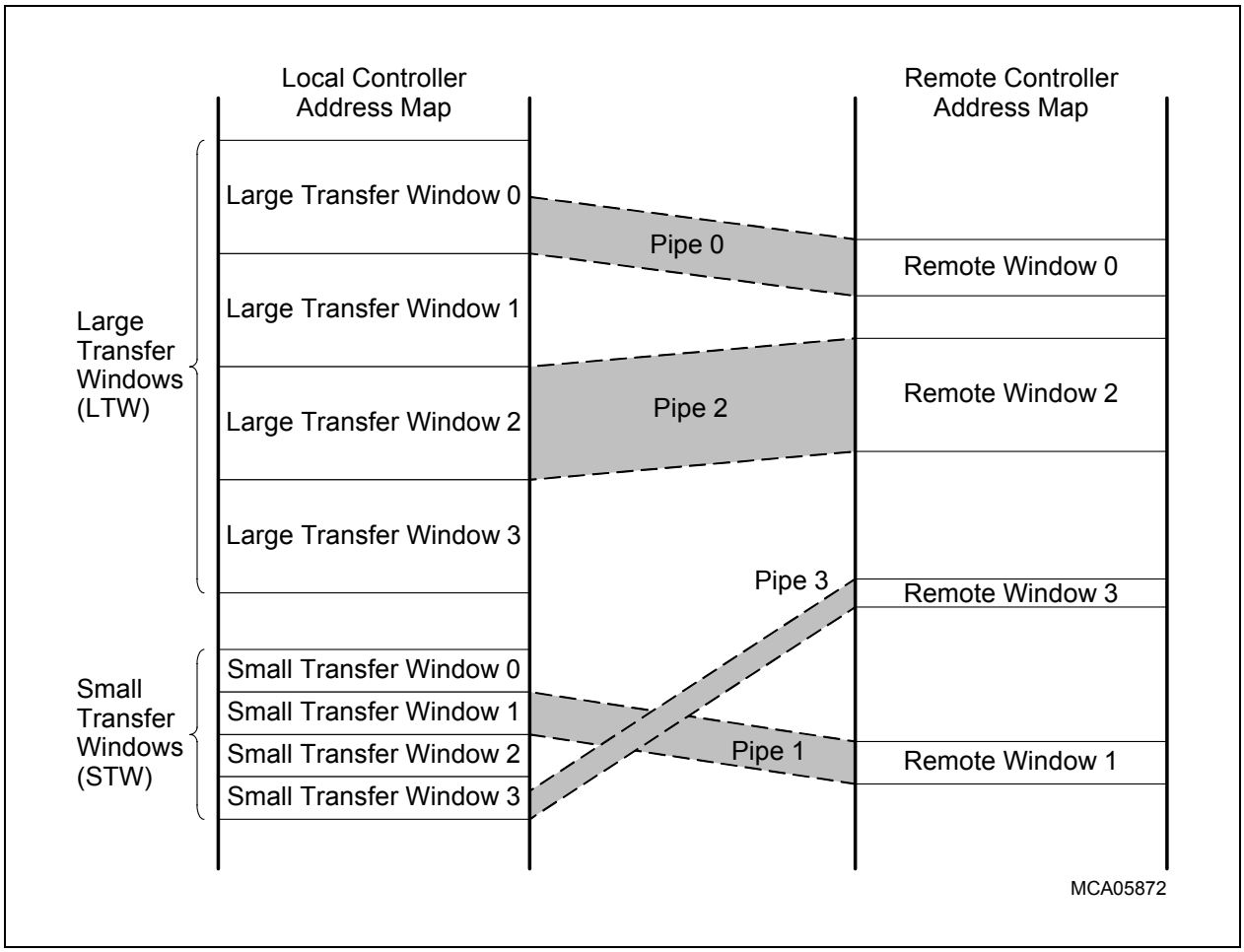


Figure 21-3 Transfer/Remote Window Assignment Example

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).

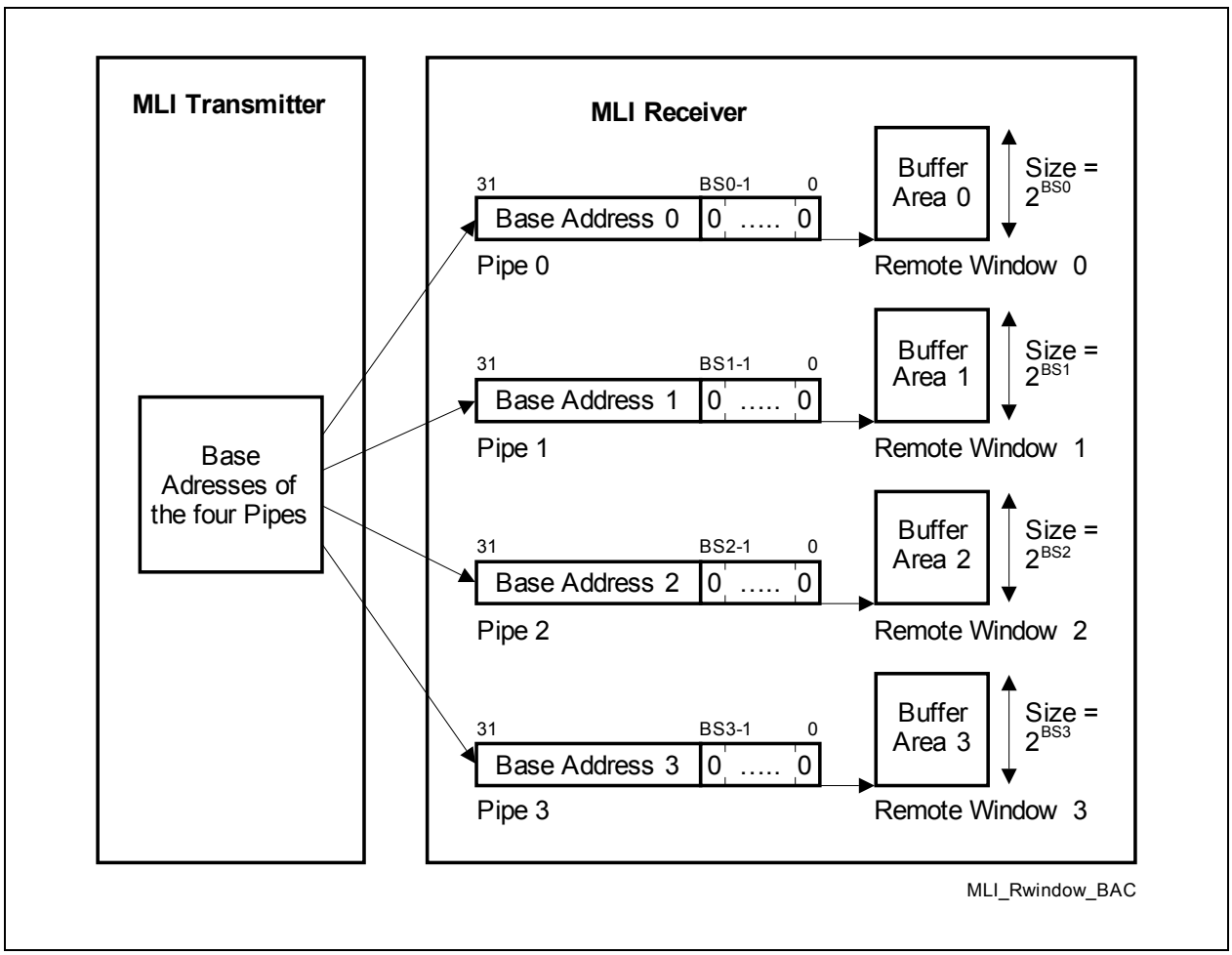


Figure 21-4 Base Address Definition of Remote Windows

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).

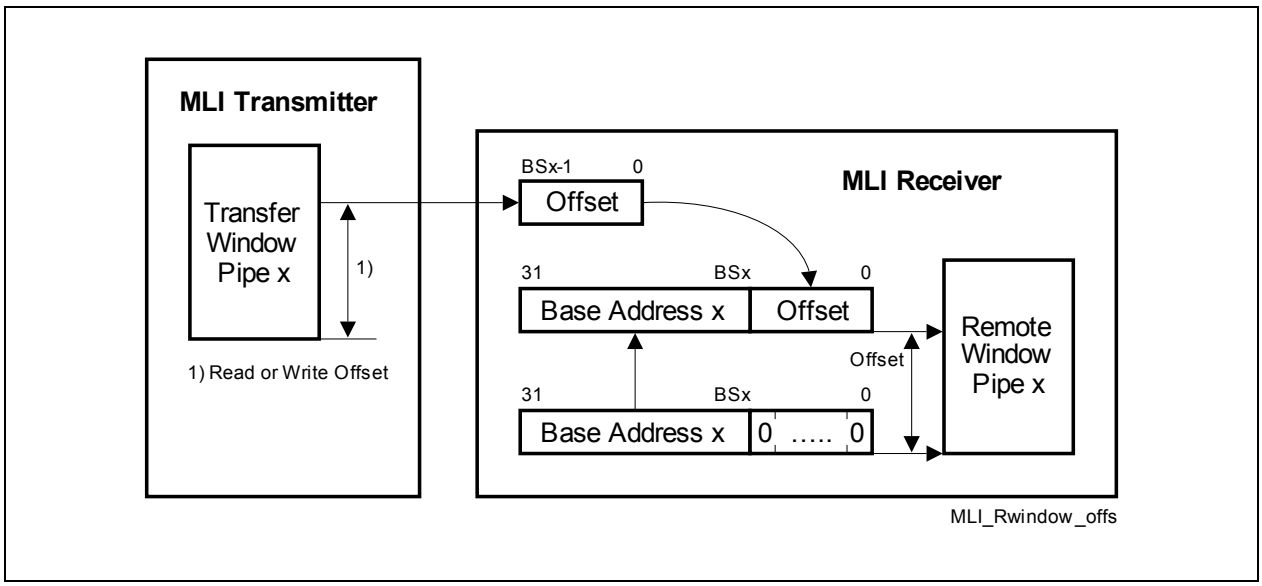


Figure 21-5 Remote Window Address Generation without Address Prediction

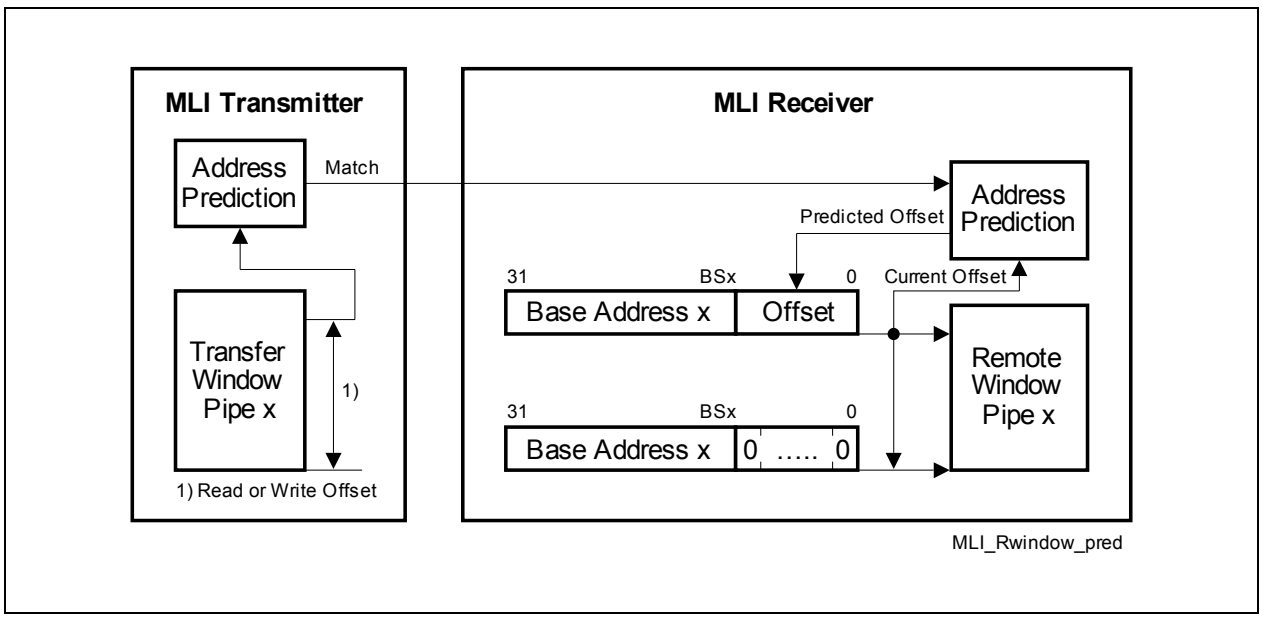


Figure 21-6 Remote Window Address Generation with Address Prediction

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.

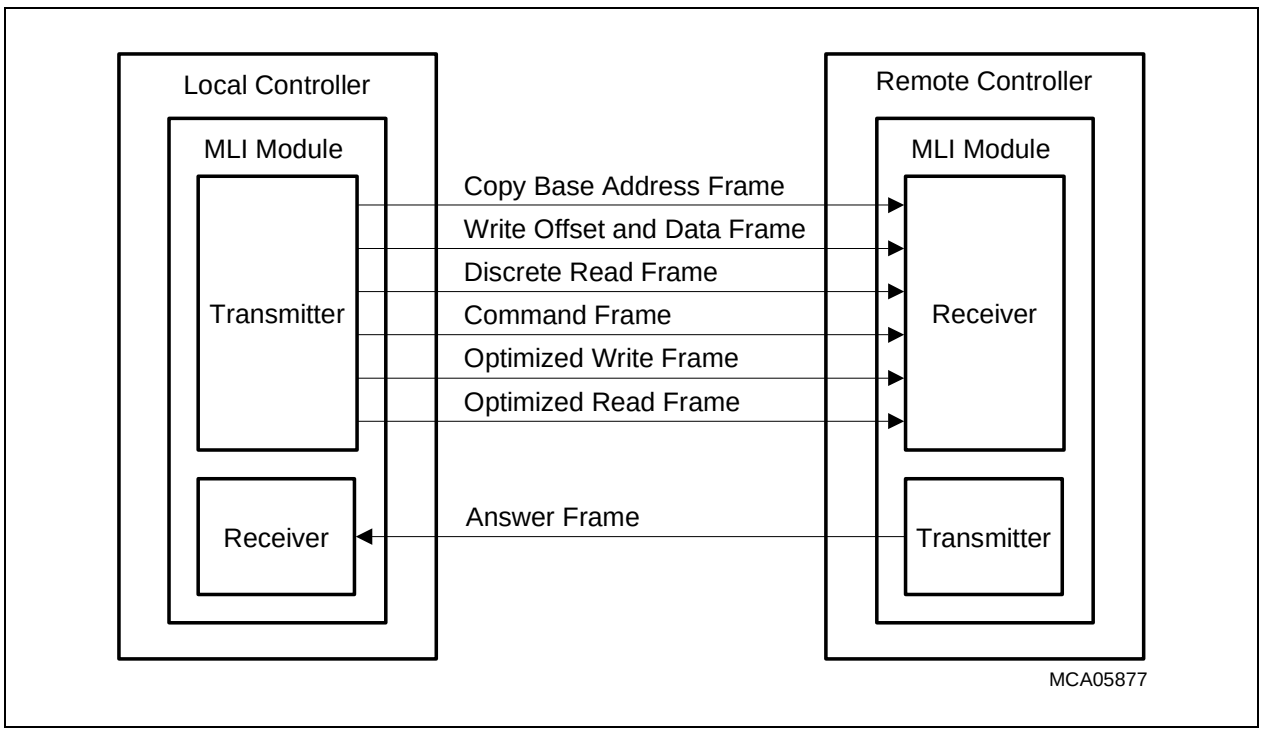


Figure 21-7 Logic Frame Assignment to Local/Remote Controller

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 21-24**), calculated over header and data field bits.

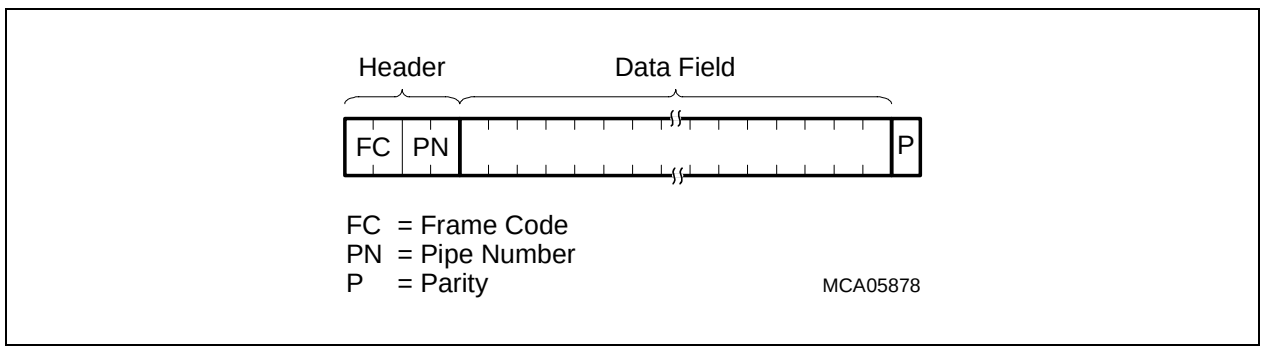


Figure 21-8 General Frame Layout

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 [bit]	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

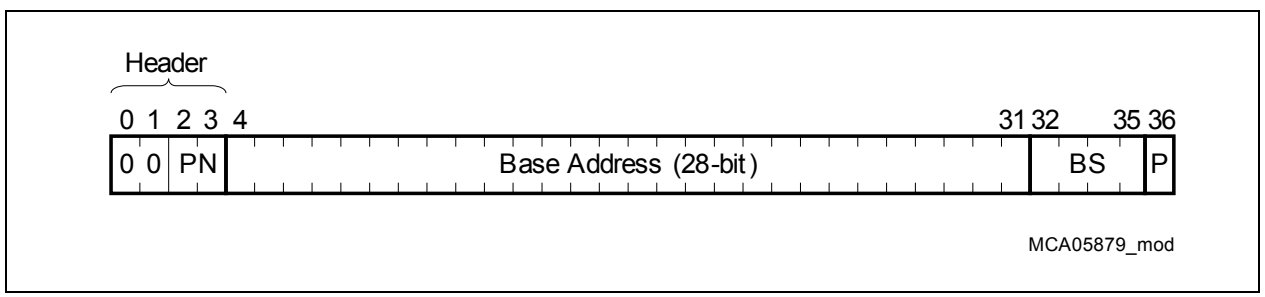


Figure 21-9 Copy Base Address Frame

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 21-26**.

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

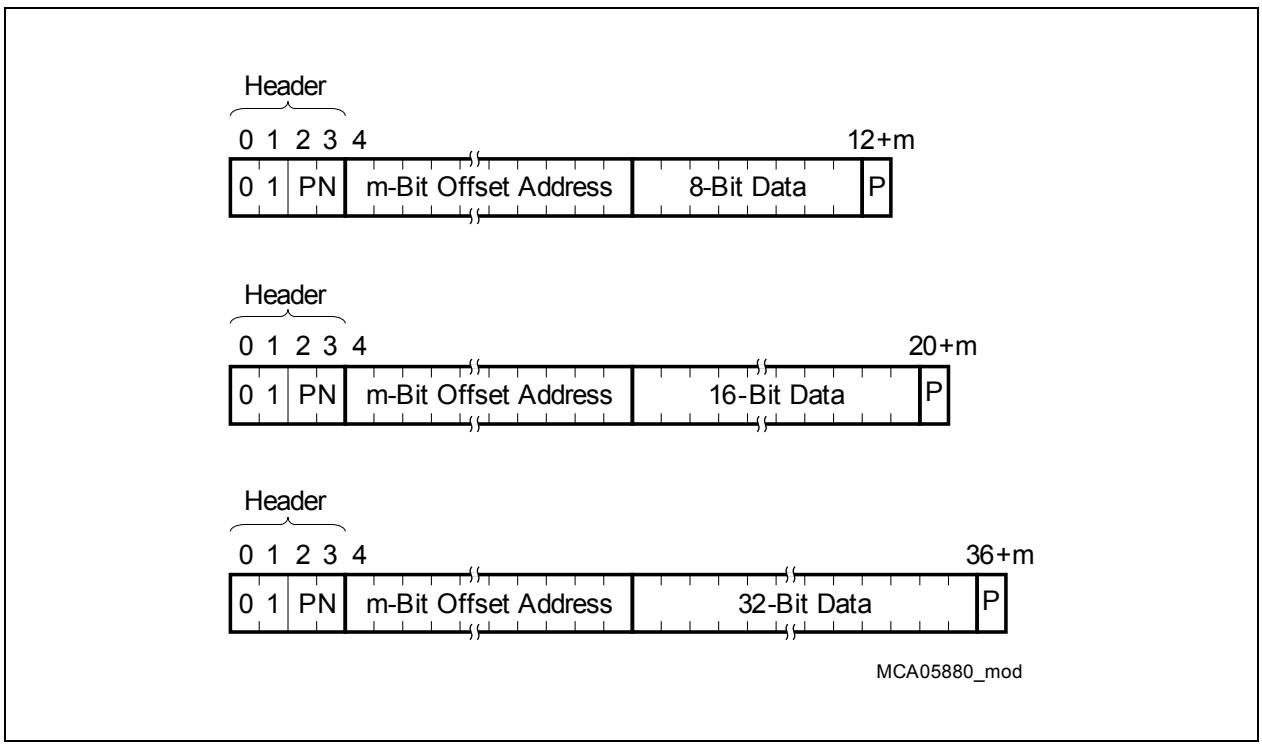


Figure 21-10 Write Offset and Data Frame

More details about the Write Offset and Data Frame handling of the MLI module are described on **Page 21-28**.

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

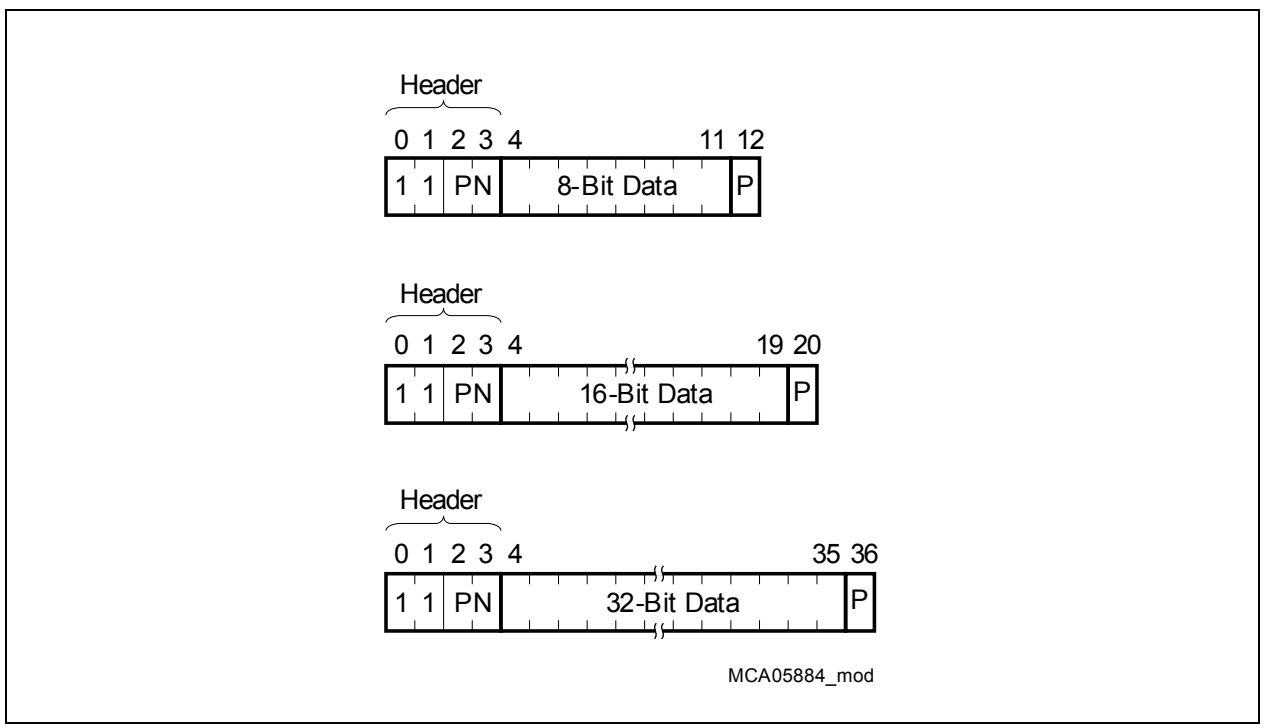


Figure 21-11 Optimized Write Frame

More details about the Optimized Write Frame handling of the MLI module are described on **Page 21-28**.

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

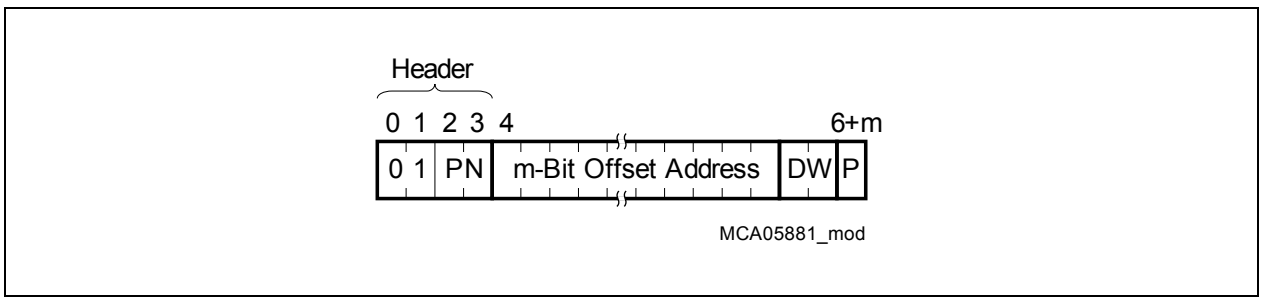


Figure 21-12 Discrete Read Frame

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 described on **Page 21-32**.

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

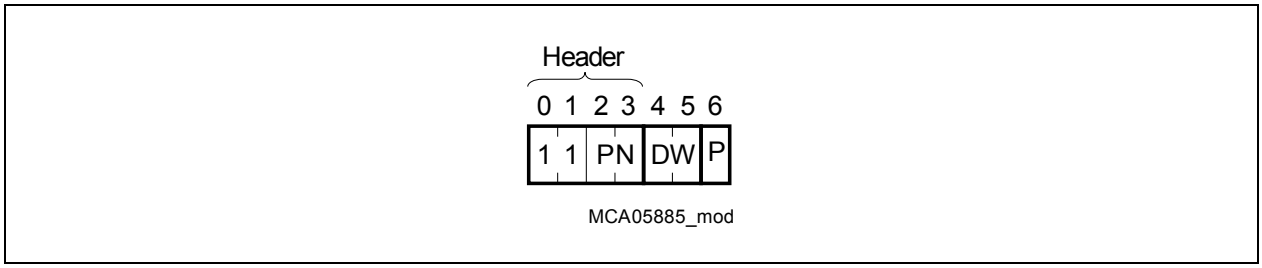


Figure 21-13 Optimized Read Frame

More details about the Optimized Read Frame handling of the MLI module are described on **Page 21-28**.

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

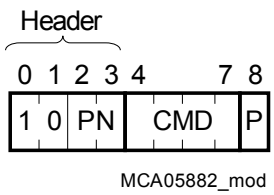


Figure 21-14 Command Frame

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 described on **Page 21-39**.

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

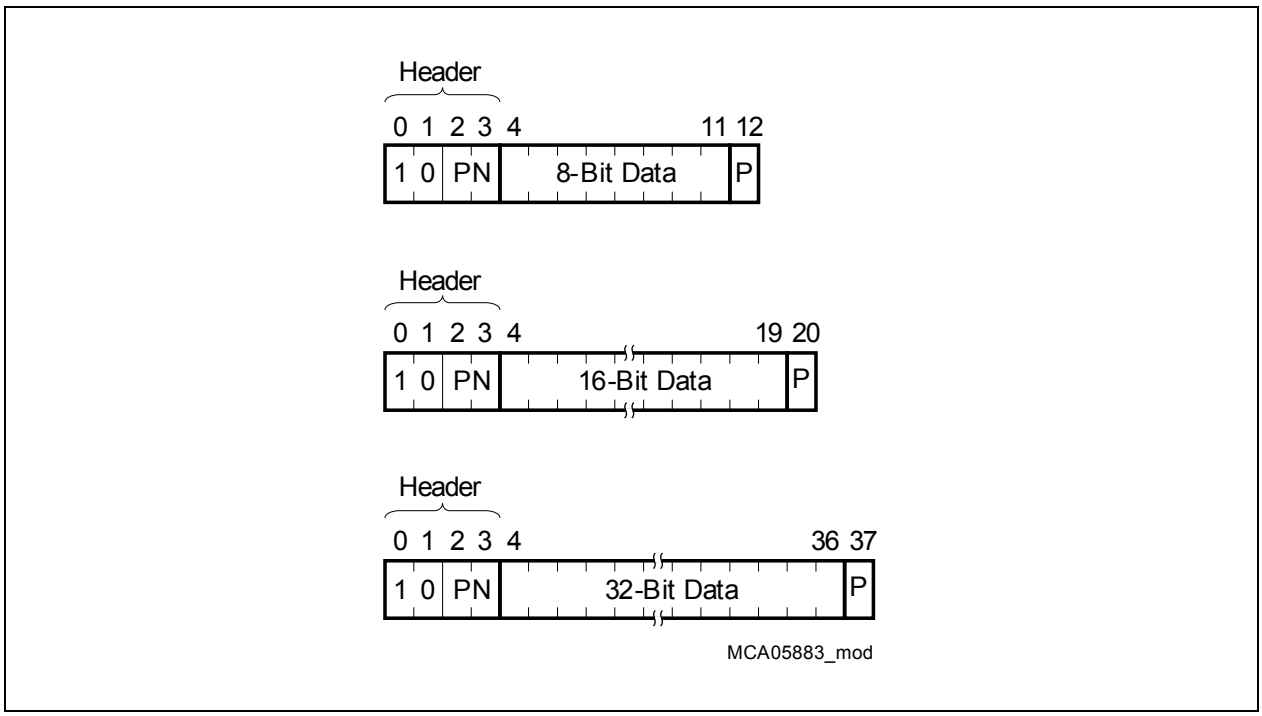


Figure 21-15 Answer Frame

More details about the Answer Frame handling of the MLI module are described on **Page 21-37**.

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”.

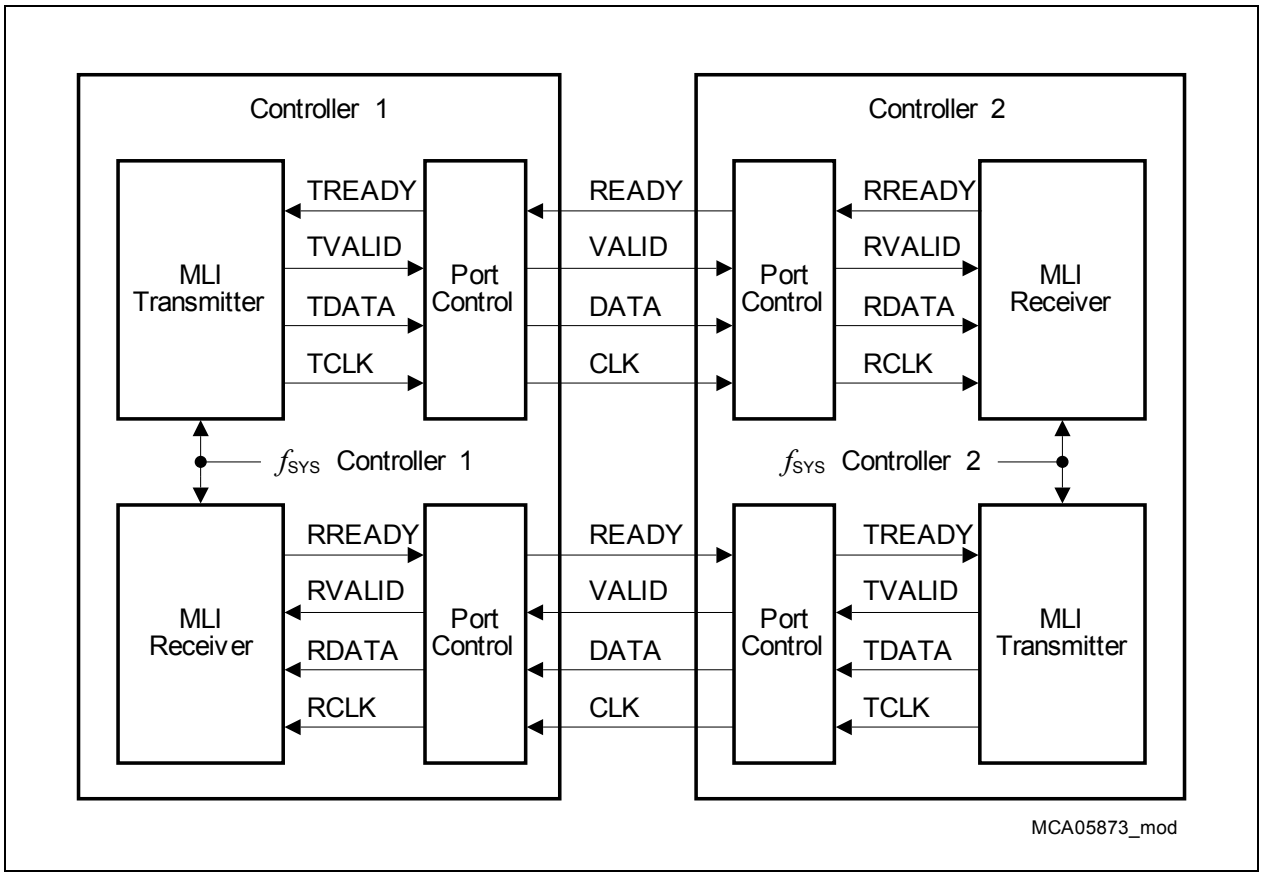


Figure 21-16 Transmitter/Receiver Signal Definitions

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 **Chapter 21.1.3.4**).

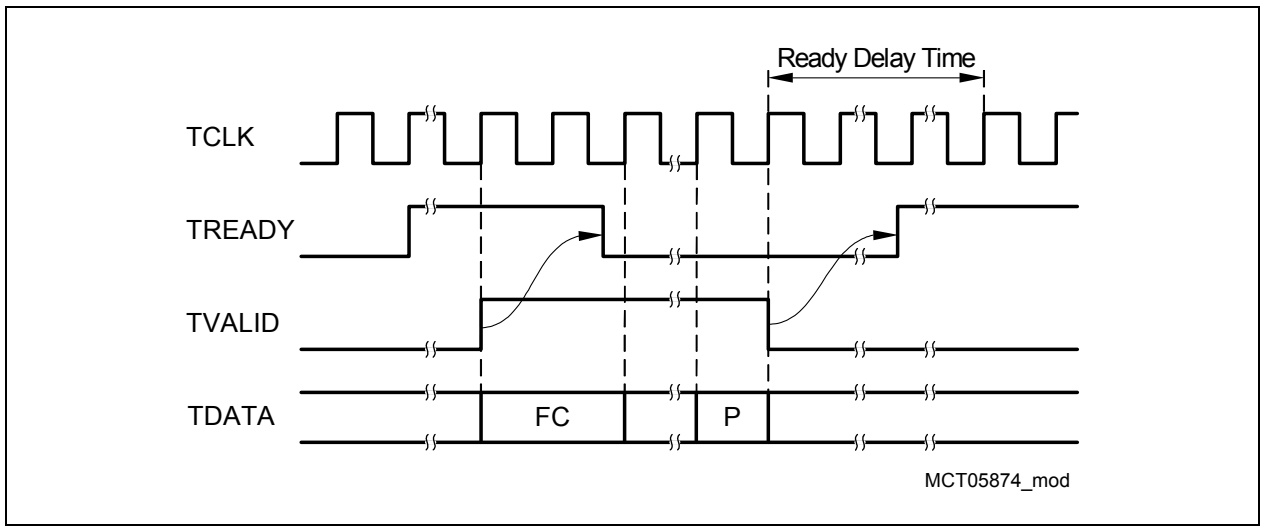


Figure 21-17 MLI Handshake without Error Indication

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.

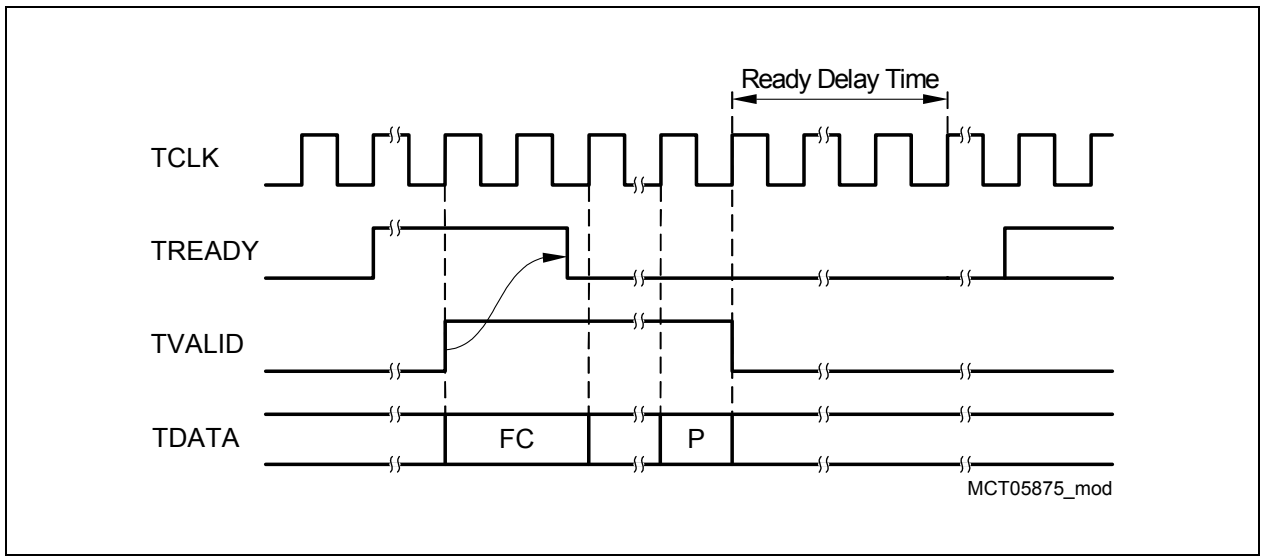


Figure 21-18 MLI Handshake with Parity Error Indication

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.

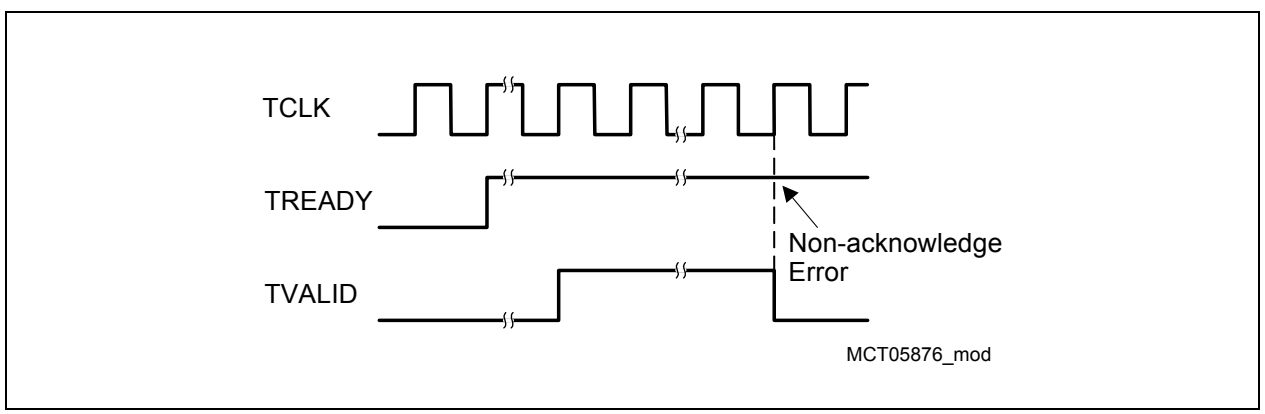


Figure 21-19 Non-Acknowledge Error

21.1.4 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 21-42**.

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 21-45**.

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 21-25**)
- The general MLI features (see **Page 21-42**)
- The interface signals (see **Page 21-49**)
- The general MLI service request structure (see **Page 21-55**)
- The MLI transmitter events (see **Page 21-57**)
- The MLI receiver events (see **Page 21-60**)
- The baud rate generation (see **Page 21-65**)

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 21-26**)
- Data Frames (see **Page 21-28**)
- Read Frames (see **Page 21-32**)
- Answer Frame (see **Page 21-37**)
- Command Frame (see **Page 21-39**)

21.2.1.1 Copy Base Address Frame

A Copy Base Address Frame defines the location and the size of a Remote Window.

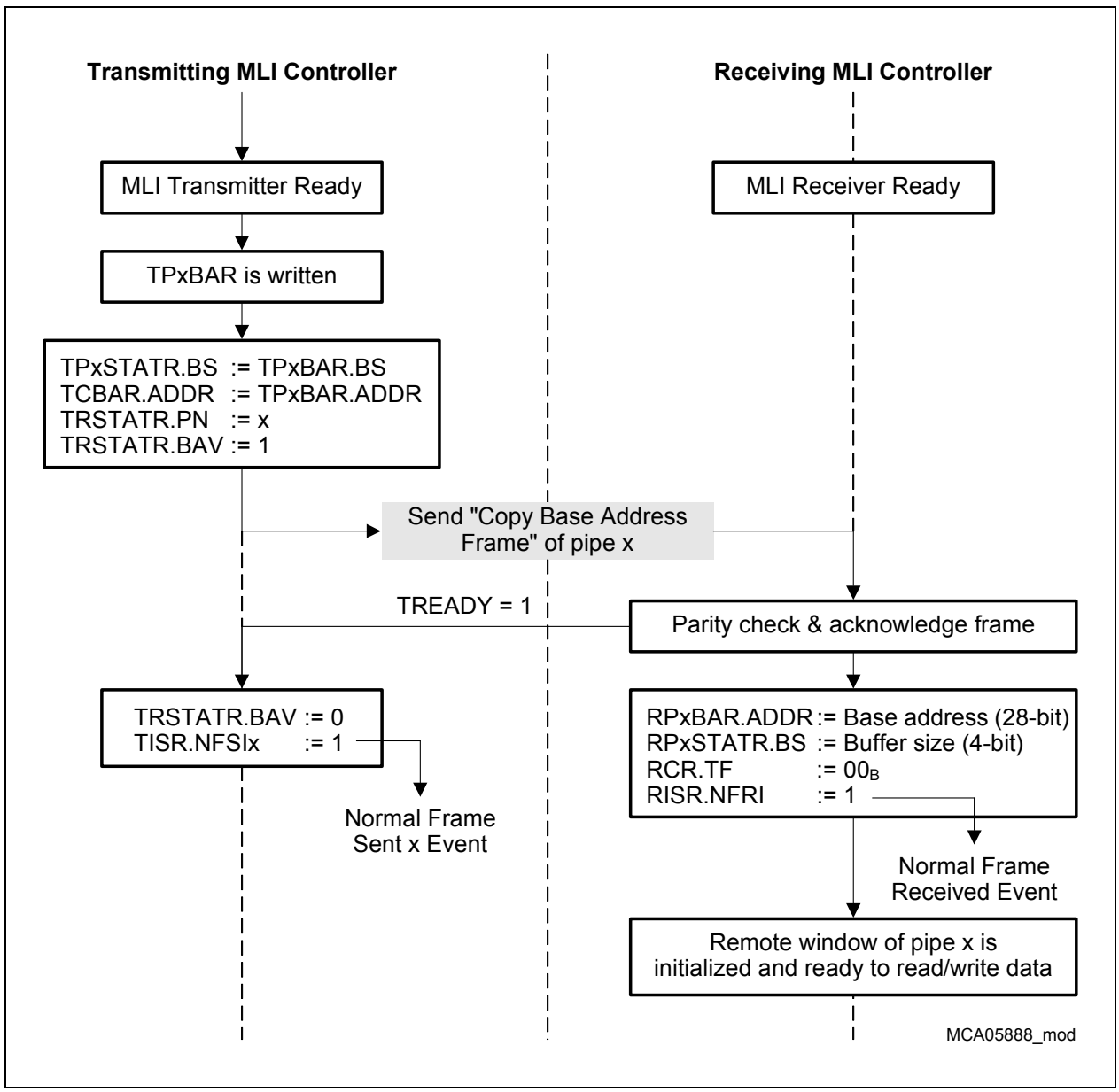


Figure 21-20 Copy Base Address Frame Flow

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.

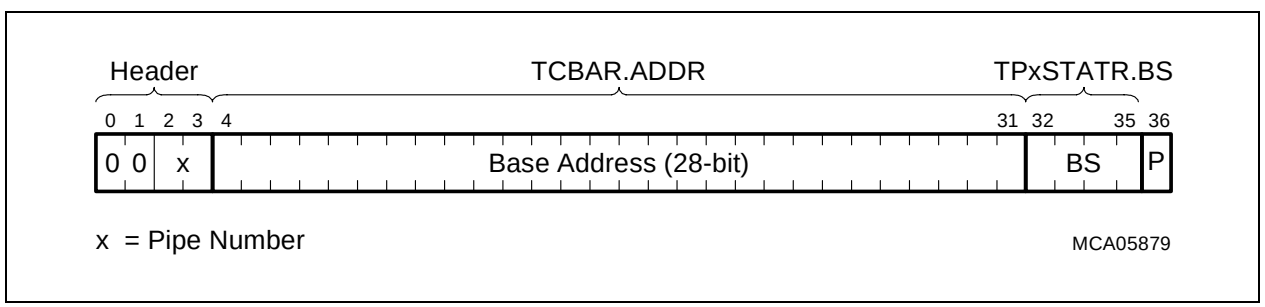


Figure 21-21 Copy Base Address Frame

21.2.1.2 Write/Data Frames

Write Frames (also named Data Frames) transmit the write data and optionally the write offset.

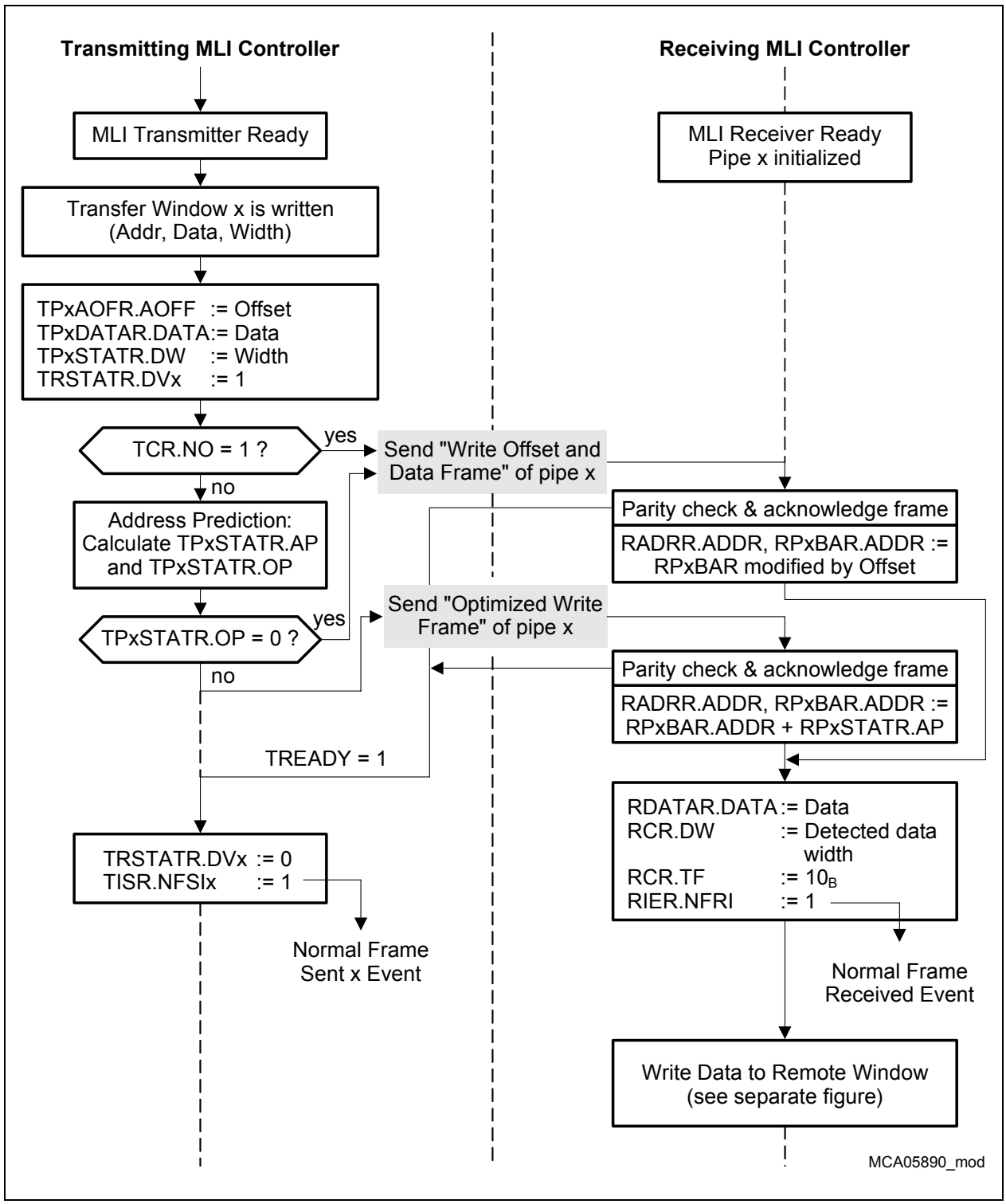


Figure 21-22 Write Frame Flow

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.
- 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 21-45**.
- 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.

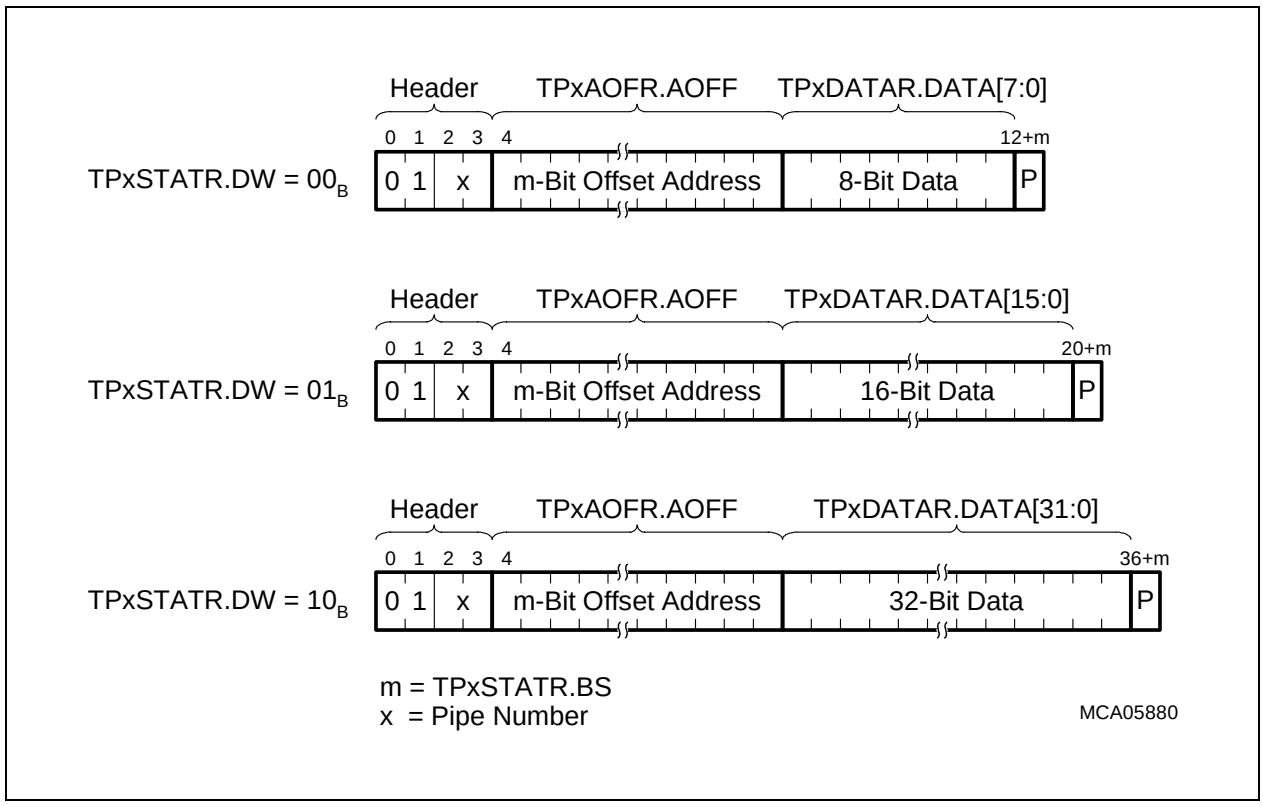


Figure 21-23 Write Offset and Data Frame

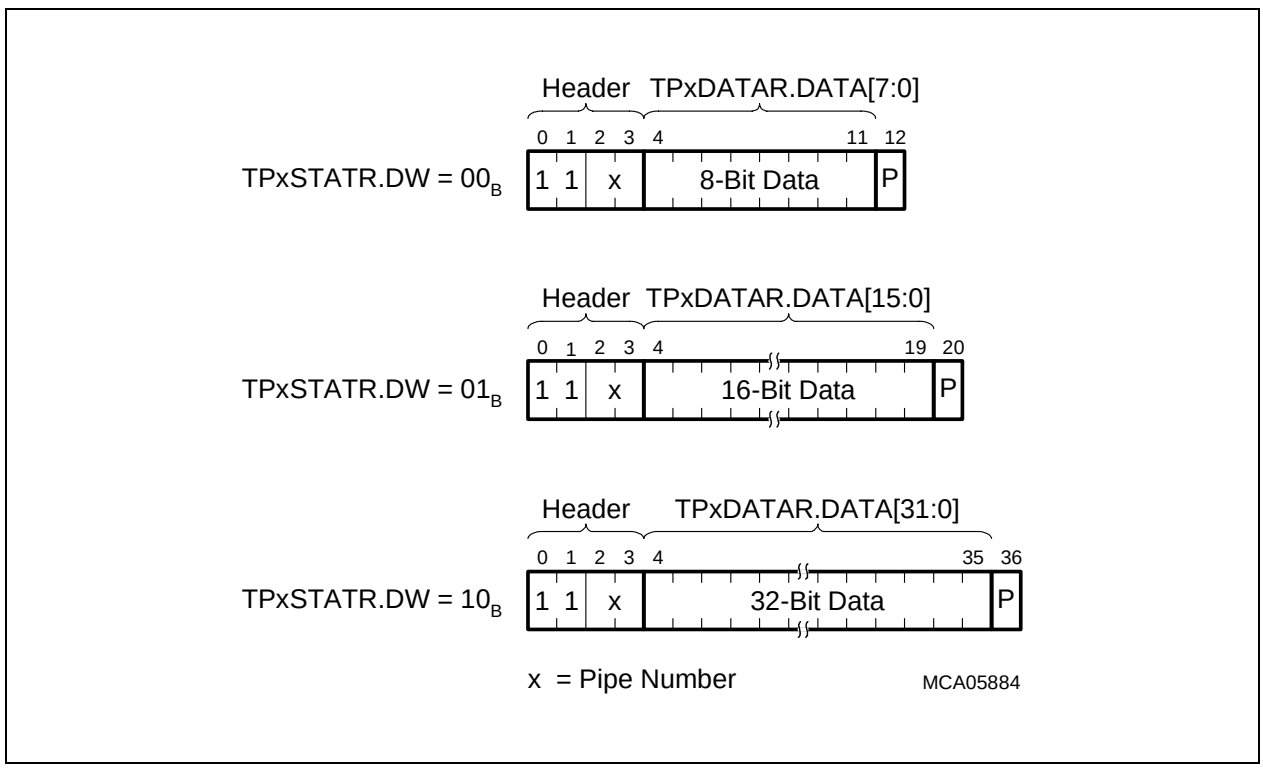


Figure 21-24 Optimized Write Frame

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.

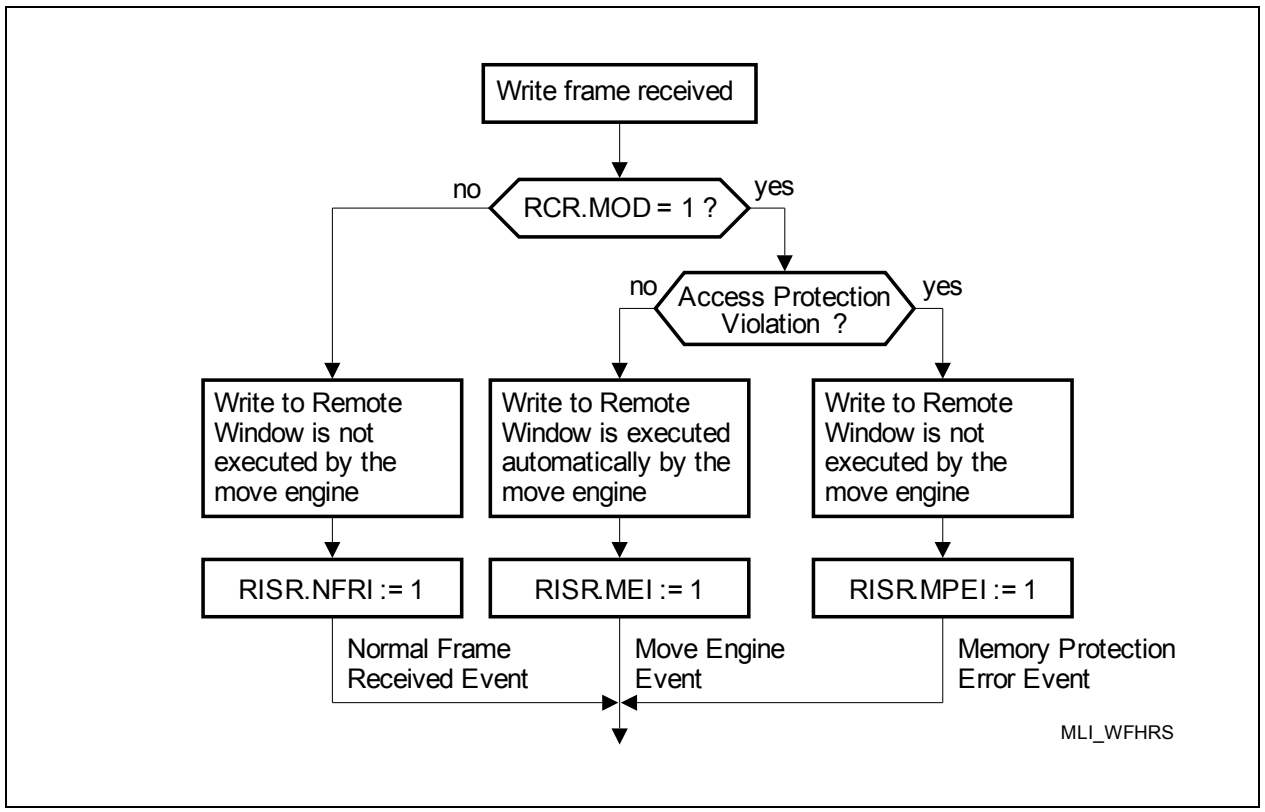


Figure 21-25 Write Frame Handling on Receiving Side

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.

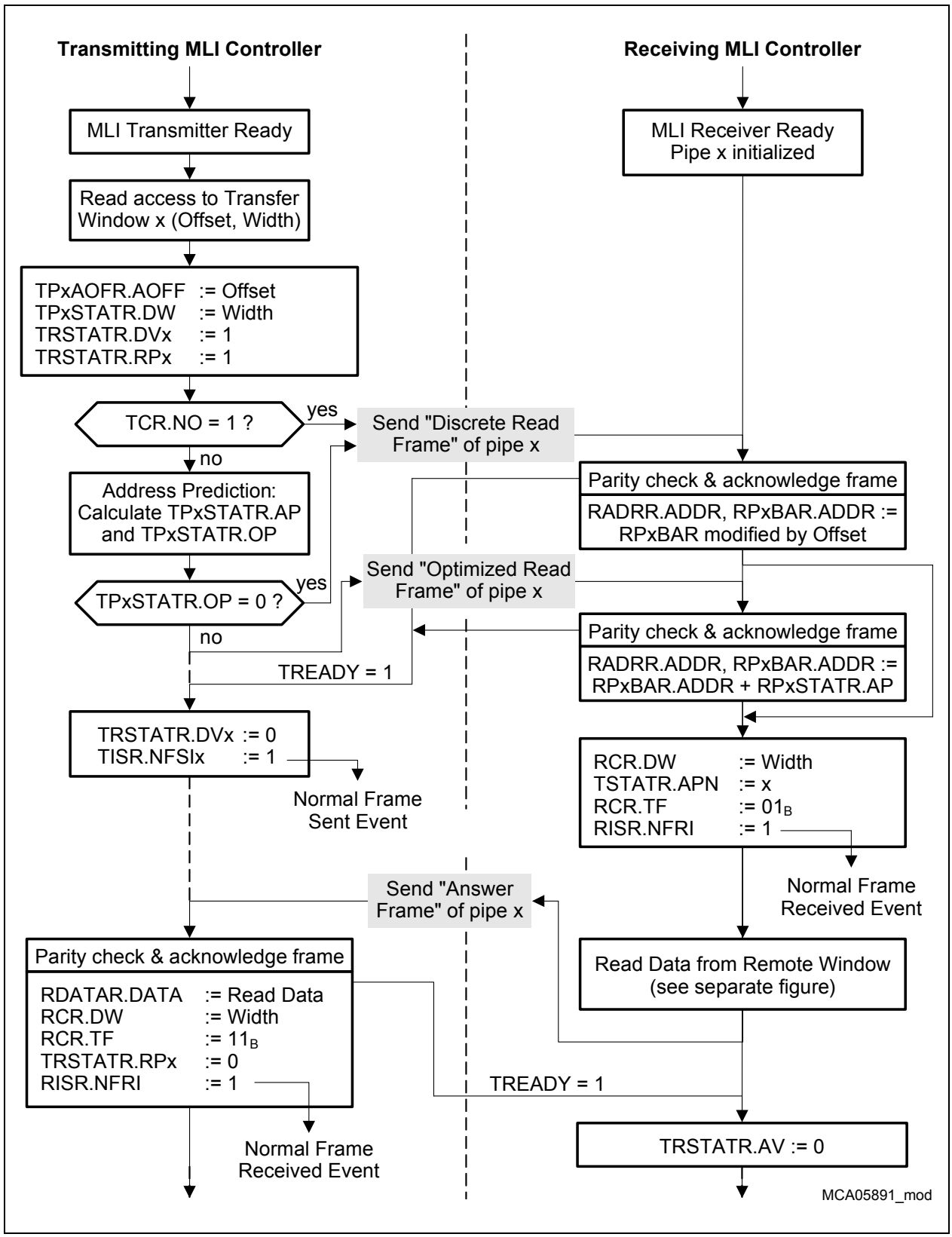


Figure 21-26 Read Frame and Answer Frame Flow

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.
- 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.

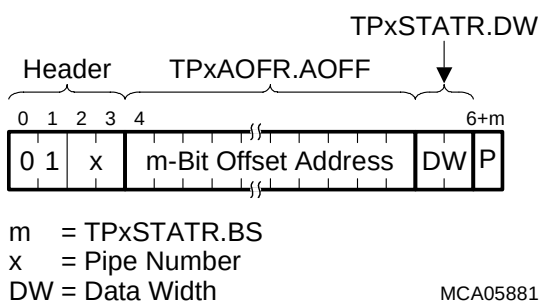


Figure 21-27 Discrete Read Frame

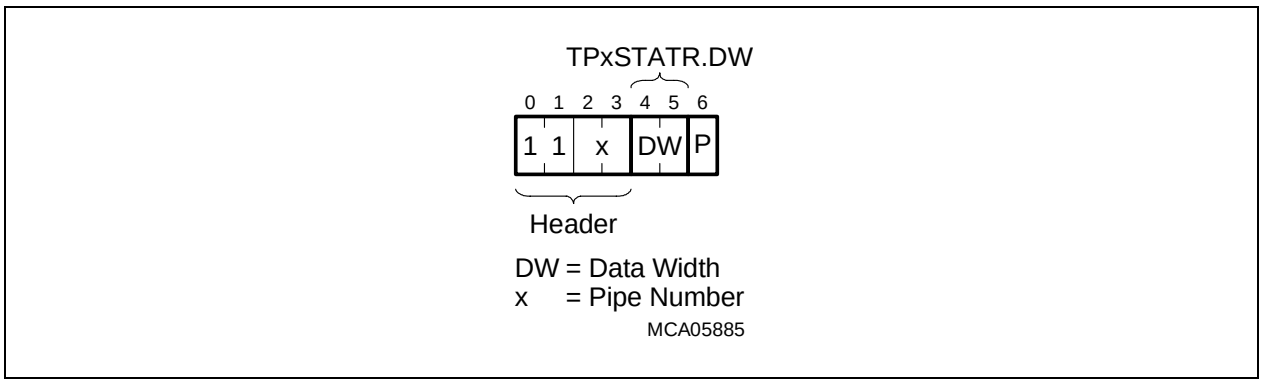


Figure 21-28 Optimized Read Frame

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 the 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.

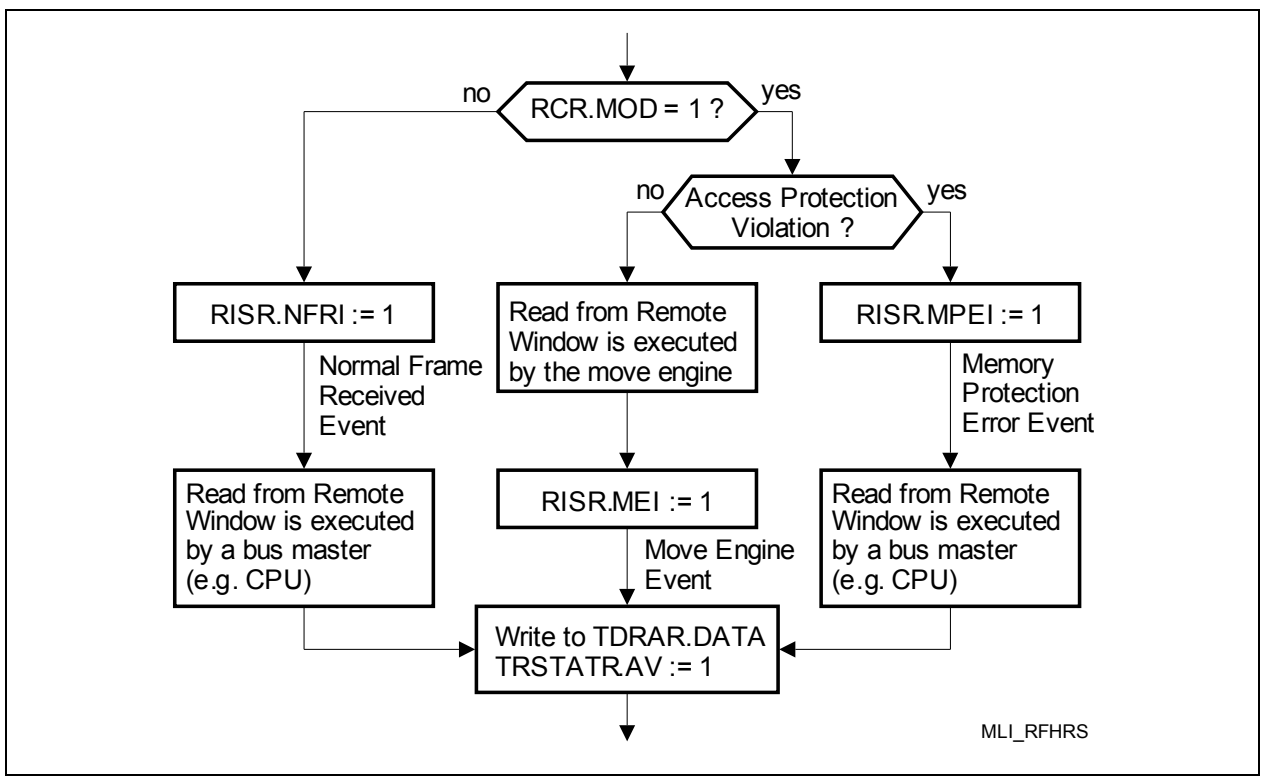


Figure 21-29 Read Frame Handling on Remote Side

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. In future versions, it is intended to clear only the flag belonging to pipe x.
- 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 21-60**).

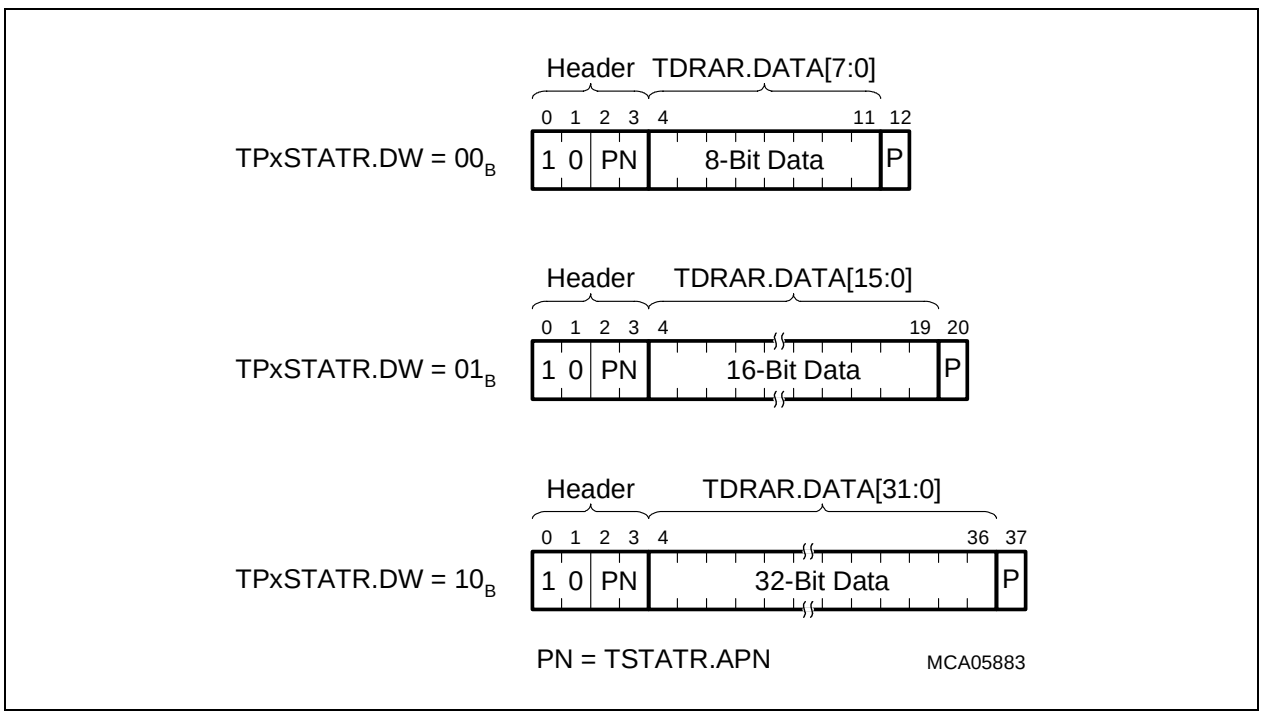


Figure 21-30 Answer Frame

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.

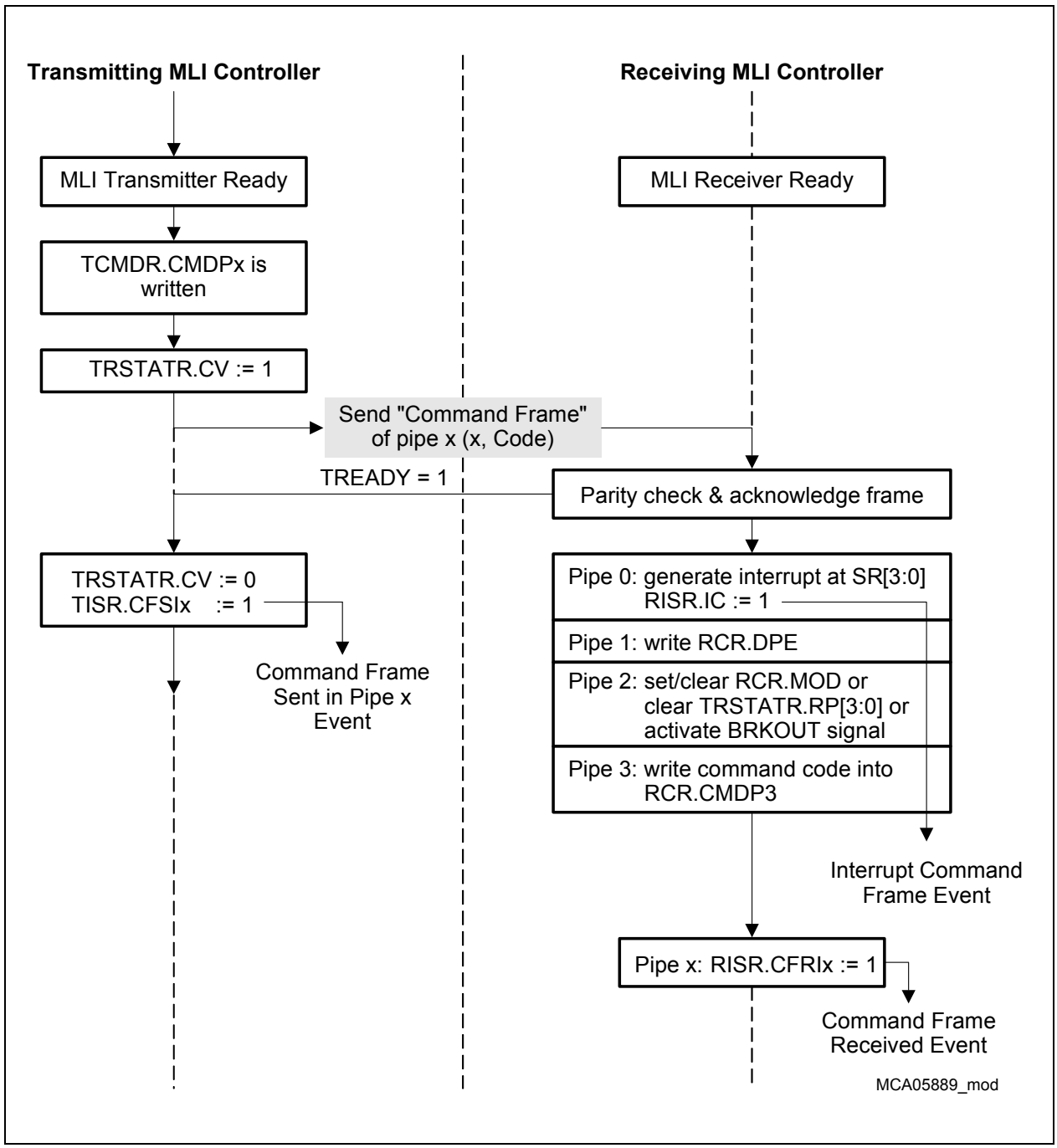


Figure 21-31 Command Frame Transaction Flow

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 21-42**)
- Non-Acknowledge error (see **Page 21-45**)
- Address prediction (see **Page 21-45**)
- Automatic data transfers (see **Page 21-46**)
- Access protection (see **Page 21-47**)
- Transmit priority (see **Page 21-48**)
- Transmission delay (see **Page 21-48**)

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 21-71**).

Note: There is no protection against frames where more than one bit is corrupted (e.g. shortened frames). In such a case, an unpredicted 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 21-39** on **Page 21-58**) 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.

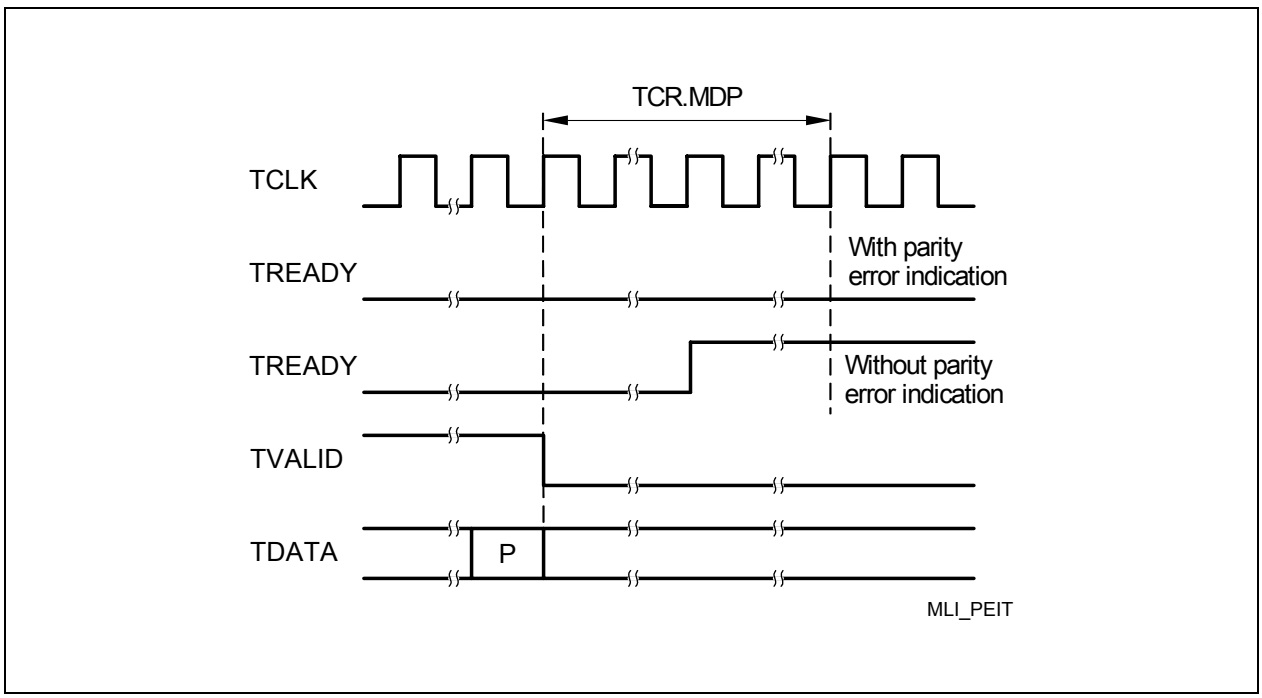


Figure 21-32 Parity Error Indication for the Transmitter

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 21-43** on **Page 21-61**) 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.

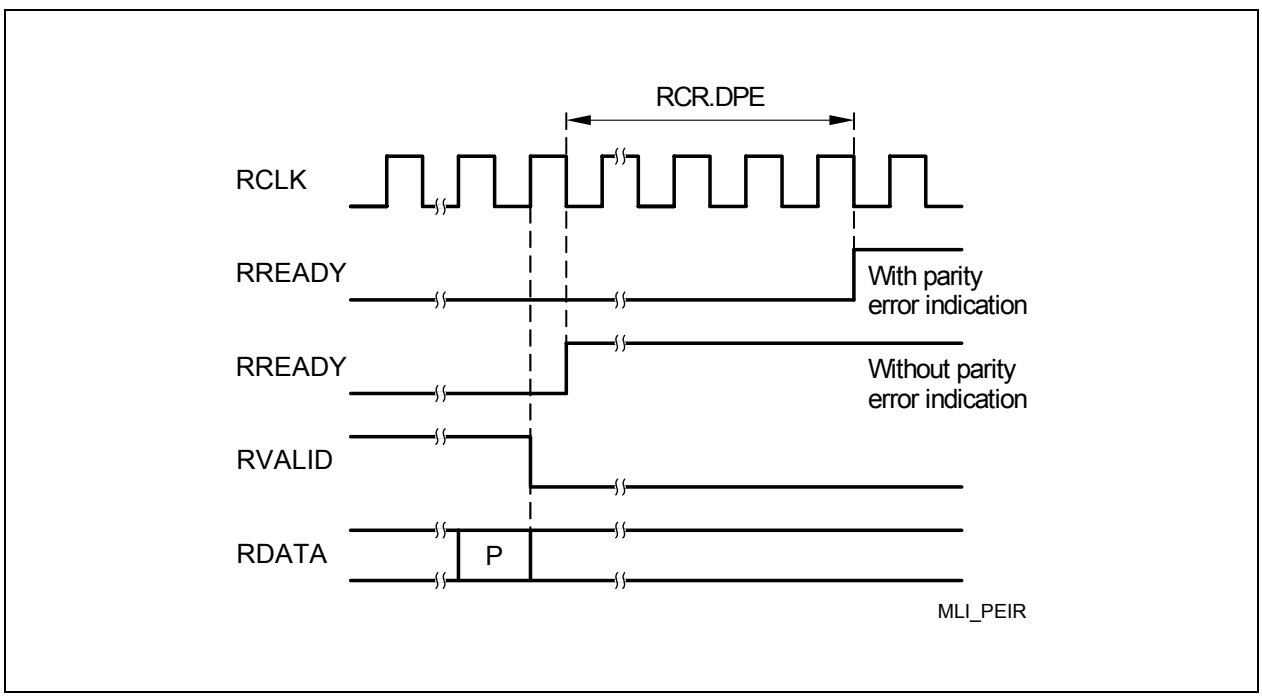


Figure 21-33 Parity Error Indication by the Receiver

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 21-39** on **Page 21-58**) 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 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 and given in ***Section 21.5.6*** *on* ***Page 21-140****.*

21.2.2.6 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.

- Answer Frame (only one frame pending allowed at a time)
- 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)

21.2.2.7 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).

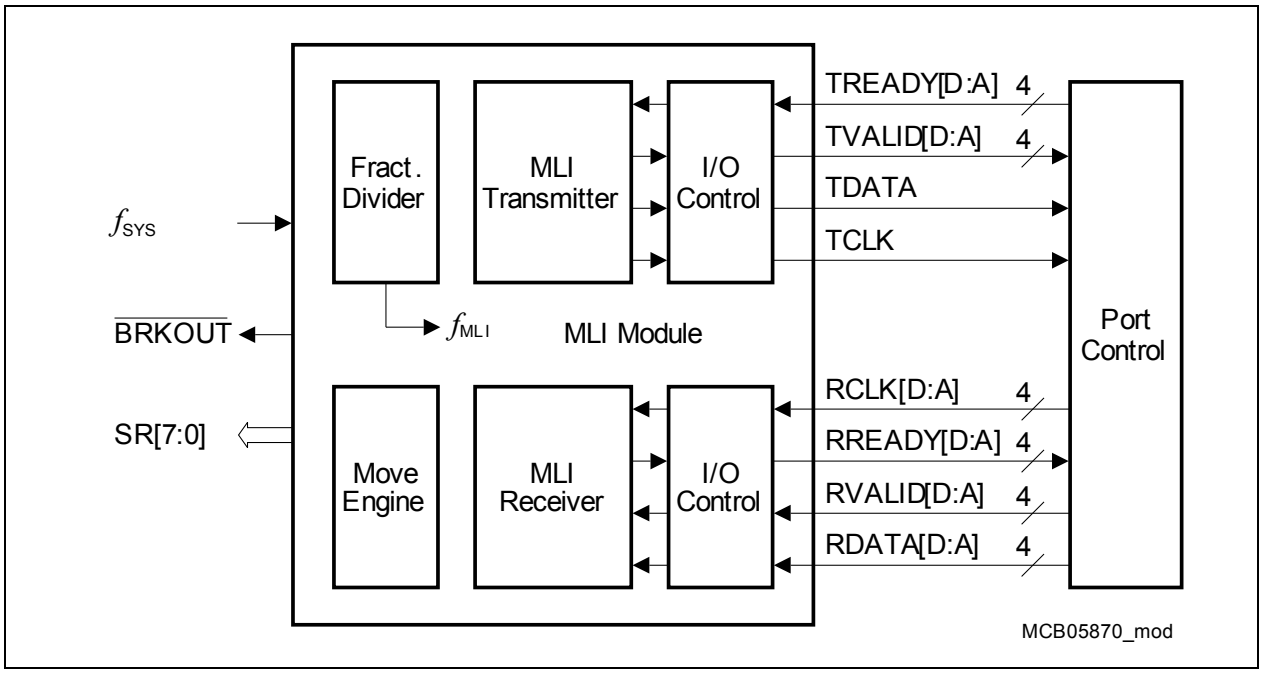


Figure 21-34 General Block Diagram of the MLI Module

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.
- Polarity inversion is disabled for all signals (no inversion).

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.

- 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]).

21.2.3.1 Transmitter I/O Line Control

Figure 21-35 shows the MLI transmitter I/O control logic.

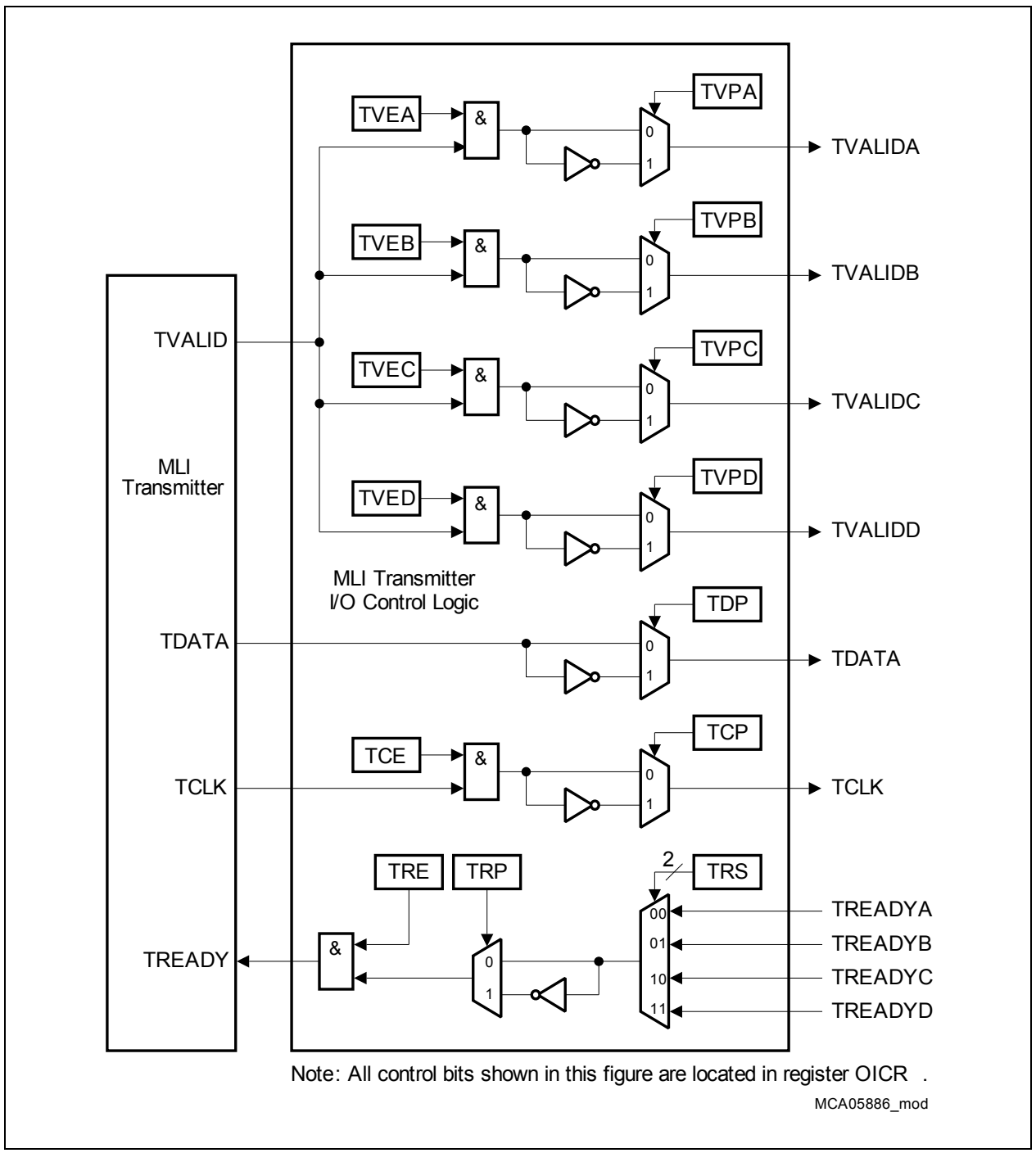


Figure 21-35 Transmitter Input/Output Control Logic

21.2.3.2 Receiver I/O Line Control

Figure 21-36 shows the MLI receiver I/O control logic.

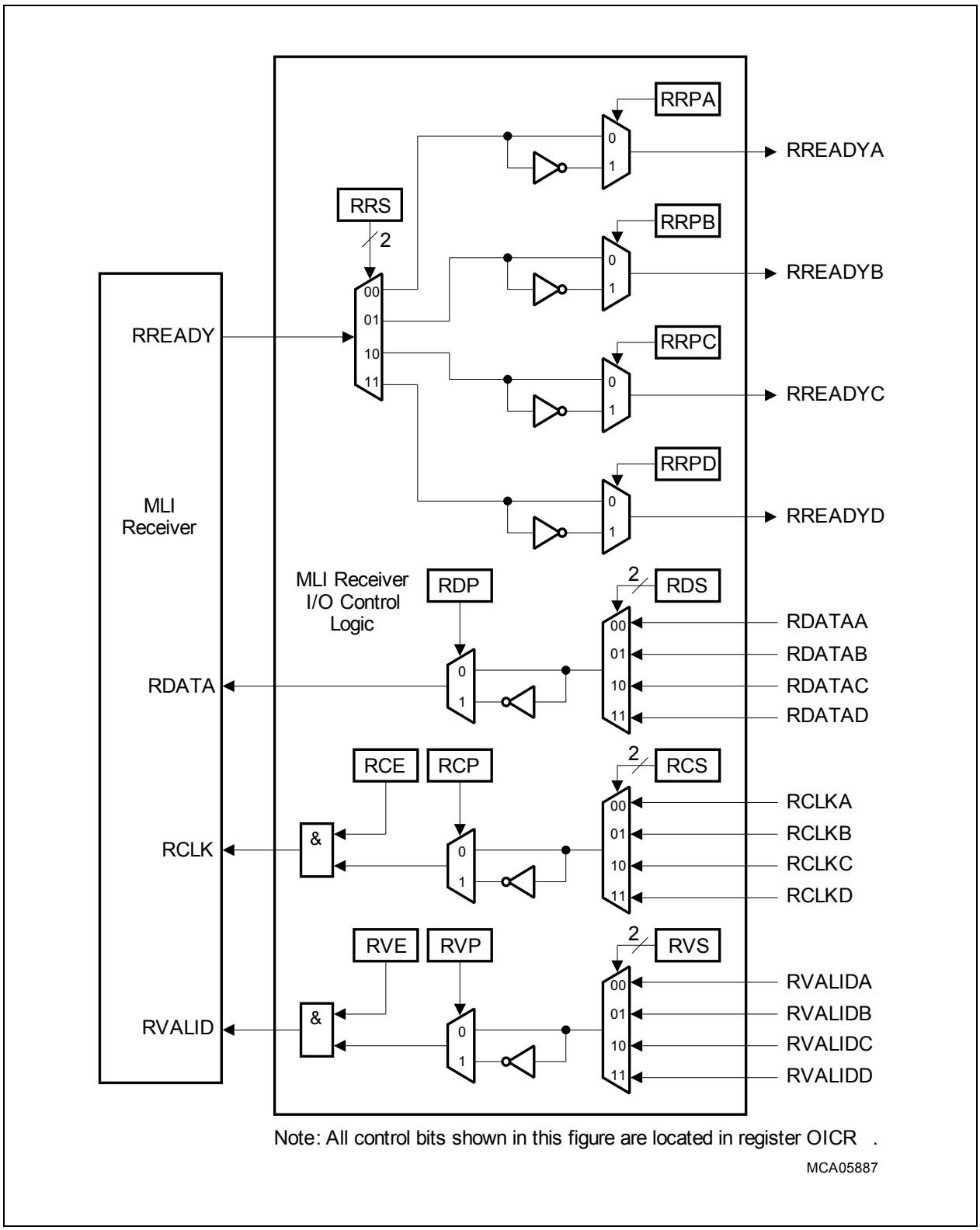


Figure 21-36 Receiver Input/Output Control Logic

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.

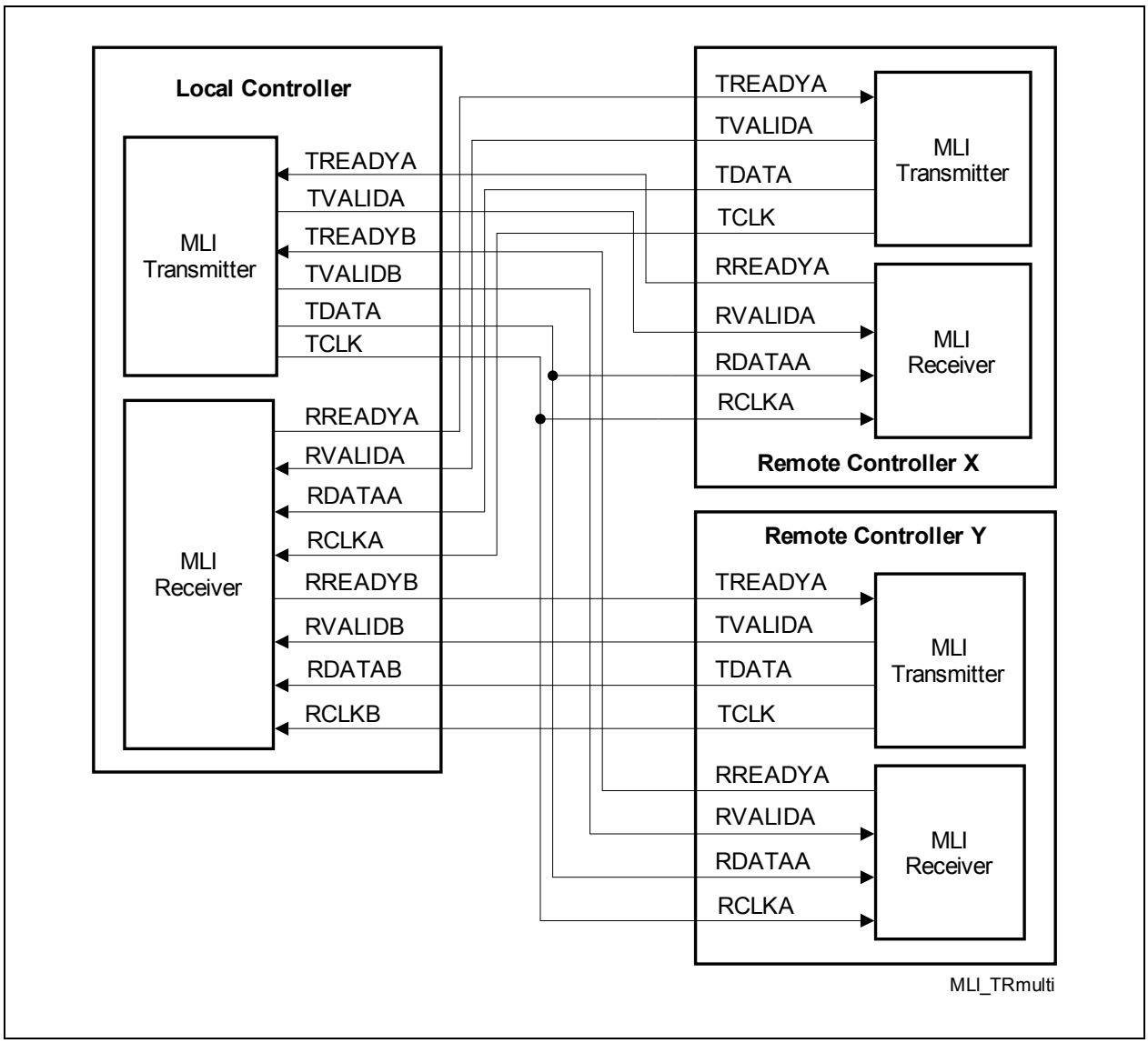


Figure 21-37 Connecting Two Remote Controllers

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 as a request pulse, 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 21-56**). 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 (for transmitter see **Page 21-105**, for receiver see **Page 21-117**). 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 21-38 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.

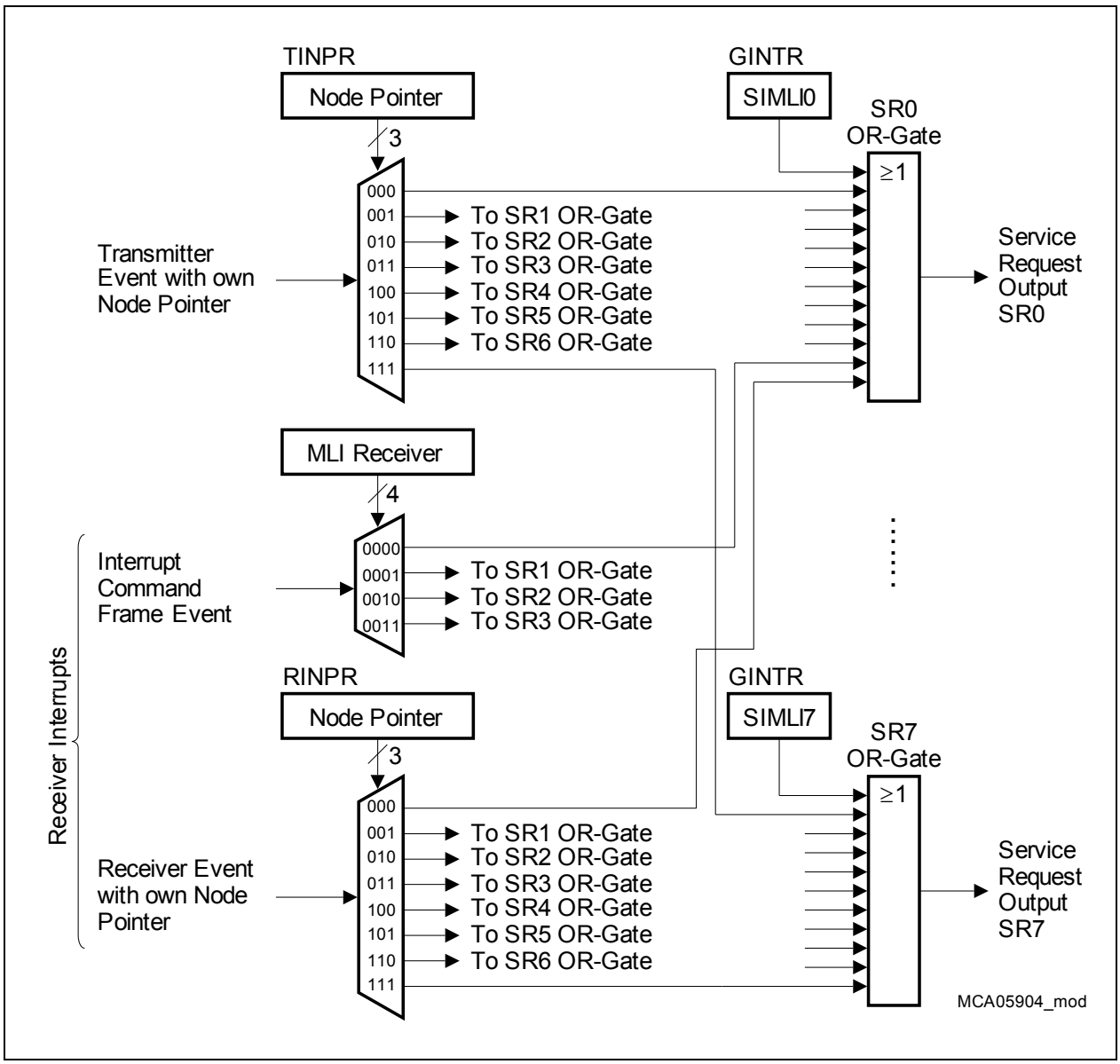


Figure 21-38 Service Request Compressor

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 21-58
Time-out Error		
Normal Frame Sent in Pipe 0	Normal Frame Sent in Pipe 0	Page 21-58
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 21-59
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.

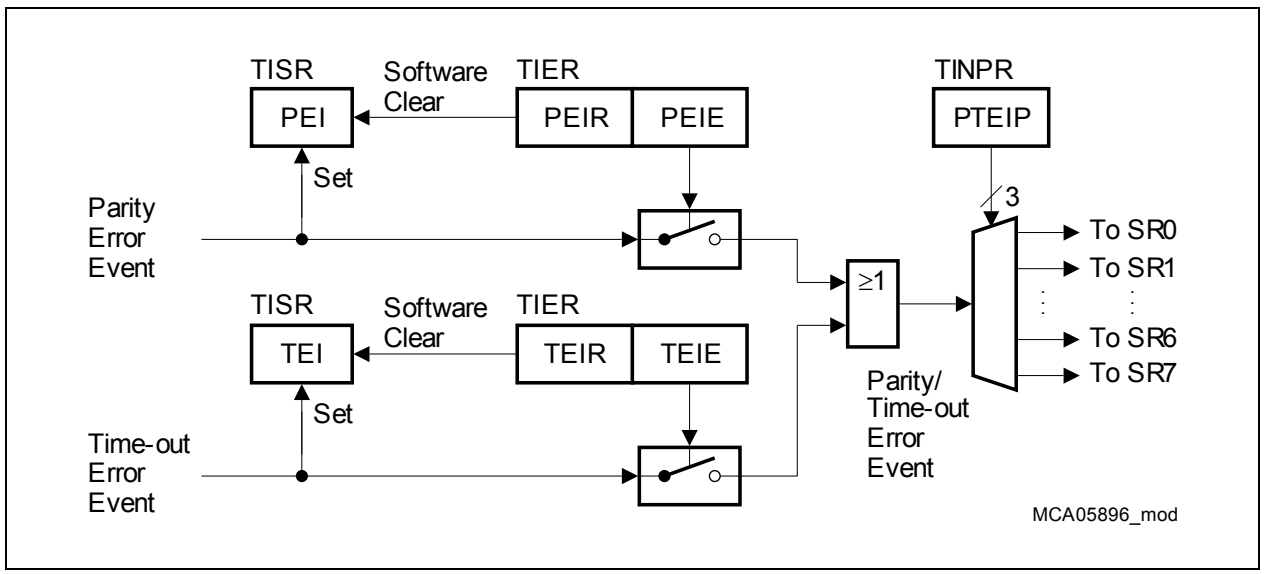


Figure 21-39 Parity/Time-out Error Event Logic

21.2.5.2 Normal Frame Sent x Event

A Normal Frame sent in pipe x (x = 0-3) event is generated when a Normal Frame has been sent and correctly received in pipe x.

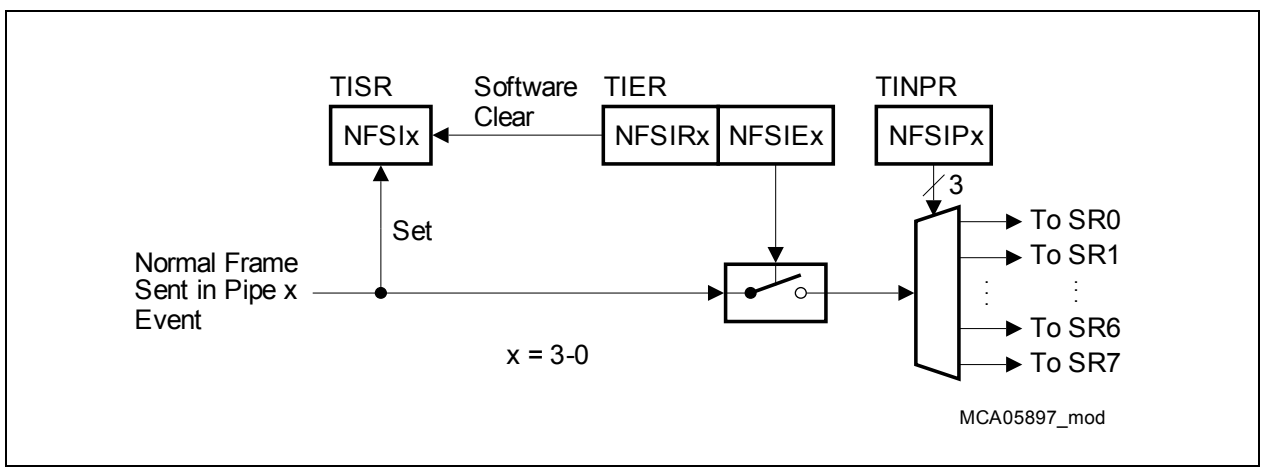


Figure 21-40 Normal Frame Sent in Pipe x Event Logic

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.

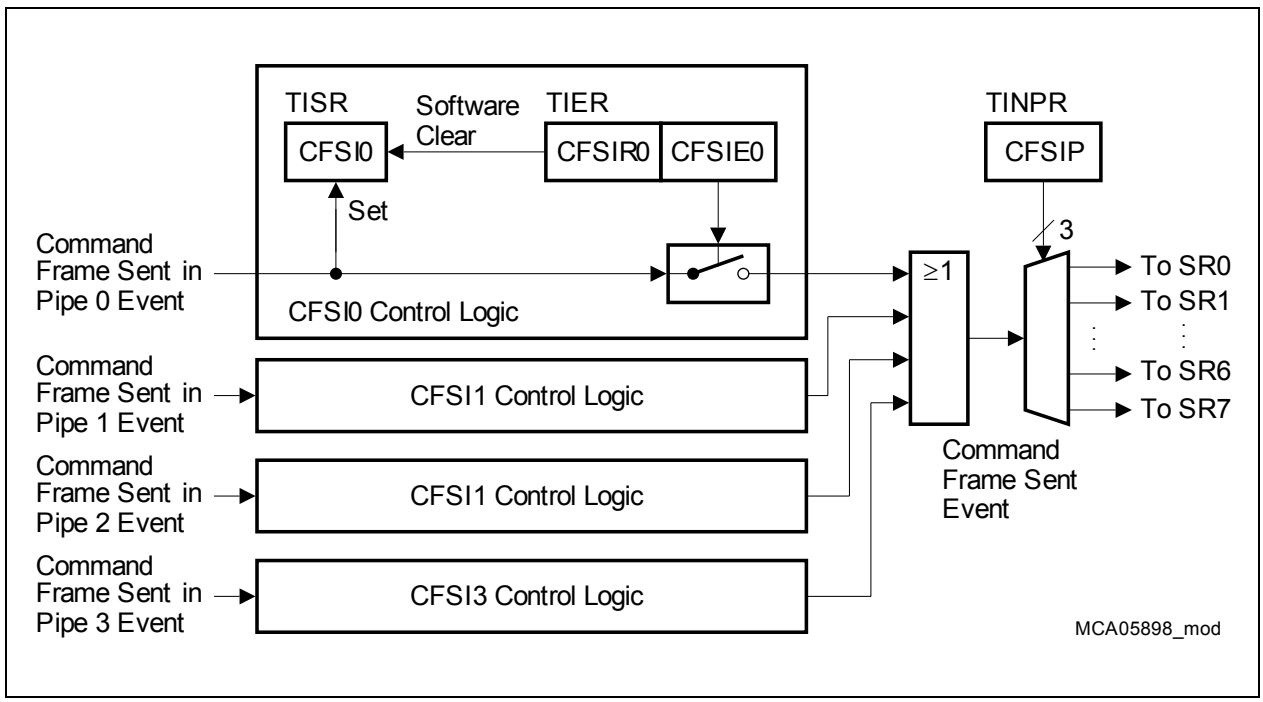


Figure 21-41 Command Frame Sent Event Logic

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 21-60
Memory Access Protection Error	Memory Access Protection/ Parity Error	Page 21-61
Parity Error		
Normal Frame Correctly Received	Normal Frame Received	Page 21-62
Move Engine Access Terminated		
Interrupt Command Frame	Interrupt Command Frame	Page 21-63
Command Frame Received on Pipe 0	Command Frame Received	Page 21-64
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.

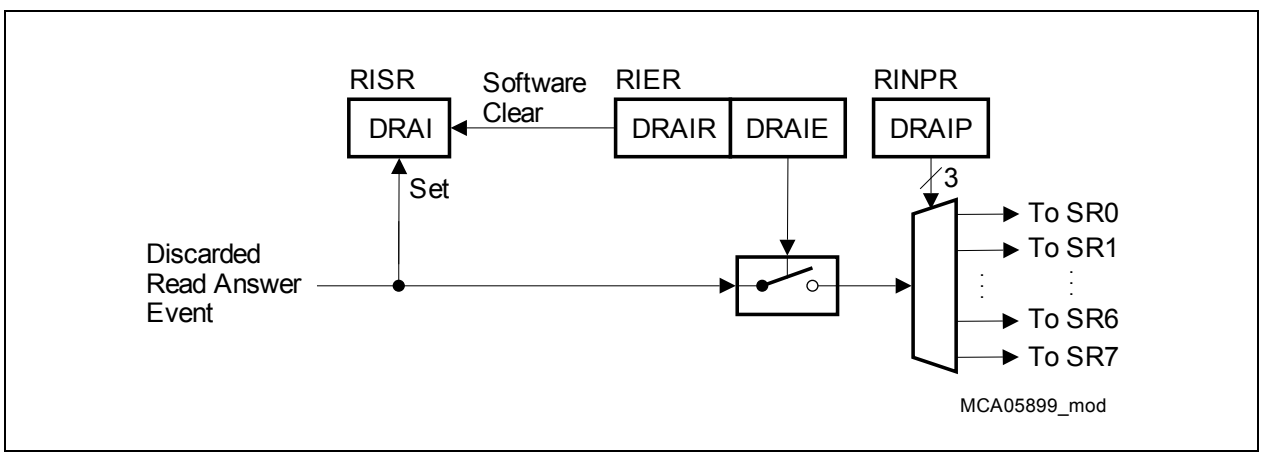


Figure 21-42 Discarded Read Answer Event Logic

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.

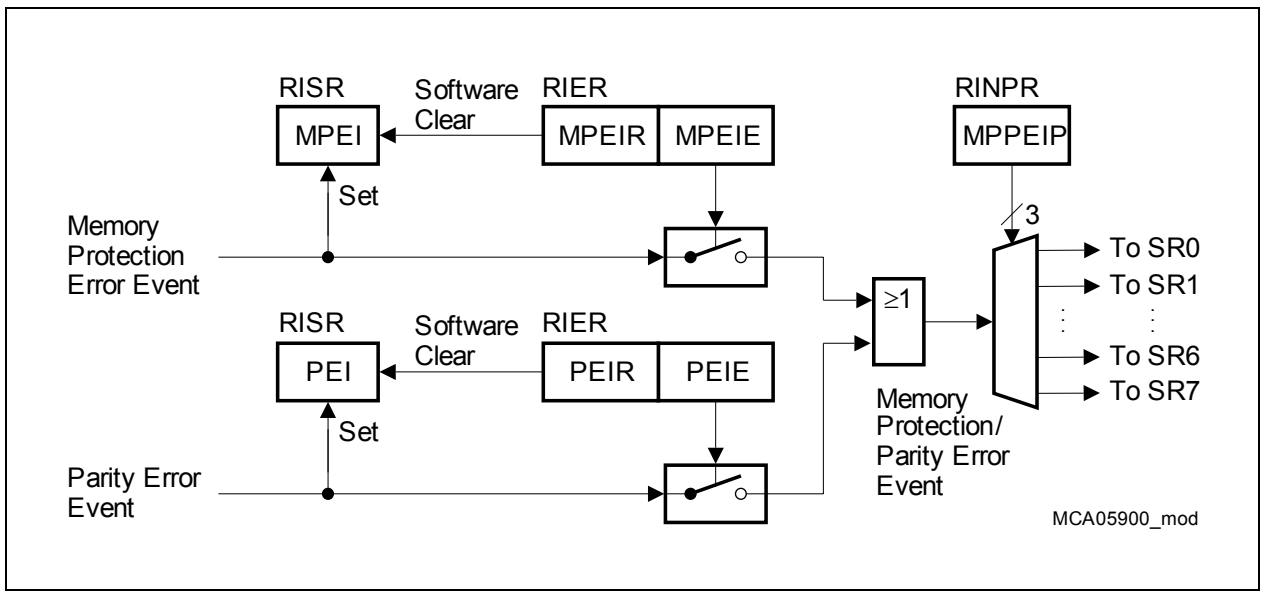


Figure 21-43 Memory Access Protection/Parity Error Event Logic

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.

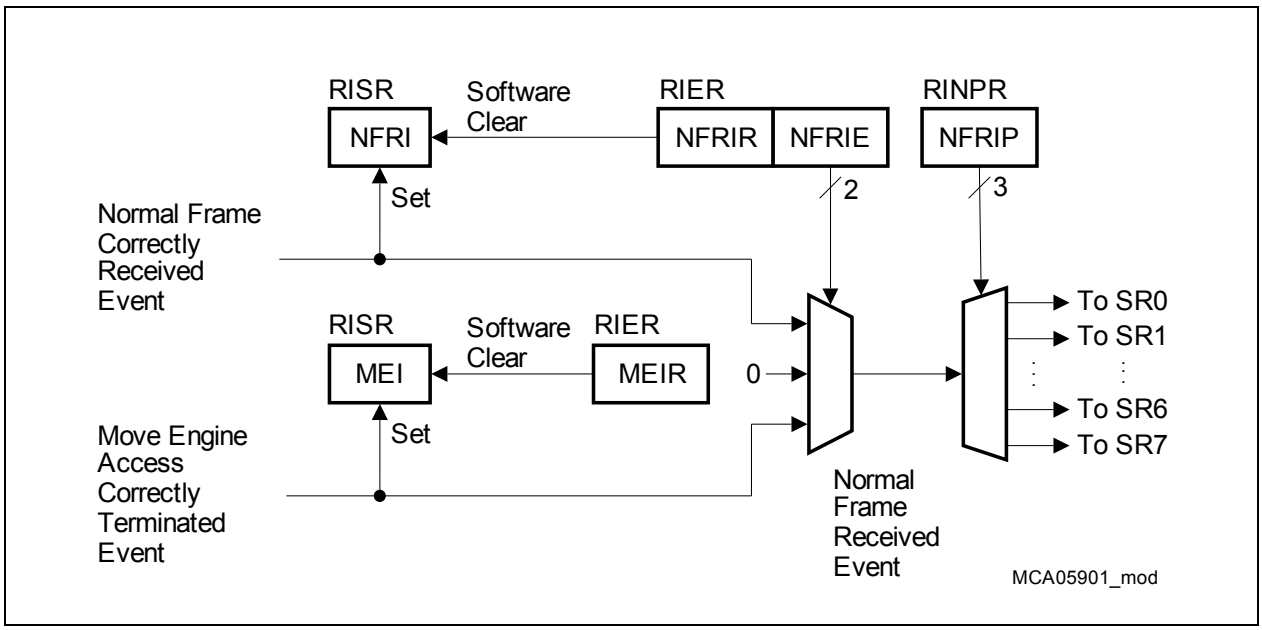


Figure 21-44 Normal Frame Received Event Logic

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.

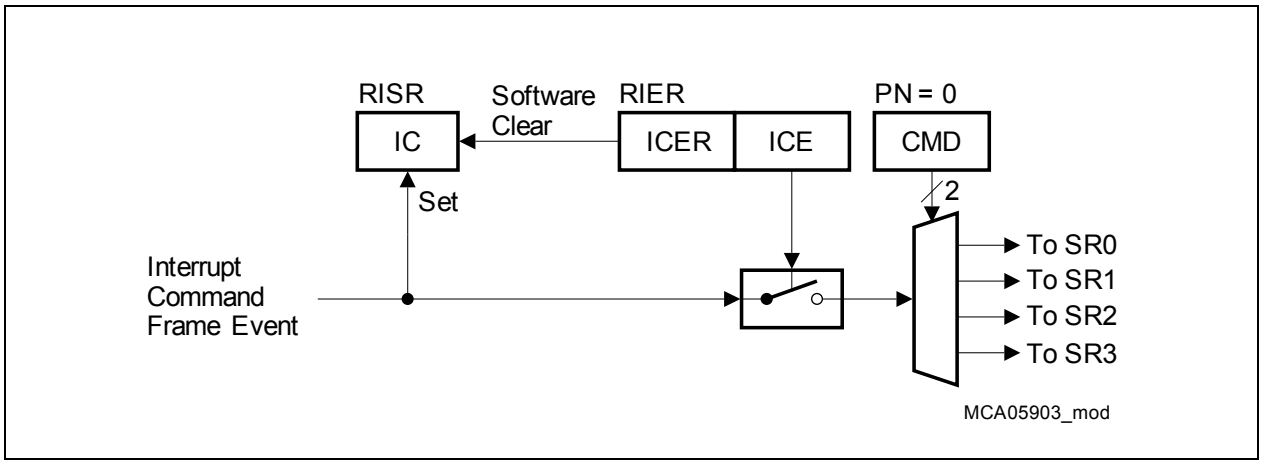


Figure 21-45 Interrupt Command Frame Event Logic

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.

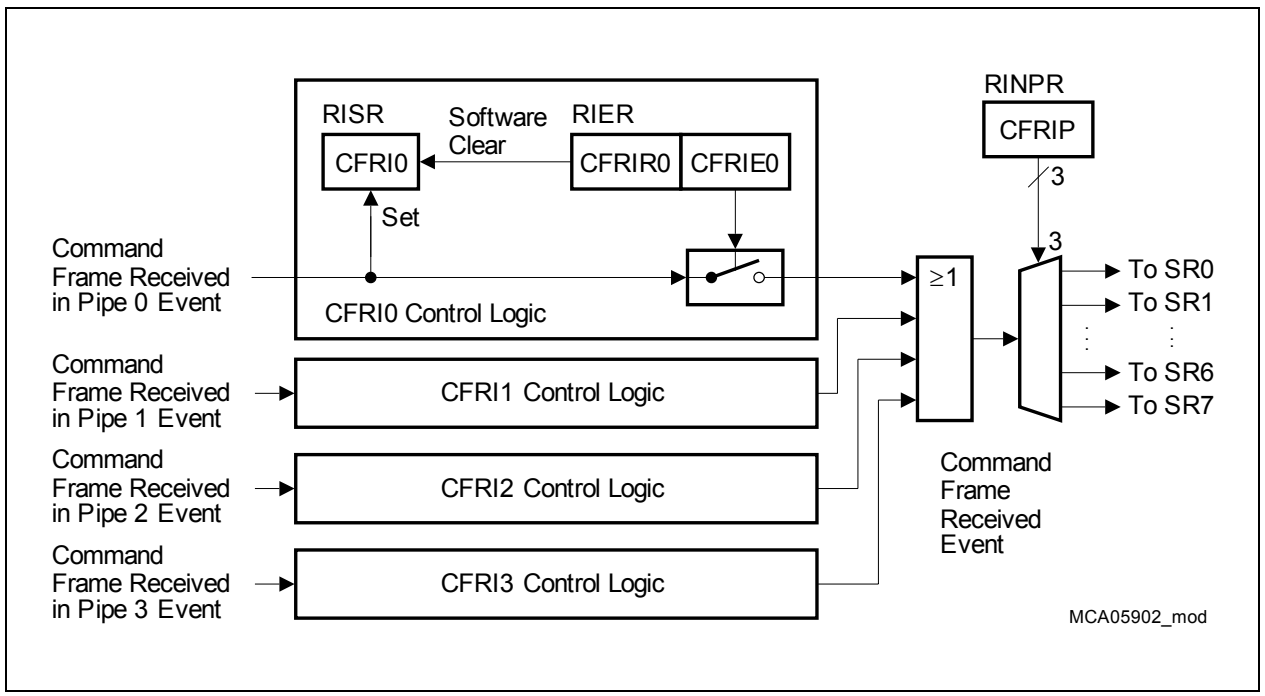


Figure 21-46 Command Frame Received Event Logic

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.

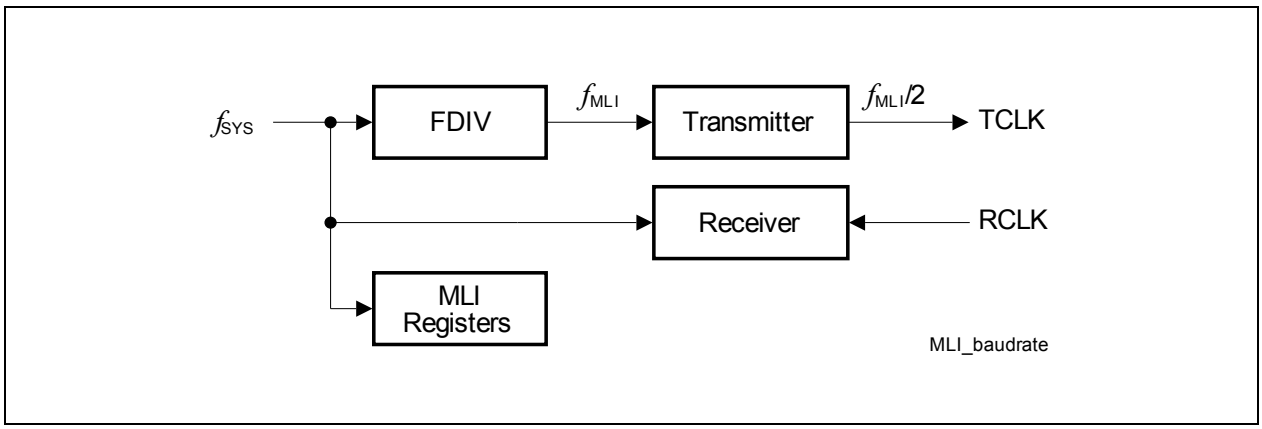


Figure 21-47 MLI Baud Rate Generation

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 21-127**).

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.

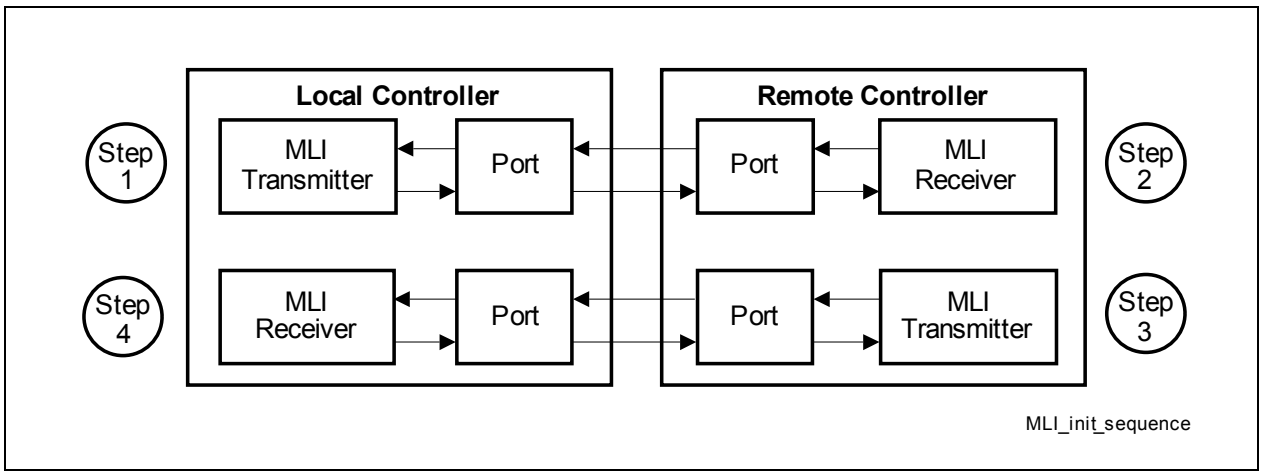


Figure 21-48 Initialization Sequence for an MLI Connection

To initialize and to operate the MLI, the following items should be taken into account:

- Connection setup (see **Page 21-68**)
- Local Controller transmitter and pipe setup (see **Page 21-69**)
- Remote Controller receiver setup (see **Page 21-69**)
- Remote Controller transmitter and Local Controller receiver setup (see **Page 21-70**)
- Delay adjustment (see **Page 21-71**)
- Connection to DMA mechanism (see **Page 21-73**)
- Connection of MLI to SPI (see **Page 21-73**)

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 f_{SYS} in this example) has to be handled by the communicating devices.
- 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 21-71**.

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).
  In local loop back mode (see **Section 21.5.1** on **Page 21-124**), 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 21-42**).

In the receiver, frames with parity error are ignored for data transfers and don't lead to internal move actions.

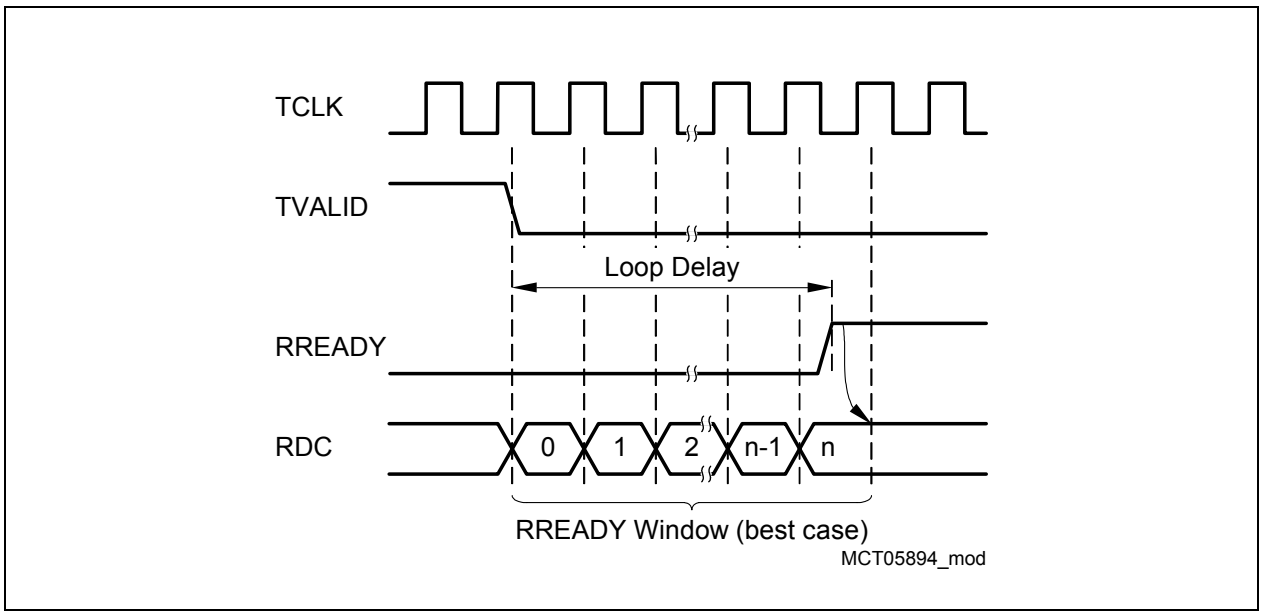


Figure 21-49 Loop Delay Measurement

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 modules. All MLI kernel register names described in this section will be referenced in other parts of the TC1766 User's Manual by the module name prefix "MLI0_" for the MLI0 interface and "MLI1_" for the MLI1 interface.

MLI Kernel Register Overview

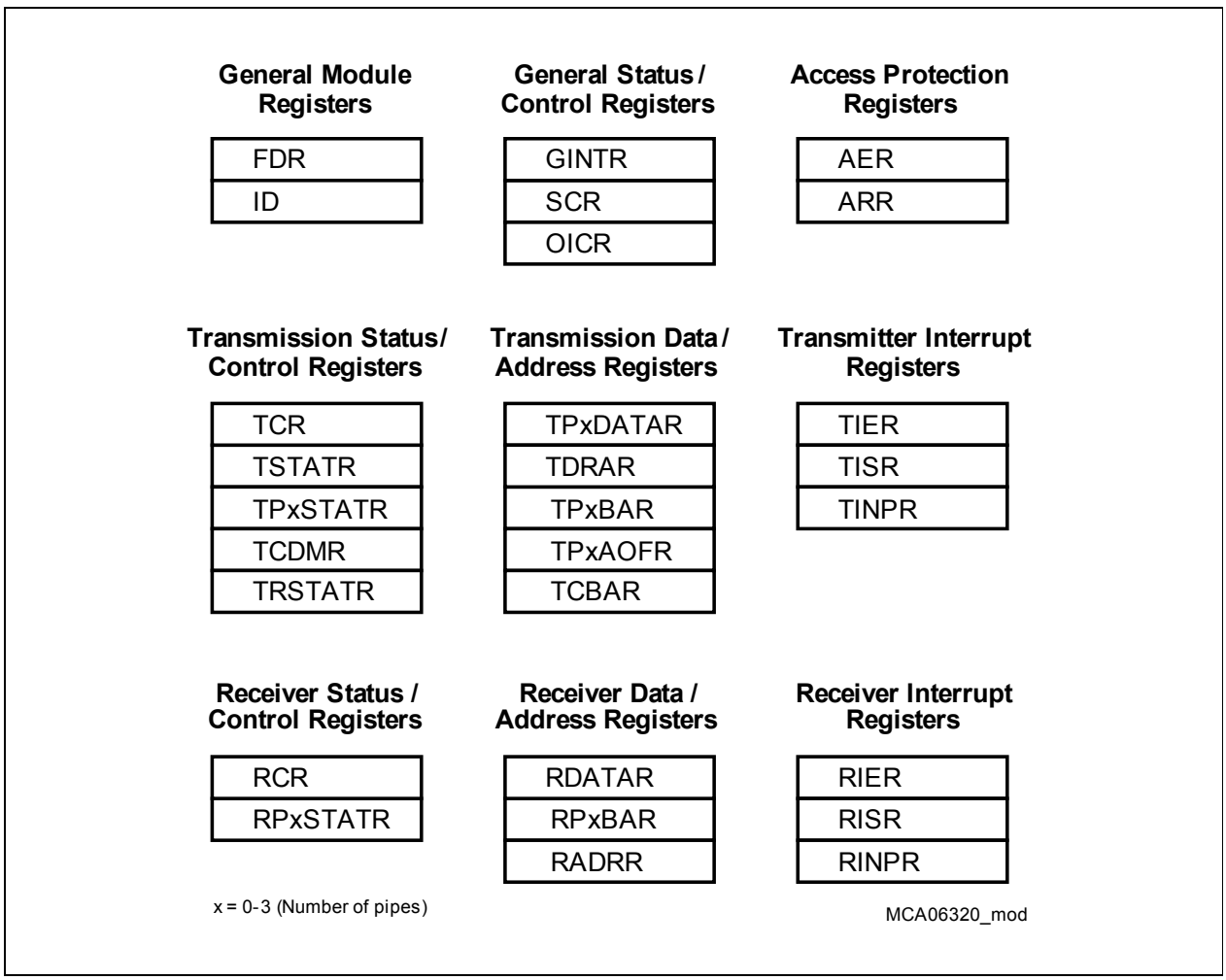


Figure 21-50 MLI Kernel Registers

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 modules is described in **Table 16-22** on **Page 16-60** of the TC1766 User's Manual System Units part (Volume 1).

Table 21-9 Registers Address Space - MLI 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	Description see
ID	Module Identification Register	08_H	Page 21-78
FDR	Fractional Divider Register	$0C_H$	Page 21-79
TCR	Transmitter Control Register	10_H	Page 21-90
TSTATR	Transmitter Status Register	14_H	Page 21-93
TPxSTATR	Transmitter Pipe x Status Register	18_H + (x * 4)	Page 21-95
TCMDR	Transmitter Command Register	28_H	Page 21-97
TRSTATR	Transmitter Receiver Status Register	$2C_H$	Page 21-99
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 21-101
TDRAR	Transmitter Data Read Answer Register	50_H	Page 21-101
TPxBAR	Transmitter Pipe x Base Address Register	54_H + (x * 4)	Page 21-102
TCBAR	Transmitter Copy Base Address Register	64_H	Page 21-104
RCR	Receiver Control Register	68_H	Page 21-110
RPxBAR	Receiver Pipe x Base Address Register	$6C_H$ + (x * 4)	Page 21-115
RPxSTATR	Receiver Pipe x Status Register	$7C_H$ + (x * 4)	Page 21-113
RADDR	Receiver Address Register	$8C_H$	Page 21-116
RDATAR	Receiver Data Register	90_H	Page 21-114
SCR	Set Clear Register	94_H	Page 21-81
TIER	Transmitter Interrupt Enable Register	98_H	Page 21-105

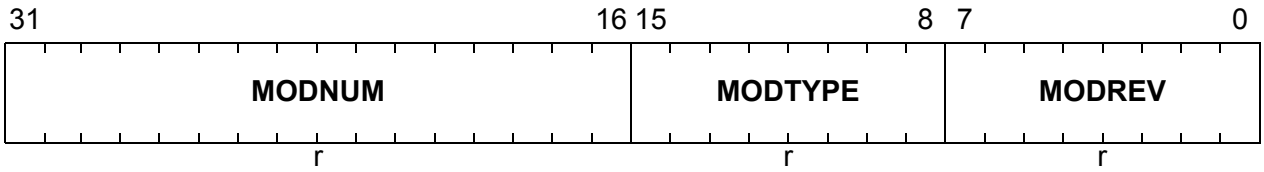
Table 21-10 Registers Overview - MLI Kernel Registers (cont'd)

Register Short Name	Register Long Name	Offset Address	Description see
TISR	Transmitter Interrupt Status Register	$9C_H$	Page 21-107
TINPR	Transmitter Interrupt Node Pointer Register	$A0_H$	Page 21-108
RIER	Receiver Interrupt Enable Register	$A4_H$	Page 21-117
RISR	Receiver Interrupt Status Register	$A8_H$	Page 21-120
RINPR	Receiver Interrupt Node Pointer Register	AC_H	Page 21-122
GINTR	Global Interrupt Set Register	$B0_H$	Page 21-83
OICR	Output Input Control Register	$B4_H$	Page 21-84
AER	Access Enable Register	$B8_H$	Page 21-88
ARR	Access Range Register	BC_H	Page 21-89

21.4.1 General Module Registers

The Module Identification Register ID contains read-only information about the MLI module version.

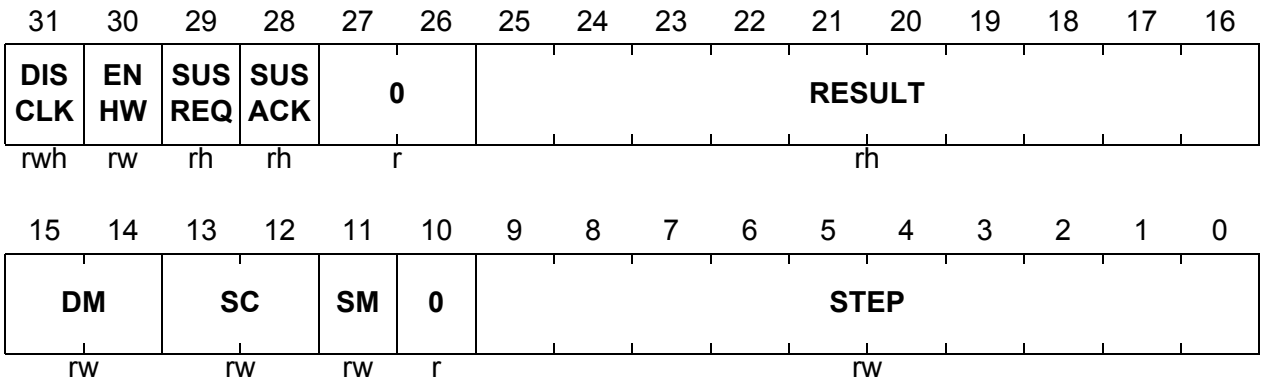
ID
Module Identification Register (08_H) Reset Value: $0025\ C007_H$



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 MLI: 0025_H

The fractional divider register allows the programmer to control the clock rate and period of the module clocks f_{MLI0} and f_{MLI0}. The period of f_{MLIx} can be either 1/STEP or a fraction of STEP/1024 (for any value of STEP from 0 to 1023) of clock f_{MLI}. Each MLI has its own fractional divider.

FDR
Fractional Divider Register **($0C_H$)** **Reset Value: 03FF 43FF$_H$**



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 or fractional divider mode.
RESULT	[25:16]	rh	**Result Value** Bit fields for the addition result.
SUSACK	28	rh	**Suspend Mode Acknowledge** Indicates the state of the SPNDACK signal.
SUSREQ	29	rh	**Suspend Mode Request** Indicates the state of the 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]	rw	**Reserved** Read as 0; should be written with 0.

*Note: Additional details on the fractional divider register functionality are described in section **"Fractional Divider Operation" on Page 3-29** of the TC1766 User's Manual System Units part (Volume 1).*

21.4.2 General Status/Control Registers

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
0				C NAE	C TPE	C RPE	C AV	0						C BAV	C MOD
		w		w	w	w	w			w				w	w

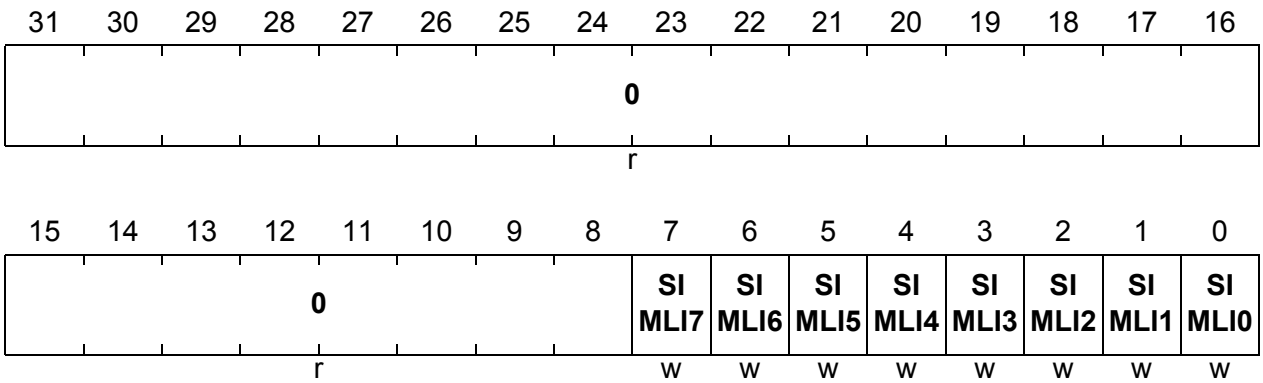
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C CV3	C CV2	C CV1	C CV0	C DV3	C DV2	C DV1	C DV0	0			S MOD	S CV3	S CV2	S CV1	S CV0
w	w	w	w	w	w	w	w		w		w	w	w	w	w

Field	Bits	Type	Description
SCVx (x = 0-3)	x	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.
CDVx (x = 0-3)	8+x	w	**Clear Data Valid x Flag** 0_B No effect 1_B Bits TRSTATR.DVx and TRSTATR.RPx are cleared.
CCVx (x = 0-3)	12+x	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.
0	[7:5], [23:18], [31:28]	w	**Reserved** Read as 0; should be written with 0.

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 21-56**).

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.

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 21-127**). Furthermore, the connection table of the MLI module signals is given there.

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. TREADYx 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

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: See **Table 21-15** on **Page 21-140** for the TC1766-specific address range definitions.*

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

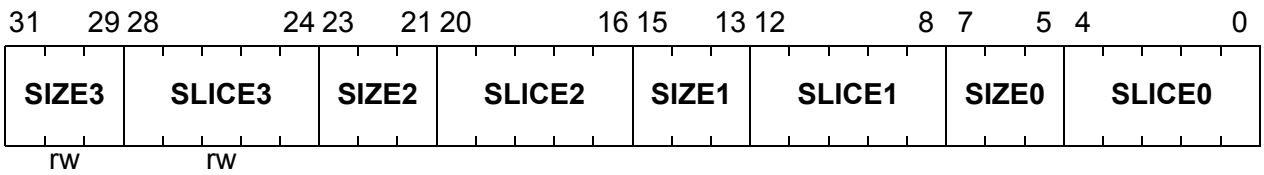
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AEN 15	**AEN 14**	**AEN 13**	**AEN 12**	**AEN 11**	**AEN 10**	**AEN 9**	**AEN 8**	**AEN 7**	**AEN 6**	**AEN 5**	**AEN 4**	**AEN 3**	**AEN 2**	**AEN 1**	**AEN 0**
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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.

The Access Range Register ARR determines size and number of the address sub-range n (n = 0-3).

*Note: See Page 23-46 for bit field definitions and **Page 21-140** for the TC1766-specific address range and address range extension definitions.*

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 Status/Control Registers

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

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
TP	NO	MDP				MNAE		MPE				0	0	DNT	MOD
rw	rw	rw				rwh		rwh				r	rw	rw	rw

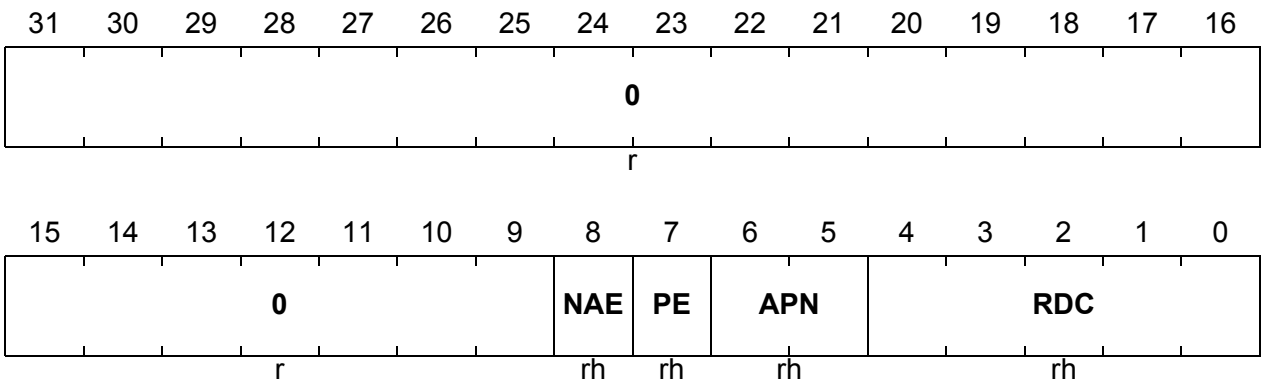
Field	Bits	Type	Description
MOD	0	rw	**Mode of Operation** This bit enables the MLI transmitter. 0_B The MLI transmitter is disabled. 1_B The MLI transmitter is enabled.
DNT	1	rw	**Data in Not Transmission** This bit determines the level of the transmitter data line TDATA when no transmission is in progress. 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 after reset; must 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 21-42**). 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 21-45**). 00_B A time-out event is generated if 1 non-ack condition is detected. 01_B A time-out event is generated if 1 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 21-45**). 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 21-24**). 0_B Even parity selected; parity bit P is set when the modulo-2 sum of frame header bits and data field bits is 1. 1_B Odd parity selected; parity bit P is set when the modulo-2 sum of frame header bits and data field bits is 0.
0	3, [31:16]	r	**Reserved** Read as 0; should be written with 0.

The Transmitter Status Register TSTATR contains transmitter specific status information.

TSTATR
Transmitter Status Register (14_H) Reset Value: $0000\ 0000_H$



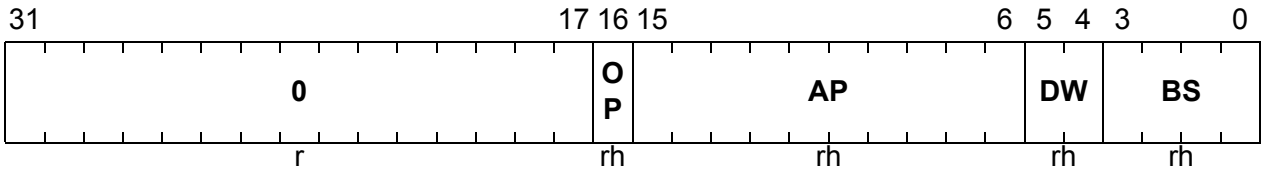
Field	Bits	Type	Description
RDC	[4:0]	rh	**Ready Delay Counter** This bit field counts TCLK periods after the end of a frame transmission. When the TVALID signal goes to low level, RDC is cleared to zero and starts counting up the TCLK clock periods until a TREADY high level is detected (see **Page 21-22**).
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 21-42**). 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 21-45**). 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 if read.

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**

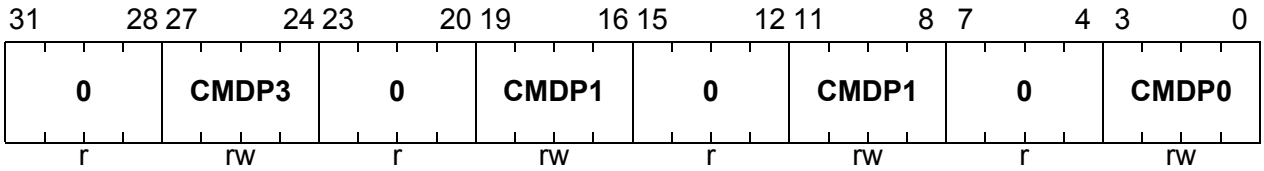


Field	Bits	Type	Description
BS	[3:0]	rh	**Buffer Size** This bit field indicates the coded buffer size of the pipe x Remote Window in the receiving controller. BS further determines how many address offset bits are transmitted in a Write Offset and Data Frame or in a Discrete Read Frame. When register TPxBAR is written for generation of a Copy Base Address Frame, BS is updated by the Copy Base Address Frame (see **Page 21-26**). 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 $\cdots_B$ … 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 21-28** and **Page 21-32**). 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 21-24** and **Page 21-45**).
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 21-24** and **Page 21-45**). 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.

The Transmitter Command Register TCMDR contains the command codes that are used during Command Frame transmission. 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 service request output SR0 0010_B Activate service request output SR1 0011_B Activate service request output SR2 0100_B Activate service request output SR3 Other bit combinations are reserved for future use; no further action occurs 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 …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 break output $\overline{\text{BRKOUT}}$ Other bit combinations are reserved for future use; no further action occurs 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.

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	0			
r						rh	rh	rh	rh	rh	rh	r			

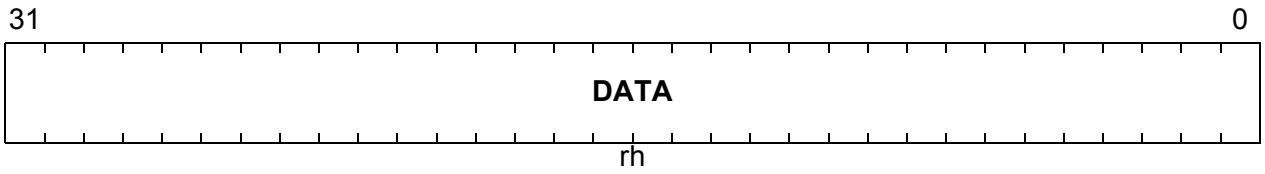
Field	Bits	Type	Description
CVx (x = 0-3)	4 + x	rh	**Command Valid** Bit is set by hardware when a TCMDR.CMDPx bit field is written. It is cleared by hardware when the Command Frame has been correctly transmitted. CVx can be set or cleared by software via bits SCR.SCVx or SCR.CCVx.
AV	8	rh	**Answer Valid** 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.
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.

Field	Bits	Type	Description
DVx (x = 0-3)	16 + x	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.
RPx (x = 0-3)	20 + x	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	[3:0], [15:10], [31:26]	r	**Reserved** Read as 0.

21.4.5 Transmitter Data/Address Registers

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.

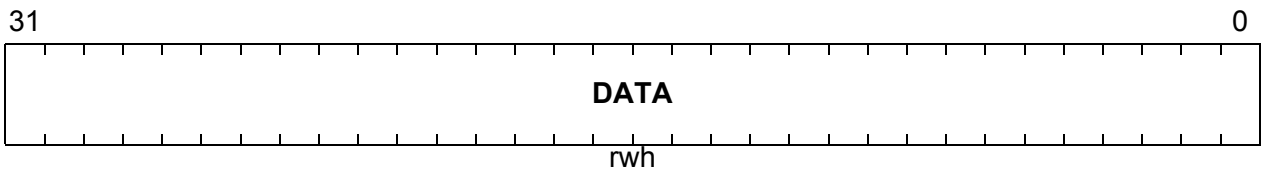
TPxDATAR (x = 0-3)
Transmitter Pipe x Data Register (40_H+4_H*x) Reset Value: 0000 0000_H



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.

The Transmitter Data Read Answer Register TDRAR contains the read data for the transmission of an Answer Frame.

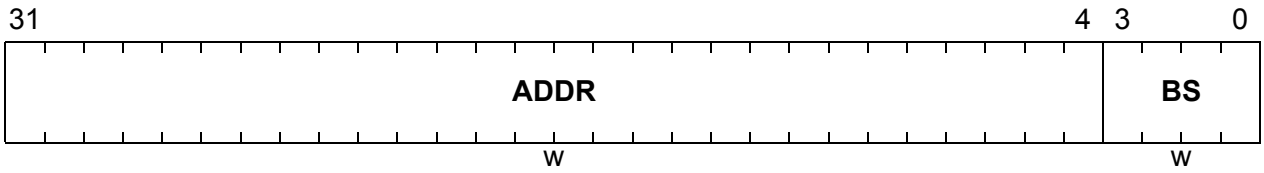
TDRAR
Transmitter Data Read Answer Register
(50_H) Reset Value: 0000 0000_H



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).

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.

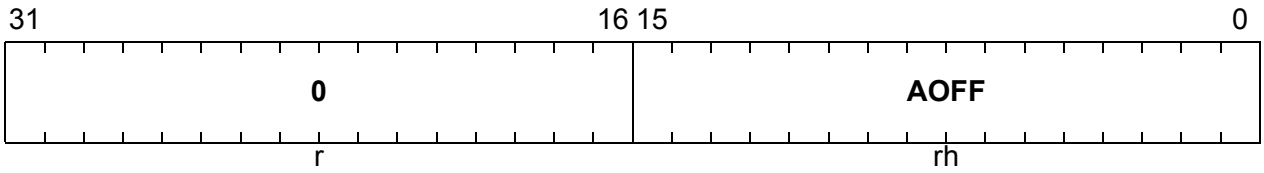
TPxBAR (x = 0-3)
Transmitter Pipe x Base Address Register
(54_H+4_H*x) **Reset Value: 0000 0000_H**



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 $\ldots_B$ … 1110_B 15-bit offset address of Remote Window 1111_B 16-bit offset address of Remote Window
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].

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.

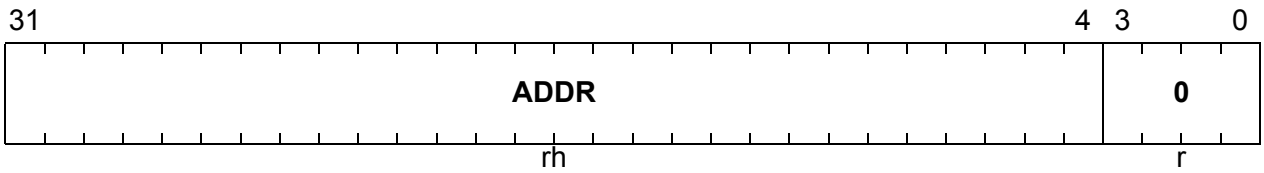
TPxAOFR (x = 0-3)
Transmitter Pipe x Address Offset Register
(30_H+4_H*x) **Reset Value: 0000 0000_H**



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.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

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

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

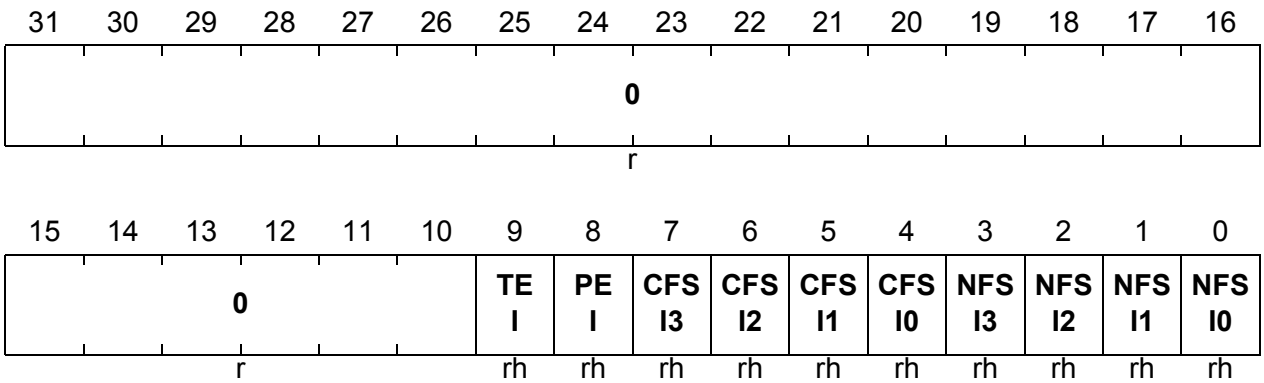
Field	Bits	Type	Description
NFSIEx (x = 0-3)	x	rw	**Normal Frame Sent in Pipe x Interrupt Enable** 0_B Normal frame sent in pipe x event is disabled for activation of an SRx line. 1_B Normal frame sent in pipe x event is enabled for activation of an SRx line.
CFSIEx (x = 0-3)	4 + x	rw	**Command Frame 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.
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.

Field	Bits	Type	Description
NFSIRx (x = 0-3)	16 + x	w	**Normal Frame Sent in Pipe x Flag Clear** 0_B No action. 1_B Clear TISR.NFSIx.
CFSIRx (x = 0-3)	20 + x	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.

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: $0000\ 0000_H$**

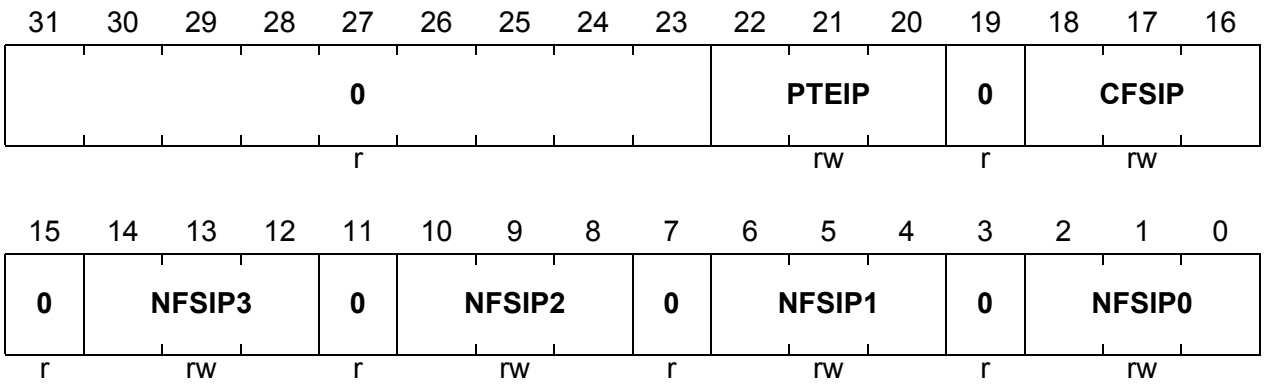


Field	Bits	Type	Description
NFSIx (x = 0-3)	x	rh	**Normal Frame Sent in Pipe x Flag** 0_B A Normal Frame has not yet been sent. 1_B A Write or Read Frame has been correctly sent and acknowledged for pipe x. The service request output that can be activated by NFSIx is defined by TINPR.NFSIPx.
CFSIx (x = 0-3)	4 + x	rh	**Command Frame Sent in Pipe x Flag** 0_B A Command Frame has not yet been sent. 1_B A Command Frame has been correctly sent and acknowledged for pipe x. The service request output that can be activated by CFSIx is defined by TINPR.CFSIP.
PEI	8	rh	**Parity Error Flag** 0_B A parity error event has not yet been detected. 1_B A parity error event has been detected. The service request output that can be activated by PEI is defined by TINPR.PTEIP.
TEI	9	rh	**Time-Out Error Flag** 0_B A time-out error event has not yet been detected. 1_B A time-out error event has been detected. The service request output that can be activated by TEI is defined by TINPR.PTEIP.

Field	Bits	Type	Description
0	[31:10]	r	**Reserved** Read as 0.

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.

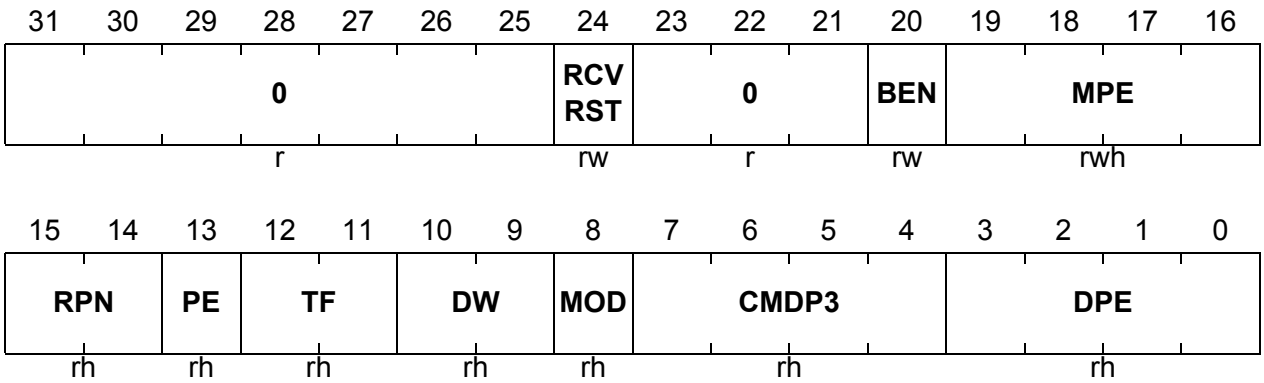
Field	Bits	Type	Description
NFSIP3	[14:12]	rw	**Normal Frame Sent in Pipe 3 Interrupt Pointer** This bit field determines which service request output SRx becomes active when a Normal Frame sent in pipe 3 event occurs (if enabled). Coding see NFSIP0.
CFSIP	[18:16]	rw	**Command Frame Sent Interrupt Pointer** 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 Status/Control Registers

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 21-66**) with a value given in the implementation chapter (see **Page 21-127**).

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 21-39**). 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.

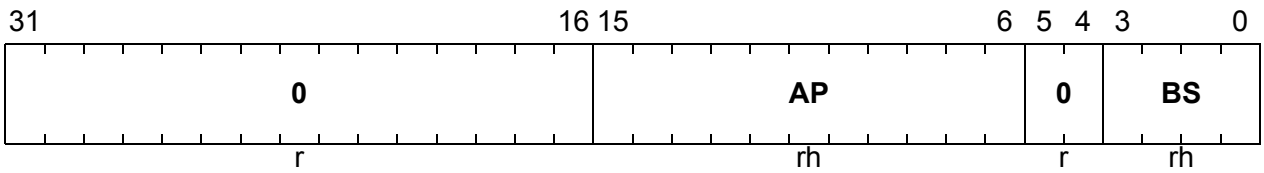
Field	Bits	Type	Description
MOD	8	rh	**Mode of Operation** This bit determines the data transfer operation mode of the MLI receiver. Bit MOD can be set by hardware with the reception of a pipe 2 Command Frame (see Page 21-98). It can be set or cleared by software via bits SCR.SMOD or SCR.CMOD. 0_B Automatic 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. DW 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 21-42). 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 21-42). 0000_B A receiver parity event is generated if 1 receiver error condition is detected. 0001_B A receiver parity event is generated if 1 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.

The Receiver Pipe x Status Register RPxSTATR indicates the coded buffer size which represents the Remote Window Size of 2 Bytes to 64KBytes 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$_H$**

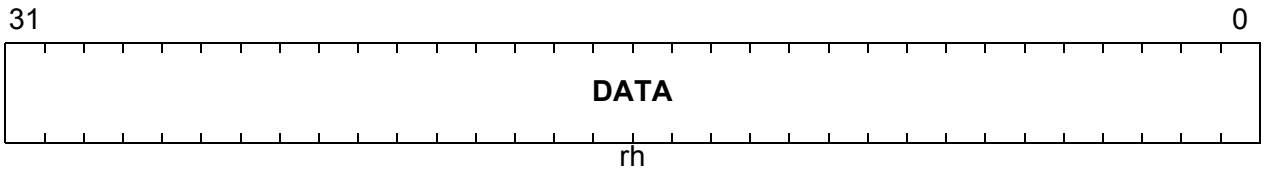


Field	Bits	Type	Description
BS	[3:0]	rh	**Buffer Size** This bit field indicates the size of pipe x Remote Window in the receiving controller. It is updated by hardware when a Copy Base Address Frame has been received (see **Page 21-26**). 0000_B 1-bit offset address of Remote Window 0001_B 2-bit offset address of Remote Window 0010_B 3-bit offset address of Remote Window …$_B$ … 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 21-45**).
0	[5:4], [31:16]	r	**Reserved** Read as 0.

21.4.8 Receiver Data/Address Registers

The Receiver Data Register RDATAR is a read-only register that stores data received by a Write Frame or an Answer Frame.

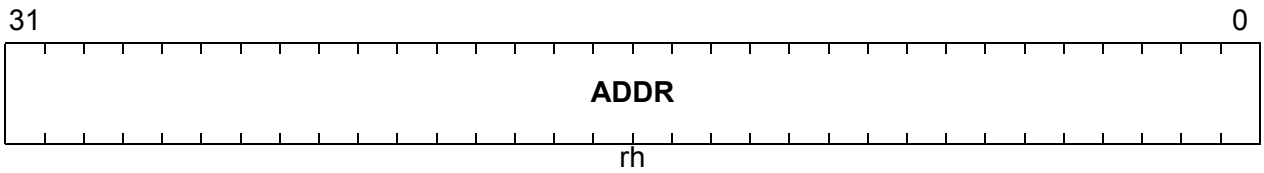
RDATAR
Receiver Data Register **(90_H)** **Reset Value: 0000 0000_H**



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)

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.

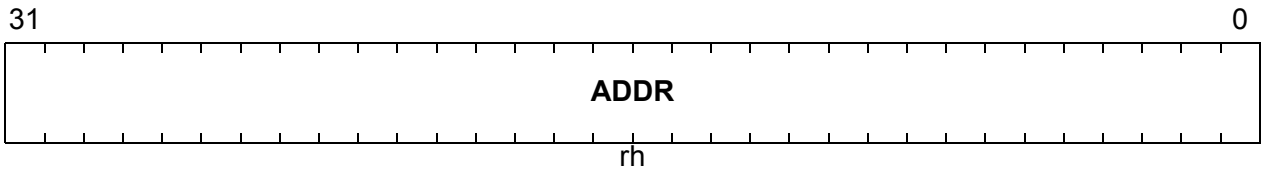
RPxBAR (x = 0-3)
Receiver Pipe x Base Address Register
($6C_H+4_H*x$) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
ADDR	[31:0]	rh	**Address** ADDR indicates the complete target address for the pipe x Remote Window. When 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. When 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. When 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. When an Answer Frame is received, ADDR is not changed.

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). When a Copy Base Address Frame is received, ADDR is unchanged. When 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 bitsADDR[m-1:0] are replaced by the received offset. When 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. When an Answer Frame is received, ADDR becomes invalid.

21.4.9 Receiver Interrupt Registers

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	

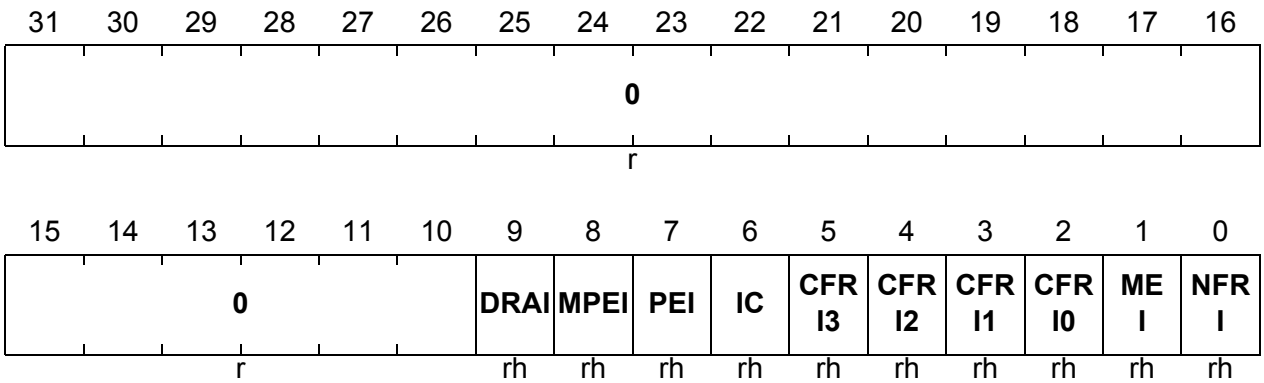
Field	Bits	Type	Description
NFRIE	[1:0]	rw	**Normal Frame Received Interrupt Enable** This bit field defines if an SRx output is activated if a Normal Frame is correctly received. 00_B The SRx activation is disabled. 01_B The selected SRx line is activated each time a Normal Frame is correctly received. 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
CFRIEx (x = 0-3)	2 + x	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.
CFRIRx (x = 0-3)	18 + x	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.

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$_H$



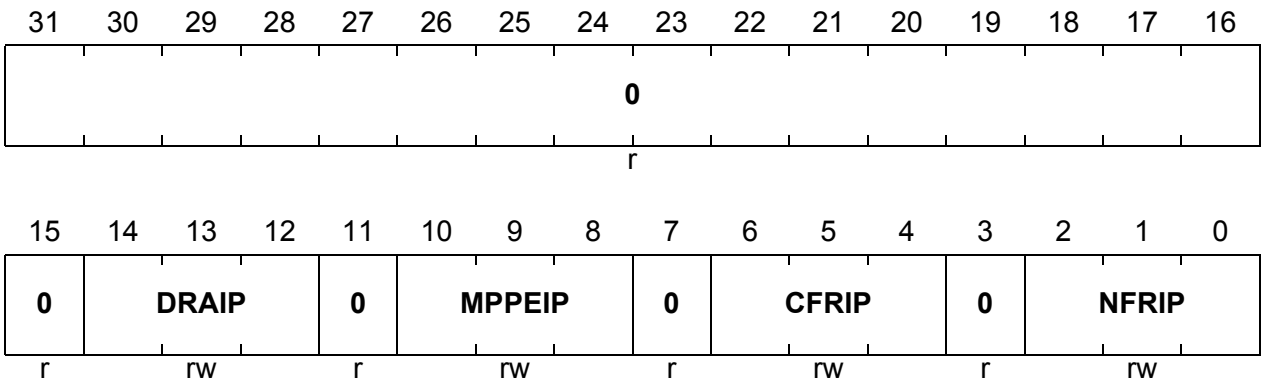
Field	Bits	Type	Description
NFRI	0	rh	**Normal Frame Received Interrupt Flag** This flag is set when a Write or a Read Frame has been received. The service request output that is activated is defined by RINPR.NFRIP.
MEI	1	rh	**MLI Move Engine Interrupt Flag** This flag is set when the move engine has finished an operation (read or write, depending on received frame). The service request output that is activated is defined by RINPR.MPPEIP.
CFRIx (x = 0-3)	2 + x	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.

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 TC1766

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 the DMA controller.

The data, clock, and control lines of each MLI receiver and transmitter are connected to GPIO lines. Alternate functions of Port 2 and Port 5 lines are assigned to the MLI0 module I/O lines while alternate functions of Port 5 lines are assigned to the MLI1 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 in the local loop back mode. In this mode, the connections of the signals in pair are:

- TCLK/RCLKD
- TREADYD/RREADYD
- TVALIDD/RVALIDD
- TDATA/RDATAD

Figure 21-51 and **Figure 21-52** show how the MLI0 and MLI1 modules are interconnected to port lines and other on-chip functional blocks.

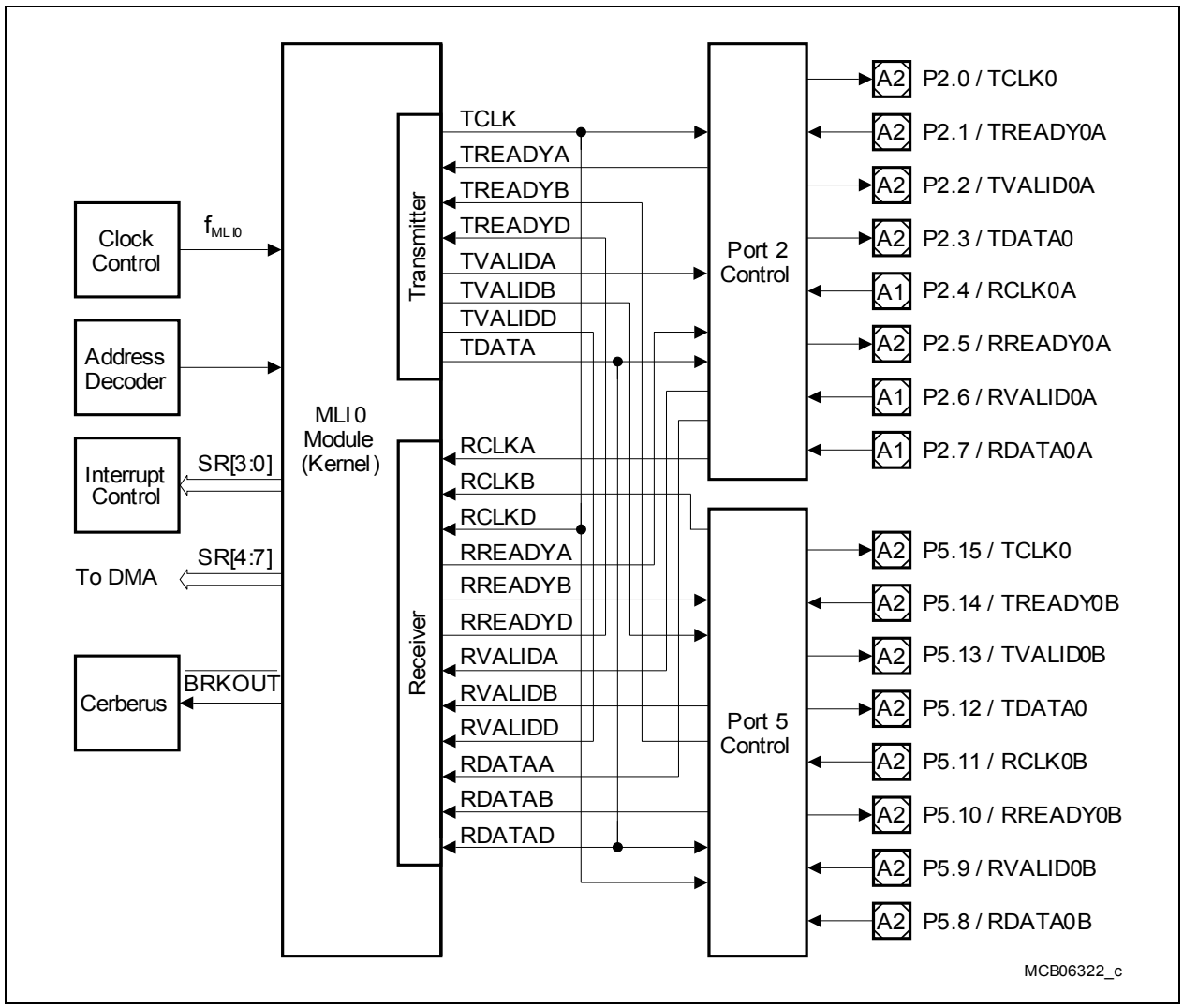


Figure 21-51 MLI0 Module Implementation and Interconnections

When programming the MLI0_OICR register, the following additional items must be considered:

- Unused transmitter/receiver output lines with index “C” (TVALIDC and RREADYC, not shown in the figure above) are not connected.
- Unused transmitter/receiver input lines with index “C” (TREADYC, RCLKC, RVALIDC, and RDATAC, not shown in the figure above) are connected to low level.

See also **Page 21-130** for additional details on I/O line control and function selection.

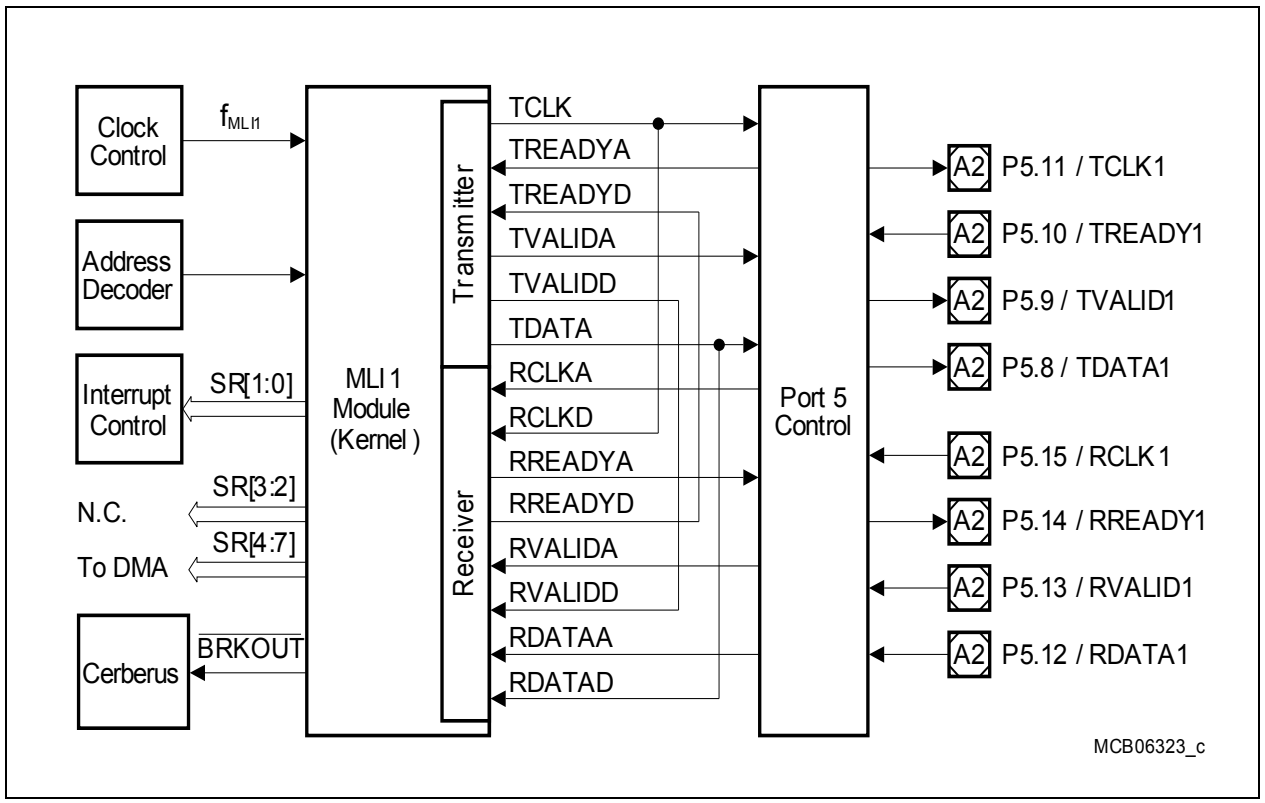


Figure 21-52 MLI1 Module Implementation and Interconnections

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 TC1766.

See also **Page 21-130** for additional details on I/O line control and function.

21.5.2 MLI Module External Registers

Figure 21-53 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.

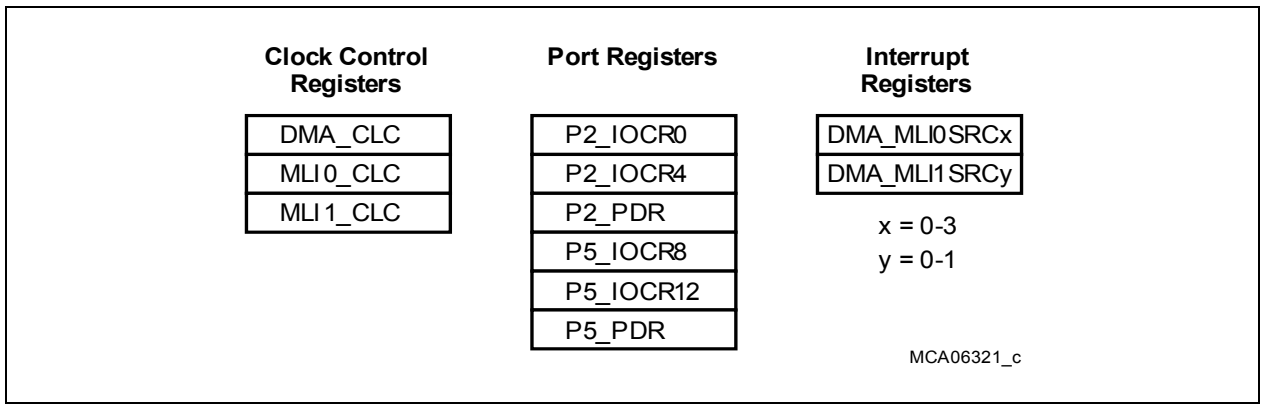


Figure 21-53 MLI0/MLI1 Implementation-Specific Special Function Registers

21.5.2.1 Automatic Register Overwrite

The following values are applied after reset (see **Page 21-66**).

- 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 21-54**.

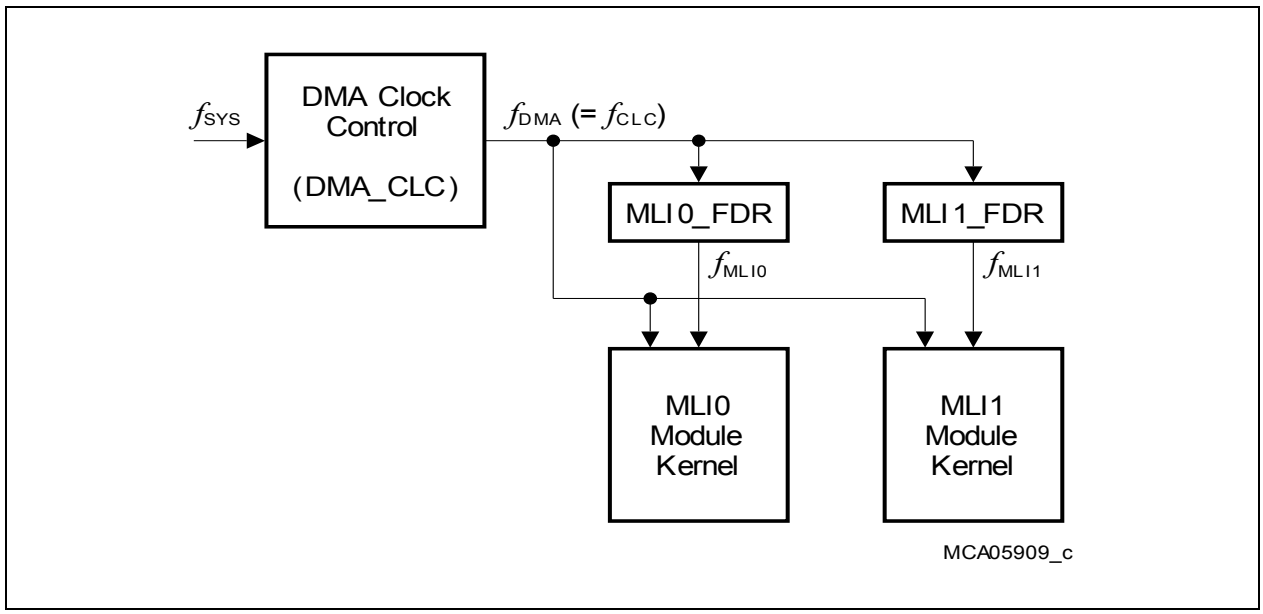


Figure 21-54 Clock Configuration of the MLI Modules

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 (FDR description see **Page 21-79** and baud rate generation see **Section 21.2.7** on **Page 21-65**).

*Note: Additional details on the fractional divider register functionality are described in section **"Fractional Divider Operation" on Page 3-29** of the TC1766 User's Manual System Units part (Volume 1).*

- 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 21-65**) 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 $f_{DMA} = f_{SYS}$.

The receiver baud rate is defined by the following formula.

$$\text{Baud rate}_{RLCKmax} = f_{MLI} \tag{21.6}$$

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 21-125** and **Page 21-126**). 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 2 and Port 5 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 21-51**, receiver I/O line control see **Page 21-51**). 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	P2.0 / TCLK0	P2_IOCR0.PC0 = $1X10_B$ [1)] MLI0_OICR.TCE = 1 [2)] MLI0_OICR.TCP = X	Output
	P2.1 / TREADY0A	P2_IOCR0.PC1 = $0XXX_B$ [1)] MLI0_OICR.TRE = 1 MLI0_OICR.TRP = X [2)] MLI0_OICR.TRS = 00_B	Input
	P2.2 / TVALID0A	P2_IOCR0.PC2 = $1X10_B$ [1)] MLI0_OICR.TVEA = 1 MLI0_OICR.TVPA = X [2)]	Output
	P2.3 / TDATA0	P2_IOCR0.PC3 = $1X10_B$ [1)] MLI0_OICR.TDP = X [2)]	Output
	P2.4 / RCLK0A	P2_IOCR4.PC4 = $0XXX_B$ [1)] MLI0_OICR.RCE = 1 MLI0_OICR.RCP = X [2)] MLI0_OICR.RCS = 00_B	Input

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	P2.5 / RREADY0A	P2_IOCR4.PC5 = $1X10_B$ [1)] MLI0_OICR.RRS = 00_B MLI0_OICR.RRPA = X [2)]	Output
	P2.6 / RVALID0A	P2_IOCR4.PC6 = $0XXX_B$ [1)] MLI0_OICR.RVE = 1 MLI0_OICR.RVP = X [2)] MLI0_OICR.RVS = 00_B	Input
	P2.7 / RDATA0A	P2_IOCR4.PC7 = $0XXX_B$ [1)] MLI1_OICR.RDP = X [2)] MLI1_OICR.RDS = 00_B	Input
	P5.15 / TCLK0	P5_IOCR12.PC15 = $1X01_B$ [1)] MLI0_OICR.TCE = 1 [2)] MLI0_OICR.TCP = X	Output
	P5.14 / TREADY0B	P5_IOCR12.PC14 = $0XXX_B$ [1)] MLI0_OICR.TRE = 1 MLI0_OICR.TRP = X [2)] MLI0_OICR.TRS = 01_B	Input
	P5.13 / TVALID0B	P5_IOCR12.PC13 = $1X10_B$ [1)] MLI0_OICR.TVEB = 1 MLI0_OICR.TVPB = X [2)]	Output
	P5.12 / TDATA0	P5_IOCR12.PC12 = $1X01_B$ [1)] MLI0_OICR.TDP = X [2)]	Output
	P5.11 / RCLK0B	P5_IOCR8.PC11 = $0XXX_B$ [1)] MLI0_OICR.RCE = 1 MLI0_OICR.RCP = X [2)] MLI0_OICR.RCS = 01_B	Input
	P5.10 / RREADY0B	P5_IOCR8.PC10 = $1X01_B$ [1)] MLI0_OICR.RRS = 01_B MLI0_OICR.RRPA = X [2)]	Output
	P5.9 / RVALID0B	P5_IOCR8.PC9 = $0XXX_B$ [1)] MLI0_OICR.RVE = 1 MLI0_OICR.RVP = X [2)] MLI0_OICR.RVS = 01_B	Input
	P5.8 / RDATA0B	P5_IOCR4.PC8 = $0XXX_B$ [1)] MLI0_OICR.RDP = X [2)] MLI0_OICR.RDS = 01_B	Input

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	P5.11 / TCLK1	P5_IOCR8.PC11 = $1X01_B$ [1)] MLI1_OICR.TCE = 1 [2)] MLI1_OICR.TCP = X	Output
	P5.10 / TREADY1	P5_IOCR8.PC10 = $0XXX_B$ [1)] MLI1_OICR.TRE = 1 MLI1_OICR.TRP = X [2)] MLI1_OICR.TRS = 00_B	Input
	P5.9 / TVALID1	P5_IOCR8.PC9 = $1X01_B$ [1)] MLI1_OICR.TVEA = 1 MLI1_OICR.TVPA = X [2)]	Output
	P5.8 / TDATA1	P5_IOCR8.PC8 = $1X01_B$ [1)] MLI1_OICR.TDP = X [2)]	Output
	P5.15 / RCLK1	P5_IOCR12.PC15 = $0XXX_B$ [1)] MLI1_OICR.RCE = 1 MLI1_OICR.RCP = X [2)] MLI1_OICR.RCS = 00_B	Input
	P5.14 / RREADY1	P5_IOCR12.PC14 = $1X01_B$ [1)] MLI1_OICR.RRS = 00_B MLI1_OICR.RRPA = X [2)]	Output
	P5.13 / RVALID1	P5_IOCR12.PC13 = $0XXX_B$ [1)] MLI1_OICR.RVE = 1 MLI1_OICR.RVP = X [2)] MLI1_OICR.RVS = 00_B	Input
	P5.12 / RDATA1	P5_IOCR12.PC12 = $0XXX_B$ [1)] MLI1_OICR.RDP = X [2)] MLI1_OICR.RDS = 00_B	Input

1) For possible PCx bit field combinations, see **Table 21-12**.

2) With polarity control bit = 0, normal polarity is selected. With polarity control bit = 1, inverted polarity is selected.

Table 21-12 PCx Coding

PCx[3:0]	I/O	Output Characteristics	Selected Pull-up/Pull-down/ Selected Output Function
$0X00_B$	Input	–	No pull device connected
$0X01_B$			Pull-down device connected
$0X10_B$ 1)			Pull-up device connected
$0X11_B$			No pull device connected
1001_B	Output	Push-pull	Output function ALT1
1101_B		Open-drain	Output function ALT1
1010_B		Push-pull	Output function ALT2
1110_B		Open-drain	Output function ALT2
1011_B		Push-pull	Output function ALT3
1111_B		Open-drain	Output function ALT3

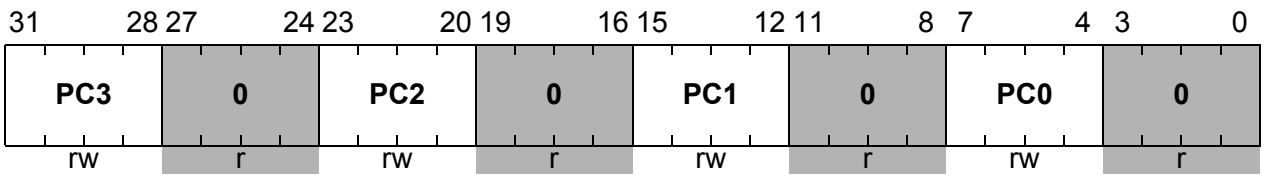
1) This bit field value is default after reset.

21.5.4.2 Input/Output Control Register

The following pages describe all IOCR registers that contain MLI0/MLI1 related input/output control bit fields. Shaded bits and bit fields are “don’t care” for MLI0/MLI1 I/O port control purposes.

P2_IOCR0
Port 2 Input/Output Control Register 0

(10_H) **Reset Value: 2020 2020_H**

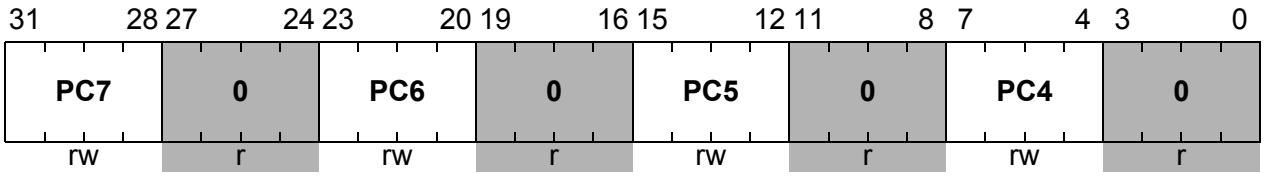


Field	Bits	Type	Description
		rw	**Port Input/Output Control for Port 2.[3:0]**1)
PC0,	[7:4],		Port input/output control for P2.0/TCLK0
PC1,	[15:12],		Port input/output control for P2.1/TREADY0A
PC2,	[23:20],		Port input/output control for P2.2/TVALID0A
PC3	[31:28]		Port input/output control for P2.3/TDATA0

1) For coding of bit field, see **Table 21-12**. Shaded bits and bit fields are “don’t care” for MLI I/O port control.

P2_IOCR4
Port 2 Input/Output Control Register 4

(14$_H$) Reset Value: 2020 2020$_H$

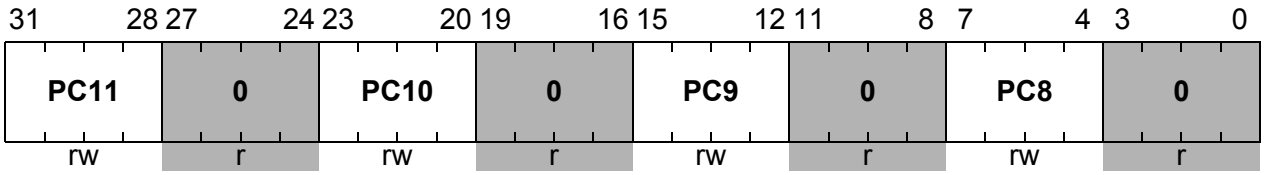


Field	Bits	Type	Description
		rw	**Port Input/Output Control for Port 2.[7:4]**[1]
PC4,	[7:4],		Port input/output control for P2.4/RCLK0A
PC5,	[15:12],		Port input/output control for P2.5/RREADY0A
PC6,	[23:20],		Port input/output control for P2.6/RVALID0A
PC7	[31:28]		Port input/output control for P2.7/RDATA0A

1) For coding of bit field, see **Table 21-12**. Shaded bits and bit fields are “don’t care” for MLI I/O port control.

P5_IOCR8
Port 5 Input/Output Control Register 8

(18$_H$) Reset Value: 2020 2020$_H$

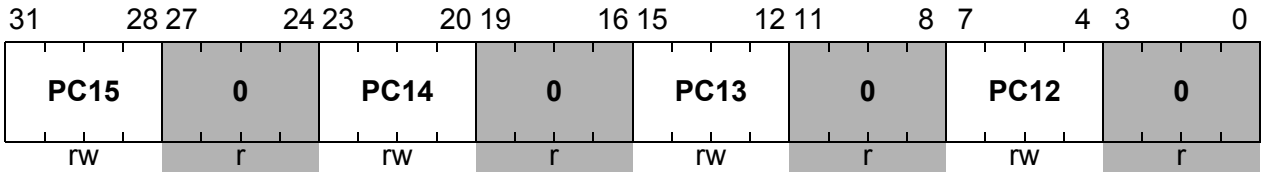


Field	Bits	Type	Description
		rw	**Port Input/Output Control for Port 5.[11:8]**[1]
PC8,	[7:4],		Port input/output control for P5.8/RDATA0B/TDATA1
PC9,	[15:12],		Port input/output control for P5.9/RVALID0B/TVALID1
PC10,	[23:20],		Port input/output control for P5.10/RREADY0B/TREADY1
PC11	[31:28]		Port input/output control for P5.11/RCLK0B/TCLK1

1) For coding of bit field, see **Table 21-12**. Shaded bits and bit fields are “don’t care” for MLI I/O port control.

P5_IOCR12
Port 5 Input/Output Control Register 12

($1C_H$) Reset Value: $2020\ 2020_H$



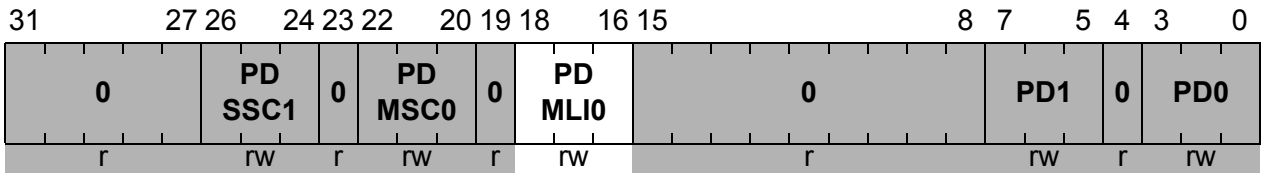
Field	Bits	Type	Description
		rw	**Port Input/Output Control for Port 5.[15:12]**[1]
PC12,	[7:4],		Port input/output control for P5.12/TDATA0/RDATA1
PC13,	[15:12],		Port input/output control for P5.13/TVALID0B/RVALID1
PC14,	[23:20],		Port input/output control for P5.14/TREADY0B/RREADY1
PC15	[31:28]		Port input/output control for P5.15/TCLK0/RCLK1

1) For coding of bit field, see **Table 21-12**. Shaded bits and bit fields are “don’t care” for MLI I/O port control.

21.5.4.3 Pad Driver Characteristics Selection

The Port 2 and Port 5 pad driver mode registers contain bit fields that determine the output driver strength and the slew rate of MLI output lines. A detailed description of all available PDx bit field combinations is given in Chapter “GPIO Ports” in the TC1766 System Units User’s Manual. Shaded bits and bit fields are “don’t care” for MLI0/MLI1 I/O port control purposes.

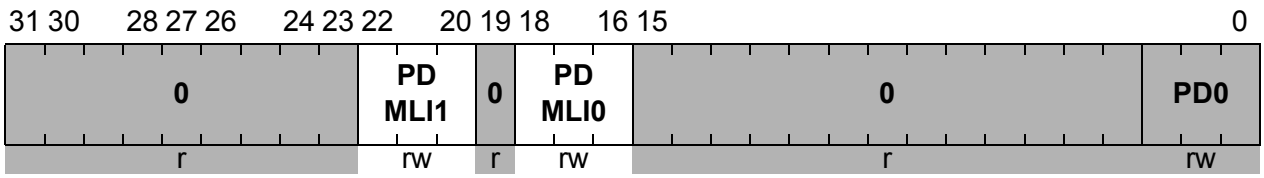
P2_PDR
Port 2 Pad Driver Mode Register (40_H) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
PDMLI0	[18:16]	rw	**Pad Driver Mode for MLI0 Outputs at P2.0/TCLK0, P2.2/TVALID0A, P2.3/TDATA0 and P2.5/RREADY0A**[1)]

1) For coding of bit fields, see **Table 21-13**. Shaded bits and bit fields are “don’t care” for MLI I/O port control.

P5_PDR
Port 5 Pad Driver Mode Register (40_H) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
PDMLI0	[18:16]	rw	**Pad Driver Mode for P5.15/TCLK0, P5.13/TVALID0B, P5.12/TDATA0 and P5.10/RREADY0B**[1)]
PDMLI1	[22:20]	rw	**Pad Driver Mode for P5.14/RREADY1, P5.11/TCLK1 and P5.9/TVALID1 and P5.8/TDATA1**[1)]

1) For coding of bit fields, see **Table 21-13**. Shaded bits and bit fields are “don’t care” for MLI I/O port control.

PDx Selection Table

Table 21-13 Pad Driver Mode Mode Selection (Class A2 Pads)

PDx Bit Field	Driver Strength	Signal Transitions
000_B	Strong driver	Sharp edge[1)]
001_B		Medium edge[1)]
010_B		Soft edge[1)]
011_B	Weak driver	–
100_B	Medium driver	–
101_B		
110_B		
111_B	Weak driver	–

1) In strong driver mode, the output driver characteristics of class A2 pads can be additionally controlled by the temperature compensation logic.

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 TC1766, 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 TC1766 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 (see **Page 11-96**) of the TC1766 System Units User's Manual.

All MLI service request output connections are listed in **Table 21-14**.

Table 21-14 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_REQI6 CH04_REQI6	DMA Channel 00 Request Input 6 DMA Channel 04 Request Input 6
	SR5	CH01_REQI6 CH05_REQI6	DMA Channel 01 Request Input 6 DMA Channel 05 Request Input 6
	SR6	CH02_REQI6 CH06_REQI6	DMA Channel 02 Request Input 6 DMA Channel 06 Request Input 6
	SR7	CH03_REQI6 CH07_REQI6	DMA Channel 03 Request Input 6 DMA Channel 07 Request Input 6

Table 21-14 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	CH00_REQI7 CH04_REQI7	DMA Channel 00 Request Input 7 DMA Channel 04 Request Input 7
	SR5	CH01_REQI7 CH05_REQI7	DMA Channel 01 Request Input 7 DMA Channel 05 Request Input 7
	SR6	CH02_REQI7 CH06_REQI7	DMA Channel 02 Request Input 7 DMA Channel 06 Request Input 7
	SR7	CH03_REQI7 CH07_REQI7	DMA Channel 03 Request Input 7 DMA Channel 07 Request Input 7

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.

21.5.6 Access Protection

21.5.6.1 Fixed Address Range Definition

Table 21-15 below shows the fixed address ranges covered by the access protection (for read and write) of the MLI modules. The access enable bits AER.AENx are related to address range x. If bit AENx is set, read/write accesses to the corresponding address range x are allowed. Some of the access enable bits AENx control more than one address range.

Table 21-15 MLI Access Protection Address Ranges

Access Protection Range			Related Module(s)
No. x	**Enable Bit in AER**	**Selected Address Range**	
0	AEN0	F000 0000$_H$ to F000 00FF$_H$ F010 C200$_H$ to F010 C2FF$_H$	SCU, incl. WDT MEMCHK
1	AEN1	F000 0100$_H$ to F000 01FF$_H$	SBCU
2	AEN2	F000 0200$_H$ to F000 02FF$_H$	STM
3	AEN3	F000 0400$_H$ to F000 04FF$_H$	OCDS
4	AEN4	F000 0800$_H$ to F000 08FF$_H$	MSC0
5	AEN5	F000 0A00$_H$ to F000 0AFF$_H$	ASC0
6	AEN6	F000 0B00$_H$ to F000 0BFF$_H$	ASC1
7	AEN7	F000 0C00$_H$ to F000 0FFF$_H$	P0, P1, P2, P3
8	AEN8	F000 1000$_H$ to F000 11FF$_H$	P4, P5
9	AEN9	F000 1800$_H$ to F000 1FFF$_H$	GPTA0
10	AEN10	F000 3C00$_H$ to F000 3EFF$_H$	DMA
11	AEN11	F000 4000$_H$ to F000 5FFF$_H$	MultiCAN
12	AEN12	F004 0000$_H$ to F005 FFFF$_H$	PCP Registers, PCP Data Memory
13	AEN13	F006 0000$_H$ to F006 7FFF$_H$	PCP Code Memory
14	AEN14	F010 0100$_H$ to F010 01FF$_H$	SSC0
15	AEN15	F010 0200$_H$ to F010 02FF$_H$	SSC1
16	AEN16	F010 0300$_H$ to F010 03FF$_H$	FADC
17	AEN17	F010 0400$_H$ to F010 05FF$_H$	ADC0
18	AEN18	–	–

Table 21-15 MLI Access Protection Address Ranges (cont'd)

Access Protection Range			Related Module(s)
No. x	**Enable Bit in AER**	**Selected Address Range**	
19	AEN19	F010 C000$_H$ to F010 C0FF$_H$ F01E 0000$_H$ to F01E 7FFF$_H$ F020 0000$_H$ to F023 FFFF$_H$	MLI0 Module, MLI0 Small TWs, MLI0 Large TWs
20	AEN20	F010 C100$_H$ to F010 C1FF$_H$ F01E 8000$_H$ to F01E FFFF$_H$ F024 0000$_H$ to F027 FFFF$_H$	MLI1, MLI1 Small TWs, MLI1 Large TWs
21	AEN21	F7E0 FF00$_H$ to F7E0 FFFF$_H$ F7E1 0000$_H$ to F7E1 FFFF$_H$ F800 0500$_H$ to F87F FFFF$_H$	CPS CPU SFRs & GPRs PMU, Flash Regs, LBCU, DMI, PMI, LFI
22	AEN22	–	–
23	AEN23	8000 0000$_H$ to 807F FFFF$_H$ A000 0000$_H$ to A07F FFFF$_H$	Program Flash Space
24	AEN24	–	–
25	AEN25	8FE0 0000$_H$ to 8FEF FFFF$_H$ AFE0 0000$_H$ to AFEF FFFF$_H$	Data Flash Space
26	AEN26	8FF0 0000$_H$ to 8FFF BFFF$_H$ AFF0 0000$_H$ to AFFF BFFF$_H$	Emulation Device Memory Space
27	AEN27	8FFF C000$_H$ to 8FFF FFFF$_H$ AFFF C000$_H$ to AFFF FFFF$_H$	Boot ROM
28	AEN28	–	–
29	AEN29	E800 0000$_H$ to E83F FFFF$_H$	LMU Image (E80x translated to C00x)
30	AEN30	E840 0000$_H$ to E84F FFFF$_H$	DMI Image (E84x translated to D00x)
31	AEN31	E850 0000$_H$ to E85F FFFF$_H$	PMI Image (E85x translated to D40x)

21.5.6.2 Programmable Address Sub-Range Definitions

In the TC1766, two of four possible programmable address range extensions are implemented for internal memory areas:

- 64-Kbyte Overlay RAM (OVRAM) assigned to address range 29;
  sub-ranges are controlled by bit fields ARR.SIZE1 and ARR.SLICE1.
  (see **Table 21-16**).
- 64-Kbyte DMI RAM assigned as address range 30;
  sub-ranges are controlled by bit fields ARR.SIZE2 and ARR.SLICE2.
  (see **Table 21-17**).

Attention: Bit fields ARR.SLICE0/ARR.SIZE0 and ARR.SLICE3/ARR.SIZE3 are not used in TC1766.

The coding of bit fields SIZE1 and SLICE1 for the OVRAM sub-range access protection are defined as shown in **Table 21-16**.

Table 21-16 OVRAM Address Protection Sub-Range Definitions

SIZE1	Sub-Ranges	SLICE1	Selected Address Range
000_B	32 sub-ranges of 512 bytes	00000_B 00001_B … 11111_B	$E800\ 0000_H$ to $E800\ 01FF_H$ $E800\ 0200_H$ to $E800\ 03FF_H$ … $E800\ 3E00_H$ to $E800\ 3FFF_H$
001_B	32 sub-ranges of 1 Kbyte	00000_B 00001_B … 11111_B	$E800\ 0000_H$ to $E800\ 03FF_H$ $E800\ 0400_H$ to $E800\ 07FF_H$ … $E800\ 7C00_H$ to $E800\ 7FFF_H$
010_B	32 sub-ranges of 2 Kbytes	00000_B 00001_B … 11111_B	$E800\ 0000_H$ to $E800\ 07FF_H$ $E800\ 0800_H$ to $E800\ 0FFF_H$ … $E800\ F800_H$ to $E800\ FFFF_H$
011_B	16 sub-ranges of 4 Kbytes	$X0000_B$ $X0001_B$ … $X1111_B$	$E800\ 0000_H$ to $E800\ 0FFF_H$ $E800\ 1000_H$ to $E800\ 1FFF_H$ … $E800\ F000_H$ to $E800\ FFFF_H$
100_B	8 sub-ranges of 8 Kbytes	$XX000_B$ $XX001_B$ … $XX111_B$	$E800\ 0000_H$ to $E800\ 1FFF_H$ $E800\ 2000_H$ to $E800\ 3FFF_H$ … $E800\ E000_H$ to $E800\ FFFF_H$

Table 21-16 OVRAM Address Protection Sub-Range Definitions (cont'd)

SIZE1	Sub-Ranges	SLICE1	Selected Address Range
101_B	4 sub-ranges of 16 Kbytes	$XXX00_B$ $XXX01_B$ $XXX10_B$ $XXX11_B$	$E800\ 0000_H$ to $E800\ 3FFF_H$ $E800\ 4000_H$ to $E800\ 7FFF_H$ $E800\ 8000_H$ to $E800\ BFFF_H$ $E800\ C000_H$ to $E800\ FFFF_H$
110_B	2 sub-ranges of 32 Kbytes	$XXXX0_B$ $XXXX1_B$	$E800\ 0000_H$ to $E800\ 7FFF_H$ $E800\ 8000_H$ to $E800\ FFFF_H$
111_B	64 Kbytes	$XXXXX_B$	$E800\ 0000_H$ to $E800\ FFFF_H$

The coding of bit fields SIZE2 and SLICE2 for the DMI RAM sub-range access protection (with address translation from E840 to D000 in the LFI) are coded as shown in Table 21-17.

Table 21-17 DMI RAM Address Protection Sub-Range Definitions

SIZE2	Sub-Ranges	SLICE2	Selected Address Range
000_B	32 sub-ranges of 512 bytes	00000_B 00001_B … 11111_B	$E840\ 0000_H$ to $E840\ 01FF_H$ $E840\ 0200_H$ to $E840\ 03FF_H$ … $E840\ 3E00_H$ to $E840\ 3FFF_H$
001_B	32 sub-ranges of 1 Kbyte	00000_B 00001_B … 11111_B	$E840\ 0000_H$ to $E840\ 03FF_H$ $E840\ 0400_H$ to $E840\ 07FF_H$ … $E840\ 7C00_H$ to $E840\ 7FFF_H$
010_B	32 sub-ranges of 2 Kbytes	00000_B 00001_B … 11111_B	$E840\ 0000_H$ to $E840\ 07FF_H$ $E840\ 0800_H$ to $E840\ 0FFF_H$ … $E840\ F800_H$ to $E840\ FFFF_H$
011_B	16 sub-ranges of 4 Kbytes	$X0000_B$ $X0001_B$ … $X1111_B$	$E840\ 0000_H$ to $E840\ 0FFF_H$ $E840\ 1000_H$ to $E840\ 1FFF_H$ … $E840\ F000_H$ to $E840\ FFFF_H$
100_B	8 sub-ranges of 8 Kbytes	$XX000_B$ $XX001_B$ … $XX111_B$	$E840\ 0000_H$ to $E840\ 1FFF_H$ $E840\ 2000_H$ to $E840\ 3FFF_H$ … $E840\ E000_H$ to $E840\ FFFF_H$

Table 21-17 DMI RAM Address Protection Sub-Range Definitions (cont'd)

SIZE2	Sub-Ranges	SLICE2	Selected Address Range
101_B	4 sub-ranges of 16 Kbytes	$XXX00_B$ $XXX01_B$ $XXX10_B$ $XXX11_B$	E840 0000_H to E840 $3FFF_H$ E840 4000_H to E840 $7FFF_H$ E840 8000_H to E840 $BFFF_H$ E840 $C000_H$ to E840 $FFFF_H$
110_B	2 sub-ranges of 32 Kbytes	$XXXX0_B$ $XXXX1_B$	E840 0000_H to E840 $7FFF_H$ E840 8000_H to E840 $FFFF_H$
111_B	64 Kbytes	$XXXXX_B$	E840 0000_H to E840 $FFFF_H$

21.5.7 MLI0/MLI1 Transfer Window Address Maps

In the TC1766, the transfer windows for the MLI0 and MLI1 modules are located in the address ranges as identified in **Table 21-18**.

Table 21-18 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$_H$

22 General Purpose Timer Array (GPTA)

This chapter describes the General Purpose Timer Array of the TC1766. This chapter contains the following sections:

- A summary on the structure and basic functionalities (see **Page 22-1**)
- Functional description of the GPTA kernel (see **Page 22-5**)
- Register descriptions of all GPTA kernel specific registers (see **Page 22-150**)
- TC1766 implementation-specific details and registers of the GPTA module including port connections and control, interrupt control, address decoding, and clock control, (see **Page 22-212**).

*Note: The GPTA kernel register names described in **Section 22.3** are referenced in the TC1766 User's Manual by the module name prefix "GPTA0_" for the GPTA0 module.*

22.1 GPTA Overview

The TC1766 contains one General Purpose Timer Array (GPTA0). **Figure 22-1** shows a global view of the GPTA module.

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, electrical motor control applications, but can also be used to generate simple and complex signal waveforms needed in other industrial applications.

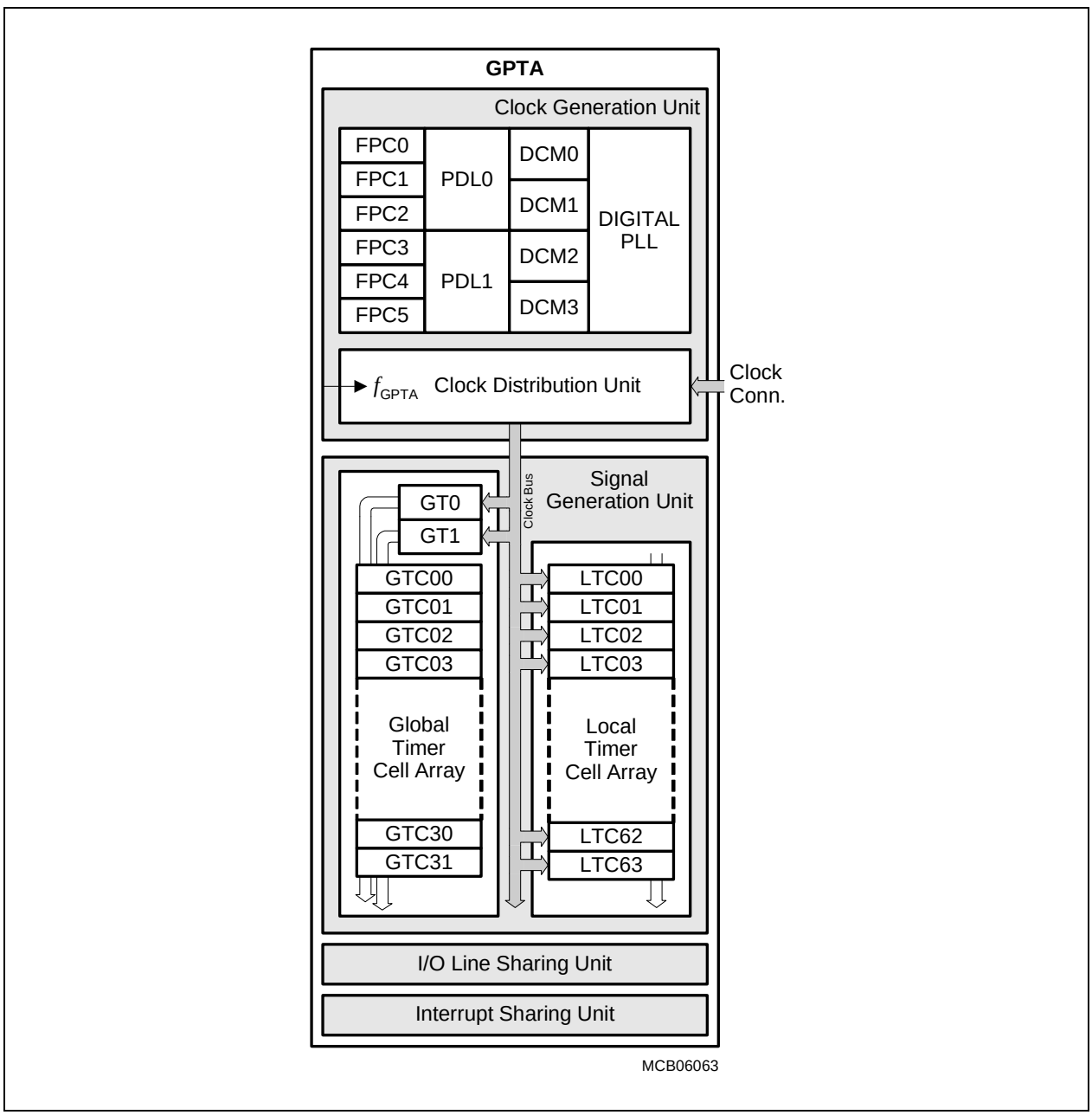


Figure 22-1 General Block Diagram of the GPTA Module in the TC1766

22.1.1 Functionality of GPTA0

The General Purpose Timer Array GPTA0 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.

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 unit.

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 Control Unit

- 111 interrupt sources, generating up to 38 service requests

I/O Sharing Unit

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

22.2 GPTA Kernel Description

The functionality of the General Purpose Timer Array GPTA0 kernel is described in this section. Clock control, address decoding, and service (interrupt) request control are managed outside the GPTA0 module kernel.

Figure 22-2 shows a global block diagram of the GPTA module kernel.

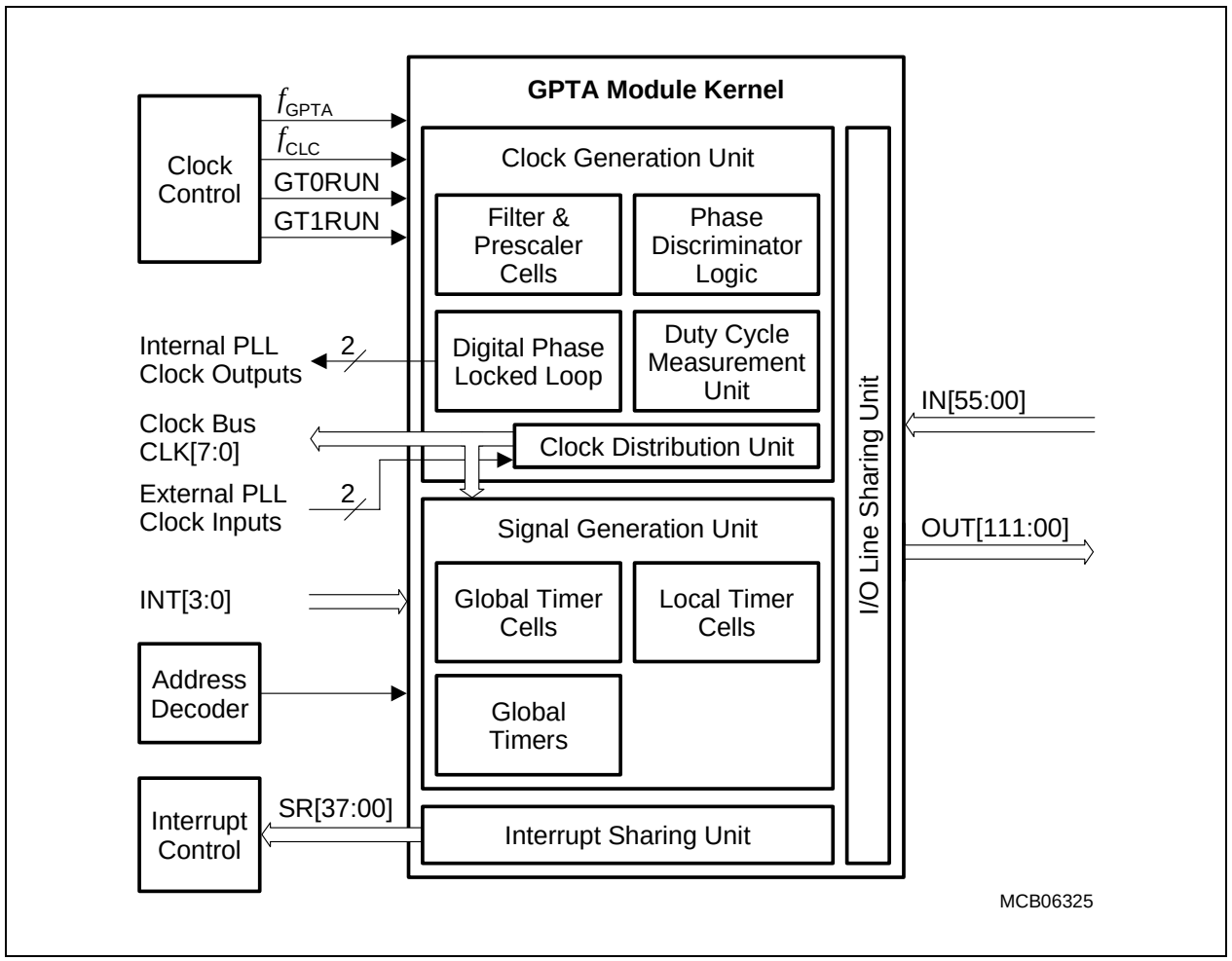


Figure 22-2 Block Diagram of GPTA Kernel

The GPTA0 kernel has 56 input signals, 112 output signals, and four input signals, that can be connected to port pins or other on-chip logic units (see **"GPTA Module Implementation" on Page 22-212** for the TC1766 specific interconnections). Further, several clock input and output signals are provided.

22.2.1 GPTA Units

The General Purpose Timer Array GPTA0 (**Figure 22-2**) is split into a Clock Generation Unit (CGU) and a Signal Generation Unit (SGU):

- The **Clock Generation Unit** (see **Page 22-7**) allows a preprocessing of the input signals using filter, timer, capture, compare and enhanced digital PLL Modules:
  - The **Filter and Prescaler Cells** (FPC) provide input noise filtering (Immediate Debounce and Delayed Debounce) and may also work as prescalers for the GPTA 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) is intended to generate a higher resolution clock from 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 Unit** (CDU) provides all local and global timer cells with a variety of different clock signals. It is equipped with GPTA module clock prescalers and multiplexers supporting alternate clock sources.

The original signals and all outputs of the preprocessing units are distributed to the Global Timers and LTCs via the clock bus.

- The **Signal Generation Unit** (see **Page 22-36**) provides a set of timers, capture and compare units:
  - 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 units, 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 units 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 resettable counter, capture or compare unit. 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.

22.2.2 Clock Generation Unit

As described in detail in the following sections, the Clock Generation Unit (CGU) provides five signal pre-processing modules:

- Filter and Prescaler Cell (FPC)
- Phase Discrimination Logic (PDL)
- Duty Cycle Measurement Unit (DCM)
- Digital Phase Locked Loop Cell (PLL)
- Clock Distribution Module (CDU)

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 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 Units** (DCM) provide signal measurement capabilities (timer plus capture register, single and double capture on rising or 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 Unit** (CDU) provides all Local and Global Timer Cells with a variety of different clock signals. It is equipped with GPTA module clock prescalers and multiplexers supporting alternate clock sources.

Figure 22-3 shows how the five modules of the CGU are interconnected. The external interface signals of the CGU are:

- GPTA module clock f_{GPTA}
- GPTA module input signals (connected to the FPCs)
- Clock bus outputs (generated by the CDU)
- PDL bus outputs
- External PLL clock inputs (fed into CDU)

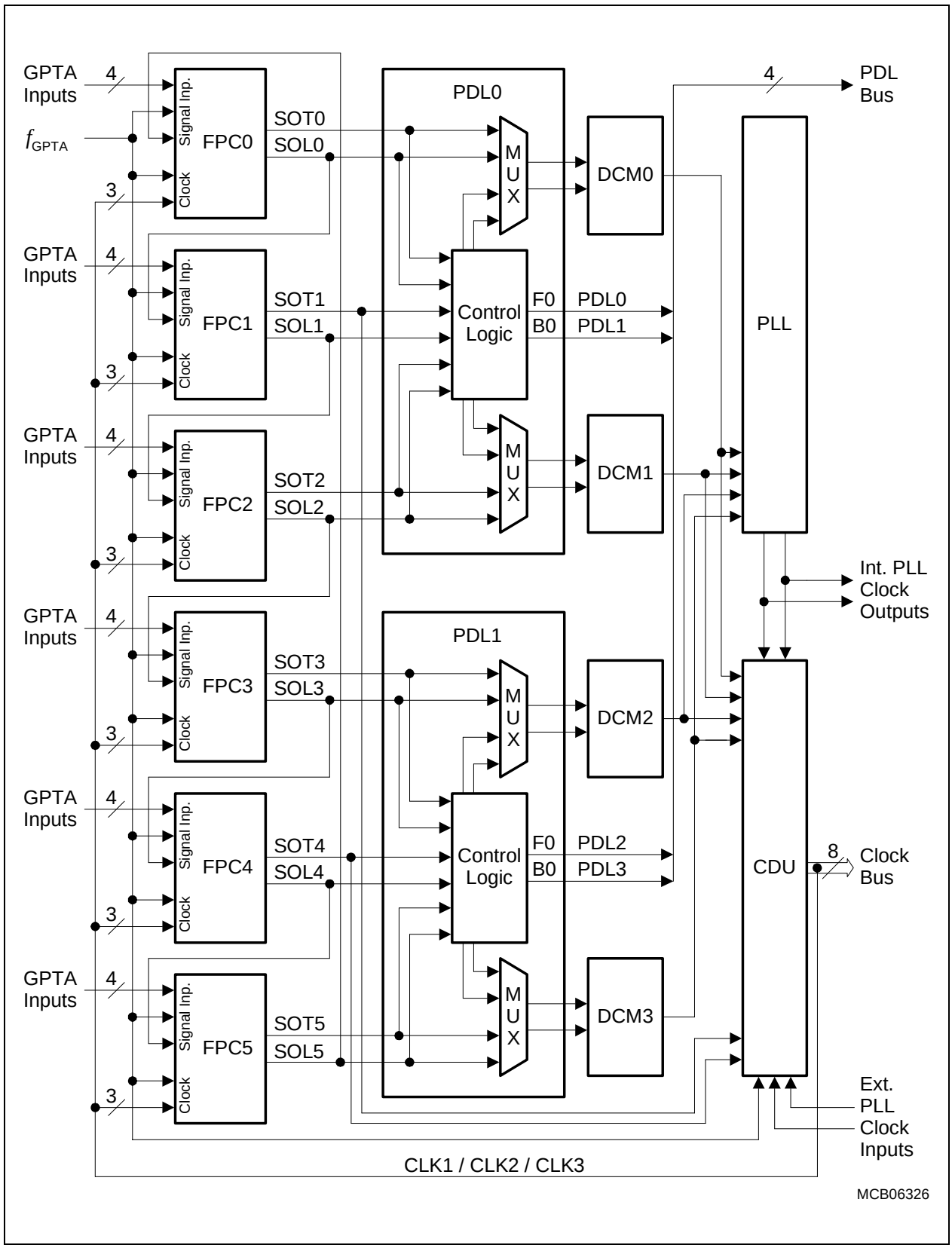


Figure 22-3 Interconnections in the CGU Modules

22.2.2.1 Filter and Prescaler Cell (FPC)

The GPTA 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 unit, a 16-bit timer, a 16-bit compare register, a 16-bit comparator, and an FPC control unit (see also **Page 22-115** for the FPC functional algorithm description).

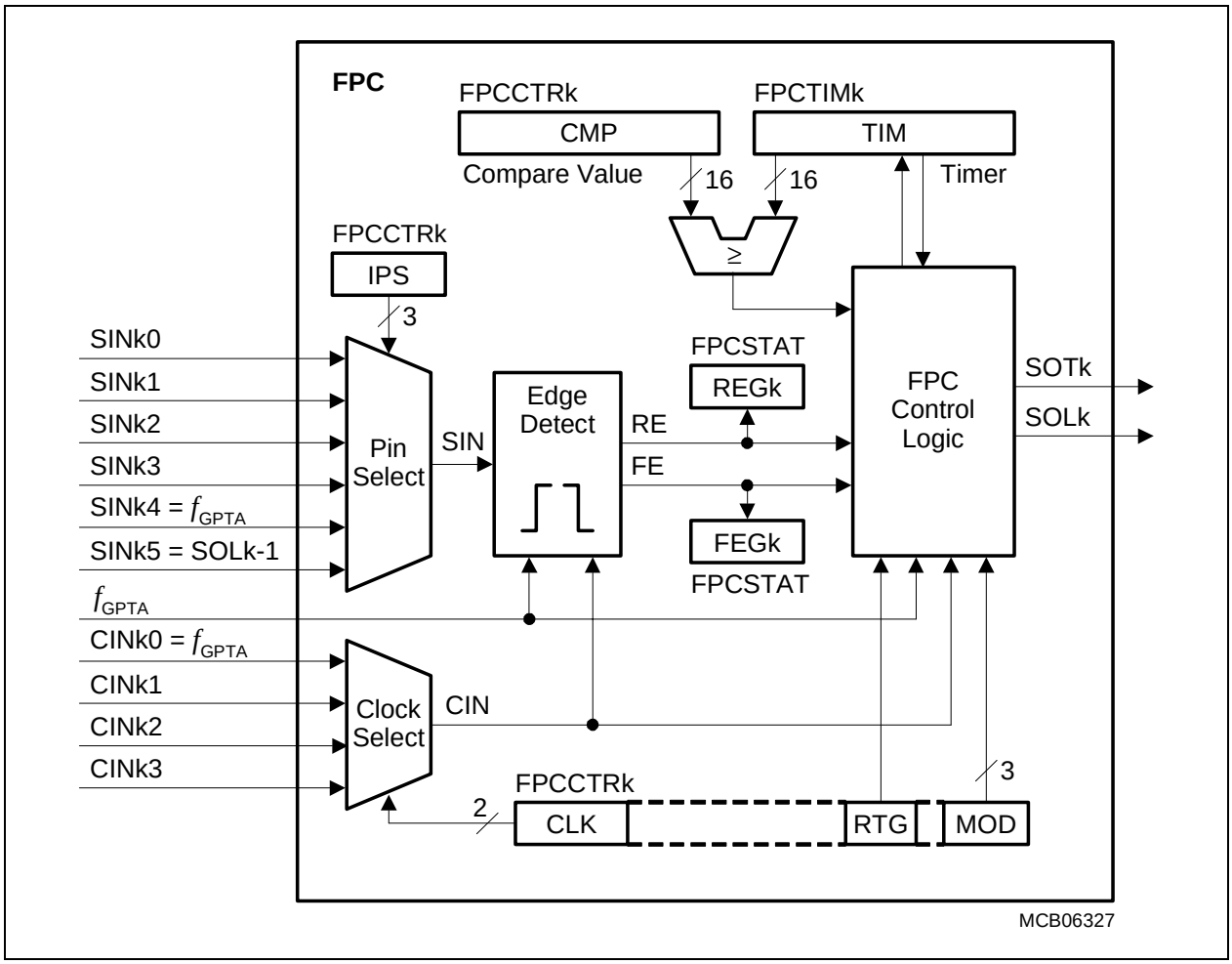


Figure 22-4 Filter and Prescaler Cell Architecture

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 22-155**)
- FPCCTRk = Filter and Prescaler Cell Control Register k (see **Page 22-156**)
- FPCTIMk = Filter and Prescaler Cell Timer Register k (see **Page 22-158**)

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 on rising edge)
- Prescaler Mode (triggered on falling edge)

The operation mode is selected by bit field FPCCTRk.MOD (**Page 22-156**).

FPC Input Signals

Bit field FPCCTRk.IPS (see **Page 22-156**) 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 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 I/O line and FPC signal inputs SINk is defined in **“FPC Input Line Selection” on Page 22-94**.

FPC Filter Clocks

Bit field FPCCTRk.CLK (see **Page 22-157**) selects one of four filter clocks for FPCk:

- Clock input line 0 (CINk0) = GPTA module clock f_{GPTA}
- Clock input line 1 (CINk1) = local PLL clock
- Clock input line 2 (CINk2) = (prescaled) GPTA 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. The uncompensated PLL clock is useful in applications in which bursts due to acceleration might disturb the filter function.

With a prescaled GPTA module clock, much longer filter periods can be achieved.

Output Signal Splitting

Two output lines are provided by each FPC module 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 unit implemented at the input of each subsequent cell. **Figure 22-5** shows the FPC output signal splitting scheme.

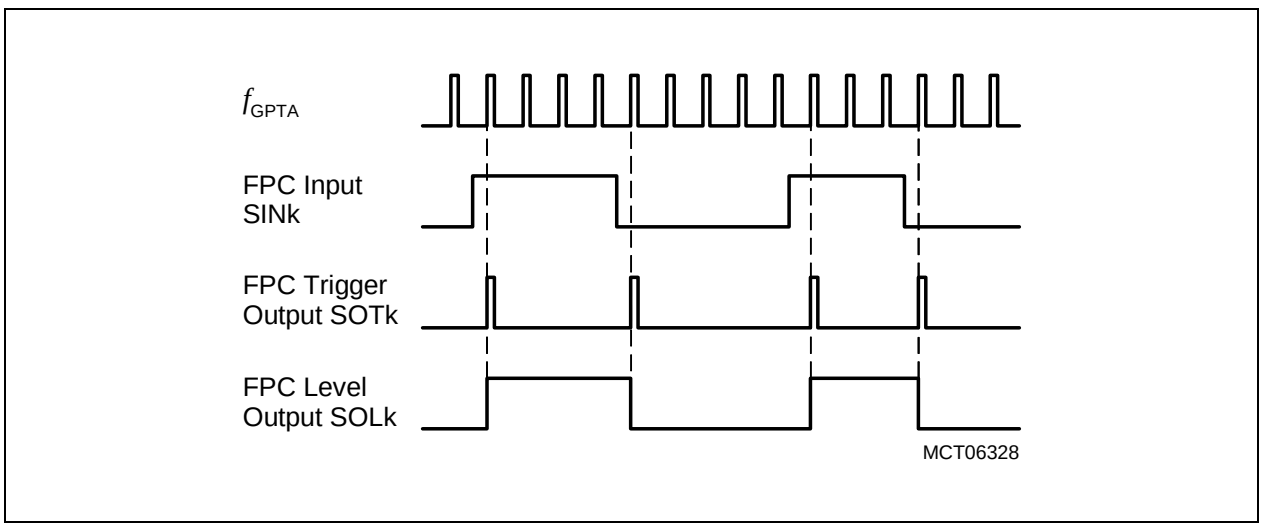


Figure 22-5 FPC Output Splitting into Trigger and Level Information

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 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 cleared 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.

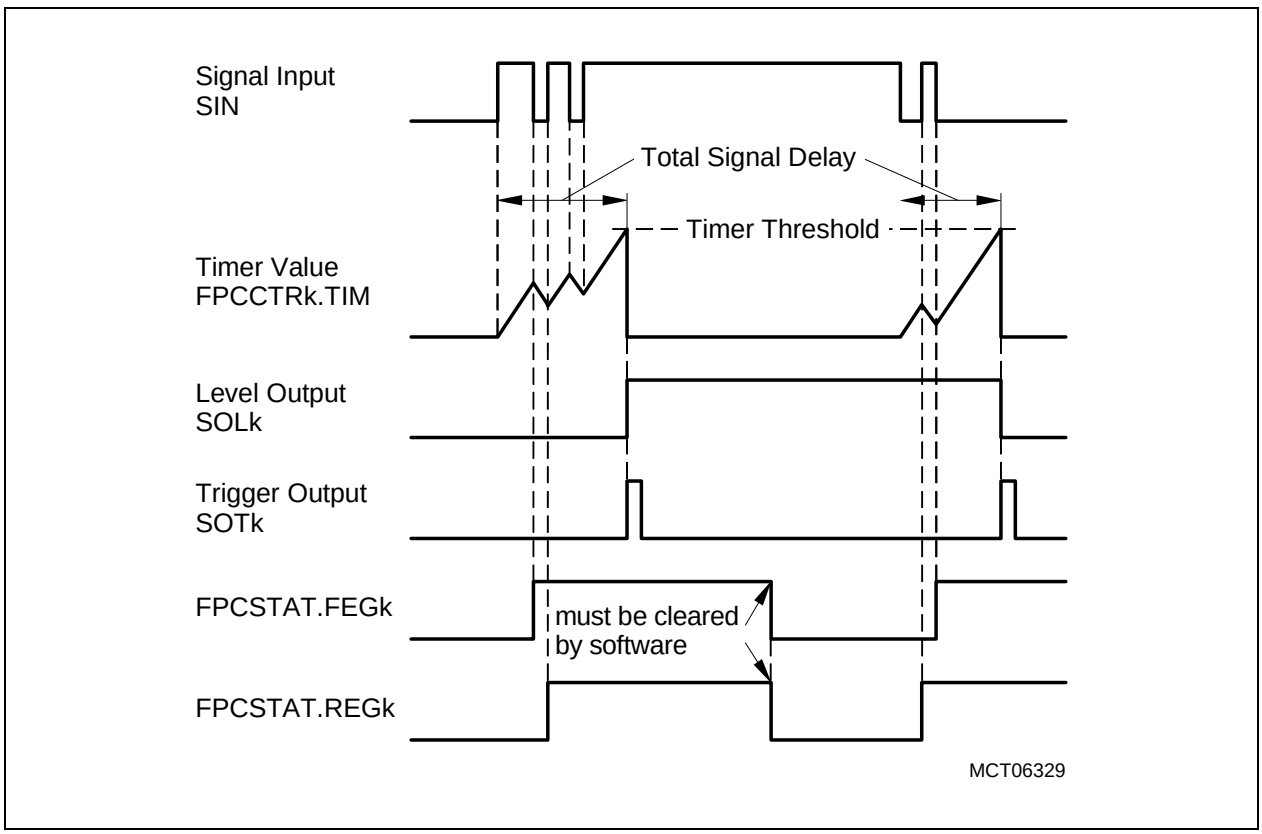


Figure 22-6 FPC Delayed Debounce Filter Algorithm with Timer Decrement

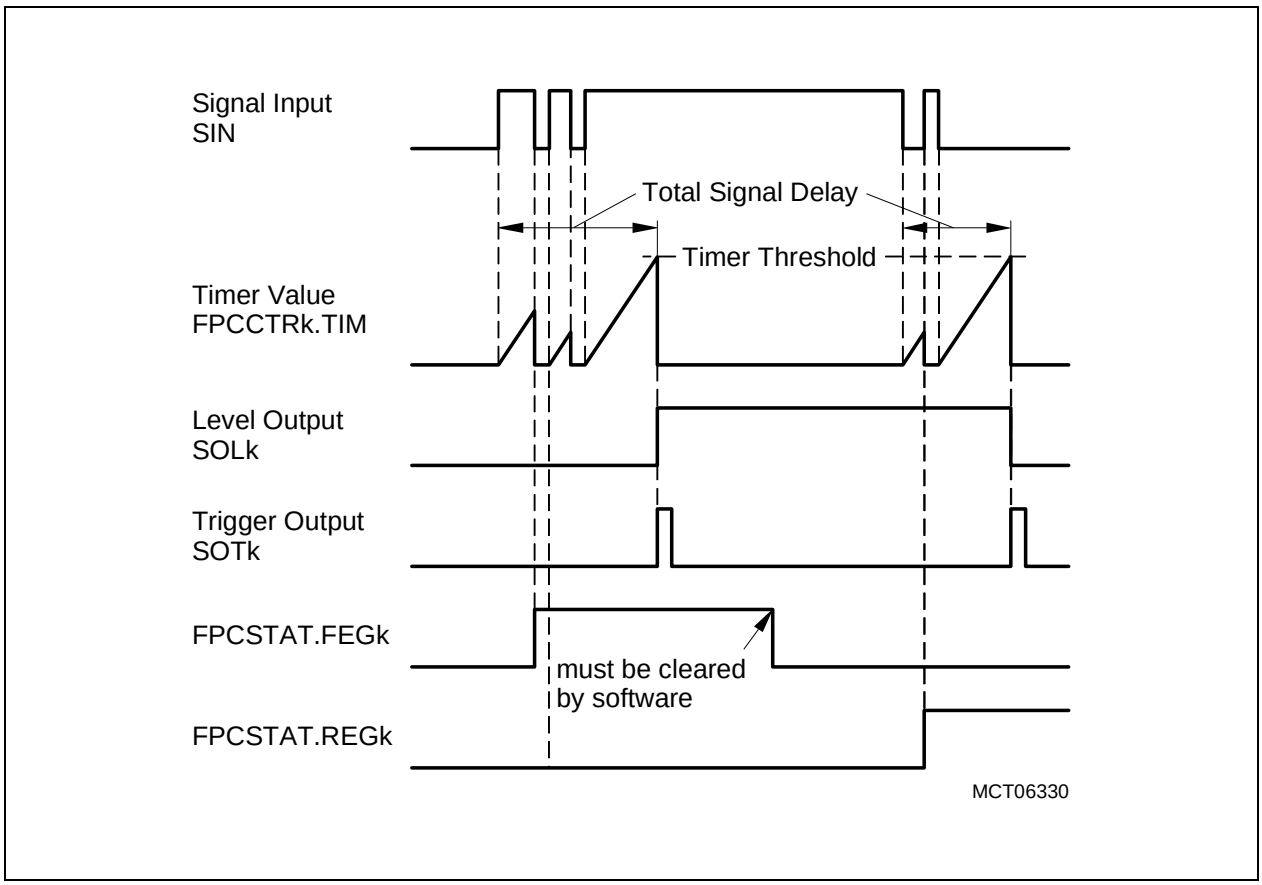


Figure 22-7 FPC Delayed Debounce Filter Algorithm with Timer Reset

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 glitch edge detection is also performed with f_{GPTA}. The further analysis (e.g. filter timer increment) is done at the selected filter clock rate of CIN. As long as the timer is reset, the FPC control unit 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 cleared by software.

The filter is bypassed 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.

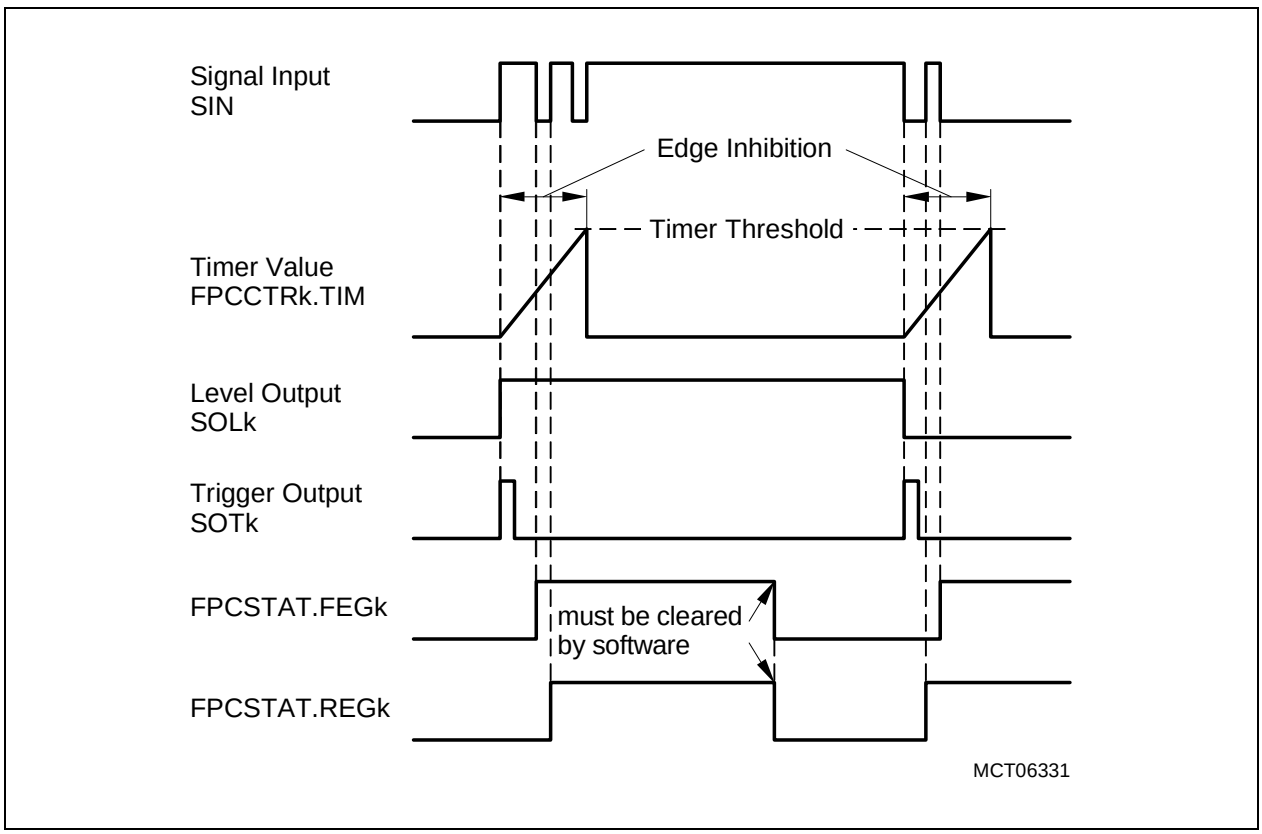


Figure 22-8 FPC Immediate Debounce Filter Algorithm on Both Edges

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 (**Figure 22-9**).

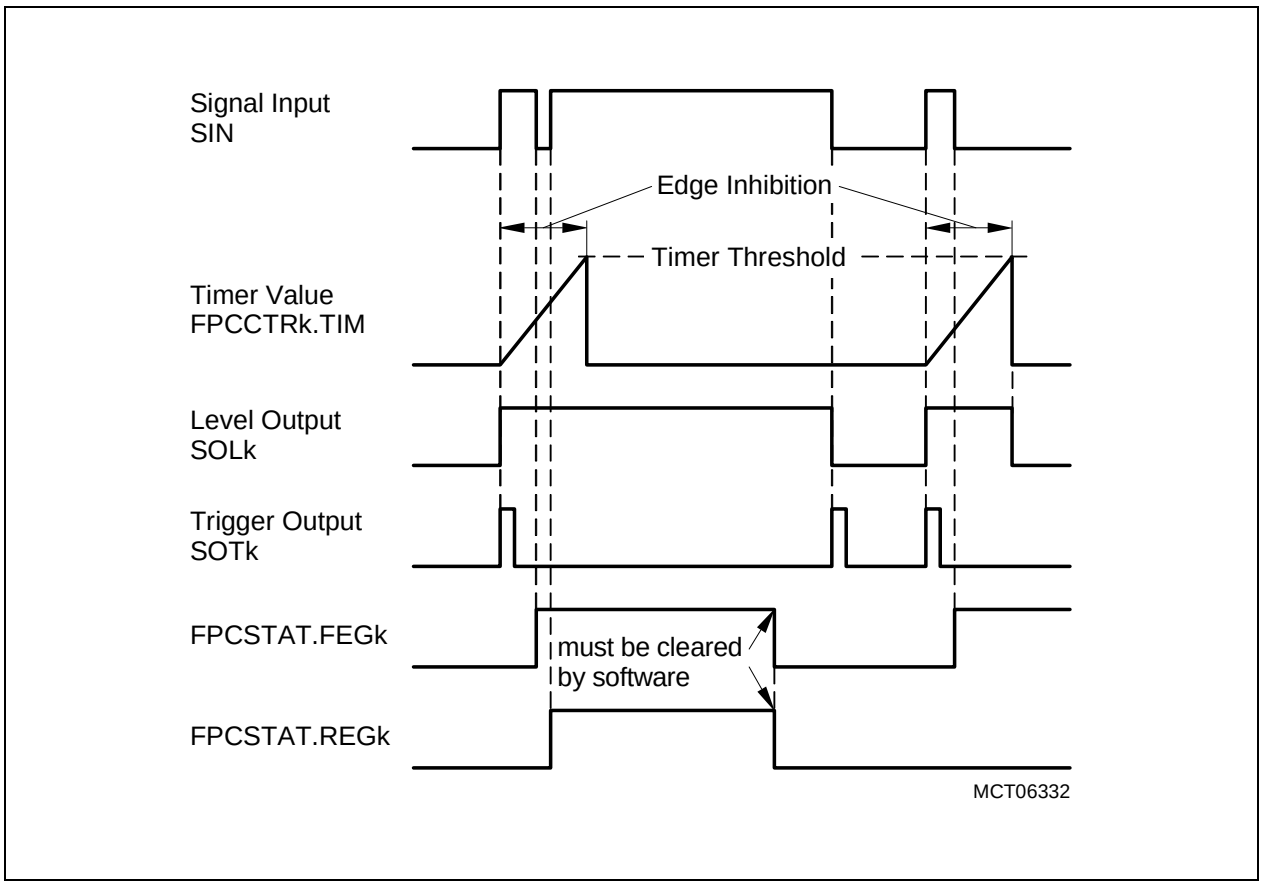


Figure 22-9 FPC Immediate Debounce Filter Algorithm on Rising Edge only

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).

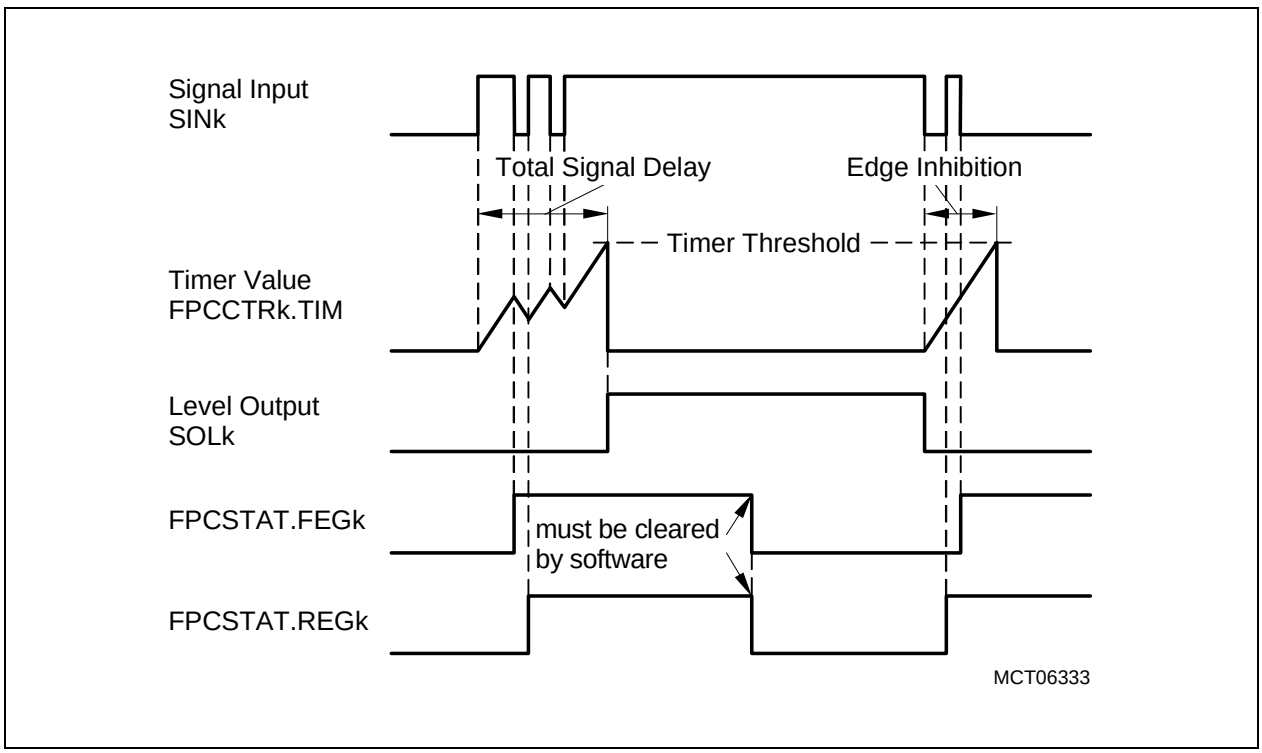


Figure 22-10 FPC Mixed Filter Algorithm

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 unit counts each rising (or falling) edge of the input signal. When the timer value matches the compare value:

- one GPTA 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

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 on rising edge
- FPCCTRk.MOD = 111_B: Prescaler Mode triggered on falling edge

22.2.2.2 Phase Discrimination Logic (PDL)

The GPTA provides two Phase Discrimination Logic modules (PDL0, PDL1) driven by two signal lines coming from an FPC cell (for description, see **Page 22-11**):

- An event input signal and
- A level input signal

Both Phase Discrimination Logic modules are controlled by the Phase Discrimination Logic Control Register PDLCTR (see **Page 22-159**).

Each PDL is equipped with an edge detection unit, a phase detection unit, a PDL control unit, and an output multiplexer. Six output lines are provided by each PDL Module:

- 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 two f_{GPTA} cycles before it changes (three f_{GPTA} cycles for a 3-sensor positioning system).

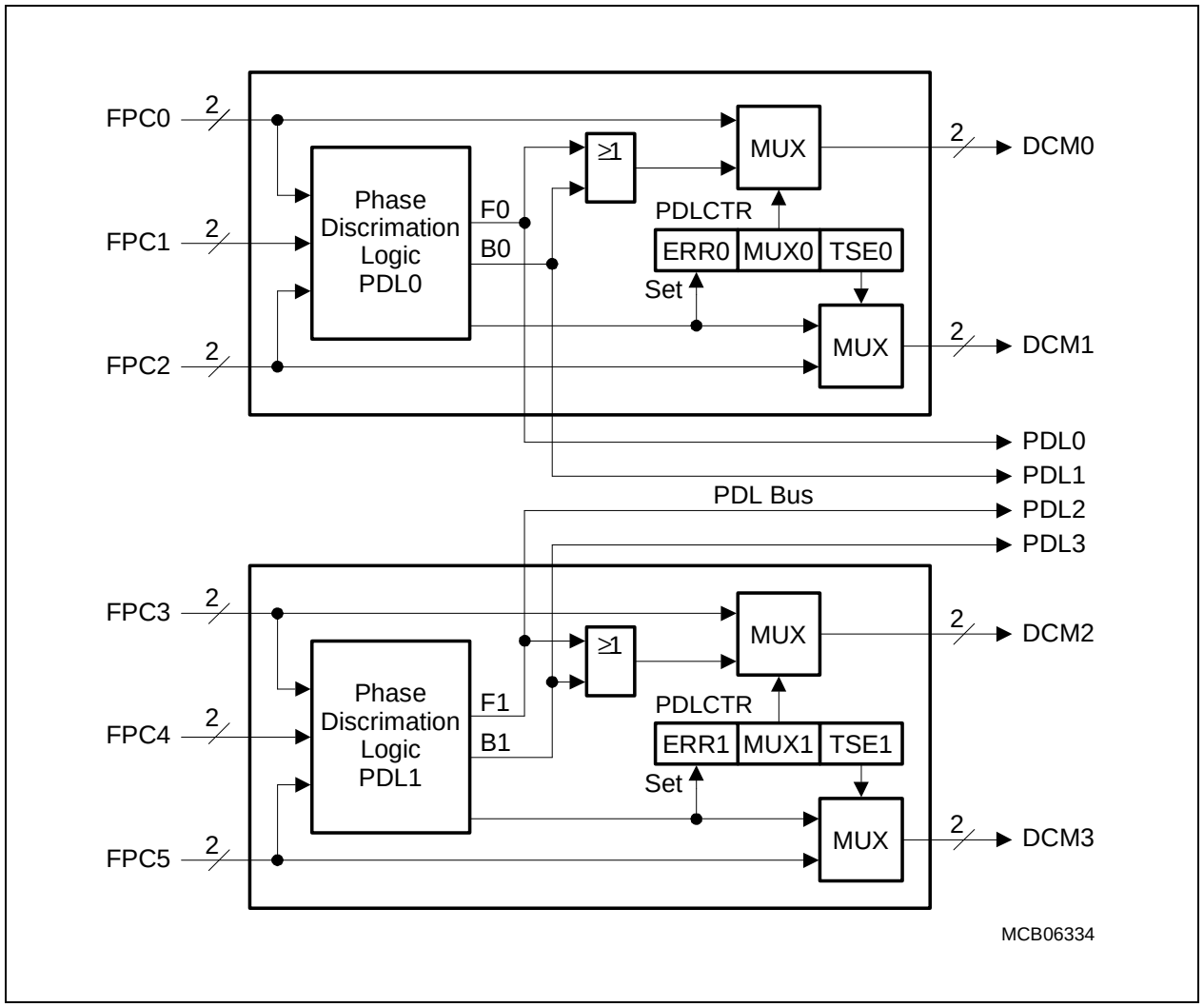


Figure 22-11 Block Diagram of Phase Discrimination Logic Unit

Positioning System With Two Sensors

The 2-sensor Mode is enabled when bit PDLCTR.TSEx is cleared. The sensors are mounted at a 90° angle to each other (see **Figure 22-12**). The third sensor input of the PDL module is internally disabled and DCM1/DCM3 cell inputs are driven by fed-through FRC2/FRC5 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

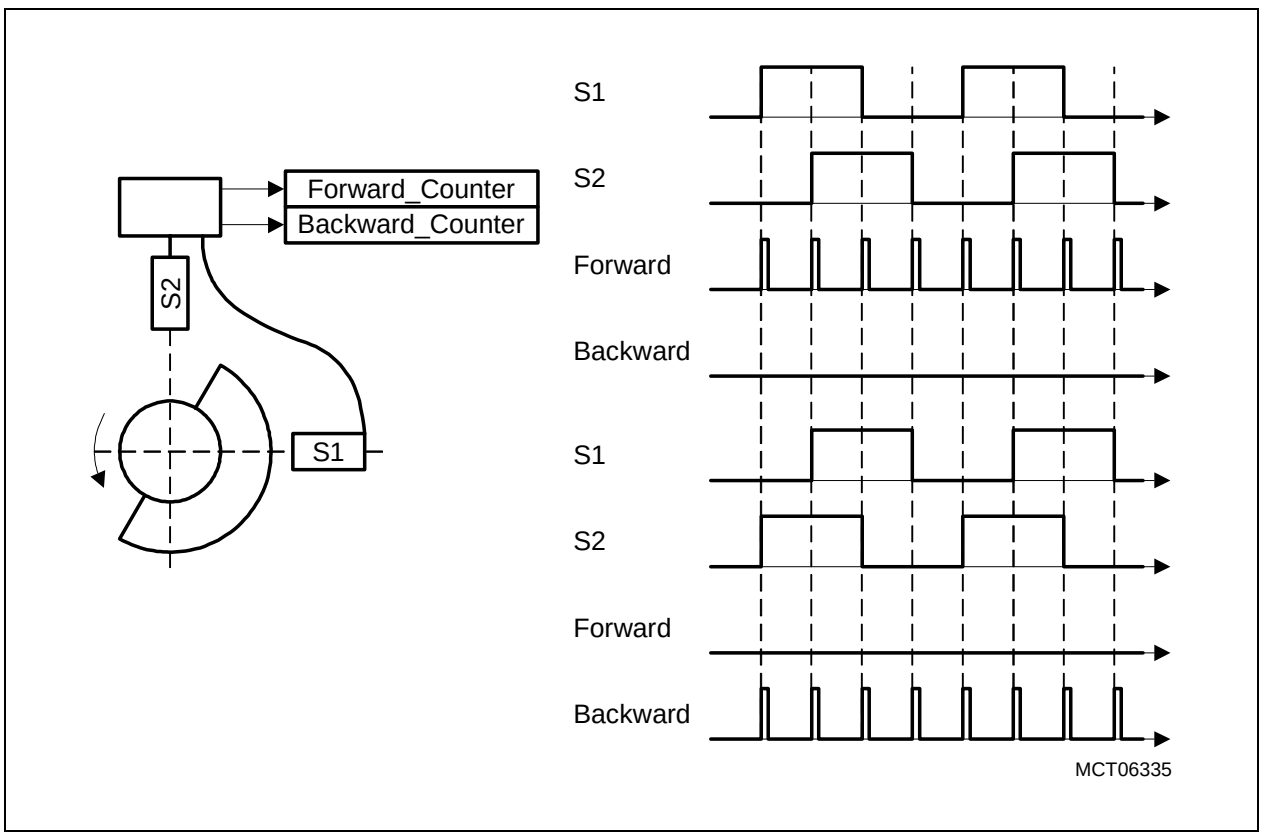


Figure 22-12 Interface Signals of a PDL in a 2-Sensor Positioning System

Figure 22-13 illustrates how the output signals of a 2-sensor system superimposed with noise are processed by the PDL unit. Jitter pulses are completely compensated if they do not occur on both signal lines simultaneously.

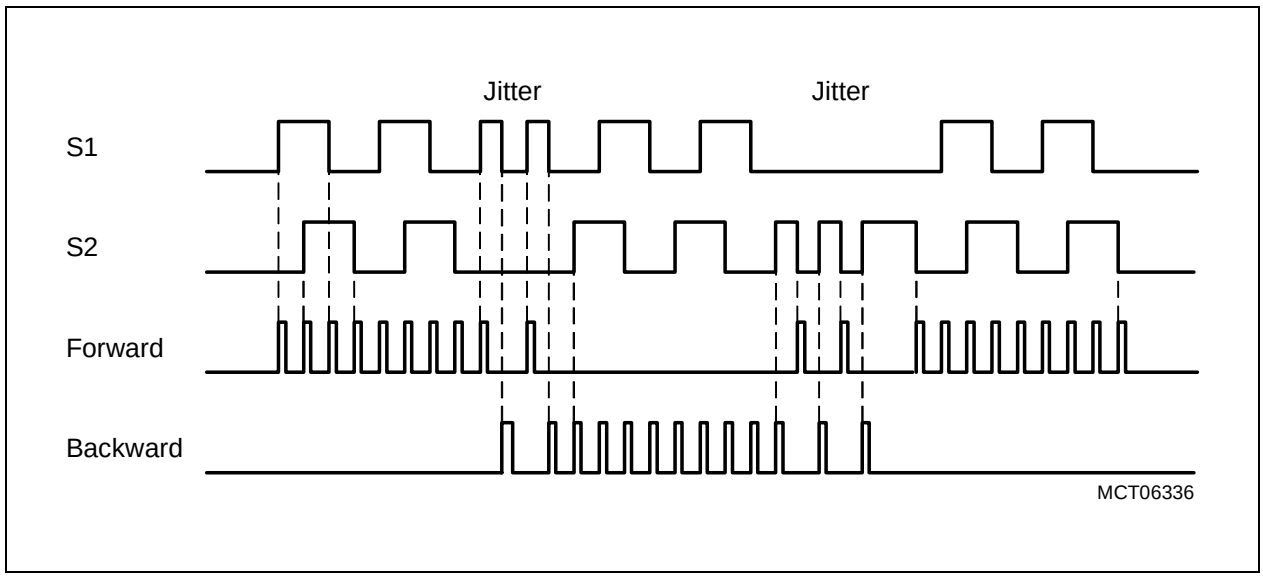


Figure 22-13 Compensation of Input Jitter

Positioning System with Three Sensors

The 3-sensor Mode is enabled when bit PDLCTR.TSEx is set to 1. The sensors are mounted at a 120° angle to each other (see Figure 22-14). 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 cleared 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

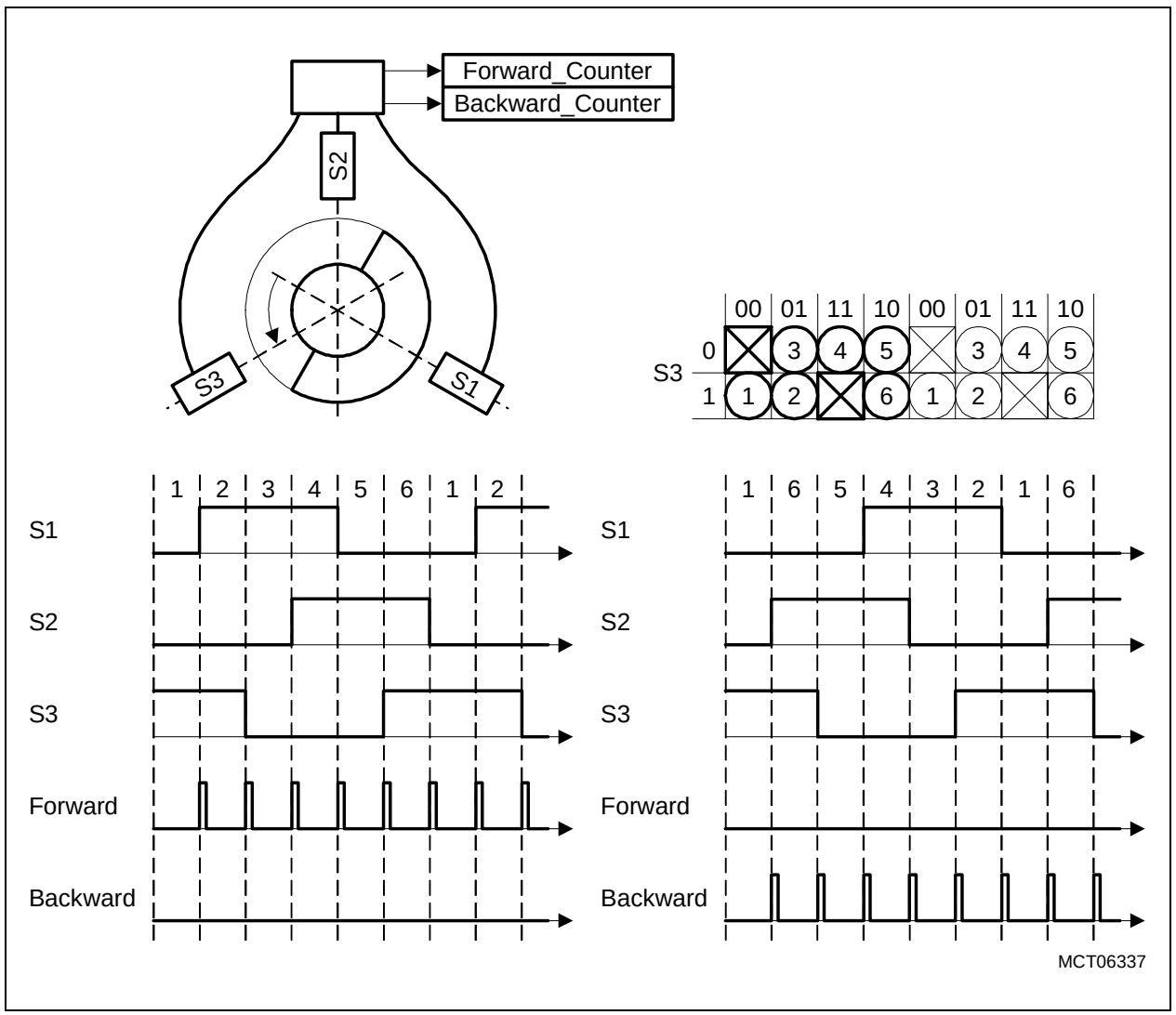


Figure 22-14 Interface Signals of a PDL in a 3-Sensor Positioning System

Jitter pulses are completely compensated as illustrated in **Figure 22-13**.

22.2.2.3 Duty Cycle Measurement Unit (DCM)

The GPTA contains four DCM units (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 built by:

- An event input, and
- A signal level input.

Each DCM unit 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 unit 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 unit (**Figure 22-15**).

The following registers are assigned to the DCM units:

- DCMCTRk = Duty Cycle Measurement Control Register k (see **Page 22-161**)
- DCMTIMk = Duty Cycle Measurement Timer Register k (see **Page 22-162**)
- DCMCAVk = Duty Cycle Measurement Capture Register k (see **Page 22-163**)
- DCMCOVk = Duty Cycle Measurement Capture/Compare Register k (also referred as “capcom”, see **Page 22-163**)
- SRSC0 = Service Request State Clear Register 0 (see **Page 22-203**)
- SRSS0 = Service Request State Set Register 0 (see **Page 22-207**)

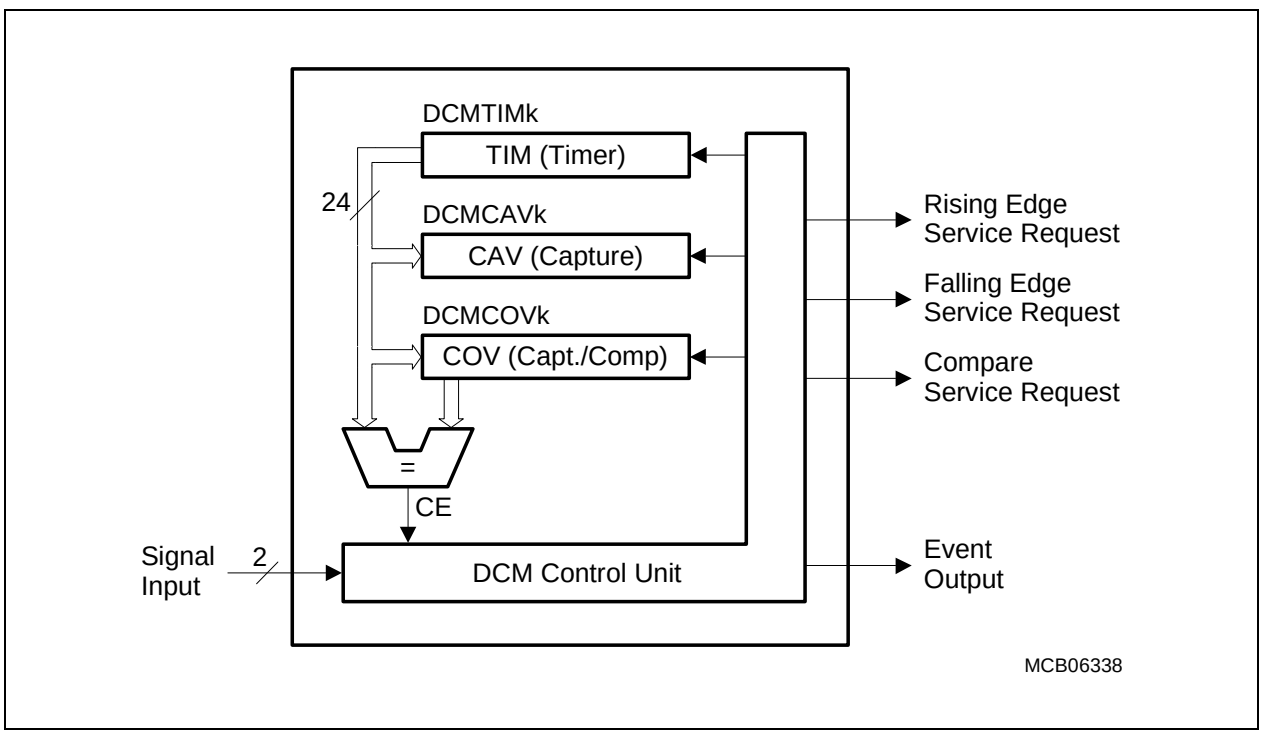


Figure 22-15 Block Diagram of a Duty Cycle Measurement Unit

The DCM unit inputs are connected to the PDL outputs. Depending on the configuration of the associated PDL cell, the DCM units can also be driven by an FPC directly (as shown in **Figure 22-11**):

- 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 unit 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 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 unit (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 unit is able to generate three service request output signals. The service request outputs of a DCMk unit are controlled as shown in **Figure 22-16**. 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 Unit (IS)” on Page 22-112**.

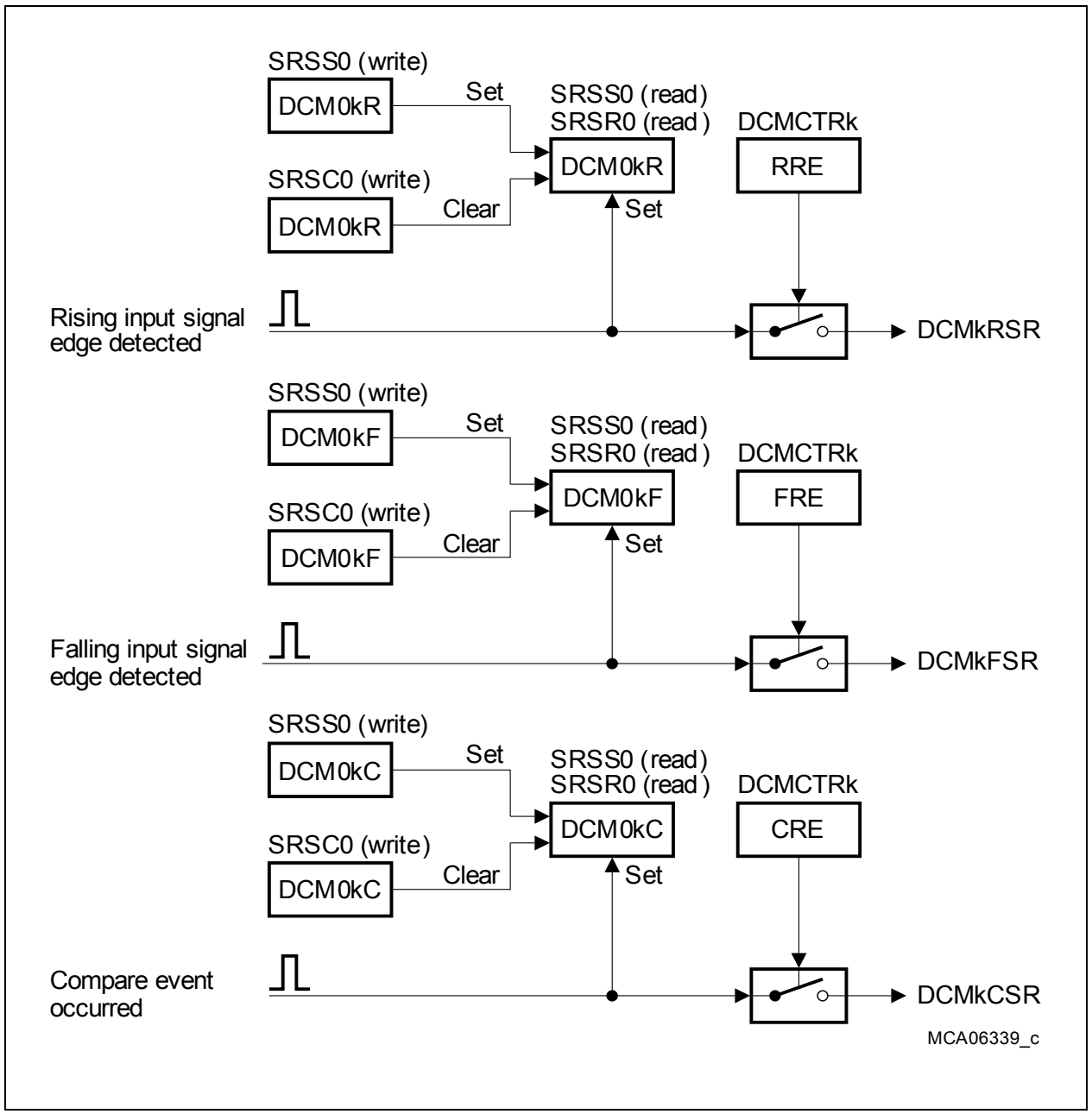


Figure 22-16 DCMk Service Request Generation

22.2.2.4 Digital Phase Locked Loop Cell (PLL)

The GPTA 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 module clocks f_{GPTA} on the output signal line. The four signal output lines of the DCM units can be used as PLL trigger input. The PLL control unit distributes the desired number of GPTA 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 unit (see **Figure 22-17**).

The following registers are assigned to the Phase Locked Loop cell:

- PLLCTR = Phase Locked Loop Control Register (see **Page 22-164**)
- PLLMTI = Phase Locked Loop Microtick Register (see **Page 22-165**)
- PLLCNT = Phase Locked Loop Counter Register (see **Page 22-166**)
- PLLSTP = Phase Locked Loop Step Register (see **Page 22-166**)
- PLLREV = Phase Locked Loop Reload Register (see **Page 22-167**)
- PLLDTR = Phase Locked Loop Delta Register (see **Page 22-167**)
- SRSC0 = Service Request State Clear Register 0 (see **Page 22-203**)
- SRSS0 = Service Request State Set Register 0 (see **Page 22-207**)

Three output signals are available on the PLL cell:

- PLL signal output line
- Uncompensated PLL signal output line
- Service request line

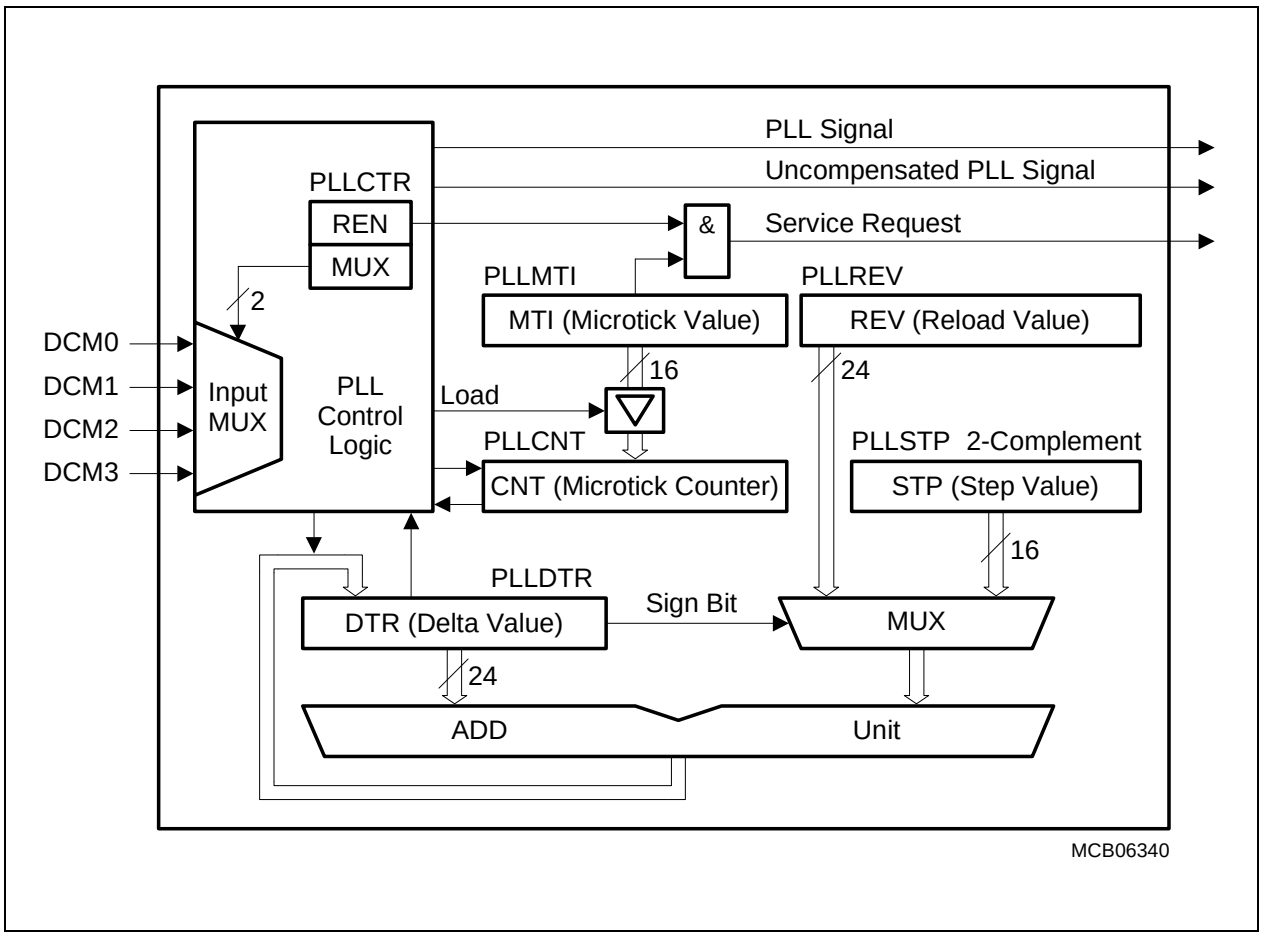


Figure 22-17 Block Diagram of Digital PLL Cell

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 the 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.

PLL Interrupt Control

The PLL unit is able to generate a service request output signal PLLSR. This signal is controlled as shown in **Figure 22-18**. 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. Further details on service request and interrupt handling are given in section **“Interrupt Sharing Unit (IS)” on Page 22-112**.

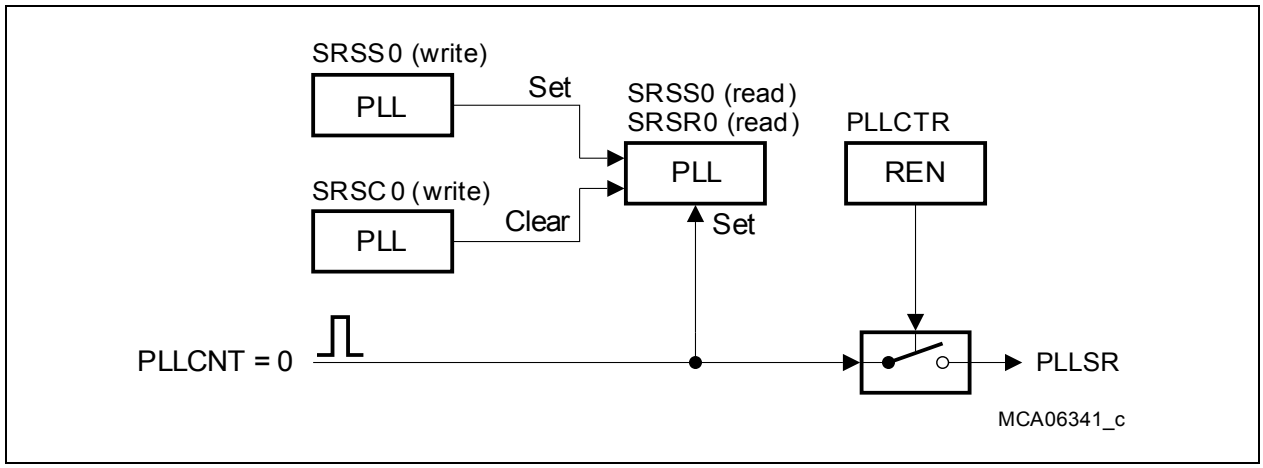


Figure 22-18 PLL Service Request Generation

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 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 22-19**). 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).

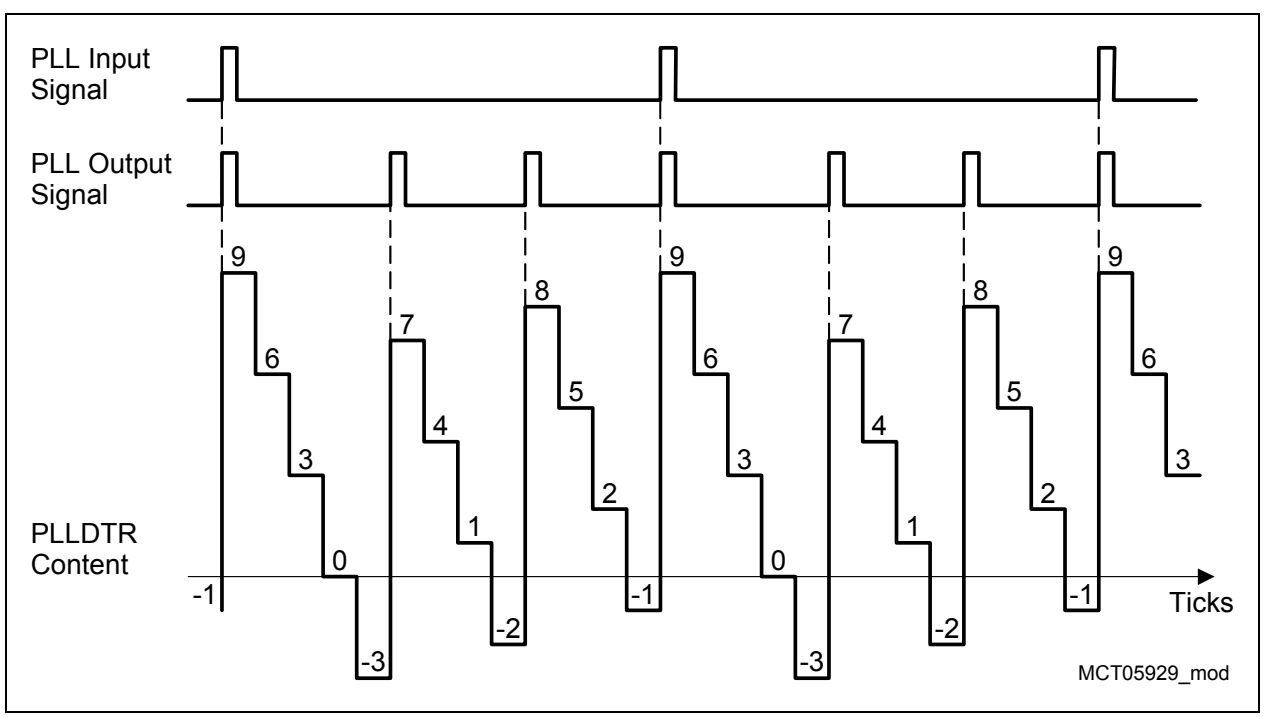


Figure 22-19 Digital PLL Steady State Simulation

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 22-20 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.

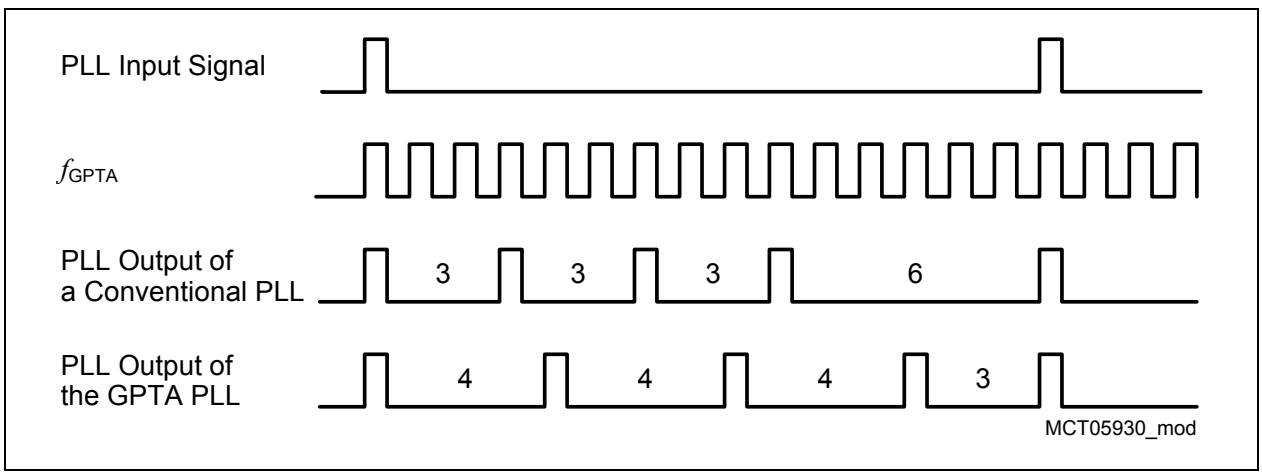


Figure 22-20 Advantage of the GPTA PLL

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 22-21**).
  - 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.

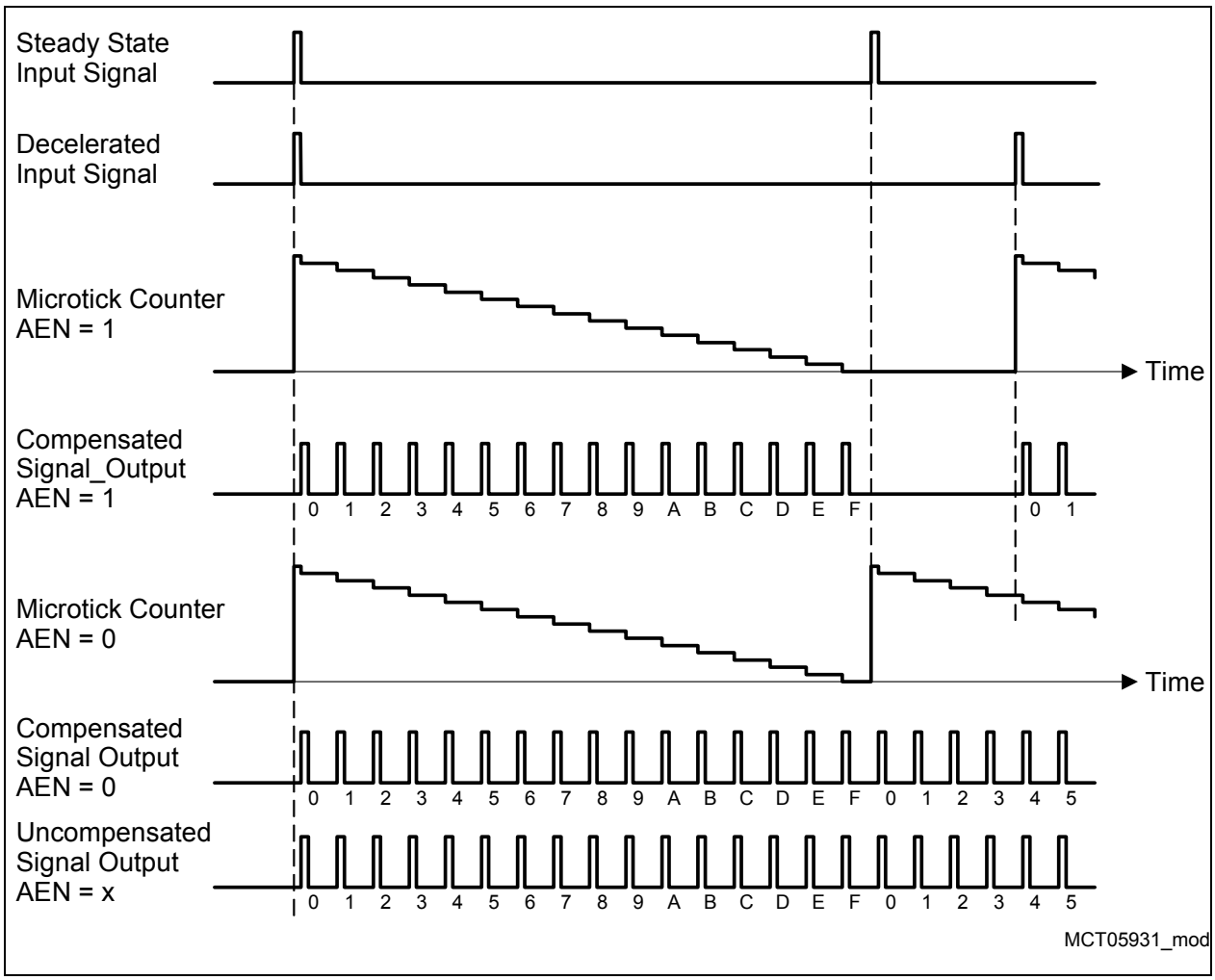


Figure 22-21 Compensation of Input Signal Deceleration

- 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. Subsequently, counter and Delta register are reloaded with their calculated values and the PLL operates at normal speed (see **Figure 22-22**).

– 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.

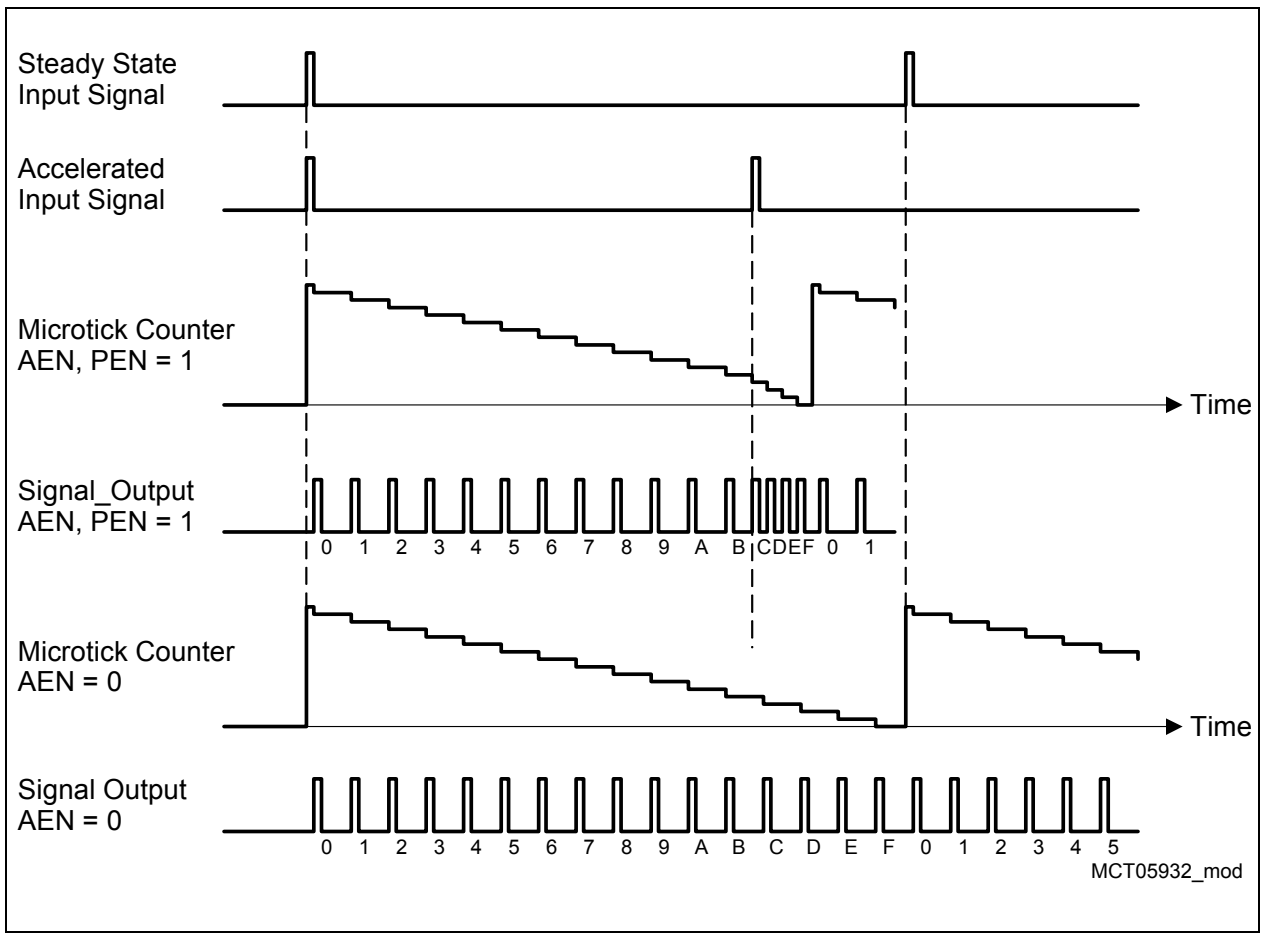


Figure 22-22 Compensation of Input Signal Acceleration

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.

22.2.2.5 Clock Distribution Unit (CDU)

The Clock Distribution Unit (CDU) 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 22-23**).

The prescalers divide the GPTA 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 CDU to each LTC except LTC63.

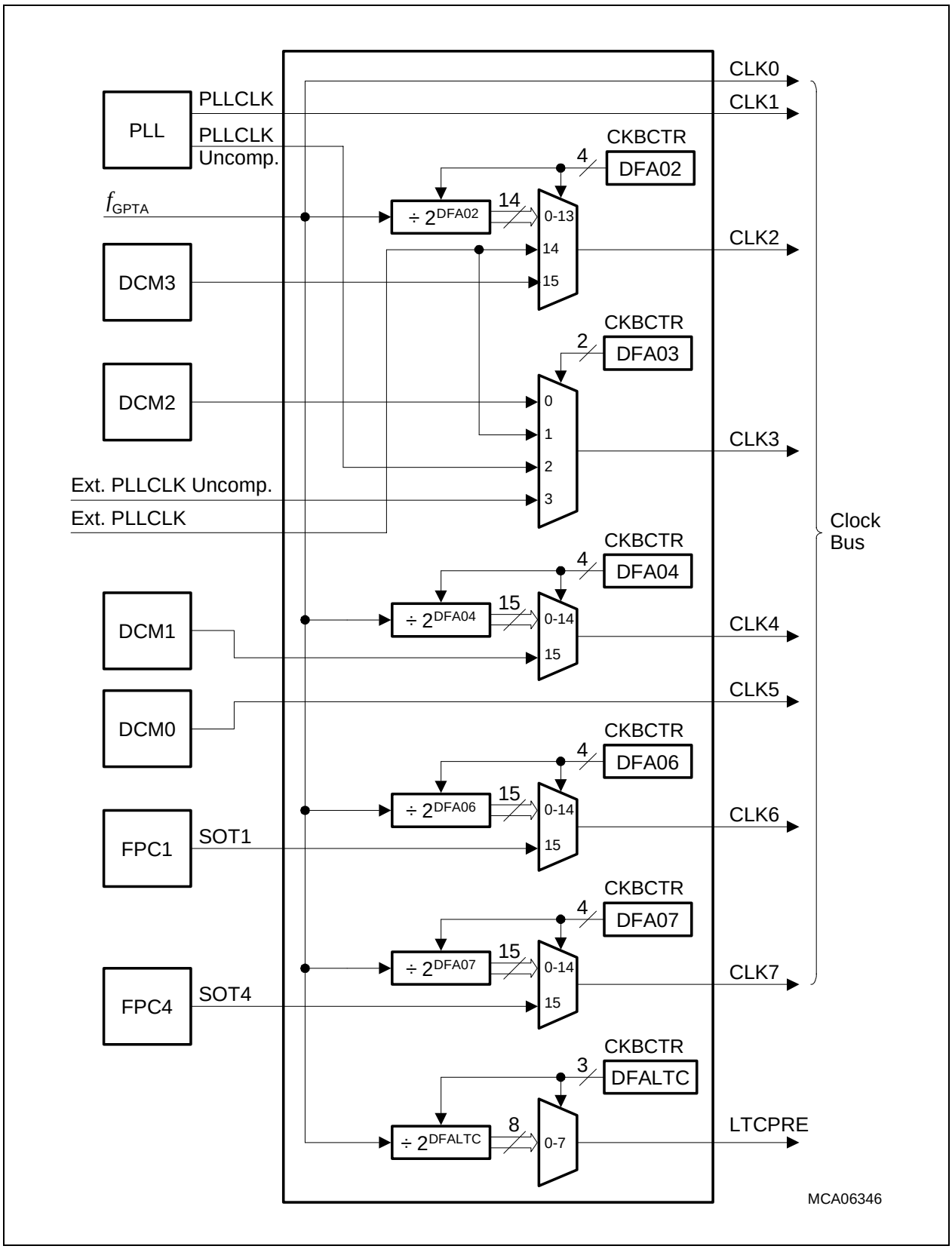


Figure 22-23 Clock Distribution Unit Block Diagram

22.2.3 Signal Generation Unit

As described in detail in the following sections, the Signal Generation Unit contains three types of modules.

- Global Timer (GT)
- Global Timer Cell (GTC)
- Local Timer Cell (LTC).

22.2.3.1 Global Timers (GT)

The GPTA 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 22-24**).

Note: Index variable k (= 0, 1) determines the number of the global timer.

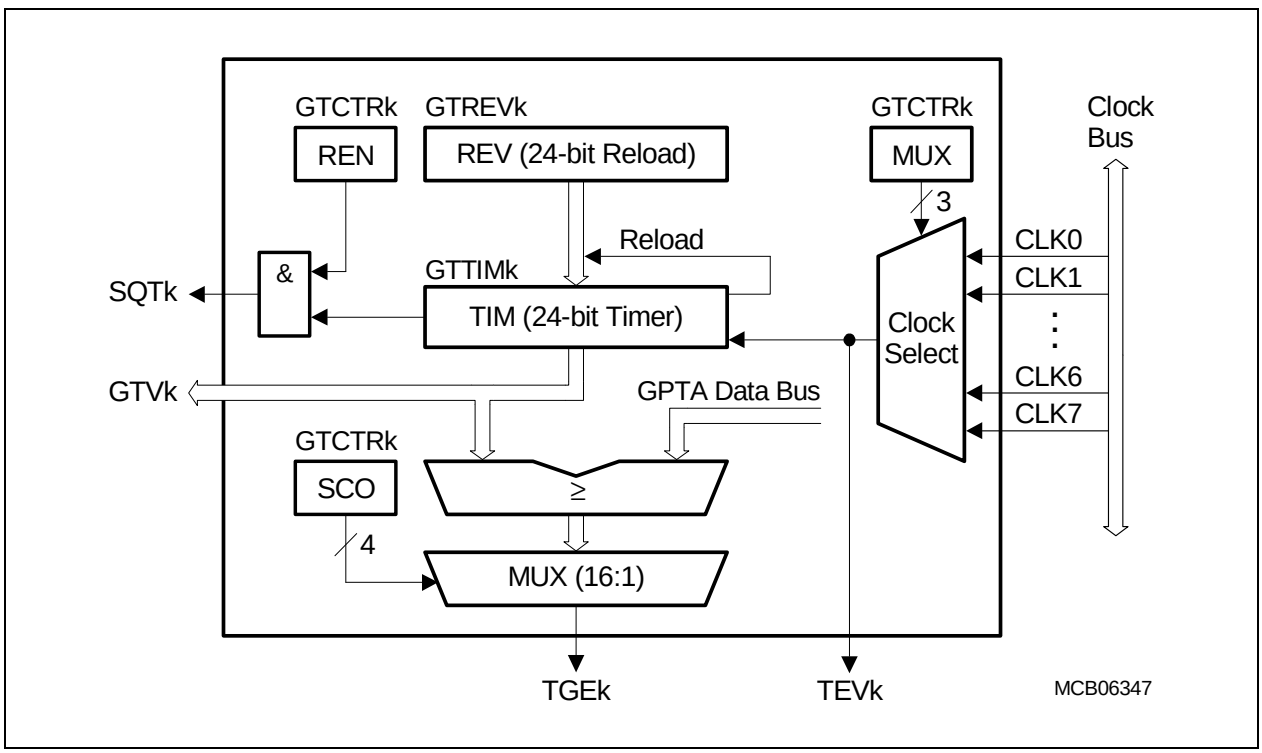


Figure 22-24 Block Diagram of Global Timer (GT)

The following registers are assigned to the Global Timers GTk (k = 0, 1):

- GTCTRk = Global Timer Control Register k (see **Page 22-170**)
- GTREVk = Global Timer Reload Value Register k (see **Page 22-171**)
- GTTIMk = Global Timer Register k (see **Page 22-171**)
- SRSC0 = Service Request State Clear Register 0 (see **Page 22-203**)
- SRSS0 = Service Request State Set Register 0 (see **Page 22-207**)

Each of the two GT modules provides the following input/output signals:

- Eight clock inputs, connected to the clock bus from the clock distribution unit (CDU)
- 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 22-55**).

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 22-25**. 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. Further details on service request and interrupt handling are given in section **"Interrupt Sharing Unit (IS)" on Page 22-112**.

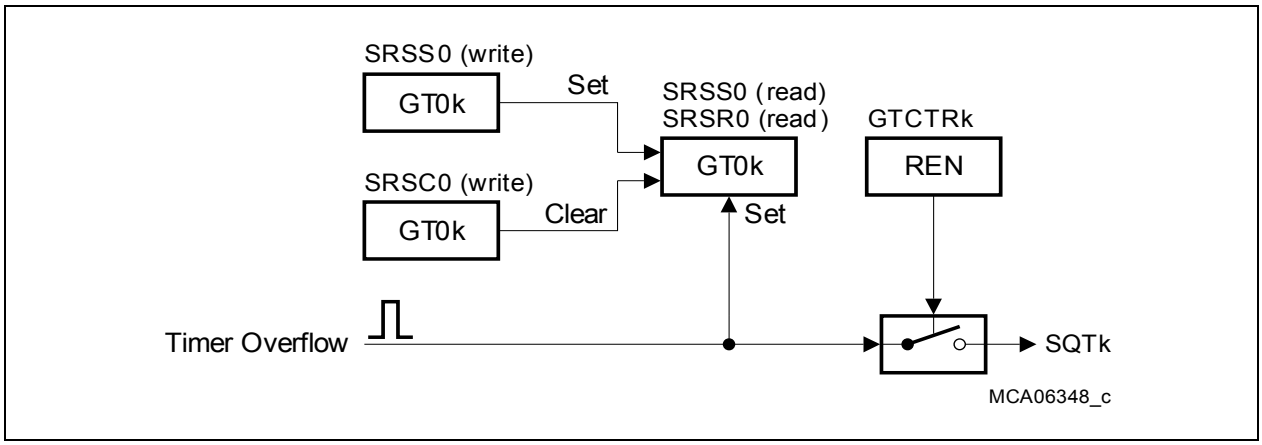


Figure 22-25 GTk Service Request Generation

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 kernel (see also **Page 22-5**). Signal GTkRUN is generated in a GPTA clock control unit. This external control capability makes it possible to control the run signals GTkRUN in a way that all Global Timers of one or more GPTA modules 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 22-53**) explains the **classical timer update problem**, and the solutions supported by the GPTA.

The two Global Timers embedded into the GPTA include a 24-bit greater/equal comparator. This comparator unit performs compare operations between the GT timer contents and the data value found on the GPTA-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 22-26** 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.

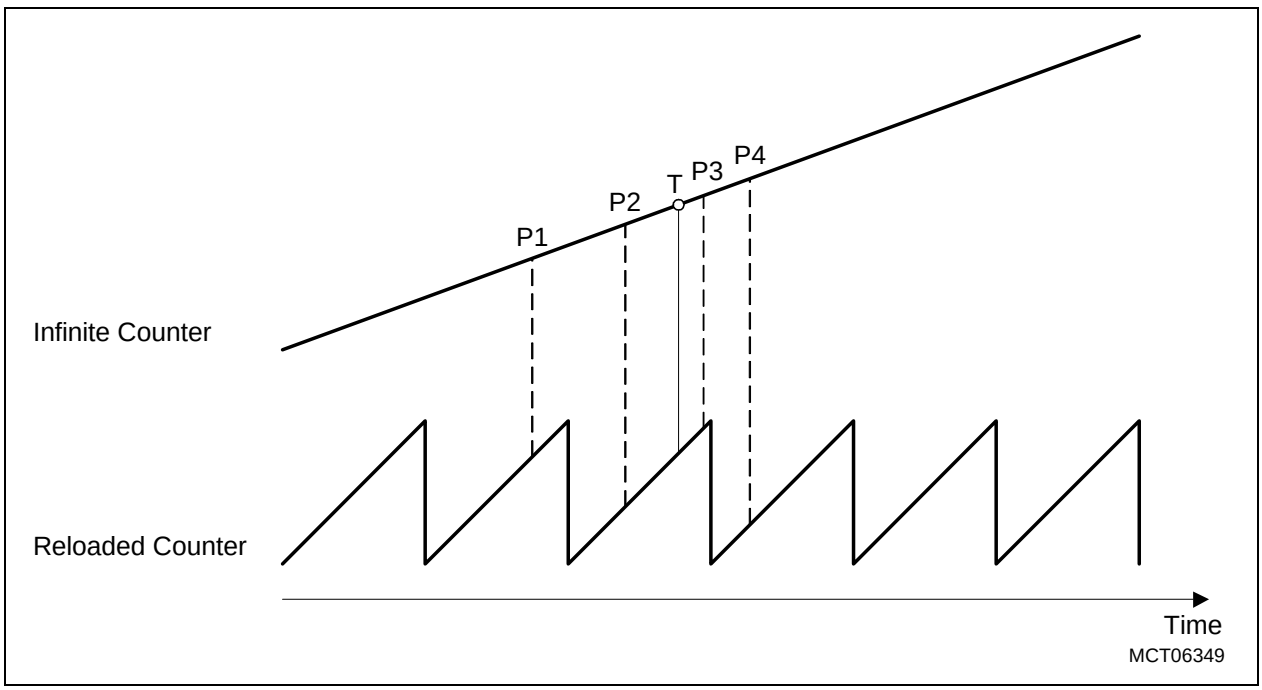


Figure 22-26 Greater/Equal Concept

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 22-27**). 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 ensure 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.

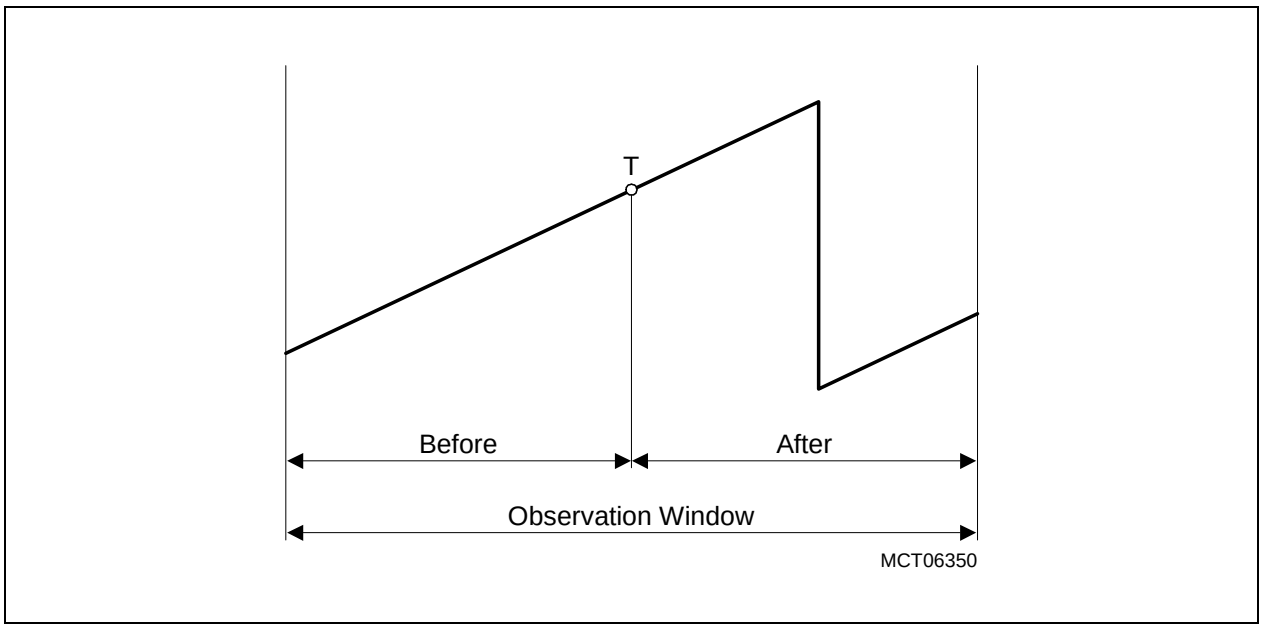


Figure 22-27 Before and After Windows

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 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 22-28**.

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.

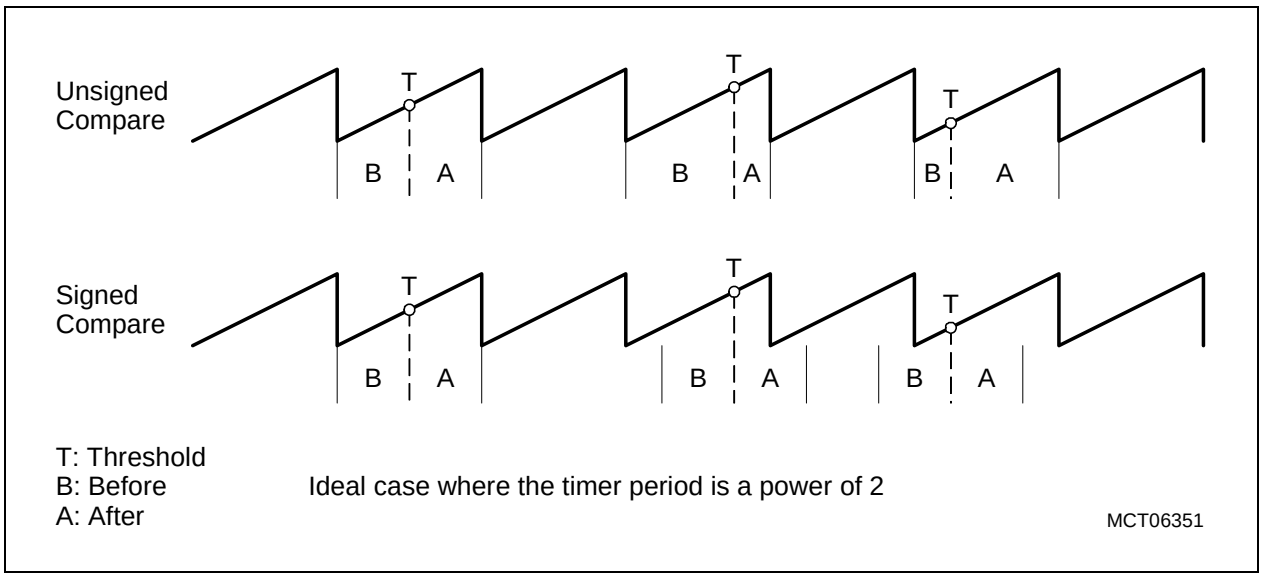


Figure 22-28 Unsigned Versus Signed Compare

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 22-29**.

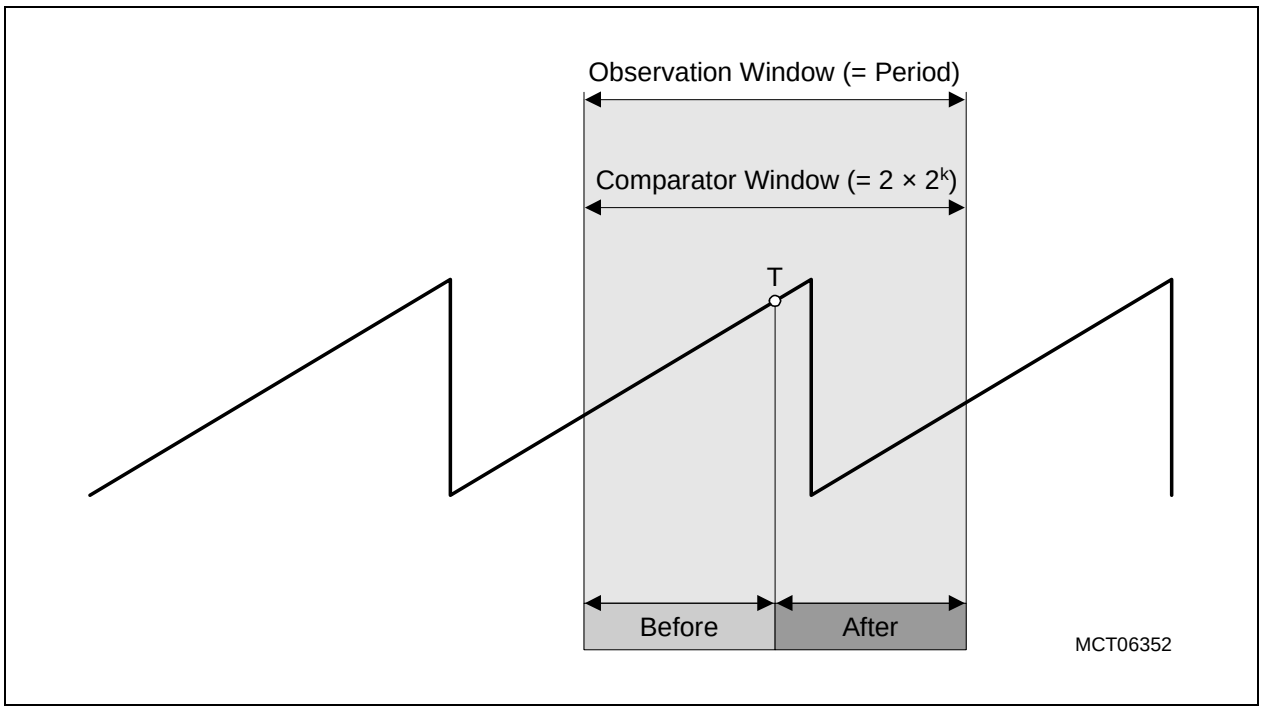


Figure 22-29 Observation and Comparator Windows (timer is a power of 2)

The scalable and signed greater/equal comparator scheme leads to a limitation that must be considered when programming the GPTA 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:

$$2^k < \text{Period} \leq 2 \times 2^k \qquad (22.1)$$

Figure 22-30 and **Figure 22-31** show that one part of the comparator window must be discarded in order to avoid inconsistency, resulting in the observation window.

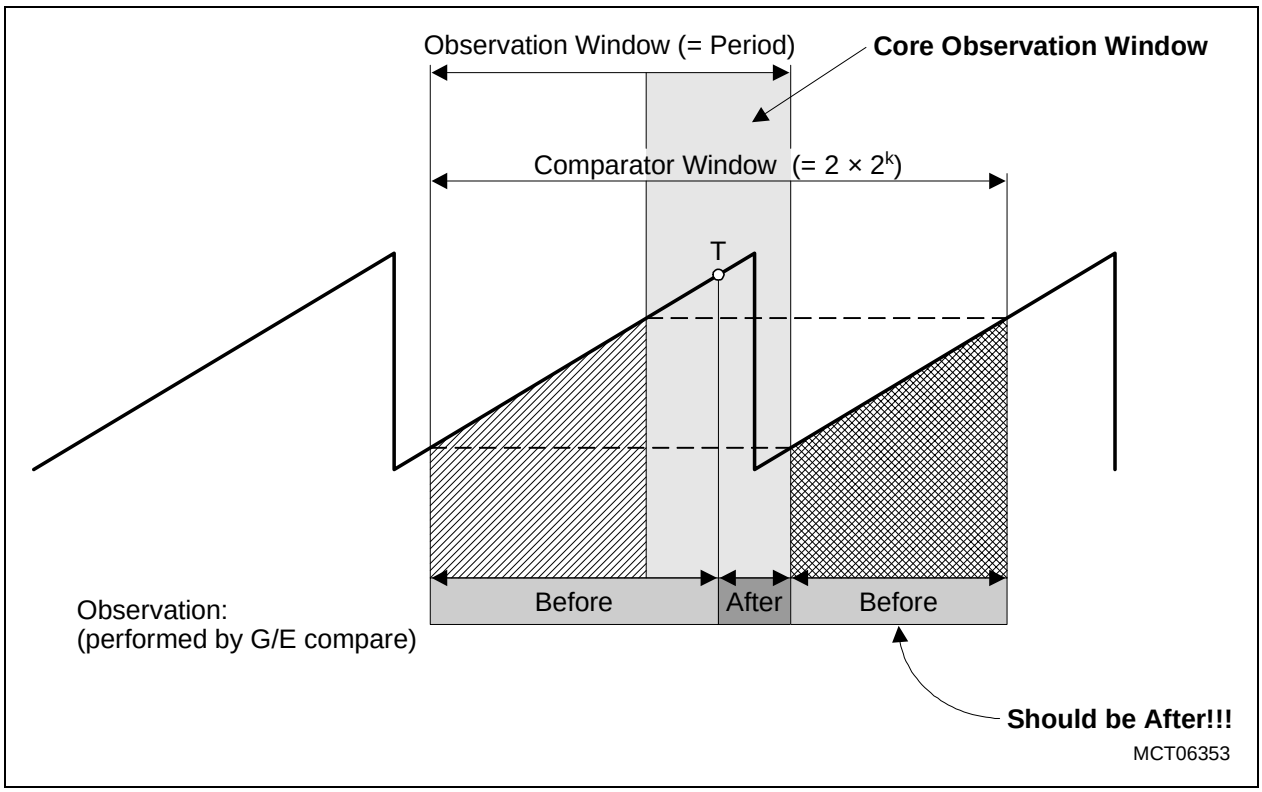


Figure 22-30 Observation Window when Threshold T is High

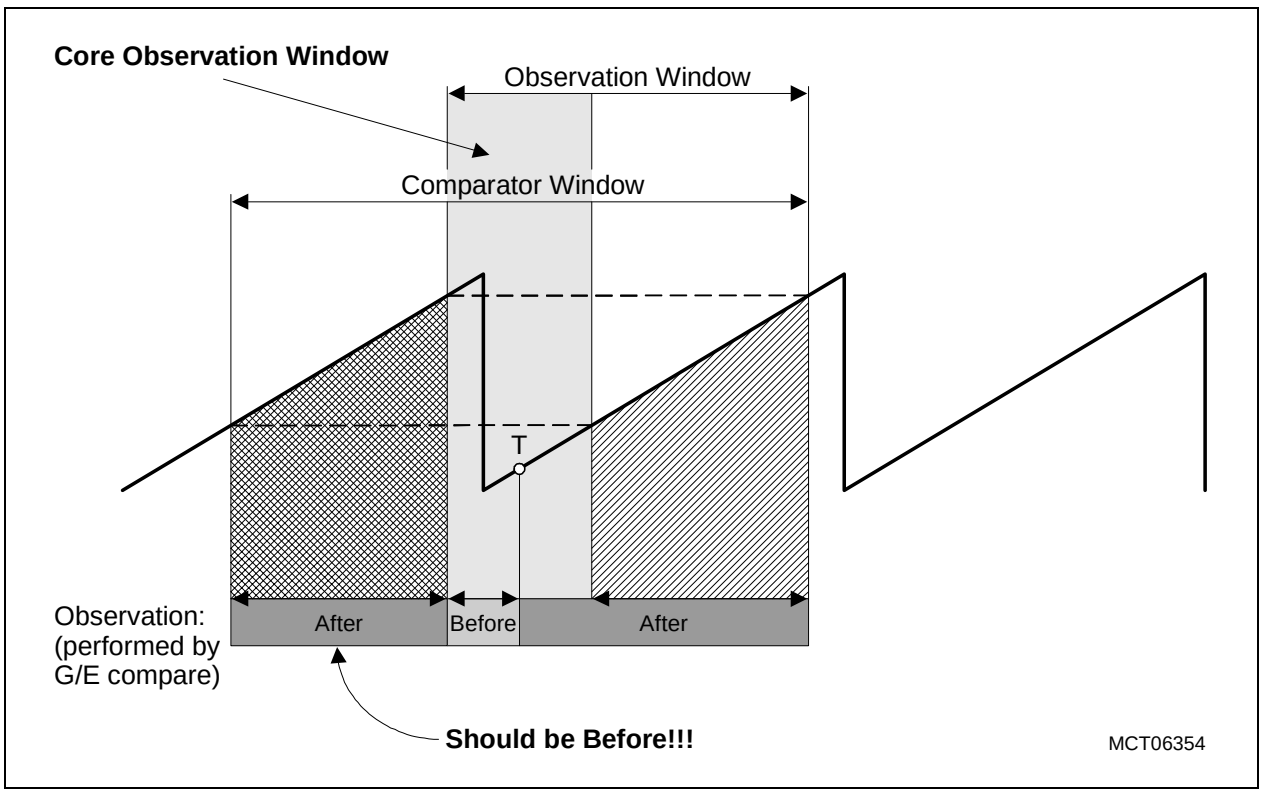


Figure 22-31 Observation Window when Threshold T is Low

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 the value is. 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 22-33**.

For any timer period (whatever the range) and any threshold position, a symmetrical core observation window of a statically defined size can be determined.

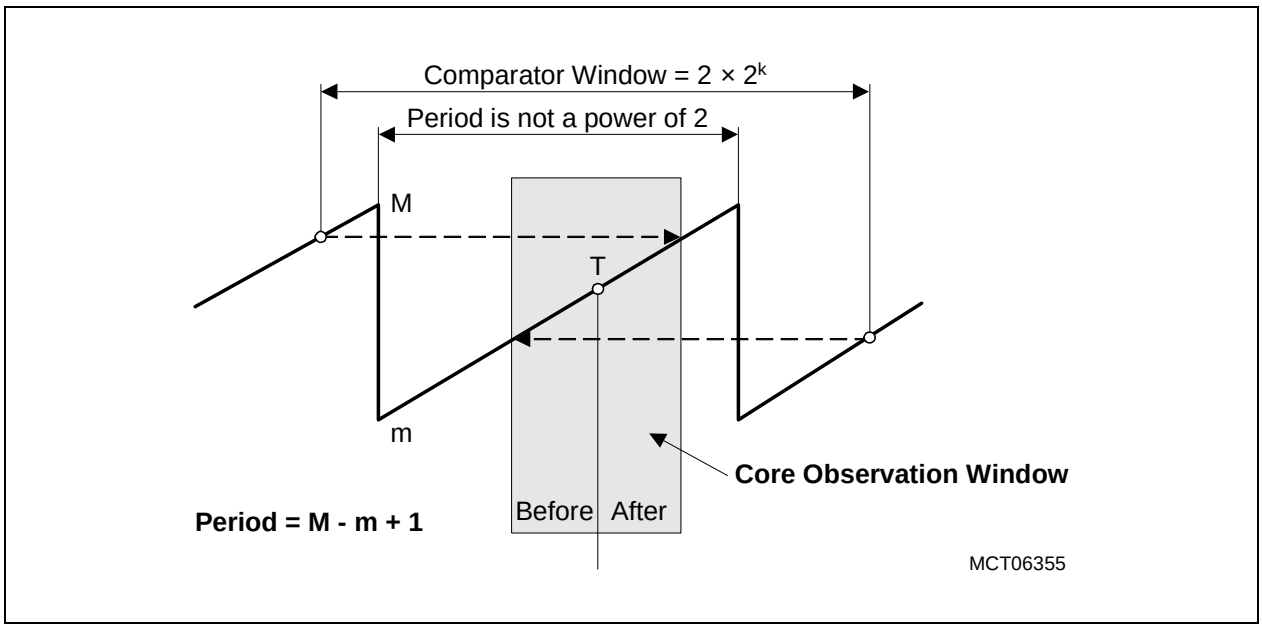


Figure 22-32 The Core Observation Window

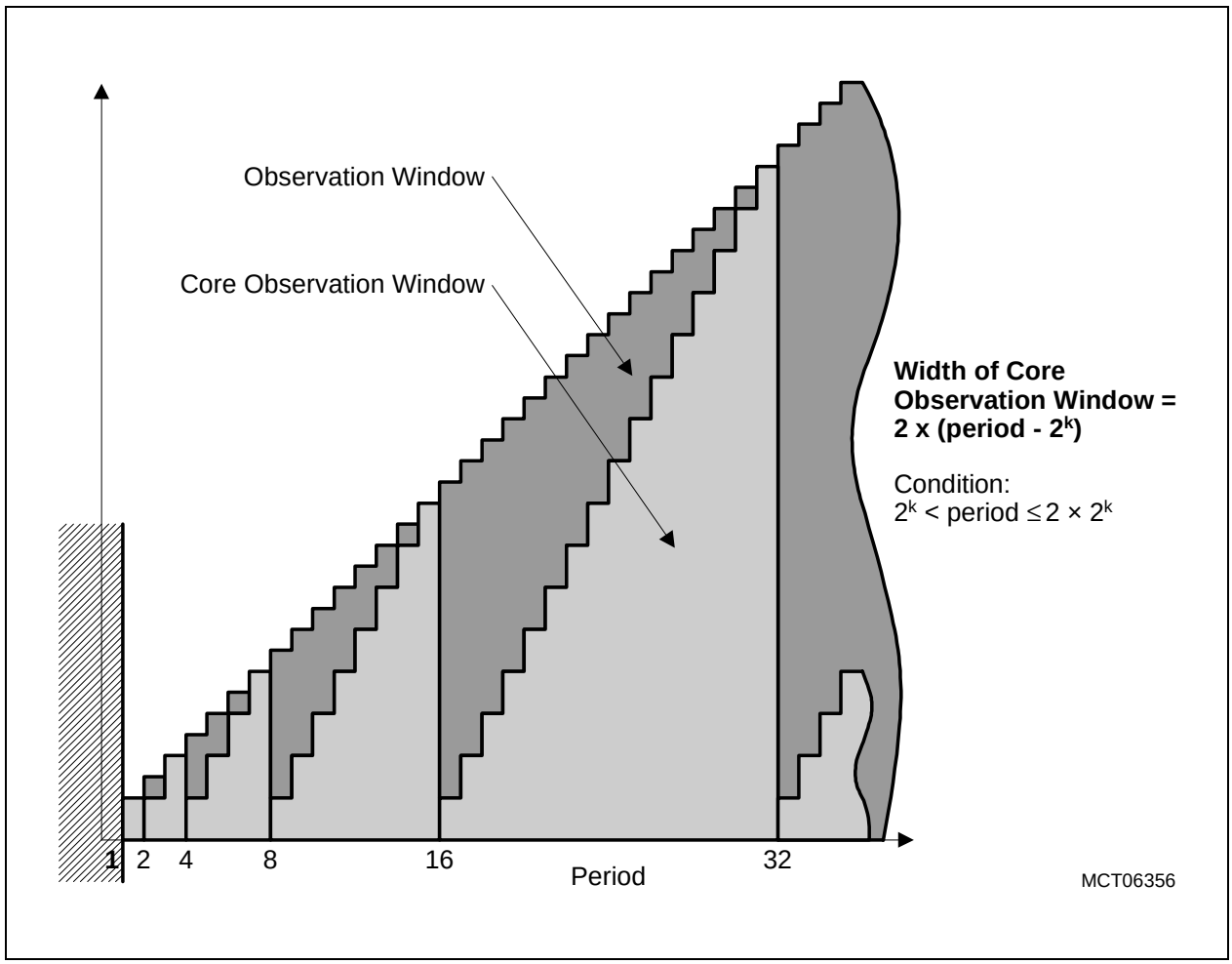


Figure 22-33 Core Observation Window Sizes Versus Period Sizes

Implementation

The hardware implementation of the **scalable and Signed/Unsigned** Greater/Equal compare is illustrated in **Figure 22-34**. 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 available for observation. One of those bits is selected according to the mode of operation (Unsigned or Signed) and the period length (bit field GTCTR.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: On how to choose one of the 15 bits is explained later.

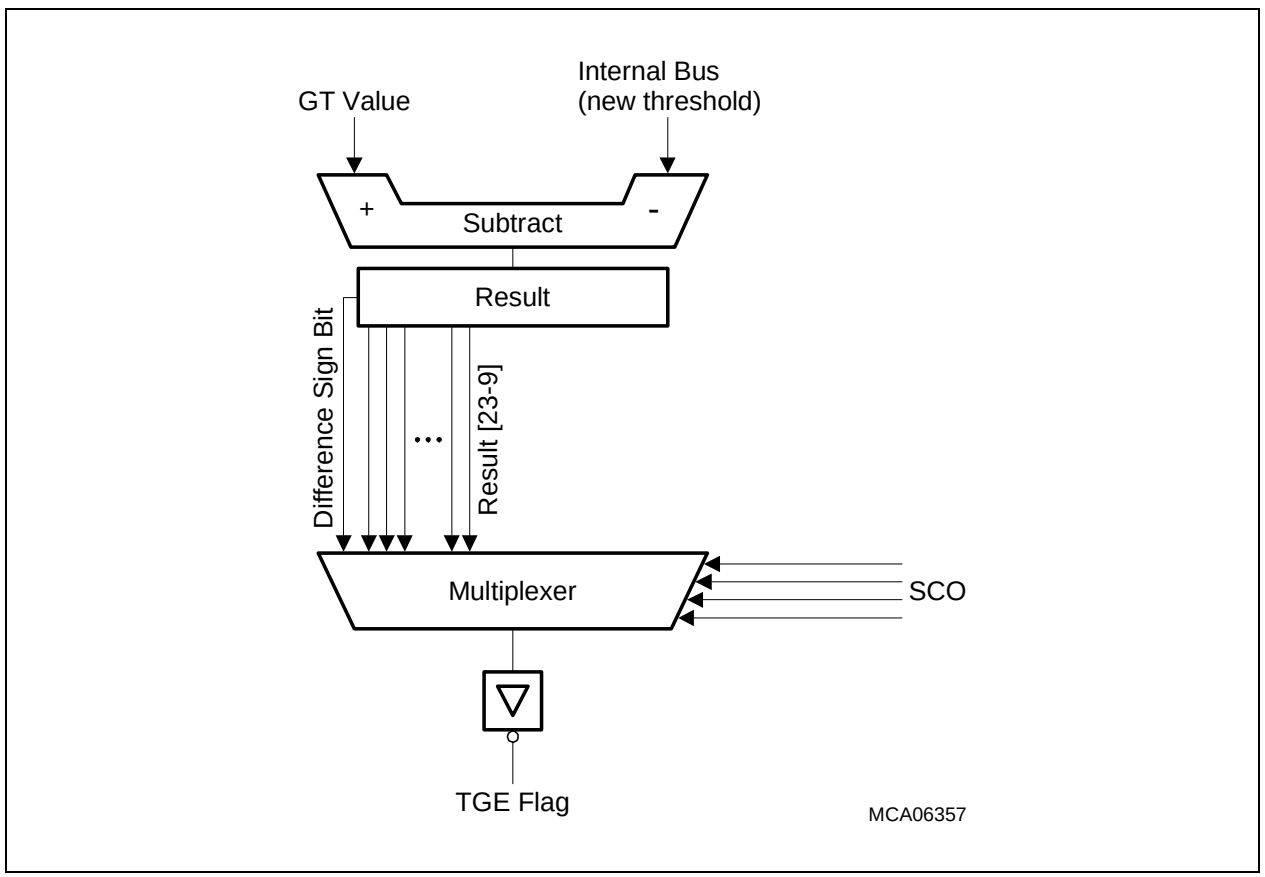


Figure 22-34 Comparator Implemented by a Subtraction Unit

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 22-35**. 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 22-35** and **Figure 22-36** shows why this proceeding leads to correct interpretation within the observation window. **Figure 22-36** shows the case of a period equal to a multiple of 2.

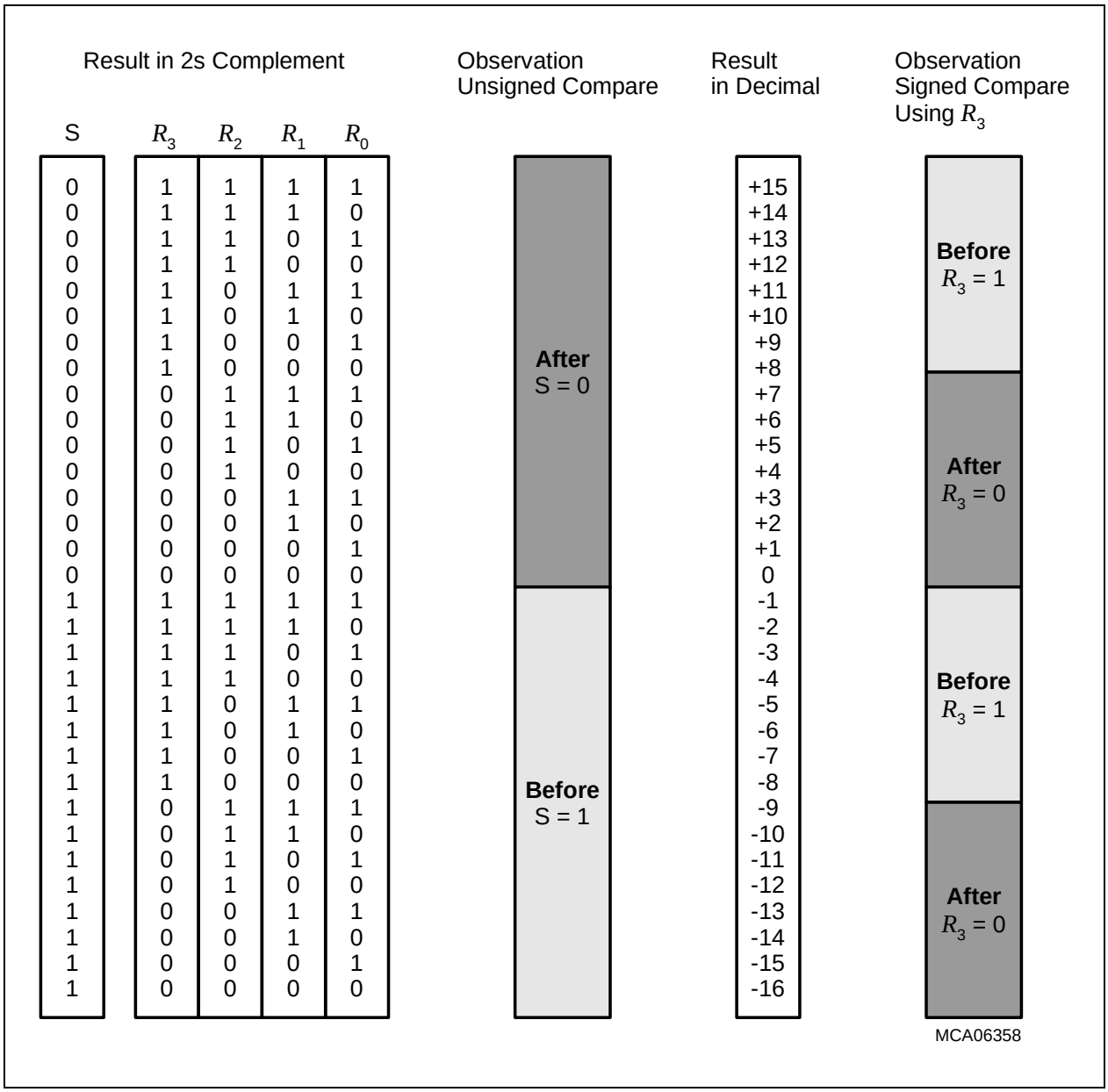


Figure 22-35 Result and Observation for a 4-bit Timer

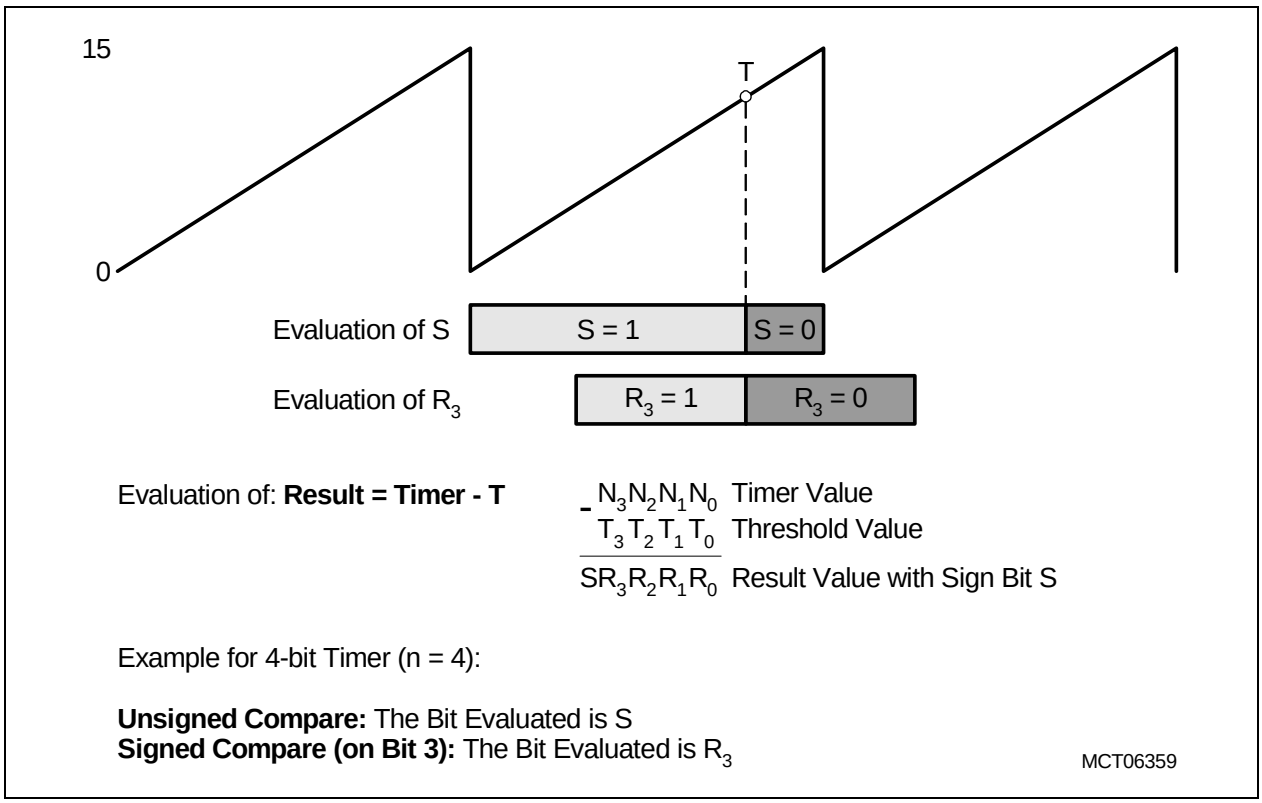


Figure 22-36 Result and Observation (Period = 16)

Figure 22-37 shows the case of a period of 12 which is not a power of 2. Here again, the Table in **Figure 22-35** applies.

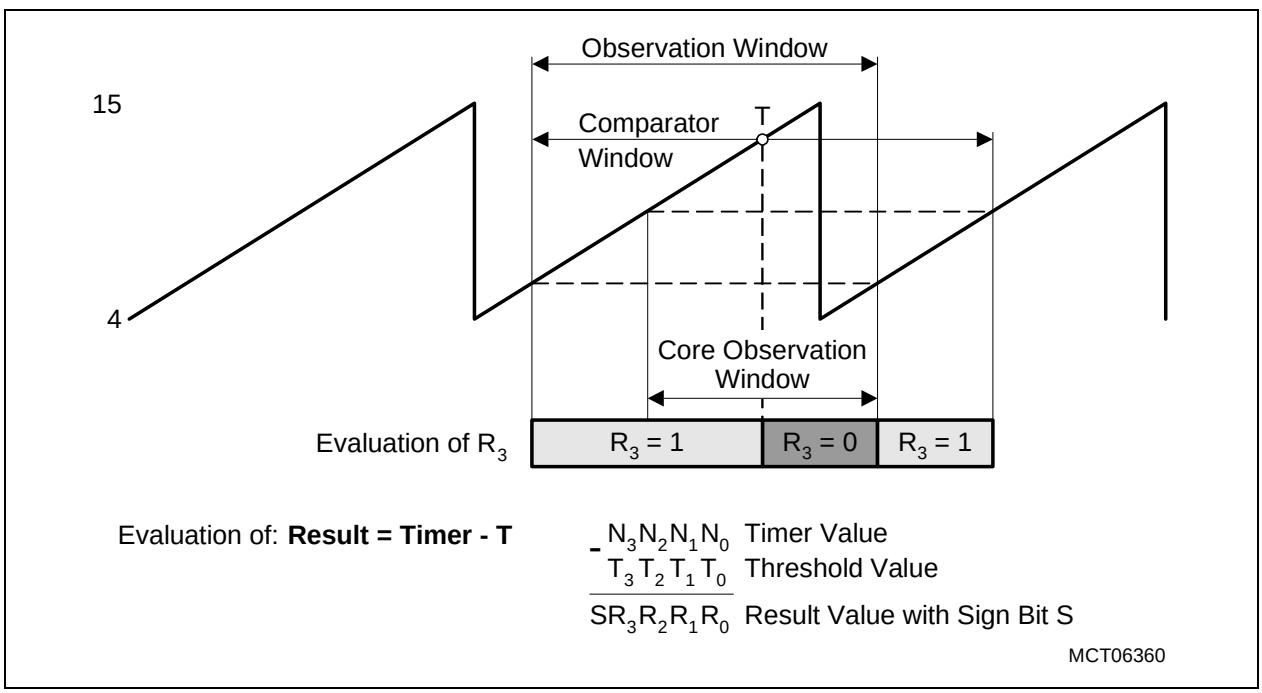


Figure 22-37 Result and Observation (Period = 12)

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.

The letter "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

$2^k < \text{Period} \leq 2 \times 2^k$	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:

$2 \times (\text{period} - 2^k)$ (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 ($m \leq T \leq M$).

Figure 22-38 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.

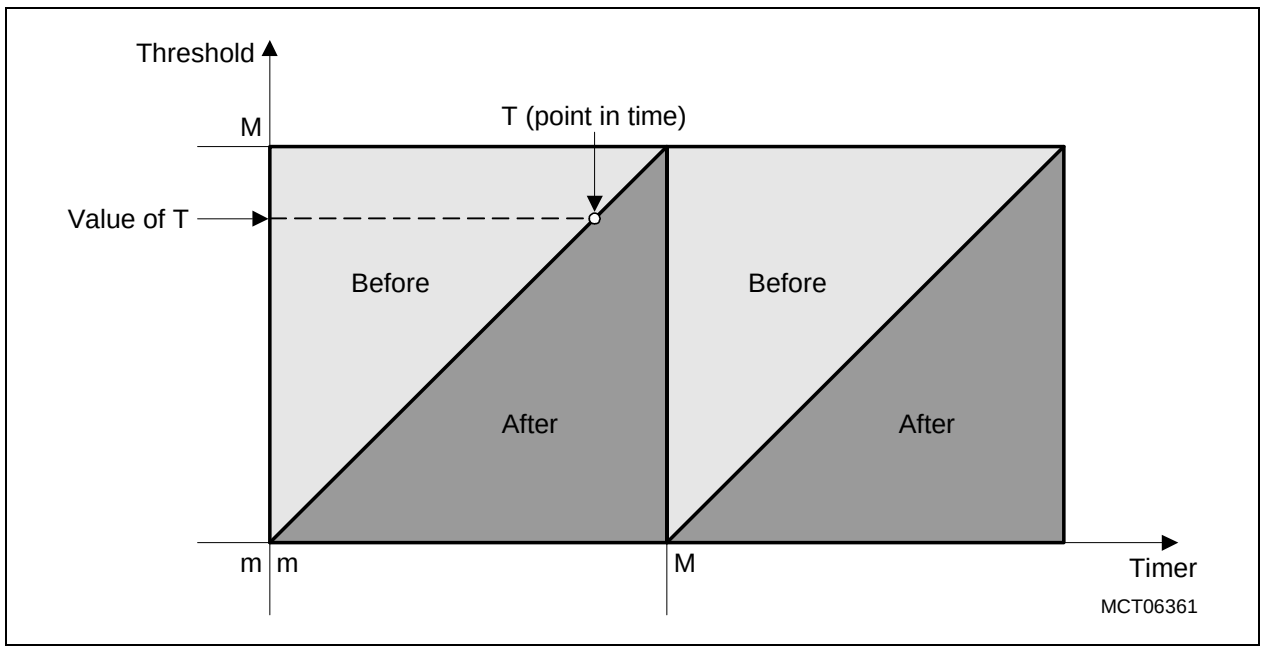


Figure 22-38 Graphical Representation of Unsigned Compare

Figure 22-39 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 **Figure 22-36** where the comparator window equals the period, and the observation window is always centered on the threshold T.

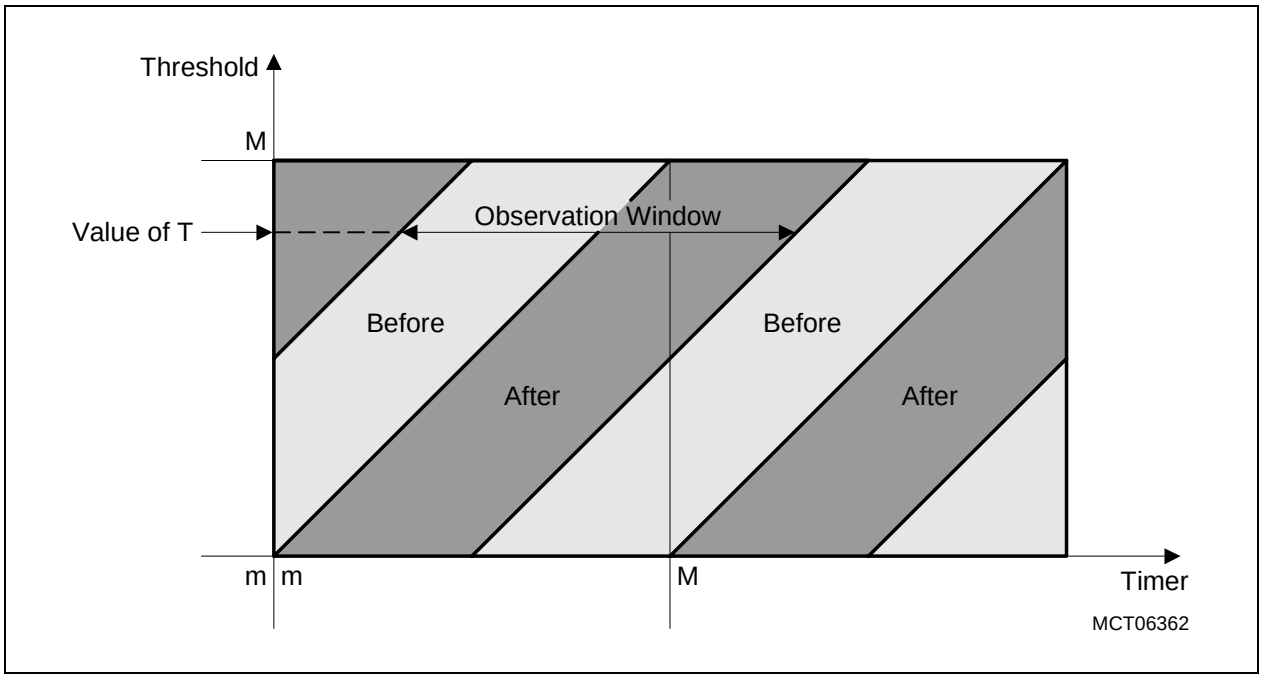


Figure 22-39 Graphical Representation of Signed Compare (Period = 2×2^k)

The **Figure 22-40** 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 22-30**.

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 22-40** (2×2^k - period = comparator window - period).

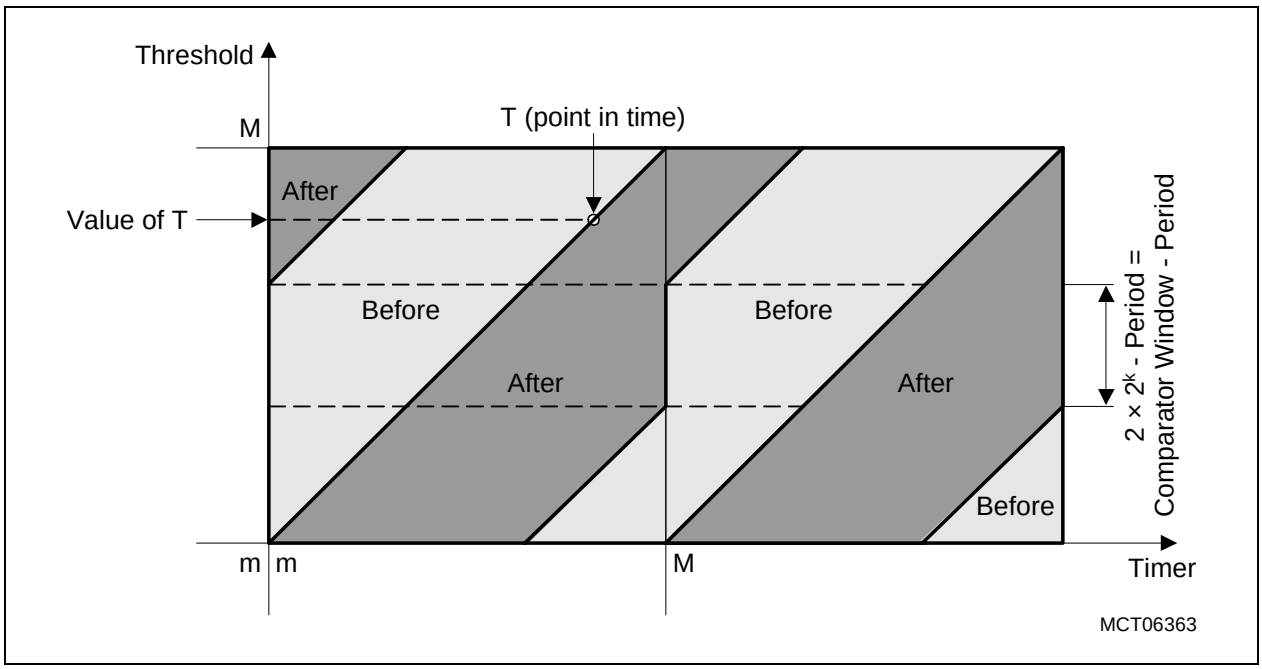


Figure 22-40 Graphical Representation of Signed Compare (2^k < Period ≤ 2 × 2^k)

Figure 22-41 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.

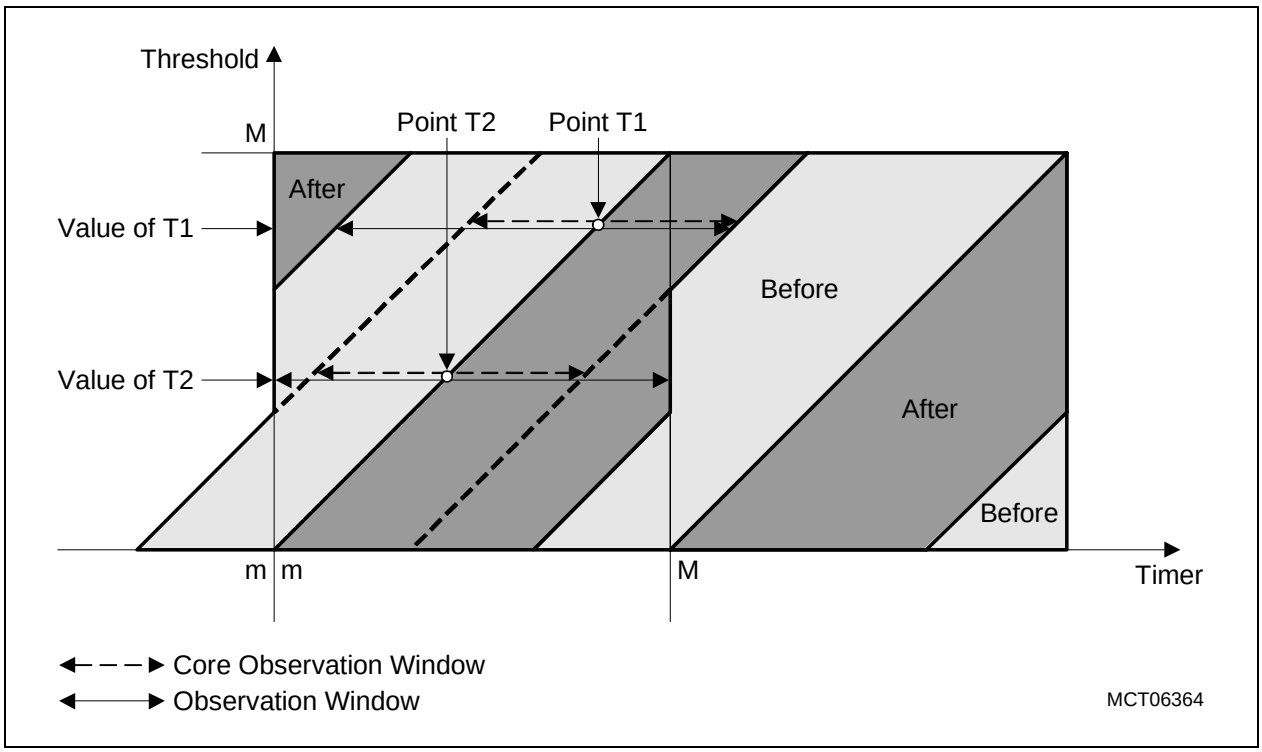


Figure 22-41 Core Observation Window in the Graphic

22.2.3.2 Global Timer Cell (GTC)

The GPTA 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 22-172**)
- GTCXRk = Global Timer Cell X Register k (see **Page 22-176**)
- SRSC1 = Service Request State Clear Register 1 (see **Page 22-204**)
- SRSS1 = Service Request State Set Register 1 (see **Page 22-208**)

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 22-42**. 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 GTCXk, 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.

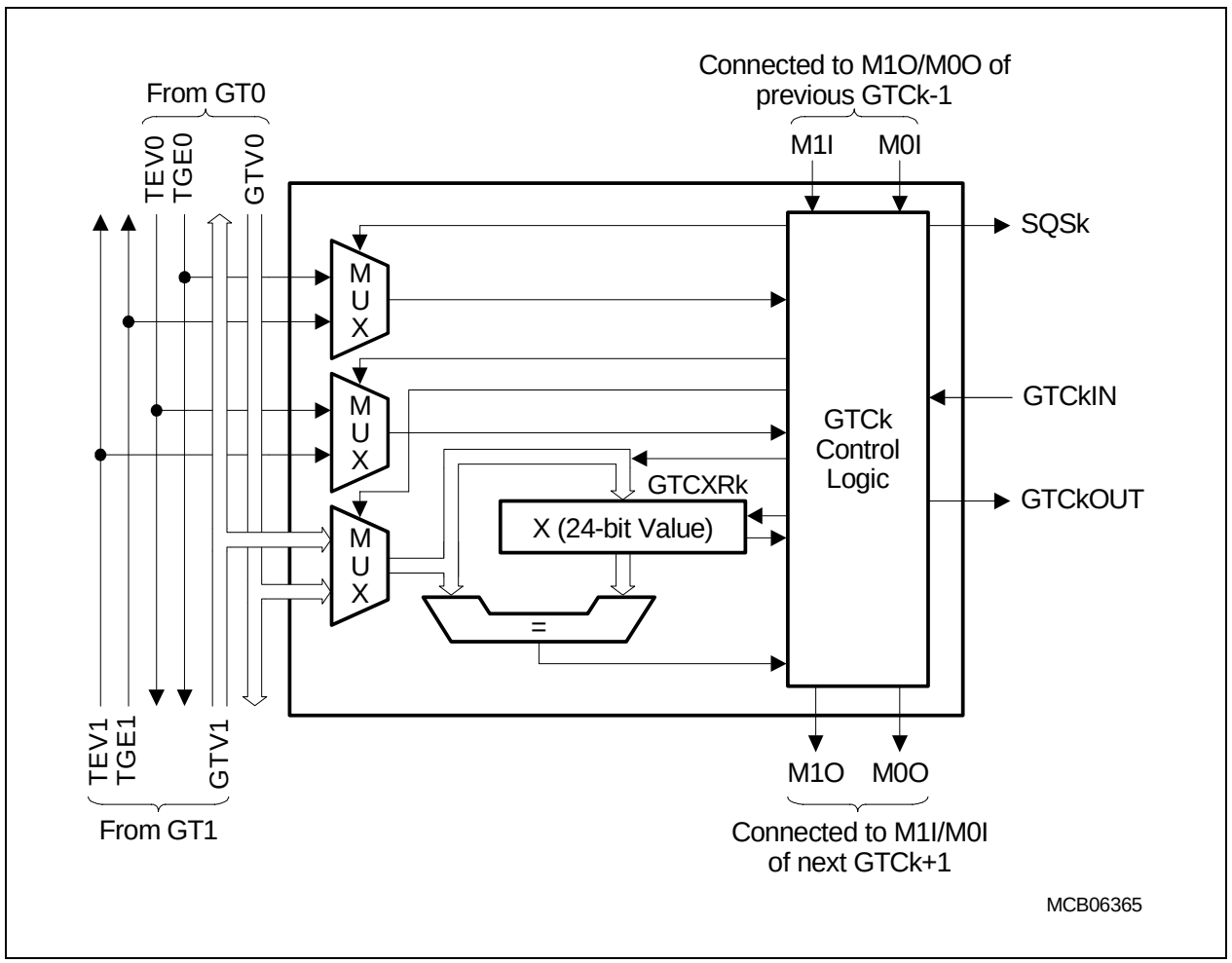


Figure 22-42 Architecture of Global Timer Cells

Figure 22-43 shows how the GTCs are arranged and connected to the adjacent GTCs and with the Global Timers GT0 and GT1.

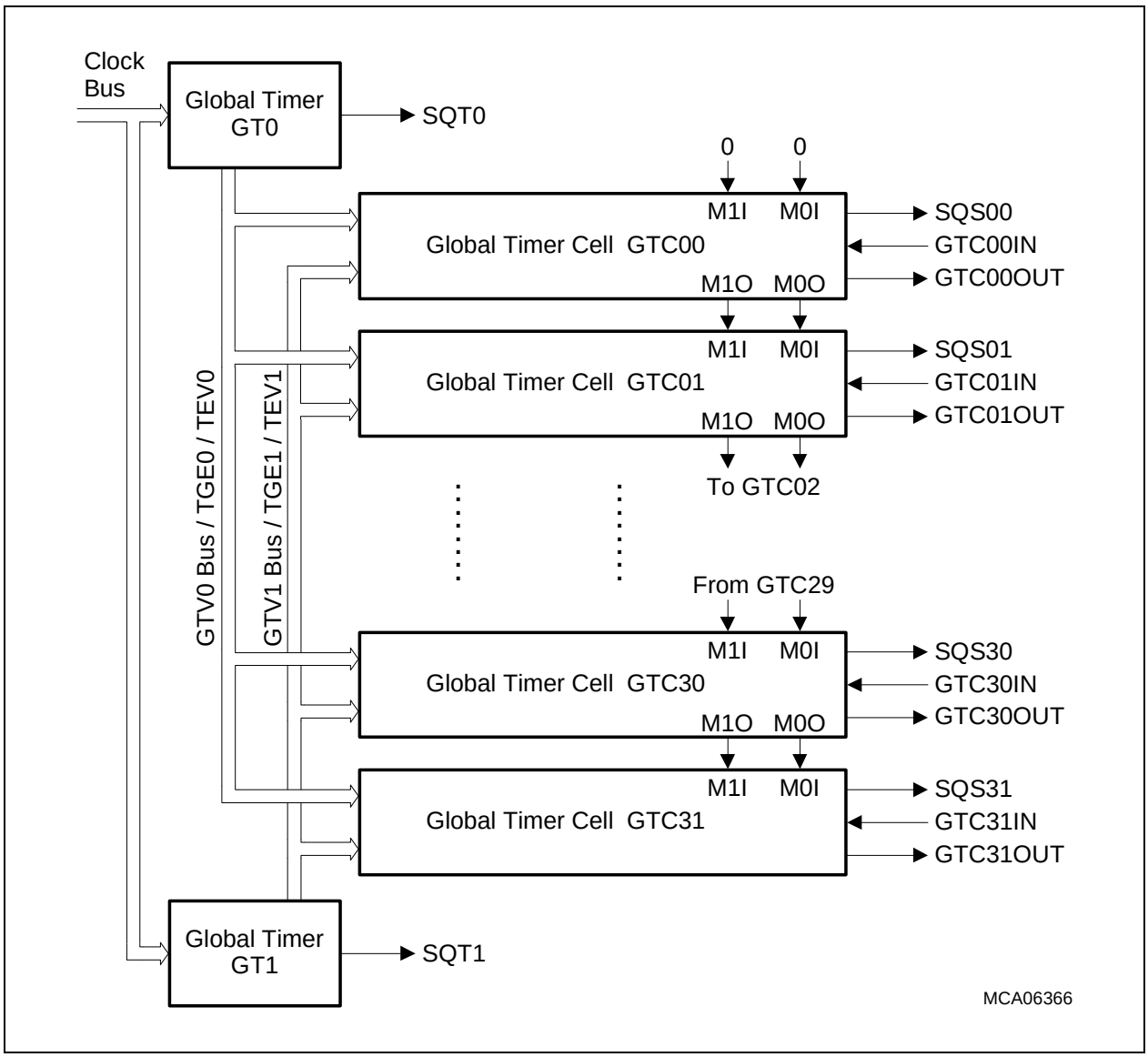


Figure 22-43 GTC Interconnections

Note: Cascading of GTCs is limited. TC1766 specific details are given on ***Page 22-231****.*

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 GTCXkR.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 a 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 Mode 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 a 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 ensure 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 22-44**).

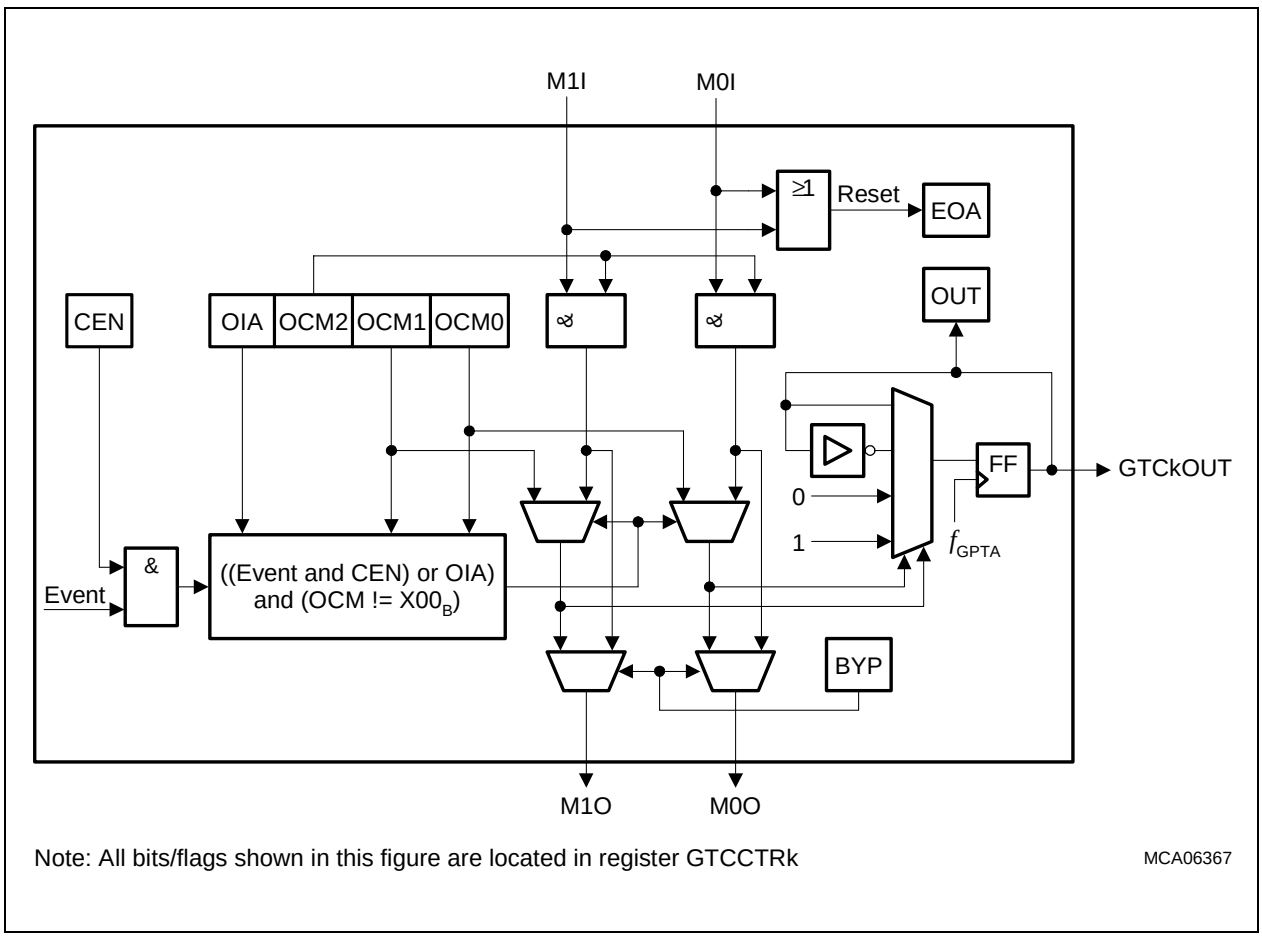


Figure 22-44 GTC Output Operation and Action Transfer

When bit GTCCTRk.OCM2 is cleared, 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 Unit (see Page 22-89). 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 Deactivation

Normally, the GTCs are always enabled. However, by programming a GTC to Capture Mode with no edge selected (GTCCTRk.FED = GTCCTRk.RED = 0), the cell becomes inactive and performs no action, but still passes action commands via the communication link from M1I/M0I to M1O/M0O.

Cell Enabling on Event

A GTC is enabled by writing (ST byte, word, half-word operation) GTCCTRk.EOA (Enable-Of-Action) with 0. Because bit EOA is hardware protected, read-modify-write operations (LDMST, ST.X, SWAP) only enable the GTC if bit EOA is modified from 1 to 0. Alternatively, a GTC can be enabled by an event in a GTC with lower index number. For this purpose, the local event function of a GTC must be disabled by initially setting GTCCTRk.EOA. This 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 22-44**. 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.

Logical Operating Units

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 unit 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 22-45**. When the GTCk service request condition becomes active, the service request flag is always set. The service request output SQSk is only activated if it is enabled by the enable bit GTCCTRk.REN. Additional information on service request and interrupt handling is given on **Page 22-112**.

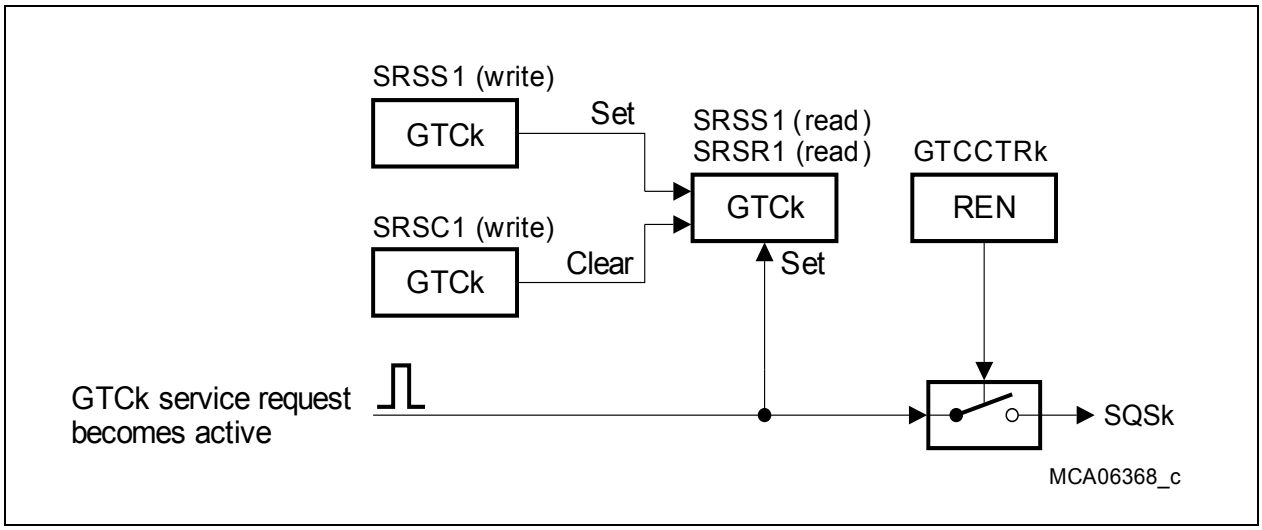


Figure 22-45 GTCk Service Request Generation

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 22-46** 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

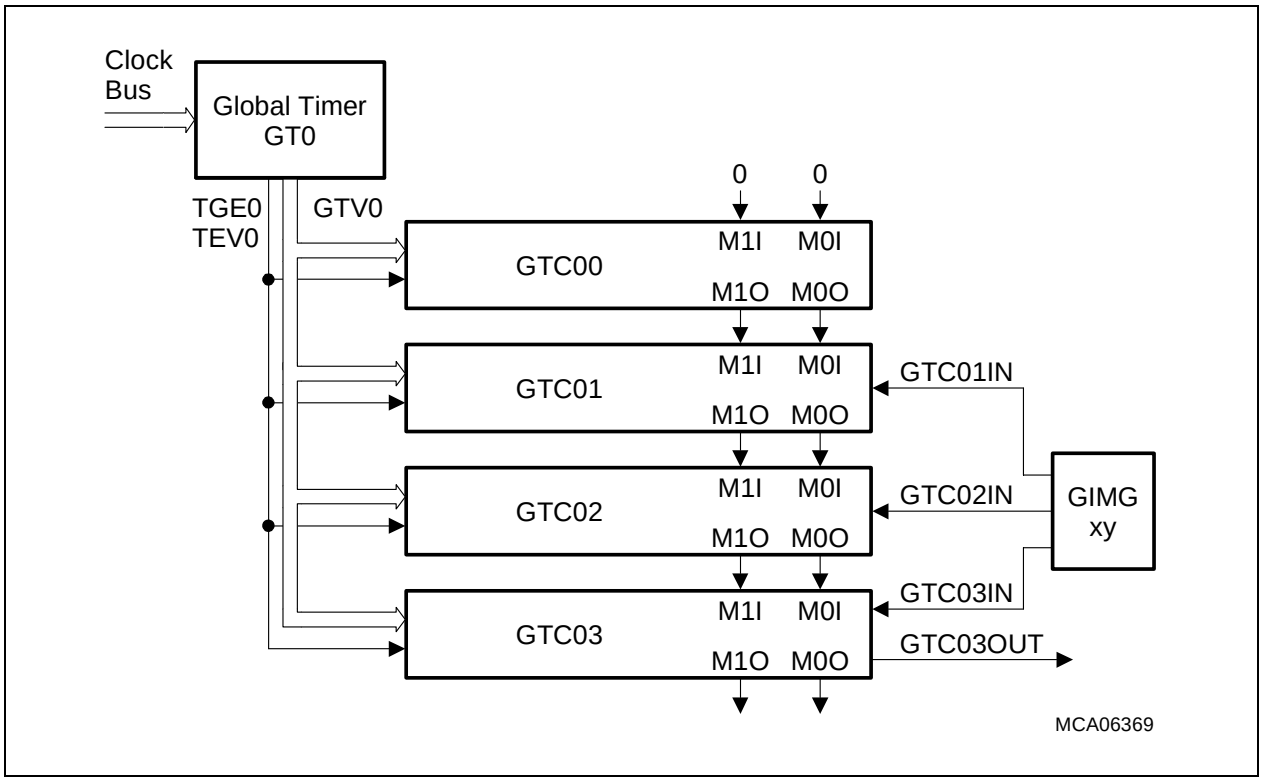


Figure 22-46 Complex Input/Output Signal Capturing/Generation with GTs and GTCs

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 22-100**). 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 hardware without any software interactions and with a resolution of the GT0 clock rate.

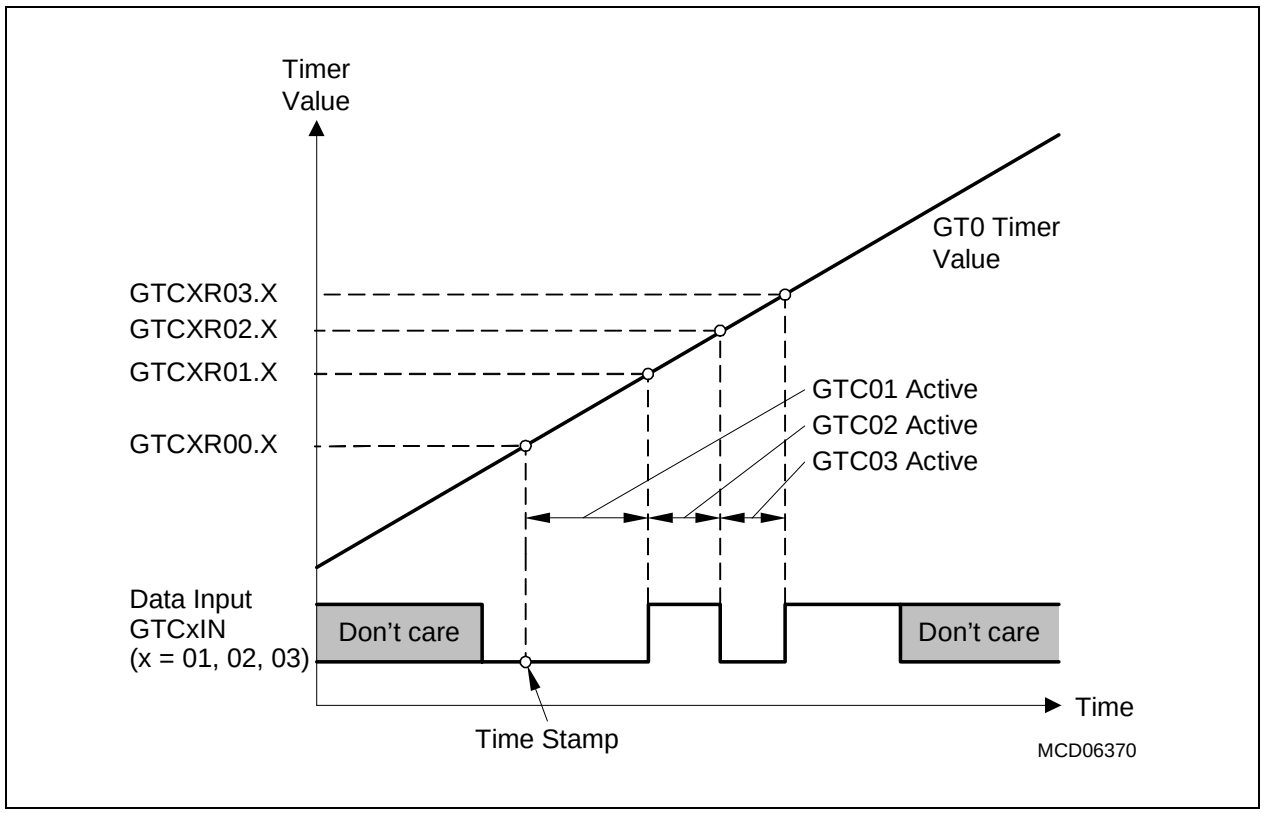


Figure 22-47 Complex Input Signal Analysis/Capturing with GTCs

Complex Periodical Output Signal Generation

This application example uses the GT/GTC configuration as shown in **Figure 22-46**. 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).

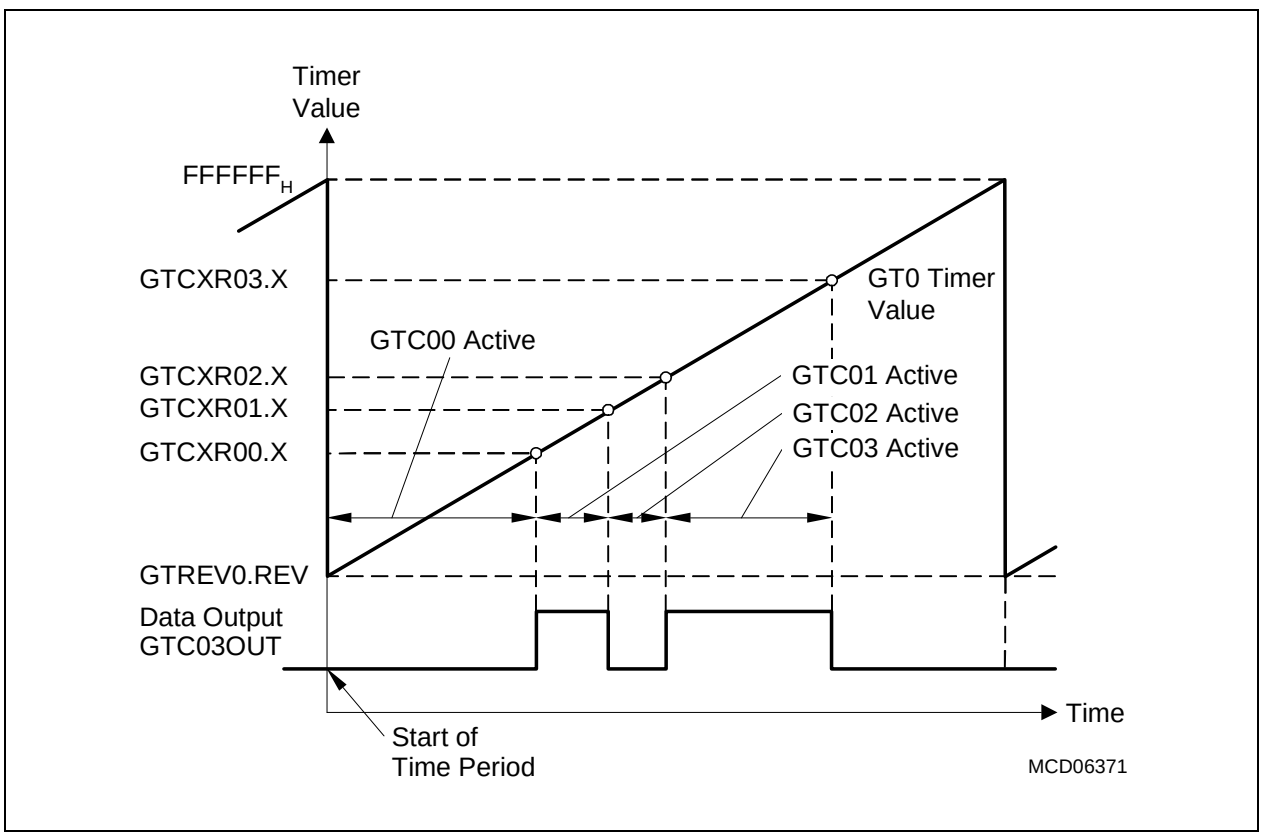


Figure 22-48 Complex Output Signal Generation with GTCs

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 GTC01 becomes 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 hardware without any software interactions and with a resolution of the GT0 clock rate.

22.2.3.3 Local Timer Cell (LTC00 to LTC62)

LTC00 to LTC62 are functionally identical. The functionality of LTC63 is different from LTC00 to LTC62 and therefore described separately on Page 22-78.

Registers

The following registers are assigned to a Local Timer Cell LTCk (k = 00-62):

- LTCCTRk = Local Timer Cell Control Register k (see Page 22-177)
- LTCXRk = Local Timer Cell X Register k (see Page 22-186)
- SRSC2 = Service Request State Clear Register 2 (see Page 22-205)
- SRSC3 = Service Request State Clear Register 3 (see Page 22-206)
- SRSS2 = Service Request State Set Register 2 (see Page 22-209)
- SRSS3 = Service Request State Set Register 3 (see Page 22-210)

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 22-49. 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)
- Two action mode inputs (M1I, M0I) coming from the adjacent LTC cell with lower order number (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 kernel input signals INTx (x = 0-3)

Each LTC 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 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
- Two mode lines (M1O, M0O) going to the adjacent LTC with higher order number.

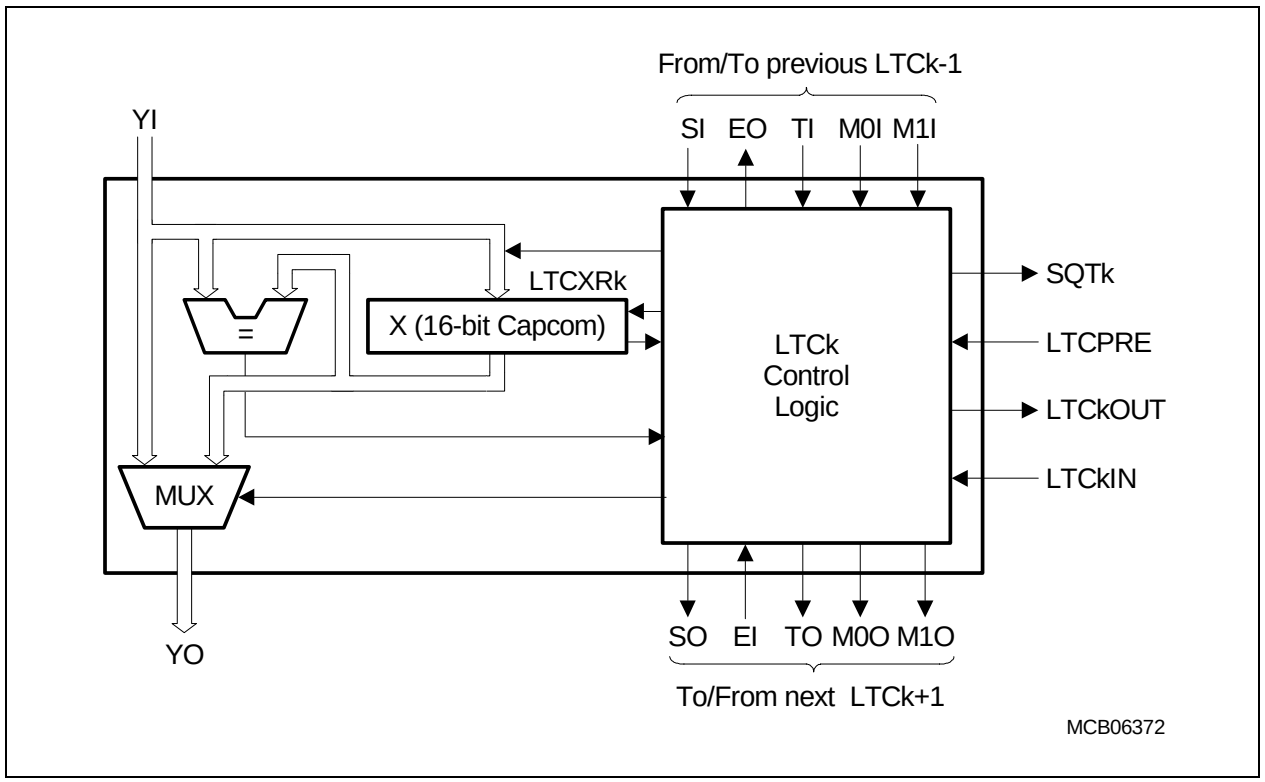


Figure 22-49 Architecture of Local Timer Cells

Figure 22-50 shows the arrangement of the LTCs and the connection to adjacent LTCs. LTC63 is a Local Timer Cell that differs from all other LTCs (LTC00 to LTC62). LTC63 is described in detail on **Page 22-78**.

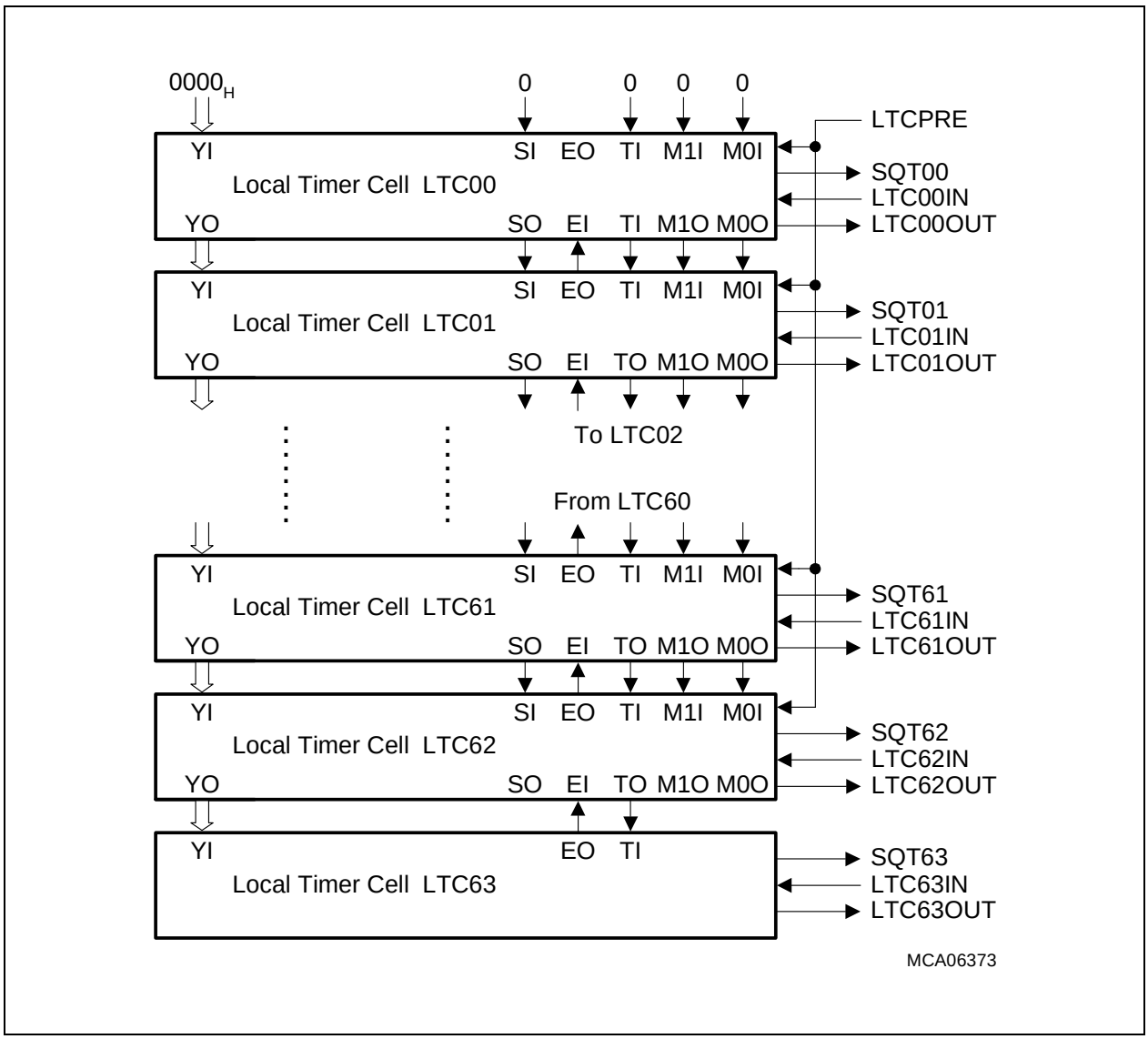


Figure 22-50 Interconnections between the LTCs

*Note: Cascading of LTCs is limited. TC1766 specific details are given on **Page 22-231**.*

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 CDU (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, 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 in 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 disable state is cleared with the next write access to control register LTCCTRk. The current state of LTCk can be checked by reading the control register flag bit LTCCTRk.CEN.

Note: The contents of register LTCXRk is write-protected for Capture_After_Compare in Single Shot Mode. Write protection is activated when the compare value is reached and is released after an access to register LTCXRk.

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 module clock frequency, or with the programmed prescaler clock LTCPRE (see **Page 22-36**).	The programmed function of the LTC cell is performed on selected edge(s).

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
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 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.
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 makes it possible to connect all LTCs via their M1I/M0I inputs and their M1O/ M0O outputs respectively (**Figure 22-51**).

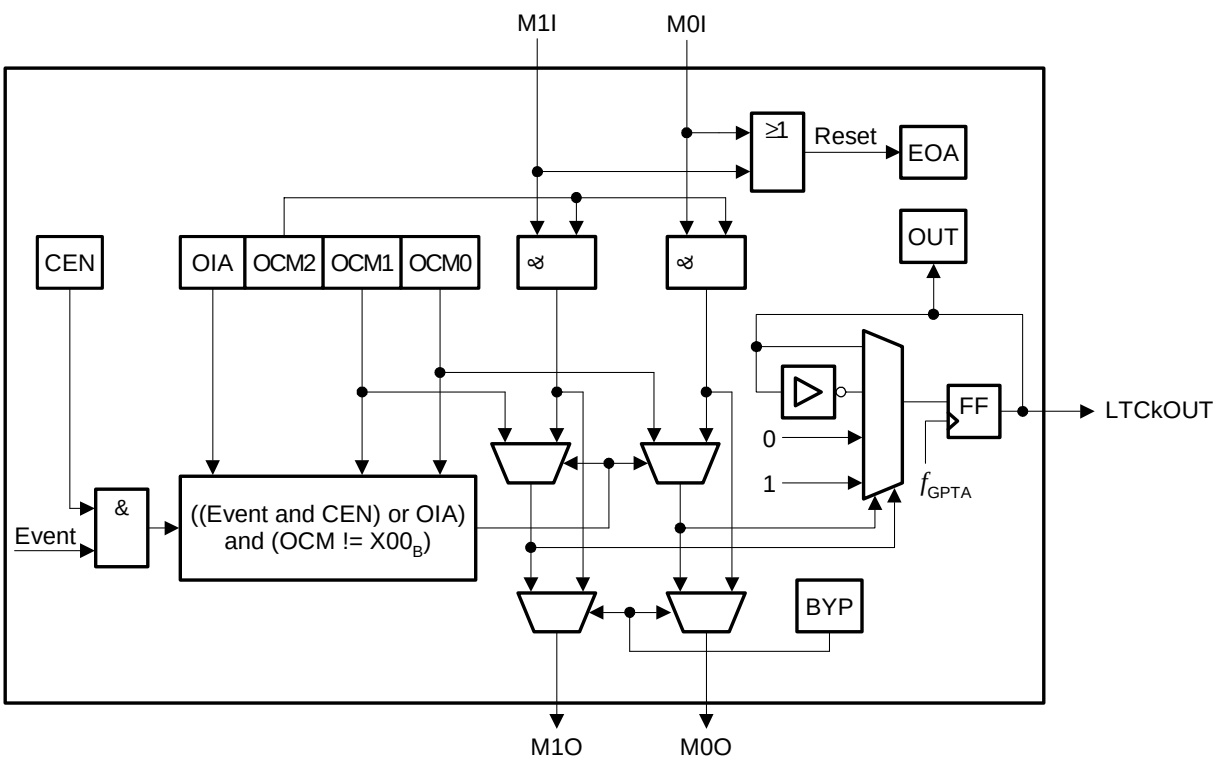


Figure 22-51 LTC Output Operation and Action Transfer

When bit LTCCTRk.OCM2 is cleared, 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 GTCCTRk.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.

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 bit LTCCTRk.BYP is set to 1, a local LTCk event will not modify the M1O/M0O output lines.

Table 22-4 Selection of LTC Output Operations and Action Transfer Modes

Bit Field OCM[2:0]	Local Capture, Compare or Timer Overflow 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 1	0 0	0 0	0 0	not modified 0
0 1 1	not occurred occurred	0 1	0 1	0 0	0 0	not modified 1
1 0 0	not occurred occurred	M1I M1I	M0I M0I	M1I M1I	M0I M0I	modified according M1I/M0I modified according M1I/M0I
1 0 1	not occurred occurred	M1I 0	M0I 1	M1I M1I	M0I M0I	modified according M1I/M0I inverted
1 1 0	not occurred occurred	M1I 1	M0I 0	M1I M1I	M0I M0I	modified according M1I/M0I 0
1 1 1	not occurred occurred	M1I 1	M0I 1	M1I M1I	M0I M0I	modified according M1I/M0I 1

The LTCkOUT output line can be connected to output ports, on-chip peripheral inputs and/or LTC inputs via the I/O Line Sharing Unit (see **Page 22-89**). 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.

Cell Deactivation

Normally, the LTCs are always enabled. However, by programming an LTC to Capture Mode with no edge selected (LTCCTRk.ILM = LTCCTRk.FED = LTCCTRk.RED = 0), the cell becomes inactive and performs no action, but still passes 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 is enabled by writing (ST byte, word, halfword operation) LTCCTRk.EOA (Enable-Of-Action) with 0 in Capture Mode or Compare Mode. Because bit EOA is hardware prodected, read-modify-write operations (LDMST, ST.X) only enable the LTC if bit EOA is modified from 1 to 0 in Capture Mode or Compare Mode. If switching to Timer Mode, the LTC cell is enabled. If in Timer Mode every write operation into bit 0..7 will enable the LTC. Alternatively, an LTC can be enabled by an event in an LTC with lower index number. For this purpose, the local event function of an LTC must be disabled by setting LTCCTRk.EOA initially. This 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 22-51**. 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 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 Units

The inter-cell communication architecture allows concatenation of several LTCs to a logical unit. A logical unit contains any number of LTCs communicating via M1 and M0 lines and ends at an LTC disabled for action input or transfer (such as 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 unit with a time base (YO) and a compare enable signal (SO). Another LTC of the same logical unit 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 unit can operate in Capture Mode triggered by a rising edge, falling edge, or both edges of a GPTA 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 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 22-52**. 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 on service request and interrupt handling is given on **Page 22-112**.

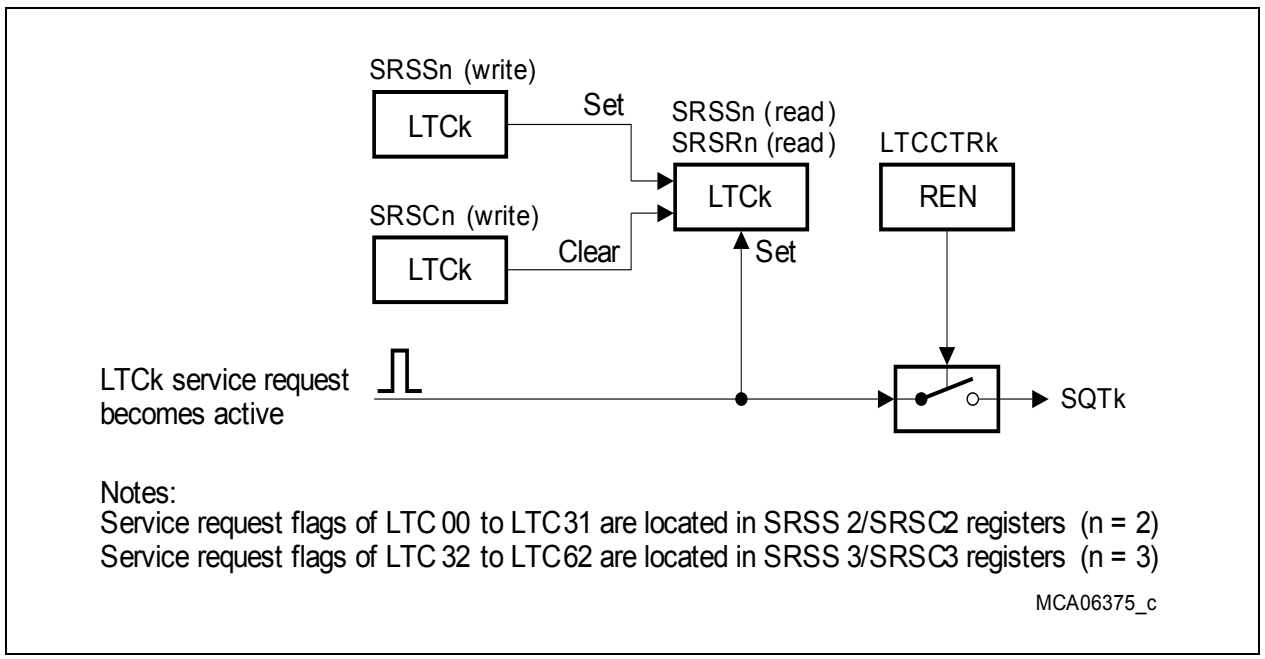


Figure 22-52 LTCk Service Request Generation

22.2.3.4 Local Timer Cell LTC63

This section describes the LTC63 registers.

Registers

The following registers are assigned to Local Timer Cell LTC63:

- LTCCTR63 = Local Timer Cell Control Register 63 (see **Page 22-185**)
- LTCXR63 = Local Timer Cell X Register 63 (see **Page 22-187**)
- SRSC3 = Service Request State Clear Register 3 (see **Page 22-206**)
- SRSS3 = Service Request State Set Register 3 (see **Page 22-210**)

Features

The GPTA Local Timer Cell array has one special cell, LTC63, which provides the following special features:

- **Compare Mode** on greater-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 22-53**).

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 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

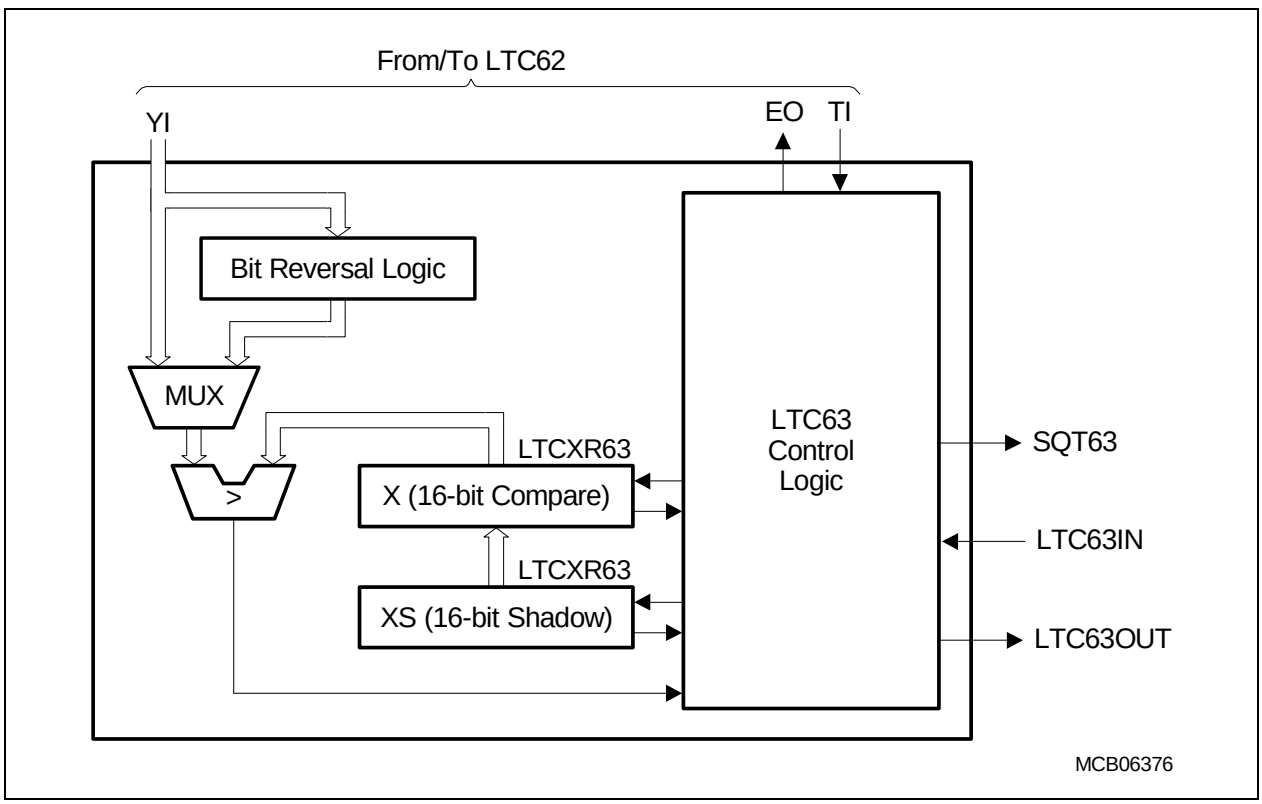


Figure 22-53 Architecture of Local Timer Cell 63

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.

Apart from the 16-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, for example, when building a D/A converter in conjunction with an external low-pass filter. The method shows minimum or no advantage in comparison to the standard PWM only for those with very short or very long duty cycles.

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 22-54 shows an PCM example for an effective period of 6 clocks.

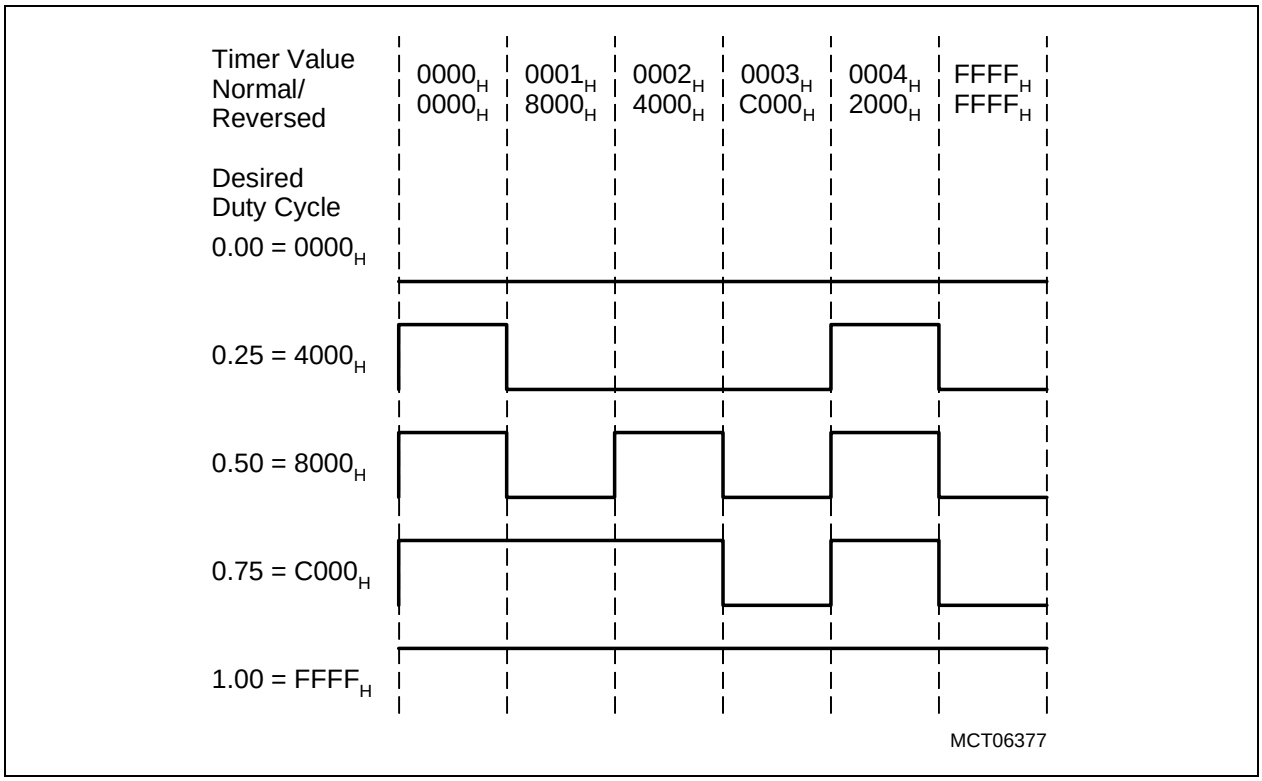


Figure 22-54 Pulse Count Modulation Example 1

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 22-55 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.

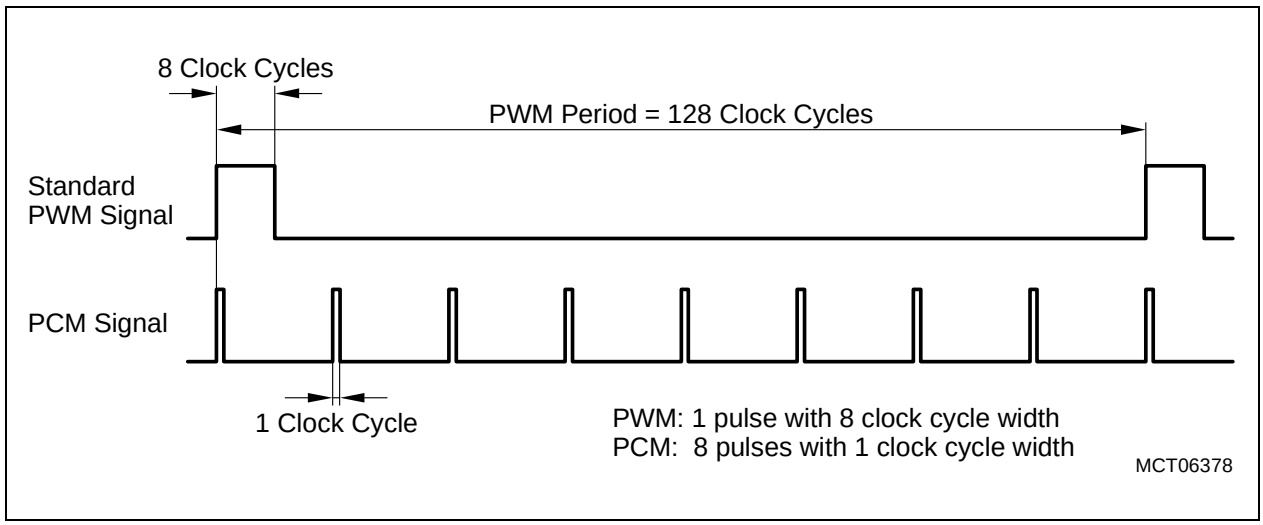


Figure 22-55 Pulse Count Modulation Example 2

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 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 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 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 given on **Page 22-112**.

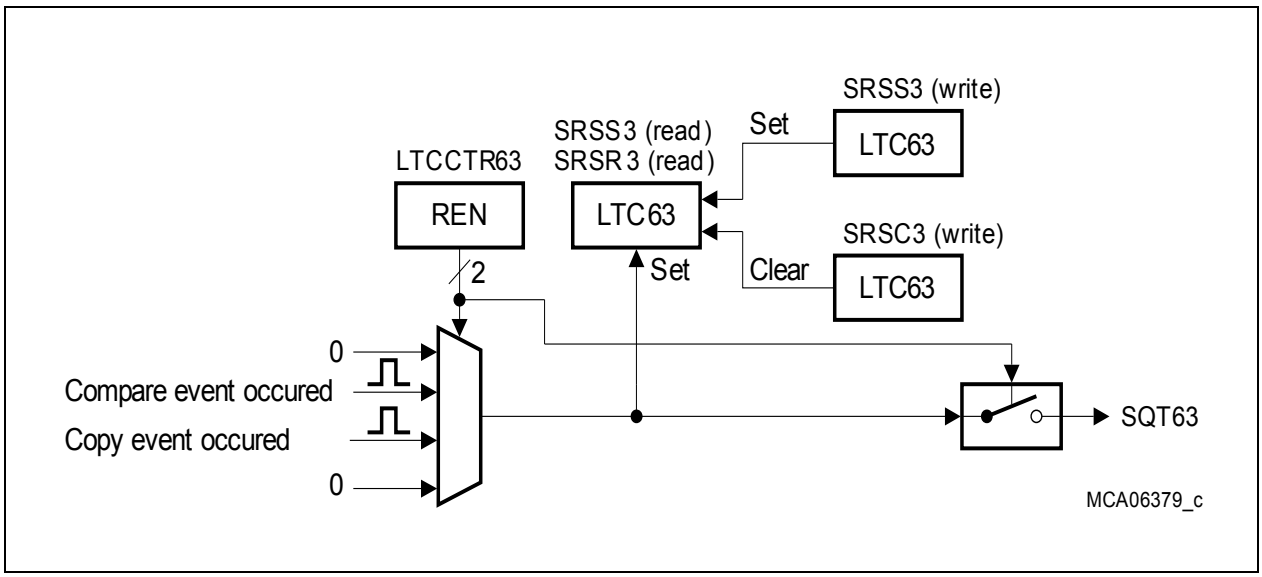


Figure 22-56 LTC63 Service Request Generation

22.2.3.5 LTC Application Examples

This section describes a PWM generation example using Local Timer Cells.

Fully Programmable PWM Signal Generation with 5 LTCs

As shown in **Figure 22-57**, a logical unit of five LTCs can be used to generate a PWM signal with a programmable duty cycle, period length, and fully coherent update of the period and duty cycle. In this example, LTC00 up to LTC04 are used to produce a PWM signal at the output of LTC04.

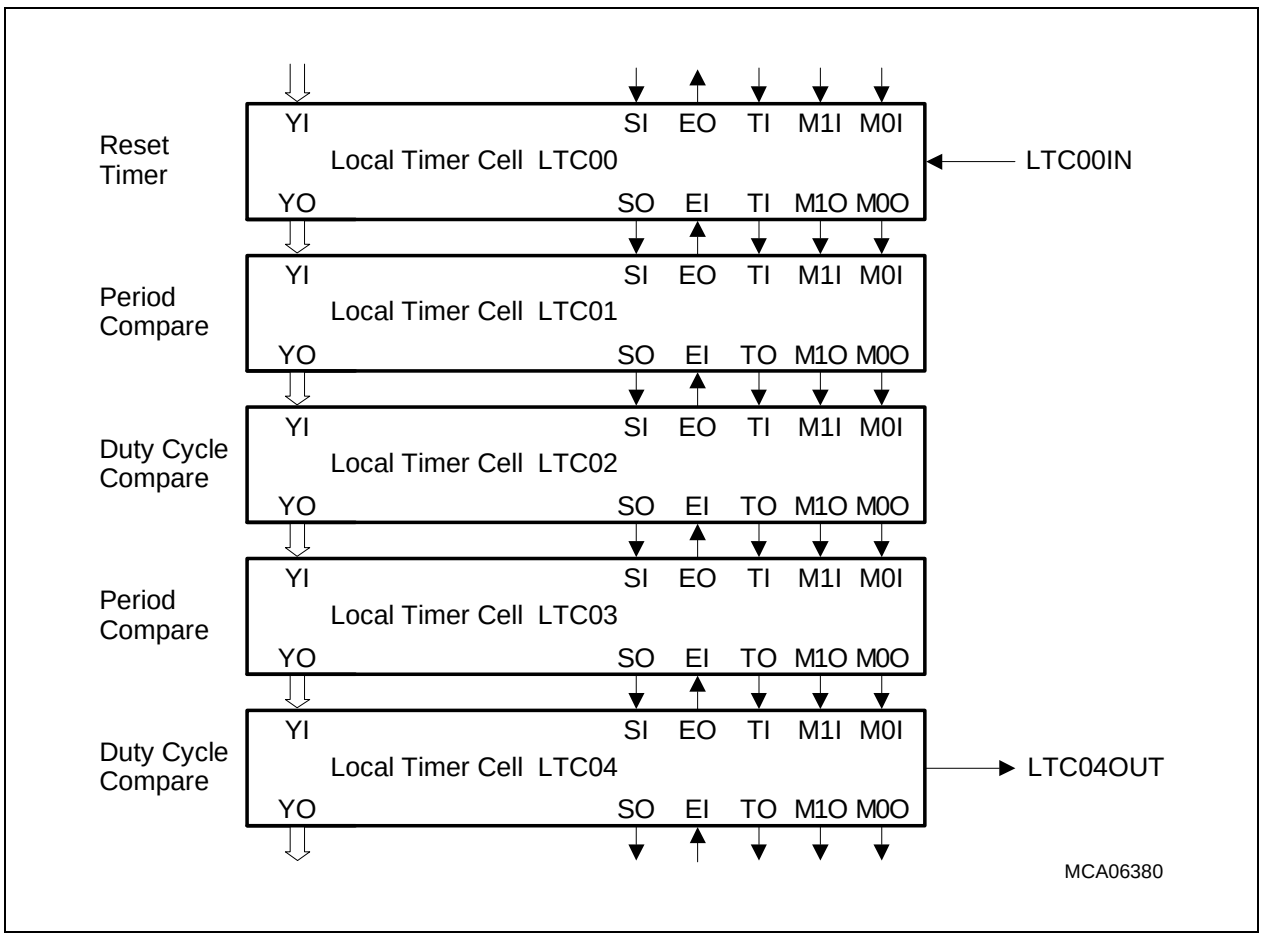


Figure 22-57 PWM Signal Generation with LTCs (Full Coherent Update)

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.

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.

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 $5DC_H$).

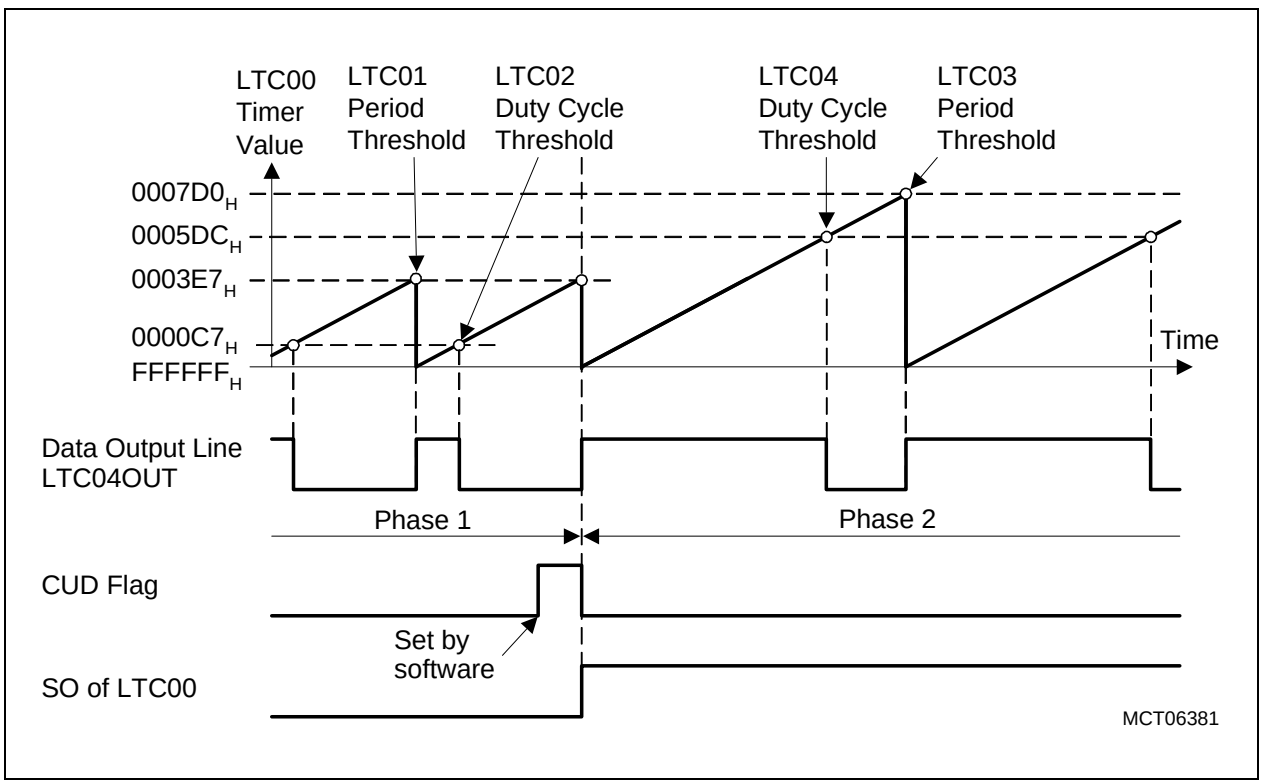


Figure 22-58 Internal Signal States of the PWM Signal Generation with 5 LTCs

LTC00 to LTC05 for the PWM example must be configured as defined in **Table 22-7**.

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	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 = 0, SOL = 1: compare enabled by low level at SI BYP = 0: 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
LTC02 Configuration Setup		
GPTA0_LTCXR02	0000 $00C7_H$	Load compare value = $C7_H$ = 199_D
GPTA0_LTCCTR02	0000 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: 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	0000 $7C21_H$	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: bypass in LTC03 is disabled EOA = 0: LTC03 enabled for local events OCM = 111_B: set LTC03OUT 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	0000 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: 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

22.2.4 Input/Output Line Sharing Unit (IOLS)

The I/O Line Sharing Unit allows the 56 inputs and 112 outputs of the GPTA unit to be routed with high flexibility between I/O lines, output lines, clock inputs, other on-chip peripherals and other GPTA cells. The GPTA module provides a total of 56 input lines and 112 output lines, assigned to seven I/O groups IOG[6:0] and seven output groups OG[6:0].

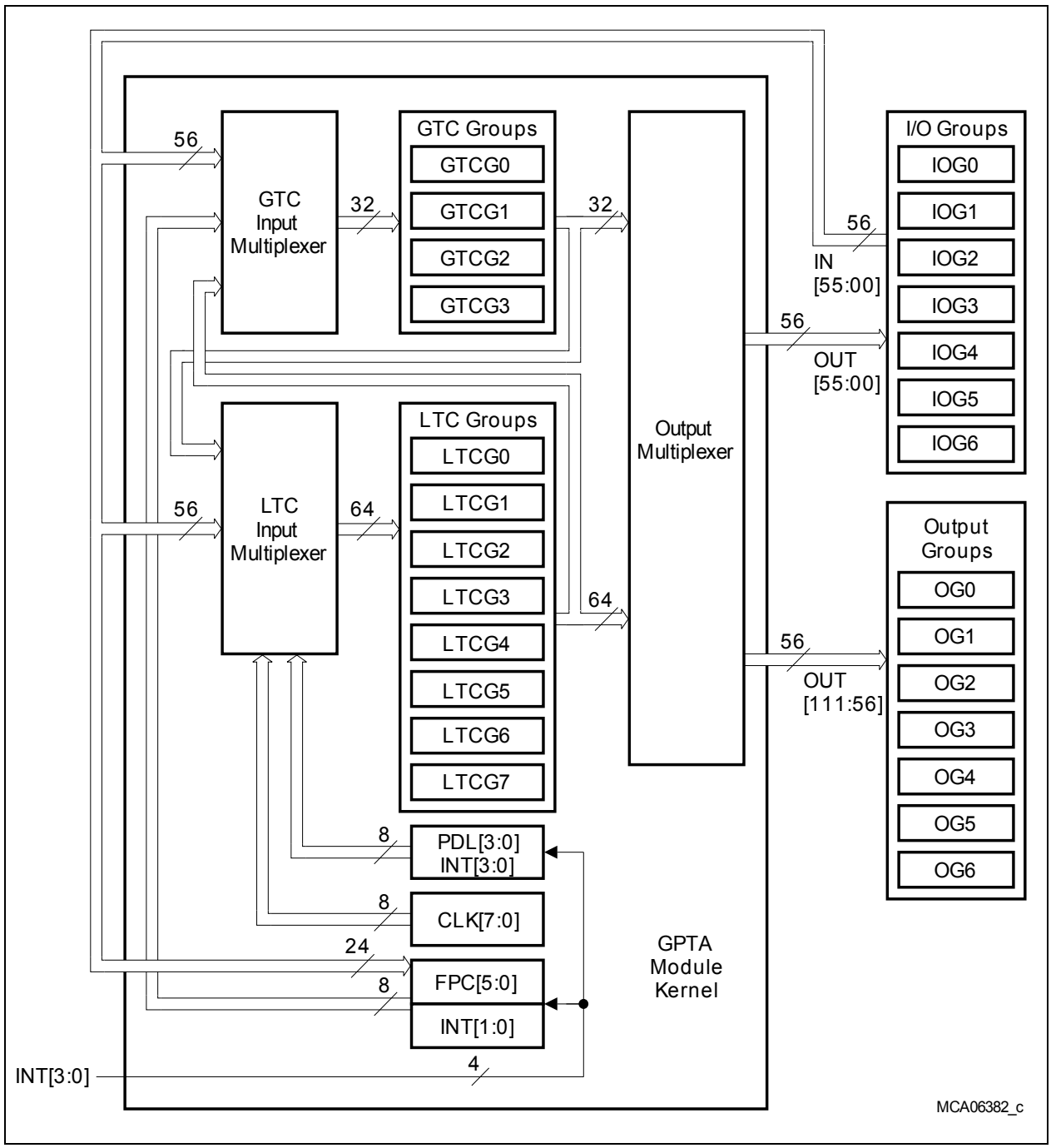


Figure 22-59 Input/Output Line Sharing Unit Overview

The I/O Line Sharing Unit does the following selections:

- FPC input line selection
- GTC and LTC output multiplexer 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, seven output groups, four GTC groups, eight LTC groups, one clock group, one FPC/INT group, and one PDL/INT group are defined.

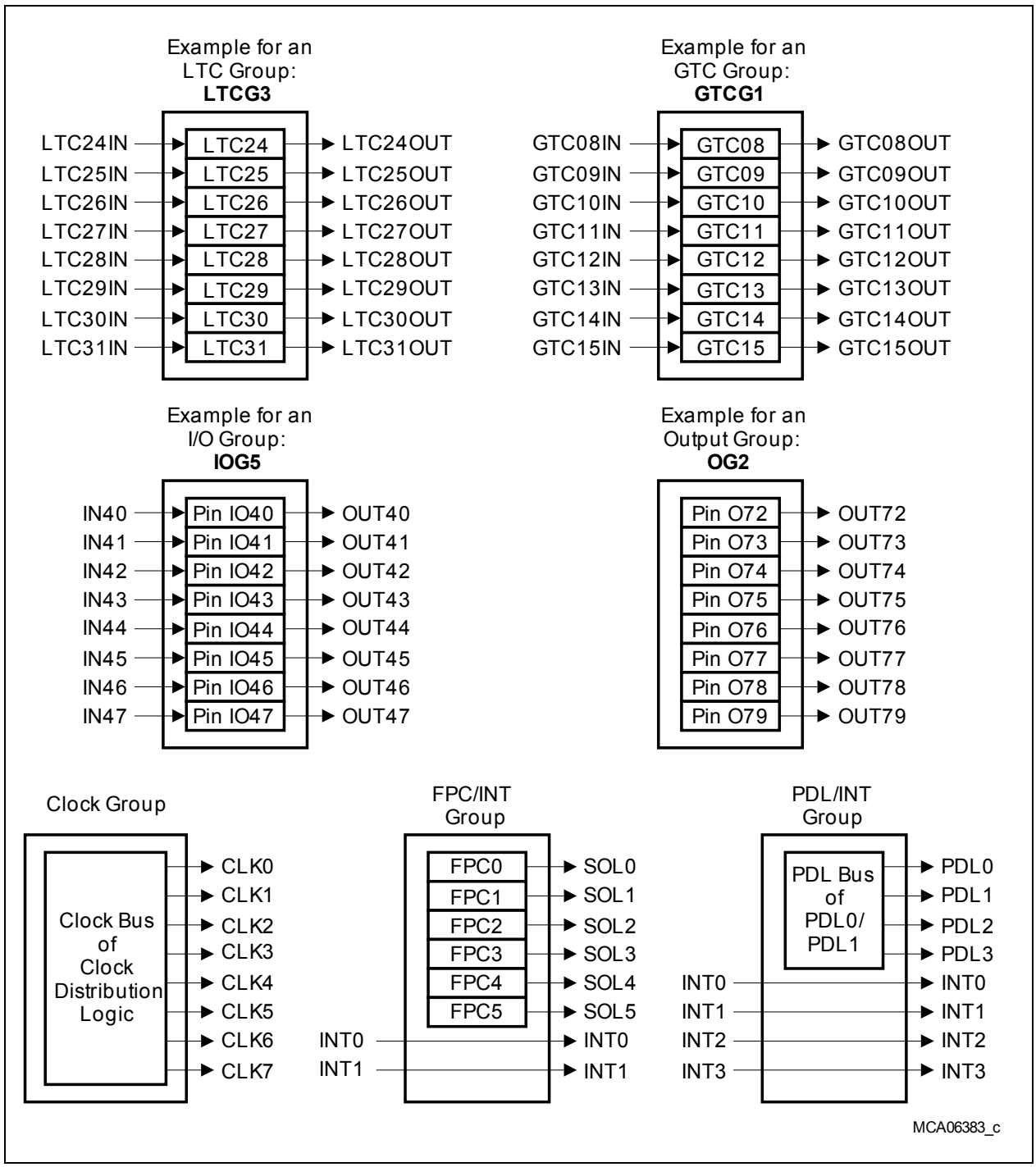


Figure 22-60 Groups Definitions for I/O Line Sharing Unit

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 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 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 unit.

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 module.

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 module.

Table 22-8 Group to I/O Line/Cell Assignment

Group/Module	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
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]

Table 22-8 Group to I/O Line/Cell Assignment (cont'd)

Group/Module	Cell/Line	Input	Output
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]
Clock Group			
–	–	–	CLK[7:0]
FPC/INT Groups			
FPC[5:0]	–	–	SOT[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]

22.2.4.1 FPC Input Line Selection

As shown on **Page 22-9**, each FPC cell can be connected to one out of four input lines SINk[3:0], to the GPTA 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 22-9** shows the FPC port pin connections.

Table 22-9 FPC Input Line Assignments

FPC Control Register	Bit Field IPS	Selected Input Pin
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

22.2.4.2 GTC and LTC Output Multiplexer Selection

The output multiplexer shown in **Figure 22-59** and **Figure 22-61** below connects the 32 GTC output lines and the 64 LTC output lines with the I/O groups ($7 \times 8 = 56$ input/output lines) and the output groups ($7 \times 8 = 56$ output lines).

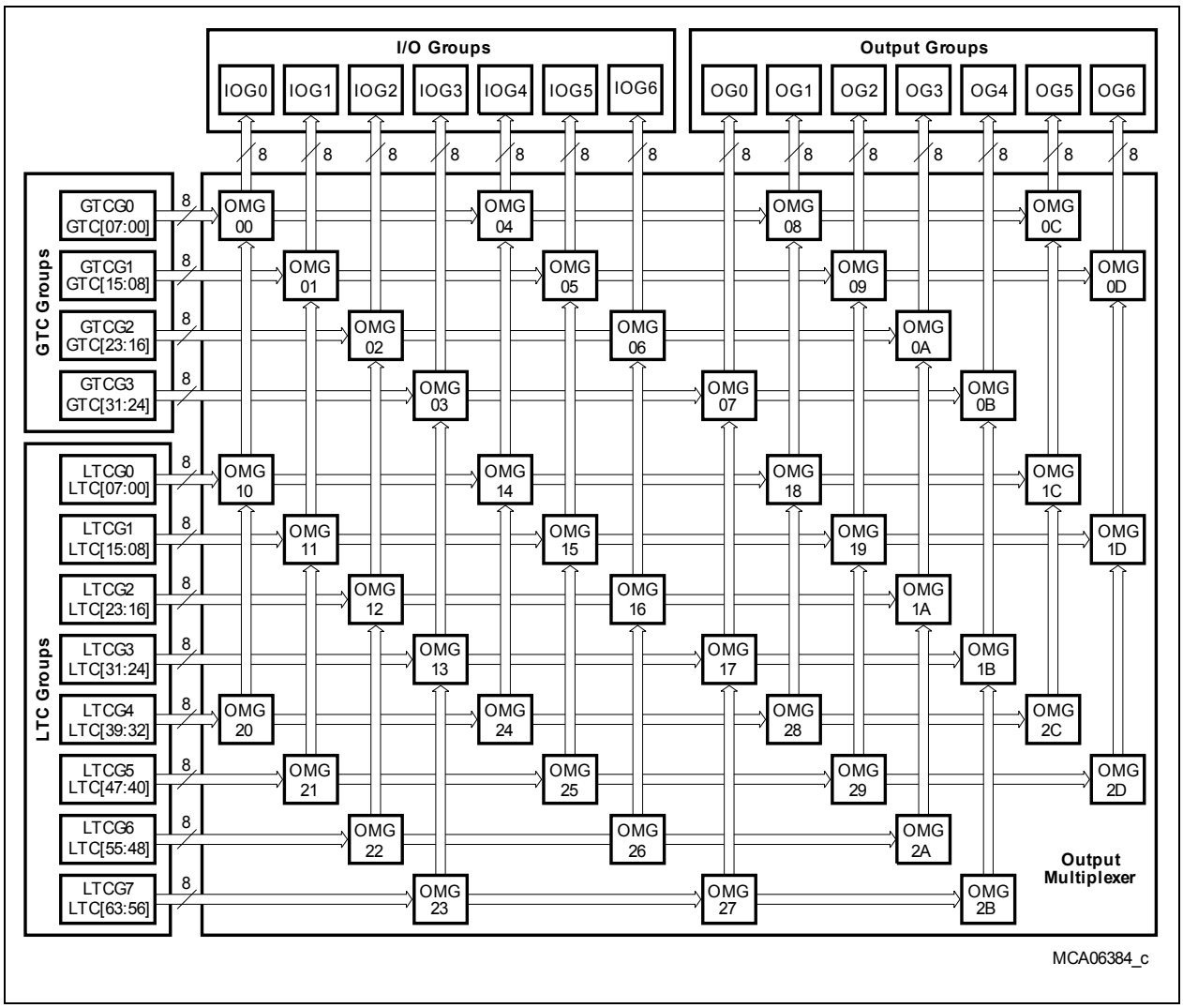


Figure 22-61 Output Multiplexer

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. IOG0 and OG0 share the same physical pins, similarly for IOG1 and OG1, IOG2 and OG2. IOG3 and IOG6 share the same physical pins for inputs and outputs.

Figure 22-62 shows the logical structure of an OMG.

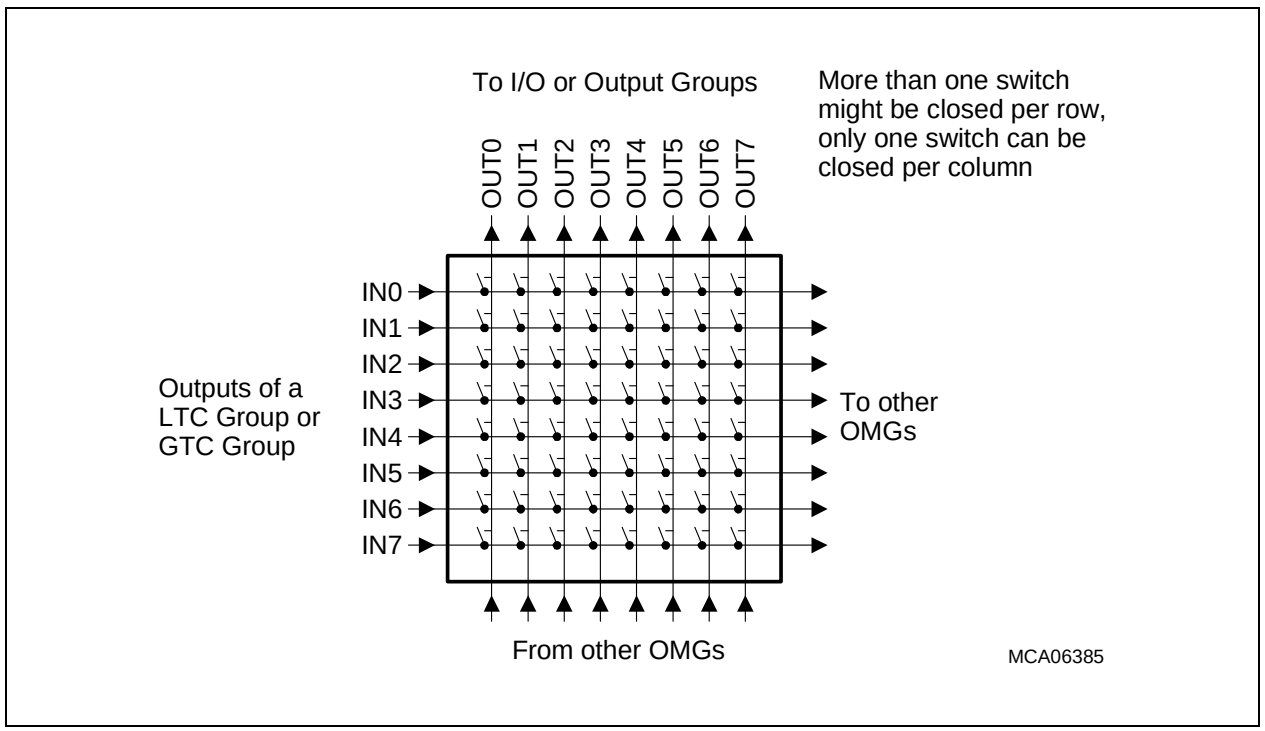


Figure 22-62 Output Multiplexer Group (OMG) Structure

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 22-61), 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 22-61), 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 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. 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 22-63**. With this logic, three GTC or LTC group signals are always combined to one output line that leads to the input of an I/O or output group. For example, when looking at **Figure 22-61**, 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).

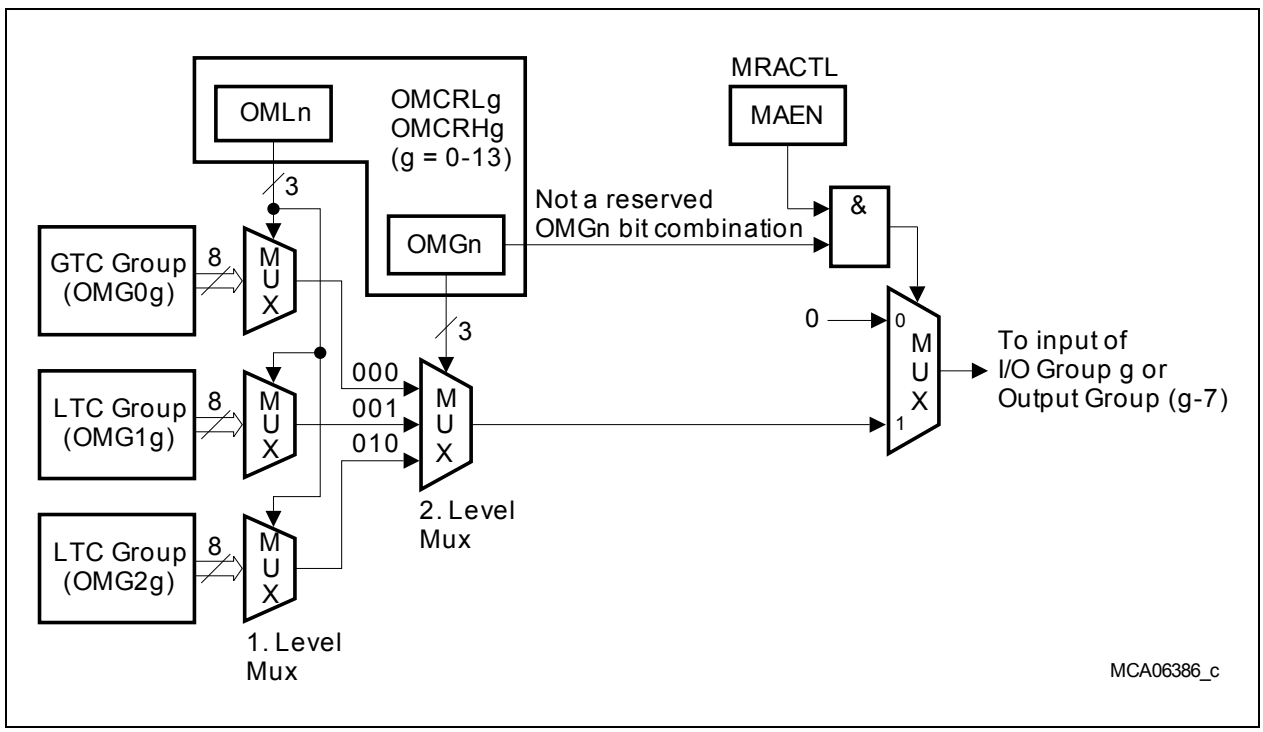


Figure 22-63 Output Multiplexer Group (Programmer's View)

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 Multiplexer Control Registers, OMCRL and OMCRH (see also Page 22-191), 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 module.

The OMCRL registers control the OMG output lines 0 to 3. The OMCRH registers control the OMG output lines 4 to 7. Table 22-10 lists all Output Multiplexer Control Registers with its control functions. Please note that Output Multiplexer Control Registers are not directly accessible but must be written or read using a FIFO array structure as described on Page 22-110.

Table 22-10 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	

Table 22-10 Output Multiplexer Control Register Assignments (cont'd)

I/O Group or Output Group		Controlled by Multiplexer Control Register	Selectable Groups via OMGng
OG4	OUT[91:88]	OMCRL11	GTCG3, LTCG3, LTCG7
	OUT[95:92]	OMCRH11	
OG5	OUT[99:96]	OMCRL12	GTCG0, LTCG0, LTCG4
	OUT[103:100]	OMCRH12	
OG6	OUT[107:104]	OMCRL13	GTCG1, LTCG1, LTCG5
	OUT[111:108]	OMCRH13	

22.2.4.3 GTC Input Multiplexer Selection

The GTC input multiplexer as shown in **Figure 22-59** and **Figure 22-64** 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.

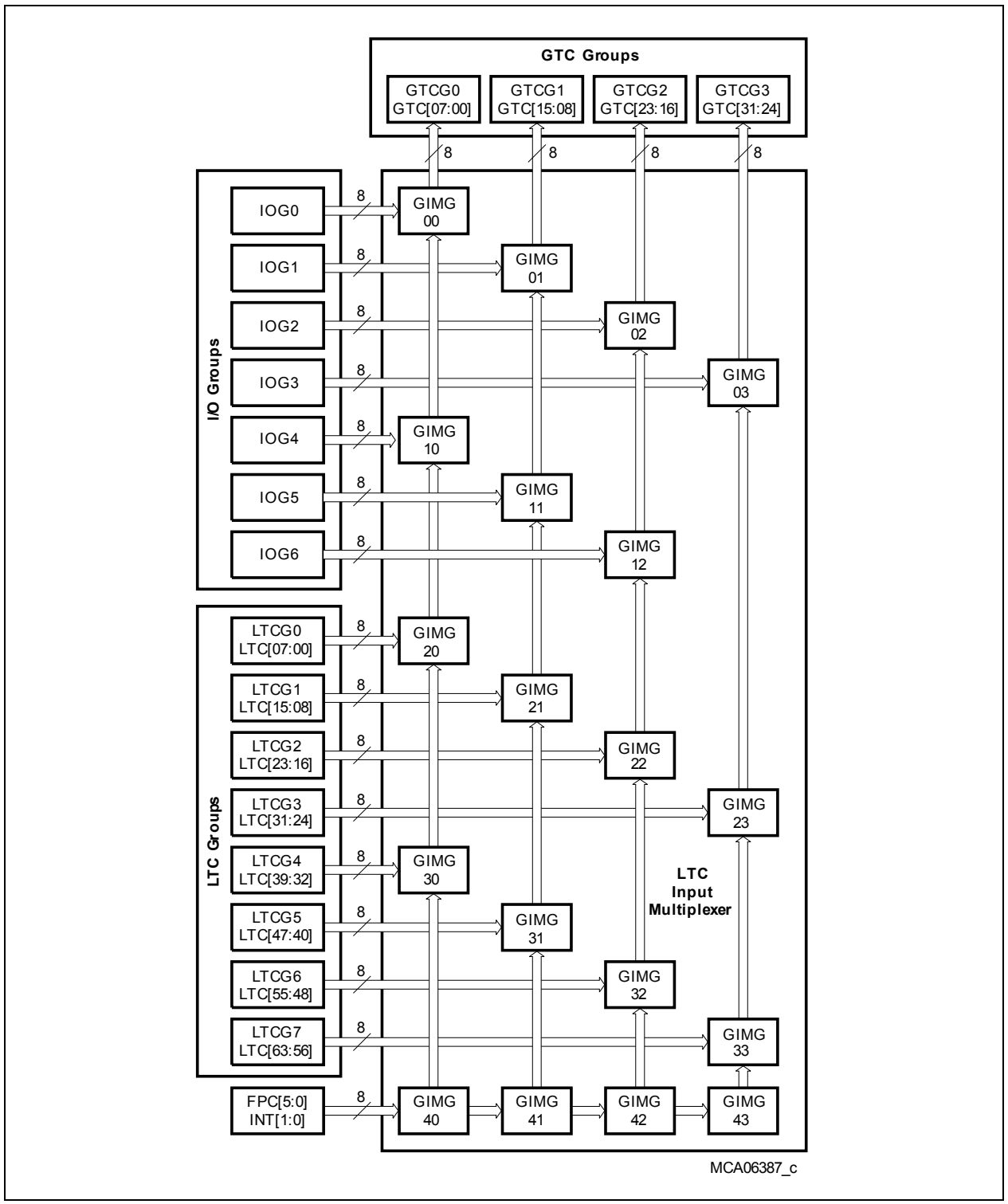


Figure 22-64 GTC Input Multiplexer

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 seven blocks of 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 22-65 shows the logical structure of a GIMG.

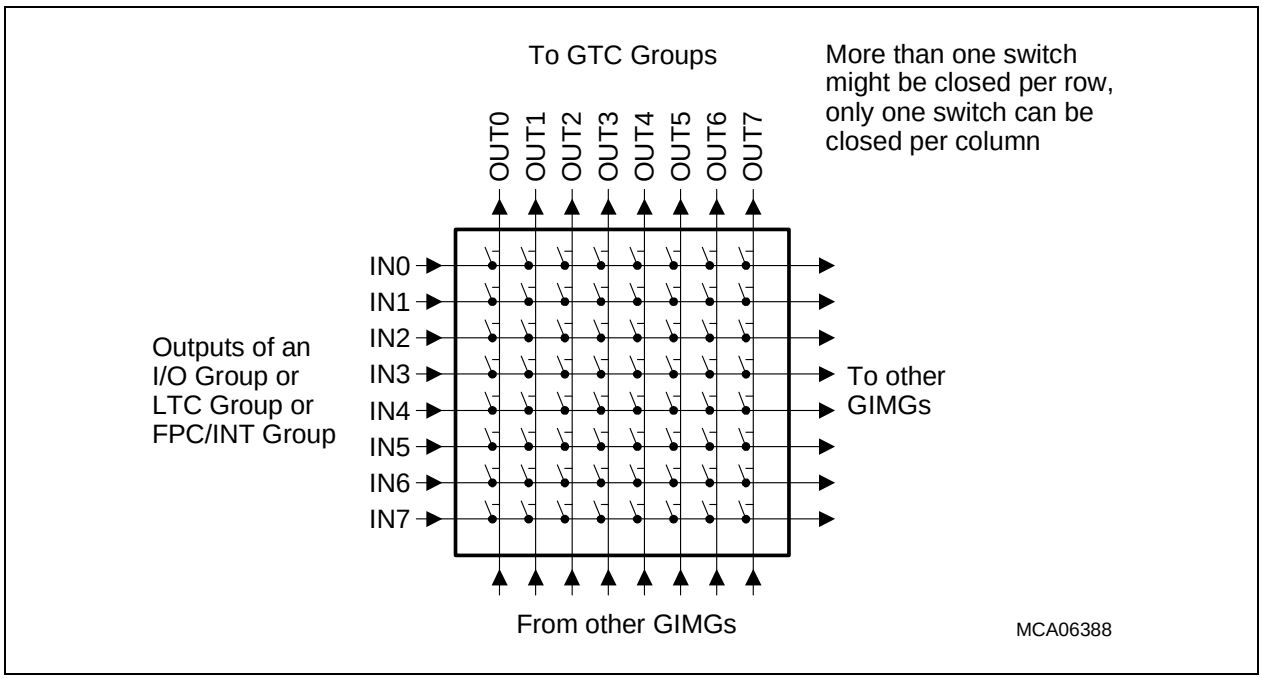


Figure 22-65 GTC Input Multiplexer Group (GIMG) Structure

Rules for connections to GTC Input Multiplexer Group GIMG:

- Within an 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 22-64**), 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 the remaining output lines OUT1 to OUT7 are connected to the GTC inputs with ascending index.
  Example: for GIMG23 (see **Figure 22-64**), 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 is 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 22-66**. 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, based on **Figure 22-64**, each of the eight GTC input multiplexer output lines to GTC group GTCG2 is connected via five GIMGn2 (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.

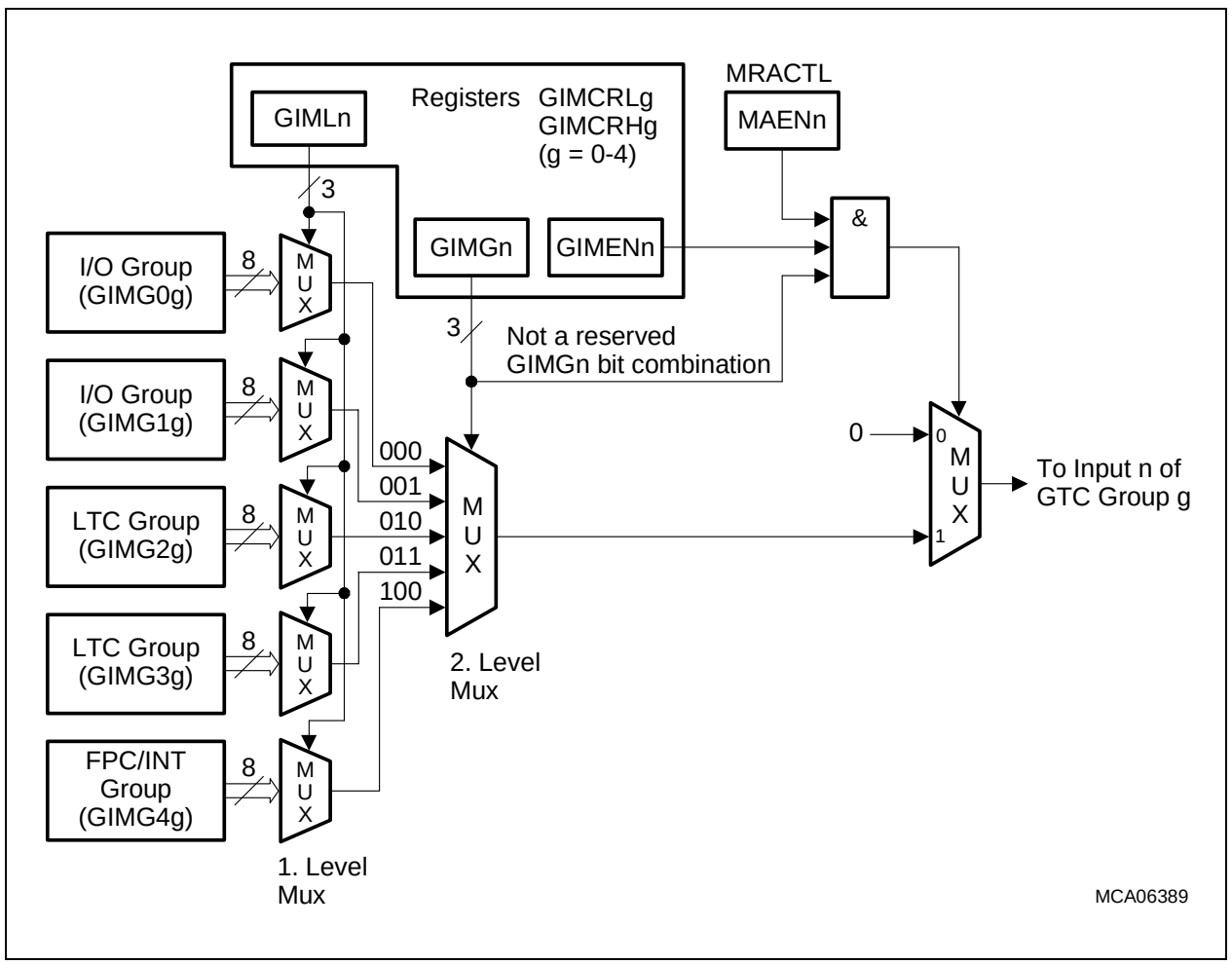


Figure 22-66 GTC Input Multiplexer Group (Programmer's View)

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 a 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 22-195**), 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 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 22-11** 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 22-110**.

Table 22-11 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	

22.2.4.4 LTC Input Multiplexer Selection

The LTC input multiplexer as shown in **Figure 22-59** and **Figure 22-67** 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.

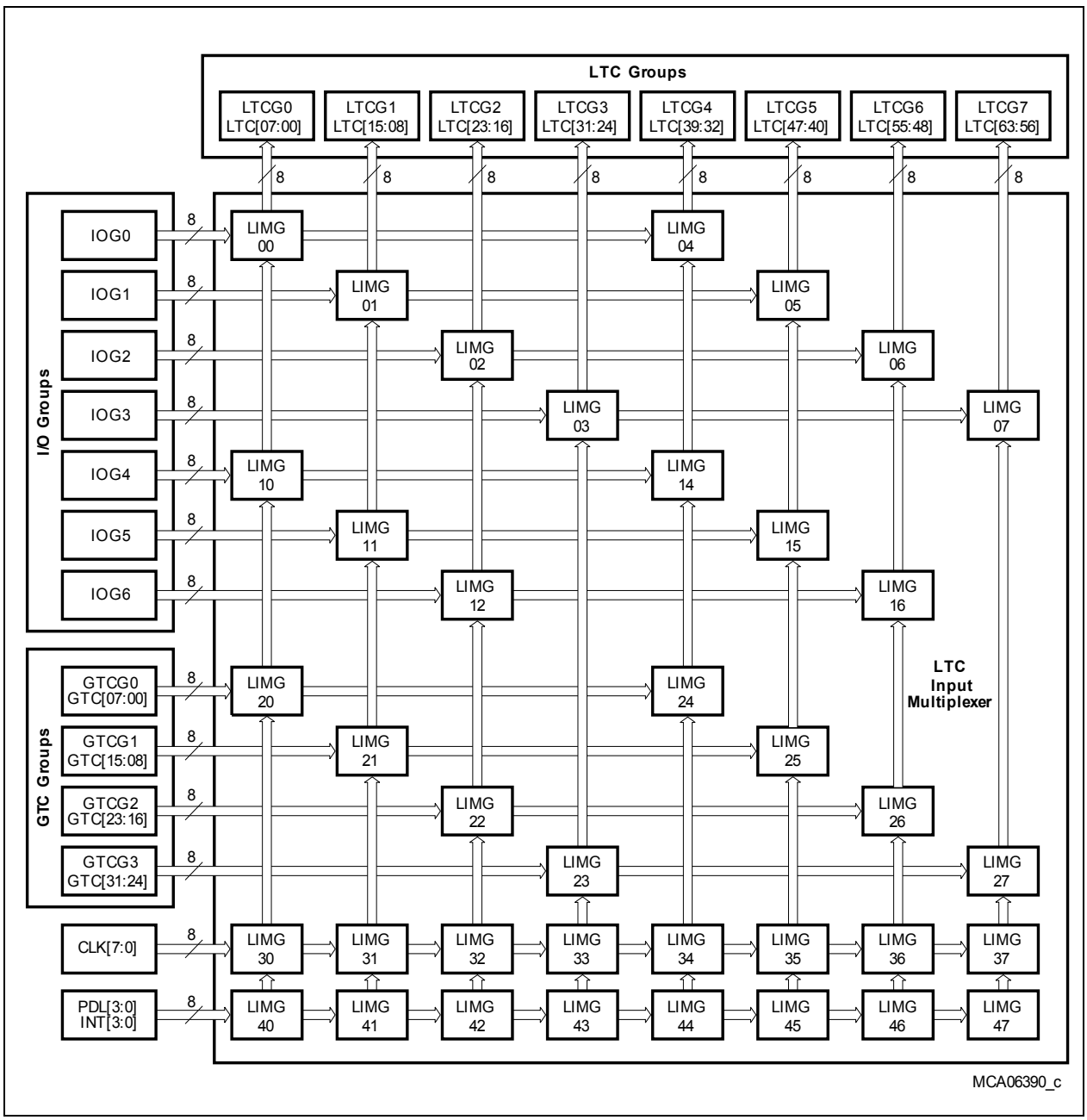


Figure 22-67 LTC Input Multiplexer

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 seven blocks of 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 unit 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.

Figure 22-68 shows the logical structure of an LIMG.

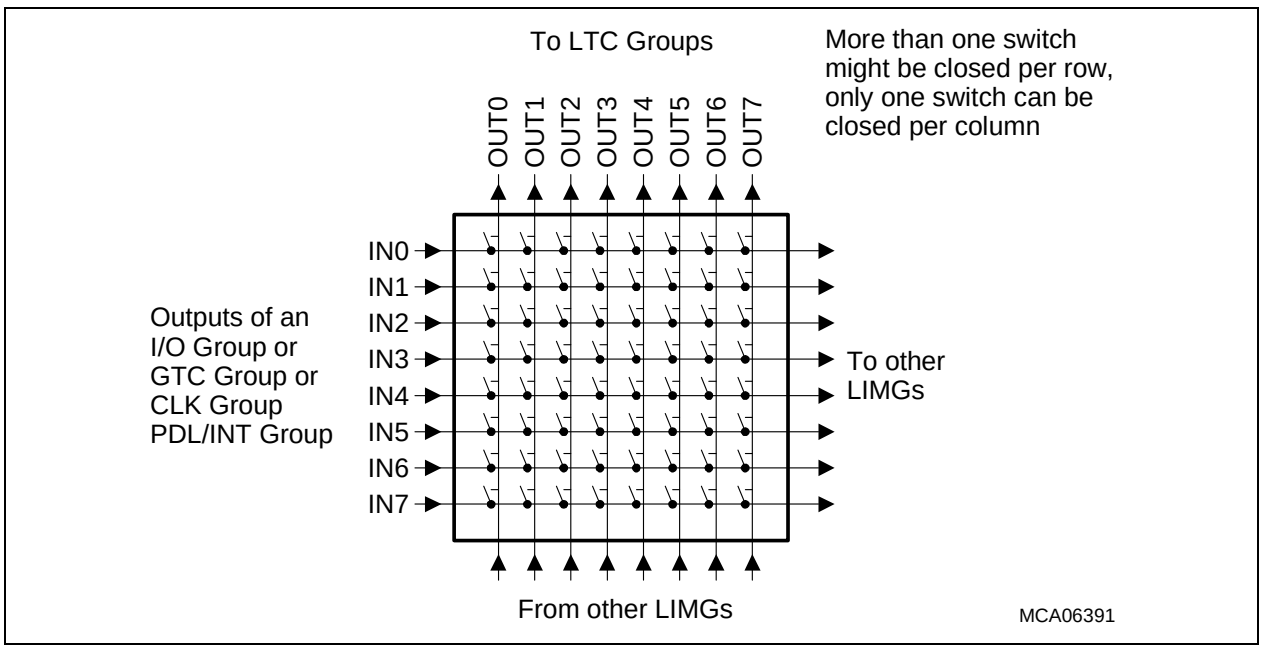


Figure 22-68 LTC Input Multiplexer Group (LIMG) Structure

Rules for connections to LTC Input Multiplexer Group LIMG:

- Within an 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 22-19**) is connected to IN[3:0] and INT[3:0] is connected to IN[7:4].
  Example: for LIMG23 (see **Figure 22-67**), 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 the remaining output lines OUT1 to OUT7 are connected to the LTC inputs with ascending index.
  Example: for LIMG23 (see **Figure 22-67**), 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 22-69**. 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, based on **Figure 22-67**, 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.

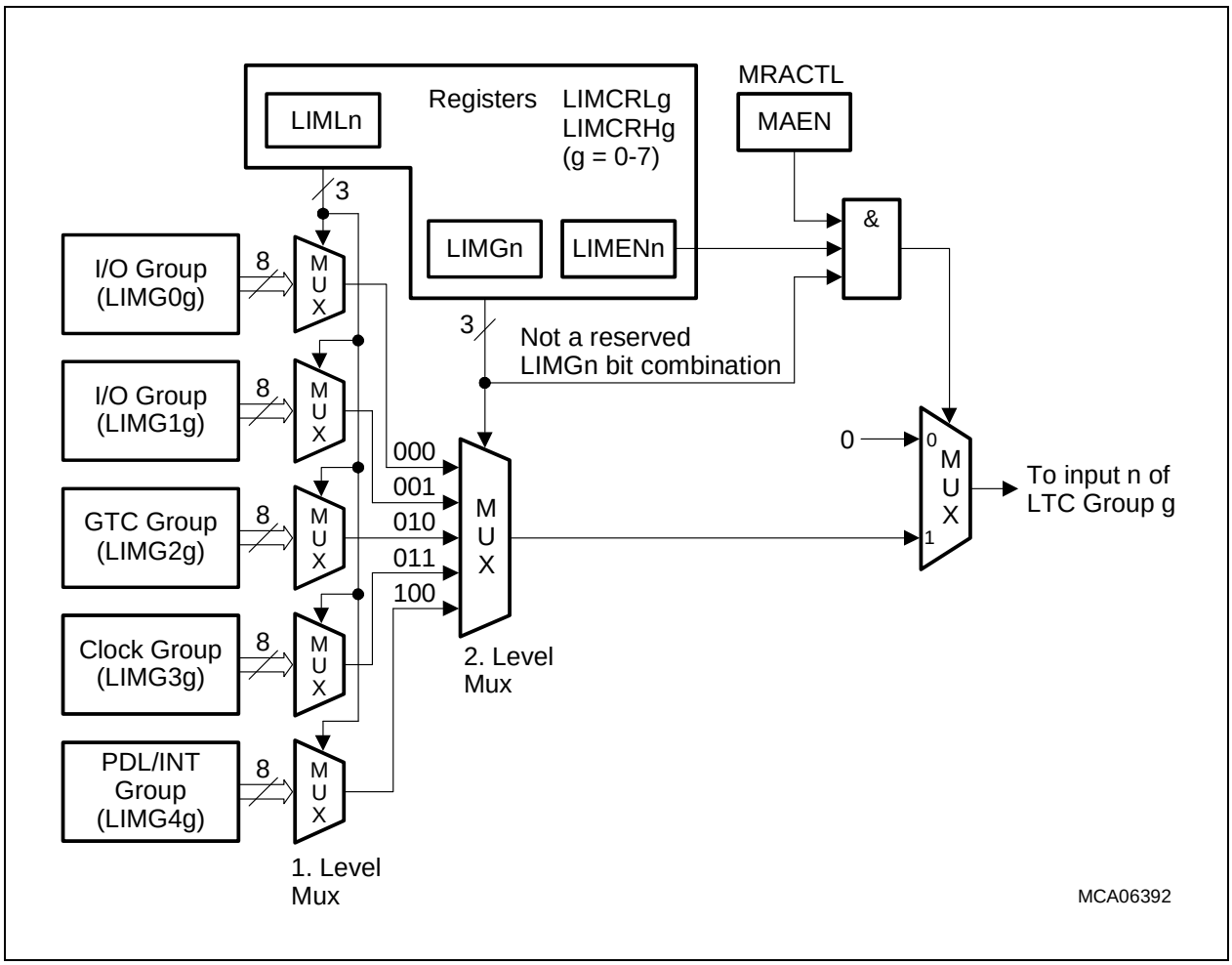


Figure 22-69 LTC Input Multiplexer Group (Programmer's View)

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 22-199**), are assigned to each of the LTC groups. Therefore, a total of sixteen registers control the connections within the LTC input multiplexer of the GPTA 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 22-12** 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 22-110**.

Table 22-12 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, IOG3, GTCG2, CLOCK, PDL/INT
	LTC[55:52]	LIMCRH6	
LTCG7	LTC[59:56]	LIMCRL7	IOG3, GTCG3, CLOCK, PDL/INT
	LTC[63:60]	LIMCRH7	

22.2.4.5 Multiplexer Register Array Programming

A total of 52 control registers are required to program the configuration of the output multiplexer and the two input multiplexers of the Input/Output Line Sharing Unit. These IOLS control registers are combined into a Multiplexer Register Array FIFO that can be 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 22-70 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 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 (52 values) into MRADIN, starting with the register values for OMCRH13, OMCRL13, … up to GIMCRH0, GIMCRL0 (see **Figure 22-70**). After the first MRADIN write operation, the content for OMCRH13 is at FIFO position 1. With each following MRADIN write operation, it becomes shifted one FIFO position upwards. After the 52. MRADIN write operation, the OMCRH13 value is at its final position. The contents of FIFO position 52 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 OMCRH13, OMCRL13, …, GIMCRH0, GIMCRL0.

Before disabling the multiplexer array FIFO, GPTA output pins that are already enabled as GPTA output should be switched to GPIO function to avoid output spikes. After enabling the multiplexer array FIFO again, the GPTA output can be switched again back to GPTA 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.

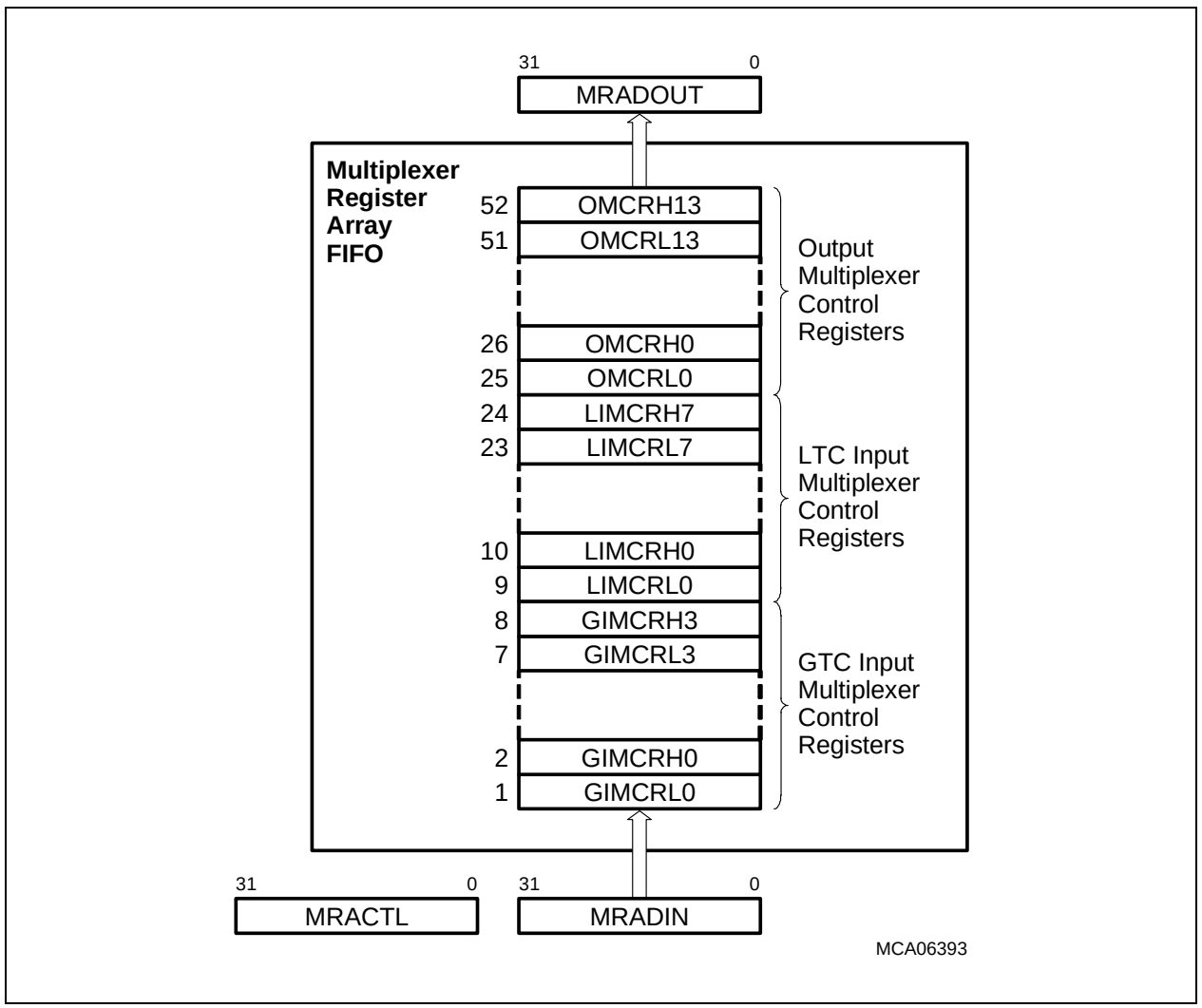


Figure 22-70 GPTA Multiplexer Array Control Register FIFO Structure

22.2.5 Interrupt Sharing Unit (IS)

The GPTA provides 111 service request sources. These service request sources are generated by different cell types, as shown in **Table 22-13**.

Table 22-13 GPTA 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 kernel with a standard interrupt node y and controlled by its SRCy register.

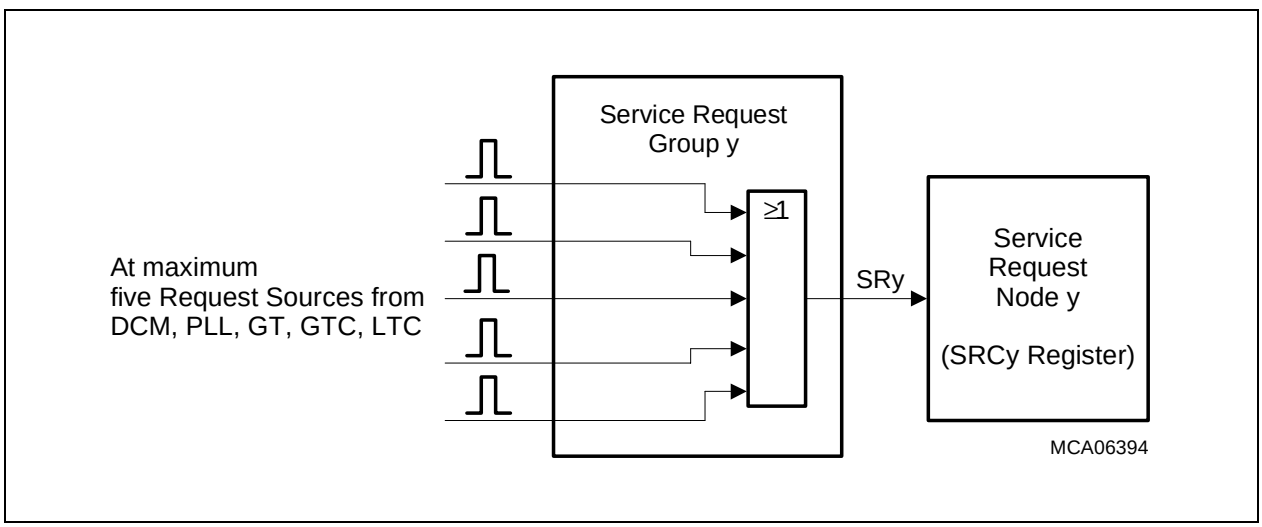


Figure 22-71 Service Request Groups

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 service request source has its own service request flag. This flag is normally set by hardware but can be set and cleared 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 module; but the service request line to the corresponding service request group becomes only active if the corresponding service request is enabled by the related module. Finally, each service request group y must be enabled by the enable flag that is located in SRC register y.

Table 22-14 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 22-14 GPTA 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)]	–	–	–

Table 22-14 GPTA 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
17	GTC22	GTC23[1)]	–	–	–
18	GTC24	GTC25[1)]	–	–	–
19	GTC26	GTC27[1)]	–	–	–
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).

22.2.6 Pseudo Code Description of GPTA Kernel Functionality

This section describes the functional algorithms of the GPTA units in a pseudo code language.

22.2.6.1 FPC Algorithm

FPCk_Control_Logic() – “to be performed every GPTA 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
```

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
```

Delayed_Filter()

```
if (FPCk.Filter_Clock[n]) then
  if (FPCk.Timer >= FPCk.Compare_Value) then
    if (FPCk.Compare_Value == 0) then    //bypass
      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
```

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

22.2.6.2 PDL-Algorithm

PDLx_Control_Logic() – “to be performed every GPTA 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	Makes it possible for the software to read PDL error

22.2.6.3 DCM-Algorithm

DCMk_Control_Logic() – “to be performed every GPTA 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

Name k = [0 to 3] for DCM	Short Name (*)DCM	Size (bits)	Function
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

22.2.6.4 PLL-Algorithm

PLL_Control_Logic() – “to be performed every GPTA 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	Allows 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

22.2.6.5 GT-Algorithm

GTm_Control_Logic() – “to be performed every GPTA 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

22.2.6.6 GTC-Algorithm

GTCk_Control_Logic() – “to be performed every GPTA 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
```

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
```

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
```

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

22.2.6.7 LTC-Algorithm for Cells 0 to 62

LTCk_Control_Logic() – “to be performed every GPTA 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]
  else
    if (LTCk.Output_Control_Mode.2) then    //bypass, input link enabled
      LTCk.Output_Mode_Out = LTCk.Output_Mode_In
    else    //bypass, input link disabled
      LTCk.Output_Mode_Out = 00B
    endif
  endif
else    //no local event
  if (LTCk.Output_Control_Mode.2)    //input link enabled
    Set_Data_Out(LTCk.Output_Mode_In)
    LTCk.Output_Mode_Out = LTCk.Output_Mode_In
  else    //input link disabled
    Set_Data_Out(00B)
    LTCk.Output_Mode_Out = 00B
  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.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	Flipflop 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.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.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

22.2.6.8 LTC Algorithm for Cell 63

LTC63_Control_Logic() “to be performed every GPTA 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

22.2.7 Programming of a GPTA Module

A hierarchical top-down design approach may be used to implement a complex signal processing unit as follows:

- Partitioning the complex signal processing unit into simple function units,
- Implementing each simple function unit 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 22-15 summarizes all of the software tasks to be implemented for getting a GPTA unit into operation.

Table 22-15 Software Tasks Controlling a GPTA Unit

GPTA Shell Initialization	
GPTA Module Clock Enable	
Fractional Divider Setting	
Module Enable	
Configuration of Interrupt Handling	
GPTA 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 Capcom Register Update	Interrupt Request Enable on End of Output Pulse Generation
Interrupt Request Enable on Input Edge or Compare Event	

Table 22-15 Software Tasks Controlling a GPTA 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
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)	

22.3 GPTA0 Kernel Registers

This section describes the kernel registers of the GPTA0 module.

GPTA Kernel Register Overview

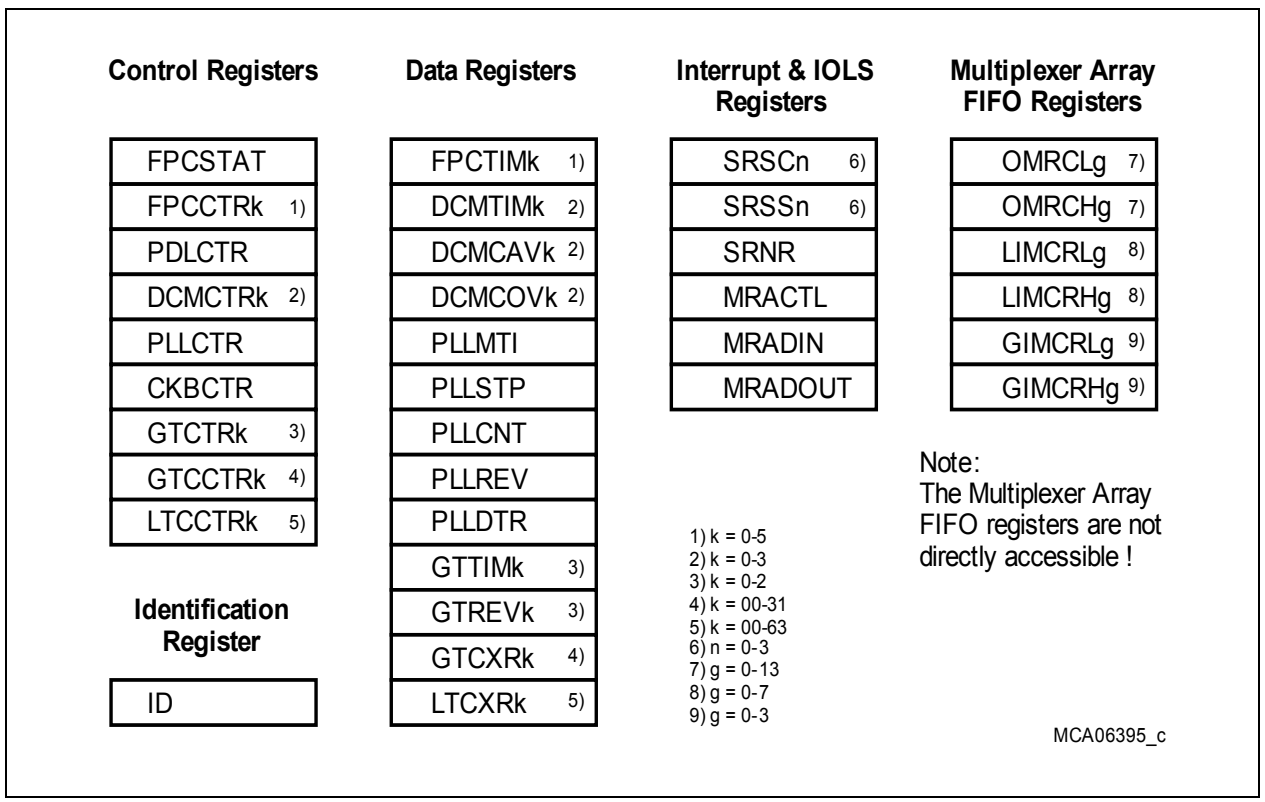


Figure 22-72 GPTA0 Kernel Registers

The complete and detailed address map of the of the GPTA0 module is described in **Table 16-15** on **Page 16-26** of the TC1766 User's Manual System Units part (Volume 1).

Table 22-16 Registers Address Space

Module	Base Address	End Address	Note
GPTA0	F000 1800$_H$	F000 1FFF$_H$	-

Table 22-17 Registers Overview - GPTA0 Kernel Registers

Register Short Name	Register Long Name	Offset Address[1)]	Description see
ID	Module Identification Register	008_H	Page 22-154
SRSCn	Service Request State Clear Register n (n = 0-3)	$010_H + n \times 8$	Page 22-203 Page 22-204 Page 22-205
SRSSn	Service Request State Set Register n (n = 0-3)	$0010_H + n \times 8 + 4$	Page 22-207 Page 22-208 Page 22-209
SRNR	Service Request Node Redirection Register	0030_H	Page 22-211
MRACTL	Multiplexer Register Array Control Register	0038_H	Page 22-188
MRADIN	Multiplexer Register Array Data In Register	$003C_H$	Page 22-190
MRADOUT	Multiplexer Register Array Data Out Register	0040_H	Page 22-190
FPCSTAT	Filter and Prescaler Cell Status Register	0044_H	Page 22-155
FPCCTRk	Filter and Prescaler Cell Control Register k (k = 0-5)	$0048_H + k \times 8$	Page 22-156
FPCTIMk	Filter and Prescaler Cell Timer Register k (k = 0-5)	$0048_H + k \times 8 + 4$	Page 22-158
PDLCTR	Phase Discrimination Logic Control Register	0078_H	Page 22-159
DCMCTRk	Duty Cycle Measurement Control Register k (k = 0-3)	$0080_H + k \times 16$	Page 22-161
DCMTIMk	Duty Cycle Measurement Timer Register k (k = 0-3)	$0080_H + k \times 16 + 4$	Page 22-162
DCMCAVk	Duty Cycle Measurement Capture Register k (k = 0-3)	$0080_H + k \times 16 + 8$	Page 22-163
DCMCOVk	Duty Cycle Measurement Capture/Compare Register k (k = 0-3)	$0080_H + k \times 16 + 12$	Page 22-163
PLLCTR	Phase Locked Loop Control Register	$00C0_H$	Page 22-164
PLLMTI	Phase Locked Loop Micro Tick Register	$00C4_H$	Page 22-165
PLLCNT	Phase Locked Loop Counter Register	$00C8_H$	Page 22-166
PLLSTP	Phase Locked Loop Step Register	$00CC_H$	Page 22-166

Table 22-17 Registers Overview - GPTA0 Kernel Registers (cont'd)

Register Short Name	Register Long Name	Offset Address[1)]	Description see
PLLREV	Phase Locked Loop Reload Register	$00D0_H$	Page 22-167
PLLDTR	Phase Locked Loop Delta Register	$00D4_H$	Page 22-167
CKBCTR	Clock Bus Control Register	$00D8_H$	Page 22-168
GTCTRk	Global Timer Control Register k (k = 0-1)	$00E0_H$ + k × 16	Page 22-170
GTREVk	Global Timer Reload Value Register k (k = 0-1)	$00E0_H$ + k × 16 + 4	Page 22-171
GTTIMk	Global Timer Register k (k = 0-1)	$00E0_H$ + k × 16 + 8	Page 22-171
GTCCTRk	Global Timer Cell Control Register k (k = 00-31)	0100_H + k × 8	Page 22-172
GTCXRk	Global Timer Cell X Register k (k = 00-31)	0100_H + k × 8 + 4	Page 22-176
LTCCTRk	Local Timer Cell Control Register k (k = 00-62)	0200_H + k × 8	Page 22-182
LTCXRk	Local Timer Cell X Register k (k = 00-62)	0200_H + k × 8 + 4	Page 22-186
LTCCTR63	Local Timer Cell Control Register 63	$03F8_H$	Page 22-185
LTCXR63	Local Timer Cell X Register 63	$03FC_H$	Page 22-187
OMCRLg	Output Multiplexer Control Register for Lower Half of Group g (g = 0-13)	not directly addressable; see Page 22-110	Page 22-191
OMCRHg	Output Multiplexer Control Register for Upper Half of Group g (g = 0-13)		Page 22-193
GIMCRLg	Input Multiplexer Control Register for Lower Half of GTC Group g (g = 0-3)		Page 22-195
GIMCRHg	Input Multiplexer Control Register for Upper Half of GTC Group g (g = 0-3)		Page 22-197
LIMCRLg	Input Multiplexer Control Register for Lower Half of LTC Group g (g = 0-7)		Page 22-199
LIMCRHg	Input Multiplexer Control Register for Upper Half of LTC Group g (g = 0-7)		Page 22-201

1) The absolute register address is calculated as follows:
Module Base Address (Table 22-16) + Offset Address (shown in this column)

Bit Protection

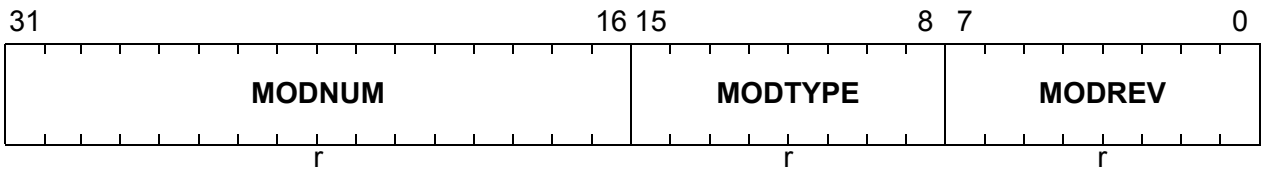
Bits with bit protection (this is valid, for example, for all bits in the Service Request State Registers) are not changed during a read-modify-write instruction, for example when hardware sets a request state bit between the read and the write of the read-modify-write sequence. For bit protected bits 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.

22.3.1 GPTA Module Identification Register

GPTA0_ID
GPTA0 Module Identification Register

(008_H) **Reset Value: $0029\ C0XX_H$**



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 GPTA: 0029_H

22.3.2 FPC Registers

FPCSTAT
Filter and Prescaler Cell Status Register

(044_H) **Reset Value: $0000\ 0000_H$**

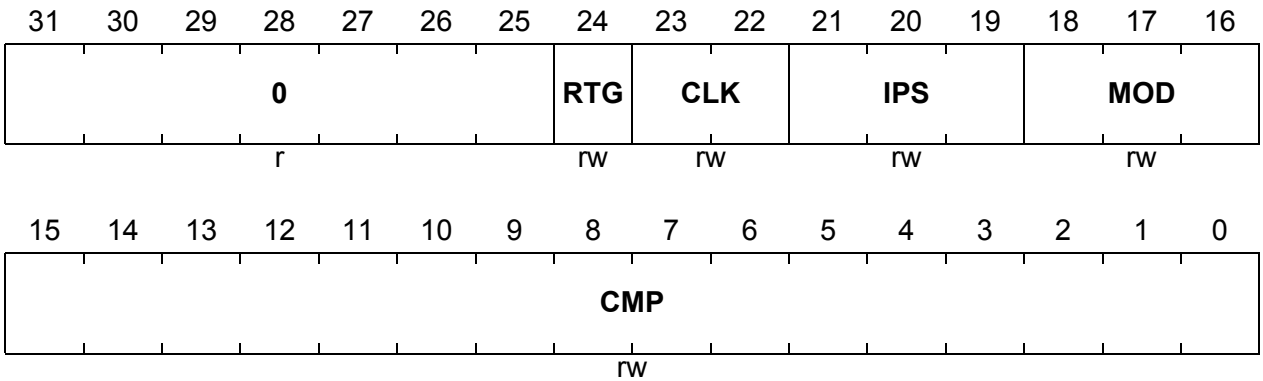
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

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 22.3.2**).
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 22.3.2**).
0	[7:6], [31:14]	r	**Reserved** Read as 0; should be written with 0.

FPCCTRk (k = 0-5)
Filter and Prescaler Cell Control Register k

(048_H+k*8_H) **Reset Value: 0000 0000_H**

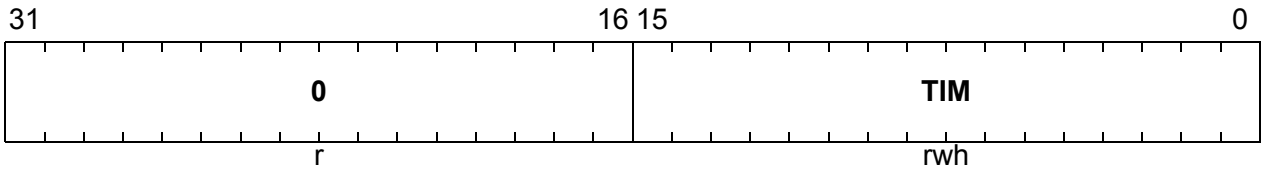


Field	Bits	Type	Description
CMP	[15:0]	rw	**Threshold Value of Filter and Prescaler Cell k** CMP is the 16-bit threshold value that is compared with the 16-bit timer value FPCTIMk.TIM.
MOD	[18:16]	rw	**Operation Mode Selection for FPCk** 000_B Delayed Debounce Filter Mode on both edges 001_B Immediate 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 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 module clock f_{GPTA}) 01_B Clock bus line 1 selected (local PLL clock) 10_B Clock bus line 2 selected (prescaled) GPTA 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.

FPCTIMk (k = 0-5)
Filter and Prescaler Cell Timer Register k

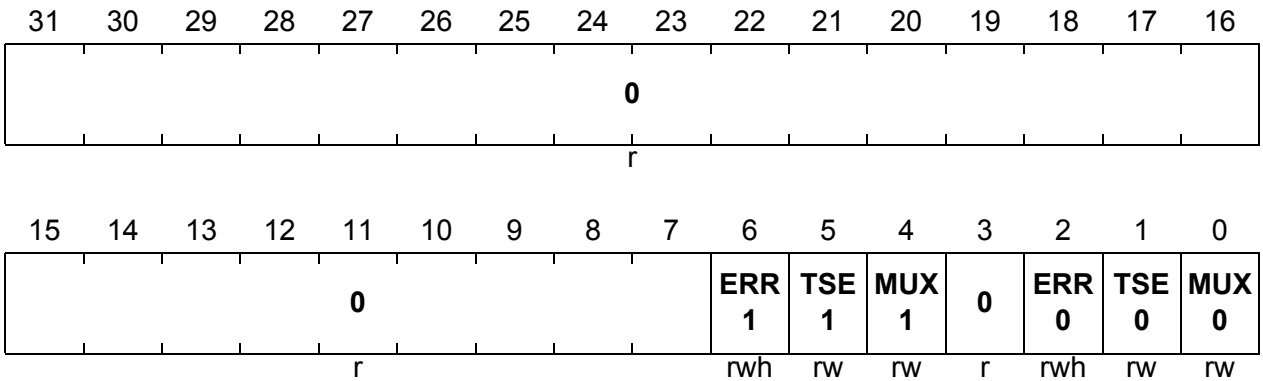
($04C_H$+k*8_H) **Reset Value: 0000 0000_H**



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.

22.3.3 Phase Discriminator Logic Register

PDLCTR
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 22-155**).
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 22-155**).
0	3, [31:7]	r	**Reserved** Read as 0; should be written with 0.

22.3.4 Duty Cycle Measurement Register

DCMCTRk (k = 0-3)
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															

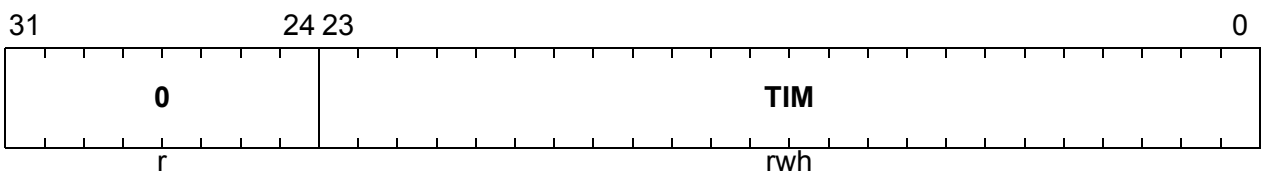
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						CRE	FRE	RRE	QCK	FCK	RCK	FZE	RZE	OCA	RCA
r						rw	rw	rw	w	rw	rw	rw	rw	rw	rw

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.

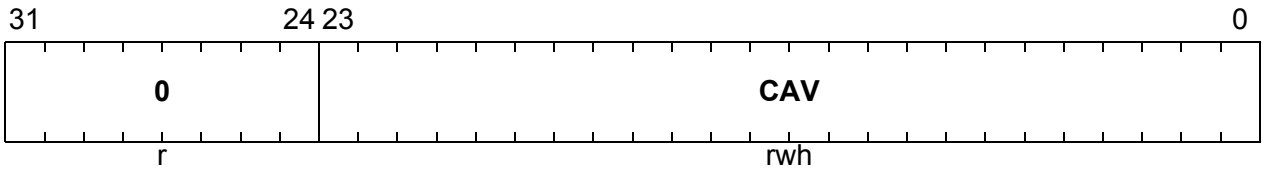
DCMTIMk (k = 0-3)
Duty Cycle Measurement Timer Register k

(084_H+k*10_H) **Reset Value: 0000 0000_H**



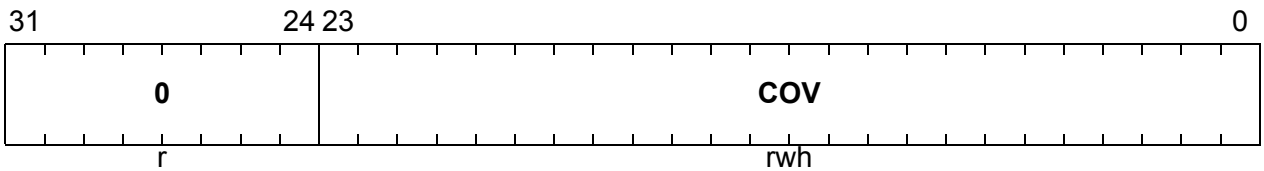
Field	Bits	Type	Description
TIM	[23:0]	rwh	**Timer Value of DCMk**
0	[31:24]	r	**Reserved** Read as 0; should be written with 0.

DCMCAVk (k = 0-3)
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.

DCMCOVk (k = 0-3)
Duty Cycle Measurement Capture/Compare Register k
($08C_H$+k*10_H) **Reset Value: 0000 0000_H**

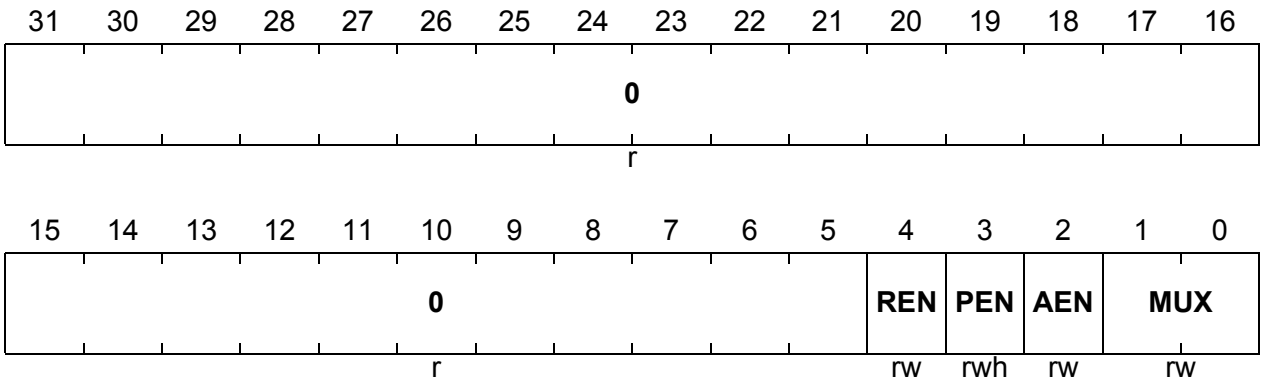


Field	Bits	Type	Description
COV	[23:0]	rwh	**Capture/Compare Register Value of DCMk**
0	[31:24]	r	**Reserved** Read as 0; should be written with 0.

22.3.5 Digital Phase Locked Loop Register

PLLCTR
Phase Locked Loop Control Register

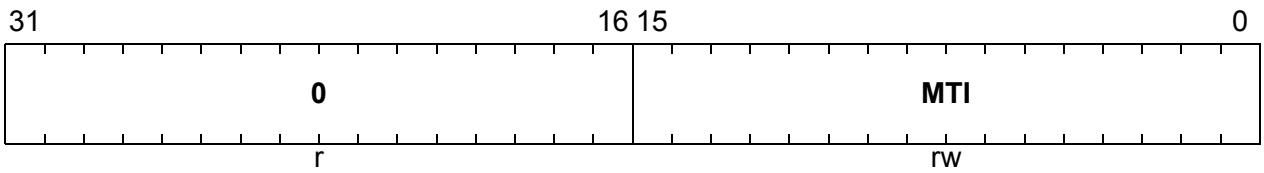
($0C0_H$) **Reset Value: $0000\ 0000_H$**



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	**Compensation of Input Period Length Variation** 0_B Compensation of input signal's period length variation is disabled 1_B Compensation of input signal's period length variation (acceleration, deceleration) is requested
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).

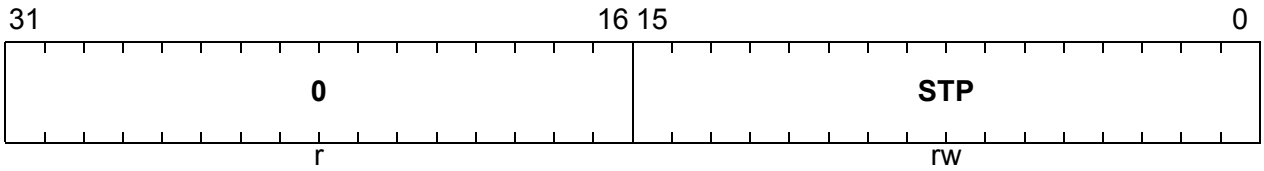
Field	Bits	Type	Description
REN	4	rw	**Interrupt Service Request Enable** 0_B Interrupt request is disabled 1_B An interrupt request is set when the number of remaining output pulses to be generated reaches zero
0	[31:5]	r	**Reserved** Read as 0; should be written with 0.

PLLMTI
Phase Locked Loop Microtick Register($0C4_H$) **Reset Value: $0000\ 0000_H$**



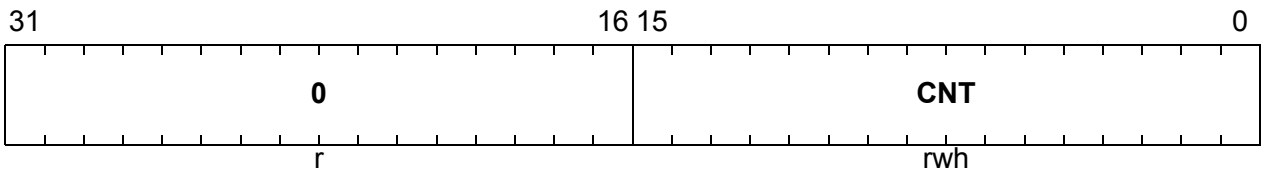
Field	Bits	Type	Description
MTI	[15:0]	rw	**Microtick Value** Number of output pulses to be generated within one input signal period.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

PLLSTP
Phase Locked Loop Step Register ($0CC_H$) Reset Value: $0000\ 0000_H$



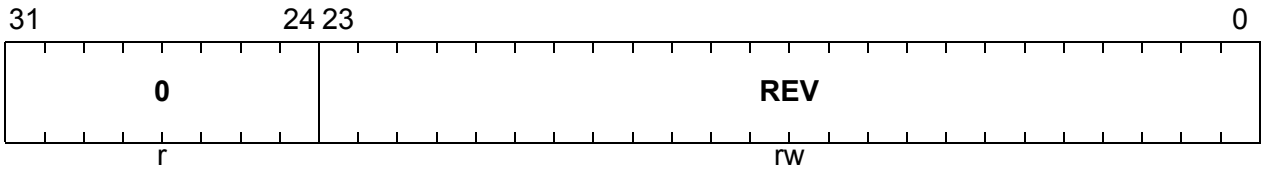
Field	Bits	Type	Description
STP	[15:0]	rw	**Step Value** Number of output pulses to be generated within one input signal period (2-complement data format).
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

PLLCNT
Phase Locked Loop Counter Register($0C8_H$) Reset Value: $0000\ 0000_H$



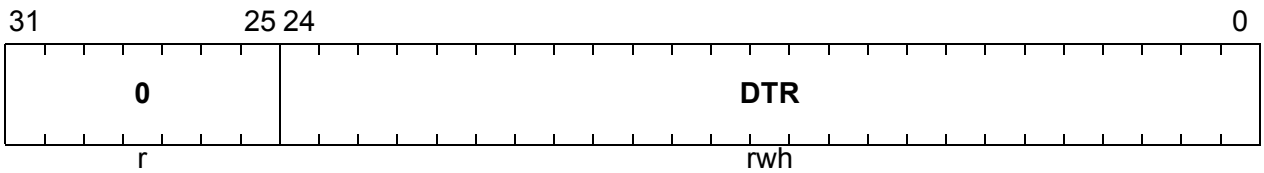
Field	Bits	Type	Description
CNT	[15:0]	rwh	**Pulse Counter** Counter for the number of remaining output pulses to be generated.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

PLLREV
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 module clocks).
0	[31:24]	r	**Reserved** Read as 0; should be written with 0.

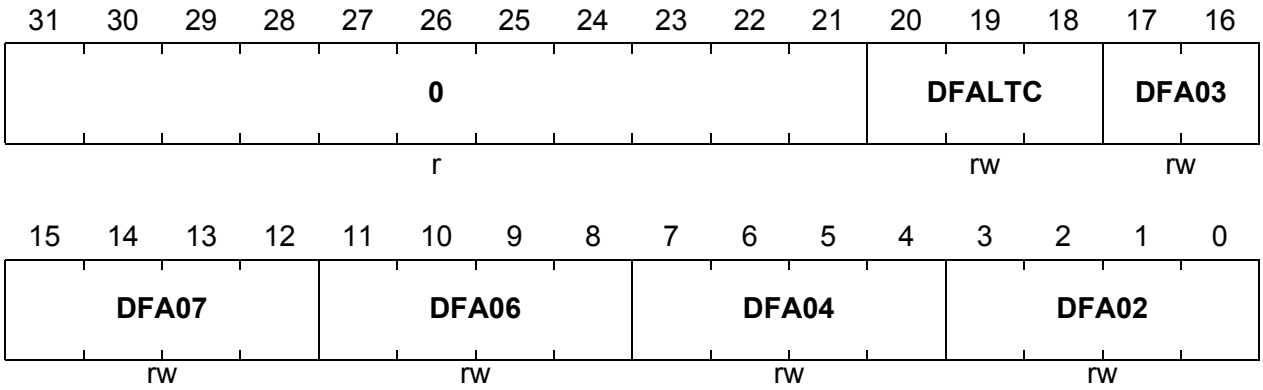
PLLDTR
Phase Locked Loop Delta Register ($0D4_H$) **Reset Value: $0000\ 0000_H$**



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.

22.3.6 Clock Bus Register

CKBCTR
Clock Bus Control Register **($0D8_H$)** **Reset Value: 0000 FFFF$_H$**

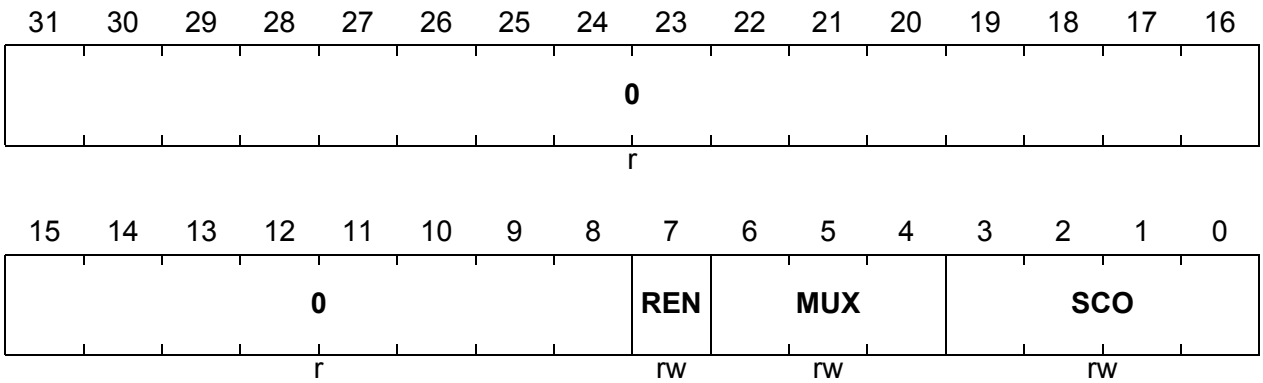


Field	Bits	Type	Description
DFA02	[3:0]	rw	**Clock Line 2 Driving Source Selection** 0_D CLK2 is provided with f_{GPTA} 1_D CLK2 is provided with f_{GPTA} divided by 2^1 …$_D$ … 12_D CLK2 is provided with f_{GPTA} divided by 2^{12} 13_D CLK2 is provided with f_{GPTA} divided by 2^{13} 14_D CLK2 is driven by PLL clock of other GPTA 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 f_{GPTA} 1_D CLK4 is provided with f_{GPTA} divided by 2^1 …$_D$ … 13_D CLK4 is provided with f_{GPTA} divided by 2^{13} 14_D CLK4 is provided with f_{GPTA} divided by 2^{14} 15_D CLK4 is driven by DCM1 output
DFA06	[11:8]	rw	**Clock Line 6 Driving Source Selection** 0_D CLK6 is provided with f_{GPTA} 1_D CLK6 is provided with f_{GPTA} divided by 2^1 …$_D$ … 13_D CLK6 is provided with f_{GPTA} divided by 2^{13} 14_D CLK6 is provided with f_{GPTA} divided by 2^{14} 15_D CLK6 is driven by FPC1 output

Field	Bits	Type	Description
DFA07	[15:12]	rw	**Clock Line 7 Driving Source Selection** 0_D CLK7 is provided with f_{GPTA} 1_D CLK7 is provided with f_{GPTA} divided by 2^1 …$_D$ … 13_D CLK7 is provided with f_{GPTA} divided by 2^{13} 14_D CLK7 is provided with f_{GPTA} divided by 2^{14} 15_D CLK7 is driven by FPC4 output
DFA03	[17:16]	rw	**Clock Line 3 Driving Source Selection** 0_D CLK3 is driven by DCM2 output 1_D CLK3 is driven by PLL clock of other GPTA unit 2_D CLK3 is driven by uncompensated PLL clock 3_D CLK3 is driven by uncompensated PLL clock of other GPTA unit
DFALTC	[20:18]	rw	**Dividing Factor for LTC Prescaler Clock Selection** The LTCPRE clock is provided with the GPTA module clock f_{GPTA} divided by 2^{DFALTC}. 0_D LTCPRE clock is f_{GPTA} 1_D LTCPRE clock is f_{GPTA} divided by 2^1 …$_D$ … 6_D LTCPRE clock is f_{GPTA} divided by 2^6 7_D LTCPRE clock is f_{GPTA} divided by 2^7
0	[31:21]	r	**Reserved** Read as 0; should be written with 0.

22.3.7 Global Timer Registers

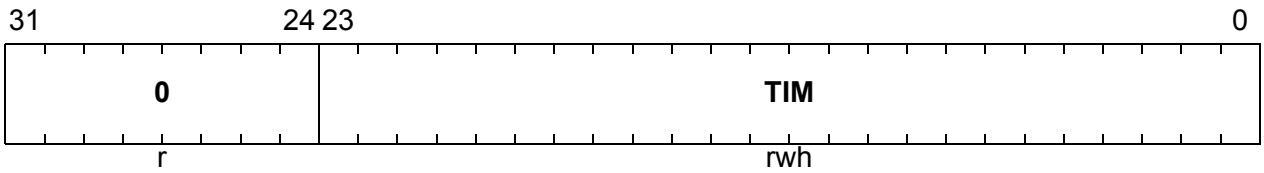
GTCTRk (k = 0-1)
Global Timer Control Register k ($0E0_H$+k*10_H) **Reset Value: 0000 0000_H**



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. $\ldots_B$ … 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
REN	7	rw	**Interrupt Request Enable** 0_B The interrupt request is disabled 1_B An interrupt request is generated when timer GTk overflows

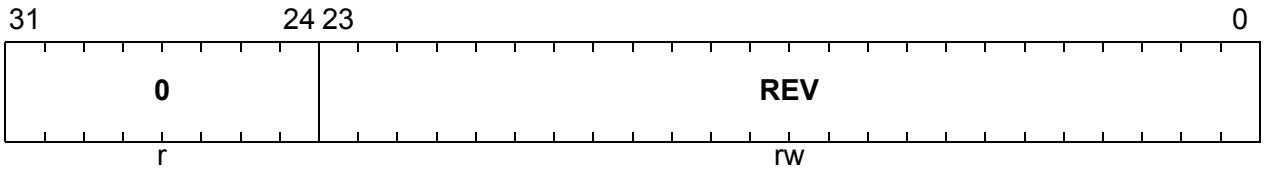
Field	Bits	Type	Description
0	[31:8]	r	**Reserved** Read as 0; should be written with 0.

GTTIMk (k = 0-1)
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.

GTREVk (k = 0-1)
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.

22.3.8 Global Timer Cell Registers

GTCCTRk (k = 00-31)
Global Timer Cell Control Register k [Capture Mode]
(100_H+k*8_H) **Reset Value: 0000 0000_H**

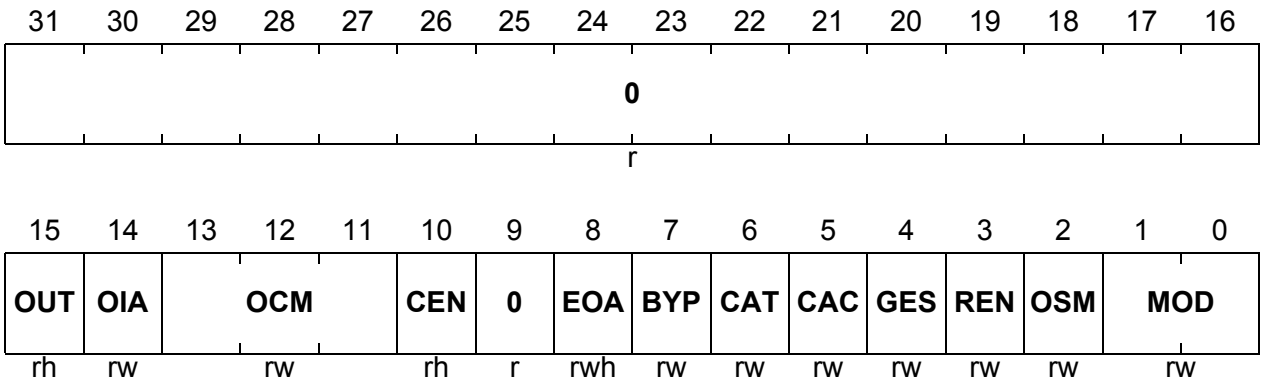
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

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.
NE	6	rw	**Not Effective** Reserved

Field	Bits	Type	Description
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. This bit is protected during read-modify-write operations (hardware will win).
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.

GTCCTRk (k = 00-31)
Global Timer Cell Control Register k [Compare Mode]
(100_H+k*8_H) **Reset Value: 0000 0000_H**

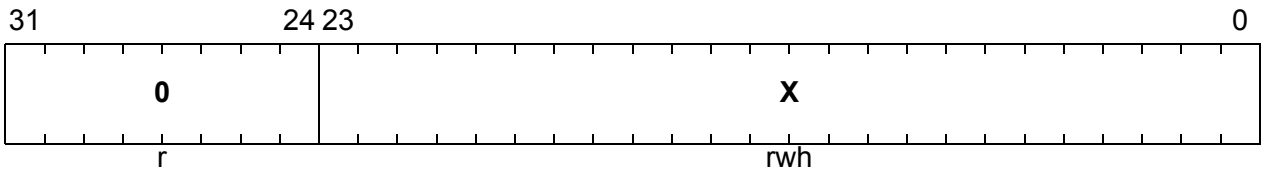


Field	Bits	Type	Description
MOD	[1:0]	rw	**Mode Control Bits** 00_B GTCk operates in Capture Mode hooked to GT0. 01_B GTCk operates in Capture Mode hooked to GT1. 10_B GTCk operates in Compare Mode hooked to GT0. 11_B GTCk operates in Compare Mode hooked to GT1.
OSM	2	rw	**One Shot Mode Enable** 0_B 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 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. This bit is protected during read-modify-write operations (hardware will win).
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.

GTCXRk (k = 00-31)
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.

22.3.9 Local Timer Cell Registers

LTCCTRk (k = 00-62)
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															
r															

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 0LTCk 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 22-153**).
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 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 cleared by software by writing bit CUDCLR with 1 and CUD with 0.

Field	Bits	Type	Description
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).
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.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

LTCCTRk (k = 00-62)
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															
r															

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.
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	**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. This bit is cleared if mode is switched to Timer Mode. OCM2 must be set in any case to enable reaction on M0I/M1I 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 **Page 22-153**). 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 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.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

LTCCTRk (k = 00-62)
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															
r															

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.

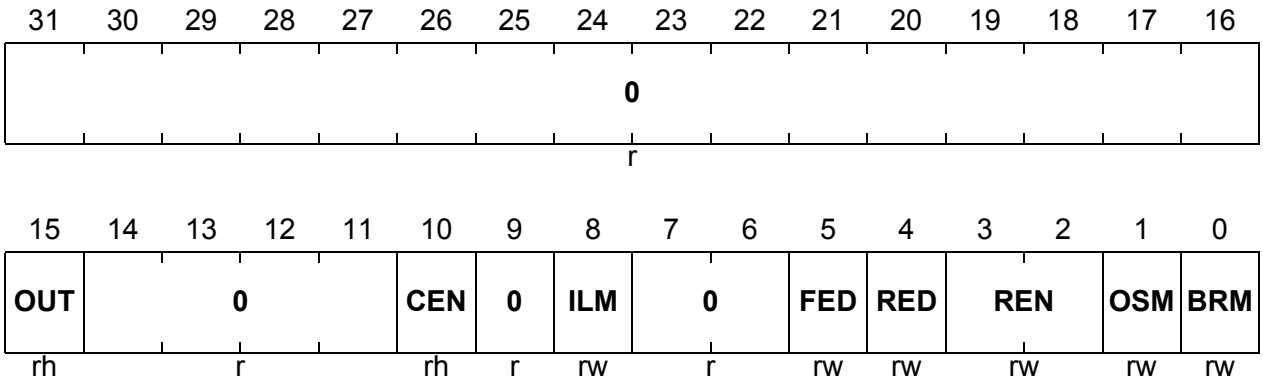
Field	Bits	Type	Description
REN	3	rw	**Request Enable** 0_B Service request is disabled. 1_B Service request SQSk is activated when a - capture event has occurred - compare event has occurred - timer overflow has happened depending on the operation mode selected by bit field MOD.
SOL	4	rw	**Compare Mode: 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)].
BYP	6	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. This bit is cleared if mode is switched to Timer Mode. OCM2 must be set in any case to enable reaction on M0I/M1I 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 **Page 22-153**). 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 module clock selection as input clock, Level Sensitive Mode must be selected. In this case the Edge Sensitive Mode will not produce any event.

Field	Bits	Type	Description
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).
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.
0	[31:16]	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.

LTCCTR63
Local Timer Cell Control Register 63 ($3F8_H$)

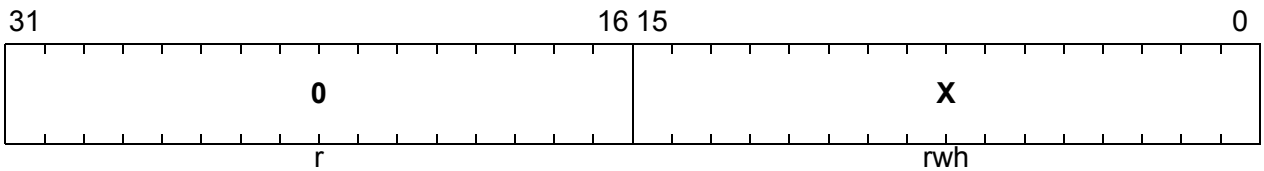
Reset Value: $0000\ 0000_H$



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.

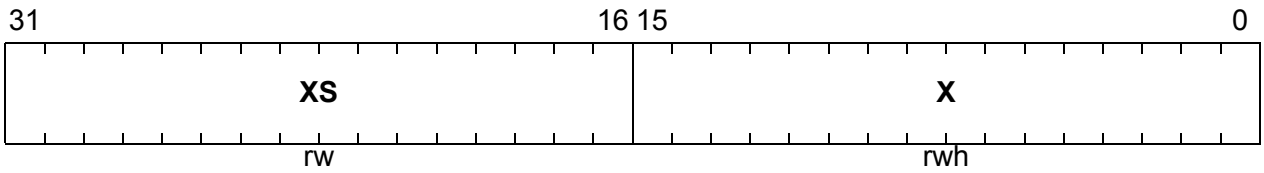
Field	Bits	Type	Description
FED	5	rw	**Falling Edge Select for Shadow Register Copy** 0_B Shadow register copy is not triggered by a falling edge on the LTC63IN input line. 1_B Shadow register copy is triggered by a falling edge on the LTC63IN input line.
ILM	8	rw	**Shadow Register Copy Input Line Mode** 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.

LTCXRk (k = 00-62)
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.

LTCXR63
Local Timer Cell X Register 63 ($3FC_H$) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
X	[15:0]	rwh	**Compare Register Value**
XS	[31:16]	rw	**Shadow Register Value**

22.3.10 I/O Sharing Unit Registers

The three registers MRACTL, MRADIN, and MRADOUT are used to write data to and read data from the GPTA Multiplexer Register Array FIFO. The Multiplexer Register Array FIFO controls the operation of the Input/Output Line Sharing Unit (see **Section 22.2.4.5**).

The Multiplexer Register Array Control register controls the operation of the Multiplexer Register Array FIFO.

MRACTL
Multiplexer Register Array Control Register
(038_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		FIFOFILLCNT						0					FIFOFULL	WCRES	MAEN
r		r						r					r	w	rw

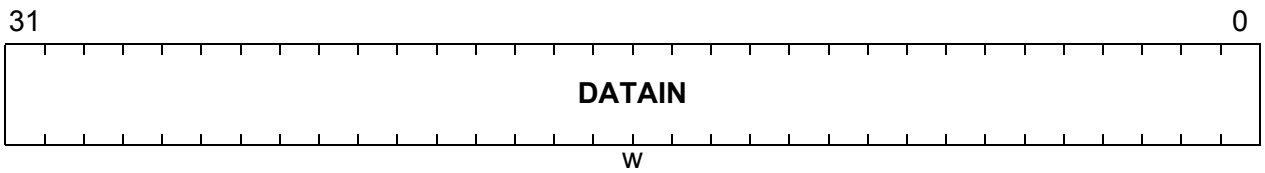
Field	Bits	Type	Description
MAEN	0	rw	**Multiplexer Array Enable** Bit field MAEN enables/disables the programming and the interconnections of the multiplexer array. 0_B Multiplexer array is disabled; all cell inputs are driven with 0, GPTA I/O lines (pins) are disconnected and FIFO writing is enabled. 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.

Field	Bits	Type	Description
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 by WCRES = 0 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.

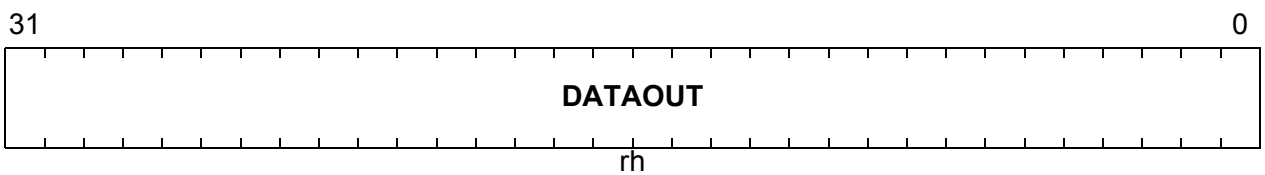
Note: For correct operation, the MRADIN and MRADIN registers must always be read or written 32-bit wide. 8-bit and 16-bit accesses are ignored but without any bus error.

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 Output Multiplexer Control Registers, the GTC Input Multiplexer Control Registers, or the LTC Input Multiplexer Control Registers.

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 Output Multiplexer Control Registers, the GTC Input Multiplexer Control Registers, or the LTC Input Multiplexer Control Registers.

22.3.11 Multiplexer Control Registers

These registers are not directly accessible and can be written and read only via the multiplexer register array FIFO (see Page 22-110).

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.

OMCRLg (g = 0-13)
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

Field	Bits	Type	Description
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.

OMCRHg (g = 0-13)
Output Multiplexer Control Register for Upper Half of Output 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 $X0X_B$ 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.

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 22.2.4.5**).*

GIMCRLg (g = 0-3)
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
GIM EN3	**GIMG3**			**0**	**GIML3**			**GIM EN2**	**GIMG2**			**0**	**GIML2**		
rw	rw			r	rw			rw	rw			r	rw		

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GIM EN1	**GIMG1**			**0**	**GIML1**			**GIM EN0**	**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 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
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 bit 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.

GIMCRHg (g = 0-3)
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	
rw		rw		r		rw		rw		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	
rw		rw		r		rw		rw		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 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
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 bit combinations are reserved.
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.

Field	Bits	Type	Description
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.

LIMCRLg (g = 0-7)
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**		
rw	rw			r	rw			rw	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**		
rw	rw			r	rw			rw	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, 7) 010_B LIMG2g selected 011_B LIMG3g selected 100_B LIMG4g selected All other bit 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.

LIMCRHg (g = 0-7)
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**		
rw	rw			r	rw			rw	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**		
rw	rw			r	rw			rw	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, 7) 010_B LIMG2g selected 011_B LIMG3g selected 100_B LIMG4g selected All other bit combinations are reserved.
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.

Field	Bits	Type	Description
0	3, 11, 19, 27	r	**Reserved** Read as 0; should be written with 0.

22.3.12 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. 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.

The service request status flags can be cleared by software when writing a 1 to the corresponding bit location in the SRSCx registers. Writing a 0 has no effect.

SRSC0
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	**DCMk[1] 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	**DCMk[1] 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.
DCM00C, DCM01C, DCM02C, DCM03C	2, 5, 8, 11	rwh	**DCMk[1] Compare Event Service Request State** 0_B No service is requested. 1_B Service is requested due to a compare event occurred in DCMk unit (k = 0-3).

Field	Bits	Type	Description
PLL	12	rwh	**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	**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	**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) k = 0-3; k = 0 refers to DCM00R, DCM00P, or DCM00C; k = 1 refers to DCM01R, DCM01P, or DCM01C; k = 2 refers to DCM02R, DCM02P, or DCM02C; k = 3 refers to DCM03R, DCM03P, or DCM03C.

SRSC1
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	**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.

SRSC2
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	**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.

SRSC3
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	**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.

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.

SRSS0
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	**DCMk[1] 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	**DCMk[1] 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	**DCMk[1] 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	**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	**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	**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) k = 0-3; k = 0 refers to DCM00R, DCM00P, or DCM00C; k = 1 refers to DCM01R, DCM01P, or DCM01C; k = 2 refers to DCM02R, DCM02P, or DCM02C; k = 3 refers to DCM03R, DCM03P, or DCM03C.

SRSS1
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.

SRSS2
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	**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.

SRSS3
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	**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.

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 22-113.

SRNR
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 22-113).
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

22.4 GPTA Module Implementation

This section describes the GPTA interfaces as implemented in TC1766 with the clock control, port and Micro Second Channel connections, interrupt control, and address decoding.

22.4.1 Interconnections of the GPTA0 Module

The following items are described in this section:

- GPTA 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 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 address map

Figure 22-73 shows the TC1766 specific implementation details and interconnections of the GPTA0 module. The module is supplied with clock control and address decoding logic.

The GPTA0 module has 56 input signals and 112 output signals which can be connected to 48 port pins and 32 MSC interface lines. Additional four inputs for internal connections (coming from the SCU) and 38 service request outputs are provided.

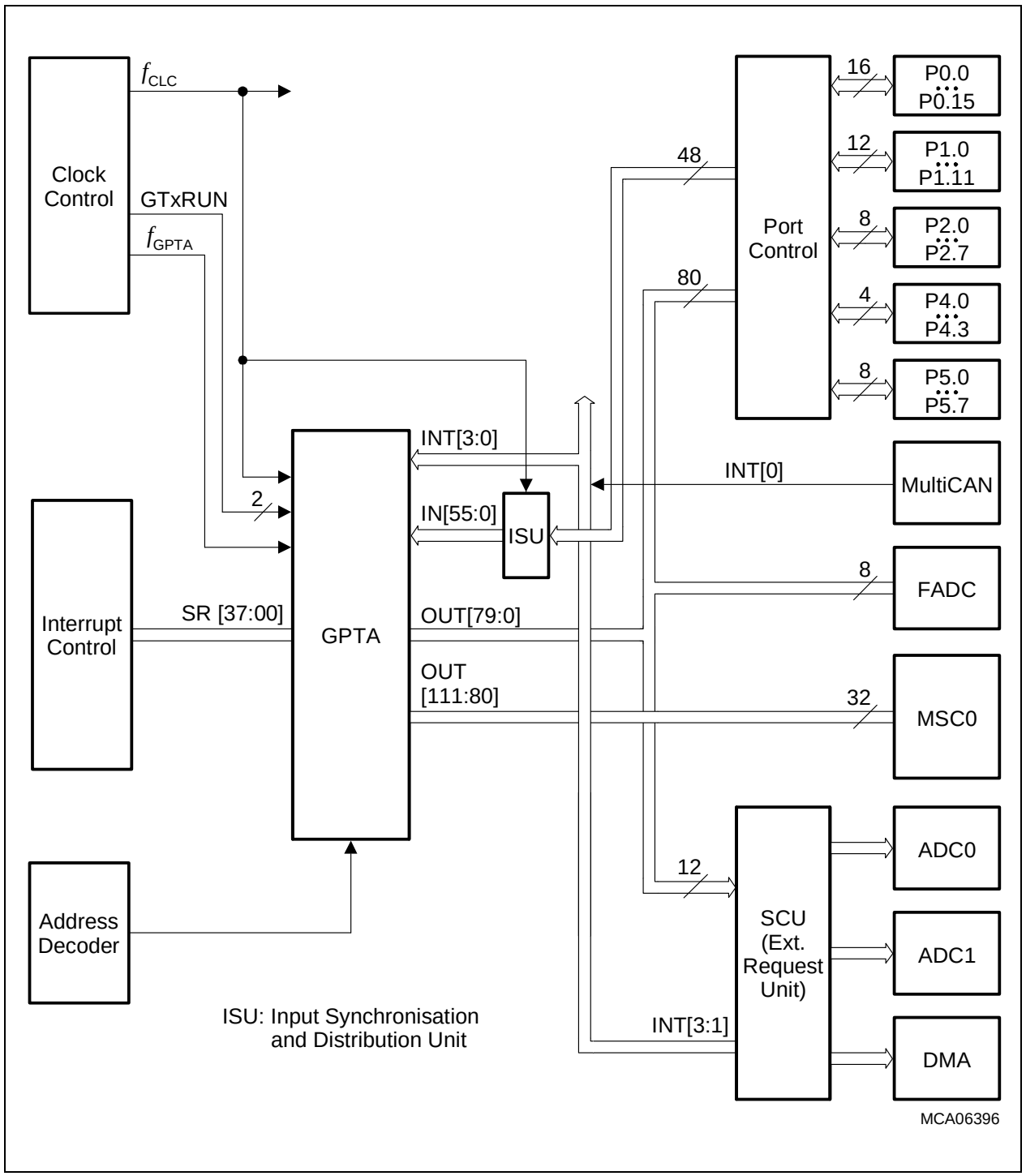


Figure 22-73 Block Diagram of GPTA Implementation

22.4.2 GPTA Module External Registers

Figure 22-74 summarizes the GPTA module related external registers that are required for GPTA0 programming. These registers are referenced and (some of it) described in detail in the following sub-sections.

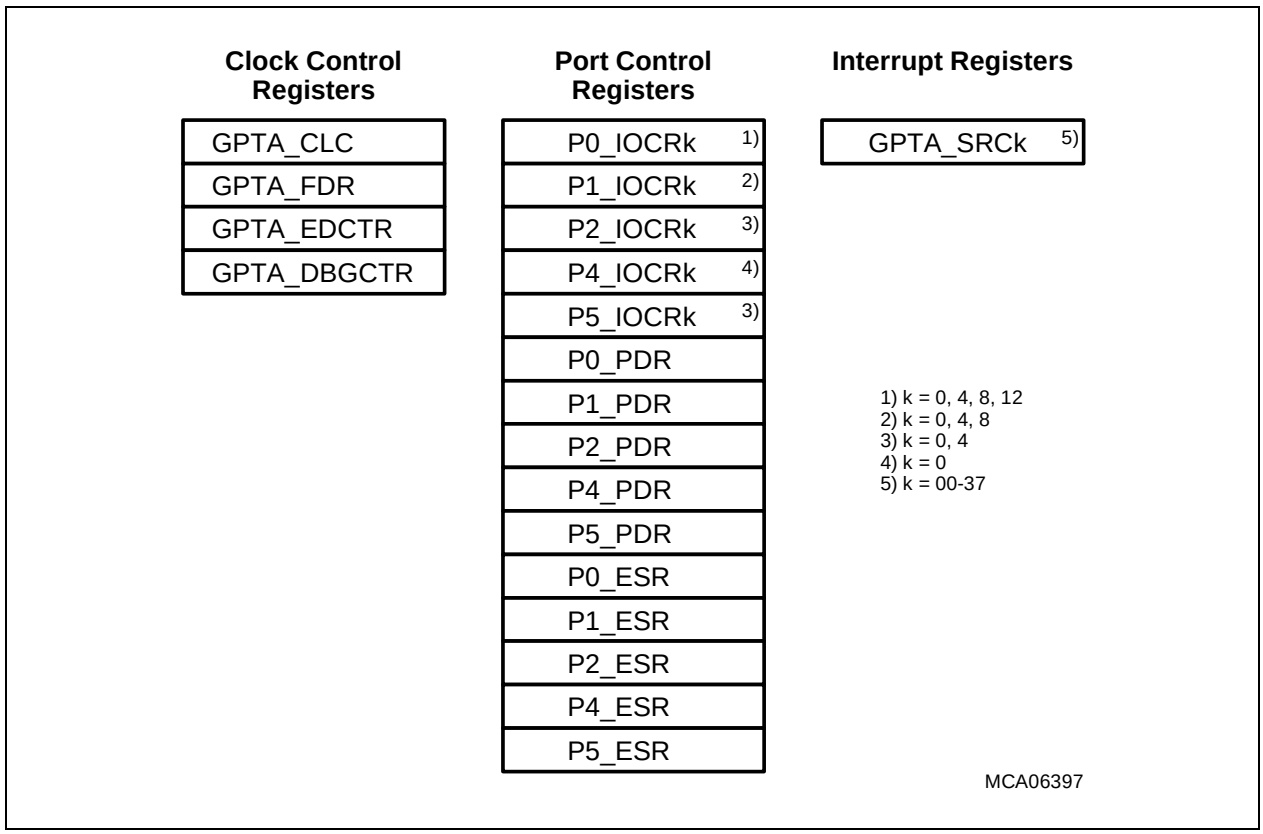


Figure 22-74 GPTA Implementation-specific Special Function Registers

22.4.3 Port Control and Connections

This section describes the I/O connections of the GPTA0 module.

22.4.3.1 I/O Port Line Assignment

In the TC1766, the seven I/O groups and three output groups of GPTA0 with their input lines IN[55:0] and output lines OUT[79:0] are assigned to five 8-bit port groups and two 4-bit port groups as shown in **Figure 22-75**. Within an 8-bit or 4-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, P0.13 is assigned to IN13/OUT13.

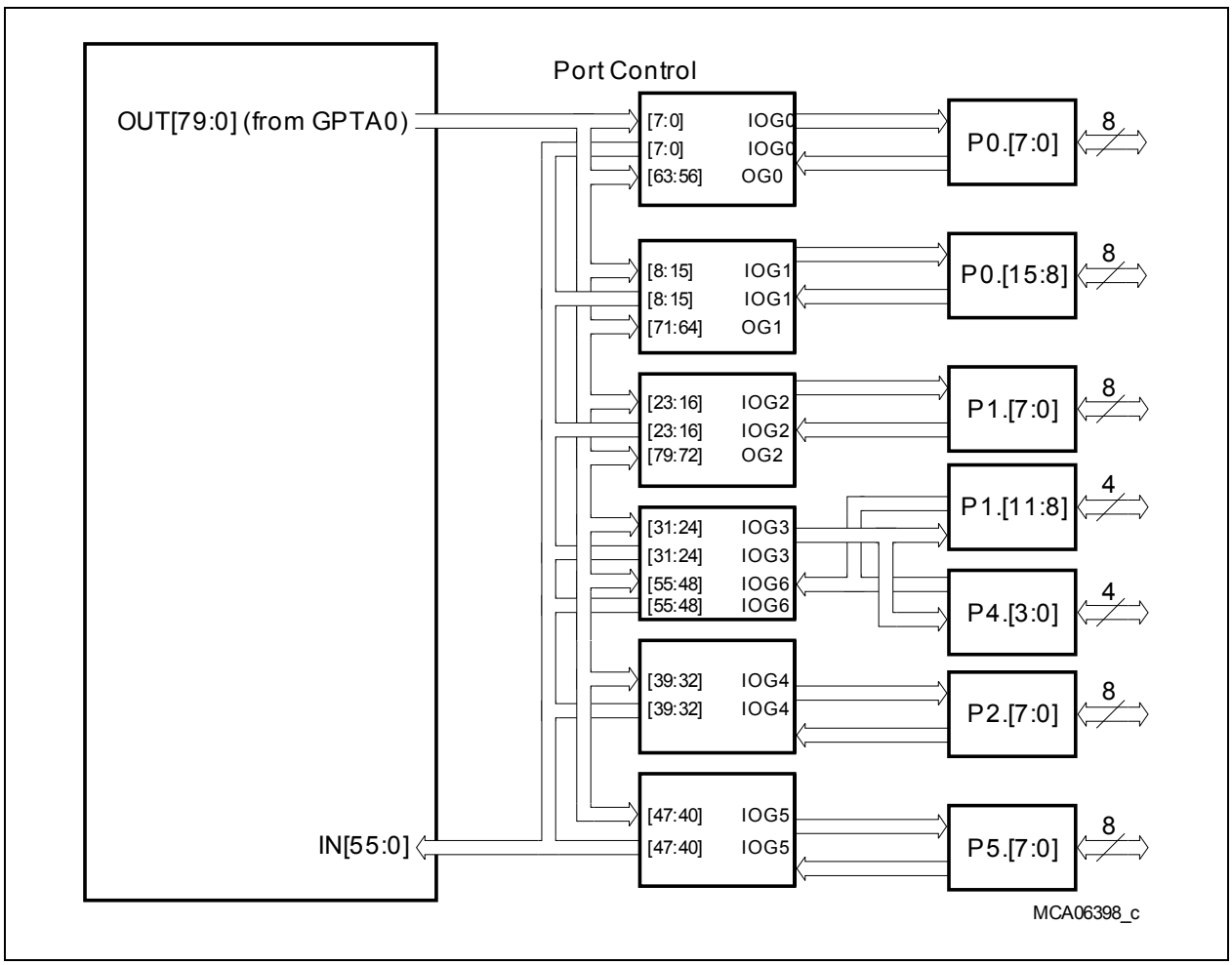


Figure 22-75 I/O Port Line Assignment

The interconnections between the GPTA0 module 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)

22.4.3.2 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 module are controlled by the port input/output control registers for Port 0, Port 1, Port 2, Port 4, and Port 5. Each of the input/output control registers controls four port lines using a 4-bit wide bit field PCx (definitions see **Table 22-19**). **Table 22-18** shows which of the input/output control register bit field is related to a specific GPTA0 module I/O line. Note that input P0.1/IN1 has special connections (see **Page 22-222**).

Table 22-18 IOCR Assignment for GPTA Port Lines

Port	Port Lines for GPTA	GPTA I/O Lines		Controlled by IOCR Register
		Input	Output	
Port 0	P0.[3:0]	IN[3:0][1)]	OUT[3:0] / OUT[59:56]	P0_IOCR0
	P0.[7:4]	IN[7:4]	OUT[7:4] / OUT[63:60]	P0_IOCR4
	P0.[11:8]	IN[11:8]	OUT[11:8] / OUT[67:64]	P0_IOCR8
	P0.[15:12]	IN[15:12]	OUT[15:12] / OUT[71:68]	P0_IOCR12
Port 1	P1.[3:0]	IN[19:16]	OUT[19:16] / OUT[75:72]	P1_IOCR0
	P1.[7:4]	IN[23:20]	OUT[23:20] / OUT[79:76]	P1_IOCR4
	P1.[11:8]	IN[27:24] / IN[51:48]	OUT[27:24] / OUT[51:48]	P1_IOCR8
Port 2	P2.[3:0]	IN[35:32]	OUT[35:32]	P2_IOCR0
	P2.[7:4]	IN[39:36]	OUT[39:36]	P2_IOCR4
Port 4	P4.[3:0]	IN[31:28] / IN[55:52]	OUT[31:28] / OUT[55:52]	P4_IOCR0
Port 5	P5.[3:0]	IN[43:40]	OUT[43:40]	P5_IOCR0
	P5.[7:4]	IN[47:44]	OUT[47:44]	P5_IOCR4

1) There is a special connection provided for GPTA input line IN1 (see **Page 22-222**).

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 22-19**.

Table 22-19 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			GPTA0 output (ALT1) selected	
1010_B			GPTA0 output (ALT2) selected	
1011_B			GPTA0 output (ALT3) selected	
1100_B		Open-drain	General purpose output selected	
1101_B			GPTA0 output (ALT1) selected	
1110_B			GPTA0 output (ALT2) selected	
1111_B			GPTA0 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 modules simultaneously as input.

Port lines selected as GPTA0 output are forced to a 0 level if the related multiplexer array in the I/O Line Sharing Unit 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.

22.4.3.3 Pad Driver Characteristics Selection

The output driver strength and the slew rate of GPTA 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 22-20** shows which of the pad driver register bit field is related to a specific GPTA0 module I/O line group.

Table 22-20 PDR Assignment for GPTA Port Lines

Port	Pad Class	PDR Register PDx Bit Field	Controlled Port Lines	Related GPTA Output Lines
Port 0	A1	P0_PDR.PD0	P0.[7:0]	OUT[7:0] / OUT[63:56]
	A1	P0_PDR.PD1	P0.[15:8]	OUT[15:8] / OUT[71:64]
Port 1	A1	P1_PDR.PD0	P1.[3:0]	OUT[19:16] / OUT[75:72]
	A1	P1_PDR.PD1	P1.[7:4]	OUT[23:20] / OUT[79:76]
	A2	P1_PDR.PDSSC1B	P1.[11:8]	OUT[27:24] / OUT[51:48]
Port 2	A2	P2_PDR.PDMLI0	P2.0, P2.[3:2], P2.5	OUT32 / OUT[35:34] / OUT37
	A2	P2_PDR.PDMSC0	P2.1	OUT33
	A1	P2_PDR.PD0	P2.4, P2.[7:6]	OUT36 / OUT[39:38]
Port 4	A1	P4_PDR.PD0	P4.[1:0]	OUT[29:28] / OUT[53:52]
	A2	P4_PDR.PD1	P4.2	OUT30 / OUT54
	A2	P4_PDR.PDSYSCLK	P4.3	OUT31 / OUT55
Port 5	A2	P5_PDR.PD0	P5.[7:0]	OUT[47:40]

PDx Selection Table

Table 22-21 Pad Driver Mode Mode Selection (Class A1/A2 Pads)

Pad Class	PDx Bit Field	Driver Strength	Signal Transitions
A1	$XX0_B$	Medium driver	–
	$XX1_B$	Weak driver	–
A2	000_B	Strong driver	Sharp edge[1)]
	001_B		Medium edge[1)]
	010_B		Soft edge[1)]
	011_B	Weak driver	–
	100_B	Medium driver	Sharp edge
	101_B		Medium edge
	110_B		Soft edge
	111_B	Weak driver	–

1) In strong driver mode, the output driver characteristics of class A2 pads can be additionally controlled by the temperature compensation logic.

22.4.3.4 Emergency Control of GPTA Output Ports Lines

Port lines connected to GPTA output pins can be selectively switched into an Emergency Mode. In this mode, GPTA output pins react immediately to an active input signal P1.4 (HWCFG1) and drive a logic level that has been programmed in the port output register. As a result, in Emergency Mode a GPTA output pin drives a predefined value instead of the corresponding logic level that is provided on the related GPTA module output line.

All GPTA pins at Port 0, Port 1, Port 2, Port 4, and Port 5 are connected to one common emergency stop signal that is generated in the System Control Unit of the TC1766. More details about the generation of this emergency stop signal are described in the “System Control Unit” chapter of the TC1766 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 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 4, and Port 5 that can be connected to the GPTA module.

Table 22-22 Emergency Control for GPTA Port Output Lines

Port	ESR Register	ESR Enable Bits	GPTA Output Lines
Port 0	P0_ESR	EN[15:0]	OUT[15:0] / OUT[71:56]
Port 1	P1_ESR	EN[11:0]	OUT[27:16] / OUT[79:72], OUT[51:48]
Port 2	P2_ESR	EN[7:0]	OUT[39:32]
Port 4	P4_ESR	EN[3:0]	OUT[31:28] / OUT[55:52]
Port 5	P5_ESR	EN[7:0]	OUT[47:40]

22.4.4 On-Chip Connections

This section describes all on-chip interconnections of the GPTA0 module except the connections to I/O ports (see **Section 22.4.3**).

22.4.4.1 MSC Controller Connections

The MSC interface (MSC0) provides a serial communication link typically used to connect power switches or other peripheral devices. Up to 32 MSC0 output extension lines (bits) can be connected to the GPTA.

Table 22-23 shows the GPTA-to-MSC interconnection and the assignment of the GPTA module's four OGx output group lines OGx.y to the output signals OUT[111:80].

Table 22-23 GPTA0 to MSC Interconnection Assignment

MSC0 Input Line	Assigned GPTA0 Output Line	MSC0 Input Line	Assigned GPTA0 Output Line
ALTIN0.0	OUT80 / OG3.0	ALTIN1.0	OUT96 / OG5.0
ALTIN0.1	OUT81 / OG3.1	ALTIN1.1	OUT97 / OG5.1
ALTIN0.2	OUT82 / OG3.2	ALTIN1.2	OUT98 / OG5.2
ALTIN0.3	OUT83 / OG3.3	ALTIN1.3	OUT99 / OG5.3
ALTIN0.4	OUT84 / OG3.4	ALTIN1.4	OUT100 / OG5.4
ALTIN0.5	OUT85 / OG3.5	ALTIN1.5	OUT101 / OG5.5
ALTIN0.6	OUT86 / OG3.6	ALTIN1.6	OUT102 / OG5.6
ALTIN0.7	OUT87 / OG3.7	ALTIN1.7	OUT103 / OG5.7
ALTIN0.8	OUT88 / OG4.0	ALTIN1.8	OUT104 / OG6.0
ALTIN0.9	OUT89 / OG4.1	ALTIN1.9	OUT105 / OG6.1
ALTIN0.10	OUT90 / OG4.2	ALTIN1.10	OUT106 / OG6.2
ALTIN0.11	OUT91 / OG4.3	ALTIN1.11	OUT107 / OG6.3
ALTIN0.12	OUT92 / OG4.4	ALTIN1.12	OUT108 / OG6.4
ALTIN0.13	OUT93 / OG4.5	ALTIN1.13	OUT109 / OG6.5
ALTIN0.14	OUT94 / OG4.6	ALTIN1.14	OUT110 / OG6.6
ALTIN0.15	OUT95 / OG4.7	ALTIN1.15	OUT111 / OG6.7

22.4.4.2 GPTA Connections with SCU, MultiCAN, FADC, DMA, Ports

The GPTA0 module of the TC1766 has several on-chip interconnections with the SCU, MultiCAN, FADC and DMA modules. **Figure 22-76** shows these interconnections.

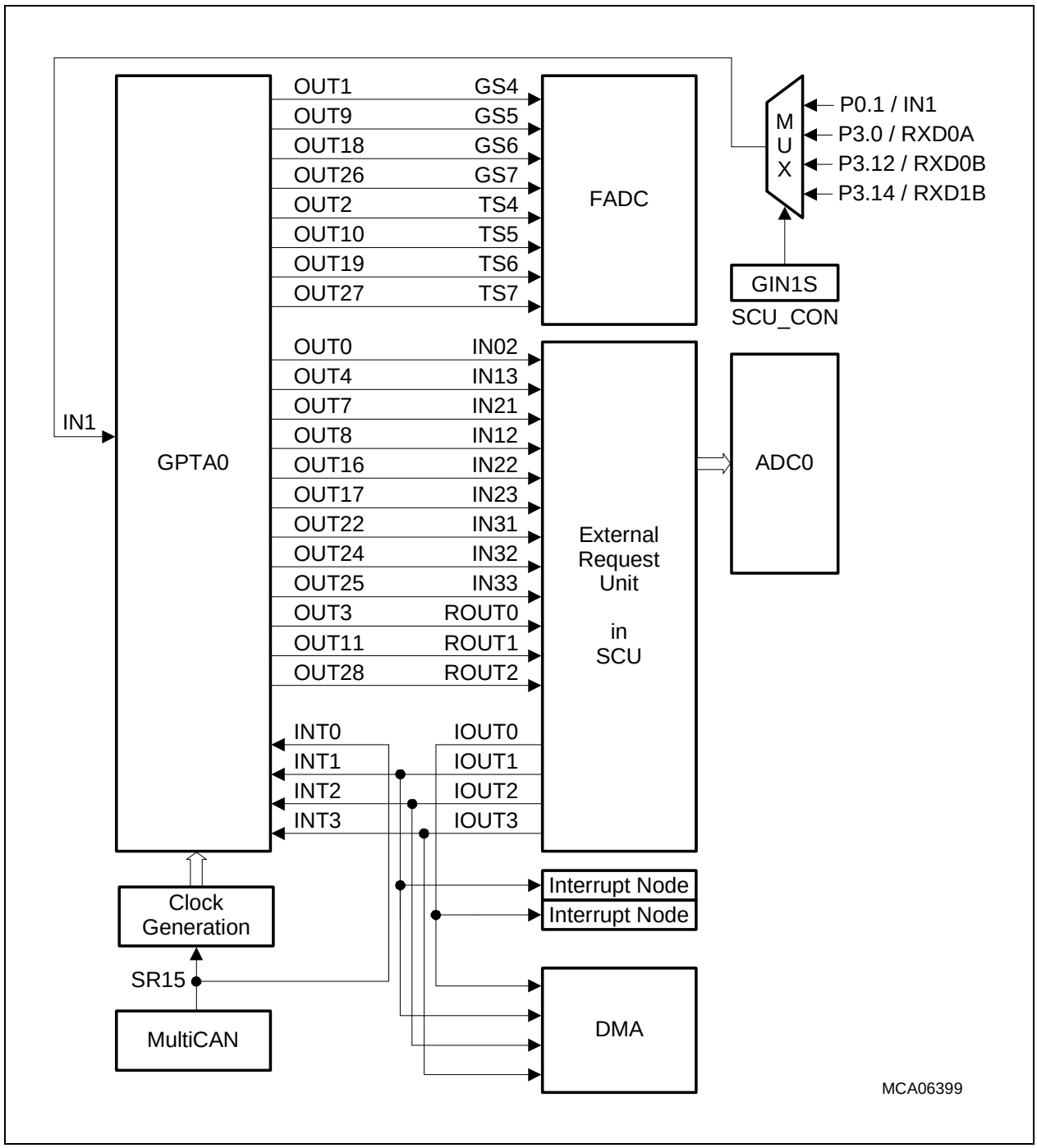


Figure 22-76 Connections of GPTA with On-Chip Modules

System Control Unit

The SCU contains the external request unit (ERU), which is especially responsible for controlling requests coming from the MSC module, port pins, or from the GPTA0 module and passing these request to AD converters, DMA controller, or interrupt nodes.

MultiCAN Connections

The MultiCAN controller has one connection to the GPTA modules:

- MultiCAN service request output SR15 is connected to the INT0 input of GPTA0.

DMA Controller

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 six GPTA output lines. Three of these four DMA request output signals are connected to the GPTA0 internal inputs INT[3:1]. These connections allow for example to trigger GTC or LTC events in the GPTA modules by a request coming from a port pin or from the MSC clock.

FADC Connections

As shown in **Figure 22-76**, eight GPTA0 output lines are connected as trigger input signals or gating input signals with 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 IN1

The input line IN1 of the GPTA0 module is connected to the output of a 4-to-1 multiplexer. This multiplexer is controlled by bit field SCU_CON.GIN1S and makes it possible to connect the GPTA0 input IN1 with one out of four port input lines. This feature especially allows the baud rates of an ASC0 of ASC1 receiver input signal to be measured by timers of the GPTA0.

Table 22-24 GPTA0 Input Line IN1 Connections

SCU_CON.GIN1S	GPTA0 Input IN1 Connected to
00_B	P0.1 / IN1 (default after reset)
01_B	P3.0 / RXD0A
10_B	P3.12 / RXD0B
11_B	P3.14 / RXD1B

22.4.5 Module Clock Generation

As shown in **Figure 22-77**, the clock signals for the GPTA0 module are generated and controlled by one clock generation unit. This clock generation unit is responsible for the enable/disable control, the clock frequency adjustment, and the debug clock control. The unit includes four registers:

- **Clock Control Register GPTA0_CLC** (see **Page 22-226**), responsible for the generation of the control clock f_{CLC} that is used by each of the three kernel modules as general control tasks.
- **Fractional Divider Register GPTA0_FDR** (see **Page 22-227**), responsible for the frequency control of the module timer clock f_{GPTA0}.
- **Clock Enable/Disable Control Register GPTA0_EDCTR** (see **Page 22-229**), responsible for the enable/disable control of the module timer clock f_{GPTA0}, and for the run control for the Global Timers in GPTA0.
- **Debug Clock Control Register GPTA0_DBGCTR** (see **Page 22-230**), responsible for the module timer clock control in Debug Mode.

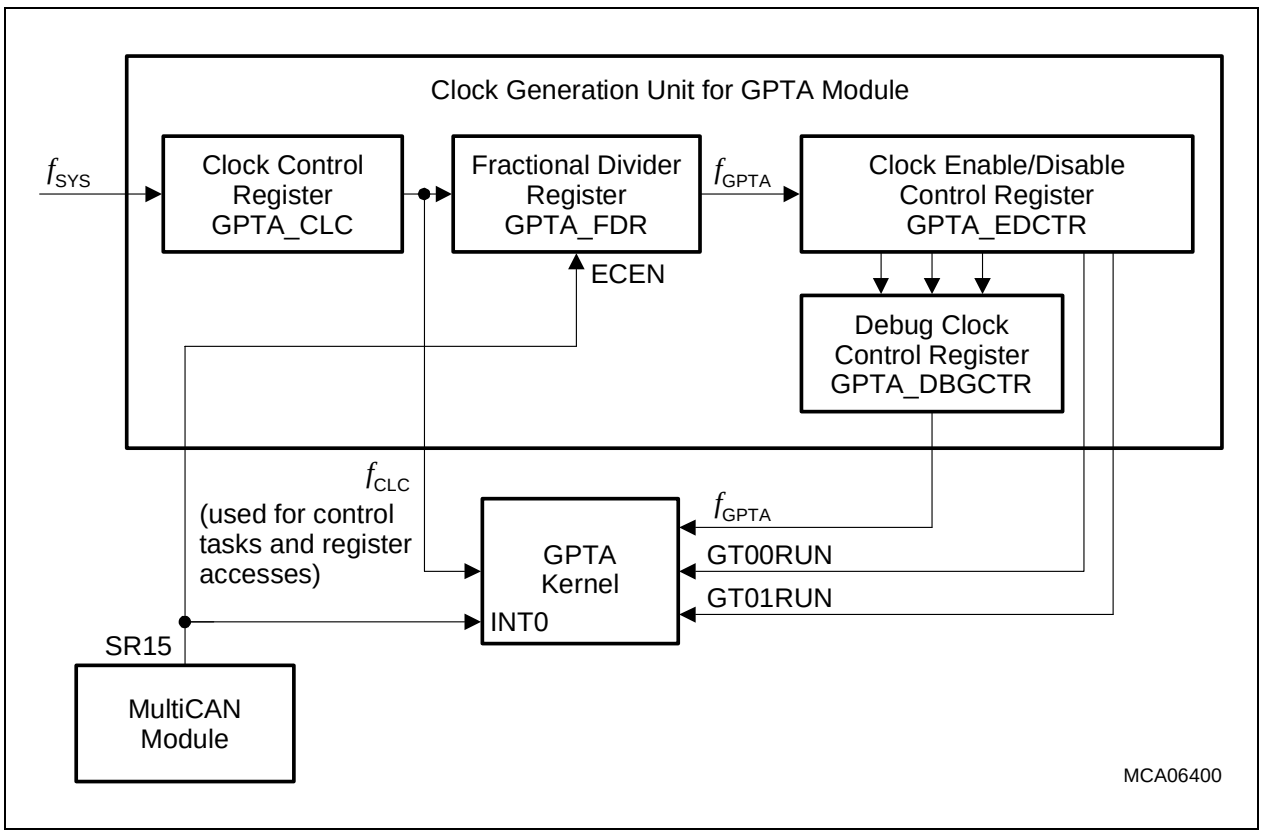


Figure 22-77 GPTA Module Clock Generation

Note: Registers GPTA0_CLC, GPTA0_FDR, GPTA0_EDCTR and GPTA0_DBGCTR are located in the address space of GPTA0.

The module clock f_{CLC} is used inside the GPTA 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 module timer clock f_{GPTA0} is used inside the GPTA module kernel as input clock for the timers. This module timer clock has the same frequency as f_{GPTA} (as selected through register GPTA0_FDR) and can be enabled/disabled through register GPTA0_ECDTR.

Attention: If f_{GPTA0} is disabled by the enable bits in register GPTA0_ECDTR, f_{CLC} keeps on running. In this case, that means that register accesses to the GPTA module 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 make it possible to control the timer clock f_{GPTA0} 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 clock f_{GPTA0} 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 cleared by hardware after the programmed number of clock pulses has been issued. This feature makes it possible to single step the GPTA module with a programmable timer clock granularity.

Attention: If the frequency of the module timer clock f_{GPTA0} is configured to be smaller than the control clock f_{CLC} (as programmed in register GPTA0_FDR) or even disabled (as programmed in register GPTA0_EDCTR), an action initiated by a write access to a module register could be significantly delayed, because the register write access is clocked by f_{CLC} and the register content is evaluated by hardware using the slower or disabled module timer clock f_{GPTA0}.

22.4.5.1 Clock Control Register

The clock control registers makes it possible to control (enable/disable) the module control clock f_{CLC}. The clock signal f_{CLC} is used by the GPTA0 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	**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_{CLC} clock is disabled (DISS set). Therefore, the GPTA modules clock generation is completely disabled.

*Note: Additional details on the clock control register functionality are described in section **"Clock Control Register CLC" on Page 3-24** of the TC1766 User's Manual System Units part (Volume 1).*

The fractional divider register controls the clock frequency of the module timer clock f_{GPTA0}.

GPTA0_FDR
GPTA Fractional Divider Register ($00C_H$) Reset Value: $0000\ 0000_H$

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									

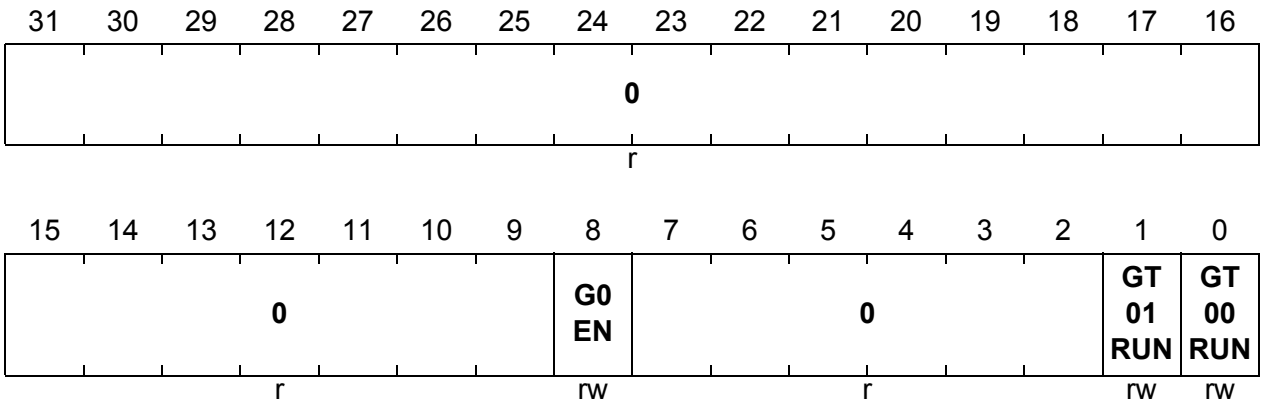
Field	Bits	Type	Description
STEP	[9:0]	rw	**Step Value** Reload or addition value for RESULT.
SM	11	rw	**Suspend Mode** SM selects between granted or immediate suspend mode.
SC	[13:12]	rw	**Suspend Control** This bit field determines the behavior of the fractional divider in suspend mode.
DM	[15:14]	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 $\overline{\text{SPNDACK}}$ 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]	rw	**Reserved** Read as 0; should be written with 0.

*Note: Further details on the fractional divider register functionality are described in section **"Fractional Divider Operation" on Page 3-29** of the TC1766 User's Manual System Units part (Volume 1).*

The clock enable/disable control register controls two functions: clock enable/disable control for each Global Timer in the GPTA0 module and enable/disable control for the module timer clock, f_{GPTA0}.

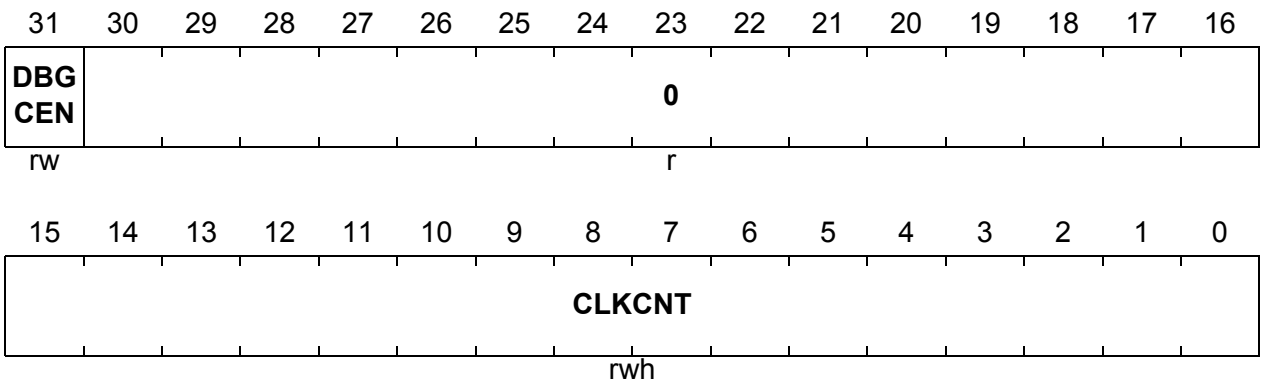
GPTA0_EDCTR
GPTA Clock Enable/Disable Control Register
(400_H) **Reset Value: $0000\ 0000_H$**



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.
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.
0	[7:2], [31:9]	r	**Reserved** Read as 0; should be written with 0.

The debug clock control register makes it possible to control the module timer clock f_{GPTA0} for debug purposes on base 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 module timer 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}.
0	[30:16]	r	**Reserved** Read as 0; should be written with 0.

22.4.6 Limits of Cascading GTCs and LTCs

As shown on **Page 22-56** and **Page 22-69**, a maximum of 32 GTCs and 64 LTCs can be cascaded. In the TC1766, however cascading of GTCs and LTCs is limited under certain conditions.

If the LTCs are running with the maximum module timer clock of $f_{GPTA} = f_{SYS}$ = 80 MHz, a maximum of 20 GTCs and 20 LTCs can be connected together. If the module timer 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 fractional divider defines the maximum number of cascaded GTCs and LTCs.

Table 22-25 Limits of Cascading GTCs and LTCs

f_{SYS}	Selected Clock Divider Ratio[1)]	Max. Number of Cascaded GTCs/LTCs
80 MHz	1 <= f_{SYS}/f_{GPTA} < 2	20 GTCs, 20 LTCs
	2 <= f_{SYS}/f_{GPTA} < 3	no limits for GTCs, 40 LTCs
	3 <= f_{SYS}/f_{GPTA}	no limits for GTCs and LTCs

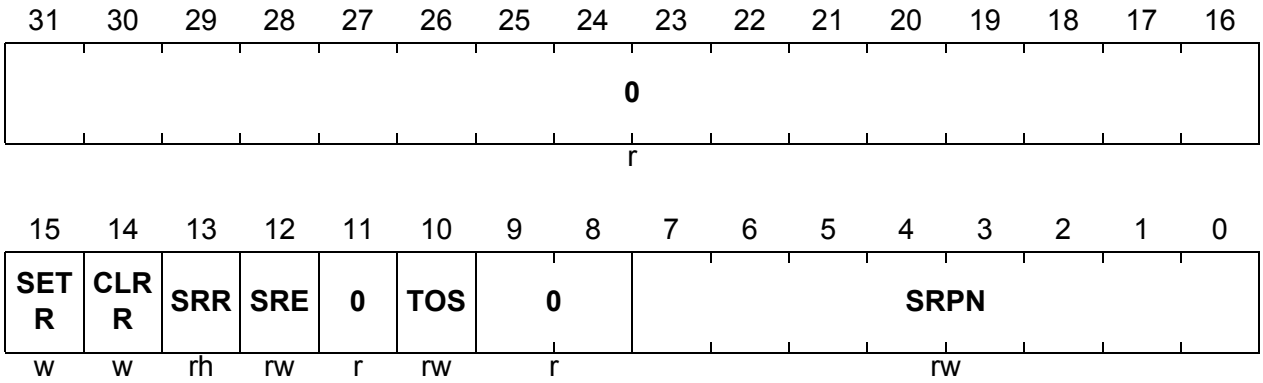
1) Selected by the fractional divider.

22.4.7 Interrupt Registers

Each of the service request outputs of the GPTA0 module kernel 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]

SRCk (k = 00-37)
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 on **Page 12-3** of the TC1766 User's Manual System Units part (Volume 1).*

22.4.8 GPTA0 Register Map

Figure 22-78 shows a graphical diagram of all kernel and implementation specific registers of the GPTA0 module .

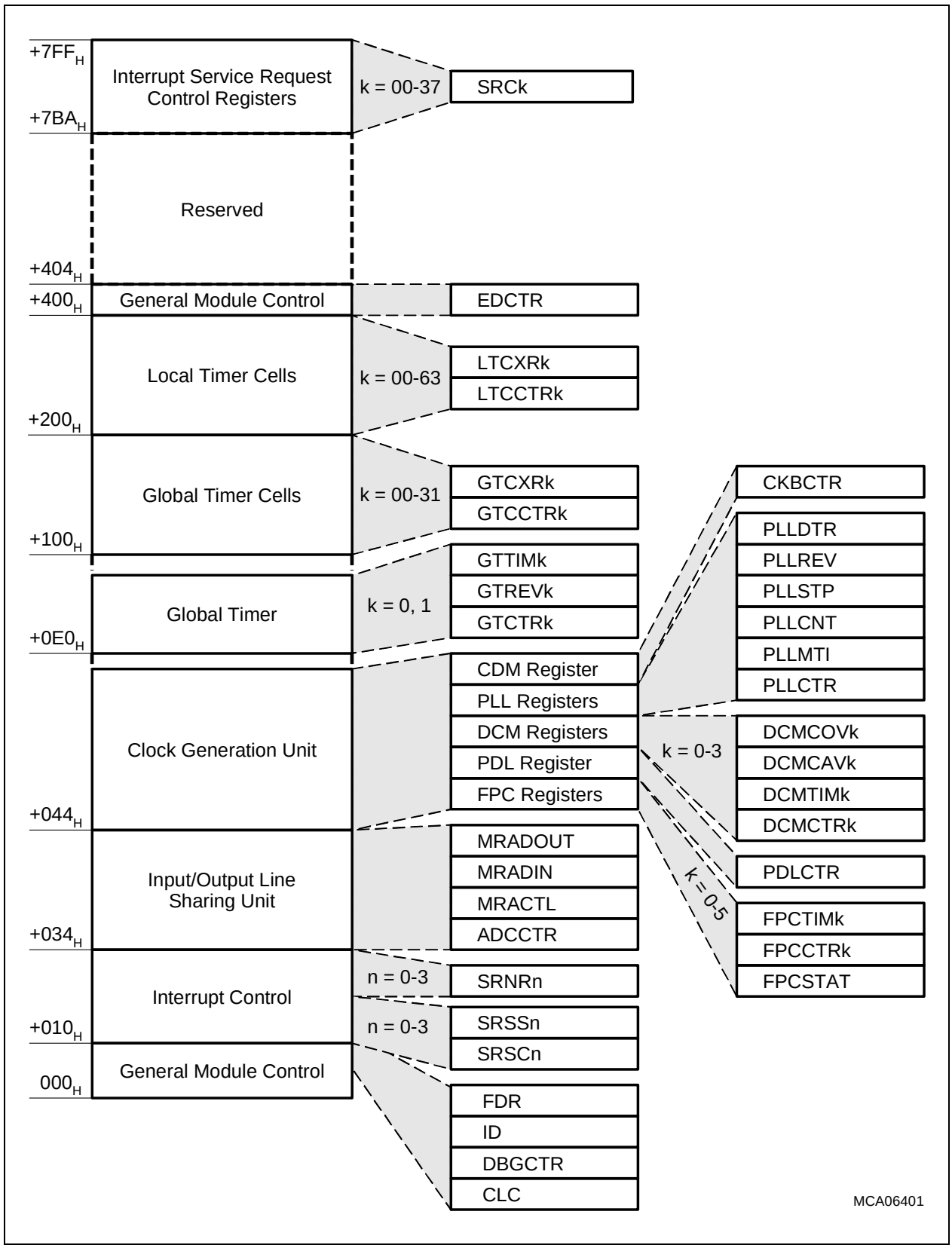


Figure 22-78 GPTA0 Register Map

23 Analog-to-Digital Converter (ADC)

The TC1766 contains a medium-speed Analog-to-Digital Converter (ADC0) and a fast Analog-to-Digital Converter (FADC).

ADC0 provides 2-3 µs conversion time @10-bit resolution and is intended primarily for single-ended signals. It offers a very flexible and comprehensive control for monitoring a large number of relatively slow signals.

The FADC offers very fast conversion rates (280 ns at 10-bit resolution) thus allowing sampling of high-frequency signals. For slow and mid-range frequency signal, heavy oversampling can be performed to avoid the usage of expensive filters.

The FADC is described in **Chapter 24**. This chapter describes in detail the Analog-to-Digital converter ADC0. It contains the following sections:

- Functional description of the ADC Kernel for ADC0 (see **Page 23-2**)
- Register descriptions of all ADC Kernel specific registers (see **Page 23-49**)
- TC1766 implementation-specific details and registers of the ADC0 module including port connections and control, service request control, address decoding, and clock control (see **Page 23-85**).

*Note: The ADC Kernel register names described in **Section 23.2** are referenced in the TC1766 User's Manual with the module name prefix "ADC0_" for the ADC0 interface.*

23.1 ADC Kernel Description

The on-chip ADC module of the TC1766 is an analog-to-digital converter with 8-bit, 10-bit, or 12-bit resolution including sample & hold functionality. The A/D converter operates by the method of successive approximation. A multiplexer selects up to 32 analog inputs that can be connected with the 16 conversion channels in the ADC module. An automatic self-calibration adjusts the ADC module to changing temperatures or process variations.

Features

- 8-bit, 10-bit, 12-bit A/D conversion
- Conversion time below 2.5 µs @ 10-bit resolution
- Extended channel status information on request source
- Successive approximation conversion method
- Total Unadjusted Error (TUE) of ±2 LSB @ 10-bit resolution
- Integrated sample & hold functionality
- Direct control of up to 16(32) analog input channels per ADC
- Dedicated control and status registers for each analog channel
- Powerful conversion request sources
- Selectable reference voltages for each channel
- Programmable sample and conversion timing schemes
- Limit checking
- Flexible ADC module service request control unit
- Automatic control of external analog multiplexers
- Equidistant samples initiated by timer
- External trigger and gating inputs for conversion requests
- Power reduction and clock control feature
- On-chip die temperature sensor output voltage measurement

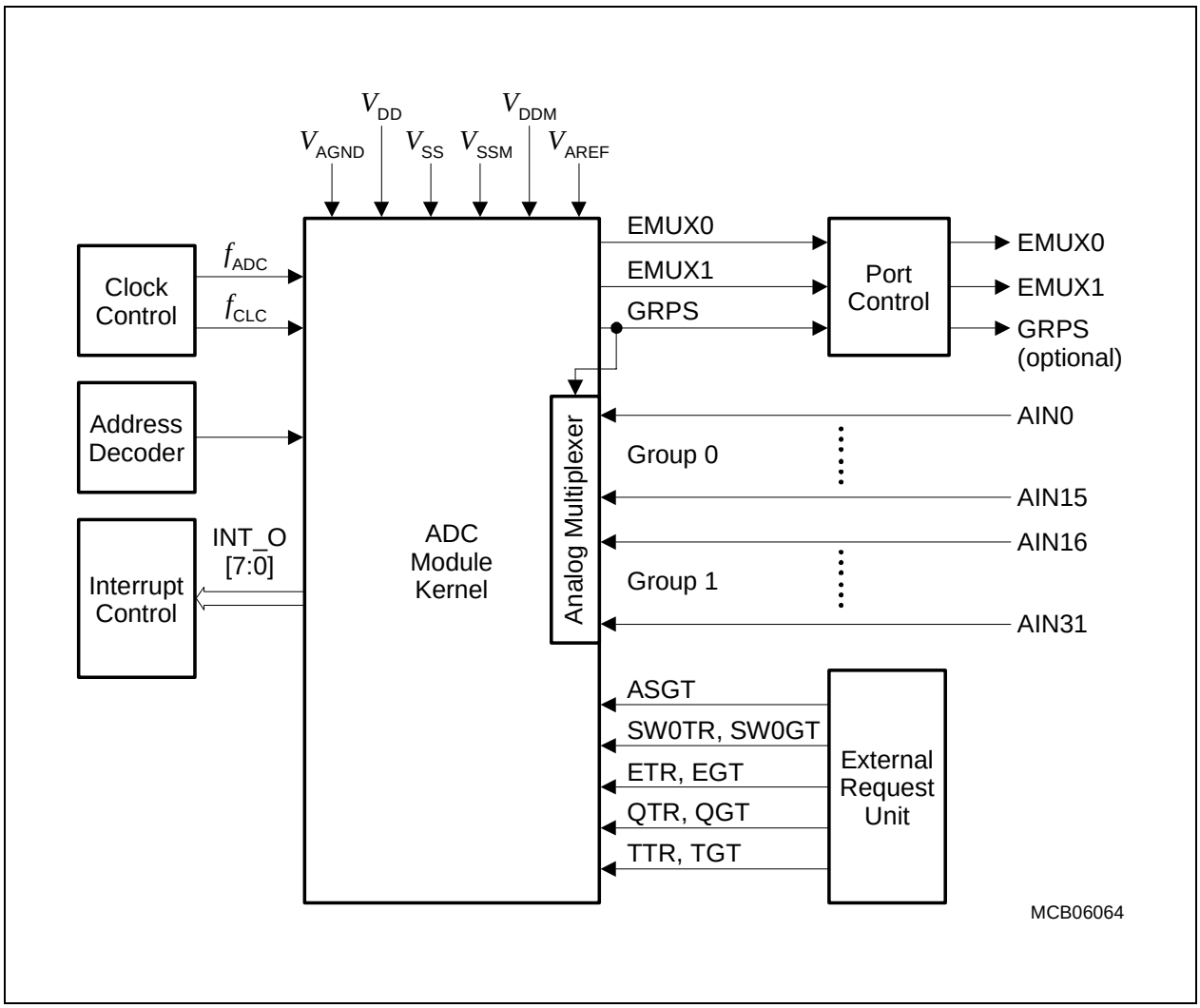


Figure 23-1 General Block Diagram of the ADC Module

As shown in **Figure 23-1**, the ADC module has 16 analog input channels. An analog multiplexer selects the input line for the analog input channels from among 32 analog inputs. Additionally, an external analog multiplexer can be used for analog input extension. External Clock control, address decoding, and service request (interrupt) control are managed outside the ADC module kernel. External trigger conditions are controlled by an External Request Unit. This unit generates the control signals for auto-scan control (ASGT), software trigger control (SW0TR, SW0GT), the event trigger control (ETR, EGT), queue control (QTR, QGT), and timer trigger control (TTR, TGT).

Figure 23-2 provides a more detailed block diagram of the ADC kernel with its main functional units.

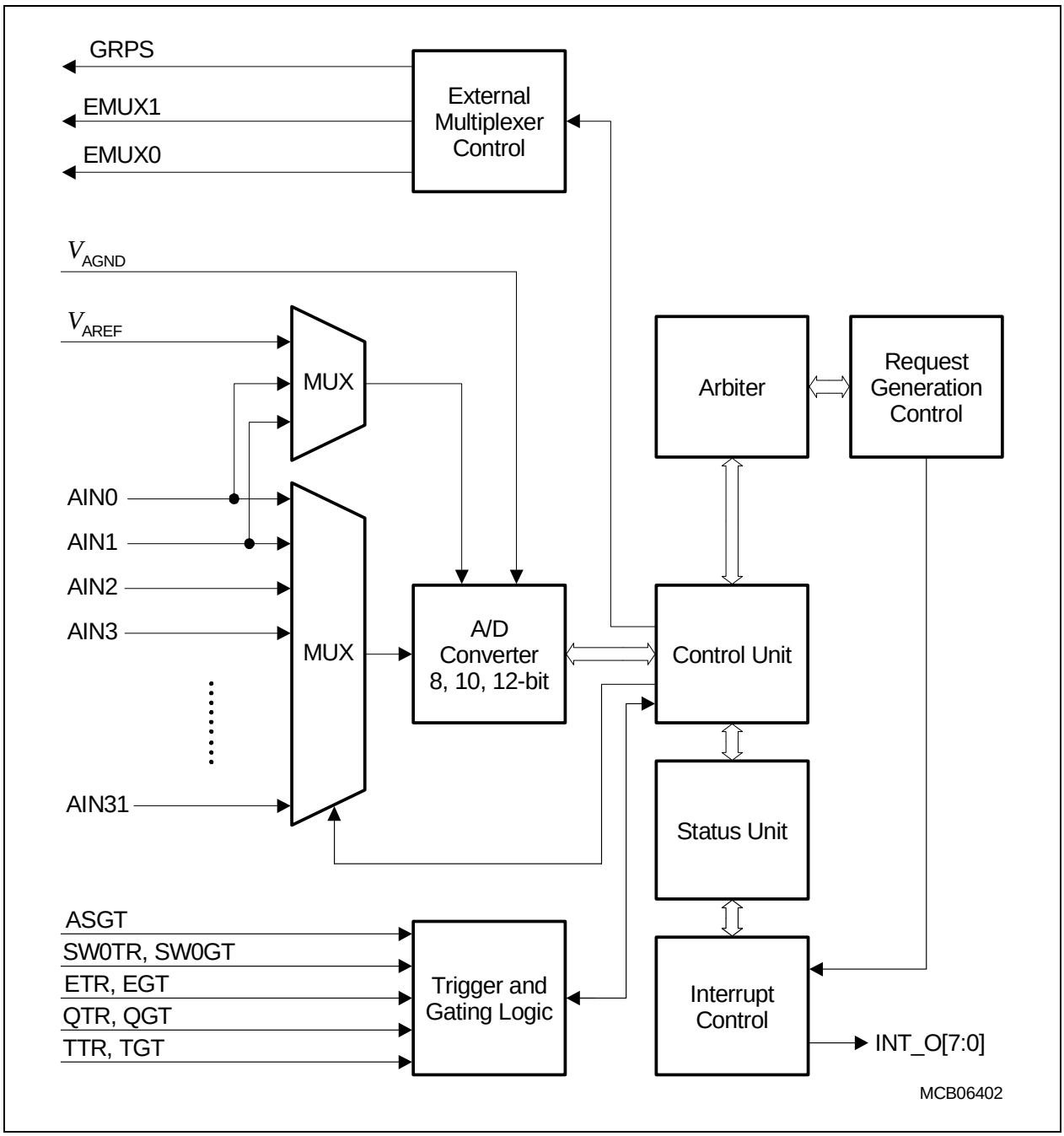


Figure 23-2 Functional Units of the ADC Kernel

23.1.1 Analog Input Connections

The 32 analog inputs AIN[31:0] of the ADC module are connected to the 16 analog input channels of the TC1766 via analog input multiplexers. These analog input multiplexers are shown in **Figure 23-3**.

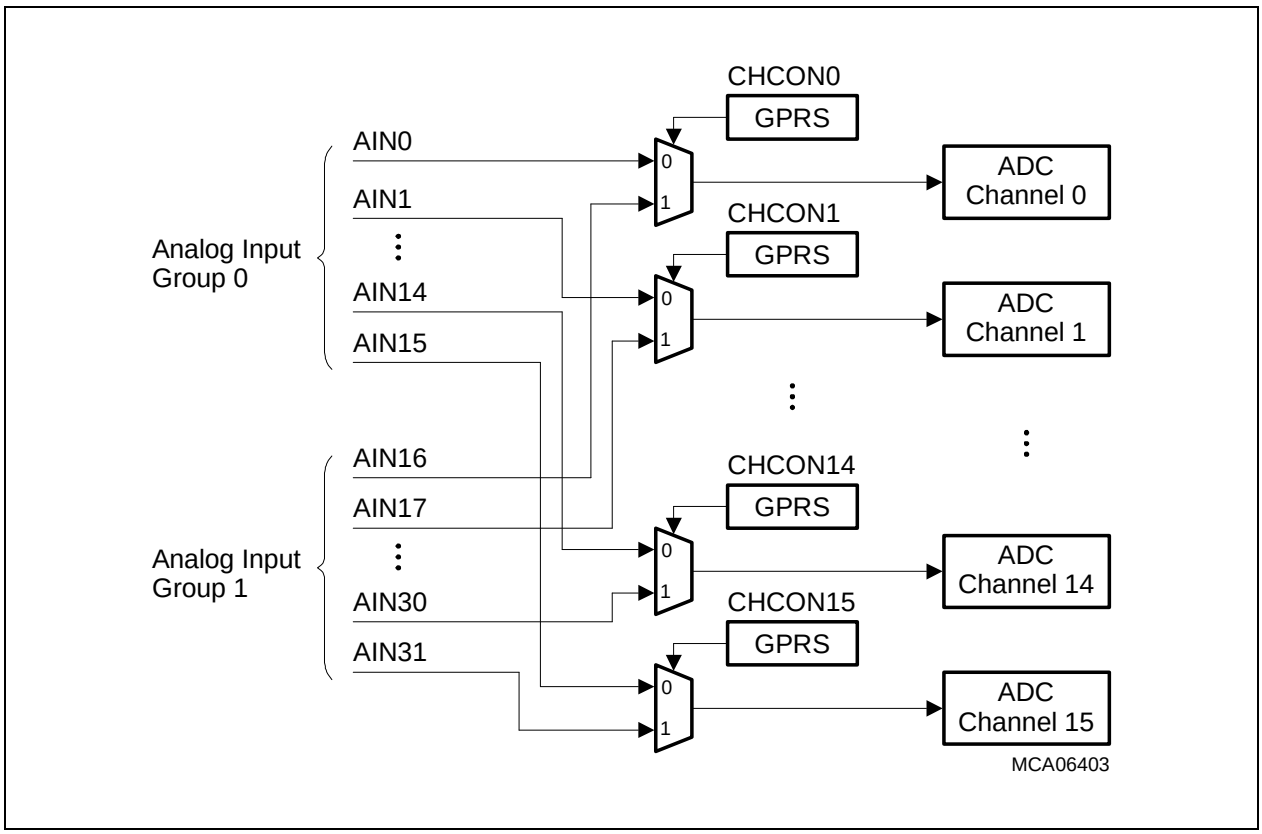


Figure 23-3 Analog Input Multiplexer Configuration

The ADC channel n (n = 0-15) can be connected to two analog inputs: AIN[n] and AIN[n+16]. The selection is made for ADC channel n by the channel control register group select bit CHCONn.GRPS. With CHCONn.GRPS = 0, an analog input of analog input group 0 is selected and with CHCONn.GRPS = 1, an analog input of analog input group 1 is selected.

23.1.2 Conversion Request Sources

The ADC module control logic provides effective methods to request and arbitrate conversions. Conversion requests for one or more analog channels can be triggered by hardware as well as by software to provide maximum flexibility in requesting A/D conversions. Up to six individual configurable conversion request sources are implemented to issue A/D conversion requests.

In principle, the conversion request sources can be assigned either to the group of parallel conversion request sources or the group of sequential conversion request sources. A global overview of parallel and sequential conversion request sources and detailed descriptions of each source are provided in the following sections.

23.1.2.1 Parallel Conversion Request Sources

Parallel conversion request sources generate one or more conversion request at a time for the analog channels. **Table 23-1** shows the available parallel conversion request sources including the associated control and status signals.

Table 23-1 Parallel Conversion Request Sources

Request Source	Control Register	Conversion Req. Pending Register	Arbitration Participation Flag	Source Arbitration Level
Timer	TTC	TCRP	AP.TP	SAL.SALT
External Event	EXTC	EXCRP	AP.EXP	SAL.SALEX
Software	REQ0	SW0CRP	AP.SW0P	SAL.SALSW0
Auto-Scan	SCN	ASCRP	AP.ASP	SAL.SALAS

A parallel conversion request source is controlled by a control register, a request pending register, an arbitration participation flag, and the source arbitration level.

Each parallel conversion control register contains 16 bits defining whether a conversion request is activated for a specific channel. The contents of the parallel conversion control register are loaded into the conversion request pending register on request source specific trigger events. If at least one bit is set in the conversion request pending register, the arbitration participation flag is set for this source. This informs the arbiter to include this parallel conversion request source into arbitration.

If a parallel conversion request source is the arbitration winner, a conversion is started for the conversion request within the conversion request pending register with the highest channel number. Starting a conversion causes the conversion request bit in the conversion request pending register to be cleared by the arbiter. If a currently running conversion initiated by the parallel source is cancelled, the arbiter restores the corresponding conversion request bit in the conversion request pending registers for this

channel. If all pending conversion requests are processed, the arbitration participation flag of this parallel source is cleared.

The bits of the conversion request pending register can be cleared globally under software control by clearing the arbitration participation flag for this source.

23.1.2.2 Sequential Conversion Request Sources

Sequential conversion request sources generate only one conversion request at a time for an analog channel. The settings for this conversion are derived from the control register of the sequential source. **Table 23-2** shows the available sequential conversion request sources including the associated control and status blocks.

Table 23-2 Sequential Conversion Request Sources

Request Source	Control Register	Back-Up Register	Arbitration Participation Flag	Source Arbitration Level
Channel Injection	CHIN	not accessible via Bus	AP.CHP	SAL.SALCHIN
Queue	QR	not accessible via Bus	AP.QP	SAL.SALQ

A sequential conversion request source consists of a control register, a back-up register, an arbitration participation flag, and the source arbitration level.

Each sequential conversion control register contains a conversion request bit, the channel number to be converted, two control bits for the external multiplexer settings, one bit for the analog input channel group selection, and control information to select the resolution of the channel. Setting the conversion request bit causes the arbitration participation flag to be set. This informs the arbiter to include the sequential conversion request source into arbitration. If this sequential source is the arbitration winner, a conversion is started for the analog channel specified within the request register. The settings of the external multiplexer and the resolution of the ADC are also derived from this conversion request control register.

Starting a conversion causes the conversion request bit to be cleared by the arbiter. The arbitration participation flag is automatically cleared if the conversion request register and the back-up register contains no valid request.

If a currently running conversion initiated by a sequential source is cancelled, the arbiter restores the conversion information (conversion request bit, external multiplexer setting, analog input channel group selection, conversion resolution) in the back-up registers for this channel. If the back-up register contains valid conversion information, the arbiter reads from the back-up register instead from the conversion request control register. Thus, the previously cancelled conversion participates in arbitration again. A new

conversion request generated in the meantime via the conversion request register will be performed after the request in the back-up register has been served.

The request bit in the control register and in the back-up register can be cancelled under software control. Clearing the arbitration participation bit clears either the request bit in the request register (if the back-up register contains no request) or the request bit in the back-up register if (the back-up register contains a valid request).

23.1.2.3 Parallel Conversion Request Source “Timer”

Periodic samples can be achieved by timer generated conversion requests. A programmable timer serves as trigger source. Service request generation logic as well as the arbitration lock mechanism are provided to ensure periodical sampling without jitter. The block diagram of the timer and its control and status blocks are shown in **Figure 23-4**.

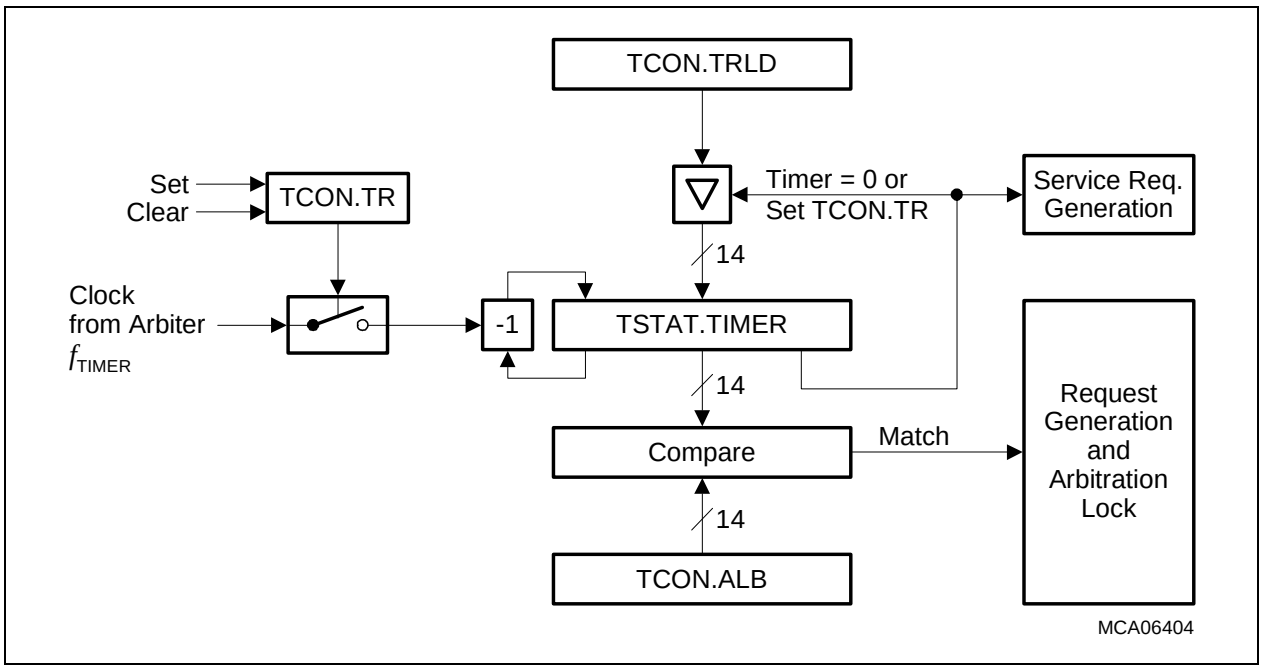


Figure 23-4 Block Diagram of Conversion Request Source “Timer”

While the timer run bit is set, the timer is clocked with f_{TIMER}, which is derived from the arbiter. This synchronizes the timer on the arbiter for jitter-free sampling. If the timer run bit TCON.TR becomes set, the timer bit field TSTAT.TIMER is loaded with the timer reload value TCON.TRLD. With each clock cycle of f_{TIMER}, the timer register is decremented and compared to the arbitration-lock-boundary value TCON.ALB. If the value of the timer register is equal to the value of the arbitration-lock-boundary, the arbitration lock bit STAT.AL is set (see **Figure 23-6**). This arbitration-lock mechanism can be used to generate samples without being delayed by a currently running conversion. When the timer = 0, the arbitration is unlocked, the timer register is reloaded, the arbitration lock bit is cleared, the timer related service request status flag (MSS1.MSRT) is set, and a trigger pulse is sent to the conversion request source “Timer”.

The timer period $t_{TPERIOD}$ can be specified within the range from microseconds up to milliseconds according to the following equation.

$$t_{TPERIOD} = (TRLD) \times \frac{(20)}{f_{ADC}} \qquad (23.1)$$

Figure 23-5 shows the control and status of the conversion request source “Timer”.

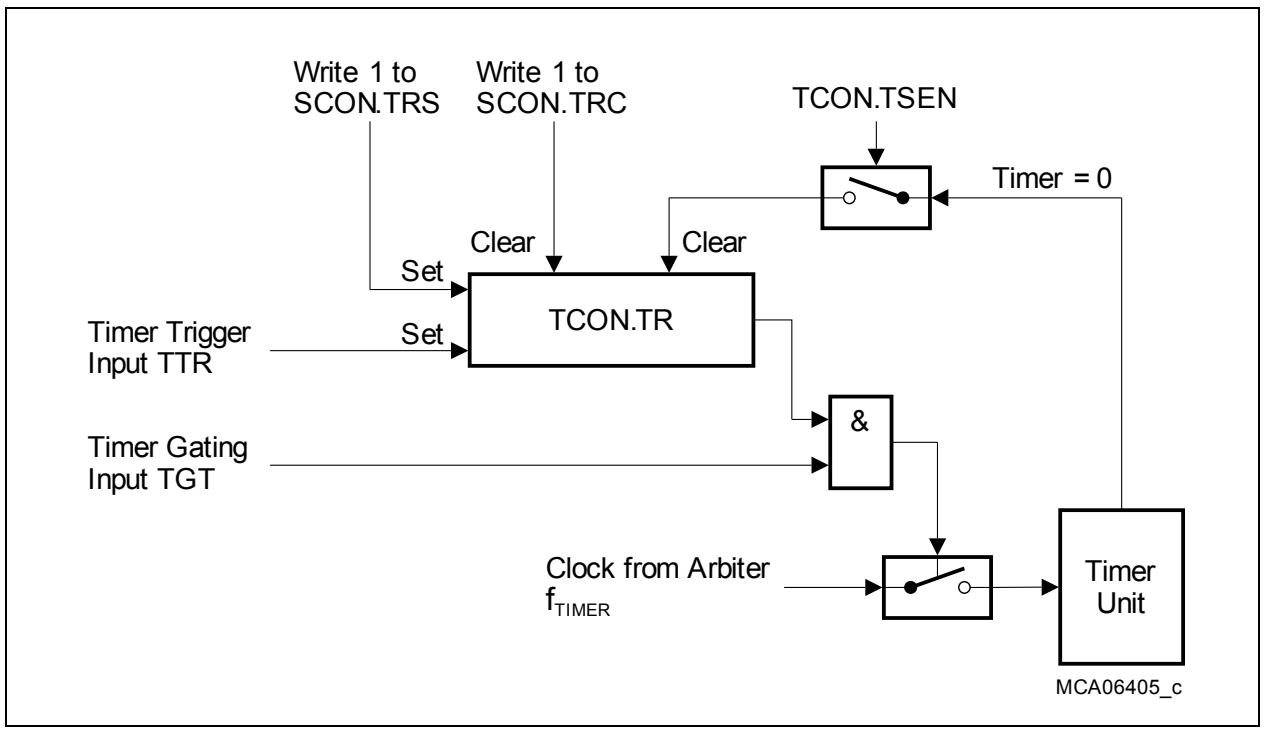


Figure 23-5 Conversion Request Source “Timer”

Up to sixteen analog input channels can be controlled by the conversion request source “Timer”. Setting request bit(s) in the timer trigger control register TTC enables the generation of a conversion request for this analog input channel(s) by the timer. If timer = 0, the contents of the timer trigger control register TTC are loaded into the timer conversion request pending register TCRP. This triggers conversion requests for the selected channel(s) if the gating input line TGT = 1. When the timer gating line TGT = 0, the pending conversion requests do not take part in the arbitration cycle. The timer can also be started by a pulse on the timer trigger input line TTR. The TTR and TGT input connections depend on the product specific implementation of the ADC module. They are described on **Page 23-96**.

The contents of the timer conversion request pending register and the arbitration lock bit are logically OR-ed. If bit STAT.AL or at least one bit is set in the timer conversion request pending register, the arbitration participation flag AP.TP is set. This informs the arbiter to include the conversion request source “Timer” in the arbitration.

If “Timer” is the arbitration winner, a conversion is started for the conversion request within register TCRP with the highest channel number. Starting a conversion causes the conversion request bit in register TCRP to be cleared by the arbiter. If a currently running timer initiated conversion is cancelled, the arbiter sets the corresponding conversion request bit in registers TCRP for this channel. If all pending conversion requests are processed, the arbiter clears the arbitration participation flag AP.TP. The contents of

register TCRP can be cleared globally under software control by clearing the timer arbitration participation flag AP.TP.

The arbitration-lock mechanism provides the means to start timer triggered conversion requests without being delayed by a currently running conversion. **Figure 23-6** shows this method in detail.

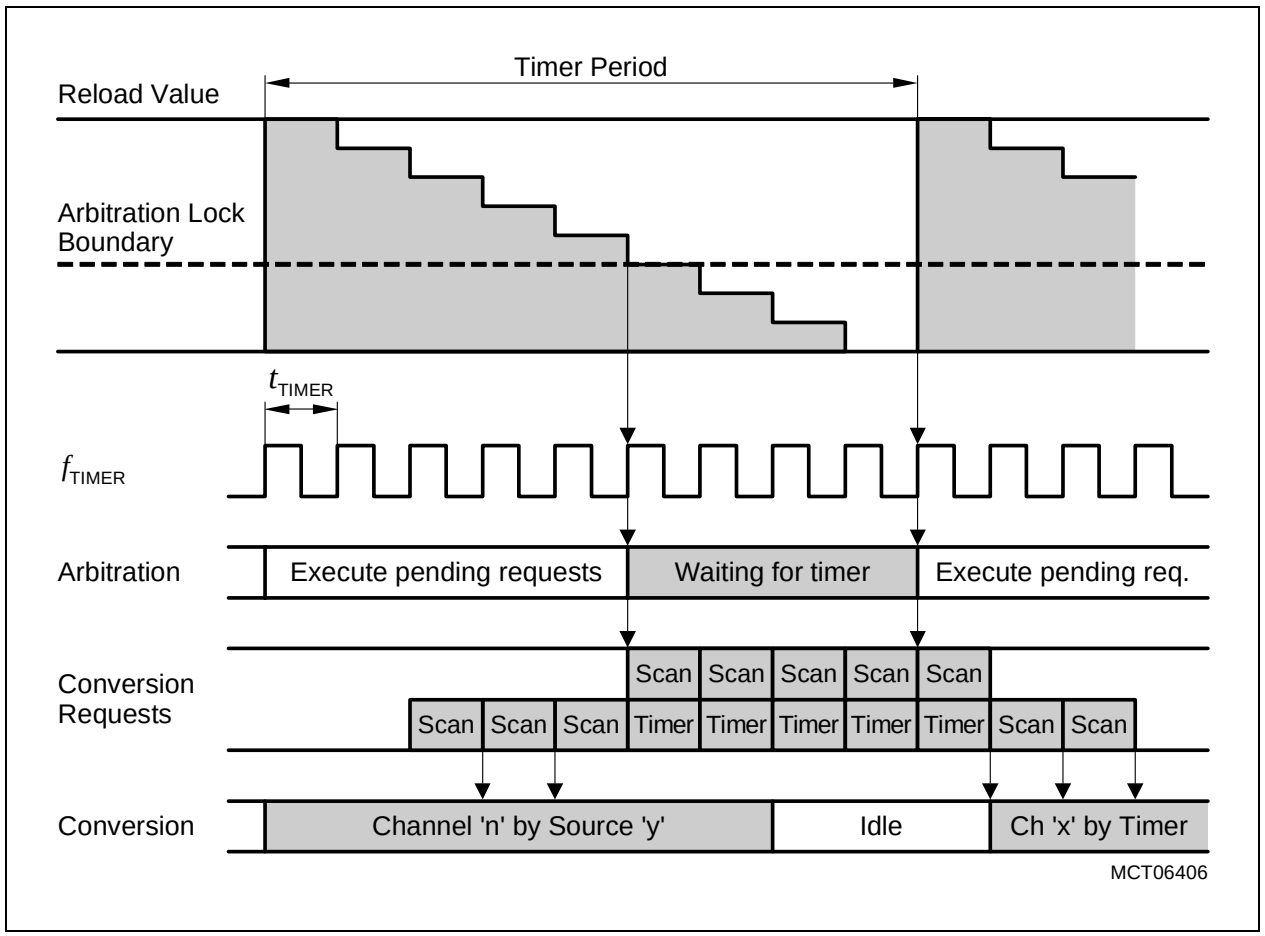


Figure 23-6 Arbitration Lock Mechanism

The arbitration must be locked before the timer is 0 in order to ensure that the running conversion has been finished and no new conversion will be started in the meantime. When the arbitration is locked, lower prioritized conversion request source than the “Timer” are blocked from performing requested conversions.

In the example of **Figure 23-6**, the conversion request source “Auto Scan” has triggered conversion request(s) that are not served according to a currently running conversion and the locked arbitration. On timer = 0, the conversion requested by the timer is started (it is assumed that the “Timer” is programmed to a higher priority than the “Auto Scan”). Arbitration Lock Mode is enabled by setting bit field TCON.ALB to any value greater than zero.

The value of the arbitration lock boundary is also used to specify the time t_{LOCK} for which the arbitration is locked. Running in Arbitration Lock Mode, the current value of the timer register is compared to the arbitration lock boundary. Note that the arbitration will always be locked if the reload value is selected to be equal to or less than the arbitration lock boundary. On a compare match, the arbitration logic is locked (STAT.AL = 1), while timer = 0 removes the arbitration lock. Bit STAT.AL is either cleared if timer = 0 or after clearing bit TCON.TR.

23.1.2.4 Parallel Conversion Request Source “External Event”

Externally triggered conversion requests are mandatory for various control applications. The conversion request source “External Event” receives the trigger line ETR from the external world via the interrupt input unit. **Figure 23-7** shows the structure of the conversion request source “External Event”.

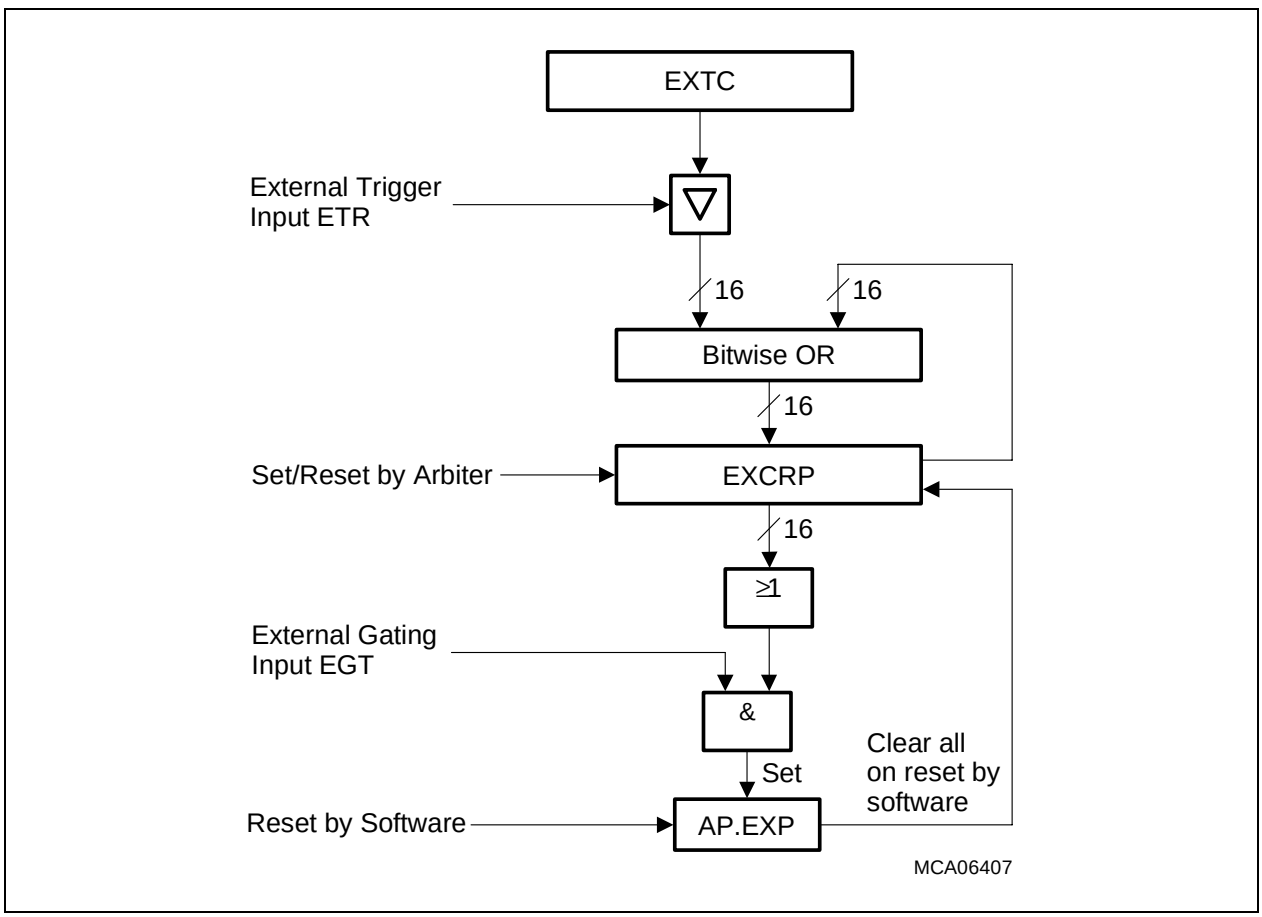


Figure 23-7 Conversion Request Source “External Event”

Up to sixteen individually selectable analog input channels in the external trigger control register EXTC can be controlled by the conversion request source “External Event”. Setting request bit EXTC.ETCHn enables the generation of a conversion request for analog channel n on external events. A trigger issued on an external event loads the

contents of EXTC into the external conversion request pending register EXCRP. This triggers conversion requests for the selected channel(s) while the external gating input EGT = 1. When the external gating line EGT = 0, the pending external conversion requests do not take part in the arbitration cycle.

If an external event is detected (input ETR = 1), the contents of the external trigger control register EXTC are loaded into the external conversion request pending register. “Load” means that the contents of EXTC and the contents of external conversion request pending register EXCRP are bit-wise OR-ed, as shown in **Figure 23-7**.

The ETR and EGT input connections depend on the product specific implementation of the ADC module. They are described on **Page 23-96**.

If at least one bit is set in register EXCRP, the arbitration participation flag AP.EXP is set. This informs the arbiter to include the conversion request source “External Event” into arbitration. If “External Event” is the arbitration winner, a conversion is started for the conversion request within register EXCRP with the highest channel number. Starting a conversion causes the conversion request bit in register EXCRP to be cleared by the arbiter. If a currently running “External Event” initiated conversion is cancelled, the arbiter sets the corresponding conversion request bit in register EXCRP for this channel. If all pending conversion requests are processed, the arbitration participation flag AP.EXP is cleared.

The contents of register EXCRP can be reset globally under software control by clearing the “External Event” arbitration participation flag. Note that conversion requests caused by external trigger pulses are lost if the flag for this channel is already set in the external conversion request pending register.

23.1.2.5 Parallel Conversion Request Source “Software”

The conversion request source “Software” makes it possible to generate conversion request under software control, as shown in **Figure 23-8**.

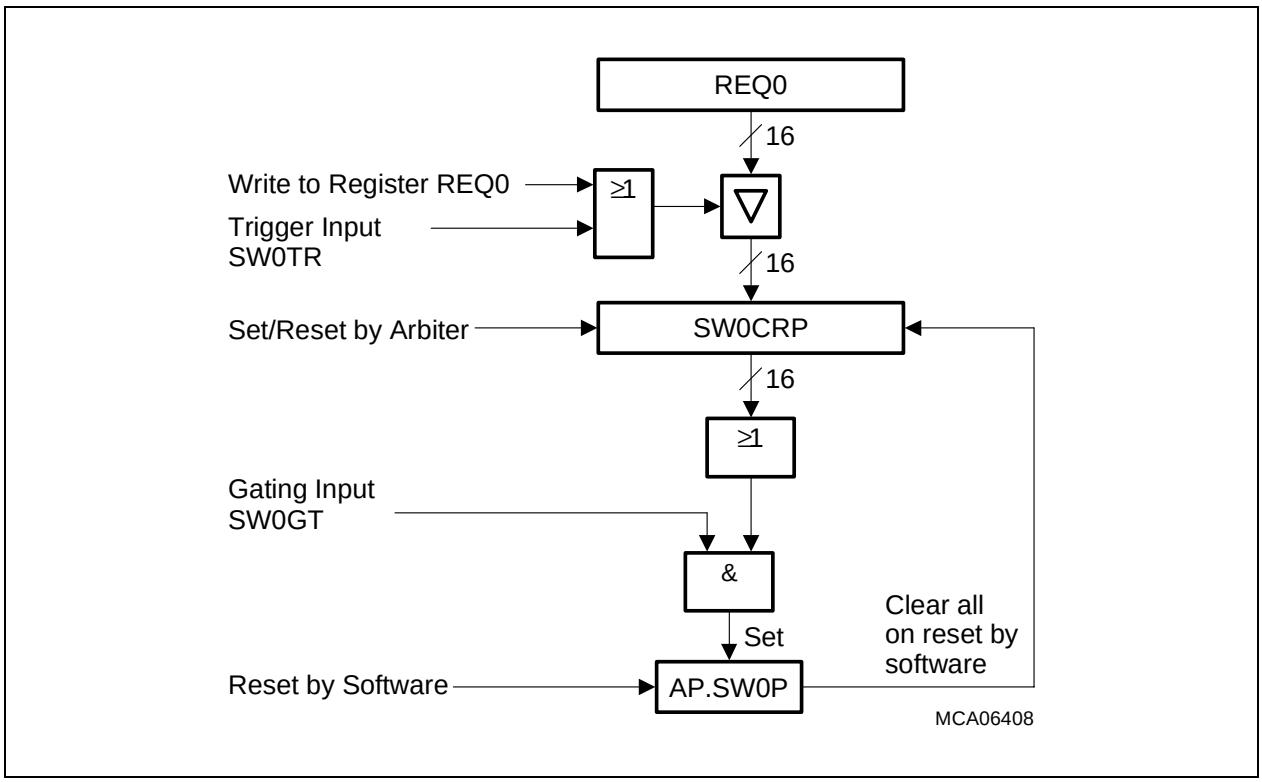


Figure 23-8 Conversion Request Source “Software”

One or more request bits can be set at a time by software, resulting in a conversion request for the designated analog channel(s). Writing to the register REQ0 automatically loads its contents into register SW0CRP. The contents of REQ0 remains unchanged after a load operation. A trigger pulse (SW0TR = 1) also leads to this load operation.

If at least one bit is set in REQ0 when the software gating input SW0GT = 1, the arbitration participation flag AP.SW0P is set. This informs the arbiter to include the conversion request source “Software” in the arbitration. If “Software” is the arbitration winner, a conversion is started for the conversion request within register SW0CRP with the highest channel number. Starting a conversion causes the conversion request bit in register SW0CRP to be cleared by the arbiter. If a currently running “Software” initiated conversion is cancelled, the arbiter sets the corresponding conversion request bit in register SW0CRP for this channel. If all pending conversion requests are processed, the arbitration participation flag AP.SW0P becomes 0.

The contents of register SW0CRP can be reset under software control either bit-wise by writing a 0 to the corresponding bit position in register REQ0 or globally by clearing the “Software” arbitration participation flag.

The SW0TR and SW0GT input connections depend on the product specific implementation of the ADC module. They are described in **Section 23.3.6.2**.

23.1.2.6 Parallel Conversion Request Source “Auto-Scan”

The conversion request source “Auto-scan” allows continuous conversions of a selectable group of analog channels with almost zero software effort in generating and controlling these conversion requests. Auto-scan provides a single conversion sequence mode as well as continuous conversion sequence mode. Each analog channel can be configured individually to participate in an auto-scan sequence.

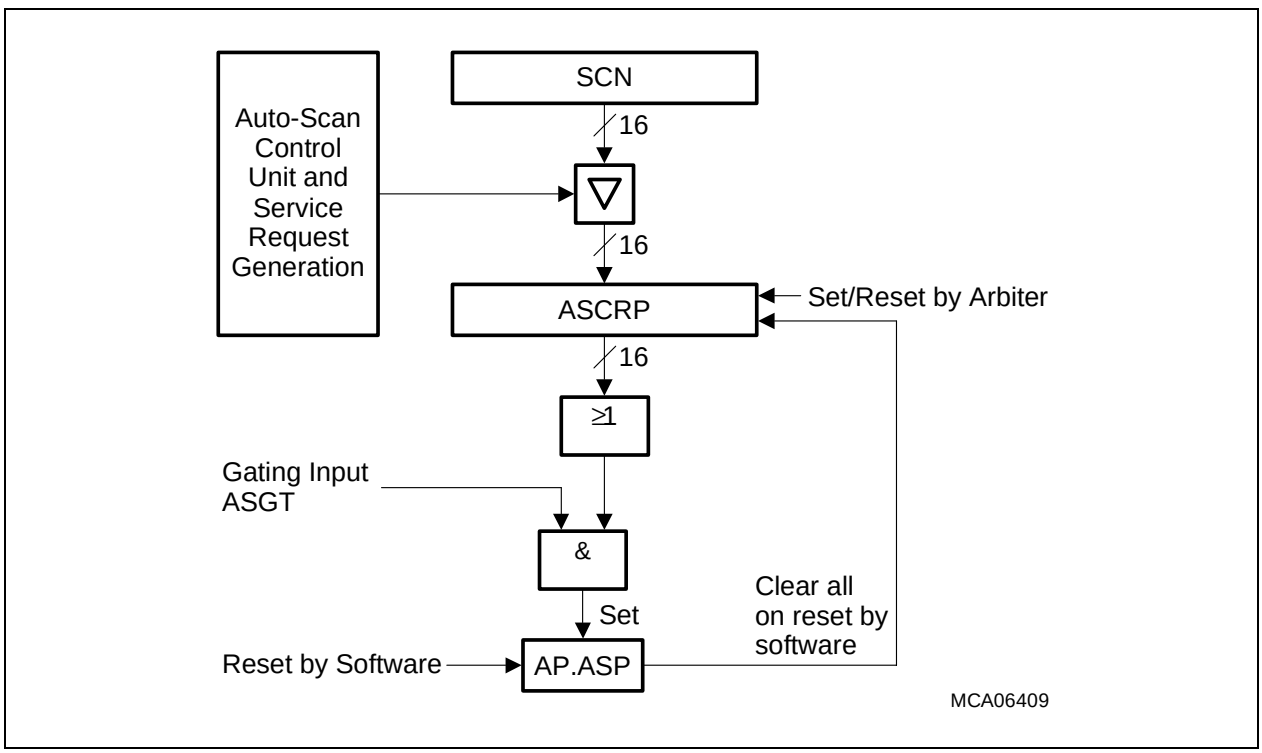


Figure 23-9 Conversion Request Source “Auto-Scan”

The group of analog channels to be auto-scanned is determined by setting the corresponding channel request flags SRQn in the auto-scan conversion request register SCN. The auto-scan sequence is started by programming an auto-scan mode in the A/D Converter control register (bit field CON.SCNM). Selecting an auto-scan mode loads the contents of the auto-scan conversion request register SCN into the auto-scan conversion request pending register ASCRP.

If at least one bit is set in the auto-scan conversion request pending register when the auto-scan gating line ASGT = 1, the arbitration participation flag AP.ASP is set. This informs the arbiter to include the conversion request source “Auto-scan” into the arbitration. If “Auto-scan” is the arbitration winner, a conversion is started for the conversion request within register ASCRP with the highest channel number. Pending conversion requests for the auto-scan channels are processed in the sequence from the

highest to the lowest channel number. Starting a conversion causes the conversion request pending flag in register ASCRP to be cleared.

The auto-scan sequence is complete if the channel with the lowest number selected to be auto-scanned has been converted (all bits in register ASCRP are cleared).

In single conversion sequence mode, the bit field CON.SCNM is automatically cleared and the conversion request source “Auto-scan” is disabled.

In continuous conversion sequence mode, the conversion request source “Auto-scan” automatically requests a new auto-scan sequence. Results previously stored in the specific channel status register(s) will be overwritten. Continuous auto-scan sequence is performed until auto-scan is stopped under software control.

If a currently running “Auto-scan” initiated conversion is cancelled, the arbiter sets the corresponding conversion request bit in register ASCRP for this channel.

After the conversion of the last channel within an auto-scan sequence has been finished, the source service request flag MSS1.MSRAS is set. Auto-scan service requests can be generated only if the auto-scan service request is enabled by SRNP.ENPAS = 1.

The ASGT input connection depends on the product specific implementation of the ADC module. It is described on **Page 23-96**.

The auto-scan control functionality is described in **Table 23-3**, **Table 23-4** and **Table 23-5**. This description of the auto-scan includes the actions to be performed on changes in the auto-scan mode or the channels to be auto-scanned as well as clearing the auto-scan arbitration participation flag.

Table 23-3 describes the action to be performed on a change of bit field CON.SCNM.

Table 23-3 Change of Auto-Scan Mode Bit Field CON.SCNM

Value of CON.SCNM		Action
Current Value of CON.SCNM	**Value after Write Action to CON.SCNM**	
00_B	00_B	No action
	01_B	Load SCN contents into register ASCRP, set bit AP.ASP, and start single auto-scan sequence if at least one channel is specified in register SCN to participate in auto-scan mode; otherwise, clear bit field CON.SCNM.
	10_B	Load SCN contents to register ASCRP, set bit AP.ASP, and start continuous auto-scan sequence if at least one channel is specified in register SCN to participate in auto-scan mode; otherwise, clear bit field CON.SCNM.
	11_B	Clear bit field CON.SCNM
01_B	00_B	Finish currently performed auto-scan sequence and generate a service request (if enabled) at the end of the sequence.
	01_B	Continue to perform auto-scan sequence and generate a service request (if enabled) at the end of the sequence.
	10_B	Finish currently performed auto-scan conversion, generate a service request (if enabled) if this is the last channel of the auto-scan sequence. Load SCN contents in register ASCRP and start a continuous auto-scan sequence.
	11_B	Clear bit field CON.SCNM, finish auto-scan sequence, and generate service request (if enabled) at the end of the sequence.

Table 23-3 Change of Auto-Scan Mode Bit Field CON.SCNM (cont'd)

Value of CON.SCNM		Action
Current Value of CON.SCNM	**Value after Write Action to CON.SCNM**	
10_B	00_B	Finish auto-scan sequence and generate service request (if enabled) at the end of the sequence.
	01_B	Finish currently performed auto-scan conversion and generate a service request (if enabled) at the end of the conversion if this was the last channel of the sequence. Load SCN contents to register ASCRP and start single auto-scan sequence.
	10_B	Continue to perform continuous auto-scan sequence and generate a service request (if enabled) at the end of the sequence. Load SCN contents to register ASCRP and start continuous auto-scan sequence.
	11_B	Clear bit field CON.SCNM and finish auto-scan sequence. Generate a service request (if enabled) at the end of the sequence.

Table 23-4 shows the actions to be taken on a change of the auto-scan conversion request register SCN.

Table 23-4 Change of the Auto-Scan Conversion Request Register SCN

Value of SCN		Action
Current Value of SCN	**Value after Write Action to SCN**	
≠0	0000_H	Bit field CON.SCNM is cleared independently of the auto-scan mode. Finish currently performed auto-scan sequence and generate a service request (if enabled) if this was the last channel of the sequence. No new auto-scan sequence is started.
	≠0	**In case of CON.SCNM = 00_B, 01_B, or 11_B:** Clear bit field CON.SCNM. Finish currently performed auto-scan sequence and generate a service request (if enabled) if this was the last channel of the sequence. No new auto-scan sequence is started.
		In case of CON.SCNM = 10_B: Finish the currently performed auto-scan conversion and generate a service request (if enabled) if this was the last channel of the sequence. Start a new continuous auto-scan sequence.

Table 23-5 shows the actions to be taken on a change of the auto-scan arbitration participation flag.

Table 23-5 Change of the Auto-Scan Arbitration Participation Flag AP.ASP

Current Value of ASP	Write to ASP	Action
0	0	No action
	1	No action
1	0	**In case of bit field CON.SCNM = 00_B, 01_B, or 11_B:** Bit field SCNM is cleared. Finish currently performed auto-scan conversion. Generate a service request (if enabled) if this was the last channel of auto-scan sequence.
		In case of bit field CON.SCNM = 10_B: Finish currently performed auto-scan conversion and generate a service request (if enabled) if this was the last channel of auto-scan sequence. Start new continuous auto-scan sequence.
1	1	Don’t care

23.1.2.7 Sequential Conversion Request Source “Channel Injection”

The conversion request source “Channel Injection” generates sequential conversion requests for analog channels either with Wait-Inject or Cancel-Inject-Repeat functionality. “Channel Injection” consists of the channel injection control register, the back-up register, and the channel injection arbitration participation flag.

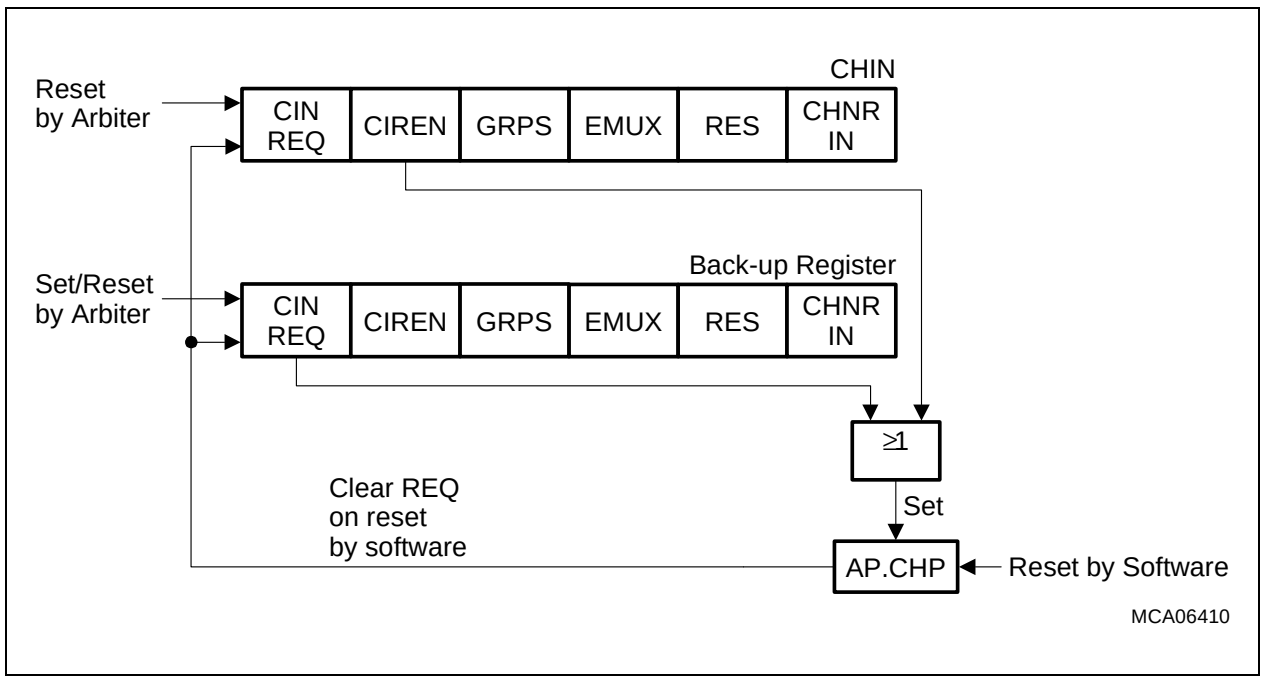


Figure 23-10 Conversion Request Source Channel Injection

The channel injection control register CHIN contains a conversion request bit CINREQ, a control bit CIREN for selecting the cancel inject repeat feature, control information for analog input group selection (GRPS) and external multiplexer settings (EMUX), control information (RES) for resolution selection, and the channel number (CHNRIN) of the ADC channel to be converted.

Setting the channel injection request bit CINREQ causes the arbitration participation flag AP.CHP to be set. This informs the arbiter to include the conversion request source “Channel Injection” into arbitration. If “Channel Injection” is the arbitration winner, a conversion is started for the analog channel specified in the channel injection control register CHIN. The settings of the external multiplexer and the resolution of the ADC are also taken from this register.

Starting a conversion causes the channel injection request bit CINREQ to be cleared. The channel injection arbitration participation flag AP.CHP is automatically cleared if the channel injection control register and the back-up register contain no valid request.

If a currently running conversion initiated by “Channel Injection” is cancelled, the arbiter restores the conversion information in a back-up register for this channel. In this context, conversion information refers to the conversion request bit, the setting for the external

multiplexer, and the settings of the ADC's resolution. If the back-up register contains valid conversion information, the arbiter reads from the back-up register instead from the channel injection control register. Thus, the previously cancelled conversion participates in arbitration once again. A new conversion requested via the conversion request control register will be performed after the request in the back-up register is served.

The request bit of the channel injection control register and the backup register can be cleared under software control. Clearing the arbitration participation bit clears either the request bit in the request register (if the back-up register contains no request) or the request bit in the back-up register (if the back-up register contains a valid request).

As already mentioned, "Channel Injection" generates sequential conversion requests for analog channels either with the Inject-Wait or the Cancel-Inject-Repeat functionality.

- **Channel Injection with Inject-Wait** provides the means to wait until the current conversion with higher priority is finished before the requested conversion is injected.
- **Channel Injection with Cancel-Inject-Repeat** "Cancels" a currently performed conversion, "Injects" the requested conversion, and finally "Repeats" the previously cancelled conversion. The Cancel-Inject-Repeat feature is enabled if bit CHIN.CIREN is set. When using this feature, the currently performed conversion is cancelled if its source arbitration level is lower than the source arbitration level of channel injection. If a currently performed conversion is cancelled, a new request is generated for this conversion. Thus, the previously cancelled conversion participates in the arbitration again.

The following examples provides an overview on the behavior of the conversion request source "Channel Injection".

Figure 23-11 shows the functionality of conversion requests generated by "Channel Injection" with Inject-Wait feature. The conversion requested with a source-arbitration-level of 'L3' waits until the currently performed conversion with a source-arbitration-level of 'L1' is finished. The second channel injection request is delayed until both conversions requested with a source-arbitration-level of 'L2' are finished.

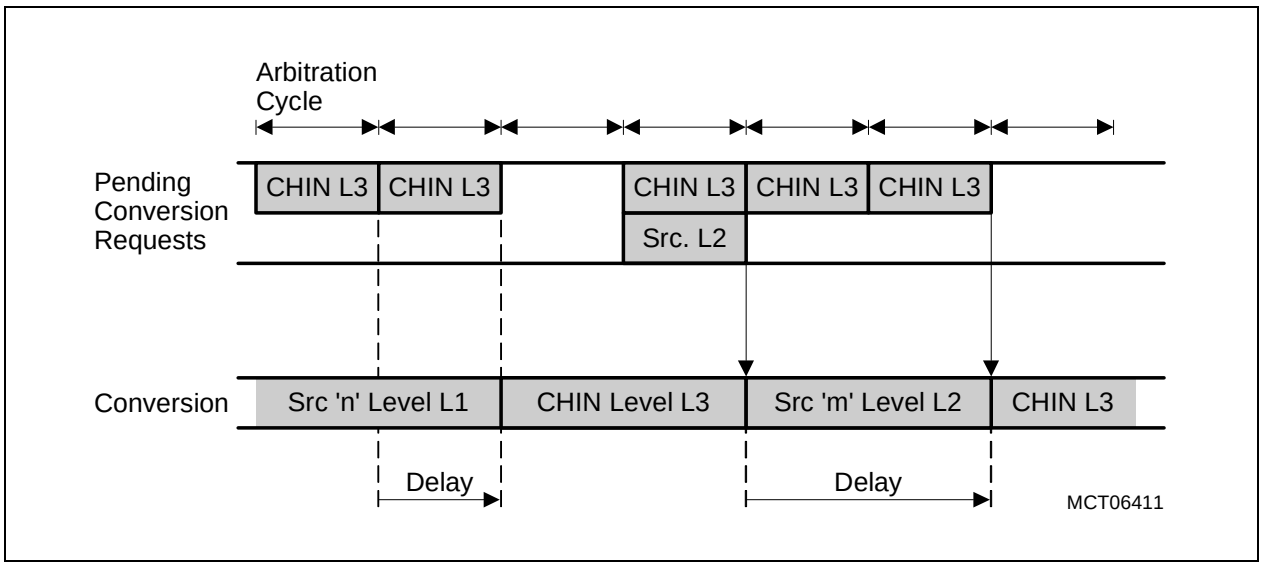


Figure 23-11 Channel Injection with Inject-Wait

Figure 23-12 shows the behavior of conversion requests generated by “Channel Injection” using the Cancel-Inject-Repeat feature. In the first case, the currently performed conversion is cancelled, because its source arbitration level of ‘L2’ is below the source arbitration level of ‘L1’ of “Channel Injection”. A new conversion request is generated for the cancelled conversion in order to restart this cancelled conversion later. This new request participates in arbitration and will be selected for repetition due to its priority level. The second injection request with a source arbitration level of ‘L4’ is delayed, even if the Cancel-Inject-Repeat feature is enabled.

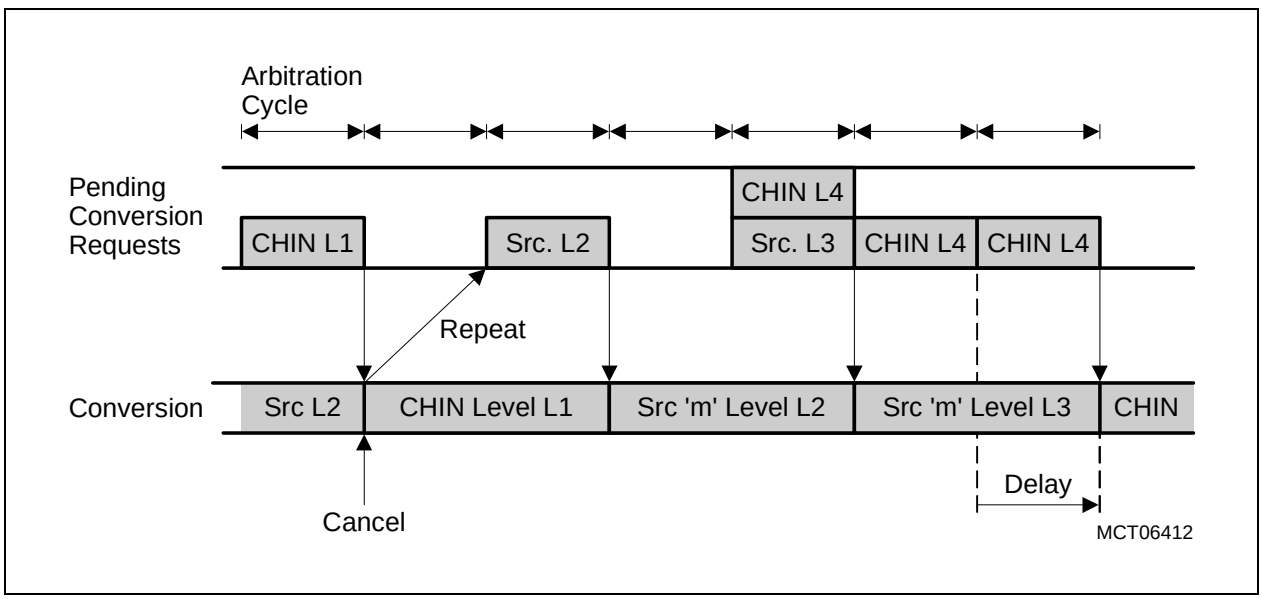


Figure 23-12 Channel Injection with Cancel-Inject-Repeat Feature

Figure 23-13 shows the cooperation of conversions requested by “Channel Injection” and conversions triggered by “Timer” running in Arbitration Lock Mode. First, a conversion is requested by “Channel Injection” with a source arbitration level of ‘L3' using the Cancel-Inject-Repeat feature, during which the arbitration is locked by the timer. This request is delayed until the timer triggered conversion is finished or until “Channel Injection” is programmed to a higher priority than the timer. Second, a conversion is requested by “Channel Injection” with a source arbitration level of ‘L1’ with the Cancel-Inject-Repeat feature selected, during which the arbitration is locked by the timer. In this case, the arbitration lock is not taken into account because the timer was programmed on source arbitration level ‘L2’. Even a currently running timer triggered conversion would have been cancelled and participates in arbitration anew.

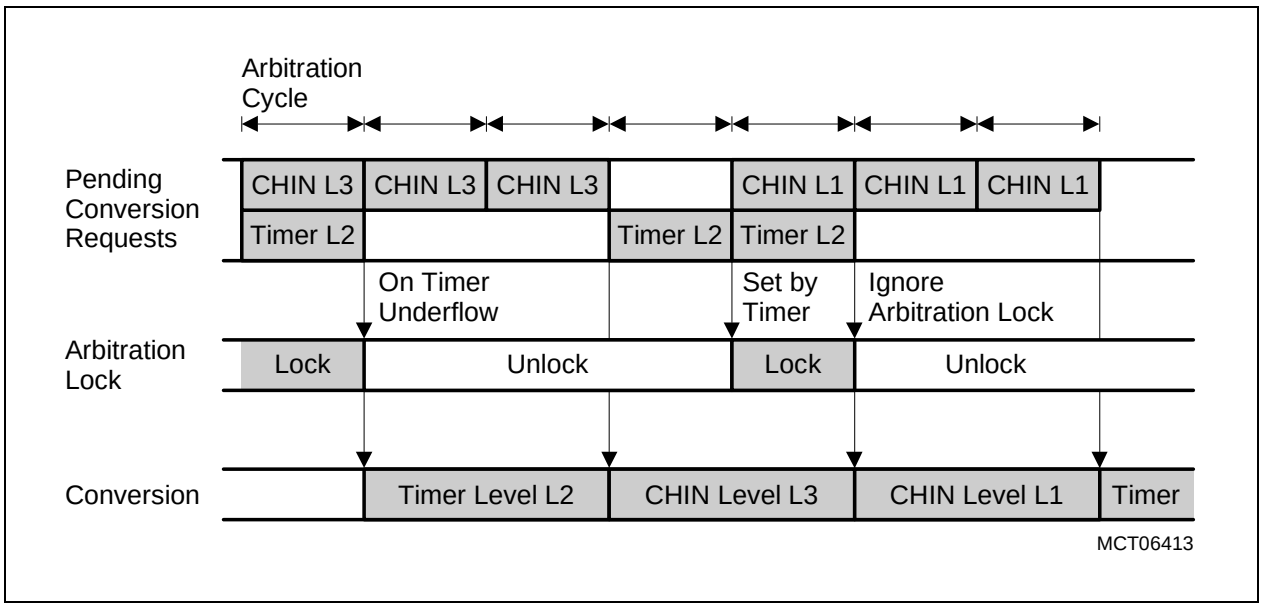


Figure 23-13 Channel Injection and Timer Triggered Conversion

23.1.2.8 Sequential Conversion Request Source “Queue”

The conversion request source “Queue” with its queue storage block is designed to handle and store burst transfers of conversion requests. Dedicated queue filling-state control logic can be used to request the next burst transfer of data while the queue’s filling level is below a predefined level.

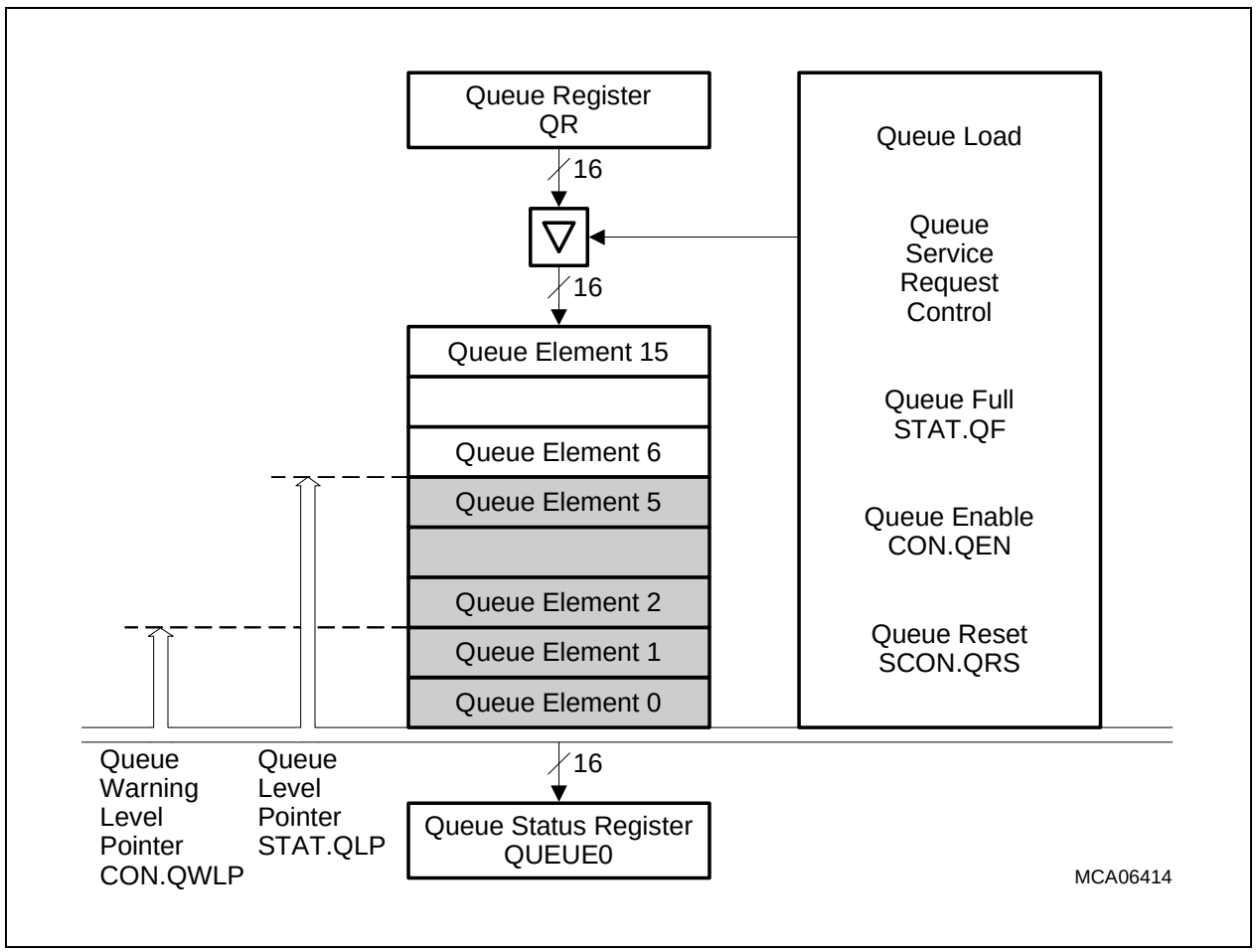


Figure 23-14 Queue Storage Block Diagram

As shown in **Figure 23-14**, the queue consists of a queue register QR, sixteen queue elements, queue status register QUEUE0, and the queue control logic.

The queue control logic includes the queue load logic, a queue level pointer, a queue warning limit pointer, the queue based service request control block, as well as control and status flags to monitor and control the queue state.

The queue register, the queue status register, and each of the sixteen queue elements contain a valid bit (V), an analog input group selection bit (GRPS), external multiplexer control bits (EMUX), A/D conversion resolution control bits (RES), and the channel number for which a conversion should be started (CHNR).

The queue is automatically filled by writing valid data to the queue register QR. Valid data means that at least the V bit is set, while “zero” is a valid option for the external multiplexer setting, the resolution control bit field and the channel number. Valid data in the queue register (QR.V, QR.GRPS, QR.EMUX, QR.RES and QR.CHNR data) is then copied to the next empty queue element determined by the queue level pointer STAT.QLP. The queue load operation causes the valid bit in the queue register to be cleared automatically. Any software access to the queue register is denied during this copy operation. No queue load is performed if the queue state is full (STAT.QF is set) and the queue register contains valid data.

As shown in **Figure 23-14**, queue elements zero to five contain valid data; therefore, the queue register’s contents are copied to queue element six.

The queue level pointer STAT.QLP indicates the number of valid queue elements. It is incremented after a queue load operation. It is decremented after a queue based conversion is started or after the queue participation flag is cleared. STAT.QLP is cleared after a queue reset operation by setting the queue reset bit. Note that there are sixteen valid queue elements in the queue if the queue level pointer is $0F_H$ and the queue full bit is set.

The queue warning limit pointer CON.QWLP can be used to generate service requests, based on a queue element state change. The value of the queue warning limit pointer must be programmed with a value “n” in order to focus on a state change from valid to invalid of queue element “n”. A queue based service request can be triggered in this case, thus requesting the next transfer of data to the queue. If the queue element specified by (CON.QWLP)+1 becomes invalid after a conversion, the module service request flag MSS1.MSRQR is automatically set. The service request destination node pointer (PQR) must be configured and enabled (ENPQR) in order to trigger a service request node assigned to the queue.

The conversion request source “Queue” consists of the queue status register QUEUE0, a backup register, and a queue arbitration participation flag AP.QP, as shown in **Figure 23-15**. The contents of queue element number zero are represented in the queue status register QUEUE0. Therefore, set/clear actions of the valid-bit of the queue status register QUEUE0 are also performed on queue element zero.

If at least one queue element contains valid data, this (these) valid bit(s) cause(s) the queue arbitration participation flag to be set. This informs the arbiter to include the conversion request source “Queue” into arbitration. If “Queue” is the arbitration winner, a conversion is started for the analog channel specified within the queue status register. The settings of the external multiplexer and the resolution of the A/D Converter are also derived from this register.

Starting a queue based conversion causes the valid bit of the queue status register QUEUE0 to be cleared by the arbiter. The contents of all queue elements containing valid data slides one step down. For example, queue element one contains valid data, this data “slides down” to queue element zero. Queue based conversion requests are

generated for the control information of register QUEUE0, if the queue is enabled (bit CON.QEN = 1), the queue status register contains valid data (QUEUE0.V is set) and the queue gating line QGT = 1. The arbitration participation flag is automatically cleared if all queue elements, the queue status register (remember: QUEUE0 represents the contents of queue element number zero) and the back-up register contain no valid request.

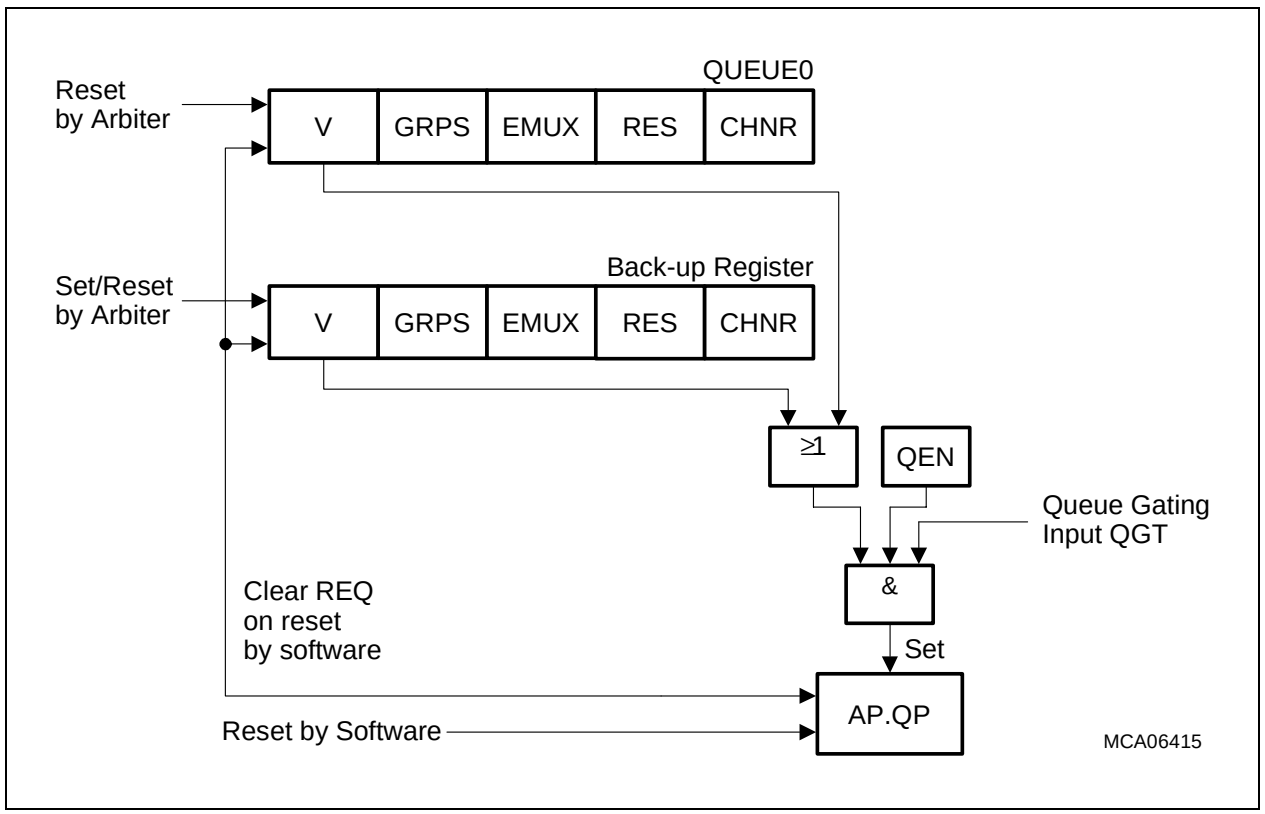


Figure 23-15 Conversion Request Source “Queue”

If a currently running conversion initiated by “Queue” is cancelled, the arbiter restores the conversion information in the back-up for this channel. In this context, conversion information refers to the conversion request bit, the setting for the external multiplexer and the settings of the A/D Converter’s resolution. If the back-up register contains valid conversion information, the arbiter reads from the back-up register instead of the queue status register. Thus, the previously cancelled conversion participates in arbitration once again. A conversion requested via the queue storage block (register QUEUE0) will be performed after the request in the back-up register is served.

The valid bit (V bit) of the queue status register and the back-up register can be cancelled under software control. Clearing the queue arbitration participation bit clears either the valid bit in the queue status register (if the back-up register contains no request) or the request bit in the back-up register (if the back-up register contains a valid request). If the valid bit of the queue status register is cleared, a slide operation is performed equal to the slide operation after starting a queue based conversion. The

enable bit CON.QEN can be set by hardware if the queue trigger request line QTR = 1. The QTR and QGT input connections depend on the product specific implementation of the ADC module. They are described in **Section 23.3.6.2**.

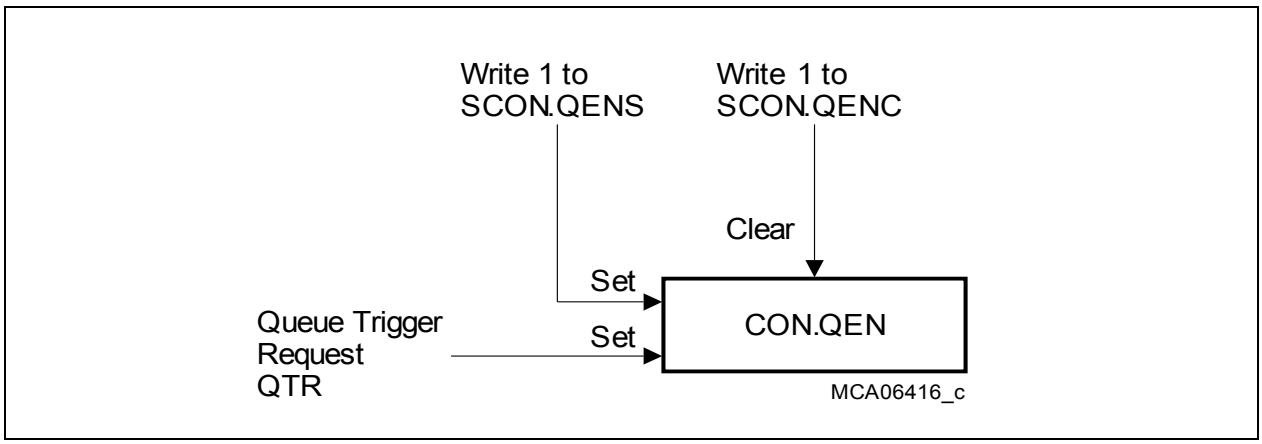


Figure 23-16 Modification of Bit CON.QEN

23.1.3 Conversion Request Arbitration

Because several conversion request sources can generate conversion requests at the same time, an arbitration mechanism is implemented in order to detect the conversion request source and channel with the highest priority. **Figure 23-17** shows the arbitration scheme with the associated controls.

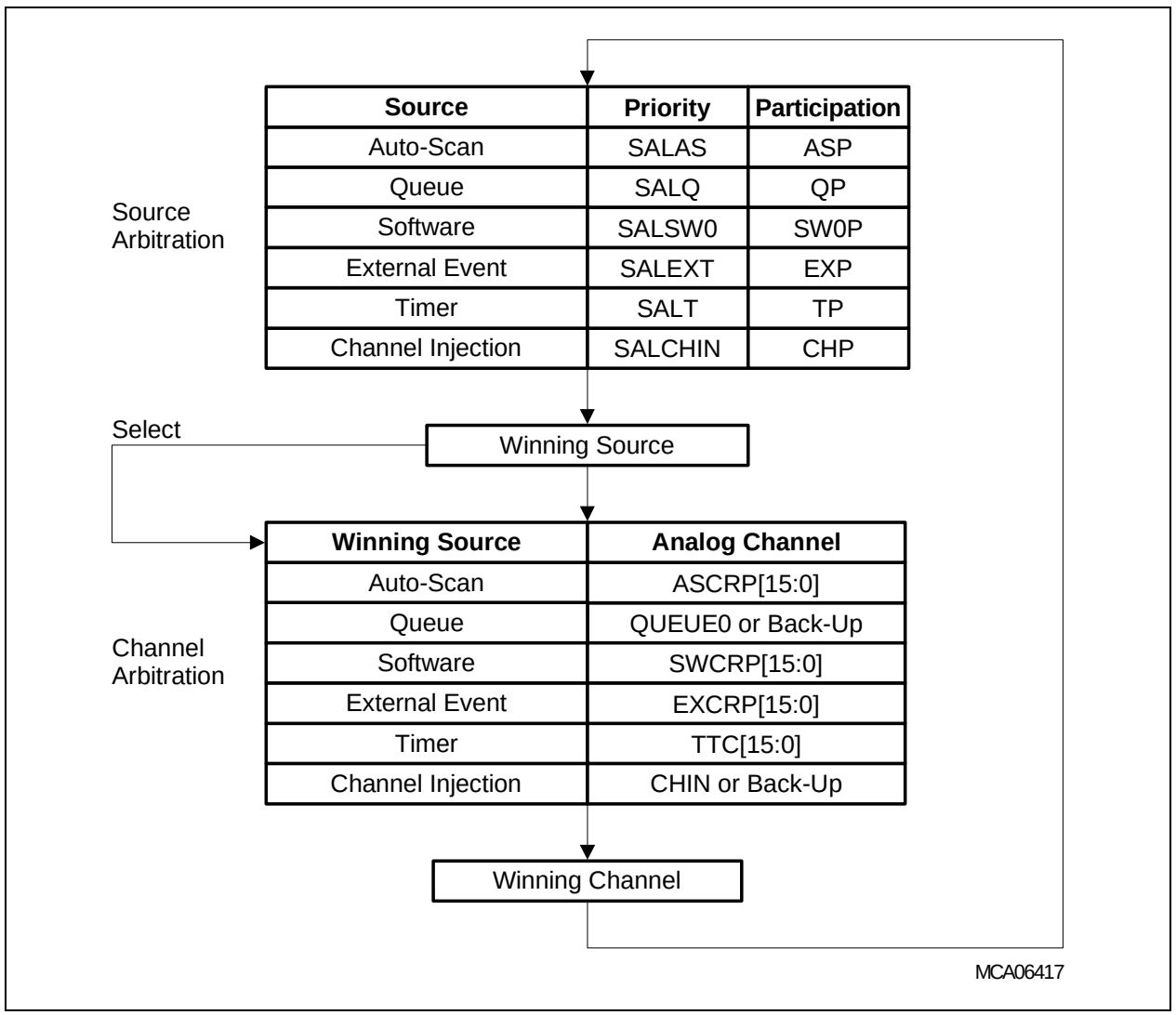


Figure 23-17 Arbitration

Arbitration of pending conversion requests is performed according to the following two stage prioritization algorithm:

- **Source arbitration** is the first stage in the arbitration algorithm. Starting with the conversion request source “Auto-Scan” up to “Channel Injection”, each source is checked if its arbitration participation flag is set. If the participation flag is set and its priority is higher than the priority of the other selected sources, that source is the winner of the arbitration.

- **Channel arbitration** follows after source arbitration. For the winning source, channel arbitration is performed. Within the second stage of the arbitration algorithm, the pending conversion request with the highest priority is detected. If a parallel source is the winning source, the flag representing the highest channel number within the conversion request pending register is determined. If a sequential source is the winning source, the channel in the request register or in the back-up register is determined. Note that a pending request in the back-up register is preferred.

The arbitration result consists of the winning source and channel number. A start of conversion can occur, if the A/D Converter is idle or if the arbitration winner has permission to cancel a currently running conversion. After the conversion has started, the corresponding pending conversion request is automatically reset. Attempt to start a conversion for this arbitration result will be repeated until either the start is successful (other conversion is currently running) or a new result (source and channel number with a higher priority) was arbitrated.

23.1.3.1 Source Arbitration Level

The priority of each conversion request source can be programmed individually in the corresponding bit fields of the source arbitration level register SAL. The priority of a source is named as source arbitration level and it determines the order in which pending conversion requests from different sources are performed. A low number of the source-arbitration-level represents a high priority and vice versa.

After initialization, an individual source arbitration level is assigned to each source. “Channel Injection” has the highest priority, while “Auto-Scan” has the lowest priority. These predefined priority levels can be reprogrammed to adapt the ADC’s functionality to the requirements of the application.

It is recommended that source arbitration levels should be reprogrammed while no conversion request is pending, as any modification of the source arbitration level register immediately affects the arbitration scheme. Each source should have an individual priority level. Nevertheless, if several conversion request sources have been programmed to the same priority level, the first detected source within this group of identical levels is taken into account.

23.1.3.2 Arbitration Participation Flags

Each source has an arbitration participation flag located in the arbitration participation register AP. An arbitration participation flag set to 1 indicates that at least one conversion request has been generated by this source and that this source participates in the arbitration.

An arbitration participation flag is automatically cleared by hardware if no conversion request is pending for this source (if all requested conversions have been started).

The arbitration participation flag can also be cleared under software control. Writing a 0 to the corresponding flag clears the arbitration participation flag. All bits in the corresponding conversion request pending register are cleared if a participation flag of a parallel source is cleared under software control. If a participation flag of a sequential source is cleared, the following action is performed:

- **ONLY** the request bit of the back-up register is cleared, **if the back-up register contains valid data**. The request bit of the corresponding conversion request register (CHIN or QUEUE0) is **not** cleared in this case.
- **OR** the request bit of the corresponding conversion request register (CHIN or QUEUE0) is cleared, **if the backup register does not contain valid data**.

Note: Writing a 1 to a participation bit has no effect.

23.1.3.3 Cancel Functionality

Channel Injection has the ability to cancel a currently running conversion. If a conversion is cancelled, one of the following actions are performed:

- If a conversion initiated by a parallel source is cancelled, the conversion request flag is automatically set again in the corresponding conversion request pending register.
- If a conversion initiated by a sequential source is cancelled, the control information (such as resolution, external multiplexer information, etc.) of the cancelled conversion is rescued into the backup register (for example: queue based conversion is cancelled, so the queue backup register receives the control information of the cancelled conversion).

Following that, the request participates in the arbitration anew and will be served according to its source-arbitration level.

23.1.3.4 Clear of Pending Conversion Requests

This feature can be used to save conversion time by handling more than one conversion request at the same time.

Clear of Pending Conversion Requests in Parallel Sources

If several conversion requests are pending for the same analog channel and a conversion for this analog channel has been started, all pending conversion requests of **parallel** sources can be cancelled for this analog channel by the arbiter (CON.CPR = 1). For example, if timer, software and auto-scan trigger a conversion request for the same analog channel, only one conversion is started for this analog channel. The other two pending conversion requests will be automatically cancelled by the arbiter. Note that the conversion will be started for the arbitration winner, the source with the highest priority. The conversion result is valid for all parallel sources which requested this channel. A service request is generated only for the source that caused the processed conversion.

Individual Clear of Pending Conversion Requests

If several conversion requests are pending for the same analog channel, this channel will be converted several times until all pending conversion requests are performed. This is the default setting after reset (CON.CPR = 0).

23.1.3.5 Arbitration Lock

If the timer runs in Arbitration Lock Mode and the current timer value TSTAT.TIMER is equal to or below the arbitration lock boundary, the arbitration lock bit STAT.AL is set. Setting the arbitration lock bit also sets the timer participation flag. In this way, the timer source can participate in the arbitration cycle without any pending request. Such an arbitration participation by the timer without a pending request denies all currently pending sources that have a source-arbitration-level below the timer source as arbitration winner. All sources with a source-arbitration-level greater than the timer source retain their possibility to win the arbitration. If the timer wins the arbitration without a pending request, no conversion will be started for this arbitration winner. This case can occur if bit AP.TP is set while no bit is set in register TCRP. This feature can be used to guarantee that no conversions can be started for lower prioritized sources.

Note: The timer participation flag is also set by any pending timer conversion request in register TCRP.

Note: If any source has the same source-arbitration-level as the timer source, the result of the arbitration cycle depends on the position of this source compared to the timer source. If this source is checked before the timer source, this can be the arbitration winner. If this source is checked after the timer source, this source cannot be the arbitration winner.

23.1.4 Clock Circuit

The clock divider blocks shown in **Figure 23-18** determine the clock frequencies in the ADC module and the conversion and sample timing.

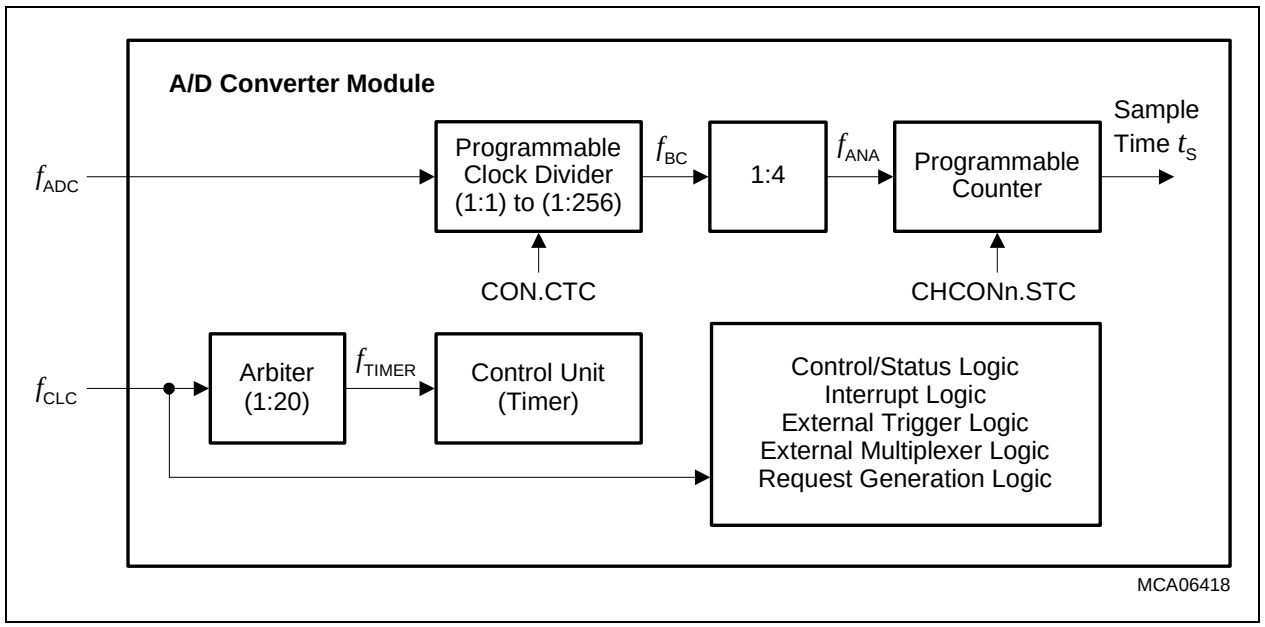


Figure 23-18 Clock Control Structure

The following definitions for the A/D Converter clocks are used in this chapter:

- f_{CLC}: Module control clock
- f_{ADC}: Module timing clock
- f_{BC}: Basic operating clock
- f_{ANA}: Internal A/D Converter clock
- f_{TIMER}: Arbiter clock

The conversion time is composed of the sample time, the time for the successive approximation, and the calibration time. **Table 23-6** shows the conversion time t_C based on the sample time t_S, basic operating clock frequency f_{BC} and the module timing clock f_{ADC} (t_{BC} = 1 / f_{BC}, t_{ADC} = 1 / f_{ADC}).

Table 23-6 Conversion Time t_C

A/D Converter Resolution	Conversion Time t_c
8-bit	t_S + 40 t_{BC} + 2 t_{ADC}
10-bit	t_S + 48 t_{BC} + 2 t_{ADC}
12-bit	t_S + 56 t_{BC} + 2 t_{ADC}

Note: The TC1766 basic operating clock frequency f_{BC} influences the maximum allowable internal resistance of the used reference voltage supply.

23.1.4.1 Conversion Principles

After reset, a power-up calibration is automatically performed in order to correct gain and offset errors of the A/D Converter. The ongoing power-up calibration is indicated in the A/D Converter status register by an activated calibrate bit STAT.CAL. To achieve best calibration results, the reference voltages as well as the supply voltages must be stable during the power-up calibration.

When a conversion is started, the capacitances of the converter are loaded via the respective analog input channel to the analog input voltage. The time to load the capacitances is referred to as sample time t_S. The sample phase is indicated by an activated status bit STAT.SMPL in the A/D Converter status register. Next, the sampled voltage is converted to a digital value. Finally, an internal self calibration adapts the analog converter module to changing temperatures and device tolerances. The conversion and calibration phase is indicated by the busy signal STAT.BUSY, which goes inactive at the end of the calibration phase.

Note: During the power-up calibration, no conversion should be started.

23.1.4.2 Conversion Timing Control (CTC and CPS)

The module clock f_{ADC} is generated in the ADC Clock Generation unit (see **Page 23-88**). The A/D Converter's basic operating clock frequency f_{BC} is derived from f_{ADC} via the programmable clock divider CTC, which provides dividing factors from 1:1 to 1:256.

$$f_{BC} = \frac{f_{ADC}}{CTC + 1} \tag{23.2}$$

The A/D Converter's basic operating clock frequency f_{BC} must not exceed 40 MHz. It must also not drop below 2 MHz.

The internal A/D Converter clock frequency f_{ANA} is a quarter of the basic operating clock frequency f_{BC} (min. 0.5 MHz, max. 10 MHz). The internal A/D Converter clock f_{ANA} is related to f_{ADC} according to the following equation:

$$f_{ANA} = \frac{f_{BC}}{4} = \frac{1}{4} \times \frac{f_{ADC}}{CTC + 1} \tag{23.3}$$

With the clock control bit field CON.CTC, the internal A/D Converter clock f_{ANA} can be adjusted to different module timing clock frequencies f_{ADC} in order to optimize the performance of the TC1766 A/D Converter. Note that CON.CTC may be changed during a conversion, but will be evaluated after the currently performed conversion is finished.

Table 23-7 Conversion Timing Control

CON.CTC	f_{BC}	t_{BC}	f_{ANA}	t_{ANA}
00_H	f_{ADC}	$1 / f_{ADC}$	$f_{ADC} / 4$	$4 / f_{ADC}$
01_H	$f_{ADC} / 2$	$2 / f_{ADC}$	$f_{ADC} / 8$	$8 / f_{ADC}$
02_H	$f_{ADC} / 3$	$3 / f_{ADC}$	$f_{ADC} / 12$	$12 / f_{ADC}$
03_H	$f_{ADC} / 4$	$4 / f_{ADC}$	$f_{ADC} / 16$	$16 / f_{ADC}$
…	…	…	…	…
FF_H	$f_{ADC} / 256$	$256 / f_{ADC}$	$f_{ADC} / 1024$	$1024 / f_{ADC}$

23.1.4.3 Sample Timing Control

The sample time control determines the duration of the sample phase of a conversion, that is, the period during which the channel input capacitance is charged/discharged by the selected analog signal source. The duration of the sample phase is programmed individually for each channel via sample time control bit field CHCONn.STC. Any modification of CHCONn.STC will be evaluated after the currently performed conversion is terminated.

The sample time t_S depends on the basic operating clock f_{BC} and the programmable value of bit field CHCONn.STC. The sample time t_S is selected in periods of $t_{BC} = 1 / f_{BC}$ within the range from $8 \times t_{BC}$ up to $1028 \times t_{BC}$.

The sample time t_S is calculated according to the following equation:

$$t_S = 4 \times (STC + 2) \times t_{BC} \quad (23.4)$$

Table 23-8 shows the selectable values of CON.STC and the resulting ADC basic operating clock f_{BC} and sample time t_S.

Table 23-8 Sample Time Control

CHCONn.STC	Sample Time t_S
00_H	$8 \times t_{BC}$
01_H	$12 \times t_{BC}$
02_H	$16 \times t_{BC}$
03_H	$20 \times t_{BC}$
…	…
FF_H	$1028 \times t_{BC}$

Note: The duration of the sample phase influences the maximum allowable internal resistance of the respective analog input signal source.

23.1.4.4 Power-Up Calibration Time

When the A/D Converter becomes clocked after a reset operation, a power-up calibration is automatically performed in order to correct gain and offset errors of the A/D Converter. A running power-up calibration is indicated by status flag STAT.CAL. To achieve best calibration results, the reference voltages as well as the supply voltages must be stable during the power-up calibration. The first A/D conversion can be started after the power-up calibration is finished (STAT.CAL = 0).

The power-up calibration takes 3840 clock cycles of the A/D Converter clock f_{ANA}. After a reset operation, the A/D Converter is disabled and becomes enabled when registers ADC0_CLC and ADC0_FDR are written with appropriate values. Power-up calibration starts when the analog part of the A/D Converter is clocked after a reset operation.

The following example shows the setup for the fastest achievable (best case) power-up calibration time at f_{SYS} = 80 MHz. Note that the maximum frequency of f_{ANA} must not exceed 10 MHz. See also **Page 23-88** for further details on the ADC module clock generation.

Example (f_{ADC} = 80 MHz):

- $f_{SYS} = f_{CLC}$ = 80 MHz
- $f_{ADC} = f_{CLC}$ / 2 = 40 MHz (Setting FDR for divider ratio 2)
- $f_{BC} = f_{ADC}$ / 1 = 40 MHz (CON.CTC = 00_H)
- $f_{ANA} = f_{BC}$ / 4 = 10 MHz (fixed divider, f_{ANA} max. frequency)
- $t_{ANA} = 1 / f_{ANA}$ = 0.1 µs

These values result in a best case power-up calibration time of 3840 × t_{ANA} = 384 µs.

The best case power-up calibration time as discussed above is of course increased by decreasing the f_{ANA} clock frequency. Note that the changing of clock related parameters is not recommended during a running power-up calibration.

23.1.5 Reference Voltages (V_{AREF} and V_{AGND})

The digital result of a conversion represents the analog input as a fraction of the reference (V_{AREF} - V_{AGND}) in steps of 2^{-n} by n-bit resolution:

$$\text{Result} = 2^n \times (V_{AIN} - V_{AGND}) / (V_{AREF} - V_{AGND}) \quad (23.5)$$

The ADC module offers the choice of four selectable reference voltages $V_{AREF}[0]$ to $V_{AREF}[3]$. The reference voltage can independently be selected for each analog channel via the respective bit field CHCONn.REF. $V_{AREF}[0]$ corresponds to the positive reference voltage V_{AREF} and is used for self calibration of the A/D Converter. Therefore, it must be stable during all conversions, even for those that use another reference voltage.

The reference voltages must fulfill the following specifications:

$$V_{AREF}[3:0] \leq V_{DDM} + 0.05\ V;\ V_{DDM} \leq 3.3\ V \quad (23.6)$$

A conversion with low reference voltage affects the accuracy of the A/D Converter. The TUE of an A/D Converter that is operated at a reduced positive reference voltage can be evaluated according to the following equations:

$$\text{TUE}|_A \rightarrow \text{TUE}|_B = K \times \text{TUE}|_A,\ (K \geq 1) \quad (23.7)$$

with factor K as:

$$V_{AREF}|_A \rightarrow V_{AREF}|_B = 1/K \times V_{AREF}|_A \quad (23.8)$$

where

- $V_{AREF}|_A$: minimum positive reference voltage range is specified for $0\ V \leq V_{AREF} \leq V_{DDM} + 0.05\ V$
- $V_{AREF}|_B$: positive reference voltage, which is below the specified range
- $\text{TUE}|_A$: total unadjusted error for reference voltages within the specified range
- $\text{TUE}|_B$: total unadjusted error for reference voltages below the specified range

Note: Due to the proximity of the SYSCLK output on P4.0 and the analog input pins, it is recommended to reduce the usage of the SYSCLK output functionality during the ADC/FADC operation. The performance of the ADC/FADC may be influenced by the SYSCLK activities. The magnitude of degradation of the ADC performance depends on the PCB layout and should be verified for specific applications.

Note: All unused analog input pins should be connected to a fixed potential either V_{AGND} or $V_{AREF}[0]$ to avoid disturbance of active analog inputs.

Note: It is not recommended in general to set V_{AREF} below 50% of V_{DDM}.

Note: The analog input voltages V_{AIN} must be in the range between V_{AGND} and the selected V_{AREF}.

23.1.6 Error through Overload Conditions

An additional error can occur when overloading an analog input (such as channel D). In this case, an additional leakage current exists between the analog input D and the adjacent analog inputs D ± 1, affecting the conversion result of an analog input channel D by an additional error AEL based on the additional sampled voltage V_{AEL} (Analog Error Leakage).

$$V_{AEL}|_{D\pm1} = R_{AIN}|_{D} \times |I_{OV}|_{D} \times k_A \quad (23.9)$$

The coupling factor k_A determines the physical relation of two adjacent analog inputs. The resulting error AEL is given by:

$$AEL|_{D\pm1} = \frac{R_{AIN}|_{D} \times |I_{OV}|_{D} \times k_A}{V_{AREF}} \quad (23.10)$$

where

- V_{AREF}: reference voltage for conversion
- $R_{AIN}|_{D}$: resistance of the analog input channel D
- $I_{OV}|_{D}$: overload current of the analog input D
- AEL: additional error caused by a leakage current, related to V_{AREF}
- k_A: coupling factor for the analog input D

Note: If AEL is calculated in bit units, AEL must be multiplied by 2^n-1.

23.1.7 Limit Checking

Limit checking provides the means to check conversion results on exceeding or becoming lower than a defined limit. The checking parameters can be configured individually for each analog channel. Service requests can be generated for each analog channel on limit checking results such as on a limit violation or on successful limit checks.

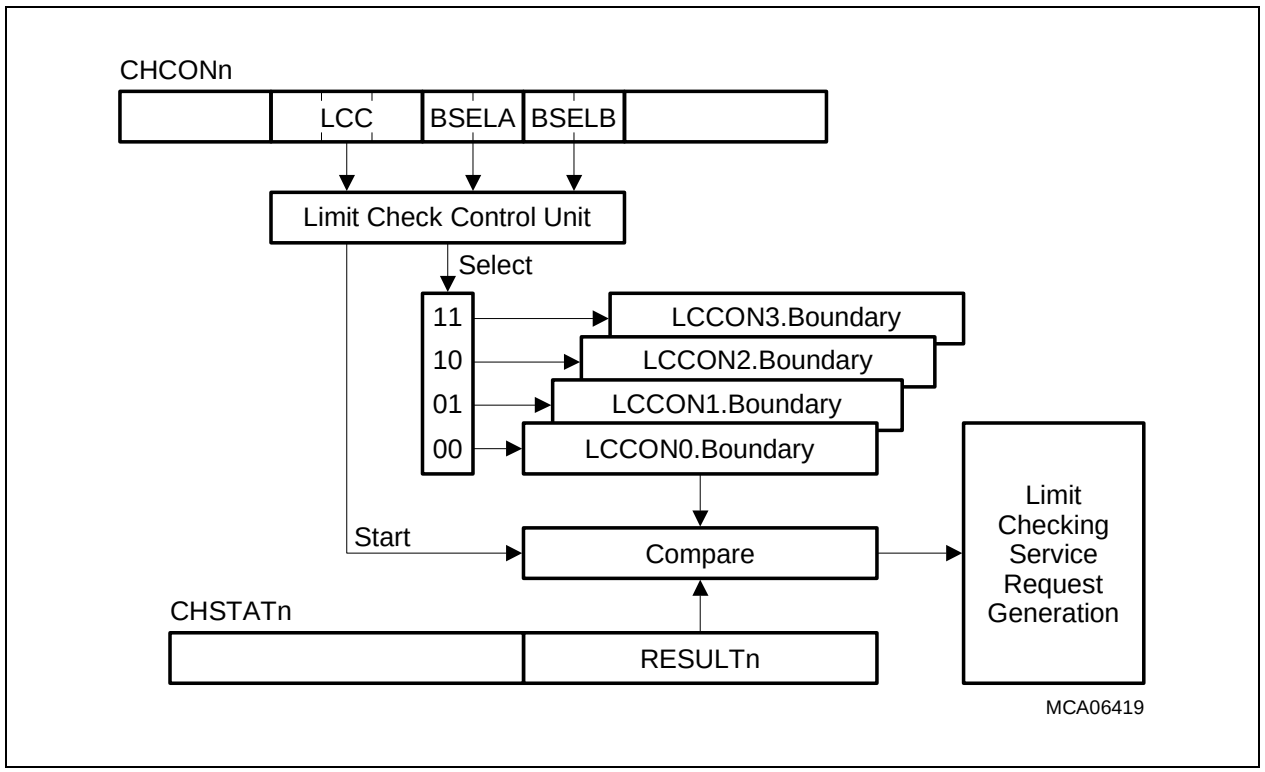


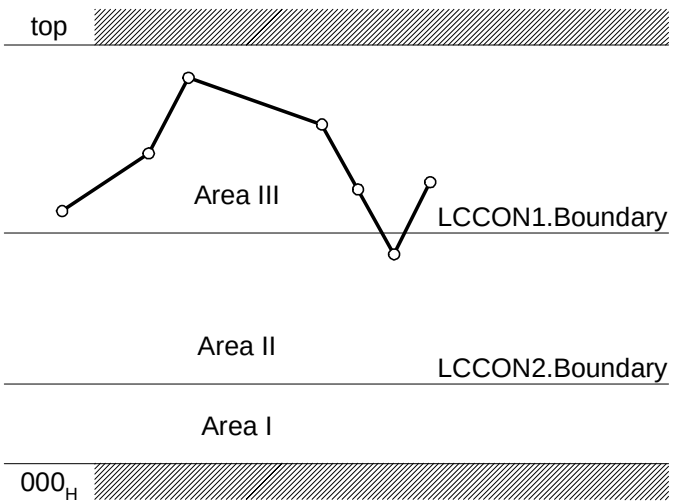
Figure 23-19 Limit Check Unit

As shown in **Figure 23-19**, a limit check is performed for the conversion result stored in a specific channel status register. For limit checking, the A/D Converter's measuring range is divided into three areas in order to check whether the conversion result meets the specified range. Two out of four boundaries can be selected and programmed per limit check. The boundaries are selected for each analog channel via the bit fields CHCONn.BSELA and CHCONn.BSELB, n = 0-15. Four boundaries can be set individually in the limit check control register LCCON0/1/2/3.

The limit check control bit field specifies if a limit check is performed for the current conversion result and which area must be met or avoided by the current conversion result (see **Figure 23-20**).

Depending on the selected limit check control parameter CHCONn.LCC, the service request flag is not set, is set if the selected area is hit, or is set if the selected area is missed for the related conversion result.

Figure 23-20 shows the selectable parameters for limit check.



LCC	Description
0 0 0	Neither Limit Check nor Interrupt
0 0 1	Not in Area I
0 1 0	Not in Area II
0 1 1	Not in Area III
1 0 0	Interrupt on Write Conv. Result
1 0 1	In Area I
1 1 0	In Area II
1 1 1	In Area III

Figure 23-20 Limit Checking

The A/D Converter's measuring range is divided into the following three areas:

- Area I: From 000_H to (including) the lower boundary
- Area II: excluding the lower boundary to (including) the upper boundary
- Area III: excluding the upper boundary to top (top due to the selected resolution)

The value stored in LCCONn.BOUNDARY represents a boundary, that is selected by the channel-specific bit fields CHCONn.BSELA/B. Neither boundary A (selected by CHCONn.BSELA) nor boundary B (selected by CHCONn.BSELB) is fixed in its assignment as a lower or upper one. The boundary's value specifies whether it is assumed to be the upper or lower one.

In this example, channel number 5 is configured for limit checking. CHCON5.BSELA is set to 10_B and selects the boundary stored in LCCON2.BOUNDARY. CHCON5.BSELB is configured to 01_B and selects the boundary stored in LCCON1.BOUNDARY. Since the value of LCCON1.Boundary is above than the value of LCCON2.BOUNDARY, it is assumed to be the upper one while the boundary stored in LCCON2.BOUNDARY is the lower one.

23.1.8 Expansion of Analog Channels

The number of analog inputs can be expanded in a very flexible and powerful way to satisfy the increased needs for analog inputs. In addition to the internal analog input channel group selection, external analog multiplexers can be connected to each analog channel if the following items are considered:

- Inverse current injection (overload) behavior
- ON resistance of the external multiplexer and load capacitance
- Timing of the external multiplexer
- Noise due to adjacent digital input pins

Note: The characteristics of the external multiplexers influence the accuracy of the A/D Converters. An accuracy of ±2 LSB @ 10-bit resolution is no longer guaranteed.

Two control lines are provided to drive external multiplexers, as shown in **Figure 23-21**.

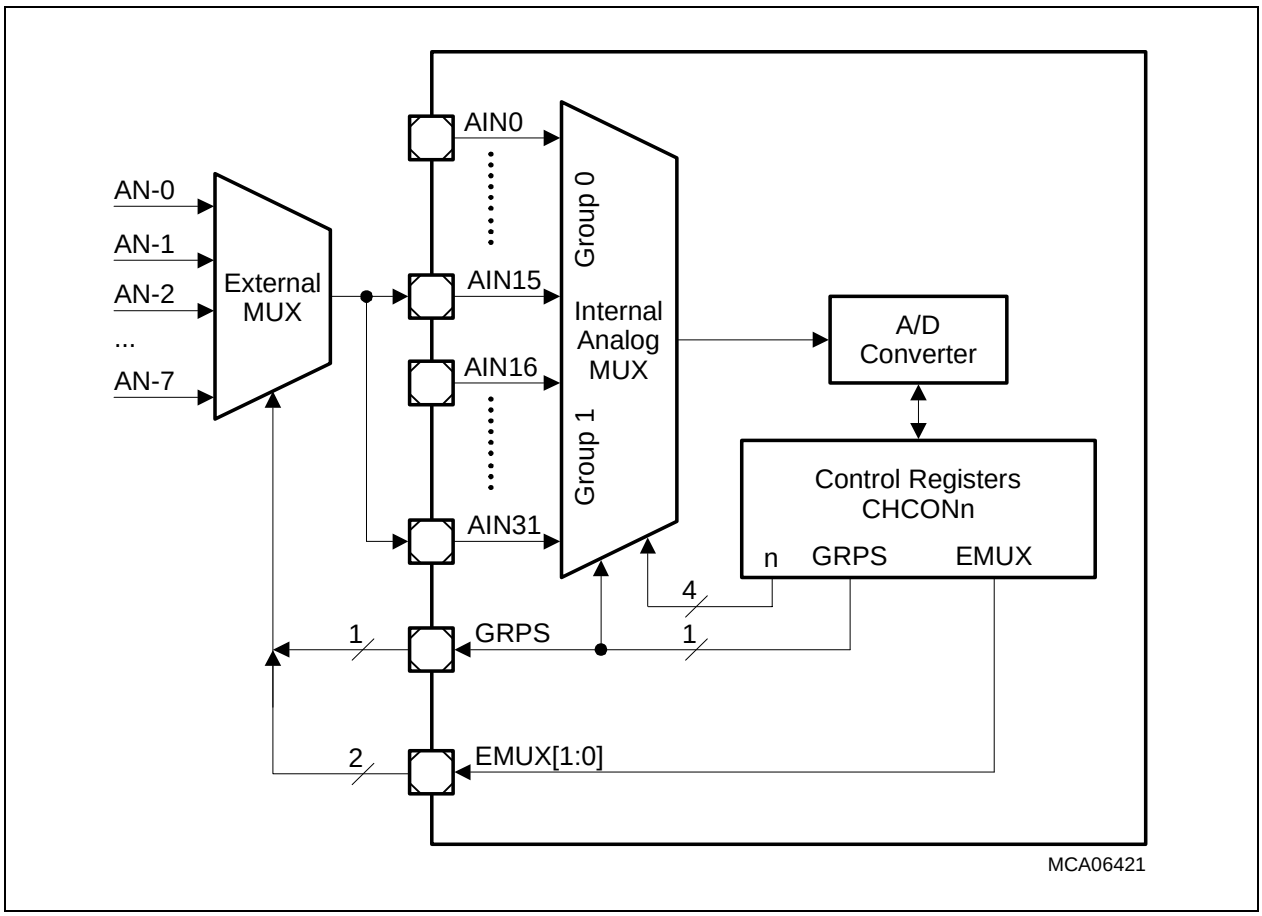


Figure 23-21 External Expansion of Analog Channels

External analog multiplexers receive the select information CHCONn.EMUX[1:0] (and optional GRPS, too) from the channel-specific control register individually for each analog channel. Sequential sources derive the external multiplexer control information from the conversion request control register (bit field CHIN.EMUX and QUEUE0.EMUX).

The external multiplexer controls (bit field CHCONn.EMUX) from the channel-specific control registers are not taken into account for sequential sources.

23.1.8.1 Inverse Current Injection (Overload) Behavior

An overload condition occurs when the analog input voltage is above or below the supply range. An overload condition at a channel connected to an external multiplexer (such as AN-0 in **Figure 23-21**) can affect the conversion of another channel connected to the same external multiplexer (such as AN-1 to AN-3), depending on the overload capability of the external multiplexer. In case of an overload condition at one channel while another channel of the same external multiplexer is sampled by the A/D Converter, an even higher conversion error must be expected.

Note: The overload behavior of every channel that is directly connected to the internal multiplexer or through another external multiplexer does not change.

23.1.8.2 On Resistance of the External Multiplexer

If an external multiplexer is connected to an analog input channel, a typical application might add an RC filter before the external multiplexer to each additional external analog inputs. For example, each of the external analog inputs AN-0 to AN-3 in **Figure 23-21** is adapted by an RC filter. In this case, the resistance of the external multiplexer reduces the efficiency of the external capacitors of the RC filter. An additional blocking capacitor between the external multiplexer and the analog input line could improve the noise suppression capability. However, in this case, the capacitance that must be charged, would be increased by the size of the blocking capacitor.

23.1.8.3 Timing of the External Multiplexer

An analog input channel of an external analog multiplexer is selected after the arbitration round is finished. Therefore, the information to drive an external multiplexer is available before the sample time begins.

23.1.8.4 Load Capacitance

Because each analog input of the external multiplexer might be applied by different analog voltages, the total input capacitance of the A/D Converter must be recharged within the sample time, each time an analog input channel of an external multiplexer is measured.

For analog input channels that are directly applied to the analog input pin of the A/D Converter, the input capacitance does not change. The analog voltage source of such channels must solely recharge the switched input capacitance of the A/D Converter.

23.1.9 Service Request Processing

A fully configurable and very flexible service request control structure is implemented in the A/D Converter Module (see **Figure 23-22**). The A/D Converter module provides a total of sixteen channels and four parallel/serial service request sources. Each service request source can be programmed to generate one out of eight service request output signals SR[7:0]. The service request compressor logic allows more than one service request source to be assigned to one service request output.

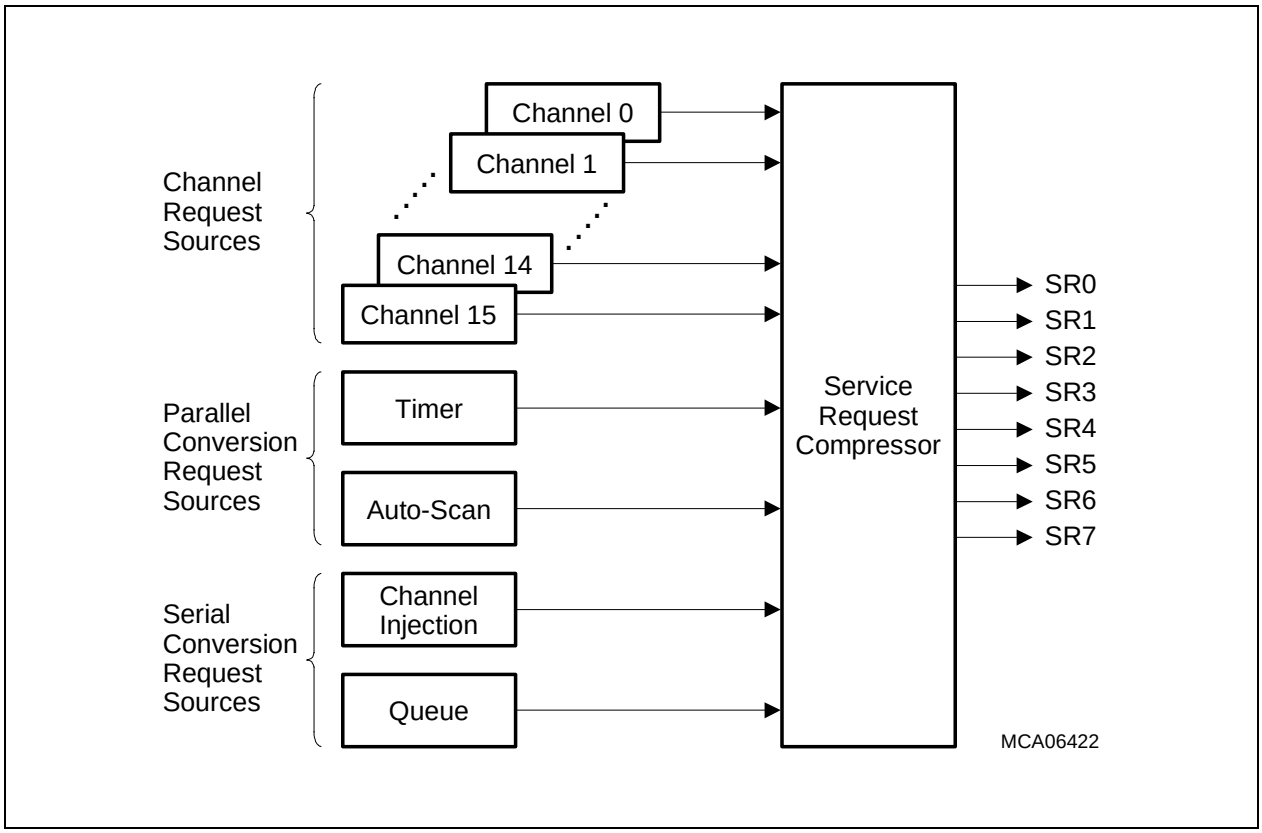


Figure 23-22 Service Request Configuration

Depending on the implementation of the A/D Converter Module in a specific microcontroller, the service request output signals SR[7:0] can be connected either to an interrupt node (service request control register) or can be used as DMA request input of a DMA controller unit. The TC1766 specific service request output configuration is described in **Figure 23-27** and on **Page 23-102**.

The control logic for each of the service request sources provides the following functionality:

- Service request source selection (channel request sources only)
- Service request flag, that can be set/clear by software, too
- Service request enable bit
- Service request node pointer

Table 23-9 lists the service request sources of the A/D Converter module with the related control and status flags/bits.

Table 23-9 Service Request Control/Status Bits/Flags

Service Request Source	Source Selection	Status Flag	Enable Bit	Service Request Node Pointer
Channel n[1)]	CHCONn.LCC [1)]	MSS0.MSRCHn [1)]	CHCONn.ENCH [1)]	CHCONn.INP [1)]
Timer	–	MSS1.MSRT	SRNP.ENPT	SRNP.PT
Queue		MSS1.MSRQR	SRNP.ENPQR	SRNP.PQR
Auto-scan		MSS1.MSRAS	SRNP.ENPAS	SRNP.PAS

1) n = 0-15

23.1.9.1 Channel Request Source Control

Figure 23-23 shows the request control logic of each conversion channel n. Bit field CHCONn.LCC determines which of the different request conditions is selected as channel request. These conditions are:

- Conversion result value is inside/outside a specified range (limit checking enabled)
- CHSTATn.RESULT is updated (limit checking disabled)
- No action (no channel request generated)

When a selected request condition becomes active, the channel n request status flag MSS0.MSRCHn is set and the channel n service request event is reported to the service request compressor logic (if enabled by CHCONn.ENCH). The service request flag MSS0.MSRCHn can only be cleared by software. For test purposes (CON.SRTEST = 1), the service request flag can be additionally set by hardware. Additional details on MSS0.MSRCHn software control is described on **Page 23-48**.

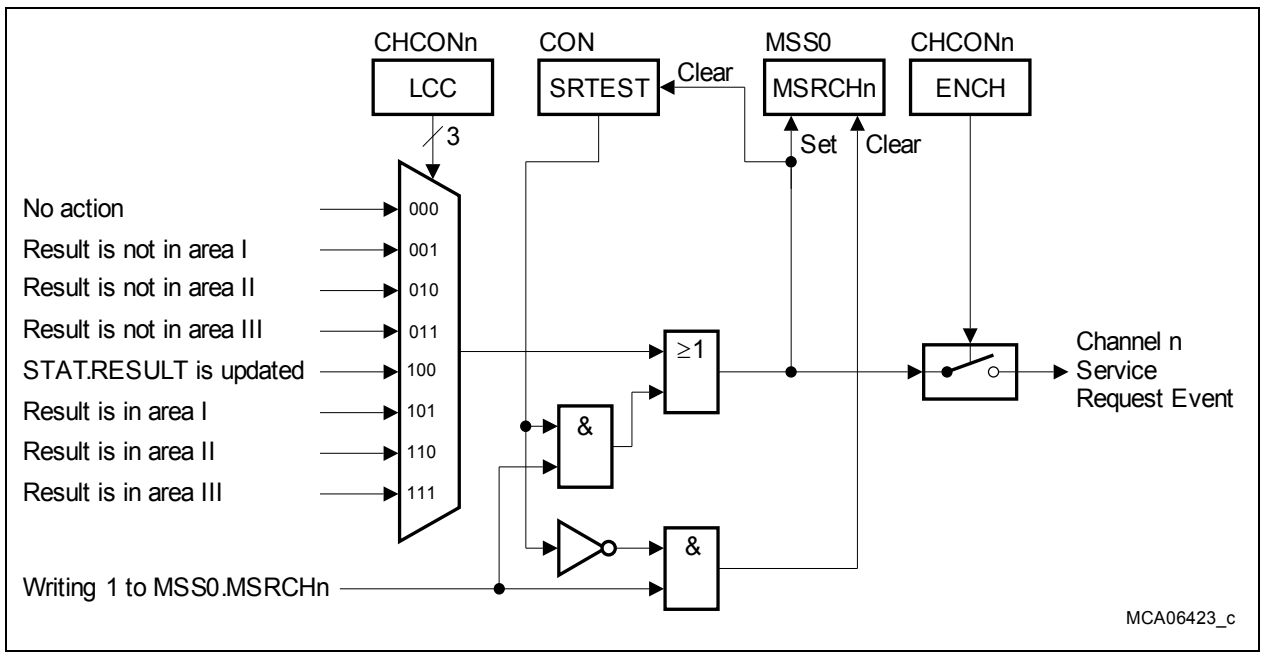


Figure 23-23 Channel n Request Source Logic

23.1.9.2 Parallel/Serial Request Source Control

The parallel/serial request control logic structure as shown in **Figure 23-24** is almost identical with the one shown in **Figure 23-23** except that the request input is controlled by one parallel/serial service request event. As defined in **Table 23-9**, each of the four serial/parallel request source (Timer, Queue, Auto-Scan) has its own request status flag and request enable bit. The service request flags that are located in register MSS1 can also be set or cleared by software.

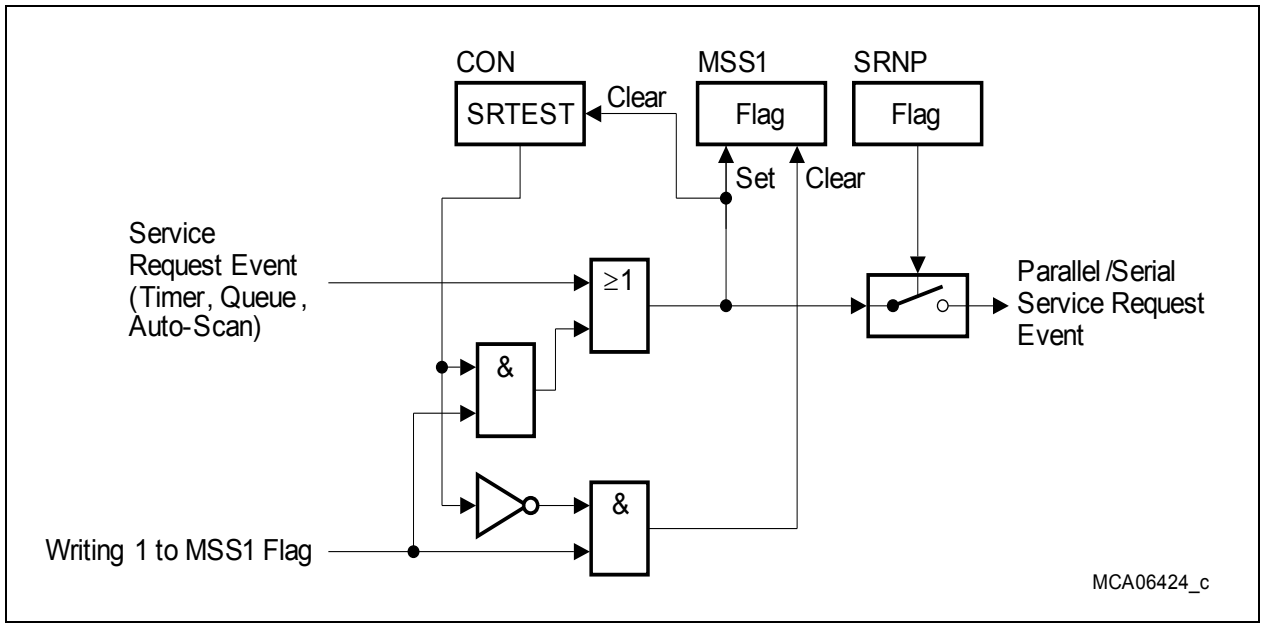


Figure 23-24 Parallel/Serial Request Source Logic

23.1.9.3 Service Request Compressor

As shown in **Figure 23-22**, the A/D Converter module is equipped with 20 service request sources and eight service request output lines SR[7:0]. In the service request compressor each of the 20 service request sources can be assigned to one of the eight service request output lines. One service request event can only be assigned to one service request output but one service request output can be used by multiple service request events.

Figure 23-25 illustrates the request compressor logic for two service request events. Each request event is controlled by one node pointer. This node pointer assigns a request event to one of the service request outputs. Channel n request events are controlled by a node pointer that is located in each channel n control register (CHCONn.INP). The node pointers for the four parallel/serial request events are located in register SRPN (see **Table 23-9**). The open inputs of the OR gates are connected to the remaining 18 service request event demultiplexers.

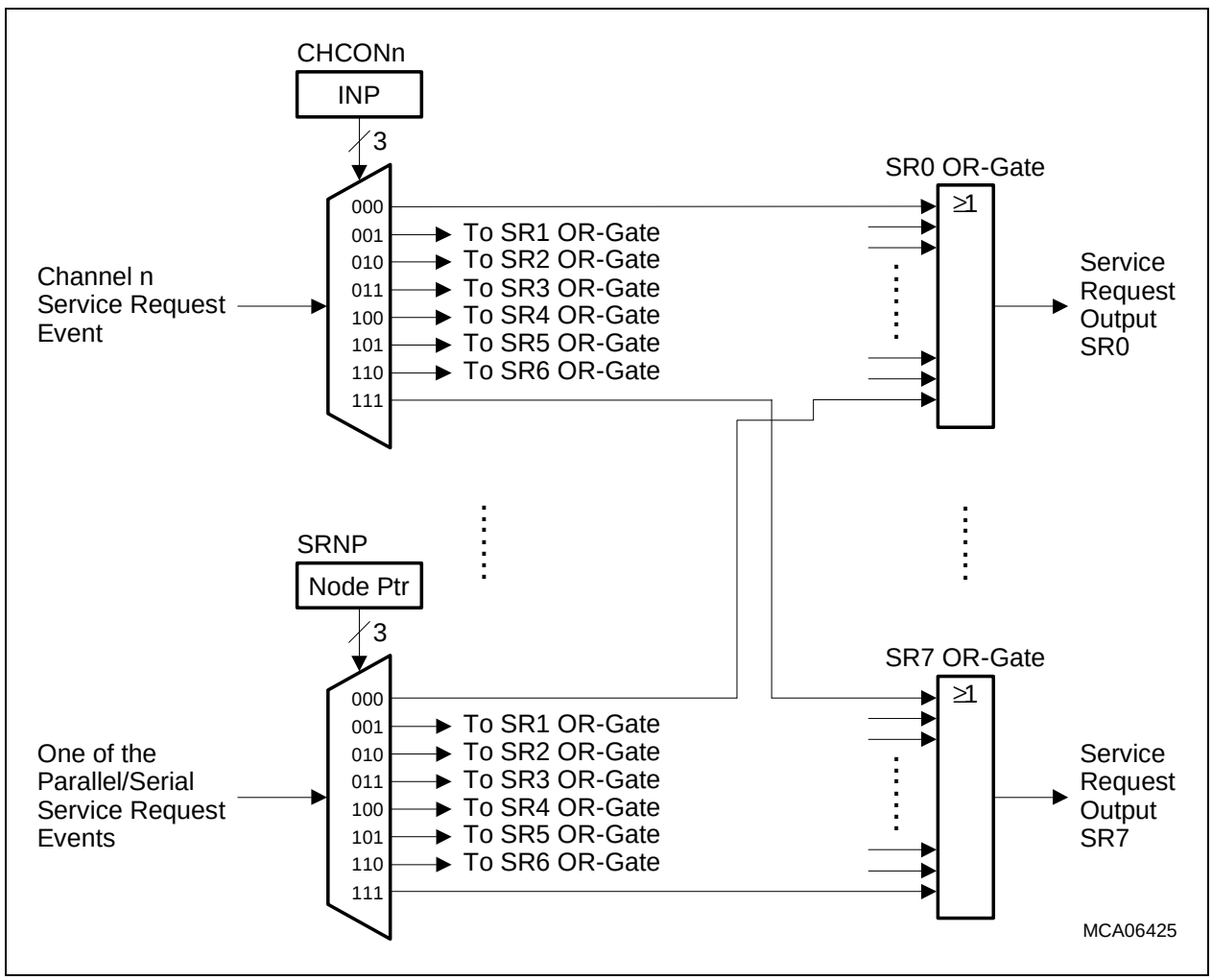


Figure 23-25 Service Request Compressor Logic

23.1.9.4 Service Request Flag Control

As shown in **Figure 23-23** and **Figure 23-24**, the service request status flags are always set by hardware but can be cleared only via software (or a module reset operation). A service request status flag is cleared under software control by writing a 1 to the corresponding bit position in the MSS0/MSS1 register.

When bit CON.SRTEST is set, service request flags can also be set by software. A service request flag is set when CON.SRTEST = 1 is set and a 1 is written to the corresponding bit position in the MSS0/MSS1 register. After this MSS0/MSS1 write action is executed, bit CON.SRTEST becomes automatically cleared again.

Table 23-10 summarizes the actions that are performed when a write action on a MSS0/MSS1 register service request status flag occurs.

Table 23-10 Service Request Status Flag Set/Clear Operations

SCON.SRTEST	MSS Flag Current Value	Value Written to MSS Flag	MSS Flag New Value After Write	Comment
0	0	0	0	No action
		1		No action
	1	0	1	No action
		1	0	MSS flag is cleared by software
1	0	0	0	No action
		1	1	MSS flag is set by software and a service request is generated
	1	0	1	No action
		1	1	MSS flag is set by software and a service request is generated

23.2 ADC Kernel Registers

This section describes the kernel registers of the ADC module. All ADC kernel register names described in this section will be referenced in other parts of the TC1766 User's Manual by the module name prefix "ADC0_" for the ADC0 interface.

ADC Kernel Register Overview

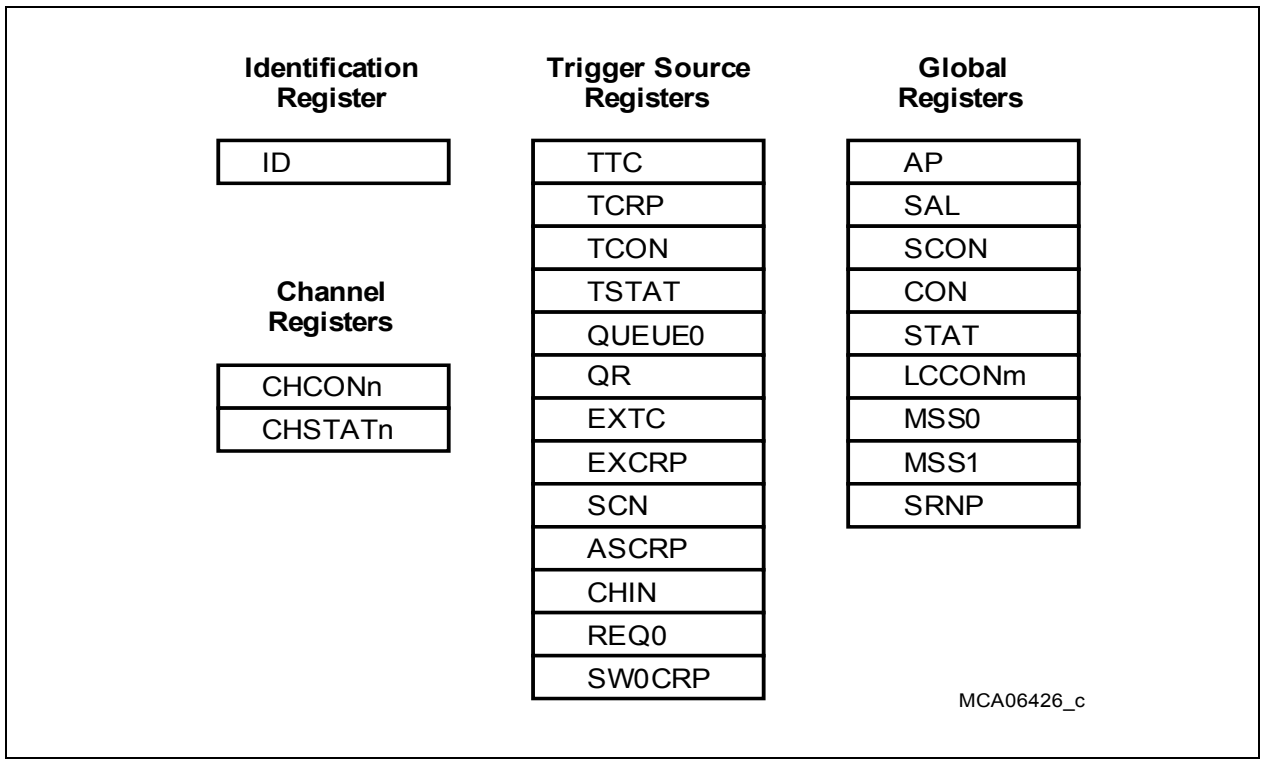


Figure 23-26 ADC Kernel Registers

The complete and detailed address map of the ADC0 module is described in Table 16-21 on Page 16-55 of the TC1766 User's Manual System Units part (Volume 1).

Attention: For documentation automation purposes, the CHCON and CHSTAT registers use index "m" while LCCON registers use index "x" in the ADC module kernel register description. These indexes are represented by "n" for CHCON and CHSTAT registers and "m" for LCCON register in other sections of the chapter.

Table 23-11 Registers Address Space - ADC Kernel Registers

Module	Base Address	End Address	Note
ADC0	$F010\ 0400_H$	$F010\ 05FF_H$	-

Table 23-12 Registers Overview - ADC Kernel Registers

Register Short Name	Register Long Name	Offset Address[1)]	Description see
ID	Module Identification Register	0008_H	Page 23-52
CHCONm	Channel Control Register m(m = 0-15)	$0010_H + m \times 4_H$	Page 23-53
AP	Arbitration Participation Register	0084_H	Page 23-69
SAL	Source Arbitration Level Register	0088_H	Page 23-70
TTC	Timer Trigger Control Register	$008C_H$	Page 23-57
EXTC	External Trigger Control Register	0090_H	Page 23-64
SCON	Source Control Register	0098_H	Page 23-72
LCCONx	Limit Check Control Register x (x = 0-3)	$0100_H + x \times 4_H$	Page 23-71
TCON	Timer Control Register	0114_H	Page 23-58
CHIN	Channel Injection Control Register	0118_H	Page 23-77
QR	Queue Register	$011C_H$	Page 23-62
CON	Converter Control Register	0120_H	Page 23-73
SCN	Auto Scan Control Register	0124_H	Page 23-66
REQ0	Conversion Request Register SW0	0128_H	Page 23-79
CHSTATm	Channel Status Register m(m = 0-15)	$0130_H + m \times 4_H$	Page 23-55
QUEUE0	Queue Status Register	0170_H	Page 23-61
SW0CRP	Software SW0 Conv. Req. Pending Reg.	0180_H	Page 23-80

Table 23-12 Registers Overview - ADC Kernel Registers (cont'd)

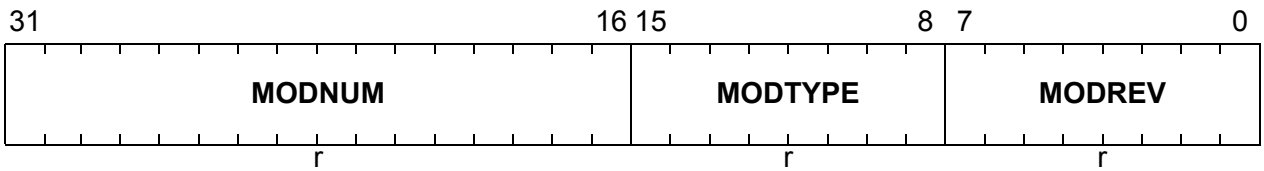
Register Short Name	Register Long Name	Offset Address[1)]	Description see
ASCRP	Auto Scan Conversion Req. Pending Register	0188_H	Page 23-67
TSTAT	Timer Status Register	$01B0_H$	Page 23-59
STAT	Converter Status Register	$01B4_H$	Page 23-75
TCRP	Timer Conversion Req. Pending Register	$01B8_H$	Page 23-60
EXCRP	External Conversion Req. Pending Register	$01BC_H$	Page 23-65
MSS0	Module Service Request Status Register 0	$01D0_H$	Page 23-81
MSS1	Module Service Request Status Register 1	$01D4_H$	Page 23-82
SRNP	Service Request Node Pointer Register	$01DC_H$	Page 23-83

1) The absolute register address is calculated as follows:
Module Base Address (Table 23-11) + Offset Address (shown in this column)

23.2.1 ADC Module Identification Register

ADC0_ID
Module Identification Register

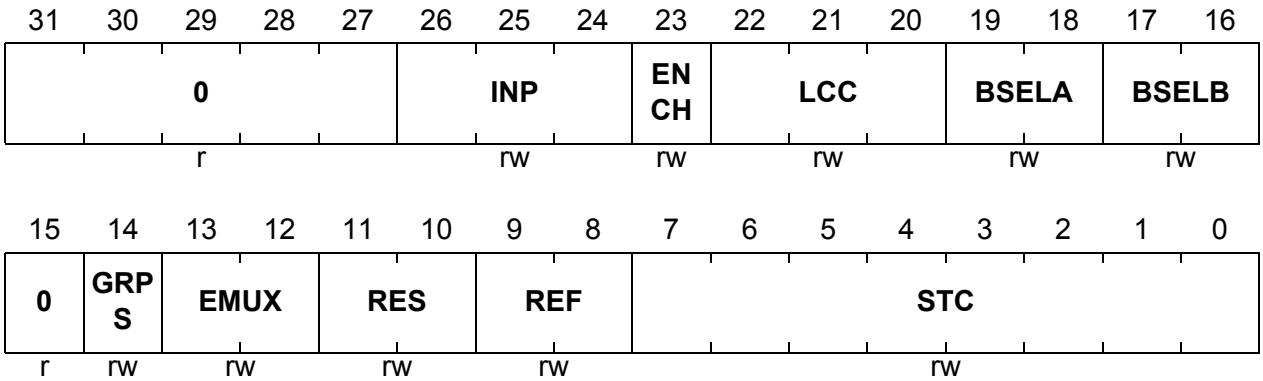
(008_H) **Reset Value: 0030 C0XX$_H$**



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 ADC: 0030_H

23.2.2 Channel Registers

CHCONm (m = 0-15)
Channel Control Register m **(010_H+m*4)** **Reset Value: 0000 0000_H**

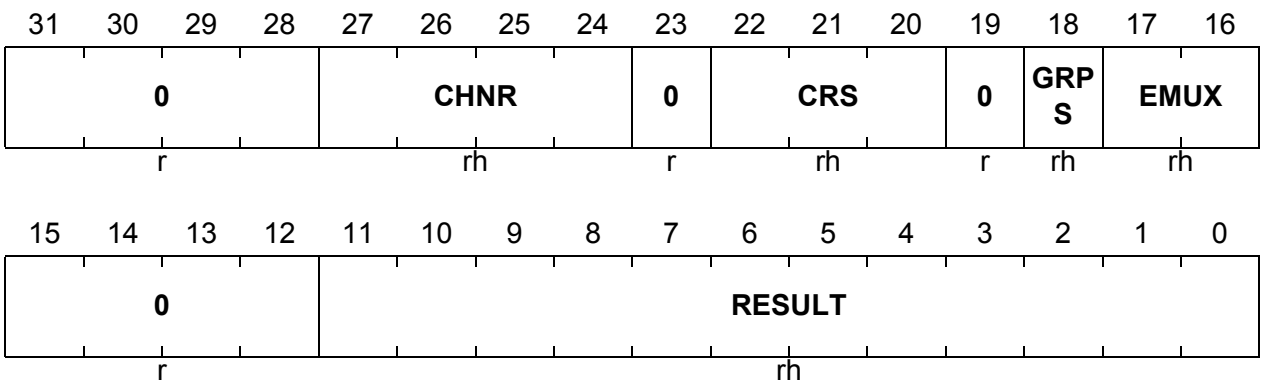


Field	Bits	Type	Description
STC	[7:0]	rw	**Sample Time Control** STC determines the duration of the sample phase for channel n. Any modification of this bit field is taken into account after the currently running conversion is finished.
REF	[9:8]	rw	**Analog Reference Voltage Control** This bit determines the reference voltage for channel n. 00_B V_{AREF}[0] is selected as reference voltage 01_B V_{AREF}[1] is selected as reference voltage 10_B V_{AREF}[2] is selected as reference voltage 11_B Reserved See **Page 23-95** for TC1766 specific implementation.
RES	[11:10]	rw	**Conversion Resolution Control** RES determines the resolution of the A/D Converter for the conversion of channel n. Any modification of this bit field is taken into account after the currently running conversion is finished. 00_B 10-bit resolution selected 01_B 12-bit resolution selected 10_B 8-bit resolution selected 11_B Reserved
EMUX	[13:12]	rw	**External Multiplexer Control** EMUX drives the external multiplexer control lines. EMUX[0] is assigned to EMUX0 output. EMUX[1] is assigned to EMUX1 output.

Field	Bits	Type	Description
GRPS	14	rw	**Analog Input Multiplexer Group Select** This bit selects the analog input multiplexer group for analog input channel n. 0_B Analog input multiplexer group 0 selected. Analog inputs AIN[15:0] are assigned to analog input channels [15:0]. 1_B Analog input multiplexer group 1 selected. Analog inputs AIN[31:16] are assigned to analog input channels [15:0].
BSELA, BSELB	[17:16], [19:18]	rw	**Boundary Select Control** This bit selects two limit check control registers for limit checking. 00_B LCCON0 (BOUNDARY0) is selected 01_B LCCON1 (BOUNDARY1) is selected 10_B LCCON2 (BOUNDARY2) is selected 11_B LCCON3 (BOUNDARY3) is selected
LCC	[22:20]	rw	**Limit Check Control** 000_B Neither limit check is performed nor a service request is generated on write of the conversion result to bit field STAT.RESULT. 001_B Generate a service request if conversion result is in not area I. 010_B Generate a service request if conversion result is not in area II. 011_B Generate a service request if conversion result is not in area III. 100_B Generate a service request on write of conversion result to bit field STAT.RESULT. 101_B Generate a service request result if conversion result is in area I. 110_B Generate a service request result if conversion result is in area II. 111_B Generate a service request result if conversion result is in area III.
ENCH	23	rw	**Enable Channel Service Request Generation** This bit enables the generation of a channel service request at the end of a channel conversion. 0_B The channel service request generation is disabled. 1_B The channel service request generation is enabled.

Field	Bits	Type	Description
INP	[26:24]	rw	**Interrupt Node Pointer** This bit field selects which service request output line will be activated if the channel conversion is finished and the channel service request generation is enabled. 000_B Service request line SR0 selected 001_B Service request line SR1 selected 010_B Service request line SR2 selected 011_B Service request line SR3 selected 100_B Service request line SR4 selected 101_B Service request line SR5 selected 110_B Service request line SR6 selected 111_B Service request line SR7 selected
0	15, [31:27]	r	**Reserved** Read as 0; should be written with 0.

CHSTATm (m = 0-15)
Channel Status Register m **(130_H+m*4)** **Reset Value: 0000 0000_H**



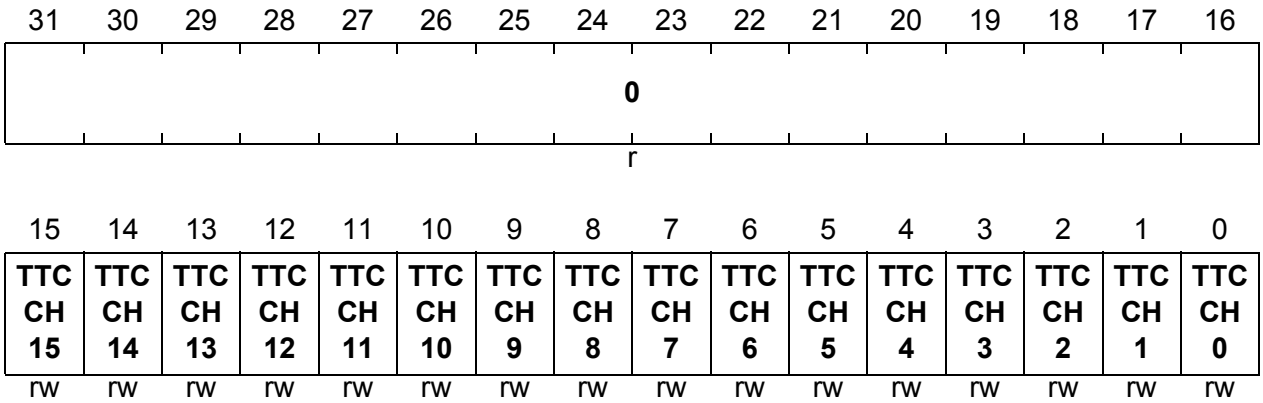
Field	Bits	Type	Description
RESULT	[11:0]	rh	**Result of the Last Conversion** This bit field contains the left aligned conversion result of the last conversion of channel n. 8-bit: Conversion result is stored in RESULT[11:4] 10-bit: Conversion result is stored in RESULT[11:2] 12-bit: Conversion result is stored in RESULT[11:0] Unused bits of RESULT are 0.

Field	Bits	Type	Description
EMUX	[17:16]	rh	**Status of External Multiplexer** This bit field indicates the setting of the external multiplexer control of DMA channel n. This information is either derived from CHCONn.EMUX (parallel conversion request sources) or from CHIN.EMUX and QUEUE.EMUX (sequential conversion request sources).
GRPS	18	rh	**Status of Analog Input Multiplexer Group Select** This bit indicates the status of the analog input multiplexer group select bit for analog input channel n.
CRS	[22:20]	rh	**Conversion Request Source** This bit field indicates for channel n the origin of the conversion result, stored in bit field RESULT. 000_B Channel injection 001_B Timer 010_B Reserved 011_B External event 100_B Software SW0 101_B Reserved 110_B Queue 111_B Auto-Scan
CHNR	[27:24]	rh	**Channel Number** This bit field holds the channel number n.
0	[15:12], 19,23, [31:28]	r	**Reserved** Read as 0.

Note: The former contents of a CHSTATn register are overwritten with the new result for the same channel.

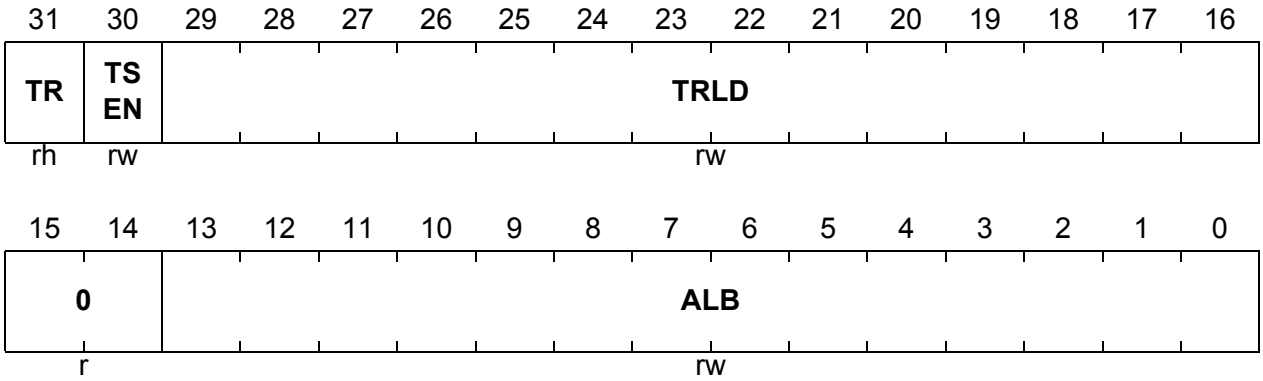
23.2.3 Timer Registers

TTC
Time Trigger Control Register (08C$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
TTCCHn (n = 0-15)	n	rw	**Timer Trigger Control for Channel n** TTCCHn determines whether a conversion request is triggered for channel n on timer underflow or not. 0_B No conversion request is triggered for channel n. 1_B A conversion request is triggered for channel n.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

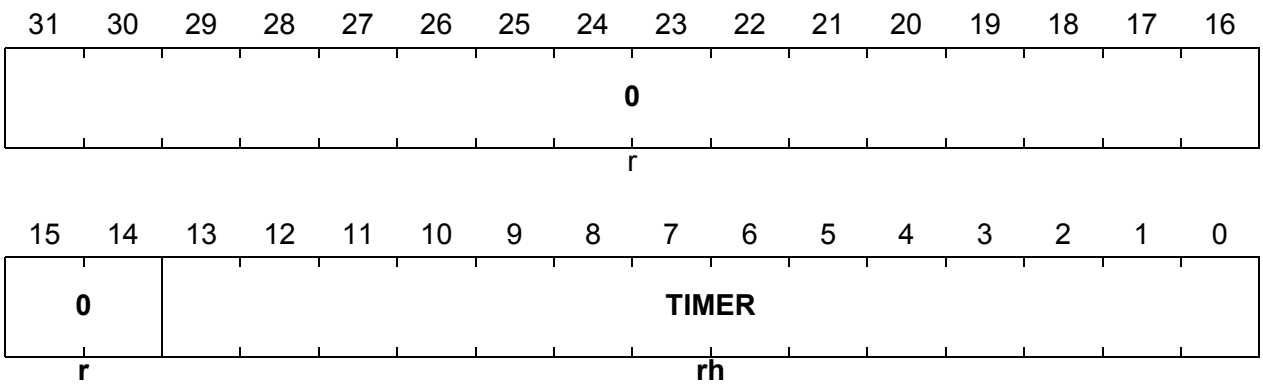
TCON
Timer Control Register **(114_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
ALB	[13:0]	rw	**Arbitration Lock Boundary** The arbitration lock boundary is used to specify the arbitration lock time t_{LOCK}. Arbitration Lock Mode is automatically enabled if any value greater than zero is written to ALB. *Note: The arbitration is locked if the value of ALB is above TRLD.*
TRLD	[29:16]	rw	**Timer Reload Value** TRLD is loaded into the timer register TSTAT.TIMER when TSTAT.TIMER = 0 or each time when SCON.TRS is set. *Note: If TRLD is zero, the timer should not be enabled, timer lock is always active and a service request can be generated for each timer clock.*
TSEN	30	rw	**Timer Stop Enable** 0_B TSTAT.TIMER = 0 has no effect on the timer run bit TCON.TR. 1_B Timer run bit TCON.TR is cleared when TSTAT.TIMER = 0.

Field	Bits	Type	Description
TR	31	rh	**Timer Run Control** 0_B Timer TSTAT.TIMER is stopped. 1_B Timer TSTAT.TIMER is running and is decremented with clock f_{TIMER}. *Note: Clearing it TR causes the arbitration lock to be removed, if it is set.*
0	[15:14]	r	**Reserved** Read as 0; should be written with 0.

TSTAT
Timer Status Register **(1B0$_H$)** **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
TIMER	[13:0]	rh	**Timer Register** This bit field contains the current value of the timer.
0	[31:14]	r	**Reserved** Read as 0.

TCRP
Timer Conversion Request Pending Register
($1B8_H$)

Reset Value: $0000\ 0000_H$

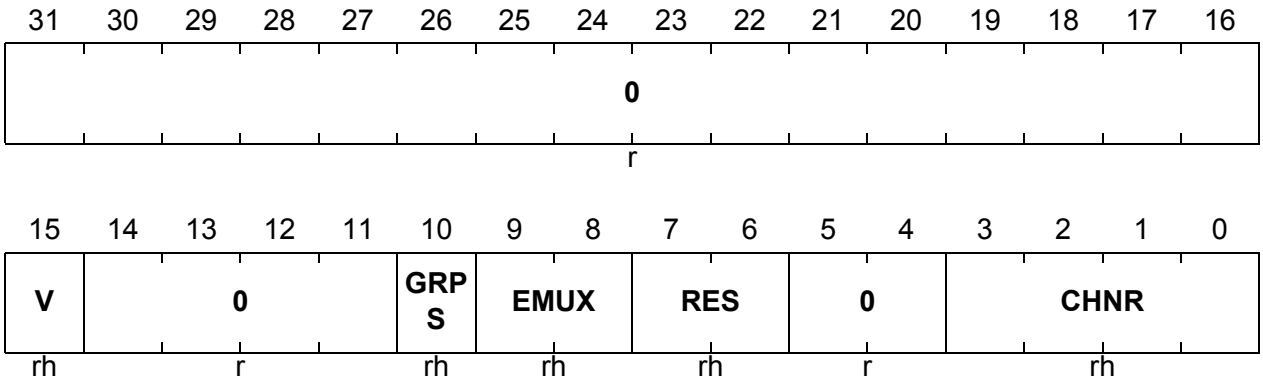
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRP 15	**TRP 14**	**TRP 13**	**TRP 12**	**TRP 11**	**TRP 10**	**TRP 9**	**TRP 8**	**TRP 7**	**TRP 6**	**TRP 5**	**TRP 4**	**TRP 3**	**TRP 2**	**TRP 1**	**TRP 0**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
TRPn (n = 0-15)	n	rh	**Timer Conversion Request Pending Flag for Channel n** A pending flag TRPn is set each time a conversion request is generated for channel n on timer underflow that could not be serviced immediately. A start of conversion of the pending request leads automatically to a cleared of the pending flag. All pending request flags can also be cleared under software control, if bit AP.TP is cleared. 0_B No timer based conversion request is pending for channel n 1_B A timer based conversion request is pending for channel n
0	[31:16]	r	**Reserved** Read as 0.

23.2.4 Queue Registers

QUEUE0
Queue Status Register **(170_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
CHNR	[3:0]	rh	**Channel to be Converted** This bit field holds the channel number of the channel to be converted.
RES	[7:6]	rh	**Conversion Resolution Status** This bit field indicates the resolution of the A/D Converter for the conversion of the analog channel determined by CHNR. Any modification of this bit field is taken into account after the currently running conversion is finished. 00_B 10-bit resolution 01_B 12-bit resolution 10_B 8-bit resolution 11_B Reserved
EMUX	[9:8]	rh	**External Multiplexer Control Line Status** This bit field indicates the external multiplexer control line status of the analog channel determined by CHNR.
GRPS	10	rh	**Analog Input Multiplexer Group Select Line Status** This bit field indicates the status of the analog input multiplexer group select bit of the analog channel determined by CHNR. 0_B Analog input multiplexer group 0 selected. 1_B Analog input multiplexer group 1 selected.

Field	Bits	Type	Description
V	15	rh	**Valid Status** This bit field indicates whether the information of register QR is valid or invalid. 0_B CHNR, RES, EMUX and GRPS are invalid 1_B CHNR, RES, EMUX and GRPS are valid; a queue conversion request is pending.
0	[5:4], [14:11], [31:16]	r	**Reserved** Read as 0.

QR
Queue Register **($11C_H$)** **Reset Value: $0000\ 0000_H$**

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

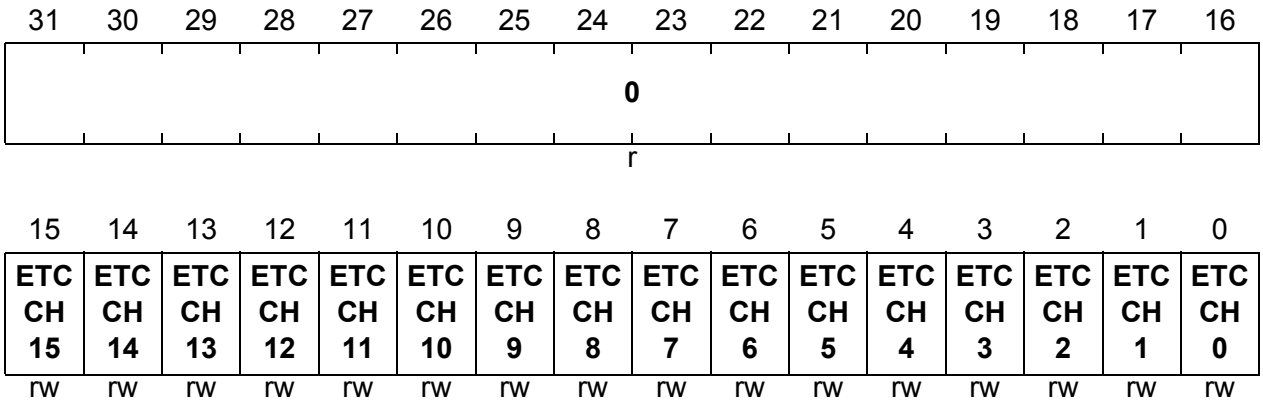
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
V	**0**				**GRPS**	**EMUX**		**RES**		**0**		**CHNR**			
rwh	r				rw	rw		rw		r		rw			

Field	Bits	Type	Description
CHNR	[3:0]	rw	**Channel to be Converted**
RES	[7:6]	rw	**Conversion Resolution Control** This bit field selects the resolution of the A/D Converter for the conversion of the analog channel as programmed for CHNR. Any modification of this bit field is taken into account after the currently running conversion is finished. 00_B 10-bit resolution 01_B 12-bit resolution 10_B 8-bit resolution 11_B Reserved

Field	Bits	Type	Description
EMUX	[9:8]	rw	**External Multiplexer Control** This bit field determines the settings of the external multiplexer control lines for the conversion of the analog channel as programmed for CHNR.
GRPS	10	rw	**Analog Input Multiplexer Group Select** This bit selects the analog input multiplexer group for the conversion of the analog channel as programmed for CHNR. 0_B Analog input multiplexer group 0 selected 1_B Analog input multiplexer group 1 selected
V	15	rwh	**Valid Control** This bit indicates whether the information of register QR is valid or invalid. Bit V is cleared by hardware when the QR contents are transferred to the queue. 0_B CHNR, RES, EMUX and GRPS are invalid 1_B CHNR, RES, EMUX and GRPS are valid
0	[5:4], [14:11], [31:16]	r	**Reserved** Read as 0; should be written with 0.

23.2.5 External Trigger Registers

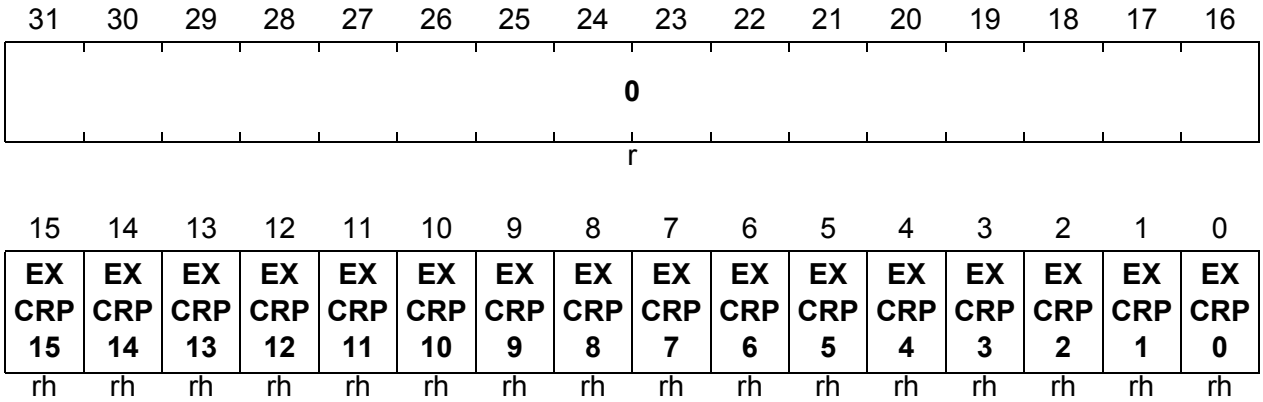
EXTC
External Trigger Control Register (090_H) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
ETCHn (n = 0-15)	n	rw	**External Trigger Control for Channel n** ETCHn specifies if a conversion request is triggered on an event on the selected input line (including gating) for channel n. 0_B No conversion request is triggered for channel n. 1_B A conversion request is triggered for channel n
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

EXCRP
External Conversion Request Pending Register

(1BC$_H$) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
EXCRPn (n = 0-15)	n	rh	**External Event Conversion Request Pending Flag for Channel n** EXCRPn is set each time a conversion request is generated for channel n by an external event that could not be serviced immediately. A start of conversion of the pending request leads automatically to a clear of the pending flag. All pending request flags can also be cleared under software control, if bit AP.EXP is cleared. 0_B No external event based conversion request is pending for channel n 1_B An external event based conversion request is pending for channel n
0	[31:16]	r	**Reserved** Read as 0.

23.2.6 Auto-Scan Registers

SCN
Auto-Scan Control Register (124_H) Reset Value: $0000\ 0000_H$

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SRQ 15	SRQ 14	SRQ 13	SRQ 12	SRQ 11	SRQ 10	SRQ 9	SRQ 8	SRQ 7	SRQ 6	SRQ 5	SRQ 4	SRQ 3	SRQ 2	SRQ 1	SRQ 0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
SRQn (n = 0-15)	n	rw	**Auto-Scan Request for Channel n** SRQn determines whether or not channel n participates in an auto-scan sequence. 0_B Channel n does not participate in an auto-scan sequence. 1_B Channel n participates in an auto-scan sequence. *Note: Bits SRQn maintain their values after auto-scan control bit field CON.SCNM is cleared.*

Field	Bits	Type	Description
GRPC	[31:30]	rw	**Group Control** This bit field determines the behavior of the analog input group selection for the auto-scan sequence. 00_B Group select bit CHCONn.GRPS is taken into account for the conversion of the corresponding channel if the conversion is triggered by auto-scan. Bit ASCRP.GRPS is set to 0. 01_B Bit ASCRP.GRPS is set to 0 and it is taken into account for all conversions triggered by auto-scan (only group 0 is covered by the auto-scan). 10_B Bit ASCRP.GRPS is set to 1 and it is taken into account for all conversions triggered by auto-scan (only group 1 is covered by the auto-scan). 11_B Bit ASCRP.GRPS is toggled at the end of each auto-scan sequence. It is taken into account for all conversions triggered by auto-scan (group 0 is covered by an auto-scan sequence and group 1 by the following sequence, followed by group 0, etc.).
0	[29:16]	r	**Reserved** Read as 0; should be written with 0.

ASCRP
Auto-Scan Conversion Request Pending Register

(188_H) **Reset Value: 0000 0000_H**

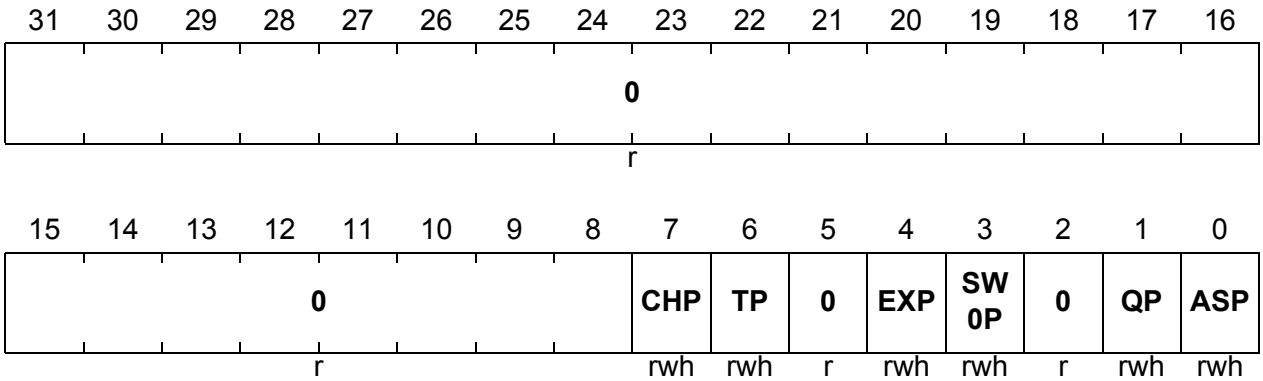
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
GRPS								**0**							
rh								r							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ASCRP15	**ASCRP14**	**ASCRP13**	**ASCRP12**	**ASCRP11**	**ASCRP10**	**ASCRP9**	**ASCRP8**	**ASCRP7**	**ASCRP6**	**ASCRP5**	**ASCRP4**	**ASCRP3**	**ASCRP2**	**ASCRP1**	**ASCRP0**
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
ASCRPn (n = 0-15)	n	rh	**Auto-Scan Conversion Request Pending Flag for Channel n** The pending flag ASCRPn is set each time a conversion request is generated for channel n by auto-scan that could not be serviced immediately. A start of conversion of the pending request leads automatically to a clear of the pending flag. All pending request flags can also be cleared under software control, if bit AP.ASP is cleared. 0_B No auto-scan based conversion request is pending for channel n. 1_B An auto-scan based conversion request is pending for channel n.
GRPS	31	rh	**Group Select for Auto-Scan Conversions** This flag indicates which analog input group is taken for conversions triggered by auto-scan. It is only taken into account if SCN.GRPC > 00_B. 0_B The channels of group 0 are covered by the auto-scan sequence. 1_B The channels of group 1 are covered by the auto-scan sequence.
0	[30:16]	r	**Reserved** Read as 0.

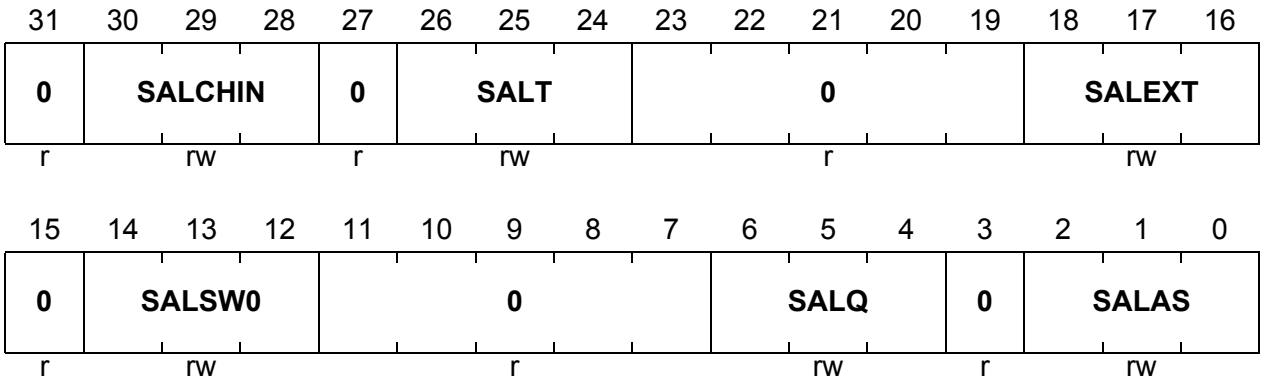
23.2.7 Other Control/Status Registers

AP
Arbitration Participation Register (084_H) Reset Value: $0000\ 0000_H$



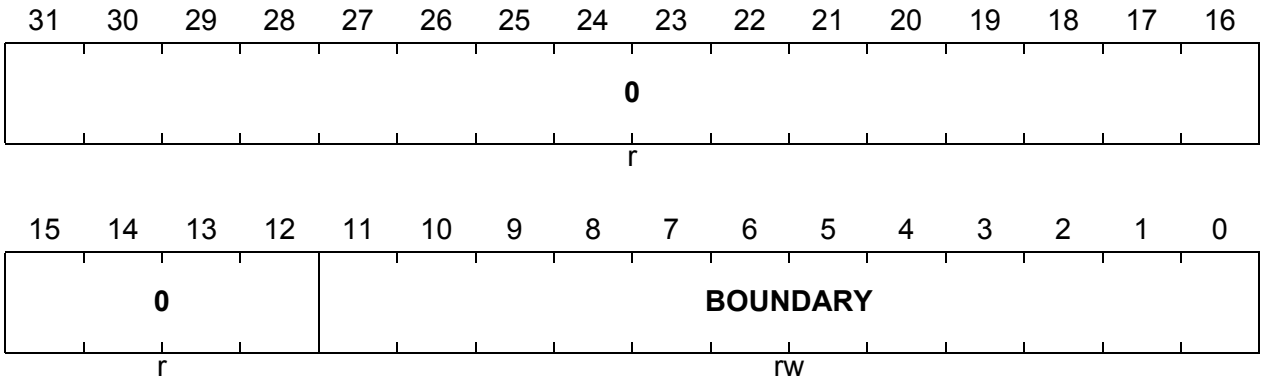
Field	Bits	Type	Description
ASP	0	rwh	**Auto-Scan Arbitration Participation** 0_B Source does not participate in arbitration 1_B Source participates in arbitration
QP	1	rwh	**Queue Arbitration Participation** 0_B Source does not participate in arbitration 1_B Source participates in arbitration
SW0P	3	rwh	**Software SW0 Arbitration Participation Flag** 0_B Source does not participate in arbitration 1_B Source participates in arbitration
EXP	4	rwh	**External Event Arbitration Participation Flag** 0_B Source does not participate in arbitration 1_B Source participates in arbitration
TP	6	rwh	**Timer Arbitration Participation Flag** 0_B Source does not participate in arbitration 1_B Source participates in arbitration
CHP	7	rwh	**Channel Injection Arbitration Participation Flag** 0_B Source does not participate in arbitration 1_B Source participates in arbitration
0	2,5, [31:8]	r	**Reserved** Read as 0; should be written with 0.

SAL
Source Arbitration Level Register (088_H) Reset Value: 0103 4067$_H$



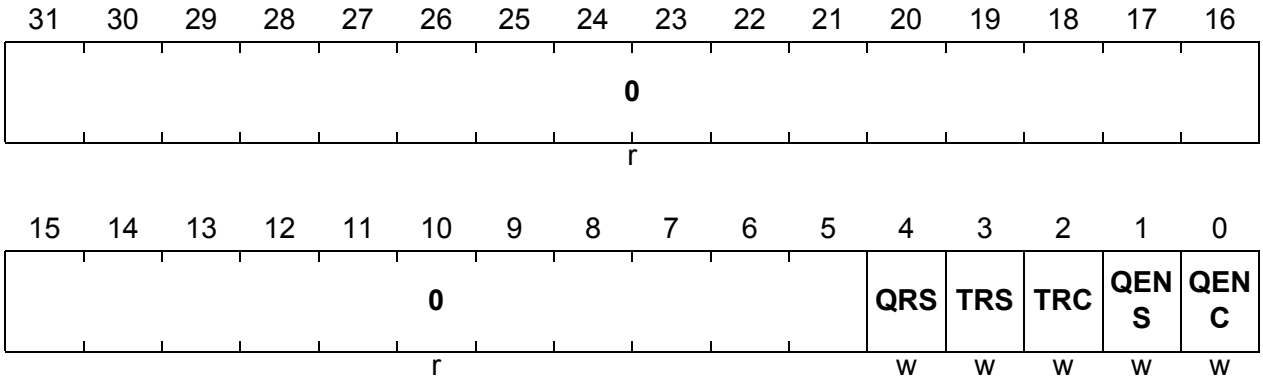
Field	Bits	Type	Description
SALAS	[2:0]	rw	**Auto-Scan Source Arbitration Level** 000_B Highest priority for arbitration selected 111_B Lowest priority for arbitration selected
SALQ	[6:4]	rw	**Queue Source Arbitration Level** 000_B Highest priority for arbitration selected 111_B Lowest priority for arbitration selected
SALSW0	[14:12]	rw	**Software Source Arbitration Level** 000_B Highest priority for arbitration selected 111_B Lowest priority for arbitration selected
SALEXT	[18:16]	rw	**External Event Source Arbitration Level** 000_B Highest priority for arbitration selected 111_B Lowest priority for arbitration selected
SALT	[26:24]	rw	**Timer Source Arbitration Level** 000_B Highest priority for arbitration selected 111_B Lowest priority for arbitration selected
SALCHIN	[30:28]	rw	**Channel Injection Source Arbitration Level** 000_B Highest priority for arbitration selected 111_B Lowest priority for arbitration selected
0	3, [11:7], 15, [23:19], 27, 31	r	**Reserved** Read as 0; should be written with 0.

LCCONx (x = 0-3)
Limit Check Control Register x (100$_H$+x*4) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
BOUNDARY	[11:0]	rw	**Boundary for Limit Checks** This bit field contains the boundary value used for limit checking. The relevant bits of this bit field for the different resolutions are: 8-bit: BOUNDARY[11:4] 10-bit: BOUNDARY[11:2] 12-bit: BOUNDARY[11:0] Unused bits should be written with 0.
0	[31:12]	r	**Reserved** Read as 0; should be written with 0.

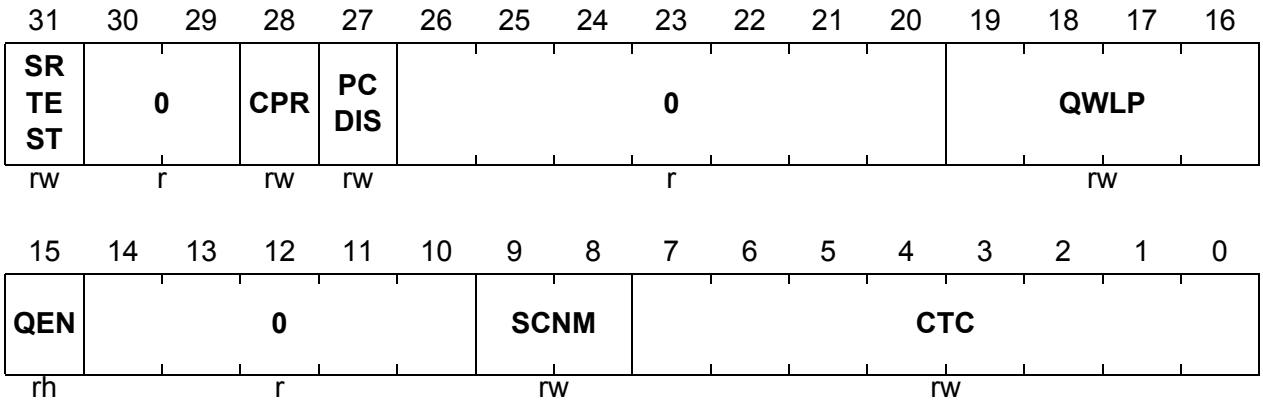
SCON
Source Control Register **(098_H)** **Reset Value: $0000\ 0000_H$**



Field	Bits	Type	Description
QENC	0	w	**Queue Enable Clear** Writing a 1 to this bit clears bit CON.QEN (also if QENS has been set simultaneously).
QENS	1	w	**Queue Enable Set** Writing a 1 to this bit and a 0 to QENC sets bit CON.QEN.
TRC	2	w	**Timer Run Bit Clear** Writing a 1 to this bit, clears bit TCON.TR (also if TRS has been set simultaneously).
TRS	3	w	**Timer Run Bit Set** Writing a 1 to this bit and a 0 to TRC sets bit TCON.TR.
QRS	4	w	**Queue Reset** Setting bit QRS tags all queue elements invalid (clears V bit of each queue element) and clears bits STAT.QF and STAT.QLP.
0	[31:5]	r	**Reserved** Read as 0; should be written with 0.

Note: All SCON bits are write-only bits. Reading SCON always delivers $0000\ 0000_H$.

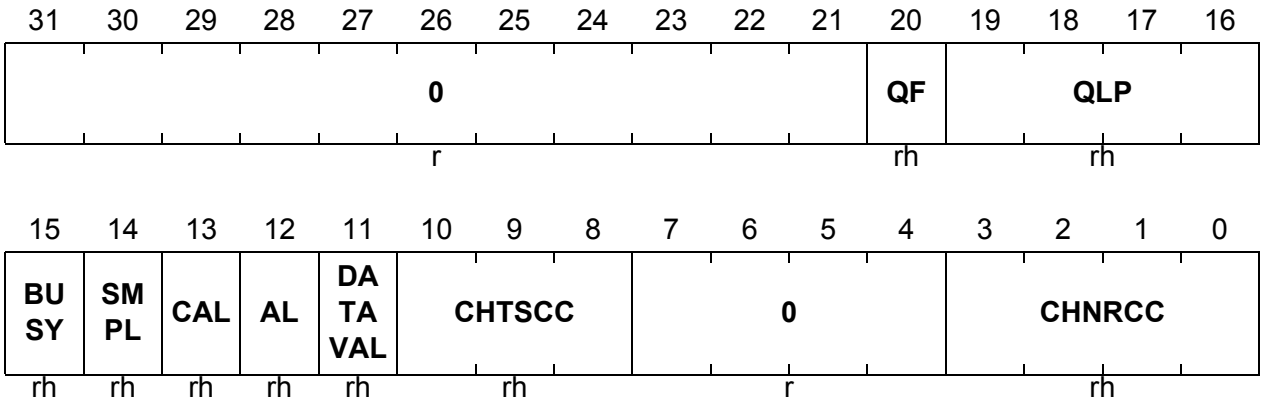
CON
Converter Control Register **(120_H)** **Reset Value: $0000\ 0001_H$**



Field	Bits	Type	Description
CTC	[7:0]	rw	**Conversion Time Control** This bit determines the period of the ADC basic operating clock f_{BC}. Any modification of this bit field is taken into account after the currently performed conversion is finished.
SCNM	[9:8]	rw	**Auto-Scan Mode** This bit enables the auto-scan mode. 00_B Auto-scan mode disabled 01_B Auto-scan single sequence mode enabled 10_B Auto-scan continuous sequence mode enabled 11_B Reserved
QEN	15	rh	**Queue Enable** This bit specifies if queue controlled conversions are enabled/disabled and queue based conversion requests are generated. 0_B Queue is disabled 1_B Queue is enabled *Note: The queue load is not affected by a queue disable condition.*
QWLP	[19:16]	rw	**Queue Warning Limit Pointer** The value of the QWLP specifies the queue element to be watched.

Field	Bits	Type	Description
PCDIS	27	rw	**Post Calibration Disable** This bit disables the automatic post calibration. Post calibration takes place after each conversion. The next conversion is started after the end of the post calibration. If post calibration, the actual conversion length increases by two f_{ANA} analog clock cycles due to the post calibration). 0_B The post calibration is enabled. 1_B The post calibration is disabled.
CPR	28	rw	**Clear of Pending Conversion Requests in Parallel Sources by Arbiter** Bit CPR determines whether or not all pending conversion requests for an A/D channel, indicated by STAT.CHNRCC, are cancelled by the arbiter, when the conversion for this channel has been started. 0_B The individual clear by arbiter is enabled. Only the conversion request of channel n of the winning source is cleared when a conversion of channel n is started. 1_B The global clear by arbiter is enabled. All conversion requests for channel n are cleared in parallel sources if a conversion of channel n is started.
SRTEST	31	rw	**Service Request Test Mode** This bit is used to set a source service request flag under software control.
0	[14:10], [26:20], [30:29]	r	**Reserved** Read as 0; should be written with 0.

STAT
Converter Status Register **($1B4_H$)** **Reset Value: $0000\ 0000_H$**

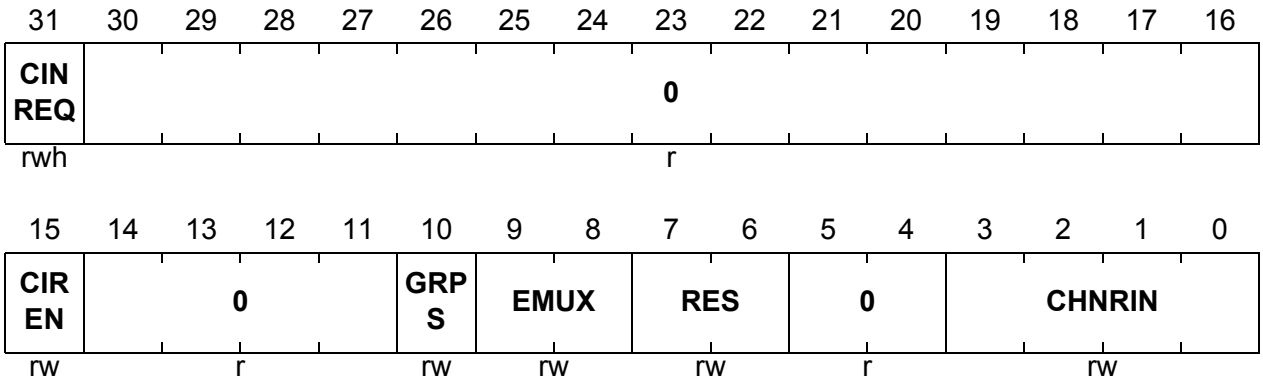


Field	Bits	Type	Description
CHNRCC	[3:0]	rh	**Number of Channel Currently Converted** This bit field indicates the number of the channel that is currently converted. 0000_B Channel 0 is currently converted. 0001_B Channel 1 is currently converted. $\ldots_B$ … 1110_B Channel 14 is currently converted. 1111_B Channel 15 is currently converted.
CHTSCC	[10:8]	rh	**Trigger Source of Channel Currently Converted** This bit field indicates the origin of a conversion request that triggered the channel currently converted. 000_B Channel Injection 001_B Timer 010_B Reserved 011_B External events 100_B Software SW0 101_B Reserved 110_B Queue 111_B Auto-scan
DATAVAL	11	rh	**Data Valid** This bit indicates whether conversion data is valid. It is set when the conversion is finished and cleared one f_{ADC} clock cycle after the next conversion has been started.

Field	Bits	Type	Description
AL	12	rh	**Arbitration Lock** This bit is set, if the timer running in Arbitration Lock Mode meets the value specified in TCON.ALB. Bit is cleared on timer underflow. 0_B Arbitration Lock Mode is inactive. 1_B Arbitration Lock Mode is active.
CAL	13	rh	**Power-Up Calibration Status** This bit indicates the status of the power-up calibration phase. 0_B Power-up calibration is finished. 1_B The ADC is in power-up calibration phase.
SMPL	14	rh	**Sample Phase Status** This bit indicates whether the ADC is in a sample phase or not. 0_B The ADC is currently not in the sample phase. 1_B The ADC currently samples the analog input voltage (sample phase).
BUSY	15	rh	**Busy Status** This bit indicates whether the ADC performs a conversion or not. 0_B The ADC is currently idle. 1_B The ADC currently performs a conversion.
QLP	[19:16]	rh	**Queue Level Pointer** This bit field points to the empty queue element with the lowest queue element number. It is incremented on a queue load operation; it is decremented after a queue based conversion is started.
QF	20	rh	**Queue Full Status** This bit is set on a write action to the last empty queue element. It is cleared if at least one queue element is empty. 0_B At least on queue element is empty. 1_B Queue is full.
0	[7:4], [31:21]	r	**Reserved** Read as 0.

23.2.8 Channel Inject Register

CHIN
Channel Injection Control Register (118_H) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
CHNRIN	[3:0]	rw	**Channel Number to be Injected** This bit field indicates the channel number of the analog channel which is used for channel injection.
RES	[7:6]	rw	**Conversion Resolution Control** This bit field controls the resolution of the A/D Converter for the conversion of the analog channel determined by CHNRIN. Any modification of this bit field is taken into account after the currently running conversion is finished. 00_B 10-bit resolution 01_B 12-bit resolution 10_B 8-bit resolution 11_B Reserved
EMUX	[9:8]	rw	**External Multiplexer Control** This bit indicates the settings of the external multiplexer control lines that is used during an A/D conversion for the analog channel determined by CHNRIN.
GRPS	10	rw	**Analog Input Multiplexer Group Select** This bit selects the analog input multiplexer group for the conversion of the analog channel determined by CHNRIN. 0_B Analog input multiplexer group 0 selected. 1_B Analog input multiplexer group 1 selected.

Field	Bits	Type	Description
CIREN	15	rw	**Cancel, Inject and Repeat Enable** This bit enables the Cancel, Inject and Repeat feature. 0_B Cancel, Inject and Repeat feature is disabled. 1_B Cancel, Inject and Repeat feature is enabled.
CINREQ	31	rwh	**Channel Injection Request** This bit is the request bit for Channel Injection. It is automatically cleared after the requested conversion is injected. 0_B No Channel Injection requested 1_B Channel Injection requested *Note: Clearing bit AP.CHP causes bit CHIN.CINREQ to be cleared.*
0	[5:4], [14:11], [30:16]	r	**Reserved** Read as 0; should be written with 0.

23.2.9 Software Request Registers

REQ0
Conversion Request Register SW0 (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
REQ0 15	REQ0 14	REQ0 13	REQ0 12	REQ0 11	REQ0 10	REQ0 9	REQ0 8	REQ0 7	REQ0 6	REQ0 5	REQ0 4	REQ0 3	REQ0 2	REQ0 1	REQ0 0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
REQ0n (n = 0-15)	n	rw	**Software SW0 Conversion Request for Channel n** 0_B No conversion is requested for channel n. 1_B A conversion is requested for channel n.
0	[31:16]	r	**Reserved** Read as 0.

SW0CRP
Software SW0 Conversion Request Pending Register

(180$_H$) Reset Value: 0000 0000$_H$

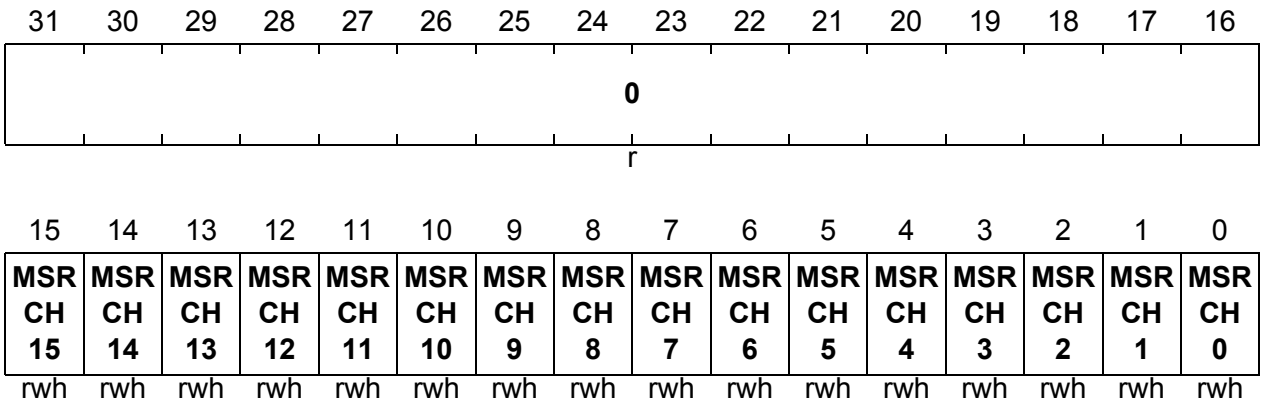
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SW0CRP15	SW0CRP14	SW0CRP13	SW0CRP12	SW0CRP11	SW0CRP10	SW0CRP9	SW0CRP8	SW0CRP7	SW0CRP6	SW0CRP5	SW0CRP4	SW0CRP3	SW0CRP2	SW0CRP1	SW0CRP0
rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
SW0CRPn (n = 0-15)	n	rh	**Software SW0 Conversion Request Pending Flag for Channel n** A SW0CRPn pending flag is set each time a conversion request is generated for channel n by SW0, that could not be serviced immediately. A start of the conversion for a pending request leads automatically to a clear of the pending flag. All pending request flags can also be cleared under software control, if bit AP.SW0P is cleared. 0_B No SW0 based conversion request is pending for channel n. 1_B A SW0 based conversion request is pending for channel n.
0	[31:16]	r	**Reserved** Read as 0.

23.2.10 Service Request Registers

MSS0
Module Service Request Status Register 0
(1D0$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
MSRCHn (n = 0-15)	n	rwh	**Module Service Request Status for Channel n** A MSRCHn pending flag determines if a channel service request has been generated by A/D Converter channel n. 0_B No channel service request has been generated by channel n. 1_B A channel service request has been generated by channel n. These bits are cleared by writing a 1 to the corresponding bit position.
0	[31:16]	r	**Reserved** Read as 0; should be written with 0.

MSS1
Module Service Request Status Register 1

(1D4$_H$) **Reset Value: 0000 0000$_H$**

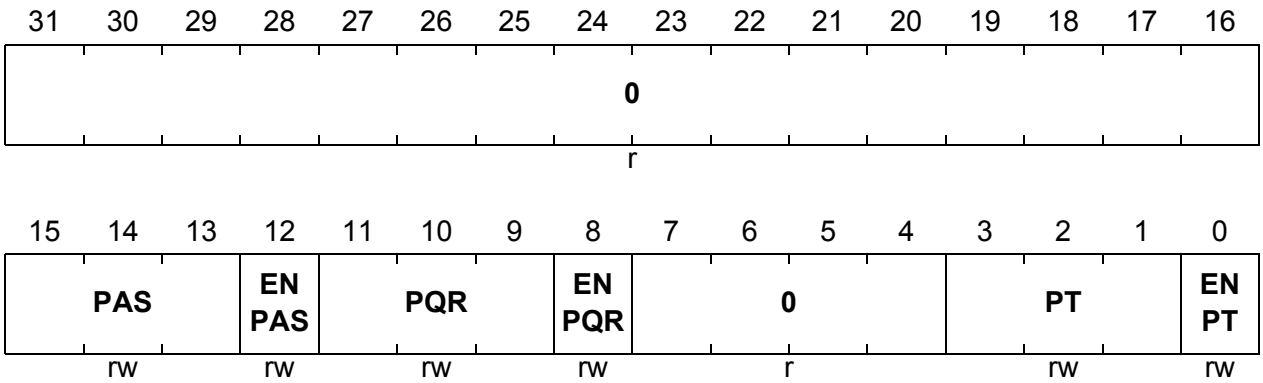
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0												MSRAS	MSRQR	0	MSRT
r												rwh	rwh	r	rwh

Field	Bits	Type	Description
MSRT	0	rwh	**Module Service Request Status for Source Timer** This bit specifies if a timer source service request has been generated. 0_B No timer source service request has been generated. 1_B A timer source service request has been generated. This bit is cleared by writing a 1 to its bit position.
MSRQR	2	rwh	**Module Service Request Status for Source Queue** This bit specifies if a queue source service request has been generated. 0_B No queue source service request has been generated. 1_B A queue source service request has been generated. This bit is cleared by writing a 1 to its bit position.
MSRAS	3	rwh	**Module Service Request Status for Source Auto-Scan** This bit specifies if a auto-scan source service request has been generated. 0_B No auto-scan source service request has been generated. 1_B A auto-scan source service request has been generated. This bit is cleared by writing a 1 to its bit position.

Field	Bits	Type	Description
0	[31:4], 1	r	**Reserved** Read as 0; should be written with 0.

SRNP
Service Request Node Pointer Register
(1DC$_H$) **Reset Value: 0000 0000$_H$**



Field	Bits	Type	Description
ENPT	0	rw	**Timer Service Request Enable** This bit enables the generation of a service request when a timer service request event occurs. 0_B Timer service request generation is disabled. 1_B Timer service request generation is enabled.
PT	[3:1]	rw	**Timer Service Request Node Pointer** These bit fields selects which service request output line will be activated when a timer service request event occurs and ENPT = 1. 000_B Service request output SR0 is selected. 001_B 0Service request output SR1 is selected. 010_B Service request output SR2 is selected. 011_B Service request output SR3 is selected. 100_B Service request output SR4 is selected. 101_B Service request output SR5 is selected. 110_B Service request output SR6 is selected. 111_B Service request output SR7 is selected.

Field	Bits	Type	Description
ENPQR	8	rw	**Queue Service Request Enable** This bit enables the generation of a service request when a queue service request event occurs. 0_B Queue service request generation is disabled. 1_B Queue service request generation is enabled.
PQR	[11:9]	rw	**Queue Service Request Node Pointer** This bit fields selects which service request output line will be activated if the queue service request becomes active and ENPQR = 1. 000_B Service request output SR0 is selected. 001_B Service request output SR1 is selected. 010_B Service request output SR2 is selected. 011_B Service request output SR3 is selected. 100_B Service request output SR4 is selected. 101_B Service request output SR5 is selected. 110_B Service request output SR6 is selected. 111_B Service request output SR7 is selected.
ENPAS	12	rw	**Auto-Scan Service Request Enable** This bit enables the generation of a service request when an auto-scan service request event occurs. 0_B Auto-scan service request generation is disabled. 1_B Auto-scan service request is enabled.
PAS	[15:13]	rw	**Auto-Scan Service Request Node Pointer** This bit fields selects which service request output line will be activated if the auto-scan service request becomes active and ENPAS = 1. 000_B Service request output SR0 is selected. 001_B Service request output SR1 is selected. 010_B Service request output SR2 is selected. 011_B Service request output SR3 is selected. 100_B Service request output SR4 is selected. 101_B Service request output SR5 is selected. 110_B Service request output SR6 is selected. 111_B Service request output SR7 is selected.
0	[31:16], [7:4]	r	**Reserved** Read as 0; should be written with 0.

23.3 Implementation of ADC0

This section describes the ADC0 module related external functions such as port connections, interrupt and service request control, DMA and GPTA connections, address decoding, and clock control.

23.3.1 Interface Connections of the ADC Module

Figure 23-27 shows the TC1766 specific implementation details and interconnections of the ADC0 module.

The ADC module is supplied with a separate clock control, address decoding, and interrupt control logic. Four of the eight module's service request outputs are connected to interrupt nodes and four to the DMA controller.

An external analog input pin AN31 is multiplexed with the die temperature measurement sensor to channel 31 of ADC0. The multiplexer is controlled by a bit field in register SCU_CON. Please refer to the User's Manual, System Control Unit Chapter.

The remaining 31 analog inputs are wired to the analog inputs of the ADC module with a fixed scheme. One analog input of ADC0 is connected to an on-chip temperature measurement unit. Control outputs for external analog input extensions are available at Port 1 as alternate outputs. The external request unit that is located in the TC1766 System Control Unit (SCU) controls A/D conversion trigger and gating input signals that are generated from port lines, MSC outputs, or GPTA outputs.

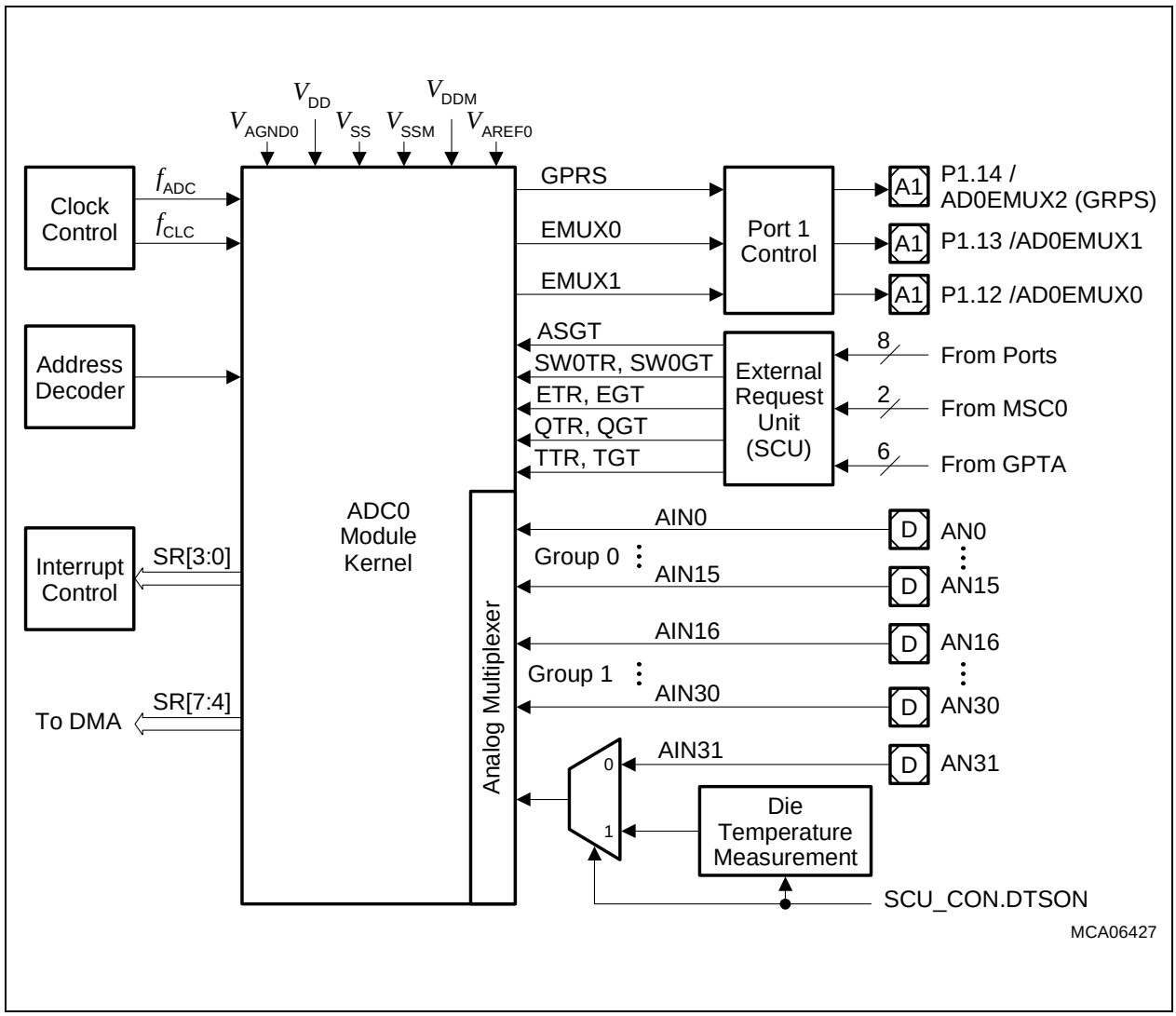


Figure 23-27 ADC0 Module Implementation and Interconnections

23.3.2 ADC0 Module Related External Registers

Figure 23-28 shows the implementation specific ADC0 external registers that can be used for ADC0 programming.

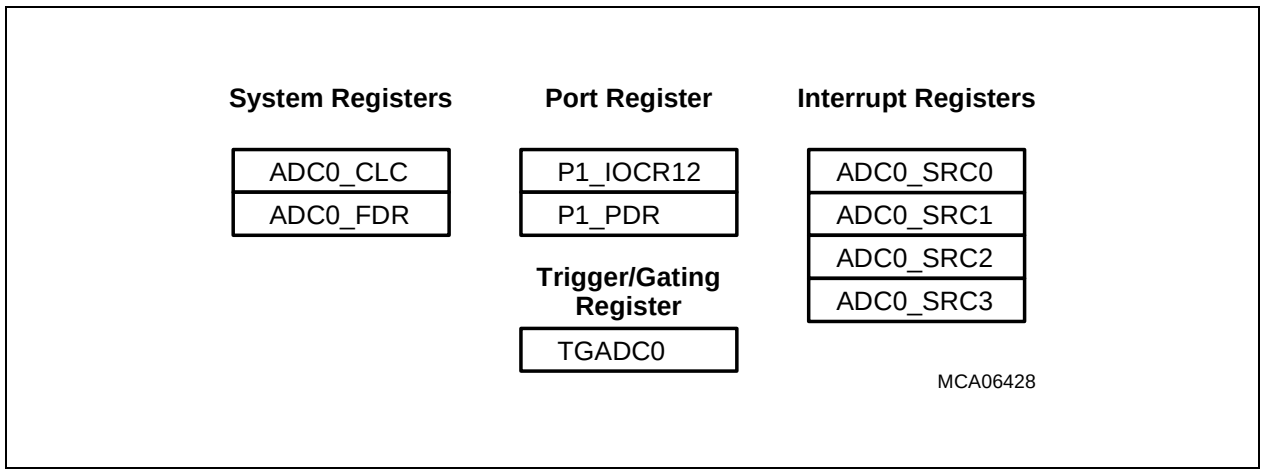


Figure 23-28 ADC0 Implementation-specific Special Function Registers

23.3.3 Clock Control

The ADC0 module is provided with the following clock signals:

- f_{CLC}
  This is the module control clock that is used inside the ADC0 kernel for control purposes such as for clocking of control logic and register operations. The frequency of f_{CLC} is equal to the system clock frequency f_{SYS}. The clock control register ADC0_CLC makes it possible to enable/disable f_{CLC} under certain conditions.
- f_{ADC}
  This clock is the module timing clock that is used in the ADC0 kernel as basic and timing reference clock for the analog part. The fractional divider registers ADC0_FDR controls the frequency of f_{ADC} and makes it possible to enable/disable it independently of f_{CLC}. The fractional divider's external clock enable feature is not used.

Signal RST_EXT_DIV of the fractional divider which is controlled by bit fields SC and DM of the ADC0_FDR register makes it possible to put the analog parts of the ADCs in its reset state. When setting ADC0_FDR.SC = 11_B (and afterwards back to the previous state of ADC0_FDR.SC), the analog parts of ADC0 are reset. This feature makes it possible, for example, to start a power-up calibration without a reset operation of the complete ADC.

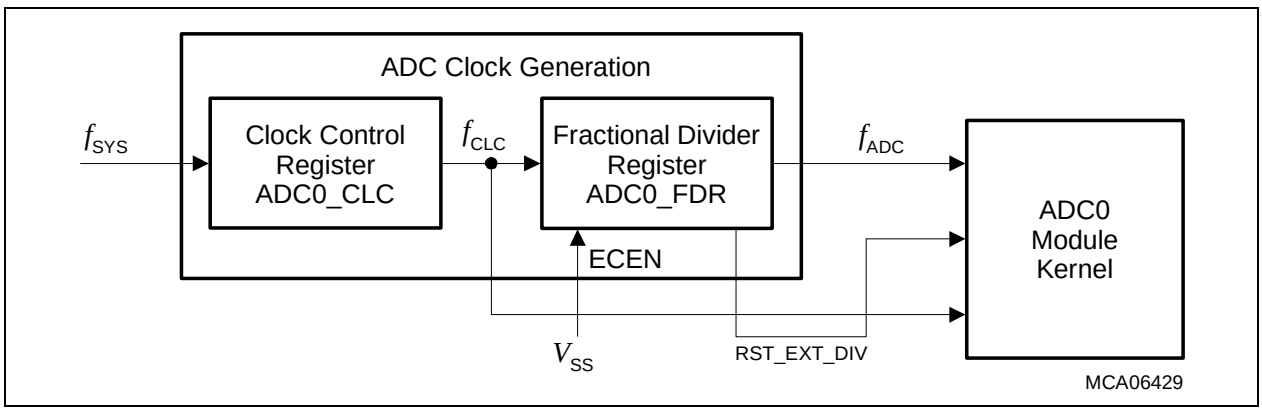


Figure 23-29 ADC0 Clock Generation

The following formulas define the frequency of f_{ADC}:

$$f_{ADC} = f_{SYS} \times \frac{1}{n} \quad \text{with n = 1024 - FDR.STEP} \tag{23.11}$$

$$f_{ADC} = f_{SYS} \times \frac{n}{1024} \quad \text{with n = 0-1023} \tag{23.12}$$

Equation (23.11) is valid for ADC0_FDR.DM = 01_B (normal divider mode).
Equation (23.12) is valid for ADC0_FDR.DM = 10_B (fractional divider mode).

23.3.3.1 ADC0 Clock Control Register

The clock control register makes it possible to control (enable/disable) the clock signal f_{CLC} under certain conditions. After a reset operation, the ADC0 module is disabled and the module clock signals f_{CLC} and f_{ADC} are switched off.

ADC0_CLC
ADC0 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

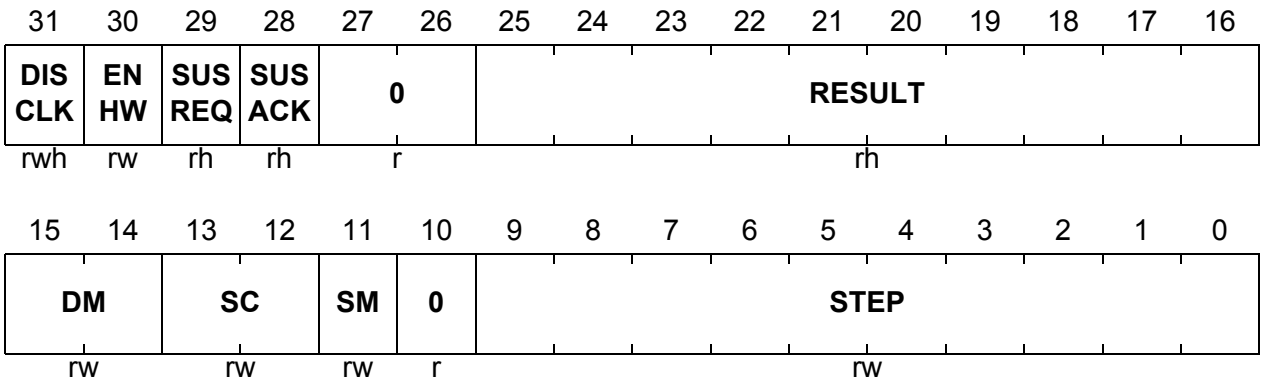
Field	Bits	Type	Description
DISR	0	rw	**Module Disable Request Bit** Used for enable/disable control of the module.
DISS	1	r	**Module Disable Status Bit** Bit indicates the current status of the module.
SPEN	2	rw	**Module Suspend Enable for OCDS** Used to enable the suspend mode.
EDIS	3	rw	**Sleep Mode 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.

*Note: Additional details on the clock control register functionality are described in section **"Clock Control Register CLC" on Page 3-24** of the TC1766 User's Manual System Units part (Volume 1).*

23.3.3.2 ADC0 Fractional Divider Register

The fractional divider register allows the programmer to control the clock rate of the module timing clock f_{ADC}.

ADC0_FDR
ADC0 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 $\overline{\text{SPNDACK}}$ 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]	rw	**Reserved** Read as 0; should be written with 0.

*Note: Additional details on the fractional divider register functionality are described in section **"Fractional Divider Operation" on Page 3-29** of the TC1766 User's Manual System Units part (Volume 1).*

23.3.4 Port Control

The external multiplexer control outputs and the group select output GRPS of the ADC module are controlled in the port logics. For these output lines the following port control operations must be executed:

- Input/output function selection (IOCR registers)
- Pad driver characteristics selection for outputs (PDR registers)

Input/Output Function Selection

The port input/output control registers contain the bit fields that select the ADC's digital I/O driver characteristics such as port direction (input/output), push-pull/open-drain capabilities, and alternate output selections. The output lines for the ADC module are controlled by the port input/output control register P1_IOCR12.

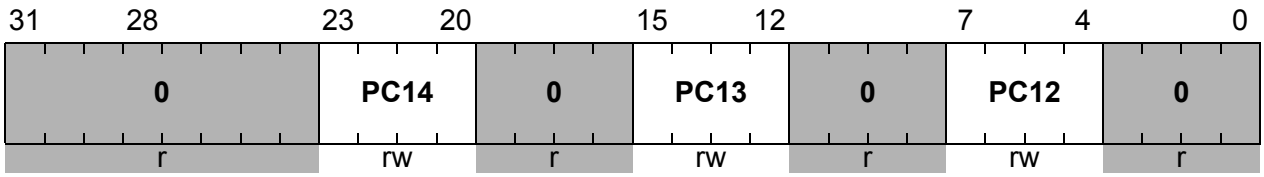
Table 23-13 shows how bits and bit fields must be programmed for the required I/O functionality of the ADC0 I/O lines.

Table 23-13 ADC0 I/O Control Selection and Setup

Module	Port Lines	Input/Output Control Register Bits	I/O
ADC0	P1.12/AD0EMUX0	P1_IOCR12.PC1 = $1X01_B$	Output
		P1_IOCR12.PC1 = $1X10_B$	
	P1.13/AD0EMUX1	P1_IOCR13.PC2 = $1X01_B$	Output
		P1_IOCR13.PC2 = $1X10_B$	
	P1.14/AD0EMUX2	P1_IOCR14.PC3 = $1X01_B$	Output
		P1_IOCR14.PC3 = $1X10_B$	

23.3.4.1 I/O Control Register

P1_IOCR12
Port 1 Input/Output Control Register 12(18_H) **Reset Value: 2020 2020_H**



Field	Bits	Type	Description
		rw	**Port Output Control for Port 1 Pins 12-14**[1)] These bit fields determine the output port functionality:
PC12	[7:4]		Port input/output control for P1.12/AD0EMUX0
PC13	[15:12]		Port input/output control for P1.13/AD0EMUX1
PC14	[23:20]		Port input/output control for P1.14/AD0EMUX2

1) For coding of bit field, see **Table 23-14**. Shaded bits and bit fields are “don’t care” for ADC I/O port control.

Table 23-14 PCx Coding for ADC Outputs

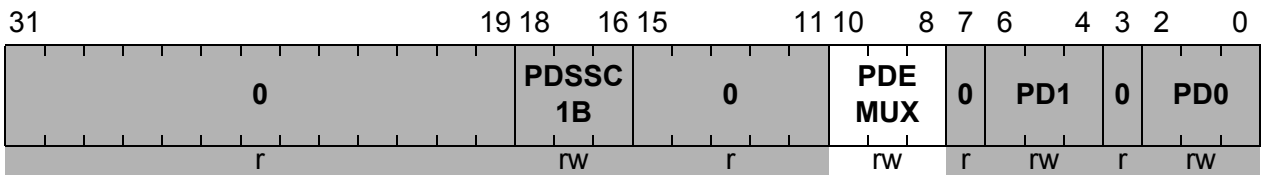
PCx[3:0]	I/O	Output Characteristics	Selected Pull-up/Pull-down/ Selected Output Function
$0X00_B$	Input	–	No pull device connected
$0X01_B$			Pull-down device connected
$0X10_B$[1)]			Pull-up device connected
$0X11_B$			No pull device connected
1001_B	Output	Push-pull	Output function ALT1
1101_B		Open-drain	Output function ALT1
1010_B		Push-pull	Output function ALT2
1110_B		Open-drain	Output function ALT2
1011_B		Push-pull	Output function ALT3
1111_B		Open-drain	Output function ALT3

1) This bit field value is default after reset.

23.3.4.2 Pad Driver Mode Register

The Port 1 pad driver mode register contains bit fields that determine the output driver strength and the slew rate of ADC external multiplexer control outputs. (Class A1 outputs).

P1_PDR
Port 1 Pad Driver Mode Register (40_H) Reset Value: $0000\ 0000_H$



Field	Bits	Type	Description
PDEMUX	[10:8]	rw	**Pad Driver Mode for P1.[14:12]/AD0EMUX[2:0]** $XX0_B$ Medium driver selected for P1.[14:12] $XX1_B$ Weak driver selected for P1.[14:12]

23.3.5 Analog Input Line to Analog Input Channel Connections

Table 23-15 defines the connections between the analog input lines AN[31:0] of the TC1766 and the analog inputs AIN[31:0] of the ADC0 module. The bit GRPS (group select) located in several kernel registers determines the analog input multiplexer group number that is used for channel selection.

Table 23-15 AN[31:0] to ADC0 Analog Input Connections

Analog Input Pin of TC1766	ADC0 Module		Analog Input Pin of TC1766	ADC0 Module	
	Connected to Analog Input of Module	GRPS		Connected to Analog Input of Module	GRPS
AN0	AIN0	0	AN16	AIN16	1
AN1	AIN1		AN17	AIN17	
AN2	AIN2		AN18	AIN18	
AN3	AIN3		AN19	AIN19	
AN4	AIN4		AN20	AIN20	
AN5	AIN5		AN21	AIN21	
AN6	AIN6		AN22	AIN22	
AN7	AIN7		AN23	AIN23	
AN8	AIN8		AN24	AIN24	
AN9	AIN9		AN25	AIN25	
AN10	AIN10		AN26	AIN26	
AN11	AIN11		AN27	AIN27	
AN12	AIN12		AN28	AIN28	
AN13	AIN13		AN29	AIN29	
AN14	AIN14		AN30	AIN30	
AN15	AIN15		AN31	AIN31/On-chip Die Temperature Sensor	

23.3.6 On-Chip Connections

This section describes the on-chip interconnections of the ADC0 module.

23.3.6.1 Reference Voltage Selection

Generally, the ADC module allows the analog reference voltage for each analog channel to be selected via the CHCONn.REF bit field. In the TC1766, ADC0 is able to select among three analog reference voltages: V_{AREF0}, AIN0, and AIN1 (see **Figure 23-30**).

The selection ADC0_CHCONn.REF = 11_B must not be used.

For safety reasons, the measurement of the ADC channels 0-15 (GRPS = 0) is always related to the reference input pin (the programmed selection is not taken into account). The measurement of the ADC channels 16-31 (GRPS = 1) is always related to the programmed reference (the programmed selection is taken into account).

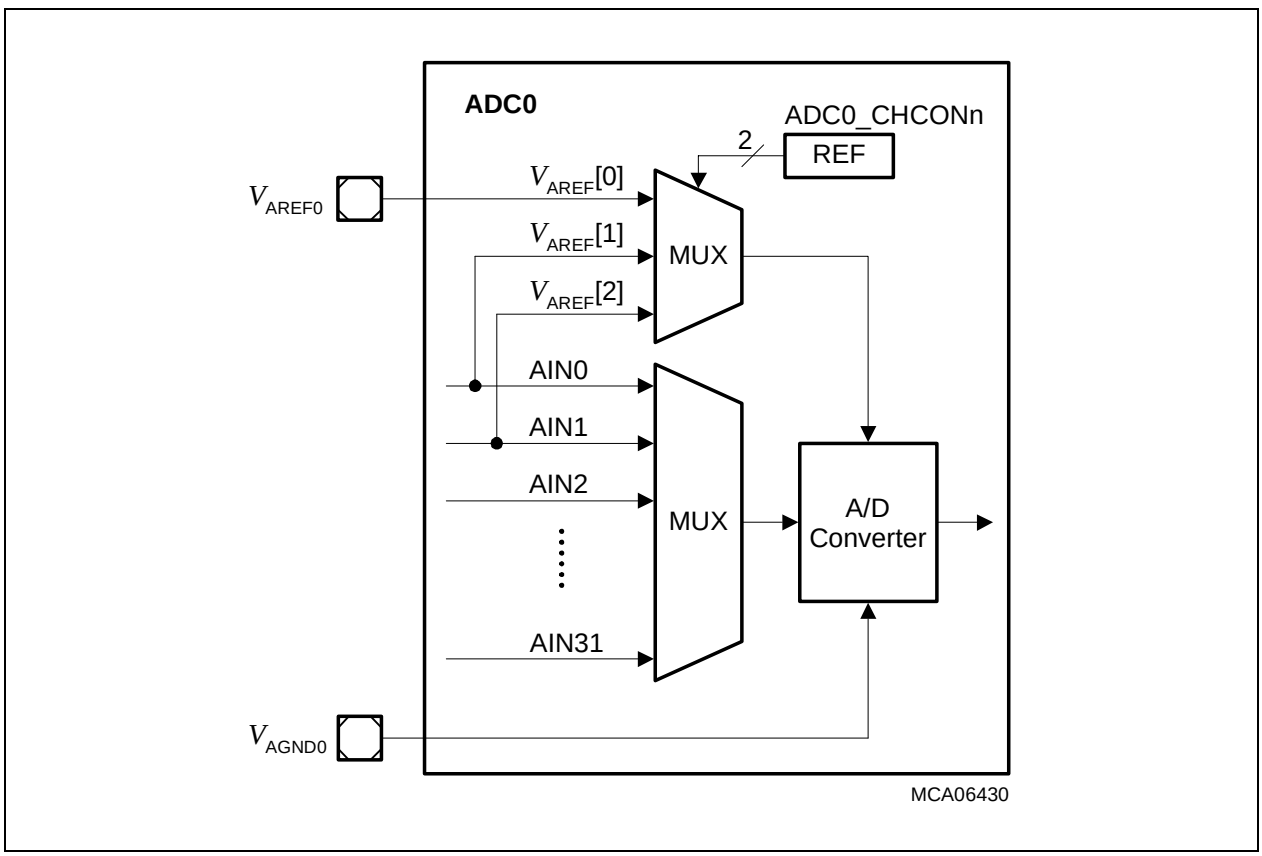


Figure 23-30 ADC0 Reference Voltage Selections

23.3.6.2 Request/Gating Input Signal Connections

The on-chip interconnections of the A/D Converters request and trigger inputs are shown in **Figure 23-31**. Request and gating signals for ADC0, derived by pins, MSC outputs, or GPTA outputs are generated by the External Request Unit (ERU). The ERU is located in the System Control Unit (SCU) of the TC1766. Three output signals of GPTA0 are able to directly trigger A/D Converter channels.

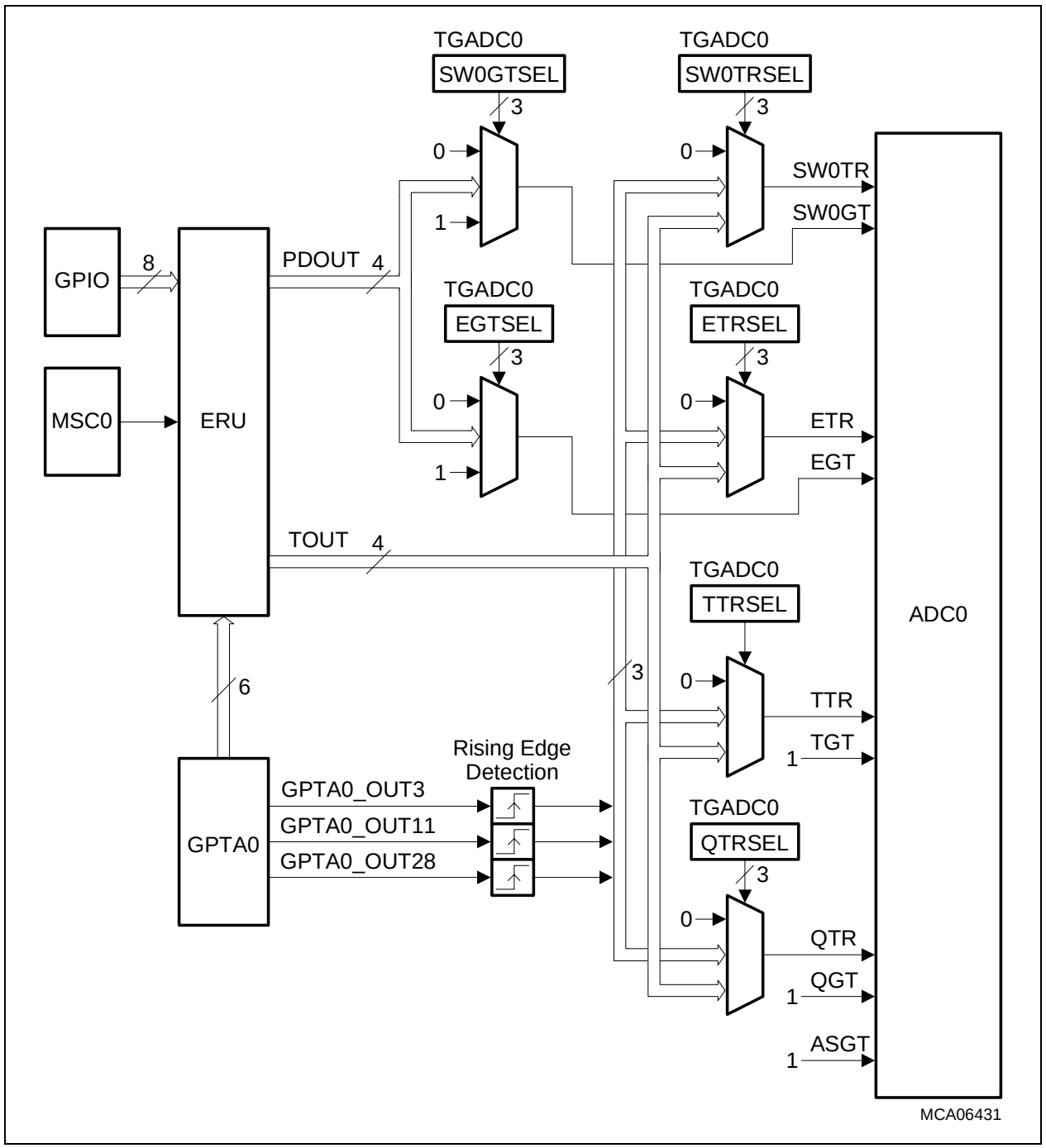


Figure 23-31 Request/Gating Input Signal Connections (ADC0)

The signal source for trigger and gating signals of the A/D Converter, ADC0, is selected via register TGADC0.

Three gating input signals are connected to high level. This means, that its gating functionality is not used and trigger requests are always passed in the corresponding trigger control logic.

Trigger inputs can be completely disabled, connected to the TOUT signals of the ERU, or connected to one of three GPTA0 output signals. This third selection capability makes it possible to directly trigger A/D conversions with GPTA output signal transitions. When connected to the TOUT signals of the ERU, trigger signal source and behavior must be selected in the ERU. The functionality of the ERU is described in detail in Chapter 5 “System Control Unit” of the TC1766 System Units User’s Manual.

Table 23-16 summarizes the trigger/gating capabilities for each trigger source and gives references to the figures in which the functionality of trigger/gating signals are shown.

Table 23-16 Trigger/Gating Source Input Selection

Trigger Source	Trigger/Gating Input	Selected Trigger/Gating Source
Timer (see **Figure 23-5**)	TTR	ERU trigger outputs TROUT[3:0] or three GPTA0 channels direct trigger outputs; selected by TGADC0.TTRSEL
	TGT	TGT = 1; timer gating always enabled;
External Event (see **Figure 23-7**)	ETR	ERU trigger outputs TROUT[3:0] or three GPTA0 channels direct trigger outputs; selected by TGADC0.ETRSEL
	EGT	non-inverted or inverted ERU pattern detection outputs PDOUT[3:1]; selected by TGADC0.EGTSEL
Software (see **Figure 23-8**)	SW0TR	ERU trigger outputs TROUT[3:0] or three GPTA0 channels direct trigger outputs; selected by TGADC0.SW0TRSEL
	SW0GT	non-inverted or inverted ERU pattern detection outputs PDOUT[0,2,3]; selected by TGADC0.SW0GTSEL
Auto-Scan (see **Figure 23-9**)	ASGT	ASGT = 1; auto-scan gating always enabled;

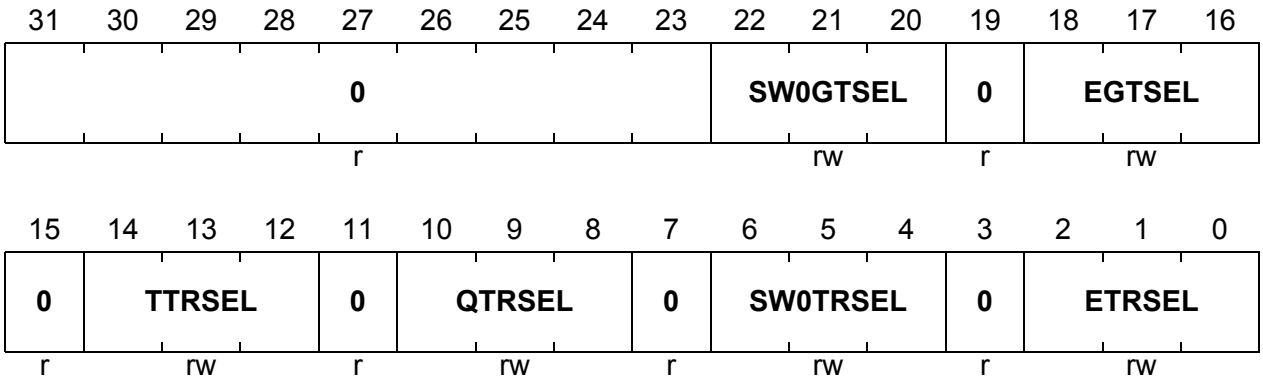
Table 23-16 Trigger/Gating Source Input Selection (cont'd)

Trigger Source	Trigger/Gating Input	Selected Trigger/Gating Source
Queue (see **Figure 23-15** and **Figure 23-16**)	QTR	ERU trigger outputs TROUT[3:0] or three GPTA0 channels direct trigger outputs; selected by TGADC0.QTRSEL
	QGT	QGT = 1; queue gating always enabled;

Trigger Gating Register

The registers TGADC0 contain bit fields that determine which output signals of the external request unit and the GPTA are used as trigger and gating inputs of ADC0.

TGADC0
Trigger Gating ADC0 Register ($9C_H$) Reset Value: 0000 0000_H



Field	Bits	Type	Description
ETRSEL	[2:0]	rw	**External Trigger Request Selection** This bit determines which trigger source will be used for the ADC0 external trigger request input ETR. 000_B ETR = 0; no trigger function of ETR. 001_B ETR is connected to GPTA0_OUT3. 010_B ETR is connected to GPTA0_OUT11. 011_B ETR is connected to GPTA0_OUT28. 100_B ETR is connected to TOUT0. 101_B ETR is connected to TOUT1. 110_B ETR is connected to TOUT2. 111_B ETR is connected to TOUT3.
SW0TRSEL	[6:4]	rw	**SW0 Trigger Request Selection** This bit determines which trigger source will be used for the ADC0 SW0 trigger request input. 000_B SW0TR = 0; no trigger function of SW0TR. 001_B SW0TR is connected to GPTA0_OUT3. 010_B SW0TR is connected to GPTA0_OUT11. 011_B SW0TR is connected to GPTA0_OUT28. 100_B SW0TR is connected to TOUT0. 101_B SW0TR is connected to TOUT1. 110_B SW0TR is connected to TOUT2. 111_B SW0TR is connected to TOUT3.

Field	Bits	Type	Description
QTRSEL	[10:8]	rw	**Queue Trigger Request Selection** This bit determines which trigger source will be used for the ADC0 queue trigger request input QTR. 000_B QTR = 0; no trigger function of QTR. 001_B QTR is connected to GPTA0_OUT3. 010_B QTR is connected to GPTA0_OUT11. 011_B QTR is connected to GPTA0_OUT28. 100_B QTR is connected to TOUT0. 101_B QTR is connected to TOUT1. 110_B QTR is connected to TOUT2. 111_B QTR is connected to TOUT3.
TTRSEL	[14:12]	rw	**Timer Trigger Request Selection** This bit determines which trigger source will be used for the ADC0 timer trigger request input TTR. 000_B TTR = 0; no trigger function of TTR. 001_B TTR is connected to GPTA0_OUT3. 010_B TTR is connected to GPTA0_OUT11. 011_B TTR is connected to GPTA0_OUT28. 100_B TTR is connected to TOUT0. 101_B TTR is connected to TOUT1. 110_B TTR is connected to TOUT2. 111_B TTR is connected to TOUT3.
EGTSEL	[18:16]	rw	**External Gating Selection** This bit determines which trigger source will be used for the ADC0 external gating input EGT. 000_B EGT = 0; conversion request source “External Event” is permanently disabled. 001_B EGT is connected to PDOUT1. 010_B EGT is connected to PDOUT2. 011_B EGT is connected to PDOUT3. 100_B EGT = 1; conversion request source “External Event” is permanently enabled. 101_B EGT is connected to $\overline{\text{PDOUT1}}$. 110_B EGT is connected to $\overline{\text{PDOUT2}}$. 111_B EGT is connected to $\overline{\text{PDOUT3}}$.

Field	Bits	Type	Description
SW0GTSEL	[22:20]	rw	**SW0 Gating Selection** This bit determines which trigger source will be used for the ADC0 software gating input SW0GT. 000_B SW0GT = 0; conversion request source “Software” is permanently disabled. 001_B SW0GT is connected to PDOUT0. 010_B SW0GT is connected to PDOUT2. 011_B SW0GT is connected to PDOUT3. 100_B SW0GT = 1; conversion request source “Software” is permanently enabled. 101_B SW0GT is connected to $\overline{\text{PDOUT0}}$. 110_B SW0GT is connected to $\overline{\text{PDOUT2}}$. 111_B SW0GT is connected to $\overline{\text{PDOUT3}}$.
0	3, 7, 11, 15, 19, [31:23]	r	**Reserved** Read as 0; should be written with 0.

23.3.6.3 Service Request Output Lines

The ADC0 module has eight service request outputs. Four of these service request outputs (SR[3:0]) are connected to interrupt service request nodes while the remaining four service request outputs (SR[7:4]) are connected as DMA request line with the DMA controller of the TC1766 (see **Table 23-17**).

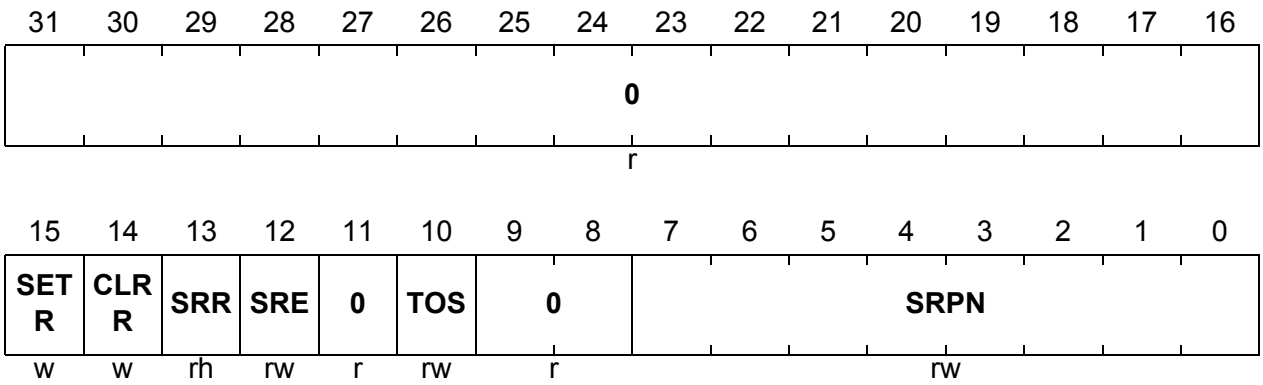
Table 23-17 Service Request Lines of ADC0/ADC1

Module	Service Request Line	Connected To	Description
ADC0	SR0	ADC0_SRC0	ADC0 Service Request Node 0
	SR1	ADC0_SRC1	ADC0 Service Request Node 1
	SR2	ADC0_SRC2	ADC0 Service Request Node 2
	SR3	ADC0_SRC3	ADC0 Service Request Node 3
	SR4	CH00_REQI3	DMA Channel 00 Request Input 3
	SR5	CH01_REQI3	DMA Channel 01 Request Input 3
	SR6	CH02_REQI3	DMA Channel 02 Request Input 3
	SR7	CH03_REQI3	DMA Channel 03 Request Input 3

23.3.6.4 Service Request Control Register

Each of the four interrupt request nodes of ADC0 are controlled by a service request control register.

ADC0_SRCm (m = 0-3)
ADC0 Service Request Control Register m
($1FC_H$-m*4) **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 on **Page 12-3** of the TC1766 User's Manual System Units part (Volume 1).*

23.3.6.5 Die Temperature Sensor

AIN31 of ADC0 is connected to the output of an on-chip die temperature measurement sensor (see also **Page 23-86**). Detailed information about this die temperature sensor is provided in the section **"Die Temperature Sensor" on Page 5-48** of the TC1766 User's Manual System Unit (Volume 1).

24 Fast Analog-to-Digital Converter (FADC)

The TC1766 contains a medium speed Analog-to-Digital Converters (ADC0) and a Fast Analog-to-Digital Converter (FADC).

ADC0 provides 2-3 µs conversion time (10-bit) and is intended mainly for single-ended signals. The FADC offers conversion rates of less than 500 ns, thus allowing sampling of high frequency signals. For slow and mid-range frequency signals, heavy over-sampling can be performed to avoid the usage of expensive filters.

ADC0 is described in **Chapter 23**. This chapter describes in detail the FADC and contains the following sections:

- Functional description of the FADC Kernel (see **Page 24-2**)
- FADC kernel register descriptions (see **Page 24-28**)
- TC1766 implementation-specific details and registers of the FADC module, including on-chip interconnections, service request control, address decoding, and clock control (see **Page 24-55**)

*Note: The FADC Kernel register names described in **Section 24.2** are referenced in the TC1766 User's Manual by the module name prefix "FADC_" for the FADC interface.*

24.1 FADC Kernel Description

The on-chip FADC module of the TC1766 is primarily a two-channel A/D Converter with 10-bit resolution that operates by the method of successive approximation.

Features

- Extreme fast conversion, 21 cycles of f_{FADC} clock (262.5 ns @ f_{FADC} = 80 MHz)
- 10-bit A/D conversion
  – Higher resolution by averaging of consecutive conversions is supported
- Successive approximation conversion method
- Two differential input channels
- Offset and gain calibration support for each channel
- Differential input amplifier with programmable gain of 1, 2, 4 and 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 anti-aliasing and data reduction filter block

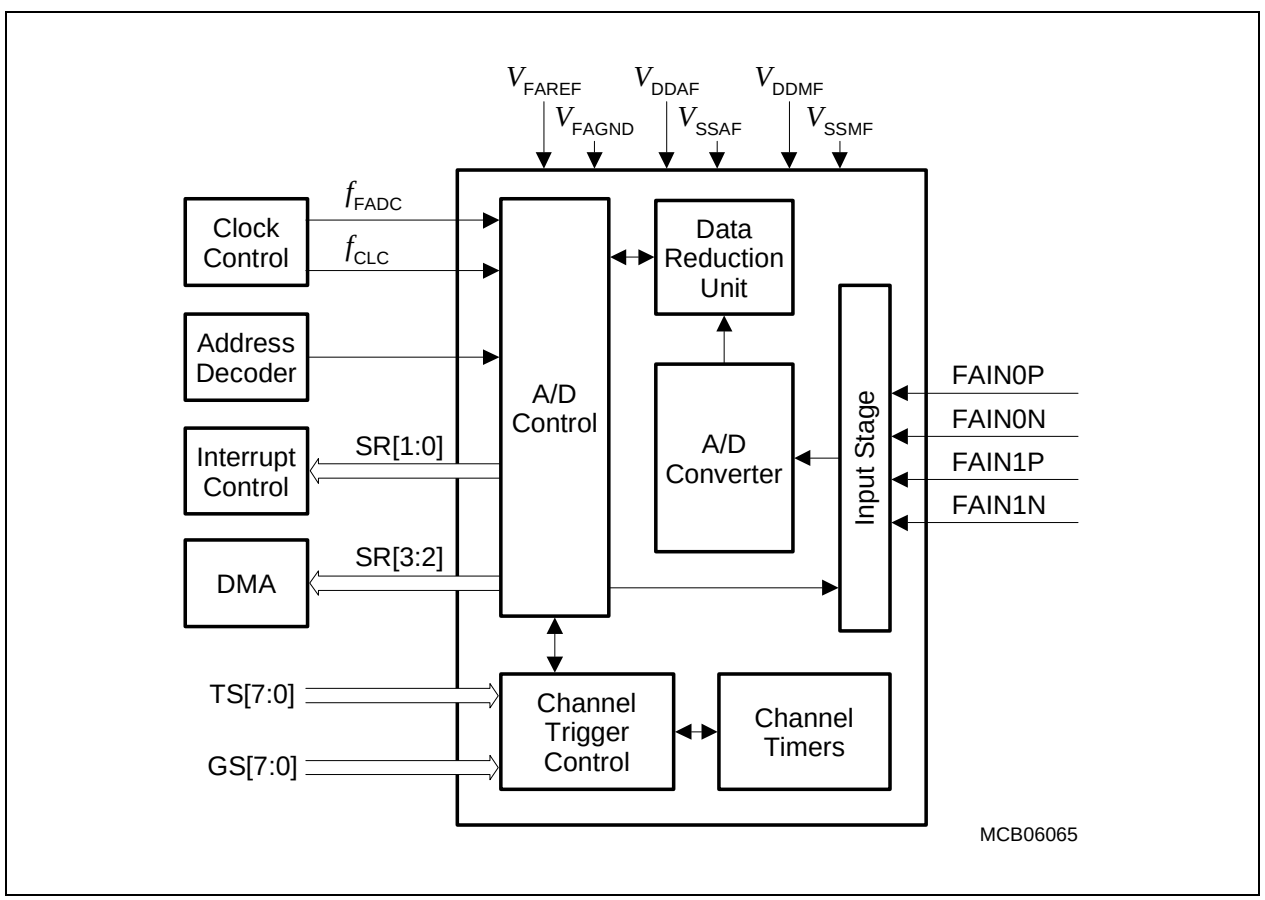


Figure 24-1 Block Diagram of the FADC Module

As shown in **Figure 24-1**, the main FADC functional blocks are:

- The Input Stage – contains the differential inputs and the programmable amplifier
- The A/D Converter – is responsible for the analog-to-digital conversion
- The Data Reduction Unit – contains programmable antialiasing and data reduction filters
- The Channel Trigger Control block – determines the trigger and gating conditions for the two FADC channels
- The Channel Timers – can independently trigger the conversion of each FADC channel
- The A/D Control block is responsible for the overall FADC functionality

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

- V_{DDMF}/V_{DDMF}: FADC Analog Part Power Supply (3.3 V)
- V_{DDAF}/V_{DDAF}: FADC Analog Part Logic Power Supply (1.5 V)
- V_{FAREF}/V_{FAGND}: FADC Reference Voltage (3.3 V) / FADC Reference Ground

24.1.1 Analog Inputs

Two analog inputs are assigned to each of the two FADC channels. The FADC analog input stage contains a differential channel amplifier for each channel and a common differential amplifier controlling the gain. The two input lines FAINxP/FAINxN (x = 0-1) of each channel analog input stage can be enabled independently in order to support single-ended measurements.

The gain of the common amplifier used during an A/D conversion is selected individually for each of the two channels depending on the currently active channel x.

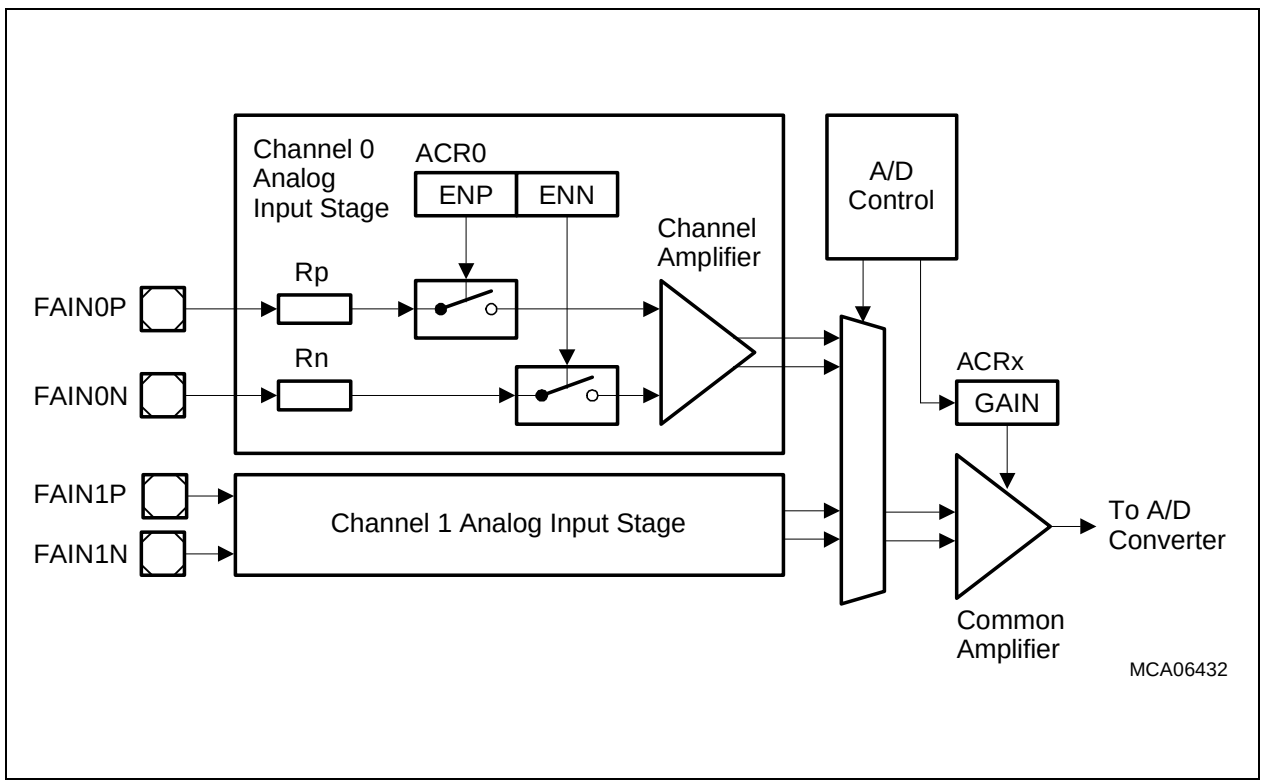


Figure 24-2 Analog Input Stage

In Multiplexer Test Mode (GCR.MUXTM = 1), the channel amplifiers are disconnected from the common amplifier. The measured conversion result in multiplexer test mode should be 10 0000 0000_B = (200_H) plus/minus the offset of the common amplifier.

24.1.1.1 Analog Input Stage Configurations

The analog input stage makes it possible to select different configurations to be selected independently for each FADC channel x. These combinations are shown in **Figure 24-3**.

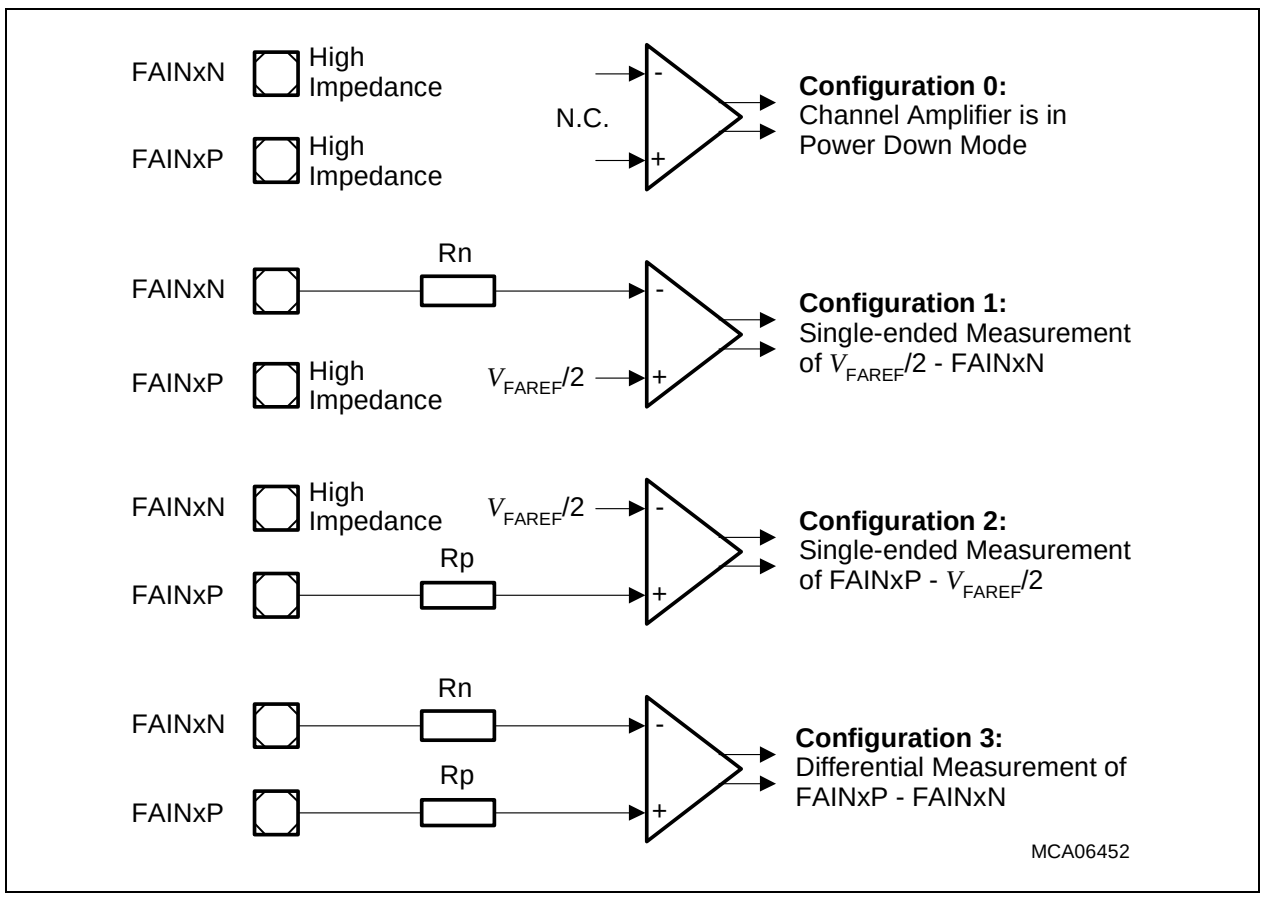


Figure 24-3 Analog Input Stage Configurations

Configuration 0 (ACRx.ENP = ACRx.ENN = 0)

FADC channel x inputs FAINxP and FAINxN are in a high impedance state and the channel amplifier of the analog input stage is in power-down mode.

Configuration 1 (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 (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 (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.

Table 24-1 provides a summary of the conversion results that are available at the different measurement modes. This table assumes a reference voltage of V_{FAREF} = 3.3 V.

Table 24-1 Conversion Results in the Different Measurement Modes

Measurements	ACRx. ENP	ACRx. ENN	FAINxP	FAINxN	ACRx. GAIN	Conversion Results
Single-ended Measurement Mode (Configuration 1)	0	1	"don't care"	0	00_B	768
				3.3		256
				0	01_B	1023
				3.3		0
Single-ended Measurement Mode (Configuration 2)	1	0	0	"don't care"	00_B	256
			3.3			768
			0		01_B	0
			3.3			1023
Differential Measurement Mode (Configuration 3)	1	1	0	1.65	00_B	256
			0	3.3		0
			1.65	0	01_B	768
			3.3	0		1023
			3.3	1.65	00_B	768
			0	1.65		0
			1.65	0		1023
			3.3	1.65	01_B	1023

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 TC1766 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.

The conversion result for FADC channel x is given by the following equation:

$$\text{Conversion Result } V_{Mx} = \text{GAIN}_x \left((V_{FAINxP} - V_{FAREFM}) + (V_{FAREFM} - V_{FAINxN}\right) \quad \text{with } V_{FAREFM} = V_{FAGND} + (V_{FAREF} - V_{FAGND})/2 \tag{24.1}$$

The absolute value of the result V_{Mx} is limited to V_{FAREF}. The mapping of the conversion result V_{Mx} to the binary result RCHx.ADRES is as follows (see also **Table 24-1**):

V_{Mx}	= -V_{FAREF}	leads to RCHx.ADRES = 00 0000 0000_B
V_{Mx}	= 0	leads to RCHx.ADRES = 10 0000 0000_B
V_{Mx}	= +V_{FAREF}	leads to RCHx.ADRES = 11 1111 1111_B

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 $V_{FAINxN} = V_{FAREFM}$

24.1.2 Conversion Timing

The conversion time of the FADC is determined by the frequency of clock f_{FADC}. The conversion time defined below includes sampling, converting, and storing of the conversion result.

$$\text{Conversion Time}_{FADC} = 21 \times \frac{1}{f_{FADC}} \tag{24.2}$$

Clock f_{FADC} is generated outside the FADC kernel in a product specific clock control unit (see **Page 24-56**).

24.1.3 Channel Triggers

As shown in **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 signals TS[7:0]
- 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 (controlling the external gating source inputs GS[7:0]) 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.

This control logic does the following control tasks:

- 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 control

All these control tasks are executed independently in each of the two FADC channels.

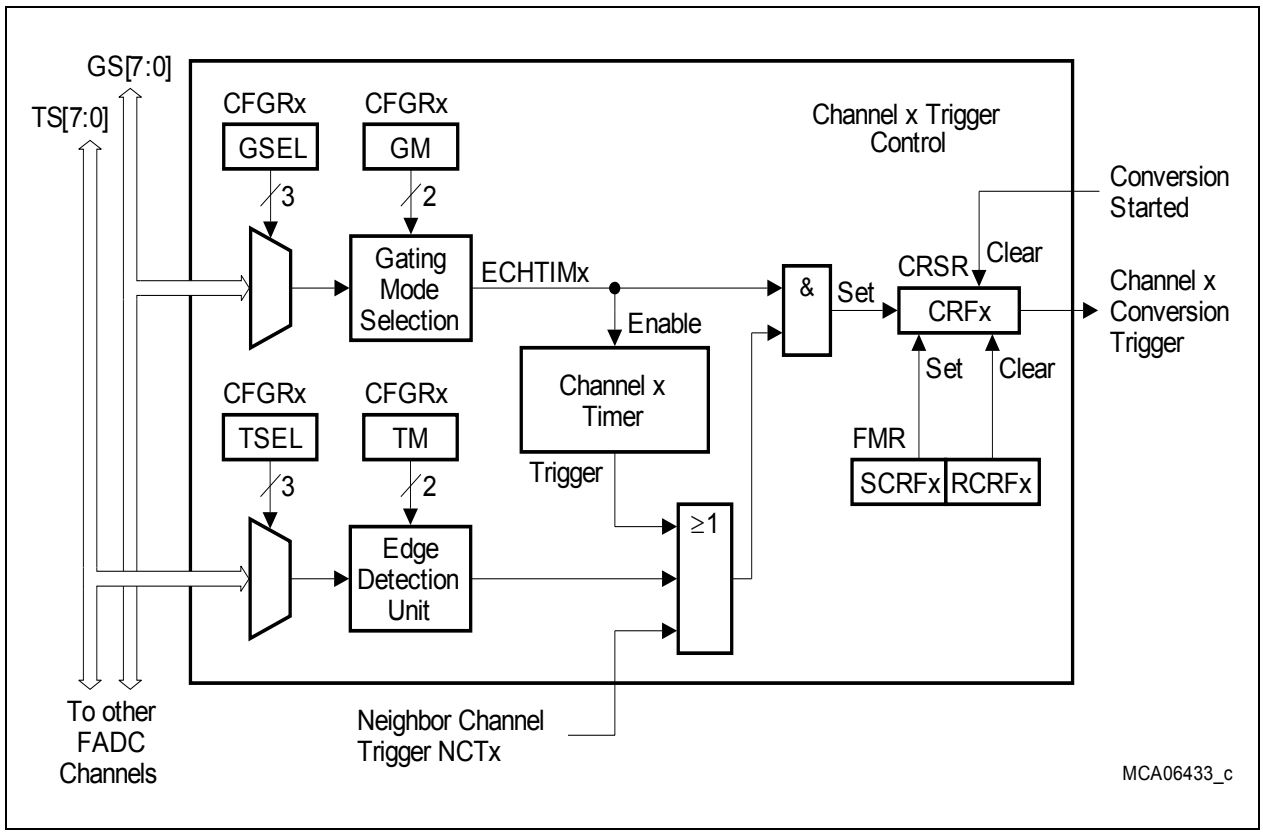


Figure 24-4 Channel Trigger Control Logic

Table 24-2 describes the possible gating modes (enabled, disabled, active gating source input polarity) of an FADC channel.

Table 24-2 Gating Modes

CFGRx.GM	Gating Mode
00_B	Conversion requests are disabled and Channel Timer is stopped. CRFx never becomes set (by hardware).
01_B	Conversion requests and Channel Timer are always enabled. 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 flag 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 flag 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 24-3 Trigger Modes

CFGRx.TM	Trigger Mode
00_B	No trigger signal generated. Edge detection unit always delivers a 0.
01_B	A conversion request trigger signal is generated on a rising edge of trigger source input TSn (as selected by CFGRx.TSEL).
10_B	A conversion request trigger signal is generated on a falling edge of trigger source input TSn (as selected by CFGRx.TSEL).
11_B	A conversion request trigger signal is generated on a rising or falling edge of trigger source input TSn (as selected by CFGRx.TSEL).

The Conversion Request Flag CRFx is cleared by hardware when the conversion of channel x is started. CRFx can also be 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).

24.1.4 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 flag CRSR.CRFx is also 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, 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.

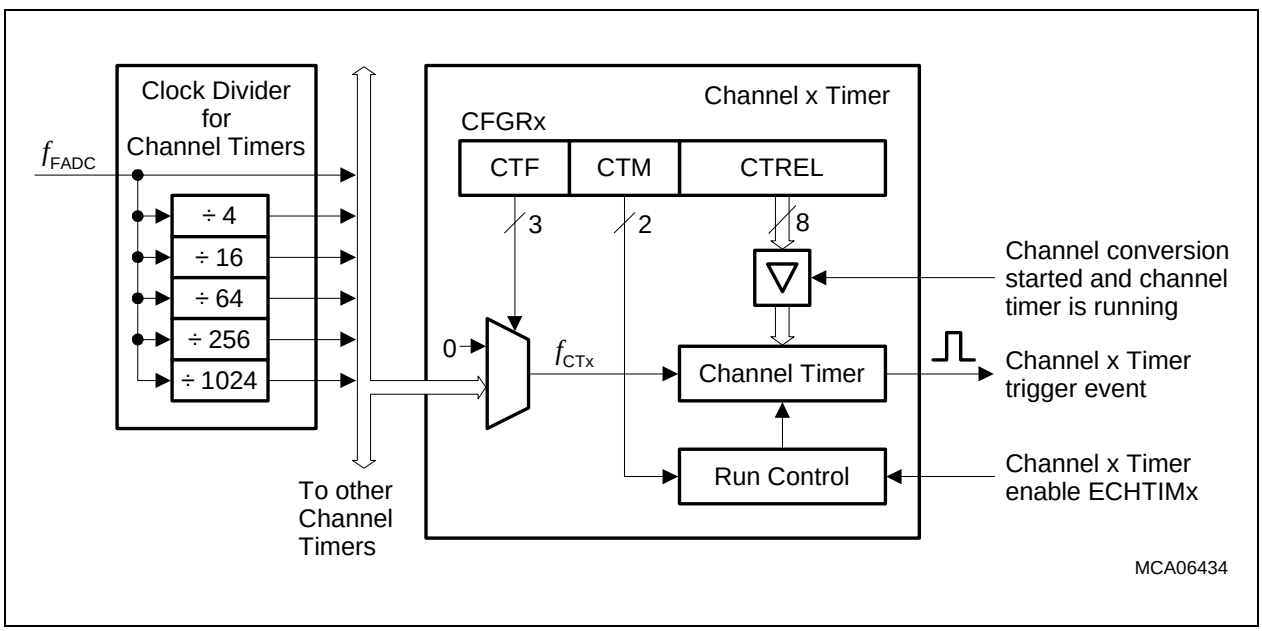


Figure 24-5 Channel Timer Block Diagram

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 Timer x 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_{CLC} / f_{CTx}:

$f_{CLC} / f_{CTx} = 1$ Channel x timer divide factor = CTREL + 2

$1 < f_{CLC} / f_{CTx} < 2$ Not recommended to be used!

$2 \leq f_{CLC} / f_{CTx}$ Channel x timer divide factor = CTREL + 1

In case of a f_{CLC} / 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.

24.1.5 Control Logic

24.1.5.1 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 cleared by hardware when the conversion starts.

24.1.5.2 Static Channel Priority

If more than one conversion request flags CRSR.CRFx (x = 0-1) 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 24-4 Static Channel Request Priority

Priority	GCR.CRPRIO			
	00_B	01_B	10_B	11_B
High	Channel 0	Channel 1	Channel 0	Channel 0
Low	Channel 1	Channel 0	Channel 1	Channel 1

24.1.5.3 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 two 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.

The dynamic priority assignment feature ensures that two channels can be sampled in equidistant intervals.

24.1.5.4 Clock Generation

As shown in **Figure 24-1**, 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. Clock f_{FADC} is used inside the FADC kernel as the clock for the Channel Timer and the complete analog part.

The clock control as implemented in the TC1766 is described on **Page 24-56**.

24.1.5.5 Suspend Mode Behavior

When a suspend/idle mode request is generated for the FADC module, 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.

24.1.6 Data Reduction Unit

If one or more channels of the FADC is operating in a fast continuous mode (for example, by using the Channel Timers as request sources with fast conversion data request rates), it can be sometimes difficult or even impossible for a CPU or another bus master to collect all conversion results without the risk of losing conversion data. Therefore, a Data Reduction Unit is implemented in the FADC that operates as a kind of antialiasing 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.

The Data Reduction Unit contains two filter blocks. Each filter block allows selection of its input data source. The input data sources are the conversion result registers of the two A/D converter channel. Both filter blocks can also be concatenated. 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.

In order to achieve a higher resolution of the conversion result by averaging over several single conversions, the filter blocks only accumulates single conversion results.

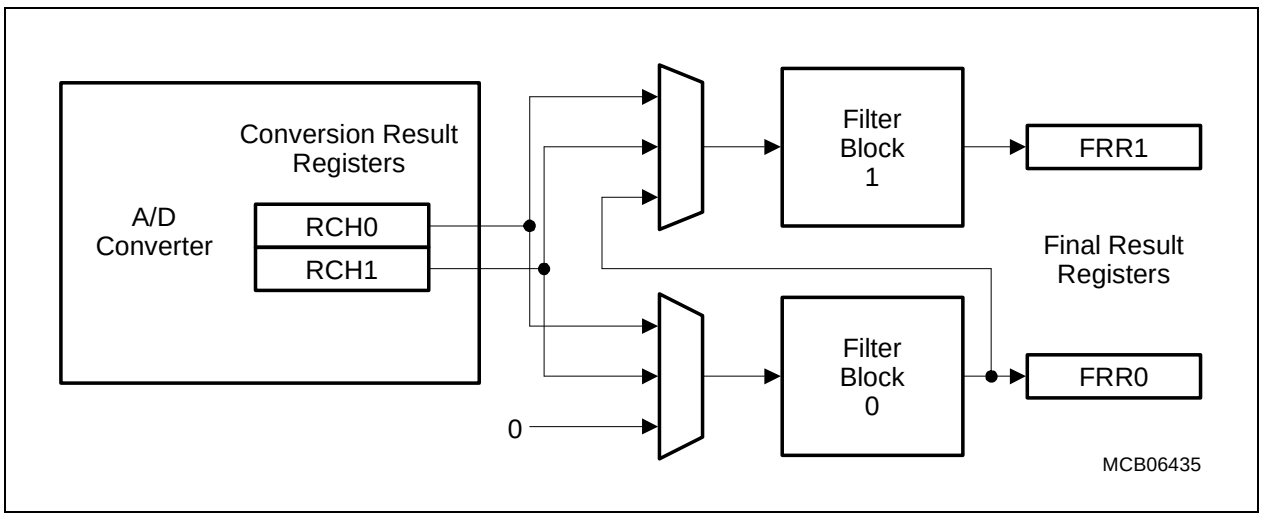


Figure 24-6 FADC Filter Blocks

24.1.6.1 Filter Block Structure

The filter block consists of an adder and several result registers for calculating filter output data from the filter input data. 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 as a kind of

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 are fully transparent and can be read at any time. Please note that only one intermediate result register (IRR11) is available in filter block 1.

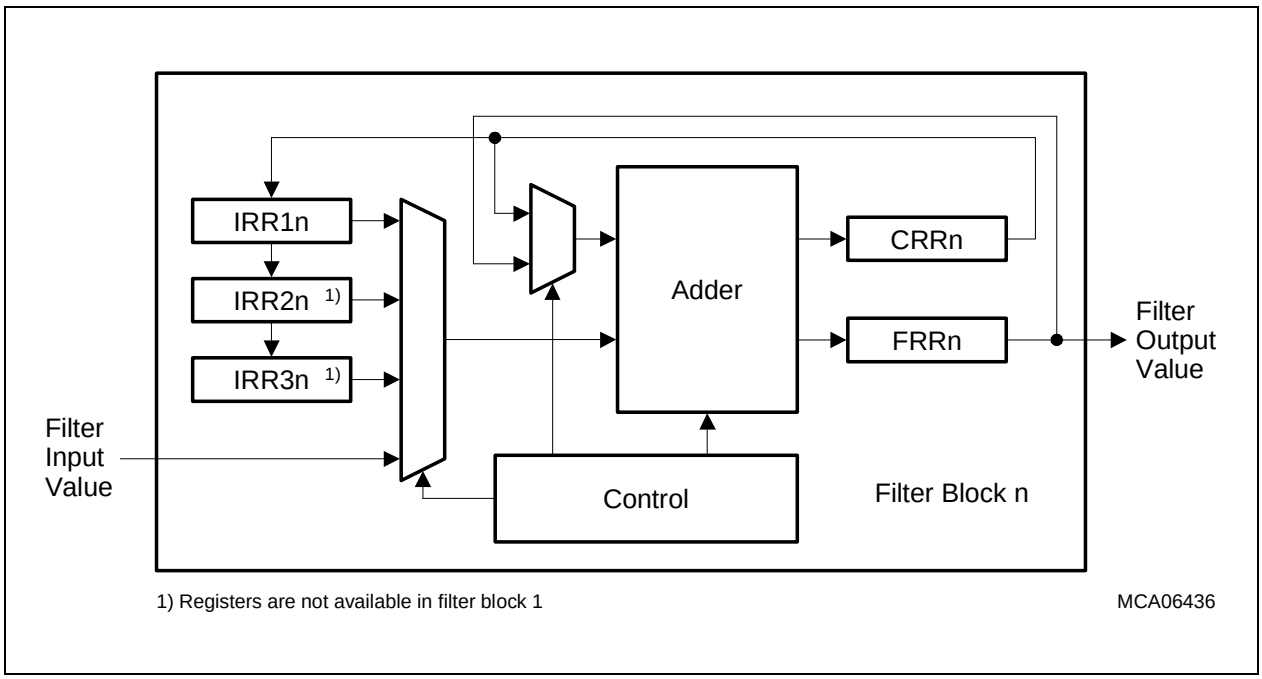


Figure 24-7 Filter Block Structure

24.1.6.2 Filter Block Operation

Figure 24-8 illustrates how filter output values are calculated in a filter block. The following conditions are set for this example:

- A continuous A/D conversion is running on channel x.
- The filter input selection is set to channel x (FCRn.INSEL = $10X_B$ + x).
- The addition length is 4 (FCRn.ADDL = 011_B). This means that four conversion results build one intermediate result.
- The final result is calculated by a moving average over the last four intermediate results (FCRn.MAVL = 11_B).
- The filter registers are at initial state (loaded with 0000_H).

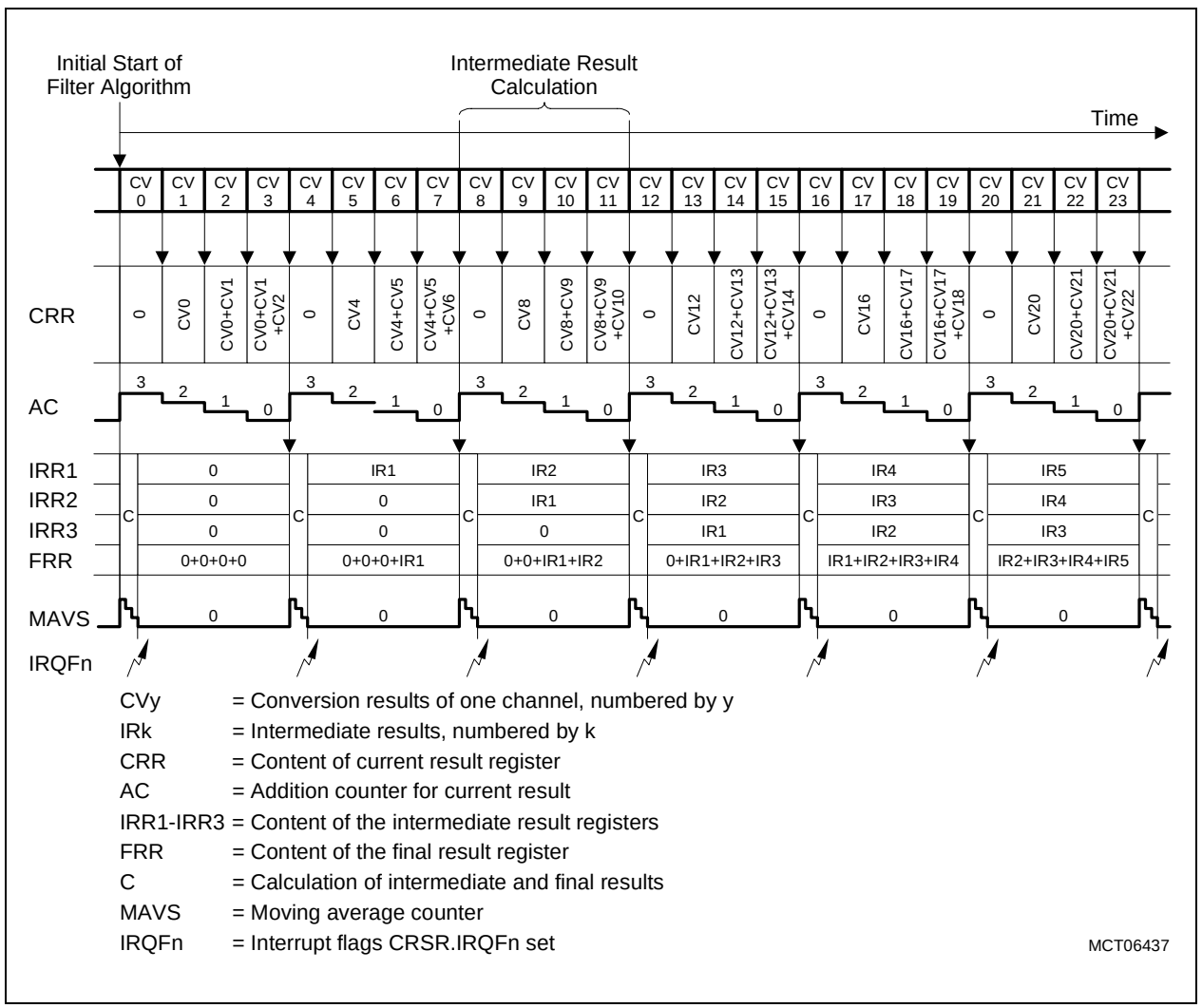


Figure 24-8 Filter Block Algorithm

Intermediate Result Calculation

When the continuous conversion of channel x has been started, the Current Result Register CRRn (n = filter block number) is set to 0. Each following conversion result is added to the content of CRRn until a programmable number of conversion results have been summed up in CRRn. At that point, the calculation of the next final result value is triggered. In the example of **Figure 24-8**, one intermediate result is built by the sum of four conversion results (FCRn.ADDL = 011_B). The state of the current result calculation cycle is indicated by the addition counter AC which is located in register CRRn.

Final Result Calculation

The calculation of a final result is always started after an intermediate result calculation cycle has been finished. This new intermediate result (stored in CRRn) plus the contents of the intermediate registers are added according to:

- Filter 0: FRR0 := CRR0 + IRR10 + IRR20 + IRR30
- Filter 1: FRR1 := CRR1 + IRR11

Therefore, FRRn always contains the sum of maximum four (for filter block 0) or two (for filter block 1) intermediate results. Bit field FCRn.MAVL determines the number of intermediate results that are used for the final result calculation.

After a final result calculation, the old contents of IRR20 are transferred into IRR30 and IRR10 are transferred into IRR20 (for filter block 0). In filter block 1, the contents of CRR1 are stored in IRR11. The old contents of IRR30 or IRR11 are lost. Finally, the value of CRRn is transferred as intermediate result into the intermediate result register IRR1n. Thereafter, the CRRn register is set to 0 for the next current result calculation cycle.

Each update of a result register FRRn with a new final result value generates a filter block n service request. During a final result calculation phase (phase C in **Figure 24-8**), the contents of the FRRn registers change. Therefore, it is recommended to read a final result from the FRRn registers immediately (for example, by a DMA operation) after a corresponding interrupt request flag CRSR.IRQFn has been set.

Filter Control Parameters

The Filter n Control Register FCRn contains several parameters that determine the operation of filter n:

- **FCRn.INSEL:**
  This parameter selects the channel(s) whose conversion results are used as filter n input. A single channel, all two channels, or the final result of filter 0 (for filter 1 concatenation) can be selected. The filter n operation can also be stopped via FCRn.INSEL.
- **FCRn.ADDL:**
  This parameter determines how many conversion results are added in filter n to calculate one intermediate result. FCRn.ADDL is loaded into CRRn.AC at the start of

a filter sequence and after each intermediate result storage. CRRn.AC is decremented after each conversion result addition and indicates how many additions are still to be executed until the next intermediate result is stored. Values for one up to eight conversions can be selected for FCRn.ADDL.

- **FCRn.MAVL:**
  This parameter determines the number of intermediate result registers that are used for a final result calculation. FCRn.MAVL is loaded into CRRn.MAVS at the start of a filter sequence. Values for none up to three intermediate result registers can be selected for FCRn.MAVL.

Initial State

In order to start a filter algorithm for filter block n, the following actions must be executed:

1. Program bit field FCRn.ADDL with FCRn.INSEL = 000_B (filter disabled)
2. Select filter n input source and operation by writing FCRn.INSEL with the appropriate value
3. Reset filter block n by writing GCR.RSTFn = 1
4. Start a continuous conversion

24.1.6.3 Filter Concatenation

As shown in **Figure 24-6**, it is possible to concatenate the two filters by using the result value of filter 0 as input value for filter 1. **Figure 24-9** shows an example for the filter concatenation.

Filter 0 operates with following parameters:

- Intermediate results are calculated from four filter input values (conversion results)
- A final result is calculated by four intermediate results

Filter 1 operates with following parameters:

- Concatenation enabled (FCR1.INSEL = 010_B)
- Intermediate results are calculated from three filter 1 input values
- A final result is calculated by two intermediate results

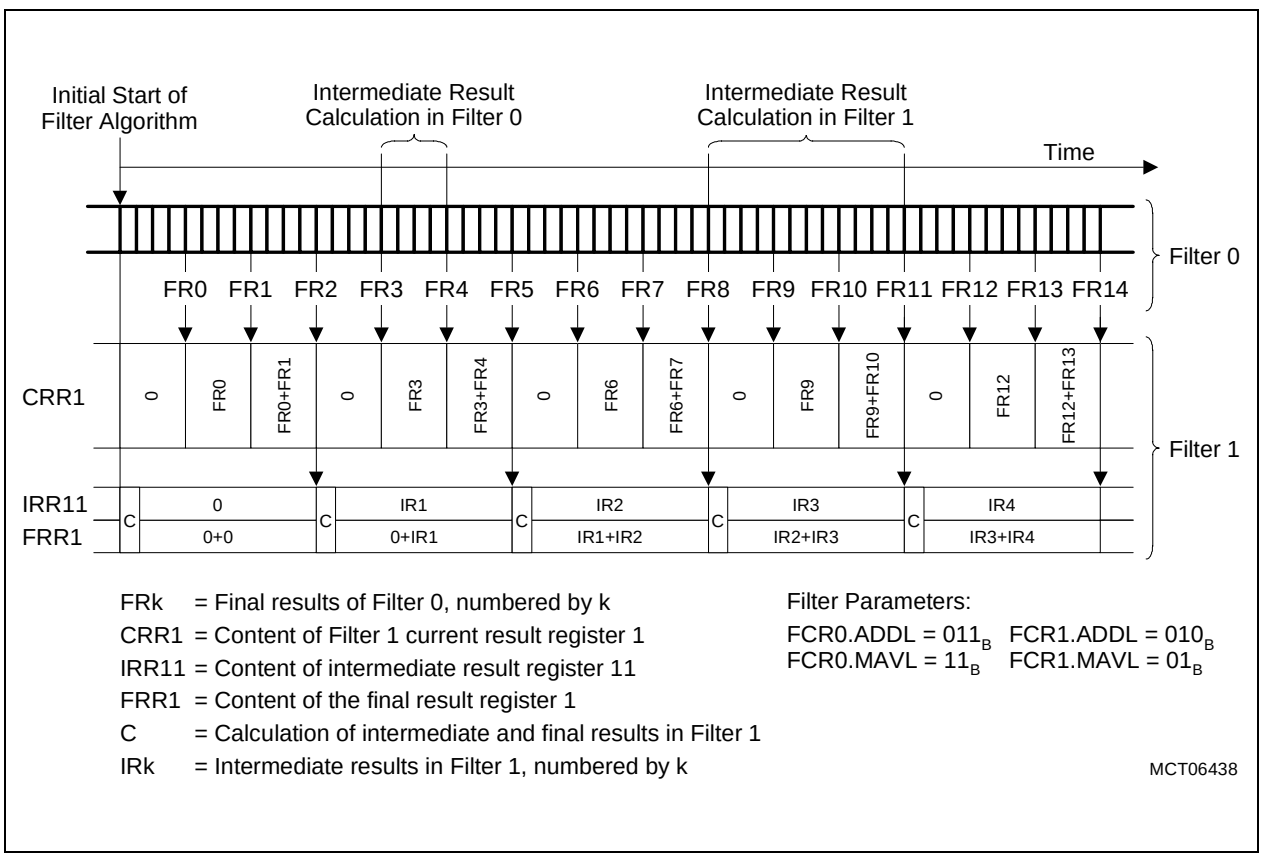


Figure 24-9 Filter Concatenation Example

Filter 0 operates with the same parameters as in the example shown in **Figure 24-8**. A Filter 1 input value is given by each final result of filter 0. Filter 1 can store only one intermediate result in its IRR11 register.

24.1.6.4 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.

An FADC conversion result always has a width of ten bits. A maximum of eight conversion results can be added in the current result registers for an intermediate result. This results in 10 + 3 = 13 bit width for the current result register and for the intermediate result registers. The final result is built by the addition of maximum four current/intermediate result registers. Therefore, the width of the final result register is 13 + 2 = 15. These values for the result widths are valid for filter 0.

When both filters are concatenated, the width of FRR0 (15-bit) must be taken into account when the filter 1 result register width is calculated. Filter 1 also allows eight input values (filter 0 final results) to be added in its current result register for an intermediate result. This results in 15 + 3 = 18 bits width for the current result register and for the intermediate result registers of filter 1. The final result is built in filter 1 by the addition of maximum two current/intermediate result values. Therefore, the width of the filter 1 final result register is 18 + 1 = 19. Caused by the design, the two missing intermediate result registers of filter 1 must be considered and the width of the filter 1 final result register is additionally increased to 19 + 1 = 20.

Table 24-5 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

24.1.7 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 all 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.

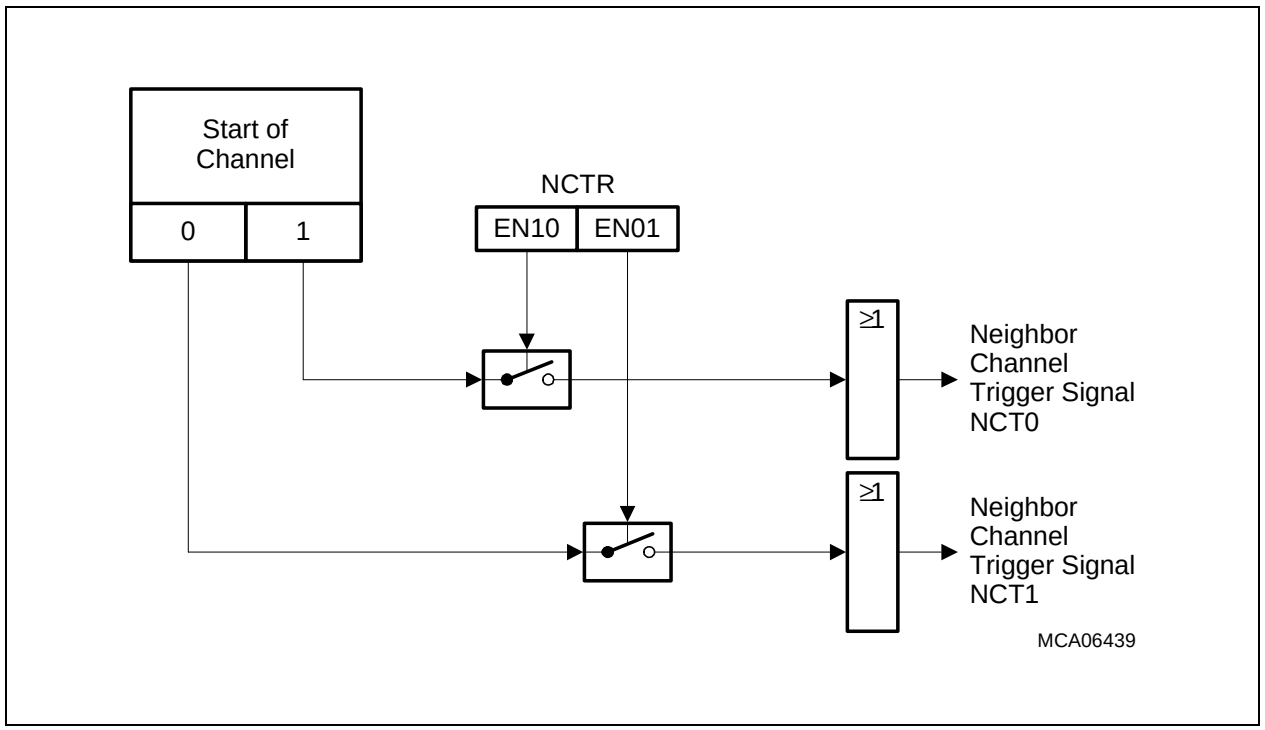


Figure 24-10 Neighbor Channel Trigger

24.1.8 Offset and Gain Calibration

The offset and gain 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 channel can be used without restrictions.

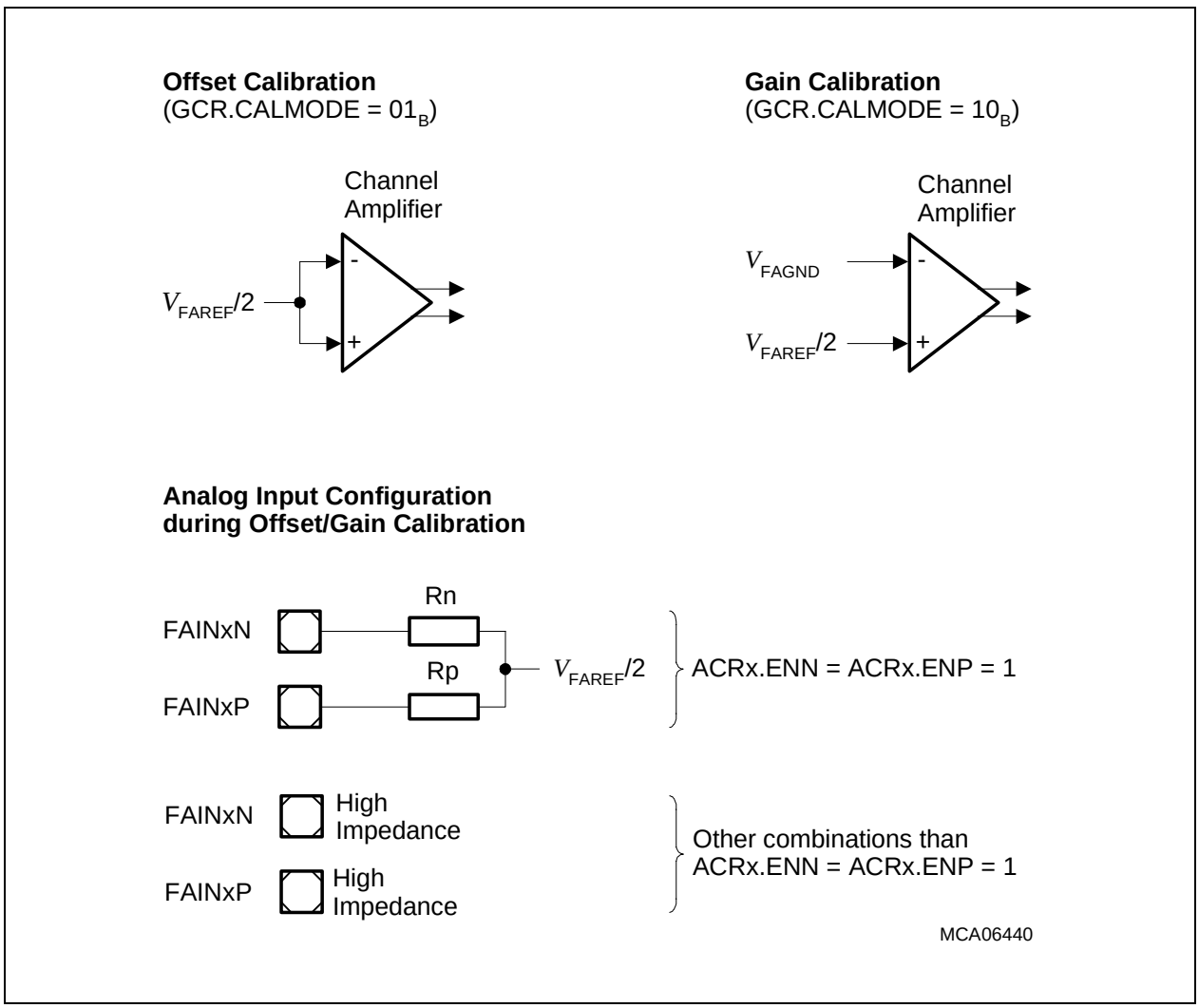


Figure 24-11 Analog Input and Channel Amplifier Configuration at Calibration

Figure 24-11 shows the channel amplifier configuration as well as the analog input pin configurations that are selected during offset and gain 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.

24.1.8.1 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.

After an offset calibration has been finished for a channel, its gain can also be calibrated.

24.1.8.2 Gain Calibration

When gain calibration is selected (GCR.CALMODE = 10_B), the gain can be adjusted by software to cover the complete input voltage range. The procedure is similar to the one for the offset calibration.

24.1.9 Interrupt Generation

A flexible service request control structure is implemented in the FADC. The FADC provides two channel conversion request sources and two 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.

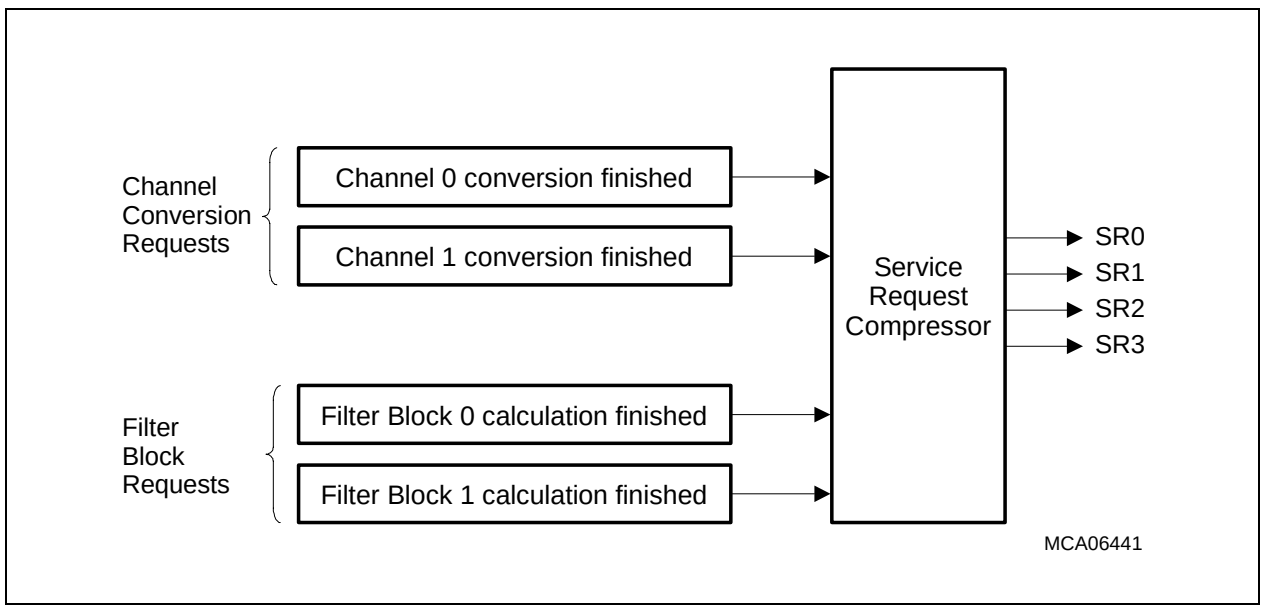


Figure 24-12 Service Request Configuration

All six service requests are controlled by an identical control logic. This control logic as shown in **Figure 24-13** provides the following functionality:

- Service Request Flag
- Set/Clear Request Flag Control Bits
- Service Request Enable Bit
- Service Request Node Pointer

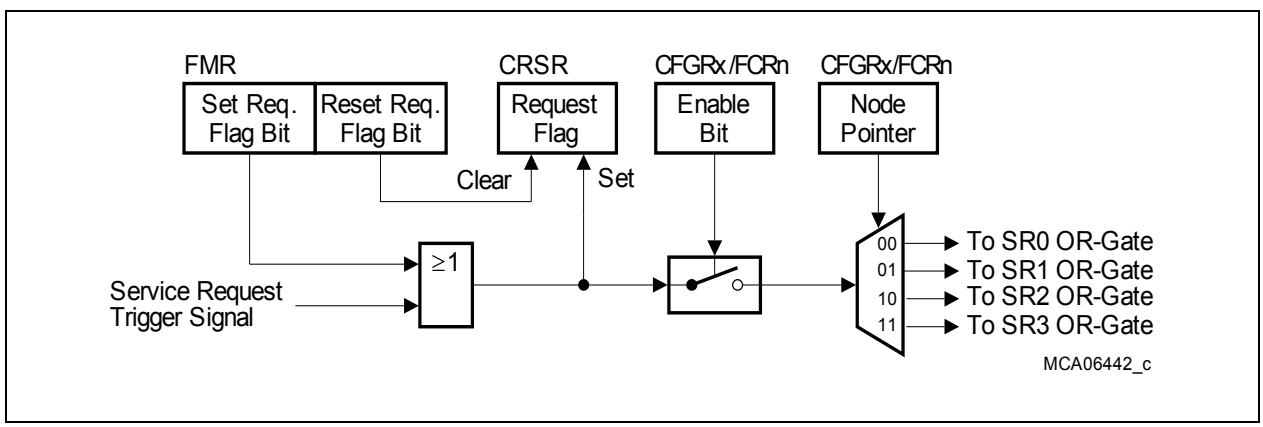


Figure 24-13 Service Request Control Logic

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 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 24-6 lists the four service request sources of the A/D Converter module with its related control and status flags/bits.

Table 24-6 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 = 0-1)	CRSR.IRQx	CFGRx.IEN	FMR.SIRQx / FMR.RIRQx	CFGRx.INP
Filter Block n Request (n = 0, 10)	CRSR.IRQFn	FCRn.IEN	FMR.SIRQFn / FMR.RIRQFn	FCRn.INP

In the service request compressor logic shown in Figure 24-14, 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 only be assigned to one of the four service request outputs but one service request output can be used by multiple service request events.

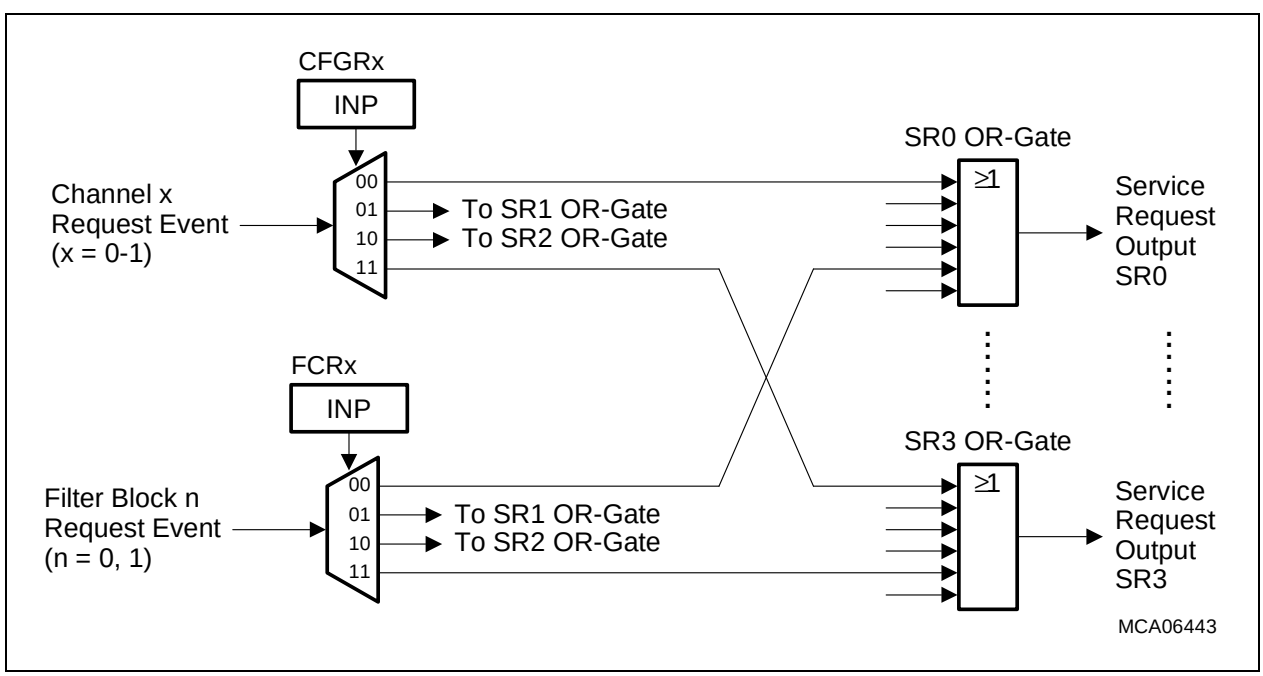


Figure 24-14 Service Request Compressor Logic

Depending on the implementation of the FADC Module in a specific microcontroller, 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 TC1766 specific request output connections are described in **Figure 24-16** and on **Page 24-64**.

24.2 FADC Kernel Registers

This section describes the kernel registers of the FADC module. The complete and detailed address maps of the FADC module is described in **Table 16-20** on **Page 16-52** of the TC1766 User's Manual System Units part (Volume 1).

All FADC kernel register names described in this section are referenced in other parts of the TC1766 User's Manual by the module name prefix "FADC_".

FADC Kernel Register Overview

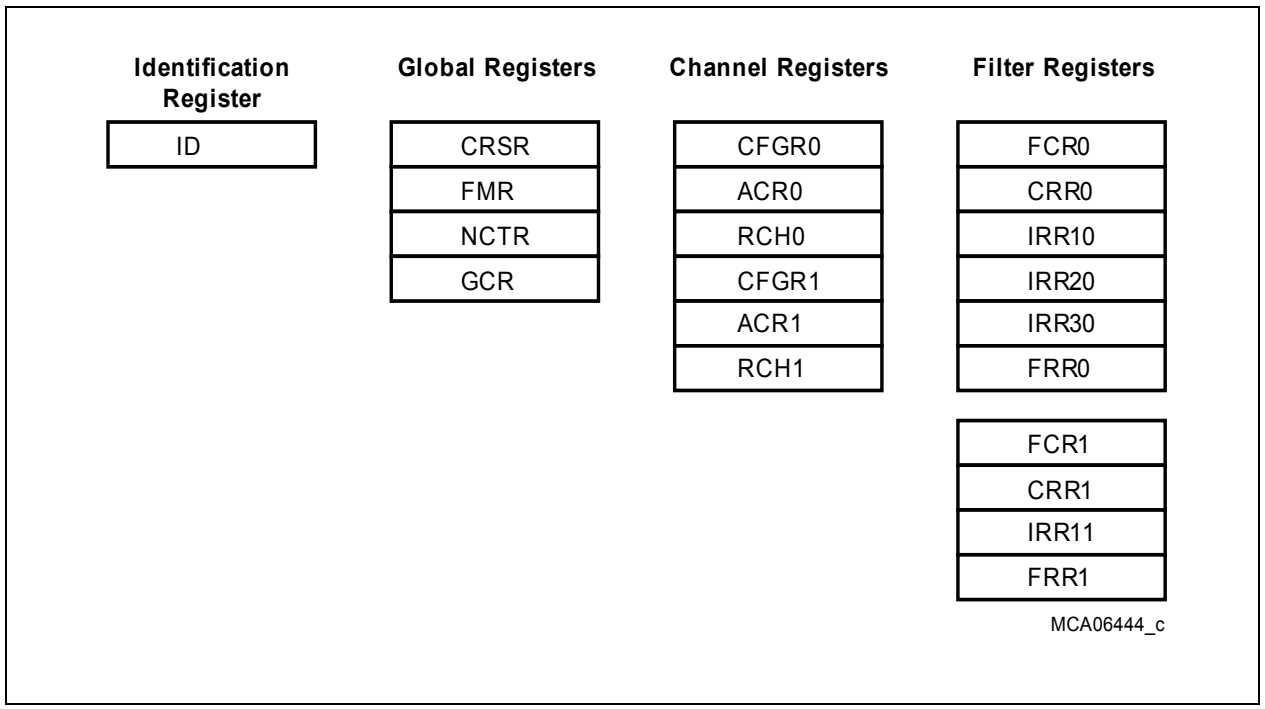


Figure 24-15 FADC Kernel Registers

In the TC1766, the registers of the FADC module are located in the following address range.

Table 24-7 Registers Address Space - FADC Module

Module	Base Address	End Address	Note
FADC	$F010\ 0300_H$	$F010\ 03FF_H$	-

Table 24-8 Registers Overview - FADC Kernel Registers

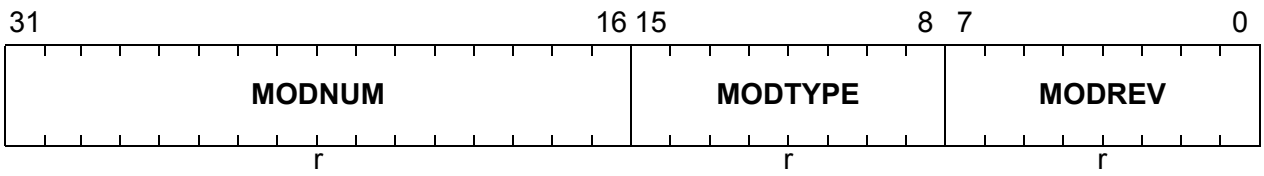
Register Short Name	Register Long Name	Offset Address	Description see
Global Registers			
ID	Module Identification Register	08_H	Page 24-30
CRSR	Conversion Request Status Register	10_H	Page 24-31
FMR	Flag Modification Register	14_H	Page 24-33
NCTR	Neighbor Channel Trigger Register	18_H	Page 24-35
GCR	Global Control Register	$1C_H$	Page 24-36
Channel Registers			
CFGRx	Channel x Configuration Register (x = 0-1)	$20_H + (x \times 4)$	Page 24-40
ACRx	Channel x Analog Control Register (x = 0-1)	$30_H + (x \times 4)$	Page 24-44
RCHx	Channel x Conversion Result Register (x = 0-1)	$40_H + (x \times 4)$	Page 24-46
Filter Registers			
FCRn	Filter n Control Register (n = 0-1)	$60_H + (n \times 20_H)$	Page 24-47
CRRn	Filter n Current Result Register (n = 0-1)	$64_H + (n \times 20_H)$	Page 24-50 Page 24-51
IRR1n	Filter n Intermediate Result Register 1 (n = 0-1)	$68_H + (n \times 20_H)$	Page 24-53
IRR2n	Filter n Intermediate Result Register 2 (n = 0)	$6C_H + (n \times 20_H)$	
IRR3n	Filter n Intermediate Result Register 3 (n = 0)	$70_H + (n \times 20_H)$	
FRRn	Filter n Final Result Register (n = 0-1)	$74_H + (n \times 20_H)$	Page 24-54

24.2.1 FADC Module Identification Register

The FADC Module Identification Register ID contains read-only information about the FADC module version.

ID
Module Identification Register

(08_H) **Reset Value: 0027 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 FADC: 0027_H

24.2.2 Global Registers

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 (10_H) **Reset Value: $0000\ 0000_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0										IRQF1	IRQF0	0		IRQ1	IRQ0
r										rh	rh	r		rh	rh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						BSY1	BSY0	0						CRF1	CRF0
r						rh	rh	r						rh	rh

Field	Bits	Type	Description
CRFx (x = 0-1)	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/reset by software via bits FMR.RCRFx and FMR.SCRFx (see Page 24-33). If a set and a reset condition for CRFx occur simultaneously (generated by hardware and/or software), the reset condition always wins.
BSYx (x = 0-1)	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.

Field	Bits	Type	Description
IRQx (x = 0-1)	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/reset by software via bits FMR.SIRQx and FMR.RIRQx (see **Page 24-33**).
IRQFn (n = 0-1)	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/reset by software via bits FMR.SIRQFn and FMR.RIRQFn (see **Page 24-33**).
0	[7:2], [15:10], [19:18], [31:22]	r	**Reserved** Read as 0; should be written with 0.

The bits of the Flag Modification Register FMR allow the flags of the conversion request status register to be set/reset by software.

FMR
Flag Modification Register **(14_H)** **Reset Value: 0000 0000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0		SIRQF1	SIRQF0	0		SIRQ1	SIRQ0	0		RIRQF1	RIRQF0	0		RIRQ1	RIRQ0
r		w	w	r		w	w	r		w	w	r		w	w

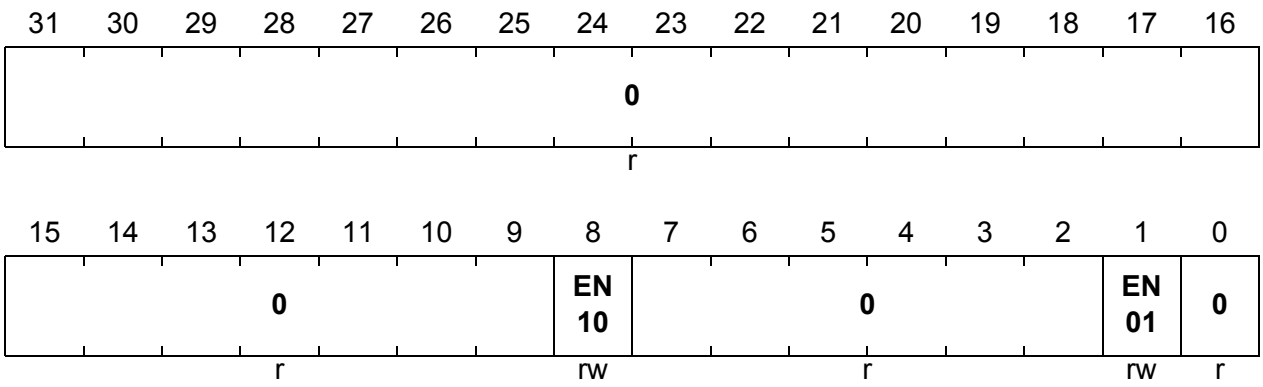
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						SCRF1	SCRF0	0						RCRF1	RCRF0
r						w	w	r						w	w

Field	Bits	Type	Description
RCRFx (x = 0-1)	x	w	**Reset 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 (also if bit SCRFx is written simultaneously with 1)
SCRFx (x = 0-1)	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 = 0-1)	16 + x	w	**Reset 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
RIRQFn (n = 0-1)	20 + n	w	**Reset 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

Field	Bits	Type	Description
SIRQx (x = 0-1)	24 + x	w	**Set Interrupt Request Flag** This bit allows bit CRSR.IRQx to be set by software (if bit RIRQx is written simultaneously with 0). 0_B No operation 1_B Bit CRSR.IRQx is set. An interrupt is generated if CFGRx.IEN = 1.
SIRQFn (n = 0-1)	28 + n	w	**Set Interrupt Request Flag for Filter n** This bit allows bit CRSR.IRQFn to be set by software (if bit RIRQFn is written simultaneously with 0). 0_B No operation 1_B Bit CRSR.IRQFn is set. An interrupt is generated if FCRn.IEN = 1.
0	[7:2], [15:10], [19:18], [23:22], [27:26], [31:30]	r	**Reserved** Read as 0; should be written with 0.

The Neighbor Channel Trigger Register NCTR contains the enable bits for the neighbor channel trigger signal (NCTx) generation (see Page 24-22).

NCTR
Neighbor Channel Trigger Register (18_H) Reset Value: $0000\ 0000_H$



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.
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.
0	0, [7:2], [31:9]	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.

The Global Control Register GCR contains bits used to clear the Channel Timers, the filters and to control global FADC settings.

GCR
Global Control Register **($1C_H$)** **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					RSTF1	RSTF0	RCD	0						RCT1	RCT0
r					w	w	w	r						w	w

Field	Bits	Type	Description
RCTx (x = 0-1)	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 = 0-1)	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 which 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 common amplifier. 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. (default) 1_B The analog part is enabled.
CALMODE	[25:24]	rw	**Calibration Mode** This bit field enables the calibration for offset and gain 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 The analog channel selected by CALCH is in gain calibration mode. The other channels are are in normal mode. 11_B Reserved

Field	Bits	Type	Description
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.
0	[7:2], [15:11], [23:22], [31:28]	r	**Reserved** Read as 0; should be written with 0.

24.2.3 Channel 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 = 0-1)
Channel x Configuration Register (20_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 GS0 selected 001_B Gating source input signal GS1 selected 010_B Gating source input signal GS2 selected 011_B Gating source input signal GS3 selected 100_B Gating source input signal GS4 selected 101_B Gating source input signal GS5 selected 110_B Gating source input signal GS6 selected 111_B Gating source input signal GS7 selected
TSEL	[5:3]	rw	**Trigger Selection** This bit field selects the trigger source input signal for channel x. 000_B Trigger source input signal TS0 selected 001_B Trigger source input signal TS1 selected 010_B Trigger source input signal TS2 selected 011_B Trigger source input signal TS3 selected 100_B Trigger source input signal TS4 selected 101_B Trigger source input signal TS5 selected 110_B Trigger source input signal TS6 selected 111_B Trigger source input signal TS7 selected

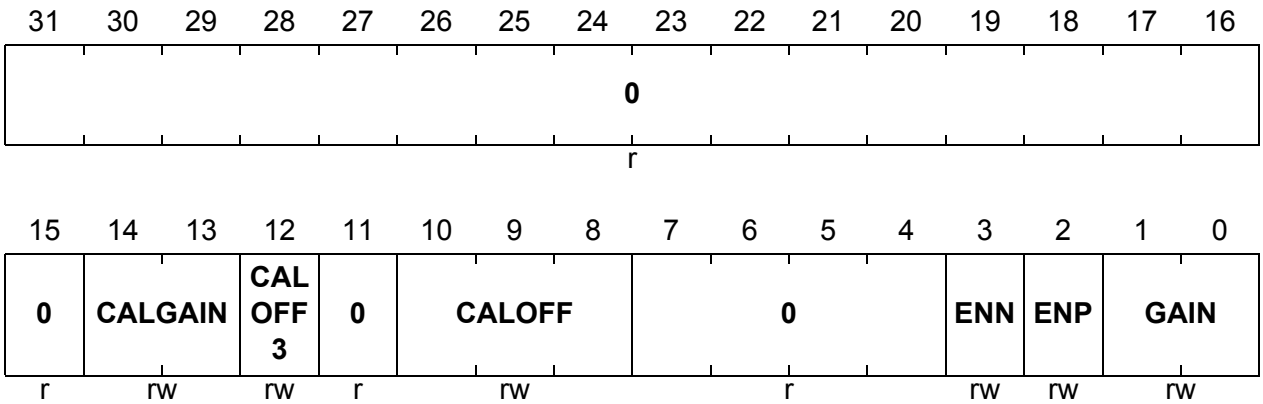
Field	Bits	Type	Description
GM	[7:6]	rw	**Gating Mode** This bit field determines the functionality of the gating (enable) signal. It determines whether and under which condition the generation of conversion requests by trigger signals is possible. 00_B Conversion requests are disabled and the Channel 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.
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).

Field	Bits	Type	Description
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.
CTREL	[23:16]	rw	**Channel Timer Reload Value** This bit field determines the reload value of the Channel Timer CHTIMx. The divider factor for the channel x timer is given by: CTREL + 1, if $2 \leq f_{CLC} / f_{CTx}$ CTREL + 2, if $f_{CLC} / f_{CTx} = 1$ If CTREL = 0, no trigger event is generated. See also description on **Page 24-12**.
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.

Field	Bits	Type	Description
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.

The Channel x Analog Control Register ACRx contains the bits that control the analog input stage.

ACRx (x = 0-1)
Channel x Analog Control Register (30_H+x*4_H) **Reset Value: 0000 0000_H**



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.

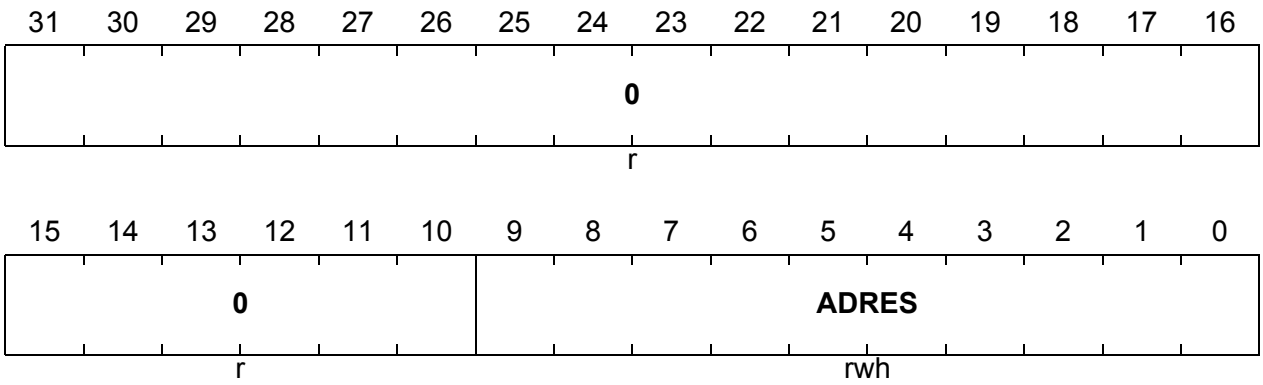
Field	Bits	Type	Description
ENN	3	rw	**Enable Negative Input** This bit enables the voltage measurement on the FAINxN analog input. 0_B Analog input FAINxN is high-impedance. The lower half of the measuring range is not available. 1_B Analog input FAINxN line is connected to the channel amplifier. The 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	
CALGAIN	[14:13]	rw	**Calibrate Gain** This bit field determines the value applied for the gain calibration for channel x.
0	[7:4], 1, [31:15]	r	**Reserved** Read as 0. Should be written with 0.

Note: If ENN = 0 and ENP = 0, the channel amplifier is in power-down mode. The conversion result is not valid in this case.

The Channel x Conversion Result Register RCHx contains the conversion result ADRES of channel x.

RCHx (x = 0-1)
Channel x Conversion Result Register

(40_H+x*4_H) **Reset Value: 0000 0000_H**

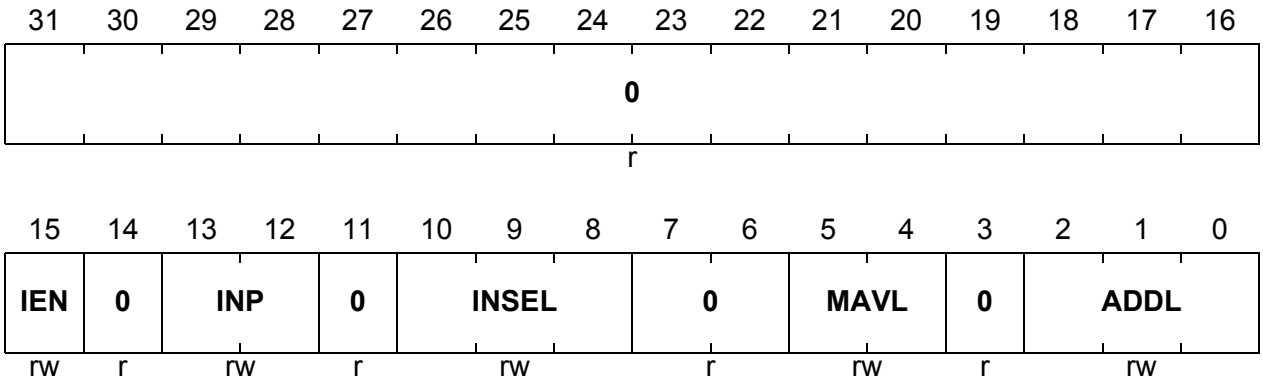


Field	Bits	Type	Description
ADRES	[9:0]	rwh	**AD Conversion Result** This bit field contains the conversion result of channel x. ADRES can only be overwritten by software if GCR.RESWEN = 1 (see also register GCR description on **Page 24-36**).
0	[31:10]	r	**Reserved** Read as 0. Should be written with 0.

24.2.4 Filter Registers

Filter blocks 0 and 1 are controlled by bits in the Filter n Control Registers FCRn.

FCRn (n = 0-1)
Filter n Control Register **(60_H+n*20_H)** **Reset Value: 0000 0000_H**



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.

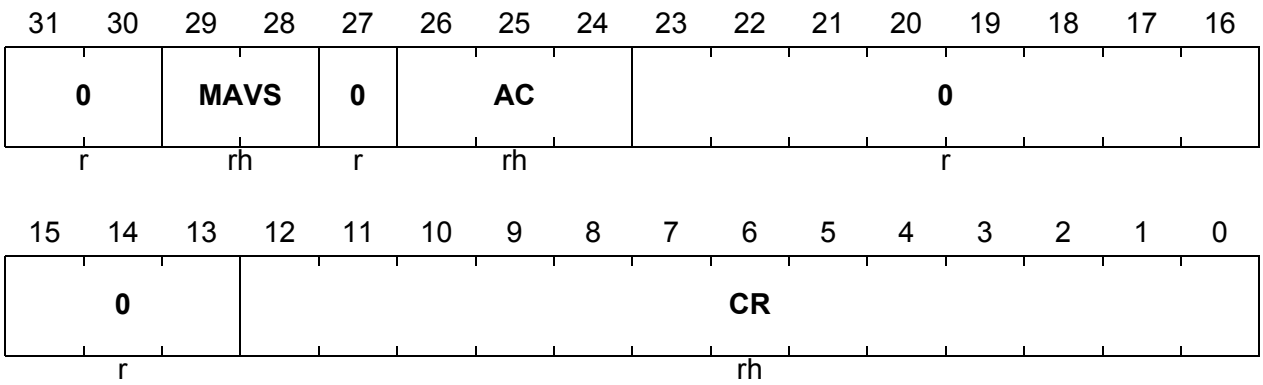
Field	Bits	Type	Description
MAVL	[5:4]	rw	**Moving Average Length** This bit field determines the number of intermediate results that are added up for a final result. 00_B No moving average is selected. Each intermediate result is considered as final result value: FRRn.FR = CRRn.CR 01_B A moving average 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 block 1.
INSEL	[10:8]	rw	**Input Selection** This bit field enables the filter block and determines which input value is taken for filter block n. 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 each 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. 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 Reserved; 0 is used as input value. 111_B Reserved; 0 is used as input value.

Field	Bits	Type	Description
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
0	3, [7:6], 11, 14, [31:16]	r	**Reserved** Read as 0.

Note: If the length of the moving average is reduced, the execution time (number of clock cycles) is also reduced. In any case, the filter calculation and execution time is much smaller than a conversion time.

The Current Result Registers CRRn stores the current result of filter n. Further, status information of filter block n can be read from CRRn.

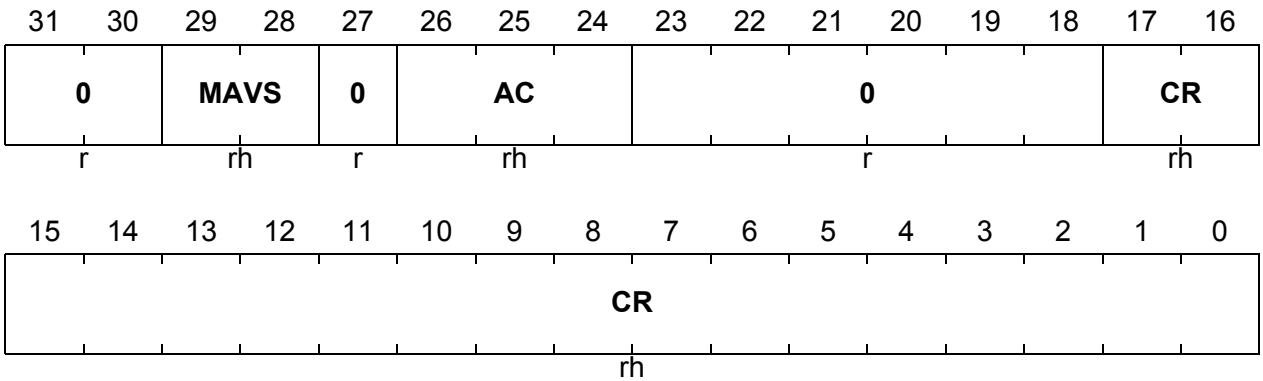
CRR0
Filter 0 Current Result Register (64_H) Reset Value: 0000 0000$_H$



Field	Bits	Type	Description
CR	[12:0]	rh	**Current Result** This bit field ([12:0] for filter 0, [17:0] for filter 1) 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. CR is cleared when writing GCR.RSTFn = 1.
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 only when the corresponding interrupt request flag CRSR.IRQFn is set. MAVS is cleared when writing GCR.RSTFn = 1.

Field	Bits	Type	Description
0	[23:13], 27, [31:30]	r	**Reserved** Read as 0.

CRR1
Filter 1 Current Result Register (84_H) Reset Value: 0000 0000_H



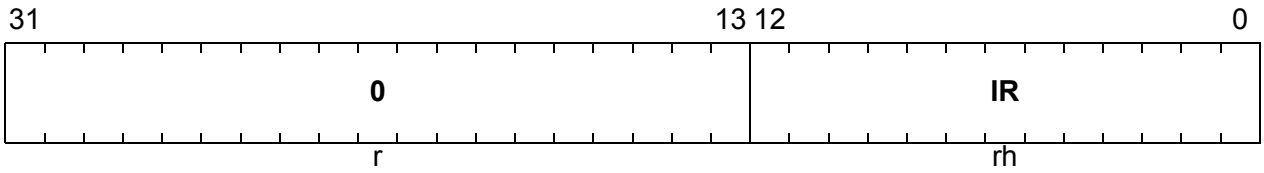
Field	Bits	Type	Description
CR	[17:0]	rh	**Current Result** This bit field ([12:0] for filter 0, [17:0] for filter 1) contains the right-aligned current result value of filter 0. CR is cleared when writing GCR.RSTFn = 1.
AC	[26:24]	rh	**Addition Count** This bit field indicates the number of additions 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. CR is cleared 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 only when the corresponding interrupt request flag CRSR.IRQFn is set. MAVS is cleared when writing GCR.RSTFn = 1.
0	[23:18], 27, [31:30]	r	**Reserved** Read as 0.

The Intermediate Result Registers IRRxn hold the intermediate results x of filter n.

IRRx0 (x = 1-3)
Filter 0 Intermediate Result Register x

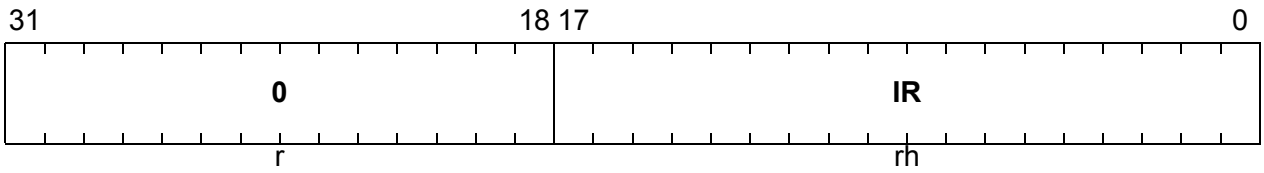
(64_H+x*4_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
IR	[12:0]	rh	**Intermediate Result** This bit field contains the right-aligned intermediate result of filter 0. IR is cleared when writing GCR.RSTFn = 1.
0	[31:13]	r	**Reserved** Read as 0.

IRR11
Filter 1 Intermediate Result Register 1

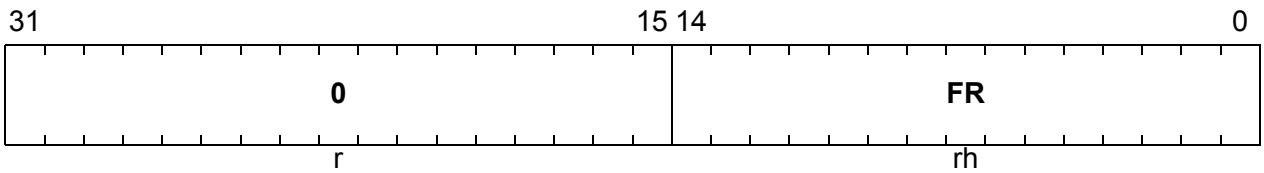
(88_H) **Reset Value: 0000 0000_H**



Field	Bits	Type	Description
IR	[17:0]	rh	**Intermediate Result** This bit field contains the right-aligned intermediate result of filter 1. IR is cleared when writing GCR.RSTFn = 1.
0	[31:18]	r	**Reserved** Read as 0.

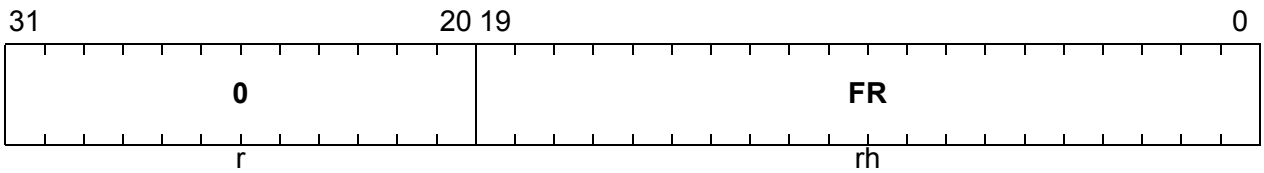
The Final Result Registers for filters 0 and 1 hold the final result of the filter operations (additions).

FRR0
Filter 0 Final Result Register (74_H) Reset Value: 0000 0000_H



Field	Bits	Type	Description
FR	[14:0]	rh	**Final Result** This bit field contains the right-aligned 15-bit final result of filter 0. FR is cleared when writing GCR.RSTFn = 1.
0	[31:15]	r	**Reserved** Read as 0.

FRR1
Filter 1 Final Result Register (94_H) Reset Value: 0000 0000_H



Field	Bits	Type	Description
FR	[19:0]	rh	**Final Result** This bit field contains the right-aligned 20-bit final result of filter 1. FR is cleared when writing GCR.RSTFn = 1.
0	[31:20]	r	**Reserved** Read as 0.

24.3 Implementation of FADC

This section describes the FADC module related external functions, such as port connections, interrupt and service request control, DMA connections, address decoding and clock control.

24.3.1 Interfaces of the FADC Module

Figure 24-16 shows the TC1766 specific implementation details and interconnections of the FADC module.

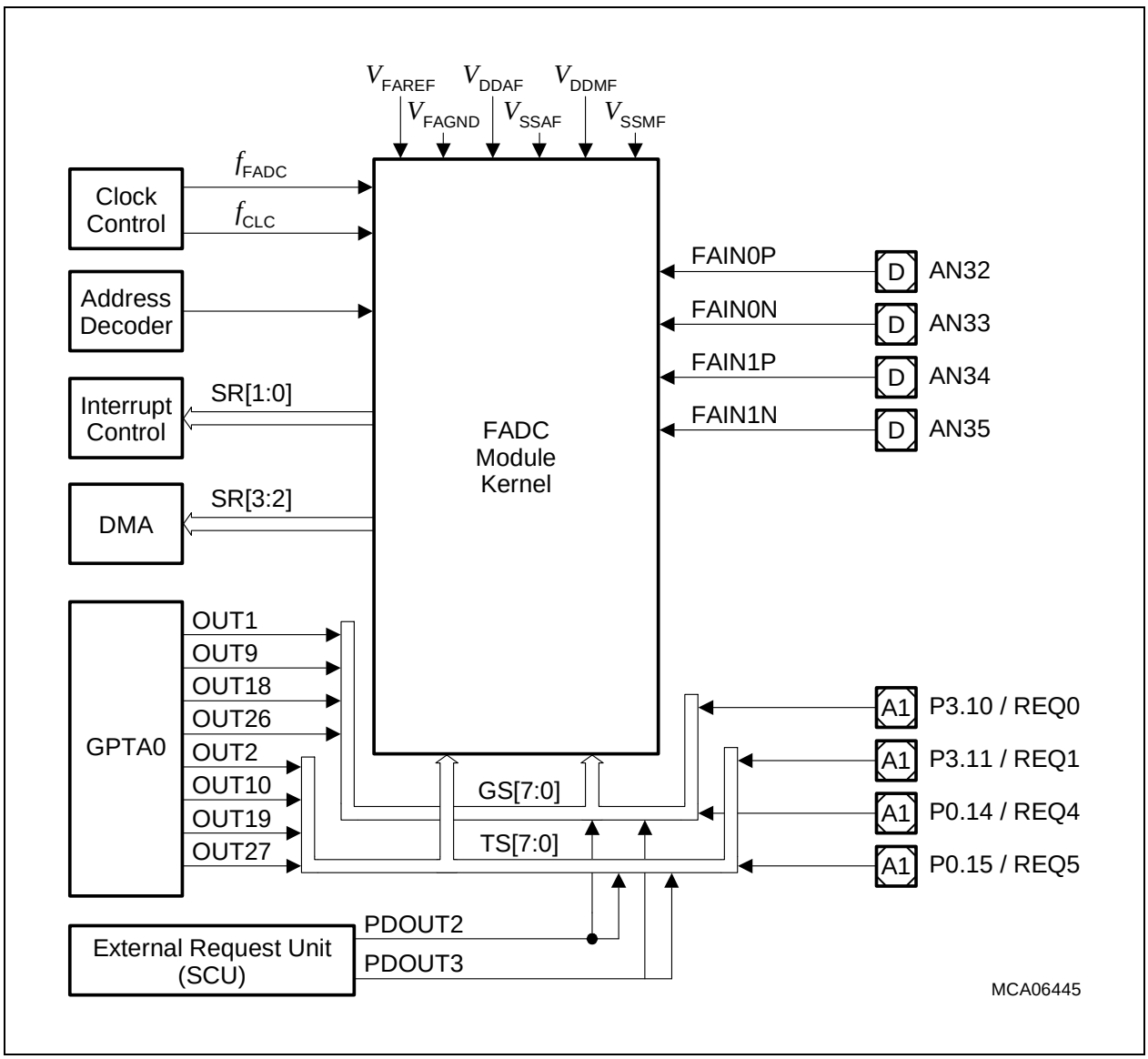


Figure 24-16 FADC Module Implementation and Interconnections

The FADC module is supplied with a clock control, address decoding, and interrupt control logic. Two of the four service request outputs are each connected to an interrupt

node while the other two service request outputs are wired to request inputs of the DMA controller. The 2 × 8 gating and trigger source input lines are connected to four request input pins, eight outputs of the GPTA0 module, and two outputs of the external request unit. The four analog inputs are wired to analog inputs of the TC1766.

24.3.2 FADC Module Related External Registers

Figure 24-17 shows the implementation specific FADC external registers that can be used for FADC programming.

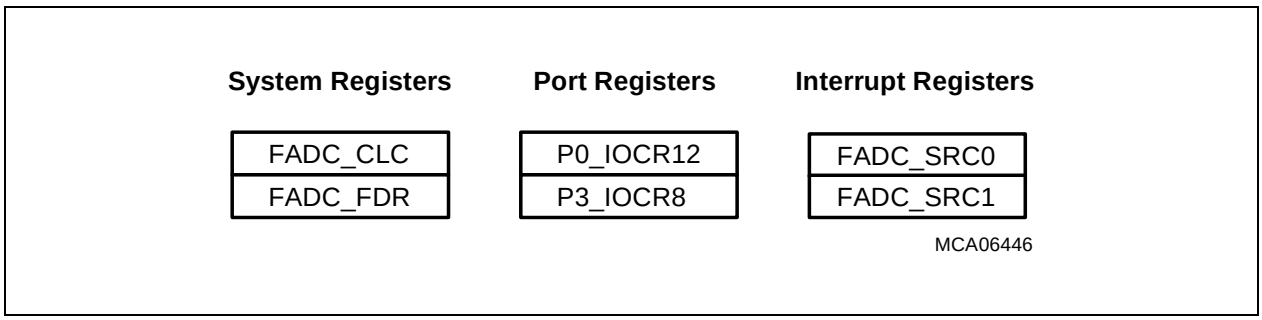


Figure 24-17 FADC Implementation-specific Special Function Registers

24.3.3 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}. The clock control register FADC_CLC makes it possible to enable/disable f_{CLC} under certain conditions.
- f_{FADC}
  This clock is the module clock that is used in the FADC as the clock for the channel timer and the analog part. It also controls the conversion timing (see also **Page 24-8**). The fractional divider registers FADC_FDR controls the frequency of f_{FADC} and allows it to be enabled/disabled independently of f_{CLC}.

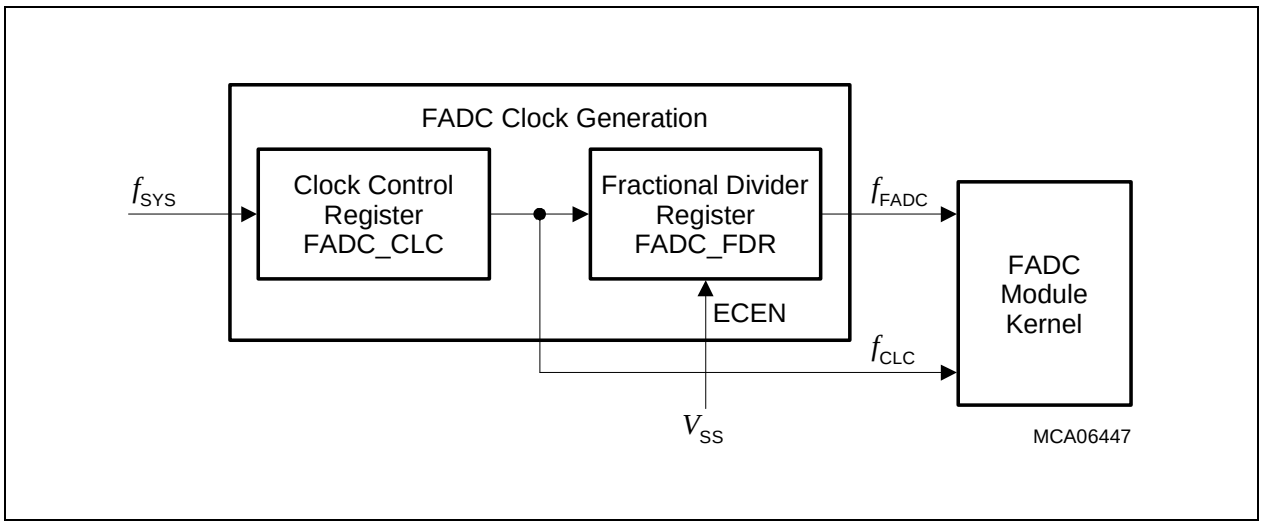


Figure 24-18 FADC Clock Generation

The following formulas define the frequency of f_{FADC}:

$$f_{FADC} = f_{SYS} \times \frac{1}{n} \quad \text{with n = 1024 - FDR.STEP} \tag{24.3}$$

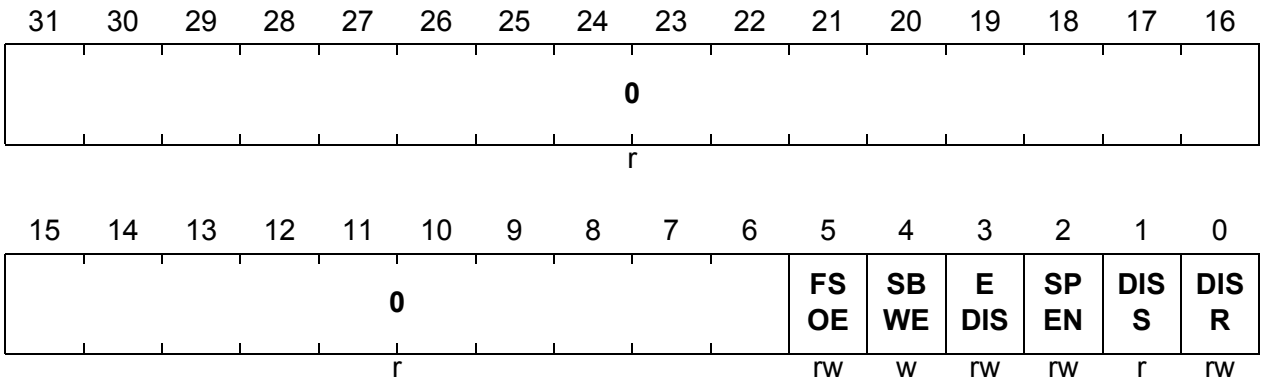
$$f_{FADC} = f_{SYS} \times \frac{n}{1024} \quad \text{with n = 0-1023} \tag{24.4}$$

Equation (24.3) is valid for FADC_FDR.DM = 01_B (normal divider mode).
Equation (24.4) is valid for FADC_FDR.DM = 10_B (fractional divider mode).

24.3.3.1 Clock Control Register

The Clock Control Register allows the programmer to control (enable/disable) the clock signal f_{CLC} under certain conditions. The register table below shows the clock control register functionality as it is implemented for the FADC module. After a reset operation, the FADC module is disabled and its module clock signal f_{CLC} is switched off.

FADC_CLC
FADC Clock Control Register (00_H) Reset Value: $0000\ 0003_H$



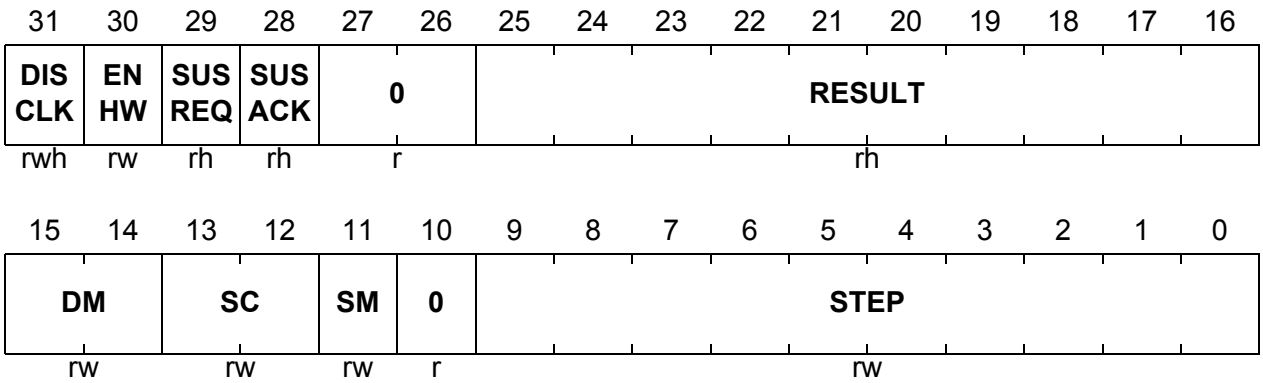
Field	Bits	Type	Description
DISR	0	rw	**Module Disable Request Bit** Used for enable/disable control of the module.
DISS	1	r	**Module Disable Status Bit** Bit indicates the current status of the module.
SPEN	2	rw	**Module Suspend Enable for OCDS** Used to enable the suspend mode.
EDIS	3	rw	**Sleep Mode 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 for fast clock switch off in Suspend Mode.
0	[31:6]	r	**Reserved** Read as 0. Should be written with 0.

Note: Additional details on the clock control register functionality are described in section ***"Clock Control Register CLC" on Page 3-24*** *of the TC1766 User's Manual System Units part (Volume 1).*

24.3.3.2 Fractional Divider Register

The Fractional Divider Register allows the programmer to control the clock rate of the module clock f_{FADC}.

FADC_FDR
FADC Fractional Divider Register ($0C_H$) 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 $\overline{\text{SPNDACK}}$ 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]	rw	**Reserved** Read as 0. Should be written with 0.

Note: Additional details on the fractional divider register functionality are described in section ***"Fractional Divider Operation" on Page 3-29*** *of the TC1766 User's Manual System Units part (Volume 1).*

24.3.4 Port Control

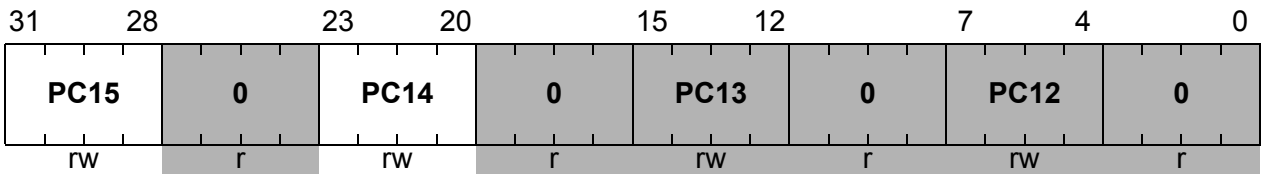
The external request inputs REQ0, REQ1, REQ4, and REQ5 used by the FADC module are controlled in the port logic. If these input lines are used as FADC input, the input function selection (IOCR registers) must be checked.

Input Function Selection

The port input/output control registers contain the bit fields that select the FADC's digital input driver characteristics such as pull-up/pull-down device connection capabilities. The request input lines used by the FADC module are controlled by the port input/output control registers P0_IOCR12 and P3_IOCR8.

After reset, for the external request inputs the input function with a pull-up device is selected. For reference, the P0_IOCR12 and P3_IOCR8 functionality is shown on the next page in respect to the FADC external request inputs.

P0_IOCR12
Port 0 Input/Output Control Register 12
($1C_H$) **Reset Value: 2020 2020_H**

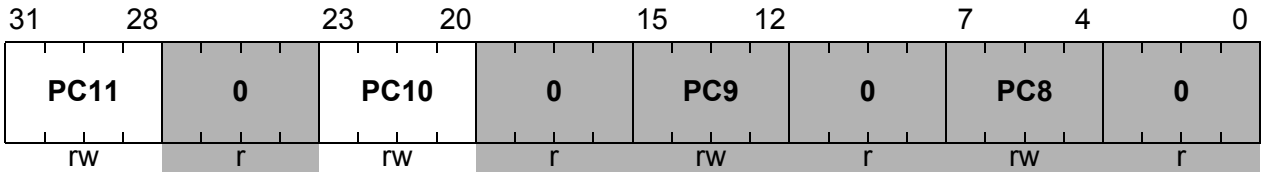


Field	Bits	Type	Description
PC14, PC15	[23:20], [31:28]	rw	**Port Output Control for Port 0.[15:12]**[1)] These bit fields determine the output port functionality: Port input/output control for P0.14/REQ4 Port input/output control for P0.15/REQ5

1) For coding of bit fields, see **Table 24-9**. Shaded bits and bit fields are “don’t care” for FADC I/O port control.

P3_IOCR8
Port 3 Input/Output Control Register 8

(18_H) **Reset Value: 2020 2020_H**



Field	Bits	Type	Description
PC10, PC11	[23:20], [31:28]	rw	**Port Output Control for Port 3[11:8]**[1)] These bit fields determine the output port functionality: Port input/output control for P3.10/REQ0 Port input/output control for P3.11/REQ1

1) For coding of bit fields, see **Table 24-9**. Shaded bits and bit fields are “don’t care” for FADC I/O port control.

PCx Coding

Table 24-9 PCx Coding for FADC Inputs

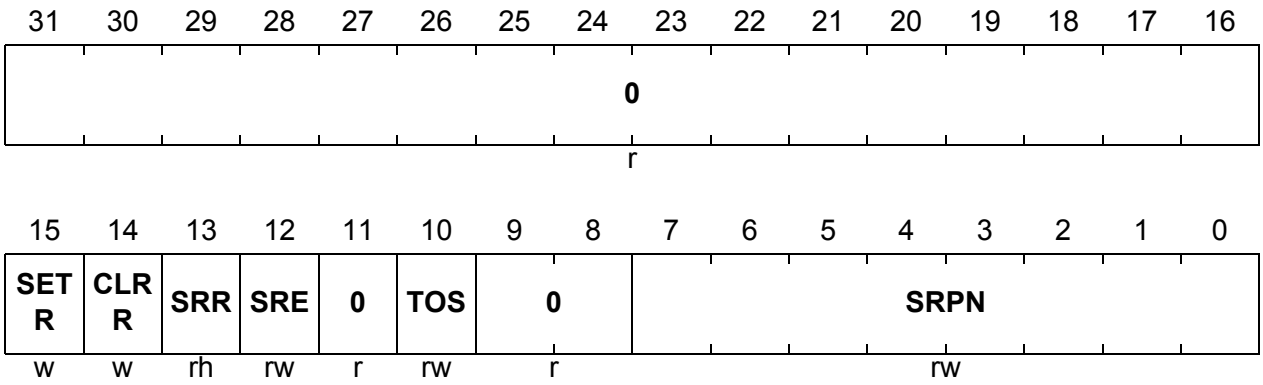
PCx[3:0]	I/O	Selected Input Function
$0X00_B$	Input	No pull device connected
$0X01_B$		Pull-down device connected
$0X10_B$[1)]		Pull-up device connected
$0X11_B$		No pull device connected

1) Default after reset.

24.3.5 Interrupt Control

Each of the two interrupts of the FADC is controlled by a service request control register.

FADC_SRCm (m = 0-1)
FADC Service Request Control Register m
(FC_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 on **Page 12-3** of the TC1766 User's Manual System Units part (Volume 1).*

24.3.6 On-Chip Connections

This section describes the on-chip connections of the FADC module.

24.3.6.1 Analog Input Lines

The analog inputs FAINxP/FAINxN (x = 0-1) are connected to the TC1766 analog input pins AN[35:32] as shown on Page 24-55.

24.3.6.2 Trigger/Gating Source Input Connections

Gating source inputs GS[7:0] and trigger source inputs TS[7:0] of the FADC module are connected to GPTA0 module outputs, of GPIO port lines, and external request unit outputs according to Table 24-10.

Table 24-10 FADC Trigger/Gating Source Input Connections

Inputs	Signal	From Module
Gating Source Inputs		
GS0	P3.10 / REQ0	Port 3
GS1	P0.14 / REQ4	Port 0
GS2	PDOUT2	External Request Unit ERU (located in the SCU)
GS3	PDOUT3	
GS4	OUT1	GPTA0
GS5	OUT9	
GS6	OUT18	
GS7	OUT26	
Trigger Source Inputs		
TS0	P3.11 / REQ1	Port 3
TS1	P0.15 / REQ5	Port 0
TS2	PDOUT2	External Request Unit ERU (located in the SCU)
TS3	PDOUT3	
TS4	OUT2	GPTA0
TS5	OUT10	
TS6	OUT19	
TS7	OUT27	

24.3.6.3 Service Request Lines

The FADC module has four service request output lines. These service request outputs are connected to interrupt service request nodes with its control register (see Page 24-62) and with several request inputs of the TC1766's DMA controller. Table 24-11 shows the FADC service request line interconnections.

Table 24-11 Service Request Lines Interconnections

Module	Service Request Line	Connected To	Description
FADC	SR0	FADC_SRC0	FADC Service Request Node 0
	SR1	FADC_SRC1	FADC Service Request Node 1
	SR2	CH00_REQI2 CH02_REQI2	DMA Channel 00 Request Input 2 DMA Channel 02 Request Input 2
	SR3	CH01_REQI2 CH03_REQI2	DMA Channel 01 Request Input 2 DMA Channel 03 Request Input 2

Keyword Index

This section lists a number of keywords which refer to specific details of the TC1766 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 TC1766 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

G

I

L

M

N

O

P

R

W

Register Index

This section lists the references to the Special Function Registers of the TC1766. It refers to page numbers in both parts of the TC1766 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

W

w w w . i n f i n e o n . c o m

Published by Infineon Technologies AG