XE166 Derivatives

16-Bit Single-Chip Real Time Signal Controller

Volume 1 (of 2): System Units

Microcontrollers

Never stop thinking

Edition 2008-08

Published by
Infineon Technologies AG
81726 Munich, Germany

Legal Disclaimer

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

Information

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

Warnings

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

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

XE166 Derivatives

16-Bit Single-Chip Real Time Signal Controller

Volume 1 (of 2): System Units

Microcontrollers

Never stop thinking

XE16x
Revision History: V2.1, 2008-08

Previous Version(s):
V2.0, 2007-12
V1.0, 2007-10

Page	Subjects (major changes since last revision)
1-5	Derivative synopsis table replaced by reference to Data Sheets
3-49ff	Recommendations for Flash usage added
6-1ff	Various descriptions refined and corrected (throughout the chapter)
6-77ff	Description of EVR registers corrected
6-112ff	Software Boot Support added
6-155ff	Description of register ISSR corrected
6-161ff	Description of Double Watchdog Timer Error corrected
6-182	Memory Content Protection added (was in memory chapter before)
8-2	Description of pin TRef updated
9-1ff	Minor updates in description of EBC
10-2ff	Status information added
10-29	Bootstrap loader information improved
11-1ff	Minor updates in debug chapter

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

Summary Of Chapters

This User's Manual consists of two Volumes, "System Units" and "Peripheral Units". For a quick overview this table of chapters summarizes both volumes, so you immediately can find the reference to the desired section in the corresponding document ([1] or [2]).

Table Of Contents

This User's Manual consists of two Volumes, "System Units" and "Peripheral Units". For your convenience this table of contents (and also the keyword and register index) lists both volumes, so you can immediately find the reference to the desired section in the corresponding document ([1] or [2]).

Table Of Contents

1 Introduction

The rapidly growing area of embedded control applications is representing one of the most time-critical operating environments for today's microcontrollers. Complex control algorithms have to be processed based on a large number of digital as well as analog input signals, and the appropriate output signals must be generated within a defined maximum response time. Embedded control applications also are often sensitive to board space, power consumption, and overall system cost.

Embedded control applications therefore require microcontrollers, which:

- offer a high level of system integration
- eliminate the need for additional peripheral devices and the associated software overhead
- provide system security and fail-safe mechanisms
- provide effective means to control (and reduce) the device's power consumption

The increasing complexity of embedded control applications requires microcontrollers for new high-end embedded control systems to possess a significant CPU performance and peripheral functionality. To achieve this high performance goal, Infineon has decided to develop its families of 16/32-bit CMOS microcontrollers without the constraints of backward compatibility wih previous architectures.

Nonetheless the architectures of these microcontroller families pursue successful hardware and software concepts, which have been established in Infineon's popular 8-bit controller families, while delivering 32-bit performance.

This established functionality, which has been the basis for system solutions in a wide range of application areas, is amended with flexible peripheral modules and effective power control features. The sum of this provides the prerequisites for powerful, yet efficient systems-on-chip.

Solutions for Industrial Systems

The XE166 derivatives provide solutions for the requirements of manifold industrial applications by combining versatile controlling power with DSP functionality and sophisticated peripheral modules. Thus, it supports a wide field of applications:

- Motor Control (pumps, fans, compressors, servos, CNC-machines, robots, process control, conveyor belts, …)
- Renewable Energy (wind energy converters, photovoltaics, fuel cells, battery storage, hydro generators, micro turbines, …)
- Power Supply (Uninterruptable Power Supplies, general power supplies, battery chargers, lamp ballast, …)
- Transportation (locomotives, trains, subways, buses, trucks, fork lifts, agricultural vehicles, traffic lights, …)

Infineon's high quality standards make the XE166 Real-Time Controllers an ideal choice for robust and reliable systems.

About this Manual

This manual describes the functionality of a number of microcontroller types of the Infineon XE166 Family.

These microcontrollers provide identical functionality to a large extent, but each device type has specific unique features as indicated here.

The descriptions in this manual cover a superset of the provided features.
The **“Summary of Basic Features” on Page 1-5** lists the derivatives covered by this manual for a quick summary and comparison.

This manual is valid for these derivatives and describes all variations of the different available temperature ranges and packages.

For simplicity, these various device types are referred to by the collective term **XE16x** throughout this manual. The complete Pro Electron conforming designations are listed in the respective Data Sheets.

Some sections of this manual do not refer to all of the XE16x derivatives which are currently available or planned (such as devices with different types of on-chip memory or peripherals). These sections contain respective notes wherever possible.

1.1 Members of the 16-bit Microcontroller Family

The microcontrollers in the Infineon 16-bit family have been designed to meet the high performance requirements of real-time embedded control applications. The architecture of this family has been optimized for high instruction throughput and minimized response time to external stimuli (interrupts). Intelligent peripheral subsystems have been integrated to reduce the need for CPU intervention to a minimum extent. This also minimizes the need for communication via the external bus interface. The high flexibility of this architecture allows to serve the diverse and varying needs of different application areas such as automotive, industrial control, or data communications.

The core of the 16-bit family has been developed with a modular family concept in mind. All family members execute an efficient control-optimized instruction set (additional instructions for members of the second generation). This allows easy and quick implementation of new family members with different internal memory sizes and technologies, different sets of on-chip peripherals, and/or different numbers of IO pins.

The XBUS/LXBus concept (internal representation of the external bus interface) provides a straightforward path for building application-specific derivatives by integrating application-specific peripheral modules with the standard on-chip peripherals.

As programs for embedded control applications become larger, high level languages are favored by programmers, because high level language programs are easier to write, to debug and to maintain. The C166 Family supports this starting with its 2nd generation.

The 80C166-type microcontrollers were the **first generation** of the 16-bit controller family. These devices established the C166 architecture.

The C165-type and C167-type devices are members of the **second generation** of this family. This second generation is even more powerful due to additional instructions for HLL support, an increased address space, increased internal RAM, and highly efficient management of various resources on the external bus.

Enhanced derivatives of this second generation provide more features such as additional internal high-speed RAM, an integrated CAN-Module, an on-chip PLL, etc.

The design of more efficient systems may require the integration of application-specific peripherals to boost system performance while minimizing the part count. These efforts are supported by the XBUS, defined for the Infineon 16-bit microcontrollers (second generation). The XBUS is an internal representation of the external bus interface which opens and simplifies the integration of peripherals by standardizing the required interface. One representative taking advantage of this technology is the integrated CAN module.

The C165-type devices are reduced functionality versions of the C167 because they do not have the A/D converter, the CAPCOM units, and the PWM module. This results in a smaller package, reduced power consumption, and design savings.

The C164-type devices, the C167CS derivatives, and some of the C161-type devices are further enhanced by a flexible power management and form the **third generation** of

the 16-bit controller family. This power management mechanism provides an effective means to control the power that is consumed in a certain state of the controller and thus minimizes the overall power consumption for a given application.

The XC16x derivatives represent the **fourth generation** of the 16-bit controller family. The XC166 Family dramatically increases the performance of 16-bit microcontrollers by several major improvements and additions. The MAC-unit adds DSP-functionality to handle digital filter algorithms and greatly reduces the execution time of multiplications and divisions. The 5-stage pipeline, single-cycle execution of most instructions, and PEC-transfers within the complete addressing range increase system performance. Debugging the target system is supported by integrated functions for On-Chip Debug Support (OCDS).

The present XE166 Family of microcontrollers builds the **fifth generation** of 16-bit microcontrollers which provides 32-bit performance and takes users and applications a considerable step towards industry's target of systems on chip. Integrated memories and peripherals allow compact systems, the integrated core power supply and control reduces system requirements to one single voltage supply, the powerful combination of CPU and MAC-unit is unleashed by optimized compilers. This leaves no performance gap towards 32-bit systems.

A variety of different versions is provided which offer various kinds of on-chip program memory[1)]:

- Mask-programmable ROM
- Flash memory
- OTP memory
- ROMless without non-volatile memory.

Also there are devices with specific functional units.

The devices may be offered in different packages, temperature ranges and speed classes.

Additional standard and application-specific derivatives are planned and are in development.

Note: Not all derivatives will be offered in all temperature ranges, speed classes, packages, or program memory variations.

Information about specific versions and derivatives will be made available with the devices themselves. Contact your Infineon representative for up-to-date material or refer to **http://www.infineon.com/microcontrollers**.

Note: As the architecture and the basic features, such as the CPU core and built-in peripherals, are identical for most of the currently offered versions of the XE16x, descriptions within this manual that refer to the "XE16x" also apply to the other variations, unless otherwise noted.

1) Not all derivatives are offered with all kinds of on-chip memory.

1.2 Summary of Basic Features

The XE16x devices are enhanced members of the Infineon XE166 Family of full featured single-chip CMOS microcontrollers.
This manual covers several device types are covered in this manual. The various derivates are referred to as XC2200 throughout this manual.

Note: Please refer to the corresponding Data Sheets for a description of the features of a specific device type.

The XE16x combines the extended functionality and performance of the C166SV2 Core with powerful on-chip peripheral subsystems and on-chip memory units and provides several means for power reduction.
The following key features contribute to the high performance of the XE16x:

Intelligent On-Chip Peripheral Subsystems

- Two synchronizable A/D Converters with programmable resolution (10-bit or 8-bit) and conversion time (down to approx. 1 µs), up to 24 analog input channels, auto scan modes, channel injection, data reduction features
- One Capture/Compare Unit with 2 independent time bases,
  very flexible PWM unit/event recording unit with different operating modes,
  includes two 16-bit timers/counters, maximum resolution f_{SYS}
- Up to Four Capture/Compare Units for flexible PWM Signal Generation (CCU6)
  (3/6 Capture/Compare Channels and 1 Compare Channel)
- Two Multifunctional General Purpose Timer Units:
  – GPT1: three 16-bit timers/counters, maximum resolution $f_{SYS}/4$
  – GPT2: two 16-bit timers/counters, maximum resolution $f_{SYS}/2$
- Up to Six Serial Channels with baud rate generator, receive/transmit FIFOs, programmable data length and shift direction, usable as UART, SPI-like, IIC, IIS, and LIN interface
- Controller Area Network (MultiCAN) Module, Rev. 2.0B active,
  up to five nodes operating independently or exchanging data via a gateway function, Full-CAN/Basic-CAN
- Real Time Clock with alarm interrupt
- Watchdog Timer with programmable time intervals
- Bootstrap Loaders for flexible system initialization
- Protection management for system configuration and control registers

Integrated On-Chip Memories

- 2 Kbytes Dual-Port RAM (DPRAM) for variables, register banks, and stacks
- Up to 16 Kbytes on-chip high-speed Data SRAM (DSRAM) for variables and stacks
- Up to 64 Kbytes on-chip high-speed Program/Data SRAM (PSRAM) for code and data
- Up to 768 Kbytes on-chip Flash Program Memory for instruction code or constant data

Note: The system stack can be located in any memory area within the complete addressing range.

High Performance 16-bit CPU with Five-Stage Pipeline and MAC Unit

- Single clock cycle instruction execution
- 1 cycle minimum instruction cycle time (most instructions)
- 1 cycle multiplication (16-bit × 16-bit)
- 4 + 17 cycles division (32-bit / 16-bit), 4 cycles delay, 17 cycles background execution
- 1 cycle multiply and accumulate instruction (MAC) execution
- Automatic saturation or rounding included
- Multiple high bandwidth internal data buses
- Register-based design with multiple, variable register banks
- Two additional fast register banks
- Fast context switching support
- 16 Mbytes of linear address space for code and data (von Neumann architecture)
- System stack cache support with automatic stack overflow/underflow detection
- High performance branch, call, and loop processing
- Zero-cycle jump execution

Control Oriented Instruction Set with High Efficiency

- Bit, byte, and word data types
- Flexible and efficient addressing modes for high code density
- Enhanced boolean bit manipulation with direct addressability of 6 Kbits for peripheral control and user-defined flags
- Hardware traps to identify exception conditions during runtime
- HLL support for semaphore operations and efficient data access

Power Management Features

- Two IO power domains fulfill system requirements from 3 V to 5 V
- Gated clock concept for improved power consumption and EMC
- Programmable system slowdown via clock generation unit
- Flexible management of peripherals, can be individually disabled
- Programmable frequency output

16-Priority-Level Interrupt System

- 96 interrupt nodes with separate interrupt vectors on 15 priority levels (8 group levels)
- 7 cycles minimum interrupt latency in case of internal program execution
- Fast external interrupts
- Programmable external interrupt source selection
- Programmable vector table (start location and step-width)

8-Channel Peripheral Event Controller (PEC)

- Interrupt driven single cycle data transfer
- Programmable PEC interrupt request level, (15 down to 8)
- Transfer count option
  (standard CPU interrupt after programmable number of PEC transfers)
- Separate interrupt level for PEC termination interrupts selectable
- Overhead from saving and restoring system state for interrupt requests eliminated
- Full 24-bit addresses for source and destination pointers, supporting transfers within the total address space

On-Chip Debug Support

- On-chip debug controller and related interface to JTAG controller
- JTAG interface and break interface
- Hardware, software and external pin breakpoints
- Up to 4 instruction pointer breakpoints
- Debug event control, e.g. with monitor call or CPU halt or trigger of data transfer
- Dedicated DEBUG instructions with control via JTAG interface
- Access to any internal register or memory location via JTAG interface
- Single step support and watchpoints with MOV-injection

Input/Output Lines With Individual Bit Addressability

- Tri-stated in input mode
- Push/pull or open drain output mode
- Programmable port driver control
- Two I/O power domains with a supply voltage range from 3.0 V to 5.5 V
  (core-logic and oscillator input voltage is 1.5 V)

Infineon CMOS Process

- Low power CMOS technology enables power saving modes with flexible power management.

Green Plastic Low-Profile Quad Flat Pack (LQFP) Packages

- PG-LQFP-144, 20 × 20 mm body, 0.5 mm (19.7 mil) lead spacing,
  surface mount technology
- PG-LQFP-100, 14 × 14 mm body, 0.5 mm (19.7 mil) lead spacing,
  surface mount technology

Complete Development Support

For the development tool support of its microcontrollers, Infineon follows a clear third party concept. Currently around 120 tool suppliers world-wide, ranging from local niche

manufacturers to multinational companies with broad product portfolios, offer powerful development tools for the Infineon C500, C800, XC800, C166, XC166, XC2000, XE166, and TriCore microcontroller families, guaranteeing a remarkable variety of price-performance classes as well as early availability of high quality key tools such as compilers, assemblers, simulators, debuggers or in-circuit emulators.

Infineon incorporates its strategic tool partners very early into the product development process, making sure embedded system developers get reliable, well-tuned tool solutions, which help them unleash the power of Infineon microcontrollers in the most effective way and with the shortest possible learning curve.

The tool environment for the Infineon 16-bit microcontrollers includes the following tools:

- Compilers (C/C++)
- Macro-assemblers, linkers, locators, library managers, format-converters
- Architectural simulators
- HLL debuggers
- Real-time operating systems
- VHDL chip models
- In-circuit emulators (based on bondout or standard chips)
- Plug-in emulators
- Emulation and clip-over adapters, production sockets
- Logic analyzer disassemblers
- Starter kits
- Evaluation boards with monitor programs
- Industrial boards (also for CAN, FUZZY, PROFIBUS, FORTH applications)
- Low level driver software (CAN, PROFIBUS, LIN)
- Chip configuration code generation tool (DAvE)

1.3 Abbreviations

The following acronyms and terms are used within this document:

ADC	Analog Digital Converter
ALE	Address Latch Enable
ALU	Arithmetic and Logic Unit
ASC	Asynchronous/synchronous Serial Channel
CAN	Controller Area Network (License Bosch)
CAPCOM	CAPture and COMpare unit
CISC	Complex Instruction Set Computing
CMOS	Complementary Metal Oxide Silicon
CPU	Central Processing Unit
DMU	Data Management Unit
EBC	External Bus Controller
ESFR	Extended Special Function Register
EVVR	Embedded Validated Voltage Regulator
Flash	Non-volatile memory that may be electrically erased
GPR	General Purpose Register
GPT	General Purpose Timer unit
HLL	High Level Language
IIC	Inter Integrated Circuit (Bus)
IIS	Inter Integrated Circuit Sound (Bus)
IO	Input/Output
JTAG	Joint Test Access Group
LIN	Local Interconnect Network
LPR	Low Power Reference
LQFP	Low Profile Quad Flat Pack
LXBus	Internal representation of the external bus
MAC	Multiply/Accumulate (unit)
OCDS	On-Chip Debug Support
OTP	One-Time Programmable memory
PEC	Peripheral Event Controller

PLA	Programmable Logic Array
PLL	Phase Locked Loop
PMU	Program Management Unit
PVC	Power Validation Circuit
PWM	Pulse Width Modulation
RAM	Random Access Memory
RISC	Reduced Instruction Set Computing
ROM	Read Only Memory
RTC	Real Time Clock
SFR	Special Function Register
SSC	Synchronous Serial Channel
SWD	Supply Watchdog
UART	Universal Asynchronous Receiver/Transmitter
USIC	Universal Serial Interface Channel

1.4 Naming Conventions

The diverse bitfields used for control functions and status indication and the registers housing them are equipped with unique names wherever applicable. Thereby these control structures can be referred to by their names rather than by their location. This makes the descriptions by far more comprehensible.

To describe regular structures (such as ports) indices are used instead of a plethora of similar bit names, so bit 3 of port 5 is referred to as P5.3.

Where it helps to clarify the relation between several named structures, the next higher level is added to the respective name to make it unambiguous.

The term ADC0_GLOBCTR clearly identifies register GLOBCTR as part of module ADC0, the term SYSCON0.CLKSEL clearly identifies bitfield CLKSEL as part of register SYSCON0.

2 Architectural Overview

The architecture of the XE16x core combines the advantages of both RISC and CISC processors in a very well-balanced way. This computing and controlling power is completed by the DSP-functionality of the MAC-unit. The XE16x integrates this powerful CPU core with a set of powerful peripheral units into one chip and connects them very efficiently. On-chip memory blocks with dedicated buses and control units store code and data. This combination of features results in a high performance microcontroller, which is the right choice not only for today's applications, but also for future engineering challenges. One of the buses used concurrently on the XE16x is the LXBus, an internal representation of the external bus interface. This bus provides a standardized method for integrating additional application-specific peripherals into derivatives of the standard XE16x.

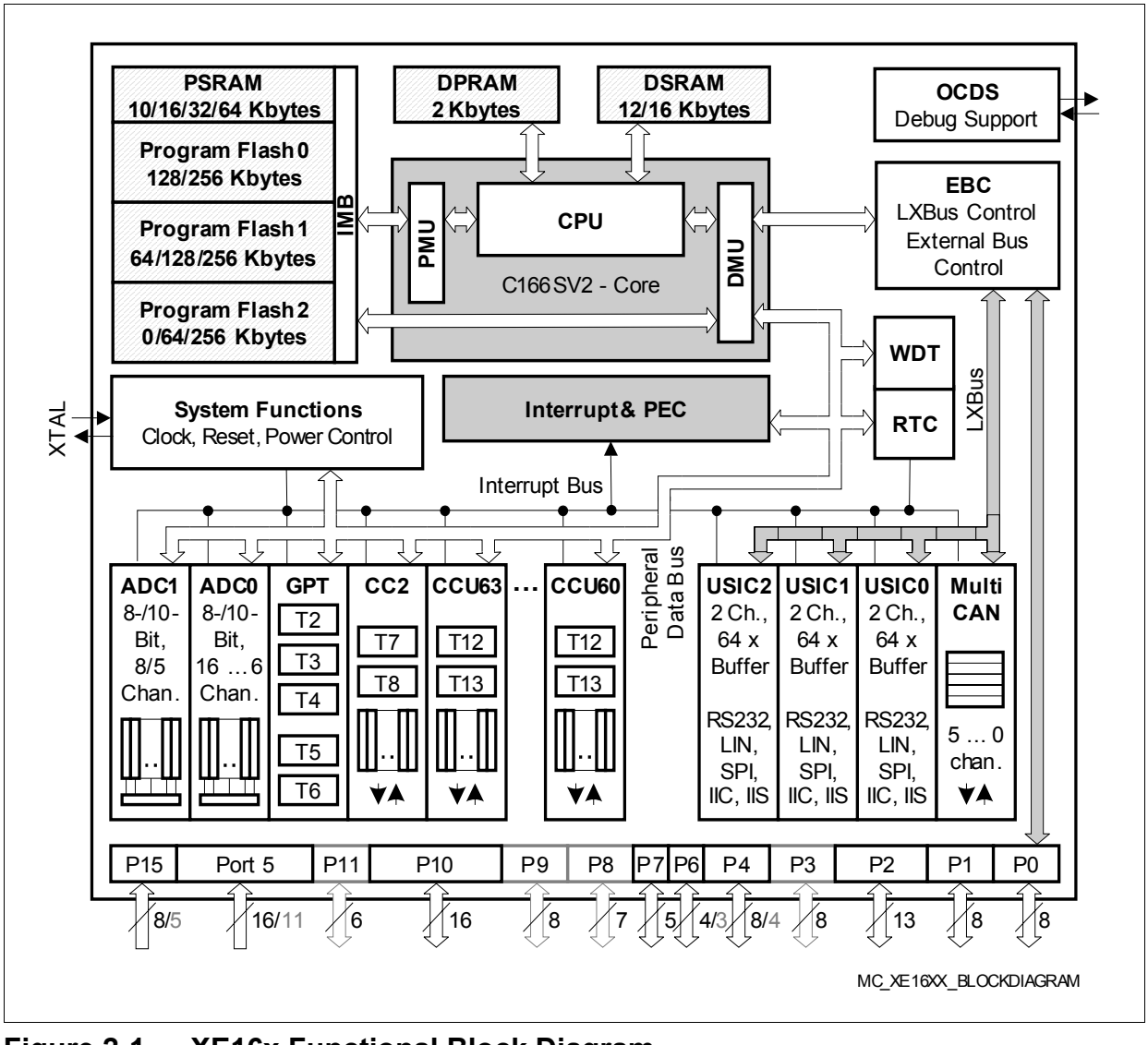


Figure 2-1 XE16x Functional Block Diagram

2.1 Basic CPU Concepts and Optimizations

The main core of the CPU consists of a set of optimized functional units including the instruction fetch/processing pipelines, a 16-bit Arithmetic and Logic Unit (ALU), a 40-bit Multiply and Accumulate Unit (MAC), an Address and Data Unit (ADU), an Instruction Fetch Unit (IFU), a Register File (RF), and dedicated Special Function Registers (SFRs).

Single clock cycle execution of instructions results in superior CPU performance, while maintaining C166 code compatibility. Impressive DSP performance, concurrent access to different kinds of memories and peripherals boost the overall system performance.

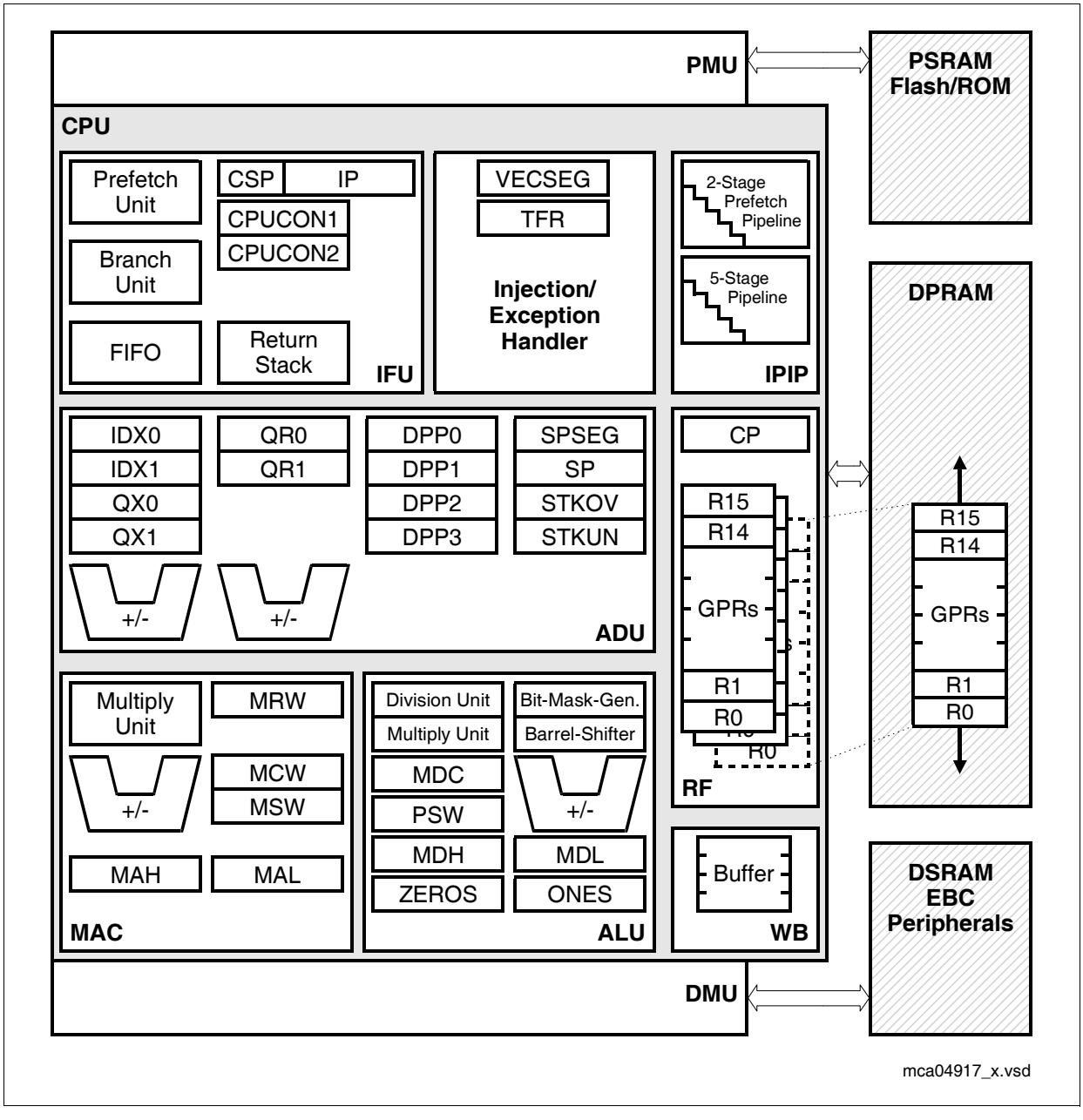


Figure 2-2 CPU Block Diagram

Summary of CPU Features

- Opcode fully upward compatible with C166 Family
- 2-stage instruction fetch pipeline with FIFO for instruction pre-fetching
- 5-stage instruction execution pipeline
- Pipeline forwarding controls data dependencies in hardware
- Multiple high bandwidth buses for data and instructions
- Linear address space for code and data (von Neumann architecture)
- Nearly all instructions executed in one CPU clock cycle
- Fast multiplication (16-bit × 16-bit) in one CPU clock cycle
- Fast background execution of division (32-bit/16-bit) in 21 CPU clock cycles
- Built-in advanced MAC (Multiply Accumulate) Unit:
  - Single cycle MAC instruction with zero cycle latency including a 16 × 16 multiplier
  - 40-bit barrel shifter and 40-bit accumulator to handle overflows
  - Automatic saturation to 32 bits or rounding included with the MAC instruction
  - Fractional numbers supported directly
  - One Finite Impulse Response Filter (FIR) tap per cycle with no circular buffer management
- Enhanced boolean bit manipulation facilities
- High performance branch-, call-, and loop-processing
- Zero cycle jump execution
- Register-based design with multiple variable register banks (byte or word operands)
- Two additional fast register banks
- Variable stack with automatic stack overflow/underflow detection
- “Fast interrupt” and “Fast context switch” features

The high performance and flexibility of the CPU is achieved by a number of optimized functional blocks (see **Figure 2-2**). Optimizations of the functional blocks are described in detail in the following sections.

2.1.1 High Instruction Bandwidth/Fast Execution

Based on the hardware provisions, most of the XE16x's instructions can be executed in just one clock cycle ($1/f_{SYS}$). This includes arithmetic instructions, logic instructions, and move instructions with most addressing modes.

Special instructions such as JMPS take more than one machine cycle. Divide instructions are mainly executed in the background, so other instructions can be executed in parallel. Due to the prediction mechanism (see **Section 4.2**), correctly predicted branch instructions require only one cycle or can even be overlaid with another instruction (zero-cycle jump).

The instruction cycle time is dramatically reduced through the use of instruction pipelining. This technique allows the core CPU to process portions of multiple sequential instruction stages in parallel. Up to seven stages can operate in parallel:

The two-stage instruction fetch pipeline fetches and preprocesses instructions from the respective program memory:

PREFETCH: Instructions are prefetched from the PMU in the predicted order. The instructions are preprocessed in the branch detection unit to detect branches. The prediction logic determines if branches are assumed to be taken or not.

FETCH: The instruction pointer for the next instruction to be fetched is calculated according to the branch prediction rules. The branch folding unit preprocesses detected branches and combines them with the preceding instructions to enable zero-cycle branch execution. Prefetched instructions are stored in the instruction FIFO, while stored instructions are moved from the instruction FIFO to the instruction processing pipeline.

The five-stage instruction processing pipeline executes the respective instructions:

DECODE: The previously fetched instruction is decoded and the GPR used for indirect addressing is read from the register file, if required.

ADDRESS: All operand addresses are calculated. For instructions implicitly accessing the stack the stack pointer (SP) is decremented or incremented.

MEMORY: All required operands are fetched.

EXECUTE: The specified operation (ALU or MAC) is performed on the previously fetched operands. The condition flags are updated. Explicit write operations to CPU-SFRs are executed. GPRs used for indirect addressing are incremented or decremented, if required.

WRITE BACK: The result operands are written to the specified locations. Operands located in the DPRAM are stored via the write-back buffer.

2.1.2 Powerful Execution Units

The 16-bit Arithmetic and Logic Unit (ALU) performs all standard (word) arithmetic and logical operations. Additionally, for byte operations, signals are provided from bits 6 and 7 of the ALU result to set the condition flags correctly. Multiple precision arithmetic is provided through a 'CARRY-IN' signal to the ALU from previously calculated portions of the desired operation.

Most internal execution blocks have been optimized to perform operations on either 8-bit or 16-bit quantities. Instructions have been provided as well to allow byte packing in memory while providing sign extension of bytes for word wide arithmetic operations. The internal bus structure also allows transfers of bytes or words to or from peripherals based on the peripheral requirements.

A set of consistent flags is updated automatically in the PSW after each arithmetic, logical, shift, or movement operation. These flags allow branching on specific conditions. Support for both signed and unsigned arithmetic is provided through user-specifiable branch tests. These flags are also preserved automatically by the CPU upon entry into an interrupt or trap routine.

A 16-bit barrel shifter provides multiple bit shifts in a single cycle. Rotates and arithmetic shifts are also supported.

The Multiply and Accumulate Unit (MAC) performs extended arithmetic operations such as 32-bit addition, 32-bit subtraction, and single-cycle 16-bit × 16-bit multiplication. The combined MAC operations (multiplication with cumulative addition/subtraction) represent the major part of the DSP performance of the CPU.

The Address Data Unit (ADU) contains two independent arithmetic units to generate, calculate, and update addresses for data accesses. The ADU performs the following major tasks:

- The Standard Address Unit supports linear arithmetic for the short, long, and indirect addressing modes. It also supports data paging and stack handling.
- The DSP Address Generation Unit contains an additional set of address pointers and offset registers which are used in conjunction with the CoXXX instructions only.

The CPU provides a lot of powerful addressing modes for word, byte, and bit data accesses (short, long, indirect). The different addressing modes use different formats and have different scopes.

Dedicated bit processing instructions provide efficient control and testing of peripherals while enhancing data manipulation. These instructions provide direct access to two operands in the bit-addressable space without requiring them to be moved into temporary flags. Logical instructions allow the user to compare and modify a control bit for a peripheral in one instruction. Multiple bit shift instructions (single cycle execution) avoid long instruction streams of single bit shift operations. Bitfield instructions allow the modification of multiple bits from one operand in a single instruction.

2.1.3 High Performance Branch-, Call-, and Loop-Processing

Pipelined execution delivers maximum performance with a stream of subsequent instructions. Any disruption requires the pipeline to be refilled and the new instruction to step through the pipeline stages. Due to the high percentage of branching in controller applications, branch instructions have been optimized to require pipeline refilling only in special cases. This is realized by detecting and preprocessing branch instructions in the prefetch stage and by predicting the respective branch target address.

Prefetching then continues from the predicted target address. If the prediction was correct subsequent instructions can be fed to the execution pipeline without a gap, even if a branch is executed, i.e. the code execution is not linear. Branch target prediction (see also **Section 4.2.1**) uses the following rules:

- **Unconditional branches:** Branch prediction is trivial in this case, as the branches will always be taken and the target address is defined. This applies to implicitly unconditional branches such as JMPS, CALLR, or RET as well as to branches with condition code “unconditional” such as JMPI cc_UC.
- **Fixed prediction:** Branch instructions which are often used to realize loops are assumed to be taken if they branch backward to a previous location (the begin of the loop). This applies to conditional branches such as JMPR cc_XX or JNB.
- **Variable prediction:** In this case the respective prediction (taken or not taken) is coded into the instruction and can, therefore, be selected for each individual branch instruction. Thus, the software designer can optimize the instruction flow to the specific code to be executed[1)]. This applies to the branch instructions JMPA and CALLA.
- **Conditional indirect branches:** These branches are always assumed to be not taken. This applies to branch instructions JMPI cc_XX, [Rw] and CALLI cc_XX, [Rw].

The system state information is saved automatically on the internal system stack, thus avoiding the use of instructions to preserve state upon entry and exit of interrupt or trap routines. Call instructions push the value of the IP on the system stack, and require the same execution time as branch instructions. Additionally, instructions have been provided to support indirect branch and call instructions. This feature supports implementation of multiple CASE statement branching in assembler macros and high level languages.

1) The programming tools accept either dedicated mnemonics for each prediction leaving the choice up to programmer, or they accept generic mnemonics and apply their own prediction rules.

2.1.4 Consistent and Optimized Instruction Formats

To obtain optimum performance in a pipelined design, an instruction set has been designed which incorporates concepts from Reduced Instruction Set Computing (RISC). These concepts primarily allow fast decoding of the instructions and operands while reducing pipeline holds. These concepts, however, do not preclude the use of complex instructions required by microcontroller users. The instruction set was designed to meet the following goals:

- Provide powerful instructions for frequently-performed operations which traditionally have required sequences of instructions. Avoid transfer into and out of temporary registers such as accumulators and carry bits. Perform tasks in parallel such as saving state upon entry into interrupt routines or subroutines.
- Avoid complex encoding schemes by placing operands in consistent fields for each instruction and avoid complex addressing modes which are not frequently used. Consequently, the instruction decode time decreases and the development of compilers and assemblers is simplified.
- Provide most frequently used instructions with one-word instruction formats. All other instructions use two-word formats. This allows all instructions to be placed on word boundaries: this alleviates the need for complex alignment hardware. It also has the benefit of increasing the range for relative branching instructions.

The high performance of the CPU-hardware can be utilized efficiently by a programmer by means of the highly functional XE16x instruction set which includes the following instruction classes:

- Arithmetic Instructions
- DSP Instructions
- Logical Instructions
- Boolean Bit Manipulation Instructions
- Compare and Loop Control Instructions
- Shift and Rotate Instructions
- Prioritize Instruction
- Data Movement Instructions
- System Stack Instructions
- Jump and Call Instructions
- Return Instructions
- System Control Instructions
- Miscellaneous Instructions

Possible operand types are bits, bytes, words, and doublewords. Specific instructions support the conversion (extension) of bytes to words. Various direct, indirect, and immediate addressing modes are provided to specify the required operands.

2.1.5 Programmable Multiple Priority Interrupt System

The XE16x provides 96 separate interrupt nodes that may be assigned to 16 priority levels with 8 group priorities on each level. Most interrupt sources are connected to a dedicated interrupt node. In some cases, multi-source interrupt nodes are incorporated for efficient use of system resources. These nodes can be activated by several source requests and are controlled via interrupt subnode control registers.

The following enhancements within the XE16x allow processing of a large number of interrupt sources:

- Peripheral Event Controller (PEC): This processor is used to off-load many interrupt requests from the CPU. It avoids the overhead of entering and exiting interrupt or trap routines by performing single-cycle interrupt-driven byte or word data transfers between any two locations with an optional increment of the PEC source pointer, the destination pointer, or both. Only one cycle is ‘stolen’ from the current CPU activity to perform a PEC service.
- Multiple Priority Interrupt Controller: This controller allows all interrupts to be assigned any specified priority. Interrupts may also be grouped, which enables the user to prevent similar priority tasks from interrupting each other. For each of the interrupt nodes, there is a separate control register which contains an interrupt request flag, an interrupt enable flag, and an interrupt priority bitfield. After being accepted by the CPU, an interrupt service can be interrupted only by a higher prioritized service request. For standard interrupt processing, each of the interrupt nodes has a dedicated vector location.
- Multiple Register Banks: Two local register banks for immediate context switching add to a relocatable global register bank. The user can specify several register banks located anywhere in the internal DPRAM and made of up to sixteen general purpose registers. A single instruction switches from one register bank to another (switching banks flushes the pipeline, changing the global bank requires a validation sequence).

The XE16x is capable of reacting very quickly to non-deterministic events because its interrupt response time is within a very narrow range of typically 7 clock cycles (in the case of internal program execution). Its fast external interrupt inputs are sampled every clock cycle and allow even very short external signals to be recognized.

The XE16x also provides an excellent mechanism to identify and process exceptions or error conditions that arise during run-time, so called ‘Hardware Traps’. A hardware trap causes an immediate non-maskable system reaction which is similar to a standard interrupt service (branching to a dedicated vector table location). The occurrence of a hardware trap is additionally signified by an individual bit in the trap flag register (TFR). Unless another, higher prioritized, trap service is in progress, a hardware trap will interrupt any current program execution. In turn, a hardware trap service can normally not be interrupted by a standard or PEC interrupt.

Software interrupts are supported by means of the ‘TRAP’ instruction in combination with an individual trap (interrupt) number.

2.1.6 Interfaces to System Resources

The CPU of the XE16x interfaces to the system resources via several bus systems which contribute to the overall performance by transferring data concurrently. This avoids stalling the CPU because instructions or operands need to be transferred.

The Dual Port RAM (DPRAM) is directly coupled to the CPU because it houses the global register banks. Transfers from/to these locations affect the performance and are, therefore, carefully optimized.

The Program Management Unit (PMU) controls accesses to the on-chip program memory blocks such as the ROM/Flash module and the Program/Data RAM (PSRAM) and also fetches instructions from external memory.

The 64-bit interface between the PMU and the CPU delivers the instruction words, which are requested by the CPU. The PMU decides whether the requested instruction word has to be fetched from on-chip memory or from external memory.

The Data Management Unit (DMU) controls accesses to the on-chip Data RAM (DSRAM), to the on-chip peripherals connected to the peripheral bus, and to resources on the external bus. External accesses (including accesses to peripherals connected to the on-chip LXBus) are executed by the External Bus Controller (EBC).

The 16-bit interface between the DMU and the CPU handles all data transfers (operands). Data accesses by the CPU are distributed to the appropriate buses according to the defined address map.

PMU and DMU are directly coupled to perform cross-over transfers with high speed. Crossover transfers are executed in both directions:

- **PMU via DMU:** Code fetches from external locations are redirected via the DMU to EBC. Thus, the XE16x can execute code from external resources. No code can be fetched from the Data RAM (DSRAM).
- **DMU via PMU:** Data accesses can also be executed to on-chip resources controlled by the PMU. This includes the following types of transfers:
  - Read a constant from the on-chip program ROM/Flash
  - Read data from the on-chip PSRAM
  - Write data to the on-chip PSRAM (required prior to executing out of it)
  - Program/Erase the on-chip Flash memory

2.2 On-Chip System Resources

The XE16x controllers provide a number of powerful system resources designed around the CPU. The combination of CPU and these resources results in the high performance of the members of this controller family.

Peripheral Event Controller (PEC) and Interrupt Control

The Peripheral Event Controller enables response to an interrupt request with a single data transfer (word or byte) which consumes only one instruction cycle and does not require saving and restoring the machine status. Each interrupt source is prioritized for every machine cycle in the interrupt control block. If PEC service is selected, a PEC transfer is started. If CPU interrupt service is requested, the current CPU priority level stored in the PSW register is tested to determine whether a higher priority interrupt is currently being serviced. When an interrupt is acknowledged, the current state of the machine is saved on the internal system stack and the CPU branches to the system specific vector for the peripheral.

The PEC contains a set of SFRs which store the count value and control bits for eight data transfer channels. In addition, the PEC uses a dedicated area of RAM which contains the source and destination addresses. The PEC is controlled in a manner similar to any other peripheral: through SFRs containing the desired configuration of each channel.

An individual PEC transfer counter is implicitly decremented for each PEC service except in the continuous transfer mode. When this counter reaches zero, a standard interrupt is performed to the vector location related to the corresponding source. PEC services are very well suited, for example, to moving register contents to/from a memory table. The XE16x has eight PEC channels, each of which offers such fast interrupt-driven data transfer capabilities.

Memory Areas

The memory space of the XE16x is configured in a Von Neumann architecture. This means that code memory, data memory, registers, and IO ports are organized within the same linear address space which covers up to 16 Mbytes. The entire memory space can be accessed bytewise or wordwise. Particular portions of the on-chip memory have been made directly bit addressable as well.

Note: The actual memory sizes depend on the selected device type. This overview describes the maximum block sizes.

Up to 768 Kbytes of on-chip Flash memory store code or constant data. The on-chip Flash memory consists of 2 or 3 Flash modules, each built up from 4-Kbyte sectors. Each sector can be separately write protected[1)], erased and programmed (in blocks of 128 bytes). The complete Flash area can be read-protected. A user-defined password sequence temporarily unlocks protected areas. The Flash modules combine 128-bit read accesses with protected and efficient writing algorithms for programming and erasing. Dynamic error correction provides extremely high read data security for all read accesses. Accesses to different Flash modules can be executed in parallel.

Note: Program execution from on-chip program memory is the fastest of all possible alternatives and results in maximum performance. The size of the on-chip program memory depends on the chosen derivative. On-chip program memory also includes the PSRAM.

Up to 64 Kbytes of on-chip Program SRAM (PSRAM) are provided to store user code or data. The PSRAM is accessed via the PMU and is, therefore, optimized for code fetches. A section of the PSRAM with programmable size can be write-protected.

Up to 16 Kbytes of on-chip Data SRAM (DSRAM) are provided as a storage for general user data. The DSRAM is accessed via a separate interface and is, therefore, optimized for data accesses.

2 Kbytes of on-chip Dual-Port RAM (DPRAM) are provided as a storage for user defined variables, for the system stack, and in particular for general purpose register banks. A register bank can consist of up to 16 wordwide (R0 to R15) and/or bytewide (RL0, RH0, …, RL7, RH7) so-called General Purpose Registers (GPRs).
The upper 256 bytes of the DPRAM are directly bitaddressable. When used by a GPR, any location in the DPRAM is bitaddressable.

The CPU has an actual register context of up to 16 wordwide and/or bytewide global GPRs at its disposal, which are physically located within the on-chip RAM area. A Context Pointer (CP) register determines the base address of the active global register bank to be accessed by the CPU at a time. The number of register banks is restricted only by the available internal RAM space. For easy parameter passing, a register bank may overlap other register banks.

A system stack of up to 32 Kwords is provided as storage for temporary data. The system stack can be located anywhere within the complete addressing range and it is accessed by the CPU via the Stack Pointer (SP) register and the Stack Pointer Segment (SPSEG) register. Two separate SFRs, STKOV and STKUN, are implicitly compared against the stack pointer value upon each stack access for the detection of a stack overflow or underflow. This mechanism also supports the control of a bigger virtual stack. Maximum performance for stack operations is achieved by allocating the system stack to internal data RAM areas (DPRAM, DSRAM).

1) To save control bits, sectors are clustered for protection purposes, they remain separate for programming/ erasing.

Hardware detection of the selected memory space is placed at the internal memory decoders and allows the user to specify any address directly or indirectly and obtain the desired data without using temporary registers or special instructions.

For Special Function Registers three areas of the address space are reserved: The standard Special Function Register area (SFR) uses 512 bytes, while the Extended Special Function Register area (ESFR) uses the other 512 bytes. A range of 4 Kbytes is provided for the internal IO area (XSFR). SFRs are wordwide registers which are used for controlling and monitoring functions of the different on-chip units. Unused SFR addresses are reserved for future members of the XE166 Family with enhanced functionality. Therefore, they should either not be accessed, or written with zeros, to ensure upward compatibility.

In order to meet the needs of designs where more memory is required than is provided on chip, up to 12 Mbytes (approximately, see **Table 2-1**) of external RAM and/or ROM can be connected to the microcontroller. The External Bus Interface also provides access to external peripherals.

Table 2-1 XE16x Memory Map

Address Area	Start Loc.	End Loc.	Area Size[1)]	Notes
IMB register space	$FF'FF00_H$	$FF'FFFF_H$	256 Bytes	–
Reserved (Access trap)	$F0'0000_H$	$FF'FEFF_H$	<1 Mbyte	Minus IMB reg.
Reserved for EPSRAM	$E9'0000_H$	$EF'FFFF_H$	448 Kbytes	Mirrors EPSRAM
Emulated PSRAM	$E8'0000_H$	$E8'FFFF_H$	64 Kbytes	Flash timing
Reserved for PSRAM	$E1'0000_H$	$E7'FFFF_H$	448 Kbytes	Mirrors PSRAM
Program SRAM	$E0'0000_H$	$E0'FFFF_H$	64 Kbytes	Maximum speed
Reserved for pr. mem.	$CC'0000_H$	$DF'FFFF_H$	<1.25 Mbytes	–
Program Flash 2	$C8'0000_H$	$CB'FFFF_H$	256 Kbytes	–
Program Flash 1	$C4'0000_H$	$C7'FFFF_H$	256 Kbytes	–
Program Flash 0	$C0'0000_H$	$C3'FFFF_H$	256 Kbytes	2)
External memory area	$40'0000_H$	$BF'FFFF_H$	8 Mbytes	–
Available Ext. IO area[3)]	$20'5800_H$	$3F'FFFF_H$	< 2 Mbytes	Minus USIC/CAN
USIC registers	$20'4000_H$	$20'57FF_H$	6 Kbytes	Accessed via EBC
MultiCAN registers	$20'0000_H$	$20'3FFF_H$	16 Kbytes	Accessed via EBC
External memory area	$01'0000_H$	$1F'FFFF_H$	< 2 Mbytes	Minus segment 0
SFR area	$00'FE00_H$	$00'FFFF_H$	0.5 Kbyte	–
Dual-Port RAM	$00'F600_H$	$00'FDFF_H$	2 Kbytes	–
Reserved for DPRAM	$00'F200_H$	$00'F5FF_H$	1 Kbyte	–

Table 2-1 XE16x Memory Map (cont'd)

Address Area	Start Loc.	End Loc.	Area Size[1)]	Notes
ESFR area	$00'F000_H$	$00'F1FF_H$	0.5 Kbyte	–
XSFR area	$00'E000_H$	$00'EFFF_H$	4 Kbytes	–
Data SRAM	$00'A000_H$	$00'DFFF_H$	16 Kbytes	–
Reserved for DSRAM	$00'8000_H$	$00'9FFF_H$	8 Kbytes	–
External memory area	$00'0000_H$	$00'7FFF_H$	32 Kbytes	–

1) The areas marked with "<" are slightly smaller than indicated, see column "Notes".

2) The uppermost 4-Kbyte sector of the first Flash segment is reserved for internal use ($C0'F000_H$ to $C0'FFFF_H$).

3) Several pipeline optimizations are not active within the external IO area. This is necessary to control external peripherals properly.

Note: For an overview of the available memory sections for the different derivatives, please refer to ***"Summary of Basic Features" on Page 1-5****.*

External Bus Interface

To meet the needs of designs where more memory is required than is provided on chip, up to 12 Mbytes of external RAM/ROM/Flash or peripherals can be connected to the XE16x microcontroller via its external bus interface.

All of the external memory accesses are performed by a particular on-chip External Bus Controller (EBC). It can be programmed either to Single Chip Mode when no external memory is required, or to an external bus mode with the following possible selections[1]:

- Address Bus Width with a range of 0 … 24-bit
- Data Bus Width 8-bit or 16-bit
- Bus Operation Multiplexed or Demultiplexed

In the demultiplexed bus modes, addresses are output on Port 0 and Port 1 and data is input/output on Port 10 and Port 2. In the multiplexed bus modes both addresses and data use Port 10 and Port 2 for input/output. The high order address (segment) lines use Port 2. The number of active segment address lines is selectable, restricting the external address space to 8 Mbytes … 64 Kbytes. This is required when interface lines are assigned to Port 2.

For up to five address areas the bus mode (multiplexed/demultiplexed), the data bus width (8-bit/16-bit) and even the length of a bus cycle (waitstates, signal delays) can be selected independently. This allows access to a variety of memory and peripheral components directly and with maximum efficiency.

Access to very slow memories or modules with varying access times is supported via a particular ‘Ready’ function. The active level of the control input signal is selectable.

A $\overline{\text{HOLD}}$/$\overline{\text{HLDA}}$ protocol is available for bus arbitration and allows the sharing of external resources with other bus masters.

The external bus timing is related to the rising edge of the reference clock output CLKOUT. The external bus protocol is compatible with that of the standard C166 Family.

For applications which require less than 64 Kbytes of address space, a non-segmented memory model can be selected, where all locations can be addressed by 16 bits. Thus, the upper Port 2 is not needed as an output for the upper address bits (Axx … A16), as is the case when using the segmented memory model.

The EBC also controls accesses to resources connected to the **on-chip LXBus**. The LXBus is an internal representation of the external bus and allows accessing integrated peripherals and modules in the same way as external components.

The MultiCAN module and the USIC modules are connected to and accessed via the LXBus.

[1] Bus modes are switched dynamically if several address windows with different mode settings are used.

2.3 On-Chip Peripheral Blocks

The XE166 Family clearly separates peripherals from the core. This structure permits the maximum number of operations to be performed in parallel and allows peripherals to be added or deleted from family members without modifications to the core. Each functional block processes data independently and communicates information over common buses. Peripherals are controlled by data written to the respective Special Function Registers (SFRs). These SFRs are located within either the standard SFR area ($00'FE00_H$ … $00'FFFF_H$), the extended ESFR area ($00'F000_H$ … $00'F1FF_H$), or within the internal IO area ($00'E000_H$ … $00'EFFF_H$).

These built-in peripherals either allow the CPU to interface with the external world or provide functions on-chip that otherwise would need to be added externally in the respective system.

The XE16x generic peripherals are:

- **General Purpose Timer Unit (GPT1, GPT2)**
- **Watchdog Timer**
- **Capture/Compare Unit (CAPCOM2)**
- Up to Four **Capture/Compare Units CCU6** (CCU60, CCU61, CCU62, CCU63)
- Two 10-bit **Analog/Digital Converters (ADC0, ADC1)**
- **Real Time Clock (RTC)**
- Thirteen/Nine **Parallel Ports** with a total of 118/75 I/O lines

Because the LXBus is the internal representation of the external bus, it does not support bit-addressing. Accesses are executed by the EBC as if it were external accesses. The LXBus connects on-chip peripherals to the CPU:

- Optional **MultiCAN Module** with up to 5 CAN nodes and gateway functionality
- Up to Three **Universal Serial Interface Channel Modules (USIC)**

Each peripheral also contains a set of Special Function Registers (SFRs) which control the functionality of the peripheral and temporarily store intermediate data results. Each peripheral has an associated set of status flags. Individually selected clock signals are generated for each peripheral from binary multiples of the master clock.

Note: For an overview of the available peripherals for the different derivatives, please refer to ***"Summary of Basic Features" on Page 1-5***.

Peripheral Interfaces

The on-chip peripherals generally have two different types of interfaces: an interface to the CPU and an interface to external hardware. Communication between the CPU and peripherals is performed through Special Function Registers (SFRs) and interrupts. The SFRs serve as control/status and data registers for the peripherals. Interrupt requests are generated by the peripherals based on specific events which occur during their operation, such as operation complete, error, etc.

To interface with external hardware, specific pins of the parallel ports are used, when an input or output function has been selected for a peripheral. During this time, the port pins are controlled either by the peripheral (when used as outputs) or by the external hardware which controls the peripheral (when used as inputs). This is called the ‘alternate (input or output) function’ of a port pin, in contrast to its function as a general purpose I/O pin.

Peripheral Timing

Internal operation of the CPU and peripherals is based on the system clock (f_{SYS}). The clock generation unit uses external (e.g. a crystal) or internal clock sources to generate the system clock signal. Peripherals can be disconnected from the clock signal either temporarily to save energy or permanently if they are not used in a specific application. Peripheral SFRs may be accessed by the CPU once per state. When an SFR is written to by software in the same state where it is also to be modified by the peripheral, the software write operation has priority. Further details on peripheral timing are included in the specific sections describing each peripheral.

Programming Hints

- **Access to SFRs:** All SFRs reside in data page 3 of the memory space. The following addressing mechanisms allow access to the SFRs:
  - Indirect or direct addressing with **16-bit (mem) addresses** must guarantee that the used data page pointer (DPP0 … DPP3) selects data page 3.
  - Accesses via the Peripheral Event Controller (**PEC**) use the SRCPx and DSTPx pointers instead of the data page pointers.
  - **Short 8-bit (reg) addresses** to the standard SFR area do not use the data page pointers but directly access the registers within this 512-byte area.
  - **Short 8-bit (reg) addresses** to the extended **ESFR** area require switching to the 512-byte Extended SFR area. This is done via the EXTension instructions EXTR, EXTP(R), EXTS(R).
- **Byte Write Operations** to wordwide SFRs via indirect or direct 16-bit (mem) addressing or byte transfers via the PEC force zeros in the non-addressed byte. Byte write operations via short 8-bit (reg) addressing can access only the low byte of an SFR and force zeros in the high byte. It is therefore recommended, to use the bitfield

instructions (BFLDL and BFLDH) to write to any number of bits in either byte of an SFR without disturbing the non-addressed byte and the unselected bits.

- **Write Operations to Write-Only Bits/Registers** usually modify bits within other registers. In some cases this modification is controlled by state machines. Therefore, the effect of the write operation may not be visible, when the modified register is read immediately after the write access that triggers the modification.
- **Reserved Bits:** Some of the bits which are contained in the XE16x's SFRs are marked as 'Reserved'. User software should never write '1's to reserved bits. These bits are currently not implemented and may be used in future products to invoke new functions. In that case, the active state for those new functions will be '1', and the inactive state will be '0'. Therefore writing only '0's to reserved locations allows portability of the current software to future devices. After read accesses, reserved bits should be ignored or masked out.

Capture/Compare Unit (CAPCOM2)

The CAPCOM units support generation and control of timing sequences on up to 16 channels with a maximum resolution of 1 system clock cycle (8 cycles in staggered mode). The CAPCOM unit is typically used to handle high speed I/O tasks such as pulse and waveform generation, pulse width modulation (PMW), Digital to Analog (D/A) conversion, software timing, or time recording relative to external events.

Two 16-bit timers (T7/T8) with reload registers provide two independent time bases for each capture/compare register.

The input clock for the timers is programmable to several prescaled values of the internal system clock, or may be derived from an overflow/underflow of timer T6 in module GPT2. This provides a wide range of variation for the timer period and resolution and allows precise adjustments to the application specific requirements. In addition, external count inputs for CAPCOM timer T7 allow event scheduling for the capture/compare registers relative to external events.

The capture/compare register array contains 16 dual purpose capture/compare registers, each of which may be individually allocated to either CAPCOM timer T7 or T8 and programmed for capture or compare function.
All registers of each module have each one port pin associated with it which serves as an input pin for triggering the capture function, or as an output pin to indicate the occurrence of a compare event.

Table 2-2 Compare Modes (CAPCOM2)

Compare Modes	Function
Mode 0	Interrupt-only compare mode; several compare interrupts per timer period are possible
Mode 1	Pin toggles on each compare match; several compare events per timer period are possible
Mode 2	Interrupt-only compare mode; only one compare interrupt per timer period is generated
Mode 3	Pin set ‘1’ on match; pin reset ‘0’ on compare timer overflow; only one compare event per timer period is generated
Double Register Mode	Two registers operate on one pin; pin toggles on each compare match; several compare events per timer period are possible
Single Event Mode	Generates single edges or pulses; can be used with any compare mode

When a capture/compare register has been selected for capture mode, the current contents of the allocated timer will be latched (‘captured’) into the capture/compare register in response to an external event at the port pin which is associated with this register. In addition, a specific interrupt request for this capture/compare register is generated. Either a positive, a negative, or both a positive and a negative external signal transition at the pin can be selected as the triggering event.

The contents of all registers which have been selected for one of the five compare modes are continuously compared with the contents of the allocated timers.

When a match occurs between the timer value and the value in a capture/compare register, specific actions will be taken based on the selected compare mode.

Capture/Compare Units CCU6

The CCU6 units support generation and control of timing sequences on up to three 16-bit capture/compare channels plus one independent 16-bit compare channel.
In compare mode, the CCU6 units provide two output signals per channel which have inverted polarity and non-overlapping pulse transitions (deadtime control). The compare channel can generate a single PWM output signal and is further used to modulate the capture/compare output signals.
In capture mode the contents of compare timer T12 is stored in the capture registers upon a signal transition at pins CCx.

The output signals can be generated in edge-aligned or center-aligned PWM mode. They are generated continuously or in single-shot mode.

Compare timers T12 and T13 are free running timers which are clocked by the prescaled system clock.

For motor control applications (brushless DC-drives) both subunits may generate versatile multichannel PWM signals which are basically either controlled by compare timer T12 or by a typical hall sensor pattern at the interrupt inputs (block commutation). The latter mode provides noise filtering for the hall inputs and supports automatic rotational speed measurement.

The trap function offers a fast emergency stop without CPU activity. Triggered by an external signal ($\overline{\text{CTRAP}}$) the outputs are switched to selectable logic levels which can be adapted to the connected power stages.

Note: The number of available CCU6 units and channels depends on the selected device type.

General Purpose Timer Unit (GPT1, GPT2)

The GPT12E unit represents a very flexible multifunctional timer/counter structure which may be used for many different time related tasks such as event timing and counting, pulse width and duty cycle measurements, pulse generation, or pulse multiplication.

The GPT12E unit incorporates five 16-bit timers which are organized in two separate blocks, GPT1 and GPT2. Each timer in each block may operate independently in a number of different modes, or may be concatenated with another timer of the same block.

Each of the three timers T2, T3, T4 of **block GPT1** can be configured individually for one of four basic modes of operation, which are Timer, Gated Timer, Counter, and Incremental Interface Mode. In Timer Mode, the input clock for a timer is derived from the system clock, divided by a programmable prescaler, while Counter Mode allows a timer to be clocked in reference to external events.
Pulse width or duty cycle measurement is supported in Gated Timer Mode, where the operation of a timer is controlled by the ‘gate’ level on an external input pin. For these purposes, each timer has one associated port pin (TxIN) which serves as gate or clock input. The maximum resolution of the timers in block GPT1 is 4 system clock cycles.

The count direction (up/down) for each timer is programmable by software or may additionally be altered dynamically by an external signal on a port pin (TxEUD) to facilitate e.g. position tracking.

In Incremental Interface Mode the GPT1 timers (T2, T3, T4) can be directly connected to the incremental position sensor signals A and B via their respective inputs TxIN and TxEUD. Direction and count signals are internally derived from these two input signals, so the contents of the respective timer Tx corresponds to the sensor position. The third position sensor signal TOP0 can be connected to an interrupt input.

Timer T3 has an output toggle latch (T3OTL) which changes its state on each timer overflow/underflow. The state of this latch may be output on pin T3OUT e.g. for time out monitoring of external hardware components. It may also be used internally to clock timers T2 and T4 for measuring long time periods with high resolution.

In addition to their basic operating modes, timers T2 and T4 may be configured as reload or capture registers for timer T3. When used as capture or reload registers, timers T2 and T4 are stopped. The contents of timer T3 is captured into T2 or T4 in response to a signal at their associated input pins (TxIN). Timer T3 is reloaded with the contents of T2 or T4 triggered either by an external signal or by a selectable state transition of its toggle latch T3OTL. When both T2 and T4 are configured to alternately reload T3 on opposite state transitions of T3OTL with the low and high times of a PWM signal, this signal can be constantly generated without software intervention.

With its maximum resolution of 2 system clock cycles, the **GPT2 block** provides precise event control and time measurement. It includes two timers (T5, T6) and a capture/reload register (CAPREL). Both timers can be clocked with an input clock which is

derived from the CPU clock via a programmable prescaler or with external signals. The count direction (up/down) for each timer is programmable by software or may additionally be altered dynamically by an external signal on a port pin (TxEUD). Concatenation of the timers is supported via the output toggle latch (T6OTL) of timer T6, which changes its state on each timer overflow/underflow.

The state of this latch may be used to clock timer T5, and/or it may be output on pin T6OUT. The overflows/underflows of timer T6 can additionally be used to clock the CAPCOM1/2 timers, and to cause a reload from the CAPREL register.

The CAPREL register may capture the contents of timer T5 based on an external signal transition on the corresponding port pin (CAPIN), and timer T5 may optionally be cleared after the capture procedure. This allows the XE16x to measure absolute time differences or to perform pulse multiplication without software overhead.

The capture trigger (timer T5 to CAPREL) may also be generated upon transitions of GPT1 timer T3's inputs T3IN and/or T3EUD. This is especially advantageous when T3 operates in Incremental Interface Mode.

Real Time Clock (RTC)

The Real Time Clock (RTC) module of the XE16x is directly clocked wih a separate clock signal. Several internal and external clock sources can be selected via register RTCCLKCON. It is, therefore, independent from the selected clock generation mode of the XE16x.

The RTC basically consists of a chain of divider blocks:

- Selectable 32:1 and 8:1 dividers (on - off)
- The reloadable 16-bit timer T14
- The 32-bit RTC timer block (accessible via registers RTCH and RTCL), made of:
  - a reloadable 10-bit timer
  - a reloadable 6-bit timer
  - a reloadable 6-bit timer
  - a reloadable 10-bit timer

All timers count up. Each timer can generate an interrupt request. All requests are combined to a common node request.

Note: The registers associated with the RTC are not affected by an application reset in order to maintain the contents even when intermediate resets are executed.

The RTC module can be used for different purposes:

- System clock to determine the current time and date
- Cyclic time based interrupt, to provide a system time tick independent of CPU frequency and other resources
- 48-bit timer for long term measurements

Analog/Digital Converters (ADC0, ADC1)

For analog signal measurement, two 10-bit A/D converters (ADC0, ADC1) with 16 (or 8) multiplexed input channels including a sample and hold circuit have been integrated on-chip. They use the method of successive approximation. The sample time (for loading the capacitors) and the conversion time are programmable and can thus be adjusted to the external circuitry. The A/D converters can also operate in 8-bit conversion mode, where the conversion time is further reduced.

Several independent conversion result registers, selectable interrupt requests, and highly flexible conversion sequences provide a high degree of programmability to fulfill the requirements of the respective application. Both modules can be synchronized to allow parallel sampling of two input channels.

For applications that require more analog input channels, external analog multiplexers can be controlled automatically.

For applications that require less analog input channels, the remaining channel inputs can be used as digital input port pins.

The A/D converters of the XE16x support two types of request sources which can be triggered by several internal and external events.

- Parallel requests are activated at the same time and then executed in a predefined sequence.
- Queued requests are executed in a user-defined sequence.

In addition, the conversion of a specific channel can be inserted into a running sequence without disturbing this sequence. All requests are arbitrated according to the priority level that has been assigned to them.

Data reduction features, such as limit checking or result accumulation, reduce the number of required CPU accesses and so allow the precise evaluation of analog inputs (high conversion rate) even at low CPU speed.

The Peripheral Event Controller (PEC) may be used to control the A/D converters or to automatically store conversion results into a table in memory for later evaluation, without requiring the overhead of entering and exiting interrupt routines for each data transfer. Therefore, each A/D converter contains 8 result registers which can be concatenated to build a result FIFO. Wait-for-read mode can be enabled for each result register to prevent loss of conversion data.

In order to decouple analog inputs from digital noise and to avoid input trigger noise those pins used for analog input can be disconnected from the digital input stages under software control. This can be selected for each pin separately via registers P5_DIDIS and P15_DIDIS (Port x Digital Input Disable).

The Auto-Power-Down feature of the A/D converters minimizes the power consumption when no conversion is in progress.

Note: The number of available analog channels depends on the selected device type.

Universal Serial Interface Channel Modules (USIC)

Each USIC channel can be individually configured to match the application needs, e.g. the protocol can be selected or changed during run time without the need for a reset. The following protocols are supported:

- **UART** (ASC, asynchronous serial channel)
  - module capability: receiver/transmitter with max. baud rate $f_{sys}/4$
  - application target baud rate range: 1.2 kBaud to 3.5 MBaud
  - number of data bits per data frame 1 to 63
  - MSB or LSB first
- **LIN** Support by HW (low-cost network, baud rate up to 20 kBaud)
  - data transfers based on ASC protocol
  - baud rate detection possible by built-in capture event of baud rate generator
  - checksum generation under SW control for higher flexibility
- **SSC/SPI** (synchronous serial channel with or without slave select lines)
  - module capability: slave mode with max. baud rate f_{sys}
  - module capability: master mode with max. baud rate f_{sys} /2
  - application target baud rate range: 2 kBaud to 10 MBaud
  - number of data bits per data frame 1 to 63, more with explicit stop condition
  - MSB or LSB first
- **IIC** (Inter-IC Bus)
  - application baud rate 100 kBaud to 400 kBaud
  - 7-bit and 10-bit addressing supported
  - full master and slave device capability
- **IIS** (infotainment audio bus)
  - module capability: receiver with max. baud rate f_{SYS}
  - module capability: transmitter with max. baud rate f_{SYS} /2
  - application target baud rate range: up to 26 MBaud

In addition to the flexible choice of the communication protocol, the USIC structure has been designed to reduce the system load (CPU load) allowing efficient data handling. The following aspects have been considered:

- **Data buffer capability**
  The standard buffer capability includes a double word buffer for receive data and a single word buffer for transmit data. This allows longer CPU reaction times (e.g. interrupt latency).
- **Additional FIFO buffer capability**
  In addition to the standard buffer capability, the received data and the data to be transmitted can be buffered in a FIFO buffer structure. The size of the receive and the transmit FIFO buffer can be programmed independently. Depending on the application needs, a total buffer capability of 64 data words can be assigned to the receive and transmit FIFO buffers of a USIC module (the two channels of the USIC module share the 64 data word buffer).

In addition to the FIFO buffer, a bypass mechanism allows the introduction of high-priority data without flushing the FIFO buffer.

- **Transmit control information**
  For each data word to be transmitted, a 5-bit transmit control information has been added to automatically control some transmission parameters, such as word length, frame length, or the slave select control for the SPI protocol. The transmit control information is generated automatically by analyzing the address where the user SW has written the data word to be transmitted (32 input locations = 2^5 = 5 bit transmit control information).
  This feature allows individual handling of each data word, e.g. the transmit control information associated to the data words stored in a transmit FIFO can automatically modify the slave select outputs to select different communication targets (slave devices) without CPU load. Alternatively, it can be used to control the frame length.
- **Flexible frame length control**
  The number of bits to be transferred within a data frame is independent of the data word length and can be handled in two different ways. The first option allows automatic generation of frames up to 63 bits with a known length. The second option supports longer frames (even unlimited length) or frames with a dynamically controlled length.
- **Interrupt capability**
  The events of each USIC channel can be individually routed to one of 4 service request outputs, depending on the application needs. Furthermore, specific start and end of frame indications are supported in addition to protocol-specific events.
- **Flexible interface routing**
  Each USIC channel offers the choice between several possible input and output pins connections for the communications signals. This allows a flexible assignment of USIC signals to pins that can be changed without resetting the device.
- **Input conditioning**
  Each input signal is handled by a programmable input conditioning stage with programmable filtering and synchronization capability.
- **Baud rate generation**
  Each USIC channel contains an own baud rate generator. The baud rate generation can be based either on the internal module clock or on an external frequency input. This structure allows data transfers with a frequency that can not be generated internally, e.g. to synchronize several communication partners.
- **Transfer trigger capability**
  In master mode, data transfers can be triggered events generated outside the USIC module, e.g. at an input pin or a timer unit (transmit data validation). This feature allows time base related data transmission.
- **Debugger support**
  The USIC offers specific addresses to read out received data without interaction with the FIFO buffer mechanism. This feature allows debugger accesses without the risk of a corrupted receive data sequence.

To reach a desired baud rate, two criteria have to be respected, the module capability and the application environment. The module capability is defined with respect to the module's input clock frequency f_{sys}, being the base for the module operation. Although the module's capability being much higher (depending on the module clock and the number of module clock cycles needed to represent a data bit), the reachable baud rate is generally limited by the application environment. In most cases, the application environment limits the maximum reachable baud rate due to driver delays, signal propagation times, or due to EMI reasons.

Note: Depending on the selected additional functions (such as digital filters, input synchronization stages, sample point adjustment, data structure, etc.), the maximum reachable baud rate can be limited. Please also take care about additional delays, such as (internal or external) propagation delays and driver delays (e.g. for collision detection in ASC mode, for IIC, etc.).

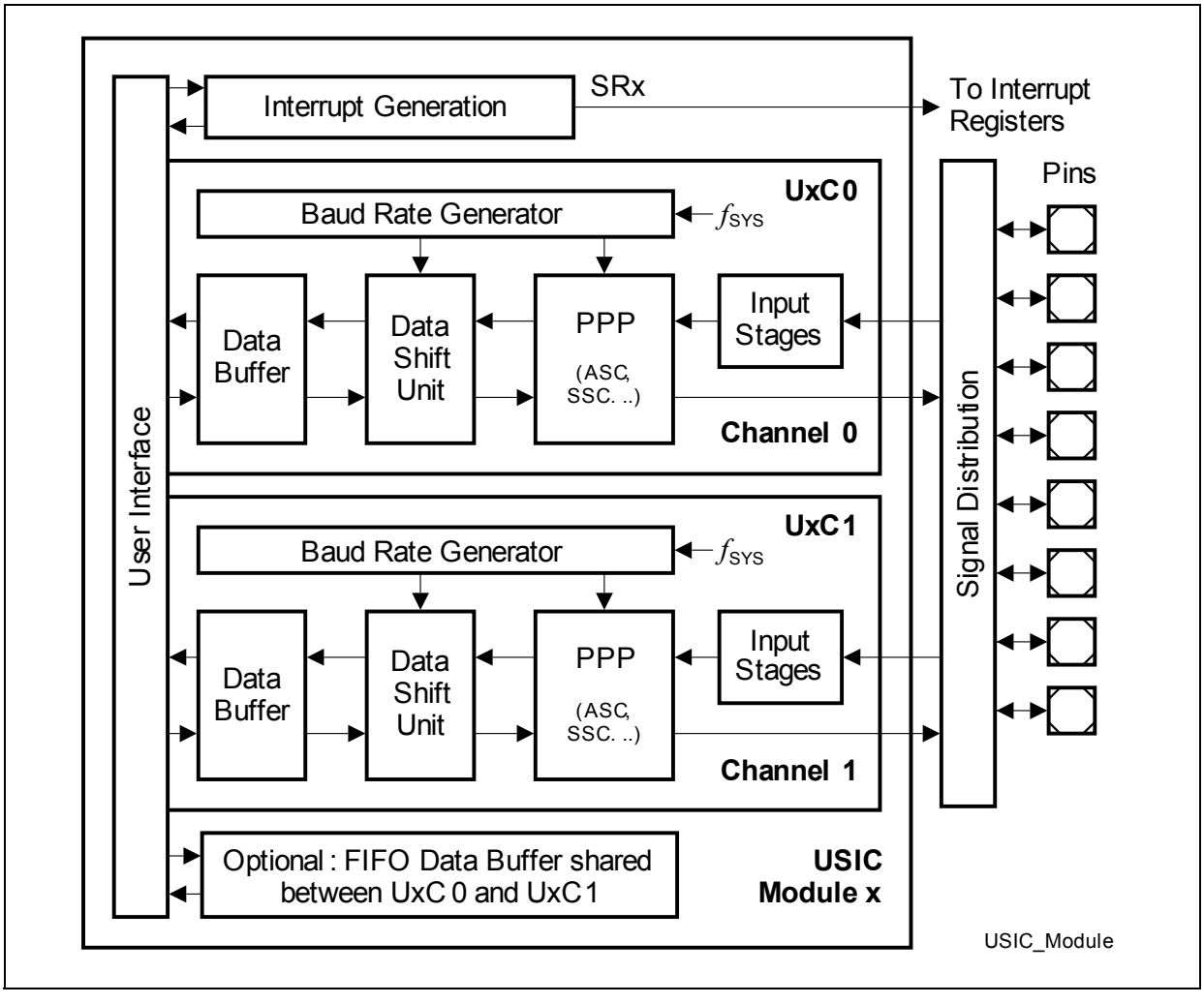


Figure 2-3 USIC Channel Structure

The USIC module contains two independent communication channels, with structure shown in **Figure 2-3**.
The data shift unit and the data buffering of each channel support full-duplex data transfers. The protocol-specific actions are handled by protocol pre-processors (PPP). In order to simplify data handling, an additional FIFO data buffer is optionally available for each USIC module to store transmit and receive data for each channel. This FIFO data buffer is not necessarily available in all devices (please refer to USIC implementation chapter for details).
Due to the independent channel control and baud rate generation, the communication protocol, baud rate and the data format can be independently programmed for each communication channel.

MultiCAN Module

The MultiCAN module contains up to five independently operating CAN nodes with Full-CAN functionality which 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.

Note: The number of available CAN nodes depends on the selected device type.

All CAN nodes share a common set of up to 128 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 its own message object list, and it transmits only messages belonging to this message object list. A powerful, command-driven list controller performs all message object list operations.

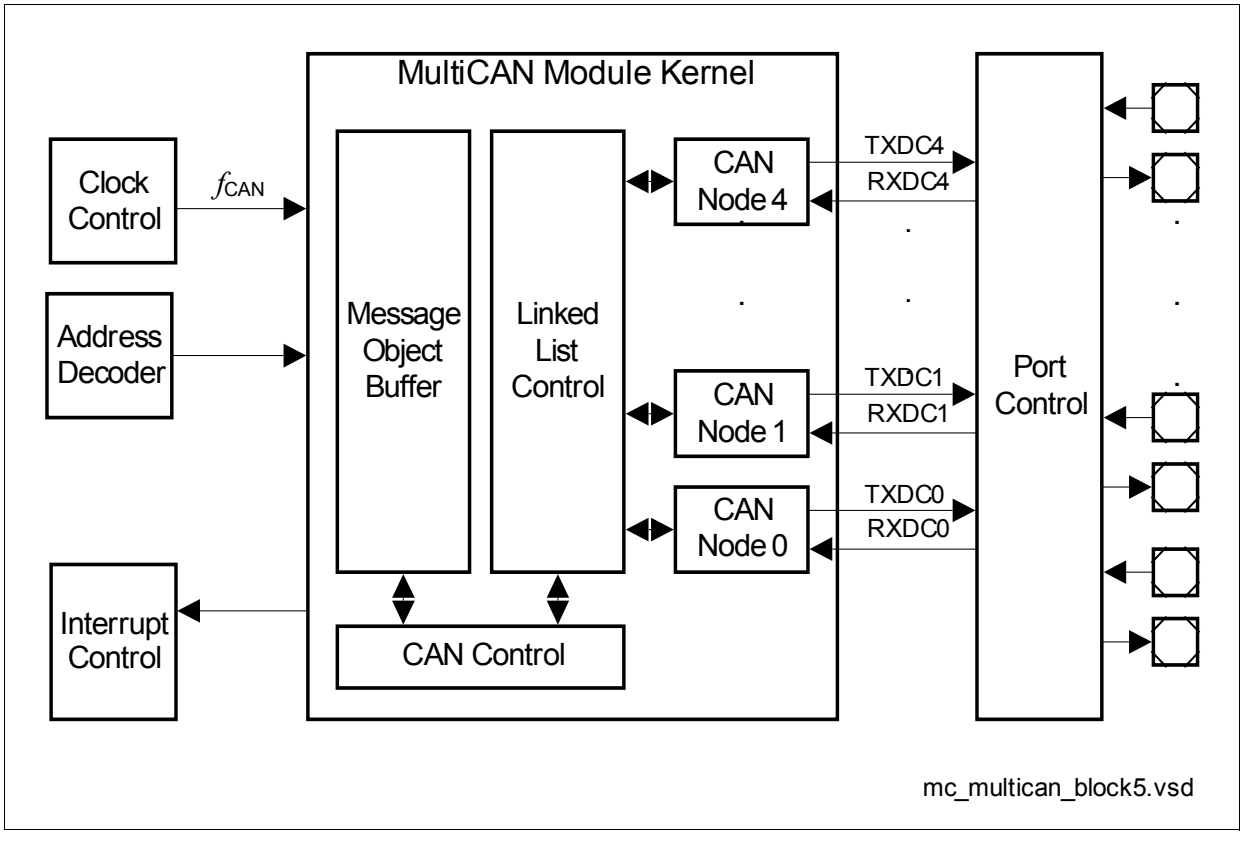


Figure 2-4 Block Diagram of the MultiCAN Module

MultiCAN Features

- CAN functionality conforms to CAN specification V2.0 B active for each CAN node (compliant to ISO 11898)
- Up to Five independent CAN nodes
- Up to 128 independent message objects (shared by the CAN nodes)
- Dedicated control registers for each CAN node
- Data transfer rate up to 1 Mbit/s, individually programmable for each node
- Flexible and powerful message transfer control and error handling capabilities
- Full-CAN functionality for message objects:
  - Can be assigned to one of the CAN nodes
  - Configurable as transmit or receive objects, or as message buffer FIFO
  - Handle 11-bit or 29-bit identifiers with programmable acceptance mask for filtering
  - Remote Monitoring Mode, and frame counter for monitoring
- Automatic Gateway Mode support
- 16 individually programmable interrupt nodes
- Analyzer mode for CAN bus monitoring

Watchdog Timer

The Watchdog Timer represents one of the fail-safe mechanisms which have been implemented to prevent the controller from malfunctioning for longer periods of time.

The Watchdog Timer is always enabled after a reset of the chip, and can be disabled and enabled at any time by executing instructions DISWDT and ENWDT. Thus, the chip's start-up procedure is always monitored. The software has to be designed to restart the Watchdog Timer before it overflows. If, due to hardware or software related failures, the software fails to do so, the Watchdog Timer overflows and generates a reset request.

The Watchdog Timer is a 16-bit timer, clocked with the system clock divided by 16,384 or 256. The high byte of the Watchdog Timer register can be set to a prespecified reload value (stored in WDTREL) to allow further variation of the monitored time interval. Each time it is serviced by the application software, the high byte of the Watchdog Timer is reloaded and the low byte is cleared.

Thus, time intervals between 3.9 μs and 16.3 s can be monitored (@ 66 MHz).
The default Watchdog Timer interval after reset is 6.5 ms (@ 10 MHz).

Parallel Ports

The XE16x derivatives are available in two different packages:

- In LQFP-144, they provide up to 118 I/O lines which are organized into 11 input/output ports and 2 input ports.
- In LQFP-100, they provide up to 75 I/O lines which are organized into 7 input/output ports and 2 input ports.

All port lines are bit-addressable, and all input/output lines can be individually (bit-wise) configured via port control registers. This configuration selects the direction (input/output), push/pull or open-drain operation, activation of pull devices, and edge characteristics (shape) and driver characteristics (output current) of the port drivers. The I/O ports are true bidirectional ports which are switched to high impedance state when configured as inputs. During the internal reset, all port pins are configured as inputs without pull devices active.

All port lines have programmable alternate input or output functions associated with them. These alternate fucntions can be assigned to various port pins to support the optimal utilization for a given application. For this reason, certain functions appear several times in **Table 2-3**.

All port lines that are not used for these alternate functions may be used as general purpose IO lines.

Table 2-3 Summary of the XE16x's Parallel Ports

Port	Width 144[1)]	Width 100[1)]	Alternate Functions
Port 0	8	8	Address lines, Serial interface lines of USIC1, CAN0, and CAN1, Input/Output lines for CCU61
Port 1	8	8	Address lines, Serial interface lines of USIC1 and USIC2, Input/Output lines for CCU62, OCDS control, interrupts
Port 2	13	13	Address and/or data lines, bus control, Serial interface lines of USIC0, CAN0, and CAN1, Input/Output lines for CCU60, CCU63, and CAPCOM2, Timer control signals, JTAG, interrupts, system clock output
Port 3	8	---	Bus arbitration signals, Serial interface lines of USIC0, USIC2, CAN3, and CAN4

Table 2-3 Summary of the XE16x's Parallel Ports (cont'd)

Port	Width 144[1)]	Width 100[1)]	Alternate Functions
Port 4	8	4	Chip select signals, Serial interface lines of CAN2, Input/Output lines for CAPCOM2, Timer control signals
Port 5	16	11	Analog input channels to ADC0, Input/Output lines for CCU6x, Timer control signals, JTAG, OCDS control, interrupts
Port 6[2)]	4	3	ADC control lines, Serial interface lines of USIC1, Timer control signals, OCDS control
Port 7	5	5	ADC control lines, Serial interface lines of USIC0 and CAN4, Input/Output lines for CCU62, Timer control signals, JTAG, OCDS control,system clock output
Port 8	7	---	Input/Output lines for CCU60, JTAG, OCDS control
Port 9	8	---	Serial interface lines of USIC2, Input/Output lines for CCU60 and CCU63, OCDS control
Port 10	16	16	Address and/or data lines, bus control, Serial interface lines of USIC0, USIC1, CAN2, CAN3, and CAN4, Input/Output lines for CCU60, JTAG, OCDS control
Port 11	6	---	Input/Output lines for CCU63
Port 15	8	5	Analog input channels to ADC1, Timer control signals

1) These columns describe the availability of port pins in the different packages.

2) The drivers of these pins are supplied by V_{DDPA}.

2.4 Clock Generation

The Clock Generation Unit uses a programmable on-chip PLL with multiple prescalers to generate the clock signals for the XE16x with high flexibility. The system clock f_{SYS} is the reference clock signal, which can be output to the external system. The system clock f_{SYS} can be derived from several internal and external clock sources.

The on-chip high-precision oscillator (OSC_HP) can drive an external crystal or accepts an external clock signal. The oscillator clock frequency can be multiplied by the on-chip PLL (by a programmable factor) or can be divided by a programmable prescaler factor.

An internal clock source can provide a clock signal without requiring an external crystal.

The Oscillator Watchdog (OWD) supervises the input clock and enables an emergency clock if the input clock appears as not reliable.

2.5 Power Management

The XE16x can operate within a wide supply voltage range from 3 V to 5 V. The internal core supply voltage is generated via on-chip Embedded Voltage Regulators and is supervised by on-chip Power Validation Circuits.

Two IO power domains help to reduce heat dissipation by supplying the major part of the device with a low voltage (3 V), while still connecting analog 5 V sensor signals to the ADCs (5 V).

The XE16x provides several means to control the power it consumes either at a given time or averaged over a certain timespan. Three mechanisms can be used (partly in parallel):

- **Supply Voltage Management** allows to switch off the supply voltage. This drastically reduces the power consumed because of leakage current, in particular at high temperature. A power-on reset restarts the system.
- **Clock Generation Management** controls the distribution and the frequency of internal and external clock signals. While the clock signals for currently inactive parts of logic are disabled automatically, the user can reduce the XE16x's CPU clock frequency which drastically reduces the consumed power.
  External circuitry can be controlled via the programmable frequency output EXTCLK.
- **Peripheral Management** permits temporary disabling of peripheral modules. Each peripheral can separately be disabled/enabled.

Note: When selecting the supply voltage and the clock source and generation method, the required parameters must be carefully written to the respective bitfields, to avoid unintended intermediate states. Recommended sequences are provided which ensure the intended operation of power supply system and clock system.

2.6 On-Chip Debug Support (OCDS)

The On-Chip Debug Support system provides a broad range of debug and emulation features built into the XE16x. The user software running on the XE16x can thus be debugged within the target system environment.

The OCDS is controlled by an external debugging device via the debug interface, consisting of the IEEE-1149-conforming JTAG port and a break interface. The debugger controls the OCDS via a set of dedicated registers accessible via the JTAG interface. Additionally, the OCDS system can be controlled by the CPU, e.g. by a monitor program. An injection interface allows the execution of OCDS-generated instructions by the CPU.

Multiple breakpoints can be triggered by on-chip hardware, by software, or by an external trigger input. Single stepping is supported as well as the injection of arbitrary instructions and read/write access to the complete internal address space. A breakpoint trigger can be answered with a CPU-halt, a monitor call, a data transfer, or/and the activation of an external signal.

The data transferred at a watchpoint (see above) can be obtained via the JTAG interface or via the external bus interface for increased performance.

The debug interface uses a set of 4 to 6 interface signals (4 JTAG lines, 1 or 2 optional break lines) to communicate with external circuitry.

3 Memory Organization

The memory space of the XE16x is configured in a “Von Neumann” architecture. This means that code and data are accessed within the same linear address space. All of the physically separated memory areas, including internal ROM and Flash, internal RAM, the internal Special Function Register Areas (SFRs and ESFRs), the internal IO area, and external memory are mapped into one common address space.

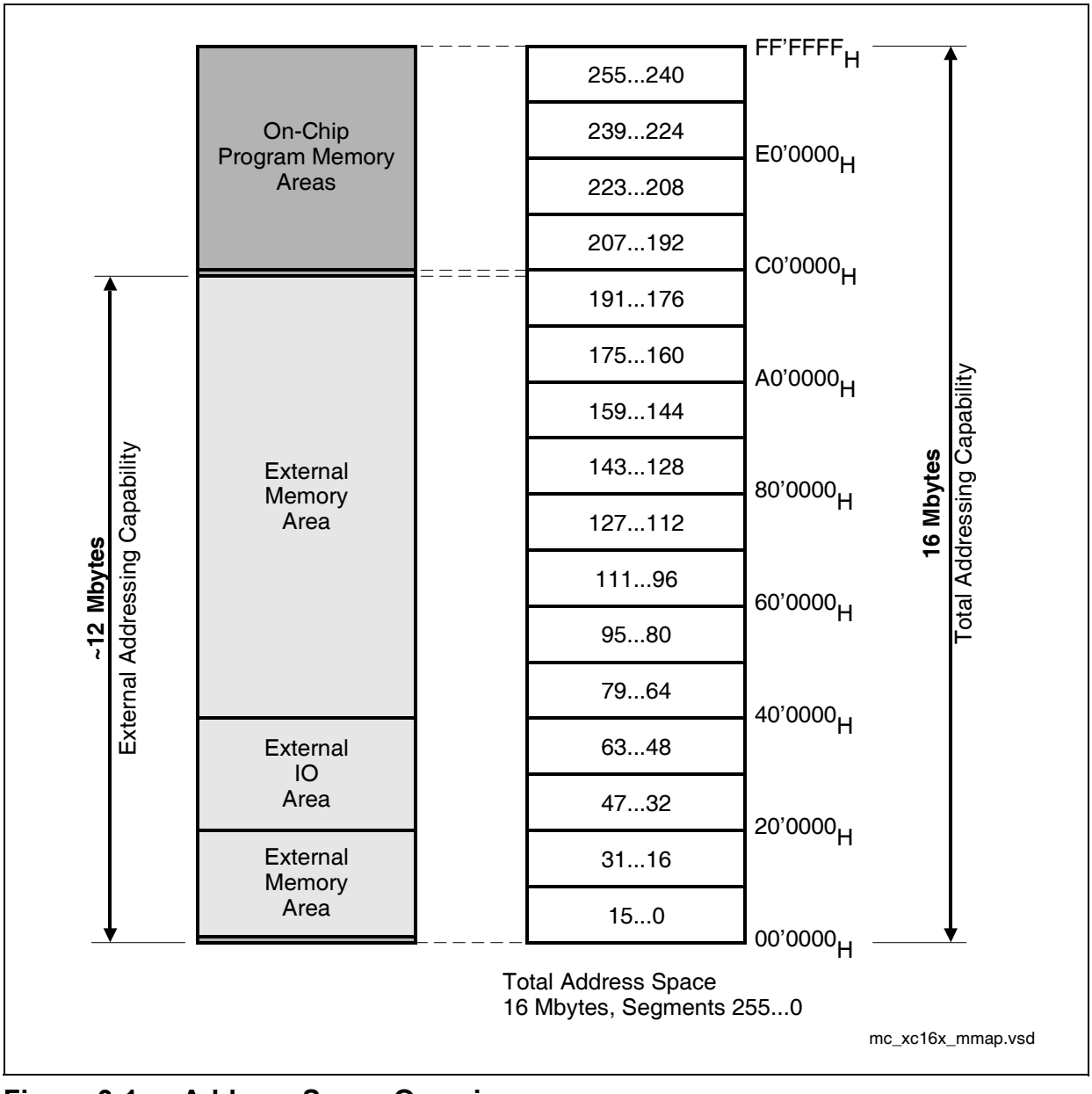


Figure 3-1 Address Space Overview

The XE16x provides a total addressable memory space of 16 Mbytes. This address space is arranged as 256 segments of 64 Kbytes each, and each segment is again subdivided into four data pages of 16 Kbytes each (see **Figure 3-1**).

Bytes are stored at even or odd byte addresses. Words are stored in ascending memory locations with the low byte at an even byte address being followed by the high byte at the next odd byte address (“little endian”). Double words (code only) are stored in ascending memory locations as two subsequent words. Single bits are always stored in the specified bit position at a word address. Bit position 0 is the least significant bit of the byte at an even byte address, and bit position 15 is the most significant bit of the byte at the next odd byte address. Bit addressing is supported for a part of the Special Function Registers, a part of the internal RAM and for the General Purpose Registers.

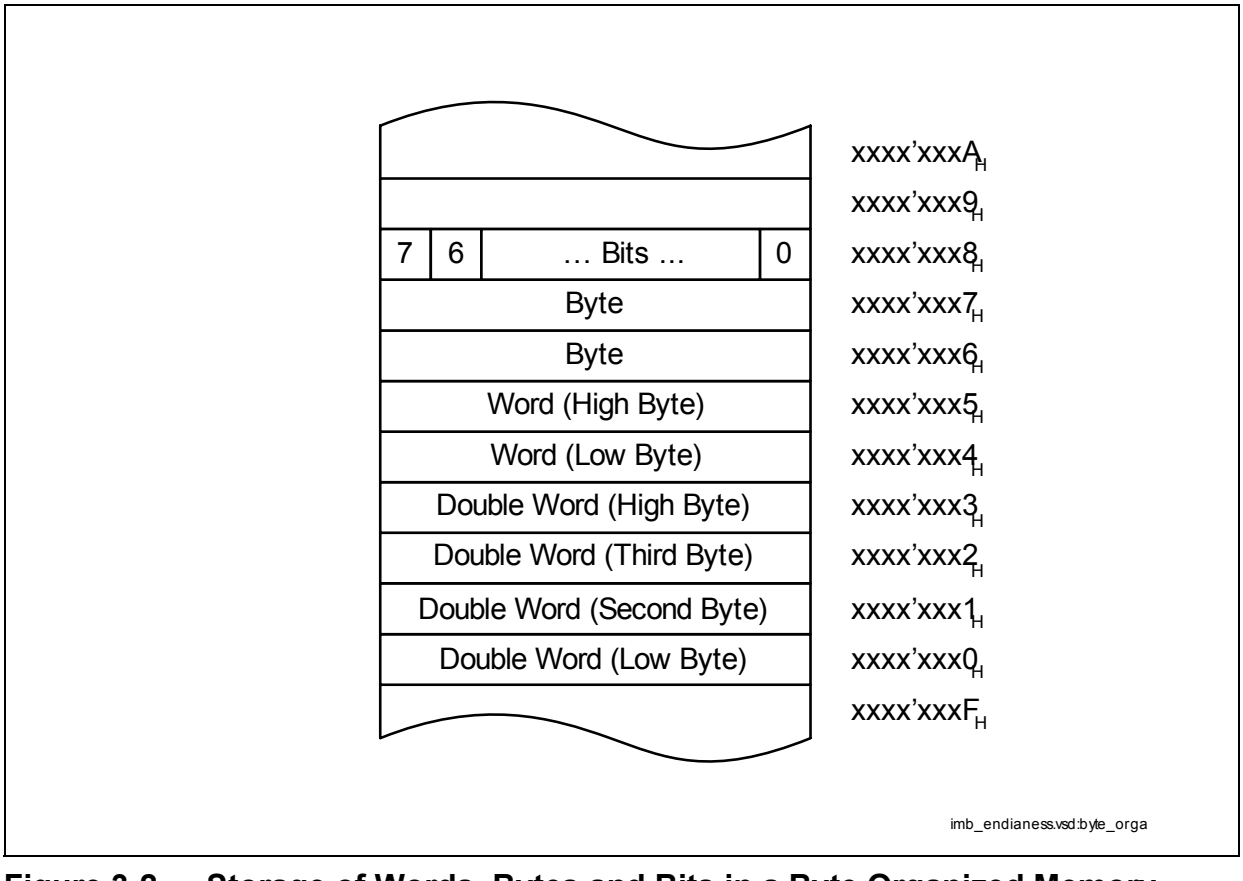


Figure 3-2 Storage of Words, Bytes and Bits in a Byte Organized Memory

Note: Byte units forming a single word or a double word must always be stored within the same physical (internal, external, ROM, RAM) and organizational (page, segment) memory area.

3.1 Address Mapping

All the various memory areas and peripheral registers (see **Table 3-1**) are mapped into one contiguous address space. All sections can be accessed in the same way. The memory map of the XE16x contains some reserved areas, so future derivatives can be enhanced in an upward-compatible fashion.

*Note: **Table 3-1** shows the maximum available memory areas. The actual available memory areas depend on the selected device type.*

Table 3-1 XE16x Memory Map [1)]

Address Area	Start Loc.	End Loc.	Area Size[2)]	Notes
IMB register space	$FF'FF00_H$	$FF'FFFF_H$	256 Bytes	
Reserved (access trap)	$F0'0000_H$	$FF'FEFF_H$	< 1 MByte	Minus IMB registers
Reserved for EPSRAM	$E9'0000_H$	$EF'FFFF_H$	448 KBytes	Mirrors EPSRAM
Emulated PSRAM	$E8'0000_H$	$E8'FFFF_H$	64 KBytes	With Flash timing
Reserved for PSRAM	$E1'0000_H$	$E7'FFFF_H$	448 KBytes	Mirrors PSRAM
PSRAM	$E0'0000_H$	$E0'FFFF_H$	64 KBytes	Program SRAM
Reserved for Flash	$CC'0000_H$	$DF'FFFF_H$	<1.25 MBytes	
Flash 2	$C8'0000_H$	$CB'FFFF_H$	256 KBytes	
Flash 1	$C4'0000_H$	$C7'FFFF_H$	256 KBytes	
Flash 0	$C0'0000_H$	$C3'FFFF_H$	252 KBytes[3)]	Minus res. seg.
External memory area	$40'0000_H$	$BF'FFFF_H$	8 MBytes	
External IO area[4)]	$20'5800_H$	$3F'FFFF_H$	< 2 MBytes	Minus CAN/USIC
USIC registers	$20'4000_H$	$20'57FF_H$	6 KBytes	Accessed via EBC
MultiCAN registers	$20'0000_H$	$20'3FFF_H$	16 KBytes	Accessed via EBC
External memory area	$01'0000_H$	$1F'FFFF_H$	< 2 MBytes	Minus segment 0
SFR area	$00'FE00_H$	$00'FFFF_H$	0.5 KBytes	
Dualport RAM (DPRAM)	$00'F600_H$	$00'FDFF_H$	2 KBytes	
Reserved for DPRAM	$00'F200_H$	$00'F5FF_H$	1 KBytes	
ESFR area	$00'F000_H$	$00'F1FF_H$	0.5 KBytes	
XSFR area	$00'E000_H$	$00'EFFF_H$	4 KBytes	
Data SRAM (DSRAM)	$00'A000_H$	$00'DFFF_H$	16 KBytes	
Reserved for DSRAM	$00'8000_H$	$00'9FFF_H$	8 KBytes	
External memory area	$00'0000_H$	$00'7FFF_H$	32 KBytes	

1) Accesses to the shaded areas are reserved. In devices with external bus interface these accesses generate external bus accesses.

2) The areas marked with “<” are slightly smaller than indicated, see column “Notes”.

3) The uppermost 4-Kbyte sector of the first Flash segment is reserved for internal use (C0’F000$_H$ to C0’FFFF$_H$).

4) Several pipeline optimizations are not active within the external IO area. This is necessary to control external peripherals properly.

3.2 Special Function Register Areas

The Special Function Registers (SFRs) controlling the system and peripheral functions of the XE16x can be accessed via four dedicated address areas:

- 512-byte SFR area (located above the internal RAM: 00'FFFF$_H$ … 00'FE00$_H$).
- 512-byte ESFR area (located below the internal RAM: 00'F1FF$_H$ … 00'F000$_H$).
- 4-Kbyte XSFR area (located below the ESFR area: 00'EFFF$_H$ … 00'E000$_H$).
- 256-byte IMB SFR area (located in: FF'FF00$_H$ … FF'FFFF$_H$)[1].

This arrangement provides upward compatibility with the derivatives of the C166 and XC166 families.

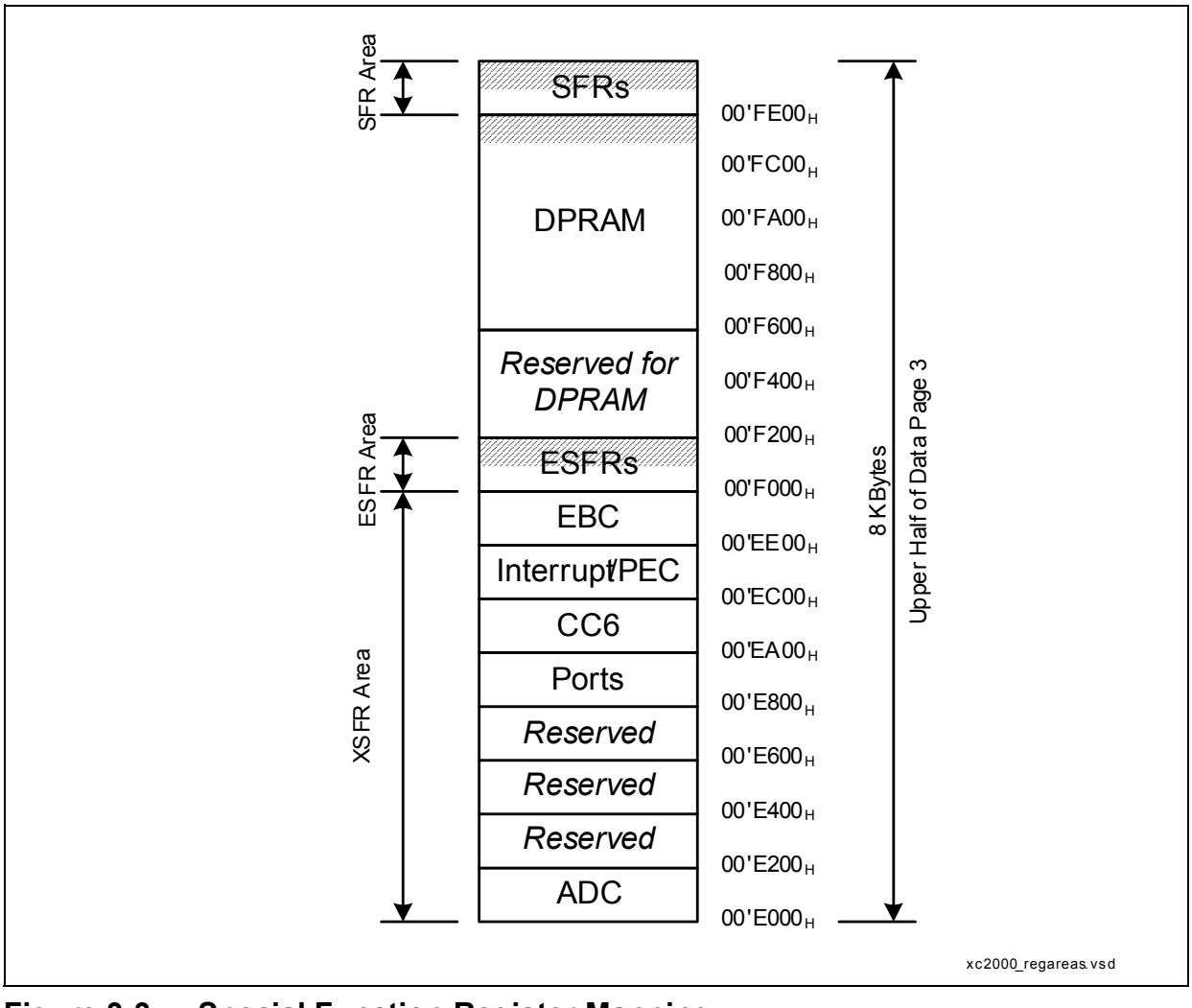


Figure 3-3 Special Function Register Mapping

*Note: The upper 256 bytes of SFR area, ESFR area, and internal RAM are bit-addressable (see hashed blocks in **Figure 3-3**).*

[1] Attention: the IMB SFR area is not recognized by the CPU as special IO area (see **Section 3.6**).

Special Function Registers

The functions of the CPU, the bus interface, the IO ports, and the on-chip peripherals of the XE16x are controlled via a number of Special Function Registers (SFRs).

All Special Function Registers can be addressed via indirect and long 16-bit addressing modes. The (word) SFRs and their respective low bytes in the SFR/ESFR areas can be addressed using an 8-bit offset together with an implicit base address. However, this **does not work** for the respective high bytes!

Note: Writing to any byte of an SFR causes the not addressed complementary byte to be cleared.

The upper half of the SFR-area (00'FFFF$_H$ … 00'FF00$_H$) and the ESFR-area (00'F1FF$_H$ … 00'F100$_H$) is bit-addressable, so the respective control/status bits can be modified directly or checked using bit addressing.

When accessing registers in the ESFR area using 8-bit addresses or direct bit addressing, an Extend Register (EXTR) instruction is required beforehand to switch the short addressing mechanism from the standard SFR area to the Extended SFR area. This is not required for 16-bit and indirect addresses. The GPRs R15 … R0 are duplicated, i.e. they are accessible within both register blocks via short 2-, 4-, or 8-bit addresses without switching.

```
ESFR_SWITCH_EXAMPLE:
EXTR  #4                  ;Switch to ESFR area for next 4 instr.
MOV   STMREL, #data16     ;STMREL uses 8-bit reg addressing
BFLDL STMCON, #mask, #data8 ;Bit addressing for bitfields
BSET  WUCR.CLRTRG         ;Bit addressing for single bits
MOV   T8REL, R1           ;T8REL uses 16-bit mem address,
                          ;R1 is duplicated into the ESFR space
                          ;(EXTR is not required for this access)
;---- ;---------------    ;The scope of the EXTR #4 instruction …
                          ;… ends here!
MOV   T8REL, R1           ;T8REL uses 16-bit mem address,
                          ;R1 is accessed via the SFR space
```

In order to minimize the use of the EXTR instructions the ESFR area mostly holds registers which are mainly required for initialization and mode selection. Registers that need to be accessed frequently are allocated to the standard SFR area, wherever possible.

Note: The tools are equipped to monitor accesses to the ESFR area and will automatically insert EXTR instructions, or issue a warning in case of missing or excessive EXTR instructions.

Accesses to registers in the XSFR area use 16-bit addresses and require no specific addressing modes or precautions.

General Purpose Registers

The General Purpose Registers (GPRs) use a block of 16 consecutive words either within the global register bank or within one of the two local register banks. The bit-field BANK in register PSW selects the currently active register bank. The global register bank is mirrored to a section in the DPRAM, the Context Pointer (CP) register determines the base address of the currently active global register bank section. This register bank may consist of up to 16 Word-GPRs (R0, R1, … R15) and/or of up to 16 byte-GPRs (RL0,RH0, … RL7, RH7). The sixteen byte-GPRs are mapped onto the first eight Word GPRs (see **Table 3-2**).

In contrast to the system stack, a register bank grows from lower towards higher address locations and occupies a maximum space of 32 bytes. The GPRs are accessed via short 2-, 4-, or 8-bit addressing modes using the Context Pointer (CP) register as base address for the global bank (independent of the current DPP register contents). Additionally, each bit in the currently active register bank can be accessed individually.

Table 3-2 Mapping of General Purpose Registers to DPRAM Addresses

DPRAM Address	High Byte Registers	Low Byte Registers	Word Registers
<CP> + $1E_H$	–	–	R15
<CP> + $1C_H$	–	–	R14
<CP> + $1A_H$	–	–	R13
<CP> + 18_H	–	–	R12
<CP> + 16_H	–	–	R11
<CP> + 14_H	–	–	R10
<CP> + 12_H	–	–	R9
<CP> + 10_H	–	–	R8
<CP> + $0E_H$	RH7	RL7	R7
<CP> + $0C_H$	RH6	RL6	R6
<CP> + $0A_H$	RH5	RL5	R5
<CP> + 08_H	RH4	RL4	R4
<CP> + 06_H	RH3	RL3	R3
<CP> + 04_H	RH2	RL2	R2
<CP> + 02_H	RH1	RL1	R1
<CP> + 00_H	RH0	RL0	R0

The XE16x supports fast register bank (context) switching. Multiple global register banks can physically exist within the DPRAM at the same time. Only the global register bank

selected by the Context Pointer register (CP) is active at a given time, however. Selecting a new active global register bank is simply done by updating the CP register. A particular Switch Context (SCXT) instruction performs register bank switching by automatically saving the previous context and loading the new context. The number of implemented register banks (arbitrary sizes) is limited only by the size of the available DPRAM.

Note: The local GPR banks are not memory mapped and the GPRs cannot be accessed using a long or indirect memory address.

PEC Source and Destination Pointers

The source and destination address pointers for data transfers on the PEC channels are located in the XSFR area.

Each channel uses a pair of pointers stored in two subsequent word locations with the source pointer (SRCPx) on the lower and the destination pointer (DSTPx) on the higher word address (x = 7 … 0). An additional segment register stores the associated source and destination segments, so PEC transfers can move data from/to any location within the complete addressing range.

Whenever a PEC data transfer is performed, the pair of source and destination pointers (selected by the specified PEC channel number) accesses the locations referred to by these pointers independently of the current DPP register contents.

If a PEC channel is not used, the corresponding pointer locations can be used for other purposes.

Note: Writing to any byte of the PEC pointers causes the not addressed complementary byte to be cleared.

3.3 Data Memory Areas

The XE16x provides two on-chip RAM areas exclusively for data storage:

- The **Dual Port RAM (DPRAM)** can be used for global register banks (GPRs), system stack, storage of variables and other data, in particular for MAC operands.
- The **Data SRAM (DSRAM)** can be used for system stack (recommended), storage of variables and other data.

*Note: Data can also be stored in the PSRAM (see **Section 3.10**). However, both data memory areas provide the fastest access.*

Depending on the device additional on-chip memory areas may exist with the special purpose to retain data while the system power domain is switched off. The XE16x contains:

- The **Marker Memory (MKMEM)**.

Dual-Port RAM (DPRAM)

The XE16x provides 2 Kbytes of DPRAM ($00'F600_H$ … $00'FDFF_H$). Any word or byte data in the DPRAM can be accessed via indirect or long 16-bit addressing modes, if the selected DPP register points to data page 3. Any word data access is made on an even byte address. The highest possible word data storage location in the DPRAM is $00'FDFE_H$.

For PEC data transfers, the DPRAM can be accessed independent of the contents of the DPP registers via the PEC source and destination pointers.

The upper 256 bytes of the DPRAM ($00'FD00_H$ through $00'FDFF_H$) are provided for single bit storage, and thus they are bit addressable.

Note: Code cannot be executed out of the DPRAM.

An area of 3 Kbytes is dedicated to DPRAM ($00'F200_H$ … $00'FDFF_H$). The locations without implemented DPRAM are reserved.

Data SRAM (DSRAM)

The XE16x provides 16 Kbytes of DSRAM ($00'A000_H$ … $00'DFFF_H$). Any word or byte data in the DSRAM can be accessed via indirect or long 16-bit addressing modes, if the selected DPP register points to data page 3 (for the range $00'C000_H$ … $00'DFFF_H$) or to data page 2 (for the range $00'A000_H$ … $00'BFFF_H$). Any word data access is made on an even byte address. The highest possible word data storage location in the DSRAM is $00'DFFE_H$.

For PEC data transfers, the DSRAM can be accessed independent of the contents of the DPP registers via the PEC source and destination pointers.

Note: Code cannot be executed out of the DSRAM.

An area of 24 Kbytes is dedicated to DSRAM ($00'8000_H$ … $00'DFFF_H$). The locations without implemented DSRAM are reserved.

Marker Memory (MKMEM)

The MKMEM provides 4 bytes of memory supplied by the wake-up power domain. Its purpose is the same as the SBRAM.

The MKEM consists of 2 16-bit SFRs that are accessible as all other SFRs. Details are described in **Section 3.11**.

Note: Code cannot be executed out of the MKMEM.

3.4 Program Memory Areas

The XE16x provides two on-chip program memory areas for code/data storage:

- The **Program Flash/ROM** stores code and constant data. Flash memory is (re-) programmed by the application software or flash loaders, ROM is mask-programmed in the factory.
- The **Program SRAM (PSRAM)** stores temporary code sequences and other data. For example higher level boot loader software can be written to the PSRAM and then be executed to program the on-chip Flash memory.

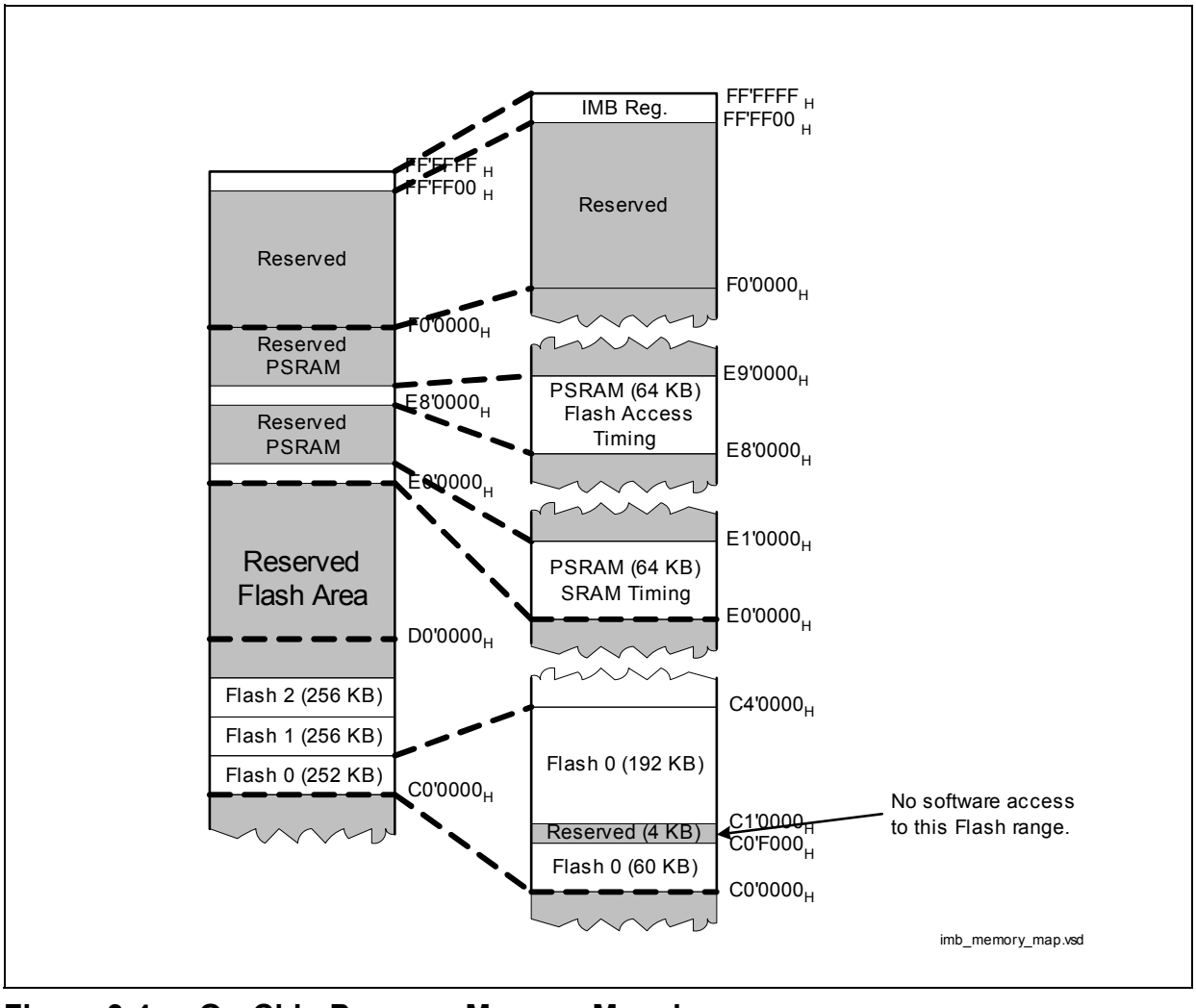


Figure 3-4 On-Chip Program Memory Mapping

3.4.1 Program/Data SRAM (PSRAM)

The XE16x provides 64 Kbytes of PSRAM ($E0'0000_H$ … $E0'FFFF_H$). The PSRAM provides fast code execution without initial delays. Therefore, it supports non-sequential code execution, for example via the interrupt vector table.

Any word or byte data in the PSRAM can be accessed via indirect or long 16-bit addressing modes, if the selected DPP register points to one of its data pages 896 – 899. Any word data access is made on an even byte address. The highest possible word data storage location in the PSRAM is $E0'FFFE_H$.

For PEC data transfers, the PSRAM can be accessed independent of the contents of the DPP registers via the PEC source and destination pointers.

Any data can be stored in the PSRAM. Because the PSRAM is optimized for code fetches, however, data accesses to the data memories provide higher performance.

Note: The PSRAM is not bit-addressable.

Note: The upper 256 Bytes of the PSRAM may be altered during the initialization phase after a reset. This area, therefore, should not store data to be preserved beyond a reset.
Also, during bootstrap loader operation, the serially received data is stored in the PSRAM starting at location $E0'0000_H$.

An area of 512 Kbytes is dedicated to PSRAM ($E0'0000_H$ … $F7'FFFF_H$). The locations without implemented PSRAM are reserved.

Flash Emulation

During code development the PSRAM will often be used for storing code or data that the production chip will later contain in the flash memory. In order to ensure similar execution time the PSRAM supports a second access path in the range $E8'0000_H$ … $EF'FFFF_H$ with timing parameters that correspond to Flash timing. The number of wait-cycles is determined by the flash access timing configuration (see **IMB_IMBCTRL**.WSFLASH). Writes are always performed without wait-cycles.

This flash access timing imitation is nearly cycle accurate because the same read logic as for reading the flash memory is used[1]. Discrepancies might occur if the software uses the PSRAM for flash emulation and directly as PSRAM. During emulation access conflicts can cause a slightly different timing as in the product chip where these conflicts do not occur.

Another source of timing differences can be access conflicts at the flash modules in the product chip. Data reads and instruction fetches that target different flash modules can

[1] The dual use of the flash read logic might cause unexpected behavior: while the IMB Core is busy with updating the protection configuration (after startup or after changing the security pages) read accesses to the flash emulation range of the PSRAM are blocked because Flash data reads would be blocked also.

be executed concurrently whereas if they target the same flash module they are executed sequentially with the data access as first. In the flash emulation this type of conflict can not occur. The data and the instruction access will both incur the defined number of wait-cycles (as if they would target different flash modules) and if they collide at the PSRAM interface the instruction fetch will see an additional wait-cycle.

Data Integrity

The PSRAM contains its own error control. Details are described in the SCU chapter.

Write Protection

As the PSRAM is often used to store timing critical code or constant data it is supplied with a write protection. After storing critical data in the PSRAM the register field **IMB_IMBCTRH**.PSPROT can be used to split the PSRAM into a read-only and a writable part. Write accesses to the read-only part are blocked and a trap can be activated.

3.4.2 Non-Volatile Program Memory (Flash)

The XE16x provides up to 764 Kbytes of program Flash ($C0'0000_H$ … $CB'FFFF_H$). Code and data fetches are always 64-bit aligned, using byte select lines for word and byte data.

Any word or byte data in the program memory can be accessed via indirect or long 16-bit addressing modes, if the selected DPP register points to one of the respective data pages. Any word data access is made on an even byte address. The highest possible word data storage location in the program memory is $CB'FFFE_H$.

For PEC data transfers, the program memory can be accessed independent of the contents of the DPP registers via the PEC source and destination pointers.

Note: The program memory is not bit-addressable.

An area of 2 Mbytes is dedicated to program memory ($C0'0000_H$ … $DF'FFFF_H$). The locations without implemented program memory are reserved.

A more detailed description can be found in **“Embedded Flash Memory” on Page 3-18**.

3.5 System Stack

The system stack may be defined anywhere within the XE16x's memory areas (including external memory).

For all system stack operations the respective stack memory is accessed via a 24-bit stack pointer. The Stack Pointer (SP) register provides the lower 16 bits of the stack pointer (stack pointer offset), the Stack Pointer Segment (SPSEG) register adds the upper 8 bits of the stack pointer (stack segment). The system stack grows downward from higher towards lower locations as it is filled. Only word accesses are supported to the system stack.

Register SP is decremented before data is pushed on the system stack, and incremented after data has been pulled from the system stack. Only word accesses are supported to the system stack.

By using register SP for stack operations, the size of the system stack is limited to 64 KBytes. The stack must be located in the segment defined by register SPSEG.

The stack pointer points to the latest system stack entry, rather than to the next available system stack address.

A stack overflow (STKOV) register and a stack underflow (STKUN) register are provided to control the lower and upper limits of the selected stack area. These two stack boundary registers can be used both for protection against data corruption.

For best performance it is recommended to locate the stack to the DPRAM or to the DSRAM. Using the DPRAM may conflict with register banks or MAC operands.

3.6 IO Areas

The following areas of the XE16x's address space are marked as IO area:

- The **external IO area** is provided for external peripherals (or memories) and also comprises the on-chip LXBus-peripherals, such as the CAN or USIC modules. It is located from $20'0000_H$ to $3F'FFFF_H$ (2 Mbytes).
- The **internal IO area** provides access to the internal peripherals and is split into three blocks:
  - The SFR area, located from $00'FE00_H$ to $00'FFFF_H$ (512 bytes).
  - The ESFR area, located from $00'F000_H$ to $00'F1FF_H$ (512 bytes).
  - The XSFR area, located from $00'E000_H$ to $00'EFFF_H$ (4 Kbytes).

Note: The external IO area supports real byte accesses. The internal IO area does not support real byte transfers, the complementary byte is cleared when writing to a byte location.

The IO areas have special properties, because peripheral modules must be controlled in a different way than memories:

- Accesses are not buffered and cached, the write back buffers and caches are not used to store IO read and write accesses.
- Speculative reads are not executed, but delayed until all speculations are solved (e.g. pre-fetching after conditional branches).
- Data forwarding is disabled, an IO read access is delayed until all IO writes pending in the pipeline are executed, because peripherals can change their internal state after a write access.

3.7 External Memory Space

The XE16x is capable of using an address space of up to 16 Mbytes. Only parts of this address space are occupied by internal memory areas or are reserved. A total area of approximately 12 Mbytes references external memory locations. This external memory is accessed via the XE16x's external bus interface.

Selectable memory bank sizes are supported: The maximum size of a bank in the external memory space depends on the number of activated address bits. It can vary from 64 Kbytes (with A15 … A0 activated) to 12 Mbytes (with A23 … A0 activated). The logical size of a memory bank and its location in the address space is defined by programming the respective address window. It can vary from 4 Kbytes to 12 Mbytes.

- Non-segmented mode:
  – 64 Kbytes with A15 … A0 on PORT0 or PORT1.
- 1-bit segmented mode:
  – 128 Kbytes with A16 on Port 4
  – and A15 … A0 on PORT0 or PORT1.
- 2-bit … 7-bit segmented mode:
  – with Ax … A16 on Port 4
  – and A15 … A0 on PORT0 or PORT1.
- 8-bit segmented mode:
  – 12 Mbytes with A23 … A16 on Port 4
  – and A15 … A0 on PORT0 or PORT1.

Each bank can be directly addressed via the address bus, while the programmable chip select signals can be used to select various memory banks.

The XE16x also supports **four different bus types**:

- Multiplexed 16-bit Bus with address and data on PORT0 (default after Reset).
- Multiplexed 8-bit Bus with address and data on PORT0/P0L.
- Demultiplexed 16-bit Bus with address on PORT1 and data on PORT0.
- Demultiplexed 8-bit Bus with address on PORT1 and data on P0L.

Memory model and bus mode are preselected during reset by pin $\overline{EA}$ and PORT0 pins. For further details about the external bus configuration and control please refer to Chapter XX (The External Bus Controller).

External word and byte data can only be accessed via indirect or long 16-bit addressing modes using one of the four DPP registers. There is no short addressing mode for external operands. Any word data access is made to an even byte address.

For PEC data transfers the external memory can be accessed independent of the contents of the DPP registers via the PEC source and destination pointers.

Note: The external memory is not bit addressable.

3.8 Crossing Memory Boundaries

The address space of the XE16x is implicitly divided into equally sized blocks of different granularity and into logical memory areas. Crossing the boundaries between these blocks (code or data) or areas requires special attention to ensure that the controller executes the desired operations.

Memory Areas are partitions of the address space assigned to different kinds of memory (if provided at all). These memory areas are the SFR areas, the on-chip program or data RAM areas, the on-chip ROM/Flash (if available), the on-chip LXBus-peripherals (if integrated), and the external memory.

Accessing subsequent data locations which belong to different memory areas is no problem. However, when executing code, the different memory areas must be switched explicitly via branch instructions. Sequential boundary crossing is not supported and leads to erroneous results.

Note: Changing from the external memory area to the on-chip RAM area takes place within segment 0.

Segments are contiguous blocks of 64 Kbytes each. They are referenced via the Code Segment Pointer CSP for code fetches and via an explicit segment number for data accesses overriding the standard DPP scheme.

During code fetching, segments are not changed automatically, but rather must be switched explicitly. The instructions JMPS, CALLS and RETS will do this.

In larger sequential programs, make sure that the highest used code location of a segment contains an unconditional branch instruction to the respective following segment to prevent the pre-fetcher from trying to leave the current segment.

Data Pages are contiguous blocks of 16 Kbytes each. They are referenced via the data page pointers DPP3 … DPP0 and via an explicit data page number for data accesses overriding the standard DPP scheme. Each DPP register can select one of the possible 1024 data pages. The DPP register which is used for the current access is selected via the two upper bits of the 16-bit data address. Therefore, subsequent 16-bit data addresses which cross the 16-Kbytes data page boundaries will use different data page pointers, while the physical locations need not be subsequent within memory.

3.9 Embedded Flash Memory

This chapter describes the embedded flash memory of the XE16x:

- **Section 3.9.1** defines the flash specific nomenclature and the structure of the flash memory.
- **Section 3.9.2** describes the operating modes.
- **Section 3.9.3** contains all operations.
- **Section 3.9.4** gives the details of operating sequences.
- The three sections **Section 3.9.5**, **Section 3.9.6** and **Section 3.9.7** look more into depth of maintaining data integrity and protection issues.
- **Section 3.9.8** discusses Flash EEPROM emulation.
- **Section 3.9.9** describes interrupt generation by the flash memory.

The **Chapter 3.10** describes how the flash memory is embedded into the memory architecture of the XE16x and lists all SFRs that affect its behavior.

3.9.1 Definitions

This section defines the nomenclature and some abbreviations as a base for the rest of the document. The used flash memory is a non-volatile memory (“**NVM**”) based on a floating gate one-transistor cell. It is called “non-volatile” because the memory content is kept when the memory power supply is shut off.

Logical and Physical States

Flash memory content can not be changed directly as in SRAMs. Changing data is a complicated process with a typically much longer duration than reading.

- **Erasing**: The erased state of a cell is logical 0. Forcing an flash cell to this state is called “erasing”. Erasing is possible with a minimum granularity of one page (see below). A device is delivered with completely erased flash memory.
- **Programming**: The programmed state of a cell is logical 1. Changing an erased cell to this state is called “programming”. A page must only be programmed once and has to be erased before it can be programmed again.

The above listed processes have certain limitations:

- **Retention**: This is the time during which the data of a flash cell can be read reliably. The retention time is a statistical figure that depends on the operating conditions of the flash array (temperature profile) and the accesses to the flash array. With an increasing number of program/erase cycles (see endurance) the retention is lowered. Drain and gate disturbs decrease data retention as well.
- **Endurance**: As described above the data retention is reduced with an increasing number of program/erase cycles. A flash cell incurs one cycle whenever its page or sector is erased. This number is called “endurance”. As said for the retention it is a statistical figure that depends on operating conditions and the use of the flash cells and not to forget on the required quality level.

- **Drain Disturb**: Because of using a so called “one-transistor” flash cell each program access disturbs all pages of the same sector slightly. Over long these “drain disturbs” make 0 and 1 values indistinguishable and thus provoke read errors. This effect is again interrelated with the retention. A cell that incurred a high number of drain disturbs will have a lower retention. The physical sectors of the flash array are isolated from each other. So pages of a different sector do not incur a drain disturb. This effect must be therefor considered when the page erase feature is used.

The durations of programming and erasing as well as the limits for endurance, retention and drain disturbs are documented in the data sheet.

Attention: No means exist in the device that prevent the application from violating these limitation.

Array Structure

The flash memory is hierarchically structured:

- **Block**: A block consists of 128 user data bits (i.e. 16 bytes) and 9 ECC bits. One read access delivers one block.
- **Page**: A page consists of 8 blocks (i.e. 128 bytes). Programming changes always complete pages.
- **Sector**: A sector consists of 32 pages (i.e. 4096 bytes). The pages of one sector are affected by drain disturb as described above. The pages of different sectors are isolated from each other.
- **Array**: Each array has in the XE16x 64 sectors[1)]. Usually when referring to an “array” this contains as well all accompanying logic as assembly buffer, high voltage logic and the digital logic that allows to operate them in parallel.
- **Memory**: The complete flash memory of the XE16x consists of 3 flash arrays.

This structure is visualized in **Figure 3-5**.

1) In the Flash0 one sector is reserved for device internal purposes. It is not accessible by software.

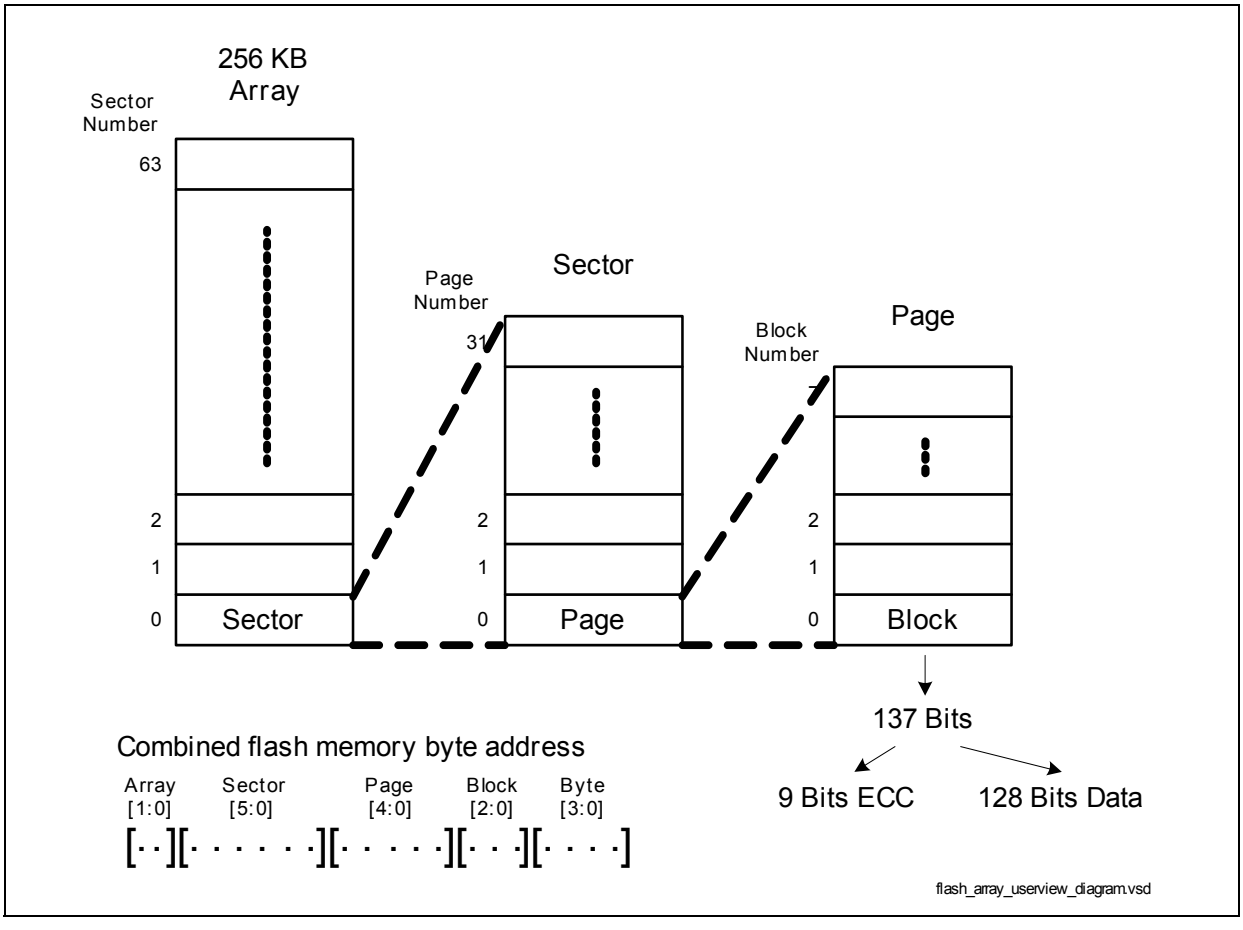


Figure 3-5 Flash Structure

3.9.2 Operating Modes

The IMB and the flash memory and each flash module have certain modes of operation. Some modes define clocking and power supply and the operating state of the analog logic as oscillators and voltage pumps. Overall system modes (e.g. startup mode) influence the behavior or the flash memory as well.

Other modes define the functional behavior. These will be discussed here.

3.9.2.1 Standard Read Mode

After reset and after performing a clean startup the flash memory with all its modules is in “standard read mode”. In this mode it behaves as an on-chip ROM. This mode is entered:

- After reset when the complete start-up has been performed.
- After completion of a longer lasting command like “erase” or “program” which is acknowledged by clearing the “busy” flag.
- Immediately after each other command execution.

- In case of detecting an execution error like attempting to write to a write protected range, sending a wrong password, after all sequence errors.

For the long lasting commands the read mode stays active until the last command of the sequence is received and the operation is started.

3.9.2.2 Command Mode

After receiving the last command of a command sequence the addressed flash module (not the whole flash memory!) is placed into command mode. For most commands this will not be noticed by the user as the command executes immediately and afterwards the flash module is placed again into read mode. For the long lasting commands the flash module stays in command mode for several milliseconds. This is reported by setting the corresponding “busy” flag. The data of a busy flash module cannot be read. New command sequences are not accepted (even if they target different flash modules) and cause a sequence error until the running operation has finished.

Read accesses to busy flash modules stall the CPU until the read mode is entered again. A stalled CPU responds only to the reset. As no interrupts can be handled this state must be avoided. Nevertheless this feature can be used to execute code from a flash module that erases or programs data in the same flash module.

The IMB Core is limited to control only one running operation. Consequently when one flash module is in command mode no other commands to either modules are accepted but the other modules stay readable.

3.9.2.3 Page Mode

The page mode is entered with the “**Enter Page Mode**” command. Please find its description below. A flash module that is in page mode can still be read (so it is concurrently in “read mode”). At a time only one flash module can be in page mode.

When the flash memory is in page mode — i.e. one of the flash modules is in page mode — some command sequences are not allowed. These are all erase sequences and the “change read margin” sequence. These are ignored and a sequence error is reported.

3.9.3 Operations

The flash memory supports the following operations:

- Instruction fetch.
- Data read.
- Command sequences to change data and control the protection.

3.9.3.1 Instruction Fetch from Flash Memory

Instructions are fetched by the PMU in groups of aligned 64 bits. These code requests are forwarded to the flash memory. It needs a varying number of cycles (depending on the system clock frequency) to perform the read access. The number of cycles must be known to the IMB Core because the flash does not signal data availability. The number of wait-cycles is therefore stored in the **IMB_IMBCTRL** register.

One read access to the flash memory delivers 128 data bits and a 9-bit ECC value. The ECC value is used to detect and possibly correct errors. The addressed 64-bit part of the 128-bit chunk is sent to the PMU. The complete 128 data bits and the 9 ECC bits are stored in the IMB Core with their address. If a succeeding fetch request matches this address the data is delivered from the buffer without performing a read access in the flash memory. The delivery from the buffer happens after one cycle. The flash read wait-cycles are not waited.

The stored data are a kind of instruction cache. In order to support self-modifying code (e.g. boot loaders) this cache is invalidated when the corresponding address is written (i.e. erased or programmed).

In addition to this fetch buffer the IMB Core has an additional performance increasing feature — the Linear Code Pre-Fetch. When this feature is enabled with **IMB_IMBCTRL**.DLCPF = 0 the IMB Core fetches autonomously the following instructions while the CPU executes from its own buffers or the fetch buffer. As this feature is fetching only the linear successors (it does not analyze the code stream) it is most effective for code with longer linear sequences. For code with a high density of jumps and calls it can even cause a reduction of performance and should be switched off.

3.9.3.2 Data Reads from Flash Memory

Data reads are issued by the DMU. Data is always requested in 16-bit words. The flash memory delivers for every read request 128 bits plus ECC as described in **“Instruction Fetch from Flash Memory” on Page 3-22**.

The IMB Core has to get all 128 bits to evaluate the ECC data. The requested 16 bits will be delivered to the DMU. All data and ECC bits are kept in the data register and their address is kept in the address register. For all following data reads the address is compared with the address register and in case of a match the data is delivered after one cycle from the data register. Every data read that is not delivered from this cache invalidates the cache content. When the requested data arrives the cache contains again valid data.

This small data cache is invalidated when a write (i.e. erase or program) access to this address happens.

For data reads the IMB Core does not perform any autonomous pre-fetching.

3.9.3.3 Data Writes to Flash Memory

Flash memory content can not be changed by directly writing data to this memory. Command sequences are used to execute all other operations in the flash except reading. Command sequences consist of data writes with certain data to the flash memory address range. All data moves targeting this range are interpreted as command sequences. If they do not match a defined one or if the IMB Core is busy with executing a sequence (i.e. it is in “command mode”) a sequence error is reported.

3.9.3.4 Command Sequences

As described before changing data in the flash memory is performed with command sequences.

Table 3-3 Command Sequence Overview

Command Sequence	Description	Details on Page
Reset to Read	Reset Flash into read mode and clear error flags.	Page 3-26
Clear Status	Clear error and status flags.	Page 3-26
Change Read Margin	Change read margins.	Page 3-26
Enter Page Mode	Prepare page for programming.	Page 3-27
Enter Security Page Mode	Prepare security page for programming.	Page 3-28
Load Page Word	Load page with data.	Page 3-28
Program Page	Start page programming process.	Page 3-29
Erase Sector	Start sector erase process.	Page 3-30
Erase Page	Start page erase process.	Page 3-31
Erase Security Page	Start security page erase process.	Page 3-32
Disable Read Protection	Disable temporarily read protection with password.	Page 3-32
Disable Write Protection	Disable temporarily write protection with password.	Page 3-33
Re-Enable Read/Write Protection	Re-enable protection.	Page 3-34

3.9.4 Details of Command Sequences

The description defines the command sequence with pseudo assembler code. It is “pseudo” because all addresses are direct addresses which is generally not possible in real assembler code.

The commands are called by a sequence of one to six data moves into the flash memory range. The data moves must be of the “word” type, i.e. not byte move instructions. The following sections describe each command. The following abbreviations for addresses and data will be used:

- PA: “Page Address”. This is the base address of the destination page. For example the very first page has the address $C0'0000_H$. The page 13 of the second array has the PA = $C0'0000_H$ + 1·256·1024 (for the array) + 0·4·1024 (for the sector) + 13·128 (for the page) = $C4'0680_H$.
- SECPA: “Security Page Address”. This is the virtual address of a security page. It is “virtual” because SECPA is just used as argument of the command sequence to identify the security page but the physical storage of the security page is hidden.
  Two security pages are defined:
  SecP0: address $C0'0000_H$.
  SecP1: address $C0'0080_H$.
- WD: “Write Data”. This is a 16-bit data word that is written into the assembly buffer.
- SA: “Sector Address”. This is the physical sector number as defined in **Figure 3-6** based on the address of the flash module. Two examples as clarification:
  1. Physical sector number 16 of the first array that is based on $C0'0000_H$ is addressed with SA = $C0'0000_H$ + 16·4·1024 = $C1'0000_H$.
  2. The second 256 KB array has the base address $C4'0000_H$ (as shown in **Table 3-1**). So its physical sector number 3 has the SA = $C4'0000_H$ + 3·4·1024 = $C4'3000_H$.
- PWD: “Password”. This is a 64-bit password. It is transferred in 4 16-bit data words PWD0 = PWD[15:0], PWD1 = PWD[31:16], PWD2 = PWD[47:32] and PWD3 = PWD[63:48].
- Address XX followed by two hexadecimal digits, for example “$XXAA_H$”. If the command targets a certain flash module the XX must be translated to its base address. So “$XXAA_H$” means $C0'00AA_H$ for all commands addressing flash 0, $C4'00AA_H$ for flash 1 and $C8'00AA_H$ for flash 2. If a command (e.g. “Clear Status”) addresses the complete flash memory the base address of flash module 0 must be used.
- Data XX followed by two hexadecimal digits, e.g. $XXA5_H$. This is a “don’t care” data word where only the low byte must match a certain pattern. So in this example all data words like $12A5_H$ or $79A5_H$ can be used.
- MR: “Margin”. This 8-bit number defines the read margin. MR can take the values 00_H (normal read), 01_H (hard read 0), 02_H (alternate hard read 0), 05_H (hard read 1), 06_H (alternate hard read 1). All other values of MR are reserved.

Reset to Read

Arguments: –

Definition:

```
MOV XXAA_H, XXF0_H
```

Timing: One cycle command that does not set any “BUSY” flags. But note that an immediately following write access to the IMB Core is stalled for a few clock cycles during which the IMB Core is busy with aborting a previous command.

Description: The internal command state machine is reset to initial state and returns to read mode. An already started programming or erase operation is not affected and will be continued (the “**Reset to Read**” command — i.e. all commands — will anyhow not be accepted while the IMB Core is busy).

The “**Reset to Read**” command is a single cycle command. It can be used during a command sequence to reset the command interpreter and return the IMB Core into its initial state. It clears also all error flags in the Flash Status Register IMB_FSR and an active page mode is aborted. Because all commands are rejected with a SQER while the IMB Core is busy “**Reset to Read**” can not be used to abort an active command mode.

This command clears: PROER, PAGE, SQER, OPER, ISBER, IDBER, DSBER, DDBER.

Clear Status

Arguments: –

Definition:

```
MOV XXAA_H, XXF5_H
```

Timing: 1-cycle command that does not set any busy flags.

Description: The flags OPER, SQER, PROER, ISBER, IDBER, DSBER, DDBER in Flash status register are cleared. Additionally, the process status bits (PROG, ERASE, POWER, MAR) are cleared.

Change Read Margin

Arguments: MR

Definition:

```
MOV XXAA_H, XXB0_H
MOV XX54_H, XXMR_H
```

Timing: 2-cycle command that sets “BUSY” for around 30 micro seconds.

Description: This command sequence changes the read margin of one flash module. The address XX of the second move identifies the targeted flash module. The flash module needs some time to change its read voltage. During this time BUSY is set and this flash module cannot be accessed. The other flash modules stay readable.

The argument “MR” defines the read margin:

- 00_H: normal read margin.
- 01_H: hard read 0 margin.
- 02_H: alternate hard read 0 margin.
- 05_H: hard read 1 margin.
- 06_H: alternate hard read 1 margin.
- Other values: reserved.

For understanding the read margins please refer to **“Margin Reads” on Page 3-36**.

This command must not be issued when the flash memory is in page mode. In this case it is ignored and a sequence error is reported.

*Note: As noted in **“Margin Control” on Page 3-61** the command sequences “**Program Page**”, “**Erase Sector**”, “**Erase Page**” and “**Erase Security Page**” reset the read margin back to 00_H, i.e. to the normal read margin. The same happens in case of a flash wake-up.*

Enter Page Mode

Arguments: PA

Definition:

```
MOV XXAAH, XX50H
MOV PA, XXAAH
```

Timing: 2-cycle command that sets “BUSY” for around 100 clock cycles.

Description: The page mode is entered to prepare a page programming operation on page address PA. (Write data are accepted only with the “**Load Page Word**” command.)

With this command, the IMB Core initializes the write pointer of its block assembly register to zero so that it points to the first word. The page mode is indicated in the status register IMB_FSR with the PAGE bit, separately for each flash module. The page mode and the read mode are allowed in parallel at the same time and in the same flash module so the flash module stays readable. When the addressed page PA is read the content of the flash memory is delivered. The page mode can be aborted and the related PAGE bit in IMB_FSR be cleared with the “**Reset to Read**” command. A new “**Enter Page Mode**” command during page mode aborts the actual page mode, which is indicated with the error flag SQER, and restarts a new page operation. So as mentioned above only one of the flash modules can be in page mode at a time. If one of the erase commands or the “**Change Read Margin**” command are received while in page mode it is ignored and a sequence error is reported.

If write protection is installed for the sector to be programmed, the “**Enter Page Mode**” command is only accepted when write protection has before been disabled using the unlock command sequence “**Disable Write Protection**” with four passwords. If global write protection is installed with read protection, also the command “**Disable Read Protection**” can be used if no sector specific protection is installed. If write protection is

not disabled when the “**Enter Page Mode**” command is received, the command is not executed, and the protection error flag PROER is set in the IMB_FSR.

Enter Security Page Mode

Arguments: SECPA

Definition:

```
MOV XXAA_H, XX55_H
MOV SECPA, XXAA_H
```

Timing: 2-cycle command that sets “BUSY” for around 100 clock cycles.

Description: This command is identical to the “**Enter Page Mode**” command (see above), with the following exceptions: The addressed page (SECPA) belongs to the security pages of the flash memory and not to the user flash range. This command can only be executed after disabling of read protection and of sector write protection. Only if protection is not installed (e.g. for the very first installation of keywords), read/write protection need not be disabled. This command is not accepted and a protection error is reported if any protection is installed and active.

The use of this command to install passwords and to disable them again is described in **“Protection Handling Details” on Page 3-38**.

Load Page Word

Arguments: WD

Definition:

```
MOV XXF2_H, WD
```

Timing: 1-cycle command that does not set any “BUSY” flags. But note that an immediately following write access to the IMB Core or read from the flash memory is stalled for a few clock cycles if it arrives while the IMB Core is busy with copying its block assembly register content into the flash module assembly buffer. During this stall time the CPU can not perform any action! So either the user software can accept this stall time (which must be taken into account for the worst-case interrupt latency) or the software must avoid the blocking accesses.

Description: Load the IMB Core block assembly register with a 16-bit word and increment the write pointer. The 128 byte assembly buffer (i.e. a complete page) is filled by a sequence of 64 “Load Page Word” commands. The word address is not determined by the command but the “**Enter Page Mode**” command sets a write word pointer to zero which is incremented after each “**Load Page Word**” command.

This (sequential) data write access to the block assembly register belongs to and is only accepted in Page Mode. The command address of this single cycle command is always the same ($F2_H$). These low order address bits also identify the “**Load Page Word**” command and the sequential write data to be loaded into the block assembly register.

The high order bits XX should address the target page. The IMB Core takes always the page address that was used by the last “**Enter Page Mode**” command.

When the 128-bit block assembly register of the IMB Core is filled completely after 8 “**Load Page Word**” commands the IMB Core calculates the 9 ECC bits and transfers the block into the assembly buffer of the flash module. After that it sets the write pointer of the block assembly register back to zero. The following 8 “**Load Page Word**” commands fill again the block. After all 8 blocks are filled the “**Program Page**” command can be used to trigger the program process that transfers the assembly buffer content into the flash array.

While the IMB Core transfers the completed block assembly register to the flash module it can not accept new data for a few cycles. A “**Load Page Word**” command arriving during this time is stalled by the IMB Core.

If “**Program Page**” is called before all blocks of the assembly buffer have received new data then the remaining bits are cleared.

If more than 8 times 8 commands are used the additional data is lost. The overflow condition is indicated by the sequence error flag, but the execution of a following “**Program Page**” command is not suppressed (the page mode is not aborted).

When a “**Load Page Word**” command is received and the flash is not in page mode, a sequence error is reported in IMB_FSR with SQER flag. In case of a new “**Enter Page Mode**” command or a “**Reset to Read**” command during page mode, or in case of an Application Reset, the write data in the assembly buffer is lost. The current page mode is aborted and in case of a new “**Enter Page Mode**” command entered again for the new address.

Program Page

Arguments: –

Definition:

```
MOV XXAAH, XXA0H
MOV XX5AH, XXAAH
```

Timing: 2-cycle command that sets “BUSY” for the whole programming duration.

Description: The assembly buffer of the flash module is programmed into the flash array. If the last block of data was not filled completely this command finalizes its ECC calculation and copies its data into the assembly buffer before it starts the program process. The selection of the flash module and the page to be programmed depends on the page address used by the last “**Enter Page Mode**” command. The user software should always address the targeted page.

The programming process is autonomously performed by the selected flash module. The CPU is not occupied and can continue with its application.

The “**Program Page**” command is only accepted if the addressed flash module is in Page Mode (otherwise, a sequence error is reported instead of execution). With the “**Program Page**” command, the page mode is terminated, indicated by resetting the related PAGE flag and the command mode is entered and the PROG flag in the status register IMB_FSR is activated and the BUSY flag for the addressed module is set in IMB_FSR. While BUSY is set the IMB Core does not accept any further commands.

When the program process has finished BUSY is cleared but PROG stays set. It indicates which operation has finished and will be cleared by a System Reset or by “**Clear Status**”.

Read accesses to the busy flash module are not possible. Reading a busy flash module stalls until the flash module becomes ready again.

If write protection is installed for the sector to be programmed, the “**Program Page**” command is not accepted because the Flash is not in Page Mode (see description of the “**Enter Page Mode**” command).

If the page to be programmed is a security page (accepted only in security page mode), the new protection configuration (including keywords or protection confirmation code) is valid directly after execution of this command.

While the IMB Core reads the new protection configuration all DMU accesses to any flash module are stalled.

Erase Sector

Arguments: SA

Definition:

```
MOV XXAA_H, XX80_H
MOV XX54_H, XXAA_H
MOV SA, XX33_H
```

Timing: 3-cycle command that sets BUSY for the whole erasing duration.

Description: The addressed physical sector in the flash array is erased. Following data reads deliver all-zero data with correct ECC.

The erasing process is autonomously performed by the selected flash module. The CPU is not occupied and can continue with its application.

The sector to be erased is addressed by SA (sector address) in the last command cycle.

With the last cycle of the “**Erase Sector**” command, the command mode is entered, indicated by activation of the ERASE flag and after start of erase operation also by the related busy flag in the status register IMB_FSR. The BUSY flag is cleared after finishing the operation but ERASE stays set. It can be cleared by a System Reset or the “**Clear Status**” command.

Read accesses to the busy flash module are not possible. Read accesses to the not busy flash module are especially supported. Reading a busy flash module stalls until the flash module becomes ready again.

If write protection is installed for the sector to be erased, the Erase Sector command is only accepted when write protection has before been disabled using the unlock command sequence "**Disable Write Protection**". If global write protection is installed with read protection, also the command "**Disable Read Protection**" can be used if no sector specific protection is installed. If write protection is not disabled when the "**Erase Sector**" command is received, the command is not executed, and the protection error flag PROER is set in the IMB_FSR.

This command must not be issued when the flash memory is in page mode. In this case it is ignored and a sequence error is reported.

Erase Page

Arguments: PA

Definition:

```
MOV XXAA_H, XX80_H
MOV XX54_H, XXAA_H
MOV PA, XX03_H
```

Timing: 3-cycle command that sets BUSY for the whole erasing duration.

Description: The addressed page is erased. Following data reads deliver all-zero data with correct ECC.

With the last cycle of the "**Erase Page**" command, the command mode is entered, indicated by activation of the ERASE flag and after start of erase operation also by the related BUSY flag in the status register IMB_FSR. BUSY is cleared automatically after finishing the operation but ERASE stays set. It is cleared by a System Reset or the "**Clear Status**" command.

Read accesses to the busy flash array are not possible. Read accesses to the not busy flash modules are especially supported. Reading a busy flash module stalls until the flash module becomes ready again.

If the page to be erased belongs to a sector which is write protected, the command is only executed when write protection has before been disabled (see "**Erase Sector**" command).

In case of using the page erase care must be taken not to exceed the drain disturb limit of the other pages of the same sector.

This command must not be issued when the flash memory is in page mode. In this case it is ignored and a sequence error is reported.

Erase Security Page

Arguments: SECPA

Definition:

```
MOV XXAA_H, XX80_H
MOV XX54_H, XXA5_H
MOV SECPA, XX53_H
```

Timing: 3-cycle command that sets BUSY for the whole erasing duration.

Description: The addressed security page is erased.

This command is identical to the "**Erase Page**" command with the following exceptions: The addressed page (SecP0 or SecP1) belongs not to the user visible flash memory range. This command can only be executed after disabling of read protection and of sector write protection.

See **"Protection Handling Examples" on Page 3-45** for a detailed description of re-programming security pages.

The structure of the two security pages (SecP0 and SecP1) is described in **"Layout of the Security Pages" on Page 3-43**.

After erasing a security page the new protection configuration (including keywords or protection confirmation code) is valid directly after execution of this command.

While the IMB Core reads the protection configuration all DMU accesses to any flash module are stalled.

This command must not be issued when the flash memory is in page mode. In this case it is ignored and a sequence error is reported.

Disable Read Protection

Arguments: PWD

Definition:

```
MOV XX3C_H, XXXX_H
MOV XX54_H, PWD0
MOV XXAA_H, PWD1
MOV XX54_H, PWD2
MOV XXAA_H, PWD3
MOV XX5A_H, XX55_H
```

Timing: 6-cycle command that does not set any busy flag.

Description: Disable temporarily Flash read protection and — if activated — global write protection of the whole flash memory. The RPA bit in IMB_IMBCTR is reset.

This is a protected command sequence, using four user defined passwords to release this command or to check the programmed keywords. For every password one command cycle is required. If the second or fourth password represents the code of the

"**Reset to Read**" command, it is interpreted as password and the reset is not executed. The 16-bit passwords are internally compared with the keywords out of the "Security Page 0". If one or more passwords are not identical to their related keywords, the protected sectors remain in the locked state and a protection error (PROER) is indicated in the Flash status register. In this case, a new "**Disable Read Protection**" command or a "**Disable Write Protection**" command is only accepted after the next Application Reset.

Note: During execution of the "Disable Read" (or Write) Protection command a password compare error is only indicated after all four passwords have been compared with the related keywords.

Note: This command sequence is also used to check the correctness of keywords before the protection is confirmed in the Security Page 1. A wrong keyword is indicated by the IMB_FSR flag PROER.

After correct execution of this command, the whole flash memory is unlocked and the read protection disable bit RPRODIS is set in the Flash Status Register (IMB_FSR). Erase and program operations on all sectors are then possible, if the flash memory was also globally write protected (WPA=1), and if they are not separately write protected. The read protection (including global write protection, if so selected) remains disabled until the command "**Re-Enable Read/Write Protection**" is executed, or until the next Application Reset (including HW and SW reset).

Disable Write Protection

Arguments: PWD

Definition:

```
MOV XX3C_H, XXXX_H
MOV XX54_H, PWD0
MOV XXAA_H, PWD1
MOV XX54_H, PWD2
MOV XXAA_H, PWD3
MOV XX5A_H, XX05_H
```

Timing: 6-cycle command that does not set any busy flag.

Description: Disable temporarily the global flash write protection or/and the sector write protection of all protected sectors. The WPA bit in IMB_IMBCTR is reset.

This is a protected command sequence, using four user defined passwords to release this command (as described above for the "**Disable Read Protection**" command).

After correct execution of this command, all write-protected sectors are unlocked, which is indicated in the Flash Status Register (IMB_FSR) with the WPRODIS bit. Erase and program operations on all sectors are now possible, until

- The command "**Re-Enable Read/Write Protection**" is executed, or

- The next Application Reset (including HW and SW reset) is received.

Re-Enable Read/Write Protection

Arguments: –

Definition:

```
MOV XX5EH, XXXXH
```

Timing: 1-cycle command that does not set any busy flags.

Description: Flash read and write protection is resumed.

This single-cycle command clears RPRODIS and WPRODIS. The IMB Core is triggered to restore the protection states RPA and WPA from the content of the security page 0 as defined in **Table 3-4 ""Flash State" Determining RPA and WPA" on Page 3-40**. So in effect this command resumes all kinds of temporarily disabled protection installations.

This command is released immediately after execution.

3.9.5 Data Integrity

This section describes means for detecting and preventing the inadvertent modification of data in the flash memory.

3.9.5.1 Error Correcting Codes (ECC)

With very low probability a flash cell can become disturbed or lose its data value faster than specified. In order to reach the defined overall device reliability each 128-bit block of flash data is accompanied with a 9-bit ECC value. This redundancy supplies SEC-DED capability, meaning “single error correction and double error detection”. All single bit errors are corrected (and the incident is detected), all double bit errors are detected and even most triple bit errors are detected but some of these escape as valid data or corrected data.

A detected error is reported in the register **IMB_FSR_PROT**. Software can select which type of error should trigger a trap by the means of register **IMB_INTCTR**. In the system control further means exist to modify the handling of errors. The enabled trap requests by the flash module are handled there as “Flash Access Trap”. In case of a double-bit error the read data is always replaced with a dummy data word.

3.9.5.2 Aborted Program/Erase Detection

Where the ECC should protect from intrinsic failures of the flash memory that affect usually only single bits; an interruption of a running program or erase process might cause massive data corruption:

- The erase process programs first all cells to 1 before it erases them. So depending on the time when it is interrupted the data might be in a different state. This can be the old data, all-one, a random value, a weak all-zero or finally all-zero.
- The program process programs all bits concurrently from 0 to 1. If it is interrupted not all set bits might read as 1 or contain a weak 1.

The register **IMB_FSR_OP** contains the bits ERASE and PROG. These bits stay set until the next “**Clear Status**” command or System Reset. So if an erase or program process is interrupted by an Application Reset one of these bits is still set which allows to detect the interruption. It lies in the responsibility of the software to send the “**Clear Status**” command after a finalized program/erase process to enable this evaluation.

Another possible measure against aborted program/erase processes is to prevent resets by configuring the SCU appropriately.

If a program or erase process was aborted by a Power-On Reset (e.g. due to a power failure) there do not exist reliable means to detect this by reading the affected flash range. Even with margin reads an early or late aborted process might go unnoticed although it might in the long-term affect reliability.

Therefore the application must ensure that flash processes can perform uninterrupted and under the defined operating conditions, e.g. by early brown-out warning that prevents the software from starting flash processes.

After a flash process aborted the affected address range must be erased and re-programmed.

3.9.5.3 Margin Reads

Margin reads can be used to verify that flash data is readable with a certain margin. This is typically used as additional check directly after end-of-line programming. As explained above this is not a reliable method for detecting interrupted program or erase processes but the probability of detecting such cases can be increased.

Reading with “hard read 0 margin” returns weak 0s as 1s and reading with “hard read 1 margin” returns weak 1s as 0s. Changing the read margin is done with the command sequence “**Change Read Margin**” and is reported by the status register “**IMB_MAR**”.

3.9.5.4 Protection Overview

The flash memory supports read and write protection for the whole memory and separate write protection for each logical sector. The logical sector structure is depicted in **Figure 3-6**.

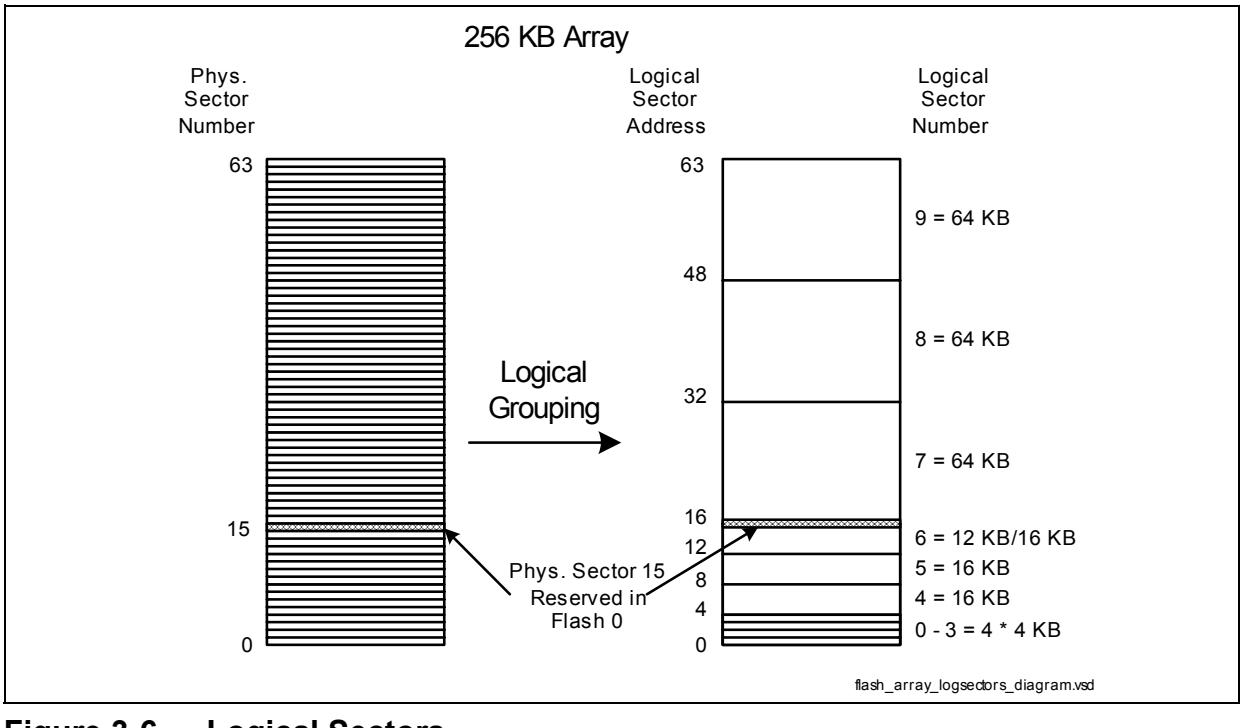


Figure 3-6 Logical Sectors

If read protection is installed and active, any flash read access is disabled in case of start after reset from external memory or from internal RAM. Debug access is as well disabled

and thus the execution of injected OCDS instructions. In case of start after reset in internal flash, all flash access operations are controlled by the flash-internal user code and are therefore allowed, as long as not especially disabled by the user, e.g. before enabling the debug interface.

Per default, the read protection includes a full (global) flash memory write protection covering all flash modules. This is necessary to eliminate the possibility to program a dump routine into the Flash, which reads the whole Flash and writes it out via the external bus or a serial interface. Program and erase accesses to the flash during active read protection are only possible, if write protection is separately disabled. Flash write and read protection can be temporarily disabled, if the user authorizes himself with correct passwords.

The device also features a sector specific write protection. Software locking of flash memory sectors is provided to protect code and data. This feature disables both program and erase operations for all protected sectors. With write protection it is supported to protect the flash memory or parts of it from unauthorized programming or erase accesses and to provide virus-proof protection for all sectors.

Read and write protection is installed by specific security configuration words which are programmed by the user directly into two “Security Pages” (SecP0/1). After any reset, the security configuration is checked by the command state machine (IMB Core) and installations are stored (and indicated) in related registers. If any protection is enabled also the security pages are especially protected.

For authorization of short-term disabling of read protection or/and of write protection a password checking feature is provided. Only with correct 64-bit password a temporary unprotected state is taken and the protected command sequences are enabled. If not finished by the command “**Re-Enable Read/Write Protection**”, the unprotected state is terminated with the next reset. Password checking is based on four 16-bit keywords (together 64 bits) which are programmed by the user directly into the “Security Page 0” (SecP0).

Special support is provided to protect also the protection installation itself against any stressing or beaming aggressors. The codes of configuration bits are selected, so that in case of any violation in the flash array, on the read path or in registers the protected state is taken per default. In registers and security pages, protection control bits are coded always with two bits, having both codes, “00_B” and “11_B” as indication of illegal and therefore protected state.

3.9.6 Protection Handling Details

As shortly described in **"Protection Overview" on Page 3-36** the flash memory can be in different protection states. The protection handling can be separated into different layers that interact which each other (see **Figure 3-7**).

- The lowest layer consists of the physical content of the security pages SecP0 and SecP1. This information is used to initialize the protection system during startup.
- The next layer consists of registers that report the state of the physical layer (IMB_PROCONx) and the protection state (IMB_FSR). The protection state can be temporarily changed with command sequences which is reflected in the IMB_FSR.
- The highest layer is represented by 4 fields of the IMB_IMBCTR register. These fields define the protection rights of the customer software (are read or write accesses currently allowed or not).

The IMB Core controls the protection state of all connected flash modules centrally. In this position it can supervise all accesses that are issued by the CPU.

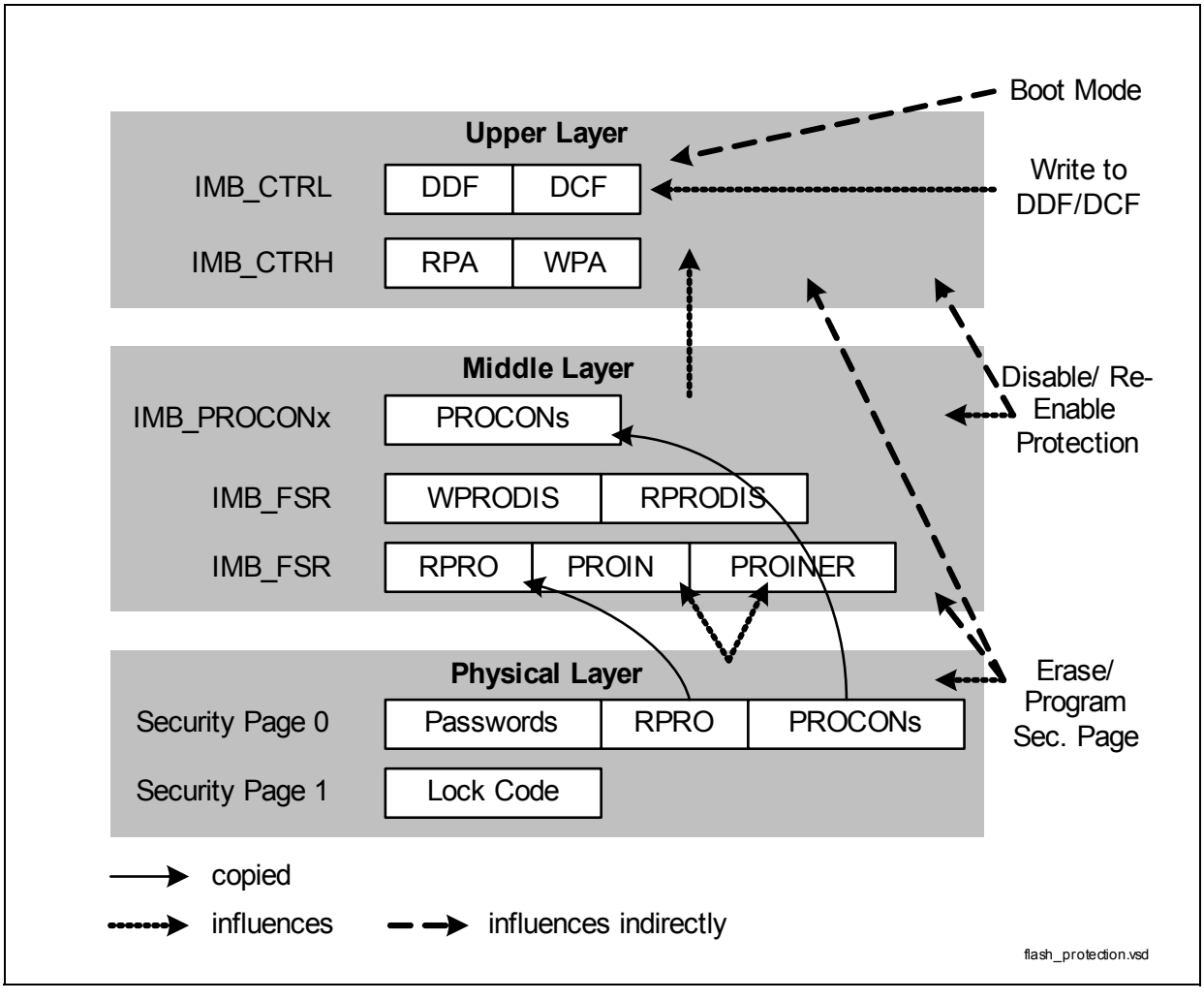


Figure 3-7 Protection Layers

3.9.6.1 The Lower Layer “Physical State”

After reset the protection state of the device is restored from the following information:

- The security page 1 contains a “lock code”. This consists of two words of data (32 bits). If it has the value $AA55AA55_H$ then security page 0 determines the protection state. Otherwise (i.e. the lock code was not found) the device is in the “non-protected state”. The content of the security page 0 is still copied into the registers as described in **“The Middle Layer “Flash State”” on Page 3-39** but their values are ignored in the non-protected state.
- The security page 0 contains the RPRO double bit, the write protection bits SnU and 4 passwords. If the field RPRO contains a valid 01_B or 10_B entry the page is valid and the device is in the “protection installed state”. The page content determines the security settings after startup. If SecP0 contains an invalid RPRO entry the device is in the “errored protection” state.

To summarize: the content of the security pages determines if the device is in the “non-protected state”, “protection installed state” or “errored protection state”. These states are reflected in the register settings of the next layer.

The device is usually delivered in the “non-protected state”.

The exact layout of the security pages is described in **“Layout of the Security Pages” on Page 3-43**.

3.9.6.2 The Middle Layer “Flash State”

The middle layer consists of the registers IMB_PROCONx and IMB_FSRx and commands that manipulate them and the content of the security pages.

During startup the physical state is examined by the IMB Core and it is reflected in the following bit settings:

- “non-protected state”: IMB_FSR.PROIN = 0, IMB_FSR.PROINER = 0.
- “protection installed state”: IMB_FSR.PROIN = 1, IMB_FSR.PROINER = 0.
- “errored protection state”: IMB_FSR.PROIN = 0, IMB_FSR.PROINER = 1.

The fourth possible setting PROIN=1 and PROINER=1 is invalid and can not occur.

The IMB_PROCONx registers are initialized during startup with the content of the security page 0. The bits DSBER and DDBER indicate if an ECC error occurred. The customer software has thus the possibility to detect disturbed security pages and it can refresh their content.

Commands

Other bits of the IMB_FSR: RPRODIS, WPRODIS, PROER can be manipulated with command sequences and define together with the other bits the protection effective for the next layer. All three bits are 0 after system startup.

The command “**Disable Read Protection**” sets RPRODIS to 1 if the correct passwords that are stored in SecP0 are supplied. If incorrect passwords are entered the bit PROER is set and RPRODIS stays unchanged. As protection against “brute force attacks” that search the correct password the password detection is locked. So after supplying the first incorrect password all following passwords even the correct ones are rejected with PROER. This state is only left by an Application Reset or by erasing SecP0.

The disabled protection can be enabled again by the Application Reset or by the command “**Re-Enable Read/Write Protection**” which clears RPRODIS again.

The bit PROER can be reset by an Application Reset or by the commands “**Reset to Read**” and “Clear Status”.

The command “**Disable Write Protection**” sets WPRODIS to 1 if the correct passwords are supplied. It behaves analog to RPRODIS as described above.

The command “**Re-Enable Read/Write Protection**” clears RPRODIS and WPRODIS.

The commands “**Enter Page Mode**”, “**Enter Security Page Mode**”, “**Erase Page**”, “**Erase Security Page**” and “**Erase Sector**” set PROER if the write access to the addressed range is not allowed. If a write access is allowed or not is determined by the next level.

Table 3-4 summarizes how the “Flash State” of protection determines the RPA and WPA fields of IMB_IMBCTR. For the double bits a short notation is used here and in the following sections: 1 means active, 0 means inactive, ‘#’ means invalid and ‘–’ means do not care including invalid states. The symbol ‘|’ means logic or.

Table 3-4 “Flash State” Determining RPA and WPA

IMB_FSR.PROIN	IMB_FSR.PROINER	IMB_FSR.RPRO	IMB_FSR.RPRODIS	IMB_FSR.WPRODIS	Resulting Security Level in RPA and WPA
0	0	–	–	–	Non-protected state: RPA = 0, WPA = 0.
1	0				Protection installed state (possibly disabled, see below):
		0	–	0	RPA = 0, WPA = 1.
		0	0	1	RPA = 0, WPA = 0.
		1 | #	0	0	RPA = 1, WPA = 1.
		–	1	1	RPA = 0, WPA = 0 (all disabled).
		1 | #	0	1	RPA = 1, WPA = 0.
		1 | #	1	0	RPA = 0, WPA = 1.

Table 3-4 “Flash State” Determining RPA and WPA (cont’d)

IMB_FSR.PROIN	IMB_FSR.PROINER	IMB_FSR.RPRO	IMB_FSR.RPRODIS	IMB_FSR.WPRODIS	Resulting Security Level in RPA and WPA
0	1				Errored protection state (see below):
		–	0	0	RPA = 1, WPA = 1.
		–	0	1	RPA = 1, WPA = 0.
		–	1	0	RPA = 0, WPA = 1.
		–	1	1	RPA = 0, WPA = 0.

3.9.6.3 The Upper Layer “Protection State”

This layer consists mainly of the 4 fields DCF, DDF, WPA and RPA of the IMB_IMBCTR register. These determine the effective protection state together with registers of the lower layers. Some of the above mentioned command sequences directly influence these fields as well. In order to increase the resistance against beaming or power supply manipulation all 4 fields are coded with 2 bits. Generally “01” means active, “10” inactive and the two other states “00” and “11” are invalid and are recognized as “attacked” state.

Effective Security Level

The effective security level based on these 4 double-bits is summarized in **Table 3-5** and **Table 3-6**. For the double bits the same short notation is used as before: 1 means active, 0 means inactive, ‘#’ means invalid and ‘–’ means do not care including invalid states.

Table 3-5 Effective Read Security

RPA	DCF	DDF	Security Level
0	–	–	No read protection.
1 | #	0	0	No read protection.
	–	1 | #	Data reads prohibited.
	1 | #	–	Code fetches prohibited.

Table 3-6 Effective Write Security

WPA	RPA	Security Level
0	–	No write protection

Table 3-6 Effective Write Security (cont'd)

WPA	RPA	Security Level
1 | #	1 | #	Global write protection.
1 | #	0	Sector specific write protection depending on IMB_PROCONx.

To summarize:

- Read protection is always globally affecting the whole flash memory range. Code fetches and data reads can be separately controlled.
- Write protection can be global when the read protection is effective or it can be specific for each logical sector.

The lower and the middle security layers determine how the 4 effective IMB_IMBCTR fields are preset, changed and how software can access them. This is discussed in the following paragraphs.

Initialization of the Effective Security Level

After Application Reset protection is activated so that RPA, WPA, DDF and DCF are set. During startup the IMB Core determines the stored security level as described in **"The Lower Layer "Physical State"" on Page 3-39** and sets IMB_FSR.PROIN and IMB_FSR.PROINER and IMB_PROCONx as described in **"The Middle Layer "Flash State"" on Page 3-39**. The IMB Core further initializes the IMB_IMBCTR fields RPA and WPA according to the rules of **Table 3-4**.

The bits DDF and DCF of the IMB_IMBCTR are not initialized by the IMB Core. During system startup they are initialized depending on the startup condition. If code fetching starts in the flash memory then they are set to the inactive state. In all other cases they are activated to prevent read access to the flash memory without proving password knowledge.

Changing the Effective Security Level

During run-time the effective security level can be changed. This can be done by directly writing to the IMB_IMBCTR register or indirectly by changing the bits of the middle layer by commands as "**Disable Write Protection**" or even double indirectly by changing the content of the security pages which changes bits in the middle layer and influences the effective security level.

Writing directly to IMB_IMBCTR:

- DCF and DDF can be deactivated only if RPA is inactive. They can always be activated.

Indirectly by using a command sequence:

- A successful "**Disable Read Protection**" sets RPRODIS and clears RPA.

- A successful “**Disable Write Protection**” sets WPRODIS and clears WPA.
- “**Re-Enable Read/Write Protection**” clears RPRODIS and WPRODIS and sets RPA and WPA according to **Table 3-4** depending on PROIN, PROINER and RPRO.

Double indirect by changing security pages. After executing a command sequence that changed the content of a security page the IMB Core immediately reads back the pages and determines all resulting security data as described for system startup in **“Initialization of the Effective Security Level” on Page 3-42**. The examples in **“Protection Handling Examples” on Page 3-45** will show how this can be used for installing and removing protection or changing passwords.

3.9.6.4 Reaction on Protection Violation

If software tries to violate the protection rules the following happens:

- Reading data when read protection is effective: The bit IMB_FSR.PROER is set and the Flash access trap can be triggered via the SCU if IMB_INTCTR.DPROTRP is 0. Default data is delivered.
- Fetching code when read protection is effective: the trap code “TRAP 15_D” is delivered instead.
- Programming or erasing memory ranges when they are write protected: PROER is set.

3.9.6.5 Layout of the Security Pages

The previous sections just mentioned the content of the security pages. This section depicts their exact layout. **Figure 3-8** depicts symbolically the layout of the security pages 0 and 1.

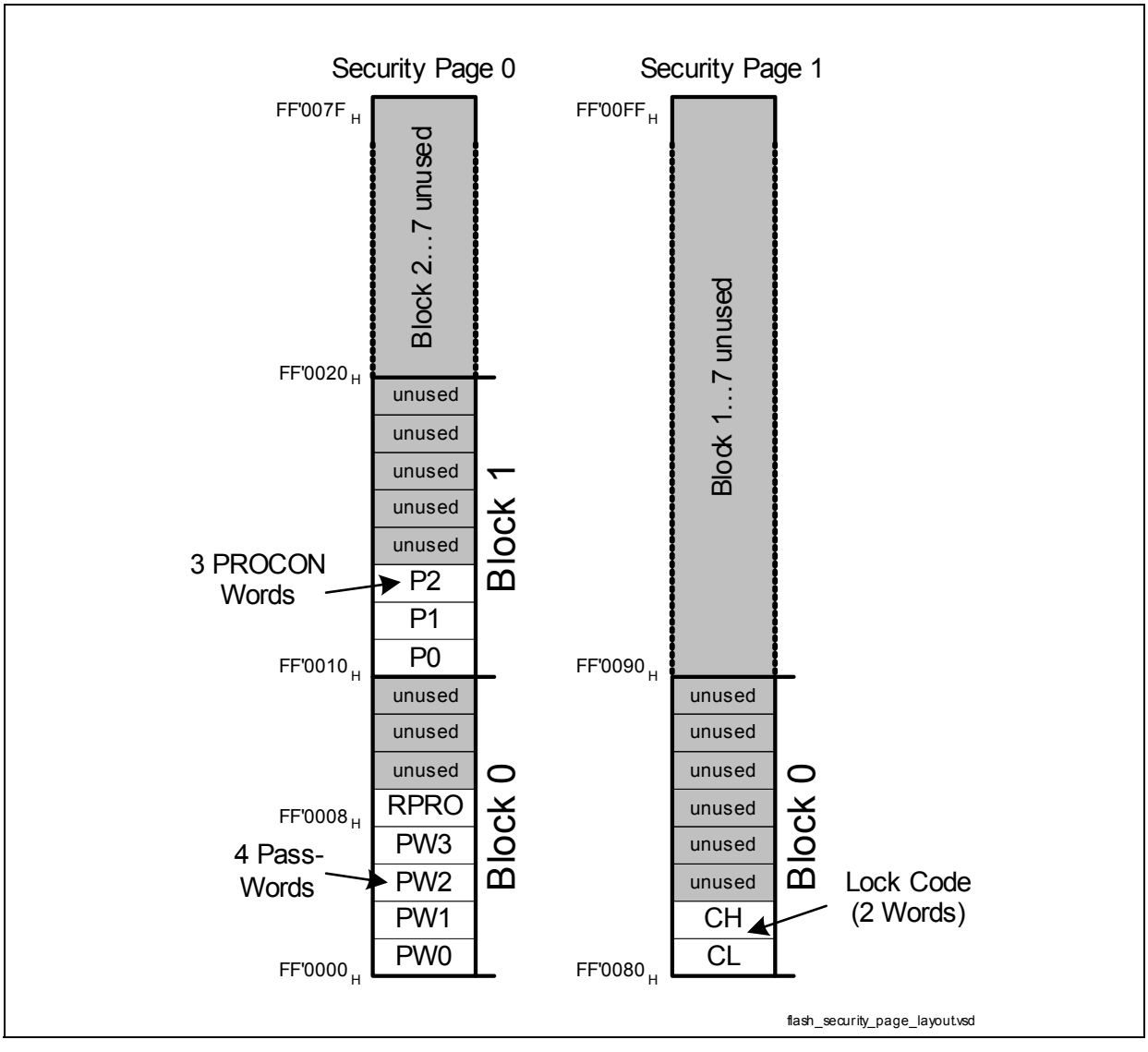


Figure 3-8 Layout of Security Pages

Generally the 16-bit words are stored as always in the XE16x in little endian format.

- The PWx words contain the passwords.
- The double bit RPRO is stored as in the related ISFR **IMB_FSR_PROT** in the bits 15 and 14. The other bits of this word are unused and should be kept all-zero.
- The PROCON data is stored as defined in the **IMB_PROCONx (x=0-2)** ISFR.
- The lock code consists of the two words CL and CH. Both contain "$AA55_H$" to form the correct lock code.

All bytes of the used blocks of the security pages (block 0 and 1 of SecP0 and block 0 of SecP1) are to be considered as "reserved" and must be kept erased, i.e. with all-zero content. The unused blocks of the security pages (blocks 2 to 7 of SecP0 and blocks 1 to 7 of SecP1) shall be programmed with all-one data.

3.9.7 Protection Handling Examples

Some examples on how to work with the protection system.

Delivery State

The device is delivered in the “non-protected state”.

Security page 1 is erased (so it does not contain the “lock code” $AA55AA55_H$).

Security page 0 is erased and so “invalid” but because SecP1 is erased this data is anyhow not evaluated. Only its content is copied into corresponding the registers.

During startup the bits DDF and DCF are set depending on the start mode but as RPA and WPA are inactive all accesses to the flash memory are allowed.

The data sectors of the flash memory are delivered in the erased state as well. All sectors can be programmed. After uploading the software the customer can install write and read protection.

First Time Password Installation

In order to install a password generally the lock code in SecP1 has to be erased. In this case the code is not present.

After that SecP0 must be erased with “**Erase Security Page**” in order to be able to change RPRO. Erasing SecP0 clears RPRO to “00_B” which is an invalid state. After finishing the erase command the IMB Core restores the IMB_FSR and IMB_IMBCTR fields from the flash data.

Because no lock code is present in SecP1 the invalid state of RPRO has no effect on the user visible protection. Still all parts of the flash memory can be written.

The second step is to program the information of SecP0 with the required security information. Again the IMB Core reads immediately back the stored data and initializes the security system. As SecP1 still does not contain the lock code the device stays in the “non-protected” mode.

The security pages cannot be read directly by customer software. The data programmed into SecP0 can therefore only be verified indirectly. The data of the RPRO and SnU fields can be checked by reading the IMB_PROCON and IMB_FSR registers. The passwords can be verified with the command “**Disable Read Protection**”. If the password does not match the bit PROER is set. But because of the erased SecP1 the flash memory stays writable. So after erasing SecP0 the correct password can be programmed again.

After the SecP0 was verified successfully SecP1 gets programmed with the lock code $AA55AA55_H$ which enables the security settings of SecP0.

Because the password validation left RPRODIS set the command “**Re-Enable Read/ Write Protection**” must be used to finally activate the new protection.

Changing Passwords or Security Settings

Changing the passwords is a delicate operation. The interrelation of the two security pages must be kept in mind.

Usually in the protected state the SecP1 contains the lock code. First write protection must be disabled with the correct passwords. Then the lock code in SecP1 is erased. If this operation was successful PROIN will be cleared by the IMB Core. Now SecP0 can be safely erased.

From this point on the security pages are in the factory delivery state and the new passwords and security settings can be installed as described above.

Attention: The number of times a security page may be changed is noted in the data sheet.

3.9.8 EEPROM Emulation

The flash memory of the XE16x is used for three purposes:

1. Storage of program code. Updates happen usually very seldom. The main criteria to be fulfilled is a retention of the life-time of the product.
2. Storage of constant data: this data is stored together with program code. So this data is very seldom updated. Endurance is of no issue here but retention identical to the code memory is required.
3. Data updated during run-time: this might be data with a very high frequency of updates like a mileage counter or access keys for key-less entry. Other data might be changed only in case of failures and other data might only be transferred from RAM to non-volatile memory before the system is powered down.

Especially for the third type of data the non-volatile memory needs EEPROM like characteristics:

- Fine program/erase granularity which is in EEPROMs typically 1 byte.
- Higher endurance than the intrinsic endurance of flash cells.
- Short program and erase duration per byte. Especially for storing data in an emergency (e.g. power failure) short latencies might be required.

A basic requirement for changing data during run-time is that code execution can still resume, especially interrupt requests must still be serviced. This requirement is fulfilled in the XE16x because all three flash modules work independently. If one is busy with program or erase then code can still be executed from the other two.

The other requirements are more difficult to fulfill because the XE16x does not have an EEPROM available but only the flash memory with the already frequently mentioned limitations: big program/erase granularity, moderately long program/erase duration, limited cell endurance with reduced retention at high number of program/erase cycles, pages not isolated but affected by drain disturbs.

In order to alleviate these effects on run-time storage of data software is used to emulate EEPROM. There is quite a number of algorithms for efficiently using flash memory as EEPROM. The following section describes one (the most simple) of these algorithms.

It should be noted that the XE16x does not offer the customer any hardware means for EEPROM emulation. All of the following must be realized by software.

3.9.8.1 The Traditional EEPROM Emulation

The key point is to solve the limited endurance by storing data in N different physical places. In XE16x the algorithm could use N sequential pages or groups of pages. If data is currently stored in the page “x” then the next program happens to the page “(x+1) mod N”.

After boot up the last correct page group must be found. This could be done by either evaluating a counter (from 0 to 2*N-1) or the old entries are invalidated by erasing the

page after programming the new one. Additionally a CRC check could be performed over the group.

As all involved pages are re-used cyclically the endurance from customer perspective is increased by the factor N. N must be chosen high enough to fulfill endurance and retention requirements. Disturbs in the group of N pages are no issue because they incur at most N-1 disturbs before they get written with new data. Care must be taken however if one sector accommodates different groups of pages with different update behavior. In this case the updates of one group of pages could exceed the disturb limits of the other group. So generally one sector should be used only by one such EEPROM cyclic buffer.

The algorithm keeps the old data until the new data is verified so power failure during programming can only destroy the last update but the older data is still available. There are still some issues with power failure that need special treatment:

- Power is cut during programming: the following boot-up might find an apparently correctly programmed page. However the cells might be not fully programmed and thus have a much lower retention or the read data is unstable (e.g. changing operating conditions cause read errors).
  If the power is cut early the page can appear as erased although some cells are partly programmed. When programming different data to this apparently erased page read errors might occur.
- Power is cut during erase: the same as above can happen. Data may appear as erased but the retention is lowered. A power failure during a page-erase can inhibit readability of all data in its physical sector. Therefore an algorithm is advantageous that performs erases only in sectors that don’t contain anymore current data.

The algorithm can be improved to be more robust against such cases, e.g. program always two pages, mark the end of an erase process by programming a page. But generally aborting flash processes is a forbidden “operating condition”.

The main deficiency of the described algorithm is that the software designer is required to plan the use of the flash memory thoroughly. The user has to choose the correct value of N. Then all data has to be allocated to pages. Data sharing one page should have a similar or better identical update pattern (otherwise unchanged data is unnecessarily written). If one set of data does not fill a complete sector the available pages must be possibly left unused because they might incur too many drain disturbs.

There are other algorithms that try to alleviate these efforts by monitoring the flash usage and adapt automatically the assignment of data to flash cells.

3.9.9 Interrupt Generation

Long lasting processes (these are mainly: program page, erase page, erase sector and margin changes) set the IMB_FSR.BUSY flag of one flash module when accepting the request and reset this flag after finishing the process. Software is required to poll the busy flag in order to determine the end of the operation. In order to release the software from this burden an interrupt can be generated. If the interrupt is enabled by IMB_INTCTRL.IEN then all transitions from 1 to 0 of one of the 3 IMB_FSR.BUSY flags send an interrupt request.

The “**Enter Page Mode**” command sets BUSY only for around 100 clock cycles. It is usually not advisable to enable the interrupt for this command.

The register **IMB_INTCTR** contains fields for the interrupt status “ISR”, an enable for the interrupt request “IEN” and fields for clearing the status flag “ICLR” or setting if “ISET”. It should be noted that the interrupt request is only sent when ISR becomes 1 and IEN was already 1. No interrupt is sent when IEN becomes 1 when ISR was already 1 or both are set to 1 at the same time.

3.9.10 Recommendations for Optimized Flash Usage

This section describes best practices for using the flash in certain application scenarios, e.g. how to use effectively ECC and margin reads. For a description of the hardware features consult **“Data Integrity” on Page 3-35**.

3.9.10.1 Programming Code and Constant Data

Code and constant data are programmed only few times during life-time of a device, e.g. end-of-line in ECU production or when service updates are performed. As the readability of this data is decisive for the product quality customers might want to implement the elaborate “best practice” advice.

Basic Advice

Always ensure correct operating conditions and prevent power failures during flash operation.

As basic protection against handling errors all data should be verified after programming. Single-bit ECC errors should be ignored. The appearance of small numbers of single-bit errors is a consequence of known physical effects.

Best Practice

This approach offers best possible quality but risks that programming steps need to repeated even unnecessarily (“false negatives”):

- Use “Erase Sector” to erase complete sectors.

- Program the sector with data. A common protection against software crashes is to fill the unused part of the sector with trap codes.
- Change the read level to hard margin 0.
- Verify the programmed data, note comparison errors and double-bit ECC errors and count single-bit ECC errors. Take care to evaluate the ECC error flags only once per 128-bit data block and clear them afterwards.
- Repeat this check with hard margin 1.
- After programming all sectors:
  – Erase and re-program all sectors with comparison or double-bit ECC errors.
  – If a flash module contained more than a certain number (e.g. 10) of single-bit ECC errors it is recommended to erase and re-program the affected sectors (i.e. those containing at least one single-bit error).
  – Attention: a high number of single-bit errors indicates usually a violation of operating conditions.

The threshold of allowed single-bit errors could be increased for in-service updates in order to reduce the risk of false negatives.

3.9.10.2 EEPROM Emulation

For EEPROM emulation the goal is usually not readability over device life-time but highest possible robustness (against violated operating conditions, power failures, even failing flash pages e.g. due to over-cycling). The risk of false negatives should be minimized.

A good robustness is achieved with the following approach:

- Verify data after programming with the normal read level. Single-bit ECC errors should be ignored.
- In case of comparison error or double-bit ECC error the data should be programmed again to the next flash range (e.g. next page or sector).
- The number of re-programming trials should be limited (e.g. to 3) to protect against violated operating conditions.

Obviously this jumping over failed pages can be only used optimally when the algorithm does not expect data on fixed addresses.

Failing pages can prevent “Erase Sector” from erasing any data in the affected sector. The “Erase Page” command however could still erase all other pages. These other pages stay readable and programmable.

3.10 On-Chip Program Memory Control

The internal memory block “IMB” contains all memories of the so called “on-chip program memory area” in the address range from $C0'0000_H$ to $FF'FFFF_H$. Included are the program SRAM, the embedded flash memories and central control logic called “IMB Core”.

In the XE16x device the IMB contains the following memories:

- 764 KB flash memory in three independent modules.
- 64 KB program SRAM (see **Section 3.4.1**).

The IMB connects these memories to the CPU data bus and the instruction fetch bus. Each memory can contain instruction code, data or a mixture of both. The IMB manages accesses to the memories and supports flash programming and erase.

3.10.1 Overview

The **Figure 3-9** shows how the IMB and its memories are integrated into the device architecture. Only the main data streams are included. The data buses are usually accompanied by address and control signals and check-sum data like parity or ECC.

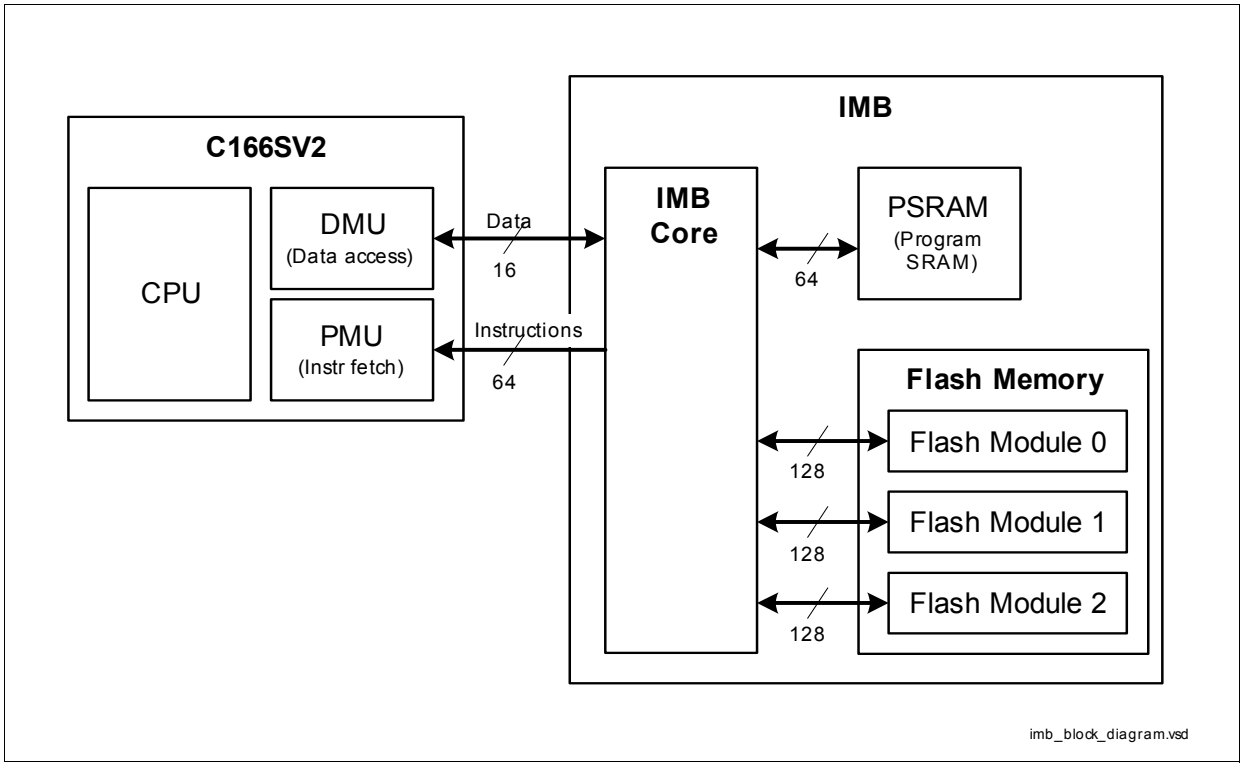


Figure 3-9 IMB Block Diagram

The CPU has two independent busses. The instruction fetch bus is controlled by the program management unit “PMU” of the CPU. It fetches instructions in aligned groups of 64 bits. The instruction fetch unit of the CPU predicts the outcome of jumps and fetches

instructions on the predicted branch in advance. In case of a misprediction this interface can abort outstanding requests and continues fetching on the correct branch. As the CPU can consume up to one 32-bit instruction per clock cycle the performance of this interface determines the CPU performance.

The data bus is controlled by the data management unit “DMU” of the CPU. It reads data in words of 16 bits. Write accesses address as well 16-bit words but additional byte enables allow changing single bytes.

Because of the CPU’s “von Neumann” architecture data and instructions (and “special function registers” to complete the list) share a common address range. When instructions are used as data (e.g. when copying code from an IO interface to the PSRAM) they are accessed via the data bus. The pipelined behavior of the CPU can cause that code fetches and data accesses are requested simultaneously. The IMB takes care that accesses can perform concurrently if they address different memories or flash modules.

Additional connections of the IMB to central system control units exist. These are not shown in the block diagram.

3.10.2 Register Interface

The **"IMB Registers" on Page 3-53** describes the special function registers of the IMB. In **"System Control Registers" on Page 3-HIDDEN** the special function registers that influence the IMB but are not allocated to the IMB address range are described.

3.10.2.1 IMB Registers

The section describes all IMB special function registers.

Table 3-7 Registers Overview

Register Short Name	Register Long Name	Offset Address	Page Number
IMB_IMBCTRL	IMB Control Low	FF FF00$_H$	**Page 3-53**
IMB_IMBCTRH	IMB Control High	FF FF02$_H$	**Page 3-55**
IMB_INTCTR	Interrupt Control	FF FF04$_H$	**Page 3-56**
IMB_FSR_BUSY	Flash State Busy	FF FF06$_H$	**Page 3-58**
IMB_FSR_OP	Flash State Operations	FF FF08$_H$	**Page 3-58**
IMB_FSR_PROT	Flash State Protection	FF FF0A$_H$	**Page 3-60**
IMB_MAR	Margin	FF FF0C$_H$	**Page 3-62**
IMB_PROCON0	Protection Configuration 0	FF FF10$_H$	**Page 3-63**
IMB_PROCON1	Protection Configuration 1	FF FF12$_H$	**Page 3-63**
IMB_PROCON2	Protection Configuration 2	FF FF14$_H$	**Page 3-63**

IMB Control

Global IMB control.

Both IMB_IMBCTRL and IMB_IMBCTRH are reset by an Application Reset.

The write access to both registers is controlled by the register security mechanism as defined in the SCU chapter **"Register Control" on Page 6-191**. Please note that the register write-protection is not activated automatically again after an access to IMB_IMBCTR because this happens only for SCU internal registers.

IMB_IMBCTRL
IMB Control Low **ISFR (FF FF00$_H$)** **Reset value: 558C$_H$**

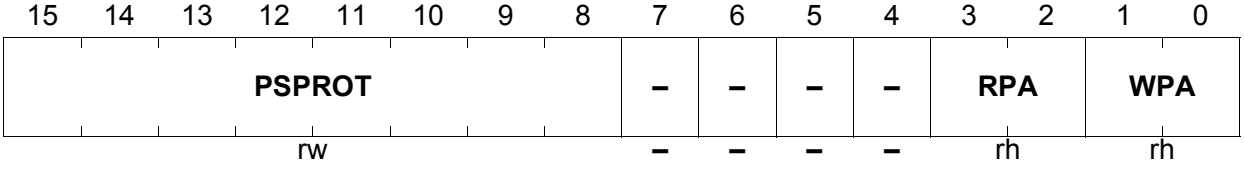
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DDF		DCF		-	-	-	-	-	-	-	-	DLC PF	WSFLASH		
rw		rw		-	-	-	-	-	-	-	-	rw	rw		

Field	Bits	Typ	Description
WSFLASH	[2:0]	rw	**Wait States for Flash Access** Number of wait cycles after which the IMB expects read data from the flash memory. This field determines as well the read timing of the PSRAM in the flash emulation address range. See **“Flash Emulation” on Page 3-12**. *Note: WSFLASH must not be 0. This value is forbidden!*
DLCPF	3	rw	**Disable Linear Code Pre-Fetch** 0: “High Speed Mode”: When the next read request will be delivered from the buffer and so the flash memory would be idle, the IMB Core autonomously increments the last address and reads the next 128-bit block from the flash memory. 1: “Low Power Mode”: This feature is disabled. Usually for code with power minimization requirements or for code with short linear code sections this feature should be disabled (DLCPF = 1). Enabling this feature is only advantageous for code section with longer linear sequences. With lower values of WSFLASH the performance gain of DLCPF=0 is reduced. In case of low WSFLASH settings DLCPF=1 might even lead to better performance than with linear code pre-fetch.
DCF	[13:12]	rw	**Disable Code Fetch from Flash Memory** “01”: Short notation DCF = 1. If RPA = 1 instructions cannot be fetched from flash memory. If RPA = 0 this field has no effect. “10”: Short notation DCF = 0. Instructions can be fetched independent of RPA. “00” | “11”: Illegal state. Has the same effect as “01”. This state can only be left by an Application Reset. During startup or test mode or when RPA = 0 software can change this field to any value. Otherwise code fetch can only be disabled but not enabled anymore until the next Application Reset.

Field	Bits	Typ	Description
DDF	[15:14]	rw	**Disable Data Read from Flash Memory** “01”: Short notation DDF = 1. If RPA = 1 data cannot be read from flash memory. If RPA = 0 this field has no effect. “10”: Short notation DDF = 0. Data can be read independent of RPA. “00” | “11”: Illegal state. Has the same effect as “01”. This state can only be left by an Application Reset. During startup or test mode or when RPA = 0 software can change this field to any value. Otherwise data reads can only be disabled but not enabled anymore until the next Application Reset.

IMB control high word. The WPA and RPA fields are described in **“Protection Handling Details” on Page 3-38**.

IMB_IMBCTRH
IMB Control High **ISFR (FF FF02$_H$)** **Reset value: 0005$_H$**



Field	Bits	Typ	Description
WPA	[1:0]	rh	**Write Protection Activated** “01”: Short notation WPA = 1. The write protection of the flash memory is activated. “10”: Short notation WPA = 0. The write protection is not activated. “00” | “11”: Illegal state. Same effect as “01”. The illegal state can only be left by an Application Reset. This field is only changed by the IMB Core. Software writes are ignored.

Field	Bits	Typ	Description
RPA	[3:2]	rh	**Read Protection Activated** “01”: Short notation RPA = 1. The read protection of the flash memory is activated. “10”: Short notation RPA = 0. The read protection is not activated. “00” | “11”: Illegal state. Same effect as “01”. The illegal state can only be left by an Application Reset. This field is only changed by the IMB Core. Software writes are ignored.
PSPROT	[15:8]	rw	**PSRAM Write Protection** This 8-bit field determines the address up to which the PSRAM is write protected. The start address of the writable range is E0’0000$_H$ + 1000$_H$*PSPROT. The end address is determined by the implemented memory. The equivalent range in the PSRAM area with flash access timing is protected as well. Here the writable range starts at E8’0000$_H$ + 1000$_H$*PSPROT and ends at E8’FFFF$_H$ for XE16x. So with PSPROT=00$_H$ the complete PSRAM is writable. In case of XE16x with PSPROT=10$_H$ or bigger the complete implemented PSRAM is write-protected.

Interrupt Control

Interrupt control and status.

Reset by Application Reset.

IMB_INTCTR
Interrupt Control **ISFR (FF FF04$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ISR	**PSER**	**–**	**–**	**–**	**PSERCLR**	**ISET**	**ICLR**	**–**	**–**	**–**	**–**	**DPROTRP**	**DDDTRP**	**DIDTRP**	**IEN**
rh	rh	–	–	–	w	w	w	–	–	–	–	rw	rw	rw	rw

Field	Bits	Typ	Description
IEN	0	rw	**Interrupt Enable** If set, the interrupt signal of the IMB gets activated when ISR is set.
DIDTRP	1	rw	**Disable Instruction Fetch Double Bit Error Trap** If set, a double bit ECC error does not cause the replacement of the fetched data by a trap instruction.
DDDTRP	2	rw	**Disable Data Read Double Bit Error Trap** If set, a double bit ECC error during data read does not trigger the Flash access hardware trap.
DPROTRP	3	rw	**Disable Protection Trap** If set, a read request from read protected flash memory does not trigger the Flash access hardware trap.
ICLR	8	w	**Interrupt Clear** When written with 1 the ISR is cleared. Reading this bit delivers always 0. Writing a 0 is ignored.
ISET	9	w	**Interrupt Set** When written with 1 the ISR is set and if IEN is set the interrupt signal is activated. Reading this bit delivers always 0. Writing a 0 is ignored. When writing ISET and ICLR to 1 concurrently ISET takes priority so ISR is set.
PSERCLR	10	w	**Clear PSRAM Error Flag** When written with 1 the PSER is cleared. Reading this bit delivers always 0. Writing a 0 is ignored.
PSER	14	rh	**PSRAM Error Flag** This flag is set when write requests to the write protected or not implemented PSRAM range are detected. This flag can be cleared by writing 1 to PSERCLR.
ISR	15	rh	**Interrupt Service Request** If set, it indicates that at least one IMB_FSR.BUSY bit changed from 1 to 0. If IEN was set an interrupt request is sent to the interrupt controller. After servicing the interrupt the software handler clears this flag by writing a 1 to ICLR.

Flash State

Flash state. Split into 3 registers IMB_FSR_BUSY, IMB_FSR_OP, and IMB_FSR_PROT. The protection relevant fields or IMB_FSR_PROT are described in **"Protection Handling Details" on Page 3-38**.

The registers are reset by the Application Reset with the exception of "ERASE", "PROG", and "OPER". These three fields are only reset by a System Reset.

IMB_FSR_BUSY
Flash State Busy **ISFR (FF FF06$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
–	–	–	–	–	PAGE			–	–	–	–	–	BUSY		
–	–	–	–	–		rh		–	–	–	–	–		rh	

Field	Bits	Typ	Description
BUSY	[2:0]	rh	**Busy** A flash module is busy with a task. Each bit position corresponds to one of the 3 flash modules. The task is indicated by the bits MAR, POWER, ERASE or PROG of IMB_FSR_OP. BUSY is automatically cleared when the task has finished. The corresponding task indication is not cleared in order to allow an interrupt handler to determine the finished task.
PAGE	[10:8]	rh	**Page Mode Indication** Set as long the corresponding flash module is in page mode. Page mode is entered by the "**Enter Page Mode**" commands and finished by a "**Program Page**" command. The page mode can be also left by a "**Reset to Read**" command. Also an Application Reset clears this bit.

IMB_FSR_OP
Flash State Operations **ISFR (FF FF08$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–	–	–	OPER	SQER	MAR	POWER	ERASE	PROG
–	–	–	–	–	–	–	–	–	–	rh	rh	rh	rh	rh	rh

Field	Bits	Typ	Description
PROG	0	rh	**Program Task Indication** This bit is set when a program task is started. The affected flash module is indicated by a BUSY bit. The PROG bit is not automatically reset but must be cleared by a “**Clear Status**” command. This bit is not cleared by an Application Reset but only by a System Reset.
ERASE	1	rh	**Erase Task Indication** This bit is set when an erase task is started. The affected flash module is indicated by a BUSY bit. The ERASE bit is not automatically reset but must be cleared by a “**Clear Status**” command. This bit is not cleared by an Application Reset but only by a System Reset.
POWER	2	rh	**Power Change Indication** This bit indicates that a flash module is in its startup phase or in a shutdown phase. The BUSY bits indicate which flash module is busy. This bit is not automatically reset but must be cleared by a “**Clear Status**” command.
MAR	3	rh	**Margin Change Indication** If a read margin modification is requested this bit is set together with the corresponding BUSY bit. The BUSY bit is cleared when the margin change is effective and the flash module can be read again. The MAR bit must be cleared by a “**Clear Status**” command.
SQER	4	rh	**Sequence Error** This bit is set by a errored command sequence or a command that is not accepted. It is cleared by “**Clear Status**” and “**Reset to Read**”.

Field	Bits	Typ	Description
OPER	5	rh	**Operation Error** The IMB Core maintains internal bits that are set when starting a program or erase process. They are cleared when this process finishes. These bits are not reset by an Application Reset but only by a System Reset. If one of these bits is set after Application Reset the IMB Core sets OPER. So this signals that a running erase or program process was interrupted by an Application Reset. The OPER is cleared by “**Reset to Read**”, “**Clear Status**” or a System Reset.

IMB_FSR_PROT
Flash State Protection **ISFR (FF FF0A$_H$)** **Reset value: x000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RPRO		–	–	DDBER	DSBER	IDBER	ISBER	–	–	–	PROER	WPRODIS	RPRODIS	PROINER	PROIN
rh		–	–	rh	rh	rh	rh	–	–	–	rh	rh	rh	rh	rh

Field	Bits	Typ	Description
PROIN	0	rh	**Flash Protection Installed** Modified by the IMB Core. Cleared by Application Reset.
PROINER	1	rh	**Flash Protection Installation Error** Modified by the IMB Core. Cleared by Application Reset.
RPRODIS	2	rh	**Read Protection Disabled** The read protection was temporarily disabled with the “**Disable Read Protection**” command. Modified by the IMB Core. Cleared by Application Reset.
WPRODIS	3	rh	**Write Protection Disabled** The write protection was temporarily disabled with the “**Disable Write Protection**” command. Modified by the IMB Core. Cleared by Application Reset.

Field	Bits	Typ	Description
PROER	4	rh	**Protection Error** Set by a violation of the installed protection. Reset by the “**Clear Status**” and “**Reset to Read**” commands or an Application Reset.
ISBER	8	rh	**Instruction Fetch Single Bit Error** Set if during instruction fetch a single-bit ECC error was detected (and corrected). Reset by “**Clear Status**” or “**Reset to Read**” commands or an Application Reset.
IDBER	9	rh	**Instruction Fetch Double Bit Error** Set if during instruction fetch a double-bit ECC error was detected (and not corrected). Reset by “**Clear Status**” or “**Reset to Read**” commands or an Application Reset.
DSBER	10	rh	**Data Read Single Bit Error** Same as ISBER for data reads.
DDBER	11	rh	**Data Read Double Bit Error** Same as IDBER for data reads.
RPRO	[15:14]	rh	**Read Protection Configuration** This field is copied by the IMB Core from the corresponding field in the security page 0. After Application Reset read protection is activated.

Margin Control

Read margin control. Each field corresponds to one flash module. A hard read 0 detects not completely erased cells. These are read as “1”. A hard read 1 detects not completely programmed cells. These are read as “0”. Read margin changes are caused by the command sequence “**Change Read Margin**”. The resulting read margin is reflected in this status register.

The command sequences “**Program Page**”, “**Erase Sector**”, “**Erase Page**” and “**Erase Security Page**” resets the read margin back to “normal”. The same happens in case of a flash wake-up.

Reset by Application Reset.

IMB_MAR
Margin Control **ISFR (FF FF0C$_H$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	HREAD2			HREAD1			HREAD0		
–	–	–	–	–	–	–	rh			rh			rh		

Field	Bits	Typ	Description
HREAD0	[2:0]	rh	**Hard Read 0** Active read margin of flash module 0. “000”:Normal read. “001”:Hard read 0. “010”: Alternate hard read 0 (usually harder than 001). “101”:Hard read 1. “110”: Alternate hard read 1 (usually harder than 101). other codes:Reserved.
HREAD1	[5:3]	rh	**Hard Read 1** Same for flash module 1.
HREAD2	[8:6]	rh	**Hard Read 2** Same for flash module 2.

Protection Configuration

Protection configuration register of each implemented flash module. In XE16x PROCON0, PROCON1 and PROCON2 are implemented. PROCON0 is described below. PROCON1 (at address FF’FF12$_H$) and PROCON2 (at address FF’FF14$_H$) have the same functionality for the other two flash modules. The logical sector numbering is depicted in **Figure 3-6**.

Each bit of the PROCONs is related to a logical sector. If it is cleared the write access to the corresponding logical sector (this means to the range of physical sectors) is locked under the conditions that are documented in **“Protection Handling Details” on Page 3-38**. The PROCON registers are exclusively modified by the IMB Core.

Reset by Application Reset.

IMB_PROCONx (x=0-2)
Protection Configuration. **ISFR (FF FF10$_H$+2*x)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
–	–	–	–	–	–	S9U	S8U	S7U	S6U	S5U	S4U	S3U	S2U	S1U	S0U
–	–	–	–	–	–	rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Typ	Description
SsU (s=0-9)	s	rh	**Sector 0 to 9 Unlock** s: Logical sector s of flash module 0 is write-protected.

3.10.3 Error Reporting Summary

The Table 3-8 summarizes the types of detected errors and the possible reactions.

Table 3-8 IMB Error Reporting

Error	Reaction
Data read from PSRAM with parity error.	If PECON.PEENPS: HW trap (see Section 6.13).
Instruction fetch from PSRAM with parity error.	If PECON.PEENPS: HW trap (see Section 6.13).
Data read from flash memory with single bit error.	Silently corrected. Bit IMB_FSR.DSBER set.
Data read from flash memory with double bit error.	Bit IMB_FSR.DDBER set. If IMB_INTCTR.DDDTRP = 0: Flash access trap[1] and default data is delivered.
Instruction fetch from flash memory with single bit error.	Silently corrected. Bit IMB_FSR.ISBER set.
Instruction fetch from flash memory with double bit error.	Bit IMB_FSR.IDBER set. If IMB_INTCTR.DIDTRP = 0: “TRAP 15_D” delivered instead of corrupted data.
Data read from protected flash memory.	IMB_FSR.PROER set. If IMB_INTCTR.DPROTRP = 0: Flash access trap[1] and default data is delivered.
Instruction fetch from protected flash memory.	“TRAP 15_D” delivered.
Program/erase request of write protected flash range.	Only bit PROER in IMB_FSR set.
Data read or instruction fetch from busy flash memory.	Read access stalled until end of busy state.
Instruction fetch from ISFR addresses.	Default data (“TRAP 15_D”) delivered.
Data read from not implemented ISFRs.	Default data delivered.
Data writes to not implemented ISFRs.	Silently ignored.
Data read from not implemented address range.	Unpredictable. Mirrored data from other memories might be returned or default values.

Table 3-8 IMB Error Reporting (cont'd)

Error	Reaction
Instruction fetch from not implemented address range.	Unpredictable. Mirrored data from other memories might be returned or default values.
Data written to not implemented PSRAM or write protected PSRAM address range (both determined by IMB_IMBCTR.PSPROT).	Bit IMB_INTCTR.PSER set. Flash access trap[1] and no data is changed in the PSRAM.
Program or erase command targeting not implemented flash memory.	Unpredictable. Access is ignored[2] or mirrored into implemented flash memory[3].
Data read from powered-down flash modules.	Considered as access to not-implemented memory range. Default data or data from implemented flash modules will be returned.
Instruction fetch from powered-down flash modules.	Considered as access to not-implemented memory range. Default data ("TRAP 15_D") will be returned or data from implemented flash modules.
Program or erase command targeting powered-down flash modules.	Silently ignored[2].
Shutdown or power-down request received while the command sequence interpreter is waiting for the last words of a command sequence.	The command interpreter is reset and a "**Reset to Read**" command sequence is executed.

1) More information about the Flash Access Trap can be found in chapter "SCU".

2) Attention: when an access (i.e. MOV) is ignored, the command sequence interpreter will still wait for this outstanding MOV. So the next command sequence might cause a SQER because it delivers an unexpected MOV.

3) The flash protection can not be by-passed by accessing the reserved memory ranges.

3.11 Data Retention Memories

This section describes the usage of the special purpose data memories Stand-By RAM (SBRAM) and Marker Memory (MKMEM). Depending on the device not all of them are available. The XE16x contains:

- MKMEM.

4 Central Processing Unit (CPU)

Basic tasks of the Central Processing Unit (CPU) are to fetch and decode instructions, to supply operands for the Arithmetic and Logic unit (ALU) and the Multiply and Accumulate unit (MAC), to perform operations on these operands in the ALU and MAC, and to store the previously calculated results. As the CPU is the main engine of the XE16x microcontroller, it is also affected by certain actions of the peripheral subsystem.

Because a five-stage processing pipeline (plus 2-stage fetch pipeline) is implemented in the XE16x, up to five instructions can be processed in parallel. Most instructions of the XE16x are executed in one single clock cycle due to this parallelism.

This chapter describes how the pipeline works for sequential and branch instructions in general, and the hardware provisions which have been made to speed up execution of jump instructions in particular. General instruction timing is described, including standard timing, as well as exceptions.

While internal memory accesses are normally performed by the CPU itself, external peripheral or memory accesses are performed by a particular on-chip External Bus Controller (EBC) which is invoked automatically by the CPU whenever a code or data address refers to the external address space.

Whenever possible, the CPU continues operating while an external memory access is in progress. If external data are required but are not yet available, or if a new external memory access is requested by the CPU before a previous access has been completed, the CPU will be held by the EBC until the request can be satisfied. The EBC is described in a separate chapter.

The on-chip peripheral units of the XE16x work nearly independently of the CPU with a separate clock generator. Data and control information are interchanged between the CPU and these peripherals via Special Function Registers (SFRs).

Whenever peripherals need a non-deterministic CPU action, an on-chip Interrupt Controller compares all pending peripheral service requests against each other and prioritizes one of them. If the priority of the current CPU operation is lower than the priority of the selected peripheral request, an interrupt will occur.

There are two basic types of interrupt processing:

- **Standard interrupt processing** forces the CPU to save the current program status and return address on the stack before branching to the interrupt vector jump table.
- **PEC interrupt processing** steals only one machine cycle from the current CPU activity to perform a single data transfer via the on-chip Peripheral Event Controller (PEC).

System errors detected during program execution (hardware traps) and external non-maskable interrupts are also processed as standard interrupts with a very high priority.

In contrast to other on-chip peripherals, there is a closer conjunction between the watchdog timer and the CPU. If enabled, the watchdog timer expects to be serviced by

the CPU within a programmable period of time, otherwise it will reset the chip. Thus, the watchdog timer is able to prevent the CPU from going astray when executing erroneous code. After reset, the watchdog timer starts counting automatically but, it can be disabled via software, if desired.

In addition to its active operation state, the CPU can enter idle mode by executing the IDLE instruction. In idle mode the CPU stops program execution but still reacts to interrupt or PEC requests. Transition to the active state can be forced by an interrupt request or a reset.

A set of Special Function Registers is dedicated to the CPU core (CSFRs):

- CPU Status Indication and Control: **PSW, CPUCON1, CPUCON2**
- Code Access Control: **IP, CSP**
- Data Paging Control: **DPP0, DPP1, DPP2, DPP3**
- Global GPRs Access Control: **CP**
- System Stack Access Control: **SP, SPSEG, STKUN, STKOV**
- Multiply and Divide Support: **MDL, MDH, MDC**
- Indirect Addressing Offset: **QR0, QR1, QX0, QX1**
- MAC Address Pointers: **IDX0, IDX1**
- MAC Status Indication and Control: **MCW, MSW, MAH, MAL, MRW**
- ALU Constants Support: **ZEROS, ONES**

The CPU also uses CSFRs to access the General Purpose Registers (GPRs). Since all CSFRs can be controlled by any instruction capable of addressing the SFR/CSFR memory space, there is no need for special system control instructions.

However, to ensure proper processor operation, certain restrictions on the user access to some CSFRs must be imposed. For example, the instruction pointer (CSP, IP) cannot be accessed directly at all. These registers can only be changed indirectly via branch instructions. Registers PSW, SP, and MDC can be modified not only explicitly by the programmer, but also implicitly by the CPU during normal instruction processing.

Note: Note that any explicit write request (via software) to an CSFR supersedes a simultaneous modification by hardware of the same register.

All CSFRs may be accessed wordwise, or bytewise (some of them even bitwise). Reading bytes from word CSFRs is a non-critical operation. Any write operation to a single byte of a CSFR clears the non-addressed complementary byte within the specified CSFR.

Attention: Reserved CSFR bits must not be modified explicitly, and will always supply a read value of 0. If a byte/word access is preferred by the programmer or is the only possible access the reserved CSFR bits must be written with 0 to provide compatibility with future versions.

4.1 Components of the CPU

The high performance of the CPU results from the cooperation of several units which are optimized for their respective tasks (see **Figure 4-1**). **Prefetch Unit** and **Branch Unit** feed the pipeline minimizing CPU stalls due to instruction reads. The **Address Unit** supports sophisticated addressing modes avoiding additional instructions needed otherwise. **Arithmetic and Logic Unit** and **Multiply and Accumulate Unit** handle differently sized data and execute complex operations. **Three memory interfaces** and **Write Buffer** minimize CPU stalls due to data transfers.

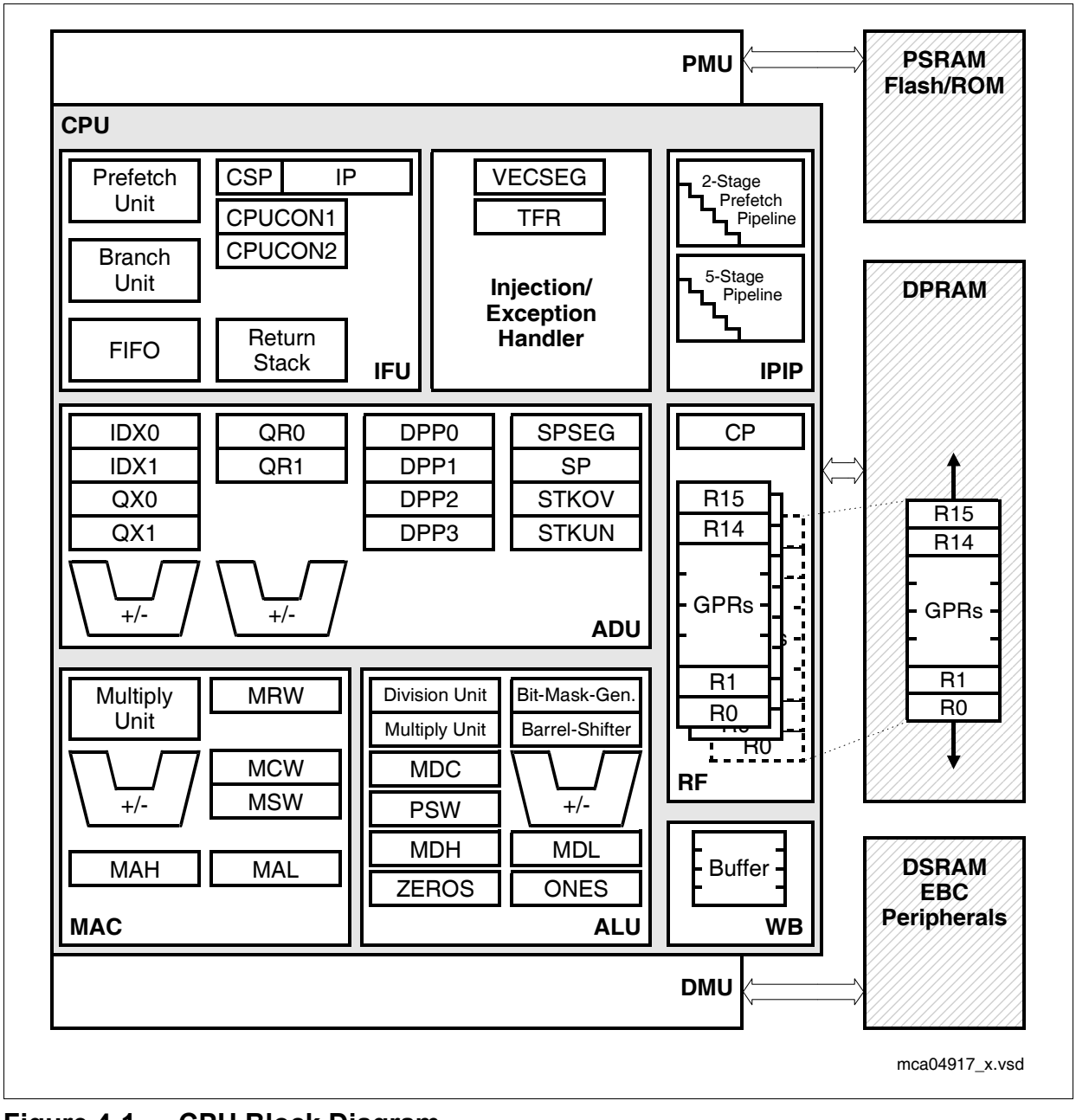


Figure 4-1 CPU Block Diagram

In general the instructions move through 7 pipeline stages, where each stage processes its individual task (see **Section 4.3** for a summary):

- the 2-stage fetch pipeline prefetches instructions from program memory and stores them into an instruction FIFO
- the 5-stage processing pipeline executes each instruction stored in the instruction FIFO

Because passing through one pipeline stage takes at least one clock cycle, any isolated instruction takes at least five clock cycles to be completed. Pipelining, however, allows parallel (i.e. simultaneous) processing of up to five instructions (with branches up to six instructions). Therefore, most of the instructions appear to be processed during one clock cycle as soon as the pipeline has been filled once after reset.

The pipelining increases the average instruction throughput considered over a certain period of time.

4.2 Instruction Fetch and Program Flow Control

The Instruction Fetch Unit (IFU) prefetches and preprocesses instructions to provide a continuous instruction flow. The IFU can fetch simultaneously at least two instructions via a 64-bit wide bus from the Program Management Unit (PMU). The prefetched instructions are stored in an instruction FIFO.

Preprocessing of branch instructions enables the instruction flow to be predicted. While the CPU is in the process of executing an instruction fetched from the FIFO, the prefetcher of the IFU starts to fetch a new instruction at a predicted target address from the PMU. The latency time of this access is hidden by the execution of the instructions which have already been buffered in the FIFO. Even for a non-sequential instruction execution, the IFU can generally provide a continuous instruction flow. The IFU contains two pipeline stages: the Prefetch Stage and the Fetch Stage.

During the prefetch stage, the Branch Detection and Prediction Logic analyzes up to three prefetched instructions stored in the first Instruction Buffer (can hold up to six instructions). If a branch is detected, then the IFU starts to fetch the next instructions from the PMU according to the prediction rules. After having been analyzed, up to three instructions are stored in the second Instruction Buffer (can hold up to three instructions) which is the input register of the Fetch Stage.

In the case of an incorrectly predicted instruction flow, the instruction fetch pipeline is bypassed to reduce the number of dead cycles.

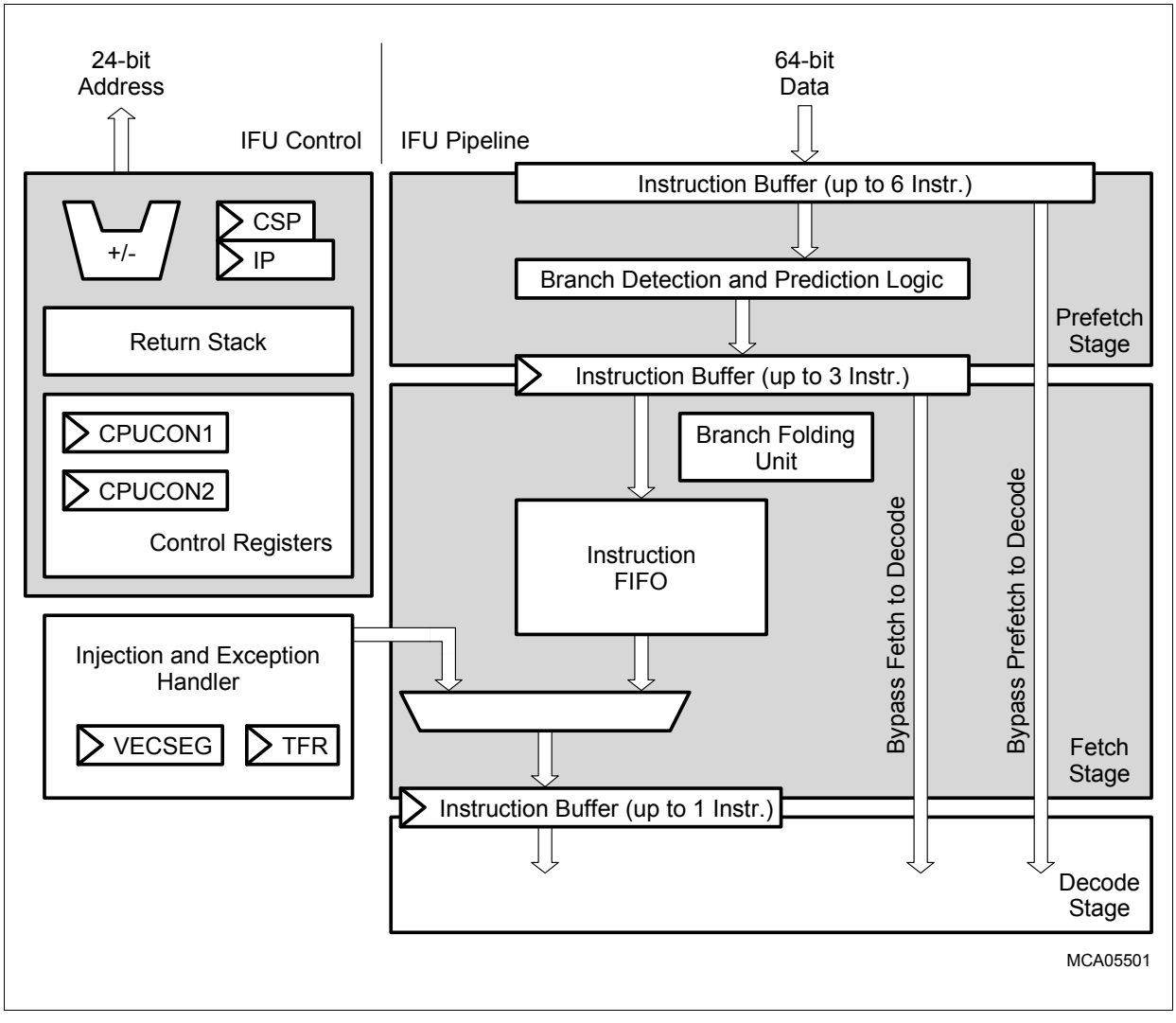


Figure 4-2 IFU Block Diagram

On the Fetch Stage, the prefetched instructions are stored in the instruction FIFO. The Branch Folding Unit (BFU) allows processing of branch instructions in parallel with preceding instructions. To achieve this the BFU preprocesses and reformats the branch instruction. First, the BFU defines (calculates) the absolute target address. This address — after being combined with branch condition and branch attribute bits — is stored in the same FIFO step as the preceding instruction. The target address is also used to prefetch the next instructions.

For the Processing Pipeline, both instructions are fetched from the FIFO again and are executed in parallel. If the instruction flow was predicted incorrectly (or FIFO is empty), the two stages of the IFU can be bypassed.

Note: Pipeline behavior in case of a incorrectly predicted instruction flow is described in the following sections.

4.2.1 Branch Detection and Branch Prediction Rules

The Branch Detection Unit preprocesses instructions and classifies detected branches. Depending on the branch class, the Branch Prediction Unit predicts the program flow using the following rules:

Table 4-1 Branch Classes and Prediction Rules

Branch Instruction Classes	Instructions	Prediction Rule (Assumption)
Inter-segment branch instructions	JMPS seg, caddr CALLS seg, caddr	The branch is always taken
Branch instructions with user programmable branch prediction	JMPA- xcc, caddr JMPA+ xcc, caddr CALLA- xcc, caddr CALLA+ xcc, caddr	User-specified[1] via bit 8 (‘a’) of the instruction long word: …+: branch ‘taken’ (a = 0) …-: branch ‘not taken’ (a = 1)
Indirect branch instructions	JMPI cc, [Rw] CALLI cc, [Rw]	Unconditional: branch ‘taken’ Conditional: ‘not taken’
Relative branch instructions with condition code	JMPR cc, rel	Unconditional or backward: branch ‘taken’ Conditional forward: ‘not taken’
Relative branch instructions without condition code	CALLR rel	The branch is always taken
Branch instructions with bit-condition	JB(C) bitaddr, rel JNB(S) bitaddr, rel	Backward: branch ‘taken’ Forward: ‘not taken’
Return instructions	RET, RETP RETS, RETI	The branch is always taken

1) This bit can be also set/cleared automatically by the Assembler for generic JMPA and CALLA instructions depending on the jump condition (condition is cc_Z: ‘not taken’, otherwise: ‘taken’).

4.2.2 Correctly Predicted Instruction Flow

Table 4-2 shows the continuous execution of instructions, assuming a 0-waitstate program memory. In this example, most of the instructions are executed in one CPU cycle while instruction I_{n+6} takes two CPU cycles (general example for multicycle instructions). The diagram shows the sequential instruction flow through the different pipeline stages. **Figure 4-3** shows the corresponding program memory section.
The instructions for the processing pipeline are fetched from the Instruction FIFO while the IFU prefetches the next instructions to fill the FIFO. As long as the instruction flow is correctly predicted by the IFU, both processes are independent.

In this example with a fast Internal Program Memory, the Prefetcher is able to fetch more instructions than the processing pipeline can execute. In T_{n+4}, the FIFO and prefetch buffer are filled and no further instructions can be prefetched. The PMU address stays

stable (T_{n+4}) until a whole 64-bit double word can be buffered (T_{n+7}) in the 96-bit prefetch buffer again.

Table 4-2 Correctly Predicted Instruction Flow (Sequential Execution)

	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}	T_{n+6}	T_{n+7}	T_{n+8}
PMU Address	I_{a+16}	I_{a+24}	I_{a+32}	I_{a+40}	I_{a+40}	I_{a+40}	I_{a+40}	I_{a+48}	I_{a+48}
PMU Data 64bit	I_{d+1}	I_{d+2}	I_{d+3}	I_{d+4}	I_{d+5}	I_{d+5}	I_{d+5}	I_{d+5}	I_{d+7}
PREFETCH 96-bit Buffer	I_{n+6} … I_{n+9}	I_{n+9} … I_{n+11}	I_{n+12} I_{n+13}	I_{n+14} I_{n+15}	I_{n+15} … I_{n+19}	I_{n+15} … I_{n+19}	I_{n+16} … I_{n+19}	I_{n+17} … I_{n+19}	I_{n+18} … I_{n+21}
FETCH Instruction Buffer	I_{n+5}	I_{n+6} I_{n+7} I_{n+8}	I_{n+9} I_{n+10} I_{n+11}	I_{n+12} I_{n+13}	I_{n+14}	–	I_{n+15}	I_{n+16}	I_{n+17}
FIFO contents	I_{n+3} … I_{n+5}	I_{n+4} … I_{n+8}	I_{n+5} … I_{n+11}	I_{n+6} … I_{n+13}	I_{n+7} … I_{n+14}	I_{n+7} … I_{n+14}	I_{n+8} … I_{n+15}	I_{n+9} … I_{n+16}	I_{n+10} … I_{n+17}
Fetch from FIFO	I_{n+4}	I_{n+5}	I_{n+6}	I_{n+7}	I_{n+7}	I_{n+8}	I_{n+9}	I_{n+10}	I_{n+11}
DECODE	I_{n+3}	I_{n+4}	I_{n+5}	I_{n+6}	I_{n+6}	I_{n+7}	I_{n+8}	I_{n+9}	I_{n+10}
ADDRESS	I_{n+2}	I_{n+3}	I_{n+4}	I_{n+5}	I_{n+6}	I_{n+6}	I_{n+7}	I_{n+8}	I_{n+9}
MEMORY	I_{n+1}	I_{n+2}	I_{n+3}	I_{n+4}	I_{n+5}	I_{n+6}	I_{n+6}	I_{n+7}	I_{n+8}
EXECUTE	I_n	I_{n+1}	I_{n+2}	I_{n+3}	I_{n+4}	I_{n+5}	I_{n+6}	I_{n+6}	I_{n+7}
WRITE BACK	–	I_n	I_{n+1}	I_{n+2}	I_{n+3}	I_{n+4}	I_{n+5}	I_{n+6}	I_{n+6}

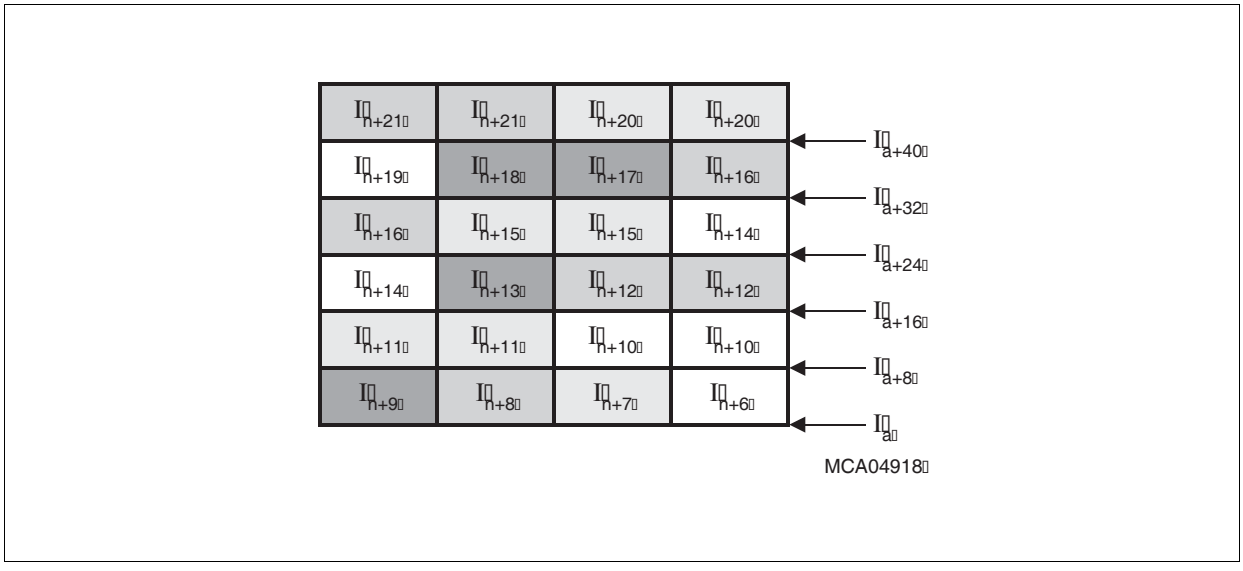


Figure 4-3 Program Memory Section for Correctly Predicted Flow

4.2.3 Incorrectly Predicted Instruction Flow

If the CPU detects that the IFU made an incorrect prediction of the instruction flow, then the pipeline stages and the Instruction FIFO containing the wrong prefetched instructions are canceled. The entire instruction fetch is restarted at the correct point of the program.

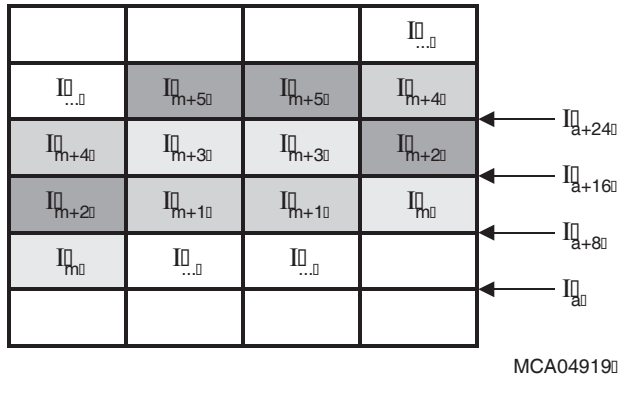
Table 4-3 shows the restarted execution of instructions, assuming a 0-waitstate program memory. **Figure 4-4** shows the corresponding program memory section.

During the cycle T_n, the CPU detects an incorrectly prediction case which leads to a canceling of the pipeline. The new address is transferred to the PMU in T_{n+1} which delivers the first data in the next cycle T_{n+2}. But, the target instruction crosses the 64-bit memory boundary and a second fetch in T_{n+3} is required to get the entire 32-bit instruction. In T_{n+4}, the Prefetch Buffer contains two 32-bit instructions while the first instruction I_m is directly forwarded to the Decode stage.

The prefetcher is now restarted and prefetches further instructions. In T_{n+5}, the instruction I_{m+1} is forwarded from the Fetch Instruction Buffer directly to the Decode stage as well. The Fetch row shows all instructions in the Fetch Instruction Buffer and the instructions fetched from the Instruction FIFO. The instruction I_{m+3} is the first instruction fetched from the FIFO during T_{n+6}. During the same cycle, instruction I_{m+2} was still forwarded from the Fetch Instruction Buffer to the Decode stage.

Table 4-3 Incorrectly Predicted Instruction Flow (Restarted Execution)

	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}	T_{n+6}	T_{n+7}	T_{n+8}
PMU Address	I…	I_a	I_{a+8}	I_{a+16}	I_{a+24}	I…	I…	I…	I…
PMU Data 64bit	I…	–	I_d	I_{d+1}	I_{d+2}	I_{d+3}	I…	I…	I…
PREFETCH 96-bit Buffer	I…	–	–	–	I_m I_{m+1}	I_{m+2} I_{m+3}	I_{m+4} I_{m+5}	I…	I…
FETCH Instruction Buffer	I_{next+2}	–	–	–	–	I_{m+1}	I_{m+2} I_{m+3}	I_{m+4} I_{m+5}	I…
Fetch from FIFO	–	–	–	–	–	–	I_{m+3}	I_{m+4}	I_{m+5}
DECODE	I_{next+1}	–	–	–	I_m	I_{m+1}	I_{m+2}	I_{m+3}	I_{m+4}
ADDRESS	I_{next}	–	–	–	–	I_m	I_{m+1}	I_{m+2}	I_{m+3}
MEMORY	I_{branch}	–	–	–	–	–	I_m	I_{m+1}	I_{m+2}
EXECUTE	I_n	I_{branch}	–	–	–	–	–	I_m	I_{m+1}
WRITE BACK	–	I_n	I_{branch}	–	–	–	–	–	I_m



64-bit wide Program Memory with four 16 bit packages

Figure 4-4 Program Memory Section for Incorrectly Predicted Flow

4.3 Instruction Processing Pipeline

The XE16x uses five pipeline stages to execute an instruction. All instructions pass through each of the five stages of the instruction processing pipeline. The pipeline stages are listed here together with the 2 stages of the fetch pipeline:

1st -> PREFETCH: This stage prefetches instructions from the PMU in the predicted order. The instructions are preprocessed in the branch detection unit to detect branches. The prediction logic decides if the branches are assumed to be taken or not.

2nd -> FETCH: The instruction pointer of the next instruction to be fetched is calculated according to the branch prediction rules. For zero-cycle branch execution, the Branch Folding Unit preprocesses and combines detected branches with the preceding instructions. Prefetched instructions are stored in the instruction FIFO. At the same time, instructions are transported out of the instruction FIFO to be executed in the instruction processing pipeline.

3rd -> DECODE: The instructions are decoded and, if required, the register file is accessed to read the GPR used in indirect addressing modes.

4th -> ADDRESS: All the operand addresses are calculated. Register SP is decremented or incremented for all instructions which implicitly access the system stack.

5th -> MEMORY: All the required operands are fetched.

6th -> EXECUTE: An ALU or MAC-Unit operation is performed on the previously fetched operands. The condition flags are updated. All explicit write operations to CPU-SFRs and all auto-increment/auto-decrement operations of GPRs used as indirect address pointers are performed.

7th -> WRITE BACK: All external operands and the remaining operands within the internal DPRAM space are written back. Operands located in the internal SRAM are buffered in the Write Back Buffer.

Specific so-called injected instructions are generated internally to provide the time needed to process instructions requiring more than one CPU cycle for processing. They are automatically injected into the decode stage of the pipeline, then they pass through the remaining stages like every standard instruction. Program interrupt, PEC transfer, and OCE operations are also performed by means of injected instructions. Although these internally injected instructions will not be noticed in reality, they help to explain the operation of the pipeline.

The performance of the CPU (pipeline) is decreased by bandwidth limitations (same resource is accessed by different stages) and data dependencies between instructions. The XE16x's CPU has dedicated hardware to detect and to resolve different kinds of dependencies. Some of those dependencies are described in the following section.

Because up to five different instructions are processed simultaneously, additional hardware has been dedicated to deal with dependencies which may exist between instructions in different pipeline stages. This extra hardware supports 'forwarding' of the operand read and write values and resolves most of the possible conflicts — such as

multiple usage of buses — in a time optimized way without performance loss. This makes the pipeline unnoticeable for the user in most cases. However, there are some rare cases in which the pipeline requires attention by the programmer. In these cases, the delays caused by the pipeline conflicts can be used for other instructions to optimize performance.

Note: The XE16x has a fully interlocked pipeline, which means that these conflicts do not cause any malfunction. Instruction re-ordering is only required for performance reasons.

The following examples describe the pipeline behavior in special cases and give principle rules to improve the performance by re-ordering the execution of instructions.

4.3.1 Pipeline Conflicts Using General Purpose Registers

The GPRs are the working registers of the CPU and there are a lot of possible dependencies between instructions using GPRs. A high-speed five-port register file prevents bandwidth conflicts. Dedicated hardware is implemented to detect and resolve the data dependencies. Special forwarding busses are used to forward GPR values from one pipeline stage to another. In most cases, this allows the execution of instructions without any delay despite of data dependencies.

```
Conflict_GPRs_Resolved:
In    ADD R0,R1      ;Compute new value for R0
In+1  ADD R3,R0      ;Use R0 again
In+2  ADD R6,R0      ;Use R0 again
In+3  ADD R6,R1      ;Use R6 again
In+4  ...
```

Table 4-4 Resolved Pipeline Dependencies Using GPRs

Stage	T_n	T_{n+1}	T_{n+2}	T_{n+3} 1)	T_{n+4} 2)	T_{n+5} 3)
DECODE	I_n = ADD R0, R1	I_{n+1} = ADD R3, R0	I_{n+2} = ADD R6, R0	I_{n+3} = ADD R6, R1	I_{n+4}	I_{n+5}
ADDRESS	I_{n-1}	I_n = ADD R0, R1	I_{n+1} = ADD R3, R0	I_{n+2} = ADD R6, R0	I_{n+3} = ADD R6, R1	I_{n+4}
MEMORY	I_{n-2}	I_{n-1}	I_n = ADD R0, R1	I_{n+1} = ADD R3, **R0**	I_{n+2} = ADD R6, **R0**	I_{n+3} = ADD **R6**, R1
EXECUTE	I_{n-3}	I_{n-2}	I_{n-1}	I_n = ADD **R0**, R1	I_{n+1} = ADD R3, R0	I_{n+2} = ADD **R6**, R0
WR.BACK	I_{n-4}	I_{n-3}	I_{n-2}	I_{n-1}	I_n = ADD **R0**, R1	I_{n+1} = ADD R3, R0

1) R0 forwarded from EXECUTE to MEMORY.

2) R0 forwarded from WRITE BACK to MEMORY.

3) R6 forwarded from EXECUTE to MEMORY.

However, if a GPR is used for indirect addressing the address pointer (i.e. the GPR) will be required already in the DECODE stage. In this case the instruction is stalled in the address stage until the operation in the ALU is executed and the result is forwarded to the address stage.

```
Conflict_GPRs_Pointer_Stall:
In    ADD R0,R1      ;Compute new value for R0
In+1  MOV R3,[R0]    ;Use R0 as address pointer
In+2  ADD R6,R0
In+3  ADD R6,R1
In+4  ...
```

Table 4-5 Pipeline Dependencies Using GPRs as Pointers (Stall)

Stage	T_n	T_{n+1}	T_{n+2} [1)]	T_{n+3} [2)]	T_{n+4}	T_{n+5}
DECODE	I_n = ADD R0, R1	I_{n+1} = MOV R3, [R0]	I_{n+2}	I_{n+2}	I_{n+2}	I_{n+3}
ADDRESS	I_{n-1}	I_n = ADD R0, R1	I_{n+1} = MOV R3, [R0]	I_{n+1} = MOV R3, [R0]	I_{n+1} = MOV R3, [**R0**]	I_{n+2}
MEMORY	I_{n-2}	I_{n-1}	I_n = ADD R0, R1	–	–	I_{n+1} = MOV R3, [R0]
EXECUTE	I_{n-3}	I_{n-2}	I_{n-1}	I_n = ADD **R0**, R1	–	–
WR.BACK	I_{n-4}	I_{n-3}	I_{n-2}	I_{n-1}	I_n = ADD R0, R1	–

1) New value of R0 not yet available.

2) R0 forwarded from EXECUTE to ADDRESS (next cycle).

To avoid these stalls, one multicycle instruction or two single cycle instructions may be inserted. These instructions must not update the GPR used for indirect addressing.

```
Conflict_GPRs_Pointer_NoStall:
I_n     ADD R0,R1       ;Compute new value for R0
I_n+1   ADD R6,R0       ;R0 is not updated, just read
I_n+2   ADD R6,R1
I_n+3   MOV R3,[R0]     ;Use R0 as address pointer
I_n+4   ...
```

Table 4-6 Pipeline Dependencies Using GPRs as Pointers (No Stall)

Stage	T_n	T_{n+1}	T_{n+2}	T_{n+3} 1)	T_{n+4}	T_{n+5}
DECODE	I_n = ADD R0, R1	I_{n+1} = ADD R6, R0	I_{n+2} = ADD R6, R1	I_{n+3} = MOV R3, [R0]	I_{n+4}	I_{n+5}
ADDRESS	I_{n-1}	I_n = ADD R0, R1	I_{n+1} = ADD R6, R0	I_{n+2} = ADD R6, R1	I_{n+3} = MOV R3, [**R0**]	I_{n+4}
MEMORY	I_{n-2}	I_{n-1}	I_n = ADD R0, R1	I_{n+1} = ADD R6, R0	I_{n+2} = ADD R6, R1	I_{n+3} = MOV R3, [R0]
EXECUTE	I_{n-3}	I_{n-2}	I_{n-1}	I_n = ADD **R0**, R1	I_{n+1} = ADD R6, R0	I_{n+2} = ADD R6, R1
WR.BACK	I_{n-4}	I_{n-3}	I_{n-2}	I_{n-1}	I_n = ADD R0, R1	I_{n+1} = ADD R6, R0

1) R0 forwarded from EXECUTE to ADDRESS (next cycle).

4.3.2 Pipeline Conflicts Using Indirect Addressing Modes

In the case of read accesses using indirect addressing modes, the Address Generation Unit uses a speculative addressing mechanism. The read data path to one of the different memory areas (DPRAM, DSRAM, etc.) is selected according to a history table before the address is decoded. This history table has one entry for each of the GPRs. The entries store the information of the last accessed memory area using the corresponding GPR. In the case of an incorrect prediction of the memory area, the read access must be restarted.

It is recommended that the GPRs used for indirect addressing always point to the same memory area. If an updated GPR points to a different memory area, the next read operation will access the wrong memory area. The read access must be repeated, which leads to pipeline stalls.

```
Conflict_GPRs_Pointer_WrongHistory:
I_n     ADD R3,[R0]      ;R0 points to DPRAM (e.g.)
I_n+1   MOV R0,R4
...
I_i     MOV DPPX, ...    ;change DPPx
...
I_m     ADD R6,[R0]      ;R0 now points to SRAM (e.g.)
I_m+1   MOV R6,R1
I_m+2   ...
```

Table 4-7 Pipeline Dependencies with Pointers (Valid Speculation)

Stage	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}
DECODE	I_n = ADD R3, [R0]	I_{n+1} = MOV R0, R4	I_{n+2}	I_{n+3}	I_{n+4}	I_{n+5}
ADDRESS	I_{n-1}	I_n = ADD R3, [R0]	I_{n+1} = MOV R0, R4	I_{n+2}	I_{n+3}	I_{n+4}
MEMORY	I_{n-2}	I_{n-1}	I_n = ADD R3, [R0]	I_{n+1} = MOV R0, R4	I_{n+2}	I_{n+3}
EXECUTE	I_{n-3}	I_{n-2}	I_{n-1}	I_n = ADD R3, [R0]	I_{n+1} = MOV R0, R4	I_{n+2}
WR.BACK	I_{n-4}	I_{n-3}	I_{n-2}	I_{n-1}	I_n = ADD R3, [R0]	I_{n+1} = MOV R0, R4

Table 4-8 Pipeline Dependencies with Pointers (Invalid Speculation)

Stage	T_m	T_{m+1}	T_{m+2} [1)]	T_{m+3}	T_{m+4}	T_{m+5}
DECODE	I_m = ADD R6, [R0]	I_{m+1} = MOV R6, R1	I_{m+1} = MOV R6, R1	I_{m+2}	I_{m+3}	I_{m+4}
ADDRESS	I_{m-1}	I_m = ADD R6, [R0]	I_m = ADD R6, [R0]	I_{m+1} = MOV R6, R1	I_{m+2}	I_{m+3}
MEMORY	I_{m-2}	I_{m-1}	–	I_m = ADD R6, [R0]	I_{m+1} = MOV R6, R1	I_{m+2}
EXECUTE	I_{m-3}	I_{m-2}	I_{m-1}	–	I_m = ADD R6, [R0]	I_{m+1} = MOV R6, R1
WR.BACK	I_{m-4}	I_{m-3}	I_{m-2}	I_{m-1}	–	I_m = ADD R6, [R0]

1) Access to location [R0] must be repeated due to wrong history (target area was changed).

4.3.3 Pipeline Conflicts Due to Memory Bandwidth

Memory bandwidth conflicts can occur if instructions in the pipeline access the same memory area at the same time. Special access mechanisms are implemented to minimize conflicts. The DPRAM of the CPU has two independent read/write ports; this allows parallel read and write operation without delays. Write accesses to the DSRAM can be buffered in a Write Back Buffer until read accesses are finished.

All instructions except the CoXXX instructions can read only one memory operand per cycle. A conflict between the read and one write access cannot occur because the DPRAM has two independent read/write ports. Only other pipeline stall conditions can generate a DPRAM bandwidth conflict. The DPRAM is a synchronous pipelined memory. The read access starts with the valid addresses on the address stage. The data are delivered in the Memory stage. If a memory read access is stalled in the Memory stage and the following instruction on the Address stage tries to start a memory read, the new read access must be delayed as well. But, this conflict is hidden by an already existing stall of the pipeline.

The CoXXX instructions are the only instructions able to read two memory operands per cycle. A conflict between the two read and one pending write access can occur if all three operands are located in the DPRAM area. This is especially important for performance in the case of executing a filter routine. One of the operands should be located in the DSRAM to guarantee a single-cycle execution of the CoXXX instructions.

```
Conflict_DPRAM_Bandwidth:
I_n     ADD op1,R1
I_n+1   ADD R6,R0
I_n+2   CoMAC [IDX0],[R0]
I_n+3   MOV R3,[R0]
I_n+4   ...
```

Table 4-9 Pipeline Dependencies in Case of Memory Conflicts (DPRAM)

Stage	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4} [1)]	T_{n+5}
DECODE	I_n = ADD op1, R1	I_{n+1} = ADD R6, R0	I_{n+2} = CoMAC …	I_{n+3} = MOV R3, [R0]	I_{n+4}	I_{n+4}
ADDRESS	I_{n-1}	I_n = ADD op1, R1	I_{n+1} = ADD R6, R0	I_{n+2} = CoMAC …	I_{n+3} = MOV R3, [R0]	I_{n+3} = MOV R3, [R0]
MEMORY	I_{n-2}	I_{n-1}	I_n = ADD op1, R1	I_{n+1} = ADD R6, R0	I_{n+2} = CoMAC …	I_{n+2} = CoMAC …
EXECUTE	I_{n-3}	I_{n-2}	I_{n-1}	I_n = ADD op1, R1	I_{n+1} = ADD R6, R0	–
WR.BACK	I_{n-4}	I_{n-3}	I_{n-2}	I_{n-1}	I_n = ADD op1, R1	I_{n+1} = ADD R6, R0

1) COMAC instruction stalls due to memory bandwidth conflict.

The DSRAM is a single-port memory with one read/write port. To reduce the number of bandwidth conflict cases, a Write Back Buffer is implemented. It has three data entries. Only if the buffer is filled and a read access and a write access occur at the same time, must the read access be stalled while one of the buffer entries is written back.

```
Conflict_DSRAM_Bandwidth:
In     ADD op1,R1
In+1   ADD R6,R0
In+2   ADD R6,op2
In+3   MOV R3,R2
In+4   ...
```

Table 4-10 Pipeline Dependencies in Case of Memory Conflicts (DSRAM)

Stage	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4} [1)]	T_{n+5}
DECODE	I_n = ADD op1, R1	I_{n+1} = ADD R6, R0	I_{n+2} = ADD R6, op2	I_{n+3} = MOV R3, R2	I_{n+4}	I_{n+4}
ADDRESS	I_{n-1}	I_n = ADD op1, R1	I_{n+1} = ADD R6, R0	I_{n+2} = ADD R6, op2	I_{n+3} = MOV R3, R2	I_{n+3} = MOV R3, R2
MEMORY	I_{n-2}	I_{n-1}	In = ADD op1, R1	I_{n+1} = ADD R6, R0	I_{n+2} = ADD R6, op2	I_{n+2} = ADD R6, op2
EXECUTE	I_{n-3}	I_{n-2}	I_{n-1}	I_n = ADD op1, R1	I_{n+1} = ADD R6, R0	–
WR.BACK	I_{n-4}	I_{n-3}	I_{n-2}	I_{n-1}	I_n = ADD op1, R1	I_{n+1} = ADD R6, R0
WB.Buffer	full	full	full	full	full	full

1) ADD R6, op2 instruction stalls due to memory bandwidth conflict.

4.3.4 Pipeline Conflicts Caused by CPU-SFR Updates

CPU-SFRs control the CPU functionality and behavior. Changes and updates of CSFRs influence the instruction flow in the pipeline. Therefore, special care is required to ensure that instructions in the pipeline always work with the correct CSFR values. CSFRs are updated late on the EXECUTE stage of the pipeline. Meanwhile, without conflict detection, the instructions in the DECODE, ADDRESS, and MEMORY stages would still work without updated register values. The CPU detects conflict cases and stalls the pipeline to guarantee a correct execution. For performance reasons, the CPU differentiates between different classes of CPU-SFRs. The flow of instructions through the pipeline can be improved by following the given rules used for instruction re-ordering.

There are three classes of CPU-SFRs:

- CSFRs not generating pipeline conflicts (ONES, ZEROS, MCW)
- CSFR result registers updated late in the EXECUTE stage, causing one stall cycle
- CSFRs affecting the whole CPU or the pipeline, causing canceling

CSFR Result Registers

The CSFR result registers MDH, MDL, MSW, MAH, MAL, and MRW of the ALU and MAC-Unit are updated late in the EXECUTE stage of the pipeline. If an instruction (except CoSTORE) accesses explicitly these registers in the memory stage, the value cannot be forwarded. The instruction must be stalled for one cycle on the MEMORY stage.

```
Conflict_CSFR_Update_Stall:
In     MUL R0,R1
In+1   MOV R6,MDL
In+2   ADD R6,R1
In+3   MOV R3,[R0]
In+4   ...
```

Table 4-11 Pipeline Dependencies with Result CSFRs (Stall)

Stage	T_n	T_{n+1}	T_{n+2}	T_{n+3} [1)]	T_{n+4}	T_{n+5}
DECODE	I_n = MUL R0, R1	I_{n+1} = MOV R6, MDL	I_{n+2} = ADD R6, R1	I_{n+3} = MOV R3, [R0]	I_{n+3} = MOV R3, [R0]	I_{n+4}
ADDRESS	I_{n-1}	I_n = MUL R0, R1	I_{n+1} = MOV R6, MDL	I_{n+2} = ADD R6, R1	I_{n+2} = ADD R6, R1	I_{n+3} = MOV R3, [R0]
MEMORY	I_{n-2}	I_{n-1}	I_n = MUL R0, R1	I_{n+1} = MOV R6, MDL	I_{n+1} = MOV R6, MDL	I_{n+2} = ADD R6, R1
EXECUTE	I_{n-3}	I_{n-2}	I_{n-1}	I_n = MUL R0, R1	–	I_{n+1} = MOV R6, MDL
WR.BACK	I_{n-4}	I_{n-3}	I_{n-2}	I_{n-1}	I_n = MUL R0, R1	–

1) Cannot read MDL here.

By reordering instructions, the bubble in the pipeline can be filled with an instruction not using this resource.

```
Conflict_CSFR_Update_Resolved:
In     MUL R0,R1
In+1   MOV R3,[R0]
In+2   MOV R6,MDL
In+3   ADD R6,R1
In+4   ...
```

Table 4-12 Pipeline Dependencies with Result CSFRs (No Stall)

Stage	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4} 1)	T_{n+5}
DECODE	I_n = MUL R0, R1	I_{n+1} = MOV R3, [R0]	I_{n+2} = MOV R6, MDL	I_{n+3} = ADD R6, R1	I_{n+4}	I_{n+5}
ADDRESS	I_{n-1}	I_n = MUL R0, R1	I_{n+1} = MOV R3, [R0]	I_{n+2} = MOV R6, MDL	I_{n+3} = ADD R6, R1	I_{n+4}
MEMORY	I_{n-2}	I_{n-1}	I_n = MUL R0, R1	I_{n+1} = MOV R3, [R0]	I_{n+2} = MOV R6, MDL	I_{n+3} = ADD R6, R1
EXECUTE	I_{n-3}	I_{n-2}	I_{n-1}	I_n = MUL R0, R1	I_{n+1} = MOV R3, [R0]	I_{n+2} = MOV R6, MDL
WR.BACK	I_{n-4}	I_{n-3}	I_{n-2}	I_{n-1}	I_n = MUL R0, R1	I_{n+1} = MOV R3, [R0]

1) MDL can be read now, no stall cycle necessary.

CSFRs Affecting the Whole CPU

Some CSFRs affect the whole CPU or the pipeline before the Memory stage. The CPU-SFRs CPUCON1, CP, SP, STKUN, STKOV, VECSEG, TFR, and PSW affect the overall CPU function, while the CPU-SFRs IDX0, IDX1, QX1, QX0, DPP0, DPP1, DPP2, and DPP3 only affect the DECODE, ADDRESS, and MEMORY stage when they are modified **explicitly**. In this case the pipeline behavior depends on the instruction and addressing mode used to modify the CSFR.
In the case of modification of these CSFRs by “POP CSFR” or by instructions using the reg,#data16 addressing mode, a special mechanism is implemented to improve performance during the initialization.

For further explanation, the instruction which modifies the CSFR can be called “instruction_modify_CSFR”. This special case is detected in the DECODE stage when the instruction_modify_CSFR enters the processing pipeline. Further on, instructions described in the following list are held in the DECODE stage (all other instructions are not held):

- Instructions using long addressing mode (mem)
- Instructions using indirect addressing modes ([R_w], [R_w+]…), except JMPI and CALLI
- ENWDT, DISWDT, EINIT
- All CoXXX instructions

If the CPUCON1, CP, SP, STKUN, STKOV, VECSEG, TFR, or the PSW are modified and the instruction_modify_CSFR reaches the EXECUTE stage, the pipeline is canceled. The modification affects the entire pipeline and the instruction prefetch. A clean cancel and restart mechanism is required to guarantee a correct instruction flow. In case of modification of IDX0, IDX1, QX1, QX0, DPP0, DPP1, DPP2, or DPP3 only the DECODE, ADDRESS, and MEMORY stages are affected and the pipeline needs not to be canceled. The modification does not affect the instructions in the ADDRESS, MEMORY stage because they are not using this resource. Other kinds of instructions are held in the DECODE stage until the CSFR is modified.

The following example shows a case in which the pipeline is stalled. The instruction “MOV R6, R1” after the “MOV IDX1, #12” instruction which modifies the CSFR will be held in DECODE Stage until the IDX1 register is updated. The next example shows an optimized initialization routine.

```
Conflict_Canceling:
In     MOV IDX1,#12
In+1   MOV R6,mem
In+2   ADD R6,R1
In+3   MOV R3,[R0]
```

Table 4-13 Pipeline Dependencies with Control CSFRs (Canceling)

Stage	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}
DECODE	I_n = MOV IDX1, #12	I_{n+1} = MOV R6, mem	I_{n+1} = MOV R6, mem	I_{n+1} = MOV R6, mem	I_{n+1} = MOV R6, mem	I_{n+2} = ADD R6, R1
ADDRESS	I_{n-1}	I_n = MOV IDX1, #12	–	–	–	I_{n+1} = MOV R6, mem
MEMORY	I_{n-2}	I_{n-1}	I_n = MOV IDX1, #12	–	–	–
EXECUTE	I_{n-3}	I_{n-2}	I_{n-1}	I_n = MOV IDX1, #12	–	–
WR.BACK	I_{n-4}	I_{n-3}	I_{n-2}	I_{n-1}	I_n = MOV IDX1, #12	–

```
Conflict_Canceling_Optimized:
In     MOV IDX1,#12
In+1   MOV MAH,#23
In+2   MOV MAL,#25
In+3   MOV R3,#08
In+4   ...
```

Table 4-14 Pipeline Dependencies with Control CSFRs (Optimized)

Stage	T_n	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}
DECODE	I_n = MOV IDX1, #12	I_{n+1} = MOV MAH, #23	I_{n+2} = MOV MAL, #25	I_{n+3} = MOV R3, #08	I_{n+4}	I_{n+5}
ADDRESS	I_{n-1}	I_n = MOV IDX1, #12	I_{n+1} = MOV MAH, #23	I_{n+2} = MOV MAL, #25	I_{n+3} = MOV R3, #08	I_{n+4}
MEMORY	I_{n-2}	I_{n-1}	I_n = MOV IDX1, #12	I_{n+1} = MOV MAH, #23	I_{n+2} = MOV MAL, #25	I_{n+3} = MOV R3, #08
EXECUTE	I_{n-3}	I_{n-2}	I_{n-1}	I_n = MOV IDX1, #12	I_{n+1} = MOV MAH, #23	I_{n+2} = MOV MAL, #25
WR.BACK	I_{n-4}	I_{n-3}	I_{n-2}	I_{n-1}	I_n = MOV IDX1, #12	I_{n+1} = MOV MAH, #23

For all the other instructions that modify this kind of CSFR, a simple stall and cancel mechanism guarantees the correct instruction flow.
A possible explicit write-operation to this kind of CSFRs is detected on the MEMORY stage of the pipeline. The following instructions on the ADDRESS and DECODE Stage are stalled. If the instruction reaches the EXECUTE stage, the entire pipeline and the Instruction FIFO of the IFU are canceled. The instruction flow is completely re-started.

```
Conflict_Canceling_Completely:
I_n     MOV PSW,R4
I_n+1   MOV R6,R1
I_n+2   ADD R6,R1
I_n+3   MOV R3,[R0]
I_n+4   ...
```

Table 4-15 Pipeline Dependencies with Control CSFRs (Cancel All)

Stage	T_{n+1}	T_{n+2}	T_{n+3}	T_{n+4}	T_{n+5}	T_{n+6}
DECODE	I_{n+1} = MOV R6, R1	I_{n+2} = ADD R6, R1	I_{n+2} = ADD R6, R1	–	–	I_{n+1} = MOV R6, R1
ADDRESS	I_n = MOV PSW, R4	I_{n+1} = MOV R6, R1	I_{n+1} = MOV R6, R1	–	–	–
MEMORY	I_{n-1}	I_n = MOV PSW, R4	–	–	–	–
EXECUTE	I_{n-2}	I_{n-1}	I_n = MOV PSW, R4	–	–	–
WR.BACK	I_{n-3}	I_{n-2}	I_{n-1}	I_n = MOV PSW, R4	–	–

4.4 CPU Configuration Registers

The CPU configuration registers select a number of general features and behaviors of the XE16x's CPU core. In general, these registers must not be modified by application software (exceptions will be documented, e.g. in an errata sheet).

Note: The CPU configuration registers are protected by the register security mechanism after the EINIT instruction has been executed.

CPUCON1
CPU Control Register 1 **SFR ($FE18_H/0C_H$)** **Reset Value: 0007_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	VECSC		WDT CTL	SGT DIS	INTS CXT	BP	ZCJ
-	-	-	-	-	-	-	-	-	rw		rw	rw	rw	rw	rw

Field	Bits	Type	Description
VECSC	[6:5]	rw	**Scaling Factor of Vector Table** 00 Space between two vectors is 2 words[1)] 01 Space between two vectors is 4 words 10 Space between two vectors is 8 words 11 Space between two vectors is 16 words
WDTCTL	4	rw	**Configuration of Watchdog Timer** 0 DISWDT executable only until End Of Init[2)] 1 DISWDT/ENWDT always executable (enhanced WDT mode)
SGTDIS	3	rw	**Segmentation Disable/Enable Control** 0 Segmentation enabled 1 Segmentation disabled
INTSCXT	2	rw	**Enable Interruptibility of Switch Context** 0 Switch context is not interruptible 1 Switch context is interruptible
BP	1	rw	**Enable Branch Prediction Unit** 0 Branch prediction disabled 1 Branch prediction enabled
ZCJ	0	rw	**Enable Zero Cycle Jump Function** 0 Zero cycle jump function disabled 1 Zero cycle jump function enabled

1) The default value (2 words) is compatible with the vector distance defined in the C166 Family architecture.

2) The DISWDT (executed after EINIT) and ENWDT instructions are internally converted in a NOP instruction.

CPUCON2
CPU Control Register 2 **SFR ($FE1A_H/0D_H$)** **Reset Value: $8FBB_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FIFODEPTH				FIFOFED		BYPPF	BYPF	EIOIAEN	STEN	LFIC	OVRUN	RETST	-	DAID	SL
rw				rw		rw	rw	rw	rw	rw	rw	rw	-	rw	rw

Field	Bits	Type	Description
FIFODEPTH	[15:12]	rw	**FIFO Depth Configuration** 0000 No FIFO (entries) 0001 One FIFO entry … … 1000 Eight FIFO entries 1001 reserved … … 1111 reserved
FIFOFED	[11:10]	rw	**FIFO Fed Configuration** 00 FIFO disabled 01 FIFO filled with up to one instruction per cycle 10 FIFO filled with up to two instructions per cycle 11 FIFO filled with up to three instruction per cycle
BYPPF	9	rw	**Prefetch Bypass Control** 0 Bypass path from prefetch to decode disabled 1 Bypass path from prefetch to decode available
BYPF	8	rw	**Fetch Bypass Control** 0 Bypass path from fetch to decode disabled 1 Bypass path from fetch to decode available
EIOIAEN	7	rw	**Early IO Injection Acknowledge Enable** 0 Injection acknowledge by destructive read not guaranteed 1 Injection acknowledge by destructive read guaranteed
STEN	6	rw	**Stall Instruction Enable** (for debug purposes) 0 Stall Instruction disabled 1 Stall Instruction enabled (see example below)
LFIC	5	rw	**Linear Follower Instruction Cache** 0 Linear Follower Instruction Cache disabled 1 Linear Follower Instruction Cache enabled

Field	Bits	Type	Description
OVRUN	4	rw	**Pipeline Control** 0 Overrun of pipeline bubbles not allowed 1 Overrun of pipeline bubbles allowed
RETST	3	rw	**Enable Return Stack** 0 Return Stack is disabled 1 Return Stack is enabled
DAID	1	rw	**Disable Atomic Injection Deny** 0 Injection-requests are denied during Atomic 1 Injection-requests are not denied during Atomic
SL	0	rw	**Enables Short Loop Mode** 0 Short loop mode disabled 1 Short loop mode enabled

Example for dedicated stall debug instructions:

```
STALLAM da,ha,dm,hm  ;Opcode: 44 dahadmhm
STALLEW de,he,dw,hw  ;Opcode: 45 dehedwhw
                     ;Stalls the corresponding pipeline
                     ;stage after “d” cycles for “h” cycles
                     ;(“d” and “h” are 6-bit values)
```

Note: In general, these registers must not be modified by application software (exceptions will be documented, e.g. in an errata sheet).

4.5 Use of General Purpose Registers

The CPU uses several banks of sixteen dedicated registers R0, R1, R2, … R15, called General Purpose Registers (GPRs), which can be accessed in one CPU cycle. The GPRs are the working registers of the arithmetic and logic units and many also serve as address pointers for indirect addressing modes.

The register banks are accessed via the 5-port register file providing the high access speed required for the CPU's performance. The register file is split into three independent physical register banks. There are **two types of register banks**:

- **Two local register banks** which are a part of the register file
- **A global register bank** which is memory-mapped and cached in the register file

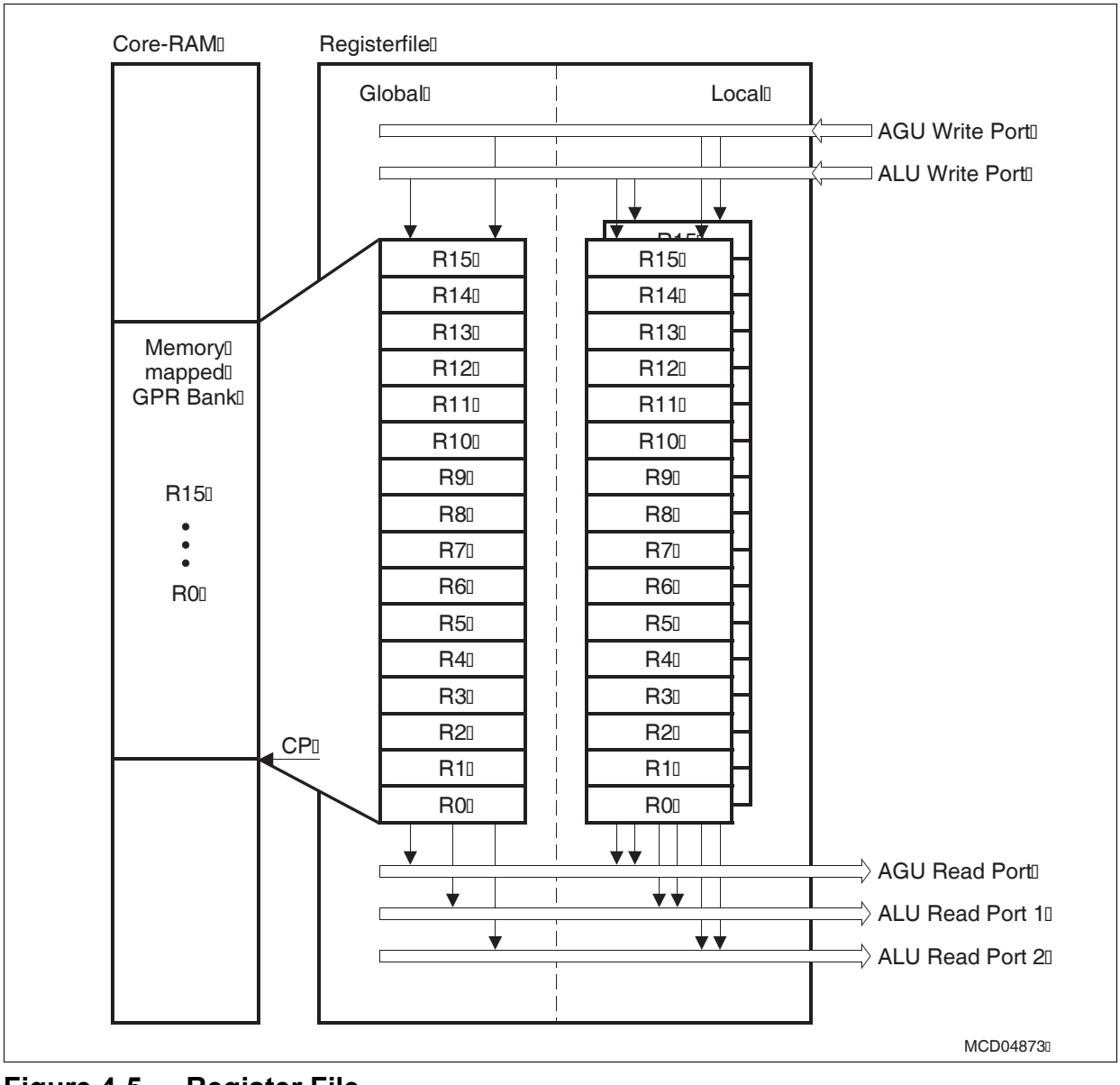


Figure 4-5 Register File

Bitfield BANK in register PSW selects which of the three physical register banks is activated. The selected bank can be changed explicitly by any instruction which writes to the PSW, or implicitly by a RETI instruction, an interrupt or hardware trap. In case of an interrupt, the selection of the register bank is configured via registers BNKSELx in the Interrupt Controller ITC. Hardware traps always use the global register bank.

The local register banks are built of dedicated physical registers, while the global register bank represents a cache. The banks of the memory-mapped GPRs (global bank) are located in the internal DPRAM. One bank uses a block of 16 consecutive words. A Context Pointer (CP) register determines the base address of the current selected bank. To provide the required access speed, the GPRs located in the DPRAM are cached in the 5-port register file (only one memory-mapped GPR bank can be cached at the time). If the global register bank is activated, the cache will be validated before further instructions are executed. After validation, all further accesses to the GPRs are redirected to the global register bank.

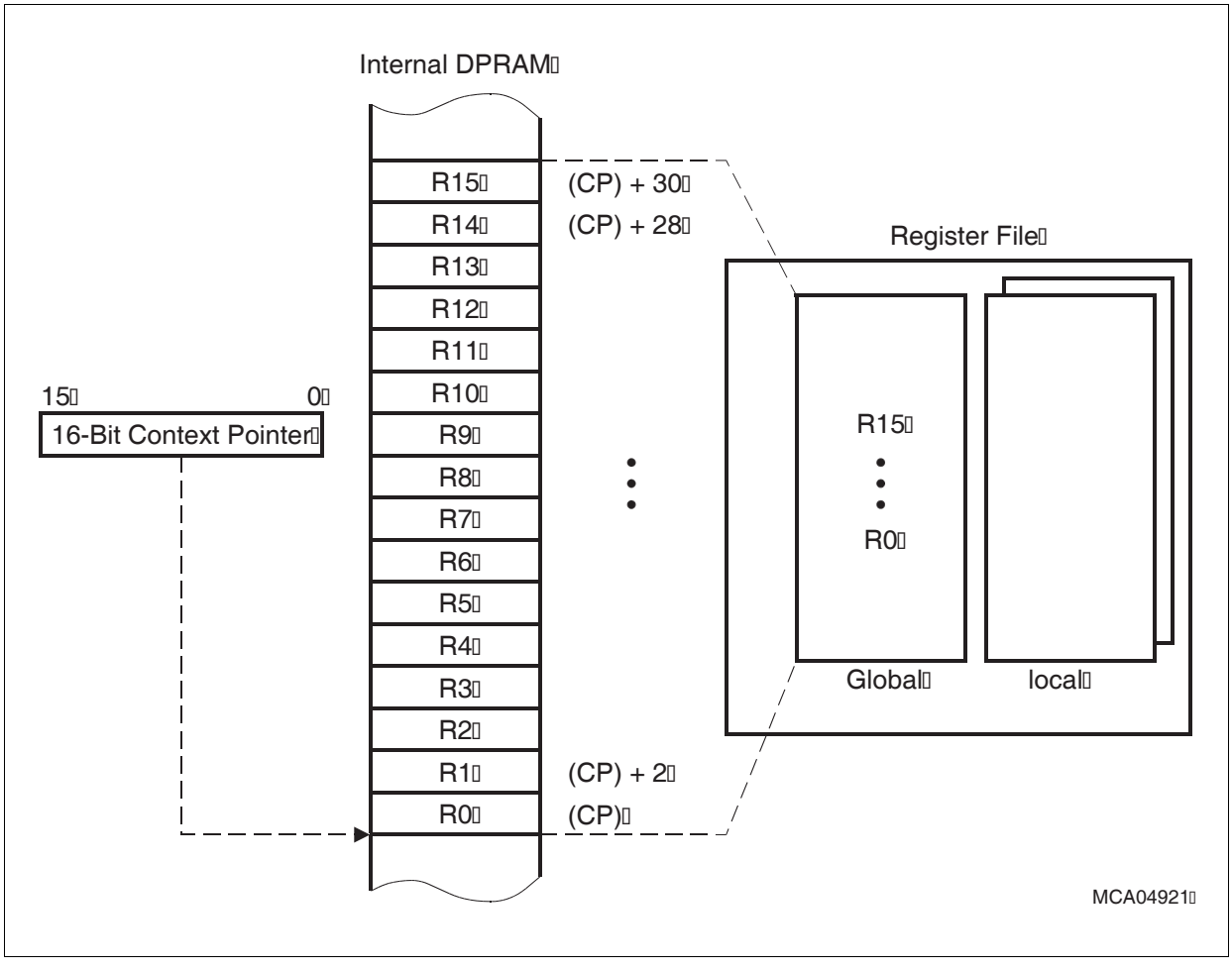


Figure 4-6 Register Bank Selection via Register CP

4.5.1 GPR Addressing Modes

Because the GPRs are the working registers and are accessed frequently, there are three possible ways to access a register bank:

- **Short GPR Address** (mnemonic: Rw or Rb)
- **Short Register Address** (mnemonic: reg or bitoff)
- **Long Memory Address** (mnemonic: mem), for the global bank only

Short GPR Addresses specify the register offset within the current register bank (selected via bitfield BANK). Short 4-bit GPR addresses can access all sixteen registers, short 2-bit addresses (used by some instructions) can access the lower four registers.

Depending on whether a relative word (Rw) or byte (Rb) GPR address is specified, the short GPR address is either multiplied by two (Rw) or not (Rb) before it is used to physically access the register bank. Thus, both byte and word GPR accesses are possible in this way.

Note: GPRs used as indirect address pointers are always accessed wordwise.

For the local register banks the resulting offset is used directly, for the global register bank the resulting offset is logically added to the contents of register CP which points to the memory location of the base of the current global register bank (see **Figure 4-7**).

Short 8-Bit Register Addresses within a range from $F0_H$ to FF_H interpret the four least significant bits as short 4-bit GPR addresses, while the four most significant bits are ignored. The respective physical GPR address is calculated in the same way as for short 4-bit GPR addresses. For single bit GPR accesses, the GPR's word address is calculated in the same way. The accessed bit position within the word is specified by a separate additional 4-bit value.

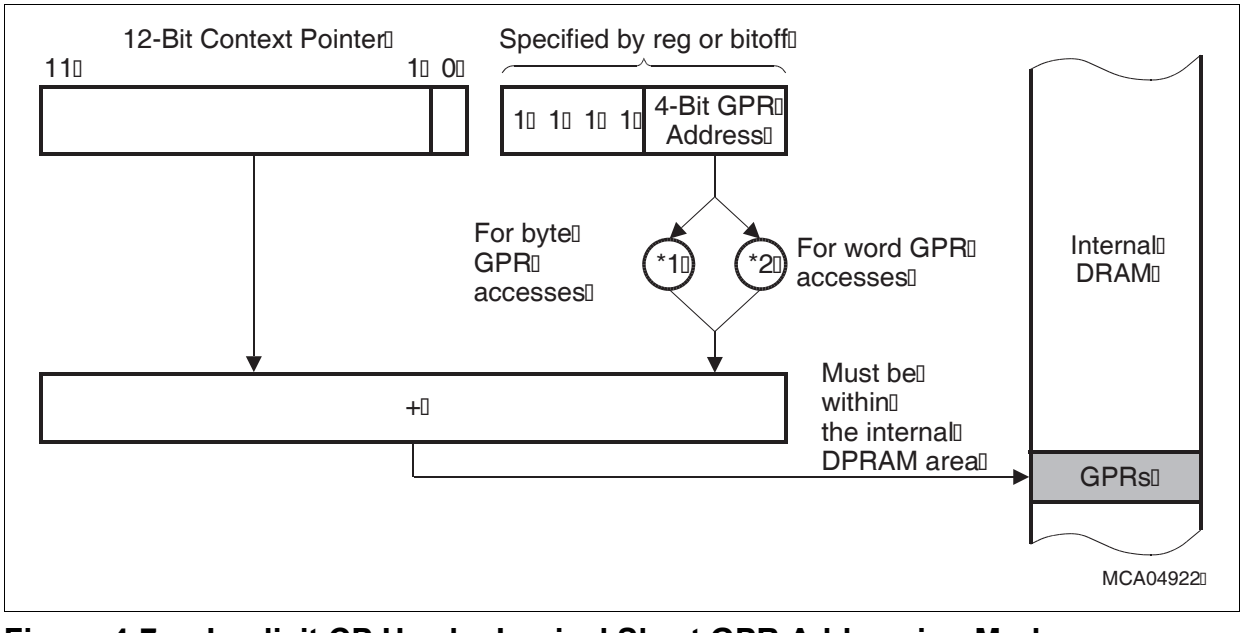


Figure 4-7 Implicit CP Use by Logical Short GPR Addressing Modes

24-Bit Memory Addresses can be directly used to access GPRs located in the DPRAM (not applicable for local register banks). In case of a memory read access, a hit detection logic checks if the accessed memory location is cached in the global register bank. In case of a cache hit, an additional global register bank read access is initiated. The data that is read from cache will be used and the data that is read from memory will be discarded. This leads to a delay of one CPU cycle (MOV R4, **mem** [CP ≤ mem ≤ CP + 31]). In case of a memory write access, the hit detection logic determines a cache hit in advance. Nevertheless, the address conversion needs one additional CPU cycle. The value is directly written into the global register bank without further delay (MOV **mem**, R4).

Note: The 24-bit GPR addressing mode is not recommended because it requires an extra cycle for the read and write access.

Table 4-16 Addressing Modes to Access GPRs

Word Registers[1]		Byte Registers		Short Address[2]		
Name	**Mem. Addr.[3]**	**Name**	**Mem. Addr.[3]**	**8-Bit**	**4-Bit**	**2-Bit**
R0	(CP) + 0	RL0	(CP) + 0	$F0_H$	0_H	0_H
R1	(CP) + 2	RH0	(CP) + 1	$F1_H$	1_H	1_H
R2	(CP) + 4	RL1	(CP) + 2	$F2_H$	2_H	2_H
R3	(CP) + 6	RH1	(CP) + 3	$F3_H$	3_H	3_H
R4	(CP) + 8	RL2	(CP) + 4	$F4_H$	4_H	---
R5	(CP) + 10	RH2	(CP) + 5	$F5_H$	5_H	---
R6	(CP) + 12	RL3	(CP) + 6	$F6_H$	6_H	---
R7	(CP) + 14	RH3	(CP) + 7	$F7_H$	7_H	---
R8	(CP) + 16	RL4	(CP) + 8	$F8_H$	8_H	---
R9	(CP) + 18	RH4	(CP) + 9	$F9_H$	9_H	---
R10	(CP) + 20	RL5	(CP) + 10	FA_H	A_H	---
R11	(CP) + 22	RH5	(CP) + 11	FB_H	B_H	---
R12	(CP) + 24	RL6	(CP) + 12	FC_H	C_H	---
R13	(CP) + 26	RH6	(CP) + 13	FD_H	D_H	---
R14	(CP) + 28	RL7	(CP) + 14	FE_H	E_H	---
R15	(CP) + 30	RH7	(CP) + 15	FF_H	F_H	---

1) The first 8 GPRs (R7 … R0) may also be accessed bytewise. Writing to a GPR byte does not affect the other byte of the respective GPR.

2) Short addressing modes are usable for all register banks.

3) Long addressing mode only usable for the memory mapped global GPR bank.

4.5.2 Context Switching

When a task scheduler of an operating system activates a new task or an interrupt service routine is called or terminated, the working context (i.e. the registers) of the left task must be saved and the working context of the new task must be restored. The CPU context can be changed in two ways:

- Switching the selected register bank
- Switching the context of the global register

Switching the Selected Physical Register Bank

By updating bitfield BANK in register PSW the active register bank is switched immediately. It is possible to switch between the current memory-mapped GPR bank cached in the global register bank (BANK = 00_B), local register bank 1 (BANK = 10_B), and local register bank 2 (BANK = 11_B).

In case of an interrupt service, the bank switch can be automatically executed by updating bitfield BANK from registers BNKSELx in the interrupt controller. By executing a RETI instruction, bitfield BANK will automatically be restored and the context will switched to the original register bank.

The switch between the three physical register banks of the register file can also be executed by writing to bitfield BANK. Because of pipeline dependencies an explicit change of register PSW must cancel the pipeline.

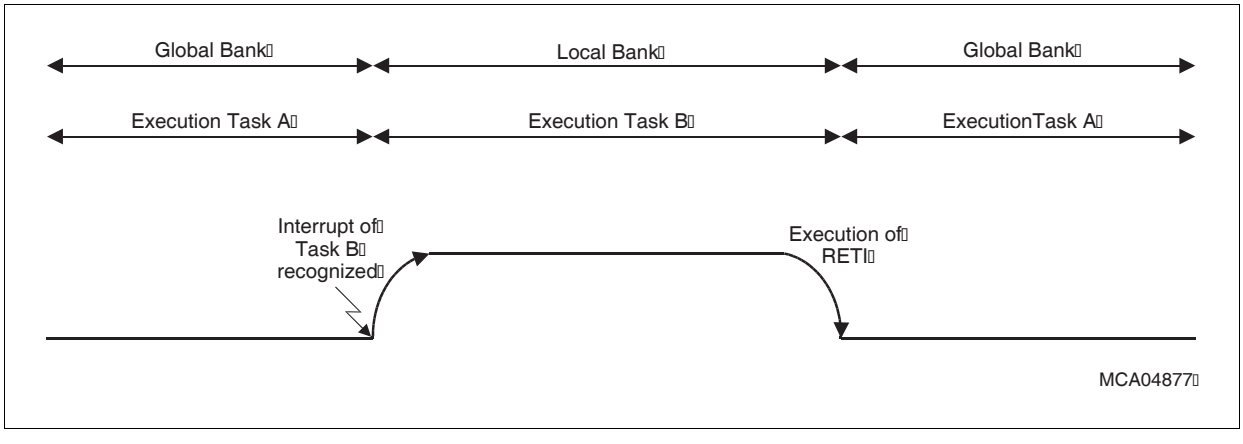


Figure 4-8 Context Switch by Changing the Physical Register Bank

After a switch to a local register bank, the new bank is immediately available. After switching to the global register bank, the cached memory-mapped GPRs must be valid before any further instructions can be executed. If the global register bank is not valid at this time (in case if the context switch process has been interrupted), the cache validation process is repeated automatically.

Switching the Context of the Global Register Bank

The contents of the global register bank are switched by changing the base address of the memory-mapped GPR bank. The base address is given by the contents of the Context Pointer (CP).

After the CP has been updated, a state machine starts to store the old contents of the global register bank and to load the new one. The store and load algorithm is executed in nineteen CPU cycles: the execution of the cache validation process takes sixteen cycles plus three cycles to stall an instruction execution to avoid pipeline conflicts upon the completion of the validation process. The context switch process has two phases:

- **Store phase:** The contents of the global register bank[1)] is stored back into the DPRAM by executing eight injected STORE instructions. After the last STORE instruction the contents of the global register bank are invalidated.
- **Load phase:** The global register bank is loaded with the new context by executing eight injected LOAD instructions. After the last LOAD instruction the contents of the global register bank are validated.

The code execution is stopped until the global register bank is valid again. A hardware interrupt can occur during the validation process. The way the validation process is completed depends on the type of register bank selected for this interrupt:

- If the interrupt also uses a global register bank the validation process is finished before executing the service routine (see **Figure 4-9**).
- If the interrupt uses a local register bank the validation process is interrupted and the service routine is executed immediately (see **Figure 4-10**). After switching back to the global register bank, the validation process is finished:
  - If the interrupt occurred during the store phase, the entire validation process is restarted from the very beginning.
  - If the interrupt occurred during the load phase, only the load phase is repeated.

If a local-bank interrupt routine (Task B in **Figure 4-11**) is again interrupted by a global-bank interrupt (Task C), the suspended validation process must be finished before code of Task C can be executed. This means that the validation process of Task A does not affect the interrupt latency of Task B but the latency of Task C.

*Note: If Task C would immediately interrupt Task A, the register bank validation process of Task A would be finished first. The worst case interrupt latency is identical in both cases (see **Figure 4-9** and **Figure 4-11**).*

1) During the store phase of the context switch the complete register bank is written to the DPRAM even if the application only uses a part of this register bank. A register bank must not be located above $FDE0_H$, otherwise the store phase will overwrite SFRs (beginning at $FE00_H$).

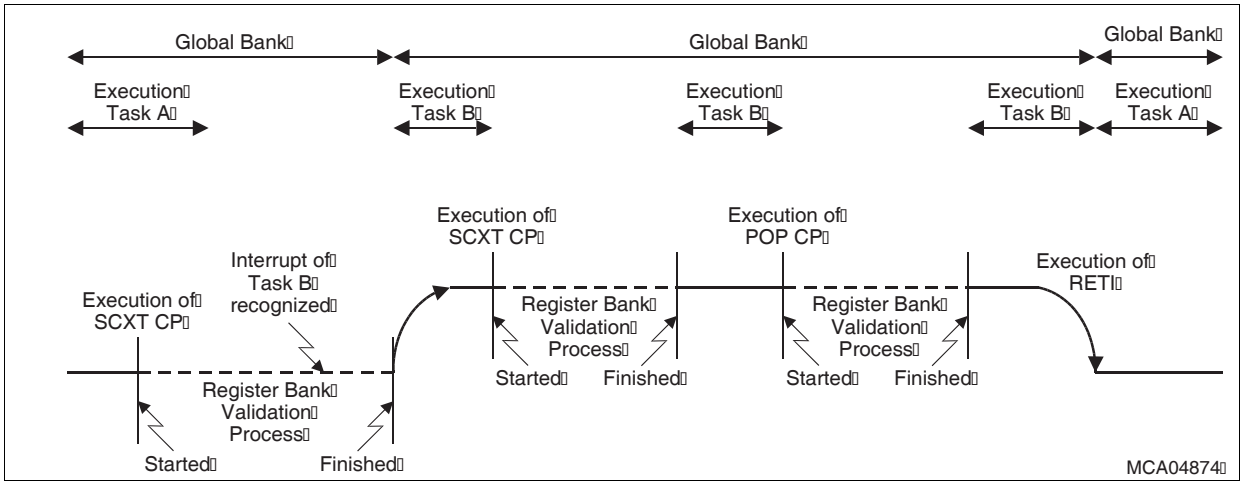


Figure 4-9 Validation Process Interrupted by Global-Bank Interrupt

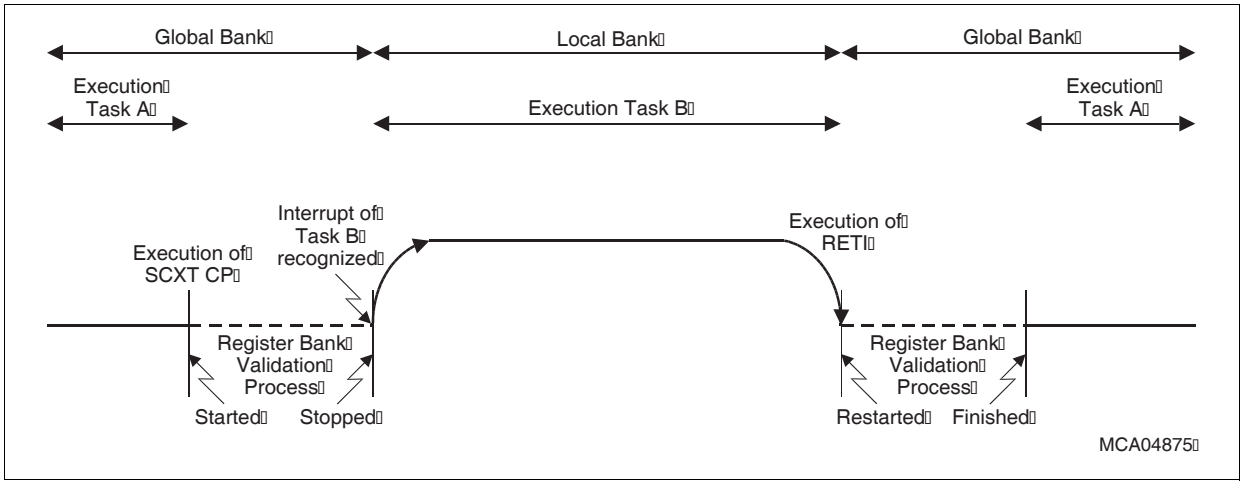


Figure 4-10 Validation Process Interrupted by Local-Bank Interrupt

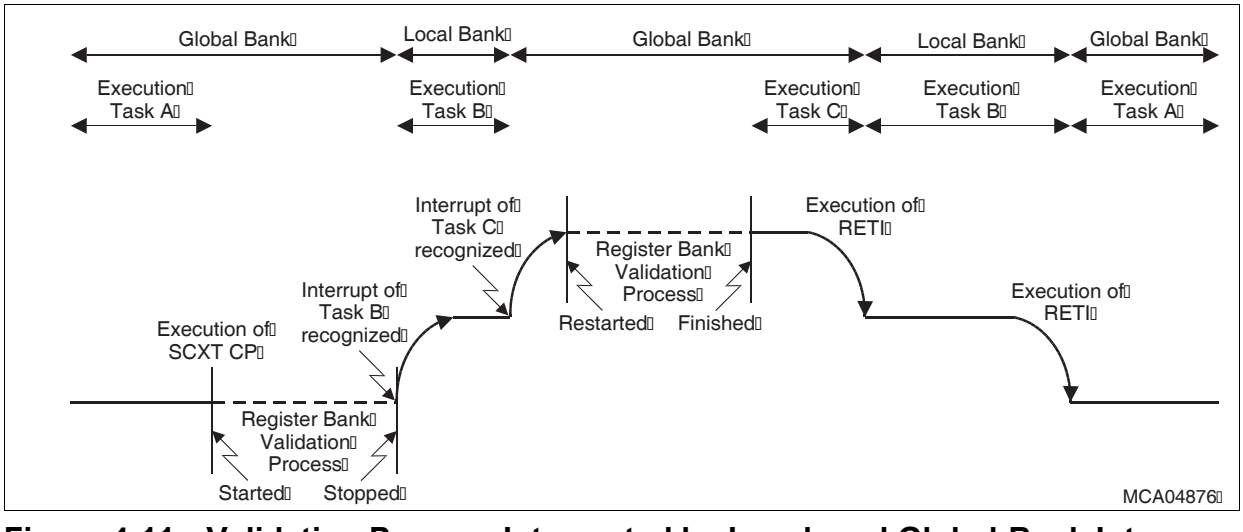


Figure 4-11 Validation Process Interrupted by Local- and Global-Bank Intr.

4.5.2.1 The Context Pointer (CP)

This non-bit-addressable register selects the current global register bank context. It can be updated via any instruction capable of modifying SFRs.

CP
Context Pointer **SFR ($FE10_H/08_H$)** **Reset Value: $FC00_H$**

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

Field	Bits	Type	Description
cp	[11:1]	rw	**Modifiable Portion of Register CP** Specifies the (word) base address of the current global (memory-mapped) register bank. When writing a value to register CP with bits CP[11:9] = 000_B, bits CP[11:10] are set to 11_B by hardware.

Note: It is the user's responsibility to ensure that the physical GPR address specified via CP register plus short GPR address is always an internal DPRAM location. If this condition is not met, unexpected results may occur. Do not set CP below the internal DPRAM start address. Do not set CP above $FDE0_H$, otherwise the store phase will overwrite SFRs (beginning at $FE00_H$).

The XE16x switches the complete memory-mapped GPR bank with a single instruction. After switching, the service routine executes within its own separate context.

The instruction "SCXT CP, #New_Bank" pushes the value of the current context pointer (CP) into the system stack and loads CP with the immediate value "New_Bank", which selects a new register bank. The service routine may now use its "own registers". This memory register bank is preserved when the service routine terminates, i.e. its contents are available on the next call.
Before returning from the service routine (RETI), the previous CP is simply popped from the system stack which returns the registers to the original bank.

Note: Due to the internal instruction pipeline, a write operation to the CP register stalls the instruction flow until the register file context switch is really executed. The instruction immediately following the instruction that updates CP register can use the new value of the changed CP.

4.6 Code Addressing

The XE16x provides a total addressable memory space of 16 Mbytes. This address space is arranged as 256 segments of 64 Kbytes each. A dedicated 24-bit code address pointer is used to access the memories for instruction fetches. This pointer has two parts: an 8-bit code segment pointer CSP and a 16-bit offset pointer called Instruction Pointer (IP). The concatenation of the CSP and IP results directly in a correct 24-bit physical memory address.

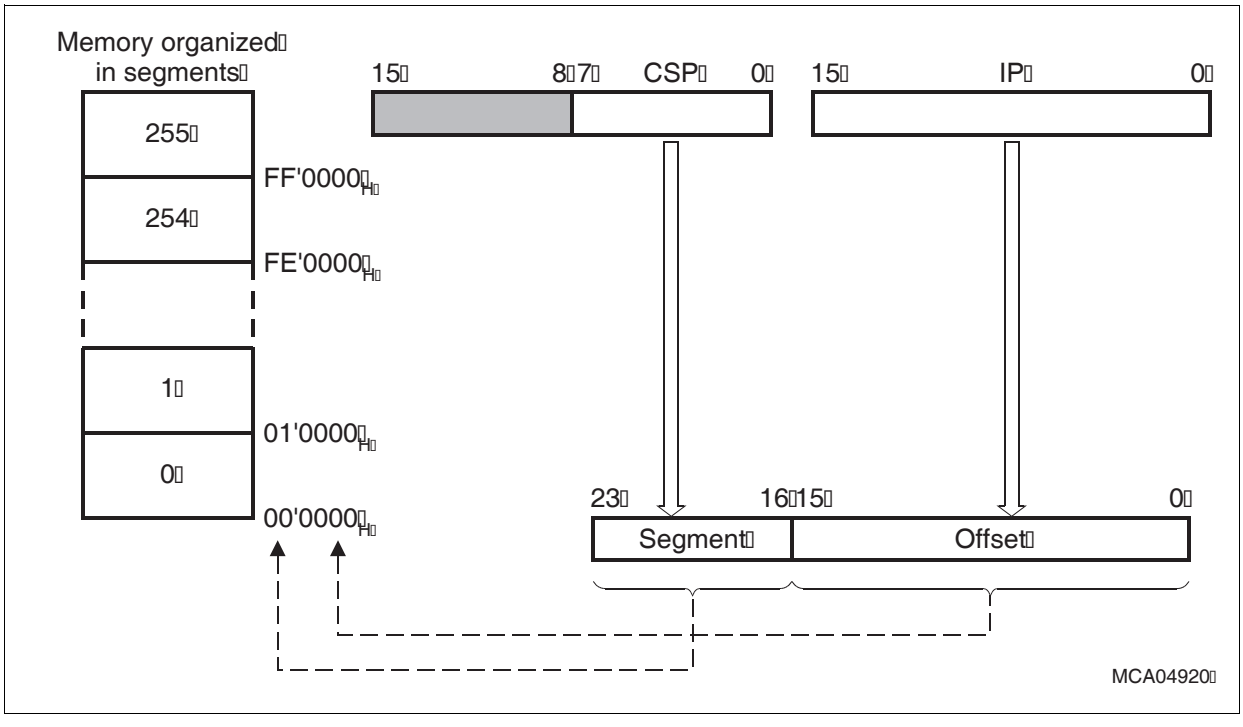


Figure 4-12 Addressing via the Code Segment and Instruction Pointer

tbd RAS

The Code Segment Pointer CSP selects the code segment being used at run-time to access instructions. The lower 8 bits of register CSP select one of up 256 segments of 64 Kbytes each, while the higher 8 bits are reserved for future use. The reset value is specified by the contents of the VECSEG register (**Section 5.3**).

Note: Register CSP can only be read but cannot be written by data operations.

In segmented memory mode (default after reset), register CSP is modified either directly by JMPS and CALLS instructions, or indirectly via the stack by RETS and RETI instructions.

In non-segmented memory mode (selected by setting bit SGTDIS in register CPUCON1), CSP is fixed to the segment of the instruction that disabled segmentation. Modification by inter-segment CALLs or RETurns is no longer possible.

For processing an accepted interrupt or a TRAP, register CSP is automatically loaded with the segment of the vector table (defined in register VECSEG).

Note: For the correct execution of interrupt tasks in non-segmented memory mode, the contents of VECSEG must select the same segment as the current value of CSP, i.e. the vector table must be located in the segment pointed to by the CSP.

CSP
Code Segment Pointer **SFR ($FE08_H/04_H$)** **Reset Value: $xxxx_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	SEGNR							
-	-	-	-	-	-	-	-	rh							

Field	Bits	Type	Description
SEGNR	[7:0]	rh	Specifies the code segment from which the current instruction is to be fetched.

Note: After a reset, register CSP is automatically loaded from register VECSEG.

The Instruction Pointer IP determines the 16-bit intra-segment address of the currently fetched instruction within the code segment selected by the CSP register. Register IP is not mapped into the XE16x's address space; thus, it is not directly accessible by the programmer. However, the IP can be modified indirectly via the stack by means of a return instruction. IP is implicitly updated by the CPU for branch instructions and after instruction fetch operations.

IP
Instruction Pointer **- - - (- - - -/- -)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ip															0
(r)(w)h															r

Field	Bits	Type	Description
ip	[15:1]	h	Specifies the intra segment offset from which the current instruction is to be fetched. IP refers to the current segment <SEGNR>.

4.7 Data Addressing

The Address Data Unit (ADU) contains two independent arithmetic units to generate, calculate, and update addresses for data accesses, the Standard Address Generation Unit (SAGU) and the DSP Address Generation Unit (DAGU). The ADU performs the following major tasks:

- Standard Address Generation (SAGU)
- DSP Address Generation (DAGU)
- Data Paging (SAGU)
- Stack Handling (SAGU)

The SAGU supports linear arithmetic for the indirect addressing modes and also generates the address in case of all other short and long addressing modes.

The DAGU contains an additional set of address pointers and offset registers which are used in conjunction with the CoXXX instructions only.

The CPU provides a lot of powerful addressing modes (short, long, indirect) for word, byte, and bit data accesses. The different addressing modes use different formats and have different scopes.

4.7.1 Short Addressing Modes

Short addressing modes allow access to the GPR, SFR or bit-addressable memory space. All of these addressing modes use an offset (8/4/2 bits) together with an implicit base address to specify a 24-bit physical address:

Table 4-17 Short Addressing Modes

Mnemo-nic	Base Address[1)]	Offset	Short Address Range	Scope of Access
Rw	(CP)	2 × Rw	0 … 15	GPRs (word)
Rb	(CP)	1 × Rb	0 … 15	GPRs (byte)
reg	$00'FE00_H$ $00'F000_H$ (CP) (CP)	2 × reg 2 × reg 2 × (reg ∧ $0F_H$) 1 × (reg ∧ $0F_H$)	00_H … EF_H 00_H … EF_H $F0_H$ … FF_H $F0_H$ … FF_H	SFRs (word, low byte) ESFRs (word, low byte) GPRs (word) GPRs (bytes)
bitoff	$00'FD00_H$ $00'FF00_H$ $00'F100_H$ (CP)	2 × bitoff 2 × (bitoff ∧ $7F_H$) 2 × (bitoff ∧ $7F_H$) 2 × (bitoff ∧ $0F_H$)	00_H … $7F_H$ 80_H … EF_H 80_H … EF_H $F0_H$ … FF_H	RAM Bit word offset SFR Bit word offset ESFR Bit word offset GPR Bit word offset
bitaddr	Bit word see bitoff	Immediate bit position	0 … 15	Any single bit

1) Accesses to general purpose registers (GPRs) may also access local register banks, instead of using CP.

Physical Address = Base Address + $\Delta \times$ Short Address

Note: Δ is 1 for byte GPRs, Δ is 2 for word GPRs.

Rw, Rb: Specifies direct access to any GPR in the currently active context (global register bank or local register bank). Both ‘Rw’ and ‘Rb’ require four bits in the instruction format. The base address of the global register bank is determined by the contents of register CP. ‘Rw’ specifies a 4-bit word GPR address, ‘Rb’ specifies a 4-bit byte GPR address within a local register bank or relative to (CP).

reg: Specifies direct access to any (E)SFR or GPR in the currently active context (global or local register bank). The ‘reg’ value requires eight bits in the instruction format. Short ‘reg’ addresses in the range from 00_H to EF_H always specify (E)SFRs. In that case, the factor ‘Δ’ equates 2 and the base address is $00'FE00_H$ for the standard SFR area or $00'F000_H$ for the extended ESFR area. The ‘reg’ accesses to the ESFR area require a preceding EXT*R instruction to switch the base address. Depending on the opcode, either the total word (for word operations) or the low byte (for byte operations) of an SFR can be addressed via ‘reg’. Note that the high byte of an SFR cannot be accessed via the ‘reg’ addressing mode. Short ‘reg’ addresses in the range from $F0_H$ to FF_H always specify GPRs. In that case, only the lower four bits of ‘reg’ are significant for physical address generation and, therefore, it is identical to the address generation described for the ‘Rb’ and ‘Rw’ addressing modes.

bitoff: Specifies direct access to any word in the bit addressable memory space. The ‘bitoff’ value requires eight bits in the instruction format. The specified ‘bitoff’ range selects different base addresses to generate physical addresses (see **Table 4-17**). The ‘bitoff’ accesses to the ESFR area require a preceding EXT*R instruction to switch the base address.

bitaddr: Any bit address is specified by a word address within the bit addressable memory space (see ‘bitoff’) and a bit position (‘bitpos’) within that word. Therefore, ‘bitaddr’ requires twelve bits in the instruction format.

4.7.2 Long Addressing Modes

Long addressing modes specify 24-bit addresses and, therefore, can access any word or byte data within the entire address space. Long addresses can be specified in different ways to generate the full 24-bit address:

- **Use one of the four Data Page Pointers (DPP registers):** The used 16-bit pointer selects a DPP with bits 15 … 14, bits 13 … 0 specify the 14-bit data page offset (see **Figure 4-13**).
- **Select the used data page directly:** The data page is selected by a preceeding EXTP(R) instruction, bits 13 … 0 of the used 16-bit pointer specify the 14-bit data page offset.
- **Select the used segment directly:** The segment is selected by a preceeding EXTS(R) instruction, the used 16-bit pointer specifies the 16-bit segment offset.

Note: Word accesses on odd byte addresses are not executed. A hardware trap will be triggered.

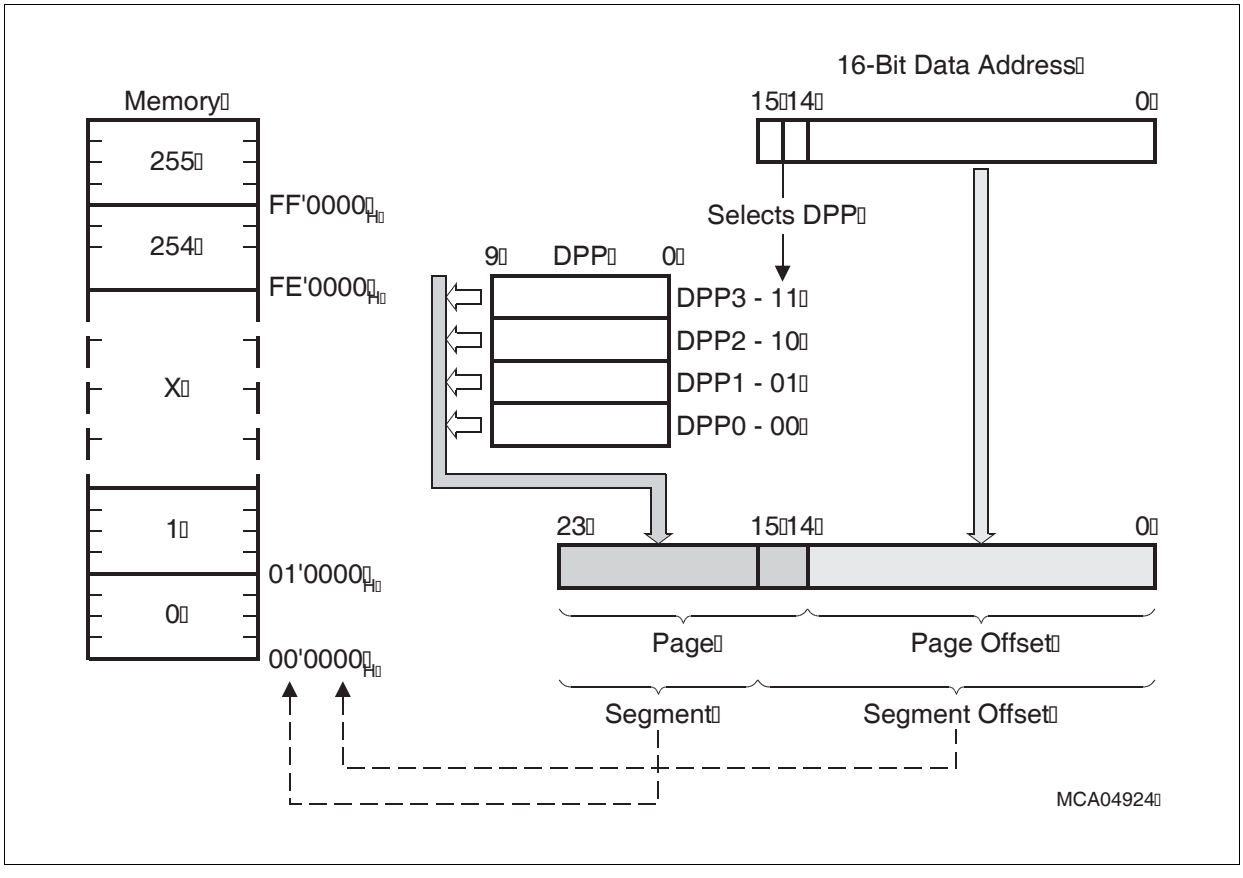


Figure 4-13 Data Page Pointer Addressing

4.7.2.1 Data Page Pointers DPP0, DPP1, DPP2, DPP3

These four non-bit-addressable registers select up to four different data pages to be active simultaneously at run-time. The lower 10 bits of each DPP register select one of the 1024 possible 16-Kbyte data pages; the upper 6 bits are reserved for future use.

DPP0
Data Page Pointer 0 **SFR ($FE00_H/00_H$)** **Reset Value: 0000_H**
DPP1
Data Page Pointer 1 **SFR ($FE02_H/01_H$)** **Reset Value: 0001_H**
DPP2
Data Page Pointer 2 **SFR ($FE04_H/02_H$)** **Reset Value: 0002_H**
DPP3
Data Page Pointer 3 **SFR ($FE06_H/03_H$)** **Reset Value: 0003_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	DPPxPN									
-	-	-	-	-	-	rw									

Field	Bits	Type	Description
DPPxPN	[9:0]	rw	**Data Page Number of DPPx** Specifies the data page selected via DPPx.

The DPP registers allow access to the entire memory space in pages of 16 Kbytes each. The DPP registers are implicitly used whenever data accesses to any memory location are made via indirect or direct long 16-bit addressing modes (except for override accesses via EXTended instructions and PEC data transfers). After reset, the Data Page Pointers are initialized in such a way that all indirect or direct long 16-bit addresses result in identical 18-bit addresses. This allows access to data pages 3 … 0 within segment 0 as shown in **Figure 4-13**. If the user does not want to use data paging, no further action is required.

Data paging is performed by concatenating the lower 14 bits of an indirect or direct long 16-bit address with the contents of the DPP register selected by the upper two bits of the 16-bit address. The contents of the selected DPP register specify one of the 1024 possible data pages. This data page base address together with the 14-bit page offset forms the physical 24-bit address (even if segmentation is disabled).

The selected number of segment address bits (via bitfield SALSEL) of the respective DPP register is output on the respective segment address pins for all external data accesses.

A DPP register can be updated via any instruction capable of modifying an SFR.

Note: Due to the internal instruction pipeline, a write operation to the DPPx registers could stall the instruction flow until the DPP is actually updated. The instruction that immediately follows the instruction which updates the DPP register can use the new value of the changed DPPx.

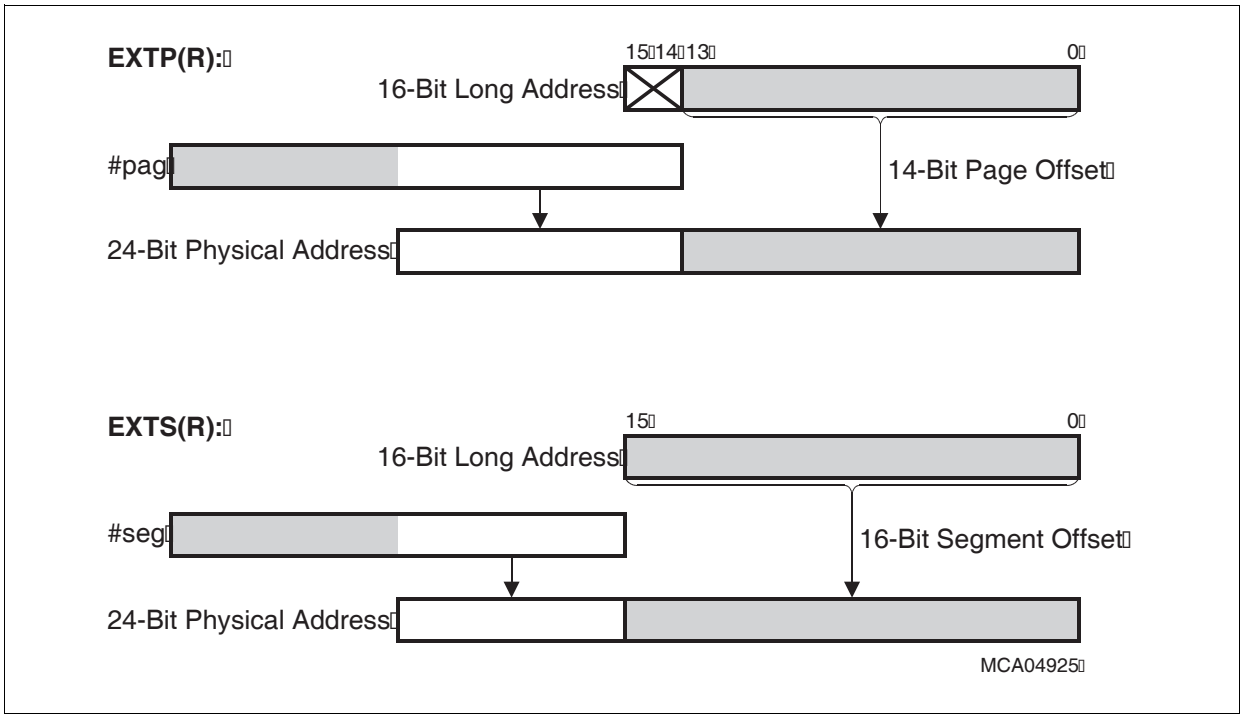


Figure 4-14 Overriding the DPP Mechanism

Note: The overriding page or segment may be specified as a constant (#pag, #seg) or via a word GPR (Rw).

Table 4-18 Long Addressing Modes

Mnemonic	Base Address[1)]	Offset	Scope of Access
mem	(DPPx)	mem $\wedge$ $3FFF_H$	Any Word or Byte
mem	pag	mem $\wedge$ $3FFF_H$	Any Word or Byte
mem	seg	mem	Any Word or Byte

1) Represents either a 10-bit data page number to be concatenated with a 14-bit offset, or an 8-bit segment number to be concatenated with a 16-bit offset.

4.7.3 Indirect Addressing Modes

Indirect addressing modes can be considered as a combination of short and long addressing modes. This means that the “long” 16-bit pointer is provided indirectly by the contents of a word GPR which itself is specified directly by a short 4-bit address (‘Rw’ = 0 … 15).

There are indirect addressing modes, which add a constant value to the GPR contents before the long 16-bit address is calculated. Other indirect addressing modes can decrement or increment the indirect address pointers (GPR contents) by 2 or 1 (referring to words or bytes) or by the contents of the offset registers QR0 or QR1.

Table 4-19 Generating Physical Addresses from Indirect Pointers

Step	Executed Action	Calculation	Notes
1	Calculate the address of the indirect pointer (word GPR) from its short address	**GPR Address = 2 × Short Addr. [+ (CP)]**	see **Table 4-17**
2	Pre-decrement indirect pointer (‘-Rw’) depending on datatype (Δ = 1 or 2 for byte or word operations)	**(GPR Address) = (GPR Address) -** Δ	Optional step, executed only if required by addressing mode
3	Adjust the pointer by a constant value (‘Rw + const16’)	**Pointer = (GPR Address) + Constant**	Optional step, executed only if required by addressing mode
4	Calculate the physical 24-bit address using the resulting pointer	**Physical Addr. = Page/Segment + Pointer offset**	Uses DPPs or page/segment override mechanisms, see **Table 4-18**
5	Post-in/decrement indirect pointer (‘Rw±’) depending on datatype (Δ = 1 or 2 for byte or word operations), or depending on offset registers (Δ = QRx)[1)]	**(GPR Address) = (GPR Address)** ± Δ	Optional step, executed only if required by addressing mode

1) Post-decrement and QRx-based modification is provided only for CoXXX instructions.

Note: Some instructions only use the lowest four word GPRs (R3 … R0) as indirect address pointers, which are specified via short 2-bit addresses in that case.

The following indirect addressing modes are provided:

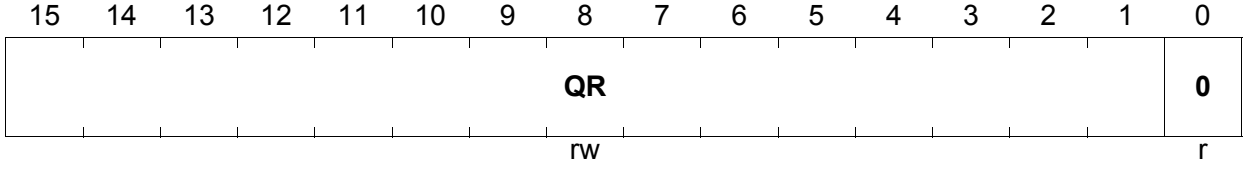
Table 4-20 Indirect Addressing Modes

Mnemonic	Particularities
[Rw]	Most instructions accept any GPR (R15 … R0) as indirect address pointer. Some instructions accept only the lower four GPRs (R3 … R0).
[Rw+]	The specified indirect address pointer is automatically post-incremented by 2 or 1 (for word or byte data operations) after the access.
[-Rw]	The specified indirect address pointer is automatically pre-decremented by 2 or 1 (for word or byte data operations) before the access.
[Rw + #data16]	The specified 16-bit constant is added to the indirect address pointer, before the long address is calculated.
[Rw-]	The specified indirect address pointer is automatically post-decremented by 2 (word data operations) after the access.
[Rw + QRx]	The specified indirect address pointer is automatically post-incremented by QRx (word data operations) after the access.
[Rw - QRx]	The specified indirect address pointer is automatically post-decremented by QRX (word data operations) after the access.

4.7.3.1 Offset Registers QR0 and QR1

The non-bit-addressable offset registers QR0 and QR1 are used with CoXXX instructions. For possible instruction flow stalls refer to **Section 4.3.4**.

QR0
Offset Register **ESFR ($F004_H/02_H$)** **Reset Value: 0000_H**
QR1
Offset Register **ESFR ($F006_H/03_H$)** **Reset Value: 0000_H**



Field	Bits	Type	Description
QR	[15:1]	rw	**Modifiable Portion of Register QRx** Specifies the 16-bit word offset address for indirect addressing modes (LSB always zero).

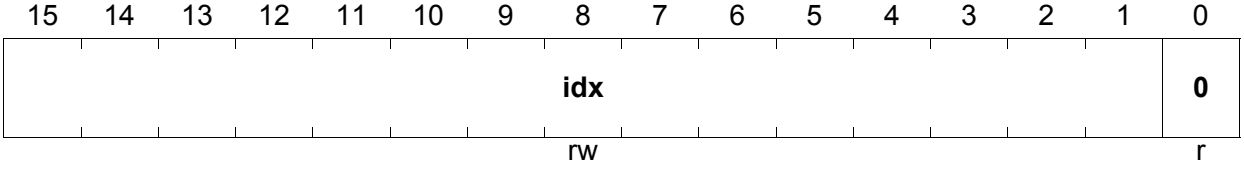
4.7.4 DSP Addressing Modes

In addition to the Standard Address Generation Unit (SAGU), the DSP Address Generation Unit (DAGU) provides an additional set of pointer registers (IDX0, IDX1) and offset registers (QX0, QX1). The additional set of pointer registers IDX0 and IDX1 allows the execution of DSP specific CoXXX instructions in one CPU cycle. An independent arithmetic unit allows the update of these dedicated pointer registers in parallel with the GPR-pointer modification of the SAGU. The DAGU only supports indirect addressing modes that use the special pointer registers IDX0 and IDX1.

The address pointers can be used for arithmetic operations as well as for the special CoMOV instruction. The generation of the 24-bit memory address is different:

- For **CoMOV** instructions, the IDX pointers are concatenated with the DPPs or the selected page/segment address, as described for long addressing modes (see **Figure 4-13** for a summary).
- For **arithmetic CoXXX** instructions, the IDX pointers are automatically extended to a 24-bit memory address pointing to the internal DPRAM area, as shown in **Figure 4-15**.

IDX0
Address Pointer **SFR ($FF08_H/84_H$)** **Reset Value: 0000_H**
IDX1
Address Pointer **SFR ($FF0A_H/85_H$)** **Reset Value: 0000_H**



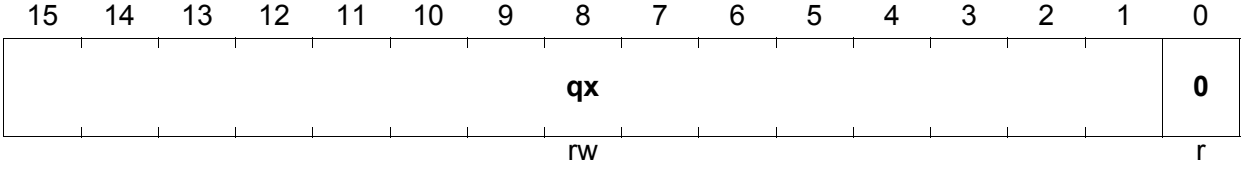
Field	Bits	Type	Description
idx	[15:1]	rw	**Modifiable Portion of Register IDXx** Specifies the 16-bit word address pointer

*Note: During the initialization of the IDX registers, instruction flow stalls are possible. For the proper operation, refer to **Section 4.3.4**.*

There are indirect addressing modes which allow parallel data move operations before the long 16-bit address is calculated (see **Figure 4-16** for an example). Other indirect addressing modes allow decrementing or incrementing the indirect address pointers (IDXx contents) by 2 or by the contents of the offset registers QX0 and QX1 (used in conjunction with the IDX pointers).

QX0
Offset Register **ESFR ($F000_H/00_H$)** **Reset Value: 0000_H**

QX1
Offset Register **ESFR ($F002_H/01_H$)** **Reset Value: 0000_H**



Field	Bits	Type	Description
qx	[15:1]	rw	**Modifiable Portion of Register QXx** Specifies the 16-bit word offset for indirect addressing modes

*Note: During the initialization of the QX registers, instruction flow stalls are possible. For the proper operation, refer to **Section 4.3.4**.*

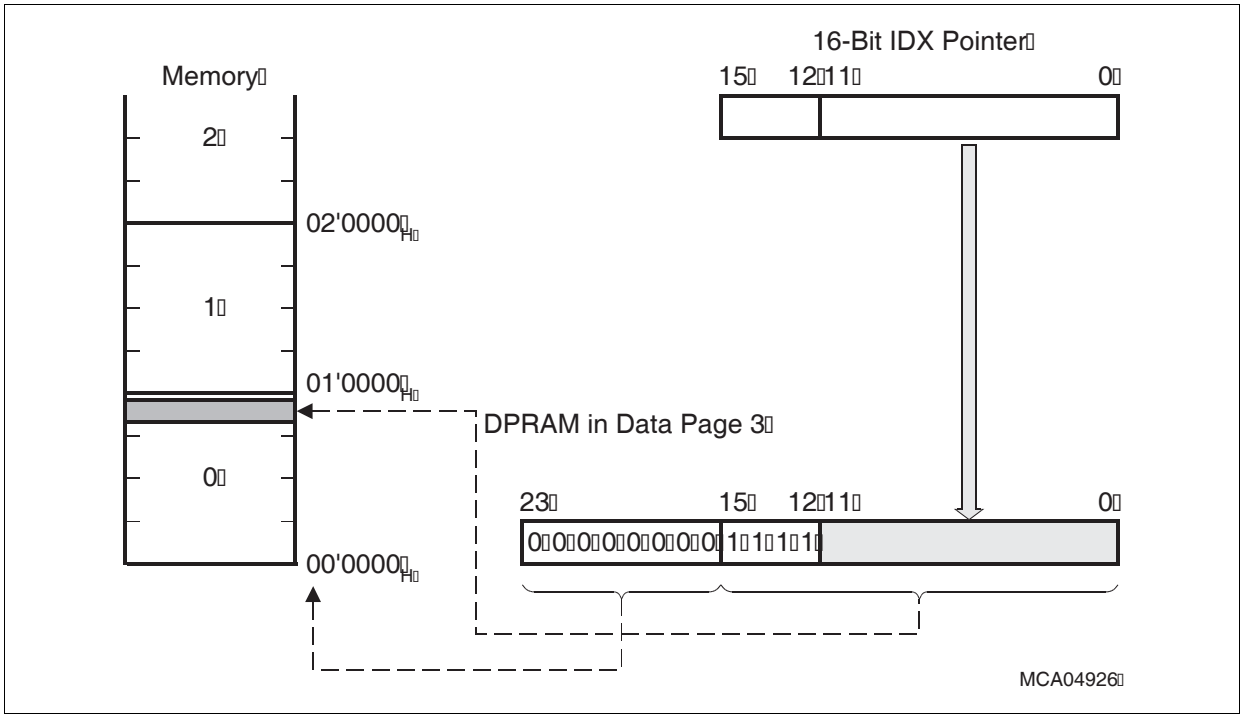


Figure 4-15 Arithmetic MAC Operations and Addressing via the IDX Pointers

Table 4-21 Generating Physical Addresses from Indirect Pointers (IDXx)

Step	Executed Action	Calculation	Notes
1	Determine the used IDXx pointer	**---**	–
2	Calculate an intermediate long address for the parallel data move operation and in/decrement indirect pointer (‘IDXx±’) by 2 (Δ = 2), or depending on offset registers (Δ = QXx)	**Interm. Addr. = (IDXx Address)** ± Δ	Optional step, executed only if required by instruction CoXXXM and addressing mode
3	Calculate long 16-bit address	**Long Address = (IDXx Pointer)**	–
4	Calculate the physical 24-bit address using the resulting pointer	**Physical Addr. = Page/Segment + Pointer offset**	Uses DPPs or page/segment override mechanisms, see **Table 4-18** and **Figure 4-15**
5	Post-in/decrement indirect pointer (‘IDXx±’) by 2 (Δ = 2), or depending on offset registers (Δ = QXx)	**(IDXx Pointer) = (IDXx Pointer)** ± Δ	Optional step, executed only if required by addressing mode

The following indirect addressing modes are provided:

Table 4-22 DSP Addressing Modes

Mnemonic	Particularities
[IDXx]	Most CoXXX instructions accept IDXx (IDX0, IDX1) as an indirect address pointer.
[IDXx+]	The specified indirect address pointer is automatically post-incremented by 2 after the access.
with parallel data move	In case of a CoXXXM instruction, the address stored in the specified indirect address pointer is automatically pre-decremented by 2 for the parallel move operation. The pointer itself is not pre-decremented. Then, the specified indirect address pointer is automatically post-incremented by 2 after the access.
[IDXx-]	The specified indirect address pointer is automatically post-decremented by 2 after the access.
with parallel data move	In case of a CoXXXM instruction, the address stored in the specified indirect address pointer is automatically pre-incremented by 2 for the parallel move operation. The pointer itself is not pre-incremented. Then, the specified indirect address pointer is automatically post-decremented by 2 after the access.
[IDXx + QXx]	The specified indirect address pointer is automatically post-incremented by QXx after the access.
with parallel data move	In case of a CoXXXM instruction, the address stored in the specified indirect address pointer is automatically pre-decremented by QXx for the parallel move operation. The pointer itself is not pre-decremented. Then, the specified indirect address pointer is automatically post-incremented by QXx after the access.
[IDXx - QXx]	The specified indirect address pointer is automatically post-decremented by QXx after the access.
with parallel data move	In case of a CoXXXM instruction, the address stored in the specified indirect address pointer is automatically pre-incremented by QXx for the parallel move operation. The pointer itself is not pre-incremented. Then, the specified indirect address pointer is automatically post-decremented by QXx after the access.

*Note: An example for parallel data move operations can be found in **Figure 4-16**.*

The CoREG Addressing Mode

The CoSTORE instruction utilizes the special CoREG addressing mode for immediate storage of the MAC-Unit register after a MAC operation. The address of the MAC-Unit register is coded in the CoSTORE instruction format as described in **Table 4-23**:

Table 4-23 Coding of the CoREG Addressing Mode

Mnemonic	Register	Coding of wwww:w bits [31:27]
MSW	MAC-Unit Status Word	00000
MAH	MAC-Unit Accumulator High Word	00001
MAS	Limited MAC-Unit Accumulator High Word	00010
MAL	MAC-Unit Accumulator Low Word	00100
MCW	MAC-Unit Control Word	00101
MRW	MAC-Unit Repeat Word	00110

The example in **Figure 4-16** shows the complex operation of CoXXXM instructions with a parallel move operation based on the descriptions about addressing modes given in **Section 4.7.3** (**Indirect Addressing Modes**) and **Section 4.7.4** (**DSP Addressing Modes**).

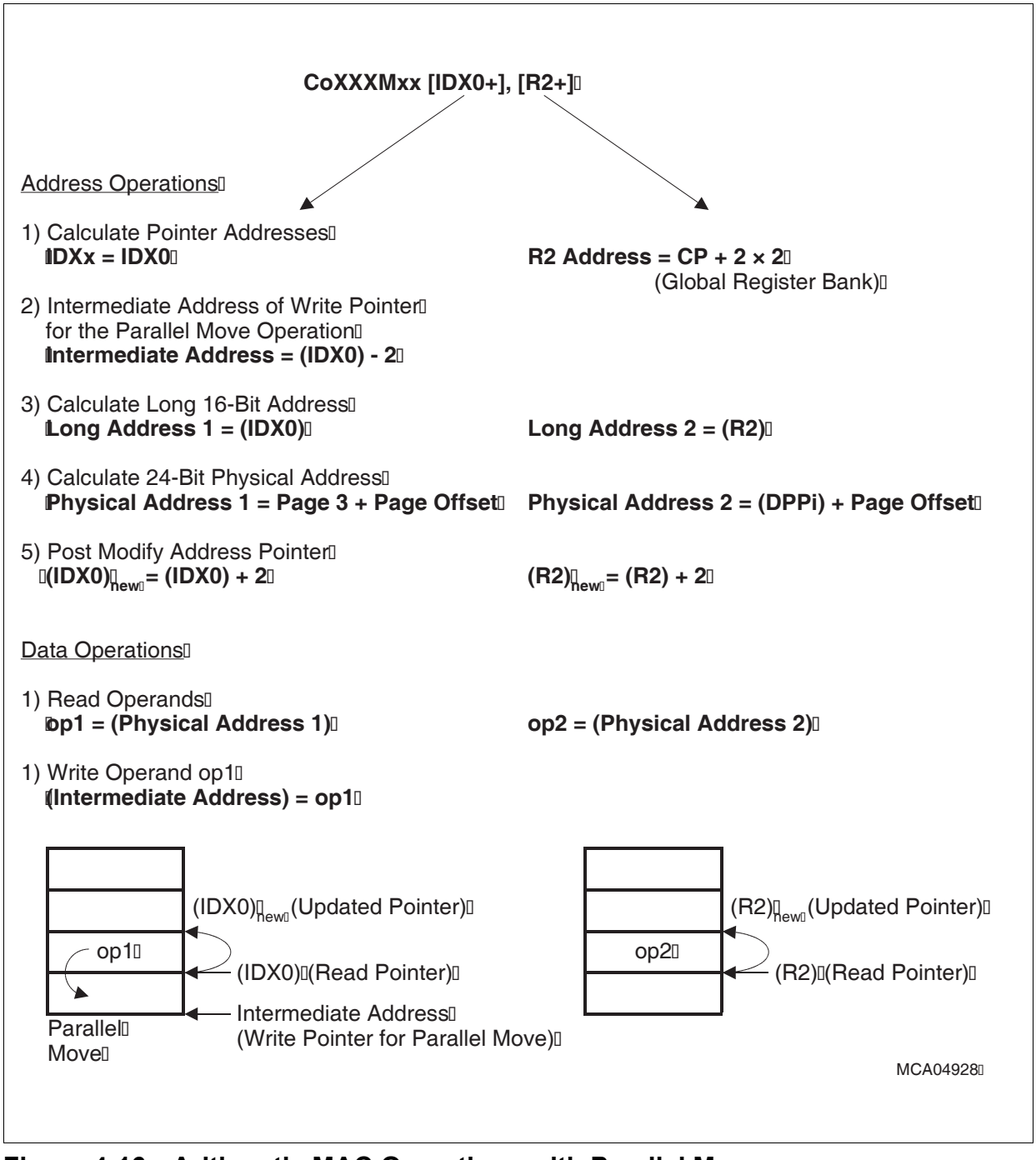


Figure 4-16 Arithmetic MAC Operations with Parallel Move

4.7.5 The System Stack

The XE16x supports a system stack of up to 64 Kbytes. The stack can be located internally in one of the on-chip memories or externally. The 16-bit Stack Pointer register (SP) addresses the stack within a 64-Kbyte segment selected by the Stack Pointer Segment register (SPSG). A virtual stack (usually bigger than 64 Kbytes) can be implemented by software. This mechanism is supported by the Stack Overflow register STKOV and the Stack Underflow register STKUN (see descriptions below).

4.7.5.1 The Stack Pointer Registers SP and SPSEG

Register SPSEG (not bitaddressable) selects the segment being used at run-time to access the system stack. The lower eight bits of register SPSEG select one of up 256 segments of 64 Kbytes each, while the higher 8 bits are reserved for future use.

The Stack Pointer SP (not bitaddressable) points to the top of the system stack (TOS). SP is pre-decremented whenever data is pushed onto the stack, and it is post-incremented whenever data is popped from the stack. Therefore, the system stack grows from higher towards lower memory locations.

System stack addresses are generated by directly extending the 16-bit contents of register SP by the contents of register SPSG, as shown in **Figure 4-17**.

The system stack cannot cross a 64-Kbyte segment boundary.

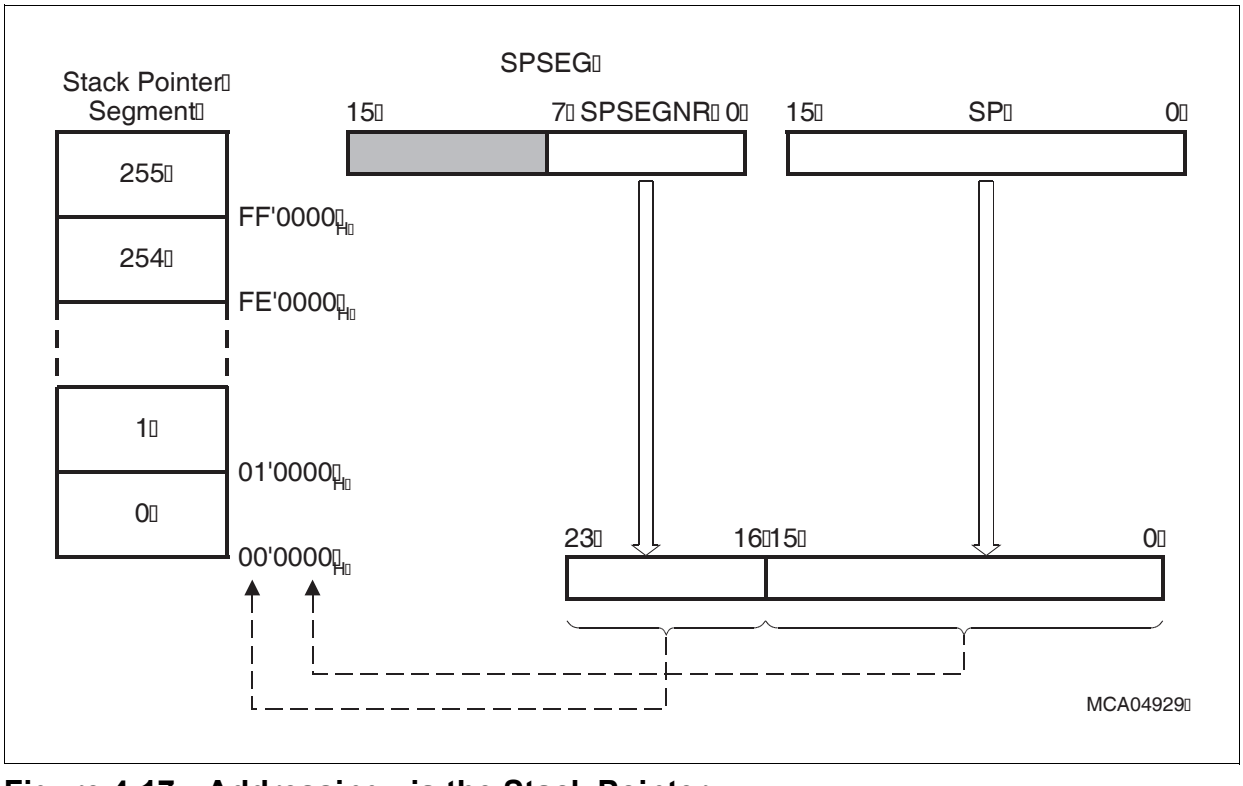
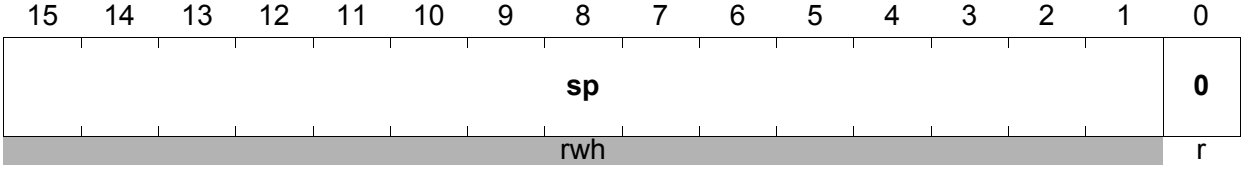


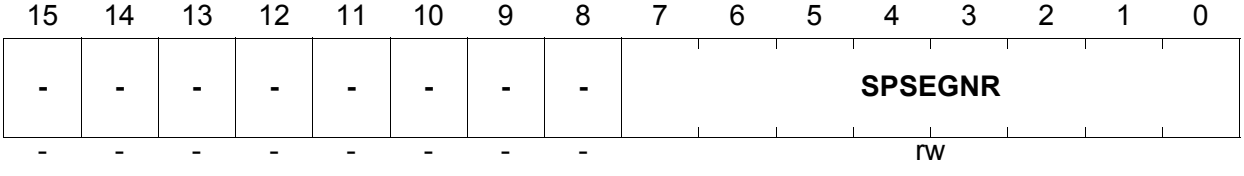
Figure 4-17 Addressing via the Stack Pointer

SP
Stack Pointer Register **SFR ($FE12_H/09_H$)** **Reset Value: $FC00_H$**



Field	Bits	Type	Description
sp	[15:1]	rwh	**Modifiable Portion of Register SP** Specifies the top of the system stack.

SPSEG
Stack Pointer Segment **SFR ($FF0C_H/86_H$)** **Reset Value: 0000_H**



Field	Bits	Type	Description
SPSEGNR	[7:0]	rw	**Stack Pointer Segment Number** Specifies the segment where the stack is located.

Note: SPSEG and SP can be updated via any instruction capable of modifying a 16-bit SFR. Due to the internal instruction pipeline, a write operation to SPSG or SP stalls the instruction flow until the register is really updated. The instruction immediately following the instruction updating SPSG or SP can use the new value. Extreme care should be taken when changing the contents of the stack pointer registers. Improper changes may result in erroneous system behavior.

4.7.5.2 The Stack Overflow/Underflow Pointers STKOV/STKUN

These limit registers (not bit-addressable) supervise the stack pointer. A trap is generated when the stack pointer reaches its upper or lower limit. The Stack Pointer Segment Register SPSG is not taken into account for the stack pointer comparison. The system stack cannot cross a 64-Kbyte segment.

STKOV is compared with SP before each implicit write operation which decrements the contents of SP (instructions CALLA, CALLI, CALLR, CALLS, PCALL, TRAP, SCXT, or PUSH). If the contents of SP are equal to the contents of STKOV a stack overflow trap is triggered.

STKUN is compared with SP before each implicit read operation which increments the contents of SP (instructions RET, RETS, RETP, RETI, or POP). If the contents of SP are equal to the contents of STKUN a stack underflow trap is triggered.

The Stack Overflow/Underflow Traps may be used in two different ways:

- **Fatal error indication** treats the stack overflow as a system error and executes the associated trap service routine.
  In case of a stack overflow trap, data in the bottom of the stack may have been overwritten by the status information stacked upon servicing the trap itself.
- **Virtual stack control** allows the system stack to be used as a ‘Stack Cache’ for a bigger external user stack: flush cache in case of an overflow, refill cache in case of an underflow.

Scope of Stack Limit Control

The stack limit control implemented by the register pair STKOV and STKUN detects cases in which the Stack Pointer (SP) crosses the defined stack area as a result of an implicit change.

If the stack pointer was explicitly changed as a result of move or arithmetic instruction, SP is not compared to the contents of STKOV and STKUN. In this case, a stack violation will not be detected if the modified stack pointer is on or outside the defined limits, i.e. below (STKOV) or above (STKUN). Stack overflow/underflow is detected only in case of implicit SP modification.

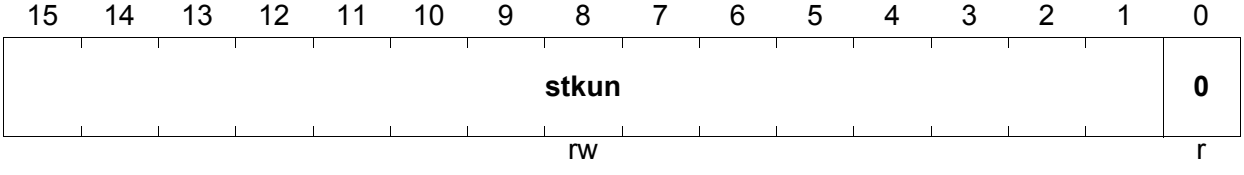
SP may be operated outside the permitted SP range without triggering a trap. However, if SP reaches the limit of the permitted SP range from outside the range as a result of an implicit change (PUSH or POP, for example), the respective trap will be triggered.

Note: STKOV and STKUN can be updated via any instruction capable of modifying an SFR. If a stack overflow or underflow event occurs in an ATOMIC/EXT sequence, the stack operations that are part of the sequence are completed. The trap is issued after the completion of the entire ATOMIC/EXT sequence.

STKOV
Stack Overflow Reg. **SFR ($FE14_H/0A_H$)** **Reset Value: $FA00_H$**

Field	Bits	Type	Description
stkov	[15:1]	rw	**Modifiable Portion of Register STKOV** Specifies the segment offset address of the lower limit of the system stack.

STKUN
Stack Underflow Reg. **SFR ($FE16_H/0B_H$)** **Reset Value: $FC00_H$**



Field	Bits	Type	Description
stkun	[15:1]	rw	**Modifiable Portion of Register STKUN** Specifies the segment offset address of the upper limit of the system stack.

4.8 Standard Data Processing

All standard arithmetic, shift-, and logical operations are performed in the 16-bit ALU. In addition to the standard functions, the ALU of the XE16x includes a bit-manipulation unit and a multiply and divide unit. Most internal execution blocks have been optimized to perform operations on either 8-bit or 16-bit numbers. After the pipeline has been filled, most instructions are completed in one CPU cycle. The status flags are automatically updated in register PSW after each ALU operation and reflect the current state of the microcontroller. These flags allow branching upon specific conditions. Support of both signed and unsigned arithmetic is provided by the user selectable branch test. The status flags are also preserved automatically by the CPU upon entry into an interrupt or trap routine. Another group of bits represents the current CPU interrupt status. Two separate bits (USR0 and USR1) are provided as general purpose flags.

PSW
Processor Status Word **SFR** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ILVL				IEN	HLD EN	BANK		USR 1	USR 0	MUL IP	E	Z	V	C	N
rwh				rw	rw	rwh		rwh	rwh	r	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
ILVL	[15:12]	rwh	**CPU Priority Level** 0H Lowest Priority … … FH Highest Priority
IEN	11	rw	**Global Interrupt/PEC Enable Bit** 0 Interrupt/PEC requests are disabled 1 Interrupt/PEC requests are enabled
HLDEN	10	rw	**Hold Enable** 0 External bus arbitration disabled 1 External bus arbitration enabled *Note: The selected arbitration mode is activated when HLDEN is set for the first time.*
BANK	[9:8]	rwh	**Reserved for Register File Bank Selection** 00 Global register bank 01 Reserved 10 Local register bank 1 11 Local register bank 2

Field	Bits	Type	Description
USR1	7	rwh	**General Purpose Flag** May be used by application
USR0	6	rwh	**General Purpose Flag** May be used by application
MULIP	5	r	**Multiplication/Division in Progress** *Note: Always set to 0 (MUL/DIV not interruptible), for compatibility with existing software.*
E	4	rwh	**End of Table Flag** 0 Source operand is neither 8000H nor 80H 1 Source operand is 8000H or 80H
Z	3	rwh	**Zero Flag** 0 ALU result is not zero 1 ALU result is zero
V	2	rwh	**Overflow Flag** 0 No Overflow produced 1 Overflow produced
C	1	rwh	**Carry Flag** 0 No carry/borrow bit produced 1 Carry/borrow bit produced
N	0	rwh	**Negative Result** 0 ALU result is not negative 1 ALU result is negative

ALU/MAC Status (N, C, V, Z, E, USR0, USR1)

The condition flags (N, C, V, Z, E) within the PSW indicate the ALU status after the most recently performed ALU operation. They are set by most of the instructions according to specific rules which depend on the ALU or data movement operation performed by an instruction.

After execution of an instruction which explicitly updates the PSW register, the condition flags cannot be interpreted as described below because any explicit write to the PSW register supersedes the condition flag values which are implicitly generated by the CPU. Explicitly reading the PSW register supplies a read value which represents the state of the PSW register after execution of the immediately preceding instruction.

Note: After reset, all of the ALU status bits are cleared.

N-Flag: For most of the ALU operations, the N-flag is set to 1, if the most significant bit of the result contains a 1; otherwise, it is cleared. In the case of integer operations, the N-flag can be interpreted as the sign bit of the result (negative: N = 1, positive: N = 0).

Negative numbers are always represented as the 2's complement of the corresponding positive number. The range of signed numbers extends from -8000$_H$ to +7FFF$_H$ for the word data type, or from -80$_H$ to +7F$_H$ for the byte data type. For Boolean bit operations with only one operand, the N-flag represents the previous state of the specified bit. For Boolean bit operations with two operands, the N-flag represents the logical XORing of the two specified bits.

C-Flag: After an addition, the C-flag indicates that a carry from the most significant bit of the specified word or byte data type has been generated. After a subtraction or a comparison, the C-flag indicates a borrow which represents the logical negation of a carry for the addition.
This means that the C-flag is set to 1, if **no** carry from the most significant bit of the specified word or byte data type has been generated during a subtraction, which is performed internally by the ALU as a 2's complement addition, and, the C-flag is cleared when this complement addition caused a carry.
The C-flag is always cleared for logical, multiply and divide ALU operations, because these operations cannot cause a carry.
For shift and rotate operations, the C-flag represents the value of the bit shifted out last. If a shift count of zero is specified, the C-flag will be cleared. The C-flag is also cleared for a prioritize ALU operation, because a 1 is never shifted out of the MSB during the normalization of an operand.
For Boolean bit operations with only one operand, the C-flag is always cleared. For Boolean bit operations with two operands, the C-flag represents the logical ANDing of the two specified bits.

V-Flag: For addition, subtraction, and 2's complementation, the V-flag is always set to 1 if the result exceeds the range of 16-bit signed numbers for word operations (-8000$_H$ to +7FFF$_H$), or 8-bit signed numbers for byte operations (-80$_H$ to +7F$_H$). Otherwise, the V-flag is cleared. Note that the result of an integer addition, integer subtraction, or 2's complement is not valid if the V-flag indicates an arithmetic overflow.
For multiplication and division, the V-flag is set to 1 if the result cannot be represented in a word data type; otherwise, it is cleared. Note that a division by zero will always cause an overflow. In contrast to the result of a division, the result of a multiplication is valid whether or not the V-flag is set to 1.
Because logical ALU operations cannot produce an invalid result, the V-flag is cleared by these operations.

The V-flag is also used as a 'Sticky Bit' for rotate right and shift right operations. With only using the C-flag, a rounding error caused by a shift right operation can be estimated up to a quantity of one half of the LSB of the result. In conjunction with the V-flag, the C-flag allows evaluation of the rounding error with a finer resolution (see **Table 4-24**).

For Boolean bit operations with only one operand, the V-flag is always cleared. For Boolean bit operations with two operands, the V-flag represents the logical ORing of the two specified bits.

Table 4-24 Shift Right Rounding Error Evaluation

C-Flag	V-Flag	Rounding Error Quantity
0	0	No rounding error
0	1	0 < Rounding error < $^1/_2$ LSB
1	0	Rounding error = $^1/_2$ LSB
1	1	Rounding error > $^1/_2$ LSB

Z-Flag: The Z-flag is normally set to 1 if the result of an ALU operation equals zero, otherwise it is cleared.
For the addition and subtraction with carry, the Z-flag is only set to 1, if the Z-flag already contains a 1 and the result of the current ALU operation also equals zero. This mechanism is provided to support multiple precision calculations.
For Boolean bit operations with only one operand, the Z-flag represents the logical negation of the previous state of the specified bit. For Boolean bit operations with two operands, the Z-flag represents the logical NORing of the two specified bits. For the prioritize ALU operation, the Z-flag indicates whether the second operand was zero.

E-Flag: End of table flag. The E-flag can be altered by instructions which perform ALU or data movement operations. The E-flag is cleared by those instructions which cannot be reasonably used for table search operations. In all other cases, the E-flag value depends on the value of the source operand to signify whether the end of a search table is reached or not. If the value of the source operand of an instruction equals the lowest negative number which is representable by the data format of the corresponding instruction (8000_H for the word data type, or 80_H for the byte data type), the E-flag is set to 1; otherwise, it is cleared.

General Control Functions (USR0, USR1, BANK, HLDEN)

A few bits in register PSW are dedicated to general control functions. Thus, they are saved and restored automatically upon task switches and interrupts.

USR0/USR1-Flags: These bits can be set automatically during the execution of repeated MAC instructions. These bits can also be used as general flags by an application.

BANK: Bitfield BANK selects the currently active register bank (local or global). Bitfield BANK is updated implicitly by hardware upon entering an interrupt service routine, and by a RETI instruction. It can be also modified explicitly via software by any instruction which can write to PSW.

HLDEN: Setting this bit for the first time activates the selected bus arbitration mode (see **Section 9.3.8**). Bus arbitration can be disabled by temporarily clearing bit HLDEN. In this case the bus is locked, while the bus arbitration mode remains selected.

CPU Interrupt Status (IEN, ILVL)

IEN: The Interrupt Enable bit allows interrupts to be globally enabled (IEN = 1) or disabled (IEN = 0).

ILVL: The four-bit Interrupt Level field (ILVL) specifies the priority of the current CPU activity. The interrupt level is updated by hardware on entry into an interrupt service routine, but it can also be modified via software to prevent other interrupts from being acknowledged. If an interrupt level 15 has been assigned to the CPU, it has the highest possible priority; thus, the current CPU operation cannot be interrupted except by hardware traps or external non-maskable interrupts. For details refer to **Chapter 5**.

After reset, all interrupts are globally disabled, and the lowest priority (ILVL = 0) is assigned to the initial CPU activity.

4.8.1 16-bit Adder/Subtracter, Barrel Shifter, and 16-bit Logic Unit

All standard arithmetic and logical operations are performed by the 16-bit ALU. In case of byte operations, signals from bits 6 and 7 of the ALU result are used to control the condition flags. Multiple precision arithmetic is supported by a “CARRY-IN” signal to the ALU from previously calculated portions of the desired operation.

A 16-bit barrel shifter provides multiple bit shifts in a single cycle. Rotations and arithmetic shifts are also supported.

4.8.2 Bit Manipulation Unit

The XE16x offers a large number of instructions for bit processing. These instructions either manipulate software flags within the internal RAM, control on-chip peripherals via control bits in their respective SFRs, or control IO functions via port pins.

Unlike other microcontrollers, the XE16x features instructions that provide direct access to two operands in the bit addressable space without requiring them to be moved to temporary locations. Multiple bit shift instructions have been included to avoid long instruction streams of single bit shift operations. These instructions require a single CPU cycle.

The instructions BSET, BCLR, BAND, BOR, BXOR, BMOV, BMOVN explicitly set or clear specific bits. The bitfield instructions BFLDL and BFLDH allow manipulation of up to 8 bits of a specific byte at one time. The instructions JBC and JNBS implicitly clear or set the specified bit when the jump is taken. The instructions JB and JNB (also conditional jump instructions that refer to flags) evaluate the specified bit to determine if the jump is to be taken.

Note: Bit operations on undefined bit locations will always read a bit value of ‘0’, while the write access will not affect the respective bit location.

All instructions that manipulate single bits or bit groups internally use a read-modify-write sequence that accesses the whole word containing the specified bit(s).

This method has several consequences:

- The read-modify-write approach may be critical with hardware-affected bits. In these cases, the hardware may change specific bits while the read-modify-write operation is in progress; thus, the writeback would overwrite the new bit value generated by the hardware. The solution is provided by either the implemented hardware protection (see below) or through special programming (see **Section 4.3**).
- Bits can be modified only within the internal address areas (internal RAM and SFRs). External locations cannot be used with bit instructions.

The upper 256 bytes of SFR area, ESFR area, and internal DPRAM are bit-addressable; so, the register bits located within those respective sections can be manipulated directly using bit instructions. The other SFRs must be accessed byte/word wise.

Note: All GPRs are bit-addressable independently from the allocation of the register bank via the Context Pointer (CP). Even GPRs which are allocated to non-bit-addressable RAM locations provide this feature.

Protected bits are not changed during the read-modify-write sequence, such as when hardware sets an interrupt request flag between the read and the write of the read-modify-write sequence. The hardware protection logic guarantees that only the intended bit(s) is/are affected by the write-back operation.

Note: If a conflict occurs between a bit manipulation generated by hardware and an intended software access, the software access has priority and determines the final value of the respective bit.

4.8.3 Multiply and Divide Unit

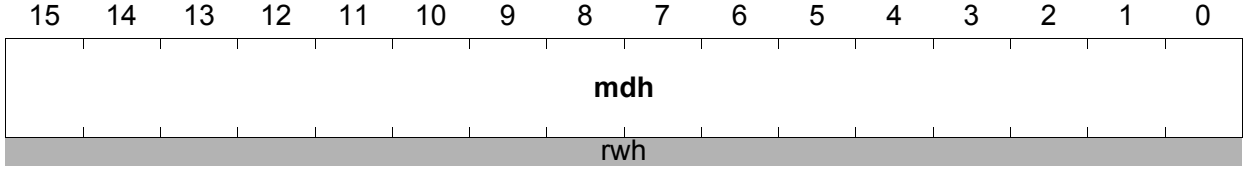
The XE16x's multiply and divide unit has two separated parts. One is the fast 16 × 16-bit multiplier that executes a multiplication in one CPU cycle. The other one is a division sub-unit which performs the division algorithm in 18 … 21 CPU cycles (depending on the data and division types). The divide instruction requires four CPU cycles to be executed. For performance reasons, the rest of the division algorithm runs in the background during the following seventeen CPU cycles, while further instructions are executed in parallel. Interrupt tasks can also be started and executed immediately without any delay. If an instruction (from the original instruction stream or from the interrupt task) tries to use the unit while a division is still running, the execution of this new instruction is stalled until the previous division is finished.

To avoid these stalls, the multiply and division unit should not be used during the first fourteen CPU cycles of the interrupt tasks. For example, this requires up to fourteen one-cycle instructions to be executed between the interrupt entry and the first instruction which uses the multiply and divide unit again (worst case).

Multiplications and divisions implicitly use the 32-bit multiply/divide register MD (represented by the concatenation of the two non-bit-addressable data registers MDH and MDL) and the associated control register MDC. This bit-addressable 16-bit register is implicitly used by the CPU when it performs a division or multiplication in the ALU.

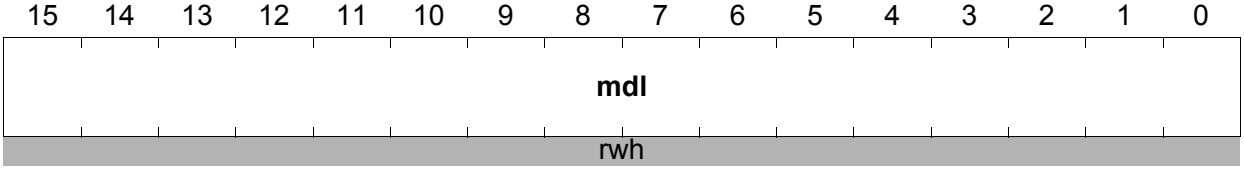
After a multiplication, MD represents the 32-bit result. For long divisions, MD must be loaded with the 32-bit dividend before the division is started. After any division, register MDH represents the 16-bit remainder, register MDL represents the 16-bit quotient.

MDH
Multiply/Divide High Reg. **SFR ($FE0C_H/06_H$)** **Reset Value: 0000_H**



Field	Bits	Type	Description
mdh	[15:0]	rwh	**High Part of MD** The high order sixteen bits of the 32-bit multiply and divide register MD.

MDL
Multiply/Divide Low Reg. **SFR ($FE0E_H/07_H$)** **Reset Value: 0000_H**



Field	Bits	Type	Description
mdl	[15:0]	rwh	**Low Part of MD** The low order sixteen bits of the 32-bit multiply and divide register MD.

Whenever MDH or MDL is updated via software, the Multiply/Divide Register In Use flag (MDRIU) in the Multiply/Divide Control register (MDC) is set to ‘1’. The MDRIU flag is cleared, whenever register MDL is read via software.

MDC
Multiply/Divide Control Reg. **SFR ($FF0E_H/87_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	MDRIU	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	r(w)h	-	-	-	-

Field	Bits	Type	Description
MDRIU	4	rwh	**Multiply/Divide Register In Use** 0 Cleared when MDL is read via software. 1 Set when MDL or MDH is written via software, or when a multiply or divide instruction is executed.

Note: The MDRIU flag indicates the usage of register MD (MDL and MDH). In this case MD must be saved prior to a new multiplication or division operation.

4.9 DSP Data Processing (MAC Unit)

The new CoXXX arithmetic instructions are performed in the MAC unit. The MAC unit provides single-instruction-cycle, non-pipelined, 32-bit additions; 32-bit subtraction; right and left shifts; 16-bit by 16-bit multiplication; and multiplication with cumulative subtraction/addition. The MAC unit includes the following major components, shown in **Figure 4-18**:

- 16-bit by 16-bit signed/unsigned multiplier with signed result[1)]
- Concatenation Unit
- Scaler (one-bit left shifter) for fractional computing
- 40-bit Adder/Subtracter
- 40-bit Signed Accumulator
- Data Limiter
- Accumulator Shifter
- Repeat Counter

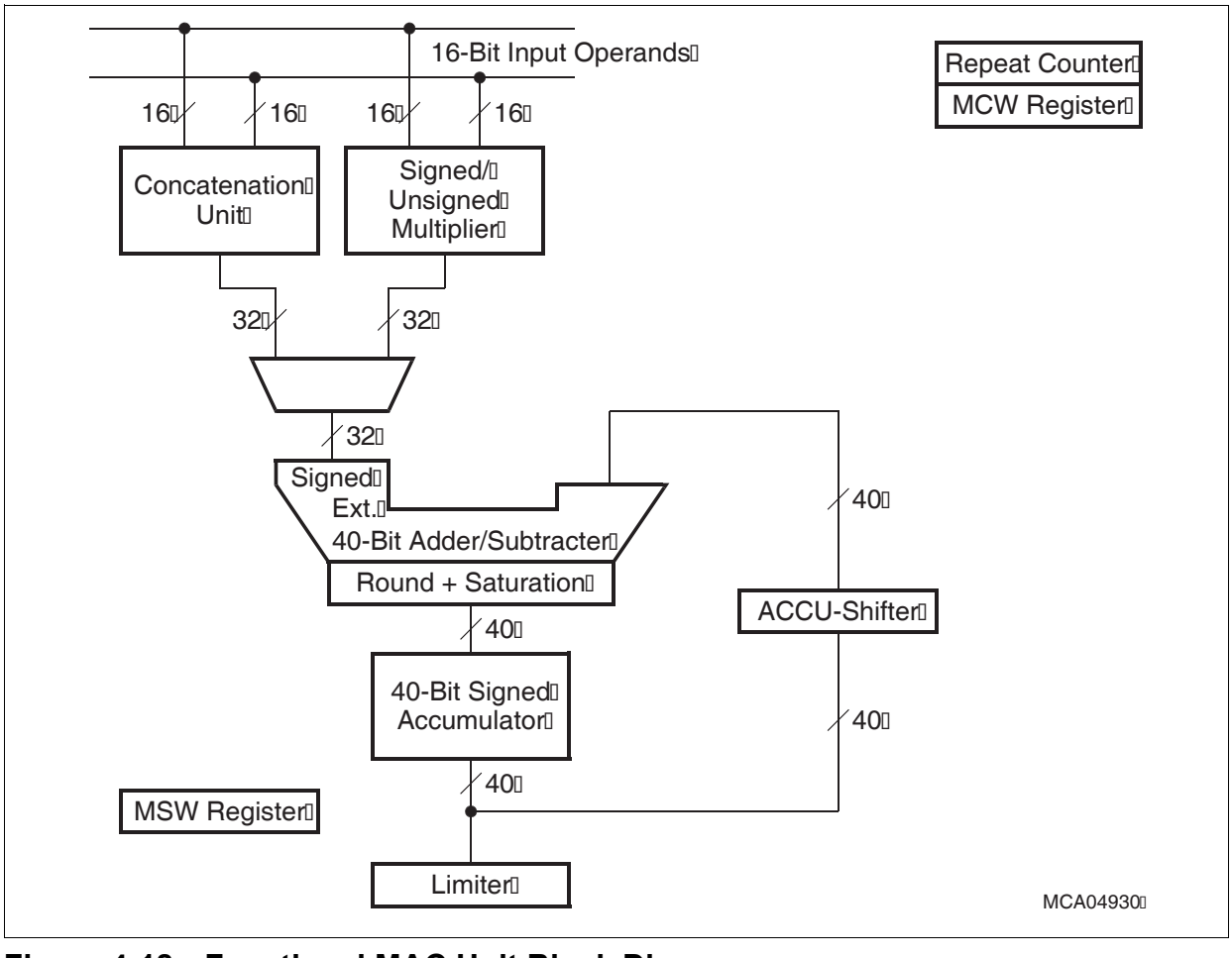


Figure 4-18 Functional MAC Unit Block Diagram

1) The same hardware-multiplier is used in the ALU.

4.9.1 MAC Unit Control

The working register of the MAC unit is a dedicated 40-bit accumulator register. A set of consistent flags is automatically updated in status register MSW after each MAC operation. These flags allow branching on specific conditions. Unlike the PSW flags, these flags are not preserved automatically by the CPU upon entry into an interrupt or trap routine. All dedicated MAC registers must be saved on the stack if the MAC unit is shared between different tasks and interrupts. General properties of the MAC unit are selected via the MAC control word MCW.

MCW
MAC Control Word **SFR ($FFDC_H/EE_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	**MP**	**MS**	-	-	-	-	-	-	-	-	-
-	-	-	-	-	rw	rw	-	-	-	-	-	-	-	-	-

Field	Bits	Type	Description
MP	10	rw	**One-Bit Scaler Control** 0 Multiplier product shift disabled 1 Multiplier product shift enabled for signed multiplications
MS	9	rw	**Saturation Control** 0 Saturation disabled 1 Saturation to 32-bit value enabled

4.9.2 Representation of Numbers and Rounding

The XE16x supports the 2's complement representation of binary numbers. In this format, the sign bit is the MSB of the binary word. This is set to zero for positive numbers and set to one for negative numbers. Unsigned numbers are supported only by multiply/multiply-accumulate instructions which specify whether each operand is signed or unsigned.

In 2's complement fractional format, the N-bit operand is represented using the 1.[N-1] format (1 signed bit, N-1 fractional bits). Such a format can represent numbers between -1 and +1 - $2^{-[N-1]}$. This format is supported when bit MP of register MCW is set.

The XE16x implements 2's complement rounding. With this rounding type, one is added to the bit to the right of the rounding point (bit 15 of MAL), before truncation (MAL is cleared).

4.9.3 The 16-bit by 16-bit Signed/Unsigned Multiplier and Scaler

The multiplier executes 16-bit by 16-bit parallel signed/unsigned fractional and integer multiplication in one CPU-cycle. The multiplier allows the multiplication of unsigned and signed operands. The result is always presented in a signed fractional or integer format.

The result of the multiplication feeds a one-bit scaler to allow compensation for the extra sign bit gained in multiplying two 16-bit 2's complement numbers.

4.9.4 Concatenation Unit

The concatenation unit enables the MAC unit to perform 32-bit arithmetic operations in one CPU cycle. The concatenation unit concatenates two 16-bit operands to a 32-bit operand before the 32-bit arithmetic operation is executed in the 40-bit adder/subtracter. The second required operand is always the current accumulator contents. The concatenation unit is also used to pre-load the accumulator with a 32-bit value.

4.9.5 One-bit Scaler

The one-bit scaler can shift the result of the concatenation unit or the output of the multiplier one bit to the left. The scaler is controlled by the executed instruction for the concatenation or by control bit MP in register MCW.

If bit MP is set the product is shifted one bit to the left to compensate for the extra sign bit gained in multiplying two 16-bit 2's-complement numbers. The enabled automatic shift is performed only if both input operands are signed.

4.9.6 The 40-bit Adder/Subtracter

The 40-bit Adder/Subtracter allows intermediate overflows in a series of multiply/accumulate operations. The Adder/Subtracter has two input ports. The 40-bit port is the feedback of the accumulator output through the ACCU-Shifter to the Adder/Subtracter. The 32-bit port is the input port for the operand coming from the one-bit Scaler. The 32-bit operands are signed and extended to 40 bits before the addition/subtraction is performed.

The output of the Adder/Subtracter goes to the accumulator. It is also possible to round the result and to saturate it on a 32-bit value automatically after every accumulation. The round operation is performed by adding $00'0000'8000_H$ to the result. Automatic saturation is enabled by setting the saturation control bit MS in register MCW.

When the accumulator is in the overflow saturation mode and an overflow occurs, the accumulator is loaded with either the most positive or the most negative value representable in a 32-bit value, depending on the direction of the overflow as well as on the arithmetic used. The value of the accumulator upon saturation is either $00'7FFF'FFFF_H$ (positive) or $FF'8000'0000_H$ (negative).

4.9.7 The Data Limiter

Saturation arithmetic is also provided to selectively limit overflow when reading the accumulator by means of a **CoSTORE <destination>., MAS** instruction. Limiting is performed on the MAC-Unit accumulator. If the contents of the accumulator can be represented in the destination operand size without overflow, then the data limiter is disabled and the operand is not modified. If the contents of the accumulator cannot be represented without overflow in the destination operand size, the limiter will substitute a “limited” data as explained in **Table 4-25**:

Table 4-25 Limiter Output

ME-flag	MN-flag	Output of Limiter
0	x	unchanged
1	0	$7FFF_H$
1	1	8000_H

Note: In this particular case, both the accumulator and the status register are not affected. MAS is readable by means of a CoSTORE instruction only.

4.9.8 The Accumulator Shifter

The accumulator shifter is a parallel shifter with a 40-bit input and a 40-bit output. The source accumulator shifting operations are:

- No shift (Unmodified)
- Up to 16-bit Arithmetic Left Shift
- Up to 16-bit Arithmetic Right Shift

Notice that bits ME, MSV, and MSL in register MSW are affected by left shifts; therefore, if the saturation mechanism is enabled (MS) the behavior is similar to the one of the Adder/Subtracter.

Note: Certain precautions are required in case of left shift with saturation enabled. Generally, if MAE contains significant bits, then the 32-bit value in the accumulator is to be saturated. However, it is possible that left shift may move some significant bits out of the accumulator. The 40-bit result will be misinterpreted and will be either not saturated or saturated incorrectly. There is a chance that the result of left shift may produce a result which can saturate an original positive number to the minimum negative value, or vice versa.

4.9.9 The 40-bit Signed Accumulator Register

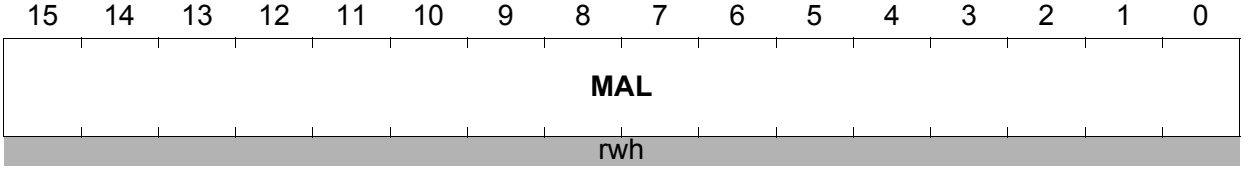
The 40-bit accumulator consists of three concatenated registers MAE, MAH, and MAL. MAE is 8 bits wide, MAH and MAL are 16 bits wide. MAE is the Most Significant Byte of the 40-bit accumulator. This byte performs a guarding function. MAE is accessed as the lower byte of register MSW.

When MAH is written, the value in the accumulator is automatically adjusted to signed extended 40-bit format. That means MAL is cleared and MAE will be automatically loaded with zeros for a positive number (the most significant bit of MAH is 0), and with ones for a negative number (the most significant bit of MAH is 1), representing the extended 40-bit negative number in 2's complement notation. One may see that the extended 40-bit value is equal to the 32-bit value without extension. In other words, after this extension, MAE does not contain significant bits. Generally, this condition is present when the highest 9 bits of the 40-bit signed result are the same.

During the accumulator operations, an overflow may happen and the result may not fit into 32 bits and MAE will change. The extension flag "E" in register MSW is set when the signed result in the accumulator has exceeded the 32-bit boundary. This condition is present when the highest 9 bits of the 40-bit signed result are not the same, i.e. MAE contains significant bits.

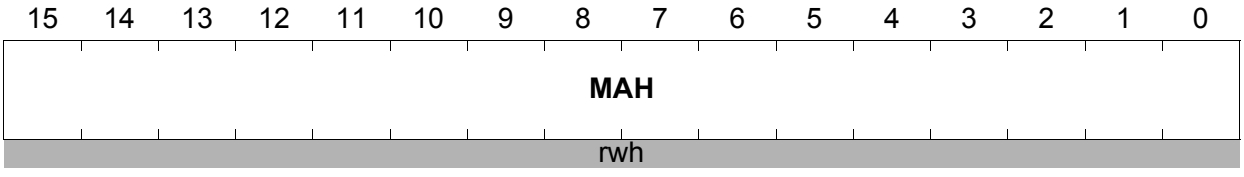
Most CoXXX operations specify the 40-bit accumulator register as a source and/or a destination operand.

MAL
Accumulator Low Word **SFR ($FE5C_H/2E_H$)** **Reset Value: 0000_H**



Field	Bits	Type	Description
MAL	[15:0]	rwh	**Low Part of Accumulator** The 40-bit accumulator is completed by the accumulator high word (MAH) and bitfield MAE

MAH
Accumulator High Word **SFR ($FE5E_H/2F_H$)** **Reset Value: 0000_H**

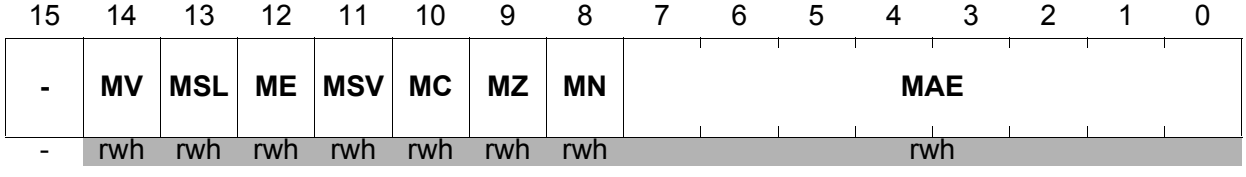


Field	Bits	Type	Description
MAH	[15:0]	rwh	**High Part of Accumulator** The 40-bit accumulator is completed by the accumulator low word (MAL) and bitfield MAE

4.9.10 The MAC Unit Status Word MSW

The upper byte of register MSW (bit-addressable) shows the current status of the MAC Unit. The lower byte of register MSW represents the 8-bit MAC accumulator extension, building the 40-bit accumulator together with registers MAH and MAL.

MSW
MAC Status Word **SFR ($FFDE_H/EF_H$)** **Reset Value: 0000_H**



Field	Bits	Type	Description
MV	14	rwh	**Overflow Flag** 0 No Overflow produced 1 Overflow produced
MSL	13	rwh	**Sticky Limit Flag** 0 Result was not saturated 1 Result was saturated
ME	12	rwh	**MAC Extension Flag** 0 MAE does not contain significant bits 1 MAE contains significant bits
MSV	11	rwh	**Sticky Overflow Flag** 0 No Overflow occurred 1 Overflow occurred
MC	10	rwh	**Carry Flag** 0 No carry/borrow produced 1 Carry/borrow produced
MZ	9	rwh	**Zero Flag** 0 MAC result is not zero 1 MAC result is zero
MN	8	rwh	**Negative Result** 0 MAC result is positive 1 MAC result is negative
MAE	[7:0]	rwh	**MAC Accumulator Extension** The most significant bits of the 40-bit accumulator, completing registers MAH and MAL

MAC Unit Status (MV, MN, MZ, MC, MSV, ME, MSL)

These condition flags indicate the MAC status resulting from the most recently performed MAC operation. These flags are controlled by the majority of MAC instructions according to specific rules. Those rules depend on the instruction managing the MAC or data movement operation.

After execution of an instruction which explicitly updates register MSW, the condition flags may no longer represent an actual MAC status. An explicit write operation to register MSW supersedes the condition flag values implicitly generated by the MAC unit. An explicit read access returns the value of register MSW after execution of the immediately preceding instruction. Register MSW can be accessed via any instruction capable of accessing an SFR.

Note: After reset, all MAC status bits are cleared.

MN-Flag: For the majority of the MAC operations, the MN-flag is set to 1 if the most significant bit of the result contains a 1; otherwise, it is cleared. In the case of integer operations, the MN-flag can be interpreted as the sign bit of the result (negative: MN = 1, positive: MN = 0). Negative numbers are always represented as the 2's complement of the corresponding positive number. The range of signed numbers extends from $80'0000'0000_H$ to $7F'FFFF'FFFF_H$.

MZ-Flag: The MZ-flag is normally set to 1 if the result of a MAC operation equals zero; otherwise, it is cleared.

MC-Flag: After a MAC addition, the MC-flag indicates that a "Carry" from the most significant bit of the accumulator extension MAE has been generated. After a MAC subtraction or a MAC comparison, the MC-flag indicates a "Borrow" representing the logical negation of a "Carry" for the addition. This means that the MC-flag is set to 1 if **no** "Carry" from the most significant bit of the accumulator has been generated during a subtraction. Subtraction is performed by the MAC Unit as a 2's complement addition and the MC-flag is cleared when this complement addition caused a "Carry".
For left-shift MAC operations, the MC-flag represents the value of the bit shifted out last. Right-shift MAC operations always clear the MC-flag. The arithmetic right-shift MAC operation can set the MC-flag if the enabled round operation generates a "Carry" from the most significant bit of the accumulator extension MAE.

MSV-Flag: The addition, subtraction, 2's complement, and round operations always set the MSV-flag to 1 if the MAC result exceeds the maximum range of 40-bit signed numbers. If the MSV-flag indicates an arithmetic overflow, the MAC result of an operation is not valid.
The MSV-flag is a 'Sticky Bit'. Once set, other MAC operations cannot affect the status of the MSV-flag. Only a direct write operation can clear the MSV-flag.

ME-Flag: The ME-flag is set if the accumulator extension MAE contains significant bits, that means if the nine highest accumulator bits are not all equal.

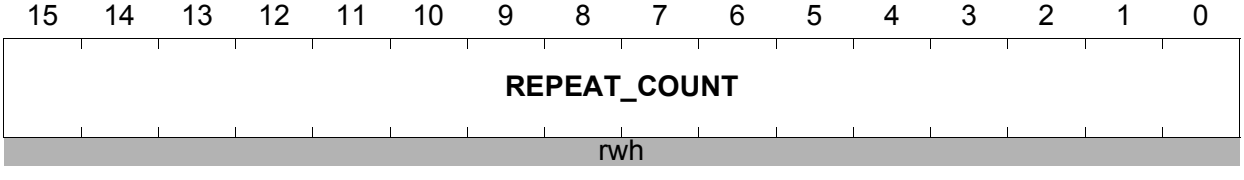
MSL-Flag: The MSL-flag is set if an automatic saturation of the accumulator has happened. The automatic saturation is enabled if bit MS in register MCW is set. The MSL-Flag can be also set by instructions which limit the contents of the accumulator. If the accumulator has been limited, the MSL-Flag is set.
The MSL-Flag is a 'Sticky Bit'. Once set, it cannot be affected by the other MAC operations. Only a direct write operation can clear the MSL-flag.

MV-Flag: The addition, subtraction, and accumulation operations set the MV-flag to 1 if the result exceeds the maximum range of signed numbers ($80'0000'0000_H$ to $7F'FFFF'FFFF_H$); otherwise, the MV-flag is cleared. Note that if the MV-flag indicates an arithmetic overflow, the result of the integer addition, integer subtraction, or accumulation is not valid.

4.9.11 The Repeat Counter MRW

The Repeat Counter MRW controls the number of repetitions a loop must be executed. The register must be pre-loaded before it can be used with -USRx CoXXX operations. MAC operations are able to decrement this counter. When a -USRx CoXXX instruction is executed, MRW is checked for zero **before** being decremented. If MRW equals zero, bit USRx is set and MRW is not further decremented. Register **MRW** can be accessed via any instruction capable of accessing a SFR.

MRW
MAC Repeat Word **SFR ($FFDA_H/ED_H$)** **Reset Value: 0000_H**



Field	Bits	Type	Description
REPEAT_ COUNT	[15:0]	rwh	16-bit loop counter

All CoXXX instructions have a 3-bit wide repeat control field 'rrr' (bit positions [31:29]) in the operand field to control the MRW repeat counter. **Table 4-26** lists the possible encodings.

Table 4-26 Encoding of MAC Repeat Word Control

Code in ‘rrr’	Effect on Repeat Counter
000_B	regular CoXXX instruction
001_B	RESERVED
010_B	‘-USR0 CoXXX’ instruction, decrements repeat counter and sets bit USR0 if MRW is zero
011_B	‘-USR1 CoXXX’ instruction, decrements repeat counter and sets bit USR1 if MRW is zero
$1XX_B$	RESERVED

Note: Bit USR0 has been a general purpose flag also in previous architectures. To prevent collisions due to using this flag by programmer or compiler, use ‘-USR0 C0XXX’ instructions very carefully.

The following example shows a loop which is executed 20 times. Every time the CoMACM instruction is executed, the MRW counter is decremented.

```
          MOV    MRW, #19          ;Pre-load loop counter
loop01:
-USR1     CoMACM [IDX0+], [R0+]    ;Calculate and decrement MSW
          ADD    R2,#0002H
          JMPA   cc_nusr1, loop01 ;Repeat loop until USR1 is set
```

Note: Because correctly predicted JMPA is executed in 0-cycle, it offers the functionality of a repeat instruction.

4.10 Constant Registers

All bits of these bit-addressable registers are fixed to 0 or 1 by hardware. These registers can be read only. Register ZEROS/ONES can be used as a register-addressable constant of all zeros or all ones, for example for bit manipulation or mask generation. The constant registers can be accessed via any instruction capable of addressing an SFR.

ZEROS
Zeros Register **SFR ($FF1C_H/8E_H$)** **Reset Value: 0000_H**

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

Field	Bits	Type	Description
0	[15:0]	r	**Constant Zero Bit**

ONES
Ones Register **SFR ($FF1E_H/8F_H$)** **Reset Value: $FFFF_H$**

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

Field	Bits	Type	Description
1	[15:0]	r	**Constant One Bit**

5 Interrupt and Trap Functions

The architecture of the XE16x supports several mechanisms for fast and flexible response to service requests from various internal or external sources. Different kinds of exceptions are handled in a similar way:

- Interrupts generated by the Interrupt Controller (ITC)
- DMA transfers issued by the Peripheral Event Controller (PEC)
- Traps caused by the TRAP instruction or issued by faults or specific system states

Normal Interrupt Processing

The CPU temporarily suspends current program execution and branches to an interrupt service routine to service an interrupt requesting device. The current program status (IP, PSW, also CSP in segmentation mode) is saved on the internal system stack. A prioritization scheme with 16 priority levels allows the user to specify the order in which multiple interrupt requests are to be handled.

Interrupt Processing via the Peripheral Event Controller (PEC)

A faster alternative to normal software controlled interrupt processing is servicing an interrupt requesting device with the XE16x's integrated Peripheral Event Controller (PEC). Triggered by an interrupt request, the PEC performs a single word or byte data transfer between any two locations through one of eight programmable PEC Service Channels. During a PEC transfer, normal program execution of the CPU is halted. No internal program status information needs to be saved. The same prioritization scheme is used for PEC service as for normal interrupt processing.

Trap Functions

Trap functions are activated in response to special conditions that occur during the execution of instructions. A trap can also be caused externally via the External Service Request pins, $\overline{\text{ESR}}$x. Several hardware trap functions are provided to handle erroneous conditions and exceptions arising during instruction execution. Hardware traps always have highest priority and cause immediate system reaction. The software trap function is invoked by the TRAP instruction that generates a software interrupt for a specified interrupt vector. For all types of traps, the current program status is saved on the system stack.

External Interrupt Processing

Although the XE16x does not provide dedicated interrupt pins, it allows connection of external interrupt sources and provides several mechanisms to react to external events including standard inputs, non-maskable interrupts, and fast external interrupts. Except for the non-maskable interrupt and the reset input, these interrupt functions are alternate port functions.

5.1 Interrupt System Structure

The XE16x provides 96 separate interrupt nodes assignable to 16 priority levels, with 8 sub-levels (group priority) on each level. In order to support modular and consistent software design techniques, most sources of an interrupt or PEC request are supplied with a separate interrupt control register and an interrupt vector. The control register contains the interrupt request flag, the interrupt enable bit, and the interrupt priority of the associated source. Each source request is then activated by one specific event, determined by the selected operating mode of the respective device. For efficient resource usage, multi-source interrupt nodes are also incorporated. These nodes can be activated by several source requests, such as by different kinds of errors in the serial interfaces. However, specific status flags which identify the type of error are implemented in the serial channels' control registers. Additional sharing of interrupt nodes is supported via interrupt subnode control registers.

The XE16x provides a vectored interrupt system. In this system specific vector locations in the memory space are reserved for the reset, trap, and interrupt service functions. Whenever a request occurs, the CPU branches to the location that is associated with the respective interrupt source. This allows direct identification of the source which caused the request. The Class B hardware traps all share the same interrupt vector. The status flags in the Trap Flag Register (TFR) can then be used to determine which exception caused the trap. For the special software TRAP instruction, the vector address is specified by the operand field of the instruction, which is a seven bit trap number.

The reserved vector locations build a jump table in the low end of a segment (selected by register VECSEG) in the XE16x's address space. The jump table consists of the appropriate jump instructions which transfer control to the interrupt or trap service routines and which may be located anywhere within the address space. The entries of the jump table are located at the lowest addresses in the selected code segment. Each entry occupies 2, 4, 8, or 16 words (selected by bitfield VECSC in register CPUCON1), providing room for at least one doubleword instruction. The respective vector location results from multiplying the trap number by the selected step width ($2^{(VECSC+2)}$).

All pending interrupt requests are arbitrated. The arbitration winner is indicated to the CPU together with its priority level and action request. The CPU triggers the corresponding action based on the required functionality (normal interrupt, PEC, jump table cache, etc.) of the arbitration winner.

An action request will be accepted by the CPU if the requesting source has a higher priority than the current CPU priority level and interrupts are globally enabled. If the requesting source has a lower (or equal) interrupt level priority than the current CPU task, it remains pending.

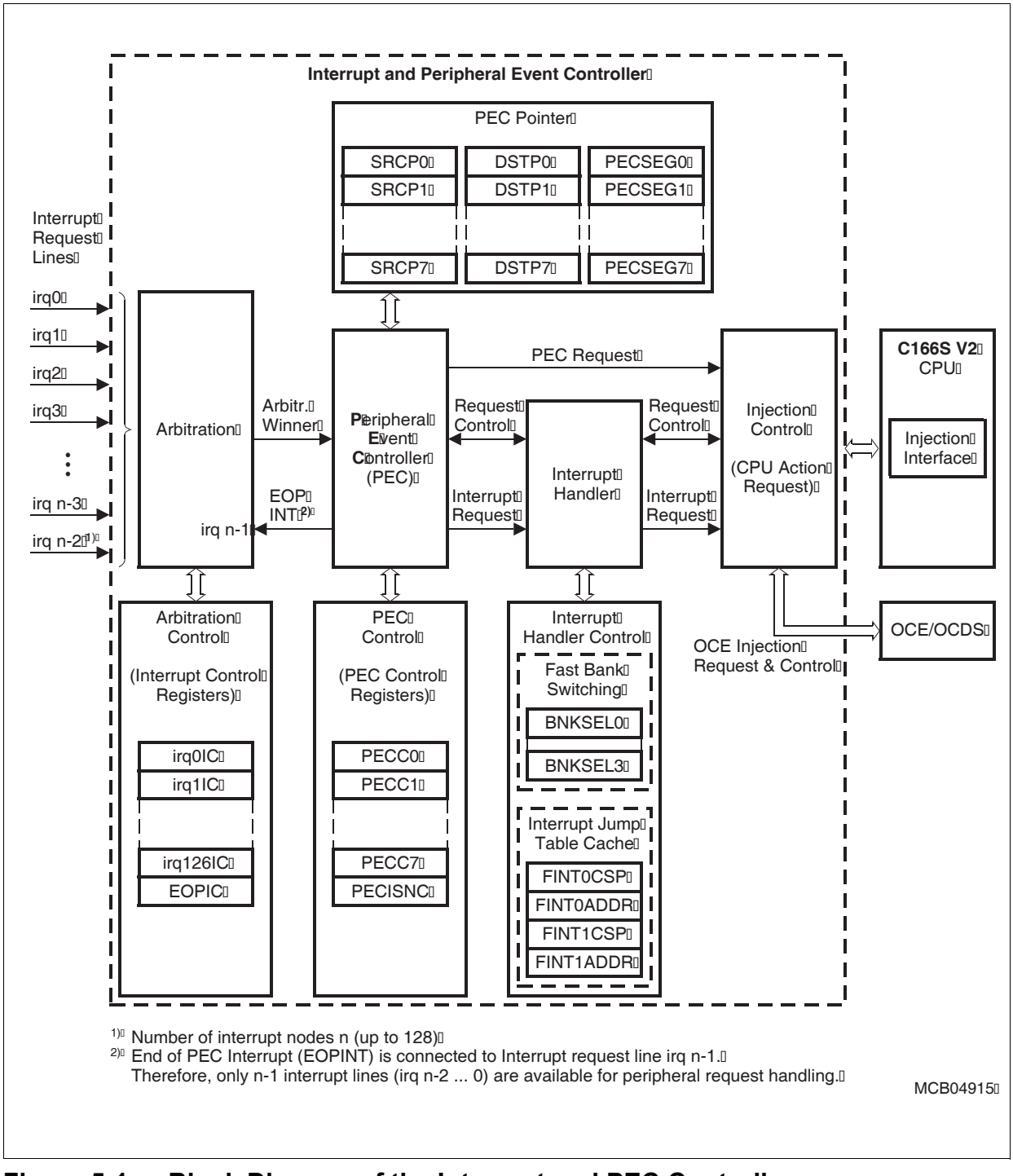


Figure 5-1 Block Diagram of the Interrupt and PEC Controller

5.2 Interrupt Arbitration and Control

The XE16x's interrupt arbitration system handles interrupt requests from up to 80 sources. Interrupt requests may be triggered either by the on-chip peripherals or by external inputs.

Interrupt processing is controlled globally by register PSW through a general interrupt enable bit (IEN) and the CPU priority field (ILVL). Additionally, the different interrupt sources are controlled individually by their specific interrupt control registers (… IC). Thus, the acceptance of requests by the CPU is determined by both the individual interrupt control registers and by the PSW. PEC services are controlled by the respective PECCx register and by the source and destination pointers which specify the task of the respective PEC service channel.

An interrupt request sets the associated interrupt request flag xxIR. If the requesting interrupt node is enabled by the associated interrupt enable bit xxIE arbitration starts with the next clock cycle, or after completion of an arbitration cycle that is already in progress. All interrupt requests pending at the beginning of a new arbitration cycle are considered, independently from when they were actually requested.

Figure 5-2 shows the three-stage interrupt prioritization scheme:

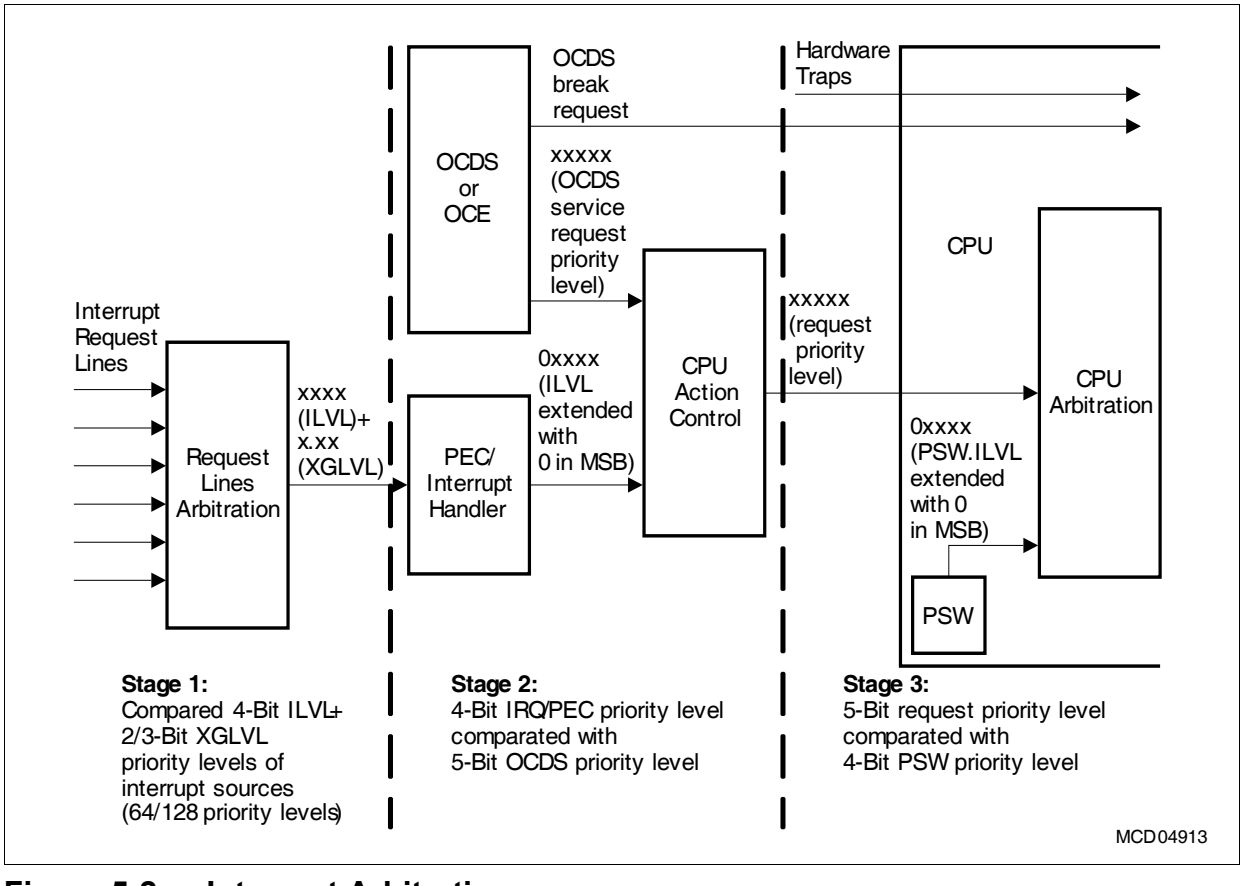


Figure 5-2 Interrupt Arbitration

The interrupt prioritization is done in three stages:

- Select one of the active interrupt requests
- Compare the priority levels of the selected interrupt request and an eventual OCDS service request
- Compare the priority level of the resulting request with the actual CPU priority level

The First Arbitration Stage

compares the priority levels of the active interrupt request lines. The interrupt priority level of each requestor is defined by bitfield ILVL in the respective xxIC register. The extended group priority level XGLVL (combined from bitfields GPX and GLVL) defines up to eight sub-priorities within one interrupt level. The group priority level distinguishes interrupt requests assigned to the same priority level, so one winner can be determined.

Note: All interrupt request sources that are enabled and programmed to the same interrupt priority level (ILVL) must have different group priority levels. Otherwise, an incorrect interrupt vector will be generated.

The Second Arbitration Stage

compares the priority of the first stage winner with the priority of OCDS service requests. OCDS service requests bypass the first stage of arbitration and go directly to the CPU Action Control Unit. The CPU Action Control Unit compares the winner's 4-bit priority level (disregarding the group level) with the 5-bit OCDS service request priority. The 4-bit ILVL of the interrupt request is extended to a 5-bit value with MSB = 0. This means that any OCDS request with MSB = 1 will always win the second stage arbitration. However, if there is a conflict between an OCDS request and an interrupt request, the interrupt request wins.

The Third Arbitration Stage

compares the priority level of the second stage winner with the priority of the current CPU task. An action request will be accepted by the CPU only if the priority level of the request is higher than the current CPU priority level (bitfield ILVL in register PSW) and if interrupt and PEC requests are globally enabled by the global interrupt enable flag IEN in register PSW. To compare with the 5-bit priority level of the second stage winner, the 4-bit CPU priority level is extended to a 5-bit value with MSB = 0. This means that any request with MSB = 1 will always interrupt the current CPU task. If the requestor has a priority level lower than or equal to the current CPU task, the request remains pending.

Note: Priority level 0000_B is the default level of the CPU. Therefore, a request on interrupt priority level 0000_B will be arbitrated, but the CPU will never accept an action request on this level. However, every individually enabled interrupt request (including all denied interrupt requests and priority level 0000_B requests) triggers a CPU wake-up from idle state independent of the global interrupt enable bit IEN.

Both the OCDS break requests and the hardware traps bypass the arbitration scheme and go directly to the core (see also **Figure 5-2**).

The arbitration process starts with an enabled interrupt request and stays active as long as an interrupt request is pending. If no interrupt request is pending the arbitration is stopped to save power.

Interrupt Control Registers

The control functions for each interrupt node are grouped in a dedicated interrupt control register (xxIC, where “xx” stands for a mnemonic for the respective node). All interrupt control registers are organized identically. The lower 9 bits of an interrupt control register contain the complete interrupt control and status information of the associated source required during one round of prioritization (arbitration cycle); the upper 7 bits are reserved for future use. All interrupt control registers are bit-addressable and all bits can be read or written via software. Therefore, each interrupt source can be programmed or modified with just one instruction.

xxIC
Interrupt Control Register **(E)SFR ($yyyy_H/zz_H$)** **Reset Value: - 000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	GPX	xxIR	xxIE	ILVL				GLVL	
-	-	-	-	-	-	-	rw	rwh	rw	rw				rw	

Field	Bits	Type	Description
GPX	8	rw	**Group Priority Extension** Completes bitfield GLVL to the 3-bit group level
xxIR[1)]	7	rwh	**Interrupt Request Flag** 0 No request pending 1 This source has raised an interrupt request
xxIE	6	rw	**Interrupt Enable Control Bit** (individually enables/disables a specific source) 0 Interrupt request is disabled 1 Interrupt request is enabled
ILVL	[5:2]	rw	**Interrupt Priority Level** FH Highest priority level … … 0H Lowest priority level

Field	Bits	Type	Description
GLVL	[1:0]	rw	**Group Priority Level** (Is completed by bit GPX to the 3-bit group level) 3H Highest priority level … … 0H Lowest priority level

1) Bit xxIR supports bit-protection.

When accessing interrupt control registers through instructions which operate on word data types, their upper 7 bits (15 … 9) will return zeros when read, and will discard written data. It is recommended to always write zeros to these bit positions. The layout of the interrupt control registers shown below applies to each xxIC register, where “xx” represents the mnemonic for the respective source.

The **Interrupt Request Flag** is set by hardware whenever a service request from its respective source occurs. It is cleared automatically upon entry into the interrupt service routine or upon a PEC service. In the case of PEC service, the Interrupt Request flag remains set if the COUNT field in register PECCx of the selected PEC channel decrements to zero and bit EOPINT is cleared. This allows a normal CPU interrupt to respond to a completed PEC block transfer on the same priority level.

Note: Modifying the Interrupt Request flag via software causes the same effects as if it had been set or cleared by hardware.

The **Interrupt Enable Control Bit** determines whether the respective interrupt node takes part in the arbitration process (enabled) or not (disabled). The associated request flag will be set upon a source request in any case. The occurrence of an interrupt request can so be polled via xxIR even while the node is disabled.

Note: In this case the interrupt request flag xxIR is not cleared automatically but must be cleared via software.

Interrupt Priority Level and Group Level

The four bits of bitfield ILVL specify the priority level of a service request for the arbitration of simultaneous requests. The priority increases with the numerical value of ILVL: so, 0000_B is the lowest and 1111_B is the highest priority level.

When more than one interrupt request on a specific level becomes active at the same time, the values in the respective bitfields GPX and GLVL are used for second level arbitration to select one request to be serviced. Again, the group priority increases with the numerical value of the concatenation of bitfields GPX and GLVL, so 000_B is the lowest and 111_B is the highest group priority.

Note: All interrupt request sources enabled and programmed to the same priority level must always be programmed to different group priorities. Otherwise, an incorrect interrupt vector will be generated.

Upon entry into the interrupt service routine, the priority level of the source that won the arbitration and whose priority level is higher than the current CPU level, is copied into bitfield ILVL of register PSW after pushing the old PSW contents onto the stack.

The interrupt system of the XE16x allows nesting of up to 15 interrupt service routines of different priority levels (level 0 cannot be arbitrated).

Interrupt requests programmed to priority levels 15 … 8 (i.e., ILVL = $1XXX_B$) can be serviced by the PEC if the associated PEC channel is properly assigned and enabled (please refer to **Section 5.4**). Interrupt requests programmed to priority levels 7 through 1 will always be serviced by normal interrupt processing.

Note: Priority level 0000_B is the default level of the CPU. Therefore, a request on level 0 will never be serviced because it can never interrupt the CPU. However, an individually enabled interrupt request (independent of bit IEN) on level 0000_B will reactivate the CPU.

General Interrupt Control Functions in Register PSW

The acceptance of an interrupt request depends on the current CPU priority level (bitfield ILVL in register PSW) and the global interrupt enable control bit IEN in register PSW (see **Section 4.8**).

CPU Priority ILVL defines the current level for the operation of the CPU. This bitfield reflects the priority level of the routine currently executed. Upon entry into an interrupt service routine, this bitfield is updated with the priority level of the request being serviced. The PSW is saved on the system stack before the request is serviced. The CPU level determines the minimum interrupt priority level which will be serviced. Any request on the same or a lower level will not be acknowledged. The current CPU priority level may be adjusted via software to control which interrupt request sources will be acknowledged. PEC transfers do not really interrupt the CPU, but rather “steal” a single cycle, so PEC services do not influence the ILVL field in the PSW.

Hardware traps switch the CPU level to maximum priority (i.e. 15) so no interrupt or PEC requests will be acknowledged while an exception trap service routine is executed.

Note: The TRAP instruction does not change the CPU level, so software invoked trap service routines may be interrupted by higher requests.

Interrupt Enable bit IEN globally enables or disables PEC operation and the acceptance of interrupts by the CPU. When IEN is cleared, no new interrupt requests are accepted by the CPU (see also **Section 4.3.4**). When IEN is set to 1, all interrupt sources, which have been individually enabled by the interrupt enable bits in their associated control registers, are globally enabled. Traps are non-maskable and are, therefore, not affected by the IEN bit.

Note: To generate requests, interrupt sources must be also enabled by the interrupt enable bits in their associated control register.

Register Bank Select bitfield BANK defines the currently used register bank for the CPU operation. When the CPU enters an interrupt service routine, this bitfield is updated to select the register bank associated with the serviced request:

- Requests on priority levels 15 … 12 use the register bank pre-selected via the respective bitfield GPRSELx in the corresponding BNKSEL register
- Requests on priority levels 11 … 1 always use the global register bank, i.e. BANK = 00_B
- Hardware traps always use the global register bank, i.e. BANK = 00_B
- The TRAP instruction does not change the current register bank

Temporary Control of Interrupts

Interrupt requests may be temporarily disabled and enabled during the execution of the software. This may be required to exclude specific interrupt sources based on the current status of the application. In particular, this is necessary to achieve a deterministic execution of time-critical code sequences.

Interrupt requests in the XE16x can be disabled and enabled on three different levels:

- Control all interrupt requests globally
- Control configurable groups of interrupt requests
- Control single interrupt requests

Global interrupt control is achieved with a single instruction:

```
BCLR IEN              ;Clear IEN flag (causes pipeline restart)
```

Groups of interrupts (classes) are defined and controlled by software as described in **Section 5.5**.

Specific interrupt control is achieved by controlling the enable bits in the associated interrupt control registers.

```
BCLR T2IE             ;Clear enable flag to disable intr.node
```

Due to pipeline effects, however, an interrupt request may be executed after the corresponding node was disabled, if the request coincides with clearing the enable flag.

If the application must avoid this, the following sequence can be used, ensuring that no interrupt requests from this source will be serviced after disabling the interrupt node:

```
BCLR IEN              ;Globally disable interrupts
BCLR T2IE             ;Disable Timer 2 interrupt node
JNB  T2IE, Next       ;Any instruction reading T2IC can be used
Next:                 ;(assures that T2IC is written by BCLR
                      ;before being read by JNB or other instr.)
BSET IEN              ;Globally enable interrupts again
```

Please note that the sequence above blindly controls the global enable flag. If the global setting must not be changed, the code sequence can be enhanced, as shown below:

```
JNB  IEN, GlobalIntOff
BCLR IEN              ;Globally disable interrupts
BCLR T2IE             ;Disable Timer 2 interrupt node
JNB  T2IE, Next       ;Any instruction reading T2IC can be used
Next:                 ;(assures that T2IC is written by BCLR
                      ;before being read by JNB or other instr.)
BSET IEN              ;Globally enable interrupts again
JMPR cc_uc, Continue
GlobalIntOff:         ;Interrupts are globally disabled anyway
BCLR T2IE             ;Disable Timer 2 interrupt node

JNB  T2IE, Continue ;Reading T2IC can be omitted if the next
Continue:           ;few instructions do not set IEN
...
```

The same function can easily be implemented as a C macro:

```
#define Disable_One_Interrupt(IE_bit) \\
{if(IEN) {IEN=0; IE_bit=0; while (IE_bit); IEN=1;} else
{IE_bit=0; while IE_bit);}}
Usage Example:
Disable_One_Interrupt(T2IE) ; // T2 interrupt enable flag
```

ATOMIC or EXTend sequences preserve the status of the interrupt arbitration when they begin. An accepted request is processed after the ATOMIC/EXTend sequence. Therefore, the following code sequence may not produce the desired result:

```
AvoidThis:
ATOMIC #3
BCLR T2IE             ;Disable Timer 2 interrupt node
NOP
NOP                   ;Timer 2 request may be processed
                      ;after this instruction!!!
```

5.3 Interrupt Vector Table

The XE16x provides a vectored interrupt system. This system reserves a set of specific memory locations, which are accessed automatically upon the respective trigger event. Entries for the following events are provided:

- Reset (hardware, software, watchdog)
- Traps (hardware-generated by fault conditions or via TRAP instruction)
- Interrupt service requests

Whenever a request is accepted, the CPU branches to the location associated with the respective trigger source. This vector position directly identifies the source causing the request, with **two exceptions**:

- Class B hardware traps all share the same interrupt vector. The status flags in the Trap Flag Register (TFR) are used to determine which exception caused the trap. For details, see **Section 5.11**.
- An interrupt node may be shared by several interrupt requests, e.g. within a module. Additional flags identify the requesting source, so the software can handle each request individually. For details, see **Section 5.7**.

The reserved vector locations build a vector table located in the address space of the XE16x. The vector table usually contains the appropriate jump instructions that transfer control to the interrupt or trap service routines. These routines may be located anywhere within the address space. The location and organization of the vector table is programmable.

The Vector Segment register VECSEG defines the segment of the Vector Table (can be located in all segments, except for reserved areas).

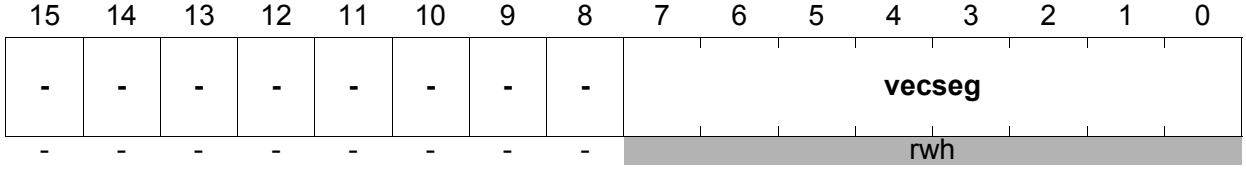
Bitfield VECSC in register CPUCON1 defines the space between two adjacent vectors (can be 2, 4, 8, or 16 words). For a summary of register CPUCON1, please refer to **Section 4.4**.

Each vector location has an offset address to the segment base address of the vector table (given by VECSEG). The offset can be easily calculated by multiplying the vector number with the vector space programmed in bitfield VECSC.

Table 0-2 lists all sources capable of requesting interrupt or PEC service in the XE16x, the associated interrupt vector locations, the associated vector numbers, and the associated interrupt control registers.

Note: All interrupt nodes which are currently not used by their associated modules or are not connected to a module in the actual derivative may be used to generate software controlled interrupt requests by setting the respective IR flag.

VECSEG
Vector Segment Pointer **SFR ($FF12_H/89_H$)** **Reset Value: Table 5-1**



Field	Bits	Type	Description
vecseg	[7:0]	rwh	Segment number of the Vector Table

The reset value of register VECSEG, that means the initial location of the vector table, depends on the reset configuration. **Table 5-1** lists the possible locations. This is required because the vector table also provides the reset vector.

Table 5-1 Reset Values for Register VECSEG

Initial Value	Reset Configuration
0000_H	Standard start from external memory
$00C0_H$	Standard start from Internal Program Memory
$00E0_H$	Execute bootstrap loader code

Table 5-2 XE16x Interrupt Nodes

Source of Interrupt or PEC Service Request	Control Register	Vector Location[1)]	Trap Number
CAPCOM Register 16, or ERU Request 0	CC2_CC16IC	xx'0040$_H$	10$_H$ / 16$_D$
CAPCOM Register 17, or ERU Request 1	CC2_CC17IC	xx'0044$_H$	11$_H$ / 17$_D$
CAPCOM Register 18, or ERU Request 2	CC2_CC18IC	xx'0048$_H$	12$_H$ / 18$_D$
CAPCOM Register 19, or ERU Request 3	CC2_CC19IC	xx'004C$_H$	13$_H$ / 19$_D$
CAPCOM Register 20, or USIC0 Request 6	CC2_CC20IC	xx'0050$_H$	14$_H$ / 20$_D$
CAPCOM Register 21, or USIC0 Request 7	CC2_CC21IC	xx'0054$_H$	15$_H$ / 21$_D$
CAPCOM Register 22, or USIC1 Request 6	CC2_CC22IC	xx'0058$_H$	16$_H$ / 22$_D$
CAPCOM Register 23, or USIC1 Request 7	CC2_CC23IC	xx'005C$_H$	17$_H$ / 23$_D$
CAPCOM Register 24, or ERU Request 0	CC2_CC24IC	xx'0060$_H$	18$_H$ / 24$_D$
CAPCOM Register 25, or ERU Request 1	CC2_CC25IC	xx'0064$_H$	19$_H$ / 25$_D$
CAPCOM Register 26, or ERU Request 2	CC2_CC26IC	xx'0068$_H$	1A$_H$ / 26$_D$
CAPCOM Register 27, or ERU Request 3	CC2_CC27IC	xx'006C$_H$	1B$_H$ / 27$_D$
CAPCOM Register 28, or USIC2 Request 6	CC2_CC28IC	xx'0070$_H$	1C$_H$ / 28$_D$
CAPCOM Register 29, or USIC2 Request 7	CC2_CC29IC	xx'0074$_H$	1D$_H$ / 29$_D$
CAPCOM Register 30	CC2_CC30IC	xx'0078$_H$	1E$_H$ / 30$_D$
CAPCOM Register 31	CC2_CC31IC	xx'007C$_H$	1F$_H$ / 31$_D$
GPT1 Timer 2	GPT12E_T2IC	xx'0080$_H$	20$_H$ / 32$_D$
GPT1 Timer 3	GPT12E_T3IC	xx'0084$_H$	21$_H$ / 33$_D$
GPT1 Timer 4	GPT12E_T4IC	xx'0088$_H$	22$_H$ / 34$_D$

Table 5-2 XE16x Interrupt Nodes (cont'd)

Source of Interrupt or PEC Service Request	Control Register	Vector Location[1)]	Trap Number
GPT2 Timer 5	GPT12E_T5IC	xx'008C$_H$	23$_H$ / 35$_D$
GPT2 Timer 6	GPT12E_T6IC	xx'0090$_H$	24$_H$ / 36$_D$
GPT2 CAPREL Register	GPT12E_CRIC	xx'0094$_H$	25$_H$ / 37$_D$
CAPCOM Timer 7	CC2_T7IC	xx'0098$_H$	26$_H$ / 38$_D$
CAPCOM Timer 8	CC2_T8IC	xx'009C$_H$	27$_H$ / 39$_D$
A/D Converter Request 0	ADC_0IC	xx'00A0$_H$	28$_H$ / 40$_D$
A/D Converter Request 1	ADC_1IC	xx'00A4$_H$	29$_H$ / 41$_D$
A/D Converter Request 2	ADC_2IC	xx'00A8$_H$	2A$_H$ / 42$_D$
A/D Converter Request 3	ADC_3IC	xx'00AC$_H$	2B$_H$ / 43$_D$
A/D Converter Request 4	ADC_4IC	xx'00B0$_H$	2C$_H$ / 44$_D$
A/D Converter Request 5	ADC_5IC	xx'00B4$_H$	2D$_H$ / 45$_D$
A/D Converter Request 6	ADC_6IC	xx'00B8$_H$	2E$_H$ / 46$_D$
A/D Converter Request 7	ADC_7IC	xx'00BC$_H$	2F$_H$ / 47$_D$
CCU60 Request 0	CCU60_0IC	xx'00C0$_H$	30$_H$ / 48$_D$
CCU60 Request 1	CCU60_1IC	xx'00C4$_H$	31$_H$ / 49$_D$
CCU60 Request 2	CCU60_2IC	xx'00C8$_H$	32$_H$ / 50$_D$
CCU60 Request 3	CCU60_3IC	xx'00CC$_H$	33$_H$ / 51$_D$
CCU61 Request 0	CCU61_0IC	xx'00D0$_H$	34$_H$ / 52$_D$
CCU61 Request 1	CCU61_1IC	xx'00D4$_H$	35$_H$ / 53$_D$
CCU61 Request 2	CCU61_2IC	xx'00D8$_H$	36$_H$ / 54$_D$
CCU61 Request 3	CCU61_3IC	xx'00DC$_H$	37$_H$ / 55$_D$
CCU62 Request 0	CCU62_0IC	xx'00E0$_H$	38$_H$ / 56$_D$
CCU62 Request 1	CCU62_1IC	xx'00E4$_H$	39$_H$ / 57$_D$
CCU62 Request 2	CCU62_2IC	xx'00E8$_H$	3A$_H$ / 58$_D$
CCU62 Request 3	CCU62_3IC	xx'00EC$_H$	3B$_H$ / 59$_D$
CCU63 Request 0	CCU63_0IC	xx'00F0$_H$	3C$_H$ / 60$_D$
CCU63 Request 1	CCU63_1IC	xx'00F4$_H$	3D$_H$ / 61$_D$
CCU63 Request 2	CCU63_2IC	xx'00F8$_H$	3E$_H$ / 62$_D$
CCU63 Request 3	CCU63_3IC	xx'00FC$_H$	3F$_H$ / 63$_D$
CAN Request 0	CAN_0IC	xx'0100$_H$	40$_H$ / 64$_D$

Table 5-2 XE16x Interrupt Nodes (cont'd)

Source of Interrupt or PEC Service Request	Control Register	Vector Location[1)]	Trap Number
CAN Request 1	CAN_1IC	xx'0104$_H$	41$_H$ / 65$_D$
CAN Request 2	CAN_2IC	xx'0108$_H$	42$_H$ / 66$_D$
CAN Request 3	CAN_3IC	xx'010C$_H$	43$_H$ / 67$_D$
CAN Request 4	CAN_4IC	xx'0110$_H$	44$_H$ / 68$_D$
CAN Request 5	CAN_5IC	xx'0114$_H$	45$_H$ / 69$_D$
CAN Request 6	CAN_6IC	xx'0118$_H$	46$_H$ / 70$_D$
CAN Request 7	CAN_7IC	xx'011C$_H$	47$_H$ / 71$_D$
CAN Request 8	CAN_8IC	xx'0120$_H$	48$_H$ / 72$_D$
CAN Request 9	CAN_9IC	xx'0124$_H$	49$_H$ / 73$_D$
CAN Request 10	CAN_10IC	xx'0128$_H$	4A$_H$ / 74$_D$
CAN Request 11	CAN_11IC	xx'012C$_H$	4B$_H$ / 75$_D$
CAN Request 12	CAN_12IC	xx'0130$_H$	4C$_H$ / 76$_D$
CAN Request 13	CAN_13IC	xx'0134$_H$	4D$_H$ / 77$_D$
CAN Request 14	CAN_14IC	xx'0138$_H$	4E$_H$ / 78$_D$
CAN Request 15	CAN_15IC	xx'013C$_H$	4F$_H$ / 79$_D$
USIC0 Request 0	U0C0_0IC	xx'0140$_H$	50$_H$ / 80$_D$
USIC0 Request 1	U0C0_1IC	xx'0144$_H$	51$_H$ / 81$_D$
USIC0 Request 2	U0C0_2IC	xx'0148$_H$	52$_H$ / 82$_D$
USIC0 Request 3	U0C1_0IC	xx'014C$_H$	53$_H$ / 83$_D$
USIC0 Request 4	U0C1_1IC	xx'0150$_H$	54$_H$ / 84$_D$
USIC0 Request 5	U0C1_2IC	xx'0154$_H$	55$_H$ / 85$_D$
USIC1 Request 0	U1C0_0IC	xx'0158$_H$	56$_H$ / 86$_D$
USIC1 Request 1	U1C0_1IC	xx'015C$_H$	57$_H$ / 87$_D$
USIC1 Request 2	U1C0_2IC	xx'0160$_H$	58$_H$ / 88$_D$
USIC1 Request 3	U1C1_0IC	xx'0164$_H$	59$_H$ / 89$_D$
USIC1 Request 4	U1C1_1IC	xx'0168$_H$	5A$_H$ / 90$_D$
USIC1 Request 5	U1C1_2IC	xx'016C$_H$	5B$_H$ / 91$_D$
USIC2 Request 0	U2C0_0IC	xx'0170$_H$	5C$_H$ / 92$_D$
USIC2 Request 1	U2C0_1IC	xx'0174$_H$	5D$_H$ / 93$_D$
USIC2 Request 2	U2C0_2IC	xx'0178$_H$	5E$_H$ / 94$_D$

Table 5-2 XE16x Interrupt Nodes (cont'd)

Source of Interrupt or PEC Service Request	Control Register	Vector Location[1)]	Trap Number
USIC2 Request 3	U2C1_0IC	xx'017C$_H$	5F$_H$ / 95$_D$
USIC2 Request 4	U2C1_1IC	xx'0180$_H$	60$_H$ / 96$_D$
USIC2 Request 5	U2C1_2IC	xx'0184$_H$	61$_H$ / 97$_D$
Unassigned node	–	xx'0188$_H$	62$_H$ / 98$_D$
Unassigned node	–	xx'018C$_H$	63$_H$ / 99$_D$
Unassigned node	–	xx'0190$_H$	64$_H$ / 100$_D$
Unassigned node	–	xx'0194$_H$	65$_H$ / 101$_D$
Unassigned node	–	xx'0198$_H$	66$_H$ / 102$_D$
Unassigned node	–	xx'019C$_H$	67$_H$ / 103$_D$
Unassigned node	–	xx'01A0$_H$	68$_H$ / 104$_D$
Unassigned node	–	xx'01A4$_H$	69$_H$ / 105$_D$
Unassigned node	–	xx'01A8$_H$	6A$_H$ / 106$_D$
SCU Request 1	SCU_1IC	xx'01AC$_H$	6B$_H$ / 107$_D$
SCU Request 0	SCU_0IC	xx'01B0$_H$	6C$_H$ / 108$_D$
Program Flash Modules	PFM_IC	xx'01B4$_H$	6D$_H$ / 109$_D$
RTC	RTC_IC	xx'01B8$_H$	6E$_H$ / 110$_D$
End of PEC Subchannel	EOPIC	xx'01BC$_H$	6F$_H$ / 111$_D$

1) Register VECSEG defines the segment where the vector table is located to.
Bitfield VECSC in register CPUCON1 defines the distance between two adjacent vectors. This table represents the default setting, with a distance of 4 (two words) between two vectors.

Table 5-3 lists the vector locations for hardware traps and the corresponding status flags in register TFR. It also lists the priorities of trap service for those cases in which more than one trap condition might be detected within the same instruction. After any reset (hardware reset, software reset instruction SRST, or reset by watchdog timer overflow) program execution starts at the reset vector at location xx'0000$_H$. Reset conditions have priority over every other system activity and, therefore, have the highest priority (trap priority III).

Software traps may be initiated to any defined vector location. A service routine entered via a software TRAP instruction is always executed on the current CPU priority level which is indicated in bitfield ILVL in register PSW. This means that routines entered via the software TRAP instruction can be interrupted by all hardware traps or higher level interrupt requests.

Table 5-3 Hardware Trap Summary

Exception Condition	Trap Flag	Trap Vector	Vector Location[1)]	Trap Number	Trap Priority
Reset Functions	–	RESET	xx'0000$_H$	00$_H$	III
Class A Hardware Traps:					
• System Request 0	SR0	SR0TRAP	xx'0008$_H$	02$_H$	II
• Stack Overflow	STKOF	STOTRAP	xx'0010$_H$	04$_H$	II
• Stack Underflow	STKUF	STUTRAP	xx'0018$_H$	06$_H$	II
• Software Break	SOFTBRK	SBRKTRAP	xx'0020$_H$	08$_H$	II
Class B Hardware Traps:					
• System Request 1	SR1	BTRAP	xx'0028$_H$	0A$_H$	I
• Undefined Opcode	UNDOPC	BTRAP	xx'0028$_H$	0A$_H$	I
• Memory Access Error	ACER	BTRAP	xx'0028$_H$	0A$_H$	I
• Protected Instruction Fault	PRTFLT	BTRAP	xx'0028$_H$	0A$_H$	I
• Illegal Word Operand Access	ILLOPA	BTRAP	xx'0028$_H$	0A$_H$	I
Reserved	–	–	[2C$_H$ - 3C$_H$]	[0B$_H$ - 0F$_H$]	–
Software Traps: • TRAP Instruction	–	–	Any [xx'0000$_H$ - xx'01FC$_H$] in steps of 4$_H$	Any [00$_H$ - 7F$_H$]	Current CPU Priority

1) Register VECSEG defines the segment where the vector table is located to.
Bitfield VECSC in register CPUCON1 defines the distance between two adjacent vectors. This table represents the default setting, with a distance of 4 (two words) between two vectors.

Interrupt Jump Table Cache

Servicing an interrupt request via the vector table usually incurs two subsequent branches: an implicit branch to the vector location and an explicit branch to the actual service routine. The interrupt servicing time can be reduced by the Interrupt Jump Table Cache (ITC, also called “fast interrupt”). This feature eliminates the second explicit branch by directly providing the CPU with the service routine’s location.

The ITC provides two 24-bit pointers, so the CPU can directly branch to the respective service routines. These fast interrupts can be selected for two interrupt sources on priority levels 15 … 12.

The two pointers are each stored in a pair of interrupt jump table cache registers (FINTxADDR, FINTxCSP), which store a pointer’s segment and offset along with the priority level it shall be assigned to (select the same priority that is programmed for the respective interrupt node).

FINT0ADDR
Fast Interrupt Address Reg. 0 **XSFR ($EC02_H$/--)** **Reset Value: 0000_H**
FINT1ADDR
Fast Interrupt Address Reg. 1 **XSFR ($EC06_H$/--)** **Reset Value: 0000_H**

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

Field	Bits	Type	Description
ADDR	[15:1]	rw	**Address of Interrupt Service Routine** Specifies address bits 15 … 1 of the 24-bit pointer to the interrupt service routine. This word offset is concatenated with FINTxCSP.SEG.

FINT0CSP
Fast Interrupt Control Reg. 0 **XSFR ($EC00_H$/--)** **Reset Value: 0000_H**
FINT1CSP
Fast Interrupt Control Reg. 1 **XSFR ($EC04_H$/--)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN	-	-	GPX	ILVL		GLVL		SEG							
rw	-	-	rw	rw		rw		rw							

Field	Bits	Type	Description
EN	15	rw	**Fast Interrupt Enable** 0 The interrupt jump table cache is not used 1 The interrupt jump table cache is enabled, the vector table entry for the specified request is bypassed, the cache pointer is used
GPX	12	rw	**Group Priority Extension** Used together with bitfield GLVL
ILVL	[11:10]	rw	**Interrupt Priority Level** This selects the interrupt priority (15 … 12) of the request this pointer shall be assigned to 00 Interrupt priority level 12 (1100B) 01 Interrupt priority level 13 (1101B) 10 Interrupt priority level 14 (1110B) 11 Interrupt priority level 15 (1111B)
GLVL	[9:8]	rw	**Group Priority Level** Together with bit GPX this selects the group priority of the request this pointer shall be assigned to
SEG	[7:0]	rw	**Segment Number of Interrupt Service Routine** Specifies address bits 23 … 16 of the 24-bit pointer to the interrupt service routine, is concatenated with FINTxADDR.

5.4 Operation of the Peripheral Event Controller Channels

The XE16x's Peripheral Event Controller (PEC) provides 8 PEC service channels which move a single byte or word between any two locations. A PEC transfer can be triggered by an interrupt service request and is the fastest possible interrupt response. In many cases a PEC transfer is sufficient to service the respective peripheral request (for example, serial channels, etc.).

PEC transfers do not change the current context, but rather "steal" cycles from the CPU, so the current program status and context needs not to be saved and restored as with standard interrupts.

The PEC channels are controlled by a dedicated set of registers which are assigned to dedicated PEC resources:

- A 24-bit source pointer for each channel
- A 24-bit destination pointer for each channel
- A Channel Counter/Control register (PECCx) for each channel, selecting the operating mode for the respective channel
- Two interrupt control registers to control the operation of block transfers

5.4.1 The PECC Registers

The PECC registers control the action performed by the respective PEC channel.

Transfer Size (bit BWT) controls whether a byte or a word is moved during a PEC service cycle. This selection controls the transferred data size and the increment step for the pointer(s) to be modified.

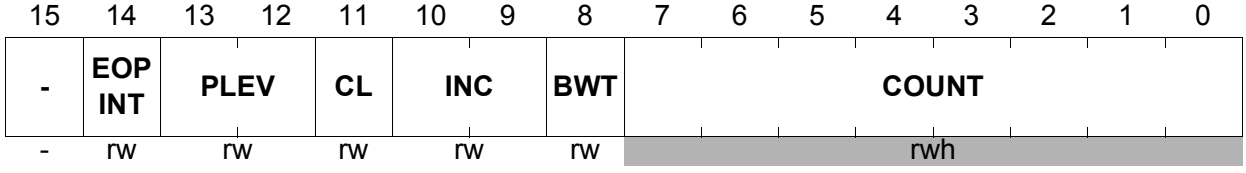
Pointer Modification (bitfield INC) controls, which of the PEC pointers is incremented after the PEC transfer. If the pointers are not modified (INC = 00_B), the respective channel will always move data from the same source to the same destination.

Transfer Control (bitfield COUNT) controls if the respective PEC channel remains active after the transfer or not. Bitfield COUNT also generally enables a PEC channel (COUNT > 00_H).

The PECC registers also select the assignment of PEC channels to interrupt priority levels (bitfield PLEV) and the interrupt behavior after PEC transfer completion (bit EOPINT).

Note: All interrupt request sources that are enabled and programmed for PEC service should use different channels. Otherwise, only one transfer will be performed for all simultaneous requests. When COUNT is decremented to 00_H, and the CPU is to be interrupted, an incorrect interrupt vector will be generated.
PEC transfers are executed only if their priority level is higher than the CPU level.

PECCx (x=0-7)
PEC Channel Control Reg. x SFR(FEC0$_H$+2*x) Reset Value: 0000$_H$



Field	Bits	Type	Description
EOPINT	14	rw	**End of PEC Interrupt Selection** 0_B End of PEC interrupt on the same (PEC) level 1_B End of PEC interrupt via separate node EOPIC
PLEV	[13:12]	rw	**PEC Level Selection** This bitfield controls the PEC channel assignment to an arbitration priority level (see section below)
CL	11	rw	**Channel Link Control** 0_B PEC channels work independently 1_B Pairs of PEC channels are linked together[1)]
INC	[10:9]	rw	**Increment Control** (Pointer Modification)[2)] 00_B Pointers are not modified 01_B Increment DSTPx by 1 or 2 (BWT = 1 or 0) 10_B Increment SRCPx by 1 or 2 (BWT = 1 or 0) 11_B Increment both DSTPx and SRCPx by 1 or 2
BWT	8	rw	**Byte/Word Transfer Selection** 0_B Transfer a word 1_B Transfer a byte
COUNT	[7:0]	rwh	**PEC Transfer Count** Counts PEC transfers and influences the channel's action (see **Section 5.4.3**)

1) For a functional description see "**Channel Link Mode for Data Chaining**".

2) Pointers are incremented/decremented only within the current segment.

Table 5-4 PEC Control Register Addresses

Register	Address	Reg. Space	Register	Address	Reg. Space
PECC0	FEC0$_H$ / 60$_H$	SFR	PECC4	FEC8$_H$ / 64$_H$	SFR
PECC1	FEC2$_H$ / 61$_H$	SFR	PECC5	FECA$_H$ / 65$_H$	SFR
PECC2	FEC4$_H$ / 62$_H$	SFR	PECC6	FECC$_H$ / 66$_H$	SFR
PECC3	FEC6$_H$ / 63$_H$	SFR	PECC7	FECE$_H$ / 67$_H$	SFR

The PEC channel number is derived from the respective ILVL (LSB) and GLVL, where the priority band (ILVL) is selected by the channel's bitfield PLEV (see **Table 5-5**). So, programming a source to priority level 15 (ILVL = 1111_B) selects the PEC channel group 7 … 4 with PLEV = 00_B; programming a source to priority level 14 (ILVL = 1110_B) selects the PEC channel group 3 … 0 with PLEV = 00_B; programming a source to priority level 10 (ILVL = 1010_B) selects the PEC channel group 3 … 0 with PLEV = 10_B. The actual PEC channel number is then determined by the group priority (levels 3 … 0, i.e. GPX = 0).

Simultaneous requests for PEC channels are prioritized according to the PEC channel number, where channel 0 has lowest and channel 7 has highest priority.

Note: All sources requesting PEC service must be programmed to different PEC channels. Otherwise, an incorrect PEC channel may be activated.

Table 5-5 PEC Channel Assignment

Selected PEC Channel	Group Level	Used Interrupt Priorities Depending on Bitfield PLEV			
		PLEV = 00_B	PLEV = 01_B	PLEV = 10_B	PLEV = 11_B
7	3	15	13	11	9
6	2				
5	1				
4	0				
3	3	14	12	10	8
2	2				
1	1				
0	0				

Table 5-6 shows in a few examples which action is executed with a given programming of an interrupt control register and a PEC channel.

Table 5-6 Interrupt Priority Examples

Priority Level		Type of Service		
Interr. Level	**Group Level**	**COUNT = 00_H, PLEV = XX_B**	**COUNT ≠ 00_H, PLEV = 00_B**	**COUNT ≠ 00_H, PLEV = 01_B**
1 1 1 1	1 1 1	CPU interrupt, level 15, group prio 7	CPU interrupt, level 15, group prio 7	CPU interrupt, level 15, group prio 7
1 **1 1 1**	**0 1 1**	CPU interrupt, level 15, group prio 3	PEC service, channel 7	CPU interrupt, level 15, group prio 3
1 **1 1 1**	**0 1 0**	CPU interrupt, level 15, group prio 2	PEC service, channel 6	CPU interrupt, level 15, group prio 2
1 **1 1 0**	**0 1 0**	CPU interrupt, level 14, group prio 2	PEC service, channel 2	CPU interrupt, level 14, group prio 2
1 1 0 1	1 1 0	CPU interrupt, level 13, group prio 6	CPU interrupt, level 13, group prio 6	CPU interrupt, level 13, group prio 6
1 **1 0 1**	**0 1 0**	CPU interrupt, level 13, group prio 2	CPU interrupt, level 13, group prio 2	PEC service, channel 6
0 0 0 1	0 1 1	CPU interrupt, level 1, group prio 3	CPU interrupt, level 1, group prio 3	CPU interrupt, level 1, group prio 3
0 0 0 1	0 0 0	CPU interrupt, level 1, group prio 0	CPU interrupt, level 1, group prio 0	CPU interrupt, level 1, group prio 0
0 0 0 0	X X X	No service!	No service!	No service!

Note: PEC service is only achieved when bit GPX = 0 and COUNT ≠ 0. Requests on levels 7 … 1 cannot initiate PEC transfers. They are always serviced by an interrupt service routine: no PECC register is associated and no COUNT field is checked.

5.4.2 The PEC Source and Destination Pointers

The PEC channels' source and destination pointers specify the locations between which the data is to be moved. Both 24-bit pointers are built by concatenating the 16-bit offset register (SRCPx or DSTPx) with the respective 8-bit segment bitfield (SRCSEGx or DSTSEGx, combined in register PECSEGx).

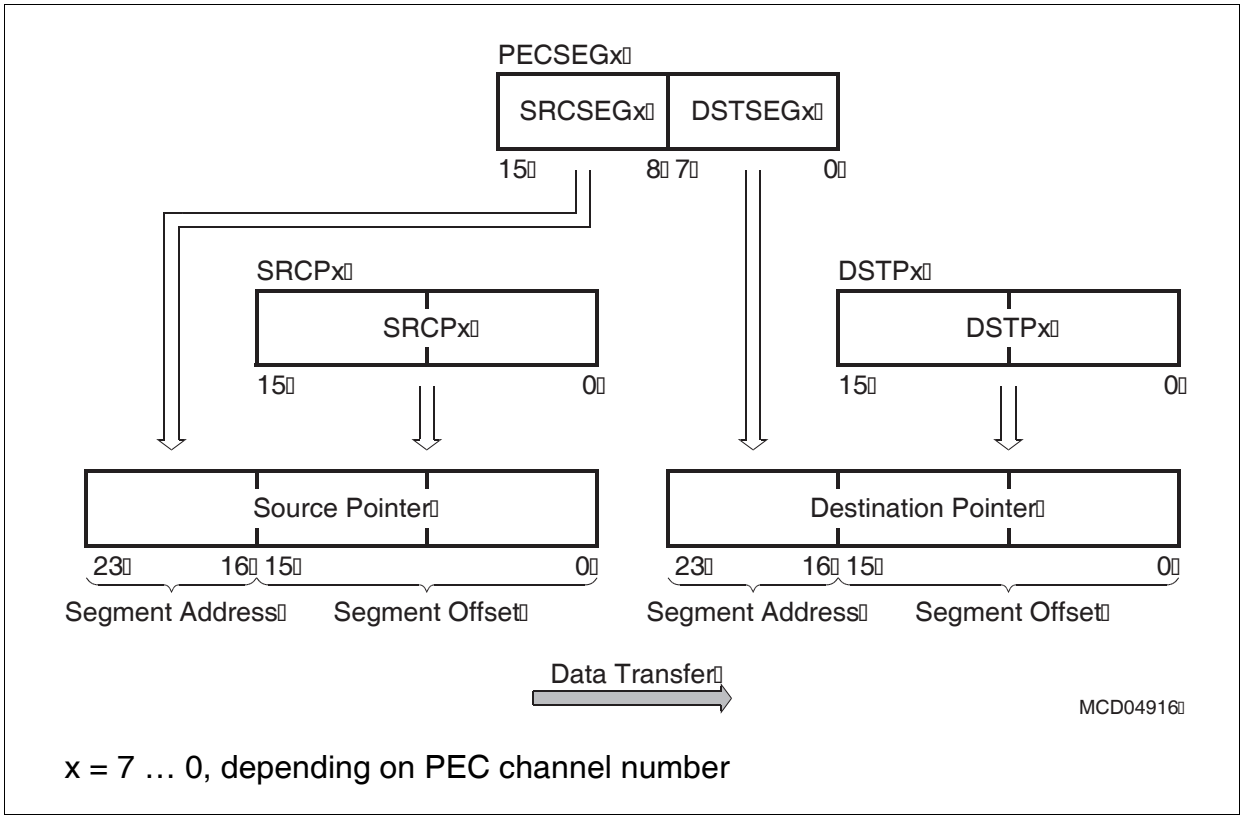


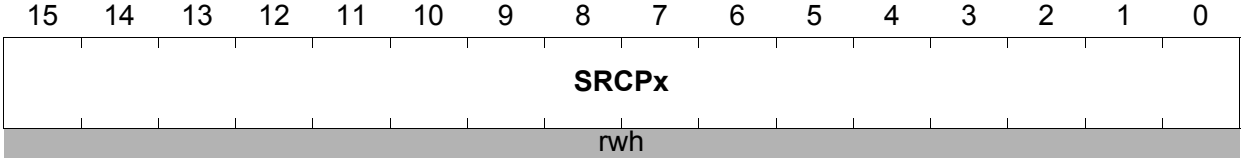
Figure 5-3 PEC Data Pointers

When a PEC pointer is automatically incremented after a transfer, only the offset part is incremented (SRCPx and/or DSTPx), while the respective segment part is not modified by hardware. Thus, a pointer may be incremented within the current segment, but may not cross the segment boundary. When a PEC pointer reaches the maximum offset ($FFFE_H$ for word transfers, $FFFF_H$ for byte transfers), it is not incremented further, but keeps its maximum offset value. This protects memory in adjacent segments from being overwritten unintentionally.

No explicit error event is generated by the system in case of a pointer saturation; therefore, it is the user's responsibility to prevent this condition.

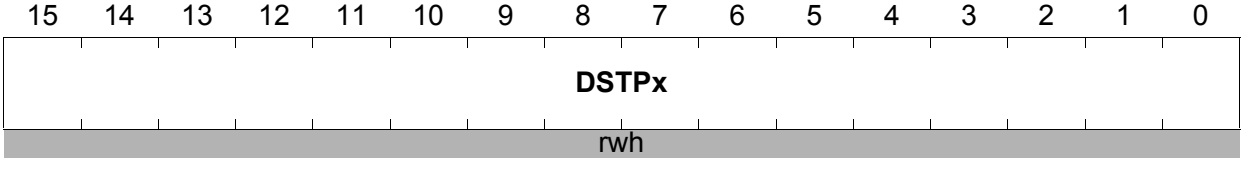
Note: PEC data transfers do not use the data page pointers DPP3 … DPP0. Unused PEC pointers may be used for general data storage.

SRCPx (x=0-7)
PEC Source Pointer x **XSFR(EC40$_H$+4*x)** **Reset Value: 0000$_H$**



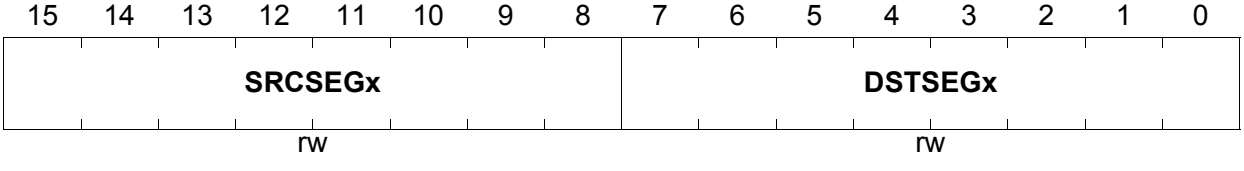
Field	Bits	Type	Description
SRCPx	[15:0]	rwh	**Source Pointer Offset of Channel x** Source address bits 15 … 0

DSTPx (x=0-7)
PEC Destination Pointer x **XSFR(EC42$_H$+4*x)** **Reset Value: 0000$_H$**



Field	Bits	Type	Description
DSTPx	[15:0]	rwh	**Destination Pointer Offset of Channel x** Destination address bits 15 … 0

PECSEGx (x=0-7)
PEC Segment Pointer x **XSFR(EC80$_H$+2*x)** **Reset Value: 0000$_H$**



Field	Bits	Type	Description
SRCSEGx	[15:8]	rw	**Source Pointer Segment of Channel x** Source address bits 23 … 16
DSTSEGx	[7:0]	rw	**Destination Pointer Segment of Channel x** Destination address bits 23 … 16

Table 5-7 PEC Data Pointer Register Addresses

Channel #	0	1	2	3	4	5	6	7
PECSEGx	$EC80_H$	$EC82_H$	$EC84_H$	$EC86_H$	$EC88_H$	$EC8A_H$	$EC8C_H$	$EC8E_H$
SRCPx	$EC40_H$	$EC44_H$	$EC48_H$	$EC4C_H$	$EC50_H$	$EC54_H$	$EC58_H$	$EC5C_H$
DSTPx	$EC42_H$	$EC46_H$	$EC4A_H$	$EC4E_H$	$EC52_H$	$EC56_H$	$EC5A_H$	$EC5E_H$

Note: If word data transfer is selected for a specific PEC channel (BWT = 0), the respective source and destination pointers must both contain a valid word address which points to an even byte boundary. Otherwise, the Illegal Word Access trap will be invoked when this channel is used.

5.4.3 PEC Transfer Control

The PEC Transfer Count Field COUNT controls the behavior of the respective PEC channel. The contents of bitfield COUNT select the action to be taken at the time the request is activated. COUNT may allow a specified number of PEC transfers, unlimited transfers, or no PEC service at all. **Table 5-8** summarizes, how the COUNT field, the interrupt requests flag IR, and the PEC channel action depend on the previous contents of COUNT.

Table 5-8 Influence of Bitfield COUNT

Previous COUNT	Modified COUNT	IR after Service	Action of PEC Channel and Comments
FF_H	FF_H	0	Move a Byte/Word Continuous transfer mode, i.e. COUNT is not modified
FE_H … 02_H	FD_H … 01_H	0	Move a Byte/Word and decrement COUNT
01_H	00_H	1	**EOPINT = 0** (channel-specific interrupt) Move a Byte/Word and leave request flag set, which triggers another request
		0	**EOPINT = 1** (separate end-of-PEC interrupt) Move a Byte/Word and clear request flag, set the respective PEC subnode request flag CxIR instead[1)]
00_H	00_H	–	**No PEC action!** Activate interrupt service routine rather than PEC channel

1) Setting a subnode request flag also sets flag EOPIR if the subnode request is enabled (CxIE = 1).

The PEC transfer counter allows service of a specified number of requests by the respective PEC channel, and then (when COUNT reaches 00_H) activation of an interrupt service routine, either associated with the PEC channel's priority level or with the general end-of-PEC interrupt. After each PEC transfer, the COUNT field is decremented (except for COUNT = FF_H) and the request flag is cleared to indicate that the request has been serviced.

When COUNT contains the value 00_H, the respective PEC channel remains idle and the associated interrupt service routine is activated instead. This allows servicing requests on all priority levels by standard interrupt service routines.

Continuous transfers are selected by the value FF_H in bitfield COUNT. In this case, COUNT is not modified and the respective PEC channel services any request until it is disabled again.

When COUNT is decremented from 01_H to 00_H after a transfer, a standard interrupt is requested which can then handle the end of the PEC block transfer (channel-specific interrupt or common end-of-PEC interrupt, see **Table 5-8**).

5.4.4 Channel Link Mode for Data Chaining

In channel link mode, every two PEC channels build a pair (channels 0+1, 2+3, 4+5, 6+7), where the two channels of a pair are activated in turn. Requests for the even channel trigger the currently active PEC channel (or the end-of-block interrupt), while requests for the odd channel only trigger its associated interrupt node. When the transfer count of one channel expires, control is switched to the other channel, and back. This mode supports data chaining where independent blocks of data can be transferred to the same destination (or vice versa), e.g. to build communication frames from several blocks, such as preamble, data, etc.

Channel link mode for a pair of channels is enabled if at least one of the channel link control bits (bit CL in register PECCx) of the respective pair is set. A linked channel pair is controlled by the priority-settings (level, group) for its even channel. After enabling channel link mode the even channel is active.

Channel linking is executed if the active channel's link control bit CL is 1 at the time its transfer count decrements from 1 to 0 (count > 0 before) and the transfer count of the other channel is non-zero. In this case the active channel issues an EOP interrupt request and the respective other channel of the pair is automatically selected.

Note: Channel linking always begins with the even channel.

Channel linking is terminated if the active channel's link control bit CL is 0 at the time its transfer count decrements from 1 to 0, or if the transfer count of the respective linked channel is zero. In this case an interrupt is triggered as selected by bit EOPINT (channel-specific or general EOP interrupt).

A data-chaining sequence using PEC channel linking is programmed by setting bit CL together with a transfer count value (> 0). This is repeated, triggered by the channel link interrupts, for the complete sequence. For the last transfer, the interrupt routine should clear the respective bit CL, so, at the end of the complete transfer, either a standard or an END of PEC interrupt can be selected by bit EOPINT of the last channel.

Note: To enable linking, initially both channels must receive a non-zero transfer count. For the rest of the sequence only the channel with the expired transfer count needs to be reconfigured.

5.4.5 PEC Interrupt Control

When the selected number of PEC transfers has been executed, the respective PEC channel is disabled and a standard interrupt service routine is activated instead. Each PEC channel can either activate the associated channel-specific interrupt node, or activate its associated PEC subnode request flag in register PECISNC, which then activates the common node request flag in register EOPIC (see **Figure 5-4**).

PECISNC
PEC Intr. Sub-Node Ctrl. Reg. **SFR ($FFD8_H/EC_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C7IR	C7IE	C6IR	C6IE	C5IR	C5IE	C4IR	C4IE	C3IR	C3IE	C2IR	C2IE	C1IR	C1IE	C0IR	C0IE
rwh	rw	rwh	rw	rwh	rw	rwh	rw	rwh	rw	rwh	rw	rwh	rw	rwh	rw

Field	Bits	Type	Description
CxIR (x = 0-7)	2*x+1	rwh	**Interrupt Request Flag of PEC Channel x** 0_B No request from PEC channel x pending 1_B PEC channel x has raised an end-of-PEC interrupt request *Note: These request flags must be cleared by SW.*
CxIE (x = 0-7)	2*x	rw	**Interrupt Enable Control Bit of PEC Channel x** (individually enables/disables a specific source) 0_B End-of-PEC request of channel x disabled 1_B End-of-PEC request of channel x enabled[1)]

1) It is recommended to clear an interrupt request flag (CxIR) before setting the respective enable flag (CxIE). Otherwise, former requests still pending cannot trigger a new interrupt request.

EOPIC
End-of-PEC Intr. Ctrl. Reg. **ESFR ($F19E_H/CF_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	GPX	EOP IR	EOP IE	ILVL				GLVL	
-	-	-	-	-	-	-	rw	rwh	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

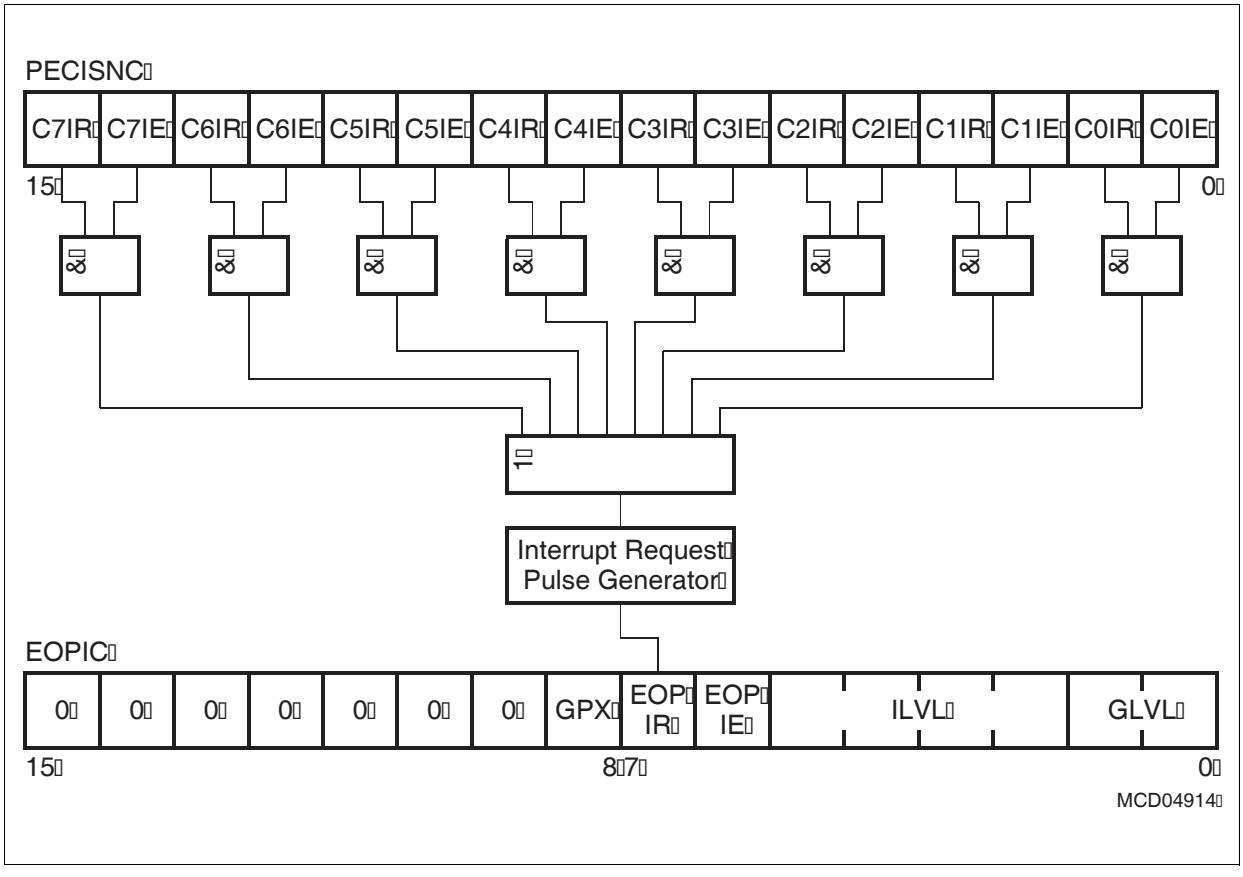


Figure 5-4 End of PEC Interrupt Sub Node

Note: The interrupt service routine must service and clear all currently active requests before terminating. Requests occurring later will set EOPIR again and the service routine will be re-entered.

5.5 Prioritization of Interrupt and PEC Service Requests

Interrupt and PEC service requests from all sources can be enabled so they are arbitrated and serviced (if they win), or they may be disabled, so their requests are disregarded and not serviced.

Enabling and disabling interrupt requests may be done via three mechanisms:

- Control Bits
- Priority Level
- ATOMIC and EXTended Instructions

Control Bits allow switching of each individual source “ON” or “OFF” so that it may generate a request or not. The control bits (xxIE) are located in the respective interrupt control registers. All interrupt requests may be enabled or disabled generally via bit IEN in register PSW. This control bit is the “main switch” which selects if requests from any source are accepted or not.
For a specific request to be arbitrated, the respective source’s enable bit and the global enable bit must both be set.

The Priority Level automatically selects a certain group of interrupt requests to be acknowledged and ignores all other requests. The priority level of the source that won the arbitration is compared against the CPU’s current level and the source is serviced only if its level is higher than the current CPU level. Changing the CPU level to a specific value via software blocks all requests on the same or a lower level. An interrupt source assigned to level 0 will be disabled and will never be serviced.

The ATOMIC and EXTend instructions automatically disable all interrupt requests for the duration of the following 1 … 4 instructions. This is useful for semaphore handling, for example, and does not require to re-enable the interrupt system after the inseparable instruction sequence.

Interrupt Class Management

An interrupt class covers a set of interrupt sources with the same importance, i.e. the same priority from the system’s viewpoint. Interrupts of the same class must not interrupt each other. The XE16x supports this function with two features:

- **Classes with up to eight members** can be established by using the same interrupt priority (ILVL) and assigning a dedicated group level to each member. This functionality is built-in and handled automatically by the interrupt controller.
- **Classes with more than eight members** can be established by using a number of adjacent interrupt priorities (ILVL) and the respective group levels (eight per ILVL). Each interrupt service routine within this class sets the CPU level to the highest interrupt priority within the class. All requests from the same or any lower level are blocked now, i.e. no request of this class will be accepted.

The example shown below establishes 3 interrupt classes which cover 2 or 3 interrupt priorities, depending on the number of members in a class. A level 6 interrupt disables all other sources in class 2 by changing the current CPU level to 8, which is the highest priority (ILVL) in class 2. Class 1 requests or PEC requests are still serviced, in this case.

In this way, the interrupt sources (excluding PEC requests) are assigned to 3 classes of priority rather than to 7 different levels, as the hardware support would do.

Table 5-9 Software Controlled Interrupt Classes (Example)

ILVL (Priority)	Group Level								Interpretation
	7	6	5	4	3	2	1	0	
15									PEC service on up to 8 channels
14									
13									
12	X	X	X	X	X	X	X	X	Interrupt Class 1 9 sources on 2 levels
11	X								
10									
9									
8	X	X	X	X	X	X	X	X	Interrupt Class 2 17 sources on 3 levels
7	X	X	X	X	X	X	X	X	
6	X								
5	X	X	X	X	X	X	X	X	Interrupt Class 3 9 sources on 2 levels
4	X								
3									
2									
1									
0									No service!

5.6 Context Switching and Saving Status

Before an interrupt request that has been arbitrated is actually serviced, the status of the current task is automatically saved on the system stack. The CPU status (PSW) is saved together with the location at which execution of the interrupted task is to be resumed after returning from the service routine. This return location is specified through the Instruction Pointer (IP) and, in the case of a segmented memory model, the Code Segment Pointer (CSP). Bit SGTDIS in register CPUCON1 controls how the return location is stored.

The system stack receives the PSW first, followed by the IP (unsegmented), or followed by CSP and then IP (segmented mode). This optimizes the usage of the system stack if segmentation is disabled.

The CPU priority field (ILVL in PSW) is updated with the priority of the interrupt request to be serviced, so the CPU now executes on the new level.

The register bank select field (BANK in PSW) is changed to select the register bank associated with the interrupt request. The association between interrupt requests and register banks are partly pre-defined and can partly be programmed.

The interrupt request flag of the source being serviced is cleared. IP and CSP are loaded with the vector associated with the requesting source, and the first instruction of the service routine is fetched from the vector location which is expected to branch to the actual service routine (except when the interrupt jump table cache is used). All other CPU resources, such as data page pointers and the context pointer, are not affected.

When the interrupt service routine is exited (RETI is executed), the status information is popped from the system stack in the reverse order, taking into account the value of bit SGTDIS.

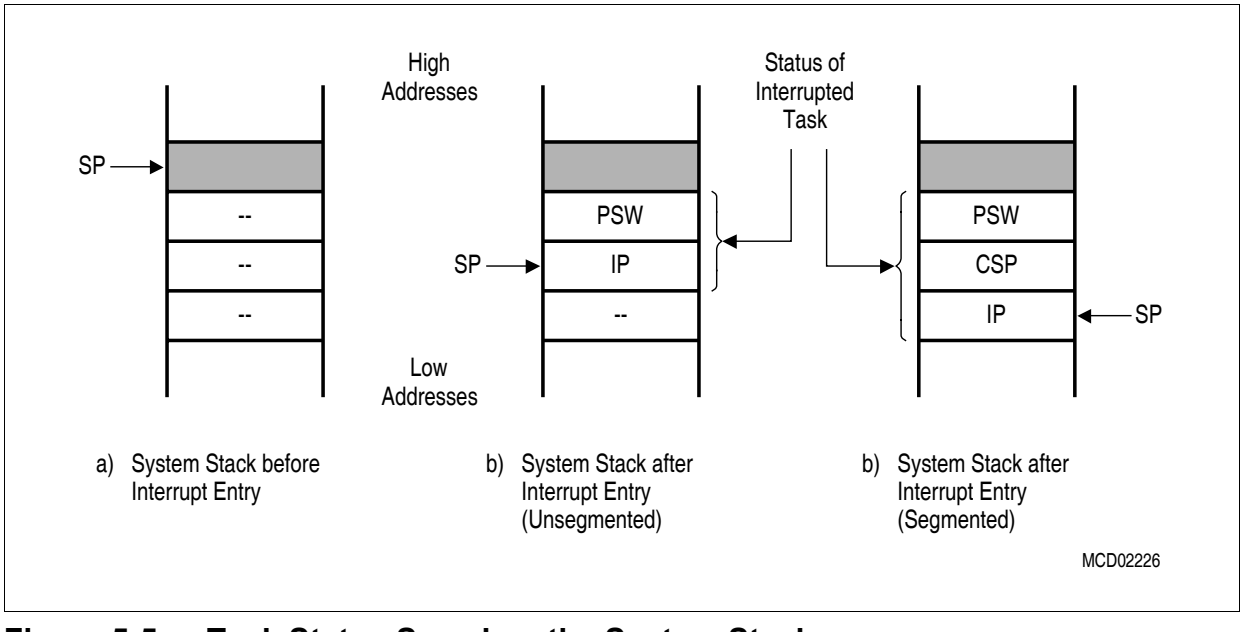


Figure 5-5 Task Status Saved on the System Stack

Context Switching

An interrupt service routine usually saves all the registers it uses on the stack and restores them before returning. The more registers a routine uses, the more time is spent saving and restoring. The XE16x allows switching the complete bank of CPU registers (GPRs) either automatically or with a single instruction, so the service routine executes within its own separate context (see also **Section 4.5.2**).

There are two ways to switch the context in the XE16x core:

Switching Context of the Global Register Bank changes the complete global register bank of CPU registers (GPRs) by changing the Context Pointer with a single instruction, so the service routine executes within its own separate context. The instruction “SCXT CP, #New_Bank” pushes the contents of the context pointer (CP) on the system stack and loads CP with the immediate value “New_Bank”; this in turn, selects a new register bank. The service routine may now use its “own registers”. This register bank is preserved when the service routine terminates, i.e. its contents are available on the next call. Before returning (RETI), the previous CP is simply POPped from the system stack, which returns the registers to the original global bank.

Resources used by the interrupting program, such as the DPPs, must eventually be saved and restored.

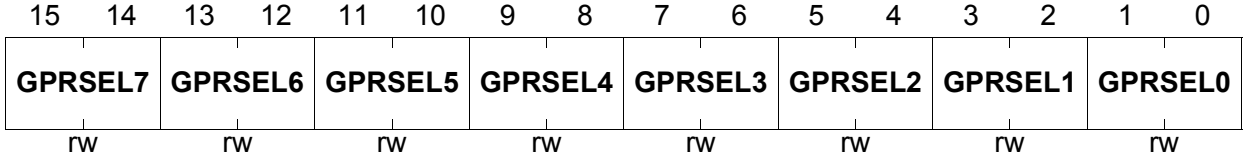
Note: There are certain timing restrictions during context switching that are associated with pipeline behavior.

Switching Context by changing the selected register bank automatically updates bitfield BANK to select one of the two local register banks or the current global register bank, so the service routine may now use its “own registers” directly. This local register bank is preserved when the service routine is terminated; thus, its contents are available on the next call.
When switching to the global register bank, the service routine usually must also switch the context of the global register bank to get a private set of GPRs, because the global bank is likely to be used by several tasks.

For interrupt priority levels 15 … 12 the target register bank can be pre-selected and then be switched automatically. The register bank selection registers BNKSELx provide a 2-bit field for each possible arbitration priority level. The respective bitfield is then copied to bitfield BANK in register PSW to select the register bank, as soon as the respective interrupt request is accepted.

Table 5-10 identifies the arbitration priority level assignment to the respective bitfields within the four register bank selection registers.

BNKSEL0
Register Bank Selection Reg. 0 **XSFR($EC20_H$)** **Reset Value: 0000_H**
BNKSEL1
Register Bank Selection Reg. 1 **XSFR($EC22_H$)** **Reset Value: 0000_H**
BNKSEL2
Register Bank Selection Reg. 2 **XSFR($EC24_H$)** **Reset Value: 0000_H**
BNKSEL3
Register Bank Selection Reg. 3 **XSFR($EC26_H$)** **Reset Value: 0000_H**



Field	Bits	Type	Description
GPRSEL0, **GPRSEL1,** **GPRSEL2,** **GPRSEL3,** **GPRSEL4,** **GPRSEL5,** **GPRSEL6,** **GPRSEL7**	[1:0], [3:2], [5:4], [7:6], [9:8], [11:10], [13:12], [15:14]	rw	**Register Bank Selection** 00_B Global register bank 01_B Reserved 10_B Local register bank 1 11_B Local register bank 2

Table 5-10 Assignment of Register Bank Control Fields

Bank Select Control Register		Interrupt Node Priority		Notes
Register Name	**Bitfields**	**Intr. Level**	**Group Levels**	
BNKSEL0 ($EC20_H$/--)	GPRSEL0 … 3	12	0 … 3	Lower group levels
	GPRSEL4 … 7	13	0 … 3	
BNKSEL1 ($EC22_H$/--)	GPRSEL0 … 3	14	0 … 3	
	GPRSEL4 … 7	15	0 … 3	
BNKSEL2 ($EC24_H$/--)	GPRSEL0 … 3	12	4 … 7	Upper group levels
	GPRSEL4 … 7	13	4 … 7	
BNKSEL3 ($EC26_H$/--)	GPRSEL0 … 3	14	4 … 7	
	GPRSEL4 … 7	15	4 … 7	

5.7 Interrupt Node Sharing

Interrupt nodes may be shared among several module requests if either the requests are generated mutually exclusively or the requests are generated at a low rate. If more than one source is enabled in this case, the interrupt handler will first need to determine the requesting source. However, this overhead is not critical for low rate requests.

This node sharing is either controlled via interrupt sub-node control registers (ISNC) which provide separate request flags and enable bits for each supported request source, or via register ISSR, where each bit selects one of two interrupt sources. The interrupt level used for arbitration is determined by the node control register (… IC).

The specific request flags within ISNC registers must be reset by software, contrary to the node request bits which are cleared automatically.

Table 5-11 Sub-Node Control Bit Allocation

Interrupt Node	Interrupt Sources	Control
EOPIC	PEC channels 7 … 0	PECISNC
RTC_IC	RTC: overflow of T14, CNT0 … CNT3	RTC_ISNC
CC2_CC16IC	CAPCOM2 request, ERU request 0	ISSR
CC2_CC17IC	CAPCOM2 request, ERU request 1	ISSR
CC2_CC18IC	CAPCOM2 request, ERU request 2	ISSR
CC2_CC19IC	CAPCOM2 request, ERU request 3	ISSR
CC2_CC20IC	CAPCOM2 request, USIC0 request 6	ISSR
CC2_CC21IC	CAPCOM2 request, USIC0 request 7	ISSR
CC2_CC22IC	CAPCOM2 request, USIC1 request 6	ISSR
CC2_CC23IC	CAPCOM2 request, USIC1 request 7	ISSR
CC2_CC24IC	CAPCOM2 request, ERU request 0	ISSR
CC2_CC25IC	CAPCOM2 request, ERU request 1	ISSR
CC2_CC26IC	CAPCOM2 request, ERU request 2	ISSR
CC2_CC27IC	CAPCOM2 request, ERU request 3	ISSR
CC2_CC28IC	CAPCOM2 request, USIC2 request 6	ISSR
CC2_CC29IC	CAPCOM2 request, USIC2 request 7	ISSR

5.8 External Interrupts

Although the XE16x has no dedicated INTR input pins, it supports many possibilities to react to external asynchronous events. It does this by using a number of IO lines for interrupt input. The interrupt function may be either combined with the pin's main function or used instead of it if the main pin function is not required.

The **External Request Unit** provides flexible trigger signals with selectable qualifiers, which can directly control peripherals (ADC, MultiCAN) or generate additional interrupt/PEC requests from external input signals.

Table 5-12 Pins Usable as External Interrupt Inputs

Port Pin	Original Function	Control Register
P4.7-0/CC31-24IO	CAPCOM Register 31-24 Capture Input	CC31-CC24
P2.10-3/CC23-16IO	CAPCOM Register 23-16 Capture Input[1)]	CC23-CC16
P4.2/T2IN	Auxiliary timer T2 input pin	T2CON
P4.6/T4IN	Auxiliary timer T4 input pin	T4CON
P2.10/CAPIN	GPT2 capture input pin[1)]	T5CON

1) Pin P2.10 overlays two possible input functions.

For each of these pins, either a positive, a negative, or both a positive and a negative external transition can be selected to cause an interrupt or PEC service request. The edge selection is performed in the control register of the peripheral device associated with the respective port pin (separate control for ERU inputs). The peripheral must be programmed to a specific operating mode to allow generation of an interrupt by the external signal. The priority of the interrupt request is determined by the interrupt control register of the respective peripheral interrupt source, and the interrupt vector of this source will be used to service the external interrupt request.

Note: In order to use any of the listed pins as an external interrupt input, it must be switched to input mode via its port control register.

When port pins CCxIO are to be used as external interrupt input pins, bitfield CCMODx in the control register of the corresponding capture/compare register CCx must select capture mode. When CCMODx is programmed to 001_B, the interrupt request flag CCxIR in register CCxIC will be set on a positive external transition at pin CCxIO. When CCMODx is programmed to 010_B, a negative external transition will set the interrupt request flag. When CCMODx = 011_B, both a positive and a negative transition will set the request flag. In all three cases, the contents of the allocated CAPCOM timer will be latched into capture register CCx, independent of whether or not the timer is running. When the interrupt enable bit CCxIE is set, a PEC request or an interrupt request for vector CCxINT will be generated.

Pins T2IN or T4IN can be used as external interrupt input pins when the associated auxiliary timer T2 or T4 in block GPT1 is configured for capture mode. This mode is selected by programming the mode control fields T2M or T4M in control registers T2CON or T4CON to 101_B. The active edge of the external input signal is determined by bitfields T2I or T4I. When these fields are programmed to $X01_B$, interrupt request flags T2IR or T4IR in registers T2IC or T4IC will be set on a positive external transition at pins T2IN or T4IN, respectively. When T2I or T4I is programmed to $X10_B$, then a negative external transition will set the corresponding request flag. When T2I or T4I is programmed to $X11_B$, both a positive and a negative transition will set the request flag. In all three cases, the contents of the core timer T3 will be captured into the auxiliary timer registers T2 or T4 based on the transition at pins T2IN or T4IN. When the interrupt enable bits T2IE or T4IE are set, a PEC request or an interrupt request for vector T2INT or T4INT will be generated.

Pin CAPIN differs slightly from the timer input pins as it can be used as external interrupt input pin without affecting peripheral functions. When the capture mode enable bit T5SC in register T5CON is cleared to ‘0’, signal transitions on pin CAPIN will only set the interrupt request flag CRIR in register CRIC, and the capture function of register CAPREL is not activated.

So register CAPREL can still be used as reload register for GPT2 timer T5, while pin CAPIN serves as external interrupt input. Bitfield CI in register T5CON selects the effective transition of the external interrupt input signal. When CI is programmed to 01_B, a positive external transition will set the interrupt request flag. CI = 10_B selects a negative transition to set the interrupt request flag, and with CI = 11_B, both a positive and a negative transition will set the request flag. When the interrupt enable bit CRIE is set, an interrupt request for vector CRINT or a PEC request will be generated.

5.9 OCDS Requests

The OCDS module issues high-priority break requests or standard service requests. The break requests are routed directly to the CPU (like the hardware trap requests) and are prioritized there. Therefore, break requests ignore the standard interrupt arbitration and receive highest priority.

The standard OCDS service requests are routed to the CPU Action Control Unit together with the arbitrated interrupt/PEC requests. The service request with the higher priority is sent to the CPU to be serviced. If both the interrupt/PEC request and the OCDS request have the same priority level, the interrupt/PEC request wins.

This approach ensures precise break control, while affecting the system behavior as little as possible.

The CPU Action Control Unit also routes back request acknowledges and denials from the core to the corresponding requestor.

5.10 Service Request Latency

The numerous service requests of the XE16x (requests for interrupt or PEC service) are generated asynchronously with respect to the execution of the instruction flow. Therefore, these requests are arbitrated and are inserted into the current instruction stream. This decouples the service request handling from the currently executed instruction stream, but also leads to a certain latency.

The request latency is the time from activating a request signal at the interrupt controller (ITC) until the corresponding instruction reaches the pipeline's execution stage. **Table 5-13** lists the consecutive steps required for this process.

Table 5-13 Steps Contributing to Service Request Latency

Description of Step	Interrupt Response	PEC Response
Request arbitration in 3 stages, leads to acceptance by the CPU (see **Section 5.2**)	3 cycles	3 cycles
Injection of an internal instruction into the pipeline's instruction stream	4 cycles	4 cycles
The first instruction fetched from the interrupt vector table reaches the pipeline's execution stage	4 cycles / 0[1)]	- - -
Resulting minimum request latency	11/7 cycles	7 cycles

1) Can be saved by using the interrupt jump table cache (see **Section 5.3**).

Sources for Additional Delays

Because the service requests are inserted into the current instruction stream, the properties of this instruction stream can influence the request latency.

Table 5-14 Additional Delays Caused by System Logic

Reason for Delay	Interrupt Response	PEC Response
Interrupt controller busy, because the previous interrupt request is still in process	max. 7 cycles	max. 7 cycles
Pipeline is stalled, because instructions preceding the injected instruction in the pipeline need to write/read data to/from a peripheral or memory	$2 \times T_{ACCmax}$ 1)	$2 \times T_{ACCmax}$
Pipeline cancelled, because instructions preceding the injected instruction in the pipeline update core SFRs	4 cycles	4 cycles
Memory access for stack writes (if not to DPRAM or DSRAM)	$2/3 \times T_{ACC}$ 2)	- - -
Memory access for vector table read (except for intr. jump table cache)	$2 \times T_{ACC}$	- - -

1) This is the longest possible access time within the XE16x system.

2) Depending on segmentation off/on.

The actual response to an interrupt request may be delayed further depending on programming techniques used by the application. The following factors can contribute:

- Actual interrupt service routine is only reached via a JUMP from the interrupt vector table.
  Time-critical instructions can be placed directly into the interrupt vector table, followed by a branch to the remaining part of the interrupt service routine. The space between two adjacent vectors can be selected via bitfield VECSC in register CPUCON1.
- Context switching is executed before the intended action takes place (see **Section 5.6**)
  Time-critical instructions can be programmed “non-destructive” and can be executed before switching context for the remaining part of the interrupt service routine.

5.11 Trap Functions

Traps interrupt current execution in a manner similar to standard interrupts. However, trap functions offer the possibility to bypass the interrupt system's prioritization process for cases in which immediate system reaction is required. Trap functions are not maskable and always have priority over interrupt requests on any priority level.

The XE16x provides two different kinds of trapping mechanisms: **Hardware Traps** are triggered by events that occur during program execution (such as illegal access or undefined opcode); **Software Traps** are initiated via an instruction within the current execution flow.

Software Traps

The TRAP instruction causes a software call to an interrupt service routine. The vector number specified in the operand field of the trap instruction determines which vector location in the vector table will be branched to.

Executing a TRAP instruction causes an effect similar to the occurrence of an interrupt at the same vector. PSW, CSP (in segmentation mode), and IP are pushed on the internal system stack and a jump is taken to the specified vector location. When a trap is executed, the CSP for the trap service routine is loaded from register VECSEG. No Interrupt Request flags are affected by the TRAP instruction. The interrupt service routine called by a TRAP instruction must be terminated with a RETI (return from interrupt) instruction to ensure correct operation.

Note: The CPU priority level and the selected register bank in register PSW are not modified by the TRAP instruction, so the service routine is executed on the same priority level from which it was invoked. Therefore, the service routine entered by the TRAP instruction uses the original register bank and can be interrupted by other traps or higher priority interrupts, other than when triggered by a hardware event.

Hardware Traps

Hardware traps are issued by faults or specific system states which occur during runtime of a program (not identified at assembly time). A hardware trap may also be triggered intentionally, for example: to emulate additional instructions by generating an Illegal Opcode trap. The XE16x distinguishes nine different hardware trap functions. When a hardware trap condition has been detected, the CPU branches to the trap vector location for the respective trap condition. The instruction which caused the trap is completed before the trap handling routine is entered.

Hardware traps are non-maskable and always have priority over every other CPU activity. If several hardware trap conditions are detected within the same instruction cycle, the highest priority trap is serviced (see **Table 5-3**).

PSW, CSP (in segmentation mode), and IP are pushed on the internal system stack and the CPU level in register PSW is set to the highest possible priority level (level 15), disabling all interrupts. The global register bank is selected. Execution branches to the respective trap vector in the vector table. A trap service routine must be terminated with the RETI instruction.

The nine hardware trap functions of the XE16x are divided into two classes:

Class A traps are:

- System Request 0 (SR0)
- Stack Overflow
- Stack Underflow trap
- Software Break

These traps share the same trap priority, but have individual vector addresses.

Class B traps are:

- System Request 1 (SR1)
- Undefined Opcode
- Memory Access Error
- Protection Fault
- Illegal Word Operand Access

The Class B traps share the same trap priority and the same vector address.

The bit-addressable Trap Flag Register (TFR) allows a trap service routine to identify the kind of trap which caused the exception. Each trap function is indicated by a separate request flag. When a hardware trap occurs, the corresponding request flag in register TFR is set to ‘1’.

The reset functions may be regarded as a type of trap. Reset functions have the highest system priority (trap priority III).

Class A traps have the second highest priority (trap priority II), on the 3rd rank are Class B traps, so a Class A trap can interrupt a Class B trap. If more than one Class A trap occur at a time, they are prioritized internally, with the SR0 trap at the highest and the software break trap at the lowest priority.

In the case where e.g. an Undefined Opcode trap (Class B) occurs simultaneously with an SR0 trap (Class A), both the SR0 and the UNDOPC flag is set, the IP of the instruction with the undefined opcode is pushed onto the system stack, but the SR0 trap is executed. After return from the SR0 service routine, the IP is popped from the stack and immediately pushed again because of the pending UNDOPC trap.

Note: The trap service routine must clear the respective trap flag; otherwise, a new trap will be requested after exiting the service routine. Setting a trap request flag by software causes the same effects as if it had been set by hardware.

TFR
Trap Flag Register **SFR ($FFAC_H/D6_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SR0	STKOF	STKUF	SOFTBRK	SR1	-	-	-	UNDOPC	-	-	ACER	PRTFLT	ILLOPA	-	-
rwh	rwh	rwh	rwh	rwh	-	-	-	rwh	-	-	rwh	rwh	rwh	-	-

Field	Bits	Type	Description
SR0	15	rwh	**System Request 0 Flag** 0_B No trigger detected 1_B The selected condition has been detected
STKOF	14	rwh	**Stack Overflow Flag** 0_B No stack overflow event detected 1_B The current stack pointer value falls below the contents of register STKOV
STKUF	13	rwh	**Stack Underflow Flag** 0_B No stack underflow event detected 1_B The current stack pointer value exceeds the contents of register STKUN
SOFTBRK	12	rwh	**Software Break** 0_B No software break event detected 1_B Software break event detected
SR1	11	rwh	**System Request 1 Flag** 0_B No trigger detected 1_B The selected condition has been detected
UNDOPC	7	rwh	**Undefined Opcode** 0_B No undefined opcode event detected 1_B The currently decoded instruction has no valid XE16x opcode
ACER	4	rwh	**Memory Access Error** 0_B No access error event detected 1_B Illegal or erroneous access detected
PRTFLT	3	rwh	**Protection Fault** 0_B No protection fault event detected 1_B A protected instruction with an illegal format has been detected

Field	Bits	Type	Description
ILLOPA	2	rwh	**Illegal Word Operand Access** 0_B No illegal word operand access event detected 1_B A word operand access (read or write) to an odd address has been attempted

Class A Traps

Class A traps are generated by the high priority system request SR0 or by special CPU events such as the software break, a stack overflow, or an underflow event. Class A traps are not used to indicate hardware failures. After a Class A event, a dedicated service routine is called to react on the events. Each Class A trap has its own vector location in the vector table. Class A traps cannot interrupt atomic/extend sequences and I/O accesses in progress, because after finishing the service routine, the instruction flow must be further correctly executed. For example, an interrupted extend sequence cannot be restarted. All Class A traps are generated in the pipeline during the execution of instructions, except for SR0, which is an asynchronous external event. Class A trap events can be generated only during the memory stage of execution, so traps cannot be generated by two different instructions in the pipeline in the same CPU cycle. The execution of instructions which caused a Class A trap event is always completed. In the case of an atomic/extend sequence or I/O read access in progress, the complete sequence is executed. Upon completion of the instruction or sequence, the pipeline is canceled and the IP of the instruction following the last one executed is pushed on the stack. Therefore, in the case of a Class A trap, the stack always contains the IP of the first not-executed instruction in the instruction flow.

Note: The Branch Folding Unit allows the execution of a branch instruction in parallel with the preceding instruction. The pre-processed branch instruction is combined with the preceding instruction. The branch is executed together with the instruction which caused the Class A trap. The IP of the first following not-executed instruction in the instruction flow is then pushed on the stack.

If more than one Class A trap occur at the same time, they are prioritized internally. The SR0 trap has the highest priority and the software break has the lowest.

Note: In the case of two different Class A traps occurring simultaneously, both trap flags are set. The IP of the instruction following the last one executed is pushed on the stack. The trap with the higher priority is executed. After return from the service routine, the IP is popped from the stack and immediately pushed again because of the other pending Class A trap (unless the trap related to the second trap flag in TFR has been cleared by the first trap service routine).

Class B Traps

Class B traps are generated by unrecoverable hardware failures. In the case of a hardware failure, the CPU must immediately start a failure service routine. Class B traps can interrupt an atomic/extend sequence and an I/O read access. After finishing the Class B service routine, a restoration of the interrupted instruction flow is not possible.

All Class B traps have the same priority (trap priority I). When several Class B traps become active at the same time, the corresponding flags in the TFR register are set and the trap service routine is entered. Because all Class B traps have the same vector, the priority of service of simultaneously occurring Class B traps is determined by software in the trap service routine.

The access error (ACER) and system request 1 (SR1) are asynchronous external (to the CPU) events, while all other Class B traps are generated in the pipeline during the execution of instructions. Class B trap events can be generated only during the memory stage of execution, so traps cannot be generated by two different instructions in the pipeline in the same CPU cycle. Instructions which caused a Class B trap event are always executed, then the pipeline is canceled and the IP of the instruction following the one which caused the trap is pushed on the stack. Therefore, the stack always contains the IP of the first following not-executed instruction in the instruction flow.

Note: The Branch Folding Unit allows the execution of a branch instruction in parallel with the preceding instruction. The pre-processed branch instruction is combined with the preceding instruction. The branch is executed together with the instruction causing the Class B trap. The IP of the first following not-executed instruction in the instruction flow is pushed on the stack.

A Class A trap occurring during the execution of a Class B trap service routine will be serviced immediately. During the execution of a Class A trap service routine, however, any Class B trap occurring will not be serviced until the Class A trap service routine is exited with a RETI instruction. In this case, the occurrence of the Class B trap condition is stored in the TFR register, but the IP value of the instruction which caused this trap is lost.

Note: If a Class A trap occurs simultaneously with a Class B trap, both trap flags are set. The IP of the instruction following the one which caused the trap is pushed into the stack, and the Class A trap is executed. If this occurs during execution of an atomic/extend sequence or I/O read access in progress, then the presence of the Class B trap breaks the protection of atomic/extend operations and the Class A trap will be executed immediately without waiting for the sequence completion. After return from the service routine, the IP is popped from the system stack and immediately pushed again because of the other pending Class B trap. In this situation, the restoration of the interrupted instruction flow is not possible.

System Request 0 Trap (A)

Whenever a high-to-low transition on the respective CPU-input is detected (i.e. the defined condition has become true), the SR0 flag in register TFR is set and the CPU will enter the SR0 trap routine.

Stack Overflow Trap (A)

Whenever the stack pointer is implicitly decremented and the stack pointer is equal to the value in the stack overflow register STKOV, the STKOF flag in register TFR is set and the CPU will enter the stack overflow trap routine.

For recovery from stack overflow, it must be ensured that there is enough excess space on the stack to save the current system state twice (PSW, IP, in segmented mode also CSP). Otherwise, a system reset should be generated.

Stack Underflow Trap (A)

Whenever the stack pointer is implicitly incremented and the stack pointer is equal to the value in the stack underflow register STKUN, the STKUF flag is set in register TFR and the CPU will enter the stack underflow trap routine.

Software Break Trap (A)

When the instruction currently being executed by the CPU is a SBRK instruction, the SOFTBRK flag is set in register TFR and the CPU enters the software break debug routine. The flag generation of the software break instruction can be disabled by the On-chip Emulation Module. In this case, the instruction only breaks the instruction flow and signals this event to the debugger, the flag is not set and the trap will not be executed.

System Request 1 Trap (B)

Whenever a high-to-low transition on the respective CPU-input is detected (i.e. the defined condition has become true), the SR1 flag in register TFR is set and the CPU will enter the SR1 trap routine.

Undefined Opcode Trap (B)

When the instruction currently decoded by the CPU does not contain a valid XE16x opcode, the UNDOPC flag is set in register TFR and the CPU enters the undefined opcode trap routine. The instruction that causes the undefined opcode trap is executed as a NOP.

This can be used to emulate unimplemented instructions. The trap service routine can examine the faulting instruction to decode operands for unimplemented opcodes based on the stacked IP. In order to resume processing, the stacked IP value must be incremented by the size of the undefined instruction, which is determined by the user, before a RETI instruction is executed.

Memory Access Error (B)

When a memory access error is detected, the ACER flag is set in register TFR and the CPU enters the access error trap routine. The access error is reported in the following cases:

- access to Flash memory while it is disabled
- access to Flash memory from outside while read-protection is active
- double bit error detected when reading Flash memory
- access to reserved locations (see memory map in **Table 3-1**)
- parity error during an access to RAM

In case of an access error, additionally the soft-trap code $1E9B_H$ is issued.

Protection Fault Trap (B)

Whenever one of the special protected instructions is executed where the opcode of that instruction is not repeated twice in the second word of the instruction and the byte following the opcode is not the complement of the opcode, the PRTFLT flag in register TFR is set and the CPU enters the protection fault trap routine. The protected instructions include DISWDT, EINIT, IDLE, PWRDN, SRST, ENWDT and SRVWDT. The instruction that causes the protection fault trap is executed like a NOP.

Illegal Word Operand Access Trap (B)

Whenever a word operand read or write access is attempted to an odd byte address, the ILLOPA flag in register TFR is set and the CPU enters the illegal word operand access trap routine.

6 System Control Unit (SCU)

The System Control Unit (SCU) of the XE16x handles all system control tasks besides the debug related tasks which are controlled by the OCDS/Cerberus. All functions described in this chapter are tightly coupled, thus, they are conveniently handled by one unit, the SCU.

The SCU contains the following functional sub-blocks:

- Clock Generation (see **Chapter 6.1**)
- Wake-up Timer (see **Chapter 6.2**)
- Reset Operation (see **Chapter 6.3**)
- External Service Requests (see **Chapter 6.4**)
- Power Supply and Control (see **Chapter 6.5**)
- Global State Control (see **Chapter 6.6**)
- Software Boot Support (see **Chapter 6.7**)
- External Request Unit (see **Chapter 6.8**)
- Interrupt Generation (see **Chapter 6.9**)
- Temperature Compensation (see **Chapter 6.10**)
- Watchdog Timer (see **Chapter 6.11**)
- Trap Generation (see **Chapter 6.12**)
- Memory Content Protection (see **Chapter 6.13**)
- Register Access Control (see **Chapter 6.14**)
- Miscellaneous System Registers (see **Chapter 6.15**)
- SCU Registers and Address map (see **Chapter 6.16**)

Important Information: Register Programming

The System Control Unit contains special function registers, which can not be programmed in an arbitrary order in particular due to the usage of an internal voltage regulator. In order to prevent critical system conditions because of an improper setup and to provide means for easy and quick configuration and control of sensitve features such as power supply and clock generation, recommendations and examples for the programming sequence of the registers will be given in the Programmer's Guide.

In particular the registers listed below have to be updated with care:

- Clock Generation Unit: WUOSCCON, HPOSCCON, PLLOSCCON, PLLCONx
- Power Supply: EVR1CON0, EVR1SET15VHP,
  EVRMCON0,EVRMSET15VHP,
  PVC1CON0, PVCMCON0,
  SWDCON0
- System: SYSCON0

6.1 Clock Generation Unit

The Clock Generation Unit (CGU) allows a very flexible clock generation for the XE16x. During user program execution the frequency can be programmed for an optimal ratio between performance and power consumption in the actual application state.

6.1.1 Overview

The CGU can convert a low-frequency external clock to a high-speed system clock or can create a high-speed system clock without external input.

The CGU consists of a Clock Generator and a Clock Control Unit (CCU).

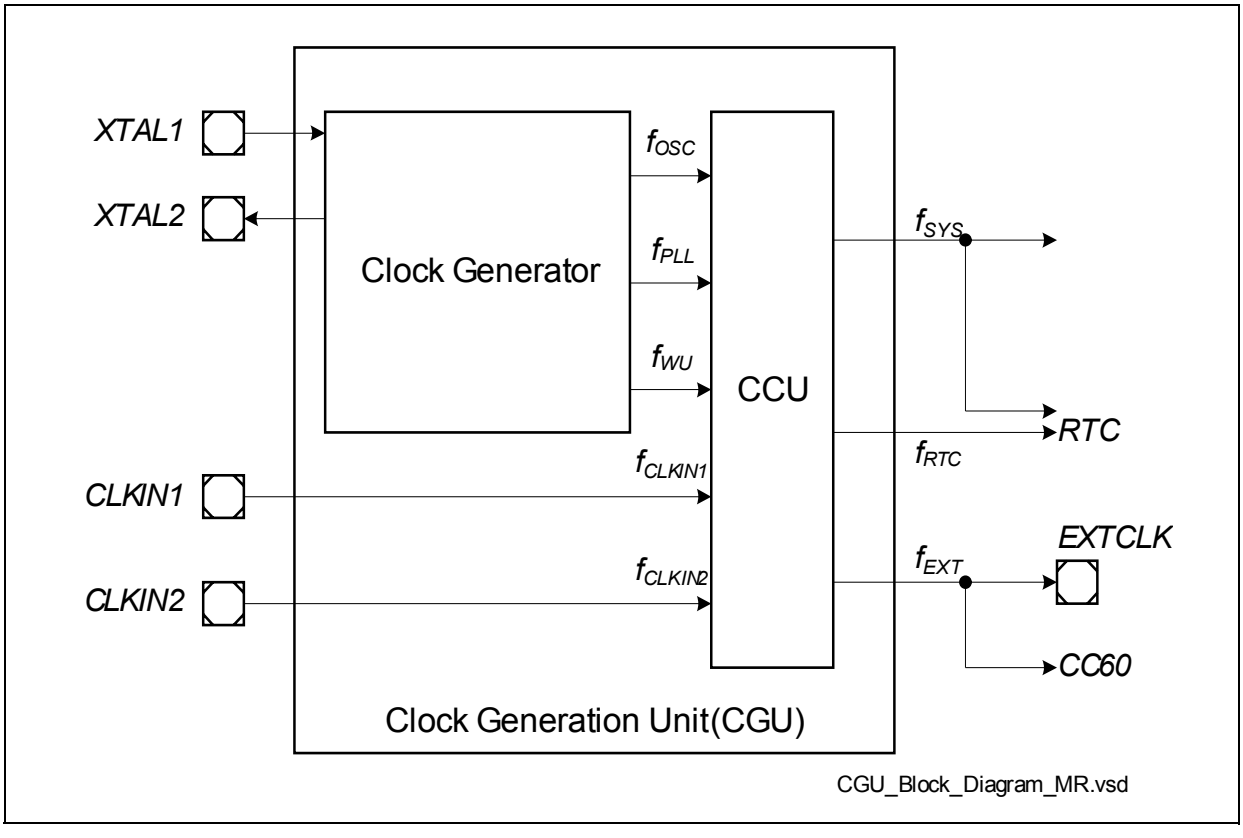


Figure 6-1 Clock Generation Unit Block Diagram

The input connections of the CGU are described in **Chapter 6.17.1**.

The following clock signals are generated:

- System clock f_{SYS}
- RTC count clock f_{RTC} (
- Wake-Up Timer (WUT) clock f_{WUT}
- External clock f_{EXT}

Chapter 6.1.5 and **Chapter 6.1.6** describe which clock signals is generated out of which selectable clocks.

Register Overview

The CGU is controlled by a number of registers shown in the following figure.

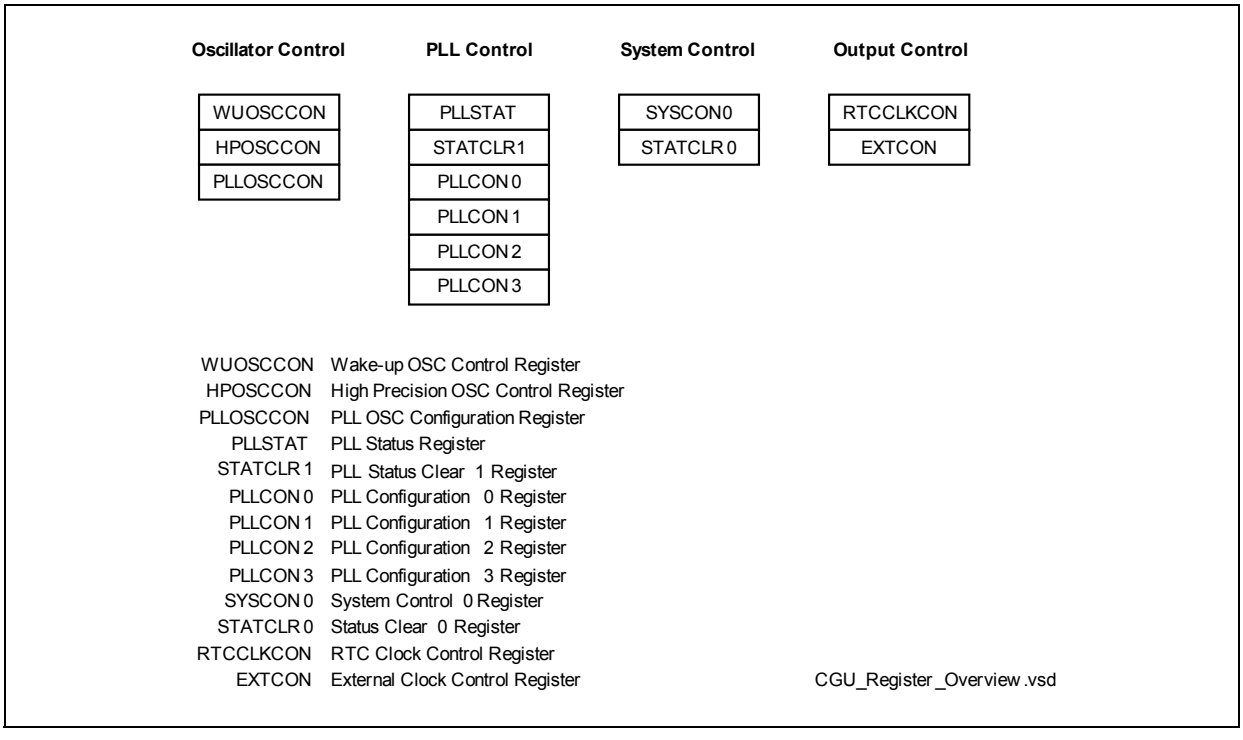


Figure 6-2 Clock Generation Unit Register Overview

The following sections describe the different parts of the CGU.

6.1.2 Trimmed Current Controlled Wake-up Clock (OSC_WU)

The trimmed current controlled wake-up clock source provides a clock to control internal operations independent of the standard clock supplies and requires no external components. Its output frequency f_{WU} is configured via bit field WUOSCCON.FREQSEL and has a typical range from 130 kHz to 500 kHz.

6.1.3 High Precision Oscillator Circuit (OSC_HP)

The high precision oscillator circuit can drive an external crystal or accepts an external clock source. It consists of an inverting amplifier with XTAL1 as input, and XTAL2 as output.

Figure 6-4 and **Figure 6-3** show the recommended external circuitries for both operating modes, External Crystal Mode and External Input Clock Mode.

6.1.3.1 External Input Clock Mode

An external clock signal is supplied directly not using an external crystal and bypassing the amplifier of the oscillator. The maximum allowed input frequency depends on the characteristics of pin XTAL1.

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

Note: Voltages on XTAL1 must comply to the voltage defined in the data sheet.

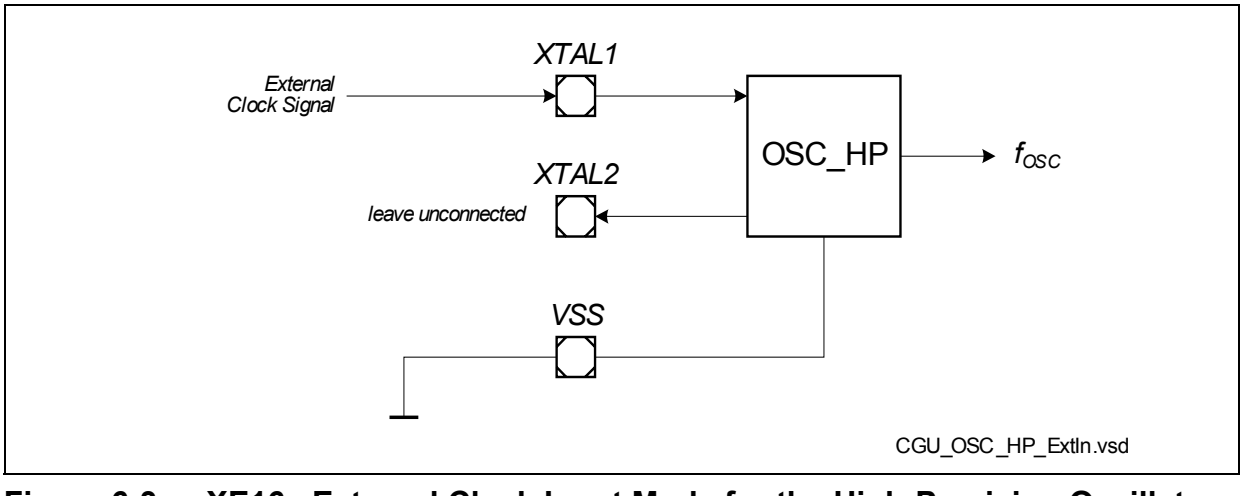


Figure 6-3 XE16x External Clock Input Mode for the High-Precision Oscillator

6.1.3.2 External Crystal Mode

An external oscillator load circuitry must be used, connected to both pins, XTAL1 and XTAL2. It consists normally of the two load capacitances C1 and C2. For some crystals a series damping resistor might be necessary. The exact values and related operating

range depend on the crystal and have to be determined and optimized together with the crystal vendor using the negative resistance method.

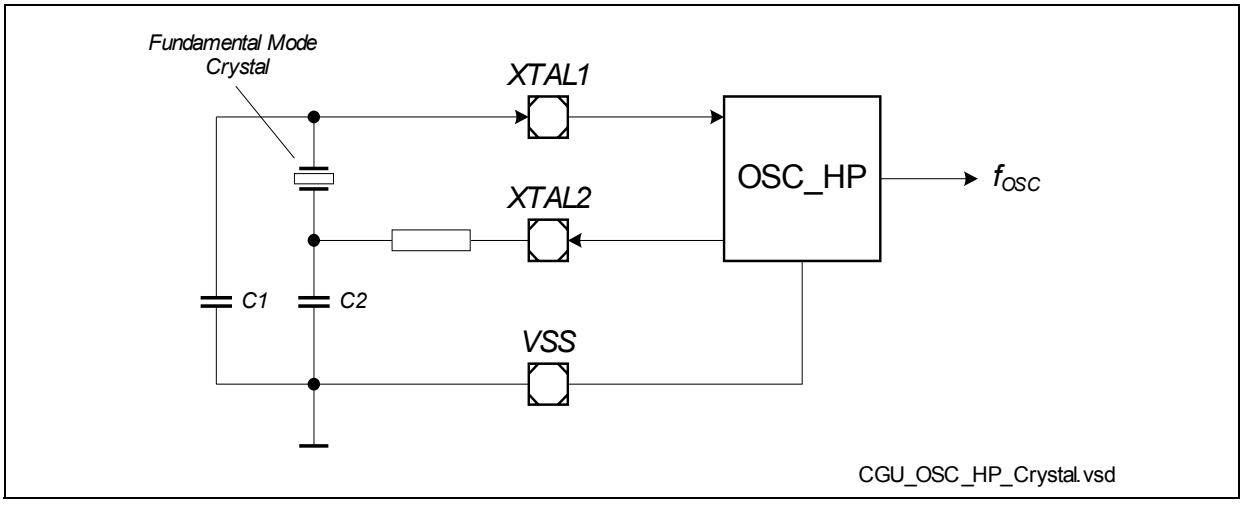


Figure 6-4 XE16x External Crystal Mode Circuitry for the High-Precision Oscillator

6.1.4 Phase-Locked Loop (PLL) Module

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

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

6.1.4.1 Features

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

- VCO lock detection
- 4-bit input divider **P**: (divide by PDIV+1)
- 6-bit feedback divider **N**: (multiply by NDIV+1)
- 10-bit output divider **K2**: (divide K2DIV+1)
- 10-bit VCO bypass divider **K1**: (divide by either by K1DIV+1)
- Oscillator run detection and Watchdog
- Different operating modes
  - Prescaler Mode
  - Unlocked Mode
  - Normal Mode
- Different power saving modes
  - Power Down

– Sleep Mode (VCO Power Down)
- Glitchless programming of output divider K2 and VCO bypass divider K1
- Glitchless switching between Normal Mode and Prescaler Mode
- Trimmed current controlled clock source

6.1.4.2 PLL Functional Description

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

The following figure shows the PLL block structure.

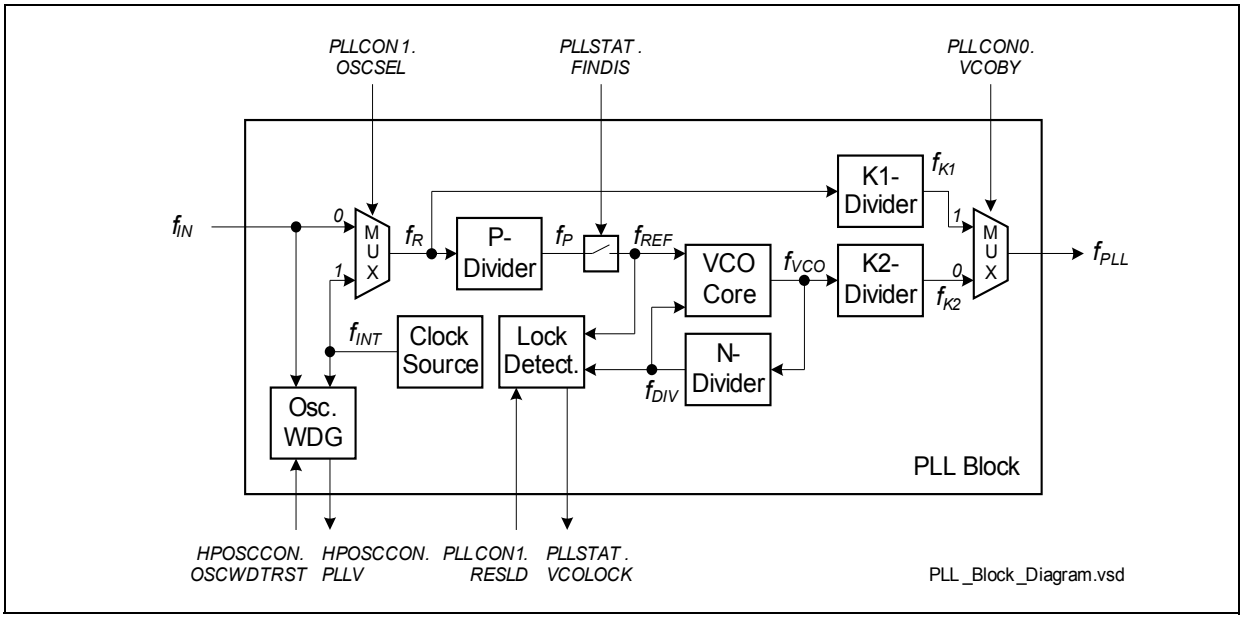


Figure 6-5 PLL Block Diagram

Clock Source Control

The reference frequency f_R can be selected to be either taken from the trimmed current controlled clock source f_{INT} or from an external clock source f_{IN}.

PLL Modes

The PLL clock f_{PLL} is generated from f_R in one of the following software selectable modes:

- Normal Mode
- Prescaler Mode
- Unlocked Mode

In Normal Mode the reference frequency f_R is divided by a factor P, multiplied by a factor N and then divided by a factor K2. The output frequency is given by

(6.1)

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

In Prescaler Mode the reference frequency f_R is divided by a factor K1. The output frequency is given by

(6.2)

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

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

(6.3)

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

PLL Power Saving Modes

PLL Power Down Mode The PLL offers a Power Down Mode to save power if the PLL is not needed at all. While the PLL is in Power Down Mode no PLL output frequency is generated.

PLL Sleep Mode The PLL offers a Sleep Mode (also called VCO Power Down Mode) to save power within the PLL. While the PLL is in Sleep Mode only the Prescaler Mode can be used.

6.1.4.3 Configuration and Operation of the PLL Modes

The following section describes the configuration and the operation of the different PLL modes. Further information can be found in the Programmer's Guide.

Configuration and Operation of the Unlocked Mode

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

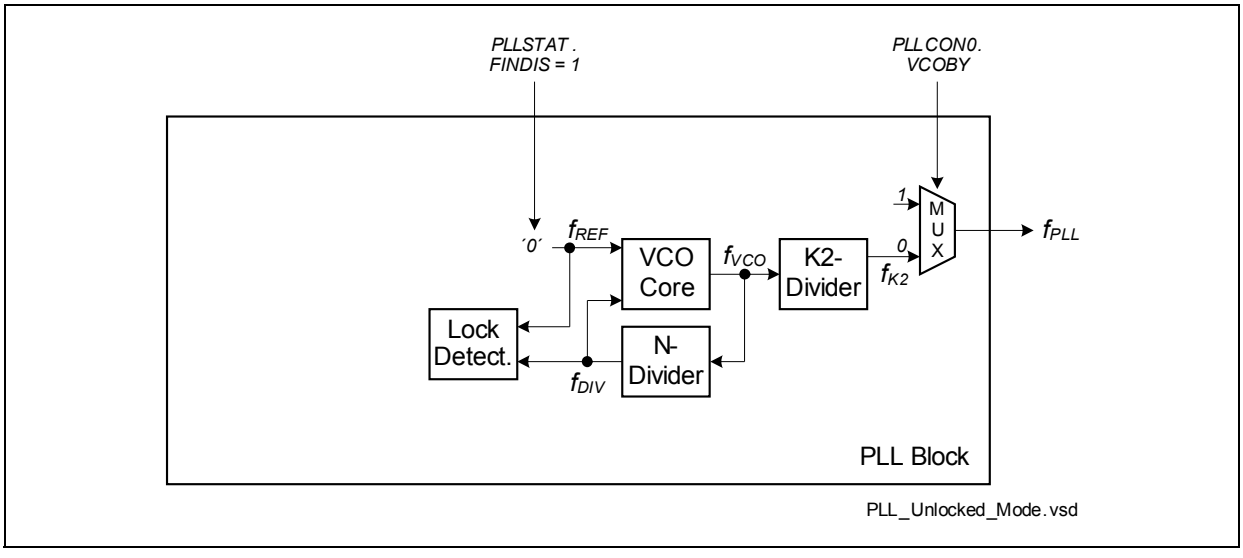


Figure 6-6 PLL Unlocked Mode Diagram

The Unlocked Mode is selected by the following settings:

- STATCLR1.SETFINDIS = 1
- PLLCON0.VCOBY = 0

The Unlocked Mode is entered when all following conditions are true:

- PLLSTAT.FINDIS = 1
- PLLSTAT.VCOBYST = 1

Operation in Unlocked Mode does not require an input clock f_{IN}. The Unlocked Mode is automatically entered on a PLL VCO Loss-of-Lock event if bit PLLCON1.EMFINDISEN is cleared. This mechanism allows a fail-safe operation of the PLL as in emergency cases still a clock is available.

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

Note: Changing the system operation frequency by changing the value of the K2-Divider or the VCO range has a direct influence on the power consumption of the device. Therefore, this has to be done carefully.

Configuration and Operation of the Normal Mode

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

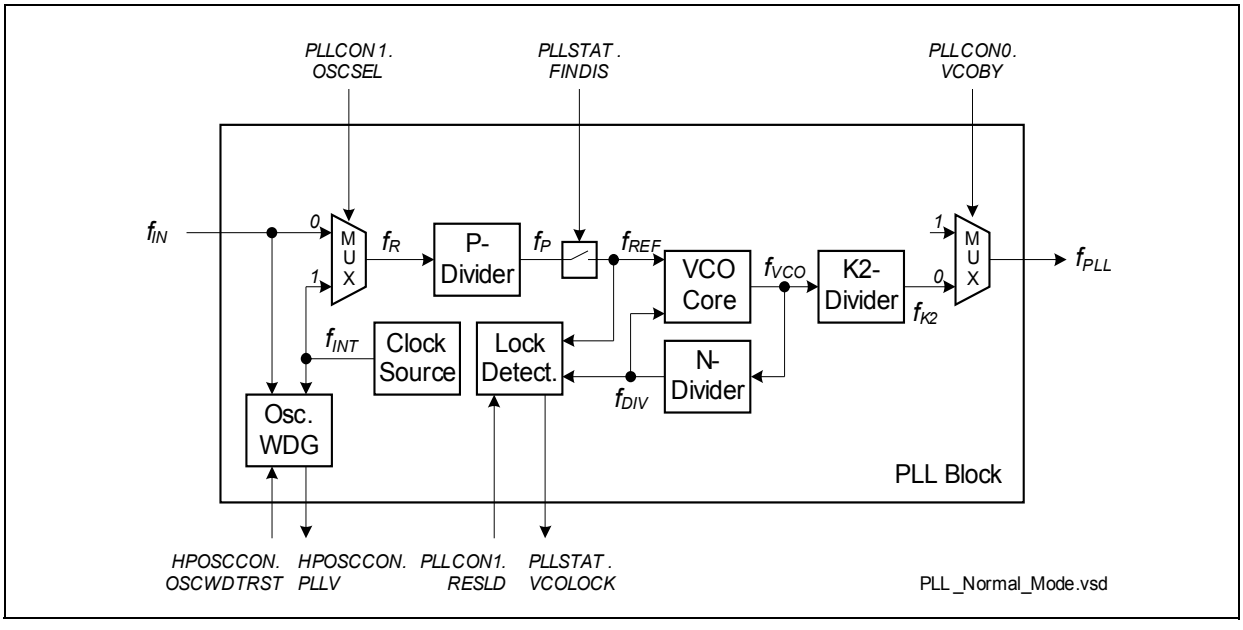


Figure 6-7 PLL Normal Mode Diagram

The Normal Mode is selected by the following settings:

- PLLCON0.VCOBY = 0
- STATCLR1.CLRFINDIS = 1

The Normal Mode is entered when all following conditions are true:

- PLLSTAT.FINDIS = 0
- PLLSTAT.VCOBYST = 1
- PLLSTAT.VCOLOCK = 1
- HPOSCCON.PLLV = 1

Operation in Normal Mode requires a clock frequency of f_R. When f_{IN} is selected as source for f_R it is recommended to check and monitor if an input frequency f_R is available at all by checking HPOSCCON.PLLV.

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

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

To modify or enter the Normal Mode frequency, follow the sequence described below:

Configure and enter Prescaler Mode. For more details see the Prescaler Mode.

Disable the trap generation for the VCO Lost-of-Lock.

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

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

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

Note: It is recommended to reset the VCO Lock detection (PLLCON1.RESLD = 1) after the new values of the dividers have been configured to get a defined VCO lock check time.

When this happens the switch from Prescaler Mode to Normal Mode can be done. Normal Mode is requested by clearing PLLCON0.VCOBY. The Normal Mode is entered when the status bit PLLSTAT.VCOBYST is set.

Now the Normal Mode is entered. The trap status flag for the VCO Lock trap should be cleared and then enabled again.

The intended PLL output target frequency can be configured by changing only the K2-Divider. Depending on the selected divider value of the K2-Divider, the duty cycle of the clock is selected. This can have an impact on the operation with an external communication interface. In order to avoid too big frequency changes it might be neccessary to change the K2-Divider in multiple steps. When the value of the K2-Diver was changed the next update of this value should not be done before bit PLLSTAT.K2RDY is set.

Note: The Programmers's Guide describes a smooth frequency stepping to achieve an appropriate load regulation of the internal voltage regulator.

PLL VCO Lock Detection

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

PLL VCO Loss-of-Lock Event

The PLL may become unlocked, caused by a break of the crystal or the external clock line. In such a case, a trap is generated if the according trap is enabled. Additionally, the clock f_R is disconnected from the PLL VCO to avoid unstable operation due to noise or

sporadic clock pulses coming from the oscillator circuit. Without a clock input f_R, the PLL gradually slows down to its VCO base frequency and remains there. The automatic disconnection of the VCO from its input clock f_R in case of a VCO Loss-of-Lock event can be enabled by setting bit PLLCON1.EMFINDISEN. If this bit is cleared the clock f_R remains connected to the VCO.

Configuration and Operation of the Prescaler Mode

In Prescaler Mode, the PLL is running at frequency f_{PLL}, where f_R is divided by the K1-Divider.

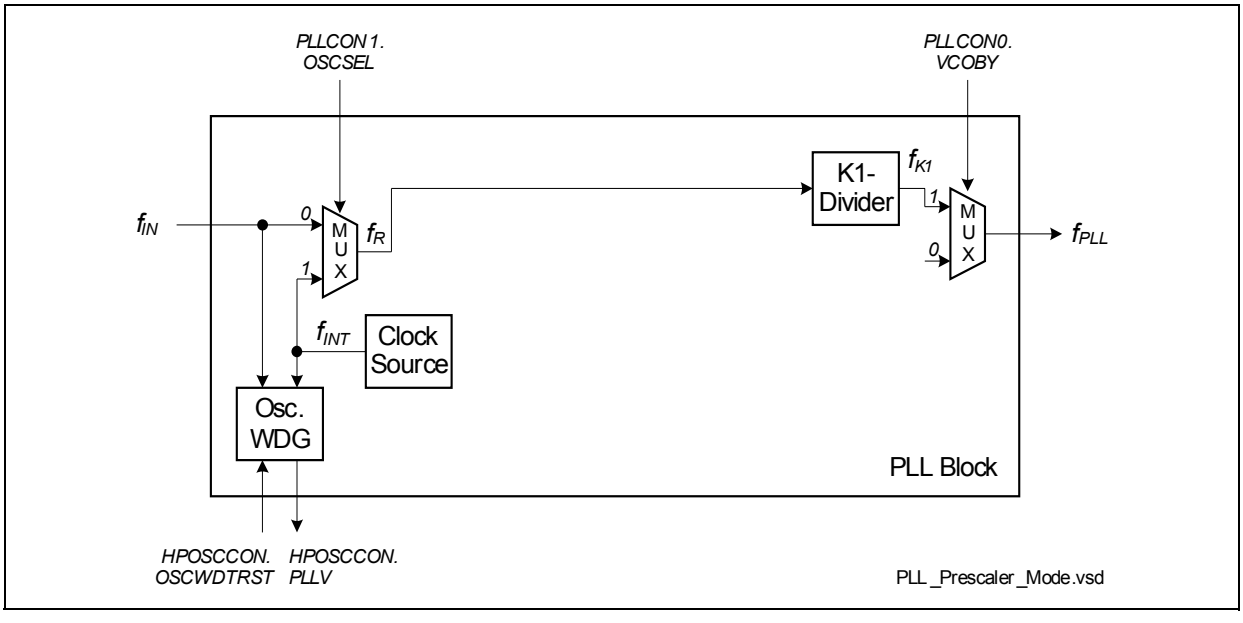


Figure 6-8 PLL Prescaler Mode Diagram

The Prescaler Mode is selected by the following setting:

- PLLCON0.VCOBY = 1

The Prescaler Mode is entered when all following conditions are true:

- PLLSTAT.VCOBYST = 0
- HPOSCCON.PLLV = 1

Operation in Prescaler Mode requires an input clock frequency f_R. If f_{IN} is selected as clock source for f_R it is recommended to check and monitor if an input frequency f_{OSC} is available at all by checking HPOSCCON.PLLV. There are no requirements regarding the frequency of f_R.

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

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

The duty cycle of the clock signal depends on the selected value of the K1-Divider. This can have an impact for the operation with an external communication interface.

The Prescaler Mode is requested from the Unlocked or Normal Mode by setting bit PLLCON0.VCOBY. The Prescaler Mode is entered when the status bit PLLSTAT.VCOBYST is cleared.

Before the Prescaler Mode is requested the K1-Divider should be configured with a value generating a PLL output frequency f_{PLL} that matches the one generated by the Unlocked or Normal Mode as much as possible. In this way the frequency change resulting out of the mode change is reduced to a minimum.

The Prescaler Mode is requested to be left by clearing bit PLLCON0.VCOBY. The Prescaler Mode is left when the status bit PLLSTAT.VCOBYST is set.

Configuration and Operation of the PLL Power Down Mode

The Power Down Mode is entered by setting bit PLLCON0.PLLPWD. While the PLL is in Power Down Mode no PLL output frequency is generated.

Configuration and Operation of the PLL Sleep Mode

The Sleep Mode (also called VCO Power Down Mode) is entered by setting bit PLLCON0.VCOPWD. While the PLL is in Sleep Mode only the Prescaler Mode is operable. Selecting the Sleep Mode does not automatically switch to the Prescaler Mode. Therefore, before the Sleep Mode is entered the Prescaler Mode must be active.

6.1.4.4 Trimmed Current Controlled Clock

The trimmed current controlled clock source can provide a clock f_{INT} for the PLL. This is configured via bit PLLCON1.OSCSEL.

Note: The clock f_{INT} is also required for the operation of the oscillator watchdog.

6.1.4.5 Oscillator Watchdog

The oscillator watchdog continuously monitors the input clock f_{IN}. If the input frequency becomes too low or if the input clock fails, this oscillator fail condition is indicated by HPOSCCON.PLLV = 0 and an interrupt request is generated.

By setting bit HPOSCCON.OSCWDTRST the detection can be restarted without a reset of the complete PLL, e.g. in case of a VCO loss-of-lock condition.

Note: The oscillator watchdog requires the trimmed currenct controlled clock f_{INT} as a reference. Therefore, it can only be used (HPOSCCON.PLLV is valid) while the clock source is active.

6.1.4.6 Switching PLL Parameters

The following restriction applies when changing PLL parameters inside the PLLCON0 to PLLCON3 registers:

- The VCO bypass switch may be used at any time, however, it has to be ensured that the maximum operating frequency of the device (see data sheet) will not be exceeded.
- Prescaler Mode should be selected.
- After switching to Prescaler Mode, NDIV and PDIV can be adjusted.
- Before deselecting the Prescaler Mode, the RESLD bit has to be set and then the VCOLOCK flag has to be checked. Only when the VCOLOCK flag is set again, the Prescaler Mode may be deselected.
- Before changing VCOSEL, the Prescaler Mode must be selected.

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

6.1.5 Clock Control Unit

The Clock Control Unit (CCU) selects the current clock sources for the clock signals used in the XE16x. It generates the following clocks:

- System clock f_{SYS}
- RTC count clock f_{RTC}
- Output clock f_{EXT}

The following clock signals can be selected:

- PLL clock f_{PLL}
- The oscillator clock (OSC_HP) f_{OSC}
- Wake-up clock f_{WU}
- Input CLKIN1 as Direct Clock Input f_{CLKIN1}
- Input CLKIN2 as Direct Clock Input f_{CLKIN2}

6.1.5.1 Clock Generation

Different clock sources can be selected for the generated clock signals.

Note: The selected clock sources are affected by the start-up procedure. See chapter Device Status after Start-up for the register values set by the different start-up procedures.

System Clock Generation

The system clock f_{SYS} can be selected from the following clock sources in the CCU:

- Wake-up clock f_{WU}
- The oscillator clock (OSC_HP) f_{OSC}
- PLL clock f_{PLL}
- Input CLKIN1 as Direct Clock Input f_{CLKIN1}

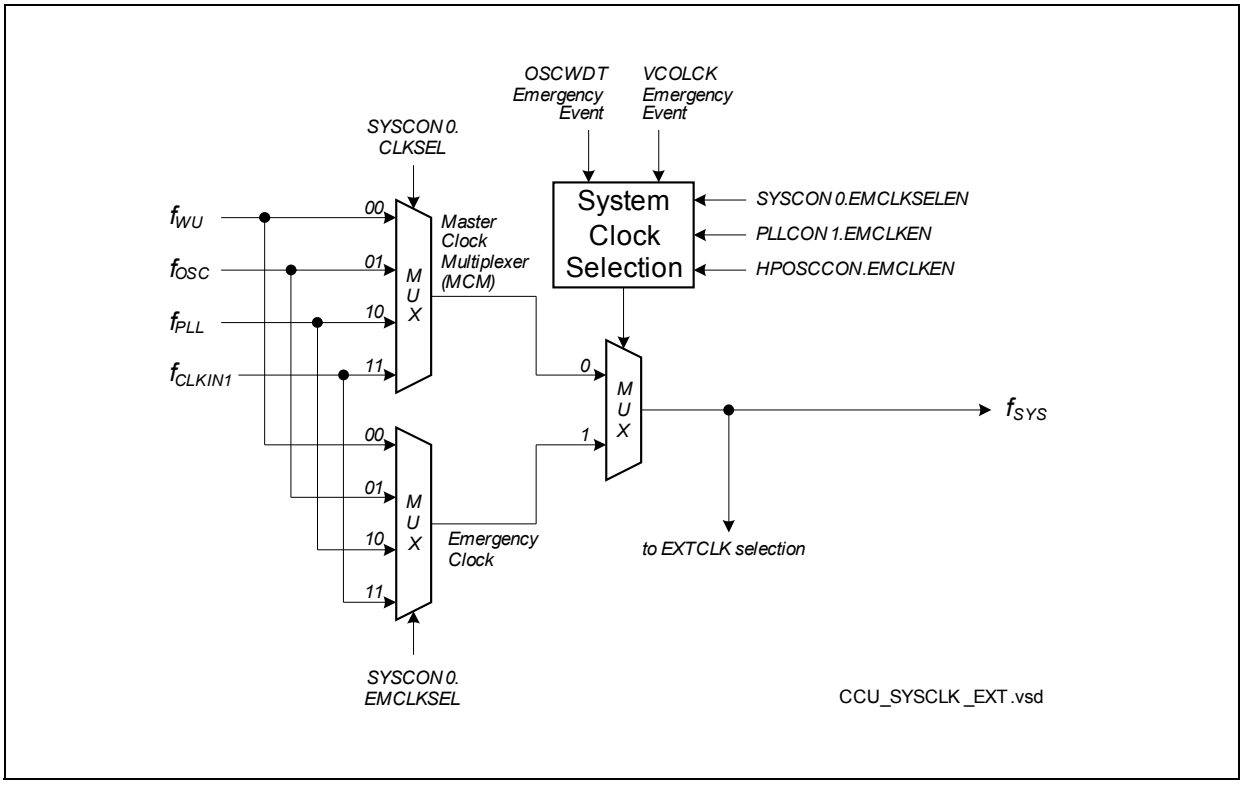


Figure 6-9 Clock Control Unit, System Clock Generation

RTC Clock Generation

For the RTC module it is possible to select the operation in synchronous or asynchronous mode in the module itself. The asynchronous clock for the RTC can be selected out of following clock sources in the CCU:

- PLL clock f_{PLL}
- The oscillator clock (OSC_HP) f_{OSC}
- Input CLKIN2 as Direct Clock Input f_{CLKIN2}
- Wake-up clock f_{WU}

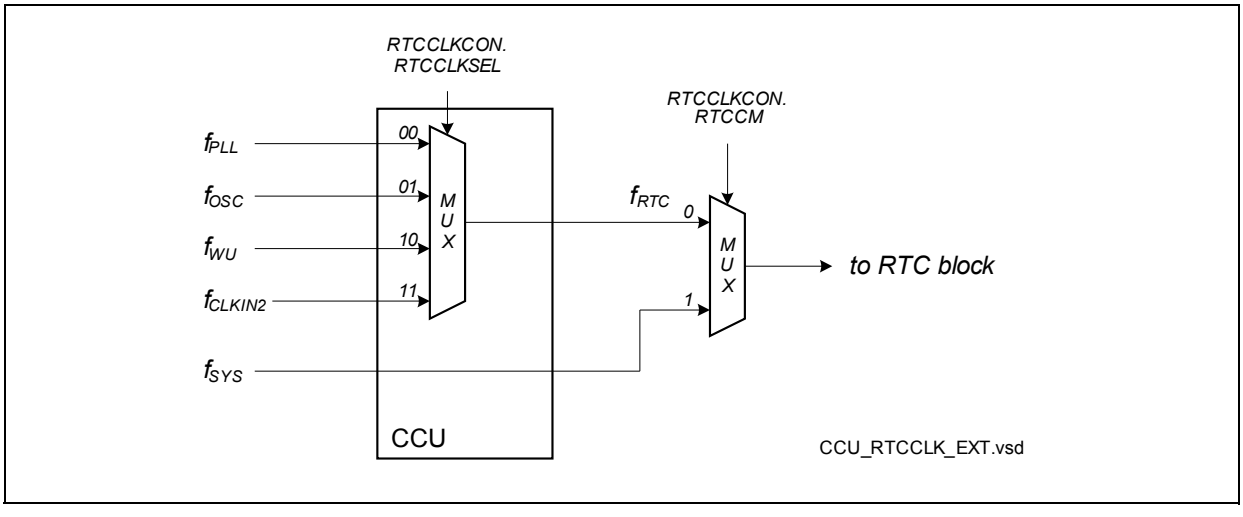


Figure 6-10 Clock Control Unit, RTC Clock Generation

6.1.5.2 Selecting and Changing the Operating Frequency

When selecting the clock source and the clock generation method, the required parameters must be carefully written to the respective bit fields, to avoid unintended intermediate states.

Many applications change the frequency of the system clock f_{SYS} during operation to optimize performance and power consumption of the system. Modifying the operating frequency changes the consumed switching current, which influences the power supply. Therefore, while the core voltage is generated by the on-chip Embedded Voltage Regulators (EVRs), the operating frequency may only be changed according to the rules given in the data sheet.

Note: To avoid the indicated problems, specific sequences are recommened that ensure the intended operation of the clock system interacting with the power system. Please refer to the document "Programmer's Guide".

6.1.5.3 System Clock Emergency Handling

The generation of the system clock f_{SYS} can be affected, if either the PLL is no more locked to its input signal f_{IN}, or if the input clock f_{IN} is no more active. Both events can be detected and are indicated to the application software. The clock system takes appropriate actions where necessary, so the device and the application is never left without an alternate clock signal.

Oscillator Watchdog Event

If the clock frequency of the external source drops below a limit value the oscillator watchdog (OSCWDT) (see **Chapter 6.1.4.5**) then the clock source for the system clock f_{SYS} is switched to an alternate clock source, if enabled (HPOSCCON.EMCLKEN = 1). In this case following information is available:

- The oscillator watchdog trap flag (TRAPSTAT.OSCWDTT) is set and a trap request to the CPU is activated, if enabled (TRAPDIS.OSCWDTT = 0).
- Bit HPOSCCON.PLLV = 0, while the clock f_{IN} is missing
- Bit SYSCON0.EMSOSC is set, if SYSCON0.EMCLKSELEN is set
- The source of the system clock f_{SYS} is switched to alternate clock source selected by SYSCON0.EMCLKSEL, if enabled (SYSCON0.EMCLKSELEN = 1). This is indicated by bit SYSCON0.SELSTAT = 1.

PLL VCO Loss-of-Lock Event

If the PLL output frequency is no longer locked to its input frequency f_{IN}, the PLL switches from PLL Normal mode to the Unlocked mode, if enabled (PLLCON1.EMFINDISEN = 1). In this case following information is available:

- The PLL VCO loss of lock trap flag (TRAPSTAT.VCOLCKT) is set and a trap request to the CPU is activated, if enabled (TRAPDIS.VCOLCKT = 0).

- Bit PLLSTAT.VCOLOCK = 0, while the PLL is not locked
- Bit SYSCON0.EMSVCO is set, if SYSCON0.EMCLKSELEN is set
- The PLL VCO clock input is disconnected (PLLSTAT.FINDIS = 1) and the PLL clock slows down to its VCO base frequency.

System Behavior

Emergency routines can be executed with the alternate clock (emergency clock or VCO base frequency). The application can then enter a safe status and stop operation, or it can switch to an emergency operating mode, where a reduced performance and/or feature set is provided.

The Programmer's Guide describes both, how to enable these features, and how to react properly on each of the two events.

6.1.6 External Clock Output

An external clock output can be provided via pin EXTCLK to clock an external system or to observe one of the selectable device clocks. This external clock is enabled by setting bit EXTCON.EN and by selecting the clock signal as alternate output function at pin EXTCLK. Following clocks can be selected by EXTCON.SEL for external clock f_{EXT}:

- System clock f_{SYS}
- Programmable clock output f_{OUT}
- Direct Clock from oscillator OSC_HP f_{OSC}
- PLL clock f_{PLL}
- Wake-up clock f_{WU}
- RTC clock f_{RTC}

Note: Changing bit field EXTCON.SEL can lead to spikes at pin EXTCLK.

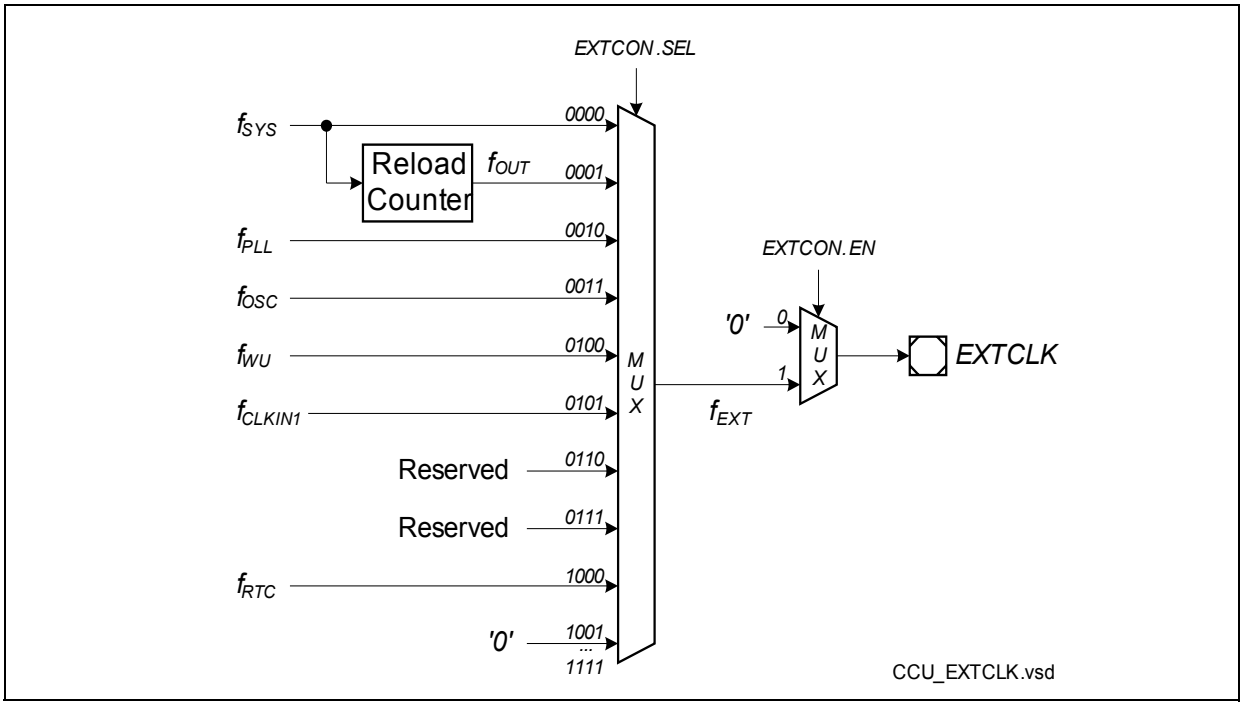


Figure 6-11 EXTCLK Generation

6.1.6.1 Programmable Frequency Output

The programmable frequency output f_{OUT} can be selected as clock output (EXTCLK). This clock can be controlled via software, and so can be adapted to the requirements of the connected external circuitry. The programmability also extends the power management to a system level, as also circuitry (peripherals, etc.) outside the XE16x can be run at a scalable frequency or can temporarily be left without a clock.

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

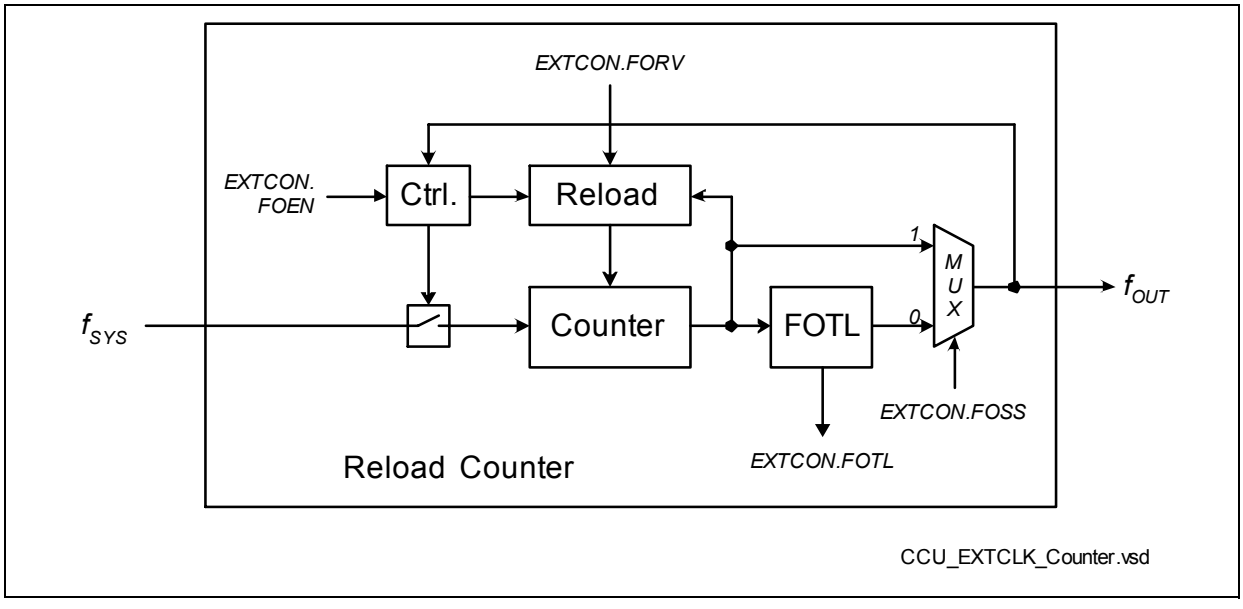


Figure 6-12 Programmable Frequency Output Generation

f_{OUT} always provides complete output periods (provided f_{SYS} is available):

- When f_{OUT} is started (EXTCON.FOEN is set) counter FOCNT is loaded from EXTCON.FORV
- When OUT is stopped (EXTCON.FOEN is cleared) counter FOCNT is stopped when f_{OUT} has reached (or is) '0'.

Register EXTCON provides control over the output generation (frequency, waveform, activation) as well as all status information (EXTCON.FOTL).

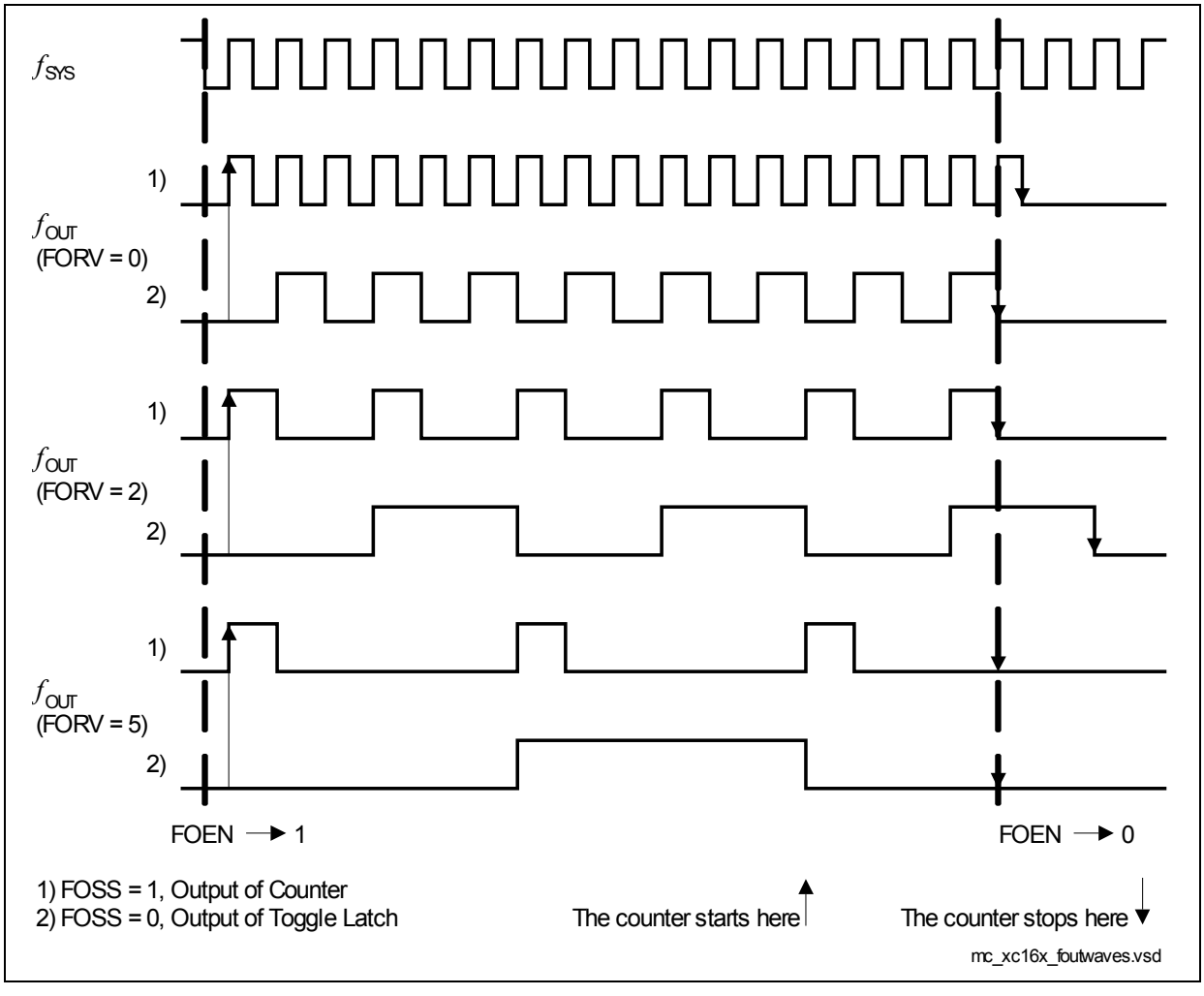


Figure 6-13 Output Waveforms Examples

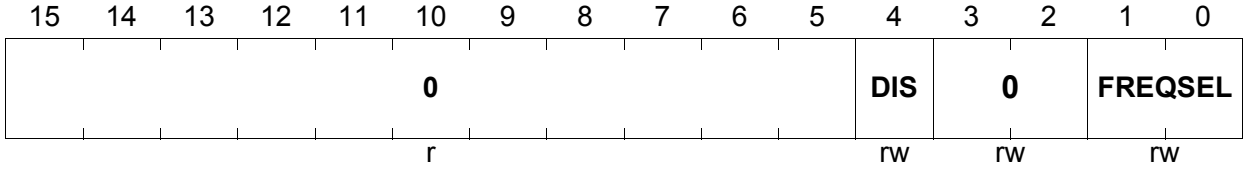
Note: The output (for EXTCON.FOSS= 1) is high for the duration of one f_{SYS} cycle for all reload values EXTCON.FORV > 0. For EXTCON.FORV = 0 the output frequency corresponds to f_{SYS}.
When a reference clock is required (e.g. for the bus interface), f_{SYS} must be selected directly.

6.1.7 CGU Registers

6.1.7.1 Wake-up Clock Register

This register controls the settings of OSC_WU.

WUOSCCON
Wake-up OSC Control Register ESFR ($F1AE_H$/$D7_H$) **Reset Value: 0000_H**



Field	Bits	Type	Description
FREQSEL	[1:0]	rw	**Frequency Selection** 00_B f_{WU} is approximately 500 kHz 01_B f_{WU} is approximately 300 kHz 10_B f_{WU} is approximately 200 kHz 11_B f_{WU} is approximately 130 kHz *Note: This value must not be changed while f_{WU} is used as clock source for any logic.*
0	[3:2]	rw	**Reserved** Must be written with reset value 00_B.
DIS	4	rw	**Clock Disable** 0_B The oscillator is switched on and the clock is enabled 1_B The oscillator is swiched off and the clock is disabled
0	[15:5]	r	**Reserved** Read as 0; should be written with 0.

6.1.7.2 High Precision Oscillator Register

This register controls the setting of OSC_HP.

HPOSCCON
High Precision OSC Control Register

ESFR ($F1B4_H$/DA_H) **Reset Value: $053C_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0			OSC2L0	OSC2L1	EMFINDISEN	EMCLKEN	SHBY	X1DEN	X1D	GAINSEL		MODE		OSCWDTRST	PLLV
r			rh	rh	rw	rw	rw	rw	rh	r		rw		w	rh

Field	Bits	Type	Description
PLLV	0	rh	**Oscillator for PLL Valid Status Bit** This bit indicates whether the frequency output of OSC_HP is usable. This is checked by the Oscillator Watchdog of the PLL. 0_B The OSC_HP frequency is not usable. The frequency is below the limit. 1_B The OSC_HP frequency is usable. The frequency is not below the limit. For more information see **Chapter 6.1.4.5**.
OSCWDTRST	1	w	**Oscillator Watchdog Reset** 0_B No action 1_B The Oscillator Watchdog of the PLL is reset and restarted
MODE	[3:2]	rw	**Oscillator Mode** 00_B The oscillator is active (crystal or ext. input) 01_B Reserved, do not use 10_B Reserved, do not use 11_B OSC_HP is disabled and in power-saving mode

Field	Bits	Type	Description
GAINSEL	[5:4]	r	**Oscillator Gain Selection** 00_B Reserved 01_B Reserved 10_B Reserved 11_B The gain control is configured for frequencies from 4 MHz to 25 MHz *Note: Used for testing only.*
X1D	6	rh	**XTAL1 Data Value** This bit reflects the inverted level of pin XTAL1. This bit is sampled with f_{SYS} while X1DEN is set. *Note: Voltages on XTAL1 must comply to the voltage defined in the data sheet.*
X1DEN	7	rw	**XTAL1 Data Enable** 0_B Bit X1D is not updated 1_B Bit X1D can be updated
SHBY	8	rw	**Shaper Bypass** The shaper forms a proper signal from the input signal. This bit must be 0 for proper operation. 0_B The shaper is not bypassed 1_B The shaper is bypassed
EMCLKEN	9	rw	**OSCWDT Emergency System Clock Source Select Enable** This bit requests the master clock multiplexer (MCM) to switch to an alternate clock (selected by bit field SYSCON0.EMCLKSEL) in an OSCWDT emergency case. 0_B MCM remains controlled by SYSCON0.CLKSEL 1_B MCM is controlled by SYSCON0.EMCLKSEL
EMFINDISEN	10	rw	**Emergency Input Clock Disconnect Enable** This bit defines if bit PLLSTAT.FINDIS is set in an OSCWDT emergency case. 0_B No action 1_B PLLSTAT.FINDIS is set in an emergency case *Note: Please refer to the Programmer's Guide for a description of the proper handling.*

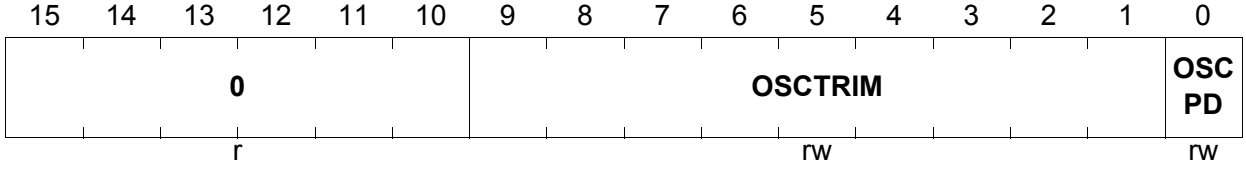
Field	Bits	Type	Description
OSC2L1	11	rh	**OSC_HP Not Usable Frequency Event** This sticky bit indicates if bit PLLV has been cleared since OSC2L1 has last been cleared (by writing 1 to bit STATCLR1.OSC2L1CLR). 0_B No change of PLLV detected 1_B Bit PLLV has been cleared at least once
OSC2L0	12	rh	**OSC_HP Usable Frequency Event** This sticky bit indicates if bit PLLV has been set since OSC2L0 has last been cleared (by writing 1 to bit STATCLR1.OSC2L0CLR). 0_B No change of PLLV detected 1_B PLLV has been set at least once
0	[15:13]	r	**Reserved** Read as 0; should be written with 0.

6.1.7.3 PLL Control Register

This register controls the trimmed current controlled clock source.

PLLOSCCON
PLL OSC Control Register

ESFR ($F1B6_H$/DB_H) **Reset Value: 0000_H**



Field	Bits	Type	Description
OSCPD	0	rw	**Clock Source Power Saving Mode** 0_B Trimmed current controlled clock source is active 1_B Trimmed current controlled clock source is off
OSCTRIM	[9:1]	rw	**Clock Source Trim Configuration** This value is used to adjust the frequency range of the current controlled clock source. Do not change this value when writing to this register.
0	[15:10]	r	**Reserved** Read as 0; should be written with 0.

6.1.7.4 PLL Registers

These registers control the settings of the PLL.

PLLSTAT
PLL Status Register **ESFR ($F0BC_H$/$5E_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0			REGSTAT	VCOL1	VCOL0	FINDIS	K2RDY	K1RDY	NRDY	PRDY	0	VCOLOCK	OSCSELST	PWDSTAT	VCOBYST
r			rh	rh	rh	rh	rh	rh	rh	rh	r	rh	rh	rh	rh

Field	Bits	Type	Description
VCOBYST	0	rh	**VCO Bypass Status** 0_B The PLL clock is derived from divider K1 (Prescaler Mode) 1_B The PLL clock is derived from divider K2 (Normal / Unlocked Mode) *Note: Coding of PLLCON0.VCOBY and VCOBYST are different.*
PWDSTAT	1	rh	**PLL Power-saving Mode Status** 0_B The PLL is operable 1_B The digital part of the PLL is disabled
OSCSELST	2	rh	**Oscillator Input Selection Status** 0_B External input clock source for the PLL (f_{IN}) 1_B Internal input clock source for the PLL
VCOLOCK	3	rh	**PLL VCO Lock Status** 0_B The frequency difference of f_{REF} and f_{DIV} is greater than allowed. The PLL cannot lock. 1_B The PLL clock f_{PLL} is locked to f_{REF} and is stable. *Note: In case of a loss of lock, the VCO frequency f_{VCO} approaches to the upper/lower boundary of the selected VCO band if the reference frequency is higher/lower than possible for locking.*

Field	Bits	Type	Description
PRDY	5	rh	**P-Divider Ready Status** 0_B Bit field PLLCON1.PDIV has been changed, new K1 divider value not yet used. 1_B The P-Divider operates with the value defined in bit field PLLCON1.PDIV.
NRDY	6	rh	**N-Divider Ready Status** 0_B Bit field PLLCON0.NDIV has been changed, new K1 divider value not yet used. 1_B The P-Divider operates with the value defined in bit field PLLCON0.NDIV.
K1RDY	7	rh	**K1-Divider Ready Status** 0_B Bit field PLLCON2.K1DIV has been changed, new K1 divider value not yet used. 1_B The K1-Divider operates with the value defined in bit field PLLCON2.K1DIV.
K2RDY	8	rh	**K2-Divider Ready Status** 0_B Bit field PLLCON3.K2DIV has been changed, new K2 divider value not yet used. 1_B The K2-Divider operates with the value defined in bit field PLLCON3.K2DIV.
FINDIS	9	rh	**Input Clock Disconnect Select Status** 0_B The VCO is connected to the reference clock 1_B The VCO is disconnected from the reference clock *Note: Software can control this bit by writing 1 to bits SETFINDIS or CLRFINDIS in register STATCLR1.*
VCOL0	10	rh	**VCO Lock Detection Lost Status** This sticky bit indicates if bit VCOLOCK has been cleared since VCOL0 has last been cleared (by writing 1 to bit STATCLR1.VCOL0CLR). 0_B No falling edge detected 1_B PLLV has been cleared at least once (VCO lock was lost)

Field	Bits	Type	Description
VCOL1	11	rh	**VCO Lock Detection Reached Status** This sticky bit indicates if bit VCOLOCK has been set since VCOL1 has last been cleared (by writing 1 to bit STATCLR1.VCOL1CLR). 0_B No rising edge detected 1_B VCO lock was reached
REGSTAT	12	rh	**PLL Power Regulator Status** The PLL is powered by a separate internal regulator. 0_B The PLL is not powered (off) 1_B The PLL is powered (operation possible) *Note: Software can control this bit by writing 1 to bits REGENSET or REGENCLR in register PLLCON0.*
0	4, [15:13]	r	**Reserved** Read as 0; should be written with 0.

STATCLR1
PLL Status Clear 1 Register **ESFR ($F0E2_H$/71_H)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0										CLRFINDIS	SETFINDIS	OSC2L0CLR	OSC2L1CLR	VCOL1CLR	VCOL0CLR
r										w	w	w	w	w	w

Field	Bits	Type	Description
VCOL0CLR	0	w	**VCOL0 Clear Trigger** 0_B No action 1_B Bit PLLSTAT.VCOL0 is cleared
VCOL1CLR	1	w	**VCOL1 Clear Trigger** 0_B No action 1_B Bit PLLSTAT.VCOL1 is cleared
OSC2L1CLR	2	w	**OSC2L1 Clear Trigger** 0_B No action 1_B Bits HPOSCCON.OSC2L1 is cleared
OSC2L0CLR	3	w	**OSC2L0 Clear Trigger** 0_B No action 1_B Bit HPOSCCON.OSC2L0 is cleared
SETFINDIS	4	w	**Set Status Bit PLLSTAT.FINDIS** 0_B No action 1_B Bit PLLSTAT.FINDIS is set. The VCO input clock is disconnected.
CLRFINDIS	5	w	**Clear Status Bit PLLSTAT.FINDIS** 0_B No action 1_B Bit PLLSTAT.FINDIS is cleared. The VCO input clock is connected.
0	[15:6]	r	**Reserved** Read as 0; should be written with 0.

PLLCON0
PLL Configuration 0 Register ESFR ($F1B8_H$/DC_H) Reset Value: 1302_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NACK	0	NDIV						0		REGENSET	REGENCLR	VCOSEL		VCOPWD	VCOBY
rw	r	rw						r		w	w	rw		rw	rw

Field	Bits	Type	Description
VCOBY	0	rw	**VCO Bypass** 0_B Select divider K2 for PLL clock (Normal / Unlocked Mode) 1_B Select divider K1 for PLL clock (Prescaler Mode, i.e. VCO is bypassed) Bit PLLSTAT.VCOBYST shows the actually selected divider. *Note: Coding of VCOBY and PLLSTAT.VCOBYST are different.*
VCOPWD	1	rw	**VCO Power Saving Mode** 0_B Normal behavior 1_B The VCO is put into a power saving mode and can no longer be used. Only the Prescaler Mode is active if previously selected.
VCOSEL	[3:2]	rw	**VCO Range Select** The values for the different settings are listed in the data sheet.
REGENCLR	4	w	**Power Regulator Enable Clear** 0_B No action 1_B Switch off the PLL's power regulator. The PLL is not powered (no operation possible).
REGENSET	5	w	**Power Regulator Enable Set** 0_B No action 1_B Switch on the PLL's power regulator. The PLL is powered (operation possible).

Field	Bits	Type	Description
NDIV	[13:8]	rw	**N-Divider Value** The value the N-Divider operates is NDIV+1. Only values between N = 8 and N = 28 are allowed for VCOSEL = 00_B. Only values between N = 16 and N = 40 are allowed for VCOSEL = 01_B. Outside of this range no stable operation is guaranteed.
NACK	15	rw	**N-Divider Ready Acknowledge** Setting this bit provides the acknowledge signal to NRDY.
0	6, 7 14	r	**Reserved** Read as 0; should be written with 0.

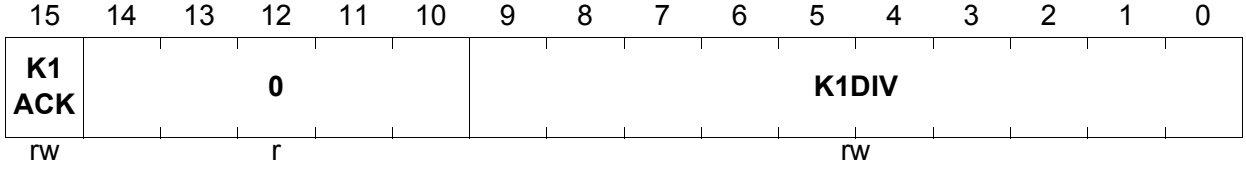
PLLCON1
PLL Configuration 1 Register ESFR ($F1BA_H/DD_H$) Reset Value: $000A_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PACK	0			PDIV				0	EMFINDISEN	EMCLKEN	0	AOSCSEL	RESLD	OSCSEL	PLLPWD
rw	r			rw				r	rw	rw	r	rw	w	rw	rw

Field	Bits	Type	Description
PLLPWD	0	rw	**PLL Power Saving Mode** 0_B Normal behavior 1_B Complete PLL block is put into a power saving mode and no longer operates
OSCSEL	1	rw	**Oscillator Input Selection** 0_B Select external clock as input for PLL 1_B Select trimmed current controlled clock as input for PLL
RESLD	2	w	**Restart VCO Lock Detection** Setting this bit will reset bit PLLSTAT.VCOLOCK and restart the VCO lock detection.
AOSCSEL	3	rw	**Asynchronous Oscillator Input Selection** This bit overrules the setting of bit OSCSEL. 0_B Configuration is controlled via bit OSCSEL 1_B Select asynchronously trimmed current controlled clock as input for PLL
EMCLKEN	5	rw	**VCOLCK Emergency System Clock Source Select Enable** This bit requests the master clock multiplexer (MCM) to switch to an alternate clock (selected by bit field SYSCON0.EMCLKSEL) in a VCOLCK emergency case. 0_B MCM remains controlled by SYSCON0.CLKSEL 1_B MCM is controlled by SYSCON0.EMCLKSEL

Field	Bits	Type	Description
EMFINDISEN	6	rw	**Emergency Input Clock Disconnect Enable** This bit defines if bit PLLSTAT.FINDIS is set in a VCOLCK emergency case. 0_B No action 1_B PLLSTAT.FINDIS is set in a VCOLCK emergency case *Note: Please refer to the Programmer's Guide for a description of the proper handling.*
PDIV	[11:8]	rw	**P-Divider Value** The value the P-Divider operates is PDIV+1.
PACK	15	rw	**P-Divider Ready Acknowledge** Setting this bit provides the acknowledge to PRDY.
0	4, 7, [14:12]	r	**Reserved** Read as 0; should be written with 0.

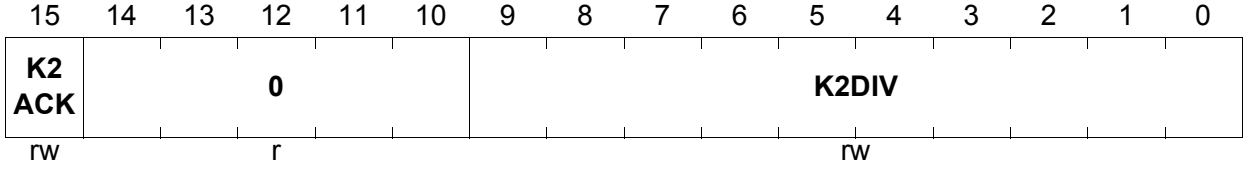
PLLCON2
PLL Configuration 2 Register ESFR ($F1BC_H/DE_H$) Reset Value: 0001_H



Field	Bits	Type	Description
K1DIV	[9:0]	rw	**K1-Divider Value** The value the K1-Divider operates is K1DIV+1.
K1ACK	15	rw	**K1-Divider Ready Acknowledge**[1)] Setting this bit provides the acknowledge to K1RDY.
0	[14:10]	r	**Reserved** Read as 0; should be written with 0.

1) Please refer to the Programmer's Guide for a description of the proper handling.

PLLCON3
PLL Configuration 3 Register ESFR ($F1BE_H/DF_H$) Reset Value: $00CB_H$



Field	Bits	Type	Description
K2DIV	[9:0]	rw	**K2-Divider Value** The value the K2-Divider operates is K2DIV+1.
K2ACK	15	rw	**K2-Divider Ready Acknowledge**[1)] Setting this bit provides the acknowledge to K2RDY.
0	[14:10]	r	**Reserved** Read as 0; should be written with 0.

1) Please refer to the Programmer's Guide for a description of the proper handling.

6.1.7.5 System Clock Control Registers

These registers control the system level clock behavior.

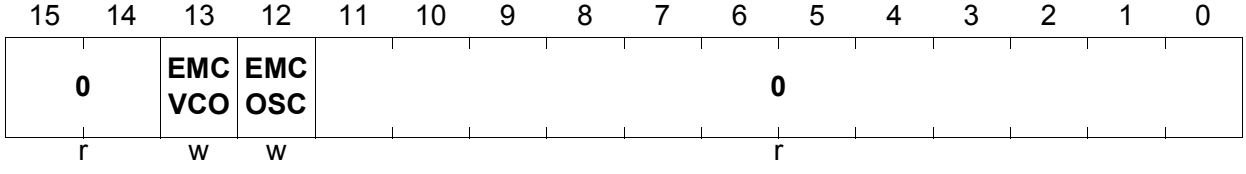
SYSCON0
System Control 0 Register **SFR ($FF4A_H/A5_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SELSTAT	0	EMSVCO	EMSOSC	0					EMCLKSELEN	0	EMCLKSEL		0	CLKSEL	
rh	r	rh	rh	r					rw	r	rw		r	rw	

Field	Bits	Type	Description
CLKSEL	[1:0]	rw	**Clock Select** This bit field defines the clock source that is used as system clock for normal operation. 00_B The Wake-up clock f_{WU} is used 01_B The oscillator clock (OSC_HP) f_{OSC} is used 10_B The PLL clock f_{PLL} is used 11_B CLKIN1 as direct input clock f_{CLKIN1} is used
EMCLKSEL	[4:3]	rw	**Emergency Clock Select** This bit field defines the clock source that is used as system clock in case of an OSCWDT or VCOLCK emergency event. 00_B The Wake-up clock f_{WU} is used 01_B The oscillator clock (OSC_HP) f_{OSC} is used 10_B The PLL clock f_{PLL} is used 11_B CLKIN1 as direct input clock f_{CLKIN1} is used
EMCLKSELEN	6	rw	**Emergency Clock Select Enable** Controls switching the system clock to an alternate source in case of an OSCWDT or VCOLCK event. 0_B The switching is disabled 1_B The switching is enabled
EMSOSC	12	rh	**OSCWDT Emergency Event Source Status** 0_B No OSCWDT emergency event occurred since EMSOSC has been cleared last 1_B An OSCWDT emergency event has occurred *Note: This bit is only set if EMCLKSELEN is set.*

Field	Bits	Type	Description
EMSVCO	13	rh	**VCOLCK Emergency Event Source Status** 0_B No VCOLCK emergency event occurred since EMSVCO has been cleared last 1_B A VCOLCK emergency event has occurred *Note: This bit is only set if EMCLKSELEN is set.*
SELSTAT	15	rh	**Clock Select Status** 0_B The standard configuration from bit field CLKSEL is used currently 1_B The configuration from bit field EMCLKSEL is used currently
0	2, 5, [11:7], 14	r	**Reserved** Read as 0; should be written with 0.

STATCLR0
Status Clear 0 Register **ESFR ($F0E0_H/70_H$)** **Reset Value: 0000_H**

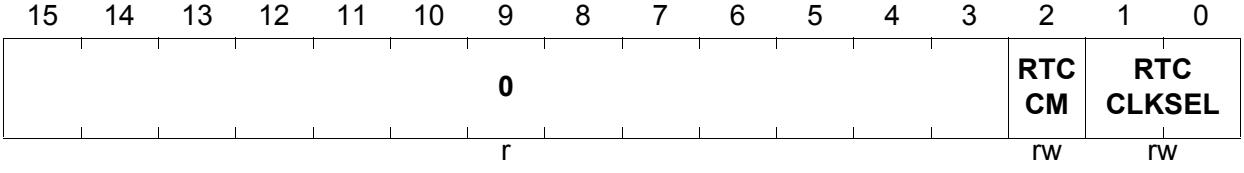


Field	Bits	Type	Description
EMCOSC	12	w	**EMSOSC Clear Trigger** 0_B No action 1_B Bit SYSCON0.EMSOSC is cleared
EMCVCO	13	w	**EMSVCO Clear Trigger** 0_B No action 1_B Bit SYSCON0.EMSVCO is cleared
0	[11:0], [15:14]	r	**Reserved** Read as 0; should be written with 0.

6.1.7.6 RTC Clock Control Register

Note: Only change register RTCCLKCON while the RTC is off.

RTCCLKCON
RTC Clock Control Register **SFR ($FF4E_H/A7_H$)** **Reset Value: 0006_H**



Field	Bits	Type	Description
RTCCLKSEL	[1:0]	rw	**RTC Clock Select** This bit field defines the count clock source for the RTC. 00_B The PLL clock f_{PLL} is used 01_B The oscillator clock (OSC_HP) f_{OSC} is used 10_B The Wake-up clock signal f_{WU} is used 11_B CLKIN2 as direct input clock f_{CLKIN2} is used
RTCCM	2	rw	**RTC Clocking Mode** 0_B Asynchronous Mode: The RTC internally operates with f_{RTC}. No register access is possible. 1_B Synchronous Mode: The RTC internally operates with f_{SYS} clock. Registers can be read and written.
0	[15:3]	r	**Reserved** Read as 0; should be written with 0.

6.1.7.7 External Clock Control Register

This register control the setting of external clock for pin 2.8 and 7.1.

EXTCON
External Clock Control Register SFR ($FF5E_H/AF_H$) **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FOEN	FOSS	FORV						0	FOTL	0	SEL				EN
rw	rw	rw						r	rh	r	rw				rw

Field	Bits	Type	Description
EN	0	rw	**External Clock Enable** 0_B No external clock signal is provided. The signal is tied to zero. 1_B The configured external clock signal is provided as alternate output signal
SEL	[4:1]	rw	**External Clock Select** Selects the clock signal to be routed to the EXTCLK pin: 0000_B System clock f_{SYS} 0001_B Programmable clock signal f_{OUT} 0010_B PLL output clock f_{PLL} 0011_B Oscillator clock f_{OSC} 0100_B Wake-up clock f_{WU} 0101_B Direct Input clock f_{CLKIN1} 1000_B RTC count clock f_{RTC} All other combination are reserved, do not use.
FOTL	6	rh	**Frequency Output Toggle Latch** Toggled upon each underflow of FOCNT.
FORV	[13:8]	rw	**Frequency Output Reload Value** Copied to FOCNT upon each underflow of FOCNT.
FOSS	14	rw	**Frequency Output Signal Select** 0_B Output of the toggle latch 1_B Output of the reload counter: duty cycle depends on FORV

Field	Bits	Type	Description
FOEN	15	rw	**Frequency Output Enable** 0_B Frequency output generation stops when f_{OUT} is/becomes low. 1_B FOCNT is running, f_{OUT} is gated to pin. First reload after 0 - 1 transition.
0	5, 7	r	**Reserved** Read as 0; should be written with 0.

6.2 Wake-up Timer (WUT)

The Wake-up Timer provides an additional resource to trigger system functions after a specific period of time.

The master clock f_{SYS} is prescaled and drives a simple counter. All functions are controlled by register **WUCR**.

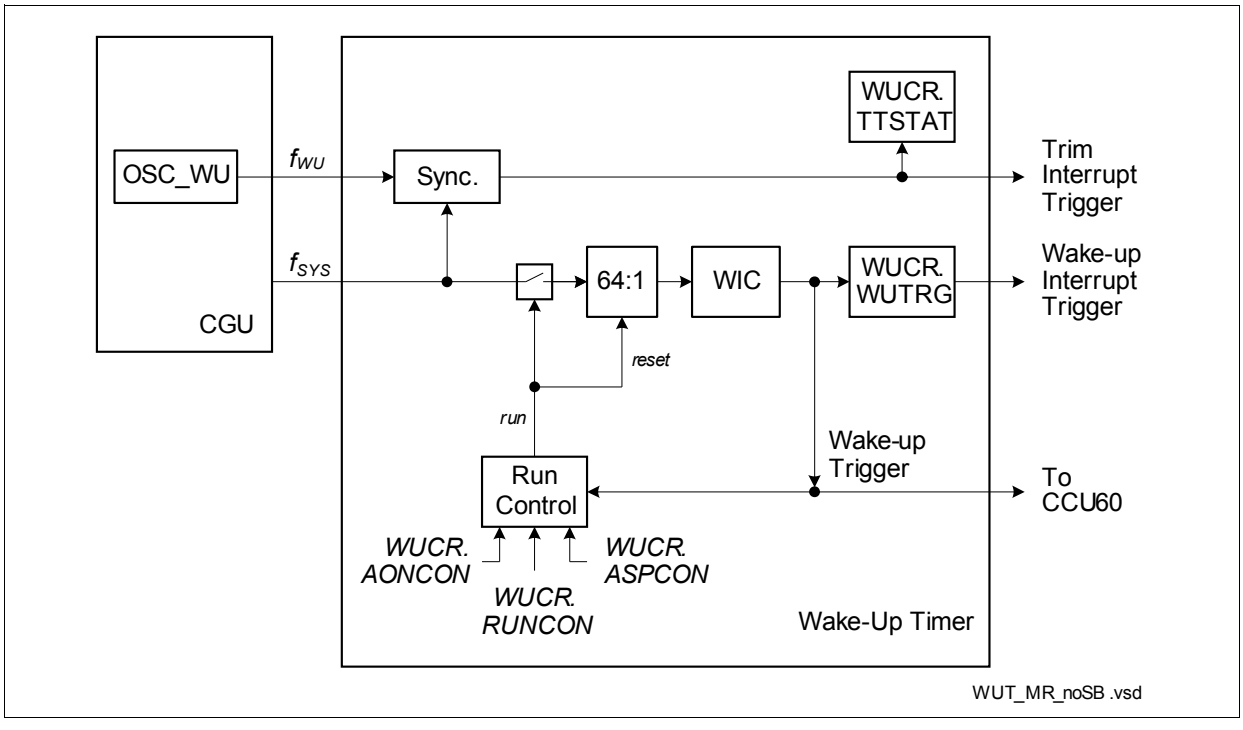


Figure 6-14 Wake-up Timer Logic

6.2.1 Wake-up Timer Operation

The Wake-up Timer start and stop is controlled by the Run Control logic. The timer can be started in the following way:

- bit WUCR.RUN is set

When the timer is started the prescaler is reset and the counter WIC starts to count down.

The wake-up interval counter (WIC) is clocked with f_{SYS}/64, and counts down until it reaches zero. It then generates a wake-up trigger and sets bit WUCR.WUTRG.

The timer is stopped in the following ways:

- bit WUCR.RUN is cleared
- bit WUCR.ASP is set AND a wake-up trigger is generated

If the counter WIC is not stopped by its zero trigger it continues counting down from $FFFF_H$.

Determination of Wake-up Period

The actual frequency of the trimmed current controlled wake-up clock (OSC_WU) can be measured prior to entering power-save mode in order to adjust the number of clock cycles to be counted (value written to WIC), and such, to define the time until wake-up. The period of OSC_WU can be measured by evaluating its (synchronized) clock output, which can generate an interrupt request or which can be monitored via bit WUCR.TTSTAT.

As using an interrupt together with software contain some uncertainty there is a second way to determine the wake-up period. The wake-up triggers generated by the WUT are forwarded to the CCU60 and can there be evaluated compared to the accurate system clock.

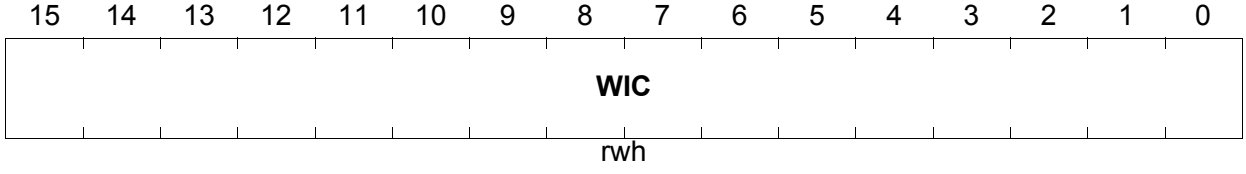
6.2.2 WUT Registers

6.2.2.1 Register WICR

Via this register the status and configuration of the WIC counter is done.

WICR
Wake-up Interval Count Register

ESFR ($F0B0_H$/58_H) **Reset Value: $FFFF_H$**



Field	Bits	Type	Description
WIC	[15:0]	rwh	**Wake-up Interval Counter** This free-running 16-bit counter counts down and issues a trigger when its count reaches zero.

6.2.2.2 Register WUCR

This register the status and control bits for the WUT.

WUCR
Wake-up Control Register **ESFR ($F1B0_H$/$D8_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WU TRG	TTS TAT	0			ASP	AON	RUN	CLR TRG	0	ASP CON		AON CON		RUN CON	
rh	rh	r			rh	rh	rh	w	r	w		w		w	

Field	Bits	Type	Description
RUNCON	[1:0]	w	**Control Field for RUN** 00_B No action 01_B Set bit RUN 10_B Clear bit RUN 11_B Reserved, do not use this combination
AONCON	[3:2]	w	**Control Field for AON** 00_B No action 01_B Reserved, do not use this combination 10_B Clear bit AON 11_B Reserved, do not use this combination
ASPCON	[5:4]	w	**Control Field for ASP** 00_B No action 01_B Set bit ASP 10_B Clear bit ASP 11_B Reserved, do not use this combination
CLRTRG	7	w	**Clear Bit WUTRG** 0_B No action 1_B Clear bit WUTRG
RUN	8	rh	**Run Indicator** 0_B Wake-up counter is stopped 1_B Wake-up counter is counting down *Note: Clearing this bit via a write action to bit field RUNCON stops the WUT after four cycles of f_{WUT}.*

Field	Bits	Type	Description
AON	9	rh	**Auto-Start Indicator** 0_B Wake-up counter is started by software only 1_B Reserved, do not use this combination *Note: This bit is cleared by writing 01_B to bit field AONCON.*
ASP	10	rh	**Auto-Stop Indicator** 0_B Wake-up counter runs continuously 1_B Wake-up counter stops after generating a trigger when reaching zero
TTSTAT	14	rh	**Trim Trigger Status** 0_B No trim trigger event is active. No trim interrupt trigger is generated. 1_B A trim trigger event is active. A trim interrupt trigger is generated. *Note: This bit is not valid if $f_{SYS} = f_{WU}$ is configured by SYSCON0.CLKSEL*
WUTRG	15	rh	**WUT Trigger Indicator** 0_B No trigger event has occurred since WUTRG has been cleared last. No interrupt trigger is generated. 1_B A wake-up trigger event has occurred. A wake-up interrupt trigger is generated.
0	6, [13:11]	r	**Reserved** Read as 0; should be written with 0.

Note: The bits in the upper byte of register WUCR indicate the current status of the wake-up counter logic. They are not influenced by a write access, but are controlled by their associated control fields (lower byte) or by hardware.
The control bit(field)s in the lower byte of register WUCR determine the state of the status bits (upper byte) of the wake-up counter logic. Setting bits by software triggers the associated action, writing 0 has no effect.

6.3 Reset Operation

All resets are generated by the Reset Control Block. It handles the control of the reset triggers as well as the length of a reset and the reset timing. A reset leads the system, or a part of the system depending on the reset, to a initialization into a defined state.

6.3.1 Reset Architecture

The XE16x contains a very sophisticated reset architecture to offer the greatest amount of flexibility for the support of different applications. The reset architecture supports the different power domains.

Different reset types for the complete system are supported.

6.3.1.1 Device Reset Hierarchy

The device reset hierarchy is divided according to the power domains (see **Chapter 6.5**) into following linked levels:

Level 1: I/O domain (power domain DMP_B)

Level 2: System power domain DMP_M

Level 3: System power domain DMP_1

If a power domain (level) is deactivated all resets of the deactivated level and all resets of all lower power domains are asserted.

6.3.1.2 Reset Types

The following summary shows the different reset types.

Power Reset

- Power-on Reset
  This reset leads to a defined state of the complete system. This reset should only be requested on a real power-on event and not by any non power related event.
- Power Reset for DMP_M and DMP_1 power domains
  This reset regains data consistency upon a power fail in the DMP_M or DMP_1 power domains.

Functional / User Reset

- Debug Reset
  This reset leads to a defined state of the complete debug system.
- Internal Application Reset
  This reset leads to a defined state of the complete application system with the following parts: all peripherals (except the RTC), the CPU and partially the SCU and the flash memory.

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

After a reset has been executed, the Reset Status registers RSTSTATx indicate the latest reset that has occured.

To identify the type and the trigger of the latest reset registers RSTSTATx and may be **SWDCON1**evaluated according to the following table. The latest reset that has occured is always the reset of the highest type. If two reset triggers of the same type are indicated, this means that the two triggers have been active at the same time. If two or more reset triggers of a different type are reported, always the reset of the highest type is the latest one.

Table 6-1 Identification of a reset

Type of Reset (in hierarchical order, highest on top)	**Identification**
Power-on Reset	SWDCON1.PON = 1_B RSTSTAT1.STM = 11_B RSTSTAT1.ST1 = 11_B Further action: clear PON bit to be able to identify a Power Reset for DMP_M and DMP_1 power domains.
Power Reset for DMP_M and DMP_1 power domains	SWDCON1.PON = 0_B (unchanged after clearing) RSTSTAT1.STM = 11_B RSTSTAT1.ST1 = 11_B
Internal Application Reset	RSTSTAT1.ST1 = 00_B RSTSTATx.y = 10_B
Application Reset	RSTSTAT1.ST1 = 00_B RSTSTATx.y = 11_B

6.3.2 General Reset Operation

A reset is generated if an enabled reset request trigger is asserted. Most reset request triggers can be configured for the reset type it should initiate. No action (disabled) is one possible configuration and can be selected for a reset request trigger by setting the respective bit field in a Reset Configuration Register to 00_B. The debug reset can only be requested by dedicated reset request triggers and can not be selected via a Reset Configuration Register. For more information see also registers **RSTCON0** and **RSTCON1**.

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

6.3.2.1 Reset Counters (RSTCNTA and RSTCNTD)

RSTCNTA is the reset counter that controls the reset length for all application relevant resets (Internal Application Reset, and Application Reset). RSTCNTD is the reset counter that controls the reset length for the debug reset.

The reset counters control the length of the internal resets. This can be used to configure the duration of a reset output via the ESRx pins, so this matches with the reset input requirements of external blocks connected to these signals.

A reset counter RSTCNT is an 8-bit counter counting down from the reload value defined by **RSTCNTCON**.RELx (x = A or D). The counter is started by the reset control block as soon as a reset request trigger condition becomes active (for more information see **Table 6-2** and **Table 6-3**). Whether the counter has to be started or not depends on the reset request trigger and whether the counter is already active or not. In case of that the counter is inactive, not counting down, it is always started. While the counter is already active it depends on the reset type of the new reset request trigger that was asserted anew if the counter is restarted or not. This behavior is summarized in **Table 6-2** and **Table 6-3**.

Table 6-2 Restart of RSTCNTA

Reset Active	New Reset Trigger			
	Power-On	Debug Reset	Internal Application Reset	Application Reset
Internal Application Reset	Restart with default delay	No Change	No Change	No Change
Application Reset	Restart with default delay	No Change	Restart with defined delay	No Change

Table 6-3 Restart of RSTCNTD

Reset Active	New Reset Trigger			
	Power-On	Debug Reset	Internal Application Reset	Application Reset
Debug Reset	Restart with default delay	No Change	No Change	No Change

The reset counters RSTCNTx ensure a configurable minimum duration of a generated reset. If a reset request trigger remains asserted after the respective counter has counted down, the counter is not started again, instead the reset control block keeps the reset asserted until the reset request trigger is deasserted.

6.3.2.2 De-assertion of a Reset

The reset of a dedicated type is de-asserted when all of the following conditions are fulfilled:

- The reset counter has been expired (reached zero).
- No reset request trigger that is configured to generate a reset of the dedicated type (or higher) is currently asserted.

Example1

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

Example2

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

6.3.3 Debug Reset Assertion

Unlike the other reset types a Debug Reset can only be asserted if the following two conditions are valid:

- A reset request trigger is asserted that request a debug reset
- An Application Reset is already active in the system

6.3.4 Coupling of Reset Types

The different reset types are coupled for a better usage:

- The assertion of a Power-on Reset automatically asserts also the following reset types:
  - Debug Reset
  - Internal Application Reset
  - Application Reset
- The assertion of an Internal Application Reset automatically asserts also the following reset type:
  - Application Reset

6.3.5 Reset Request Trigger Sources

The following overview summarizes the different reset request trigger sources within the system.

Power-On Reset Pin $\overline{\text{PORST}}$

A Power-on Reset is requests asynchronously, by driving the $\overline{\text{PORST}}$ pin low.

Supply Watchdog (SWD)

If the power supply for I/O domain is below the value required for proper functionality, a non-synchronized reset request trigger is generated if the SWD reset generation is enabled. This ensures a reproducible behavior in the case of power-fail. This can also be used to restart the system without the usage of the $\overline{\text{PORST}}$ pin. As long as the I/O power domain does not get the required voltage level the system is held in the reset.

Core Power Validation (PVC_M and PVC_1)

If the core power supply is below the value required for proper functionality of the main power domain (PVC_M), a reset request trigger can be forwarded to the system. The generation of a Power-on Reset is configured by bit PVCMCON0.L1RSTEN = 1_B. If the bit PVCMCON0.L1RSTEN = 1_B a request trigger is asserted for PVC_M1 upon a level check match. If the bit PVCMCON0.L2RSTEN = 1_B a request trigger is asserted for PVC_M2 upon a level check match.

If the core power supply is below the value required for proper functionality of the application power domain (PVC_1), a reset request trigger can be forwarded to the system. The generation of a Power-on Reset (Application Power Domain only) is configured by bit PVC1CON0.L1RSTEN = 1_B. If bit PVC1CON0.L1RSTEN = 1_B a request trigger is asserted for PVC_11 upon a level check match. If the bit PVC1CON0.L2RSTEN = 1_B a request trigger is asserted for PVC_12 upon a level check match.

For more information about the Power Validation Circuit see **Chapter 6.5.2**.

$\overline{\text{ESRx}}$

An $\overline{\text{ESRx}}$ reset request trigger leads to a configurable reset. The type of reset can be configured via **RSTCON1**.ESRx.

The pins $\overline{\text{ESRx}}$ can serve as an external reset input as well as a reset output (open drain) for Internal Application and Application Resets. Furthermore, several GPIO pad triggers, that can be enabled additionally via register ESREXCONx (x = 1, 2), interfere with the ESR pin function. GPIO and $\overline{\text{ESRx}}$ pin triggers can be enabled/disabled individually and are combined for the reset trigger generation.

If pin $\overline{\text{ESRx}}$ is enabled as reset output and the input level is low while the output stage is disabled (indicating that it is still driven low externally), the reset circuitry holds the chip in reset until a high level is detected on $\overline{\text{ESRx}}$. Minimum value for RSTCNTCON.RELA must be the reset value.

Note: The reset output is only driven low for the duration the reset counter RSTCNTA is active. During a possible reset extension the reset output is no longer driven.

Software

A software reset request trigger leads to a configurable reset. The type of reset can be configured via **RSTCON0**.SW.

Watchdog Timer

A WDT reset request trigger leads to a configurable reset. The type of reset can be configured via **RSTCON1**.WDT. A WDT reset is requested on a WDT overflow event. For more information see **Chapter 6.11**.

CPU

A CPU reset request trigger leads to a configurable reset. The type of reset can be configured via **RSTCON0**.CPU. A CPU reset is requested when instruction SRST is executed.

Memory Parity

A MP reset request trigger leads to a configurable reset. The type of reset can be configured via **RSTCON1**.MP. For more information see **Chapter 6.13.1**.

OCDS Block

The OCDS block has several options to request different reset types:

1. A Debug Reset either via the OCDS reset function or via bit CBS_OJCONF.RSTCL1 AND CBS_OJCONF.RSTCL3
2. An Internal Application Reset via bit CBS_OJCONF.RSTCL2
3. An Application Reset via bit CBS_OJCONF.RSTCL3

6.3.5.1 Reset Sources Overview

The connection of the reset sources and the activated reset types are shown in **Table 6-4**.

Table 6-4 Effects of Reset Types for Reset Activation

Reset Request Trigger	Application Reset	Internal Application Reset	Debug Reset
$\overline{\text{PORST}}$	Activated	Activated	Activated
SWD	Activated	Activated	Activated
PVC_M1	Activated	Activated	Activated
PVC_M2	Activated	Activated	Activated
PVC_11	Activated	Activated	Activated
PVC_12	Activated	Activated	Activated
$\overline{\text{ESR0}}$	Configurable	Configurable	Not Activated
$\overline{\text{ESR1}}$	Configurable	Configurable	Not Activated
$\overline{\text{ESR2}}$	Configurable	Configurable	Not Activated
WDT	Configurable	Configurable	Not Activated
SW	Configurable	Configurable	Not Activated
CPU	Configurable	Configurable	Not Activated
MP	Configurable	Configurable	Not Activated
OCDS Reset	Not Activated	Not Activated	Activated[1)]
CBS_OJCONF.RSTCL1	Not Activated	Not Activated	Activated[1)]
CBS_OJCONF.RSTCL2	Activated	Activated	Not Activated
CBS_OJCONF.RSTCL3	Activated	Not Activated	Not Activated

[1)] Only if an Application Reset is active or is requested in parallel.

6.3.6 Module Reset Behavior

Table 6-5 lists how the various functions of the XE16x are affected through a reset depending on the reset type. A “X” means that this block has at least some register/bits that are affected by this reset type.

Table 6-5 Effect of Reset on Device Functions

Module / Function		Application Reset	Internal Application Reset	Debug Reset
CPU Core		X	X	X
Peripherals (except SCU and RTC)		X	X	X
SCU		X	Not affected	Not affected
RTC		Not affected	Not affected	X
On-chip Static RAMs[1]	**DPRAM**	Not affected, reliable	Not affected, reliable	Not affected, reliable
	PSRAM	Not affected, reliable	Not affected, reliable	Not affected, reliable
	DSRAM	Not affected, reliable	Not affected, reliable	Not affected, reliable
Flash Memory		X [2]	X [2]	Not affected, reliable
JTAG Interface		Not affected	Not affected	Not affected
OCDS		Not affected	Not affected	X
Oscillator, PLL		Not affected	Not affected	Not affected
Port Pins		Not affected	X	Not affected
Pins $\overline{\text{ESRx}}$		Not affected	X	Not affected

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

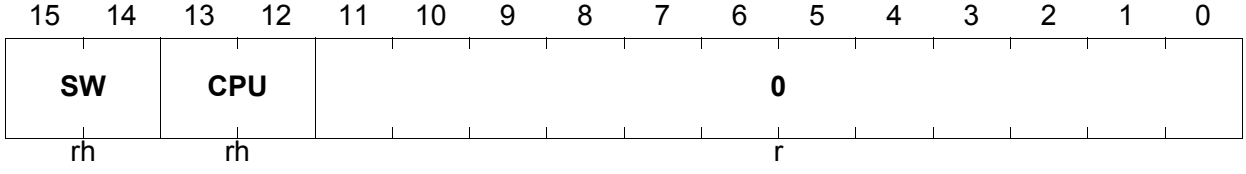
2) Parts of the flash memory block are only reset by a Power-on Reset. For more detail see the flash chapter.

6.3.7 Reset Controller Registers

6.3.7.1 Status Registers

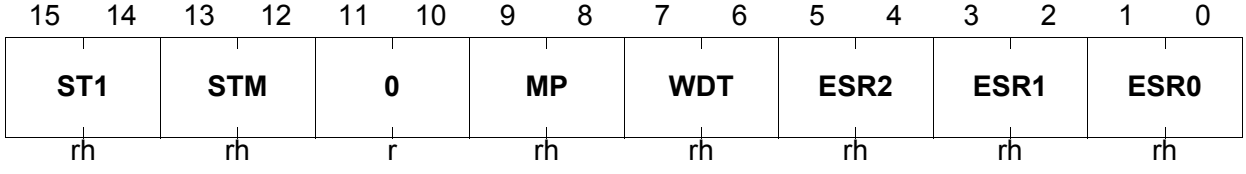
After a reset has been executed, the Reset Status registers provide information on the type of the last reset.

RSTSTAT0
Reset Status 0 Register **ESFR ($F0B2_H/59_H$)** **Reset Value: 0000_H**



Field	Bits	Type	Description
CPU	[13:12]	rh	**CPU Reset Type Status** 00_B The CPU reset trigger was not relevant for the last reset 01_B Reserved 10_B The CPU reset trigger was relevant for the last reset. Internal Application and Application Resets were generated. 11_B The CPU reset trigger was relevant for the last reset. Application Reset was generated.
SW	[15:14]	rh	**Software Reset Type Status** 00_B The Software reset trigger was not relevant for the last reset 01_B Reserved 10_B The Software reset trigger was relevant for the last reset. Internal Application and Application Resets were generated. 11_B The Software reset trigger was relevant for the last reset. Application Reset was generated.
0	[11:0]	r	**Reserved** Read as 0; should be written with 0.

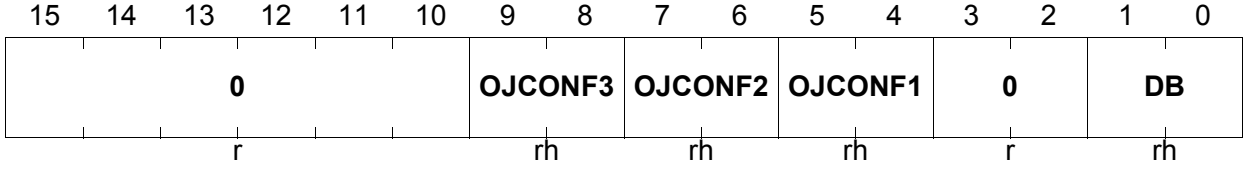
RSTSTAT1
Reset Status 1 Register **ESFR ($F0B4_H/5A_H$)** **Reset Value: $F000_H$**



Field	Bits	Type	Description
ESR0	[1:0]	rh	**$\overline{ESR0}$ Reset Status** 00_B The $\overline{ESR0}$ reset trigger was not relevant for the last reset 01_B Reserved 10_B The $\overline{ESR0}$ reset trigger was relevant for the last reset. Internal Application, and Application Resets were generated. 11_B The $\overline{ESR0}$ reset trigger was relevant for the last reset. Application Reset was generated.
ESR1	[3:2]	rh	**$\overline{ESR1}$ Reset Status** 00_B The $\overline{ESR1}$ reset trigger was not relevant for the last reset 01_B Reserved 10_B The $\overline{ESR1}$ reset trigger was relevant for the last reset. Internal Application, and Application Resets were generated. 11_B The $\overline{ESR1}$ reset trigger was relevant for the last reset. Application Reset was generated.
ESR2	[5:4]	rh	**$\overline{ESR2}$ Reset Status** 00_B The $\overline{ESR2}$ reset trigger was not relevant for the last reset 01_B Reserved 10_B The $\overline{ESR2}$ reset trigger was relevant for the last reset. Internal Application, and Application Resets were generated. 11_B The $\overline{ESR2}$ reset trigger was relevant for the last reset. Application Reset was generated.

Field	Bits	Type	Description
WDT	[7:6]	rh	**WDT Reset Status** 00_B The WDT reset trigger was not relevant for the last reset 01_B Reserved 10_B The WDT reset trigger was relevant for the last reset. Internal Application, and Application Resets were generated. 11_B The WDT reset trigger was relevant for the last reset. Application Reset was generated.
MP	[9:8]	rh	**MP Reset Status** 00_B The MP reset trigger was not relevant for the last reset 01_B Reserved 10_B The MP reset trigger was relevant for the last reset. Internal Application, and Application Resets were generated. 11_B The MP reset trigger was relevant for the last reset. Application Reset was generated.
STM	[13:12]	rh	**Power-on for DMP_M Reset Status** 00_B The power-on reset for DMP_M reset trigger was not relevant for the last reset 01_B The power-on reset for DMP_M reset trigger was not relevant for the last reset 10_B The power-on reset for DMP_M reset trigger was not relevant for the last reset 11_B The power-on reset for DMP_M reset trigger was relevant for the last reset
ST1	[15:14]	rh	**Power-on for DMP_1 Reset Status** 00_B The power-on reset for DMP_1 reset trigger was not relevant for the last reset 01_B The power-on reset for DMP_1 reset trigger was not relevant for the last reset 10_B The power-on reset for DMP_1 reset trigger was not relevant for the last reset 11_B The power-on reset for DMP_1 reset trigger was relevant for the last reset
0	[11:10]	r	**Reserved** Read as 0; should be written with 0.

RSTSTAT2
Reset Status 2 Register **ESFR ($F0B6_H/5B_H$)** **Reset Value: 0000_H**



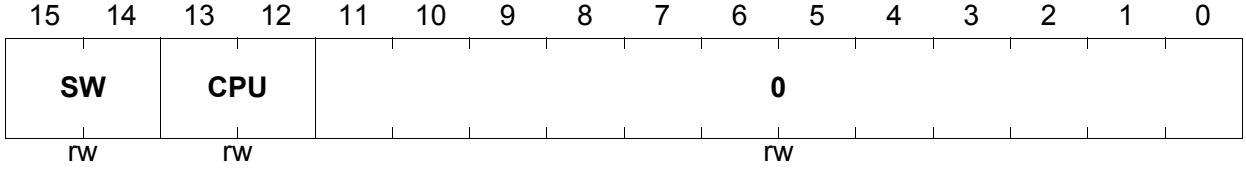
Field	Bits	Type	Description
DB	[1:0]	rh	**Debug Reset Status** 00_B The DB reset trigger was not relevant for the last reset 01_B The DB reset trigger was not relevant for the last reset 10_B The DB reset trigger was not relevant for the last reset 11_B The DB reset trigger was relevant for the last reset
OJCONF1	[5:4]	rh	**OJCONF1 Reset Status** 00_B The OJCONF1 reset trigger was not relevant for the last reset 01_B The OJCONF1 reset trigger was not relevant for the last reset 10_B The OJCONF1 reset trigger was not relevant for the last reset 11_B The OJCONF1 reset trigger was relevant for the last reset. Debug Reset was generated.
OJCONF2	[7:6]	rh	**OJCONF2 Reset Status** 00_B The OJCONF2 reset trigger was not relevant for the last reset 01_B The OJCONF2 reset trigger was not relevant for the last reset 10_B The OJCONF2 reset trigger was relevant for the last reset. Internal Application, and Application Resets were generated. 11_B The OJCONF2 reset trigger was not relevant for the last reset

Field	Bits	Type	Description
OJCONF3	[9:8]	rw	**OJCONF3 Reset Status** 00_B The OJCONF3 reset trigger was not relevant for the last reset 01_B The OJCONF3 reset trigger was not relevant for the last reset 10_B The OJCONF3 reset trigger was not relevant for the last reset 11_B The OJCONF3 reset trigger was relevant for the last reset. Application Reset was generated.
0	[3:2], [15:10]	r	**Reserved** Read as 0; should be written with 0.

6.3.7.2 Configuration Registers

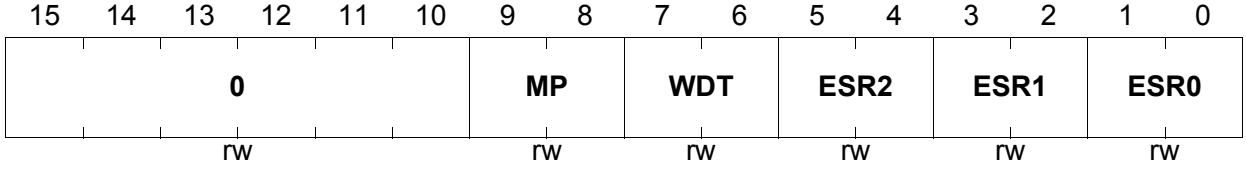
These registers allow the behavioral configuration for the various reset trigger sources.

RSTCON0
Reset Configuration 0 Register ESFR ($F0B8_H/5C_H$) **Reset Value: 0000_H**



Field	Bits	Type	Description
CPU	[13:12]	rw	**CPU Reset Type Selection** This bit field defines which reset types are generated by a CPU reset request trigger. 00_B No reset is generated 01_B Reserved, do not use this combination 10_B Internal Application, and Application Resets are generated 11_B Application Reset is generated
SW	[15:14]	rw	**Software Reset Type Selection** This bit field defines which reset types are generated by a software reset request trigger. 00_B No reset is generated 01_B Reserved, do not use this combination 10_B Internal Application, and Application Resets are generated 11_B Application Reset is generated
0	[11:0]	rw	**Reserved** Must be written with reset value 0.

RSTCON1
Reset Configuration 1 Register ESFR ($F0BA_H/5D_H$) **Reset Value: 0002_H**

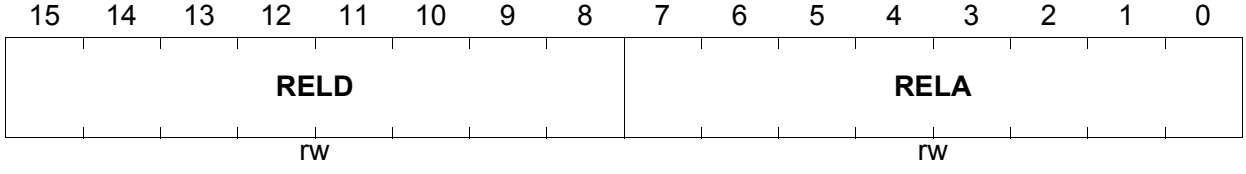


Field	Bits	Type	Description
ESR0	[1:0]	rw	**ESR0 Reset Type Selection** This bit field defines which reset types are generated by a ESR0 reset request trigger. 00_B No reset is generated 01_B Reserved, do not use this combination 10_B Internal Application, and Application Resets are generated 11_B Application Reset is generated
ESR1	[3:2]	rw	**ESR1 Reset Type Selection** This bit field defines which reset types are generated by a ESR1 reset request trigger. 00_B No reset is generated 01_B Reserved, do not use this combination 10_B Internal Application, and Application Resets are generated 11_B Application Reset is generated
ESR2	[5:4]	rw	**ESR2 Reset Type Selection** This bit field defines which reset types are generated by a ESR2 reset request trigger. 00_B No reset is generated 01_B Reserved, do not use this combination 10_B Internal Application, and Application Resets are generated 11_B Application Reset is generated

Field	Bits	Type	Description
WDT	[7:6]	rw	**WDT Reset Type Selection** This bit field defines which reset types are generated by a WDT reset request trigger. 00_B No reset is generated 01_B Reserved, do not use this combination 10_B Internal Application, and Application Resets are generated 11_B Application Reset is generated
MP	[9:8]	rw	**MP Reset Type Selection** This bit field defines which reset types are generated by a MP reset request trigger. 00_B No reset is generated 01_B Reserved, do not use this combination 10_B Internal Application, and Application Resets are generated 11_B Application Reset is generated
0	[15:10]	rw	**Reserved** Should be written with 0.

RSTCNTCON

Reset Counter Control RegisterESFR ($F1B2_H/D9_H$) **Reset Value: $0A0A_H$**

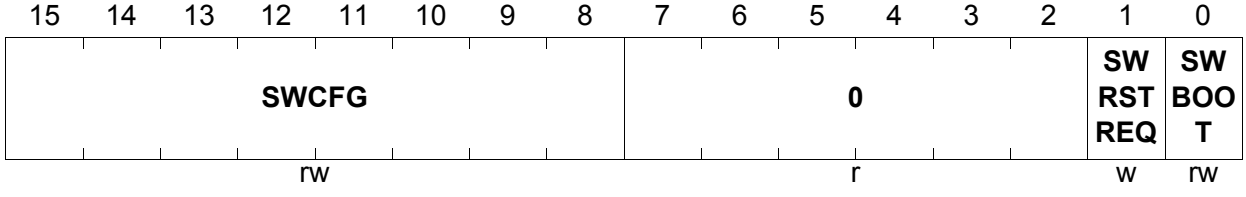


Field	Bits	Type	Description
RELA	[7:0]	rw	**Application Reset Counter Reload Value** This bit field defines the reload value of RSTCNTA. This value is always used when counter RSTCNTA is started. This counter value is used for Internal Application, and Application Resets. In case of an ESRx reset the counter value must be not less than the reset value.
RELD	[15:8]	rw	**Debug Reset Counter Reload Value** This bit field defines the reload value of RSTCNTD. This value is always used when counter RSTCNTD is started. This counter value is used for the Debug Reset. In case of an ESRx reset the counter value must be not less than the reset value.

Software Reset Control Register

This register controls the software reset operation.

SWRSTCON
Software Reset Control RegisterESFR ($F0AE_H/57_H$) **Reset Value: 0000_H**



Field	Bits	Type	Description
SWBOOT	0	rw	**Software Boot Configuration Selection** 0_B Bit field STSTAT.HWCFG is not updated with the content of SWCFG upon an Application Reset 1_B Bit field STSTAT.HWCFG is updated with the content of SWCFG upon an Application Reset
SWRSTREQ	1	w	**Software Reset Request** 0_B No software reset is requested 1_B A software reset request trigger is generated
SWCFG	[15:8]	rw	**Software Boot Configuration** A valid software boot configuration (also different from the external applied hardware configuration) can be specified with these bits. The configuration encoding is equal to the HWCFG encoding in register STSTAT.
0	[7:2]	r	**Reserved** Read as 0; should be written with 0.

6.4 External Service Request (ESR) Pins

The $\overline{\text{ESR}}$ pins serve as multi-functional pins for an amount of different options:

- Act as reset trigger input
- Act as reset output
- Act as trap input
- Act as trigger input for the GSC
- Independent pad configuration

6.4.1 General Operation

Each $\overline{\text{ESR}}$ pin is equipped with an edge detection that allows the selection of the edges used as triggers. One, both, or no edge can be selected via bit field ESRCFGx.SEDCON if a clock is active. Additionally, there is a digital (3-stage median) filter (DF) to suppress spikes. The signal at $\overline{\text{ESRx}}$ pin has to be held at the active signal level for at least 2 system clock cycles (f_{SYS}) in order to generate a trigger. The digital filter can be disabled by clearing bit ESRCFGx.DFEN.

Each $\overline{\text{ESRx}}$ pin can be individually configured.

If an $\overline{\text{ESR}}$ trigger is generated please note that triggers for all purposes (reset, trap, GSC, and non SCU module functions) are generated. If some of the actions resulting out of such a trigger should not occur this has to be disabled by each feature for its own.

The pins that should be used as trigger input for an ESR operation have to be configured as input pin.

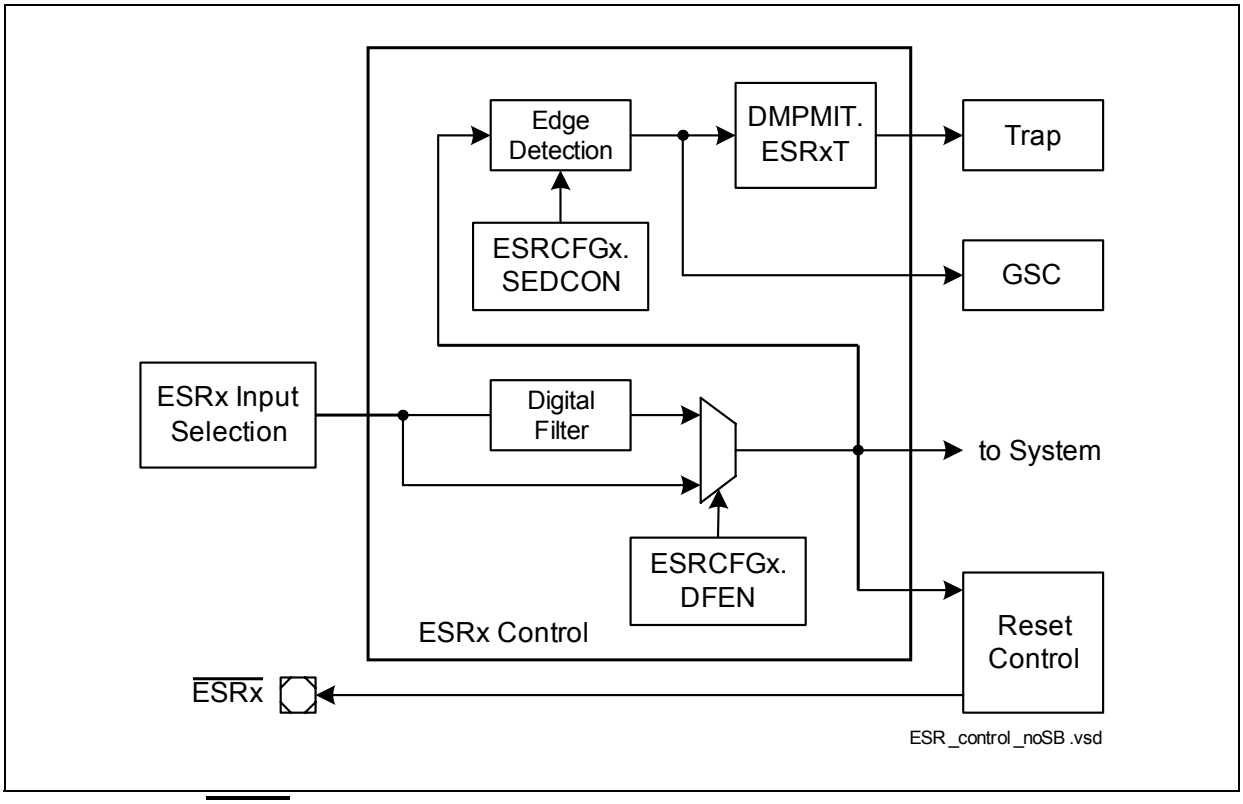


Figure 6-15 $\overline{\text{ESRx}}$ Control

Up to three $\overline{\text{ESR}}$ pins ($\overline{\text{ESR0}}$/$\overline{\text{ESR1}}$/$\overline{\text{ESR2}}$) are available. The availability of pins $\overline{\text{ESR1}}$ and $\overline{\text{ESR2}}$ is device and package dependent and is described in the data sheet.

6.4.1.1 $\overline{\text{ESR}}$ as Reset Input

The pins $\overline{\text{ESR}}$x can serve as an external reset input as well as a reset output (open drain) for Internal Application and Application Resets. Additionally several GPIO pad triggers that can be enabled additionally via register ESREXCONx interfere with the ESR pin function. GPIO and $\overline{\text{ESR}}$ pin triggers can be enabled/disabled individually and are combined for the reset trigger generation. For more information about the reset system see **Chapter 6.3**.

Note: The reset output is only asserted for the duration the reset counter RSTCNTA is active. During a possible reset extension the reset output is not longer asserted.

6.4.1.2 $\overline{\text{ESR}}$ as Reset Output

If pin $\overline{\text{ESR}}$x is enabled as reset output and the input level is low while the output stage is disabled (indicating that it is still driven low externally), the reset circuitry holds the chip in reset until a high level is detected on $\overline{\text{ESR}}$x. The internal output stage drives a low level during reset only while RSTCNTA is active. It deactivates the output stage when the time defined by RSTCNTCON.RELA has passed. For more information about the reset system see **Chapter 6.3**.

6.4.1.3 $\overline{\text{ESR}}$ as Trap Trigger

The $\overline{\text{ESR}}$ can request traps. The control mechanism if and which trap is requested is located in the trap control logic. For more information see **Chapter 6.12**.

6.4.1.4 $\overline{\text{ESR}}$ as Trigger Input for the GSC

The $\overline{\text{ESR}}$ can be used to request a change in the Control Mode. For more information see **Chapter 6.6**.

6.4.1.5 Pad Configuration for $\overline{\text{ESR}}$ Pads

The configuration is selected via bit field ESRCFGx.PC.

The pad functionality control can be configured independently for each pin, comprising:

- A selection of the driver type (open-drain or push-pull)
- An enable function for the output driver (input and/or output capability)
- An enable function for the pull-up/down resistance

The following table defines the coding of the bit fields PC in registers ESRCFG0, ESRCFG1, and ESRCFG2.

Note: The coding is the same as for the port register bit fields Pn_IOCRx.PC.

Table 6-6 PC Coding

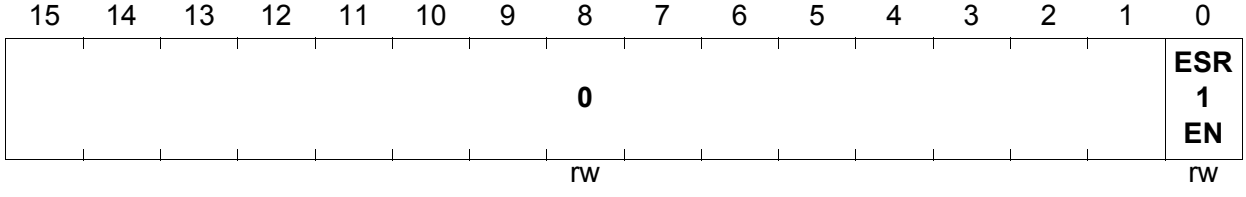
PCx[3:0]	Selected Pull-up/Pull-down / Selected Output Function	I/O	Output Characteristics
0000_B	No pull device activated	Input is not inverted, the input stage is active in power-down mode	
0001_B	Pull-down device activated		
0010_B	Pull-up device activated		
0011_B	No pull device activated		
0100_B	No pull device activated	Input is inverted, the input stage is active in power-down mode	
0101_B	Pull-down device activated		
0110_B	Pull-up device activated		
0111_B	No pull device activated		
1000_B	Output of ESRCFGx.OUT	Output, the input stage is not inverted and active in power-down mode	Push-pull
1001_B	Output of ESRCFGx.OUT		
1010_B	Output drives a 0 for an Internal Application Reset, a 1 otherwise.		
1011_B	Output drives a 0 for an Application Reset, a 1 otherwise.		
1100_B	Output of ESRCFGx.OUT		Open-drain, a pull-up device is activated while the output is not driving a 0
1101_B	Output of ESRCFGx.OUT		
1110_B	Output drives a 0 for an Internal Application Reset		
1111_B	Output drives a 0 for an Application Reset		

6.4.2 $\overline{\text{ESR}}$ Control Registers

6.4.2.1 Configuration Registers

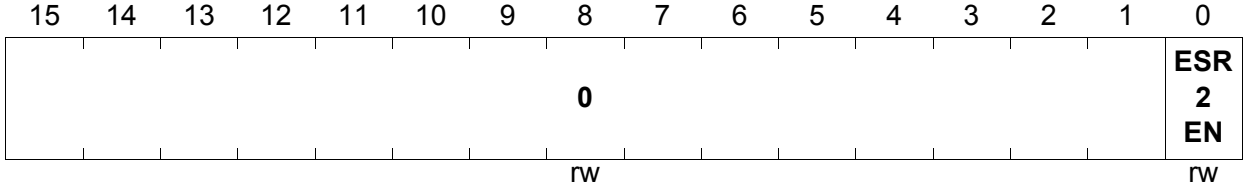
$\overline{\text{ESR}}$ External Control Register

ESREXCON1
ESR1 External Control Register SFR ($FF32_H/99_H$) **Reset Value: 0001_H**



Field	Bits	Type	Description
ESR1EN	0	rw	**$\overline{\text{ESR1}}$ Pin Enable** This bit enables/disables the $\overline{\text{ESR1}}$ pin for the activation of all $\overline{\text{ESR1}}$ related actions. 0_B The input from pin $\overline{\text{ESR1}}$ is disabled 1_B The input from pin $\overline{\text{ESR1}}$ is enabled
0	[15:1]	rw	**Reserved** Must be written with reset value 0.

ESREXCON2
ESR2 External Control Register SFR ($FF34_H/9A_H$) **Reset Value: 0001_H**



Field	Bits	Type	Description
ESR2EN	0	rw	**$\overline{\text{ESR2}}$ Pin Enable** This bit enables/disables the $\overline{\text{ESR2}}$ pin for the activation of all $\overline{\text{ESR2}}$ related actions. 0_B The input from pin $\overline{\text{ESR2}}$ is disabled 1_B The input from pin $\overline{\text{ESR2}}$ is enabled
0	[15:1]	rw	**Reserved** Must be written with reset value 0.

$\overline{\text{ESR}}$ Configuration Register

The $\overline{\text{ESR}}$ configuration registers contains bits required for the behavioral control of the $\overline{\text{ESR}}$ pins.

ESRCFG0
ESR0 Configuration Register **ESFR ($F100_H/80_H$)** **Reset Value: $000E_H$**
ESRCFG1
ESR1 Configuration Register **ESFR ($F102_H/81_H$)** **Reset Value: 0002_H**
ESRCFG2
ESR2 Configuration Register **ESFR ($F104_H/82_H$)** **Reset Value: 0002_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0					0		SEDCON		IN	OUT	DF EN	PC			
r					rw		rw		rh	rh	rw	rw			

Field	Bits	Type	Description
PC	[3:0]	rw	**Pin Control of $\overline{\text{ESRx}}$** This bit field controls the behavior of the associated $\overline{\text{ESRx}}$ pin. The coding is described in **Table 6-6**.
DFEN	4	rw	**Digital Filter Enable** This bit defines if the 3-stage median filter of the $\overline{\text{ESRx}}$ is used or bypassed. 0_B The filter is bypassed 1_B The filter is used
OUT	5	rh	**Data Output** This bit can be used as output value for the associated $\overline{\text{ESRx}}$ pin. 0_B If selected, the output level is 0 1_B If selected, the output level is 1 This bit is controlled via bit field ESRDAT.MOUTx.
IN	6	rh	**Data Input** This bit monitors the input value at the associated $\overline{\text{ESRx}}$ pin.

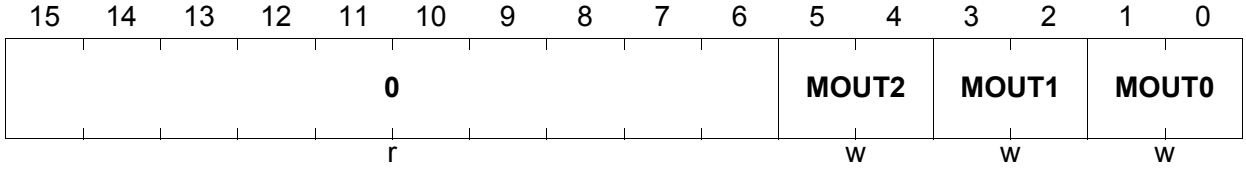
Field	Bits	Type	Description
SEDCON	[8:7]	rw	**Synchronous Edge Detection Control** This bit field defines the edges that lead to an $\overline{\text{ESRx}}$ trigger of the synchronous path. 00_B No trigger is generated 01_B A trigger is generated upon a raising edge 10_B A trigger is generated upon a falling edge 11_B A trigger is generated upon a raising AND falling edge
0	[10:9]	rw	**Reserved** Must be written with reset value 00_B.
0	[15:11]	r	**Reserved** Read as 0; should be written with 0.

6.4.3 ESR Data Register

6.4.3.1 ESRDAT

The $\overline{\text{ESR}}$ data register contains bits required if $\overline{\text{ESRx}}$ are used as data ports.

ESRDAT
ESR Data Register **ESFR ($F106_H/83_H$)** **Reset Value: 0000_H**



Field	Bits	Type	Description
MOUT0	[1:0]	w	**Modification of ESRCFG0.OUT** Writing to this bit field can modify the content of bit ESRCFG0.OUT for $\overline{\text{ESR0}}$. It always reads 0. 00_B Bit ESRCFG0.OUT is unchanged 01_B Bit ESRCFG0.OUT is set 10_B Bit ESRCFG0.OUT is cleared 11_B Reserved, do not use this combination
MOUT1	[3:2]	w	**Modification of ESRCFG1.OUT** Writing to this bit field can modify the content of bit ESRCFG1.OUT for $\overline{\text{ESR1}}$. It always reads 0. 00_B Bit ESRCFG1.OUT is unchanged 01_B Bit ESRCFG1.OUT is set 10_B Bit ESRCFG1.OUT is cleared 11_B Reserved, do not use this combination
MOUT2	[5:4]	w	**Modification of ESRCFG2.OUT** Writing to this bit field can modify the content of bit ESRCFG2.OUT for $\overline{\text{ESR2}}$. It always reads 0. 00_B Bit ESRCFG2.OUT is unchanged 01_B Bit ESRCFG2.OUT is set 10_B Bit ESRCFG2.OUT is cleared 11_B Reserved, do not use this combination
0	[15:6]	w	**Reserved** Read as 0; should be written with 0.

6.5 Power Supply and Control

The XE16x can run from a single external power supply. The core supply voltages can be generated by on-chip Embedded Voltage Regulators (EVRs).

Power Domains

The I/O part is divided in two parts DMP_A and DMP_B. DMP_A contains all ADC related I/Os and DMP_B the remaining system and communication I/Os.

The major part of the on-chip logic is located in an independent core power domain (DMP_1). A second power domain (DMP_M), marked grey in the figure below, controls important device infrastructure plus a Standby RAM (SBRAM).

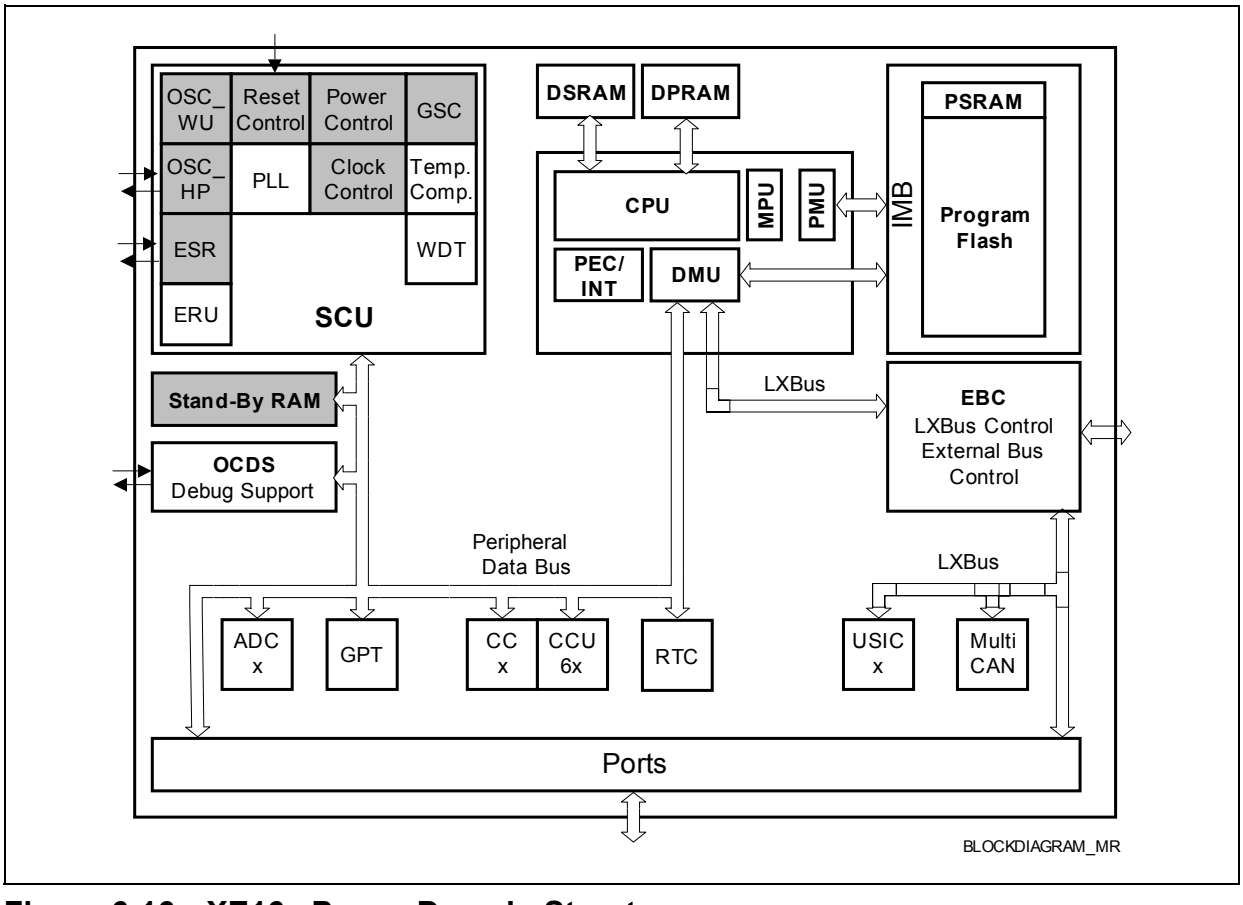


Figure 6-16 XE16x Power Domain Structure

Power Supply and Control Functions

The power supply and control is divided into following parts:

- monitoring of the supply voltage
- controlling and adjusting the supply voltage

The supply voltage of pad IO domain for system and communication I/Os (power domain DMP_B) is monitored by a Supply WatchDog (SWD, see **Chapter 6.5.1**).

The core voltage for each of the two core supply domains is supervised by a separate Power Validation Circuit (PVC) that provides two monitoring levels. Each monitoring level can request an interrupt (e.g. power-fail warning) or a reset depending on the voltage level. A PVC is used to detect under voltage due to an external short (see **Chapter 6.5.2**).

By controlling the regulator, the core power can be switched off to save the leakage current (see **Chapter 6.5.3**).

Table 6-7 XE16x Power Domains Supply and Control

Power Domain	Supply Source	Supply Voltage [V]	Supply Checked by
Pad IO domain (DMP_B)	External supply	V_{DDPB}: 3.0 … 5.5 typ See data sheet	SWD
ADC IO domain (DMP_A)	External supply	V_{DDPA}: 3.0 … 5.5 typ See data sheet	-
Core domain (DMP_M and DMP_1)	EVR_M EVR_1	V_{DDIM}, V_{DDI1}: 1.5 typ See data sheet	PVC_1, PVC_M

6.5.1 Supply Watchdog (SWD)

The supply voltage of the pad I/O domain for systems and communication I/Os (DMP_B) is monitored to validate the overall power supply. The external supply voltage is monitored for following purposes:

- POR
  Detecting the ramp-up of the external supply voltage, so the device can be started without requiring an external power-on reset ($\overline{\text{PORST}}$).
- Brown-out
  Detecting the ramp-down of the external supply voltage, so the device can be brought into a save state without requiring an external power-on reset ($\overline{\text{PORST}}$).
- Monitoring the external power supply allows the usage of a low-cost regulator without additional status signals (standard 3-pin device).
- Guarantee that the supply voltage for the EVRs is sufficient to generate a valid core voltage under every operating condition

Feature list

The following list is a summary of the SWD functions.

- Trigger a power-on reset whenever the supply falls and as long as the supply remains below V_{VAL}
- Two completely independent threshold levels and comparators
- 16 selectable threshold levels
- Power Saving Mode (only V_{VAL} detection active)

Operating the SWD

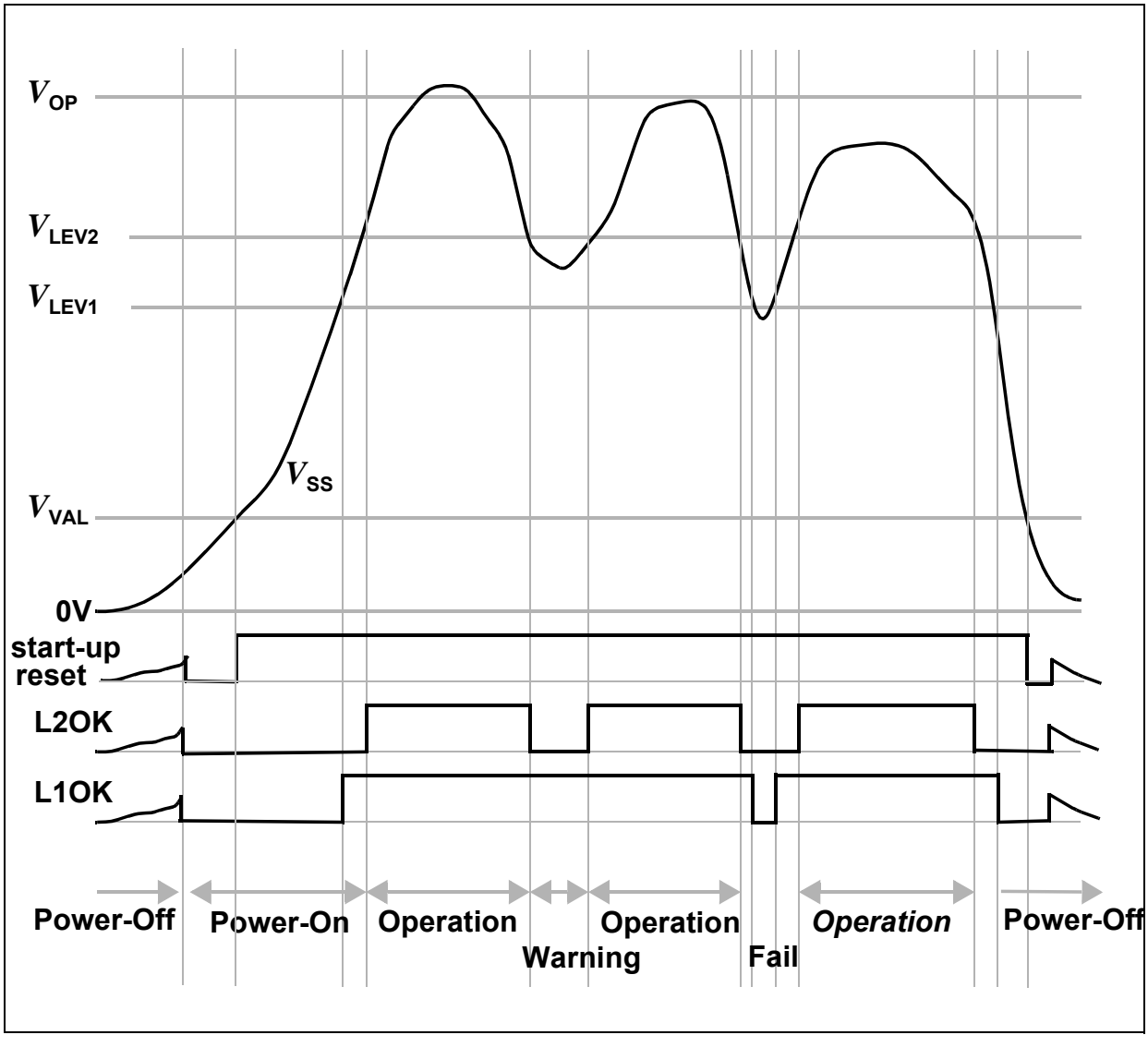


Figure 6-17 SWD Power Validation Example

The lower fix threshold V_{VAL} defines the absolute minimum operation voltage for the IO domain. If V_{VAL} has not been reached the device is held in reset. When V_{DDPB} raises above V_{VAL}, bit **SWDCON1**.PON is set.

Note: The physical value for V_{VAL} can found in the XE16x data sheet.

The SWD provides two adjustable threshold levels (LEV1 and LEV2) that can be individually programmed, via **SWDCON0**.LEV1V and SWDCON0.LEV2V, and deliver a compare value each. The two compare results can be monitored via bits SWDCON0.L1OK and SWDCON0.L2OK. A reset or interrupt request can be generated while the voltage level is below or equal/above the configured level of a threshold. If an

action and which action is triggered by each threshold can be configured via bit field SWDCON0.LxACON and bit fieldSWDCON0.LxALEV (x = 1,2).

The SWD control (programming of the threshold levels) is done by software only.

With these features, an external supply watchdog, e.g. integrated in some external VR, can be replaced. It detects the minimum specified supply voltage level and can be configured to monitor other voltage levels.

Note: If the $\overline{PORST}$ pin is used it has the same functionality as the SWD.

Power-Saving Mode of the SWD

The two configurable thresholds can be disabled if not needed. This is called the SWD Power Saving Mode. The minimum operating voltage detection (POR/Brown-out detection) can not be disabled and it is always active. The SWD Power Saving Mode is entered by setting bit **SWDCON1**.POWENSET and exit by setting bit SWDCON1.POWENCLR. If the SWD Power Saving Mode is active is indicated by bit SWDCON1.POWEN.

Note: The reset request and interrupt request action should be switched off before entering power-save mode.

6.5.1.1 SWD Control Registers

The following registers are the software interface for the SWD.

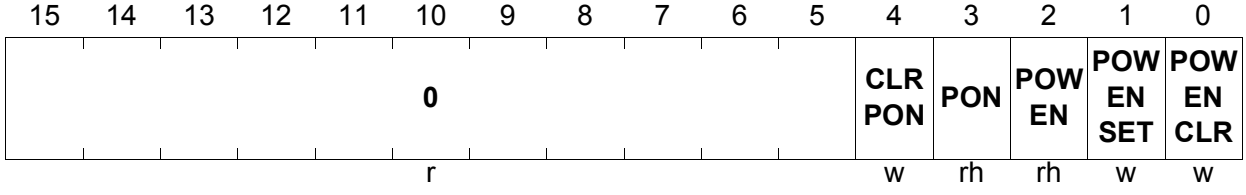
SWDCON0
SWD Control 0 Register **ESFR ($F080_H/40_H$)** **Reset Value: 0941_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
L2 A LEV	L2ACON		L2 OK	LEV2V				L1 A LEV	L1ACON		L1 OK	LEV1V			
rw	rw		rh	rw				rw	rw		rh	rw			

Field	Bits	Type	Description
LEV1V	[3:0]	rw	**Level Threshold 1 Voltage** This bit field defines the voltage level that is used as threshold 1 check level. The values of the level thresholds are listed in the data sheet.
L1OK	4	rh	**Level Threshold 1 Check Result** 0_B The supply voltage is below the Level Threshold 1 voltage LEV1V 1_B The supply voltage is equal or above the Level Threshold 1 voltage LEV1V
L1ACON	[6:5]	rw	**Level Threshold 1 Action Control** This bit field defines which actions are requested if the supply voltage comparison matches the action level L1ALEV. Following actions can be requested: 00_B No action is requested 01_B An interrupt is requested 10_B A reset is requested 11_B A reset and an interrupt are requested
L1ALEV	7	rw	**Level Threshold 1 Action Level** 0_B When the supply voltage is below the Level Threshold 1 voltage LEV1V the actions configured by bit field L1ACON are requested 1_B When the supply voltage is equal or above the Level Threshold 1 voltage LEV1V the actions configured by bit field L1ACON are requested

Field	Bits	Type	Description
LEV2V	[11:8]	rw	**Level Threshold 2 Voltage** This bit field defines the voltage level that is used as check level threshold 2. The values of the level thresholds are listed in the data sheet.
L2OK	12	rh	**Level Threshold 2 Check Result** 0_B The supply voltage is below the Level Threshold 2 voltage LEV2V 1_B The supply voltage is equal or above the Level Threshold 2 voltage LEV2V
L2ACON	[14:13]	rw	**Level Threshold 2 Action Control** This bit field defines which actions are requested if the supply voltage comparison matches the action level L2ALEV. Following actions can be requested: 00_B No action is requested 01_B An interrupt is requested 10_B A reset is requested 11_B A reset and an interrupt are requested
L2ALEV	15	rw	**Level Threshold 2 Action Level** 0_B When the supply voltage is below the Level Threshold 2 voltage LEV2V the actions configured by bit field L2ACON are requested 1_B When the supply voltage is equal or above the Level Threshold 2 voltage LEV2V the actions configured by bit field L2ACON are requested

SWDCON1
SWD Control 1 Register **ESFR ($F082_H/41_H$)** **Reset Value: 0000_H**



Field	Bits	Type	Description
POWENCLR	0	w	**SWD Power Saving Mode Enable Clear** 0_B No action 1_B Bit POWEN is cleared
POWENSET	1	w	**SWD Power Saving Mode Enable Set** 0_B No action 1_B Bit POWEN is set
POWEN	2	rh	**SWD Power Saving Mode Enable** 0_B All SWD functions are enabled 1_B The SWD Power Saving Mode is enabled. Comparators are disabled.
PON	3	rh	**Power-On Status Flag** 0_B No power-on event occurred 1_B A power-on event occurred (V_{DDP} became greater than V_{VAL}).
CLRPON	4	w	**Clear Power-On Status Flag** 0_B No action 1_B Bit PON is cleared
0	[15:5]	r	**Reserved** Read as 0; should be written with 0.

6.5.2 Monitoring the Voltage Level of a Core Domain

A Power Validation Circuit (PVC) monitors the internal core supply voltage of a core domain. It can be configured to monitor two programmable independent voltage levels.

The voltage of the core domain is monitored by PVC_1 and PVC_M.

Feature list

The following list summarizes the features of a PVC.

- Two independent comparators
- Threshold levels selectable
- Shut-off, which disables the complete module
- Configurable action level

A PVC provides two adjustable threshold levels (LEV1 and LEV2) that can be individually programmed via PVCxCON0.LEV1V and PVCxCON0.LEV2V (x = M or 1)PVC1CON0.LEV1V and PVC1CON0.LEV2V. The current supply level of a domain is compared with the threshold values. The two compare results can be monitored via bits PVCxCON0.LEV1OK and PVCxCON0.LEV2OK (x = M or 1) PVC1CON0.LEV1OK and PVC1CON0.LEV2OK

A reset or interrupt request can be generated in case the core domain voltage level is below or equal / above the configured threshold level. An interrupt is requested if bit PVCxCON0.L1INTEN and / or PVCxCON0.L2INTEN (x = M or 1) PVC1CON0.L1INTEN and / or PVC1CON0.L2INTEN is set. A reset is requested if bit PVCxCON0.L1RSTEN and / or PVCxCON0.L2RSTEN (x = M or 1) PVC1CON0.L1RSTEN and / or PVC1CON0.L2RSTEN is set.

Note: For a single threshold both interrupt and reset request generation should not be enabled at the same time.

6.5.2.1 PVC Status and Control Registers

These registers are the software interface for PVC_1 and PVC_M.

PVC1CON0
PVC_1 Control Step 0 Register

ESFR ($F014_H$/$0A_H$) **Reset Value: 0504_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
L2 AS EN	L2 RST EN	L2 INT EN	L2 A LEV	LEV 2 OK	LEV2V			L1 AS EN	L1 RST EN	L1 INT EN	L1 A LEV	LEV 1 OK	LEV1V		
rw	rw	rw	rw	rh	rw			rw	rw	rrw	rw	rh	rw		

Field	Bits	Type	Description
LEV1V	[2:0]	rw	**Level Threshold 1 Voltage** This bit field defines the Level Threshold 1 that is compared with the DMP_1 core voltage. The values for the different configurations are listed in the data sheet.
LEV1OK	3	rh	**Level Threshold 1 Check Result** 0_B The core supply voltage of the DMP_1 is below Level Threshold 1 voltage LEV1V 1_B The core supply voltage of the DMP_1 is equal or above the Level Threshold 1 voltage LEV1V
L1ALEV	4	rw	**Level Threshold 1 Action Level** 0_B When the core supply voltage is below Level Threshold 1 voltage LEV1V the action configured by bits L1INTEN, L1RSTEN, and L1ASEN are requested 1_B When the core supply voltage is equal or above Level Threshold 1 voltage LEV1V the actions configured by bits L1INTEN, L1RSTEN, and L1ASEN are requested
L1INTEN	5	rw	**Level Threshold 1 Interrupt Request Enable** This bit defines if an interrupt request trigger is requested if the comparison check was successful. When a check is successful is defined via bit L1ALEV. 0_B No interrupt is requested 1_B An interrupt is requested

Field	Bits	Type	Description
L1RSTEN	6	rw	**Level Threshold 1 Reset Request Enable** This bit defines if a reset request trigger is requested if the comparison check was successful. When a check is successful is defined via bit L1ALEV. 0_B No reset is requested 1_B An reset is requested
L1ASEN	7	rw	**Level Threshold 1 Asynchronous Action Enable** This bit defines if asynchronous action can be performed if the comparison check was successful. When a check is successful is defined via bit L1ALEV. 0_B No asynchronous actions are performed 1_B Asynchronous actions can be performed
LEV2V	[10:8]	rw	**Level Threshold 2 Voltage** This bit field defines the level of threshold 2 that is compared with the DMP_1 core voltage.. The values for the different configurations are listed in the data sheet.
LEV2OK	11	rh	**Level Threshold 2 Check Result** 0_B The core supply voltage of the DMP_1 is below the Level Threshold 2 LEV2V 1_B The core supply voltage of the DMP_1 is equal or above the Level Threshold 2 LEV2V
L2ALEV	12	rw	**Level Threshold 2 Action Level** 0_B When the core supply voltage is below the Level Threshold 2 voltage LEV2V the action configured by bits L2INTEN, L2RSTEN, and L2ASEN are requested 1_B When the core supply voltage is equal or above the Level Threshold 2 voltage LEV2V the action configured by bits L2INTEN, L2RSTEN, and L2ASEN are requested
L2INTEN	13	rw	**Level Threshold 2 Interrupt Request Enable** This bit defines if an interrupt request trigger is requested if the comparison check was successful. When a check is successful is defined via bit L2ALEV. 0_B No interrupt is requested 1_B An interrupt is requested

Field	Bits	Type	Description
L2RSTEN	14	rw	**Level Threshold 2 Reset Request Enable** This bit defines if a reset request trigger is requested if the comparison check was successful. When a check is successful is defined via bit L2ALEV. 0_B No reset is requested 1_B An reset is requested
L2ASEN	15	rw	**Level Threshold 2 Asynchronous Action Enable** This bit defines if asynchronous action can be performed if the comparison check was successful. When a check is successful is defined via bit L2ALEV. 0_B No asynchronous actions are performed 1_B Asynchronous actions can be performed

PVCMCON0
PVC_M Control Step 0 Register

MEM ($F1E4_H$/--) **Reset Value: 0544_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
L2 AS EN	L2 RST EN	L2 INT EN	L2 A LEV	LEV 2 OK	LEV2V			L1 AS EN	L1 RST EN	L1 INT EN	L1 A LEV	LEV 1 OK	LEV1V		
rw	rw	rw	rw	rh	rw			rw	rw	rw	rw	rh	rw		

Field	Bits	Type	Description
LEV1V	[2:0]	rw	**Level Threshold 1 Voltage** This bit field defines the Level Threshold 1 that is compared with the DMP_M core supply voltage. The values for the different configurations are listed in the data sheet.
LEV1OK	3	rh	**Level Threshold 1 Check Result** 0_B The core supply voltage of the DMP_M is below Level Threshold 1 voltage LEV1V 1_B The core supply voltage of the DMP_M is equal or above the Level Threshold 1 voltage LEV1V
L1ALEV	4	rw	**Level Threshold 1 Action Level** 0_B When the core supply voltage is below Level Threshold 1 voltage LEV1V the action configured by bits L1INTEN, L1RSTEN, and L1ASEN are requested 1_B When the core supply voltage is equal or above Level Threshold 1 voltage LEV1V the actions configured by bits L1INTEN, L1RSTEN, and L1ASEN are requested
L1INTEN	5	rw	**Level Threshold 1 Interrupt Request Enable** This bit defines if an interrupt request trigger is requested if the comparison check was successful. When a check is successful is defined via bit L1ALEV. 0_B No interrupt is requested 1_B An interrupt is requested

Field	Bits	Type	Description
L1RSTEN	6	rw	**Level Threshold 1 Reset Request Enable** This bit defines if a reset request trigger is requested if the comparison check was successful. When a check is successful is defined via bit L1ALEV. 0_B No reset is requested 1_B An reset is requested
L1ASEN	7	rw	**Level Threshold 1 Asynchronous Action Enable** This bit defines if asynchronous action can be performed if the comparison check was successful. When a check is successful is defined via bit L1ALEV. 0_B No asynchronous actions are performed 1_B Asynchronous actions can be performed
LEV2V	[10:8]	rw	**Level Threshold 2 Voltage** This bit field defines the Level Threshold 2 that is compared with the DMP_M core supply voltage. The values for the different configurations are listed in the data sheet.
LEV2OK	11	rh	**Level Threshold 2 Check Result** 0_B The core supply voltage of the DMP_M is below Level Threshold 2 voltage LEV2V 1_B The core supply voltage of the DMP_M is equal or above the Level Threshold 2 voltage LEV2V
L2ALEV	12	rw	**Level Threshold 2 Action Level** 0_B When the core supply voltage is below the Level Threshold 2 voltage LEV2V the action configured by bits L2INTEN, L2RSTEN, and L2ASEN are requested 1_B When the core supply voltage is equal or above the Level Threshold 2 voltage LEV2V the action configured by bits L2INTEN, L2RSTEN, and L2ASEN are requested
L2INTEN	13	rw	**Level Threshold 2 Interrupt Request Enable** This bit defines if an interrupt request trigger is requested if the comparison check was successful. When a check is successful is defined via bit L2ALEV. 0_B No interrupt is requested 1_B An interrupt is requested

Field	Bits	Type	Description
L2RSTEN	14	rw	**Level Threshold 2 Reset Request Enable** This bit defines if a reset request trigger is requested if the comparison check was successful. When a check is successful is defined via bit L2ALEV. 0_B No reset is requested 1_B An reset is requested
L2ASEN	15	rw	**Level Threshold 2 Asynchronous Action Enable** This bit defines if asynchronous action can be performed if the comparison check was successful. When a check is successful is defined via bit L2ALEV. 0_B No asynchronous actions are performed 1_B Asynchronous actions can be performed

6.5.3 Controlling the Voltage Level of a Core Domain

The core powercan be controlled within certain limits. The voltage level is controlled by two **Embedded Voltage Regulator**s (EVR).

The power domain is controlled by both EVR_M and EVR_1.

6.5.3.1 Embedded Voltage Regulator

The main part of the device logic operates at a typical voltage level of 1.5 V. This supply voltage is generated by the Embedded Power Regulators (EVRs) out of the pad voltage. External buffer caps are required for stable regulation.

Feature list:

- Multiple core voltage levels including zero
- Core voltage generation based on a High Precision Bandgap
- External supply possible via capacitor-pin while EVR is switched-off
- Core current limit

The EVR configurations to select the desired voltage and reference pair are combined within EVR settings EVRxSETyyV (x = M or 1 and yy = 15). Each setting contains a bit field (VRSEL) to select the voltage level and reference and a bit field to fine-tune the voltage level (VLEV). One out of the possible settings is used to control each of the EVRs, but only in the allowed combinations for the two EVRs. The EVRsusea High Precision Bandgap (HP) as reference

The BG voltage of each setting can be adjusted to compensate application and environmental influences by the bit field EVRxSETyyV.VLEV. VLEV is set by default or trimmed by each device during production test to reach the default setting targets.

High Precision Bandgap (HP)

The HP bandgap of the system is used for following purposes:

- Provide a very stable reference for the two EVRs
- Provide an accurate reference for the flash memory. For more information see the flash memory description.

The HP bandgap can be enabled / disabled via the bit **EVRMCON1**.HPEN.

EVR Status and Control Registers

EVR1CON0
EVR_1 Control 0 Register **ESFR ($F088_H/44_H$)** **Reset Value: $DF20_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EVR DIS	**-**	**0**	**CC DIS**	**CCLEV**		**-**	**RES 1**	**0**		**RES0**			**0**		
rh	-	rw	rh	rw		-	rw	rw		rw			r		

Field	Bits	Type	Description
RES0	[5:3]	rw	**Reserved** Do not change this value when writing to this register.
0	[7:6]	rw	**Reserved** Must be written with reset value 00_B.
RES1	8	rw	**Reserved** Must be written with reset value 1_B.
CCLEV	[11:10]	rw	**Current Control Level** The values for the different configurations are listed in the data sheet.
CCDIS	12	rh	**Current Control Disable** 0_B The current control is enabled 1_B The current control is disabled This bit is updated by bit EVR1SETy.CCDIS.
0	13	rw	**Reserved** Must be written with reset value 0.
EVRDIS	15	rh	**EVR_1 Disable** 0_B The EVR_1 is enabled 1_B The EVR_1 is disabled This bit is updated by bit EVR1SETy.EVRDIS.
0	[2:0]	r	**Reserved** Read as 0; should be written with 0.

EVR1SET15VHP
EVR_1 Setting for 1.5 V HP Register

ESFR ($F09E_H/4F_H$) Reset Value: $001B_H$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EVR DIS	0		CC DIS	0		RES	0	VRSEL		VLEV					
rw	rw		rw	rw		rw	rw	rw		rw					

Field	Bits	Type	Description
VLEV	[5:0]	rw	**Voltage Level Adjust** This bit field adjusts the BG voltage and is trimmed by each device during production test to reach the default setting targets. Do not change this value when writing to this register.
VRSEL	[7:6]	rw	**Voltage Reference Selection** 00_B 15VHP - Full Voltage with high precision bandgap selected 01_B Reserved, do not use this combination 10_B Reserved, do not use this combination 11_B Reserved, do not use this combination *Note: The reset value should always be written to this bit field.*
RES	9	rw	**Reserved** Must be written with 1_B.
CCDIS	12	rw	**Current Control Disable** 0_B The current control is enabled 1_B The current control is disabled This bit updates bit EVR1CON0.CCDIS. *Note: Before switching off the current control the CCLEV setting in EVR1CON0 has to be set to 00_B.*
EVRDIS	15	rw	**EVR_1 Disable** 0_B The EVR_1 is enabled 1_B The EVR_1 is disabled This bit updates bit EVR1CON0.EVRDIS.

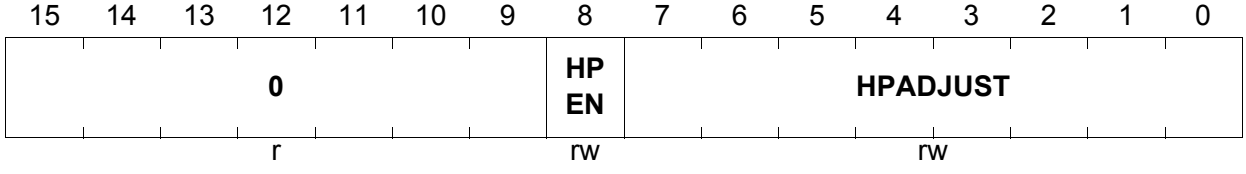
Field	Bits	Type	Description
0	8, [11:10], [14:13]	rw	**Reserved** Must be written with reset value 0.

EVRMCON0
EVR_M Control 0 Register **ESFR ($F084_H/42_H$)** **Reset Value: $0D20_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EVR DIS	0		CC DIS	CCLEV		-	RES 1	0		RES0			0		
rh	r		rh	rw		-	rw	rw		rw			r		

Field	Bits	Type	Description
RES0	[5:3]	rw	**Reserved** Do not change this value when writing to this register.
0	[7:6]	rw	**Reserved** Must be written with reset value 00_B.
RES1	8	rw	**Reserved** Must be written with 1.
CCLEV	[11:10]	rw	**Current Control Level** The values for the different configurations are listed in the data sheet.
CCDIS	12	rh	**Current Control Disable** 0_B The current control is enabled 1_B The current control is disabled This bit is updated by bit EVRMSETy.CCDIS.
EVRDIS	15	rh	**EVR_M Disable** 0_B The EVR_M is enabled 1_B The EVR_M is disabled This bit is updated by bit EVRMSETy.EVRDIS.
0	[2:0], 8 [14:13]	r	**Reserved** Read as 0; should be written with 0.

EVRMCON1
EVR_M Control 1 Register **ESFR ($F086_H/43_H$)** **Reset Value: 0101_H**



Field	Bits	Type	Description
HPADJUST	[7:0]	rw	**HP Bandgap Adjustment** This bit field is a device specific trimmvalue for the HP bandgap. Do not change this value when writing to this register.
HPEN	8	rw	**HP Bandgap Enable** 0_B The HP bandgap is disabled 1_B The HP bandgap is enabled
0	[15:9]	r	**Reserved** Read as 0; should be written with 0.

EVRMSET15VHP
EVR_M Setting for 1.5 V HP Register

ESFR ($F096_H/4B_H$) **Reset Value: $001B_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EVR DIS	0		CC DIS	0		RES 0	0	VRSEL		VLEV					
rw	rw		rw	rw		rw	rw	rw		rw					

Field	Bits	Type	Description
VLEV	[5:0]	rw	**Voltage Level Adjust** This bit field adjusts the BG voltage and is trimmed by each device during production test to reach the default setting targets. Do not change this value when writing to this register.
VRSEL	[7:6]	rw	**Voltage Reference Selection** 00_B 15VHP - Full Voltage with high precision bandgap selected 01_B Reserved, do not use this combination 10_B Reserved, do not use this combination 11_B Reserved, do not use this combination *Note: The reset value should always be written to this bit field.*
RES0	9	rw	**Reserved** Must always be written with 1_B.
CCDIS	12	rw	**Current Control Disable** 0_B The current control is enabled 1_B The current control is disabled This bit updates bit EVRMCON0.CCDIS. *Note: Before switching off the current control the CCLEV setting in EVRMCON0 has to be set to 00_B.*
EVRDIS	15	rw	**EVR_M Disable** 0_B The EVR_M is enabled 1_B The EVR_M is disabled This bit updates bit EVR1CON0.EVRDIS.

Field	Bits	Type	Description
0	8, [11:10], [14:13]	rw	**Reserved** Must be written with reset value 0.

6.5.3.2 Sources for Core Supply Voltage

The on-chip EVRs can generate the XE16x's core supply voltage from the (externally supplied) IO voltage.

Core Supply via On-chip EVRs

Generating the core supply voltage via the integrated EVRs is the preferred operating mode, because it saves an additional external voltage regulator. The integrated EVRs are fed from supply voltage V_{DDPB}.

Proper operation of the EVRs requires external buffer capacitances. Please refer to the respective Data Sheet for the recommended values. The current is delivered by the integrated pass devices.

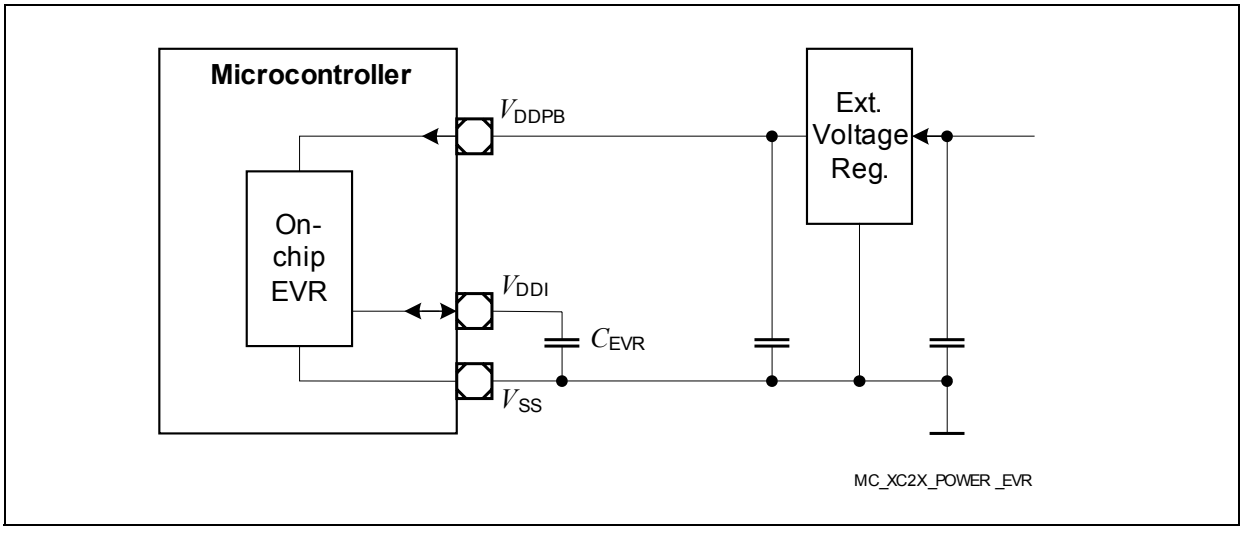


Figure 6-18 Selecting the EVR for Core Supply

Generating the core supply voltage with on-chip resources provides full control of power reduction modes, so the application can control and minimize the energy consumption of the XE16x using built-in mechanisms without requiring additional external circuitry.

6.5.4 Handling the Power System

Using the power system correctly is the key to power saving. Depending on the application different operating states can be defined in order to save maximal power. The XE16x supports following power saving mechanisms:

- Reduction of the system performance
  - the power consumption depends directly on the frequency of the system
  - the system performance is controlled with the clock operation mechanism
- Stopping single unused peripheral
  - a peripheral not needed for an application can be disabled
  - the module operation is controlled via register MOD_KSCCFG
- Stopping multiple unused peripherals
  - peripherals not needed for an application can be disabled
  - system peripheral operation is controlled via the Global State Controller (GSC)
- Stopping single unused analog parts
  - an analog part not needed for an application can be stopped
  - the operation is controlled via register either located in the SCU (PLL, clocks, PVCs, SWD, Temperature Compensation) or the ADC

6.6 Global State Controller (GSC)

Mode Control for the system peripherals provides besides power saving modes and the clock management an additional opportunity for configuring the system to the application needs.

Mode Control is described in detail in this chapter and is implemented by the Global State Controller (GSC). The GSC enables the user to configure one operating mode in a fast and easy way, reacting fast and explicit to needs of an application.

Feature Overview

The following issues are handled by the GSC:

- Control of peripheral clock operation
- Suspend control for debugging
- Arbitration between the different request sources

According to the requests coming from the OCDS, the SWD pre-warning detection or other blocks, the GSC does an internal prioritization. The result is forwarded as command request broadcast to all peripherals. The GSC internal prioritization scheme for the implemented request sources is shown in **Table 6-8**.

6.6.1 GSC Control Flow

The sequence begins when at least one request source asserts its trigger in order to request a mode change in the SoC. If several requests are pending there is an arbitration mechanism that treats this issue. Request triggers are not stored by the GSC, therefore a trigger source has to assert its trigger until the trigger is no longer valid or needed.

A request trigger is kept asserted as long as either the request is still pending or the resulting command of the request was entered and acknowledged by the system. The communication of the GSC and the peripherals is based on commands. Three different commands are defined resulting in three modes:

- Wake-up command: requests Normal Mode
- Clock-off command: requests Stop Mode
- Debug command: requests Suspend Mode

The specific behavior in these three modes is defined for each peripheral in its module register mod_KSCCFG.

6.6.1.1 Request Source Arbitration

The highest priority for the arbitration is zero (see **Table 6-8**).

Each system clock cycle a new arbitration round is started. The winner of an arbitration round requests the next command towards the SoC. Please note that winning an arbitration does not lead automatically to a new command raised. Only if currently no command is broadcast in the SoC a new command can be generated and broadcast. If

the winner of the arbitration round is the same request trigger as in the previous round or if no winner was detected no new command request is generated.

Table 6-8 Connection of the Request Sources

Request Source	Priority
OCDS exit	4
ESR0	5
ESR1	6
ESR2	7
WUT	8
ITC	9
GPT12E	10
SW1	11
SW2	12
OCDS entry	14

6.6.1.2 Generation of a New Command

When a new request trigger was detected and arbitrated a new command request is generated if currently no command request is broadcast that is not received by all slaves.

Table 6-9 Request Source and Command Request Coupling

Request Source	Command Description
OCDS exit	Wake-up; Normal Mode
ESR0	Wake-up; Normal Mode
ESR1	Wake-up; Normal Mode
ESR2	Clock-off; Stop Mode
WUT	Wake-up; Normal Mode
ITC	Wake-up; Normal Mode
GPT12E	Wake-up; Normal Mode
SW1	Wake-up; Normal Mode
SW2	Clock-off; Stop Mode
OCDS entry	Debug; Suspend Mode

6.6.1.3 Usage of Commands

The complete control mechanism for the different operation modes of the various slaves is divided into two parts:

- A central control and configuration part; the Global State Controller (GSC)
- One local control part in each slave; the Kernel State Controller (KSC)

Via the GSC either different hardware sources (e.g. the WUT or the OCDS) or the software can request the system to enter a specific mode. The parts that are affected by the mode can be pre-defined locally for each part via the KSC. For each command a specific reaction can be pre-configured in each KSC for each individual part.

Note: When a GSC mode request has been successfully entered, the GSC arbiter is open for any new request. In case a request occurs to enter the current mode, this request is pipelined and remains pending.
*It is recommended to request a command by a software trigger. In particular the clock-off command should be triggered by SW2. The usage of commands requested by hardware has to be done carefully. Only hardware resources requesting Normal Mode should be selected. If the software has detected a wake-up then pending mode change requests can be removed by clearing the bits of the selected sources in **GSCEN** and then enabling the bits in GSCEN again.*

6.6.1.4 Terminating a Request Trigger

A request trigger is no longer taken into account for the arbitration after the de-assertion of the request trigger, if it is not enabled or when its respective enable bit is cleared.

6.6.1.5 Suspend Control Flow

The suspend feature is controlled by the OCDS block. The GSC operates only as control and communication interface towards the system. The suspend feature is composed out of two requirements:

The mode that has to be entered when the Suspend Mode is requested.

The mode that has to be entered when the Suspend Mode is left.

The request to enter Suspend Mode is forwarded from the OCDS. When the Suspend Mode is requested the system is expected to be stopped as soon as possible in an idle state where no internal process is pending and in a way that this system state does not lead to any damage internally or externally and can also be left without any damage. Therefore all peripherals in the system are requested to enter a mode where the clock can be stopped. This is done by sending a debug command.

Leaving the Suspend Mode should serve the goal that debugging is a non-intrusive operation. Therefore leaving the Suspend Mode can not lead to only one dedicated system mode, instead it leads to the system mode the system left when it was requested to exit the Suspend Mode. The system mode is stored when a Suspend Mode request

is detected by the GSC and is used as target system mode when a leave Suspend Mode trigger is detected by the GSC.

6.6.1.6 Error Feedback for a Mode Transition

In case at least one peripheral reports an error the error flag in register GSCSTAT is set. If no error is currently detected upon a new assertion of a system mode by the GSC the error flag is cleared. To inform the system of this erroneous state an interrupt can be generated.

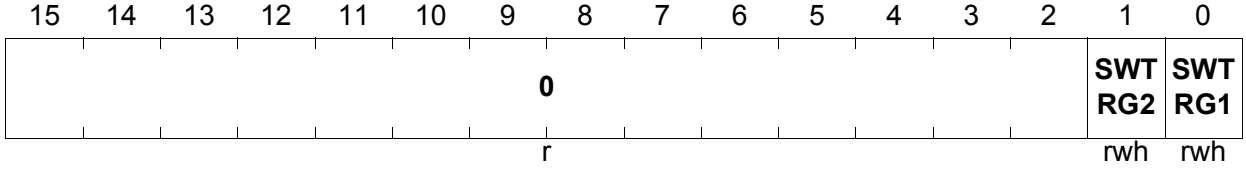
6.6.2 GSC Registers

6.6.2.1 GSC Control and Status Registers

The following register control and configure the behavior of the GSC.

GSCSWREQ
GSC Software Request Register

SFR ($FF14_H/8A_H$) **Reset Value: 0000_H**



Field	Bits	Type	Description
SWTRG1	0	rwh	**Software Trigger 1 (SW1)** 0_B No SW1 request trigger is generated 1_B A SW1 request trigger is generated This bit is automatically cleared if the SW1 request trigger wins the arbitration and was broadcast to the system.
SWTRG2	1	rwh	**Software Trigger 2 (SW2)** 0_B No SW2 request trigger is generated 1_B A SW2 request trigger is generated This bit is automatically cleared if the SW2 request trigger wins the arbitration and was broadcast to the system.
0	[15:2]	r	**Reserved** Read as 0; should be written with 0.

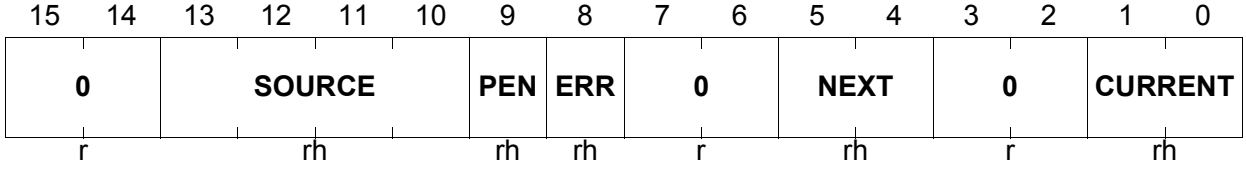
GSCEN
GSC Enable Register **SFR ($FF16_H/8B_H$)** **Reset Value: $7FFF_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	OCDSENEN	RES1	SW2EN	SW1EN	GPTEN	ITCEN	WUTEN	ESR2EN	ESR1EN	ESR0EN	OCDSEXEN	RES0			
r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw			

Field	Bits	Type	Description
RES0	[3:0]	rw	**Reserved** Must be written with 0000_B.
OCDSEXEN	4	rw	**OCDS Exit Request Trigger Enable** 0_B OCDS exit request trigger is not taken into account (disabled) 1_B OCDS exit request trigger is taken into account (enabled)
ESR0EN	5	rw	**ESR0 Request Trigger Enable** 0_B ESR0 request trigger is not taken into account (disabled) 1_B ESR0 request trigger is taken into account (enabled)
ESR1EN	6	rw	**ESR1 Request Trigger Enable** 0_B ESR1 request trigger is not taken into account (disabled) 1_B ESR1 request trigger is taken into account (enabled)
ESR2EN	7	rw	**ESR2 Request Trigger Enable** 0_B ESR2 request trigger is not taken into account (disabled) 1_B ESR2 request trigger is taken into account (enabled)
WUTEN	8	rw	**WUT Request Trigger Enable** 0_B WUT request trigger is not taken into account (disabled) 1_B WUT request trigger is taken into account (enabled)

Field	Bits	Type	Description
ITCEN	9	rw	**ITC Request Trigger Enable** 0_B ITC request trigger is not taken into account (disabled) 1_B ITC request trigger is taken into account (enabled)
GPTEN	10	rw	**GTP12E Request Trigger Enable** 0_B GPT12E request trigger is not taken into account (disabled) 1_B GPT12E request trigger is taken into account (enabled)
SW1EN	11	rw	**Software 1 Request Trigger Enable** 0_B SW1 request trigger is not taken into account (disabled) 1_B SW1 request trigger is taken into account (enabled)
SW2EN	12	rw	**Software 2 Request Trigger Enable** 0_B SW2 request trigger is not taken into account (disabled) 1_B SW2 request trigger is taken into account (enabled)
RES1	13	rw	**Reserved** Read as 1 after reset; returns the value that is written.
OCDSENEN	14	rw	**OCDS Entry Request Trigger Enable** 0_B OCDS entry request trigger is not taken into account (disabled) 1_B OCDS entry request trigger is taken into account (enabled) OCDS entry is the request source belonging to the according connector interface.
0	15	r	**Reserved** Read as 0; should be written with 0.

GSCSTAT
GSC Status Register **SFR ($FF18_H/8C_H$)** **Reset Value: $3C00_H$**



Field	Bits	Type	Description
CURRENT	[1:0]	rh	**Currently used Command** This bit field states the currently used system mode.
NEXT	[5:4]	rh	**Next to use Command** This bit field states the next to be used system mode.
ERR	8	rh	**Error Status Flag** This bit flags if with the last command that was broadcast was acknowledge with at least one error. This bit is automatically cleared when a new command is broadcast.
PEN	9	rh	**Command Pending Flag** This flag states if currently a command is pending or not. A command is pending after the broadcast as long as no all blocks acknowledge that they finished the operation requested by the command.

Field	Bits	Type	Description
SOURCE	[13:10]	rh	**Requesting Source Status** This bit field monitors the source that triggered the last request. 0000_B Reserved 0001_B Reserved 0010_B Reserved 0011_B Reserved 0100_B OCDS exit 0101_B $\overline{\text{ESR0}}$ 0110_B $\overline{\text{ESR1}}$ 0111_B ESR2 1000_B WUT 1001_B ITC 1010_B GPT12E 1011_B SW1 1100_B SW2 1101_B Reserved, do not use this combination 1110_B OCDS entry 1111_B Reserved, do not use this combination
0	[3:2], [7:6], [15:14]	r	**Reserved** Read as 0; should be written with 0.

6.7 Software Boot Support

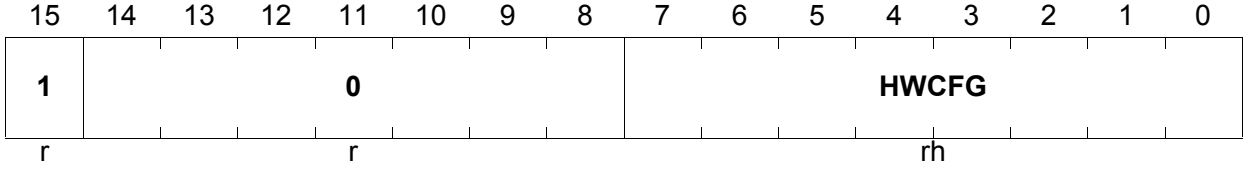
In order to determine the correct starting point of operation for the software a minimum of hardware support is required. As much as possible is done via software. Some decisions have to be done in hardware because they must be known before any software is operational.

6.7.1 Start-up Registers

6.7.1.1 Start-up Status Register

Register STSTAT contains the information required by the boot software to identify the different start-up settings that can be selected.

STSTAT
Start-up Status Register **MEM ($F1E0_H$/--)** **Reset Value: 8000_H**



Field	Bits	Type	Description
HWCFG	[7:0]	rh	**Hardware Configuration Setting** This bit field contains the value that is used by the boot software. This bit field is updated in case of an Application Reset with the content by register SWRSTCON.SWCFG if bit SWRSTCON.SWBOOT is set.
0	[14:8]	r	**Reserved** Read as 0; should be written with 0.
1	15	r	**Reserved** Read as 1; should be written with 1.

6.8 External Request Unit (ERU)

The External Request Unit (ERU) is a versatile event and pattern detection unit. Its major task is the **generation of interrupts based on selectable trigger events at different inputs**, e.g. to generate external interrupt requests if an edge occurs at an input pin.

The detected events can also be used by other modules to trigger or to gate module-specific actions, such as conversions of the ADC module.

6.8.1 Introduction

The ERU of the XE16x can be split in three main functional parts:

- 4 independent **Input Channels x** for input selection and conditioning of trigger or gating functions
- Event distribution: A **Connecting Matrix** defines the events of the Input Channel x that lead to a reaction of an Output Channel y.
- 4 independent **Output Channels y** for combination of events, definition of their effects and distribution to the system (interrupt generation, ADC conversion triggers)

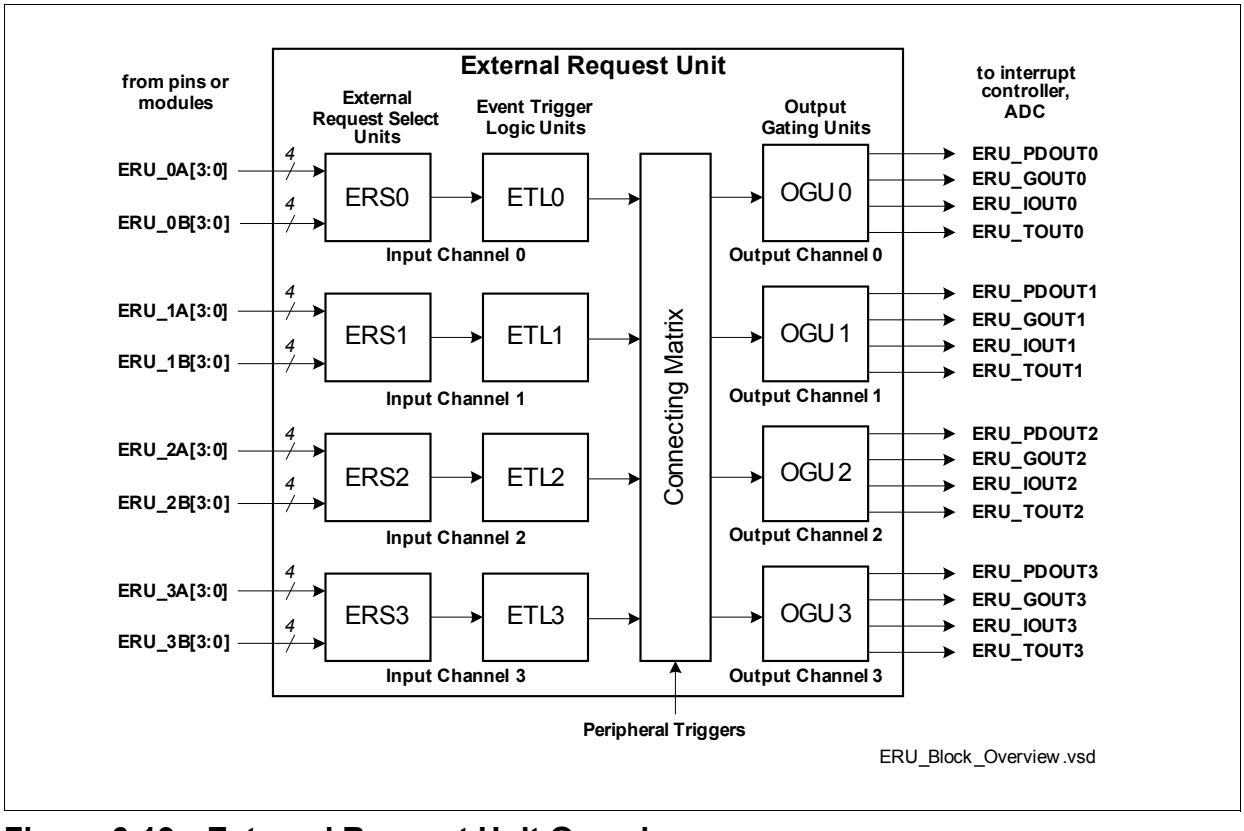


Figure 6-19 External Request Unit Overview

These tasks are handled by the following building blocks:

- An **External Request Select Unit (ERSx)** per Input Channel allows the selection of one out of two or a logical combination of two input signals (ERU_xA, ERU_xB) to a

common trigger. For each of these two signals, an input vector of 4 possible inputs is available (e.g. the actual input ERU_xA can be selected from one of the ERU inputs ERU_xA[3:0], similar for ERU_xB).

- An **Event Trigger Logic (ETLx)** per Input Channel allows the definition of the transition (edge selection, or by software) that lead to a trigger event and can also store this status. Here, the input levels of the selected signals are translated into events (event detected = event flag becomes set, independent of the polarity of the original input signals).
- The **Connecting Matrix** distributes the events and status flags generated by the Input Channels to the Output Channels. Additionally, some Peripheral Trigger signals from other modules (e.g. CC2) are made available and can be combined with the triggers generated by the Input Channels of the ERU.
- An **Output Gating Unit (OGUy)** per Output Channel that combines the available trigger events and status information from the Input Channels. An event of one Input Channel can lead to reactions of several Output Channels, or also events of several Input Channels can be combined to a reaction of one Output Channel (pattern detection).
  Different types of reactions are possible, e.g. interrupt generation (based on signals ERU_IOUTy), triggering of ADC conversions (based on signals ERU_TOUTy), or gating of ADC conversions (based on signals ERU_GOUTy).

The ERU is controlled by a number of registers, shown in **Figure 6-20**, and described in **Section 6.8.8**.

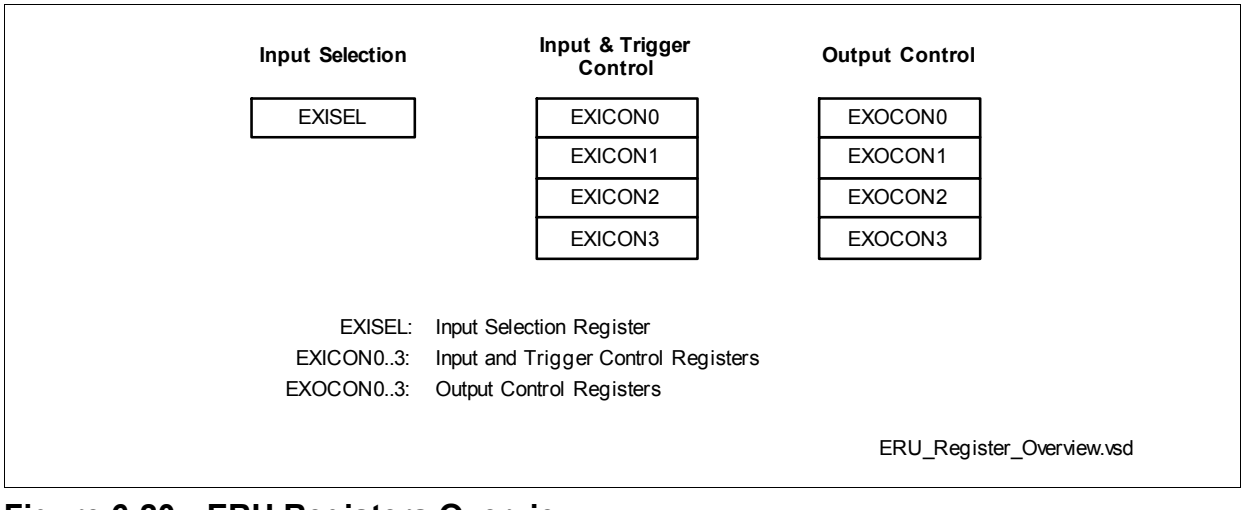


Figure 6-20 ERU Registers Overview

6.8.2 ERU Pin Connections

Figure 6-21 shows the ERU input connections, either directly with pins or via communication modules, such as USIC or MultiCAN. These communication modules provide their input signals (e.g. CAN receive input, or USIC data, clock, or control inputs) that have been selected in these modules. With this structure, the number of possible input pins is significantly increased, because not only the selection capability of the ERU is used, but also the selection capability of the communication modules.

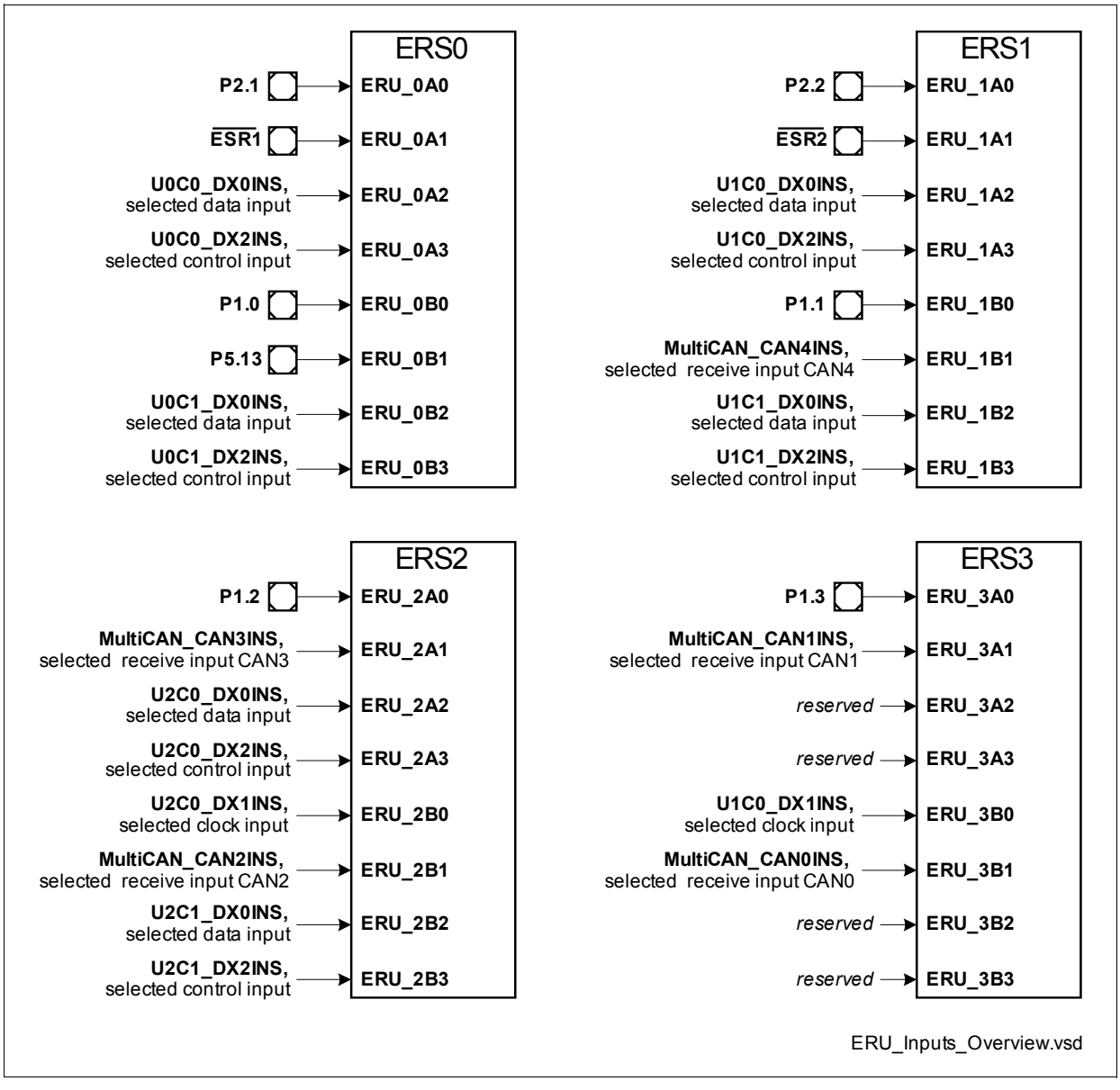


Figure 6-21 ERU Inputs Overview

The inputs to the ERU can be selected from a large number of input signals. While some of the inputs come directly from a pin, other inputs use signals from various peripheral

modules, such as the USIC (signals named with prefix UxCy to indicate which the communication channel) and the MultiCAN modules. These signals are input signals from the pin that has been selected as input for a USIC or MultiCAN function. The selection of the input is made within the respective USIC or MultiCAN module.

Usually, such signals would be selected for an ERU function when the input function to the USIC or MultiCAN module is not used otherwise, or the module is not used at all. However, it is also possible to select a input which is actually needed in a USIC or MultiCAN module, and to use it also in the ERU to provide for certain trigger functions, eventually combined with other signals (e.g. to generate an interrupt trigger in case a start of frame is detected at a selected communication).

The following table describes the ERU input connections for the ERSx stages. The selection is defined by the bit fields in register **EXISEL**.

Note: All functional inputs of the ERU are synchronized to f_{SYS} before they can affect the internal logic. The resulting delay of $2/f_{SYS}$ and an uncertainty of $1/f_{SYS}$ have to be taken into account for precise timing calculation.
An edge of an input can only be correctly detected if both, the high phase and the low phase of the input are each longer than $1/f_{SYS}$.

Table 6-10 ERSx Connections in XE16x

Input	from/to Module	I/O to ERSx	Can be used to/as
ERS0 Inputs			
ERU_0A0	P2.1	I	ERS0 input A
ERU_0A1	$\overline{\text{ESR1}}$	I	
ERU_0A2	U0C0_DX0INS	I	
ERU_0A3	U0C0_DX2INS	I	
ERU_0B0	P1.0	I	ERS0 input B
ERU_0B1	P5.13	I	
ERU_0B2	U0C1_DX0INS	I	
ERU_0B3	U0C1_DX2INS	I	
ERS1 Inputs			
ERU_1A0	P2.2	I	ERS1 input A
ERU_1A1	$\overline{\text{ESR2}}$	I	
ERU_1A2	U1C0_DX0INS	I	
ERU_1A3	U1C0_DX2INS	I	
ERU_1B0	P1.1	I	ERS1 input B
ERU_1B1	MultiCAN_CAN4INS	I	
ERU_1B2	U1C1_DX0INS	I	
ERU_1B3	U1C1_DX2INS	I	
ERS2 Inputs			
ERU_2A0	P1.2	I	ERS2 input A
ERU_2A1	MultiCAN_CAN3INS	I	
ERU_2A2	U2C0_DX0INS	I	
ERU_2A3	U2C0_DX2INS	I	

Table 6-10 ERSx Connections in XE16x (cont'd)

Input	from/to Module	I/O to ERSx	Can be used to/as
ERU_2B0	U2C0_DX1INS	I	ERS2 input B
ERU_2B1	MultiCAN_CAN2INS	I	
ERU_2B2	U2C1_DX0INS	I	
ERU_2B3	U2C1_DX2INS	I	
ERS3 Inputs			
ERU_3A0	P1.3	I	ERS3 input A
ERU_3A1	MultiCAN_CAN1INS	I	
ERU_3A2	0	I	
ERU_3A3	0	I	
ERU_3B0	U1C0_DX1INS	I	ERS3 input B
ERU_3B1	MultiCAN_CAN0INS	I	
ERU_3B2	0	I	
ERU_3B3	0	I	

6.8.3 External Request Select Unit (ERSx)

For each Input Channel x (x = 0-3), an ERSx unit handles the input selection for the associated ETLx unit. Each ERSx performs a logical combination of two signals (Ax, Bx) to provide one combined output ERSxO to the associated ETLx. Input Ax can be selected from 4 options of the input vector ERU_xA[3:0] and can be optionally inverted. A similar structure exists for input Bx (selection from ERU_xB[3:0]).

In addition to the direct choice of either input Ax or Bx or their inverted values, the possible logical combinations for two selected inputs are a logical AND or a logical OR.

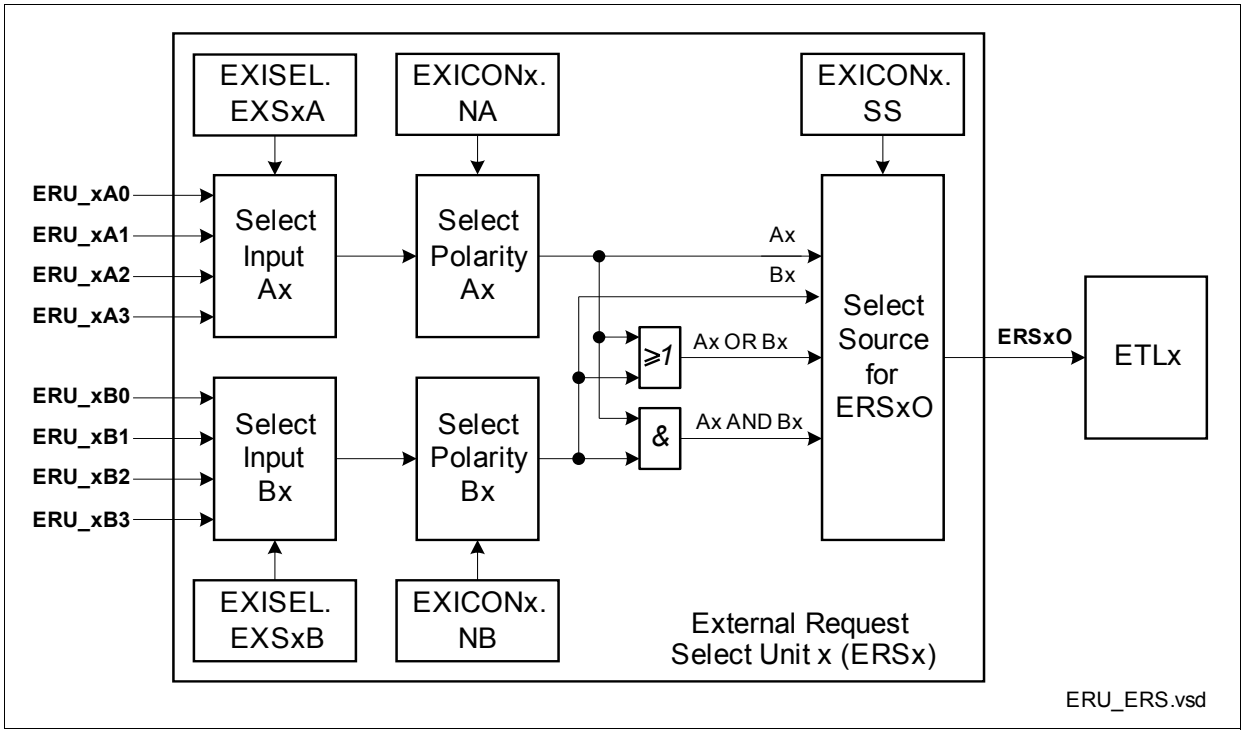


Figure 6-22 External Request Select Unit Overview

The ERS units are controlled via register **EXISEL** (one register for all four ERSx units) and registers EXICONx (one register for each ERSx and associated ETLx unit, e.g. **EXICON0** for Input Channel 0).

6.8.4 Event Trigger Logic (ETLx)

For each Input Channel x (x = 0-3), an event trigger logic ETLx derives a trigger event and a status from the input ERUxO delivered by the associated ERSx unit. Each ETLx is based on an edge detection block, where the detection of a rising or a falling edge can be individually enabled. Both edges lead to a trigger event if both enable bits are set (e.g. to handle a toggling input).

Each of the four ETLx units has an associated EXICONx register, that controls all options of an ETLx (the register also holds control bits for the associated ERSx unit, e.g. **EXICON0** to control ESR0 and ETL0).

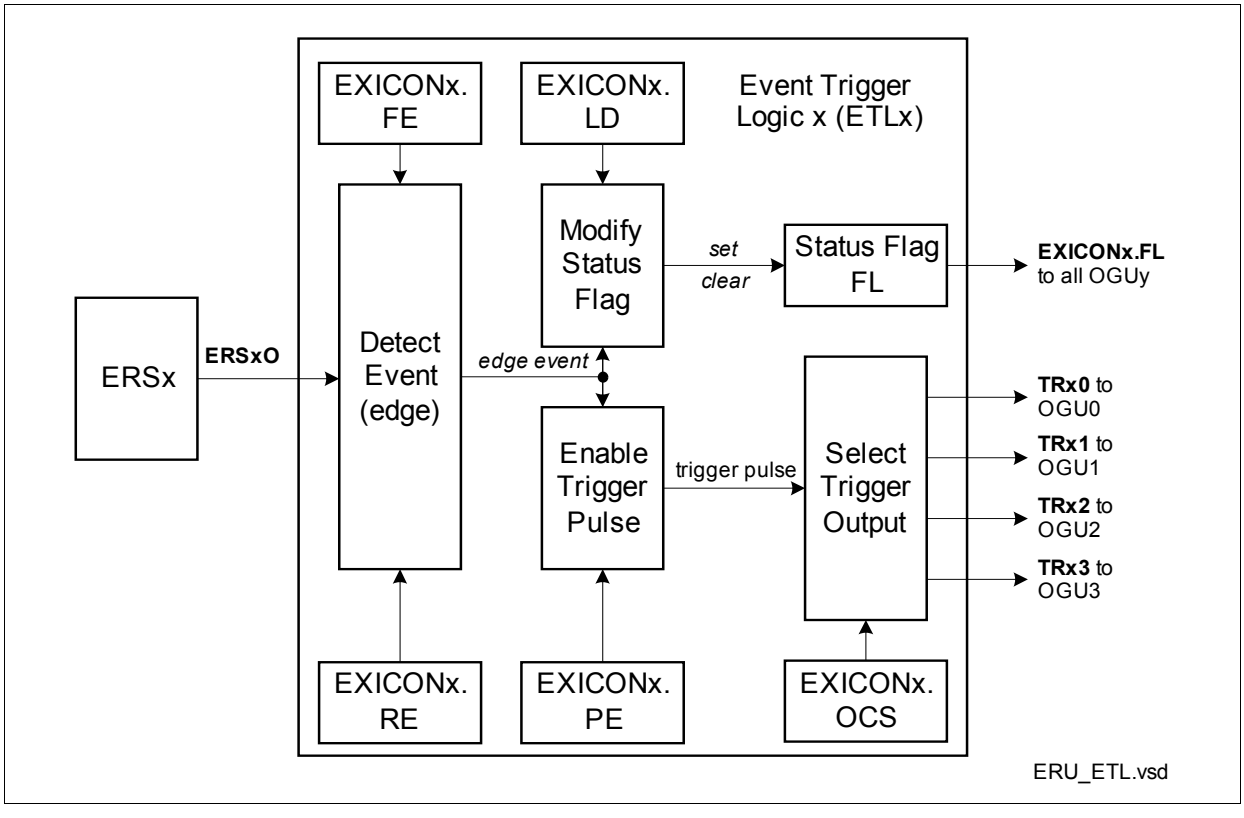


Figure 6-23 Event Trigger Logic Overview

When the selected event (edge) is detected, the status flag EXICONx.FL becomes set. This flag can also be modified by software (set or clear). Two different operating modes are supported by this status flag.

It can be used as “sticky” flag, that is set by hardware when the desired event has been detected and has to be cleared by software. In this operating mode, it indicates that the event has taken place, but without indicating the actual status of the input.

In the second operating mode, it is cleared automatically if the “opposite” event is detected. For example, if only the falling edge detection is enabled to set the status flag, it is cleared when the rising edge is detected. In this mode, it can be used for pattern

detection where the actual status of the input is important (enabling both edge detections is not useful in this mode).

The output of the status flag is connected to all following Output Gating Units (OGUy) in parallel (see **Figure 6-24**) to provide **pattern detection capability of all OGUy** units based on different or the same status flags.

In addition to the modification of the status flag, a trigger pulse output TRxy of ETLx can be enabled (by bit EXICONx.PE) and selected to **trigger actions in one of the OGUy** units. The target OGUy for the trigger is selected by bit field EXICON.OCS.

The trigger becomes active when the selected edge event is detected, independently from the status flag EXICONx.FL.

6.8.5 Connecting Matrix

The connecting matrix distributes the trigger signals (TRxy) and status signals (EXICONx.FL) from the different ETLx units between the OGUy units. In addition, it receives peripheral trigger signals that can be OR-combined with the ETLx trigger signals in the OGUy units. **Figure 6-24** provides a complete overview of the connections between the ETLx and the OGUy units.

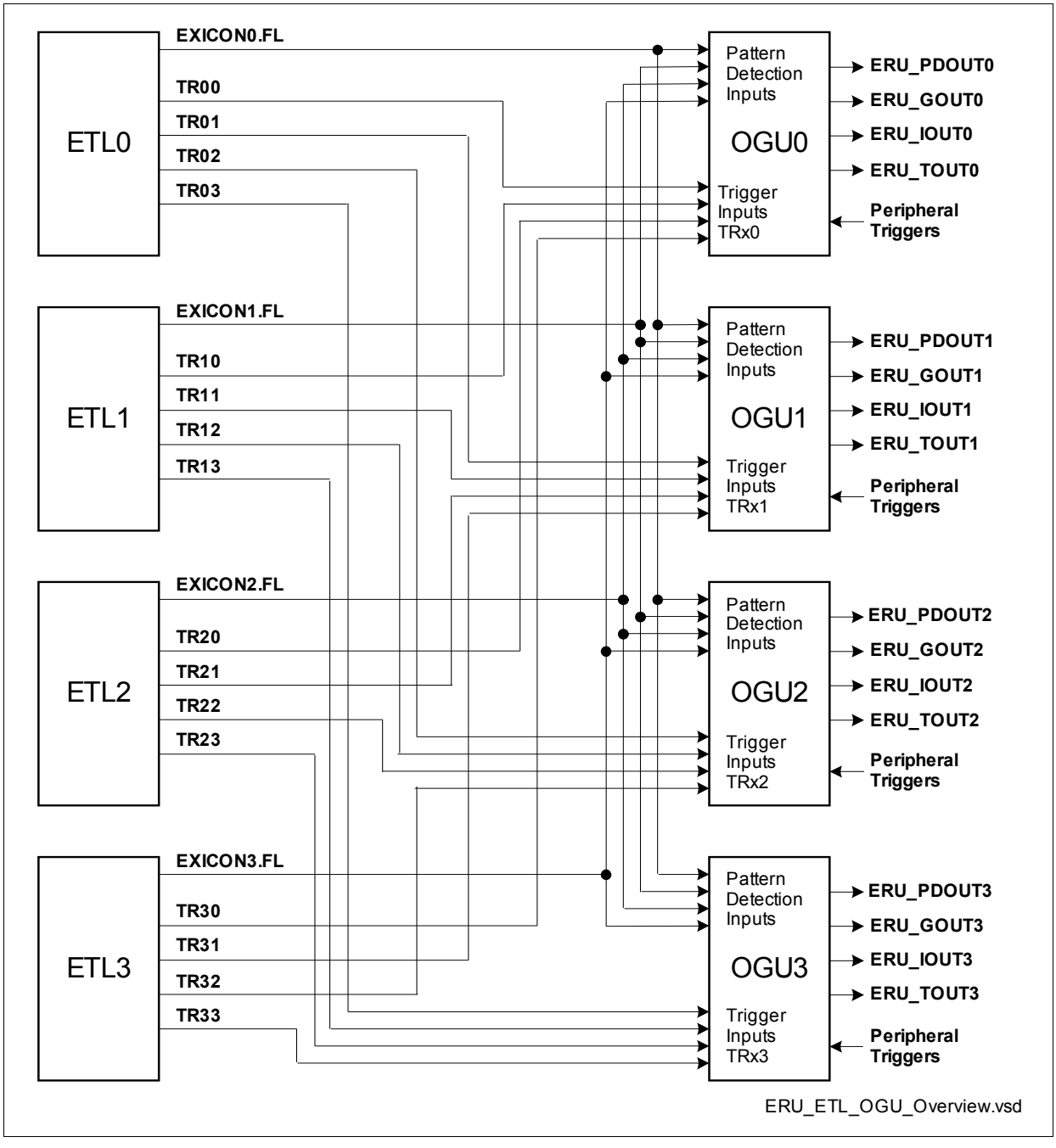


Figure 6-24 Connecting Matrix between ETLx and OGUy

6.8.6 Output Gating Unit (OGUy)

Each OGUy (y = 0-3) unit combines the available trigger events and status flags from the Input Channels and distributes the results to the system. **Figure 6-25** illustrates the logic blocks within an OGUy unit. All functions of an OGUy unit are controlled by its associated EXOCONy register, e.g. **EXOCON0** for OGU0. The function of an OGUy unit can be split into two parts:

- **Trigger combination** (see **Section 6.8.6.1**):
  All trigger signals TRxy from the Input Channels that are enabled and directed to OGUy, a selected peripheral-related trigger event, and a pattern change event (if enabled) are logically OR-combined.
- **Pattern detection** (see **Section 6.8.6.2**):
  The status flags EXICONx.FL of the Input Channels can be enabled to take part in the pattern detection. A pattern match is detected while all enabled status flags are set.

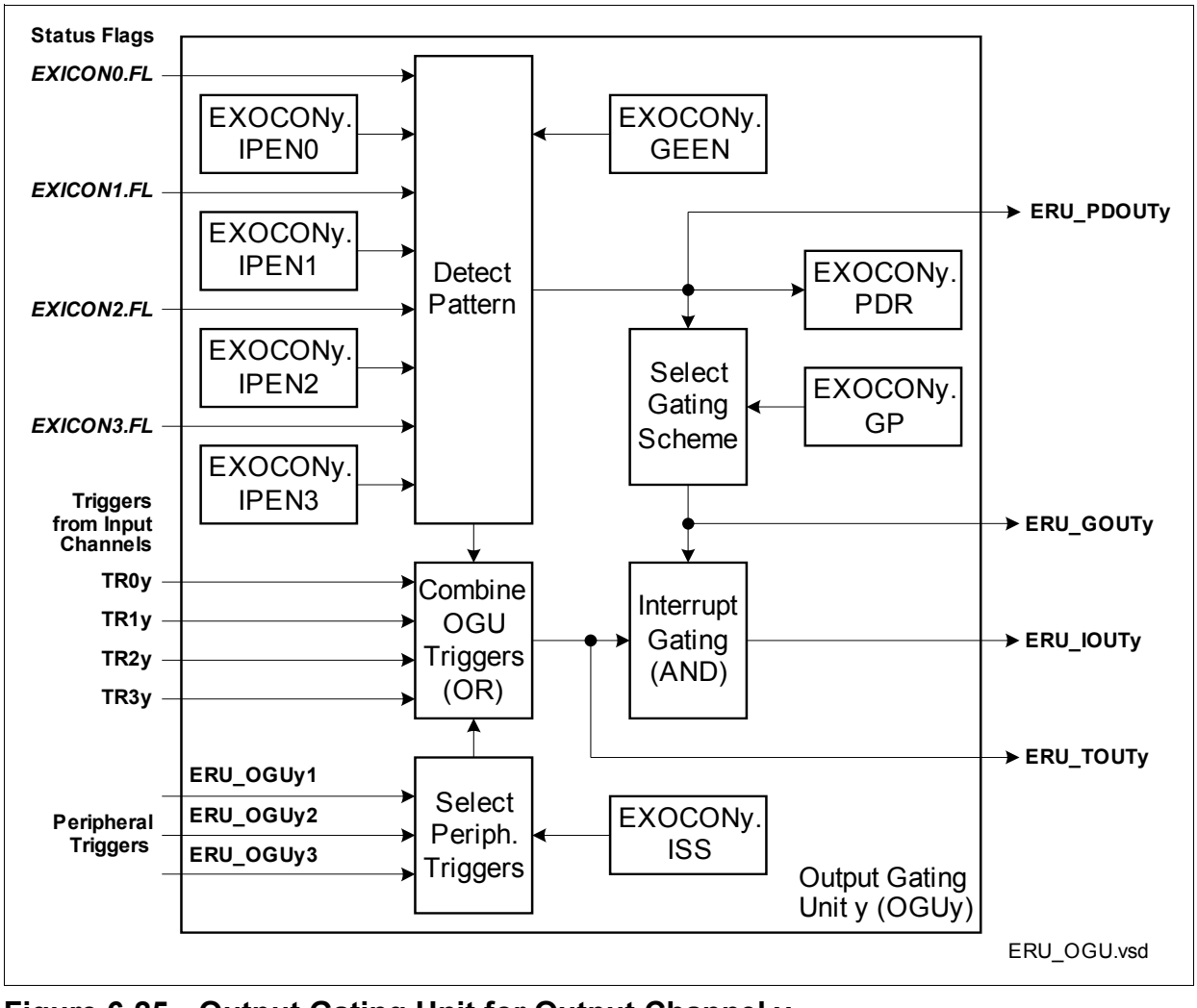


Figure 6-25 Output Gating Unit for Output Channel y

Each OGUy unit generates 4 output signals that are distributed to the system (not all of them are necessarily used, please refer to **Section 6.8.7**):

- **ERU_PDOUTy** to directly output the pattern match information for gating purposes in other modules (pattern match = 1).
- **ERU_GOUTy** to output the pattern match or pattern miss information (inverted pattern match), or a permanent 0 or 1 under software control for gating purposes in other modules.
- **ERU_TOUTy** as combination of a peripheral trigger, a pattern detection result change event, or the ETLx trigger outputs TRxy to trigger actions in other modules.
- **ERU_IOUTy** as gated trigger output (ERU_GOUTy logical AND-combined with ERU_TOUTy) to trigger interrupts (e.g. the interrupt generation can be gated to allow interrupt activation during a certain time window).

6.8.6.1 Trigger Combination

The trigger combination logically OR-combines different trigger inputs to form a common trigger ERU_TOUTy. Possible trigger inputs are:

- In each ETLx unit of the **Input Channels**, the trigger output TRxy can be enabled and the trigger event can be directed to one of the OGUy units.
- One out of three **peripheral trigger** signals per OGUy can be selected as additional trigger source. These peripheral triggers are generated by on-chip peripheral modules, such as capture/compare or timer units. The selection is done by bit field EXOCONy.ISS.
- In the case that at least one **pattern detection** input is enabled (EXOCONy.IPENx) and a change of the pattern detection result from pattern match to pattern miss (or vice-versa) is detected, a trigger event is generated to indicate a pattern detection result event (if enabled by ECOCONy.GEEN).

The trigger combination offers the possibility to program different trigger criteria for several input signals (independently for each Input Channel) or peripheral signals, and to combine their effects to a single output, e.g. to generate an interrupt or to start an ADC conversion. This combination capability allows the generation of an interrupt per OGU that can be triggered by several inputs (multitude of request sources -> one reaction).

The following table describes the peripheral trigger connections for the OGUy stages. The selection is defined by the bit fields ISS in registers **EXOCON0** (for OGU0), **EXOCON1** (for OGU1), **EXOCON2** (for OGU2), or **EXOCON3** (for OGU3).

Table 6-11 OGUy Peripheral Trigger Connections in XE16x

Input	from/to Module	I/O to OGUy	Can be used to/as
OGU0 Inputs			
ERU_OGU01	CCU60_MCM_ST	I	Peripheral triggers for OGU0
ERU_OGU02	CCU60_T13_PM	I	
ERU_OGU03	CC2_28	I	
OGU1 Inputs			
ERU_OGU11	CCU61_MCM_ST	I	Peripheral triggers for OGU1
ERU_OGU12	CCU61_T13_PM	I	
ERU_OGU13	CC2_29	I	
OGU2 Inputs			
ERU_OGU21	CCU62_MCM_ST	I	Peripheral triggers for OGU2
ERU_OGU22	CCU62_T13_PM	I	
ERU_OGU23	CC2_30	I	
OGU3 Inputs			
ERU_OGU31	CCU63_MCM_ST	I	Peripheral triggers for OGU3
ERU_OGU32	CCU63_T13_PM	I	
ERU_OGU33	CC2_31	I	

6.8.6.2 Pattern Detection

The pattern detection logic allows the combination of the status flags of all ETLx units. Each status flag can be individually included or excluded from the pattern detection for each OGUy, via control bits EXOCONy.IPENx. The pattern detection block outputs the following pattern detection results:

- **Pattern match** (EXOCONy.PDR = 1 and ERU_PDOUTy = 1):
  A pattern match is indicated while all status flags FL that are included in the pattern detection are 1.
- **Pattern miss** (EXOCONy.PDR = 0 and ERU_PDOUTy = 0):
  A pattern miss is indicated while at least one of the status flags FL that are included in the pattern detection is 0.

In addition, the pattern detection can deliver a trigger event if the pattern detection result changes from match to miss or vice-versa (if enabled by EXOCONy.GEEN = 1). The pattern result change event is logically OR-combined with the other enabled trigger events to support interrupt generation or to trigger other module functions (e.g. in the ADC). The event is indicated when the pattern detection result changes and EXOCONy.PDR becomes updated.

The interrupt generation in the OGUy is based on the trigger ERU_TOUTy that can be gated (masked) with the pattern detection result ERU_PDOUTy. This allows an automatic and reproducible generation of interrupts during a certain time window, where the request event is elaborated by the trigger combination block and the time window information (gating) is given by the pattern detection. For example, interrupts can be issued on a regular time base (peripheral trigger input from capture/compare unit is selected) while a combination of input signals occurs (pattern detection based on ETLx status bits).

A programmable gating scheme introduces flexibility to adapt to application requirements and allows the generation of interrupt requests ERU_IOUTy under different conditions:

- **Pattern match** (EXOCONy.GP = 10_B):
  An interrupt request is issued when a trigger event occurs while the pattern detection shows a pattern match.
- **Pattern miss** (EXOCONy.GP = 11_B):
  An interrupt request is issued when the trigger event occurs while the pattern detection shows a pattern miss.
- **Independent** of pattern detection (EXOCONy.GP = 01_B):
  In this mode, each occurring trigger event leads to an interrupt request. The pattern detection output can be used independently from the trigger combination for gating purposes of other peripherals (independent use of ERU_TOUTy and ERU_PDOUTy with interrupt requests on trigger events).
- **No interrupts** (EXOCONy.GP = 00_B, default setting)
  In this mode, an occurring trigger event does not lead to an interrupt request. The pattern detection output can be used independently from the trigger combination for gating purposes of other peripherals (independent use of ERU_TOUTy and ERU_PDOUTy without interrupt requests on trigger events).

6.8.7 ERU Output Connections

This section describes the connections of the ERU output signals for gating or triggering other module functions, as well as the connections to the interrupt control registers.

Table 6-12 ERU Output Connections in XE16x

Output	from/to Module	I/O to OGUy	Can be used to/as
OGU0 Outputs			
ERU_PDOUT0	not connected	O	Pattern detection output
ERU_GOUT0	ADC0 (REQGT0A) ADC0 (REQGT1A) ADC0 (REQGT2A) ADC1 (REQGT0A) ADC1 (REQGT1A) ADC1 (REQGT2A)	O	Gated pattern detection output
ERU_TOUT0	not connected	O	Trigger output
ERU_IOUT0	ITC (CC2_CC16IC)	O	Interrupt output
OGU1 Outputs			
ERU_PDOUT1	not connected	O	Pattern detection output
ERU_GOUT1	ADC0 (REQGT0B) ADC0 (REQGT1B) ADC0 (REQGT2B) ADC1 (REQGT0B) ADC1 (REQGT1B) ADC1 (REQGT2B)	O	Gated pattern detection output
ERU_TOUT1	ADC0 (REQTR0A) ADC0 (REQTR1A) ADC0 (REQTR2A) ADC1 (REQTR0A) ADC1 (REQTR1A) ADC1 (REQTR2A)	O	Trigger output
ERU_IOUT1	ITC (CC2_CC17IC)	O	Interrupt output
OGU2 Outputs			
ERU_PDOUT2	not connected	O	Pattern detection output
ERU_GOUT2	not connected	O	Gated pattern detection output

Table 6-12 ERU Output Connections in XE16x (cont'd)

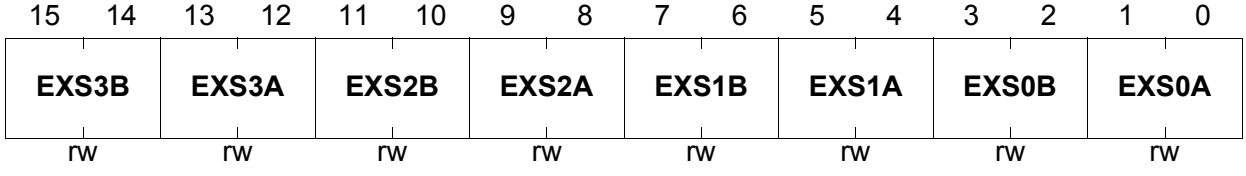
Output	from/to Module	I/O to OGUy	Can be used to/as
ERU_TOUT2	not connected	O	Trigger output
ERU_IOUT2	ITC (CC2_CC18IC)	O	Interrupt output
OGU3 Outputs			
ERU_PDOUT3	not connected	O	Pattern detection output
ERU_GOUT3	not connected	O	Gated pattern detection output
ERU_TOUT3	not connected	O	Trigger output
ERU_IOUT3	ITC (CC2_CC19IC)	O	Interrupt output

6.8.8 ERU Registers

6.8.8.1 External Input Selection Register EXISEL

This register selects the A and B inputs for all four ERS units. The possible input signals are given in **Table 6-10**.

EXISEL
External Input Select Register ESFR ($F1A0_H$/$D0_H$) Reset Value: 0000_H



Field	Bits	Type	Description
EXS0A	[1:0]	rw	**External Source Select for A0 (ERS0)** This bit field defines which input is selected for A0. 00_B Input ERU_0A0 is selected 01_B Input ERU_0A1 is selected 10_B Input ERU_0A2 is selected 11_B Input ERU_0A3 is selected
EXS0B	[3:2]	rw	**External Source Select for B0 (ERS0)** This bit field defines which input is selected for B0. 00_B Input ERU_0B0 is selected 01_B Input ERU_0B1 is selected 10_B Input ERU_0B2 is selected 11_B Input ERU_0B3 is selected
EXS1A	[5:4]	rw	**External Source Select for A1 (ERS1)** This bit field defines which input is selected for A1. 00_B Input ERU_1A0 is selected 01_B Input ERU_1A1 is selected 10_B Input ERU_1A2 is selected 11_B Input ERU_1A3 is selected
EXS1B	[7:6]	rw	**External Source Select for B1 (ERS1)** This bit field defines which input is selected for B1. 00_B Input ERU_1B0 is selected 01_B Input ERU_1B1 is selected 10_B Input ERU_1B2 is selected 11_B Input ERU_1B3 is selected

Field	Bits	Type	Description
EXS2A	[9:8]	rw	**External Source Select for A2 (ERS2)** This bit field defines which input is selected for A2. 00_B Input ERU_2A0 is selected 01_B Input ERU_2A1 is selected 10_B Input ERU_2A2 is selected 11_B Input ERU_2A3 is selected
EXS2B	[11:10]	rw	**External Source Select for B2 (ERS2)** This bit field defines which input is selected for B2. 00_B Input ERU_2B0 is selected 01_B Input ERU_2B1 is selected 10_B Input ERU_2B2 is selected 11_B Input ERU_2B3 is selected
EXS3A	[13:12]	rw	**External Source Select for A3 (ERS3)** This bit field defines which input is selected for A3. 00_B Input ERU_3A0 is selected 01_B Input ERU_3A1 is selected 10_B Input ERU_3A2 is selected 11_B Input ERU_3A3 is selected
EXS3B	[15:14]	rw	**External Source Select for B3 (ERS3)** This bit field defines which input is selected for B3. 00_B Input ERU_3B0 is selected 01_B Input ERU_3B1 is selected 10_B Input ERU_3B2 is selected 11_B Input ERU_3B3 is selected

6.8.8.2 External Input Control Registers EXICONx

These registers control the inputs of the ERSx unit and the trigger functions of the ETLx units (x = 0..3).

EXICON0
External Input Control 0 Register
ESFR ($F030_H/18_H$) **Reset Value: 0000_H**

EXICON1
External Input Control 1 Register
ESFR ($F032_H/19_H$) **Reset Value: 0000_H**

EXICON2
External Input Control 2 Register
ESFR ($F034_H/1A_H$) **Reset Value: 0000_H**

EXICON3
External Input Control 3 Register
ESFR ($F036_H/1C_H$) **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0				**NB**	**NA**	**SS**		**FL**	**OCS**			**FE**	**RE**	**LD**	**PE**
r				rw	rw	rw		rwh	rw			rw	rw	rw	rw

Field	Bits	Type	Description
PE	0	rw	**Output Trigger Pulse Enable for ETLx** This bit enables the generation of an output trigger pulse at TRxy when the selected edge is detected (set condition for the status flag FL). 0_B The trigger pulse generation is disabled 1_B The trigger pulse generation is enabled

Field	Bits	Type	Description
LD	1	rw	**Rebuild Level Detection for Status Flag for ETLx** This bit selects if the status flag FL is used as “sticky” bit or if it rebuilds the result of a level detection. 0_B The status flag FL is not cleared by hardware and is used as “sticky” bit. Once set, it is not influenced by any edge until it becomes cleared by software. 1_B The status flag FL rebuilds a level detection of the desired event. It becomes automatically set with a rising edge if RE = 1 or with a falling edge if FE = 1. It becomes automatically cleared with a rising edge if RE = 0 or with a falling edge if FE = 0.
RE	2	rw	**Rising Edge Detection Enable ETLx** This bit enables/disables the rising edge event as edge event as set condition for the status flag FL or as possible trigger pulse for TRxy. 0_B A rising edge is not considered as edge event 1_B A rising edge is considered as edge event
FE	3	rw	**Falling Edge Detection Enable ETLx** This bit enables/disables the falling edge event as edge event as set condition for the status flag FL or as possible trigger pulse for TRxy. 0_B A falling edge is not considered as edge event 1_B A falling edge is considered as edge event
OCS	[6:4]	rw	**Output Channel Select for ETLx Output Trigger Pulse** This bit field defines which Output Channel OGUy is targeted by an enabled trigger pulse TRxy. 000_B Trigger pulses are sent to OGU0 001_B Trigger pulses are sent to OGU1 010_B Trigger pulses are sent to OGU2 011_B Trigger pulses are sent to OGU3 $1XX_B$ Reserved, do not use this combination
FL	7	rwh	**Status Flag for ETLx** This bit represents the status flag that becomes set or cleared by the edge detection. 0_B The enabled edge event has not been detected 1_B The enabled edge event has been detected

Field	Bits	Type	Description
SS	[9:8]	rw	**Input Source Select for ERSx** This bit field defines which logical combination is taken into account as ESRxO. 00_B Input A without additional combination 01_B Input B without additional combination 10_B Input A OR input B 11_B Input A AND input B
NA	10	rw	**Input A Negation Select for ERSx** This bit selects the polarity for the input A. 0_B Input A is used directly 1_B Input A is inverted
NB	11	rw	**Input B Negation Select for ERSx** This bit selects the polarity for the input B. 0_B Input B is used directly 1_B Input B is inverted
0	[15:12]	r	**Reserved** Read as 0; should be written with 0.

6.8.8.3 Output Control Registers EXOCONy

These registers control the outputs of the Output Gating Unit y (y = 0..3).

EXOCON0
External Output Trigger Control 0 Register
SFR ($FE30_H/18_H$) **Reset Value: 0000_H**

EXOCON1
External Output Trigger Control 1 Register
SFR ($FE32_H/19_H$) **Reset Value: 0000_H**

EXOCON2
External Output Trigger Control 2 Register
SFR ($FE34_H/1A_H$) **Reset Value: 0000_H**

EXOCON3
External Output Trigger Control 3 Register
SFR ($FE36_H/1B_H$) **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IPEN3	IPEN2	IPEN1	IPEN0	0						GP		PDR	GEEN	ISS	
rw	rw	rw	rw	r						rw		rh	rw	rw	

Field	Bits	Type	Description
ISS	[1:0]	rw	**Internal Trigger Source Selection** This bit field defines which input is selected as peripheral trigger input for OGUy. The possible input signals are given in **Table 6-11**. 00_B The peripheral trigger function is disabled 01_B Input ERU_OGUy1 is selected 10_B Input ERU_OGUy2 is selected 11_B Input ERU_OGUy3 is selected
GEEN	2	rw	**Gating Event Enable** Bit GEEN enables the generation of a trigger event when the result of the pattern detection changes from match to miss or vice-versa. 0_B The event detection is disabled 1_B The event detection is enabled
PDR	3	rh	**Pattern Detection Result Flag** This bit represents the pattern detection result. 0_B A pattern miss is detected 1_B A pattern match is detected

Field	Bits	Type	Description
GP	[5:4]	rw	**Gating Selection for Pattern Detection Result** This bit field defines the gating scheme for the interrupt generation (relation between the OGU output ERU_PDOUTy and ERU_GOUTy). 00_B ERU_GOUTy is always disabled and ERU_IOUTy can not be activated 01_B ERU_GOUTy is always enabled and ERU_IOUTy becomes activated with each activation of ERU_TOUTy 10_B ERU_GOUTy is equal to ERU_PDOUTy and ERU_IOUTy becomes activated with an activation of ERU_TOUTy while the desired pattern is detected (pattern match PDR = 1) 11_B ERU_GOUTy is inverted to ERU_PDOUTy and ERU_IOUTy becomes activated with an activation of ERU_TOUTy while the desired pattern is not detected (pattern miss PDR = 0)
IPENx (x = 0-3)	12+x	rw	**Pattern Detection Enable for ETLx** Bit IPENx defines whether the trigger event status flag EXICONx.FL of ETLx takes part in the pattern detection of OGUy. 0_B Flag EXICONx.FL is excluded from the pattern detection 1_B Flag EXICONx.FL is included in the pattern detection
0	[11:6]	r	**Reserved** Read as 0; should be written with 0.

6.9 SCU Interrupt Generation

The interrupt structure of the SCU is shown in **Figure 6-26**.

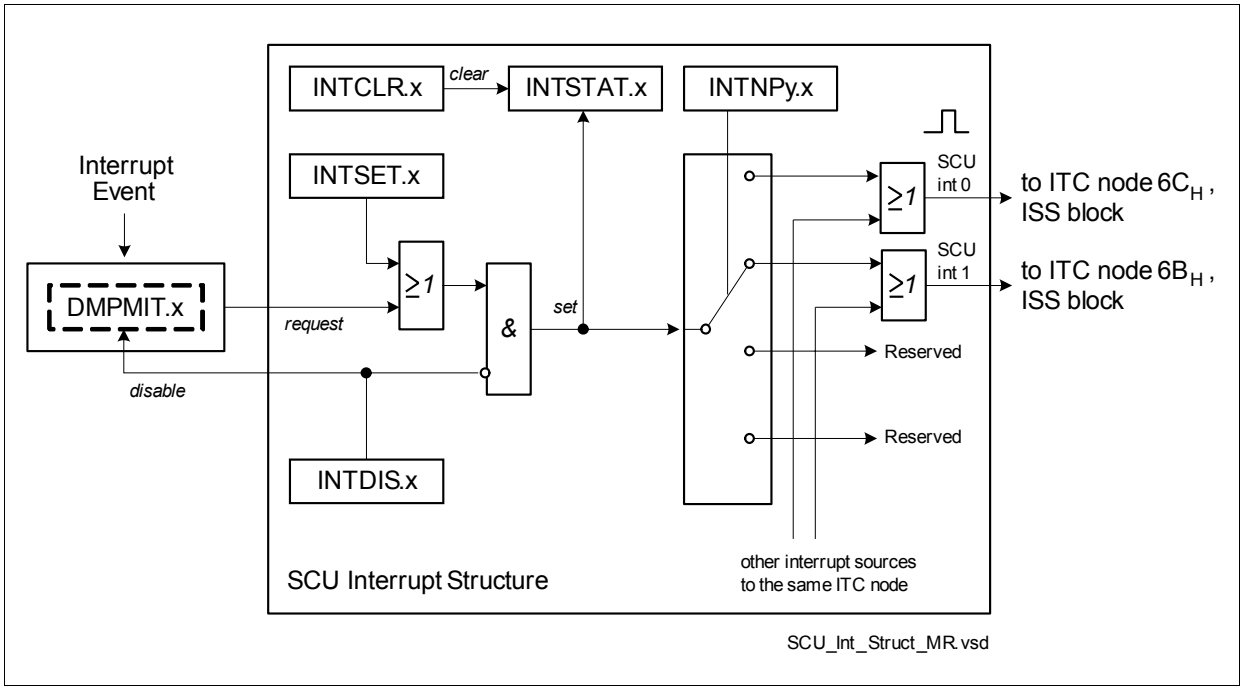


Figure 6-26 SCU Interrupt Structure

If enabled by the corresponding bit in register **INTDIS**, an interrupt is triggered either by the incoming interrupt request line, or by a software set of the respective bit in register **INTSET**. The trigger sets the respective flag in register **INTSTAT** and is gated to one of the interrupt nodes, selected by the node pointer registers **INTNP0** or **INTNP1**.

The interrupt flag can be cleared by software by writing to the corresponding bit in register **INTCLR**.

If more than one interrupt source is connected to the same interrupt node pointer (in register INTNPx), the requests are combined to one common line.

Interrupt Node Assignment

The interrupt sources of the SCU module can be mapped to the dedicated interrupt node $6C_H$ or $6B_H$ by programming the interrupt node pointer registers INTNP0 and INTNP1.

The default assignment of the interrupt sources to the nodes and their corresponding control registers are shown in **Table 6-13**.

6.9.1 Interrupt Support

Some of the interrupt requests are first fed through a sticky flag register in the DMP_M domain. These flags are set with a trigger and if set trigger the interrupt generation in the

DMP_1.. Please note that the disable control of register INTDIS also influences the sticky bit in register **DMPMIT** (see **Section 6.9.3.7**).

Which of the interrupt requests have a sticky flag in register DMPMIT is listed in **Table 6-13**.

Note: When servicing an SCU interrupt request, make sure that all related request flags are cleared after the identified request has been handled. To clear an interrupt request that is stored in register DMPMIT, first clear the request source of the source (e.g. WUTRG), clear the request within DMP_M via DMPMITCLR, and then clear the request within DMP_1 via INTCLR.

6.9.2 SCU Interrupt Sources

The SCU receives the interrupt request lines listed in **Table 6-13**.

Table 6-13 SCU Interrupt Overview

Source of Interrupt	Short Name	Sticky Flag in DMPMIT	Default Interrupt Node Assignment in INTNPx
SWD OK 1 Interrupt	SWDI1	yes	$6C_H$
SWD OK 2 Interrupt	SWDI2	yes	$6B_H$
PVC_M OK 1 Interrupt	PVCMI1	yes	$6C_H$
PVC_M OK 2 Interrupt	PVCMI2	yes	$6B_H$
PVC_1 OK 1 Interrupt	PVC1I1	yes	$6C_H$
PVC_1 OK 2 Interrupt	PVC1I2	yes	$6B_H$
Wake-up Timer Interrupt	WUTI	yes	$6B_H$
Wake-up Trim Interrupt	WUI	yes	$6C_H$
Watchdog Timer Interrupt	WDTI	---	$6B_H$
GSC Interrupt	GSCI	yes	$6C_H$

6.9.3 Interrupt Control Registers

6.9.3.1 Register INTSTAT

This register contains the status flags for all interrupt request trigger sources of the SCU. For setting and clearing of these status bits by software see registers INTSET and INTCLR, respectively.

INTSTAT
Interrupt Status Register **SFR ($FF00_H/80_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						GSCI	WDTI	WUI	WUTI	PVC1I2	PVC1I1	PVCMI2	PVCMI1	SWDI2	SWDI1
rh						rh	rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
SWDI1	0	rh	**SWD Interrupt Request Flag 1** This bit is set if bit DMPMIT.SWDI1 is set. 0_B No SWDI1 interrupt trigger has occured since this bit was cleared the last time 1_B A SWDI1 interrupt trigger has occured since this bit was cleared the last time
SWDI2	1	rh	**SWD Interrupt Request Flag 2** This bit is set if bit DMPMIT.SWDI2 is set. 0_B No SWDI2 interrupt trigger has occured since this bit was cleared the last time 1_B A SWDI2 interrupt trigger has occured since this bit was cleared the last time
PVCMI1	2	rh	**PVC_M Interrupt Request Flag 1** This bit is set if bit DMPMIT.PVCMI1 is set. 0_B No PVCMI1 interrupt trigger has occured since this bit was cleared the last time 1_B A PVCMI1 interrupt trigger has occured since this bit was cleared the last time

Field	Bits	Type	Description
PVCMI2	3	rh	**PVC_M Interrupt Request Flag 2** This bit is set if bit DMPMIT.PVCMI2 is set. 0_B No PVCMI2 interrupt trigger has occured since this bit was cleared the last time 1_B A PVCMI2 interrupt trigger has occured since this bit was cleared the last time
PVC1I1	4	rh	**PVC_1 Interrupt Request Flag 1** This bit is set if bit DMPMIT.PVC1I1 is set. 0_B No PVC1I1 interrupt trigger has occured since this bit was cleared the last time 1_B A PVCI1 interrupt trigger has occured since this bit was cleared the last time
PVC1I2	5	rh	**PVC_1 Interrupt Request Flag 2** This bit is set if bit DMPMIT.PVC1I2 is set. 0_B No PVC1I2 interrupt trigger has occured since this bit was cleared the last time 1_B A PVC1I2 interrupt trigger has occured since this bit was cleared the last time
WUTI	6	rh	**Wake-up Timer Trim Interrupt Request Flag** This bit is set if the WUT trim trigger event occur and bit is INTDIS.WUTI = 0. 0_B No WUT interrupt trigger has occured since this bit was cleared the last time 1_B A WUT interrupt trigger has occured since this bit was cleared the last time
WUI	7	rh	**Wake-up Timer Interrupt Request Flag** This bit is set if the WU trigger event occur and bit is INTDIS.WUI = 0. 0_B No WU interrupt trigger has occured since this bit was cleared the last time 1_B A WU interrupt trigger has occured since this bit was cleared the last time
WDTI	8	rh	**Watchdog Timer Interrupt Request Flag** This bit is set if the WDT Prewarning Mode is entered and bit is INTDIS.WDTI = 0. 0_B No WDT interrupt trigger has occured since this bit was cleared the last time 1_B A WDT interrupt trigger has occured since this bit was cleared the last time

Field	Bits	Type	Description
GSCI	9	rh	**GSC Interrupt Request Flag** This bit is set if the GSC error bit is set and bit is INTDIS.GSCI = 0. 0_B No GSC interrupt trigger has occured since this bit was cleared the last time 1_B A GSC interrupt trigger has occured since this bit was cleared the last time
0	[15:10]	rh	**Reserved** Read as 0; should be written with 0.

6.9.3.2 Register INTCLR

This register contains the software clear option for all status flags of all interrupt request trigger sources of the SCU.

INTCLR
Interrupt Clear Register **SFR ($FE82_H/41_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						GSCI	WDTI	WUI	WUTI	PVC1I2	PVC1I1	PVCMI2	PVCMI1	SWDI2	SWDI1
w						w	w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
SWDI1	0	w	**Clear SWD Interrupt Request Flag 1** 0_B No action 1_B Bit INTSTAT.SWDI1 is cleared
SWDI2	1	w	**Clear SWD Interrupt Request Flag 2** 0_B No action 1_B Bit INTSTAT.SWDI2 is cleared
PVCMI1	2	w	**Clear PVC_M Interrupt Request Flag 1** 0_B No action 1_B Bit INTSTAT.PVCMI1 is cleared
PVCMI2	3	w	**Clear PVC_M Interrupt Request Flag 2** 0_B No action 1_B Bit INTSTAT.PVCMI2 is cleared
PVC1I1	4	w	**Clear PVC_1 Interrupt Request Flag 1** 0_B No action 1_B Bit INTSTAT.PVC1I1 is cleared
PVC1I2	5	w	**Clear PVC_1 Interrupt Request Flag 2** 0_B No action 1_B Bit INTSTAT.PVC1I2 is cleared
WUTI	6	w	**Clear Wake-up Trim Interrupt Request Flag** 0_B No action 1_B Bit INTSTAT.WUTI is cleared
WUI	7	w	**Clear Wake-up Interrupt Request Flag** 0_B No action 1_B Bit INTSTAT.WUI is cleared

Field	Bits	Type	Description
WDTI	8	w	**Clear Watchdog Timer Interrupt Request Flag** 0_B No action 1_B Bit INTSTAT.WDTI is cleared
GSCI	9	w	**Clear GSC Interrupt Request Flag** 0_B No action 1_B Bit INTSTAT.GSCI is cleared
0	[15:10]	w	**Reserved** Must be written with 0.

6.9.3.3 Register INTSET

This register contains the software set option for all status flags of all interrupt request trigger sources of the SCU.

INTSET
Interrupt Set Register **SFR ($FE80_H/40_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						GSCI	WDTI	WUI	WUTI	PVC1I2	PVC1I1	PVCMI2	PVCMI1	SWDI2	SWDI1
w						w	w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
SWDI1	0	w	**Set SWD Interrupt Request Flag 1** 0_B No action 1_B Bit INTSTAT.SWDI1 is set
SWDI2	1	w	**Set SWD Interrupt Request Flag 2** 0_B No action 1_B Bit INTSTAT.SWDI2 is set
PVCMI1	2	w	**Set PVC_M Interrupt Request Flag 1** 0_B No action 1_B Bit INTSTAT.PVCMI1 is set
PVCMI2	3	w	**Set PVC_M Interrupt Request Flag 2** 0_B No action 1_B Bit INTSTAT.PVCMI2 is set
PVC1I1	4	w	**Set PVC_1 Interrupt Request Flag 1** 0_B No action 1_B Bit INTSTAT.PVC1I1 is set
PVC1I2	5	w	**Set PVC_1 Interrupt Request Flag 2** 0_B No action 1_B Bit INTSTAT.PVC1I2 is set
WUTI	6	w	**Set Wake-up Trim Interrupt Request Flag** 0_B No action 1_B Bit INTSTAT.WUTI is set
WUI	7	w	**Set Wake-up Interrupt Request Flag** 0_B No action 1_B Bit INTSTAT.WUI is set

Field	Bits	Type	Description
WDTI	8	w	**Set Watchdog Timer Interrupt Request Flag** 0_B No action 1_B Bit INTSTAT.WDTI is set
GSCI	9	w	**Set GSC Interrupt Request Flag** 0_B No action 1_B Bit INTSTAT.GSCI is set
0	[15:10]	w	**Reserved** Must be written with 0.

6.9.3.4 Register INTDIS

This register contains the software disable control for all interrupt request trigger sources of the SCU.

INTDIS
Interrupt Disable Register **SFR ($FE84_H/42_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						GSCI	WDTI	WUI	WUTI	PVC1I2	PVC1I1	PVCMI2	PVCMI1	SWDI2	SWDI1
rw						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
SWDI1	0	rw	**Disable SWD Interrupt Request 1** 0_B SWDI1 interrupt request enabled 1_B SWDI1 interrupt request disabled
SWDI2	1	rw	**Disable SWD Interrupt Request 2** 0_B SWDI2 interrupt request enabled 1_B SWDI2 interrupt request disabled
PVCMI1	2	rw	**Disable PVC_M Interrupt Request 1** 0_B PVCMI1 interrupt request enabled 1_B PVCMI1 interrupt request disabled
PVCMI2	3	rw	**Disable PVC_M Interrupt Request 2** 0_B PVCMI2 interrupt request enabled 1_B PVCMI2 interrupt request disabled
PVC1I1	4	rw	**Disable PVC_1 Interrupt Request 1** 0_B PVC1I1 interrupt request enabled 1_B PVC1I1 interrupt request disabled
PVC1I2	5	rw	**Disable PVC_1 Interrupt Request 2** 0_B PVC1I2 interrupt request enabled 1_B PVC1I2 interrupt request disabled
WUTI	6	rw	**Disable Wake-up Trim Interrupt Request** 0_B WUT interrupt request enabled 1_B WUT interrupt request disabled
WUI	7	rw	**Disable Wake-up Interrupt Request** 0_B WU interrupt request enabled 1_B WU interrupt request disabled

Field	Bits	Type	Description
WDTI	8	rw	**Disable Watchdog Timer Interrupt Request** 0_B WDT interrupt request enabled 1_B WDT interrupt request disabled
GSCI	9	rw	**Disable GSC Interrupt Request** 0_B GSC interrupt request enabled 1_B GSC interrupt request disabled
0	[15:10]	rw	**Reserved** Should be written with 0.

6.9.3.5 Registers INTNP0 and INPNP1

These registers contain the control for the interrupt node pointers of all interrupt request trigger sources of the SCU.

INTNP0
Interrupt Node Pointer 0 Register

SFR ($FE86_H/43_H$) **Reset Value: 4444_H**

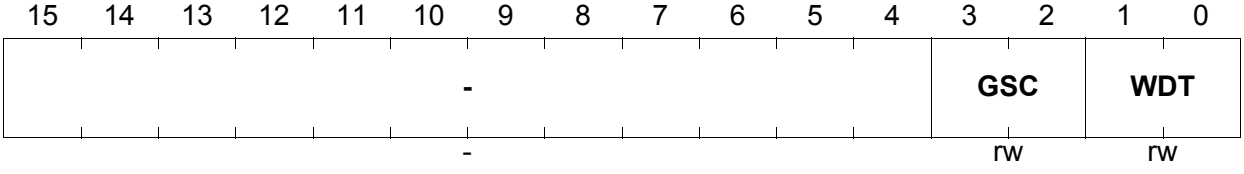
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WU		WUT		PVC12		PVC11		PVCM2		PVCM1		SWD2		SWD1	
rw		rw		rw		rw		rw		rw		rw		rw	

Field	Bits	Type	Description
SWD1	[1:0]	rw	**Interrupt Node Pointer for SWD 1 Interrupts** This bit field defines the interrupt node, which is requested due to the set condition for bit INTSTAT.SWDI1 (if enabled by bit INTDIS.SWDI1). 00_B Interrupt node $6C_H$ is selected 01_B Interrupt node $6B_H$ is selected 10_B Reserved, do not use this combination 11_B Reserved, do not use this combination
SWD2	[3:2]	rw	**Interrupt Node Pointer for SWD 2 Interrupts** This bit field defines the interrupt node, which is requested due to the set condition for bit INTSTAT.SWDI2 (if enabled by bit INTDIS.SWDI2). 00_B Interrupt node $6C_H$ is selected 01_B Interrupt node $6B_H$ is selected 10_B Reserved, do not use this combination 11_B Reserved, do not use this combination
PVCM1	[5:4]	rw	**Interrupt Node Pointer for PVC_M 1 Interrupts** This bit field defines the interrupt node, which is requested due to the set condition for bit INTSTAT.PCVMI1 (if enabled by bit INTDIS.PVCMI1). 00_B Interrupt node $6C_H$ is selected 01_B Interrupt node $6B_H$ is selected 10_B Reserved, do not use this combination 11_B Reserved, do not use this combination

Field	Bits	Type	Description
PVCM2	[7:6]	rw	**Interrupt Node Pointer for PVC_M 2 Interrupts** This bit field defines the interrupt node, which is requested due to the set condition for bit INTSTAT.PCVMI2 (if enabled by bit INTDIS.PVCMI2). 00_B Interrupt node $6C_H$ is selected 01_B Interrupt node $6B_H$ is selected 10_B Reserved, do not use this combination 11_B Reserved, do not use this combination
PVC11	[9:8]	rw	**Interrupt Node Pointer for PVC_1 1 Interrupts** This bit field defines the interrupt node, which is requested due to the set condition for bit INTSTAT.PCV1I1 (if enabled by bit INTDIS.PVC1I1). 00_B Interrupt node $6C_H$ is selected 01_B Interrupt node $6B_H$ is selected 10_B Reserved, do not use this combination 11_B Reserved, do not use this combination
PVC12	[11:10]	rw	**Interrupt Node Pointer for PVC_1 2 Interrupts** This bit field defines the interrupt node, which is requested due to the set condition for bit INTSTAT.PCV1I2 (if enabled by bit INTDIS.PVC1I2). 00_B Interrupt node $6C_H$ is selected 01_B Interrupt node $6B_H$ is selected 10_B Reserved, do not use this combination 11_B Reserved, do not use this combination
WUT	[13:12]	rw	**Interrupt Node Pointer for WU Trim Interrupts** This bit field defines the interrupt node, which is requested due to the set condition for bit INTSTAT.WUTI (if enabled by bit INTDIS.WUTI). 00_B Interrupt node $6C_H$ is selected 01_B Interrupt node $6B_H$ is selected 10_B Reserved, do not use this combination 11_B Reserved, do not use this combination
WU	[15:14]	rw	**Interrupt Node Pointer for WU Interrupts** This bit field defines the interrupt node, which is requested due to the set condition for bit INTSTAT.WUI (if enabled by bit INTDIS.WUI). 00_B Interrupt node $6C_H$ is selected 01_B Interrupt node $6B_H$ is selected 10_B Reserved, do not use this combination 11_B Reserved, do not use this combination

INTNP1
Interrupt Node Pointer 1 Register

SFR ($FE88_H/44_H$) Reset Value: 0001_H



6.9.3.6

Field	Bits	Type	Description
WDT	[1:0]	rw	**Interrupt Node Pointer for WDT Interrupts** This bit field defines the interrupt node, which is requested due to the set condition for bit INTSTAT.WDTI (if enabled by bit INTDIS.WDTI). 00_B Interrupt node $6C_H$ is selected 01_B Interrupt node $6B_H$ is selected 10_B Reserved, do not use this combination 11_B Reserved, do not use this combination
GSC	[3:2]	rw	**Interrupt Node Pointer for GSC Interrupts** This bit field defines the interrupt node, which is requested due to the set condition for bit INTSTAT.GSCI (if enabled by bit INTDIS.GSCI). 00_B Interrupt node $6C_H$ is selected 01_B Interrupt node $6B_H$ is selected 10_B Reserved, do not use this combination 11_B Reserved, do not use this combination

6.9.3.7 Register DMPMIT

This register contains additional sticky interrupt and trap flags within the DMP_M power domain.

DMPMIT
DMP_M Interrupt and Trap Trigger Register

SFR ($FE96_H/4B_H$) **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RAT	0	ESR2T	ESR1T	ESR0T	0		GSC	WUI	WUTI	PVC1I2	PVC1I1	PVCMI2	PVCMI1	SWDI2	SWDI1
rh	r	rh	rh	rh	r		rh	rh	rh	rh	rh	rh	rh	rh	rh

Field	Bits	Type	Description
SWDI1	0	rh	**SWD Interrupt Request Flag 1** This bit is set if bit SWDCON0.L1OK is cleared and SWDCON0.L1ACON = 01_B and INTDIS.SWDI1 = 0. 0_B No SWDI1 interrupt was requested since this bit was cleared the last time 1_B A SWDI1 interrupt was requested since this bit was cleared the last time
SWDI2	1	rh	**SWD Interrupt Request Flag 2** This bit is set if bit SWDCON0.L2OK is cleared and SWDCON0.L2ACON = 01_B and INTDIS.SWDI2 = 0. 0_B No SWDI2 interrupt was requested since this bit was cleared the last time 1_B A SWDI2 interrupt was requested since this bit was cleared the last time
PVCMI1	2	rh	**PVC_M Interrupt Request Flag 1** This bit is set if bit PVCMCON0.LEV1OK is cleared and PVCMCON0.L1INTEN = 1_B and INTDIS.PVCMI1 = 0. 0_B No PVCMI1 interrupt was requested since this bit was cleared the last time 1_B A PVCMI1 interrupt was requested since this bit was cleared the last time

Field	Bits	Type	Description
PVCMI2	3	rh	**PVC_M Interrupt Request Flag 2** This bit is set if bit PVCMCON0.LEV2OK is cleared and PVCMCON0.L2INTEN = 1_B and INTDIS.PVCMI2 = 0. 0_B No PVCMI2 interrupt was requested since this bit was cleared the last time 1_B A PVCMI2 interrupt was requested since this bit was cleared the last time
PVC1I1	4	rh	**PVC_1 Interrupt Request Flag 1** This bit is set if bit PVC1CON0.LEV1OK is cleared and PVC1CON0.L1INTEN = 1_B and INTDIS.PVC1I1 = 0. 0_B No PVC1I1 interrupt was requested since this bit was cleared the last time 1_B A PVC1I1 interrupt was requested since this bit was cleared the last time
PVC1I2	5	rh	**PVC_1 Interrupt Request Flag 2** This bit is set if bit PVC1CON0.LEV2OK is cleared and PVC1CON0.L2INTEN = 1_B and INTDIS.PVC1I2 = 0. 0_B No PVC1I2 interrupt was requested since this bit was cleared the last time 1_B A PVC1I2 interrupt was requested since this bit was cleared the last time
WUTI	6	rh	**Wake-up Trim Interrupt Request Flag** This bit is set if a wake-up trim trigger occurs and INTDIS.WUTI = 0. 0_B No WUT interrupt was requested since this bit was cleared the last time 1_B A WUT interrupt was requested since this bit was cleared the last time
WUI	7	rh	**Wake-up Interrupt Request Flag** This bit is set if a wake-up trigger occurs and INTDIS.WUI = 0. 0_B No WU interrupt was requested since this bit was cleared the last time 1_B A WU interrupt was requested since this bit was cleared the last time

Field	Bits	Type	Description
GSC	8	rh	**GSC Interrupt Request Flag** This bit is set if a GSC trigger occurs and INTDIS.GSCI = 0. 0_B No GSC interrupt was requested since this bit was cleared the last time 1_B A GSC interrupt was requested since this bit was cleared the last time
ESR0T	11	rh	**$\overline{\text{ESR0}}$ Trap Request Flag** This bit is set if pin $\overline{\text{ESR0}}$ is asserted. 0_B No ESR0 trap was requested since this bit was cleared the last time 1_B An ESR0 trap was requested since this bit was cleared the last time
ESR1T	12	rh	**$\overline{\text{ESR1}}$ Trap Request Flag** This bit is set if pin $\overline{\text{ESR1}}$ is asserted. 0_B No ESR1 trap was requested since this bit was cleared the last time 1_B An ESR1 trap was requested since this bit was cleared the last time
ESR2T	13	rh	**$\overline{\text{ESR2}}$ Trap Request Flag** This bit is set if pin $\overline{\text{ESR2}}$ is asserted. 0_B No ESR2 trap was requested since this bit was cleared the last time 1_B An ESR2 trap was requested since this bit was cleared the last time
RAT	15	rh	**Register Access Trap Request Flag** This bit is set a protected register is written by an non-authorized access. 0_B No RA trap was requested since this bit was cleared the last time 1_B A RA trap was requested since this bit was cleared the last time
0	[10:9], 14	r	**Reserved** Read as 0; should be written with 0.

6.9.3.8 Register DMPMITCLR

This register contains the software clear option for all sticky status flags of all interrupt and trap request trigger sources of the DMP_M power domain.

DMPMITCLR
DMP_M Interrupt and Trap Clear Register

SFR ($FE98_H/4C_H$) **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RAT	0	ESR2T	ESR1T	ESR0T	0		GSC	WUI	WUTI	PVC1I2	PVC1I1	PVCMI2	PVCMI1	SWDI2	SWDI1
w	r	w	w	w	r		w	w	w	w	w	w	w	w	w

6.9.3.9

Field	Bits	Type	Description
SWDI1	0	w	**Clear SWD1 Interrupt Request Flag 1** 0_B No action 1_B Bit DMPMIT.SWDI1 is cleared
SWDI2	1	w	**Clear SWD Interrupt Request Flag 2** 0_B No action 1_B Bit DMPMIT.SWDI2 is cleared
PVCMI1	2	w	**Clear PVC_M Interrupt Request Flag 1** 0_B No action 1_B Bit DMPMIT.PVCMI1 is cleared
PVCMI2	3	w	**Clear PVC_M Interrupt Request Flag 2** 0_B No action 1_B Bit DMPMIT.PVCMI2 is cleared
PVC1I1	4	w	**Clear PVC_1 Interrupt Request Flag 1** 0_B No action 1_B Bit DMPMIT.PVC1I1 is cleared
PVC1I2	5	w	**Clear PVC_1 Interrupt Request Flag 2** 0_B No action 1_B Bit DMPMIT.PVC1I2 is cleared
WUTI	6	w	**Clear Wake-up Trim Interrupt Request Flag** 0_B No action 1_B Bit DMPMIT.WUTI is cleared
WUI	7	w	**Clear Wake-up Interrupt Request Flag** 0_B No action 1_B Bit DMPMIT.WUI is cleared

Field	Bits	Type	Description
GSC	8	w	**Clear GSC Interrupt Request Flag** 0_B No action 1_B Bit DMPMIT.GSCI is cleared
ESR0T	11	w	**Clear $\overline{\text{ESR0}}$ Trap Request Flag** 0_B No action 1_B Bit DMPMIT.ESR0T is cleared
ESR1T	12	w	**Clear $\overline{\text{ESR1}}$ Trap Request Flag** 0_B No action 1_B Bit DMPMIT.ESR1T is cleared
ESR2T	13	w	**Clear $\overline{\text{ESR2}}$ Trap Request Flag** 0_B No action 1_B Bit DMPMIT.ESR2T is cleared
RAT	15	w	**Clear Register Access Trap Request Flag** 0_B No action 1_B Bit DMPMIT.RAT is cleared
0	[10:9], 14	r	**Reserved** Read as 0; should be written with 0.

6.9.3.10 Alternate Interrupt Source Assignment

In order to map the interrupt request sources in the complete system to the available interrupt nodes, interrupt nodes are shared between selected modules.

ISSR
Interrupt Source Select Register

SFR ($FF2E_H/97_H$) **Reset Value: 0000_H**

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

Field	Bits	Type	Description
ISS0	0	rw	**Interrupt Source Select for CC2_CC16IC** 0_B CC2 channel 16 interrupt is selected 1_B External interrupt request ERU_IOUT0 is selected
ISS1	1	rw	**Interrupt Source Select for CC2_CC17IC** 0_B CC2 channel 17 interrupt is selected 1_B External interrupt request ERU_IOUT1 is selected
ISS2	2	rw	**Interrupt Source Select for CC2_CC18IC** 0_B CC2 channel 18 interrupt is selected 1_B External interrupt request ERU_IOUT2 is selected
ISS3	3	rw	**Interrupt Source Select for CC2_CC19IC** 0_B CC2 channel 19 interrupt is selected 1_B External interrupt request ERU_IOUT3 is selected
ISS4	4	rw	**Interrupt Source Select for CC2_CC20IC** 0_B CC2 channel 20 interrupt is selected 1_B USIC0 channel 0 SR3 is selected
ISS5	5	rw	**Interrupt Source Select for CC2_CC21IC** 0_B CC2 channel 21 interrupt is selected 1_B USIC0 channel 1 SR3 is selected
ISS6	6	rw	**Interrupt Source Select for CC2_CC22IC** 0_B CC2 channel 22 interrupt is selected 1_B USIC1 channel 0 SR3 is selected
ISS7	7	rw	**Interrupt Source Select for CC2_CC23IC** 0_B CC2 channel 23 interrupt is selected 1_B USIC1 channel 1 SR3 is selected

Field	Bits	Type	Description
ISS8	8	rw	**Interrupt Source Select for CC2_CC24IC** 0_B CC2 channel 24 interrupt is selected 1_B External interrupt request ERU_IOUT0 is selected
ISS9	9	rw	**Interrupt Source Select for CC2_CC25IC** 0_B CC2 channel 25 interrupt is selected 1_B External interrupt request ERU_IOUT1 is selected
ISS10	10	rw	**Interrupt Source Select for CC2_CC26IC** 0_B CC2 channel 26 interrupt is selected 1_B External interrupt request ERU_IOUT2 is selected
ISS11	11	rw	**Interrupt Source Select for CC2_CC27IC** 0_B CC2 channel 27 interrupt is selected 1_B External interrupt request ERU_IOUT3 is selected
ISS12	12	rw	**Interrupt Source Select for CC2_CC28IC** 0_B CC2 channel 28 interrupt is selected 1_B USIC2 channel 0 SR3 is selected
ISS13	13	rw	**Interrupt Source Select for CC2_CC29IC** 0_B CC2 channel 29 interrupt is selected 1_B USIC2 channel 1 SR3 is selected
ISS14	14	rw	**Interrupt Source Select for CC2_CC30IC** 0_B CC2 channel 30 interrupt is selected 1_B SCU Interrupt 2 is selected
ISS15	15	rw	**Interrupt Source Select for CC2_CC31IC** 0_B CC2 channel 31 interrupt is selected 1_B SCU Interrupt 3 is selected

6.10 Temperature Compensation Unit

The temperature compensation for the port drivers provides driver output characteristics which are stable (within a certain band of parameter variation) over the specified temperature range.

The temperature compensation oscillator (sensor) provides a reference clock from a free-running temperature-dependent oscillator. An enable trigger is used to define counting cycles where the reference clock pulses are accumulated to build the sensor value TCLR.THCOUNT. The enable trigger is derived from the system clock by a prescaler and a programmable divider (see **Figure 6-27**). The value for the programmable divider must be written by the user according to the selected system frequency.

After the count cycle, the resulting count value, i.e. the number of reference clock cycles, is copied to bit field TCLR.THCOUNT. Thus, TCLR.THCOUNT is updated after every count cycle while the temperature compensation is enabled.

Software can compare the temperature-related count value (TCLR.THCOUNT) to several thresholds (temperature levels) in order to determine the control values TCCR.TCC.

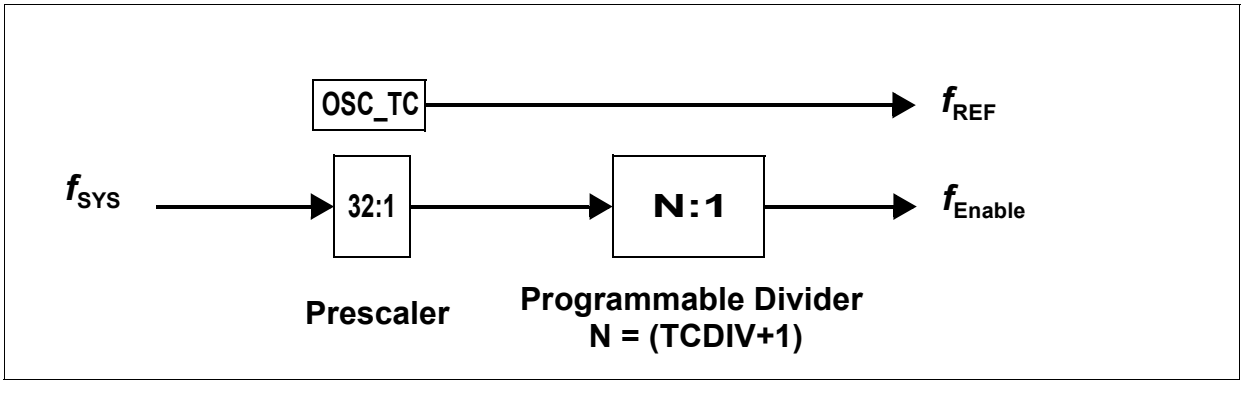


Figure 6-27 Temperature Compensation Clock Generation

The clock divider is programmed via bit field TCCR.TCDIV. The value that should be used for bit field TCCR.TCDIV can be calculated using the following formula documented in the data sheet.

Generally, temperature compensation is a user-controlled feature. The Temperature Compensation Control Register TCCR provides access to the actual compensation value (generated by the sensor) and allows software control of the pads. During operation the device (i.e. the pads) can be controlled by the value of the on-chip sensor, or by externally provided compensation values. Register TCCR also provides the programmable divider value.

Note: The relation between the counter value and the temperature can differ between two devices and need to be evaluated for each device individually.

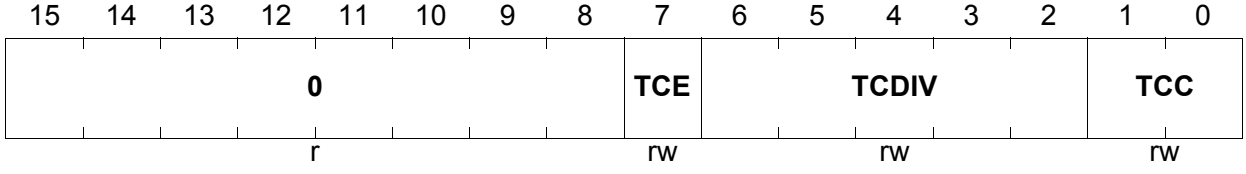
Note: The temperature compensation circuit does not generate temperature compensation values continously. The idea is, that the SW frequently updates the pad control with the value currently found in the tempcomp register (e.g. by an interrupt generated by a timer). Since temparature is a continous function it is not relevant, whether the temperature value read is new or the value of a previous meaurement.

6.10.1 Temperature Compensation Registers

6.10.1.1 TCCR

This register contains the control options.

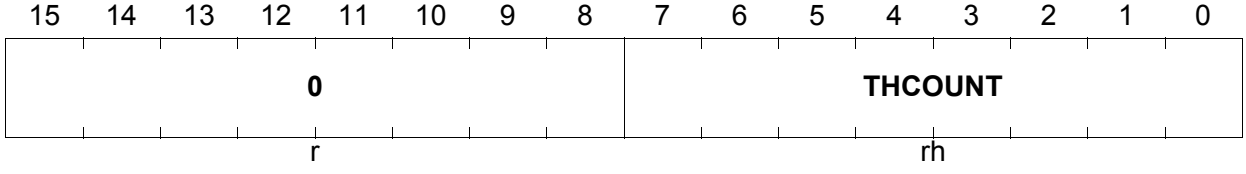
TCCR
Temperature Compensation Control Register
ESFR ($F1AC_H$/$D6_H$) **Reset Value: 0003_H**



Field	Bits	Type	Description
TCC	[1:0]	rw	**Temperature Compensation Control** The value which controls the temperature compensation inputs of the pads. 00_B Maximum reduction = min. driver strength, i.e. very low temperature 11_B No reduction = max. driver strength, i.e. very high temperature
TCDIV	[6:2]	rw	**Temperature Compensation Clock Divider** This value adjusts the temperature compensation logic to the selected operating frequency.
TCE	7	rw	**Temperature Compensation Enable** 0_B No action 1_B Enable counting to generate new temperature values. Clearing this bit also stops the temperature compensation oscillator.
0	[15:8]	r	**Reserved** Read as 0; should be written with 0.

TCLR
Temperature Comp. Level Register

ESFR ($F0AC_H/56_H$) **Reset Value: 0000_H**



Field	Bits	Type	Description
THCOUNT	[7:0]	rh	**Threshold Counter** Returns the result of the most recent count cycle of the temperature sensor, to be compared with the thresholds.
0	[15:8]	r	**Reserved** Read as 0; should be written with 0.

Note: The threshold counter will not overflow but rather stop at count 255.

6.11 Watchdog Timer (WDT)

The following part describes the Watchdog Timer (WDT) and its functionality.

6.11.1 Introduction

The Watchdog Timer (WDT) is a secure mechanism to overcome life- and dead-locks. An enabled WDT generates a reset for the system if not serviced in a configured time frame.

Features

The following list is a summary of the WDT functions:

- 16-bit Watchdog Timer
- Selectable operating frequency: f_{IN} / 256 or f_{IN} / 16384
- Timer overflow error detection
- Individual disable for timer functionality
- Double Reset Detection

Figure 6-28 provides an overview on the registers of the Watchdog Timer.

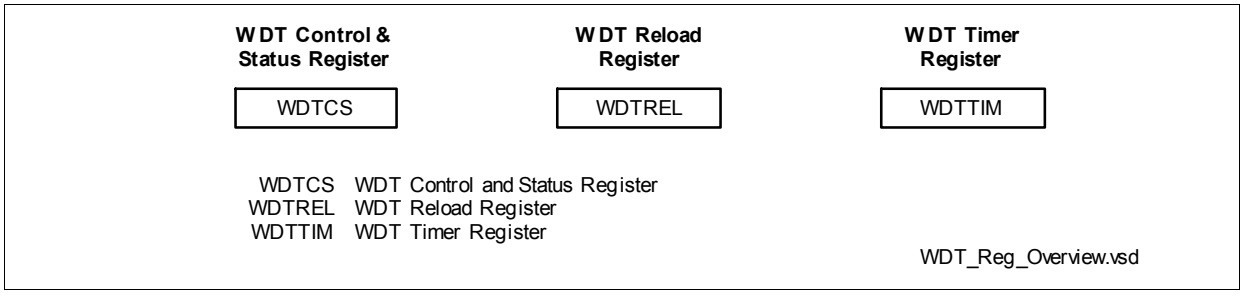


Figure 6-28 Watchdog Timer Register Overview

6.11.2 Overview

The Watchdog Timer (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 XE16x in a user-specified time period. When enabled, the WDT will cause the XE16x 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 WDT reset request trigger. Hence, regular service of the WDT confirms that the system is functioning properly.

A further feature of the Watchdog Timer is its reset prewarning operation. Instead of immediately resetting the device on the detection of an error, a prewarning output is given to the system via an interrupt request. This makes it possible to bring the system into a defined and predictable status, before the reset is finally issued.

6.11.3 Functional Description

The following part describes all functions of the WDT.

6.11.3.1 Timer Operation

The timer is enabled when instruction ENWDT (Enable Watchdog Timer) is executed correctly.

The WDT uses the input clock f_{IN} which is equal to the system clock f_{sys}. A clock divider in front of the WDT timer provides two output frequencies, f_{IN} / 256 and f_{IN} / 16384. The selection of the counting rate is done via bit **WDTCS**.IR.

WDT Periods

The general formula to calculate a Watchdog period is:

$$\text{period} = \frac{\left(2^{16} - \text{startvalue}\right) \cdot 256 \cdot 2^{(1-\text{IR}) \cdot 6}}{f_{IN}} \quad (6.4)$$

The parameter <startvalue> represents either the user-programmable reload value WDTREL.RELV (default value $FFFC_H$) for the calculation of the period in Normal Mode or the fixed value $FFFF_H$ for the calculation of the period in Prewarning Mode.

WDT Timer Reload

The counter is reloaded and the prescaler is cleared when one of the following conditions occurs:

- A successful access to register WDTREL
- The WDT is serviced via instruction SRVWDT
- A WDT overflow condition (Prewarning Mode is entered).
  The different reload value for the counter in Prewarning Mode is $FFFF_H$.
- The Disable Mode is entered (when instruction DISWDT is executed)
- Upon any reset

6.11.3.2 Timer Modes

The Watchdog Timer provides following modes:

- Normal Mode
- Disable Mode
- Prewarning Mode

Figure 6-29 provides a state diagram of the different Timer Modes and the transition possibilities. Please refer to the description of the conditions for changing from one mode to the other.

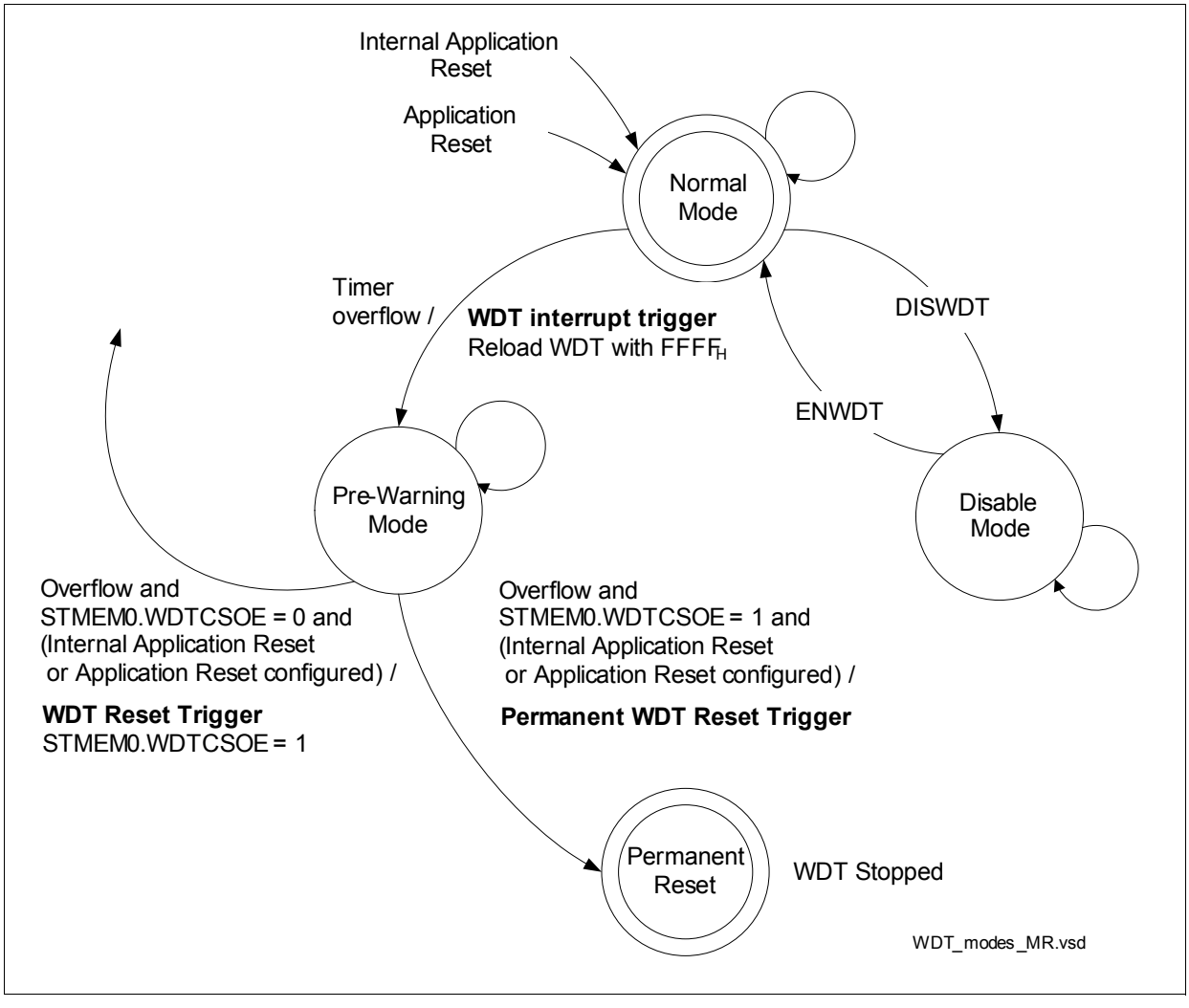


Figure 6-29 State Diagram of the Timer Modes

Normal Mode

Normal Mode is the default mode after an Application Resetor or an Internal Application Reset. Normal Mode can be entered from Disable Mode only when instruction ENWDT is executed.

The timer is loaded with RELV when the Normal Mode is entered, and it starts counting upwards. After reset the timer is loaded with $FFFC_H$ (default value of RELV).

It has to be serviced before the counter overflows. Servicing is performed by the CPU via instructions SRVWDT and/or ENWDT.

If the WDT is not serviced before the timer overflows, a system malfunction is assumed, and following operations are done:

- An WDT interrupt trigger is issued
- Prewarning Mode is entered

- Timer is reloaded with $FFFF_H$

Disable Mode

Disable Mode is provided for applications that do not require the Watchdog Timer function. Disable Mode is entered when instruction DISWDT is executed, either before End-of-Init, if CPUCON1.WDTCTL = 0, or at any time, if CPUCON1.WDTCTL = 1.

The timer is reloaded with the value of WDTREL.RELV when Disable Mode is entered.

A transition from Disable Mode to Normal Mode is performed when instruction ENWDT is executed while CPUCON1.WDTCTL = 1.

Prewarning Mode

Prewarning Mode is entered always when a Watchdog error is detected. This is an overflow in Normal Mode. Instead of immediately requesting a reset of the device, the WDT enables the system to enter a secure state by issuing the prewarning output before the reset occurs. Receiving the prewarning, the CPU and the system are requested to finish all pending transaction requests and to not generate new ones. The prewarning is signalled via an interrupt. The CPU can recognize the WDT prewarning interrupt via register INTSTAT. After finishing all pending transactions, the CPU should execute the IDLE instruction to stop all further processing before the coming reset.

In Prewarning Mode, the WDT starts counting from $FFFF_H$ upwards, and then requests a WDT reset on the overflow of the WDT from $FFFF_H$ to 0000_H. A reset request of the type as configured in **RSTCON1**.WDT can not be avoided. No reset will be requested If RSTCON1.WDT is cleared. The WDT does not react anymore to accesses to its registers and to the ENWDT or DISWDT instruction, nor will it change its state until it is reset.

A further feature of the WDT detects double errors and sets the whole system into a permanent WDT reset. This feature prevents the XE16x from executing random wrong code for longer than the occurence of the overflow, and prevents the XE16x from being repeatedly reset by the WDT.

Double WDT Reset

If the Watchdog induced reset (Application or Internal Application Reset) occurs twice back-to-back, a severe system malfunction is assumed and the XE16x is held in a reset of the type as configured in **RSTCON1**.WDT (or just not) until a Power-on 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.

Note: Triggering a PORST upon a WDT reset will never result in a double WDT overflow.

If the WDT is configured by RSTCON1.WDT to request an Application Reset or an Internal Application Reset the second reset request will be permanently asserted

resulting (without any change in the reset configuration) in a permanent reset of the type configured by RSTCON1.WDT.

The information about the first WDT reset request is stored in bit **STMEM0**.WDTCSOE (see **Section 10.3.1**). The bit is set when a WDT overflow had occured in Prewarning Mode and a reset was generated. If the bit STMEM0.WDTCSOE is already set then a double WDT reset event has occurred and a permanent reset request is generated.

The bit is cleared by any Power-on Reset or when bit STMEM0.WDTCSOE is cleared. A correct service of the WDT does not clear this bit nor do any access to the WDT related registers or commands. Therefore, if correct WDT-servicing has been done after the first WDT reset and a next WDT reset must not immediately lead to a double error state, application software has to clear STMEM0.WDTCSOE, too.

Note: After the double WDT reset request trigger is generated the counter is stopped after the overflow.

Port Configuration during WDT Reset

The behavior of the ESRx ports can be defined with respect to the reset type by bit field ESRCFGx.PC. For the coding of PC see **Table 6-6**. The allows to signal the occurence of a reset.

The configuration of the GPIOS ports depends on the reset type. In case of an Application Reset the pad configuration is unchanged[1)], in case of an Internal Application Reset the ports are configured for input.

6.11.3.3 Suspend Mode Support

In an enabled and active debug session, the Watchdog functionality can lead to unintended resets. Therefore, to avoid these resets, the OCDS can control whether the WDT is enabled or disabled (default after reset). This is done via bit CBS_IOSR.DB.

Table 6-14 OCDS Behavior of WDT

WDTCS.DS	CBS_DBGSR.DBGEN	CBS_IOSR.DB	WDT Action
1	X	X	Stopped
0	0	X	Running
0	1	0	Stopped
0	1	1	Running

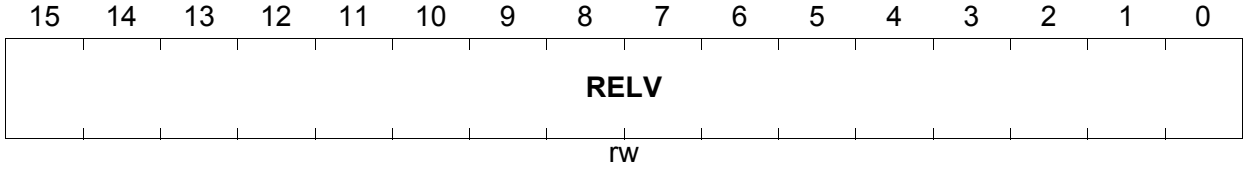
1) Ports P2.[2:0] and P10.[12:0] are set to input during the execution of the Application Reset.

6.11.4 WDT Kernel Registers

6.11.4.1 WDT Reload Register

This register defines the WDT reload value.

WDTREL
WDT Reload Register **ESFR ($F0C8_H/64_H$)** **Reset Value: $FFFC_H$**



Field	Bits	Type	Description
RELV	[15:0]	rw	**Reload Value for the Watchdog Timer** This bit field defines the reload value for the WDT.

6.11.4.2 WDT Control and Status Register

The Control and Status Register can only be accessed in Secured Mode.

WDTCS
WDT Control and Status Register

ESFR ($F0C6_H/63_H$) **Reset Value: 0000_H**

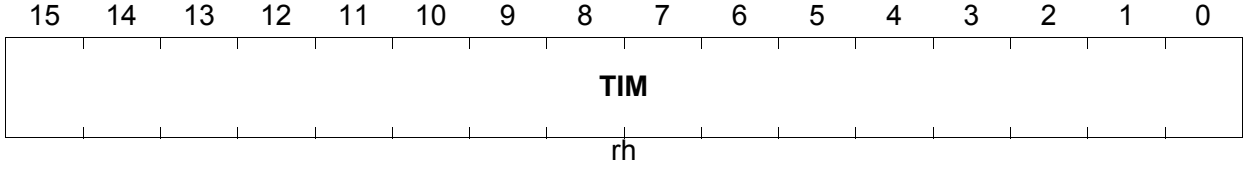
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							IR	0					PR	DS	OE
r							rw	r					rh	rh	rh

Field	Bits	Type	Description
OE	0	rh	**Overflow Error Status Flag** 0_B No WDT overflow error 1_B A WDT overflow error has occurred. This bit is set by hardware when the Watchdog Timer overflows from $FFFF_H$ to 0000_H. This bit is only cleared through: • any Power-on Reset • an executed SRVWDT or ENWDT instruction *Note: It is not possible to clear this bit in Prewarning Mode with the SRVWDT or ENWDT instruction.*
DS	1	rh	**Timer Enable/Disable Status Flag** 0_B Timer is enabled (default after reset) 1_B Timer is disabled This bit is cleared when instruction ENWDT was executed and CPUCON1.WDTCTL = 1. This bit is set when instruction DISWDT was executed before EINIT or CPUCON1.WDTCTL = 1. *Note: ENWDT and DISWDT instruction will be reflected in this bit but in Prewarning Mode the WDT mode is not changed.*
PR	2	rh	**Prewarning Mode Flag** 0_B Normal Mode (default after reset) 1_B Prewarning Mode

Field	Bits	Type	Description
IR	8	rw	**Input Frequency Request Bit** 0_B Request to set input frequency to f_{IN} / 16384 1_B Request to set input frequency to f_{IN} / 256 An update of this bit is taken into account after the next successful execution of instruction SRVWDT or ENWDT, on a write to register WDTREL, and always when the WDT is in Disable Mode.
0	[7:3], [15:9]	r	**Reserved** Read as 0; should be written with 0;

6.11.4.3 WDT Timer Register

WDTTIM
WDT Timer Register **ESFR ($F0CA_H/65_H$)** **Reset Value: $FFFC_H$**



Field	Bits	Type	Description
TIM	[15:0]	rh	**Timer Value** Reflects the current contents of the Watchdog Timer.

6.12 SCU Trap Generation

The basic trap structure of the SCU is shown in **Figure 6-30**.

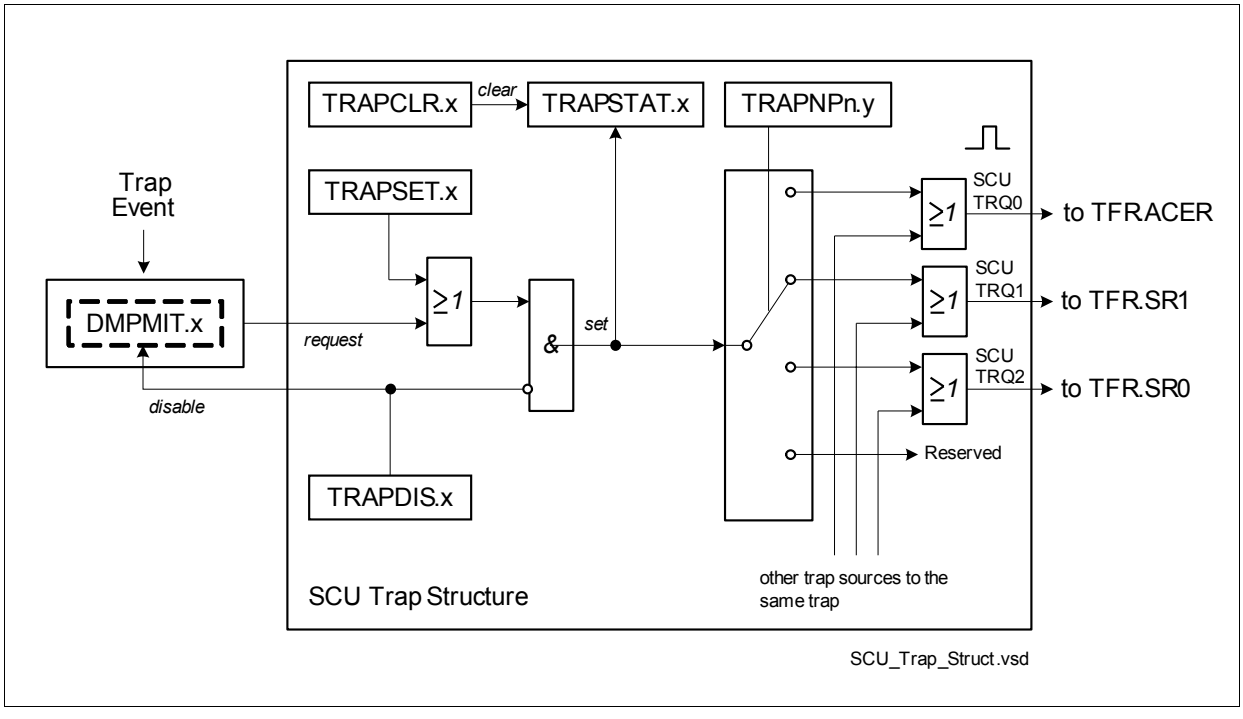


Figure 6-30 SCU Trap Structure

If enabled by the corresponding bit in register **TRAPDIS**, a trap is triggered either by a pulse on the incoming trap line, or by a software set of the respective bit in register **TRAPSET**. The trigger sets the respective flag in register **TRAPSTAT** and is gated to one of the trap nodes, selected by the node pointer register **TRAPNP**.

The trap flag in register TRAPSTAT can be cleared by software by writing to the corresponding bit in register **TRAPCLR**.

If more than one trap source is connected to the same trap (via register TRAPNP), the requests are combined to one common line.

Trap Node Assignment

The trap sources of the system can be mapped to three trap nodes by programming the trap node pointer register TRAPNP. The default assignment of the trap sources to the nodes and their corresponding control register is listed in **Table 6-15**.

6.12.1 Trap Support

Some of the trap requests are first fed through a sticky flag register in the DMP_M domain. These flags are set with a trigger and if set trigger the trap generation in the

DMP_1. . Please note that the disable control of register TRAPDIS also influences the sticky bit in register **DMPMIT** (see **Section 6.9.3.7**).

Which of the trap requests have a sticky flag in register DMPMIT is listed in **Table 6-15**.

Note: When servicing an SCU trap request, make sure that all related request flags are cleared after the identified request has been handled. To clear a trap request that is stored in register DMPMIT, first clear the request source of the source, clear the request within DMP_M via DMPMITCLR, and then clear the request within DMP_1 via TRAPCLR.

6.12.2 SCU Trap Sources

The SCU receives the trap lines listed in **Table 6-15**.

Table 6-15 SCU Trap Request Overview

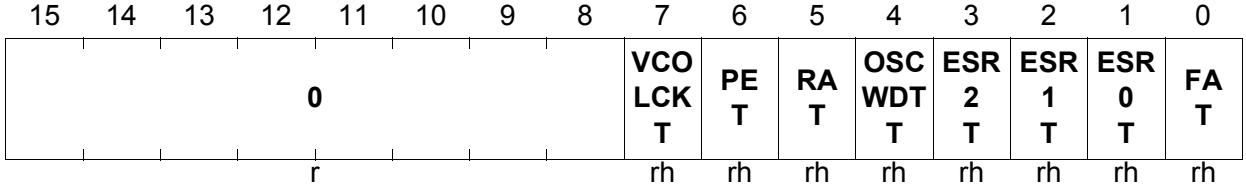
Source of Trap	Short Name	Sticky Flag in DMPMIT	Default Trap Flag Assignment in Register TFR
Flash Access Trap	FAT	---	TFR.ACER (SCU_TRQ0)
$\overline{\text{ESR0}}$ Trap	ESR0T	yes	TFR.SR1 (SCU_TRQ1)
$\overline{\text{ESR1}}$ Trap	ESR1T	yes	TFR.SR1 (SCU_TRQ1)
$\overline{\text{ESR2}}$ Trap	ESR2T	yes	TFR.SR1 (SCU_TRQ1)
PLL Trap	OSCWDTT	---	TFR.SR1 (SCU_TRQ1)
Register Access Trap	RAT	yes	TFR.ACER (SCU_TRQ0)
Parity Error Trap	PET	---	TFR.ACER (SCU_TRQ0)
VCO Lock Trap	VCOLCKT	---	TFR.SR0 (SCU_TRQ2)

6.12.3 SCU Trap Control Registers

6.12.3.1 Register TRAPSTAT

This register contains the status flags for all trap request trigger sources of the SCU. For setting and clearing of these status bits by software see registers TRAPSET and TRAPCLR, respectively.

TRAPSTAT
Trap Status Register **SFR ($FF02_H/81_H$)** **Reset Value: 0000_H**



Field	Bits	Type	Description
FAT	0	rh	**Flash Access Trap Request Flag** TRAPSTAT.FAT is set when a flash access violation occurs and TRAPDIS.FAT = 0. 0_B No FA trap trigger has occured since this bit was cleared the last time 1_B A FA trap trigger has occured since this bit was cleared the last time
ESR0T	1	rh	**ESR0 Trap Request Flag** TRAPSTAT.ESR0T is set when bit DMPMIT.ESR0T is set and TRAPDIS.ESR0T = 0. 0_B No ESR0 trap trigger has occured since this bit was cleared the last time 1_B An ESR0 trap trigger has occured since this bit was cleared the last time
ESR1T	2	rh	**ESR1 Trap Request Flag** TRAPSTAT.ESR1T is set when bit DMPMIT.ESR1T is set and TRAPDIS.ESR1T = 0. 0_B No ESR1 trap trigger has occured since this bit was cleared the last time 1_B An ESR1 trap trigger has occured since this bit was cleared the last time

Field	Bits	Type	Description
ESR2T	3	rh	**ESR2 Trap Request Flag** TRAPSTAT.ESR2T is set when bit DMPMIT.ESR0T is set and TRAPDIS.ESR2T = 0. 0_B No ESR2 trap trigger has occured since this bit was cleared the last time 1_B An ESR2 trap trigger has occured since this bit was cleared the last time
OSCWDTT	4	rh	**OSCWDT Trap Request Flag** TRAPSTAT.OSCWDTT is set when an OSCWDT emergency event occurs and TRAPDIS.OSCWDTT = 0. 0_B No OSCWDT trap trigger has occured since this bit was cleared the last time 1_B An OSCWDT trap trigger has occured since this bit was cleared the last time
RAT	5	rh	**Register Access Trap Request Flag** TRAPSTAT.RAT is set when bit DMPMIT.RAT is set and TRAPDIS.RAT = 0. 0_B No RA trap trigger has occured since this bit was cleared the last time 1_B A RA trap trigger has occured since this bit was cleared the last time
PET	6	rh	**Parity Error Trap Request Flag** TRAPSTAT.PET is set when a memory parity error occurs and TRAPDIS.PET = 0. 0_B No PE trap trigger has occured since this bit was cleared the last time 1_B A PE trap trigger has occured since this bit was cleared the last time
VCOLCKT	7	rh	**VCOLCK Trap Request Flag** TRAPSTAT.VCOLCKT is set when a VCOLCK emergency event occurs and TRAPDIS.VCOLCKT = 0. 0_B No VCOLCK trap trigger has occured since this bit was cleared the last time 1_B A VCOLCK trap trigger has occured since this bit was cleared the last time
0	[15:8]	r	**Reserved** Read as 0; should be written with 0.

6.12.3.2 Register TRAPCLR

This register contains the software clear control for the trap status flags in register TRAPSTAT. Clearing a bit in this register has no effect, reading a bit always returns zero.

TRAPCLR
Trap Clear Register **SFR ($FE8E_H/47_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								VCOLCKT	PET	RAT	OSCWDTT	ESR2T	ESR1T	ESR0T	FAT
r								w	w	w	w	w	w	w	w

Field	Bits	Type	Description
FAT	0	w	**Clear Flash Access Trap Request Flag** 0_B No action 1_B Flag TRAPSTAT.FAT is cleared
ESR0T	1	w	**Clear ESR0 Trap Request Flag** 0_B No action 1_B Flag TRAPSTAT.ESR0T is cleared
ESR1T	2	w	**Clear ESR1 Trap Request Flag** 0_B No action 1_B Flag TRAPSTAT.ESR1T is cleared
ESR2T	3	w	**Clear ESR2 Trap Request Flag** 0_B No action 1_B Flag TRAPSTAT.ESR2T is cleared
OSCWDTT	4	w	**Clear OSCWDT Trap Request Flag** 0_B No action 1_B Flag TRAPSTAT.OSCWDTT is cleared
RAT	5	w	**Clear Register Access Trap Request Flag** 0_B No action 1_B Flag TRAPSTAT.RAT is cleared
PET	6	w	**Clear Parity Error Access Trap Request Flag** 0_B No action 1_B Flag TRAPSTAT.PET is cleared
VCOLCKT	7	w	**Clear VCOLCK Trap Request Flag** 0_B No action 1_B Flag TRAPSTAT.VCOLCKT is cleared

Field	Bits	Type	Description
0	[15:8]	r	**Reserved** Read as 0; should be written with 0

6.12.3.3 Register TRAPSET

This register contains the software set control for the trap status flags in register TRAPSTAT. Clearing a bit in this register has no effect, reading a bit always returns zero.

TRAPSET
Trap Set Register **SFR ($FE8C_H/46_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								VCOLCKT	PET	RAT	OSCWDTT	ESR2T	ESR1T	ESR0T	FAT
r								w	w	w	w	w	w	w	w

Field	Bits	Type	Description
FAT	0	w	**Set Flash Access Trap Request Flag** 0_B No action 1_B Flag TRAPSTAT.FAT is set
ESR0T	1	w	**Set ESR0 Trap Request Flag** 0_B No action 1_B Flag TRAPSTAT.ESR0T is set
ESR1T	2	w	**Set ESR1 Trap Request Flag** 0_B No action 1_B Flag TRAPSTAT.ESR1T is set
ESR2T	3	w	**Set ESR2 Trap Request Flag** 0_B No action 1_B Flag TRAPSTAT.ESR2T is set
OSCWDTT	4	w	**Set OSCWDT Trap Request Flag** 0_B No action 1_B Flag TRAPSTAT.OSCWDTT is set
RAT	5	w	**Set Register Access Trap Request Flag** 0_B No action 1_B Flag TRAPSTAT.RAT is set
PET	6	w	**Set Parity Error Access Trap Request Flag** 0_B No action 1_B Flag TRAPSTAT.PET is set
VCOLCKT	7	w	**Set VCOLCK Trap Request Flag** 0_B No action 1_B Flag TRAPSTAT.VCOLCKT is set

Field	Bits	Type	Description
0	[15:8]	r	**Reserved** Read as 0; should be written with 0.

6.12.3.4 Register TRAPDIS

This register contains the software disable control for all trap request trigger sources. Note that the bits ESRxT and RAT in this register also disable the setting of the respective flags in register DMPMIT (see **Section 6.9.1**).

TRAPDIS
Trap Disable Register **SFR ($FE90_H/48_H$)** **Reset Value: $009E_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0								VCOLCKT	PET	RAT	OSCWDTT	ESR2T	ESR1T	ESR0T	FAT
r								rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
FAT	0	rw	**Disable Flash Access Trap Request** 0_B FA trap request enabled 1_B FA trap request disabled
ESR0T	1	rw	**Disable ESR0 Trap Request** 0_B ESR0 trap request enabled 1_B ESR0 trap request disabled
ESR1T	2	rw	**Disable ESR1 Trap Request** 0_B ESR1 trap request enabled 1_B ESR1 trap request disabled
ESR2T	3	rw	**Disable ESR2 Trap Request** 0_B ESR2 trap request enabled 1_B ESR2 trap request disabled
OSCWDTT	4	rw	**Disable OSCWDT Trap Request** 0_B OSCWDT trap request enabled 1_B OSCWDT trap request disabled
RAT	5	rw	**Disable Register Access Trap Request** 0_B RA trap request enabled 1_B RA trap request disabled
PET	6	rw	**Disable Parity Error Trap Request** 0_B PE trap request enabled 1_B PE trap request disabled
VCOLCKT	7	rw	**Disable VCOLCK Trap Request** 0_B VCOLCK trap request enabled 1_B VCOLCK trap request disabled

Field	Bits	Type	Description
0	[15:8]	r	**Reserved** Read as 0; should be written with 0.

6.12.3.5 Register TRAPNP

This register contains the control for the trap node pointers of all SCU trap request trigger sources.

TRAPNP
Trap Node Pointer Register **SFR ($FE92_H/49_H$)** **Reset Value: 8254_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VCOLCK		PE		RA		OSCWDT		ESR2		ESR1		ESR0		FA	
rw		rw		rw		rw		rw		rw		rw		rw	

Field	Bits	Type	Description
FA	[1:0]	rw	**Trap Node Pointer for Flash Access Traps** TRAPNP.FA selects the trap request output for an enabled FAT trap request. 00_B Select request output SCU_TRQ0 (TFR.ACER) 01_B Select request output SCU_TRQ1 (TFR.SR1) 10_B Select request output SCU_TRQ2 (TFR.SR0) 11_B Reserved, do not use this combination
ESR0	[3:2]	rw	**Trap Node Pointer for ESR0 Traps** TRAPNP.ESR0 selects the trap request output for an enabled ESR0 trap request. 00_B Select request output SCU_TRQ0 (TFR.ACER) 01_B Select request output SCU_TRQ1 (TFR.SR1) 10_B Select request output SCU_TRQ2 (TFR.SR0) 11_B Reserved, do not use this combination
ESR1	[5:4]	rw	**Trap Node Pointer for ESR1 Traps** TRAPNP.ESR1 selects the trap request output for an enabled ESR1 trap request. 00_B Select request output SCU_TRQ0 (TFR.ACER) 01_B Select request output SCU_TRQ1 (TFR.SR1) 10_B Select request output SCU_TRQ2 (TFR.SR0) 11_B Reserved, do not use this combination

Field	Bits	Type	Description
ESR2	[7:6]	rw	**Trap Node Pointer for ESR2 Traps** TRAPNP.ESR2 selects the trap request output for an enabled ESR2 trap request. 00_B Select request output SCU_TRQ0 (TFR.ACER) 01_B Select request output SCU_TRQ1 (TFR.SR1) 10_B Select request output SCU_TRQ2 (TFR.SR0) 11_B Reserved, do not use this combination
OSCWDT	[9:8]	rw	**Trap Node Pointer for OSCWDT Traps** TRAPNP.OSCWDT selects the trap request output for an enabled OSCWDT trap request. 00_B Select request output SCU_TRQ0 (TFR.ACER) 01_B Select request output SCU_TRQ1 (TFR.SR1) 10_B Select request output SCU_TRQ2 (TFR.SR0) 11_B Reserved, do not use this combination
RA	[11:10]	rw	**Trap Node Pointer for Register Access Traps** TRAPNP.RA selects the trap request output for an enabled RAT trap request. 00_B Select request output SCU_TRQ0 (TFR.ACER) 01_B Select request output SCU_TRQ1 (TFR.SR1) 10_B Select request output SCU_TRQ2 (TFR.SR0) 11_B Reserved, do not use this combination
PE	[13:12]	rw	**Trap Node Pointer for Parity Error Traps** TRAPNP.PE selects the trap request output for an enabled PET trap request. 00_B Select request output SCU_TRQ0 (TFR.ACER) 01_B Select request output SCU_TRQ1 (TFR.SR1) 10_B Select request output SCU_TRQ2 (TFR.SR0) 11_B Reserved, do not use this combination
VCOLCK	[15:14]	rw	**Trap Node Pointer for VCOLCK Traps** TRAPNP.VCOLCK selects the trap request output for an enabled VCOLCK trap request. 00_B Select request output SCU_TRQ0 (TFR.ACER) 01_B Select request output SCU_TRQ1 (TFR.SR1) 10_B Select request output SCU_TRQ2 (TFR.SR0) 11_B Reserved, do not use this combination

6.13 Memory Content Protection

For supervising the content of the on-chip memories (Flash memory is not considered here) Parity Checking is provided.

6.13.1 Parity Error Handling

The on-chip RAM modules check parity information during read accesses and in case of an error a signal can be generated if enabled. These signals are combined and trigger a trap. If a parity error is detected during the trap handler routine a reset request trigger is generated. Register PECON controls the behavior of parity errors.

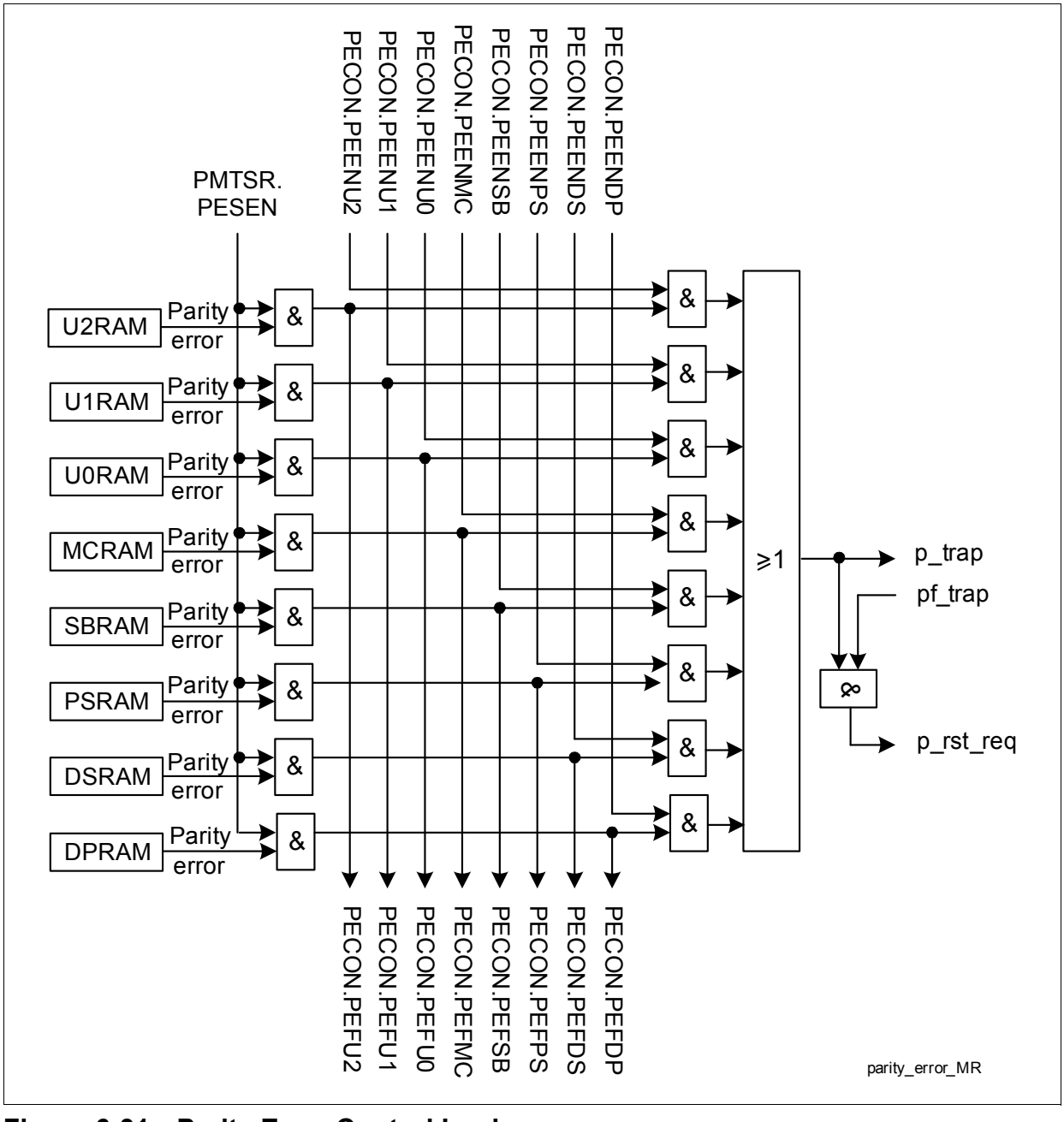


Figure 6-31 Parity Error Control Logic

A parity error, detected while the respective trap flag TFR.ACER is set, generates a reset request. The second error trap cannot be detected and handled by the CPU.

Note: The parity trap trigger should activate the Access Error trap (ACER).

The parity reset request trigger (*p_rst_req*) is generated when a parity error trap is request AND flag TFR.ACER is set.

6.13.1.1 Parity Software Testing Support

To support testing algorithms for the parity error trap routines a memory parity test logic is implemented for the C166SV2 subsystem memories (PSRAM, DSRAM, and DPRAM) and SBRAM.

The logic is controlled by registers PMTSR and PMTPR. Via bit field PMTPR.PWR a parity value can be writing to any address of every memory. The parity control software test update has to be enabled with bit PMTSR.MTEx for each memory individually. Otherwise a write to the parity control has no effect. With each read access to a memory the parity from the memory parity control is stored in register PMTPR.PRD.

The width and therefore the valid bits in register PMTPR is listed in **Table 6-16**.

Table 6-16 Memory Widths

Memory	Number of Parity Bits	Valid Bits in PWR/PRD
Dual Port (DP) Memory	2	PWR[1:0]/PRD[9:8]
Data SRAM (DS) Memory	2	PWR[1:0]/PRD[9:8]
Program SRAM (PS) Memory	8	PWR[7:0]/PRD[15:8]
Standby RAM (SB) Memory	2	PWR[1:0]/PRD[9:8]

Test software should be located in external memory and should be written in a way that no pre-fetching is performed.

6.13.1.2 Parity Error Registers

Register PECON

The following register controls the functional parity check mechanism.

PECON
Parity Error Control Register **ESFR ($F0C4_H/41_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PEFSB	**PEFMC**	**PEFU2**	**PEFU1**	**PEFU0**	**PEFPS**	**PEFDS**	**PEFDP**	**PEENSB**	**PEENMC**	**PEENU2**	**PEENU1**	**PEENU0**	**PEENPS**	**PEENDS**	**PEENDP**
rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rw	rw	rw	rw	rw	rw	rw	rw

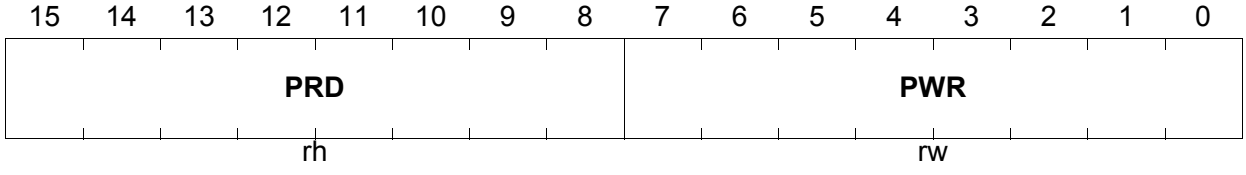
Field	Bits	Type	Description
PEENDP	0	rw	**Parity Error Trap Enable for Dual Port Memory** 0_B No Parity trap is requested for dual port memory parity errors 1_B A Parity trap is requested for dual port memory parity errors
PEENDS	1	rw	**Parity Error Trap Enable for Data SRAM** 0_B No Parity trap is requested for data SRAM parity errors 1_B A Parity trap is requested for data SRAM parity errors
PEENPS	2	rw	**Parity Error Trap Enable for Program SRAM** 0_B No Parity trap is requested for program SRAM parity errors 1_B A Parity trap is requested for program SRAM parity errors
PEENU0	3	rw	**Parity Error Trap Enable for USIC0 Memory** 0_B No Parity trap is requested for USIC0 memory parity errors 1_B A Parity trap is requested for USIC0 memory parity errors
PEENU1	4	rw	**Parity Error Trap Enable for USIC1 Memory** 0_B No Parity trap is requested for USIC1 memory parity errors 1_B A Parity trap is requested for USIC1 memory parity errors

Field	Bits	Type	Description
PEENU2	5	rw	**Parity Error Trap Enable for USIC2 Memory** 0_B No Parity trap is requested for USIC2 memory parity errors 1_B A Parity trap is requested for USIC2 memory parity errors
PEENMC	7	rw	**Parity Error Trap Enable for MultiCAN Memory** 0_B No Parity trap is requested for MultiCAN memory parity errors 1_B A Parity trap is requested for MultiCAN memory parity errors
PEENSB	8	rw	**Parity Error Trap Enable for Standby Memory** 0_B No Parity trap is requested for Standby memory parity errors 1_B A Parity trap is requested for Standby memory parity errors
PEFDP	8	rwh	**Parity Error Flag for Dual Port Memory** 0_B No Parity errors have been detected for dual port memory 1_B A Parity error is indicated and can trigger a trap request trigger, if enabled for dual port memory The bit is only set by the enabled parity error from the dual port memory. This bit can only be cleared via software. Writing a zero to this bit does not change the content. Writing a one to this bit does clear the bit.
PEFDS	9	rwh	**Parity Error Flag for Data SRAM** 0_B No Parity errors have been detected for data SRAM 1_B A Parity error is indicated and can trigger a trap request trigger, if enabled for data SRAM The bit is only set by the enabled parity error from the data SRAM. This bit can only be cleared via software. Writing a zero to this bit does not change the content. Writing a one to this bit does clear the bit.

Field	Bits	Type	Description
PEFPS	10	rwh	**Parity Error Flag for Program SRAM** 0_B No Parity errors have been detected for program SRAM 1_B A Parity error is indicated and can trigger a trap request trigger, if enabled for program SRAM The bit is only set by the enabled parity error from the program SRAM. This bit can only be cleared via software. Writing a zero to this bit does not change the content. Writing a one to this bit does clear the bit.
PEFU0	11	rwh	**Parity Error Flag for USIC0 Memory** 0_B No Parity errors have been detected for USIC0 memory 1_B A Parity error is indicated and can trigger a trap request trigger, if enabled for USIC0 memory The bit is only set by the enabled parity error from the USIC0 memory. This bit can only be cleared via software. Writing a zero to this bit does not change the content. Writing a one to this bit does clear the bit.
PEFU1	12	rwh	**Parity Error Flag for USIC1 Memory** 0_B No Parity errors have been detected for USIC1 memory 1_B A Parity error is indicated and can trigger a trap request trigger, if enabled for USIC1 memory The bit is only set by the enabled parity error from the USIC1 memory. This bit can only be cleared via software. Writing a zero to this bit does not change the content. Writing a one to this bit does clear the bit.
PEFU2	13	rwh	**Parity Error Flag for USIC2 Memory** 0_B No Parity errors have been detected for USIC2 memory 1_B A Parity error is indicated and can trigger a trap request trigger, if enabled for USIC2 memory The bit is only set by the enabled parity error from the USIC2 memory. This bit can only be cleared via software. Writing a zero to this bit does not change the content. Writing a one to this bit does clear the bit.

Field	Bits	Type	Description
PEFMC	14	rwh	**Parity Error Flag for MultiCAN Memory** 0_B No Parity errors have been detected for MultiCAN memory 1_B A Parity error is indicated and can trigger a trap request trigger, if enabled for MultiCAN memory The bit is only set by the enabled parity error from the MutliCAN memory. This bit can only be cleared via software. Writing a zero to this bit does not change the content. Writing a one to this bit does clear the bit.
PEFSB	15	rwh	**Parity Error Flag for Standby Memory** 0_B No Parity errors have been detected for Standby memory 1_B A Parity error is indicated and can trigger a trap request trigger, if enabled for Standby memory The bit is only set by the enabled parity error from the Standby memory. This bit can only be cleared via software. Writing a zero to this bit does not change the content. Writing a one to this bit does clear the bit.

PMTPR
Parity Memory Test Pattern RegisterESFR ($F0E4_H/72_H$) **Reset Value: 0000_H**



Field	Bits	Type	Description
PRD	[15:8]	rh	**Parity Read Values for Memory Test** For each byte of a memory module the parity bits generated during the most recent read access are indicated here.
PWR	[7:0]	rw	**Parity Write Values for Memory Test** For each byte of a memory module the parity bits corresponding to the next write access are stored here.

PMTSR
Parity Memory Test Select RegisterESFR ($F0E6_H/73_H$) **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							PESEN	MTENSB	0				MTENPS	MTENDS	MTENDP
r							rw	rw	rw				rw	rw	rw

Field	Bits	Type	Description
MTENDP	0	rw	**Memory Test Enable Control for Dual Port Memory** Controls the test multiplexer for the dual port memory. 0_B Standard operation 1_B Test parity bits used (from PMTPR)
MTENDS	1	rw	**Memory Test Enable Control for Data SRAM** Controls the test multiplexer for the data SRAM. 0_B Standard operation 1_B Test parity bits used (from PMTPR)
MTENPS	2	rw	**Memory Test Enable Control for Program SRAM** Controls the test multiplexer for the program SRAM. 0_B Standard operation 1_B Test parity bits used (from PMTPR)
MTENSB	7	rw	**Memory Test Enable Control for Standby Memory** Controls the test multiplexer for the Standby memory. 0_B Standard operation 1_B Test parity bits used (from PMTPR)
PESEN	8	rw	**Parity Error Sensitivity Enable** 0_B Parity errors have no effect 1_B Parity errors are indicated and can trigger a trap, if enabled
0	[6:3]	rw	**Reserved** Must be written with reset value 0.
0	[15:9]	r	**Reserved** Read as 0; should be written with 0.

6.14 Register Control

This block handles the register accesses of the SCU and the register access control for all system register that use one of the following protection modes:

- Unprotected Mode
- Write Protection Mode
- Secured Mode

6.14.1 Register Access Control

There are some dedicated registers that control critical system functions and modes. These registers are protected by a special register security mechanism so these vital system functions cannot be changed inadvertently after the executing of the EINIT instruction. However, as these registers control central system behavior they need to be accessed during operation. The system control software gets this access via a special security state machine.

If an access violation is detected a trap trigger request is generated.

This security mechanism controls the following security levels wich can be configured via register SLC:

- **Unprotected Mode**
  No protection is active. Registers can be written at any time. This mode is entered after the Application Reset.
- **Write Protected Mode**
  Protected registers are locked against any write access. Write accesses have no effect on these registers. This mode is entered automatically after the EINIT instruction is executed.
- **Secured Mode**
  Protected registers can be written using a special command. Registers that are protected by this mode are marked in **Table 6-19** as Sec protected.
  Access in Secured Mode can be achieved by preceding the intended write access with writing “command 4” to register SLC. After writing “command 4” to register SLC the register protection mechanism remains disabled until the next write to a register on the PD+Bus (SFR, ESFR, XSFR area), i.e. accesses to registers (e.g. CSFR) outside this area do not enable the protection again automatically. Therefore, the lock mechanism after writing “command 4” works differently depending on the register address. Normally one single write access to a protected register is enabled. After this write access the protected registers are locked again automatically. Thereafter, “command 4” has to be written again in order to enable the next write to a protected register. The lock mechanism is not enabled again after a write access to a CSFR register or to a LXBus peripheral register (XLOC area, e.g. USIC, CAN, IMB).

Note: In Secured Mode the re-enabling of register protection with respect to the write address after “command 4” can lead to an unexpected, not obvious behaviour of

an application:
In case the succeeding write to a protected register is delayed due to an interrupt and the ISR itself uses the "command 4" mechanism. After writing "command 4" inside the ISR the protection is expectedly re-installed instead of released and the following write will lead to an ACER trap within the ISR. An ATOMIC instruction, which couples the unlock with the write to the protected register could be used.
In case the succeeding write is to a register which does not re-enable the protection mechanism again then the write itself will succeed, but in a following "command 4" sequence the write to SLC register re-locks the protection again and the write to a protected register fails.

All registers that are equipped with this protection mechanism have additional to normal access parameters (e.g. read only, bit type r or rh) the access limitations defined by the selected security level. Independently of the security level all protected registers can also be read.

6.14.1.1 Controlling the Security Level

The two registers Security Level Command register (SLC) and Security Level Status register (SLS) control the security level. The SLC register accepts the commands to control the state machine modifying the security level, while the SLS register shows the actual password, the actual security level, and the state of the state machine.

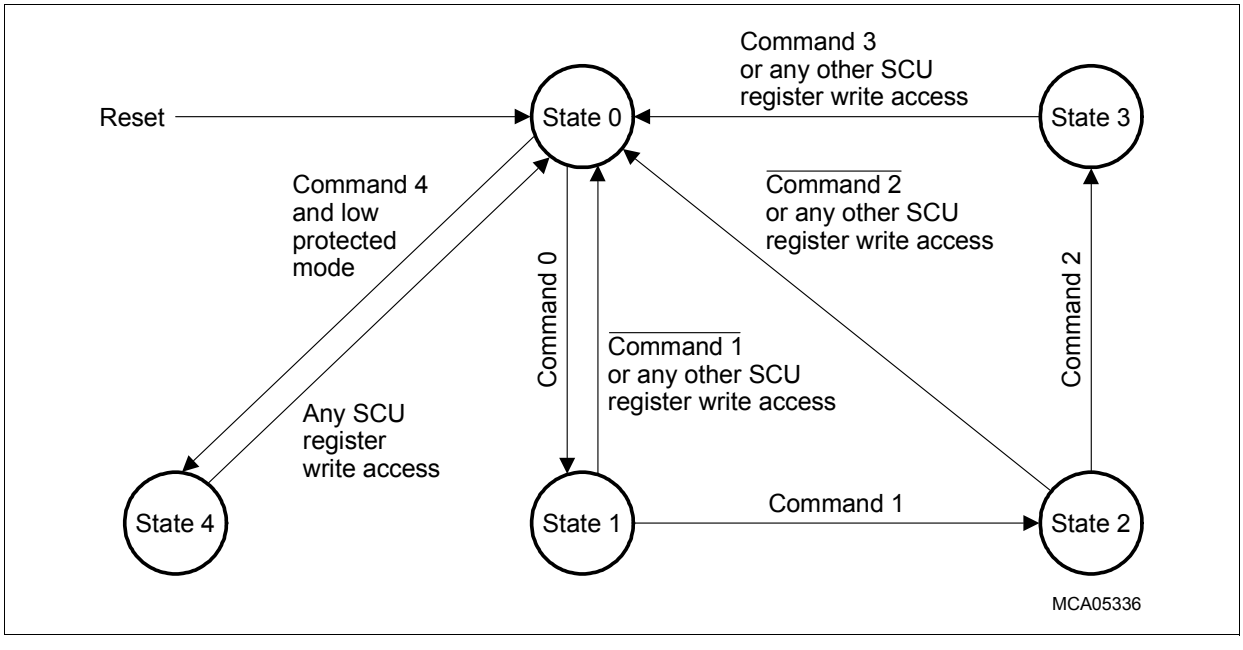


Figure 6-32 State Machine for Security Level Controlling

The following mechanism is used to control the actual security level:

- **Changing the security level**
  can be done by executing the following command sequence:

“command 0 - command 1 - command 2 - command 3”.
This sequence establishes a new security level and/or a new password.

Table 6-17 Commands for Security Level Control

Command	Definition	Note
Command 0	$AAAA_H$	
Command 1	5554_H	
Command 2	96_H +[1] <inverse password>	
Command 3	000_B + <new level> + 000_B + <new password>	
Command 4	$8E_H$ + <inverse password>	Secured Mode only

1) ’+’ denotes a bit field concatenation

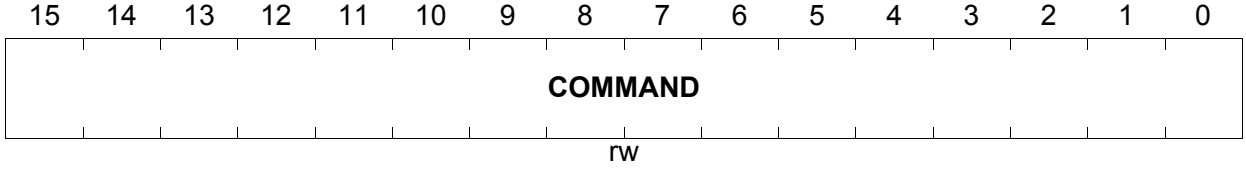
Note: It is recommended to lock all command sequences with an atomic sequence.

6.14.2 Register Protection Registers

Register SLC

This register is the interface for the protection commands.

SLC
Security Level Command RegisterESFR ($F0C0_H/60_H$) **Reset Value: 0000_H**

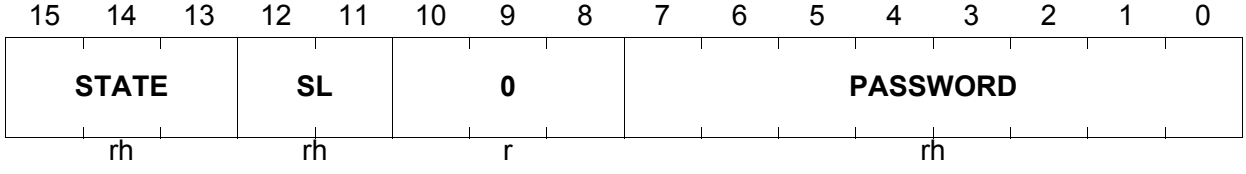


Field	Bits	Type	Description
COMMAND	[15:0]	rw	**Security Level Control Command** The commands to control the security level must be written to this register (see table)

Register SLS

This register monitors the status of the register protection.

SLS
Security Level Status Register ESFR ($F0C2_H$/61_H) Reset Value: 0000_H



Field	Bits	Type	Description
PASSWORD	[7:0]	rh	**Current Security Control Password** Default after reset = 00_H
SL	[12:11]	rh	**Security Level** [1)] 00_B Unprotected Mode (default) 01_B Secured Mode 10_B Reserved, Do not use this combination 11_B Write Protected Mode
STATE	[15:13]	rh	**Current State of Switching State Machine** 000_B Awaiting command 0 or command 4 (default) 001_B Awaiting command 1 010_B Awaiting command 2 011_B Awaiting new security level and password 100_B Next access granted in Secured Mode 101_B Reserved, do not use this combination $11X_B$ Reserved, do not use this combination
0	[10:8]	r	**Reserved** Read as 0; should be written with 0;

1) While the security level is “unprotected” after reset, it changes to “write protected” after the execution of instruction EINIT.

6.15 Miscellaneous System Registers

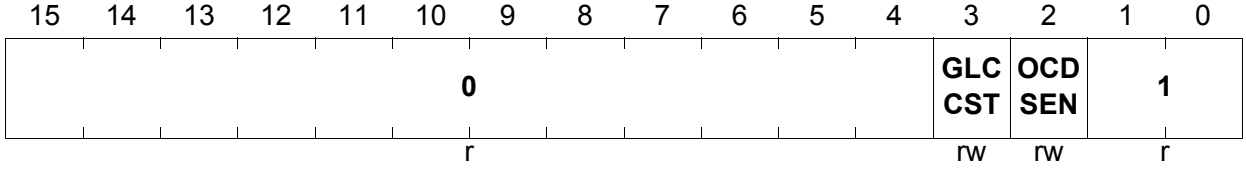
This chapter acts as container for various register that are not connected to one specific application topic.

6.15.1 System Registers

6.15.1.1 System Control Register

The following register serve several different system tasks.

SYSCON1
System Control 1 Register **SFR ($FF4C_H/A6_H$)** **Reset Value: 0003_H**



Field	Bits	Type	Description
1	[1:0]	r	**Reserved** Should be written with reset value 11_B. Will be changed in future versions.
OCDSEN	2	rw	**OCDS/Cerberus Enable** 0_B OCDS and Cerberus are still in reset state 1_B ODCS and Cerberus are operable
GLCCST	3	rw	**Global CAPCOM Start** This bit starts all CAPCOM units synchronously if enabled. 0_B CAPCOM timer start is controlled locally in each unit 1_B All CAPCOM timers are started synchronously This bit needs to be cleared via software before setting starts a new CAPCOM start.
0	[15:4]	r	**Reserved** Read as 0; should be written with 0.

6.15.2 Identification Block

For identification of the most important silicon parameters a set of identification registers is defined that provide information on the chip manufacturer, the chip type and its properties.

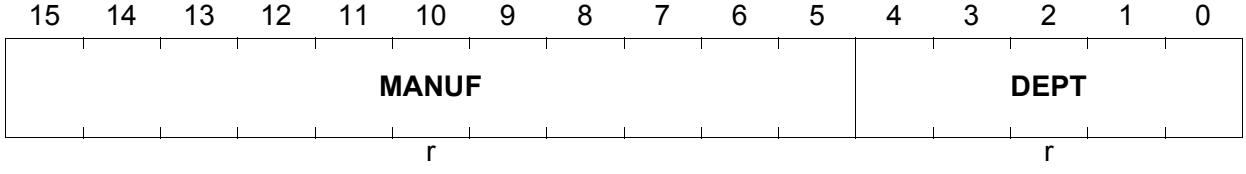
6.15.2.1 Identification Registers

Register IDMANUF

This register contains information about the manufacturer.

IDMANUF
Manufacturer Identification Register

ESFR ($F07E_H/3F_H$) **Reset Value: 1820_H**

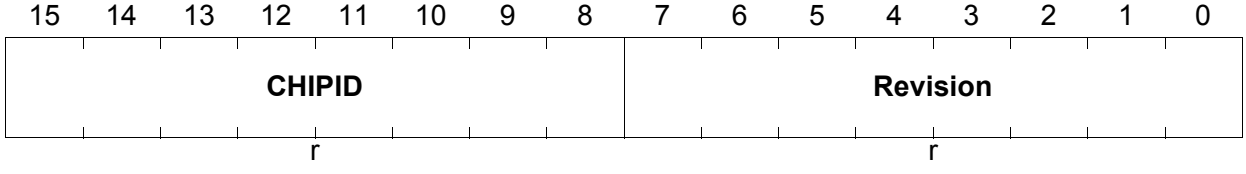


Field	Bits	Type	Description
DEPT	[4:0]	r	**Department** Indicates the department within Infineon. 00_HAIM MC
MANUF	[15:5]	r	**Manufacturer** This is the JEDEC normalized manufacturer code. $0C1_H$Infineon Technologies AG

Register IDCHIP

This register contains information about the device.

IDCHIP
Chip Identification Register **ESFR ($F07C_H/3E_H$)** **Reset Value: $XXXX_H$**

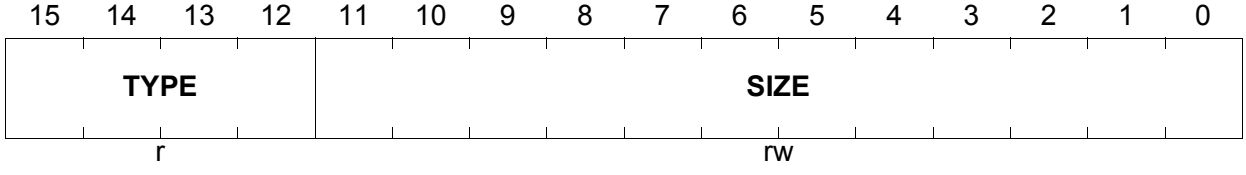


Field	Bits	Type	Description
Revision	[7:0]	r	**Device Revision Code** Identifies the device step. Please refer to the data sheet for the device specific value.
CHIPID	[15:8]	r	**Device Identification** Identifies the device name. Please refer to the data sheet for the device specific value.

Register IDMEM

This register contains information about the program memory.

IDMEM
Program Memory Identification Register
ESFR ($F07A_H/3D_H$) **Reset Value: $3XXX_H$**



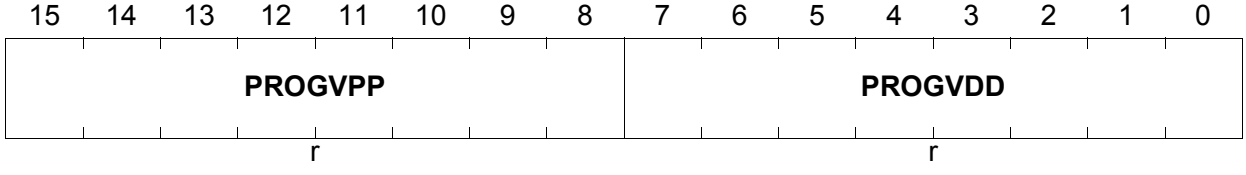
Field	Bits	Type	Description
SIZE	[11:0]	rw	**Size of on-chip Program Memory** The size of the implemented program memory in terms of 4 K blocks, i.e. memory size = <SIZE>*4 Kbyte. Please refer to the data sheet for the device specific value.
TYPE	[15:12]	r	**Type of on-chip Program Memory** Identifies the memory type on this silicon. Please refer to the data sheet for the device specific value.

Register IDPROG

This register contains information about the flash programming voltage.

IDPROG
Programming Voltage Id. Register

ESFR ($F078_H/3C_H$) **Reset Value: 1313_H**



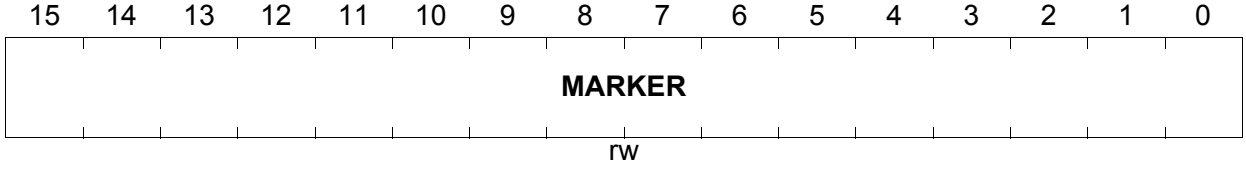
Field	Bits	Type	Description
PROGVDD	[7:0]	r	**Programming VDD Voltage** The voltage of the standard power supply required to program or erase (if applicable) the on-chip program memory. Please refer to the data sheet for the device specific value.
PROGVPP	[15:8]	r	**Programming VPP Voltage** The voltage of the special programming power supply (if existent) required to program or erase (if applicable) the on-chip program memory. Please refer to the data sheet for the device specific value.

6.15.3 Marker Memory

6.15.3.1 Marker Memory Registers

The marker memory consists of following SFRs located in the DMP_M for free usage of the user software.

MKMEM0
Marker Memory 0 Register **SFR ($FED0_H/68_H$)** **Reset Value: 0000_H**
MKMEM1
Marker Memory 1 Register **SFR ($FED2_H/69_H$)** **Reset Value: 0000_H**



Field	Bits	Type	Description
MARKER	[15:0]	rw	**Marker Content**

6.16 SCU Register Addresses

The SCU registers are within the (E)SFR space of the XE16x. Therefore, their specified addresses equal an offset from zero.

Table 6-18 Registers Address Space

Module	Base Address	End Address	Note
SCU	$00\ 0000_H$	$00\ FFFE_H$	

SCU Register Overview

Table 6-19 Register Overview of SCU

Short Name	Register Long Name	Offset Addr.	Protection[1]	Reset [2]
WUOSCCON	Wake-up OSC Control Register	$F1AE_H$	Sec	Power-on Reset
HPOSCCON	High Precision Oscillator Configuration Register	$F1B4_H$	Sec	Power-on Reset
PLLOSCCON	PLL Control Register	$F1B6_H$	Sec	Power-on Reset
PLLSTAT	PLL Status Register	$F0BC_H$	-	Power-on Reset
STATCLR1	PLL Status Clear 1 Register	$F0E2_H$	Sec	Power-on Reset
PLLCON0	PLL Configuration 0 Register	$F1B8_H$	Sec	Power-on Reset
PLLCON1	PLL Configuration 1 Register	$F1BA_H$	Sec	Power-on Reset
PLLCON2	PLL Configuration 2 Register	$F1BC_H$	Sec	Power-on Reset
PLLCON3	PLL Configuration 3 Register	$F1BE_H$	Sec	Power-on Reset
SYSCON0	System Configuration 0 Register	$FF4A_H$	Sec	Power-on Reset
STATCLR0	Status Clear 0 Register	$F0E0_H$	Sec	Power-on Reset
RTCCLKCON	RTC Clock Control Register	$FF4E_H$	Sec	Power-on Reset

Table 6-19 Register Overview of SCU

Short Name	Register Long Name	Offset Addr.	Protection[1)]	Reset [2)]
EXTCON	External Clock Control Register	$FF5E_H$	Sec	Power-on Reset
WICR	Wake-up Interval Count Register	$F0B0_H$	Sec	Power-on Reset
WUCR	Wake-up Control Register	$F1B0_H$	Sec	Power-on Reset
RSTSTAT0	Reset Status 0 Register	$F0B2_H$	-	Power-on Reset
RSTSTAT1	Reset Status 1 Register	$F0B4_H$	-	Power-on Reset
RSTSTAT2	Reset Status 2 Register	$F0B6_H$	-	Power-on Reset
RSTCON0	Reset Configuration 0 Register	$F0B8_H$	Sec	Power-on Reset
RSTCON1	Reset Configuration 1 Register	$F0BA_H$	Sec	Power-on Reset
RSTCNTCON	Reset Counter Configuration Register	$F1B2_H$	Sec	Power-on Reset
SWRSTCON	SW Reset Control Register	$F0AE_H$	Sec	Power-on Reset
ESREXCON1	ESR 1 External Control Register	$FF32_H$	Sec	Power-on Reset
ESREXCON2	ESR 2 External Control Register	$FF34_H$	Sec	Power-on Reset
ESRCFG0	ESR 0 Configuration Register	$F100_H$	Sec	Power-on Reset
ESRCFG1	ESR 1 Configuration Register	$F102_H$	Sec	Power-on Reset
ESRCFG2	ESR 2 Configuration Register	$F104_H$	Sec	Power-on Reset
ESRDAT	ESR Data Register	$F106_H$	-	Power-on Reset
SWDCON0	SWD Control 0 Register	$F080_H$	Sec	Power-on Reset

Table 6-19 Register Overview of SCU

Short Name	Register Long Name	Offset Addr.	Protection[1]	Reset [2]
SWDCON1	SWD Control 1 Register	$F082_H$	Sec	Power-on Reset
PVC1CON0	PVC_1 Control for Step 0 Register	$F014_H$	Sec	Power-on Reset
PVCMCON0	PVC_M Control for Step 0 Register	$F1E4_H$	Sec	Power-on Reset
EVR1CON0	EVR_1 Control 0 Register	$F088_H$	Sec	Power-on Reset
EVR1SET15VHP	EVR_1 Setting for 1.5V HP Register	$F09E_H$	Sec	Power-on Reset
EVRMCON0	EVR_M Control 0 Register	$F084_H$	Sec	Power-on Reset
EVRMCON1	EVR_M Control 1 Register	$F086_H$	Sec	Power-on Reset
EVRMSET15VHP	EVR_M Setting for 1.5V HP Register	$F096_H$	Sec	Power-on Reset
GSCSWREQ	GSC SW Request Register	$FF14_H$	Sec	Application Reset
GSCEN	GSC Enable Register	$FF16_H$	Sec	Application Reset
GSCSTAT	GSC Status Register	$FF18_H$	-	Application Reset
STSTAT	Start-up Status Register	$F1E0_H$	-	Application Reset
EXISEL	External Interrupt Input Select Register	$F1A0_H$	Sec	Application Reset
EXICON0	External Interrupt Input Trigger Control 0 Register	$F030_H$	Sec	Application Reset
EXICON1	External Interrupt Input Trigger Control 1 Register	$F032_H$	Sec	Application Reset
EXICON2	External Interrupt Input Trigger Control 2 Register	$F034_H$	Sec	Application Reset
EXICON3	External Interrupt Input Trigger Control 3 Register	$F036_H$	Sec	Application Reset

Table 6-19 Register Overview of SCU

Short Name	Register Long Name	Offset Addr.	Protection[1]	Reset [2]
EXOCON0	External Output Trigger Control 0 Register	$FE30_H$	Sec	Application Reset
EXOCON1	External Output Trigger Control 1 Register	$FE32_H$	Sec	Application Reset
EXOCON2	External Output Trigger Control 2 Register	$FE34_H$	Sec	Application Reset
EXOCON3	External Output Trigger Control 3 Register	$FE36_H$	Sec	Application Reset
INTSTAT	Interrupt Status Register	$FF00_H$	-	Application Reset
INTCLR	Interrupt Clear Register	$FE82_H$	Sec	Application Reset
INTSET	Interrupt Set Register	$FE80_H$	Sec	Application Reset
INTDIS	Interrupt Disable Register	$FE84_H$	Sec	Application Reset
INTNP0	Interrupt Node Pointer 0 Register	$FE86_H$	Sec	Application Reset
INTNP1	Interrupt Node Pointer 1 Register	$FE88_H$	Sec	Application Reset
DMPMIT	DMP_M Interrupt and Trap Trigger Register	$FE96_H$	-	Power-on Reset
DMPMITCLR	DMP_M Interrupt and Trap Clear Register	$FE98_H$	Sec	Power-on Reset
ISSR	Interrupt Source Select Register	$FF2E_H$	Sec	Application Reset
TCCR	Temperature Compensation Control Register	$F1AC_H$	Sec	Application Reset
TCLR	Temperature Compensation Level Register	$F0AC_H$	Sec	Application Reset
WDTREL	WDT Reload Register	$F0C8_H$	Sec	Application Reset

Table 6-19 Register Overview of SCU

Short Name	Register Long Name	Offset Addr.	Protection[1)]	Reset [2)]
WDTCS	WDT Control and Status Register	$F0C6_H$	Sec	Application Reset
WDTTIM	WDT Timer Register	$F0CA_H$	Sec	Application Reset
TRAPSTAT	Trap Status Register	$FF02_H$	-	Power-on Reset
TRAPCLR	Trap Clear Register	$FE8E_H$	Sec	Power-on Reset
TRAPSET	Trap Set Register	$FE8C_H$	Sec	Power-on Reset
TRAPDIS	Trap Disable Register	$FE90_H$	Sec	Power-on Reset
TRAPNP	Trap Node Pointer Register	$FE92_H$	Sec	Power-on Reset
PECON	Parity Error Control Register	$F0C4_H$	Sec	Application Reset
PMTPR	Parity Memory Test Pattern Register	$F0E4_H$	Sec	Application Reset
PMTSR	Parity Memory Test Select Register	$F0E6_H$	Sec	Application Reset
SLC	Security Level Command Register	$F0C0_H$	-	Application Reset
SLS	Security Level Status Register	$F0C2_H$	-	Application Reset
SYSCON1	System Control 1 Register	$FF4C_H$	Sec	Application Reset
IDMANUF	Manufacturer Identification Register	$F07E_H$	-	Power-on Reset
IDCHIP	Chip Identification Register	$F07C_H$	-	Power-on Reset
IDMEM	Program Memory Identification Register	$F07A_H$	-	Power-on Reset
IDPROG	Programming Voltage Identification Register	$F078_H$	-	Power-on Reset

Table 6-19 Register Overview of SCU

Short Name	Register Long Name	Offset Addr.	Protection[1)]	Reset [2)]
MKMEM0	Marker Memory 0 Register	$FED0_H$	Sec	Power-on Reset
MKMEM1	Marker Memory 1 Register	$FED2_H$	Sec	Power-on Reset

1) Register write protection mechanism: “Sec” = register security mechanism, “-” = always accessible (no protection), otherwise no access is possible.

2) Reset types are defined in Chapter 6.3.1.2.

6.17 Implementation

This section shows the connections of the module to the system.

6.17.1 Clock Generation Unit

The following table shows the input connection of the Clock Genation Unit.

Table 6-20 CGU Input Connection

Input	Connected to
XTAL 1	XTAL 1
XTAL 2	XTAL 2
CLKIN1	Port 2.9
CLKIN2	Port 4.4

7 Parallel Ports

The XE16x provides a set of General Purpose Input/Output (GPIO) ports that can be controlled by software and by the on-chip peripheral units:

Table 7-1 Ports of the XE16x

Group	Width	I/O	Connected Modules
P0	8	I/O	EBC (A7...A0), CCU6, USIC, CAN
P1	8	I/O	EBC (A15...A8), CCU6
P2	13	I/O	EBC (READY, $\overline{\text{BHE}}$, A23...A16, AD15...AD13, D15...D13), CAN, CCU2, GPT12E, USIC, JTAG
P3	8	I/O	EBC arbitration ($\overline{\text{BREQ}}$, $\overline{\text{HLDA}}$, $\overline{\text{HOLD}}$), CAN, USIC
P4	8	I/O	EBC ($\overline{\text{CS4}}$...$\overline{\text{CS0}}$), CCU2, CAN, GPT12E
P5	16	I	Analog Inputs, CCU6, JTAG, GPT12E, CAN
P6	4	I/O	ADC, GPT12E
P7	5	I/O	P7.0 J-LINK, CAN, GPT12E, SCU, JTAG, CCU6, ADC
P8	7	I/O	CCU6, JTAG
P9	8	I/O	CCU6, JTAG, CAN
P10	16	I/O	EBC(ALE, $\overline{\text{RD}}$, $\overline{\text{WR}}$, AD12...AD0, D12...D0), CCU6, USIC, JTAG, CAN
P11	6	I/O	CCU6
P15	8	I	Analog Inputs, GPT12E, CCU6

Note: The availability of ports and port pins depends on the selected device type. This chapter describes the maximum set of ports.

7.1 General Description

This chapter describes the architecture of the digital control circuit for a single port pin.

7.1.1 Basic Port Operation

There are three types of digital control circuits: with/without hardware override for digital GPIOs, and for one for analog inputs. Each port pin contains one of them.

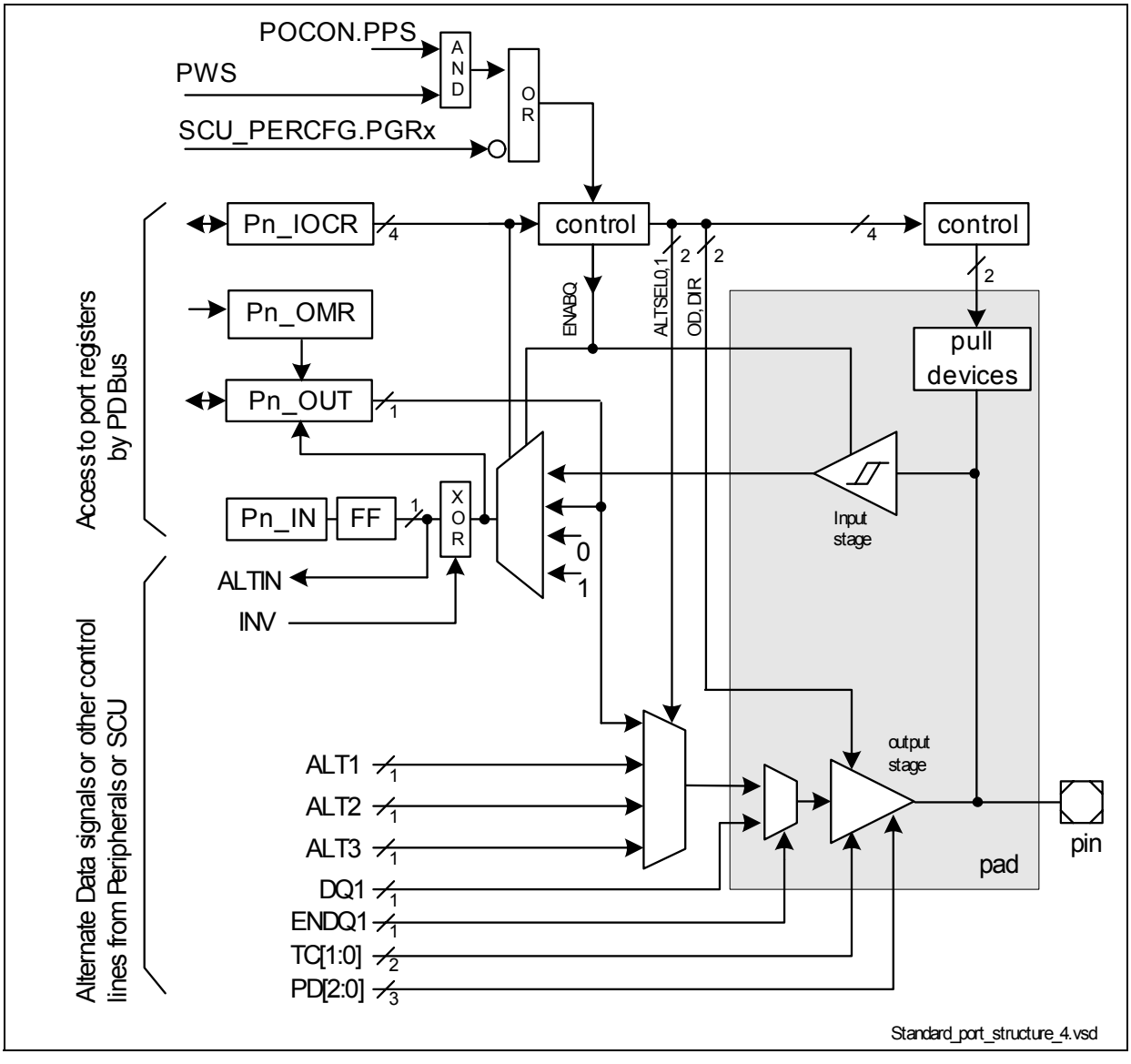


Figure 7-1 Structure of the Ports without Hardware Override Functionality

Note: INV signal is derived from Pn_IOCR.PC[3:2].

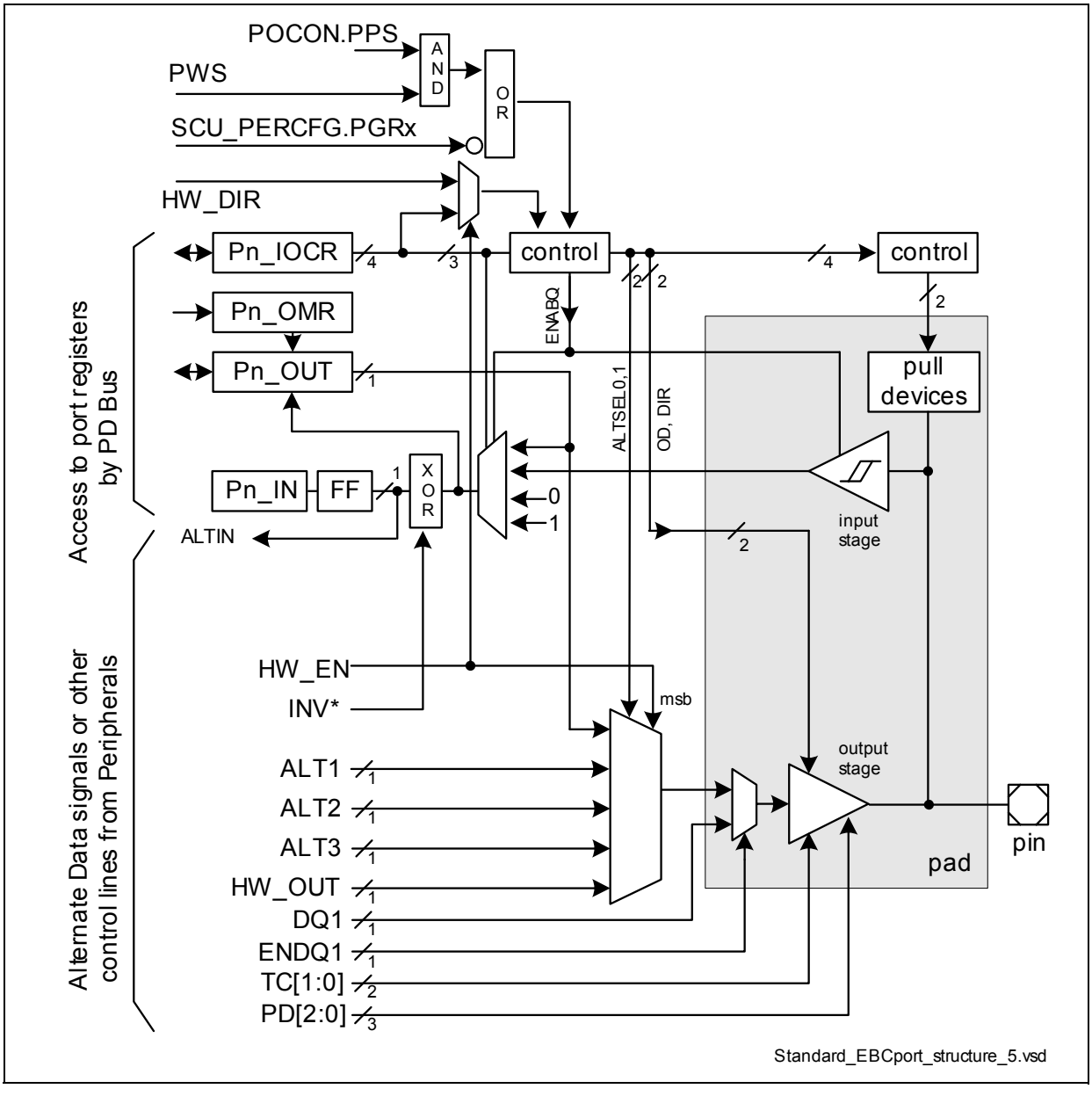


Figure 7-2 Structure of the Ports with Hardware Override Functionality

Note: If HW_EN is activated, INV signal is always zero.*

Note: When HW_EN is disabled, the respective ports go to Power Save Mode as all other ports. When HW_EN is active, then the user should set the POCON.PPSx=0.

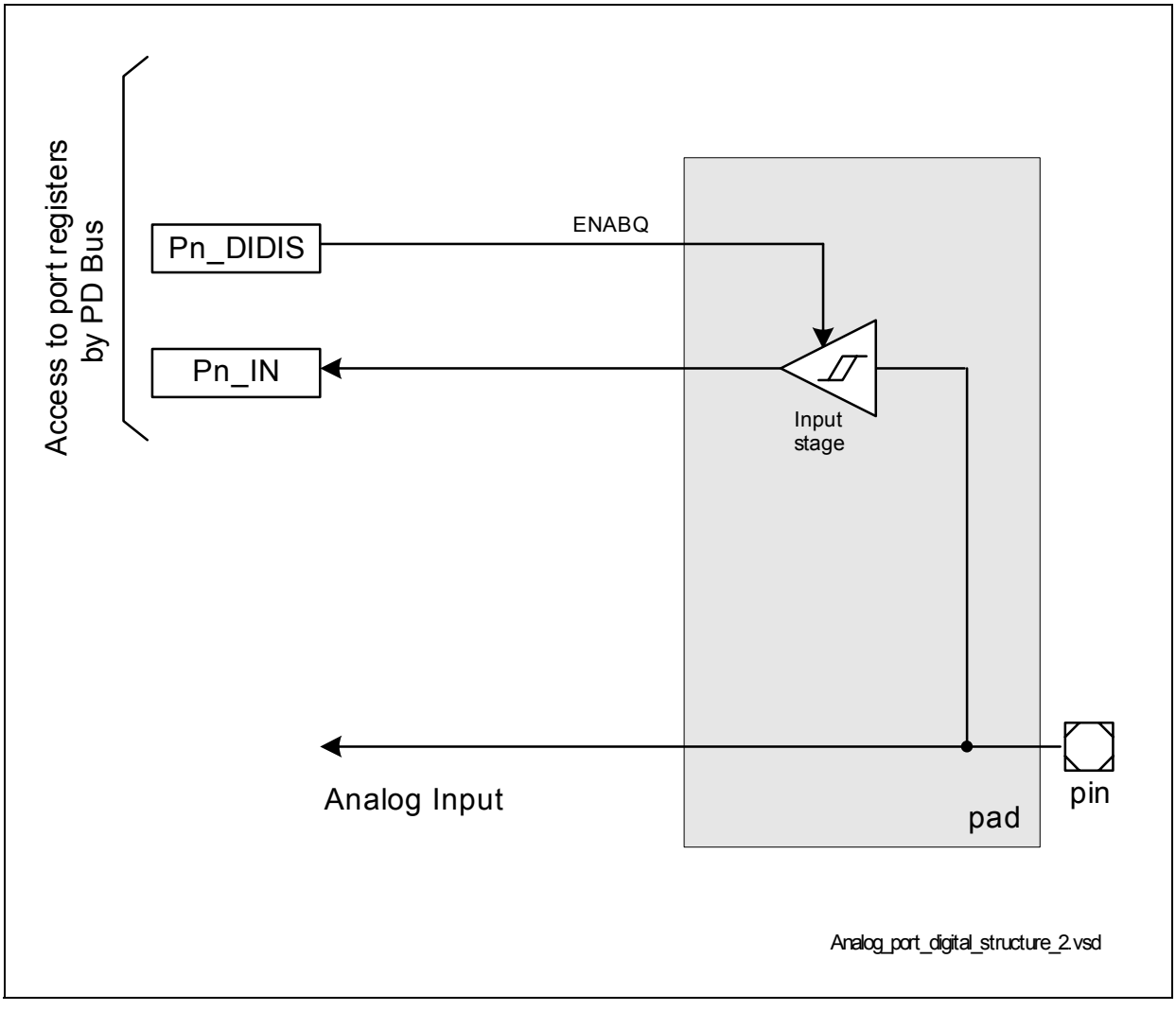


Figure 7-3 Structure of Port 5 and Port 15

Note: There is always a standard digital input connected in parallel to each analog input.

7.1.2 Input Stage Control

An input stage consists of a Schmitt trigger, which can be enabled or disabled via software, and an input multiplexer that by default selects the output of the input Schmitt trigger.

A disabled input driver drives high logical level. During and after reset, all input stages are enabled by default.

7.1.3 Output Driver Control

An output stage consists of an output driver, output multiplexer, and register bit fields for their control.

7.1.3.1 Active Mode Behavior

Each output driver can be configured in a push-pull or an open-drain mode, or it can be deactivated (three-stated). An output multiplexer in front of the output driver selects the signal source, choosing either the appropriate bit of the Pn_OUT register, or one of maximum three lines coming from a peripheral unit, see **Figure 7-1**. The selection is done via the Pn_IOCR register. Software can set or clear the bit Pn_OUT.Px, which drives the port pin in case it is selected by the output multiplexer.

An output driver with hardware override can select an additional output signal coming from a peripheral. While the hardware override is activated, this signal has higher priority than all other output signals and can not be deselected by the port. In this case, the peripheral controls the direction of the pin.

7.1.3.2 Power Saving Mode Behavior

In Power Saving Mode (core and IO supply voltages available), the behavior of a pin depends on the setting of the POCONx.PPSx bit. Basically, groups of four pins within a port can be configured to react to Power Save Mode Request or to ignore it. In case a pin group is configured to react to a Power Save Mode Request, each pin within a group reacts according to its own configuration according to the **Table 7-4**.

7.1.3.3 Reset Behavior

During reset, all output stages of GPIO pins go to tri-state mode without any pull-up or pull-down device.

7.1.3.4 Power-fail Behavior

When the core supply fails while the pad supply remains stable, the output stages go into tri-state mode.

7.2 Port Register Description

7.2.1 Pad Driver Control

The pad structure used in this device offers the possibility to select the output driver strength and the slew rate. These selections are independent from the output port functionality, such as open-drain, push/pull or input only.

In order to minimize EMI problems, the driver strength can be adapted to the application requirements by bit fields PDMx. The selection is done in groups of four pins.

The **Port Output Control registers** POCON provide the corresponding control bits. A 4-bit control field configures the driver strength and the edge shape. Word ports consume four control nibbles each, byte ports consume two control nibbles each, where each control nibble controls 4 pins of the respective port.

Note: P2_POCON register in the XE16x *contains an exception regarding the additional strong output driver connected in parallel to the standard output driver of the P2.8 pin. See port 2 section.*

Px_POCON (x=0-4)
Port x Output Control Register XSFR ($E8A0_H$+2*x) **Reset Value: 0000_H**
Px_POCON (x=6-11)
Port x Output Control Register XSFR ($E8A0_H$+2*x) **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PPS 3		**PDM3**		**PPS 2**		**PDM2**		**PPS 1**		**PDM1**		**PPS 0**		**PDM0**	
rw		**rwr**		**rw**		**rw**		**rw**		**rw**		**rw**		**rw**	

Field	Bit	Type	Description
PDM0, **PDM1,** **PDM2,** **PDM3**	[2:0], [6:4], [10:8], [14:12]	rw	**Port Driver Mode x** Code Driver strength [1] — Edge Shape[2] 000 Strong driver — Sharp edge mode 001 Strong driver — Medium edge mode 010 Strong driver — Soft edge mode 011 Weak driver 100 Medium driver 101 Medium driver 110 Medium driver 111 Weak driver
PPS0, **PPS1,** **PPS2,** **PPS3**	3, 7, 11, 15	rw	**Pin Power Save** 0 Pin behaves like in the Active Mode. Power Save Management is ignored. 1 Behavior in the Power Save Mode described in the **Table 7-4**.

1) Defines the current the respective driver can deliver to the external circuitry.

2) Defines the switching characteristics to the respective new output level. This also influences the peak currents through the driver when producing an edge, i.e. when changing the output level.

Mapping of the POCON Registers to Pins and Ports

The table below lists the defined POCON registers and the allocation of control bit fields and port pins.

Table 7-2 Port Output Control Register Allocation

Control Register	Controlled Pins (by Px_POCON.[y:z])[1]				Port Width
	[15:12]	[11:8]	[7:4]	[3:0]	
P0_POCON	---	---	P0.[7:4]	P0.[3:0]	8
P1_POCON	---	---	P1.[7:4]	P1.[3:0]	8
P2_POCON	CLOCKOUT driver[2] at P2.8	P2.[11:8] + P2.12	P2.[7:4]	P2.[3:0]	13
P3_POCON	---	---	P3.[7:4]	P3.[3:0]	8
P4_POCON	---	---	P4.[7:4]	P4.[3:0]	8
P6_POCON	---	---	---	P6.[3:0]	4
P7_POCON	---	---	P7.4	P7.[3:0]	5
P8_POCON	---	---	P8.[6:4]	P8.[3:0]	7
P9_POCON	---	---	P9.[7:4]	P9.[3:0]	8
P10_POCON	P10.[15:12]	P10.[11:8]	P10.[7:4]	P10.[3:0]	16
P11_POCON	---	---	P11.[5:4]	P11.[3:0]	6

1) x denotes the port number, while [y:z] represents the bit field range.

2) The high-speed clock driver at P2.8 is enabled instead of the standard driver, while P2_POCON.PDM3 = $xx1_B$. The standard driver for P2.8 is added to the next lower pin group and is controlled via P2_POCON.PDM2. Bit P2_POCON.PPS3 has no function. See also **Section 7.3.3**.

Note: When assigning functional signals to port pins, please consider the fact that the driver strength is selected for pin groups. Assign functions with similar requirements to pins within the same POCON control group.

7.2.2 Port Output Register

The port output register defines the values of the output pins if the pin is used as GPIO output.

Pn_OUT (n=0-4)
Port n Output Register **SFR ($FFA2_H$+2*n)** **Reset Value: 0000_H**
Pn_OUT (n=6-11)
Port n Output Register **SFR ($FFA2_H$+2*n)** **Reset Value: 0000_H**

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 Output Bit x** This bit defines the level at the output pin of port Pn, pin x if the output is selected as GPIO output. 0 The output level of Pn.x is 0. 1 The output level of Pn.x is 1.

7.2.3 Port Output Modification Register

The port output modification register contains the bits to individually set, clear, or toggle the value of the port n output register.

P2_OMRH
Port 2 Output Modification Register HighXSFR ($E9CA_H$) **Reset Value: $XXXX_H$**
P10_OMRH
Port 10 Output Modification Register HighXSFR ($E9EA_H$) **Reset Value: $XXXX_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PC 15	**PC 14**	**PC 13**	**PC 12**	**PC 11**	**PC 10**	**PC 9**	**PC 8**	**PS 15**	**PS 14**	**PS 13**	**PS 12**	**PS 11**	**PS 10**	**PS 9**	**PS 8**
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Field	Bits	Type	Description
PSx (x = 8-15)	x-8	w	**Port Set Bit x** Setting this bit sets or toggles the corresponding bit in the port output register Pn_OUT (see **Table 7-3**). On a read access, this bit returns X.
PCx (x = 8-15)	x	w	**Port Clear Bit x** Setting this bit clears or toggles the corresponding bit in the port output register Pn_OUT. (see **Table 7-3**). On a read access, this bit returns X.

Pn_OMRL (n=0-4)
Port n Output Modification Register LowXSFR ($E9C0_H$+4*n) **Reset Value: $XXXX_H$**
Pn_OMRL (n=6-11)
Port n Output Modification Register LowXSFR ($E9C0_H$+4*n) **Reset Value: $XXXX_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PC 7	**PC 6**	**PC 5**	**PC 4**	**PC 3**	**PC 2**	**PC 1**	**PC 0**	**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 = 0-7)	x	w	**Port Set Bit x** Setting this bit sets or toggles the corresponding bit in the port output register Pn_OUT (see **Table 7-3**). On a read access, this bit returns X.
PCx (x = 0-7)	x + 8	w	**Port Clear Bit x** Setting this bit clears or toggles the corresponding bit in the port output register Pn_OUT. (see **Table 7-3**). On a read access, this bit returns X.

Function of the PCx and PSx bit fields

Table 7-3 Function of the Bits PCx and PSx

PCx	PSx	Function
0 or no write access	0 or no write access	Bit Pn_OUT.Px is not changed.
0 or no write access	1	Bit Pn_OUT.Px is set.
1	0 or no write access	Bit Pn_OUT.Px is cleared.
1	1	Bit Pn_OUT.Px is toggled.

Note: If a bit position is not written (one out of two bytes not targeted by a byte write), the corresponding value is considered as 0. Toggling a bit requires one 16-bit write.

7.2.4 Port Input Register

The port input register contains the values currently read at the input pins, also if a port line is assigned as output.

Pn_IN (n=0-11)
Port n Input Register **SFR ($FF80_H$+2*n)** **Reset Value: 0000_H[1)]**
P15_IN
Port 15 Input Register **SFR ($FF9E_H$)** **Reset Value: 0000_H[1)]**

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

1) Px bits for non implemented I/O lines are always read as 0.

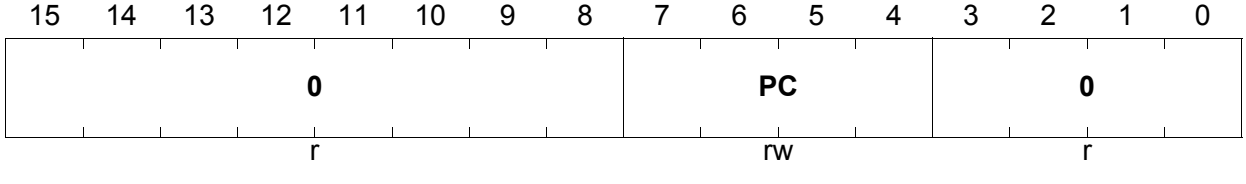
Field	Bits	Type	Description
Px **(x = 0-15)**	x	rh	**Port Input Bit x** This bit indicates the level at the input pin of port Pn, pin x. 0 The input level of Pn.x is 0. 1 The input level of Pn.x is 1.

7.2.5 Port Input/Output Control Registers

The port input/output control registers contain the bit fields to 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 coding of the options is shown in **Table 7-4**.

Depending on the port functionality not all of the input/output control registers may be implemented. The structure with one control bit field for each port pin located in different register offers the possibility to configure port pin functionality of a single pin without accessing some other PCx in the same register by word-oriented writes.

P0_IOCRx (x=00-07)
Port 0 Input/Output Control Register x XSFR ($E800_H$+2*x) **Reset Value: 0000_H**
P1_IOCRx (x=00-07)
Port 1 Input/Output Control Register x XSFR ($E820_H$+2*x) **Reset Value: 0000_H**
P2_IOCRx (x=00-12)
Port 2 Input/Output Control Register x XSFR ($E840_H$+2*x) **Reset Value: 0000_H**
P3_IOCRx (x=00-07)
Port 3 Input/Output Control Register x XSFR ($E860_H$+2*x) **Reset Value: 0000_H**
P4_IOCRx (x=00-07)
Port 4 Input/Output Control Register x XSFR ($E880_H$+2*x) **Reset Value: 0000_H**
P6_IOCRx (x=00-03)
Port 6 Input/Output Control Register x XSFR ($E8C0_H$+2*x) **Reset Value: 0000_H**
P7_IOCRx (x=00-04)
Port 7 Input/Output Control Register x XSFR ($E8E0_H$+2*x) **Reset Value: 0000_H**
P8_IOCRx (x=00-06)
Port 8 Input/Output Control Register x XSFR ($E900_H$+2*x) **Reset Value: 0000_H**
P9_IOCRx (x=00-07)
Port 9 Input/Output Control Register x XSFR ($E920_H$+2*x) **Reset Value: 0000_H**
P10_IOCRx (x=00-15)
Port 10 Input/Output Control Register x XSFR ($E940_H$+2*x) **Reset Value: 0000_H**
P11_IOCRx (x=00-05)
Port 11 Input/Output Control Register x XSFR ($E960_H$+2*x) **Reset Value: 0000_H**



Field	Bits	Type	Description
PC	[7:4]	rw	**Port Input/Output Control Bit** see **Table 7-4**
0	[3:0], [15:8]	r	reserved

Coding of the PC bit field

The coding of the GPIO port behavior is done by the bit fields in the port control registers Pn_IOCRx. There's a control bit field PC for each port pin. The bit fields PC are located in separate control registers in order to allow modifying a port pin (without influencing the others) with simple move operations.

Note: When the pin direction is switched to output and the mode is test mode, the output characteristic must be push-pull only.

Table 7-4 PC Coding

PC[3:0]	I/O	Selected Pull-up/down / Selected Output Function	Behavior in Power Saving Mode[1)]
0000_B	Direct Input	No pull device connected	Input value = Pn_OUT.x; no pull
0001_B		Pull-down device connected	Input value = 0; pull-down
0010_B		Pull-up device connected	Input value = 1; pull-up
0011_B		No pull device connected. Bit Pn_OUT.x reflects the current input value	Input value = Pn_OUT.x; no pull Pn_OUT.x contains the input value before entering in power save mode = freeze of input value
0100_B	Inverted Input	No pull device connected	Input value = $\overline{\text{Pn\_OUT}}$; no pull
0101_B		Pull-down device connected	Input value = 1; pull-down
0110_B		Pull-up device connected	Input value = 0; pull-up
0111_B		No pull device connected Bit Pn_OUT.x reflects the current input value	Input value = $\overline{\text{Pn\_OUT}}$; no pull Pn_OUT.x contains the input value before entering in power save mode = freeze of input value[2)]

Table 7-4 PC Coding

PC[3:0]	I/O	Selected Pull-up/down / Selected Output Function	Behavior in Power Saving Mode[1)]
1000_B	Output (Direct input) Push-pull	General purpose Output	Output driver off. Input Schmitt trigger off. Pn_OUT delivered to the internal logic; no pull
1001_B		Output function ALT1	
1010_B		Output function ALT2	
1011_B		Output function ALT3	
1100_B	Output (Direct input) Open-drain	General purpose Output	
1101_B		Output function ALT1	
1110_B		Output function ALT2	
1111_B		Output function ALT3	

1) In power saving mode, the input Schmitt trigger is always switched off. A defined input value is driven to the internal circuitry instead of the level detected at the input pin.

2) If the IOCR setting is “inverted input”, then an inverted signal Pn_OUT is driven internally. The Pn_OUT register itself always contains the real, non-inverted input value of the pin. See **Figure 7-1** and **Figure 7-2**.

7.2.6 Port Digital Input Disable Register

Ports 5 and 15 have, additionally to the analog input functionality, digital input functionality too. In order to save switching of the internal Schmitt triggers of the digital inputs, they can be disabled by means of Px_DIDIS Register. P5_DIDIS is a 16-bit register, and P15_DIDIS is an 8-bit register.

P5_DIDIS
Port 5 Digital Input Disable RegisterSFR ($FE8A_H$) **Reset Value: 0000_H**
P15_DIDIS
Port 15 Digital Input Disable RegisterSFR ($FE9E_H$) **Reset Value: 0000_H**

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**
rw	**rw**	**rw**	**rw**	**rw**	**rw**	**rw**	**rw**	**rw**	**rw**	**rw**	**rw**	**rw**	**rw**	**rw**	**rw**

Field	Bit	Type	Description
Py **(y = 0-15)**	y	rw	Port 5 Bit y Digital Input Control 0 Digital input stage (schmitt trigger) is enabled. 1 Digital input stage (schmitt trigger) is disabled, necessary if pin is used as analog input.

7.3 Port Description

The bit positions in the port registers always start right-aligned. For example, a port comprising only 8 pins only uses the bit positions [7:0] of the corresponding register. The remaining bit positions are filled with 0 (r).
The pad driver mode registers may be different for each port. As a result, they are described independently for each port in the corresponding chapter.

7.3.1 Port 0

Port 0 is an 8-bit GPIO port. The registers of Port 0 are shown in **Figure 7-4**.
For this port, all pins can be read as GPIO, from the Port Input Register.

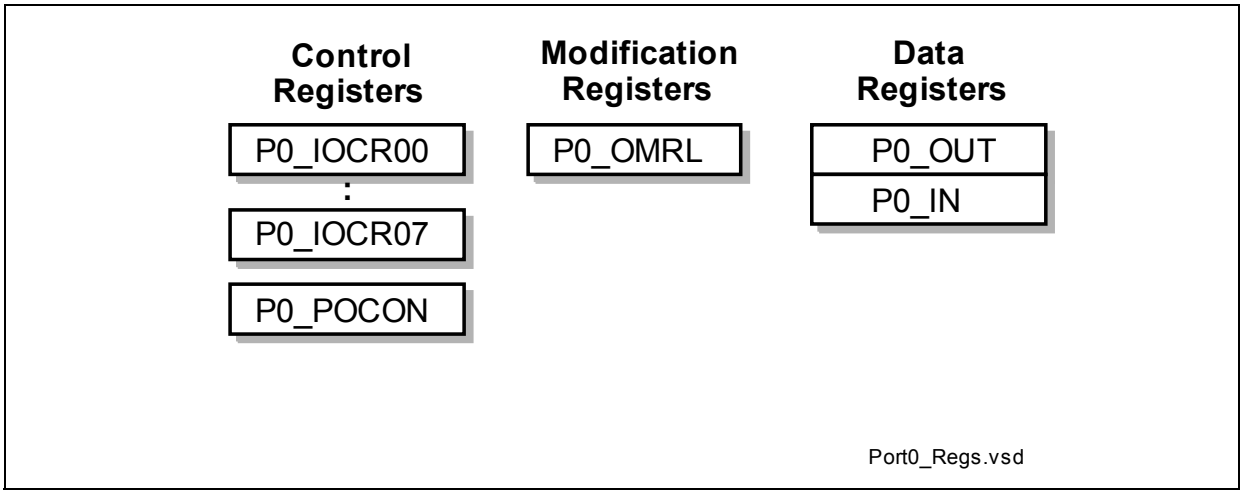


Figure 7-4 Port 0 Register Overview

Table 7-5 Port 0 Registers

Register Short Name	Register Long Name	Address Offset	Reset Value
P0_OUT	Port 0 Output Register	$FFA2_H$	0000_H
P0_IN	Port 0 Input Register	$FF80_H$	0000_H
P0_OMRL	Port 0 Output Modification Register Low	$E9C0_H$	$XXXX_H$
P0_POCON	Port 0 Output Control Register	$E8A0_H$	0000_H
P0_IOCR00	Port 0 Input/Output Control Register 0	$E800_H$	0000_H
P0_IOCR01	Port 0 Input/Output Control Register 1	$E802_H$	0000_H
P0_IOCR02	Port 0 Input/Output Control Register 2	$E804_H$	0000_H
P0_IOCR03	Port 0 Input/Output Control Register 3	$E806_H$	0000_H
P0_IOCR04	Port 0 Input/Output Control Register 4	$E808_H$	0000_H
P0_IOCR05	Port 0 Input/Output Control Register 5	$E80A_H$	0000_H
P0_IOCR06	Port 0 Input/Output Control Register 6	$E80C_H$	0000_H
P0_IOCR07	Port 0 Input/Output Control Register 7	$E80E_H$	0000_H

7.3.2 Port 1

Port 1 is an 8-bit GPIO port. Theregisters of Port 1 are shown in Figure 7-5.

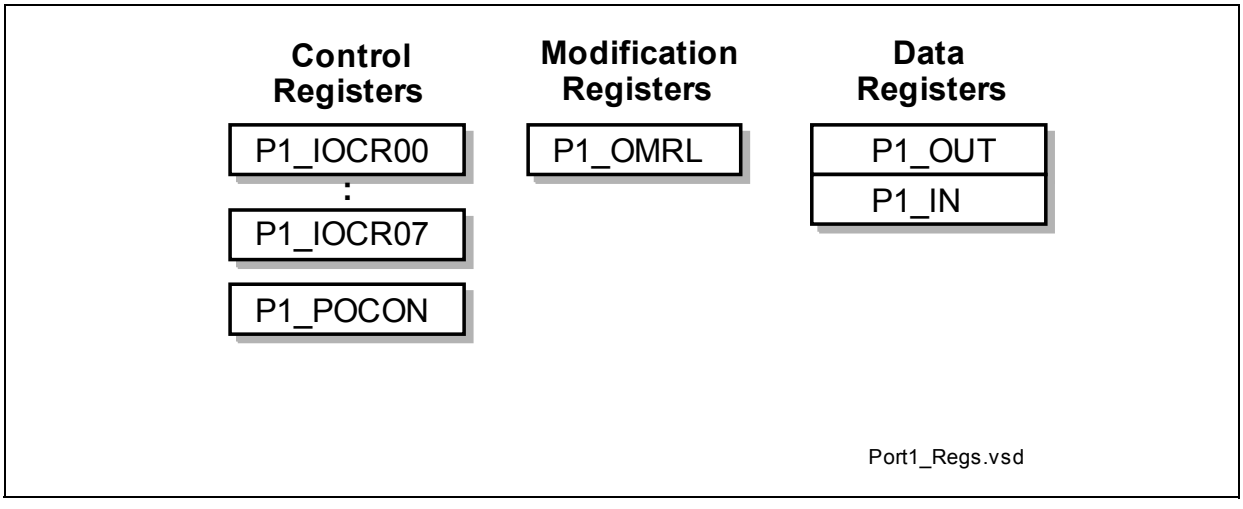


Figure 7-5 Port 1 Register Overview

For this port, all pins can be read as GPIO, from the Port Input Register.

Table 7-6 Port 1 Registers

Register Short Name	Register Long Name	Address Offset	Reset Value
P1_OUT	Port 1 Output Register	$FFA4_H$	0000_H
P1_IN	Port 1 Input Register	$FF82_H$	0000_H
P1_OMRL	Port 1 Output Modification Register Low	$E9C4_H$	$XXXX_H$
P1_POCON	Port 1 Output Control Register	$E8A2_H$	0000_H
P1_IOCR00	Port 1 Input/Output Control Register 0	$E820_H$	0000_H
P1_IOCR01	Port 1 Input/Output Control Register 1	$E822_H$	0000_H
P1_IOCR02	Port 1 Input/Output Control Register 2	$E824_H$	0000_H
P1_IOCR03	Port 1 Input/Output Control Register 3	$E826_H$	0000_H
P1_IOCR04	Port 1 Input/Output Control Register 4	$E828_H$	0000_H
P1_IOCR05	Port 1 Input/Output Control Register 5	$E82A_H$	0000_H
P1_IOCR06	Port 1 Input/Output Control Register 6	$E82C_H$	0000_H
P1_IOCR07	Port 1 Input/Output Control Register 7	$E82E_H$	0000_H

7.3.3 Port 2

Port 2 is an 13-bit GPIO port. The registers of Port 2 are shown in Figure 7-6.
For this port, all pins can be read as GPIO, from the Port Input Register.

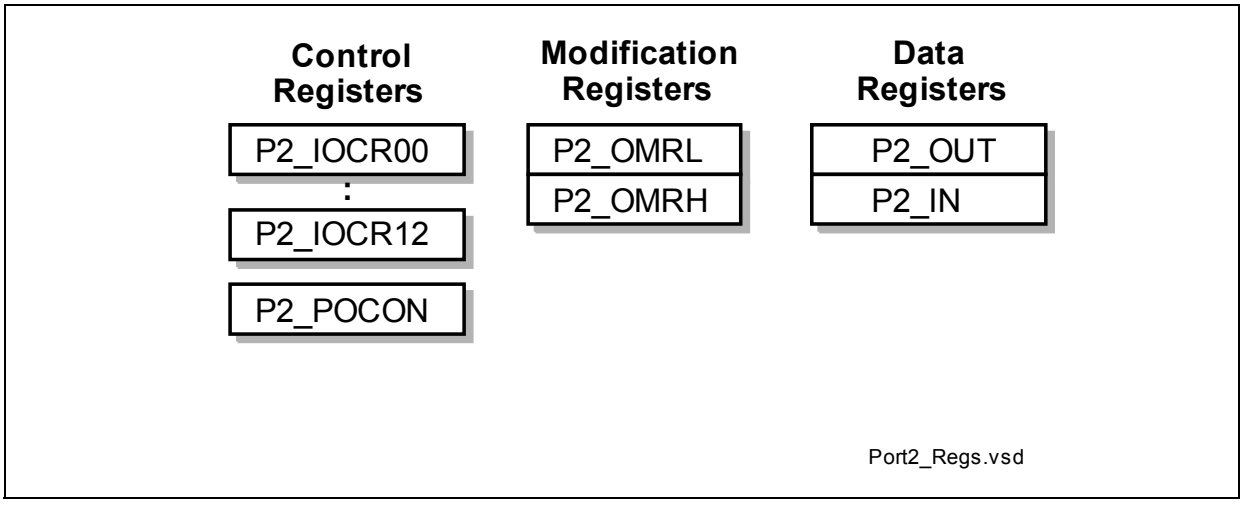


Figure 7-6 Port 2 Register Overview

Table 7-7 Port 2 Registers

Register Short Name	Register Long Name	Address Offset	Reset Value
P2_OUT	Port 2 Output Register	$FFA6_H$	0000_H
P2_IN	Port 2 Input Register	$FF84_H$	0000_H
P2_OMRL	Port 2 Output Modification Register Low	$E9C8_H$	$XXXX_H$
P2_OMRH	Port 2 Output Modification Register High	$E9CA_H$	$XXXX_H$
P2_POCON	Port 2 Output Control Register	$E8A4_H$	0000_H
P2_IOCR00	Port 2 Input/Output Control Register 0	$E840_H$	0000_H
P2_IOCR01	Port 2 Input/Output Control Register 1	$E842_H$	0000_H
P2_IOCR02	Port 2 Input/Output Control Register 2	$E844_H$	0000_H
P2_IOCR03	Port 2 Input/Output Control Register 3	$E846_H$	0000_H
P2_IOCR04	Port 2 Input/Output Control Register 4	$E848_H$	0000_H
P2_IOCR05	Port 2 Input/Output Control Register 5	$E84A_H$	0000_H
P2_IOCR06	Port 2 Input/Output Control Register 6	$E84C_H$	0000_H
P2_IOCR07	Port 2 Input/Output Control Register 7	$E84E_H$	0000_H
P2_IOCR08	Port 2 Input/Output Control Register 8	$E850_H$	0000_H
P2_IOCR09	Port 2 Input/Output Control Register 9	$E852_H$	0000_H

Table 7-7 Port 2 Registers (cont'd)

Register Short Name	Register Long Name	Address Offset	Reset Value
P2_IOCR10	Port 2 Input/Output Control Register 10	$E854_H$	0000_H
P2_IOCR11	Port 2 Input/Output Control Register 11	$E856_H$	0000_H
P2_IOCR12	Port 2 Input/Output Control Register 12	$E858_H$	0000_H

The CLKOUT Pad P2.8

In order to drive high frequency clock signals, a strong driver is connected parallel to the normal output driver of pad P2.8. This clock driver only operates in strong driver sharp edge mode. It is enabled instead of the standard driver, while bitfield P2_POCON.PDM3 = $xx1_B$. Bit P2_POCON.PPS3 has no function.

The standard driver of pin P2.8 is controlled by bits P2_POCON.PDM2 and PPS2, along with pins P2.7 … P2.4. The standard driver is the default selection.

Register P2_IOCR08 controls the driver that is currently active.

7.3.4 Port 3

Port 3 is an 8-bit GPIO port. The registers of Port 3 are shown in Figure 7-7.
For this port, all pins can be read as GPIO, from the Port Input Register.

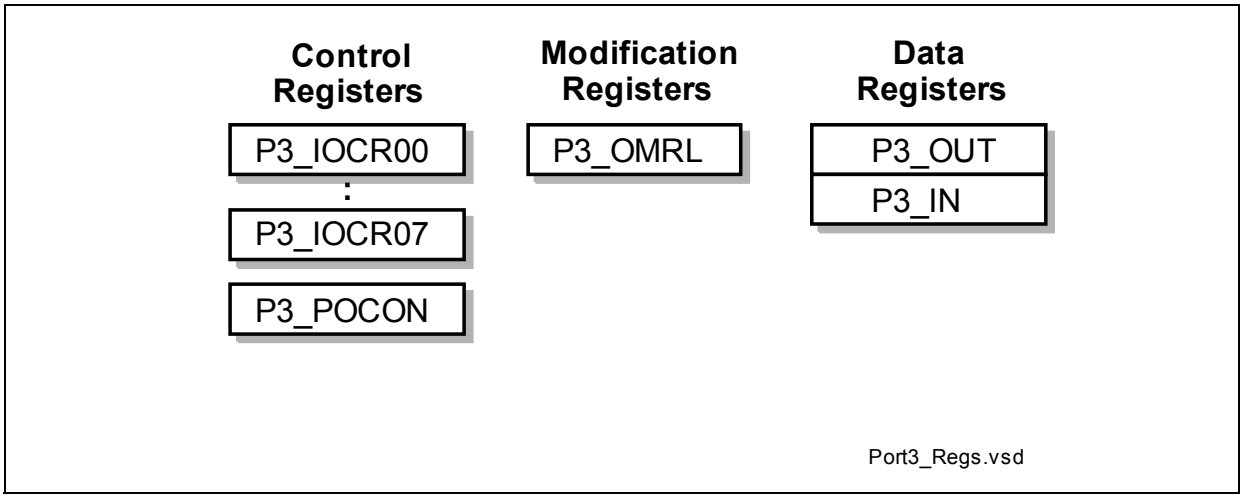


Figure 7-7 Port 3 Register Overview

Table 7-8 Port 3 Registers

Register Short Name	Register Long Name	Address Offset	Reset Value
P3_OUT	Port 3 Output Register	$FFA8_H$	0000_H
P3_IN	Port 3 Input Register	$FF86_H$	0000_H
P3_OMRL	Port 3 Output Modification Register Low	$E9CC_H$	$XXXX_H$
P3_POCON	Port 3 Output Control Register	$E8A6_H$	0000_H
P3_IOCR00	Port 3 Input/Output Control Register 0	$E860_H$	0000_H
P3_IOCR01	Port 3 Input/Output Control Register 1	$E862_H$	0000_H
P3_IOCR02	Port 3 Input/Output Control Register 2	$E864_H$	0000_H
P3_IOCR03	Port 3 Input/Output Control Register 3	$E866_H$	0000_H
P3_IOCR04	Port 3 Input/Output Control Register 4	$E868_H$	0000_H
P3_IOCR05	Port 3 Input/Output Control Register 5	$E86A_H$	0000_H
P3_IOCR06	Port 3 Input/Output Control Register 6	$E86C_H$	0000_H
P3_IOCR07	Port 3 Input/Output Control Register 7	$E86E_H$	0000_H

7.3.5 Port 4

Port 4 is an 8-bit GPIO port. The registers of Port 4 are shown in Figure 7-8.
For this port, all pins can be read as GPIO, from the Port Input Register.

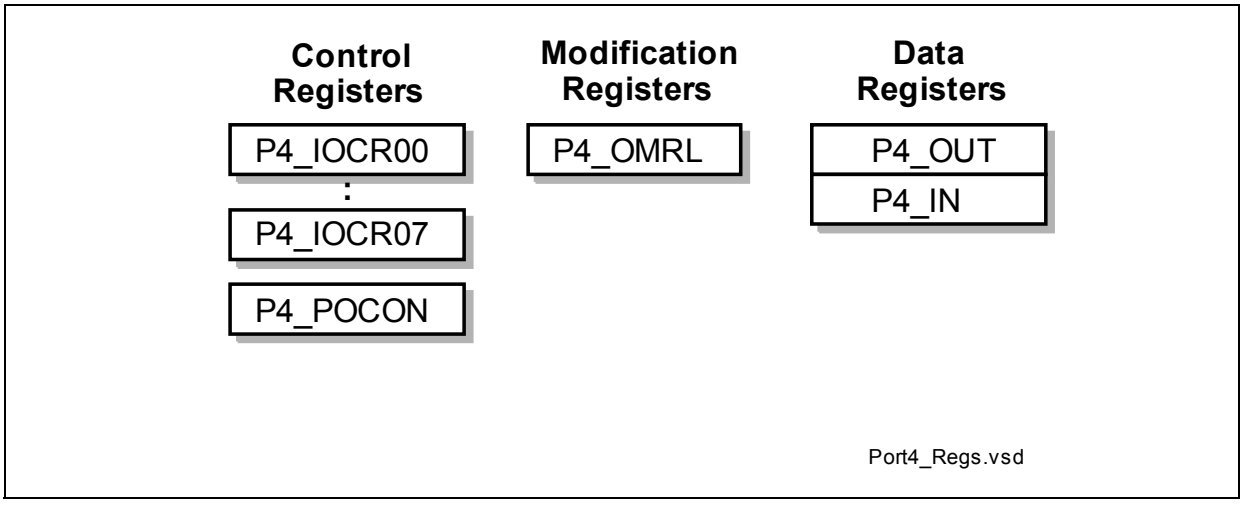


Figure 7-8 Port 4 Register Overview

Table 7-9 Port 4 Registers

Register Short Name	Register Long Name	Address Offset	Reset Value
P4_OUT	Port 4 Output Register	$FFAA_H$	0000_H
P4_IN	Port 4 Input Register	$FF88_H$	0000_H
P4_OMRL	Port 4 Output Modification Register Low	$E9D0_H$	$XXXX_H$
P4_POCON	Port 4 Output Control Register	$E8A8_H$	0000_H
P4_IOCR00	Port 4 Input/Output Control Register 0	$E880_H$	0000_H
P4_IOCR01	Port 4 Input/Output Control Register 1	$E882_H$	0000_H
P4_IOCR02	Port 4 Input/Output Control Register 2	$E884_H$	0000_H
P4_IOCR03	Port 4 Input/Output Control Register 3	$E886_H$	0000_H
P4_IOCR04	Port 4 Input/Output Control Register 4	$E888_H$	0000_H
P4_IOCR05	Port 4 Input/Output Control Register 5	$E88A_H$	0000_H
P4_IOCR06	Port 4 Input/Output Control Register 6	$E88C_H$	0000_H
P4_IOCR07	Port 4 Input/Output Control Register 7	$E88E_H$	0000_H

7.3.6 Port 5

Port 5 is an 16-bit analog or digital input port.

To use the Port 5 as an analog input, the Schmitt trigger in the input stage must be disabled. This is achieved by setting the corresponding bit in the register P5_DIDIS.

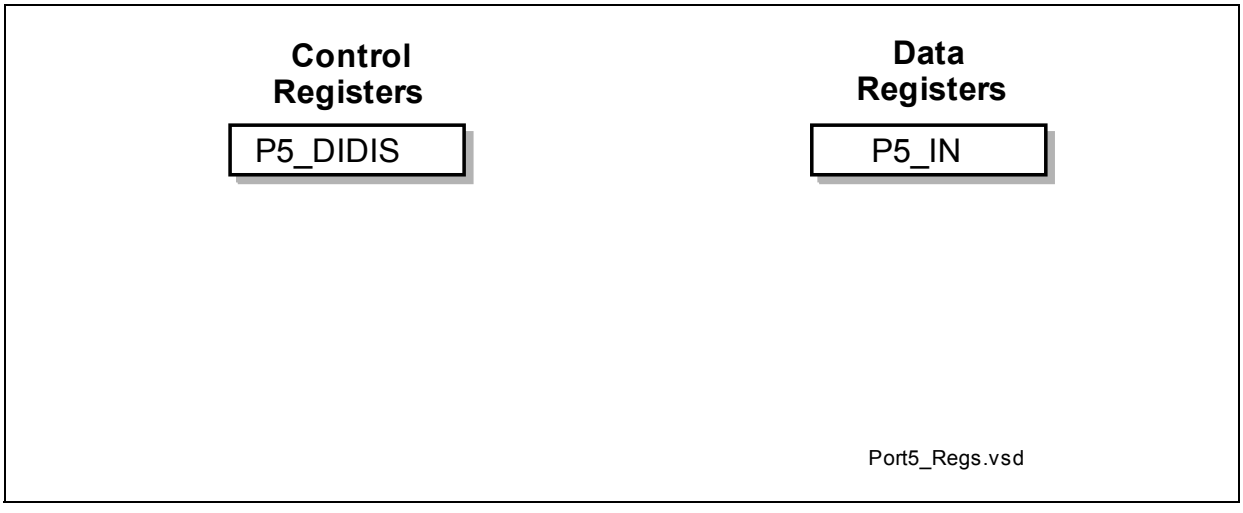


Figure 7-9 Port 5 Register Overview

Table 7-10 Port 5 Registers

Register Short Name	Register Long Name	Address Offset	Reset Value
P5_IN	Port 5 Input Register	$FF8A_H$	0000_H
P5_DIDIS	Port 5 Digital Input Disable Register	$FE8A_H$	0000_H

7.3.7 Port 6

Port 6 is an 4-bit GPIO port. The registers of Port 6 are shown in **Figure 7-10**.
For this port, all pins can be read as GPIO, from the Port Input Register.

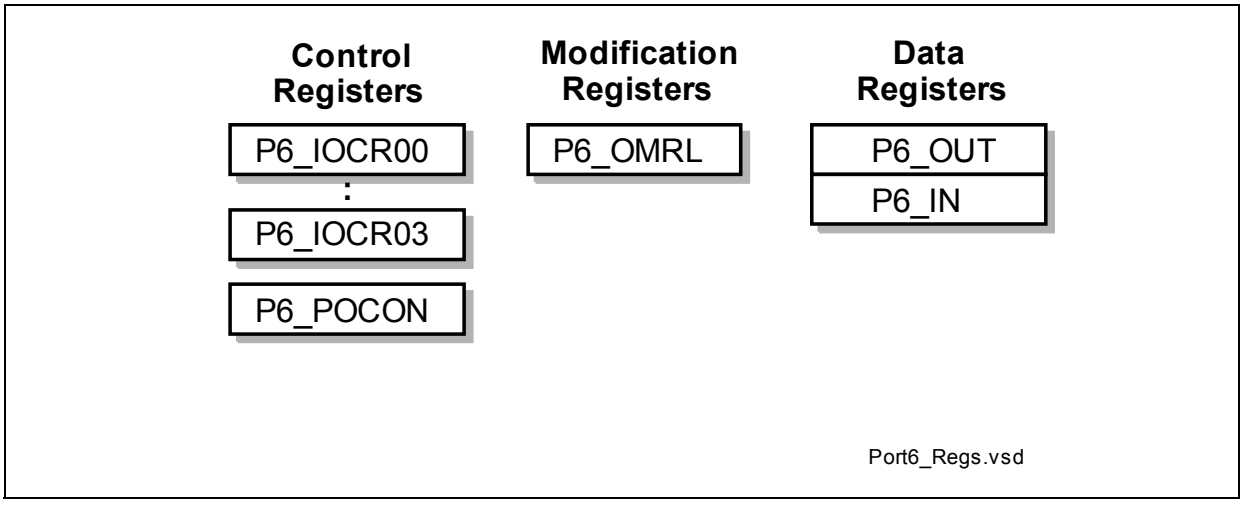


Figure 7-10 Port 6 Register Overview

Table 7-11 Port 6 Registers

Register Short Name	Register Long Name	Address Offset	Reset Value
P6_OUT	Port 6 Output Register	$FFAE_H$	0000_H
P6_IN	Port 6 Input Register	$FF8C_H$	0000_H
P6_OMRL	Port 6 Output Modification Register Low	$E9D8_H$	$XXXX_H$
P6_POCON	Port 6 Output Control Register	$E8AC_H$	0000_H
P6_IOCR00	Port 6 Input/Output Control Register 0	$E8C0_H$	0000_H
P6_IOCR01	Port 6 Input/Output Control Register 1	$E8C2_H$	0000_H
P6_IOCR02	Port 6 Input/Output Control Register 2	$E8C4_H$	0000_H
P6_IOCR03	Port 6 Input/Output Control Register 4	$E8C6_H$	0000_H

7.3.8 Port 7

Port 7 is a 5-bit GPIO port. The port registers of Port 7 are shown in **Figure 7-11**.
For this port, all pins can be read as GPIO, from the Port Input Register.

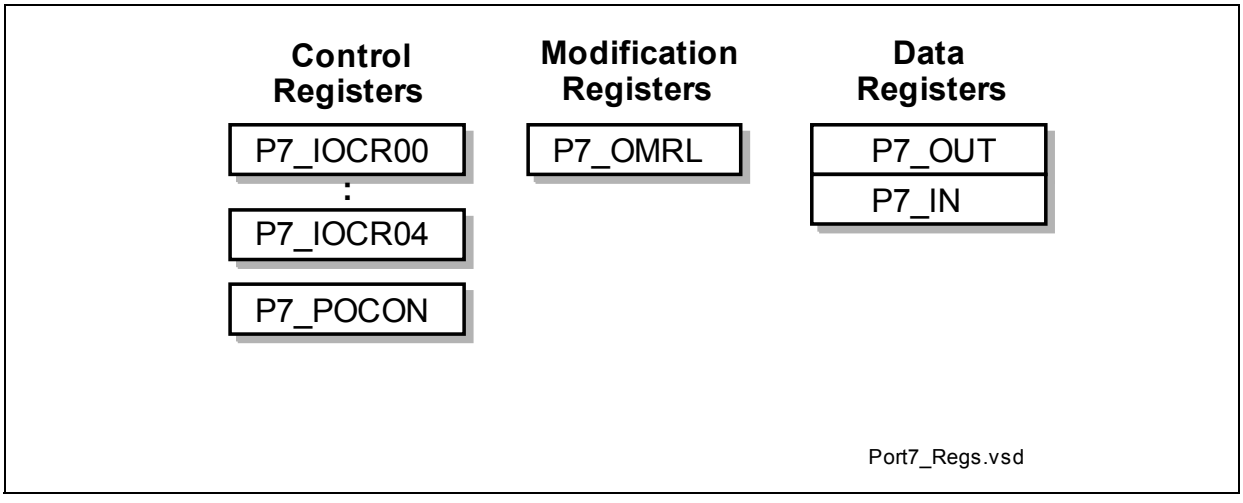


Figure 7-11 Port 7 Register Overview

Table 7-12 Port 7 Registers

Register Short Name	Register Long Name	Address Offset	Reset Value
P7_OUT	Port 7 Output Register	$FFB0_H$	0000_H
P7_IN	Port 7 Input Register	$FF8E_H$	0000_H
P7_OMRL	Port 7 Output Modification Register Low	$E9DC_H$	$XXXX_H$
P7_POCON	Port 7 Output Control Register	$E8AE_H$	0000_H
P7_IOCR00	Port 7 Input/Output Control Register 0	$E8E0_H$	0000_H
P7_IOCR01	Port 7 Input/Output Control Register 1	$E8E2_H$	0000_H
P7_IOCR02	Port 7 Input/Output Control Register 2	$E8E4_H$	0000_H
P7_IOCR03	Port 7 Input/Output Control Register 3	$E8E6_H$	0000_H
P7_IOCR04	Port 7 Input/Output Control Register 4	$E8E8_H$	0000_H

7.3.9 Port 8

Port 8 is an 7-bit GPIO port. The registers of Port 8 are shown in **Figure 7-12**.
For this port, all pins can be read as GPIO, from the Port Input Register.

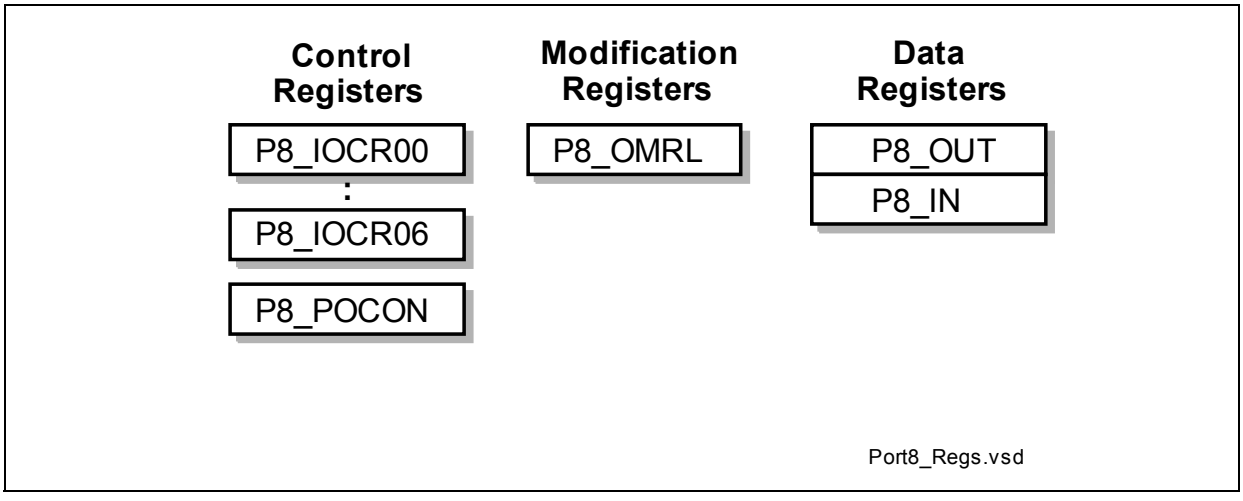


Figure 7-12 Port 8 Register Overview

Table 7-13 Port 8 Registers

Register Short Name	Register Long Name	Address Offset	Reset Value
P8_OUT	Port 8 Output Register	$FFB2_H$	0000_H
P8_IN	Port 8 Input Register	$FF90_H$	0000_H
P8_OMRL	Port 8 Output Modification Register Low	$E9E0_H$	$XXXX_H$
P8_POCON	Port 8 Output Control Register	$E8B0_H$	0000_H
P8_IOCR00	Port 8 Input/Output Control Register 0	$E900_H$	0000_H
P8_IOCR01	Port 8 Input/Output Control Register 1	$E902_H$	0000_H
P8_IOCR02	Port 8 Input/Output Control Register 2	$E904_H$	0000_H
P8_IOCR03	Port 8 Input/Output Control Register 3	$E906_H$	0000_H
P8_IOCR04	Port 8 Input/Output Control Register 4	$E908_H$	0000_H
P8_IOCR05	Port 8 Input/Output Control Register 5	$E90A_H$	0000_H
P8_IOCR06	Port 8 Input/Output Control Register 6	$E90C_H$	0000_H

7.3.10 Port 9

Port 9 is an 8-bit GPIO port. The port registers of Port 9 are shown in Figure 7-13.
For this port, all pins can be read as GPIO, from the Port Input Register.

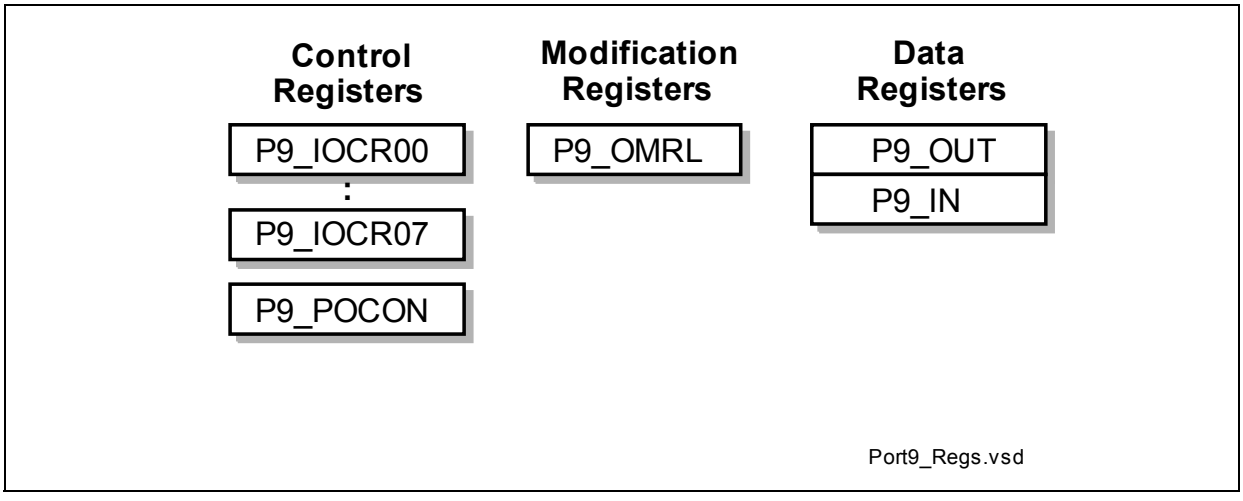


Figure 7-13 Port 9 Register Overview

Table 7-14 Port 9 Registers

Register Short Name	Register Long Name	Address Offset	Reset Value
P9_OUT	Port 9 Output Register	$FFB4_H$	0000_H
P9_IN	Port 9 Input Register	$FF92_H$	0000_H
P9_OMRL	Port 9 Output Modification Register Low	$E9E4_H$	$XXXX_H$
P9_POCON	Port 9 Output Control Register	$E8B2_H$	0000_H
P9_IOCR00	Port 9 Input/Output Control Register 0	$E920_H$	0000_H
P9_IOCR01	Port 9 Input/Output Control Register 1	$E922_H$	0000_H
P9_IOCR02	Port 9 Input/Output Control Register 2	$E924_H$	0000_H
P9_IOCR03	Port 9 Input/Output Control Register 3	$E926_H$	0000_H
P9_IOCR04	Port 9 Input/Output Control Register 4	$E928_H$	0000_H
P9_IOCR05	Port 9 Input/Output Control Register 5	$E92A_H$	0000_H
P9_IOCR06	Port 9 Input/Output Control Register 6	$E92C_H$	0000_H
P9_IOCR07	Port 9 Input/Output Control Register 7	$E92E_H$	0000_H

7.3.11 Port 10

Port 10 is a 16-bit GPIO port. The registers of Port 10 are shown in Figure 7-14.
For this port, all pins can be read as GPIO, from the Port Input Register.

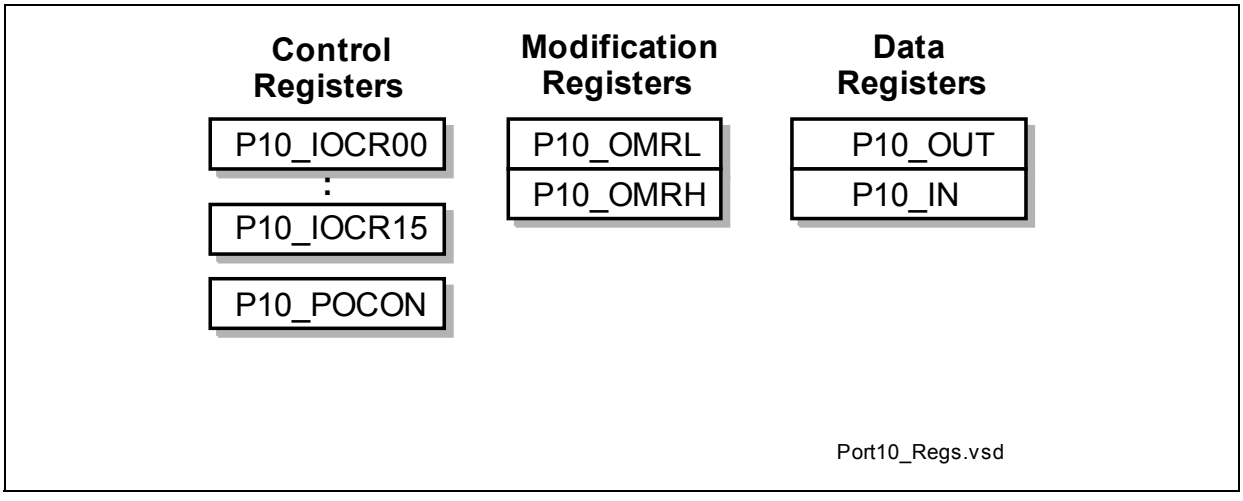


Figure 7-14 Port 10 Register Overview

Table 7-15 Port 10 Registers

Register Short Name	Register Long Name	Address Offset	Reset Value
P10_OUT	Port 10 Output Register	$FFB6_H$	0000_H
P10_IN	Port 10 Input Register	$FF94_H$	0000_H
P10_OMRL	Port 10 Output Modification Register Low	$E9E8_H$	$XXXX_H$
P10_OMRH	Port 10 Output Modification Register High	$E9EA_H$	$XXXX_H$
P10_POCON	Port 10 Output Control Register	$E8B4_H$	0000_H
P10_IOCR00	Port 10 Input/Output Control Register 0	$E940_H$	0000_H
P10_IOCR01	Port 10 Input/Output Control Register 1	$E942_H$	0000_H
P10_IOCR02	Port 10 Input/Output Control Register 2	$E944_H$	0000_H
P10_IOCR03	Port 10 Input/Output Control Register 3	$E946_H$	0000_H
P10_IOCR04	Port 10 Input/Output Control Register 4	$E948_H$	0000_H
P10_IOCR05	Port 10 Input/Output Control Register 5	$E94A_H$	0000_H
P10_IOCR06	Port 10 Input/Output Control Register 6	$E94C_H$	0000_H
P10_IOCR07	Port 10 Input/Output Control Register 7	$E94E_H$	0000_H
P10_IOCR08	Port 10 Input/Output Control Register 8	$E950_H$	0000_H
P10_IOCR09	Port 10 Input/Output Control Register 9	$E952_H$	0000_H

Table 7-15 Port 10 Registers (cont'd)

Register Short Name	Register Long Name	Address Offset	Reset Value
P10_IOCR10	Port 10 Input/Output Control Register 10	$E954_H$	0000_H
P10_IOCR11	Port 10 Input/Output Control Register 11	$E956_H$	0000_H
P10_IOCR12	Port 10 Input/Output Control Register 12	$E958_H$	0000_H
P10_IOCR13	Port 10 Input/Output Control Register 13	$E95A_H$	0000_H
P10_IOCR14	Port 10 Input/Output Control Register 14	$E95C_H$	0000_H
P10_IOCR15	Port 10 Input/Output Control Register 15	$E95E_H$	0000_H

7.3.12 Port 11

Port 11 is an 6-bit GPIO port. The registers of Port 11 are shown in Figure 7-15.
For this port, all pins can be read as GPIO, from the Port Input Register.

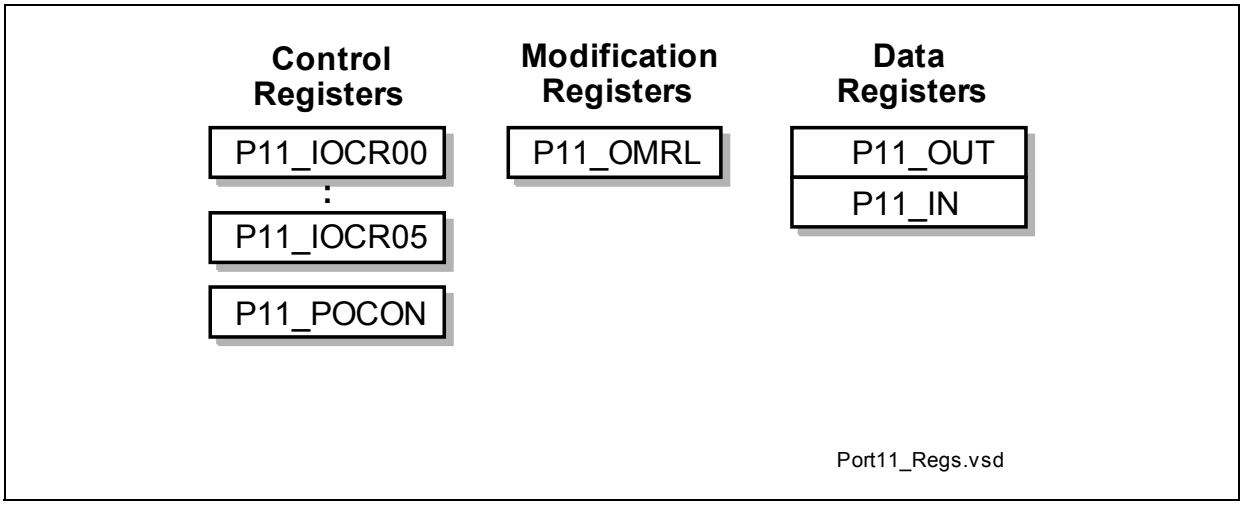


Figure 7-15 Port 11 Register Overview

Table 7-16 Port 11 Registers

Register Short Name	Register Long Name	Address Offset	Reset Value
P11_OUT	Port 11 Output Register	$FFB8_H$	0000_H
P11_IN	Port 11 Input Register	$FF96_H$	0000_H
P11_OMRL	Port 11 Output Modification Register Low	$E9EC_H$	$XXXX_H$
P11_POCON	Port 11 Output Control Register	$E8B6_H$	0000_H
P11_IOCR00	Port 11 Input/Output Control Register 0	$E960_H$	0000_H
P11_IOCR01	Port 11 Input/Output Control Register 1	$E962_H$	0000_H
P11_IOCR02	Port 11 Input/Output Control Register 2	$E964_H$	0000_H
P11_IOCR03	Port 11 Input/Output Control Register 3	$E966_H$	0000_H
P11_IOCR04	Port 11 Input/Output Control Register 4	$E968_H$	0000_H
P11_IOCR05	Port 11 Input/Output Control Register 5	$E96A_H$	0000_H

7.3.13 Port 15

Port 15 is an 8-bit analog or digital input port. To use the Port 15 as an analog input, the Schmitt trigger in the input stage must be disabled. This is achieved by setting the corresponding bit in the register P15_DIDIS.

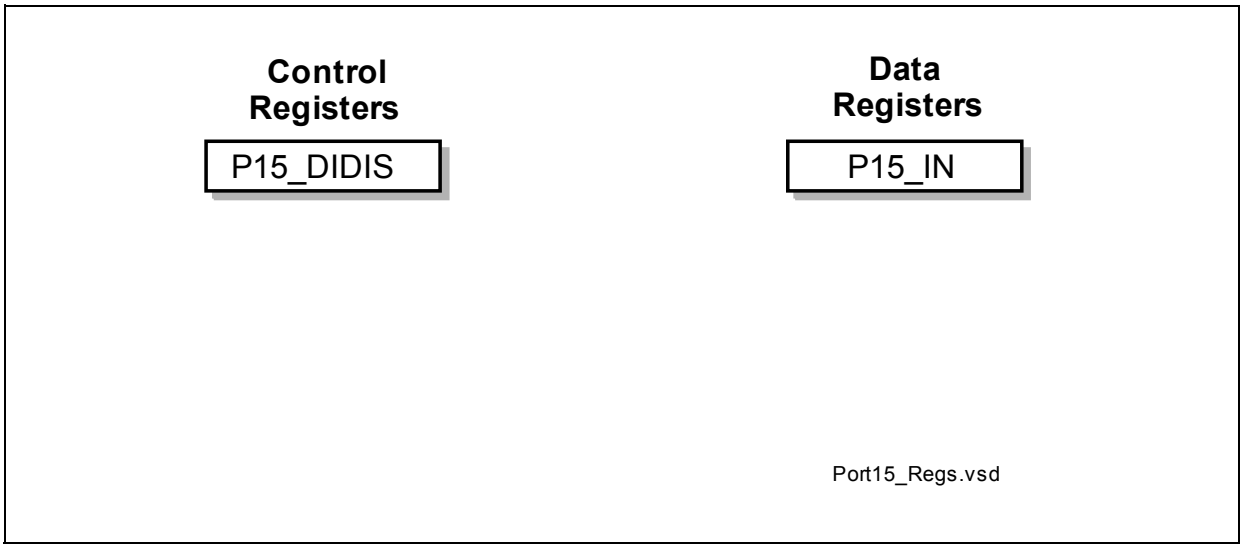


Figure 7-16 Port 15 Register Overview

Table 7-17 Port 15 Registers

Register Short Name	Register Long Name	Address Offset	Reset Value
P15_IN	Port 15 Input Register	$FF9E_H$	0000_H
P15_DIDIS	Port 15 Digital Input Disable Register	$FE9E_H$	0000_H

7.4 Pin Description

Each port pin of the XE16x can serve several functions of different modules. Also, most functions are available on several port pins. This enables an application so select the optimal connections for its specific circumstances.

A pin can output its own port output signal or one of up to three signals coming from the peripherals. Its input signal is available in its own input register and at several peripherals.

Note: Output signals are selected at the respective port pin, input signals are selected at the respective peripheral.

Optionally a pin can be fully controlled by a peripheral, in case the peripheral is enabled (for example, EBC).

Table 7-18 summarizes the various function of each pin of the XE16x.

Notes to Pin Definitions

1. ***Ctrl.**: The output signal for a port pin is selected via bitfield PC in the associated register Px_IOCRy. Output O0 is selected by setting the respective bitfield PC to $1x00_B$, output O1 is selected by $1x01_B$, etc.*
   Output signal OH is controlled by hardware.
2. ***Type**: Indicates the employed pad type (St=standard pad, Sp=special pad, DP=double pad, In=input pad, PS=power supply) and its power supply domain (A, B, M, 1).*

Table 7-18 Pin Definitions and Functions

Pin	Symbol	Ctrl.	Type	Function
3	$\overline{\text{TESTM}}$	I	In/B	**Testmode Enable** Enables factory test modes, must be held HIGH for normal operation (connect to V_{DDPB}). An internal pullup device will hold this pin high when nothing is driving it.
4	P7.2	O0 / I	St/B	**Bit 2 of Port 7, General Purpose Input/Output**
	TxDC4	O2	St/B	**CAN Node 4 Transmit Data Output**
	CCU62_CCPOS0A	I	St/B	**CCU62 Position Input 0**
	TDI_C	I	St/B	**JTAG Test Data Input**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
5	P8.4	O0 / I	St/B	**Bit 4 of Port 8, General Purpose Input/Output**
	CCU60_COUT61	O1	St/B	**CCU60 Channel 1 Output**
	TMS_D	I	St/B	**JTAG Test Mode Selection Input**
6	$\overline{\text{TRST}}$	I	In/B	**Test-System Reset Input** For normal system operation, pin $\overline{\text{TRST}}$ should be held low. A high level at this pin at the rising edge of $\overline{\text{PORST}}$ activates the XE16x's debug system. In this case, pin $\overline{\text{TRST}}$ must be driven low once to reset the debug system. An internal pulldown device will hold this pin low when nothing is driving it.
7	P8.3	O0 / I	St/B	**Bit 3 of Port 8, General Purpose Input/Output**
	CCU60_COUT60	O1	St/B	**CCU60 Channel 0 Output**
	TDI_D	I	St/B	**JTAG Test Data Input**
8	P7.0	O0 / I	St/B	**Bit 0 of Port 7, General Purpose Input/Output**
	T3OUT	O1	St/B	**GPT1 Timer T3 Toggle Latch Output**
	T6OUT	O2	St/B	**GPT2 Timer T6 Toggle Latch Output**
	TDO	OH	St/B	**JTAG Test Data Output**
	ESR2_1	I	St/B	**ESR2 Trigger Input 1**
	RxDC4B	I	St/B	**CAN Node 4 Receive Data Input**
9	P7.3	O0 / I	St/B	**Bit 3 of Port 7, General Purpose Input/Output**
	U0C1_DOUT	O2	St/B	**USIC0 Channel 1 Shift Data Output**
	U0C0_DOUT	O3	St/B	**USIC0 Channel 0 Shift Data Output**
	CCU62_CCPOS1A	I	St/B	**CCU62 Position Input 1**
	TMS_C	I	St/B	**JTAG Test Mode Selection Input**
	U0C1_DX0F	I	St/B	**USIC0 Channel 1 Shift Data Input**
10	P8.2	O0 / I	St/B	**Bit 2 of Port 8, General Purpose Input/Output**
	CCU60_CC62	O1 / I	St/B	**CCU60 Channel 2 Input/Output**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
11	P7.1	O0 / I	St/B	**Bit 1 of Port 7, General Purpose Input/Output**
	EXTCLK	O1	St/B	**Programmable Clock Signal Output**
	TxDC4	O2	St/B	**CAN Node 4 Transmit Data Output**
	CCU62_CTRAPA	I	St/B	**CCU62 Emergency Trap Input**
	$\overline{\text{BRKIN\_C}}$	I	St/B	**OCDS Break Signal Input**
12	P7.4	O0 / I	St/B	**Bit 4 of Port 7, General Purpose Input/Output**
	U0C1_DOUT	O2	St/B	**USIC0 Channel 1 Shift Data Output**
	U0C1_SCLK	O3	St/B	**USIC0 Channel 1 Shift Clock Output**
	CCU62_CCPOS2A	I	St/B	**CCU62 Position Input 2**
	TCK_C	I	St/B	**JTAG Clock Input**
	U0C0_DX0D	I	St/B	**USIC0 Channel 0 Shift Data Input**
	U0C1_DX1E	I	St/B	**USIC0 Channel 1 Shift Clock Input**
13	P8.1	O0 / I	St/B	**Bit 1 of Port 8, General Purpose Input/Output**
	CCU60_CC61	O1 / I	St/B	**CCU60 Channel 1 Input/Output**
14	P8.0	O0 / I	St/B	**Bit 0 of Port 8, General Purpose Input/Output**
	CCU60_CC60	O1 / I	St/B	**CCU60 Channel 0 Input/Output**
16	P6.0	O0 / I	St/A	**Bit 0 of Port 6, General Purpose Input/Output**
	EMUX0	O1	St/A	**External Analog MUX Control Output 0 (ADC0)**
	$\overline{\text{BRKOUT}}$	O3	St/A	**OCDS Break Signal Output**
	ADCx_REQGTyC	I	St/A	**External Request Gate Input for ADC0/1**
	U1C1_DX0E	I	St/A	**USIC1 Channel 1 Shift Data Input**
17	P6.1	O0 / I	St/A	**Bit 1 of Port 6, General Purpose Input/Output**
	EMUX1	O1	St/A	**External Analog MUX Control Output 1 (ADC0)**
	T3OUT	O2	St/A	**GPT1 Timer T3 Toggle Latch Output**
	U1C1_DOUT	O3	St/A	**USIC1 Channel 1 Shift Data Output**
	ADCx_REQTRyC	I	St/A	**External Request Trigger Input for ADC0/1**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
18	P6.2	O0 / I	St/A	**Bit 2 of Port 6, General Purpose Input/Output**
	EMUX2	O1	St/A	**External Analog MUX Control Output 2 (ADC0)**
	T6OUT	O2	St/A	**GPT2 Timer T6 Toggle Latch Output**
	U1C1_SCLK	O3	St/A	**USIC1 Channel 1 Shift Clock Output**
	U1C1_DX1C	I	St/A	**USIC1 Channel 1 Shift Clock Input**
19	P6.3	O0 / I	St/A	**Bit 3 of Port 6, General Purpose Input/Output**
	T3OUT	O2	St/A	**GPT1 Timer T3 Toggle Latch Output**
	U1C1_SELO 0	O3	St/A	**USIC1 Channel 1 Select/Control 0 Output**
	U1C1_DX2D	I	St/A	**USIC1 Channel 1 Shift Control Input**
	ADCx_REQT RyD	I	St/A	**External Request Trigger Input for ADC0/1**
21	P15.0	I	In/A	**Bit 0 of Port 15, General Purpose Input**
	ADC1_CH0	I	In/A	**Analog Input Channel 0 for ADC1**
22	P15.1	I	In/A	**Bit 1 of Port 15, General Purpose Input**
	ADC1_CH1	I	In/A	**Analog Input Channel 1 for ADC1**
23	P15.2	I	In/A	**Bit 2 of Port 15, General Purpose Input**
	ADC1_CH2	I	In/A	**Analog Input Channel 2 for ADC1**
	T5IN	I	In/A	**GPT2 Timer T5 Count/Gate Input**
24	P15.3	I	In/A	**Bit 3 of Port 15, General Purpose Input**
	ADC1_CH3	I	In/A	**Analog Input Channel 3 for ADC1**
	T5EUD	I	In/A	**GPT2 Timer T5 External Up/Down Control Input**
25	P15.4	I	In/A	**Bit 4 of Port 15, General Purpose Input**
	ADC1_CH4	I	In/A	**Analog Input Channel 4 for ADC1**
	T6IN	I	In/A	**GPT2 Timer T6 Count/Gate Input**
26	P15.5	I	In/A	**Bit 5 of Port 15, General Purpose Input**
	ADC1_CH5	I	In/A	**Analog Input Channel 5 for ADC1**
	T6EUD	I	In/A	**GPT2 Timer T6 External Up/Down Control Input**
27	P15.6	I	In/A	**Bit 6 of Port 15, General Purpose Input**
	ADC1_CH6	I	In/A	**Analog Input Channel 6 for ADC1**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
28	P15.7	I	In/A	**Bit 7 of Port 15, General Purpose Input**
	ADC1_CH7	I	In/A	**Analog Input Channel 7 for ADC1**
29	V_{AREF1}	-	PS/A	**Reference Voltage for A/D Converter ADC1**
30	V_{AREF0}	-	PS/A	**Reference Voltage for A/D Converter ADC0**
31	V_{AGND}	-	PS/A	**Reference Ground for A/D Converters ADC0/1**
32	P5.0	I	In/A	**Bit 0 of Port 5, General Purpose Input**
	ADC0_CH0	I	In/A	**Analog Input Channel 0 for ADC0**
33	P5.1	I	In/A	**Bit 1 of Port 5, General Purpose Input**
	ADC0_CH1	I	In/A	**Analog Input Channel 1 for ADC0**
34	P5.2	I	In/A	**Bit 2 of Port 5, General Purpose Input**
	ADC0_CH2	I	In/A	**Analog Input Channel 2 for ADC0**
	TDI_A	I	In/A	**JTAG Test Data Input**
35	P5.3	I	In/A	**Bit 3 of Port 5, General Purpose Input**
	ADC0_CH3	I	In/A	**Analog Input Channel 3 for ADC0**
	T3IN	I	In/A	**GPT1 Timer T3 Count/Gate Input**
39	P5.4	I	In/A	**Bit 4 of Port 5, General Purpose Input**
	ADC0_CH4	I	In/A	**Analog Input Channel 4 for ADC0**
	CCU63_T12HRB	I	In/A	**External Run Control Input for T12 of CCU63**
	T3EUD	I	In/A	**GPT1 Timer T3 External Up/Down Control Input**
	TMS_A	I	In/A	**JTAG Test Mode Selection Input**
40	P5.5	I	In/A	**Bit 5 of Port 5, General Purpose Input**
	ADC0_CH5	I	In/A	**Analog Input Channel 5 for ADC0**
	CCU60_T12HRB	I	In/A	**External Run Control Input for T12 of CCU60**
41	P5.6	I	In/A	**Bit 6 of Port 5, General Purpose Input**
	ADC0_CH6	I	In/A	**Analog Input Channel 6 for ADC0**
42	P5.7	I	In/A	**Bit 7 of Port 5, General Purpose Input**
	ADC0_CH7	I	In/A	**Analog Input Channel 7 for ADC0**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
43	P5.8	I	In/A	**Bit 8 of Port 5, General Purpose Input**
	ADC0_CH8	I	In/A	**Analog Input Channel 8 for ADC0**
	CCU6x_T12HRC	I	In/A	**External Run Control Input for T12 of CCU60/1/2/3**
	CCU6x_T13HRC	I	In/A	**External Run Control Input for T13 of CCU60/1/2/3**
44	P5.9	I	In/A	**Bit 9 of Port 5, General Purpose Input**
	ADC0_CH9	I	In/A	**Analog Input Channel 9 for ADC0**
	CC2_T7IN	I	In/A	**CAPCOM2 Timer T7 Count Input**
45	P5.10	I	In/A	**Bit 10 of Port 5, General Purpose Input**
	ADC0_CH10	I	In/A	**Analog Input Channel 10 for ADC0**
	$\overline{\text{BRKIN\_A}}$	I	In/A	**OCDS Break Signal Input**
46	P5.11	I	In/A	**Bit 11 of Port 5, General Purpose Input**
	ADC0_CH11	I	In/A	**Analog Input Channel 11 for ADC0**
47	P5.12	I	In/A	**Bit 12 of Port 5, General Purpose Input**
	ADC0_CH12	I	In/A	**Analog Input Channel 12 for ADC0**
48	P5.13	I	In/A	**Bit 13 of Port 5, General Purpose Input**
	ADC0_CH13	I	In/A	**Analog Input Channel 13 for ADC0**
	EX0BINB	I	In/A	**External Interrupt Trigger Input**
49	P5.14	I	In/A	**Bit 14 of Port 5, General Purpose Input**
	ADC0_CH14	I	In/A	**Analog Input Channel 14 for ADC0**
50	P5.15	I	In/A	**Bit 15 of Port 5, General Purpose Input**
	ADC0_CH15	I	In/A	**Analog Input Channel 15 for ADC0**
51	P2.12	O0 / I	St/B	**Bit 12 of Port 2, General Purpose Input/Output**
	U0C0_SELO4	O1	St/B	**USIC0 Channel 0 Select/Control 4 Output**
	U0C1_SELO3	O2	St/B	**USIC0 Channel 1 Select/Control 3 Output**
	READY	I	St/B	**External Bus Interface READY Input**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
52	P2.11	O0 / I	St/B	**Bit 11 of Port 2, General Purpose Input/Output**
	U0C0_SELO 2	O1	St/B	**USIC0 Channel 0 Select/Control 2 Output**
	U0C1_SELO 2	O2	St/B	**USIC0 Channel 1 Select/Control 2 Output**
	$\overline{BHE}$/$\overline{WRH}$	OH	St/B	**External Bus Interf. High-Byte Control Output** Can operate either as Byte High Enable ($\overline{BHE}$) or as Write strobe for High Byte ($\overline{WRH}$).
53	P11.5	O0 / I	St/B	**Bit 5 of Port 11, General Purpose Input/Output**
55	P2.0	O0 / I	St/B	**Bit 0 of Port 2, General Purpose Input/Output**
	CCU63_CC6 0	O2 / I	St/B	**CCU63 Channel 0 Input/Output**
	AD13	OH / I	St/B	**External Bus Interface Address/Data Line 13**
	RxDC0C	I	St/B	**CAN Node 0 Receive Data Input**
56	P2.1	O0 / I	St/B	**Bit 1 of Port 2, General Purpose Input/Output**
	TxDC0	O1	St/B	**CAN Node 0 Transmit Data Output**
	CCU63_CC6 1	O2 / I	St/B	**CCU63 Channel 1 Input/Output**
	AD14	OH / I	St/B	**External Bus Interface Address/Data Line 14**
	ESR1_5	I	St/B	**ESR1 Trigger Input 5**
	EX0AINA	I	St/B	**External Interrupt Trigger Input**
57	P11.4	O0 / I	St/B	**Bit 4 of Port 11, General Purpose Input/Output**
58	P2.2	O0 / I	St/B	**Bit 2 of Port 2, General Purpose Input/Output**
	TxDC1	O1	St/B	**CAN Node 1 Transmit Data Output**
	CCU63_CC6 2	O2 / I	St/B	**CCU63 Channel 2 Input/Output**
	AD15	OH / I	St/B	**External Bus Interface Address/Data Line 15**
	ESR2_5	I	St/B	**ESR2 Trigger Input 5**
	EX1AINA	I	St/B	**External Interrupt Trigger Input**
59	P11.3	O0 / I	St/B	**Bit 3 of Port 11, General Purpose Input/Output**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
60	P4.0	O0 / I	St/B	**Bit 0 of Port 4, General Purpose Input/Output**
	CC2_24	O3 / I	St/B	**CAPCOM2 CC24IO Capture Inp./ Compare Out.**
	$\overline{\text{CS0}}$	OH	St/B	**External Bus Interface Chip Select 0 Output**
61	P2.3	O0 / I	St/B	**Bit 3 of Port 2, General Purpose Input/Output**
	U0C0_DOUT	O1	St/B	**USIC0 Channel 0 Shift Data Output**
	CCU63_COUT63	O2	St/B	**CCU63 Channel 3 Output**
	CC2_16	O3 / I	St/B	**CAPCOM2 CC16IO Capture Inp./ Compare Out.**
	A16	OH	St/B	**External Bus Interface Address Line 16**
	ESR2_0	I	St/B	**ESR2 Trigger Input 0**
	U0C0_DX0E	I	St/B	**USIC0 Channel 0 Shift Data Input**
	U0C1_DX0D	I	St/B	**USIC0 Channel 1 Shift Data Input** *Note: Not available in step AA.*
	RxDC0A	I	St/B	**CAN Node 0 Receive Data Input**
62	P11.2	O0 / I	St/B	**Bit 2 of Port 11, General Purpose Input/Output**
	CCU63_CCPOS2A	I	St/B	**CCU63 Position Input 2**
63	P4.1	O0 / I	St/B	**Bit 1 of Port 4, General Purpose Input/Output**
	TxDC2	O2	St/B	**CAN Node 2 Transmit Data Output**
	CC2_25	O3 / I	St/B	**CAPCOM2 CC25IO Capture Inp./ Compare Out.**
	$\overline{\text{CS1}}$	OH	St/B	**External Bus Interface Chip Select 1 Output**
64	P2.4	O0 / I	St/B	**Bit 4 of Port 2, General Purpose Input/Output**
	U0C1_DOUT	O1	St/B	**USIC0 Channel 1 Shift Data Output** *Note: Not available in step AA.*
	TxDC0	O2	St/B	**CAN Node 0 Transmit Data Output**
	CC2_17	O3 / I	St/B	**CAPCOM2 CC17IO Capture Inp./ Compare Out.**
	A17	OH	St/B	**External Bus Interface Address Line 17**
	ESR1_0	I	St/B	**ESR1 Trigger Input 0**
	U0C0_DX0F	I	St/B	**USIC0 Channel 0 Shift Data Input**
	RxDC1A	I	St/B	**CAN Node 1 Receive Data Input**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
65	P11.1	O0 / I	St/B	**Bit 1 of Port 11, General Purpose Input/Output**
	CCU63_CCPOS1A	I	St/B	**CCU63 Position Input 1**
66	P11.0	O0 / I	St/B	**Bit 0 of Port 11, General Purpose Input/Output**
	CCU63_CCPOS0A	I	St/B	**CCU63 Position Input 0**
67	P2.5	O0 / I	St/B	**Bit 5 of Port 2, General Purpose Input/Output**
	U0C0_SCLKOUT	O1	St/B	**USIC0 Channel 0 Shift Clock Output**
	TxDC0	O2	St/B	**CAN Node 0 Transmit Data Output**
	CC2_18	O3 / I	St/B	**CAPCOM2 CC18IO Capture Inp./ Compare Out.**
	A18	OH	St/B	**External Bus Interface Address Line 18**
	U0C0_DX1D	I	St/B	**USIC0 Channel 0 Shift Clock Input**
68	P4.2	O0 / I	St/B	**Bit 2 of Port 4, General Purpose Input/Output**
	TxDC2	O2	St/B	**CAN Node 2 Transmit Data Output**
	CC2_26	O3 / I	St/B	**CAPCOM2 CC26IO Capture Inp./ Compare Out.**
	$\overline{CS2}$	OH	St/B	**External Bus Interface Chip Select 2 Output**
	T2IN	I	St/B	**GPT1 Timer T2 Count/Gate Input**
69	P2.6	O0 / I	St/B	**Bit 6 of Port 2, General Purpose Input/Output**
	U0C0_SELO0	O1	St/B	**USIC0 Channel 0 Select/Control 0 Output**
	U0C1_SELO1	O2	St/B	**USIC0 Channel 1 Select/Control 1 Output**
	CC2_19	O3 / I	St/B	**CAPCOM2 CC19IO Capture Inp./ Compare Out.**
	A19	OH	St/B	**External Bus Interface Address Line 19**
	U0C0_DX2D	I	St/B	**USIC0 Channel 0 Shift Control Input**
	RxDC0D	I	St/B	**CAN Node 0 Receive Data Input**
70	P4.4	O0 / I	St/B	**Bit 4 of Port 4, General Purpose Input/Output**
	CC2_28	O3 / I	St/B	**CAPCOM2 CC28IO Capture Inp./ Compare Out.**
	$\overline{CS4}$	OH	St/B	**External Bus Interface Chip Select 4 Output**
	DRTC	I	St/B	**RTC Count Clock Signal Input**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
71	P4.3	O0 / I	St/B	**Bit 3 of Port 4, General Purpose Input/Output**
	CC2_27	O3 / I	St/B	**CAPCOM2 CC27IO Capture Inp./ Compare Out.**
	$\overline{CS3}$	OH	St/B	**External Bus Interface Chip Select 3 Output**
	RxDC2A	I	St/B	**CAN Node 2 Receive Data Input**
	T2EUD	I	St/B	**GPT1 Timer T2 External Up/Down Control Input**
75	P0.0	O0 / I	St/B	**Bit 0 of Port 0, General Purpose Input/Output**
	U1C0_DOUT	O1	St/B	**USIC1 Channel 0 Shift Data Output**
	CCU61_CC60	O3 / I	St/B	**CCU61 Channel 0 Input/Output**
	A0	OH	St/B	**External Bus Interface Address Line 0**
	U1C0_DX0A	I	St/B	**USIC1 Channel 0 Shift Data Input**
76	P4.5	O0 / I	St/B	**Bit 5 of Port 4, General Purpose Input/Output**
	CC2_29	O3 / I	St/B	**CAPCOM2 CC29IO Capture Inp./Compare Out.**
77	P4.6	O0 / I	St/B	**Bit 6 of Port 4, General Purpose Input/Output**
	CC2_30	O3 / I	St/B	**CAPCOM2 CC30IO Capture Inp./ Compare Out.**
	T4IN	I	St/B	**GPT1 Timer T4 Count/Gate Input**
78	P2.7	O0 / I	St/B	**Bit 7 of Port 2, General Purpose Input/Output**
	U0C1_SELO0	O1	St/B	**USIC0 Channel 1 Select/Control 0 Output**
	U0C0_SELO1	O2	St/B	**USIC0 Channel 0 Select/Control 1 Output**
	CC2_20	O3 / I	St/B	**CAPCOM2 CC20IO Capture Inp./ Compare Out.**
	A20	OH	St/B	**External Bus Interface Address Line 20**
	U0C1_DX2C	I	St/B	**USIC0 Channel 1 Shift Control Input**
	RxDC1C	I	St/B	**CAN Node 1 Receive Data Input**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
79	P0.1	O0 / I	St/B	**Bit 1 of Port 0, General Purpose Input/Output**
	U1C0_DOUT	O1	St/B	**USIC1 Channel 0 Shift Data Output**
	TxDC0	O2	St/B	**CAN Node 0 Transmit Data Output**
	CCU61_CC61	O3 / I	St/B	**CCU61 Channel 1 Input/Output**
	A1	OH	St/B	**External Bus Interface Address Line 1**
	U1C0_DX0B	I	St/B	**USIC1 Channel 0 Shift Data Input**
	U1C0_DX1A	I	St/B	**USIC1 Channel 0 Shift Clock Input**
80	P2.8	O0 / I	DP/B	**Bit 8 of Port 2, General Purpose Input/Output**
	U0C1_SCLKOUT	O1	DP/B	**USIC0 Channel 1 Shift Clock Output**
	EXTCLK	O2	DP/B	**Programmable Clock Signal Output** 1)
	CC2_21	O3 / I	DP/B	**CAPCOM2 CC21IO Capture Inp./ Compare Out.**
	A21	OH	DP/B	**External Bus Interface Address Line 21**
	U0C1_DX1D	I	DP/B	**USIC0 Channel 1 Shift Clock Input**
81	P4.7	O0 / I	St/B	**Bit 7 of Port 4, General Purpose Input/Output**
	CC2_31	O3 / I	St/B	**CAPCOM2 CC31IO Capture Inp./ Compare Out.**
	T4EUD	I	St/B	**GPT1 Timer T4 External Up/Down Control Input**
82	P2.9	O0 / I	St/B	**Bit 9 of Port 2, General Purpose Input/Output**
	U0C1_DOUT	O1	St/B	**USIC0 Channel 1 Shift Data Output**
	TxDC1	O2	St/B	**CAN Node 1 Transmit Data Output**
	CC2_22	O3 / I	St/B	**CAPCOM2 CC22IO Capture Inp./ Compare Out.**
	A22	OH	St/B	**External Bus Interface Address Line 22**
	DIRIN	I	St/B	**Clock Signal Input**
	TCK_A	I	St/B	**JTAG Clock Input**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
83	P0.2	O0 / I	St/B	**Bit 2 of Port 0, General Purpose Input/Output**
	U1C0_SCLKOUT	O1	St/B	**USIC1 Channel 0 Shift Clock Output**
	TxDC0	O2	St/B	**CAN Node 0 Transmit Data Output**
	CCU61_CC62	O3 / I	St/B	**CCU61 Channel 2 Input/Output**
	A2	OH	St/B	**External Bus Interface Address Line 2**
	U1C0_DX1B	I	St/B	**USIC1 Channel 0 Shift Clock Input**
84	P10.0	O0 / I	St/B	**Bit 0 of Port 10, General Purpose Input/Output**
	U0C1_DOUT	O1	St/B	**USIC0 Channel 1 Shift Data Output**
	CCU60_CC60	O2 / I	St/B	**CCU60 Channel 0 Input/Output**
	AD0	OH / I	St/B	**External Bus Interface Address/Data Line 0**
	ESR1_2	I	St/B	**ESR1 Trigger Input 2**
	U0C0_DX0A	I	St/B	**USIC0 Channel 0 Shift Data Input**
	U0C1_DX0A	I	St/B	**USIC0 Channel 1 Shift Data Input**
85	P3.0	O0 / I	St/B	**Bit 0 of Port 3, General Purpose Input/Output**
	U2C0_DOUT	O1	St/B	**USIC2 Channel 0 Shift Data Output**
	BREQ	OH	St/B	**External Bus Request Output**
	ESR1_1	I	St/B	**ESR1 Trigger Input 1**
	U2C0_DX0A	I	St/B	**USIC2 Channel 0 Shift Data Input**
	RxDC3B	I	St/B	**CAN Node 3 Receive Data Input**
	U2C0_DX1A	I	St/B	**USIC2 Channel 0 Shift Clock Input**
86	P10.1	O0 / I	St/B	**Bit 1 of Port 10, General Purpose Input/Output**
	U0C0_DOUT	O1	St/B	**USIC0 Channel 0 Shift Data Output**
	CCU60_CC61	O2 / I	St/B	**CCU60 Channel 1 Input/Output**
	AD1	OH / I	St/B	**External Bus Interface Address/Data Line 1**
	U0C0_DX0B	I	St/B	**USIC0 Channel 0 Shift Data Input**
	U0C0_DX1A	I	St/B	**USIC0 Channel 0 Shift Clock Input**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
87	P0.3	O0 / I	St/B	**Bit 3 of Port 0, General Purpose Input/Output**
	U1C0_SELO 0	O1	St/B	**USIC1 Channel 0 Select/Control 0 Output**
	U1C1_SELO 1	O2	St/B	**USIC1 Channel 1 Select/Control 1 Output**
	CCU61_COU T60	O3	St/B	**CCU61 Channel 0 Output**
	A3	OH	St/B	**External Bus Interface Address Line 3**
	U1C0_DX2A	I	St/B	**USIC1 Channel 0 Shift Control Input**
	RxDC0B	I	St/B	**CAN Node 0 Receive Data Input**
88	P3.1	O0 / I	St/B	**Bit 1 of Port 3, General Purpose Input/Output**
	U2C0_DOUT	O1	St/B	**USIC2 Channel 0 Shift Data Output**
	TxDC3	O2	St/B	**CAN Node 3 Transmit Data Output**
	HLDA	OH / I	St/B	**External Bus Hold Acknowledge Output/Input** Output in master mode, input in slave mode.
	U2C0_DX0B	I	St/B	**USIC2 Channel 0 Shift Data Input**
89	P10.2	O0 / I	St/B	**Bit 2 of Port 10, General Purpose Input/Output**
	U0C0_SCLK OUT	O1	St/B	**USIC0 Channel 0 Shift Clock Output**
	CCU60_CC6 2	O2 / I	St/B	**CCU60 Channel 2 Input/Output**
	AD2	OH / I	St/B	**External Bus Interface Address/Data Line 2**
	U0C0_DX1B	I	St/B	**USIC0 Channel 0 Shift Clock Input**
90	P0.4	O0 / I	St/B	**Bit 4 of Port 0, General Purpose Input/Output**
	U1C1_SELO 0	O1	St/B	**USIC1 Channel 1 Select/Control 0 Output**
	U1C0_SELO 1	O2	St/B	**USIC1 Channel 0 Select/Control 1 Output**
	CCU61_COU T61	O3	St/B	**CCU61 Channel 1 Output**
	A4	OH	St/B	**External Bus Interface Address Line 4**
	U1C1_DX2A	I	St/B	**USIC1 Channel 1 Shift Control Input**
	RxDC1B	I	St/B	**CAN Node 1 Receive Data Input**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
92	TRef	IO	Sp/1	**Control Pin for Core Voltage Generation** Connect TRef to V_{DDPB} to use the on-chip EVRs. Connect TRef to V_{DDI1} for external core voltage supply (on-chip EVRs off).
93	P3.2	O0 / I	St/B	**Bit 2 of Port 3, General Purpose Input/Output**
	U2C0_SCLKOUT	O1	St/B	**USIC2 Channel 0 Shift Clock Output**
	TxDC3	O2	St/B	**CAN Node 3 Transmit Data Output**
	U2C0_DX1B	I	St/B	**USIC2 Channel 0 Shift Clock Input**
	HOLD	I	St/B	**External Bus Master Hold Request Input**
94	P2.10	O0 / I	St/B	**Bit 10 of Port 2, General Purpose Input/Output**
	U0C1_DOUT	O1	St/B	**USIC0 Channel 1 Shift Data Output**
	U0C0_SELO3	O2	St/B	**USIC0 Channel 0 Select/Control 3 Output**
	CC2_23	O3 / I	St/B	**CAPCOM2 CC23IO Capture Inp./ Compare Out.**
	A23	OH	St/B	**External Bus Interface Address Line 23**
	U0C1_DX0E	I	St/B	**USIC0 Channel 1 Shift Data Input**
	CAPIN	I	St/B	**GPT2 Register CAPREL Capture Input**
95	P10.3	O0 / I	St/B	**Bit 3 of Port 10, General Purpose Input/Output**
	CCU60_COUT60	O2	St/B	**CCU60 Channel 0 Output**
	AD3	OH / I	St/B	**External Bus Interface Address/Data Line 3**
	U0C0_DX2A	I	St/B	**USIC0 Channel 0 Shift Control Input**
	U0C1_DX2A	I	St/B	**USIC0 Channel 1 Shift Control Input**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
96	P0.5	O0 / I	St/B	**Bit 5 of Port 0, General Purpose Input/Output**
	U1C1_SCLKOUT	O1	St/B	**USIC1 Channel 1 Shift Clock Output**
	U1C0_SELO2	O2	St/B	**USIC1 Channel 0 Select/Control 2 Output**
	CCU61_COUT62	O3	St/B	**CCU61 Channel 2 Output**
	A5	OH	St/B	**External Bus Interface Address Line 5**
	U1C1_DX1A	I	St/B	**USIC1 Channel 1 Shift Clock Input**
	U1C0_DX1C	I	St/B	**USIC1 Channel 0 Shift Clock Input**
97	P3.3	O0 / I	St/B	**Bit 3 of Port 3, General Purpose Input/Output**
	U2C0_SELO0	O1	St/B	**USIC2 Channel 0 Select/Control 0 Output**
	U2C1_SELO1	O2	St/B	**USIC2 Channel 1 Select/Control 1 Output**
	U2C0_DX2A	I	St/B	**USIC2 Channel 0 Shift Control Input**
	RxDC3A	I	St/B	**CAN Node 3 Receive Data Input**
98	P10.4	O0 / I	St/B	**Bit 4 of Port 10, General Purpose Input/Output**
	U0C0_SELO3	O1	St/B	**USIC0 Channel 0 Select/Control 3 Output**
	CCU60_COUT61	O2	St/B	**CCU60 Channel 1 Output**
	AD4	OH / I	St/B	**External Bus Interface Address/Data Line 4**
	U0C0_DX2B	I	St/B	**USIC0 Channel 0 Shift Control Input**
	U0C1_DX2B	I	St/B	**USIC0 Channel 1 Shift Control Input**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
99	P3.4	O0 / I	St/B	**Bit 4 of Port 3, General Purpose Input/Output**
	U2C1_SELO 0	O1	St/B	**USIC2 Channel 1 Select/Control 0 Output**
	U2C0_SELO 1	O2	St/B	**USIC2 Channel 0 Select/Control 1 Output**
	U0C0_SELO 4	O3	St/B	**USIC0 Channel 0 Select/Control 4 Output**
	U2C1_DX2A	I	St/B	**USIC2 Channel 1 Shift Control Input**
	RxDC4A	I	St/B	**CAN Node 4 Receive Data Input**
100	P10.5	O0 / I	St/B	**Bit 5 of Port 10, General Purpose Input/Output**
	U0C1_SCLK OUT	O1	St/B	**USIC0 Channel 1 Shift Clock Output**
	CCU60_COU T62	O2	St/B	**CCU60 Channel 2 Output**
	AD5	OH / I	St/B	**External Bus Interface Address/Data Line 5**
	U0C1_DX1B	I	St/B	**USIC0 Channel 1 Shift Clock Input**
101	P3.5	O0 / I	St/B	**Bit 5 of Port 3, General Purpose Input/Output**
	U2C1_SCLK OUT	O1	St/B	**USIC2 Channel 1 Shift Clock Output**
	U2C0_SELO 2	O2	St/B	**USIC2 Channel 0 Select/Control 2 Output**
	U0C0_SELO 5	O3	St/B	**USIC0 Channel 0 Select/Control 5 Output**
	U2C1_DX1A	I	St/B	**USIC2 Channel 1 Shift Clock Input**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
102	P0.6	O0 / I	St/B	**Bit 6 of Port 0, General Purpose Input/Output**
	U1C1_DOUT	O1	St/B	**USIC1 Channel 1 Shift Data Output**
	TxDC1	O2	St/B	**CAN Node 1 Transmit Data Output**
	CCU61_COUT63	O3	St/B	**CCU61 Channel 3 Output**
	A6	OH	St/B	**External Bus Interface Address Line 6**
	U1C1_DX0A	I	St/B	**USIC1 Channel 1 Shift Data Input**
	CCU61_CTRAPA	I	St/B	**CCU61 Emergency Trap Input**
	U1C1_DX1B	I	St/B	**USIC1 Channel 1 Shift Clock Input**
103	P10.6	O0 / I	St/B	**Bit 6 of Port 10, General Purpose Input/Output**
	U0C0_DOUT	O1	St/B	**USIC0 Channel 0 Shift Data Output**
	TxDC4	O2	St/B	**CAN Node 4 Transmit Data Output**
	U1C0_SELO0	O3	St/B	**USIC1 Channel 0 Select/Control 0 Output**
	AD6	OH / I	St/B	**External Bus Interface Address/Data Line 6**
	U0C0_DX0C	I	St/B	**USIC0 Channel 0 Shift Data Input**
	U1C0_DX2D	I	St/B	**USIC1 Channel 0 Shift Control Input**
	CCU60_CTRAPA	I	St/B	**CCU60 Emergency Trap Input**
104	P3.6	O0 / I	St/B	**Bit 6 of Port 3, General Purpose Input/Output**
	U2C1_DOUT	O1	St/B	**USIC2 Channel 1 Shift Data Output**
	TxDC4	O2	St/B	**CAN Node 4 Transmit Data Output**
	U0C0_SELO6	O3	St/B	**USIC0 Channel 0 Select/Control 6 Output**
	U2C1_DX0A	I	St/B	**USIC2 Channel 1 Shift Data Input**
	U2C1_DX1B	I	St/B	**USIC2 Channel 1 Shift Clock Input**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
105	P10.7	O0 / I	St/B	**Bit 7 of Port 10, General Purpose Input/Output**
	U0C1_DOUT	O1	St/B	**USIC0 Channel 1 Shift Data Output**
	CCU60_COUT63	O2	St/B	**CCU60 Channel 3 Output**
	AD7	OH / I	St/B	**External Bus Interface Address/Data Line 7**
	U0C1_DX0B	I	St/B	**USIC0 Channel 1 Shift Data Input**
	CCU60_CCPOS0A	I	St/B	**CCU60 Position Input 0**
	RxDC4C	I	St/B	**CAN Node 4 Receive Data Input**
106	P0.7	O0 / I	St/B	**Bit 7 of Port 0, General Purpose Input/Output**
	U1C1_DOUT	O1	St/B	**USIC1 Channel 1 Shift Data Output**
	U1C0_SELO3	O2	St/B	**USIC1 Channel 0 Select/Control 3 Output**
	A7	OH	St/B	**External Bus Interface Address Line 7**
	U1C1_DX0B	I	St/B	**USIC1 Channel 1 Shift Data Input**
	CCU61_CTRAPB	I	St/B	**CCU61 Emergency Trap Input**
107	P3.7	O0 / I	St/B	**Bit 7 of Port 3, General Purpose Input/Output**
	U2C1_DOUT	O1	St/B	**USIC2 Channel 1 Shift Data Output**
	U2C0_SELO3	O2	St/B	**USIC2 Channel 0 Select/Control 3 Output**
	U0C0_SELO7	O3	St/B	**USIC0 Channel 0 Select/Control 7 Output**
	U2C1_DX0B	I	St/B	**USIC2 Channel 1 Shift Data Input**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
111	P1.0	O0 / I	St/B	**Bit 0 of Port 1, General Purpose Input/Output**
	U1C0_MCLKOUT	O1	St/B	**USIC1 Channel 0 Master Clock Output**
	U1C0_SELO4	O2	St/B	**USIC1 Channel 0 Select/Control 4 Output**
	A8	OH	St/B	**External Bus Interface Address Line 8**
	ESR1_3	I	St/B	**ESR1 Trigger Input 3**
	EX0BINA	I	St/B	**External Interrupt Trigger Input**
	CCU62_CTRAPB	I	St/B	**CCU62 Emergency Trap Input**
112	P9.0	O0 / I	St/B	**Bit 0 of Port 9, General Purpose Input/Output**
	CCU63_CC60	O1 / I	St/B	**CCU63 Channel 0 Input/Output**
113	P10.8	O0 / I	St/B	**Bit 8 of Port 10, General Purpose Input/Output**
	U0C0_MCLKOUT	O1	St/B	**USIC0 Channel 0 Master Clock Output**
	U0C1_SELO0	O2	St/B	**USIC0 Channel 1 Select/Control 0 Output**
	AD8	OH / I	St/B	**External Bus Interface Address/Data Line 8**
	CCU60_CCPOS1A	I	St/B	**CCU60 Position Input 1**
	U0C0_DX1C	I	St/B	**USIC0 Channel 0 Shift Clock Input**
	$\overline{\text{BRKIN\_B}}$	I	St/B	**OCDS Break Signal Input**
114	P9.1	O0 / I	St/B	**Bit 1 of Port 9, General Purpose Input/Output**
	CCU63_CC61	O1 / I	St/B	**CCU63 Channel 1 Input/Output**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
115	P10.9	O0 / I	St/B	**Bit 9 of Port 10, General Purpose Input/Output**
	U0C0_SELO 4	O1	St/B	**USIC0 Channel 0 Select/Control 4 Output**
	U0C1_MCLK OUT	O2	St/B	**USIC0 Channel 1 Master Clock Output**
	AD9	OH / I	St/B	**External Bus Interface Address/Data Line 9**
	CCU60_CCP OS2A	I	St/B	**CCU60 Position Input 2**
	TCK_B	I	St/B	**JTAG Clock Input**
116	P1.1	O0 / I	St/B	**Bit 1 of Port 1, General Purpose Input/Output**
	CCU62_COU T62	O1	St/B	**CCU62 Channel 2 Output**
	U1C0_SELO 5	O2	St/B	**USIC1 Channel 0 Select/Control 5 Output**
	U2C1_DOUT	O3	St/B	**USIC2 Channel 1 Shift Data Output**
	A9	OH	St/B	**External Bus Interface Address Line 9**
	ESR2_3	I	St/B	**ESR2 Trigger Input 3**
	EX1BINA	I	St/B	**External Interrupt Trigger Input**
	U2C1_DX0C	I	St/B	**USIC2 Channel 1 Shift Data Input**
117	P10.10	O0 / I	St/B	**Bit 10 of Port 10, General Purpose Input/Output**
	U0C0_SELO 0	O1	St/B	**USIC0 Channel 0 Select/Control 0 Output**
	CCU60_COU T63	O2	St/B	**CCU60 Channel 3 Output**
	AD10	OH / I	St/B	**External Bus Interface Address/Data Line 10**
	U0C0_DX2C	I	St/B	**USIC0 Channel 0 Shift Control Input**
	TDI_B	I	St/B	**JTAG Test Data Input**
	U0C1_DX1A	I	St/B	**USIC0 Channel 1 Shift Clock Input**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
118	P10.11	O0 / I	St/B	**Bit 11 of Port 10, General Purpose Input/Output**
	U1C0_SCLKOUT	O1	St/B	**USIC1 Channel 0 Shift Clock Output**
	$\overline{\text{BRKOUT}}$	O2	St/B	**OCDS Break Signal Output**
	AD11	OH / I	St/B	**External Bus Interface Address/Data Line 11**
	U1C0_DX1D	I	St/B	**USIC1 Channel 0 Shift Clock Input**
	RxDC2B	I	St/B	**CAN Node 2 Receive Data Input**
	TMS_B	I	St/B	**JTAG Test Mode Selection Input**
119	P9.2	O0 / I	St/B	**Bit 2 of Port 9, General Purpose Input/Output**
	CCU63_CC62	O1 / I	St/B	**CCU63 Channel 2 Input/Output**
120	P1.2	O0 / I	St/B	**Bit 2 of Port 1, General Purpose Input/Output**
	CCU62_CC62	O1 / I	St/B	**CCU62 Channel 2 Input/Output**
	U1C0_SELO6	O2	St/B	**USIC1 Channel 0 Select/Control 6 Output**
	U2C1_SCLKOUT	O3	St/B	**USIC2 Channel 1 Shift Clock Output**
	A10	OH	St/B	**External Bus Interface Address Line 10**
	ESR1_4	I	St/B	**ESR1 Trigger Input 4**
	CCU61_T12HRB	I	St/B	**External Run Control Input for T12 of CCU61**
	EX2AINA	I	St/B	**External Interrupt Trigger Input**
	U2C1_DX0D	I	St/B	**USIC2 Channel 1 Shift Data Input**
	U2C1_DX1C	I	St/B	**USIC2 Channel 1 Shift Clock Input**
121	P10.12	O0 / I	St/B	**Bit 12 of Port 10, General Purpose Input/Output**
	U1C0_DOUT	O1	St/B	**USIC1 Channel 0 Shift Data Output**
	TxDC2	O2	St/B	**CAN Node 2 Transmit Data Output**
	TDO	O3	St/B	**JTAG Test Data Output**
	AD12	OH / I	St/B	**External Bus Interface Address/Data Line 12**
	U1C0_DX0C	I	St/B	**USIC1 Channel 0 Shift Data Input**
	U1C0_DX1E	I	St/B	**USIC1 Channel 0 Shift Clock Input**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
122	P9.3	O0 / I	St/B	**Bit 3 of Port 9, General Purpose Input/Output**
	CCU63_COUT60	O1	St/B	**CCU63 Channel 0 Output**
	$\overline{\text{BRKOUT}}$	O2	St/B	**OCDS Break Signal Output**
123	P10.13	O0 / I	St/B	**Bit 13 of Port 10, General Purpose Input/Output**
	U1C0_DOUT	O1	St/B	**USIC1 Channel 0 Shift Data Output**
	TxDC3	O2	St/B	**CAN Node 3 Transmit Data Output**
	U1C0_SELO3	O3	St/B	**USIC1 Channel 0 Select/Control 3 Output**
	$\overline{\text{WR}}$/$\overline{\text{WRL}}$	OH	St/B	**External Bus Interface Write Strobe Output** Active for each external write access, when $\overline{\text{WR}}$, active for ext. writes to the low byte, when $\overline{\text{WRL}}$.
	U1C0_DX0D	I	St/B	**USIC1 Channel 0 Shift Data Input**
124	P1.3	O0 / I	St/B	**Bit 3 of Port 1, General Purpose Input/Output**
	CCU62_COUT63	O1	St/B	**CCU62 Channel 3 Output**
	U1C0_SELO7	O2	St/B	**USIC1 Channel 0 Select/Control 7 Output**
	U2C0_SELO4	O3	St/B	**USIC2 Channel 0 Select/Control 4 Output**
	A11	OH	St/B	**External Bus Interface Address Line 11**
	ESR2_4	I	St/B	**ESR2 Trigger Input 4**
	CCU62_T12HRB	I	St/B	**External Run Control Input for T12 of CCU62**
	EX3AINA	I	St/B	**External Interrupt Trigger Input**
125	P9.4	O0 / I	St/B	**Bit 4 of Port 9, General Purpose Input/Output**
	CCU63_COUT61	O1	St/B	**CCU63 Channel 1 Output**
	U2C0_DOUT	O2	St/B	**USIC2 Channel 0 Shift Data Output**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
126	P9.5	O0 / I	St/B	**Bit 5 of Port 9, General Purpose Input/Output**
	CCU63_COUT62	O1	St/B	**CCU63 Channel 2 Output**
	U2C0_DOUT	O2	St/B	**USIC2 Channel 0 Shift Data Output**
	U2C0_DX0E	I	St/B	**USIC2 Channel 0 Shift Data Input**
	CCU60_CCPOS2B	I	St/B	**CCU60 Position Input 2**
128	P10.14	O0 / I	St/B	**Bit 14 of Port 10, General Purpose Input/Output**
	U1C0_SELO1	O1	St/B	**USIC1 Channel 0 Select/Control 1 Output**
	U0C1_DOUT	O2	St/B	**USIC0 Channel 1 Shift Data Output**
	$\overline{RD}$	OH	St/B	**External Bus Interface Read Strobe Output**
	ESR2_2	I	St/B	**ESR2 Trigger Input 2**
	U0C1_DX0C	I	St/B	**USIC0 Channel 1 Shift Data Input**
	RxDC3C	I	St/B	**CAN Node 3 Receive Data Input**
129	P1.4	O0 / I	St/B	**Bit 4 of Port 1, General Purpose Input/Output**
	CCU62_COUT61	O1	St/B	**CCU62 Channel 1 Output**
	U1C1_SELO4	O2	St/B	**USIC1 Channel 1 Select/Control 4 Output**
	U2C0_SELO5	O3	St/B	**USIC2 Channel 0 Select/Control 5 Output**
	A12	OH	St/B	**External Bus Interface Address Line 12**
	U2C0_DX2B	I	St/B	**USIC2 Channel 0 Shift Control Input**
130	P10.15	O0 / I	St/B	**Bit 15 of Port 10, General Purpose Input/Output**
	U1C0_SELO2	O1	St/B	**USIC1 Channel 0 Select/Control 2 Output**
	U0C1_DOUT	O2	St/B	**USIC0 Channel 1 Shift Data Output**
	U1C0_DOUT	O3	St/B	**USIC1 Channel 0 Shift Data Output**
	ALE	OH	St/B	**External Bus Interf. Addr. Latch Enable Output**
	U0C1_DX1C	I	St/B	**USIC0 Channel 1 Shift Clock Input**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
131	P1.5	O0 / I	St/B	**Bit 5 of Port 1, General Purpose Input/Output**
	CCU62_COUT60	O1	St/B	**CCU62 Channel 0 Output**
	U1C1_SELO3	O2	St/B	**USIC1 Channel 1 Select/Control 3 Output**
	$\overline{\text{BRKOUT}}$	O3	St/B	**OCDS Break Signal Output**
	A13	OH	St/B	**External Bus Interface Address Line 13**
	U2C0_DX0C	I	St/B	**USIC2 Channel 0 Shift Data Input**
132	P9.6	O0 / I	St/B	**Bit 6 of Port 9, General Purpose Input/Output**
	CCU63_COUT63	O1	St/B	**CCU63 Channel 3 Output**
	CCU63_COUT62	O2	St/B	**CCU63 Channel 2 Output**
	CCU63_CTRAPA	I	St/B	**CCU63 Emergency Trap Input**
	CCU60_CCPOS1B	I	St/B	**CCU60 Position Input 1**
133	P1.6	O0 / I	St/B	**Bit 6 of Port 1, General Purpose Input/Output**
	CCU62_CC61	O1 / I	St/B	**CCU62 Channel 1 Input/Output**
	U1C1_SELO2	O2	St/B	**USIC1 Channel 1 Select/Control 2 Output**
	U2C0_DOUT	O3	St/B	**USIC2 Channel 0 Shift Data Output**
	A14	OH	St/B	**External Bus Interface Address Line 14**
	U2C0_DX0D	I	St/B	**USIC2 Channel 0 Shift Data Input**
134	P9.7	O0 / I	St/B	**Bit 7 of Port 9, General Purpose Input/Output**
	CCU63_CTRAPB	I	St/B	**CCU63 Emergency Trap Input**
	U2C0_DX1D	I	St/B	**USIC2 Channel 0 Shift Clock Input**
	CCU60_CCPOS0B	I	St/B	**CCU60 Position Input 0**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
135	P1.7	O0 / I	St/B	**Bit 7 of Port 1, General Purpose Input/Output**
	CCU62_CC60	O1 / I	St/B	**CCU62 Channel 0 Input/Output**
	U1C1_MCLKOUT	O2	St/B	**USIC1 Channel 1 Master Clock Output**
	U2C0_SCLKOUT	O3	St/B	**USIC2 Channel 0 Shift Clock Output**
	A15	OH	St/B	**External Bus Interface Address Line 15**
	U2C0_DX1C	I	St/B	**USIC2 Channel 0 Shift Clock Input**
136	XTAL2	O	Sp/1	**Crystal Oscillator Amplifier Output**
137	XTAL1	I	Sp/1	**Crystal Oscillator Amplifier Input** To clock the device from an external source, drive XTAL1, while leaving XTAL2 unconnected. Voltages on XTAL1 must comply to the core supply voltage V_{DDI1}.
138	$\overline{\text{PORST}}$	I	In/B	**Power On Reset Input** A low level at this pin resets the XE16x completely. A spike filter suppresses input pulses <10 ns. Input pulses >100 ns safely pass the filter. The minimum duration for a safe recognition should be 120 ns. An internal pullup device will hold this pin high when nothing is driving it.
139	$\overline{\text{ESR1}}$	O0 / I	St/B	**External Service Request 1**
	U1C0_DX0F	I	St/B	**USIC1 Channel 0 Shift Data Input**
	U1C0_DX2C	I	St/B	**USIC1 Channel 0 Shift Control Input**
	U1C1_DX0C	I	St/B	**USIC1 Channel 1 Shift Data Input**
	U1C1_DX2B	I	St/B	**USIC1 Channel 1 Shift Control Input**
	U2C1_DX2C	I	St/B	**USIC2 Channel 1 Shift Control Input**
	EX0AINB	I	St/B	**External Interrupt Trigger Input**

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
140	$\overline{\text{ESR2}}$	O0 / I	St/B	**External Service Request 2**
	U1C1_DX0D	I	St/B	**USIC1 Channel 1 Shift Data Input**
	U1C1_DX2C	I	St/B	**USIC1 Channel 1 Shift Control Input**
	U2C1_DX0E	I	St/B	**USIC1 Channel 1 Shift Data Input**
	U2C1_DX2B	I	St/B	**USIC2 Channel 1 Shift Control Input**
	EX1AINB	I	St/B	**External Interrupt Trigger Input**
141	$\overline{\text{ESR0}}$	O0 / I	St/B	**External Service Request 0** *Note: After power-up, ESR0 operates as open-drain bidirectional reset with a weak pull-up.*
	U1C0_DX0E	I	St/B	**USIC1 Channel 0 Shift Data Input**
	U1C0_DX2B	I	St/B	**USIC1 Channel 0 Shift Control Input**
142	P8.6	O0 / I	St/B	**Bit 6 of Port 8, General Purpose Input/Output**
	CCU60_COUT63	O1	St/B	**CCU60 Channel 3 Output**
	CCU60_CTRAPB	I	St/B	**CCU60 Emergency Trap Input**
	$\overline{\text{BRKIN\_D}}$	I	St/B	**OCDS Break Signal Input**
143	P8.5	O0 / I	St/B	**Bit 5 of Port 8, General Purpose Input/Output**
	CCU60_COUT62	O1	St/B	**CCU60 Channel 2 Output**
	TCK_D	I	St/B	**JTAG Clock Input**
15	V_{DDIM}	-	PS/M	**Digital Core Supply Voltage for Domain M** Decouple with a ceramic capacitor, see Data Sheet for details.
54, 91, 127	V_{DDI1}	-	PS/1	**Digital Core Supply Voltage for Domain 1** Decouple with a ceramic capacitor, see Data Sheet for details. All V_{DDI1} pins must be connected to each other.
20	V_{DDPA}	-	PS/A	**Digital Pad Supply Voltage for Domain A** Connect decoupling capacitors to adjacent V_{DDP}/ V_{SS} pin pairs as close as possible to the pins. *Note: The A/D_Converters and ports P5, P6, and P15 are fed from supply voltage V_{DDPA}.*

Table 7-18 Pin Definitions and Functions (cont'd)

Pin	Symbol	Ctrl.	Type	Function
2, 36, 38, 72, 74, 108, 110, 144	V_{DDPB}	-	PS/B	**Digital Pad Supply Voltage for Domain B** Connect decoupling capacitors to adjacent V_{DDP}/ V_{SS} pin pairs as close as possible to the pins. *Note: The on-chip voltage regulators and all ports except P5, P6, and P15 are fed from supply voltage V_{DDPB}.*
1, 37, 73, 109	V_{SS}	-	PS/--	**Digital Ground** All V_{SS} pins must be connected to the ground-line or ground-plane.

1) To generate the reference clock output for bus timing measurement, f_{SYS} must be selected as source for EXTCLK and P2.8 must be selected as output pin. Also the high-speed clock pad must be enabled. This configuration is referred to as reference clock output signal CLKOUT.

8 Dedicated Pins

Most of the input/output or control signals of the functional the XE16x are realized as alternate functions of pins of the parallel ports. There is, however, a number of signals that use separate pins, including the oscillator, special control signals and, of course, the power supply.

Table 8-1 summarizes the dedicated pins of the XE16x.

Table 8-1 XE16x Dedicated Pins

Pin(s)	Function
$\overline{\text{PORST}}$	Power-On Reset Input
$\overline{\text{ESR0}}$	External Service Request Input 0
$\overline{\text{ESR1}}$	External Service Request Input 1
$\overline{\text{ESR2}}$	External Service Request Input 2
XTAL1, XTAL2	Oscillator Input/Output (main oscillator)
$\overline{\text{TESTM}}$	Test Mode Enable
$\overline{\text{TRST}}$	Test-System Reset Input
TRef	Control Pin for Core Voltage Generation
V_{AREFx}, V_{AGND}	Power Supply for the Analog/Digital Converter(s)
V_{DDIM}	Digital Core Supply for Domain M (1 pin)
V_{DDI1}	Digital Core Supply for Domain 1 (3 pins)
V_{DDPA}	Digital Pad Supply for Domain A (1 pin)
V_{DDPB}	Digital Pad Supply for Domain B (8 pins)
V_{SS}	Digital Ground (4 pins)

The Power-On Reset Input $\overline{\text{PORST}}$ allows to put the XE16x into the well defined reset condition either at power-up or external events like a hardware failure or manual reset.

The External Service Request Inputs $\overline{\text{ESR0}}$, **$\overline{\text{ESR1}}$**, and **$\overline{\text{ESR2}}$** can be used for several system-related functions:

- trigger interrupt or trap (Class A or Class B) requests via an external signal (e.g. a power-fail signal)
- generate wake-up request signals
- generate hardware reset requests (**$\overline{\text{ESR0}}$** is bidirectional by default, **$\overline{\text{ESR1}}$** and **$\overline{\text{ESR2}}$** can optionally output a reset signal)
- data/control input for CCU6x, MultiCAN, and USIC (**$\overline{\text{ESR1}}$** or **$\overline{\text{ESR2}}$**)
- software-controlled input/output signal

The Oscillator Input XTAL1 and Output XTAL2 connect the internal **Main Oscillator** to the external crystal. The oscillator provides an inverter and a feedback element. The standard external oscillator circuitry comprises the crystal, two low end capacitors and series resistor to limit the current through the crystal. The main oscillator is intended for the generation of a high-precision operating clock signal for the XE16x.

An external clock signal may be fed to the input XTAL1, leaving XTAL2 open. The current logic state of input XTAL1 can be read via a status flag, so XTAL1 can be used as digital input if neither the oscillator interface nor the clock input is required.

Note: Pin XTAL1 belongs to the core power domain DMP_M. All input signals, therefore, must be within the core voltage range.

The Test Mode Input $\overline{\text{TESTM}}$ puts the XE16x into a test mode, which is used during the production tests of the device. In test mode, the XE16x behaves different from normal operation. Therefore, pin $\overline{\text{TESTM}}$ must be held HIGH (connect to V_{DDPB}) for normal operation in an application system.

The Test Reset Input $\overline{\text{TRST}}$ puts the XE16x's debug system into reset state. During normal operation this input should be held low. For debugging purposes the on-chip debugging system can be enabled by driving pin $\overline{\text{TRST}}$ high at the rising edge of $\overline{\text{PORST}}$.

The Control Pin for Core Voltage Generation TRef was used to control the generation method for the core supply voltage V_{DDI} in step AA. For that step, pin TRef must be connected to V_{DDPB} (use the on-chip EVRs).

This connection is no more required from step AB on. For the current step, pin TRef is logically not connected.

Future derivatives will feature an additional general purpose IO pin at this position.

The Analog Reference Voltage Supply pins V_{AREFx} and V_{AGND} provide separate reference voltage for the on-chip Analog/Digital-Converter(s). This reduces the noise that is coupled to the analog input signals from the digital logic sections and so improves the stability of the conversion results, when V_{AREF} and V_{AGND} are properly discoupled from V_{DD} and V_{SS}. Also, because conversion results are generated in relation to the reference voltages, ratiometric conversions are easily achieved.

Note: Channel 0 of each module can be used as an alternate reference voltage input.

The Core Supply pins V_{DDIM}/V_{DDI1} serve two purposes: While the on-chip EVVRs provide the power for the core logic of the XE16x these pins connect the EVVRs to their external buffer capacitors. For external supply, the core voltage is applied to these pins. The respective V_{DDI}/V_{SS} pairs should be decoupled as close to the pins as possible. Use ceramic capacitors and observe their values recommended in the respective Data Sheet.

The Power Supply pins V_{DDPA}/V_{DDPB} provide the power supply for all the analog and digital logic of the XE16x. Each power domain (DMP_A and DMP_B) can be supplied with an arbitrary voltage within the specified supply voltage range (please refer to the corresponding Data Sheets). These pins supply the output drivers as well as the on-chip EVVRs (V_{DDPB}), except for external core voltage supply. The respective V_{DDP}/V_{SS} pairs should be decoupled as close to the pins as possible.

The Ground Reference pins V_{SS} provide the ground reference voltage for the power supplies as well as the reference voltage for the input signals.

Note: All V_{DDx} pins and all V_{SS} pins must be connected to the power supplies and ground, respectively.

9 The External Bus Controller EBC

All external memory accesses are performed by a particular on-chip External Bus Controller (EBC). It can be programmed either to Single Chip Mode when no external memory is required at all, or dynamically (depending on the selected address range, belonging to a chip-select signal) to one of four different external memory access modes, which are as follows:

- 1 … 24-bit Addresses, 16-bit Data, Demultiplexed
- 1 … 24-bit Addresses, 16-bit Data, Multiplexed
- 1 … 24-bit Addresses, 8-bit Data, Multiplexed
- 1 … 24-bit Addresses, 8-bit Data, Demultiplexed

*Note: The following description refers to the general EBC feature set. In the 100-pin package, some features are not available, see **Table 9-1**. In the 64-pin package, no external bus interface is available.*

In the multiplexed bus modes, the intra-segment address outputs (A15 … A0) and data input/outputs are overlaid on 16 port pins. The higher segment address outputs (A23 … A16) are mapped to separate port pins. In the demultiplexed bus modes, address outputs and data input/outputs are not overlaid but mapped to the port pins separately. For applications which do not use all address lines for external devices, the external address space can be restricted by enabling only the required address lines. Up to 5 external $\overline{CS}$ signals can be generated in order to save external glue logic. Memories or peripherals with variable access time are supported via a particular ‘Ready’ function. A $\overline{HOLD}$/$\overline{HLDA}$ protocol is available for bus arbitration.

The XE16x External Bus Controller (EBC) allows access to external peripherals/memories and to internal LXBus modules. The LXBus is an internal representation of the ExtBus and it controls accesses to integrated peripherals and modules in the same way as accesses to external components.

The function of the EBC is controlled via a set of configuration registers. The basic and general behaviour is programmed via the mode-selection registers EBCMOD0 and EBCMOD1.

Similar to the supported external bus chip-select channels, LXBus modules are selected by a specific chip select signal (both access types are handled as ‘external’ accesses by the EBC).

The Function CONtrol register for $\overline{CSx}$ (FCONCSx) register specifies the external bus/LXBus cycles in terms of address (multiplexed/demultiplexed), data (16-bit/8-bit), READY control, and chip-select enable. The timing of the bus access is controlled by the Timing CONfiguration registers for $\overline{CSx}$ (TCONCSx), which specify the timing of the bus cycle with the lengths of the different access phases. All these parameters are used for accesses within a specific address area that is defined via the corresponding ADDRess SELect register ADDRSELx.

The five register sets (FCONCSx/TCONCSx/ADDRSELx, x = 7, 4, 3, 2, 1) define five independent “address windows”, whereas all external accesses outside these windows are controlled via registers FCONCS0 and TCONCS0. Chip Select signals $\overline{CS0}$ … $\overline{CS4}$ and their associated programmable address windows belong to access to resources on the external bus. Chip Select signal $\overline{CS7}$ and its associated fixed address window is used for access to the internal MultiCAN and USIC modules on the LXBus.

The external bus timing is related to the reference CLocK OUTput (CLKOUT[1)]). All bus signals are generated in relation to the rising edge of this clock. The external bus protocol is compatible with those of the C166 Family and the XC166 Family.

These improvements are configured via an enhanced register set (see above) in comparison to C166 Family. The C16x registers SYSCON and BUSCONx are no longer used. But because the configuration of the external bus controller is done during the application initialization, only some initialization code has to be adapted for using the new EBC module instead of the C16x external bus controller.

1) The term CLKOUT refers to the reference clock output signal which is generated by selecting f_{SYS} as source signal for the clock output signal EXTCLK on pin P2.8 and by enabling the high-speed clock driver on this pin.

9.1 External Bus Signals

The External Bus uses the following I/O signals[1]:

Table 9-1 EBC Bus Signals

Signal	I/O	Port Pins	Description
Signals available both in the 100-pin and 144-pin package			
ALE	O	P10	Address Latch Enable; active high
$\overline{RD}$	O		ReaD strobe: activated for every read access (active low)
$\overline{WR}$, $\overline{WRL}$	O		$\overline{WR}$ite/WRite Low byte strobe (active low) $\overline{WR}$-mode: activated for every write access. $\overline{WRL}$-mode: activated for low byte write accesses on a 16-bit bus and for every data write access on an 8-bit bus.
$\overline{BHE}$, $\overline{WRH}$	O	P2	$\overline{Byte}$ High Enable/WRite High byte strobe (active low) $\overline{BHE}$-mode: activated for every data access to the upper $\overline{byte}$ of the 16-bit bus (handled as additional address bit) $\overline{WRH}$-mode: activated for high byte write accesses on a 16-bit bus.
$\overline{READY}$/ READY	I	P2	READY; used for dynamic wait state insertion; programmable active high or low
AD[12..0] **AD[15..13]**	I/O	P10 P2	Address/Data bus; in multiplexed mode this bus is used for both address and data, in demultiplexed mode it is data bus only
A[7..0] **A[15..8]** **A[23..16]**	O	P0 P1 P2	Address bus
$\overline{CS}$[3..0]	O	P4	$\overline{Chip}$ Select; active low; $\overline{CS}$7-used for internal LXBus access to MultiCAN and USICs
Signals available additionally in the 144-pin package			
$\overline{BREQ}$	O	P3	Bus REQuest; active low
$\overline{HLDA}$	I/O		HoLD Accepted output (by the master); active low HoLD Accepted input (at the slave); active low
$\overline{HOLD}$	I		HOLD request; active low
CS4	O	P6	$\overline{Chip}$ Select; active low; $\overline{CS}$7-used for internal LXBus access to MultiCAN and USICs

1) In the 64-pin package the External Bus is not available.

Table 9-2 Write Configurations (see Chapter 9.3.2)

Written Byte		General Write Configuration			Separated Byte Low/High Writes		
Low	**High**	**$\overline{WR}$**	**$\overline{BHE}$**	**ADDR[0]**	**$\overline{WRL}$**	**$\overline{WRH}$**	**ADDR[0]**
–	–	inactive	don't care	0/1	inactive	inactive	0/1
write	–	active	inactive	0	active	inactive	0/1
–	write	active	active	1	inactive	active	0/1
write	write	active	active	0	active	active	0/1

9.2 Timing Principles

The external bus timing is subdivided into six different timing phases (A-F).

9.2.1 Basic Bus Cycle Protocols

The phases A-F define the flow of all control signals needed for any access sequence to external devices. At the beginning of a phase, the output signals may change within a given output delay time. After the output delay time, the values of the control output signals are stable within this phase. The output delay times are specified in the AC characteristics (see Data Sheet). Each phase can occupy a programmable number of clock cycles. The number of clock cycles is programmed in the TCONCSx register selected via the related address range and $\overline{CSx}$.

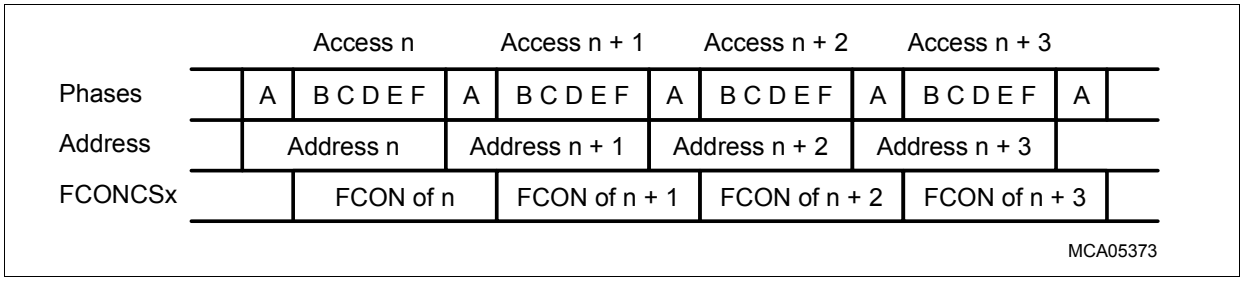


Figure 9-1 Phases of a Sequence of Several Accesses

Phase A is used for tristating databus drivers from the previous cycle (tristate wait states after $\overline{CS}$ switch). Phase A cycles are not inserted at every access cycle but only when changing the $\overline{CS}$. If an access using one $\overline{CS}$ ($\overline{CSx}$) was finished and the next access with a different $\overline{CS}$ ($\overline{CSy}$) is started then Phase A cycle(s) are performed according to the control bits as set in the **first** $\overline{CS}$ ($\overline{CSx}$).

The A Phase cycles are inserted while the addresses and ALE of the next cycle are already applied.

The following diagrams show the 6 timing phases for read and write accesses on the demultiplexed bus and the multiplexed bus.

9.2.1.1 Demultiplexed Bus

During demultiplexed access, the address and data signals exists on the bus in parallel.

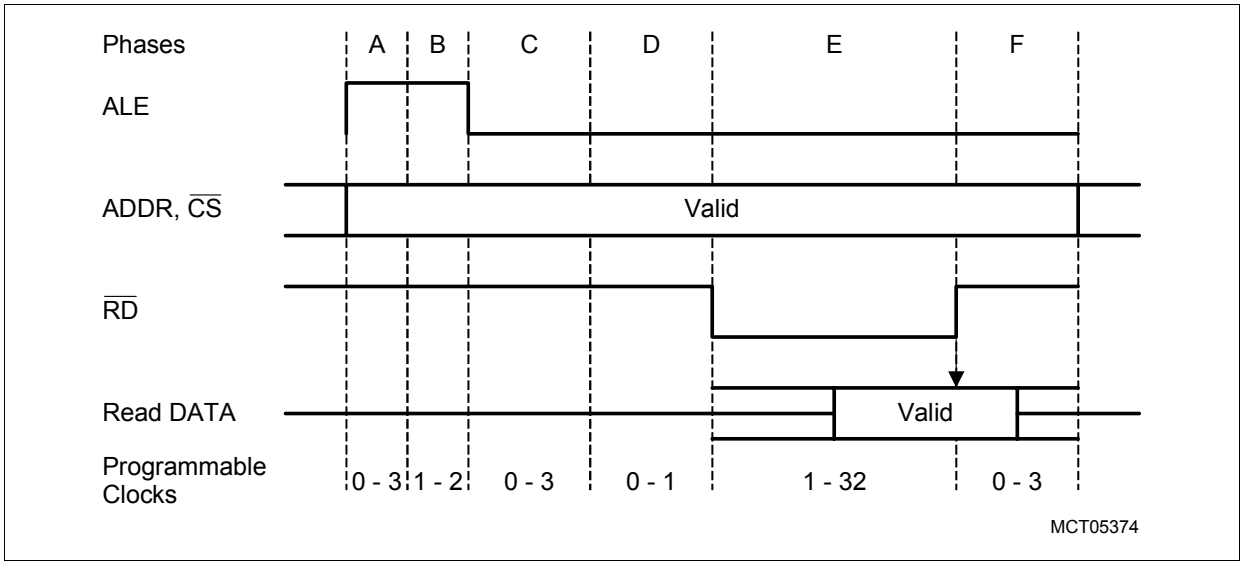


Figure 9-2 Demultiplexed Bus Read

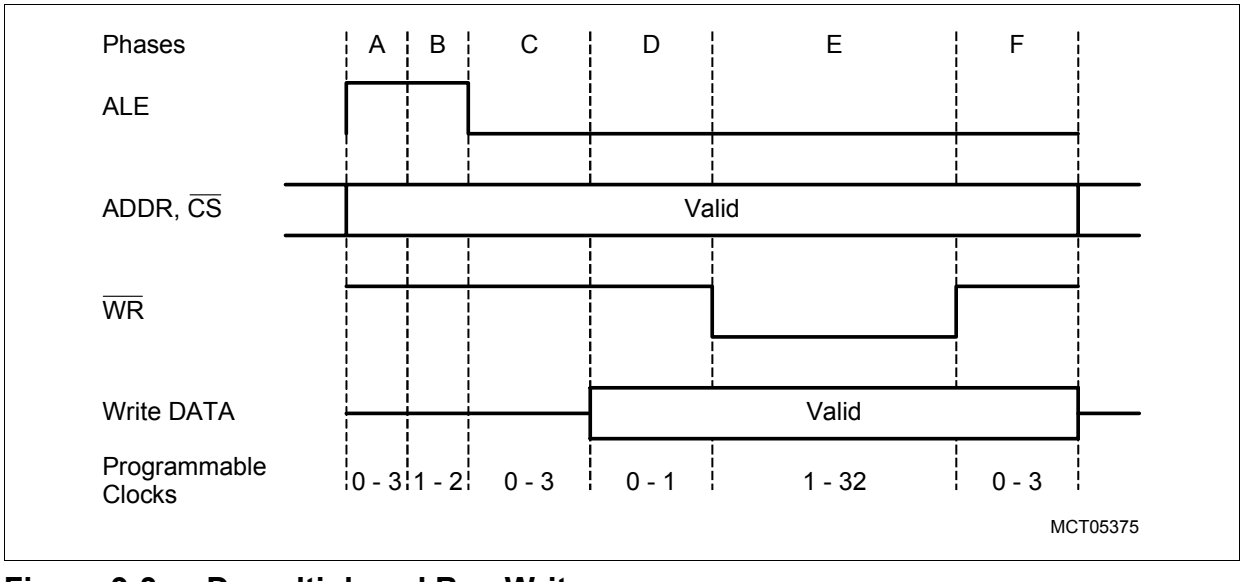


Figure 9-3 Demultiplexed Bus Write

- A phase: Addresses valid, ALE high, no command. $\overline{CS}$ switch tristate wait states
- B phase: Addresses valid, ALE high, no command. ALE length
- C phase: Addresses valid, ALE low, no command. R/W delay
- D phase: Write data valid, ALE low, no command. Data valid for write cycles
- E phase: Command (read or write) active. Access time
- F phase: Command inactive, address hold. Read data tristate time, write data hold time

9.2.1.2 Multiplexed Bus

During time multiplexed access, the address and data signals share the same external lines.

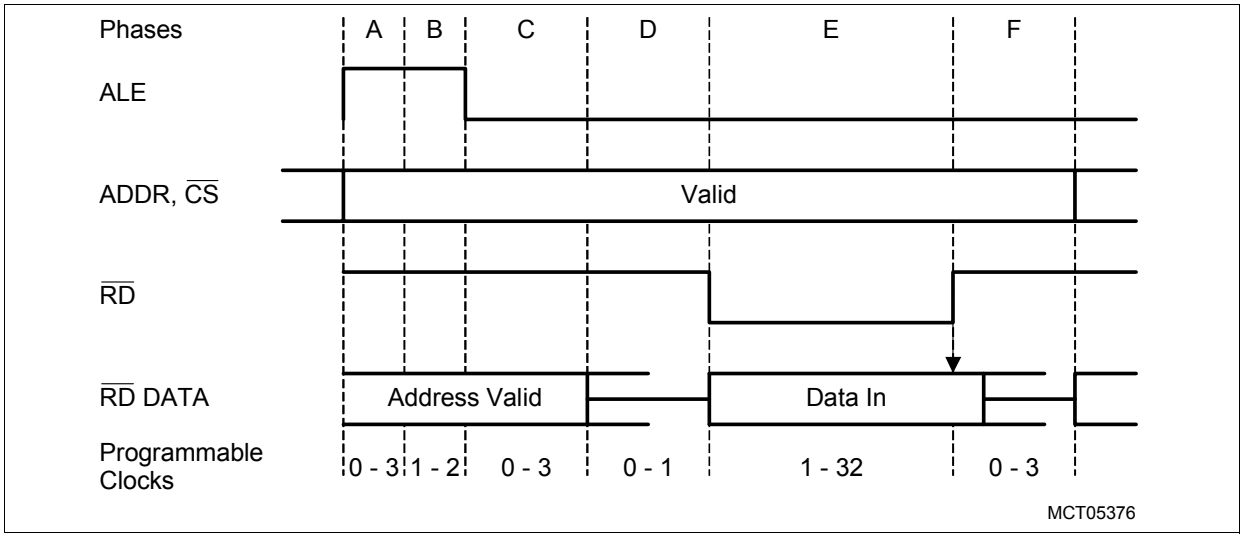


Figure 9-4 Multiplexed Bus Read

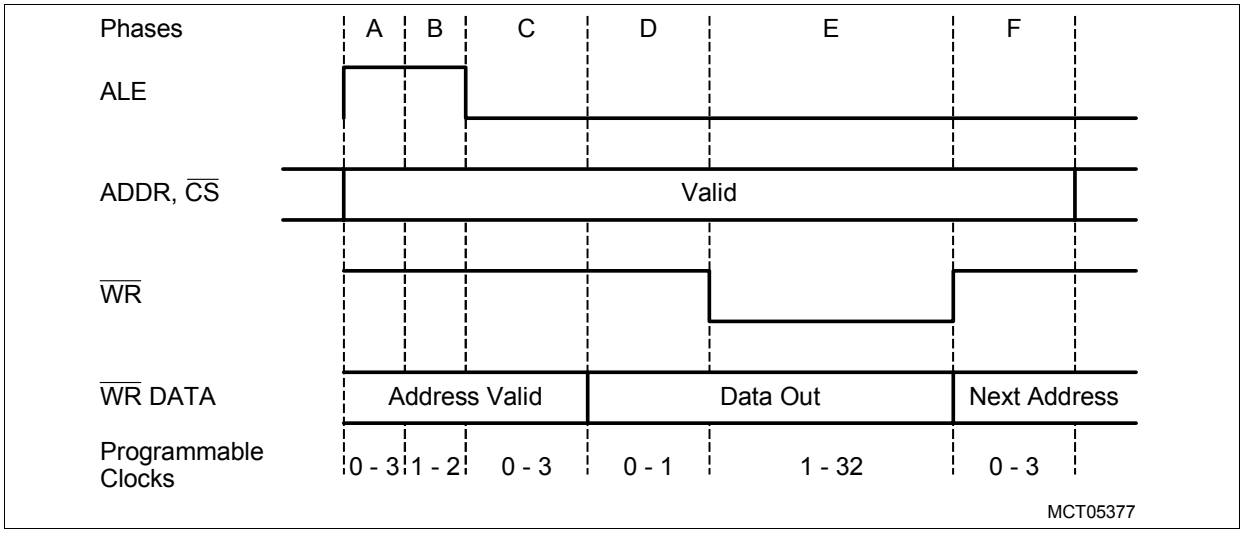


Figure 9-5 Multiplexed Bus Write

- A phase: addresses valid, ALE high, no command. CS switch tristate wait states
- B phase: addresses valid, ALE high, no command. ALE length
- C phase: addresses valid, ALE low, no command. Address hold, R/W delay
- D phase: address tristate for read cycles, data valid for write cycles, ALE low, no command
- E phase: command (read or write) active. Access time
- F phase: command inactive, address hold. Read data tristate time, write data hold time

9.2.2 Bus Cycle Phases

This chapter provides a detail description of each phase of an external memory bus access cycle.

9.2.2.1 A Phase - $\overline{CS}$ Change Phase

The A phase can take 0-3 clocks. It is used for tristating databus drivers from the previous cycle (tristate wait states after chip select switch).
A phase cycles are not inserted at every access cycle, but only when changing the $\overline{CS}$. If an access using one $\overline{CS}$ ($\overline{CSx}$) ends and the next access with a different $\overline{CS}$ ($\overline{CSy}$) is started, then A phase cycles are performed according to the bits set in the **first** $\overline{CS}$ ($\overline{CSx}$). This feature is used to optimize wait states with devices having a long turn-off delay at their databus drivers, such as EPROMs and flash memories.
The A phase cycles are inserted while the addresses and ALE of the next cycle are already applied.
If there are some idle cycles between two accesses, these clocks are taken into account and the A phase is shortened accordingly. For example, if there are three tristate cycles programmed and two idle cycles occur, then the A phase takes only one clock.

9.2.2.2 B Phase - Address Setup/ALE Phase

The B phase can take 1-2 clocks. It is used for addressing devices before giving a command, and defines the length of time that ALE is active. In multiplexed bus mode, the address is applied for latching.

9.2.2.3 C Phase - Delay Phase

The C phase is similar to the A an B phases but ALE is already low. It can take 0-3 clocks. In multiplexed bus mode, the address is held in order to be latched safely. Phase C cycles can be used to delay the command signals (RW delay).

9.2.2.4 D Phase - Write Data Setup/MUX Tristate Phase

The D phase can take 0-1 clocks. It is used to tristate the address on the multiplexed bus when a read cycle is performed. For all write cycles, it is used to ensure that the data are valid on the bus before the command is applied.

9.2.2.5 E Phase - $\overline{RD}$/$\overline{WR}$ Command Phase

The E phase is the command or access phase, and takes 1-32 clocks. Read data are fetched, write data are put onto the bus, and the command signals are active. Read data are registered with the terminating clock of this phase.
The READY function lengthens this phase, too. READY-controlled access cycles may have an unlimited cycle time.

9.2.2.6 F Phase - Address/Write Data Hold Phase

The F phase is at the end of an access. It can take 0-3 clocks.
Addresses and write data are held while the command is inactive. The number of wait states inserted during the F phase is independently programmable for read and write accesses. The F phase is used to program tristate wait states on the bidirectional data bus in order to avoid bus conflicts for read accesses and to assure data hold times for write accesses. The F phase is configured individually for read and write accesses.

9.2.3 Bus Cycle Examples: Fastest Access Cycles

The fastest possible bus cycle in a system depends also on the pad timing. Therefore, the number of required cycles for a bus access depends on the current system frequency. The minimum bus cycles shown below cannot be achieved at very high system frequencies.

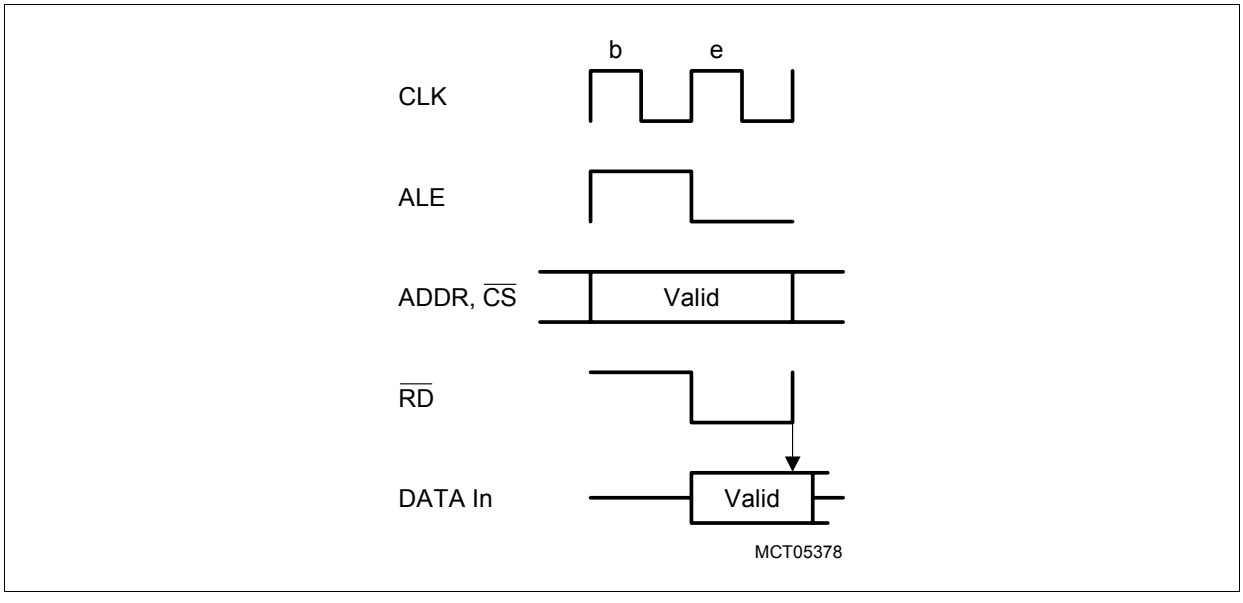


Figure 9-6 Fastest Read Cycle Demultiplexed Bus

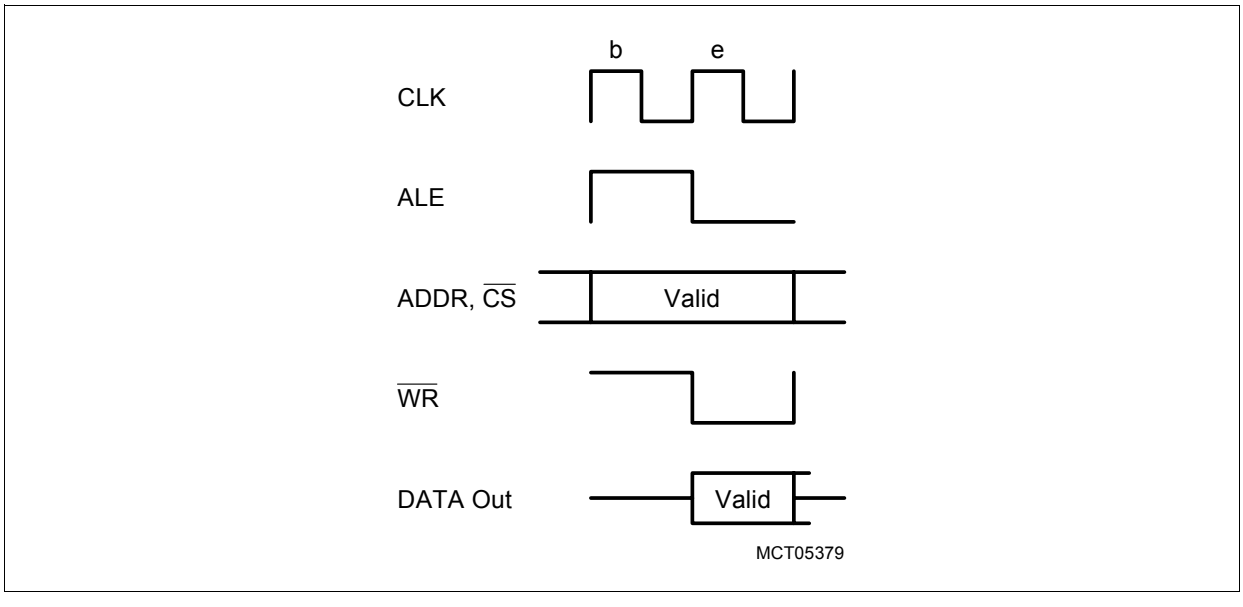


Figure 9-7 Fastest Write Cycle Demultiplexed Bus

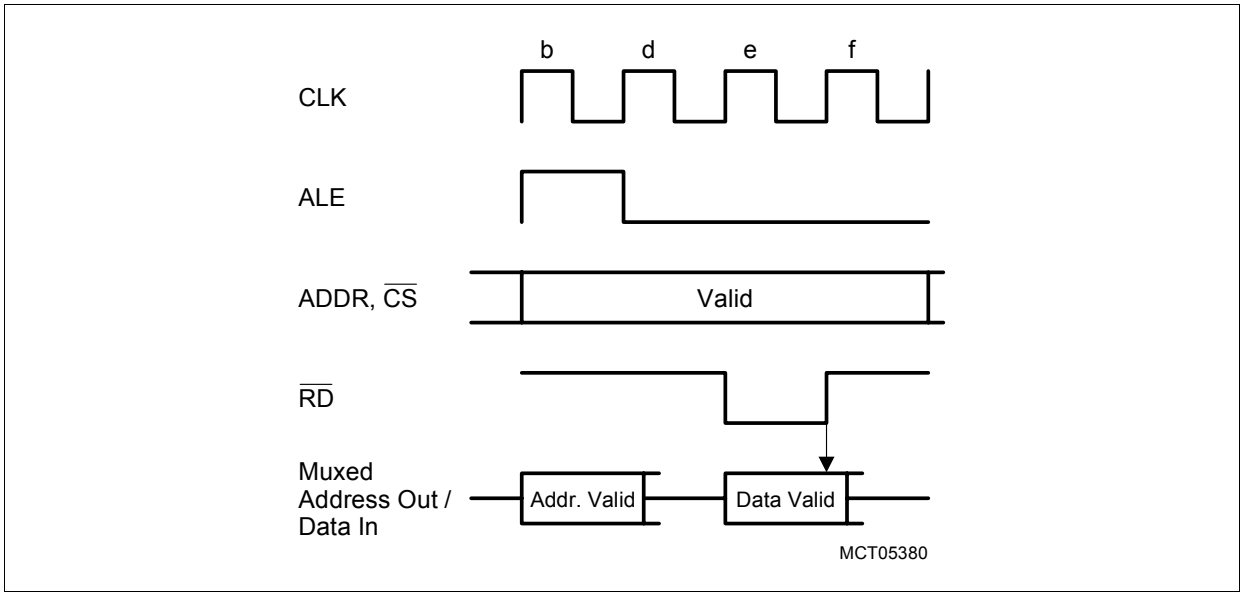


Figure 9-8 Fastest Read Cycle Multiplexed Bus

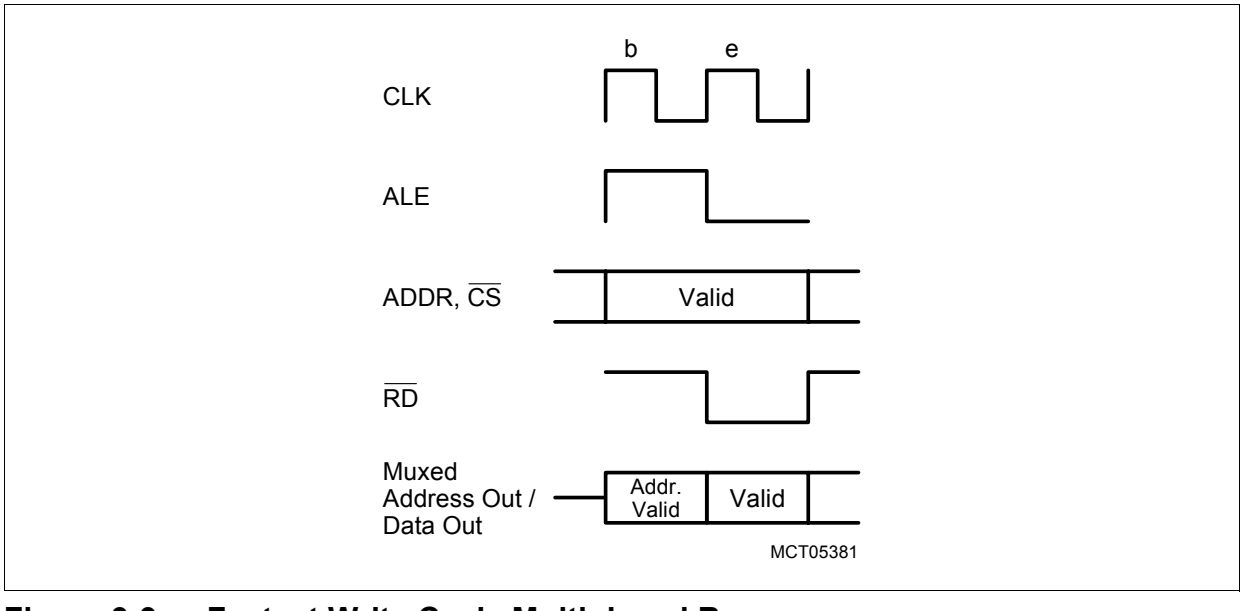


Figure 9-9 Fastest Write Cycle Multiplexed Bus

9.3 Functional Description

The following section describes the EBC registers and their settings.

9.3.1 Configuration Register Overview

There are 3 groups of EBC registers:

- EBC mode registers influencing the global functions.
- Chip-select-related registers controlling the functionality linked to one $\overline{CS}$.
- MultiCAN and USIC related registers are used to control the access to the internal LXBus.

$\overline{CS0}$ is the default chip-select signal that is active whenever no other chip-select or internal address space is addressed. Therefore, $\overline{CS0}$ has no ADDRSEL register.

Note: All EBC registers are write-protected by the EINIT protection mechanism. Thus, after execution of the EINIT instruction, these registers are not writable anymore.

A 128-byte address space is occupied/reserved by the EBC. **Figure 9-10** shows the mapping in this address space, **Table 9-3** lists all implemented configuration registers with address and reset value.

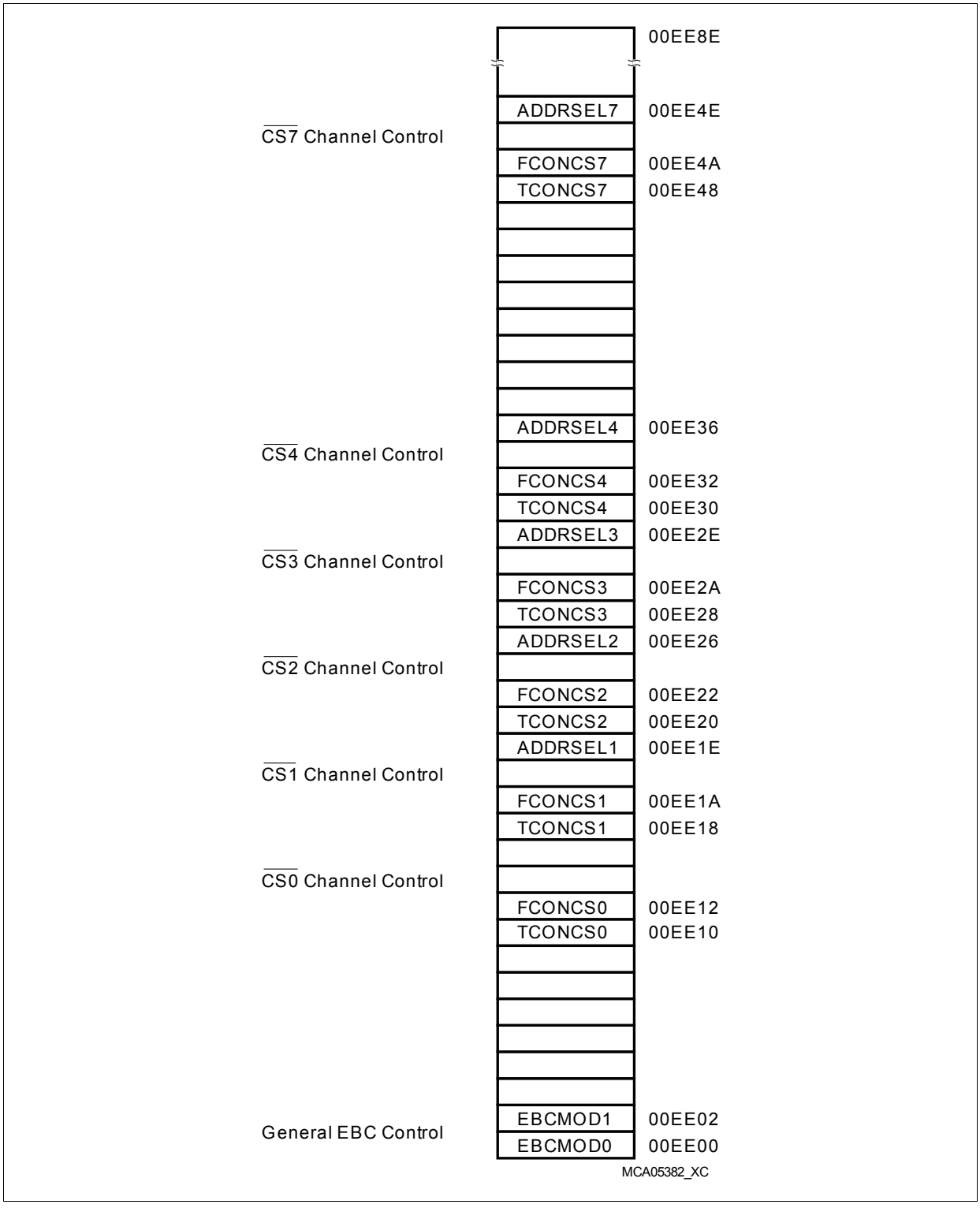


Figure 9-10 Mapping of EBC Registers into the XSFR Space

Note: $\overline{CS5}$ and $\overline{CS6}$ register sets are not available (reserved for future LXBus peripherals).

Table 9-3 EBC Memory Table (ordered by physical address)

Name	Physical Address	Description	Reset Value[1)]
EBCMOD0	$EE00_H$	EBC Mode Register 0	5000_H
EBCMOD1	$EE02_H$	EBC Mode Register 1	$003F_H$
TCONCS0	$EE10_H$	$\overline{CS0}$ Timing Configuration Register	$7C3D_H$
FCONCS0	$EE12_H$	$\overline{CS0}$ Function Configuration Register	0011_H
TCONCS1	$EE18_H$	$\overline{CS1}$ Timing Configuration Register	0000_H
FCONCS1	$EE1A_H$	$\overline{CS1}$ Function Configuration Register	0000_H
ADDRSEL1	$EE1E_H$	$\overline{CS1}$ Address Size and Range Register	0000_H
TCONCS2	$EE20_H$	$\overline{CS2}$ Timing Configuration Register	0000_H
FCONCS2	$EE22_H$	$\overline{CS2}$ Function Configuration Register	0000_H
ADDRSEL2	$EE26_H$	$\overline{CS2}$ Address Size and Range Register	0000_H
TCONCS3	$EE28_H$	$\overline{CS3}$ Timing Configuration Register	0000_H
FCONCS3	$EE2A_H$	$\overline{CS3}$ Function Configuration Register	0000_H
ADDRSEL3	$EE2E_H$	$\overline{CS3}$ Address Size and Range Register	0000_H
TCONCS4	$EE30_H$	$\overline{CS4}$ Timing Configuration Register	0000_H
FCONCS4	$EE32_H$	$\overline{CS4}$ Function Configuration Register	0000_H
ADDRSEL4	$EE36_H$	$\overline{CS4}$ Address Size and Range Register	0000_H
TCONCS7	$EE48_H$	$\overline{CS7}$ Timing Configuration Register	0000_H
FCONCS7	$EE4A_H$	$\overline{CS7}$ Function Configuration Register	0027_H
ADDRSEL7	$EE4E_H$	$\overline{CS7}$ Address Size and Range Register	2003_H
reserved	$EE50_H$ - $EEFF_H$	reserved - do not use	–

1) **NOTE:** Reserved (and not listed) addresses are always read as $FFFF_H$. However, for enabling future enhancements without any compatibility problems, these addresses should neither be written nor be used as read value by the software.

9.3.2 The EBC Mode Register 0

EBCMODe Register 0

EBCMOD0
EBC Mode Register 0 **XSFR ($EE00_H$/--)** **Reset Value: 5000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RDY POL	RDY DIS	ALE DIS	BYT DIS	WR CFG	EBC DIS	SLA VE	ARB EN	CSPEN				SAPEN			
rw	rw	rw	rw	rw	rw	rw	rw	rw				rw			

Field	Bits	Type	Description
RDYPOL	15	rw	**READY Pin Polarity**[1)] 0_B READY is active low 1_B READY is active high
RDYDIS	14	rw	**READY Pin Disable**[1)] 0_B READY enabled 1_B READY disabled
ALEDIS	13	rw	**ALE Pin Disable** 0_B ALE enabled 1_B ALE disabled
BYTDIS	12	rw	**$\overline{BHE}$ Pin Disable** 0_B $\overline{BHE}$ enabled 1_B $\overline{BHE}$ disabled
WRCFG[2)]	11	rw	**Configuration for Pins $\overline{WR}$/$\overline{WRL}$, $\overline{BHE}$/$\overline{WRH}$** 0_B $\overline{WR}$ and $\overline{BHE}$ 1_B $\overline{WRL}$ and $\overline{WRH}$
EBCDIS	10	rw	**EBC Pins Disable** 0_B EBC is using the pins for external bus 1_B EBC pins disabled
SLAVE	9	rw	**SLAVE Mode Enable** 0_B Bus arbiter acts in master mode 1_B Bus arbiter acts in slave mode
ARBEN	8	rw	**BUS Arbitration Pins Enable** 0_B $\overline{HOLD}$, $\overline{HLDA}$ and $\overline{BREQ}$ pins are disabled 1_B Pins act as $\overline{HOLD}$, $\overline{HLDA}$, and $\overline{BREQ}$

Field	Bits	Type	Description
CSPEN	[7:4]	rw	**$\overline{CSx}$ Pins Enable (only external $\overline{CSx}$)** 0000_B All external Chip Select pins disabled. 0001_B $\overline{CS0}$ pin enabled 0010_B CS1 and $\overline{CS0}$ pin enabled … … 0101_B Five $\overline{CSx}$ pins enabled: $\overline{CS4}$ - $\overline{CS0}$ Else not supported (reserved)
SAPEN	[3:0]	rw	**Segment Address Pins Enable** 0000_B All segment address pins disabled 0001_B One: A[16] enabled … … 1000_B Eight: A[23:16] enabled Else not supported (reserved)

1) Not available in the 100-pin package.

2) A change of the bit content is not valid before the next external bus access cycle.

Notes

1. *Disabled pins are used for general purpose IO or for alternate functions (see port and pin descriptions).*
2. *Bit field CSPEN controls the number of available $\overline{CSx}$ **pins**. The related address windows and bus functions are enabled with the specific ENCSx bits in the FCONCSx registers (see **Page 9-21**). There, an additional chip select ($\overline{CS7}$) is defined for internal access to the LXBus peripherals MultiCAN and USIC.*
3. *The external bus arbitration pins have a separate ARBitration ENable bit (ARBEN) that has to be set in order to use the pins for arbitration and not for General Purpose IO (GPIO). If ARBEN is cleared, the arbitration inputs $\overline{HLDA}$ and $\overline{HOLD}$ are fixed internally to an inactive high state. Additionally, the master/slave setting of the arbiter is done with a separate bit (SLAVE).*
4. *The reset value depends on the selected startup configuration.*

9.3.3 The EBC Mode Register 1

EBC MODe register 1 controls the general use of port pins for external bus.

EBCMOD1
EBC Mode Register 1 **XSFR ($EE02_H$/--)** **Reset Value: $003F_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	WRP DIS	DHP DIS	ALP DIS	A0P DIS	APDIS			
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw			

Field	Bits	Type	Description
WRPDIS	7	rw	**$\overline{WR}$/$\overline{WRL}$ Pin Disable** 0_B $\overline{WR}$/$\overline{WRL}$ pin enabled 1_B $\overline{WR}$/$\overline{WRL}$ pin disabled
DHPDIS	6	rw	**Data High Port Pins Disable** 0_B Address/Data bus pins 15-8 enabled 1_B Address/Data bus pins 15-8 disabled
ALPDIS	5	rw	**Address Low Pins Disable** 0_B Address bus pins 7-0 generally enabled (depending on APDIS/A0PDIS) 1_B Address bus pins 7-0 disabled
A0PDIS	4	rw	**Address Bit 0 Pin Disable** 0_B Address bus pin 0 enabled 1_B Address bus pin 0 disabled
APDIS	[3:0]	rw	**Address Port Pins Disable** 0000_B Address bus pins 15-1 enabled 0001_B Pin A15 disabled, A14-A1 enabled 0010_B Pins A15-A14 disabled, A13-A1 enabled 0011_B Pins A15-A13 disabled, A12-A1 enabled … … 1110_B Pins A15-A2 disabled, A1 enabled 1111_B Address bus pins 15-1 disabled

Notes

1. *Disabled bus pins may be used for general purpose IO or for alternate functions (see port and pin descriptions).*
2. *After reset, the address and data bus pins are enabled, but in Idle state.*

9.3.4 The Timing Configuration Registers TCONCSx

The timing control registers are used to program the described cycle timing for the different access phases (see **Section 9.2.2**). The timing control registers may be reprogrammed during code fetches from the affected address window. The new settings are first valid for the next access.

TCONCS0
Timing Cfg. Reg. for $\overline{CS0}$ **XSFR ($EE10_H$/--)** **Reset Value: $7C3D_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	WRPHF		RDPHF		PHE					PHD	PHC		PHB	PHA	
-	rw		rw		rw					rw	rw		rw	rw	

Field	Bits	Type	Description
WRPHF	[14:13]	rw	**Write Phase F** 00_B 0 clock cycles … … 11_B 3 clock cycles (default)
RDPHF	[12:11]	rw	**Read Phase F** 00_B 0 clock cycles (default) … … 11_B 3 clock cycles
PHE	[10:6]	rw	**Phase E** 00000_B1 clock cycle … … (default: 9 clock cycles) 11111_B32 clock cycles
PHD	5	rw	**Phase D** 0_B 0 clock cycles (default) 1_B 1 clock cycle
PHC	[4:3]	rw	**Phase C** 00_B 0 clock cycles (default) … … 11_B 3 clock cycles
PHB	2	rw	**Phase B** 0_B 1 clock cycle (default) 1_B 2 clock cycles

Field	Bits	Type	Description
PHA	[1:0]	rw	**Phase A** 00_B 0 clock cycles … … 11_B 3 clock cycles (default)

TCONCSx (x = 1-4)
Timing Cfg. Reg. for $\overline{CSx}$ **XSFR ($EE10_H$ + x*8/--)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	WRPHF		RDPHF		PHE					PHD	PHC		PHB	PHA	
-	rw		rw		rw					rw	rw		rw	rw	

Field	Bits	Type	Description
WRPHF	[14:13]	rw	**Write Phase F** 00_B 0 clock cycles … … 11_B 3 clock cycles
RDPHF	[12:11]	rw	**Read Phase F** 00_B 0 clock cycles … … 11_B 3 clock cycles
PHE	[10:6]	rw	**Phase E** 00000_B1 clock cycle … … 11111_B32 clock cycles
PHD	5	rw	**Phase D** 0_B 0 clock cycles 1_B 1 clock cycle
PHC	[4:3]	rw	**Phase C** 00_B 0 clock cycles … … 11_B 3 clock cycles
PHB	2	rw	**Phase B** 0_B 1 clock cycle 1_B 2 clock cycles

Field	Bits	Type	Description
PHA	[1:0]	rw	**Phase A** 00_B 0 clock cycles … … 11_B 3 clock cycles

Note: The register TCONCS4 controls the chip select $\overline{CS4}$, that is available only in the 144-pin package.

TCONCS7
Timing Cfg. Reg. for $\overline{CS7}$ **XSFR ($EE48_H$/--)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	WRPHF		RDPHF		PHE					PHD	PHC		PHB	PHA	
-	r		r		r					r	r		r	r	

Field	Bits	Type	Description
WRPHF	[14:13]	r	**Write Phase F** 00_B 0 clock cycles
RDPHF	[12:11]	r	**Read Phase F** 00_B 0 clock cycles
PHE	[10:6]	r	**Phase E** 00000_B1 clock cycle
PHD	5	r	**Phase D** 0_B 0 clock cycles
PHC	[4:3]	r	**Phase C** 00_B 0 clock cycles
PHB	2	r	**Phase B** 0_B 1 clock cycle
PHA	[1:0]	r	**Phase A** 00_B 0 clock cycles

Note: $\overline{CS7}$ is used and defined for internal access to the LXBus peripherals MultiCAN and USIC.

9.3.5 The Function Configuration Registers FCONCSx

The Function Control registers are used to control the bus and READY functionality for a selected address window. It can be distinguished between 8 and 16-bit bus and multiplexed and demultiplexed accesses. Furthermore it can be defined whether the address window (and its chip select signal $\overline{CSx}$) is generally enabled or not.

FCONCS0
Function Cfg. Reg. for $\overline{CS0}$ **XSFR ($EE12_H$/--)** **Reset Value: 0011_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	BTYP		-	RDY MOD	RDY EN	EN CS
-	-	-	-	-	-	-	-	-	-	rw		-	rw	rw	rw

Field	Bits	Type	Description
BTYP	[5:4]	rw	**Bus Type Selection** 00_B 8 bit Demultiplexed 01_B 08 bit Multiplexed 10_B 16 bit Demultiplexed 11_B 16 bit Multiplexed
RDYMOD	2	rw	**Ready Mode** 0_B Asynchronous READY 1_B Synchronous READY
RDYEN	1	rw	**Ready Enable** 0_B Access time is controlled by bit field PHEx 1_B Access time is controlled by bit field PHEx and READY signal
ENCS[1)]	0	rw	**Enable Chip Select** 0_B Disable 1_B Enable

1) Disabling a chip select not only effects the chip select output signal, it also deactivates the respective address window of the disabled chip select. A disabled address window is also ignored by an address window arbitration (see **Chapter 9.3.6.3**).

FCONCSx (x = 1-4)
Function Cfg. Reg. for $\overline{CSx}$ XSFR ($EE12_H$ + x*8/--) Reset Value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	BTYP		-	RDY MOD	RDY EN	EN CS
-	-	-	-	-	-	-	-	-	-	rw		-	rw	rw	rw

Field	Bits	Type	Description
BTYP	[5:4]	rw	**Bus Type Selection** 00_B 8 bit Demultiplexed 01_B 8 bit Multiplexed 10_B 16 bit Demultiplexed 11_B 16 bit Multiplexed
RDYMOD	2	rw	**Ready Mode** 0_B Asynchronous READY 1_B Synchronous READY
RDYEN	1	rw	**Ready Enable** 0_B Access time is controlled by bit field PHEx 1_B Access time is controlled by bit field PHEx and READY signal
ENCS[1)]	0	rw	**Enable Chip Select** 0_B Disable 1_B Enable

1) Disabling a chip select not only effects the chip select output signal, it also deactivates the respective address window of the disabled chip select. A disabled address window is also ignored by an address window arbitration (see **Chapter 9.3.6.3**).

Notes

1. *The register FCONCS4 controls the chip select $\overline{CS4}$, that is available only in the 144-pin package.*
2. *The specific ENCSx bits in the FCONCSx registers enable the related address windows and bus functions and the corresponding chip select signal $\overline{CSx}$. But it depends on the definition of bit field CSPEN in register EBCMOD0 how many $\overline{CSx}$ **pins** are available and used for the external system. If an address window is enabled but no external pin is available for the $\overline{CSx}$, the external bus cycle is executed without chip select signal.*

FCONCS7
Function Cfg. Reg. for $\overline{CS7}$ **XSFR ($EE4A_H$/--)** **Reset Value: 0027_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	BTYP		-	RDY MOD	RDY EN	EN CS
-	-	-	-	-	-	-	-	-	-	r		-	r	r	r

Field	Bits	Type	Description
BTYP	[5:4]	r	**Bus Type Selection** 10_B 16 bit Demultiplexed
RDYMOD	2	r	**Ready Mode** 1_B Synchronous READY
RDYEN	1	r	**Ready Enable** 1_B Access time is controlled by bit field PHEx and READY signal
ENCS	0	r	**Enable Chip Select** 1_B Enable

Notes

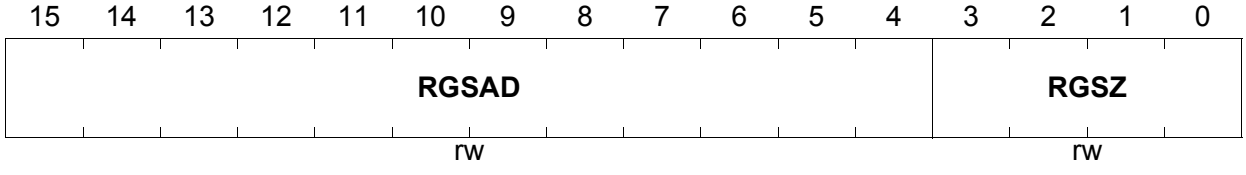
1. *With ENCS7 the chip select $\overline{CS7}$ and its related register set is enabled and defined for internal access to the LXBus peripherals MultiCAN and USIC.*

9.3.6 The Address Window Selection Registers ADDRSELx

Each chip select signal is associated with an ADDRSEL register.

9.3.6.1 Registers ADDRSELx

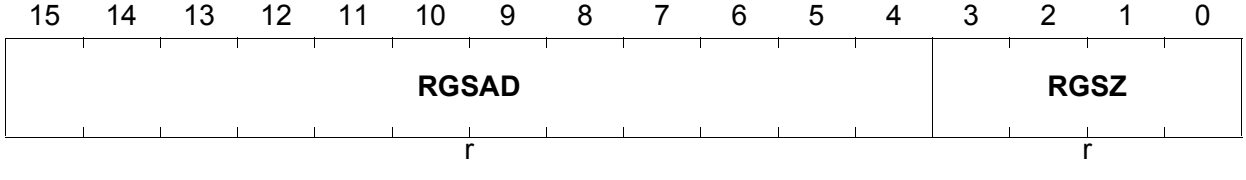
ADDRSELx (x = 1-4)
Address Range/Size for $\overline{CSx}$ XSFR ($EE16_H$ + x*8/--) Reset Value: 0000_H



Field	Bits	Type	Description
RGSAD	[15:4]	rw	Address Range Start Address Selection
RGSZ	[3:0]	rw	Address Range Size Selection (see **Table 9-4**)

Note: There is no register ADDRSEL0, as register set FCONCS0/TCONCS0 controls all external accesses outside the address windows built by the enabled (by ENCS bit in FCONCSx) address selects ADDRSELx. The register ADDRSEL4 controls the chip select $\overline{CS4}$, that is available only in the 144-pin package.

ADDRSEL7
Address Range/Size for $\overline{CS7}$ XSFR ($EE4E_H$/--) Reset Value: 2003_H



Field	Bits	Type	Description
RGSAD	[15:4]	r	Address Range Start Address $20'0000_H$
RGSZ	[3:0]	r	Address Range Size 32 Kbytes (see **Table 9-4**)

9.3.6.2 Definition of Address Areas

The enabled register sets FCONCSx/TCONCSx/ADDRSELx (x = 1 … 4, 7) define separate address areas within the address space of the XE16x. Within each of these address areas the conditions of external accesses and LXBus accesses (x = 7) can be

controlled separately, whereby the different address areas (windows) are defined by the ADDRSELx registers. Each ADDRSELx register cuts out an address window, where the corresponding parameters of the registers FCONCSx and TCONCSx are used to control external accesses. The range start address of such a window defines the most significant address bits of the selected window which are consequently not needed to address the memory/module in this window (**Table 9-4**). The size of the window chosen by ADDRSELx.RGSZ defines the relevant bits of ADDRSELx.RGSAD (marked with 'R') which are used to select with the most significant bits of the request address the corresponding window. The other bits of the request address are used to address the memory locations inside this window. The lower bits of ADDRSELx.RGSAD (marked 'x') are disregarded.

The address area from 00'8000$_H$ to 00'FFFF$_H$ (32 Kbytes) is reserved for CPU internal registers and data RAM, the area from BF'0000$_H$ to BF'7FFF$_H$ (32 Kbytes) for internal startup memory and the area from C0'0000$_H$ to FF'FFFF$_H$ (4 Mbytes) is used by the internal program memory. Therefore, these address areas cannot be used by external resources connected to the external bus.

Table 9-4 Address Range and Size for ADDRSELx

ADDRSELx		Address Window	
Range Size RGSZ	**Relevant (R) Bits of RGSAD**	**Selected Address Range**	**Range Start Address A[23:0] Selected with R-bits of RGSAD**
3 … 0	**15 … 4**	**Size**	**A23 … A0**
0000	RRRR RRRR RRRR	4 Kbytes	RRRR RRRR RRRR 0000 0000 0000
0001	RRRR RRRR RRRx	8 Kbytes	RRRR RRRR RRR0 0000 0000 0000
0010	RRRR RRRR RRxx	16 Kbytes	RRRR RRRR RR00 0000 0000 0000
0011	RRRR RRRR Rxxx	32 Kbytes	RRRR RRRR R000 0000 0000 0000
0100	RRRR RRRR xxxx	64 Kbytes	RRRR RRRR 0000 0000 0000 0000
0101	RRRR RRRx xxxx	128 Kbytes	RRRR RRR0 0000 0000 0000 0000
0110	RRRR RRxx xxxx	256 Kbytes	RRRR RR00 0000 0000 0000 0000
0111	RRRR Rxxx xxxx	512 Kbytes	RRRR R000 0000 0000 0000 0000
1000	RRRR xxxx xxxx	1 Mbytes	RRRR 0000 0000 0000 0000 0000
1001	RRRx xxxx xxxx	2 Mbytes	RRR0 0000 0000 0000 0000 0000
1010	RRxx xxxx xxxx	4 Mbytes	RR00 0000 0000 0000 0000 0000
1011	Rxxx xxxx xxxx	8 Mbytes	R000 0000 0000 0000 0000 0000
11xx	xxxx xxxx xxxx	reserved[1)]	---- ---- ---- ---- ---- ----

1) The complete address space of 12 Mbytes can be selected by the default chip select $\overline{CS0}$.

*Note: The range start address can only be on boundaries specified by the selected range size according to **Table 9-4**.*

9.3.6.3 Address Window Arbitration

For each external access the EBC compares the current address with all address select registers (programmable ADDRSELx and hard wired address select registers for startup memory) of enabled windows. This comparison is done in four levels:

Priority 1:

Registers ADDRSELx [x = 2, 4] are evaluated first. A window match with one of these registers directs the access to the respective external area using the corresponding set of control registers FCONCSx/TCONCSx and ignoring registers ADDRSELy. An overlapping of windows of this group will lead to an undefined behaviour.

Priority 2:

A match with registers ADDRSELy [y = 1, 3, 7] directs the access to the respective external area using the corresponding set of control registers FCONCSy/TCONCSy. An overlapping of windows of this group will lead to an undefined behaviour. Overlaps with priority 2 ADDRSELx are only allowed for the (x, y) pairs (2, 1) and (4, 3).

Priority 3:

If there is no match with any address select register (neither the hardware ones nor the programmable ADDRSEL) the access to the external bus uses the general set of control registers FCONCS0/TCONCS0 if enabled.

9.3.7 Ready Controlled Bus Cycles

In cases, where the response (access) time of a peripheral is not constant, or where the programmable wait states are not enough, the EBC provides external bus cycles that are terminated via a READY input signal.

A READY controlled bus cycle requires one synchronization cycle to terminate. Programmed phase F cycles include this synchronization cycle. Therefore, setting TCONCSx phase F to 0 clock cycles will have the same effect as setting it to 1 clock cycle.

9.3.7.1 General

If READY is enabled, the EBC counts a programmable number of clock cycles (1 … 32) during phase E and then monitors the internal READY line (see **Figure 9-11**) to determine the actual end of the current bus cycle. The external device drives READY active in order to indicate that data has been latched (write cycle) or is available (read cycle).

The READY pin is generally enabled by setting the bit RDYDIS in EBCMOD0 to ‘0’ in order to switch the corresponding port pin. Also the polarity of the READY is defined inside the EBCMOD0 register on the RDYPOL bit.

For a specific address window the READY function is enabled via the RDYEN bit in the FCONCSx register. With FCONCSx.RDYMOD the READY is handled either in synchronous or in asynchronous mode (see also **Figure 9-11**).

When the READY function is enabled for a specific address window, each bus cycle within this window must be terminated with an active READY signal. Otherwise the controller hangs until the next reset. This is also the case for an enabled RDYEN but a disabled READY port pin.

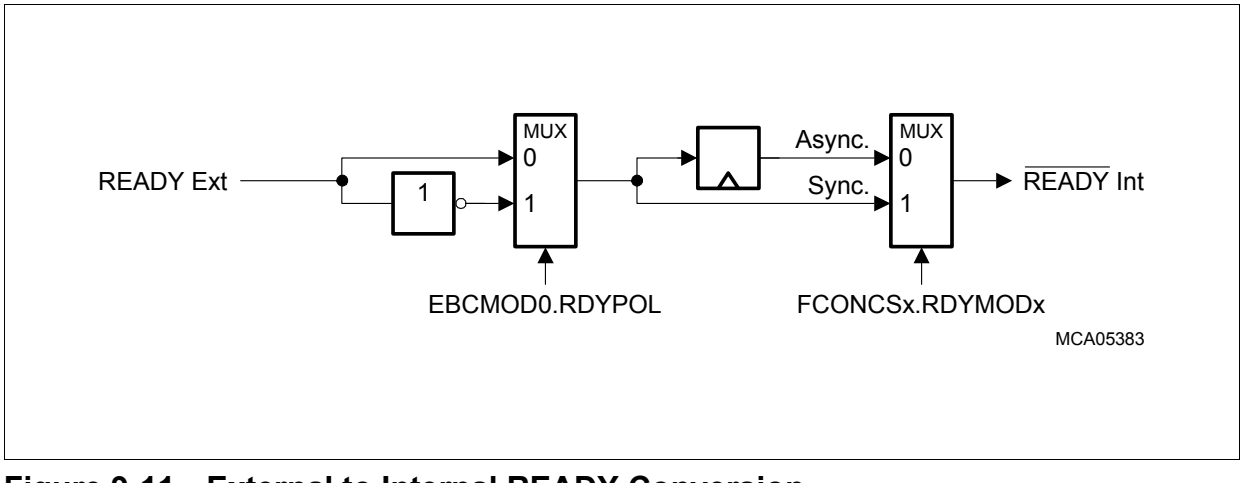


Figure 9-11 External to Internal READY Conversion

9.3.7.2 The Synchronous/Asynchronous READY

The **synchronous** READY provides faster bus cycles, but requires setup and hold times to be met. The CLKOUT signal should be enabled and may be used by the peripheral logic to control the READY timing in this case.

The **asynchronous** READY is less restrictive, but requires one additional wait state caused by the internal synchronization. As the asynchronous READY is sampled earlier programmed wait states may be necessary to provide proper bus cycles.

A READY signal (especially asynchronous READY) that has been activated by an external device may be deactivated in response to the trailing (rising) edge of the respective command ($\overline{RD}$ or $\overline{WR}$).

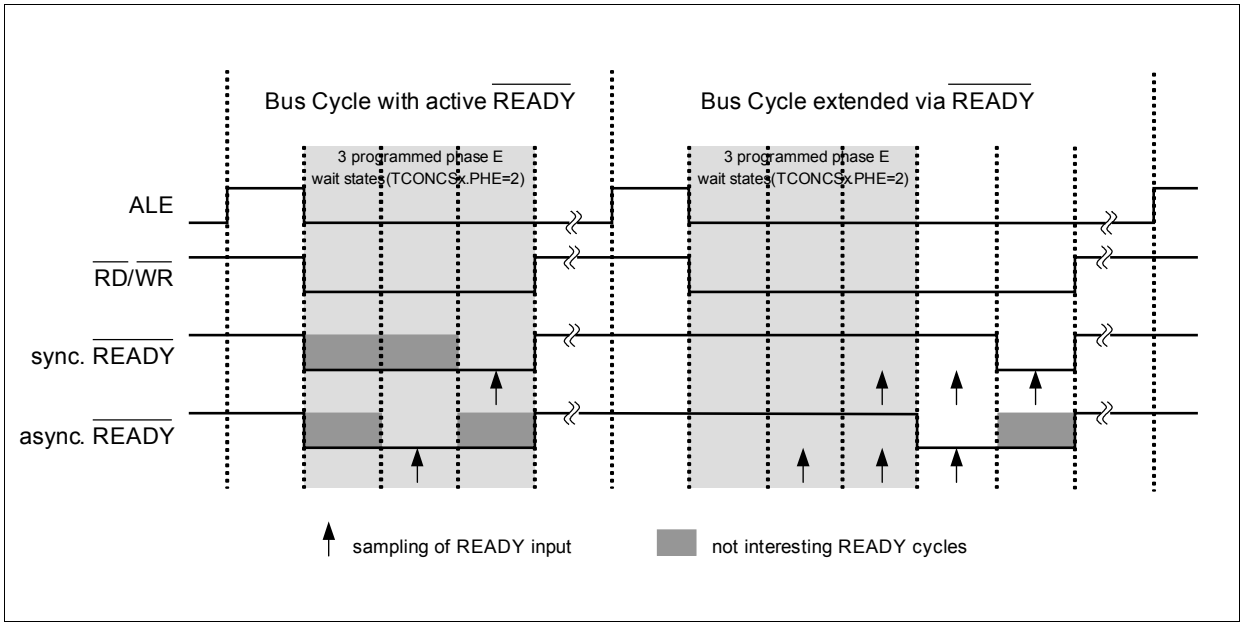


Figure 9-12 READY Controlled Bus Cycles

9.3.7.3 Combining the READY Function with Predefined Wait States

Typically an external wait state or READY control logic takes a while to generate the READY signal when a cycle was started. After a predefined number of clock cycles the EBC will start checking its READY line to determine the end of the bus cycle.

When using the READY function with so-called ‘normally-ready’ peripherals, it may lead to erroneous bus cycles, if the READY line is sampled too early. These peripherals pull their READY output active, while they are idle. When they are accessed, they drive READY inactive until the bus cycle is complete, then drive it active again. If, however, the peripheral drives READY inactive a little late, after the first sample point of the XE16x, the controller samples an active READY and terminates the current bus cycle too early. By inserting predefined wait states the first READY sample point can be shifted to a time, where the peripheral has safely controlled the READY line.

9.3.8 External Bus Arbitration

The XE16x supports multi master systems on the external bus by its external bus arbitration. This bus arbitration allows an external master to request the external bus. The XE16x will release the external bus and will float the data, address bus, and control lines.

Note: In this case, the pins are controlled by their respective IOCR registers (lower 3 bits). It is recommended to activate a pull-up or pull-down device by pre-configuring the IOCR registers. The default setting leaves the pins floating. In particular it is strongly recommended to configure pull-ups at least for the control lines RD, WR, and BREQ, and a pull-down for ALE. External pull resistors serve the same purpose, of course.

9.3.8.1 Initialization of Arbitration

During reset all arbitration pins are tristate. After reset the XE16x EBC always starts in ‘init mode’ where the external bus is available but no arbitration is enabled. All arbitration pins are ignored in this state. Other XE16x EBCs connected to the external bus, also assume to have the bus, so potential bus conflicts are not resolved. For a multi master system the arbitration should be initialized first before starting any bus access. The EBC can either be chosen as arbitration master or as arbitration slave by programming bit SLAVE in register EBCMOD0. The selected mode and the arbitration gets active by the first setting of bit HLDEN in (CPU) register PSW. Afterwards a change of the slave/master mode is not possible without resetting the device. Of course, for arbitration the dedicated pins have to be activated by setting bit EBCMOD0.ARBEN.

9.3.8.2 Arbitration Master Scheme

If the XE16x EBC is configured as arbitration master, it is default owner of the external bus, controls the arbitration protocol and drives the bus also during idle phases with no bus requests. To perform the arbitration handshake, a HOLD input allows the request of the external bus from the arbitration master. When the arbitration master hands over the bus to the requester, this is signaled by driving the hold acknowledge pin HLDA low, which remains at this level until the arbitration slave frees the bus by releasing its request on the HOLD input. If the arbitration master is not the owner of the bus it releases the external bus interface.

Note: Pull devices provide a safe inactive state of the external bus during the bus owner transition. Make sure to enable these pull devices on the respective port pins before using the bus arbitration (see also note above).

In this state the arbitration slave can take over the bus.

If the arbitration master requires the bus again, it can request the bus via the bus request signal BREQ. As soon as the arbitration master regains the bus it releases the BREQ signal and drives HLDA to high.

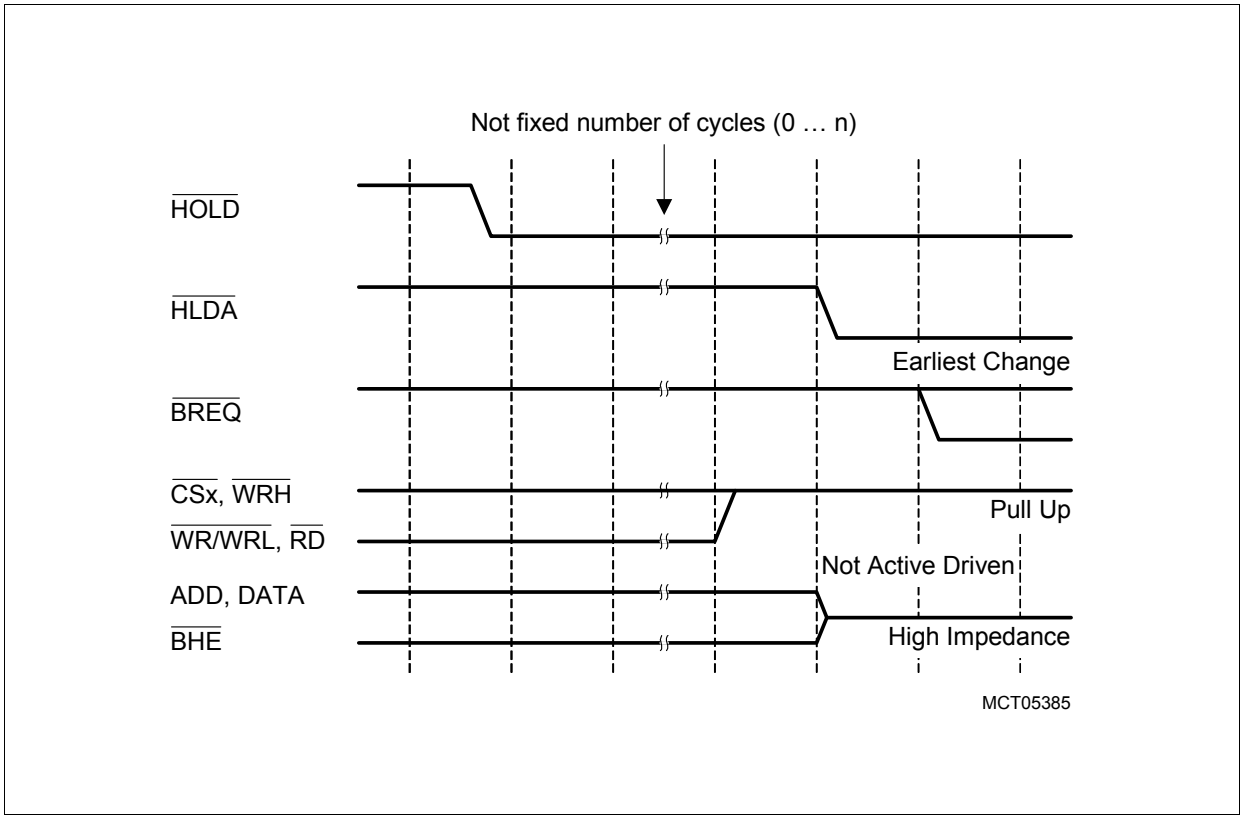


Figure 9-13 Releasing the Bus by the Arbitration Master

*Note: **Figure 9-13** shows the first possibility for $\overline{BREQ}$ to get active. The XE16x will complete the currently running bus cycle before granting the external bus as indicated by the broken lines.*

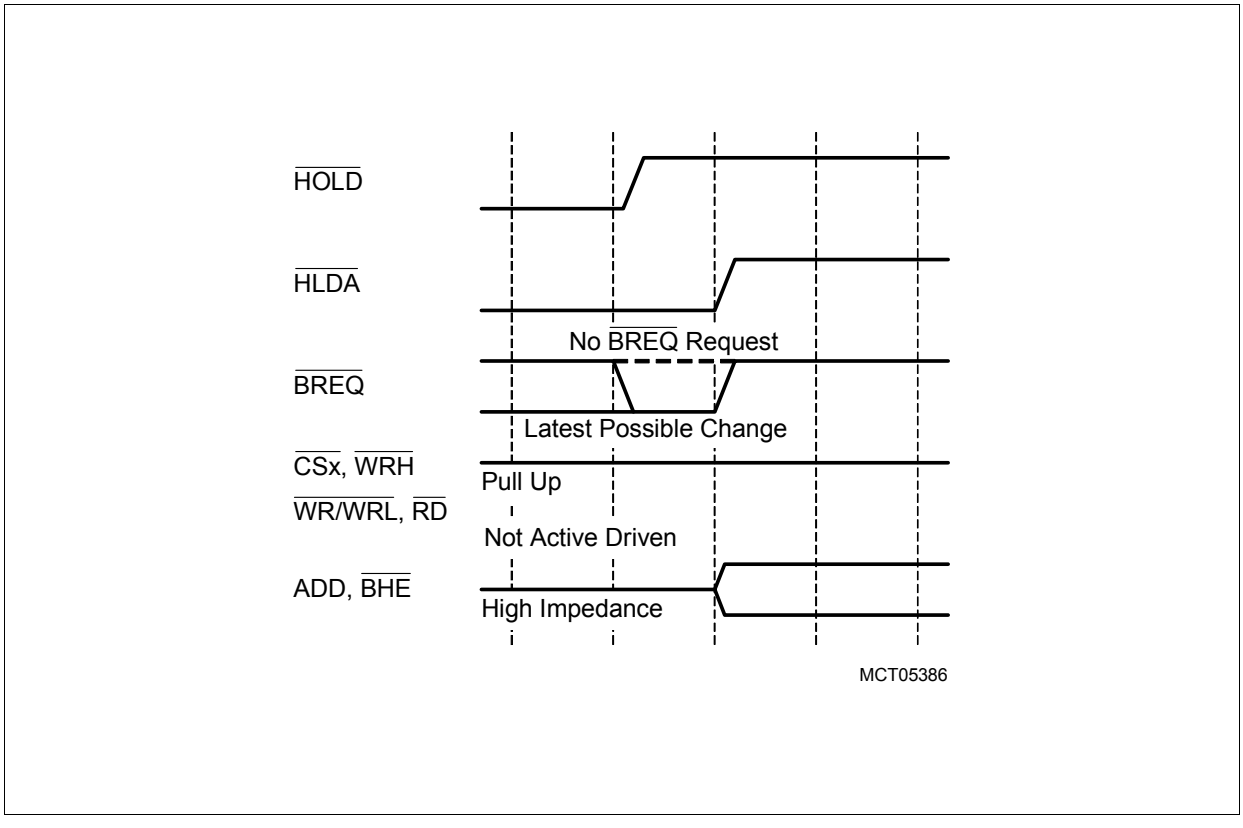


Figure 9-14 Regaining the Bus by the Arbitration Master

Note: The falling BREQ edge shows the last chance for BREQ to trigger the indicated regain-sequence. Even if BREQ is activated earlier the regain-sequence is initiated by HOLD going high. Please note that HOLD may also be deactivated without the XE16x requesting the bus.

9.3.8.3 Arbitration Slave Scheme

If the EBC is configured as arbitration slave it is by default not owner of the external bus and has to request the bus first. As long as it has not finished all its queued requests and the arbitration master is not requesting the bus the arbitration slave stays owner of the bus. For the description of the signal handling of the handshake see **Chapter 9.3.8.2**. For the arbitration slave the hold acknowledge pin HLDA is configured as input.

9.3.8.4 Bus Lock Function

If an application in a multi master system requires a sequence of undisturbed bus access it has the possibility (independently of being arbitration slave or master) to lock[1] the bus by setting the PSW bit HLDEN to ‘0’. In this case the locked EBC will not answer to HOLD requests from other external bus master until HLDEN is set to ‘1’ again. Of course a locked bus master not owning the bus can request the external bus. If a master and a slave are requesting the external bus at the same time for several accesses, they toggle the ownership after each access cycle if the bus is not locked.

9.3.8.5 Direct Master Slave Connection

If one XE16x is configured as master and the other as slave and both are working on the same external bus as bus master, they can be connected directly together for bus arbitration as shown in **Figure 9-15**. As both EBCs assume after reset to own the external bus, the ‘slave’ CPU has to be released from reset and initialized first, before starting the ‘master’ CPU. The other way is to start both systems at the same time but then both EBC must be configured from internal memory and the PSW.HLDEN bits set before the first external bus request.

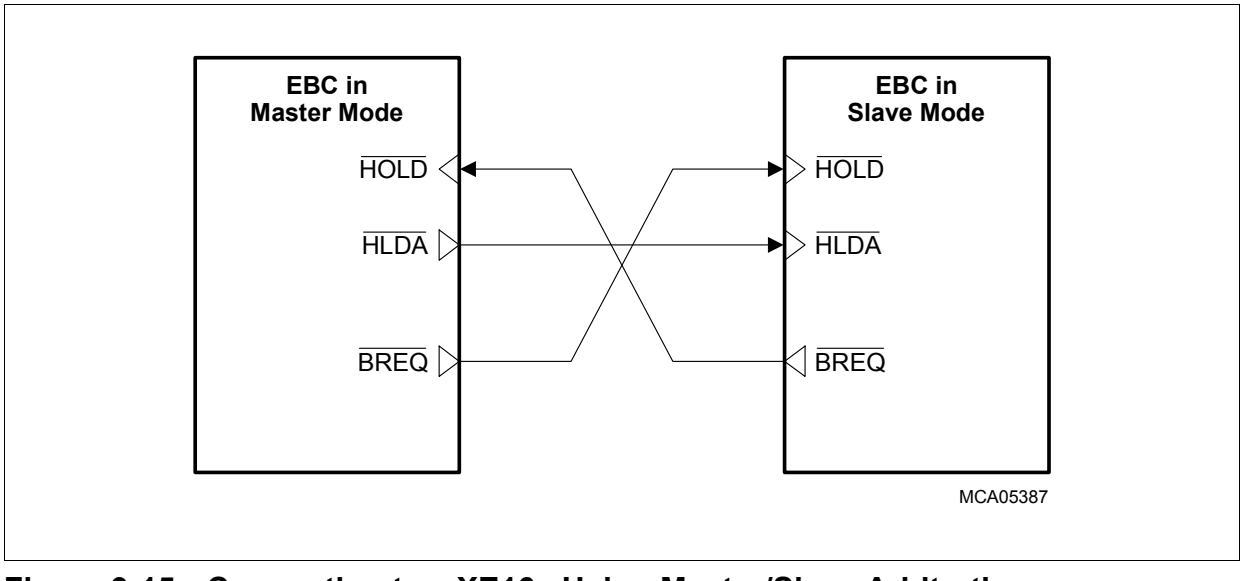


Figure 9-15 Connecting two XE16x Using Master/Slave Arbitration

When multiple (more than two) bus masters (XE16x or other masters) shall share the same external resources an additional external bus arbiter logic is required that determines the currently active bus master and that controls the necessary signal sequences.

1) It is not allowed to lock the bus by resetting the EBCMOD0.ARBEN bit, as this can lead to bus conflicts.

9.3.9 Shutdown Control

In case of a shutdown request from the SCU the EBC ensures that all the different functions of the EBC are in a non-active state before the whole chip is switched to a power save mode. A running bus cycle is finished, still requested bus cycles are executed. Depending on the master/slave configuration of EBC, the external bus arbiter is controlled for regaining the bus (master) before performing the requested cycles, or the external bus must be released after complete execution of still requested bus cycles (see **Table 9-5**). Only when this shutdown sequence is terminated, the shutdown acknowledge is generated from EBC (and from other modules, as described for SCU) and the chip can enter the requested mode.

Table 9-5 gives an overview of the shutdown control in EBC depending on the EBC configuration.

Table 9-5 EBC Shutdown Control

Arbitration Mode	**Master Mode**		**Slave Mode**	
Bus Control	**With Control of the Bus**	**Without Control of the Bus**	**With Control of the Bus**	**Without Control of the Bus**
–	Finish all pending cycle requests. Send shutdown acknowledge with the control of the bus.	Ask for the bus. Finish all pending cycle requests. Send shutdown acknowledge with the control of the bus.	Finish all pending requests. Send shutdown acknowledge after leaving the bus.	Ask for the bus if needed and finish all requests. Send shutdown acknowledge after leaving the bus.

9.4 LXBus Access Control and Signal Generation

Access control to the LXBus[1] is required for the on-chip peripherals MultiCAN and USIC. For these accesses, $\overline{CS7}$ and its fixed control registers ADDRSEL7, TCONCS7, and FCONCS7 are used. The address range (from $20'0000_H$ to $20'7FFF_H$), defined by the ADDRSEL7 value of 2003_H, is located in the 'External IO Area'. Only for the External IO Area (within the total external address range) it is guaranteed that a read access is executed after a preceding write access.

The value of the bus function control register FCONCS7 is selected according to the requirements of the MultiCAN and USIC: 16-bit demultiplexed bus, access time controlled with synchronous READY. This function control is represented by the FCONCS7 value of 0027_H.

The minimum LXBus cycle timing (as controlled with register TCONCS7) will be lengthened with waitstate(s) controlled by the MultiCAN/USIC itself with the READY function. This timing control is defined by the TCONCS7 value of 0000_H.

Accesses to the LXBus do not generate valid external bus cycles on an enabled external bus interface:

- the configured chip select signals are driven high
- the external control signal pins ($\overline{RD}$, $\overline{WR}$, ALE) are driven inactive and then switched to input
- the other bus pins are switched to input, the address lines may reflect the LXBus address for a short period

Note: In this case, the pins are controlled by their respective IOCR registers (lower 3 bits). It is recommended to activate a pull-up or pull-down device by pre-configuring the IOCR registers. The default setting leaves the pins floating. In particular it is strongly recommended to configure pull-ups at least for the control lines $\overline{RD}$ and $\overline{WR}$, and a pull-down for ALE. External pull resistors serve the same purpose, of course.

1) The LXBus is an internal (local) extension of the external bus. It is controlled by the EBC identically to the external bus, using the select and cycle control functions as described for the external bus.

10 Startup Configuration and Bootstrap Loading

After start-up, the XE16x executes code out of an on-chip or off-chip program memory. The initial code source can be selected via hardware configuration (i.e. defined levels on specific pins):

- **Internal Start** Mode: executes code out of the on-chip program Flash.
- **External Start** Mode: executes code out of an off-chip memory connected to the External Bus Interface.
- **Bootstrap Loading** Modes: execute code out of the on-chip program SRAM (PSRAM). This code is downloaded beforehand via a selectable serial interface.

10.1 Start-Up Mode Selection

After any start-up the currently valid start-up configuration is indicated in bitfield HWCFG of register SCU_STSTAT. **Table 10-1** summarizes the defined start up modes.

A start-up configuration can be selected in two ways:

1. Via an externally applied hardware configuration upon a Power-on or Internal Application reset
   The hardware configuration is applied to Port 10 pins (P10.[3:0]).
   The hardware that activates a startup configuration during reset may be simple pull resistors for systems that use this feature upon every reset. You may want to use a switchable solution (via jumpers or an external signal) for systems that only temporarily use a hardware configuration.
2. By executing the following software sequence (using SCU_SWRSTCON and SCU_RSTCON1 registers):
   a) Write respective configuration value (refer to **Table 10-1**) to bitfield SCU_SWRSTCON.SWCFG;
   b) Assign desired type of reset to the software request trigger by writing into SCU_RSTCON1.SW bitfield (by default SCU_RSTCON1.SW=00_B meaning no reset generated by software request trigger)
   c) Set Software Boot Configuration bit: SCU_SWRSTCON.SWBOOT = 1;
   d) Trigger a software reset by activating Software Reset Request: SCU_SWRSTCON.SWRSTREQ = 1.

There is a differentiation of XE16x behavior in case of Application reset which must be noted:

- an Application reset triggered by hardware request (for example WDT, ESRx) does not cause evaluation of the P10 pins - the same start-up configuration is used as after the previous reset;
- an Application reset triggered by software (with setting SCU_SWRSTCON.SWRSTREQ=1) can have different consequences:
  – if Software Boot Configuration is selected (SCU_SWRSTCON.SWBOOT=1) - the start-up configuration is updated from SCU_SWRSTCON.SWCFG bitfield;

– otherwise - the same start-up configuration is used as after the previous reset.

Table 10-1 XE16x Start-Up Mode Configuration

Start-Up Mode	STSTAT.HWCFG Value [1]	Configuration Pins P10.[3:0] [2]			
Internal Start from Flash	$0000'0011_B$	x	x	1	1
Standard UART Bootloader mode	$0000'0110_B$	x	1	1	0
Enhanced UART Bootloader mode	$0000'0010_B$	x	0	1	0
CAN Bootloader mode	$0000'0101_B$	x	1	0	1
SSC Bootloader mode	$0000'1001_B$	1	0	0	1
External Start	$0000'0000_B$	0	0	0	0

1) Bitfield HWCFG can be loaded from Port 10 or from bitfield SWCFG in register SWRSTCON.

2) x means that the level on the corresponding pin is irrelevant.

10.2 Device Status after Start-Up

The main parameters of XE16x-status at the point of time when the first user instruction is executed are summarized below.

10.2.1 Registers modified by the Start-Up Procedure

Table 10-2 shows the XE16x registers which are initialized during the start-up procedure with values different from their reset-content (defined into respective register-descriptions).

There are two groups of registers regarding the way they are affected by start-up procedure:

1. registers initialized after any start-up;
2. registers initialized after start-up triggered by a power-on in DMP_1 power domain.

Note: Power-on in DMP_M domain means power-on also in DMP_1.

The registers in **Table 10-2** are grouped in accordance to the above differentiation.

Table 10-2 XE16x Registers installed by the Start-Up Procedure

Register	Value	Comments
1. After any start-up:		
TRAPDIS	$009F_H$	All SCU-controlled traps disabled except PET and RAT
RSTCON1	UU: **11**uu:U_H	Application Reset request generated by WDT
WDTCS	UUU: uuu**0**$_H$	Overflow Error Status is reset ; the original flag is saved in **STMEM0.WDTCSOE** (refer to **Page 10-4**)
TRAPDIS	$009F_H$	All SCU-controlled traps disabled except PET and RAT
PMTSR	0100_H	Parity Error Sensitivity Enabled
2. After power-on in DMP_1:		
PLLCON0	$0F00_H$	PLL in Normal Mode, N-divider = 16
PLLCON1	$000A_H$	Internal clock as input for PLL
PLLCON2	0000_H	K1-divider = 1
PLLCON3	8007_H	K2-divider = 8
SYSCON0	0002_H	The PLL output (f_{PLL}) used as system clock
WUOSCCON	0000_H	Wake up Oscillator enabled with f_{WU} approx. 500kHz
HPOSCCON	U:u**0**uu: UU_H	PLLSTAT.FINDIS bit will not be set in an OSCWDT emergency case
PLLOSCCON	$XXXX_H$	Device-specific value (chip-to-chip trimming)
EVRMCON0	0110_H	EVR_M Control 0 register
EVR1CON0	$0D10_H$	EVR_1 Control 0 register
EVRMCON1	0101_H	EVR_M Control 1 register
EVRMSET15VHP	$001B_H$	EVR_1 Setting for 1.5V HP register
EVR1SET15VHP	$001B_H$	EVR_1 Setting for 1.5V HP register
PVCMCON0	2544_H	PVC_M Control for Step 0 register
PVC1CON0	2544_H	PVC_1 Control for Step 0 register
SWDCON0	0941_H	SWD Control 0 register
EVRMSET10V	$005B_H$	EVR_M Setting for 1.0V register
EVRMSET15VLP	$00DD_H$	EVR_M Setting for 1.5V LP register
EVR1SET10V	$005B_H$	EVR_1 Setting for 1.0V register
EVR1SET15VLP	$00DD_H$	EVR_1 Setting for 1.5V LP register

Two additional points regarding register-content after start-up must be taken into account:

- The register-modifications shown in **Table 10-2** happen independently on the start-up mode currently selected, which means also in **Internal Start** mode.
  Next to these, in other modes - **External Start** and **Bootstrap Loading** (**Chapter 10.5**, **Chapter 10.6**) - more registers are additionally modified during start-up, as described into respective Specific Settings chapters for any of the modes.
- The values seen in some bits after start-up can be affected not only by the reset procedure itself but also by other events during and even before the last start-up - for example an Emergency Event can change the clock-system status.
  Therefore occasional exceptions are possible from the above values (as well as from the default register content after reset), mainly for some clock control/status flags. For more information on such special cases and their handling - refer to XC2000 Programmer's Guide.

10.2.2 System Frequency

The system clock which is active when the first user instruction is executed, depends on the currently selected start-up mode and the last start-up trigger:

- after power-on in all modes except CAN Bootstrap Loader (**Chapter 10.6.4**) - 10MHz (nominal value) from the XE16x internal oscillator (doubled frequency);
- after power-on in CAN Bootstrap Loader mode (**Chapter 10.6.4**) - the frequency of an external crystal connected to XTAL-pins, 4MHz minimum;
- after any functional (not power-on) reset - the clock system configuration is not changed by device start-up, respectively the system frequency remains as before the reset.

10.2.3 Watchdog Timer handling

The Watchdog Timer (WDT) in XE16x is always enabled by the start-up procedure and configured to generate Application Reset.

Therefore, the user software must:

- if WDT-usage is foreseen by the code - service it for a first time within approx. 65500 system clock cycles after start-up;
- otherwise - disable it within the same time frame as above but before to execute End of Init (EINIT).

Watchdog Timer Double Error

"Watchdog Timer Double Error" in XE16x means the following sequence of events:

1. WDT overflow happens - WDT rolls from $FFFF_H$ to 0000_H and enters Prewarning Mode;
2. device starts-up anew - could be due to WDT-reset (unavoidable once Prewarning Mode is entered) or another reset request triggered before Prewarning Mode been expired;
3. a following WDT-reset causes the next - after the start-up according to **2.** - device restart.

In other words, in this scenario WDT overflow happens two times one after another with one device restart in between.

Such a situation is considered as indicating major and systematic malfunction and the device enters **Start-up Error state** (refer to **Chapter 10.2.4**).

This (WDT Double Error) feature is supported in XE16x by **WDTCSOE** bit in **STMEM0** register (refer to **Page 10-7**) which bit is copied by the start-up procedure from SCU_WDTCS.OE (refer to **WDT Control and Status Register**, **Section 6.11.4.2**), SCU_WDTCS.OE is reset immediately afterwards. Next, upon any device restart the following conditions are checked by the start-up procedure:

- Is **STMEM0**.**WDTCSOE** bit set? AND
- Is the start-up caused by a WDT-reset?

If both the above conditions are true - a Watchdog Timer Double Error is recognized and **Start-up Error state** is immediately entered by the device.

Therefore, if entering power-save mode upon WDT Double Error is not desired for some reasons, the user software must care to reset **STMEM0**.**WDTCSOE** bit. Different possibilities for handling exist:

- reset the bit always at the beginning of application code or inside the WDT Prewarning Mode handler routine (if any) - in such a case WDT Double Error will be never recognized;
- reset the bit together with servicing the WDT - in such a case WDT Double Error will be recognized only if WDT has not been serviced at all after the previous overflow and device restart.

*Note: If after the first WDT overflow/device restart a second reset is triggered by another source but not WDT, the **STMEM0**.**WDTCSOE** bit will be automatically reset by the start-up procedure.*

10.2.4 Start-up Error state

To prevent possible negative consequences for the device and/or the system, upon unrecoverable error during startup XE16x is put onto a stable, passive and neutral to the external world state - power-save mode with DMP_1 shut down and DMP_M powered with 1V.

This state can be exited with power-on reset only.

10.3 Special Start-up Features

XE16x supports some special features, which allow the user software to influence the device start-up, providing additional functionality next to the above (in **Chapter 10.1**) described.

10.3.1 Supplementary Start-up Information from/to the User

The special start-up features require/provide additional information from/to the application software, using a dedicated register inside the System Control Unit - STMEM0.

STMEM0 Register

The SCU_STMEM0 register is located in DMP_M power-supply domain and is Security-protected.

The following start-up information can be exchanged with application software using this register:

1. the user software can influence the next device start-up by writing into STMEM0 bits[15,13:11];
   The supported feature (SRAM initialization) is described in **Chapter 10.3.3**.
2. the emergency-status flags indicated in SCU_SYSCON0 bits[15:12] upon device start-up can be read by user software from SCU_STMEM0 bits[3:0];
   The background here is that the start-up procedure itself could cause (for example due to clock-system reconfiguration) a change in some of these flags.
3. the handling of **Watchdog Timer Double Error** is supported by SCU_STMEM0 bit[4] - refer to **Page 10-4**;
4. the handling of “Flash not operable” scenario (refer to **Chapter 10.3.2**) is supported by SCU_STMEM0 bit[14] - refer to **Chapter 10.3.2**.

STMEM0
Start-up Memory 0 Register **ESFR ($F0A0_H/50_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
USSET	FNOP	RINDP	RINDS	RINPS	0						WDT CSOE	SEL STAT	EMS PWR	EMS VCO	EMS OSC
rw	rw	rw	rw	rw	r								rw		

Field	Bits	Typ	Description
EMSOSC	0	rw	**OSCWDT Emergency Event Source Status** [1]
EMSVCO	1	rw	**VCOLCK Emergency Event Source Status** [1]
EMSPWR	2	rw	**PVC1 Emergency Event Source Status** [1]
SELSTAT	3	rw	**Clock Select Status** [1]
WDTCSOE	4	rw	**Watchdog Timer Overflow Error status flag** [2]
0	[10:5]	r	**Reserved, do not change these bits**
RINPS	11	rw	**Initialization of the PSRAM:** 0 not requested 1 will be performed upon start-up
RINDS	12	rw	**Initialization of the DSRAM:** 0 not requested 1 will be performed upon start-up
RINDP	13	rw	**Initialization of the DPRAM:** 0 not requested 1 will be performed upon start-up
FNOP	14	r	**Flash operability:** 0 Flash operable 1 Flash not operable (no functional resets allowed)
USSET	15	rw	**RAM Initialization upon start-up:** 0 not requested 1 requested in STMEM0 [13:11]

1) Bits copied from SYSCON0 register upon startup.

2) Bit copied from WDTCS.OE upon startup.

10.3.2 Support for Power-saving Modes

XE16x allows several power-saving modes - for information about these modes and how they can be controlled by user software refer to XC2000 Programmer's Guide.

One rule regarding power-system handling in XE16x is: if a power-state is entered with DMP_1 supply below the minimum value at which the Flash is operable, this state must be exited only by power-on (wake-up from power-save mode) but not by a functional reset.

STMEM0.**FNOP** bit is defined to support application-compliance with this rule as follows:

- if **FNOP** is set upon device restart; AND
- this restart is caused by a functional (not power-on) reset

the start-up procedure terminates and device enters **Start-up Error state**.

Therefore the user software controlling the power-states must:

1. before to enter a power-state in which Flash is not operable - set **STMEM0**.**FNOP**;
2. after exiting this power-state (Flash not operable) by power-on in DMP_1 - reset **STMEM0**.**FNOP**.

*Note: After power-on in DMP_M the **FNOP** bit as the complete **STMEM0** register will be anyway reset.*

10.3.3 Preparing to activate Parity

XE16x supports parity as memory content protection mechanism, which can request trap or reset upon a single-bit data error (refer to **Memory Content Protection**, **Section 6.13**).

The user software must activate parity trap/reset generation for any one of PSRAM, DSRAM and DPRAM only after every location from the respective memory is written at least once after the last power reset. In other words, parity must be activated for a memory only after this memory has been initialized.

Note: The RAMs in USIC and MultiCAN modules implement special access mechanism, which assures a memory location will be read only after it has been written before, therefore for these memories no initialization is necessary.

The parity can be activated by user software using the sequence shown at **Figure 10-1**:

- if **STMEM0**[15]=0 after power-on in DMP_1 (indicated by SCU_RSTSTAT1.ST1=11_B) - RAM initialization is needed and not been requested:
  - optionally - if the application will run with system clock faster than 10MHz (system frequency after power-on) - the clock reconfiguration can be done still here to use increased speed for a faster RAM initialization;
  - install request for RAMs initialization by setting **STMEM0**[15:11]=10111_B;
    It is also possible to set selectively only some of the bits[13:11] corresponding to

the memories in which the parity will be activated (refer to **STMEM0**-description and **Figure 10-1**).
  – trigger application reset to cause a new device start-up
    During this new device start-up the RAMs are initialized as requested in **STMEM0**[13:11].
- if **STMEM0**[15]=1 after startup - meaning RAMs have been just initialized:
  – RAM-initialization request is cleared - **STMEM0**[15:11]=00000_B;
  – parity is configured/enabled as required by the application
- if **STMEM0**[15]=0 after functional reset (not power-on) - RAM initialization is not needed and the request is not active:
  – parity is configured as required by the application - this is needed because enable bits in SCU_PECON register are reset upon any start-up;
- continue with further system initialization (if any) and starting the application.

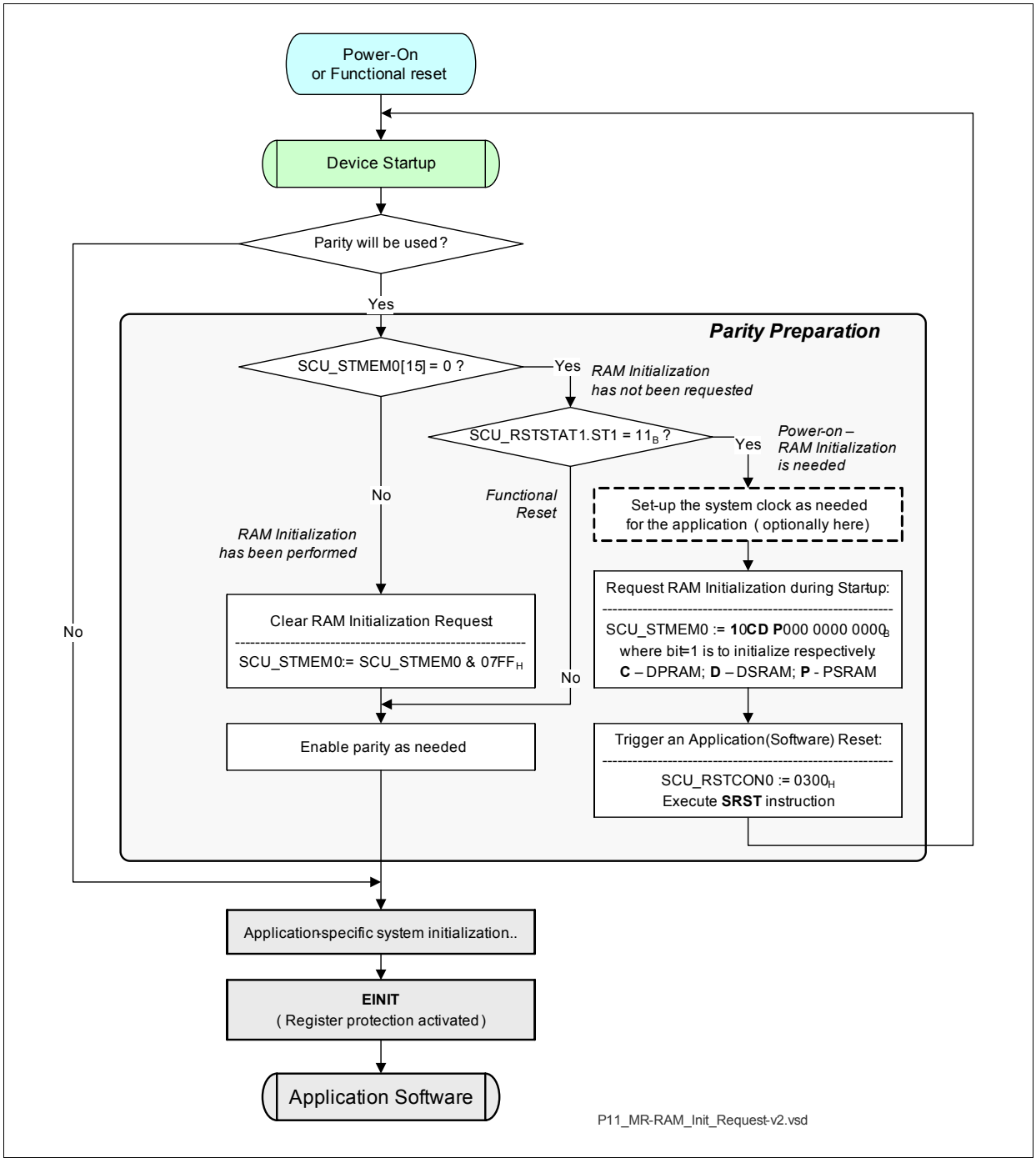


Figure 10-1 Software sequence to prepare Parity usage

10.4 Internal Start

When internal start mode is configured, the XE16x immediately begins executing code out of the on-chip Flash memory (first instruction from location C0'0000$_H$).

No additional configuration options are required, when selecting internal startup mode.

Note: Because internal start mode is expected to be the configuration used in most cases, this mode can be selected by pulling high just 2 pins.

10.5 External Start

When external start mode is configured, the XE16x begins executing code out of an off-chip memory (first instruction from location 00'0000$_H$), connected to the XE16x's external bus interface.

The External Bus Controller is adjusted to the employed external memory by evaluating additional configuration pins.

Seven pins of P10 are used to select the EBC mode (P10.[10:8]), the address width (P10.[12:11]), and the number of chip select lines (P10.[14:13]). The following tables summarize the available options.

Table 10-3 EBC Configuration: EBC Mode

EBC Startup Mode	Cfg. Pins P10[10:8]			Pins Used by the EBC
8-Bit Data, Multiplexed	0	0	0	P2.0 … P2.2, P10.0 … P10.15
8-Bit Data, Demultiplexed	0	0	1	P0.0 … P0.7, P1.0 … P1.7, P2.0 … P2.2, P10.0 … P10.7, P10.13, P10.14
16-Bit Data, MUX, $\overline{\text{BHE}}$ mode	0	1	0	P2.0 … P2.2, P2.11, P10.0 … P10.15
16-Bit Data, MUX, $\overline{\text{WRH}}$ mode	0	1	1	P2.0 … P2.2, P2.11, P10.0 … P10.15
16-Bit Data, DeMUX, $\overline{\text{BHE}}$ mode, A0	1	0	0	P0.0 … P0.7, P1.0 … P1.7, P2.0 … P2.2, P2.11, P10.0 … P10.14
16-Bit Data, DeMUX, $\overline{\text{WRH}}$ mode, A0	1	0	1	P0.0 … P0.7, P1.0 … P1.7, P2.0 … P2.2, P2.11, P10.0 … P10.14
16-Bit Data, DeMUX, $\overline{\text{BHE}}$ mode, A1	1	1	0	P2.0 … P2.2, P10.0 … P10.15
16-Bit Data, DeMUX, $\overline{\text{WRH}}$ mode, A1	1	1	1	P0.0 … P0.7, P1.0 … P1.7, P2.0 … P2.2, P10.0 … P10.7, P10.13, P10.14

Table 10-4 EBC Configuration: Address Width

Available Address Lines	Cfg. Pins P10[12:11]		Additional Address Pins
A15 … A0	0	0	None
A17 … A0	0	1	P2.3, P2.4
A19 … A0	1	0	P2.3 … P2.6
A23 … A0	1	1	P2.3 … P2.10

Table 10-5 EBC Configuration: Chip Select Lines

Available Chip Select Lines	Cfg. Pins P10[14:13]		Used Pins
$\overline{\text{CS0}}$ … $\overline{\text{CS4}}$	0	0	P4.0 … P4.4
CS0	0	1	P4.0
CS0 … CS1	1	0	P4.0, P4.1
None	1	1	None

10.5.1 Specific Settings

When the XE16x has entered External Start mode, the configuration is automatically set: according to **Table 10-6** and **Table 10-7**.

Note, that the startup procedure does not configure any address window within ADDRSELx registers. Therefore, even if some $\overline{CS}$ signal is configured (refer to **Table 10-5** and **Table 10-6**), the startup procedure only makes the proper settings to assure the adequate pin-functionality in regard to the selected EBC mode. The user software must take care:

- to configure the address window (in ADDRSELx register) for the $\overline{CSx}$ pin(s) which will be used;
- to enable those pins by setting FCONCSx.ENCS.

Table 10-6 External start mode-Specific State in EBC Registers

Configuration at P10[10:8]	EBCMOD0 [15:8]	EBCMOD1	FCONCSx [1]	Comment (EBC Mode)
000_B	30_H	$001F_H$	0011_H	8-Bit Multiplexed
001_B	70_H	0020_H	0001_H	8-Bit Demultiplexed
010_B	40_H	0000_H	0031_H	16-Bit MUX, $\overline{BHE}$
011_B	48_H	0000_H	0031_H	16-Bit MUX, $\overline{WRH}$
100_B	60_H	0000_H	0021_H	16-Bit DeMUX, $\overline{BHE}$, A0
101_B	61_H	0000_H	0021_H	16-Bit DeMUX, $\overline{WRH}$, A0
110_B	60_H	0010_H	0021_H	16-Bit DeMUX, $\overline{BHE}$, A1
111_B	61_H	0010_H	0021_H	16-Bit DeMUX, $\overline{WRH}$, A1

1) Which FCONCSx registers are affected is dependant on the configuration at P10[14:13] as follows:
11_B or 01_B - FCONCS0 is affected
10_B - FCONCS0 and FCONCS1 are affected
00_B - FCONCS0..FCONCS4 are affected
The other (unaffected) FCONCS registers retain their default values - refer to **Section 9.3.5**.

Table 10-7 External start mode-Specific State in EBCMOD0[7:0]

Configuration at P10[14:13]	Configuration at P10[12:11]			
	00_B (0 Segm.)	**01_B** (2 Segm.)	**10_B** (4 Segm.)	**11_B** (8 Segm.)
00_B (5 CS)	50_H	52_H	54_H	58_H
01_B (1 CS)	10_H	12_H	14_H	18_H
10_B (2 CS)	20_H	22_H	24_H	28_H
11_B (0 CS)	00_H	02_H	04_H	08_H

10.6 Bootstrap Loading

Bootstrap Loading is the technique of transferring code to the XE16x via a certain interface (usually serial) before the regular code execution out of non-volatile program memory commences. Instead, the XE16x executes the previously received code.

This boot-code may be complete (e.g. temporary software for testing or calibration), amend existing code in non-volatile program memory (e.g. with product-specific data or routines), or load additional code (e.g. using higher or more secure protocols). A possible application for bootstrap loading is the programming of virgin Flash memory at the end of a production line, with no external memory or internal Flash required for the initialization code.

The BSL mechanism may be used for standard system startup as well as only for special occasions like system maintenance (firmware update) or end-of-line programming or testing.

The XE16x supports bootstrap loading using several protocols/modes:

- Standard UART protocol, loading 32 bytes (see **Section 10.6.2.1**)
- UART protocol, Enhanced bootstrap loader transferring arbitrary number of bytes (see **Section 10.6.2.2**)
- Synchronous serial protocol (see **Section 10.6.3**)
- CAN protocol (see **Section 10.6.4**)

For a summary of these modes, see also **Table 10-14**

10.6.1 General Functionality

Even though each bootstrap loader has its particular functionality, the general handling is the same for all of them.

Entering a Bootstrap Loader

Bootstrap loaders are enabled by selecting a specific start-up configuration (see **Section 10.1**).

The required configuration patterns are described in **Table 10-14** for the bootstrap loaders, and are summarized in **Table 10-1**.

Loading the Startup Code

After establishing communication, the BSL enters a loop to receive the respective number of bytes. These bytes are stored sequentially into the on-chip PSRAM, starting at location E0'0000$_H$. To execute the loaded code the BSL then points register VECSEG to location E0'0000$_H$, i.e. the first loaded instruction, and then jumps to this instruction.

The loaded code may be the final application code or another, more sophisticated, loader routine that adds a transmission protocol to enhance the integrity of the loaded code or data. It may also contain a code sequence to change the system configuration and enable the bus interface to store the received data into external memory.

This process may go through several iterations or may directly execute the final application.

Note: Data fetches from a protected Flash will not be executed.

Exiting Bootstrap Loader Mode

The watchdog timer and the debug system are disabled as long as the Bootstrap loader is active. Watchdog timer and debug system are released automatically when the BSL terminates after having received the last byte from the host.

If 2nd level loaders are used, the loader routine should deactivate the watchdog timer via instruction DISWDT to allow for an extended download period.

The XE16x will start executing out of user memory as externally configured after a non-BSL reset .

Interface to the Host

The bootstrap loader communicates with the external host over a predefined set of interface pins. These interface pins are automatically enabled and controlled by the bootstrap loader. The host must connect to these predefined interface pins.

Table 10-14 indicates the interface pins that are used in each bootstrap loader mode.

10.6.2 Bootstrap Loaders using UART Protocol

XE16x users have different possibilities to download code/data in which the communication is based on UART (Universal Asynchronous Receiver and Transmitter) protocol.

10.6.2.1 Standard UART Bootstrap Loader

The standard UART bootstrap loader transfers program code/data via channel 0 of USIC0 (U0C0) into the PSRAM. The U0C0 receiver is only enabled after the identification byte has been transmitted. A half duplex connection to the host is, therefore, sufficient to feed the BSL.

Data is transferred from the external host to the XE16x using asynchronous eight-bit data frames without parity (1 start bit, 1 stop bit). The number of data bytes to be received in standard UART boot mode is fixed to 32 bytes, which allows up to 16 two-byte instructions.

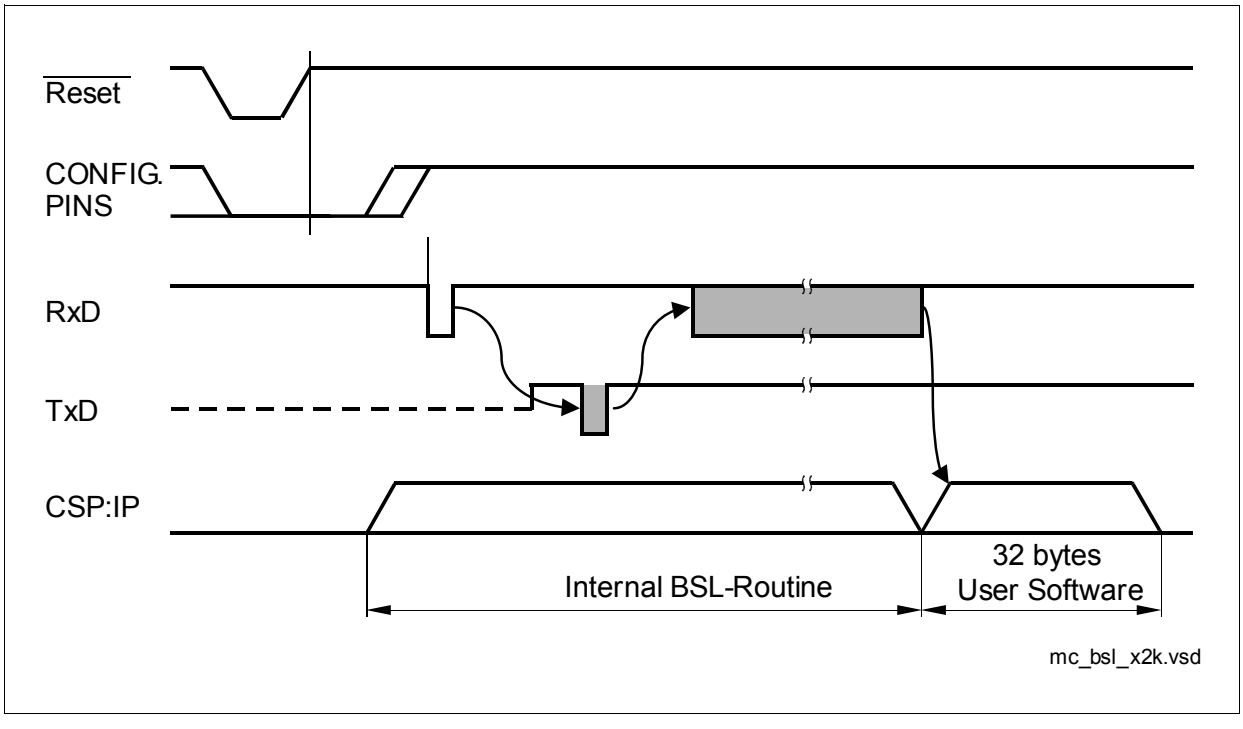


Figure 10-2 Bootstrap Loader Sequence

The XE16x scans the RxD line to receive a zero byte after entering UART BSL mode and the respective initialization. The zero byte is considered as containing one start bit, eight 0 data bits and one stop bit. From the duration of this zero byte it calculates the corresponding baudrate factor with respect to the current CPU clock, initializes the serial interface U0C0 accordingly and switches pin TxD to output. Using this baudrate, an identification byte ($D5_H$) is returned to the host that provides the loaded data.

Once the identification byte is transmitted, the BSL enters a loop to receive 32 bytes via U0C0. These bytes are stored sequentially into locations E0'0000$_H$ through E0'001F$_H$ of the internal PSRAM and then executed.

Note: For loading more code, two possibilities exist:
- via a 2nd-level loader - see below
*- using the **Enhanced UART Bootstrap Loader** - refer to **Section 10.6.2.2***

Second Level Bootloader

Most probably the initially loaded routine will load additional code or data, as an average application is likely to require substantially more instructions than could fit into 32 bytes. This second receive loop may directly use the pre-initialized interface U0C0 to receive data and store it to arbitrary user-defined locations.

The example code below shows how to fit such a 2nd-level loader into the available 32 bytes. This is possible due to the pre-initialized serial channel and the pre-set registers (see **Table 10-8**).

```
;Example for Secondary UART Bootstrap Loader Routine
;---------------------------------------------------------------
TargetStart LIT  '0E00020H'      ;Definition of target area:
TargetEnd   LIT  '0E001FFH'      ;480 bytes in this example
StartOfCode LIT  '0E00100H'      ;Continue executing here...
                                 ;...after download
Level2Loader:
     DISWDT                      ;No WDT for further download
     MOV   DPP0,#(PAG  TargetStart)
     MOV   R10, #(DPP0:TargetStart);Set pointer to target area
Level2MainLoop:
     MOV   [R1],R3               ;Clear RIF for new byte
Level2RecLoop:
     MOV   R4,  [R0]             ;Access PSR
     JNB   R4.14,Level2RecLoop   ;Wait for RIF
     MOVB  [R10],[R2]            ;Copy new byte to target
     CMPI1 R10,  #POF (TargetEnd);All bytes received??
     JMPR  cc_NE,Level2MainLoop  ;Repeat for complete area
Level2Terminate:
     JMPS   SEG StartOfCode, SOF StartOfCode
```

Specific Settings

The following configuration is automatically set when the XE16x has entered Standard UART BSL mode:

Table 10-8 Standard UART BSL-Specific State

Item	Value	Comments
U0C0_CCR	0002_H	ASC mode selected for USIC0 Channel 0
U0C0_PCRL	0401_H	1 stop bit, three RxD-samples at point 4
U0C0_SCTRL	0002_H	Passive data level = 1
U0C0_SCTRH	0707_H	8 data bits
U0C0_FDRL	$43FF_H$	Normal divider mode 1:1 selected
U0C0_BRGH	$0XXX_H$	Measured PDIV value (zero-byte) in bits[9:0]
U0C0_BRGL	$1C00_H$	Normal mode, FDIV, 8 clocks/bit
U0C0_DX0CR	0003_H	Data input selection
DPP1	0081_H	Points to USIC0 base address 1)
R0	4044_H	Pointer to U0C0_PSR 1)
R1	4048_H	Pointer to U0C0_PSCR 1)
R2	$405C_H$	Pointer to U0C0_RBUF 1)
R3	4000_H	Mask to clear RIF 1)
Devices in 144/100-pin package:		
P7_IOCR03	$00B0_H$	P7.3 is push/pull output (TxD)
P7_IOCR04	0020_H	P7.4 is input with pull-up (RxD)
Devices in 64-pin package:		
P2_IOCR03	$00B0_H$	P2.3 is push/pull output (TxD)
P2_IOCR04	0020_H	P2.4 is input with pull-up (RxD)

1) This register setting is provided for a 2nd-level loader routine (see at Page 10-17).

The identification byte identifies the device to be booted. The following codes are defined:

55_H: 8xC166.
$A5_H$: Previous versions of the C167 (obsolete).
$B5_H$: Previous versions of the C165.
$C5_H$: C167 derivatives.
$D5_H$: All devices equipped with identification registers (including the XE16x).

Note: The identification byte $D5_H$ does not directly identify a specific derivative. This information can, in this case, be obtained from the identification registers.

10.6.2.2 Enhanced UART Bootstrap Loader

The enhanced UART bootstrap loader transfers program code/data via Channel 0 of USIC0 Module (U0C0) into PSRAM.

Data is transferred from the external host to the XE16x using asynchronous eight-bit data frames without parity (1 start bit, 1 stop bit). The length of the code/data is not fixed as in the **Standard UART Bootstrap Loader** but can be arbitrary up to the PSRAM total size minus 256 bytes. Also the code execution can start from arbitrary PSRAM address, as well as the initial baudrate can be changed - e.g. increased for faster transfer of long code/data blocks.

The initial steps of this bootloader are the same as of the **Standard UART Bootstrap Loader**. XE16x first scans the RxD line to receive a zero byte, i.e. one start bit, eight 0 data bits and one stop bit. From the duration of this zero byte it calculates the corresponding baudrate factor with respect to the current CPU clock, initializes the serial interface U0C0 accordingly and switches pin TxD to output. Using this baudrate, an identification byte (DA_H) is returned to the host.

The next steps in this mode are to process the so-called Bootloader Header as follows:

1. XE16x sends the current PDIV divider from U0C0_BRGH register - the 10-bit value is sent in 2 bytes

Note: In this bootloader, the multi-byte values are sent in high-to-low order.

2. XE16x receives and sends back to the host a Header_Code (1B)
3. XE16x receives and sends back to the host number of bytes to be transferred Code_Length (3B)
   - the allowed range for this number is between 1 and the PSRAM size for the device minus 256 bytes
4. XE16x receives and sends back to the host the start address STADD for code-execution (3B)
   - the segment address (highest STADD byte) must equal $E0_H$ for XE16x
5. XE16x receives and sends back to the host a value for PDIV divider (2B, bits[9:0] effective only)
   - if the new value is different from the current - the new one is written into U0C0_BRGH register and a zero confirmation byte is sent back to the host with baudrate already changed
6. XE16x receives and sends back to the host a Trailer_Code (1B)
   a) if both the Header_Code and Trailer_Code are equal to the XE16x identification byte (DA_H) - the Bootloader sends to the Host a zero byte and continues further;
   b) if the above condition is not true - the Bootloader sends an identification byte (DA_H) to the host and restarts Header processing again from point **1.**

Once the Header is successfully processed according to the above steps, the Bootstrap loader receives Code_Length bytes and stores them sequentially starting from the beginning of PSRAM at address E0'0000$_H$.

The Bootstrap loader starts code-execution after the last byte is received and stored. The execution is started from address STADD as received within the header.

Specific Settings

The following configuration is automatically set when the XE16x has entered Enhanced UART BSL mode:

Table 10-9 Enhanced UART BSL-Specific State

Item	Value	Comments
U0C0_CCR	0002$_H$	ASC mode selected for USIC0 Channel 0
U0C0_PCRL	0401$_H$	1 stop bit, three RxD-samples at point 4
U0C0_SCTRL	0002$_H$	Passive data level = 1
U0C0_SCTRH	0707$_H$	8 data bits
U0C0_FDRL	43FF$_H$	Normal divider mode 1:1 selected
U0C0_BRGH	0XXX$_H$	PDIV-value as sent by the host inside header
U0C0_BRGL	1C00$_H$	Normal mode, FDIV, 8 clocks/bit
U0C0_DX0CR	0003$_H$	Data input selection
Devices in 144/100-pin package:		
P7_IOCR03	00B0$_H$	P7.3 is push/pull output (TxD)
P7_IOCR04	0020$_H$	P7.4 is input with pull-up (RxD)
Devices in 64-pin package:		
P2_IOCR03	00B0$_H$	P2.3 is push/pull output (TxD)
P2_IOCR04	0020$_H$	P2.4 is input with pull-up (RxD)

The identification byte identifies the device to be booted. XE16x is the first microcontroller family supporting Enhanced UART BSL mode, the code defined for it is DA$_H$.

Note: The identification byte does not directly identify a specific derivative. This information can, in this case, be obtained from the identification registers.

10.6.2.3 Choosing the Baudrate for the BSL

The calculation of the serial baudrate for U0C0 from the length of the first zero byte that is received, allows the operation of the bootstrap loader of the XE16x with a wide range of baudrates. However, the upper and lower limits have to be kept, in order to ensure proper data transfer.

The XE16x uses bitfield PDIV to measure the length of the initial zero byte. The quantization uncertainty of this measurement implies the deviation from the real baudrate.

For a correct data transfer from the host to the XE16x the maximum deviation between the internal initialized baudrate for U0C0 and the real baudrate of the host should be below 2.5%. The deviation (F_B, in percent) between host baudrate and XE16x baudrate can be calculated via **Equation (10.1)**:

$$F_B = \left| \frac{B_{Contr} - B_{Host}}{B_{Contr}} \right| \times 100\% \qquad F_B \leq 2.5\% \tag{10.1}$$

Note: Function (F_B) does not consider the tolerances of oscillators and other devices supporting the serial communication.

This baudrate deviation is a nonlinear function depending on the system clock and the baudrate of the host. The maxima of the function (F_B) increase with the host baudrate due to the smaller baudrate prescaler factors and the implied higher quantization error (see **Figure 10-3**).

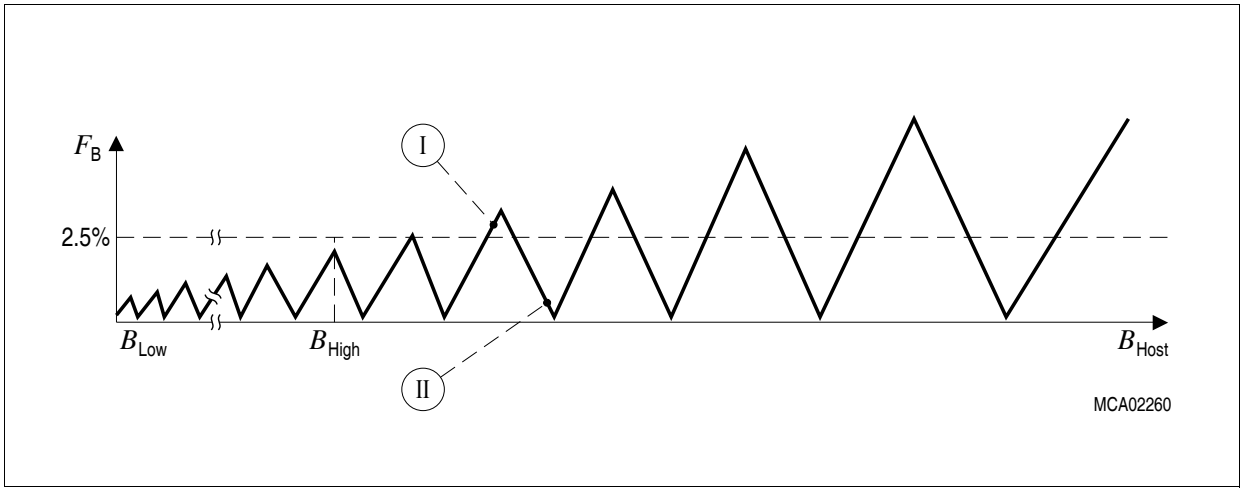


Figure 10-3 Baudrate Deviation between Host and XE16x

The minimum baudrate (B_{Low} in **Figure 10-3**) is determined by the maximum count capacity of bitfield PDIV, when measuring the zero byte, i.e. it depends on the system clock. The minimum baudrate is obtained by using the maximum PDIV count 2^{10} in the baudrate formula. Baudrates below B_{Low} would cause PDIV to overflow. In this case U0C0 cannot be initialized properly and the communication with the external host is likely to fail.

The maximum baudrate (B_{High} in **Figure 10-3**) is the highest baudrate where the deviation still does not exceed the limit, i.e. all baudrates between B_{Low} and B_{High} are below the deviation limit. B_{High} marks the baudrate up to which communication with the external host will work properly without additional tests or investigations.

Higher baudrates, however, may be used as long as the actual deviation does not exceed the indicated limit. A certain baudrate (marked I) in **Figure 10-3**) may e.g. violate the deviation limit, while an even higher baudrate (marked II) in **Figure 10-3**) stays very well below it. Any baudrate can be used for the bootstrap loader provided that the following three prerequisites are fulfilled:

- the baudrate is within the specified operating range for U0C0
- the external host is able to use this baudrate
- the computed deviation error is below the limit.

Note: When the bootstrap loader mode is entered after a power reset, the bootstrap loader will begin to operate with $f_{SYS} = f_{IOSC} \times 2$ (approximately 10 MHz) which will limit the maximum baudrate for U0C0.
Higher levels of the bootstrapping sequence can then switch the clock generation mode in order to achieve higher baudrates for the download.

10.6.3 Synchronous Serial Channel Bootstrap Loader

The Synchronous Serial Channel (SSC) bootstrap loader transfers program code/data from an external serial EEPROM via channel 0 of USIC0 (U0C0) into the PSRAM. The XE16x is the master, so no additional elements (except for the EEPROM) are required.

The SSC bootstrap loading is a convenient way for initial and basic (go/fail) testing during software development - it allows many various code-versions to be easy started on the target system by re-programming a serial EEPROM.

During SSC bootstrap loading data is transferred from the external EEPROM to the XE16x using synchronous eight-bit data frames with MSB first. The number of data bytes to be received in SSC boot mode is user-selectable. The serial clock rate is set to $f_{SYS}/10$, which results in 1 MHz after a power reset.

Once SSC BSL mode is entered and the respective initialization done, the XE16x first reads the header from the first addresses (00...0) of the target EEPROM.
This header consists of two items:

- The memory identification byte: $D5_H$
- The data size field: 1 byte or 2 bytes, depending on the EEPROM's addressing mode (8-bit or 16-bit, see **Section 10.6.3.1**)

If both items are valid the BSL enters a loop to read the number of bytes defined by the data size field (maximum is FF_H or $FF00_H$, depending on the EEPROM) via U0C0.

These bytes are stored sequentially into PSRAM starting at location $E0'0000_H$ and are then executed. Therefore, the size of the PSRAM in the respective derivative determines the real maximum block size to be downloaded.

An invalid header (identification byte $\neq \underline{D5_H}$, data size field = 0 or greater than $65280/FF00_H$) is indicated by toggling the $\overline{CS}$ line low 3 times. This helps debugging during the system setup phase.

10.6.3.1 Supported EEPROM Types

The XE16x's SSC bootstrap loader assumes an SPI-compatible EEPROM (25xxx series). It supports devices with 8-bit addressing as well as with 16-bit addressing. The connected EEPROM type is determined by examining the received header bytes, as indicated in **Table 10-10**.

Table 10-10 Determining the EEPROM Type

SSC Frame Number	Meaning of Transmitted Data	Received Data from 8-bit Addr. Device	Received Data from 16-bit Addr. Device
1	03_H: Read command	XX_H (default level)	XX_H (default level)
2	00_H: Address byte (high for 16-bit addr.)	XX_H (default level)	XX_H (default level)
3	00_H: Address byte low	$D5_H$: Identification byte	XX_H (default level)
4	00_H: Dummy byte	Size **n** in bytes	$D5_H$: Identification byte
5	00_H: Dummy byte	Data byte 1	Size **n** in bytes, MSB
6	00_H: Dummy byte	Data byte 2	Size **n** in bytes, LSB
7	00_H: Dummy byte	Data byte 3	Data byte 1
…	00_H: Dummy byte	Data byte 4 … **n** n=1..FF_H	Data byte 2 … **n** n=1..$FF00_H$

Note: The value of the returned default bytes (indicated as XX_H) depends on the employed EEPROM type.

10.6.3.2 Specific Settings

When the XE16x has entered the SSC BSL mode, the following configuration is automatically set:

Table 10-11 SSC BSL-Specific State

Item	Value	Comments
U0C0_CCR	0001_H	SSC mode selected for USIC0 Channel 0
U0C0_PCRL	0011_H	SSC master mode, frequency from fPPP
U0C0_PCRH	8000_H	MCLK generation is enabled
U0C0_SCTRL	0103_H	MSB first, passive data level=1
U0C0_SCTRH	$073F_H$	8 data bits, infinite frame
U0C0_DX0CR	0015_H	Data input selection
U0C0_FDRL	$43FF_H$	Normal divider mode 1:1 selected
U0C0_BRGL	0000_H	Normal mode, FDIV - default value after reset
U0C0_BRGH	8004_H	Passive levels MCLK/SCLK=0, PDIV=4
P2_IOCR03	$00D0_H$	P2.3 is open-drain output (MTSR)
P2_IOCR04	0020_H	P2.4 is input with pull-up (MRST)
P2_IOCR05	$00D0_H$	P2.5 is open-drain output (SCLK)
P2_IOCR06	$00C0_H$	P2.6 is open-drain output (SLS)

10.6.4 CAN Bootstrap Loader

The CAN bootstrap loader transfers program code/data via node 0 of the MultiCAN module into the PSRAM. Data is transferred from the external host to the XE16x using eight-byte data frames. The number of data frames to be received is programmable and determined by the 16-bit data message count value DMSGC.

The communication between XE16x and external host is based on the following three CAN standard frames:

- Initialization frame - sent by the external host to the XE16x
- Acknowledge frame - sent by the XE16x to the external host
- Data frame(s) - sent by the external host to the XE16x

The initialization frame is used in the XE16x for baud rate detection. After a successful baud rate detection is reported to the external host by sending the acknowledge frame, data is transmitted using data frames. **Table 10-12** shows the parameters and settings for the three utilized CAN standard frames.

Note: The CAN bootstrap loader requires a point-to-point connection with the host, i.e. the XE16x must be the only CAN node connected to the network. A crystal with at least 4 MHz is required for CAN bootstrap loader operation.

Initialization Phase

The first BSL task is to determine the CAN baud rate at which the external host is communicating. Therefore the external host must send initialization frames continuously to the XE16x. The first two data bytes of the initialization frame must include a 2-byte baud rate detection pattern (5555_H), an 11-bit (sent in 2 bytes) 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(s).

The CAN baud rate is determined by analyzing the received baud rate detection pattern (5555_H) and the baud rate registers of the MultiCAN module are set accordingly. The XE16x is now ready to receive CAN frames with the baud rate of the external host.

Acknowledge Phase

In the acknowledge phase, the bootstrap loader waits until it receives the next correctly recognized initialization frame from the external host, and acknowledges this frame by generating a dominant bit in its ACK slot. Afterwards, the bootstrap loader transmits an acknowledge frame back to the external host, indicating that it is now ready to receive data frames. The acknowledge frame uses the message identifier ACKID that has been received with the initialization frame.

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.

To summarize: the external host must send initialization frames (the content as above defined) continuously until an acknowledge frame is received back from the XE16x having the same message identifier as sent by the host in data bytes 2/3 from the initialization frame, then the **Data Transmission Phase** begins.

Data Transmission Phase

In the data transmission phase, data frames are sent by the external host and received by the XE16x. The data frames use the 11-bit data message identifier DMSGID that has been sent with the initialization frame. Eight data bytes are transmitted with each data frame. The first data byte is stored in PSRAM at $E0'0000_H$. 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 bootstrap loader finishes and executes the loaded code.

Timing Parameters

There are no general restrictions for CAN timings of the external host. During the initialization phase the external host transmits initialization frames. If no acknowledge frame is sent back within a certain time as defined in the external host (e.g. after a dedicated number of initialization frame transmissions), the external host can decide that the XE16x is not able to establish the CAN boot communication link.

Table 10-12 CAN Bootstrap Loader Frames

Frame Type	Parameter	Description
Initialization Frame	Identifier	11-bit, don't care
	DLC = 8	Data length code, 8 bytes within CAN frame
	Data bytes 0/1	Baud rate detection pattern (5555_H)
	Data bytes 2/3	Acknowledge message identifier ACKID (complete register contents)
	Data bytes 4/5	Data message count DMSGC, 16-bit
	Data bytes 6/7	Data message identifier DMSGID (complete register contents)
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 within CAN frame
	Data bytes 0/1	Contents of bit-timing register
	Data bytes 2/3	Copy of acknowledge identifier from initialization frame

Table 10-12 CAN Bootstrap Loader Frames (cont'd)

Frame Type	Parameter	Description
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 within CAN frame
	Data bytes 0 to 7	Data bytes, assigned to increasing destination (PSRAM) addresses

10.6.4.1 Specific Settings

When the XE16x has entered the CAN BSL mode, the following configuration is automatically set:

Table 10-13 CAN BSL-Specific State

Item	Value	Comments
P2_IOCR05	$00A0_H$	P2.5 is push/pull output (TxD)
P2_IOCR06	0020_H	P2.6 is input with pull-up (RxD)
SCU_HPOSCCON	0030_H	OSC_HP enabled, External Crystal/Clock mode
SCU_SYSCON0	0001_H	OSC_HP selected as system clock
CAN_MOCTR0L	0008_H	Message Object 0 Control, low
CAN_MOCTR0H	$00A0_H$	Message Object 0 Control, high
CAN_MOCTR1L	0000_H	Message Object 1 Control, low
CAN_MOCTR1H	$0F28_H$	Message Object 1 Control, high
CAN_MOFCR1H	0400_H	Message Object Function Control, high
CAN_MOAMR0H	$1FFF_H$	Message Object 0 - Acceptance Mask bit set
CAN_NPCR0	0003_H	Data input selection

10.6.5 Summary of Bootstrap Loader Modes

This table summarizes the external hardware provisions that are required to activate a bootstrap loader in a system.

Table 10-14 Configuration Data for Bootstrap Loader Modes

Bootstrap Loader Mode	Configuration on P10.[3:0] [1]	Receive Line from Host	Transmit Line to Host	Transferred Data	Supported Host Speed
Standard UART	$x110_B$	RxD = P7.4 (100/144-pin)	TxD = P7.3 (100/144-pin)	32 bytes	2.4 - 19.2 kbaud
		RxD = P2.4 (64-pin)	TxD = P2.3 (64-pin)		
Enhanced UART	$x010_B$	RxD = P7.4 (100/144-pin)	TxD = P7.3 (100/144-pin)	l bytes [2]	2.4-19.2 kbaud at start, then changeable by Header
		RxD = P2.4 (64-pin)	TxD = P2.3 (64-pin)		
Sync. Serial	1001_B	MRST = P2.4	MTSR = P2.3 SCLK = P2.5 SLS = P2.6	m bytes [3]	--- (controlled by XE16x)
MultiCAN	$x101_B$	RxDC0 = P2.6	TxDC0 = P2.5	8 × n bytes [4]	125 - 500 kBaud

1) x means that the level on the corresponding pin is irrelevant.

2) l = Code_Length sent by the host, the values allowed are 1...(PSRAM_size-256).

3) m = data size read from EEPROM, the values allowed are 1...(PSRAM_size-256).

4) n = DMSGC, Data Message Count sent by the host with Initialization frame, the values allowed are 1... (PSRAM_size-256)/8.

11 Debug System

The XE16x includes an On-Chip Debug Support (OCDS) system, which provides convenient debugging, controlled directly by an external tool via debug interface pins.

On-Chip Debug Support (OCDS)

The OCDS system supports a broad range of debug features including breakpoints and tracing memory locations. Typical application of the OCDS is to debug the user software running on the XE16x in a real time system environment.

The OCDS system is controlled by an external tool via the JTAG **Debug Interface** and an optional break interface with one or two pins (**Figure 11-1**). The break interface supports very low latency triggers between XE16x and tool and/or system environment if needed. The memory mapped OCDS registers are accessible via the JTAG interface using Cerberus. In addition there is a limited set of special Cerberus debug IO instructions. As an alternative the OCDS can be controlled by a debug monitor program, which communicates with the tool over a user interface like CAN. The OCDS system interacts with the core through an injection interface to allow execution of Cerberus-generated instructions, and through a break port.

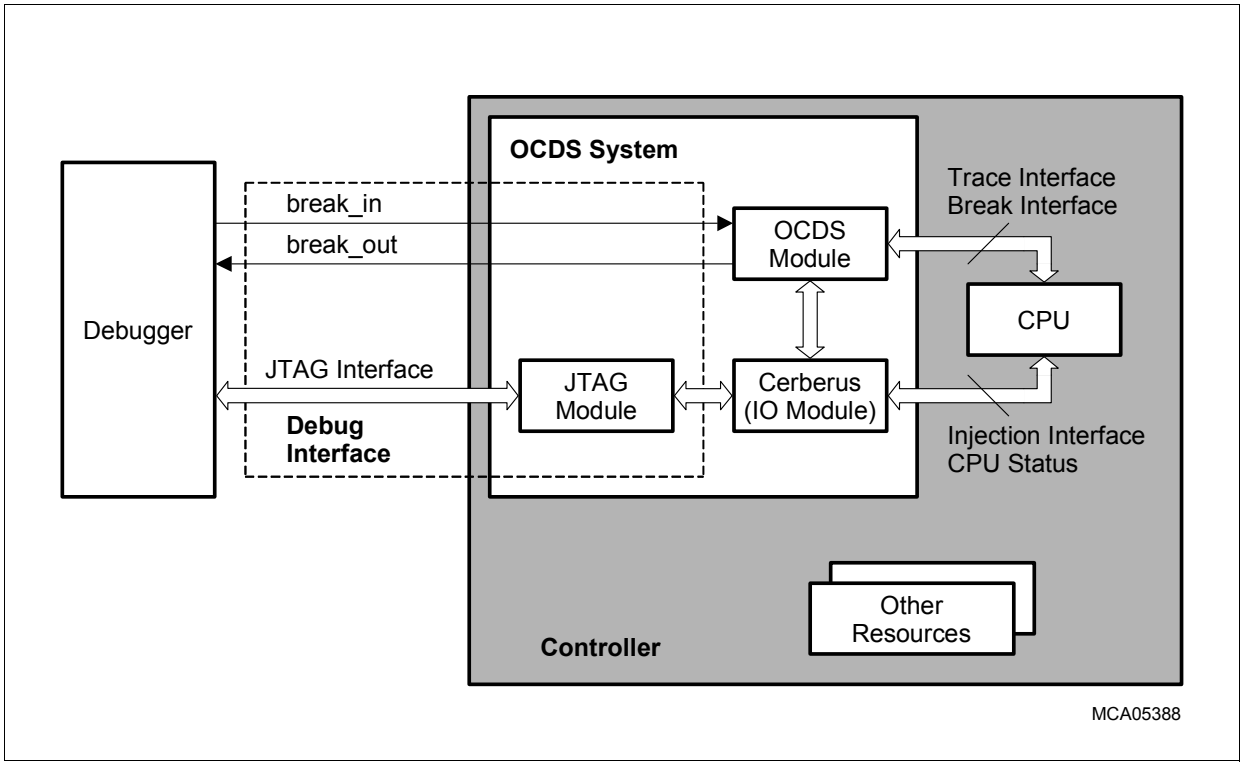


Figure 11-1 OCDS Overall Structure

The OCDS system consists of the three components **Debug Interface**, **OCDS Module** and **Cerberus**.

OCDS System Features

- Hardware, software and external pin breakpoints
- Reaction on break with CPU-Halt, monitor call, data transfer and external signal
- Read/write access to the whole address space
- Single stepping
- Debug Interface pins for JTAG interface and break interface
- Injection of arbitrary instructions
- Fast memory tracing through transfer to external bus
- Analysis and status registers

11.1 Debug Interface

The Debug Interface is a channel to access OCDS resources. Through it data can be transferred to/from all on- and off-chip (if any) memories and memory mapped control registers.

Features and Functions

- Independent interface for OCDS
- JTAG port based on the IEEE 1149.1-2001 JTAG standard
- Break interface for external trigger input and signaling of internal triggers
- Generic memory access functionality
- Independent data transfer channel for e.g. programming of flash memory

The Debug Interface is represented by:

- Standard **JTAG Interface** with 4 pins
- Two optional trigger pins - **OCDS Break-Interface**

Note: The JTAG clock frequency must be below the current CPU frequency.

JTAG Interface

The JTAG interface is a standardized and dedicated port usually used for boundary scan and for chip internal tests. Because both of these applications are not enabled during normal operation of the device in a system, the JTAG port is an ideal interface for debugging tasks.

This interface holds the JTAG IEEE.1149.1-2001 standard signals:

- **TDI** - Serial data input
- **TDO** - Serial data output
- **TCK** - JTAG clock
- **TMS** - State machine control signal
- **TRST** - Reset/Module enable

OCDS Break-Interface

Two additional signals are used to implement a direct asynchronous-break channel between the Debugger and XE16x **OCDS Module**:

- **BRKIN** (BReaK IN request) allows the Debugger asynchronously to interrupt the CPU and force it to a predefined status/action.
- **BRKOUT** (BReaK OUT signal) can be activated by OCDS to notify the external world that some predefined debug event has happened.

11.1.1 Routing of Debug Signals

The signals used to connect an external debugger via the JTAG interface and the break interface usually conflict with the requirements of the application, which needs as many IO pins as possible. In the XE16x, these signals are only provided as alternate functions (no dedicated pins). To minimize the impact caused by the debug interface pins, these signals can be mapped to several pins. Thus, each application can select the variant with the least impact. This is controlled via the Debug Pin Routing Register DBGPRR. Pin BRKOUT can be assigned to pins P6.0, P10.11, P1.5, or P9.3 as a standard alternate output signal via the respective IOC register.

11.1.1.1 Register DBGPRR

This register controls the pin mapping of the JTAG pins.

DBGPRR
Debug Pin Routing Register **ESFR ($F06E_H$/37_H)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRSTL	**0**					**DPRBRKIN**		**DPRTCK**		**DPRTMS**		**DPRTDI**		**DPRTDO**	
rh	r					rw		rw		rw		rw		rw	

Field	Bits	Type	Description
DPRTDO	[1:0]	rw	**Debug Pin Routing for TDO** 00 P7.0 01 P10.12 10 Reserved, do not use 11 Reserved, do not use

Field	Bits	Type	Description
DPRTDI	[3:2]	rw	**Debug Pin Routing for TDI** 00 P5.2 01 P10.10 10 P7.2 11 P8.3
DPRTMS	[5:4]	rw	**Debug Pin Routing for TMS** 00 P5.4 01 P10.11 10 P7.3 11 P8.4
DPRTCK	[7:6]	rw	**Debug Pin Routing for TCK** 00 P2.9 01 P10.9 10 P7.4 11 P8.5
DPRBRKIN	[9:8]	rw	**Debug Pin Routing for $\overline{\text{BRKIN}}$** 00 P5.10 01 P10.8 10 P7.1 11 P8.6
TRSTL	15	rh	**$\overline{\text{TRST}}$ Pin Start-up Value** This bit indicates if the Debug Mode can be entered or not. 0 A debugger can not be connected 1 A debugger can be connected
0	[14:10]	r	**Reserved** read as 0; should be written with 0.

Note: Make sure that the selected pins are available to the debug system at all times. In particular, do not enable power save behavior (see register POCON) for these pins.

11.2 OCDS Module

The application of the OCDS Module is to debug the user software running on the CPU in the customer's system. This is done with an external debugger that controls the OCDS Module via the independent **Debug Interface**.

Features

- Hardware, software and external pin breakpoints
- Up to 4 instruction pointer breakpoints
- Masked comparisons for hardware breakpoints
- The OCDS can also be configured by a monitor
- Support of multi CPU/master system
- Single stepping with monitor or CPU halt
- PC is visible in halt mode (IO_READ_IP instruction injection via **Cerberus**)

Basic Concept

The on chip debug concept is split up into two parts. The first part covers the generation of debug events and the second part defines what actions are taken when a debug event is generated.

- Debug events:
  - **Hardware Breakpoints**
  - Decoding of a **SBRK Instruction**
  - **Break Pin Input** activated
- Debug event actions:
  - **Halt Mode** of the CPU
  - **Call a Monitor**
  - **Trigger Transfer**
  - **Activate External Pin** Output

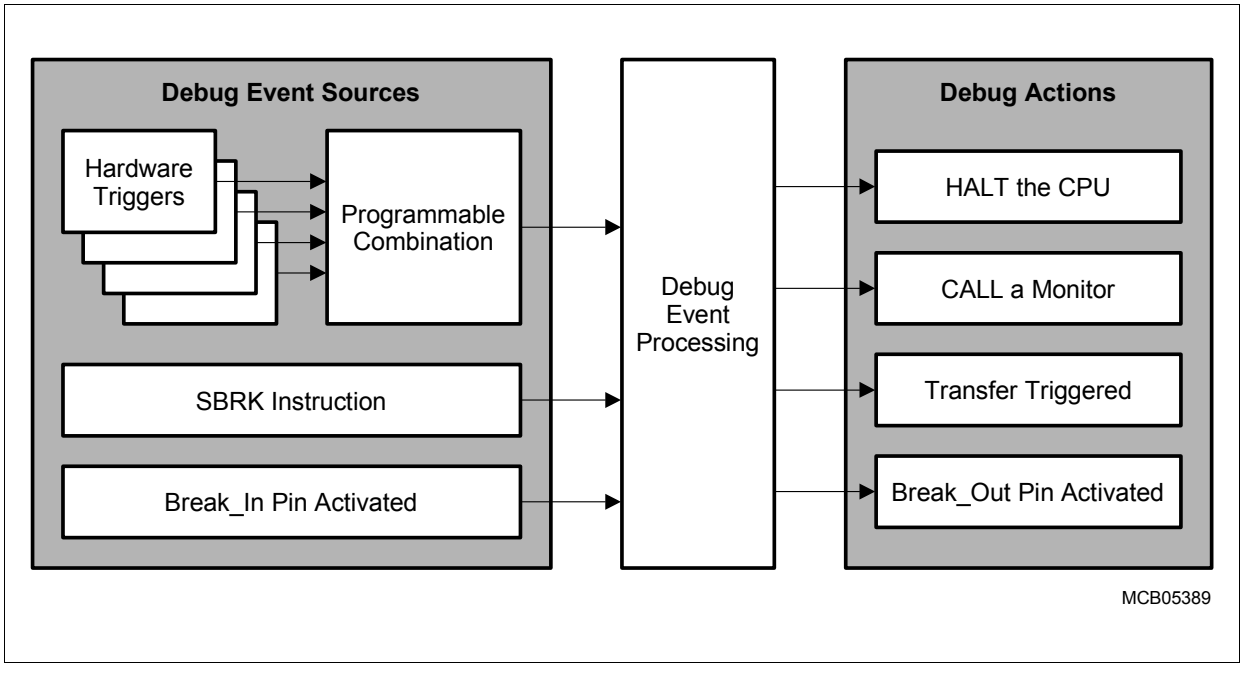


Figure 11-2 OCDS Concept: Block Diagram

11.2.1 Debug Events

The Debug Events can come from a few different sources.

Hardware Breakpoints

The Hardware Breakpoint is a debug-event, raised when a single or a combination of multiple trigger-signals are matching with the programmed conditions.

The following hardware trigger sources can be used:

Table 11-1 Hardware Triggers

Trigger Source	Size
Task Identifier	16 bits
Instruction Pointer	24 bits
Data address of reads (two busses monitored)	2 × 24 bits
Data address of writes	24 bits
Data value (reads or writes)	16 bits

SBRK Instruction

This is a mechanism through which the software can explicitly generate a debug event. It can be used for instance by a debugger to temporarily patch code held in RAM in order to implement **Software Breakpoints**.

A special SBRK (Software BReaK) instruction is defined with opcode 0x8C00. When this instruction has been decoded and it reaches the Execute stage, the whole pipeline is canceled including the SBRK itself. Hence in fact the SBRK instruction is never “executed” by itself.

The further behavior is dependent on how OCDS has been programmed:

- if the OCDS is enabled and the software breakpoints are also enabled, then the CPU goes into **Halt Mode**
- if the OCDS is disabled or the software breakpoints are disabled, then the Software Break Trap (SBRKTRAP) is executed-Class A Trap, number 08_H

Break Pin Input

An external debug break pin ($\overline{\text{BRKIN}}$) is provided to allow the debugger to asynchronously interrupt the processor.

11.2.2 Debug Actions

When the OCDS is enabled and a debug event is generated, one of the following actions is taken:

Trigger Transfer

One of the actions that can be specified to occur on a debug event being raised is to trigger the **Cerberus**:

- to execute a Data Transfer - this can be used in critical routines where the system cannot be interrupted to transfer a memory location
- to inject an instruction to the Core - using this mechanism, an arbitrary instruction can be injected into the XE16x pipeline

Halt Mode

Upon this Action the OCDS Module sends a Break-Request to the Core.

The Core accepts this request, if the OCDS Break Level is higher than current CPU priority level. In case a Break-Request is accepted, the system suspends execution with halting the instruction flow.

The Halt Mode can be still interrupted by higher priority user interrupts. It then relies on the external debugger system to interrogate the target purely through reading and updating via the debug interface.

Call a Monitor

One of the possible actions to be taken when a debug event is raised is to call a Monitor Program.

This short entry to a Monitor allows a flexible debug environment to be defined which is capable of satisfying many of the requirements for efficient debugging of a real time system. In the common case the Monitor has the highest priority and can not be interrupted from any other requesting source.

It is also possible to have an Interruptible Monitor Program. In such a case safety critical code can be still served while the Monitor (Debugger) is active, which gives a maximum flexibility to the user.

Activate External Pin

This action activates the external pin $\overline{\text{BRKOUT}}$ of the **OCDS Break-Interface**. It can be used in critical routines where the system cannot be interrupted to signal to the external world that a particular event has happened. The feature could also be useful to synchronize the internal and external debug hardware.

11.3 Cerberus

Cerberus is the module which provides and controls all the operations necessary to interact between the external debugger (via the **Debug Interface**), the **OCDS Module** and the internal system of XE16x.

Features

- JTAG interface is used as control and data channel
- Generic memory read/write functionality (RW mode) with access to the whole address space
- Reading and writing of general-purpose registers (GPRs)
- Injection of arbitrary instructions
- External host controls all transactions
- All transactions are available at normal run time and in halt mode
- Priority of transactions can be configured
- Full support for communication between the monitor and an external host (debugger)
- Optional error protection
- Tracing memory locations through transferring values to the external bus
- Analysis Register for internal bus locking situations

The target application of Cerberus is to use the JTAG interface as an independent port for On Chip Debug Support. The external debugger can access the OCDS registers and arbitrary memory locations with the injection mechanism.

11.3.1 Functional Overview

Cerberus is operated by an external debugger across the **JTAG Interface**. The Debugger supplies Cerberus IO Instructions and performs bidirectional data-transfers.

The **Cerberus** distinguishes between two main modes of operation:

Read/Write Mode of Operation

Read/Write (RW) Mode is the most typical way to operate Cerberus. This mode is used to read and write memory locations or to inject instructions. The injection interface to the core is actively used in this mode.

In this mode an external Debugger (host), using JTAG Interface, can:

- read and write memory locations from the target system (data-transfer);
- inject arbitrary instructions to be executed by the Core.

All Cerberus IO Instructions can be used in RW mode. The dedicated IO_READ_IP instruction is provided in RW mode to read the IP of the CPU while in Break.

The access to any memory location is performed with injected instructions, as PEC transfer. The following Cerberus IO Instructions can be used in their generic meaning:

- IO_READ_WORD, IO_WRITE_WORD
- IO_READ_BLOCK, IO_WRITE_BLOCK
- IO_WRITE_BYTE

Within these instructions, the host writes/reads data to/from a dedicated register/memory, while the Cerberus itself takes care of the rest: to perform a PEC transfer by injection of the appropriate instructions to the Core.

Communication Mode of Operation

In this mode the external host (debugger) communicates with a program (Monitor) running on the CPU. The data-transfers are made via a PDBus+ register. The external host is master of all transactions, requesting the monitor to write or read a value.

The difference to **Read/Write Mode of Operation** is that the read or write request now is not actively executed by the Cerberus, but it sets request bits in a CPU accessible register to signal the Monitor, that the host wants to send (IO_WRITE_WORD) or receive (IO_READ_WORD) a value. The Monitor has to poll this status register and perform respectively the proper actions

Communication Mode is the default mode after reset. Only the IO_WRITE_WORD and IO_READ_WORD Instructions are effectively used in Communication Mode.

The Host and the Monitor exchange data directly with the dedicated data-register. For a synchronization of Host (Debugger) and Monitor accesses, there are associated control bits in a Cerberus status register.

11.4 Boundary-Scan

The XE16x eases board-level analysis in the application system by providing Boundary-Scan according to the IEEE standard 1149.1. It supports testing of the interconnections between several devices mounted on a PCB.

Boundary-Scan is accomplished via the JTAG module, using standard JTAG instructions (IEEE1149.1).

Note: For Boundary-Scan to operate properly, the JTAG interface must use the default pins. The reset value of register DBGPRR ensures this.

Initialization of Boundary-Scan

The following sequence is defined to activate Boundary-Scan mode:

- Set $\overline{\text{PORST}}$ = 1; $\overline{\text{TRST}}$ = 1; $\overline{\text{TESTM}}$ = 1
- Negative Pulse on $\overline{\text{PORST}}$
- Wait for Power Domain to startup.
- Negative pulse on $\overline{\text{TRST}}$ to reset the JTAG controller.

Now the test access port for Boundary-Scan is enabled. The Boundary-Scan test can be used for board test with instructions like PRELOAD and EXTEST.

12 Instruction Set Summary

This chapter briefly summarizes the XE16x's instructions ordered by instruction classes. This provides a basic understanding of the XE16x's instruction set, the power and versatility of the instructions and their general usage.

A detailed description of each single instruction, including its operand data type, condition flag settings, addressing modes, length (number of bytes) and object code format is provided in the **"Instruction Set Manual"** for the XE166 Family. This manual also provides tables ordering the instructions according to various criteria, to allow quick references.

Summary of Instruction Classes

Grouping the various instruction into classes aids in identifying similar instructions (e.g. SHR, ROR) and variations of certain instructions (e.g. ADD, ADDB). This provides an easy access to the possibilities and the power of the instructions of the XE16x.

Note: The used mnemonics refer to the detailed description.

Table 12-1 Arithmetic Instructions

Addition of two words or bytes:	ADD	ADDB
Addition with Carry of two words or bytes:	ADDC	ADDCB
Subtraction of two words or bytes:	SUB	SUBB
Subtraction with Carry of two words or bytes:	SUBC	SUBCB
16 × 16 bit signed or unsigned multiplication:	MUL	MULU
16/16 bit signed or unsigned division:	DIV	DIVU
32/16 bit signed or unsigned division:	DIVL	DIVLU
1's complement of a word or byte:	CPL	CPLB
2's complement (negation) of a word or byte:	NEG	NEGB

Table 12-2 Logical Instructions

Bitwise ANDing of two words or bytes:	AND	ANDB
Bitwise ORing of two words or bytes:	OR	ORB
Bitwise XORing of two words or bytes:	XOR	XORB

Table 12-3 Compare and Loop Control Instructions

Comparison of two words or bytes:	CMP	CMPB
Comparison of two words with post-increment by either 1 or 2:	CMPI1	CMPI2
Comparison of two words with post-decrement by either 1 or 2:	CMPD1	CMPD2

Table 12-4 Boolean Bit Manipulation Instructions

Manipulation of a maskable bit field in either the high or the low byte of a word:	BFLDH	BFLDL
Setting a single bit (to ‘1’):	BSET	–
Clearing a single bit (to ‘0’):	BCLR	–
Movement of a single bit:	BMOV	–
Movement of a negated bit:	BMOVN	–
ANDing of two bits:	BAND	–
ORing of two bits:	BOR	–
XORing of two bits:	BXOR	–
Comparison of two bits:	BCMP	–

Table 12-5 Shift and Rotate Instructions

Shifting right of a word:	SHR	–
Shifting left of a word:	SHL	–
Rotating right of a word:	ROR	–
Rotating left of a word:	ROL	–
Arithmetic shifting right of a word (sign bit shifting):	ASHR	–

Table 12-6 Prioritize Instruction

Determination of the number of shift cycles required to normalize a word operand (floating point support):	PRIOR	–

Table 12-7 Data Movement Instructions

Standard data movement of a word or byte:	MOV	MOVB
Data movement of a byte to a word location with either sign or zero byte extension:	MOVBS	MOVBZ

Note: The data movement instructions can be used with a big number of different addressing modes including indirect addressing and automatic pointer in-/ decrementing.

Table 12-8 System Stack Instructions

Pushing of a word onto the system stack:	PUSH	–
Popping of a word from the system stack:	POP	–
Saving of a word on the system stack, and then updating the old word with a new value (provided for register bank switching):	SCXT	–

Table 12-9 Jump Instructions

Conditional jumping to an either absolutely, indirectly, or relatively addressed target instruction within the current code segment:	JMPA	JMPI	JMPR
Unconditional jumping to an absolutely addressed target instruction within any code segment:	JMPS	–	–
Conditional jumping to a relatively addressed target instruction within the current code segment depending on the state of a selectable bit:	JB	JNB	–
Conditional jumping to a relatively addressed target instruction within the current code segment depending on the state of a selectable bit with a post-inversion of the tested bit in case of jump taken (semaphore support):	JBC	JNBS	–

Table 12-10 Call Instructions

Conditional calling of an either absolutely or indirectly addressed subroutine within the current code segment:	CALLA	CALLI
Unconditional calling of a relatively addressed subroutine within the current code segment:	CALLR	–
Unconditional calling of an absolutely addressed subroutine within any code segment:	CALLS	–
Unconditional calling of an absolutely addressed subroutine within the current code segment plus an additional pushing of a selectable register onto the system stack:	PCALL	–
Unconditional branching to the interrupt or trap vector jump table in code segment <VECSEG>:	TRAP	–

Table 12-11 Return Instructions

Returning from a subroutine within the current code segment:	RET	–
Returning from a subroutine within any code segment:	RETS	–
Returning from a subroutine within the current code segment plus an additional popping of a selectable register from the system stack:	RETP	–
Returning from an interrupt service routine:	RETI	–

Table 12-12 System Control Instructions

Resetting the XE16x via software:	SRST	–
Entering the Idle mode:	IDLE	–
No function, do not use[1]:	PWRDN	–
Servicing the Watchdog Timer:	SRVWDT	–
Disabling the Watchdog Timer:	DISWDT	–
Enabling the Watchdog Timer (can only be executed in WDT enhanced mode):	ENWDT	–
Signifying the end of the initialization routine (switches the register security mechanism to “protected” and disables the effect of any later execution of a DISWDT instruction in WDT compatibility mode):	EINIT	–

1) Instruction PWRDN is used to enter Power Down mode in previous 16-bit architectures. In the XE16x devices, however, PWRDN has no effect and is executed like a NOP instruction.

Table 12-13 Miscellaneous

Null operation which requires 2 Bytes of storage and the minimum time for execution:	NOP	–
Definition of an unseparable instruction sequence:	ATOMIC	–
Switch ‘reg’, ‘bitoff’ and ‘bitaddr’ addressing modes to the Extended SFR space:	EXTR	–
Override the DPP addressing scheme using a specific data page instead of the DPPs, and optionally switch to ESFR space:	EXTP	EXTPR
Override the DPP addressing scheme using a specific segment instead of the DPPs, and optionally switch to ESFR space:	EXTS	EXTSR

Note: The ATOMIC and EXT instructions provide support for uninterruptable code sequences e.g. for semaphore operations. They also support data addressing beyond the limits of the current DPPs (except ATOMIC), which is advantageous for bigger memory models in high level languages.*

Table 12-14 MAC-Unit Instructions

Multiply (and Accumulate):	CoMUL	CoMAC
Add/Subtract:	CoADD	CoSUB
Shift right/Shift left:	CoSHR	CoSHL
Arithmetic Shift right:	CoASHR	–
Load Accumulator:	CoLOAD	–
Store MAC register:	CoSTORE	–
Compare values:	CoCMP	–
Minimum/Maximum:	CoMIN	CoMAX
Absolute value:	CoABS	–
Rounding:	CoRND	–
Move data:	CoMOV	–
Negate accumulator:	CoNEG	–
Null operation:	CoNOP	–

Protected Instructions

Some instructions of the XE16x which are critical for the functionality of the controller are implemented as so-called Protected Instructions. These protected instructions use the maximum instruction format of 32 bits for decoding, while the regular instructions only use a part of it (e.g. the lower 8 bits) with the other bits providing additional information like involved registers. Decoding all 32 bits of a protected doubleword instruction increases the security in cases of data distortion during instruction fetching. Critical operations like a software reset are therefore only executed if the complete instruction is decoded without an error. This enhances the safety and reliability of a microcontroller system.

13 Device Specification

The device specification describes the electrical parameters of the device. It lists DC characteristics like input, output or supply voltages or currents, and AC characteristics like timing characteristics and requirements.

Other than the architecture, the instruction set, or the basic functions of the XE16x core and its peripherals, these DC and AC characteristics are subject to changes due to device improvements or specific derivatives of the standard device.

Therefore, these characteristics are not contained in this manual, but rather provided in a separate Data Sheet, which can be updated more frequently.

Please refer to the current version of the Data Sheet of the respective device for all electrical parameters.

Note: In any case the specific characteristics of a device should be verified, before a new design is started. This ensures that the used information is up to date.

The XE16x derivatives are shipped in several packages. **Figure 13-1** and **Figure 13-2** show the basic pin diagrams of the XE16x. They show the location of the different supply and IO pins. A detailed description of all the pins and their selectable functions can be found in the corresponding Data Sheet.

*Note: Not all alternate functions shown in **Figure 13-1** are supported by all derivatives. Please refer to the corresponding descriptions in the data sheets.*

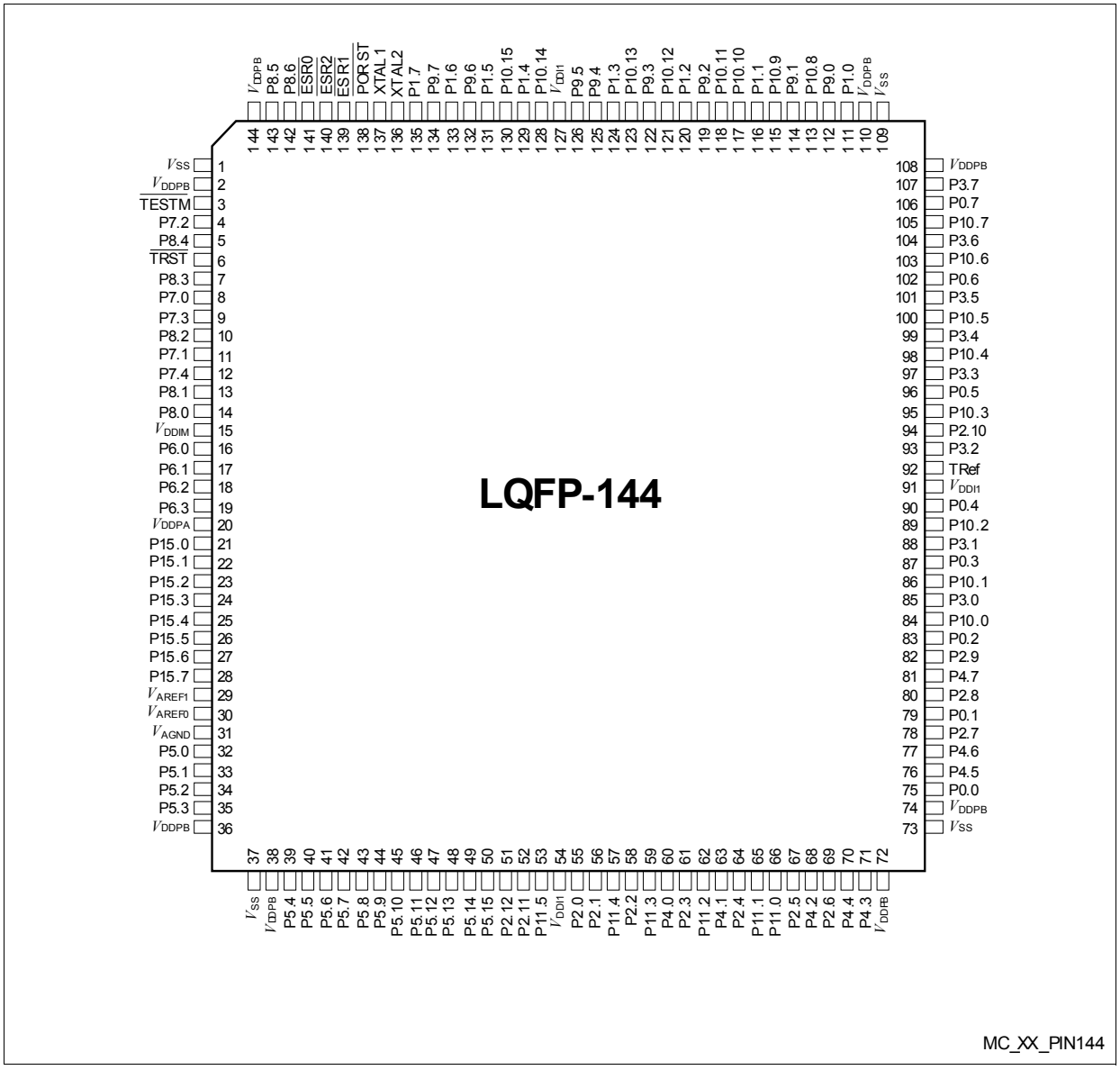


Figure 13-1 Pin Configuration PG-LQFP-144 Package (top view)

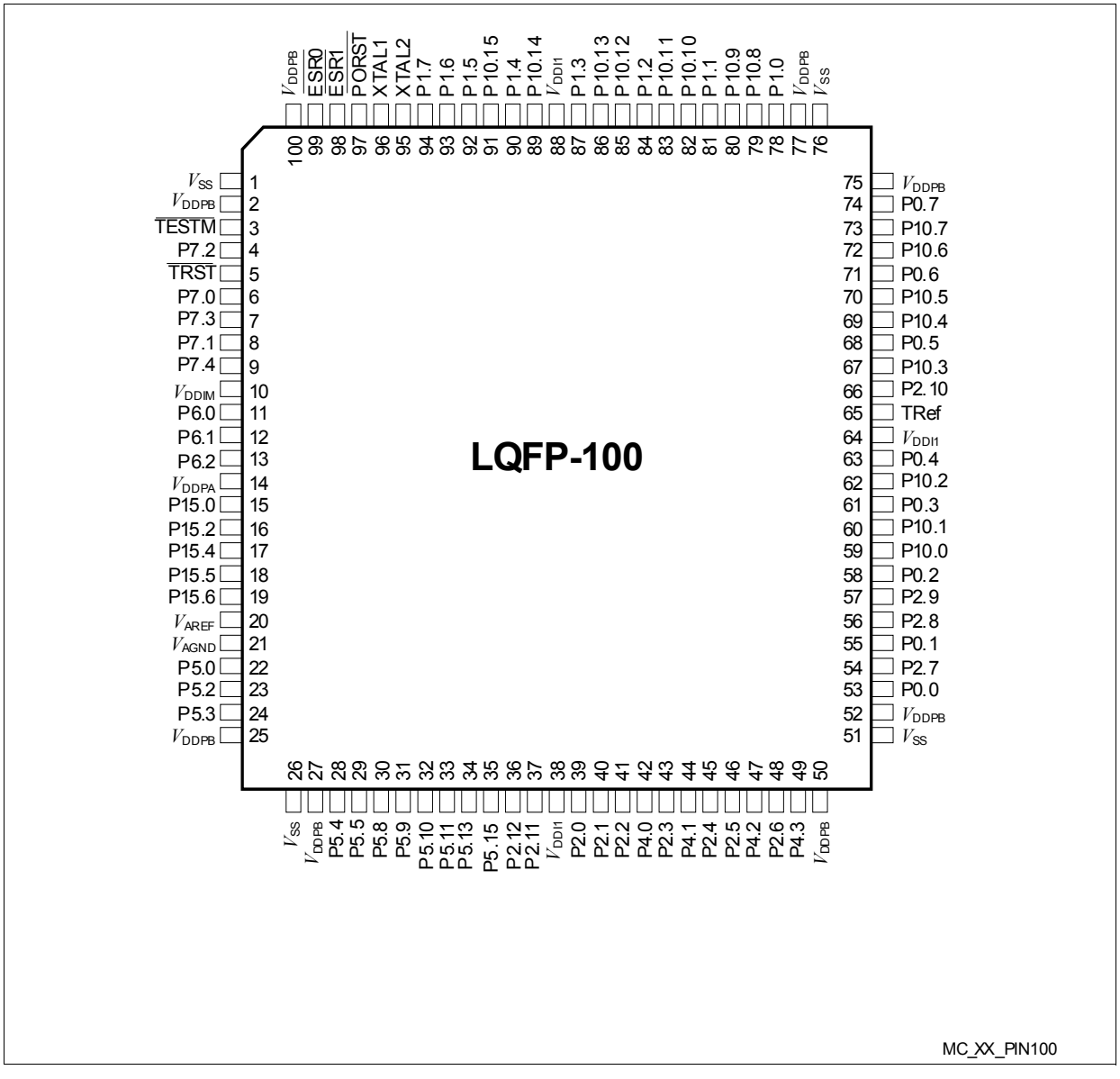


Figure 13-2 Pin Configuration PG-LQFP-100 Package (top view)

Keyword Index

This section lists a number of keywords which refer to specific details of the XE16x in terms of its architecture, its functional units or functions. This helps to quickly find the answer to specific questions about the XE16x.

This User's Manual consists of two Volumes, "System Units" and "Peripheral Units". For your convenience this keyword index refers to both volumes, so you can immediately find the reference to the desired section in the corresponding document ([1] or [2]).

*Note: Registers are listed in a separate index: **Register Index**.*

D

E

F

W

Register Index

This section lists the registers of the XE16x. This helps to quickly find the reference to the description of the respective register.

This User's Manual consists of two Volumes, "System Units" and "Peripheral Units". For your convenience this register index refers to both volumes, so you can immediately find the reference to the desired section in the corresponding document ([1] or [2]).

*Note: Keywords are listed in a separate index: **Keyword Index**.*

A

C

D

E

F

G

I

M

O

P

Q

R

S

T

U

V

X

Z

w w w . i n f i n e o n . c o m

B158-H9138-G1-X-7600

Published by Infineon Technologies AG