XC164CM

16-Bit Single-Chip Microcontroller with C166SV2 Core

Volume 1 (of 2): System Units

Microcontrollers

Never stop thinking

Edition 2006-03

Published by
Infineon Technologies AG
81726 München, Germany

© Infineon Technologies AG 2006.
All Rights Reserved.

Legal Disclaimer

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

Information

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

Warnings

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

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

XC164CM

16-Bit Single-Chip Microcontroller with C166SV2 Core

Volume 1 (of 2): System Units

Microcontrollers

Never stop thinking

XC164CM, Volume 1 (of 2): System Units
Revision History: V1.2, 2006-03

Previous Version(s):
V1.1, 2005-11
V1.0, 2005-06

Page	Subjects
Major Changes from V1.1, 2005-11 to V1.2, 2006-02	
9-1	Chapter name of the EBC module modified, “External” -> “LXBus”.
1-2	List of derivatives extended with new derivatives.
9-9	Section “LXBus Access Control and Signal Generation” contained only duplicate information. Therefore it is removed.
Major Changes from V1.0, 2005-06 to V1.1, 2005-11	
	General fixing of documentation bugs.
	Ports, CPU, and Architectural Overview chapters corrected not to erroneously refer to EBC external accesses, but only internal accesses to the LXBUS.
5-35	Fast External Interrupts: six available, references to EX6IN and EX7IN removed.
6-14	RSTCON register reset value description enhanced.
6-33	SYSSTAT.OSCSTAB bit added.
7-1 …	Ports chapter corrected: Port 0, Port 3.
3-24	“Interaction between Program Flash and Security Sector Programming” description added.

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

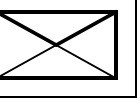


Table of Contents

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

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 increase in CPU performance and peripheral functionality over conventional 8-bit controllers. To achieve this high performance goal Infineon has decided to develop its families of 16-bit CMOS microcontrollers without the constraints of backward compatibility.

Nonetheless the architectures of the 16-bit microcontroller families pursue successful hardware and software concepts, which have been established in Infineon's popular 8-bit controller families.

About this Manual

This manual describes the functionality of a number of 16-bit microcontrollers of the Infineon XC166 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 and refer to the following derivatives:

Table 1-1 XC164xx Derivative Map

Derivative	xx	Number of Derivative Specific Modules		
		TwinCAN	ADC	CAPCOM6
XC164xx-8F 64 Kbytes Program FLASH 2 Kbytes DSRAM 2 Kbytes PSRAM 2 Kbytes DPRAM 2 × ASC, 2 × SSC 1 × GPT12E, 1 × CAPCOM2	**CM**	1	1	1
	GM	1	1	–
	SM	–	1	1
	TM	–	1	–
	KM	1	–	–
	LM	–	–	–
XC164xx-4F 32 Kbytes Program FLASH 0 Kbytes DSRAM 2 Kbytes PSRAM 2 Kbytes DPRAM 2 × ASC, 2 × SSC 1 × GPT12E, 1 × CAPCOM2	**CM**	1	1	1
	GM	1	1	–
	SM	–	1	1
	TM	–	1	–
	KM	1	–	–
	LM	–	–	–

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 **XC164CM** 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 XC164CM 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 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.

Introduction

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).

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 XC164CM, descriptions within this manual that refer to the "XC164CM" 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 XC164CM devices are enhanced members of the Infineon family of full featured 16-bit single-chip CMOS microcontrollers. The XC164CM combines the extended functionality and performance of the C166SV2 Core with powerful on-chip peripheral subsystems and on-chip memory units and provides a means for power reduction. Several key features contribute to the high performance of the XC164CM:

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

- Gated clock concept for improved power consumption and EMC
- Programmable system slowdown via clock generation unit
- Flexible management of peripherals, can be individually disabled
- Sleep-mode supports wake-up via fast external interrupts or on-chip RTC
- Programmable frequency output

Integrated On-Chip Memory

- 2 Kbytes Dual-Port RAM (DPRAM) for variables, register banks, and stacks
- Up to 2 Kbytes[1)] on-chip high-speed Data SRAM (DSRAM) for variables and stacks
- 2 Kbytes on-chip high-speed Program/Data SRAM (PSRAM) for code and data
- 64/32 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.

16-Priority-Level Interrupt System

- 80 interrupt nodes with separate interrupt vectors on 15 priority levels (8 group levels)
- 13 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

Intelligent On-Chip Peripheral Subsystems

- 14-channel A/D Converter with programmable resolution (10-bit or 8-bit) and conversion time (down to 2.55 µs or 2.15 µs), auto scan modes, channel injection
- 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}
- Capture/Compare Unit for flexible PWM Signal Generation (CAPCOM6)
  (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 $f_{SYS}/2$

1) Depends on the respective derivative. The derivatives are listed in **“About this Manual” on Page 1-2**.

- Two Asynchronous/Synchronous Serial Channels (USARTs)
  with baud rate generator, parity, framing, and overrun error detection,
  with auto baud rate detection, receive/transmit FIFOs, and IrDA support
- Two High Speed Synchronous Serial Channels (SPI-compatible)
  with programmable data length and shift direction
- Controller Area Network (TwinCAN) Module, Rev. 2.0B active, two 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 Loader for flexible system initialization
- Protection management for system configuration and control registers

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

Up to 47 IO Lines With Individual Bit Addressability

- Tri-stated in input mode
- Selectable input thresholds (not on all pins)
- Push/pull or open drain output mode
- Programmable port driver control
- I/O voltage is 5 V (core-logic and oscillator input voltage is 2.5 V)

Various Temperature Ranges[1)]

- -40 to +85 °C
- -40 to +125 °C

Infineon CMOS Process

- Low power CMOS technology enables power saving Idle, Sleep, and Power Down modes with flexible power management.

1) Not all derivatives are offered in all temperature ranges.

64-Pin Plastic Thin Quad Flat Pack (TQFP) Package

- PG-TQFP, 10 × 10 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, 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:

JTAG	Joint Test Access Group
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
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)
IO	Input/Output
LXBus	Internal representation of the external bus
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
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

1.4 Naming Conventions

The manifold bitfields used for control functions and status indication and the registers housing them are equipped with unique names wherever applicable. Thereby these control structured 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 ADC_CTR0 clearly identifies register CTR0 as part of module ADC, the term SYSCON1.CPSYS clearly identifies bitfield CPSYS as part of register SYSCON1.

2 Architectural Overview

The architecture of the XC164CM 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 XC164CM 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 XC164CM 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 XC164CM.

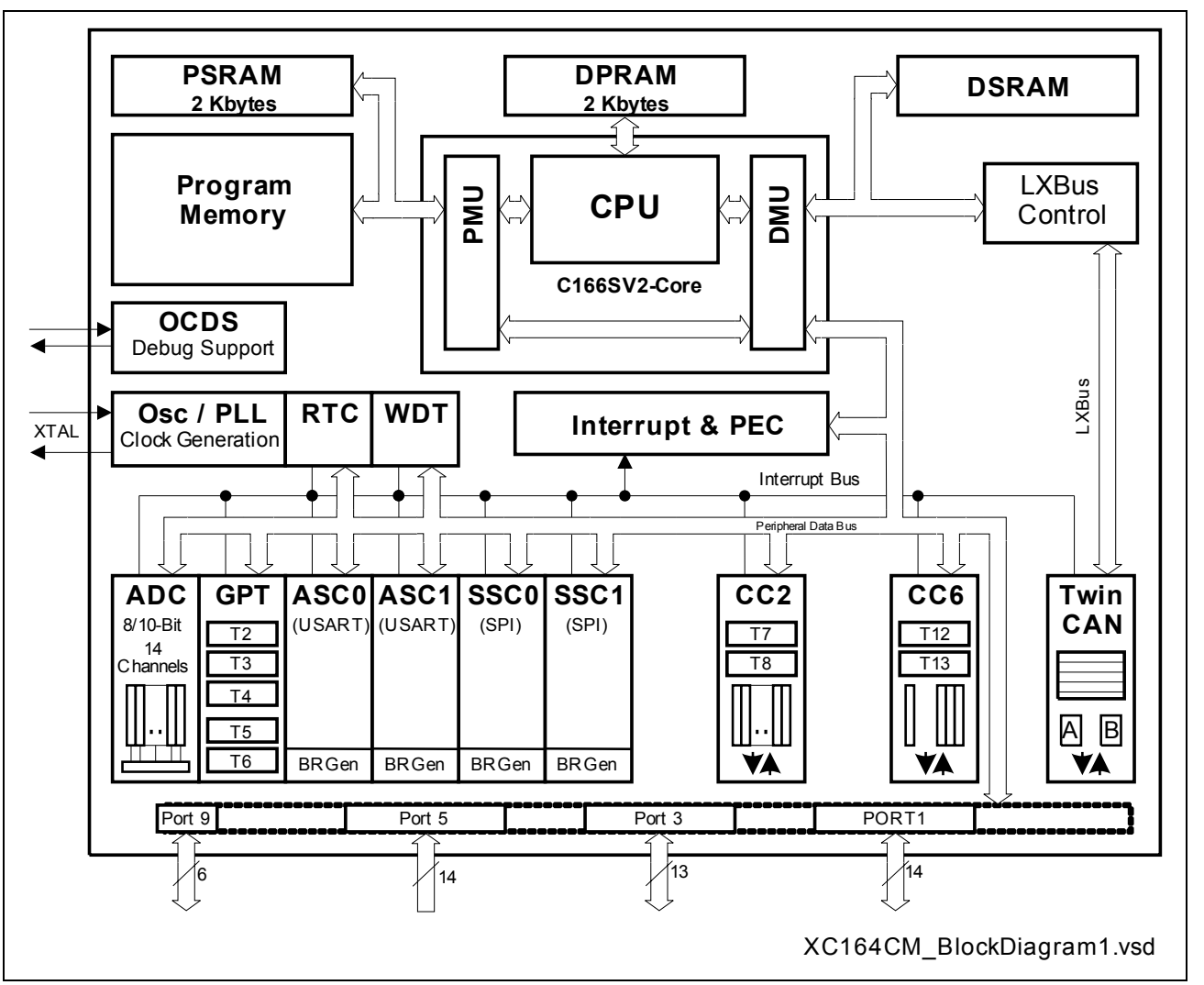


Figure 2-1 XC164CM Functional Block Diagram

The available program memory and DSRAM and the exact combination of the available peripherals depend on the respective derivative. The derivatives are listed on **Page 1-2**.

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.

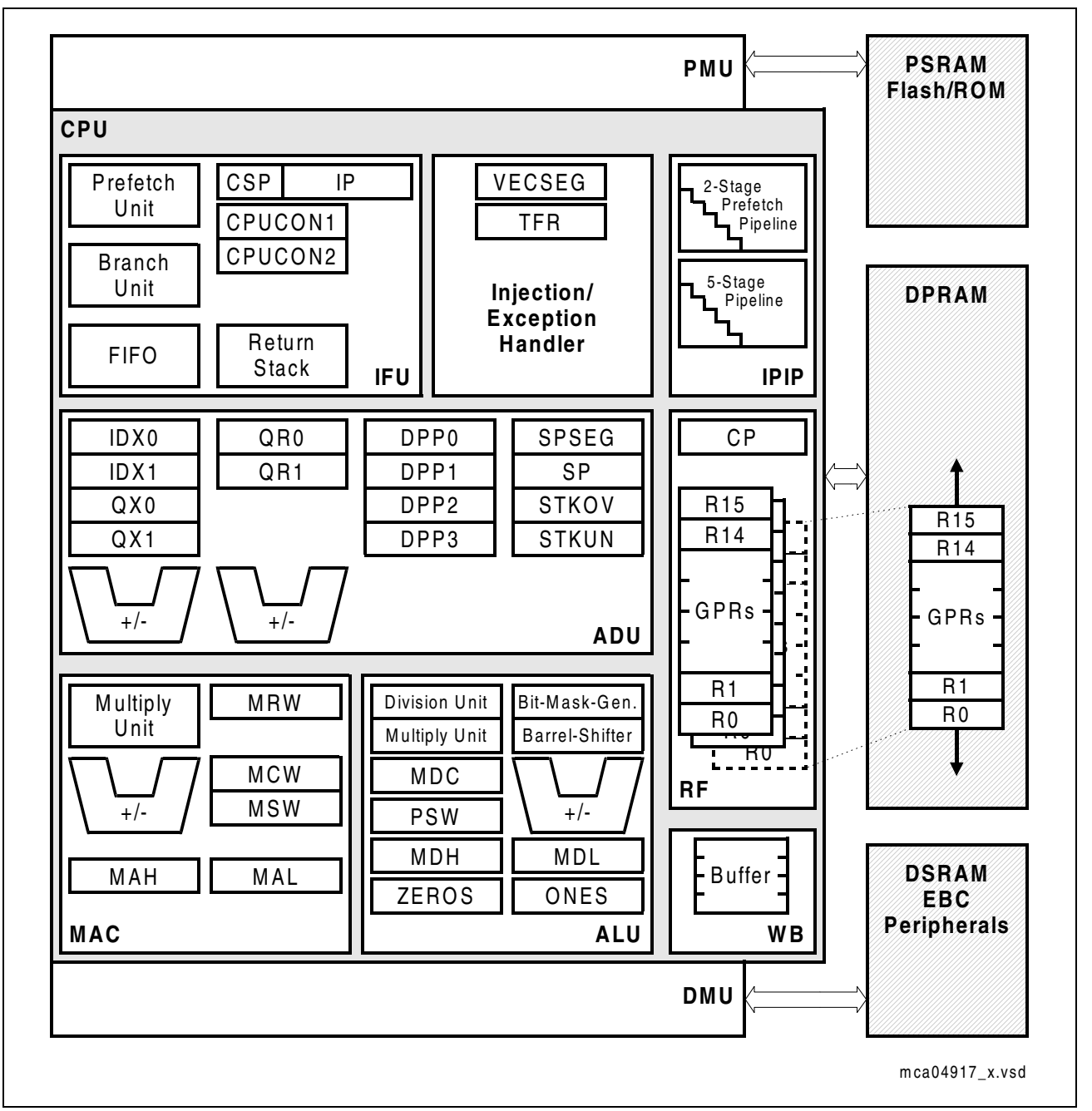


Figure 2-2 CPU Block Diagram

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 local register banks providing fast context switch
- Variable stack with automatic stack overflow/underflow detection
- “Fast interrupt” feature

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 XC164CM's instructions can be executed in just one clock cycle ($1/f_{CPU}$). This includes arithmetic instructions, logic instructions, and move instructions with most addressing modes.

Special instructions such as SRST or PWRDN 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 $\times$ 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 XC164CM 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 XC164CM provides 80 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 XC164CM 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 XC164CM is capable of reacting very quickly to non-deterministic events because its interrupt response time is within a very narrow range of typically 13 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 XC164CM 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 XC164CM 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).

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), and to the on-chip peripherals connected to the peripheral bus and including accesses to peripherals connected to the on-chip LXBus. The internal LXBus accesses 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 should be normally redirected via the DMU to the EBC. However, it must be taken into consideration that:
  - The XC164CM is a device that does not support accesses to external resources via the EBC.
  - 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 XC164CM 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 XC164CM has eight PEC channels, each of which offers such fast interrupt-driven data transfer capabilities.

Memory Areas

The memory space of the XC164CM 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.

64 Kbytes of on-chip Flash or ROM memory[1)] store code or constant data. The on-chip Flash memory is organized as four 8-Kbyte sectors, and one 32-Kbyte[1)] sector. Each sector can be separately write protected[2)], erased and programmed (in blocks of 128 bytes). The complete Flash area can be read-protected. A password sequence

1) Depends on the respective derivative. The derivatives are listed in **“About this Manual” on Page 1-2**.

2) Each two 8-Kbyte sectors are combined for write-protection purposes.

temporarily unlocks protected areas. The Flash module combines very fast 64-bit one-cycle[1] 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.
Programming typically takes 2 ms per 128-byte block (5 ms max.), erasing a sector typically takes 200 ms (500 ms max.).

The ROM is mask programmed at the factory.

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

2 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.

2 Kbytes of on-chip Data SRAM (DSRAM)[2] are provided as a storage for general user data.The DSRAM is accessed via the DMU 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) Flash accesses may require wait states, depending on the actual operating frequency.

2) Depends on the respective derivative. The derivatives are listed in **“About this Manual” on Page 1-2**.

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 XC166 Family with enhanced functionality. Therefore, they should either not be accessed, or written with zeros, to ensure upward compatibility.

Table 2-1 XC164CM Memory Map

Address Area	Start Loc.	End Loc.	Area Size[1)]	Notes
Flash register space	$FF'F000_H$	$FF'FFFF_H$	4 Kbytes	2)
Reserved (Acc. trap)	$F8'0000_H$	$FF'EFFF_H$	< 0.5 Mbytes	Minus Flash registers
Reserved for PSRAM	$E0'0800_H$	$F7'FFFF_H$	< 1.5 Mbytes	Minus PSRAM
Program SRAM	$E0'0000_H$	$E0'07FF_H$	2 Kbytes	Maximum
Reserved for program memory	$C1'0000_H$	$DF'FFFF_H$	< 2 Mbytes	Minus Flash
Program Flash	$C0'0000_H$	$C0'FFFF_H$	64 Kbytes	3)
Reserved	$40'0000_H$	$BF'FFFF_H$	8 Mbytes	–
Reserved	$20'0800_H$	$3F'FFFF_H$	< 2 Mbytes	Minus TwinCAN
TwinCAN registers	$20'0000_H$	$20'07FF_H$	2 Kbytes	Accessed via EBC
Reserved	$01'0000_H$	$1F'FFFF_H$	< 2 Mbytes	Minus segment 0
Data RAMs and SFRs	$00'8000_H$	$00'FFFF_H$	32 Kbytes	Partly used
Reserved	$00'0000_H$	$00'7FFF_H$	32 Kbytes	–

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

2) Not defined register locations return a trap code ($1E9B_H$).

3) Depends on the respective derivative. The derivatives are listed in **“About this Manual” on Page 1-2**.

2.3 On-Chip Peripheral Blocks

The XC166 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 XC164CM generic peripherals are:

- Two General Purpose Timer Blocks (GPT1 and GPT2)
- Two Asynchronous/Synchronous Serial Interfaces (ASC0 and ASC1)
- Two High Speed Serial Interfaces (SSC0 and SSC1)
- A Watchdog Timer
- One Capture/Compare unit (CAPCOM2)
- Enhanced Capture/Compare unit (CAPCOM6)
- A 10-bit Analog/Digital Converter (ADC)
- A Real Time Clock (RTC)
- Four I/O ports with a total of 47 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:

- TwinCAN module with 2 CAN nodes and gateway functionality

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.

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 master clock (f_{MC}). The clock generation unit uses the on-chip oscillator to derive the master clock from the crystal or from the external clock signal. The clock signal gated to the peripherals is independent from the clock signal that feeds the CPU. During Idle mode, the CPU’s clock is stopped while the peripherals continue their operation. 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 bit field 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.
- **Reserved Bits:** Some of the bits which are contained in the XC164CM'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 CAPCOM2 unit supports 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 CAPCOM2 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 the capture/compare register array.

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, an external count input for the CAPCOM timer T7 allows 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.
10 registers of the CAPCOM2 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 Unit CAPCOM6

The CAPCOM6 unit supports generation and control of timing sequences on up to three 16-bit capture/compare channels plus one independent 10-bit compare channel.
In compare mode the CAPCOM6 unit provides 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 (16-bit) and T13 (10-bit) 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.

General Purpose Timer (GPT12E) Unit

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 over-flow/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 CAPCOM2 timer T7, 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 XC164CM 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

The Real Time Clock (RTC) module of the XC164CM is directly clocked via a separate clock driver with the prescaled on-chip main oscillator frequency ($f_{RTC} = f_{OSCm}/32$). It is therefore independent from the selected clock generation mode of the XC164CM.

The RTC basically consists of a chain of divider blocks:

- a selectable 8:1 divider (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. Additionally, T14 can generate a separate node request.

Note: The registers associated with the RTC are not affected by a reset in order to maintain the correct system time even when intermediate resets are executed.

The RTC module can be used for different purposes:

- System clock to determine the current time and date,
  optionally during idle mode, sleep mode, and power down mode
- Cyclic time based interrupt, to provide a system time tick independent of CPU frequency and other resources, e.g. to wake-up regularly from idle mode
- 48-bit timer for long term measurements (maximum timespan is > 100 years)
- Alarm interrupt for wake-up on a defined time

A/D Converter

For analog signal measurement, a 10-bit A/D converter with 14 multiplexed input channels and a sample and hold circuit has been integrated on-chip. It uses the method of successive approximation. The sample time (for loading the capacitors) and the conversion time is programmable (in two modes) and can thus be adjusted to the external circuitry. The A/D converter can also operate in 8-bit conversion mode, where the conversion time is further reduced.

Overrun error detection/protection is provided for the conversion result register (ADDAT): either an interrupt request will be generated when the result of a previous conversion has not been read from the result register at the time the next conversion is complete, or the next conversion is suspended in such a case until the previous result has been read.

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

The A/D converter of the XC164CM supports four different conversion modes. In the standard Single Channel conversion mode, the analog level on a specified channel is sampled once and converted to a digital result. In the Single Channel Continuous mode, the analog level on a specified channel is repeatedly sampled and converted without software intervention. In the Auto Scan mode, the analog levels on a prespecified number of channels are sequentially sampled and converted. In the Auto Scan Continuous mode, the prespecified channels are repeatedly sampled and converted. In addition, the conversion of a specific channel can be inserted (injected) into a running sequence without disturbing this sequence. This is called Channel Injection Mode.

The Peripheral Event Controller (PEC) may be used to automatically store the conversion results into a table in memory for later evaluation, without requiring the overhead of entering and exiting interrupt routines for each data transfer.

After each reset and also during normal operation the ADC automatically performs calibration cycles. This automatic self-calibration constantly adjusts the converter to changing operating conditions (e.g. temperature) and compensates process variations.

These calibration cycles are part of the conversion cycle, so they do not affect the normal operation of the A/D converter. The calibration cycles after a conversion can be disabled, so the overall conversion time is reduced again.

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 IO or input stages under software control. This can be selected for each pin separately via register P5DIDIS (Port 5 Digital Input Disable).

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

Asynchronous/Synchronous Serial Interfaces (ASC0/ASC1)

The Asynchronous/Synchronous Serial Interfaces ASC0/ASC1 (USARTs) provide serial communication with other microcontrollers, processors, terminals or external peripheral components. They are upward compatible with the serial ports of the Infineon 8-bit microcontroller families and support full-duplex asynchronous communication and half-duplex synchronous communication. A dedicated baud rate generator with a fractional divider precisely generates all standard baud rates without oscillator tuning.

In asynchronous mode, 8- or 9-bit data frames (with optional parity bit) are transmitted or received, preceded by a start bit and terminated by one or two stop bits. For multiprocessor communication, a mechanism to distinguish address from data bytes has been included (8-bit data plus wake-up bit mode). IrDA data transmissions up to 115.2 kbit/s with fixed or programmable IrDA pulse width are supported. An autobaud detection unit allows to detect asynchronous data frames with its baudrate and mode with automatic initialization of the baudrate generator and the mode control bits.

In synchronous mode, bytes (8 bits) are transmitted or received synchronously to a shift clock which is generated by the ASC0/1.

The LSB is always shifted first. In both modes, transmission and reception of data is FIFO-buffered (8 entries per direction). A loop-back option is available for testing purposes. Five separate interrupt vectors are provided for transmit buffer, transmission, reception, autobaud detection, and error handling.

A number of optional hardware error detection capabilities has been included to increase the reliability of data transfers. A parity bit can automatically be generated on transmission or be checked on reception. Framing error detection allows to recognize data frames with missing stop bits. An overrun error will be generated, if the last character received has not been read out of the receive buffer register at the time the reception of a new character is complete.

Summary of Features

- Full-duplex asynchronous operating modes
  - 8- or 9-bit data frames, LSB first, one or two stop bits, parity generation/checking
  - Baudrate from 2.5 Mbit/s to 0.6 bit/s (@ 40 MHz)
  - Multiprocessor mode for automatic address/data byte detection
  - Support for IrDA data transmission/reception up to max. 115.2 kbit/s (@ 40 MHz)
  - Loop-back capability
  - Auto baudrate detection
- Half-duplex 8-bit synchronous operating mode at 5 Mbit/s to 406.9 bit/s (@ 40 MHz)
- Buffered transmitter/receiver with FIFO support (8 entries per direction)
- Loop-back option available for testing purposes
- Interrupt generation on transmitter buffer empty condition, last bit transmitted condition, receive buffer full condition, error condition (frame, parity, overrun error), start and end of an autobaud detection

High Speed Synchronous Serial Channels (SSC0/SSC1)

The High Speed Synchronous Serial Channels SSC0/SSC1 support full-duplex and half-duplex synchronous communication. They may be configured so they interface with serially linked peripheral components, full SPI functionality is supported.

A dedicated baud rate generator allows to set up all standard baud rates without oscillator tuning.

The SSC transmits or receives characters of 2 … 16 bits length synchronously to a shift clock which can be generated by the SSC (master mode) or by an external master (slave mode). The SSC can start shifting with the LSB or with the MSB and allows the selection of shifting and latching clock edges as well as the clock polarity.

A loop-back option is available for testing purposes.

Three separate interrupt vectors are provided for transmission, reception, and error handling.

A number of optional hardware error detection capabilities has been included to increase the reliability of data transfers. Transmit error and receive error supervise the correct handling of the data buffer. Phase error and baudrate error detect incorrect serial data.

Summary of Features

- Master or Slave mode operation
- Full-duplex or Half-duplex transfers
- Baudrate generation from 20 Mbit/s to 305.18 bit/s (@ 40 MHz)
- Flexible data format
  - Programmable number of data bits: 2 to 16 bits
  - Programmable shift direction: LSB-first or MSB-first
  - Programmable clock polarity: idle low or idle high
  - Programmable clock/data phase: data shift with leading or trailing clock edge
- Loop back option available for testing purposes
- Interrupt generation on transmitter buffer empty condition, receive buffer full condition, error condition (receive, phase, baudrate, transmit error)
- Three pin interface with flexible SSC pin configuration

On-Chip TwinCAN Module

The integrated TwinCAN module handles the completely autonomous transmission and reception of CAN frames in accordance with the CAN specification V2.0 part B (active), i.e. the on-chip TwinCAN module can receive and transmit standard frames with 11-bit identifiers as well as extended frames with 29-bit identifiers.

Two Full-CAN nodes share the TwinCAN module's resources to optimize the CAN bus traffic handling and to minimize the CPU load. The module provides up to 32 message objects, which can be assigned to one of the CAN nodes and can be combined to FIFO-structures. Each object provides separate masks for acceptance filtering.

The flexible combination of Full-CAN functionality and FIFO architecture reduces the efforts to fulfill the real-time requirements of complex embedded control applications. Improved CAN bus monitoring functionality as well as the number of message objects permit precise and comfortable CAN bus traffic handling.

Gateway functionality allows automatic data exchange between two separate CAN bus systems, which reduces CPU load and improves the real time behavior of the entire system.

The bit timing for both CAN nodes is derived from the master clock and is programmable up to a data rate of 1 Mbit/s. Each CAN node uses two pins to interface to an external bus transceiver.

Summary of Features

- CAN functionality according to CAN specification V2.0 B active
- Data transfer rate up to 1 Mbit/s
- Flexible and powerful message transfer control and error handling capabilities
- Full-CAN functionality and Basic CAN functionality for each message object
- 32 flexible message objects
  - Assignment to one of the two CAN nodes
  - Configuration as transmit object or receive object
  - Concatenation to a 2-, 4-, 8-, 16-, or 32-message buffer with FIFO algorithm
  - Handling of frames with 11-bit or 29-bit identifiers
  - Individual programmable acceptance mask register for filtering for each object
  - Monitoring via a frame counter
  - Configuration for Remote Monitoring Mode
- Up to eight individually programmable interrupt nodes can be used
- CAN Analyzer Mode for bus monitoring is implemented

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 until the EINIT instruction has been executed (compatible mode), or it can be disabled and enabled at any time by executing instructions DISWDT and ENWDT (enhanced mode). 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 an internal hardware reset.

The Watchdog Timer is a 16-bit timer, clocked with the system clock divided by 2/4/128/256. 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 13 μs and 419 ms can be monitored (@ 40 MHz).
The default Watchdog Timer interval after reset is 3.28 ms (@ 40 MHz).

Parallel Ports

The XC164CM provides up to 47 I/O lines which are organized into three input/output ports and one input port. All port lines are bit-addressable, and all input/output lines are individually (bit-wise) programmable as inputs or outputs via direction registers. The I/O ports are true bidirectional ports which are switched to high impedance state when configured as inputs. The output drivers of some I/O ports can be configured (pin by pin) for push/pull operation or open-drain operation via control registers. During the internal reset, all port pins are configured as inputs.

The edge characteristics (shape) and driver characteristics (output current) of the port drivers can be selected via registers POCONx.

The input threshold of some ports is selectable (TTL or CMOS like), where the special CMOS like input threshold reduces noise sensitivity due to the input hysteresis. The input threshold may be selected individually for each byte of the respective ports.

All port lines have programmable alternate input or output functions associated with them. 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 XC164CM's Parallel Ports

Port	Control	Alternate Functions
Port1	Pad drivers	Capture inputs or compare outputs, Serial interface lines, Fast external interrupt inputs
Port 3	Pad drivers, Open drain, Input threshold	Timer control signals, serial interface lines, System clock output CLKOUT (or FOUT)
Port 5	–	Analog input channels to the A/D converter, Timer control signals
Port 9	Pad drivers, Open drain, Input threshold	Capture inputs or compare outputs
		CAN interface lines[1)]

1) Can be assigned by software.

2.4 Clock Generation

The Clock Generation Unit uses a programmable on-chip PLL with multiple prescalers to generate the clock signals for the XC164CM with high flexibility. The master clock f_{MC} is the reference clock signal, and is used for TwinCAN and is output to the external system. The CPU clock f_{CPU} and the system clock f_{SYS} are derived from the master clock either directly (1:1) or via a 2:1 prescaler ($f_{SYS} = f_{CPU} = f_{MC}/2$).

The on-chip oscillator 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.

If the bypass mode is used (direct drive or prescaler) the PLL can deliver an independent clock to monitor the clock signal generated by the on-chip oscillator. This PLL clock is independent from the XTAL1 clock. When the expected oscillator clock transitions are missing the Oscillator Watchdog (OWD) activates the PLL Unlock/OWD interrupt node and supplies the CPU with an emergency clock, the PLL clock signal. Under these circumstances the PLL will oscillate with its basic frequency.

The oscillator watchdog can be disabled by switching the PLL off. This reduces power consumption, but also no interrupt request will be generated in case of a missing oscillator clock.

2.5 Power Management Features

The basic power reduction modes (Idle and Power Down) are enhanced by additional power management features (see below). These features can be combined to reduce the controller's power consumption to correspond to the application's possible minimum.

- Basic power saving modes
- Flexible clock generation
- Flexible peripheral management (peripherals can be disabled and enabled)
- Periodic wake-up from Idle mode via RTC timer

The listed features provide effective means to realize standby conditions for the system with an optimum balance between power reduction (standby time) and peripheral operation (system functionality).

Basic Power Saving Modes

The XC164CM can be switched into special operating modes (control via instructions) where its power consumption (and functionality) is reduced.

- **Idle Mode** stops the CPU while the peripherals can continue to operate.
- **Sleep Mode** and **Power Down Mode** stop all clock signals and all operation (RTC may optionally continue running). Sleep Mode can be terminated by external interrupt signals.

Flexible Clock Generation

The flexible clock generation system combines a variety of improved mechanisms (partly user controllable) to provide the XC164CM modules with clock signals. This is especially important in power sensitive modes such as standby operation.

The power optimized oscillator generally reduces the amount of power which is consumed in order to generate the clock signal within the XC164CM.

The clock system controls the distribution and the frequency of internal and external clock signals. The user can reduce the XC164CM's CPU clock frequency which drastically reduces the consumed power.
External circuitry can be controlled via the programmable frequency output FOUT.

Flexible Peripheral Management

Flexible peripheral management provides a mechanism to enable and disable each peripheral module separately. In each situation (such as several system operating modes, standby, etc.) only those peripherals may be kept running which are required for the specified functionality, for example, to maintain communication channels. All others may be switched off. The registers of disabled peripherals can still be accessed.

Periodic Wake-up from Idle or Sleep Mode

Periodic wake-up from Idle mode or from Sleep mode combines the drastically reduced power consumption in Idle/Sleep mode (in conjunction with the additional power management features) with a high level of system availability. External signals and events can be scanned (at a lower rate) by periodically activating the CPU and selected peripherals which then return to powersave mode after a short time. This greatly reduces the system's average power consumption. Idle/Sleep mode can also be terminated by external interrupt signals.

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 XC164CM. The user software running on the XC164CM 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.

The debug interface uses a set of 6 interface signals (4 JTAG lines, 2 break lines) to communicate with external circuitry. These interface signals are implemented as alternate functions on Port 3 pins.

2.7 Protected Bits

The XC164CM provides a special mechanism to protect bits which can be modified by the on-chip hardware from being changed unintentionally by software accesses to related bits (see also **Section 4.8.2**). The following bits are protected:

Table 2-4 XC164CM Protected Bits

Register	Bit Name	Notes
GPT12E_T3CON	**T3OTL**	GPT1 timer output toggle latches
GPT12E_T6CON	**T6OTL**	GPT2 timer output toggle latches
ASC0_CON	**REN**	ASC0 receiver enable flag
ASC1_CON	**REN**	ASC1 receiver enable flag
SSC0_CON	**BSY**	SSC0 busy flag
SSC0_CON	**BE, PE, RE, TE**	SSC0 error flags
SSC1_CON	**BSY**	SSC1 busy flag
SSC1_CON	**BE, PE, RE, TE**	SSC1 error flags
ADC_CON/ ADC_CTR0	**ADST, ADCRQ**	ADC start flag/injection request flag
TFR	**TFR.15, 14, 13, 12**	Class A trap flags
TFR	**TFR.7, 4, 3, 2**	Class B trap flags
PECISNC	**C7IR … C0IR**	All channel interrupt request flags
CC2_SEE	**SEE.15 … SEE.0**	Single event enable bits
CC2_OUT	**CC15IO … CC0IO**	Compare output bits
P1L	**P1L.7**	Those bits of PORT1 used for CAPCOM2
P1H	**P1H.5-4, P1H.0**	Those bits of PORT1 used for CAPCOM2
P9	**P9.5 … P9.0**	All bits of Port 9 used for CAPCOM2
RTC_ISNC	**T14IR, CNT3IR … CNT0IR**	Interrupt node sharing request flags
CC2_CC31-16IC	**CC31IR … CC16IR**	CAPCOM2 interrupt request flags
CC2_T8-7IC	**T7IR, T8IR**	CAPCOM2 timer interrupt request flags
CCU6_IC	**CIR**	CAPCOM6 interrupt request flag
CCU6_EIC	**EIR**	CAPCOM6 error interrupt request flag
CCU6_T12IC	**T12IR**	CAPCOM6 timer T12 interrupt request flag
CCU6_T13IC	**T13IR**	CAPCOM6 timer T13 interrupt request flag
GPT12E_T6-2IC	**T6IR … T2IR**	GPT timer interrupt request flags

Table 2-4 XC164CM Protected Bits (cont'd)

Register	Bit Name	Notes
GPT12E_CRIC	**CRIR**	GPT2 CAPREL interrupt request flag
ADC_CIC	**ADCIR**	ADC end-of-conversion intr. request flag
ADC_EIC	**ADEIR**	ADC overrun interrupt request flag
ASC0_T(B)IC	**TIR, TBIR**	ASC0 transmit (buffer) intr. request flags
ASC0_RIC, ASC0_EIC	**RIR, EIR**	ASC0 receive/error interrupt request flags
ASC0_ABIC	**ABIR**	ASC0 autobaud interrupt request flags
ASC1_T(B)IC	**TIR, TBIR**	ASC1 transmit (buffer) intr. request flags
ASC1_RIC, ASC1_EIC	**RIR, EIR**	ASC1 receive/error interrupt request flags
ASC1_ABIC	**ABIR**	ASC1 autobaud interrupt request flags
SSC0_TIC, SSC0_RIC	**TIR, RIR**	SSC0 transmit/receive intr. request flags
SSC0_EIC	**EIR**	SSC0 error interrupt request flag
SSC1_TIC, SSC1_RIC	**TIR, RIR**	SSC1 transmit/receive intr. request flags
SSC1_EIC	**EIR**	SSC1 error interrupt request flag
PLLIC	**PLLIR**	PLL/OWD interrupt request flag
EOPIC	**EOPIR**	End-of-PEC interrupt request flag
CAN_7IC, CAN_0IC	**CAN7IR … CAN0IR**	TwinCAN interrupt request flags
RTC_IC	**RTCIR**	RTC interrupt request flag
-----	**XX16IR … XX0IR**	"Unassigned node" interrupt request flags

3 Memory Organization

The memory space of the XC164CM 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 Flash, internal RAM, the internal Special Function Register Areas (SFRs and ESFRs), the internal IO area, and the external memory space (containing only the on-chip TwinCAN module residing on the internal LX-bus) are mapped into one common address space.

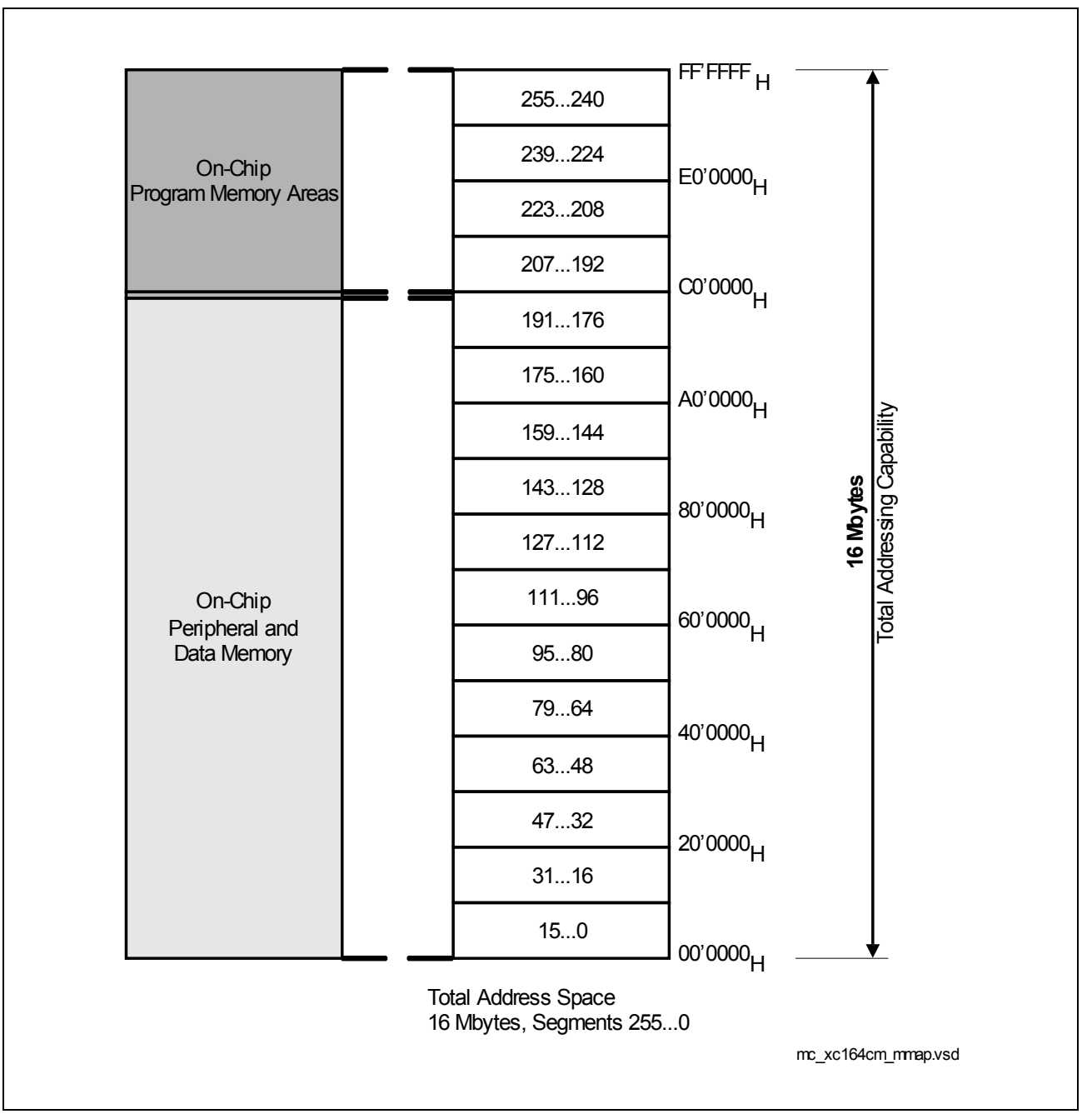


Figure 3-1 Address Space Overview

The XC164CM 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. 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.

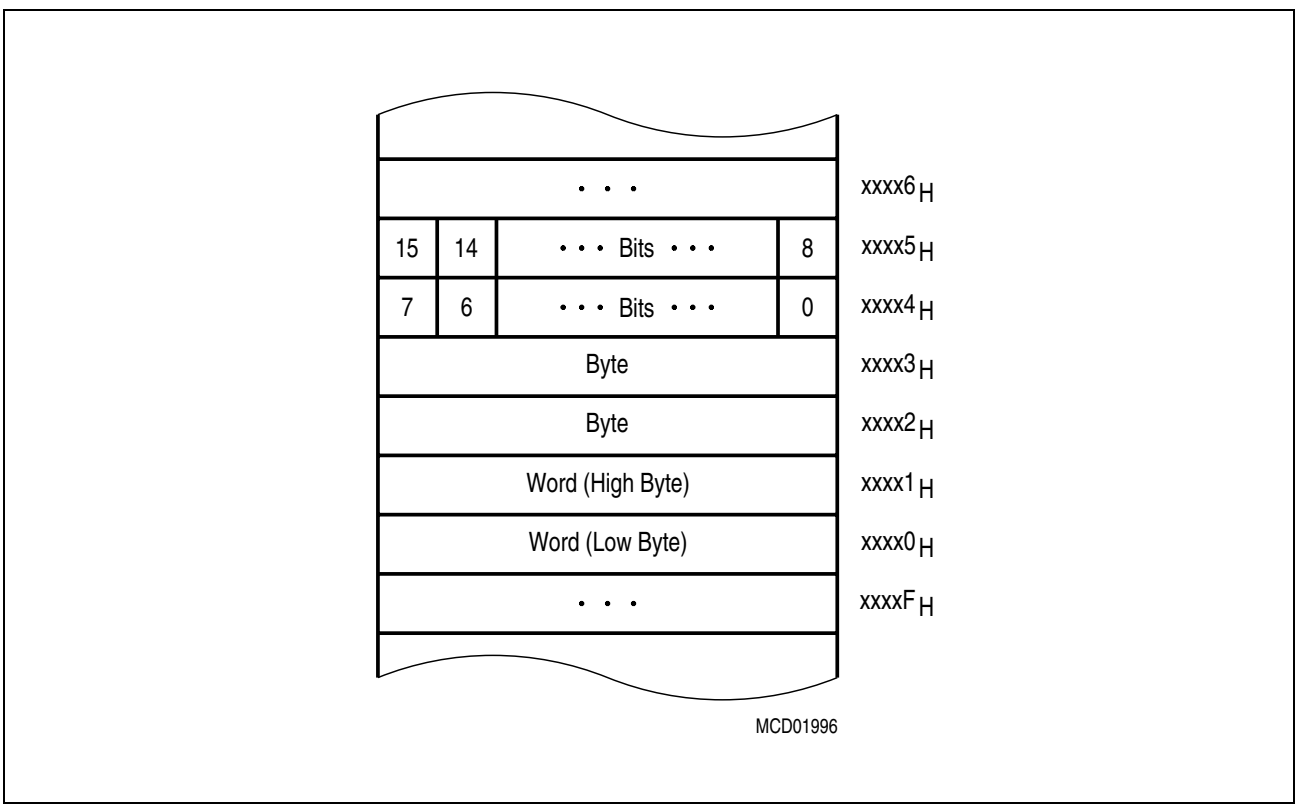


Figure 3-2 Storage of Words, Bytes, and Bits in a Byte Organized Memory

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 XC164CM contains some reserved areas, so future derivatives can be enhanced in an upward-compatible fashion.

Table 3-1 XC164CM Memory Map

Address Area	Start Loc.	End Loc.	Area Size[1)]	Notes
Flash register space	$FF'F000_H$	$FF'FFFF_H$	4 Kbytes	[2)]
Reserved (Acc. trap)	$F8'0000_H$	$FF'EFFF_H$	< 0.5 Mbytes	Minus Flash regs
Reserved for PSRAM	$E0'0800_H$	$F7'FFFF_H$	< 1.5 Mbytes	Minus PSRAM
Program SRAM	$E0'0000_H$	$E0'07FF_H$	2 Kbytes	Maximum[3)]
Reserved for pr. mem.	$C1'0000_H$	$DF'FFFF_H$	< 2 Mbytes	Minus Flash
Program Flash	$C0'0000_H$	$C0'FFFF_H$	64 Kbytes	[4)]
Reserved	$40'0000_H$	$BF'FFFF_H$	8 Mbytes	–
Reserved	$20'0800_H$	$3F'FFFF_H$	< 2 Mbytes	Minus TwinCAN
TwinCAN registers	$20'0000_H$	$20'07FF_H$	2 Kbytes	Accessed via EBC
Reserved	$01'0000_H$	$1F'FFFF_H$	< 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	–
ESFR area	$00'F000_H$	$00'F1FF_H$	0.5 Kbyte	–
XSFR area	$00'E000_H$	$00'EFFF_H$	4 Kbytes	–
Reserved	$00'C800_H$	$00'DFFF_H$	6 Kbytes	–
Data SRAM[4)]	$00'C000_H$	$00'C7FF_H$	2 Kbytes	XC164CM-8F only
Reserved for DSRAM	$00'8000_H$	$00'BFFF_H$	16 Kbytes	–
Reserved	$00'0000_H$	$00'7FFF_H$	32 Kbytes	–

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

2) Not defined register locations return a trap code.

3) This is the maximum implemented in the derivatives described in this manual.

4) Depends on the respective derivative. The derivatives are listed in **“About this Manual” on Page 1-2**.

3.2 Special Function Register Areas

The Special Function Registers (SFRs) controlling the system and peripheral functions of the XC164CM can be accessed via three 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$_H$)

This arrangement provides upward compatibility with the derivatives of the C166 Family.

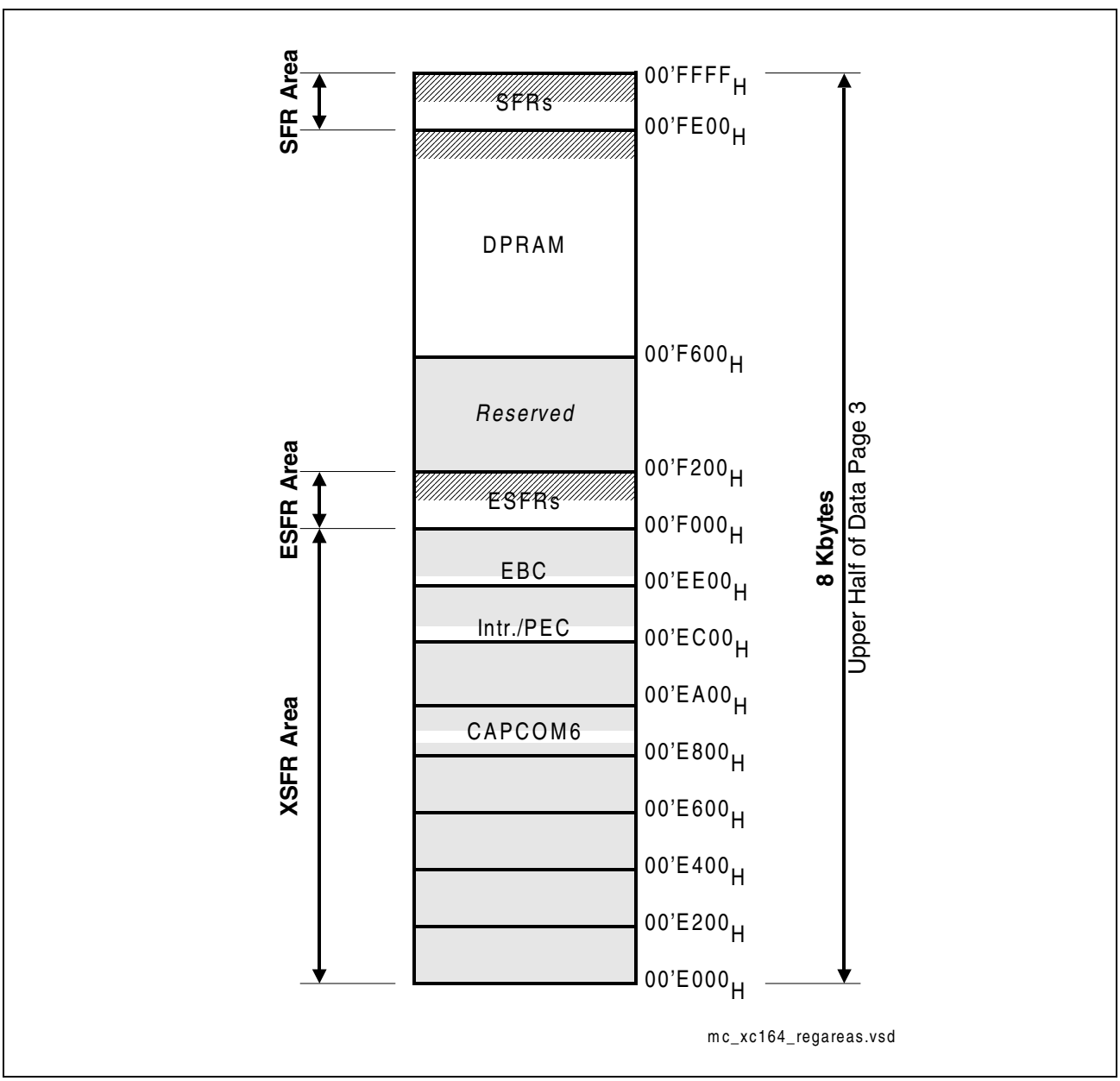


Figure 3-3 Special Function Register Mapping

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

Special Function Registers

The functions of the CPU, the bus interface, the IO ports, and the on-chip peripherals of the XC164CM 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   ODP9, #data16       ;ODP9 uses 8-bit reg addressing
BFLDL DP9, #mask, #data8  ;Bit addressing for bit fields
BSET  DP1H.0              ;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. Bitfield 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 Register
<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 XC164CM 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.

For more details about the use of the source and destination pointers for PEC data transfers see **Section 5.4**.

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

3.3 Data Memory Areas

The XC164CM provides two on-chip RAM areas 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 3.4**). However, the data memory areas provide the fastest access.*

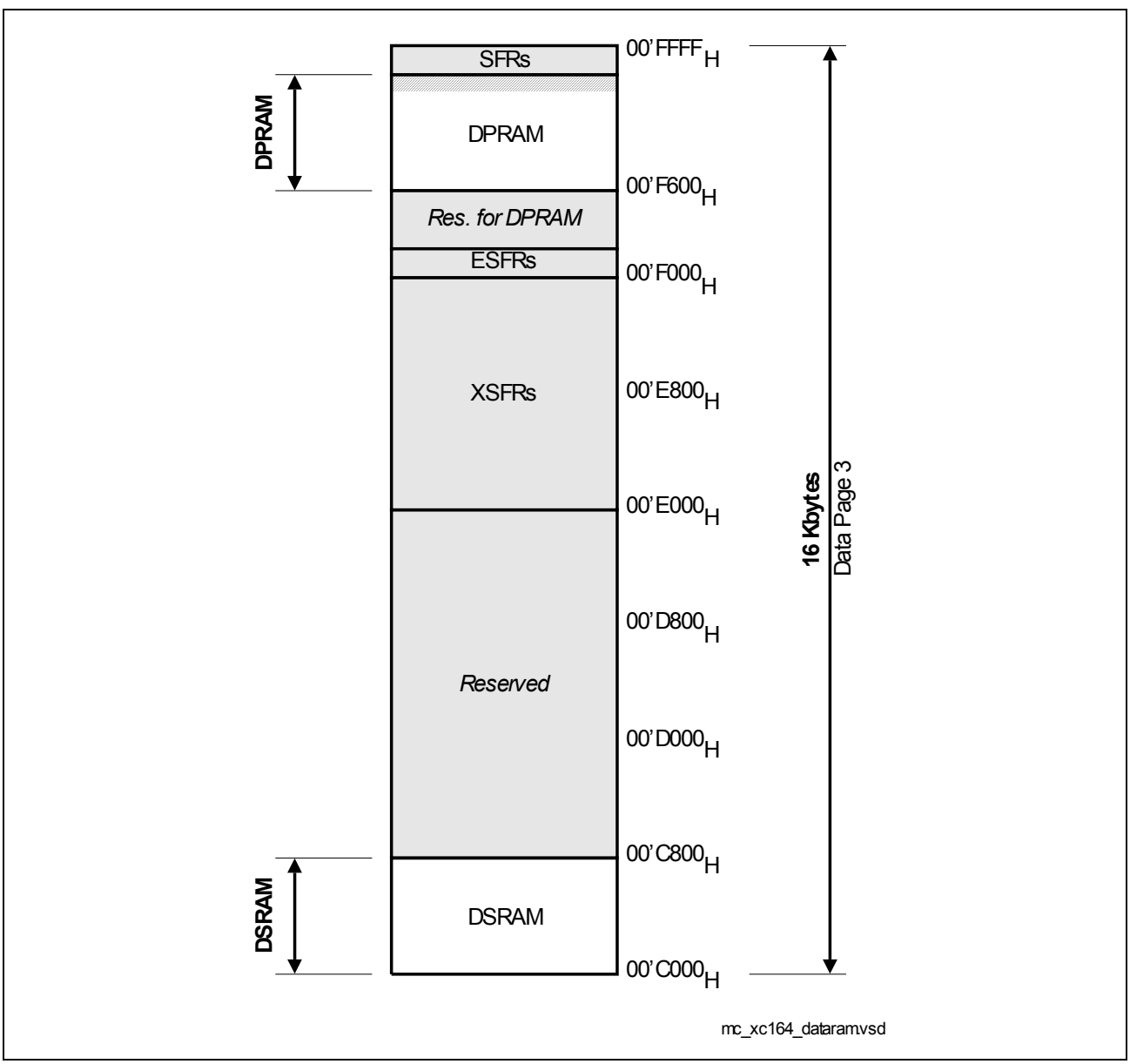


Figure 3-4 On-Chip Data RAM Mapping

*Note: The DSRAM size depends on the respective derivative. The derivatives are listed in **"About this Manual" on Page 1-2**.*

Dual-Port RAM (DPRAM)

The XC164CM 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 bitaddressable (see hashed block in **Figure 3-4**).

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)[1)]

The XC164CM provides 2 Kbytes of DSRAM ($00'C000_H$ … $00'C7FF_H$). 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. Any word data access is made on an even byte address. The highest possible word data storage location in the DSRAM is $00'C7FE_H$.

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 20 Kbytes is dedicated to DSRAM ($00'8000_H$ … $00'CFFF_H$). The locations without implemented DSRAM are reserved.

1) Depends on the respective derivative. The derivatives are listed in **"About this Manual" on Page 1-2**.

3.4 Program Memory Areas

The XC164CM provides two on-chip program memory areas for code/data storage:

- **The Program Flash** stores code and constant data. Flash memory is (re-) programmed by the application software.
- **The Program SRAM (PSRAM)** stores temporary code sequences and other data. For example higher level bootloader software can be written to the PSRAM and then be executed to program the on-chip Flash memory.

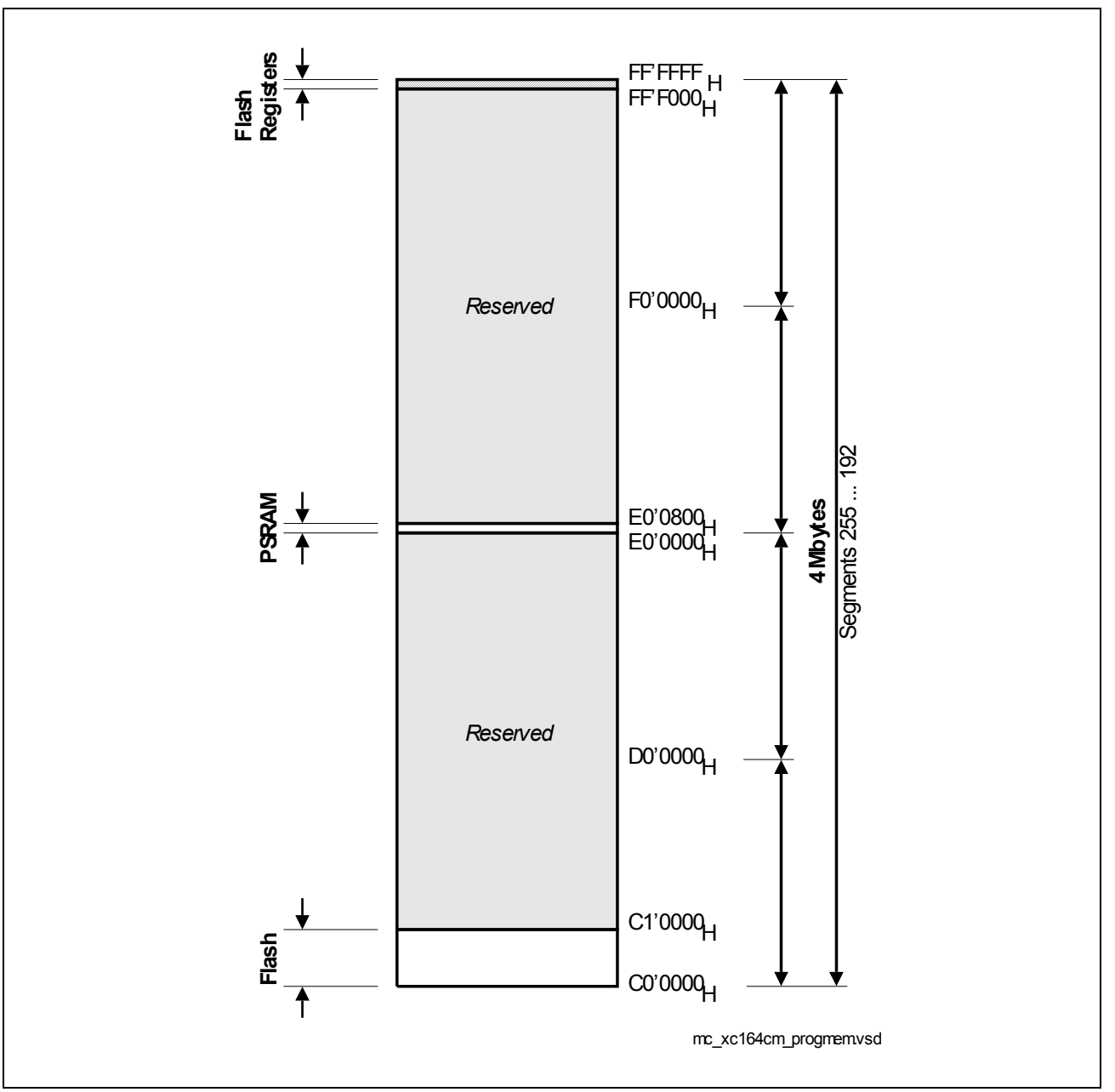


Figure 3-5 On-Chip Program Memory Mapping

Program/Data SRAM (PSRAM)

The XC164CM provides 2 Kbytes of PSRAM ($E0'0000_H$ … $E0'07FF_H$). 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 data page 896. Any word data access is made on an even byte address. The highest possible word data storage location in the PSRAM is $E0'07FE_H$.

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 bitaddressable.

An area of 1.5 Mbytes is dedicated to PSRAM ($E0'0000_H$ … $F7'FFFF_H$). The locations without implemented PSRAM are reserved.

Non-Volatile Program Memory (Flash)

The XC164CM provides 64 Kbytes[1)] of program Flash ($C0'0000_H$ … $C0'FFFF_H$). 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 $C0'FFFE_H$[1)].

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 bitaddressable.

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

1) Depends on the respective derivative. The derivatives are listed in **"About this Manual" on Page 1-2**.

3.5 System Stack

The system stack may be defined anywhere within the XC164CM's memory areas.

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 XC164CM's address space are marked as IO area:

- **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 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. prefetching 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 Crossing Memory Boundaries

The address space of the XC164CM 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 Flash, the on-chip LXBus-peripherals (if integrated), and the external memory (not available at XC164CM).

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 prefetcher 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-Kbyte data page boundaries will use different data page pointers, while the physical locations need not be subsequent within memory.

3.8 The On-Chip Program Flash Module

The XC164CM incorporates 64 Kbytes[1)] of embedded Flash memory (starting at location C0'0000$_H$, see **Figure 3-5**) for code or constant data. It is operated from the 5 V pad supply and requires no additional programming voltage. The on-chip voltage generators require a power stabilization time of approx. 250 µs. The Flash array is organized in five sectors of 4 × 8 Kbytes, and 1 × 32 Kbytes[1)]. It combines the advantages of very fast read accesses with protected but simple writing algorithms for programming and erasing. The 64-bit code read accesses realize maximum CPU performance by fetching two double word instructions (or four single word instructions) in a single access cycle.

Data integrity is enhanced by an error correction code enabling dynamic correction of single bit errors. Additionally, special margin checks are provided to detect and correct problematic bits before they lead to actual malfunctions.

All Flash operations are controlled by command sequences (according to the JEDEC single-power-supply Flash standard). The algorithms for programming and erasing are executed automatically by the internal Flash state control machine. This avoids inadvertent destruction of the Flash contents at a reasonably low software overhead. Command sequences consist of subsequent write (or read) accesses to virtual locations within the Flash space or the Flash register space. The virtual Flash locations are defined by special addresses (see command sequence table).

For optimized programming efficiency, paging mode (burst mode) allows 128 bytes to be loaded into a page buffer with fast CPU accesses before this buffer is programmed into the Flash with one single store command (2 ms typical[2)]). Each sector can be erased separately (200 ms typical[2)]).

Note: Erased Flash memory cells contain all '0's, contrary to standard EPROMs.

Security is provided by a general read/write protection (complete Flash array) and a sector-specific[3)] write protection. The temporary disabling of these hardware protection features is secured with a password check sequence. The lock information and the keywords used for the password check sequence are stored apart from the user's code and data in a separate security sector (see **Section 3.8.4**).

A dedicated Flash status register returns global and sector-specific status information. The correct execution of an operation and the general status of the Flash module can be checked via the Flash status register at any time.

The physical address range of the Flash module covers byte addresses from 0'0000$_H$ to 0'FFFF$_H$[1)]. These physical addresses are mapped to the XC164CM's program memory area starting at C0'0000$_H$. Also the separate security sector is mapped to this area. Access conflicts are avoided by special security commands.

1) Depends on the respective derivative. The derivatives are listed in **"About this Manual" on Page 1-2**.

2) For exact parameters please refer to the data sheet.

3) For write protection two 8-Kbyte sectors are combined to one lockable 16-Kbyte section.

In-System-Programming is supported by the automatic program/erase algorithms and the large page buffer, which may be filled by a programming routine executed out of the Flash memory itself. During the actual program/erase algorithm Flash read accesses are stalled. Also completely erased Flash modules can be programmed within the system. The built-in bootstrap loader can load an initial programming routine via the serial interface, which in turn can then program the Flash module. This is useful for the initial programming (virgin Flash) as well as in case of a problem (e.g. power failure) during reprogramming, when no safety routines are provided.

Note: Accesses to a protected Flash are totally disabled during bootstrap mode. Before any program/erase operation the protection must be temporarily disabled using the correct password sequence.

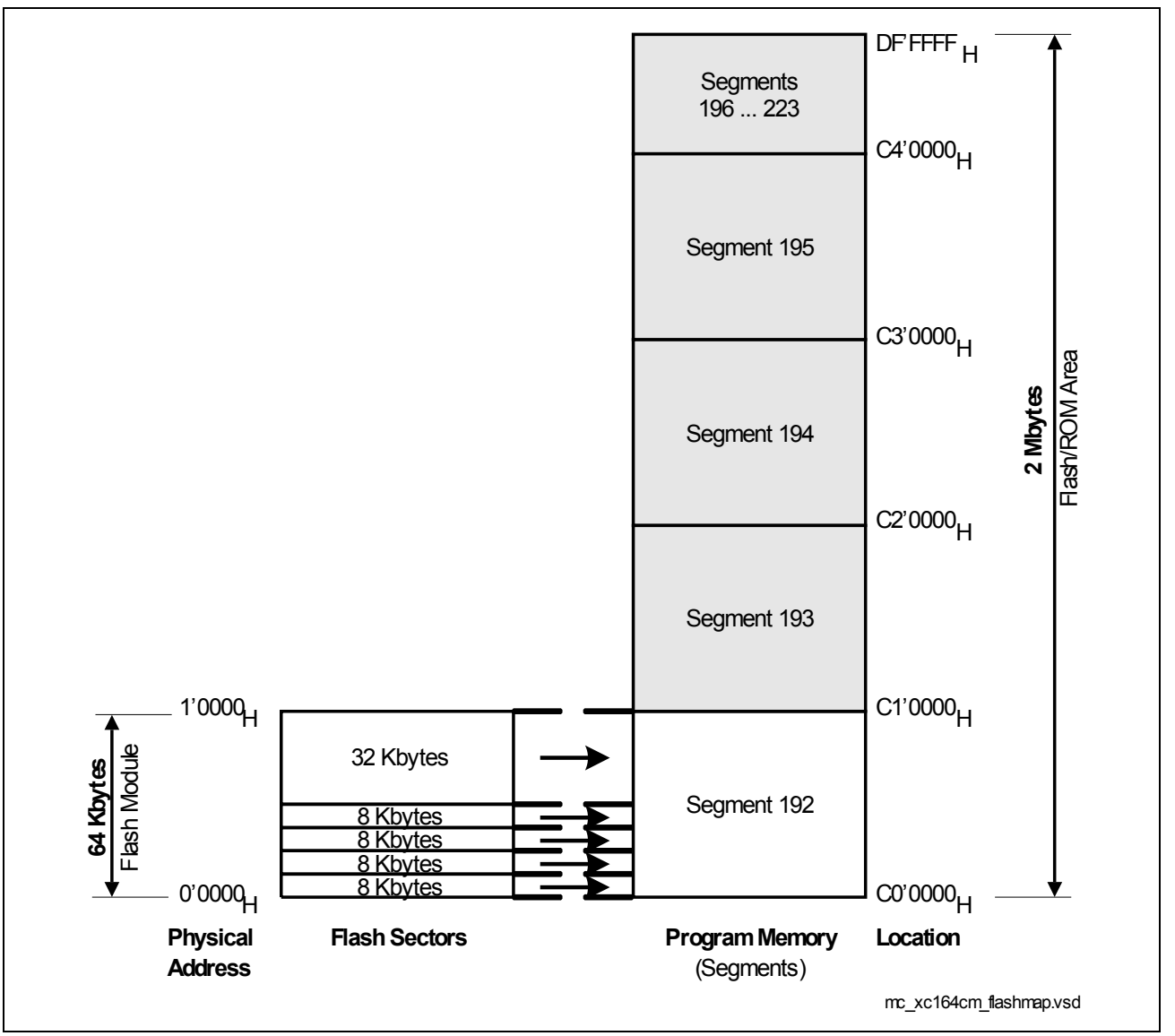


Figure 3-6 Mapping of the On-Chip Flash Module Sectors

Note: The program memory segment 192 is mirrored to the segment 193.

3.8.1 Flash Operating Modes

Two basic operating modes of the on-chip Flash module can be distinguished:

- **Standard read mode:** code and data can be read from the Flash module
- **Command mode:** the Flash module executes a previously defined command

Standard Read Mode

In standard read mode (the normal operating mode) the Flash memory appears like a standard ROM, allowing code and data accesses in any addressing mode.

Standard read mode is entered in the following cases:

- After the deactivation of the system reset (after power stabilization)
- After execution of the reset command, if no program or erase operation is active
- After every completed command execution (program, erase, etc.)
- When a command sequence error is detected
- When a protection violation is detected (program or erase a protected sector)

Note: Standard read mode is indicated by status bit BUSY = '0'.

Standard read mode is terminated when the last command of a command sequence is decoded and a Flash array operation is started (program or erase). Therefore, all steps of a command sequence before the last command (in particular the loading of the page buffer) can be executed by code read from the Flash module itself.

Each read access to the Flash memory activates the automatic error detection. Double-bit errors are detected and indicated, single-bit errors are detected, indicated, and automatically corrected (see **Section 3.8.3**).

Note: Single bit errors can be located and avoided by a margin check operation.

Command Mode

All Flash operation except for standard read operations are initiated by command sequences written to the (virtual) Flash command register (a location within the Flash space). Protected commands additionally require four passwords for validation.

Command mode is entered after the last command of a command sequence has been written. For all other command sequences, which activate a Flash array operation such as erase sector, the command execution and thus the command mode remains active for a defined time. While in command mode (busy) read accesses to the Flash array are delayed until the Flash module returns to standard read mode.

Note: Command mode is indicated by status bit BUSY = '1'.

Command mode is terminated by the correct execution of the command or by an error condition as indicated in the status register.

Command sequences not starting Flash operations (e.g. Enter Page Mode) are executed immediately and command mode is not entered.

3.8.2 Command Sequences

All operations besides normal read operations are initiated and controlled by command sequences written to the Flash state machine. The different write cycles of command sequences define the intended command, but also establish a fail-safe mechanism to protect against inadvertent operations. Commands not directly controlling Flash array operations are single cycle commands for performance reasons, commands affecting the Flash array require several cycles, commands affecting security issues require a 64-bit security code (four passwords) to be accepted. Command cycles need not be consecutively received (pauses allowed).

Command sequences can be performed simultaneously to instruction fetch operations, so instructions for command sequences also can be executed out of the on-chip Flash, as long as the Flash module is in read mode and not executing an erase or programming operation. Command sequences for polling the status register are allowed in any state, also during erase and programming operations, if they are executed out of memory outside the Flash module. Otherwise, instruction fetching is stalled.

Writing incorrect address and data values or writing them in the improper sequence will abort the intended operation, reset the module to read mode, and set the sequence error flag in the status register.

Read Status commands address the separate Flash register space and do not require command sequences. Register write cycles are only executed with a command cycle.

Programming operations are supported by a 128-byte page buffer which can be loaded with maximum speed, and is then programmed with one single command sequence. Programming is done in three steps:

- Initialize the page buffer with the Enter Page Mode command (this also defines the target page address).
- Load the page buffer with consecutive Load Page command (the page buffer offset is incremented automatically).
- Program the complete buffer with the Write Page command.

Erase operations clear all bits of a selected sector or of a 256-byte wordline. Erase command sequences include the address of the target sector or wordline.

After being requested the program/erase operation is executed automatically and requires no additional user control. The operation itself and its termination are indicated by status flags. A Power Down request is delayed until the termination of the program/erase operation. A reset aborts the program/erase operation within the power stabilization time, indicated by an operation error (OPER) in the Flash status register.

The three tables below summarize the implemented command sequences for:

- organizational Flash accesses (**Table 3-3**),
- programming and erasing (**Table 3-4**),
- protection control (**Table 3-5**).

Note: Each command sequence lists the required address (A = …) and data (D = …).

Organizational Commands

Table 3-3 Command Sequence Definitions (Organizational Accesses)

Cycle	Reset to Read Mode	Clear Status	Read Flash Status or Margin	Write Margin
1	A = Cx'xxAA$_H$ D = xxF0$_H$	A = Cx'xxAA$_H$ D = xxF5$_H$	A = RLOC D = <status>	A = Cx'xxAA$_H$ D = xxFA$_H$
2	–	–	–	A = FF'F00C$_H$ D = margin

Notes:

RLOC is the respective register offset (rr) within the Flash register area starting at FF'F000$_H$ (FF'F0rr$_H$).
<status> is the returned status word.
margin is the control word used for margin control.

The shown virtual address (Cx'xxAA$_H$) must point to the Flash space (e.g. C0'00AA$_H$).

The "Read Flash status" command may be executed during command mode in order to check the BUSY bit of the Flash module.

The Reset To Read command aborts not completed command sequences and clears the error flags in the status register FSR. The reset command can be issued at any point during the command sequence, except for parts of the password check sequence. It does not terminate command mode, i.e. abort busy state.

The Clear Status command clears the error flags and the write status bits PROG and ERASE (the hardware-controlled indication flags are not affected). The clear status command is only accepted in Read Mode and otherwise generates a sequence error.

The Read Register command returns the contents of the following registers:

- The Flash Status Register FSR providing general Flash status information.
- The Protection Configuration Register PROCON indicating the protected sectors.
- The Margin Control Register MAR indicating the selected Flash read margin.

The Write Margin Register command is used for verify operations and for user-controlled refresh operations to identify and correct problematic bits (see **Section 3.8.3**).

Program/Erase Commands

Table 3-4 Command Sequence Definitions (Programming & Erasing)

Cycle	Enter Page Mode[1)]	Load Page Data Word[2)]	Write Page[3)]	Erase Sector[1)]	Erase Wordline[1)]
1	A = Cx'xxAA$_H$ D = xx50$_H$	A = Cx'xxF2$_H$ D = WDAT	A = Cx'xxAA$_H$ D = xxA0$_H$	A = Cx'xxAA$_H$ D = xx80$_H$	A = Cx'xxAA$_H$ D = xx80$_H$
2	A = WLOC D = xxAA$_H$	–	A = Cx'xx5A$_H$ D = xxAA$_H$	A = Cx'xx54$_H$ D = xxAA$_H$	A = Cx'xx54$_H$ D = xxAA$_H$
3	–	–	–	A = SLOC D = xx33$_H$	A = WLA D = xx03$_H$

1) While protection is enabled, this command sequence is rejected.

2) Words written in excess of the buffer capacity of 128 bytes are lost.

3) This command sequence is only accepted if page mode has been entered before.

Notes:

WLOC is the first (lowest) location of the 128-byte block to which the 128-byte buffer shall be written, e.g. C0'AB80$_H$ or C0'AC00$_H$ (128-byte boundary).
WDAT is the data word which shall be stored in the buffer.
SLOC is the first (lowest) location within the target sector, e.g. C0'6000$_H$ for sector 3.
WLA is the first (lowest) location of the 256-byte wordline to be erased, e.g. C0'7F00$_H$ for the uppermost 256 bytes (top of sector 3).

The shown virtual addresses (Cx'xx..$_H$) must point to the Flash space (e.g. C0'00AA$_H$).

The "Read Flash status" command may be executed during command mode in order to check the BUSY bit of the Flash module.

Caution: Writing to a Flash page (space for the 128-byte buffer) **more than once** before erasing may destroy data stored in neighbor cells! This is especially important for programming algorithms that do not write to sequential locations.

The Enter Page Mode command prepares the programming of a 128-byte page by clearing the page buffer and initializing the internal word assembly pointer. Bit PAGE in the status register FSR is set to indicate this. Issuing the Enter Page Mode command during page mode aborts the current operation and starts a new page operation. The data written to the page buffer during the aborted page operation are lost. The Enter Page Mode command also defines the location of the 128-byte page to be programmed.

Note: The Enter Page Mode command is only accepted while protection is disabled.

The Load Page Data Word command adds the accompanying data word to the page buffer. The offset within the page buffer is determined by the internal buffer pointer which is incremented after each load operation. Data words written in excess of the buffer capacity of 128 bytes are lost (no error indicated).

Note: The Load Page Data Word command is only accepted while page mode is active.

The Write Page command writes (programs) the contents of the 128-byte page buffer (including the error correction code) to the Flash array. The address of the programmed page is defined by the preceding Enter (Security) Page Mode command.

After the Write Page command the Flash module enters command mode, indicated by PAGE = 0, PROG = 1, BUSY = 1. Read accesses to the Flash module are delayed until command mode is terminated. The programming operation itself is executed automatically and requires no additional user control.

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

Note: The Write Page command is only accepted while page mode is active.

The Erase Sector command clears all bits within the selected sector (see SLOC).

After the Erase Sector command the Flash module enters command mode, indicated by ERASE = 1, BUSY = 1. Read accesses to the Flash module are delayed until command mode is terminated. The erase operation itself is executed automatically and requires no additional user control.

Note: The Erase Sector command is only accepted while protection is disabled.

The Erase Wordline command clears all bits within the selected 256-byte wordline (see WLA).

After the Erase Wordline command the Flash module enters command mode, indicated by ERASE = 1, BUSY = 1. Read accesses to the Flash module are delayed until command mode is terminated. The erase operation itself is executed automatically and requires no additional user control.

Note: The Erase Wordline command is only accepted while protection is disabled.

Protection Control Commands

Table 3-5 Command Sequence Definitions (Protection Control)

Cycle	Disable Read Protection	Disable Write Protection	Re-Enable Protection	Erase Security Wordline[1)]	Enter Security Page Mode[1)]
1	A = Cx'xx3C$_H$ D = xx00$_H$	A = Cx'xx3C$_H$ D = xx00$_H$	A = Cx'xx5E$_H$ D = xx5E$_H$	A = Cx'xxAA$_H$ D = xx80$_H$	A = Cx'xxAA$_H$ D = xx55$_H$
2	A = Cx'xx54$_H$ D = PW1	A = Cx'xx54$_H$ D = PW1	–	A = Cx'xx54$_H$ D = xxA5$_H$	A = SECLOC D = xxAA$_H$
3	A = Cx'xxAA$_H$ D = PW2	A = Cx'xxAA$_H$ D = PW2	–	A = SECWLA D = xx53$_H$	–
4	A = Cx'xx54$_H$ D = PW3	A = Cx'xx54$_H$ D = PW3	–	–	–
5	A = Cx'xxAA$_H$ D = PW4	A = Cx'xxAA$_H$ D = PW4	–	–	–
6	A = Cx'xx5A$_H$ D = xx55$_H$	A = Cx'xx5A$_H$ D = xx05$_H$	–	–	–

1) While protection is enabled, this command sequence is rejected.

Note: A Reset-To-Read command cannot be executed while the 2nd or the 4th password is expected. In this case the command is taken as a password.

Notes:

SECLOC is the first (lowest) location of the 128-byte block within the security sector to which the 128-byte buffer shall be written, e.g. C0'0080$_H$ or C0'0100$_H$.
SECWLA is the first (lowest) location of the 256-byte security wordline to be erased, e.g. C0'0100$_H$ for the upper 256-byte wordline.
PWn is one of the four passwords building the 64-bit security code (n = 1 … 4).

The shown virtual addresses (Cx'xx..$_H$) must point to the Flash space (e.g. C0'00AA$_H$).

The "Read Flash status" command may be executed during command mode in order to check the BUSY bit of the Flash module.

The Disable Read Protection command temporarily disables the general Flash read protection (including the general write protection), indicated by PRODI = 1. Read protection remains disabled until the execution of the Re-Enable Protection command or until the next reset.

While read protection is disabled, Flash read accesses including injected OCDS instructions are executed. Program/Erase operations can be executed as long as the respective sector is not locked by a sector-specific write protection.

Note: This command sequence can also be used to verify the programmed keywords before the protection is locked with the confirmation. A wrong keyword is indicated by bit PROER in the Flash Status Register FSR.

This is a protected command sequence requiring the 64-bit security code (four user-defined passwords) for validation (see **Section 3.8.4**).

The Disable Sector Write Protection command temporarily disables the sector-specific write protection for all write-protected sectors, indicated by SUL = 1. Write protection remains disabled until the execution of the Re-Enable Protection command or until the next reset.

While write protection is disabled, all Flash operations can be executed as long as the respective sector is not locked by the general read/write protection.

Note: This command sequence can also be used to verify the programmed keywords before the protection is locked with the confirmation. A wrong keyword is indicated by bit PROER in the Flash Status Register FSR.

This is a protected command sequence requiring the 64-bit security code (four user-defined passwords) for validation (see **Section 3.8.4**).

The Re-Enable Protection command immediately resumes all installed but temporarily disabled protection features (general read/write protection and/or sector-specific write protection).

The Erase Security Wordline command clears all bits within the selected wordline (see SECLOC).

After the Erase Security Wordline command the Flash module enters command mode, indicated by ERASE = 1, BUSY = 1. Read accesses to the Flash module are delayed until command mode is terminated. The erase operation itself is executed automatically and requires no additional user control.

After the erase operation, the protection configuration (including keywords or protection confirmation code) is valid directly after execution of this command (see **Section 3.8.4**).

Note: The Erase Security Wordline command is only accepted while protection is disabled.

The Enter Security Page Mode command prepares the programming of a 128-byte page within the security sector by clearing the page buffer and initializing the internal word assembly pointer. Bit PAGE in the status register FSR is set to indicate this. Issuing the Enter Security Page Mode command during page mode aborts the current operation and starts a new page operation. The data written to the page buffer during the aborted page operation are lost. The Enter Security Page Mode command also defines the location of the 128-byte page to be programmed. Also refer to **Section 3.8.4**.

Note: The Enter Security Page Mode command is only accepted while any protection is disabled.

Interaction between Program Flash and Security Sector Programming

The on-chip Flash module of the XC164CM uses specific internal status information for the Program Flash area and for the Security Sector. This internal status information is updated with an erase operation or a programming operation (write page command). After reset, always the status information for the program flash is selected.

To ensure correct programming, make sure that a programming operation to either area (program/security) is always preceded by a programming or erase operation to the same area, otherwise wrong data may be stored.

An erase operation to either area (program/security) is always executed correctly, independent of the preceding operation.

In other words:

make sure that no erase/program operation to the program flash or reset occurs between erasing and programming of an area in the security sector.

make sure that no erase/program operation to the security sector is executed between erasing and programming of an area in the program flash.

3.8.3 Error Correction and Data Integrity

Data integrity is supported by the Error Correction Code (ECC). This ECC is dynamically generated during Flash write operations and stored in the Flash array together with the corresponding data. For each read access the associated 8-bit ECC is fetched together with the 64-bit read data and is evaluated.

Single bit errors are detected and automatically corrected on-the-fly (during run-time). Therefore, single bit errors do not affect system operation.

Double bit errors are detected and trigger an Access Fault trap. This prevents erroneous instructions or data from being used.

Each read error condition is indicated by a dedicated flag (SBER, DBER) in the Flash Status Register FSR.

The probability of a double bit error (not automatically correctable by ECC) is extremely low. Double bit errors can be avoided by performing a recovery operation after a single bit error has been detected. For the recovery operation the following steps must be done:

- Detect the wordline containing the erroneous bit
- Store the contents of the wordline temporarily
- Erase this wordline
- Reprogram the erased wordline (requires two write page operations)

The wordline data copied to the temporary buffer are valid, because a single bit error during reading is automatically corrected via the ECC. Erasing and programming is done using standard command sequences.

Verify Operation

The violated wordline can be detected by a verify operation. After clearing bit SBER a certain area of the Flash memory is read. Since the Flash array always delivers 64-bit data, the read address can be incremented by 8 after every access, which minimizes the number of necessary read cycles. After reading the defined area bit SBER indicates if or if not this area contains the single bit error. The verify algorithm can gradually decrease the size of the checked area down to the size of a wordline, or can check all wordlines sequentially.

Refresh Operation

Even single bit errors can be avoided by detecting problematic (moving) bits before they lead to a read error (and a recovery operation during runtime) and by reprogramming (refreshing) them in advance. Problematic bits can be detected by combining the verify operation with margin check control.

Margin Check Control

Flash cells store charges to represent bit levels. If the charge stored in a cell changes (e.g. due to charge coupling during operations on neighbor cells) the respective bit may be read wrong. As the charges change slowly this effect can be detected before a bit is actually read wrong. In this case also a preventive correction (via software) is possible.

A problematic bit (i.e. a bit with a changed charge) can be detected by applying a more severe comparator margin when reading a Flash location. This margin is controlled by the Margin Control Register MAR, accessible with the special command sequences Read/Write Margin (see **Table 3-3**).

A severe margin is selected by writing the value MARLEVSEL = 0001_B or 0100_B to register MAR. A bit that returns a 1 when read with low level margin, while returning a 0 when read with standard margin, represents a problematic bit, called weak zero. A bit that returns a 0 when read with high level margin, while returning a 1 when read with standard margin, represents a problematic bit, called weak one. Compare operations over a certain memory area using standard and severe margins reveal these problematic bits.

Note: Read operations may directly follow a MAR change operation.

MAR
Margin Control Register **SFR (FF'F00C$_H$)** **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	MARWV	-	-	-	MARLEVSEL			
-	-	-	-	-	-	-	-	rw	-	-	-	rw			

Field	Bits	Type	Description
MARWV	7	rw	**Margin Write Validation** 0 Reset value. MARWV must not be written 0. 1 Must be set (MARWV = 1) with every write access to register MAR, independent of the purpose of the write access.
MARLEVSEL	[3:0]	rw	**Margin Level Selection** 0000 Standard read margin (regular operation) 0001 Low level margin (used to verify weak zeros) 0100 High level margin (used to verify weak ones) other **Reserved**

Note: Margin values can only be written via the Write Margin command. Bit MARWV must be set with every write access.

3.8.4 Protection and Security Features

The Flash module provides powerful and flexible protection of data and code against destruction (i.e. erasure) and undesired modification (i.e. reprogramming) as well as against undesired read access to Flash contents. Two protection mechanisms can be activated:

- **Sector-specific write protection** protects individual sectors against erasing and programming. This is important for the integrity of boot software and also avoids modifications of code/data by malfunction or even manipulation.
- **General read/write protection** protects the complete program Flash area against all accesses from outside the module itself. This includes data read accesses, instruction fetches (i.e. jumps into the program Flash area), and OCDS operations. The general read/write protection also disables erasing and programming. Command sequences and register accesses are executed, however.

Each protection feature is installed by user software. Protection features may be disabled temporarily to reprogram portions of the Flash memory or to call an external subroutine. Disabling and re-enabling is done under software control. However, after a reset all installed protection features are active (enabled) automatically.

By combining the two protection features a flexible protection scheme can be installed to protect the Flash memory or parts of it against unauthorized programming or erasing according to the application's requirements.

Note: Protection is provided for the Program Flash only, there is no protection for the Program SRAM.

Passwords and Security Code

All protection feature control (install, disable, re-enable) is accomplished through command sequences similar to the program/erase sequences (see **Table 3-5**). The two command sequences that temporarily suspend the protection feature are additionally secured by a password check sequence (64-bit security code) to ensure maximum safety against undesired accesses.

During password checking, the four passwords entered via the command sequence are compared to the four keywords (building the 64-bit security code) stored in the security sector. If any mismatch is detected the respective protection feature remains active, the sector(s) remain(s) locked, and a protection error (PROER) is indicated in the Flash status register. In this case, a new Disable Sector Write Protection command or a Disable Read Protection command is only accepted **after the next system-reset**.

Security Feature Installation

The security features are installed by programming the following data (see **Figure 3-7**) into the security sector:

- Security control bits, selecting the security feature(s) to be installed
- 64-bit security code (four keywords)
- 16-bit confirmation code

Note: If any protection is enabled also the security sector itself is protected.

The security control bits can be checked via register PROCON. The same bit-layout must be used when programming the security control bits.

PROCON
Protection Control Register **SFR (FF'F004$_H$)** **Reset Value: xxxx$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RPRO	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**SL3**	**SL2**	**SL1**	**SL0**
rh	-	-	-	-	-	-	-	-	-	-	-	rh	rh	rh	rh

Field	Bits	Type	Description
RPRO	15	rh	**Read/Write Protection Configuration** 0 No general protection installed 1 General read/write protection is installed
SLn **(n = 3 … 0)**	3, 2, 1, 0	rh	**Sector Lock Bit n** 0 Sector is unprotected 1 Write protection installed for sector n *Note: Each two 8-Kbyte sectors are combined to a 16-Kbyte region that can be jointly locked by bits SL1 and SL0.*

Note: The security configuration can be checked by reading register PROCON. To modify the security configuration the security sector must be modified.

Note: In case of the 64K Flash device the non existing second 64K sector "5" (controlled by the bit SL3) must be kept in the "write protected" state if any other sector "0" to "4" shall be write protected.
In case of the 32K Flash device the non existing second 64K sector "5" (controlled by the bit SL3) must be kept in the "write protected" state if any other sector "0" to "3" shall be write protected. The bit SL2 is, in this case, irrelevant.

The 64-bit security code (e.g. 494E’4649’4E45’4F4E$_H$) must be correctly entered for commands that temporarily disable security features. Any failure to enter all four words correctly aborts the command and freezes the current security state until the next system reset.

The 16-bit confirmation code (8AFE$_H$) is required to validate the security feature installation. The installed configuration can be verified prior to validating it.

The security information and the confirmation code are stored in separate wordlines so they can be programmed and erased independently from each other.

Each byte of the security information is stored three times and completed with a zero-byte, so each 16-bit word to be stored uses the space of two doublewords (see example in **Figure 3-7**). All three copies of a data byte are used for evaluation which provides extreme reliability.

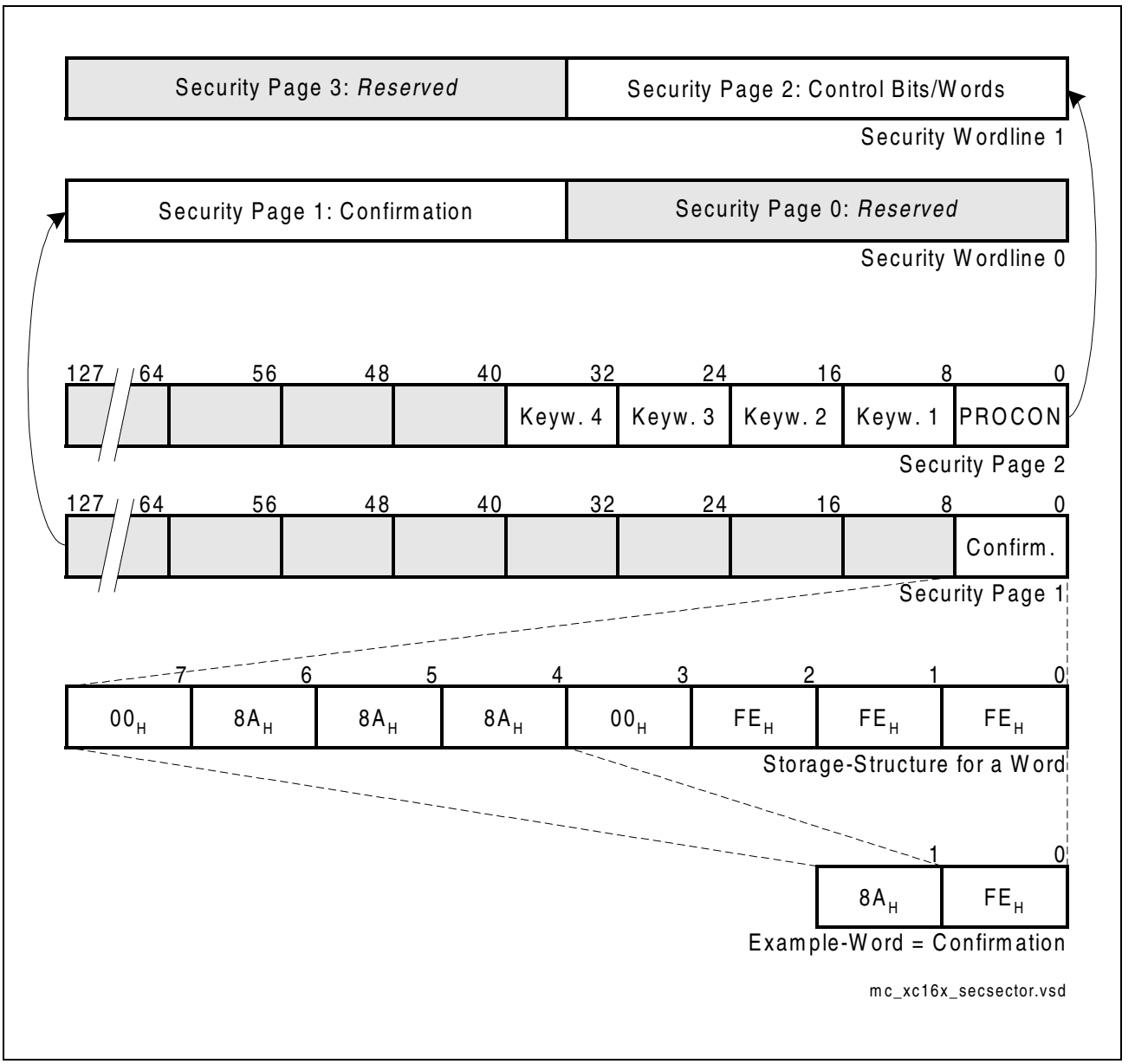


Figure 3-7 Security Sector Structure

Whenever the security configuration is modified (installation, modification, de-installation) the following procedure should be performed:

- Clear confirmation code by erasing security wordline 0.
  This uninstalls all protection features (PROIN = 0).
- Erase security wordline 1.
- Program the intended configuration and keywords into security page 2.
- Verify the programmed configuration and keywords.
- Program the confirmation code into security page 1.
  This installs the new protection features.

Following these steps prevents dead-locks resulting for example from programming erroneous keywords (e.g. due to power problems during programming) with existing confirmation code. The security features would be immediately active in this case whereas the erroneous keywords are not known.

Read/Write Protection Control

Read protection can be activated for code fetches and data reads separately via the control bits DCF (Disable Code Fetch) and DDF (Disable Data Fetch) in register IMBCTR. Read accesses are blocked as long as the respective disable flag (DCF, DDF) is set **and** read protection is active, indicated by bit RPA (Read Protection Active) in register IMBCTR. An access to the protected Flash will deliver a dummy value of $1E9B_H$. While read protection is disabled (RPA = 0), bits DCF and DDF have no effect on read accesses.

After a reset starting execution out of the on-chip Flash module bits DCF and DDF are cleared. This enables all accesses while code is executed from a safe source. Bit DDF can be set by user software to prevent data reads from the Flash module while still enabling code execution.

After any other reset (including boot mode) both bits are set (if protection is installed). By entering the 64-bit security code the read protection can be disabled temporarily by software executed out of external sources.

Note: Bits DCF and DDF can only be set via software, they cannot be cleared.

Attention: Be sure not to set DCF while executing out of on-chip Flash with read protection active.

Read/write protection is active (RPA = 1) if it has been installed (RPRO = 1) and is currently not disabled (PRODI = 0).

Read/Write Protection Handling

After reset, bit RPA indicates if the read/write protection is installed or not. User software can disable the read/write protection temporarily (indicated by RPA = 0). Bits DCF and DDF prevent Flash read accesses while RPA = 1. Because DCF and DDF are cleared after starting from the on-chip Flash memory, the user software is responsible for the protection handling.

If the read/write protection is enabled, the debug system is disabled to avoid not-authorized accesses to the Flash via the debug interface. Only if explicitly enabled by user software, the debug interface can be temporarily activated, even if the read/write protection is enabled.

The following rules ensure a safe read/write protection:

- no JUMPs or CALLs to external memory locations
- no execution of code loaded via any interface
- set DCF and DDF before transferring control to external locations (no return!)
- leave the debug system disabled

Note: Of course, external code can be executed intermediately while the read/write protection is disabled. Also the debug interface can be enabled, so protected devices can be debugged.
However, this should only be done after validation (e.g. by a specific security key), because read/write protection does not work during these phases.

3.8.5 Flash Status Information

The Flash Status Register FSR provides status information about all functions of the Flash module:

- Operating state
- Error conditions
- Security level

The FSR should be read before and after the execution of command sequences. The FSR cannot be written directly. The “Clear Status” command clears the error flags and the status flags PROG and ERASE, the “Reset to Read” command clears the error flags.

FSR
Flash Status Register **SFR (FF’F000$_H$)** **Reset Value: 0xxx$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	SUL	-	PROIN	PRODI	DBER	SBER	PROER	SQER	-	OPER	PAGE	ERASE	PROG	BUSY
-	-	rh	-	rh	rh	rh	rh	rh	rh	-	rh	rh	rh	rh	rh

Field	Bits	Type	Description
SUL	13	rh	**Sectors Unlocked** 0 Sectors are protected according to the installation 1 All sectors are temporarily unlocked (check general protection)
PROIN	11	rh	**Protection Installed** 0 No security features installed 1 General read/write protection and/or sector-specific write protection installed (see register PROCON)
PRODI	10	rh	**Protection Disabled** 0 General read/write protection active (if installed) 1 General read/write is temporarily disabled
DBER	9	rh	**Double Bit Error** (Cleared via “Clear status”, “Reset-to-read”) 0 No double bit error has occurred 1 A double bit error was detected (no correction possible)

Field	Bits	Type	Description
SBER	8	rh	**Single Bit Error** (Cleared via “Clear status”, “Reset-to-read”) 0 Read/fetch accesses executed without error 1 A single bit error was detected and automatically corrected
PROER	7	rh	**Protection Error** (Cleared via “Clear status”, “Reset-to-read”) 0 No protection error detected 1 Protection error has occurred: attempt to program/erase a locked sector or invalid security code[1)]
SQER	6	rh	**Command Sequence Error** (Cleared via “Clear status”, “Reset-to-read”) 0 No command sequence error detected 1 State machine operation aborted due to invalid command step *Note: SQER is not set when a command sequence is aborted with a “Reset to Read” command. SQER is set when a “Clear Status” command is attempted while the Flash module is busy (PROG or ERASE are not cleared).*
OPER	4	rh	**Operation Error** (Cleared via “Clear status”, “Reset-to-read”) 0 Flash operation successfully finished or currently in progress 1 Flash operation not successfully terminated (abortion)
PAGE	3	rh	**Page Mode** (Cleared via “Reset-to-read”) 0 Flash not in page mode 1 Flash in page mode, page buffer being filled *Note: Page mode can be active during standard read mode.*
ERASE	2	rh	**Erase State** (Cleared via “Clear status”, “Reset-to-read”) 0 There is no erase operation in progress 1 Flash busy with erase operation

Field	Bits	Type	Description
PROG	1	rh	**Programming State** (Cleared via “Clear status”, “Reset-to-read”) 0 There is no programming operation in progress 1 Flash busy with programming operation (write page)
BUSY	0	rh	**Flash Busy** 0 **Ready:** Flash command execution is completed. Module is in standard read mode. 1 **Busy:** Embedded algorithm for command execution is in progress or Flash module is in ramp-up state[2]. Module not in read mode.

1) After the occurrence of a protection error the next password sequence is only accepted after a reset.

2) After a system reset BUSY will be active for approx. 250 µs until the internal voltages have settled.

Note: By evaluating bits PROG and ERASE together with bits BUSY and OPER the control software can determine if an operation is in progress, has terminated, or has been aborted.

3.8.6 Operation Control and Error Handling

Command execution is started with the last command of the respective command sequence and is indicated by the respective state flag (PROG for programming, ERASE for erasing) as well as by the summarizing BUSY flag. While polling BUSY is sufficient to detect the end of a command execution it is recommended to check the error flags afterwards to find erroneous operations.

The following general structure for command execution is recommended:

- Clear status
- Write command sequence to Flash module
- Ensure correct sequence by checking bits SQER and PROER
- If error: clear flags via “Clear Status” or “Reset” and act upon it (e.g. with a retry operation)
- Check for the correct command by polling bits PROG and ERASE
- Poll BUSY to determine the command termination
- Check error flags

The error bits in status register FSR are registered bits (flipflops) and indicate a fault condition as long as the error bit is set. It is therefore necessary to clear the error flags by commands.

Table 3-6 gives examples of software actions to be taken after a specific error has been detected:

Table 3-6 Software Reactions to Error Conditions

Detected Error	Fault Condition	Software Reaction
SQER Sequence Error	Wrong register address, wrong command/sector/wordline address, wrong command code, illegal command sequence	Check address or code and repeat with correct values
OPER Operation Error	Aborted programming or erase operation due to SW reset, WDT reset, or warm HW reset	Repeat Flash operation (PROG and ERASE indicate the failed operation)
PROER Protection Error	Begin of write operation (Enter Page Mode) to protected sector, General password failure	Retry operation after disabling protection, Retry operation after reset
SBER Single Bit Error	The Error Correction Code (ECC) has revealed a single bit error	Refresh faulty wordline (see **Section 3.8.3**)
DBER[1)] Double Bit Error	The Error Correction Code (ECC) has revealed a double bit error	Double bit error triggers a trap

1) Does not occur if a refresh operation is executed after a single bit error (see **Section 3.8.3**).

Reset and Power-Down Processing

Upon a reset the Flash module resets its state machine and enters the standard read mode after the internal voltages have stabilized. The internal voltages need to ramp up (e.g. after power down) or to ramp down (e.g. after an interrupted programming or erase operation). This power stabilization phase is indicated by flag BUSY. Accesses during the power stabilization phase are delayed until power has stabilized.

The Flash module is requested to ramp down its internal voltages by entering Power Down mode, Sleep mode or Idle mode (with Flash off), by disabling it via SYSCON3, or by executing a software reset. After completing execution and termination of the running operation (including program or erase operation) the request is acknowledged and the CPU can complete the intended action.

Note: The delay caused by the stabilization phase must also be considered when calculating delays for wake-up from idle, sleep, or power down states.

3.9 Program Memory Control

The internal program memory block IMB consists of an interface part (program memory interface PMI) to control the accesses to the memories and the following memory blocks:

- Internal program Flash memory (including error correction ECC), starting at address $C0'0000_H$
- 2 Kbytes program SRAM, starting at address $E0'0000_H$

The Flash memory block and the program SRAM block can contain the program code, but can also store data, which can be accessed by the CPU.

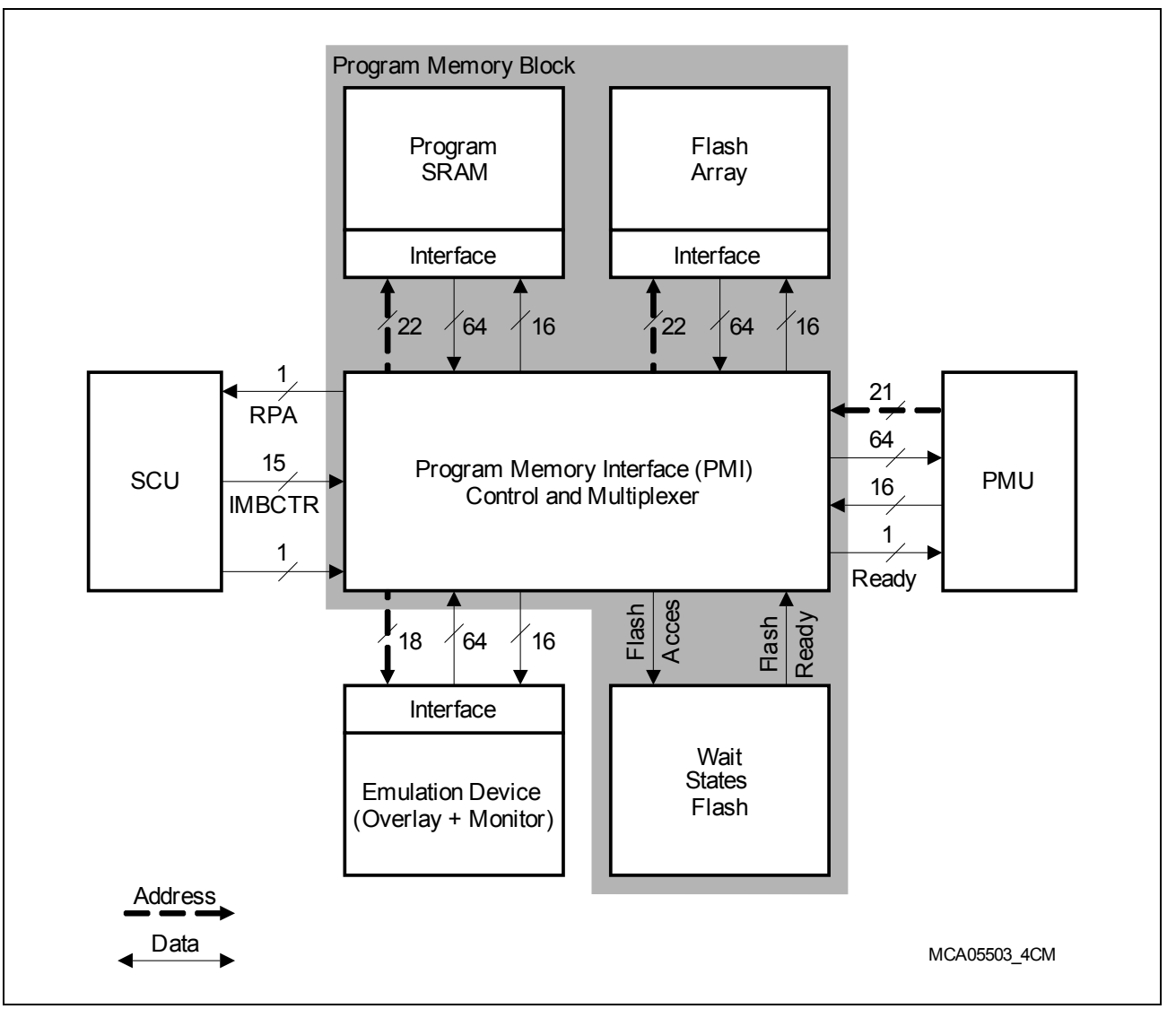


Figure 3-8 Overview of the Internal Program Memory Block IMB

Figure 3-8 shows the main blocks of the IMB, specific control signals are not mentioned for simplicity reasons.

The behavior of the memories is adaptable to the requirements of the application. If the program is executed from the on-chip Flash memory, or from the internal SRAM, the

latencies have to be identical in some cases. To solve this problem, the access times of the SRAM can be programmed to be equal to the Flash timings. In the best case, the internal SRAM will allow single cycle accesses. A programmable wait state generation logic is part of the program memory interface (PMI) inside the IMB.

The number of access cycles can be programmed independently for the SRAM and the Flash memory.

3.9.1 Address Map

The address map of the program memory blocks is shown in **Figure 3-9**.

The program Flash memoryalways starts at address C0'0000$_H$ and the program SRAM at address E0'0000$_H$.

The read-only Flash status registers can be accessed starting at address FF'F000$_H$. Any write access to this address range must be avoided.

The access to addresses, which are not explicitly mentioned as valid memory/register area is forbidden.

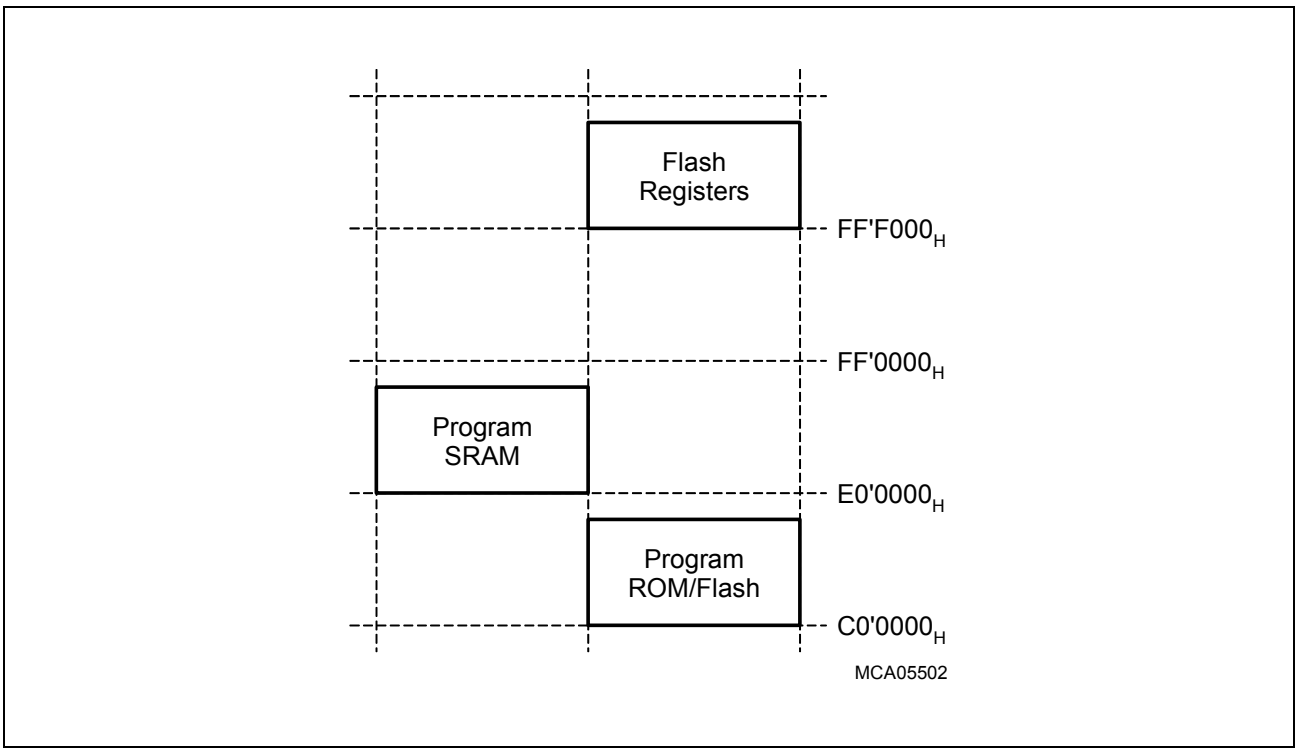


Figure 3-9 Address Map of the Program Memory Block IMB

3.9.2 Flash Memory Access

The internal functional structure of the interface between the PMU/PMI and the Flash memory is shown in **Figure 3-10**. The access is done in two phases:

- The Flash array delivers the accessed data within a fixed time of 50 ns maximum. The duration of the first access phase (1+WS) must cover the Flash Array's access time. Waitstates must be selected accordingly (bitfield WSFLASH in register IMBCTRL).
  Example: Operating at 40 MHz results in a cycle time of 25 ns. Therefore, the access phase requires 2 cycles, so one waitstate must be selected (1+1).
- The error correction (ECC) and the PMU require one additional clock cycle each.

The CPU receives requested data after 1+WS+2 cycles (4 cycles if 1 WS is selected). However, this delay only becomes effective for an isolated access (read from a non-linear address). A prefetching mechanism overlaps phase 1 of a subsequent access with phase 2 of the previous access, so the sustained performance for linear accesses (e.g. code fetches) is considerably higher.

Flash accesses can be serviced every 1+WS cycles, because the Flash array itself only requires phase 1.

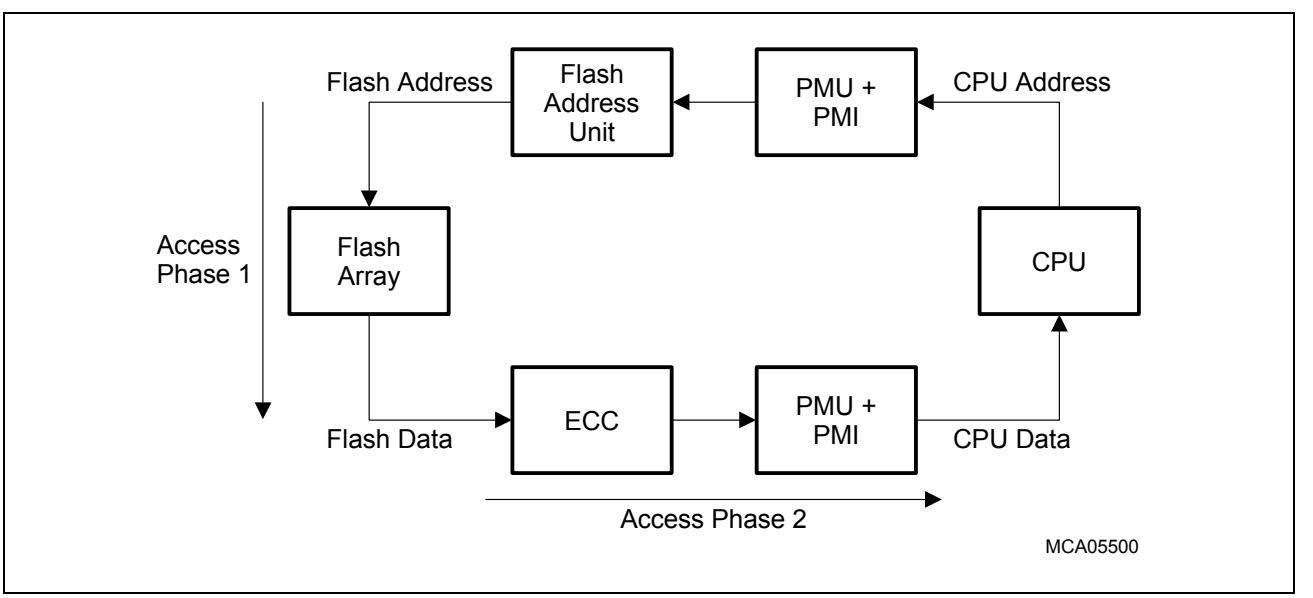


Figure 3-10 Flash - PMI Structure

Example for Flash Accesses with one Wait State (WS = 1)

After the first access (e.g. after a jump to the first address delivered by the CPU), four clock cycles are necessary to fetch the corresponding data (1+1+2).

This leads to the following sequence of clock cycles between the delivery of subsequent data words: 4 - 2 - 2 - 2 - 2 - …

In the case that the CPU requests another address than the one proposed by the prefetcher (e.g. in case of a jump), the Flash address unit immediately changes to the new address and begins a new sequence (4 - …).

Note: If the Flash access phase takes more than two cycles (more than 1 WS), prefetch accesses make no sense, so the Flash prefetching mechanism is disabled.

3.9.3 IMB Control Functions

Wait State Generation

The generation of wait states is handled by a wait state unit, which indicates when the requested data (or instruction word) is available. The address window for the Flash memory starts at address C0'0000$_H$ and selects an address range of 2 Mbytes.

The reset value defines a two cycle Flash memory access. The program SRAM can be accessed with a one cycle read.

IMB Control Register

Register IMBCTR contains the bitfields controlling the wait state generation for the Flash memory and the other IMB memory blocks. One wait state represents one clock cycle. The wait states have to be introduced in order to adapt the memory access time in clock cycles (depending on the clock frequency) to the Flash access time. User PSRAM can be accessed using the Flash timing, e.g. for emulation purposes.

This register is protected against undesired modification by the register security mechanism. This register is only reset by a hardware reset, a SW reset or a WDT reset do not change the bits.

IMBCTR
IMB Control Register **ESFR (F0FE$_H$/7F$_H$)** **Reset Value: xx01$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RPA	-					DDF	DCF	-				WS ROM	WS RAM	WS FLASH	
rh	-					rwh	rwh	-				rw	rw	rw	

Field	Bits	Type	Description
RPA	15	rh	**Read Protection Activated** This bit monitors the status of the Flash-internal read protection. 0 The Flash-internal read protection is not activated. Bits DCF, DDF are not taken into account. 1 The Flash-internal read protection is activated. Bits DCF, DDF are taken into account.

Field	Bits	Type	Description
DDF	9	rwh	**Disable Data Read from Flash Memory** This bit enables/disables the data read access from the internal Flash memory area. Once set, this bit can only be cleared by a HW reset. 0 The data read access from the Flash memory area is allowed. 1 The data read access from the Flash memory area is not allowed. This bit is not taken into account while RPA = 0.
DCF	8	rwh	**Disable Code Fetch from Flash Memory** This bit enables/disables the code fetch from the internal Flash memory area. Once set, this bit can only be cleared by a HW reset. 0 The code fetch from the Flash memory area is allowed. 1 The code fetch from the Flash memory area is not allowed. This bit is not taken into account while RPA = 0.
WSROM	3	rw	**Wait State Control for User ROM Access** This bit defines the behavior of a memory in the user ROM area in the IMB for a read access. This memory area is located in the address range from C0'0000$_H$ to DF'FFFF$_H$. The write access to this memory area is not possible. 0 The user ROM area is accessed with the maximum access speed, which is a single cycle read access. 1 The user ROM behaves exactly like the user Flash. The pipelined structure and the access time are taken into account to rebuild the identical behavior. *Note: Bit WSROM is only available in the ROM-derivatives of the XC164CM, of course.*

Field	Bits	Type	Description
WSRAM	2	rw	**Wait State Control for Program RAM Access** This bit defines the behavior of a memory in the program SRAM area in the IMB for a read access. This memory area is located in the address range from $E0'0000_H$ to $F7'FFFF_H$. The write access to this memory area is always handled within one clock cycle for the memory. 0 The program SRAM area is accessed with the maximum access speed, which is a single cycle read access. 1 The program SRAM behaves exactly like the user Flash. The pipelined structure and the access time are taken into account to rebuild the identical behavior.
WSFLASH	[1:0]	rw	**Wait States for the Flash Memory** This bitfield defines the number of additional wait states, which are added for a read access from the Flash memory area, which is located in the address range from $C0'0000_H$ to $DF'FFFF_H$. 00 No additional wait state is introduced for the Flash read access. This corresponds to a Flash read access in one clock cycle. 01 One additional wait state is introduced for the Flash read access. This corresponds to a Flash read access in two clock cycles. (default) 10 Two additional wait states are introduced for the Flash read access. This corresponds to a Flash read access in three clock cycles. 11 Three additional wait states are introduced for the Flash read access. This corresponds to a Flash read access in four clock cycles.

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 XC164CM 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 XC164CM, up to five instructions can be processed in parallel. Most instructions of the XC164CM 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.

Note: In the following description, the term “external access” refers to accesses to the resources controlled by the EBC, that is to the external bus and/or to the internal LX-bus.

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 XC164CM 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 normal operation state, the CPU has the following particular states:

- **Reset state:** Any reset (hardware, software, watchdog) forces the CPU into a predefined active state.
- **IDLE state:** The clock signal to the CPU itself is switched off, while the clocks for the on-chip peripherals keep running.
- **SLEEP state:** All of the on-chip clocks are switched off (RTC clock selectable), external interrupt inputs are enabled.
- **POWER DOWN state:** All of the on-chip clocks are switched off (RTC clock selectable), all inputs are disregarded.

Transition to an active CPU state is forced by an interrupt (if in IDLE or SLEEP mode) or by a reset (if in POWER DOWN mode).
The IDLE, SLEEP, POWER DOWN, and RESET states can be entered by specific XC164CM system control instructions.

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.

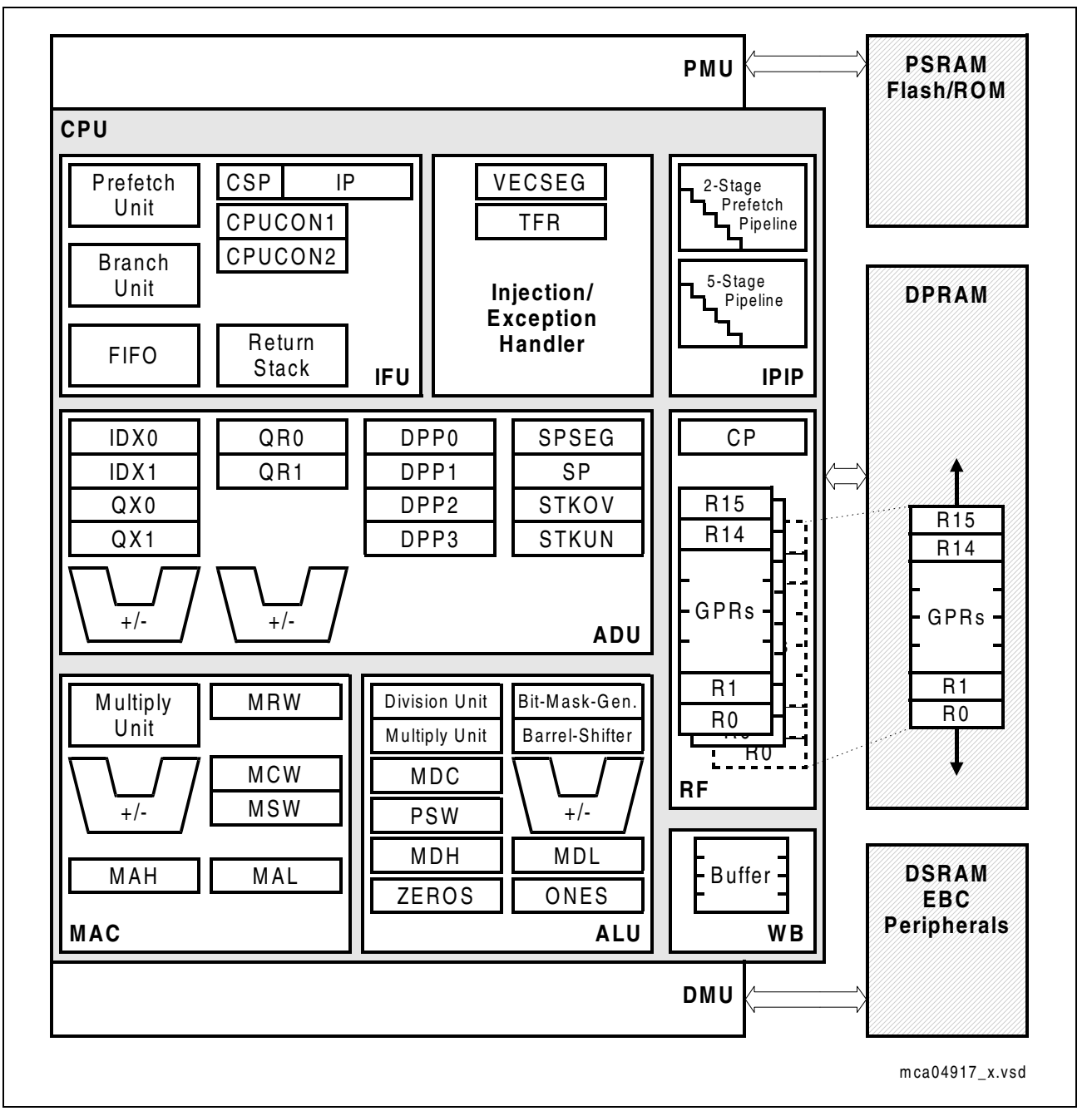


Figure 4-1 CPU Block Diagram

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.

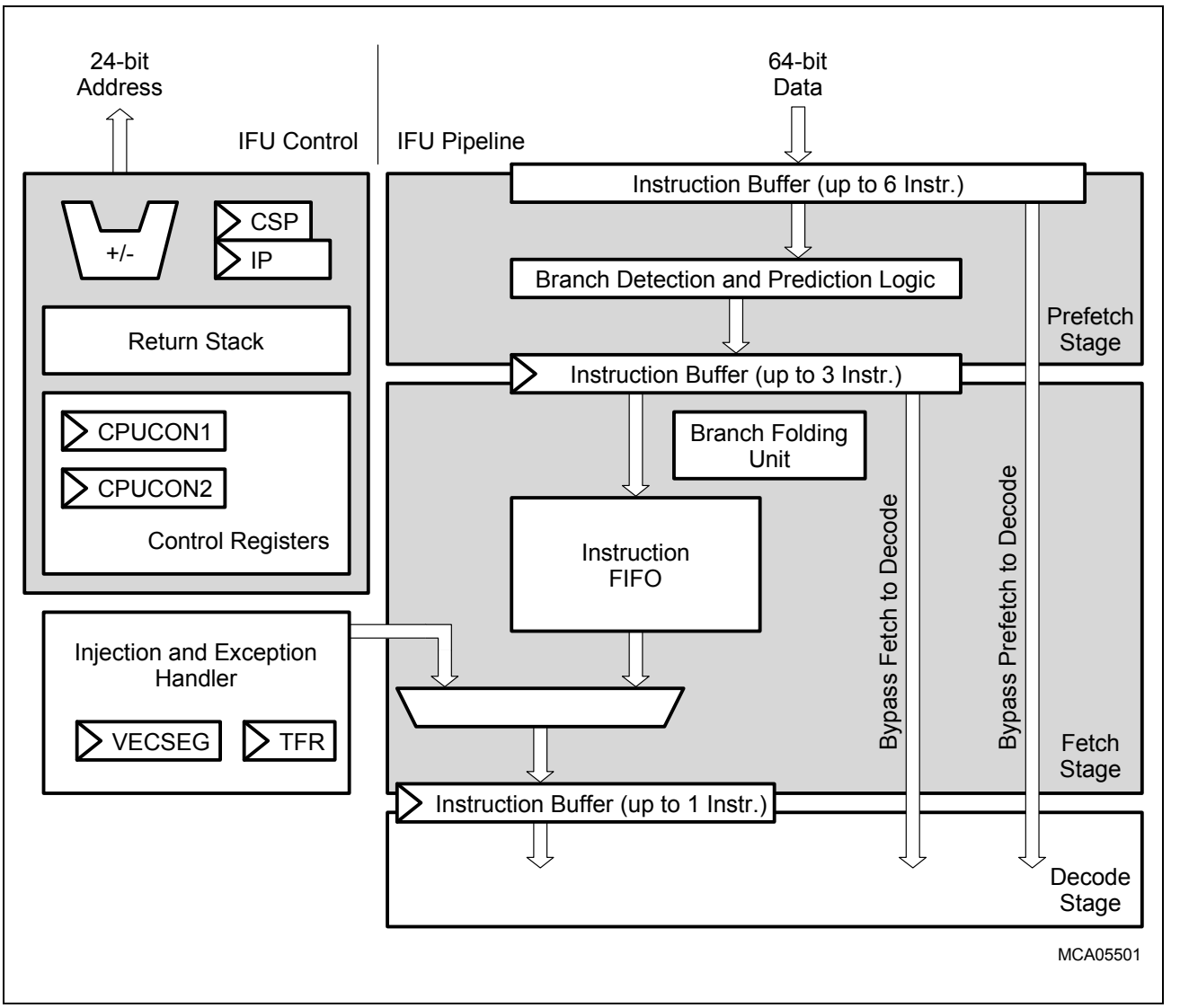


Figure 4-2 IFU Block Diagram

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[1] 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.

1) Flash memory may require waitstates, depending on the actual operating frequency.

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}

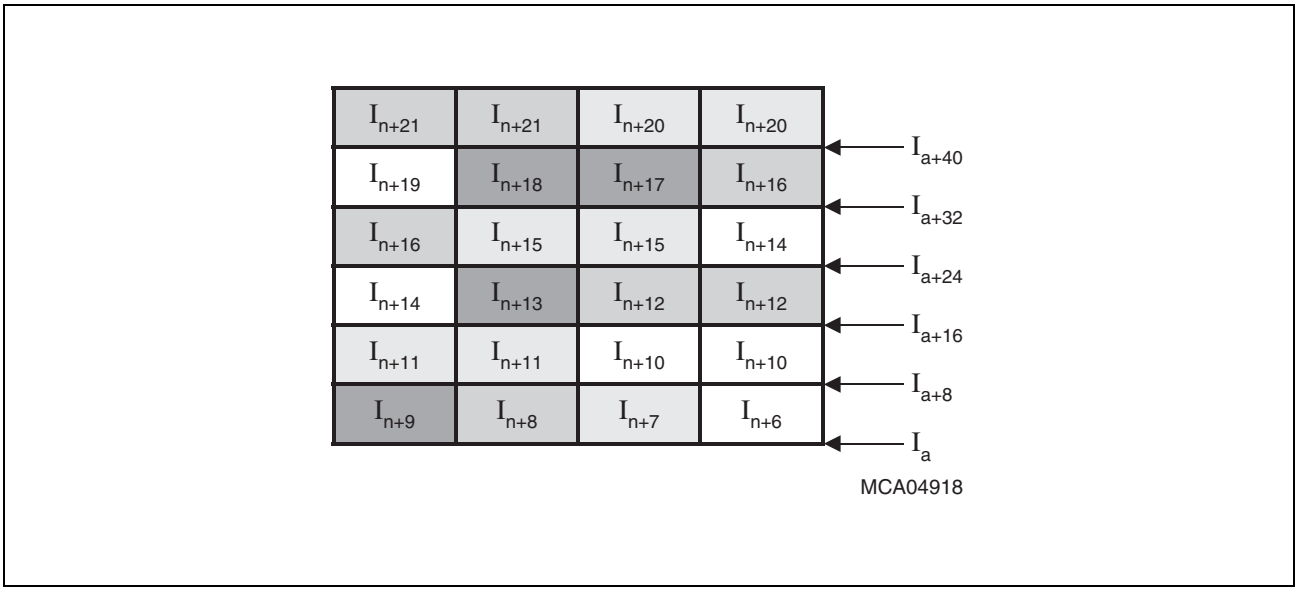


Figure 4-3 Program Memory Section for Correctly Predicted Flow

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.

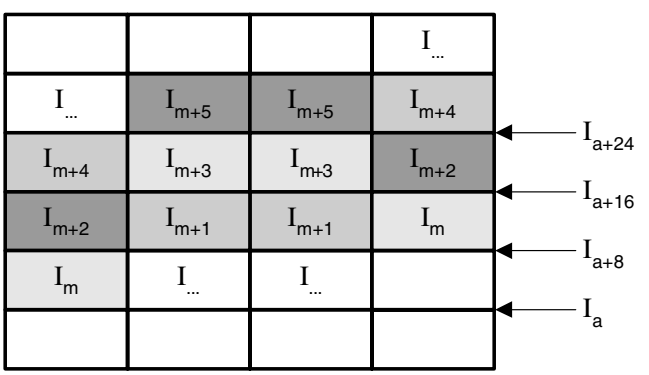
Table 4-3 shows the restarted execution of instructions, assuming a 0-waitstate program memory. **Figure 4-4** shows the corresponding program memory section.

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

MCA04919

Figure 4-4 Program Memory Section for Incorrectly Predicted Flow

4.3 Instruction Processing Pipeline

The XC164CM 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 XC164CM'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 XC164CM 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:
In    ADD R0,R1      ;Compute new value for R0
In+1  ADD R6,R0      ;R0 is not updated, just read
In+2  ADD R6,R1
In+3  MOV R3,[R0]    ;Use R0 as address pointer
In+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:
In    ADD R3,[R0]      ;R0 points to DPRAM (e.g.)
In+1  MOV R0,R4
...
Ii    MOV DPPX, ...    ;change DPPx
...
Im    ADD R6,[R0]      ;R0 now points to SRAM (e.g.)
Im+1  MOV R6,R1
Im+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:
In     MOV PSW,R4
In+1   MOV R6,R1
In+2   ADD R6,R1
In+3   MOV R3,[R0]
In+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 XC164CM'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

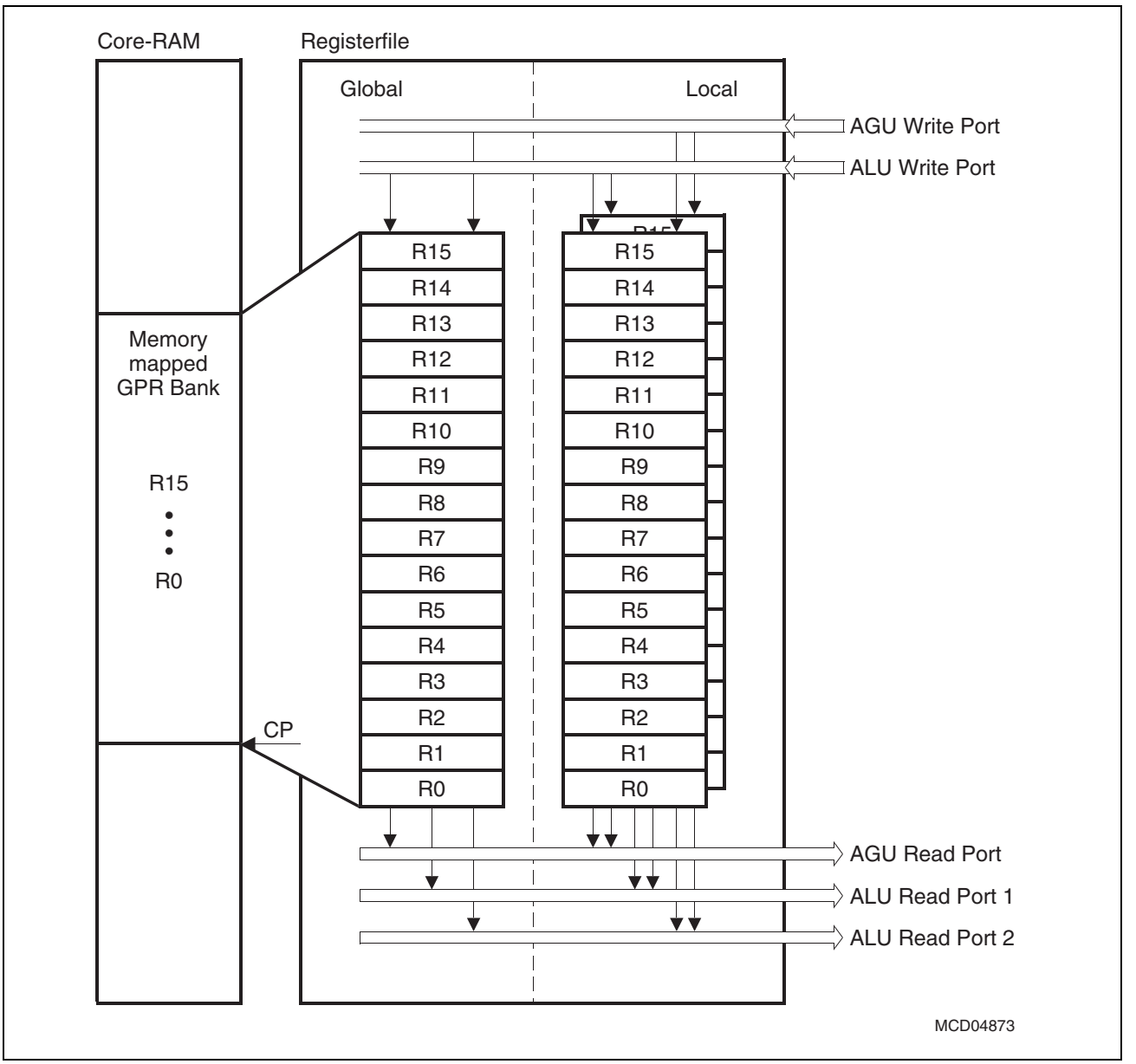


Figure 4-5 Register File

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.

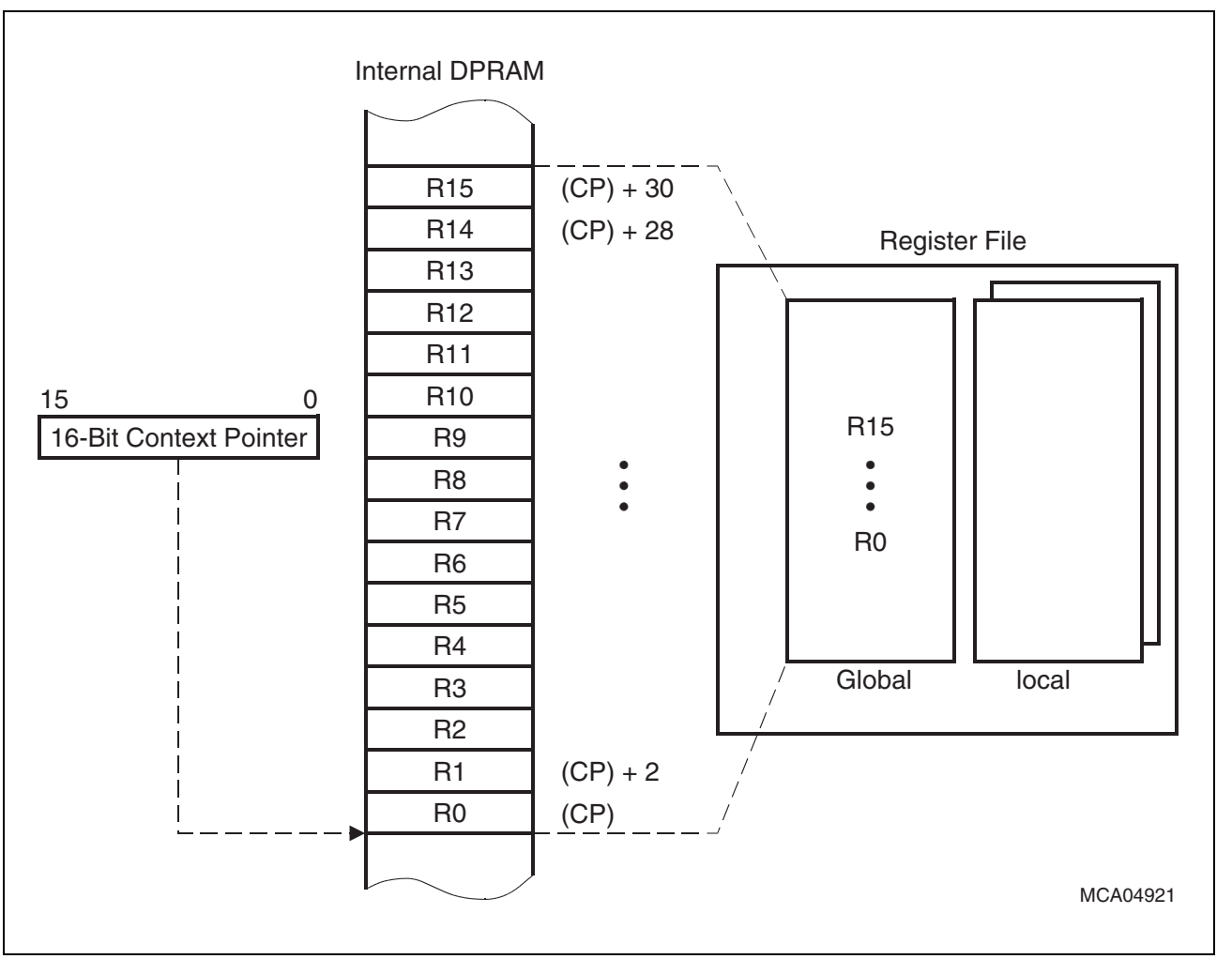


Figure 4-6 Register Bank Selection via Register CP

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.

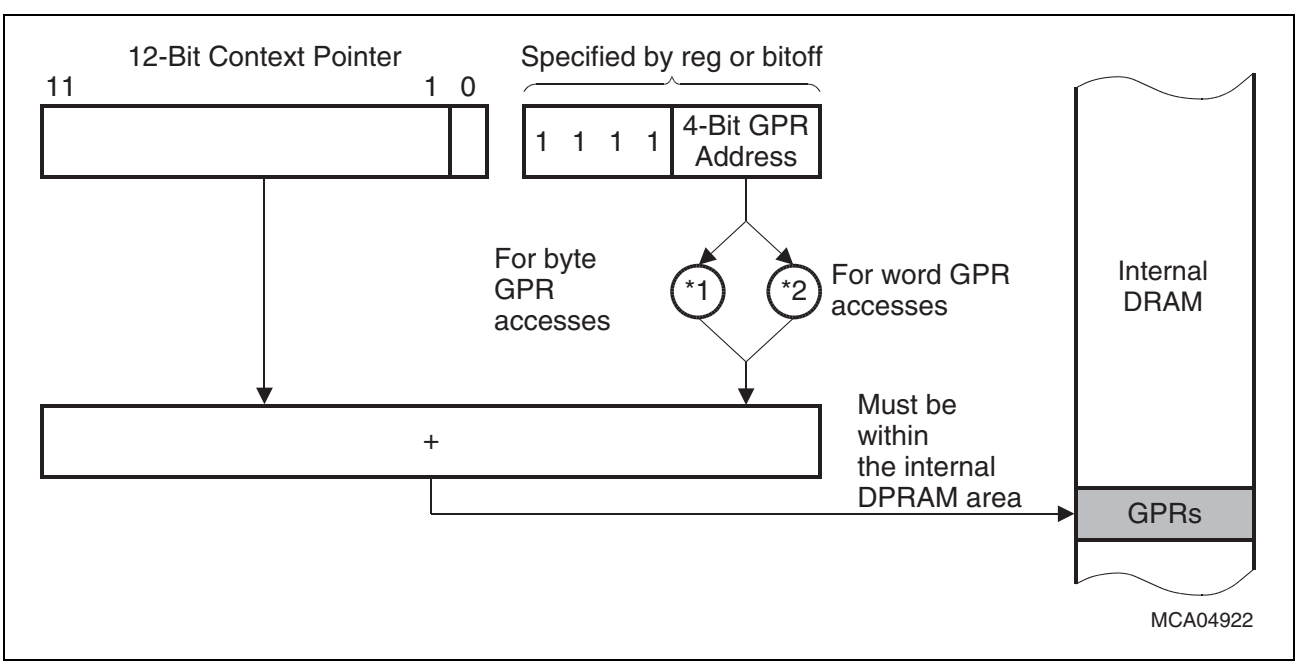


Figure 4-7 Implicit CP Use by Logical Short GPR Addressing Modes

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.

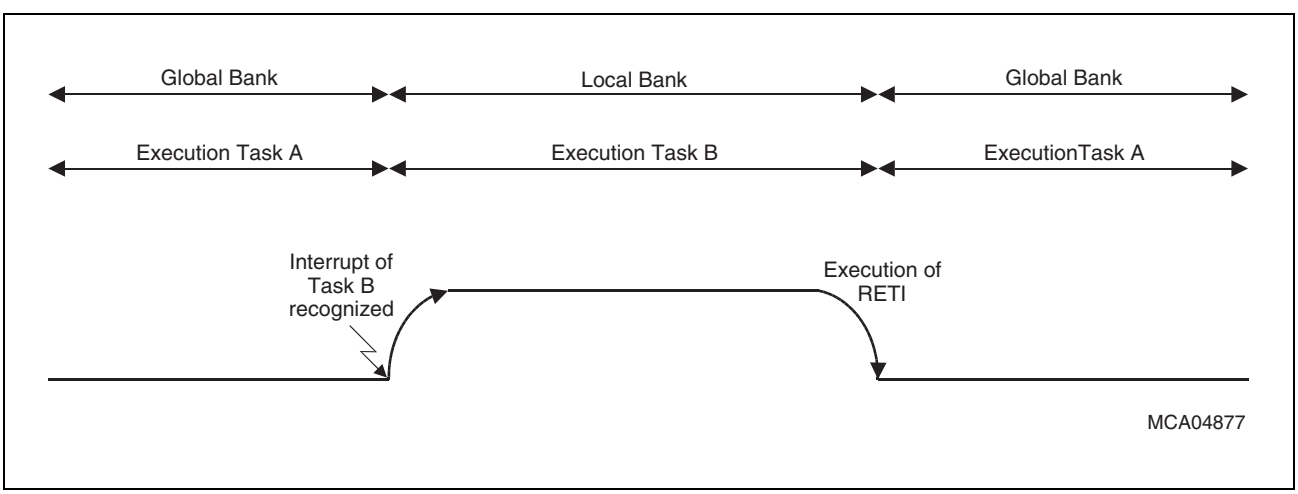


Figure 4-8 Context Switch by Changing the Physical Register Bank

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 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**).*

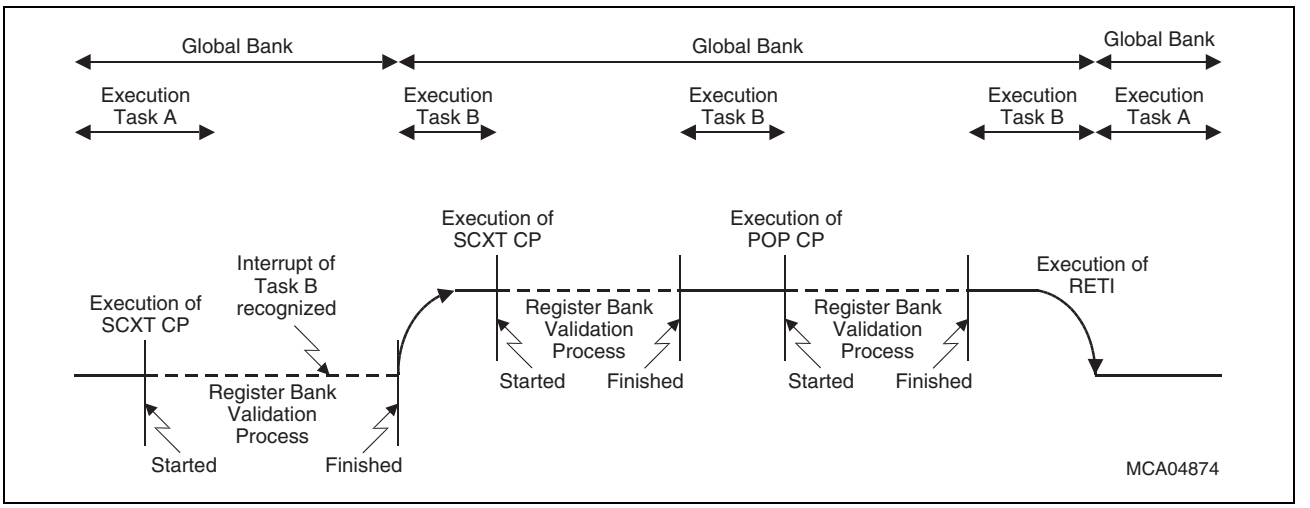


Figure 4-9 Validation Process Interrupted by Global-Bank Interrupt

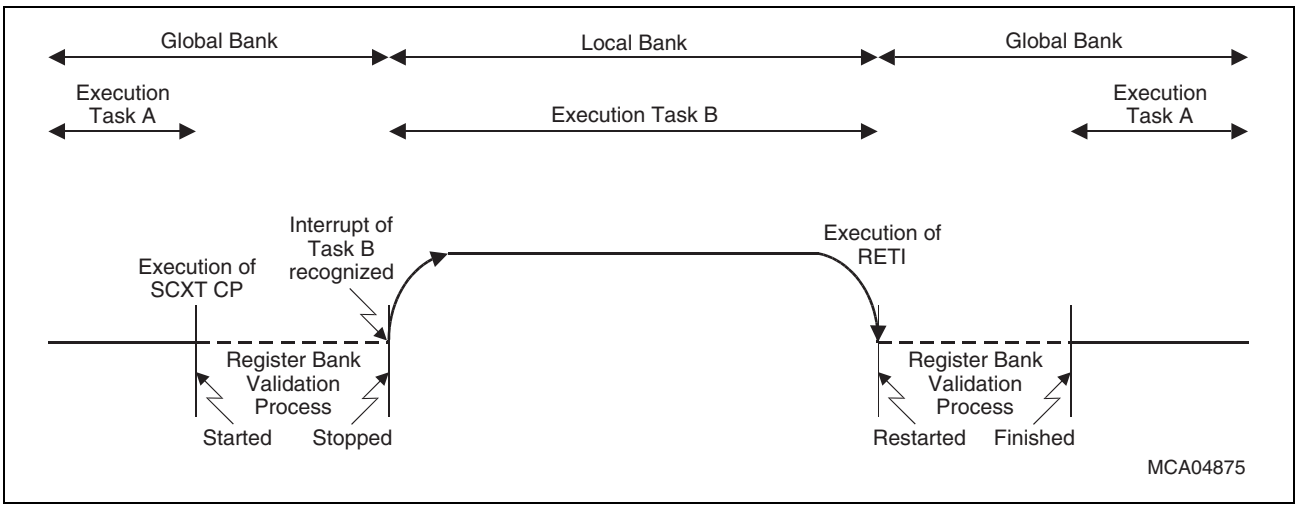


Figure 4-10 Validation Process Interrupted by Local-Bank Interrupt

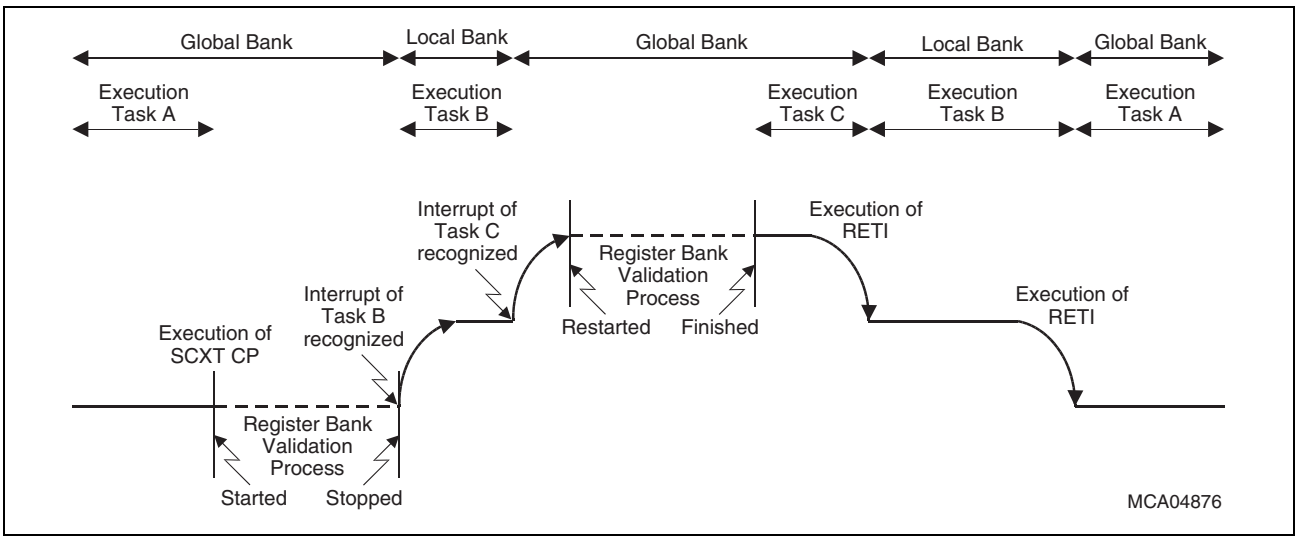


Figure 4-11 Validation Process Interrupted by Local- and Global-Bank Intr.

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.

The XC164CM 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 XC164CM 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.

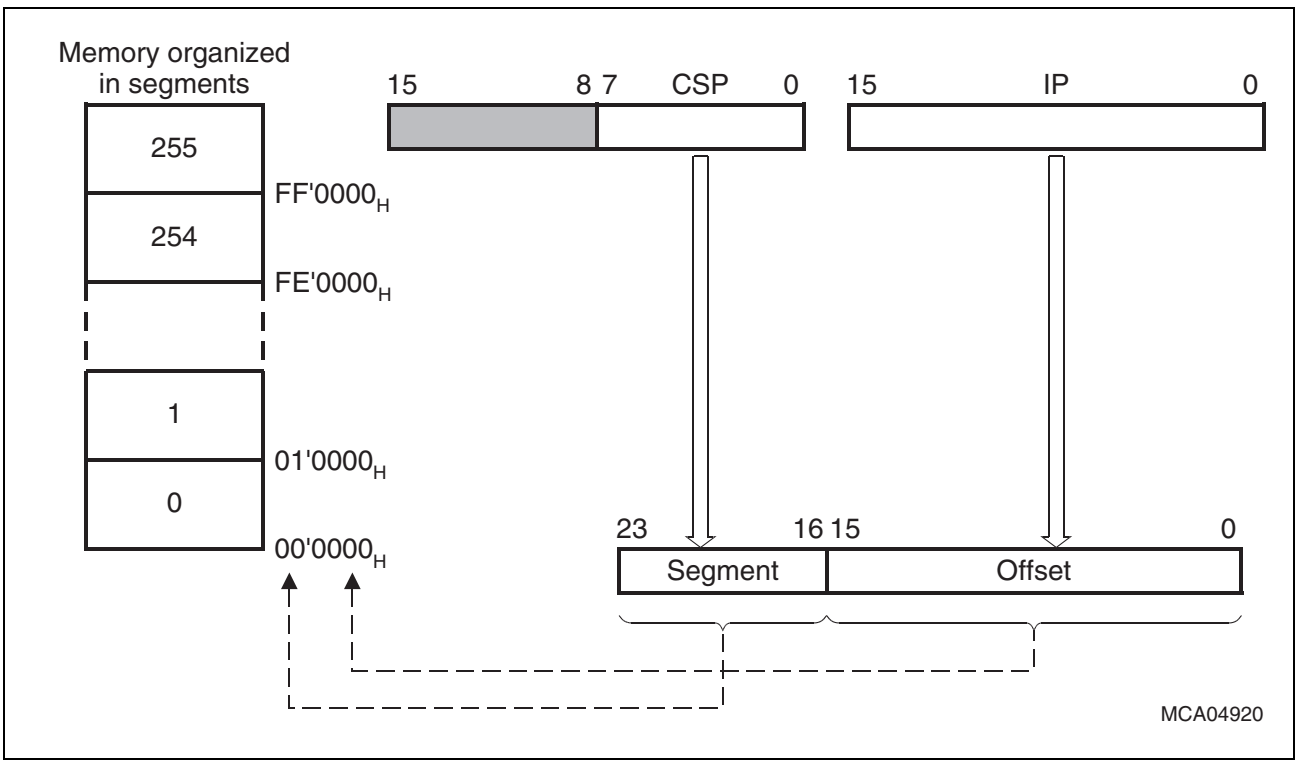


Figure 4-12 Addressing via the Code Segment and Instruction Pointer

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.

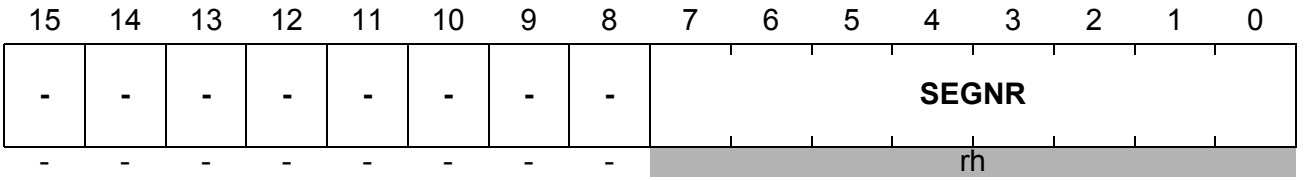
In segmented memory mode (default after reset), register CSP is modified either directly by JMPS and CALLS instructions, or indirectly via the stack by RETS and RETI instructions.

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: $xxxx_H$**

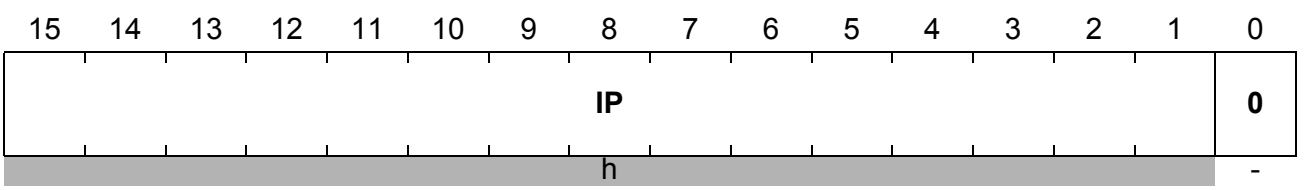


Field	Bits	Type	Description
SEGNR	[7:0]	rh	Specifies the code segment from which the current instruction is to be fetched.

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 XC164CM'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**



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>.
0	0	–	IP is always word-aligned.

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.

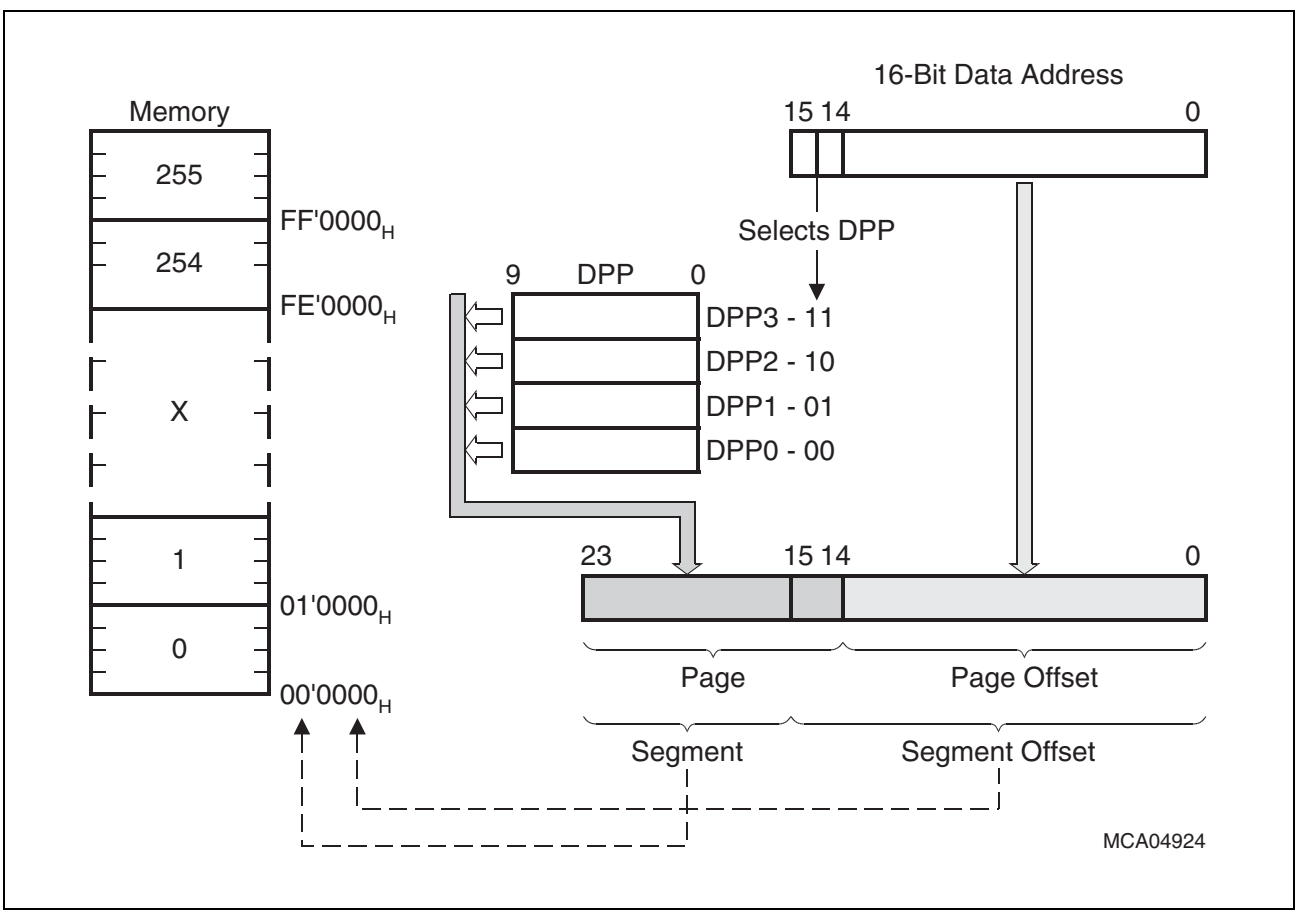


Figure 4-13 Data Page Pointer Addressing

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**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	DPP0PN									
-	-	-	-	-	-	rw									

DPP1
Data Page Pointer 1 **SFR ($FE02_H/01_H$)** **Reset Value: 0001_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	DPP1PN									
-	-	-	-	-	-	rw									

DPP2
Data Page Pointer 2 **SFR ($FE04_H/02_H$)** **Reset Value: 0002_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	DPP2PN									
-	-	-	-	-	-	rw									

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
-	-	-	-	-	-	DPP3PN									
-	-	-	-	-	-	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.

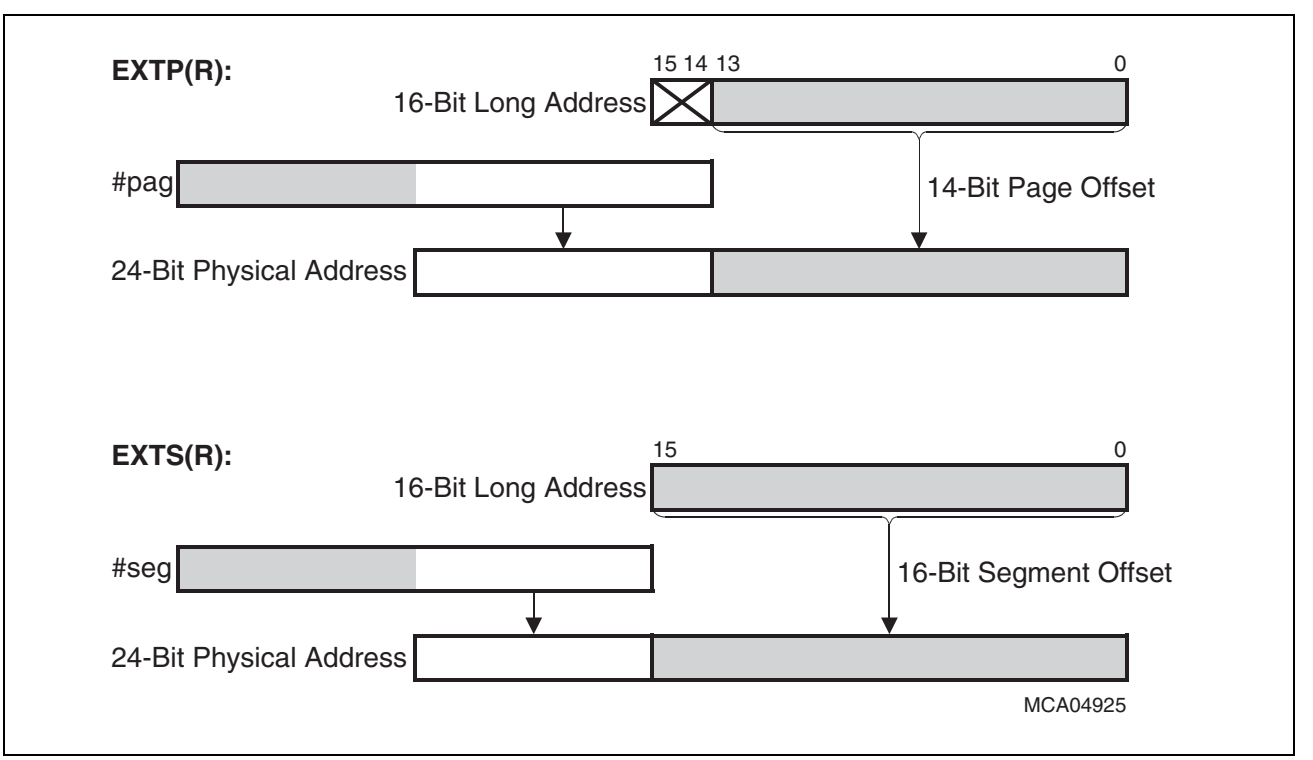


Figure 4-14 Overriding the DPP Mechanism

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.

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**

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

QR1
Offset Register **ESFR ($F006_H/03_H$)** **Reset Value: 0000_H**

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

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).

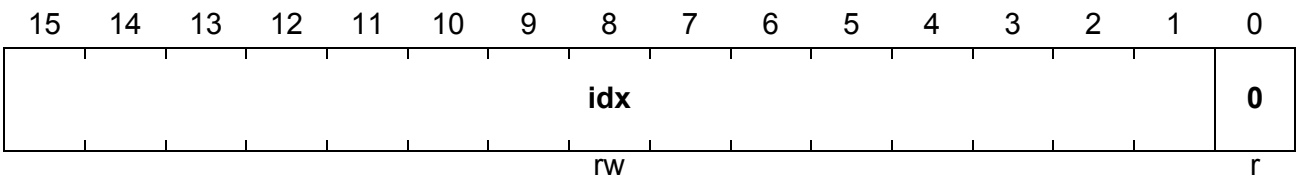
4.7.4 DSP Addressing Modes

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.

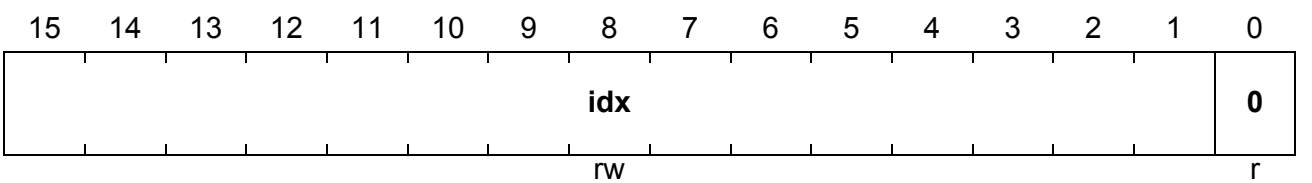
The address pointers can be used for arithmetic operations as well as for the special CoMOV instruction. The generation of the 24-bit memory address is different:

- 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**

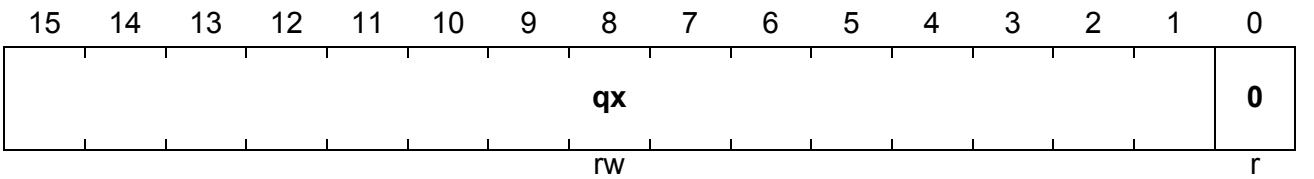


Field	Bits	Type	Description
idx	[15:1]	rw	**Modifiable Portion of Register IDXx** Specifies the 16-bit word address pointer

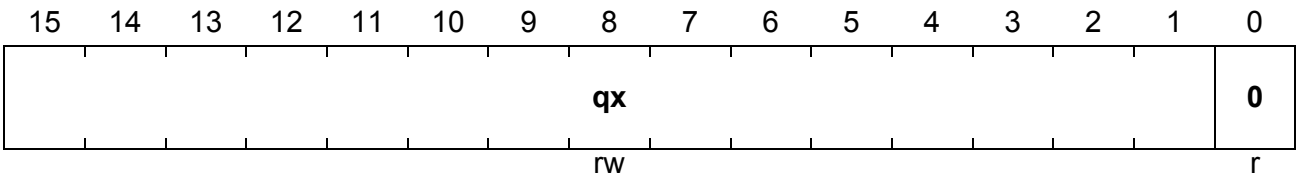
*Note: During the initialization of the IDX registers, instruction flow stalls are possible. For the proper operation, refer to **Section 4.3.4**.*

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**.*

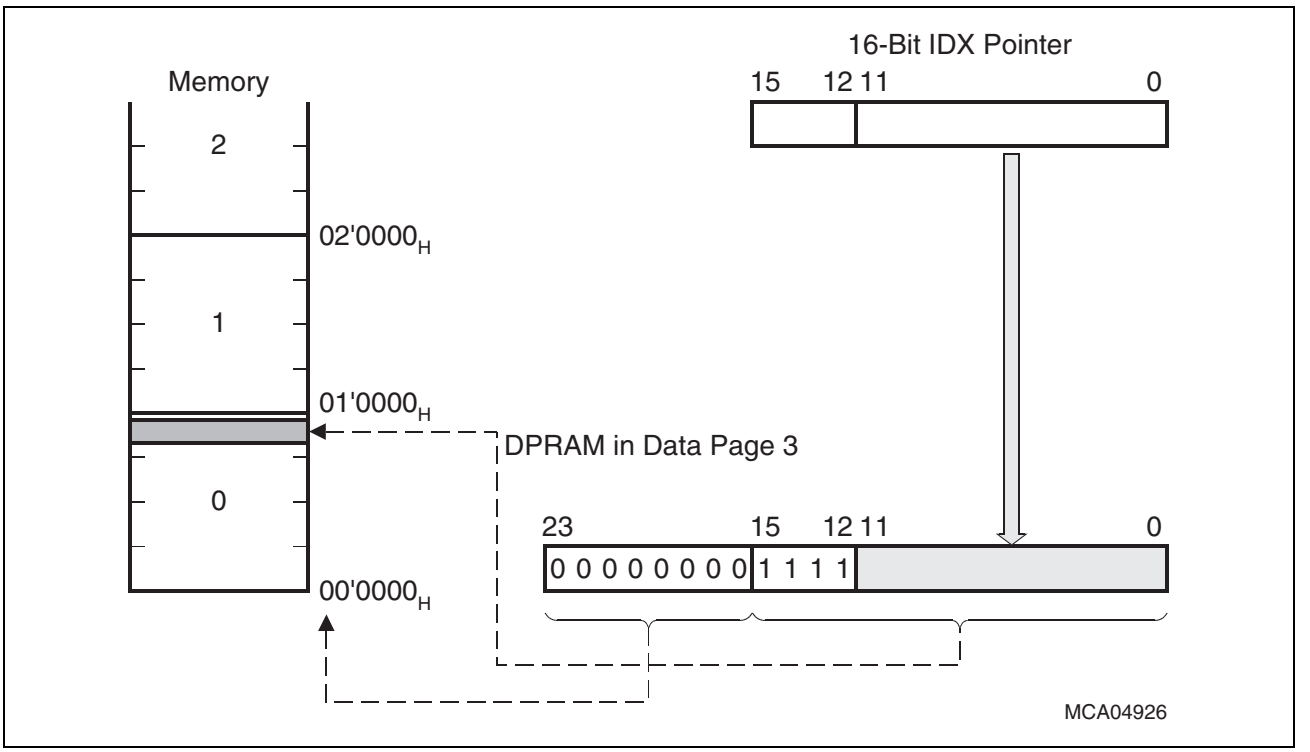


Figure 4-15 Arithmetic MAC Operations and Addressing via the IDX Pointers

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**).

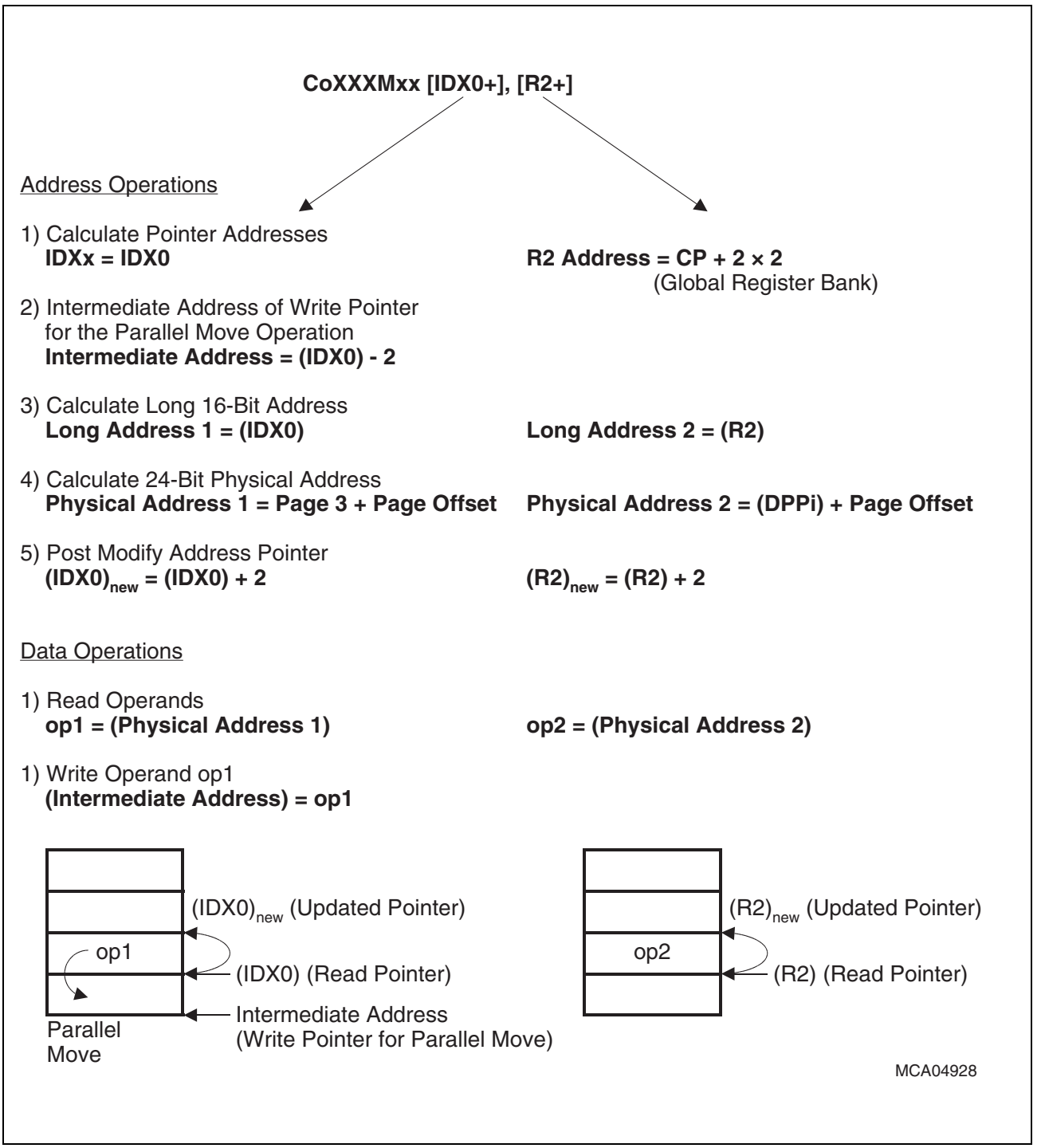


Figure 4-16 Arithmetic MAC Operations with Parallel Move

4.7.5 The System Stack

The XC164CM 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 (SPSEG). 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).

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 SPSEG, as shown in **Figure 4-17**.

The system stack cannot cross a 64-Kbyte segment boundary.

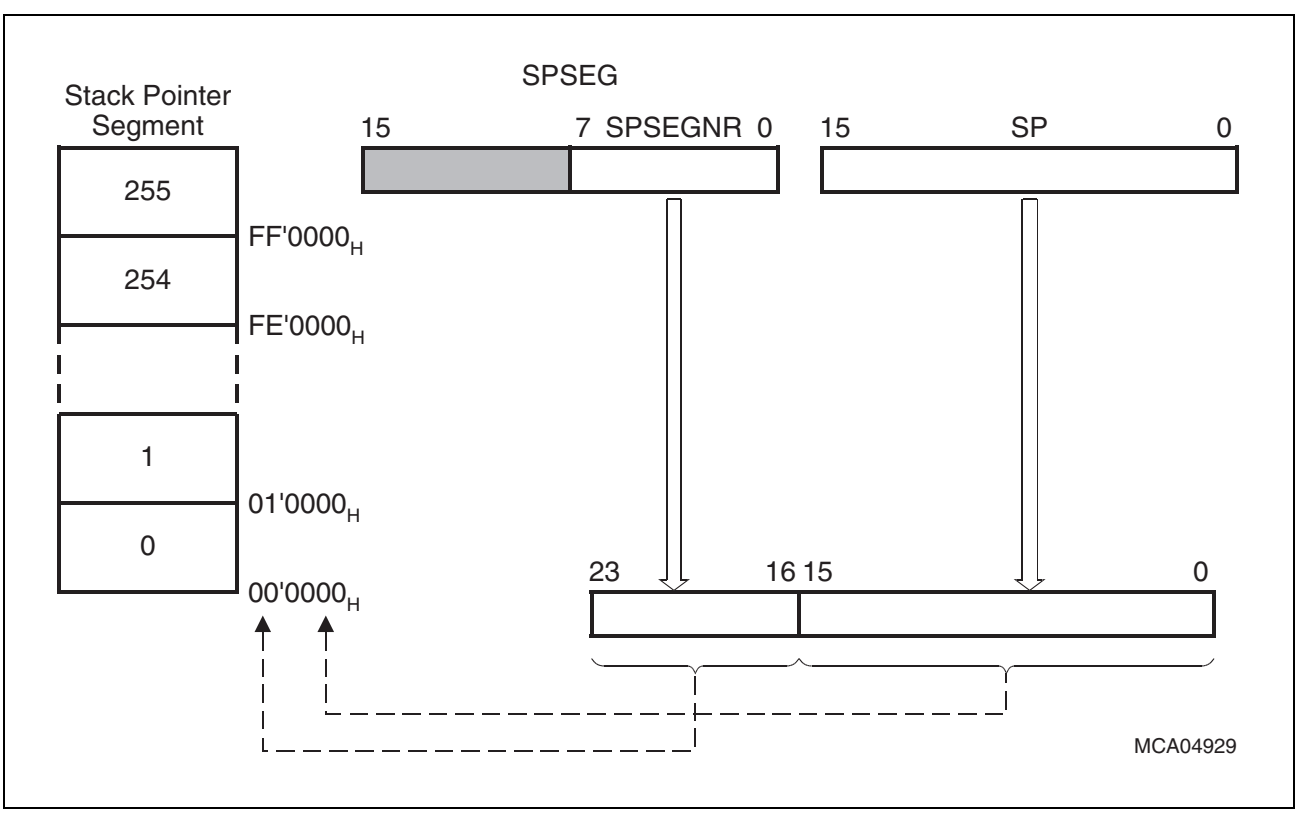
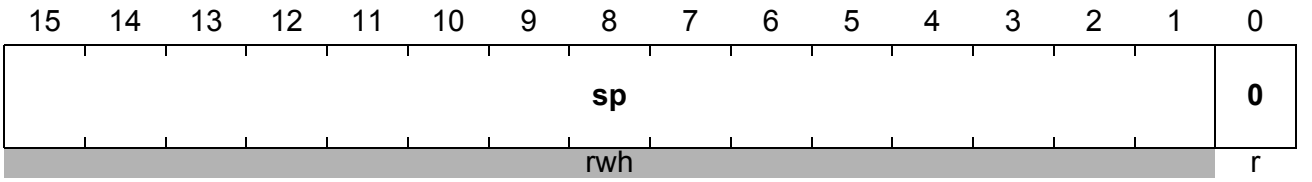


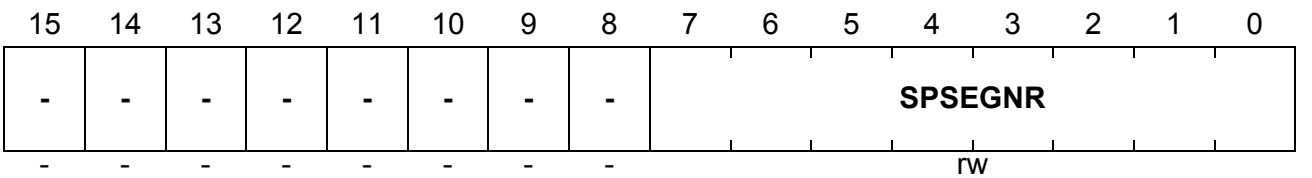
Figure 4-17 Addressing via the Stack Pointer

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 SPSEG or SP stalls the instruction flow until the register is really updated. The instruction immediately following the instruction updating SPSEG 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.

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 SPSEG 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 XC164CM 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(FF10$_H$/88$_H$)** **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ILVL				IEN	-	BANK		USR 1	USR 0	MUL IP	E	Z	V	C	N
rwh				rw	-	rwh		rwh	rwh	r	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
ILVL	[15:12]	rwh	**CPU Priority Level** 0_H Lowest Priority … … F_H Highest Priority
IEN	11	rw	**Global Interrupt/PEC Enable Bit** 0 Interrupt/PEC requests are disabled 1 Interrupt/PEC requests are enabled
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
USR1	7	rwh	**General Purpose Flag** May be used by application
USR0	6	rwh	**General Purpose Flag** May be used by application

Field	Bits	Type	Description
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 8000_H nor 80_H 1 Source operand is 8000_H or 80_H
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)

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.

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 XC164CM 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 XC164CM 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. A summary of the protected bits implemented in the XC164CM can be found in **Section 2.7**.

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

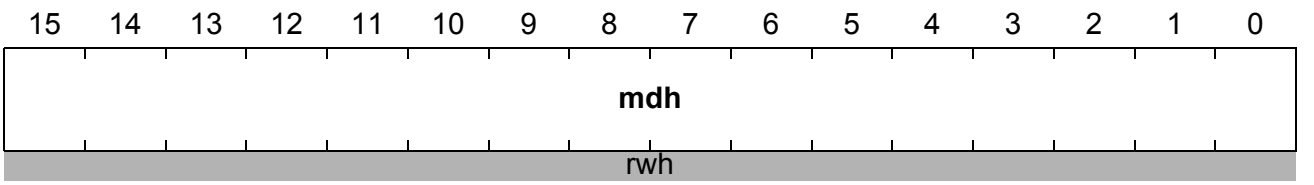
The XC164CM's multiply and divide unit has two separated parts. One is the fast 16 × 16-bit multiplier that executes a multiplication in one CPU cycle. The other one is a division sub-unit which performs the division algorithm in 18 … 21 CPU cycles (depending on the data and division types). The divide 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**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
mdl															
rwh															

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

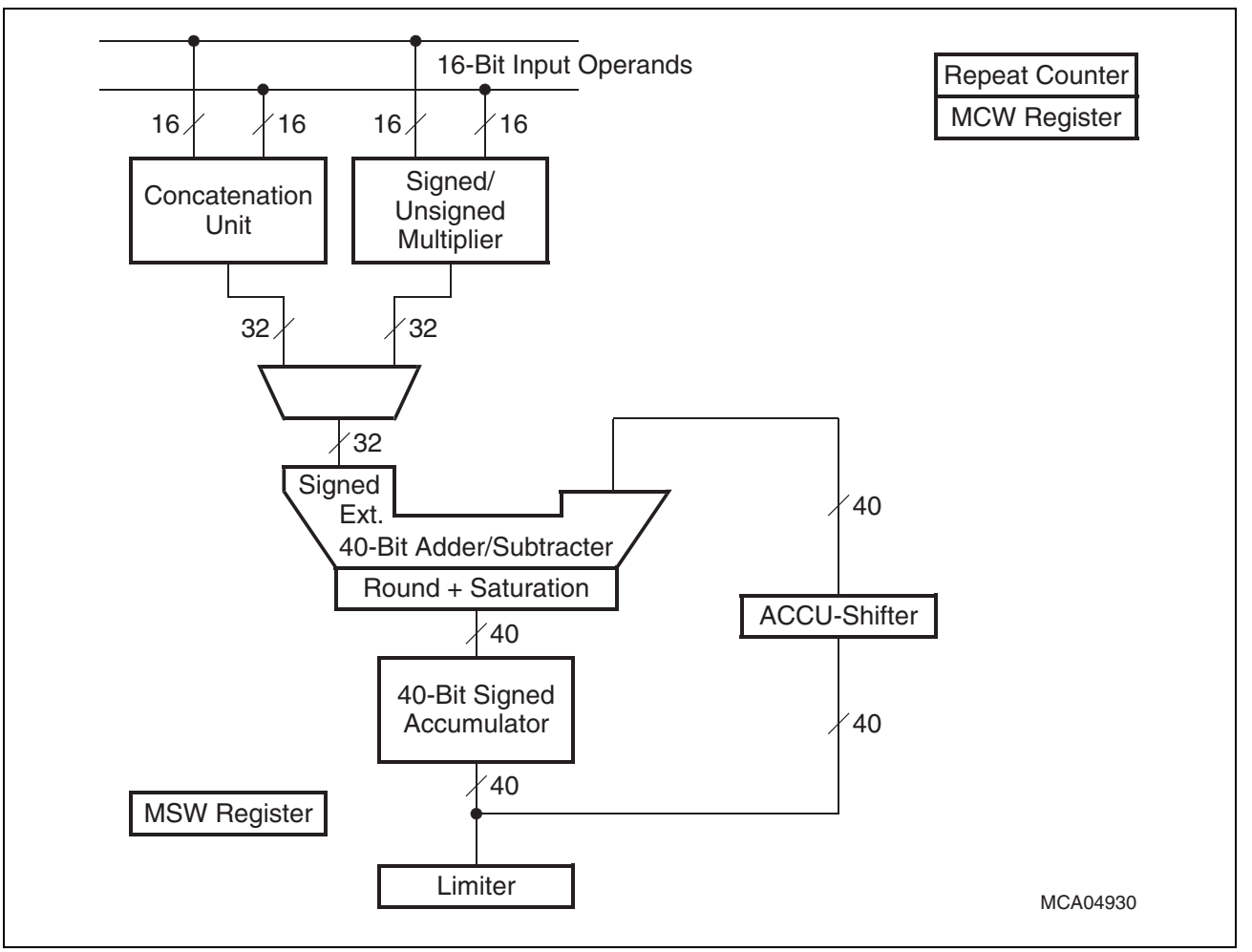


Figure 4-18 Functional MAC Unit Block Diagram

1) The same hardware-multiplier is used in the ALU.

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

4.9.1 Representation of Numbers and Rounding

The XC164CM 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 XC164CM 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).

4.9.2 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.

4.9.3 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.

4.9.4 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.

4.9.5 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).

4.9.6 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.

4.9.7 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.

4.9.8 The 40-bit Signed Accumulator Register

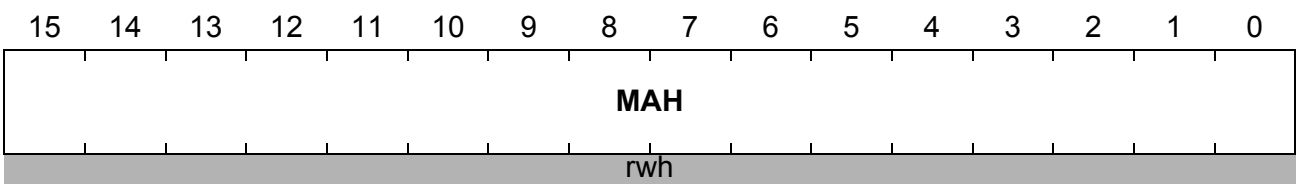
The 40-bit accumulator consists of three concatenated registers MAE, MAH, and MAL. MAE is 8 bits wide, MAH and MAL are 16 bits wide. MAE is the Most Significant Byte of the 40-bit accumulator. This byte performs a guarding function. MAE is accessed as the lower byte of register MSW.

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.

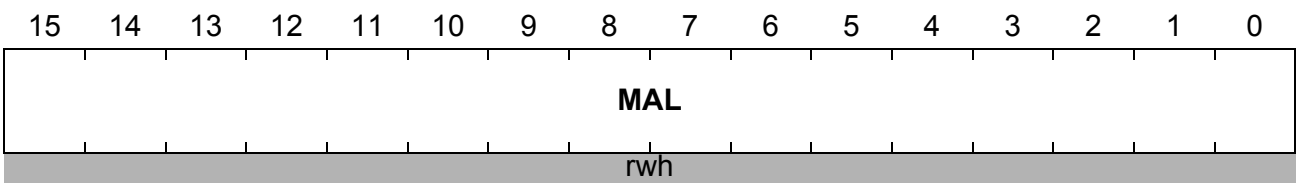
During the accumulator operations, an overflow may happen and the result may not fit into 32 bits and MAE will change. The extension flag "E" in register MSW is set when the signed result in the accumulator has exceeded the 32-bit boundary. This condition is present when the highest 9 bits of the 40-bit 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.

MAH
Accumulator High Word **SFR ($FE5E_H/2F_H$)** **Reset Value: 0000_H**



MAL
Accumulator Low Word **SFR ($FE5C_H/2E_H$)** **Reset Value: 0000_H**



Field	Bits	Type	Description
MAH, MAL	[15:0]	rwh	**High and Low Part of Accumulator** The 40-bit accumulator is completed by MAE

4.9.9 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**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	MV	MSL	ME	MSV	MC	MZ	MN	MAE							
-	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh							

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.

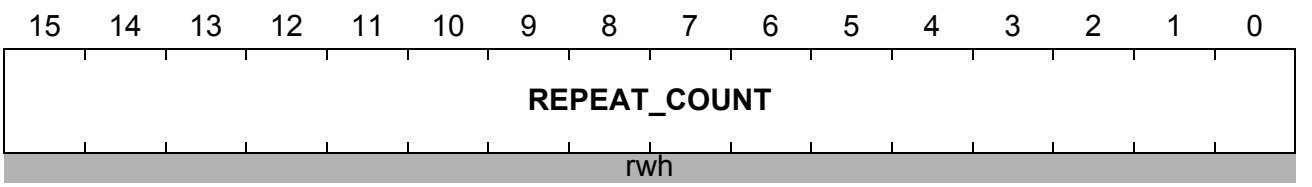
MSL-Flag: The MSL-flag is set if an automatic saturation of the accumulator has happened. The automatic saturation is enabled if bit MS in register MCW is set. The MSL-Flag can be also set by instructions which limit the contents of the accumulator. If the accumulator has been limited, the MSL-Flag is 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.

4.9.10 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

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

5 Interrupt and Trap Functions

The architecture of the XC164CM supports several mechanisms for fast and flexible response to service requests from various sources internal or external to the microcontroller. 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 XC164CM'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 by the Non-Maskable Interrupt pin, $\overline{\text{NMI}}$. 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 XC164CM 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 XC164CM provides 63 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 XC164CM 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 XC164CM'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.

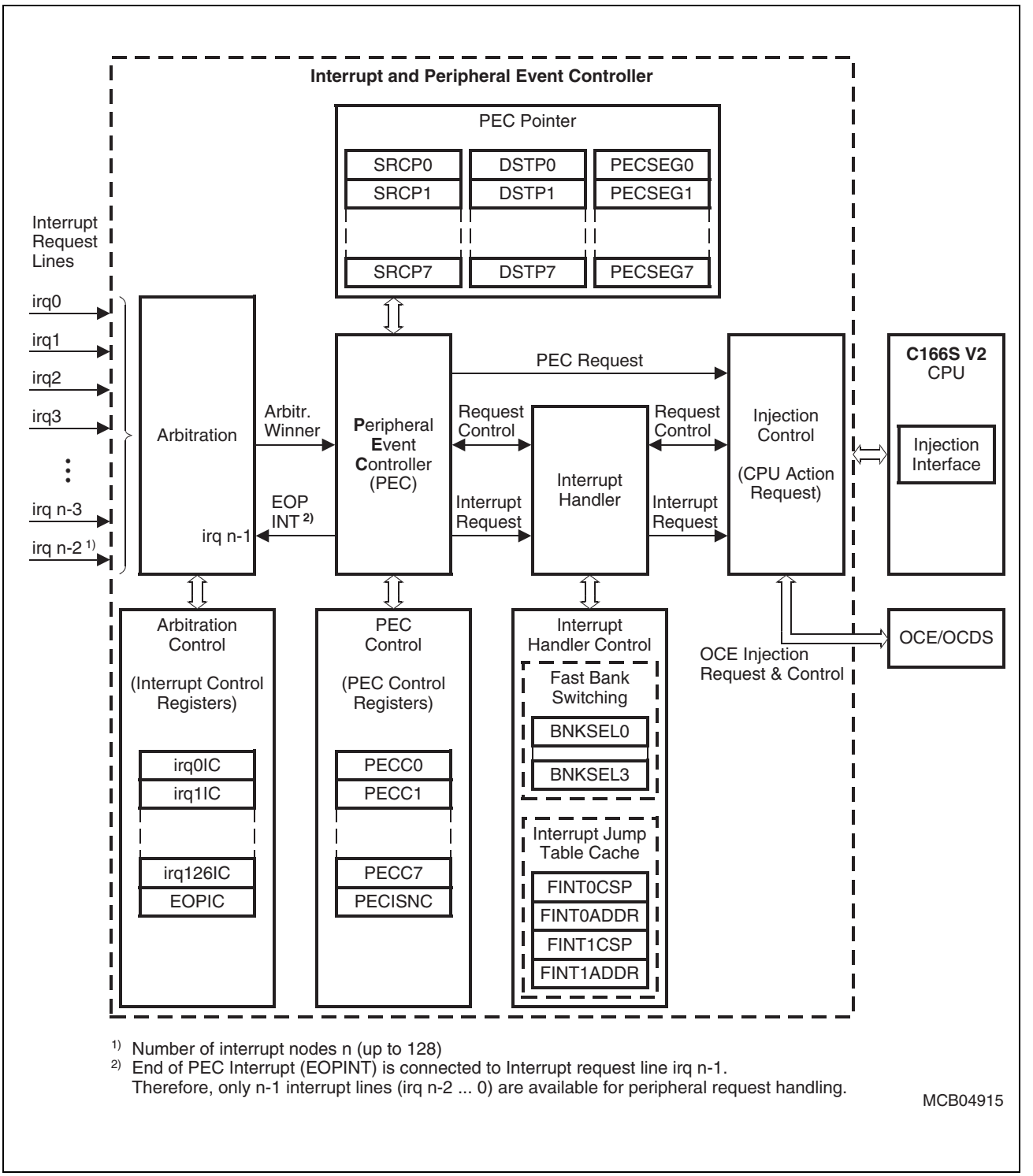


Figure 5-1 Block Diagram of the Interrupt and PEC Controller

5.2 Interrupt Arbitration and Control

The XC164CM'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:

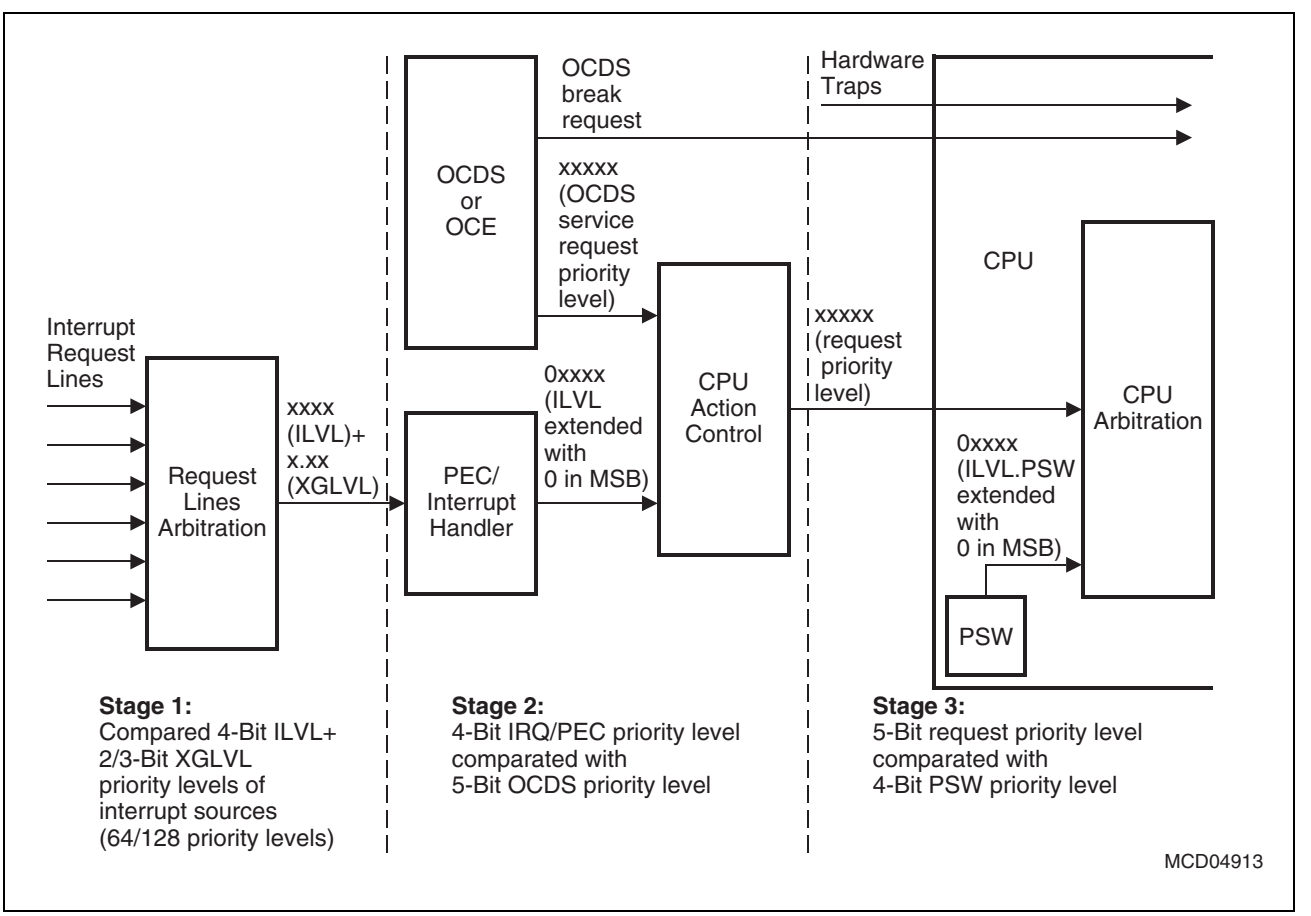


Figure 5-2 Interrupt Arbitration

The interrupt prioritization is done in three stages:

- Select one of the active interrupt requests
- Compare the priority levels of the selected request and an OCDS service request
- Compare the priority level of the final request with the 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.

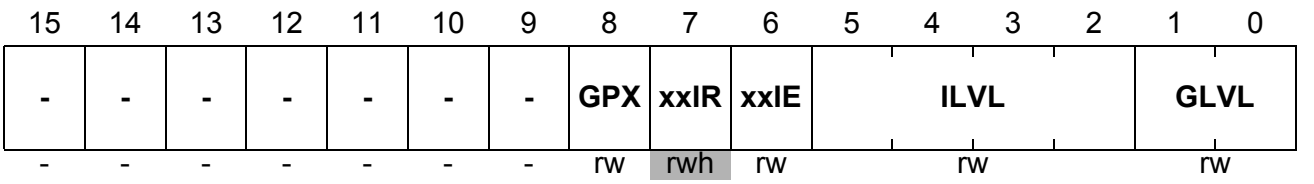
Both the OCDS break requests and the hardware traps bypass the arbitration scheme and go directly to the core (see also **Figure 5-2**).

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**



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** F_H Highest priority level … … 0_H Lowest priority level
GLVL	[1:0]	rw	**Group Priority Level** (Is completed by bit GPX to the 3-bit group level) 3_H Highest priority level … … 0_H 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 XC164CM 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 terminate the XC164CM's Idle mode and 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

5.3 Interrupt Vector Table

The XC164CM 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 XC164CM. 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).

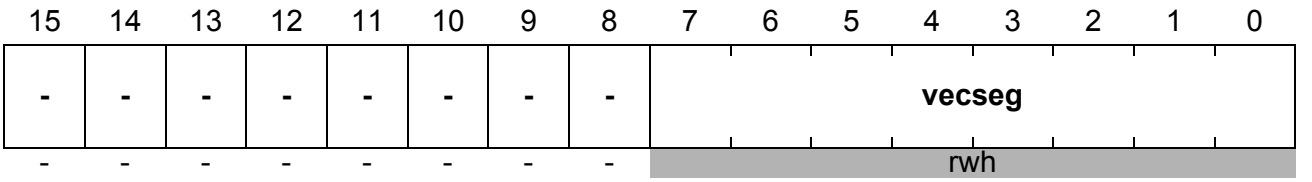
Bitfield VECSC in register CPUCON1 defines the space between two adjacent vectors (can be 2, 4, 8, or 16 words). For a summary of register CPUCON1, please refer to **Section 4.4**.

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 5-2 lists all sources capable of requesting interrupt or PEC service in the XC164CM, 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
$00C0_H$	Standard start from Internal Program Memory
$00E0_H$	Execute bootstrap loader code

Table 5-2 XC164CM Interrupt Nodes

Source of Interrupt or PEC Service Request	Control Register	Vector Location[1)]	Vector Number
EX0IN	CC1_CC8IC	xx'0060$_H$	18$_H$ / 24$_D$
EX1IN	CC1_CC9IC	xx'0064$_H$	19$_H$ / 25$_D$
EX2IN	CC1_CC10IC	xx'0068$_H$	1A$_H$ / 26$_D$
EX3IN	CC1_CC11IC	xx'006C$_H$	1B$_H$ / 27$_D$
EX4IN	CC1_CC12IC	xx'0070$_H$	1C$_H$ / 28$_D$
EX5IN	CC1_CC13IC	xx'0074$_H$	1D$_H$ / 29$_D$
CAPCOM Register 16	CC2_CC16IC	xx'00C0$_H$	30$_H$ / 48$_D$
CAPCOM Register 17	CC2_CC17IC	xx'00C4$_H$	31$_H$ / 49$_D$
CAPCOM Register 18	CC2_CC18IC	xx'00C8$_H$	32$_H$ / 50$_D$
CAPCOM Register 19	CC2_CC19IC	xx'00CC$_H$	33$_H$ / 51$_D$
CAPCOM Register 20	CC2_CC20IC	xx'00D0$_H$	34$_H$ / 52$_D$
CAPCOM Register 21	CC2_CC21IC	xx'00D4$_H$	35$_H$ / 53$_D$
CAPCOM Register 22	CC2_CC22IC	xx'00D8$_H$	36$_H$ / 54$_D$
CAPCOM Register 23	CC2_CC23IC	xx'00DC$_H$	37$_H$ / 55$_D$
CAPCOM Register 24	CC2_CC24IC	xx'00E0$_H$	38$_H$ / 56$_D$
CAPCOM Register 25	CC2_CC25IC	xx'00E4$_H$	39$_H$ / 57$_D$
CAPCOM Register 26	CC2_CC26IC	xx'00E8$_H$	3A$_H$ / 58$_D$
CAPCOM Register 27	CC2_CC27IC	xx'00EC$_H$	3B$_H$ / 59$_D$
CAPCOM Register 28	CC2_CC28IC	xx'00F0$_H$	3C$_H$ / 60$_D$
CAPCOM Register 29	CC2_CC29IC	xx'0110$_H$	44$_H$ / 68$_D$
CAPCOM Register 30	CC2_CC30IC	xx'0114$_H$	45$_H$ / 69$_D$
CAPCOM Register 31	CC2_CC31IC	xx'0118$_H$	46$_H$ / 70$_D$
CAPCOM Timer 7	CC2_T7IC	xx'00F4$_H$	3D$_H$ / 61$_D$
CAPCOM Timer 8	CC2_T8IC	xx'00F8$_H$	3E$_H$ / 62$_D$
GPT1 Timer 2	GPT12E_T2IC	xx'0088$_H$	22$_H$ / 34$_D$
GPT1 Timer 3	GPT12E_T3IC	xx'008C$_H$	23$_H$ / 35$_D$
GPT1 Timer 4	GPT12E_T4IC	xx'0090$_H$	24$_H$ / 36$_D$
GPT2 Timer 5	GPT12E_T5IC	xx'0094$_H$	25$_H$ / 37$_D$
GPT2 Timer 6	GPT12E_T6IC	xx'0098$_H$	26$_H$ / 38$_D$

Table 5-2 XC164CM Interrupt Nodes (cont'd)

Source of Interrupt or PEC Service Request	Control Register	Vector Location[1)]	Vector Number
GPT2 CAPREL Reg.	GPT12E_CRIC	xx'009C$_H$	27$_H$ / 39$_D$
A/D Conversion Compl.	ADC_CIC	xx'00A0$_H$	28$_H$ / 40$_D$
A/D Overrun Error	ADC_EIC	xx'00A4$_H$	29$_H$ / 41$_D$
ASC0 Transmit	ASC0_TIC	xx'00A8$_H$	2A$_H$ / 42$_D$
ASC0 Transmit Buffer	ASC0_TBIC	xx'011C$_H$	47$_H$ / 71$_D$
ASC0 Receive	ASC0_RIC	xx'00AC$_H$	2B$_H$ / 43$_D$
ASC0 Error	ASC0_EIC	xx'00B0$_H$	2C$_H$ / 44$_D$
ASC0 Autobaud	ASC0_ABIC	xx'017C$_H$	5F$_H$ / 95$_D$
SSC0 Transmit	SSC0_TIC	xx'00B4$_H$	2D$_H$ / 45$_D$
SSC0 Receive	SSC0_RIC	xx'00B8$_H$	2E$_H$ / 46$_D$
SSC0 Error	SSC0_EIC	xx'00BC$_H$	2F$_H$ / 47$_D$
PLL/OWD	PLL_IC	xx'010C$_H$	43$_H$ / 67$_D$
ASC1 Transmit	ASC1_TIC	xx'0120$_H$	48$_H$ / 72$_D$
ASC1 Transmit Buffer	ASC1_TBIC	xx'0178$_H$	5E$_H$ / 94$_D$
ASC1 Receive	ASC1_RIC	xx'0124$_H$	49$_H$ / 73$_D$
ASC1 Error	ASC1_EIC	xx'0128$_H$	4A$_H$ / 74$_D$
ASC1 Autobaud	ASC1_ABIC	xx'0108$_H$	42$_H$ / 66$_D$
End of PEC Subchannel	EOPIC	xx'0130$_H$	4C$_H$ / 76$_D$
CAPCOM6 Timer T12	CCU6_T12IC	xx'0134$_H$	4D$_H$ / 77$_D$
CAPCOM6 Timer T13	CCU6_T13IC	xx'0138$_H$	4E$_H$ / 78$_D$
CAPCOM6 Emergency	CCU6_EIC	xx'013C$_H$	4F$_H$ / 79$_D$
CAPCOM6	CCU6_IC	xx'0140$_H$	50$_H$ / 80$_D$
SSC1 Transmit	SSC1_TIC	xx'0144$_H$	51$_H$ / 81$_D$
SSC1 Receive	SSC1_RIC	xx'0148$_H$	52$_H$ / 82$_D$
SSC1 Error	SSC1_EIC	xx'014C$_H$	53$_H$ / 83$_D$
CAN0	CAN_0IC	xx'0150$_H$	54$_H$ / 84$_D$
CAN1	CAN_1IC	xx'0154$_H$	55$_H$ / 85$_D$
CAN2	CAN_2IC	xx'0158$_H$	56$_H$ / 86$_D$
CAN3	CAN_3IC	xx'015C$_H$	57$_H$ / 87$_D$
CAN4	CAN_4IC	xx'0164$_H$	59$_H$ / 89$_D$

Table 5-2 XC164CM Interrupt Nodes (cont'd)

Source of Interrupt or PEC Service Request	Control Register	Vector Location[1)]	Vector Number
CAN5	CAN_5IC	xx'0168$_H$	5A$_H$ / 90$_D$
CAN6	CAN_6IC	xx'016C$_H$	5B$_H$ / 91$_D$
CAN7	CAN_7IC	xx'0170$_H$	5C$_H$ / 92$_D$
RTC	RTC_IC	xx'0174$_H$	5D$_H$ / 93$_D$
Unassigned node	---	xx'0040$_H$	10$_H$ / 16$_D$
Unassigned node	---	xx'0044$_H$	11$_H$ / 17$_D$
Unassigned node	---	xx'0048$_H$	12$_H$ / 18$_D$
Unassigned node	---	xx'004C$_H$	13$_H$ / 19$_D$
Unassigned node	---	xx'0050$_H$	14$_H$ / 20$_D$
Unassigned node	---	xx'0054$_H$	15$_H$ / 21$_D$
Unassigned node	---	xx'0058$_H$	16$_H$ / 22$_D$
Unassigned node	---	xx'005C$_H$	17$_H$ / 23$_D$
Unassigned node	---	xx'0078$_H$	1E$_H$ / 30$_D$
Unassigned node	---	xx'007C$_H$	1F$_H$ / 31$_D$
Unassigned node	---	xx'0080$_H$	20$_H$ / 32$_D$
Unassigned node	---	xx'0084$_H$	21$_H$ / 33$_D$
Unassigned node	---	xx'00FC$_H$	3F$_H$ / 63$_D$
Unassigned node	---	xx'0100$_H$	40$_H$ / 64$_D$
Unassigned node	---	xx'0104$_H$	41$_H$ / 65$_D$
Unassigned node	---	xx'012C$_H$	4B$_H$ / 75$_D$
Unassigned node	---	xx'0160$_H$	58$_H$ / 88$_D$

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)]	Vector Number	Trap Priority
Reset Functions	–				
• Hardware Reset		RESET	xx'0000_H	00_H	III
• Software Reset		RESET	xx'0000_H	00_H	III
• W-dog Timer Overflow		RESET	xx'0000_H	00_H	III
Class A Hardware Traps:					
• Non-Maskable Interrupt	NMI	NMITRAP	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:					
• Undefined Opcode	UNDOPC	BTRAP	xx'0028_H	$0A_H$	I
• PMI Access Error	PACER	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[1)]	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.

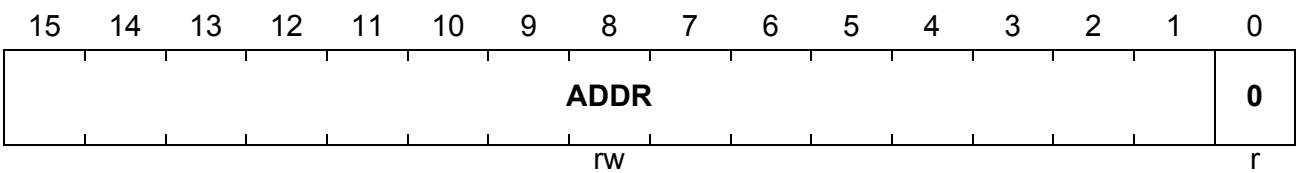
Interrupt Jump Table Cache

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.

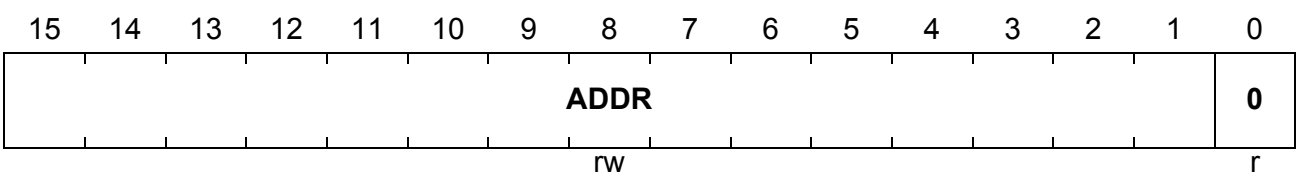
The ITC provides two 24-bit pointers, so the CPU can directly branch to the respective service routines. These fast interrupts can be selected for two interrupt sources on priority levels 15 … 12.

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**



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**

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							

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 (1100_B) 01 Interrupt priority level 13 (1101_B) 10 Interrupt priority level 14 (1110_B) 11 Interrupt priority level 15 (1111_B)
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 XC164CM'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

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
PEC Control Reg. **SFR (FECy$_H$/6z$_H$, Table 5-4)** **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	EOP INT	PLEV		CL	INC		BWT	COUNT							
-	rw	rw		rw	rw		rw	rwh							

Field	Bits	Type	Description
EOPINT	14	rw	**End of PEC Interrupt Selection** 0 End of PEC interrupt on the same (PEC) level 1 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 PEC channels work independently 1 Pairs of PEC channels are linked together[1)]
INC	[10:9]	rw	**Increment Control** (Pointer Modification)[2)] 00 Pointers are not modified 01 Increment DSTPx by 1 or 2 (BWT = 1 or 0) 10 Increment SRCPx by 1 or 2 (BWT = 1 or 0) 11 Increment both DSTPx and SRCPx by 1 or 2
BWT	8	rw	**Byte/Word Transfer Selection** 0 Transfer a word 1 Transfer a byte
COUNT	[7:0]	rwh	**PEC Transfer Count** Counts PEC transfers and influences the channel's action (see **Section 5.4.2**)

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.

5.4.1 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).

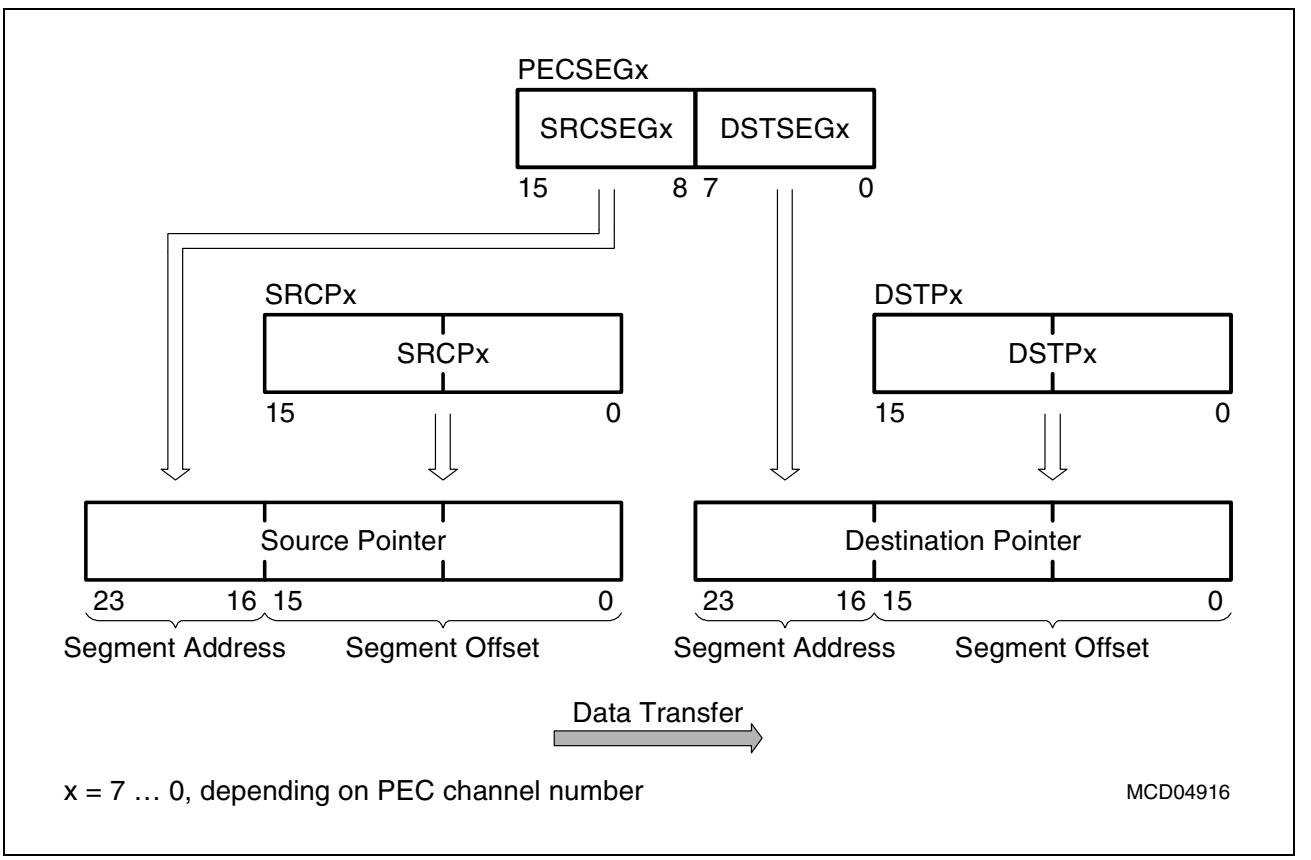


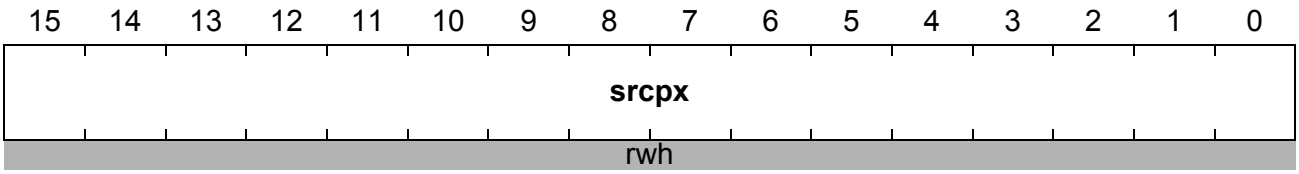
Figure 5-3 PEC Data Pointers

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.

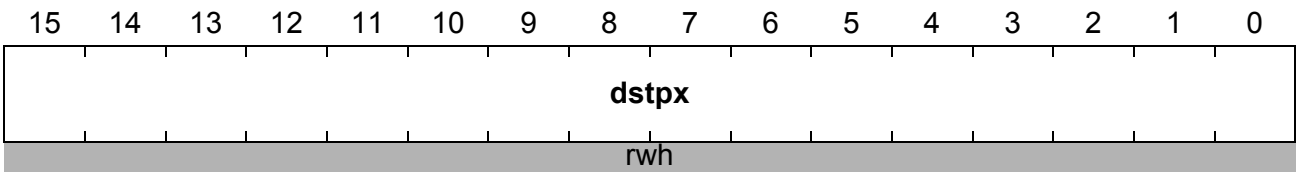
Note: PEC data transfers do not use the data page pointers DPP3 … DPP0. Unused PEC pointers may be used for general data storage.

SRCPx
PEC Source Pointer **XSFR (ECyy$_H$/--, Table 5-7)** **Reset Value: 0000$_H$**



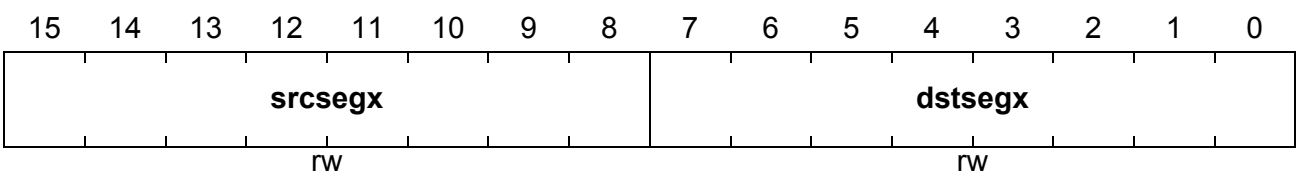
Field	Bits	Type	Description
srcpx	[15:0]	rwh	**Source Pointer Offset of Channel x** Source address bits 15 … 0

DSTPx
PEC Destination Pointer **XSFR (ECyy$_H$/--, Table 5-7)** **Reset Value: 0000$_H$**



Field	Bits	Type	Description
dstpx	[15:0]	rwh	**Destination Pointer Offset of Channel x** Destination address bits 15 … 0

PECSEGx
PEC Segment Pointer **XSFR (ECyy$_H$/--, Table 5-7)** **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.

5.4.2 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 Leave request flag set, which triggers another request
		0	**EOPINT = 1** (separate end-of-PEC interrupt) Move a Byte/Word 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**).

5.4.3 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.

5.4.4 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 ($FFA8_H/D4_H$)** **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 = 7 … 0**	[2x+1]	rwh	**Interrupt Request Flag of PEC Channel x** 0 No request from PEC channel x pending 1 PEC channel x has raised an end-of-PEC interrupt request *Note: These request flags must be cleared by SW.*
CxIE **x = 7 … 0**	[2x]	rw	**Interrupt Enable Control Bit of PEC Channel x** (individually enables/disables a specific source) 0 End-of-PEC request of channel x disabled 1 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 ($F180_H/C0_H$)** **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.

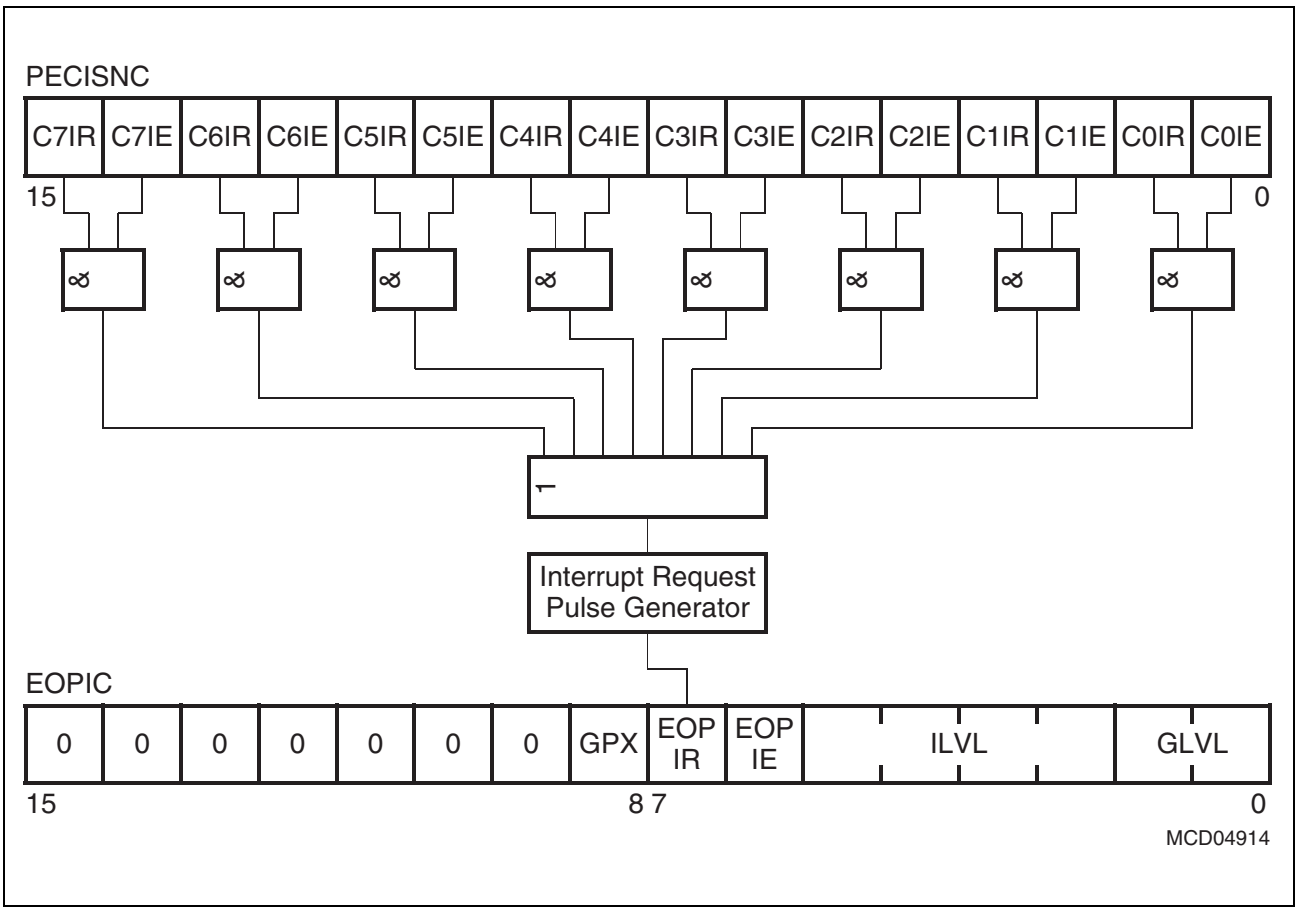


Figure 5-4 End of PEC Interrupt Sub Node

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 XC164CM 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.

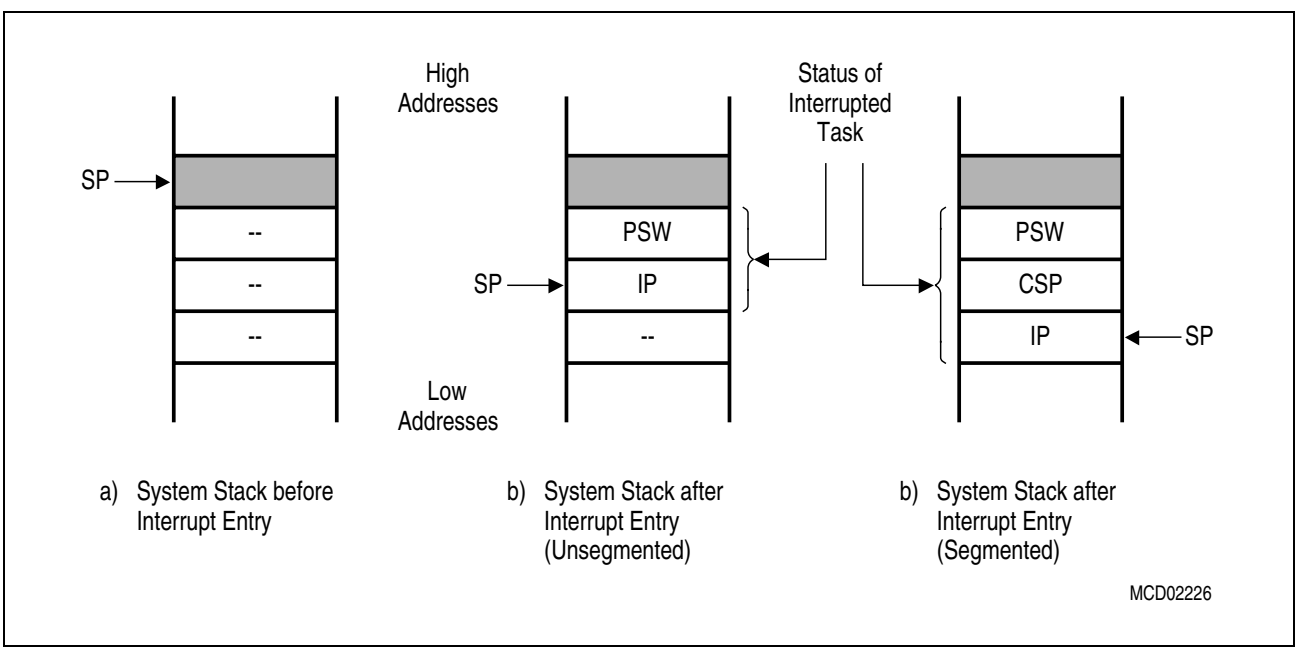


Figure 5-5 Task Status Saved on the System Stack

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 XC164CM 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 XC164CM 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.

BNKSELx
Register Bank Select Reg. x **XSFR (Table 5-10)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPRSEL7		GPRSEL6		GPRSEL5		GPRSEL4		GPRSEL3		GPRSEL2		GPRSEL1		GPRSEL0	
rw		rw		rw		rw		rw		rw		rw		rw	

Field	Bits	Type	Description
GPRSELy (y = 7 … 0)	[2y+1 :2y]	rw	**Register Bank Selection** 00 Global register bank 01 Reserved 10 Local register bank 1 11 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 the involved request sources are simply ORed to trigger the common node. 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
ASC0_ABIC	ASC0: autobaud detect start, error request	ORed
ASC1_ABIC	ASC0: autobaud detect start, error request	ORed

5.8 External Interrupts

Although the XC164CM 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 **Fast External Interrupt** detection provides flexible wake-up signals even in sleep mode. This function can also generate additional interrupt requests from external input signals.

Table 5-12 Pins Usable as External Interrupt Inputs

Port Pin	Original Function	Control Register
P1H.5-4/CC25-24IO	CAPCOM Register 25-24 Capture Input	CC25-CC24
P1H.0/CC23IO	CAPCOM Register 23 Capture Input	CC23
P1L.7/CC22IO	CAPCOM Register 22 Capture Input	CC22
P9.5-0/CC21-16IO	CAPCOM Register 21-16 Capture Input	CC21-CC16
P3.2/CAPIN	GPT2 capture input pin	T5CON
P3.7/T2IN	Auxiliary timer T2 input pin	T2CON
P3.5/T4IN	Auxiliary timer T4 input pin	T4CON

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 fast external interrupts). 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 direction control bit DPx.y in the respective port direction control register DPx.

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.

Note: The non-maskable interrupt input pin $\overline{NMI}$ and the reset input $\overline{RSTIN}$ provide another possibility for the CPU to react to an external input signal. $\overline{NMI}$ and $\overline{RSTIN}$ are dedicated input pins which cause hardware traps.

Fast External Interrupts

The fast external interrupt pins are sampled every system clock cycle; that is, external events are scanned and detected in time frames of $1/f_{SYS}$. The arbitration and processing of these interrupt requests, however, is done with the normal timing.

The External Interrupt Control register EXICON selects the trigger transition (rising, falling or both) individually for each of 8 fast external interrupts.

These fast external interrupts use the interrupt nodes and vectors of the CAPCOM channels CC13 … CC8, so the capture/compare function cannot be used.

EXICON
External Intr. Control Reg. **ESFR ($F1C0_H/E0_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-		-		EXI5ES		EXI4ES		EXI3ES		EXI2ES		EXI1ES		EXI0ES	
-		-		rw		rw		rw		rw		rw		rw	

Field	Bits	Type	Description
EXIxES (x = 5 … 0)	[11:10] … [1:0]	rw	**External Interrupt x Edge Selection Field** 00 Fast external interrupts disabled: std. mode 01 Interrupt on positive edge (rising) 10 Interrupt on negative edge (falling) 11 Interrupt on any edge (rising or falling)

External Interrupt Source Control

The input source for each of the fast external interrupts (controlled via register EXICON) can be derived from up to three associated port pins (standard pin EXnIN or two alternate sources). Activating an alternate input source, for example, allows the detection of transitions on the interface lines of disabled interfaces. Upon this trigger, the respective interface can be reactivated and respond to the detected activity.

Source selection is controlled via registers EXISEL0 and EXISEL1. Besides selecting one of the three possible input pins, two or all of them can also be logically combined. This can be used to increase the number of wake-up lines or to define specific signal combinations to trigger a wake-up interrupt.

EXISEL0
Ext. Interrupt Source Reg. 0 ESFR ($F1DA_H/ED_H$) Reset Value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXI3SS				EXI2SS				EXI1SS				EXI0SS			
rw				rw				rw				rw			

EXISEL1
Ext. Interrupt Source Reg. 1 ESFR ($F1D8_H/EC_H$) Reset Value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-				-				EXI5SS				EXI4SS			
-				-				rw				rw			

Field	Bits	Type	Description
EXIxSS (x = 5 … 0)	[15:12] … [3:0]	rw	**External Interrupt x Source Selection Field** 0000 Input from associated EXxIN pin 0001 Input from alternate pin AltA 0010 Input from alternate pin AltB 0011 Input from pin EXxIN ORed with alternate pin AltA 0100 Input from pin EXxIN ANDed with alternate pin AltA 0101 Input from alternate pin AltA ORed with alternate pin AltB 0110 Input from alternate pin AltA ANDed with alternate pin AltB 0111 Input from pin EXxIN ORed with pin AltA ORed with pin AltB 1XXX Reserved, do not use

The **Table 5-13** summarizes the association of the bitfields of register EXISEL (i.e. the interrupt lines) with the respective input pins.

Table 5-13 Connection of Interrupt Inputs to External Interrupt Nodes

Control Bitfield	Std. Pin EXnIN	Alternate Pin AltA	Alternate Pin AltB	Interrupt Ctrl. Reg.	Associated Interface	Notes
EXI0SS	P1H.0	P1H.3	P1H.0	CC8IC	SSC1	–
EXI1SS	P1H.1	P3.1	–	CC9IC	ASC1	–
EXI2SS	P1H.2	P3.11	P3.10	CC10IC	ASC0	–
EXI3SS	P1H.3	P3.13	–	CC11IC	SSC0	–
EXI4SS	P1H.4	P9.2	–	CC12IC	CAN_A	The actual interface pin is programmable
EXI5SS	P1H.5	P9.0	–	CC13IC	CAN_B	

External Interrupts During Sleep Mode

During Sleep Mode, all peripheral clock signals are deactivated. This also disables the standard edge detection logic for the fast external interrupts. However, transitions on these interrupt inputs must be recognized in order to initiate the wake-up. This is accomplished by a special edge detection logic for the fast external interrupts which requires no clock signal (therefore also works in Sleep mode) and is equipped with an analog noise filter. This filter suppresses spikes (generated by noise) up to 10 ns. Input pulses with a duration of 100 ns minimum are recognized and generate an interrupt request.

This filter delays the recognition of an external wake-up signal by approximately 100 ns, but the spike suppression ensures safe and robust operation of the sleep/wake-up mechanism in an active environment.

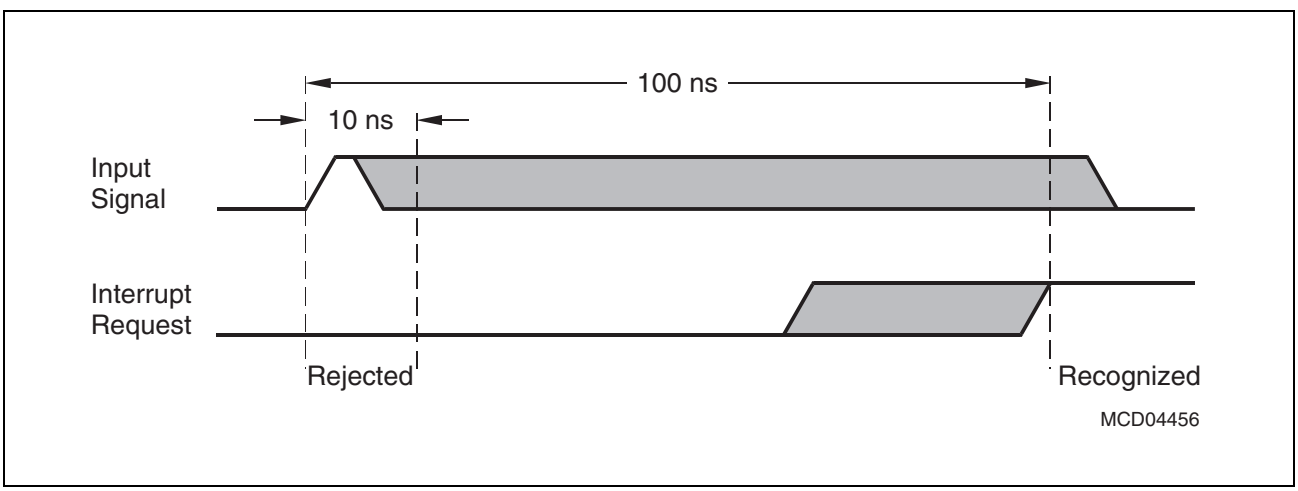


Figure 5-6 Input Noise Filter Operation

External Interrupt Pulse Timing

External interrupt inputs are evaluated by a synchronous logic and by an asynchronous logic. The synchronous logic supports the recognition of short interrupt pulses at higher system frequencies, the asynchronous logic ensures recognition of interrupt pulses during sleep mode, when no system clock is available.

An external interrupt signal is safely recognized in two cases:

- If it is active for more than 100 ns (async. logic with spike filter), or
- If it is active for more than 2 cycles of f_{SYS} (sync. logic).

The interrupt signal is recognized after whatever condition becomes true first.

*Note: After wake-up from Sleep mode, the time span until the PLL becomes locked is **not** critical for new external interrupt pulses to be correctly synchronized, because in this case the asynchronous logic will detect the external interrupt correctly, if it is active for at least 100 ns.*

Note: The $\overline{NMI}$ input features the same spike filter and the same timing requirements.

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 XC164CM (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 5-14** lists the consecutive steps required for this process.

Table 5-14 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**)	9 cycles	9 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	17/13 cycles	13 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 5-15 Additional Delays Caused by System Logic

Reason for Delay	Interrupt Response	PEC Response
Interrupt controller busy, because it is just executing an arbitration cycle	max. 9 cycles	max. 9 cycles
Pipeline is stalled, because the 2 instructions already in the pipeline preceding the injected instruction (PEC or ITRAP) need to complete before the injected instruction can be executed. For example, the instructions may need to write/read data to/from a peripheral or memory, or may need extra cycles to complete.	$2 \times T_{ACCmax}$	$2 \times T_{ACCmax}$
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)	$2/3 \times T_{ACC}$ [1)]	- - -
Memory access for vector table read (except for intr. jump table cache)	$2 \times T_{ACC}$	- - -

1) 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 XC164CM 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 XC164CM distinguishes eight 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 eight hardware trap functions of the XC164CM are divided into two classes:

Class A traps are:

- External Non-Maskable Interrupt (NMI)
- Stack Overflow
- Stack Underflow trap
- Software Break

These traps share the same trap priority, but have individual vector addresses.

Class B traps are:

- Undefined Opcode
- Program 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 (hardware, software, watchdog) 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 NMI 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 NMI trap (Class A), both the NMI and the UNDOPC flag is set, the IP of the instruction with the undefined opcode is pushed onto the system stack, but the NMI trap is executed. After return from the NMI 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
NMI	STKOF	STKUF	SOFTBRK	-	-	-	-	UNDOPC	-	-	PACER	PRTFLT	ILLOPA	-	-
rwh	rwh	rwh	rwh	-	-	-	-	rwh	-	-	rwh	rwh	rwh	-	-

Field	Bits	Type	Description
NMI	15	rwh	**Non Maskable Interrupt Flag** 0 No non-maskable interrupt detected 1 A negative transition (falling edge) has been detected at pin $\overline{NMI}$
STKOF	14	rwh	**Stack Overflow Flag** 0 No stack overflow event detected 1 The current stack pointer value falls below the contents of register STKOV
STKUF	13	rwh	**Stack Underflow Flag** 0 No stack underflow event detected 1 The current stack pointer value exceeds the contents of register STKUN
SOFTBRK	12	rwh	**Software Break** 0 No software break event detected 1 Software break event detected
UNDOPC	7	rwh	**Undefined Opcode** 0 No undefined opcode event detected 1 The currently decoded instruction has no valid XC164CM opcode
PACER	4	rwh	**Program Memory Access Error** 0 No access error event detected 1 Illegal or erroneous access detected
PRTFLT	3	rwh	**Protection Fault** 0 No protection fault event detected 1 A protected instruction with an illegal format has been detected
ILLOPA	2	rwh	**Illegal Word Operand Access** 0 No illegal word operand access event detected 1 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 $\overline{\text{NMI}}$ 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 $\overline{\text{NMI}}$, 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 NMI 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 is an asynchronous external (to the CPU) event 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.

External NMI Trap

Whenever a high to low transition on the dedicated external $\overline{\text{NMI}}$ pin (Non-Maskable Interrupt) is detected, the NMI flag in register TFR is set and the CPU will enter the NMI trap routine.

Stack Overflow Trap

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

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

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.

Undefined Opcode Trap

When the instruction currently decoded by the CPU does not contain a valid XC164CM 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.

Program Memory Access Error

When a program memory access error is detected, the PACER flag is set in register TFR and the CPU enters the PMI 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**)
- access to Monitor RAM, if not in emulation mode

In case of an access error, additionally the soft-trap code $1E9B_H$ is issued.

Protection Fault Trap

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

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 General System Control Functions

The XC164CM System Control Unit (SCU) summarizes a number of central control tasks and product specific features. These features include functional modules such as the Watchdog Timer (WDT) or the Clock Generation Unit (CGU), as well as basic functions such as the register protection mechanism or the reset generation.

The following general functions are provided:

- The **System Reset** is generated by the Reset Control Block and handles the reset and startup behavior (internal initialization) of the chip. It controls the reset triggers as well as the reset timing. This block controls also the basic configuration of the XC164CM via external hardware.
- The **Clock Generation Unit (CGU)** provides the on-chip oscillator and the Phase Locked Loop (PLL). This block generates all clock signals for the XC164CM and distributes them to the respective modules. Also the status of the clock generation system is indicated.
- The **Central System Control Functions** comprise all central control tasks like security level selection and system behavior in Sleep mode and Powerdown mode. Depending on the application state, different security levels (like protected and unprotected mode) are supported by the security level control state machine.
- The **Watchdog Timer (WDT)** represents one of the fail-safe mechanisms which have been implemented to prevent the controller from malfunctioning. It can detect long term malfunctions and is always enabled after chip initialization. The WDT can operate in Compatible mode or in Enhanced WDT mode.
- The **Identification Control Block** supports a set of six identification registers for identification of the most important silicon parameters (chip manufacturer, chip type and its properties). This information can be used for automatic test selection.

6.1 System Reset

The internal system reset function provides initialization of the XC164CM into a defined default state. The default state is invoked either by asserting a hardware reset signal on pin RSTIN (Hardware Reset Input), by executing the SRST instruction (Software Reset), or by an overflow of the watchdog timer.

Whenever one of these conditions occurs, the microcontroller is reset into a predefined default state through an internal reset procedure. When a software reset is initiated, pending internal hold states are cancelled and the current internal access cycle (if any) is completed. An LX-bus access cycle is completed. Afterwards, the IO pin drivers are switched off (tristate). Hardware reset and watchdog reset immediately abort all actions.

The internal reset procedure is executed in several consecutive phases. The order of these phases depends on the reset source. In general, reset is triggered asynchronously (external) or synchronously (internal), it is always terminated synchronously.

Table 6-1 Sequence of Reset Phases

Phase	Hardware Reset[1)]	Watchdog Reset	Software Reset
1	**External Reset Phase** Covers the time until the external trigger is removed (RSTIN = 1), the device is reset asynchronously	-------skipped-------	**Prereset Phase (Shut down)** Covers the time until the running and pending actions of on-chip modules are completed
2	**Internal Reset Phase** The appropriate parts of the chip (peripheral system and/or CPU) are in reset state (except for the reset control block, of course). The internal reset phase covers the time specified by the reset event timer.		
3	**Initialization Phase** The appropriate parts of the chip (peripheral system and/or CPU) are set up according to the default configuration: • Internal startup: A fixed default configuration is used. • Bootstrap loader: Program code is loaded from the external system.		
4	**Operation** (Reset phases are terminated) The user software is executed from now on.		

1) A hardware reset must always be asserted while the supply voltages are outside their defined operating ranges, for example, during Power-On.

6.1.1 Reset Sources and Phases

The XC164CM executes a reset in several phases whose sequence depends on the reset trigger (see **Table 6-1**).

External Reset Phase

A hardware reset is asynchronously triggered when the reset input signal $\overline{\text{RSTIN}}$ is recognized low. A spike suppression input filter in the $\overline{\text{RSTIN}}$ line suppresses all signals shorter than 10 ns. To ensure the recognition of the $\overline{\text{RSTIN}}$ signal, it must be held low for at least 100 ns so it will safely pass the reset input filter. This is also required after the supply voltages have become stable.

Note: The minimum duration of the external reset must ensure that the hardware configuration signals have reached their intended logic levels.

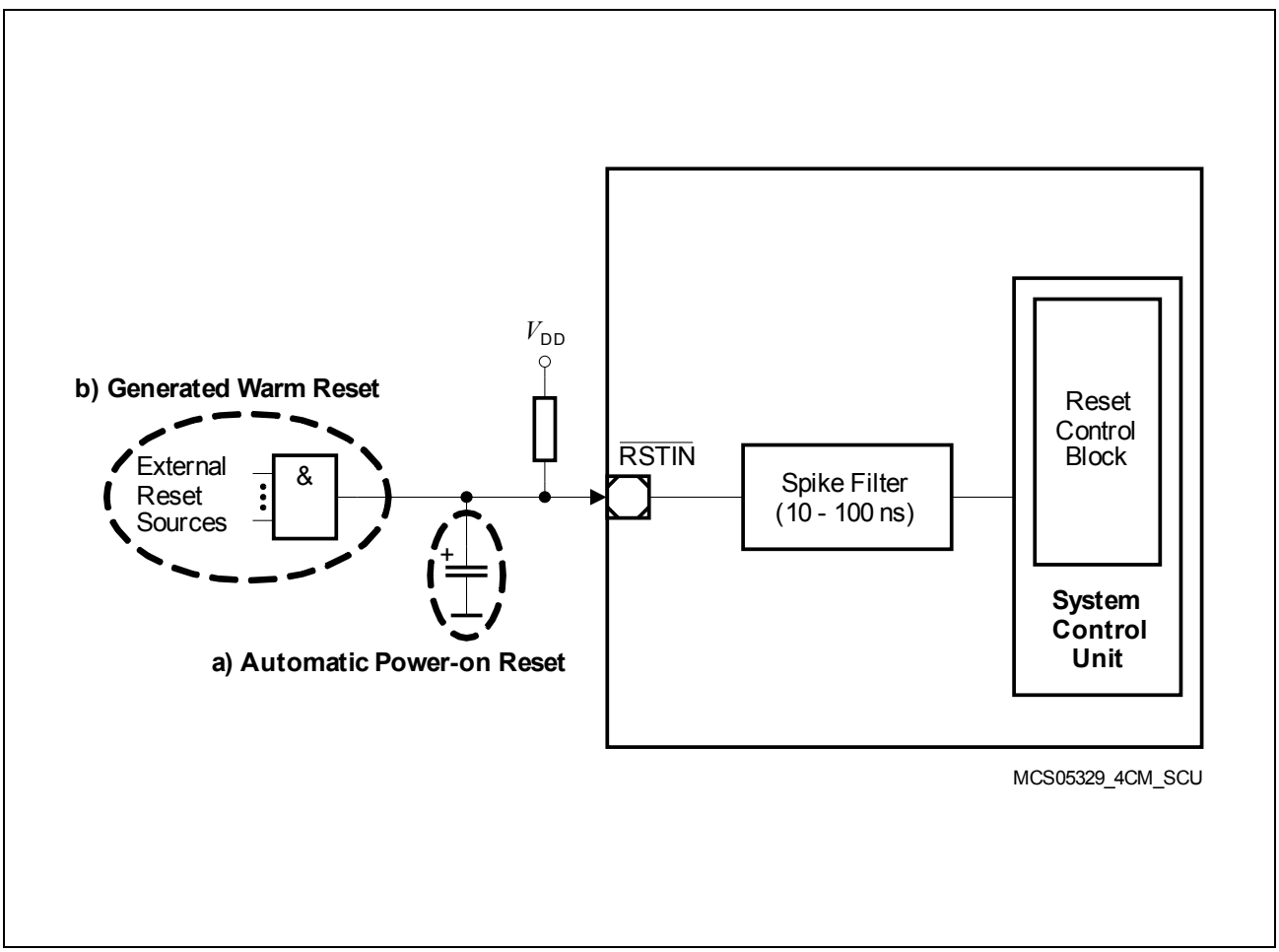


Figure 6-1 External Reset Circuitry

A hardware reset on input $\overline{\text{RSTIN}}$ may be triggered in several ways (see **Figure 6-1**).

- An external pull-up device connected to an external capacitor is sufficient for an automatic power-on reset.
- An external pull-up device connected to an external switch provides a manual reset.
- $\overline{\text{RSTIN}}$ may also be connected to the output of other logic for generating a warm reset.

Note: During the external reset phase the complete chip is in reset state. The external reset phase is left synchronously, when the $\overline{\text{RSTIN}}$ level goes inactive (high).

Pre-Reset Phase

The pre-reset phase is triggered by a software reset. During the pre-reset phase, the CPU first runs its pipeline (including all write back buffers) empty, and then indicates the software reset request to the system control unit. The pipeline stays empty after this request trigger is activated.

As soon as the software reset request occurs, the SCU requests a shutdown from the active modules equipped with shutdown handshake (see **Section 6.3.3**). The pre-reset phase is complete as soon as all modules acknowledge the shutdown state.

Upon a shutdown request the EBC will finish the currently running bus cycle.

Internal Reset Phase

At the beginning of the internal reset phase the internal reset condition becomes active, that means, the internal reset signal is actually applied to the modules. If the reset was triggered by hardware, it may be active already.

Note: The reset control block (including the watchdog timer) is not reset, of course.

The duration of the internal reset phase is determined by the reset-length-control bitfield RSTLEN in register RSTCON. The WDT low byte is used for counting the reset duration. When entering the internal reset phase, the timer is cleared and then counts up with frequency f_{WDT}.
The default count frequency after a hardware reset is $f_{WDT} = f_{SYS}/2 = f_{MC}/2$. Internal reset triggers do not change the current clock setting and the value of bitfield WDTIN, so the previously selected f_{WDT} is used.

The actual duration of the internal reset sequence can therefore be calculated using the following formula:

$$t_{RST} = \frac{2^{(RSTLEN)}}{f_{WDT}} \qquad (6.1)$$

Reset Termination (Initialization Phase)

When the end of the internal reset phase has been reached, the following actions take place, before control is passed to the software:

- Set the reset indication flags in register SYSSTAT accordingly
- Select initial configuration and reset start address
- Deactivate the internal reset signals
- Execute bootstrap loader if selected

Note: The WDT continues counting up from zero.

6.1.2 Status After Reset

Most units of the XC164CM enter a well-defined default status after a reset is completed. This ensures repeatable start conditions and avoids spurious activities after reset.

Reset Values for the XC164CM Registers

During the reset sequence, the registers of the XC164CM are preset with a default value. Most SFRs, including system registers and peripheral control and data registers, are cleared to zero, so all peripherals and the interrupt system are off or idle after reset. A few exceptions to this rule provide a first pre-initialization, which is either fixed or controlled by input pins (see **Table 6-2**). A number of registers have specific behavior and reset value upon a hardware, software or WDT reset, like PLLCON, RSTCON, FOCON, SYSCON0, RTC_T14, RTC_T14REL, RTC_RTCL and RTC_RTCH. For more information see the corresponding register description.

Table 6-2 Non-Zero Registers after Reset

Register Name	Initial Value	Comments
DPP1	0001_H	Points to data page 1
DPP2	0002_H	Points to data page 2
DPP3	0003_H	Points to data page 3
CP	$FC00_H$	–
STKUN	$FC00_H$	–
STKOV	$FA00_H$	–
SP	$FC00_H$	–
CPUCON1	0007_H	–
CPUCON2	$8FBB_H$	–
ONES	$FFFF_H$	Fixed value
PLLCON	$27X0_H$	Depends on startup mode

Table 6-2 Non-Zero Registers after Reset (cont'd)

Register Name	Initial Value	Comments
RSTCON	$00X0_H$	Depends on startup mode
VECSEG	$00XX_H$	Depends on startup mode
SYSSTAT	$XXXX_H$	Depends on current status
SYSCON3	$9FD0_H$	RTC, TwinCAN, Flash, GPT, SSC0, ASC0, ADC enabled
FCONCS7	0027_H	–
ADDRSEL7	2000_H	–
CCU6_INP	3940_H	–
ADC_CTR0	1000_H	–
CAN_ACR	0001_H	–
CAN_BCR	0001_H	–
RTCCON	8003_H	–
RTC_T14	$UUUU_H$	Affected by the RTC reset only, triggered by setting the bit RTCRST of the register SYSCON0, and not by hardware, software and watchdog resets.
RTC_T14REL	$UUUU_H$	
RTC_RTCL	$UUUU_H$	
RTC_RTCH	$UUUU_H$	

Operation after Reset

After the internal reset condition is removed, the XC164CM fetches the first instruction from the selected program memory location (depending on the configuration). As a rule, this first location holds a branch instruction to the actual initialization routine that may be located anywhere in the address space.

Note: If the Bootstrap Loader Mode was activated during a hardware reset, the XC164CM does not fetch instructions from the program memory.
The standard bootstrap loader expects data via serial interface ASC0.

Watchdog Timer Operation after Reset

The watchdog timer continues running after the internal reset is complete. It will be clocked with the currently selected clock signal f_{WDT}. After a watchdog/software reset f_{WDT} is not changed, after a hardware reset the frequency is $f_{WDT} = f_{SYS}/2 = f_{MC}/2$. The default reload value is 00_H. Thus, a watchdog timer overflow will occur 2^{16} clock cycles (2^{17} f_{MC} cycles after a hardware reset) after completion of the internal reset (depending on the selected reset length), unless it is disabled, serviced, or reprogrammed in the meantime. If the system reset was caused by a watchdog timer overflow, the WDTR (Watchdog Timer Reset Indication) flag in register SYSSTAT will be set to 1. This indicates the cause of the internal reset to the software initialization routine. WDTR is reset to 0 after each other reset. After the internal reset is complete, the operation of the watchdog timer can be disabled by the DISWDT (Disable Watchdog Timer) instruction prior to the EINIT instruction if using compatibility mode, or anytime in enhanced mode.

The On-Chip RAM Areas after Reset

The contents of the major parts of the on-chip RAMs are preserved during a software reset and a WDT reset. There are two exceptions to this rule:

- A part of the DPRAM (the area $00'FBA0_H$ … $00'FC1F_H$) may be altered during the initialization phase (see **Table 6-1**) and, therefore, should not store data to be preserved beyond a WDT/SW reset.
- During bootstrap loader operation the serially received data is stored in the PSRAM starting at location $E0'0000_H$.

Because a hardware reset can occur asynchronously to an internal operation, it may interrupt a current write operation and so inadvertently corrupt the contents of on-chip RAM. RAM contents are preserved if the hardware reset occurs during Power-Down mode, during Sleep mode, or during Idle mode with no PEC transfers enabled.

Note: After a power-up hardware reset the RAM contents are undefined, of course.

Ports after Reset

During the internal reset sequence, all port pins of the XC164CM are configured as inputs by clearing the associated direction registers, and their pin drivers are switched to the high impedance state. This ensures that the XC164CM and external devices will not try to drive the same pin to different levels.

When the on-chip bootstrap loader was activated during reset, pin TxD0 (alternate port function) will be switched to output mode after the reception of the zero byte.

All other pins remain in the high-impedance state until they are changed by software or peripheral operation.

6.1.3 Application-Specific Initialization Routine

After a reset, the modules of the XC164CM must be initialized to enable their operation on a given application. This initialization depends on the task to be performed by the XC164CM in that application and on some system properties such as operating frequency, external circuitry connected, etc.

Typically, the following initialization should be done before the XC164CM is prepared to run the actual application software:

Bootstrap Loader (optional)

Bootstrap loader mode can be used for initial Flash programming.

System Stack

The default setup for the system stack (size, stack pointer, upper and lower limit registers) can be adjusted to application-specific values. After reset, registers SP and STKUN contain the same reset value 00'FC00$_H$, while register STKOV contains 00'FA00$_H$. With the default reset initialization, 256 words of system stack are available in the DPRAM, where the system stack selected by the SP grows downwards from 00'FBFE$_H$. The system stack may be moved to the DSRAM and its size can be adjusted to the application's requirements.

Note: The interrupt system, which is disabled upon completion of the internal reset, should remain disabled until the SP is initialized.
Traps (including NMI) may occur, although the interrupt system is still disabled.

Register Bank

The location of a global register bank is defined by the context pointer (CP) and can be adjusted to an application-specific bank before the general purpose registers (GPRs) are used. After reset, register CP contains the value FC00$_H$, i.e. the register bank selected by the CP grows upward from 00'FC00$_H$.

On-Chip RAM

Depending on the application, the user may wish to initialize portions of the internal writable memory (DPRAM/DSRAM/PSRAM) before normal program operation. After the register bank has been selected by programming the CP register, the desired portions of the internal memory can easily be initialized via indirect addressing.

Interrupt System

After reset, the individual interrupt nodes and the global interrupt system are disabled. In order to enable interrupt requests, the nodes must be assigned to their respective interrupt priority levels and must be enabled. The vector table can be adjusted if the default properties do not fit. Register VECSEG defines the vector table's location, bitfield VECSC in register CPUCON1 defines the vector spacing. The vector locations must receive branch instructions to the respective exception handlers, except if the fast interrupt mechanism is used. The interrupt system must globally be enabled by setting bit IEN in register PSW. To avoid such problems as the corruption of internal memory locations caused by stack operations using an uninitialized stack pointer, care must be taken not to enable the interrupt system before the initialization is complete.

Ports

Generally, all ports of the XC164CM are switched to input upon reset activation. After reset, some pins may be automatically controlled, such as TxD in Boot mode. Pins to be used for general purpose IO must be initialized via software. The required mode (input/output, open drain/push pull, input threshold, etc.) depends on the intended function for a given pin.

Peripherals

Upon reset activation the XC164CM's on-chip peripheral modules enter a defined default state (see respective peripheral description) in which they are disabled from operation. In order to use a certain peripheral it must be initialized according to its intended operation in the application.

This includes enabling the peripheral, selecting the operating mode (such as counter/timer), operating parameters (such as baudrate), enabling interface pins (if required), assigning interrupt nodes to the respective priority levels, etc.

After these standard initialization actions, application-specific actions may be required, such as asserting certain levels to output pins, sending codes via interfaces, latching input levels, etc.

Watchdog Timer

After reset, the watchdog timer is active and counting its default period. If the watchdog timer is to remain active the desired period should be programmed by selecting the appropriate prescaler value and reload value. Otherwise, the watchdog timer must be disabled before EINIT.

Termination of Initialization

The software initialization routine should be terminated with the EINIT instruction. This instruction has been implemented as a protected instruction.

Execution of the EINIT instruction has the following effects:

- Disables the action of the DISWDT instruction (unless enhanced mode is selected),
- Switches the register security level to “write-protected mode” (see **Section 6.3.5**),

6.1.4 System Startup Configuration

Although most of the programmable features of the XC164CM are selected by software either during the initialization phase or repeatedly during program execution, some features must be selected earlier because they are used for the first access of the program execution.

These configurations are accomplished by latching the logic levels at a number of pins at the end of the internal reset sequence. During reset, internal pull-up devices are active on those lines. They ensure inactive/default levels at pins which are not driven externally. External pull-down/pull-up devices may override the default levels in order to select a specific configuration. In addition, not all pins must be controlled for all modes.

Many configurations can, therefore, be coded with a minimum of external circuitry. If pin TRST is latched low all four configuration pins are ignored (see **Table 6-3**).

Note: The load on those pins to be latched for configuration must be small enough for the internal pull-up/pull-down device to sustain the default level, or external pull-up/pull-down devices must ensure this level.
Those pins whose default level will be overridden must be pulled low/high externally.
Ensure that the valid target levels are reached by the end of the reset sequence. There is a specific application note to illustrate this.
A hardware reset can be terminated as soon as the target levels are reached. The XC164CM automatically waits until oscillator, PLL, and Flash are up and operable.

The levels on pins P9.4, P9.5, P1H.4, and P1H.5 are latched whenever the internal reset phase is left after a hardware reset (see **Table 6-1**). Depending on the intended configuration, two to four pins must be controlled externally (see **Table 6-3**).

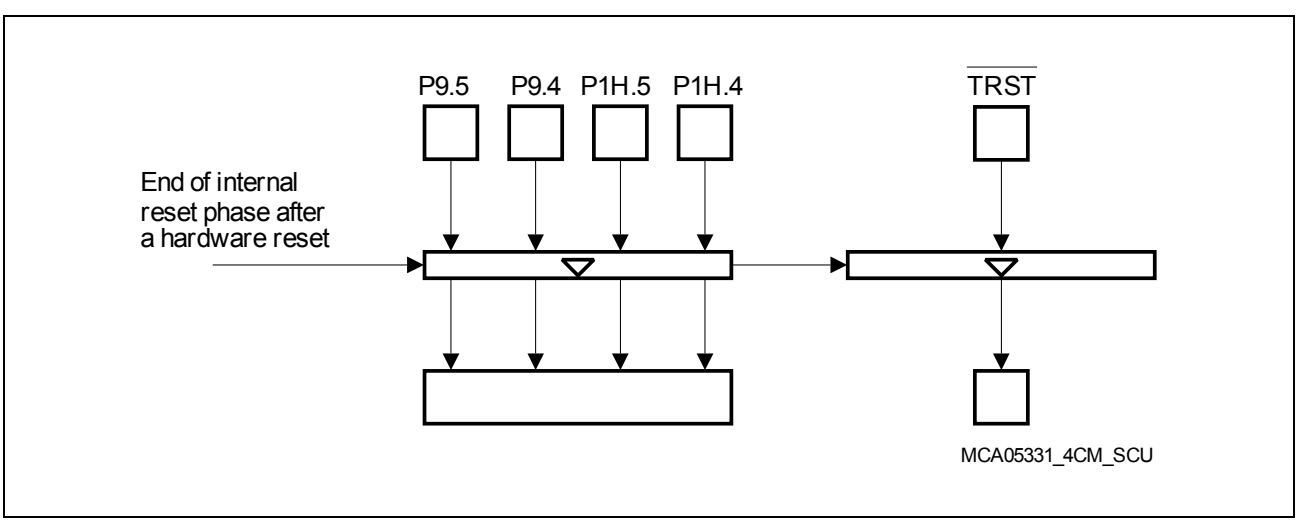


Figure 6-2 Latching Configuration

Table 6-3 Basic Startup Configuration via External Circuitry

Latched Configuration	$\overline{TRST}$	P1H.5	P1H.4	P9.5	P9.4
Start internal (OCDS disabled)	0	X	X	X	X
Start internal	1	X	X	1	1
Boot via ASC0	1	X	X	0	1
Adapt Mode	1	1	1	0	0
Reserved	1	All other combinations			

Note: The pull-ups on the configuration pins are activated while $\overline{TRST}$ = 1 and the hardware reset signal is active.

The initial clock generation mode of the XC164CM is defined by the reset value of register PLLCON, selecting bypass mode.

- Internal start: PLLCON = 2710_H (2:1 prescaler to cover maximum frequency range)
- Bootstrap loader: PLLCON = 2700_H (direct drive to reach maximum baudrate)

The user can change this startup configuration at any time.

The On-Chip Bootstrap Loader allows the start code to be moved into the internal PSRAM of the XC164CM via the serial interface ASC0. The XC164CM will then execute the loaded start code out of the PSRAM.

Default: The XC164CM starts fetching code from location $C0'0000_H$, the bootstrap loader is off.

Adapt Mode

In this mode, the XC164CM goes into a passive state similar to its state during reset. The pins of the XC164CM float to tristate or are deactivated via internal pull-up/pull-down devices, as described for the reset state. The on-chip oscillator and the real-time clock are disabled.

This mode allows a XC164CM mounted to a board to be virtually switched off. This enables an emulator to control the board's circuitry even though the original XC164CM remains in place. The original XC164CM may resume control of the board after a reset sequence not selecting Adapt Mode.

Default: Adapt Mode is off.

Note: When XTAL1 is fed by an external clock generator (while XTAL2 is left open), this clock signal may also be used to drive the emulator device.
However, if a crystal is used, the emulator device's oscillator can use this crystal only if at least XTAL2 of the original device is disconnected from the circuitry (the output XTAL2 will be driven high in Adapt Mode).
Adapt mode can be activated only upon an external reset.

6.1.5 Reset Behavior Control

The reset behavior is controlled by a set of control/status registers. The status information can be used by the initialization code to execute different actions depending on the reset source.

The reset control register RSTCON is used by the application to program the length of the internal reset phase.

RSTCON
Reset Control Register **mem ($F1E0_H$/--)** **Reset Value: $00X0_H$[1)2)]**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	-	RSTLEN		
-	-	-	-	-	-	-	-	-	-	-	-	-	rw		

1) The reset value is only valid for a hardware reset.
2) Reset value: 0010_H after Internal and ASC bootstrap startup.

Field	Bits	Type	Description
RSTLEN	[2:0]	rw	**Reset Length Control**[1)] The duration of the next internal reset phase is $t_{RST} = t_{WDT} \times 2^{RSTLEN}$. 000 1 t_{WDT}: default duration after hardware reset … … 111 128 t_{WDT}: maximum duration

1) RSTLEN is always valid for the **next** reset sequence. An initial power up reset, however, is controlled by external hardware and is expected to last considerably longer than any configurable reset sequence.

*Note: RSTCON is protected by the register security mechanism (see **Section 6.3.5**). RSTCON can only be accessed via its long (mem) address.*

6.2 Clock Generation

All activities of the XC164CM's controller hardware and its on-chip peripherals are controlled by clock signals which are generated by the Clock Generation Unit (CGU).

This reference clock is generated in three stages:

Oscillator

The on-chip Pierce oscillator (main oscillator) can either run with an external crystal and appropriate oscillator circuitry or it can be driven by an external oscillator or another clock source.

Clock Generation and Frequency Control

The input clock signal of the main oscillator feeds the controller hardware:

- directly, divided by a programmable prescaler factor (1 … 60), either providing phase-coupled operation (factor = 1) or operating the device at low frequencies to reduce power consumption (factor >> 1)
- via an on-chip Phase Locked Loop (PLL) providing maximum performance on low input frequency

Clock Distribution

The clock signals are distributed via separate clock drivers which feed the CPU itself and groups of peripheral modules. Certain sections of the device can be supplied with a prescaled clock signal.

Note: The RTC is fed with the prescaled main oscillator clock via a separate clock driver, so it is not affected by the clock control functions.

6.2.1 Oscillator

The main oscillator of the XC164CM is a power optimized Pierce oscillator providing an inverter and a feedback element. Pins XTAL1 and XTAL2 connect the inverter to the external crystal. The standard external oscillator circuitry (see **Figure 6-3**) comprises the crystal, two low end capacitors and series resistor (Rx2) to limit the current through the crystal. A test resistor (R_Q) may be temporarily inserted to measure the oscillation allowance (negative resistance) of the oscillator circuitry.

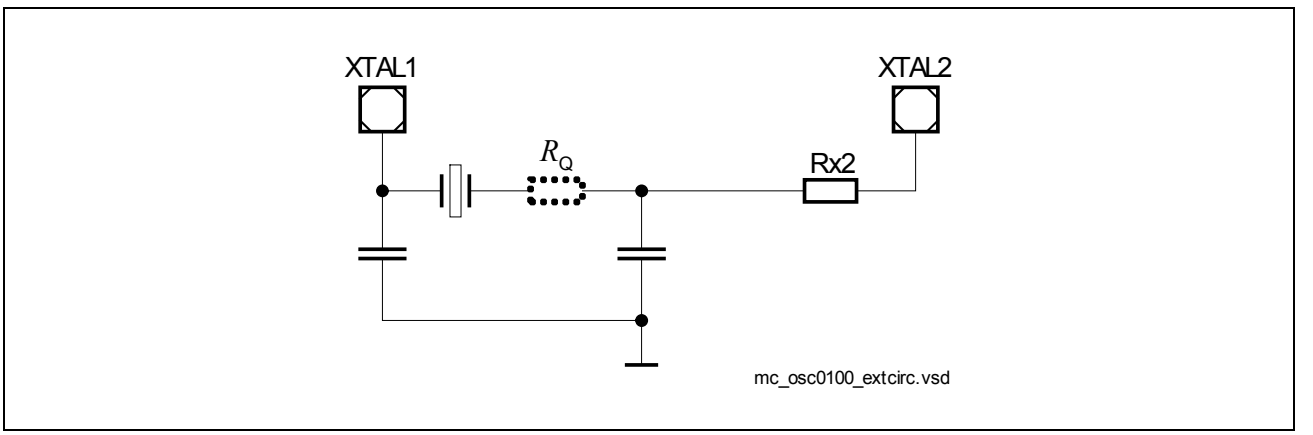


Figure 6-3 External (Main) Oscillator Circuitry

The on-chip oscillator is optimized for an input frequency range of 4 to 16 MHz.

An external clock signal (e.g. from an external oscillator or from a master device) may be fed to the input XTAL1. The Pierce oscillator then is not required to support the oscillation itself but is rather driven by the input signal. In this case the input frequency range may be 0 to 50 MHz (please note that the maximum applicable input frequency is limited by the device's maximum clock frequency).

Note: ***Oscillator measurement*** *within the final target system is strongly recommended to verify the input amplitude and to determine the actual oscillation allowance (margin or negative resistance) for the oscillator-crystal system.*
The measurement technique is described in a specific application note about oscillators (available via your representative or www).

The main oscillator is automatically switched off during Power-Down mode and Sleep mode, unless the RTC remains on. Switching off the main oscillator is useful to further reduce the power consumption during phases where only minimum system life functions must be maintained.

Main Oscillator Gain Reduction

The main oscillator starts with a high drive level (gain) during and after a hardware reset to ensure safe startup behavior in the beginning (force the crystal oscillation). The beginning of the crystal oscillation is indicated by bit OSCLOCK = 1. When a stable oscillation has been reached after oscillator startup (amplitude more than 90% of its maximum), the gain of the main oscillator can be reduced. This reduces the oscillator's power consumption which is especially important in power reduction modes.

This gain reduction is induced by software and so is transparent in existing software. The oscillator gain is reduced by setting bit OSCGRED in register SYSCON0 (see **Section 6.3**). Because the oscillator amplitude is not measured directly, a delay of approximately 2^{15} oscillator clock cycles is required before enabling the gain reduction.

The occurrence of 2^{15} consecutive oscillator clock cycles is indicated by bit OSCSTAB (= 1) in register SYSSTAT.

The oscillator gain reduction is disabled while OSCSTAB = 0. Therefore, software can set bit OSCGRED at any time. If OSCGRED is set before OSCSTAB = 1, the gain reduction is automatically delayed.

Note: After the delay indicated by OSCSTAB = 1 the oscillation has reached more than 90% of its maximum amplitude with an optimized oscillator circuitry.
***Oscillator measurement** (margin or negative resistance) for the oscillator-crystal system must be executed also in reduced-gain mode if this mode is intended in the application.*
If the main oscillator is switched off during sleep mode, both bits OSCLOCK and OSCSTAB are cleared and the oscillator startup begins anew. This ensures a safe oscillator startup after wake-up.

6.2.2 Clock Generation and Frequency Control

The Clock Generation Unit uses a programmable on-chip PLL with multiple prescalers to generate the clock signals for the XC164CM with high flexibility. The internal operation of the XC164CM is controlled by the internal master clock f_{MC}. The master clock f_{MC} is the reference clock signal, and is used for TwinCAN and is output to the external system.

CPU and EBC are clocked with the CPU clock signal f_{CPU}. The CPU clock can have the same frequency as the master clock ($f_{CPU} = f_{MC}$) or can be the master clock divided by two: $f_{CPU} = f_{MC}/2$. This factor is selected by bit CPSYS in register SYSCON1.

The other peripherals are supplied with the system clock signal f_{SYS} which has the same frequency as the CPU clock signal ($f_{SYS} = f_{CPU}$).

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. With these options the master clock can be adjusted to a wide range of frequencies. PLL operation achieves maximum performance even from moderate crystal frequencies, dividing the oscillator clock runs the system at low frequency, greatly reducing its power consumption.

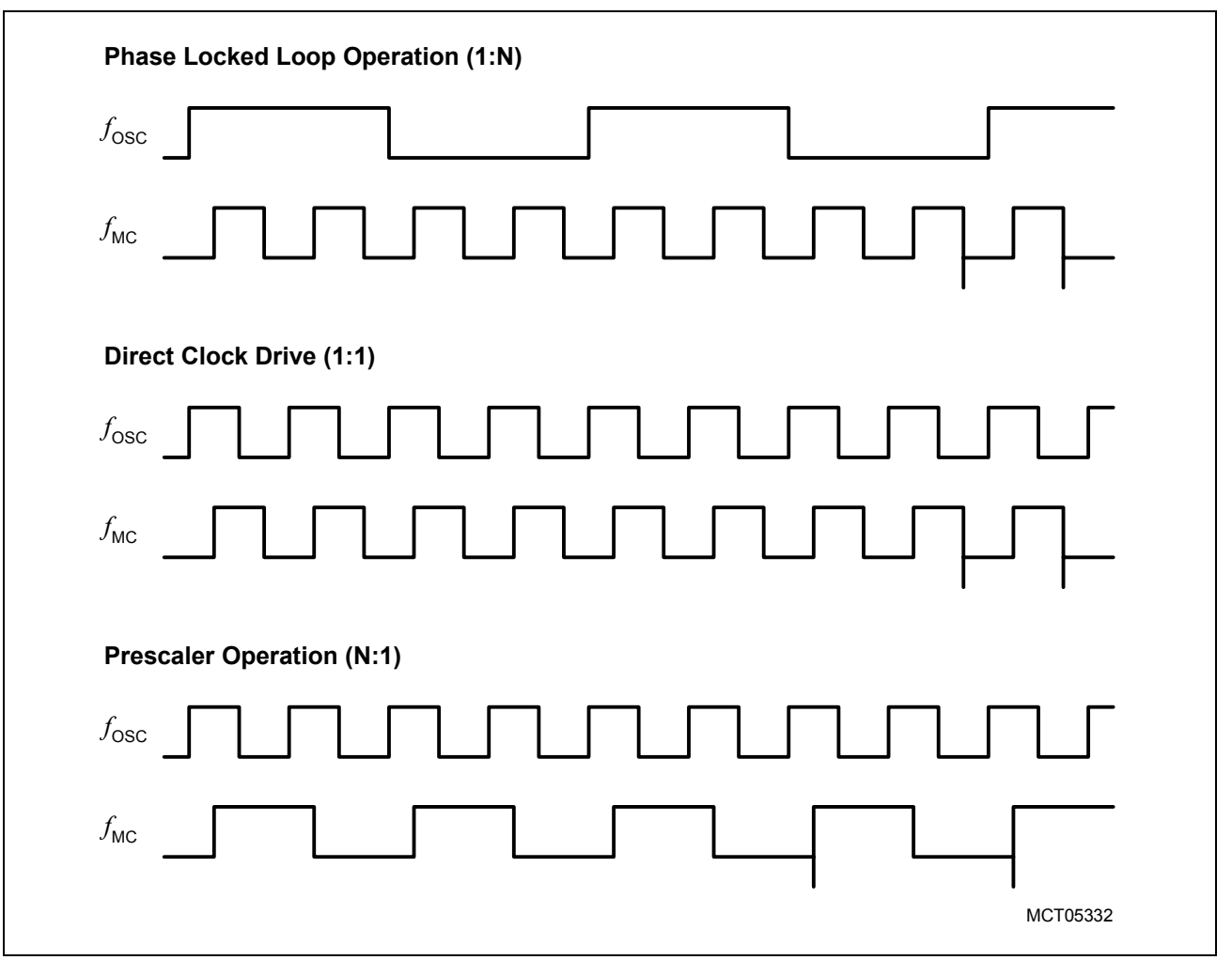


Figure 6-4 Generation Mechanisms for the Master Clock

*Note: The example for PLL operation shown in **Figure 6-4** refers to a PLL factor of 1:4, the example for prescaler operation refers to a divider factor of 2:1.*

The Clock Generation Unit (CGU) summarizes the following required functions to generate the clock signals used in the XC164CM:

- Generation of the master clock signal from the oscillator clock according to user-programmed mode and factor
- Generation of clock signals for specific functional areas
- Control the oscillator operation according to the XC164CM's operating mode
- Generation of an interrupt request in case of detected malfunctions of the clock system

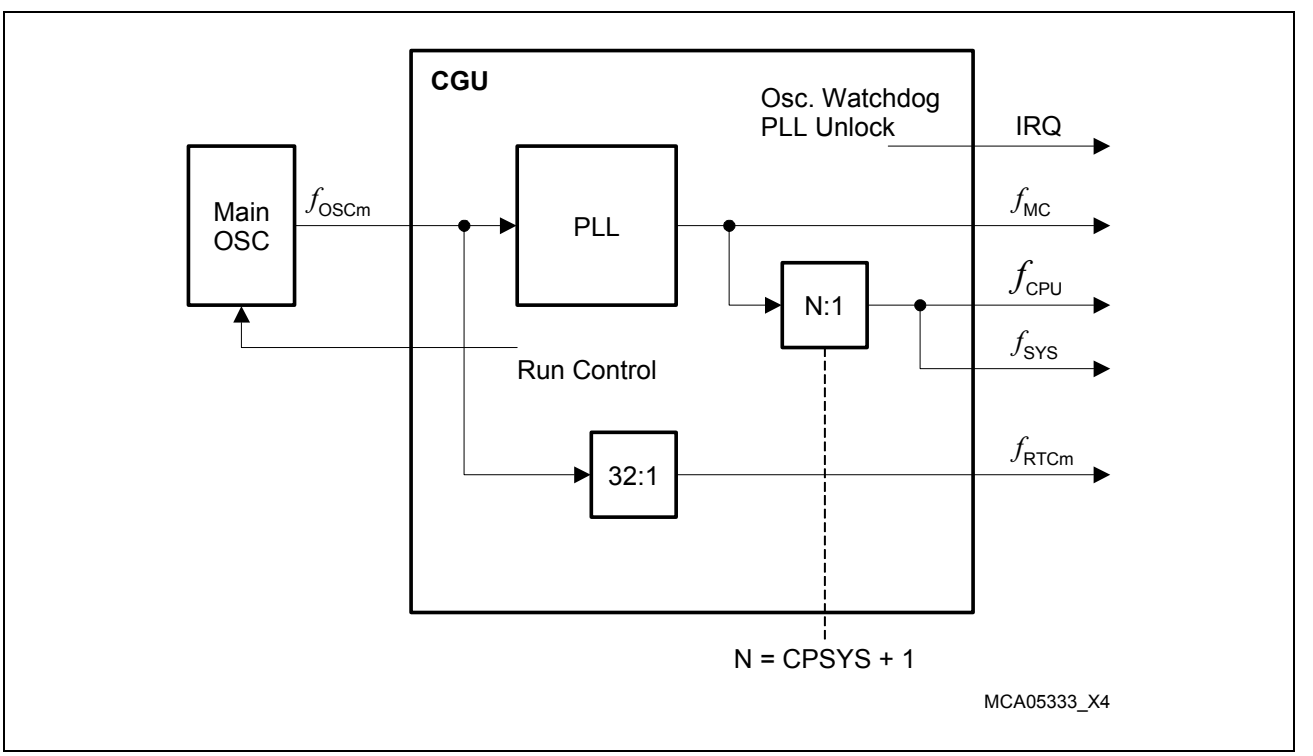


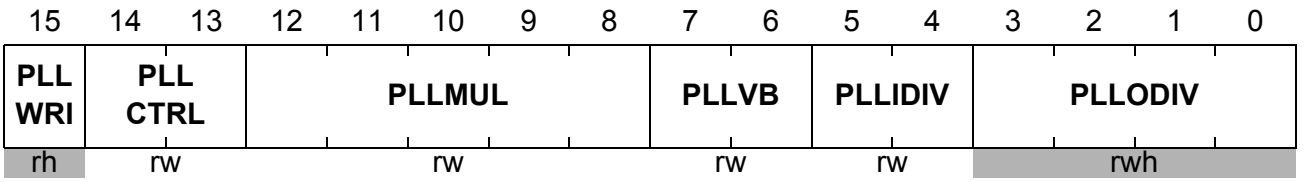
Figure 6-5 Basic Structure of the Clock Generation Unit

Note: The divider factor for the CPU clock and the system clock is selected by bit CPSYS in register SYSCON1.

The master clock signal is generated by the highly-flexible on-chip PLL. The PLL block can multiply the oscillator clock frequency by a programmable factor (1:0.15 … 1:10) to achieve high performance even from moderate crystal frequencies. In bypass mode the oscillator clock is divided by a factor of 1:1 … 60:1 to achieve direct coupling to the oscillator clock signal (1:1) or reduce the system frequency to save power.

The used mechanism to generate the master clock and the respective parameters are selected via the PLL control register PLLCON.

PLLCON
PLL Control Register **ESFR ($F1D0_H/E8_H$)** **Reset Value: $27X0_H$**



Field	Bits	Type	Description
PLLWRI	15	rh	**PLLCON Write Ignore Flag** 0 Register PLLCON may be written 1 Write cycles to register PLLCON are ignored
PLLCTRL	[14:13]	rw	**PLL Operation Control** 00 Bypass PLL clock mult., the VCO is off 01 Bypass PLL clock mult., the VCO is running 10 VCO clock used, input clock switched off 11 VCO clock used, input clock connected
PLLMUL	[12:8]	rw	**PLL Multiplication Factor** … by which the PLL multiplies its input frequency (valid values: $1'1111_B$ … $0'0111_B$)[1] $f_{VCO} = f_{IN} \times (PLLMUL+1)$
PLLVB	[7:6]	rw	**PLL VCO Band Select**[2] Value, VCO output frequency, Base frequency 00 100 … 150 MHz, 20 … 80 MHz 01 150 … 200 MHz, 40 … 130 MHz 10 200 … 250 MHz, 60 … 180 MHz 11 Reserved
PLLIDIV	[5:4]	rw	**PLL Input Divider** Adjusts the oscillator frequency to the defined input frequency range of the PLL (valid values: 11_B … 00_B) $f_{IN} = f_{OSC} / (PLLIDIV+1)$
PLLODIV	[3:0]	rwh	**PLL Output Divider** Scales the PLL output frequency to the desired CPU frequency (valid values: 1110_B … 0000_B)[3] $f_{MC} = f_{VCO} / (PLLODIV+1)$

1) Multiplication factors below N = 8 (PLLMUL = 7) may affect stability. For example, this may lead to undesired VCO frequencies due to increased noise-susceptibility.

2) The VCO band must be selected to contain the intended VCO frequency (8 MHz × 20 = 160 MHz --> band 01_B).

3) Value 1111_B is reserved for emergency mode operation and cannot be entered via software.

*Note: PLLCON is protected by the register security mechanism (see **Section 6.3.5**). The reset value depends on the initial system startup configuration.*
Register PLLCON is not affected by a software reset. It is affected by hardware and watchdog resets, which restore the hardware reset value.

The clock generation path is controlled by a state machine according to the selection in register PLLCON. Bit PLLWRI = 0 indicates when this state machine is ready to accept a new selection value in PLLCON. To support monitoring, register PLLCON accepts the (new) selection for clock configuration when written, and returns the actual state of the clock generation mechanism when read.

The PLL module contains the frequency multiplication logic, a set of prescalers, the lock detection, and the oscillator watchdog.

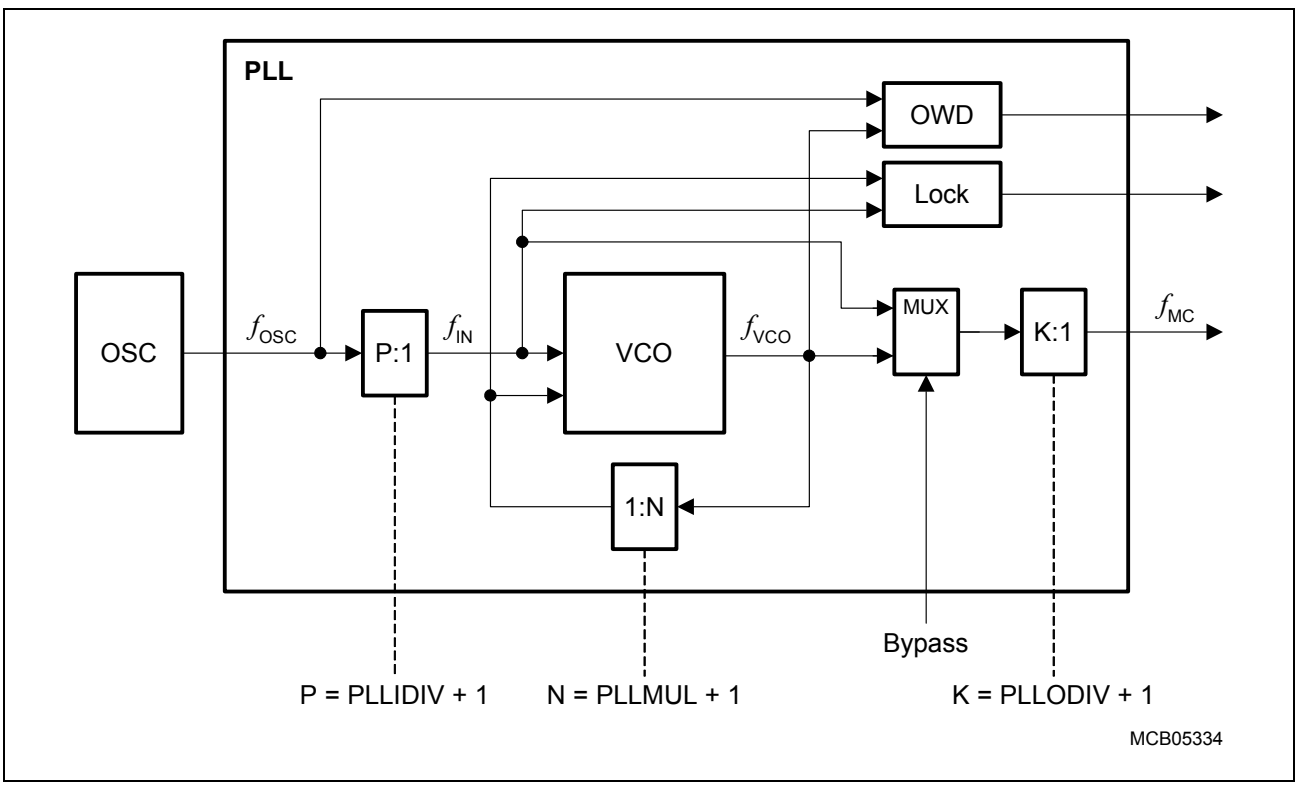


Figure 6-6 PLL Block Diagram

PLL Operation

When PLL operation is configured (PLLCTRL = 11_B), the XC164CM's input clock is fed to the on-chip Phase Locked Loop circuit which can multiply its frequency by a factor of up to **F** = 10 and generates a master clock signal with 50% duty cycle, i.e. $f_{MC} = f_{OSC} \times \mathbf{F}$.

The on-chip PLL circuit allows operation of the XC164CM on a low frequency external clock while still providing maximum performance. The PLL also provides fail safe mechanisms which allow the detection of frequency deviations and the execution of emergency actions in case of an external clock failure.

When the PLL detects a missing input clock signal it generates an interrupt request. This warning interrupt indicates that the PLL frequency is no longer locked, i.e. no longer related to the oscillator frequency. This occurs when the input clock is unstable and especially when the input clock fails completely, such as due to a broken crystal. In this case the synchronization mechanism will reduce the PLL output frequency down to the PLL's base frequency (depending on the VCO band selected by bitfield PLLVB) and select the safety output divider factor K = 16. The base frequency is still generated and allows the CPU to execute emergency actions in case of a loss of the external clock. The master clock in this emergency case is, therefore, $f_{MCe} = f_{VCObase}/16$.

Note: During a hardware reset the lowest VCO band is selected together with factor 16.

On power-up the PLL provides a stable clock signal, even if there is no external clock signal (in this case the PLL will run on its base frequency). The PLL starts synchronizing with the external clock signal as soon as it is available. After stable oscillations of the external clock within the specified frequency range the PLL locks to the external clock. This means the PLL will be synchronous with this clock at a frequency of $\mathbf{F} \times f_{OSC}$.

The PLL circuit constantly synchronizes the master clock to the input clock. This synchronization is done smoothly, i.e. the master clock frequency does not change abruptly. Due to the fact that the external frequency is 1/**F**th of the PLL output frequency the output frequency may be slightly higher or lower than the desired frequency. The slight variation causes a jitter of f_{MC} which also affects the duration of individual master clock periods. This jitter is irrelevant for longer time periods. For short periods (1 … 4 CPU clock cycles) it remains below 9%.

The clock signal passes through several blocks (see **Figure 6-6**). The total clock multiplication factor **F** results from the input divider (P:1), the multiplication factor (1:N), and the output divider (K:1), so **F** = (PLLMUL+1) / ((PLLIDIV+1) × (PLLODIV+1)).

The input clock divider adjusts the oscillator clock frequency to the input frequency range for which the PLL is optimized ($f_{IN} = f_{OSC}$ / (PLLIDIV+1) = 4 … 35 MHz).

The PLL core multiplies the adjusted input frequency within the selected VCO band by a selectable factor ($f_{VCO} = f_{IN} \times$ (PLLMUL+1)). The valid VCO band (PLLVB) must be selected according to the intended VCO frequency.

Note: This PLL core can be bypassed, e.g. while the PLL is locking to a given factor, to ensure a proper CPU clock signal, or if the PLL is not used for clock generation.

The output clock divider scales the VCO's output frequency by a selectable factor to generate the master clock signal ($f_{MC} = f_{VCO}$ / (PLLODIV+1)). Adjusting this factor may be used to control the operating frequency of the XC164CM without having to reprogram the PLL core itself.

Figure 6-7 summarizes the subsequent steps the master clock generation.

The maximum multiplication factor $\mathbf{F}_{max}$ is employed when the highest possible master clock frequency (40 MHz) shall be generated from the lowest possible input clock frequency (4 MHz). Therefore, the maximum usable factor $\mathbf{F}_{max}$ is 10.

Application software can select the optimum clock generation mode via registers PLLCON and SYSCON1. After reset the XC164CM enters a default clock generation mode.

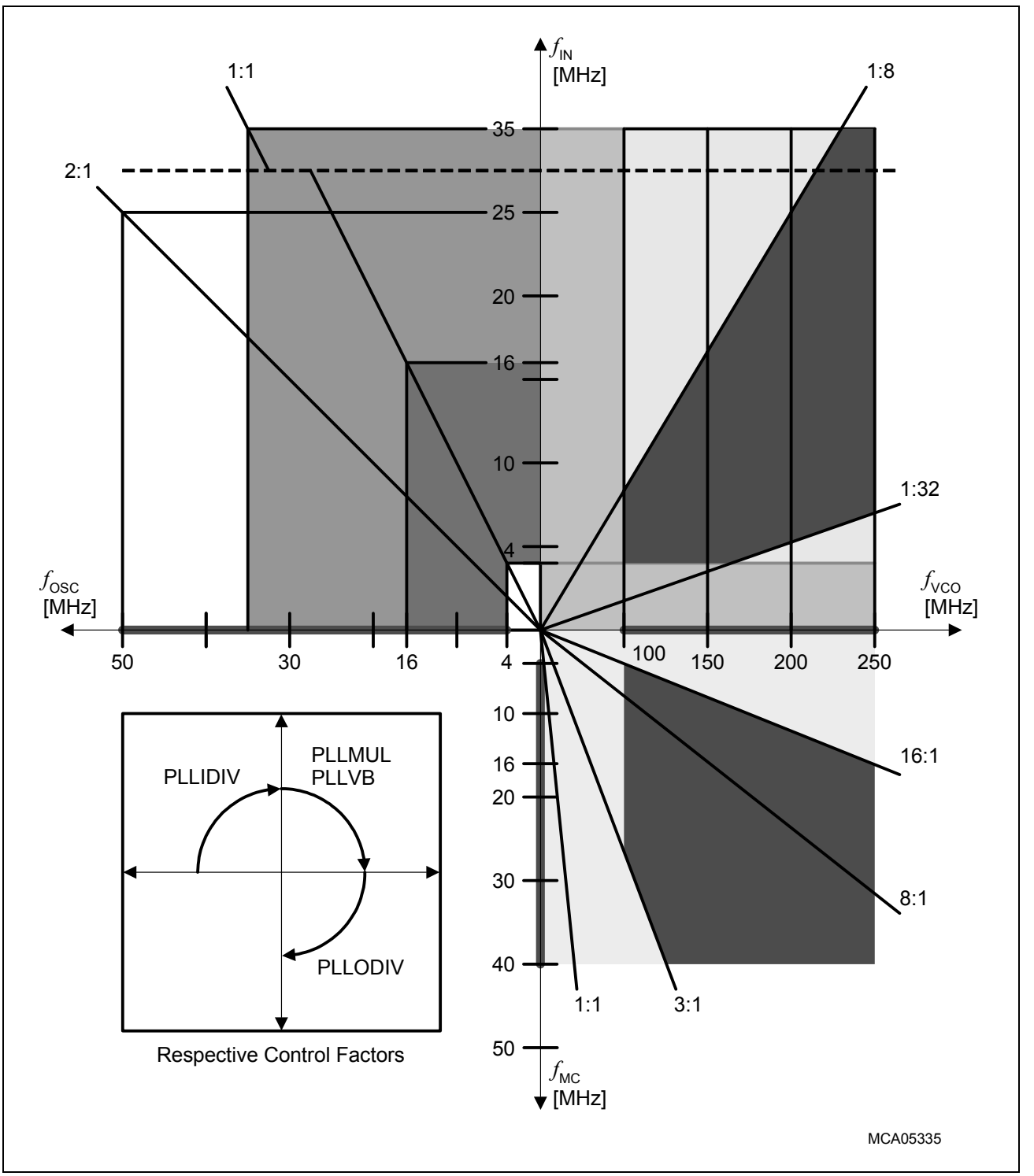


Figure 6-7 Valid Clock Frequency Bands

Bypass Operation

When bypass operation is configured (PLLCTRL = $0X_B$) the clock signal does not pass through the PLL core and the master clock is derived from the internal oscillator (input clock signal XTAL1) through the input- and output-prescalers:

$f_{MC} = f_{OSC}$ / ((PLLIDIV+1) × (PLLODIV+1)).

If both divider factors are selected as 1 (PLLIDIV = PLLODIV = 0) the frequency of f_{MC} directly follows the frequency of f_{OSC} so the high and low time of f_{MC} is defined by the duty cycle of the input clock f_{OSC}.

The lowest master clock frequency is achieved by selecting the maximum values for both divider factors: $f_{MCmin} = f_{OSC}$ / ((3+1) × (14+1)) = f_{OSC} / 60.

Master Clock Duty Cycle

The master clock signal f_{MC} is formed by the output divider (factor K = PLLODIV+1). The duty cycle of f_{MC} depends on the selected output divider factor. For all even factors the duty cycle is 50%, for all odd factors the duty cycle is (K - 1)/(K × 2). The worst case here is K = 3, which leads to a duty cycle of (3 - 1)/(3 × 2) = 2/6 = 33%.

6.2.3 Clock Distribution

The operating clock signals are distributed to the controller hardware via several clock drivers. This establishes the corresponding clock domains summarized in **Table 6-4**. The real time clock RTC is clocked via a separate clock driver which delivers the prescaled main oscillator clock.

Table 6-4 Clock Domains

Clock Domain	Domain Clock	Active Mode	Idle Mode	Sleep, P. Down	Connected Circuitry	Module Clock
LXBus	f_{MC}	ON	ON	Off	TwinCAN	f_{CAN}
					SCU[1)]	–
CPU	f_{CPU}	ON	Off	Off	CPU, DPRAM, EBC, OCDS, Flash/ROM, PSRAM, DSRAM	–
PDBus	f_{SYS}	ON	ON	Off	ADC	f_{ADC}
					ASC0, ASC1	f_{ASC}
					CAPCOM2	f_{CC}
					CAPCOM6	f_{CC6}
					GPT12	f_{GPT}
					SSC0, SSC1	f_{SSC}
					Ports, RTC[2)], WDT, SCU (Intr. Ctrl., Reg. access)[1)]	–
RTC	f_{RTC}	ON	ON	ON/Off	RTC[2)]	–

1) As the clock generation unit is part of the SCU, the SCU consequently belongs to more than one clock domain.

2) The RTC is part of the PDBus clock domain which provides its operating clock. The count clock signal is derived directly from the main oscillator (as selected).

*Note: All PDBus peripherals are provided with the clock signal f_{SYS}. Within a peripheral description, however, this clock signal is called according to the peripheral's name. **Table 6-4** shows this in column "Module Clock".*

6.2.4 Oscillator Watchdog

The XC164CM provides an Oscillator Watchdog (OWD) which monitors the clock signal fed to input XTAL1 of the on-chip oscillator (either with a crystal or via external clock drive) in bypass mode (not if the PLL provides the master clock). For this operation, the PLL provides a VCO clock signal (base frequency) which is used to supervise transitions on the oscillator clock. This VCO clock is independent from the XTAL1 clock. When the expected oscillator clock transitions are missing, the OWD activates the PLL Unlock/OWD interrupt node and supplies the CPU with an emergency clock instead of the selected oscillator clock. Under these circumstances the VCO will oscillate with its base frequency in the selected VCO band. The emergency clock frequency is $f_{VCO}/16$.

If the oscillator clock fails while the PLL provides the master clock the system will be supplied with the PLL base frequency anyway.

With this emergency clock signal the CPU can either execute a controlled shutdown sequence bringing the system into a defined and safe idle state, or it can provide an emergency operation of the system with reduced performance based on this (normally slower) emergency clock.

Note: In special cases, when bypass mode is selected with a high prescaler factor, the emergency clock frequency may be higher than the originally intended frequency.

The oscillator watchdog can be disabled by switching the VCO off (PLLCTRL = 00_B). In this case the VCO remains idle and provides no clock signal, while the master clock signal is derived directly from the oscillator clock. This reduces power consumption, but also no interrupt request will be generated in case of a missing oscillator clock.

6.2.5 Interrupt Generation

When the PLL leaves its locked state or when the OWD detects an improper clock input signal, the CGU issues an interrupt request.

PLL_IC
PLL Interrupt 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	PLL IR	PLL 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.

6.2.6 Generation of an External Clock Signal

The external circuitry can be provided with a clock signal either to operate external peripherals or for reference purposes. Two types can be selected:

- CLKOUT directly outputs the master clock signal f_{MC} and is mainly used as a timing reference
- FOUT outputs a clock signal with a programmable frequency and can be used to drive and control external circuitry

The programmable frequency output signal f_{OUT} can be controlled via software (contrary to CLKOUT), 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 XC164CM can be influenced, i.e. run at a scalable frequency or temporarily can be switched off completely.

This clock signal is generated via a reload counter, so the output frequency can be selected in small steps. An optional toggle latch provides a clock signal with a 50% duty cycle.

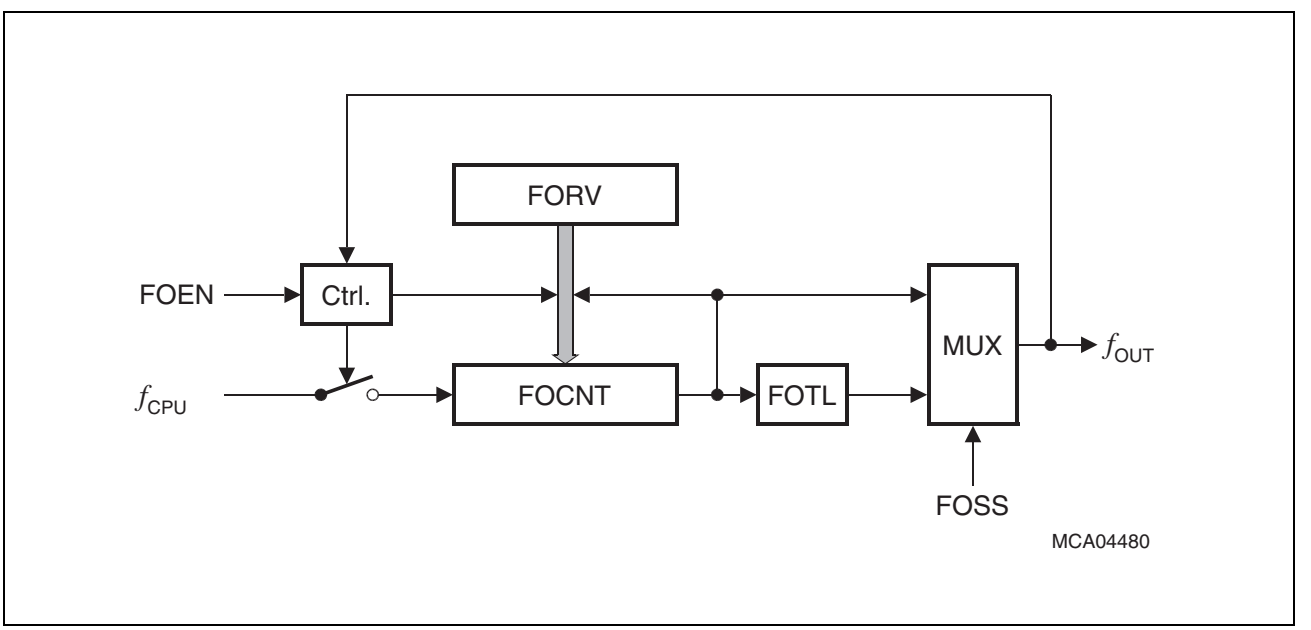


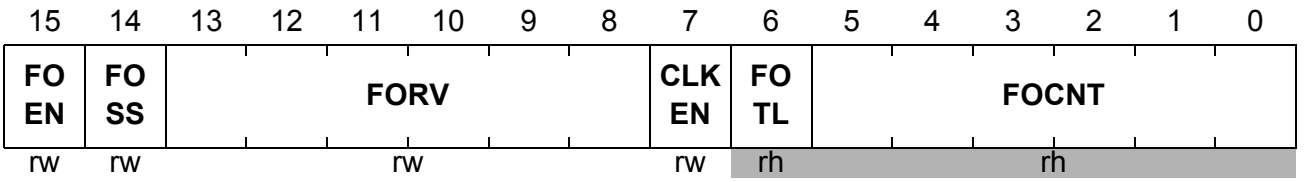
Figure 6-8 Clock Output Signal Generation

Signal f_{OUT} always provides complete output periods (see **Figure 6-9**):

- When f_{OUT} is started (FOEN --> 1), FOCNT is loaded from FORV
- When f_{OUT} is stopped (FOEN --> 0), FOCNT is stopped when f_{OUT} has reached (or is) 0.

Register FOCON provides control over the output signal generation (output signal type, frequency, waveform, activation) as well as all status information (counter value, FOTL).

FOCON
Frequ. Output Control Reg. **SFR ($FFAA_H/D5_H$)** **Reset Value: 0000_H**



Field	Bits	Type	Description
FOEN	15	rw	**Frequency Output Enable** 0 Frequency output generation stops when signal f_{OUT} is or becomes low. 1 FOCNT is running, f_{OUT} is gated to pin. First reload after 0-1 transition.
FOSS	14	rw	**Frequency Output Signal Select** 0 Output of the toggle latch: duty cycle = 50%. 1 Output of the reload counter: duty cycle depends on FORV.
FORV	[13:8]	rw	**Frequency Output Reload Value** Is copied to FOCNT upon each underflow of FOCNT.
CLKEN	7	rw	**CLKOUT Enable** 0 CLKOUT signal disabled, P3.15 is IO or outputs FOUT (default) 1 P3.15 outputs signal CLKOUT ($f = f_{MC}$)
FOTL	6	rh	**Frequency Output Toggle Latch** Is toggled upon each underflow of FOCNT.
FOCNT	[5:0]	rh	**Frequency Output Counter**

Note: Bitfield FOCNT and bit FOTL cannot be written. This prevents the generation of invalid clock cycles when writing to register FOCON, for example to change the output frequency or to stop the output clock signal.
*FOCON is write protected after the execution of EINIT by the register security mechanism (see **Section 6.3.5**).*

During the generation of CLKOUT and f_{OUT} the shared pin is automatically switched to output.

While f_{OUT} is disabled, the pin is controlled by the port latch and the direction latch. Pin FOUT must be switched to output and the port latch must be 0 in order to maintain the f_{OUT} inactive level at the pin.

Signals CLKOUT and f_{OUT} in the XC164CM are alternate output pin functions. A priority ranking determines which function controls the shared pin:

Table 6-5 Priority Ranking for Shared Output Pin

Priority	Function	Control
1	CLKOUT	CLKEN = 1, FOEN = x
2	FOUT	CLKEN = 0, FOEN = 1
3	General purpose IO	CLKEN = 0, FOEN = 0

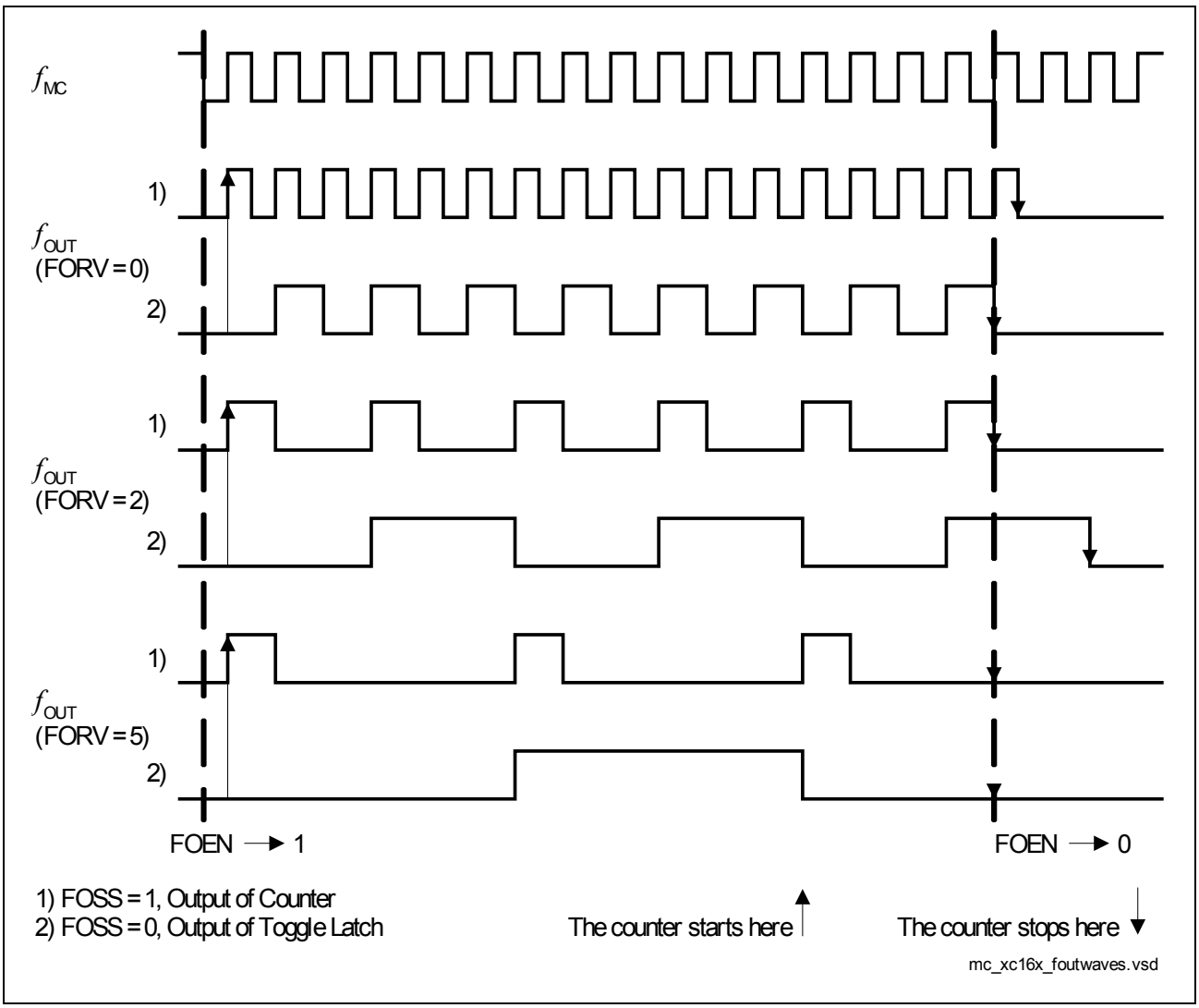


Figure 6-9 Signal Waveforms

Note: The output signal (for FOSS = 1) is high for the duration of one f_{MC} cycle for all reload values FORV > 0. For FORV = 0 the output signal corresponds to f_{MC}.

Output Frequency Calculation

The output frequency can be calculated as $f_{OUT} = f_{MC} / ((FORV + 1) \times 2^{(1 - FOSS)})$,
so $f_{OUTmin} = f_{MC} / 128$ (FORV = $3F_H$, FOSS = 0),
and $f_{OUTmax} = f_{MC} / 1$ (FORV = 00_H, FOSS = 1).

Table 6-6 Selectable Output Frequency Range for f_{OUT}

f_{MC}	f_{OUT} in [kHz] for FORV = xx, FOSS = 1/0					FORV for f_{OUT} = 1 MHz	
	00_H	01_H	02_H	$3E_H$	$3F_H$	FOSS = 0	FOSS = 1
4 MHz	4000 2000	2000 1000	1333.33 666.67	63.492 31.746	62.5 31.25	01_H	03_H
10 MHz	10000 5000	5000 2500	3333.33 1666.67	158.73 79.365	156.25 78.125	04_H	09_H
12 MHz	12000 6000	6000 3000	4000 2000	190.476 95.238	187.5 93.75	05_H	$0B_H$
16 MHz	16000 8000	8000 4000	5333.33 2666.67	253.968 126.984	250 125	07_H	$0F_H$
20 MHz	20000 10000	10000 5000	6666.67 3333.33	317.46 158.73	312.5 156.25	09_H	13_H
25 MHz	25000 12500	12500 6250	8333.33 4166.67	396.825 198.413	390.625 195.313	$0B_H$ (1.04167) $0C_H$ (0.96154)	18_H
33 MHz	33000 16500	16500 8250	11000 5500	523.810 261.905	515.625 257.813	$0F_H$ (1.03125) 10_H (0.97059)	20_H

6.3 Central System Control Functions

Most control functions and status information are tightly coupled to the respective peripheral modules of the XC164CM. However, some of these functions are valid for the complete device, rather than for a specific module. These functions, including the associated control- and status-bits, are part of the SCU.

SYSCON0
General System Control Reg. ESFR ($F1BE_H/DF_H$) Reset Value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTCRST	RTCCM	-	OSCGRED	-	-	-	-	-	-	-	-	-	-	-	-
rwh	rw	-	rw	-	-	-	-	-	-	-	-	-	-	-	-

Field	Bits	Type	Description
RTCRST	15	rwh	**RTC Reset Trigger** 0 No action 1 The RTC module is reset[1)] *Note: RTCRST returns to 0 one SCU clock after being set.*
RTCCM	14	rw	**RTC Clocking Mode[1)]** 0 Synchronous mode: The RTC operates with the system clock. Registers can be read and written. 1 Asynchronous mode:[2)] The RTC operates with the (asynchronous) count clock. No write access is possible.
OSCGRED	12	rw	**Oscillator Gain Reduction Control** 0 No reduction, retain initial gain level 1 Reduce gain (see **Section 6.2.1**)

1) After an RTC reset, the RTC immediately enters the clocking mode currently selected by bit RTCCM.

2) Asynchronous mode is required if the system clock is slower than the $4 \times f_{COUNT}$. This is, of course, the case in Sleep mode or Powerdown mode, where the system clock is disabled, while the RTC shall continue to run (see also **Chapter 15**).

*Note: SYSCON0 is protected by the register security mechanism (see **Section 6.3.5**). The reset value is only valid for a hardware reset.*

Register SYSCON1 selects the following functions:

- Master clock prescaler factor for the system
- Program Flash behavior in Idle/Sleep mode
- Port driver behavior during Sleep mode and Powerdown mode
- Selection of Idle mode or Sleep mode

SYSCON1
System Control Reg. 1 **ESFR ($F1DC_H/EE_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	CP SYS	-	-	PF CFG		PD CFG		SLEEP CON	
-	-	-	-	-	-	-	rw	-	-	rw		rw		rw	

Field	Bits	Type	Description
CPSYS	8	rw	**Clock Prescaler for System** (see **Section 6.2.2**) The clock signal for the CPU is prescaled: 0 $f_{CPU} = f_{MC}$ 1 $f_{CPU} = f_{MC}/2$
PFCFG	[5:4]	rw	**Program Flash Configuration**[1)] 00 Program Flash is always ON (default) 01 Program Flash is off in IDLE or Sleep mode 10 Reserved 11 Reserved
PDCFG	[3:2]	rw	**Port Driver Configuration** 00 Port drivers are always ON (default) 01 Port drivers are off in IDLE or Sleep mode 10 Port drivers are off in Powerdown mode 11 Reserved
SLEEPCON	[1:0]	rw	**SLEEP Mode Configuration** (mode entered upon the IDLE instruction) 00 Enter normal IDLE mode 01 Enter SLEEP mode 10 Reserved 11 Reserved

1) In Powerdown mode the Program Flash will always be off.

*Note: SYSCON1 is protected by the register security mechanism (see **Section 6.3.5**).*

6.3.1 Status Indication

The system status register SYSSTAT indicates the status of the clock generation unit and the recent reset with a number of flags.

SYSSTAT
System Status Register **mem ($F1E4_H$/--)** **Reset Value: $XXXX_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OSCLOCK	PLLLOCK	CLKHIX	CLKLOX	OSCSTAB	PLLEM	-	-	-	-	-	-	-	HWR	SWR	WDTR
rh	rh	rh	rh	rh	rh	-	-	-	-	-	-	-	rh	rh	rh

Field	Bits	Type	Description
OSCLOCK	15	rh	**Oscillator Signal Status Bit** 0 The oscillator is unlocked 1 The oscillator is locked (2048 f_{OSC} periods have been counted, so it is assumed as stable)
PLLLOCK	14	rh	**PLL Signal Status Bit** 0 PLL unlocked (base frequency or adjusting) 1 The PLL is locked (stable output frequency)
CLKHIX	13	rh	**Input Clock High Limit Exceeded** 0 The input clock frequency is below the upper limit of the monitored range 1 The input clock frequency is too high
CLKLOX	12	rh	**Input Clock Low Limit Exceeded** 0 The input clock frequency is above the lower limit of the monitored range 1 The input clock frequency is too low
OSCSTAB	11	rh	**Oscillator Stable Flag** 0 The oscillator is starting up 1 The oscillator counter has reached its upper threshold (2^{15} f_{OSC} periods). With default gain, this indicates that the oscillator has reached 90% of its maximum amplitude. OSCSTAB is cleared upon a hardware reset or after a wakeup trigger when the oscillator was off.

Field	Bits	Type	Description
PLLEM	10	rh	**PLL Emergency Mode Flag** 0 No clock generation problem encountered 1 A clock generation problem has occurred *Note: PLLEM is cleared automatically if the oscillator has locked after a wake-up from sleep mode. Otherwise it remains set until hardware reset.*
HWR	2	rh	**Hardware Reset Indication Flag** 0 Last reset was no hardware reset 1 Last reset was a hardware reset
SWR	1	rh	**Software Reset Indication Flag** 0 Last reset was no software reset 1 Last reset was a software reset
WDTR	0	rh	**Watchdog Timer Reset Indication Flag** 0 Last reset was no watchdog timer reset 1 Last reset was a watchdog timer reset

Note: The reset value of register SYSSTAT depends on the active status flags.

6.3.2 Reset Source Indication

Reset indication flags in register SYSSTAT provide information about the source of the last reset. After the XC164CM starts execution, the initialization software may check these flags to determine if the recent reset event was triggered by an external hardware signal (via $\overline{\text{RSTIN}}$), by software, or by an overflow of the watchdog timer. The initialization and further operation of the microcontroller system can thus be adapted to the respective circumstances. For instance, a special routine may verify software integrity after a watchdog timer reset.

The reset indication flags are mutually exclusive; only one flag is set after reset depending on its source.

Hardware Reset is indicated when the $\overline{\text{RSTIN}}$ input is sampled low (active).

Software Reset is indicated after a reset triggered by the execution of instruction SRST.

Watchdog Timer Reset is indicated after a reset triggered by an overflow of the watchdog timer.

6.3.3 Peripheral Shutdown Handshake

When executing a software reset or when entering Powerdown mode the SCU requests a shutdown from those peripheral units that are currently active and provide the shutdown handshake mechanism. Upon this request the respective peripheral unit completes the currently active action (if any) and then acknowledges the shutdown request to the SCU.

These units are: PMU, DMU, EBC, ADC, and Program Flash.

The shutdown handshake sequence is completed as soon as all units have acknowledged the shutdown request.

6.3.4 Debug System Control

The debug and emulation circuitry is controlled by two registers:

- The Emulation Control register EMUCON controls basic OCDS functions.
- The OCE/OCDS Peripheral Suspend Enable register OPSEN selects the peripherals that will be halted by the suspend signal. OPSEN is structured identically to register SYSCON3.

EMUCON
Emulation Control Reg. **SFR ($FE0A_H/05_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	-	OCEN	OCDSIOEN	-
-	-	-	-	-	-	-	-	-	-	-	-	-	rw	rw	-

Field	Bits	Type	Description
OCEN	2	rw	**OCDS/Cerberus Enable** 0 OCDS and Cerberus are still in reset state 1 ODCS and Cerberus are operable
OCDSIOEN	1	rw	**OCDS Break Input/Output Enable** 0 OCDS break input/output $\overline{BRKIN}/\overline{BRKOUT}$ are disabled. 1 OCDS break input/output $\overline{BRKIN}/\overline{BRKOUT}$ are enabled.

*Note: EMUCON is write protected after the execution of EINIT by the register security mechanism (see **Section 6.3.5**).*

OPSEN
OCE/OCDS P-Susp. En. Reg. ESFR ($FE58_H/2C_H$) Reset Value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SSC 1 SEN	RTC SEN	CAN SEN	-	-	ASC 1 SEN	-	CC6 SEN	CC2 SEN	-	PFM SEN	-	GPT SEN	SSC 0 SEN	ASC 0 SEN	ADC SEN
rw	rw	rw	rw	-	-	-	rw	rw	-	rw	-	rw	rw	rw	rw

Field	Bits	Type	Description
xxSEN	15 … 13, 10, 8, 7, 5, 3 … 0	rw	**Module xx Suspend Enable** 0 Respective module remains active during suspend 1 Respective module halts operation during suspend

*Note: OPSEN is write protected after the execution of EINIT by the register security mechanism (see **Section 6.3.5**).*

When a breakpoint is hit, the on-chip peripherals selected in register OPSEN are stopped and placed in power-down mode the same way as if disabled via register SYSCON3. Registers of peripherals which are stopped this way can be read, but not written. A read access will not trigger any actions with a disabled peripheral.

The SYSCON3 bits return the shutdown status independently of the reason for the shutdown (static shutdown via SYSCON 3 or intermediate shutdown via OPSEN). That means that when SYSCON3 is read via the debugger after a breakpoint has been hit, it returns the contents of SYSCON3 OR-ed bit-wise with the contents of OPSEN.

It is recommended to leave bit OPSEN.5 (PFMSEN) at default value ’0’. Otherwise, the program flash is deactivated when a breakpoint is hit (i.e. it can not be read), and it has to rump up when program execution is resumed (i.e. synchronization between software and peripherals is lost).

6.3.5 Register Security Mechanism

There are some dedicated registers which control critical functions and modes. These registers are protected by a special register security mechanism so these vital system functions cannot be changed inadvertently.

This security mechanism controls three different security levels:

- **Write Protected Mode** (entered after the execution of EINIT)
  Protected registers are locked against any write access (read only).
- **Secured Mode**
  Protected registers can be written using a special command sequence.
- **Unprotected Mode** (entered after reset)
  No protection is active. Registers can be written at any time.

*Note: The selected security level applies to all protected registers throughout the XC164CM (see **Table 6-8**).*

Controlling the Security Level

Two registers build the interface for controlling the security level. The security level command register SCUSLC accepts the commands to control the state machine modifying the security level (the required command sequence is safeguarded with a password). The security level status register SCUSLS (read only) shows the actual password, the actual security level, and the state of the switching state machine.

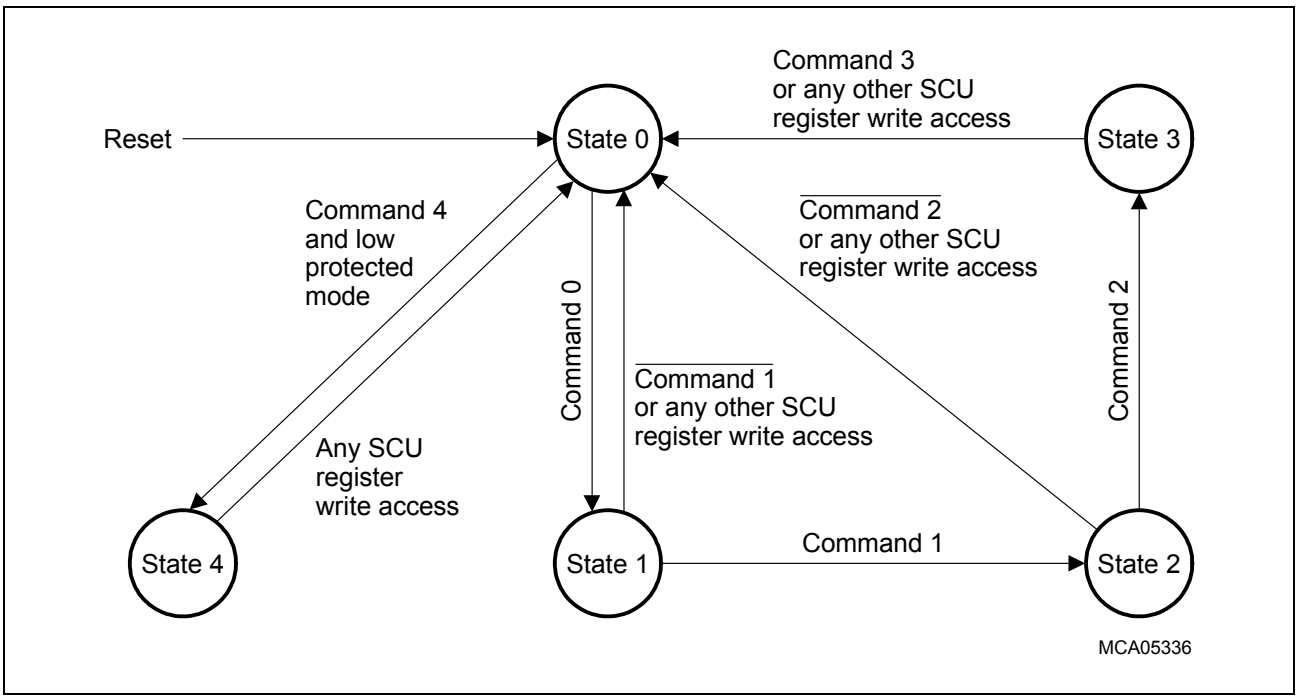


Figure 6-10 State Machine for Security Level Switching

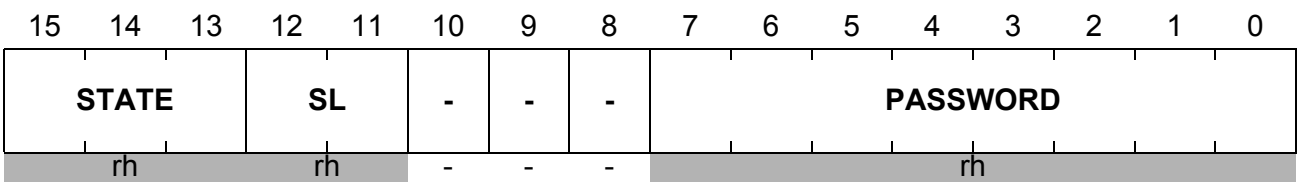
Two mechanisms can be used to control the actual security level:

- **Changing the security level**
  can be done by executing the following command sequence:
  “command0-command1-command2-command3”.
  This sequence establishes a new security level and/or a new password.
- **Access in secured mode**
  can be achieved by preceding the intended write access with writing “command4” to register SCUSLC. This quick access is only possible while secured mode is selected.

Read accesses are always possible to all registers of the SCU and will not influence the command sequences. In register SCUSLS the actual status of the command state machine can always be read.

Note: After writing command4 in secured mode the lock mechanism remains disabled until the next write access to an SCU register or a register on the PD bus, i.e. accesses to registers outside this area do not re-activate the protection.
Default after reset is the unprotected state. This state can be changed via command sequence only after execution of the EINIT instruction.

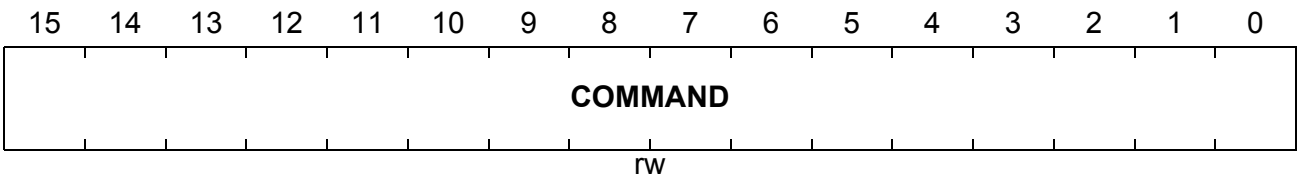
SCUSLS
Sec. Level Status Reg. **ESFR ($F0C2_H/61_H$)** **Reset Value: 0000_H**



Field	Bits	Type	Description
STATE	[15:13]	rh	**Current State of Switching State Machine** 000 Awaiting command0 or command4 (default) 001 Awaiting command1 010 Awaiting command2 011 Awaiting new security level and password 100 Next access granted in secured mode 101 Reserved 11X Reserved
SL	[12:11]	rh	**Security Level** 00 Unprotected mode (default after reset) 01 Secured mode 10 Reserved 11 Write protected mode (entered after EINIT)

Field	Bits	Type	Description
PASSWORD	[7:0]	rh	**Current Security Control Password** Default after reset = 00_H

SCUSLC
Sec. Level Command Reg. **ESFR ($F0C0_H/60_H$)** **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 6-7**)

Note: Register SCUSLC is not protected by the security mechanism. This is required to be able to change the security level in any state.

Table 6-7 Commands for Security Level Control

Command	Definition	Note
Command0	$AAAA_H$	–
Command1	5554_H	–
Command2	96_H || <inverse password>	–
Command3	000_B || <new level> || 000_B || <new password>	–
Command4	$8E_H$ || <inverse password>	Secured mode only

Note: It is recommended to lock all command sequences with an atomic sequence.

Programming Examples

```
EXTR #4                 ;Sequence to change the security level
MOV  SCUSLC, #0AAAAH    ;Command0
MOV  SCUSLC, #05554H    ;Command1
MOV  SCUSLC, #096FFH    ;Command2: current password = 00H
MOV  SCUSLC, #008EDH    ;Command3: level = 01, new password = EDH

EXTR #1                 ;Access sequence in secured mode
MOV  SCUSLC, #8E12H     ;Command4: current password = EDH
MOV  register, data     ;Access enabled by the preceding Command4
```

The Register Security Mechanism protects not only the SCU registers but also a number of registers in other modules. **Table 6-8** summarizes these registers.

Table 6-8 Registers Protected by the Security Mechanism

Register Name	Function
RSTCON	Reset control
SYSCON0	General system control
SYSCON1	Power management
PLLCON	Clock generation control
SYSCON3	Peripheral management
FOCON	Peripheral management (CLKOUT/FOUT)
IMBCTR	Control of internal instruction memory block
OPSEN	Emulation control
EMUCON	Emulation control
WDTCON	Watchdog timer properties
EXICON	Ext. interrupt control
EXISEL0, EXISEL1	Ext. interrupt control
CPUCON1, CPUCON2	CPU configuration, protected after EINIT
TCONCS7	EBC timing configuration (internal TwinCAN access)
FCONCS7	EBC function configuration (internal TwinCAN access)
ADDRSEL7	EBC address window configuration (internal TwinCAN access)

6.4 Power Management

The power consumption of the XC164CM can be reduced by several mechanisms. The level of power reduction depends on the level of system performance that is required

under these circumstances. The architecture of the XC164CM provides three major means of reducing its power consumption under software control:

- Power reduction modes (Idle, Sleep, Powerdown) to deactivate CPU, ports and control logic
- Reduction of the CPU frequency and system frequency
- Selection of the active peripheral modules (Flexible Peripheral Management)

This enables the application (that is, the programmer) to choose the optimum constellation for each operating condition, so the power consumption can be adapted to conditions like maximum performance, partial performance, or standby.

These three means can be applied independent from each other and thus provide a maximum of flexibility for each application.

6.4.1 Power Reduction Modes

Three different general power reduction modes with different levels of power reduction have been implemented in the XC164CM, which are selected by dedicated instructions:

IDLE selects Idle Mode or Sleep Mode, PWRDN selects Powerdown Mode.

Attention: Upon a reset all power reduction modes are terminated.

Idle Mode

In Idle Mode all enabled peripherals, **including** the watchdog timer, continue to operate normally, only the CPU operation is halted.

Note: Peripherals that have been disabled via software also remain disabled in Idle mode, of course.

Idle Mode is entered after the IDLE instruction has been executed (bitfield SLEEPCON in register SYSCON1 must be 00_B) and the instruction before the IDLE instruction has been completed. To prevent unintentional entry into Idle Mode, the IDLE instruction has been implemented as a protected 32-bit instruction.

Idle Mode is terminated by interrupt requests from any enabled interrupt source whose individual interrupt enable flag was set before the Idle Mode was entered, regardless of bit IEN.

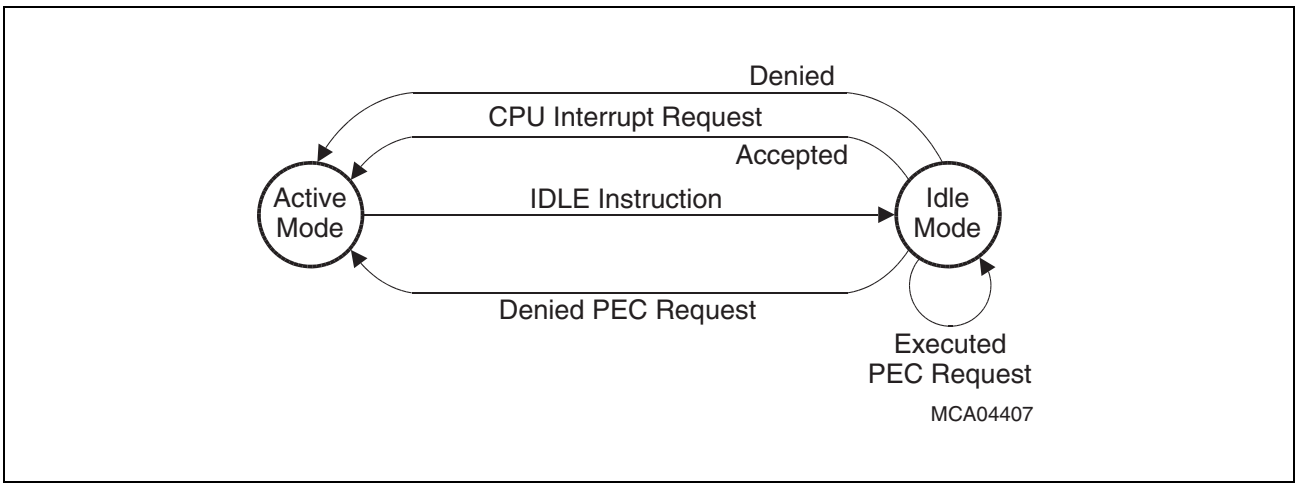


Figure 6-11 Transitions between Idle Mode and Active Mode

For a request selected for CPU interrupt service the associated interrupt service routine is entered if the priority level of the requesting source is higher than the current CPU priority and the interrupt system is globally enabled. After the RETI (Return from Interrupt) instruction of the interrupt service routine is executed the CPU continues executing the program with the instruction following the IDLE instruction. Otherwise, if the interrupt request cannot be serviced because of a too low priority or a globally disabled interrupt system the CPU immediately resumes normal program execution with the instruction following the IDLE instruction.

For a request which was programmed for PEC service a PEC data transfer is performed if the priority level of this request is higher than the current CPU priority and the interrupt system is globally enabled. After the PEC data transfer has been completed the CPU remains in Idle Mode. Otherwise, if the PEC request cannot be serviced because of a too low priority or a globally disabled interrupt system the CPU does not remain in Idle Mode but continues program execution with the instruction following the IDLE instruction.

Idle Mode can also be terminated by a Non Maskable Interrupt, i.e. a high-to-low transition on the $\overline{\text{NMI}}$ pin. After Idle Mode has been terminated by an interrupt or NMI request, the interrupt arbitration block performs a round of prioritization to determine the highest priority request. In the case of an NMI request, the NMI trap will always be entered.

Any interrupt request whose individual interrupt enable flag was set before Idle Mode was entered will terminate Idle Mode regardless of the current CPU priority. The CPU will **not** go back into Idle Mode when a CPU interrupt request is detected, even when the interrupt was not serviced because of a higher CPU priority or a globally disabled interrupt system (IEN = 0). The CPU will **only** go back into Idle Mode when the interrupt system is globally enabled (IEN = 1) **and** a PEC service on a priority level higher than the current CPU level is requested and executed.

Note: An interrupt request which is individually enabled and assigned to priority level 0 will terminate Idle Mode. The associated interrupt vector will not be accessed, however.

The watchdog timer may be used to monitor the Idle Mode: an internal watchdog timer reset will be generated if no interrupt or NMI request occurs before the watchdog timer overflows. To prevent the watchdog timer from overflowing during Idle Mode it must be programmed to a reasonable time interval before Idle Mode is entered.

Sleep Mode

In Sleep Mode clocking of all internal blocks (including WDT) is stopped. The contents of the internal RAM, however, are preserved through the voltage supplied via the V_{DD} pins.

Sleep Mode is entered after the IDLE instruction has been executed (bitfield SLEEPCON in register SYSCON1 must be 01_B), the instruction before the IDLE instruction has been completed, and all individual enabled interrupt request flags are inactive. To prevent unintentional entry into Sleep Mode, the IDLE instruction has been implemented as a protected 32-bit instruction.

Sleep Mode is terminated by an RTC interrupt (if enabled), by a transition on the external interrupt line or the respective alternate input line, when the respective level detection is enabled, and by an NMI request. The external interrupt level detection is controlled by register EXICON. The action, which is taken after the wake-up, depends on the individual interrupt enable flag's setting of the fast external interrupt, which has triggered the wake-up, and on the setting of the global interrupt enable flag PSW.IEN. The XC164CM switches from Sleep Mode to Idle Mode if the individual interrupt enable flag was not set by software before the wake-up has been triggered. Otherwise, the XC164CM switches to Active Mode if the enable flag is 1. Whether a branch to the interrupt service routine or to the instruction following the IDLE is executed, depends on the setting of the global interrupt enable flag and on the interrupt level. This behavior is identical to the Idle Mode.

Note: The receive lines of serial interfaces may be internally routed to external interrupt inputs via registers EXISELn. All peripherals are stopped and hence cannot generate an interrupt request.

The total power consumption in Sleep Mode depends mainly on the current that flows through the port drivers. To minimize the consumed current all pin drivers can be disabled (pins switched to tristate) via a central control bit in register SYSCON1. If an application requires one or more port drivers to remain active even in Sleep Mode also individual port drivers can be disabled simply by configuring them for input.

Powerdown Mode

In Powerdown Mode both the CPU and all peripherals are stopped. The contents of the internal RAM, however, are preserved through the voltage supplied via the V_{DD} pins.

Powerdown Mode is entered after the PWRDN instruction has been executed, the instruction before the PWRDN instruction has been completed, and the NMI (Non Maskable Interrupt) pin is externally pulled low while the PWRDN instruction is executed. To prevent unintentional entry into Powerdown Mode, the PWRDN instruction has been implemented as a protected 32-bit instruction and is additionally validated by the NMI signal. If pin NMI is not low at this time, Powerdown Mode will not be entered and program execution continues.

Powerdown Mode is terminated only by a hardware reset (an internal reset cannot occur).

The total power consumption in Powerdown Mode depends mainly on the current that flows through the port drivers. To minimize the consumed current all pin drivers can be disabled (pins switched to tristate) via a central control bit in register SYSCON1. If an application requires one or more port drivers to remain active even in Powerdown Mode also individual port drivers can be disabled simply by configuring them for input.

6.4.2 Reduction of Clock Frequencies

The power consumption of the XC164CM is linearly dependent on the logic's switching frequency. The means to control this are described in **Section 6.2**, **Clock Generation**.

6.4.3 Flexible Peripheral Management

The power consumed by the XC164CM also depends on the amount of active logic. Peripheral management deactivates those on-chip peripherals that are not required in a given system status (e.g. a certain interface mode or standby). This reduces the amount of clocked circuitry. All modules that remain active, however, will still deliver their usual performance.

Note: A read access to a register of a disabled peripheral returns the valid register content, whereas a write access to this register is ignored.
A read access will not trigger any actions within a disabled peripheral.

While a peripheral is disabled, its associated output pins remain in the state they had at the time of disabling.

Software controls this flexible peripheral management via register SYSCON3 where each control bit is associated with an on-chip peripheral module.

Writing SYSCON3 requests deactivation/activation of the respective peripheral(s). When writing to register SYSCON3, make sure that all undefined bits are written with 1 s.

Reading SYSCON3 returns the peripherals' actual status according to the shutdown handshake mechanism (see **Section 6.3.3**).

SYSCON3
System Control Reg. 3 **ESFR ($F1D4_H/EA_H$)** **Reset Value: $9FD0_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SSC 1 DIS	RTC DIS	CAN DIS	-	-	ASC 1 DIS	-	CC6 DIS	CC2 DIS	-	PFM DIS	-	GPT DIS	SSC 0 DIS	ASC 0 DIS	ADC DIS
rw	rw	rw	-	-	rw	-	rw	rw	-	rw	-	rw	rw	rw	rw

Field	Bits	Type	Description
SSC1DIS	15	rw	**Synchronous Serial Channel SSC1**
RTCDIS	14	rw	**Real Time Clock**
CANDIS	13	rw	**On-chip CAN Module**
ASC1DIS	10	rw	**USART ASC1**
CC6DIS	8	rw	**CAPCOM Unit 6**
CC2DIS	7	rw	**CAPCOM Unit 2**
PFMDIS	5	rw	**Program Flash Module**[1)]
GPTDIS	3	rw	**General Purpose Timer Blocks**
SSC0DIS	2	rw	**Synchronous Serial Channel SSC0**
ASC0DIS	1	rw	**USART ASC0**
ADCDIS	0	rw	**Analog/Digital Converter**

1) When the program flash module is deactivated in active mode (by setting bit PFMDIS), the next access to the program flash module will be answered with the trap code $1E9B_H$ and produce a “Program Memory Access Error” trap.

With the reset value $9FD0_H$, the program memory and a basic set of peripherals is enabled. The other peripherals of the XC164CM must be enabled during initialization before they can be used.

For some derivatives of the XC164CM the reset value $9FD0_H$ seems to select peripherals not available in the respective derivative (see **Table 1-1**). To ensure a robust software concept, it is recommended to write a valid value to register SYSCON3 during the initialization phase.

Note: The allocation of peripheral disable bits within register SYSCON3 is device specific and may be different in other derivatives than the XC164CM.
*SYSCON3 is write protected after the execution of EINIT by the register security mechanism (see **Section 6.3.5**).*

6.5 Watchdog Timer (WDT)

To allow recovery from software or hardware failure, the XC164CM provides a Watchdog Timer. If the software fails to service this timer before an overflow occurs, an internal reset sequence will be initiated. If the watchdog timer is enabled and the software has been designed to service it regularly before it overflows, the watchdog timer will supervise the program execution so it will overflow only if the program does not progress properly. The watchdog timer will also time out if a software error was caused by hardware related failures. This prevents the controller from malfunctioning for a time longer than specified by the user.

The watchdog timer provides two registers:

- a read-only timer register containing the current count, and
- a control register for initialization and reset source detection.

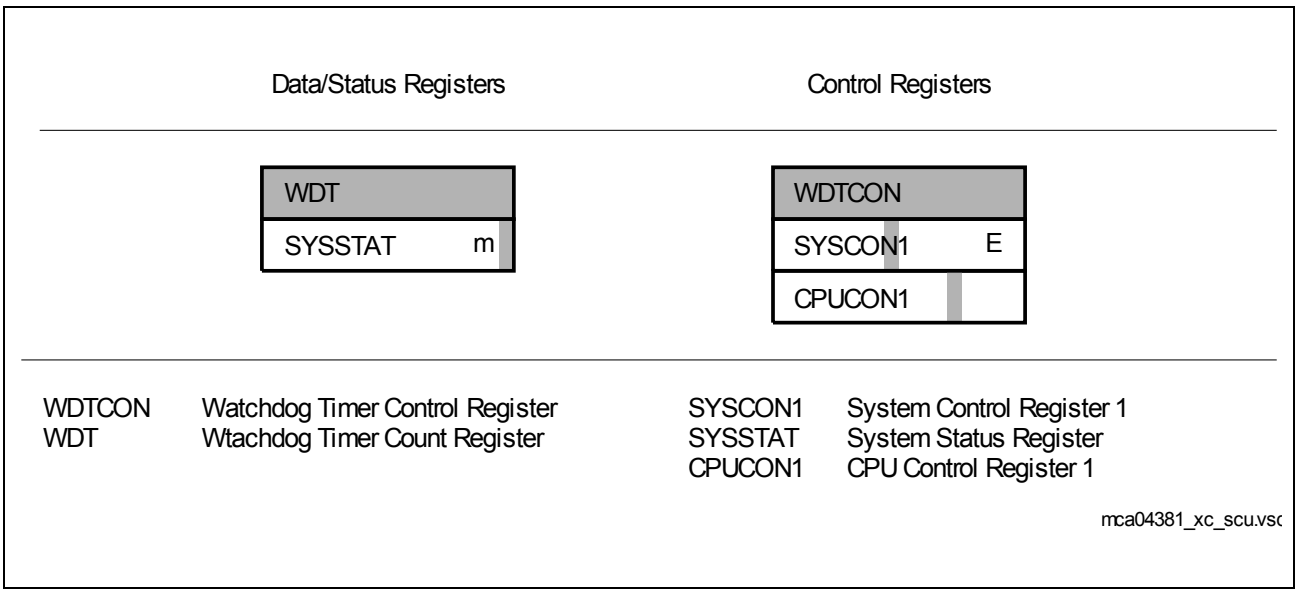


Figure 6-12 SFRs and Port Pins Associated with the Watchdog Timer

The watchdog timer is a 16-bit up counter which is clocked with the prescaled system clock (f_{SYS}). The prescaler divides the system clock:

- by 2 (WDTIN = 00_B), or
- by 4 (WDTIN = 10_B), or
- by 128 (WDTIN = 01_B), or
- by 256 (WDTIN = 11_B).

The 16-bit watchdog timer is implemented as two concatenated 8-bit timers (see **Figure 6-13**). The upper 8 bits of the watchdog timer can be preset to a user-programmable value via a watchdog service access in order to vary the watchdog expire time. The lower 8 bits are reset after each service access.

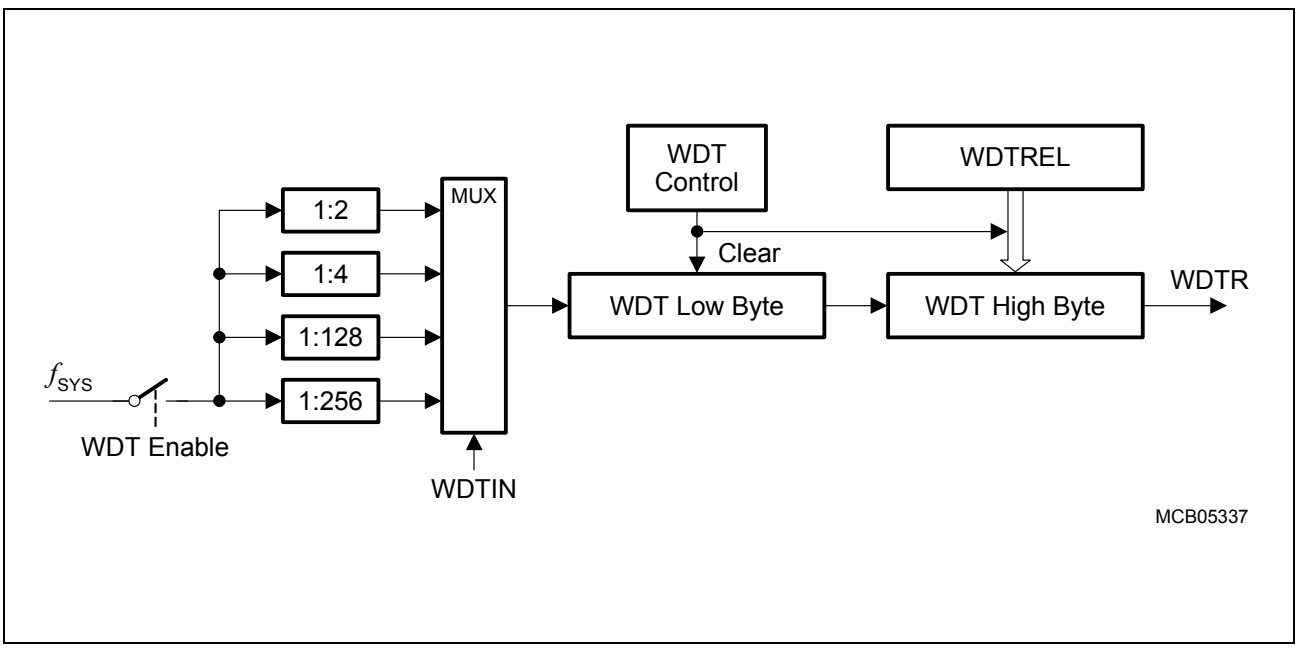


Figure 6-13 Watchdog Timer Block Diagram

Operation of the Watchdog Timer

The current count value of the Watchdog Timer is contained in the Watchdog Timer Register WDT which is a non-bitaddressable read-only register. Operation of the Watchdog Timer is controlled by its bitaddressable Watchdog Timer Control Register WDTCON. This register specifies the reload value for the high byte of the timer, and selects the input clock prescaling factor.

After any reset (except as noted) the watchdog timer is enabled and starts counting up from 0000_H with the default frequency $f_{WDT} = f_{SYS}/2$. The default input frequency may be changed to another frequency ($f_{WDT} = f_{SYS}/4$, 128, 256) by programming the prescaler (bitfield WDTIN).

The watchdog timer can be disabled by executing the instruction DISWDT (Disable Watchdog Timer). Instruction DISWDT is a protected 32-bit instruction.

In compatible WDT mode instruction DISWDT will ONLY be executed during the time between a reset and execution of either the EINIT or the SRVWDT instruction. Either one of these instructions disables the execution of DISWDT. Once disabled, the WDT can only be enabled by a reset.

In enhanced WDT mode the watchdog timer can be disabled and enabled at any time (independent of the EINIT instruction). This is controlled by executing instructions DISWDT and ENWDT, respectively. Instruction ENWDT is a protected 32-bit instruction. If the watchdog timer is re-enabled via instruction ENWDT it is implicitly serviced.

The basic control mode (compatible/enhanced) is selected by bit WDTCTL in register CPUCON1.

Note: After a hardware reset that activates the Bootstrap Loader the watchdog timer will be disabled. The WDT is enabled, when the loaded software begins executing.

When the watchdog timer is not disabled via instruction DISWDT it will continue counting up, even in Idle Mode. If it is not serviced by the time the count reaches $FFFF_H$ the watchdog timer will overflow and cause an internal reset. The Watchdog Timer Reset Indication Flag (WDTR) in register SYSSTAT will be set in this case.

Attention: A watchdog timer reset is unconditional. All current data/code accesses are aborted.

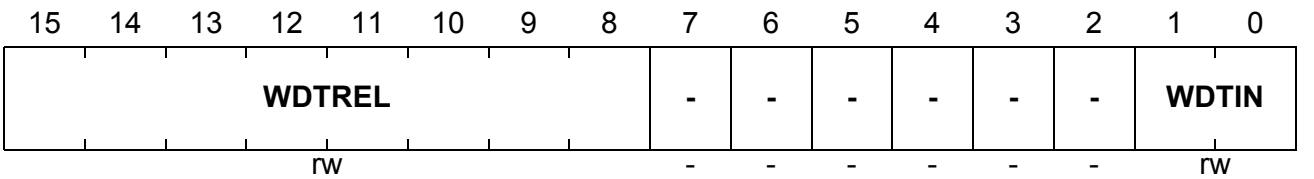
To prevent the watchdog timer from overflowing, it must be serviced periodically by the user software. The watchdog timer is serviced by three different actions:

- by executing instruction SRVWDT which is a protected 32-bit instruction
- by writing to register WDTCON
- by executing instruction ENWDT (in enhanced WDT mode)

Servicing the watchdog timer clears the low byte and reloads the high byte of the watchdog timer register WDT with the preset value from bitfield WDTREL which is the high byte of register WDTCON. After servicing, the watchdog timer resumes counting up from the value (<WDTREL> $\times$ 2^8).

Instruction SRVWDT has been encoded in such a way that the chance of unintentionally servicing the watchdog timer is minimized (such as by fetching and executing a bit pattern from a wrong location). When instruction SRVWDT does not match the format for protected instructions, the Protection Fault Trap will be entered, rather than executing the instruction.

WDTCON
WDT Control Register **SFR ($FFAE_H/D7_H$)** **Reset Value: 0000_H**



Field	Bits	Type	Description
WDTREL	[15:8]	rw	**Watchdog Timer Reload Value** (for the high byte of WDT)
WDTIN	[1:0]	rw	**Watchdog Timer Input Frequency Select** 00 $f_{WDT} = f_{SYS}/2$ 01 $f_{WDT} = f_{SYS}/128$ 10 $f_{WDT} = f_{SYS}/4$ 11 $f_{WDT} = f_{SYS}/256$

*Note: WDTCON is protected by the register security mechanism (see **Section 6.3.5**).*

The time period for an overflow of the watchdog timer is programmable in two ways:

- **Input frequency** to the watchdog timer can be selected via a prescaler controlled by bitfield WDTIN in register WDTCON to be $f_{SYS}/2, f_{SYS}/4, f_{SYS}/128$ or $f_{SYS}/256$.
- **Reload value** WDTREL for the high byte of WDT can be programmed in register WDTCON.

The period P_{WDT} between servicing the watchdog timer and the next overflow can therefore be determined by the following formula:

$$P_{WDT} = \frac{2^{(1 + <WDTIN.1> + <WDTIN.0> \times 6)} \times (2^{16} - <WDTREL> \times 2^{8})}{f_{SYS}} \quad (6.2)$$

Table 6-9 lists the possible ranges (depending on the prescaler bitfield WDTIN) for the watchdog time which can be achieved using a certain system clock.

Table 6-9 Watchdog Time Ranges

System Clock f_{SYS}	Prescaler		Reload Value in WDTREL		
	WDTIN	f_{WDT}	**FF_H**	**$7F_H$**	**00_H**
10 MHz	00_B	f_{SYS} / 2	51.20 µs	6.61 ms	13.11 ms
	10_B	f_{SYS} / 4	102.4 µs	13.21 ms	26.21 ms
	01_B	f_{SYS} / 128	3.28 ms	422.7 ms	838.9 ms
	11_B	f_{SYS} / 256	6.55 ms	845.4 ms	1678 ms
20 MHz	00_B	f_{SYS} / 2	25.60 µs	3.30 ms	6.55 ms
	10_B	f_{SYS} / 4	51.20 µs	6.61 ms	13.11 ms
	01_B	f_{SYS} / 128	1.64 ms	211.4 ms	419.4 ms
	11_B	f_{SYS} / 256	3.28 ms	422.7 ms	838.9 ms
30 MHz	00_B	f_{SYS} / 2	17.07 µs	2.20 ms	4.37 ms
	10_B	f_{SYS} / 4	34.13 µs	4.40 ms	8.74 ms
	01_B	f_{SYS} / 128	1.09 ms	140.1 ms	279.6 ms
	11_B	f_{SYS} / 256	2.19 ms	281.8 ms	559.2 ms
40 MHz	00_B	f_{SYS} / 2	12.80 µs	1.65 ms	3.28 ms
	10_B	f_{SYS} / 4	25.60 µs	3.30 ms	6.55 ms
	01_B	f_{SYS} / 128	0.82 ms	105.7 ms	209.7 ms
	11_B	f_{SYS} / 256	1.64 ms	211.4 ms	419.4 ms

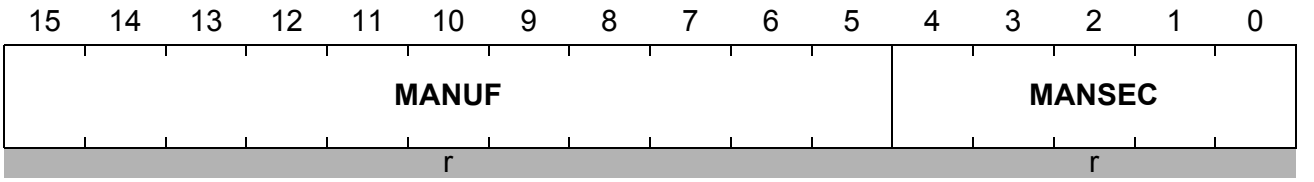
Note: The user is advised to rewrite WDTCON each time before the watchdog timer is serviced, particularly when the register security mechanism is disabled or when the software concept uses alternating watchdog periods.

6.6 Identification Control 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. These ID registers can be used for automatic test selection as well as for identification of unknown silicon.

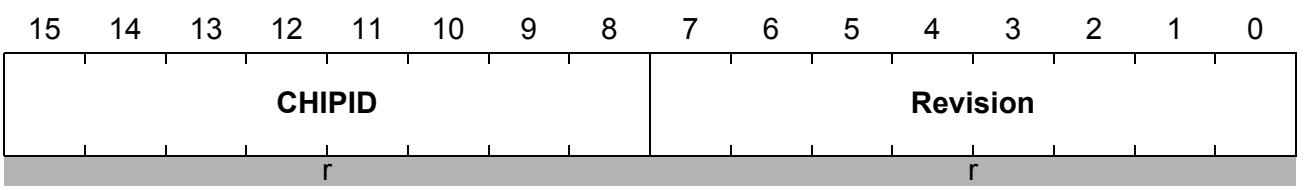
Note: The not defined locations within the area 00'F070$_H$ … 00'F07E$_H$ are reserved for future identification features.

IDMANUF
Manufacturer Ident. Reg. **ESFR (F07E$_H$/3F$_H$)** **Reset Value: 1820$_H$**



Field	Bits	Type	Description
MANUF	[15:5]	r	**Manufacturer** This is the JEDEC normalized manufacturer code. 0C1$_H$ Infineon Technologies AG
MANSEC	[4:0]	r	**Section within Manufacturer** 00$_H$ Standard microcontroller

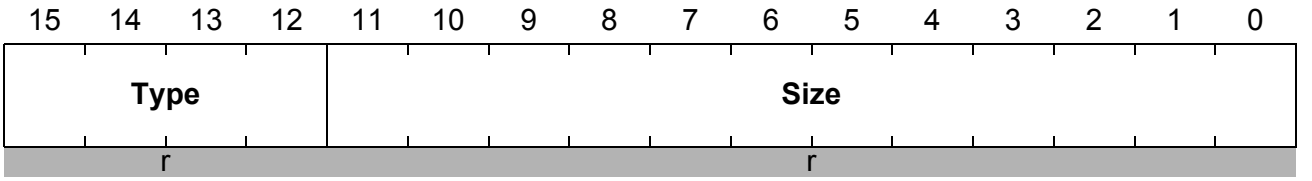
IDCHIP
Chip Identification Reg. **ESFR (F07C$_H$/3E$_H$)** **Reset Value: 23XX$_H$**



Field	Bits	Type	Description
CHIPID	[15:8]	r	**Device Identification** Identifies the device name (reference via table).
Revision	[7:0]	r	**Device Revision Code** Identifies the device step.

IDMEM

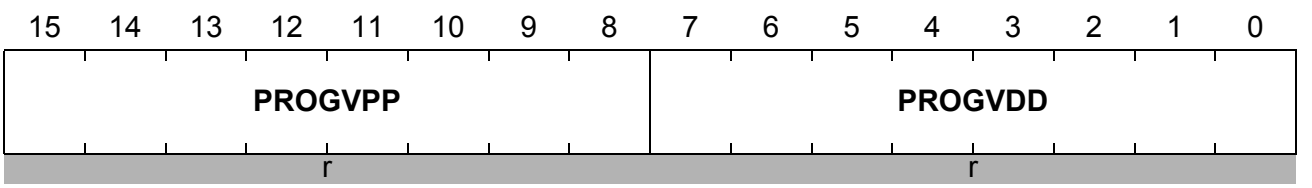
Program Mem. Ident. Reg. **ESFR ($F07A_H/3D_H$)** **Reset Value: 3010_H**



Field	Bits	Type	Description
Type	[15:12]	r	**Type of on-chip Program Memory** Identifies the memory type on this silicon. 0_H No on-chip program memory 1_H Mask-programmable ROM 2_H EEPROM memory 3_H Flash memory 4_H OTP memory
Size	[11:0]	r	**Size of on-chip Program Memory** The size of the program memory in terms of 4-Kbyte blocks, i.e. Mem-size = <Size> × 4 Kbytes.

IDPROG

Progr. Voltage Ident. Reg. **ESFR ($F078_H/3C_H$)** **Reset Value: 4040_H**



Field	Bits	Type	Description
PROGVPP	[15:8]	rw	**Programming V_{PP} Voltage**[1)] The voltage of the special programming power supply required to program or erase (if applicable) the on-chip program memory. Formula: V_{PP} = 20 × <PROGVPP> / 256 [V][2)]
PROGVDD	[7:0]	r	**Programming V_{DD} Voltage**[1)] The voltage of the standard power supply required to program or erase the on-chip program memory. Formula: V_{DD} = 20 × <PROGVDD> / 256 [V]

1) ROM derivatives are not programmable. Therefore, register IDPROG returns the value 0000_H.

2) The XC164CM needs no special programming voltage and PROGVPP = PROGVDD.

7 Parallel Ports

This chapter describes the implementation details of the Parallel Ports in XC164CM. This includes the definition of registers associated with each Port, the assignment and control of alternate functions to each port pin, and configuration diagrams for each port pin.

The XC164CM's IO lines are organized into nine input/output ports and one input port.

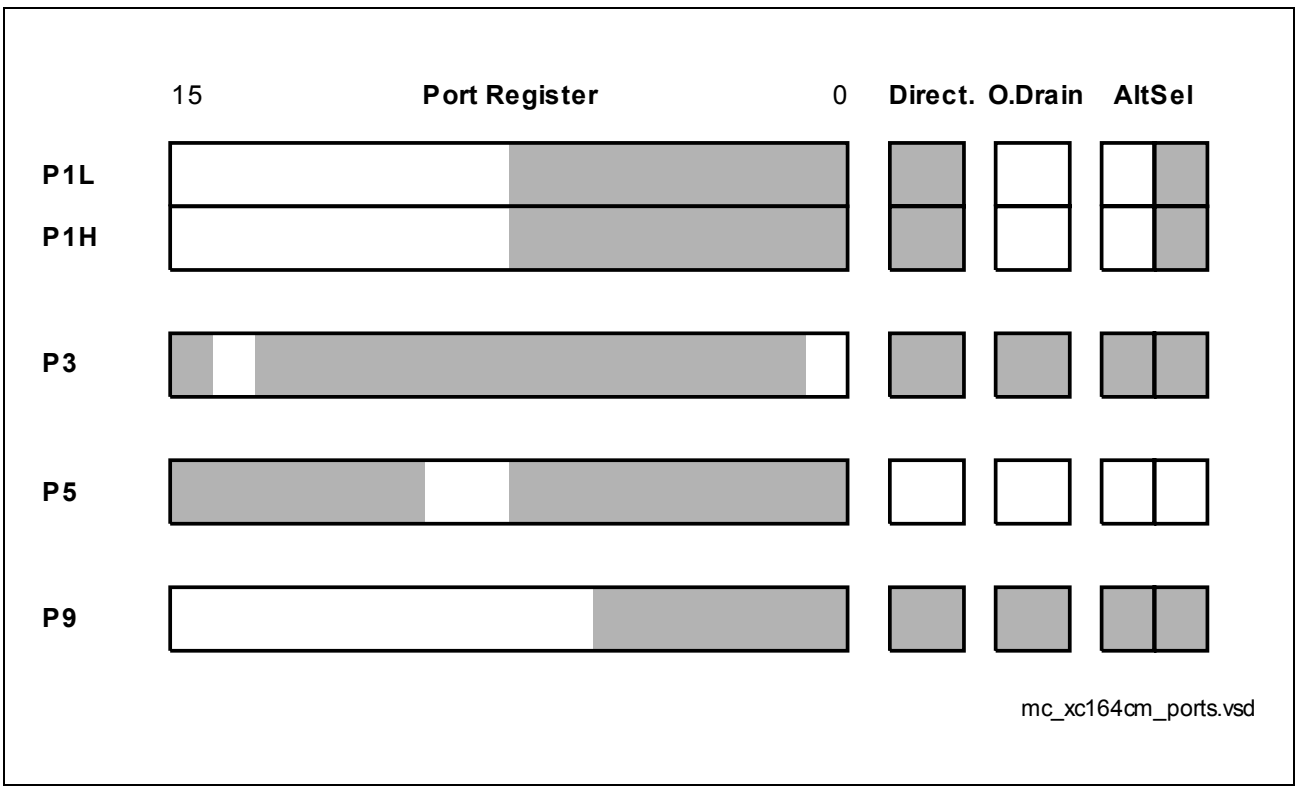


Figure 7-1 Port Overview of XC164CM

7.1 Input Threshold Control

The standard inputs of the XC164CM determine the status of input signals according to TTL levels. In order to accept and recognize noisy signals, CMOS-like input thresholds can be selected instead of the standard TTL thresholds for all pins of specific ports. These special thresholds are defined above the TTL thresholds and feature a defined hysteresis to prevent the inputs from toggling while the respective input signal level is near the thresholds.

The Port Input Control register PICON allows to select these thresholds for each byte of the indicated ports, i.e. 8-bit ports are controlled by one bit each while 16-bit ports are controlled by two bits each.

PICON
Port Input Control Register **ESFR ($F1C4_H/E2_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-						-	-	P9 LIN	-	-	-	P3 HIN	P3 LIN	-	-
-						-	-	rw	-	-	-	rw	rw	-	-

Field	Bits	Type	Description
PxHIN	3	rw	**Port x High Byte Input Level Selection** 0 Pins Px[15-8] switch on standard TTL input levels 1 Pins Px[15-8] switch on special threshold input levels
PxLIN	2, 7	rw	**Port x Low Byte Input Level Selection** 0 Pins Px[7-0] switch on standard TTL input levels 1 Pins Px[7-0] switch on special threshold input levels

All options for individual direction and output mode control are available for each pin independent from the selected input threshold. The input hysteresis provides stable inputs from noisy or slowly changing external signals.

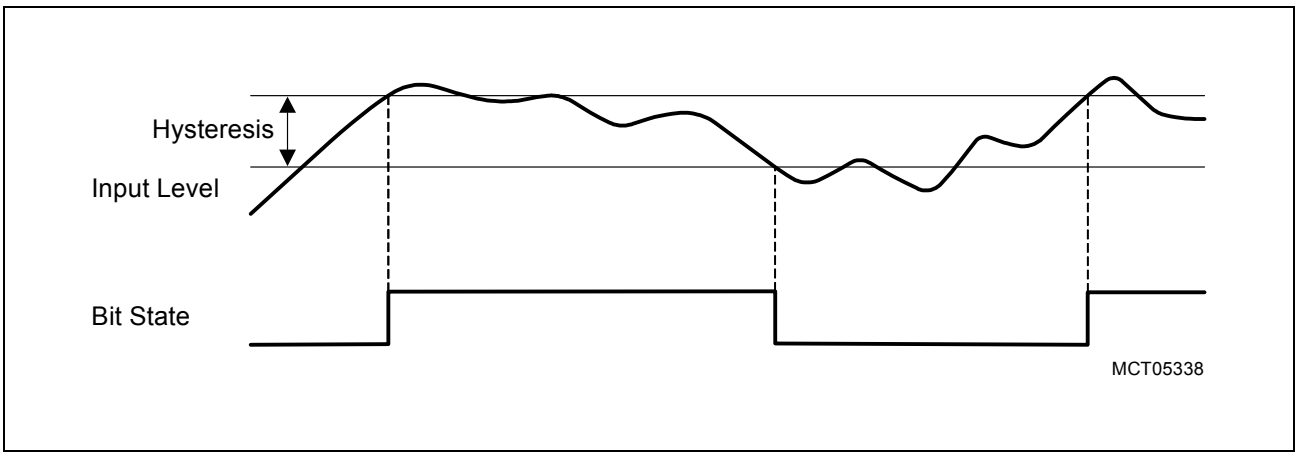


Figure 7-2 Hysteresis for Special Input Thresholds

7.2 Output Driver Control

The output driver of a port pin is activated by switching the respective pin to output, i.e. DPx.y = ‘1’. The value that is driven to the pin is determined by the port output latch or by the associated alternate function (e.g. address, peripheral IO, etc.). The user software can control the characteristics of the output driver via the following mechanisms:

- **Open Drain Mode:** The upper (push) transistor is always disabled. Only ‘0’ is driven actively, an external pull-up is required.
- **Driver Characteristic:** The driver strength can be selected.
- **Edge Characteristic:** The rise/fall time of an output signal can be selected.

Open Drain Mode

In the XC164CM certain ports provide Open Drain Control, which allows to switch the output driver of a port pin from a push/pull configuration to an open drain configuration. In push/pull mode a port output driver has an upper and a lower transistor, thus it can actively drive the line either to a high or a low level. In open drain mode the upper transistor is always switched off, and the output driver can only actively drive the line to a low level. When writing a ‘1’ to the port latch, the lower transistor is switched off and the output enters a high-impedance state. The high level must then be provided by an external pull-up device. With this feature, it is possible to connect several port pins together to a Wired-AND configuration, saving external glue logic and/or additional software overhead for enabling/disabling output signals.

This feature is controlled through the respective Open Drain Control Registers ODPx which are provided for each port that has this feature implemented. These registers allow the individual bit-wise selection of the open drain mode for each port line.

If the respective control bit ODPx.y is ‘0’ (default after reset), the output driver is in the push/pull mode. If ODPx.y is ‘1’, the open drain configuration is selected. Note that all ODPx registers are located in the ESFR space.

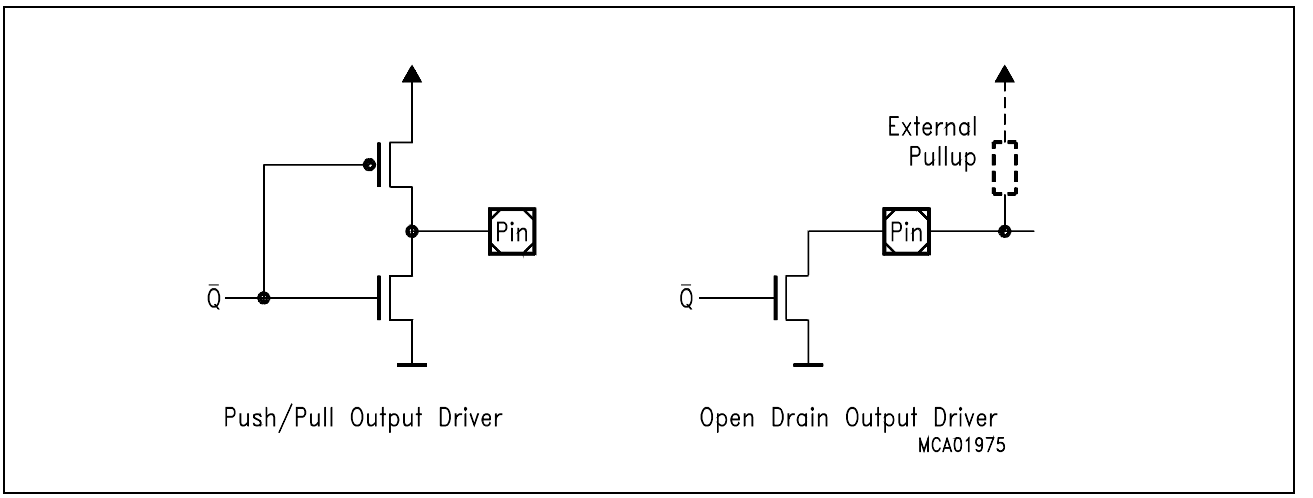


Figure 7-3 Output Drivers in Push/Pull Mode and in Open Drain Mode

Driver Characteristic

This defines either the general driving capability of the respective driver, or if the driver strength is reduced after the target output level has been reached or not. Reducing the driver strength increases the output's internal resistance which attenuates noise that is imported/exported via the output line. For driving LEDs or power transistors, however, a stable high output current may still be required.

The controllable output drivers of the XC164CM pins feature three differently sized transistors (strong, medium and weak) for each direction (push and pull). The time of activating/deactivating these transistors determines the output characteristics of the respective port driver.

The strength of the driver can be selected to adapt the driver characteristics to the application's requirements:

In Strong Driver Mode, the medium and strong transistors are activated. In this mode the driver provides maximum output current even after the target signal level is reached.

In Medium Driver Mode, only the medium transistors are activated while the other transistors remain off.

In Weak Driver Mode, only the weak transistor is activated while the other transistors remain off. This results in smooth transitions with low current peaks (and reduced susceptibility for noise) on the cost of increased transition times, i.e. slower edges, depending on the capacitive load.

Edge Characteristic

This defines the rise/fall time for the respective output, i.e. the output transition time. Soft edges reduce the peak currents that are drawn when changing the voltage level of an external capacitive load. For a bus interface, however, sharp edges may still be required. Edge characteristic effects the pre-driver which controls the final output driver stage.

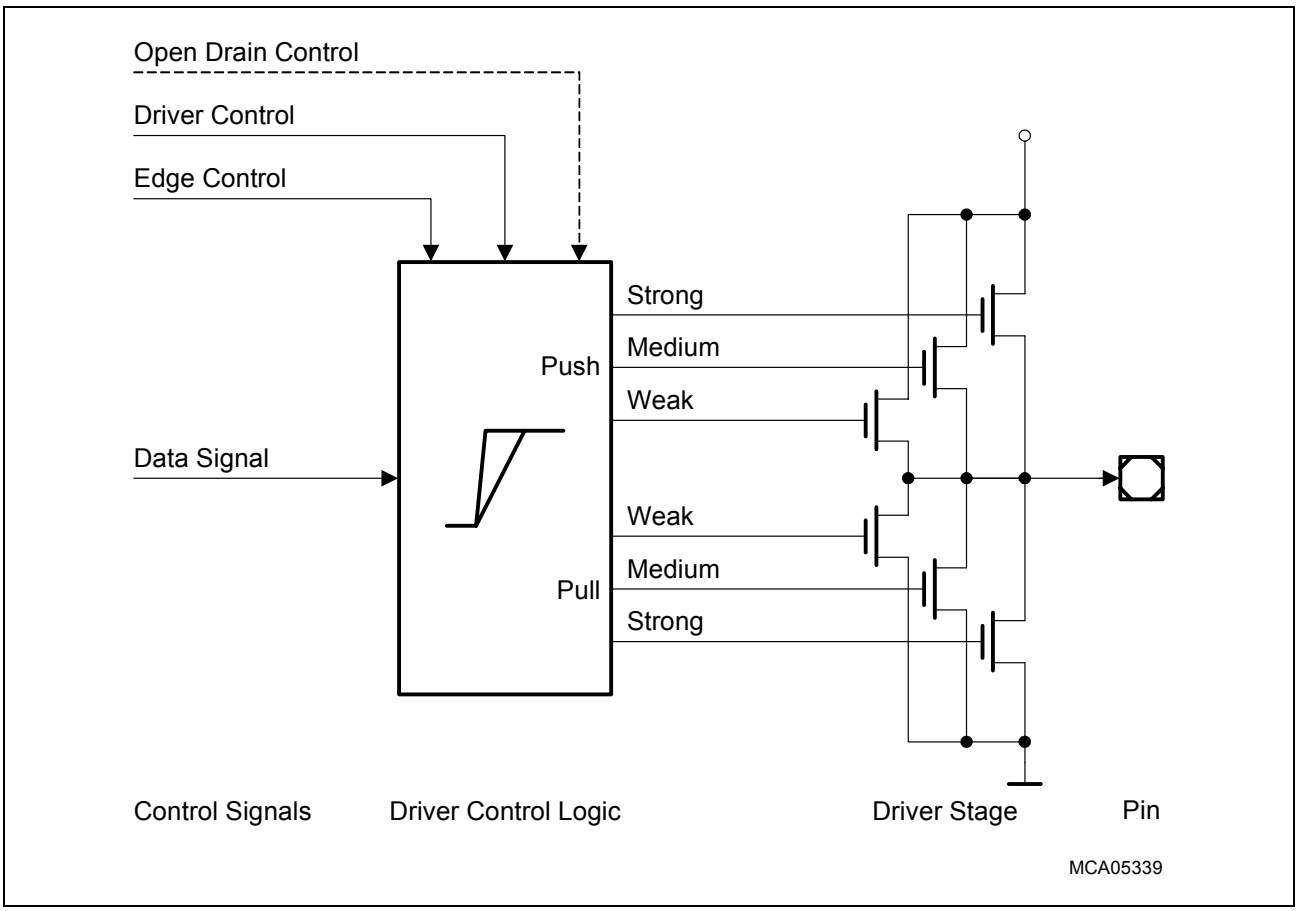
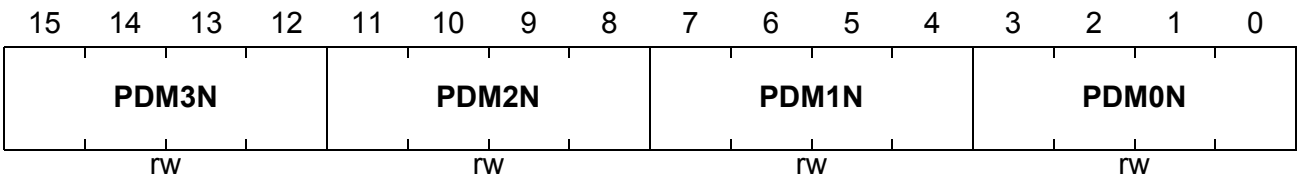


Figure 7-4 Structure of Three-Level Output Driver with Edge Control

Note: The upper (push) transistors are always off for output pins that operate in open drain mode.
***Figure 7-4** only shows the functional structure of the output drivers, not the real implementation.*

The **Port Output Control registers** POCONx 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. **Table 7-1** lists the defined POCON registers and the allocation of control bitfields and port pins.

POCON*
Port Output Ctrl. Reg.* **ESFR ($F0xx_H/yy_H$)** **Reset Value: 0000_H**



Field	Bits	Type	Description
PDMxN	[3:0], x = 0 [7:4], x = 1 [11:8], x = 2 [15:12], x = 3	rw	**Port Driver Mode, Nibble x** **Code, Driver strength[1], Edge Shape[2]** 0000 Strong driver, Sharp edge mode 0001 Strong driver, Medium edge mode 0010 Strong driver, Soft edge mode 0011 Weak driver, Standard edge[3] 0100 Medium driver, Standard edge[3] 0101 Reserved, do not use! 0110 Reserved, do not use! 0111 Reserved, do not use! 1xxx Reserved, do not use!

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.

3) No additional edge shaping can be selected at this driver level.

Table 7-1 Port Output Control Register Allocation

Control Register	Address	Controlled Pins (by POCONx.y-z)[1]				Notes
		.15-12	.11-8	.7-4	.3-0	
POCON9	$F094_H$ / $4A_H$	---	---	P9.5-4	P9.3-0	–
POCON3	$F08A_H$ / 45_H	P3.15, P3.13	P3.11-8	P3.7-4	P3.3-1	–
POCON1H	$F086_H$ / 43_H	---	---	P1H.5-4	P1H.3-0	–
POCON1L	$F084_H$ / 42_H	---	---	P1L.7-4	P1L.3-0	–

1) x denotes the port number, while y-z represents the bitfield range.

7.3 Alternate Port Functions

In order to provide maximum flexibility for different applications and their specific IO requirements, port lines have programmable alternate input or output functions associated with them.

If an **alternate output function** of a pin is to be used, the direction of this pin must be programmed for output (DPx.y = ‘1’), except for some signals that are used directly after reset and are configured automatically. Otherwise the pin remains in the high-impedance state and is not effected by the alternate output function. The software controlled output functions of a port are selected with one or two specific ALTSELnPx registers (n = 0 or 1).

If an **alternate input function** is used, the direction of the pin must be programmed for input (DPx.y = ‘0’) if an external device is driving the pin. The input direction is the default after reset. Alternate inputs are supported for some peripherals and for the external interrupt inputs. Alternate inputs for a peripheral are selected with the peripheral’s PISEL register, for example the CAN_PISEL register. Alternate external interrupt inputs are selected with the registers EXISEL0 and EXISEL1 in the SCU.

All port lines that are not used for these alternate functions may be used as general purpose IO lines. When using port pins for general purpose output, the initial output value should be written to the port latch prior to enabling the output drivers, in order to avoid undesired transitions on the output pins. This applies to single pins as well as to pin groups. In this case, the input operation reads the value stored in the port output latch. This can be used for testing purposes to allow a software trigger of an input function by writing to the port output latch.

7.4 Port 1

The 6-bit port P1H and the 8-bit port P1L represent the higher and lower byte of Port 1, respectively. Both halves of Port 1 can be written (e.g. via a PEC transfer) without effecting the other half.
If this port is used for general purpose IO, the direction of each line can be configured via the corresponding direction registers DP1H and DP1L.

P1L
Port 1 Low Register **SFR ($FF04_H/82_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								P7	P6	P5	P4	P3	P2	P1	P0
-								rwh	rw	rw	rw	rw	rw	rw	rw

P1H
Port 1 High Register **SFR ($FF06_H/83_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-										P5	P4	P3	P2	P1	P0
-										rwh	rwh	rw	rw	rw	rwh

Field	Bits	Type	Description
P1X.y	[7:0]	rw(h)	**Port Data Register P1H or P1L Bit y**

Note: Bits P1L.7, P1H.0 and P1H.4-5 are bit-protected for CAPCOM2 Output.

DP1L
P1L Direction Ctrl. Register **ESFR ($F104_H/82_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								P7	P6	P5	P4	P3	P2	P1	P0
-								rw	rw	rw	rw	rw	rw	rw	rw

DP1H
P1H Direction Ctrl. Register **ESFR ($F106_H/83_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-										P5	P4	P3	P2	P1	P0
-										rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
DP1X.y	[7:0]	rw	**Port Direction Register DP1H or DP1L Bit y** 0 Port line P1X.y is an input (high-impedance) 1 Port line P1X.y is an output

The alternate functions of the CAPCOM2 and the SSC1 are selected via the registers ALTSEL0P1L and ALTSEL0P1H.

ALTSEL0P1L
P1L Alternate Select Reg. 0 **ESFR ($F130_H/98_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								P7	P6	P5	P4	P3	P2	P1	P0
-								rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
ALTSEL0 P1L.y	[7:0]	rw	**P1L Alternate Select Register 0 Bit y** 0 associated peripheral output is not selected as alternate function 1 associated peripheral output is selected as alternate function

ALTSEL0P1H
P1H Alternate Select Reg. 0 **ESFR ($F120_H/90_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-										P5	P4	P3	P2	P1	P0
-										rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
ALTSEL0 P1H.y	[5:0]	rw	**P1H Alternate Select Register 0 Bit y** 0 associated peripheral output is not selected as alternate function 1 associated peripheral output is selected as alternate function

Alternate Functions of Port 1

Port 1 IO and alternate functions are shown in **Figure 7-5**.

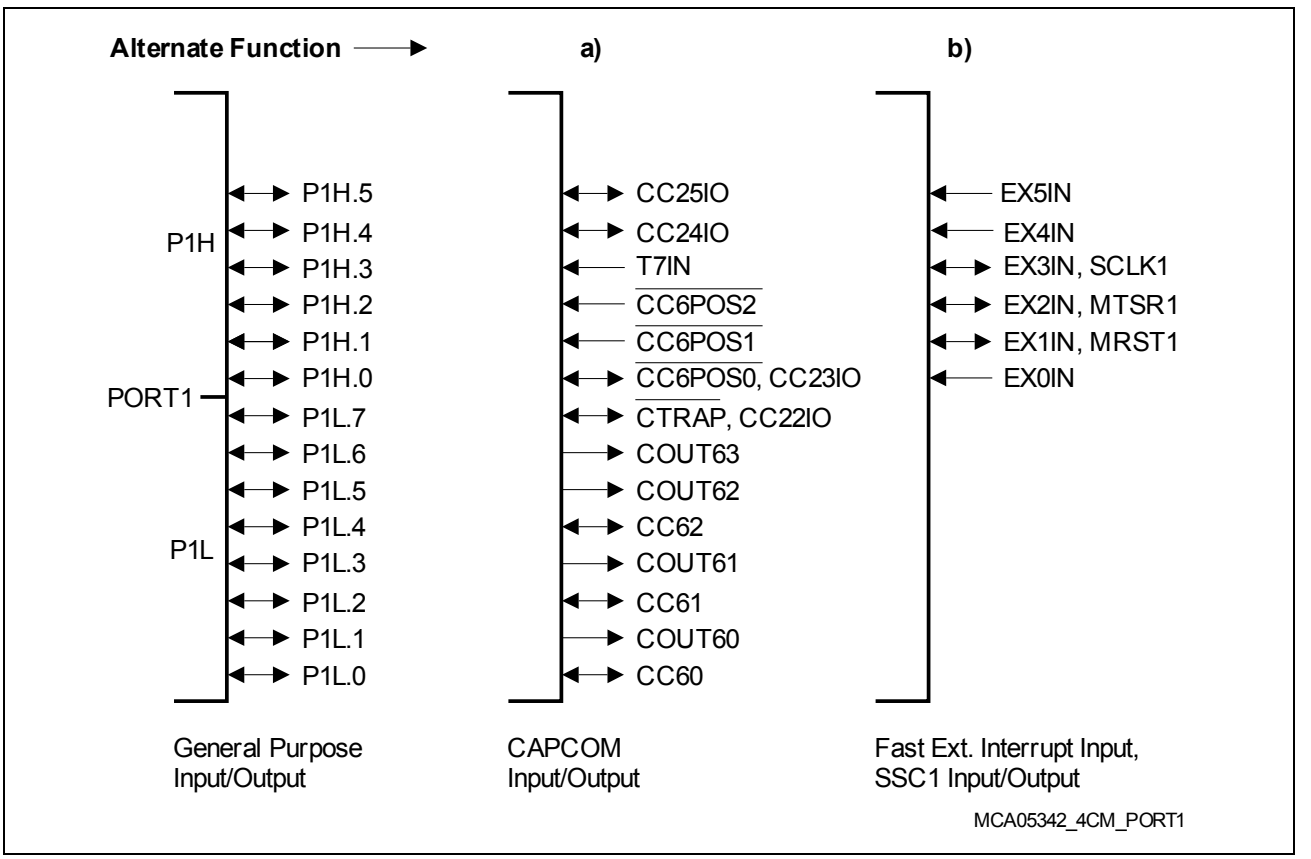


Figure 7-5 Port 1 IO and Alternate Functions

Table 7-2 shows how the functions of each Port 1 pin can be set.

Note: The compare output signals listed here are derived from the CAPCOM2 unit's OUT register. If the CAPCOM unit controls the port latch directly, the output multiplexer must select general purpose output.

Table 7-2 Port 1 Functions

Port Pin	Pin Function	Associated Register/ Module	Alternate Function	Control Direction
P1L.0	General purpose input	P1L.0	ALTSEL0P1L.P0 = 0	DP1L.P0 = 0
	General purpose output			DP1L.P0 = 1
	CC60I Capture Input	CAPCOM6	–	DP1L.P0 = 0
	CC60O Compare Output	CAPCOM6	ALTSEL0P1L.P0 = 1	DP1L.P0 = 1
P1L.1	General purpose input	P1L.1	ALTSEL0P1L.P1 = 0	DP1L.P1 = 0
	General purpose output			DP1L.P1 = 1
	COUT60 Compare Output	CAPCOM6	ALTSEL0P1L.P1 = 1	DP1L.P0 = 1
P1L.2	General purpose input	P1L.2	ALTSEL0P1L.P2 = 0	DP1L.P2 = 0
	General purpose output			DP1L.P2 = 1
	CC61I Capture Input	CAPCOM6	–	DP1L.P2 = 0
	CC61O Compare Output	CAPCOM6	ALTSEL0P1L.P2 = 1	DP1L.P2 = 1
P1L.3	General purpose input	P1L.3	ALTSEL0P1L.P3 = 0	DP1L.P3 = 0
	General purpose output			DP1L.P3 = 1
	COUT61 Compare Output	CAPCOM6	ALTSEL0P1L.P3 = 1	DP1L.P3 = 1
P1L.4	General purpose input	P1L.4	ALTSEL0P1L.P4 = 1	DP1L.P4 = 0
	General purpose output			DP1L.P4 = 1
	CC62I Capture Input	CAPCOM6	–	DP1L.P4 = 0
	CC62O Compare Output	CAPCOM6	ALTSEL0P1L.P4 = 1	DP1L.P4 = 1
P1L.5	General purpose input	P1L.5	ALTSEL0P1L.P5 = 0	DP1L.P5 = 0
	General purpose output			DP1L.P5 = 1
	COUT62 Compare Output	CAPCOM6	ALTSEL0P1L.P5 = 1	DP1L.P5 = 1

Table 7-2 Port 1 Functions (cont'd)

Port Pin	Pin Function	Associated Register/ Module	Alternate Function	Control Direction
P1L.6	General purpose input	P1L.6	ALTSEL0P1L.P6 = 0	DP1L.P6 = 0
	General purpose output			DP1L.P6 = 1
	COUT63 Compare Output	CAPCOM6	ALTSEL0P1L.P6 = 1	DP1L.P6 = 1
P1L.7	General purpose input	P1L.7	ALTSEL0P1L.P7 = 0	DP1L.P7 = 0
	General purpose output			DP1L.P7 = 1
	CC22I Capture Input	CAPCOM2	–	DP1L.P7 = 0
	CC22O Compare Output		ALTSEL0P1L.P7 = 1	DP1L.P7 = 1
	$\overline{\text{CTRAP}}$ Trap input	CAPCOM6	–	DP1L.P7 = 0
P1H.0	General purpose input	P1H.0	ALTSEL0P1H.P0 = 0	DP1H.P0 = 0
	General purpose output			DP1H.P0 = 1
	CC23I Capture Input	CAPCOM2	–	DP1H.P0 = 0
	CC23O Compare Output		ALTSEL0P1H.P0 = 1	DP1H.P0 = 1
	$\overline{\text{CC6POS0}}$ Position 0 Input	CAPCOM6	–	DP1H.P0 = 0
	Fast External Interrupt input EX0IN	–	–	DP1H.P0 = 0
P1H.1	General purpose input	P1H.1	ALTSEL0P1H.P1 = 0	DP1H.P1 = 0
	General purpose output			DP1H.P1 = 1
	SSC1 master receive input MRST1	SSC1	–	DP1H.P1 = 0
	SSC1 slave transmit output MRST1		ALTSEL0P1H.P1 = 1	DP1H.P1 = 1
	$\overline{\text{CC6POS1}}$ Position 1 Input	CAPCOM6	–	DP1H.P1 = 0
	Fast External Interrupt input EX1IN	–	–	DP1H.P1 = 0

Table 7-2 Port 1 Functions (cont'd)

Port Pin	Pin Function	Associated Register/ Module	Alternate Function	Control Direction
P1H.2	General purpose input	P1H.2	ALTSEL0P1H.P2 = 0	DP1H.P2 = 0
	General purpose output			DP1H.P2 = 1
	SSC1 slave receive input MTSR1	SSC1	–	DP1H.P2 = 0
	SSC1 master transmit output MTSR1		ALTSEL0P1H.P2 = 1	DP1H.P2 = 1
	$\overline{\text{CC6POS2}}$ Position 2 Input	CAPCOM6	–	DP1H.P2 = 0
	Fast External Interrupt input EX2IN	–	–	DP1H.P2 = 0
P1H.3	General purpose input	P1H.3	ALTSEL0P1H.P3 = 0	DP1H.P3 = 0
	General purpose output			DP1H.P3 = 1
	SSC1 clock input SCLK1	SSC1	–	DP1H.P3 = 0
	SSC1 clock output SCLK1		ALTSEL0P1H.P3 = 1	DP1H.P3 = 1
	Fast External Interrupt input EX3IN	–	–	DP1H.P3 = 0
P1H.x (x = 5-4)	General purpose input	P1H.x	ALTSEL0P1H.Px = 0	DP1H.Px = 0
	General purpose output			DP1H.Px = 1
	CC25 to CC24 Capture Input	CAPCOM2	–	DP1H.Px = 0
	CC25 to CC24 Compare Output		ALTSEL0P1H.Px = 1	DP1H.Px = 1
	Fast External Interrupt input EXxIN	–	–	DP1H.Px = 0

For a complete list of the external interrupt connections, see **Table 5-13**.

The subsequent figures show the different configurations of a Port 1 pins.

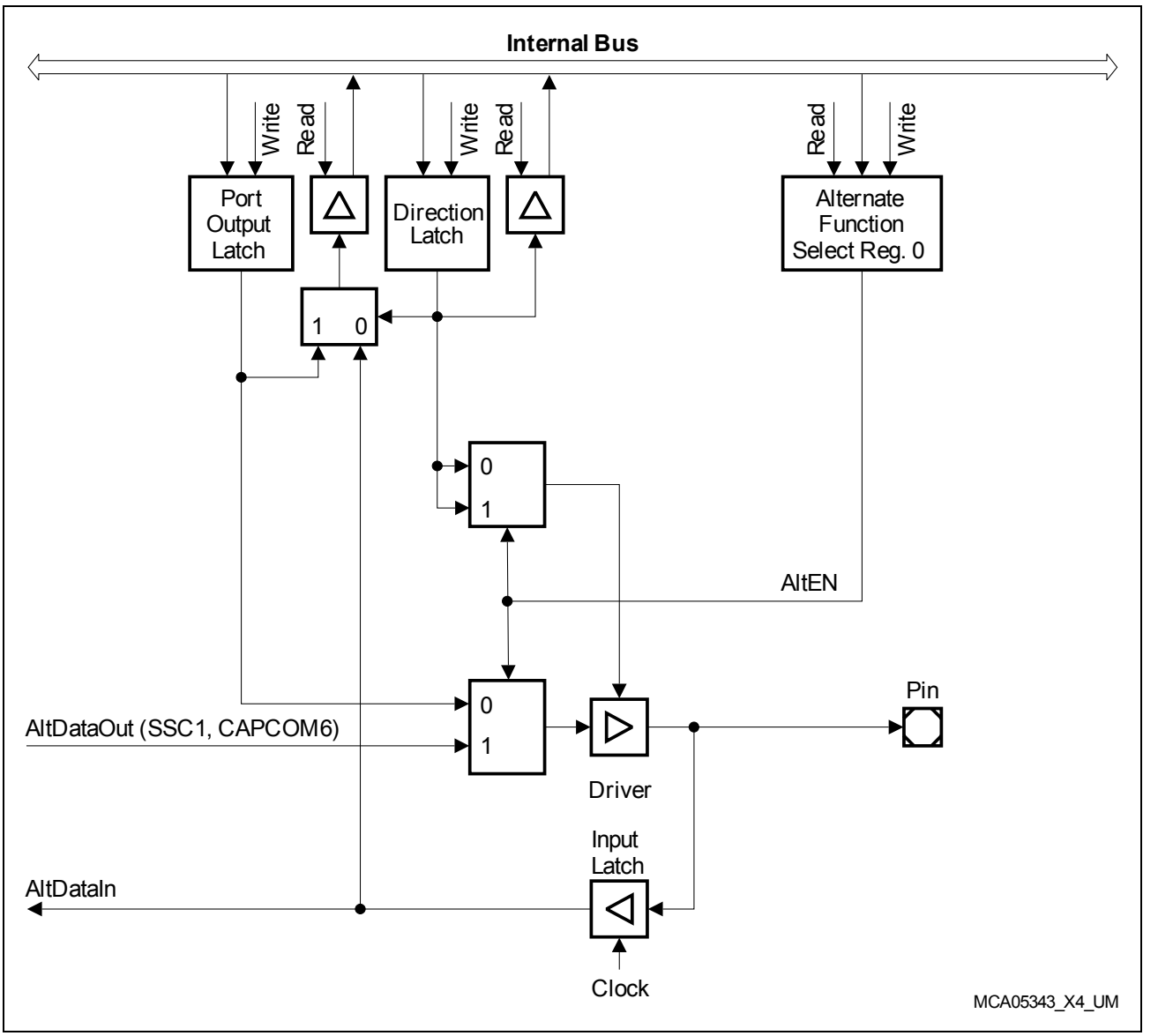


Figure 7-6 P1L.0 to P1L.6 and P1H.1 to P1H.3 Port Configuration

Table 7-3 P1L.0 to P1L.6 and P1H.1 to P1H.3 Alternate Function Control

Pins	Control Lines		Registers		Function
	AltEN	AltDIR	DP1L/ DP1H	ALTSEL0 P1L/P1H	
P1L.y (y = 0-6) P1H.x (x = 1-3)	0	–	0 or 1	0	GPIO
	–		0	–	SSC1, CAPCOM6 capture input
	1		1	1	SSC1, CAPCOM6 compare output

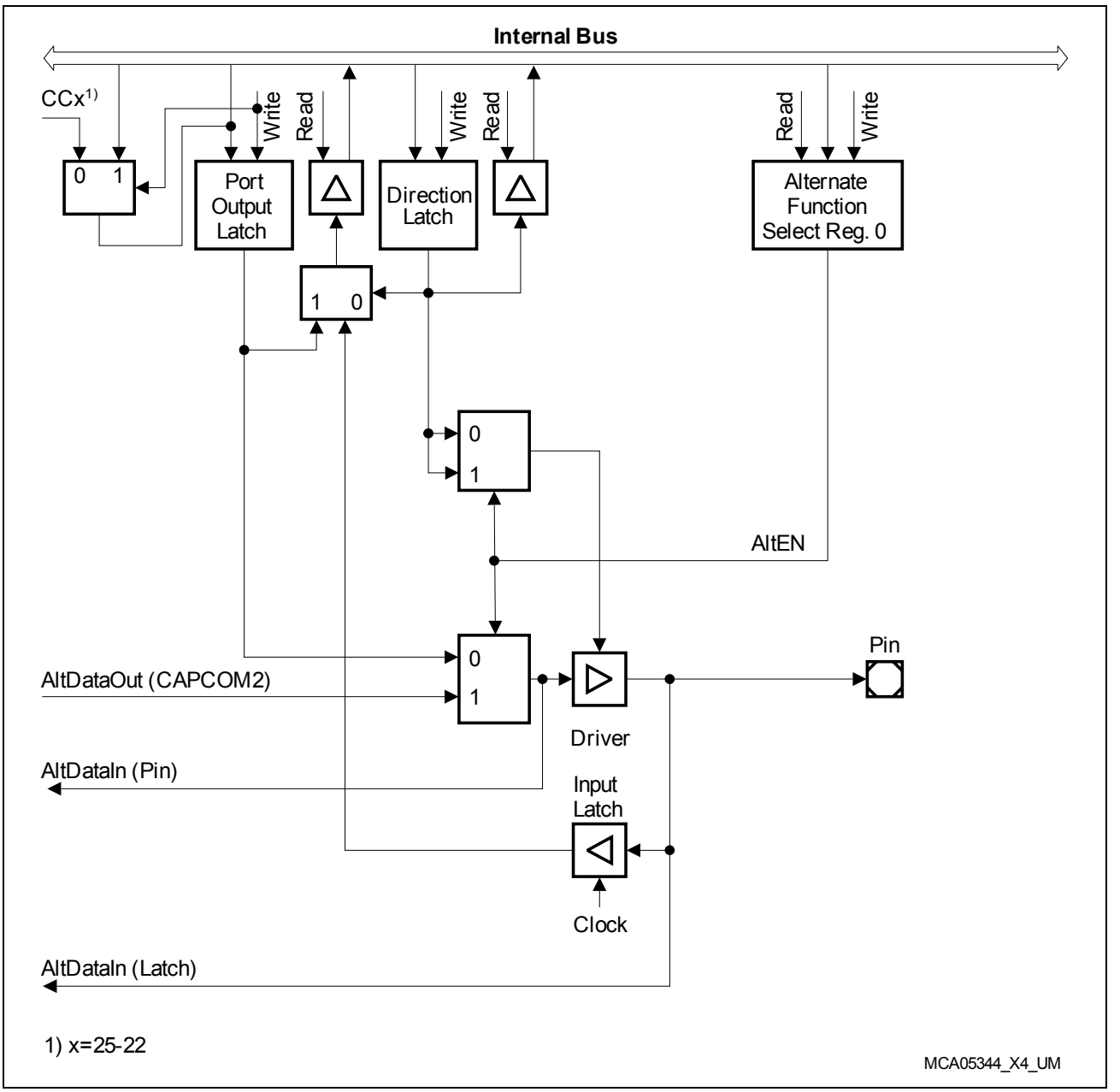


Figure 7-7 P1L.7, P1H.0, P1H.4, and P1H.5 Port Configuration

Table 7-4 P1L.7, P1H.0, P1H.4 and P1H.5 Alternate Function Control

Pins	Control Lines		Registers		Function
	AltEN	AltDIR	DP1L/ DP1H	ALTSEL0 P1L/P1H	
P1L.7	0	–	0 or 1	0	GPIO
P1H.0	–		0	–	CAPCOM2 capture input
P1H.x (x = 4, 5)	1		1	1	CAPCOM2 compare output

7.5 Port 3

If this 13-bit port is used for general purpose IO, the direction of each line can be configured via the corresponding direction register DP3. Each port line can be switched into push/pull or open drain mode via the open drain control register ODP3.

P3
Port 3 Data Register **SFR ($FFC4_H/E2_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P15	**-**	**P13**	**-**	**P11**	**P10**	**P9**	**P8**	**P7**	**P6**	**P5**	**P4**	**P3**	**P2**	**P1**	**-**
rw	-	rw	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	-

Field	Bits	Type	Description
P3.y	[11:1], 13, 15	rw	**Port Data Register P3 Bit y**

DP3
P3 Direction Ctrl. Register **SFR ($FFC6_H/E3_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P15	**-**	**P13**	**-**	**P11**	**P10**	**P9**	**P8**	**P7**	**P6**	**P5**	**P4**	**P3**	**P2**	**P1**	**-**
rw	-	rw	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	-

Field	Bits	Type	Description
DP3.y	[11:1], 13, 15	rw	**Port Direction Register DP3 Bit y** 0 Port line P3.y is an input (high-impedance) 1 Port line P3.y is an output

ODP3
P3 Open Drain Ctrl. Reg. **ESFR ($F1C6_H/E3_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	P13	-	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	-
-	-	rw	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	-

Field	Bits	Type	Description
ODP3.y	[11:1], 13	rw	**Port 3 Open Drain Control Register Bit y** 0 Port line P3.y output driver in push/pull mode 1 Port line P3.y output driver in open drain mode

Note: Pin P3.15 does not support open drain mode.

The alternate functions of the SSC0, ASC0, ASC1 and GPT are selected via the registers ALTSEL0P3 and ALTSEL1P3.

Note: For the exact selection of a peripheral output as alternate function, refer to ***Table 7-5****.*

ALTSEL0P3
P3 Alternate Select Reg. 0 **ESFR ($F126_H/93_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	P13	-	P11	P10	P9	P8	-	-	P5	-	P3	-	P1	-
-	-	rw	-	rw	rw	rw	rw	-	-	rw	-	rw	-	rw	-

Field	Bits	Type	Description
ALTSEL0 P3.y	1, 3, 5, [11:8], 13	rw	**P3 Alternate Select Register 0 Bit y** 0 Associated peripheral output is not selected as alternate function 1 Associated peripheral output is selected as alternate function

ALTSEL1P3
P3 Alternate Select Reg. 1 **ESFR ($F128_H/94_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	-	-	**P1**	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	rw	-

Field	Bits	Type	Description
ALTSEL1 P3.y	1	rw	**P3 Alternate Select Register 1 Bit y** 0 Associated peripheral output is not selected as alternate function 1 Associated peripheral output is selected as alternate function

Alternate Functions of Port 3

During normal mode, Port 3 serves for various functions which include external timer control lines, two serial interfaces, and CLKOUT/FOUT. The Port 3 IO and alternate functions are shown in **Figure 7-8**.

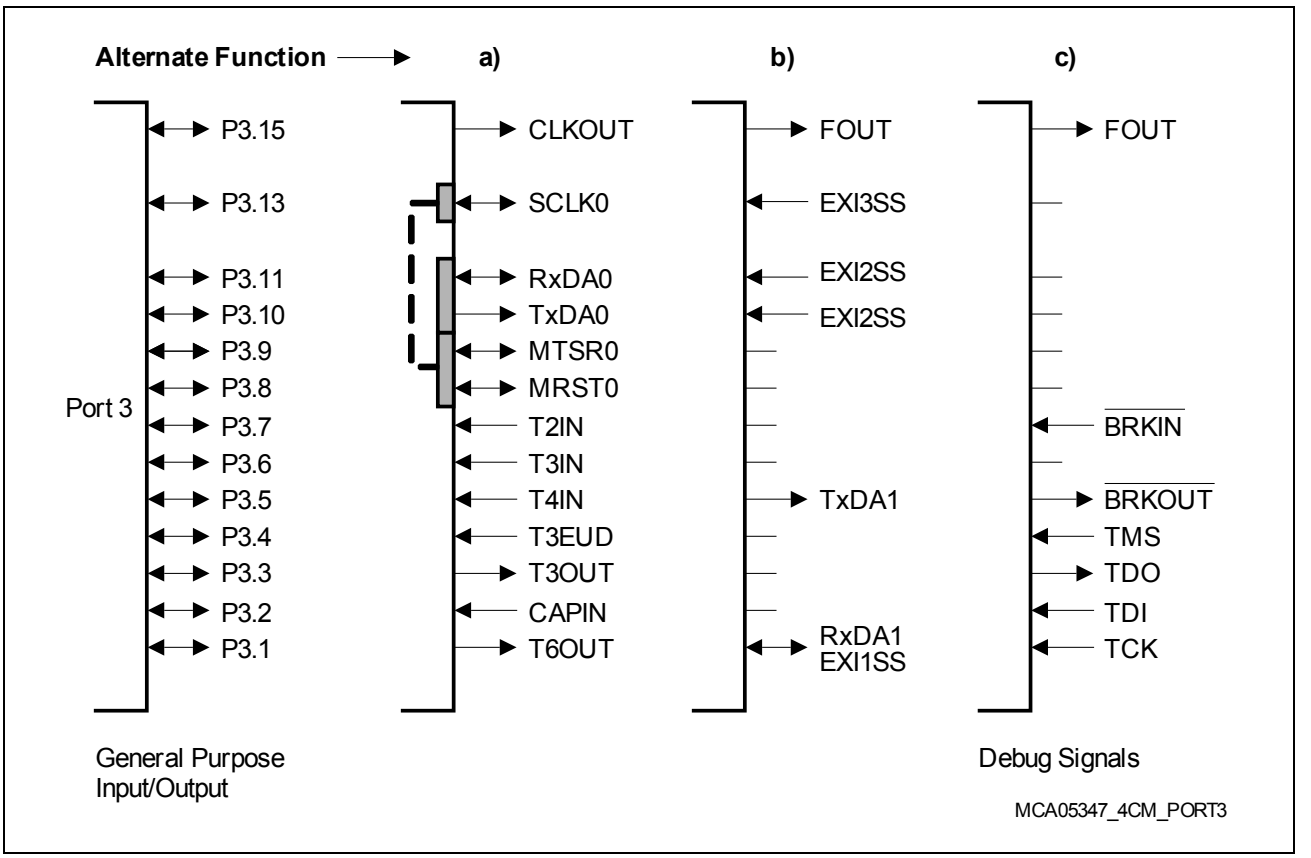


Figure 7-8 Port 3 IO and Alternate Functions

The alternate output functions - TxDA1, T6OUT, T3OUT, MRST0, MTSR0, TxDA0, RxDA0 and SCLK0 - when selected, is ANDed with the port output latch line (general purpose output).

A complete listing of Port 3 functions is found in **Table 7-5**.

Table 7-5 Port 3 Functions

Port Pin	Pin Function	Associated Register/ Module	Alternate Function	Control Direction
P3.0	Not Implemented	–	–	–

Table 7-5 Port 3 Functions (cont'd)

Port Pin	Pin Function	Associated Register/ Module	Alternate Function	Control Direction
P3.1	General purpose input	P3.1	ALTSEL0P3.P1 = 0 and ALTSEL1P3.P1 = 0	DP3.P1 = 0
	General purpose output			DP3.P1 = 1
	Timer T6 Toggle Latch output, T6OUT	GPT	ALTSEL0P3.P1 = 0 and ALTSEL1P3.P1 = 1 and P3.P1 = 1	DP3.P1 = 1
	ASC1 receiver input RxDA1, used as input	ASC1	–	DP3.P1 = 0
	ASC1 receiver input RxDA1, used as output		ALTSEL0P3.P1 = 1	DP3.P1 = 1
	JTAG Clock Input, TCK	OCDS	–	DP3.P1 = 0
P3.2	General purpose input	P3.2	–	DP3.P2 = 0
	General purpose output			DP3.P2 = 1
	GPT12E Capture input CAPIN	GPT		DP3.P2 = 0
	JTAG Data In, TDI	OCDS	–	DP3.P2 = 0
P3.3	General purpose input	P3.3	ALTSEL0P3.P3 = 0	DP3.P3 = 0
	General purpose output			DP3.P3 = 1
	Timer 3 Toggle Latch output, T3OUT	GPT	ALTSEL0P3.P3 = 1 and P3.P3 = 1	DP3.P3 = 1
	JTAG Data Out, TDO	OCDS	AltEN1 = 1	–
P3.4	General purpose input	P3.4	–	DP3.P4 = 0
	General purpose output			DP3.P4 = 1
	Timer 3 external up/down input, T3EUD	GPT		DP3.P4 = 0
	JTAG Test Mode Select input, TMS	OCDS	–	DP3.P4 = 0

Table 7-5 Port 3 Functions (cont'd)

Port Pin	Pin Function	Associated Register/ Module	Alternate Function	Control Direction
P3.5	General purpose input	P3.5	ALTSEL0P3.P5 = 0	DP3.P5 = 0
	General purpose output			DP3.P5 = 1
	Timer 4 count input, T4IN	GPT	–	DP3.P5 = 0
	ASC1 transmitter output TxDA1	ASC1	ALTSEL0P3.P5 = 1	DP3.P5 = 1
	Debug System: Break Out, $\overline{\text{BRKOUT}}$	OCDS	AltEN1 = 1	–
P3.6	General purpose input	P3.6	–	DP3.P6 = 0
	General purpose output			DP3.P6 = 1
	Timer 3 count input, T3IN	GPT	–	DP3.P6 = 0
P3.7	General purpose input	P3.7	AltEN1.7 = 0	DP3.P7 = 0
	General purpose output			DP3.P7 = 1
	Timer 2 count input, T2IN	GPT	–	DP3.P7 = 0
	Debug System: Break In, $\overline{\text{BRKIN}}$	OCDS	–	DP3.P7 = 0
P3.8	General purpose input	P3.8	ALTSEL0P3.P8 = 0	DP3.P8 = 0
	General purpose output			DP3.P8 = 1
	SSC0 master receive input, MRST0	SSC0	–	DP3.P8 = 0
	SSC0 slave transmit output, MRST0		ALTSEL0P3.P8 = 1 and P3.P8 = 1	DP3.P8 = 1
P3.9	General purpose input	P3.9	ALTSEL0P3.P9 = 0	DP3.P9 = 0
	General purpose output			DP3.P9 = 1
	SSC0 slave receive input MTSR0	SSC0	–	DP3.P9 = 0
	SSC0 master transmit output, MTSR0		ALTSEL0P3.P9 = 1 and P3.P9 = 1	DP3.P9 = 1

Table 7-5 Port 3 Functions (cont'd)

Port Pin	Pin Function	Associated Register/ Module	Alternate Function	Control Direction
P3.10	General purpose input	P3.10	ALTSEL0P3.P10 = 0	DP3.P10 = 0
	General purpose output			DP3.P10 = 1
	ASC0 transmitter output TxDA0	ASC0	ALTSEL0P3.P10 = 1 and P3.P10 = 1	DP3.P10 = 1
P3.11	General purpose input	P3.11	ALTSEL0P3.P11 = 0	DP3.P11 = 0
	General purpose output			DP3.P11 = 1
	ASC0 receiver input RxDA0, used as input	ASC0	–	DP3.P11 = 0
	ASC0 receiver input RxDA0, used as output		ALTSEL0P3.P11 = 1 and P3.P11 = 1	DP3.P11 = 1
P3.12	Not Implemented			
P3.13	General purpose input	P3.13	ALTSEL0P3.P13 = 0	DP3.P13 = 0
	General purpose output			DP3.P13 = 1
	SSC0 slave clock input, SCLK0	SSC0	–	DP3.P13 = 0
	SSC0 master clock output, SCLK0		ALTSEL0P3.P13 = 1 and P3.P13 = 1	DP3.P13 = 1
P3.14	Not Implemented			
P3.15	General purpose input	P3.15	Both CLKOUT and FOUT disabled	DP3.P15 = 0
	General purpose output			DP3.P15 = 1
	System Clock output CLKOUT	–	CLKOUT enabled (AltEN1.15)	Output (AltDIR = 1)
	Programmable Freq. output, FOUT	–	CLKOUT disabled and FOUT enabled (AltEN2.15)	Output (AltDIR = 1)

For a complete list of the external interrupt connections, see **Table 5-13**.

The subsequent figures show the different configurations of Port 3 pins.

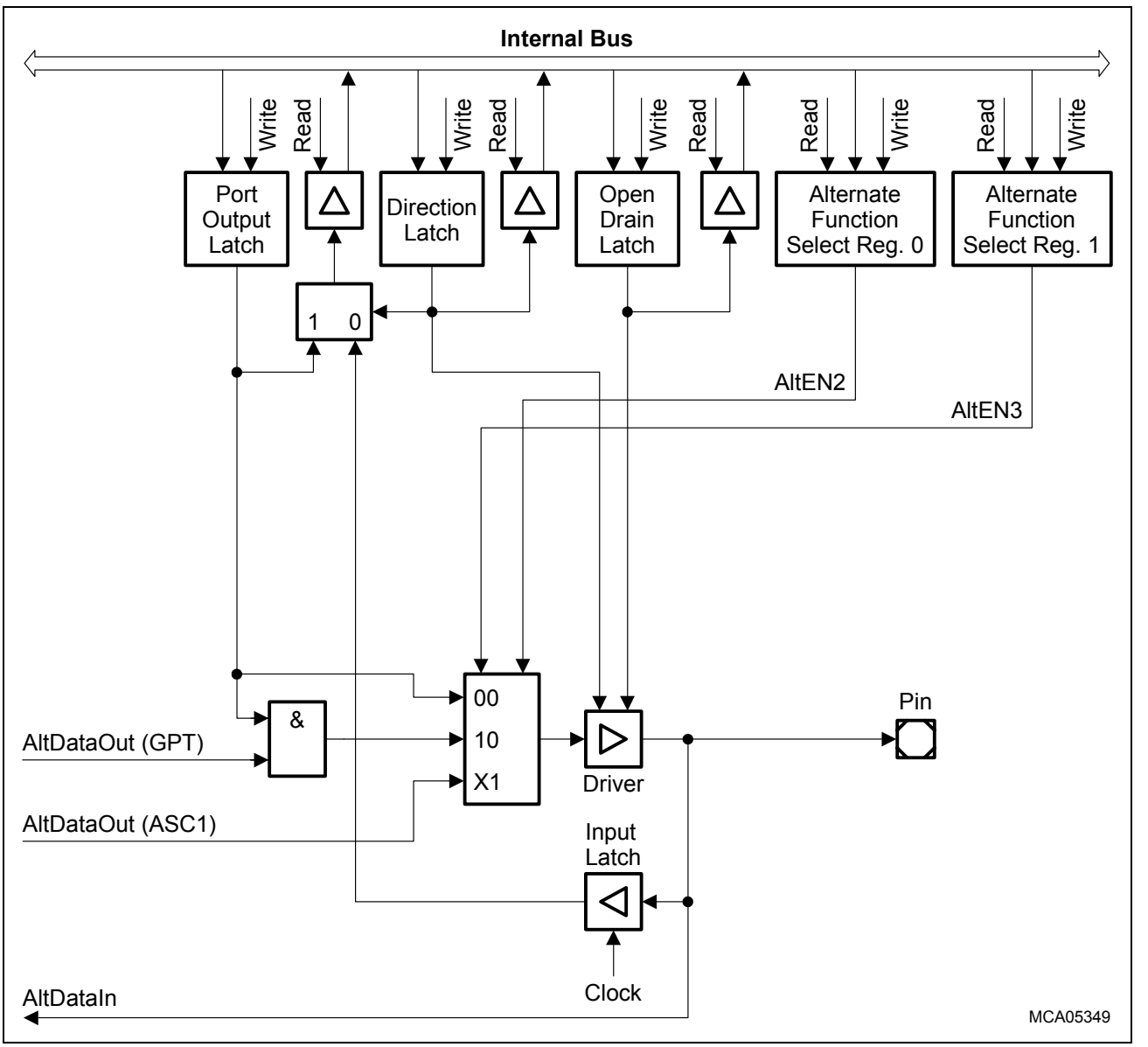


Figure 7-9 P3.1 Port Configuration

Table 7-6 P3.1 Alternate Function Control

Pins	Control Lines				Registers				Function
	AltEN			AltDIR	DP3L	P3	ALTSEL0 P3	ALTSEL1 P3	
	3	2	1						
P3.1	0	0	–	–	0 or 1	0 or 1	0	0	GPIO
	1	0			1	1	0	1	GPT output
	X	1			1	–	1	–	ASC1 output

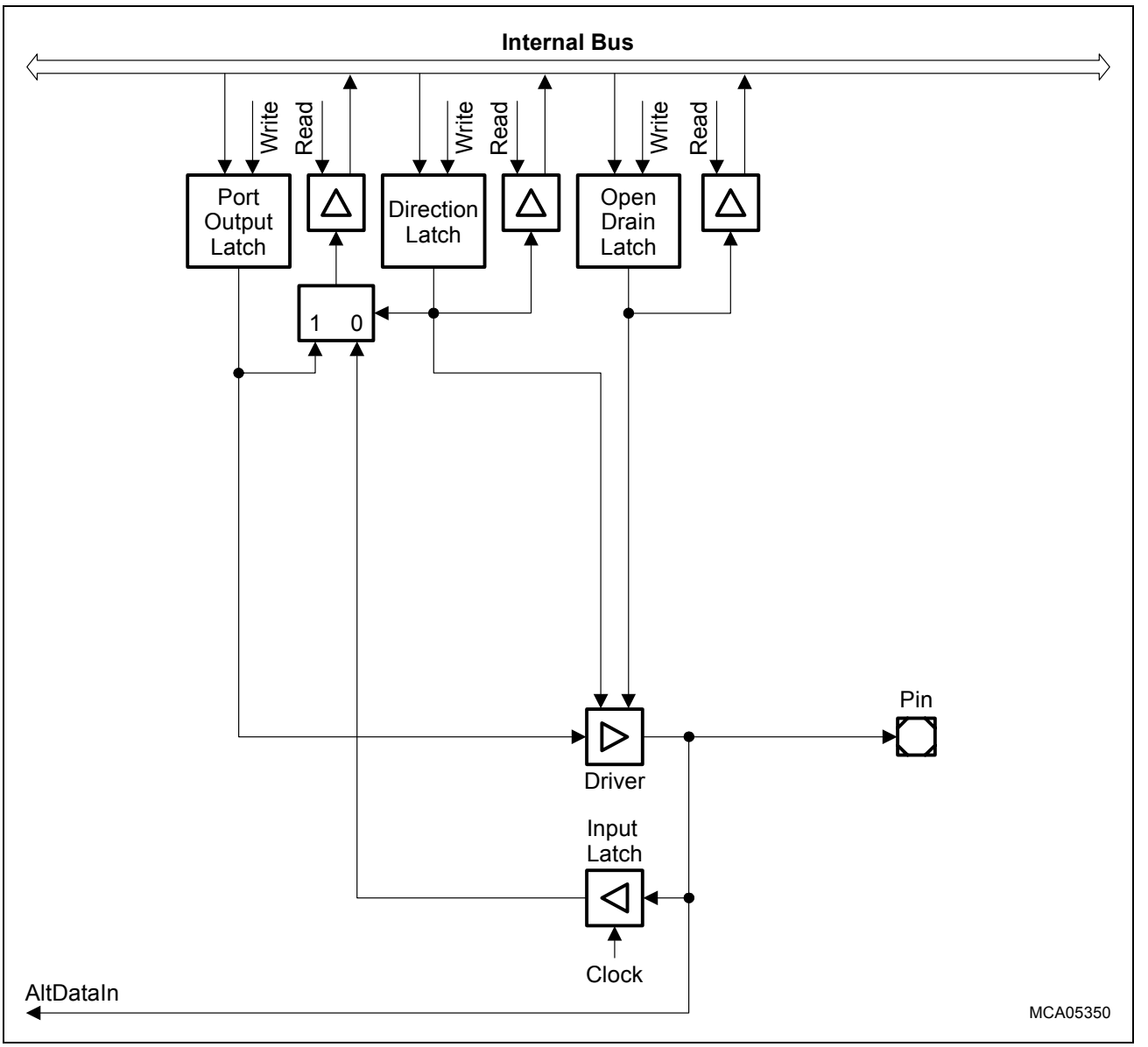


Figure 7-10 P3.2, P3.4, P3.6 and P3.7 Port Configuration

Table 7-7 P3.2, P3.4, P3.6, and P3.7 Alternate Function Control

Pins	Control Lines			Registers		Function
	AltEN		AltDIR	DP3L	ALTSEL0 P3	
	2	1				
P3.2, P3.4, P3.6, P3.7	–	–	–	0 or 1	–	GPIO
				0		GPT input

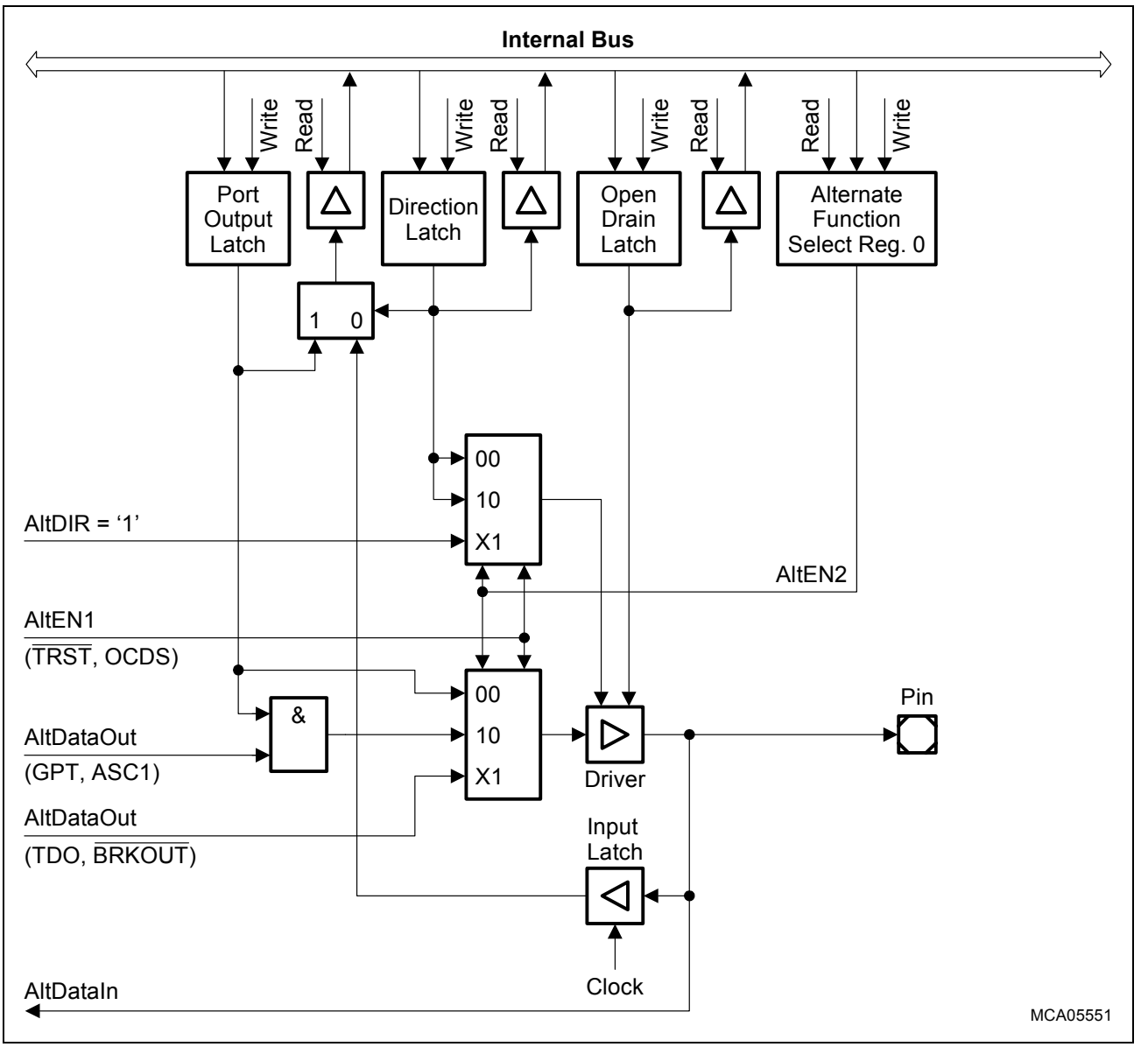


Figure 7-11 P3.3 and P3.5 Port Configuration

Table 7-8 P3.3[1] and P3.5 Alternate Function Control

Pins	Control Lines			Registers			Function
	AltEN		AltDIR	DP3L	P3	ALTSEL0 P3	
	2	1					
P3.3 P3.5	0	0	–	0 or 1	0 or 1	0	GPIO
	1	0		1	1[2]	1	GPT output, ASC1 output
	x	1	1	–	–	–	Debug output

1) Pin P3.3 has no alternate input function assigned.

2) Pin P3.5 has no AND combining the alternate output signal with the port latch signal.

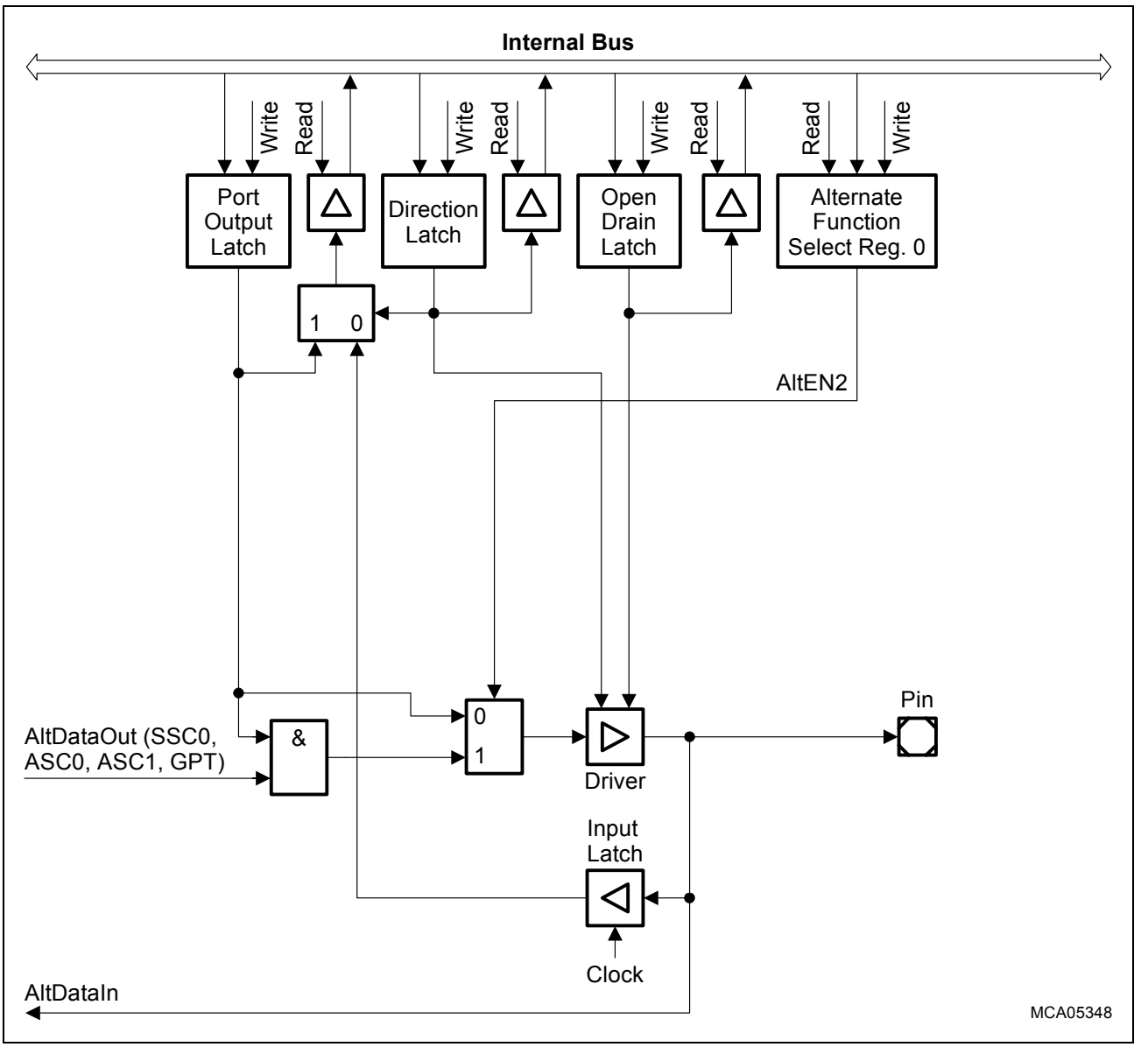


Figure 7-12 P3.8 to P3.11, and P3.13 Port Configuration

Table 7-9 P3.8 to P3.11, P3.13 Alternate Function Control

Pins	Control Lines			Registers			Function
	AltEN		AltDIR	DP3L	P3	ALTSEL0 P3	
	2	1					
P3.8-11 P3.13	0	–	–	0 or 1	0 or 1	0	GPIO
	1			1	1	1	SSC0, ASC0, ASC1, GPT output
	–		–	0	–	–	CAPCOM1, SSC0, ASC0 input

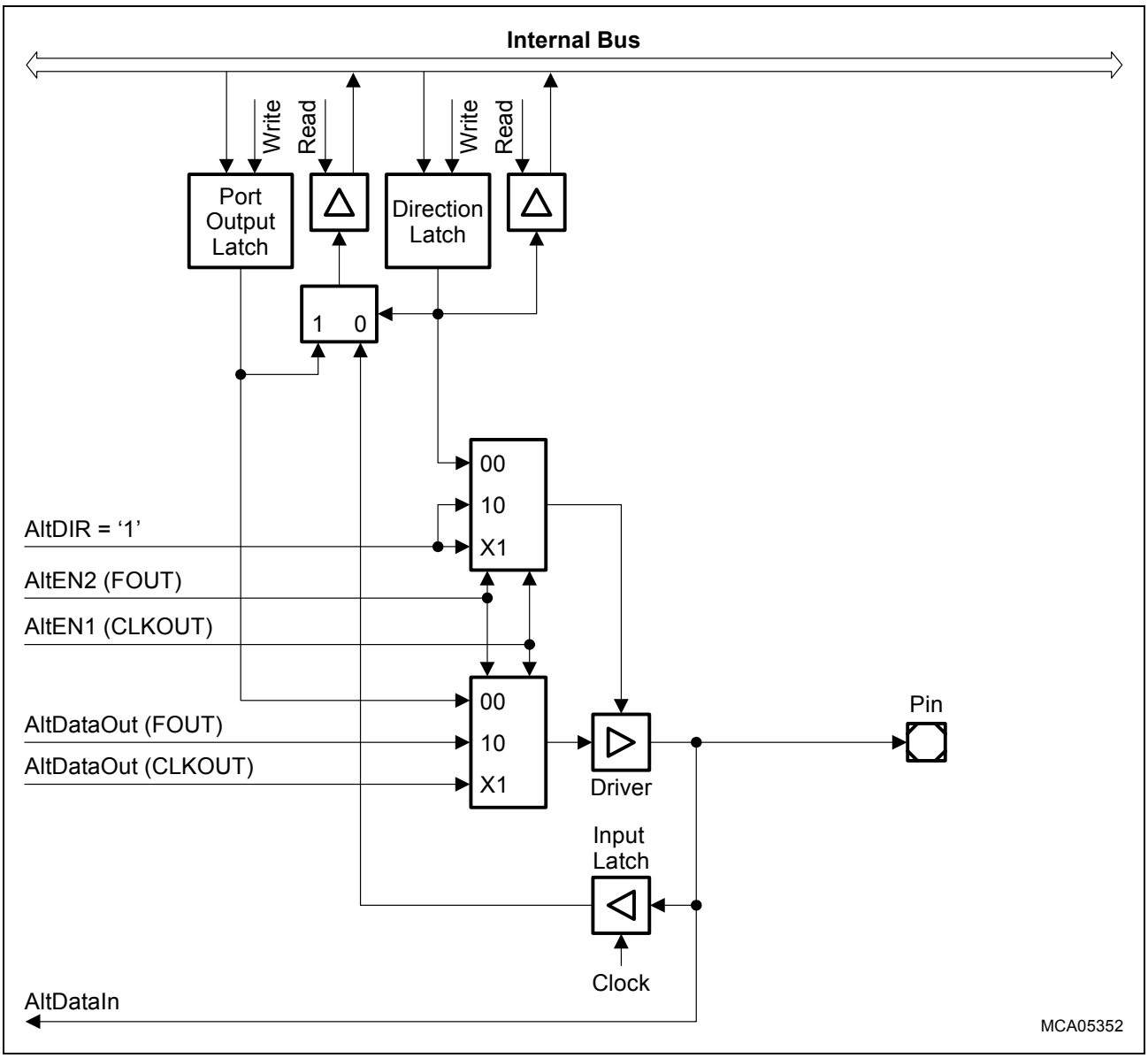


Figure 7-13 P3.15 Port Configuration

Table 7-10 P3.15 Alternate Function Control

Pins	Control Lines			Registers		Function
	AltEN		AltDIR	DP3L	ALTSEL0 P3	
	2	1				
P3.15	0	0	–	0 or 1	–	GPIO
	x	1	1	–		CLKOUT
	1	0				FOUT

7.6 Port 5

Port 5 is a 14-bit input port. There is no output latch and no direction register. Data written to P5 will be lost.

P5
Port 5 Data Register **SFR ($FFA2_H/D1_H$)** **Reset Value: $XXXX_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P15	**P14**	**P13**	**P12**	**P11**	**P10**	**-**	**-**	**P7**	**P6**	**P5**	**P4**	**P3**	**P2**	**P1**	**P0**
r	r	r	r	r	r	-	-	r	r	r	r	r	r	r	r

Field	Bits	Type	Description
P5.y	[15:10], [7:0]	r	**Port Data Register P5 Bit y** (Read only)

Alternate Functions of Port 5

Each line of Port 5 is connected to the input multiplexer of the Analog/Digital Converter. The upper 6 bits are also used as alternate input functions of the GPT. The IO and alternate functions of Port 5 are shown in **Figure 7-14**.

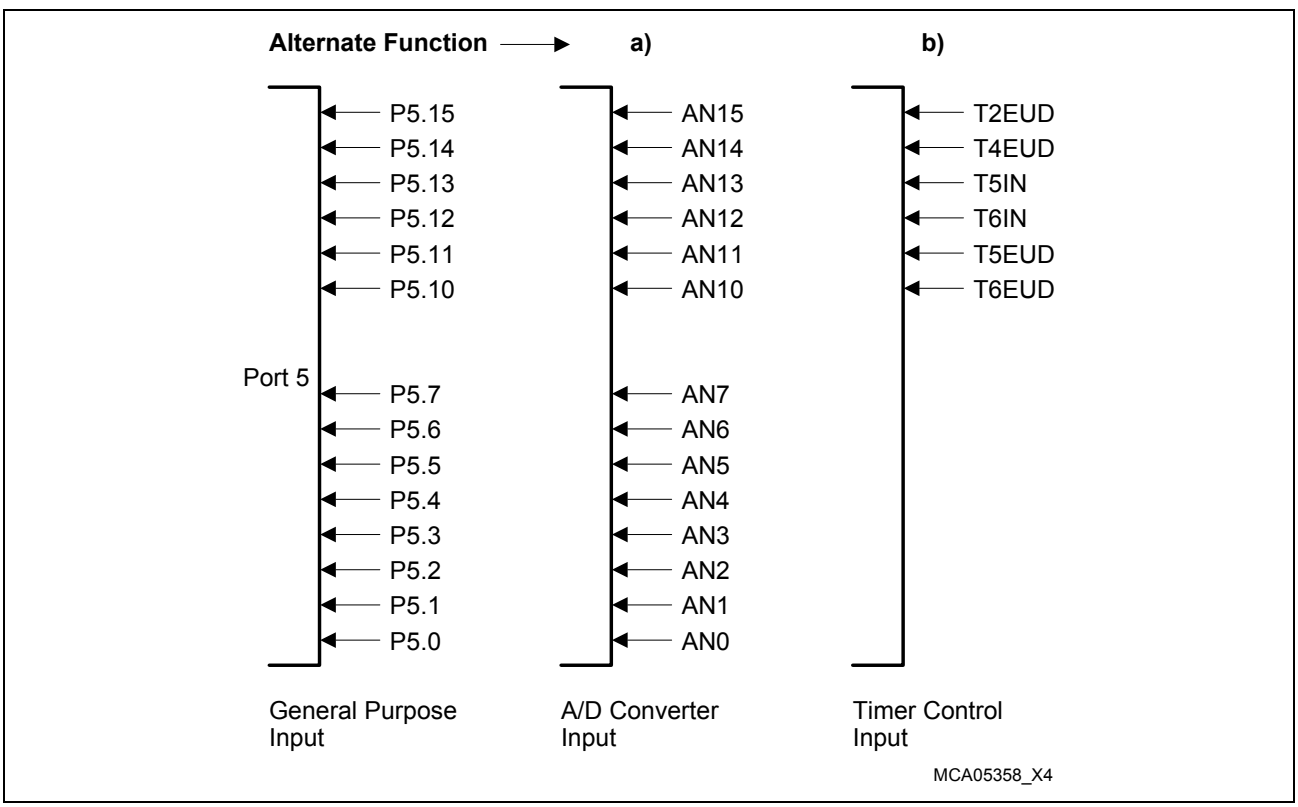


Figure 7-14 Port 5 IO and Alternate Functions

Port 5 Digital Input Control

Port 5 pins may be used for both digital and analog input. To use a Port 5 pin as an analog input, the Schmitt-trigger in its input stage must be disabled. This is achieved by setting the corresponding bit in the register P5DIDIS.

After reset, Port 5 pins are enabled for digital inputs.

P5DIDIS
Port 5 Digital Inp. Disable Reg. SFR ($FFA4_H/D2_H$) 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**	**-**	**-**	**P7**	**P6**	**P5**	**P4**	**P3**	**P2**	**P1**	**P0**
rw	rw	rw	rw	rw	rw	-	-	rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
P5DIDIS.y	[15:10], [7:0]	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.

The functions of Port 5 pins are listed in **Table 7-11**.

Table 7-11 Port 5 Functions

Port Pin	Pin Function	Associated Register/ Module	Alternate Function	Control Direction
P5.x (x = 7-0)	General purpose input	P5.x	P5DIDIS.Px = 0	Input Only
	Analog input channel ANx	ADC	P5DIDIS.Px = 1	
P5.10	General purpose input	P5.10	P5DIDIS.P10 = 0	Input Only
	Analog input channel AN10	ADC	P5DIDIS.P10 = 1	
	Timer 6 external up/down input, T6EUD	GPT12E	P5DIDIS.P10 = 0	

Table 7-11 Port 5 Functions (cont'd)

Port Pin	Pin Function	Associated Register/ Module	Alternate Function	Control Direction
P5.11	General purpose input	P5.11	P5DIDIS.P11 = 0	Input Only
	Analog input channel AN11	ADC	P5DIDIS.P11 = 1	
	Timer 5 external up/down input, T5EUD	GPT12E	P5DIDIS.P11 = 0	
P5.12	General purpose input	P5.12	P5DIDIS.P12 = 0	Input Only
	Analog input channel AN12	ADC	P5DIDIS.P12 = 1	
	Timer 6 count input, T6IN	GPT	P5DIDIS.P12 = 0	
P5.13	General purpose input	P5.13	P5DIDIS.P13 = 0	Input Only
	Analog input channel AN13	ADC	P5DIDIS.P13 = 1	
	Timer 5 count input, T5IN	GPT	P5DIDIS.P13 = 0	
P5.14	General purpose input	P5.14	P5DIDIS.P14 = 0	Input Only
	Analog input channel AN14	ADC	P5DIDIS.P14 = 1	
	Timer 4 external up/down input, T4EUD	GPT	P5DIDIS.P14 = 0	
P5.15	General purpose input	P5.15	P5DIDIS.P15 = 0	Input Only
	Analog input channel AN15	ADC	P5DIDIS.P15 = 1	
	Timer 2 external up/down input, T2EUD	GPT	P5DIDIS.P15 = 0	

The configuration of Port 5 is shown in **Figure 7-15**.

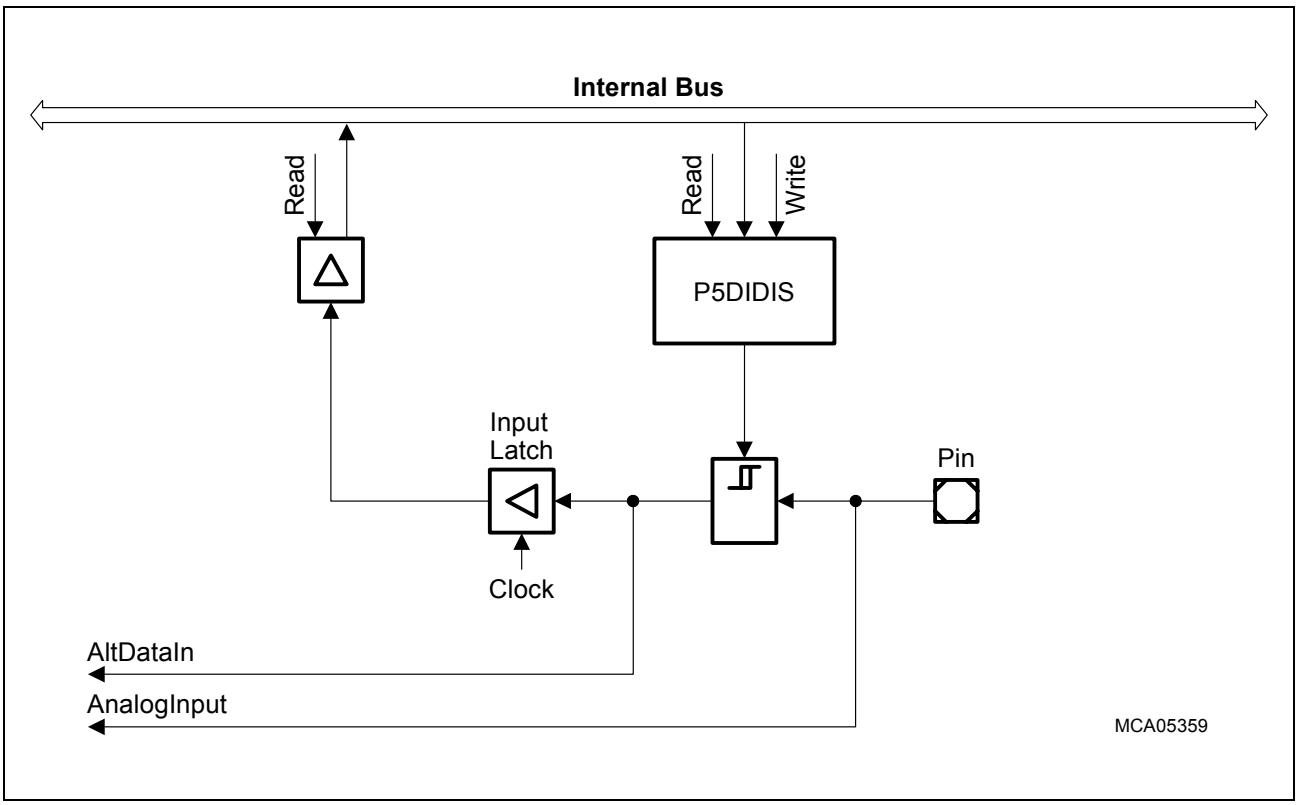


Figure 7-15 P5 Port Configuration

7.7 Port 9

If this 6-bit port is used for general purpose IO, the direction of each line can be configured via the corresponding direction register DP9. Each port line can be switched into push/pull or open drain mode via the open drain control register ODP9.

P9
Port 9 Data Register **SFR ($FF16_H/8B_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								-	-	P5	P4	P3	P2	P1	P0
-								-	-	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
P9.y	[5:0]	rwh	**Port Data Register P9 Bit y**

Note: Bits P9.0 - P9.5 are bit-protected for CAPCOM2 Output.

DP9
P9 Direction Ctrl. Register **SFR ($FF18_H/8C_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								-	-	P5	P4	P3	P2	P1	P0
-								-	-	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
DP9.y	[5:0]	rw	**Port Direction Register DP9 Bit y** 0 Port line P9.y is an input (high-impedance) 1 Port line P9.y is an output

ODP9
P9 Open Drain Ctrl. Reg. **SFR ($FF1A_H/8D_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								-	-	P5	P4	P3	P2	P1	P0
-								-	-	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
ODP9.y	[5:0]	rw	**Port 9 Open Drain Control Register Bit y** 0 Port line P9.y output driver in push/pull mode 1 Port line P9.y output driver in open drain mode

The alternate functions of the TwinCAN and CAPCOM2 modules are selected via the register ALTSEL0P9 and ALTSEL1P9.

ALTSEL0P9
P9 Alternate Select Reg. 0 **ESFR ($F138_H/9C_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								-	-	-	-	P3	-	P1	-
-								-	-	-	-	rw	-	rw	-

Field	Bits	Type	Description
ALTSEL0 P9.y	3, 1	rw	**P9 Alternate Select Register 0 Bit y** 0 associated peripheral output is not selected as alternate function 1 associated peripheral output is selected as alternate function

ALTSEL1P9
P9 Alternate Select Reg. 1 **ESFR ($F13A_H/9D_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								-	-	P5	P4	P3	P2	P1	P0
-								-	-	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
ALTSEL1 P9.y	[5:0]	rw	**P9 Alternate Select Register 1 Bit y** 0 associated peripheral output is not selected as alternate function 1 associated peripheral output is selected as alternate function

Alternate Functions of Port 9

Port 9 pins can be used as the transmitter and receiver lines of the TwinCAN or alternatively, the CAPCOM2 input/output lines.

Figure 7-16 shows the IO and alternate functions of Port 9.

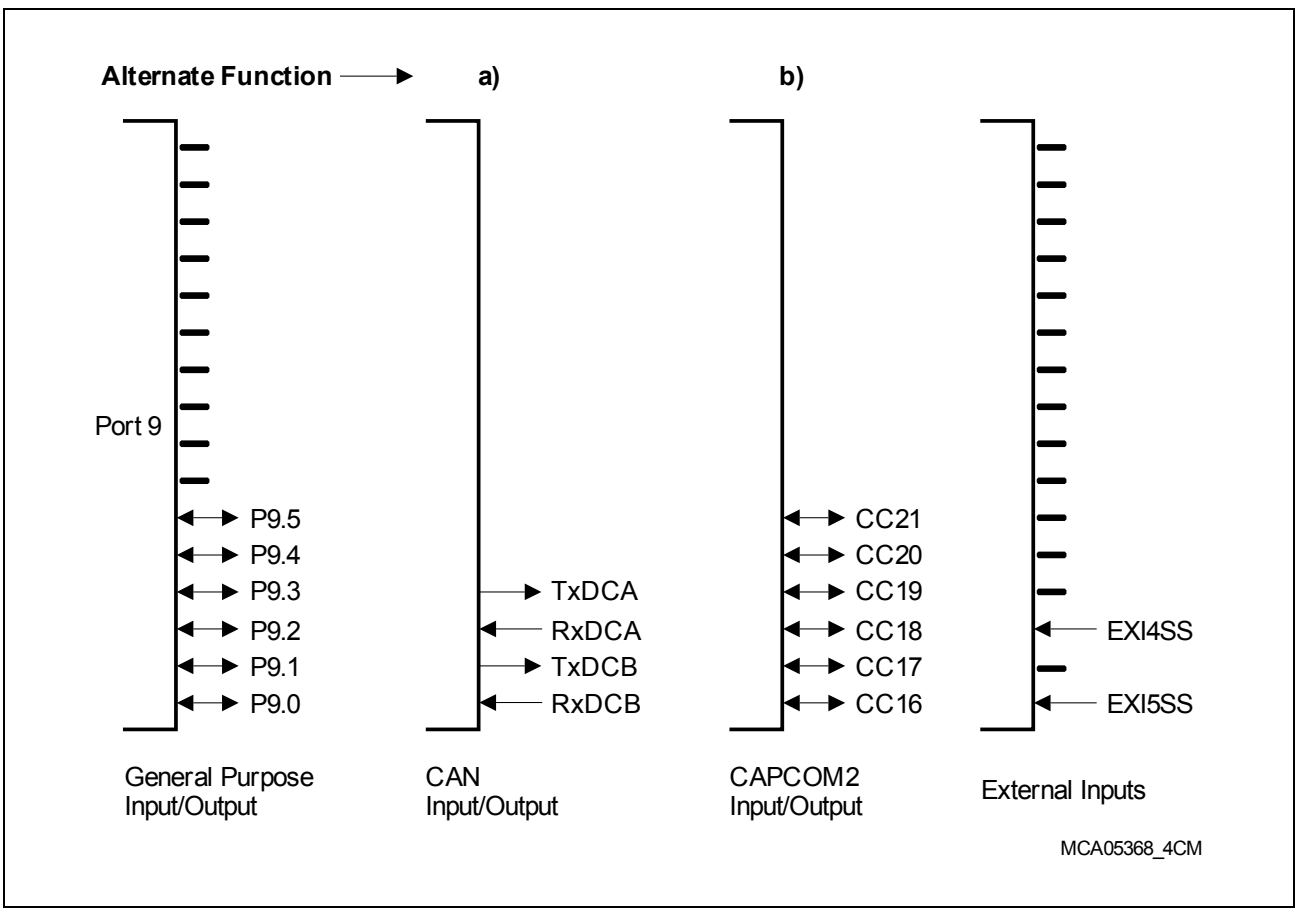


Figure 7-16 Port 9 IO and Alternate Functions

The functions of Port 9 pins are listed in **Table 7-12**.

Table 7-12 Port 9 Functions

Port Pin	Pin Function	Associated Register/ Module	Alternate Function	Control Direction
P9.0	General purpose input	P9.0	ALTSEL0P9.P0 = 0 and ALTSEL1P9.P0 = 0	DP9.P0 = 0
	General purpose output			DP9.P0 = 1
	CC16I Capture input	CAPCOM2	–	DP9.P0 = 0
	CC16O Compare output		ALTSEL0P9.P0 = 0 and ALTSEL1P9.P0 = 1	DP9.P0 = 1
	TwinCAN Receiver input, RxDCA, RxDCB	TwinCAN	–	DP9.P0 = 0
P9.1	General purpose input	P9.1	ALTSEL0P9.P1 = 0, and ALTSEL1P9.P1 = 0	DP9.P1 = 0
	General purpose output			DP9.P1 = 1
	CC17I Capture input	CAPCOM2	–	DP9.P1 = 0
	CC17O Compare output		ALTSEL0P9.P1 = 0, and ALTSEL1P9.P1 = 1	DP9.P1 = 1
	TwinCAN Transmitter output, TxDCB	TwinCAN	ALTSEL0P9.P1 = 1, and ALTSEL1P9.P1 = 1	DP9.P1 = 1

Table 7-12 Port 9 Functions (cont'd)

Port Pin	Pin Function	Associated Register/ Module	Alternate Function	Control Direction
P9.2	General purpose input	P9.2	ALTSEL0P9.P2 = 0 and ALTSEL1P9.P2 = 0	DP9.P2 = 0
	General purpose output			DP9.P2 = 1
	CC18I Capture input	CAPCOM2	–	DP9.P2 = 0
	CC18O Compare output		ALTSEL0P9.P2 = 0 and ALTSEL1P9.P2 = 1	DP9.P2 = 1
	TwinCAN Receiver input, RxDCA, RxDCB	TwinCAN	–	DP9.P2 = 0
P9.3	General purpose input	P9.3	ALTSEL0P9.P3 = 0 and ALTSEL1P9.P3 = 0	DP9.P3 = 0
	General purpose output			DP9.P3 = 1
	CC19I Capture input	CAPCOM2	–	DP9.P3 = 0
	CC19O Compare output		ALTSEL0P9.P3 = 0 and ALTSEL1P9.P3 = 1	DP9.P3 = 1
	TwinCAN Transmitter output, TxDCA	TwinCAN	ALTSEL0P9.P3 = 1 and ALTSEL1P9.P3 = 1	DP9.P3 = 1

Table 7-12 Port 9 Functions (cont'd)

Port Pin	Pin Function	Associated Register/ Module	Alternate Function	Control Direction
P9.4	General purpose input	P9.4	ALTSEL0P9.P4 = 0 and ALTSEL1P9.P4 = 0	DP9.P4 = 0
	General purpose output			DP9.P4 = 1
	CC20I Capture input	CAPCOM2	–	DP9.P4 = 0
	CC20O Compare output		ALTSEL0P9.P4 = 0 and ALTSEL1P9.P4 = 1	DP9.P4 = 1
P9.5	General purpose input	P9.5	ALTSEL0P9.P5 = 0 and ALTSEL1P9.P5 = 0	DP9.P5 = 0
	General purpose output			DP9.P5 = 1
	CC21I Capture input	CAPCOM2	–	DP9.P5 = 0
	CC21O Compare output		ALTSEL0P9.P5 = 0 and ALTSEL1P9.P5 = 1	DP9.P5 = 1

For a complete list of the external interrupt connections, see **Table 5-13**.

The configuration of Port 9 pins is shown in the subsequent figures.

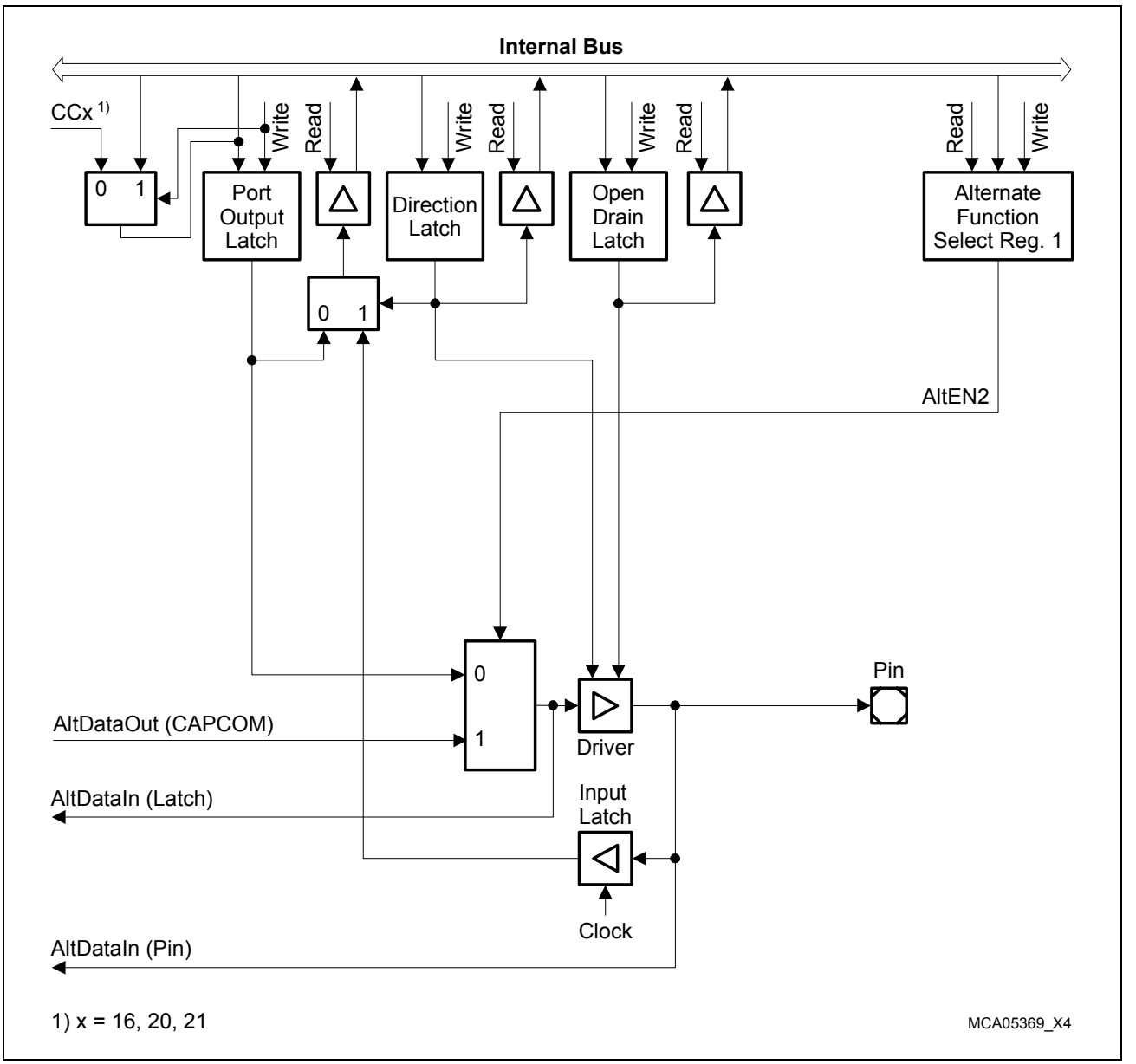


Figure 7-17 P9.0, P9.2, P9.4 and P9.5 Port Configuration

Table 7-13 P9.0, P9.2, P9.4 and P9.5 Alternate Function Control

Pins	Control Lines	Registers			Function
	AltEN2	DP9	ALTSEL1 P9	ALTSEL0 P9	
P9.0, P9.2, P9.4, P9.5	0	0 or 1	0	–	GPIO
	–	0	–		CAN input, CAPCOM2 input
	1	1	1		CAPCOM2 compare output

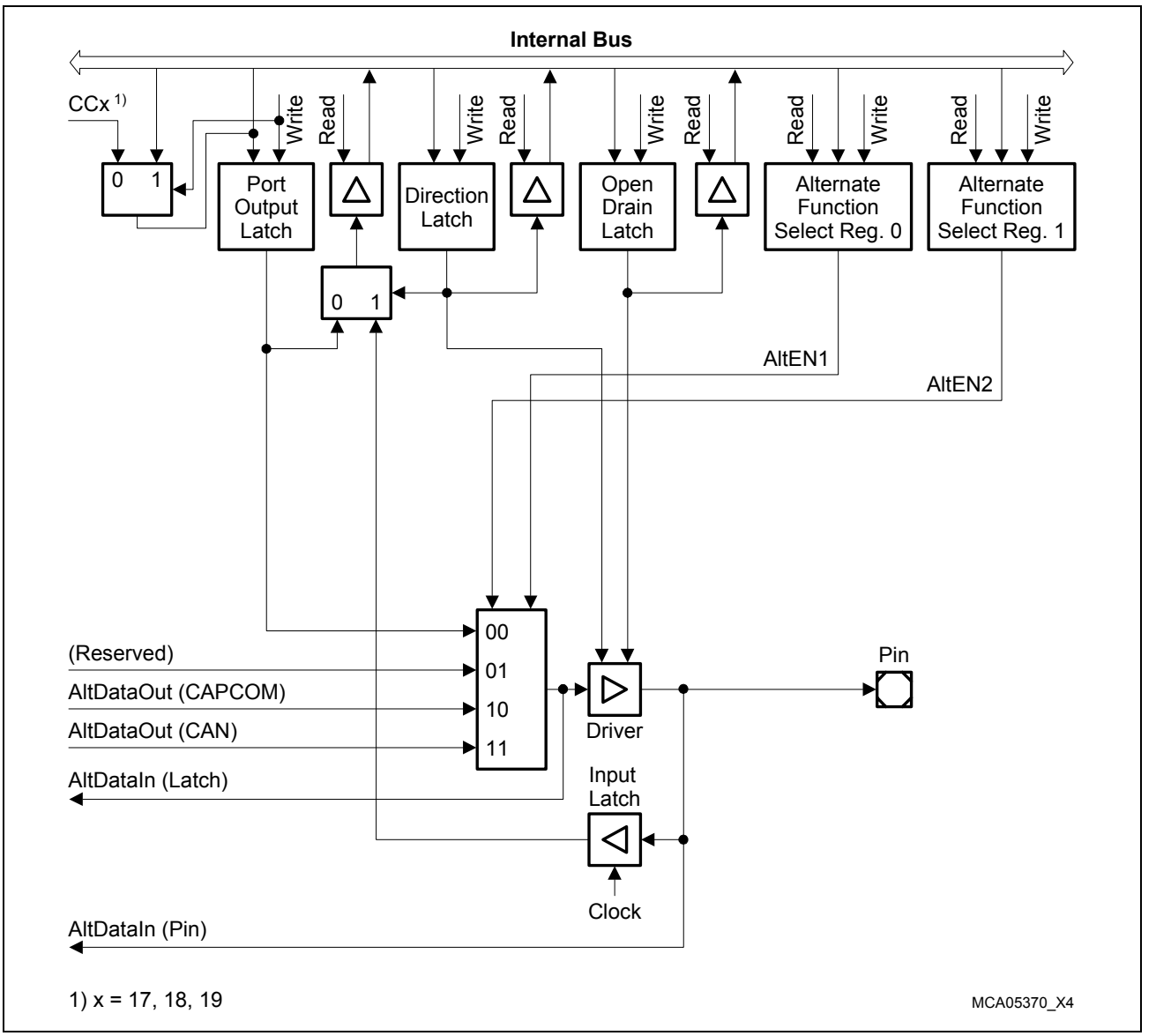


Figure 7-18 P9.1 and P9.3 Port Configuration

Table 7-14 P9.1 and P9.3 Alternate Function Control

Pins	Control Lines		Registers			Function
	AltEN		DP9	ALTSEL1 P9	ALTSEL0 P9	
	2	1				
P9.1, P9.3	0	0	0 or 1	0	0	GPIO
	–		0	–		CAN, CAPCOM2 input
	0	1	1	0	1	Reserved
	1	1	1	1	1	CAN output
	1	0	1	1	0	CAPCOM2 compare output

8 Dedicated Pins

Most of the input/output or control signals of the functional the XC164CM 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 17 dedicated pins of the XC164CM.

Table 8-1 XC164CM Dedicated Pins

Pin(s)	Function
NMI	Non-Maskable Interrupt Input
XTAL1, XTAL2	Oscillator Input/Output (main oscillator)
RSTIN	Reset Input
TRST	Test Reset Input for the Debug System
V_{AREF}, V_{AGND}	Power Supply for Analog/Digital Converter
V_{DDI}	Digital Power Supply for Internal Logic (2 pins)
V_{DDP}	Digital Power Supply for Port Drivers (4 pins)
V_{SS}	Digital Reference Ground (4 pins)

The Non-Maskable Interrupt Input $\overline{\text{NMI}}$ allows to trigger a high priority trap via an external signal (e.g. a power-fail signal). It also serves to validate the PWRDN instruction that switches the XC164CM into Power-Down mode. The $\overline{\text{NMI}}$ pin is sampled with every system clock cycle to detect transitions.

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 (see **Section 6.2.1**) 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 the basic operating clock signal of the XC164CM.

An external clock signal may be fed to the input XTAL1, leaving XTAL2 open.

The Reset Input $\overline{\text{RSTIN}}$ allows to put the XC164CM into the well defined reset condition either at power-up or external events like a hardware failure or manual reset.

The reset configuration is described in **Section 6.1.4**.

The Test Reset Input $\overline{\text{TRST}}$ puts the XC164CM's debug system into reset state. During normal operation this input should be held active. For debugging purposes the on-chip debugging system can be enabled by releasing pin $\overline{\text{TRST}}$.

The Power Supply pins for the Analog/Digital Converter V_{AREF} and V_{AGND} provide a separate power supply (reference voltage) for the on-chip ADC. 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}.

The Power Supply pins V_{DDI}/V_{DDP} and V_{SS} provide the power supply for the digital logic of the XC164CM. The respective V_{DD}/V_{SS} pairs should be decoupled as close to the pins as possible. The V_{DDI} pins (2.5 V) supply the internal logic blocks of the XC164CM, while the V_{DDP} pins (5.0 V) supply the output port drivers.

Note: All V_{DD} pins and all V_{SS} pins must be connected to the power supplies and ground, respectively.

9 The LXBus Controller (EBC)

The XC164CM External Bus Controller (EBC) in the XC164CM allows access only to the internal LXBus module TwinCAN. The LXBus is an internal representation of the external bus and it controls accesses to integrated peripherals and modules.

Note: The XC164CM's EBC does not provide access to external components.

The function of the EBC is controlled via a set of configuration registers.

The Function CONtrol register FCONCS7 register specifies the LXBus cycles in terms of address (multiplexed/demultiplexed), data (16-bit/8-bit), and chip-select enable. The timing of the bus access is controlled by the Timing CONfiguration register TCONCS7, which specifies 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 ADDRSEL7.

The register set (FCONCS7/TCONCS7/ADDRSEL7) defines a programmable "address window", where Chip Select signal $\overline{CS7}$ is used for access to the internal TwinCAN module on LXBus.

9.1 Timing Principles

9.1.1 Basic Bus Cycle Protocols

The bus timing is defined by six different timing phases (A-F). These phases define all control signals needed for any access sequence. 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. Each phase can occupy a programmable number of clock cycles.

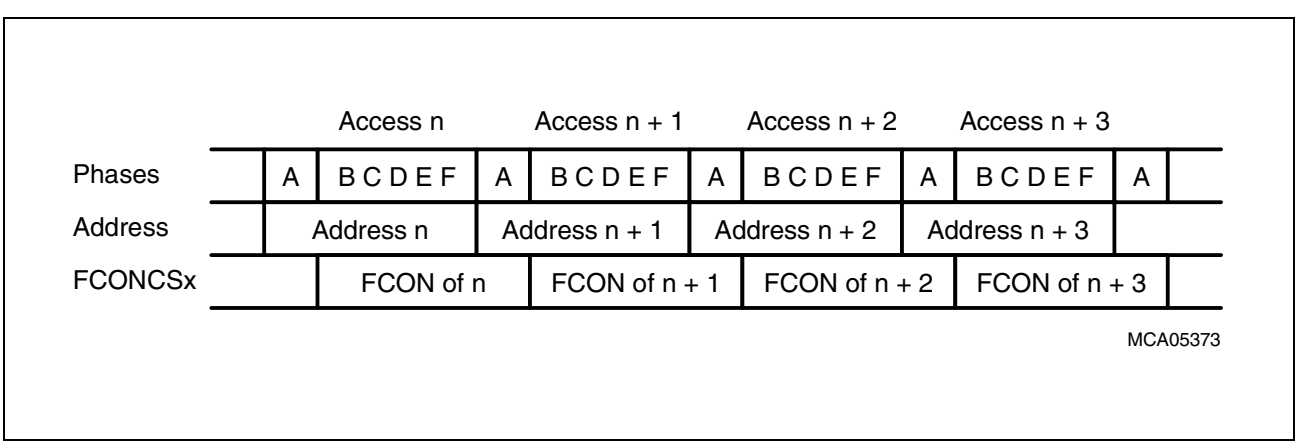


Figure 9-1 Phases of a Sequence of Several Accesses

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.

9.1.2 Bus Cycle Examples: Fastest Access Cycles

The TwinCAN uses the fastest demultiplexed bus cycles, controlled by the READY signal.

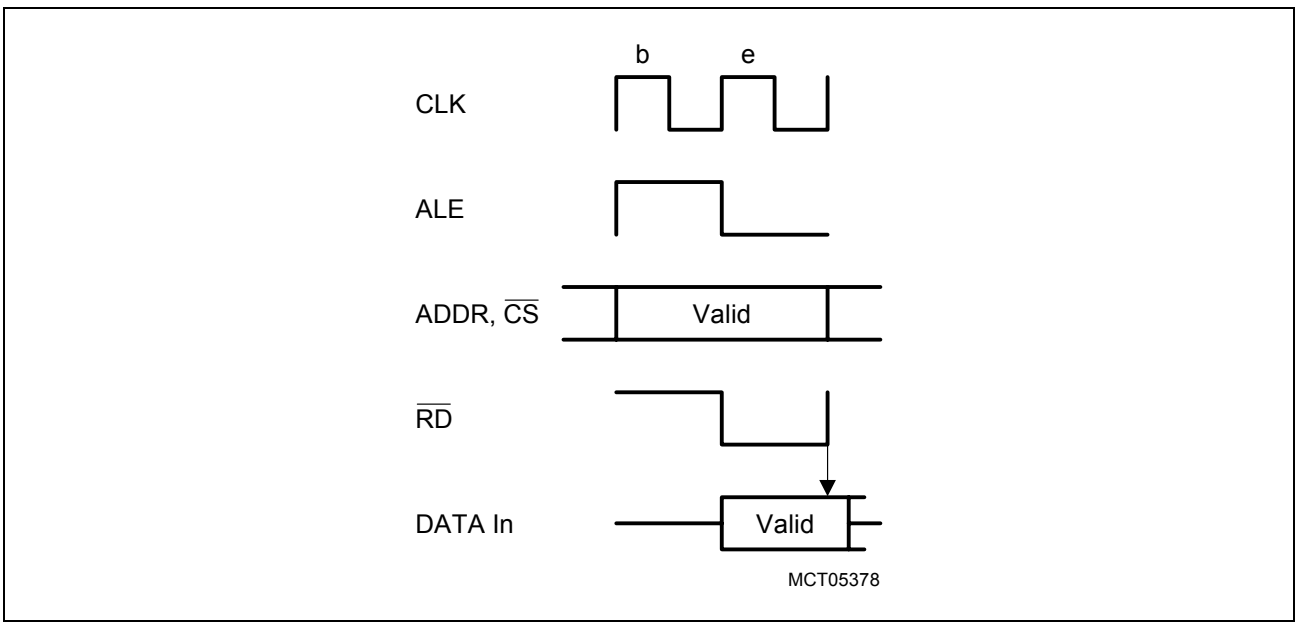


Figure 9-2 Fastest Read Cycle Demultiplexed Bus

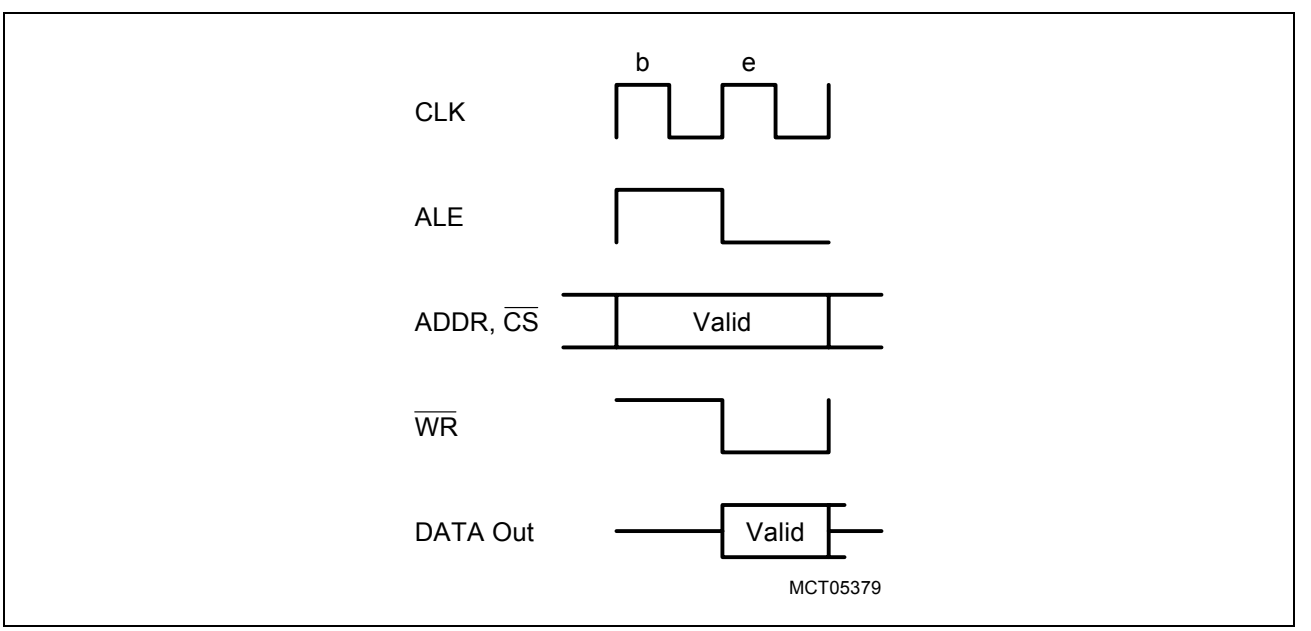


Figure 9-3 Fastest Write Cycle Demultiplexed Bus

9.2 Functional Description

9.2.1 Configuration Register Overview

The XC164CM's EBC is compatible with other members of the XC166 Family. A 128-byte address space is occupied/reserved by the EBC.

Note: All EBC registers are write-protected by the EINIT protection mechanism. Thus, after execution of the EINIT instruction, these registers are not writable any more.

9.2.2 The Timing Configuration Register TCONCS7

The timing control register is used to program the described cycle timing for the different access phases. The timing control register may be reprogrammed during an access from the affected address window. The new settings are first valid for the next access.

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	
-	rw		rw		rw					rw	rw		rw	rw	

Note: TCONCS7 belongs to chip select $\overline{CS7}$ which is used and defined for internal access to the LXBus peripheral TwinCAN.

Field	Bits	Type	Description
WRPHF	[14:13]	rw	**Write Phase F** 00 0 clock cycles … … 11 3 clock cycles
RDPHF	[12:11]	rw	**Read Phase F** 00 0 clock cycles … … 11 3 clock cycles
PHE	[10:6]	rw	**Phase E** 00000: 1 clock cycle … … 11111: 32 clock cycles

Field	Bits	Type	Description
PHD	5	rw	**Phase D** 0 0 clock cycles 1 1 clock cycle
PHC	[4:3]	rw	**Phase C** 00 0 clock cycles … … 11 3 clock cycles
PHB	2	rw	**Phase B** 0 1 clock cycle 1 2 clock cycles
PHA	[1:0]	rw	**Phase A** 00 0 clock cycles … … 11 3 clock cycles

9.2.3 The Function Configuration Register FCONCS7

The Function Control register is used to control the bus 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 $\overline{CS7}$) is generally enabled or not.

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
-	-	-	-	-	-	-	-	-	-	rw		-	rw	rw	rw

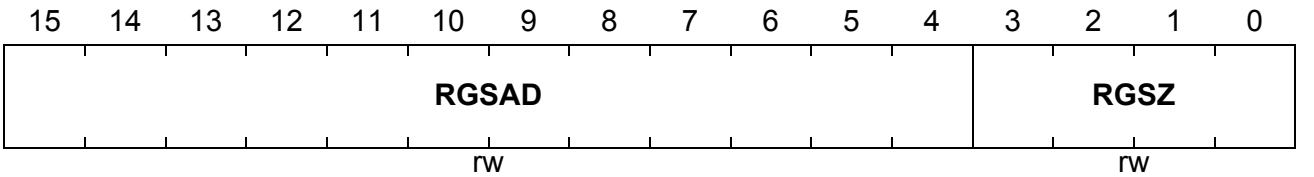
Note: FCONCS7 belongs to chip select $\overline{CS7}$ which is used and defined for internal access to the LXBus peripheral TwinCAN.

Field	Bits	Type	Description
BTYP	[5:4]	rw	**Bus Type Selection** 00 8 bit Demultiplexed 01 8 bit Multiplexed 10 16 bit Demultiplexed 11 16 bit Multiplexed
RDYMOD	2	rw	**Ready Mode** 0 Asynchronous READY 1 Synchronous READY
RDYEN	1	rw	**Ready Enable** 0 Access time is controlled by bitfield PHEx 1 Access time is controlled by bitfield PHEx and READY signal
ENCS	0	rw	**Enable Chip Select** 0 Disable 1 Enable

Note: With ENCS7 the chip select $\overline{CS7}$ and its related register set is enabled and defined for internal access to the LXBus peripheral TwinCAN.

9.2.4 The Address Window Selection Register ADDRSEL7

ADDRSEL7
Address Range/Size for $\overline{CS7}$ **XSFR ($EE4E_H$/--)** **Reset Value: 2000_H**



Field	Bits	Type	Description
RGSAD	[15:4]	rw	Address Range Start Address Selection
RGSZ	[3:0]	rw	Address Range Size Selection (see **Table 9-1**)

Definition of Address Areas

The enabled register set FCONCS7/TCONCS7/ADDRSEL7 defines a separate address area within the address space of the XC164CM. Within this address area the conditions of LXBus accesses can be controlled separately, whereby the address area (window) is defined by the ADDRSEL7 register. 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 9-1**). The size of the window chosen by ADDRSEL7.RGSZ defines the relevant bits of ADDRSEL7.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 ADDRSEL7.RGSAD (marked ‘x’) are disregarded.

Table 9-1 Address Range and Size for ADDRSEL7

ADDRSEL7		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	---- ---- ---- ---- ---- ----

*Note: The range start address can only be on boundaries specified by the selected range size according to **Table 9-1**.*

9.2.5 Access Control to TwinCAN

Access control to LXBus is required for accesses to the TwinCAN module.

For accesses to the TwinCAN, $\overline{CS7}$ and its control registers, the ADDRSEL7, TCONCS7 and the FCONCS7 are used. The selection of LXBus is controlled with $\overline{CS7}$. The address range, defined in ADDRSEL7, is recommended to be located in the 'External IO Range' (range from $20'0000_H$ to $3F'0000_H$). Only for the External IO Range of the total external address range it is guaranteed that a read access is executed **after** a preceding write access.

After reset (controlled by the startup program sequence), the TwinCAN address range is adjusted per default to the area from address $20'0000_H$ to $20'0FFF_H$ (4 KB), resulting in the ADDRSEL7 default-code of 2000_H. This initial value of ADDRSEL7 may be changed afterwards by the user.

The initial default value of the bus function control register FCONCS7 is selected according to the requirements of the TwinCAN: 16-bit demultiplexed bus, access time controlled with synchronous READY. This function control is represented by the default value for FCONCS7 of 0027_H.

The initial LXBus cycle timing as controlled with register TCONCS7 after reset is the shortest possible timing using two clock cycles for one bus cycle. But this minimum timing will be lengthened with waitstate(s) controlled by the TwinCAN itself with the READY function. This timing control is controlled by the reset value of TCONCS7 (0000_H).

Attention: Although the TwinCAN's register set is programmable, it is strongly recommended not to change the default values, except for ADDRSEL7, because the TwinCAN is an on-chip peripheral.

9.2.6 Shutdown Control

In case of a shutdown request from the SCU it must be insured by the EBC that all the different functions of the EBC are in a non-active state before the whole chip is switched in an Idle, Powerdown, Sleep or Software Reset mode. A running bus cycle is finished, still requested bus cycles are executed. 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.

Shutdown control in EBC:

- Finish all pending cycle requests.
- Send shutdown acknowledge with the control of the bus.

9.3 EBC Register Table

Table 9-2 lists all EBC Configuration Registers which are implemented in the XC164CM ordered by their physical address. The registers are all located in the XSFR space (internal IO space).

Table 9-2 EBC Memory Table (ordered by physical address)

Name	Physic. Addr.	Description	Reset Value[1)]
reserved	$EE00_H$ - $EE46_H$	reserved - do not use	–
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	2000_H
reserved	$EE50_H$ - $EEFF_H$	reserved - do not use	–

1) **NOTE:** For enabling future enhancements without any compatibility problems, reserved (and not listed) addresses should neither be written nor be used as read value by the software.

10 The Bootstrap Loader

The built-in bootstrap loader of the XC164CM provides a mechanism to load the startup program, which is executed after reset, via the serial interface. In this case no external memory or an internal Flash/ROM/OTP is required for the initialization code.

The bootstrap loader moves code/data into the internal PSRAM. Flash/ROM memory is not necessary. However, it may be used to provide lookup tables or may provide “core-code”, i.e. a set of general purpose subroutines, e.g. for IO operations, number crunching, system initialization, etc.

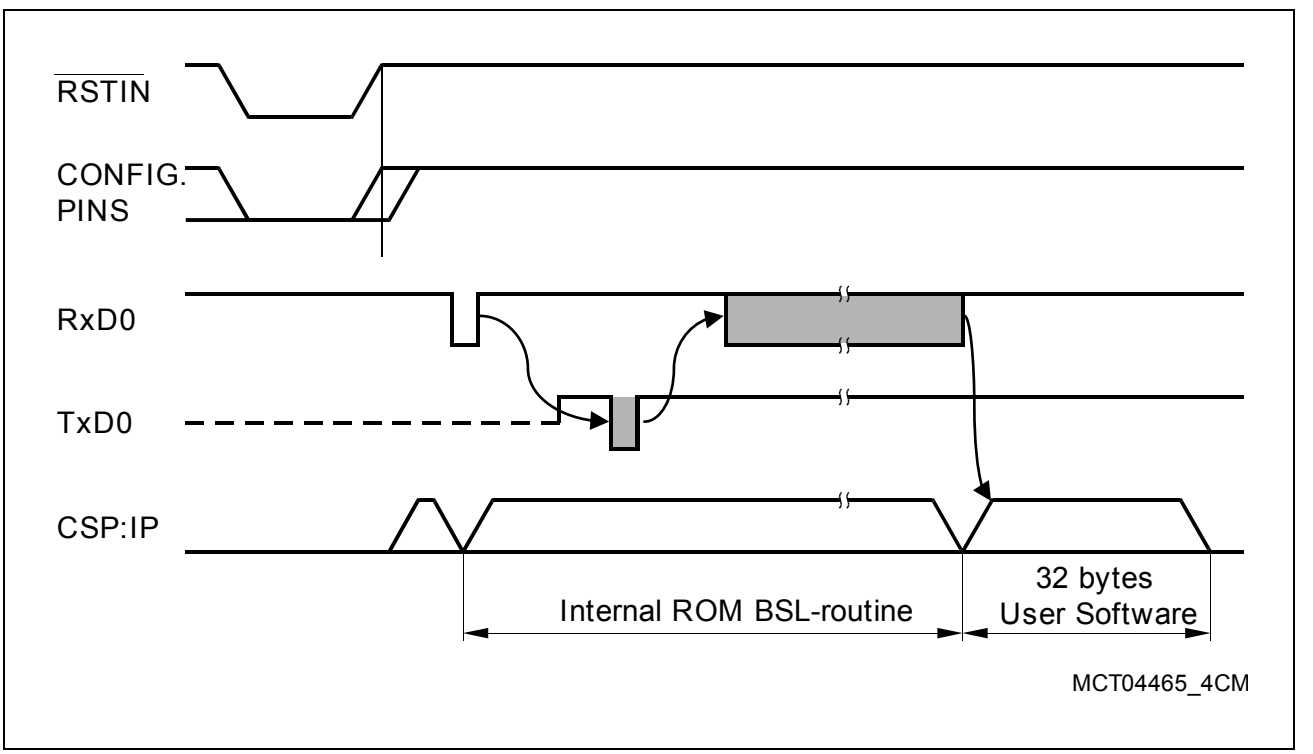


Figure 10-1 Bootstrap Loader Sequence

The Bootstrap Loader may be used to load the complete application software into ROMless systems, it may load temporary software into complete systems for testing or calibration, it may also be used to load a programming routine for Flash devices.

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.

10.1 Entering the Bootstrap Loader

The XC164CM enters BSL mode triggered by external configuration during a hardware reset:

- when pins P9.5 = 0, P9.4 = 1, and $\overline{\text{TRST}}$ = 1 at the end of an internal reset.

The bootstrap loader code is stored in a special Boot-ROM, no part of the Flash memory area is required for this.

The hardware that activates the BSL during reset may be a simple pull-down resistor for systems that use this feature upon every hardware reset. You may want to use a switchable solution (via jumper or an external signal) for systems that only temporarily use the bootstrap loader.

The ASC0 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.

*Note: The proper reset configuration for BSL mode requires a set of pins to be driven to defined logic levels (see **Section 6.1.4**).*

Initial State in BSL Mode

After entering BSL mode and the respective initialization the XC164CM scans the RxD0 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 ASC0 accordingly and switches pin TxD0 to output. Using this baudrate, an identification byte is returned to the host that provides the loaded data.

This 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, inclusive XC16x

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.

When the XC164CM has entered BSL mode, the following configuration is automatically set:

Register/Bitfield	Reset Value
Watchdog Timer	Disabled
P3.10/TxD0	‘1’
DP3.10	‘1’
ALTSEL0P3.10	‘1’
ASC0_BG	$XXXX_H$
ASC0_CON	8811_H
GPT12E_T6CON	0880_H
GPT12E_T6	$XXXX_H$
PLLCON	2700_H

Other than after a normal reset the watchdog timer is disabled, so the bootstrap loading sequence is not time limited. Pin TxD0 is configured as output, so the XC164CM can return the identification byte.

10.2 Loading the Startup Code

After sending the identification byte the BSL enters a loop to receive 32 Bytes via ASC0. These bytes are stored sequentially into locations $E0'0004_H$ through $E0'0023_H$ of the internal PSRAM. So up to 16 instructions may be placed into the PSRAM area. The first two words of the PSRAM are loaded with the DISWDT instruction. To execute the loaded code the BSL then points register VECSEG to location $E0'0000_H$, i.e. the first loaded instruction[1]. The bootstrap loading sequence terminates by executing a software reset. Most probably the initially loaded routine will load additional code or data, as an average application is likely to require substantially more than 16 instructions. This second receive loop may directly use the pre-initialized interface ASC0 to receive data and store it to arbitrary user-defined locations.

This second level of loaded code may be the final application code. It may also be 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.

This process may go through several iterations or may directly execute the final application.

Note: Data fetches from a protected Flash/ROM will not be executed.

10.3 Exiting Bootstrap Loader Mode

After the bootstrap loader has been activated, the watchdog timer and the debug system are disabled. The debug system is released automatically when the BSL terminates after having received the 32nd byte from the host. In order to activate the watchdog timer, if required, it must be enabled via instruction ENWDT (before executing the EINIT instruction). Also a reset will re-enable the WDT:

- a software reset (ignoring the external configuration)
- a hardware reset, not configuring BSL mode

After the (non-BSL) reset the XC164CM will start executing out of user memory as externally configured via Port 9, Port 1H, and $\overline{\text{TRST}}$.

1) This includes the execution of the initial DISWDT instruction, ensuring that the 2nd level loader is not aborted by the watchdog timer.

10.4 Choosing the Baudrate for the BSL

The calculation of the serial baudrate for ASC0 from the length of the first zero byte that is received, allows the operation of the bootstrap loader of the XC164CM with a wide range of baudrates. However, the upper and lower limits have to be kept, in order to ensure proper data transfer.

$$B_{MC} = \frac{f_{SYS}}{32 \times (ASC0BG + 1)} \quad (10.1)$$

The XC164CM uses timer GPT12E_T6 to measure the length of the initial zero byte. The quantization uncertainty of this measurement implies the first deviation from the real baudrate, the next deviation is implied by the computation of the ASC0_BG reload value from the timer contents. **Equation (10.2)** shows the association:

$$ASC0BG = \frac{T6 - 36}{72} \qquad T6 = \frac{9}{4} \times \frac{f_{SYS}}{B_{Host}} \quad (10.2)$$

For a correct data transfer from the host to the XC164CM the maximum deviation between the internal initialized baudrate for ASC0 and the real baudrate of the host should be below 2.5%. The deviation (F_B, in percent) between host baudrate and XC164CM baudrate can be calculated via **Equation (10.3)**:

$$F_B = \left| \frac{B_{Contr} - B_{Host}}{B_{Contr}} \right| \times 100\% \qquad F_B \leq 2.5\% \quad (10.3)$$

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 CPU 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 10-2**).

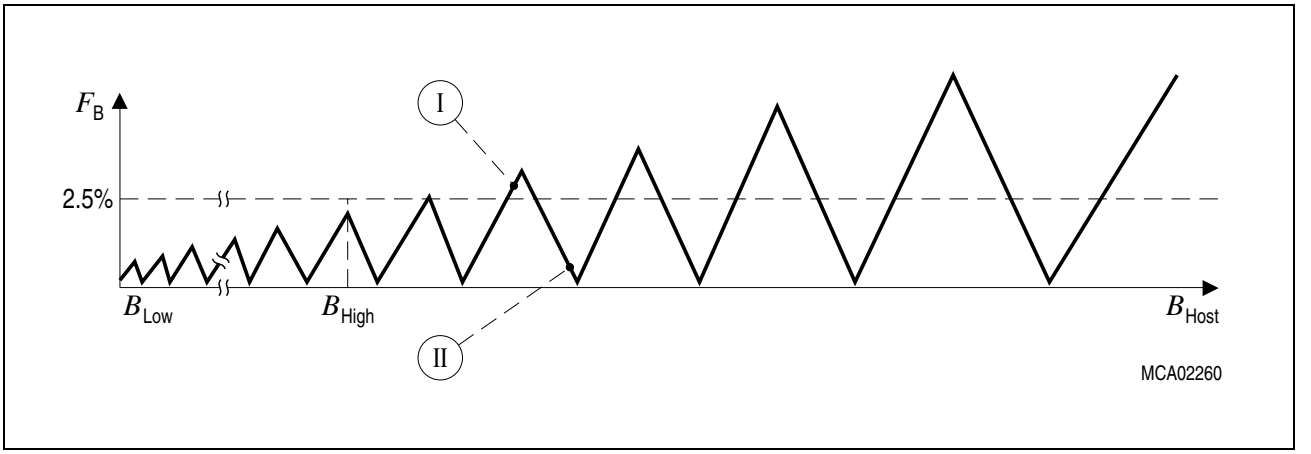


Figure 10-2 Baudrate Deviation between Host and XC164CM

The minimum baudrate (B_{Low} in **Figure 10-2**) is determined by the maximum count capacity of timer GPT12E_T6, when measuring the zero byte, i.e. it depends on the system clock. The minimum baudrate is obtained by using the maximum GPT12E_T6 count 2^{16} in the baudrate formula. Baudrates below B_{Low} would cause GPT12E_T6 to overflow. In this case ASC0 cannot be initialized properly and the communication with the external host is likely to fail.

The maximum baudrate (B_{High} in **Figure 10-2**) 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 10-2**) may e.g. violate the deviation limit, while an even higher baudrate (marked II) in **Figure 10-2**) 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 the ASC0
- the external host is able to use this baudrate
- the computed deviation error is below the limit

Table 10-1 Bootstrap Loader Baudrate Ranges

f_{SYS} [MHz]	**10**	**12**	**16**	**20**	**25**	**33**
B_{MAX}	312,500	375,000	500,000	625,000	781,250	1,031,250
B_{High}	9,600	19,200	19,200	19,200	38,400	38,400
B_{STDmin}	600	600	600	1,200	1,200	1,200
B_{Low}	344	412	550	687	859	1,133

Note: When the bootstrap loader mode is entered, the default configuration selects direct drive mode for clock generation. In this case the bootstrap loader will begin to operate with $f_{SYS} = f_{OSC}$ which will limit the maximum baudrate for ASC0 at low input frequencies intended for PLL operation.
Higher levels of the bootstrapping sequence can then switch the clock generation mode (via register PLLCON) e.g. to PLL in order to achieve higher baudrates for the download.

11 Debug System

11.1 Introduction

The XC164CM includes an On-Chip Debug Support (OCDS) system, which provides convenient debugging, controlled directly by an external device via debug interface pins.

On-Chip Debug Support (OCDS)

The OCDS system supports a broad range of debug features including setting up breakpoints and tracing memory locations. Typical application of OCDS is to debug the user software running on the XC164CM in the customer's system environment.

The OCDS system is controlled by an external debugging device via the **Debug Interface**, including an independent JTAG interface and a break interface (**Figure 11-1**). The debugger manages the debugging tasks through a set of OCDS registers accessible via the JTAG interface, and through a set of special debug IO instructions. Additionally, the OCDS system can be controlled by the CPU, e.g. by the monitor program. The OCDS system interacts with the core through an injection interface to allow execution of Cerberus-generated instructions, and through a break port.

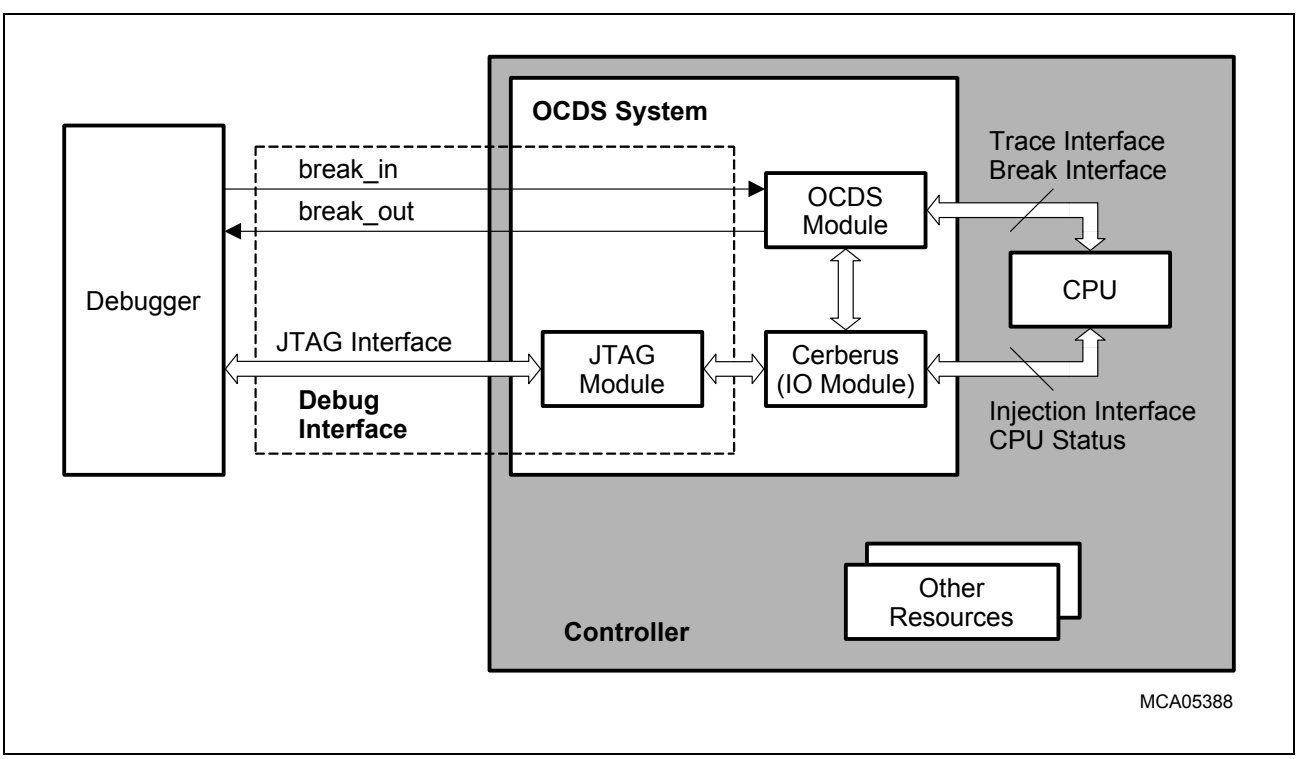


Figure 11-1 OCDS Overall Structure

The OCDS system functions are represented and controlled by the **Debug Interface**, the **OCDS Module** and by the debug IO control module (**Cerberus**) which provides all the

functionality necessary to interact between the debug interface (the external debugger) and the internal system.

The OCDS system provides the following basic 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
- Analysis and status registers

11.2 Debug Interface

The **Debug Interface** is a channel to access XC164CM On-Chip Debug Support (OCDS) resources. Through it data can be transferred to/from all on- and off-chip (if any) memories and control registers.

Features and Functions

- Independent interface for On-Chip Debug Support (OCDS)
- JTAG port based on the IEEE 1149 JTAG standard
- Break interface for external trigger and indication of breaks
- Generic memory access functionality
- Independent data transfer channel for e.g. programming of on-chip non volatile memory

The Debug Interface is represented by:

- Standard **JTAG Interface**
- Two additional XC164CM specific signals - **OCDS Break-Interface**

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-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 XC164CM **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, while not interrupting the CPU and using its pin(s).

11.3 OCDS Module

The application of **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
- IP 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

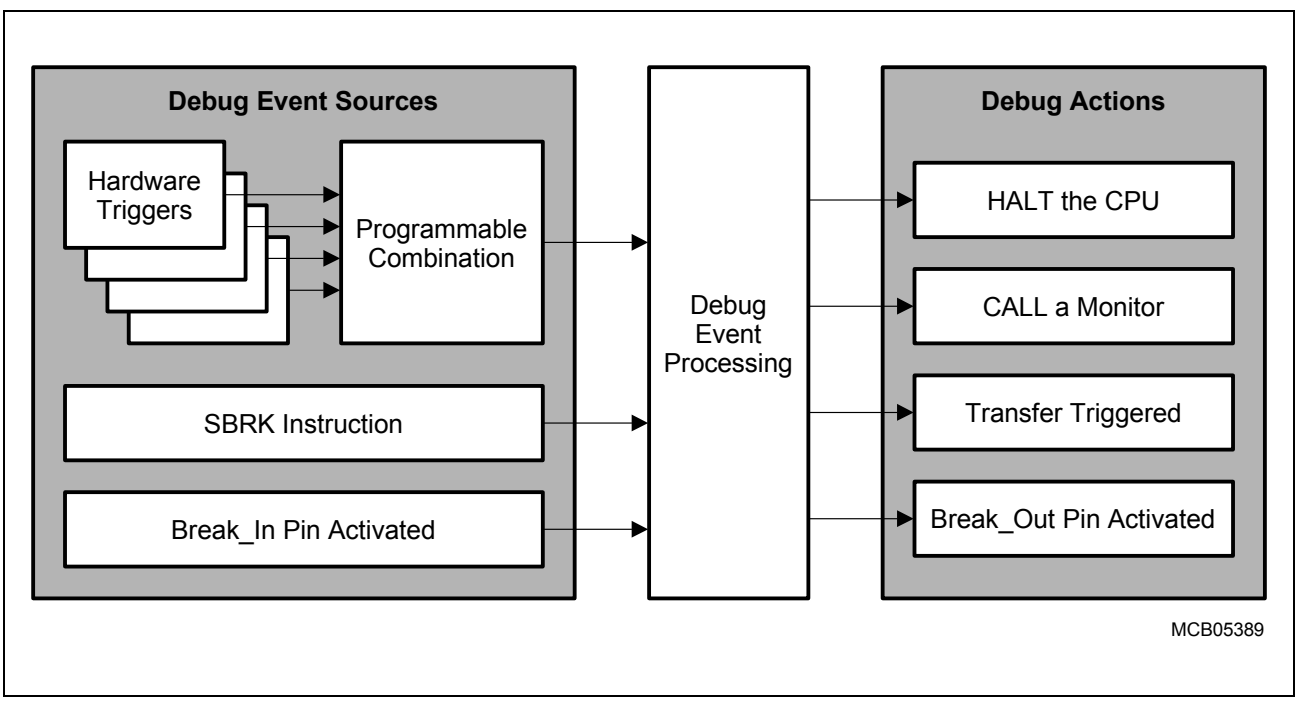


Figure 11-2 OCDS Concept: Block Diagram

11.3.1 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 $8C00_H$. 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.

11.3.2 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 XC164CM 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.

11.4 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 XC164CM.

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
- 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.

11.4.1 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.

12 Instruction Set Summary

This chapter briefly summarizes the XC164CM's instructions ordered by instruction classes. This provides a basic understanding of the XC164CM'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 XC166 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 XC164CM.

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 XC164CM via software:	SRST	–
Entering the Idle mode or Sleep mode:	IDLE	–
Entering the Power Down mode:	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 (disables the effect of any later execution of a DISWDT instruction in WDT compatibility mode):	EINIT	–

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 XC164CM 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 XC164CM 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.

Figure 13-1 shows the pin diagram of the XC164CM. It shows the location of the different supply and IO pins. A detailed description of all the pins is also found in the 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.*

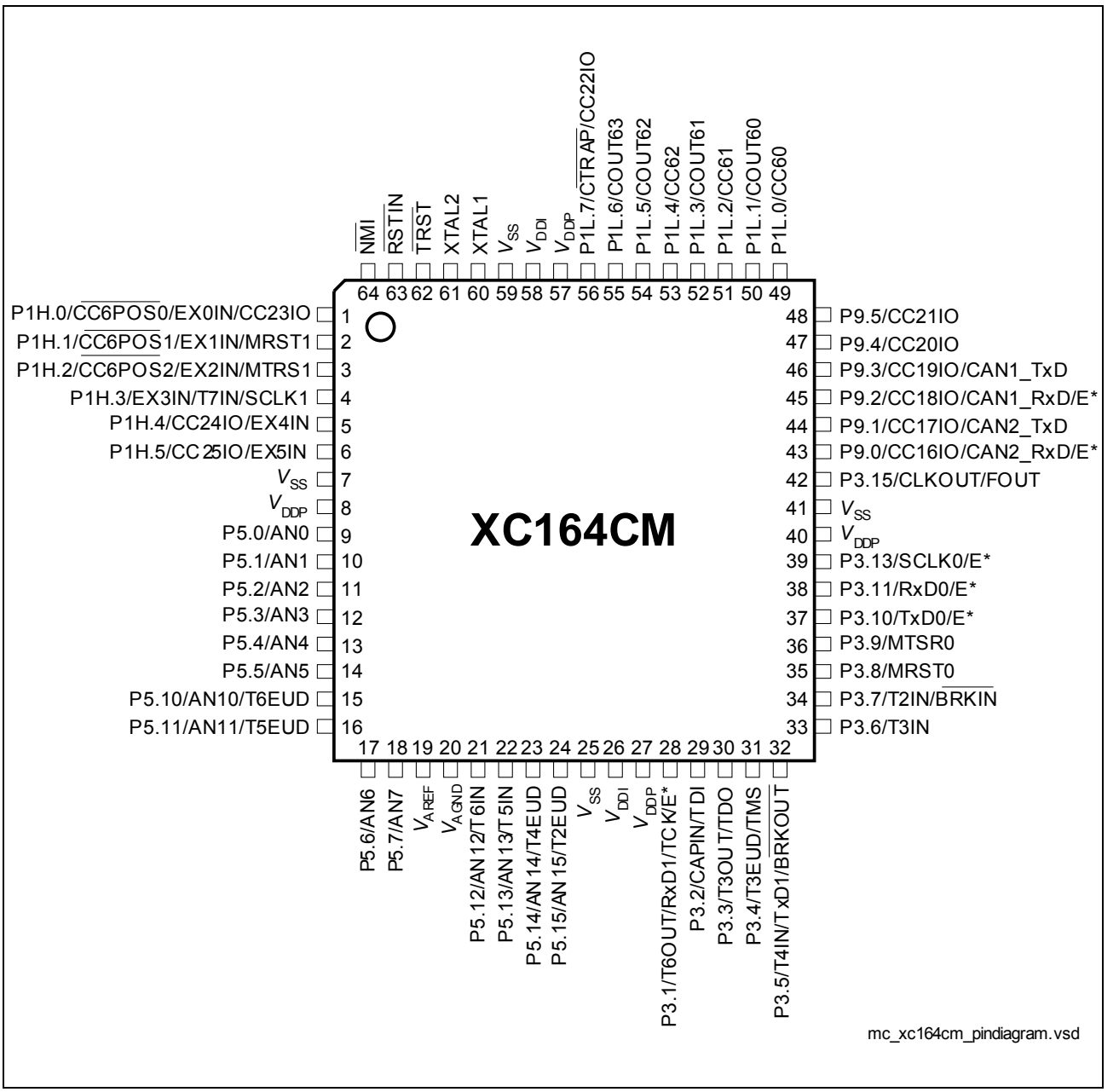


Figure 13-1 Pin Configuration P-TQFP-144 Package (top view)

Note: Fast External Interrupt Inputs are available at the indicated pins (E).*

Keyword Index

This section lists a number of keywords which refer to specific details of the XC164CM in terms of its architecture, its functional units or functions. This helps to quickly find the answer to specific questions about the XC164CM.

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

D

E

F

G

H

I

Q

R

S

T

V

W

Z

w w w . i n f i n e o n . c o m

Published by Infineon Technologies AG