User's Manual, V3.1, Feb. 2002

C164CI/CL
C164SI/SL

16-Bit Single-Chip Microcontroller

Microcontrollers

Infineon technologies

Never stop thinking.

Edition 2002-02

Published by Infineon Technologies AG,
St.-Martin-Strasse 53,
D-81541 München, Germany

© Infineon Technologies AG 2002.
All Rights Reserved.

Attention please!

The information herein is given to describe certain components and shall not be considered as warranted characteristics.

Terms of delivery and rights to technical change reserved.

We hereby disclaim any and all warranties, including but not limited to warranties of non-infringement, regarding circuits, descriptions and charts stated herein.

Infineon Technologies is an approved CECC manufacturer.

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office in Germany or our Infineon Technologies Representatives worldwide.

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.

C164CI/CL
C164SI/SL

16-Bit Single-Chip Microcontroller

Microcontrollers

Never stop thinking.

C164CI
Revision History: V3.1, 2002-02

Previous Version: V3.0, 2002-01 (intermediate version)
V2.0, 1999-09
V1.1, 1998-08
V1.0, 11.97 (last printed version)

Page	Subjects (major changes from V2.0, 1999-09 to V3.0, 2002-01)
all	Converted to new company layout, figures have been redrawn
2-9	Description of EEPROM removed
3-9	Description of external XRAM access removed
5-3	Interrupt nodes 52 … 55, 60, 65, 68 … 70 removed
5-6	Description added for Interrupt Enable Control Bit
6-1ff	Minor improvements in the description
7-4ff	Description of Port “Output Driver Control” reworked
8-2	V_{PP} function added to pin $\overline{EA}/V_{PP}$
9-28	XBUS interface description improved
10-6	GPT timing tables improved
11-12ff	Description of ASC0 baudrate generation improved
12-13	Description of SSC baudrate generation improved
13-5	Table “Watchdog Time Ranges” improved
15-1ff	Chapter “Bootstrap Loader” reworked
16-1, 16-23	Surplus interrupt control registers removed
16-7	More frequency tables added
17-2, 17-18ff	Trap functionality defined for reduced CAPCOM6 version
17-4	Section “Clocking Scheme” removed
17-16	Block commutation sequence corrected
17-18	Description of Trap function improved
23-4, 23-11	Register description marks improved, surplus interrupt control registers removed

Previous Version: V3.0, 2002-01 (intermediate version)
V2.0, 1999-09
V1.1, 1998-08
V1.0, 11.97 (last printed version)

Page	Subjects (major changes from V3.0, 2002-01 to V3.1, 2002-02)[1)]
Several	Typos corrected
5-2	Number of interrupt nodes corrected
19-36	Figure corrected
23-4ff	Register XP1IC removed
24-1	Page header corrected

1) No functional changes were incorporated here. V3.1 was introduced to correct some errors and to improve the layout for printing.

Note: This revision history does not list changes beyond revision V2.0.
This also excludes V1.0, the last printed version.

Controller Area Network (CAN): License of Robert Bosch GmbH

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

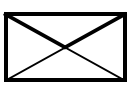


Table of Contents **Page**

1	**Introduction**	1-1
1.1	Members of the 16-bit Microcontroller Family	1-3
1.2	Summary of Basic Features	1-5
1.3	Abbreviations	1-8
2	**Architectural Overview**	2-1
2.1	Basic CPU Concepts and Optimizations	2-2
2.1.1	High Instruction Bandwidth / Fast Execution	2-3
2.1.2	Programmable Multiple Priority Interrupt System	2-7
2.2	On-Chip System Resources	2-8
2.3	On-Chip Peripheral Blocks	2-11
2.4	Power Management Features	2-18
2.5	Protected Bits	2-20
3	**Memory Organization**	3-1
3.1	Internal ROM Area	3-3
3.2	Internal RAM and SFR Area	3-4
3.3	The On-Chip XRAM	3-9
3.4	External Memory Space	3-10
3.5	Crossing Memory Boundaries	3-11
3.6	Protection of the On-Chip Mask ROM	3-12
3.7	OTP Memory Programming	3-13
3.7.1	Selecting an OTP Programming Mode	3-15
3.7.2	OTP Module Addressing	3-17
3.7.3	Read Protection Control	3-20
4	**Central Processing Unit (CPU)**	4-1
4.1	Instruction Pipelining	4-3
4.2	Particular Pipeline Effects	4-6
4.3	Bit-Handling and Bit-Protection	4-10
4.4	Instruction State Times	4-11
4.5	CPU Special Function Registers	4-12
5	**Interrupt and Trap Functions**	5-1
5.1	Interrupt System Structure	5-2
5.1.1	Interrupt Control Registers	5-5
5.2	Operation of the PEC Channels	5-11
5.3	Prioritization of Interrupt and PEC Service Requests	5-15
5.4	Saving Status during Interrupt Service	5-17
5.5	Interrupt Response Times	5-19
5.6	PEC Response Times	5-22
5.7	Interrupt Node Sharing	5-24
5.8	External Interrupts	5-25
5.9	Trap Functions	5-31

Table of Contents Page

6 Clock Generation 6-1
6.1 Oscillator 6-3
6.2 Frequency Control 6-5
6.3 Oscillator Watchdog 6-9
6.4 Clock Drivers 6-10

7 Parallel Ports 7-1
7.1 Input Threshold Control 7-2
7.2 Output Driver Control 7-4
7.3 Alternate Port Functions 7-10
7.4 PORT0 7-12
7.5 PORT1 7-16
7.6 Port 3 7-21
7.7 Port 4 7-26
7.8 Port 5 7-30
7.9 Port 8 7-33

8 Dedicated Pins 8-1

9 External Bus Interface 9-1
9.1 Single Chip Mode 9-2
9.2 External Bus Modes 9-3
9.3 Programmable Bus Characteristics 9-12
9.4 Controlling the External Bus Controller 9-18
9.5 EBC Idle State 9-27
9.6 The XBUS Interface 9-28
9.6.1 Accessing the On-chip XBUS Peripherals 9-29
9.6.2 External Accesses to XBUS Peripherals 9-30

10 General Purpose Timer Unit 10-1
10.1 Timer Block GPT1 10-1
10.1.1 GPT1 Core Timer T3 10-3
10.1.2 GPT1 Auxiliary Timers T2 and T4 10-12
10.1.3 Interrupt Control for GPT1 Timers 10-20

11 Asynchronous/Synchronous Serial Interface 11-1
11.1 Asynchronous Operation 11-5
11.2 Synchronous Operation 11-8
11.3 Hardware Error Detection Capabilities 11-10
11.4 ASC0 Baud Rate Generation 11-11
11.5 ASC0 Interrupt Control 11-15

12 High-Speed Synchronous Serial Interface 12-1
12.1 Full-Duplex Operation 12-7
12.2 Half-Duplex Operation 12-10

Table of Contents **Page**

12.3 Continuous Transfers 12-11
12.4 Port Control 12-12
12.5 Baud Rate Generation 12-13
12.6 Error Detection Mechanisms 12-15
12.7 SSC Interrupt Control 12-17

13 Watchdog Timer (WDT) 13-1
13.1 Operation of the Watchdog Timer 13-3
13.2 Reset Source Indication 13-6

14 Real Time Clock 14-1

15 Bootstrap Loader 15-1
15.1 Entering the Bootstrap Loader 15-2
15.2 Loading the Startup Code 15-5
15.3 Exiting Bootstrap Loader Mode 15-5
15.4 Choosing the Baudrate for the BSL 15-6

16 Capture/Compare Unit CAPCOM2 16-1
16.1 CAPCOM2 Timers 16-4
16.2 CAPCOM2 Unit Timer Interrupts 16-9
16.3 Capture/Compare Registers 16-10
16.4 Capture Mode 16-13
16.5 Compare Modes 16-14
16.6 Capture/Compare Interrupts 16-22

17 Capture/Compare Unit CAPCOM6 17-1
17.1 Output Signal Level Control 17-4
17.2 Edge Aligned Mode 17-5
17.3 Center Aligned Mode 17-7
17.3.1 Timing Relationships 17-8
17.4 Burst Mode 17-10
17.5 Capture Mode 17-11
17.6 Combined Multi-Channel Modes 17-12
17.6.1 Output Signals in Multi-Channel Mode 17-13
17.6.2 Block Commutation Mode 17-16
17.7 Trap Function 17-18
17.8 Register Descriptions 17-21
17.9 The CAPCOM6 Interrupt Structure 17-32

18 Analog/Digital Converter 18-1
18.1 Mode Selection and Operation 18-3
18.2 Conversion Timing Control 18-13
18.3 A/D Converter Interrupt Control 18-15

Table of Contents **Page**

19 On-Chip CAN Interface 19-1
19.1 Functional Blocks of the CAN Module 19-2
19.2 General Functional Description 19-7
19.2.1 CAN Interrupt Handling 19-9
19.2.2 Configuration of the Bit Timing 19-11
19.2.3 Mask Registers 19-15
19.3 The Message Object 19-18
19.4 Controlling the CAN Module 19-30
19.5 Configuration Examples for Message Objects 19-34
19.6 CAN Application Interface 19-36

20 System Reset 20-1
20.1 Reset Sources 20-2
20.2 Status After Reset 20-5
20.3 Application-Specific Initialization Routine 20-9
20.4 System Startup Configuration 20-12
20.4.1 System Startup Configuration upon an External Reset 20-13
20.4.2 System Startup Configuration at Single-Chip Mode Reset 20-20
20.5 System Configuration via Software 20-22

21 Power Management 21-1
21.1 Idle Mode 21-3
21.2 Sleep Mode 21-5
21.3 Power Down Mode 21-6
21.3.1 Output Pins Status During Power Reduction Modes 21-8
21.4 Slow Down Operation 21-10
21.5 Flexible Peripheral Management 21-14
21.6 Programmable Frequency Output Signal 21-16
21.7 Security Mechanism 21-21

22 System Programming 22-1
22.1 Stack Operations 22-4
22.2 Register Banking 22-9
22.3 Procedure Call Entry and Exit 22-9
22.4 Table Searching 22-12
22.5 Floating Point Support 22-12
22.6 Peripheral Control and Interface 22-13
22.7 Trap/Interrupt Entry and Exit 22-13
22.8 Inseparable Instruction Sequences 22-14
22.9 Overriding the DPP Addressing Mechanism 22-14
22.10 Handling the Internal Code Memory 22-16
22.11 Pits, Traps, and Mines 22-18

23 Register Set 23-1

Table of Contents **Page**

23.1 Register Description Format 23-1
23.2 CPU General Purpose Registers (GPRs) 23-2
23.3 Registers Ordered by Name 23-4
23.4 Registers Ordered by Address 23-11
23.5 Special Notes 23-18

24 Instruction Set Summary 24-1

25 Device Specification 25-1

26 Keyword Index 26-1

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 family of 16-bit CMOS microcontrollers without the constraints of backward compatibility.

Nonetheless the architecture of the 16-bit microcontroller family pursues 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 C166 Family, the C164 group.

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:

- **C164CI-8R** 64 KByte Program ROM, full-function CAPCOM6, CAN module
- **C164SI-8R** 64 KByte Program ROM, full-function CAPCOM6
- **C164CL-8R** 64 KByte Program ROM, reduced CAPCOM6, CAN module
- **C164SL-8R** 64 KByte Program ROM, reduced CAPCOM6
- **C164CI-L** No Program memory, full-function CAPCOM6, CAN module
- **C164CI-8E** 64 KByte Program OTP, full-function CAPCOM6, CAN module
- **C164CL-6R** 48 KByte Program ROM, reduced CAPCOM6, CAN module
- **C164SL-6R** 48 KByte Program ROM, reduced CAPCOM6

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 **C164CI** 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 C164CI 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.

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.

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

- 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.

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 C164CI, descriptions within this manual that refer to the “C164CI” also apply to the other variations, unless otherwise noted.

1.2 Summary of Basic Features

The C164CI devices are enhanced members of the Infineon family of full featured 16-bit single-chip CMOS microcontrollers. The C164CI combines high CPU performance (up to 12.5 million instructions per second) with high peripheral functionality and provides a means for power reduction.
Several key features contribute to the high performance of the C164CI (the indicated timings refer to a CPU clock of 25 MHz).

High Performance 16-bit CPU with Four-Stage Pipeline

- 80 ns minimum instruction cycle time, with most instructions executed in 1 cycle
- 400 ns multiplication (16-bit × 16-bit), 800 ns division (32-bit / 16-bit)
- Multiple high bandwidth internal data buses
- Register-based design with multiple, variable register banks
- Single-cycle 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

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

- Programmable system slowdown via Slow Down Divider (SDD)
- Flexible management of peripherals, can be individually disabled
- Sleep-mode supports wake-up via external interrupts
- Programmable frequency output

Integrated On-Chip Memory

- 2 KBytes Internal RAM (IRAM) for variables, register banks, system stack, and code
- 2 KBytes on-chip high-speed extension RAM (XRAM) for variables, user stacks, and code
- 64 KBytes on-chip Program memory (OTP or Mask ROM, not for ROMless devices)

External Bus Interface

- Multiplexed or demultiplexed bus configurations
- Segmentation capability and chip select signal generation
- 8-bit or 16-bit data bus
- Bus cycle characteristics selectable for five programmable address areas

16-Priority-Level Interrupt System

- 32 interrupt nodes with separate interrupt vectors
- 240 ns typical interrupt latency (400 ns maximum) in case of internal program execution
- Fast external interrupts

8-Channel Peripheral Event Controller (PEC)

- Interrupt driven single cycle data transfer
- Transfer count option (standard CPU interrupt after programmable number of PEC transfers)
- Overhead from saving and restoring system state for interrupt requests eliminated

Intelligent On-Chip Peripheral Subsystems

- 8-channel 10-bit A/D Converter with programmable conversion time (7.8 µs minimum), auto scan modes, channel injection mode
- Two Capture/Compare Units with independent time bases, very flexible PWM unit/event recording unit with different operating modes
- Multifunctional General Purpose Timer Unit with three 16-bit timers/counters, maximum resolution $f_{CPU}/8$
- Asynchronous/Synchronous Serial Channel (USART) with baud rate generator, parity, framing, and overrun error detection
- High Speed Synchronous Serial Channel programmable data length and shift direction
- Controller Area Network (CAN) Module, Rev. 2.0B active, with 15 Message Objects, Full-CAN/Basic-CAN
- Real Time Clock
- Watchdog Timer with programmable time intervals
- Bootstrap Loader for flexible system initialization

59 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

Various Temperature Ranges

- 0 to +70 °C
- -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.

80-Pin Plastic Metric Quad Flat Pack (MQFP) Package

- P-MQFP, 14 × 14 mm body, 0.65 mm (25.6 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 and C166 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, MODULA2, FORTH)
- 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)
- Network driver software (CAN, PROFIBUS)

1.3 Abbreviations

The following acronyms and terms are used within this document:

ADC	Analog Digital Converter
ALE	Address Latch Enable
ALU	Arithmetic and Logic Unit
ASC	Asynchronous/synchronous Serial Controller
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
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
OTP	One-Time Programmable memory
PEC	Peripheral Event Controller
PLA	Programmable Logic Array
PLL	Phase Locked Loop
PWM	Pulse Width Modulation
RAM	Random Access Memory
RISC	Reduced Instruction Set Computing
ROM	Read Only Memory
RTC	Real Time Clock
SDD	Slow Down Divider
SFR	Special Function Register
SSC	Synchronous Serial Controller
XBUS	Internal representation of the External Bus
XRAM	On-chip extension RAM

2 Architectural Overview

The architecture of the C164CI core combines the advantages of both RISC and CISC processors in a very well-balanced way. The C164CI integrates this powerful CPU core with a set of powerful peripheral units into one chip and connects them very efficiently. 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 four buses used concurrently on the C164CI is the XBUS, 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 C164CI.

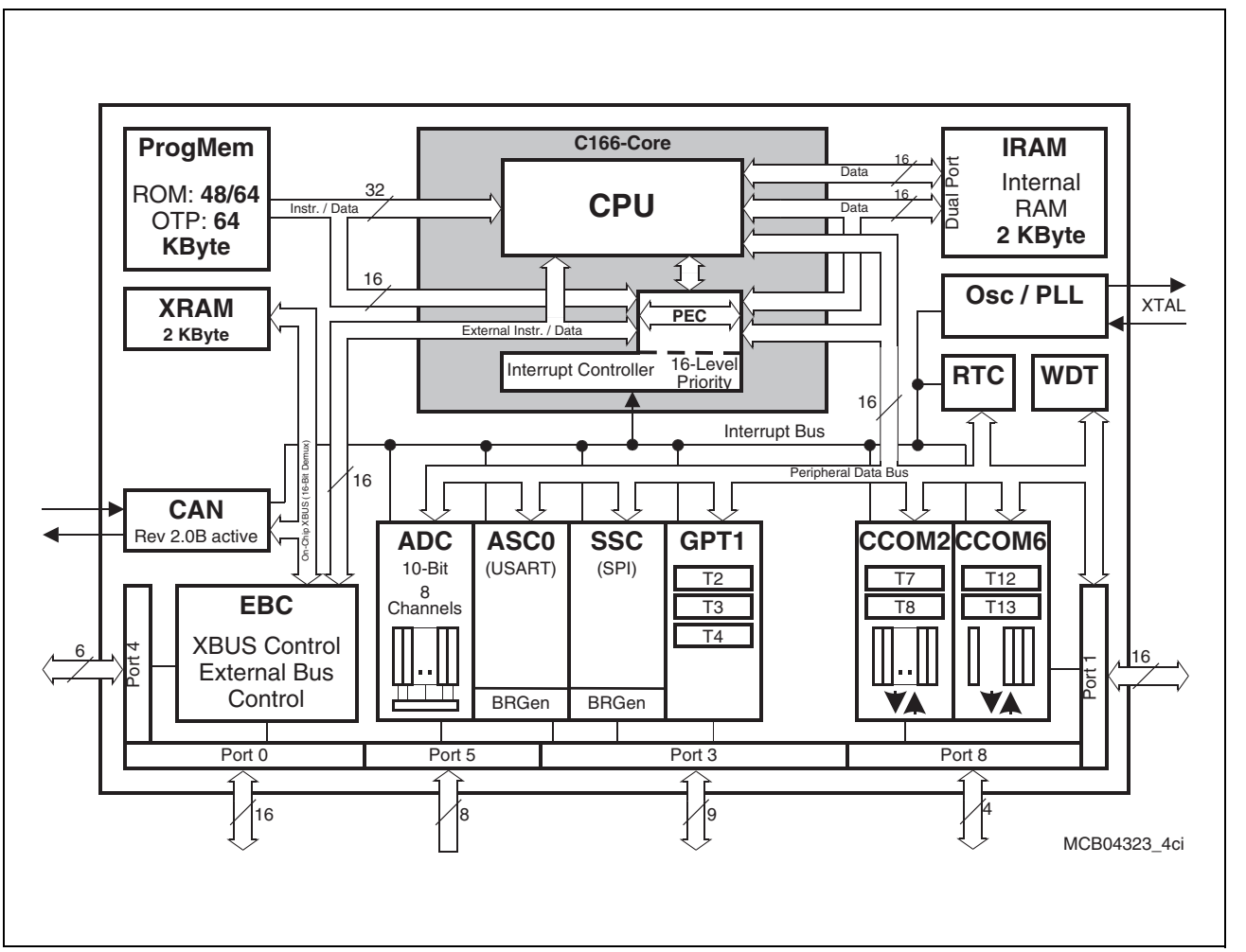


Figure 2-1 C164CI Functional Block Diagram

2.1 Basic CPU Concepts and Optimizations

The main core of the CPU consists of a four-stage instruction pipeline, a 16-bit Arithmetic and Logic Unit (ALU) and dedicated Special Function Registers (SFRs). Additional hardware is provided for a separate multiply and divide unit, a bit-mask generator, and a barrel shifter.

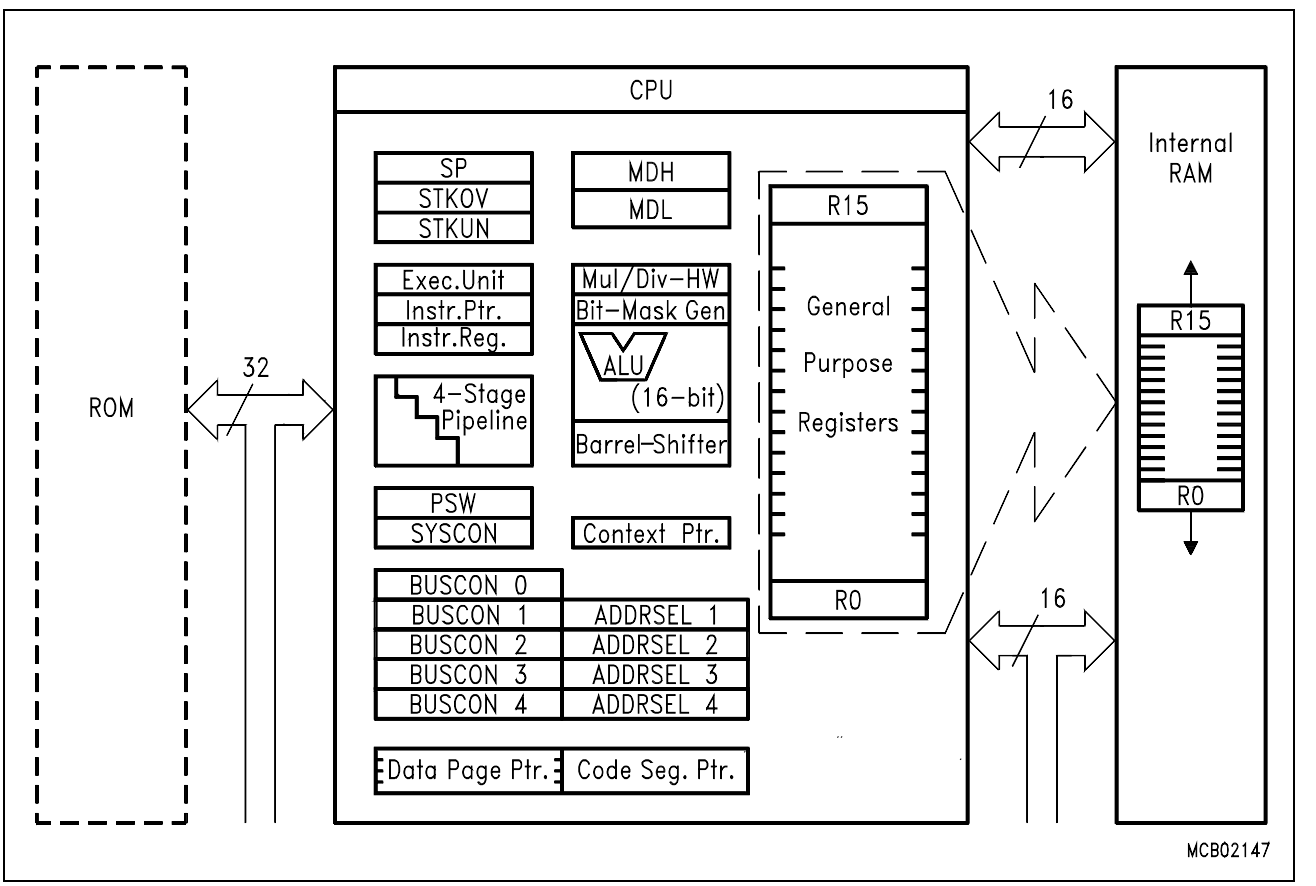


Figure 2-2 CPU Block Diagram

To meet the demand for greater performance and flexibility, a number of functional blocks of the CPU have been optimized. These blocks are controlled by signals from the instruction decode logic. Optimizations of the functional blocks are summarized below and described in detail in the following sections:

1. High Instruction Bandwidth / Fast Execution
2. High Function 8-bit and 16-bit Arithmetic and Logic Unit
3. Extended Bit Processing and Peripheral Control
4. High Performance Branch-, Call-, and Loop Processing
5. Consistent and Optimized Instruction Formats
6. Programmable Multiple Priority Interrupt Structure

2.1.1 High Instruction Bandwidth / Fast Execution

Based on the hardware provisions, most of the C164CI's instructions can be executed in just one machine cycle, which requires two CPU clock cycles ($2 \times 1/f_{CPU}$ = 4 TCL). For example, shift and rotate instructions are always processed within one machine cycle, independent of the number of bits to be shifted.

Branch-, multiply- and divide instructions normally take more than one machine cycle. These instructions, however, have also been optimized. For example, branch instructions require an additional machine cycle only when a branch is taken. Subsequent branches taken in loops require no additional machine cycles at all, due to the Jump Cache feature.

A 32-bit / 16-bit division requires 20 CPU clock cycles, a 16-bit $\times$ 16-bit multiplication requires 10 CPU clock cycles.

The instruction cycle time has been dramatically reduced through the use of instruction pipelining. This technique allows the core CPU to process portions of multiple sequential instruction stages in parallel. The following four-stage pipeline provides the optimum balancing for the CPU core:

FETCH: In this stage, an instruction is fetched from the internal ROM or RAM or from the external memory, based on the current IP value.

DECODE: In this stage, the previously fetched instruction is decoded and the required operands are fetched.

EXECUTE: In this stage, the specified operation is performed on the previously fetched operands.

WRITE BACK: In this stage, the result is written to the specified location.

If this technique were not used, each instruction would require four machine cycles. This increased performance allows a greater number of tasks and interrupts to be processed.

Instruction Decoder

Instruction decoding is generated primarily from Programmable Logic Array (PLA) outputs based on the selected opcode. No microcode is used and each pipeline stage receives control signals staged in control registers from the decode stage PLAs. Pipeline holds are primarily caused by wait states for external memory accesses and cause the holding of signals in the control registers. Multiple-cycle instructions are performed through instruction injection and simple internal state machines which modify required control signals.

High Function 8-bit and 16-bit Arithmetic and Logic Unit

All standard arithmetic and logical operations are performed in a 16-bit ALU. 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. Once the pipeline has been filled, one instruction is completed per machine cycle, except for multiply and divide. An advanced Booth algorithm has been incorporated to allow four bits to be multiplied and two bits to be divided per machine cycle. Thus, these operations use two coupled 16-bit registers, MDL and MDH, and require four and nine machine cycles, respectively, to perform a 16-bit by 16-bit (or 32-bit by 16-bit) calculation plus one machine cycle to setup and adjust the operands and the result. Even these longer multiply and divide instructions can be interrupted during their execution to allow for very fast interrupt response. 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.

All targets for branch calculations are also computed in the central ALU.

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

Extended Bit Processing and Peripheral Control

A large number of instructions has been dedicated to bit processing. These instructions provide efficient control and testing of peripherals while enhancing data manipulation. Unlike other microcontrollers, these instructions provide direct access to two operands in the bit-addressable space without requiring them to be moved into temporary flags.

The same logical instructions available for words and bytes are also supported for bits. This allows the user to compare and modify a control bit for a peripheral in one instruction. Multiple bit shift instructions have been included to avoid long instruction streams of single bit shift operations. These instructions are also performed in a single machine cycle.

Bit field instructions have been provided as well to allow the modification of multiple bits from one operand in a single instruction.

High Performance Branch-, Call-, and Loop Processing

Due to the high percentage of branching in controller applications, branch instructions have been optimized to require one extra machine cycle only when a branch is taken. This is implemented by precalculating the target address while decoding the instruction. To decrease loop execution overhead, three enhancements have been provided:

- Single cycle branch execution after the first iteration of a loop:
  The first solution provides that only one machine cycle is lost during the execution of the entire loop. In loops which fall through upon completion, no machine cycles are lost when exiting the loop. No special instructions are required to perform loops, and loops are automatically detected during execution of branch instructions.
- Detection of the end of a table:
  The second loop enhancement avoids the use of two compare instructions embedded in loops. One simply places the lowest negative number at the end of the specific table and specifies branching if neither its value nor the compared value have been found. Otherwise, the loop is terminated if either condition has been met. The terminating condition can then be tested.
- Compare and Increment or Decrement instructions:
  The third loop enhancement provides a more flexible solution than the Decrement and Skip on Zero instruction found in other microcontrollers. The use of Compare and Increment or Decrement instructions enables the user to make comparisons to any value. This allows loop counters to cover any range and is particularly advantageous in table searching.

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.

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 C164CI instruction set which includes the following instruction classes:

- Arithmetic 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 and words. Specific instructions support the conversion (extension) of bytes to words. Various direct, indirect, and immediate addressing modes are provided to specify the required operands.

2.1.2 Programmable Multiple Priority Interrupt System

The following enhancements within the C164CI 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 in segment 0 with an optional increment of either the PEC source or the destination pointer. 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 possible interrupt sources, 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 possible interrupt sources has a dedicated vector location.
- Multiple Register Banks: This feature allows the user to specify up to sixteen general purpose registers located anywhere in the internal RAM. A single one-machine-cycle instruction allows register banks to switch from one task to another.
- Interruptible Multiple Cycle Instructions: Reduced interrupt latency is provided by allowing multiple-cycle instructions (multiply, divide) to be interruptible.

The C164CI is capable of reacting very quickly to non-deterministic events because its interrupt response time is within a very narrow range of only 5 to 10 CPU clock cycles (in the case of internal program execution). Its fast external interrupt inputs are sampled every CPU clock cycle and allow even very short external signals to be recognized.

The C164CI 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.

2.2 On-Chip System Resources

The C164CI 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 C164CI has eight PEC channels, each of which offers such fast interrupt-driven data transfer capabilities.

Memory Areas

The memory space of the C164CI 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.

2 KBytes of 16-bit wide Internal RAM provide fast access to General Purpose Registers (GPRs), user data (variables), and system stack. The internal RAM may also be used for code. A unique decoding scheme provides flexible user register banks in the internal memory while optimizing the remaining RAM for user data.

The CPU has an actual register context of up to 16 wordwide and/or bytewide 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 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 1024 words is provided as storage for temporary data. The system stack is also located within the on-chip RAM area and it is accessed by the CPU via the Stack Pointer (SP) 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.

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.

2 KBytes of 16-bit wide on-chip XRAM provide fast access to user data (variables), user stacks, and code. The on-chip XRAM is implemented as an X-Peripheral and appears to the software as an external RAM. Therefore, it cannot store register banks and is not bitaddressable. The XRAM allows 16-bit accesses with maximum speed.

For Special Function Registers 1024 Bytes 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. (E)SFRs are wordwide registers which are used for controlling and monitoring functions of the different on-chip units. Unused (E)SFR addresses are reserved for future members of the C166 microcontroller family with enhanced functionality.

Optional on-chip OTP or ROM memory provides both code and constant data storage. This memory area is connected to the CPU via a 32-bit-wide bus. Thus, an entire double-word instruction can be fetched in only one machine cycle. The ROM is mask programmed at the factory while the OTP memory can also be programmed within the application. Program execution from on-chip program memory is the fastest of all possible alternatives.

The type of the on-chip program memory (OTP/ROM/none) depends on the chosen derivative.

External Bus Interface

To meet the needs of designs where more memory is required than is provided on chip, up to 4 MBytes of external RAM and/or ROM can be connected to the C164CI microcontroller via its external bus interface. The integrated External Bus Controller (EBC) allows very flexible access to external memory and/or peripheral resources. For up to five address areas the bus mode (multiplexed / demultiplexed), the data bus width (8-bit/16-bit) and even the length of a bus cycle (waitstates, signal delays) can be selected independently. This allows access to a variety of memory and peripheral components directly and with maximum efficiency. If the device does not run in Single Chip Mode, where no external memory is required, the EBC can control external accesses in one of the following external access modes:

- 16-/18-/20-/22-bit Addresses, 16-bit Data, Demultiplexed
- 16-/18-/20-/22-bit Addresses, 8-bit Data, Demultiplexed
- 16-/18-/20-/22-bit Addresses, 16-bit Data, Multiplexed
- 16-/18-/20-/22-bit Addresses, 8-bit Data, Multiplexed

The demultiplexed bus modes use PORT1 for addresses and PORT0 for data input/output. The multiplexed bus modes use PORT0 for both addresses and data input/output. Port 4 is used for the upper address lines (A16 …, if selected) and for the $\overline{CS}$ lines ($\overline{CS0}$ …, if selected).

Important timing characteristics of the external bus interface (waitstates, ALE length, and Read/Write Delay) have been made programmable to allow the user select a wide range of different types of memories and/or peripherals.

For applications which require less than 64 KBytes of address space, a non-segmented memory model can be selected, where all locations can be addressed by 16 bits. Thus, Port 4 is not needed as an output for the upper address bits (Axx … A16), as is the case when using the segmented memory model.

The on-chip XBUS is an internal representation of the external bus. It allows access to integrated application-specific peripherals/modules in the same way as external components. It provides a defined interface for these customized peripherals. Both the on-chip XRAM and the on-chip CAN-Module are examples for these X-Peripherals.

2.3 On-Chip Peripheral Blocks

The C166 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 ($00'FE00_H$ … $00'FFFF_H$) or within the extended ESFR area ($00'F000_H$ … $00'F1FF_H$).

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 C164CI generic peripherals are:

- A General Purpose Timer Block (GPT1)
- Two Serial Interfaces (ASC0 and SSC)
- A Watchdog Timer
- Two Capture / Compare units (CAPCOM2 and CAPCOM6)
- A 10-bit Analog / Digital Converter
- A Real Time Clock
- Six I/O ports with a total of 59 I/O lines

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 CPU 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 CPU clock (f_{CPU}). The on-chip oscillator derives the CPU 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 C164CI'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.

Serial Channels

Serial communication with other microcontrollers, processors, terminals, or external peripheral components is provided by two serial interfaces with different functionality: an Asynchronous/Synchronous Serial Channel (**ASC0**) and a High-Speed Synchronous Serial Channel (**SSC**).

The ASC0 is upward compatible with the serial ports of the Infineon 8-bit microcontroller families and supports full-duplex asynchronous communication at up to 780 KBit/s and half-duplex synchronous communication at up to 3.1 MBit/s @ 25 MHz CPU clock.
A dedicated baud rate generator allows all standard baud rates to be set up without oscillator tuning. Four separate interrupt vectors are provided for transmission, reception, and error handling. In asynchronous mode, 8- or 9-bit data frames are transmitted or received, preceded by a start bit and terminated by one or two stop bits. For multiprocessor communication, a mechanism has been included to distinguish address bytes from data bytes (8-bit data plus wake-up bit mode). In synchronous mode, the ASC0 transmits or receives bytes (8 bits) synchronously to a shift clock which is generated by the ASC0. The ASC0 always shifts the Least Significant Bit (LSB) first. A loop back option is available for testing purposes.

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

The SSC supports full-duplex synchronous communication at up to 6.25 Mbit/s @ 25 MHz CPU clock. It may be configured so that it interfaces with serially linked peripheral components. A dedicated baud rate generator allows set up of all standard baud rates without oscillator tuning. Three separate interrupt vectors are provided for transmission, reception, and error handling.
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 Most Significant Bit (MSB) and allows selection of shifting and latching clock edges as well as the clock polarity.
A number of optional hardware error detection capabilities has been included to increase the reliability of data transfers. Transmit and receive error supervise the correct handling of the data buffer. Phase and baudrate error detect incorrect serial data.

On-Chip CAN Module

The integrated CAN 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 CAN Module can receive and transmit standard frames with 11-bit identifiers as well as extended frames with 29-bit identifiers.

The module provides Full CAN functionality on up to 15 message objects. Message object 15 may be configured for Basic CAN functionality. Both modes provide separate masks for acceptance filtering which allows acceptance of a number of identifiers in Full CAN mode and also allows a number of identifiers in Basic CAN mode to be disregarded. All message objects can be updated independently from the other objects and are equipped for the maximum message length of 8 bytes.

The bit timing is derived from the CPU clock and is programmable up to a data rate of 1 Mbit/s. The CAN Module uses two pins (configurable) to interface to a bus transceiver.

Parallel Ports

The C164CI provides up to 59 I/O lines which are organized into five input/output ports and one input port. All port lines are bit-addressable, and all input/output lines are individually programmable (bit-wise) 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 three 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.

All port lines have programmable alternate input or output functions associated with them. PORT0 and PORT1 may be used as address and data lines when accessing external memory, while Port 4 outputs the additional segment address bits A21/19/17 … A16 in systems where segmentation is used to access more than 64 KBytes of memory. Port 4 may also output the optional chip select signals $\overline{CS3}$ … $\overline{CS0}$. PORT1 provides input and output signals for the CAPCOM units. Port 3 includes alternate functions of timers, serial interfaces, the optional bus control signal $\overline{BHE}$, and the system clock output (CLKOUT/FOUT). Port 5 is used for timer control signals and for the analog inputs to the A/D Converter. Port 8 provides inputs/outputs for the CAPCOM2 unit. All port lines not used for these alternate functions may be used as general purpose I/O lines.

A/D Converter

For analog signal measurement, a 10-bit Analog/Digital (A/D) Converter with eight 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 are programmable and can so be adjusted to the external circuitry.

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 fewer analog input channels, the remaining channel inputs can be used as digital input port pins.

The A/D Converter of the C164CI 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.

Real Time Clock

The C164CI contains a Real Time Clock (RTC) which serves different purposes:

- System clock to determine the current time and date,
  even during idle mode and power down mode (optionally).
- Cyclic time based interrupt
  (for example: to provide a system time tick independent of the CPU frequency without loading the general purpose timers, or to wake up regularly from idle mode).
- 48-bit timer for long term measurements
  (the maximum usable timespan is more than 100 years).

The RTC module consists of a chain of three divider blocks, a fixed 8:1 divider, the reloadable 16-bit timer T14, and the 32-bit RTC timer (accessible via registers RTCH and RTCL). Both timers count upwards.

General Purpose Timer (GPT) Unit

The GPT1 unit utilizes 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.

Each timer may operate independently in a number of different modes, or may be concatenated with another timer of the same module.

Each timer can be configured individually for one of four basic modes of operation: Timer, Gated Timer, Counter Mode, and Incremental Interface Mode. In Timer Mode, the input clock for a timer is derived from the internal CPU clock divided by a programmable prescaler, while Counter Mode allows a timer to be clocked in reference to external events (via TxIN).
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 its external input pin TxIN.
In Incremental Interface Mode, the GPT1 timers can be directly connected to the incremental position sensor signals A and B via the respective inputs TxIN and TxEUD. Direction and count signals are internally derived from these two input signals, so, the contents of timer Tx corresponds to the sensor position. The third position sensor signal TOP0 can be connected to an interrupt input.

The count direction (up/down) for each timer is programmable by software or may additionally be altered dynamically by an external signal (TxEUD) to facilitate tasks such as position tracking.

The core timer T3 has an output toggle latch (T3OTL) which changes its state on each timer overflow/underflow. The state of this latch may be used internally to concatenate the core timer with the respective auxiliary timers resulting in 32/33-bit timers/counters for measuring long time periods with high resolution.

Various reload or capture functions can be selected to reload timers or capture a timer’s contents triggered by an external signal or a selectable transition of toggle latch T3OTL.

The maximum resolution of the timers in module GPT1 is 8 CPU clock cycles (= 16 TCL).

Capture/Compare (CAPCOM) Units

The CAPCOM units are typically used to handle high speed I/O tasks such as pulse and waveform generation, pulse width modulation (PWM), Digital to Analog (D/A) conversion, software timing, or time recording relative to external events.

A number of dedicated timers with reload registers provide independent time bases for the capture/compare channels. The input clock for the timers is programmable to several prescaled values of the internal CPU clock, or may be derived from an overflow/ underflow of timer T3 in module GPT1 (for CAPCOM2 timers). This provides a wide range of variation for the timer period and resolution and allows precise adjustments to the application specific requirements. In addition, external inputs for the CAPCOM units allow event scheduling for the capture/compare registers relative to external events.

The CAPCOM2 unit supports generation and control of timing sequences on up to 16 channels (8 I/O pins) with a maximum resolution of 8 CPU clock cycles. The capture/ compare register array contains 16 dual purpose capture/compare registers, each of which may be individually allocated to either CAPCOM2 timer T7 or T8, and programmed for capture or compare function. Eight registers have port pins associated with them: they serve as input pins for triggering the capture function, or as output pins to indicate the occurrence of a compare event.

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.

The CAPCOM6 unit provides three capture/compare channels and one additional compare channel. The three capture/compare channels can control two output lines each, which can be programmed to generate non-overlapping pulse patterns. The additional compare channel may either generate a separate output signal or modulate the output signals of the three other channels. The active level for each output can be selected individually.

Versatile multichannel PWM signals can be generated: controlled either internally via a timer or externally, for example via hall sensors. The trap function allows the outputs to be driven to a defined level in response to an external signal.

Note: Multichannel PWM modes are only available in devices with a full-function CAPCOM6, not in the reduced CAPCOM6.

Watchdog Timer

The Watchdog Timer is one of the fail-safe mechanisms implemented in the C164CI 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 only for the time interval before the EINIT (end of initialization) instruction has been executed. Thus, the chip's start-up procedure is always monitored. The software must be designed to service the Watchdog Timer before it overflows. If the software fails to do so, due to either hardware or software related failures, the Watchdog Timer overflows and generates an internal hardware reset and pulls the $\overline{\text{RSTOUT}}$ pin low to allow external hardware components to reset.

The Watchdog Timer is a 16-bit timer, clocked with the CPU clock divided by 2, 4, 128, or 256. The high byte of the Watchdog Timer register can be set to a prespecified reload value (stored in WDTREL) to allow further variation of the monitored time interval. Each time it is serviced by the application software, the high byte of the Watchdog Timer is reloaded. Thus, time intervals between 21 µs and 671 ms can be monitored (@ 25 MHz). The default Watchdog Timer interval after reset is 5.2 ms (@ 25 MHz).

2.4 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.

- Flexible clock generation
- Flexible peripheral management (peripherals can be dis/enabled separately or in groups)
- Periodic wakeup 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).

Flexible Clock Generation

The flexible clock generation system combines a variety of improved mechanisms (partly user controllable) to provide the C164CI 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 C164CI.

The clock system efficiently controls the amount of power which is consumed in order to distribute the clock signal within the C164CI.

Slowdown operation is achieved by dividing the oscillator clock by a programmable factor (1 … 32) resulting in a low frequency device operation which significantly reduces the overall power consumption.

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. All others may be switched off. It also allows the operation control of entire groups of peripherals including the power required for generating and distributing their clock input signal. Other peripherals may remain active: for example, in order to maintain communication channels. The registers of separately disabled peripherals (not within a disabled group) can still be accessed.

Periodic Wakeup from Idle or Sleep Mode

Periodic wakeup 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.

2.5 Protected Bits

The C164CI 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 **Chapter 4**).

The following bits are protected:

Table 2-1 C164CI Protected Bits

Register	Bit Name	Notes
T2IC, T3IC, T4IC	**T2IR, T3IR, T4IR**	GPT1 timer interrupt request flags
T3CON	**T3OTL**	GPT1 timer output toggle latches
T7IC, T8IC	**T7IR, T8IR**	CAPCOM2 timer interrupt request flags
S0TIC, S0TBIC	**S0TIR, S0TBIR**	ASC0 transmit (buffer) interrupt request flags
S0RIC, S0EIC	**S0RIR, S0EIR**	ASC0 receive/error interrupt request flags
S0CON	**S0REN**	ASC0 receiver enable flag
SSCTIC, SSCRIC	**SSCTIR, SSCRIR**	SSC transmit/receive interrupt request flags
SSCEIC	**SSCEIR**	SSC error interrupt request flag
SSCCON	**SSCBSY**	SSC busy flag
SSCCON	**SSCBE, SSCPE**	SSC error flags
SSCCON	**SSCRE, SSCTE**	SSC error flags
ADCIC, ADEIC	**ADCIR, ADEIR**	ADC end-of-conv./overrun intr. request flag
ADCON	**ADST, ADCRQ**	ADC start flag/injection request flag
CC31IC … CC16IC	**CC31IR … CC16IR**	Fast external interrupt request flags
TFR	**TFR.15, 14, 13**	Class A trap flags
TFR	**TFR.7, 3, 2, 1, 0**	Class B trap flags
P1H	**P1H.7 … P1H.4**	Those bits of PORT1 used for CAPCOM2
P8	**P8.3 … P8.0**	All bits of Port 8 used for CAPCOM2
ISNC	**RTCIR**	Interrupt node sharing request flag
XP0IC, XP3IC	**XP0IR, XP3IR**	CAN and PLL/RTC interrupt request flags

Σ = 58 protected bits.

3 Memory Organization

The memory space of the C164CI is configured in a “Von Neumann” architecture. This means that code and data are accessed within the same linear address space. All of the physically separated memory areas, including internal ROM/Flash/OTP (where integrated), internal RAM, the internal Special Function Register Areas (SFRs and ESFRs), the address areas for integrated XBUS peripherals and external memory are mapped into one common address space.

The C164CI 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**).

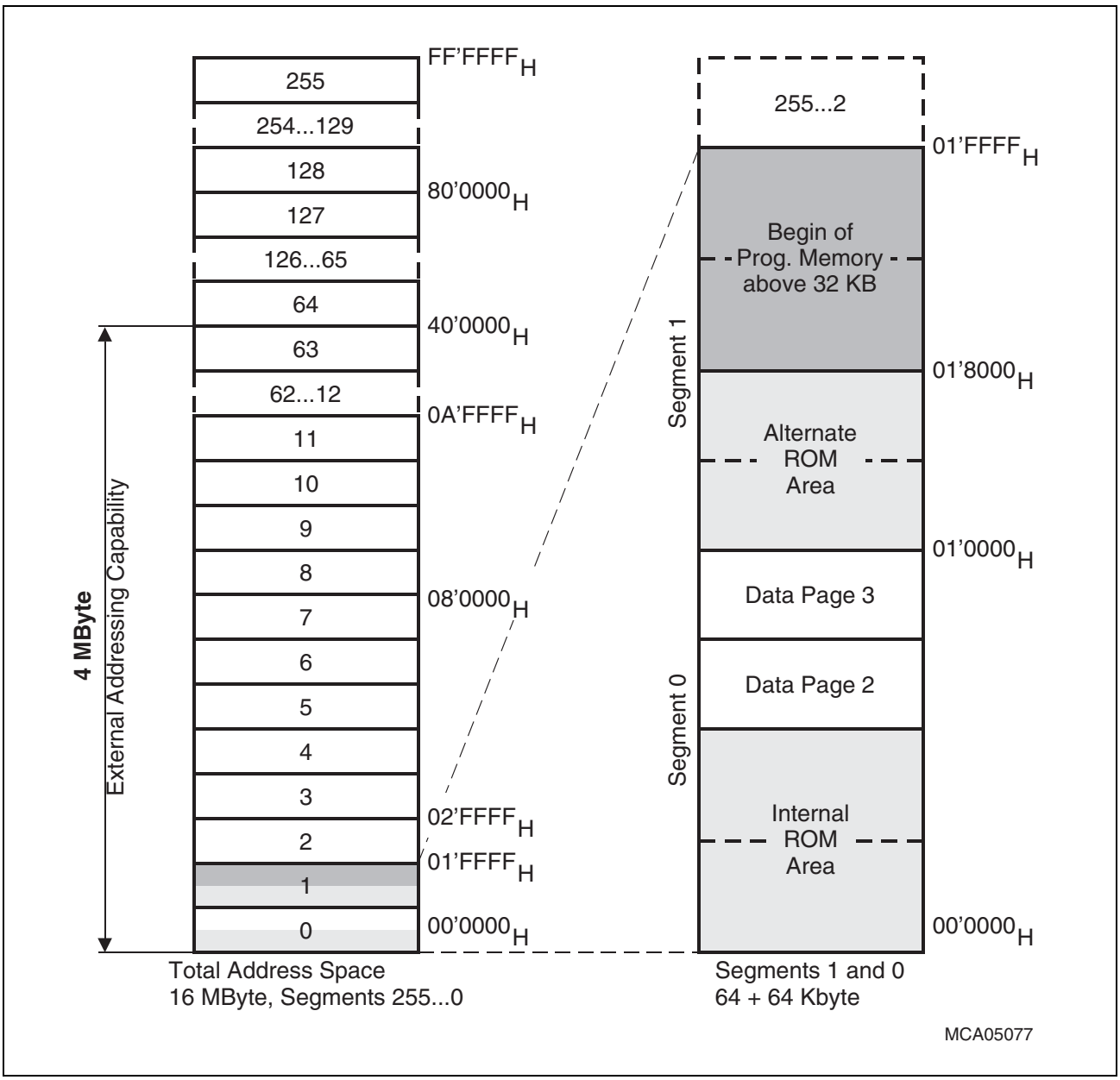


Figure 3-1 Address Space Overview

Most internal memory areas are mapped into segment 0, the system segment. The upper 4 KBytes of segment 0 ($00'F000_H$ … $00'FFFF_H$) hold the Internal RAM and Special Function Register Areas (SFR and ESFR). The lower 32 KBytes of segment 0 ($00'0000_H$ … $00'7FFF_H$) may be occupied by a portion of the on-chip ROM/Flash/OTP memory and is called the Internal ROM area. This ROM area can be remapped to segment 1 ($01'0000_H$ … $01'7FFF_H$), to enable external memory access in the lower half of segment 0, or the internal ROM may be disabled completely.

Code and data may be stored in any part of the internal memory areas, except for the SFR blocks, which may be used for control/data, but not for instructions.

Note: Accesses to the internal ROM area on ROMless devices will produce unpredictable results.

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 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 portion of the Special Function Registers, a portion of the internal RAM, and for the General Purpose Registers.

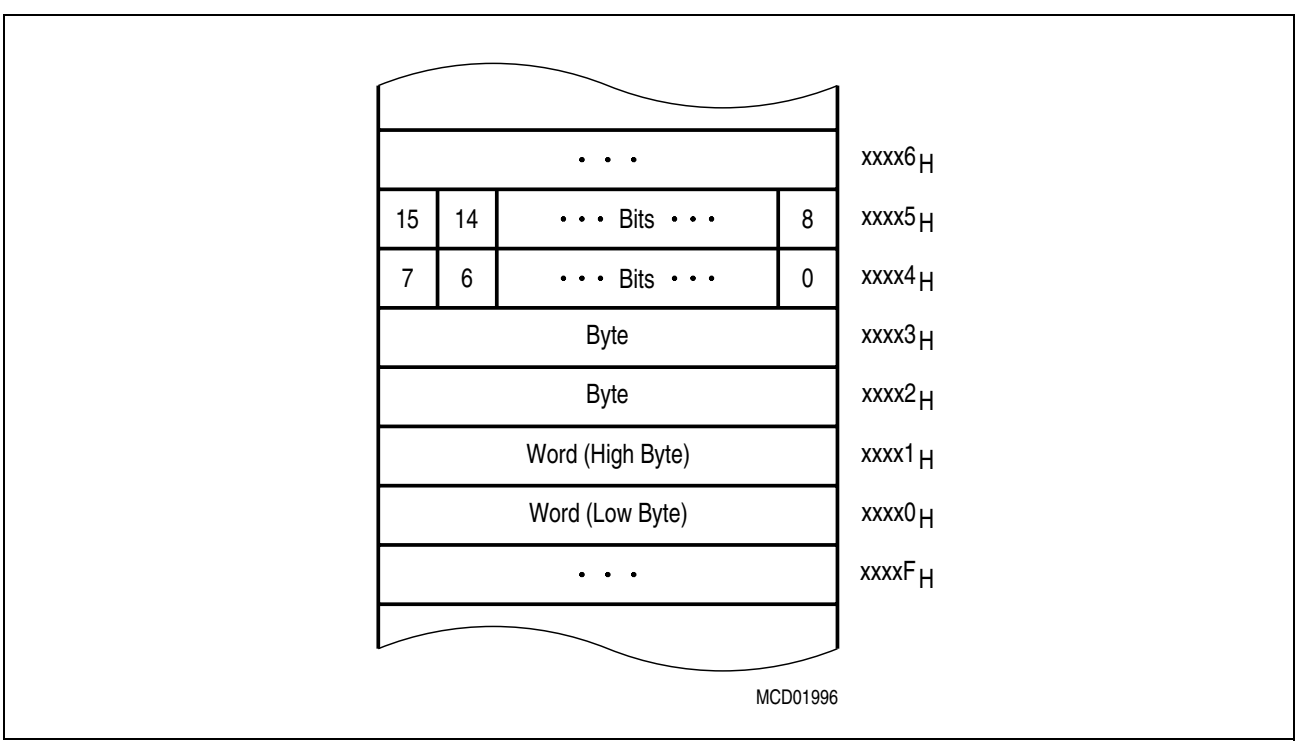


Figure 3-2 Storage of Words, Bytes, and Bits in a Byte Organized Memory

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 Internal ROM Area

The C164CI may reserve an address area of variable size (depending on the version) for on-chip mask-programmable ROM/Flash/OTP memory (organized as X × 32). The lower 32 KBytes of this on-chip memory block are referred to as “Internal ROM Area”. Internal ROM accesses are globally enabled or disabled via bit ROMEN in register SYSCON. This bit is set during reset according to the level on pin $\overline{EA}$, or may be altered via software. If enabled, the internal ROM area occupies the lower 32 KBytes of either segment 0 or segment 1 (alternate ROM area). This mapping is controlled by bit ROMS1 in register SYSCON.

Note: The size of the internal ROM area is independent of the actual size of the implemented Program Memory. Also devices with less than 32 KBytes of Program Memory or without any Program Memory will have this 32-KByte area occupied if the Program Memory is enabled. Devices with a larger Program Memory provide the mapping option only for the internal ROM area.

Devices with a Program Memory size above 32 KBytes expand the ROM area from the middle of segment 1, starting at address $01'8000_H$.

The internal Program Memory can be used for both code (instructions) and data (constants, tables, etc.) storage.

Code fetches are always made on even byte addresses. The highest possible code storage location in the internal Program Memory is either $xx'xxFE_H$ for single word instructions, or $xx'xxFC_H$ for double word instructions. The respective location must contain a branch instruction (unconditional), because sequential boundary crossing from internal Program Memory to external memory is not supported and causes erroneous results.

Any word and byte data read accesses may use the indirect or long 16-bit addressing modes. There is no short addressing mode for internal ROM operands. Any word data access is made to an even byte address. The highest possible word data storage location in the internal Program Memory is $xx'xxFE_H$. For PEC data transfers the internal Program Memory can be accessed independent of the contents of the DPP registers via the PEC source and destination pointers.

The internal Program Memory is not provided for single bit storage, and therefore it is not bit-addressable.

Note: The ‘x’ in the locations above depend on the available Program Memory and on the mapping.

The internal Program Memory may be enabled, disabled or mapped into segment 0 or segment 1 under software control. **Chapter 22** describes this and indicates precautions which must be taken to prevent system crashes.

3.2 Internal RAM and SFR Area

The IRAM/SFR area is located within data page 3 and provides access to the Internal RAM (IRAM, organized as X × 16) and to two 512-Byte blocks of Special Function Registers (SFRs).
The C164CI provides 2 KBytes of IRAM.

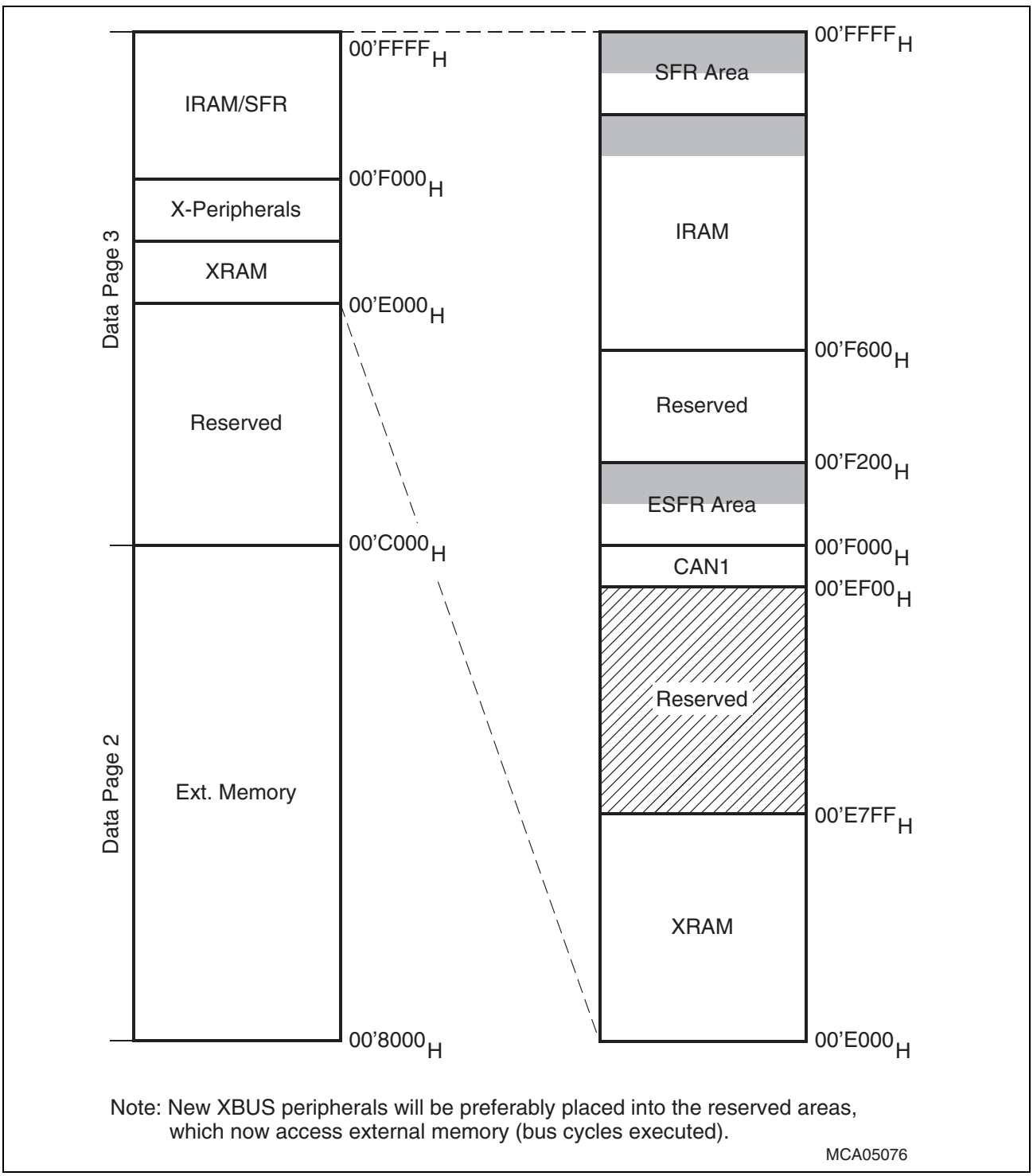


Figure 3-3 System Memory Map

*Note: The upper 256 Bytes of the SFR area, the ESFR area, and the Internal RAM are bit-addressable (see shaded blocks in **Figure 3-3**).*

Code accesses are always made on even byte addresses. The highest possible code storage location in the Internal RAM is either 00'FDFE$_H$ for single word instructions or 00'FDFC$_H$ for double word instructions. The respective location must contain a branch instruction (unconditional), because sequential boundary crossing from Internal RAM to the SFR area is not supported and causes erroneous results.

Any word and byte data in the Internal RAM 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 internal RAM is 00'FDFE$_H$. For PEC data transfers, the internal RAM can be accessed independent of the contents of the DPP registers via the PEC source and destination pointers.

The upper 256 Bytes of the Internal RAM (00'FD00$_H$ through 00'FDFF$_H$) and the GPRs of the current bank are provided for single bit storage, and thus they are bit-addressable.

System Stack

The system stack may be defined within the Internal RAM. The size of the system stack is controlled by bitfield STKSZ in register SYSCON (see **Table 3-1**).

Table 3-1 System Stack Size Encoding

<STKSZ>	Stack Size (words)	Internal RAM Addresses (words)
0 0 0$_B$	256	00'FBFE$_H$ … 00'FA00$_H$ (Default after Reset)
0 0 1$_B$	128	00'FBFE$_H$ … 00'FB00$_H$
0 1 0$_B$	64	00'FBFE$_H$ … 00'FB80$_H$
0 1 1$_B$	32	00'FBFE$_H$ … 00'FBC0$_H$
1 0 0$_B$	512	00'FBFE$_H$ … 00'F800$_H$
1 0 1$_B$	–	Reserved. Do not use this combination.
1 1 0$_B$	–	Reserved. Do not use this combination.
1 1 1$_B$	1024	00'FDFE$_H$ … 00'F600$_H$ (Note: No circular stack)

For all system stack operations the on-chip RAM is accessed via the Stack Pointer (SP) register. The stack grows downward from higher towards lower RAM address locations. Only word accesses are supported to the system stack. 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 destruction and for implementation of a circular stack with hardware-supported system stack flushing and filling (except for option '111'). The technique for implementing the circular stack is described in **Chapter 22**.

General Purpose Registers

The General Purpose Registers (GPRs) use a block of 16 consecutive words within the Internal RAM. The Context Pointer (CP) register determines the base address of the currently active register bank. 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 (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 RAM Addresses

Internal RAM Address	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 C164CI supports fast register bank (context) switching. Multiple register banks can physically exist within the Internal RAM at the same time. Only the register bank selected by the Context Pointer register (CP) is active at a given time, however. Selecting a new active register bank is done simply by updating the CP register. A particular Switch Context (SCXT) instruction performs register bank switching and automatically saves

the previous context. The number of implemented register banks (arbitrary sizes) is limited only by the size of the available internal RAM.
Details on using, switching, and overlapping register banks are described in **Chapter 22**.

PEC Source and Destination Pointers

The 16 word locations in the Internal RAM from 00'FCE0$_H$ to 00'FCFE$_H$ (just below the bit-addressable section) are provided as source and destination address pointers for data transfers on the eight PEC channels. 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).

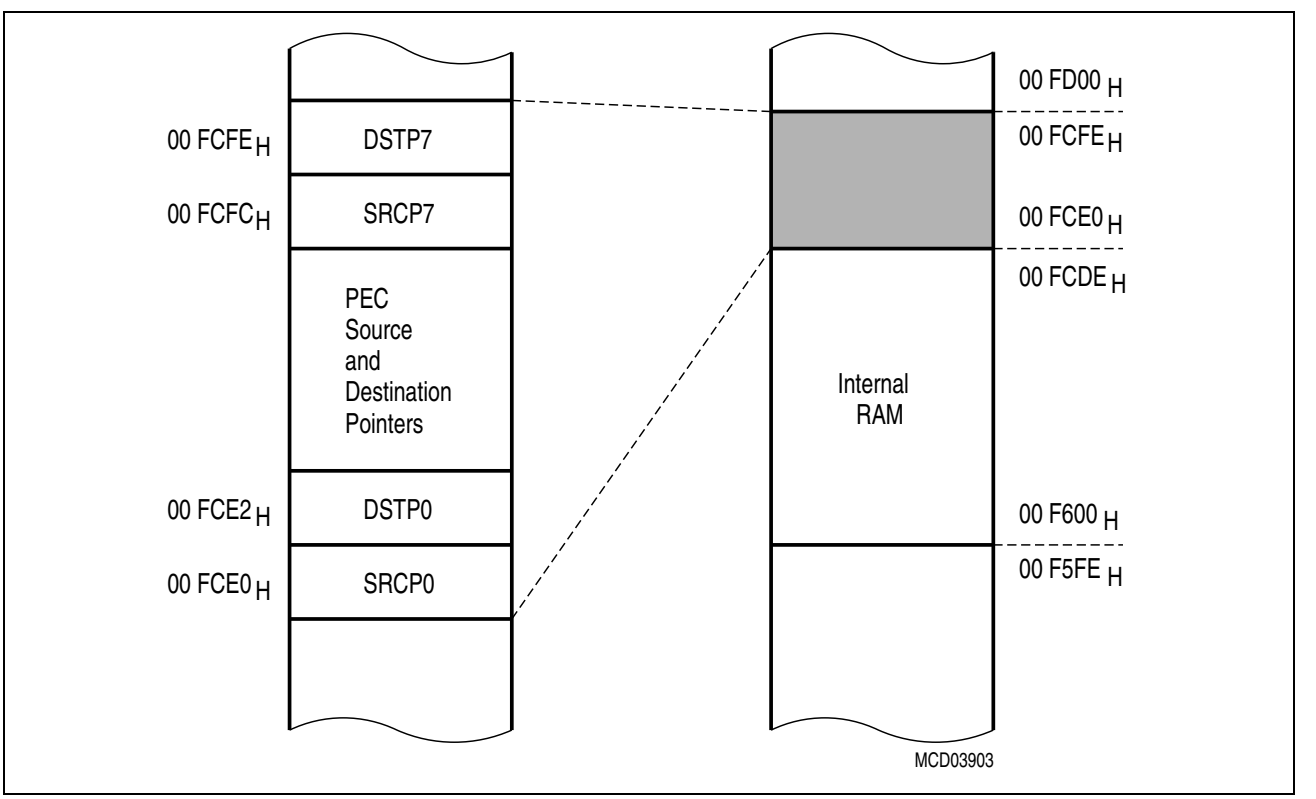


Figure 3-4 Location of the PEC Pointers

Whenever a PEC data transfer is performed, the pair of source and destination pointers (selected by the specified PEC channel number) is accessed independently of the current DPP register contents. The locations referred to by these pointers are accessed independently of the current DPP register contents as well. If a PEC channel is not used, the corresponding pointer locations are available and can be used for word or byte data storage.
For more details on the use of the source and destination pointers for PEC data transfers see **Chapter 5**.

Special Function Registers

The functions of the CPU, the bus interface, the IO ports, and the on-chip peripherals of the C164CI are controlled via a number of Special Function Registers (SFRs). These SFRs are arranged within two areas of 512 Bytes each. The first register block, the SFR area, is located in the 512 Bytes above the Internal RAM ($00'FFFF_H$ … $00'FE00_H$). The second register block, the Extended SFR (ESFR) area, is located in the 512 Bytes below the Internal RAM ($00'F1FF_H$ … $00'F000_H$).

Special Function Registers can be addressed via indirect and long 16-bit addressing modes. Using an 8-bit offset together with an implicit base address allows word SFRs and their respective low bytes to be addressed. However, this **does not work** for the respective high bytes!

Note: Writing to any byte of an SFR causes the non-addressed complementary byte to be cleared!

The upper half of each register block 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    ODP8, #data16        ;ODP2 uses 8-bit reg addressing
BFLDL  DP8, #mask, #data8   ;Bit addressing for bit fields
BSET   DP1H.7               ;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 primarily holds registers which are mainly required for initialization and mode selection. Registers which 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.

3.3 The On-Chip XRAM

The C164CI provides access to 2 KBytes of on-chip extension RAM. The XRAM is located within data page 3 (organized as 1 K × 16). As the XRAM is connected to the internal XBUS it is accessed like external memory, however, no external bus cycles are executed for these accesses. XRAM accesses are globally enabled or disabled via bit XPEN in register SYSCON. This bit is cleared after reset and may be set via software during the initialization to allow accesses to the on-chip XRAM. When the XRAM is disabled (default after reset) all accesses to the XRAM area are mapped to external locations. The XRAM may be used for both code (instructions) and data (variables, user stack, tables, etc.) storage.

Code fetches are always made on even byte addresses. The highest possible code storage location in the XRAM is either 00'E7FE$_H$ for single word instructions, or 00'E7FC$_H$ for double word instructions. The respective location must contain a branch instruction (unconditional), because sequential boundary crossing from XRAM to external memory is not supported and causes erroneous results.

Any word and byte data read accesses may use the indirect or long 16-bit addressing modes. There is no short addressing mode for XRAM operands. Any word data access is made to an even byte address. The highest possible word data storage location in the XRAM is 00'E7FE$_H$. For PEC data transfers the XRAM can be accessed independently of the contents of the DPP registers via the PEC source and destination pointers.

Note: As the XRAM appears like external memory it cannot be used for the C164CI's system stack or register banks. The XRAM is not provided for single bit storage and therefore is not bit-addressable.

The on-chip XRAM is accessed with the following bus cycles:

- Normal ALE
- No cycle time waitstates (no $\overline{\text{READY}}$ control)
- No tristate time waitstate
- No Read/Write delay
- 16-bit demultiplexed bus cycles (4 TCL)

Even though the XRAM is used like external memory it does not occupy BUSCONx/ ADDRSELx registers but rather is selected via additional dedicated XBCON/XADRS registers. These registers are mask-programmed and are not user accessible. With these registers the address area 00'E000$_H$ to 00'E7FF$_H$ is reserved for XRAM accesses.

3.4 External Memory Space

The C164CI is capable of using an address space of up to 16 MBytes. Only parts of this address space are occupied by internal memory areas. All addresses which are not used for on-chip memory (ROM/Flash/OTP or RAM) or for registers may reference external memory locations. This external memory is accessed via the C164CI's external bus interface.

Four memory bank sizes are supported:

- Non-segmented mode: 64 KBytes with A15 … A0 on PORT0 or PORT1
- 2-bit segmented mode: 256 KBytes with A17 … A16 on Port 4 and A15 … A0 on PORT0 or PORT1
- 4-bit segmented mode: 1 MByte with A19 … A16 on Port 4 and A15 … A0 on PORT0 or PORT1
- 6-bit segmented mode: 4 MBytes with A21 … A16 on Port 4 and A15 … A0 on PORT0 or PORT1

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

The C164CI also supports **four different bus types**:

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

Memory model and bus mode are selected during reset by pin $\overline{EA}$ and PORT0 pins. For further details about the external bus configuration and control please refer to **Chapter 9**.

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

For PEC data transfers the external memory in segment 0 can be accessed independently of the contents of the DPP registers via the PEC source and destination pointers.

The external memory is not provided for single bit storage and therefore it is not bitaddressable.

3.5 Crossing Memory Boundaries

The address space of the C164CI 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 Internal RAM/SFR area, the internal ROM/Flash/OTP (if available), the on-chip X-Peripherals (if integrated), and the external memory.

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

Note: Changing from the external memory area to the internal RAM/SFR 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.

3.6 Protection of the On-Chip Mask ROM

The on-chip mask ROM of the C164CI can be protected against read accesses of both code and data. ROM protection is established during the production process of the device (a ROM mask can be ordered with a ROM protection or without it). No software control is possible, i.e. the ROM protection cannot be disabled or enabled by software.

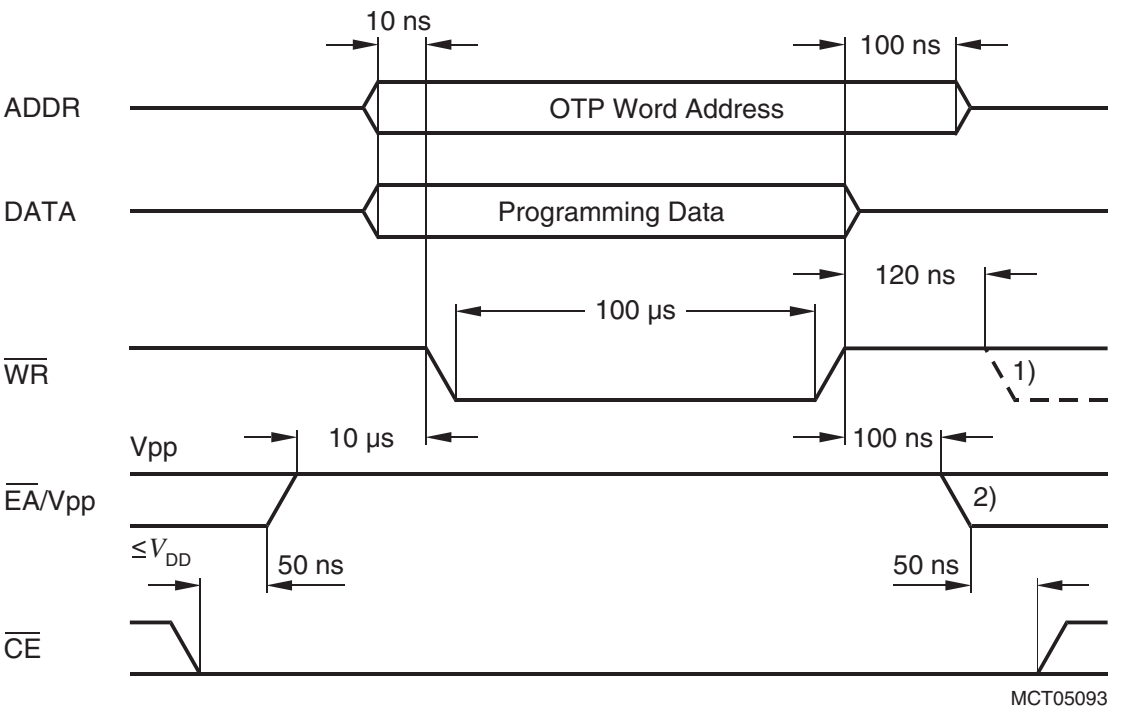
When a device has been produced with ROM protection active, the ROM contents are protected against unauthorized access by the following measures:

- **No data read accesses** to the internal ROM are permitted by any instruction which is executed from any location outside the on-chip mask ROM (including IRAM, XRAM, and external memory).
  A program cannot read any data out of the protected ROM from outside.
  The read data will be replaced by the default value $009B_H$ for any read access to any location.
- **No codes fetches** from the internal ROM can be made by any instruction which is executed from any location outside the on-chip mask ROM (including IRAM, XRAM, and external memory).
  A program cannot branch to a location within the protected ROM from outside. This applies to JUMPs as well as to RETurns. A called routine within RAM or external memory can never return to the protected ROM.
  The fetched code will be replaced by the default value $009B_H$ for any access to any location. This default value will be decoded as the instruction “TRAP #00” which will restart program execution at location $00'0000_H$.

Note: ROM protection may be used for applications where the complete software fits into the on-chip ROM, or where the on-chip ROM holds initialization software which is then replaced by external application software (for example). In the latter case no data (constants, tables, etc.) can be stored within the ROM. The ROM itself should be mapped to segment 1 before branching outside, so an interrupt vector table can be established in external memory.

3.7 OTP Memory Programming

During normal operation the One-Time-Programmable (OTP) memory appears like a standard ROM. In the special OTP programming modes, however, the OTP memory can be programmed by writing to its special programming interface. Programming is executed in units of 16-bit words and each programming cycle takes about 100 µs. OTP programming requires an external programming voltage of V_{PP} = 11.5 V ± 5% which is applied to pin $\overline{EA}/V_{PP}$.
The OTP memory can be programmed in CPU Host Mode (CHM) via software or in External Host Mode (EHM) via external hardware.



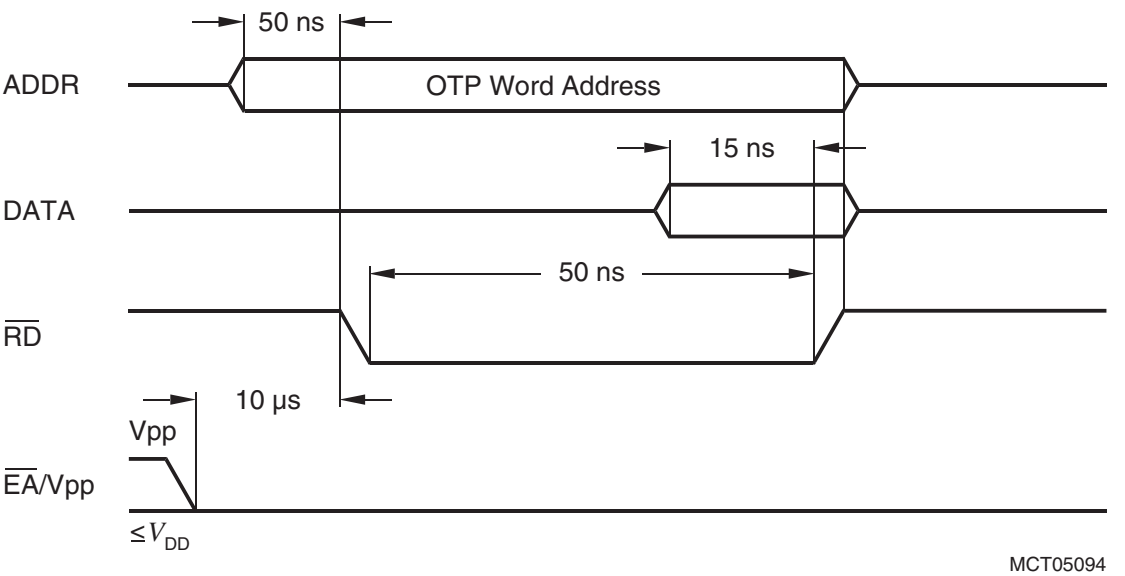
1) Earliest possible begin of next programming cycle.
2) V_{PP} must be switched off for verify accesses, it may remain on for subsequent programming cycles.
The special signal RSEL must fulfill the same timing requirements as the address lines.

Note: All timings represent minimum values.

Figure 3-5 OTP Programming Cycle

Verify cycles may be executed to ensure correct programming. Programming cycles and verify cycles may alternate in order to check each word immediately. However, the total programming time can be reduced by programming blocks of data continuously and then verifying the blocks (this saves the V_{PP} settling time).

*Note: The programming voltage V_{PP} **must be applied** for all programming cycles and **must be removed** for all other accesses, i.e. verify cycles and standard read cycles. The settling time is 10 µs in each case.*
In EHM this must be controlled by the external host, in CHM the CPU may control V_{PP} via an output port line.



The special signals $\overline{CE}$ and RSEL must fulfill the same timing requirements as the address lines.

Note: All timings represent minimum values.

Figure 3-6 OTP Verify/Read Cycle

The programming cycles can be controlled in two different ways:

In CPU Host Mode (CHM) the CPU of the C164CI itself controls the programming cycles via the OTP programming interface. The programming routine must be fetched from outside the OTP memory (on-chip RAM or external memory).

In External Host Mode (EHM) the C164CI is put into emulation mode where the CPU and the generic peripherals are disabled. The on-chip OTP memory can be accessed by an external master via the C164CI's bus interface. The bus interface signals change their direction in this mode.

3.7.1 Selecting an OTP Programming Mode

Both programming modes can only be enabled via reset configuration.

CPU Host Mode (CHM) is enabled after an external reset by pulling low pin P0L.2 in either standard startup mode or in bootstrap loader mode. Pins P0L.5 … 0 here represent $11'1011_B$ (standard) or $10'1011_B$ (BSL). After a single-chip mode reset CHM is automatically enabled without additional control.

Note: When CHM is enabled in standard startup mode program execution will always begin out of external memory, disregarding the level on pin $\overline{EA}/V_{PP}$.
When CHM is enabled in bootstrap loader mode the programming routine(s) can be loaded via the serial interface. This allows in-system programming of an empty OTP module.

External Host Mode (EHM) is enabled by selecting emulation mode (P0L.0 = ‘0’) and also pulling low pin P0L.5. Pins P0L.5 … 0 represent $01'1110_B$ in this case.

CPU Host Mode Programming

CHM is useful for in-system programming, especially combined with the bootstrap loader mode. CHM programming cycles are controlled via the C164CI’s programming interface which replaces the external bus interface signals. Pin $\overline{EA}/V_{PP}$ accepts the external programming voltage during programming cycles (see diagram).

The programming interface is realized as an XBUS peripheral and uses the address area $00'EDC0_H$ - $00'EDDF_H$. The interface is activated only in programming mode and cannot be accessed in all other cases. The OTP module’s interface signals are not externally asserted but rather controlled via three registers:

Table 3-3 OTP Programming Interface Registers

Register Name	Physical Address	Description	Reset Value
OPCTRL	$EDC0_H$	Control register, provides the control signals and the upper 8 address lines (A23 … A16).	0007_H
OPAD	$EDC2_H$	Address register provides the lower 15 address lines of the physical OTP word address (A15 … A1). **Note:** Address line A0 is not evaluated.	0000_H
OPDAT	$EDC4_H$	Data register provides the word to be stored or read from the module.	0000_H

External Host Mode Programming

In this mode the signals to control a programming cycle are generated by an external host using the C164CI's bus interface. The external host provides the data to be programmed. The C164CI itself is switched off and its OTP module can be accessed like standalone memory.

In External Host Mode the following port pins represent the interface to the C164CI's OTP module:

Table 3-4 External Host Mode Interface Signals[1)]

Signal	Pin	Description
ADDR	P1H.7 - P1L.1	Physical OTP word address (address line A0 is not evaluated)
DATA	P0H.7 - P0L.0	Word to be programmed or verified
$\overline{RD}$	$\overline{RD}$	Verify cycle control
$\overline{WR}$	$\overline{WR}$	Programming cycle control
$\overline{CE}$	P3.9	OTP enable signal
RSEL	P3.8	Control signal RSEL used for protection lock control, must be '0' for OTP programming cycles
–	P3.4, P3.6	Static high outputs
$\overline{RSTOUT}$	$\overline{RSTOUT}$	Must be held high (pullup resistor)
V_{PP}	$\overline{EA}/V_{PP}$	External programming voltage

1) The specific behavior of the C164CI in emulation mode (prerequisite for EHM) is described in **Section 20.4.1**.

The access cycles generated by the external host must fulfill the timing requirements shown in the timing diagrams above.

Note: EHM is a variety of the emulation mode where pin P0.15 (P0L.7) is inverted during the reset configuration. This influences the selected clock generation mode.
For EHM operation direct drive or prescaler mode must be configured. If the on-chip oscillator is not supplied with a clock signal the oscillator watchdog must not be disabled, so the PLL can provide the clock signal instead.

3.7.2 OTP Module Addressing

When the OTP module is read in normal mode (via its CPU interface) it appears like a standard ROM and its lower 32-KByte block within the internal ROM area can also be mapped to the respective lower half of segment 0 or segment 1:

For segment 0 mapping it uses locations $00'0000_H$ to $00'7FFF_H$ and $01'8000_H$ to $01'FFFF_H$,
for segment 1 mapping it uses locations $01'0000_H$ to $01'FFFF_H$.

In programming mode, however, the OTP module is addressed physically via the external interface or the OTP programming interface. In this case the OTP module appears as a contiguous block using the (physical) addresses $00'0000_H$ to $00'FFFF_H$.

Note: When entering a programming mode (EHM or CHM) the on-chip OTP module is disabled independent from the selection via pin $\overline{EA}$. The programming software (in CHM) must not enable the OTP module's CPU interface by setting bit ROMEN in register SYSCON.

OPCTRL
OTP Control Register **XReg ($EDC0_H$)** **Reset Value: 0007_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SEGAD								-	-	-	-	RS	CEQ	WRQ	RDQ
rw								-	-	-	-	rw	rw	rw	rw

Bit	Function
RDQ	**Read Signal** (active low) 0: OTP module selected for a verify read access 1: Read access is completed
WRQ	**Write Signal** (active low) 0: OTP module selected for a write access (programming) 1: Write access is completed
CEQ	**OTP Module Enable Signal** (active low) 0: OTP module is selected 1: OTP module is deselected, no access
RS	**Register Select Signal** (RSEL) 0: Access the OTP memory module 1: Access the control section (read protection control)
SEGAD	**Physical Segment Address** Provides the upper (physical) address lines (A23 … A16) to the OTP memory module (SEGAD must be 00_H for the C164CI)

An OTP programming/verify cycle is executed by a sequence of accesses to the programming interface which emulate the externally controlled cycles (see example below).

OTP Programming Example

The on-chip OTP memory is programmed in CHM executing the following procedure:

Note: The example below assumes segment 0 (RH3 = 00_H).

```
MOV    R1, #OTP_START      ;R1 = OTP pointer
MOV    R2, #DATA_BLOCK     ;R2 = Source data pointer

MOV    R3, #0003H          ;03H: enable module, cmd. idle
MOV    DPP3:OPCTRL, R3     ;Initially enable the OTP module
BSET   VPP_ENABLE          ;External progr. voltage ON
CALL   MICROSEC_010        ;Let VPP settle for 10 µs

PROG_OTP_WORD:
MOV    DPP3:OPAD, R1       ;Select current address
MOV    R0, [R2+]           ;Move source data word …
MOV    DPP3:OPDAT, R0      ;… to data register
MOV    R3, #0001H          ;01H: enable module, WR active
MOV    DPP3:OPCTRL, R3     ;Select OTP module for write access
CALL   MICROSEC_100        ;Keep the write signal low for 100 µs
MOV    R3, #0003H          ;03H: enable module, cmd. idle
MOV    DPP3:OPCTRL, R3     ;Trailing edge of write signal

ALT_VERIFY:                ;This block only for alternating verify
BCLR   VPP_ENABLE          ;External progr. voltage Off
CALL   MICROSEC_010        ;Let VPP settle for 10 µs
MOV    R3, #0002H          ;02H: enable module, RD active
MOV    DPP3:OPCTRL, R3     ;Select OTP module for read access
MOV    R3, #0003H          ;03H: enable module, cmd. idle
MOV    DPP3:OPCTRL, R3     ;Trailing edge of read signal
CMP    R0, DPP3:OPDAT      ;Verify data reg. with original data
JMP    cc_NE, PROG_FAILED
BSET   VPP_ENABLE          ;External progr. voltage ON
CALL   MICROSEC_010        ;Let VPP settle for 10 µs
```

```
PROG_LOOP:
CMPI2   R1, #BLOCK_LIMIT    ;Next OTP location
JMP     cc_ULE, PROG_OTP_WORD;Repeat for the whole data block

BCLR    VPP_ENABLE          ;External progr. voltage Off
CALL    MICROSEC_010        ;Let VPP settle for 10 µs

                            ;Block verification could be …
                            ;… executed here

MOV     R3, #0007H
MOV     DPP3:OPCTRL, R3     ;OTP module deselected
```

3.7.3 Read Protection Control

The on-chip OTP memory can be protected against unauthorized accesses (read or execute).

When the read protection is active …

- no programming cycles can be executed (neither in EHM nor in CHM)
- no verify cycles can be executed
- OTP locations can only be read by instructions fetched from the OTP itself

The OTP read protection is activated by a specific programming cycle which has the register select signal (RSEL) active (contrary to normal programming cycles). This special cycle must write the value 0000_H to register address $000E_H$. A verify cycle can be executed directly after activating the read protection, i.e. without leaving programming mode. The active read protection is indicated with data bit D0 = '0'.

Note: OTP read protection is irreversible. When the OTP read protection was activated once it remains active for each and every subsequent access. Also subsequent programming cycles are no more possible.

OTP Read Protection Example

The OTP read protection is activated in CHM executing the following procedure:

Note: The example below assumes segment 0 (RH0 = 00_H).

```
MOV    R0, #0003H          ;Enable module, cmd. idle
MOV    DPP3:OPCTRL, R0     ;Initially enable the OTP module
BSET   VPP_ENABLE          ;External progr. voltage ON
CALL   MICROSEC_010        ;Let VPP settle for 10 µs
MOV    R0, #000EH          ;Move special register address …
MOV    DPP3:OPAD, R0       ;… to address register
MOV    R0, #0000H          ;Move special control word …
MOV    DPP3:OPDAT, R0      ;… to data register
MOV    R0, #0009H          ;Select special OTP register …
MOV    DPP3:OPCTRL, R0     ;… for write access
CALL   MICROSEC_100        ;Keep the write signal low for 100 µs
MOV    R0, #000BH
MOV    DPP3:OPCTRL, R0     ;Trailing edge of write signal
BCLR   VPP_ENABLE          ;External progr. voltage Off
CALL   MICROSEC_010        ;Let VPP settle for 10 µs

                           ;Read protection verify could be …
                           ;… executed here

MOV    R0, #0007H
MOV    DPP3:OPCTRL, R0     ;OTP module deselected
```

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), to perform operations on these operands in the ALU, and to store the previously calculated results. As the CPU is the main engine of the C164CI microcontroller, it is also affected by certain actions of the peripheral subsystem.

Because a four stage pipeline is implemented in the C164CI, up to four instructions can be processed in parallel. Most instructions of the C164CI are executed in one machine cycle (2 CPU clock periods) 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 and exceptional timing.

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.

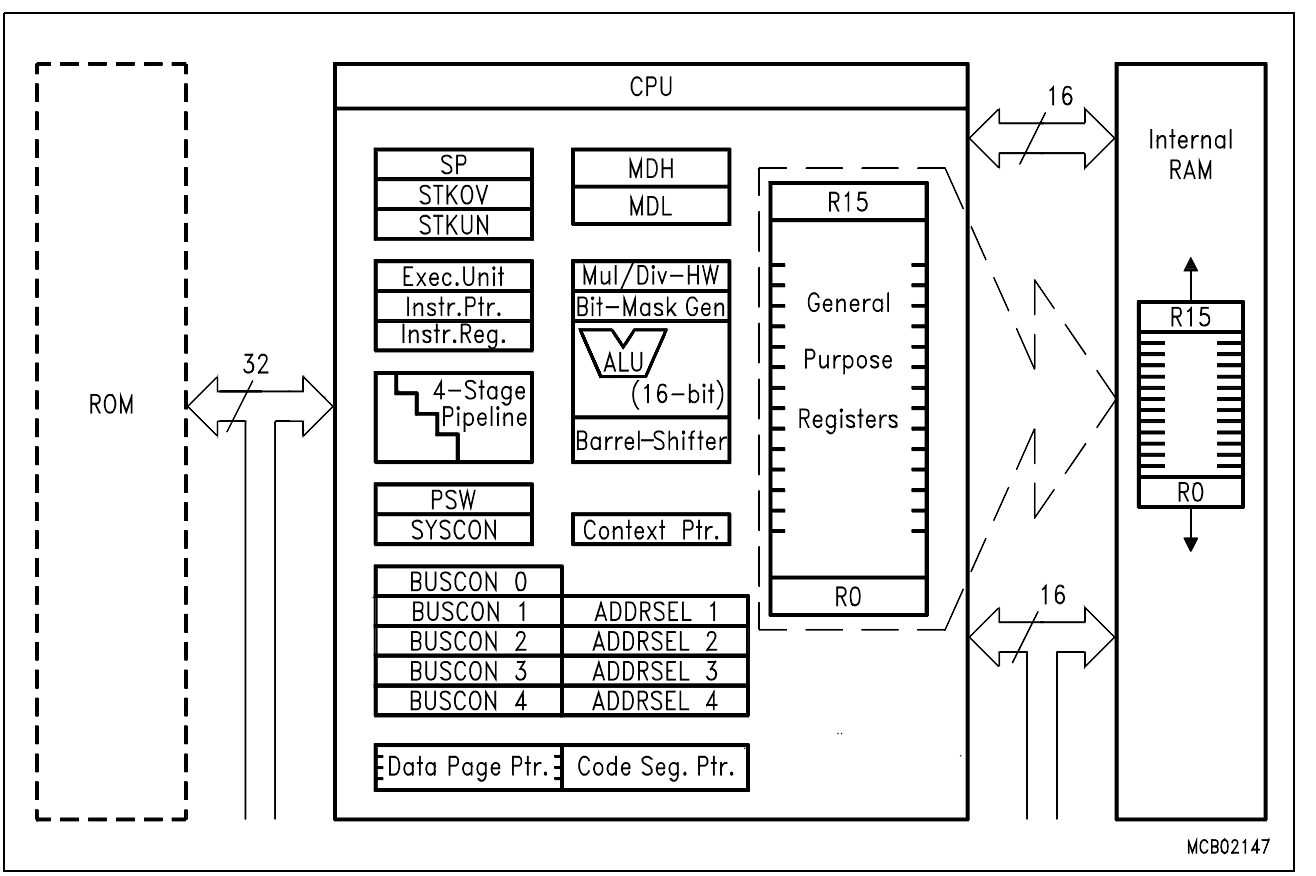


Figure 4-1 CPU Block Diagram

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 C164CI 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 C164CI system control instructions.

A set of Special Function Registers is dedicated to the functions of the CPU core:

- General System Configuration : **SYSCON (RP0H)**
- CPU Status Indication and Control : **PSW**
- Code Access Control : **IP, CSP**
- Data Paging Control : **DPP0, DPP1, DPP2, DPP3**
- GPRs Access Control : **CP**
- System Stack Access Control : **SP, STKUN, STKOV**
- Multiply and Divide Support : **MDL, MDH, MDC**
- ALU Constants Support : **ZEROS, ONES**

4.1 Instruction Pipelining

The instruction pipeline of the C164CI partitions instruction processing into four stages, each of which has a specific task:

1st → FETCH: In this stage, the instruction selected by the Instruction Pointer (IP) and the Code Segment Pointer (CSP) is fetched from either the internal ROM, internal RAM, or external memory.

2nd → DECODE: In this stage, the instructions are decoded and, if required, the operand addresses are calculated and the respective operands are fetched. For all instructions which implicitly access the system stack, the SP register is either decremented or incremented as specified. For branch instructions, the Instruction Pointer and the Code Segment Pointer are updated with the desired branch target address (provided that the branch is taken).

3rd → EXECUTE: In this stage, an operation is performed on the previously fetched operands in the ALU. Additionally, the condition flags in the PSW register are updated as specified by the instruction. Also, all explicit writes to the SFR memory space and all auto-increment or auto-decrement writes to GPRs used as indirect address pointers are performed during the execute stage of an instruction.

4th → WRITE BACK: In this stage, all external operands and the remaining operands within the internal RAM space are written back.

A special feature of the C164CI is the use of so-called injected instructions. Injected instructions are generated internally by the machine to provide the time needed to process instructions which cannot be processed within one machine cycle. These instructions are injected automatically into the decode stage of the pipeline and then they pass through the remaining stages like every standard instruction. Program interrupts are performed by means of injected instructions, as well. Although these internally injected instructions will not be noticed in reality, they are introduced here to ease the explanation of the pipeline in the following sections.

Sequential Instruction Processing

Each single instruction must pass through each of the four pipeline stages regardless of whether or not all possible stage operations are actually performed. Because passing through one pipeline stage takes at least one machine cycle, any isolated instruction takes at least four machine cycles to be completed. Pipelining, however, allows parallel (simultaneous) processing of up to four instructions. Thus, most of the instructions seem to be processed in one machine cycle as soon as the pipeline has been filled once after reset (see **Figure 4-2**).

Instruction pipelining increases the average instruction throughput considered over a certain period of time. In the following, any execution time specification for an instruction always refers to the average execution time due to pipelined parallel instruction processing.

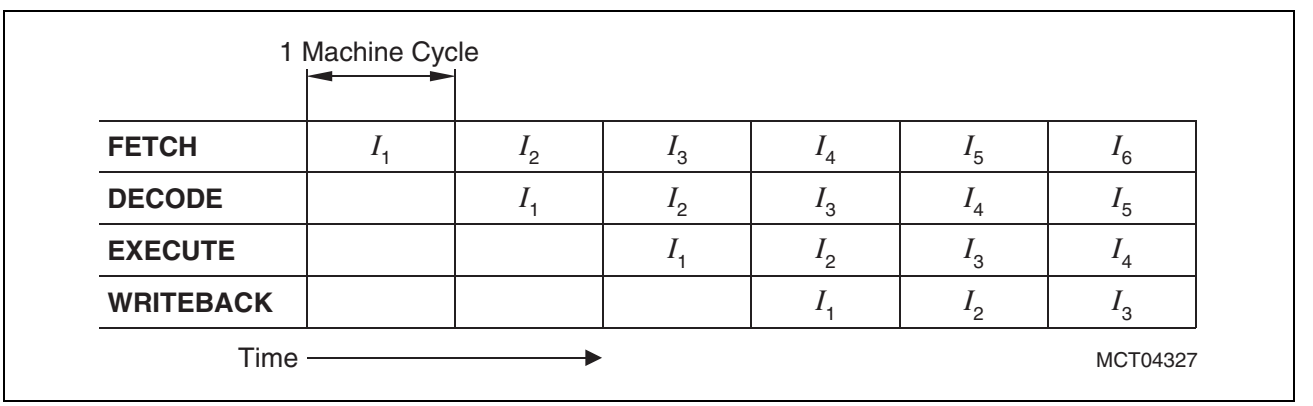


Figure 4-2 Sequential Instruction Pipelining

Standard Branch Instruction Processing

Instruction pipelining helps to speed up sequential program processing. If a branch is taken, the instruction which has already been fetched is most likely not the instruction which must be decoded next. Thus, at least one additional machine cycle is normally required to fetch the branch target instruction. This extra machine cycle is provided by means of an injected instruction (see **Figure 4-3**).

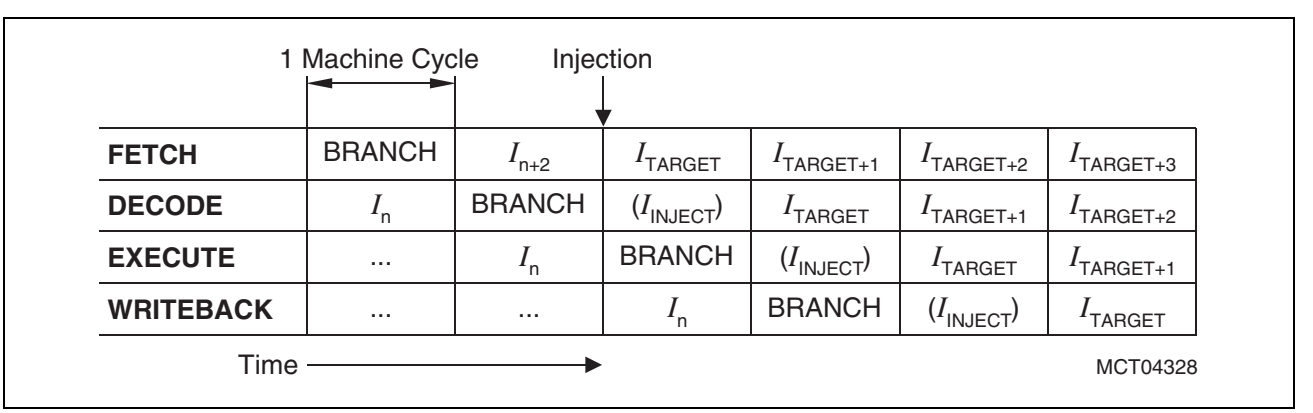


Figure 4-3 Standard Branch Instruction Pipelining

If a conditional branch is not taken, there is no deviation from the sequential program flow, and thus no extra time is required. In this case, the instruction after the branch instruction will enter the decode stage of the pipeline at the beginning of the next machine cycle after the decoding of the conditional branch instruction.

Cache Jump Instruction Processing

The C164CI incorporates a jump cache to optimize conditional jumps which are processed repeatedly within a loop. Whenever a jump on cache is taken, the extra time to fetch the branch target instruction can be saved and thus the corresponding cache jump instruction in most cases takes only one machine cycle.

This performance is achieved by the following mechanism:
Whenever a cache jump instruction passes through the decode stage of the pipeline for the first time (provided that the jump condition is met), the jump target instruction is fetched as usual, causing a time delay of one machine cycle. In contrast to standard branch instructions, however, the target instruction of a cache jump instruction (JMPA, JMPR, JB, JBC, JNB, JNBS) is additionally stored in the cache after having been fetched.

After each repeatedly following execution of the same cache jump instruction, the jump target instruction is not fetched from program memory but, rather, is taken from the cache and is injected immediately into the decode stage of the pipeline (see **Figure 4-4**).

A time saving jump on cache is always taken after the second and any further occurrence of the same cache jump instruction unless an instruction having the fundamental capability of changing the CSP register contents (JMPS, CALLS, RETS, TRAP, RETI), or any standard interrupt has been processed during the period of time between two following occurrences of the same cache jump instruction.

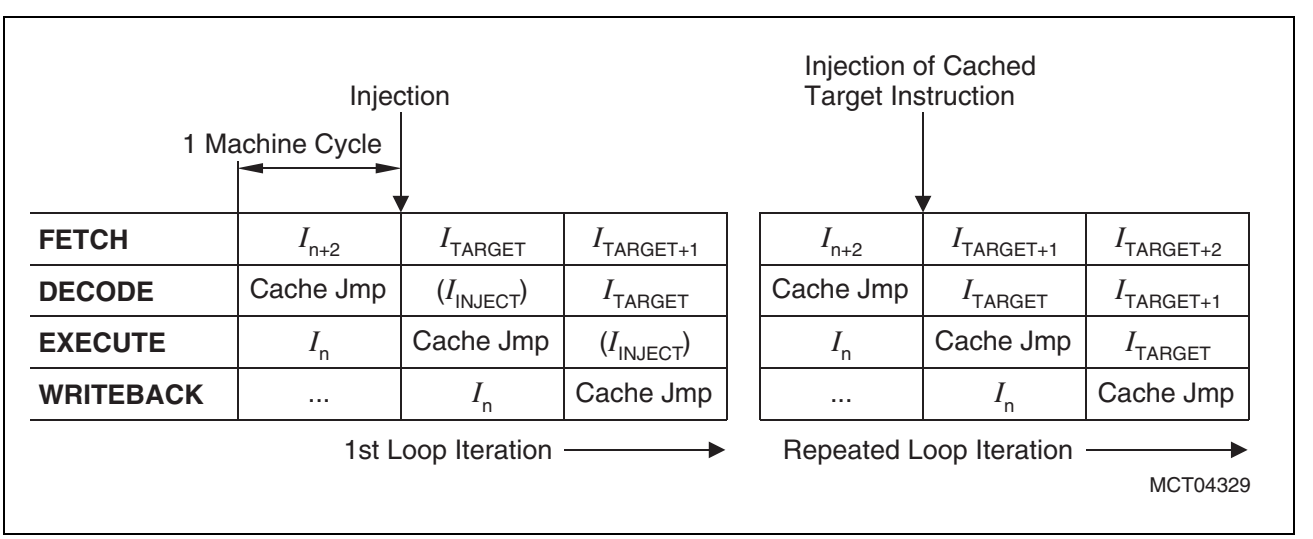


Figure 4-4 Cache Jump Instruction Pipelining

4.2 Particular Pipeline Effects

Because up to four different instructions are processed simultaneously, additional hardware has been used in the C164CI to consider all causal dependencies which may exist on instructions in different pipeline stages. This functionality is provided without a loss of performance. This extra hardware (for ‘forwarding’ operand read and write values) resolves most of the possible conflicts (such as multiple usage of buses) in a time optimized way; thus, in most cases, the pipeline operates without being noticeable to the user. However, there are some very rare circumstances in which the C164CI as a pipelined machine requires attention by the programmer. In these cases, the delays caused by pipeline conflicts can be used for other instructions in order to optimize performance.

Context Pointer Updating

An instruction which calculates a physical GPR operand address via the Context Pointer (CP) register is mostly incapable of using a new CP value, which has been updated by an immediately preceding instruction. Thus, to ensure that the new CP value is used, at least one instruction must be inserted between a CP-changing instruction and a subsequent GPR-using instruction, as shown in the following example:

```
I_n       :SCXT CP,#0FC00h   ;select a new context
I_n+1     :…                 ;must not be an instruction using a GPR
I_n+2     :MOV  R0,#dataX    ;write to GPR 0 in the new context
```

Data Page Pointer Updating

An instruction which calculates a physical operand address via a particular Data Page Pointer (DPPn) register (n = 0 to 3), is mostly incapable of using a new DPPn register value which has been updated by an immediately preceding instruction. Thus, to ensure that the new DPPn register value is used, at least one instruction must be inserted between a DPPn-changing instruction and a subsequent instruction which implicitly uses DPPn via a long or indirect addressing mode, as shown in the following example:

```
I_n       :MOV  DPP0,#4      ;select data page 4 via DPP0
I_n+1     :…                 ;must not be an instruction using DPP0
I_n+2     :MOV  DPP0:0000H,R1;move contents of R1 to
                             ;location 01'0000H(in data page 4),
                             ;supposed segment is enabled
```

Explicit Stack Pointer Updating

None of the RET, RETI, RETS, RETP or POP instructions is capable of correctly using a new SP register value which has been updated by an immediately preceding instruction. Thus, in order to use the new SP register value without erroneously performed stack accesses, at least one instruction must be inserted between an explicitly SP-writing and any subsequent use of the just mentioned implicitly SP-using instructions, as shown in the following example:

```
In      :MOV  SP,#0FA40H   ;select a new top of stack
In+1    :…                 ;must not be an instruction popping
                           ;operands from the system stack
In+2    :POP  R0           ;pop word value from new top of stack
                           ;into R0
```

Note: Conflicts with instructions writing to the stack (PUSH, CALL, SCXT) are solved internally by the CPU logic.

Controlling Interrupts

Software modifications (implicit or explicit) of the PSW are made in the execute phase of the respective instructions. To maintain fast interrupt responses, however, the current interrupt prioritization round does not consider these changes; that means that an interrupt request may be acknowledged after the instruction which disables interrupts via IEN or ILVL or after the following instructions. Time critical instruction sequences therefore should not begin directly after the instruction disabling interrupts, as shown in the following examples:

```
INTERRUPTS_OFF:
BCLR   IEN                 ;globally disable interrupts
<Instr non-crit>           ;non-critical instruction
<Instr 1st-crit>           ;begin of
. . .                      ;uninterruptable critical sequence
<Instr last-crit>          ;end of critical sequence
INTERRUPTS_ON:
BSET   IEN                 ;globally re-enable interrupts
CRITICAL_SEQUENCE:
ATOMIC #3                  ;immediately block interrupts
BCLR   IEN                 ;globally disable interrupts
. . .                      ;here is the uninterruptable sequence
BSET   IEN                 ;globally re-enable interrupts
```

Note: The described delay of one instruction also applies for enabling the interrupts system i.e. no interrupt requests are acknowledged until the instruction following the enabling instruction.

External Memory Access Sequences

The effect described here will become noticeable only when watching the external memory access sequences on the external bus (by means of a Logic Analyzer). Different pipeline stages can simultaneously put a request on the External Bus Controller (EBC). The sequence of instructions processed by the CPU may differ from the sequence of the corresponding external memory accesses performed by the EBC due to the predefined priority of external memory accesses:

1st Write Data
2nd Fetch Code
3rd Read Data

Initialization of Port Pins

Direction modification of port pins (input or output) become effective only after the instruction following the modifying instruction. As bit instructions (BSET, BCLR) use internal read-modify-write sequences which access the entire port, instructions which modify the port direction should be followed by an instruction that does not access the same port (see example below).

```
PORT_INIT_WRONG:
BSET    DP3.13              ;change direction of P3.13 to output
BSET    P3.9                ;P3.13 is still input,
                            ;rd-mod-wr reads pin P3.13
PORT_INIT_RIGHT:
BSET    DP3.13              ;change direction of P3.13 to output
NOP                         ;any instruction not accessing port 3
BSET    P3.9                ;P3.13 is now output,
                            ;rd-mod-wr reads P3.13's output latch
```

Note: Special attention must be paid to interrupt service routines that modify the same port as the software they have interrupted.

Changing the System Configuration

The instruction following an instruction that changes the system configuration via register SYSCON (e.g. the mapping of the internal ROM, segmentation, stack size) cannot use the new resources (e.g. ROM or stack). In these cases, an instruction which does not access these resources should be inserted. Code accesses to the new ROM area are possible only after an absolute branch to this area.

Note: As a rule, instructions that change ROM mapping should be executed from internal RAM or external memory.

BUSCON/ADDRSEL

The instruction following an instruction that changes the properties of an external address area cannot access operands within the new area. In these cases, an instruction that does not access this address area should be inserted. Code accesses to the new address area should be made after an absolute branch to this area.

Note: As a rule, instructions that change external bus properties should not be executed from the respective external memory area.

Timing

Instruction pipelining generally reduces the average instruction processing time significantly (from four to one machine cycles). However, there are some rare cases in which a particular pipeline situation causes the processing time for a single instruction to be extended either by one-half or by one machine cycle. Although this additional time represents only a tiny part of the total program execution time, it might be beneficial to avoid these pipeline-caused time delays in time-critical program modules.

Section 4.3 below provides a general execution time description and some hints on optimizing time-critical program parts with regard to such pipeline-caused timing issues.

4.3 Bit-Handling and Bit-Protection

The C164CI provides several mechanisms for bit manipulation. These mechanisms 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.

The instructions BSET, BCLR, BAND, BOR, BXOR, BMOV, BMOVN explicitly set or clear specific bits. The 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:

- 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 the SFR area, the ESFR area, and the internal RAM are bit-addressable (see **Chapter 3**); 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.

- 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 4.2**).

Protected bits are not changed during the read-modify-write sequence, such as when hardware sets an interrupt request flag between the read and the write of the read-modify-write sequence. The hardware protection logic guarantees that only the intended bit(s) is/are affected by the write-back operation.

Note: If a conflict occurs between a bit manipulation generated by hardware and an intended software access, the software access has priority and determines the final value of the respective bit.

A summary of the protected bits implemented in the C164CI can be found at the end of **Chapter 2**.

4.4 Instruction State Times

The time to execute an instruction depends primarily on where the instruction is fetched from, and where the possible operands are read from or written to. The fastest processing mode of the C164CI is execution of a program fetched from the internal code memory. In this case, most of the instructions can be processed within just one machine cycle, which is also the general minimum execution time.

All external memory accesses are performed by the C164CI's on-chip External Bus Controller (EBC), which works in parallel with the CPU.

This section provides a very condensed summary of the execution times. A detailed description of the execution times for the various instructions and the specific exceptions can be found in the **"C166 Family Instruction Set Manual"**.

Table 4-1 shows the minimum execution times required to process a C164CI instruction fetched from the internal code memory, the internal RAM, or from external memory. These execution times apply to most of the C164CI instructions - except for some of the branches, the multiplication, the division, and a special move instruction. In case of internal ROM program execution, there is no execution time dependency on the instruction length except for some special branch situations. The numbers in the table are in units of CPU clock cycles and assume no waitstates.

Table 4-1 Minimum Execution Times

	Instruction Fetch		Word Operand Access	
Memory Area	**Word Instruction**	**Doubleword Instruction**	**Read from**	**Write to**
Internal code memory	2	2	2	---
Internal RAM	6	8	0/1	0
16-bit Demux Bus	2	4	2	2
16-bit Mux Bus	3	6	3	3
8-bit Demux Bus	4	8	4	4
8-bit Mux Bus	6	12	6	6

Execution from the internal RAM provides flexibility in terms of loadable and modifyable code on the account of execution time.
Execution from external memory is heavily dependent on the selected bus mode and the programming of the bus cycles (waitstates).

The operand and instruction accesses listed below can extend the execution time of an instruction:

- Internal code memory operand reads (same for byte and word operand reads)
- Internal RAM operand reads via indirect addressing modes
- Internal SFR operand reads immediately after writing
- External operand reads
- External operand writes
- Jumps to non-aligned double word instructions in the internal ROM space
- Testing Branch Conditions immediately after PSW writes

4.5 CPU Special Function Registers

The core CPU requires a set of Special Function Registers (SFRs) to maintain the system state information, to supply the ALU with register-addressable constants, and to control system and bus configuration, multiply and divide ALU operations, code memory segmentation, data memory paging, and accesses to the General Purpose Registers and the System Stack.

The access mechanism for these SFRs in the CPU core is identical to the access mechanism for any other SFR. Since all SFRs can be controlled simply by means of any instruction capable of addressing the SFR memory space, significant flexibility has been gained without the need to create a set of system-specific instructions.

Note, however, that there are user access restrictions for some of the CPU core SFRs to ensure proper processor operations. The Instruction Pointer (IP) and Code Segment Pointer (CSP) cannot be accessed directly at all. They can be changed only indirectly via branch instructions.

The PSW, SP, and MDC registers can be modified not only explicitly by the programmer, but also implicitly by the CPU during normal instruction processing. Note that any explicit write request (via software) to an SFR supersedes a simultaneous modification by hardware of the same register.

Note: Any write operation to a single byte of an SFR clears the non-addressed complementary byte within the specified SFR.
Non-implemented (reserved) SFR bits cannot be modified and will always supply a read value of ‘0’.

System Configuration Register SYSCON

This bit-addressable register provides general system configuration and control functions. The reset value for register SYSCON depends on the state of the PORT0 pins during reset (see hardware effectable bits).

SYSCON
System Control Register **SFR ($FF12_H/89_H$)** **Reset Value: $0XX0_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STKSZ			ROM S1	SGT DIS	ROM EN	BYT DIS	CLK EN	WR CFG	CS CFG	-	OWD DIS	BD RST EN	XPEN	VISI-BLE	-
rw			rw	rw	rwh	rwh	rw	rwh	rw	-	rwh	rw	rw	rw	-

Bit	Function
VISIBLE	**Visible Mode Control** 0: Accesses to XBUS peripherals are done internally 1: XBUS peripheral accesses are made visible on the external pins
XPEN	**XBUS Peripheral Enable Bit** 0: Accesses to the on-chip X-Peripherals and their functions are disabled 1: The on-chip X-Peripherals are enabled and can be accessed
BDRSTEN	**Bidirectional Reset Enable Bit** 0: Pin $\overline{\text{RSTIN}}$ is an input only 1: Pin $\overline{\text{RSTIN}}$ is pulled low during the internal reset sequence after any reset
OWDDIS	**Oscillator Watchdog Disable Bit** 0: The on-chip oscillator watchdog is enabled and active 1: The on-chip oscillator watchdog is disabled and the CPU clock is always fed from the oscillator input
CSCFG	**Chip Select Configuration Control** 0: Latched $\overline{\text{CS}}$ mode. The $\overline{\text{CS}}$ signals are latched internally and driven to the (enabled) port pins synchronously 1: Unlatched $\overline{\text{CS}}$ mode. The $\overline{\text{CS}}$ signals are derived directly from the address and driven to the (enabled) port pins
WRCFG	**Write Configuration Control** (Set according to pin P0H.0 during reset) 0: Pins $\overline{\text{WR}}$ and $\overline{\text{BHE}}$ retain their normal function 1: Pin $\overline{\text{WR}}$ acts as $\overline{\text{WRL}}$, pin $\overline{\text{BHE}}$ acts as $\overline{\text{WRH}}$

Bit	Function
CLKEN	**System Clock Output Enable (CLKOUT)** 0: CLKOUT disabled: pin may be used for general purpose IO or for signal FOUT 1: CLKOUT enabled: pin outputs the system clock signal
BYTDIS	**Disable/Enable Control for Pin $\overline{\text{BHE}}$** (Set according to data bus width) 0: Pin $\overline{\text{BHE}}$ enabled 1: Pin $\overline{\text{BHE}}$ disabled, pin may be used for general purpose IO
ROMEN	**Internal ROM Enable** (Set according to pin $\overline{\text{EA}}$ during reset) 0: Internal program memory disabled, accesses to the ROM area use the external bus 1: Internal program memory enabled
SGTDIS	**Segmentation Disable/Enable Control** 0: Segmentation enabled (CSP is saved/restored during interrupt entry/exit) 1: Segmentation disabled (Only IP is saved/restored)
ROMS1	**Internal ROM Mapping** 0: Internal ROM area mapped to segment 0 ($00'0000_H$ … $00'7FFF_H$) 1: Internal ROM area mapped to segment 1 ($01'0000_H$ … $01'7FFF_H$)
STKSZ	**System Stack Size** Selects the size of the system stack (in the internal RAM) from 32 to 512 words

Note: Register SYSCON cannot be changed after execution of the EINIT instruction. The function of bits VISIBLE, WRCFG, BYTDIS, ROMEN and ROMS1 is described in more detail in ***Chapter 9****.*

System Clock Output Enable (CLKEN)

The system clock output function is enabled by setting bit CLKEN in register SYSCON to ‘1’. If enabled, port pin P3.15 takes on its alternate function as CLKOUT output pin. The clock output is a 50% duty cycle clock (except for direct drive operation where CLKOUT reflects the clock input signal, and for slowdown operation where CLKOUT mirrors the CPU clock signal) whose frequency equals the CPU operating frequency ($f_{OUT} = f_{CPU}$).

Note: The output driver of port pin P3.15 is switched on automatically when the CLKOUT function is enabled. The port direction bit is disregarded.
After reset, the clock output function is disabled (CLKEN = ‘0’).
In emulation mode, the CLKOUT function is enabled automatically.

Segmentation Disable/Enable Control (SGTDIS)

Bit SGTDIS allows selection of either the segmented or non-segmented memory mode.
In non-segmented memory mode (SGTDIS = ‘1’), it is assumed that the code address space is restricted to 64 KBytes (segment 0); thus, 16 bits are sufficient to represent all code addresses. For implicit stack operations (CALL or RET), the CSP register is totally ignored and only the IP is saved to and restored from the stack.
In segmented memory mode (SGTDIS = ‘0’) it is assumed that the entire address space is available for instructions. For implicit stack operations (CALL or RET) the CSP register and the IP are saved to and restored from the stack. After reset, the segmented memory mode is selected.

Note: Bit SGTDIS controls whether the CSP register is pushed onto the system stack in addition to the IP register before an interrupt service routine is entered, and it is repopped when the interrupt service routine is left again.

System Stack Size (STKSZ)

This bitfield defines the size of the physical system stack located in the internal RAM of the C164CI. An area of 32 … 512 words or all of the internal RAM may be dedicated to the system stack. A so-called “circular stack” mechanism allows use of a bigger virtual stack than this dedicated RAM area.

These techniques and the encoding of bitfield STKSZ are described in more detail in **Chapter 22**.

Processor Status Word PSW

This bit-addressable register reflects the current state of the microcontroller. Two groups of bits represent the current ALU status, and the current CPU interrupt status. A separate bit (USR0) within register PSW is provided as a general purpose user flag.

PSW
Program 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	-	-	-	-	USR0	MUL IP	E	Z	V	C	N
rwh				rw	-	-	-	-	rw	rwh	rwh	rwh	rwh	rwh	rwh

Bit	Function
N	**Negative Result** Set, when the result of an ALU operation is negative.
C	**Carry Flag** Set, when the result of an ALU operation produces a carry bit.
V	**Overflow Result** Set, when the result of an ALU operation produces an overflow.
Z	**Zero Flag** Set, when the result of an ALU operation is zero.
E	**End of Table Flag** Set, when the source operand of an instruction is 8000_H or 80_H.
MULIP	**Multiplication/Division In Progress** 0: There is no multiplication/division in progress. 1: A multiplication/division has been interrupted.
USR0	**User General Purpose Flag** May be used by the application software.
ILVL, IEN	**Interrupt Control Fields** Define the response to interrupt requests. (Described in **Chapter 5**)

ALU Status (N, C, V, Z, E, MULIP)

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 0 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 overflows the maximum range of signed numbers which are representable by either 16 bits for word operations ('-8000_H' to '$+7FFF_H$'), or by 8 bits 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 4-2**).
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 4-2 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 additionally 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 prioritized ALU operation, the Z-flag indicates whether the second operand was zero.

E-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 is set depending 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.

MULIP-Flag: The MULIP-flag will be set to ‘1’ by hardware upon entrance into an interrupt service routine when a multiply or divide ALU operation was interrupted before completion. Depending on the state of the MULIP bit, the hardware decides whether or not a multiplication or division must be continued after the end of an interrupt service. The MULIP bit is overwritten with the contents of the stacked MULIP-flag when the return-from-interrupt-instruction (RETI) is executed. This normally means that the MULIP-flag is cleared again afterwards.

Note: The MULIP flag is part of the task environment! When the interrupting service routine does not return to the interrupted multiply/divide instruction (as in the case of a task scheduler which switches between independent tasks), the MULIP flag must be saved as part of the task environment and must be updated accordingly for the new task before this task is entered.

CPU Interrupt Status (IEN, ILVL)

The Interrupt Enable bit allows interrupts to be globally enabled (IEN = '1') or disabled (IEN = '0'). 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**.

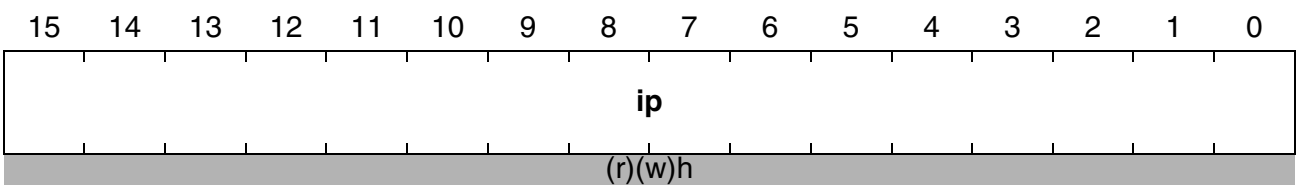
After reset, all interrupts are globally disabled, and the lowest priority (ILVL = 0) is assigned to the initial CPU activity.

Instruction Pointer IP

This register determines the 16-bit intra-segment address of the currently fetched instruction within the code segment selected by the CSP register. The IP register is not mapped into the C164CI'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.

The IP register is implicitly updated by the CPU for branch instructions and after instruction fetch operations.

IP
Instruction Pointer **- - - (- - - -/- -)** **Reset Value: 0000_H**



Bit	Function
ip	Specifies the intra-segment offset from which the current instruction is to be fetched. IP refers to the current segment <SEGNR>.

Code Segment Pointer CSP

This non-bit addressable register selects the code segment being used at run-time to access instructions. The lower 8 bits of register CSP select one of up to 256 segments of 64 KBytes each; the upper 8 bits are reserved for future use.

CSP
Code Segment Pointer **SFR ($FE08_H/04_H$)** **Reset Value: 0000_H**

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

Bit	Function
SEGNR	**Segment Number** Specifies the code segment from which the current instruction is to be fetched. SEGNR is ignored when segmentation is disabled.

Code memory addresses are generated by directly extending the 16-bit contents of the IP register by the contents of the CSP register, as shown in **Figure 4-5**.

In segmented memory mode, the selected number of segment address bits (via bitfield SALSEL) of register CSP is output on the respective segment address pins of Port 4 for all external code accesses. For non-segmented memory mode or Single Chip Mode, the content of this register is not significant because all code accesses are automatically restricted to segment 0.

Note: The CSP register can only be read but cannot be written by data operations. It is, however, modified either directly by means of the JMPS and CALLS instructions, or indirectly via the stack by means of the RETS and RETI instructions.
Upon the acceptance of an interrupt or the execution of a software TRAP instruction, the CSP register is set automatically to zero.

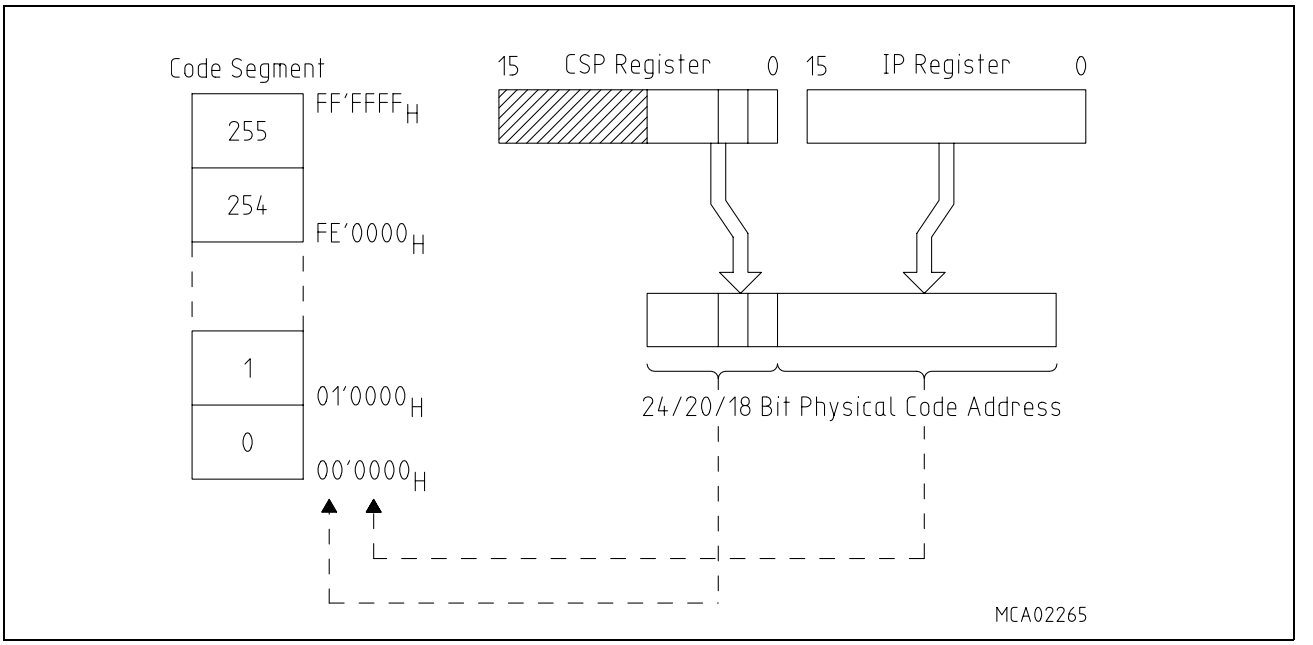


Figure 4-5 Addressing via the Code Segment Pointer

Note: When segmentation is disabled, the IP value is used directly as the 16-bit address.

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. The DPP registers allow access to the entire memory space in pages of 16 KBytes each.

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									

Bit	Function
DPPxPN	**Data Page Number of DPPx** Specifies the data page selected via DPPx. Only the least significant two bits of DPPx are significant, when segmentation is disabled.

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 4-6**. 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 (the selectable part is driven to the address pins).

In non-segmented memory mode, only the two least significant bits of the implicitly selected DPP register are used to generate the physical address. Thus, extreme care should be taken when changing the content of a DPP register if a non-segmented memory model is selected to avoid unexpected results.

In segmented memory mode, the selected number of segment address bits (via bitfield SALSEL) of the respective DPP register is output on the respective segment address pins of Port 4 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 new DPP value is not yet usable for the operand address calculation of the instruction immediately following the instruction which updates the DPP register.

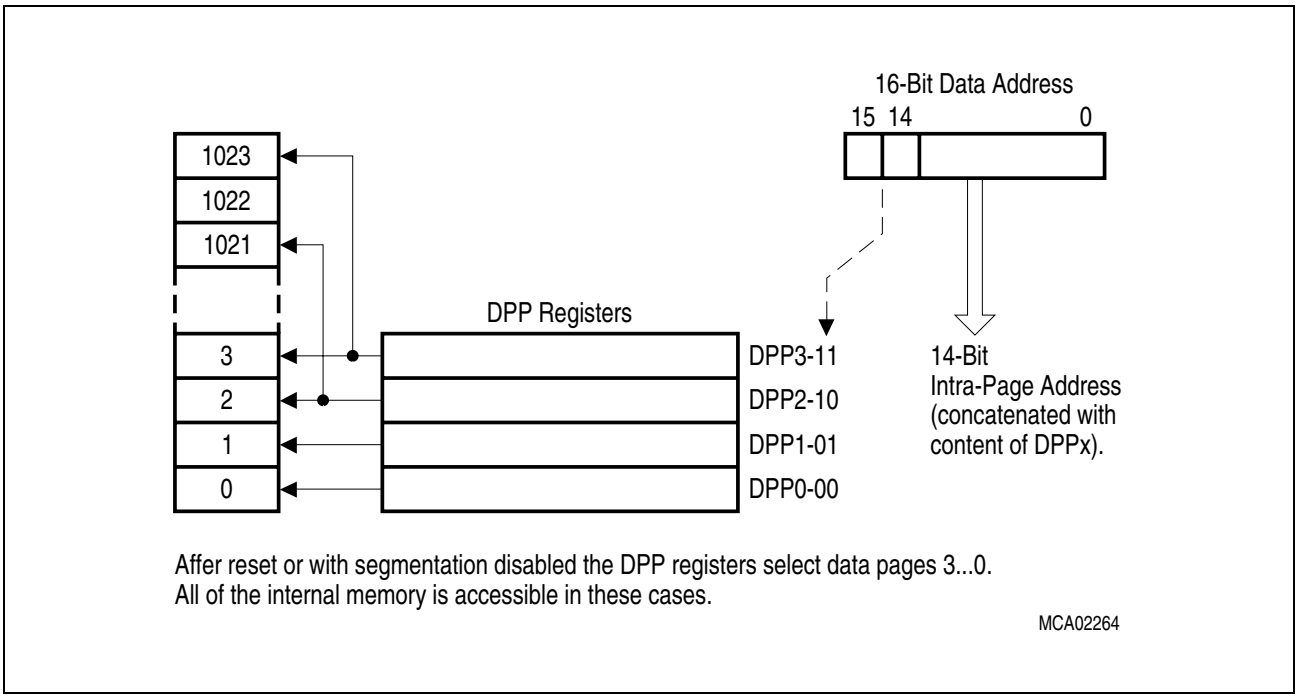


Figure 4-6 Addressing via the Data Page Pointers

Context Pointer CP

This non-bit addressable register is used to select the current register context. This means that the CP register value determines the address of the first General Purpose Register (GPR) within the current register bank of up to 16 wordwide and/or bytewide GPRs.

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

Bit	Function
cp	**Modifiable portion of register CP** Specifies the (word) base address of the current register bank. When writing a value to register CP with bits CP.11 … CP.9 = ‘000’, bits CP.11 … CP.10 are set to ‘11’ by hardware. In all other cases, all bits of the bit field “cp” receive the written value.

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 RAM location. If this condition is not met, unexpected results may occur.

- Do not set CP below the IRAM start address, i.e. $00'FA00_H/00'F600_H/00'F200_H$ (referring to an IRAM size of 1/2/3 KByte)
- Do not set CP above $00'FDFE_H$
- Be careful using the upper GPRs with CP above $00'FDE0_H$

The CP register can be updated via any instruction capable of modifying an SFR.

Note: Due to the internal instruction pipeline, a new CP value is not yet usable for GPR address calculations of the instruction immediately following the instruction which updated the CP register.

The Switch Context instruction (SCXT) allows saving the content of register CP on the stack and updating it with a new value in only one machine cycle.

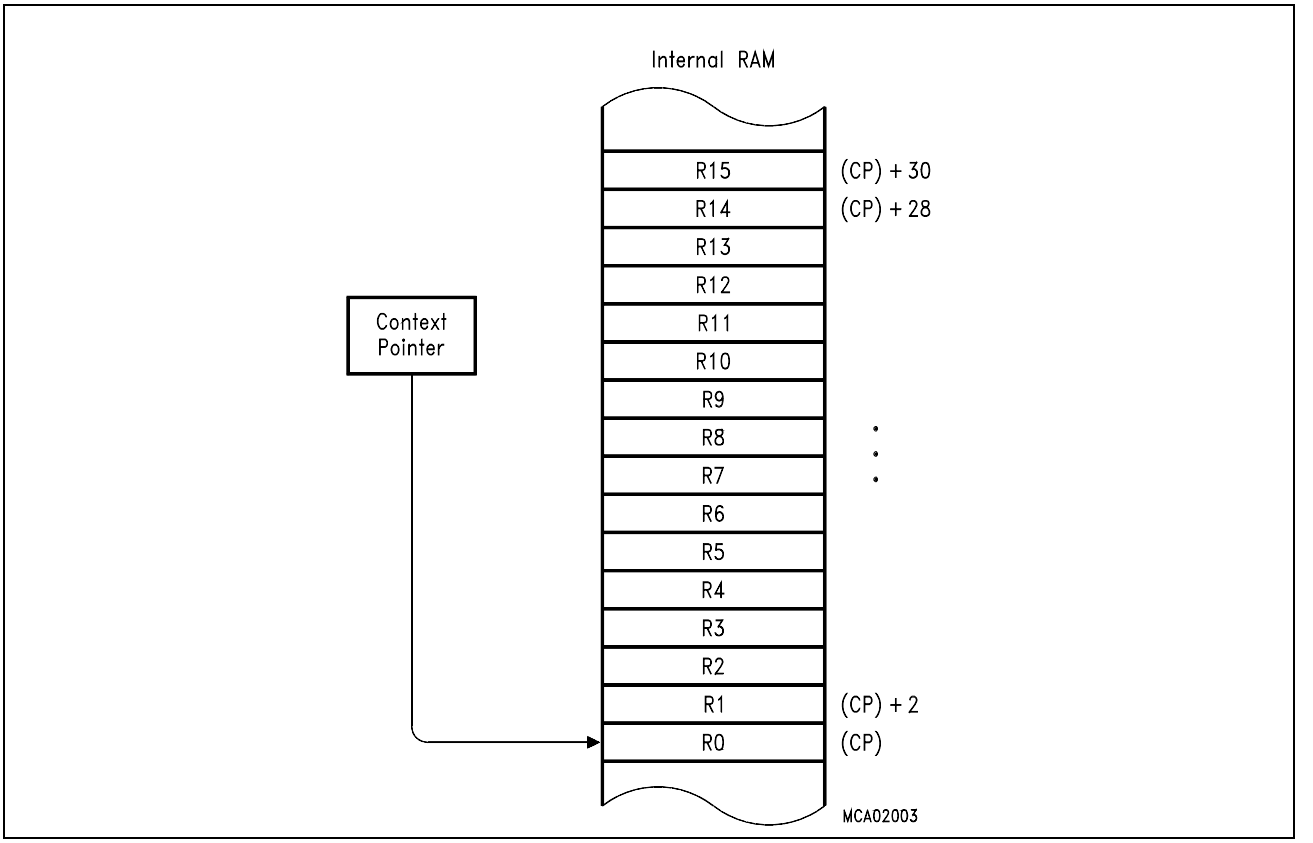


Figure 4-7 Register Bank Selection via Register CP

Several addressing modes use register CP implicitly for address calculations. The addressing modes identified below are described in **Chapter 24**.

Short 4-Bit GPR Addresses (mnemonic: Rw or Rb) specify an address relative to the memory location specified by the contents of the CP register, i.e. the base of the current register bank.
Depending on whether a relative word (Rw) or byte (Rb) GPR address is specified, the short 4-bit GPR address is either multiplied by two or not before it is added to the content of register CP (see **Figure 4-8**). Thus, both byte and word GPR accesses are possible.

GPRs used as indirect address pointers are always accessed wordwise. For some instructions, only the first four GPRs can be used as indirect address pointers. These GPRs are specified via short 2-bit GPR addresses. The respective physical address calculation is identical to that for the short 4-bit GPR addresses.

Short 8-Bit Register Addresses (mnemonic: reg or bitoff) within a range from $F0_H$ to FF_H interpret the four least significant bits as short 4-bit GPR address; the four most significant bits are ignored. The respective physical GPR address calculation is identical to that for the short 4-bit GPR addresses. For single bit accesses on a GPR, the GPR's word address is calculated as just described, but the position of the bit within the word is specified by a separate additional 4-bit value.

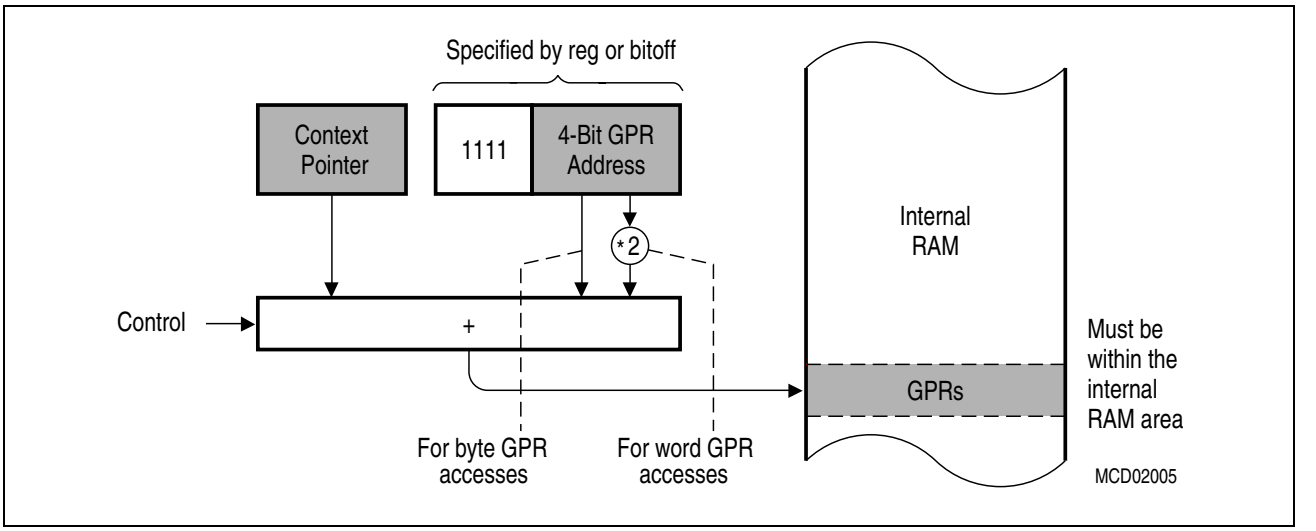


Figure 4-8 Implicit CP Use by Short GPR Addressing Modes

Stack Pointer SP

This non-bit addressable register is used to point to the top of the internal system stack (TOS). The SP register is pre-decremented whenever data is to be pushed onto the stack, and it is post-incremented whenever data is to be popped from the stack. Thus, the system stack grows from higher toward lower memory locations.

Because the least significant bit of register SP is tied to ‘0’ and bits 15 through 12 are tied to ‘1’ by hardware, the SP register can contain values from $F000_H$ to $FFFE_H$ only. This allows access to a physical stack within the internal RAM of the C164CI. A virtual stack (usually bigger) can be implemented via software. This mechanism is supported by registers STKOV and STKUN (see respective descriptions below).

The SP register can be updated via any instruction, which is capable of modifying an SFR.

Note: Due to the internal instruction pipeline, a POP or RETURN instruction must not immediately follow an instruction which updated the SP register.

SP
Stack Pointer Register **SFR ($FE12_H/09_H$)** **Reset Value: $FC00_H$**

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

Bit	Function
sp	**Modifiable portion of register SP** Specifies the top of the internal system stack.

Stack Overflow Pointer STKOV

This non-bit addressable register is compared against the SP register after each operation which pushes data onto the system stack (e.g. PUSH and CALL instructions or interrupts) and after each subtraction from the SP register. If the content of the SP register is less than the content of the STKOV register, a stack overflow hardware trap will occur.

Because the least significant bit of register STKOV is tied to ‘0’ and bits 15 through 12 are tied to ‘1’ by hardware, the STKOV register can contain values from $F000_H$ to $FFFE_H$ only.

STKOV
Stack Overflow Reg. **SFR ($FE14_H/0A_H$)** **Reset Value: $FA00_H$**

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

Bit	Function
stkov	**Modifiable portion of register STKOV** Specifies the lower limit of the internal system stack.

The Stack Overflow Trap (entered when (SP) < (STKOV)) may be used in two different ways:

- **Fatal error indication** treats the stack overflow as a system error through the associated trap service routine. Under these circumstances data in the bottom of the stack may have been overwritten by the status information stacked upon servicing the stack overflow trap.
- **Automatic system stack flushing** allows to use the system stack as a ‘Stack Cache’ for a bigger external user stack. In this case register STKOV should be initialized to a value, which represents the desired lowest Top of Stack address plus 12 according to the selected maximum stack size. This considers the worst case that will occur, when a stack overflow condition is detected just during entry into an interrupt service routine. Then, six additional stack word locations are required to push IP, PSW, and CSP for both the interrupt service routine and the hardware trap service routine.

More details about the stack overflow trap service routine and virtual stack management are given in **Chapter 24**.

Stack Underflow Pointer STKUN

This non-bit addressable register is compared against the SP register after each operation which pops data from the system stack (e.g. POP and RET instructions) and after each addition to the SP register. If the content of the SP register is greater than the content of the STKUN register, a stack underflow hardware trap will occur.

Because the least significant bit of register STKUN is tied to ‘0’ and bits 15 through 12 are tied to ‘1’ by hardware, the STKUN register can contain values from $F000_H$ to $FFFE_H$ only.

STKUN
Stack Underflow Reg. **SFR ($FE16_H/0B_H$)** **Reset Value: $FC00_H$**

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

Bit	Function
stkun	**Modifiable portion of register STKUN** Specifies the upper limit of the internal system stack.

The Stack Underflow Trap (entered when (SP) > (STKUN)) may be used in two different ways:

- **Fatal error indication** treats the stack underflow as a system error through the associated trap service routine.
- **Automatic system stack refilling** allows to use the system stack as a ‘Stack Cache’ for a bigger external user stack. In this case register STKUN should be initialized to a value, which represents the desired highest Bottom of Stack address.

More details about the stack underflow trap service routine and virtual stack management are given in **Chapter 24**.

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 is moved outside the defined stack area, either by ADD or SUB instructions or by PUSH or POP operations (explicit or implicit, i.e. CALL or RET instructions).

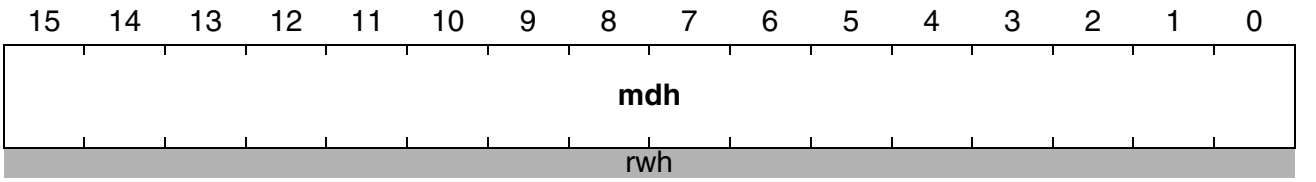
This control mechanism is not triggered, i.e. no stack trap is generated, when

- The Stack Pointer SP is directly updated via MOV instructions
- The limits of the stack area (STKOV, STKUN) are changed so that SP is outside the new limits

Multiply/Divide High Register MDH

This register is part of the 32-bit multiply/divide register which is implicitly used by the CPU when it performs a multiplication or a division. After a multiplication, this non-bit addressable register represents the high order 16 bits of the 32-bit result. For long divisions, the MDH register must be loaded with the high order 16 bits of the 32-bit dividend before the division is started. After any division, register MDH represents the 16-bit remainder.

MDH
Multiply/Divide High Reg. **SFR ($FE0C_H/06_H$)** **Reset Value: 0000_H**



Bit	Function
mdh	Specifies the high order 16 bits of the 32-bit multiply and divide reg. MD.

Whenever this register is updated via software, the Multiply/Divide Register In Use (MDRIU) flag in the Multiply/Divide Control register (MDC) is set to ‘1’.

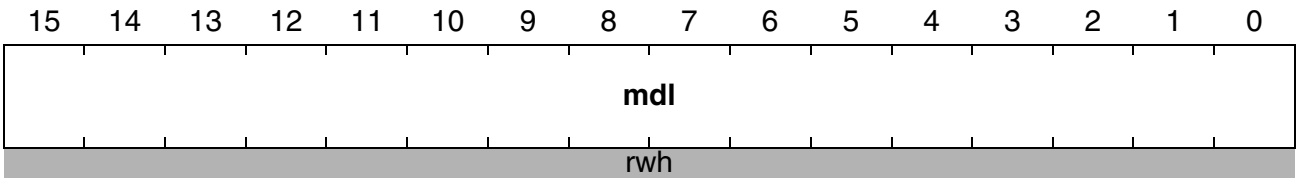
When a multiplication or division is interrupted before its completion and when a new multiply or divide operation is to be performed within the interrupt service routine, register MDH must be saved along with registers MDL and MDC to avoid erroneous results.

A detailed description of how to use the MDH register for programming multiply and divide algorithms can be found in **Chapter 22**.

Multiply/Divide Low Register MDL

This register is part of the 32-bit multiply/divide register, which is implicitly used by the CPU when it performs a multiplication or a division. After a multiplication, this non-bit addressable register represents the low order 16 bits of the 32-bit result. For long divisions, the MDL register must be loaded with the low order 16 bits of the 32-bit dividend before the division is started. After any division, register MDL represents the 16-bit quotient.

MDL
Multiply/Divide Low Reg. **SFR ($FE0E_H/07_H$)** **Reset Value: 0000_H**



Bit	Function
mdl	Specifies the low order 16 bits of the 32-bit multiply and divide reg. MD.

Whenever this register is updated via software, the Multiply/Divide Register In Use (MDRIU) flag in the Multiply/Divide Control register (MDC) is set to ‘1’. The MDRIU flag is cleared, whenever the MDL register is read via software.

When a multiplication or division is interrupted before its completion and when a new multiply or divide operation is to be performed within the interrupt service routine, register MDL must be saved along with registers MDH and MDC to avoid erroneous results.

A detailed description of how to use the MDL register for programming multiply and divide algorithms can be found in **Chapter 22**.

Multiply/Divide Control Register MDC

This bit addressable 16-bit register is implicitly used by the CPU when it performs a multiplication or a division. It is used to store the required control information for the corresponding multiply or divide operation. Register MDC is updated by hardware during each single cycle of a multiply or divide instruction.

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	r(w)h	r(w)h	r(w)h	r(w)h	r(w)h	r(w)h	r(w)h

Bit	Function
MDRIU	**Multiply/Divide Register In Use** 0: Cleared when register MDL is read via software. 1: Set when register MDL or MDH is written via software or when a multiply or divide instruction is executed.
!!	**Internal Machine Status** The multiply/divide unit uses these bits to control internal operations. Never modify these bits without saving and restoring register MDC.

When a division or multiplication was interrupted before its completion and the multiply/divide unit is required, the MDC register must first be saved along with registers MDH and MDL (to be able to restart the interrupted operation later). Then it must be cleared to prepare it for the new calculation. After completion of the new division or multiplication, the state of the interrupted multiply or divide operation must be restored.

The MDRIU flag is the only portion of the MDC register which might be of interest to the user. The remaining portions of the MDC register are reserved for dedicated use by the hardware and should never be modified by the user in any way other than described above. Otherwise, a correct continuation of an interrupted multiply or divide operation cannot be guaranteed.

A detailed description of how to use the MDC register for programming multiply and divide algorithms can be found in **Chapter 22**.

Constant Zeros Register ZEROS

All bits of this bit-addressable register are fixed to ‘0’ by hardware. This register can be read only. Register ZEROS can be used as a register-addressable constant of all zeros, i.e. for bit manipulation or mask generation. It 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

Constant Ones Register ONES

All bits of this bit-addressable register are fixed to ‘1’ by hardware. This register can be read only. Register ONES can be used as a register-addressable constant of all ones, for bit manipulation or mask generation. It can be accessed via any instruction capable of addressing an SFR.

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 C164CI supports several mechanisms for fast and flexible response to service requests from various sources internal or external to the microcontroller. These mechanisms include: Normal Interrupt Processing, Interrupt Processing via the Peripheral Event Controller, Trap Functions, and External Interrupt Processing.

Normal Interrupt Processing

The CPU temporarily suspends current program execution and branches to an interrupt service routine to service an interrupt requesting device. 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 C164CI'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 in segment 0 (data pages 0 through 3) through one of eight programmable PEC Service Channels. During a PEC transfer, normal program execution of the CPU is halted for only one instruction cycle. No internal program status information needs to be saved. The same prioritization scheme is used for PEC service as for normal interrupt processing. PEC transfers share the two highest priority levels.

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 trap types, current program status is saved on the system stack.

External Interrupt Processing

Although the C164CI 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 C164CI provides 32 separate interrupt nodes assignable to 16 priority levels. 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 the interrupt subnode control register ISNC (see **Section 5.7**).

The C164CI 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 only exceptions are the Class B hardware traps, all of which 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 the C164CI's address space (segment 0). 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 code segment 0 of the address space. Each entry occupies two words, except for the reset vector and the hardware trap vectors, which occupy four or eight words. **Table 5-1** lists all sources capable of requesting interrupt or PEC service in the C164CI, the associated interrupt vectors, their locations, and the associated trap numbers. It also lists the mnemonics of the affected Interrupt Request flags and their corresponding Interrupt Enable flags. The mnemonics consist of a part which specifies the respective source, followed by a part which specifies its function (IR = Interrupt Request flag, IE = Interrupt Enable flag).

Note: Each entry of the interrupt vector table provides room for two word instructions or one doubleword instruction. The respective vector location results from multiplying the trap number by 4 (4 bytes per entry).
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.

Table 5-1 C164CI Interrupt Nodes and Vectors

Source of Interrupt or PEC Service Request	Request Flag	Enable Flag	Interrupt Vector	Vector Location	Trap Number
Fast External Interrupt 0	CC8IR	CC8IE	CC8INT	$00'0060_H$	$18_H / 24_D$
Fast External Interrupt 1	CC9IR	CC9IE	CC9INT	$00'0064_H$	$19_H / 25_D$
Fast External Interrupt 2	CC10IR	CC10IE	CC10INT	$00'0068_H$	$1A_H / 26_D$
Fast External Interrupt 3	CC11IR	CC11IE	CC11INT	$00'006C_H$	$1B_H / 27_D$
GPT1 Timer 2	T2IR	T2IE	T2INT	$00'0088_H$	$22_H / 34_D$
GPT1 Timer 3	T3IR	T3IE	T3INT	$00'008C_H$	$23_H / 35_D$
GPT1 Timer 4	T4IR	T4IE	T4INT	$00'0090_H$	$24_H / 36_D$
A/D Conversion Complete	ADCIR	ADCIE	ADCINT	$00'00A0_H$	$28_H / 40_D$
A/D Overrun Error	ADEIR	ADEIE	ADEINT	$00'00A4_H$	$29_H / 41_D$
ASC0 Transmit	S0TIR	S0TIE	S0TINT	$00'00A8_H$	$2A_H / 42_D$
ASC0 Receive	S0RIR	S0RIE	S0RINT	$00'00AC_H$	$2B_H / 43_D$
ASC0 Error	S0EIR	S0EIE	S0EINT	$00'00B0_H$	$2C_H / 44_D$
SSC Transmit	SCTIR	SCTIE	SCTINT	$00'00B4_H$	$2D_H / 45_D$
SSC Receive	SCRIR	SCRIE	SCRINT	$00'00B8_H$	$2E_H / 46_D$
SSC Error	SCEIR	SCEIE	SCEINT	$00'00BC_H$	$2F_H / 47_D$
CAPCOM Register 16	CC16IR	CC16IE	CC16INT	$00'00C0_H$	$30_H / 48_D$
CAPCOM Register 17	CC17IR	CC17IE	CC17INT	$00'00C4_H$	$31_H / 49_D$
CAPCOM Register 18	CC18IR	CC18IE	CC18INT	$00'00C8_H$	$32_H / 50_D$
CAPCOM Register 19	CC19IR	CC19IE	CC19INT	$00'00CC_H$	$33_H / 51_D$
CAPCOM Register 24	CC24IR	CC24IE	CC24INT	$00'00E0_H$	$38_H / 56_D$
CAPCOM Register 25	CC25IR	CC25IE	CC25INT	$00'00E4_H$	$39_H / 57_D$
CAPCOM Register 26	CC26IR	CC26IE	CC26INT	$00'00E8_H$	$3A_H / 58_D$
CAPCOM Register 27	CC27IR	CC27IE	CC27INT	$00'00EC_H$	$3B_H / 59_D$
CAPCOM Timer 7	T7IR	T7IE	T7INT	$00'00F4_H$	$3D_H / 61_D$
CAPCOM Timer 8	T8IR	T8IE	T8INT	$00'00F8_H$	$3E_H / 62_D$
CAPCOM6 Interrupt	CC6IR	CC6IE	CC6INT	$00'00FC_H$	$3F_H / 63_D$
CAN	XP0IR	XP0IE	XP0INT	$00'0100_H$	$40_H / 64_D$
PLL/OWD, RTC (via ISNC)	XP3IR	XP3IE	XP3INT	$00'010C_H$	$43_H / 67_D$
ASC0 Transmit Buffer	S0TBIR	S0TBIE	S0TBINT	$00'011C_H$	$47_H / 71_D$
CAPCOM6 Timer 12	T12IR	T12IE	T12INT	$00'0134_H$	$4D_H / 77_D$
CAPCOM6 Timer 13	T13IR	T13IE	T13INT	$00'0138_H$	$4E_H / 78_D$
CAPCOM6 Emergency	CC6IR	CC6IE	CC6EINT	$00'013C_H$	$4F_H / 79_D$

Table 5-2 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 $00'0000_H$. 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 vector location between $00'0000_H$ and $00'01FC_H$. A service routine entered via a software TRAP instruction is always executed on the current CPU priority level which is indicated in bit field 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 5-2 Hardware Trap Summary

Exception Condition	Trap Flag	Trap Vector	Vector Location	Trap Number	Trap Prio
Reset Functions	–				
Hardware Reset		RESET	$00'0000_H$	00_H	III
Software Reset		RESET	$00'0000_H$	00_H	III
Watchdog Timer Overflow		RESET	$00'0000_H$	00_H	III
Class A Hardware Traps					
Non-Maskable Interrupt	NMI	NMITRAP	$00'0008_H$	02_H	II
Stack Overflow	STKOF	STOTRAP	$00'0010_H$	04_H	II
Stack Underflow	STKUF	STUTRAP	$00'0018_H$	06_H	II
Class B Hardware Traps					
Undefined Opcode	UNDOPC	BTRAP	$00'0028_H$	$0A_H$	I
Protected Instruction Fault	PRTFLT	BTRAP	$00'0028_H$	$0A_H$	I
Illegal Word Operand Access	ILLOPA	BTRAP	$00'0028_H$	$0A_H$	I
Illegal Instruction Access	ILLINA	BTRAP	$00'0028_H$	$0A_H$	I
Illegal External Bus Access	ILLBUS	BTRAP	$00'0028_H$	$0A_H$	I
Reserved	–	–	$[2C_H - 3C_H]$	$[0B_H - 0F_H]$	–
Software Traps TRAP Instruction	–	–	Any $[00'0000_H - 00'01FC_H]$ in steps of 4_H	Any $[00_H - 7F_H]$	Current CPU Priority

Normal Interrupt Processing and PEC Service

During each instruction cycle, one out of all sources requiring PEC or interrupt processing is selected according to its interrupt priority. This priority of interrupts and PEC requests is programmable in two levels. Each requesting source can be assigned to a specific priority. A second level (called “group priority”) allows to specify an internal order for simultaneous requests from a group of different sources on the same priority level. At the end of each instruction cycle, the one source request with the highest current priority will be determined by the interrupt system. This request will then be serviced if its priority is higher than the current CPU priority in register PSW.

Interrupt System Register Description

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.

5.1.1 Interrupt Control Registers

All interrupt control registers are organized identically. The lower 8 bits of an interrupt control register contain complete interrupt status information for the associated source, which is required for one round of prioritization; the upper 8 bits of the respective register are reserved. All interrupt control registers are bit-addressable and all bits can be read or written via software. This allows each interrupt source to be programmed or modified with just one instruction. When accessing interrupt control registers through instructions which operate on word data types, their upper 8 bits (15 … 8) will return zeros when read, and will discard written data.

The layout of the Interrupt Control registers shown below applies to each xxIC register, where xx represents the mnemonic for the respective source.

xxIC
Interrupt Control Register **(E)SFR ($yyyy_H/zz_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								xxIR	**xxIE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

Bit	Function
GLVL	**Group Level** Defines the internal order for simultaneous requests of the same priority. 3: Highest group priority 0: Lowest group priority
ILVL	**Interrupt Priority Level** Defines the priority level for the arbitration of requests. F_H: Highest priority level 0_H: Lowest priority level
xxIE	**Interrupt Enable Control Bit** (individually enables/disables a specific source) 0: Interrupt request is disabled 1: Interrupt Request is enabled
xxIR	**Interrupt Request Flag** 0: No request pending 1: This source has raised an interrupt request

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. This allows a normal CPU interrupt to respond to a completed PEC block transfer.

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 cycles (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 bit field 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 bit fields GLVL are used for second level arbitration to select one request to be serviced. Again, the group priority increases with the numerical value of GLVL, so 00_B is the lowest and 11_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 bit field ILVL of register PSW after pushing the old PSW contents onto the stack.

The interrupt system of the C164CI 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 or 14 (i.e., ILVL = $111X_B$) will be serviced by the PEC unless the COUNT field of the associated PECC register contains zero. In this case, the request will be serviced by normal interrupt processing instead. Interrupt requests programmed to priority levels 13 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 enabled interrupt request on level 0000_B will terminate the C164CI's Idle mode and reactivate the CPU.

For interrupt requests which are to be serviced by the PEC, the associated PEC channel number is derived from the respective ILVL (LSB) and GLVL (see **Figure 5-1**). So, programming a source to priority level 15 (ILVL = 1111_B) selects the PEC channel group 7 … 4; programming a source to priority level 14 (ILVL = 1110_B) selects the PEC channel group 3 … 0. The actual PEC channel number is then determined by the group priority field GLVL.

Simultaneous requests for PEC channels are prioritized according to the PEC channel number, where channel 0 has lowest and channel 8 has highest priority.

Note: All sources requesting PEC service must be programmed to different PEC channels. Otherwise, an incorrect PEC channel may be activated.

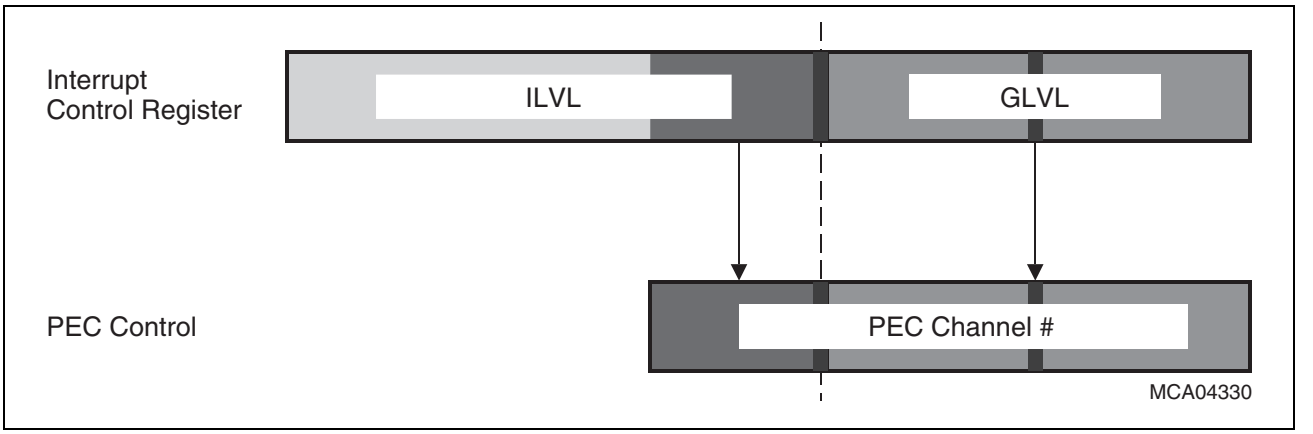


Figure 5-1 Priority Levels and PEC Channels

The **Table 5-3** shows in a few examples which action is executed with a given programming of an interrupt control register.

Table 5-3 Interrupt Priority Examples

Priority Level		Type of Service	
ILVL	**GLVL**	**COUNT = 00_H**	**COUNT ≠ 00_H**
1 1 1 **1**	**1 1**	CPU interrupt, level 15, group priority 3	PEC service, channel 7
1 1 1 **1**	**1 0**	CPU interrupt, level 15, group priority 2	PEC service, channel 6
1 1 1 **0**	**1 0**	CPU interrupt, level 14, group priority 2	PEC service, channel 2
1 1 0 1	1 0	CPU interrupt, level 13, group priority 2	CPU interrupt, level 13, group priority 2
0 0 0 1	1 1	CPU interrupt, level 1, group priority 3	CPU interrupt, level 1, group priority 3
0 0 0 1	0 0	CPU interrupt, level 1, group priority 0	CPU interrupt, level 1, group priority 0
0 0 0 0	X X	No service!	No service!

Note: Requests on levels 13 … 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.

Interrupt Control Functions in the PSW

The Processor Status Word (PSW) is functionally divided into two parts: the lower byte of the PSW basically represents the arithmetic status of the CPU; the upper byte of the PSW controls the interrupt system of the C164CI and the arbitration mechanism for the external bus interface.

*Note: Pipeline effects must be considered when enabling/disabling interrupt requests via modifications of register PSW (see **Chapter 4**).*

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	-	-	-	-	USR 0	MUL IP	E	Z	V	C	N
rwh				rw	-	-	-	-	rw	rwh	rwh	rwh	rwh	rwh	rwh

Bit	Function
N, C, V, Z, E, MULIP, USR0	**CPU status flags** (Described in **Section 4**) Define the current status of the CPU (ALU, multiplication unit).
IEN	**Interrupt Enable Control Bit** (globally enables/disables interrupt requests) 0: Interrupt requests are disabled 1: Interrupt requests are enabled
ILVL	**CPU Priority Level** Defines the current priority level for the CPU F_H: Highest priority level 0_H: Lowest priority level

CPU Priority ILVL defines the current level for the operation of the CPU. This bit field reflects the priority level of the routine currently executed. Upon entry into an interrupt service routine, this bit field 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. However, requests already in the pipeline at that time will be processed. 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.

Note: Traps are non-maskable and are, therefore, not affected by the IEN bit.

5.2 Operation of the PEC Channels

The C164CI's Peripheral Event Controller (PEC) provides 8 PEC service channels which move a single byte or word between two locations in segment 0 (data pages 3 … 0). This is the fastest possible interrupt response, and, in many cases is sufficient to service the respective peripheral request (for example, serial channels, etc.). Each channel is controlled by a dedicated PEC Channel Counter/Control register (PECCx) and a pair of pointers for source (SRCPx) and destination (DSTPx) of the data transfer.

The PECC registers control the action performed by the respective PEC channel.

PECCx
PEC Control Reg. **SFR ($FECy_H/6z_H$, see Table 5-4)** **Reset Value: 0000_H**

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

Bit	Function
COUNT	**PEC Transfer Count** Counts PEC transfers and influences the channel's action (see Table 5-5)
BWT	**Byte / Word Transfer Selection** 0: Transfer a Word 1: Transfer a Byte
INC	**Increment Control** (Modification of SRCPx or DSTPx) 0 0: Pointers are not modified 0 1: Increment DSTPx by 1 or 2 (BWT) 1 0: Increment SRCPx by 1 or 2 (BWT) 1 1: Reserved. Do not use this combination. (changed to '10' by hardware)

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

Byte/Word Transfer 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 modified pointer.

Increment Control field INC controls, whether one of the PEC pointers is incremented after the PEC transfer. It is not possible to increment both pointers, however. If the pointers are not modified (INC = '00'), the respective channel will always move data from the same source to the same destination.

Note: The reserved combination '11' is changed to '10' by hardware. However, it is not recommended to use this combination.

The PEC Transfer Count Field COUNT controls the action of a respective PEC channel. The content of bit field COUNT selects 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 5-5 summarizes, how the COUNT field, the interrupt requests flag IR, and the PEC channel action depend on the previous content of COUNT.

Table 5-5 Influence of Bitfield COUNT

Previous COUNT	Modified COUNT	IR after PEC 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'	Move a Byte/Word Leave request flag set, which triggers another request
00_H	00_H	('1')	**No action!** Activate interrupt service routine rather than PEC channel

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 the interrupt service routine associated with the priority level. After each PEC transfer, the COUNT field is decremented and the request flag is cleared to indicate that the request has been serviced.

Continuous transfers are selected by the value FF_H in bit field 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, the request flag is not cleared. This generates another request from the same source. When COUNT already contains the value 00_H, the respective PEC channel remains idle and the associated interrupt service routine is activated instead. This allows choosing whether a level 15 or 14 request should be serviced by the PEC or by the interrupt service routine.

Note: PEC transfers are executed only if their priority level is higher than the CPU level, that is, only PEC channels 7 … 4 are processed, while the CPU executes on level 14.
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.

The source and destination pointers specify the locations between which the data is to be moved. A pair of pointers (SRCPx and DSTPx) is associated with each of the eight PEC channels. These pointers do not reside in specific SFRs, but are mapped into the internal RAM of the C164CI just below the bit-addressable area (see **Figure 5-2**).

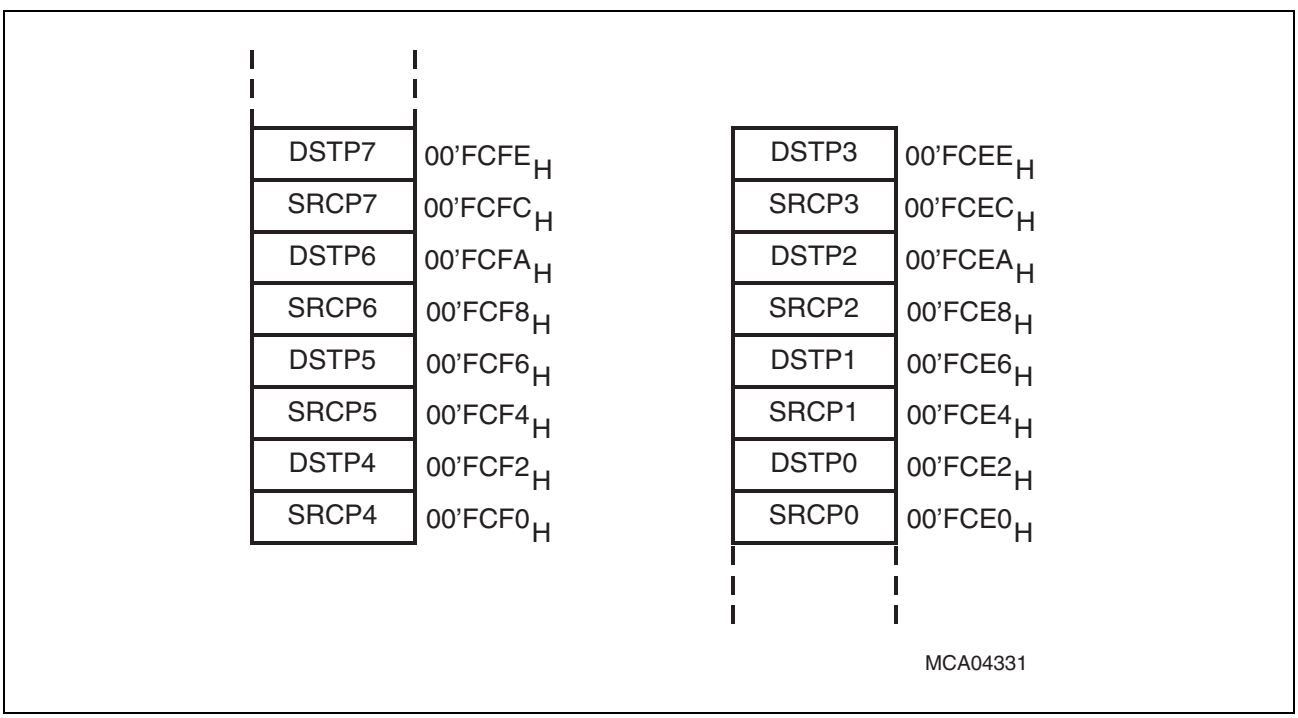


Figure 5-2 Mapping of PEC Pointers into the Internal RAM

PEC data transfers do not use the data page pointers DPP3 … DPP0. The PEC source and destination pointers are used as 16-bit intra-segment addresses within segment 0, so data can be transferred between any two locations within the first four data pages 3 … 0.

The pointer locations for inactive PEC channels may be used for general data storage. Only the required pointers occupy RAM locations.

Note: If word data transfer is selected for a specific PEC channel (i.e. 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.

5.3 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 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 (see **Chapter 22**).

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 C164CI supports this function with two features:

Classes with up to four members can be established by using the same interrupt priority (ILVL) and assigning a dedicated group level (GLVL) to each member. This functionality is built-in and handled automatically by the interrupt controller.

Classes with more than four members can be established by using a number of adjacent interrupt priorities (ILVL) and the respective group levels (four 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 24 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 5-6 Software Controlled Interrupt Classes (Example)

ILVL (Priority)	GLVL				Interpretation
	3	2	1	0	
15					PEC service on up to 8 channels
14					
13					
12	X	X	X	X	Interrupt Class 1 5 sources on 2 levels
11	X				
10					
9					
8	X	X	X	X	Interrupt Class 2 9 sources on 3 levels
7	X	X	X	X	
6	X				
5	X	X	X	X	Interrupt Class 3 5 sources on 2 levels
4	X				
3					
2					
1					
0					No service!

5.4 Saving Status during Interrupt Service

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 SYSCON 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. If a multiplication or division was in progress at the time the interrupt request was acknowledged, bit MULIP in register PSW is set to ‘1’. In this case, the return location saved on the stack is not the next instruction in the instruction flow, but rather the multiply or divide instruction itself, as this instruction has been interrupted and will be completed after returning from the service routine.

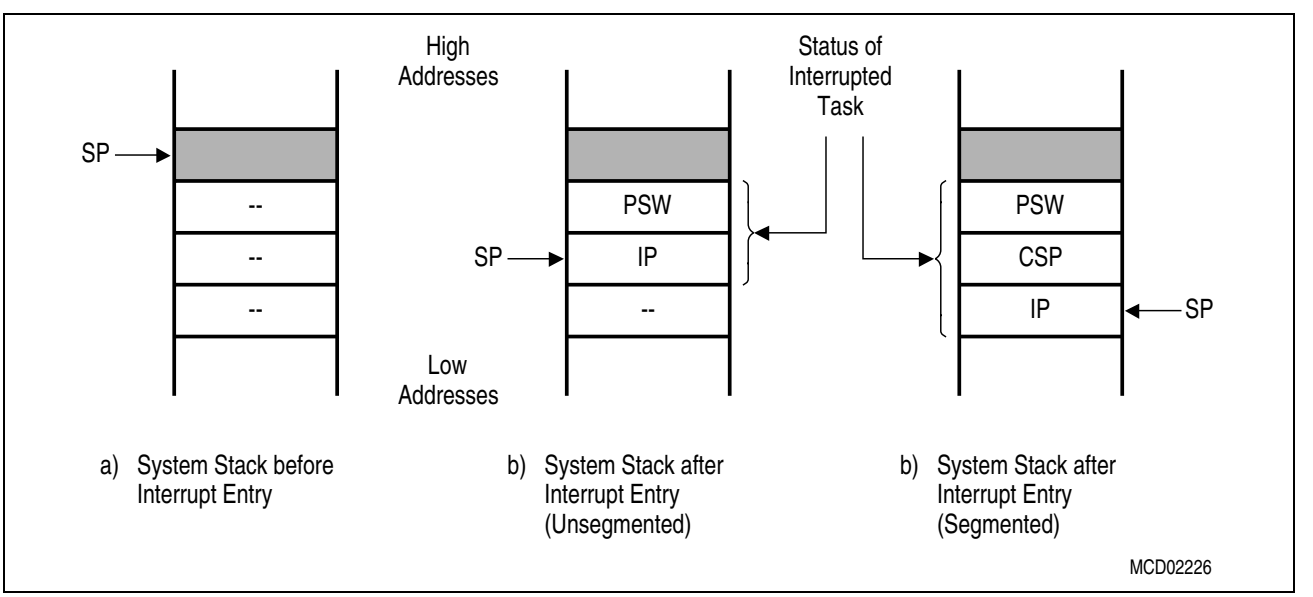


Figure 5-3 Task Status Saved on the System Stack

The interrupt request flag of the source being serviced is cleared. The IP is loaded with the vector associated with the requesting source (CSP is cleared in the case of segmentation), and the first instruction of the service routine is fetched from the vector location which is expected to branch to the service routine itself. The 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.

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. To save time, the C164CI allows switching the complete bank of CPU registers (GPRs) with a single instruction, so the service routine executes within its own separate context.

The instruction “SCXT CP, #New_Bank” pushes the content 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 bank.

Note: The first instruction following the SCXT instruction must not use a GPR.

Resources used by the interrupting program must eventually be saved and restored, e.g. the DPPs and the registers of the MUL/DIV unit.

5.5 Interrupt Response Times

The interrupt response time defines the time from the setting of an interrupt request flag of an enabled interrupt source to fetching of the first instruction (I1) from the interrupt vector location. The basic interrupt response time for the C164CI is 3 instruction cycles.

Pipeline Stage	Cycle 1	Cycle 2	Cycle 3	Cycle 4
FETCH	N	N + 1	N + 2	I1
DECODE	N - 1	N	TRAP (1)	TRAP (2)
EXECUTE	N - 2	N - 1	N	TRAP
WRITEBACK	N - 3	N - 2	N - 1	N

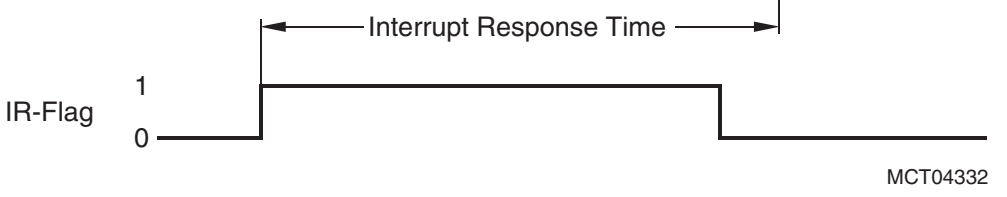


Figure 5-4 Pipeline Diagram for Interrupt Response Time

All instructions in the pipeline including instruction N (during which the interrupt request flag is set) are completed before entering the service routine. Thus, the actual execution time for these instructions (e.g. waitstates) influences the interrupt response time.

As shown in **Figure 5-4** the respective interrupt request flag is set in cycle 1 (fetching of instruction N). The indicated source wins the prioritization round (during cycle 2). In cycle 3 a TRAP instruction is injected into the decode stage of the pipeline, replacing instruction N+1 and clearing the source's interrupt request flag to '0'. Cycle 4 completes the injected TRAP instruction (save PSW, IP and CSP, if in segmented mode) and fetches the first instruction (I1) from the respective vector location.

All instructions that entered the pipeline after setting of the interrupt request flag (N+1, N+2) will be executed after returning from the interrupt service routine.

The minimum interrupt response time is 5 states (10 TCL). This requires program execution from the internal code memory, no external operand read requests, and setting the interrupt request flag during the last state of an instruction cycle. When the interrupt request flag is set during the first state of an instruction cycle, the minimum interrupt response time under these conditions is 6 state times (12 TCL).

The interrupt response time is increased by all delays of the instructions in the pipeline that are executed before entering the service routine (including N).

- When internal hold conditions between instruction pairs N-2/N-1 or N-1/N occur, or instruction N explicitly writes to the PSW or the SP, the minimum interrupt response time may be extended by 1 state time for each of these conditions.
- When instruction N reads an operand from the internal code memory, or when N is a call, return, trap, or MOV Rn, [Rm+ #data16] instruction, the minimum interrupt response time may additionally be extended by 2 state times during internal code memory program execution.
- In case instruction N reads the PSW and instruction N-1 has an effect on the condition flags, the interrupt response time may additionally be extended by 2 state times.

The worst case interrupt response time during internal code memory program execution adds to 12 state times (24 TCL).

Any reference to external locations increases the interrupt response time due to pipeline related access priorities. The following conditions must be considered:

- Instruction fetch from an external location
- Operand read from an external location
- Result write-back to an external location

Depending on where the instructions, source and destination operands are located, there are a number of combinations. Note, however, that only access conflicts contribute to the delay.

A few examples illustrate these delays:

- The worst case interrupt response time including external accesses will occur when instructions N, N+1 and N+2 are executed out of external memory, instructions N-1 and N require external operand read accesses, instructions N-3 through N write back external operands, and the interrupt vector also points to an external location. In this case, the interrupt response time is the time to perform 9 word bus accesses, because instruction I1 cannot be fetched via the external bus until all write, fetch, and read requests of preceding instructions in the pipeline are terminated.
- When the above example has the interrupt vector pointing into the internal code memory, the interrupt response time is 7 word bus accesses plus 2 states, because fetching of instruction I1 from internal code memory can start earlier.
- When instructions N, N+1 and N+2 are executed from external memory and the interrupt vector also points to an external location, but all operands for instructions N-3 through N are in internal memory, the interrupt response time is the time needed to perform 3 word bus accesses.
- When the above example has the interrupt vector pointing into the internal code memory, the interrupt response time is 1 word bus access plus 4 states.

After an interrupt service routine has been terminated by executing the RETI instruction, and if further interrupts are pending, the next interrupt service routine will not be entered until at least two instruction cycles have been executed of the interrupted program. In most cases, two instructions will be executed during this time. Typically, only one instruction will be executed if the first instruction following the RETI instruction is a branch instruction (without cache hit), or if it reads an operand from internal code memory, or if it is executed from the internal RAM.

Note: A bus access in this context includes all delays which can occur during an external bus cycle.

5.6 PEC Response Times

The PEC response time defines the time from the setting of an interrupt request flag of an enabled interrupt source to the start of the PEC data transfer. The basic PEC response time for the C164CI is 2 instruction cycles.

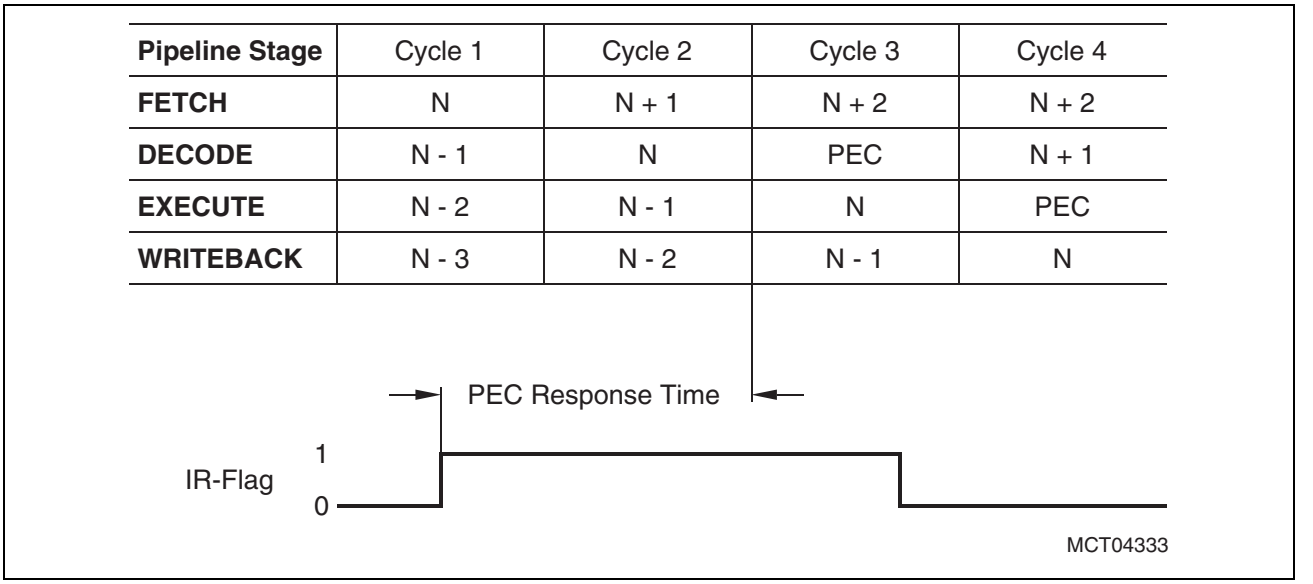


Figure 5-5 Pipeline Diagram for PEC Response Time

In **Figure 5-5** above the respective interrupt request flag is set in cycle 1 (fetching of instruction N). The indicated source wins the prioritization round during cycle 2. In cycle 3, a PEC transfer “instruction” is injected into the decode stage of the pipeline, suspending instruction N+1 and clearing the source’s interrupt request flag to ‘0’. Cycle 4 completes the injected PEC transfer and resumes the execution of instruction N+1.

All instructions that entered the pipeline after setting of the interrupt request flag (N+1, N+2) will be executed after the PEC data transfer.

Note: If instruction N reads any of the PEC control registers PECC7 … PECC0, when a PEC request wins the current round of prioritization, this round is repeated and the PEC data transfer is started one cycle later.

The minimum PEC response time is 3 states (6 TCL). This requires program execution from the internal code memory, no external operand read requests, and setting the interrupt request flag during the last state of an instruction cycle. When the interrupt request flag is set during the first state of an instruction cycle, the minimum PEC response time under these conditions is 4 state times (8 TCL).

The PEC response time is increased by all delays of the instructions in the pipeline that are executed before starting the data transfer (including N).

- When internal hold conditions between instruction pairs N-2/N-1 or N-1/N occur, the minimum PEC response time may be extended by 1 state time for each of these conditions.
- When instruction N reads an operand from the internal code memory, or when N is a call, return, trap, or MOV Rn, [Rm+ #data16] instruction, the minimum PEC response time may additionally be extended by 2 state times during internal code memory program execution.
- In the case that instruction N reads the PSW and instruction N-1 has an effect on the condition flags, the PEC response time may additionally be extended by 2 state times.

The worst case PEC response time during internal code memory program execution adds to 9 state times (18 TCL).

Any reference to external locations increases the PEC response time due to pipeline related access priorities. The following conditions must be considered:

- Instruction fetch from an external location
- Operand read from an external location
- Result write-back to an external location

Depending on where the instructions, source and destination operands are located, there are a number of combinations. Note, however, that only access conflicts contribute to the delay.

A few examples illustrate these delays:

- The worst case interrupt response time including external accesses will occur when instructions N and N+1 are executed out of external memory, instructions N-1 and N require external operand read accesses, and instructions N-3, N-2 and N-1 write back external operands. In this case, the PEC response time is the time to perform 7 word bus accesses.
- When instructions N and N+1 are executed from external memory, but all operands for instructions N-3 through N-1 are in internal memory, then the PEC response time is the time to perform 1 word bus access plus 2 state times.

After a request for PEC service has been acknowledged by the CPU, the execution of the next instruction is delayed by 2 state times plus the additional time it might take to fetch the source operand from internal code memory or external memory and to write the destination operand over the external bus in an external program environment.

Note: A bus access in this context includes all delays which can occur during an external bus cycle.

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 controlled via the sub-node interrupt control register ISNC which provides a separate request flag and enable bit for each supported request source. The interrupt level used for arbitration is determined by the node control register (… IC).

The specific request flags within ISNC must be reset by software. If the respective request is likely to be activated either at the time the request flag is cleared or shortly thereafter, the request flag should be cleared together with the corresponding enable bit. The enable bit can then be set again. This avoids undetected requests caused by pulses at the interrupt node being too short.

ISNC
Interrupt Sub-Node Ctrl. Reg. ESFR ($F1DE_H/EF_H$) Reset Value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	PLL IE	PLL IR	RTC IE	RTC IR
-	-	-	-	-	-	-	-	-	-	-	-	rw	rwh	rw	rwh

Bit	Function
xxIR	**Interrupt Request Flag for Source xx** 0: No request from source xx pending. 1: Source xx has raised an interrupt request.
xxIE	**Interrupt Enable Control Bit for Source xx** 0: Source xx interrupt request is disabled. 1: Source xx interrupt request is enabled.

Table 5-7 Sub-node Control Bit Allocation

Bit Position	Interrupt Source	Associated Node
15 … 4	*Reserved.*	*Reserved.*
3|2	PLL / OWD	XP3IC
1|0	RTC	XP3IC

Note: In order to ensure compatibility with other derivative devices application software should never set reserved bits within register ISNC.

5.8 External Interrupts

Although the C164CI 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.

Interrupt signals may be connected to:

- EX3IN … EX0IN, the fast external interrupt input pins,
- CC27IO … CC24IO, capture input / compare output lines of the CAPCOM2 unit,
- CC19IO … CC16IO, capture input / compare output lines of the CAPCOM2 unit,
- T4IN, T2IN, the timer input pins

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. 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.

Table 5-8 Pins Usable as External Interrupt Inputs

Port Pin	Original Function	Control Register
P1H.3-0/EX3-0IN	Fast external interrupt input pin	EXICON
P1H.7-4/CC27-24IO	CAPCOM Register 27-24 Capture Input	CC27-CC24
P8.3-0/CC19-16IO	CAPCOM Register 19-16 Capture Input	CC19-CC16
P5.6/T2IN	Auxiliary timer T2 input pin	T2CON
P5.7/T4IN	Auxiliary timer T4 input pin	T4CON

When port pins CCxIO are to be used as external interrupt input pins, bit field 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 bit fields 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.

Note: The non-maskable interrupt input pin $\overline{NMI}$ (sample rate 2 TCL) 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

Input pins which may be used for external interrupts are sampled every 16 TCL; that is, external events are scanned and detected in time frames of 16 TCL. The C164CI provides 4 interrupt inputs that are sampled every 2 TCL, so external events are captured faster than with standard interrupt inputs.

The lower 4 pins of Port P1H (P1H.3-P1H.0) can be programmed individually to this fast interrupt mode, where the trigger transition (rising, falling or both) also can be selected. The External Interrupt Control register EXICON controls this feature for all 4 pins.

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

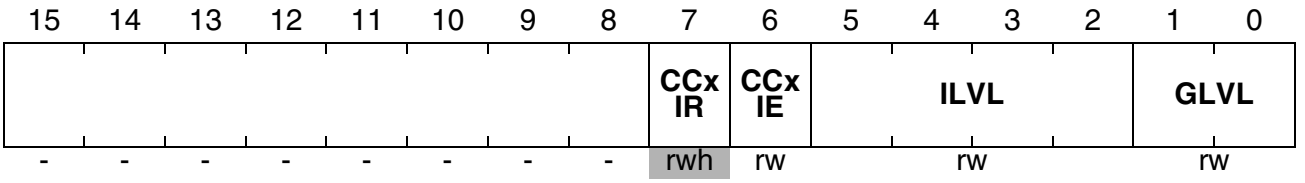
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-		-		-		-		EXI3ES		EXI2ES		EXI1ES		EXI0ES	
-		-		-		-		rw		rw		rw		rw	

Bit	Function
EXIxES	**External Interrupt x Edge Selection Field (x = 7 … 0)** 0 0: Fast external interrupts disabled: standard mode 0 1: Interrupt on positive edge (rising) 1 0: Interrupt on negative edge (falling) 1 1: Interrupt on any edge (rising or falling)

Note: The fast external interrupt inputs are sampled every 2 TCL. The interrupt request arbitration and processing, however, is executed every 8 TCL.

The interrupt control registers listed below (CC11IC … CC8IC) control the fast external interrupts of the C164CI. These fast external interrupt nodes and vectors are named according to the C167's CAPCOM channels CC11 … CC8, so interrupt nodes receive identical names throughout the architecture. See register description below.

CCxIC
CAPCOM x Intr. Ctrl. Reg. **SFR (See Table 5-9)** **Reset Value: - - 00_H**



Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

Table 5-9 Fast External Interrupt Control Register Addresses

Register	Address	External Interrupt
CC8IC	$FF88_H$ / $C4_H$	EX0IN
CC9IC	$FF8A_H$ / $C5_H$	EX1IN
CC10IC	$FF8C_H$ / $C6_H$	EX2IN
CC11IC	$FF8E_H$ / $C7_H$	EX3IN

External Interrupt Source Control

The input source for the fast external interrupts (controlled via register EXICON) can be derived either from the associated port pin EXnIN or from an alternate source. This selection is controlled via register EXISEL.

Activating the 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.

EXISEL
Ext. Interrupt Source Reg. 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
EXI7SS		EXI6SS		EXI5SS		EXI4SS		EXI3SS		EXI2SS		EXI1SS		EXI0SS	
rw		rw		rw		rw		rw		rw		rw		rw	

Bit	Function
EXIxSS	**External Interrupt x Source Selection Field (x = 7 … 0)** 00: Input from associated EXxIN pin. 01: Input from alternate pin. 10: Input from pin EXxIN ORed with alternate pin. 11: Input from pin EXxIN ANDed with alternate pin.

The **Table 5-10** summarizes the association of the bitfields of register EXISEL with the respective interface input lines.

Table 5-10 Connection of Interface Inputs to External Interrupt Nodes

Bitfield	Associated Interface Line	Notes
EXI0SS	CAN1_RxD	CAN (C164CI, C164CL) The used pin depends on the assignment for the module.
EXI2SS	RxD0	ASC0
EXI3SS	SCLK	SSC

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 wakeup. Therefore, during Sleep Mode, special edge detection logic for the fast external interrupts (EXzIN) is activated 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 wakeup signal by approximately 100 ns, but the spike suppression ensures safe and robust operation of the sleep/wakeup mechanism in an active environment.

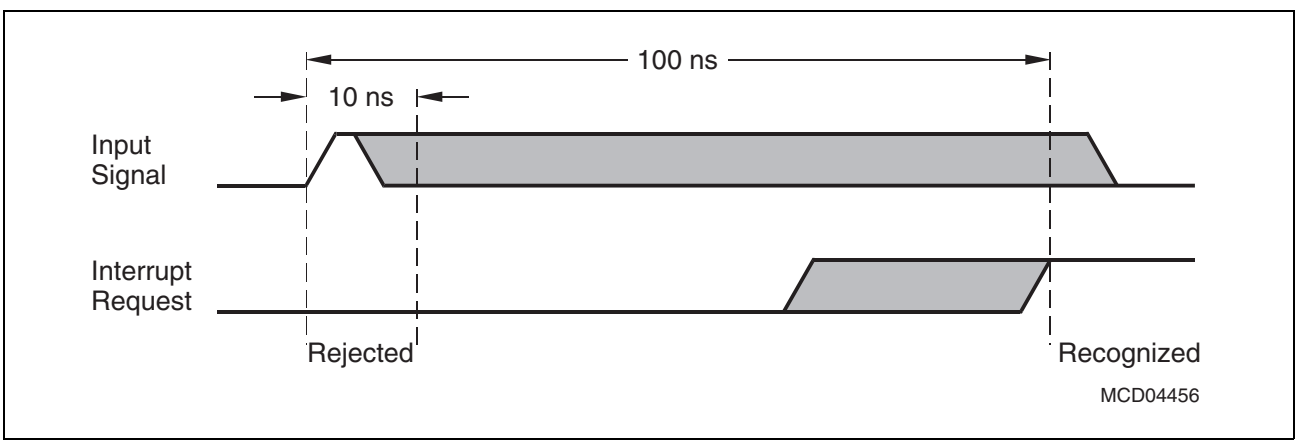


Figure 5-6 Input Noise Filter Operation

5.9 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 C164CI 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 trap number specified in the operand field of the trap instruction determines which vector location in the address range from $00'0000_H$ through $00'01FC_H$ 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 segmentation is enabled and a trap is executed, the CSP for the trap service routine is set to code segment 0. 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 level in register PSW is 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 can be interrupted by other traps or higher priority interrupts, other than when triggered by a hardware trap.

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 C164CI 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. Depending on the trap condition, the instruction which caused the trap is either completed or cancelled (i.e. it has no effect on the system state) 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 5-2**).

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 CSP is set to code segment zero, if segmentation is enabled. A trap service routine must be terminated with the RETI instruction.

The eight hardware trap functions of the C164CI are divided into two classes:

Class A traps are
- External Non-Maskable Interrupt ($\overline{\text{NMI}}$)
- Stack Overflow
- Stack Underflow trap

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

Class B traps are
- Undefined Opcode
- Protection Fault
- Illegal Word Operand Access
- Illegal Instruction Access
- Illegal External Bus Access Trap

These 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 stack underflow trap at the lowest priority.

All class B traps have the same trap priority (trap priority I). When several class B traps become active at 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 for simultaneously occurring class B traps is determined by software in the trap service routine.

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.

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	**STK OF**	**STK UF**	**-**	**-**	**-**	**-**	**-**	**UND OPC**	**-**	**-**	**-**	**PRT FLT**	**ILL OPA**	**ILL INA**	**ILL BUS**
rwh	rwh	rwh		-	-	-	-	rwh	-	-	-	rwh	rwh	rwh	rwh

Bit	Function
ILLBUS	**Illegal External Bus Access Flag** An external access has been attempted with no external bus defined.
ILLINA	**Illegal Instruction Access Flag** A branch to an odd address has been attempted.
ILLOPA	**Illegal Word Operand Access Flag** A word operand access (read or write) to an odd address has been attempted.
PRTFLT	**Protection Fault Flag** A protected instruction with an illegal format has been detected.
UNDOPC	**Undefined Opcode Flag** The currently decoded instruction has no valid C164CI opcode.
STKUF	**Stack Underflow Flag** The current stack pointer value exceeds the content of register STKUN.
STKOF	**Stack Overflow Flag** The current stack pointer value falls below the content of reg. STKOV.
NMI	**Non Maskable Interrupt Flag** A negative transition (falling edge) has been detected on pin $\overline{NMI}$.

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.

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.

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. The IP value pushed on the system stack is the address of the instruction following the one after which normal processing was interrupted by the NMI trap.

Note: The $\overline{\text{NMI}}$ pin is sampled with every CPU clock cycle to detect transitions.

Stack Overflow Trap

Whenever the stack pointer is decremented to a value which is less than 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. Which IP value will be pushed onto the system stack depends on which operation caused the decrement of the SP. When an implicit decrement of the SP is made through a PUSH or CALL instruction, or upon interrupt or trap entry, the IP value pushed is the address of the following instruction. When the SP is decremented by a subtract instruction, the IP value pushed represents the address of the instruction after the instruction following the subtract instruction.

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 incremented to a value greater than 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. Again, which IP value will be pushed onto the system stack depends on which operation caused the increment of the SP. When an implicit increment of the SP is made through a POP or return instruction, the IP value pushed is the address of the following instruction. When the SP is incremented by an add instruction, the pushed IP value represents the address of the instruction after the instruction following the add instruction.

Undefined Opcode Trap

When the instruction currently decoded by the CPU does not contain a valid C164CI opcode, the UNDOPC flag is set in register TFR and the CPU enters the undefined opcode trap routine. The IP value pushed onto the system stack is the address of the instruction that caused the trap.

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.

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, and SRVWDT. The IP value pushed onto the system stack for the protection fault trap is the address of the instruction which caused the trap.

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. The IP value pushed onto the system stack is the address of the instruction following the one which caused the trap.

Illegal Instruction Access Trap

Whenever a branch is made to an odd byte address, the ILLINA flag in register TFR is set and the CPU enters the illegal instruction access trap routine. The IP value pushed onto the system stack is the illegal odd target address of the branch instruction.

Illegal External Bus Access Trap

Whenever the CPU requests an external instruction fetch, data read or data write, and no external bus configuration has been specified, the ILLBUS flag in register TFR is set and the CPU enters the illegal bus access trap routine. The IP value pushed onto the system stack is the address of the instruction following the one which caused the trap.

6 Clock Generation

All activities of the C164CI's controller hardware and its on-chip peripherals are controlled via the system clock signal f_{CPU}.
This reference clock is generated in three stages, as shown in **Figure 6-1**:

- **Oscillator**
- **Frequency Control**
- **Clock Drivers**

Oscillator

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

Frequency Control

The input clock signal feeds the controller hardware:

- Directly, providing phase coupled operation on input frequency which is not too high
- Divided by 2 to obtain 50% duty cycle clock signal
- Via an on-chip Phase Locked Loop (PLL) providing maximum performance on low input frequency
- Via the Slow Down Divider (SDD) to reduce power consumption.

The resulting internal clock signal is referred to as "CPU clock" f_{CPU}.

Clock Drivers

The CPU clock is distributed via separate clock drivers which feed the CPU and two groups of peripheral modules. The Real Time Clock is fed with the prescaled oscillator clock (f_{RTC}) via a separate clock driver, so it is not affected by the clock control functions.

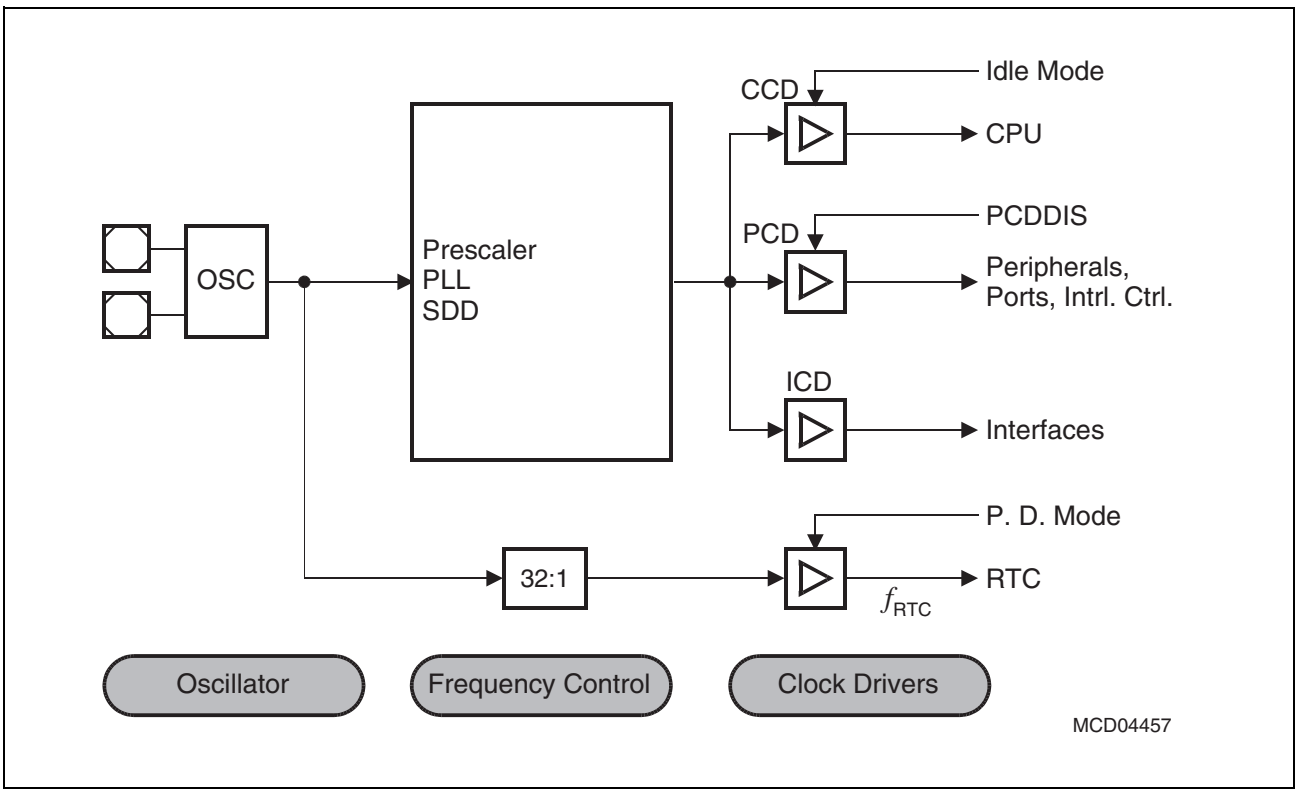


Figure 6-1 CPU Clock Generation Stages

6.1 Oscillator

The main oscillator of the C164CI 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 6-2**) comprises the crystal, two low end capacitors, and a series resistor (Rx2) to limit the current through the crystal. The additional LC combination is only required for 3rd overtone crystals to suppress oscillation in the fundamental mode. A test resistor (R_Q) may be temporarily inserted to measure the oscillation allowance of the oscillator circuitry.

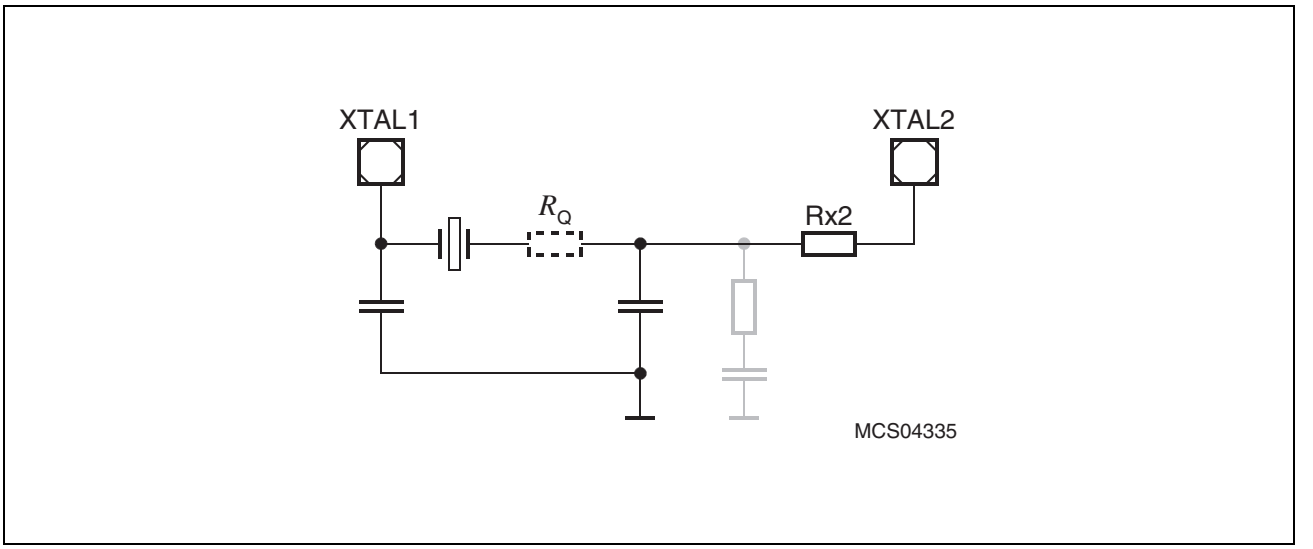


Figure 6-2 External Oscillator Circuitry

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 CPU frequency).

*Note: **Oscillator measurement** within the final target system is recommended to determine the actual oscillation allowance for the oscillator-crystal system. The measurement technique, examples for evaluated systems, and recommendations are provided in a specific application note about oscillators (available via your representative or WWW).*

For input frequencies above 25 … 30 MHz the oscillator's output should be terminated as shown in **Figure 6-3**; at lower frequencies it may be left open. This termination improves the operation of the oscillator by filtering out frequencies above the intended oscillator frequency.

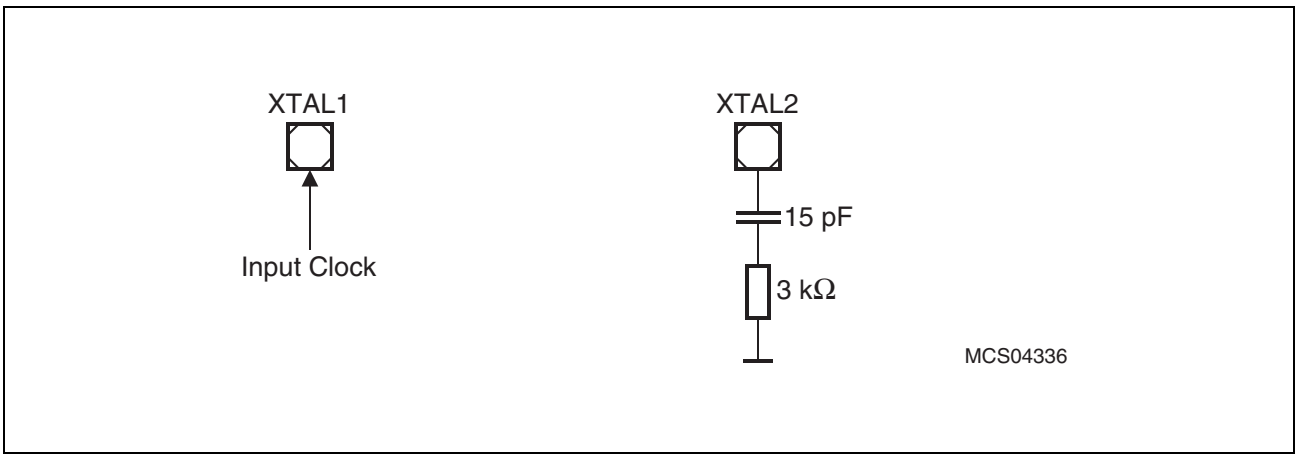


Figure 6-3 Oscillator Output Termination

Note: It is strongly recommended to measure the oscillation allowance (or margin) in the final target system (layout) to determine the optimum parameters for the oscillator operation.

6.2 Frequency Control

The CPU clock is based on either the Basic Clock or the Slow Down Clock. Both types of clock are generated from the oscillator clock and are software selectable:

The Basic Clock is the standard operating clock for the C164CI and is required to deliver the intended maximum performance. The clock configuration in register RP0H (bitfield CLKCFG = RP0H.7-5) determines one of three possible basic clock generation modes:

- Direct Drive: the oscillator clock is directly fed to the controller hardware.
- Prescaler: the oscillator clock is divided by 2 to achieve a 50% duty cycle.
- PLL: the oscillator clock is multiplied by a configurable factor of **F** = 1.5 … 5.

The Slow Down Clock is the oscillator clock divided by a programmable factor of 1 … 32 (additional 2:1 divider in prescaler mode). This alternate possibility runs the C164CI at a lower frequency, determined by the programmed slow down factor, and greatly reduces its power consumption.

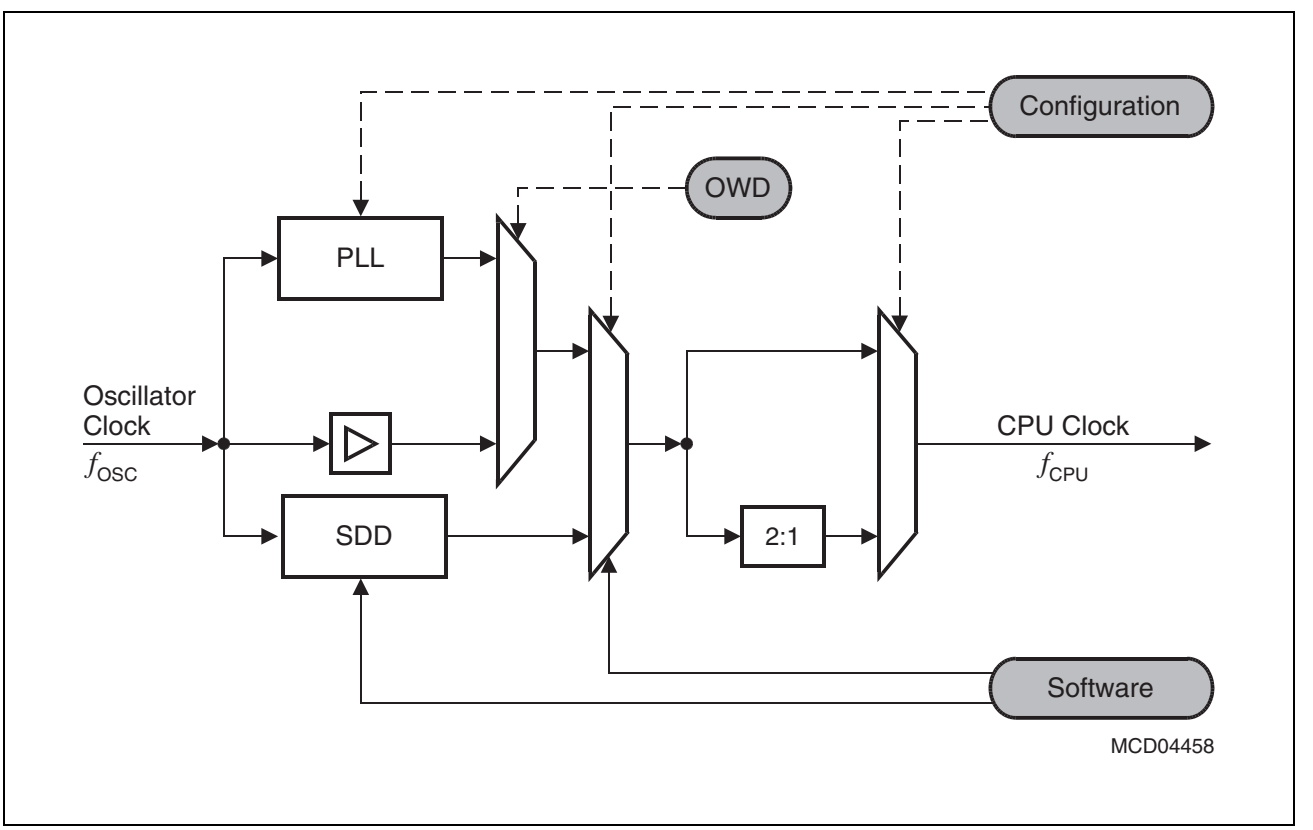


Figure 6-4 Frequency Control Paths

Note: The configuration register RP0H is loaded with the logic levels present on the upper half of PORT0 (P0H) after a long hardware reset, i.e. bitfield CLKCFG represents the logic levels on pins P0.15-13 (P0H.7-5).

The internal operation of the C164CI is controlled by the internal CPU clock f_{CPU}. Both edges of the CPU clock can trigger internal operations (example: pipeline) or external operations (example: bus cycles) operations. See **Figure 6-5**.

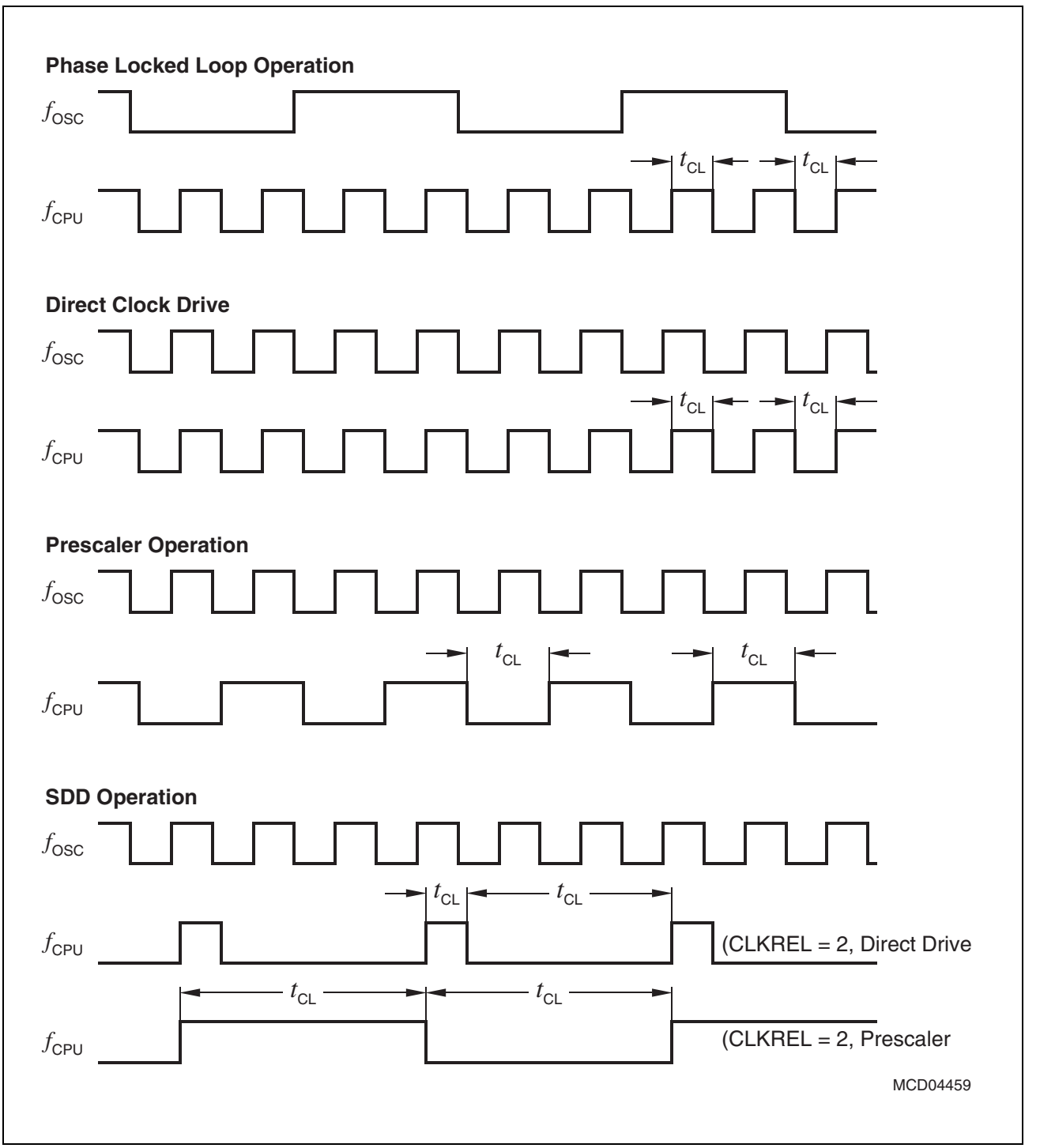


Figure 6-5 Generation Mechanisms for the CPU Clock

Direct Drive

When direct drive is configured (CLKCFG = 011_B), the C164CI's clock system is directly fed from the external clock input, i.e. $f_{CPU} = f_{OSC}$. This allows operation of the C164CI with a reasonably small fundamental mode crystal. The specified minimum values for the CPU clock phases (TCLs) must be respected. Therefore, the maximum input clock frequency depends on the clock signal's duty cycle.

Prescaler Operation

When prescaler operation is configured (CLKCFG = 001_B), the C164CI's input clock is divided by 2 to generate the CPU clock signal, i.e. $f_{CPU} = f_{OSC}/2$. This requires the oscillator (or input clock) to run on 2 times the intended operating frequency but guarantees a 50% duty cycle for the internal clock system independent of the input clock signal's waveform.

PLL Operation

When PLL operation is configured (via CLKCFG), the C164CI's input clock is fed to the on-chip Phase Locked Loop circuit which multiplies its frequency by a factor of **F** = 1.5 … 5 (selectable via CLKCFG, see **Table 6-1**) and generates a CPU clock signal with 50% duty cycle, i.e. $f_{CPU} = f_{OSC} \times \mathbf{F}$.

The on-chip PLL circuit allows operation of the C164CI on a low frequency external clock while still providing maximum performance. The PLL also provides fail safe mechanisms which allow detection of frequency deviations and 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 stable. 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 (2 … 5 MHz). The base frequency is still generated and allows the CPU to execute emergency actions in case of a loss of the external clock.

On power-up, the PLL provides a stable clock signal within about 1 ms after V_{DD} has reached the specified valid range, even if there is no external clock signal (in this case the PLL will run on its base frequency of 2 … 5 MHz). The PLL starts synchronizing with the external clock signal as soon as it is available. Within about 1 ms after stable oscillations of the external clock within the specified frequency range, the PLL will be synchronous with this clock at a frequency of $\mathbf{F} \times f_{OSC}$; meaning that the PLL locks to the external clock.

When PLL operation is selected, the CPU clock is a selectable multiple of the oscillator frequency, i.e. the input frequency.

Table 6-1 lists the possible selections.

Table 6-1 C164CI Clock Generation Modes

RP0H.7-5 (P0H.7-5)	CPU Frequency $f_{CPU} = f_{OSC} \times F$	External Clock Input Range [1]	Notes
1 1 1	$f_{OSC} \times 4$	2.5 to 6.25 MHz	Default configuration
1 1 0	$f_{OSC} \times 3$	3.33 to 8.33 MHz	–
1 0 1	$f_{OSC} \times 2$	5 to 12.5 MHz	–
1 0 0	$f_{OSC} \times 5$	2 to 5 MHz	–
0 1 1	$f_{OSC} \times 1$	1 to 25 MHz	Direct drive[2]
0 1 0	$f_{OSC} \times 1.5$	6.66 to 16.6 MHz	–
0 0 1	$f_{OSC} / 2$	2 to 50 MHz	CPU clock via prescaler
0 0 0	$f_{OSC} \times 2.5$	4 to 10 MHz	–

1) The external clock input range refers to a CPU clock range of 10 … 25 MHz.

2) The maximum frequency depends on the duty cycle of the external clock signal. In emulation mode pin P0.15 (P0H.7) is inverted, i.e. the configuration ‘111’ would select direct drive in emulation mode.

The PLL constantly synchronizes to the external clock signal. 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. This jitter is irrelevant for longer time periods. For short periods (1 … 4 CPU clock cycles), it remains below 4%.

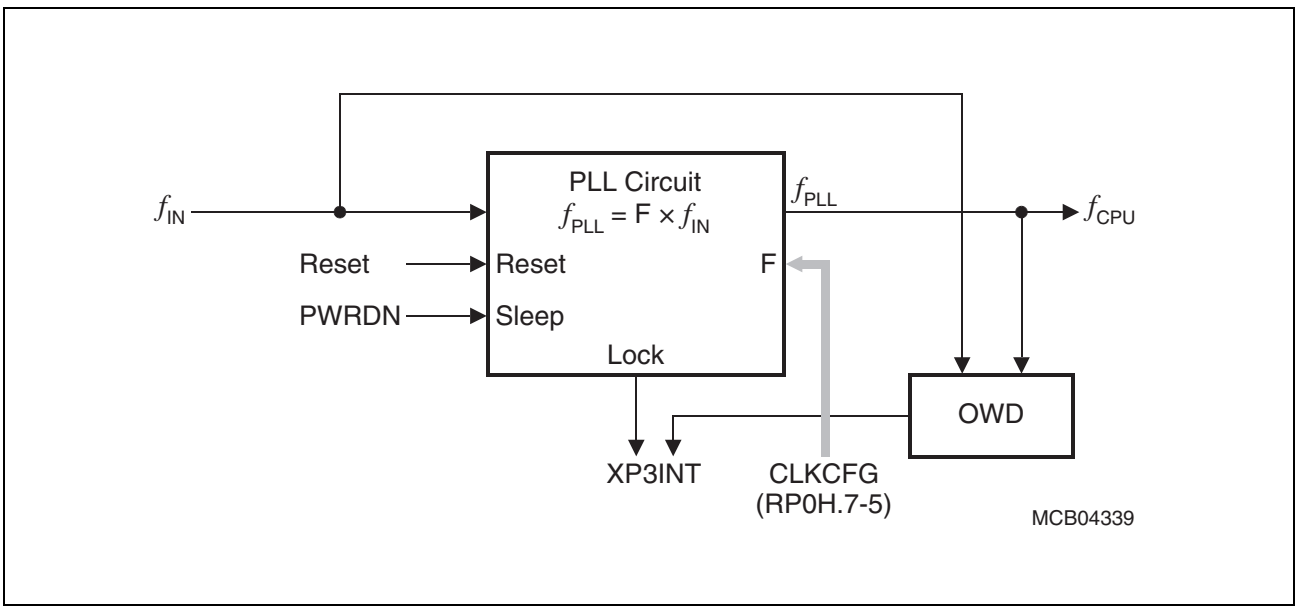


Figure 6-6 PLL Block Diagram

6.3 Oscillator Watchdog

The C164CI 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 prescaler or direct drive mode (unless the PLL provides the basic clock). For this operation, the PLL provides a clock signal (base frequency) which is used to supervise transitions on the oscillator clock. This PLL clock is independent of 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 the PLL clock signal instead of the selected oscillator clock (see **Figure 6-4**). Under these circumstances, the PLL will oscillate with its base frequency.

In direct drive mode the PLL base frequency is used directly (f_{CPU} = 2 … 5 MHz).
In prescaler mode, the PLL base frequency is divided by 2 (f_{CPU} = 1 … 2.5 MHz).

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

Using this PLL 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: The CPU clock source is switched back to the oscillator clock only after a hardware reset.

The Oscillator Watchdog can be disabled by setting bit OWDDIS in register SYSCON. In this case, the PLL remains idle and provides no clock signal, while the CPU clock signal is derived directly from the oscillator clock or via prescaler or SDD. Also, no interrupt request will be generated in case of a missing oscillator clock.

Note: At the end of an external reset, bit OWDDIS reflects the inverted level of pin $\overline{RD}$ at that time. Thus, the Oscillator Watchdog may also be disabled via hardware by (externally) pulling the $\overline{RD}$ line low upon a reset, in a manner similar to the standard reset configuration via PORT0.

The oscillator watchdog cannot provide full security while the CPU clock signal is generated by the SlowDown Divider because the OWD cannot switch to the PLL clock in this case (see **Figure 6-4**). OWD interrupts are only recognizable if f_{OSC} is still available (for instance, the input frequency is too low or for intermittent failure only).
A broken crystal cannot be detected by software (OWD interrupt server) as no SDD clock is available in such a case.

6.4 Clock Drivers

The operating clock signal f_{CPU} is distributed to the controller hardware via several clock drivers which are disabled under certain circumstances. The Real Time Clock (RTC) is clocked via a separate clock driver which delivers the prescaled oscillator clock (contrary to the other clock drivers). **Table 6-2** summarizes the different clock drivers and their functions, especially in power reduction modes:

Table 6-2 Clock Drivers Description

Clock Driver	Clock Signal	Active Mode	Idle Mode	Power Down and Sleep Mode	Connected Circuitry
CCD CPU Clock Driver	f_{CPU}	ON	Off	Off	CPU, internal memory modules (IRAM, ROM/OTP/Flash)
ICD Interface Clock Driver	f_{CPU}	ON	ON	Off	ASC0, WDT, SSC, interrupt detection circuitry
PCD Peripheral Clock Driver	f_{CPU}	Control via PCDDIS	Control via PCDDIS	Off	(X)Peripherals (timers, etc.) except those driven by ICD, interrupt controller, ports
RCD RTC Clock Driver	f_{RTC}	ON	ON	Control via PDCON / SLEEP-CON	Real Time Clock

Note: Disabling PCD by setting bit PCDDIS stops the clock signal for all connected modules. Ensure that all these modules are in a safe state before stopping their clock signal.
The port input and output values will not change while PCD is disabled (ASC0 and SSC will still operate, if active),
CLKOUT will be high if enabled.
*Please also utilize the hints given in **Section 21.5**.*

7 Parallel Ports

In order to accept or generate single external control signals or parallel data, the C164CI provides up to 59 parallel IO lines organized as follows: four 8-bit IO ports (PORT0 made of P0H and P0L, PORT1 made of P1H and P1L), one 9-bit IO port (Port 3), one 6-bit IO port (Port 4), one 4-bit IO port (Port 8), and one 8-bit input port (Port 5).

These port lines may be used for general purpose Input/Output functions controlled via software or may be used implicitly by the C164CI's integrated peripherals or by the External Bus Controller.

All port lines are bit addressable, and all input/output lines are individually (bit-wise) programmable as inputs or outputs via direction registers (excluding Port 5, which is an input only port). The IO ports are true bidirectional ports which are switched to high impedance state when configured as inputs. The output drivers of three IO ports (Ports 3, 4, and 8) can be configured (pin by pin) for push/pull operation or open-drain operation via control registers.

The logic level of a pin is clocked into the input latch once per state time, regardless of whether the port is configured for input or output.

A write operation to a port pin configured as an input causes the value to be written into the port output latch, while a read operation returns the latched state of the pin itself. A read-modify-write operation reads the value of the pin, modifies it, and writes it back to the output latch.

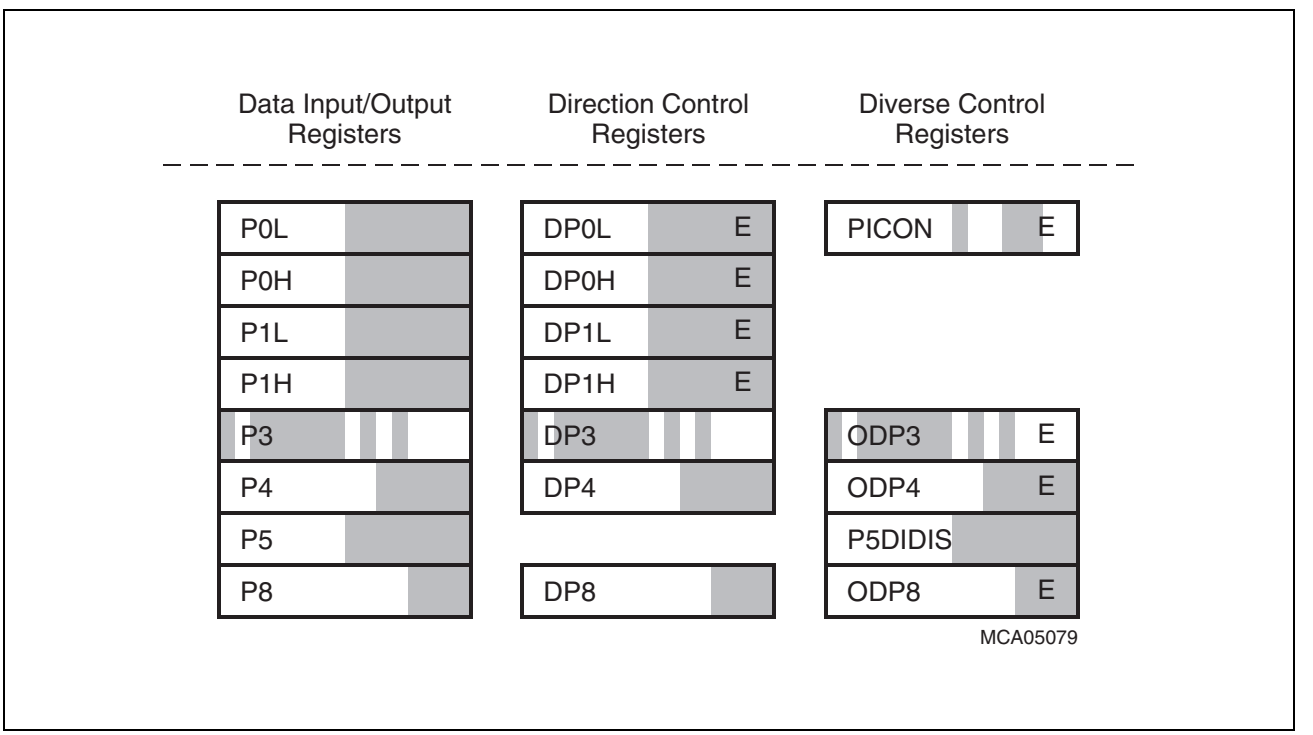


Figure 7-1 SFRs and Pins Associated with the Parallel Ports

Writing to a pin configured as an output (DPx.y = ‘1’) causes the output latch and the pin to have the written value, because the output buffer is enabled. Reading this pin returns the value of the output latch. A read-modify-write operation reads the value of the output latch, modifies it, and writes it back to the output latch, thus also modifying the level at the pin.

7.1 Input Threshold Control

The standard inputs of the C164CI 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 if the respective input signal level is near the thresholds.

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

PICON
Port Input Control Reg. **ESFR ($F1C4_H$/$E2_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								P8L IN	**-**	**-**	**P4L IN**	**P3H IN**	**P3L IN**	**-**	**-**
-	-	-	-	-	-	-	-	rw	-	-	rw	rw	rw	-	-

Bit	Function
PxLIN	**Port x Low Byte Input Level Selection** 0: Pins Px.7 … Px.0 switch on standard TTL input levels 1: Pins Px.7 … Px.0 switch on special threshold input levels
PxHIN	**Port x High Byte Input Level Selection** 0: Pins Px.15 … Px.8 switch on standard TTL input levels 1: Pins Px.15 … Px.8 switch on special threshold input levels

All options for individual direction and output mode control are available for each pin independent of the selected input threshold.

The input hysteresis provides stable inputs from noisy or slowly changing external signals.

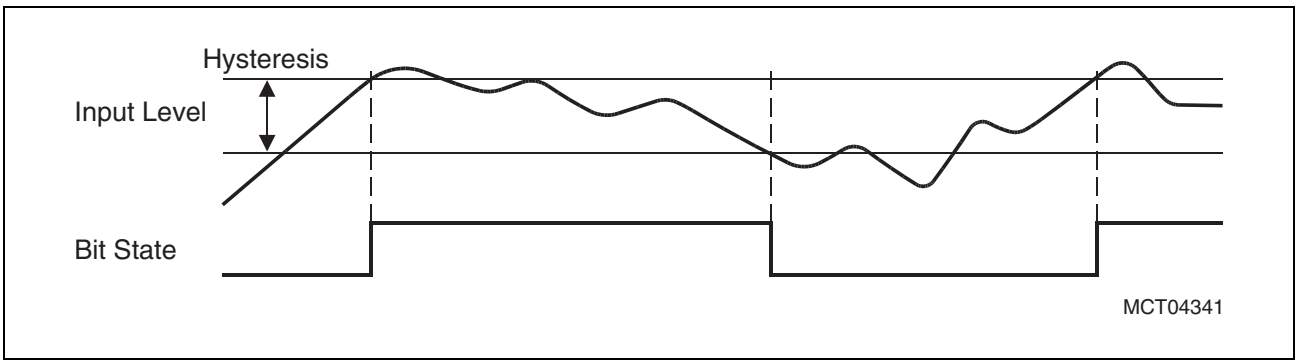


Figure 7-2 Hysteresis for Special Input Thresholds

7.2 Output Driver Control

The output driver of a port pin is activated by switching the respective pin to output, that is, DPx.y = ‘1’. The value 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 (delivered current) can be selected.
- **Edge Characteristic**: The shape of an output signal can be selected.

Open Drain Mode

In the C164CI, certain ports provide Open Drain Control, which allows switching 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 to either a high or a low level. In open drain mode, the upper transistor is always switched off, and the output driver can actively drive the line to a low level only. 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 provided for each port which 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.

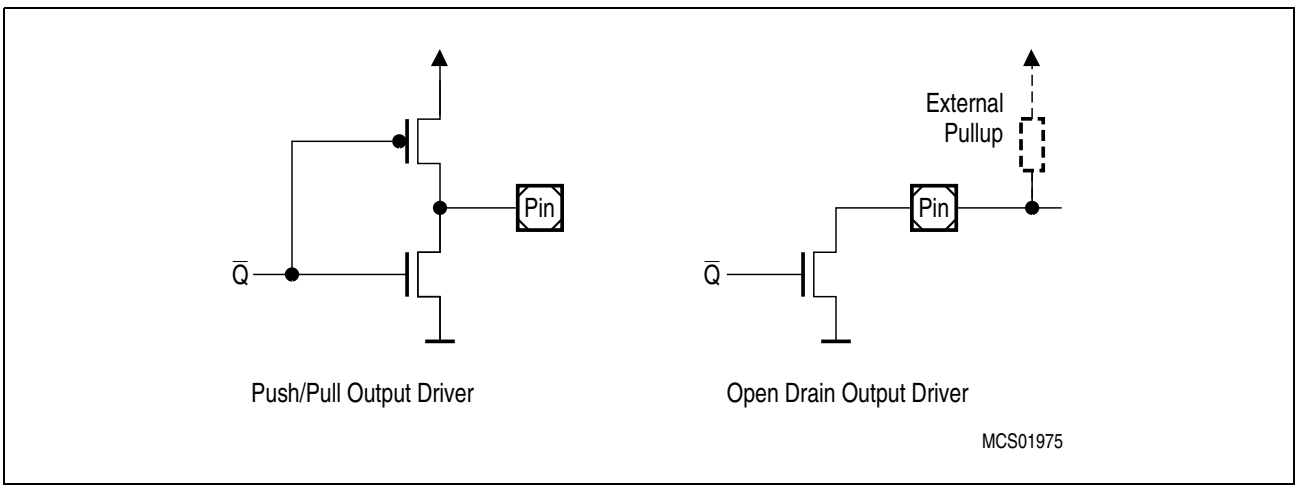


Figure 7-3 Output Drivers in Push/Pull Mode and in Open Drain Mode

Driver Characteristic

This defines the general driving capability of the respective driver. 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 C164CI pins feature differently sized transistors for each direction (push and pull). The activation/deactivation of these transistors determines the output characteristics of the respective port driver.

Three modes can be selected to adapt the driver characteristics to the application's requirements: Strong Driver Mode, Medium Driver Mode, and Weak Driver Mode.

In Strong Driver Mode, all transistors are activated. In this case, the driver provides maximum output current supporting high current applications such as LEDs or applications where fast signal transitions are required such as buses.

In Medium Driver Mode, not all transistors are activated. In this case, the driver provides a reduced output current supporting applications with moderate requirements for current or speed, while improving the EME behavior.

In Weak Driver Mode, only a small transistor is activated. In this case, the driver may drive not time critical logic loads, while switching noise is greatly reduced.

Edge Characteristic

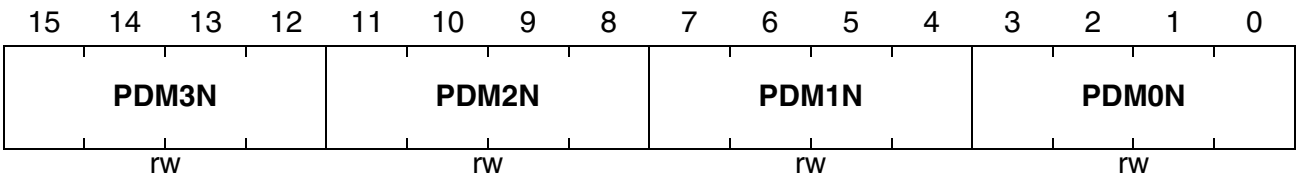
This defines the turn-on speed of the main driver stage, that is, the shape of the respective output. Soft edges reduce the peak currents drawn when changing the voltage level of an external capacitive load. For a bus interface, however, sharp edges may still be required. Edge characteristic affects the pre-driver which controls the final output driver stage.

Note: If Weak Driver Mode is selected additional edge shaping makes no sense and, therefore, is not supported.

The **Port Output Control registers** POCONx provide the corresponding control bits. For each group of four pins (that is, for each port nibble), a 4-bit control field is provided. 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.

The general register layout shown below is valid for all POCON registers. Please note that for byte ports, only two bitfields are provided (see **Table 7-1**).

POCON*
Port * Output Ctrl. Reg. **ESFR (Table 7-1)** **Reset Value: (Table 7-1)**



Bit	Function
PDMxN	**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[3)], no edge control 0100: Medium driver, Sharp edge mode 0101: Medium driver, Medium edge mode 0110: Medium driver, Soft edge mode 0111: Weak driver[3)], no edge control 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) This is the driver's minimum strength. No additional edge shaping can be selected at this level.

Table 7-1 lists the defined POCON registers and the allocation of control bitfields and port pins.

Table 7-1 Port Output Control Register Allocation

Control Register	Location	Controlled Pins (by POCONx.y-z)				Reset Value
		.15-12	.11-8	.7-4	.3-0	
POCON20[1]	$F0AA_H$ / 55_H	$\overline{RSTOUT}$	CLKOUT/ FOUT	ALE	$\overline{WR}$, $\overline{RD}$, $\overline{BHE}$/$\overline{WH}$	0000_H
POCON8	$F092_H$ / 49_H	- - -	- - -	- - -	P8.3-0	0022_H
POCON4[2]	$F08C_H$ / 46_H	- - -	- - -	P4.6-5	P4.3-0	0010_H
POCON3[3]	$F08A_H$ / 45_H	P3.15-12	P3.11-8	P3.7-4	- - -	2222_H
POCON1H	$F086_H$ / 43_H	- - -	- - -	P1H.7-4	P1H.3-0	0011_H
POCON1L	$F084_H$ / 42_H	- - -	- - -	P1L.7-4	P1L.3-0	0011_H
POCON0H	$F082_H$ / 41_H	- - -	- - -	P0H.7-4	P0H.3-0	0011_H
POCON0L	$F080_H$ / 40_H	- - -	- - -	P0L.7-4	P0L.3-0	0011_H

1) No associated port.

2) P4.7, P4.4 missing.

3) P3.14, P3.7, P3.5 missing.

Port Driver Temperature Compensation

The temperature compensation for the port drivers provides driver output characteristics which are stable (within a certain band of parameter variation) over the specified temperature range, e.g. -40 °C … +125 °C. The drive capability of the output drivers is reduced when the temperature is not in the upper range to improve the EME behavior.

The temperature compensation is based on a reference clock signal which is derived from the CPU clock by a programmable divider (see **Figure 7-4**).

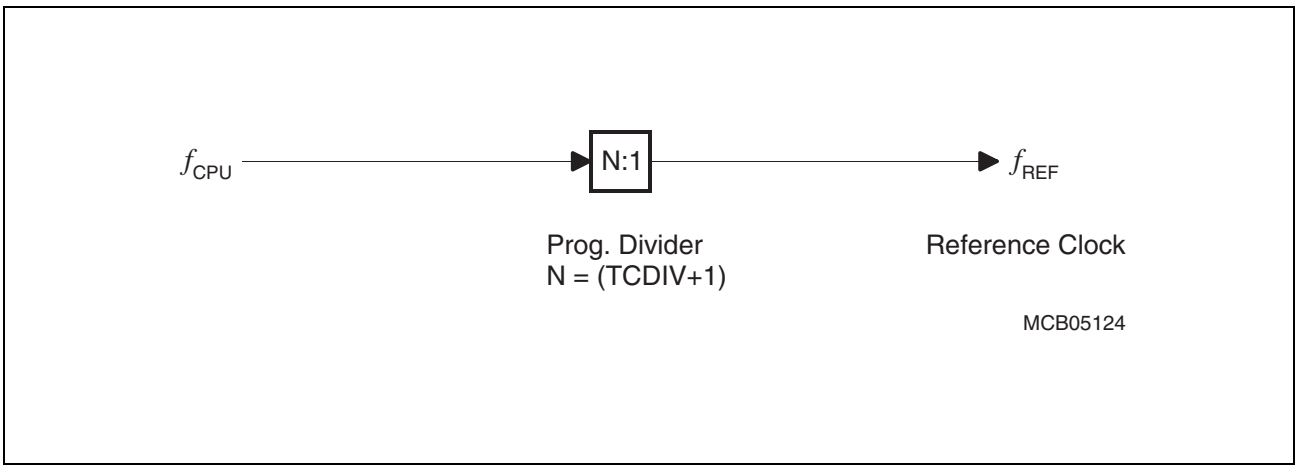


Figure 7-4 Temperature Compensation Clock Generation

The clock divider is programmed via bitfield TCDIV in register PTCR. TCDIV can be calculated using the following formula:

TCDIV = Integer (($f_{CPU} \times 6.7$) - 2) [f_{CPU} in MHz]

Example for f_{CPU} = 25 MHz:

TCDIV = Integer ((25 × 6.7) - 2) = 165 (= $A5_H$).

Generally, temperature compensation is a transparent feature. The Port Temperature Compensation Register PTCR provides access to the actual compensation value and even allows software control of this mechanism.

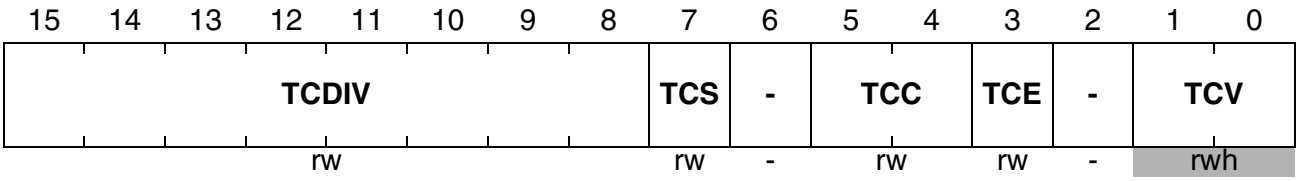
This is useful in two cases:

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

Temperature compensation is initialized using register PTCR (enable and prescaler for reference clock).

The reference clock is used to generate a temperature-related count value which is compared to three thresholds (temperature levels) at which the four control values (max, high, low, min) are switched.

PTCR
Port Temp. Comp. Reg. **ESFR ($F0AE_H/57_H$)** **Reset Value: 0000_H**



Bit	Function
TCV	**Temperature Compensation Value** The value which is currently generated by the temperature compensation sensor. This value is fed to the port logic while bit TCS = ‘1’. 00: Maximum driver strength (no reduction), i.e. very high temperature … … 11: Minimum driver strength (reduction for compensation), i.e. very low temperature *Note: Bitfield TCV returns 00_B when bit TCE = ‘0’ (always after reset).*
TCE	**Temperature Compensation Enable** 0: The temperature compensation sensor is deactivated (default). The port drivers are not reduced (TCV = 00_B). 1: The temperature compensation is active.
TCC	**Temperature Compensation Control** This value is fed to the port logic instead of the temperature compensation sensor value, while bit TCS = ‘0’. Encoding equal to TCV.
TCS	**Temperature Compensation Source** 0: Port logic is controlled by software via bitfield TCC. 1: Port logic is controlled by the temperature compensation sensor.
TCDIV	**Temperature Compensation Clock Divider** This value adjusts the temperature compensation logic to the selected operating frequency (see description).

7.3 Alternate Port Functions

To maximize flexibility for different applications and their specific IO requirements, port lines have programmable alternate input or output functions associated with them.

Table 7-2 Summary of Alternate Port Functions

Port	Alternate Function(s)	Alternate Signal(s)
PORT0	Address and data lines when accessing external resources (e.g. memory)	AD15 … AD0
PORT1	Address lines when accessing external resources (e.g. memory), Capture inputs or compare outputs of the CAPCOM units, Fast external interrupt inputs, CAPCOM timer input	A15 … A0, CC27IO … CC24IO, $\overline{\text{CTRAP}}$, CC6n, COUT6n, $\overline{\text{CC6POSn}}$, EX3IN … EX0IN, T7IN
Port 3	System clock or programmable frequency output, Optional bus control signal, Input/output functions of serial interfaces and timers	CLKOUT/FOUT, $\overline{\text{BHE}}$/$\overline{\text{WRH}}$, RxD0, TxD0, MTSR, MRST, SCLK, T3IN, T3EUD
Port 4	Selected segment address lines in systems with more than 64 KBytes of external resources Optional chip select output signals, CAN interface (when assigned)	A21 … A16, $\overline{\text{CS3}}$ … $\overline{\text{CS0}}$, CAN1_TxD, CAN1_RxD
Port 5	Analog input channels to the A/D converter, Timer control signal inputs	AN7 … AN0, T2EUD, T4EUD, T2IN, T4IN
Port 8	Capture inputs or compare outputs of the CAPCOM2 unit, CAN interface (when assigned)	CC19IO … CC16IO, CAN1_TxD, CAN1_RxD

If an **alternate output function** of a pin is to be used, the alternate data must be routed to the output driver and the driver must be enabled. For some alternate output signals, such as bus signals or X-Peripheral signals, this is done automatically via separate control lines, indicated by control line “AltEN” in the subsequent port figures. For the remaining alternate output signals this has to be accomplished by user software. If the alternate signal is combined with the port latch signal the respective port latch must be set accordingly (see individual port figures). The output driver must be enabled by switching the pin to output (DPx.y = ‘1’). Otherwise, the pin remains in the high-impedance state and is not affected by the alternate output function.

If an **alternate input function** of a pin 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. If no external device is connected to the pin, however, one can also set the direction for this pin to output. In this case, the pin reflects the state of the port output latch. Thus, the alternate input function reads the value stored in the port output latch. This can be used for testing purposes to allow a software trigger of an alternate input function by writing to the port output latch.

On most of the port lines, the user software is responsible for setting the proper direction when using an alternate input or output function of a pin. This is done by setting or clearing the direction control bit DPx.y of the pin before enabling the alternate function. There are port lines, however, for which the direction of the port line is switched automatically. For instance, in the multiplexed external bus modes of PORT0, the direction must be switched several times for an instruction fetch in order to output the addresses and to input the data. Obviously, this cannot be done through instructions. In these cases, the direction of the port line is switched automatically by hardware if the alternate function of such a pin is enabled.
To determine the appropriate level of the port output latches, check how the alternate data output is combined with the respective port latch output.

There is one basic structure for all port lines having only an alternate input function. Port lines having only an alternate output function, however, have different structures due to the way the direction of the pin is switched and depending on whether or not the pin is accessible by the user software in the alternate function mode.

All port lines 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 to avoid undesired transitions on the output pins.

This applies to single pins as well as to pin groups (see examples below).

```
OUTPUT_ENABLE_SINGLE_PIN:
BSET    P4.0              ;Initial output level is 'high'
BSET    DP4.0             ;Switch on the output driver

OUTPUT_ENABLE_PIN_GROUP:
BFLDL   P4, #05H, #05H    ;Initial output level is 'high'
BFLDL   DP4, #05H, #05H   ;Switch on the output drivers
```

*Note: When using several BSET pairs to control more pins of one port, these pairs must be separated by instructions which do not reference the respective port (see **Section 4.2**).*

Each of these ports and the alternate input and output functions are described in detail in the following subsections.

7.4 PORT0

The two 8-bit ports P0H and P0L represent the higher and lower parts of PORT0, respectively. Both halves of PORT0 can be written (e.g. via a PEC transfer) without affecting 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 DP0H and DP0L.

P0L
PORT0 Low Register **SFR ($FF00_H/80_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								P0L.7	P0L.6	P0L.5	P0L.4	P0L.3	P0L.2	P0L.1	P0L.0
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

P0H
PORT0 High Register **SFR ($FF02_H/81_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								P0H.7	P0H.6	P0H.5	P0H.4	P0H.3	P0H.2	P0H.1	P0H.0
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
P0X.y	Port data register P0H or P0L bit y

DP0L
P0L Direction Ctrl. Register **ESFR ($F100_H/80_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								DP0L.7	DP0L.6	DP0L.5	DP0L.4	DP0L.3	DP0L.2	DP0L.1	DP0L.0
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

DP0H
P0H Direction Ctrl. Register **ESFR ($F102_H/81_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								DP0H.7	DP0H.6	DP0H.5	DP0H.4	DP0H.3	DP0H.2	DP0H.1	DP0H.0
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
DP0X.y	**Port direction register DP0H or DP0L bit y** DP0X.y = 0: Port line P0X.y is an input (high-impedance) DP0X.y = 1: Port line P0X.y is an output

Alternate Functions of PORT0

When an external bus is enabled, PORT0 is used as data bus or address/data bus. Note that an external 8-bit demultiplexed bus only uses P0L, while P0H is free for IO (provided that no other bus mode is enabled).

PORT0 is also used to select the system startup configuration. During reset, PORT0 is configured to input and each line is held high through an internal pull-up device. Each line can now be individually pulled to a low level (see DC-level specifications in the respective Data Sheets) through an external pull-down device. A default configuration is selected when the respective PORT0 lines are at a high level. By pulling individual lines to a low level, this default can be changed according to the needs of the applications.

The internal pull-up devices are designed such that external pull-down resistors (see Data Sheet specification) can be used to apply a correct low level. These external pull-down resistors can remain connected to the PORT0 pins also during normal operation, however, care must be taken such that they do not disturb the normal function of PORT0 (this might be the case, for example, if the external resistor is too strong).

On completion of reset, the selected bus configuration will be written to the BUSCON0 register. The configuration of the high byte of PORT0 will be copied into the special register RP0H. This read-only register holds the selection for the number of chip selects

and segment addresses. Software can read this register in order to react according to the selected configuration, if required.

When the reset is terminated, the internal pull-up devices are switched off, and PORT0 will be switched to the appropriate operating mode.

During external accesses in multiplexed bus modes, PORT0 first outputs the 16-bit intra-segment address as an alternate output function. PORT0 is then switched to high-impedance input mode to read the incoming instruction or data. In 8-bit data bus mode, two memory cycles are required for word accesses, the first for the low byte and the second for the high byte of the word. During write cycles, PORT0 outputs the data byte or word after outputting the address.
During external accesses in demultiplexed bus modes PORT0 reads the incoming instruction or data word or outputs the data byte or word.

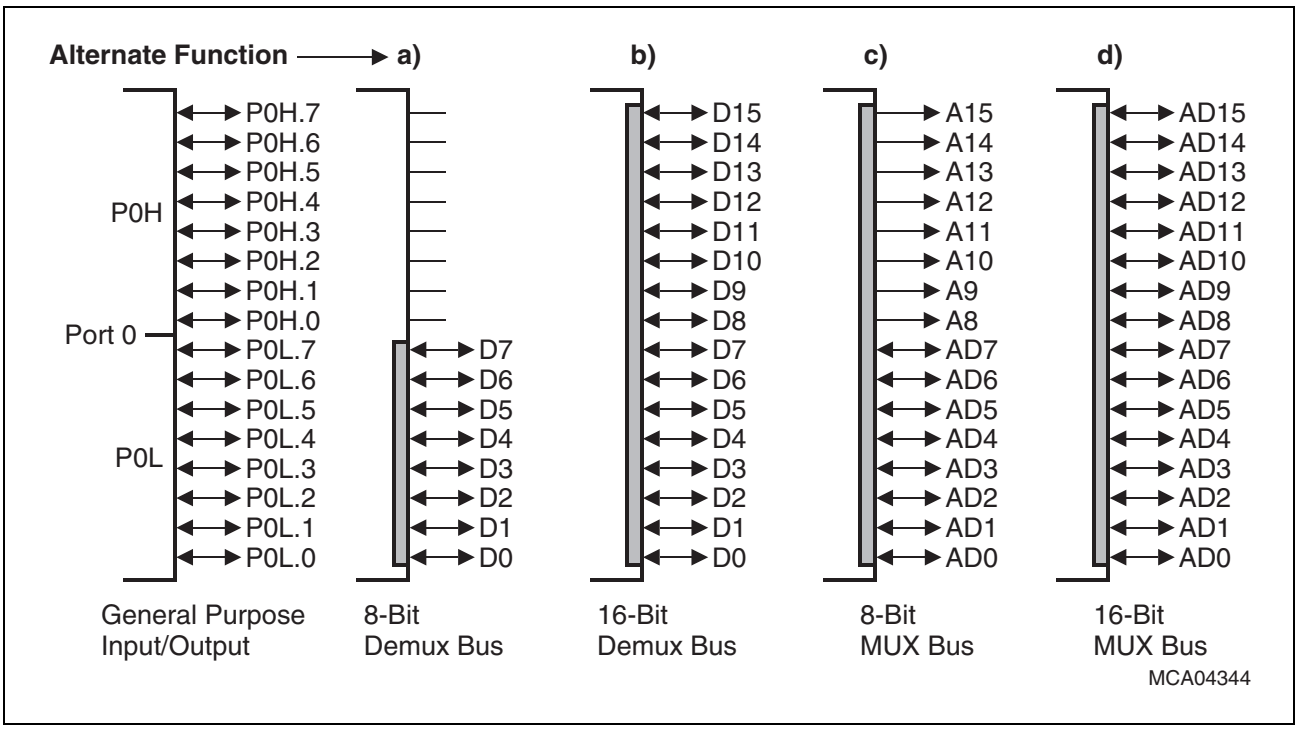
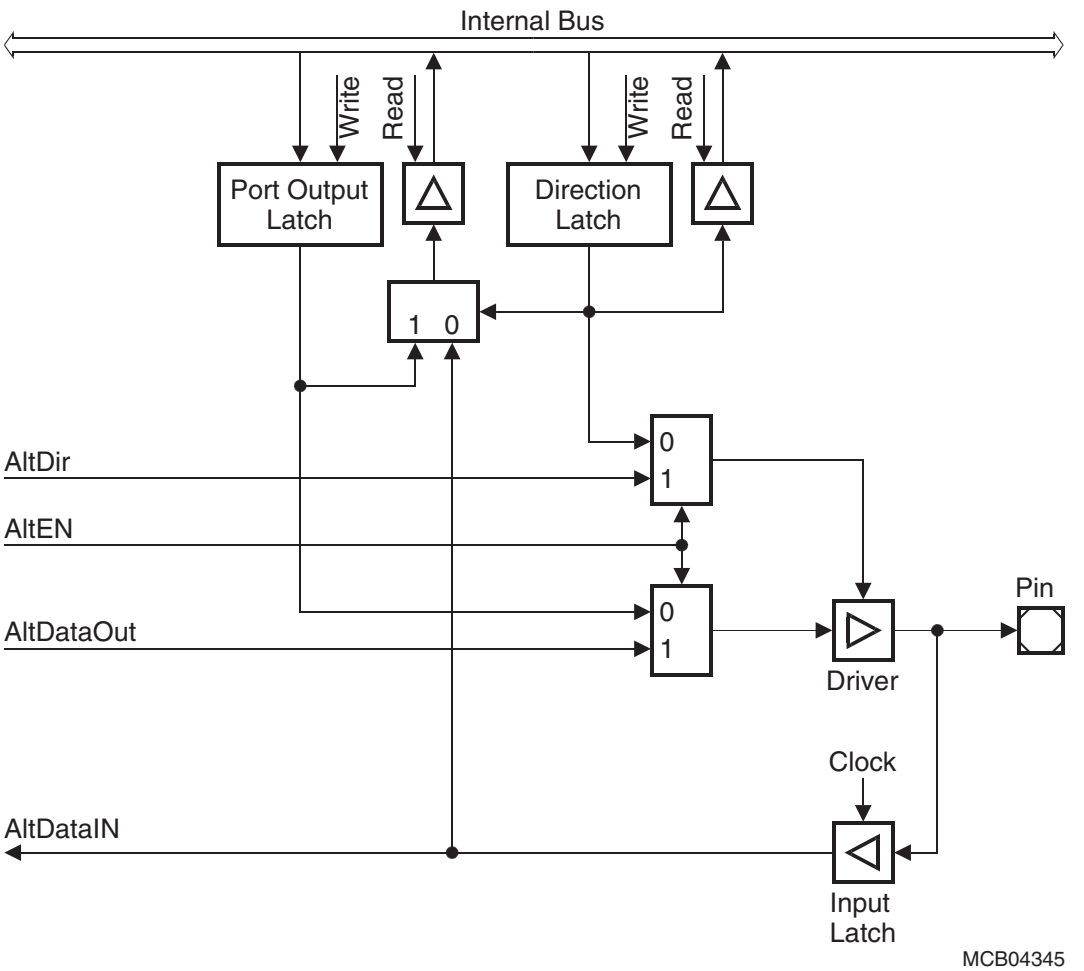


Figure 7-5 PORT0 IO and Alternate Functions

While external bus cycles are executed, PORT0 is controlled by the bus controller. The port direction is determined by the type of the bus cycle, the data are transferred directly from/to the bus controller hardware. The alternate output data can be the 16-bit intrasegment address or the 8/16-bit data information. While PORT0 is not used by the bus controller, it is controlled by its direction and output latch registers. User software must therefore be very careful when writing to PORT0 registers while the external bus is enabled. In most cases keeping the reset values will be the best choice.

Figure 7-6 shows the structure of a PORT0 pin.



P0H.7-0, P0L.7-0

Figure 7-6 Block Diagram of a PORT0 Pin

7.5 PORT1

The two 8-bit ports P1H and P1L represent the higher and lower part of PORT1, respectively. Both halves of PORT1 can be written (e.g. via a PEC transfer) without affecting 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
PORT1 Low Register **SFR ($FF04_H/82_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								P1L.7	P1L.6	P1L.5	P1L.4	P1L.3	P1L.2	P1L.1	P1L.0
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

P1H
PORT1 High Register **SFR ($FF06_H/83_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								P1H.7	P1H.6	P1H.5	P1H.4	P1H.3	P1H.2	P1H.1	P1H.0
-	-	-	-	-	-	-	-	rwh	rwh	rwh	rwh	rw	rw	rw	rw

Bit	Function
P1X.y	Port data register P1H or P1L bit y

DP1L
P1L Direction Ctrl. Register **ESFR ($F104_H/82_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								DP1 L.7	DP1 L.6	DP1 L.5	DP1 L.4	DP1 L.3	DP1 L.2	DP1 L.1	DP1 L.0
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

DP1H
P1H Direction Ctrl. Register **ESFR ($F106_H/83_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								DP1 H.7	DP1 H.6	DP1 H.5	DP1 H.4	DP1 H.3	DP1 H.2	DP1 H.1	DP1 H.0
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
DP1X.y	**Port direction register DP1H or DP1L bit y** DP1X.y = 0: Port line P1X.y is an input (high-impedance) DP1X.y = 1: Port line P1X.y is an output

Alternate Functions of PORT1

When a demultiplexed external bus is enabled, PORT1 is used as address bus.
Note that demultiplexed bus modes use PORT1 as a 16-bit port. Otherwise, all 16 port lines can be used for general purpose IO.

The lower 11 pins of PORT1 (P1H.2 … P1L.0) serve as the inputs/outputs for the CAPCOM6 unit.

Pins P1H.3 … P1H.0 accept the fast external inputs. P1H.3 also serves as input for timer T7.

The upper four pins of PORT1 (P1H.7 … P1H.4) also serve as capture inputs or compare outputs for the CAPCOM2 unit (CC27IO … CC24IO).
As all other capture inputs, the capture input function of pins P1H.7 … P1H.4 can also be used as external interrupt inputs (sample rate 16 TCL).

As a side benefit, the capture input capability of these lines can also be used in the address bus mode. In this way, changes to the upper address lines could be detected and could trigger an interrupt request in order to perform some special service routines. External capture signals can be applied only if no address output is selected for PORT1.

During external accesses in demultiplexed bus modes, PORT1 outputs the 16-bit intra-segment address as an alternate output function.

During external accesses in multiplexed bus modes, when **no** BUSCON register selects a demultiplexed bus mode, PORT1 is not used and is available for general purpose IO.

When an external bus mode is enabled, the direction of the port pin and the loading of data into the port output latch are controlled by the bus controller hardware. The input of the port output latch is disconnected from the internal bus and is switched to the line labeled “Alternate Data Output” via a multiplexer. The alternate data is the 16-bit intrasegment address. While an external bus mode is enabled, the user software should not write to the port output latch, otherwise, unpredictable results may occur. When the external bus modes are disabled, the contents of the direction register last written by the user become active.

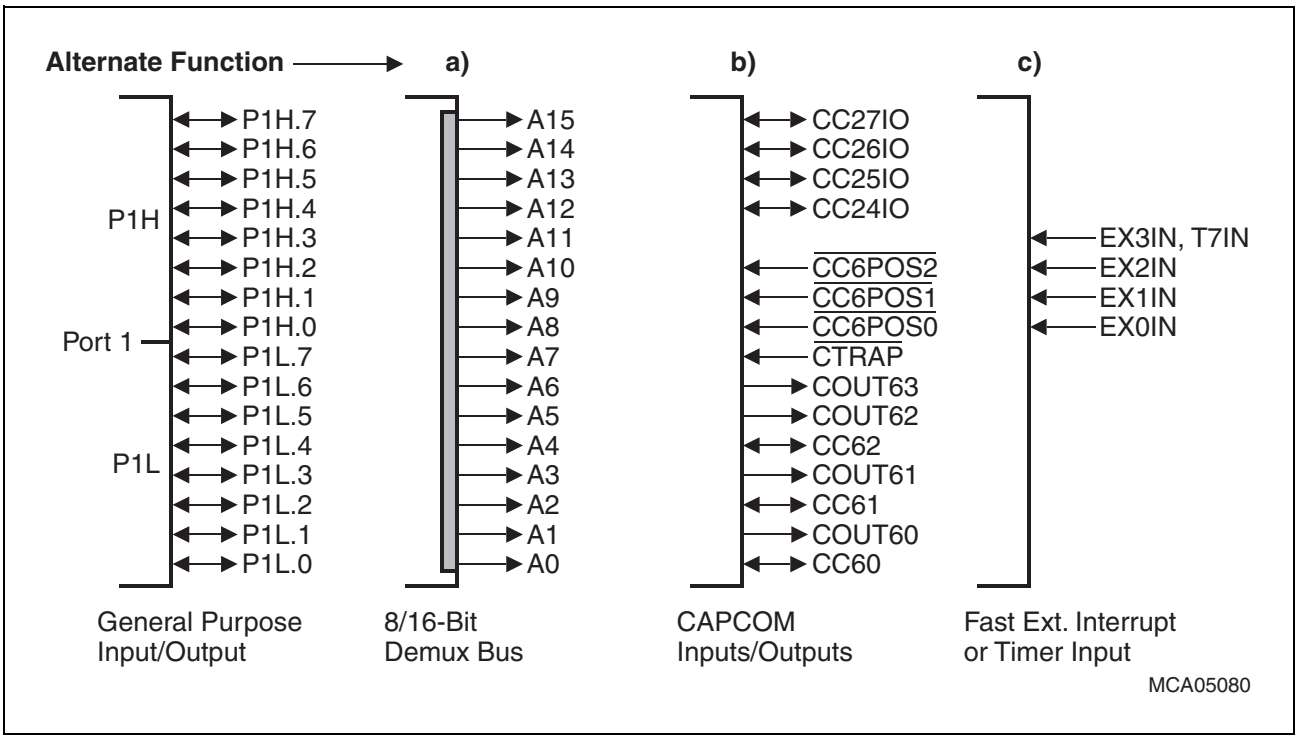
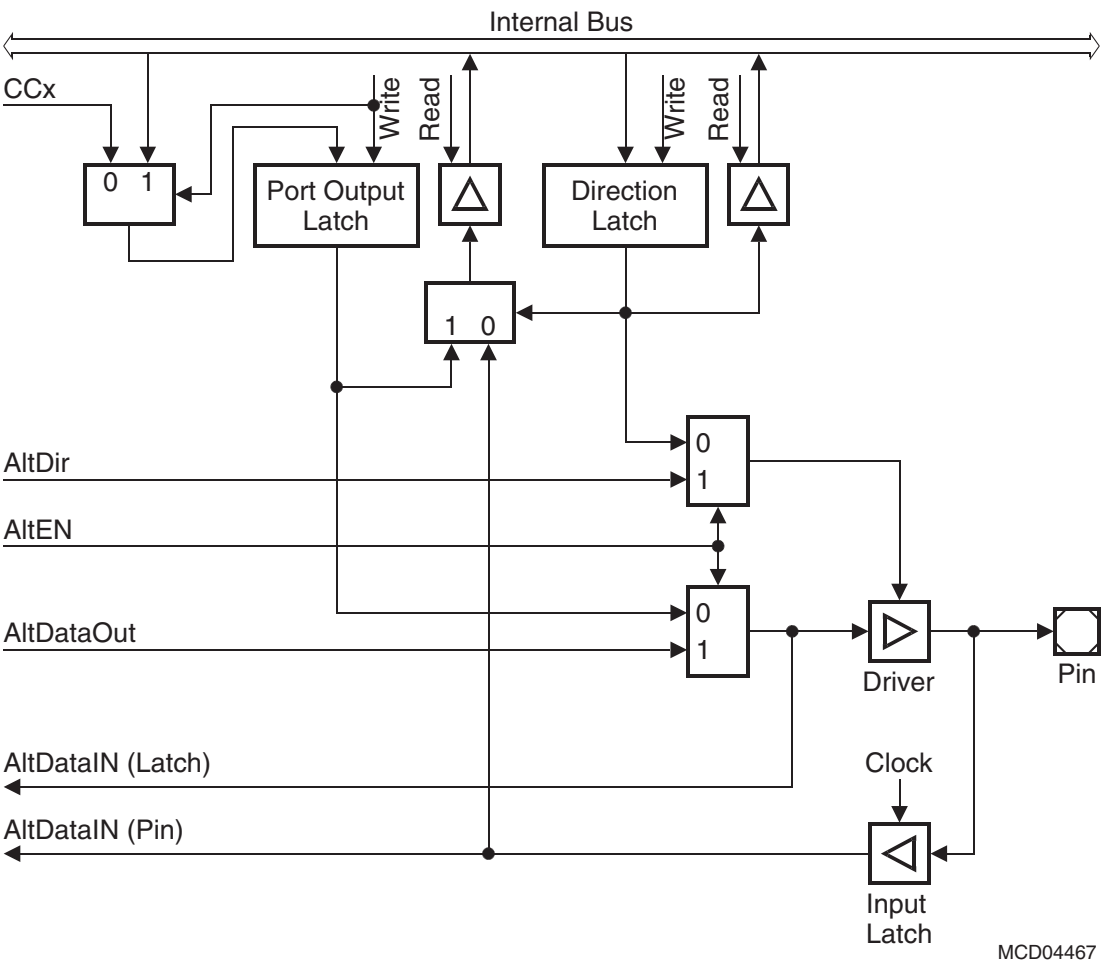


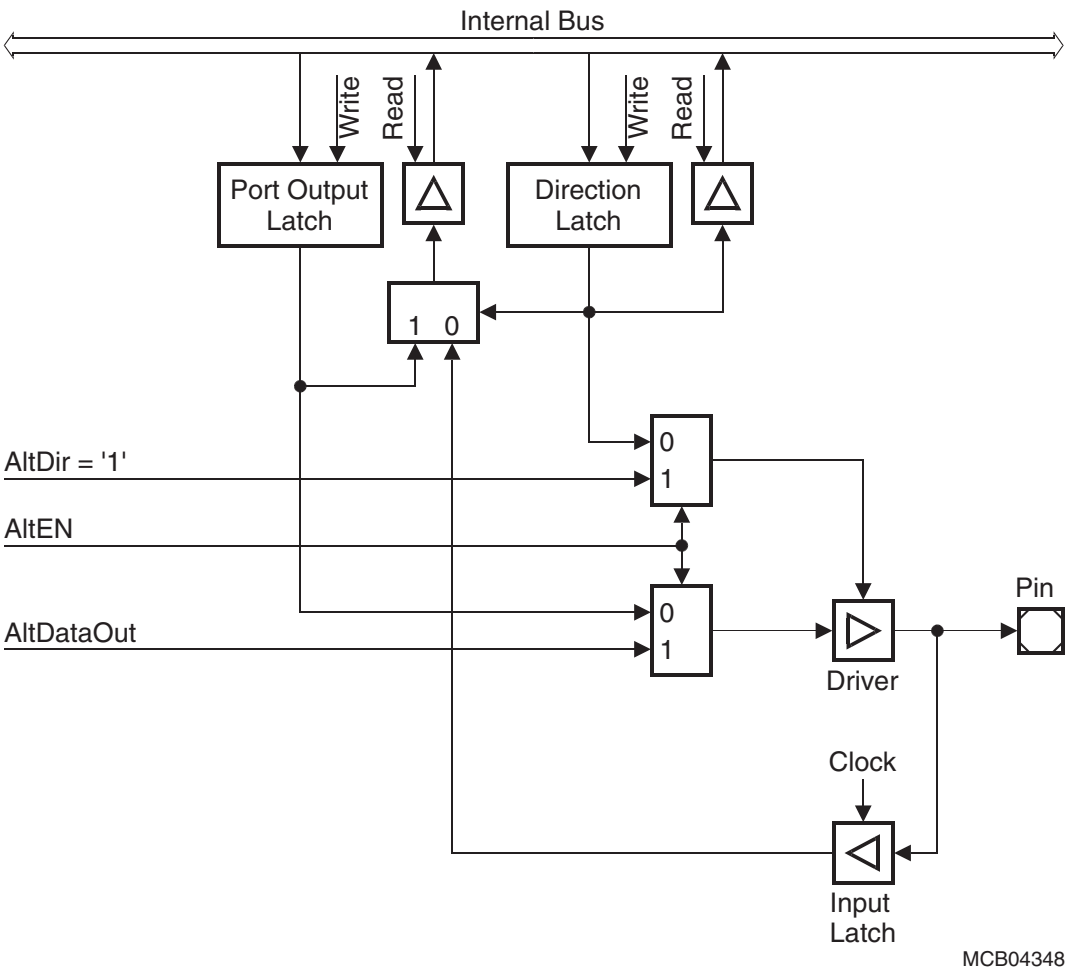
Figure 7-7 PORT1 IO and Alternate Functions

The figures below show the structure of PORT1 pins. The upper 4 pins of PORT1 combine internal bus data and alternate data output before the port latch input.



P1H.7-4, x = 27-24

Figure 7-8 Block Diagram of a PORT1 Pin with Address and CAPCOM Function



P1H.3-0, P1L.7-0

Figure 7-9 Block Diagram of a PORT1 Pin with Address and Alternate Input/Output Function

7.6 Port 3

If this 9-bit port is used for general purpose IO, the direction of each line can be configured via the corresponding direction register DP3. Most port lines can be switched to push/pull or open drain mode via the open drain control register ODP3 (pins P3.15 and P3.12 do not support open drain mode!).

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
P3 .15	**-**	**P3 .13**	**P3 .12**	**P3 .11**	**P3 .10**	**P3 .9**	**P3 .8**	**-**	**P3.6**	**-**	**P3.4**	**-**	**-**	**-**	**-**
rw	-	rw	rw	rw	rw	rw	rw	-	rw	-	rw	-	-	-	-

Bit	Function
P3.y	**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
DP3 .15	**-**	**DP3 .13**	**DP3 .12**	**DP3 .11**	**DP3 .10**	**DP3 .9**	**DP3 .8**	**-**	**DP3 .6**	**-**	**DP3 .4**	**-**	**-**	**-**	**-**
rw	-	rw	rw	rw	rw	rw	rw	-	rw	-	rw	-	-	-	-

Bit	Function
DP3.y	**Port direction register DP3 bit y** DP3.y = 0: Port line P3.y is an input (high-impedance) DP3.y = 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
-	-	ODP 3.13	-	ODP 3.11	ODP 3.10	ODP 3.9	ODP 3.8	-	ODP 3.6	-	ODP 3.4	-	-	-	-
-	-	rw	-	rw	rw	rw	rw	-	rw	-	rw	-	-	-	-

Bit	Function
ODP3.y	**Port 3 Open Drain control register bit y** ODP3.y = 0: Port line P3.y output driver in push/pull mode ODP3.y = 1: Port line P3.y output driver in open drain mode

Note: Due to pin limitations register bit P3.14 is not connected to an IO pin.
Pins P3.15 and P3.12 do not support open drain mode.

Alternate Functions of Port 3

The pins of Port 3 serve various functions including external timer control lines, the two serial interfaces, and the control lines $\overline{BHE}/\overline{WRH}$ and CLKOUT/FOUT.

Table 7-3 summarizes the alternate functions of Port 3.

Table 7-3 Alternate Functions of Port 3

Port 3 Pin	Alternate Function	
–	–	
P3.4	T3EUD	Timer 3 External Up/Down Input
–	–	
P3.6	T3IN	Timer 3 Count Input
–	–	
P3.8	MRST	SSC Master Receive / Slave Transmit
P3.9	MTSR	SSC Master Transmit / Slave Receive
P3.10	TxD0	ASC0 Transmit Data Output
P3.11	RxD0	ASC0 Receive Data Input
P3.12	$\overline{BHE}/\overline{WRH}$	Byte High Enable / Write High Output
P3.13	SCLK	SSC Shift Clock Input/Output
–	–	
P3.15	CLKOUT/ FOUT	System Clock Output/ Programmable Frequency Output

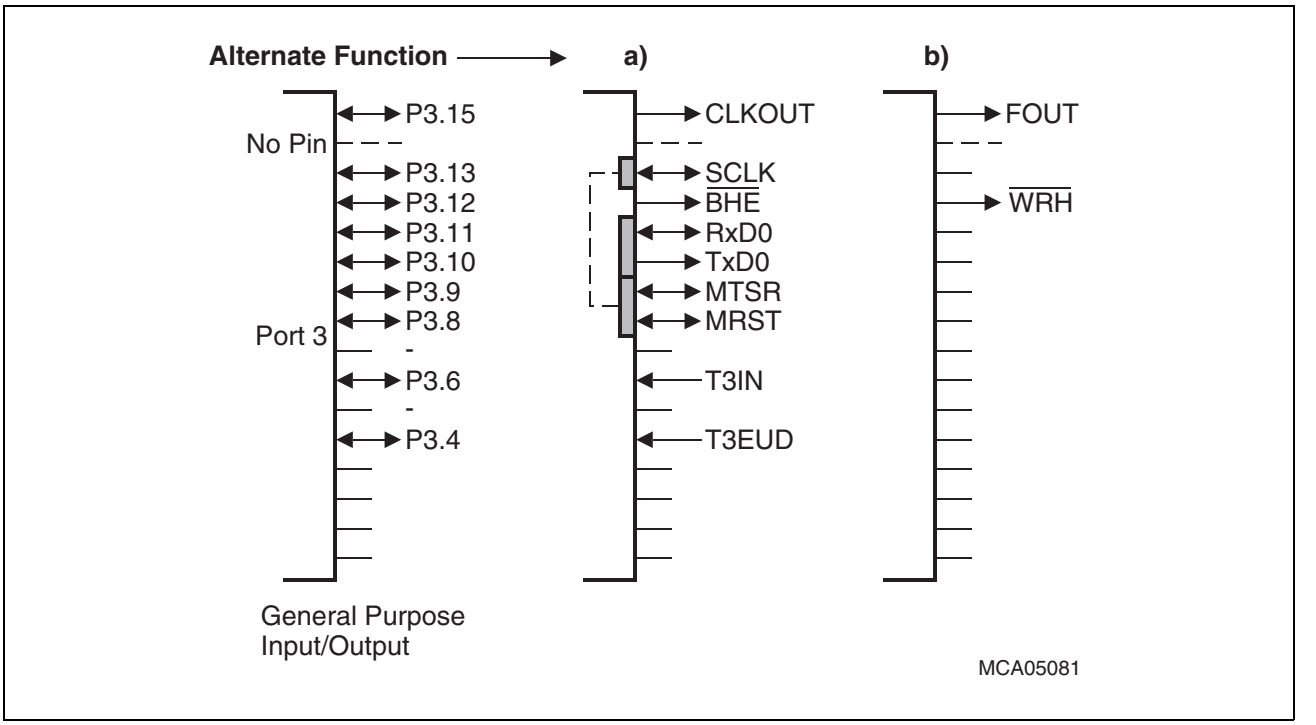


Figure 7-10 Port 3 IO and Alternate Functions

The port structure of Port 3 pins depends on their alternate function (see **Figure 7-11** and **Figure 7-12**).

When the on-chip peripheral associated with a Port 3 pin is configured to use the alternate input function, it reads the input latch representing the state of the pin via the line labeled “Alternate Data Input”. Port 3 pins with alternate input functions are:
T3IN and T3EUD.

When the on-chip peripheral associated with a Port 3 pin is configured to use the alternate output function, its “Alternate Data Output” line is ANDed with the port output latch line. When using these alternate functions, the user must set the direction of the port line to output (DP3.y = ‘1’) and must set the port output latch (P3.y = ‘1’). Otherwise, the pin is in its high-impedance state (when configured as input) or the pin is stuck at ‘0’ (when the port output latch is cleared). When the alternate output functions are not used, the “Alternate Data Output” line is in its inactive state, which is a high level (‘1’).
Port 3 pins with alternate output functions are:
TxD0 and CLKOUT/FOUT.

When the on-chip peripheral associated with a Port 3 pin is configured to use both the alternate input and output function, the descriptions above apply to the respective current operating mode. The direction must be set accordingly.
Port 3 pins with alternate input/output functions are:
MTSR, MRST, RxD0, and SCLK.

Note: Enabling the CLKOUT function automatically enables the P3.15 output driver. Setting bit DP3.15 = '1' is not required.
The CLKOUT function is automatically enabled in emulation mode.

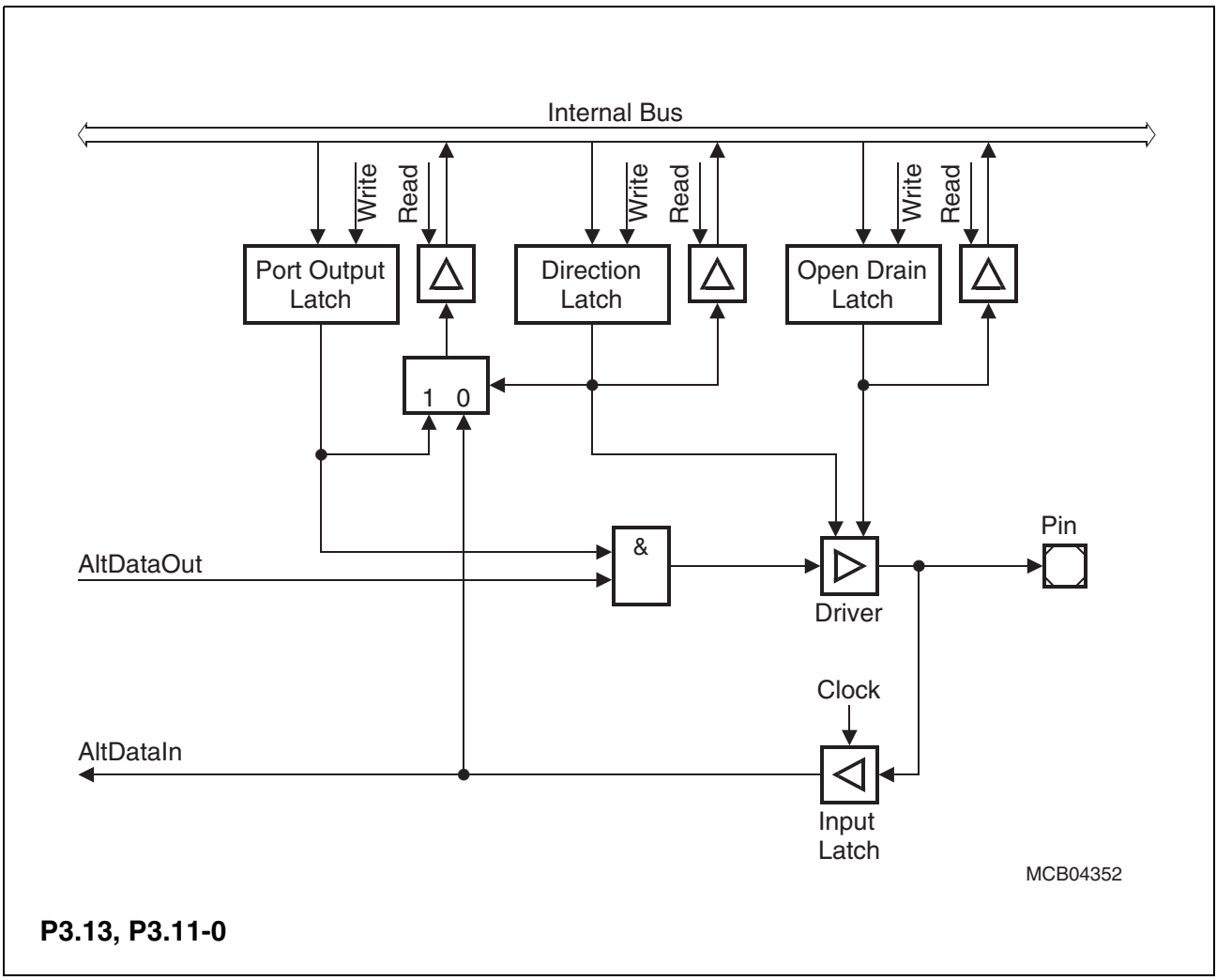
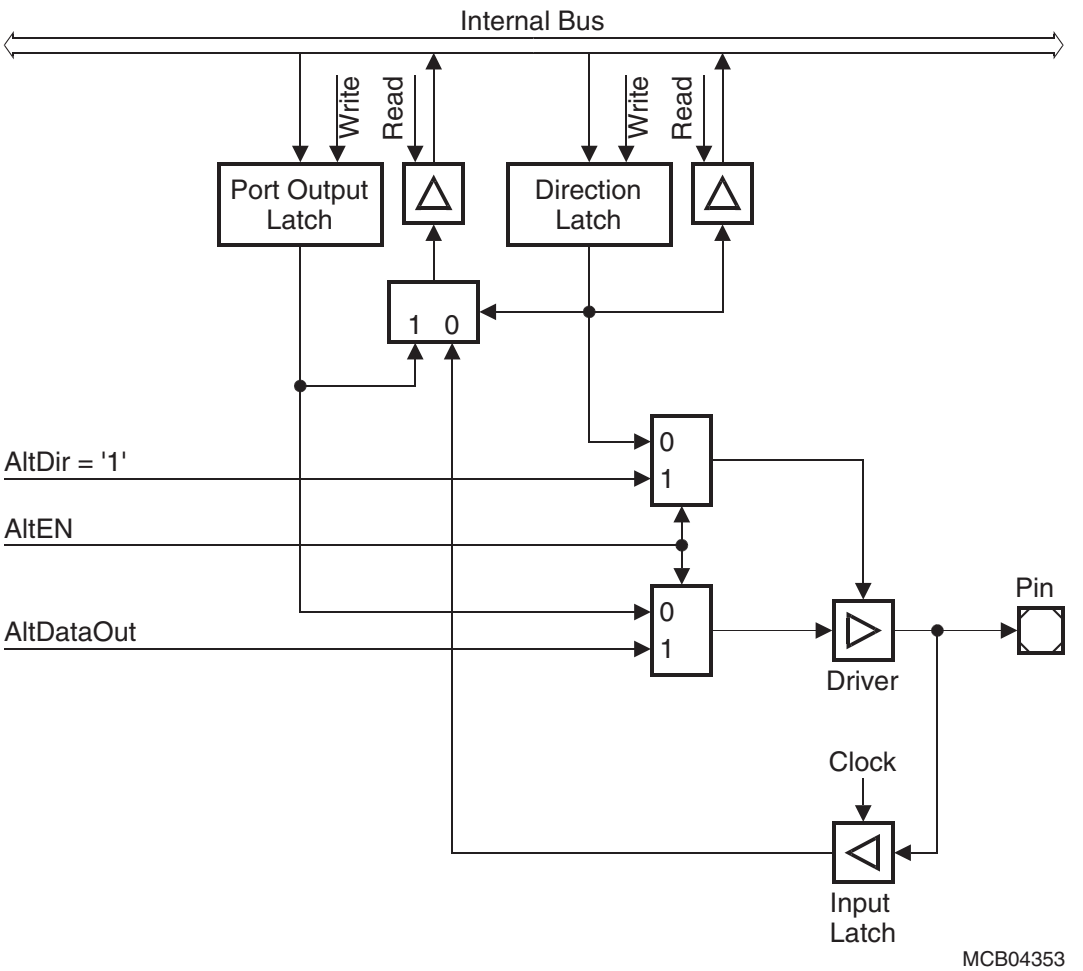


Figure 7-11 Block Diagram of a Port 3 Pin with Alternate Input or Alternate Output Function

Pin P3.12 ($\overline{BHE}$/$\overline{WRH}$) also has an alternate output function. However, its structure is slightly different (see **Figure 7-12**), because either the $\overline{BHE}$ or the $\overline{WRH}$ function must be used after reset depending on the system startup configuration. In these cases, there is no possibility to program any port latches previously. Thus, the appropriate alternate function is selected automatically. If $\overline{BHE}$/$\overline{WRH}$ is not used in the system, this pin can be used for general purpose IO by disabling the alternate function (BYTDIS = '1' / WRCFG = '0').



P3.15, P3.12

Figure 7-12 Block Diagram of Pins P3.15 (CLKOUT/FOUT) and P3.12 ($\overline{BHE}$/$\overline{WRH}$)

Note: Enabling the $\overline{BHE}$ or $\overline{WRH}$ function automatically enables the P3.12 output driver. Setting bit DP3.12 = '1' is not required.
Enabling the CLKOUT function automatically enables the P3.15 output driver. Setting bit DP3.15 = '1' is not required.

7.7 Port 4

If this 6-bit port is used for general purpose IO, the direction of each line can be configured via the corresponding direction register DP4.

P4
Port 4 Data Register **SFR ($FFC8_H/E4_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								-	P4.6	P4.5	-	P4.3	P4.2	P4.1	P4.0
-	-	-	-	-	-	-	-	-	rw	rw	-	rw	rw	rw	rw

Bit	Function
P4.y	Port data register P4 bit y

DP4
P4 Direction Ctrl. Register **SFR ($FFCA_H/E5_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								-	DP4 .6	DP4 .5	-	DP4 .3	DP4 .2	DP4 .1	DP4 .0
-	-	-	-	-	-	-	-	-	rw	rw	-	rw	rw	rw	rw

Bit	Function
DP4.y	**Port direction register DP4 bit y** DP4.y = 0: Port line P4.y is an input (high-impedance) DP4.y = 1: Port line P4.y is an output

ODP4
P4 Open Drain Ctrl. Reg. **ESFR ($F1CA_H/E5_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								-	ODP 4.6	ODP 4.5	-	ODP 4.3	ODP 4.2	ODP 4.1	ODP 4.0
-	-	-	-	-	-	-	-	-	rw	rw	-	rw	rw	rw	rw

Bit	Function
ODP4.y	**Port 4 Open Drain control register bit y** ODP4.y = 0: Port line P4.y output driver in push/pull mode ODP4.y = 1: Port line P4.y output driver in open drain mode

Alternate Functions of Port 4

During external bus cycles which use segmentation (that is, an address space above 64 KByte) a number of Port 4 pins may output the segment address lines. The number of pins used for the segment address output determines the external address space which is directly accessible.

Optionally, some Port 4 pins can output chip select signals directly selecting external modules such as memories or peripherals.

The other pins of Port 4 (if any) may be used for general purpose IO or for the CAN interface.

If segment address lines are selected, the alternate function of Port 4 may be necessary to access resources such as external memory directly after reset. For this reason, Port 4 will be switched to this alternate function automatically.

The number of segment address lines or chip select lines is selected via bitfield SALSEL or CSSEL in register RP0H. During an external reset register RP0H is configured according to the levels of PORT0. Software can adjust the number of selected segment address lines or chip select lines via register RSTCON.

The CAN interface can use 2 pins of Port 4 to interface the CAN module to an external CAN transceiver. In this case, the number of possible segment address lines is reduced.

Note: Port 4 pins which are not used for segment address output or for chip select output or for the CAN interface may be used for general purpose IO. The pins which are used for chip select output are defined via bitfield CSSEL (see register RP0H).
If more than one function is selected for a Port 4 pin, the segment address takes preference over the chip select lines, and the CAN interface takes preference over the segment address lines.

Table 7-4 summarizes the alternate functions of Port 4 depending on the number of selected segment address lines (coded via bitfield SALSEL).

Table 7-4 Alternate Functions of Port 4

Port 4 Pin	Std. Function SALSEL = 01 64 KB	Altern. Function SALSEL = 11 256 KB	Altern. Function SALSEL = 00 1 MB	Altern. Function SALSEL = 10 4 MB
P4.0	Gen. p. IO or $\overline{CS3}$	Seg. Addr. A16	Seg. Addr. A16	Seg. Addr. A16
P4.1	Gen. p. IO or $\overline{CS2}$	Seg. Addr. A17	Seg. Addr. A17	Seg. Addr. A17
P4.2	Gen. p. IO or $\overline{CS1}$	Gen. p. IO or $\overline{CS1}$	Seg. Addr. A18	Seg. Addr. A18
P4.3	Gen. p. IO or $\overline{CS0}$	Gen. p. IO or $\overline{CS0}$	Seg. Addr. A19	Seg. Addr. A19
–	–	–	–	–
P4.5	Gen. p. IO or CAN	Gen. p. IO or CAN	Gen. p. IO or CAN	S.A. A20 or CAN
P4.6	Gen. p. IO or CAN	Gen. p. IO or CAN	Gen. p. IO or CAN	S.A. A21 or CAN
–	–	–	–	–

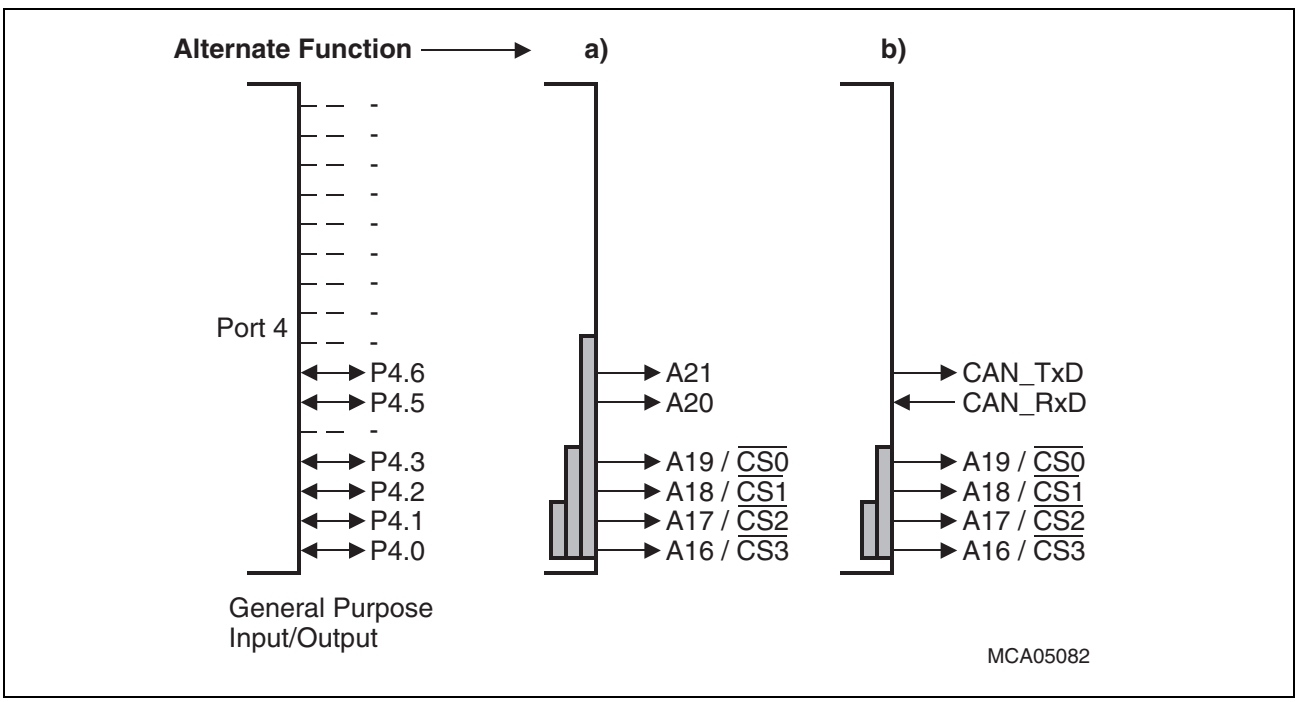


Figure 7-13 Port 4 IO and Alternate Functions

Note: The usage of Port 4 pins for CAN interface lines depends on the chosen assignments for the CAN module.
CAN interface lines will override general purpose IO and segment address lines.

The chip select lines of Port 4 additionally have an internal weak pull-up device which is switched on during any reset (including single-chip mode reset) in order to provide an inactive level on the optional chip select lines until the controller begins operation.

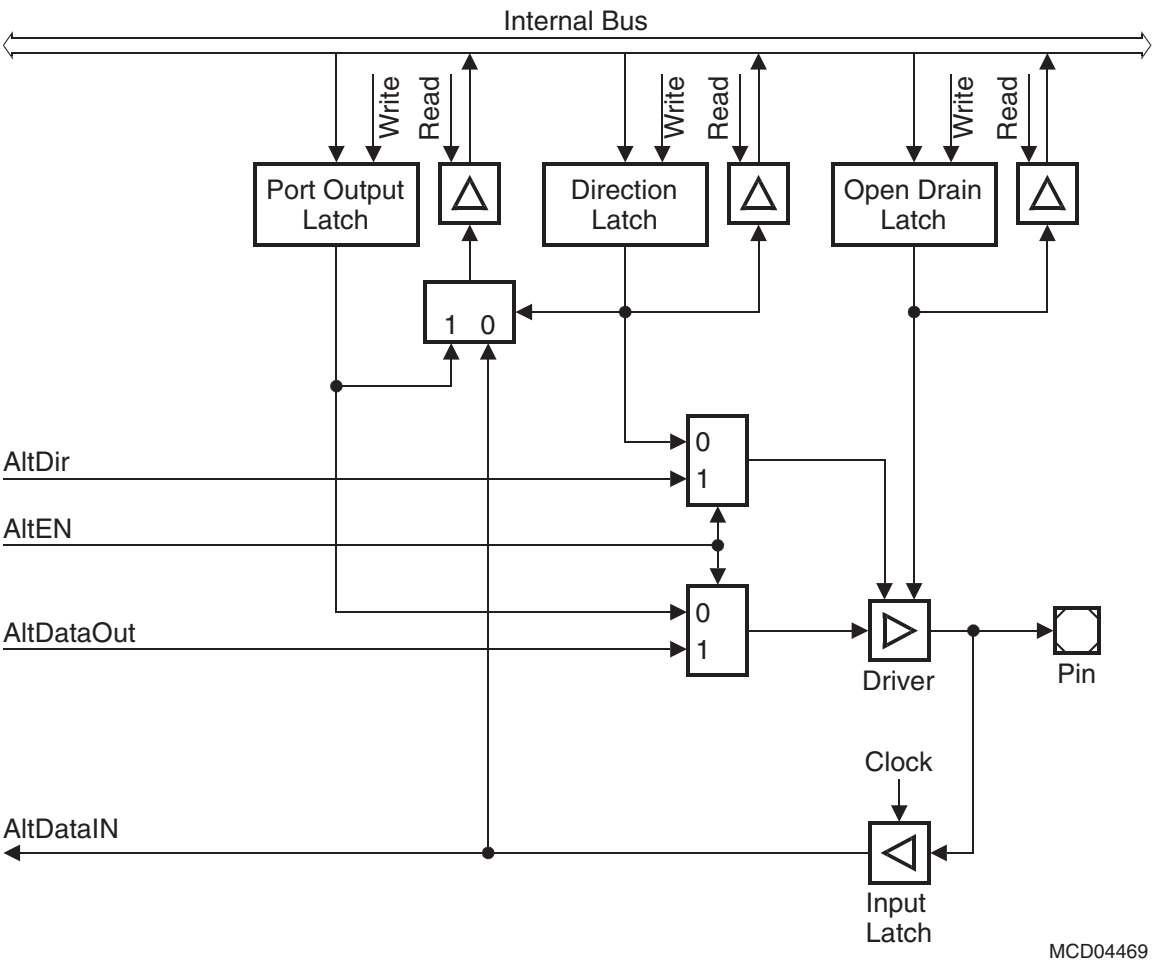


Figure 7-14 Block Diagram of a Port 4 Pin

7.8 Port 5

This 8-bit input port can only read data. There is no output latch or 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
-	-	-	-	-	-	-	-	P5.7	P5.6	P5.5	P5.4	P5.3	P5.2	P5.1	P5.0
-	-	-	-	-	-	-	-	r	r	r	r	r	r	r	r

Bit	Function
P5.y	**Port data register P5 bit y** (Read only)

Alternate Functions of Port 5

Each line of Port 5 is also connected to the input multiplexer of the Analog/Digital Converter. All port lines can accept analog signals (ANx) which can be converted by the ADC. For pins to be used as analog inputs it is recommended to disable the digital input stage via register P5DIDIS (see description below). This avoids undesired cross currents and switching noise while the (analog) input signal level is between V_{IL} and V_{IH}. Some pins of Port 5 also serve as external GPT timer control lines.

Table 7-5 summarizes the alternate functions of Port 5.

Table 7-5 Alternate Functions of Port 5

Port 5 Pin	Alternate Function a)	Alternate Function b)
P5.0	Analog Input AN0	–
P5.1	Analog Input AN1	–
P5.2	Analog Input AN2	–
P5.3	Analog Input AN3	–
P5.4	Analog Input AN4	T2EUD Timer 2 ext. Up/Down Input
P5.5	Analog Input AN5	T4EUD Timer 4 ext. Up/Down Input
P5.6	Analog Input AN6	T2IN Timer 2 Count Input
P5.7	Analog Input AN7	T4IN Timer 4 Count Input

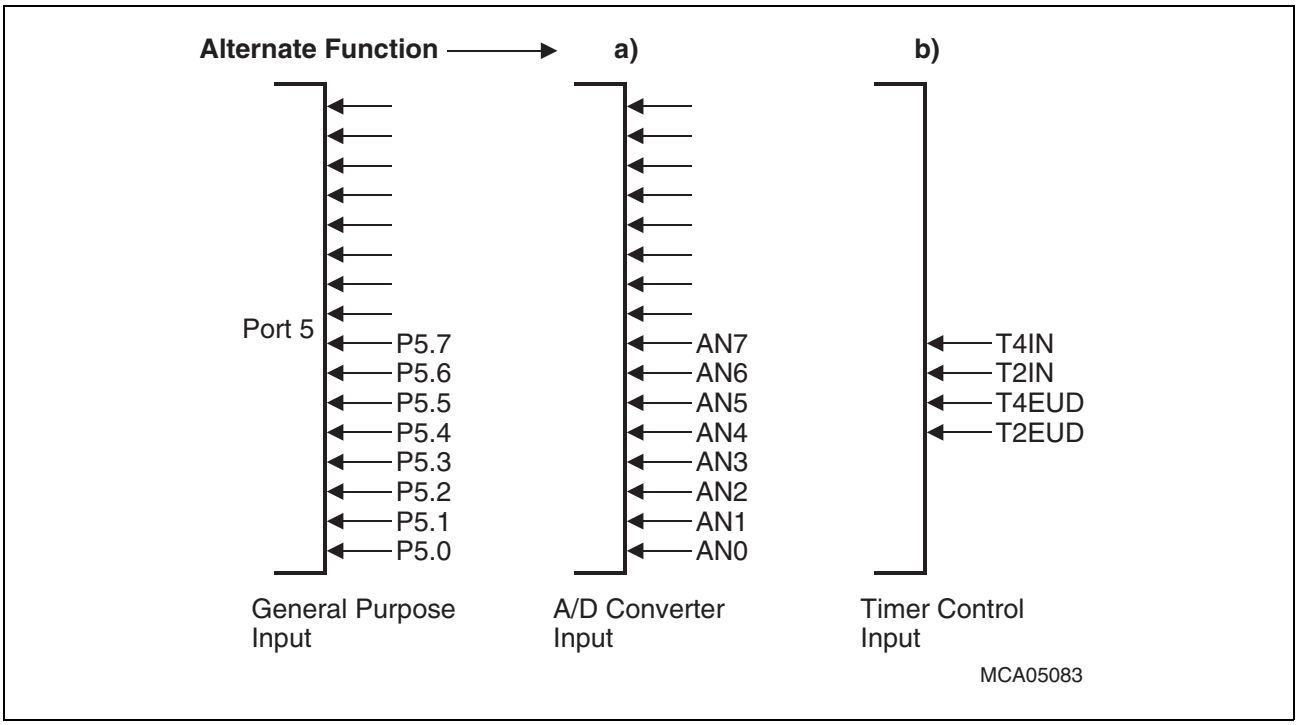


Figure 7-15 Port 5 IO and Alternate Functions

Port 5 Digital Input Control

Port 5 pins may be used for either digital or analog input. By setting the respective bit in register P5DIDIS, the digital input stage of the respective Port 5 pin can be disconnected from the pin. This is recommended when the pin is to be used as analog input as it reduces the current through the digital input stage and prevents it from toggling while the (analog) input level is between the digital low and high thresholds. Thus, the consumed power and the generated noise can be reduced.

After reset all digital input stages are enabled.

P5DIDIS
P5 Dig. 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
-	-	-	-	-	-	-	-	P5D.7	P5D.6	P5D.5	P5D.4	P5D.3	P5D.2	P5D.1	P5D.0
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
P5D.y	**Port P5 Bit y Digital Input Control** 0: Digital input stage connected to port line P5.y 1: Digital input stage disconnected from port line P5.y When being read or used as alternate input this line appears as ‘1’.

Port 5 pins have a special port structure (see **Figure 7-16**) for two reasons: First, because it is an input only port; second, because the analog input channels are connected directly to the pins rather than to the input latches.

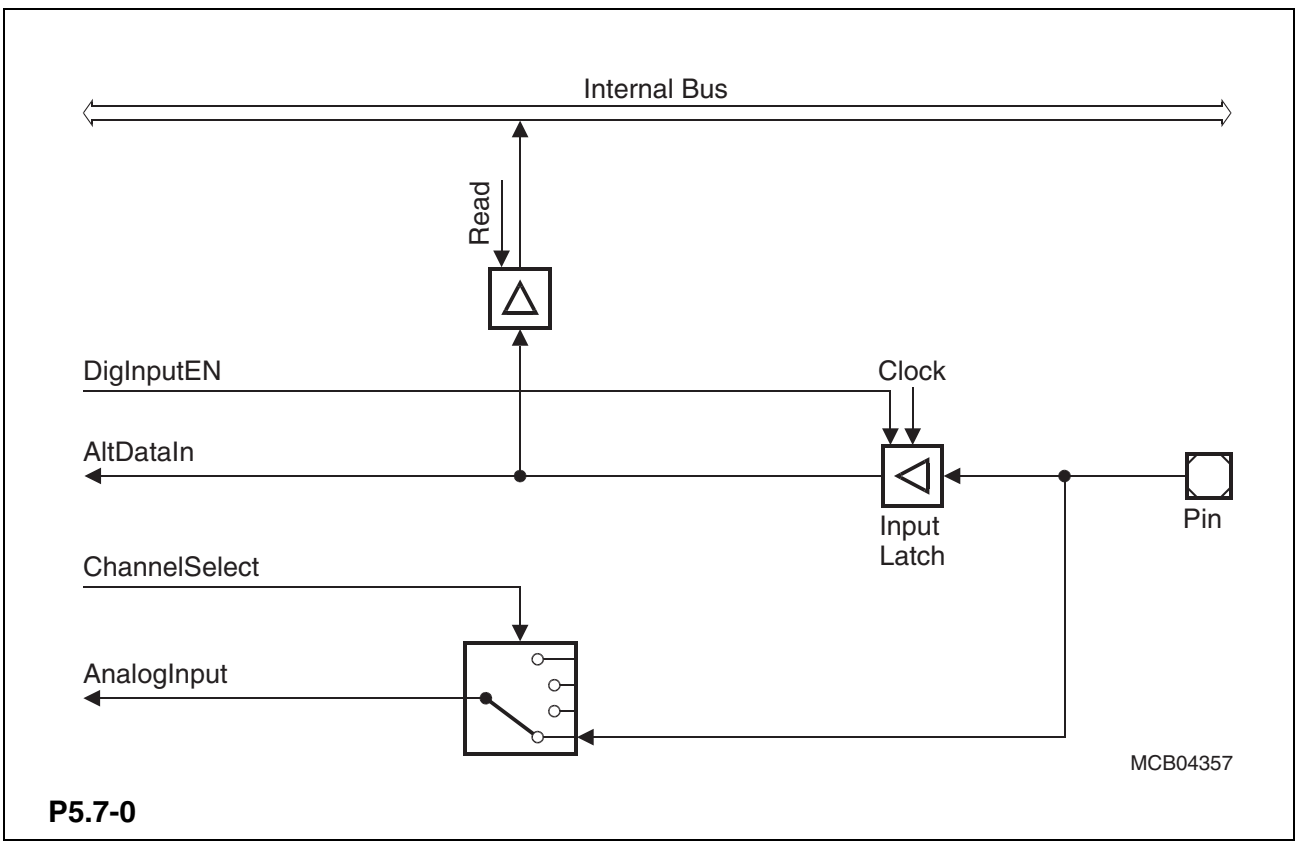


Figure 7-16 Block Diagram of a Port 5 Pin

Note: The “AltDataIn” line does not exist on all Port 5 inputs.

7.9 Port 8

If this 4-bit port is used for general purpose IO, the direction of each line can be configured via the corresponding direction register DP8. Each port line can be switched into push/pull or open drain mode via the open drain control register ODP8.

P8
Port 8 Data Register **SFR ($FFD4_H/EA_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								-	-	-	-	P8.3	P8.2	P8.1	P8.0
-	-	-	-	-	-	-	-	-	-	-	-	rwh	rwh	rwh	rwh

Bit	Function
P8.y	Port data register P8 bit y

DP8
P8 Direction Ctrl. Register **SFR ($FFD6_H/EB_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								-	-	-	-	DP8 .3	DP8 .2	DP8 .1	DP8 .0
-	-	-	-	-	-	-	-	-	-	-	-	rw	rw	rw	rw

Bit	Function
DP8.y	**Port direction register DP8 bit y** DP8.y = 0: Port line P8.y is an input (high-impedance) DP8.y = 1: Port line P8.y is an output

ODP8
P8 Open Drain Ctrl. Reg. **ESFR ($F1D6_H$/EB_H)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								-	-	-	-	ODP8.3	ODP8.2	ODP8.1	ODP8.0
-	-	-	-	-	-	-	-	-	-	-	-	rw	rw	rw	rw

Bit	Function
ODP8.y	**Port 8 Open Drain control register bit y** ODP8.y = 0: Port line P8.y output driver in push/pull mode ODP8.y = 1: Port line P8.y output driver in open drain mode

Alternate Functions of Port 8

All Port 8 lines serve as capture inputs or compare outputs (CCxIO) for the CAPCOM2 unit (see **Table 7-6**).

When a Port 8 line is used as a capture input, the state of the input latch, representing the state of the port pin, is directed to the CAPCOM unit via the line “Alternate Pin Data Input”. If an external capture trigger signal is used, the direction of the respective pin must be set to input. If the direction is set to output, the state of the port output latch will be read, because the pin represents the state of the output latch. This can be used to trigger a capture event through software by setting or clearing the port latch. Note that in the output configuration, no external device may drive the pin, otherwise conflicts would occur.

When a Port 8 line is used as a compare output (compare modes 1 and 3), the compare event (or the timer overflow in compare mode 3) directly affects the port output latch. In compare mode 1, when a valid compare match occurs, the state of the port output latch is read by the CAPCOM control hardware via the line “Alternate Latch Data Input”, is inverted, and then written back to the latch via the line “Alternate Data Output”. The port output latch is clocked by the signal “Compare Trigger” which is generated by the CAPCOM unit. In compare mode 3, when a match occurs, the value ‘1’ is written to the port output latch via the line “Alternate Data Output”. When an overflow of the corresponding timer occurs, a ‘0’ is written to the port output latch. In both cases, the output latch is clocked by the signal “Compare Trigger”. The direction of the pin should be set to output by the user; otherwise, the pin will be in the high-impedance state and will not reflect the state of the output latch.

As can be seen from the port structure below, the user software always has free access to the port pin even when it is used as a compare output. This is useful for setting up the initial level of the pin when using compare mode 1 or the double-register mode. In these modes, unlike in compare mode 3, the pin is not set to a specific value when a compare match occurs, but is toggled instead.

If the user wants to write to the port pin at the same time that a compare trigger tries to clock the output latch, the write operation of the user software has priority. Each time a CPU write access to the port output latch occurs, the input multiplexer of the port output latch is switched to the line connected to the internal bus. The port output latch will receive the value from the internal bus and the hardware triggered change will be lost.

As with all other capture inputs, the capture input function of the Port 8 pins can also be used as external interrupt input (sample rate 16 TCL).

The CAN interface can use 2 pins of Port 8 to interface the CAN Module to an external transceiver. In this case, the number of possible CAPCOM IO lines is reduced.

Table 7-6 Alternate Functions of Port 8

Port 8 Pin	Alternate Function	
P8.0	CC16IO	Capture input / compare output channel 16 or CAN
P8.1	CC17IO	Capture input / compare output channel 17 or CAN
P8.2	CC18IO	Capture input / compare output channel 18 or CAN
P8.3	CC19IO	Capture input / compare output channel 19 or CAN

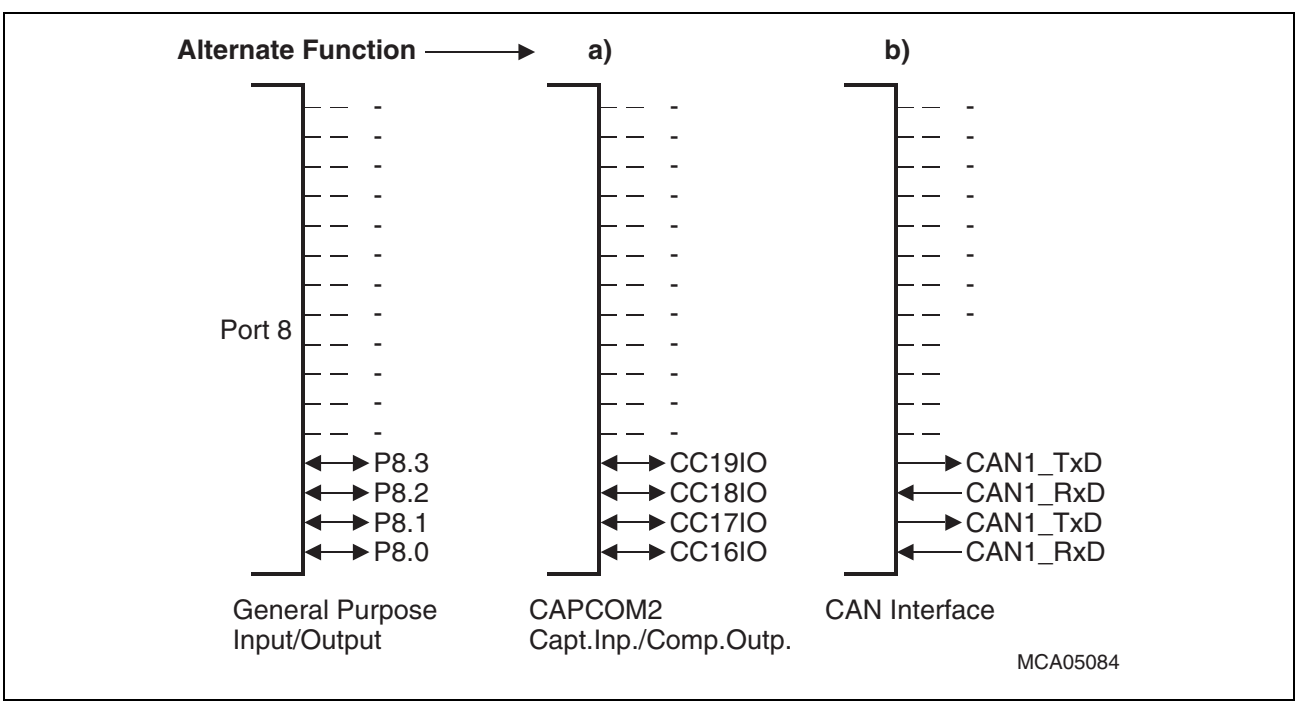


Figure 7-17 Port 8 IO and Alternate Functions

Note: The usage of Port 8 pins for CAN interface lines depends on the chosen assignment for the CAN module.
CAN interface lines will override general purpose IO and CAPCOM IO lines.

The pins of Port 8 combine internal bus data and alternate data output before the port latch input.

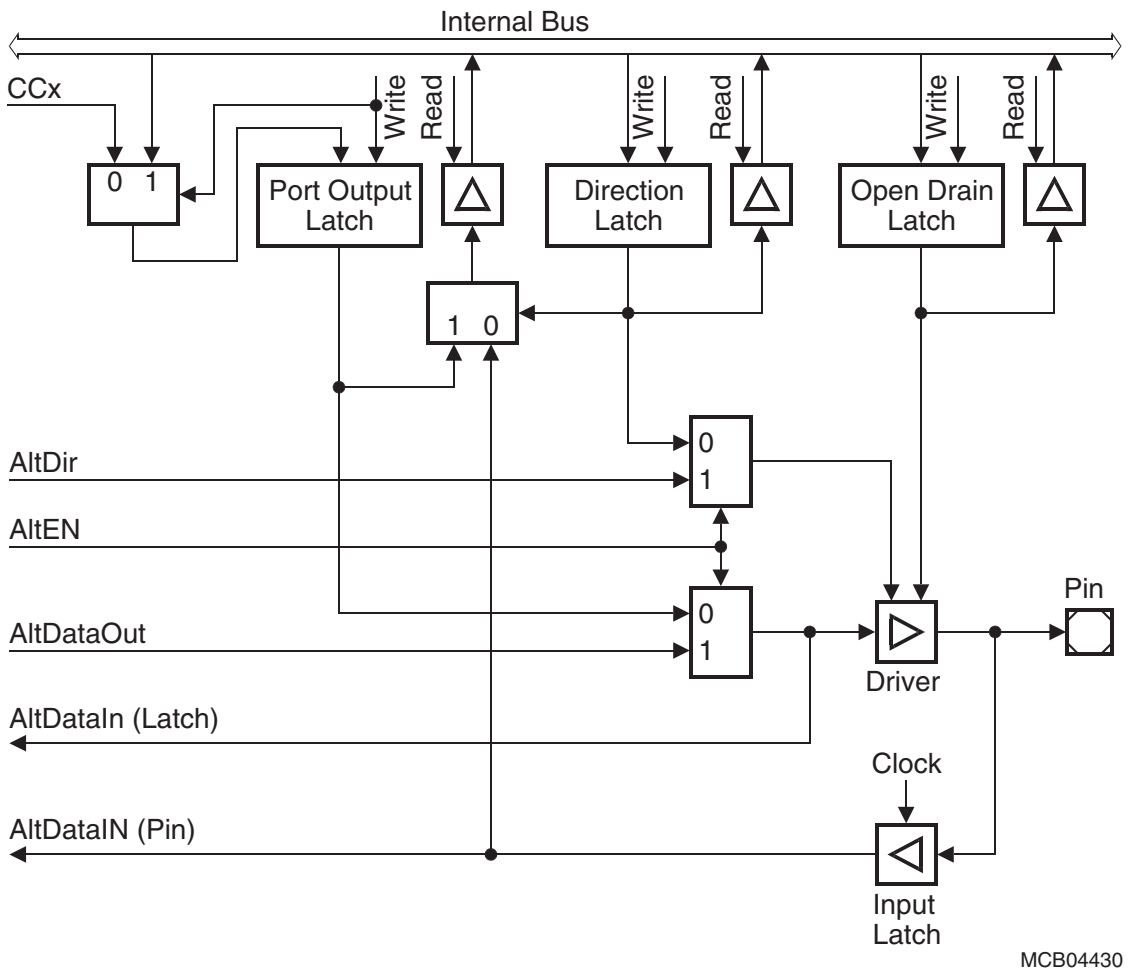


Figure 7-18 Block Diagram of Port 8 Pins with an Alternate CAPCOM IO and CAN Interface Function

8 Dedicated Pins

Most of the input/output or control signals of the C164CI are implemented as alternate functions of the parallel ports pins. There are, however, a number of signals which use separate pins, including the oscillator, special control signals, and the power supply.

Table 8-1 summarizes the 21 dedicated pins of the C164CI.

Table 8-1 C164CI Dedicated Pins

Pin(s)	Function
ALE	Address Latch Enable
$\overline{RD}$	External Read Strobe
$\overline{WR}$/$\overline{WRL}$	External Write/Write Low Strobe
$\overline{EA}$/V_{PP}	External Access Enable and External Programming Voltage
$\overline{NMI}$	Non-Maskable Interrupt Input
XTAL1, XTAL2	Oscillator Input/Output
$\overline{RSTIN}$	Reset Input
$\overline{RSTOUT}$	Reset Output
V_{AREF}, V_{AGND}	Voltage Reference for Analog/Digital Converter
V_{DD}	Digital Power Supply (5 pins)
V_{SS}	Digital Reference Ground (5 pins)

The Address Latch Enable signal ALE controls external address latches that provide a stable address in multiplexed bus modes.
ALE is activated for every external bus cycle independent of the selected bus mode, that is, it is also activated for bus cycles with a demultiplexed address bus. When an external bus is enabled (one or more of the BUSACT bits set), X-Peripheral accesses will also generate an active ALE signal.
ALE is not activated for internal accesses, that is, for accesses to ROM/OTP/Flash (if provided), the internal RAM, and the special function registers. In single chip mode, when no external bus is enabled (no BUSACT bit set), ALE will also remain inactive for X-Peripheral accesses.
During reset, an internal pull-down resistor ensures an inactive (low) level on the ALE output.

At the end of a true single-chip mode reset ($\overline{EA}$ = ‘1’) the current level on pin ALE is latched and is used for configuration (together with pin $\overline{RD}$). Pin ALE selects standard start/boot, when driven low (default) or selects alternate start/boot when driven high.

For standard configuration, pin ALE should be low or not connected.

The External Read Strobe $\overline{RD}$ controls the output drivers of external memory or peripherals when the C164CI reads data from these external devices. During accesses to on-chip X-Peripherals, $\overline{RD}$ remains inactive (high).
During reset, an internal pull-up resistor ensures an inactive (high) level on the $\overline{RD}$ output.

At the end of reset, the current level on pin $\overline{RD}$ is latched and is used for configuration.

For a reset with external access ($\overline{EA}$ = ‘0’), pin $\overline{RD}$ controls the oscillator watchdog. The latched $\overline{RD}$ level determines the reset value of bit OWDDIS in register SYSCON. The default high level on pin $\overline{RD}$ leaves the oscillator watchdog active (OWDDIS = ‘0’), while a low level disables the watchdog (OWDDIS = ‘1’) e.g. for testing purposes.

For a true single-chip mode, reset ($\overline{EA}$ = ‘1’) pin $\overline{RD}$ enables the bootstrap loader when driven low (pin ALE is evaluated together with pin $\overline{RD}$).

For standard configuration pin $\overline{RD}$ should be high or not connected.

The External Write Strobe $\overline{WR}$/$\overline{WRL}$ controls the data transfer from the C164CI to an external memory or peripheral device. This pin may either provide an general $\overline{WR}$ signal activated for both byte and word write accesses, or it may specifically control the low byte of an external 16-bit device ($\overline{WRL}$) together with the signal $\overline{WRH}$ (alternate function of P3.12/$\overline{BHE}$). During accesses to on-chip X-Peripherals, $\overline{WR}$/$\overline{WRL}$ remains inactive (high).
During reset, an internal pull-up resistor ensures an inactive (high) level on the $\overline{WR}$/$\overline{WRL}$ output.

The External Access Enable Pin $\overline{EA}$/V_{PP} determines whether, after reset, the C164CI starts fetching code from the internal ROM area ($\overline{EA}$ = ‘1’) or via the external bus interface ($\overline{EA}$ = ‘0’). Be sure to hold this input low for ROMless devices. At the end of the internal reset sequence, the $\overline{EA}$ signal is latched together with the configuration (PORT0, $\overline{RD}$, ALE).

When used as programming voltage input V_{PP}, this pin accepts the external programming voltage required to program the on-chip OTP program memory.

Note: This feature is only available in OTP-devices, of course.

The Non-Maskable Interrupt Input $\overline{NMI}$ allows triggering of a high priority trap via an external signal (e.g. a power-fail signal). It also serves to validate the PWRDN instruction which switches the C164CI into Power-Down mode. The $\overline{NMI}$ pin is sampled with every CPU 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 **Chapter 6**) consists of the crystal, two low end capacitors, and a 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 C164CI.

An external clock signal may be fed to the input XTAL1, leaving XTAL2 open or terminating it for higher input frequencies.

The Reset Input $\overline{\text{RSTIN}}$ allows the C164CI to be put into the well-defined reset condition either at power-up or on external events such as a hardware failure or manual reset. The input voltage threshold of the $\overline{\text{RSTIN}}$ pin is raised compared to the standard pins to minimize the noise sensitivity of the reset input.

In bidirectional reset mode, the C164CI's line $\overline{\text{RSTIN}}$ may be driven active by the chip logic in order to support external equipment which is required for startup (e.g. flash memory).

Bidirectional reset reflects internal reset sources (software, watchdog) to the $\overline{\text{RSTIN}}$ pin and converts short hardware reset pulses to a minimum duration of the internal reset sequence. Bidirectional reset is enabled by setting bit BDRSTEN in register SYSCON and changes $\overline{\text{RSTIN}}$ from a pure input to an open drain IO line. When an internal reset is triggered by the SRST instruction, or by a watchdog timer overflow, or by application of a low level to the $\overline{\text{RSTIN}}$ line, an internal driver pulls it low for the duration of the internal reset sequence. It is released afterwards and is then controlled by the external circuitry alone.

The bidirectional reset function is useful for applications in which external devices require a defined reset signal but cannot be connected to the C164CI's $\overline{\text{RSTOUT}}$ signal, e.g. an external flash memory which must come out of reset and deliver code well before $\overline{\text{RSTOUT}}$ can be deactivated via EINIT.

The following behavior differences must be observed when using the bidirectional reset feature in an application:

- Bit BDRSTEN in register SYSCON cannot be changed after EINIT and is cleared automatically after a reset.
- The reset indication flags always indicate a long hardware reset.
- The PORT0 configuration is treated like on a hardware reset. In particular, the bootstrap loader may be activated when P0L.4 is low.
- Pin $\overline{\text{RSTIN}}$ may be connected only to external reset devices with an open drain output driver.
- A short hardware reset is extended to the duration of the internal reset sequence.

The Reset Output $\overline{\text{RSTOUT}}$ provides a special reset signal for external circuitry. $\overline{\text{RSTOUT}}$ is activated at the beginning of the reset sequence, triggered via $\overline{\text{RSTIN}}$, a watchdog timer overflow or by the SRST instruction. $\overline{\text{RSTOUT}}$ remains active (low) until the EINIT instruction is executed. This allows initialization of the controller before the external circuitry is activated.

Note: In emulation mode, pin $\overline{\text{RSTOUT}}$ is used as an input and, therefore, must be driven by the external circuitry.

The Reference Voltage pins for the Analog/Digital Converter V_{AREF} and V_{AGND} provide a separate power supply (reference voltage) for the comparator circuitry of the on-chip ADC. This reduces the noise which 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_{DD} and V_{SS} provide the power supply for the digital logic of the C164CI. The respective V_{DD}/V_{SS} pairs should be decoupled as close to the pins as possible. For best results, it is recommended to implement two-level decoupling, for example, the widely used 100 nF in parallel with 30 … 40 pF capacitors which deliver the peak currents.

Note: All V_{DD} pins and all V_{SS} pins must be connected to the power supply and ground, respectively.

9 External Bus Interface

Although the C164CI provides a powerful set of on-chip peripherals and on-chip RAM and ROM/OTP/Flash (except for ROMless versions) areas, these internal units cover only a small fraction of its address space of up to 16 MBytes. The External Bus Interface allows access to external peripherals and additional volatile and non-volatile memory. The External Bus Interface supports a variety of configurations so it can be tailored to fit perfectly into a given application system.

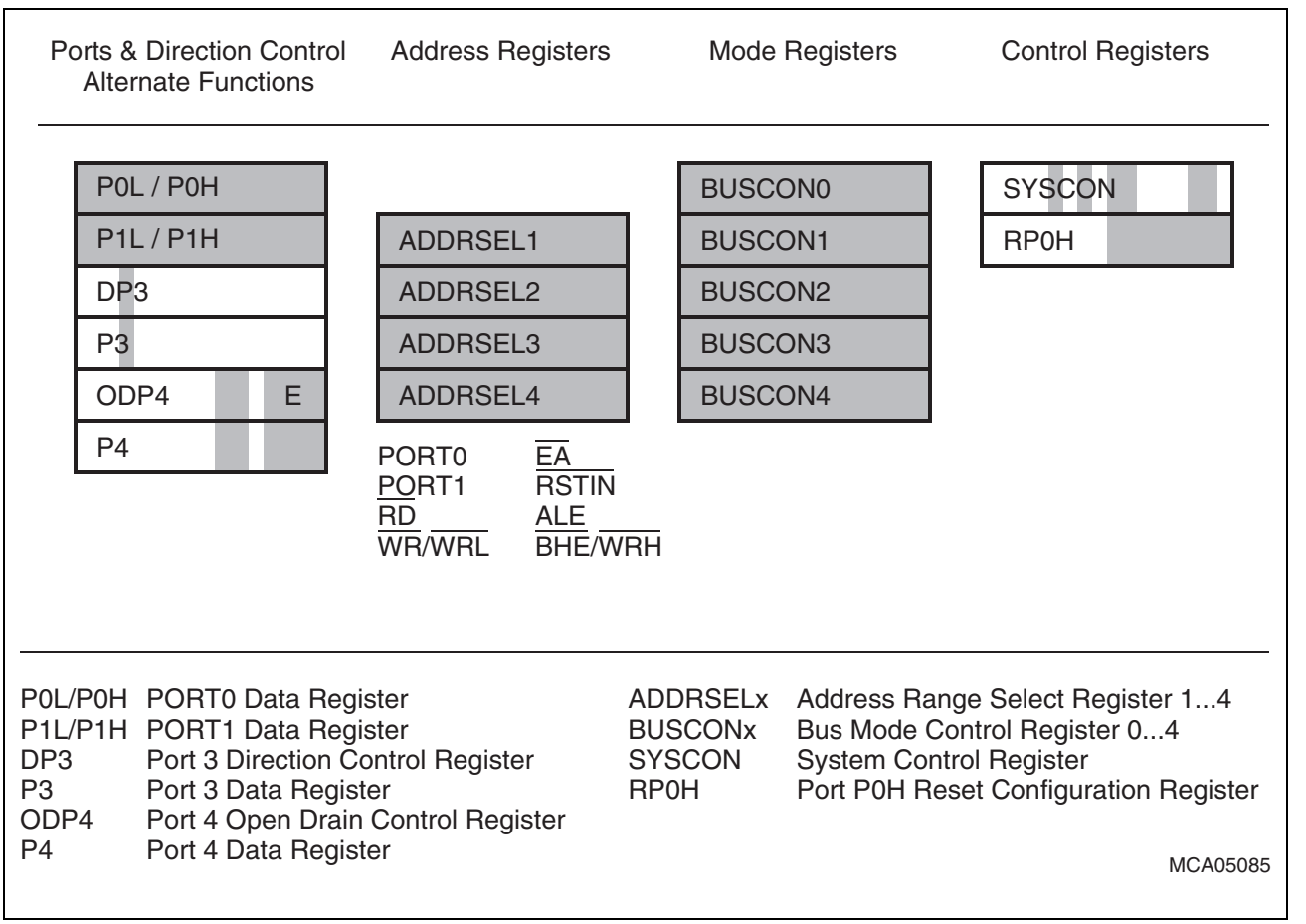


Figure 9-1 SFRs and Port Pins Associated with the External Bus Interface

Accesses to external memory or peripherals are executed by the integrated External Bus Controller (EBC). The function of the EBC is controlled via the SYSCON register and the BUSCONx and ADDRSELx registers. The BUSCONx registers specify the external bus cycles in terms of address (mux/demux), data width (16-bit/8-bit), chip selects, and length (waitstates / ALE / RW delay). These parameters are used for accesses within a specific address area as defined via the corresponding register ADDRSELx.

The four pairs BUSCON1/ADDRSEL1 … BUSCON4/ADDRSEL4 allow definition of four independent “address windows”, while all external accesses outside these windows are controlled via register BUSCON0.

9.1 Single Chip Mode

Single Chip Mode is entered when pin $\overline{EA}$ is high during reset. In this case, register BUSCON0 is initialized with $00C0_H$; this also resets bit BUSACT0, so no external bus is enabled.

In Single Chip Mode, the C164CI operates using only internal resources. No external bus is configured and no external peripherals and/or memory can be accessed. Also no port lines are occupied for the bus interface. When running in Single Chip Mode, however, external access may be enabled by configuring an external bus under software control. Single Chip Mode allows the C164CI to start execution from the internal program memory (Mask-ROM, OTP or Flash memory).

Note: Any attempt to access a location in the external memory space in Single Chip Mode results in the hardware trap ILLBUS if no external bus has been explicitly enabled by software.

9.2 External Bus Modes

When the external bus interface is enabled (bit BUSACTx = ‘1’) and configured (bitfield BTYP), the C164CI uses a subset of its port lines together with some control lines to build the external bus.

Table 9-1 Summary of External Bus Modes

BTYP Encoding	External Data Bus Width	External Address Bus Mode
0 0	8-bit Data	Demultiplexed Addresses
0 1	8-bit Data	Multiplexed Addresses
1 0	16-bit Data	Demultiplexed Addresses
1 1	16-bit Data	Multiplexed Addresses

The bus configuration (BTYP) for the address windows (BUSCON4 … BUSCON1) is selected via software, typically during the initialization of the system.

The bus configuration (BTYP) for the default address range (BUSCON0) is selected via PORT0 during reset, provided that pin $\overline{EA}$ is low during reset. Otherwise, BUSCON0 may be programmed via software, just like the other BUSCON registers.

The 16 MByte address space of the C164CI is divided into 256 segments of 64 KBytes each. The 16-bit intra-segment address is output on PORT0 for multiplexed bus modes or on PORT1 for demultiplexed bus modes. When segmentation is disabled, only one 64-KByte segment can be used and accessed. Otherwise, additional address lines may be output on Port 4 (addressing up to 4 MBytes) and/or several chip select lines may be used to select different memory banks or peripherals. These functions are selected during reset via bitfields SALSEL and CSSEL of register RP0H, respectively.

Note: Bit SGTDIS of register SYSCON determines whether or not the CSP register is saved during interrupt entry (segmentation active or segmentation disabled).

Multiplexed Bus Modes

In the multiplexed bus modes, both the 16-bit intra-segment address and the data use PORT0. The address is time-multiplexed with the data and must be latched externally. The width of the required latch depends on the selected data bus width; that is, an 8-bit data bus requires a byte latch (the address bits A15 … A8 on P0H do not change while P0L multiplexes address and data), a 16-bit data bus requires a word latch (the least significant address line A0 is not relevant for word accesses).
The upper address lines (An … A16) are permanently output on Port 4 (if segmentation is enabled) and do not require latches.

The EBC initiates an external access by generating the Address Latch Enable signal (ALE) and then placing an address on the bus. The falling edge of ALE triggers an external latch to capture the address. After a period of time during which the address must have been latched externally, the address is removed from the bus. The EBC now activates the respective command signal ($\overline{RD}$, $\overline{WR}$, $\overline{WRL}$, $\overline{WRH}$). Data is driven onto the bus either by the EBC (for write cycles) or by the external memory/peripheral (for read cycles). After a period of time determined by the access time of the memory/peripheral, data become valid.

Read cycles: Input data is latched and the command signal is now deactivated. This causes the accessed device to remove its data from the bus which is then tri-stated again.

Write cycles: The command signal is now deactivated. The data remain valid on the bus until the next external bus cycle is started.

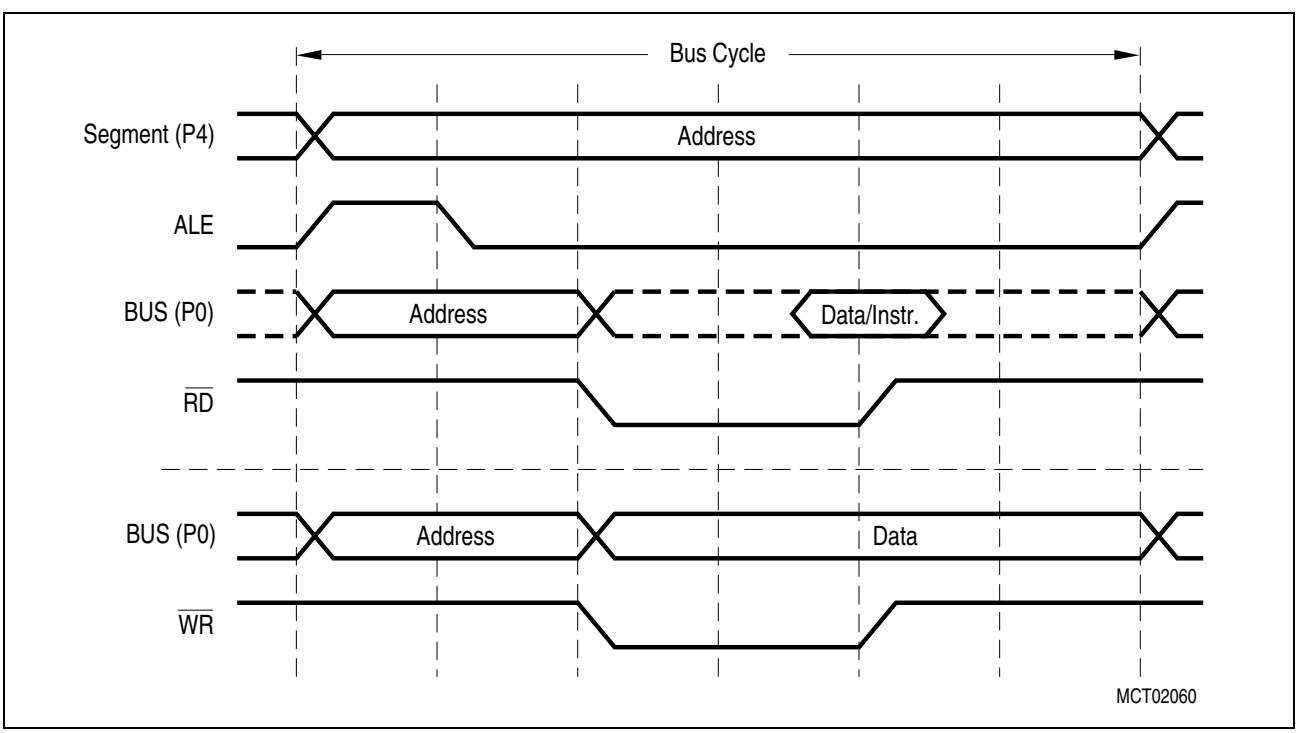


Figure 9-2 Multiplexed Bus Cycle

Demultiplexed Bus Modes

In the demultiplexed bus modes, the 16-bit intra-segment address is permanently output on PORT1 and the data uses PORT0 (16-bit data) or P0L (8-bit data).
The upper address lines are permanently output on Port 4 (if selected via SALSEL during reset). No address latches are required.

The EBC initiates an external access by placing an address on the address bus. After a programmable period of time, the EBC activates the respective command signal ($\overline{RD}$, $\overline{WR}$, $\overline{WRL}$, $\overline{WRH}$). Data is driven onto the data bus either by the EBC (for write cycles) or by the external memory/peripheral (for read cycles). After a period of time determined by the access time of the memory/peripheral, data become valid.

Read cycles: Input data is latched and the command signal is now deactivated. This causes the accessed device to remove its data from the data bus which is then tri-stated again.

Write cycles: The command signal is now deactivated. If a subsequent external bus cycle is required, the EBC places the respective address on the address bus. The data remain valid on the bus until the next external bus cycle is started.

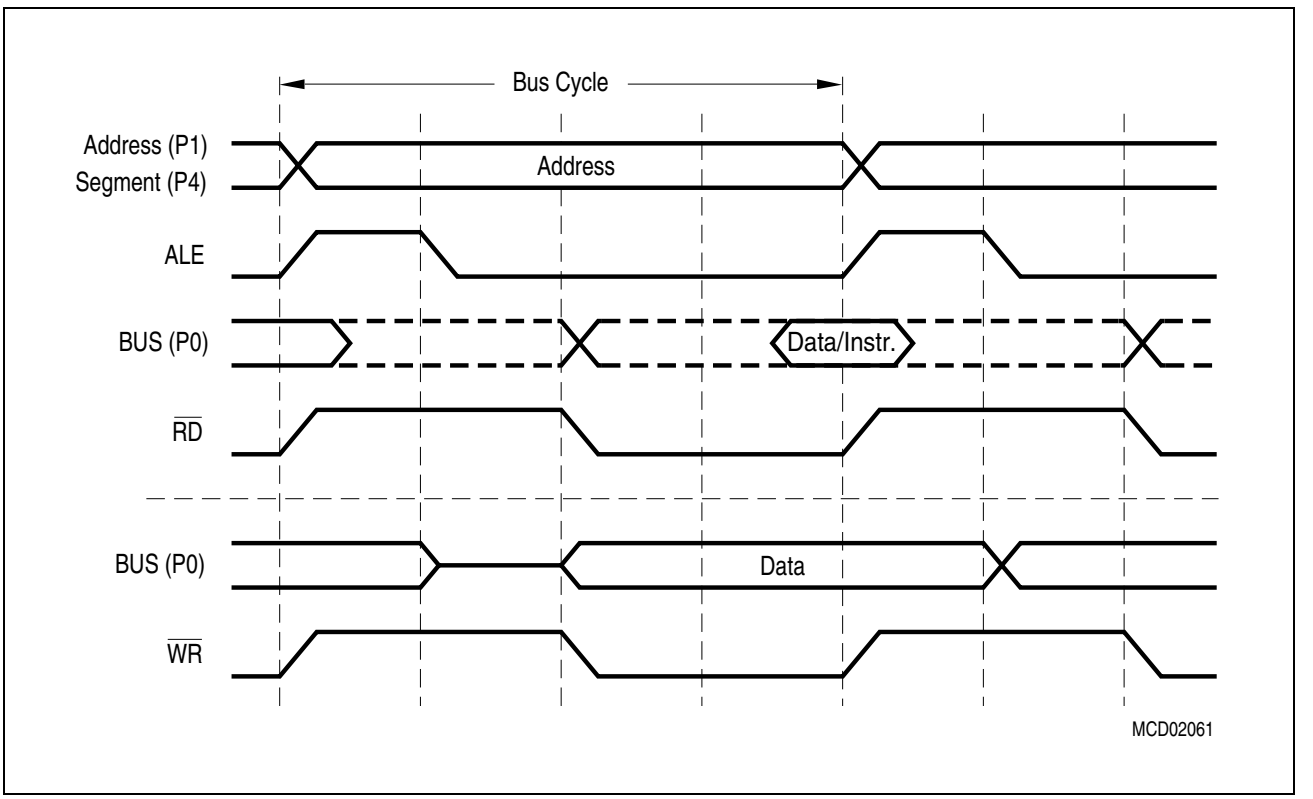


Figure 9-3 Demultiplexed Bus Cycle

Switching between Bus Modes

The EBC allows switching between the different bus modes dynamically, i.e. subsequent external bus cycles may be executed in different ways. Certain address areas may use multiplexed or demultiplexed buses, an 8-bit or 16-bit data bus, or predefined waitstates.

Changes to the external bus characteristics can be initiated in two different ways:

Switching between predefined address windows automatically selects the bus mode associated with the respective window. Predefined address windows allow use of different bus modes without any overhead, but restrict their number to the number of BUSCONs. However, as BUSCON0 controls all address areas which are not covered by the other BUSCONs, this allows to have gaps between these windows, which use the bus mode of BUSCON0.

PORT1 will output the intra-segment address when any of the BUSCON registers selects a demultiplexed bus mode, even if the current bus cycle uses a multiplexed bus mode. This allows to have an external address decoder connected to PORT1 only, while using it for all kinds of bus cycles.

The usage of the BUSCON/ADDRSEL registers is controlled via the issued addresses. When an access (code fetch or data) is initiated, the respective generated physical address defines whether the access made internally uses one of the address windows defined by ADDRSEL4 … 1 or it uses the default configuration in BUSCON0. After initializing the active registers, they are selected and evaluated automatically by interpreting the physical address. No additional switching or selecting is necessary during run time, except when more than the four address windows plus the default are to be used.

Reprogramming the BUSCON and/or ADDRSEL registers allows either changing the bus mode for a given address window or changing the size of an address window using a certain bus mode. Reprogramming allows a great number of different address windows to be used (more than the BUSCONs available) at the expense of the overhead for changing the registers and keeping appropriate tables.

Note: Be careful when changing the configuration for an address area that currently supplies the instruction stream. Due to the internal pipelining, the first instruction fetch that will use the new configuration depends on the instructions prior to the configuration change. Special care is required when changing bits like BUSACT or RDYEN, in order not to cut the instruction stream inadvertently.
Only change the other configuration bits after checking that the respective application can cope with the intended modification(s).
It is recommended to change ADDRSEL registers only while the respective BUSACT bit in the associated BUSCON register is cleared.

Switching from demultiplexed to multiplexed bus mode represents a special case. The bus cycle is started by activating ALE and driving the address to Port 4 and PORT1 as usual, if another BUSCON register selects a demultiplexed bus. However, in the

multiplexed bus modes, the address is also required on PORT0. In this special case, the address on PORT0 is delayed by one CPU clock cycle; this delays the complete (multiplexed) bus cycle and extends the corresponding ALE signal (see **Figure 9-4**).

This extra time is required to allow the previously selected device (via demultiplexed bus) to release the data bus, which would be available in a demultiplexed bus cycle.

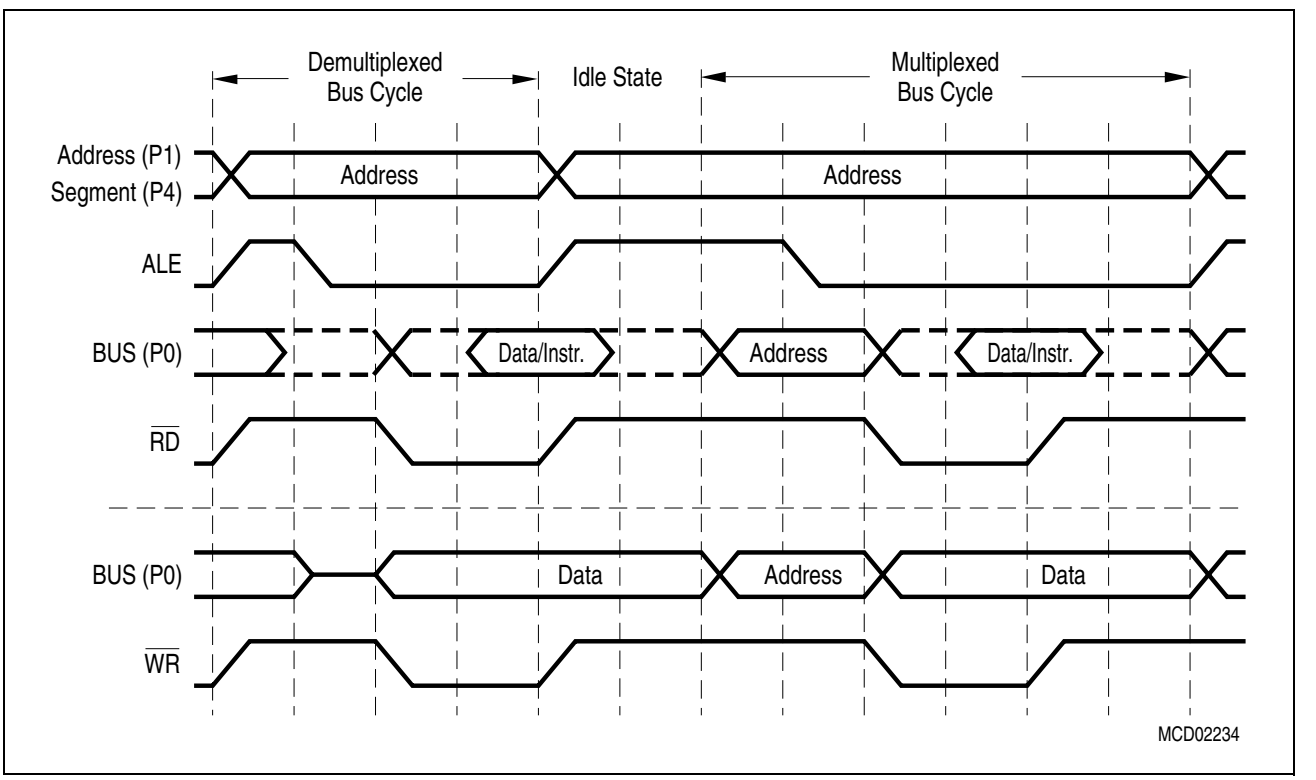


Figure 9-4 Switching from Demultiplexed to Multiplexed Bus Mode

Switching between external resources (such as different peripherals) may incur a problem if the previously accessed resource needs some time to switch off its output drivers (after a read) and the resource to be accessed next switches its output drivers on very quickly. In systems running at higher frequencies, this may lead to a bus conflict (the switch off delays are normally independent from the clock frequency).

In such a case, an additional waitstate can automatically be inserted when leaving a certain address window, i.e. when the next cycle accesses a different window. This waitstate is controlled in the same way as the waitstate when switching from demultiplexed to multiplexed bus mode, see **Figure 9-4**.

BUSCON switch waitstates are enabled via bits BSWCx in the BUSCON registers. By enabling the automatic BUSCON switch waitstate (BSWCx = ‘1’) there is no impact on the system performance as long as the external bus cycles access the same address window. Only if the following cycle accesses a different window is a waitstate inserted between the last access to the previous window and the first access to the new window.

After reset, no BUSCON switch waitstates are selected.

External Data Bus Width

The EBC can operate on 8-bit or 16-bit wide external memory/peripherals. A 16-bit data bus uses PORT0, while an 8-bit data bus uses only P0L, the lower byte of PORT0. This saves on address latches, bus transceivers, bus routing, and memory cost at the expense of transfer time. The EBC can control word accesses on an 8-bit data bus as well as byte accesses on a 16-bit data bus.

Word accesses on an 8-bit data bus are automatically split into two subsequent byte accesses, where the low byte is accessed first, then the high byte. The assembly of bytes into words and the disassembly of words into bytes is handled by the EBC and is transparent to the CPU and the programmer.

Byte accesses on a 16-bit data bus require that the upper and lower half of the memory can be accessed individually. In this case, the upper byte is selected with the $\overline{BHE}$ signal, while the lower byte is selected with the A0 signal. Thus, the two bytes of the memory can be enabled either independent from each other or together when accessing words.

When writing bytes to an external 16-bit device which has a single $\overline{CS}$ input but two $\overline{WR}$ enable inputs (for the two bytes), the EBC can directly generate these two write control signals. This saves the external combination of the $\overline{WR}$ signal with A0 or $\overline{BHE}$. In this case, pin $\overline{WR}$ serves as $\overline{WRL}$ (write low byte) and pin $\overline{BHE}$ serves as $\overline{WRH}$ (write high byte). Bit WRCFG in register SYSCON selects the operating mode for pins $\overline{WR}$ and $\overline{BHE}$. The respective byte will be written on both data bus halves.

When reading bytes from an external 16-bit device, whole words may be read and the C164CI automatically selects the byte to be input and discards the other. However, care must be taken when reading devices that change state when being read, such as FIFOs, interrupt status registers, etc. In this case, individual bytes should be selected using $\overline{BHE}$ and A0.

Table 9-2 Bus Mode versus Performance

Bus Mode	Transfer Rate (Speed factor for byte/ word/dword access)	System Requirements	Free IO Lines
8-bit Multiplexed	Very low (1.5 / 3 / 6)	Low (8-bit latch, byte bus)	P1H, P1L
8-bit Demultipl.	Low (1 / 2 / 4)	Very low (no latch, byte bus)	P0H
16-bit Multiplexed	High (1.5 / 1.5 / 3)	High (16-bit latch, word bus)	P1H, P1L
16-bit Demultipl.	Very high (1 / 1 / 2)	Low (no latch, word bus)	---

Note: PORT1 becomes available for general purpose IO, when none of the BUSCON registers selects a demultiplexed bus mode.

Disable/Enable Control for Pin $\overline{\text{BHE}}$ (BYTDIS)

Bit BYTDIS is provided for controlling the active low Byte High Enable ($\overline{\text{BHE}}$) pin. The function of the $\overline{\text{BHE}}$ pin is enabled if the BYTDIS bit contains a ‘0’. Otherwise, it is disabled and the pin can be used as standard IO pin. The $\overline{\text{BHE}}$ pin is implicitly used by the External Bus Controller to select one of two byte-organized memory chips, connected to the C164CI via a word-wide external data bus. After reset, the $\overline{\text{BHE}}$ function is enabled automatically (BYTDIS = ‘0’), if a 16-bit data bus is selected during reset, otherwise it is disabled (BYTDIS = ‘1’). It may be disabled if byte access to 16-bit memory is not required and the $\overline{\text{BHE}}$ signal is not used.

Segment Address Generation

During external accesses, the EBC generates a (programmable) number of address lines on Port 4. This extends the 16-bit address output on PORT0 or PORT1 and so increases the accessible address space. The number of segment address lines is selected during reset and coded in bit field SALSEL in register RP0H (see **Table 9-3**).

Table 9-3 Decoding of Segment Address Lines

SALSEL	Segment Address Lines	Directly accessible Address Space
1 1	Two: A17 … A16	256 KByte (Default without pull-downs)
1 0	Six: A21 … A16	4 MByte (Maximum)
0 1	None	64 KByte (Minimum)
0 0	Four: A19 … A16	1 MByte

Note: The total accessible address space may be increased by accessing several banks which are distinguished by individual chip select lines.
If Port 4 is used to output segment address lines, in most cases, the drivers must operate in push/pull mode. Ensure that OPD4 does not select open drain mode in this case.

$\overline{CS}$ Signal Generation

During external accesses, the EBC can generate a (programmable) number of $\overline{CS}$ lines on Port 4. This allows direct selection of external peripherals or memory banks without requiring an external decoder. The number of $\overline{CS}$ lines is selected during reset and coded in bit field CSSEL in register RP0H (see **Table 9-4**).

Table 9-4 Decoding of Chip Select Lines

CSSEL	Chip Select Lines	Note
1 1	Four: $\overline{CS3}$ … $\overline{CS0}$	Default without pull-downs
1 0	None	–
0 1	Two: $\overline{CS1}$ … $\overline{CS0}$	–
0 0	Three: $\overline{CS2}$ … $\overline{CS0}$	–

The $\overline{CSx}$ outputs are associated with the BUSCONx registers and are driven active (low) for any access within the address area defined for the respective BUSCON register. For any access outside this defined address area, the respective $\overline{CSx}$ signal will go inactive (high). At the beginning of each external bus cycle, the corresponding valid $\overline{CS}$ signal is determined and activated. All other $\overline{CS}$ lines are deactivated (driven high) at the same time.

Note: The $\overline{CSx}$ signals will not be updated for an access to any internal address area (when no external bus cycle is started), even if this area is covered by the respective ADDRSELx register. An access to an on-chip X-Peripheral deactivates all external $\overline{CS}$ signals.
Upon accesses to address windows without a selected $\overline{CS}$ line, all selected $\overline{CS}$ lines are deactivated.

The chip select signals support operation in four different modes (see **Table 9-5**); the mode is selected via bits CSWENx and CSRENx in the respective BUSCONx register.

Table 9-5 Chip Select Generation Modes

CSWENx	CSRENx	Chip Select Mode
0	0	Address Chip Select (Default after Reset)
0	1	Read Chip Select
1	0	Write Chip Select
1	1	Read/Write Chip Select

Read or Write Chip Select signals remain active only as long as the associated control signal ($\overline{RD}$ or $\overline{WR}$) is active. This also includes the programmable read/write delay. Read chip select is activated for read cycles only, write chip select is activated for write cycles only, read/write chip select is activated for both read and write cycles (write cycles are assumed if any of the signals $\overline{WRH}$ or $\overline{WRL}$ becomes active). These modes save external glue logic when accessing external devices such as latches or drivers which provide only a single enable input.

Address Chip Select signals remain active during the complete bus cycle. For address chip select signals, two generation modes can be selected via bit CSCFG in register SYSCON:

- A **latched** address chip select signal (CSCFG = ‘0’) becomes active with the falling edge of ALE and becomes inactive at the beginning of an external bus cycle which accesses a different address window. No spikes will be generated on the chip select lines and no changes occur as long as locations within the same address window or within internal memory (excluding X-Peripherals and XRAM) are accessed.
- An **early** address chip select signal (CSCFG = ‘1’) becomes active together with the address and $\overline{BHE}$ (if enabled) and remains active until the end of the current bus cycle. Early address chip select signals are not latched internally and may toggle intermediately while the address is changing.

Note: $\overline{CS0}$ provides a latched address chip select directly after reset (except for single chip mode) when the first instruction is fetched.

Internal pull-up devices hold all $\overline{CS}$ lines high during reset. After the end of a reset sequence, the pull-up devices are switched off and the pin drivers control the pin levels on the selected $\overline{CS}$ lines. Not selected $\overline{CS}$ lines will enter the high-impedance state and are available for general purpose IO.

Segment Address versus Chip Select

The external bus interface of the C164CI supports many configurations for the external memory. By increasing the number of segment address lines, the C164CI can address a linear address space of 256 KBytes, 1 MByte or 4 MBytes. This allows implementation of a large sequential memory area, and also allows access to a great number of external devices using an external decoder. By increasing the number of $\overline{CS}$ lines, the C164CI can access memory banks or peripherals without external glue logic. These two features may be combined to optimize the overall system performance.

*Note: If the number of segment address lines and $\overline{CS}$ lines configured at reset cause overlap (e.g. A17 … A16 **and** $\overline{CS3}$ … $\overline{CS0}$) then the segment address line function will take precedence. In this example, the segment address lines (A16 and A17) will be available but only two chip select lines ($\overline{CS0}$ and $\overline{CS1}$) will be available. Bit SGTDIS of register SYSCON determines whether or not the CSP register is saved during interrupt entry (segmentation active or segmentation disabled).*

9.3 Programmable Bus Characteristics

Important timing characteristics of the external bus interface are user programmable to allow adaptation to a wide range of different external bus and memory configurations with different types of memories and/or peripherals.

The following parameters of an external bus cycle are programmable:

- **ALE Control** defines the ALE signal length and the address hold time after its falling edge
- **Memory Cycle Time** (extendable with 1 … 15 waitstates) defines the allowable access time
- **Memory Tri-State Time** (extendable with 1 waitstate) defines the time for a data driver to float
- **Read/Write Delay Time** defines when a command is activated after the falling edge of ALE

Note: Internal accesses are executed with maximum speed and therefore are not programmable.
External accesses use the slowest possible bus cycle after reset. The bus cycle timing may then be optimized by the initialization software.

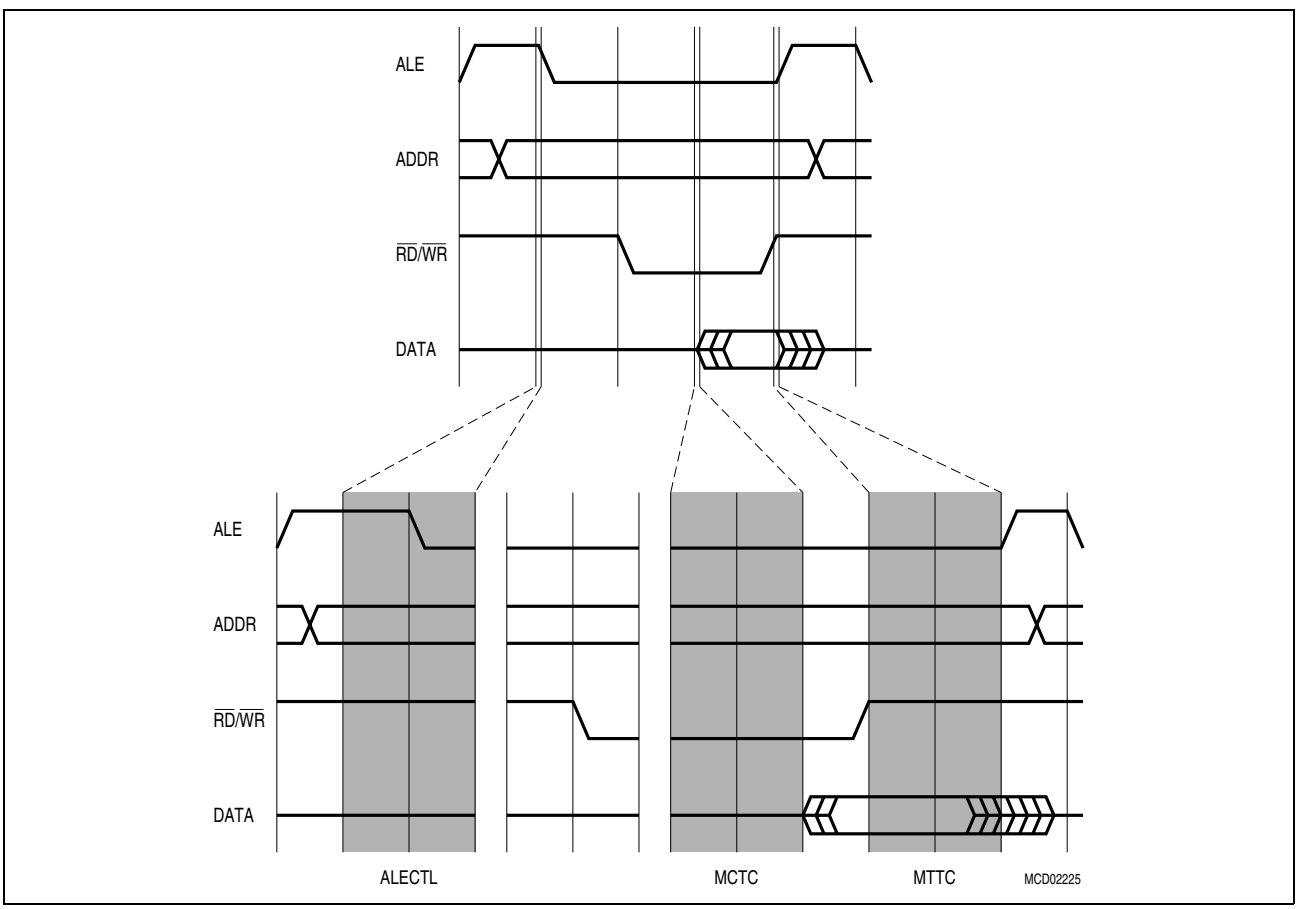


Figure 9-5 Programmable External Bus Cycle

ALE Length Control

The length of the ALE signal and the address hold time after its falling edge are controlled by the ALECTLx bits in the BUSCON registers. When bit ALECTL is set to ‘1’, external bus cycles accessing the respective address window will have their ALE signals prolonged by half a CPU clock (1 TCL). Also, the address hold time after the falling edge of ALE (on a multiplexed bus) will be prolonged by half a CPU clock, so the data transfer within a bus cycle refers to the same CLKOUT edges as usual (the data transfer is delayed by one CPU clock). This allows more time for the address to be latched.

Note: ALECTL0 is ‘1’ after reset to select the slowest possible bus cycle, the other ALECTLx are ‘0’ after reset.

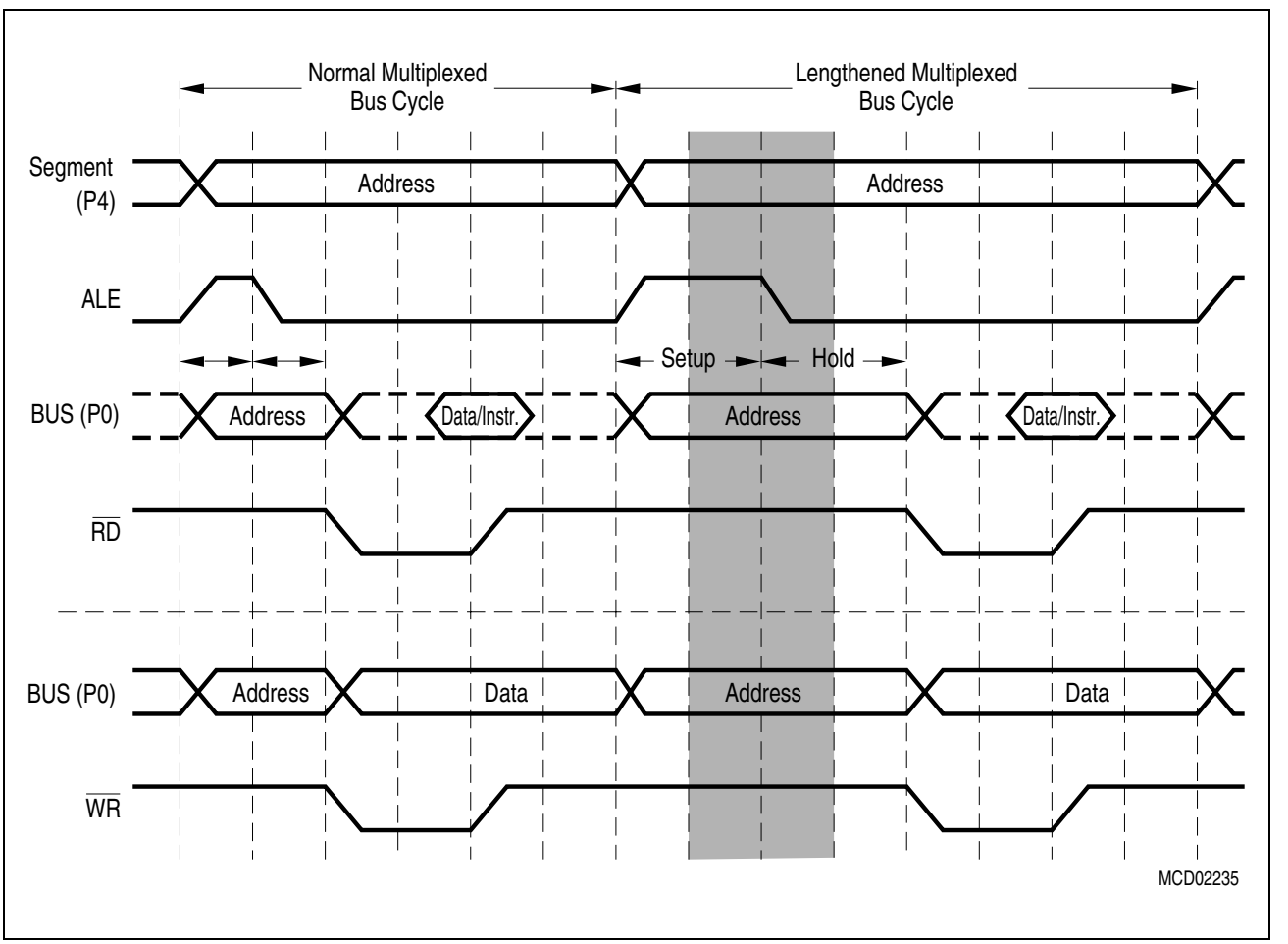


Figure 9-6 ALE Length Control

Programmable Memory Cycle Time

The C164CI allows the user to adjust the controller's external bus cycles to the access time of the respective memory or peripheral. This access time is the total time required to move the data to the destination. It represents the period of time during which the controller's signals do not change.

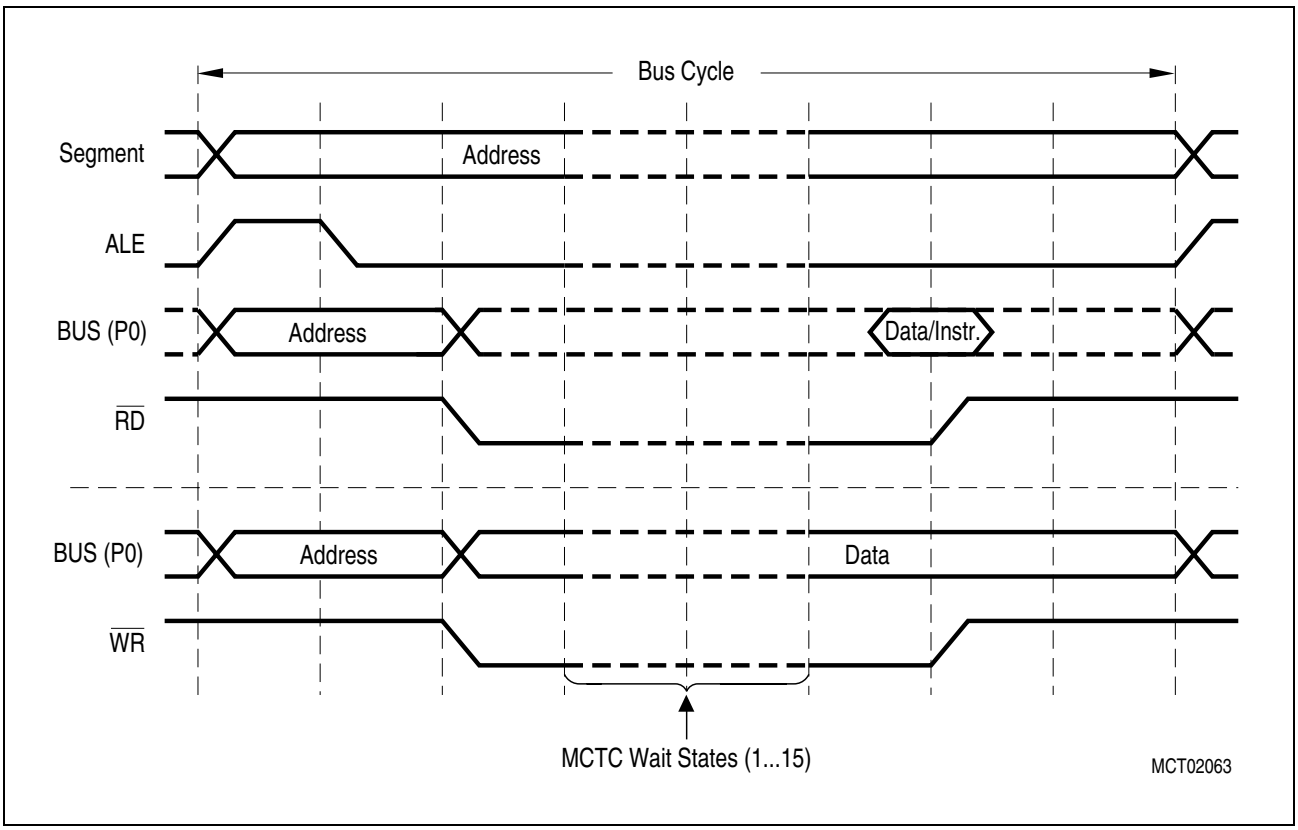


Figure 9-7 Memory Cycle Time

The external bus cycles of the C164CI can be extended for a memory or peripheral which cannot keep pace with the controller's maximum speed. This is accomplished by introducing wait states during the access (see **Figure 9-7**). During these memory cycle time wait states, the CPU is idle if this access is required for the execution of the current instruction.

The memory cycle time wait states can be programmed in increments of one CPU clock (2 TCL) within a range from 0 to 15 (default after reset) via the MCTC fields of the BUSCON registers. 15 - <MCTC> waitstates will be inserted.

Programmable Memory Tri-State Time

The C164CI allows the user to adjust the time between two subsequent external accesses to account for the tri-state time of the external device. The tri-state time defines when the external device releases the bus after deactivation of the read command ($\overline{RD}$).

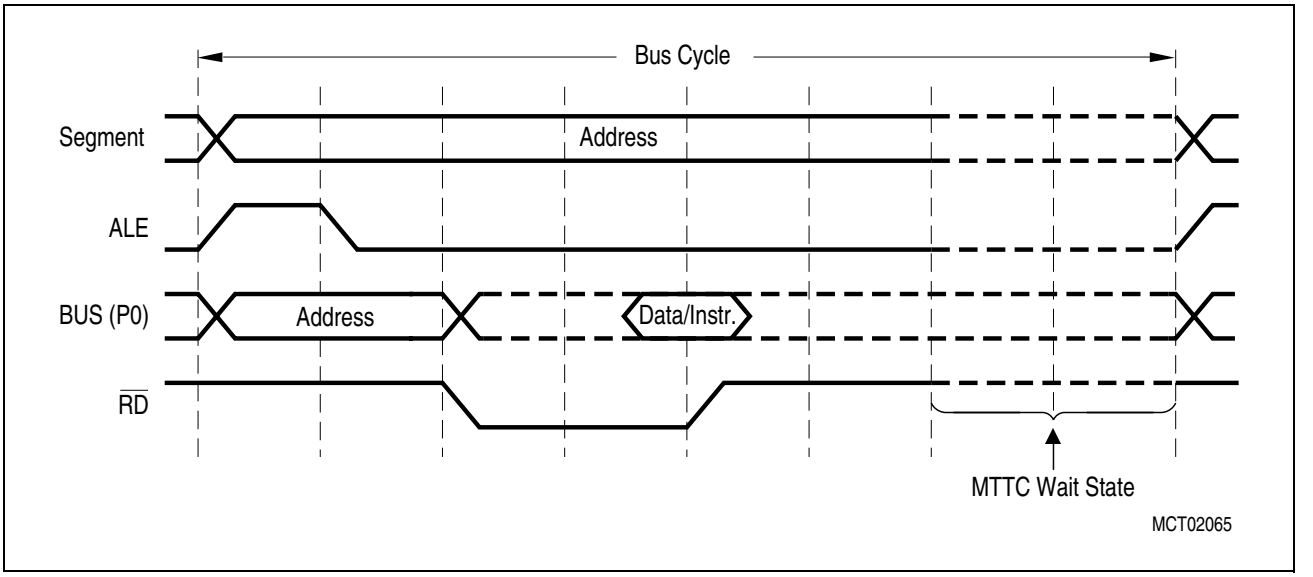


Figure 9-8 Memory Tri-State Time

The output of the next address on the external bus can be delayed for a memory or peripheral which needs more time to switch off its bus drivers. This is accomplished by introducing a wait state after the previous bus cycle (see **Figure 9-8**). During this memory tri-state time wait state, the CPU is not idle, so CPU operations will be slowed down only if a subsequent external instruction or data fetch operation is required during the next instruction cycle.

The memory tri-state time waitstate requires one CPU clock (2 TCL) and is controlled via the MTTCx bits of the BUSCON registers. A waitstate will be inserted if bit MTTCx is ‘0’ (default after reset).

Note: External bus cycles in multiplexed bus modes implicitly add one tri-state time waitstate in addition to the programmable MTTC waitstate.

Read/Write Signal Delay

The C164CI allows the user to adjust the timing of the read and write commands to account for timing requirements of external peripherals. The read/write delay controls the time between the falling edge of ALE and the falling edge of the command. Without read/write delay, the falling edges of ALE and command(s) coincide (except for propagation delays). With the delay enabled, the command(s) become active half a CPU clock (1 TCL) after the falling edge of ALE.

The read/write delay does not extend the memory cycle time, and does not slow down the controller in general. In multiplexed bus modes, however, the data drivers of an external device may conflict with the C164CI's address when the early $\overline{RD}$ signal is used. Therefore, multiplexed bus cycles should always be programmed with read/write delay.

The read/write delay is controlled via the RWDCx bits in the BUSCON registers. The command(s) will be delayed if bit RWDCx is '0' (default after reset).

Early $\overline{WR}$ Signal Deactivation

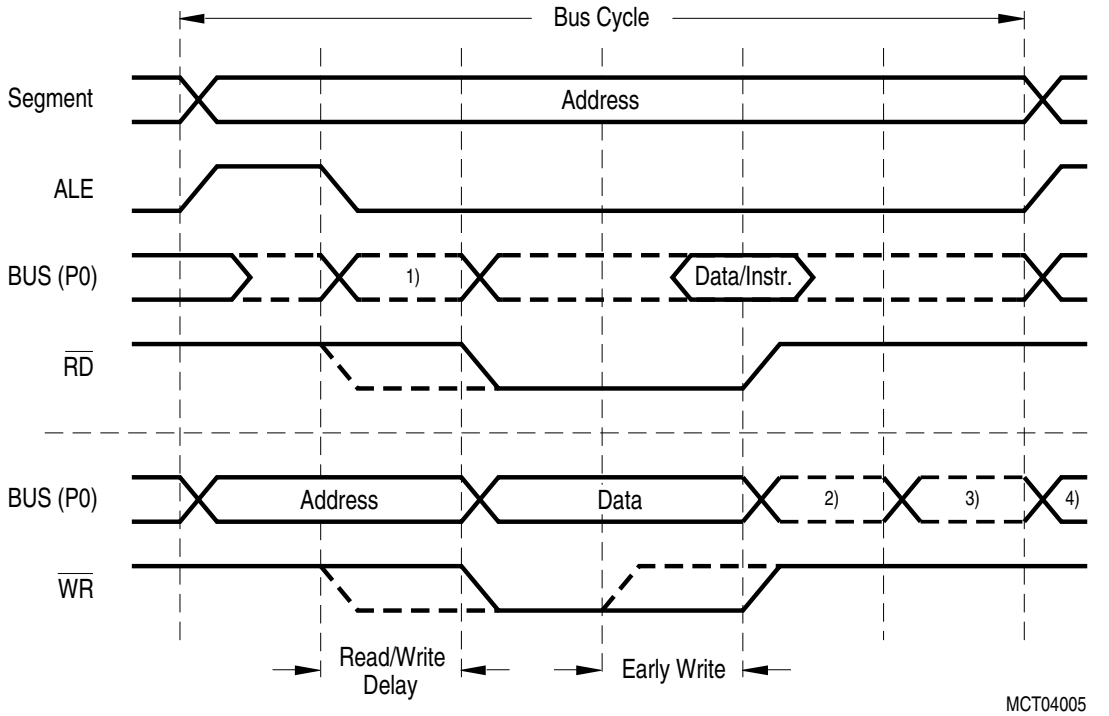
The duration of an external write access can be shortened by one TCL. The $\overline{WR}$ signal is activated (driven low) in the standard way, but can be deactivated (driven high) one TCL earlier than defined in the standard timing. In this case, the data output drivers will also be deactivated one TCL earlier.

This is especially useful in systems which operate on higher CPU clock frequencies and employ external modules (memories, peripherals, etc.) which switch on their own data drivers very quickly in response to signals such as a chip select.

Conflicts between the output drivers of the C164CI and the external peripheral's output drivers can be avoided by selecting early $\overline{WR}$ for the C164CI.

Note: Ensure that the reduced $\overline{WR}$ low time still matches the requirements of the external peripheral/memory.

Early $\overline{WR}$ deactivation is controlled via the EWENx bits in the BUSCON registers. The $\overline{WR}$ signal will be shortened if bit EWENx is '1' (default after reset is a standard $\overline{WR}$ signal, that is, EWENx = '0').



1) The data drivers from the previous bus cycle should be disabled when the $\overline{RD}$ signal becomes active.
2) Data drivers are disabled in an early-write cycle.
3) Data drivers are disabled in a demultiplexed normal-write cycle.
4) Data drivers are disabled in a multiplexed normal-write cycle.

Figure 9-9 Read/Write Signal Duration Control

9.4 Controlling the External Bus Controller

A set of registers controls the functions of the EBC. General features such as the usage of interface pins ($\overline{WR}$, $\overline{BHE}$), segmentation, and internal ROM mapping are controlled via register SYSCON. Bus cycle properties such as chip select mode, length of ALE, external bus mode, read/write delay, and waitstates are controlled via registers BUSCON4 … BUSCON0. Four of these registers (BUSCON4 … BUSCON1) have an address select register (ADDRSEL4 … ADDRSEL1) associated with them. This allows specifying up to four address areas and the individual bus characteristics within these areas. All accesses not covered by these four areas are then controlled via BUSCON0. This allows memory components or peripherals to be used with different interfaces within the same system while optimizing accesses to each of them.

SYSCON
System Control Register **SFR ($FF12_H/89_H$)** **Reset Value: $0XX0_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STKSZ			ROMS1	SGTDIS	ROMEN	BYTDIS	CLKEN	WRCFG	CSCFG	-	OWDDIS	BDRSTEN	XPEN	VISIBLE	-
rw			rw	rw	rwh	rwh	rw	rwh	rw	-	rwh	rw	rw	rw	-

Bit	Function
VISIBLE	**Visible Mode Control** 0: Accesses to XBUS peripherals are done internally. 1: XBUS peripheral accesses are made visible on the external pins.
XPEN	**XBUS Peripheral Enable Bit** 0: Accesses to the on-chip X-Peripherals and their functions are disabled. 1: The on-chip X-Peripherals are enabled and can be accessed.
BDRSTEN	**Bidirectional Reset Enable Bit** 0: Pin $\overline{RSTIN}$ is an input only. 1: Pin $\overline{RSTIN}$ is pulled low during the internal reset sequence after any reset.
OWDDIS	**Oscillator Watchdog Disable Bit** 0: The on-chip oscillator watchdog is enabled and active. 1: The on-chip oscillator watchdog is disabled and the CPU clock is always fed from the oscillator input.
CSCFG	**Chip Select Configuration Control** 0: Latched $\overline{CS}$ mode. The $\overline{CS}$ signals are latched internally and driven to the (enabled) port pins synchronously. 1: Unlatched $\overline{CS}$ mode. The $\overline{CS}$ signals are directly derived from the address and driven to the (enabled) port pins.

Bit	Function
WRCFG	**Write Configuration Control** (Set according to pin P0H.0 during reset) 0: Pins $\overline{WR}$ and $\overline{BHE}$ retain their normal function. 1: Pin $\overline{WR}$ acts as $\overline{WRL}$, pin $\overline{BHE}$ acts as $\overline{WRH}$.
CLKEN	**System Clock Output Enable** (CLKOUT) 0: CLKOUT disabled: pin may be used for general purpose IO or for signal FOUT. 1: CLKOUT enabled: pin outputs the system clock signal.
BYTDIS	**Disable/Enable Control for Pin $\overline{BHE}$** (Set according to data bus width) 0: Pin $\overline{BHE}$ enabled. 1: Pin $\overline{BHE}$ disabled, pin may be used for general purpose IO.
ROMEN	**Internal ROM Enable** (Set according to pin $\overline{EA}$ during reset) 0: Internal program memory disabled, accesses to the ROM area use the external bus. 1: Internal program memory enabled.
SGTDIS	**Segmentation Disable/Enable Control** 0: Segmentation enabled. (CSP is saved/restored during interrupt entry/exit) 1: Segmentation disabled (Only IP is saved/restored).
ROMS1	**Internal ROM Mapping** 0: Internal ROM area mapped to segment 0 ($00'0000_H$ … $00'7FFF_H$). 1: Internal ROM area mapped to segment 1 ($01'0000_H$ … $01'7FFF_H$).
STKSZ	**System Stack Size** Selects the size of the system stack (in the internal RAM) from 32 to 1024 words.

Note: Register SYSCON cannot be changed after execution of the EINIT instruction. Bit SGTDIS controls the correct stack operation (push/pop of CSP or not) during traps and interrupts.

The layout of the BUSCON registers and ADDRSEL registers is identical (respectively).

Registers BUSCON4 … BUSCON1 control the selected address windows and are completely under software control. Register BUSCON0, which for example, is also used for the very first code access after reset, is partly controlled by hardware; that is, it is initialized via PORT0 during the reset sequence. This hardware control allows defining an appropriate external bus for systems where no internal program memory is provided.

BUSCON0

Bus Control Register 0 **SFR ($FF0C_H/86_H$)** **Reset Value: $0XX0_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSWEN0	CSREN0	-	-	BSWC0	BUSACT0	ALECTL0	EWEN0	BTYP		MTTC0	RWDC0	MCTC			
rw	rw	-	-	rw	rwh	rwh	rw	rwh		rw	rw	rw			

BUSCON1

Bus Control Register 1 **SFR ($FF14_H/8A_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSWEN1	CSREN1	-	-	BSWC1	BUSACT1	ALECTL1	EWEN1	BTYP		MTTC1	RWDC1	MCTC			
rw	rw	-	-	rw	rw	rw	rw	rw		rw	rw	rw			

BUSCON2

Bus Control Register 2 **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
CSWEN2	CSREN2	-	-	BSWC2	BUSACT2	ALECTL2	EWEN2	BTYP		MTTC2	RWDC2	MCTC			
rw	rw	-	-	rw	rw	rw	rw	rw		rw	rw	rw			

BUSCON3

Bus Control Register 3 **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
CSWEN3	CSREN3	-	-	BSWC3	BUSACT3	ALECTL3	EWEN3	BTYP		MTTC3	RWDC3	MCTC			
rw	rw	-	-	rw	rw	rw	rw	rw		rw	rw	rw			

BUSCON4

Bus Control Register 4 **SFR ($FF1A_H/8D_H$)** **Reset Value: 0000_H**

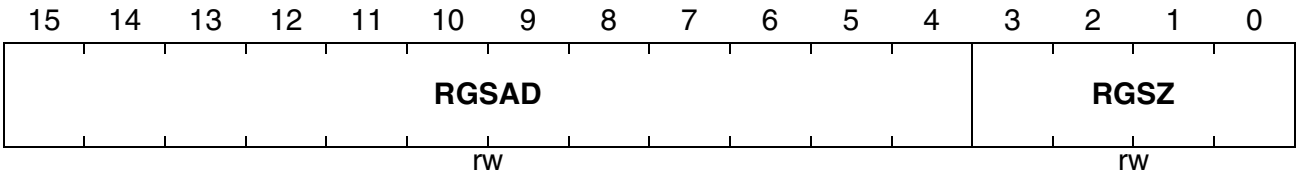
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSWEN4	CSREN4	-	-	BSWC4	BUSACT4	ALECTL4	EWEN4	BTYP		MTTC4	RWDC4	MCTC			
rw	rw	-	-	rw	rw	rw	rw	rw		rw	rw	rw			

Note: BUSCON0 is initialized with $00C0_H$ if pin $\overline{EA}$ is high during reset. If pin $\overline{EA}$ is low during reset, bits BUSACT0 and ALECTL0 are set ('1') and bit field BTYP is loaded with the bus configuration selected via PORT0.

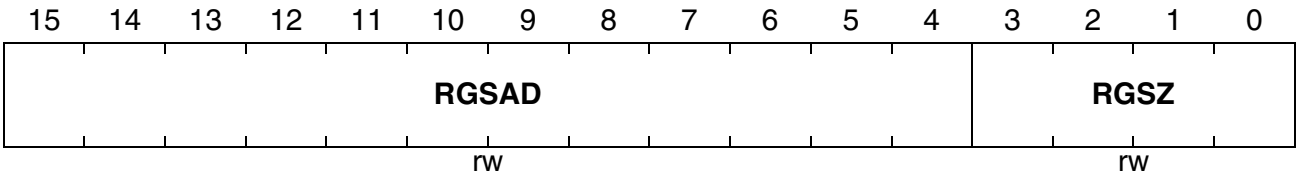
Bit	Function
MCTC	**Memory Cycle Time Control** (Number of memory cycle time wait states) 0000: 15 waitstates … (Number = 15 - <MCTC>) 1111: No waitstates
RWDCx	**Read/Write Delay Control for BUSCONx** 0: With rd/wr delay: activate command 1 TCL after falling edge of ALE 1: No rd/wr delay: activate command with falling edge of ALE
MTTCx	**Memory Tristate Time Control** 0: 1 waitstate 1: No waitstate
BTYP	**External Bus Configuration** 00: 8-bit Demultiplexed Bus 01: 8-bit Multiplexed Bus 10: 16-bit Demultiplexed Bus 11: 16-bit Multiplexed Bus *Note: For BUSCON0 BTYP is defined via PORT0 during reset.*
EWENx	**Early Write Enable** 0: Normal $\overline{WR}$ signal 1: Early write: The $\overline{WR}$ signal is deactivated and write data is tristated one TCL earlier
ALECTLx	**ALE Lengthening Control** 0: Normal ALE signal 1: Lengthened ALE signal
BUSACTx	**Bus Active Control** 0: External bus disabled 1: External bus enabled within respective address window (ADDRSEL)
BSWCx	**BUSCON Switch Control** 0: Address windows are switched immediately 1: A tristate waitstate is inserted if the next bus cycle accesses a window different from the one controlled by this BUSCON register.[1)]
CSRENx	**Read Chip Select Enable** 0: The $\overline{CS}$ signal is independent of the read command ($\overline{RD}$) 1: The $\overline{CS}$ signal is generated for the duration of the read command
CSWENx	**Write Chip Select Enable** 0: The $\overline{CS}$ signal is independent of the write cmd. ($\overline{WR}$, $\overline{WRL}$, $\overline{WRH}$) 1: The $\overline{CS}$ signal is generated for the duration of the write command

1) A BUSCON switch waitstate is enabled by bit BUSCONx.BSWCx of the address window that is left.

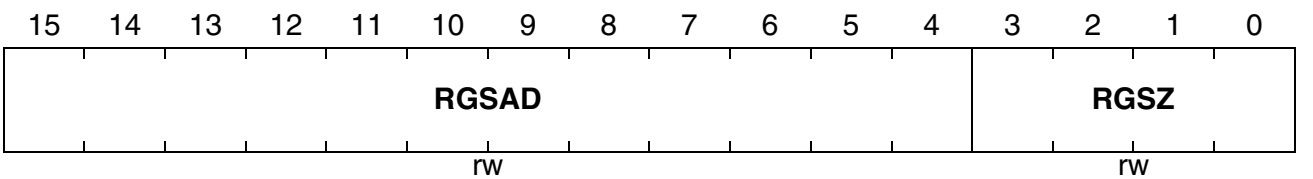
ADDRSEL1
Address Select Register 1 **SFR ($FE18_H/0C_H$)** **Reset Value: 0000_H**



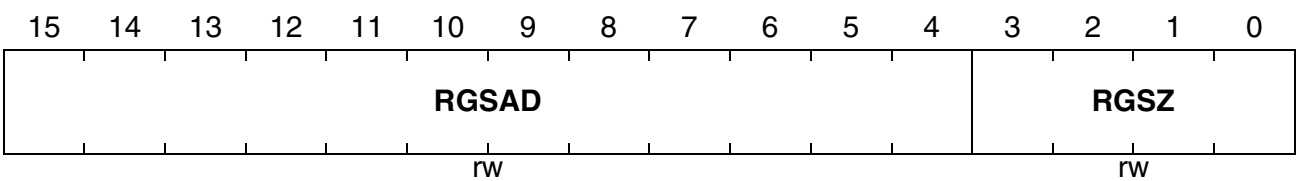
ADDRSEL2
Address Select Register 2 **SFR ($FE1A_H/0D_H$)** **Reset Value: 0000_H**



ADDRSEL3
Address Select Register 3 **SFR ($FE1C_H/0E_H$)** **Reset Value: 0000_H**



ADDRSEL4
Address Select Register 4 **SFR ($FE1E_H/0F_H$)** **Reset Value: 0000_H**



Bit	Function
RGSZ	**Range Size Selection** Defines the size of the address area controlled by the respective BUSCONx/ADDRSELx register pair. See **Table 9-6**.
RGSAD	**Range Start Address** Defines the upper bits of the start address of the respective address area. See **Table 9-6**.

Note: There is no register ADDRSEL0 because register BUSCON0 controls all external accesses outside the four address windows of BUSCON4 … BUSCON1 within the complete address space.

Definition of Address Areas

The four register pairs BUSCON4/ADDRSEL4 … BUSCON1/ADDRSEL1 allow definition of four separate address areas within the address space of the C164CI. Within each of these address areas, external accesses can be controlled by one of the four different bus modes. They are independent of each other and of the bus mode specified in register BUSCON0. Each ADDRSELx register, so to say, cuts out an address window within which the parameters in register BUSCONx are used to control external accesses. The range start address of such a window defines the upper address bits which are not used within the address window of the specified size (see **Table 9-6**). For a given window size, only those upper address bits of the start address are used (marked “R”) which are not implicitly used for addresses inside the window. The lower bits of the start address (marked “x”) are disregarded.

Table 9-6 Address Window Definition

Bit field RGSZ	Resulting Window Size	Relevant Bits (R) of Start Addr. (A12 …)
0 0 0 0	4 KByte	R R R R R R R R R R R R
0 0 0 1	8 KByte	R R R R R R R R R R R x
0 0 1 0	16 KByte	R R R R R R R R R R x x
0 0 1 1	32 KByte	R R R R R R R R R x x x
0 1 0 0	64 KByte	R R R R R R R R x x x x
0 1 0 1	128 KByte	R R R R R R R x x x x x
0 1 1 0	256 KByte	R R R R R R x x x x x x
0 1 1 1	512 KByte	R R R R R x x x x x x x
1 0 0 0	1 MByte	R R R R x x x x x x x x
1 0 0 1	2 MByte	R R R x x x x x x x x x
1 0 1 0	4 MByte	R R x x x x x x x x x x
1 0 1 1	8 MByte	R x x x x x x x x x x x
1 1 x x	Reserved.	

Address Window Arbitration

The address windows which can be defined within the address space of the C164CI may partly overlap each other. Thus, for example, small areas may be cut out of bigger windows in order to effectively utilize external resources, especially within segment 0.

For each access, the EBC compares the current address with all address select registers (programmable ADDRSELx and hardwired XADRSx). This comparison is done in four priority levels.

Priority 1: The hardwired XADRSx registers are evaluated first. A match with one of these registers directs the access to the respective X-Peripheral using the corresponding XBCONx register and ignoring all other ADDRSELx registers.

Priority 2: Registers ADDRSEL2 and ADDRSEL4 are evaluated before ADDRSEL1 and ADDRSEL3, respectively. A match with one of these registers directs the access to the respective external area using the corresponding BUSCONx register and ignoring registers ADDRSEL1/3 (see **Figure 9-9**).

Priority 3: A match with registers ADDRSEL1 or ADDRSEL3 directs the access to the respective external area using the corresponding BUSCONx register.

Priority 4: If there is no match with any XADRSx or ADDRSELx register, the access to the external bus uses register BUSCON0.

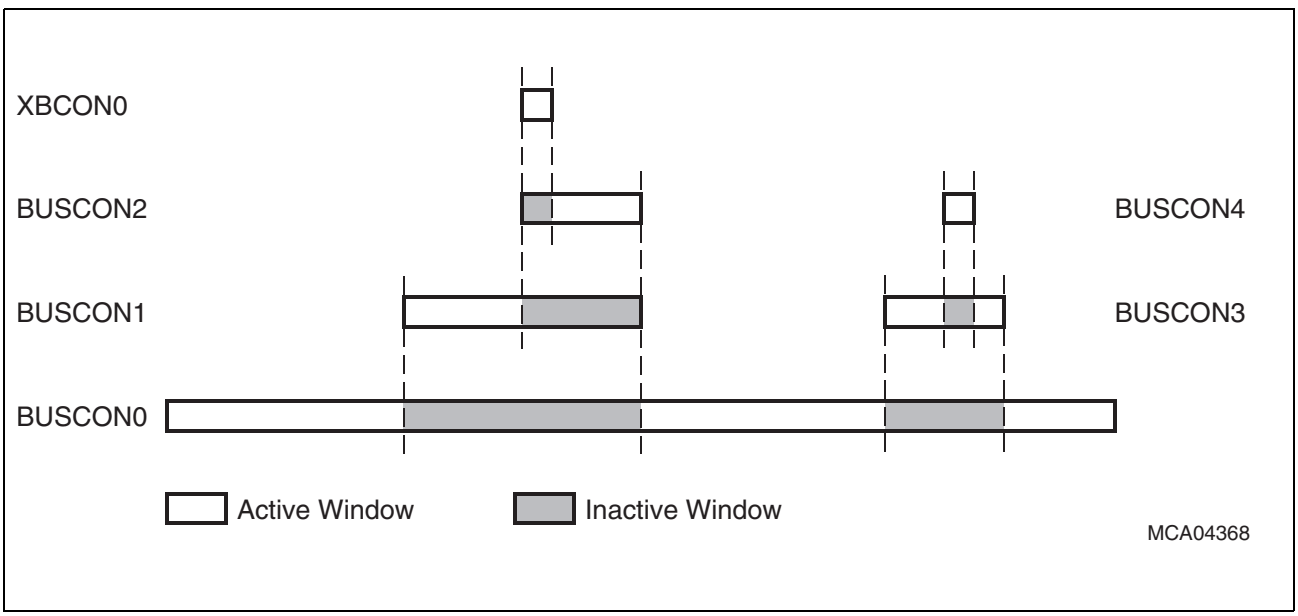
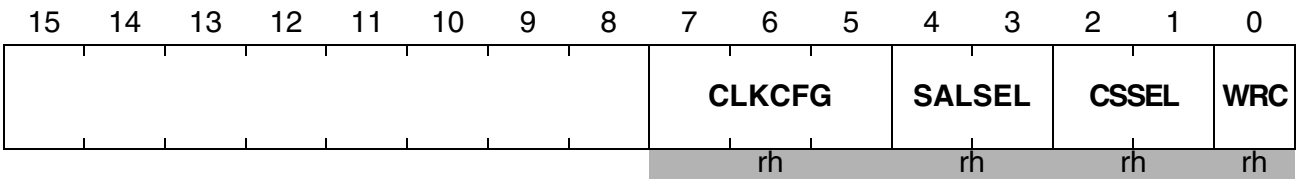


Figure 9-10 Address Window Arbitration

Note: Only the indicated overlaps are defined. All other overlaps lead to erroneous bus cycles; for example, ADDRSEL4 may not overlap ADDRSEL2 or ADDRSEL1. The hardwired XADRSx registers are defined as non-overlapping.

RP0H

Reset Value of P0H **SFR ($F108_H/84_H$)** **Reset Value: - - XX_H**



Bit	Function
WRC	**Write Configuration** 0: Pins $\overline{WR}$ and $\overline{BHE}$ operate as $\overline{WRL}$ and $\overline{WRH}$ signals 1: Pins $\overline{WR}$ and $\overline{BHE}$ operate as $\overline{WR}$ and $\overline{BHE}$ signals
CSSEL	**Chip Select Line Selection** (Number of active $\overline{CS}$ outputs) 00: 3 $\overline{CS}$ lines: $\overline{CS2}$ … $\overline{CS0}$ 01: 2 $\overline{CS}$ lines: $\overline{CS1}$ … $\overline{CS0}$ 10: No $\overline{CS}$ lines at all 11: 4 $\overline{CS}$ lines: $\overline{CS3}$ … $\overline{CS0}$ (Default without pull-downs)
SALSEL	**Segment Address Line Selection** (Nr. of active segment addr. outputs) 00: 4-bit segment address: A19 … A16 01: No segment address lines at all 10: 6-bit segment address: A21 … A16 11: 2-bit segment address: A17 … A16 (Default without pull-downs)
CLKCFG	**Clock Generation Mode Configuration** These pins define the clock generation mode, i.e. the mechanism by which the internal CPU clock is generated from the externally applied (XTAL1) input clock.

Note: RP0H is initialized during the reset configuration and permits to check the current configuration.
*This configuration (except for bit WRC) can be changed via register RSTCON (see **Section 20.5**).*

Precautions and Hints

- The external bus interface is enabled as long as at least one of the BUSCON registers has its BUSACT bit set.
- PORT1 will output the intra-segment address as long as at least one of the BUSCON registers selects a demultiplexed external bus, even for multiplexed bus cycles.
- Not all address windows defined via registers ADDRSELx may overlap each other. The operation of the EBC will be erroneous in such a case. See **“Address Window Arbitration” on Page 9-24**.
- The address windows defined via registers ADDRSELx may overlap internal address areas. Internal accesses will be executed in this case.
- For any access to an internal address area, the EBC will remain inactive (see **Section 9.5**).

9.5 EBC Idle State

When the external bus interface is enabled, but no external access is currently executed, the EBC is idle. As long as only internal resources (from an architecture point of view) such as IRAM, GPRs or SFRs, etc. are used, the external bus interface does not change (see **Table 9-7**).

Accesses to on-chip X-Peripherals are also controlled by the EBC. However, even though an X-Peripheral appears like an external peripheral to the controller, the respective accesses do not generate valid external bus cycles.

Due to timing constraints, address and write data of an XBUS cycle are reflected on the external bus interface (see **Table 9-7**). The “address” mentioned above includes PORT1, Port 4, $\overline{BHE}$, and ALE which also pulses for an XBUS cycle. The external $\overline{CS}$ signals are driven inactive (high) because the EBC switches to an internal $\overline{XCS}$ signal.

The **external control signals** ($\overline{RD}$ and $\overline{WR}$ or $\overline{WRL}$/$\overline{WRH}$ if enabled) **remain inactive** (high).

Table 9-7 Status Of The External Bus Interface During EBC Idle State

Pins	Internal Accesses only	XBUS Accesses
PORT0	Tristated (floating)	Tristated (floating) for read accesses XBUS write data for write accesses
PORT1	Last used external address (if used for the bus interface)	Last used XBUS address (if used for the bus interface)
Port 4	Last used external segment address (on selected pins)	Last used XBUS segment address (on selected pins)
	Active external $\overline{CS}$ signal corresponding to last used address	Inactive (high) for selected $\overline{CS}$ signals
$\overline{BHE}$	Level corresponding to last external access	Level corresponding to last XBUS access
ALE	Inactive (low)	Pulses as defined for X-Peripheral
$\overline{RD}$	Inactive (high)	**Inactive (high)[1)]**
$\overline{WR}$/$\overline{WRL}$	Inactive (high)	**Inactive (high)[1)]**
$\overline{WRH}$	Inactive (high)	**Inactive (high)[1)]**

1) Used and driven in visible mode.

9.6 The XBUS Interface

The C164CI provides an on-chip interface, the XBUS interface, via which integrated customer/application specific peripherals can be connected to the standard controller core. The XBUS is an internal representation of the external bus interface and is operated in the same way.

For each peripheral on the XBUS (X-Peripheral) there is a separate address window controlled by a register pair XBCONx/XADRSx (similar to registers BUSCONx and ADDRSELx). Because an interface to a peripheral is represented in many cases by just a few registers, the XADRSx registers partly select smaller address windows than the standard ADDRSEL registers. As the XBCONx/XADRSx (register pairs control integrated peripherals rather than externally connected ones, they are fixed by mask programming rather than being user programmable.

X-Peripheral accesses provide the same choices as external accesses; so, these peripherals may be bytewide or wordwide. Because the on-chip connection can be implemented very efficiently, for performance reasons, X-Peripherals are implemented only with a separate address bus (in demultiplexed bus mode). Interrupt nodes are provided for X-Peripherals to be integrated.

Note: If you plan to develop a peripheral of your own to be integrated into a C164CI device to create a customer specific version, please ask for the specification of the XBUS interface and for further support.

9.6.1 Accessing the On-chip XBUS Peripherals

Enabling of XBUS Peripherals

After reset, all on-chip XBUS peripherals are disabled. In order to be usable, an XBUS peripheral must be enabled via the global enable bit XPEN in register SYSCON.

Table 9-8 summarizes the XBUS peripherals and also the number of waitstates which are used when accessing the respective peripheral.

Table 9-8 XBUS Peripherals in the C164CI

Associated XBUS Peripheral	Waitstates
CAN1	2
XRAM 2 KByte	0

Visible Mode

The C164CI can mirror on-chip access cycles to its XBUS peripherals so these accesses can be observed or recorded by the external system. This function is enabled via bit VISIBLE in register SYSCON.

Accesses to XBUS peripherals also use the EBC. Due to timing constraints the address bus will change for all accesses using the EBC.

Note: As XBUS peripherals use demultiplexed bus cycles, the respective address is driven on PORT1 in visible mode, even if the external system uses MUX buses only.

If visible mode is activated, accesses to on-chip XBUS peripherals (including control signals $\overline{RD}$, $\overline{WR}$, and $\overline{BHE}$) are mirrored to the bus interface. Accesses to internal resources (program memory, IRAM, GPRs) do not use the EBC and cannot be mirrored to outside.

If visible mode is deactivated, however, no control signals ($\overline{RD}$, $\overline{WR}$) will be activated, that is, there will be no valid external bus cycles.

Note: Visible mode can only work if the external bus is enabled at all.

9.6.2 External Accesses to XBUS Peripherals

The on-chip XBUS peripherals of the C164CI can be accessed from outside via the external bus interface under certain circumstances. In emulation mode the XBUS peripherals are controlled by the bondout-chip.

10 General Purpose Timer Unit

The General Purpose Timer Unit GPT1 has a very flexible multifunctional timer structure which may be used for timing, event counting, pulse width measurement, pulse generation, frequency multiplication, and other purposes.

Block GPT1 contains 3 timers/counters with a maximum resolution of 16 TCL. Each timer may operate independently in a number of different modes such as gated timer or counter mode, or may be concatenated with another timer of the same block. The auxiliary timers of GPT1 may optionally be configured as reload or capture registers for the core timer. GPT1 has alternate input/output functions and specific interrupts associated with it.

10.1 Timer Block GPT1

From a programmer's point of view, the GPT1 block is composed of a set of Special Function Registers (SFRs) as summarized in **Figure 10-1**. Those portions of port and direction registers which are used for alternate functions by the GPT1 block are shaded.

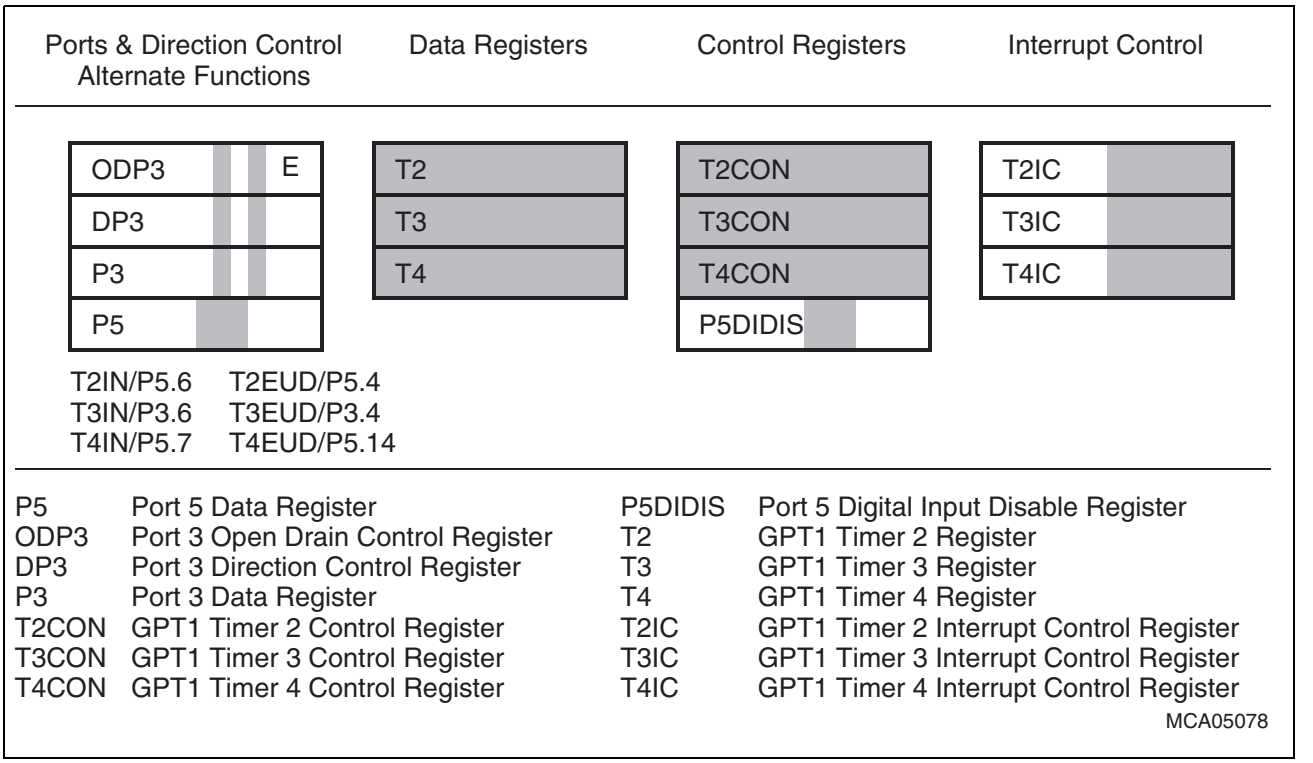


Figure 10-1 SFRs and Port Pins Associated with Timer Block GPT1

All three timers of block GPT1 (T2, T3, T4) can run in 4 basic modes: timer, gated timer, counter, and incremental interface mode. All timers can count either up or down. Each timer has an alternate input function pin (TxIN) associated with it which serves as the gate control in gated timer mode or as the count input in counter mode. The count direction (Up/Down) may be programmed via software or may be dynamically altered by a signal at an external control input pin. Each overflow/underflow of core timer T3 is latched in the toggle FlipFlop T3OTL and may be indicated on an alternate output function pin. The auxiliary timers T2 and T4 may additionally be concatenated with the core timer or may be used as capture or reload registers for the core timer.

The current contents of each timer can be read or modified by the CPU by accessing the corresponding timer registers T2, T3, or T4, located in the non-bitaddressable SFR space. When any of the timer registers is written to by the CPU in the state immediately preceding a timer increment, decrement, reload, or capture operation, the CPU write operation has priority in order to guarantee correct results.

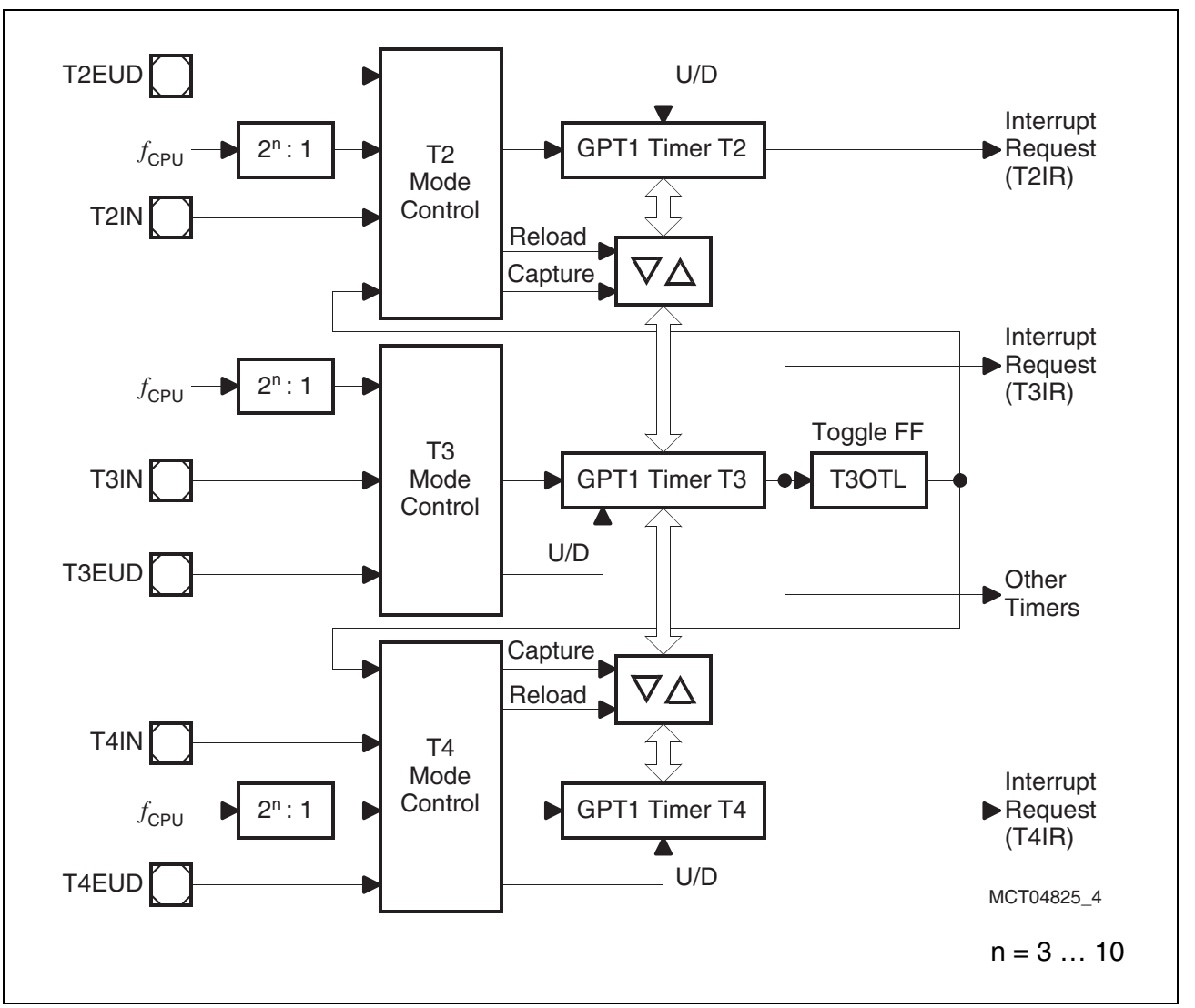


Figure 10-2 GPT1 Block Diagram

10.1.1 GPT1 Core Timer T3

The core timer T3 is configured and controlled via its bitaddressable control register T3CON.

T3CON
Timer 3 Control Register **SFR ($FF42_H/A1_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	T3 OTL	-	T3 UDE	T3 UD	T3R	T3M			T3I		
-	-	-	-	-	rwh	-	rw	rw	rw	rw			rw		

Bit	Function
T3I	**Timer 3 Input Selection** Depends on the operating mode, see respective sections.
T3M	**Timer 3 Mode Control** (Basic Operating Mode) 000: Timer Mode 001: Counter Mode 010: Gated Timer with Gate active low 011: Gated Timer with Gate active high 100: *Reserved*. Do not use this combination. 101: *Reserved*. Do not use this combination. 110: Incremental Interface Mode 111: *Reserved*. Do not use this combination.
T3R	**Timer 3 Run Bit** 0: Timer/Counter 3 stops 1: Timer/Counter 3 runs
T3UD	**Timer 3 Up/Down Control**[1)]
T3UDE	**Timer 3 External Up/Down Enable**[1)]
T3OTL	**Timer 3 Output Toggle Latch** Toggles on each overflow/underflow of T3. Can be set or reset by software.

[1)] For the effects of bits T3UD and T3UDE, refer to the direction **Table 10-1**.

Timer 3 Run Bit

The timer can be started or stopped by software through bit T3R (Timer T3 Run Bit). If T3R = ‘0’, the timer stops. Setting T3R to ‘1’ will start the timer. In gated timer mode, the timer will run only if T3R = ‘1’ and the gate is active (high or low, as programmed).

Count Direction Control

The count direction of the core timer can be controlled either by software or by the external input pin T3EUD (Timer T3 External Up/Down Control Input), an alternate input function of port pin P3.4. These options are selected by bits T3UD and T3UDE in control register T3CON. When the up/down control is provided by software (bit T3UDE = ‘0’), the count direction can be altered by setting or clearing bit T3UD. When T3UDE = ‘1’, pin T3EUD is selected to be the controlling source of the count direction. However, bit T3UD can still be used to reverse the actual count direction, as shown in **Table 10-1**. If T3UD = ‘0’ and pin T3EUD shows a low level, the timer is counting up. With a high level at T3EUD, the timer is counting down. If T3UD = ‘1’, a high level at pin T3EUD specifies counting up, and a low level specifies counting down. The count direction can be changed regardless of whether or not the timer is running.

When pin T3EUD/P3.4 is used as external count direction control input, it must be configured as input; its corresponding direction control bit DP3.4 must be set to ‘0’.

Table 10-1 GPT1 Core Timer T3 Count Direction Control

Pin TxEUD	Bit TxUDE	Bit TxUD	Count Direction
X	0	0	Count Up
X	0	1	Count Down
0	1	0	Count Up
1	1	0	Count Down
0	1	1	Count Down
1	1	1	Count Up

Note: Direction control works the same way for core timer T3 and for auxiliary timers T2 and T4. Therefore, the pins and bits are named Tx …

Timer 3 Output Toggle Latch

An overflow or underflow of timer T3 will clock the toggle bit T3OTL in control register T3CON. T3OTL can also be set or reset by software.

Additionally, T3OTL can be used in conjunction with the timer over/underflows as an input for the counter function or as a trigger source for the reload function of the auxiliary timers T2 and T4. An internal connection is provided for this option.

Timer 3 in Timer Mode

Timer mode for the core timer T3 is selected by setting bit field T3M in register T3CON to '000_B'. In this mode, T3 is clocked with the internal system clock (CPU clock) divided by a programmable prescaler, which is selected by bit field T3I. The input frequency f_{T3} for timer T3 and its resolution r_{T3} are scaled linearly with lower clock frequencies f_{CPU}, as can be seen from the following formula:

$$f_{T3} = \frac{f_{CPU}}{8 \times 2^{<T3I>}} \quad , \quad r_{T3}\,[\mu s] = \frac{8 \times 2^{<T3I>}}{f_{CPU}\,[MHz]}$$

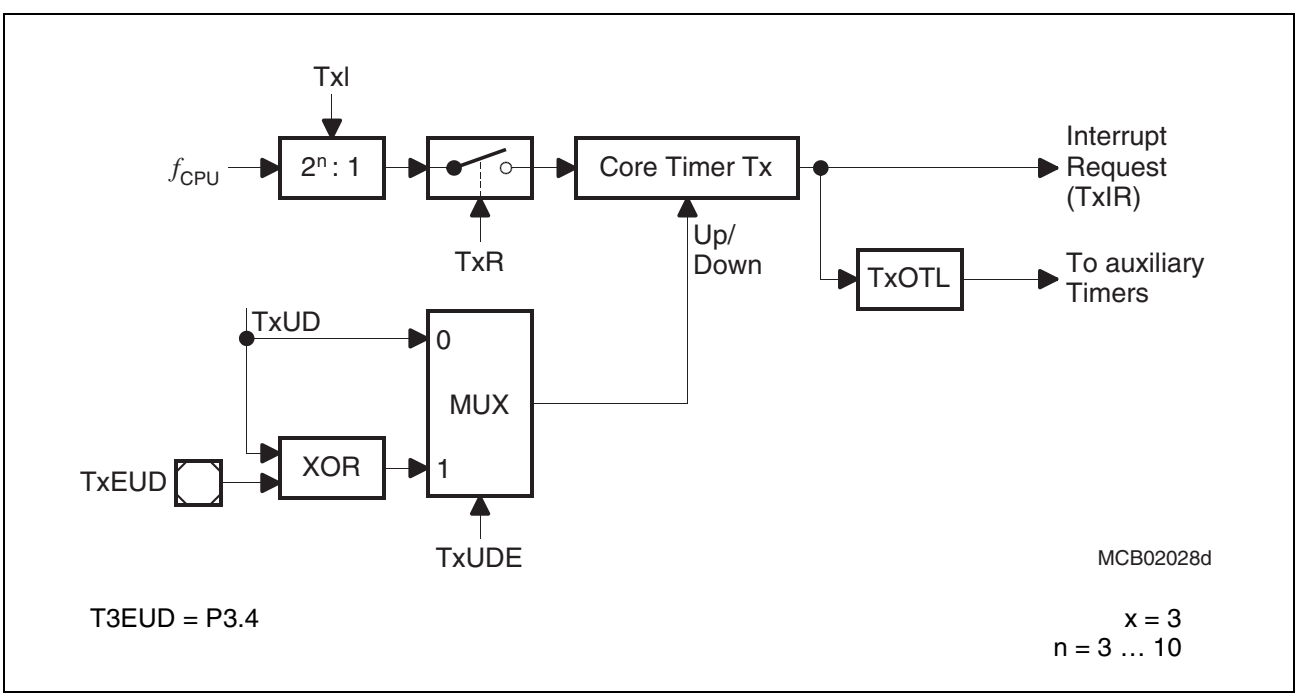


Figure 10-3 Block Diagram of Core Timer T3 in Timer Mode

Timer input frequencies, resolution, and periods resulting from the selected prescaler option are listed in **Table 10-2**. This table also applies to the Gated Timer Mode of T3 and to the auxiliary timers T2 and T4 in timer and gated timer mode. Note that some numbers may be rounded to 3 significant digits.

Table 10-2 GPT1 Timer Input Frequencies, Resolution and Periods @ 20 MHz

f_{CPU} = 20 MHz	Timer Input Selection T2I/T3I/T4I							
	000_B	001_B	010_B	011_B	100_B	101_B	110_B	111_B
Prescaler Factor	8	16	32	64	128	256	512	1024
Input Frequency	2.5 MHz	1.25 MHz	625 kHz	312.5 kHz	156.25 kHz	78.125 kHz	39.06 kHz	19.53 kHz
Resolution	400 ns	800 ns	1.6 µs	3.2 µs	6.4 µs	12.8 µs	25.6 µs	51.2 µs
Period	26.2 ms	52.5 ms	105 ms	210 ms	420 ms	840 ms	1.68 s	3.36 s

Table 10-3 GPT1 Timer Input Frequencies, Resolution and Periods @ 25 MHz

f_{CPU} = 25 MHz	Timer Input Selection T2I/T3I/T4I							
	000_B	001_B	010_B	011_B	100_B	101_B	110_B	111_B
Prescaler Factor	8	16	32	64	128	256	512	1024
Input Frequency	3.125 MHz	1.56 MHz	781.25 kHz	390.62 kHz	195.3 kHz	97.65 kHz	48.83 kHz	24.42 kHz
Resolution	320 ns	640 ns	1.28 µs	2.56 µs	5.12 µs	10.2 µs	20.5 µs	41.0 µs
Period	21.0 ms	41.9 ms	83.9 ms	168 ms	336 ms	671 ms	1.34 s	2.68 s

Table 10-4 GPT1 Timer Input Frequencies, Resolution and Periods @ 33 MHz

f_{CPU} = 33 MHz	Timer Input Selection T2I/T3I/T4I							
	000_B	001_B	010_B	011_B	100_B	101_B	110_B	111_B
Prescaler Factor	8	16	32	64	128	256	512	1024
Input Frequency	4.125 MHz	2.0625 MHz	1.031 MHz	515.62 kHz	257.81 kHz	128.91 kHz	64.45 kHz	32.23 kHz
Resolution	242 ns	485 ns	970 ns	1.94 µs	3.88 µs	7.76 µs	15.5 µs	31.0 µs
Period	15.9 ms	31.8 ms	63.6 ms	127 ms	254 ms	508 ms	1.02 s	2.03 s

Timer 3 in Gated Timer Mode

Gated timer mode for the core timer T3 is selected by setting bit field T3M in register T3CON to '010_B' or '011_B'. Bit T3M.0 (T3CON.3) selects the active level of the gate input. The same options for the input frequency are available in gated timer mode as in timer mode. However, the input clock to the timer in this mode is gated by the external input pin T3IN (Timer T3 External Input).
To enable this operation, pin T3IN must be configured as input, that is, the corresponding direction control bit must contain '0'.

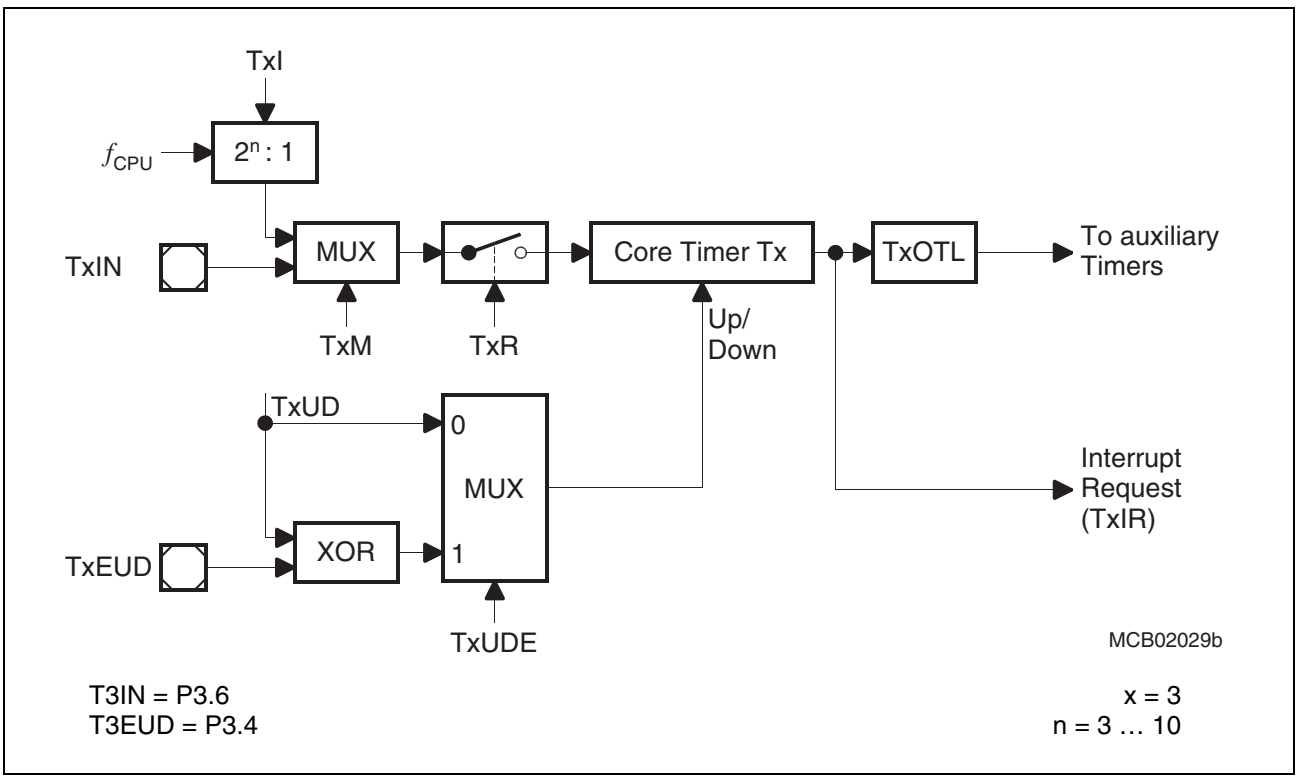


Figure 10-4 Block Diagram of Core Timer T3 in Gated Timer Mode

If T3M.0 = '0', the timer is enabled when T3IN shows a low level. A high level at this pin stops the timer. If T3M.0 = '1', pin T3IN must have a high level in order to enable the timer. Additionally, the timer can be turned on or off by software using bit T3R. The timer will only run, if T3R = '1' and the gate is active. It will stop if either T3R = '0' or the gate is inactive.

Note: A transition of the gate signal at pin T3IN does not cause an interrupt request.

Timer 3 in Counter Mode

Counter mode for the core timer T3 is selected by setting bit field T3M in register T3CON to '001_B'. In counter mode timer T3 is clocked by a transition at the external input pin T3IN. The event causing an increment or decrement of the timer can be a positive, a negative, or both a positive and a negative transition at this pin. Bit field T3I in control register T3CON selects the triggering transition (see **Table 10-5**).

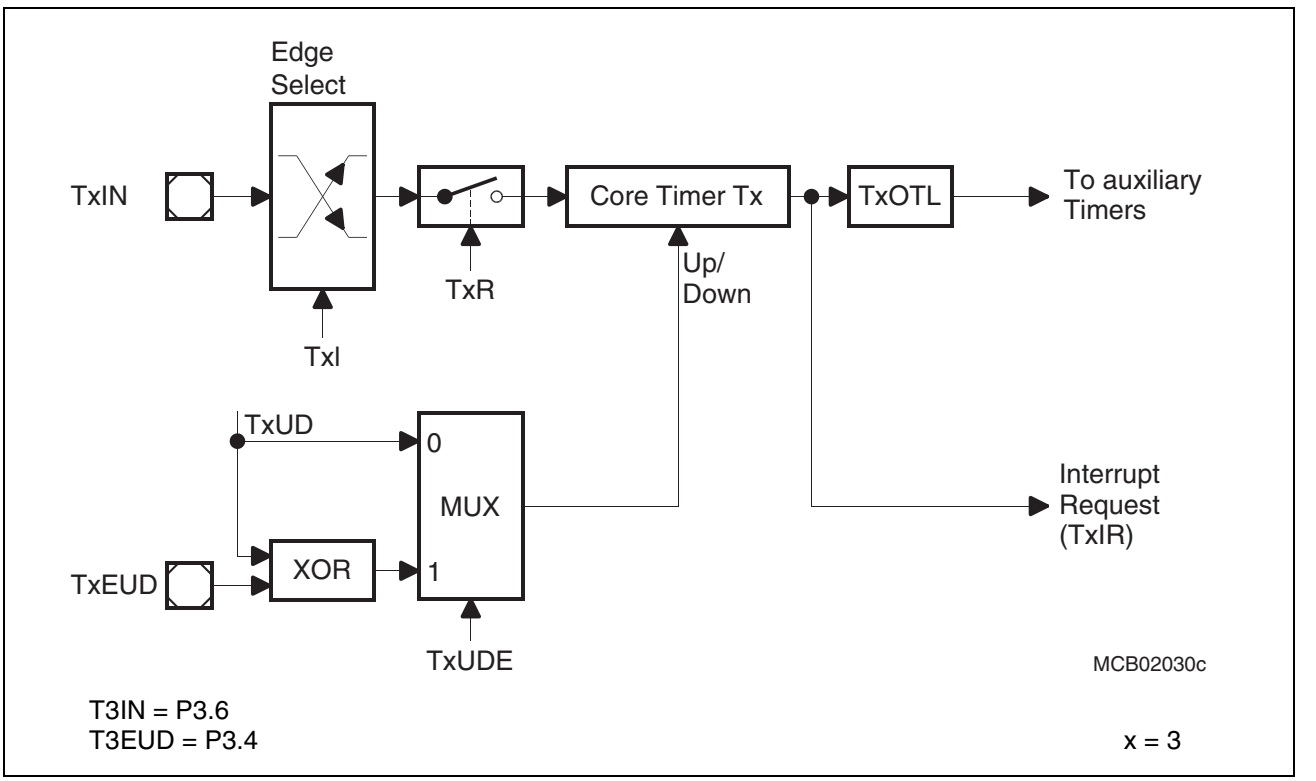


Figure 10-5 Block Diagram of Core Timer T3 in Counter Mode

Table 10-5 GPT1 Core Timer T3 (Counter Mode) Input Edge Selection

T3I	Triggering Edge for Counter Increment/Decrement
0 0 0	None. Counter T3 is disabled
0 0 1	Positive transition (rising edge) on T3IN
0 1 0	Negative transition (falling edge) on T3IN
0 1 1	Any transition (rising or falling edge) on T3IN
1 X X	Reserved. Do not use this combination

For counter operation, pin T3IN must be configured as input; the respective direction control bit DPx.y must be set to '0'. The maximum input frequency allowed in counter mode is $f_{CPU}/16$. To ensure that a transition of the count input signal applied to T3IN is recognized correctly, its level should be held high or low for at least 8 f_{CPU} cycles before it changes.

Timer 3 in Incremental Interface Mode

Incremental Interface Mode for the core timer T3 is selected by setting bit field T3M in register T3CON to '110_B'. In incremental interface mode, the two inputs associated with timer T3 (T3IN, T3EUD) are used to interface to an incremental encoder. T3 is clocked by each transition on one or both of the external input pins to provide 2-fold or 4-fold resolution of the encoder input.

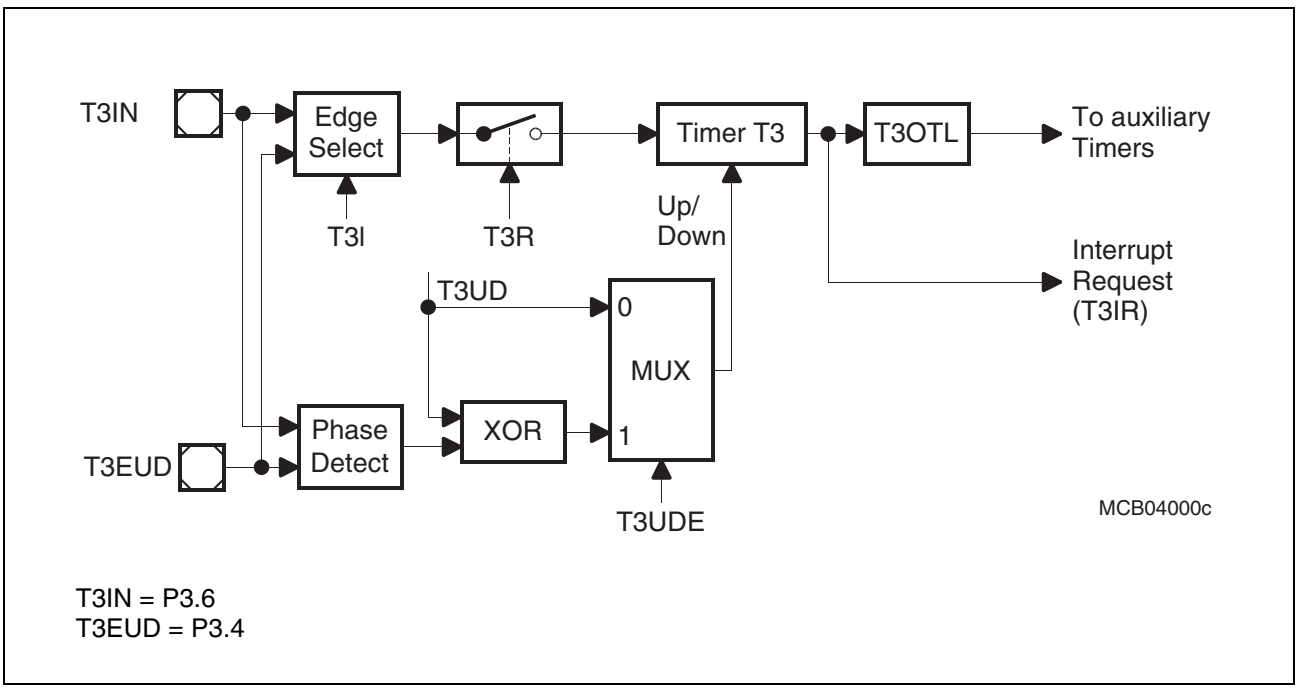


Figure 10-6 Block Diagram of Core Timer T3 in Incremental Interface Mode

Bitfield T3I in control register T3CON selects the triggering transitions (see **Table 10-6**). In this mode, the sequence of the transitions of the two input signals is evaluated and generates count pulses as well as the direction signal. T3 is modified automatically according to the speed and the direction of the incremental encoder and, therefore, its contents therefore always represent the encoder's current position.

Table 10-6 GPT1 Core Timer T3 (Incremental Interface Mode) Input Edge Selection

T3I	Triggering Edge for Counter Increment/Decrement
0 0 0	None. Counter T3 stops.
0 0 1	Any transition (rising or falling edge) on T3IN.
0 1 0	Any transition (rising or falling edge) on T3EUD.
0 1 1	Any transition (rising or falling edge) on any T3 input (T3IN or T3EUD).
1 X X	Reserved. Do not use this combination.

The incremental encoder can be connected directly to the C164CI without external interface logic. In a standard system, however, comparators will be employed to convert the encoder's differential outputs (such as A, $\overline{A}$) to digital signals (such as A). This greatly increases noise immunity.

Note: The third encoder output Top0, which indicates the mechanical zero position, may be connected to an external interrupt input and trigger a reset of timer T3 (for example via PEC transfer from ZEROS).

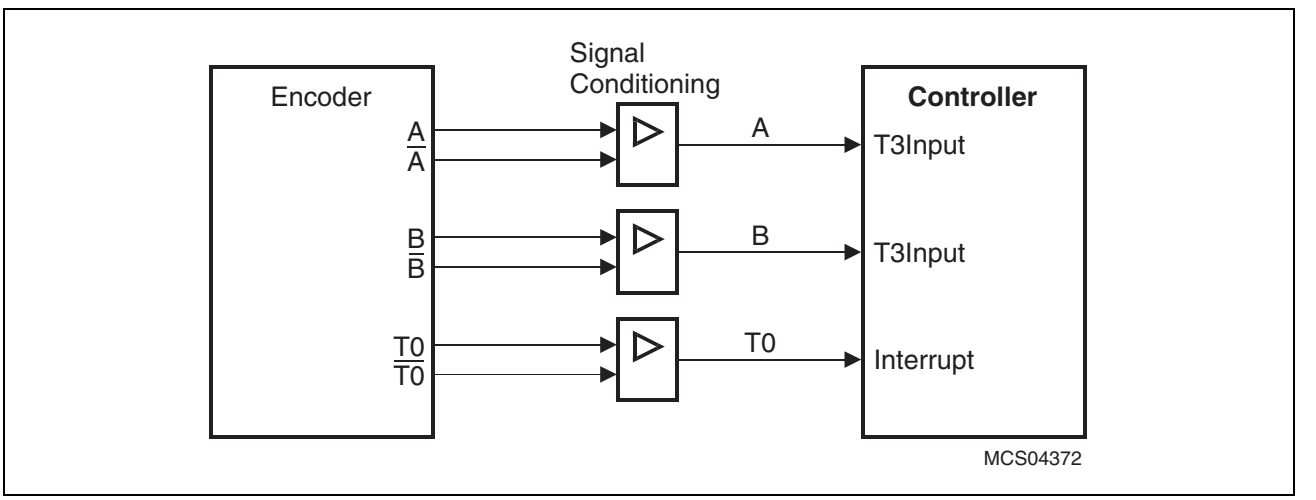


Figure 10-7 Connection of the Encoder to the C164CI

For incremental interface operation, the following conditions must be met:

- Bitfield T3M must be '110_B'.
- Both pins T3IN and T3EUD must be configured as input. e.g. the respective direction control bits must be '0'.
- Bit T3UDE must be '1' to enable automatic direction control.

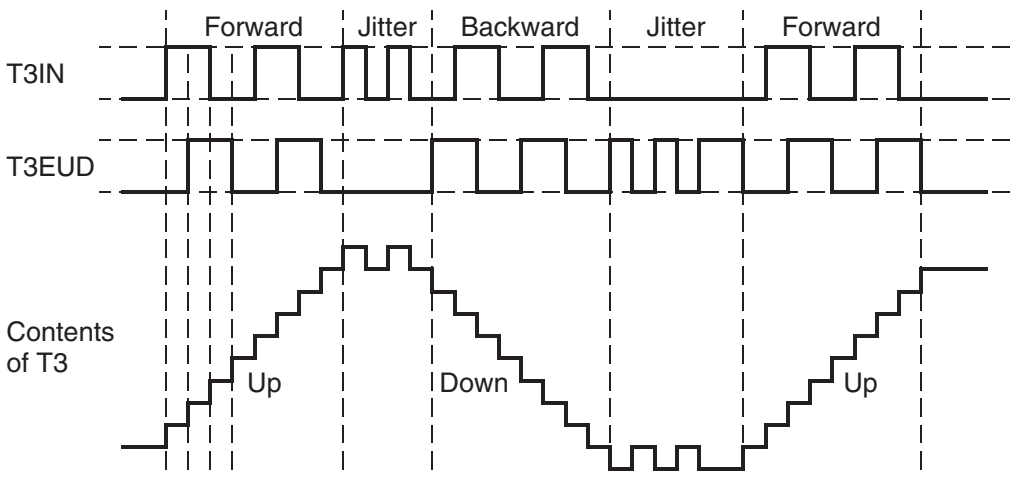
The maximum input frequency allowed in incremental interface mode is $f_{CPU}/16$. To ensure that a transition of any input signal is recognized correctly, its level should be held high or low for at least 8 f_{CPU} cycles before it changes. As in Incremental Interface Mode two input signals with a 90° phase shift are evaluated, their maximum input frequency can be $f_{CPU}/32$.

In Incremental Interface Mode, the count direction is automatically derived from the sequence in which the input signals change, which corresponds to the rotation direction of the connected sensor. **Table 10-7** summarizes the possible combinations.

Table 10-7 GPT1 Core Timer T3 (Incremental Interface Mode) Count Direction

Level on respective other input	T3IN Input		T3EUD Input	
	Rising ↗	Falling ↘	Rising ↗	Falling ↘
High	Down	Up	Up	Down
Low	Up	Down	Down	Up

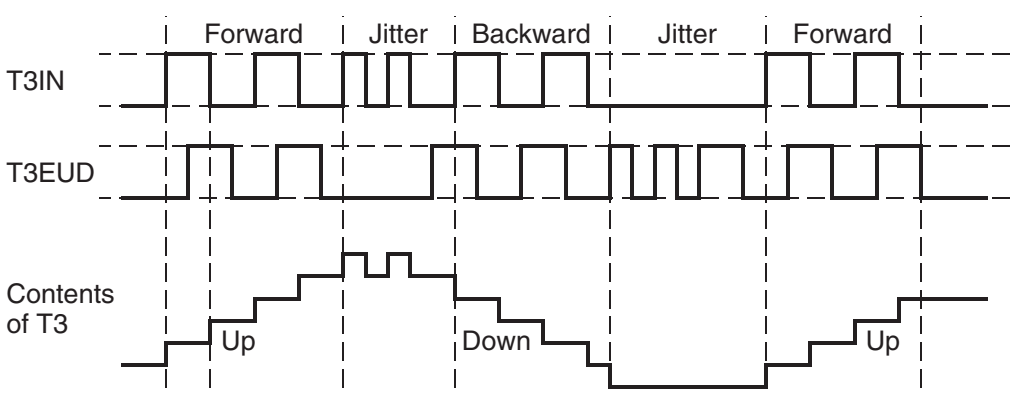
Figure 10-8 and **Figure 10-9** give examples of T3's operation, visualizing count signal generation, and direction control. They also show how input jitter is compensated, which might occur if the sensor rests near to one of its switching points.



Note: This example shows the timer behaviour assuming that T3 counts upon any transition on input, i.e. T3I = '011 $_B$'.

MCT04373

Figure 10-8 Evaluation of the Incremental Encoder Signals



Note: This example shows the timer behaviour assuming that T3 counts upon any transition on input T3IN, i.e. T3I = '001 $_B$'.

MCT04374

Figure 10-9 Evaluation of the Incremental Encoder Signals

Note: Timer T3 operating in incremental interface mode automatically provides information about the sensor's current position. Dynamic information (speed, acceleration, deceleration) may be obtained by measuring the incoming signal periods.

10.1.2 GPT1 Auxiliary Timers T2 and T4

Auxiliary timers T2 and T4 have exactly the same functionality. They can be configured for timer, gated timer, counter, or incremental interface mode with the same options for the timer frequencies and the count signal as the core timer T3. In addition to these 4 counting modes, the auxiliary timers can be concatenated with the core timer, or they may be used as reload or capture registers in conjunction with the core timer.

The individual configurations for timers T2 and T4 are determined by their bitaddressable control registers T2CON and T4CON, which are organized identically. Note that functions present in all 3 timers of block GPT1 are controlled in the same bit positions and in the same manner in each of the specific control registers.

Note: The auxiliary timers have no output toggle latch and no alternate output function.

T2CON
Timer 2 Control Register **SFR ($FF40_H/A0_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	T2UDE	T2UD	T2R	T2M			T2I		
-	-	-	-	-	-	-	rw	rw	rw	rw			rw		

T4CON
Timer 4 Control Register **SFR ($FF44_H/A2_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	T4UDE	T4UD	T4R	T4M			T4I		
-	-	-	-	-	-	-	rw	rw	rw	rw			rw		

Bit	Function
TxI	**Timer x Input Selection** Depends on the Operating Mode, see respective sections.
TxM	**Timer x Mode Control** (Basic Operating Mode) 000: Timer Mode 001: Counter Mode 010: Gated Timer with Gate active low 011: Gated Timer with Gate active high 100: Reload Mode 101: Capture Mode 110: Incremental Interface Mode 111: *Reserved*. Do not use this combination.
TxR	**Timer x Run Bit** 0: Timer/Counter x stops 1: Timer/Counter x runs
TxUD	**Timer x Up/Down Control**[1)]
TxUDE	**Timer x External Up/Down Enable**[1)]

1) For the effects of bits TxUD and TxUDE refer to **Table 10-1** (see T3 section).

Count Direction Control for Auxiliary Timers

The count direction of the auxiliary timers can be controlled in the same way as for the core timer T3. The description and the table apply accordingly.

Timers T2 and T4 in Timer Mode or Gated Timer Mode

When the auxiliary timers T2 and T4 are programmed to timer mode or gated timer mode, their operation is the same as described for the core timer T3. The descriptions, figures, and tables apply accordingly with one exception: There is no output toggle latch for T2 and T4.

Timers T2 and T4 in Incremental Interface Mode

When the auxiliary timers T2 and T4 are programmed to incremental interface mode, their operation is the same as described for the core timer T3. The descriptions, figures, and tables apply accordingly.

Timers T2 and T4 in Counter Mode

Counter mode for the auxiliary timers T2 and T4 is selected by setting bit field TxM in the respective register TxCON to '001_B'. In counter mode, timers T2 and T4 can be clocked either by a transition at the respective external input pin TxIN, or by a transition of timer T3's output toggle latch T3OTL.

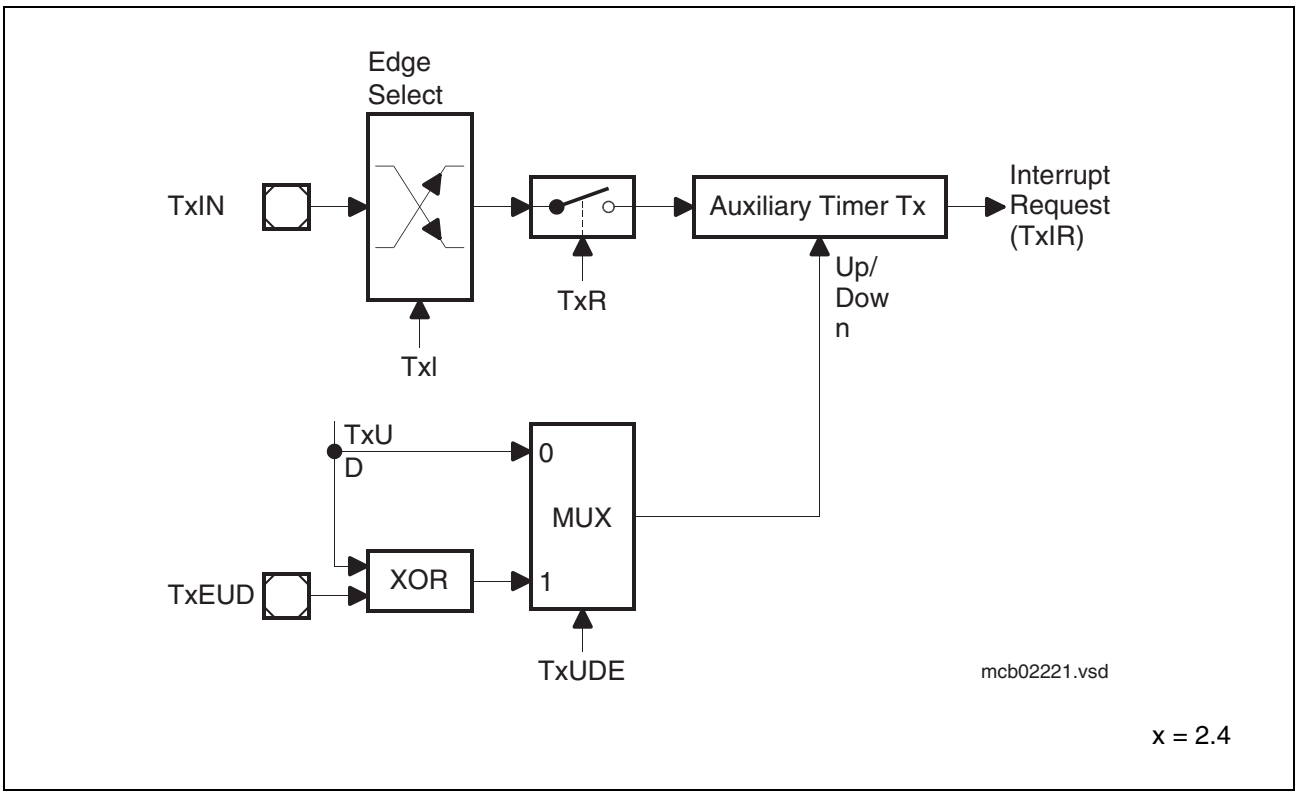


Figure 10-10 Block Diagram of an Auxiliary Timer in Counter Mode

The event causing an increment or decrement of a timer can be a positive, a negative, or both a positive and a negative transition at either the respective input pin or at the toggle latch T3OTL.
Bit field TxI in the respective control register TxCON selects the triggering transition (see **Table 10-8**).

Table 10-8 GPT1 Auxiliary Timer (Counter Mode) Input Edge Selection

T2I/T4I	Triggering Edge for Counter Increment/Decrement
X 0 0	None. Counter Tx is disabled
0 0 1	Positive transition (rising edge) on TxIN
0 1 0	Negative transition (falling edge) on TxIN
0 1 1	Any transition (rising or falling edge) on TxIN
1 0 1	Positive transition (rising edge) of output toggle latch T3OTL
1 1 0	Negative transition (falling edge) of output toggle latch T3OTL
1 1 1	Any transition (rising or falling edge) of output toggle latch T3OTL

Note: Only state transitions of T3OTL caused by the overflows/underflows of T3 will trigger the counter function of T2/T4. Modifications of T3OTL via software will NOT trigger the counter function of T2/T4.

For counter operation, pin TxIN must be configured as an input; the respective direction control bit must be ‘0’. The maximum input frequency allowed in counter mode is f_{CPU}/16. To ensure that a transition of the count input signal which is applied to TxIN is recognized correctly, its level should be held for at least 8 f_{CPU} cycles before it changes.

Timer Concatenation

Using the toggle bit T3OTL as a clock source for an auxiliary timer in counter mode concatenates the core timer T3 with the respective auxiliary timer. This concatenation forms either a 32-bit or a 33-bit timer/counter, depending on which transition of T3OTL is selected to clock the auxiliary timer.

- **32-bit Timer/Counter**: If both a positive and a negative transition of T3OTL are used to clock the auxiliary timer, this timer is clocked on every overflow/underflow of the core timer T3. Thus, the two timers form a 32-bit timer.
- **33-bit Timer/Counter**: If either a positive or a negative transition of T3OTL is selected to clock the auxiliary timer, this timer is clocked on every second overflow/underflow of the core timer T3. This configuration forms a 33-bit timer (16-bit core timer + T3OTL + 16-bit auxiliary timer).

The count directions of the two concatenated timers are not required to be the same. This offers a wide variety of different configurations.
T3 can operate in timer, gated timer or counter mode in this case.

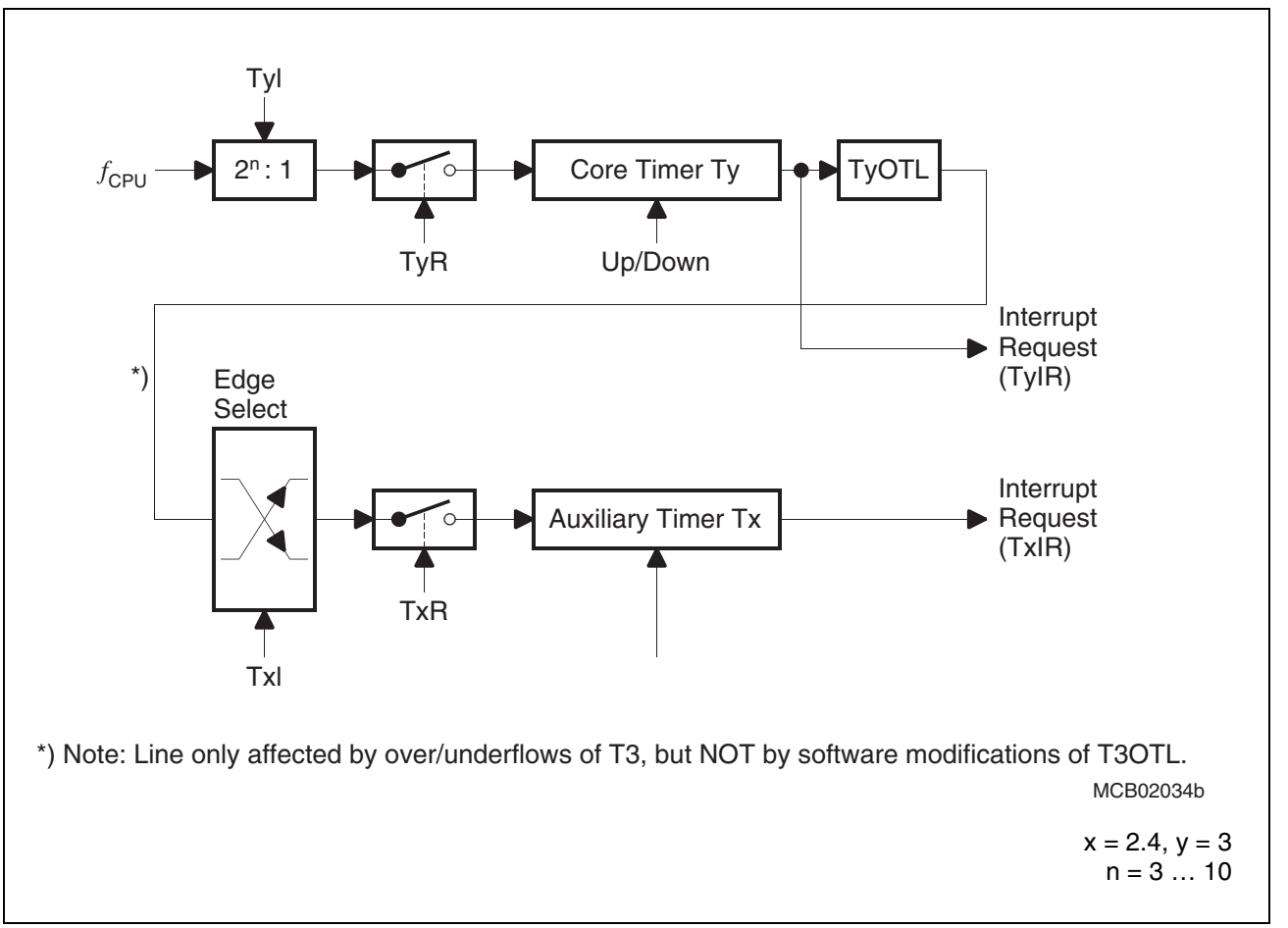


Figure 10-11 Concatenation of Core Timer T3 and an Auxiliary Timer

Auxiliary Timer in Reload Mode

Reload mode for the auxiliary timers T2 and T4 is selected by setting bit field TxM in the respective register TxCON to '100_B'. In reload mode, the core timer T3 is reloaded with the contents of an auxiliary timer register, triggered by one of two different signals. The trigger signal is selected the same way as the clock source for counter mode (see **Table 10-8**), i.e. a transition of the auxiliary timer's input or the output toggle latch T3OTL may trigger the reload.

Note: When programmed for reload mode, the respective auxiliary timer (T2 or T4) stops independently of its run flag T2R or T4R.

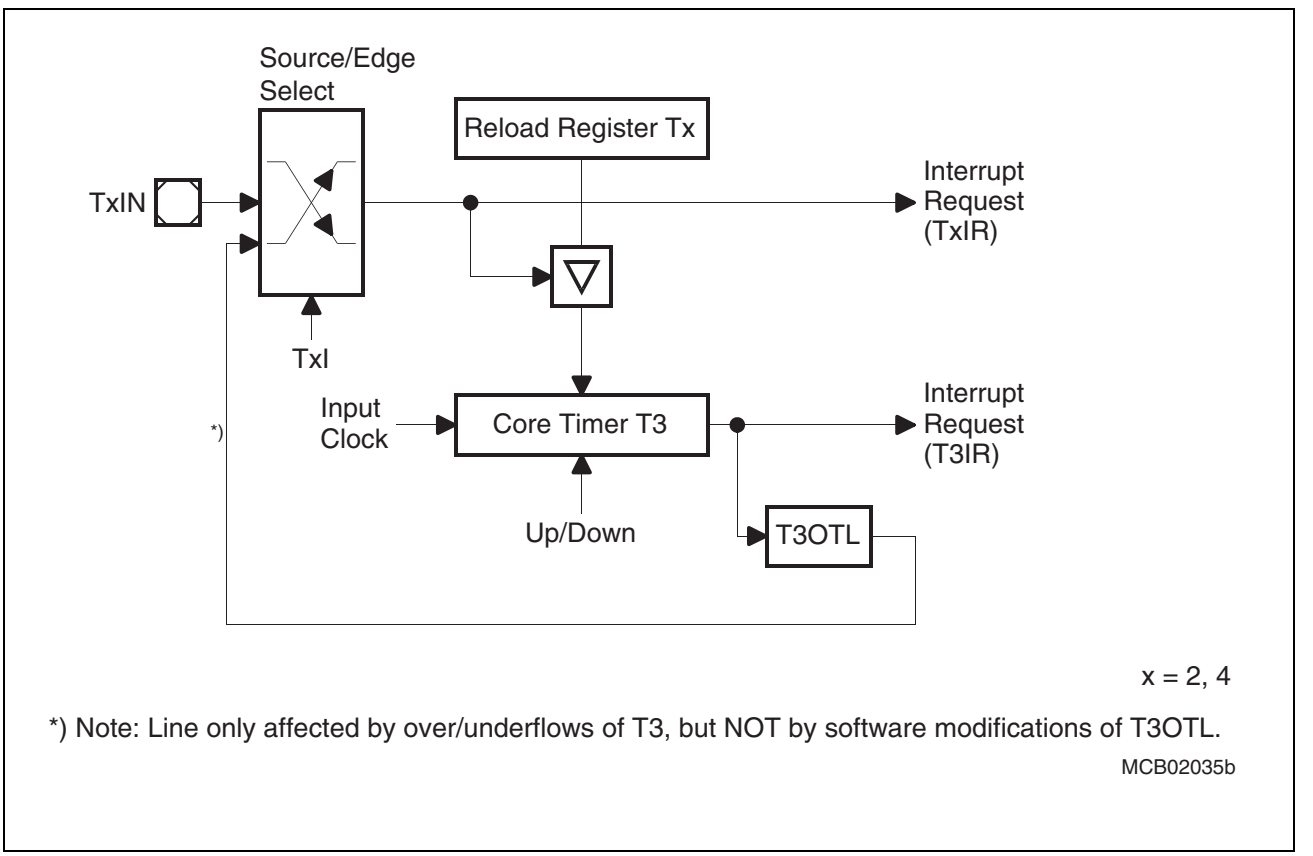


Figure 10-12 GPT1 Auxiliary Timer in Reload Mode

Upon a trigger signal, T3 is loaded with the contents of the respective timer register (T2 or T4) and the interrupt request flag (T2IR or T4IR) is set.

Note: When a T3OTL transition is selected for the trigger signal, the interrupt request flag T3IR will also be set upon a trigger, indicating T3's overflow or underflow. Modifications of T3OTL via software will NOT trigger the counter function of T2/T4.

The reload mode triggered by T3OTL can be used in a number of different configurations. The following functions can be performed, depending on the selected active transition:

- If both a positive and a negative transition of T3OTL are selected to trigger a reload, the core timer will be reloaded with the contents of the auxiliary timer each time it overflows or underflows. This is the standard reload mode (reload on overflow/ underflow).
- If either a positive or a negative transition of T3OTL is selected to trigger a reload, the core timer will be reloaded with the contents of the auxiliary timer on every second overflow or underflow.
- Using this “single-transition” mode for both auxiliary timers allows very flexible Pulse Width Modulation (PWM). One of the auxiliary timers is programmed to reload the core timer on a positive transition of T3OTL, the other is programmed for a reload on a negative transition of T3OTL. With this combination, the core timer is alternately reloaded from the two auxiliary timers.

Note: Although possible, selecting the same reload trigger event for both auxiliary timers should be avoided. In such a case, both reload registers would try to load the core timer at the same time. If this combination is selected, T2 is disregarded and the contents of T4 is reloaded.

Auxiliary Timer in Capture Mode

Capture mode for the auxiliary timers T2 and T4 is selected by setting bit field TxM in the respective register TxCON to '101_B'. In capture mode, the contents of the core timer are latched into an auxiliary timer register in response to a signal transition at the respective auxiliary timer's external input pin TxIN. The capture trigger signal can be a positive, a negative, or both a positive and a negative transition.

The two least significant bits of bit field TxI are used to select the active transition (see **Table 10-8**), while the most significant bit TxI.2 is irrelevant for capture mode. It is recommended to keep this bit cleared (TxI.2 = '0').

Note: When programmed for capture mode, the respective auxiliary timer (T2 or T4) stops independently of its run flag T2R or T4R.

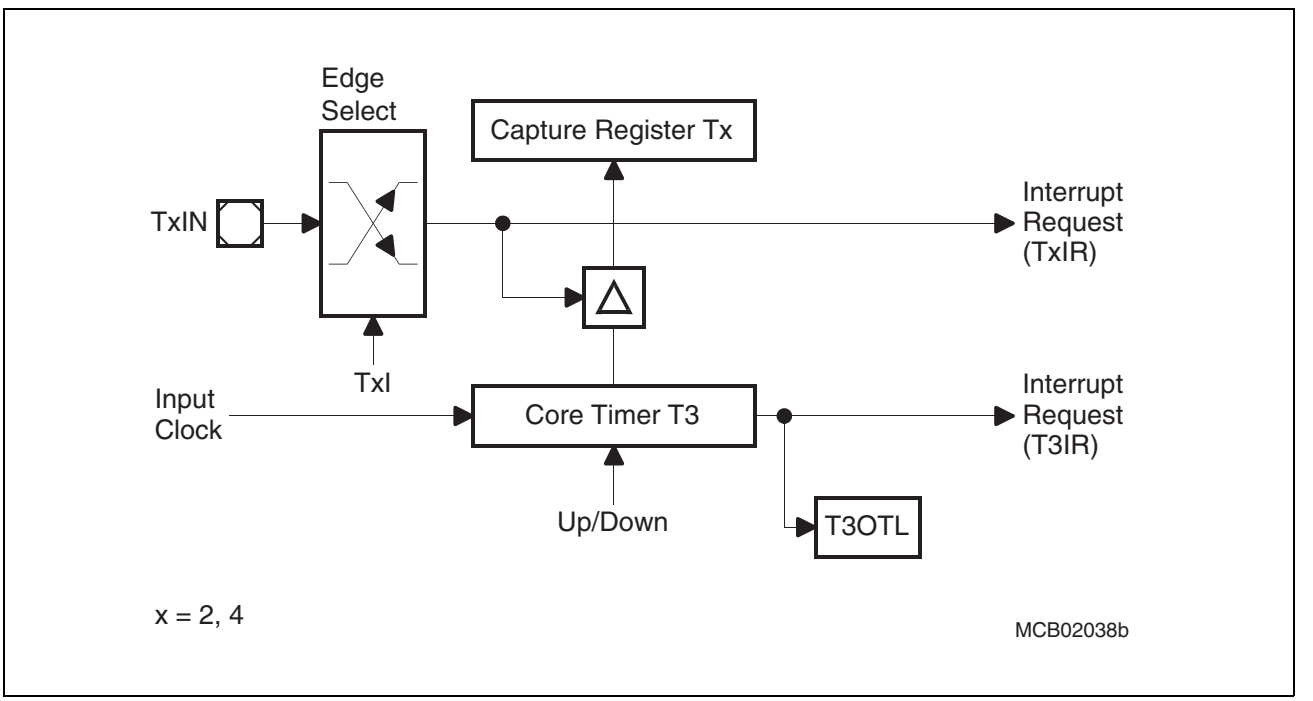


Figure 10-13 GPT1 Auxiliary Timer in Capture Mode

Upon a trigger (selected transition) at the corresponding input pin TxIN the contents of the core timer are loaded into the auxiliary timer register and the associated interrupt request flag TxIR will be set.

Note: To ensure correct edge detection, the direction control bits for T2IN and T4IN must be set to '0' and the level of the capture trigger signal should be held high or low for at least 8 f_{CPU} cycles before it changes.

10.1.3 Interrupt Control for GPT1 Timers

When a timer overflows from $FFFF_H$ to 0000_H (counting up), or when it underflows from 0000_H to $FFFF_H$ (counting down), its interrupt request flag (T2IR, T3IR or T4IR) in register TxIC will be set. This will cause an interrupt to the respective timer interrupt vector (T2INT, T3INT or T4INT) or will trigger a PEC service, if the respective interrupt enable bit (T2IE, T3IE or T4IE in register TxIC) is set. There is an interrupt control register for each of the three timers.

T2IC
Timer 2 Intr. Ctrl. Reg. **SFR ($FF60_H/B0_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								T2IR	T2IE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

T3IC
Timer 3 Intr. Ctrl. Reg. **SFR ($FF62_H/B1_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								T3IR	T3IE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

T4IC
Timer 4 Intr. Ctrl. Reg. **SFR ($FF64_H/B2_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								T4IR	T4IE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

11 Asynchronous/Synchronous Serial Interface

The Asynchronous/Synchronous Serial Interface ASC0 provides serial communication between the C164CI and other microcontrollers, microprocessors, or external peripherals.

The ASC0 supports both full-duplex asynchronous communication and half-duplex synchronous communication (for baud rate ranges see formulas and tables in **Section 11.4**). In synchronous mode, data are transmitted or received synchronous to a shift clock generated by the C164CI. In asynchronous mode, selection of 8- or 9-bit data transfer, parity generation, and the number of stop bits can be made. Parity, framing, and overrun error detection are provided to increase the reliability of data transfers. Transmission and reception of data are double-buffered. For multiprocessor communication, a mechanism is provided to distinguish address bytes from data bytes. Testing is supported by a loop-back option. A 13-bit baud rate generator provides the ASC0 with a separate serial clock signal.

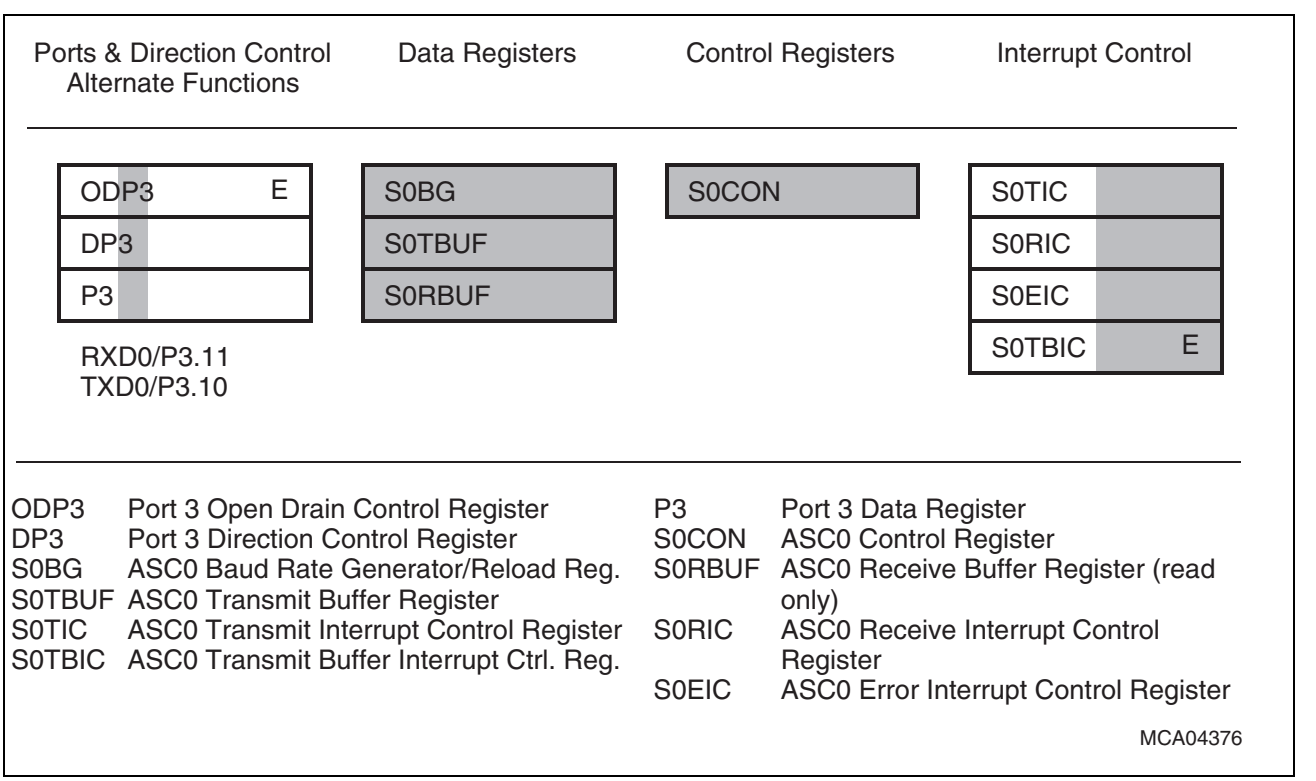


Figure 11-1 SFRs and Port Pins Associated with ASC0

The operating mode of the serial channel ASC0 is controlled by its bit-addressable control register S0CON. This register contains control bits for mode and error check selection, as well as status flags for error identification.

S0CON
ASC0 Control Register **SFR ($FFB0_H/D8_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S0R	S0 LB	S0 BRS	S0 ODD	-	S0 OE	S0 FE	S0 PE	S0 OEN	S0 FEN	S0 PEN	S0 REN	S0 STP	S0M		
rw	rw	rw	rw	-	rwh	rwh	rwh	rw	rw	rw	rwh	rw	rw		

Bit	Function
S0M	**ASC0 Mode Control** 000: 8-bit data synchronous operation 001: 8-bit data asynchronous operation 010: Reserved. Do not use this combination! 011: 7-bit data + parity asynchronous operation 100: 9-bit data asynchronous operation 101: 8-bit data + wake up bit asynchronous operation 110: Reserved. Do not use this combination! 111: 8-bit data + parity asynchronous operation
S0STP	**Number of Stop Bits Selection** asynchronous operation 0: One stop bit 1: Two stop bits
S0REN	**Receiver Enable Bit** 0: Receiver disabled 1: Receiver enabled (Reset by hardware after reception of byte in synchronous mode)
S0PEN	**Parity Check Enable Bit** asynchronous operation 0: Ignore parity 1: Check parity
S0FEN	**Framing Check Enable Bit** asynchronous operation 0: Ignore framing errors 1: Check framing errors
S0OEN	**Overrun Check Enable Bit** 0: Ignore overrun errors 1: Check overrun errors

Bit	Function
S0PE	**Parity Error Flag** Set by hardware on a parity error (S0PEN = ‘1’). Must be reset by software.
S0FE	**Framing Error Flag** Set by hardware on a framing error (S0FEN = ‘1’). Must be reset by software.
S0OE	**Overrun Error Flag** Set by hardware on an overrun error (S0OEN = ‘1’). Must be reset by software.
S0ODD	**Parity Selection Bit** 0: Even parity (parity bit set on odd number of ‘1’s in data) 1: Odd parity (parity bit set on even number of ‘1’s in data)
S0BRS	**Baudrate Selection Bit** 0: Divide clock by reload-value + constant (depending on mode) 1: Additionally reduce serial clock to 2/3
S0LB	**Loopback Mode Enable Bit** 0: Standard transmit/receive mode 1: Loopback mode enabled
S0R	**Baudrate Generator Run Bit** 0: Baudrate generator disabled (ASC0 inactive) 1: Baudrate generator enabled

A transmission is started by writing to the Transmit Buffer register S0TBUF (via an instruction or a PEC data transfer). The number of data bits to be actually transmitted is determined by the operating mode selected; that is, bits written to positions 9 through 15 of register S0TBUF are always insignificant. After a transmission has been completed, the transmit buffer register is cleared to 0000_H.

Data transmission is double-buffered so that a new character may be written to the transmit buffer register before the transmission of the previous character is complete. This allows the transmission of characters back-to-back without gaps.

Data reception is enabled by the Receiver Enable Bit S0REN. After reception of a character has been completed, the received data and if provided by the selected operating mode the received parity bit can be read from the (read-only) Receive Buffer register S0RBUF. Bits in the upper half of S0RBUF not valid in the selected operating mode will be read as zeros.

Data reception is double-buffered so that reception of a second character may begin before the previously received character has been read out of the receive buffer register. In all modes, receive buffer overrun error detection can be selected through bit S0OEN.

When enabled, the overrun error status flag S0OE and the error interrupt request flag S0EIR will be set if the receive buffer register has not been read by the time reception of a second character is complete. The previously received character in the receive buffer is overwritten.

The Loop-Back option (selected by bit S0LB) allows the data currently being transmitted to be received simultaneously in the receive buffer. This may be used to test serial communication routines at an early stage without having to provide an external network. In loop-back mode, the alternate input/output functions of the Port 3 pins are not necessary.

Note: Serial data transmission or reception is only possible when the Baud Rate Generator Run Bit S0R is set to '1'. Otherwise, the serial interface is idle.
To avoid unpredictable behavior of the serial interface do not program the mode control field S0M in register S0CON to one of the reserved combinations.

11.1 Asynchronous Operation

Asynchronous mode supports full-duplex communication in which both transmitter and receiver use the same data frame format and the same baud rate. Data is transmitted on pin TXD0 and received on pin RXD0. These signals are alternate port functions.

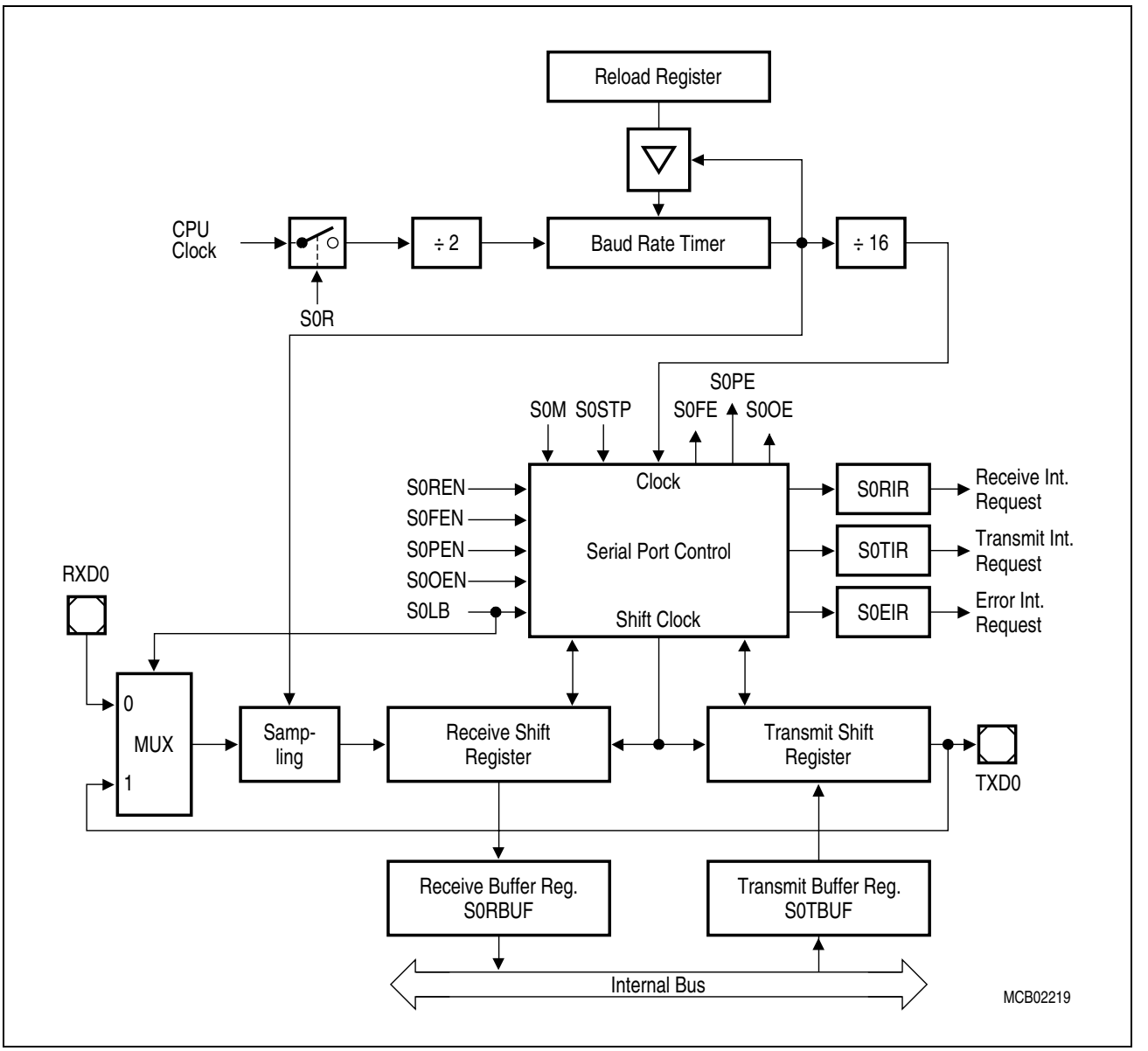


Figure 11-2 Asynchronous Mode of Serial Channel ASC0

Asynchronous Data Frames

8-bit data frames consist of either 8 data bits D7 … D0 (S0M = '001_B'), or 7 data bits D6 … D0 plus an automatically generated parity bit (S0M = '011_B'). Parity may be odd or even, depending on bit S0ODD in register S0CON. An even parity bit will be set, if the modulo-2-sum of the 7 data bits is '1'. An odd parity bit will be cleared in this case. Parity checking is enabled via bit S0PEN (always OFF in 8-bit data mode). The parity error flag

S0PE will be set along with the error interrupt request flag, if a wrong parity bit is received. The parity bit itself will be stored in bit S0RBUF.7.

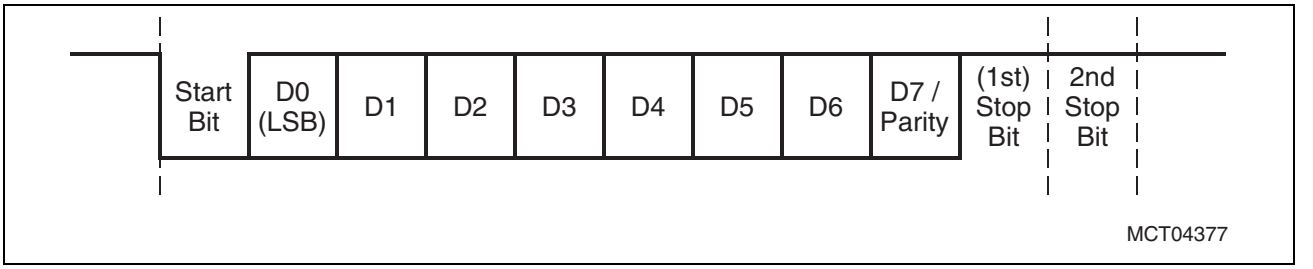


Figure 11-3 Asynchronous 8-bit Data Frames

9-bit data frames consist of either 9 data bits D8 … D0 (S0M = '100_B'), 8 data bits D7 … D0 plus an automatically generated parity bit (S0M = '111_B'), or 8 data bits D7 … D0 plus a wake-up bit (S0M = '101_B'). Parity may be odd or even, depending on bit S0ODD in register S0CON. An even parity bit will be set, if the modulo-2-sum of the 8 data bits is '1'. An odd parity bit will be cleared in this case. Parity checking is enabled via bit S0PEN (always OFF in 9-bit data and wake-up modes). The parity error flag S0PE will be set along with the error interrupt request flag if a wrong parity bit is received. The parity bit itself will be stored in bit S0RBUF.8.

In wake-up mode, received frames are transferred to the receive buffer register only if the 9^{th} bit (the wake-up bit) is '1'. If this bit is '0', no receive interrupt request will be activated and no data will be transferred.

This feature may be used to control communication in a multi-processor system: When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address byte which identifies the target slave. An address byte differs from a data byte in that the additional 9^{th} bit is a '1' for an address byte and a '0' for a data byte, so that no slave will be interrupted by a data 'byte'. An address 'byte' will interrupt all slaves (operating in 8-bit data + wake-up bit modes), so each slave can examine the 8 LSBs of the received character (the address). The addressed slave will switch to 9-bit data mode (for example, by clearing bit S0M.0), which enables it to also receive the data bytes that will be coming (having the wake-up bit cleared). The slaves not being addressed remain in 8-bit data + wake-up bit modes, ignoring the following data bytes.

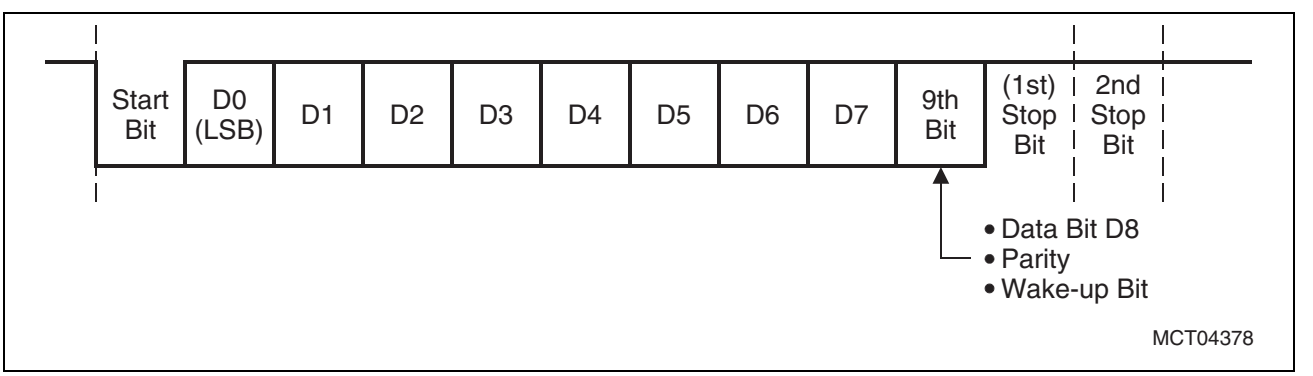


Figure 11-4 Asynchronous 9-bit Data Frames

Asynchronous transmission begins at the next overflow of the divide-by-16 counter (see **Figure 11-4**), provided that S0R is set and data has been loaded into S0TBUF. The transmitted data frame consists of three basic elements:

- the start bit
- the data field (8 or 9 bits, LSB first, including a parity bit, if selected)
- the delimiter (1 or 2 stop bits)

Data transmission is double-buffered. When the transmitter is idle, the transmit data loaded into S0TBUF is immediately moved to the transmit shift register thus freeing S0TBUF for the next data to be sent. This is indicated by the transmit buffer interrupt request flag S0TBIR being set. S0TBUF may now be loaded with the next data, while transmission of the previous one is still going on.

The transmit interrupt request flag S0TIR will be set before the last bit of a frame is transmitted, that is, before the first or the second stop bit is shifted out of the transmit shift register.
The transmitter output pin TXD0 must be configured for alternate data output, that is, the respective port output latch and the direction latch must be ‘1’.

Asynchronous reception is initiated by a falling edge (1-to-0 transition) on pin RXD0, provided that bits S0R and S0REN are set. The receive data input pin RXD0 is sampled at 16 times the rate of the selected baud rate. A majority decision of the 7^{th}, 8^{th}, and 9^{th} sample determines the effective bit value. This avoids erroneous results that may be caused by noise.

If the detected value is not a ‘0’ when the start bit is sampled, the receive circuit is reset and waits for the next 1-to-0 transition at pin RXD0. If the start bit proves valid, the receive circuit continues sampling and shifts the incoming data frame into the receive shift register.

When the last stop bit has been received, the content of the receive shift register is transferred to the receive data buffer register S0RBUF. Simultaneously, the receive interrupt request flag S0RIR is set after the 9^{th} sample in the last stop bit time slot (as programmed), whether or not valid stop bits have been received. The receive circuit then waits for the next start bit (1-to-0 transition) at the receive data input pin.

The receiver input pin RXD0 must be configured for input, that is the respective direction latch must be ‘0’.

Asynchronous reception is stopped by clearing bit S0REN. A frame currently being received is completed including the generation of the receive interrupt request and an error interrupt request, if appropriate. Start bits following this frame will not be recognized.

Note: In wake-up mode, received frames are transferred to the receive buffer register only if the 9^{th} bit (the wake-up bit) is ‘1’. If this bit is ‘0’, no receive interrupt request will be activated and no data will be transferred.

11.2 Synchronous Operation

Synchronous mode supports half-duplex communication, primarily for simple IO expansion via shift registers. Data is transmitted and received via pin RXD0, while pin TXD0 outputs the shift clock. These signals are alternate port functions. Synchronous mode is selected with S0M = '000_B'.

8 data bits are transmitted or received synchronous to a shift clock generated by the internal baud rate generator. The shift clock is active only as long as data bits are transmitted or received.

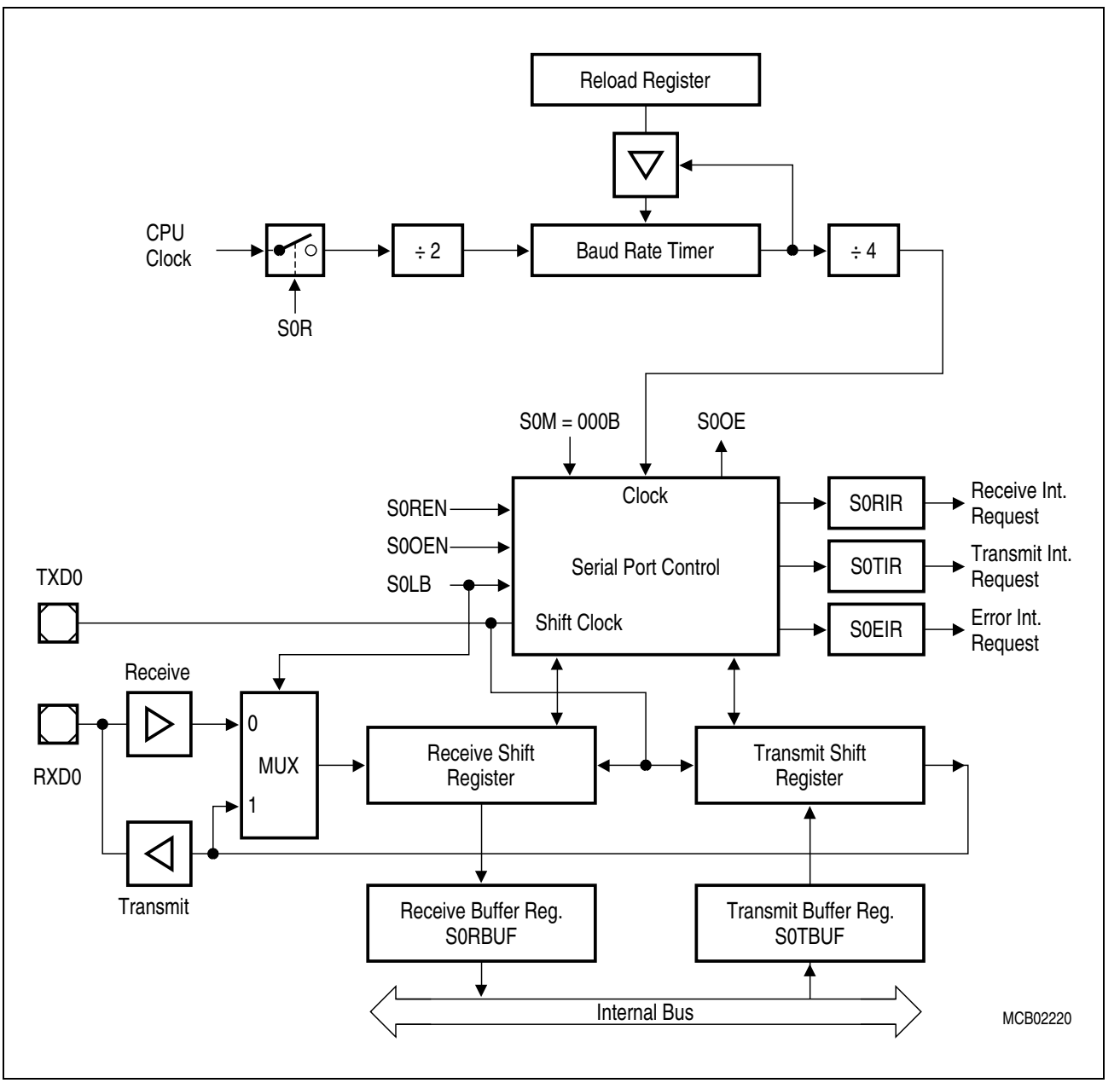


Figure 11-5 Synchronous Mode of Serial Channel ASC0

Synchronous transmission begins within 4 state times after data has been loaded into S0TBUF, provided that S0R is set and S0REN = ‘0’ (half-duplex, no reception). Data transmission is double-buffered. When the transmitter is idle, the transmit data loaded into S0TBUF is immediately moved to the transmit shift register, thus, freeing S0TBUF for the next data to be sent. This is indicated by the transmit buffer interrupt request flag S0TBIR being set. S0TBUF may now be loaded with the next data, while transmission of the previous data is still going on. The data bits are transmitted synchronous with the shift clock. After the bit time for the 8th data bit, both pins TXD0 and RXD0 will go high, the transmit interrupt request flag S0TIR is set, and serial data transmission stops.

Pin TXD0 must be configured for alternate data output, that is the respective port output latch and the direction latch must be ‘1’, in order to provide the shift clock. Pin RXD0 must also be configured for output (output/direction latch = ‘1’) during transmission.

Synchronous reception is initiated by setting bit S0REN = ‘1’. If bit S0R = ‘1’, the data applied at pin RXD0 are clocked into the receive shift register synchronous to the clock output at pin TXD0. After the 8th bit has been shifted in, the content of the receive shift register is transferred to the receive data buffer S0RBUF, the receive interrupt request flag S0RIR is set, the receiver enable bit S0REN is reset, and serial data reception stops.

Pin TXD0 must be configured for alternate data output, that is, the respective port output latch and the direction latch must be ‘1’, in order to provide the shift clock. Pin RXD0 must be configured as alternate data input, that is, the respective direction latch must be ‘0’.

Synchronous reception is stopped by clearing bit S0REN. A byte currently being received is completed including generation of the receive interrupt request and an error interrupt request, if appropriate. Writing to the transmit buffer register while a reception is in progress has no effect on reception and will not start a transmission.

If a previously received byte has not been read out of the receive buffer register at the time that reception of the next byte is complete, both the error interrupt request flag S0EIR and the overrun error status flag S0OE will be set, provided the overrun check has been enabled by bit S0OEN.

11.3 Hardware Error Detection Capabilities

To enhance reliability of serial data exchange, the serial channel ASC0 provides an error interrupt request flag, which indicates the presence of an error, and three (selectable) error status flags in register S0CON, which indicate which error has been detected during reception. Upon completion of a reception, the error interrupt request flag S0EIR will be set simultaneously with the receive interrupt request flag S0RIR, if one or more of the following conditions are met:

- If the framing error detection enable bit S0FEN is set and any expected stop bit is not high, the framing error flag S0FE is set, indicating that the error interrupt request is due to a framing error (Asynchronous mode only).
- If the parity error detection enable bit S0PEN is set in the modes in which a parity bit is received, and the parity check on the received data bits proves false, the parity error flag S0PE is set, indicating that the error interrupt request is due to a parity error (Asynchronous mode only).
- If the overrun error detection enable bit S0OEN is set and the last character received was not read out of the receive buffer by software or by PEC transfer at the time the reception of a new frame is complete, the overrun error flag S0OE is set, indicating that the error interrupt request is due to an overrun error (Asynchronous and Synchronous modes).

11.4 ASC0 Baud Rate Generation

The serial channel ASC0 has its own dedicated 13-bit baud rate generator with 13-bit reload capability, allowing baud rate generation independent of the GPT timers.

The baud rate generator is clocked with the CPU clock divided by 2 ($f_{CPU}/2$). The timer counts downwards and can be started or stopped through the Baud Rate Generator Run Bit S0R in register S0CON. Each underflow of the timer provides one clock pulse to the serial channel. The timer is reloaded with the value stored in its 13-bit reload register each time it underflows. The resulting clock is again divided according to the operating mode and controlled by the Baudrate Selection Bit S0BRS. If S0BRS = '1', the clock signal is additionally divided to 2/3rd of its frequency (see formulas and table). So the baud rate of ASC0 is determined by the CPU clock, the reload value, the value of S0BRS and the operating mode (asynchronous or synchronous).

Register S0BG is the dual-function Baud Rate Generator/Reload register. Reading S0BG returns the content of the timer (bits 15 … 13 return zero), while writing to S0BG always updates the reload register (bits 15 … 13 are insignificant).

Each time S0BG is written to, an auto-reload of the timer with the content of the reload register is performed. However, if S0R = '0' at the time the write operation to S0BG is performed, the timer will not be reloaded until the first instruction cycle after S0R = '1'.

Asynchronous Mode Baud Rates

For asynchronous operation, the baud rate generator provides a clock with 16 times the rate of the established baud rate. Every received bit is sampled at the 7th, 8th and 9th cycle of this clock. The baud rate for asynchronous operation of serial channel ASC0 and the required reload value for a given baudrate can be determined by the following formulas:

$$B_{Async} = \frac{f_{CPU}}{16 \times (2 + \langle S0BRS \rangle) \times (\langle S0BRL \rangle + 1)}$$

$$S0BRL = \left(\frac{f_{CPU}}{16 \times (2 + \langle S0BRS \rangle) \times B_{Async}}\right) - 1$$

<S0BRL> represents the contents of the reload register taken as unsigned 13-bit integer, <S0BRS> represents the value of bit S0BRS (either '0' or '1'), taken as integer.

The tables below list various commonly used baud rates and the required reload values and deviation errors compared to the intended baud rates for a number of CPU frequencies.

Note: The deviation errors given in the tables below are rounded. Using a baudrate crystal (such as 18.432 MHz) will provide correct baud rates without deviation errors.

Table 11-1 ASC0 Asynchronous Baudrate Generation for f_{CPU} = 16 MHz

Baud Rate		S0BRS = '0'		S0BRS = '1'	
		Deviation Error	Reload Value	Deviation Error	Reload Value
500	kbit/s	±0.0%	0000_H	–	–
19.2	kbit/s	+0.2%/ -3.5%	0019_H/$001A_H$	+2.1%/ -3.5%	0010_H/0011_H
9600	bit/s	+0.2%/ -1.7%	0033_H/0034_H	+2.1%/ -0.8%	0021_H/0022_H
4800	bit/s	+0.2%/ -0.8%	0067_H/0068_H	+0.6%/ -0.8%	0044_H/0045_H
2400	bit/s	+0.2%/ -0.3%	$00CF_H$/$00D0_H$	+0.6%/ -0.1%	0089_H/$008A_H$
1200	bit/s	+0.4%/ -0.1%	$019F_H$/$01A0_H$	+0.3%/ -0.1%	0114_H/0115_H
600	bit/s	+0.0%/ -0.1%	0340_H/0341_H	+0.1%/ -0.1%	$022A_H$/$022B_H$
61	bit/s	+0.1%	$1FFF_H$	+0.0%/ -0.0%	$115B_H$/$115C_H$
40	bit/s	–	–	+1.7%	$1FFF_H$

Table 11-2 ASC0 Asynchronous Baudrate Generation for f_{CPU} = 20 MHz

Baud Rate		S0BRS = '0'		S0BRS = '1'	
		Deviation Error	Reload Value	Deviation Error	Reload Value
625	kbit/s	±0.0%	0000_H	–	–
19.2	kbit/s	+1.7%/ -1.4%	$001F_H$/0020_H	+3.3%/ -1.4%	0014_H/0015_H
9600	bit/s	+0.2%/ -1.4%	0040_H/0041_H	+1.0%/ -1.4%	$002A_H$/$002B_H$
4800	bit/s	+0.2%/ -0.6%	0081_H/0082_H	+1.0%/ -0.2%	0055_H/0056_H
2400	bit/s	+0.2%/ -0.2%	0103_H/0104_H	+0.4%/ -0.2%	$00AC_H$/$00AD_H$
1200	bit/s	+0.2%/ -0.4%	0207_H/0208_H	+0.1%/ -0.2%	$015A_H$/$015B_H$
600	bit/s	+0.1%/ -0.0%	0410_H/0411_H	+0.1%/ -0.1%	$02B5_H$/$02B6_H$
75	bit/s	+1.7%	$1FFF_H$	+0.0%/ -0.0%	$15B2_H$/$15B3_H$
50	bit/s	–	–	+1.7%	$1FFF_H$

Table 11-3 ASC0 Asynchronous Baudrate Generation for f_{CPU} = 25 MHz

Baud Rate		S0BRS = '0'		S0BRS = '1'	
		Deviation Error	Reload Value	Deviation Error	Reload Value
781	kbit/s	+0.2%	0000_H	–	–
19.2	kbit/s	+1.7%/ -0.8%	0027_H/0028_H	+0.5%/ -3.1%	$001A_H$/$001B_H$
9600	bit/s	+0.5%/ -0.8%	0050_H/0051_H	+0.5%/ -1.4%	0035_H/0036_H
4800	bit/s	+0.5%/ -0.2%	$00A1_H$/$00A2_H$	+0.5%/ -0.5%	$006B_H$/$006C_H$
2400	bit/s	+0.2%/ -0.2%	0145_H/0146_H	+0.0%/ -0.5%	$00D8_H$/$00D9_H$
1200	bit/s	+0.0%/ -0.2%	$028A_H$/$028B_H$	+0.0%/ -0.2%	$01B1_H$/$01B2_H$
600	bit/s	+0.0%/ -0.1%	0515_H/0516_H	+0.0%/ -0.1%	0363_H/0364_H
95	bit/s	+0.4%	$1FFF_H$	+0.0%/ -0.0%	1569_H/$156A_H$
63	bit/s	–	–	+1.0%	$1FFF_H$

Table 11-4 ASC0 Asynchronous Baudrate Generation for f_{CPU} = 33 MHz

Baud Rate		S0BRS = '0'		S0BRS = '1'	
		Deviation Error	Reload Value	Deviation Error	Reload Value
1.031	Mbit/s	±0.0%	0000_H	–	–
19.2	kbit/s	+1.3%/ -0.5%	0034_H/0035_H	+2.3%/ -0.5%	0022_H/0023_H
9600	bit/s	+0.4%/ -0.5%	$006A_H$/$006B_H$	+0.9%/ -0.5%	0046_H/0047_H
4800	bit/s	+0.4%/ -0.1%	$00D5_H$/$00D6_H$	+0.2%/ -0.5%	$008E_H$/$008F_H$
2400	bit/s	+0.2%/ -0.1%	$01AC_H$/$01AD_H$	+0.2%/ -0.2%	$011D_H$/$011E_H$
1200	bit/s	+0.0%/ -0.1%	$035A_H$/$035B_H$	+0.2%/ -0.0%	$023B_H$/$023C_H$
600	bit/s	+0.0%/ -0.0%	$06B5_H$/$06B6_H$	+0.1%/ -0.0%	0478_H/0479_H
125	bit/s	+7.1%	$1FFF_H$	±0.0%	$157C_H$
84	bit/s	–	–	-0.9%	$1FFF_H$

Synchronous Mode Baud Rates

For synchronous operation, the baud rate generator provides a clock with 4 times the rate of the established baud rate. The baud rate for synchronous operation of serial channel ASC0 can be determined by the following formula:

$$S0BRL = \left(\frac{f_{CPU}}{4 \times (2 + <S0BRS>) \times B_{Sync}}\right) - 1$$

$$B_{Sync} = \frac{f_{CPU}}{4 \times (2 + <S0BRS>) \times (<S0BRL> + 1)}$$

<S0BRL> represents the contents of the reload register, taken as unsigned 13-bit integers, <S0BRS> represents the value of bit S0BRS (either ‘0’ or ‘1’), taken as integer.

Table 11-5 gives the limit baudrates depending on the CPU clock frequency and bit S0BRS.

Table 11-5 ASC0 Synchronous Baudrate Generation

CPU clock f_{CPU}	S0BRS = ‘0’		S0BRS = ‘1’	
	Min. Baudrate	**Max. Baudrate**	**Min. Baudrate**	**Max. Baudrate**
16 MHz	244 bit/s	2.000 Mbit/s	162 bit/s	1.333 Mbit/s
20 MHz	305 bit/s	2.500 Mbit/s	203 bit/s	1.666 Mbit/s
25 MHz	381 bit/s	3.125 Mbit/s	254 bit/s	2.083 Mbit/s
33 MHz	504 bit/s	4.125 Mbit/s	336 bit/s	2.750 Mbit/s

11.5 ASC0 Interrupt Control

Four bit-addressable interrupt control registers are provided for serial channel ASC0. Register S0TIC controls the transmit interrupt, S0TBIC controls the transmit buffer interrupt, S0RIC controls the receive interrupt, and S0EIC controls the error interrupt of serial channel ASC0. Each interrupt source also has its own dedicated interrupt vector. S0TINT is the transmit interrupt vector, S0TBINT is the transmit buffer interrupt vector, S0RINT is the receive interrupt vector, and S0EINT is the error interrupt vector.

The cause of an error interrupt request (framing, parity, overrun error) can be identified by the error status flags in control register S0CON.

Note: In contrast to the error interrupt request flag S0EIR, the error status flags S0FE/ S0PE/S0OE are not reset automatically upon entry into the error interrupt service routine, but must be cleared by software.

S0TIC
ASC0 Tx Intr. Ctrl. Reg. **SFR ($FF6C_H/B6_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								S0 TIR	S0 TIE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

S0TBIC
ASC0 Tx Buf. Intr. Ctrl. Reg. **SFR ($FF9C_H/CE_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								S0 TBIR	S0 TBIE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

S0RIC
ASC0 Rx Intr. Ctrl. Reg. **SFR ($FF6E_H/B7_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								S0 RIR	S0 RIE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

S0EIC
ASC0 Error Intr. Ctrl. Reg. **SFR ($FF70_H/B8_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								S0 EIR	S0 EIE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

Using the ASC0 Interrupts

For normal operation (other than the error interrupt), the ASC0 provides three interrupt requests to control data exchange via this serial channel:

- S0TBIR is activated when data is moved from S0TBUF to the transmit shift register.
- S0TIR is activated before the last bit of an asynchronous frame is transmitted, or after the last bit of a synchronous frame has been transmitted.
- S0RIR is activated when the received frame is moved to S0RBUF.

While the task of the receive interrupt handler is quite clear, the transmitter is serviced by two interrupt handlers. This provides advantages for the servicing software.

For single transfers, it is sufficient to use the transmitter interrupt (S0TIR) which indicates that the previously loaded data has been transmitted (except for the last bit of an asynchronous frame).

For multiple back-to-back transfers, it is necessary to load the subsequent piece of data at last until the last bit of the previous frame has been transmitted. In asynchronous mode this leaves only one bit-time for the handler to respond to the transmitter interrupt request. In synchronous mode this makes response impossible.
Using the transmit buffer interrupt (S0TBIR) to reload transmit data provides the time to transmit a complete frame for the service routine, as S0TBUF may be reloaded while the previous data is still being transmitted.

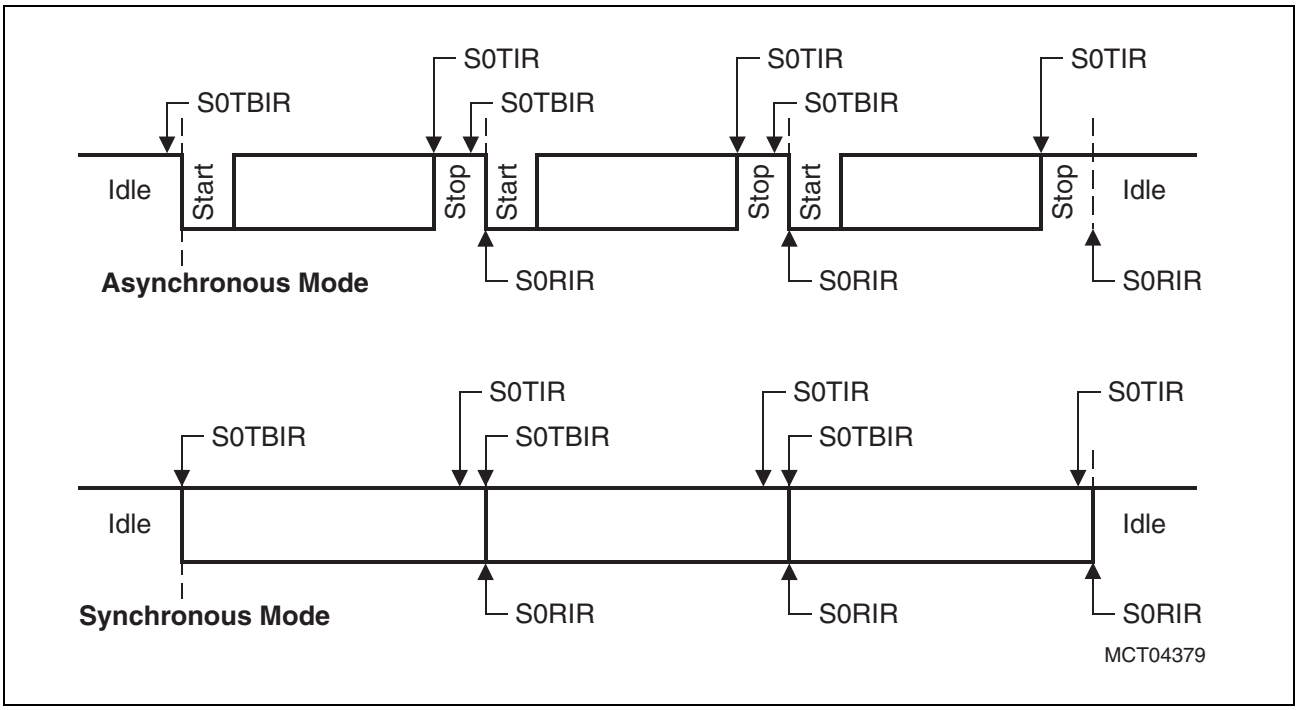


Figure 11-6 ASC0 Interrupt Generation

As shown in **Figure 11-6**, S0TBIR is an early trigger for the reload routine, while S0TIR indicates the completed transmission. Therefore, software using handshake should rely on S0TIR at the end of a data block to ensure that all data has really been transmitted.

12 High-Speed Synchronous Serial Interface

The high-speed Synchronous Serial Interface (SSC) provides flexible high-speed serial communication between the C164CI and other microcontrollers, microprocessors, or external peripherals.

The SSC supports full-duplex and half-duplex synchronous communication (for baud rate ranges see formulas and tables in **Section 12.5**). The serial clock signal can be generated by the SSC itself (master mode) or can be received from an external master (slave mode). Data width, shift direction, clock polarity, and phase are programmable. This allows communication with SPI-compatible devices. Transmission and reception of data is double-buffered. A 16-bit baud rate generator provides the SSC with a separate serial clock signal.

Configuration of the high-speed synchronous serial interface is very flexible, so it can be used with other synchronous serial interfaces (such as the ASC0 in synchronous mode), it can serve for master/slave or multimaster interconnections, or it can operate compatibly with the popular SPI interface. Thus, the SSC can be used to communicate with shift registers (IO expansion), peripherals (EEPROMs etc.) or other controllers (networking). The SSC supports half-duplex and full-duplex communication. Data is transmitted or received on pins MTSR/P3.9 (Master Transmit/Slave Receive) and MRST/P3.8 (Master Receive/Slave Transmit). The clock signal is output or input on pin SCLK/P3.13. These pins are alternate functions of Port 3 pins.

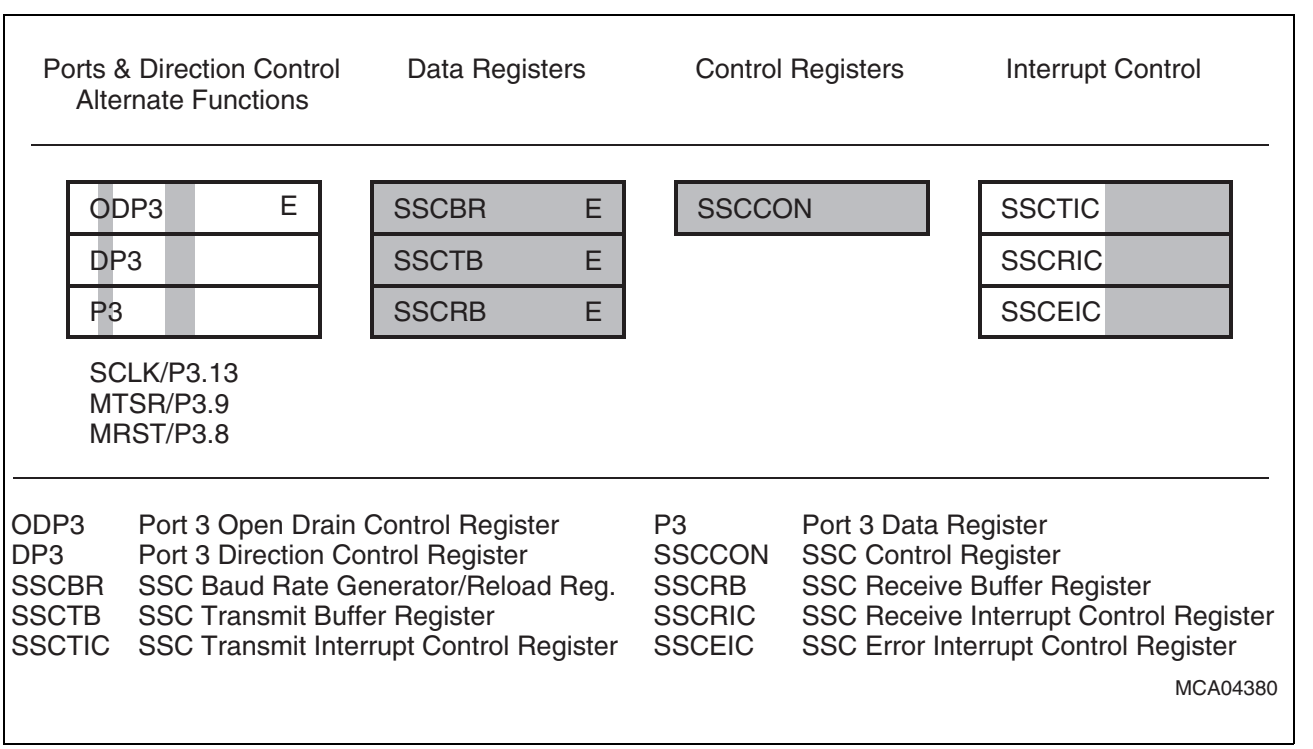


Figure 12-1 SFRs and Port Pins Associated with the SSC

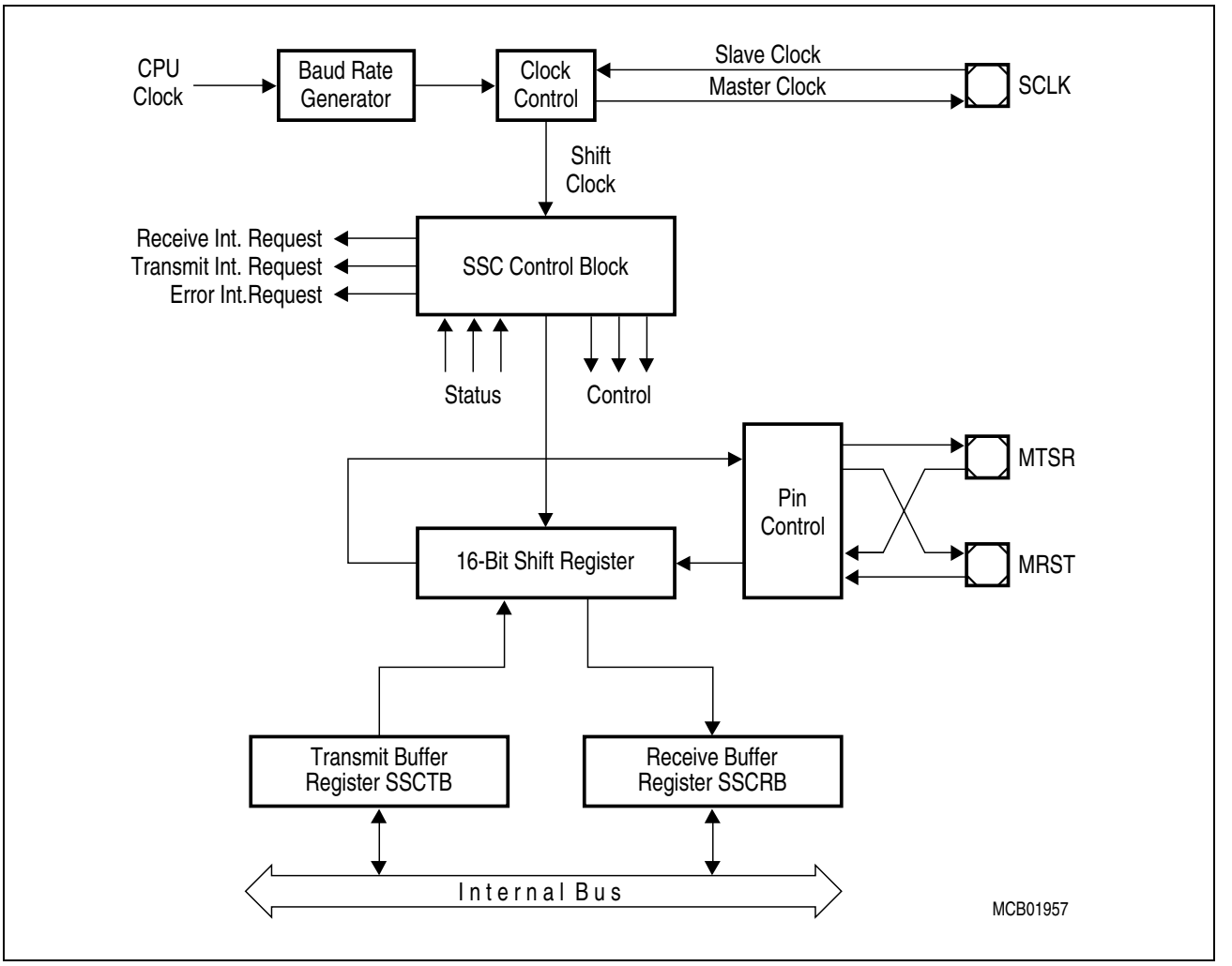


Figure 12-2 Synchronous Serial Channel SSC Block Diagram

The operating mode of the serial channel SSC is controlled by its bit-addressable control register SSCCON. This register serves two purposes:

- During programming (SSC disabled by SSCEN = ‘0’) it provides access to a set of control bits,
- During operation (SSC enabled by SSCEN = ‘1’) it provides access to a set of status flags.

Register SSCCON is shown below in each of the two modes.

SSCCON
SSC Control Reg. (Pr.M.) SFR ($FFB2_H$/$D9_H$) Reset Value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SSC EN = 0	SSC MS	-	SSC AR EN	SSC BEN	SSC PEN	SSC REN	SSC TEN	-	SSC PO	SSC PH	SSC HB	SSCBM			
rw	rw	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw			

Bit	Function (Programming Mode, SSCEN = '0')
SSCBM	**SSC Data Width Selection** 0: Reserved. Do not use this combination 1…15: Transfer Data Width is 2 … 16 bit (<SSCBM> + 1)
SSCHB	**SSC Heading Control Bit** 0: Transmit/Receive LSB First 1: Transmit/Receive MSB First
SSCPH	**SSC Clock Phase Control Bit** 0: Shift transmit data on the leading clock edge, latch on trailing edge 1: Latch receive data on leading clock edge, shift on trailing edge
SSCPO	**SSC Clock Polarity Control Bit** 0: Idle clock line is low, leading clock edge is low-to-high transition 1: Idle clock line is high, leading clock edge is high-to-low transition
SSCTEN	**SSC Transmit Error Enable Bit** 0: Ignore transmit errors 1: Check transmit errors
SSCREN	**SSC Receive Error Enable Bit** 0: Ignore receive errors 1: Check receive errors
SSCPEN	**SSC Phase Error Enable Bit** 0: Ignore phase errors 1: Check phase errors
SSCBEN	**SSC Baudrate Error Enable Bit** 0: Ignore baudrate errors 1: Check baudrate errors
SSCAREN	**SSC Automatic Reset Enable Bit** 0: No additional action upon a baudrate error 1: The SSC is automatically reset upon a baudrate error
SSCMS	**SSC Master Select Bit** 0: Slave Mode. Operate on shift clock received via SCLK 1: Master Mode. Generate shift clock and output it via SCLK
SSCEN	**SSC Enable Bit = '0'** Transmission and reception disabled. Access to control bits

SSCCON
SSC Control Reg. (Op.M.) **SFR ($FFB2_H$/$D9_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SSC EN = 1	SSC MS	-	SSC BSY	SSC BE	SSC PE	SSC RE	SSC TE	-	-	-	-	SSCBC			
rw	rw	-	rw	rw	rw	rw	rw	-	-	-	-	r			

Bit	Function (Operating Mode, SSCEN = '1')
SSCBC	**SSC Bit Count Field** Shift counter is updated with every shifted bit. **Do not write to!!!**
SSCTE	**SSC Transmit Error Flag** 1: Transfer starts with the slave's transmit buffer not being updated.
SSCRE	**SSC Receive Error Flag** 1: Reception completed before the receive buffer was read.
SSCPE	**SSC Phase Error Flag** 1: Received data changes around sampling clock edge.
SSCBE	**SSC Baudrate Error Flag** 1: More than factor 2 or less than factor 0.5 between Slave's actual and expected baudrate.
SSCBSY	**SSC Busy Flag** Set while a transfer is in progress. **Do not write to!!!**
SSCMS	**SSC Master Select Bit** 0: Slave Mode. Operate on shift clock received via SCLK. 1: Master Mode. Generate shift clock and output it via SCLK.
SSCEN	**SSC Enable Bit = '1'** Transmission and reception enabled. Access to status flags and Master/ Slave (M/S) control.

Note: The target of an access to SSCCON (control bits or flags) is determined by the state of SSCEN prior to the access, for instance, writing $C057_H$ to SSCCON in programming mode (SSCEN = '0') will initialize the SSC (SSCEN was '0') and then turn it on (SSCEN = '1').
When writing to SSCCON, make sure that reserved locations receive zeros.

The shift register of the SSC is connected to both the transmit pin and the receive pin via the pin control logic (see block diagram). Transmission and reception of serial data is synchronized and simultaneous: the same number of bits are transmitted and received.

The major steps of the state machine of the SSC are controlled by the shift clock signal (see **Figure 12-2**).
In master mode (SSCMS = '1') two clocks per bit-time are generated, each upon an underflow of the baudrate counter.
In slave mode (SSCMS = '0') one clock per bit-time is generated, when the latching edge of the external SCLK signal has been detected.

Transmit data is written into the transmit buffer SSCTB. When the contents of the buffer are moved to the shift register (immediately if no transfer is currently active) a transmit interrupt request (SSCTIR) is generated indicating that SSCTB may be reloaded again.

The busy flag SSCBSY is set when the transfer starts (with the next following shift clock in master mode, immediately in slave mode).

Note: If no data is written to SSCTB prior to a slave transfer, this transfer starts after the first latching edge of the external SCLK signal is detected. No transmit interrupt is generated in this case.

When the contents of the shift register are moved to the receive buffer SSCRB after the programmed number of bits (2 … 16) have been transferred, i.e. after the last latching edge of the current transfer, a receive interrupt request (SSCRIR) is generated.

The busy flag SSCBSY is cleared at the end of the current transfer (with the next following shift clock in master mode, immediately in slave mode).

When the transmit buffer is not empty at that time (in the case of continuous transfers) the busy flag is not cleared and the transfer goes on after moving data from the buffer to the shift register.

Software should not modify SSCBSY, as this flag is hardware controlled.

Note: Only one SSC (etc.) can be master at a given time.

The transfer of serial data bits can be programmed in many respects:

- Data width can be chosen from 2 bits to 16 bits
- Transfer may start with the LSB or the MSB
- Shift clock may be idle low or idle high
- Data bits may be shifted with the leading or trailing edge of the clock signal
- Baudrate may be set within a wide range (see baudrate generation)
- Shift clock may be generated (master) or received (slave)

This flexibility allows adaptation of the SSC to a wide range of applications which require serial data transfer.

The Data Width Selection supports the transfer of frames of any length from 2-bit “characters” up to 16-bit “characters”. Starting with the LSB (SSCHB = ‘0’) allows communication with ASC0 devices in synchronous mode (C166 Family) or 8051-like serial interfaces, for instance. Starting with the MSB (SSCHB = ‘1’) allows operation compatible with the SPI interface.
Regardless of which data width is selected and whether the MSB or the LSB is transmitted first, the transfer data is always right aligned in registers SSCTB and SSCRB, and the LSB of the transfer data in bit 0 of these registers. The data bits are rearranged for transfer by the internal shift register logic. The unselected bits of SSCTB are ignored. The unselected bits of SSCRB will be not valid and should be ignored by the receiver service routine.

The Clock Control allows adaptation of transmit and receive behavior of the SSC to a variety of serial interfaces. A specific clock edge (rising or falling) is used to shift out transmit data, while the other clock edge is used to latch in receive data. Bit SSCPH selects the leading edge or the trailing edge for each function. Bit SSCPO selects the level of the clock line in the idle state. Thus, for an idle-high clock, the leading edge is a falling one, a 1-to-0 transition. **Figure 12-3** provides a summary.

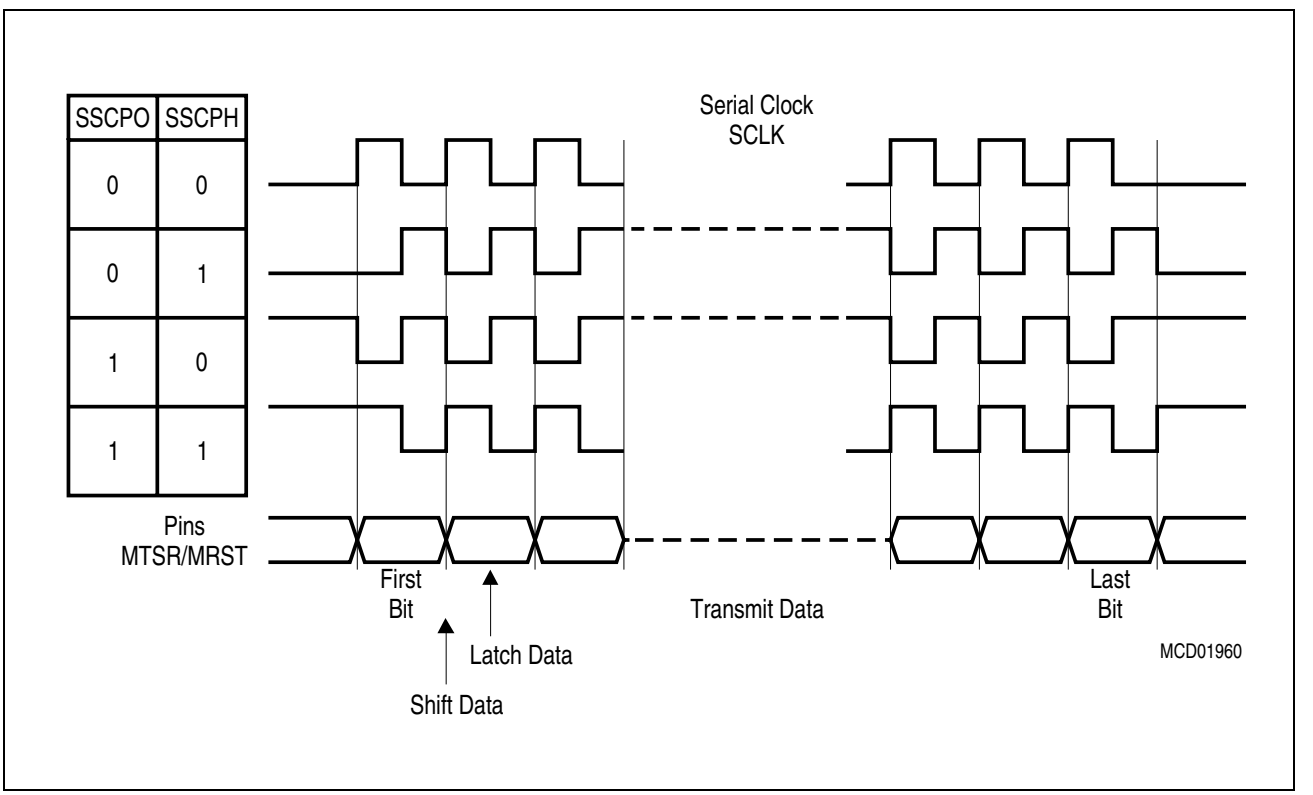


Figure 12-3 Serial Clock Phase and Polarity Options

12.1 Full-Duplex Operation

The various devices are connected through three lines. The definition of these lines is always determined by the master: The line connected to the master's data output pin MTSR is the transmit line, the receive line is connected to its data input line MRST, and the clock line is connected to pin SCLK. Only the device selected for master operation generates and outputs the serial clock on pin SCLK. All slaves receive this clock, so their pin SCLK must be switched to input mode (DP3.13 = '0'). The output of the master's shift register is connected to the external transmit line, which in turn is connected to the slaves' shift register input. The output of the slaves' shift register is connected to the external receive line in order to enable the master to receive the data shifted out of the slave. The external connections are hard-wired, the function and direction of these pins are determined by the master or slave operation of the individual device.

*Note: The shift direction shown in **Figure 12-4** applies for MSB-first operation as well as for LSB-first operation.*

When initializing the devices in this configuration, select one device for master operation (SSCMS = '1'); all others must be programmed for slave operation (SSCMS = '0'). Initialization includes the operating mode of the device's SSC and also the function of the respective port lines (see **Chapter 12.4**).

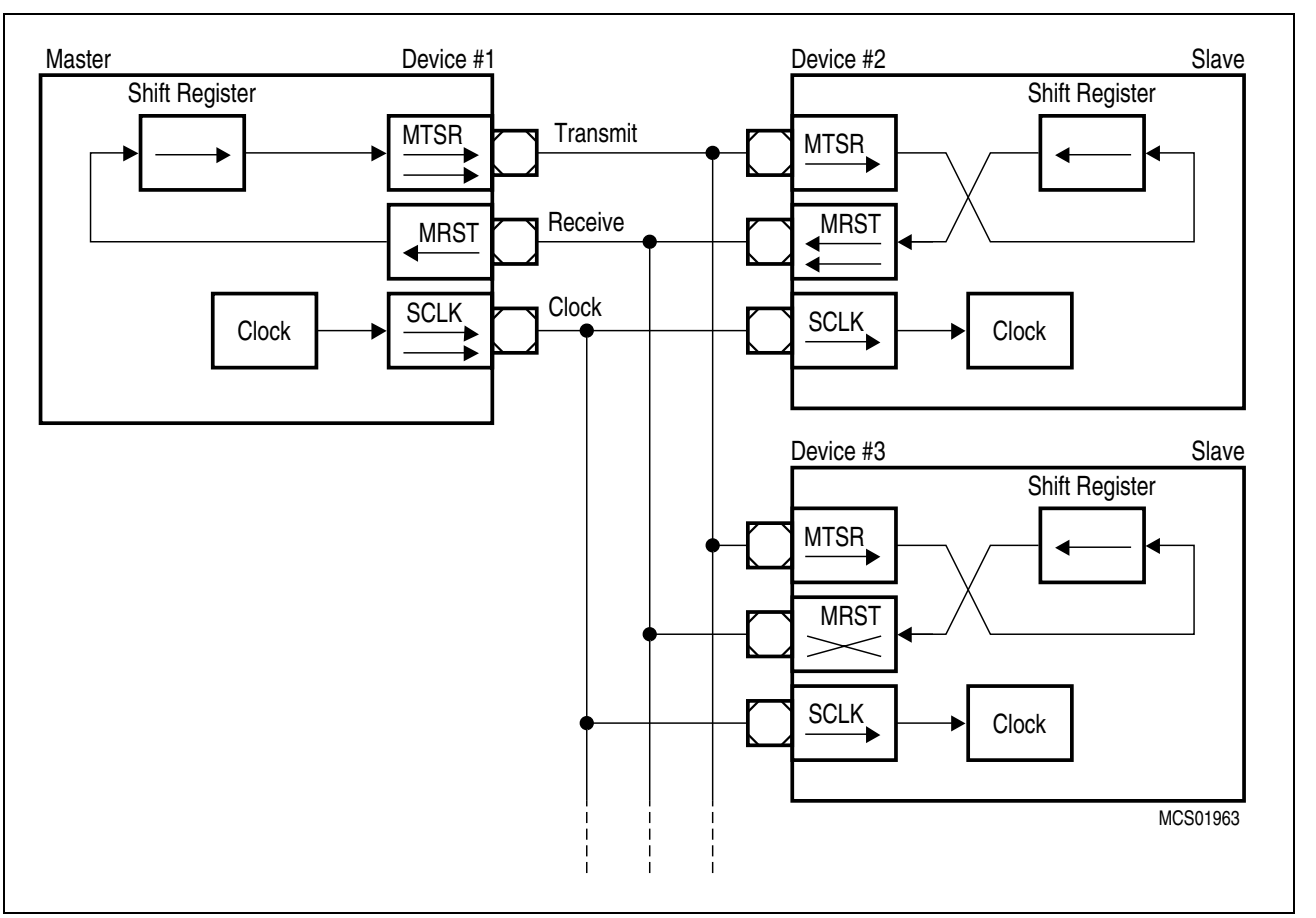


Figure 12-4 SSC Full-Duplex Configuration

The data output pins MRST of all slave devices are connected together onto the one receive line in this configuration. During a transfer, each slave shifts out data from its shift register. There are two ways to avoid collisions on the receive line caused by data from multiple slaves:

Only one slave drives the line, i.e. enables the driver of its MRST pin. All the other slaves have to program their MRST pins to input. So only one slave can put its data onto the master's receive line. Only receiving of data from the master is possible. The master selects the slave device from which it expects data, either by separate select lines or by sending a special command to this slave. The selected slave then switches its MRST line to output, until it gets a deselection signal or command.

The slaves use open drain output on MRST. This forms a Wired-AND connection. The receive line needs an external pull-up in this case. Corruption of the data on the receive line sent by the selected slave is avoided when all slaves not selected for transmission to the master send only ones ('1'). Since this high level is not actively driven onto the line, but only held through the pull-up device, the selected slave can pull this line actively to a low level when transmitting a zero bit. The master selects the slave device from which it expects data, either by separate select lines or by sending a special command to this slave.

After performing all necessary initializations of the SSC, the serial interfaces can be enabled. For a master device, the alternate clock line will now go to its programmed polarity. The alternate data line will go to either '0' or '1', until the first transfer starts.

When the serial interfaces are enabled, the master device can initiate the first data transfer by writing the transmit data into register SSCTB. This value is copied into the shift register (which is assumed to be empty at this time), and the selected first bit of the transmit data will be placed onto the MTSR line on the next clock from the baudrate generator (transmission starts only if SSCEN = '1'). Depending on the selected clock phase, a clock pulse will also be generated on the SCLK line. With the opposite clock edge, the master at the same time latches and shifts in the data detected at its input line MRST. This "exchanges" the transmit data with the receive data. Because the clock line is connected to all slaves, their shift registers will be shifted synchronously with the master's shift register, shifting out the data contained in the registers, and shifting in the data detected at the input line. After the preprogrammed number of clock pulses have occurred (via the data width selection) the data transmitted by the master is contained in all slaves' shift registers, while the master's shift register holds the data of the selected slave. In the master and in all slaves, the contents of the shift register are copied into the receive buffer SSCRB and the receive interrupt flag SSCRIR is set.

A slave device will immediately output the selected first bit (MSB or LSB of the transfer data) at pin MRST when the contents of the transmit buffer are copied into the slave's shift register. It will not wait for the next clock from the baudrate generator, as the master does. The reason for this is that, depending on the selected clock phase, the first clock

edge generated by the master may be used already to clock in the first data bit. So, the slave's first data bit must already be valid at this time.

*Note: On the SSC, a transmission **and** a reception always take place at the same time, regardless of whether valid data has been transmitted or received. This is different from asynchronous reception on ASC0.*

Initialization of the SCLK pin on the master requires some attention to avoid undesired clock transitions which could disturb the other receivers. The state of the internal alternate output lines is '1' as long as the SSC is disabled. This alternate output signal is ANDed with the respective port line output latch. Enabling the SSC with an idle-low clock (SSCPO = '0') will drive the alternate data output and (via the AND) the port pin SCLK immediately low. To avoid this, use the following sequence:

- Select the clock idle level (SSCPO = 'x')
- Load the port output latch with the desired clock idle level (P3.13 = 'x')
- Switch the pin to output (DP3.13 = '1')
- Enable the SSC (SSCEN = '1')
- If SSCPO = '0': enable alternate data output (P3.13 = '1')

The same mechanism for selecting a slave for transmission (separate select lines or special commands) may also be used to move the role of the master to another device in the network. In this case, the previous master and the future master (previous slave) will need to toggle their operating modes (SSCMS) and the direction of their port pins (see description above).

12.2 Half-Duplex Operation

In a half-duplex configuration, only one data line is necessary for both receiving **and** transmitting of data. The data exchange line is connected to both pins MTSR and MRST of each device and the clock line is connected to the SCLK pin.

The master device controls the data transfer by generating the shift clock, while the slave devices receive it. Due to the fact that all transmit and receive pins are connected to the one data exchange line, serial data may be moved between arbitrary stations.

As in full-duplex mode, there are **two ways to avoid collisions** on the data exchange line:

- Only the transmitting device may enable its transmit pin driver
- The non-transmitting devices use open drain output and send only ones

Because the data inputs and outputs are connected together, a transmitting device will clock in its own data at the input pin (MRST for a master device, MTSR for a slave). By these means, any corruptions on the common data exchange line are detected if the received data is not equal to the transmitted data.

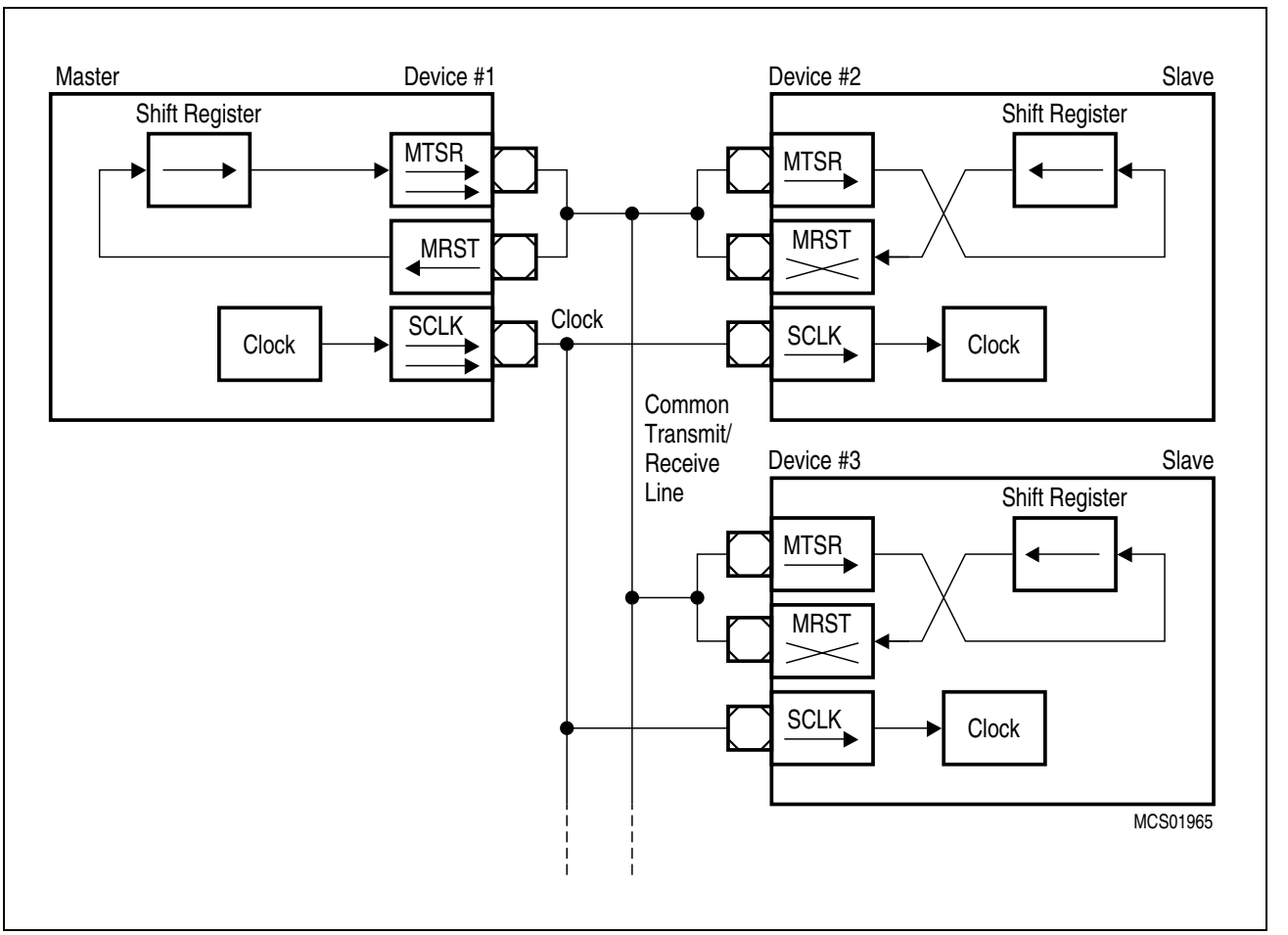


Figure 12-5 SSC Half-Duplex Configuration

12.3 Continuous Transfers

When the transmit interrupt request flag is set, it indicates that the transmit buffer SSCTB is empty and ready to be loaded with the next transmit data. If SSCTB has been reloaded by the time the current transmission is finished, the data is immediately transferred to the shift register and the next transmission will start without any additional delay. On the data line, there is no gap between the two successive frames. For example, two byte transfers would look the same as one word transfer. This feature can be used to interface with devices which can operate with or require more than 16 data bits per transfer. It is only a matter of software, how long a total data frame length can be. This option can also be used to interface to byte-wide and word-wide devices on the same serial bus, for example.

Note: Of course, total frame length can only occur in multiples of the selected basic data width as disabling/enabling of the SSC would be required to reprogram the basic data width on-the-fly.

12.4 Port Control

The SSC uses three pins of Port 3 to communicate with the external world. Pin P3.13/SCLK serves as the clock line, while pins P3.8/MRST (Master Receive/Slave Transmit) and P3.9/MTSR (Master Transmit/Slave Receive) serve as the serial data input/output lines. Operation of these pins depends on the selected operating mode (master or slave). In order to enable the alternate output functions of these pins instead of the general purpose IO operation, the respective port latches must be set to ‘1’, because the port latch outputs and the alternate output lines are ANDed. When an alternate data output line is not used (function disabled), it is held at a high level, allowing IO operations via the port latch. The direction of the port lines depends on the operating mode. The SSC will automatically use the correct alternate input or output line of the ports when switching modes. The direction of the pins, however, must be programmed by the user, as shown in the following tables. Using the open drain output feature helps to avoid bus contention problems and reduces the need for hardwired hand-shaking or slave select lines. In this case, it is not always necessary to switch the direction of a port pin. **Table 12-1** summarizes the required values for the various modes and pins.

Table 12-1 SSC Port Control

Pin	Master Mode			Slave Mode		
	Function	**Port Latch**	**Direction**	**Function**	**Port Latch**	**Direction**
SCLK	Serial Clock Output	P3.13 = ‘1’	DP3.13 = ‘1’	Serial Clock Input	P3.13 = ‘x’	DP3.13 = ‘0’
MTSR	Serial Data Output	P3.9 = ‘1’	DP3.9 = ‘1’	Serial Data Input	P3.9 = ‘x’	DP3.9 = ‘0’
MRST	Serial Data Input	P3.8 = ‘x’	DP3.8 = ‘0’	Serial Data Output	P3.8 = ‘1’	DP3.8 = ‘1’

*Note: In **Table 12-1**, an ‘x’ means that the actual value is irrelevant in the respective mode, however, it is recommended to set these bits to ‘1’, so they are already in the correct state when switching between master and slave modes.*

12.5 Baud Rate Generation

The serial channel SSC has its own dedicated 16-bit baud rate generator with 16-bit reload capability. This permits baud rate generation independent from the timers.

The baud rate generator is clocked with the CPU clock divided by 2 (f_{CPU} / 2). The timer counts downwards and can be started or stopped through the global enable bit SSCEN in register SSCCON. Register SSCBR is the dual-function Baud Rate Generator/Reload register. Reading SSCBR, while the SSC is enabled, returns the contents of the timer. Reading SSCBR, while the SSC is disabled, returns the programmed reload value. In this mode, the desired reload value can be written to SSCBR.

Note: Never write to SSCBR while the SSC is enabled.

The formulas below calculate either the resulting baud rate for a given reload value, or the required reload value for a given baudrate:

$$B_{SSC} = \frac{f_{CPU}}{2 \times (<SSCBR> + 1)} \quad , \quad SSCBR = \left(\frac{f_{CPU}}{2 \times Baudrate_{SSC}}\right) - 1$$

<SSCBR> represents the contents of the reload register taken as an unsigned 16-bit integer.

Table 12-2 lists some possible baud rates together with the required reload values and the resulting bit times, for different CPU clock frequencies.

Table 12-2 SSC Bit-Time Calculation

Bit-time for f_{CPU} = …								Reload Val. (SSCBR)
16 MHz		20 MHz		25 MHz		33 MHz		
Reserved. SSCBR must be > 0.								0000_H
250	ns	200	ns	160	ns	121	ns	0001_H
375	ns	300	ns	240	ns	182	ns	0002_H
500	ns	400	ns	320	ns	242	ns	0003_H
625	ns	500	ns	400	ns	303	ns	0004_H
1.00	µs	800	ns	640	ns	485	ns	0007_H
1.25	µs	1	µs	800	ns	606	ns	0009_H
10	µs	8	µs	6.4	µs	4.8	µs	$004F_H$
12.5	µs	10	µs	8	µs	6.1	µs	0063_H
15.6	µs	12.5	µs	10	µs	7.6	µs	$007C_H$
20.6	µs	16.5	µs	13.2	µs	10	µs	$00A4_H$
1	ms	800	µs	640	µs	485	µs	$1F3F_H$

Table 12-2 SSC Bit-Time Calculation (cont'd)

Bit-time for f_{CPU} = …								Reload Val. (SSCBR)
16 MHz		**20 MHz**		**25 MHz**		**33 MHz**		
1.25	ms	1	ms	800	µs	606	µs	$270F_H$
1.56	ms	1.25	ms	1	ms	758	µs	$30D3_H$
8.2	ms	6.6	ms	5.2	ms	4.0	ms	$FFFF_H$

Table 12-3 SSC Baudrate Calculation

Baud Rate for f_{CPU} = …								Reload Val. (SSCBR)
16 MHz		**20 MHz**		**25 MHz**		**33 MHz**		
Reserved. SSCBR must be > 0.								0000_H
4.00	**Mbit/s**	**5.00**	**Mbit/s**	**6.25**	**Mbit/s**	**8.25**	**Mbit/s**	0001_H
2.67	Mbit/s	3.33	Mbit/s	4.17	Mbit/s	5.50	Mbit/s	0002_H
2.00	Mbit/s	2.50	Mbit/s	3.13	Mbit/s	4.13	Mbit/s	0003_H
1.60	Mbit/s	2.00	Mbit/s	2.50	Mbit/s	3.30	Mbit/s	0004_H
1.00	**Mbit/s**	1.25	Mbit/s	1.56	Mbit/s	2.06	Mbit/s	0007_H
800	kbit/s	**1.0**	**Mbit/s**	1.25	Mbit/s	1.65	Mbit/s	0009_H
100	**kbit/s**	125	kbit/s	156	kbit/s	206	kbit/s	$004F_H$
80	kbit/s	**100**	**kbit/s**	125	kbit/s	165	kbit/s	0063_H
64	kbit/s	80	kbit/s	**100**	**kbit/s**	132	kbit/s	$007C_H$
48.5	kbit/s	60.6	kbit/s	75.8	kbit/s	**100**	**kbit/s**	$00A4_H$
1.0	**kbit/s**	1.25	kbit/s	1.56	kbit/s	2.06	kbit/s	$1F3F_H$
800	bit/s	**1.0**	**kbit/s**	1.25	kbit/s	1.65	kbit/s	$270F_H$
640	bit/s	800	bit/s	**1.0**	**kbit/s**	1.32	kbit/s	$30D3_H$
122.1	bit/s	152.6	bit/s	190.7	bit/s	251.7	bit/s	$FFFF_H$

12.6 Error Detection Mechanisms

The SSC is able to detect four different error conditions: Receive Error and Phase Error are detected in all modes, while Transmit Error and Baudrate Error apply only to slave mode. When an error is detected, the respective error flag is set. When the corresponding Error Enable Bit is set, an error interrupt request will also be generated by setting SSCEIR (see **Figure 12-6**). The error interrupt handler may then check the error flags to determine the cause of the error interrupt. The error flags are not reset automatically (like SSCEIR), but, rather, must be cleared by software after servicing. This allows servicing of some error conditions via interrupt, while the others may be polled by software.

Note: The error interrupt handler must clear the associated (enabled) error flag(s) to prevent repeated interrupt requests.

A **Receive Error** (Master or Slave mode) is detected when a new data frame is completely received, but the previous data was not read out of the receive buffer register SSCRB. This condition sets the error flag SSCRE and, when enabled via SSCREN, the error interrupt request flag SSCEIR. The old data in the receive buffer SSCRB will be overwritten with the new value and is irretrievably lost.

A **Phase Error** (Master or Slave mode) is detected when the incoming data at pin MRST (master mode) or MTSR (slave mode), sampled with the same frequency as the CPU clock, changes between one sample before and two samples after the latching edge of the clock signal (see “Clock Control”). This condition sets the error flag SSCPE and, when enabled via SSCPEN, the error interrupt request flag SSCEIR.

A **Baud Rate Error** (Slave mode) is detected when the incoming clock signal deviates from the programmed baud rate by more than 100%, that is, it either is more than double or less than half the expected baud rate. This condition sets the error flag SSCBE and, when enabled via SSCBEN, the error interrupt request flag SSCEIR. Using this error detection capability requires the slave’s baud rate generator to be programmed to the same baud rate as the master device. This feature detects false additional pulses or missing pulses on the clock line (within a certain frame).

Note: If this error condition occurs and bit SSCAREN = ‘1’, an automatic reset of the SSC will be performed. This is done to reinitialize the SSC if too few or too many clock pulses have been detected.

A **Transmit Error** (Slave mode) is detected when a transfer was initiated by the master (shift clock gets active), but the transmit buffer SSCTB of the slave was not updated since the last transfer. This condition sets the error flag SSCTE and, when enabled via SSCTEN, the error interrupt request flag SSCEIR. If a transfer starts while the transmit buffer is not updated, the slave will shift out the 'old' contents of the shift register, which normally is the data received during the last transfer.
This may lead to the corruption of the data on the transmit/receive line in half-duplex mode (open drain configuration), if this slave is not selected for transmission. This mode requires that slaves not selected for transmission only shift out ones; in other words, their transmit buffers must be loaded with '$FFFF_H$' prior to any transfer.

Note: A slave with push/pull output drivers, which is not selected for transmission, will normally have its output drivers switched. However, to avoid possible conflicts or misinterpretations, it is recommended to always load the slave's transmit buffer prior to any transfer.

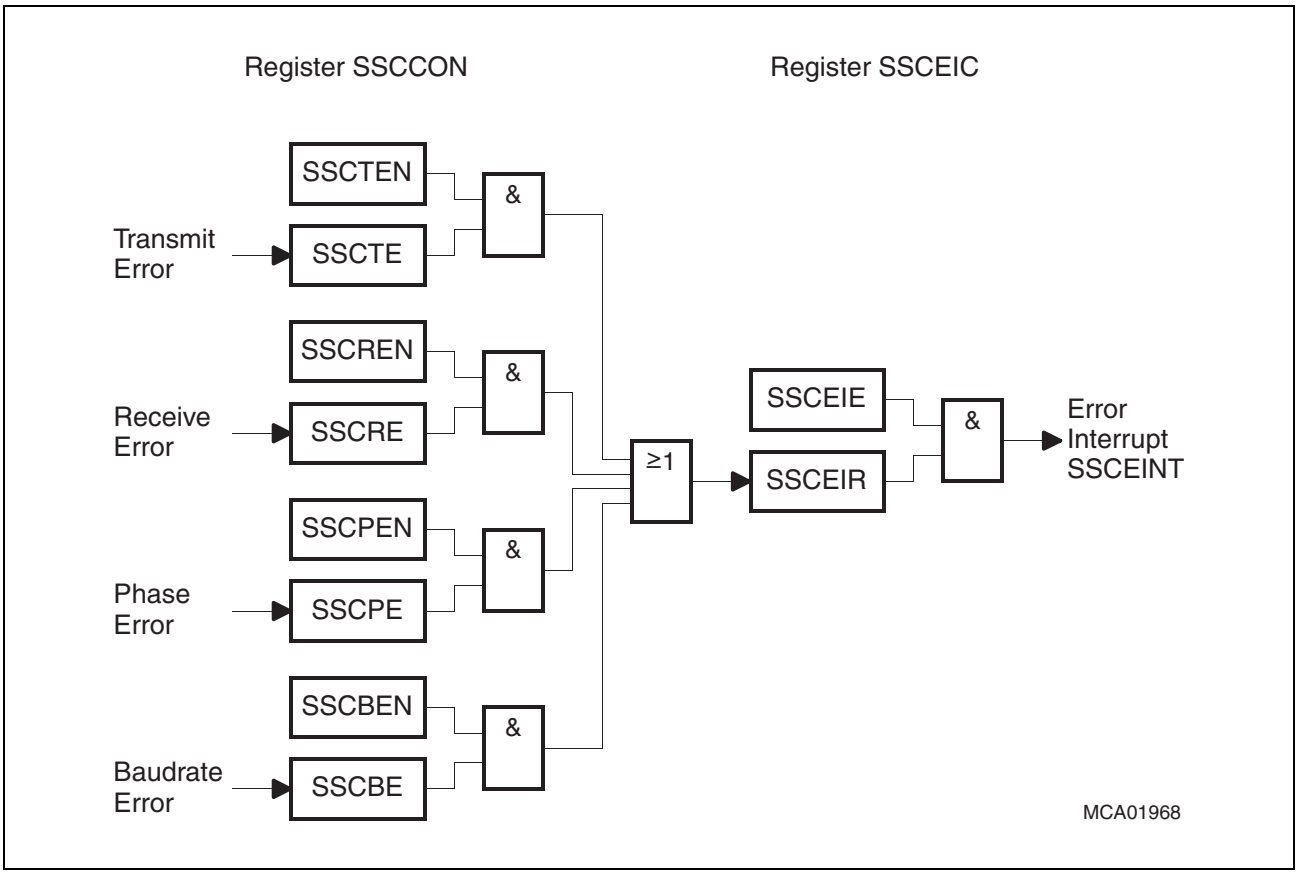


Figure 12-6 SSC Error Interrupt Control

12.7 SSC Interrupt Control

Three bit-addressable interrupt control registers are provided for serial channel SSC. Register SSCTIC controls the transmit interrupt, SSCRIC controls the receive interrupt, and SSCEIC controls the error interrupt of serial channel SSC. Each interrupt source also has its own dedicated interrupt vector. SCTINT is the transmit interrupt vector, SCRINT is the receive interrupt vector, and SCEINT is the error interrupt vector.

The cause of an error interrupt request (receive, phase, baudrate, transmit error) can be identified by the error status flags in control register SSCCON.

Note: In contrast to the error interrupt request flag SSCEIR, the error status flags SSCxE are not reset automatically upon entry into the error interrupt service routine, but must be cleared by software.

SSCTIC
SSC Transmit Intr. Ctrl. Reg. **SFR ($FF72_H/B9_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								SSC TIR	SSC TIE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

SSCRIC
SSC Receive Intr. Ctrl. Reg. **SFR ($FF74_H/BA_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								SSC RIR	SSC RIE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

SSCEIC
SSC Error Intr. Ctrl. Reg. **SFR ($FF76_H/BB_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								SSC EIR	SSC EIE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

13 Watchdog Timer (WDT)

To allow recovery from software or hardware failure, the C164CI provides a Watchdog Timer. If the software fails to service this timer before an overflow occurs, an internal reset sequence will be initiated. This internal reset will also pull the $\overline{\text{RSTOUT}}$ pin low, which in turn resets the peripheral hardware which might have caused the malfunction. 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.

Note: When the bidirectional reset is enabled, pin $\overline{\text{RSTIN}}$ will be pulled low for the duration of the internal reset sequence upon a software reset or a watchdog timer reset.

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.

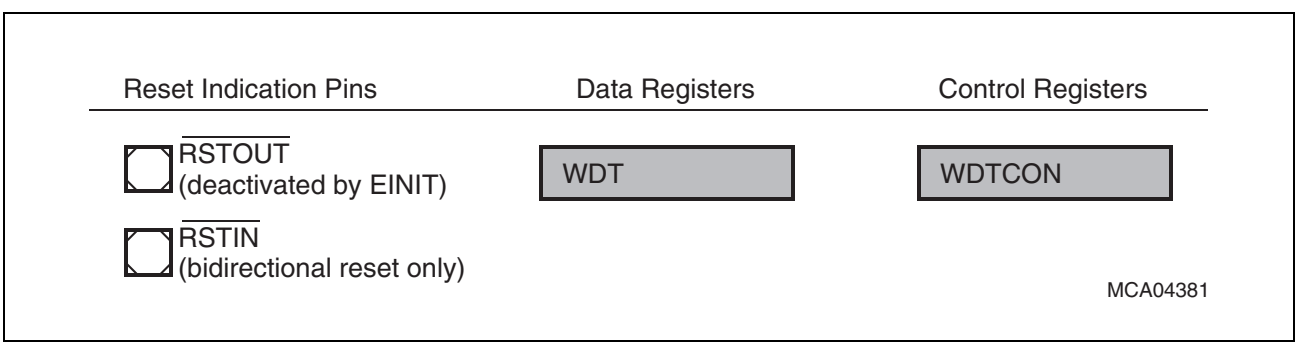


Figure 13-1 SFRs and Port Pins Associated with the Watchdog Timer

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

- by 2 (WDTIN = ‘0’, WDTPRE = ‘0’), or
- by 4 (WDTIN = ‘0’, WDTPRE = ‘1’), or
- by 128 (WDTIN = ‘1’, WDTPRE = ‘0’), or
- by 256 (WDTIN = ‘1’, WDTPRE = ‘1’).

The 16-bit watchdog timer is implemented as two concatenated 8-bit timers (see **Figure 13-2**). 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.

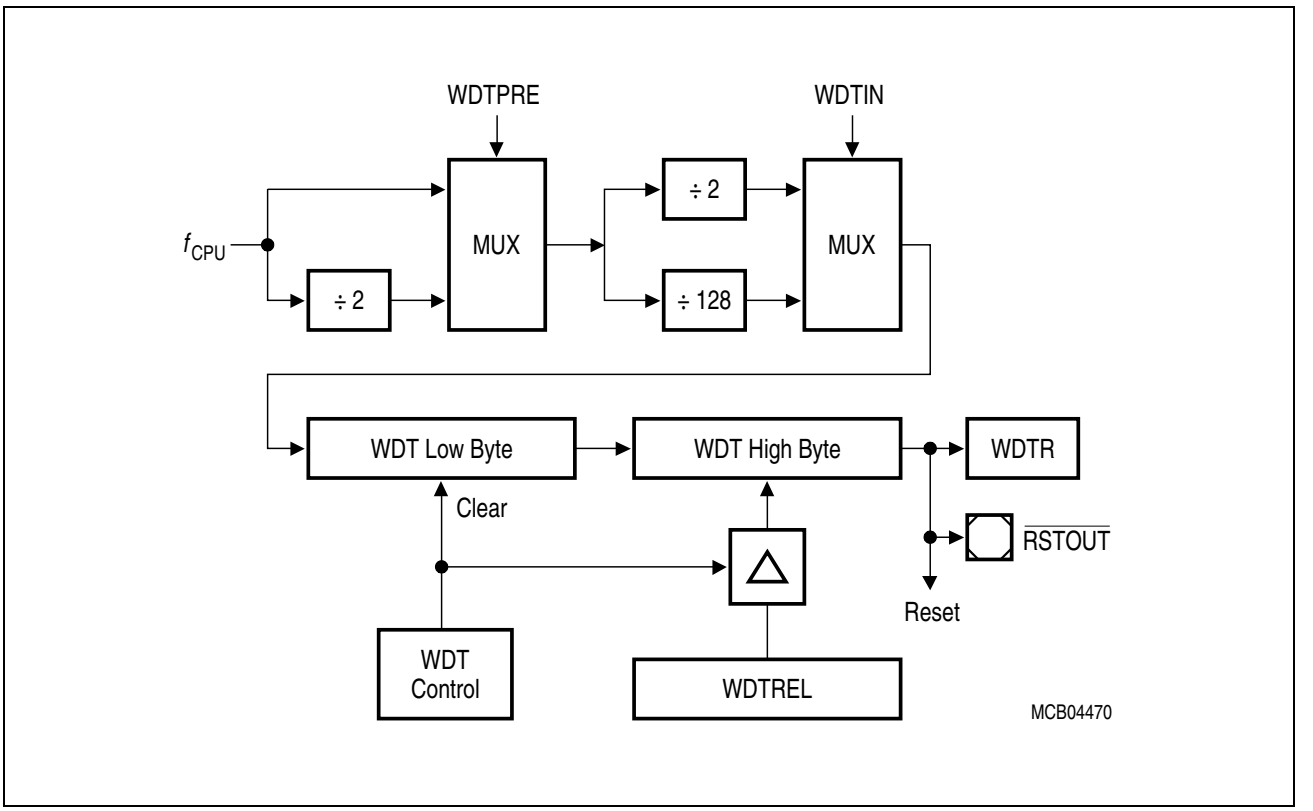


Figure 13-2 Watchdog Timer Block Diagram

13.1 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, selects the input clock prescaling factor, and also provides flags to indicate the source of a reset.

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

The watchdog timer can be disabled by executing the instruction DISWDT (Disable Watchdog Timer). Instruction DISWDT is a protected 32-bit instruction which will ONLY be executed during the time between a reset and execution of either the EINIT (End of Initialization) or the SRVWDT (Service Watchdog Timer) instruction. Either one of these instructions disables the execution of DISWDT.

Note: After a hardware reset that activates the Bootstrap Loader the watchdog timer will be disabled. The software reset which terminates the BSL mode will then enable the WDT.

When the watchdog timer is not disabled via instruction DISWDT it will continue counting up, even in Idle Mode. If it is not serviced via the instruction SRVWDT by the time the count reaches $FFFF_H$ the watchdog timer will overflow and cause an internal reset. This reset will pull the external reset indication pin $\overline{\text{RSTOUT}}$ low. The Watchdog Timer Reset Indication Flag (WDTR) in register WDTCON will be set in this case.
In bidirectional reset mode, pin $\overline{\text{RSTIN}}$ will also be pulled low for the duration of the internal reset sequence and a long hardware reset will be indicated instead.

A watchdog reset will also complete a running external bus cycle before starting the internal reset sequence.

To prevent the watchdog timer from overflowing, it must be serviced periodically by the user software. The watchdog timer is serviced with the instruction SRVWDT which is a protected 32-bit instruction. 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. Servicing the watchdog timer will also reset bit WDTR. 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: $00XX_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WDTREL								WDT PRE	-	-	LHW R	SHW R	SW R	WDT R	WDT IN
-								rw	-	-	rh	rh	rh	rh	rw

Bit	Function
WDTIN	**Watchdog Timer Input Frequency Select** Controls the input clock prescaler. See **Table 13-1**.
WDTR	**Watchdog Timer Reset Indication Flag** Cleared by a hardware reset or by the SRVWDT instruction.
SWR	**Software Reset Indication Flag**
SHWR	**Short Hardware Reset Indication Flag**
LHWR	**Long Hardware Reset Indication Flag**
WDTPRE	**Watchdog Timer Input Prescaler Control** Controls the input clock prescaler. See **Table 13-1**.
WDTREL	**Watchdog Timer Reload Value** (for the high byte of WDT)

Note: The reset value depends on the reset source (see description below).
The execution of EINIT clears the reset indication flags.

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 bits WDTPRE and WDTIN in register WDTCON to be f_{CPU} / 2, f_{CPU} / 4, f_{CPU} / 128, or f_{CPU} / 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:

$$P_{WDT} = \frac{2^{(1 + <WDTPRE> + <WDTIN> \times 6)} \times (2^{16} - <WDTREL> \times 2^{8})}{f_{CPU}}$$

Table 13-1 lists the possible ranges (depending on the prescaler bits WDTPRE and WDTIN) for the watchdog time which can be achieved using a certain CPU clock.

Table 13-1 Watchdog Time Ranges

CPU Clock f_{CPU}	Prescaler			Reload Value in WDTREL					
	WDTIN	WDTPRE	f_{WDT}	FF_H		$7F_H$		00_H	
12 MHz	0	0	f_{CPU} / 2	42.67	µs	5.50	ms	10.92	ms
	0	1	f_{CPU} / 4	85.33	µs	11.01	ms	21.85	ms
	1	0	f_{CPU} / 128	2.73	ms	352.3	ms	699.1	ms
	1	1	f_{CPU} / 256	5.46	ms	704.5	ms	1398	ms
16 MHz	0	0	f_{CPU} / 2	32.00	µs	4.13	ms	8.19	ms
	0	1	f_{CPU} / 4	64.00	µs	8.26	ms	16.38	ms
	1	0	f_{CPU} / 128	2.05	ms	264.2	ms	524.3	ms
	1	1	f_{CPU} / 256	4.10	ms	528.4	ms	1049	ms
20 MHz	0	0	f_{CPU} / 2	25.60	µs	3.30	ms	6.55	ms
	0	1	f_{CPU} / 4	51.20	µs	6.61	ms	13.11	ms
	1	0	f_{CPU} / 128	1.64	ms	211.4	ms	419.4	ms
	1	1	f_{CPU} / 256	3.28	ms	422.7	ms	838.9	ms
25 MHz	0	0	f_{CPU} / 2	20.48	µs	2.64	ms	5.24	ms
	0	1	f_{CPU} / 4	40.96	µs	5.28	ms	10.49	ms
	1	0	f_{CPU} / 128	1.31	ms	169.1	ms	335.5	ms
	1	1	f_{CPU} / 256	2.62	ms	338.2	ms	671.1	ms
33 MHz	0	0	f_{CPU} / 2	15.52	µs	2.00	ms	3.97	ms
	0	1	f_{CPU} / 4	31.03	µs	4.00	ms	7.94	ms
	1	0	f_{CPU} / 128	0.99	ms	128.1	ms	254.2	ms
	1	1	f_{CPU} / 256	1.99	ms	256.2	ms	508.4	ms

Note: For safety reasons, the user is advised to rewrite WDTCON each time before the watchdog timer is serviced.

13.2 Reset Source Indication

Reset indication flags in register WDTCON provide information about the source of the last reset. As the C164CI starts execution from location $00'0000_H$ after any possible reset event, 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 not mutually exclusive; more than one flag may be set after reset depending on its source. **Table 13-2** summarizes the possible combinations:

Table 13-2 Reset Indication Flag Combinations

Event	Reset Indication Flags[1)]			
	LHWR	SHWR	SWR	WDTR
Long Hardware Reset	1	1	1	0
Short Hardware Reset	*	1	1	0
Software Reset	*	*	1	–
Watchdog Timer Reset	*	*	1	1
EINIT instruction	0	0	0	–
SRVWDT instruction	–	–	–	0

1) Description of table entries:
'1' = flag is set, '0' = flag is cleared, '–' = flag is not affected,
'*' = flag is set in bidirectional reset mode, not affected otherwise.

Long Hardware Reset is indicated when the $\overline{\text{RSTIN}}$ input is still sampled low (active) at the end of a hardware triggered internal reset sequence.

Short Hardware Reset is indicated when the $\overline{\text{RSTIN}}$ input is sampled high (inactive) at the end of a hardware triggered internal reset sequence.

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.

Note: When bidirectional reset is enabled, the $\overline{\text{RSTIN}}$ pin is pulled low for the duration of the internal reset sequence upon any sort of reset.
Therefore a long hardware reset (LHWR) will always be recognized in any case.

14 Real Time Clock

The Real Time Clock (RTC) module of the C164CI is basically an independent timer chain clocked directly with the oscillator clock. It serves various purposes:

- System clock to determine the current time and date
- Cyclic time based interrupt
- 48-bit timer for long term measurements

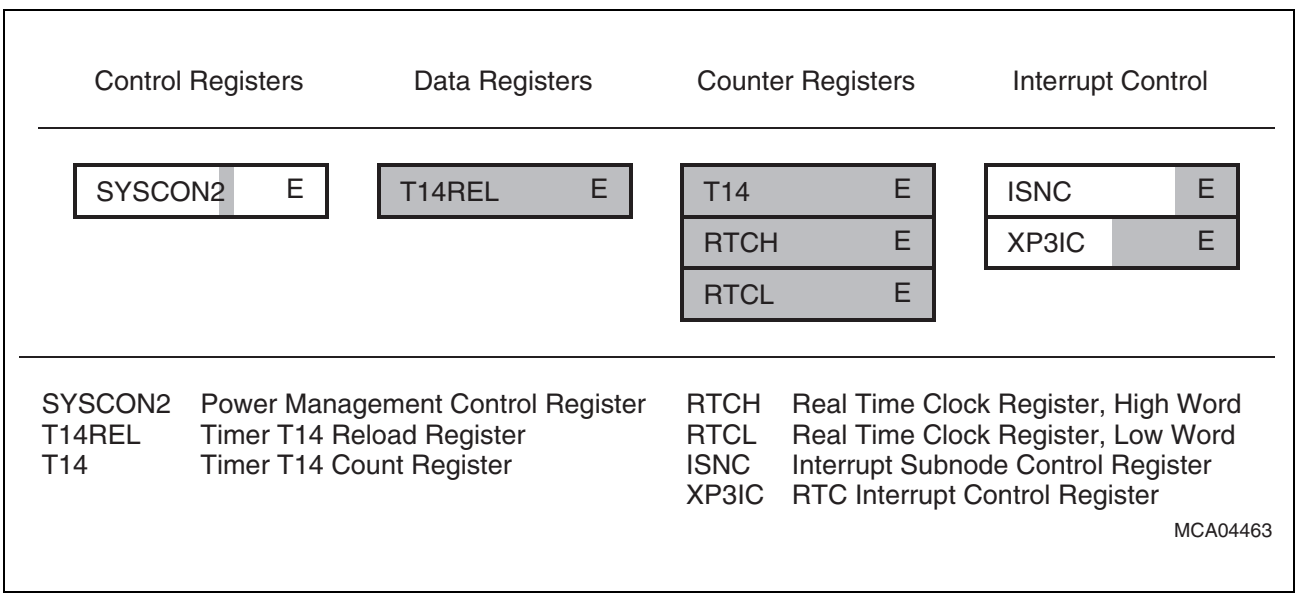


Figure 14-1 SFRs Associated with the RTC Module

The RTC module consists of a chain of 3 divider blocks, a fixed 8:1 divider, the reloadable 16-bit timer T14, and the 32-bit RTC timer (accessible via registers RTCH and RTCL). Both timers count upwards.

The clock signal for the RTC module is derived directly from the on-chip oscillator frequency (not from the CPU clock) and is fed through a separate clock driver. It is therefore independent from the selected clock generation mode of the C164CI and is controlled by the clock generation circuitry.

Table 14-1 RTC Register Location within ESFR Space

Register Name	Long / Short Address	Reset Value	Notes
T14	$F0D2_H$ / 69_H	$UUUU_H$	Prescaler timer, generates input clock for RTC register and periodic interrupt
T14REL	$F0D0_H$ / 68_H	$UUUU_H$	Timer reload register
RTCH	$F0D6_H$ / $6B_H$	$UUUU_H$	High word of RTC register
RTCL	$F0D4_H$ / $6A_H$	$UUUU_H$	Low word of RTC register

Note: The RTC registers are not affected by a reset. After a power on reset, however, they are undefined.

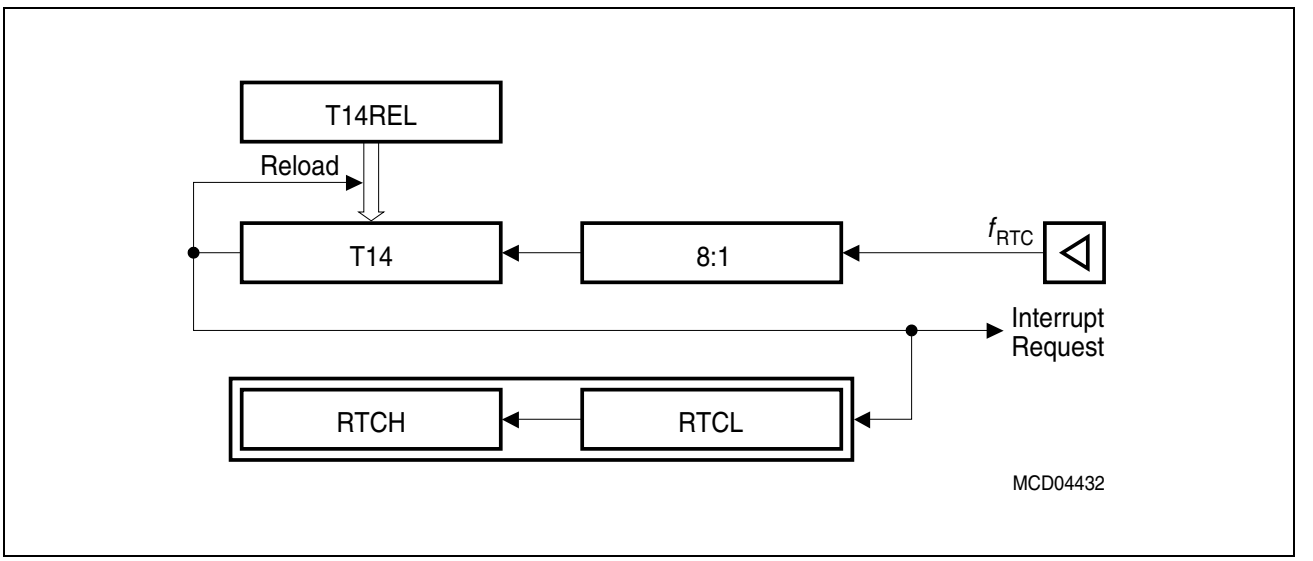


Figure 14-2 RTC Block Diagram

System Clock Operation

A real time system clock can be maintained that keeps on running also during idle mode and power down mode (optionally) and indicates the current time and date. This is possible because the RTC module is not affected by a reset.
The maximum resolution (minimum stepwidth) for this clock information is determined by the input clock of timer T14. The maximum usable timespan is achieved when T14REL is loaded with 0000_H and so T14 divides by 2^{16}.

Cyclic Interrupt Generation

The RTC module can generate an interrupt request whenever timer T14 overflows and is reloaded. This interrupt request may be used, for example, to provide a system time tick independent of the CPU frequency without loading the general purpose timers, or to wake up regularly from idle mode. The interrupt cycle time can be adjusted via the timer T14 reload register T14REL. Please refer to “RTC Interrupt Generation” below for more details.

48-bit Timer Operation

The concatenation of the 16-bit reload timer T14 and the 32-bit RTC timer can be regarded as a 48-bit timer which is clocked with the RTC input frequency divided by the fixed prescaler. The reload register T14REL should be cleared to produce a 48-bit binary timer. However, any other reload value may be used.
The maximum usable timespan is 2^{48} ($\approx 10^{14}$) T14 input clocks, which would equal more than 100 years at an oscillator frequency of 20 MHz.

RTC Register Access

The actual value of the RTC is indicated by the three registers T14, RTCL and RTCH. As these registers are concatenated to build the RTC counter chain, internal overflows occur while the RTC is running. When reading or writing the RTC value, such internal overflows must be taken into account to avoid reading/writing corrupted values. For example, reading/writing 0000_H to RTCH and then accessing RTCL will produce a corrupted value as RTCL may overflow before it can be accessed. In this case, however, RTCH would be 0001_H. The same precautions must be taken for T14 and T14REL.

RTC Interrupt Generation

The RTC interrupt shares the XPER3 interrupt node with the PLL/OWD interrupt. This is controlled by the interrupt subnode control register ISNC. The interrupt handler can determine the source of an interrupt request via the separate interrupt request and enable flags (see **Figure 14-3**) provided in register ISNC.

Note: If only one source is enabled, no additional software check is required, of course.

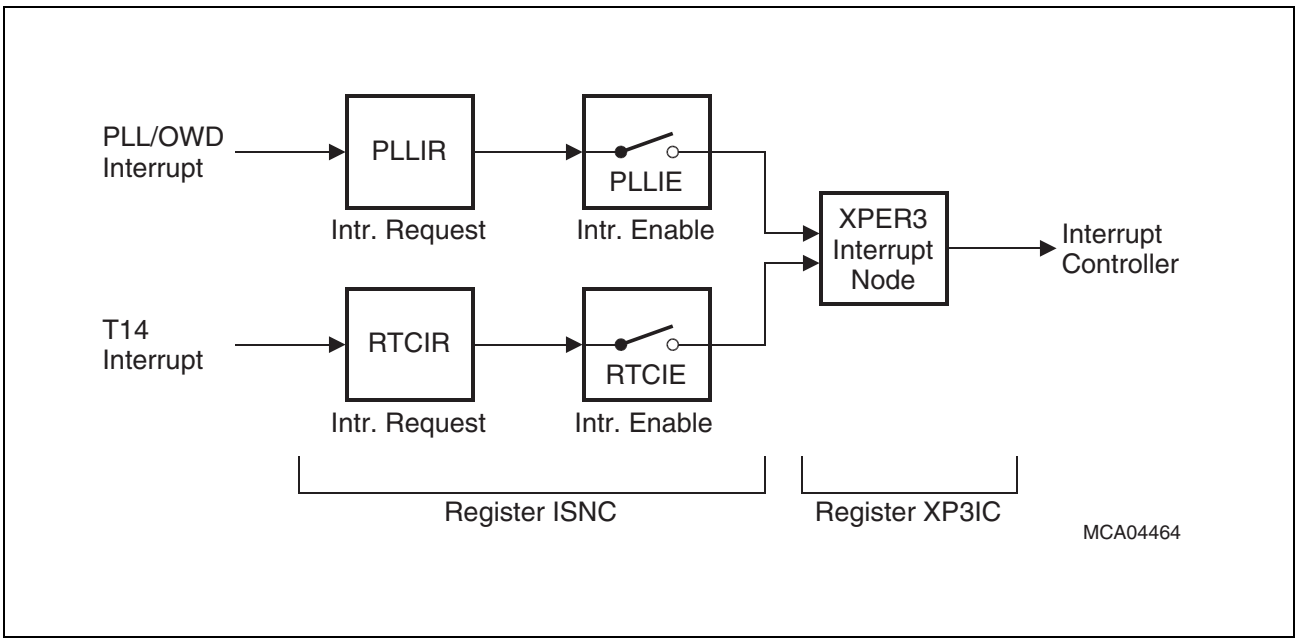


Figure 14-3 RTC Interrupt Logic

If T14 interrupts are to be used both stages the interrupt node (XP3IE = ‘1’) and the RTC subnode (RTCIE = ‘1’) must be enabled.

Please note that the node request bit XP3IR is automatically cleared when the interrupt handler is vectored to, while the subnode request bit RTCIR must be cleared by software.

Defining the RTC Time Base

The reload timer T14 determines the input frequency of the RTC timer, that is, the RTC time base, as well as the T14 interrupt cycle time. **Table 14-2** lists the interrupt period range and the T14 reload values (for a time base of 1 s and 1 ms) for several oscillator frequencies:

Table 14-2 RTC Interrupt Periods and Reload Values

Oscillator Frequency		RTC Interrupt Period		Reload Value A		Reload Value B	
		Minimum	Maximum	T14REL	Base	T14REL	Base
32.768 kHz	Aux.	244.14 µs	16.0 s	$F000_H$	1.000 s	$FFFC_H$	0.977 ms
32 kHz	Aux.	250 µs	16.38 s	$F060_H$	1.000 s	$FFFC_H$	1.000 ms
32 kHz	Main	8000 µs	524.29 s	$FF83_H$	1.000 s	–	–
4 MHz	Main	64.0 µs	4.19 s	$C2F7_H$	1.000 s	$FFF0_H$	1.024 ms
5 MHz	Main	51.2 µs	3.35 s	$B3B5_H$	0.999 s	$FFEC_H$	1.024 ms
8 MHz	Main	32.0 µs	2.10 s	$85EE_H$	1.000 s	$FFE1_H$	0.992 ms
10 MHz	Main	25.6 µs	1.68 s	$676A_H$	0.999 s	$FFD9_H$	0.998 ms
12 MHz	Main	21.3 µs	1.40 s	$48E5_H$	1.000 s	$FFD2_H$	1.003 ms
16 MHz	Main	16.0 µs	1.05 s	$0BDC_H$	1.000 s	$FFC2_H$	0.992 ms

Increased RTC Accuracy through Software Correction

The accuracy of the C164CI's RTC is determined by the oscillator frequency and by the respective prescaling factor (excluding or including T14). The accuracy limit generated by the prescaler is due to the quantization of a binary counter (where the average is zero), while the accuracy limit generated by the oscillator frequency is due to the difference between the ideal and real frequencies (and therefore accumulates over time). The total accuracy of the RTC can be further increased via software for specific applications that demand a high time accuracy.

The key to the improved accuracy is knowledge of the exact oscillator frequency. The relation of this frequency to the expected ideal frequency is a measure of the RTC's deviation. The number of cycles, N, after which this deviation causes an error of ±1 cycle can be easily computed. So, the only action is to correct the count by ±1 after each series of N cycles. This correction may be applied to the RTC register as well as to T14.

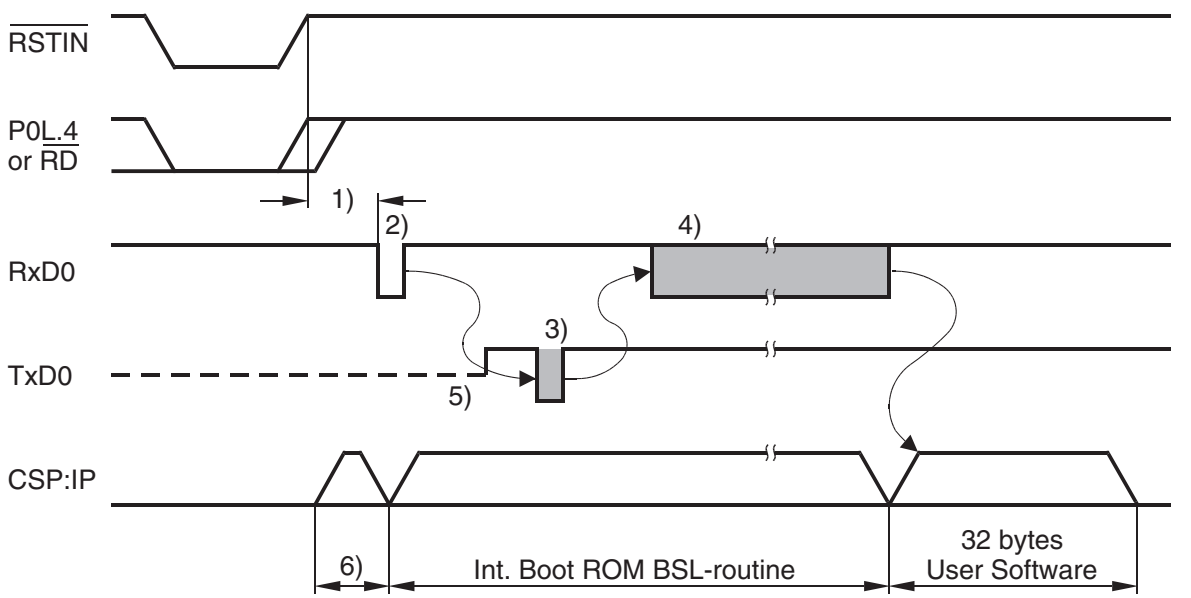
Also the correction may be made cyclically, for instance, within T14's interrupt service routine, or by evaluating a formula when the RTC registers are read (for this the respective "last" RTC value must be available somewhere).

Note: For the majority of applications, however, the standard accuracy provided by the RTC's structure will be more than sufficient.

15 Bootstrap Loader

The built-in bootstrap loader of the C164CI 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 ROM/OTP/Flash is required for the initialization code.

The bootstrap loader moves code/data into the internal RAM, but it is also possible to transfer data via the serial interface into an external RAM using a second level loader routine. ROM memory (internal or external) is not necessary. However, it may be used to provide lookup tables or to provide “core-code” (a set of general purpose subroutines, for IO operations, number crunching, system initialization, etc.).



1) BSL initialization time, < 70/f_{CPU} µs, (f_{CPU} in [MHz]).
2) Zero byte (1 start Bit, eight '0' data Bits, 1 stop Bit), sent by host.
3) Identification byte, sent by microcontroller.
4) 32 bytes of code / data, sent by host.
5) Caution: TxD0 is only driven a certain time after reception of the zero byte (< 40/f_{CPU} µs, f_{CPU} in [MHz])
6) Internal Boot ROM.

MCT04465

Figure 15-1 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, or it may be used to load a programming routine for Flash devices.

The BSL mechanism may be used for standard system startup or for special occasions such as system maintenance (firmware update), end-of-line programming, or testing.

15.1 Entering the Bootstrap Loader

The C164CI enters BSL mode triggered by external configuration during a hardware reset:

- when pin P0L.4 is sampled low at the end of an external reset ($\overline{EA}$ = '0')
- when pin $\overline{RD}$ is sampled low at the end of an internal reset ($\overline{EA}$ = '1').

In this case, the built-in bootstrap loader is activated independent of the selected bus mode. The bootstrap loader code is stored in a special Boot-ROM, no part of the standard mask ROM, OTP, or Flash memory area is required for this.

The hardware that activates the BSL during reset may be a simple pull-down resistor on P0L.4 or $\overline{RD}$ 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.

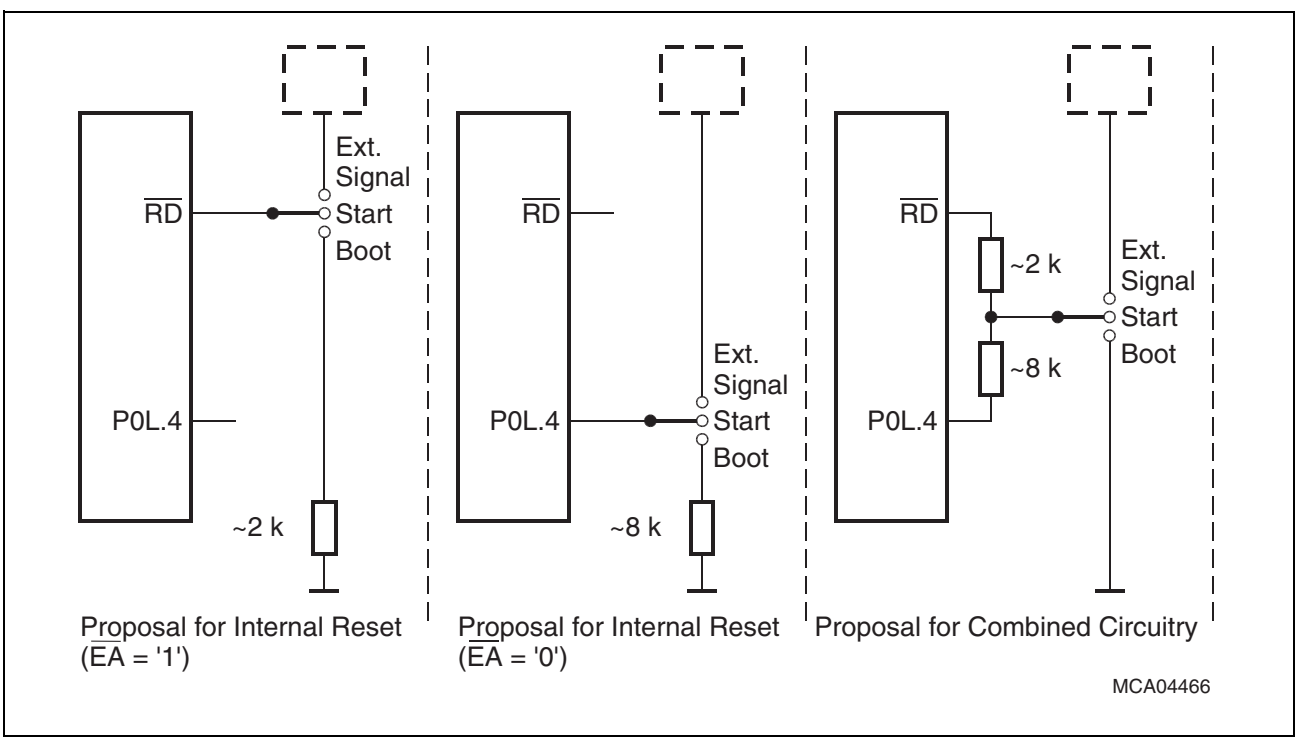


Figure 15-2 Hardware Provisions to Activate the BSL

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 more pins to be driven besides P0L.4 or $\overline{RD}$.
*For an external reset ($\overline{EA}$ = '0') bitfield SMOD must be configured as 1011_B (see **Section 20.4.1**).*
*For an internal reset ($\overline{EA}$ = '1') pin ALE must be driven to a defined level, e.g. ALE = '0' for the standard bootstrap loader (see **Section 20.4.2**).*

Initial State in BSL Mode

After entering BSL mode and the appropriate initialization[1] the C164CI scans the RxD0 line to receive a zero byte, that is, 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 data to be loaded.

This identification byte identifies the device to be booted using the following codes:

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.

Note: The identification byte $D5_H$ does not directly identify a specific derivative. That information can be obtained from the identification registers.

When the C164CI has entered BSL mode, the following configuration is automatically set (values that deviate from the normal reset values, are ***marked***):

Watchdog Timer:	***Disabled***	Register STKUN:	$FC00_H$
Context Pointer CP:	$FA00_H$	Register STKOV:	***$F600_H$***
Stack Pointer SP:	$FA40_H$	Register BUSCON0:	acc. to startup config.
Register S0CON:	***8011_H***	P3.10/TXD0:	'***1***'
Register S0BG:	acc. to '00' byte	DP3.10:	'***1***'

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 C164CI can return the identification byte.

Note: Even if the internal ROM/OTP/Flash is enabled, no code can be executed out of it while the C164CI is in BSL mode.

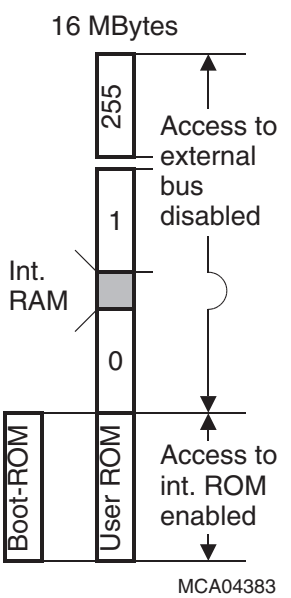
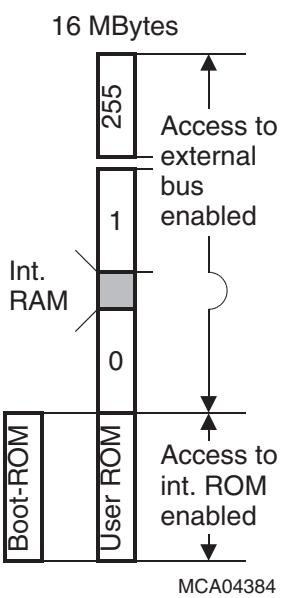
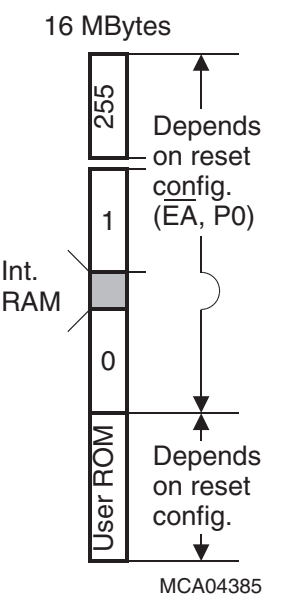
[1] The external host should not send the zero byte before the end of the BSL initialization time (see **Figure 15-1**) to ensure that it is correctly received.

Memory Configuration after Reset

After reset in Bootstrap-Loader mode, the configuration (i.e. the accessibility) of the C164CI's memory areas differs from the standard case. Pin $\overline{EA}$ is not evaluated when BSL mode is selected and accesses to the internal code memory are partly redirected while the C164CI is in BSL mode (see **Table 15-1**). All code fetches are made from the special Boot-ROM, while data accesses read from the internal code memory. Data accesses will return undefined values on ROMless devices.

Note: The code in the Boot-ROM is not an invariant feature of the C164CI. User software should not try to execute code from the internal ROM area while the BSL mode is still active, as these fetches will be redirected to the Boot-ROM.
The Boot-ROM will also "move" to segment 1, when the internal ROM area is mapped to segment 1.

Table 15-1 BSL Memory Configurations

			
BSL mode active	**Yes**	**Yes**	**No**
$\overline{EA}$ pin	high	low	acc. to application
Code fetch from internal ROM area	Boot-ROM access	Boot-ROM access	User ROM access
Data fetch from internal ROM area	User ROM access	User ROM access	User ROM access

15.2 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 in locations $00'FA40_H$ through $00'FA5F_H$ of the internal RAM. Up to 16 instructions may be placed in the RAM area. To execute the loaded code, the BSL then jumps to location $00'FA40_H$, i.e. the first loaded instruction. The bootstrap loading sequence is now terminated, but, the C164CI remains in BSL mode. 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 use the pre-initialized interface ASC0 directly 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 and enable the bus interface to store the received data into external memory.

This process may go through several iterations or may directly execute the final application. In all cases, the C164CI will continue to run in BSL mode, with the watchdog timer disabled and limited access to the internal code memory. All code fetches from the internal ROM area ($00'0000_H$ … $00'7FFF_H$ or $01'0000_H$ … $01'7FFF_H$, if mapped to segment 1) are redirected to the special Boot-ROM. Data fetches access will access the internal code memory of the C164CI, if any is available, but will return undefined data on ROMless devices.

Note: Data fetches from a protected ROM will not be executed.

15.3 Exiting Bootstrap Loader Mode

In order to execute a program in normal mode (i.e. watchdog timer active, full access to user memory, etc.), the BSL mode must first be terminated. The C164CI exits BSL mode in two ways:

- upon a software reset, ignoring the external configuration (P0L.4 or $\overline{RD}$)
- upon a hardware reset, not configuring BSL mode.

After the non-BSL reset, the C164CI will start executing out of user memory as externally configured via PORT0 or $\overline{RD}$/ALE (depending on $\overline{EA}$).

15.4 Choosing the Baudrate for the BSL

Calculation of the serial baudrate for ASC0 from the length of the first zero byte received allows the bootstrap loader of the C164CI to operate with a wide range of baudrates. However, the upper and lower limits must be maintained in order to ensure proper data transfer.

$$B_{C164CI} = \frac{f_{CPU}}{32 \times (S0BRL + 1)}$$

The C164CI uses timer T3 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 S0BRL reload value from the timer contents. The formula below shows the association:

$$S0BRL = \frac{T3 - 18}{36} \quad , \quad T3 = \frac{9}{8} \times \frac{f_{CPU}}{B_{Host}}$$

For a correct data transfer from the host to the C164CI 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 C164CI baudrate can be calculated via the formula below:

$$F_B = \left| \frac{B_{Contr} - B_{Host}}{B_{Contr}} \right| \times 100\% \quad , \quad F_B \leq 2{,}5\%$$

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 15-3**).

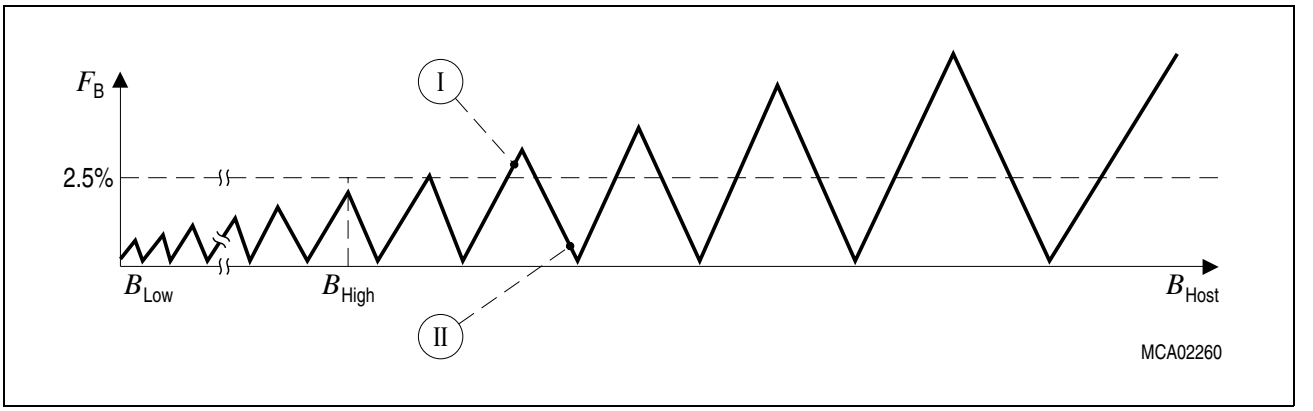


Figure 15-3 Baudrate Deviation between Host and C164CI

The minimum baudrate (B_{Low} in **Figure 15-3**) is determined by the maximum count capacity of timer T3, when measuring the zero byte, thus, it depends on the CPU clock. The minimum baudrate is obtained by using the maximum T3 count 2^{16} in the baudrate formula. Baudrates below B_{Low} would cause T3 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 15-3**) is the highest baudrate at which the deviation still does not exceed the limit; thus, 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 if the actual deviation does not exceed the indicated limit. A certain baudrate (marked I) in **Figure 15-3**) may violate the deviation limit, while an even higher baudrate (marked II) in **Figure 15-3**) stays very well below it. Any baudrate can be used for the bootstrap loader provided that the following three prerequisites are fulfilled:

- Baudrate is within the specified operating range for the ASC0
- External host is able to use this baudrate
- Computed deviation error is below the limit.

Table 15-2 Bootstrap Loader Baudrate Ranges

f_{CPU} [MHz]	10	12	16	20	25
B_{MAX}	312,500	375,000	500,000	625,000	781,250
B_{High}	9,600	19,200	19,200	19,200	38,400
B_{STDmin}	600	600	600	600	600
B_{Low}	172	206	275	343	429

Note: When the bootstrap loader mode is entered via an internal reset ($\overline{EA}$ = '1'), the default configuration selects the prescaler for clock generation. In this case the bootstrap loader will begin to operate with $f_{CPU} = f_{OSC} / 2$ 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 for example to PLL operation (via register RSTCON) to achieve higher baudrates for the subsequent download.

16 Capture/Compare Unit CAPCOM2

The C164CI provides a Capture/Compare (CAPCOM) unit which provides 16 channels (8 IO pins) which interact with 2 timers. The CAPCOM2 unit can **capture** the contents of a timer on specific internal or external events, or can **compare** a timer's contents with given values and modify output signals in case of a match. This mechanism supports generation and control of timing sequences on up to 16 channels with a minimum of software intervention.

From the programmer's point of view, the term 'CAPCOM unit' refers to a set of Special Function Registers (SFRs) associated with this peripheral, including the port pins which may be used for alternate input/output functions including their direction control bits.

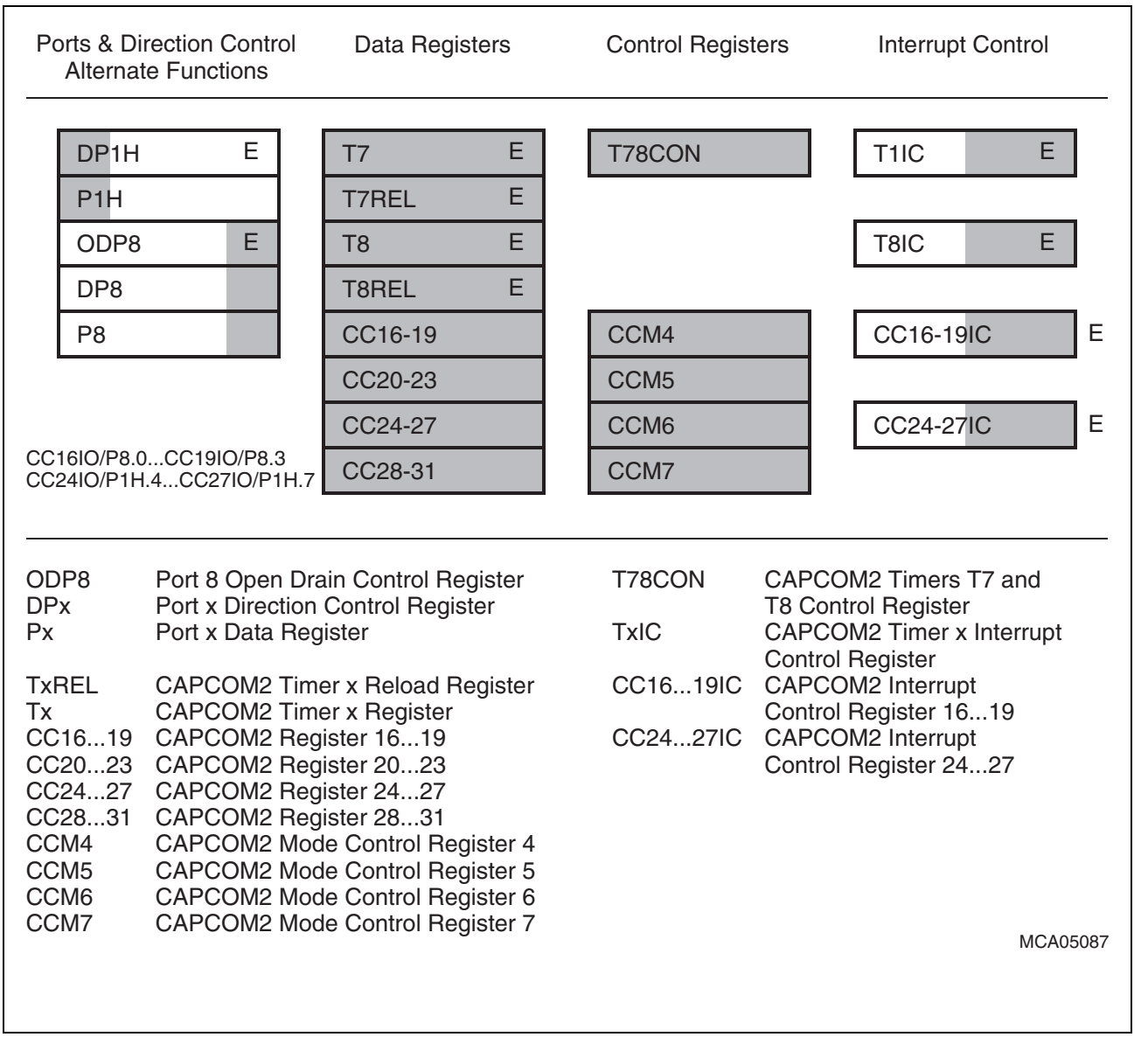


Figure 16-1 SFRs and Port Pins Associated with the CAPCOM2 Unit

The CAPCOM2 unit is typically used to handle high speed IO tasks such as pulse and waveform generation, pulse width modulation, or recording of the time at which specific events occur. It also allows the implementation of up to 16 software timers. The maximum resolution of the CAPCOM2 unit is 8 CPU clock cycles (= 16 TCL).

The CAPCOM2 unit consists of a bank of 16 dual-purpose 16-bit capture/compare registers (CC16 through CC31) and two 16-bit timers (T7/T8). Each has its own reload register (TxREL).

The input clock for the CAPCOM2 timers is programmable to several prescaled values of the CPU clock, or it can be derived from an overflow/underflow of timer T3 in block GPT1. T7 may also operate in counter mode (from an external input) where it can be clocked by external events.

Each capture/compare register may be programmed individually for the capture or compare function, and each register may be allocated to either timer. Eight capture/compare registers have an associated port pin which serves as an input pin for the capture function or as an output pin for the compare function. The capture function causes the current timer contents to be latched into the respective capture/compare register triggered by an event (transition) on its associated port pin. The compare function may cause an output signal transition on that port pin whose associated capture/compare register matches the current timer contents. Specific interrupt requests are generated upon each capture/compare event or upon timer overflow.

Figure 16-2 shows the basic structure of the CAPCOM2 unit.

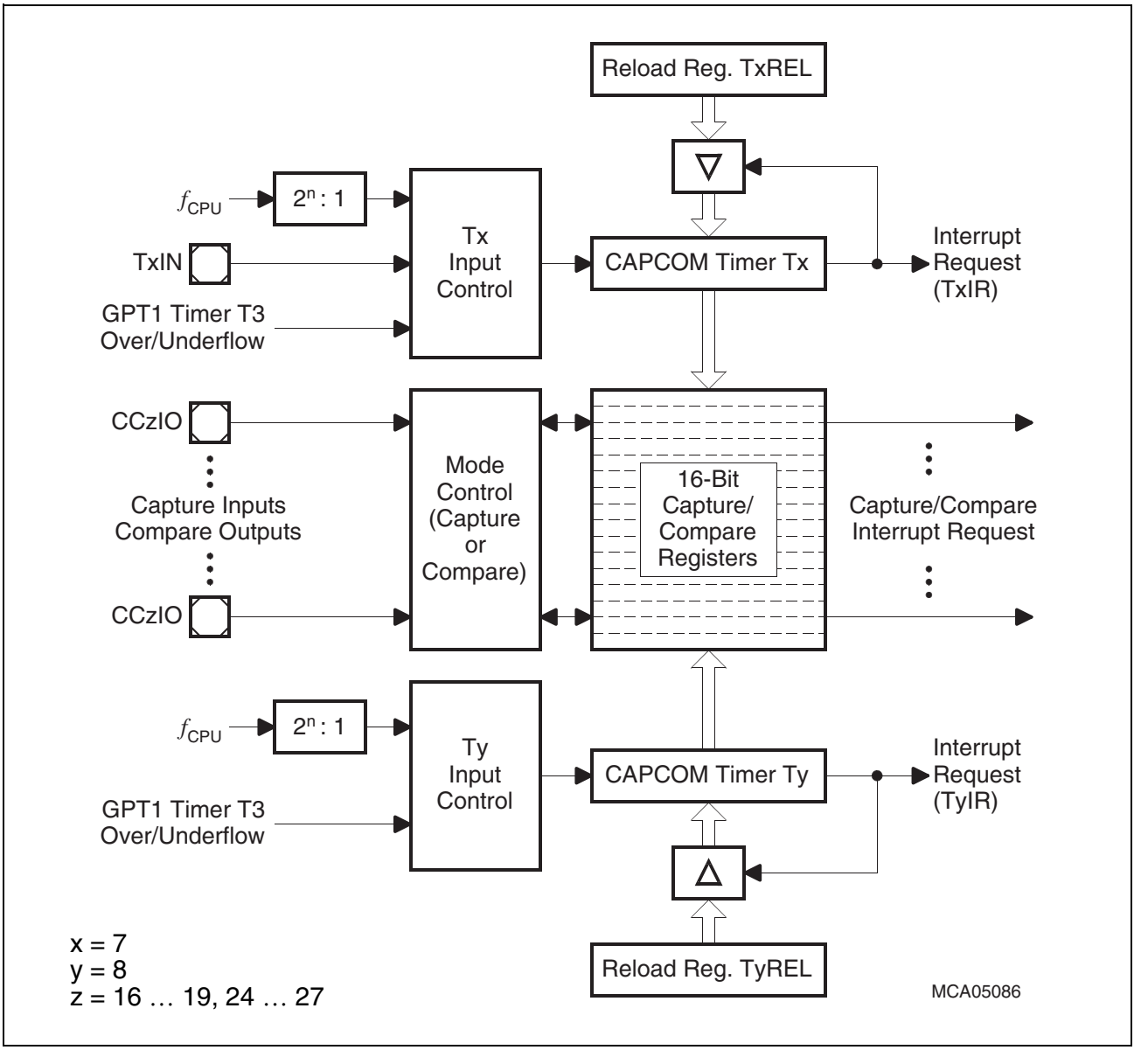


Figure 16-2 CAPCOM2 Unit Block Diagram

Table 16-1 CAPCOM2 Channel Port Connections

Unit	Channel	Port	Capture	Compare
CAPCOM2	CC16IO … CC19IO	P8.3 … P8.0	Input	Output
	CC20IO … CC23IO	–	–	–
	CC24IO … CC27IO	P1H.7 … P1H.4	Input	Output
	CC28IO … CC31IO	–	–	–
	Σ = 16	Σ = 8	Σ = 8	Σ = 8

16.1 CAPCOM2 Timers

The primary use of the timers T7 and T8 is to provide two independent time bases (16 TCL maximum resolution) for the capture/compare registers of the CAPCOM2 unit, but they may also be used independent of the capture/compare registers.
The basic structure of the two timers is identical, but the selection of input signals is different for timer T7 and timer T8 (see **Figure 16-3** and **Figure 16-4**).

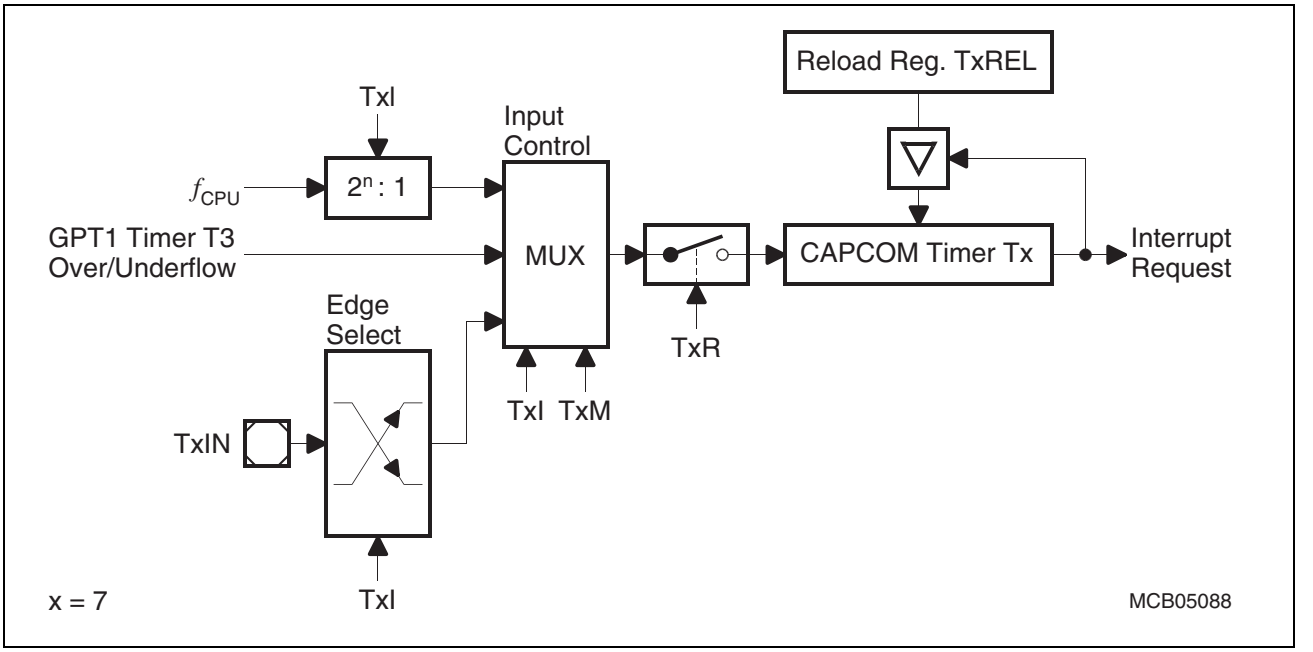


Figure 16-3 Block Diagram of CAPCOM Timer T7

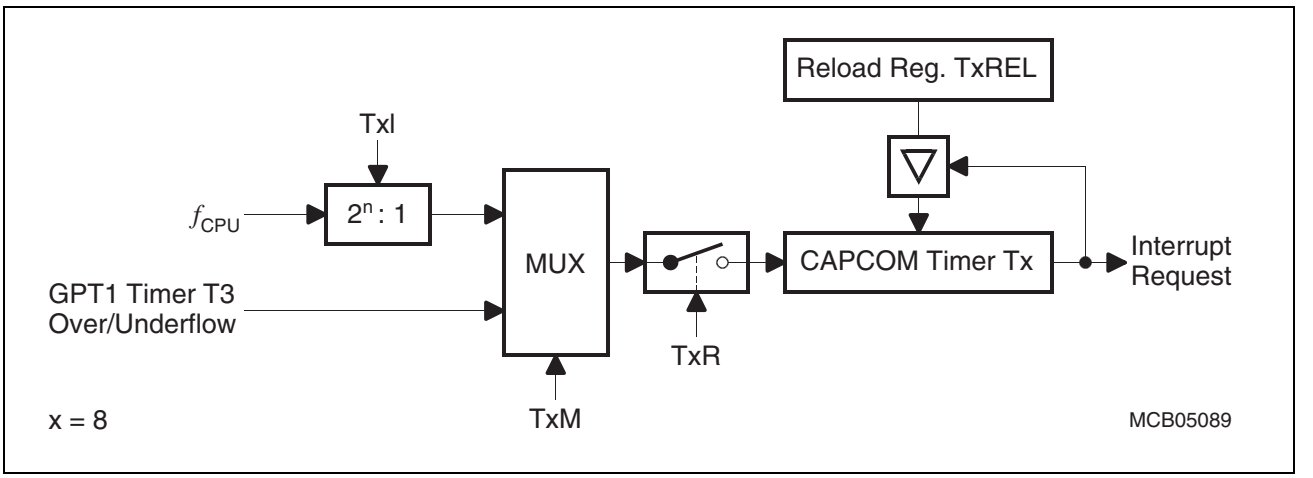


Figure 16-4 Block Diagram of CAPCOM Timer T8

The functions of the CAPCOM timers are controlled via the bit-addressable 16-bit control register T78CON. The high-byte of T78CON controls T8, the low-byte of T78CON controls T7. The control options are identical for both timers (except for external input).

T78CON
CAPCOM Timer 7/8 Ctrl. Reg. SFR ($FF20_H/90_H$) Reset Value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	T8R	-	-	T8M	T8I			-	T7R	-	-	T7M	T7I		
-	rw	-	-	rw	rw			-	rw	-	-	rw	rw		

Bit	Function
TxI	**Timer/Counter x Input Selection** Timer Mode (TxM = '0') Input Frequency = f_{CPU} / $2^{(<TxI> + 3)}$ See also table below for examples. Counter Mode (TxM = '1'): 000 Overflow/Underflow of GPT1 Timer 3 001 Positive (rising) edge on pin T7IN[1)] 010 Negative (falling) edge on pin T7IN[1)] 011 Any edge (rising and falling) on pin T7IN[1)] 1XX Reserved
TxM	**Timer/Counter x Mode Selection** 0: Timer Mode (Input derived from internal clock) 1: Counter Mode (Input from External Input or T3)
TxR	**Timer/Counter x Run Control** 0: Timer/Counter x is disabled 1: Timer/Counter x is enabled

1) This selection is available for timer T7. Timer T8 will stop at this selection!

The timer run flags T7R and T8R allow the timers to be enabled or disabled. The following description of the timer modes and operation always applies to the enabled state of the timers, that is, the respective run flag is assumed to be set to '1'.

In all modes, the timers always count upwards. The current timer values are accessible for the CPU in the timer registers Tx, which are non-bitaddressable SFRs. When the CPU writes to a register Tx in the state immediately before the respective timer increment or reload is to be performed, the CPU write operation has priority and the increment or reload is disabled to guarantee correct timer operation.

Timer Mode

The bits TxM in SFR T78CON select between timer mode or counter mode for the respective timer. In timer mode (TxM = ‘0’), the input clock for a timer is derived from the internal CPU clock divided by a programmable prescaler. The different options for the prescaler are selected separately for each timer by the bit fields TxI.

The input frequencies f_{Tx} for Tx are determined as a function of the CPU clock as follows, where <TxI> represents the contents of the bit field TxI:

$$f_{Tx} = \frac{f_{CPU}}{2^{(<TxI> + 3)}}$$

When a timer overflows from $FFFF_H$ to 0000_H, it is reloaded with the value stored in its respective reload register TxREL. The reload value determines the period P_{Tx} between two consecutive overflows of Tx as follows:

$$P_{Tx} = \frac{(2^{16} - <TxREL>) \times 2^{(<TxI> + 3)}}{f_{CPU}}$$

After a timer has been started by setting its run flag (TxR) to ‘1’, the first increment will occur within the time interval defined by the selected timer resolution. All further increments occur exactly after the time defined by the timer resolution.

When both timers of the CAPCOM2 unit are to be incremented or reloaded at the same time, T7 is always serviced one CPU clock before T8.

The timer input frequencies, resolution and periods which result from the selected prescaler option in TxI when using a certain CPU clock are listed in **Table 16-2** - **Table 16-4**. The numbers for the timer periods are based on a reload value of 0000_H. Note that some numbers may be rounded to 3 significant digits.

Table 16-2 Timer Input Frequencies, Resolution and Period @ 20 MHz

f_{CPU} = 20 MHz	Timer Input Selection TxI							
	000_B	001_B	010_B	011_B	100_B	101_B	110_B	111_B
Prescaler (1:N)	8	16	32	64	128	256	512	1024
Input Frequency	2.5 MHz	1.25 MHz	625 kHz	312.5 kHz	156.25 kHz	78.125 kHz	39.06 kHz	19.53 kHz
Resolution	400 ns	800 ns	1.6 µs	3.2 µs	6.4 µs	12.8 µs	25.6 µs	51.2 µs
Period	26 ms	52.5 ms	105 ms	210 ms	420 ms	840 ms	1.68 s	3.36 s

Table 16-3 Timer Input Frequencies, Resolution and Period @ 25 MHz

f_{CPU} = 25 MHz	Timer Input Selection TxI							
	000_B	001_B	010_B	011_B	100_B	101_B	110_B	111_B
Prescaler (1:N)	8	16	32	64	128	256	512	1024
Input Frequency	3.125 MHz	1.563 MHz	781.25 kHz	390.63 kHz	195.31 kHz	97.656 kHz	48.828 kHz	24.414 kHz
Resolution	320 ns	640 ns	1.28 µs	2.56 µs	5.12 µs	10.24 µs	20.48 µs	40.96 µs
Period	21 ms	42 ms	84 ms	168 ms	336 ms	672 ms	1.344 s	2.688 s

Table 16-4 Timer Input Frequencies, Resolution and Period @ 33 MHz

f_{CPU} = 33 MHz	Timer Input Selection TxI							
	000_B	001_B	010_B	011_B	100_B	101_B	110_B	111_B
Prescaler (1:N)	8	16	32	64	128	256	512	1024
Input Frequency	4.125 MHz	2.063 MHz	1.031 MHz	515.63 kHz	257.81 kHz	128.91 kHz	64.453 kHz	32.227 kHz
Resolution	242 ns	485 ns	970 ns	1.94 µs	3.88 µs	7.76 µs	15.52 µs	31.03 µs
Period	15.89 ms	31.78 ms	63.55 ms	127.10 ms	254.20 ms	508.40 ms	1.017 s	2.034 s

Counter Mode

The bits TxM in SFR T78CON select between timer mode or counter mode for the respective timer. In Counter mode (TxM = ‘1’) the input clock for a timer can be derived from the overflows/underflows of timer T3 in block GPT1. Additionally, timer T7 can be clocked by external events. Either a positive, a negative, or both a positive and a negative transition at pin T7IN (alternate port input function) can be selected to cause an increment of T7.

When T8 is programmed to run in counter mode, bit field TxI is used to enable the overflows/underflows of timer T3 as the count source. This is the only option for T8 and it is selected by the combination TxI = 000_B. When bit field TxI is programmed to any other valid combination, the respective timer will stop.

When T7 is programmed to run in counter mode, bit field TxI is used to select the count source and transition (if the source is the input pin) which should cause a count trigger (see description of T78CON for the possible selections).

Note: To use pin T7IN as external count input pin, the respective port pin must be configured as input: the corresponding direction control bit must be cleared (DPx.y = ‘0’).
If the respective port pin is configured as output, the associated timer may be clocked by modifying the port output latches Px.y via software, such as for testing purposes.

The maximum external input frequency to T7 in counter mode is $f_{CPU}/16$. To ensure that a signal transition is properly recognized at the timer input, an external count input signal should be held for at least eight CPU clock cycles before it changes its level again. The incremented count value appears in SFR T7 within eight CPU clock cycles after the signal transition at pin T7IN.

Reload

In both modes, a reload of a timer with the 16-bit value stored in its associated reload register is performed each time a timer would overflow from $FFFF_H$ to 0000_H. In such a case, the timer does not wrap around to 0000_H, but rather is reloaded with the contents of the respective reload register TxREL. The timer then resumes incrementing starting from the reloaded value.

The reload registers TxREL are not bit-addressable.

16.2 CAPCOM2 Unit Timer Interrupts

When a timer overflows, the corresponding timer interrupt request flag TxIR for the respective timer will be set. This flag can be used to generate an interrupt or to trigger a PEC service request, when enabled by the respective interrupt enable bit TxIE.

Each timer has its own bit-addressable interrupt control register (TxIC) and its own interrupt vector (TxINT). The organization of the interrupt control registers TxIC is identical to the other interrupt control registers.

T7IC
CAPCOM T7 Intr. Ctrl. Reg. **ESFR ($F17A_H/BE_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								T7IR	**T7IE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

T8IC
CAPCOM T8 Intr. Ctrl. Reg. **ESFR ($F17C_H/BF_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								T8IR	**T8IE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

16.3 Capture/Compare Registers

The 16-bit capture/compare registers CC16 through CC31 are used as data registers for capture or compare operations with respect to timers T7 and T8. The capture/compare registers are not bit-addressable.

Each of the registers CCx may be individually programmed for capture mode or for one of four different compare modes. Each register may be allocated individually to one of the two timers T7 or T8, respectively. A special combination of compare modes additionally allows implementation of a ‘double-register’ compare mode. When capture or compare operation is disabled for one of the CCx registers, it may be used for general purpose variable storage.

Capture/Compare Mode Registers for the CAPCOM2 Unit

The functions of the 16 capture/compare registers are controlled by four, identically organized bit-addressable 16-bit mode control registers named CCM4 … CCM7 (see description below). Each register contains bits for mode selection and timer allocation of four capture/compare registers.

Capture/Compare Mode Registers for the CAPCOM2 Unit (CC16 … CC32)

CCM4
CAPCOM Mode Ctrl. Reg. 4 **SFR ($FF22_H/91_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC 19	CCMOD19			ACC 18	CCMOD18			ACC 17	CCMOD17			ACC 16	CCMOD16		
rw	rw			rw	rw			rw	rw			rw	rw		

CCM5
CAPCOM Mode Ctrl. Reg. 5 **SFR ($FF24_H/92_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC 23	CCMOD23			ACC 22	CCMOD22			ACC 21	CCMOD21			ACC 20	CCMOD20		
rw	rw			rw	rw			rw	rw			rw	rw		

CCM6
CAPCOM Mode Ctrl. Reg. 6 **SFR ($FF26_H/93_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC 27	CCMOD27			ACC 26	CCMOD26			ACC 25	CCMOD25			ACC 24	CCMOD24		
rw	rw			rw	rw			rw	rw			rw	rw		

CCM7
CAPCOM Mode Ctrl. Reg. 7 **SFR ($FF28_H/94_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC 31	CCMOD31			ACC 30	CCMOD30			ACC 29	CCMOD29			ACC 28	CCMOD28		
rw	rw			rw	rw			rw	rw			rw	rw		

Bit	Function
CCMODx	**Mode Selection for Capture/Compare Register CCx** The available capture/compare modes are listed in **Table 16-5**.
ACCx	**Allocation Bit for Capture/Compare Register CCx** 0: CCx allocated to Timer T7 1: CCx allocated to Timer T8

Table 16-5 Selection of Capture Modes and Compare Modes

CCMODx	Selected Operating Mode
0 0 0	**Disable Capture and Compare Modes** The respective CAPCOM2 register may be used for general variable storage.
0 0 1	**Capture on Positive Transition (Rising Edge) at Pin CCxIO**
0 1 0	**Capture on Negative Transition (Falling Edge) at Pin CCxIO**
0 1 1	**Capture on Positive and Negative Transition (Both Edges) at Pin CCxIO**
1 0 0	**Compare Mode 0: Interrupt Only** Several interrupts per timer period. Enables double-register compare mode for registers CC24 … CC27.
1 0 1	**Compare Mode 1: Toggle Output Pin on each Match** Several compare events per timer period. This mode is required for double-register compare mode for registers CC16 … CC19.
1 1 0	**Compare Mode 2: Interrupt Only** Only one interrupt per timer period.
1 1 1	**Compare Mode 3: Set Output Pin on each Match** Reset output pin on each timer overflow; Only one interrupt per timer period.

The descriptions of the capture and compare modes are valid for all capture/compare channels; so, the registers, bits, and pins are referenced only by the placeholder ‘x’.

Note: Only capture/compare channels 16 … 19 and 24 … 27 are connected to pins and interrupt nodes.
A capture or compare event on channel 27 may be used to trigger a channel injection on the C164CI's A/D converter, if enabled.

16.4 Capture Mode

In response to an external event, the contents of the associated timer (T7 or T8, depending on the state of the allocation control bit ACCx) are latched into the respective capture register CCx. The external event causing a capture can be programmed to be a positive, a negative, or both a positive and a negative transition at the respective external input pin CCxIO.

The triggering transition is selected by the mode bits CCMODx in the respective CAPCOM mode control register. In any case, the event causing a capture will also set the respective interrupt request flag CCxIR, which can cause an interrupt or a PEC service request, when enabled.

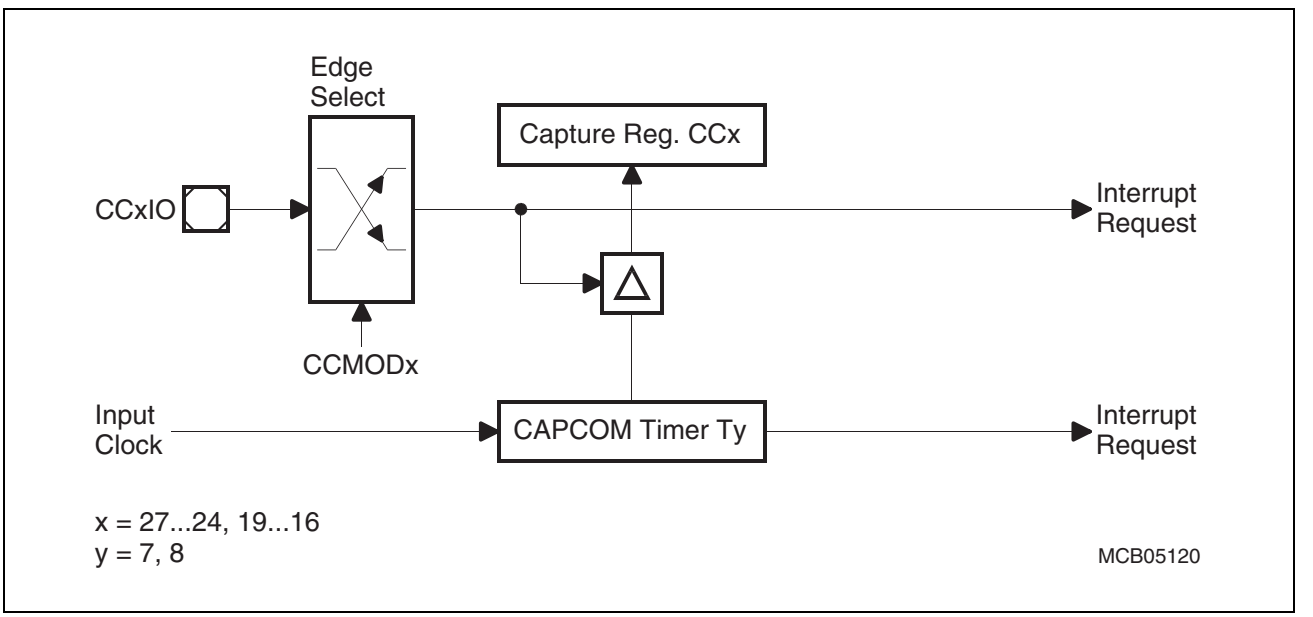


Figure 16-5 Capture Mode Block Diagram

To use the respective port pin as external capture input pin CCxIO for capture register CCx, this port pin must be configured as input; that is, the corresponding direction control bit must be set to ‘0’. To ensure that a signal transition is properly recognized, an external capture input signal should be held for at least eight CPU clock cycles before changing its level.

During these eight CPU clock cycles, the capture input signals are scanned sequentially. When a timer is modified or incremented in this process, the new timer contents will already be captured for the remaining capture registers within the current scanning sequence.

Note: When the timer modification can generate an overflow the capture interrupt routine should check if the timer overflow was serviced during these 8 CPU clock cycles.

If pin CCxIO is configured as output, the capture function may be triggered by modifying the corresponding port output latch via software, for testing purposes, for example.

16.5 Compare Modes

The compare modes allow triggering of events (interrupts and/or output signal transitions) with minimum software overhead. In all compare modes, the 16-bit value stored in compare register CCx (in the following also referred to as ‘compare value’) is continuously compared with the contents of the allocated timer (T7 or T8). If the current timer contents match the compare value, an appropriate output signal, based on the selected compare mode, can be generated at the corresponding output pin CCxIO and the associated interrupt request flag CCxIR is set, which can generate an interrupt request (if enabled).

As for capture mode, the compare registers are also processed sequentially in compare mode. When any two compare registers are programmed to the same compare value, their corresponding interrupt request flags will be set to ‘1’ and the selected output signals will be generated within eight CPU clock cycles after the allocated timer is incremented to the compare value. Further compare events on the same compare value are disabled[1)] until the timer is incremented again or is written to by software. After a reset, compare events for register CCx will become enabled only if the allocated timer has been incremented or written to by software and one of the compare modes described in the following sections has been selected for this register.

The different compare modes which can be programmed for a given compare register CCx are selected by the mode control field CCMODx in the associated capture/compare mode control register. Each of the compare modes, including the special ‘double register’ mode, is discussed in detail in the following sections.

Compare Mode 0

This is an interrupt-only mode which can be used for software timing purposes. Compare mode 0 is selected for a given compare register CCx by setting bit field CCMODx of the corresponding mode control register to ‘100_B’.

In this mode, the interrupt request flag CCxIR is set each time a match is detected between the contents of compare register CCx and the allocated timer. Several of these compare events are possible within a single timer period, when the compare value in register CCx is updated during the timer period. The corresponding port pin CCxIO is not affected by compare events in this mode and can be used as general purpose IO pin.

If compare mode 0 is programmed for one of the registers CC24 … CC27, the double-register compare mode becomes enabled for this register if the corresponding bank 1 register is programmed to compare mode 1 (see **“Double-Register Compare Mode” on Page 16-19**).

1) Compare events are detected sequentially, where a sequence (checking 8 times 2 channels each) takes 8 CPU clock cycles. Even if more sequences are executed before the timer increments (lower timer frequency) a given compare value only results in one single compare event.

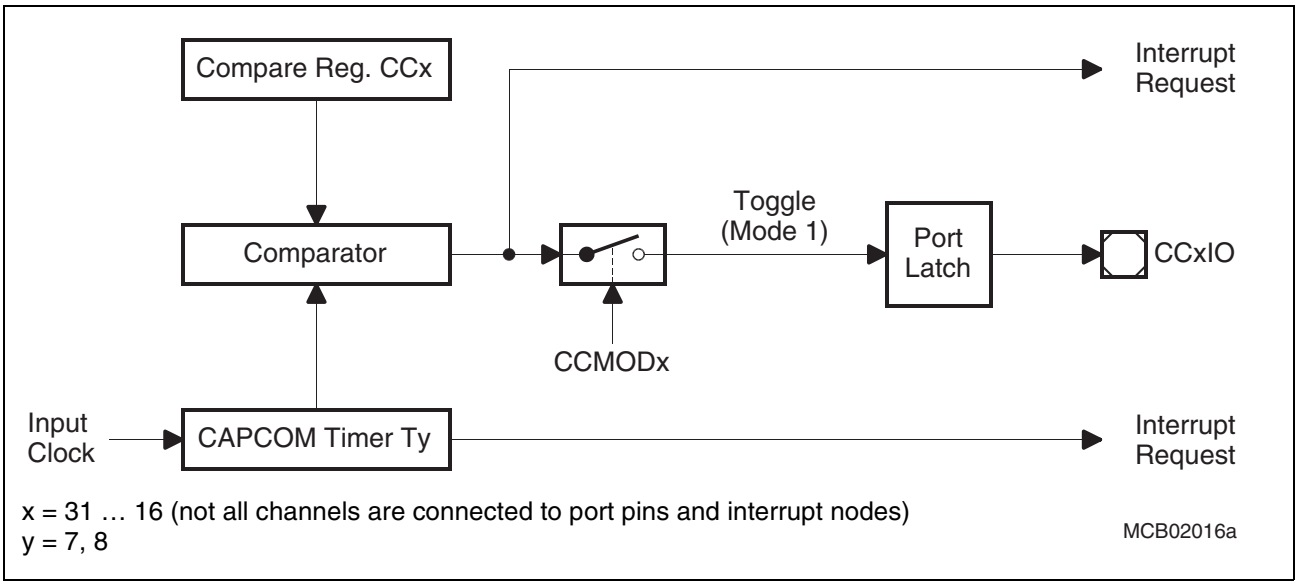


Figure 16-6 Compare Mode 0 and 1 Block Diagram

Note: The port latch and pin remain unaffected in compare mode 0.

In the example shown in **Figure 16-7**, the compare value in register CCx is modified from cv1 to cv2 after compare events #1 and #3, and from cv2 to cv1 after events #2 and #4, etc. This results in periodic interrupt requests from timer Ty, and in interrupt requests from register CCx which occur at the time specified by the user through cv1 and cv2.

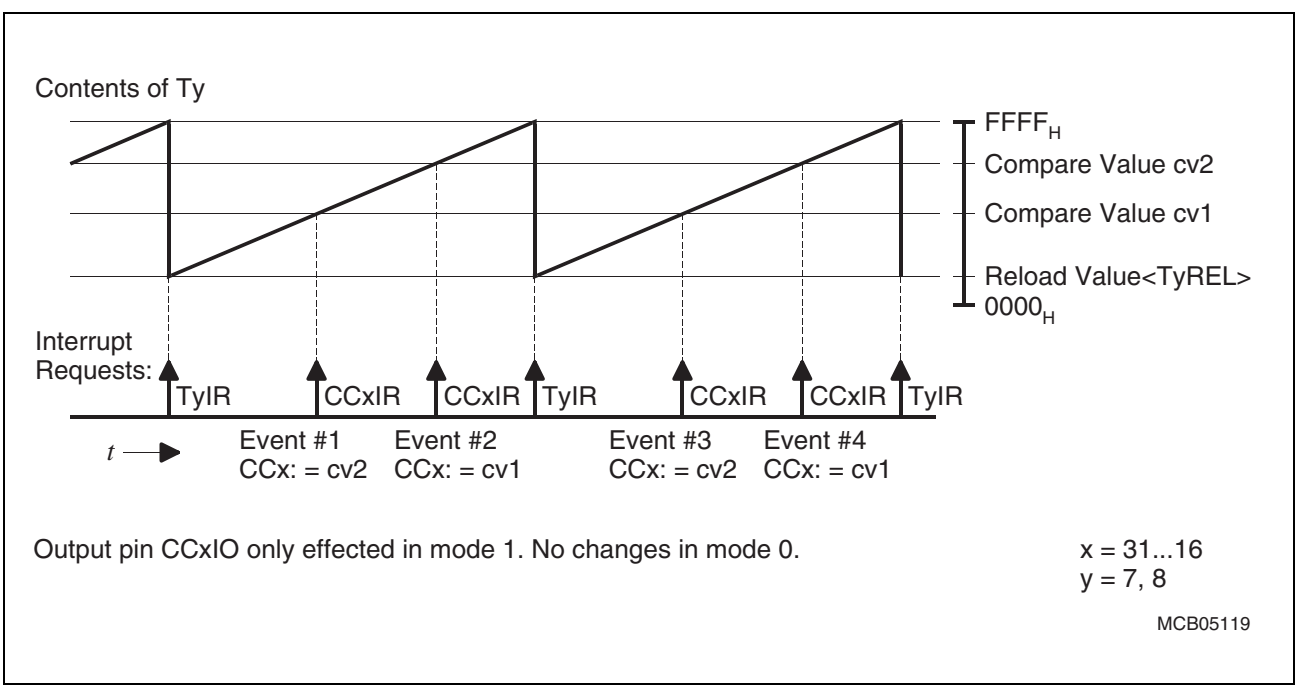


Figure 16-7 Timing Example for Compare Modes 0 and 1

Compare Mode 1

Compare mode 1 is selected for register CCx by setting bit field CCMODx of the corresponding mode control register to '101_B'.

When a match between the content of the allocated timer and the compare value in register CCx is detected in this mode, interrupt request flag CCxIR is set to '1', and the corresponding output pin CCxIO (alternate port output function) is toggled. For this purpose, the state of the respective port output latch (not the pin) is read, inverted, and then written back to the output latch.

Compare mode 1 allows several compare events within a single timer period. An overflow of the allocated timer has no effect on the output pin, nor does it disable or enable further compare events.

In order to use the respective port pin as compare signal output pin CCxIO for compare register CCx in compare mode 1, this port pin must be configured as output, i.e. the corresponding direction control bit must be set to '1'. With this configuration, the initial state of the output signal can be programmed or its state can be modified at any time by writing to the port output latch.

In compare mode 1 the port latch is toggled upon each compare event (see **Figure 16-7**).

If compare mode 1 is programmed for one of the registers CC16 … CC19, the double-register compare mode becomes enabled for this register if the corresponding bank 2 register is programmed to compare mode 0 (see **"Double-Register Compare Mode" on Page 16-19**).

Note: If the port output latch is written to by software at the same time it would be altered by a compare event, the software write will have priority. In this case, the hardware-triggered change will not become effective.
Only capture/compare channels 16 … 19 and 24 … 27 are connected to pins.

Compare Mode 2

Compare mode 2 is an interrupt-only mode similar to compare mode 0; but, only one interrupt request per timer period will be generated. Compare mode 2 is selected for register CCx by setting bit field CCMODx of the corresponding mode control register to '110_B'.

When a match is detected in compare mode 2 for the first time within a timer period, the interrupt request flag CCxIR is set to '1'. The corresponding port pin is not affected and can be used for general purpose IO. However, after the first match has been detected in this mode, all further compare events within the same timer period are disabled for compare register CCx until the allocated timer overflows. This means that after the first match, even when the compare register is reloaded with a value higher than the current timer value, no compare event will occur until the next timer period.

In the example shown in **Figure 16-8**, the compare value in register CCx is modified from cv1 to cv2 after compare event #1. Compare event #2, however, will not occur until the next period of timer Ty.

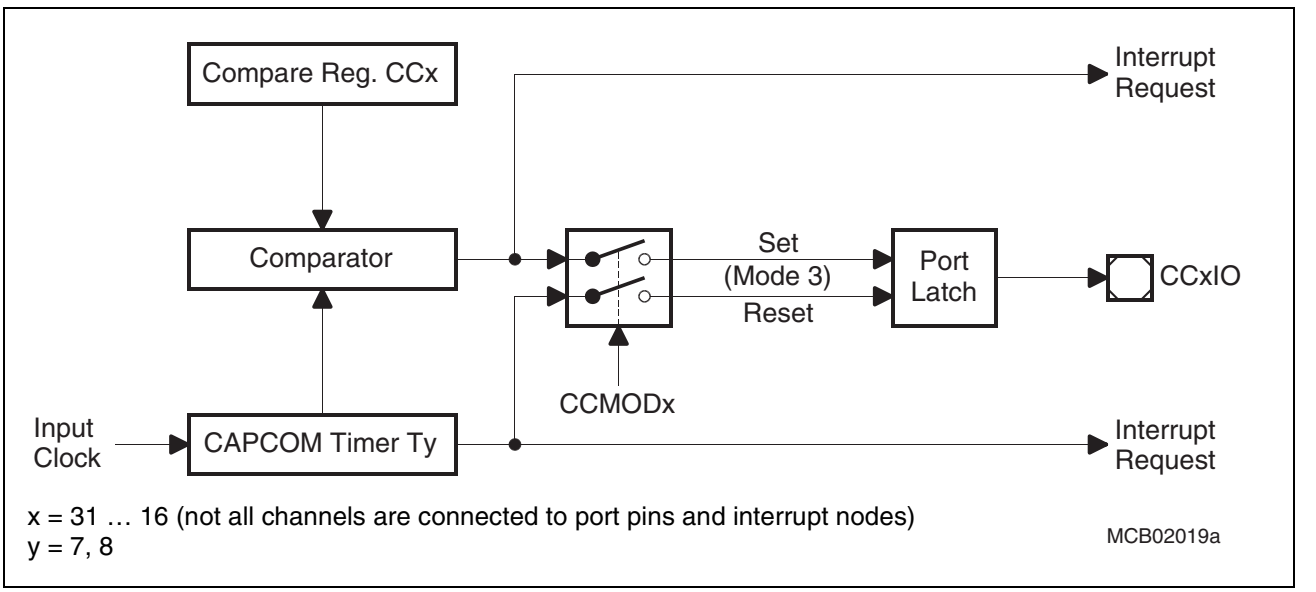


Figure 16-8 Compare Mode 2 and 3 Block Diagram

Note: The port latch and pin remain unaffected in compare mode 2.

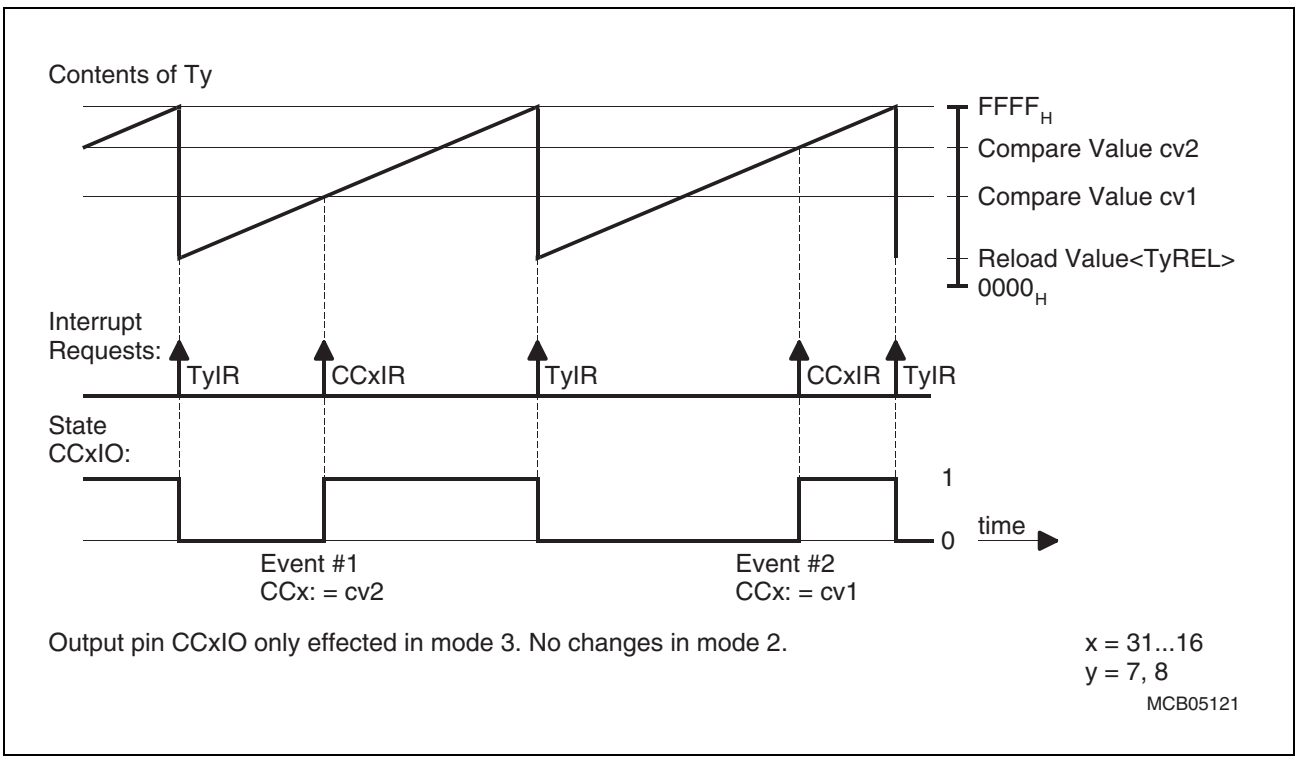


Figure 16-9 Timing Example for Compare Modes 2 and 3

Compare Mode 3

Compare mode 3 is selected for register CCx by setting bit field CCMODx of the corresponding mode control register to '111_B'. In compare mode 3, only one compare event will be generated per timer period.

When the first match within the timer period is detected, the interrupt request flag CCxIR is set to '1' and the output pin CCxIO (alternate port function) will be set to '1'. The pin will be reset to '0', when the allocated timer overflows.

If a match was found for register CCx in this mode, all further compare events during the current timer period are disabled for CCx until the corresponding timer overflows. If, after a match was detected, the compare register is reloaded with a new value, this value will not become effective until the next timer period.

To use the respective port pin as compare signal output pin CCxIO for compare register CCx in compare mode 3, this port pin must be configured as output: the corresponding direction control bit must be set to '1'. With this configuration, the initial state of the output signal can be programmed or its state can be modified at any time by writing to the port output latch.

In compare mode 3, the port latch is set upon a compare event and cleared upon a timer overflow (see **Figure 16-9**).

However, when compare value and reload value for a channel are equal, the respective interrupt requests will be generated. Only the output signal is not changed in this case (set and clear would coincide).

Note: If the port output latch is written to by software at the same time it would be altered by a compare event, the software write will have priority. In such a case, the hardware-triggered change will not become effective.
Only capture/compare channels 16 … 19 and 24 … 27 are connected to pins.

Double-Register Compare Mode

In double-register compare mode, two compare registers work together to control one output pin. This mode is selected by a special combination of modes for these two registers.

For double-register mode, the 16 capture/compare registers of the CAPCOM2 unit are regarded as two banks of 8 registers each. Registers CC16 … CC23 form bank 1 while registers CC24 … CC31 form bank 2 (respectively). For double-register mode, a bank 1 register and a bank 2 register form a register pair. Both registers of this register pair operate on the pin associated with the bank 1 register (pins CC16IO … CC19IO are available).

The relationships between the bank 1 and bank 2 registers of a pair and the affected output pins for double-register compare mode are listed in **Table 16-6**.

Table 16-6 Register Pairs for Double-Register Compare Mode

CAPCOM2 Unit		
Register Pair		Associated Output Pin
Bank 1	Bank 2	
CC16	CC24	CC16IO
CC17	CC25	CC17IO
CC18	CC26	CC18IO
CC19	CC27	CC19IO
CC23 … CC20	CC31 … CC28	-----

The double-register compare mode can be programmed individually for each register pair. To enable double-register mode, the respective bank 1 register (see **Table 16-6**) must be programmed to compare mode 1 and the corresponding bank 2 register (see **Table 16-6**) must be programmed to compare mode 0.
If the respective bank 1 compare register is disabled or programmed for a mode other than mode 1 the corresponding bank 2 register will operate in compare mode 0 (interrupt-only mode).

In the following example, a bank 2 register (programmed to compare mode 0) will be referred to as CCz while the corresponding bank 1 register (programmed to compare mode 1) will be referred to as CCx.

When a match is detected for one of the two registers in a register pair (CCx or CCz) the associated interrupt request flag (CCxIR or CCzIR) is set to ‘1’ and pin CCxIO corresponding to bank 1 register CCx is toggled. The generated interrupt always corresponds to the register that caused the match.

Note: If a match occurs simultaneously for register CCx and register CCz of the register pair, pin CCxIO will be toggled only once but two separate compare interrupt requests will be generated: one for vector CCxINT and one for vector CCzINT.

To use the respective port pin as compare signal output pin CCxIO for compare register CCx in double-register compare mode, this port pin must be configured as output: the corresponding direction control bit must be set to ‘1’. With this configuration, the output pin has the same characteristics as in compare mode 1.

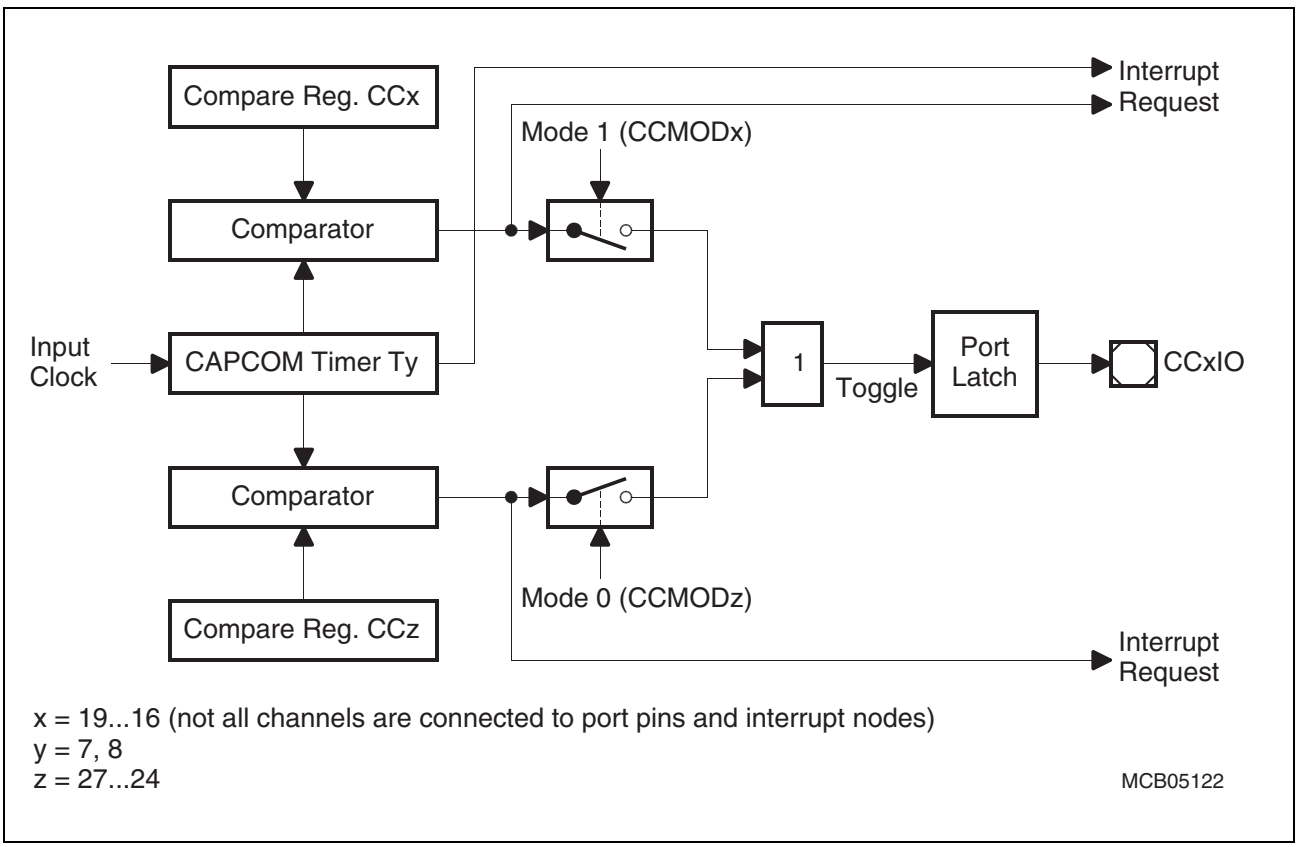


Figure 16-10 Double-Register Compare Mode Block Diagram

In this configuration example, the same timer allocation was chosen for both compare registers, but each register may also be allocated individually to one of the two timers of the CAPCOM2 unit. In the timing example for this compare mode (**Figure 16-11**), the compare values in registers CCx and CCz are not modified.

Note: The pins CCzIO (which do not serve for double-register compare mode) may be used for general purpose IO.

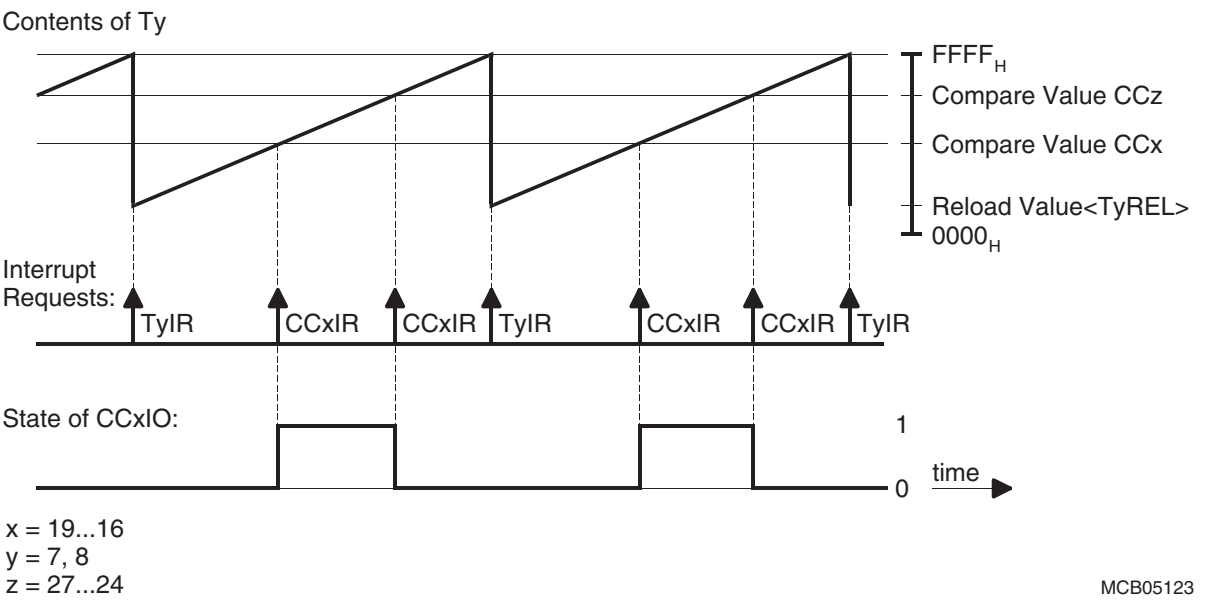


Figure 16-11 Timing Example for Double-Register Compare Mode

Note: Double-Register Compare Mode is reasonable only on channel pairs with an associated output pin.

16.6 Capture/Compare Interrupts

Upon a capture or compare event, the interrupt request flag CCxIR for the respective capture/compare register CCx is set to ‘1’. This flag can be used to generate an interrupt or to trigger a PEC service request when enabled by the interrupt enable bit CCxIE.

Capture interrupts can be regarded as external interrupt requests with the additional feature of recording the time at which the triggering event occurred (see also **Section 5.8**).

Each of the 8 capture/compare registers with an associated I/O-pin has its own bit-addressable interrupt control register (CC27IC … CC24IC, CC19IC … CC16IC) and its own interrupt vector (CC27INT … CC24INT, CC19INT … CC16INT). These registers are organized the same way as all other interrupt control registers. The basic register layout is shown here. **Table 16-7** lists the associated addresses.

CCxIC
CAPCOM Intr. Ctrl. Reg. **ESFR (*See* Table 16-7)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								CCx IR	**CCx IE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

Table 16-7 CAPCOM2 Unit Interrupt Control Register Addresses

CAPCOM2 Unit		
Register Name	**Address**	**Register Space**
CC16IC	$F160_H$ / $B0_H$	ESFR
CC17IC	$F162_H$ / $B1_H$	ESFR
CC18IC	$F164_H$ / $B2_H$	ESFR
CC19IC	$F166_H$ / $B3_H$	ESFR
CC24IC	$F170_H$ / $B8_H$	ESFR
CC25IC	$F172_H$ / $B9_H$	ESFR
CC26IC	$F174_H$ / BA_H	ESFR
CC27IC	$F176_H$ / BB_H	ESFR

17 Capture/Compare Unit CAPCOM6

The CAPCOM6 unit of the C164CI has been designed for applications which require digital signal generation and/or event capturing, such as pulse width modulation (PWM) or measuring. The C164CI supports generation and control of timing sequences on up to three 16-bit capture/compare channels plus one 10-bit compare channel.

In compare mode the CAPCOM6 unit provides two output signals per 16-bit channel which may have inverted polarity and non-overlapping pulse transitions. The 10-bit compare channel can generate a single PWM output signal and is further used to modulate the capture/compare output signals. The compare timers T12 (16-bit) and T13 (10-bit) are free running timers which are clocked by the prescaled CPU clock.

For motor control applications both subunits may generate versatile multi-channel PWM signals which are basically either controlled by compare timer T12 or by a typical hall sensor pattern at the interrupt inputs. This operating mode is called block commutation (available only in devices with a full function CAPCOM6).

In capture mode the contents of compare timer T12 are stored in the capture registers upon a programmable signal transition at pins CC6x.

From the programmer's point of view, the term 'CAPCOM unit' refers to a set of SFRs which are associated with this peripheral, including the port pins which may be used for alternate input/output functions and their direction control bits.

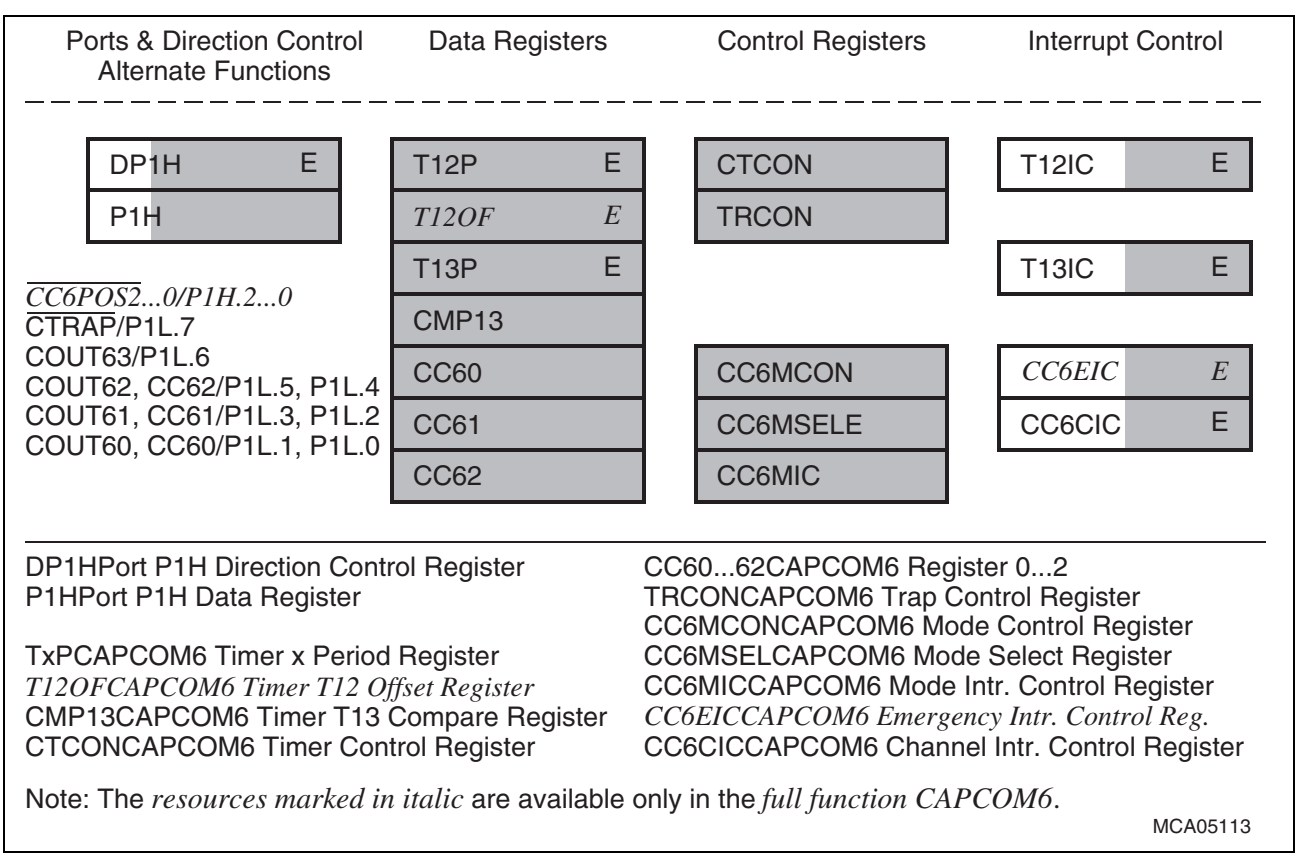
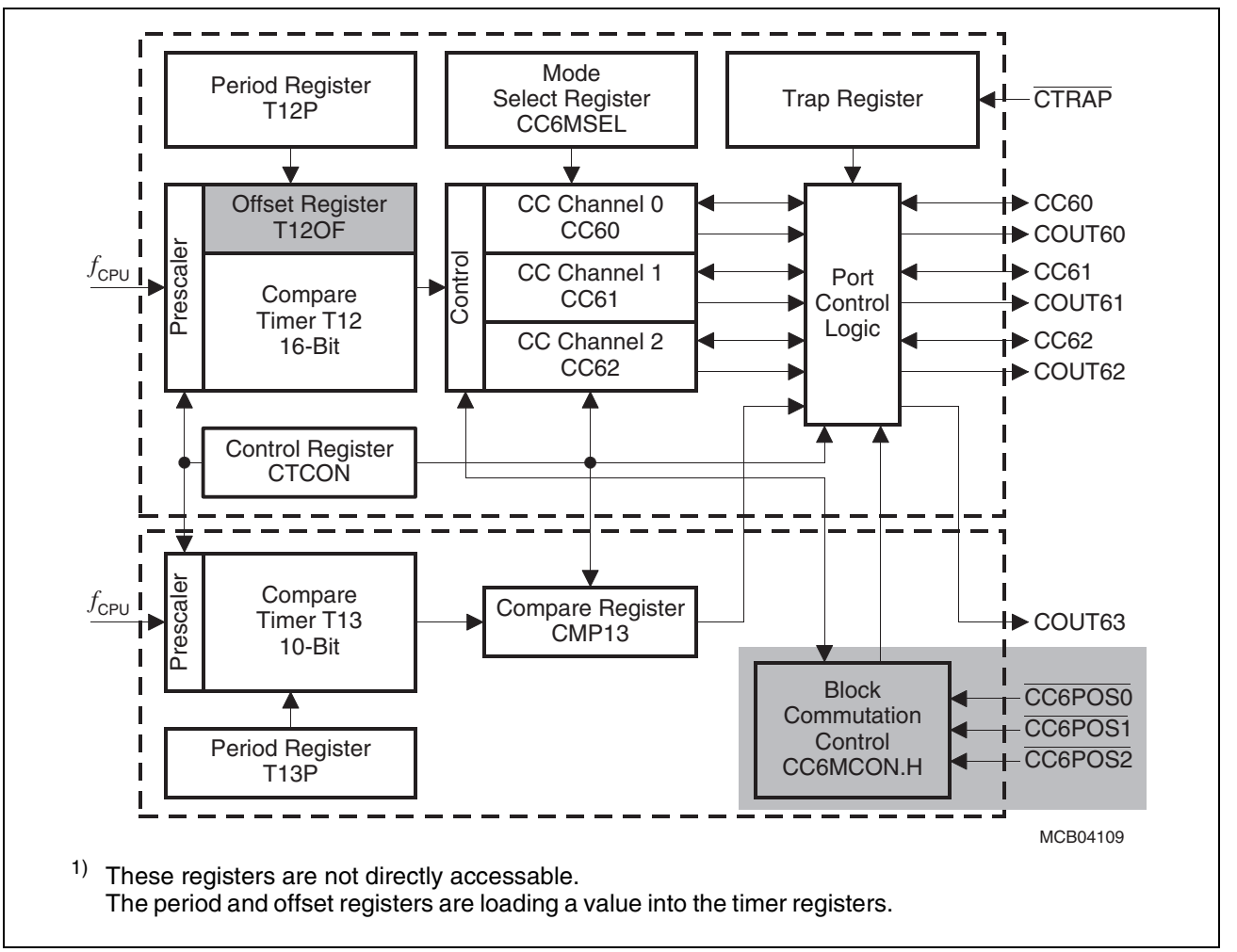


Figure 17-1 SFRs and Port Pins Associated with the CAPCOM6 Unit

The three 16-bit capture/compare channels are driven via timer T12 and each can control two output lines (see Port Control Logic). The offset register T12OF (full function module only) allows shifting of the switching points of the COUT6x output line of each channel by shifting the respective compare value.

The 10-bit compare channel is driven via timer T13 and can control one output line.

Additional control logic allows the capture/compare channel outputs to be combined with the compare channel output or with external signals. Thus flexible and complex output patterns can be generated automatically, with very little or no CPU action.



1) These registers are not directly accessable.
The period and offset registers are loading a value into the timer registers.

Figure 17-2 CAPCOM6 Block Diagram

Two basic operating modes are supported:

- Edge-Aligned Mode
- Center-Aligned Mode

In **Edge Aligned Mode** the compare timer counts up starting at 0000_H. Upon reaching the period value stored in register TxP the timer is cleared and repeats counting up. At this time the output signals are also switched to their passive state. Edge aligned mode is supported by both compare timers, T12 and T13.

In **Center Aligned Mode** the compare timer T12 counts up starting at 0000_H. Upon reaching the period value stored in register T12P the count direction is reversed and the timer counts down. The output signals are switched to their active/passive state upon a match with the compare value while counting up/down. Center aligned mode is supported by compare timer T12 only.

The compare timers T12 and T13 are free running timers which are clocked with a programmable frequency of f_{CPU} to $f_{CPU}/128$.

The respective output signals are changed (if appropriate) when the timer reaches the programmed compare value. For switching the output signals COUT60 … COUT62 the contents of the timer plus the offset value are compared against the compare value.

Timer T12 can operate in either edge aligned or center aligned PWM mode (see **Figure 17-3**), with or without a constant edge delay (a or b in **Figure 17-3**).
Timer T13 can operate in edge aligned mode without edge delay.

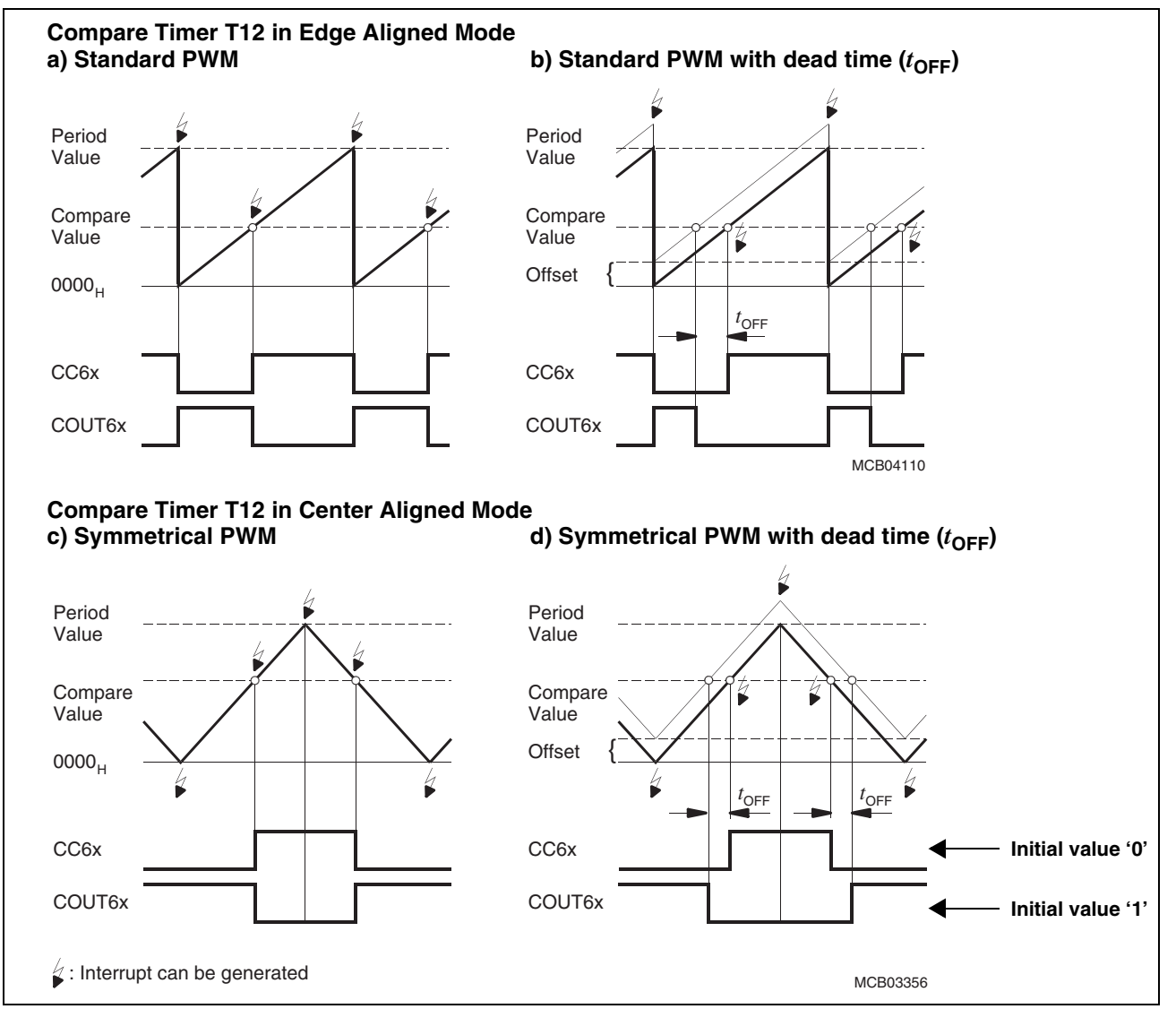


Figure 17-3 CAPCOM6 Basic Operating Modes

17.1 Output Signal Level Control

The output signals generated by the CAPCOM6 unit are characterized by the duration of their active and passive phases which define the signals' period and duty cycle. In order to adapt these output signals to the requirements of a specific application, the logic level of the passive state for each signal can be selected via register CC6MCON.

When using the trap function, the outputs are switched to their trap level upon the activation of an external (emergency) signal. The trap level is defined via the respective port output latches.

Note: Changing the state levels during operation of CAPCOM6 will immediately affect the output signals. It is recommended that the output levels be defined during initialization before the output signals are assigned and before the CAPCOM6 unit is started.

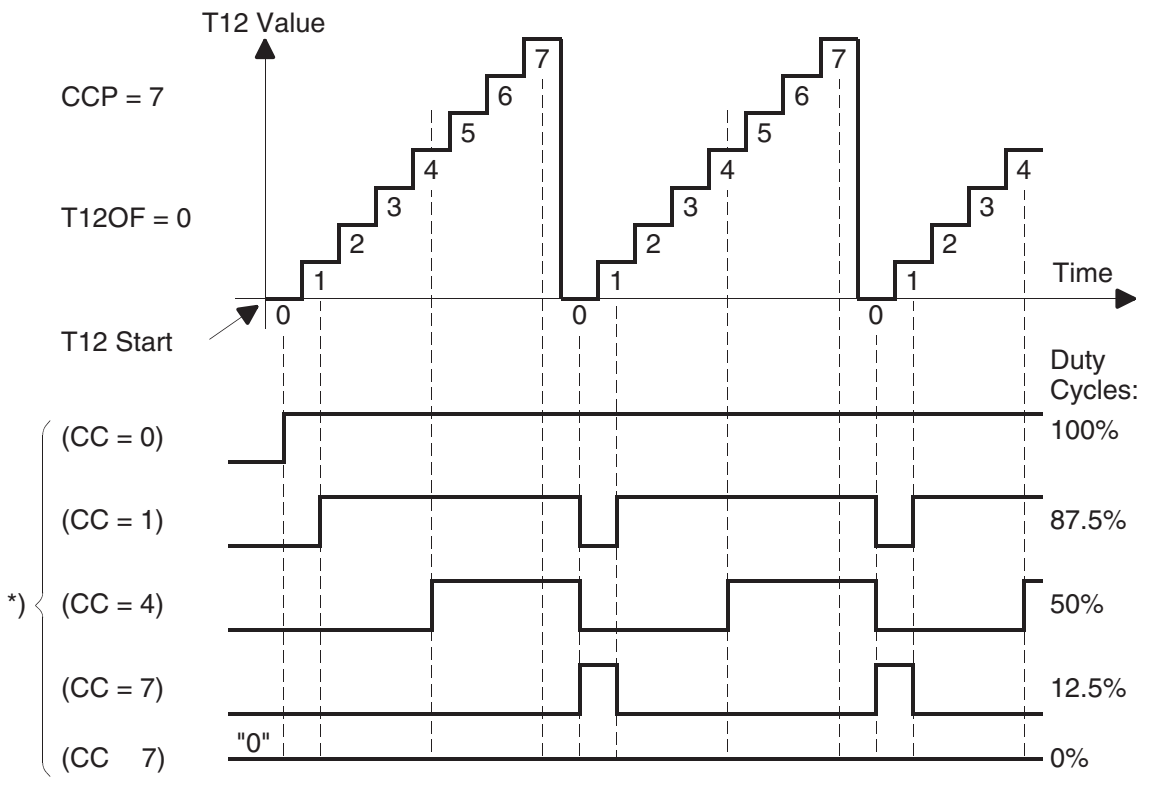
In burst and multi-channel modes the signals generated by the capture/compare channels may additionally be modulated by the signal generated by the 10-bit compare channel. Optionally, this compare channel signal may be inverted before modulating the other outputs. The compare channel's signal may be output on pin COUT63. This output function is enabled by bit ECT13O in register CTCON. If the output function is disabled COUT63 drives the defined passive level.

Note: Trap function and multi-channel modes are available in the full function module only.

17.2 Edge Aligned Mode

The compare timer counts up starting at 0000_H. When the timer contents match the respective compare value in register CC6x the associated output signal is switched to its active state. Upon reaching the period value stored in register TxP the timer is cleared and repeats counting up. At this time also the output signals are switched to their passive state.

In **Figure 17-4** the selected edge offset is zero, therefore the output signal refers to CC6x and/or COUT6x.



CC = compare value in registers CC6x
CCP = period value in register T12P T13P
*) Output signal at pin CC6x or COUT6x selected as **active high**. For an active low output the signals would appear inverted.

Figure 17-4 Operation in Edge Aligned Mode

The example above shows how to generate PWM output signals with duty cycles between 0% and 100%, including the corner values. The duty cycle directly corresponds to the programmed compare value. The indicated output signals can be output on the respective pin CC6x or COUT6x, or both. The pin allocation is controlled via bitfields CMSELx in register CC6MSEL. Register CC6MCON selects the passive level for enabled outputs. The example above uses active high signals: the passive level is low (the associated select bit is ‘0’).

In **Figure 17-5** a non-zero offset value is used. In this case the compare value is not compared with the timer contents directly, but rather with timer contents plus offset. As a consequence the active edge of signal COUT6x is shifted against CC6x.

Figure 17-5 shows some of the output signals that can be generated (compare value = '3'):

a) Standard output signal, using T12 directly, active high.
b) Shifted output signal, using T12 + T12OF, active high.
c) Same signal as b), but active low.
d) 0% output signal, compare value in CC6x > T12P + T12OF.
e) 100% output signal, compare value in CC6x = T12OF.

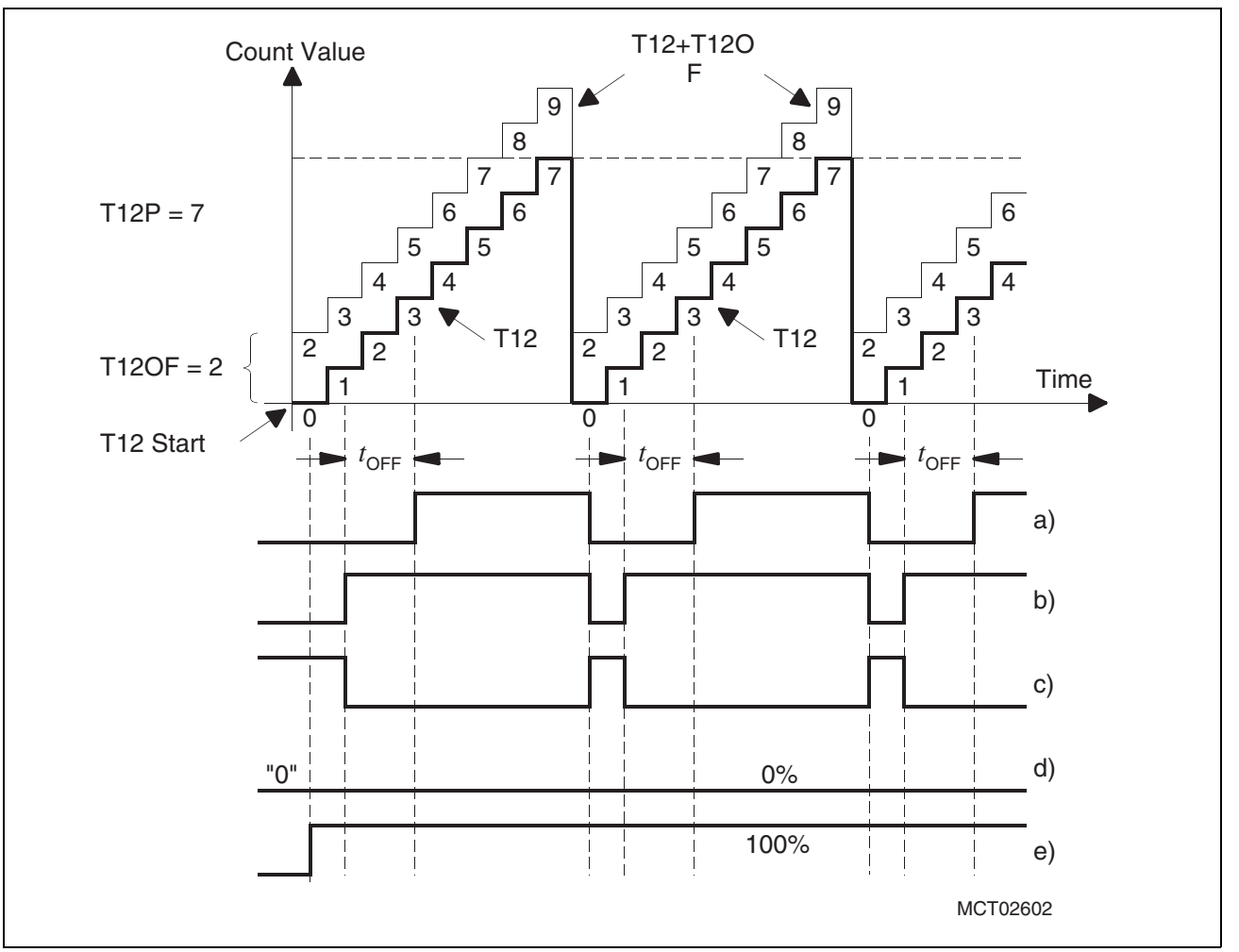


Figure 17-5 Operation with Non-zero Offset

Note: Offset operation is only available in the full function module and is possible for the three capture/compare channels on timer T12 only. The compare channel on timer T13 does not provide an offset register and has no second output signal.

17.3 Center Aligned Mode

The three capture/compare channels associated with T12 may operate in center aligned mode. The compare timer T12 counts up starting at 0000_H. When the timer contents match the respective compare value in register CC6x, the associated output signal CC6x is switched to its **active** state (while counting **up**). Upon reaching the period value stored in register T12P the count direction is reversed and the timer counts down. When the timer contents match the respective compare value in register CC6x, the associated output signal CC6x is switched to its **passive** state (while counting **down**).

The output signals COUT6x are switched upon matches of register CC6x with T12 + T12OF. Non-zero offset values shift the COUT6x edges symmetrically against the CC6x edges (see **Figure 17-6**). This allows the generation of non-overlapping signal pairs CC6x/COUT6x with arbitrary active levels. These signal pairs may e.g. be used to drive the high and low side switches of a power bridge without the risk of a branch shortcut (prevented by the programmable dead-time t_{OFF}, see **Figure 17-6**).

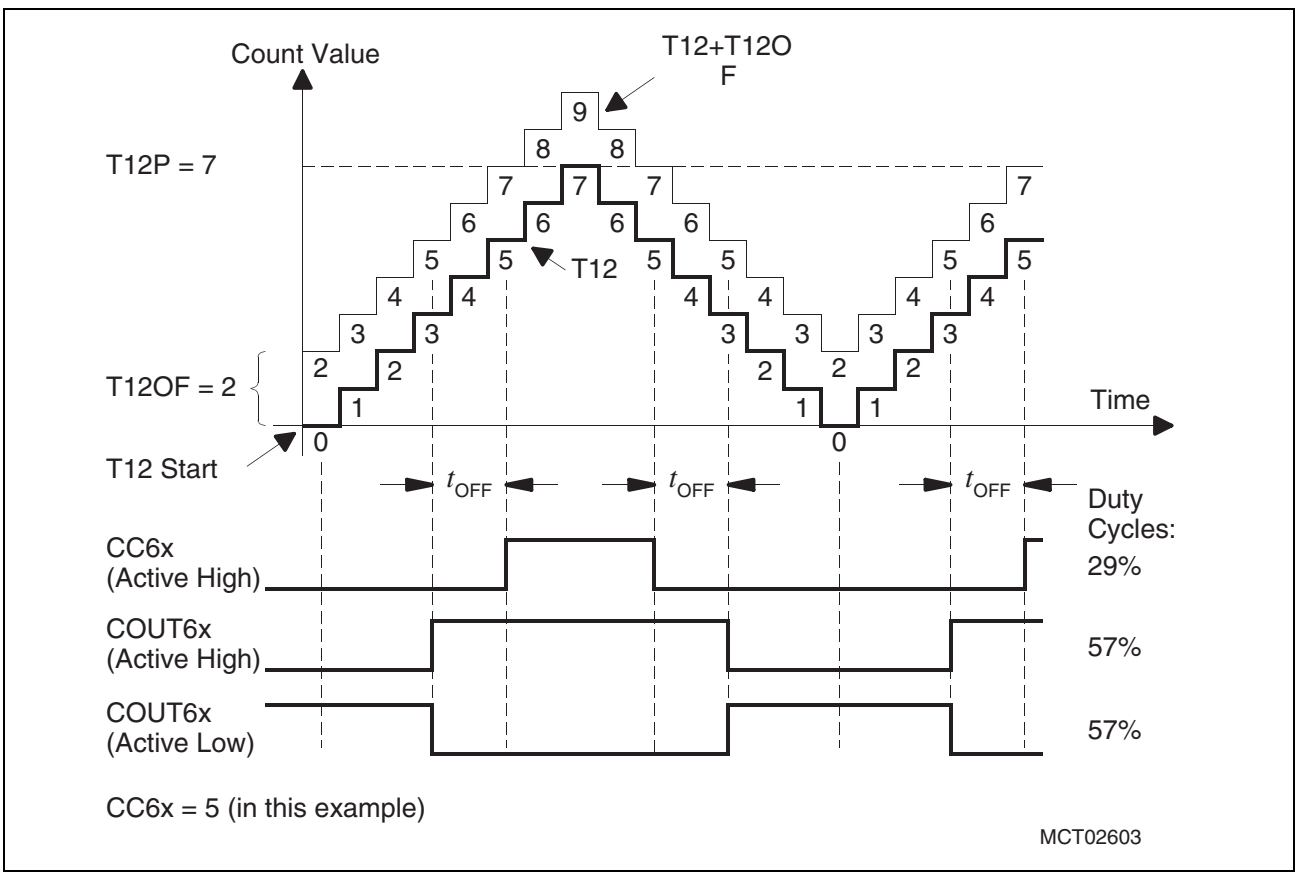


Figure 17-6 Operation in Center Aligned Mode

Note: In order to generate correct dead times for PWM signals, the offset value stored in T12OF must be lower than the value stored in the compare registers.
The offset value affects all COUT6x outputs.
Dead time generation is available only in the full function module.

17.3.1 Timing Relationships

The resolution of the compare timers depends on the selected internal clock frequency. The period range of the output signals in turn depends on the actual timer resolution (minimum value) and on the timer and period values (maximum value). **Table 17-1** lists the respective values of both compare timers for the possible clock selections.

Due to internal operation the minimum possible output period is two internal clock cycles.

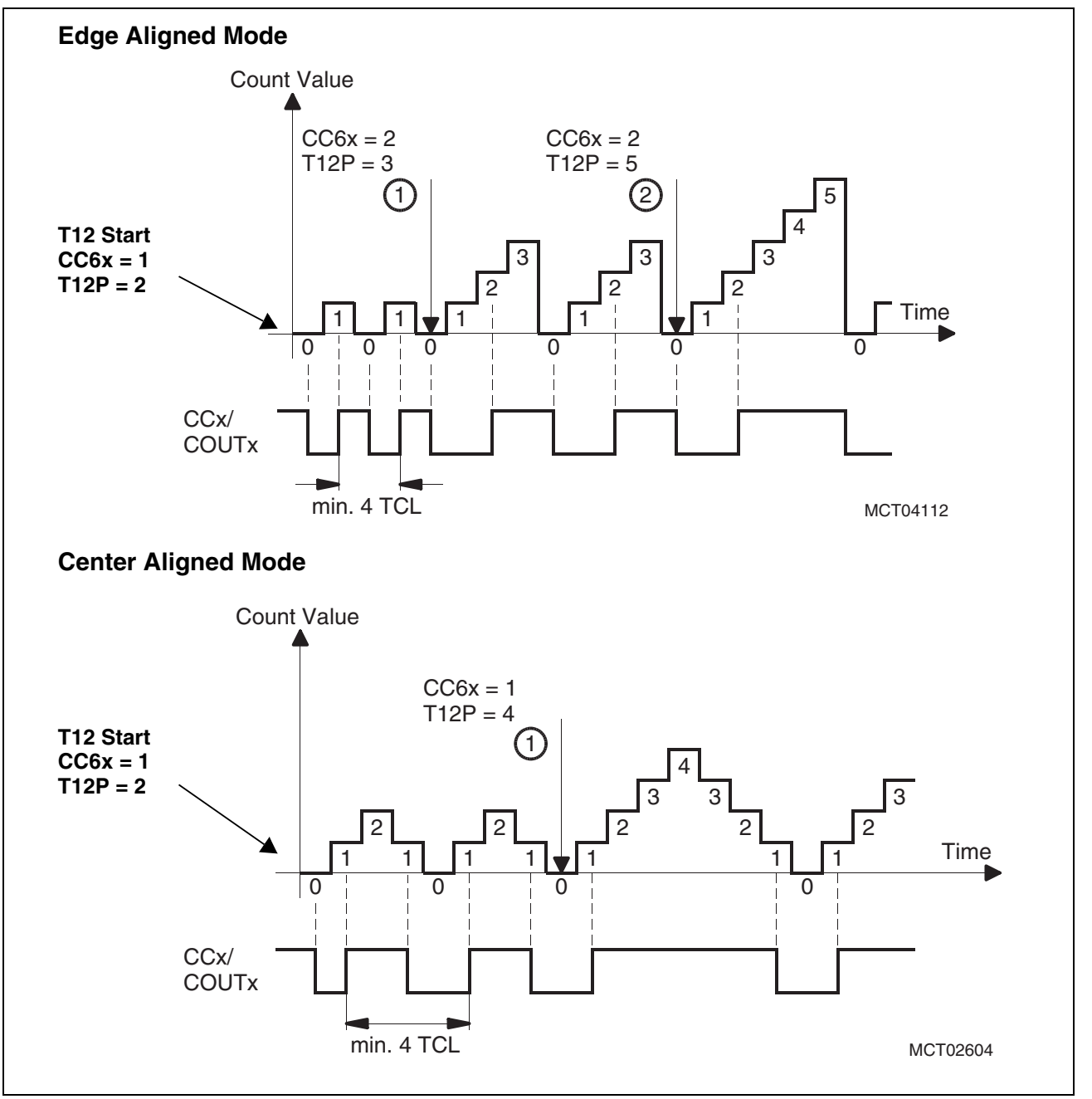


Figure 17-7 Operation in Center Aligned Mode

Table 17-1 Compare Timer Resolution and Period Range as Function of the Internal Clock @ f_{CPU} = 20 MHz

Internal Clock	Cmp. Timer Resolution	Output Signal Period Range (Txmin. - T12max. / T13max.)	
		Edge Aligned Mode	Center Aligned Mode
f_{CPU}	50 ns	100 ns - 3.28 ms / 51.2 µs	200 ns - 6.55 ms / 102.4 µs
f_{CPU} / 2	100 ns	200 ns - 6.55 ms / 102.4 µs	400 ns - 13.11 ms / 204.8 µs
f_{CPU} / 4	200 ns	400 ns - 13.11 ms / 204.8 µs	800 ns - 26.21 ms / 409.6 µs
f_{CPU} / 8	400 ns	800 ns - 26.21 ms / 409.6 µs	1.6 µs - 52.43 ms / 819.2 µs
f_{CPU} / 16	800 ns	1.6 µs - 52.43 ms / 819.2 µs	3.2 µs - 104.86 ms / 1.64 ms
f_{CPU} / 32	1.6 µs	3.2 µs - 104.86 ms / 1.64 ms	6.4 µs - 209.72 ms / 3.28 ms
f_{CPU} / 64	3.2 µs	6.4 µs - 209.72 ms / 3.28 ms	12.8 µs - 419.43 ms / 6.55 ms
f_{CPU} / 128	6.4 µs	12.8 µs - 419.43 ms / 6.55 ms	25.6 µs - 838.86 ms / 13.1 ms

Compare timer Tx period and duty cycle values can be calculated using the formulas below. The following abbreviations are used in these formulas:

pv = period value, stored in register TxP
ov = offset value, stored in register T12OF
cv = compare value, stored in register CC6x or CMP13

Note: For compare timer T13 only the output signal COUT63 in edge aligned mode is available.

Edge Aligned Mode:

$$\text{Period value} = pv + 1$$

$$\text{Duty cycle of CC6x outputs} = \left(1 - \frac{cv}{pv + 1}\right) \times 100\%$$

$$\text{Duty cycle of COUT6x outputs} = \left(1 - \frac{cv - ov}{pv + 1}\right) \times 100\%$$

Center Aligned Mode:

$$\text{Period value} = 2 \times pv$$

$$\text{Duty cycle of CC6x outputs} = \left(1 - \frac{cv}{pv}\right) \times 100\%$$

$$\text{Duty cycle of COUT6x outputs} = \left(1 - \frac{cv - ov}{pv}\right) \times 100\%$$

17.4 Burst Mode

In burst mode the output signal COUT63 of the 10-bit compare channel modulates the active phases of the output signals COUT6x of the three capture/compare channels. Burst mode is not possible on the CC6x outputs. The modulating signal typically has a higher frequency than the modulated output channels. **Figure 17-8** shows an example for a waveform generated in burst mode.

Burst mode is enabled separately for each capture/compare output by setting the respective bit CMSELx3 in register CC6MSEL.

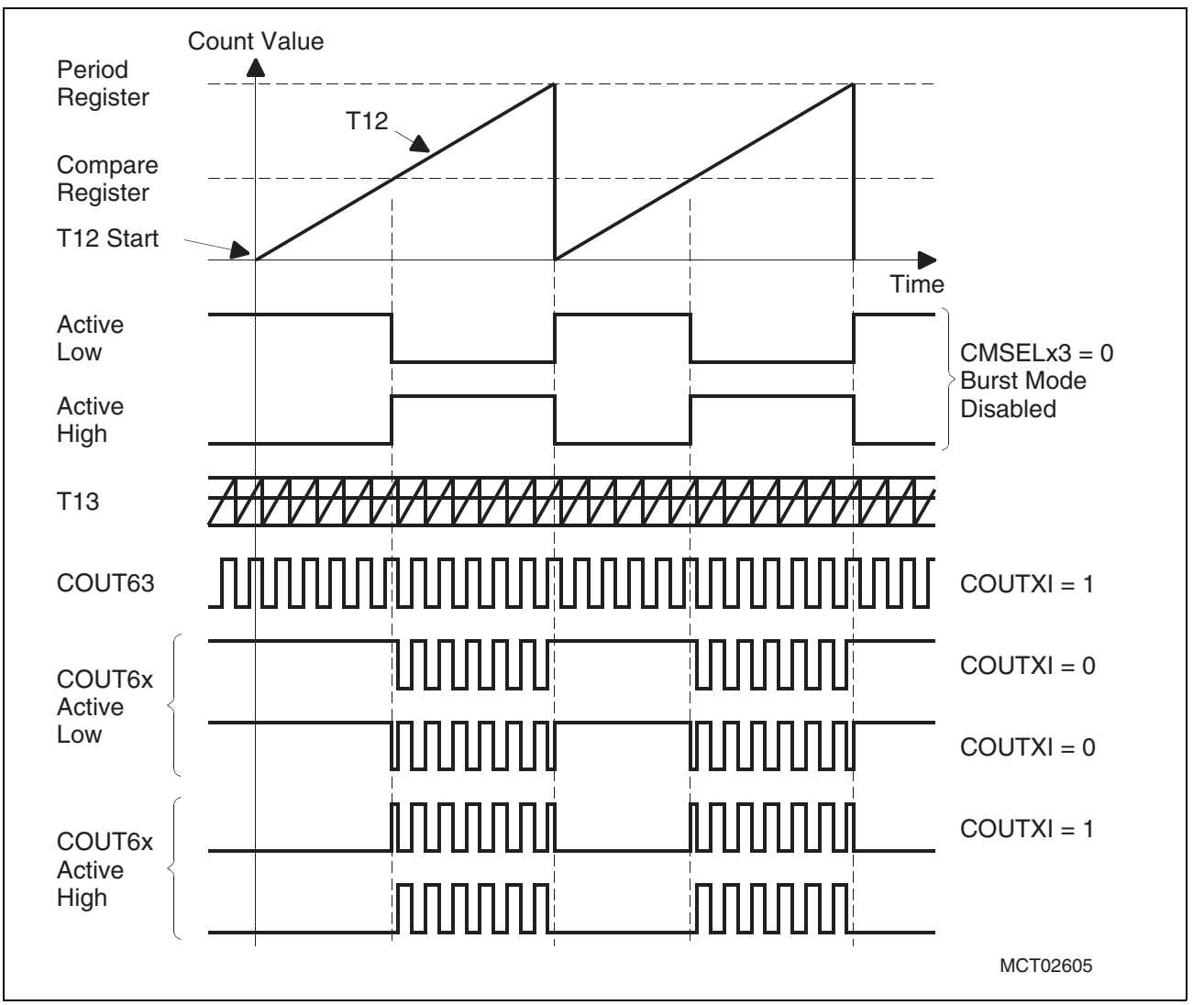


Figure 17-8 Operation in Burst Mode

17.5 Capture Mode

Each of the three capture/compare channels can be programmed individually for capture mode via bitfields CMSELx in register CC6MSEL. In capture mode the contents of timer T12 are copied to the channel's compare register CC6x upon a selectable transition (rising, falling, or both) at the associated pin CC6x. Capture mode can be enabled either in edge aligned mode or in center aligned mode. Interrupts may be generated selectively at each transition of the capture input signal.

Pins CC6x (used as inputs in capture mode) are sampled every CPU clock period.

When evaluating a series of capture events, it must be noted that every capture event overwrites the previous value in the respective register CC6x. The control software must be designed to retrieve the capture values before they are overwritten.

17.6 Combined Multi-Channel Modes

Note: Multi-channel modes are available in the full function module only.

When operating in a combined multi-channel mode, the output signals CC6x and COUT6x are controlled by the compare timers and additional conditions. Multi-channel modes are selected via register CC6MCON. In these modes a predefined signal pattern sequence is driven to the output lines.

Note: Compare timer T12 must be enabled (CT12R = '1') in order to enable proper operation of the multi-channel modes.

Multi-phase modes allow the effective generation of output signal patterns, for 4 … 6 phase unipolar drives, for example. The phase sequence can be controlled automatically by T12 overflows or by software.

Block Commutation mode is a special multi-channel mode which especially supports the control of brushless DC drives. In this mode the phase sequence is controlled by three input signals ($\overline{\text{CC6POSx}}$) generated by the drive (via hall sensors, for instance).

In all modes the output signals can be modulated during their active phases.

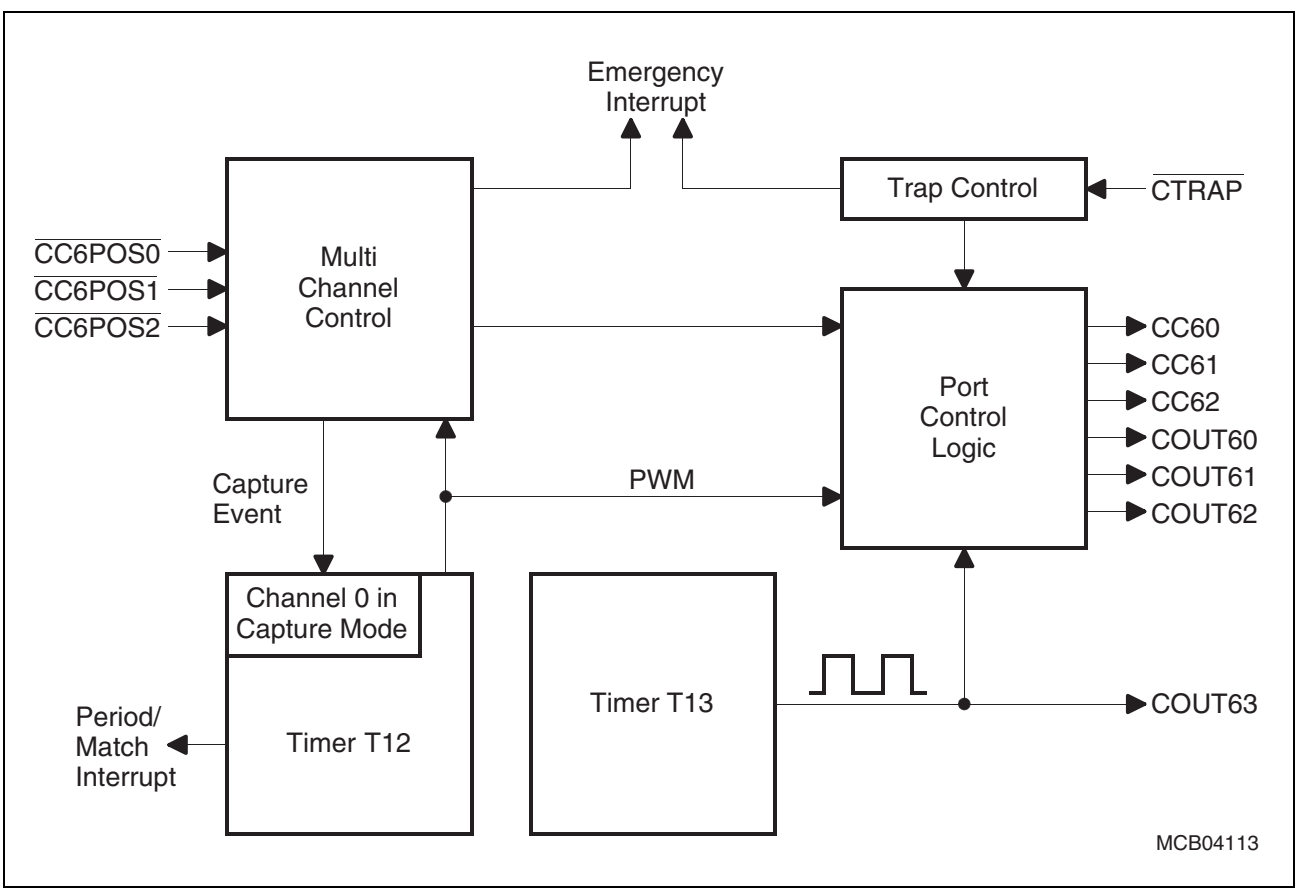
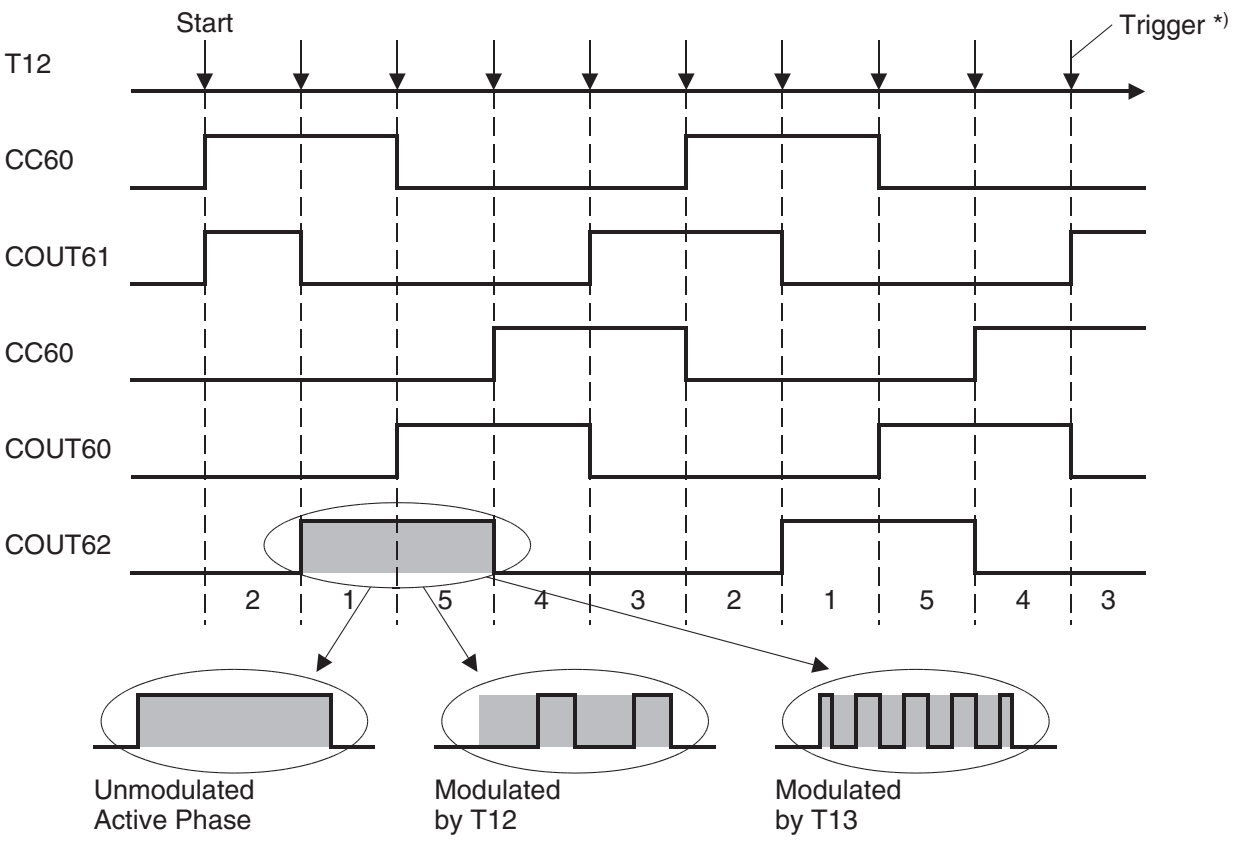


Figure 17-9 Multi-Channel Mode Control

17.6.1 Output Signals in Multi-Channel Mode

In multi-channel mode the output signals are controlled primarily by the selected phase sequence (see sequence tables below). Each output is active for two phases and remains passive for all other phases of a sequence.

The active phases of each output signal may additionally be modulated by T12 or T13. For unmodulated active phases timer T12 must operate with 100% duty cycle, that is its offset and compare registers must be cleared, and T13 modulation must be off (bits CMSELx3 must be cleared). T12 modulation is effective when T12's duty cycle is programmed below 100%, T13 modulation is enabled via bits CMSELx3 (see examples in **Figure 17-10**).



*) The trigger that switches to the next phase may be a T12 overflow or a '1' being written to bit NMCS via software. In block commutation mode the trigger is represented by a change in the input pattern on pins $\overline{\text{CC6POSx}}$.
The shown waveforms are active high.

MCA05114

Figure 17-10 Basic Five-Phase Multi-Channel Timing

Figure 17-10 shows the five-phase output waveforms as an example. For the other modes each passive phase is shortened or lengthened by one sequence phase, respectively.

The compare output signals are enabled according to the intended multi-phase mode.

Table 17-2 lists the required coding:

Table 17-2 Programming of Multi-Channel PWM Outputs

Multi-Channel PWM Mode	CMSEL2	CMSEL1	CMSEL0
Block commutation mode	011_B	011_B	011_B
4-phase multi-channel PWM	011_B	010_B	001_B
5-phase multi-channel PWM	011_B	010_B	011_B
6-phase multi-channel PWM	011_B	011_B	011_B

Note: Bit CMSELx3 (burst mode bit) defines whether or not the signal at the COUT6x pins is modulated by compare timer T13 (CMSELx3 = '1'). T13 modulation may be combined with T12 modulation.

Phase Sequence Tables

The following tables list the phase sequences for the various multi-phase modes. The sequence is defined via the follower state for each state and also the output levels for each state are listed.

The states of a phase sequence are switched in one of two ways:

- **Automatic switching** on a T12 overflow
- **Software controlled** by setting bit NMCS in register CC6MSEL.
  Bit ESMC = '1' enables software controlled state switching and disables automatic switching on T12 overflows.

Note: The actual logic levels for active and passive state are defined in register CC6MCON.
In four-phase, five-phase and six-phase multi-channel PWM mode all output signals can be modulated by timer T12 or timer T13 during their active phases.

Table 17-3 Four-Phase PWM Sequence Table

State	Output Level Definition (for actual state)						Follower State (for BCM = ...)			
	CC60	COUT61	CC62	COUT60	CC61	COUT62	01	10	00	11
0	passive	passive	passive	---	---	passive	2	1	0	5
1	ACTIVE	passive	passive	---	---	ACTIVE	4	2	0	5
2	ACTIVE	ACTIVE	passive	---	---	passive	1	3	0	5
3	passive	ACTIVE	ACTIVE	---	---	passive	2	4	0	5
4	passive	passive	ACTIVE	---	---	ACTIVE	3	1	0	5
5	passive	ACTIVE	passive	---	---	ACTIVE	2	1	0	5

Table 17-4 Five-Phase PWM Sequence Table

State	Output Level Definition (for actual state)						Follower State (for BCM = ...)			
	CC60	COUT61	CC62	COUT60	CC61	COUT62	01	10	00	11
0	passive	passive	passive	passive	---	passive	2	1	0	6
1	ACTIVE	passive	passive	passive	---	ACTIVE	5	2	0	6
2	ACTIVE	ACTIVE	passive	passive	---	passive	1	3	0	6
3	passive	ACTIVE	ACTIVE	passive	---	passive	2	4	0	6
4	passive	passive	ACTIVE	ACTIVE	---	passive	3	5	0	6
5	passive	passive	passive	ACTIVE	---	ACTIVE	4	1	0	6
6	passive	ACTIVE	passive	ACTIVE	---	ACTIVE	2	1	0	6

Table 17-5 Six-Phase PWM Sequence Table

State	Output Level Definition (for actual state)						Follower State (for BCM = ...)			
	CC60	COUT61	CC62	COUT60	CC61	COUT62	01	10	00	11
0	passive	passive	passive	passive	passive	passive	1	6	0	7
1	ACTIVE	ACTIVE	passive	passive	passive	passive	6	2	0	7
2	passive	ACTIVE	ACTIVE	passive	passive	passive	1	3	0	7
3	passive	passive	ACTIVE	ACTIVE	passive	passive	2	4	0	7
4	passive	passive	passive	ACTIVE	ACTIVE	passive	3	5	0	7
5	passive	passive	passive	passive	ACTIVE	ACTIVE	4	6	0	7
6	ACTIVE	passive	passive	passive	passive	ACTIVE	5	1	0	7
7	passive	ACTIVE	passive	ACTIVE	passive	ACTIVE	2	1	0	7

Note: To change the rotation direction idle mode must be entered first.

17.6.2 Block Commutation Mode

Block commutation mode is a special variation of the multi-channel modes in which the phase sequence is not controlled internally but rather by the three input signals $\overline{CC6POS2}$…0. The state of the six output signals is derived from the pattern present on the input signals. **Table 17-6** summarizes the possible combinations.

Table 17-6 Block Commutation Sequence Table

Block Commutation Mode (BCM)	Control Inputs $\overline{CC6POS...}$			Output Level Definition (for actual state)					
	0	**1**	**2**	**CC60**	**CC61**	**CC62**	**COUT60**	**COUT61**	**COUT62**
Rotate Left	1	0	1	passive	passive	ACTIVE	passive	ACTIVE	passive
	1	0	0	passive	passive	ACTIVE	ACTIVE	passive	passive
	1	1	0	passive	ACTIVE	passive	ACTIVE	passive	passive
	0	1	0	passive	ACTIVE	passive	passive	passive	ACTIVE
	0	1	1	ACTIVE	passive	passive	passive	passive	ACTIVE
	0	0	1	ACTIVE	passive	passive	passive	ACTIVE	passive
Rotate Right	1	1	0	ACTIVE	passive	passive	passive	ACTIVE	passive
	1	0	0	ACTIVE	passive	passive	passive	passive	ACTIVE
	1	0	1	passive	ACTIVE	passive	passive	passive	ACTIVE
	0	0	1	passive	ACTIVE	passive	ACTIVE	passive	passive
	0	1	1	passive	passive	ACTIVE	ACTIVE	passive	passive
	0	1	0	passive	passive	ACTIVE	passive	ACTIVE	passive
Rotate Left[1)] Rotate Right	0	0	0	passive	passive	passive	passive	passive	passive
	1	1	1	passive	passive	passive	passive	passive	passive
Slow Down	X	X	X	passive	passive	passive	ACTIVE	ACTIVE	ACTIVE
Idle[2)]	X	X	X	passive	passive	passive	passive	passive	passive

1) If one of these two input signal combinations is detected in rotate left or rotate right mode, bit BCERR is set. If enabled an emergency interrupt is generated. When these (error) states are encountered, the idle state is entered immediately.

2) Idle state is entered when a “wrong follower” is detected (if bit BCEM = ‘1’), or in case of an illegal input pattern (see note 1). When idle state is entered the BCERR flag is always set.
Idle state can only be left when the BCERR flag is cleared by software.

In block commutation mode CAPCOM channel 0 is automatically configured for capture mode. Any signal transition at inputs $\overline{\text{CC6POS2…0}}$ generates a capture pulse for CAPCOM channel 0 and sets the interrupt request flag CC0R. A rising edge at output pin CC60 does not generate an interrupt request in block commutation mode.

The values provide a measurement of the rotation speed of the connected drive. When evaluating the values captured from the free-running timer T12, the timer must not be stopped, as this would disturb the operation of block commutation mode.

Note: Modulation of the active phase via T12 is not supported. PWM via T13 is possible on COUT6x.
The block commutation input signals are available for the full function module only.

17.7 Trap Function

The trap function switches selectable output lines of the CAPCOM6 to predefined levels simultaneously. The trap function is triggered by an external signal connected to input $\overline{\text{CTRAP}}$. This feature provides a very efficient means of protecting external circuitry, such as power bridges for inverters or motors, which are connected to the CAPCOM6's output lines. Register TRCON enables and controls the trap function, register CC6MCON or P1L provides the output levels during trap state.

Figure 17-11 shows examples for a trap state in edge aligned mode and in center aligned mode.

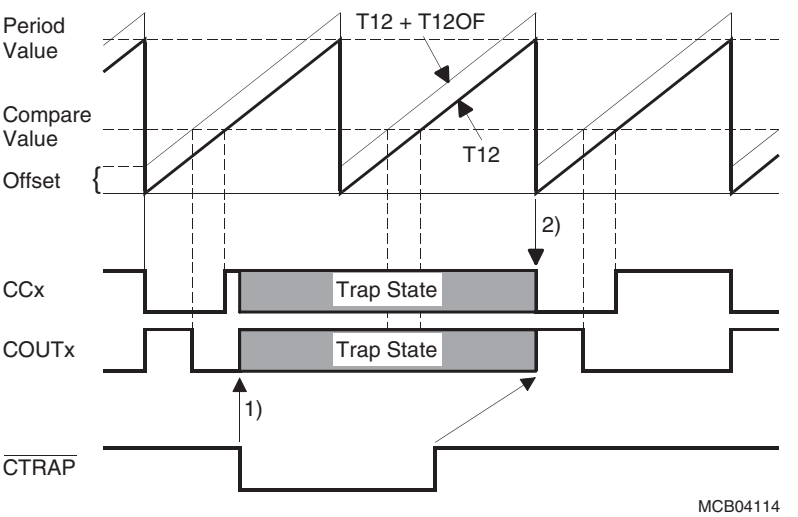
Notes

Reference Point 1):
Input $\overline{\text{CTRAP}}$ is activated, the outputs **immediately** switch to their **trap** level.

Reference Point 2):
Input $\overline{\text{CTRAP}}$ is inactive, the outputs switch to their programmed level **synchronized** to a new timer period.

Secure Trap State
During trap state the outputs CC6x and COUT6x which are selected for T13 modulation are not modulated with T13's output signal but rather with its initial value.

Trap Function In Edge Aligned Mode



Trap Function In Center Aligned Mode

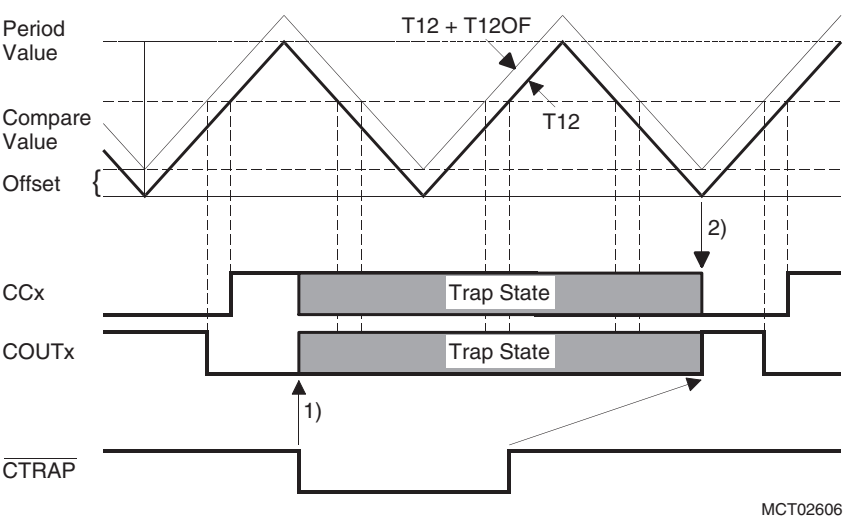


Figure 17-11 Trap Function

Entering Trap State

Bit TRPEN generally enables the trigger function of input $\overline{\text{CTRAP}}$. When enabled, a falling edge on input $\overline{\text{CTRAP}}$ activates the trap state immediately without any CPU activity (see Reference Point 1 in **Figure 17-11**). This event sets the trap flag TRF in register TRCON (to signal this event to the software) and generates an interrupt request. An interrupt is generated if the corresponding interrupt node is enabled.
If bit CT12RES in register CTCON is set timer T12 is cleared upon a trap event, otherwise it continues counting. No more transitions on the output signals will be generated, however.

Leaving Trap State

After the trap trigger is removed (input $\overline{\text{CTRAP}}$ has been sampled inactive), trap state is not left immediately, but in a synchronized way (see Reference Point 2 in **Figure 17-11**), when timer T12 reaches the value 0000_H. This “delay” automatically resumes the generation of the programmed output signals after a trap event synchronized to the next timer period. The generation of distorted (truncated) pulses is avoided.

*Note: In block commutation mode trap state is exited when timer **T13** (not T12) reaches 000_H.*

Controlling Trap State

The general trap state control signal provides the timing for the trap logic and is valid for all output signals (see “Trap Trigger” in **Figure 17-12**).

The trap enable logic determines the effect of the trap state on the individual CAPCOM6 outputs (see “Trap Enable” in **Figure 17-12**).

In the default case all outputs are switched to the level of the associated port output latch P1L.x. In this case the trap state level for each output can be predefined independent of the normal operation of the respective signal (including its initial level).

For each timer separately the standard trap function for its associated outputs can be disabled by setting bit TT12DIS or TT13DIS, respectively.

For each individual T12 channel (CC60 … COUT62) the respective initial level can be driven during trap state by setting the associated bit(s) TRENx.
For the T13 output (COUT63) no TREN bit is available, so setting bit TT13DIS disables the trap function completely for this output.

Note: When controlling inverters or electric motors the standard trap mode (using P1L) ensures safe operation (all transistors off), so in these cases bits TRENx and TTxDIS in register TRCON should be written with ‘0’ only.

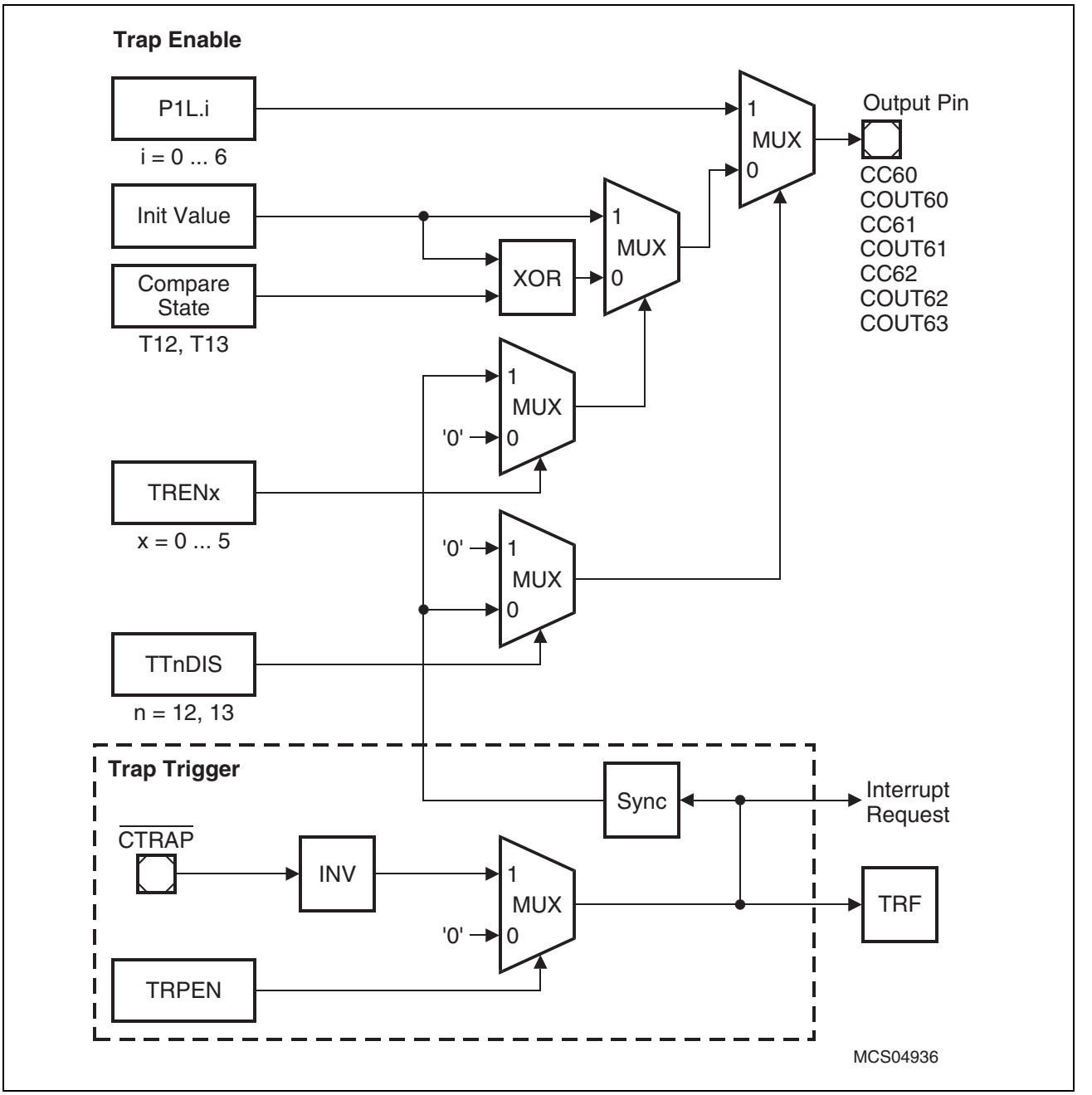


Figure 17-12 Trap Control Overview

17.8 Register Descriptions

The CAPCOM6 register set provides a number of control, data, and status bits to control the operation of the two compare timers, the generation of the output signals (up to 7) and the combination of submodules for multi-channel operation.

Note: The register bits which are available in the full function module only (not in the reduced version) are marked. This provides an immediate overview of the available registers and control/status bits in a specific derivative.

Table 17-7 summarizes the available registers. The **control registers** are described in detail in the following sections of this chapter. Data registers (such as period or compare registers) are excluded from the detailed description. Please note that the timer registers (T12, T13) are not directly accessible.

Table 17-7 CAPCOM6 Register Summary

Name		Description	Address	Read
T12P	**E**	Timer T12 period register	$F030_H$ / 18_H	Sh.L.
T12OF	**E**	Timer T12 offset register	$F034_H$ / $1A_H$	Sh.L.
T13P	**E**	Timer T13 period register	$F032_H$ / 19_H	Sh.L.
CMP13		Compare register for compare channel	$FE36_H$ / $1B_H$	Sh.L.
CC60		Compare register for capture/compare channel 0	$FE30_H$ / 18_H	Reg.
CC61		Compare register for capture/compare channel 1	$FE32_H$ / 19_H	Reg.
CC62		Compare register for capture/compare channel 2	$FE34_H$ / $1A_H$	Reg.
CTCON		Compare timer control register	$FF30_H$ / 98_H	Reg.
TRCON		Trap enable register	$FF34_H$ / $9A_H$	Reg.
CC6MCON		CAPCOM6 mode control register	$FF32_H$ / 99_H	Reg.
CC6MSEL	**E**	CAPCOM6 mode select register	$F036_H$ / $1B_H$	Reg.
CC6MIC		CAPCOM6 interrupt control register	$FF36_H$ / $9B_H$	Reg.

Note: When reading these registers, either the register itself or its shadow latch is accessed (see description below). This is indicated in column “Read” by “Reg.” = Register and “Sh.L.” = Shadow latch.

Additionally there are four interrupt node control registers associated with the CAPCOM6 unit; however, they are not part of the module (see **Page 17-33**).

Shadow Latches for Synchronous Update

The timer period, offset, and compare values are written to shadow latches rather than to the actual registers. Also the initial value bits CCxI/COUTxI in register CC6MCON are equipped with shadow latches. Thus the values for a new output signal can be programmed without disturbing the currently generated signal(s). The transfer from the latches to the registers is enabled by setting the respective shadow latch transfer enable bit STEx in register CTCON.

If the transfer is enabled the shadow latches are copied to the respective registers the next time the associated timer reaches the value zero (either being cleared in edge aligned mode or counting down from 1 in center aligned mode).

When timer T12 is operating in center aligned mode it will also copy the latches (if enabled) if it reaches the currently programmed period value (counting up).

After the transfer the respective bit STEx is automatically cleared.

Note: While T12/T13 is running, the shadow latch transfer is controlled by bit STE12/13. While T12/T13 is stopped, the shadow latch transfer is done automatically if bit CTRES12/13 is set; otherwise those latch values are not transferred.

Note: If a new compare value is written to the shadow latches while T12 is counting up, the new value must be smaller than the current period value. Otherwise no more matches will be detected and the output signals will no longer change.
If a compare value is written, while T12 is counting down, any value may be used.

CTCON
Compare Timer Control Reg. **SFR ($FF30_H/98_H$)** **Reset Value: 1010_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CT13P	ECT13O	STE13	CT13RES	CT13R	CT13CLK			CTM	ETRP	STE12	CT12RES	CT12R	CT12CLK		
rwh	rw	rwh	rw	rw	rw			rw	rw	rwh	rw	rw	rw		

Bit	Function
CTnCLK	**Compare Timer Tn Input Clock Select** Selects the input clock for timer T12 or T13 derived from the CPU clock: $f_{Tx} = f_{CPU} / 2^{<CTnCLK>}$. 000: $f_{Tx} = f_{CPU}$ … 111: $f_{Tx} = f_{CPU} / 128$
CTnR	**Compare Timer Tn Run Bit** CTnR starts and stops timer Tn (T12 or T13). Together with bit CTnRES it controls Tn's operation. 0: Timer Tn stops counting. If bit CTnRES = '1' timer Tn is cleared and the compare outputs are set to their defined idle state. 1: Timer Tn starts counting from its current value.
CTnRES	**Compare Timer Tn Reset Control** 0: No effect on timer Tn when it is stopped. 1: Timer Tn is cleared when it is stopped and the compare outputs are set to their defined idle state. *Note: For capture mode (T12 only): Clearing CT12R after a capture event while CT12RES = '1' will destroy the value stored in the capture register CC6x (all shadow registers are transparent). Keep CT12RES = '0' in capture mode.*
STE12	**Timer T12 Shadow Latch Transfer Enable** 0: Transfer from the shadow latches to the initial value bits, and the period, compare, and offset registers (T12P, CC6x, T12OF) of timer T12 is disabled. 1: Timer T12's initial value bits, and the period, compare, and offset registers are loaded from their shadow latches when T12 reaches 0000_H (cleared in edge aligned mode, counting down in center aligned mode). In center aligned mode the registers are also loaded when T12 reaches the period value. *Note: STE12 is cleared by hardware after the shadow latch transfer.*

Bit	Function
ETRP	**Emergency Trap Interrupt Enable** 0: The emergency interrupt for the CAPCOM6 trap signal is disabled. 1: The emergency interrupt for the CAPCOM6 trap signal is enabled.
CTM	**T12 Operating Mode** 0: Edge Aligned Mode: count up. 1: Center Aligned Mode: count up/down.
STE13	**Timer T13 Shadow Latch Transfer Enable** 0: Transfer from the shadow latches to the period and compare registers (CC62, CMPx) of timer T13 is disabled. 1: The period and compare registers of timer T13 are loaded from their shadow latches when T13 reaches the respective period value. *Note: STE13 is cleared by hardware after the shadow latch transfer.*
ECT13O	Enable compare timer T13 output 0: When ECT13O is cleared and timer T13 is running, signal COUT63 outputs the corresponding port latch value. 1: When ECT13O is set and timer T13 is running, timer T13 output COUT63 is enabled and outputs the PWM signal of the 10-bit compare channel.
CT13P	**Timer T13 Period Flag** The period flag CT13P is set whenever the contents of timer T13 match the contents of the timer T13 period register. This also generates an interrupt request. Bit CT13P must be cleared by software.

TRCON
Trap Enable Register **SFR ($FF34_H/9A_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TRPEN	TRF	TREN5	TREN4	TREN3	TREN2	TREN1	TREN0	-	-	-	-	-	-	TT13DIS	TT12DIS
rw	rwh	rw	rw	rw	rw	rw	rw	-	-	-	-	-	-	rw	rw

Bit	Function
TT12DIS	**Timer T12 Trap Disable Bit** 0: Standard trap levels for timer T12 controlled outputs (P1L) 1: Trap level is initial value for those timer T12 controlled outputs enabled by bits TRENx
TT13DIS	**Timer T13 Trap Disable Bit** 0: Standard trap level for timer T13 controlled output (P1L) 1: Trap function for timer T13 controlled output is disabled
TRENx	**Trap Enable for Output Pins** 0: Trap function for the respective output is disabled 1: Trap level is initial value for the respective output, if the standard trap level (from P1L) is disabled by bot TT12DIS = ‘1’
TRF	**Trap Flag** TRF is set by hardware if the trap function is enabled (TRPEN = 1) and $\overline{\text{CTRAP}}$ becomes active (low). If enabled, an interrupt is generated when TRF is set. TRF must be cleared by software.
TRPEN	**External $\overline{\text{CTRAP}}$ Trap Function Enable Bit** 0: External trap input $\overline{\text{CTRAP}}$ is disabled (default after reset). 1: External trap input $\overline{\text{CTRAP}}$ is enabled.

Note: For applications driving inverters or electric motors the standard trap mode using P1L provides maximum safety. It is therefore recommended to keep bits TRENx and TT12DIS cleared (only write ‘0’ to those bit locations).

CC6MCON
CAPCOM6 Mode Ctrl. Reg. **SFR ($FF32_H/99_H$)** **Reset Value: $00FF_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BC POL BC EM	MPWM		EB CE	BC ERR	BC EN	BCM		COUT 3I	COUT XI	COUT 2I	CC2I	COUT 1I	CC1I	COUT 0I	CC0I
rw	rw		rw	rwh	rw	rw		rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
CCnI	**Compare Output CC6n Initial Value** (n = 0 … 2) The compare output CC6n drives the value of CCnI when the compare timer T12 is not running. CCnI represents the passive output level for an enabled compare channel. *Note: The initial values are valid only for capture/compare outputs which are enabled for compare mode operation (compare output).*
COUTnI	**Compare Output COUT6n Initial Value** (n = 0 … 2) The compare output COUT6n drives the value of COUTnI when the compare timer T12 is not running. COUTnI represents the passive output level for an enabled compare channel. *Note: The initial values are valid only for capture/compare outputs which are enabled for compare mode operation (compare output).*
COUTXI	**COUT6n Inversion Control** 0: T13's output signal is directly connected to compare outputs COUT6n in burst or multi-channel mode (n = 0 … 2). 1: T13's output signal is inverted and then connected to compare outputs COUT6n in burst or multi-channel mode (n = 0 … 2).
COUT3I	**Compare Output COUT63 Initial Value** This bit defines the initial logic state of the output COUT63 before timer T13 is started the first time. Further, COUT3I defines the logic state of COUT63 when bit ECT13O is reset (COUT63 disabled).
BCM	**Multi-channel PWM Mode Output Pattern Selection** This bitfield selects the output signal pattern in all multi-channel PWM modes (also refer to bitfield MPWM). 00: Idle mode. 01: Rotate right mode. 10: Rotate left mode. 11: Slow down mode.

Bit	Function
BCEN	**Block Commutation Enable** 0: The multi-channel PWM modes of the 16-bit capture/compare channels (selected by bitfield MPWM) are disabled. 1: The multi-channel PWM modes are enabled. *Note: Before bit BCEN is set, all required PWM compare outputs should be programmed to operate as compare outputs by writing to register CC6MSEL.*
BCERR	**Block Commutation Mode Error Flag** 0: No error condition. 1: An error condition in rotate right or rotate left mode has occurred: - After a transition at $\overline{\text{CC6POSx}}$ all $\overline{\text{CC6POSx}}$ inputs are at high or low level. - A “wrong follower” condition has occurred (see description of bit BCEM). If the block commutation interrupt is enabled (EBCE = ‘1’) a CAPCOM6 emergency interrupt will also be generated. BCERR must be cleared by software.
EBCE	**Enable Block Commutation Mode Error Interrupt** 0: Block commutation mode error does not generate an interrupt. 1: The emergency interrupt is activated for a block commutation mode error. Refer to the description of bits BCERR and BCEM.
MPWM	**Multi-channel PWM Mode Selection** This bitfield selects the output signal pattern in all multi-channel PWM modes (also refer to bitfield BCM). 00: Three-phase block commutation mode. 01: Four-phase multi-channel PWM mode. 10: Five-phase multi-channel PWM mode. 11: Six-phase multi-channel PWM mode.

Bit	Function
BCPOL	**Machine polarity** (Valid only in multi-channel PWM mode) 0: Only the COUT6n outputs are switched to the timer T13 output signal during the active phase in multi-channel PWM mode. CMSELn3 must be set for that functionality. 1: All enabled compare outputs COUT6n and CC6n are switched to the timer T13 output signal during their active phase in multi-channel PWM mode.
BCEM	**Error mode select bit** (Valid only in block commutation mode) 0: A “wrong follower” condition is not notified as an error. 1: A “wrong follower” condition in rotate right or rotate left mode sets flag BCERR if EBCE is set.

*Note: When a multi-channel PWM mode is initiated the first time after reset, CC6MCON must be written twice: The first write operation is with bit BCEN cleared and all other bits set/cleared as required (BCM **must be** ‘00’ for idle mode), the second write operation has the same CC6MCON bit pattern as the first write operation **but with BCEN set**. After this second CC6MCON write operation, timer T12 can be started (setting CT12R in CTCON) and thereafter BCM can be put into a mode other than the idle mode.*

CC6MSEL
CAPCOM6 Mode Select Reg. ESFR ($F036_H/1B_H$) Reset Value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ESMC	NMCS	-	-	CMSEL23	CMSEL2			CMSEL13	CMSEL1			CMSEL03	CMSEL0		
rw	rw	-	-	rw	rw			rw	rw			rw	rw		

Bit	Function
CMSELn	**Capture/Compare Mode Selection** These bitfields select/enable the operating mode and the output/input pin configuration of the 16-bit capture/compare channels. Each channel can be programmed individually either for compare or capture operation. 000: Compare outputs disabled, CC6n/COUT6n can be used for IO. 001: Compare output on pin CC6n, COUT6n can be used for IO. 010: Compare output on pin COUT6n, CC6n can be used for IO. 011: Compare output on pins COUT6n and CC6n. 100: Capture mode, not triggered by CC6n. COUT6n is IO. 101: Capture mode, trigg'd by a rising edge on CC6n. COUT6n is IO. 110: Capture mode, trigg'd by a falling edge on CC6n. COUT6n is IO. 111: Capture mode, trigg'd by any transition on CC6n. COUT6n is IO.
CMSELn3	**COUT6n Control by Timer T13 in Compare Mode** This bit determines if the output COUT6n is modulated during its active phase (defined via register CC6MCON) by the output signal of the 10-bit compare channel, typically a higher frequency signal. 0: COUT6n drives its active level. 1: COUT6n is modulated by the output signal of the 10-bit compare channel.
NMCS	**Next Multi-Channel PWM State** (Valid when ESMC = '1') 0: Idle. 1: Select the next follower state in the 4/5/6-phase multi-channel PWM modes. NMCS is reset by hardware in the next clock cycle after it has been set.
ESMC	**Enable Software Controlled Multi-Channel PWM Modes** Defines the follower state selection in the 4/5/6-phase multi-channel PWM modes. 0: Follower state selection controlled by compare timer T12. 1: Follower state selection controlled by bit NMCS (software control).

CC6MIC
CAPCOM6 Interrupt Ctrl. Reg. SFR ($FF36_H/9B_H$) Reset Value: 0000_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CT12FP	CT12FC	CC2F	CC2R	CC1F	CC1R	CC0F	CC0R	ECTP	ECTC	CC2FEN	CC2REN	CC1FEN	CC1REN	CC0FEN	CC0REN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
CCnREN	**Capture/Compare Rising Edge Interrupt Enable** 0: Rising edge interrupt disabled. 1: An interrupt from request flag CCnR is enabled.
CCnFEN	**Capture/Compare Falling Edge Interrupt Enable** 0: Falling edge interrupt disabled. 1: An interrupt from request flag CCnF is enabled.
ECTC	**Enable Timer T12 Count Direction Change Interrupt** 0: Count direction change interrupt disabled. 1: An interrupt from request flag CT12FC is enabled. *Note: No effect in edge aligned mode.*
ECTP	**Enable Timer T12 Period Interrupt** 0: Period interrupt disabled. 1: An interrupt from request flag CT12FP is enabled.
CCnR	**Capture/Compare Rising Edge Interrupt Flag** 0: Idle. 1: The interrupt request flag is set as follows: - **Capture mode**: upon a rising edge at the corresponding CC6n input - **Compare mode**: when T12 matches compare register CC6n while counting up (in both operating modes of timer T12).
CCnF	**Capture/Compare Falling Edge Interrupt Flag** 0: Idle. 1: The interrupt request flag is set as follows: - **Capture mode**: upon a falling edge at the corresponding CC6n input - **Compare mode**: when T12 matches compare register CC6n while counting down (in center aligned mode, timer T12 only).

Bit	Function
CT12FC	**Timer T12 Count Direction Change Flag** 0: Idle. 1: An interrupt request is generated when T12 matches 0000_H (counting down in center aligned mode) and changes to counting up. There is no effect in edge aligned mode.
CT12FP	**Timer T12 Period Flag** 0: Idle. 1: An interrupt request is generated when T12 matches the period value.

Note: All CAPCOM6 interrupt request bits in register CC6MIC must be cleared by software.

17.9 The CAPCOM6 Interrupt Structure

Figure 17-13 summarizes the CAPCOM6's interrupt sources and the related status and control flags, and shows the association with the four CAPCOM6 interrupt nodes.

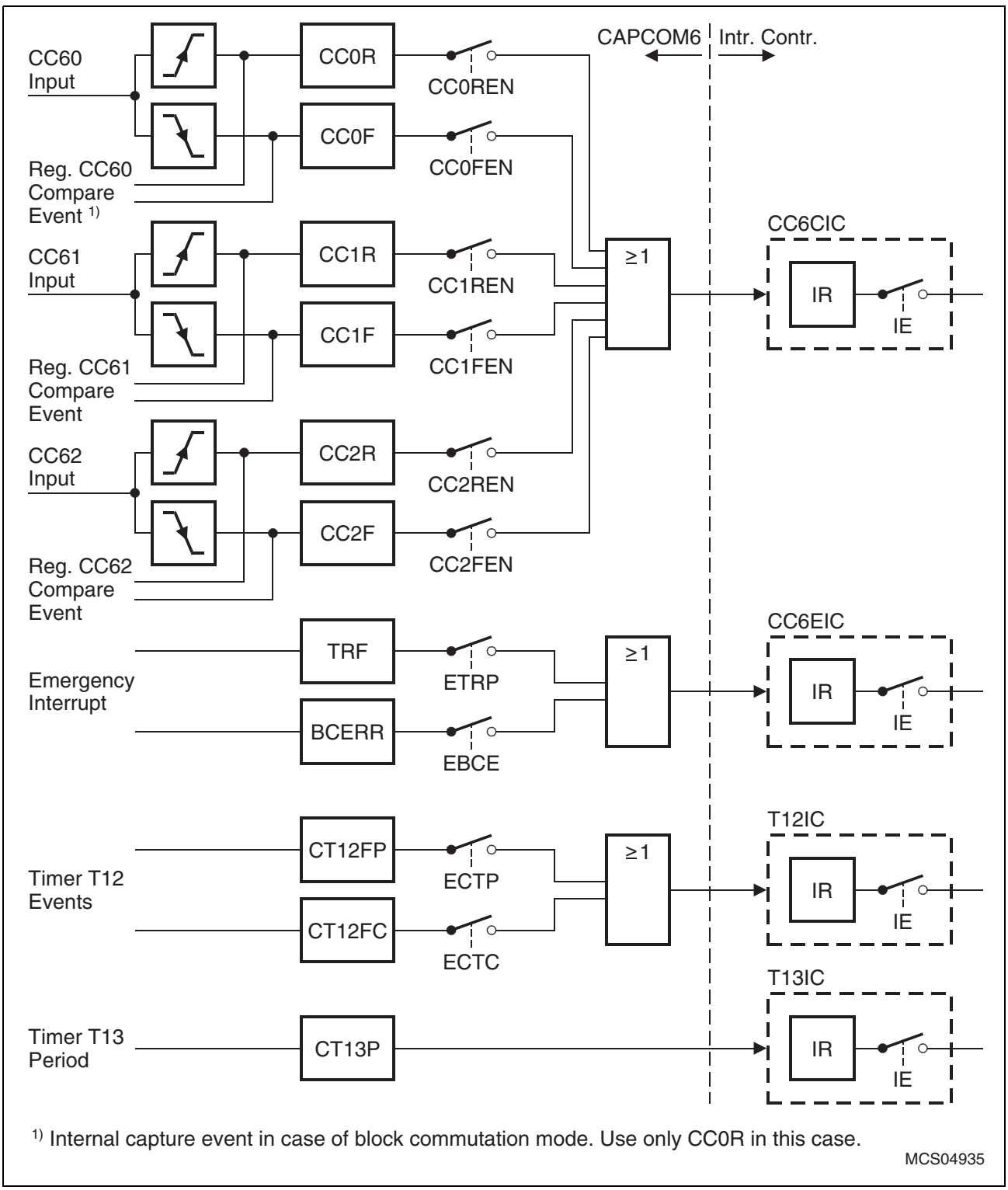


Figure 17-13 CAPCOM6 Interrupt Structure

Interrupt Node Control Registers

T12IC **ESFR ($F190_H/C8_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								T12IR	T12IE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

T13IC **ESFR ($F198_H/CC_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								T13IR	T13IE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

CC6EIC **ESFR ($F188_H/C4_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								CC6EIR	CC6EIE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

CC6CIC **ESFR ($F17E_H/BF_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								CC6IR	CC6EI	ILVL				GLVL	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

18 Analog/Digital Converter

The C164CI provides an Analog/Digital Converter (ADC) with 10-bit resolution and a sample & hold circuit on-chip. A multiplexer selects up to 8 analog input channels (alternate functions of Port 5) either via software (fixed channel modes) or automatically (auto scan modes).

To fulfill most requirements of embedded control applications, the ADC supports the following conversion modes:

- **Fixed Channel Single Conversion**
  produces just one result from the selected channel
- **Fixed Channel Continuous Conversion**
  repeatedly converts the selected channel
- **Auto Scan Single Conversion**
  produces one result from each of a selected group of channels
- **Auto Scan Continuous Conversion**
  repeatedly converts the selected group of channels
- **Wait for ADDAT Read Mode**
  start a conversion automatically when the previous result was read
- **Channel Injection Mode**
  insert the conversion of a specific channel into a group conversion (auto scan)

A set of SFRs and port pins provide access to control functions and results of the ADC.

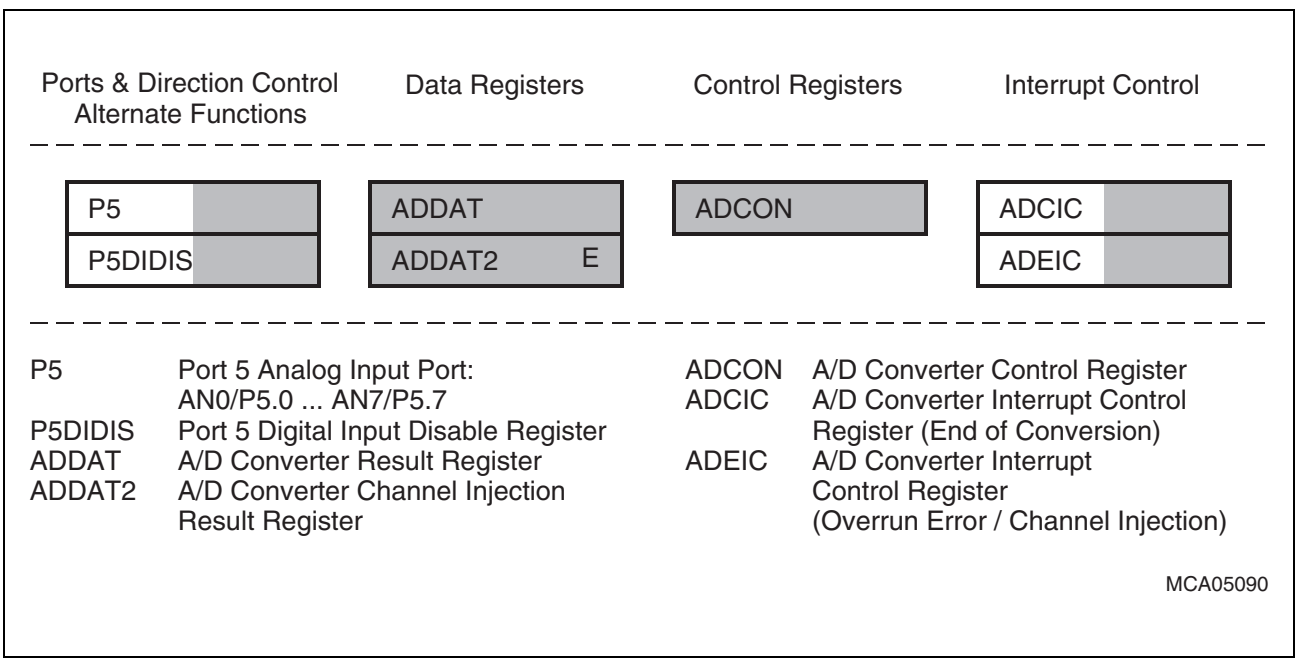


Figure 18-1 SFRs and Port Pins Associated with the A/D Converter

The external analog reference voltages V_{AREF} and V_{AGND} are fixed. The separate supply for the ADC reduces the interference with other digital signals.

The sample time and the conversion time are programmable, so the ADC can be adjusted to the internal resistances of the analog sources and/or the analog reference voltage supply.

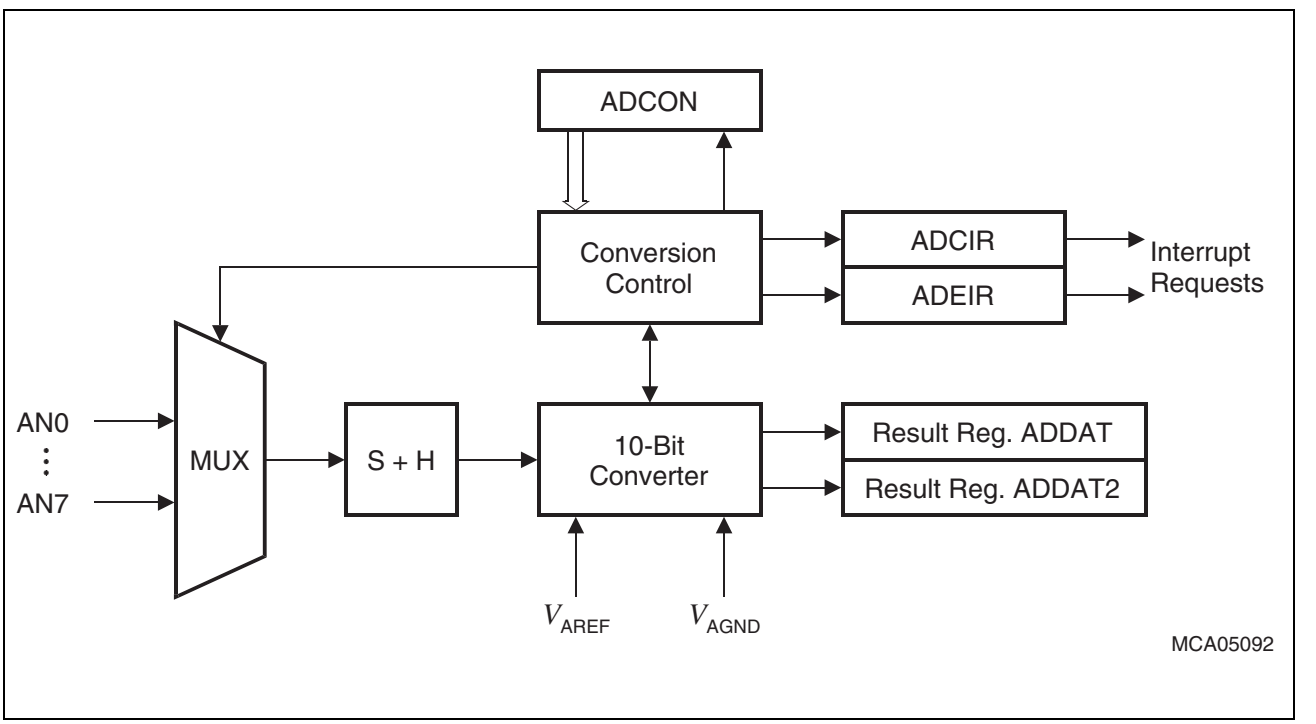


Figure 18-2 Analog/Digital Converter Block Diagram

18.1 Mode Selection and Operation

The analog input channels AN7 … AN0 are alternate functions of Port 5 which is an input-only port. The Port 5 lines may be used as either analog or digital inputs. For pins to be used as analog inputs, it is recommended to disable the digital input stage via register P5DIDIS. This avoids undesired cross currents and switching noise when the (analog) input signal level is between V_{IL} and V_{IH}.

The functions of the A/D converter are controlled by the bit-addressable A/D Converter Control Register ADCON. Its bitfields specify the analog channel to be acted upon, the conversion mode, and also reflect the status of the converter.

ADCON
ADC Control Register **SFR ($FFA0_H/D0_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADCTC		ADSTC		AD CRQ	AD CIN	AD WR	AD BSY	AD ST	-	ADM		ADCH			
rw		rw		rwh	rw	rw	rwh	rwh	-	rw		rw			

Bit	Function
ADCH	**ADC Analog Channel Input Selection** Selects the (first) ADC channel to be converted.
ADM	**ADC Mode Selection** 00: Fixed Channel Single Conversion 01: Fixed Channel Continuous Conversion 10: Auto Scan Single Conversion 11: Auto Scan Continuous Conversion
ADST	**ADC Start Bit** 0: Stop a running conversion 1: Start conversion(s)
ADBSY	**ADC Busy Flag** 0: ADC is idle 1: A conversion is active
ADWR	**ADC Wait for Read Control**
ADCIN	**ADC Channel Injection Enable**
ADCRQ	**ADC Channel Injection Request Flag**

Bit	Function
ADSTC	**ADC Sample Time Control** (Defines the ADC sample time in a certain range) 00: $t_{BC} \times 8$ 01: $t_{BC} \times 16$ 10: $t_{BC} \times 32$ 11: $t_{BC} \times 64$
ADCTC	**ADC Conversion Time Control** (Defines the ADC basic conversion clock f_{BC}) 00: $f_{BC} = f_{CPU} / 4$ 01: $f_{BC} = f_{CPU} / 2$ 10: $f_{BC} = f_{CPU} / 16$ 11: $f_{BC} = f_{CPU} / 8$

Bitfield ADCH specifies the analog input channel to be converted (first channel of a conversion sequence in auto scan modes). Bitfield ADM selects the operating mode of the A/D converter. A conversion (or a sequence) is then started by setting bit ADST. Clearing ADST stops the A/D converter after a specified operation as determined by the selected operating mode.

The busy flag (read-only) ADBSY is set as long as a conversion is in progress.

The result of a conversion is stored in the result register ADDAT, or in register ADDAT2 for an injected conversion.

Note: Bitfield CHNR of register ADDAT is loaded by the ADC to indicate the channel to which the result refers.
Bitfield CHNR of register ADDAT2 is loaded by the CPU to select the analog channel to be injected.

ADDAT

ADC Result Register **SFR ($FEA0_H/50_H$)** **Reset Value: 0000_H**

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

ADDAT2

ADC Chan. Inj. Result Reg. **ESFR ($F0A0_H/50_H$)** **Reset Value: 0000_H**

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

Bit	Function
ADRES	**A/D Conversion Result** The 10-bit digital result of the most recent conversion.
CHNR	**Channel Number** (identifies the converted analog channel)

A conversion is started by setting bit ADST = ‘1’. The busy flag ADBSY will be set. Then the converter selects and samples the input channel specified by the channel selection field ADCH in register ADCON. The sampled level will then be held internally during the conversion. When the conversion of this channel is complete, the 10-bit result and the number of the converted channel are transferred into the result register ADDAT and the interrupt request flag ADCIR is set. The conversion result is placed into bitfield ADRES of register ADDAT.

If bit ADST is reset via software while a conversion is in progress, the A/D converter will stop after the current conversion (fixed channel modes) or after the current conversion sequence (auto scan modes).

Setting bit ADST while a conversion is running will abort this conversion and start a new conversion with the parameters specified in ADCON.

Note: Abort and restart are triggered by bit ADST changing from ‘0’ to ‘1’; thus, ADST must be ‘0’ before being set.

While a conversion is in progress, the mode selection field ADM and the channel selection field ADCH may be changed. ADM will be evaluated after the current conversion. ADCH will be evaluated after the current conversion (fixed channel modes) or after the current conversion sequence (auto scan modes).

Fixed Channel Conversion Modes

These modes are selected by programming the mode selection bitfield ADM in register ADCON to ‘00_B’ (single conversion) or to ‘01_B’ (continuous conversion). After starting the converter through bit ADST, the busy flag ADBSY will be set and the channel specified in bit field ADCH will be converted. After the conversion is complete, the interrupt request flag ADCIR will be set.

In Single Conversion Mode, the converter will automatically stop and reset bits ADBSY and ADST.

In Continuous Conversion Mode, the converter will automatically start a new conversion of the channel specified in ADCH. ADCIR will be set after each completed conversion.

When bit ADST is reset by software while a conversion is in progress, the converter will complete the current conversion and then stop and reset bit ADBSY.

Auto Scan Conversion Modes

These modes are selected by programming the mode selection field ADM in register ADCON to '10_B' (single conversion) or to '11_B' (continuous conversion). Auto Scan modes automatically convert a sequence of analog channels, beginning with the channel specified in bit field ADCH and ending with channel 0, without requiring software to change the channel number.
After starting the converter through bit ADST, the busy flag ADBSY will be set and the channel specified in bit field ADCH will be converted. After the conversion is complete, the interrupt request flag ADCIR will be set and the converter will automatically start a new conversion of the next lower channel. ADCIR will be set after each completed conversion. The current sequence is complete after conversion of channel 0.

In Single Conversion Mode, the converter will automatically stop and reset bits ADBSY and ADST.

In Continuous Conversion Mode, the converter will automatically start a new sequence beginning with the conversion of the channel specified in ADCH.

When bit ADST is reset by software while a conversion is in progress, the converter will complete the current sequence (including conversion of channel 0) and then stop and reset bit ADBSY.

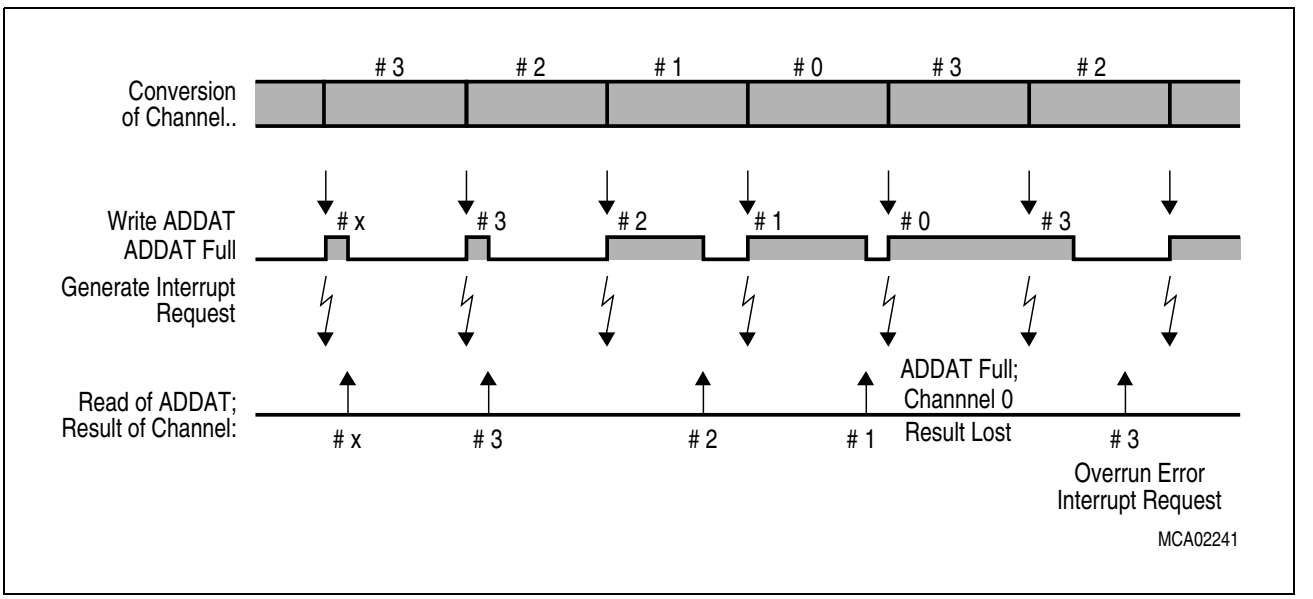


Figure 18-3 Auto Scan Conversion Mode Example

Wait for ADDAT Read Mode

In ADC default mode, if a previous conversion result has not been read out of register ADDAT by the time a new conversion is complete, the previous result in register ADDAT is lost because it is overwritten by the new value, and the A/D overrun error interrupt request flag ADEIR will be set.

To avoid error interrupts and the loss of conversion results (especially when using continuous conversion modes), the ADC can be switched to “Wait for ADDAT Read Mode” by setting bit ADWR in register ADCON.

If the value in ADDAT has not been read by the time the current conversion is complete, the new result is stored in a temporary buffer and the next conversion is suspended (ADST and ADBSY will remain set in the meantime, but no end-of-conversion interrupt will be generated). After reading the previous value from ADDAT, the temporary buffer is copied into ADDAT (generating an ADCIR interrupt) and the suspended conversion is restarted. This mechanism applies to both single and continuous conversion modes.

Note: In standard mode, continuous conversions are executed at a fixed rate (determined by the conversion time). In “Wait for ADDAT Read Mode” there may be delays due to suspended conversions. However, this affects the conversions only if the CPU (or PEC) cannot keep up with the conversion rate.

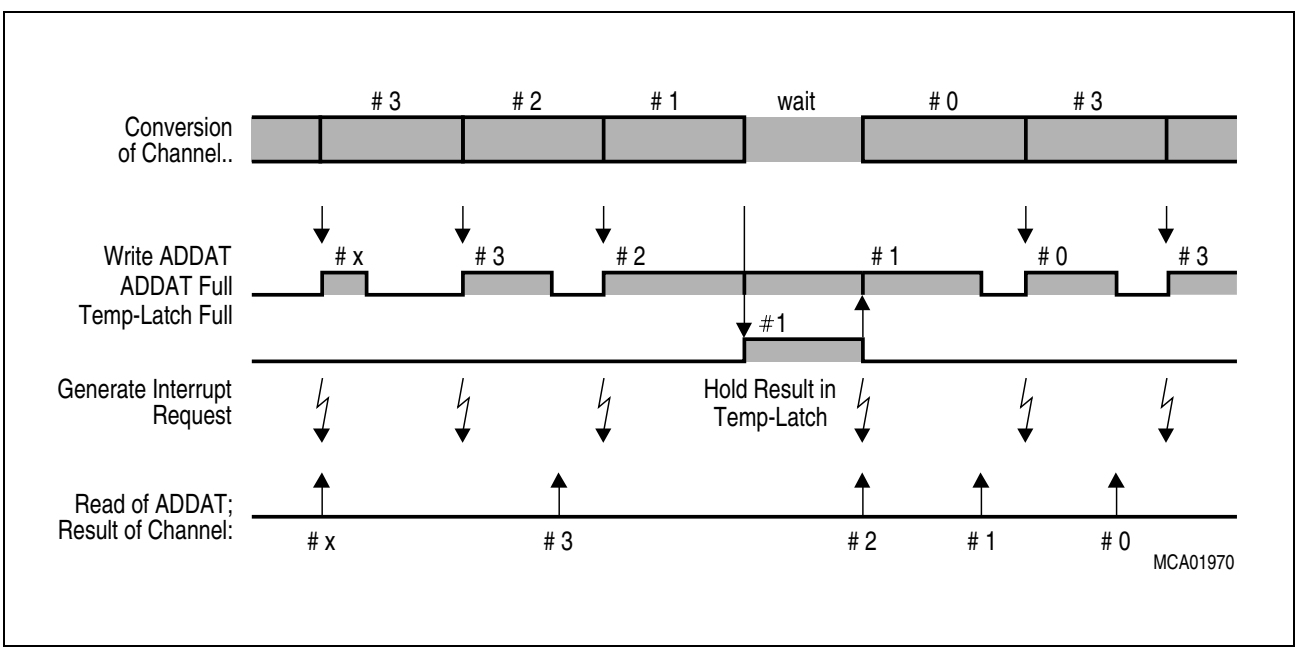


Figure 18-4 Wait for Read Mode Example

Channel Injection Mode

Channel Injection Mode allows conversion of a specific analog channel (also while the ADC is running in a continuous or auto scan mode) without changing the current operating mode. After the conversion of this specific channel, the ADC continues with the original operating mode.

Channel Injection Mode is enabled by setting bit ADCIN in register ADCON and requires the Wait for ADDAT Read Mode (ADWR = '1'). The channel to be converted in this mode is specified in bitfield CHNR of register ADDAT2.

Note: Bitfield CHNR in ADDAT2 is not modified by the A/D converter, only the ADRES bit field is modified. Because the channel number for an injected conversion is not buffered, bit field CHNR of ADDAT2 must never be modified during the sample phase of an injected conversion; otherwise, the input multiplexer will switch to the new channel. It is recommended to change the channel number only when no injected conversion is running.

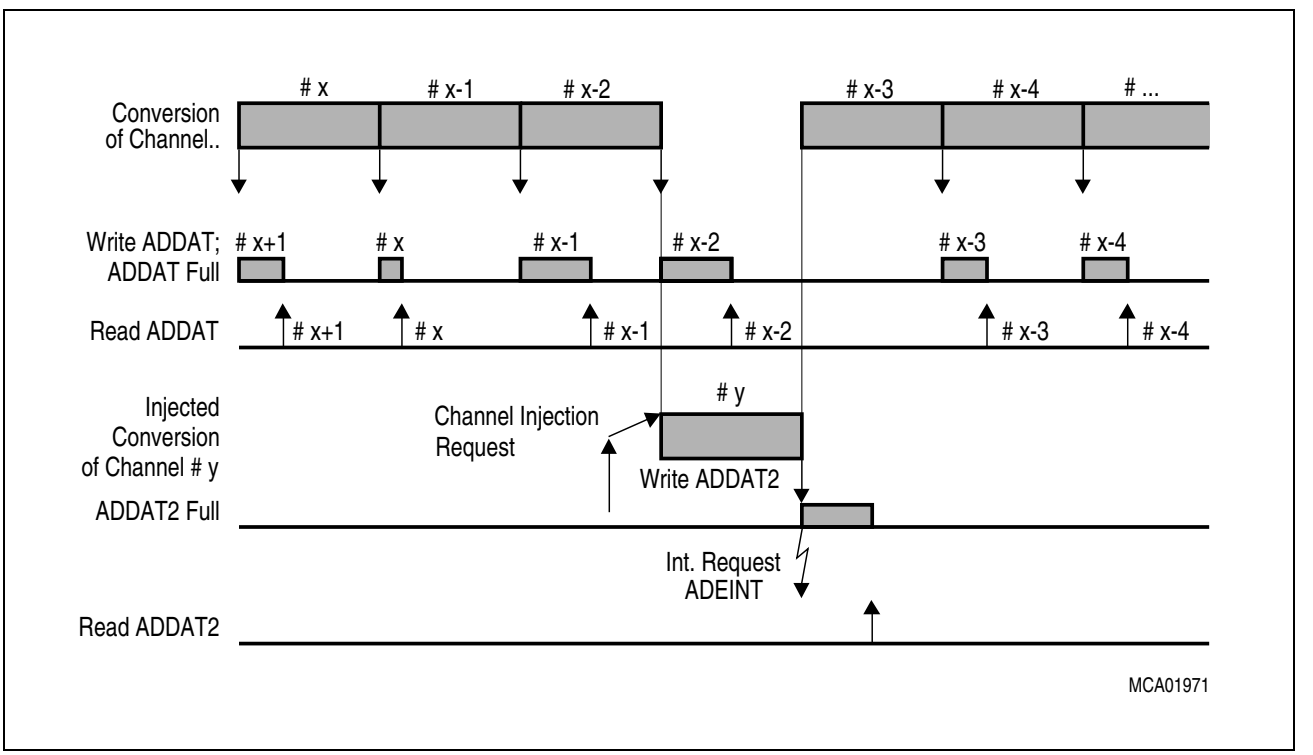


Figure 18-5 Channel Injection Example

A channel injection can be triggered in two ways:

- Set the Channel Injection Request bit ADCRQ via software
- Initiate a compare or a capture event of Capture/Compare register CC27 of the CAPCOM2 unit; this also sets bit ADCRQ.

The second method triggers a channel injection at a specific time; either on the occurrence of a predefined count value of the CAPCOM timers or on a capture event of register CC27. This can be either the positive, the negative, or both the positive and the negative edges of an external signal. Additionally, this option allows the time at which this signal occurs to be recorded.

Note: The channel injection request bit ADCRQ will be set on any interrupt request of CAPCOM2 channel CC27, regardless of whether or not the channel injection mode is enabled. It is recommended to always clear bit ADCRQ before enabling the channel injection mode.

After the completion of the current conversion (if any is in progress), the converter will start (inject) the conversion of the specified channel. When the conversion of this channel is complete, the result will be placed into the alternate result register ADDAT2. A Channel Injection Complete Interrupt request will also be generated which uses the interrupt request flag ADEIR (the Wait for ADDAT Read Mode is required for this reason).

Note: If the temporary data register used in Wait for ADDAT Read Mode is full, the next conversion (either standard or injected) will be suspended. The temporary register can hold data for ADDAT (from a standard conversion) or for ADDAT2 (from an injected conversion).

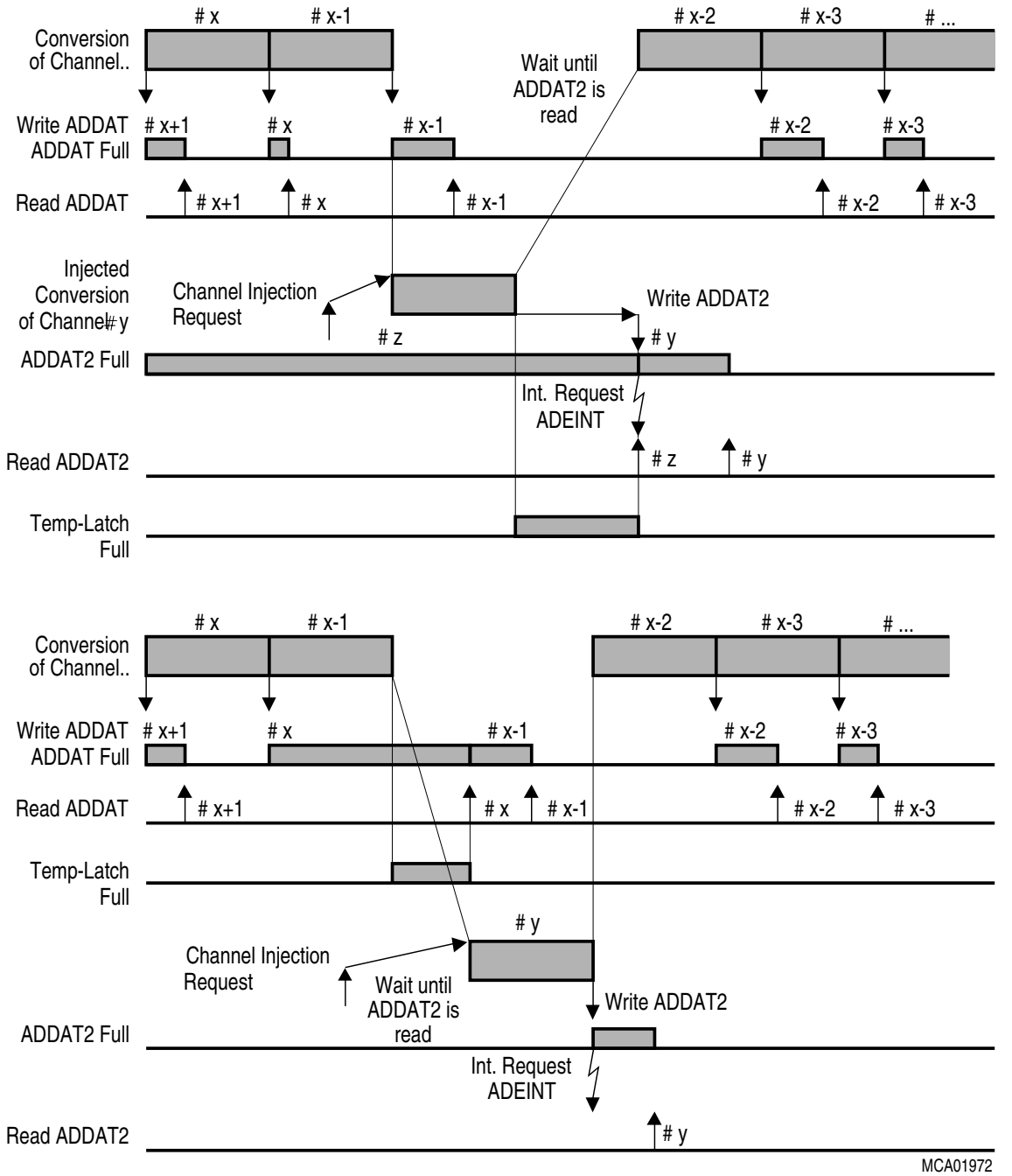


Figure 18-6 Channel Injection Example with Wait for Read

Arbitration of Conversions

Conversion requests activated while the ADC is idle immediately trigger the requested conversion. If a conversion is requested while another conversion is already in progress, the operation of the A/D converter depends on the type of conversions involved (either standard or injected).

Note: A conversion request is activated if the respective control bit (ADST or ADCRQ) is toggled from ‘0’ to ‘1’, i.e. the bit must have been zero before being set.

Table 18-1 summarizes ADC operation in the situations possible.

Table 18-1 Conversion Arbitration

Conversion in Progress	New Requested Conversion	
	Standard	**Injected**
Standard	Abort running conversion, and start requested new conversion.	Complete running conversion, start requested conversion after that.
Injected	Complete running conversion, start requested conversion after that.	Complete running conversion, start requested conversion after that. Bit ADCRQ will be ‘0’ for the second conversion, however.

18.2 Conversion Timing Control

When a conversion is started, the capacitances of the converter are loaded first, via the respective analog input pin to the current analog input voltage. The time to load the capacitances is referred to as sample time. Next, the sampled voltage is converted to a digital value in successive steps corresponding to the resolution of the ADC. During these phases (except for the sample time), the internal capacitances are repeatedly charged and discharged via pins V_{AREF} and V_{AGND}.

The amount of current to be drawn from the sources for sampling and changing charges depends on the time that each respective step takes, because the capacitors must reach their final voltage level within the given time, at least within a certain approximation. The maximum current, however, that a source can deliver, depends on its internal resistance.

The time required by the two sampling and converting phases during conversion can be programmed to be within a certain range in the C164CI relative to the CPU clock. The absolute time consumed by the different conversion steps is therefore, independent from the general speed of the controller. This allows the A/D converter of the C164CI to be adjusted to the properties of the system:

Fast Conversion can be achieved by programming the respective times to their absolute possible minimum. This is preferred for scanning high frequency signals, but the internal resistance of the analog source and analog supply must be sufficiently low.

High Internal Resistance can be achieved by programming the respective times to a higher value, or to the possible maximum. This is preferred when using analog sources and an analog supply with a high internal resistance in order to keep the current as low as possible. The conversion rate in this case may be considerably lower, however.

The conversion time is programmed via the upper two bits of register ADCON. Bitfield ADCTC (conversion time control) selects the basic conversion clock (f_{BC}), used for the operation of the A/D converter. The sample time is derived from this conversion clock. **Table 18-2** lists the possible combinations. The timings refer to CPU clock cycles where $t_{CPU} = 1 / f_{CPU}$.

The limit values for f_{BC} (see data sheet) must not be exceeded when selecting ADCTC and f_{CPU}.

Table 18-2 ADC Conversion Timing Control

ADCON.15|14 (ADCTC)	A/D Converter Basic Clock f_{BC}	ADCON.13|12 (ADSTC)	Sample Time t_S
00	f_{CPU} / 4	00	$t_{BC} \times 8$
01	f_{CPU} / 2	01	$t_{BC} \times 16$
10	f_{CPU} / 16	10	$t_{BC} \times 32$
11	f_{CPU} / 8	11	$t_{BC} \times 64$

The time for a complete conversion includes the sample time t_S, the actual conversion and the time required to transfer the digital value to the result register (2 t_{CPU}) as shown in the example below.

Note: The non-linear decoding of bit field ADCTC provides compatibility with 80C166 designs for the default value ('00' after reset).

Converter Timing Example

Assumptions:	f_{CPU}	= 25 MHz (i.e. t_{CPU} = 40 ns), ADCTC = '00', ADSTC = '00'.
Basic clock	f_{BC}	= f_{CPU} / 4 = 6.25 MHz, i.e. t_{BC} = 160 ns.
Sample time	t_S	= $t_{BC} \times 8$ = 1280 ns.
Conversion time	t_C	= t_S + 40 t_{BC} + 2 t_{CPU} = (1280 + 6400 + 80) ns = 7.76 µs.

Note: For the exact specification please refer to the data sheet of the selected derivative.

18.3 A/D Converter Interrupt Control

At the end of each conversion, the interrupt request flag ADCIR in interrupt control register ADCIC is set. This end-of-conversion interrupt request may cause an interrupt to vector ADCINT, or it may trigger a PEC data transfer which reads the conversion result from register ADDAT and stores it into a table in the internal RAM for later evaluation, for example.

The interrupt request flag ADEIR in register ADEIC will be set if either a conversion result overwrites a previous value in register ADDAT (error interrupt in standard mode), or if the result of an injected conversion has been stored into ADDAT2 (end-of-injected-conversion interrupt). This interrupt request may be used to cause an interrupt to vector ADEINT, or it may trigger a PEC data transfer.

ADCIC
ADC Conversion Intr.Ctrl.Reg. SFR ($FF98_H/CC_H$) Reset Value: - - 00_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								ADC IR	ADC IE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

ADEIC
ADC Error Intr.Ctrl.Reg. SFR ($FF9A_H/CD_H$) Reset Value: - - 00_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								ADE IR	ADE IE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rwh	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description for an explanation of the control fields.

19 On-Chip CAN Interface

The Controller Area Network (CAN) bus and its associated protocol allow highly efficient communication among a number of stations connected to this bus.
Efficiency in this context refers to:

- Transfer speed (Data rates of up to 1 Mbit/s can be achieved)
- Data integrity (The CAN protocol provides several means of error checking)
- Host processor unloading (The controller handles most of the tasks autonomously)
- Flexible and powerful message passing (The extended CAN protocol is supported)

The integrated CAN module handles the completely autonomous transmission and reception of CAN frames in accordance with the CAN specification V2.0 part B (active). Because of this, the on-chip CAN module can receive and transmit:

- **Standard frames** with 11-bit identifiers, as well as
- **Extended frames** with 29-bit identifiers.

Note: The CAN module is an XBUS peripheral and, therefore, requires bit XPEN in register SYSCON to be set in order to be operable.

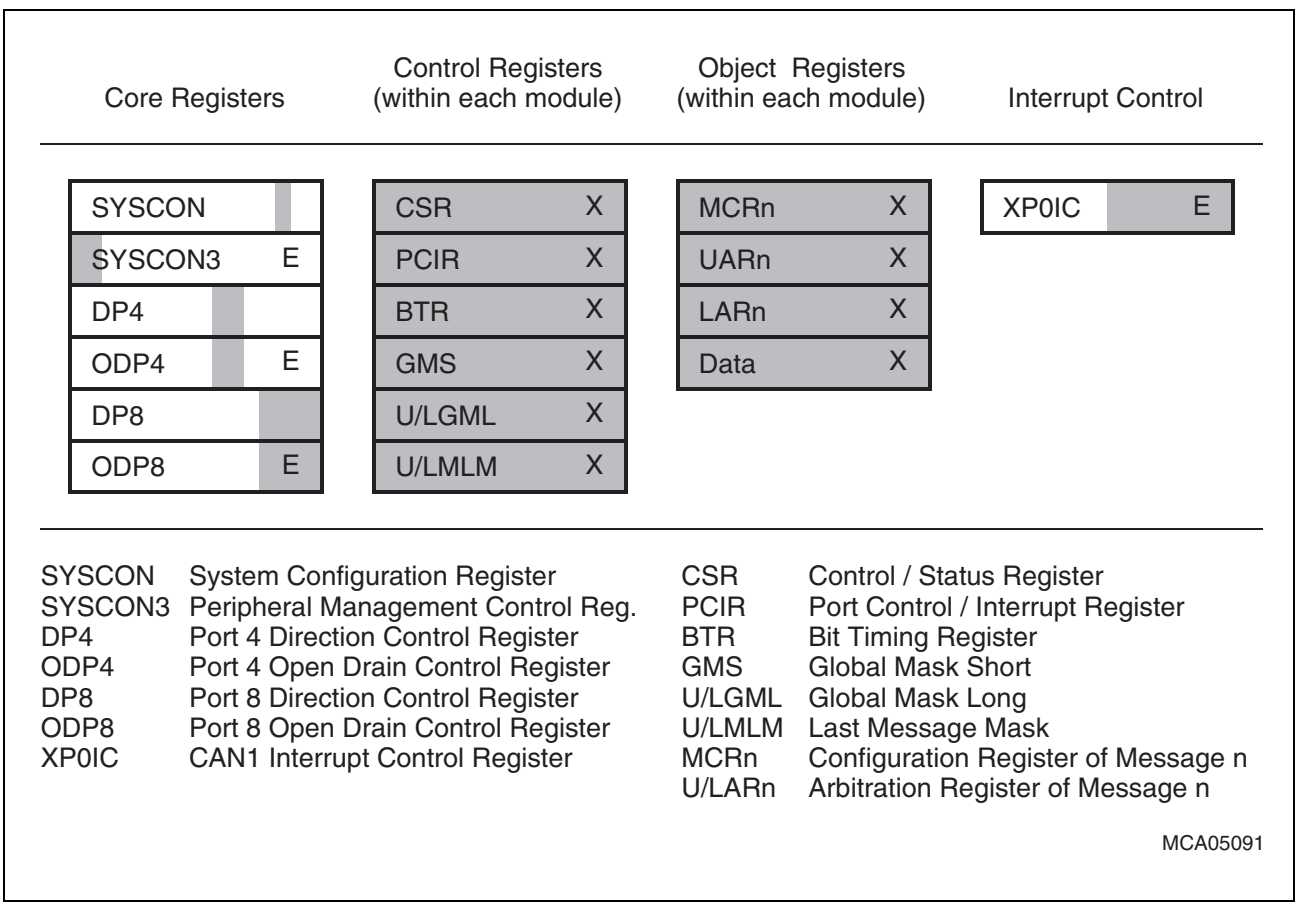


Figure 19-1 Registers Associated with the CAN Module

Bit timing is derived from the XCLK and is programmable up to a data rate of 1 Mbit/s. The minimum CPU clock frequency to achieve 1 Mbit/s is $f_{CPU} \geq 8/16$ MHz, depending on the activation of the CAN module's clock prescaler.

The CAN module uses two pins of Port 4 or Port 8 to interface to a bus transceiver.

The CAN module provides **Full CAN** functionality for up to 15 full-sized message objects (8 data bytes each). Message object 15 may be configured for **Basic CAN** functionality with a double-buffered receive object.

The Full CAN and Basic CAN modes provide separate masks for acceptance filtering to accept a number of identifiers in Full CAN mode and disregard a number of identifiers in Basic CAN mode.

All message objects can be updated independent of the others during operation of the module and are equipped with buffers for the maximum message length of 8 Bytes.

19.1 Functional Blocks of the CAN Module

The CAN module combines several functional blocks (see **Figure 19-2**) that work in parallel and contribute to the controller's performance. These units and the functions they provide are described below.

Each of the message objects has a unique identifier and its own set of control and status bits. Each object can be configured with its direction as either transmit or receive, except for the last message which is only a double receive buffer with a special mask register.

An object with its direction set as transmit can be configured to be automatically sent whenever a remote frame with a matching identifier (taking into account the respective global mask register) is received over the CAN bus. By requesting the transmission of a message with the direction set as receive, a remote frame can be sent to request that the appropriate object be sent by some other node. Each object has separate transmit and receive interrupts and status bits, giving the CPU full flexibility in detecting when a remote/data frame has been sent or received.

For general purposes, two masks for acceptance filtering can be programmed, one for identifiers of 11 bits and one for identifiers of 29 bits. However, the CPU must configure bit XTD (Normal or Extended Frame Identifier) for each valid message to determine whether a standard or extended frame will be accepted.

The last message object has its own programmable mask for acceptance filtering, allowing a large number of infrequent objects to be handled by the system.

The object layer architecture of the CAN controller is designed to be as regular and orthogonal as possible. This makes it easy to use.

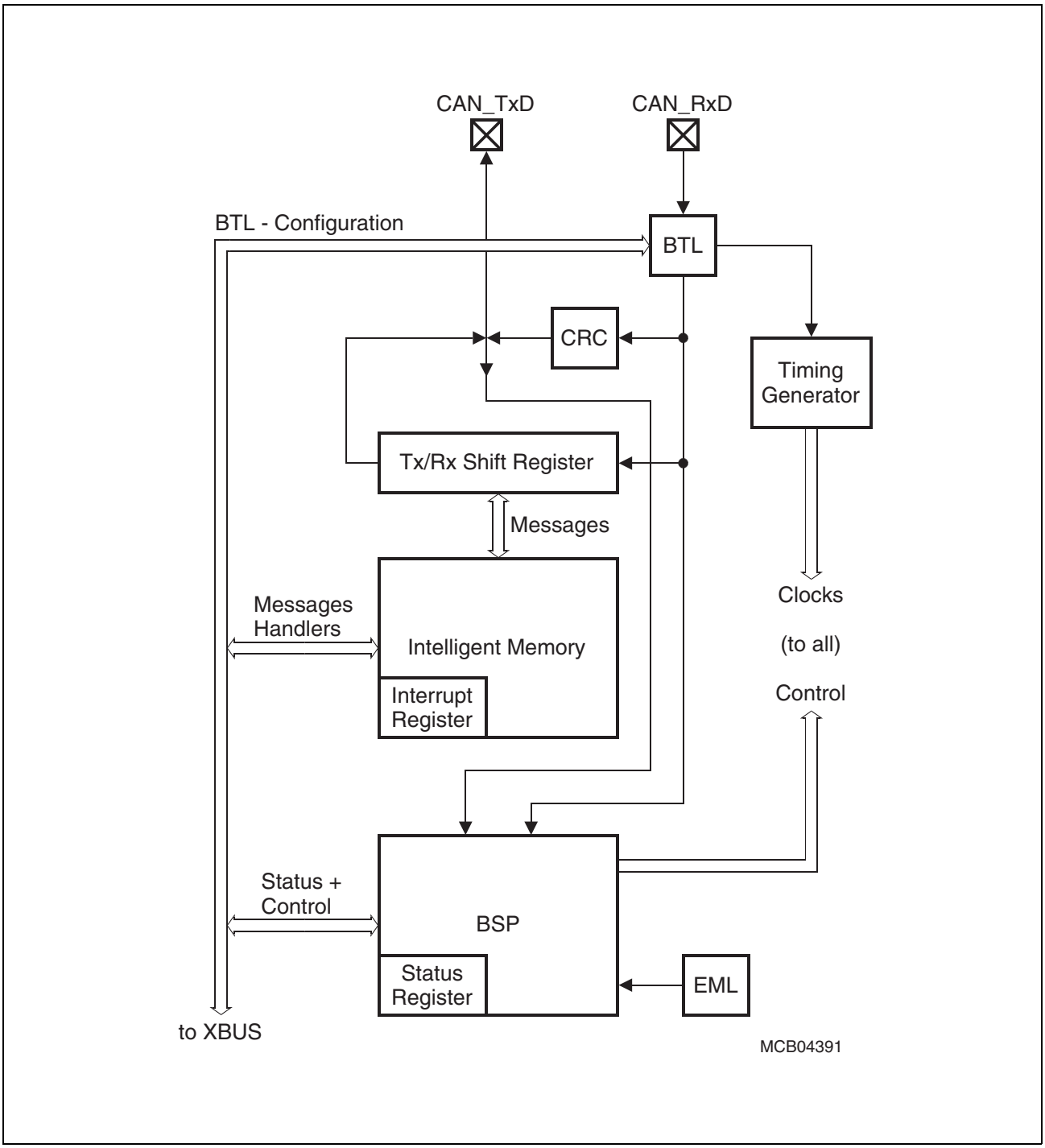


Figure 19-2 CAN Controller Block Diagram

Tx/Rx Shift Register

The Transmit/Receive Shift Register holds the destuffed bit stream from the bus line to allow parallel access to the entire data frame or remote frame for the acceptance match test and parallel transfer of the frame to and from the Intelligent Memory.

Bit Stream Processor

The Bit Stream Processor (BSP) is a sequencer controlling the sequential data stream between the Tx/Rx Shift Register, the CRC Register, and the bus line. The BSP also controls the Error Management Logic (EML) and the parallel data stream between the Tx/Rx Shift Register and the Intelligent Memory such that the processes of reception, arbitration, transmission, and error signalling are performed according to the CAN protocol. Note that the automatic retransmission of messages corrupted by noise or other external error conditions on the bus line is handled by the BSP.

Cyclic Redundancy Check Register

This register generates the Cyclic Redundancy Check (CRC) code to be transmitted after the data bytes and checks the CRC code of incoming messages. This is done by dividing the data stream by the code generator polynomial.

Error Management Logic

The Error Management Logic (EML) is responsible for the fault confinement of the CAN device. Its two counters (the Receive Error Counter and the Transmit Error Counter), are incremented and decremented by commands from the Bit Stream Processor. According to the values of the error counters, the CAN controller is set into one of three states: *error active*, *error passive*, and *busoff*.

The CAN controller states occur as follows:

- *Error active*, if both error counters are below the *error passive* limit of 128.
- *Error passive*, if at least one of the error counters equals or exceeds 128.
- *Busoff*, if the Transmit Error Counter equals or exceeds the *busoff* limit of 256. The device remains in this state until the *busoff* recovery sequence is finished.

Additionally, bit EWRN in the Status Register is set if at least one of the error counters equals or exceeds the error warning limit of 96. EWRN is reset if both error counters are less than the error warning limit.

Bit Timing Logic

The Bit Timing Logic (BTL) monitors the busline input CAN_RXD and handles the busline related bit timing according to the CAN protocol.

The BTL synchronizes on a *recessive* to *dominant* busline transition at *Start of Frame* (hard synchronization) and on any further *recessive* to *dominant* busline transition, if the CAN controller itself does not transmit a *dominant* bit (resynchronization).

The BTL also provides programmable time segments to compensate for the propagation delay time and for phase shifts and to define the position of the *Sample Point* in the bit time. The programming of the BTL depends on the baudrate and on external physical delay times.

Intelligent Memory

The Intelligent Memory (CAM/RAM Array) provides storage for up to 15 message objects of 8 data bytes maximum length. Each of these objects has a unique identifier and its own set of control and status bits. After the initial configuration, the Intelligent Memory can handle the reception and transmission of data without further CPU actions.

Organization of Registers and Message Objects

All registers and message objects of the CAN controller are located in the special CAN address area of 256 Bytes. This area is mapped into segment 0 and uses addresses $00'EF00_H$ through $00'EFFF_H$. All registers are organized as 16-bit registers, located on word addresses. However, all registers may be accessed bytewise in order to select special actions without affecting other mechanisms.

Register Naming reflects the specific name of a register as well as a general module indicator. This results in unique register names.

Example: module indicator **C1** (CAN module 1) and specific register type Control/Status Register (**CSR**) produces unique register name **C1CSR**.

*Note: The address map shown in **Figure 19-3** below lists the registers which are part of the CAN controller. There are also C164CI specific registers associated with the CAN module.*

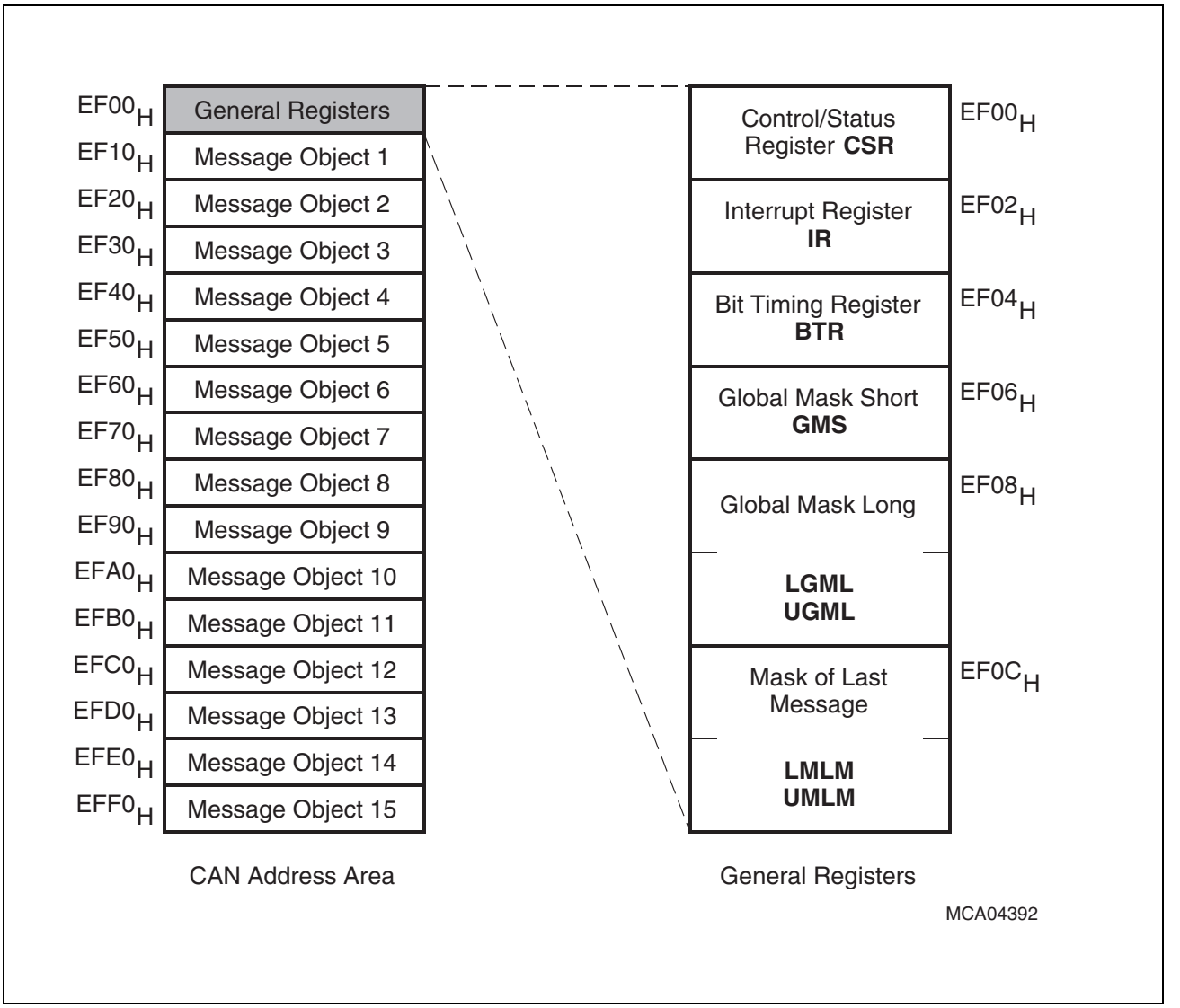


Figure 19-3 CAN Module Address Map

19.2 General Functional Description

The Control/Status Register (CSR) accepts general control settings for the module and provides general status information.

CSR
Control/Status Register **XReg ($EF00_H$)** **Reset Value: $XX01_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BOFF	EWRN	-	RXOK	TXOK	LEC			TM	CCE	0	CPS	EIE	SIE	IE	INIT
rh	rh	r	rwh	rwh	rwh			rw	rw	r	rw	rw	rw	rw	rwh

Bit	Function (Control Bits)
INIT	**Initialization** Starts the initialization of the CAN controller, when set. INIT is set – after a reset – when entering the *busoff* state – by the application software
IE	**Interrupt Enable** Enables or disables interrupt generation from the CAN module via the signal $\overline{\text{XINTR}}$. Does not affect status updates.
SIE	**Status Change Interrupt Enable** Enables or disables interrupt generation when a message transfer (reception or transmission) is successfully completed or a CAN bus error is detected (and registered in the status partition).
EIE	**Error Interrupt Enable** Enables or disables interrupt generation on a change of bit BOFF or EWRN in the status partition).
CPS	**Clock Prescaler Control Bit** 0: **Standard mode:** the input clock is divided 2:1. The minimum input frequency to achieve a baudrate of 1 Mbit/s is f_{CPU} = 16 MHz. 1: **Fast mode:** the input clock is used directly 1:1. The minimum input frequency to achieve a baudrate of 1 Mbit/s is f_{CPU} = 8 MHz.
CCE	**Configuration Change Enable** Allows or inhibits CPU access to the Bit Timing Register (BTR) and the Interface Port Control bit field (IPC) in register PCIR.
TM	**Test Mode** (must be ‘0’) This bit must always be cleared when writing to the Control Register as this bit controls a special test mode used for production testing only. During normal operation, use of this test mode may lead to undesired behavior of the device.

Bit	Function (Control Bits)
LEC	**Last Error Code** This field holds a code indicating the type of error which last occurred on the CAN bus. If a message has been transferred (reception or transmission) without error, this field will be cleared. 0 **No Error** 1 **Stuff Error:** More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed. 2 **Form Error:** Wrong format in fixed format part of a received frame. 3 **AckError:** The message transmitted by this CAN controller was not acknowledged by another node. 4 **Bit1Error:** During the transmission of a message (with the exception of the arbitration field), the device wanted to send a *recessive* level (“1”), but the monitored bus value was *dominant*. 5 **Bit0Error:** During the transmission of a message (or acknowledge bit, active error flag, or overload flag), the device wanted to send a *dominant* level (“0”), but the monitored bus value was *recessive*. During *busoff* recovery this status is set each time a sequence of 11 *recessive* bits has been monitored. This enables the CPU to monitor the proceeding of the busoff recovery sequence (indicates that the bus is not stuck at *dominant* or continuously disturbed). 6 **CRCError:** The received CRC check sum was incorrect. 7 **Unused code:** may be written by the CPU to check for updates.
TXOK	**Transmitted Message Successfully** Indicates that a message has been transmitted successfully (error free and acknowledged by at least one other node), since this bit was last reset by the CPU (the CAN controller does not reset this bit!).
RXOK	**Received Message Successfully** This bit is set each time a message has been received successfully, since this bit was last reset by the CPU (the CAN controller does not reset this bit!). RXOK is also set when a message is received that is not accepted (i.e. stored).
EWRN	**Error Warning Status** Indicates that at least one of the error counters in the EML has reached the error warning limit of 96.
BOFF	**Busoff Status** Indicates when the CAN controller is in busoff state (see EML).

Note: Reading the upper half of the Control Register (status partition) will clear the Status Change Interrupt value in the Interrupt Register, if it is pending. Using byte accesses to the lower half will avoid this.

19.2.1 CAN Interrupt Handling

The on-chip CAN module has one interrupt output. It is connected through a synchronization stage to a standard interrupt node in the C164CI in the same manner as all other interrupts of the standard on-chip peripherals. All control options are available for this interrupt, such as enabling/disabling, level and group priority, and interrupt or PEC service (see note below). The on-chip CAN module is connected to an XBUS interrupt control register.

As for all other interrupts, the node interrupt request flag is cleared automatically by hardware when this interrupt is serviced (either by standard interrupt or PEC service).

Note: As a rule, CAN interrupt requests can be serviced by a PEC channel. However, because PEC channels can execute only single predefined data transfers (there are no conditional PEC transfers), PEC service can be used only if the respective request is known to be generated by one specific source, and on condition that no other interrupt request will be generated in between. In practice, this seems to be rare.

Because an interrupt request of the CAN module can be generated by various conditions, the appropriate CAN interrupt status register must be read in the service routine to determine the cause of the interrupt request. The interrupt identifier INTID (a number) in the Port Control/Interrupt Register (PCIR) indicates the cause of an interrupt. When no interrupt is pending, the identifier will have the value 00_H.

If the value in INTID is not 00_H, then there is an interrupt pending. If bit IE in the control/status register is also set, the interrupt signal to the CPU is activated. The interrupt signal (to the interrupt node) remains active until INTID becomes 00_H (all interrupt requests have been serviced) or until interrupt generation is disabled (CSR.IE = ‘0’).

Note: The interrupt node is activated only upon a 0 → 1 transition of the CAN interrupt signal. The CAN interrupt service routine should only be exited after INTID has been verified to be 00_H.

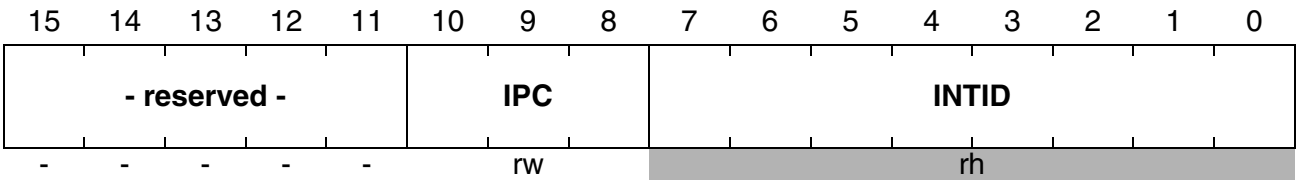
The interrupt with the lowest number has the highest priority. If a higher priority interrupt (lower number) occurs before the current interrupt is processed, INTID is updated and the new interrupt overrides the last one.

INTID is also updated after the respective source request has been processed. This is indicated by clearing the INTPND flag in the respective object’s message control register (MCRn) or by reading the status partition of register CSR (in the case of a status change interrupt). The updating of INTID is done by the CAN state machine and takes up to 6 CAN clock cycles, depending on current state of the state machine (1 CAN clock cycle = 1 or 2 CPU clock cycles, as determined by the prescaler bit CPS).

*Note: A worst case condition can occur when BRP = 00_H **AND** the CAN controller is storing a message just received **AND** the CPU is executing consecutive accesses to the CAN module. In this rare case, the maximum delay may be 26 CAN clock cycles.*
The impact of this delay can be minimized by clearing bit INTPND at an early stage

of interrupt processing, and (if required) restricting CPU accesses to the CAN module until the anticipated update is complete.

PCIR
Port Control / Interrupt Register XReg ($EF02_H$) **Reset Value: $XXXX_H$**



Bit	Function	
INTID	**Interrupt Identifier** This number indicates the cause of the interrupt (if pending).	
	00_H	**Interrupt Idle:** There is no interrupt request pending.
	01_H	**Status Change Interrupt:** The CAN controller has updated (not necessarily changed) the status in the Control Register. This can refer to a change of the error status of the CAN controller (EIE is set and BOFF or EWRN change) or to a CAN transfer incident (SIE must be set), such as reception or transmission of a message (RXOK or TXOK is set) or the occurrence of a CAN bus error (LEC is updated). The CPU may clear RXOK, TXOK, and LEC, however, writing to the status partition of the Control Register can never generate or reset an interrupt. The status partition of the Control Register must be read to update the INTID value.
	02_H	**Message 15 Interrupt:** Bit INTPND in the Message Control Register of message object 15 (last message) has been set. The last message object has the highest interrupt priority of all message objects.[1)]
	(02 + N)	**Message N Interrupt:** Bit INTPND in the Message Control Register of message object 'N' has been set (N = 1 … 14). Note that a message interrupt code is only displayed, if there is no other interrupt request with a higher priority.[1)] Example: message 1: INTID = 03_H, message 14: INTID = 10_H
IPC	**Interface Port Control** (reset value = 111_B, i.e. no port connection) The encoding of bitfield IPC is described in **Section 19.6**. *Note: Bitfield IPC can be written only while bit CCE is set.*	

1) Bit INTPND of the corresponding message object has to be cleared to give messages with a lower priority the possibility to update INTID or to reset INTID to "00_H" (idle state).

19.2.2 Configuration of the Bit Timing

According to the CAN protocol specification, a bit time is subdivided into four segments: Sync segment, propagation time segment, phase buffer segment 1 and phase buffer segment 2.

Each segment is a multiple of the time quantum t_q, with

$t_q = (BRP + 1) \times 2^{(1 - CPS)} \times t_{XCLK}$.

Note: The CAN module is connected to the CPU clock signal, therefore $t_{XCLK} = t_{CPU}$.

The Synchronization Segment (Sync Seg) is always 1 t_q long. The Propagation Time Segment and the Phase Buffer Segment 1 (combined to TSeg1) define the time before the sample point, while Phase Buffer Segment 2 (TSeg2) defines the time after the sample point. The length of these segments is programmable (except Sync-Seg) via the Bit Timing Register (BTR).

Note: For exact definition of these segments please refer to the CAN Protocol Specification.

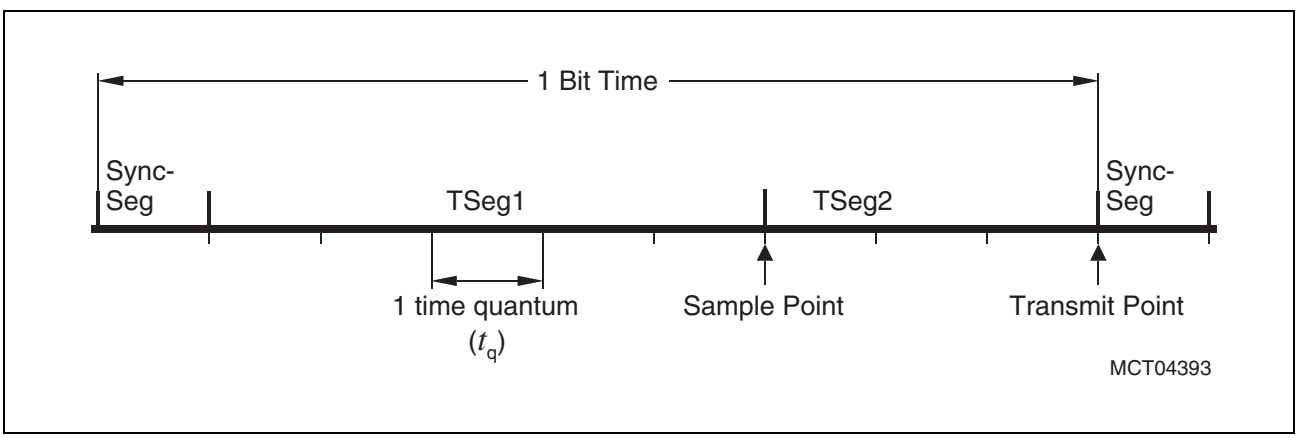


Figure 19-4 Bit Timing Definition

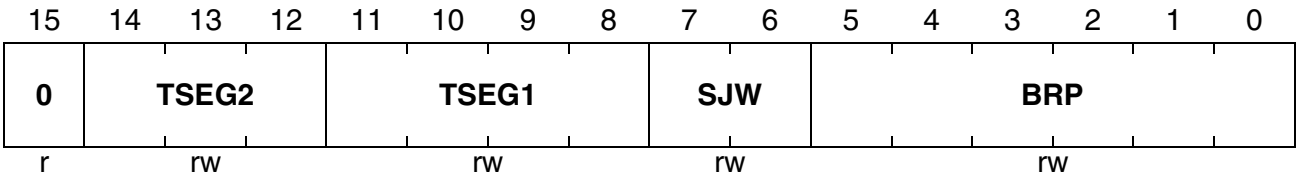
The bit time is determined by the XBUS clock period t_{XCLK}, the Baud Rate Prescaler, and the number of time quanta per bit:

$$\text{bit time} = t_{Sync\text{-}Seg} + t_{TSeg1} + t_{TSeg2} \quad [19.1]$$

$$t_{Sync\text{-}Seg} = 1 \times t_q$$
$$t_{TSeg1} = (TSEG1 + 1) \times t_q$$
$$t_{TSeg2} = (TSEG2 + 1) \times t_q$$
$$t_q = (BRP + 1) \times 2^{(1 - CPS)} \times t_{XCLK} \quad [19.2]$$

Note: TSEG1, TSEG2, and BRP are the programmed numerical values from the respective fields of the Bit Timing Register.

BTR
Bit Timing Register **XReg ($EF04_H$)** **Reset Value: $UUUU_H$**



Bit	Function
BRP	**Baud Rate Prescaler** To generate the bit time quanta, the CPU frequency f_{CPU} is divided by $2^{(1 - CPS)} \times$ (BRP + 1). See also the prescaler control bit CPS in register CSR.
SJW	**(Re)Synchronization Jump Width** Adjust the bit time by maximum (SJW + 1) time quanta for resynchronization.
TSEG1	**Time Segment before sample point** There are (TSEG1 + 1) time quanta before the sample point. Valid values for TSEG1 are “2 … 15”.
TSEG2	**Time Segment after sample point** There are (TSEG2 + 1) time quanta after the sample point. Valid values for TSEG2 are “1 … 7”.

Note: This register can only be written, if the config. change enable bit (CCE) is set.

Hard Synchronization and Resynchronization

To compensate for phase shifts between clock oscillators of different CAN controllers, any CAN controller must synchronize on any edge from recessive to dominant bus level if the edge lies between a Sample Point and the next Synchronization Segment, and on any other edge if it does not send a dominant level itself. If the Hard Synchronization is enabled (at the Start of Frame), the bit time is restarted at the Synchronization Segment; otherwise the Resynchronization Jump Width (SJW) defines the maximum number of time quanta by which a bit time may be shortened or lengthened during one Resynchronization. The current bit time is adjusted by

$t_{SJW} = (SJW + 1) \times t_q$

Note: SJW is the programmed numerical value from the respective field of the Bit Timing Register.

Calculation of the Bit Time

Programming the bit time according to the CAN Specification depends on the desired baudrate, the XCLK frequency, and the external physical delay times of the bus driver, the bus line, and the input comparator. These delay times are summarized in the Propagation Time Segment t_{Prop}, where

t_{Prop} is two times the maximum of the sum of physical bus delay, the input comparator delay, and the output driver delay rounded up to the nearest multiple of t_q.

To fulfill the requirements of the CAN specification, the following conditions must be met:

$t_{TSeg2} \geq 2 \times t_q$ = *Information Processing Time*

$t_{TSeg2} \geq t_{SJW}$

$t_{TSeg1} \geq 3 \times t_q$

$t_{TSeg1} \geq t_{SJW} + t_{Prop}$

Note: In order to achieve correct operation according to the CAN protocol, the total bit time should be at least 8 t_q, i.e. TSEG1 + TSEG2 ≥ 5.
Thus, to operate with a baudrate of 1 MBit/s, the XCLK frequency must be at least 8/16 MHz (depending on the prescaler control bit CPS in register CSR).

The maximum tolerance df for XCLK depends on the Phase Buffer Segment 1 (PB1), the Phase Buffer Segment 2 (PB2), and the Resynchronization Jump Width (SJW):

$$df \leq \frac{\min(PB1, PB2)}{2 \times (13 \times \text{bit time} - PB2)}$$

AND

$$df \leq \frac{t_{SJW}}{20 \times \text{bit time}}$$

The following examples illustrate bit timing calculations under specific circumstances.

Bit Timing Example for High Baudrate

This example makes the following assumptions:

- XCLK frequency = 20 MHz
- BRP = 00, CPS = 0
- Baudrate = 1 Mbit/s

t_q	100 ns	$= 2 \times t_{XCLK}$
bus driver delay	50 ns	
receiver circuit delay	30 ns	
bus line (40 m) delay	220 ns	
t_{Prop}	600 ns	$= 6 \times t_q$
t_{SJW}	100 ns	$= 1 \times t_q$
t_{TSeg1}	700 ns	$= t_{Prop} + t_{SJW}$
t_{TSeg2}	200 ns	$=$ *Information Processing Time*
t_{Sync}	100 ns	$= 1 \times t_q$
t_{Bit}	1000 ns	$= t_{Sync} + t_{TSeg1} + t_{TSeg2}$
tolerance for t_{XCLK}	0.39%	$= \frac{\min(PB1, PB2)}{2 \times (13 \times \text{bit time} - PB2)}$
		$= \frac{0{,}1\mu s}{2 \times (13 \times 1\mu s - 0{,}2\mu s)}$

Bit Timing Example for Low Baudrate

This example makes the following assumptions:

- XCLK frequency = 4 MHz
- BRP = 01, CPS = 0
- Baudrate = 100 kbit/s

t_q	1 µs	$= 4 \times t_{XCLK}$
bus driver delay	200 ns	
receiver circuit delay	80 ns	
bus line (40 m) delay	220 ns	
t_{Prop}	1 µs	$= 1 \times t_q$
t_{SJW}	4 µs	$= 4 \times t_q$
t_{TSeg1}	5 µs	$= t_{Prop} + t_{SJW}$
t_{TSeg2}	4 µs	$=$ *Information Processing Time* $+ 2 \times t_q$
t_{Sync}	1 µs	$= 1 \times t_q$
t_{Bit}	10 µs	$= t_{Sync} + t_{TSeg1} + t_{TSeg2}$
tolerance for f_{XCLK}	1.58%	$= \frac{\min(PB1, PB2)}{2 \times (13 \times \text{bit time} - PB2)}$
		$= \frac{4\mu s}{2 \times (13 \times 10\mu s - 4\mu s)}$

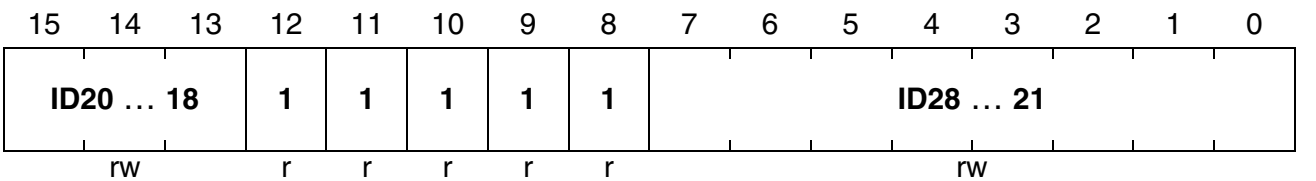
19.2.3 Mask Registers

Messages can use either standard or extended identifiers. Incoming frames are masked with their appropriate global masks. Bit IDE of the incoming message determines whether the standard 11-bit mask in Global Mask Short (GMS), or the 29-bit extended mask in Global Mask Long (UGML&LGML) is to be used. Bits holding a “0” are “don’t care”, that is, do not compare the message’s identifier in the respective bit position.

The last message object (15) has an additional individually programmable acceptance mask (Mask of Last Message, UMLM&LMLM) for the complete arbitration field. This allows classes of messages to be received in this object by masking some bits of the identifier.

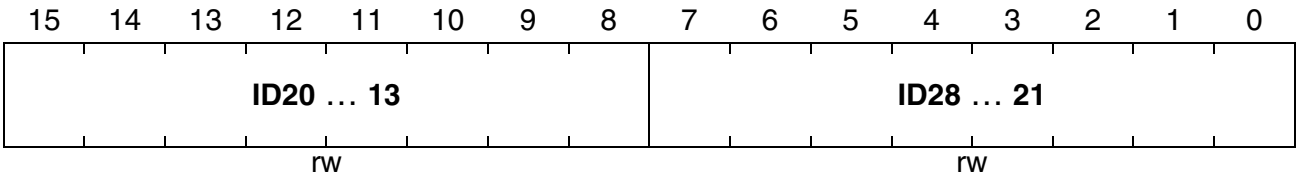
Note: The Mask of Last Message is ANDed with the Global Mask that corresponds to the incoming message.

GMS
Global Mask Short **XReg ($EF06_H$)** **Reset Value: $UFUU_H$**

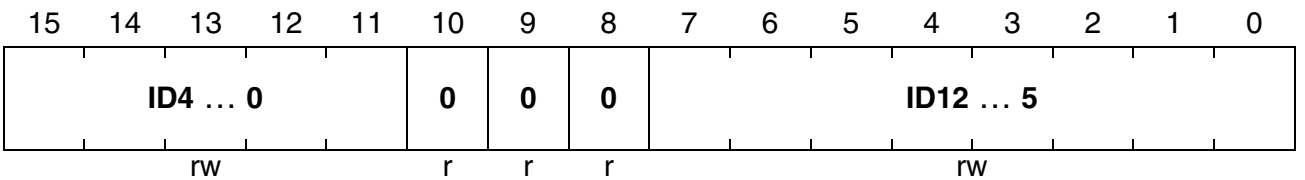


Bit	Function
ID28 … 18	**Identifier (11-bit)** Mask to filter incoming messages with standard identifier.

UGML
Upper Global Mask Long **XReg ($EF08_H$)** **Reset Value: $UUUU_H$**

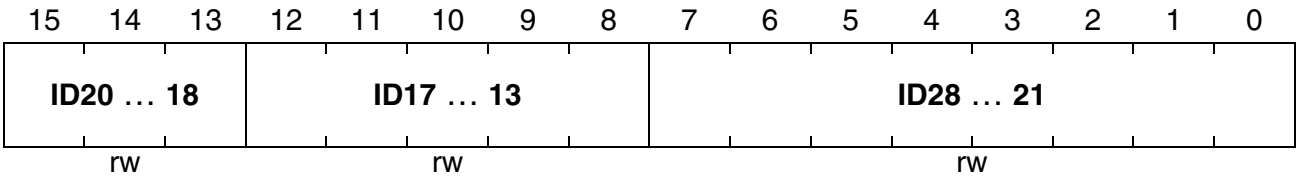


LGML
Lower Global Mask Long **XReg ($EF0A_H$)** **Reset Value: $UUUU_H$**

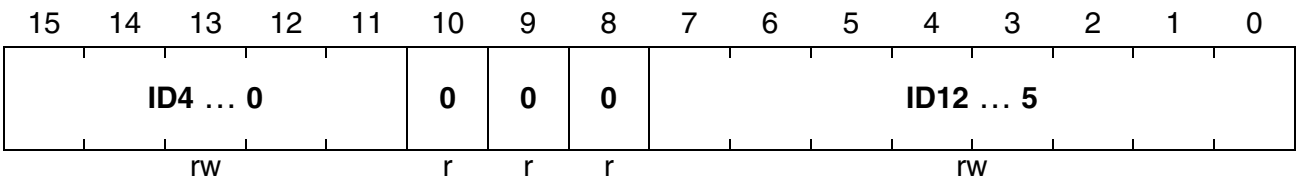


Bit	Function
ID28 … 0	**Identifier (29-bit)** Mask to filter incoming messages with extended identifier.

UMLM
Upper Mask of Last Message **XReg ($EF0C_H$)** **Reset Value: $UUUU_H$**



LMLM
Lower Mask of Last Message **XReg ($EF0E_H$)** **Reset Value: $UUUU_H$**



Bit	Function
ID28 … 0	**Identifier (29-bit)** Mask to filter the last incoming message (Nr. 15) with standard or extended identifier (as configured).

19.3 The Message Object

The message object is the primary means of communication between the CPU and the CAN controller. Each of the 15 message objects uses 15 consecutive bytes (see **Figure 19-5**) and starts at an address that is a multiple of 16.

Note: All message objects must be initialized by the CPU before clearing the INIT bit, even those which are not going to be used.

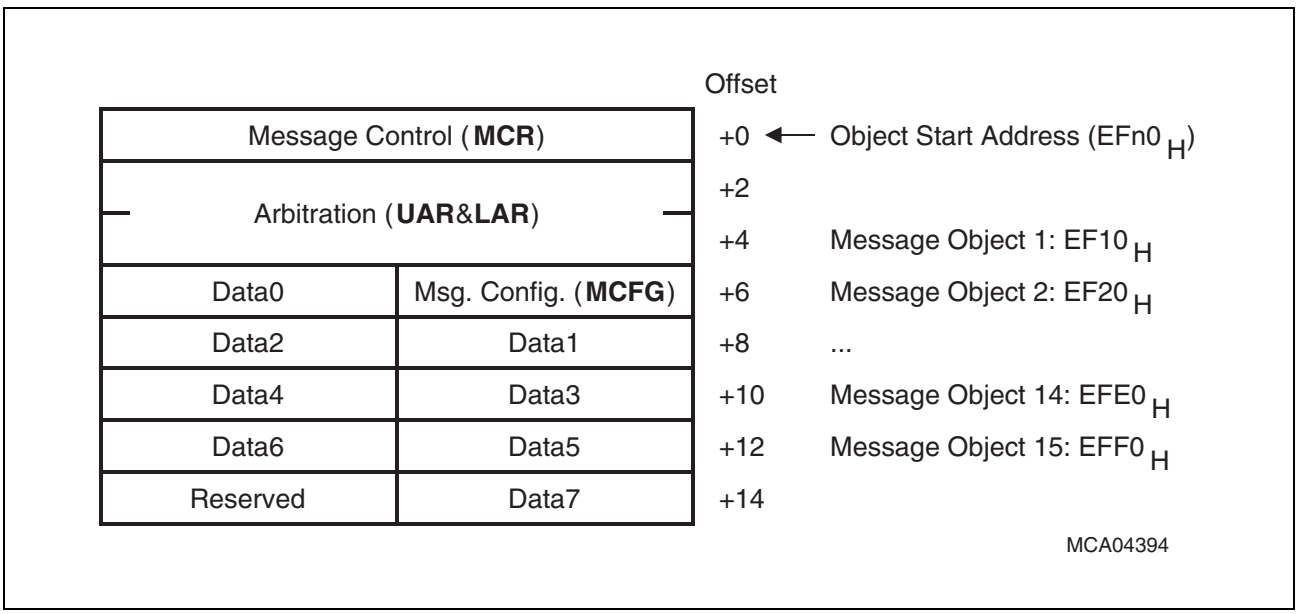


Figure 19-5 Message Object Address Map

The general properties of a message object are defined via the Message Control Register (MCR). There is a dedicated register MCRn for each message object n.

Each element of the Message Control Register consists of two complementary bits. This special mechanism allows the selective setting or resetting of specific elements (leaving others unchanged) without requiring read-modify-write cycles. None of these elements will be affected by reset.

Table 19-1 shows the functions and meanings of these 2-bit fields.

Table 19-1 MCR Bitfield Encoding

Value	Function on Write	Meaning on Read
0 0	– Reserved –	– Reserved –
0 1	Reset element	Element is reset
1 0	Set element	Element is set
1 1	Leave element unchanged	– Reserved –

MCRn
Message Control Register **XReg ($EFn0_H$)** **Reset Value: $UUUU_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RMTPND		TXRQ		MSGLST CPUUPD		NEWDAT		MSGVAL		TXIE		RXIE		INTPND	
rw		rw		rw		rw		rw		rw		rw		rw	

Bit	Function
INTPND	**Interrupt Pending** Indicates whether this message object has generated an interrupt request (see TXIE and RXIE), since this bit was last reset by the CPU.
RXIE	**Receive Interrupt Enable** Defines whether bit INTPND is set after successful reception of a frame.
TXIE	**Transmit Interrupt Enable** Defines whether bit INTPND is set after successful transmission of a frame.[1)]
MSGVAL	**Message Valid** Indicates whether the corresponding message object is valid. The CAN controller only operates on valid objects. Message objects can be tagged invalid, while they are changed, or if they are not used at all.
NEWDAT	**New Data** Indicates whether new data has been written into the data portion of this message object by CPU (transmit-objects) or CAN controller (receive-objects) since this bit was last reset.[2)]
MSGLST	**Message Lost** (This bit applies to receive-objects only!) Indicates that the CAN controller has stored a new message into this object while NEWDAT was still set; thus, the previously stored message is lost.
CPUUPD	**CPU Update** (This bit applies to transmit-objects only!) Indicates that the corresponding message object may not be transmitted now. The CPU sets this bit to inhibit transmission of a message currently being updated, or to control the automatic response to remote requests.
TXRQ	**Transmit Request** Indicates that the transmission of this message object is requested by the CPU or via a remote frame and is not yet complete. TXRQ can be disabled by CPUUPD.[1)3)]

Bit	Function
RMTPND	**Remote Pending** (Used for transmit-objects) Indicates that the transmission of this message object has been requested by a remote node, but the data has not yet been transmitted. When RMTPND is set, the CAN controller also sets TXRQ. Bits RMTPND and TXRQ are cleared when the message object has been successfully transmitted.

1) In message object 15 (last message) these bits are hardwired to “0” (inactive) in order to prevent transmission of message 15.

2) When the CAN controller writes new data into the message object, unused message bytes will be overwritten by non specified values. Usually the CPU will clear this bit before working on the data, and verify that the bit is still cleared once it has finished working to ensure that it has worked on a consistent set of data and not part of an old message and part of the new message.
For transmit-objects the CPU will set this bit along with clearing bit CPUUPD. This will ensure that, if the message is actually being transmitted during the time the message was being updated by the CPU, the CAN controller will not reset bit TXRQ. In this way bit TXRQ is only reset once the actual data has been transferred.

3) When the CPU requests the transmission of a receive-object, a remote frame will be sent instead of a data frame to request a remote node to send the corresponding data frame. This bit will be cleared by the CAN controller along with bit RMTPND when the message has been successfully transmitted, if bit NEWDAT has not been set.
If there are several valid message objects with pending transmission request, the message with the lowest message number is transmitted first. This arbitration is done when several objects are requested for transmission by the CPU, or when operation is resumed after an error frame or after arbitration has been lost.

Arbitration Registers

The Arbitration Registers (UARn&LARn) are used for acceptance filtering of incoming messages and to define the identifier of outgoing messages. A received message with a matching identifier is accepted as a data frame (matching object has DIR = ‘0’) or as a remote frame (matching object has DIR = ‘1’). For matching, the corresponding Global Mask must be considered (the Mask of Last Message must also be considered for message object 15). Extended frames (using Global Mask Long) can be stored only in message objects with XTD = ‘1’, standard frames (using Global Mask Short) can be stored only in message objects with XTD = ‘0’.

Message objects should have unique identifiers such that if some bits are masked out by the Global Mask Registers (“don’t care” bits), then the identifiers of the valid message objects should differ in the remaining bits which are used for acceptance filtering.

If a received message (data frame or remote frame) matches with more than one valid message object, it is associated with the object having the lowest message number. Thus, a received data frame is stored in the “lowest” object, or the “lowest” object is sent in response to a remote frame. The Global Mask is used for this matching.

After a transmission (data frame or remote frame) the transmit request flag of the matching object with the lowest message number is cleared. The Global Mask is not used in this case.

When the CAN controller accepts a data frame, the complete message is stored into the corresponding message object including the identifier (also masked bits, standard identifiers have bits ID17-0 filled with '0'), the data length code (DLC), and the data bytes (valid bytes indicated by DLC). This is implemented to keep the data bytes connected with the identifier even if arbitration mask registers are used.

When the CAN controller accepts a remote frame, the corresponding transmit message object (1 … 14) remains unchanged except for bits TXRQ and RMTPND, which are set. In the last message object 15 (which cannot start a transmission), the identifier bits corresponding to the "don't care" bits of the Last Message Mask are copied from the received frame. Bits corresponding to the "don't care" bits of the corresponding global mask are not copied (bits masked out by the global **and** the last message mask cannot be retrieved from object 15).

UARn
Upper Arbitration Register **XReg ($EFn2_H$)** **Reset Value: $UUUU_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID20 … 18			**ID17 … 13**					**ID28 … 21**							
			rw					rw							

LARn
Lower Arbitration Register **XReg ($EFn4_H$)** **Reset Value: $UUUU_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID4 … 0					**0**	**0**	**0**	**ID12 … 5**							
rw					r	r	r	rw							

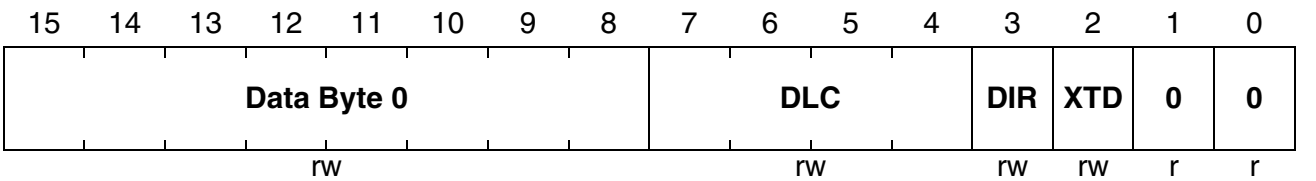
Bit	Function
ID28 … 0	**Identifier (29-bit)** Identifier of a standard message (ID28 … 18) or an extended message (ID28 … 0). For standard identifiers, bits ID17 … 0 are "don't care".

Message Configuration

The Message Configuration Register (low byte of MCFGn) holds a description of the message within this object.

Note: There is no “don’t care” option for bits XTD and DIR. So, incoming frames can only match with corresponding message objects either standard (XTD = 0) or extended (XTD = 1). Data frames match with receive-objects only; remote frames match with transmit-objects only.
When the CAN controller stores a data frame, it will write all the eight data bytes into a message object. If the data length code was less than 8, the remaining bytes of the message object will be overwritten by non-specified values.

MCFGn
Message Configuration Reg. XReg ($EFn6_H$) Reset Value: - - UU_H



Bit	Function
XTD	**Extended Identifier** 0: **Standard** This message object uses a standard 11-bit identifier. 1: **Extended** This message object uses an extended 29-bit identifier.
DIR	**Message Direction** 0: **Receive Object.** On TXRQ, a remote frame with the identifier of this message object is transmitted. On reception of a data frame with matching identifier, that message is stored in this message object. 1: **Transmit Object.** On TXRQ, the respective message object is transmitted. On reception of a remote frame with matching identifier, the TXRQ and RMTPND bits of this message object are set.
DLC	**Data Length Code** Defines the number of valid data bytes within the data area. Valid values for the data length are 0 … 8.

Note: The first data byte occupies the upper half of the message configuration register.

Data Area

The data area occupies 8 successive byte positions after the Message Configuration Register such that the data area of message object **n** covers locations 00'EFn7$_H$ through 00'EFnE$_H$.
Location 00'EFnF$_H$ is reserved.

Message data for message object 15 (last message) will be written into a two-message-alternating buffer to avoid the loss of a message, if a second message has been received, before the CPU has read the first one.

Handling of Message Objects

Figure 19-6 through **Figure 19-11** summarize the actions which must be taken to transmit and receive messages over the CAN bus. The actions taken by the CAN controller are described as well as the actions to be taken by the CPU (the servicing program).

The diagrams show these actions:

- CAN controller handling of transmit objects
- CAN controller handling of receive objects
- CPU handling of transmit objects
- CPU handling of receive objects
- CPU handling of last message object
- Handling of the last message's alternating buffer

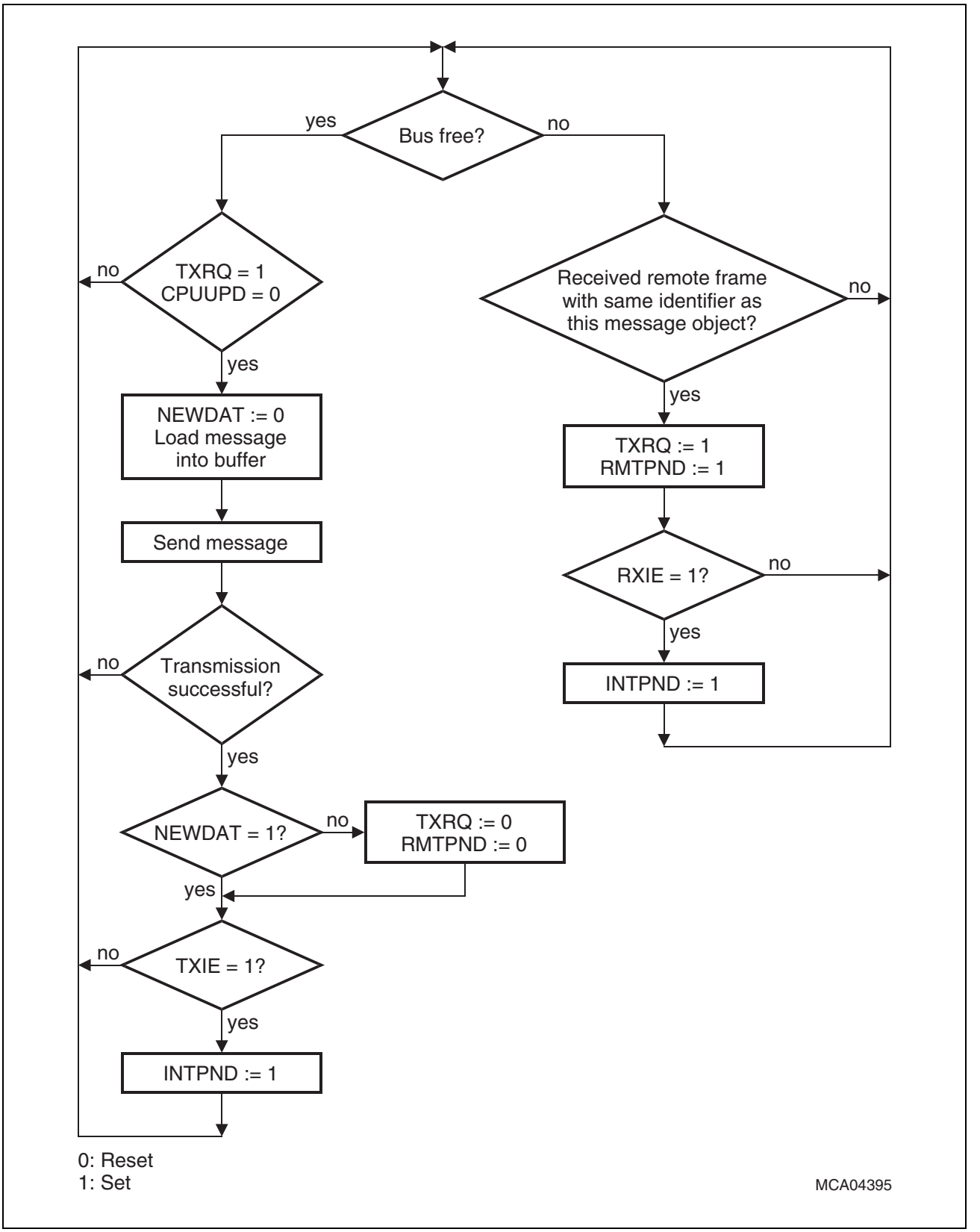


Figure 19-6 CAN Controller Handling of Transmit Objects (DIR = ‘1’)

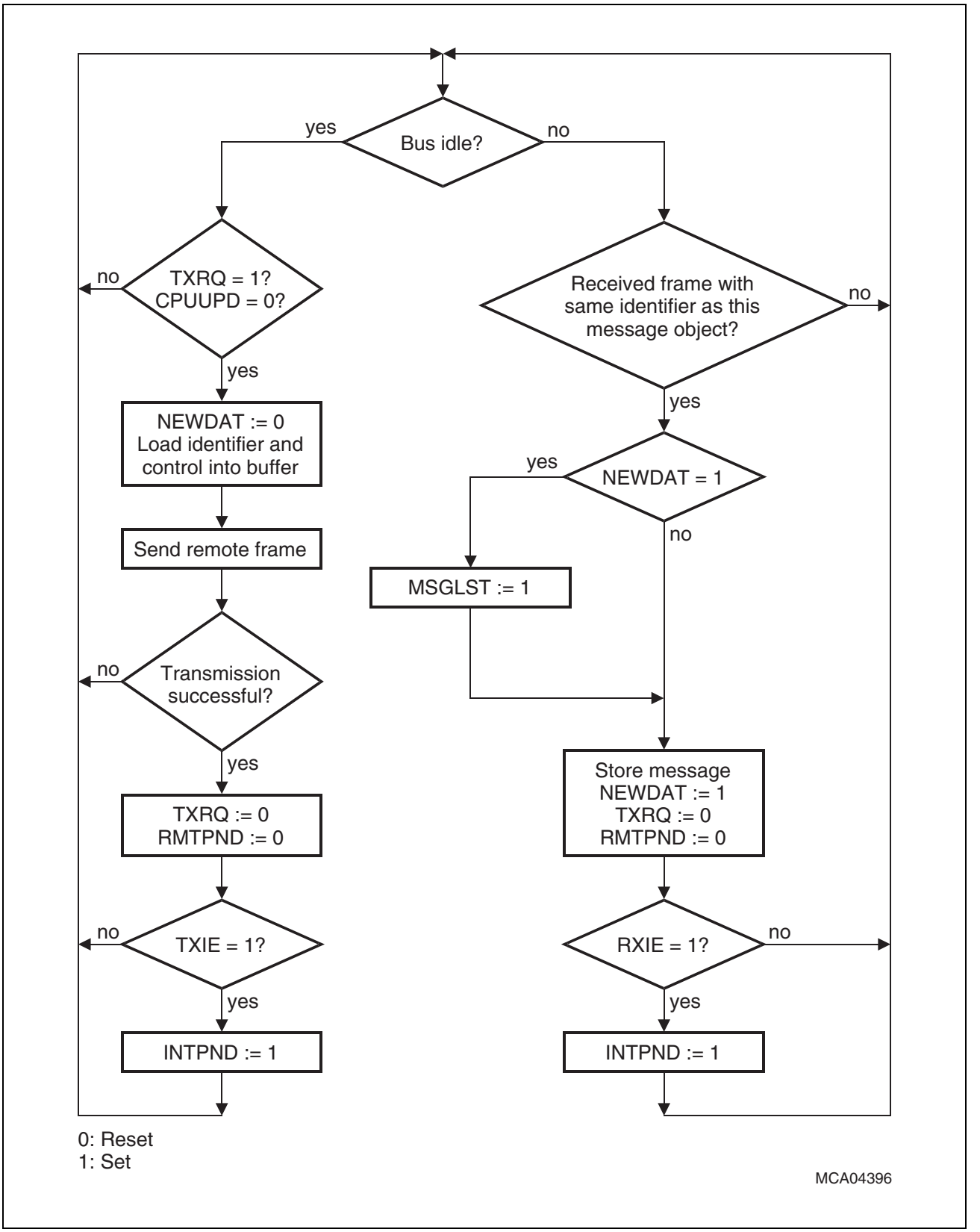


Figure 19-7 CAN Controller Handling of Receive Objects (DIR = ‘0’)

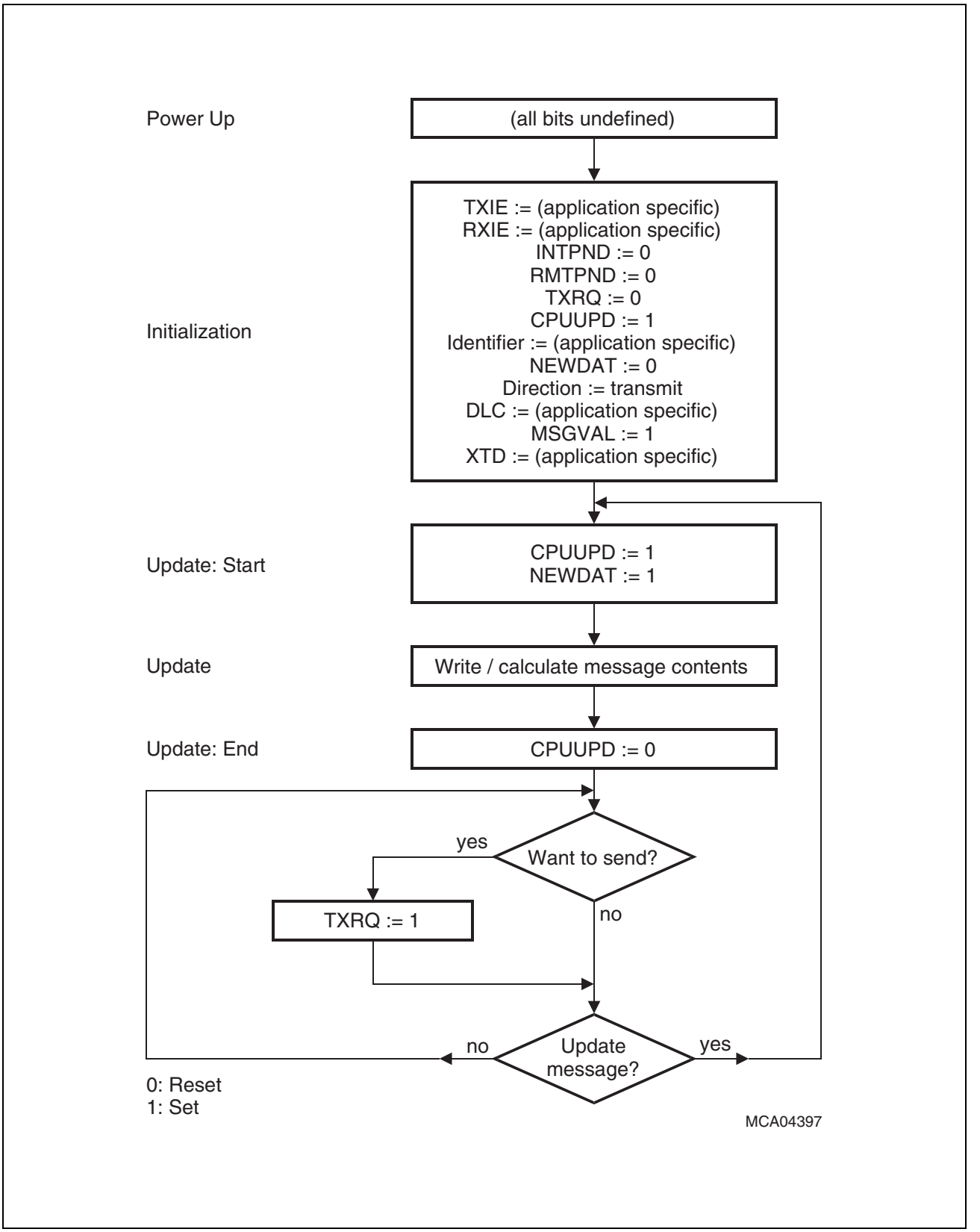


Figure 19-8 CPU Handling of Transmit Objects (DIR = ‘1’)

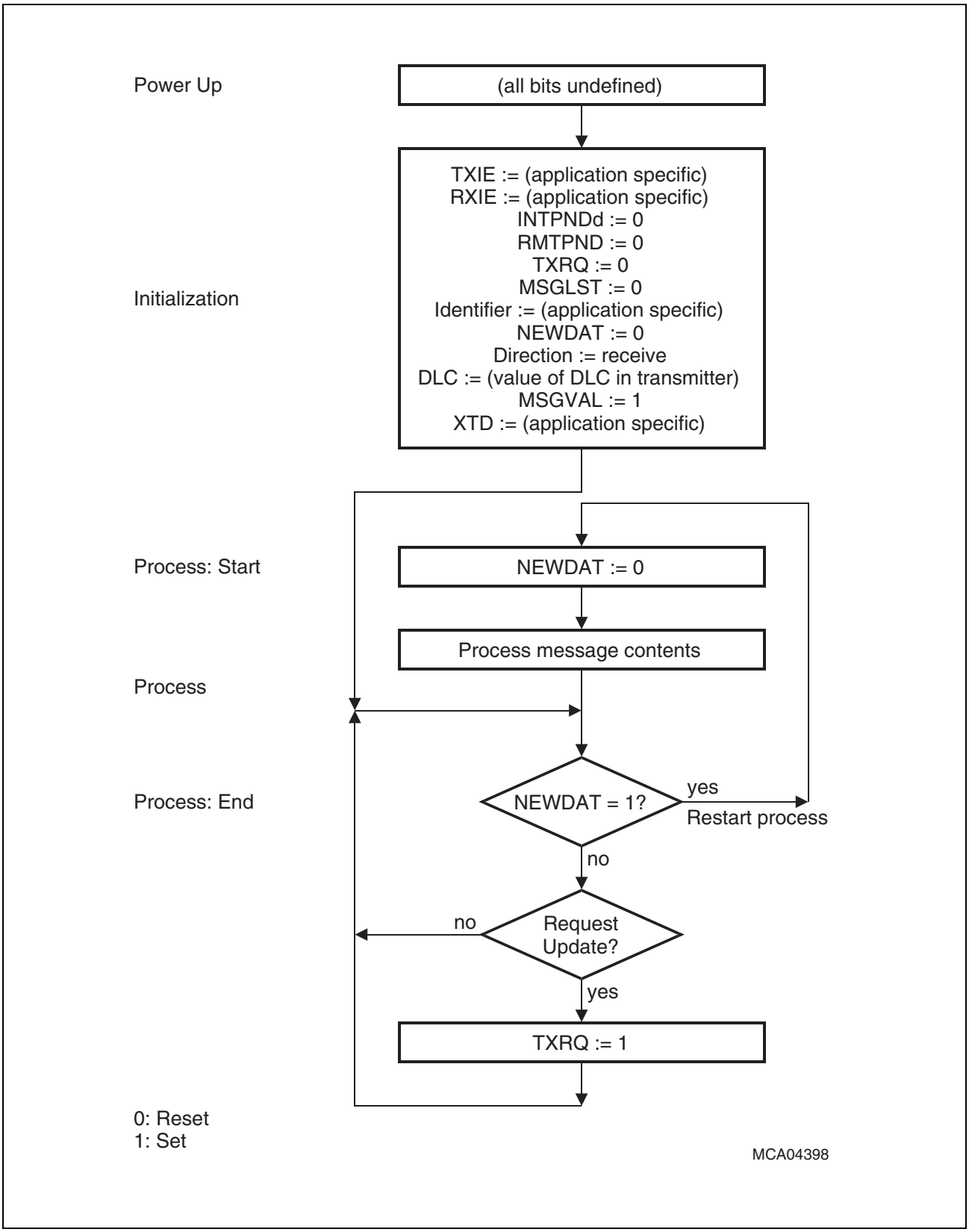


Figure 19-9 CPU Handling of Receive Objects (DIR = ‘0’)

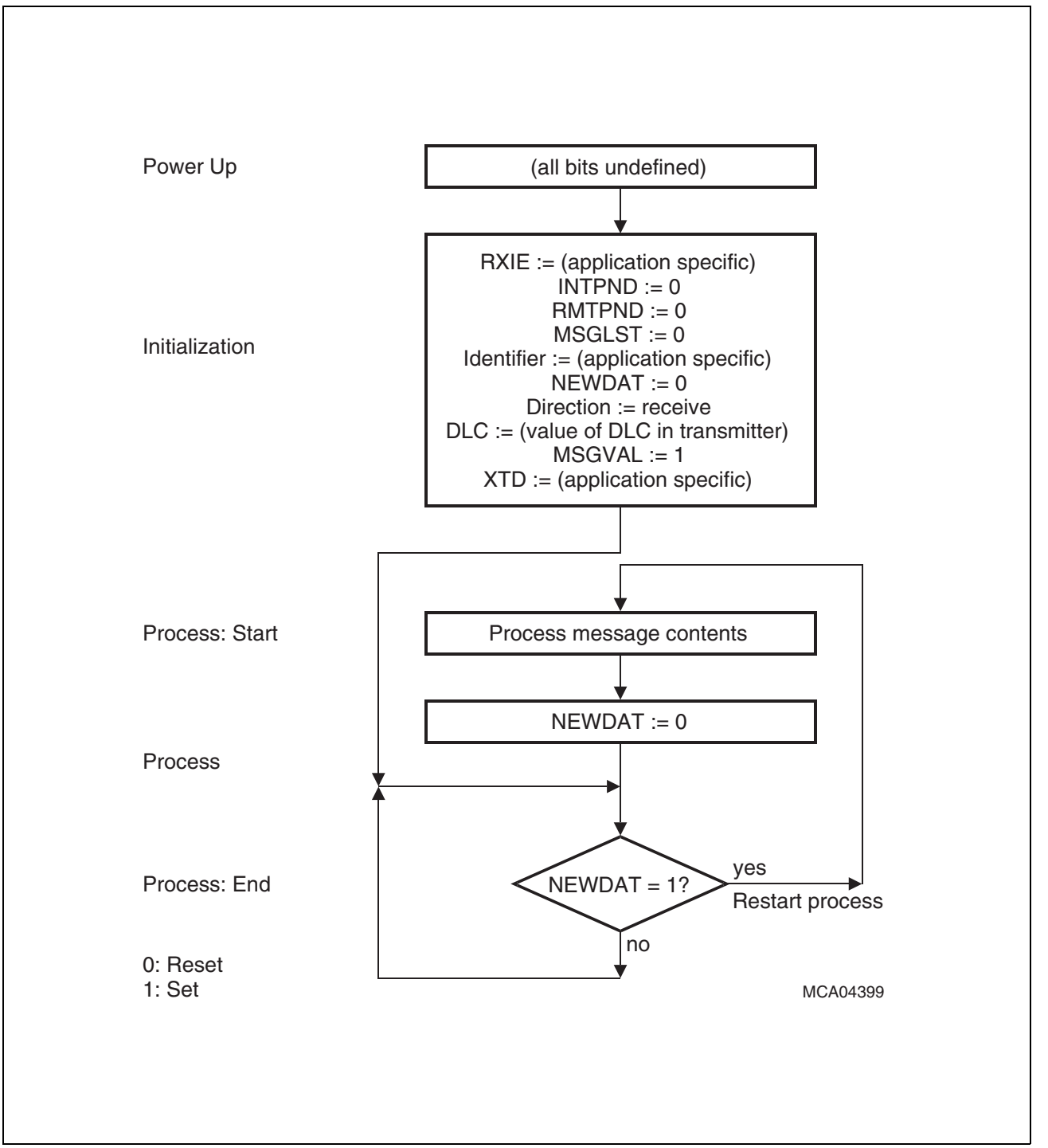


Figure 19-10 CPU Handling of the Last Message Object

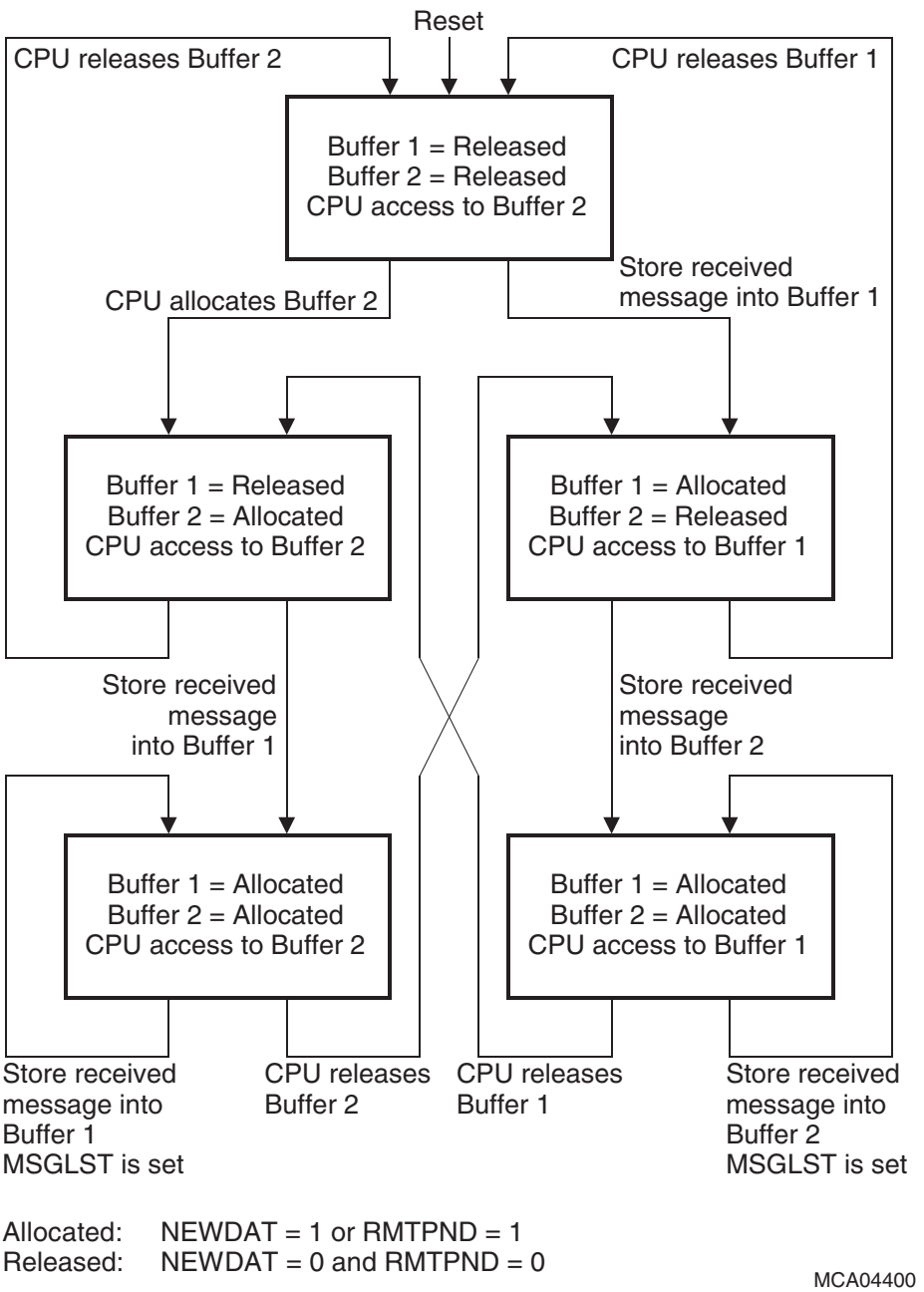


Figure 19-11 Handling of the Last Message Object's Alternating Buffer

19.4 Controlling the CAN Module

The CAN module is controlled by the C164CI via hardware signals (e.g. reset) and via register accesses executed by software.

Accessing the On-Chip CAN Module

The CAN module is implemented as an X-Peripheral and is therefore accessed like an external memory or peripheral. This means that the registers of the CAN module can be read and written using 16-bit or 8-bit direct or indirect MEM addressing modes. Bit handling is not supported via the XBUS. Since the XBUS, to which the CAN module is connected, also represents the external bus, CAN accesses follow the same rules and procedures as accesses to the external bus. CAN accesses cannot be executed in parallel to external instruction fetches or data read/writes, but are arbitrated and inserted into the external bus access stream.

Accesses to the CAN module use demultiplexed addresses, a 16-bit data bus (byte accesses are possible), two waitstates, and no tristate waitstate.

The CAN address area starts at 00'EF00$_H$ and covers 256 Bytes. This area is decoded internally, so none of the programmable address windows must be sacrificed in order to access the on-chip CAN module.

The advantage of locating the CAN address area in segment 0 is that the CAN module is accessible via data page 3. This is the 'system' data page, accessed usually through the 'system' data page pointer DPP3. In this way, internal addresses (such like SFRs, internal RAM, and the CAN registers), are all located within the same data page and form a contiguous address space.

Power Down Mode

If the C164CI enters Power Down mode, the XCLK signal will be turned off. This stops the operation of the CAN module; thus, any message transfer is interrupted. To ensure that the CAN controller is not stopped while sending a dominant level ('0') on the CAN bus, the CPU should set bit INIT in the Control Register prior to entering Power Down mode. The CPU can determine if a transmission is in progress by reading bits TXRQ and NEWDAT in the message objects and bit TXOK in the Control Register. After returning from Power Down mode via hardware reset, the CAN module must be reconfigured.

Disabling the CAN Modules

When the CAN module is disabled by setting bit CAN1DIS in register SYSCON3 (peripheral management), no register accesses are possible. The module's logic blocks are also stopped; so no CAN bus transfers are possible. When the CAN module is re-enabled (CAN1DIS = '0') it must be reconfigured (as after return from Power Down mode).

Note: Incoming message frames can still be recognized (not received) in this case by monitoring the receive line CAN1_RXD. For this purpose, the receive line CAN1_RXD can be connected to a fast external interrupt via register EXISEL.

CAN Module Reset

The on-chip CAN module is connected to the XBUS Reset signal. This signal is activated when the C164CI's reset input is activated, when a software reset is executed, and in case of a watchdog reset. Activating the CAN module's reset line triggers a hardware reset.

This hardware reset has the following effects:

- Disconnects the CAN_TXD output from the port logic
- Clears the error counters
- Resets the busoff state
- Switches the Control Register's low byte to 01_H
- Leaves the Control Register's high byte and the Interrupt Register undefined
- Does not change other registers, including the message objects (notified as UUUU)

*Note: The first hardware reset after power-on leaves the un**changed** registers in an un**defined** state, of course.*
The value 01_H in the Control Register's low byte prepares for the module initialization.

CAN Module Activation

The CAN module is disabled after a reset. Before it can be used to receive or transmit messages, the application software must activate the CAN module.

Three actions are required for this purpose:

- **General Module Enable** globally activates the CAN module by setting bit XPEN in register SYSCON after setting the corresponding selection bit in register XPERCON.
- **Pin Assignment** selects a pair of port pins to connect the CAN module to the external transceiver. This is done via bitfield IPC in register PCIR.
- **Module Initialization** determines the functionality of the CAN module (baudrate, active objects, etc.). This is the major part of the activation and is described below.

Module Initialization

Module initialization is enabled by setting bit INIT in the control register CSR. This can be done by the CPU via software, or by the CAN controller automatically on a hardware reset, or if the EML switches to busoff state.

While INIT is set:

- All message transfer from and to the CAN bus is stopped
- The CAN transmit line CAN_TXD is “1” (recessive)
- Control bits NEWDAT and RMTPND of the last message object are reset
- Counters of the EML are left unchanged.

Additionally, setting bit CCE permits configuration changes in the Bit Timing Register.

To initialize the CAN Controller, the following actions are required:

- Configure the Bit Timing Register (CCE required)
- Set the Global Mask Registers
- Initialize each message object.

If a message object is not needed, it is sufficient to clear its message valid bit (MSGVAL), that is, to define it as not valid. Otherwise, the entire message object must be initialized.

After the initialization sequence has been completed, the CPU clears bit INIT.

Now, the BSP synchronizes itself to the data transfer on the CAN bus by waiting for the occurrence of a sequence of 11 consecutive recessive bits (i.e. Bus Idle) before it can take part in bus activities and start message transfers.

Initialization of the message objects is independent of the state of bit INIT and can be done on the fly. The message objects should all be configured to particular identifiers or set to “not valid” before the BSP starts the message transfer, however.

To change the configuration of a message object during normal operation, the CPU first clears bit MSGVAL, which defines it as not valid. When the configuration is completed, MSGVAL is set again.

Busoff Recovery Sequence

If the device goes *busoff*, it will set bit BOFF and bit INIT of its own accord, stopping all bus activities. For the CAN module to take part in the CAN bus activities again, the bus-off recovery sequence must be started by clearing the bit INIT (via software). After INIT has been cleared, the module will then wait for 129 occurrences of *Bus idle* before resuming normal operation.

At the end of the *busoff* recovery sequence, the Error Management Counters will be reset. This will automatically clear bits BOFF and EWRN.

During the waiting time after the resetting of INIT each time a sequence of 11 recessive bits has been monitored, a **Bit0Error** code is written to the Control Register, enabling the CPU to determine whether the CAN bus is stuck at dominant or continuously disturbed and to monitor the progress of the busoff recovery sequence.

Note: An interrupt can be generated when entering the busoff state if bits IE and EIE are set. The corresponding interrupt code in bitfield INTID is 01_H.
The busoff recovery sequence cannot be shortened by setting or resetting INIT.

19.5 Configuration Examples for Message Objects

The two examples below represent standard applications for using CAN messages. Both examples assume that the identifier and direction have already been set up correctly.

The respective contents of the Message Control Register (MCR) are shown.

Configuration Example of a Transmission Object

This object shall be configured for transmission. It shall be transmitted automatically in response to remote frames, but no receive interrupts shall be generated for this object.

MCR (Data bytes are not written completely → CPUUPD = ‘1’)

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
0 1	**0 1**	**1 0**	**0 1**	**1 0**	**0 1**	**0 1**	**0 1**
RMTPND	TXRQ	CPUUPD	NEWDAT	MSGVAL	TXIE	RXIE	INTPND

MCR (Remote frame was received in the meantime → RMTPND = ‘1’, TXRQ = ‘1’)

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
1 0	**1 0**	**1 0**	**0 1**	**1 0**	**0 1**	**0 1**	**0 1**
RMTPND	TXRQ	CPUUPD	NEWDAT	MSGVAL	TXIE	RXIE	INTPND

After updating the message, the CPU should clear CPUUPD and set NEWDAT. The previously received remote request will then be answered.

If the CPU wants to transmit the message actively, it should also set TXRQ (otherwise TXRQ should be left unchanged).

Configuration Example of a Reception Object

This object shall be configured for reception. A receive interrupt shall be generated each time new data comes in. From time to time, the CPU sends a remote request to trigger the sending of this data from a remote node.

MCR (Message object is idle, i.e. waiting for a frame to be received)

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
0 1	**0 1**	**0 1**	**0 1**	**1 0**	**0 1**	**1 0**	**0 1**
RMTPND	TXRQ	MSGLST	NEWDAT	MSGVAL	TXIE	RXIE	INTPND

MCR (A data frame was received → NEWDAT = ‘1’, INTPND = ‘1’)

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
0 1	**0 1**	**0 1**	**1 0**	**1 0**	**0 1**	**1 0**	**1 0**
RMTPND	TXRQ	MSGLST	NEWDAT	MSGVAL	TXIE	RXIE	INTPND

To process the message, the CPU should clear INTPND and NEWDAT, process the data, and verify that NEWDAT is still clear after that. If it is not clear, the processing should be repeated.
To send a remote frame to request the data, bit TXRQ simply needs to be set. This bit will be cleared by the CAN controller after the remote frame has been sent or if the data is received before the CAN controller could transmit the remote frame.

19.6 CAN Application Interface

The on-chip CAN module of the C164CI is connected to the (external) physical layer (i.e. the CAN bus) via two signals, as shown in **Table 19-2**.

Table 19-2 CAN Interface Signals

CAN Signal	Port Pin	Function
CAN1_RXD	Controlled via C1PCIR.IPC	Receive data from the physical layer of the CAN bus 1
CAN1_TXD		Transmit data to the physical layer of the CAN bus 1

A logic low level (‘0’) is interpreted as the dominant CAN bus level, a logic high level (‘1’) is interpreted as the recessive CAN bus level.

Connection to an External Transceiver

The CAN module of the C164CI can be connected to an external CAN bus via a CAN transceiver.

Note: It is also possible to connect several CAN modules directly (on-board) without using CAN transceivers.

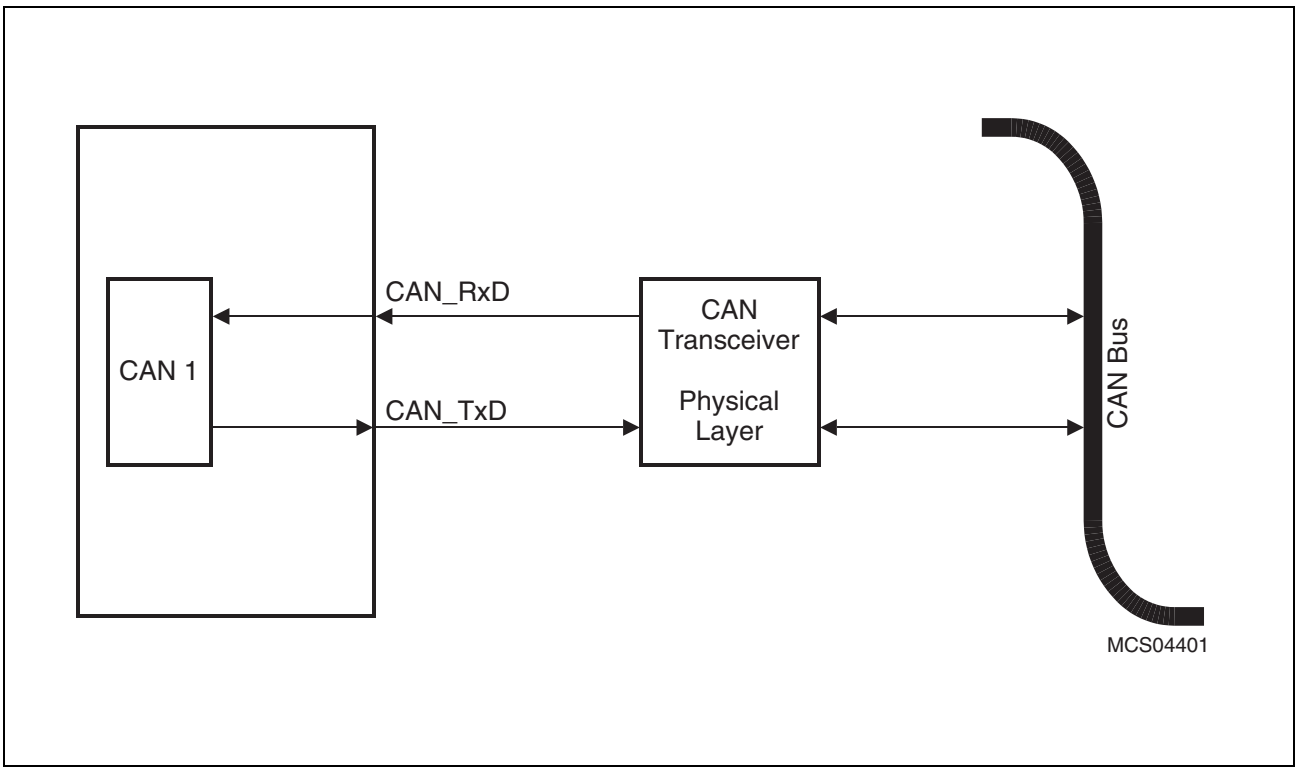


Figure 19-12 Connection to a Single CAN Bus

Port Control

The receive data line and the transmit data line of the CAN module are alternate port functions. To enable proper reception, please ensure that the respective port pin for the receive line is switched to input. The respective port driver for the transmit will automatically be switched ON.

This provides a standard pin configuration without additional software control. It also works in emulation mode where the port direction registers cannot be controlled.

The receive and transmit line of the CAN module may be assigned to several port pins of the C164CI under software control. This assignment is selected via bitfield IPC (Interface Port Connection) in register PCIR.

Table 19-3 Assignment of CAN Interface Lines to Port Pins

IPC	CAN_RXD	CAN_TXD	Notes
000	P4.5	P4.6	Compatible assignments (CAN1).[1)]
001	–	–	*Reserved*. Do not use this combination.
010	P8.0	P8.1	Port 4 available for segment address lines A21 … A16 (4 MByte external address space).
011	P8.2	P8.3	Port 4 available for segment address lines A21 … A16 (4 MByte external address space).
100	–	–	*Reserved*. Do not use this combination.
101	–	–	*Reserved*. Do not use this combination.
110	–	–	*Reserved*. Do not use this combination.
111	Idle (recessive)	Disconnected	No port assigned. Default after Reset.

1) This assignment is compatible with previous derivatives where the assignment of CAN interface lines was fixed.

The location of the CAN interface lines can now be selected via software according to the requirements of an application:

Compatible Assignment (IPC = 000_B) makes the C164CI suitable for applications with a given hardware (board layout). The CAN interface lines are connected to the port pins to which they are hardwired in previous derivatives.

Full Address Assignment (IPC = 010_B or 011_B) removes the CAN interface lines completely from Port 4. The maximum external address space available in this case is 4 MBytes.
The CAN interface lines are mapped to Port 8. Two pairs of Port 8 pins can be selected.

No Assignment (IPC = 111_B) disconnects the CAN interface lines from the port logic. This avoids undesired currents through the interface pin drivers while the C164CI is in a power saving state.
After reset the CAN interface lines are disconnected.

Note: Assigning CAN interface signals to a port pin overrides the other alternate function of the respective pin (segment address on Port 4, CAPCOM lines on Port 8).

20 System Reset

The internal system reset function provides initialization of the C164CI into a defined default state. The default state is invoked either by asserting a hardware reset signal on pin $\overline{\text{RSTIN}}$ (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 its predefined default state through an internal reset procedure. When a reset is initiated, pending internal hold states are cancelled and the current internal access cycle (if any) is completed. An external bus cycle is aborted, except for a watchdog reset (see description). Afterwards, the bus pin drivers and the IO pin drivers are switched off (tristate).

The internal reset procedure requires 516 CPU clock cycles in order to perform a complete reset sequence. This 516 cycle reset sequence is started by a watchdog timer overflow, by an SRST instruction or when the reset input signal $\overline{\text{RSTIN}}$ is latched low (hardware reset). The internal reset condition is active for at least the duration of the reset sequence and then until the $\overline{\text{RSTIN}}$ input is inactive and the PLL has locked (if the PLL is selected for the basic clock generation). When this internal reset condition is removed (reset sequence complete, $\overline{\text{RSTIN}}$ inactive, PLL locked), the reset configuration is latched from PORT0, $\overline{\text{RD}}$, and ALE (depending on the start mode). Afterwards, pins ALE, $\overline{\text{RD}}$, and $\overline{\text{WR}}$ are driven to their inactive levels.

Note: Bit ADP (which selects the Adapt mode) is latched with the rising edge of $\overline{\text{RSTIN}}$.

After the internal reset condition is removed, the microcontroller will either start program execution from external or internal memory, or it will enter boot mode.

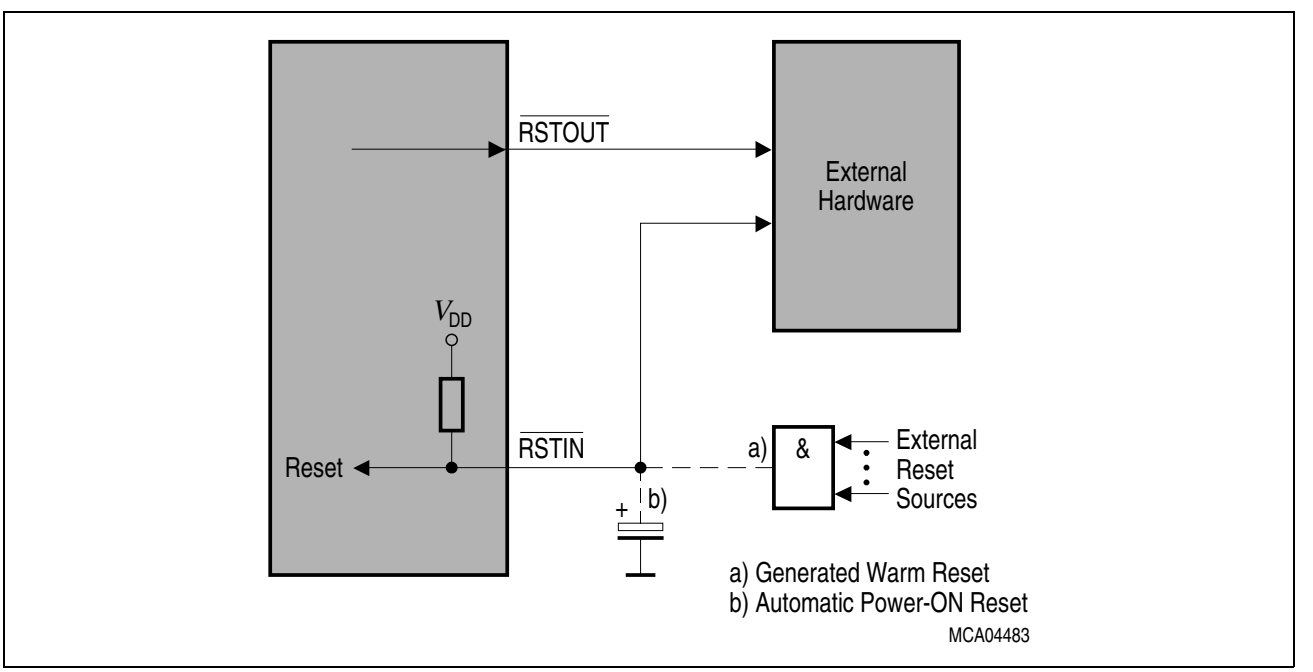


Figure 20-1 External Reset Circuitry

20.1 Reset Sources

Several external or internal sources can generate a reset for the C164CI. Software can identify the respective reset source via the reset source indication flags in register WDTCON. Generally, any reset causes the same actions on the C164CI's modules. The differences are described in the following sections.

Hardware Reset

A hardware reset is triggered when the reset input signal $\overline{\text{RSTIN}}$ is latched low. To ensure the recognition of the $\overline{\text{RSTIN}}$ signal (latching), it must be held low for at least 100 ns plus 2 CPU clock cycles (input filter plus synchronization). Shorter $\overline{\text{RSTIN}}$ pulses may also trigger a hardware reset if they coincide with the latch's sample point. The actual minimum duration for a reset pulse depends on the current CPU clock generation mode. The worst case is generating the CPU clock via the SlowDown Divider using the maximum factor while the configured basic mode uses the prescaler ($f_{CPU} = f_{OSC} / 64$ in this case).

After the reset sequence has been completed, the $\overline{\text{RSTIN}}$ input is sampled again. If the reset input signal is inactive at that time, the internal reset condition is terminated (indicated as short hardware reset, SHWR). If the reset input signal is still active at that time, the internal reset condition is prolonged until $\overline{\text{RSTIN}}$ becomes inactive (indicated as long hardware reset, LHWR).

During a hardware reset, the inputs for the reset configuration (PORT0, $\overline{\text{RD}}$, ALE) need some time to settle on the required levels, especially if the hardware reset aborts a read operation from an external peripheral. During this settling time, the configuration may intermittently be wrong. For the duration of one internal reset sequence after a reset has been recognized, the configuration latches are not transparent; thus the (new) configuration becomes valid earliest after the completion of one reset sequence. This usually covers the required settling time.

When the basic clock is generated by the PLL, the internal reset condition is automatically extended until the on-chip PLL has locked.

The input $\overline{\text{RSTIN}}$ provides an internal pull-up device equalling a resistor of 50 kΩ to 250 kΩ (the minimum reset time must be determined by the lowest value). Simply connecting an external capacitor is sufficient for an automatic power-on reset (see *b)* in **Figure 20-1**). $\overline{\text{RSTIN}}$ may also be connected to the output of other logic gates (see *a)* in **Figure 20-1**). See also **"Bidirectional Reset" on Page 22-4** in this case.

Note: A power-on reset requires an active time of two reset sequences (1036 CPU clock cycles) after a stable clock signal is available (about 10 … 50 ms, depending on the oscillator frequency, to allow the on-chip oscillator to stabilize).

Software Reset

The reset sequence can be triggered at any time via the protected instruction SRST (Software Reset). This instruction can be executed deliberately within a program, such as to exit bootstrap loader mode, or upon a hardware trap that reveals a system failure.

Note: A software reset only latches the configuration of the bus interface (SALSEL, CSSEL, WRC, BUSTYP) from PORT0 in case of an external reset.
If bidirectional reset is enabled, a software reset is executed like a long hardware reset.

Watchdog Timer Reset

If the Watchdog Timer (WDT) is not disabled during the initialization or serviced regularly during program execution, it will overflow and trigger the reset sequence. Other than after hardware and software reset, the watchdog reset completes a running external bus cycle. Then the internal reset sequence is started.

Note: A watchdog reset only latches the configuration of the bus interface (SALSEL, CSSEL, WRC, BUSTYP) from PORT0 in case of an external reset.
If bidirectional reset is enabled, a watchdog timer reset is executed like a long hardware reset.
The watchdog reset cannot occur while the C164CI is in bootstrap loader mode!

Bidirectional Reset

In a special bidirectional reset mode, the C164CI's line $\overline{\text{RSTIN}}$ (normally an input) may be driven active by the chip logic. This is useful, for instance, to support external equipment required for startup (such as flash memory).

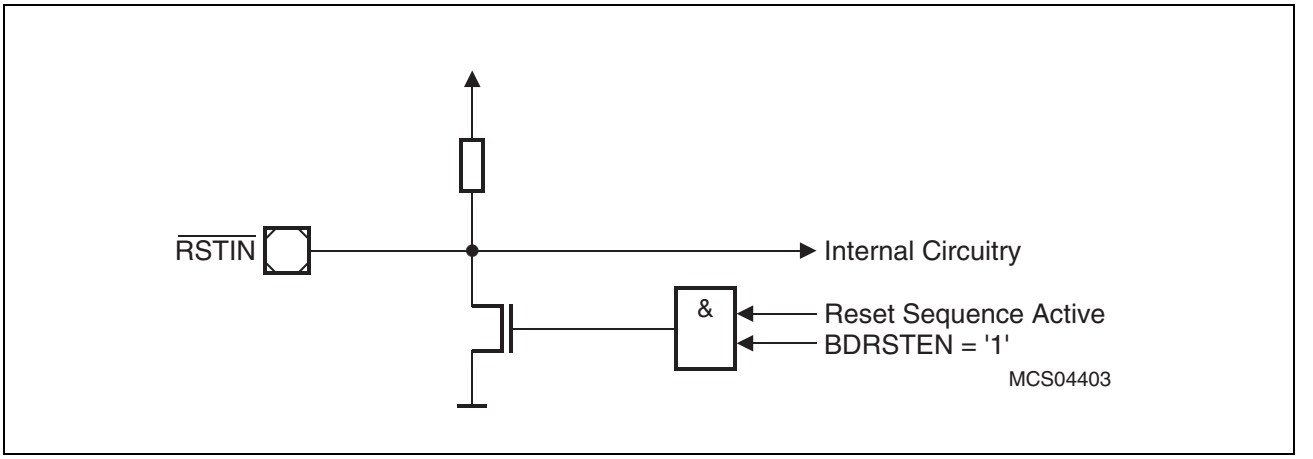


Figure 20-2 Bidirectional Reset Operation

Bidirectional reset reflects internal reset sources (software, watchdog) to the $\overline{\text{RSTIN}}$ pin and converts short hardware reset pulses to a minimum duration of the internal reset sequence. Bidirectional reset is enabled by setting bit BDRSTEN in register SYSCON; it changes $\overline{\text{RSTIN}}$ from a pure input to an open drain IO line. When an internal reset is triggered by the SRST instruction, by a watchdog timer overflow, or by a low level applied to the $\overline{\text{RSTIN}}$ line, an internal driver pulls it low for the duration of the internal reset sequence. After that, it is released and is then controlled solely by the external circuitry.

The bidirectional reset function is useful for applications in which external devices require a defined reset signal but which cannot be connected to the C164CI's $\overline{\text{RSTOUT}}$ signal; for example, an external Flash memory which must come out of reset and deliver code well before $\overline{\text{RSTOUT}}$ can be deactivated via EINIT.

The following behavior differences must be observed when using the bidirectional reset feature in an application:

- Bit BDRSTEN in register SYSCON cannot be changed after EINIT.
- Bit BDRSTEN is cleared after a reset.
- The reset indication flags always indicate a long hardware reset.
- The PORT0 configuration is treated as on a hardware reset. Especially the bootstrap loader may be activated when P0L.4 or $\overline{\text{RD}}$ is low.
- Pin $\overline{\text{RSTIN}}$ may only be connected to external reset devices with open drain output driver.
- A short hardware reset is extended to the duration of the internal reset sequence.

20.2 Status After Reset

Most units of the C164CI enter a well-defined default status after a reset is completed. This ensures repeatable start conditions and avoids spurious activities after reset.

Watchdog Timer Operation after Reset

The watchdog timer starts running after the internal reset is complete. It will be clocked with the internal system clock divided by 2 (f_{CPU} / 2), and its default reload value is 00_H. Thus a watchdog timer overflow will occur 131,072 CPU clock cycles (2×2^{16}) after completion of the internal reset, 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 WDTCON will be set to '1'. This indicates the cause of the internal reset to the software initialization routine. WDTR is reset to '0' by an external hardware reset, by servicing the watchdog timer or after EINIT. After the internal reset is complete, the operation of the watchdog timer can be disabled by the DISWDT (Disable Watchdog Timer) instruction. This instruction has been implemented as a protected instruction. For further security, its execution is enabled only in the time period after a reset until either the SRVWDT (Service Watchdog Timer) or the EINIT instruction has been executed. Thereafter, the DISWDT instruction will have no effect.

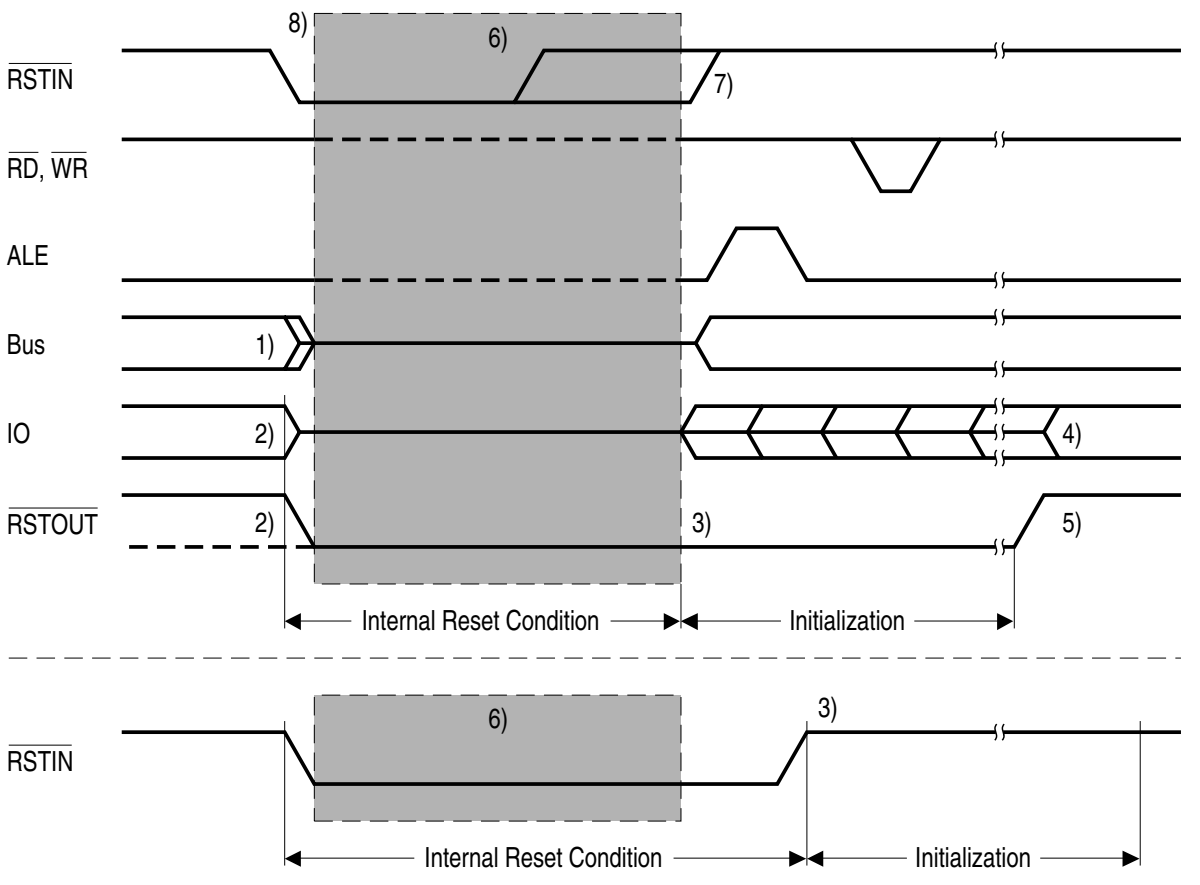
Reset Values for the C164CI Registers

During the reset sequence, the registers of the C164CI 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.

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: $FC00_H$
WDTCON: $00XX_H$ (value depends on the reset source)
S0RBUF: XX_H (undefined)
SSCRB: $XXXX_H$ (undefined)
SYSCON: $0XX0_H$ (set according to reset configuration)
BUSCON0: $0XX0_H$ (set according to reset configuration)
RP0H: XX_H (reset levels of P0H)
ONES: $FFFF_H$ (fixed value)

The C164CI's Pins after Reset

After the reset sequence, the various groups of pins of the C164CI are activated in different ways depending on their function. Bus and control signals are activated immediately after the reset sequence according to the configuration latched from PORT0, so either external accesses can take place or the external control signals will be inactive. The general purpose IO pins remain in input mode (high impedance) until reprogrammed via software (see **Figure 20-3**). The $\overline{\text{RSTOUT}}$ pin remains active (low) until the end of the initialization routine (see description).



When the internal reset condition is extended by $\overline{\text{RSTIN}}$, the activation of the output signals is delayed until the end of the internal reset condition.

1) Current bus cycle is completed or aborted.
2) Switches asynchronously with $\overline{\text{RSTIN}}$, synchronously upon software or watchdog reset.
3) The reset condition ends here. The microcontroller starts program execution.
4) Activation of the IO pins is controlled by software.
5) Execution of the EINIT instruction.
6) The shaded area designates the internal reset sequence, which starts after synchronization of $\overline{\text{RSTIN}}$.
7) A short hardware reset is extended until the end of the reset sequence in Bidirectional reset mode.
8) A software or WDT reset activates the $\overline{\text{RSTIN}}$ line in Bidirectional reset mode.

MCS02258

Figure 20-3 Reset Input and Output Signals

Ports and External Bus Configuration during Reset

During the internal reset sequence, all port pins of the C164CI 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 C164CI and external devices will not try to drive the same pin to different levels. Pin ALE is held low through an internal pull-down, and pins $\overline{RD}$ and $\overline{WR}$ are held high through internal pull-ups. Also, the pins which can be configured for $\overline{CS}$ output will be pulled high.

The registers SYSCON and BUSCON0 are initialized according to the configuration selected via PORT0.

When an external start is selected (pin $\overline{EA}$ = '0'):

- Bus Type field (BTYP) in register BUSCON0 is initialized according to P0L.7 and P0L.6
- Bit BUSACT0 in register BUSCON0 is set to '1'
- Bit ALECTL0 in register BUSCON0 is set to '1'
- Bit ROMEN in register SYSCON will be cleared to '0'
- Bit BYTDIS in register SYSCON is set according to the data bus width (set if 8-bit)
- Bit WRCFG in register SYSCON is set according to pin P0H.0 (WRC)

When an internal start is selected (pin $\overline{EA}$ = '1'):

- Register BUSCON0 is initialized to $00C0_H$
- Bit ROMEN in register SYSCON will be set to '1'
- Bit BYTDIS in register SYSCON is set, i.e. $\overline{BHE}$/$\overline{WRH}$ is disabled
- Bit WRCFG in register SYSCON is set according to pin P0H.0 (WRC)

The other bits of register BUSCON0, and the other BUSCON registers are cleared. This default initialization selects the slowest possible external accesses using the configured bus type.

When the internal reset is complete, the configuration of PORT0, PORT1, Port 4, Port 6, and of the $\overline{BHE}$ signal (High Byte Enable, alternate function of P3.12) depends on the bus type selected during reset. If any of the external bus modes was selected during reset, PORT0 will operate in the selected bus mode. Port 4 will output the selected number of segment address lines (all zero after reset). Port 6 will drive the selected number of $\overline{CS}$ lines ($\overline{CS0}$ will be '0', while the other active $\overline{CS}$ lines will be '1'). If no memory accesses above 64 K are to be performed, segmentation may be disabled.

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.

Reset Output Pin

The $\overline{\text{RSTOUT}}$ pin is dedicated to the generation of a reset signal for the system components other than the controller. $\overline{\text{RSTOUT}}$ will be driven active (low) at the begin of any reset sequence (triggered by hardware, the SRST instruction, or a watchdog timer overflow). $\overline{\text{RSTOUT}}$ stays active (low) beyond the end of the internal reset sequence until the protected EINIT (End of Initialization) instruction is executed (see **Figure 20-3**). This allows the complete configuration of the controller including its on-chip peripheral units before releasing the reset signal for the external peripherals of the system.

Note: $\overline{\text{RSTOUT}}$ will float during emulation mode or adapt mode.

The Internal RAM after Reset

The contents of the internal RAM are not affected by a system reset. However, after a power-on reset, the contents of the internal RAM are undefined. This implies that the GPRs (R15 … R0) and the PEC source and destination pointers (SRCP7 … SRCP0, DSTP7 … DSTP0) which are mapped into the internal RAM are also unchanged after a warm reset, software reset, or watchdog reset, but are undefined after a power-on reset.

The Extension RAM (XRAM) after Reset

The contents of the on-chip extension RAM are not affected by a system reset. However, after a power-on reset, the contents of the XRAM are undefined.

Operation after Reset

After the internal reset condition is removed, the C164CI fetches the first instruction from the program memory (location $00'0000_H$ for a standard start). 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 C164CI does not fetch instructions from the program memory.
The standard bootstrap loader expects data via serial interface ASC0.

20.3 Application-Specific Initialization Routine

After a reset, the modules of the C164CI must be initialized to enable their operation on a given application. This initialization depends on the task to be performed by the C164CI in that application and on some system properties such as operating frequency, external circuitry connected, etc.

Typically, the following initializations should be done before the C164CI is prepared to run the actual application software:

Memory Areas

The external bus interface can be reconfigured after an external reset because register BUSCON0 is initialized to the slowest possible bus cycle configuration. The programmable address windows can be enabled in order to adapt the bus cycle characteristics to various memory areas or peripherals. Also, after a single-chip mode reset, the external bus interface can be enabled and configured.

The internal program memory (if available) can be enabled and mapped after an external reset in order to use the on-chip resources. After a single-chip mode reset, the internal program memory can be remapped or disabled in order to utilize external memory (partially or completely).

Programmable program memory can be programmed, for instance, with data received over a serial link.

Note: Initial Flash or OTP programming will rather be done in bootstrap loader mode.

System Stack

The default setup for the system stack (size, stackpointer, 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, where the system stack selected by the SP grows downwards from 00'FBFE$_H$.

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

Register Bank

The location of a 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 00'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 (IRAM/XRAM) 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 locations must receive pointers to the respective exception handlers. 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.

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.

Ports

Generally, all ports of the C164CI are switched to input after reset. Some pins may be automatically controlled, such as bus interface pins for an external start, TxD in Boot mode, etc. 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

After reset the C164CI'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 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.

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,
- Disables write accesses to register SYSCON (all configurations regarding register SYSCON (enable CLKOUT, stacksize, etc.) must be selected before the execution of EINIT),
- Disables write accesses to registers SYSCON2 and SYSCON3
  (further write accesses to SYSCON2 and SYSCON3 can be executed only using a special unlock mechanism),
- Clears the reset source detection bits in register WDTCON,
- Causes the $\overline{\text{RSTOUT}}$ pin to go high
  (this signal can be used to indicate the end of the initialization routine and the proper operation of the microcontroller to external hardware).

20.4 System Startup Configuration

Although most of the programmable features of the C164CI 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 (for example, internal or external start selected via $\overline{EA}$).

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/pull-down 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. Many configurations can, therefore, be coded with a minimum of external circuitry.

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.

20.4.1 System Startup Configuration upon an External Reset

For an external reset ($\overline{EA}$ = ‘0’), the startup configuration uses the pins of PORT0 and pin $\overline{RD}$. The value on the upper byte of PORT0 (P0H) is latched into register RP0H upon reset, the value on the lower byte (P0L) directly influences the BUSCON0 register (bus mode) or the internal control logic of the C164CI.

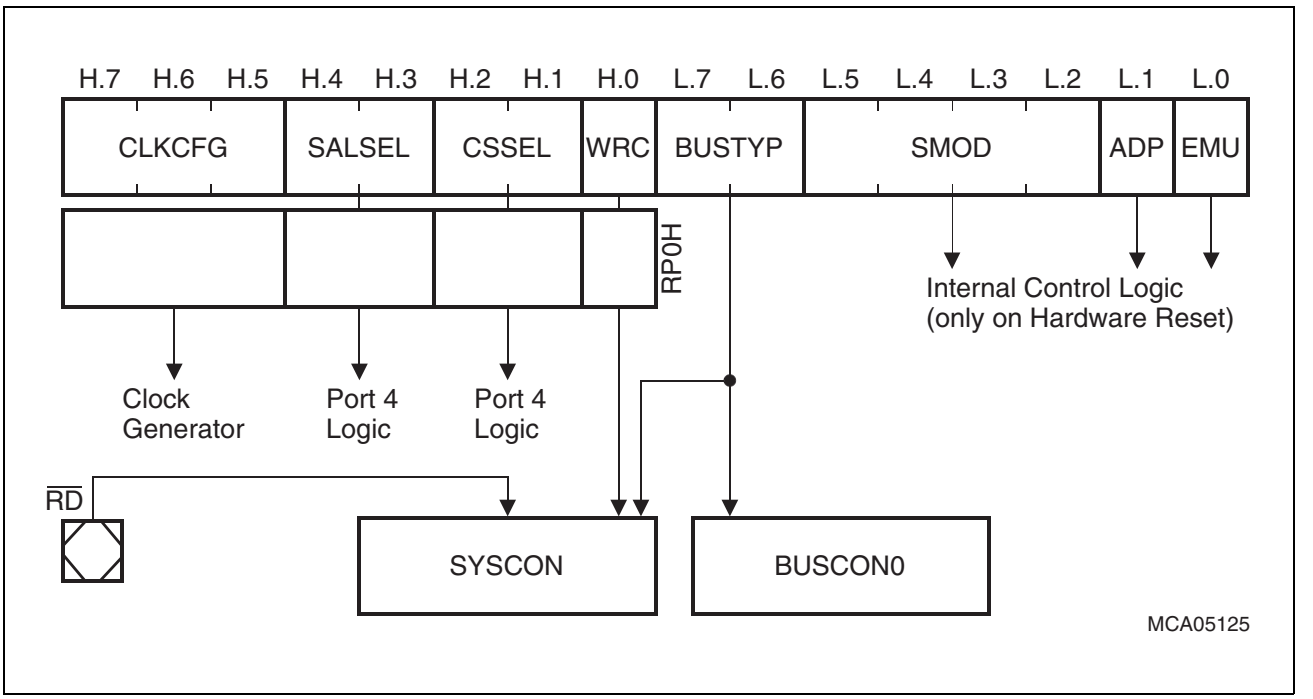


Figure 20-4 PORT0 Configuration during Reset

The pins which control operation of the internal control logic, the clock configuration, and the reserved pins are evaluated only during a hardware triggered reset sequence. The pins which influence the configuration of the C164CI are evaluated during any reset sequence, including software and watchdog timer triggered resets.

The configuration via P0H is latched in register RP0H for subsequent evaluation by software. Register RP0H is described in **Chapter 9**.

The following descriptions refer to the various selections available for reset configuration. The default modes refer to pins at high level without external pull-down devices connected.
Please also consider the note above.

Emulation Mode

Pin P0L.0 (EMU) selects the Emulation Mode when latched low at the end of reset. Because this mode is used for special emulation and testing purposes and is of minor use for standard C164CI applications, P0L.0 should be held high.

Emulation Mode provides access to integrated XBUS peripherals via the external bus interface pins (direction reversed) of the C164CI. The CPU and the generic peripherals are disabled, all modules connected via the XBUS are active. Ensure that all required input pins are driven accordingly (see **Table 20-1**) and no driver conflicts exist on the respective output pins.

The other pins retain their original function or are unused. Unused pins are switched to input and should be pulled to a stable level to avoid switching noise.

Table 20-1 Emulation Mode Summary

Pin(s)	Function	Notes
PORT0	Data input/output	–
PORT1	Address input	–
P3.12	$\overline{\text{XBHE}}$	Must be driven externally, can be kept low
$\overline{\text{RD}}$, $\overline{\text{WR}}$	Control signal input	–
ALE	Unused input	Hold LOW
$\overline{\text{RSTOUT}}$	Reset input	Drive externally for an XBUS peripheral reset
$\overline{\text{RSTIN}}$	Reset input	Standard reset for complete device
P3.9	$\overline{\text{CS}}_{\text{CAN}}$ input	Enables module CAN1 in emulation mode
P3.10	$\overline{\text{CS}}_{\text{XRAM}}$ input	Enables module XRAM in emulation mode
P3.15	CLKOUT	Automatically enabled
Port 8	XBUS peripheral interrupt output	P8.0: CAN1 P8.1: not used, always High P8.2: not used, always High P8.3: PLL

Default: Emulation Mode is off.

Note: In emulation mode pin P0.15 (P0H.7) is inverted, i.e. the configuration '111' would select direct drive in emulation mode.
Emulation mode can only be activated upon an external reset ($\overline{EA}$ = '0'). Pin P0L.0 is not evaluated upon a single-chip reset ($\overline{EA}$ = '1').

Adapt Mode

Pin P0L.1 (ADP) selects the Adapt Mode when latched low at the end of reset. In this mode, the C164CI goes into a passive state similar to its state during reset. The pins of the C164CI float to tristate or are deactivated via internal pull-up/pull-down devices, as described for the reset state. Additionally, the $\overline{\text{RSTOUT}}$ pin floats to tristate rather than being driven low. The on-chip oscillator and the realtime clock are disabled.

This mode allows a C164CI mounted to a board to be virtually switched off. This enables an emulator to control the board's circuitry even though the original C164CI remains in place. The original C164CI may resume control of the board after a reset sequence with P0L.1 high. Please note that adapt mode overrides any other configuration via PORT0.

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 ($\overline{\text{EA}}$ = '0'). Pin P0L.1 is not evaluated upon a single-chip reset ($\overline{\text{EA}}$ = '1').

Special Operation Modes

Pins P0L.5 to P0L.2 (SMOD) select special operation modes of the C164CI during reset (see **Table 20-2**). Make sure to select only valid configurations to ensure proper operation of the C164CI.

Table 20-2 Definition of Special Modes for Reset Configuration

P0.5-2 (P0L.5-2)	Special Mode	Notes
1 1 1 1	Normal Start	Default configuration. Begin of execution as defined via pin $\overline{EA}$.
1 1 1 0	CPU Host Mode (CHM)	Programming mode for OTP memory via the C164CI's CPU.
1 1 0 1	Reserved	Do not select this configuration!
1 1 0 0	Reserved	Do not select this configuration!
1 0 1 1	Standard Bootstrap Loader	Load an initial boot routine of 32 Bytes via interface ASC0.
1 0 1 0	Bootstrap Loader + CPU Host Mode	Serial programming of OTP memory via ASC0 using the bootstrap loader.
1 0 0 1	Alternate Boot	Operation not yet defined. Do not use!
1 0 0 0	Reserved	Do not select this configuration!
0 1 1 1	No emulation mode: Alternate Start	Operation not yet defined. Do not use!
	Emulation mode: External Host Mode (EHM)	Programming mode for OTP memory via external host.
0 1 1 0	Reserved	Do not select this configuration!
0 1 0 1	Reserved	Do not select this configuration!
0 1 0 0	Reserved	Do not select this configuration!
0 0 X X	Reserved	Do not select this configuration!

The On-Chip Bootstrap Loader allows the start code to be moved into the internal RAM of the C164CI via the serial interface ASC0. The C164CI will remain in bootstrap loader mode until a hardware reset deselects BSL mode or until a software reset.

Default: The C164CI starts fetching code from location $00'0000_H$, the bootstrap loader is off.

External Bus Type

Pins P0L.7 and P0L.6 (BUSTYP) select the external bus type during reset, if an external start is selected via pin $\overline{EA}$. This allows configuration of the external bus interface of the C164CI even for the first code fetch after reset. The two bits are copied into bit field BTYP of register BUSCON0. P0L.7 controls the data bus width, while P0L.6 controls the address output (multiplexed or demultiplexed). This bit field may be changed via software after reset, if required.

Table 20-3 Configuration of External Bus Type

P0L.7-6 (BTYP) Encoding	External Data Bus Width	External Address Bus Mode
0 0	8-bit Data	Demultiplexed Addresses
0 1	8-bit Data	Multiplexed Addresses
1 0	16-bit Data	Demultiplexed Addresses
1 1	16-bit Data	Multiplexed Addresses

PORT0 and PORT1 are automatically switched to the selected bus mode. In multiplexed bus modes, PORT0 drives both the 16-bit intra-segment address and the output data, while PORT1 remains in high impedance state as long as no demultiplexed bus is selected via one of the BUSCON registers. In demultiplexed bus modes, PORT1 drives the 16-bit intra-segment address, while PORT0 or P0L (according to the selected data bus width) drives the output data.
For a 16-bit data bus, $\overline{BHE}$ is automatically enabled, for an 8-bit data bus, $\overline{BHE}$ is disabled via bit BYTDIS in register SYSCON.

Default: 16-bit data bus with multiplexed addresses.

Note: If an internal start is selected via pin $\overline{EA}$, these two pins are disregarded and bit field BTYP of register BUSCON0 is cleared.

Write Configuration

Pin P0H.0 (WRC) selects the initial operation of the control pins $\overline{WR}$ and $\overline{BHE}$ during reset. When high, this pin selects the standard function, i.e. $\overline{WR}$ control and $\overline{BHE}$. When low, it selects the alternate configuration, i.e. $\overline{WRH}$ and $\overline{WRL}$. Thus, even the first access after a reset can go to a memory controlled via $\overline{WRH}$ and $\overline{WRL}$. This bit is latched in register RP0H and its inverted value is copied into bit WRCFG in register SYSCON.

Default: Standard function ($\overline{WR}$ control and $\overline{BHE}$).

Chip Select Lines

Pins P0H.2 and P0H.1 (CSSEL) define the number of active chip select signals during reset. This allows selection of which Port 4 pins drive external $\overline{CS}$ signals and which are used for general purpose IO. The two bits are latched in register RP0H.

Table 20-4 Configuration of Chip Select Lines

P0H.2-1 (CSSEL)	Chip Select Lines	Note
1 1	Four: $\overline{CS3}$ … $\overline{CS0}$	Default without pull-downs
1 0	None	–
0 1	Two: $\overline{CS1}$ … $\overline{CS0}$	–
0 0	Three: $\overline{CS2}$ … $\overline{CS0}$	–

Default: All 4 chip select lines active ($\overline{CS3}$ … $\overline{CS0}$).

*Note: The selected number of $\overline{CS}$ signals can be changed via software after reset (see **Section 20.4.2**).*

Segment Address Lines

Pins P0H.4 and P0H.3 (SALSEL) define the number of active segment address lines during reset. This allows selection of which Port 4 pins drive address lines and which are used for general purpose IO. The two bits are latched in register RP0H. Depending on the system architecture, the required address space is chosen and accessible right from the start; so, the initialization routine can directly access all locations without prior programming. The required Port 4 pins are automatically switched to address output mode.

Table 20-5 Configuration of Segment Address Lines

P0H.4-3 (SALSEL)	Segment Address Lines	Directly Accessible A. Space
1 1	Two: A17 … A16	256 KByte (Default without pull-downs)
1 0	Six: A21 … A16	4 MByte (Maximum)
0 1	None	64 KByte (Minimum)
0 0	Four: A19 … A16	1 MByte

Even if not all segment address lines are enabled on Port 4, the C164CI internally uses its complete 24-bit addressing mechanism. This allows restriction of the width of the effective address bus, while still deriving $\overline{CS}$ signals from the complete addresses.

Default: 2-bit segment address (A17 … A16) allowing access to 256 KBytes.

*Note: The selected number of segment address lines can be changed via software after reset (see **Section 20.4.2**).*

Clock Generation Control

Pins P0H.7, P0H.6 and P0H.5 (CLKCFG) select the basic clock generation mode during reset. The oscillator clock either directly feeds the CPU and peripherals (direct drive), is divided by 2 or is fed to the on-chip PLL which then provides the CPU clock signal (selectable multiple of the oscillator frequency, i.e. the input frequency). These bits are latched in register RP0H.

Table 20-6 C164CI Clock Generation Modes

(P0H.7-5) (CLKCFG)	CPU Frequency $f_{CPU} = f_{OSC} \times F$	External Clock Input Range[1)]	Notes
1 1 1	$f_{OSC} \times 4$	2.5 to 6.25 MHz	Default configuration
1 1 0	$f_{OSC} \times 3$	3.33 to 8.33 MHz	–
1 0 1	$f_{OSC} \times 2$	5 to 12.5 MHz	–
1 0 0	$f_{OSC} \times 5$	2 to 5 MHz	–
0 1 1	$f_{OSC} \times 1$	1 to 25 MHz	Direct drive[2)]
0 1 0	$f_{OSC} \times 1.5$	6.66 to 16.6 MHz	–
0 0 1	$f_{OSC} / 2$	2 to 50 MHz	CPU clock via prescaler
0 0 0	$f_{OSC} \times 2.5$	4 to 10 MHz	–

1) The external clock input range refers to a CPU clock range of 10 … 25 MHz.

2) The maximum frequency depends on the duty cycle of the external clock signal.
In emulation mode pin P0.15 (P0H.7) is inverted, i.e. the configuration ‘111’ would select direct drive in emulation mode.

Default: On-chip PLL is active with a factor of 1:4.

Watch the different requirements for frequency and duty cycle of the oscillator input clock for the possible selections.

Oscillator Watchdog Control

The on-chip oscillator watchdog (OWD) may be disabled via hardware by (externally) pulling the $\overline{RD}$ line low upon a reset, similar to the standard reset configuration via PORT0. At the end of an external reset, bit OWDDIS in register SYSCON reflects the inverted level of pin $\overline{RD}$ at that time. The software may again enable the oscillator watchdog by clearing bit OWDDIS before the execution of EINIT.

Note: If direct drive or prescaler operation is selected as basic clock generation mode (see above) the PLL is switched off whenever bit OWDDIS is set (via software or via hardware configuration).

20.4.2 System Startup Configuration at Single-Chip Mode Reset

For a single-chip mode reset (indicated by $\overline{EA}$ = '1') the configuration via PORT0 is replaced by a fixed configuration value. In this case, PORT0 needs no external circuitry (pull-ups/pull-downs) and also the internal configuration pull-ups are not activated.
The necessary startup modes are configured via pins $\overline{RD}$ and ALE.

This fixed default configuration is activated after each Long Hardware Reset and selects a safe worst-case configuration. The initialization software can then modify these parameters and select the intended configuration for a given application. **Table 20-7** lists the respective default configuration values which are selected and the bitfields which permit software modification.

Table 20-7 Default Configuration for Single-Chip Mode Reset

Configuration Parameter	Default Values ($RP0H = XX2D_H$)	External Config.[1]	Software Access[2]
CLKCFG: Generation mode of basic clock	'001' = Prescaler operation, i.e. $f_{CPU} = f_{OSC} / 2$	P0.15-13	RSTCON.15-13
SALSEL: Number of active segment address lines	'01' = No segment address lines	P0.12-11	RSTCON.12-11
CSSEL: Number of active $\overline{CS}$ lines	'10' = No chip select lines	P0.10-9	RSTCON.10-9
WRC: Write signal encoding	RP0H.0 = '1', SYSCON.WRCFG = '0', i.e. $\overline{WR}$ and $\overline{BHE}$	P0.8	SYSCON.WRCFG
BTYP: Default bustype (BUSCON0)	BUSCON0.BTYP = '11' i.e. 16-bit MUX bus	P0.7-6	BUSCON0.BTYP
SMOD: Special modes (start/boot modes)	Startup modes selected via pins $\overline{RD}$ and ALE	P0.5-2	–
ADP: Adapt Mode	Not possible	P0.1	–
EMU: Emulation Mode	Not possible	P0.0	–
OWD disable	SYSCON.OWDDIS = '0' i.e. OWD is active	$\overline{RD}$	SYSCON.OWDDIS

1) Refers to the configuration pins which are replaced by the default values.

2) Software can modify the default values via these bitfields.

Note: Single-chip mode reset cannot be selected on ROMless devices. The attempt to read the first instruction after reset will fail in such a case.

Single-Chip Startup Modes

The startup mode (operation after reset) of the C164CI can be configured during reset. In single-chip mode this configuration is selected via pins $\overline{RD}$ and ALE.

Pin $\overline{RD}$ selects start or boot mode (instead of OWD control), pin ALE selects one of two alternatives in each case.

Table 20-8 Startup Mode Configuration in Single-Chip Reset Mode

$\overline{RD}$	ALE	Startup Mode	Notes
1	0	Standard Start	Execution starts at user memory location $00'0000_H$.
1	1	Alternate Start	Operation not yet defined. Do not use!
0	0	Standard Bootstrap Loader	Load 32 bytes via ASC0.
0	1	Alternate Boot Mode	Operation not defined. Do not use!

20.5 System Configuration via Software

The system configuration selected via hardware after reset (latched pin levels or default value) can be changed via software by executing a specific code sequence. The respective control bits are located within registers SYSCON, BUSCONx, and RSTCON. Register SYSCON can be modified only before the execution of instruction EINIT, while registers BUSCONx and RSTCON (using the specific sequence) can be modified repeatedly at any time.

The clock generation mode (CLKCFG), the segment address width (SALSEL), and the number of chip select lines (CSSEL) are controlled by register RP0H. RP0H is initialized according to the selected reset mode (pins or default). The respective configuration bitfields can be copied from register RSTCON upon entering Slow Down Divider mode if enabled by bit SUE = ‘1’.

The following steps must be taken to change the current configuration (see software example as well):

- Write intended configuration value to RSTCON
- Enter SDD mode
- Return to basic clock mode

```
CHANGE_CLOCK_CONFIGURATION:   ;“RSTCON” is a mem address, no SFR
MOV R15, #11100001xxxxxxxxB   ;Load a GPR with the target value
MOV RSTCON, R15               ;Enable update with PLL factor 4
EXTR #2                       ;ESFR-access to SYSCON2
MOV SYSCON2, #0100H           ;SDD mode, PLL on, factor 1
                              ;RSTCON.15-9 is copied to RP0H.15-9
MOV SYSCON2, #0400H           ;Switch to basic clock mode
                              ;System will run on PLL (factor 4)..
                              ;..after PLL has locked
```

Note: This software example assumes execution before EINIT. Otherwise, the unlock sequence must be executed prior to each access to RSTCON/SYSCON2.

Entering SDD mode temporarily ensures a correct clock signal synchronization in cases where the clock generation mode is changed (PLL factor, for example). If the target basic clock generation mode uses the PLL, the C164CI will run in direct-drive mode until the PLL has locked.

Software modification of system configuration values is protected by the following features:

- SYSCON is locked after EINIT
- RSTCON requires the unlock sequence after EINIT
- Copying RSTCON to RP0H must be explicitly enabled by setting bit SUE

RSTCON
Reset Control Register **mem ($F1E0_H$/--)** **Reset Value: $00XX_H$**

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

Bit	Function
RSTLEN	**Reset Length Control** (duration of the next reset sequence to occur)[1)] 00: 1024 TCL: standard duration, corresponds to all other derivatives without control function 01: 2048 TCL: extended duration, may be useful, for example, to provide additional settling for external configuration signals at high CPU clock frequencies 10: Reserved 11: Reserved
SUE	**Software Update Enable** 0: Configuration cannot be changed via software 1: Software update of configuration is enabled
CSSEL	**Chip Select Line Selection** (Number of active $\overline{CS}$ outputs) 00: 3 $\overline{CS}$ lines: $\overline{CS2}$ … $\overline{CS0}$ 01: 2 $\overline{CS}$ lines: $\overline{CS1}$ … $\overline{CS0}$ 10: No $\overline{CS}$ lines at all 11: All $\overline{CS}$ lines: $\overline{CSx}$ … $\overline{CS0}$
SALSEL	**Segment Address Line Selection** (Number of active seg-addr. outputs) 00: 4-bit segment address: A19 … A16 01: No segment address lines at all 10: Full segment address: Axx … A16 11: 2-bit segment address: A17 … A16
CLKCFG	**Clock Generation Mode Configuration** These pins define the clock generation mode, i.e. the mechanism by which the internal CPU clock is generated from the externally applied (XTAL1) input clock. 000: PLL ($f \times 2.5$) 001: Prescaler ($f / 2$) 010: PLL ($f \times 1.5$) 011: Direct Drive ($f = f$) 100: PLL ($f \times 5$) 101: PLL ($f \times 2$) 110: PLL ($f \times 3$) 111: PLL ($f \times 4$)

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

*Note: RSTCON is write protected after the execution of EINIT unless it is released via the unlock sequence (see **Section 21.7**).*
RSTCON can be accessed only via its long (mem) address.

21 Power Management

An increasingly important objective for microcontroller-based systems is the significant reduction of power consumption. A contradictory objective is to reach a certain level of system performance. Besides optimization of design and technology, a microcontroller's power consumption can generally be reduced by lowering its operating frequency and/or by reducing the clocked circuitry. The architecture of the C164CI provides three major means of reducing its power consumption under software control (see **Figure 21-1**):

- Reduction of the CPU frequency for Slow Down operation (Flexible Clock Generation Management)
- Selection of the active peripheral modules (Flexible Peripheral Management)
- Special operating modes to deactivate CPU, ports, and control logic (Idle, Sleep, Power Down)

This enables the application (i.e. 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, intermittent operation or standby.

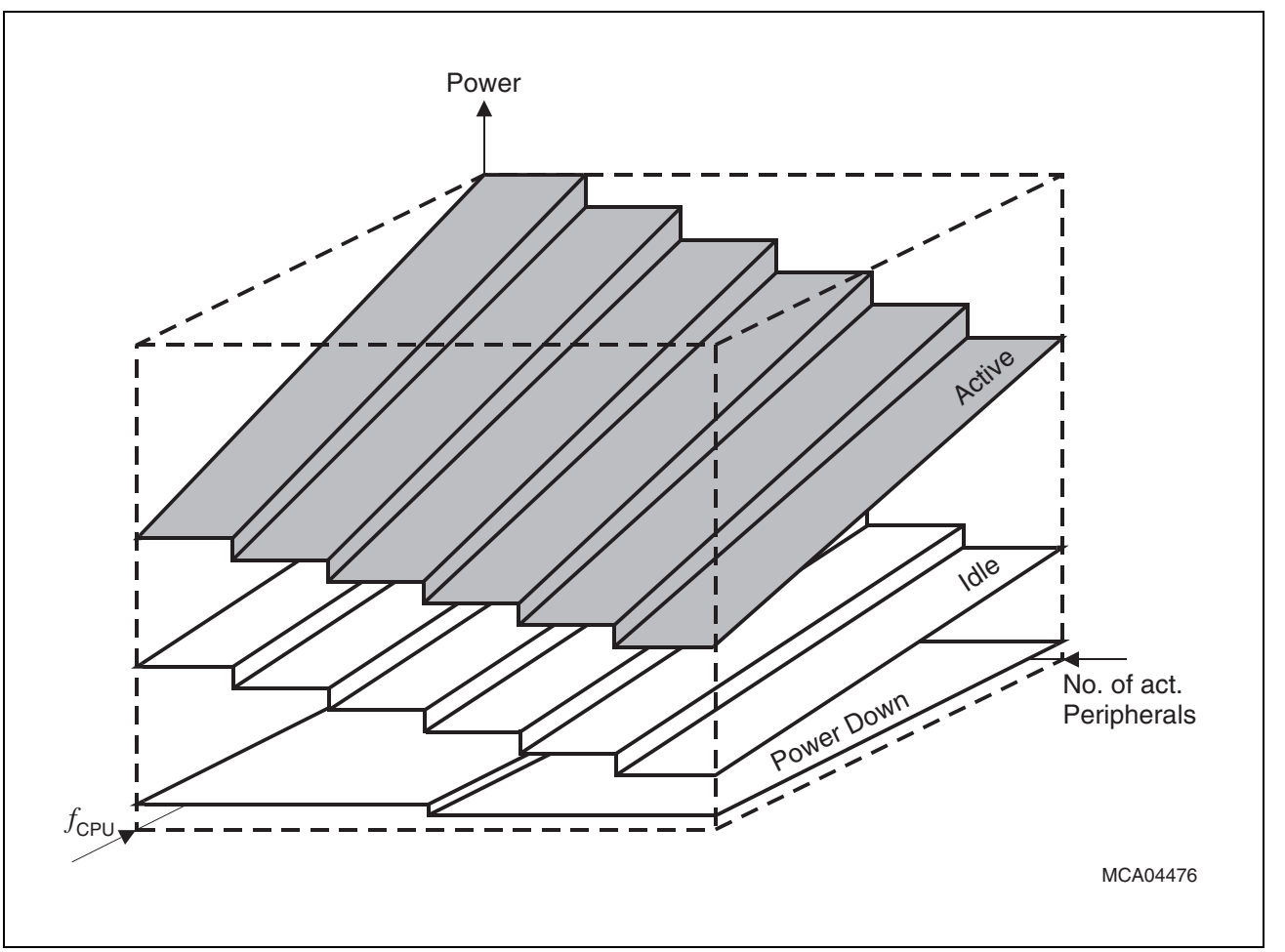


Figure 21-1 Power Reduction Possibilities

Intermittent operation (alternating phases of high performance and power saving) is supported by the cyclic interrupt generation mode of the on-chip RTC (Real Time Clock).

These three power reduction possibilities described above can be applied independently from each other and thus provide a maximum flexibility for each application.

For the basic power reduction modes (Idle, Power Down) there are dedicated instructions, while special registers control Sleep mode (SYSCON1), clock generation (SYSCON2), and peripheral management (SYSCON3).

Three different general power reduction modes with different levels of power reduction have been implemented in the C164CI. They may be entered under software control:

In **Idle Mode,** the CPU is stopped while the (enabled) peripherals continue their operation. Idle mode can be terminated by any reset or interrupt request.

In **Sleep Mode,** both the CPU and the peripherals are stopped. The real-time clock and its selected oscillator may optionally be kept running. Sleep mode can be terminated by any reset or interrupt request (mainly hardware requests as stopped peripherals cannot generate interrupt requests).

In **Power Down Mode,** both the CPU and the peripherals are stopped. The real time clock and its selected oscillator may optionally be kept running. Power Down mode can be terminated by a hardware reset only.

Note: All external bus actions are completed before a power saving mode is entered.

In addition the power management selects the current CPU frequency and controls which peripherals are active.

During **Slow Down Operation,** the basic clock generation path is bypassed and the CPU clock is generated via the programmable Slow Down Divider (SDD) from the selected oscillator clock signal.

Peripheral Management disables and enables the on-chip peripheral modules independently, reducing the amount of clocked circuitry (including the associated clock drivers).

21.1 Idle Mode

Power consumption of the C164CI microcontroller can be decreased by entering Idle mode. In this mode, all enabled peripherals continue to operate normally, **including** the watchdog timer; only the CPU operation is halted and the on-chip memory modules are disabled.

Note: Peripherals that have been disabled via software also remain disabled after entering Idle mode.

Idle mode is entered after the IDLE instruction has been executed and the instruction before the IDLE instruction has been completed (bitfield SLEEPCON in register SYSCON1 must be '00_B'). To prevent unintentionally entering 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.

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. If the interrupt request cannot be serviced because the priority is too low or the interrupt system is globally disabled, the CPU immediately resumes normal program execution with the instruction following the IDLE instruction.

For a request 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. If the PEC request cannot be serviced because the priority is too low or the interrupt system is globally disabled, the CPU does not remain in Idle mode but continues program execution with the instruction following the IDLE instruction.

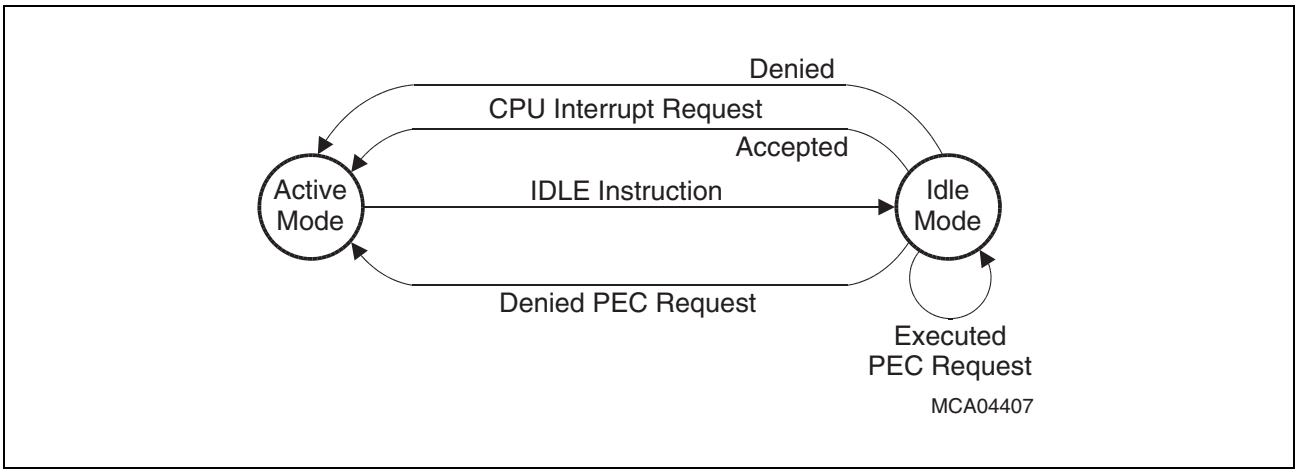


Figure 21-2 Transitions between Idle Mode and Active Mode

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 system 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 individually enabled interrupt request 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 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.

21.2 Sleep Mode

To further reduce power consumption, the microcontroller can be switched to Sleep mode. Clocking of all internal blocks is stopped (RTC and selected oscillator optionally). The contents of the internal RAM, however, are preserved through the voltage supplied via the V_{DD} pins. The watchdog timer is stopped in Sleep mode.

Sleep mode is selected via bitfield SLEEPCON in register SYSCON1 and is entered after the IDLE instruction has been executed and the instruction before the IDLE instruction has been completed.

Sleep 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. These are primarily external interrupts and the RTC (if running).

Note: The receive lines of serial interfaces may be internally routed to external interrupt inputs (see EXISEL). All peripherals except for the RTC are stopped and hence cannot generate an interrupt request.

The realtime clock (RTC) can be kept running in Sleep mode in order to maintain a valid system time as long as the supply voltage is applied. This enables a system to determine the current time and the duration of the period in Sleep mode (by comparing the current time with a timestamp stored when Sleep mode was entered). The supply current in this case remains well below 1 mA.

During Sleep mode, the voltage at the V_{DD} pins can be lowered to 2.7 V while the RTC and its selected oscillator will continue running and the contents of the internal RAM will be preserved. With the RTC (and oscillator) disabled the internal RAM is preserved down to a voltage of 2.5 V.

Note: When the RTC remains active in Sleep mode, the oscillator which generates the RTC clock signal will also continue running.
If the supply voltage is reduced, the specified maximum CPU clock frequency for this case must be taken into account.
For wakeup (input edge recognition and CPU start) the power level must be within the specified limits.

The total power consumption in Sleep mode depends on the active circuitry (whether the RTC is on or off) and on the current flowing through the port drivers. Individual port drivers can be disabled simply by configuring them for input.

The bus interface pins can be separately disabled by releasing the external bus (disable all address windows by clearing the BUSACT bits) and switching the ports to input (if necessary). The required software in this case must be executed from internal memory.

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
-	-	-	-	-	-	-	-	-	-	-	-	-	-	SLEEP CON	
-	-	-	-	-	-	-	-	-	-	-	-	-	-	rw	

Bit	Function
SLEEPCON	**SLEEP Mode Configuration** (mode entered upon the IDLE instruction) 00: Normal IDLE mode 01: SLEEP mode, with RTC running 10: Reserved. 11: SLEEP mode, with RTC and oscillator stopped

*Note: SYSCON1 is write protected after the execution of EINIT unless it is released via the unlock sequence (see **Table 21-6**).*

21.3 Power Down Mode

The microcontroller can be switched to Power Down mode to reduce the power consumption to a minimum. Clocking of all internal blocks is stopped (RTC and oscillator optionally); but, the contents of the internal RAM are preserved through the voltage supplied via the V_{DD} pins. The watchdog timer is stopped in Power Down mode. This mode can be terminated by an external hardware reset only (by asserting low level on the $\overline{\text{RSTIN}}$ pin). This reset will initialize all SFRs and ports to their default state, but will not change the contents of the internal RAM.

There are two levels of protection against unintentionally entering Power Down mode. First, the PWRDN (Power Down) instruction used to enter this mode has been implemented as a protected 32-bit instruction. Second, this instruction is effective **only** if the $\overline{\text{NMI}}$ (Non Maskable Interrupt) pin is externally pulled low while the PWRDN instruction is executed. The microcontroller will enter Power Down mode after the PWRDN instruction is completed.

This feature can be used in conjunction with an external power failure signal which pulls the $\overline{\text{NMI}}$ pin low when a power failure is imminent. The microcontroller will enter the NMI trap routine which can save the internal state into RAM. After the internal state has been saved, the trap routine may then execute the PWRDN instruction. If the $\overline{\text{NMI}}$ pin is still low at this time, Power Down mode will be entered; otherwise, program execution continues.

The initialization routine (executed upon reset) can check the reset identification flags in register WDTCON to determine whether the controller was initially switched on, or whether it was properly restarted from Power Down mode.

If the supply voltage continues to be applied, the realtime clock (RTC) can be kept running in Power Down mode to maintain a valid system time. This enables a system to determine the current time and the duration of the period in Power Down mode (by comparing the current time with a timestamp stored when Power Down mode was entered). The supply current in this case remains well below 1 mA.

During power down, the voltage at the V_{DD} pins can be lowered to 2.7 V while the RTC and its selected oscillator continue running and the contents of the internal RAM will be preserved.
With the RTC (and oscillator) disabled, the internal RAM is preserved down to a voltage of 2.5 V.

Note: When the RTC remains active in Power Down mode, the oscillator which generates the RTC clock signal will continue running.
If the supply voltage is reduced, the specified maximum CPU clock frequency for this case must be taken into account.

The total power consumption in Power Down mode depends on the active circuitry (whether the RTC is on or off) and on the current flowing through the port drivers. To minimize the consumed current, the RTC and/or all pin drivers can be disabled (pins switched to tristate) via a central control bitfield in register SYSCON2. If an application requires one or more port drivers to remain active even in Power Down mode, individual port drivers can be disabled simply by configuring them for input.

The bus interface pins can be separately disabled by releasing the external bus (disable all address windows by clearing the BUSACT bits) and switching the ports to input (if necessary). Of course, the required software in this case must be executed from internal memory.

21.3.1 Output Pins Status During Power Reduction Modes

In Idle mode, the CPU clocks are turned off and all peripherals continue their normal operation. Therefore, all port pins configured as general purpose output pins, will output the last data value written to their port output latches. If the alternate output function of a port pin is used by a peripheral, the state of the pin is determined by the operation of the peripheral.

Port pins used for bus control functions go into the state which represents the inactive state of the respective function (e.g. $\overline{WR}$), or to a defined state which is based on the last bus access (e.g. $\overline{BHE}$). Port pins used as external address/data bus hold the address/data which was output during the last external memory access before entry into Idle mode under the following conditions:

- P0H outputs the high byte of the last address if a multiplexed bus mode with 8-bit data bus is used. Otherwise, P0H is floating. P0L is always floating in Idle mode.
- PORT1 outputs the lower 16 bits of the last address if a demultiplexed bus mode is used. Otherwise, the output pins of PORT1 represent the port latch data.
- Port 4 outputs the segment address for the last access on those pins that are selected as segment address lines. Otherwise, the output pins of Port 4 represent the port latch data.

In Sleep mode, the oscillator (except for RTC operation) and the clocks to the CPU and to the peripherals are turned off. As in Idle mode, all port pins configured as general purpose output pins will output the last data value written to their port output latches.

When the alternate output function of a port pin is used by a peripheral the state of this pin is determined by the last action of the peripheral before the clocks were switched off.

In Power Down mode, the oscillator (except for RTC operation) and the clocks to the CPU and to the peripherals are turned off. As in Idle mode, all port pins configured as general purpose output pins will output the last data value written to their port output latches.

When the alternate output function of a port pin is used by a peripheral, the state of this pin is determined by the last action of the peripheral before the clocks were switched off.

Note: All pin drivers can be switched off by selecting the general port disable function prior to entering Power Down mode.
When the supply voltage is lowered in Power Down mode, the high voltage of output pins will decrease accordingly.

Table 21-1 State of C164CI Output Pins in Idle and Power Down Modes

C164CI Output Pin(s)	External Bus Enabled		No External Bus	
	Idle Mode	**Sleep and Power Down**	**Idle Mode**	**Sleep and Power Down**
CLKOUT	Active (toggling)	High	Active (toggling)	High
FOUT	Active (toggling)	Hold (high / low)	Active (toggling)	Hold (high / low)
ALE	Low		Low	
$\overline{RD}$, $\overline{WR}$	High		High	
P0L	Floating		Port Latch Data	
P0H	A15 … A8[1] / Float		Port Latch Data	
PORT1	Last Address[2] / Port Latch Data		Port Latch Data	
Port 4	Port Latch Data / Last segment		Port Latch Data	
$\overline{BHE}$	Last value		Port Latch Data	
$\overline{CSx}$	Last value[3]		Port Latch Data	
$\overline{RSTOUT}$	High if EINIT was executed before entering Idle or Power Down mode, Low otherwise.			
Other Port Output Pins	Port Latch Data / Alternate Function			

1) For multiplexed buses with 8-bit data bus.

2) For demultiplexed buses.

3) The $\overline{CS}$ signal corresponding to the last address remains active (low). All other enabled $\overline{CS}$ signals remain inactive (high). By accessing an on-chip X-Peripheral prior to entering a power save mode, all external $\overline{CS}$ signals can be deactivated.

21.4 Slow Down Operation

A separate clock path can be selected for Slow Down operation, bypassing the basic clock path used for standard operation. The programmable Slow Down Divider (SDD) divides the oscillator frequency by a factor of 1 … 32 which is specified via bitfield CLKREL in register SYSCON2 (factor = <CLKREL>+1). When bitfield CLKREL is written during SDD operation, the reload counter will output one more clock pulse with the “old” frequency in order to resynchronize internally before generating the “new” frequency.

If direct drive mode is configured, clock signal f_{DD} is directly fed to f_{CPU}. If prescaler mode is configured, clock signal f_{DD} is additionally divided by 2:1 to generate f_{CPU} (see examples below).

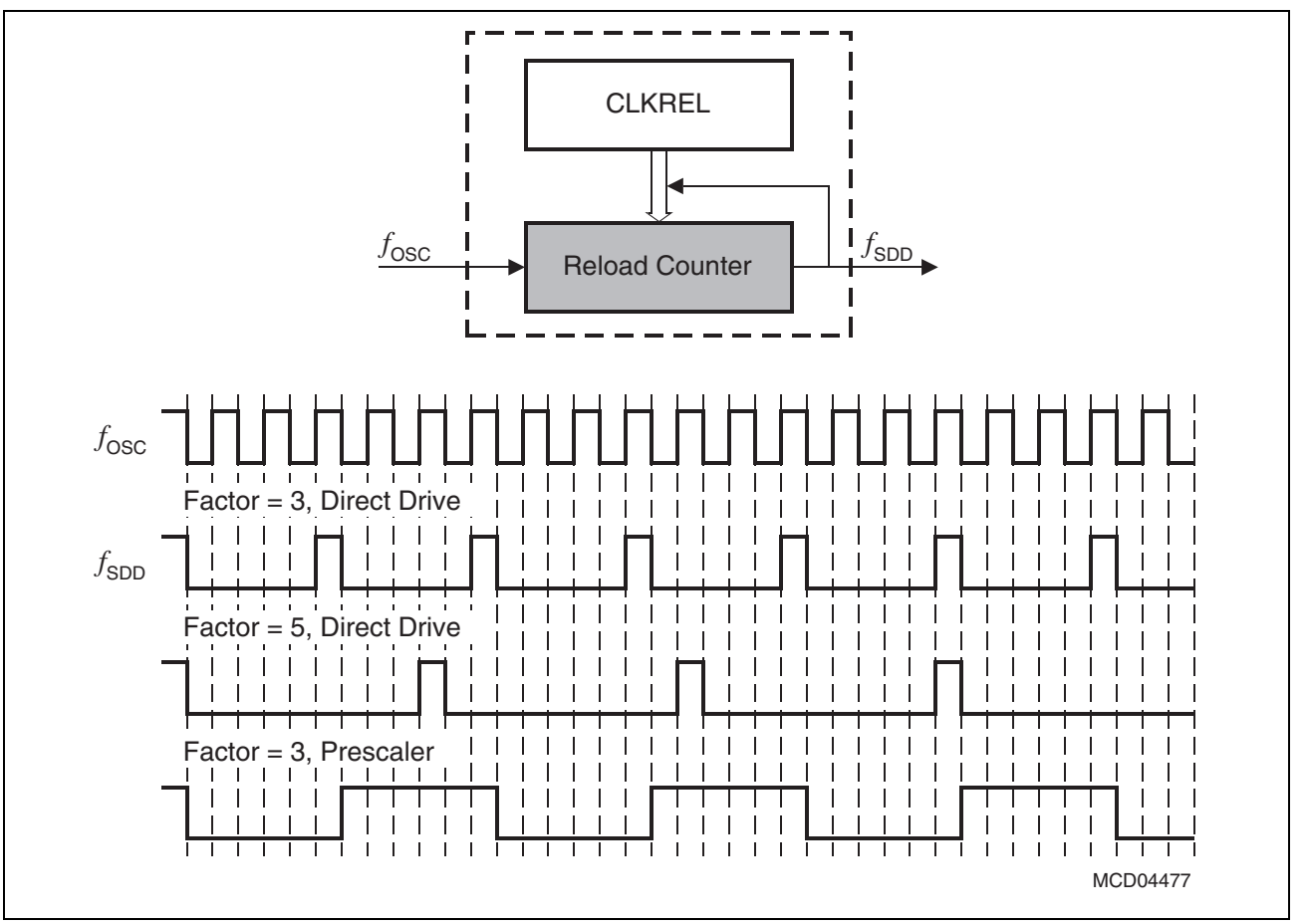


Figure 21-3 Slow Down Divider Operation

Using a 5 MHz input clock, for example, the on-chip logic may be run at a frequency down to 156.25 kHz (or 78 kHz) without an external hardware change. An implemented PLL may be switched off in this case or may continue running, depending on the requirements of the application (see **Table 21-2**).

*Note: During Slow Down operation, the entire device (including bus interface and generation of signals CLKOUT or FOUT) is clocked with the SDD clock (see **Figure 21-3**).*

Table 21-2 PLL Operation in Slow Down Mode

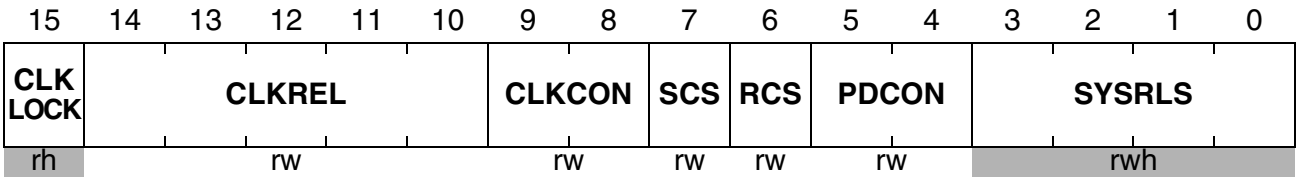
	Advantage	Disadvantage	Oscillator Watchdog
PLL running	Fast switching back to basic clock source	PLL adds to power consumption	Active if not disabled via bit OWDDIS
PLL off	PLL causes no additional power consumption	PLL must lock before switching back to the basic clock source (if the PLL is the basic clock source)	Disabled

These clock options are selected via bitfield CLKCON in register SYSCON2. A state machine controls the switching mechanism itself and ensures a continuous and glitch-free clock signal to the on-chip logic. This is especially important when switching back to PLL frequency after the PLL has temporarily been switched off. In this case, the clock source can be switched back either automatically as soon as the PLL is locked again (indicated by bit CLKLOCK in register SYSCON2), or manually, (under software control, after bit CLKLOCK has become ‘1’). The latter way is preferable if the application requires a defined point at which the frequency changes.

Note: When the PLL is the basic clock source and a reset occurs during SDD operation with the PLL off, the internal reset condition is extended so the PLL can lock before execution begins. The reset condition is terminated prematurely if no stable oscillator clock is detected. This ensures the operability of the device in case of a missing input clock signal.

Switching to Slow Down operation affects frequency sensitive peripherals such as serial interfaces, timers, PWM, etc. If these units are to be operated in Slow Down mode, their precalers or reload values must be adapted. Please note that the reduced CPU frequency decreases such things as timer resolution and increases the step width for example for baudrate generation. The oscillator frequency in such a case should be chosen to accommodate the required resolutions and/or baudrates.

SYSCON2
System Control Reg.2 **ESFR ($F1D0_H/E8_H$)** **Reset Value: $00X0_H$**



Bit	Function
SYSRLS	**Register Release Function** (Unlock field) Must be written in a defined way in order to execute the unlock sequence. See separate description (**Table 21-6**).
PDCON	**Power Down Control** (during power down mode) 00: RTC = On, Ports = On (default after reset). 01: RTC = On, Ports = Off. 10: RTC = Off, Ports = On. 11: RTC = Off, Ports = Off.
RCS	**RTC Clock Source** (not affected by a reset!) 0: Main oscillator. 1: Reserved.
SCS	**SDD Clock Source** (not affected by a reset!) 0: Main oscillator. 1: Reserved.
CLKCON	**Clock State Control** 00: Running on configured basic frequency. 01: Running on slow down frequency, PLL remains ON. 10: Running on slow down frequency, PLL switched OFF. 11: Reserved. Do not use this combination.
CLKREL	**Reload Counter Value for Slowdown Divider** (SDD factor = CLKREL + 1)
CLKLOCK	**Clock Signal Status Bit** 0: Main oscillator is unstable or PLL is unlocked. 1: Main oscillator is stable **and** PLL is locked.

*Note: SYSCON2 (except for bitfield SYSRLS) is write protected after the execution of EINIT unless it is released via the unlock sequence (see **Table 21-6**).*

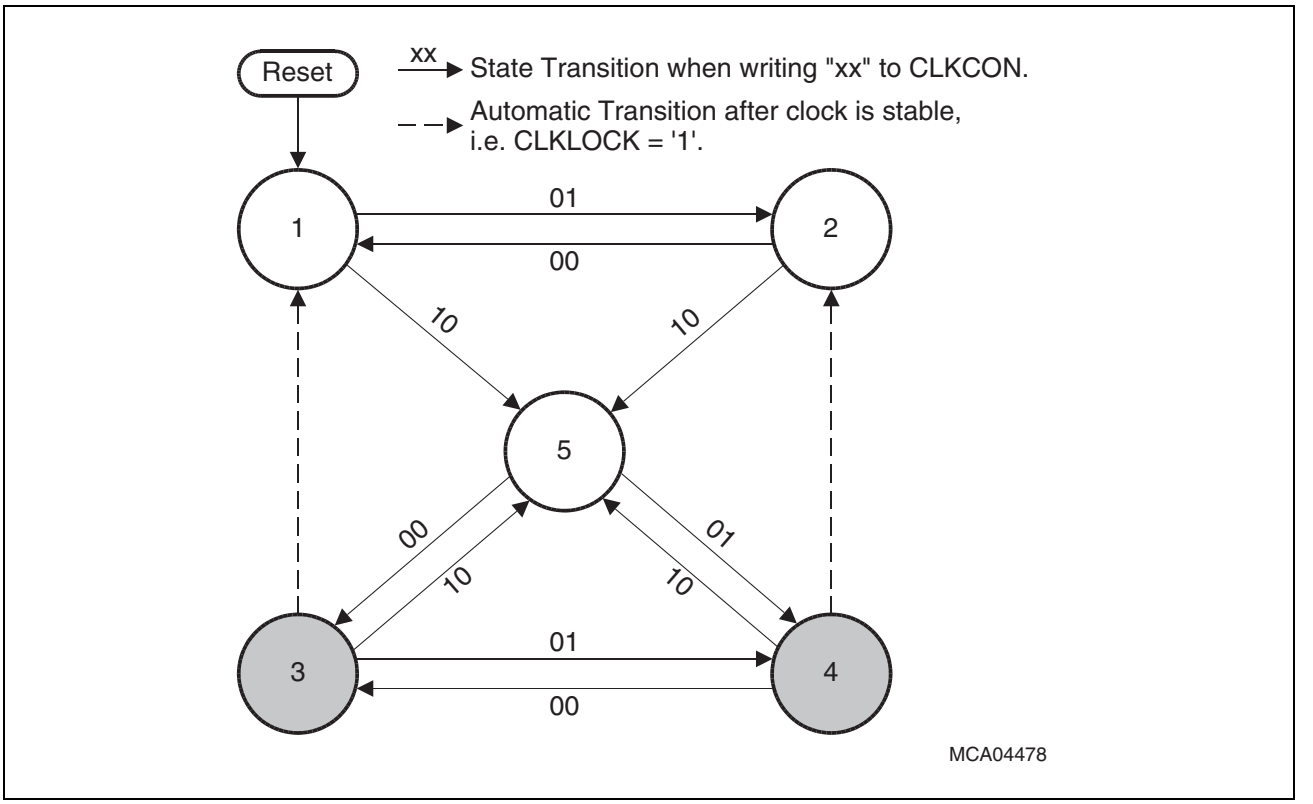


Figure 21-4 Clock Switching State Machine

Table 21-3 Clock Switching State Description

State Number	PLL Status	f_{CPU} Source	CLK CON	Note
1	Locked[1)]	Basic	00	Standard operation on basic clock frequency.
2	Locked[1)]	SDD	01	SDD operation with PLL On[1)]. Fast (without delay) or manual switch back (from 5) to basic clock frequency.
3	Transient[1)]	SDD	(00)	Intermediate state leading to state 1.
4	Transient[1)]	SDD	(01)	Intermediate state leading to state 2.
5	Off	SDD	10	SDD operation with PLL Off. Reduced power consumption.

1) The indicated PLL status applies only if the PLL is selected as the basic clock source. If the basic clock source is direct drive or prescaler, the PLL will not lock. If the oscillator watchdog is disabled (OWDDIS = ‘1’) the PLL will be off.

21.5 Flexible Peripheral Management

The power consumed by the C164CI also depends on the amount of active logic. Peripheral management enables the system designer to deactivate those on-chip peripherals not required in a given system status (such as a certain interface mode or standby). All modules remaining active will continue with their usual performance. If all modules fed by the Peripheral Clock Driver (PCD) are disabled and the other functions fed by the PCD are not required, this clock driver itself may also be disabled to save additional power.

This flexibility is accomplished by distributing the CPU clock via several clock drivers, each of which can be separately controlled, and can also be smaller.

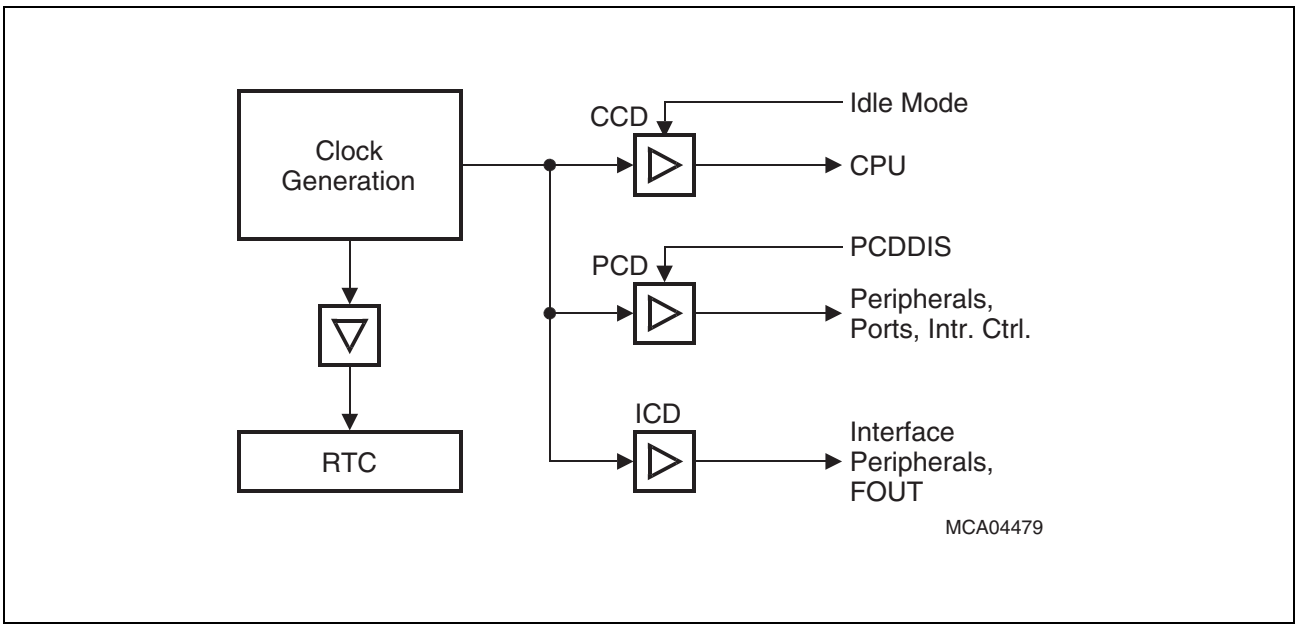


Figure 21-5 CPU Clock Distribution

Note: The Real Time Clock (RTC) is fed by a separate clock driver, so it can be kept running even in Power Down mode while all the other circuitry is disconnected from the clock.

The registers of the generic peripherals can be accessed even while the respective module is disabled, as long as PCD is running. The registers of peripherals connected to ICD can be accessed even in this case, of course. The registers of X-peripherals cannot be accessed while the respective module is disabled by any means.

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

This flexible peripheral management is controlled by software via register SYSCON3 in which each control bit is associated with an on-chip peripheral module.

SYSCON3
System Control Reg.3 **ESFR ($F1D4_H/EA_H$)** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PCD DIS	-	CAN1 DIS	-	-	-	-	CC6 DIS	CC2 DIS	-	-	-	GPT DIS	SSC DIS	ASC 0 DIS	ADC DIS
rw	-	rw	-	-	-	-	rw	rw	-	-	-	rw	rw	rw	rw

Bit	**Function** (associated peripheral module)
ADCDIS	Analog/Digital Converter
ASC0DIS	USART ASC0
SSCDIS	Synchronous Serial Channel SSC
GPTDIS	General Purpose Timer Blocks
CC2DIS	CAPCOM2 Unit
CC6DIS	CAPCOM6 Unit
CAN1DIS	On-chip CAN Module 1[1)]
PCDDIS	Peripheral Clock Driver (also X-Peripherals)

1) When bit CAN1DIS is cleared the CAN module is re-activated by an internal reset signal and must then be re-configured in order to operate properly.

Note: The allocation of peripheral disable bits within register SYSCON3 is device specific and may be different in derivatives other than the C164CI.
*SYSCON3 is write protected after the execution of EINIT unless it is released via the unlock sequence (see **Table 21-6**).*

When disabling the peripheral clock driver (PCD), the following details should be taken into account:

- The clock signal for all connected peripherals is stopped. Make sure that all peripherals enter a safe state before disabling PCD.
- The output signal CLKOUT will remain HIGH (FOUT will keep on toggling).
- Interrupt requests will still be recognized even while PCD is disabled.
- No new output values are gated from the port output latches to the output port pins and no new input values are latched from the input port pins.
- No register access is possible for generic peripherals
  (register access is possible for individually disabled generic peripherals,
  no register access is possible for disabled X-Peripherals).

21.6 Programmable Frequency Output Signal

The system clock output (CLKOUT) can be replaced by the programmable frequency output signal f_{OUT}. This signal can be controlled via software (unlike CLKOUT), and so can be adapted to the requirements of connected external circuitry. The programmability extends power management to the system level as circuitry external to the C164CI can be influenced. Peripherals, for instance, can run at a scalable frequency or can be switched off temporarily.

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

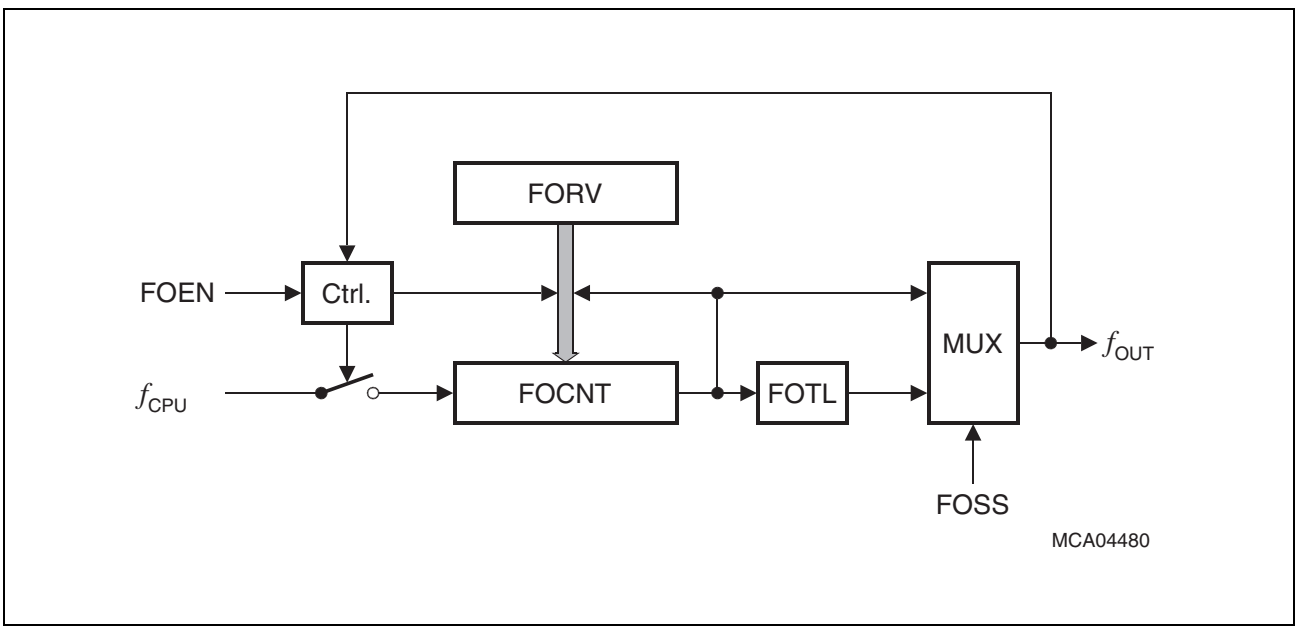


Figure 21-6 Clock Output Signal Generation

Signal f_{OUT} always provides complete output periods (see Signal Waveforms below):

- 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’.

Signal f_{OUT} is independent of the peripheral clock driver PCD. While CLKOUT will stop if PCD is disabled, f_{OUT} will continue to toggle. Thus, external circuitry may be controlled independently from on-chip peripherals.

*Note: Counter FOCNT is clocked with the CPU clock signal f_{CPU} (see **Figure 21-6**) and, therefore, will also be influenced by the SDD operation.*

Register FOCON provides control over the output signal generation (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**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FOEN	FOSS	FORV						-	FOTL	FOCNT					
rw	rw	rw						-	rwh	rwh					

Bit	Function
FOCNT	**Frequency Output Counter**
FOTL	**Frequency Output Toggle Latch** Is toggled upon each underflow of FOCNT.
FORV	**Frequency Output Reload Value** Is copied to FOCNT upon each underflow of FOCNT.
FOSS	**Frequency Output Signal Select** 0: Output of the toggle latch: duty cycle = 50%. 1: Output of the reload counter: duty cycle depends on FORV.
FOEN	**Frequency Output Enable** 0: Frequency output generation stops when signal f_{OUT} is/gets low. 1: FOCNT is running, f_{OUT} is gated to pin. First reload after 0-1 transition.

Note: It is not recommended to write to any part of bitfield FOCNT, especially while the counter is running. Writing to FOCNT prior to starting the counter is obsolete because it will be immediately reloaded from FORV. Writing to FOCNT during operation may produce unintended counter values.

Signal f_{OUT} in the C164CI is an alternate function of pin P3.15/CLKOUT/FOUT.

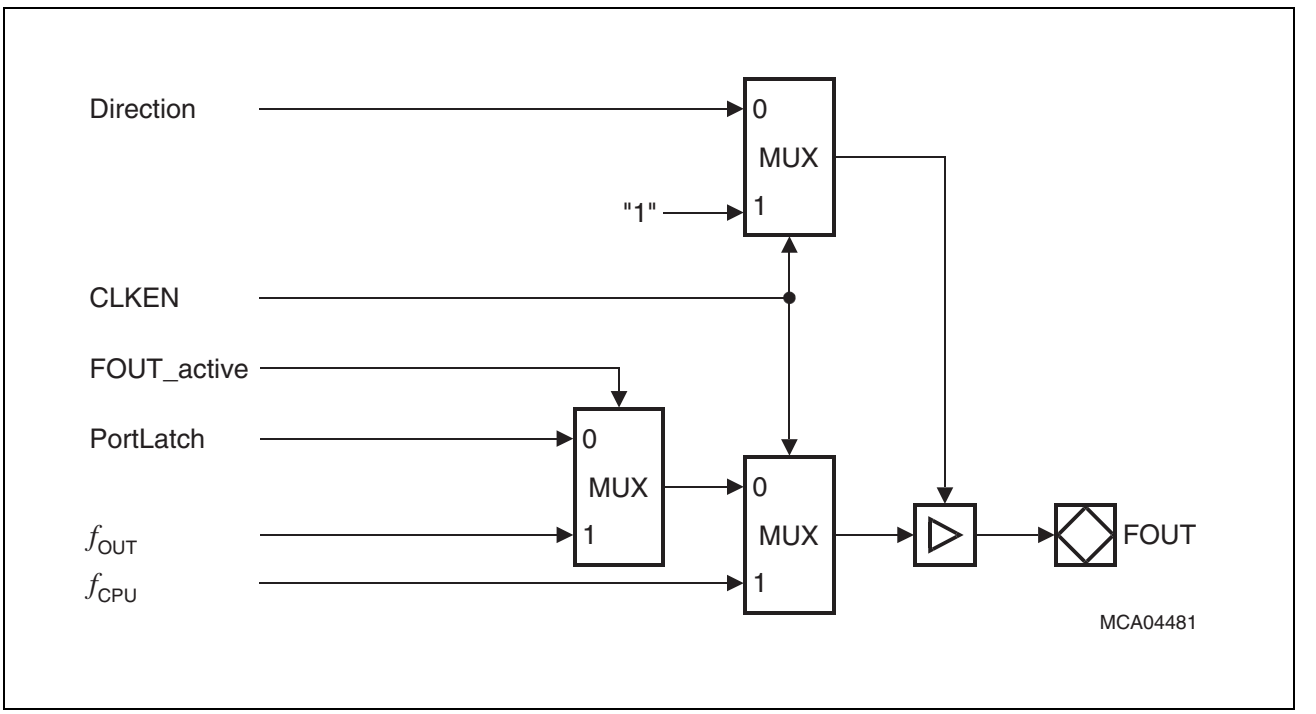


Figure 21-7 Connection to Port Logic (Functional Approach)

A priority ranking determines which function controls the shared pin:

Table 21-4 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’

*Note: For the generation of f_{OUT} pin FOUT must be switched to output, i.e. DP3.15 = ‘1’. While f_{OUT} is disabled, the pin is controlled by the port latch (see **Figure 21-7**). The port latch P3.15 must be ‘0’ in order to maintain the f_{OUT} inactive level on the pin.*

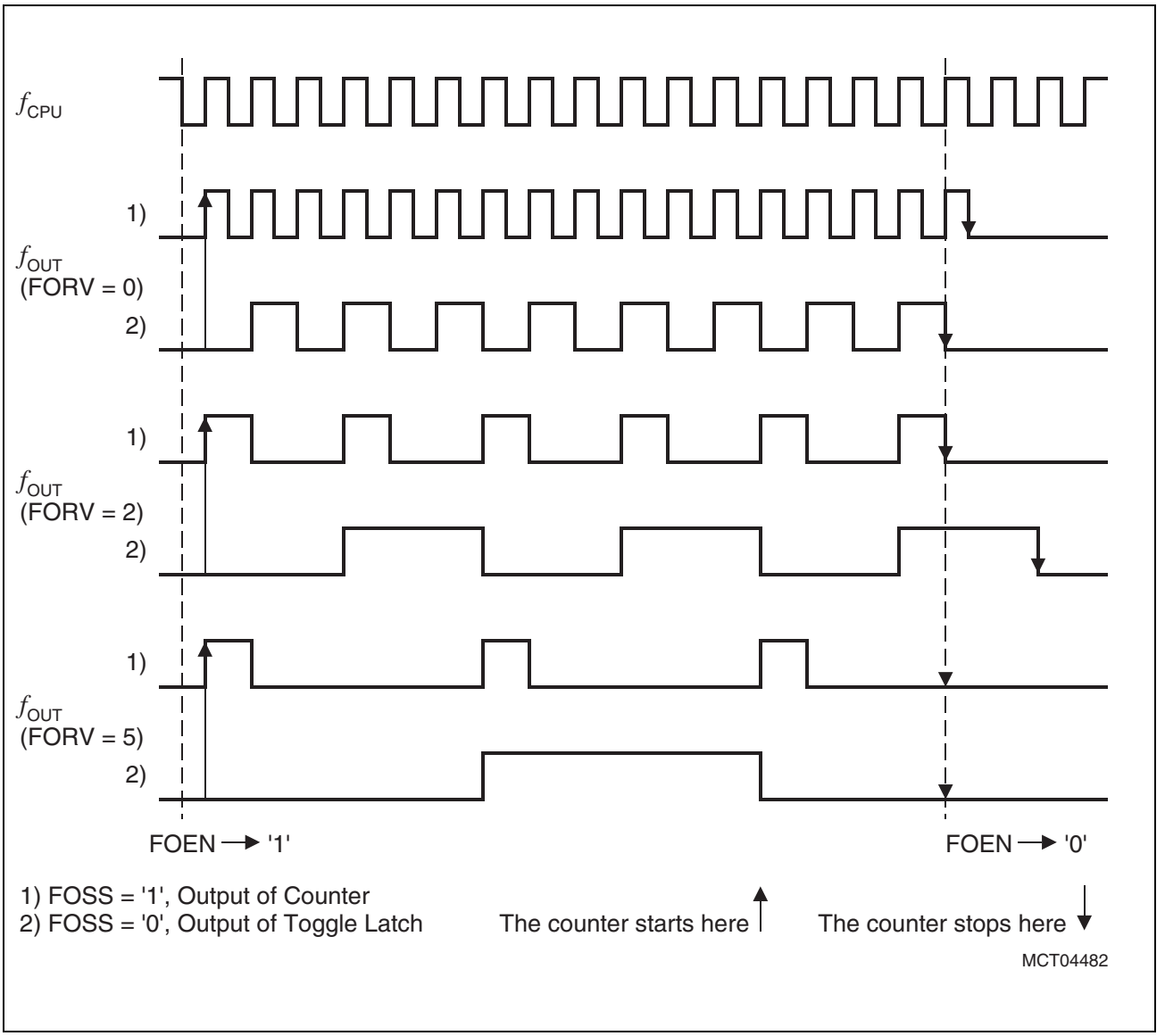


Figure 21-8 Signal Waveforms

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

Output Frequency Calculation

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

Table 21-5 Selectable Output Frequency Range for f_{OUT}

f_{CPU}	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	515.625 257.816	$0F_H$ (1.03125) 10_H (0.97059)	20_H

21.7 Security Mechanism

The power management control registers belong to a set of registers (see **Table 21-7**) which control functions and modes critical to the C164CI's operation. For this reason, they are locked (except for bitfield SYSRLS in register SYSCON2) after the execution of EINIT (like register SYSCON) to ensure that these vital system functions cannot be changed inadvertently, by software errors. However, because these registers control important functions (e.g. the power management) they need to be accessed during operation to select the appropriate mode. The system control software gains this access via a special unlock sequence which allows **one single** write access **to one register** of this set when executed properly. This provides maximum security.

Note: All these registers may be read at any time without restrictions.

The unlock sequence is executed by writing defined values to bitfield SYSRLS using defined instructions (see **Table 21-6**). The instructions of the unlock sequence (including the intended write access) must be secured with an EXTR instruction (switch to ESFR space and lock interrupts).

Note: The unlock sequence is aborted if the locked range (EXTR) does not cover the complete sequence.
The unlock sequence provides no write access to register SYSCON.

Table 21-6 Unlock Sequence for Secured Registers

Step	SYSRLS	Instruction	Notes
–	0000_B [1]	–	Status before release sequence
1	1001_B	BFLDL, OR, ORB[2], XOR, XORB[2]	Read-Modify-Write access
2	0011_B	MOV, MOVB[2], MOVBS[2], MOVBZ[2]	Write access
3	0111_B	BSET, BMOV[2], BMOVN[2], BOR[2], BXOR[2]	Read-Modify-Write access, bit instruction
4	–	–	Single (read-modify-)write access to one of the secured registers
–	0000_B [3]	–	Status after release sequence

1) SYSRLS must be set to 0000_B before the first step, if any OR command is used.

2) Usually byte accesses should not be used for Special Function Registers.

3) SYSRLS is cleared by hardware if unlock sequence and write access were successful. Otherwise, SYSRLS shows the last value written.

The following registers are secured by the described unlock sequence:

Table 21-7 Special Registers Secured by the Unlock Sequence

Register Name	Description
SYSCON1	Controls sleep mode
SYSCON2	Controls clock generation (SDD) and the unlock sequence itself
SYSCON3	Controls the flexible peripheral management
RSTCON	Controls the configuration of the C164CI (basic clock generation mode, $\overline{CS}$ lines, segment address width) and the length of the reset sequence

Code Examples

The code examples below show how the unlock sequence is used to access register SYSCON2 (marked *!* in the comment column) in an application in order to change the basic clock generation mode.

Examples where the PLL keeps running:

```
;_______________________________;_____________________________________________
ENTER_SLOWDOWN:                ;Currently running on basic clock frequ.
MOV   SYSCON2, ZEROS           ;Clear bits 3-0 (no EXTR required here)
EXTR  #4H                      ;Switch to ESFR space and lock sequence
BFLDL SYSCON2,#0FH,#09H        ;Unlock sequence, step 1 (1001B)
MOV   SYSCON2,#0003H           ;Unlock sequence, step 2 (0011B)
BSET  SYSCON2.2                ;Unlock sequence, step 3 (0111B)
                               ;Single access to one locked register
BFLDH SYSCON2,#03H,#01H        ;CLKCON=01B --> SDD frequency, PLL on *!*

;_______________________________;_____________________________________________
EXIT_SLOWDOWN:                 ;Currently running on SDD frequency
MOV   SYSCON2, ZEROS           ;Clear bits 3-0 (no EXTR required here)
EXTR  #4H                      ;Switch to ESFR space and lock sequence
BFLDL SYSCON2,#0FH,#09H        ;Unlock sequence, step 1 (1001B)
MOV   SYSCON2,#0003H           ;Unlock sequence, step 2 (0011B)
BSET  SYSCON2.2                ;Unlock sequence, step 3 (0111B)
                               ;Single access to one locked register
BFLDH SYSCON2,#03H,#00H        ;CLKCON=00B --> basic frequency      *!*
```

Examples where the PLL is disabled:

```
;_____________________________;_____________________________________________
ENTER_SLOWDOWN:              ;Currently running on basic clock frequ.
EXTR  #1H                    ;Next access to ESFR space
BCLR  ISNC.2                 ;PLLIE='0', i.e. PLL interrupt disabled
MOV   SYSCON2, ZEROS         ;Clear bits 3-0 (no EXTR required here)
EXTR  #4H                    ;Switch to ESFR space and lock sequence
BFLDL SYSCON2,#0FH,#09H      ;Unlock sequence, step 1 (1001B)
MOV   SYSCON2,#0003H         ;Unlock sequence, step 2 (0011B)
BSET  SYSCON2.2              ;Unlock sequence, step 3 (0111B)
                             ;Single access to one locked register
BFLDH SYSCON2,#03H,#02H      ;CLKCON=10B --> SDD frequency, PLL off*!*

;_____________________________;_____________________________________________
SDD_EXIT_AUTO:               ;Currently running on SDD frequency
MOV   SYSCON2, ZEROS         ;Clear bits 3-0 (no EXTR required here)
EXTR  #4H                    ;Switch to ESFR space and lock sequence
BFLDL SYSCON2,#0FH,#09H      ;Unlock sequence, step 1 (1001B)
MOV   SYSCON2,#0003H         ;Unlock sequence, step 2 (0011B)
BSET  SYSCON2.2              ;Unlock sequence, step 3 (0111B)
                             ;Single access to one locked register
BFLDH SYSCON2,#03H,#00H      ;CLKCON=00B--> basic frequ./start PLL*!*
EXTR  #1H                    ;Next access to ESFR space
BSET  ISNC.2                 ;PLLIE='1', i.e. PLL interrupt enabled
```

```
;_______________________________;_____________________________________________
SDD_EXIT_MANUAL:                ;Currently running on SDD frequency
MOV   SYSCON2, ZEROS            ;Clear bits 3-0 (no EXTR required here)
EXTR  #4H                       ;Switch to ESFR space and lock sequence
BFLDL SYSCON2,#0FH,#09H         ;Unlock sequence, step 1 (1001B)
MOV   SYSCON2,#0003H            ;Unlock sequence, step 2 (0011B)
BSET  SYSCON2.2                 ;Unlock sequence, step 3 (0111B)
                                ;Single access to one locked register
BFLDH SYSCON2,#03H,#01H         ;CLKCON=01B --> stay on SDD/start PLL *!*
                                ;
USER_CODE:                      ;Space for any user code that...
                                ;...must or can be executed before...
                                ;...switching back to basic clock
CLOCK_OK:
EXTR  #1H                       ;Next access to ESFR space
JNB   SYSCON2.15,CLOCK_OK;Wait until clock OK (CLKLOCK='1')
                                ;
MOV   SYSCON2, ZEROS            ;Clear bits 3-0 (no EXTR required here)
EXTR  #4H                       ;Switch to ESFR space and lock sequence
BFLDL SYSCON2,#0FH,#09H         ;Unlock sequence, step 1 (1001B)
MOV   SYSCON2,#0003H            ;Unlock sequence, step 2 (0011B)
BSET  SYSCON2.2                 ;Unlock sequence, step 3 (0111B)
                                ;Single access to one locked register
BFLDH SYSCON2,#03H,#00H         ;CLKCON=00B --> basic frequency      *!*
EXTR  #1H                       ;Next access to ESFR space
BSET  ISNC.2                    ;PLLIE='1', i.e. PLL interrupt enabled
```

22 System Programming

A number of features have been incorporated into the instruction set of the C164CI, to facilitate software development, including constructs for modularity, loops, and context switching. In many cases, commonly used instruction sequences have been simplified while their flexibility has been enhanced. The following programming features help to fully utilize this instruction set.

Instructions Provided as Subsets of Instructions

In many cases, instructions found in other microcontrollers are provided as subsets of more powerful instructions in the C164CI. This allows the same functionality to be provided while decreasing the hardware required and decreasing decode complexity. To assist with assembly programming, these instructions which are familiar from other microcontrollers, can be built in macros, thus providing the same names.

Direct Substitution Instructions are instructions known from other microcontrollers which can be replaced by the following instructions of the C164CI:

Table 22-1 Substitution of Instructions

Substituted Instruction		C164CI Instruction		Function
CLR	Rn	AND	Rn, #0_H	Clear register
CPLB	Bit	BMOVN	Bit, Bit	Complement bit
DEC	Rn	SUB	Rn, #1_H	Decrement register
INC	Rn	ADD	Rn, #1_H	Increment register
SWAPB	Rn	ROR	Rn, #8_H	Swap bytes within word

Modification of System Flags is performed using bit set or bit clear instructions (BSET, BCLR). All bit and word instructions can access the PSW register, so instructions such as CLEAR CARRY or ENABLE INTERRUPTS are not required.

External Memory Data Access does not require special instructions to load data pointers or explicitly load and store external data. The C164CI provides a Von Neumann memory architecture and its on-chip hardware automatically detects accesses to internal RAM, GPRs, and SFRs.

Multiplication and Division

Multiplication and division of words and double words are provided through multiple cycle instructions implementing a Booth algorithm. Each instruction implicitly uses the 32-bit register MD (MDL = lower 16 bits, MDH = upper 16 bits). The MDRIU flag (Multiply or Divide Register In Use) in register MDC is set whenever either half of this register is written to or when a multiply/divide instruction is started. It is cleared whenever the MDL

register is read. Because an interrupt can be acknowledged before the contents of register MD are saved, this flag is required to alert interrupt routines which require multiply/divide hardware, so they can preserve register MD. This register, however, only needs to be saved when an interrupt routine requires the use of the MD register and a previous task has not saved the current result. This flag is easily tested by the Jump-on-bit instructions.

Multiplication or division is simply performed by specifying the correct (signed or unsigned) version of the multiply or divide instruction. The result is then stored in register MD. The overflow flag (V) is set if the result from a multiply or divide instruction is greater than 16 bits. This flag can be used to determine whether both word halves must be transferred from register MD. The high portion of register MD (MDH) must be moved into the register file or memory first to ensure that the MDRIU flag reflects the correct state.

The following instruction sequence performs an unsigned 16 by 16-bit multiplication:

```
SAVE:
JNB      MDRIU, START;Test if MD was in use.
SCXT     MDC, #0010H ;Save and clear control register,
                     ;leaving MDRIU set
                     ;(only required for interrupted
                     ;multiply/divide instructions)
BSET     SAVED       ;Indicate the save operation
PUSH     MDH         ;Save previous MD contents …
PUSH     MDL         ;… on system stack
START:
MULU     R1, R2      ;Multiply 16·16 unsigned, Sets MDRIU
JMPR     cc_NV, COPYL;Test for only 16-bit result
MOV      R3, MDH     ;Move high portion of MD
COPYL:
MOV      R4, MDL     ;Move low portion of MD, Clears MDRIU
RESTORE:
JNB      SAVED, DONE ;Test if MD registers were saved
POP      MDL         ;Restore registers
POP      MDH
POP      MDC
BCLR     SAVED       ;Multiplication is completed,
                     ;program continues
DONE:     …
```

The save sequence shown above and the restore sequence after COPYL are only required if the current routine could have interrupted a previous routine which contained a MUL or DIV instruction. Register MDC is also saved because it is possible that a previous routine's Multiply or Divide instruction was interrupted while in progress. In this case, the information required to restart the instruction is contained in this register. Register MDC must be cleared to be correctly initialized for a subsequent multiplication or division. The old MDC contents must be popped from the stack before the RETI instruction is executed.

For division, the user must first move the dividend into the MD register. If a 16/16-bit division is specified, only the low portion of register MD must be loaded. The result is also stored into register MD. The low portion (MDL) contains the integer result of the division, while the high portion (MDH) contains the remainder.

The following instruction sequence performs a 32 by 16-bit division:

```
MOV      MDH, R1      ;Move dividend to MD register. Sets MDRIU
MOV      MDL, R2      ;Move low portion to MD
DIV      R3           ;Divide 32/16 signed, R3 holds divisor
JMPR     cc_V, ERROR ;Test for divide overflow
MOV      R3, MDH      ;Move remainder to R3
MOV      R4, MDL      ;Move integer result to R4. Clears MDRIU
```

Whenever a multiply or divide instruction is interrupted while in progress, the address of the interrupted instruction is pushed onto the stack and the MULIP flag in the PSW of the interrupting routine is set. When the interrupt routine is exited with the RETI instruction, this bit is implicitly tested before the old PSW is popped from the stack. If MULIP = '1' the multiply/divide instruction is re-read from the location popped from the stack (return address) and will be completed after the RETI instruction has been executed.

*Note: The MULIP flag is part of the **context of the interrupted task**. When the interrupting routine does not return to the interrupted task (for example, scheduler switches to another task) the MULIP flag must be set or cleared according to the context of the task to be executed next.*

BCD Calculations

No direct support for BCD calculations is provided in the C164CI. BCD calculations are performed by converting BCD data to binary data, performing the desired calculations using standard data types, and converting the result back to BCD data. Due to the enhanced performance of division instructions, binary data is quickly converted to BCD data through division by 10_D. Conversion from BCD data to binary data is enhanced by multiple bit shift instructions. This provides similar performance compared to instructions directly supporting BCD data types without requiring additional hardware.

22.1 Stack Operations

The C164CI supports two types of stacks: the system stack and the user stack. The system stack is used implicitly by the controller and is located in the internal RAM. The user stack provides stack access to the user in either the internal or external memory. Both stack types grow from high memory addresses to low memory addresses.

Internal System Stack

A system stack is provided to store return vectors, segment pointers, and processor status for procedures and interrupt routines. A system Stack Pointer register, SP, points to the top of the stack. This pointer is decremented when data is pushed onto the stack, and incremented when data is popped.

The internal system stack can also be used to temporarily store data or pass it between subroutines or tasks. Instructions are provided to push or pop registers on/from the system stack. However, in most cases, the register banking scheme provides the best performance for passing data between multiple tasks.

Note: The system stack allows the storage of words only. Bytes must either be converted to words or the respective other byte must be disregarded.
Register SP can be loaded with even byte addresses only (The LSB of SP is always ‘0’).

Detection of stack overflow/underflow is supported by two registers, STKOV (Stack Overflow Pointer) and STKUN (Stack Underflow Pointer). Specific system traps (Stack Overflow trap, Stack Underflow trap) will be entered whenever the SP reaches either boundary specified in these registers.

The contents of the stack pointer are compared to the contents of the overflow register, whenever the SP is DECREMENTED either by a CALL, PUSH, or SUB instruction. An overflow trap will be entered when the SP value is less than the value in the stack overflow register.

The contents of the stack pointer are compared to the contents of the underflow register, whenever the SP is INCREMENTED either by a RET, POP, or ADD instruction. An underflow trap will be entered when the SP value is greater than the value in the stack underflow register.

Note: When a value is MOVED into the stack pointer, NO check against the overflow/ underflow registers is performed.

In many cases, the user will place a software reset instruction (SRST) into the stack underflow and overflow trap service routines. This is an easy approach which does not require special programming. However, this approach assumes that the defined internal stack is sufficient for the current software and that exceeding its upper or lower boundary represents a fatal error.

It is also possible to use the stack underflow and stack overflow traps to cache portions of a larger external stack. Only the portion of the system stack currently being used is placed into the internal memory, thus allowing a greater portion of the internal RAM to be used for program, data, or register banking. This approach assumes no error but requires a set of control routines (see below).

Circular (Virtual) Stack

This basic technique allows pushing until the overflow boundary of the internal stack is reached. At this point, a portion of the stacked data must be saved into external memory to create space for further stack pushes. This is called “stack flushing”. When executing a number of return or pop instructions, the upper boundary is reached (since the stack empties upward to higher memory locations). Entries that have been previously saved in external memory must now be restored. This is called “stack filling”. Because procedure call instructions do not continue to nest infinitely, and call and return instructions alternate, flushing and filling normally occur very infrequently. If this is not true for a specific program environment, this technique should not be used because of the overhead of flushing and filling.

The basic mechanism is the transformation via hardware of the addresses of a virtual stack area, controlled via registers SP, STKOV and STKUN, to a defined physical stack area within the internal RAM. This virtual stack area covers all possible locations to which SP can point, i.e. $00'F000_H$ through $00'FFFE_H$. Registers STKOV and STKUN accept the same 4 KByte address range.
The size of the physical stack area within the internal RAM that is effectively used for standard stack operations is defined via bitfield STKSZ in register SYSCON (see below).

Table 22-2 Circular Stack Address Transformation

STKSZ	Stack Size (Words)	Internal RAM Addresses (Words) of Physical Stack	Significant Bits of Stack Ptr. SP
$0\ 0\ 0_B$	256	$00'FBFE_H$ … $00'FA00_H$ (Default after Reset)	SP.8 … SP.0
$0\ 0\ 1_B$	128	$00'FBFE_H$ … $00'FB00_H$	SP.7 … SP.0
$0\ 1\ 0_B$	64	$00'FBFE_H$ … $00'FB80_H$	SP.6 … SP.0
$0\ 1\ 1_B$	32	$00'FBFE_H$ … $00'FBC0_H$	SP.5 … SP.0
$1\ 0\ 0_B$	512	$00'FBFE_H$ … $00'F800_H$ (not for 1 KByte IRAM)	SP.9 … SP.0
$1\ 0\ 1_B$	–	Reserved. Do not use this combination.	–
$1\ 1\ 0_B$	–	Reserved. Do not use this combination.	–
$1\ 1\ 1_B$	512 / 1024 / 1536	$00'FDFE_H$ … $00'FX00_H$ (Note: No circular stack) $00'FX00_H$ represents the lower IRAM limit, i.e. 1 KB: $00'FA00_H$, 2 KB: $00'F600_H$, 3 KB: $00'F200_H$	SP.11 … SP.0

The virtual stack addresses are transformed to physical stack addresses by concatenating the significant bits of the Stack Pointer register SP (see **Table 22-2**) with the complementary most significant bits of the upper limit of the physical stack area (00'FBFE$_H$). This transformation is done via hardware (see **Figure 22-1**).

The reset values (STKOV = FA00$_H$, STKUN = FC00$_H$, SP = FC00$_H$, STKSZ = 000$_B$) map the virtual stack area directly to the physical stack area and allow use of the internal system stack without any changes, provided that the 256 word area is not exceeded.

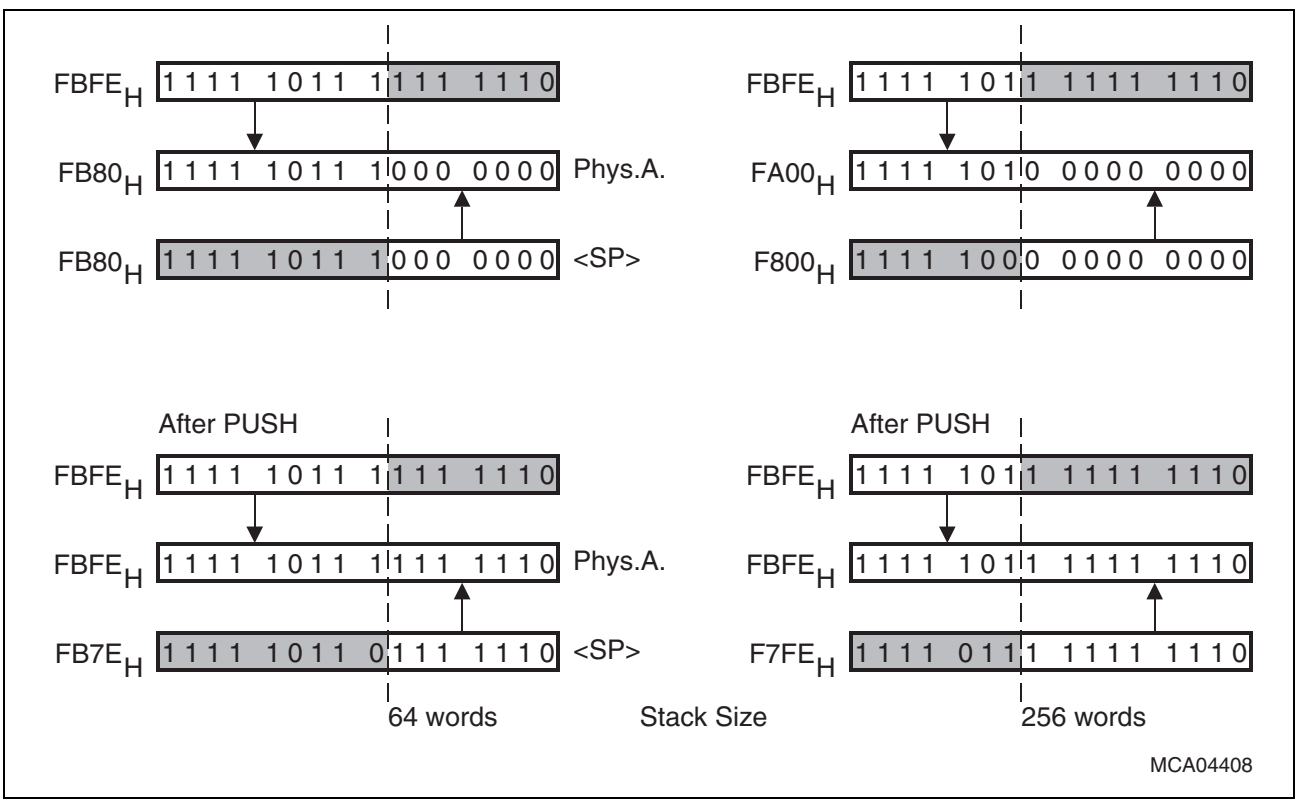


Figure 22-1 Physical Stack Address Generation

The following example demonstrates the circular stack mechanism which is also an effect of this virtual stack mapping: First, register R1 is pushed onto the lowest physical stack location according to the selected maximum stack size. With the following instruction, register R2 will be pushed onto the highest physical stack location although the SP is decremented by 2, as for the previous push operation.

```
MOV     SP, #0F802H ;Set SP before last entry …
                    ;… of physical stack of 256 words
…                   ;(SP)= F802H: Physical stack addr.= FA02H
PUSH    R1          ;(SP)= F800H: Physical stack addr.= FA00H
PUSH    R2          ;(SP)= F7FEH: Physical stack addr.= FBFEH
```

The effect of the address transformation is that the physical stack addresses wrap around from the end of the defined area to its beginning. When flushing and filling the internal stack, this circular stack mechanism requires moving only that portion of stack data which is really to be re-used (the upper part of the defined stack area) instead of the entire stack area. Stack data that remain in the lower part of the internal stack need not be moved by the distance of the space being flushed or filled, as the Stack Pointer automatically wraps around to the beginning of the freed part of the stack area.

Note: This circular stack technique is applicable for stack sizes of 32 to 512 words (STKSZ = '000_B' to '100_B'), it does not work with option STKSZ = '111_B', which uses the complete internal RAM for system stack.
In the latter case, the address transformation mechanism is deactivated.

When a boundary is reached, the stack underflow or overflow trap is entered in which the user moves a predetermined portion of the internal stack to or from the external stack. The amount of data transferred is determined by the average stack space required by routines and the frequency of calls, traps, interrupts, and returns. In most cases, this will be approximately one-quarter to one-tenth the size of the internal stack. After the transfer is complete, the boundary pointers are updated to reflect the newly allocated space on the internal stack. Thus, the user is free to write code without concern for the internal stack limits. Only the execution time required by the trap routines affects user programs.

The following procedure initializes the controller for use of the circular stack mechanism:

- Specify the size of the physical system stack area within the internal RAM (bitfield STKSZ in register SYSCON).
- Define two pointers which specify the upper and lower boundary of the external stack. These values are then tested in the stack underflow and overflow trap routines when moving data.
- Set the stack overflow pointer (STKOV) to the limit of the defined internal stack area plus six words (for the reserved space to store two interrupt entries).

The internal stack will now fill until the overflow pointer is reached. After entry into the overflow trap procedure, the top of the stack will be copied to the external memory. The internal pointers will then be modified to reflect the newly allocated space. After exiting from the trap procedure, the internal stack will wrap around to the top of the internal stack and continue to grow until the new value of the stack overflow pointer is reached.

When the underflow pointer is reached while the stack is emptied, the bottom of stack is reloaded from the external memory and the internal pointers are adjusted accordingly.

Linear Stack

The C164CI also offers a linear stack option (STKSZ = '111_B'), in which the system stack may use the entire internal RAM area. This provides a large system stack without requiring procedures to handle data transfers for a circular stack. However, this method also leaves less RAM space for variables or code. The RAM area that may be effectively consumed by the system stack is defined via the STKUN and STKOV pointers. The underflow and overflow traps in this case serve for fatal error detection only.

For the linear stack option, all modifiable bits of register SP are used to access the physical stack. Although the stack pointer may cover addresses from $00'F000_H$ up to $00'FFFE_H$ the (physical) system stack must be located within the internal RAM and, therefore, may use only the address range $00'F200_H$/$00'F600_H$/$00'FA00_H$ to $00'FDFE_H$. It is the user's responsibility to restrict the system stack to the range of the internal RAM.

Note: Avoid stack accesses below the IRAM area (ESFR space and reserved area) and within address range $00'FE00_H$ and $00'FFFE_H$ (SFR space).
Otherwise unpredictable results will occur.

User Stacks

User stacks provide the ability to create task-specific data stacks and to off-load data from the system stack. The user may push both bytes and words onto a user stack, but is responsible for using the appropriate instructions when popping data from the specific user stack. No hardware detection of overflow or underflow of a user stack is provided. The following addressing modes allow implementation of user stacks:

[–Rw], Rb or [–Rw], Rw: Pre-decrement Indirect Addressing.
Used to push one byte or word onto a user stack. This mode is only available for MOV instructions and can specify any GPR as the user stack pointer.

Rb, [Rw_i+] or Rw, [Rw_i+]: Post-increment Index Register Indirect Addressing.
Used to pop one byte or word from a user stack. This mode is available to most instructions, but only GPRs R0-R3 can be specified as the user stack pointer.

Rb, [Rw+] or Rw, [Rw+]: Post-increment Indirect Addressing.
Used to pop one byte or word from a user stack. This mode is only available for MOV instructions and can specify any GPR as the user stack pointer.

22.2 Register Banking

Register banking provides the user with an extremely fast method for switching user context. A single machine cycle instruction saves the old bank and enters a new register bank. Each register bank may assign up to 16 registers. Each register bank should be allocated during coding based on the needs of each task. After the internal memory has been partitioned into a register bank space, internal stack space, and a global internal memory area, each bank pointer is then assigned. Thus, upon entry into a new task, the appropriate bank pointer is used as the operand for the SCXT (switch context) instruction. Upon exit from a task, a simple POP instruction to the Context Pointer (CP) restores the previous task's register bank.

22.3 Procedure Call Entry and Exit

To support modular programming, a procedure mechanism is provided to allow coding of frequently used portions of code into subroutines. The CALL and RET instructions store and restore the value of the Instruction Pointer (IP) on the system stack before and after a subroutine is executed.

Procedures may be called conditionally with instructions CALLA or CALLI, or may be called unconditionally using instructions CALLR or CALLS.

Note: Any data pushed onto the system stack during execution of the subroutine must be popped before the RET instruction is executed.

Passing Parameters on the System Stack

PUSH instructions may be used to pass parameters via the system stack before the subroutine is called; POP instructions may be used during execution of the subroutine. Base plus offset indirect addressing also permits access to parameters without popping these parameters from the stack during execution of the subroutine. Indirect addressing provides a mechanism for accessing data referenced by data pointers, which are passed to the subroutine.

Additionally, two instructions have been implemented to allow one parameter to be passed on the system stack without additional software overhead.

The PCALL (push and call) instruction first pushes the 'reg' operand and the IP contents onto the system stack and then passes control to the subroutine specified by the 'caddr' operand.

When exiting from the subroutine, the RETP (return and pop) instruction first pops the IP and then the 'reg' operand from the system stack and returns to the calling program.

Cross Segment Subroutine Calls

Calls to subroutines in different segments require the use of the CALLS (call inter-segment subroutine) instruction. This instruction preserves both the CSP (Code Segment Pointer) and the IP on the system stack.

Upon return from the subroutine, a RETS (return from inter-segment subroutine) instruction must be used to restore both the CSP and IP. This ensures that the next instruction after the CALLS instruction is fetched from the correct segment.

Note: It is possible to use CALLS within the same segment, but two words of the stack are still used to store both the IP and CSP.

Providing Local Registers for Subroutines

The following methods are provided for subroutines which require local storage:

- Alternate Banks of Registers
- Saving and Restoring Registers
- Use of the System Stack for Local Registers

Alternate Bank of Registers: Upon entry into a subroutine, it is possible to specify a new set of local registers by executing the SCXT (switch context) instruction. This mechanism does not provide a method to recursively call a subroutine.

Saving and Restoring Registers: To provide local registers, the contents of the registers which are required for use by the subroutine can be pushed onto the stack and the previous values can be popped before returning to the calling routine. This is the most common technique used today and it does provide a mechanism to support recursive procedures. This method, however, requires two machine cycles per register stored on the system stack (one cycle to PUSH the register, and one cycle to POP the register).

Use of the System Stack for Local Registers: It is possible to use the SP and CP to set up local subroutine register frames. This enables subroutines to dynamically allocate local variables as needed within two machine cycles. A local frame is allocated by simply subtracting the number of required local registers from the SP, and then moving the value of the new SP to the CP.

This operation is supported through the SCXT (switch context) instruction with the addressing mode ‘reg, mem’. Using this instruction saves the old contents of the CP on the system stack and moves the value of the SP into CP (see the example below). Each local register is then accessed as if it were a normal register. Upon exit from the subroutine, first the old CP must be restored by popping it from the stack and then the number of used local registers must be added to the SP to restore the allocated local space back to the system stack.

Note: The system stack grows downward, while the register bank grows upward.

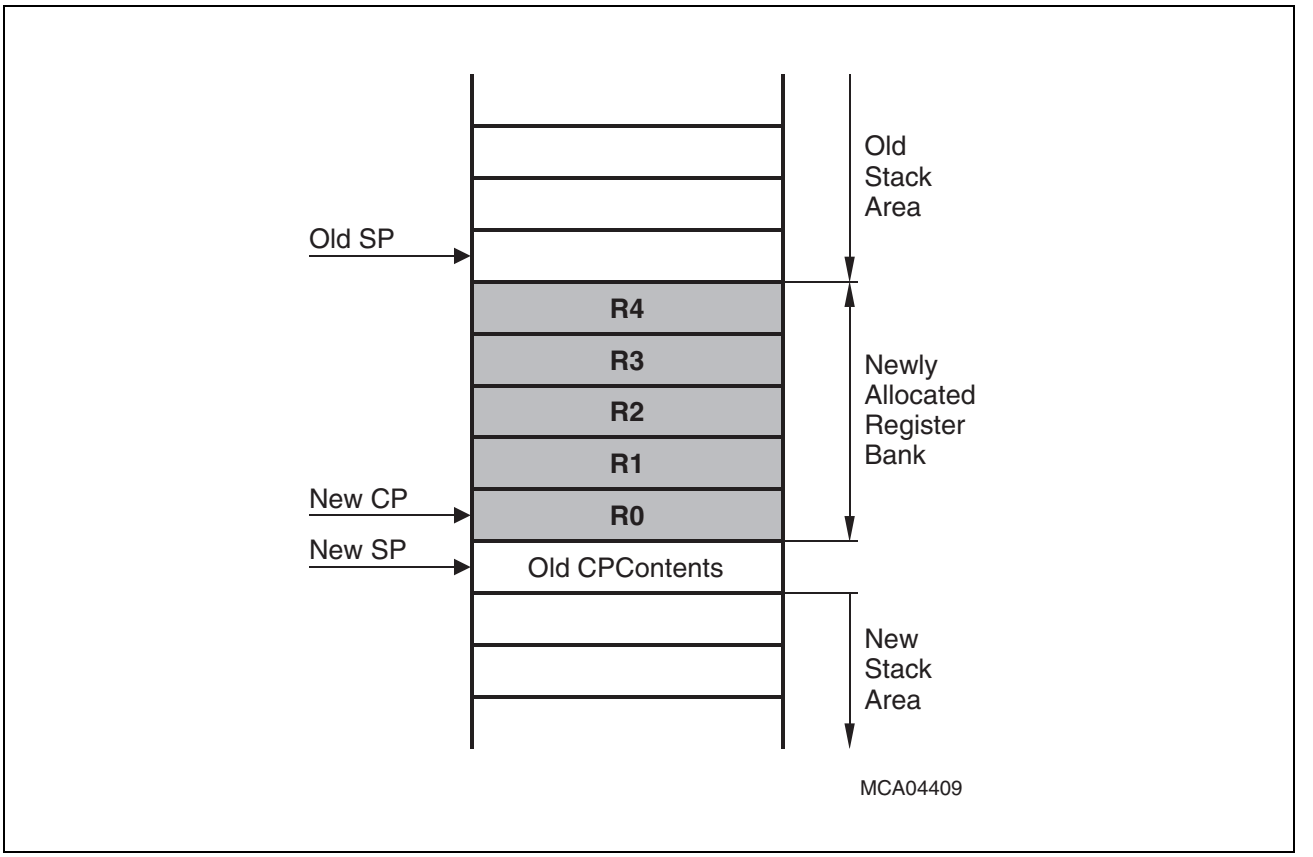


Figure 22-2 Local Registers

The software to provide the local register bank for the example shown in **Figure 22-2** is very compact:

After entering the subroutine:

```
SUB     SP, #10D    ;Free 5 words in the current system stack
SCXT    CP, SP      ;Set the new register bank pointer
```

Before exiting the subroutine:

```
POP     CP          ;Restore the old register bank
ADD     SP, #10D    ;Release the 5 words …
                    ;… of the current system stack
```

22.4 Table Searching

A number of features have been included to decrease the execution time required to search tables. First, branch delays are eliminated by the branch target cache after the first iteration of the loop. Second, in non-sequentially searched tables, the enhanced performance of the ALU allows more complicated hash algorithms to be processed to obtain better table distribution. For sequentially searched tables, the auto-increment indirect addressing mode and the E (end of table) flag stored in the PSW decrease the number of overhead instructions executed in the loop.

The two examples below illustrate searching ordered tables and non-ordered tables, respectively:

```
MOV      R0, #BASE    ;Move table base into R0
LOOP:
CMP      R1, [R0+]    ;Compare target to table entry
JMPR     cc_SGT, LOOP;Test whether target has not been found
```

Note: The last entry in the table must be greater than the largest possible target.

```
MOV      R0, #BASE    ;Move table base into R0
LOOP:
CMP      R1, [R0+]    ;Compare target to table entry
JMPR     cc_NET, LOOP;Test whether target is not found AND …
                      ;… the end of table has not been reached.
```

Note: The last entry in the table must be equal to the lowest signed integer (8000_H).

22.5 Floating Point Support

All floating point operations are performed using software. Standard multiple precision instructions are used to perform calculations on data types that exceed the size of the ALU. Multiple bit rotate and logic instructions allow easy masking and extracting of portions of floating point numbers.

To decrease the time required to perform floating point operations, two hardware features have been implemented in the CPU core. First, the PRIOR instruction aids in normalizing floating point numbers by indicating the position of the first set bit in a GPR. This result can then be used to rotate the floating point result accordingly. The second feature assists in properly rounding the result of normalized floating point numbers through the overflow (V) flag in the PSW. This flag is set when a one is shifted out of the carry bit during shift right operations. The overflow flag and the carry flag are then used to round the floating point result based on the desired rounding algorithm.

22.6 Peripheral Control and Interface

All communication between peripherals and the CPU is performed by either PEC transfers to and from internal memory or by explicit addressing of the SFRs associated with the specific peripherals. After resetting the C164CI all peripherals (except the watchdog timer) are disabled and initialized to default values. A desired configuration of a specific peripheral is programmed using MOV instructions of either constants or memory values to specific SFRs. Specific control flags may also be altered via bit instructions.

Once in operation, the peripheral operates autonomously until an end condition is reached; at which time, it requests a PEC transfer or requests CPU servicing through an interrupt routine. Information may also be polled from peripherals through read accesses to SFRs or bit operations including branch tests on specific control bits in SFRs. To ensure proper allocation of peripherals among multiple tasks, a portion of the internal memory has been made bit-addressable to allow user semaphores. Instructions have also been provided to lock out tasks via software by setting or clearing user-specific bits and conditionally branching based on these specific bits.

It is recommended that bit fields in control SFRs be updated using the BFLDH and BFLDL instructions or a MOV instruction to avoid undesired intermediate modes of operation which can occur if BCLR/BSET or AND/OR instruction sequences are used.

22.7 Trap/Interrupt Entry and Exit

Interrupt routines are entered when a requesting interrupt has a priority higher than the current CPU priority level. Traps are entered regardless of the current CPU priority. When either a trap or interrupt routine is entered, the state of the machine is preserved on the system stack and a branch to the appropriate trap/interrupt vector is made.

All trap and interrupt routines require the use of the RETI (return from interrupt) instruction to exit from the called routine. This instruction restores the system state from the system stack and then branches back to the location at which the trap or interrupt occurred.

22.8 Inseparable Instruction Sequences

The instructions of the C164CI are very efficient (most instructions execute in one machine cycle). Even the multiplication and division are interruptible in order to minimize the response latency to interrupt requests (internal and external). This is vital in many microcontroller applications.

Some special occasions, however, require certain code sequences (such as semaphore handling) to be executed uninterruptedly to function properly. This can be accomplished by inhibiting interrupts during the respective code sequence by disabling and enabling them before and after the sequence. The necessary overhead may be reduced by means of the ATOMIC instruction which allows locking 1 … 4 instructions to an inseparable code sequence, during which the interrupt system (standard interrupts and PEC requests) **and Class A Traps** ($\overline{\text{NMI}}$, stack overflow/underflow) are disabled. **Class B Traps** (illegal opcode, illegal bus access, etc.), however, will interrupt the atomic sequence, since it indicates a severe hardware problem.

The interrupt inhibit caused by an ATOMIC instruction gets active immediately; no other instruction will enter the pipeline except the one following the ATOMIC instruction, and no interrupt request will be serviced in between. All instructions requiring multiple cycles or hold states are regarded as one instruction in this case (for example, MUL is one instruction). Any instruction type can be used within an inseparable code sequence.

```
ATOMIC   #3          ;The next 3 instr. are locked (No NOP requ.)
MOV      R0, #1234H  ;Instr. 1 (no other instr. enters pipeline!)
MOV      R1, #5678H  ;Instr. 2
MUL      R0, R1      ;Instr. 3: MUL regarded as one instruction
MOV      R2, MDL     ;This instruction is out of the scope …
                     ;… of the ATOMIC instruction sequence
```

Note: As long as any Class B trap is pending (any of the class B trap flags in register TFR is set) the ATOMIC instruction will not work. Clear the respective B trap flag at the beginning of a B trap routine if ATOMIC shall be used within the routine.

22.9 Overriding the DPP Addressing Mechanism

The standard mechanism for accessing data locations uses one of the four data page pointers (DPPx), which selects a 16-KByte data page, and a 14-bit offset within this data page. The four DPPs allow immediate access to up to 64 KBytes of data. In applications with large data arrays, especially in HLL applications using large memory models, this may require frequent reloading of the DPPs, even for single accesses.

The EXTP (extend page) instruction allows switching to an arbitrary data page for 1 … 4 instructions without changing the current DPPs.

```
EXTP    R15, #1     ;The override page number is stored in R15
MOV     R0, [R14]   ;The (14-bit) page offset is stored in R14
MOV     R1, [R13]   ;This instruction uses the std. DPP scheme!
```

The EXTS (extend segment) instruction allows switching to a 64 KByte segment oriented data access scheme for 1 … 4 instructions without changing the current DPPs. In this case all 16 bits of the operand address are used as segment offset, with the segment taken from the EXTS instruction. This greatly simplifies address calculation with continuous data, such as huge arrays in “C”.

```
EXTS    #15, #1     ;The override seg. is 15 (0F'0000H..0F'FFFFH)
MOV     R0, [R14]   ;The (16-bit) segment offset is stored in R14
MOV     R1, [R13]   ;This instruction uses the std. DPP scheme!
```

Note: Instructions EXTP and EXTS inhibit interrupts the same way as ATOMIC. As long as any Class B trap is pending (any of the class B trap flags in register TFR is set) the EXTend instructions will not work. Clear the respective B trap flag at the beginning of a B trap routine if EXT shall be used within the routine.*

Short Addressing in Extended SFR (ESFR) Space

The short addressing modes of the C164CI (REG or BITOFF) implicitly access the SFR space. The additional ESFR space would need to be accessed via long addressing modes (MEM or [Rw]). The EXTR (extend register) instruction redirects accesses in short addressing modes to the ESFR space for 1 … 4 instructions, so the additional registers can be accessed this way, too.

The EXTPR and EXTSR instructions combine the DPP override mechanism with the redirection to the ESFR space using a single instruction.

Note: Instructions EXTR, EXTPR, and EXTSR inhibit interrupts the same way as ATOMIC instructions. Switching to the ESFR area and data page overriding are checked by the development tools or are handled automatically.

Nested Locked Sequences

Each described extension instruction and the ATOMIC instruction start an internal “extension counter” counting the effected instructions. When another extension or ATOMIC instruction is contained in the current locked sequence, this counter is restarted with the value of the new instruction. This allows construction of locked sequences longer than 4 instructions.

Note: Interrupt latencies may be increased when using locked code sequences. PEC requests are not serviced during idle mode, if the IDLE instruction is part of a locked sequence.

22.10 Handling the Internal Code Memory

The Mask-ROM/OTP/Flash versions of the C164CI provide on-chip code memory that may store code as well as data. The lower 32 KBytes of this code memory are referred to as the “internal ROM area”. Access to this internal ROM area is controlled during the reset configuration and via software. The ROM area may be mapped to segment 0, to segment 1, or the code memory may be disabled.

Note: The internal ROM area always occupies an address area of 32 KBytes, even if the implemented mask ROM/OTP/Flash memory is smaller than that (such as 8 KBytes).
Of course, the total implemented memory may exceed 32 KBytes.

Code Memory Configuration During Reset

The control input pin $\overline{EA}$ (External Access) enables the user to define the address area from which the first instructions after reset are fetched. When $\overline{EA}$ is low (‘0’) during reset, the internal code memory is disabled and the first instructions are fetched from external memory. When $\overline{EA}$ is high (‘1’) during reset, the internal code memory is globally enabled and the first instructions are fetched from the internal memory.

Note: Be sure not to select internal memory access after reset on ROMless devices.

Mapping the Internal ROM Area

After reset, the internal ROM area is mapped into segment 0, the “system segment” ($00'0000_H$ … $00'7FFF_H$) as a default. This is necessary to allow the first instructions to be fetched from locations $00'0000_H$ ff. The ROM area may be mapped to segment 1 ($01'0000_H$ … $01'7FFF_H$) by setting bit ROMS1 in register SYSCON. The internal ROM area may now be accessed through the lower half of segment 1, while accesses to segment 0 will now be made to external memory. This adds flexibility to the system software. The interrupt/trap vector table, which uses locations $00'0000_H$ through $00'01FF_H$, is now part of the external memory and may therefore be modified; that is to say, the system software may now change interrupt/trap handlers according to the current condition of the system. The internal code memory can still be used for fixed software routines such as IO drivers, math libraries, application specific invariant routines, tables, etc. This combines the advantage of an integrated non-volatile memory with the advantage of a flexible, adaptable software system.

Enabling and Disabling Internal Code Memory After Reset

If the internal code memory does not contain an appropriate startup code, the system may be booted from external memory, while the internal memory is enabled afterwards to provide access to library routines, tables, etc.

If the internal code memory contains only the startup code and/or test software, the system may be booted from internal memory, which may then be disabled, after the software has switched to executing from external memory (for example). This may be done to free the address space occupied by the internal code memory which would no longer be necessary.

22.11 Pits, Traps, and Mines

Although handling the internal code memory provides a powerful means of enhancing the overall performance and flexibility of a system, extreme care must be taken to avoid a system crash. Instruction memory is the most crucial resource for the C164CI and it must be ensured that it never runs out. The following precautions help to take advantage of the methods mentioned above without jeopardizing system security.

Internal code memory access after reset: When the first instructions are to be fetched from internal memory ($\overline{EA}$ = '1'), the device must contain code memory containing a valid reset vector and valid code at its destination.

Mapping the internal ROM area to segment 1: Due to instruction pipelining, any new ROM mapping will at the earliest become valid for the second instruction after the instruction which has changed the ROM mapping. To enable accesses to the ROM area after mapping a branch to the newly selected ROM area (JMPS) and reloading of all data page pointers is required.
This also applies to re-mapping the internal ROM area to segment 0.

Enabling the internal code memory after reset: When enabling the internal code memory after having booted the system from external memory, note that the C164CI will then access the internal memory using the current segment offset, rather than accessing external memory.

Disabling the internal code memory after reset: When disabling the internal code memory after having booted the system from there, note that the C164CI will not access external memory before a jump to segment 0 (in this case) is executed.

General Rules

When mapping the code memory no instruction or data accesses should be made to the internal memory, otherwise unpredictable results may occur.

To avoid these problems, the instructions which configure the internal code memory should be executed from external memory or from the on-chip RAM.

Whenever the internal code memory is disabled, enabled, or remapped, the DPPs must be explicitly (re)loaded to enable correct data accesses to the internal and/or external memory.

23 Register Set

This chapter summarizes all registers implemented in the C164CI and explains the description format used in the chapters which describe the functions and layout of the Special Function Registers (SFRs).
For easy reference, the registers are ordered according to two different keys (except for GPRs):

- Ordered by address, to identify which register a given address references,
- Ordered by register name, to find the location of a specific register.

23.1 Register Description Format

In their respective chapters, the functions and the layout of the SFRs are described in a specific format which provides various details about each special function register. The example below shows how to interpret these details.

REG_NAME
Name of Register **E/SFR *($A16_H$/$A8_H$)*** **Reset Value: $****_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<empty for byte registers>								**std**	**hw**	**read/ write bit**	**read bit**	**write bit**	**bitfield**		
-	-	-	-	-	-	-	-	rw	rwh	rw	r	w	rw		

Bit	Function
bit(field)name	Explanation of bit(field)name *Description of the functions controlled by the different possible values of this bit(field).*

Elements:

REG_NAME	Short name of this register
A16/A8	Long 16-bit address / Short 8-bit address
SFR/**ESFR**/**XReg**	Register space (SFR, ESFR or External/XBUS Register)
(**) **	Register contents after reset **0/1**: defined value, '**X**': undefined, '**U**': unchanged (undefined ('**X**') after power up)
r/w	Access modes: can be read and/or write
*h	Bits that are set/cleared by hardware are marked with a shaded access box and an '**h**' in it.

23.2 CPU General Purpose Registers (GPRs)

The General Purpose Registers (GPRs) form the register bank with which the CPU works. This register bank may be located anywhere within the internal RAM via the Context Pointer (CP). Due to the addressing mechanism, GPR banks can reside only within the internal RAM. All GPRs are bit-addressable.

Table 23-1 General Purpose Word Registers

Name	Physical Address	8-bit Address	Description	Reset Value
R0	(CP) + 0	$F0_H$	CPU General Purpose (Word) Reg. R0	$UUUU_H$
R1	(CP) + 2	$F1_H$	CPU General Purpose (Word) Reg. R1	$UUUU_H$
R2	(CP) + 4	$F2_H$	CPU General Purpose (Word) Reg. R2	$UUUU_H$
R3	(CP) + 6	$F3_H$	CPU General Purpose (Word) Reg. R3	$UUUU_H$
R4	(CP) + 8	$F4_H$	CPU General Purpose (Word) Reg. R4	$UUUU_H$
R5	(CP) + 10	$F5_H$	CPU General Purpose (Word) Reg. R5	$UUUU_H$
R6	(CP) + 12	$F6_H$	CPU General Purpose (Word) Reg. R6	$UUUU_H$
R7	(CP) + 14	$F7_H$	CPU General Purpose (Word) Reg. R7	$UUUU_H$
R8	(CP) + 16	$F8_H$	CPU General Purpose (Word) Reg. R8	$UUUU_H$
R9	(CP) + 18	$F9_H$	CPU General Purpose (Word) Reg. R9	$UUUU_H$
R10	(CP) + 20	FA_H	CPU General Purpose (Word) Reg. R10	$UUUU_H$
R11	(CP) + 22	FB_H	CPU General Purpose (Word) Reg. R11	$UUUU_H$
R12	(CP) + 24	FC_H	CPU General Purpose (Word) Reg. R12	$UUUU_H$
R13	(CP) + 26	FD_H	CPU General Purpose (Word) Reg. R13	$UUUU_H$
R14	(CP) + 28	FE_H	CPU General Purpose (Word) Reg. R14	$UUUU_H$
R15	(CP) + 30	FF_H	CPU General Purpose (Word) Reg. R15	$UUUU_H$

The first 8 GPRs (R7 … R0) may also be accessed bytewise. Other than with SFRs, writing to a GPR byte does not affect the other byte of the respective GPR.
The respective halves of the byte-accessible registers receive special names:

Table 23-2 General Purpose Byte Registers

Name	Physical Address	8-bit Address	Description	Reset Value
RL0	(CP) + 0	$F0_H$	CPU General Purpose (Byte) Reg. RL0	UU_H
RH0	(CP) + 1	$F1_H$	CPU General Purpose (Byte) Reg. RH0	UU_H
RL1	(CP) + 2	$F2_H$	CPU General Purpose (Byte) Reg. RL1	UU_H
RH1	(CP) + 3	$F3_H$	CPU General Purpose (Byte) Reg. RH1	UU_H
RL2	(CP) + 4	$F4_H$	CPU General Purpose (Byte) Reg. RL2	UU_H
RH2	(CP) + 5	$F5_H$	CPU General Purpose (Byte) Reg. RH2	UU_H
RL3	(CP) + 6	$F6_H$	CPU General Purpose (Byte) Reg. RL3	UU_H
RH3	(CP) + 7	$F7_H$	CPU General Purpose (Byte) Reg. RH3	UU_H
RL4	(CP) + 8	$F8_H$	CPU General Purpose (Byte) Reg. RL4	UU_H
RH4	(CP) + 9	$F9_H$	CPU General Purpose (Byte) Reg. RH4	UU_H
RL5	(CP) + 10	FA_H	CPU General Purpose (Byte) Reg. RL5	UU_H
RH5	(CP) + 11	FB_H	CPU General Purpose (Byte) Reg. RH5	UU_H
RL6	(CP) + 12	FC_H	CPU General Purpose (Byte) Reg. RL6	UU_H
RH6	(CP) + 13	FD_H	CPU General Purpose (Byte) Reg. RH6	UU_H
RL7	(CP) + 14	FE_H	CPU General Purpose (Byte) Reg. RL7	UU_H
RH7	(CP) + 15	FF_H	CPU General Purpose (Byte) Reg. RH7	UU_H

23.3 Registers Ordered by Name

Table 23-3 lists all registers implemented in the C164CI in alphabetical order. The following markings assist in classifying the listed registers:
"**b**" in the "Name" column marks **Bit-addressable** SFRs.
"**E**" in the "Physical Address" column marks (**E**)SFRs in the **Extended SFR-Space**.
"**m**" in the "Physical Address" column marks SFRs without short 8-bit address.
"**X**" in the "Physical Address" column marks registers within on-chip **X-Peripherals**.

Table 23-3 C164CI Registers Ordered by Name

Name	Physical Address	8-Bit Addr.	Description	Reset Value
ADCIC b	$FF98_H$	CC_H	A/D Converter End of Conversion Interrupt Control Register	0000_H
ADCON b	$FFA0_H$	$D0_H$	A/D Converter Control Register	0000_H
ADDAT	$FEA0_H$	50_H	A/D Converter Result Register	0000_H
ADDAT2	$F0A0_H$ **E**	50_H	A/D Converter 2 Result Register	0000_H
ADDRSEL1	$FE18_H$	$0C_H$	Address Select Register 1	0000_H
ADDRSEL2	$FE1A_H$	$0D_H$	Address Select Register 2	0000_H
ADDRSEL3	$FE1C_H$	$0E_H$	Address Select Register 3	0000_H
ADDRSEL4	$FE1E_H$	$0F_H$	Address Select Register 4	0000_H
ADEIC b	$FF9A_H$	CD_H	A/D Converter Overrun Error Interrupt Control Register	0000_H
BUSCON0 b	$FF0C_H$	86_H	Bus Configuration Register 0	0000_H
BUSCON1 b	$FF14_H$	$8A_H$	Bus Configuration Register 1	0000_H
BUSCON2 b	$FF16_H$	$8B_H$	Bus Configuration Register 2	0000_H
BUSCON3 b	$FF18_H$	$8C_H$	Bus Configuration Register 3	0000_H
BUSCON4 b	$FF1A_H$	$8D_H$	Bus Configuration Register 4	0000_H
C1BTR	$EF04_H$ **X**	–	CAN1 Bit Timing Register	$UUUU_H$
C1CSR	$EF00_H$ **X**	–	CAN1 Control / Status Register	$XX01_H$
C1GMS	$EF06_H$ **X**	–	CAN1 Global Mask Short	$UFUU_H$
C1LARn	$EFn4_H$ **X**	–	CAN Lower Arbitration Register (msg. **n**)	$UUUU_H$
C1LGML	$EF0A_H$ **X**	–	CAN Lower Global Mask Long	$UUUU_H$
C1LMLM	$EF0E_H$ **X**	–	CAN Lower Mask of Last Message	$UUUU_H$
C1MCFGn	$EFn6_H$ **X**	–	CAN Message Configuration Register (msg. **n**)	UU_H

Table 23-3 C164CI Registers Ordered by Name (cont'd)

Name	Physical Address	8-Bit Addr.	Description	Reset Value
C1MCRn	$EFn0_H$ **X**	–	CAN Message Control Register (msg. **n**)	$UUUU_H$
C1PCIR	$EF02_H$ **X**	–	CAN1 Port Control / Interrupt Register	$XXXX_H$
C1UARn	$EFn2_H$ **X**	–	CAN Upper Arbitration Register (msg. **n**)	$UUUU_H$
C1UGML	$EF08_H$ **X**	–	CAN Upper Global Mask Long	$UUUU_H$
C1UMLM	$EF0C_H$ **X**	–	CAN Upper Mask of Last Message	$UUUU_H$
CC10IC b	$FF8C_H$	$C6_H$	External Interrupt 2 Control Register	0000_H
CC11IC b	$FF8E_H$	$C7_H$	External Interrupt 3 Control Register	0000_H
CC16	$FE60_H$	30_H	CAPCOM Register 16	0000_H
CC16IC b	$F160_H$ **E**	$B0_H$	CAPCOM Reg. 16 Interrupt Ctrl. Reg.	0000_H
CC17	$FE62_H$	31_H	CAPCOM Register 17	0000_H
CC17IC b	$F162_H$ **E**	$B1_H$	CAPCOM Reg. 17 Interrupt Ctrl. Reg.	0000_H
CC18	$FE64_H$	32_H	CAPCOM Register 18	0000_H
CC18IC b	$F164_H$ **E**	$B2_H$	CAPCOM Reg. 18 Interrupt Ctrl. Reg.	0000_H
CC19	$FE66_H$	33_H	CAPCOM Register 19	0000_H
CC19IC b	$F166_H$ **E**	$B3_H$	CAPCOM Reg. 19 Interrupt Ctrl. Reg.	0000_H
CC20	$FE68_H$	34_H	CAPCOM Register 20	0000_H
CC21	$FE6A_H$	35_H	CAPCOM Register 21	0000_H
CC22	$FE6C_H$	36_H	CAPCOM Register 22	0000_H
CC23	$FE6E_H$	37_H	CAPCOM Register 23	0000_H
CC24	$FE70_H$	38_H	CAPCOM Register 24	0000_H
CC24IC b	$F170_H$ **E**	$B8_H$	CAPCOM Reg. 24 Interrupt Ctrl. Reg.	0000_H
CC25	$FE72_H$	39_H	CAPCOM Register 25	0000_H
CC25IC b	$F172_H$ **E**	$B9_H$	CAPCOM Reg. 25 Interrupt Ctrl. Reg.	0000_H
CC26	$FE74_H$	$3A_H$	CAPCOM Register 26	0000_H
CC26IC b	$F174_H$ **E**	BA_H	CAPCOM Reg. 26 Interrupt Ctrl. Reg.	0000_H
CC27	$FE76_H$	$3B_H$	CAPCOM Register 27	0000_H
CC27IC b	$F176_H$ **E**	BB_H	CAPCOM Reg. 27 Interrupt Ctrl. Reg.	0000_H
CC28	$FE78_H$	$3C_H$	CAPCOM Register 28	0000_H
CC29	$FE7A_H$	$3D_H$	CAPCOM Register 29	0000_H
CC30	$FE7C_H$	$3E_H$	CAPCOM Register 30	0000_H

Table 23-3 C164CI Registers Ordered by Name (cont'd)

Name	Physical Address	8-Bit Addr.	Description	Reset Value
CC31	$FE7E_H$	$3F_H$	CAPCOM Register 31	0000_H
CC60	$FE30_H$	18_H	CAPCOM 6 Register 0	0000_H
CC61	$FE32_H$	19_H	CAPCOM 6 Register 1	0000_H
CC62	$FE34_H$	$1A_H$	CAPCOM 6 Register 2	0000_H
CC6EIC **b**	$F188_H$ **E**	$C4_H$	CAPCOM 6 Emergency Interrupt Control Register	0000_H
CC6CIC **b**	$F17E_H$ **E**	BF_H	CAPCOM 6 Interrupt Control Register	0000_H
CC6MCON **b**	$FF32_H$	99_H	CAPCOM 6 Mode Control Register	$00FF_H$
CC6MIC **b**	$FF36_H$	$9B_H$	CAPCOM 6 Mode Interrupt Ctrl. Reg.	0000_H
CC6MSEL	$F036_H$ **E**	$1B_H$	CAPCOM 6 Mode Select Register	0000_H
CC8IC **b**	$FF88_H$	$C4_H$	External Interrupt 0 Control Register	0000_H
CC9IC **b**	$FF8A_H$	$C5_H$	External Interrupt 1 Control Register	0000_H
CCM4 **b**	$FF22_H$	91_H	CAPCOM Mode Control Register 4	0000_H
CCM5 **b**	$FF24_H$	92_H	CAPCOM Mode Control Register 5	0000_H
CCM6 **b**	$FF26_H$	93_H	CAPCOM Mode Control Register 6	0000_H
CCM7 **b**	$FF28_H$	94_H	CAPCOM Mode Control Register 7	0000_H
CMP13	$FE36_H$	$1B_H$	CAPCOM 6 Timer 13 Compare Reg.	0000_H
CP	$FE10_H$	08_H	CPU Context Pointer Register	$FC00_H$
CSP	$FE08_H$	04_H	CPU Code Segment Pointer Register (8 bits, not directly writable)	0000_H
CTCON **b**	$FF30_H$	98_H	CAPCOM 6 Compare Timer Ctrl. Reg.	1010_H
DP0H **b**	$F102_H$ **E**	81_H	P0H Direction Control Register	00_H
DP0L **b**	$F100_H$ **E**	80_H	P0L Direction Control Register	00_H
DP1H **b**	$F106_H$ **E**	83_H	P1H Direction Control Register	00_H
DP1L **b**	$F104_H$ **E**	82_H	P1L Direction Control Register	00_H
DP3 **b**	$FFC6_H$	$E3_H$	Port 3 Direction Control Register	0000_H
DP4 **b**	$FFCA_H$	$E5_H$	Port 4 Direction Control Register	00_H
DP8 **b**	$FFD6_H$	EB_H	Port 8 Direction Control Register	00_H
DPP0	$FE00_H$	00_H	CPU Data Page Pointer 0 Reg. (10 bits)	0000_H
DPP1	$FE02_H$	01_H	CPU Data Page Pointer 1 Reg. (10 bits)	0001_H

Table 23-3 C164CI Registers Ordered by Name (cont'd)

Name		Physical Address		8-Bit Addr.	Description	Reset Value
DPP2		$FE04_H$		02_H	CPU Data Page Pointer 2 Reg. (10 bits)	0002_H
DPP3		$FE06_H$		03_H	CPU Data Page Pointer 3 Reg. (10 bits)	0003_H
EXICON	**b**	$F1C0_H$	**E**	$E0_H$	External Interrupt Control Register	0000_H
EXISEL	**b**	$F1DA_H$	**E**	ED_H	External Interrupt Source Select Reg.	0000_H
FOCON	**b**	$FFAA_H$		$D5_H$	Frequency Output Control Register	0000_H
IDCHIP		$F07C_H$	**E**	$3E_H$	Identifier	$XXXX_H$
IDMANUF		$F07E_H$	**E**	$3F_H$	Identifier	1820_H
IDMEM		$F07A_H$	**E**	$3D_H$	Identifier	$XXXX_H$
IDPROG		$F078_H$	**E**	$3C_H$	Identifier	$XXXX_H$
IDMEM2		$F076_H$	**E**	$3B_H$	Identifier	$XXXX_H$
ISNC	**b**	$F1DE_H$	**E**	EF_H	Interrupt Subnode Control Register	0000_H
MDC	**b**	$FF0E_H$		87_H	CPU Multiply Divide Control Register	0000_H
MDH		$FE0C_H$		06_H	CPU Multiply Divide Reg. – High Word	0000_H
MDL		$FE0E_H$		07_H	CPU Multiply Divide Reg. – Low Word	0000_H
ODP3	**b**	$F1C6_H$	**E**	$E3_H$	Port 3 Open Drain Control Register	0000_H
ODP4	**b**	$F1CA_H$	**E**	$E5_H$	Port 4 Open Drain Control Register	00_H
ODP8	**b**	$F1D6_H$	**E**	EB_H	Port 8 Open Drain Control Register	00_H
ONES	**b**	$FF1E_H$		$8F_H$	Constant Value 1's Register (read only)	$FFFF_H$
OPAD		$EDC2_H$	**X**	---	OTP Progr. Interface Address Register	0000_H
OPCTRL		$EDC0_H$	**X**	---	OTP Progr. Interface Control Register	0007_H
OPDAT		$EDC4_H$	**X**	---	OTP Progr. Interface Data Register	0000_H
P0H	**b**	$FF02_H$		81_H	Port 0 High Reg. (Upper half of PORT0)	00_H
P0L	**b**	$FF00_H$		80_H	Port 0 Low Reg. (Lower half of PORT0)	00_H
P1H	**b**	$FF06_H$		83_H	Port 1 High Reg. (Upper half of PORT1)	00_H
P1L	**b**	$FF04_H$		82_H	Port 1 Low Reg. (Lower half of PORT1)	00_H
P3	**b**	$FFC4_H$		$E2_H$	Port 3 Register	0000_H
P4	**b**	$FFC8_H$		$E4_H$	Port 4 Register (7 bits)	00_H
P5	**b**	$FFA2_H$		$D1_H$	Port 5 Register (read only)	$XXXX_H$
P5DIDIS	**b**	$FFA4_H$		$D2_H$	Port 5 Digital Input Disable Register	0000_H
P8	**b**	$FFD4_H$		EA_H	Port 8 Register (8 bits)	00_H

Table 23-3 C164CI Registers Ordered by Name (cont'd)

Name	Physical Address	8-Bit Addr.	Description	Reset Value
PECC0	$FEC0_H$	60_H	PEC Channel 0 Control Register	0000_H
PECC1	$FEC2_H$	61_H	PEC Channel 1 Control Register	0000_H
PECC2	$FEC4_H$	62_H	PEC Channel 2 Control Register	0000_H
PECC3	$FEC6_H$	63_H	PEC Channel 3 Control Register	0000_H
PECC4	$FEC8_H$	64_H	PEC Channel 4 Control Register	0000_H
PECC5	$FECA_H$	65_H	PEC Channel 5 Control Register	0000_H
PECC6	$FECC_H$	66_H	PEC Channel 6 Control Register	0000_H
PECC7	$FECE_H$	67_H	PEC Channel 7 Control Register	0000_H
PICON b	$F1C4_H$ **E**	$E2_H$	Port Input Threshold Control Register	0000_H
POCON0H	$F082_H$ **E**	41_H	Port P0H Output Control Register	0000_H
POCON0L	$F080_H$ **E**	40_H	Port P0L Output Control Register	0000_H
POCON1H	$F086_H$ **E**	43_H	Port P1H Output Control Register	0000_H
POCON1L	$F084_H$ **E**	42_H	Port P1L Output Control Register	0000_H
POCON20	$F0AA_H$ **E**	55_H	Dedicated Pin Output Control Register	0000_H
POCON3	$F08A_H$ **E**	45_H	Port P3 Output Control Register	0000_H
POCON4	$F08C_H$ **E**	46_H	Port P4 Output Control Register	0000_H
POCON8	$F092_H$ **E**	49_H	Port P8 Output Control Register	0000_H
PSW b	$FF10_H$	88_H	CPU Program Status Word	0000_H
PTCR	$F0AE_H$ **E**	57_H	Port Temperature Compensation Reg.	0000_H
RP0H b	$F108_H$ **E**	84_H	System Startup Config. Reg. (Rd. only)	XX_H
RSTCON b	$F1E0_H$ **m**	–	Reset Control Register	$00XX_H$
RTCH	$F0D6_H$ **E**	$6B_H$	RTC High Register	no
RTCL	$F0D4_H$ **E**	$6A_H$	RTC Low Register	no
S0BG	$FEB4_H$	$5A_H$	Serial Channel 0 Baud Rate Generator Reload Register	0000_H
S0CON b	$FFB0_H$	$D8_H$	Serial Channel 0 Control Register	0000_H
S0EIC b	$FF70_H$	$B8_H$	Serial Channel 0 Error Interrupt Ctrl. Reg.	0000_H
S0RBUF	$FEB2_H$	59_H	Serial Channel 0 Receive Buffer Reg. (read only)	$XXXX_H$

Table 23-3 C164CI Registers Ordered by Name (cont'd)

Name		Physical Address		8-Bit Addr.	Description	Reset Value
S0RIC	**b**	$FF6E_H$		$B7_H$	Serial Channel 0 Receive Interrupt Control Register	0000_H
S0TBIC	**b**	$F19C_H$	**E**	CE_H	Serial Channel 0 Transmit Buffer Interrupt Control Register	0000_H
S0TBUF		$FEB0_H$		58_H	Serial Channel 0 Transmit Buffer Reg. (write only)	0000_H
S0TIC	**b**	$FF6C_H$		$B6_H$	Serial Channel 0 Transmit Interrupt Control Register	0000_H
SP		$FE12_H$		09_H	CPU System Stack Pointer Register	$FC00_H$
SSCBR		$F0B4_H$	**E**	$5A_H$	SSC Baudrate Register	0000_H
SSCCON	**b**	$FFB2_H$		$D9_H$	SSC Control Register	0000_H
SSCEIC	**b**	$FF76_H$		BB_H	SSC Error Interrupt Control Register	0000_H
SSCRB		$F0B2_H$	**E**	59_H	SSC Receive Buffer	$XXXX_H$
SSCRIC	**b**	$FF74_H$		BA_H	SSC Receive Interrupt Control Register	0000_H
SSCTB		$F0B0_H$	**E**	58_H	SSC Transmit Buffer	0000_H
SSCTIC	**b**	$FF72_H$		$B9_H$	SSC Transmit Interrupt Control Register	0000_H
STKOV		$FE14_H$		$0A_H$	CPU Stack Overflow Pointer Register	$FA00_H$
STKUN		$FE16_H$		$0B_H$	CPU Stack Underflow Pointer Register	$FC00_H$
SYSCON	**b**	$FF12_H$		89_H	CPU System Configuration Register	[1] $0xx0_H$
SYSCON1	**b**	$F1DC_H$	**E**	EE_H	CPU System Configuration Register 1	0000_H
SYSCON2	**b**	$F1D0_H$	**E**	$E8_H$	CPU System Configuration Register 2	0000_H
SYSCON3	**b**	$F1D4_H$	**E**	EA_H	CPU System Configuration Register 3	0000_H
T12IC	**b**	$F190_H$	**E**	$C8_H$	CAPCOM 6 Timer 12 Interrupt Ctrl. Reg.	0000_H
T12OF		$F034_H$	**E**	$1A_H$	CAPCOM 6 Timer 12 Offset Register	0000_H
T12P		$F030_H$	**E**	18_H	CAPCOM 6 Timer 12 Period Register	0000_H
T13IC	**b**	$F198_H$	**E**	CC_H	CAPCOM 6 Timer 13 Interrupt Ctrl. Reg.	0000_H
T13P		$F032_H$	**E**	19_H	CAPCOM 6 Timer 13 Period Register	0000_H
T14		$F0D2_H$	**E**	69_H	RTC Timer 14 Register	no
T14REL		$F0D0_H$	**E**	68_H	RTC Timer 14 Reload Register	no
T2		$FE40_H$		20_H	GPT1 Timer 2 Register	0000_H
T2CON	**b**	$FF40_H$		$A0_H$	GPT1 Timer 2 Control Register	0000_H

Table 23-3 C164CI Registers Ordered by Name (cont'd)

Name		Physical Address		8-Bit Addr.	Description	Reset Value
T2IC	**b**	$FF60_H$		$B0_H$	GPT1 Timer 2 Interrupt Control Register	0000_H
T3		$FE42_H$		21_H	GPT1 Timer 3 Register	0000_H
T3CON	**b**	$FF42_H$		$A1_H$	GPT1 Timer 3 Control Register	0000_H
T3IC	**b**	$FF62_H$		$B1_H$	GPT1 Timer 3 Interrupt Control Register	0000_H
T4		$FE44_H$		22_H	GPT1 Timer 4 Register	0000_H
T4CON	**b**	$FF44_H$		$A2_H$	GPT1 Timer 4 Control Register	0000_H
T4IC	**b**	$FF64_H$		$B2_H$	GPT1 Timer 4 Interrupt Control Register	0000_H
T7		$F050_H$	**E**	28_H	CAPCOM Timer 7 Register	0000_H
T78CON	**b**	$FF20_H$		90_H	CAPCOM Timer 7 and 8 Ctrl. Reg.	0000_H
T7IC	**b**	$F17A_H$	**E**	BD_H	CAPCOM Timer 7 Interrupt Ctrl. Reg.	0000_H
T7REL		$F054_H$	**E**	$2A_H$	CAPCOM Timer 7 Reload Register	0000_H
T8		$F052_H$	**E**	29_H	CAPCOM Timer 8 Register	0000_H
T8IC	**b**	$F17C_H$	**E**	BE_H	CAPCOM Timer 8 Interrupt Ctrl. Reg.	0000_H
T8REL		$F056_H$	**E**	$2B_H$	CAPCOM Timer 8 Reload Register	0000_H
TFR	**b**	$FFAC_H$		$D6_H$	Trap Flag Register	0000_H
TRCON	**b**	$FF34_H$		$9A_H$	CAPCOM 6 Trap Enable Ctrl. Reg.	$00XX_H$
WDT		$FEAE_H$		57_H	Watchdog Timer Register (read only)	0000_H
WDTCON		$FFAE_H$		$D7_H$	Watchdog Timer Control Register	[2] $00xx_H$
XP0IC	**b**	$F186_H$	**E**	$C3_H$	CAN1 Module Interrupt Control Register	0000_H
XP3IC	**b**	$F19E_H$	**E**	CF_H	PLL/RTC Interrupt Control Register	0000_H
ZEROS	**b**	$FF1C_H$		$8E_H$	Constant Value 0's Register (read only)	0000_H

[1] The system configuration is selected during reset.

[2] The reset value depends on the indicated reset source.

23.4 Registers Ordered by Address

Table 23-4 lists all registers implemented in the C164CI ordered by their physical address. The following markings assist in classifying the listed registers:
"**b**" in the "Name" column marks **Bit-addressable** SFRs.
"**E**" in the "Physical Address" column marks (**E**)SFRs in the **Extended SFR-Space**.
"**m**" in the "Physical Address" column marks SFRs without short 8-bit address.
"**X**" in the "Physical Address" column marks registers within on-chip **X-Peripherals**.

Table 23-4 C164CI Registers Ordered by Address

Name	Physical Address		8-Bit Addr.	Description	Reset Value
OPCTRL	$EDC0_H$	**X**	---	OTP Progr. Interface Control Register	0007_H
OPAD	$EDC2_H$	**X**	---	OTP Progr. Interface Address Register	0000_H
OPDAT	$EDC4_H$	**X**	---	OTP Progr. Interface Data Register	0000_H
C1CSR	$EF00_H$	**X**	–	CAN1 Control / Status Register	$XX01_H$
C1PCIR	$EF02_H$	**X**	–	CAN1 Port Control / Interrupt Register	$XXXX_H$
C1BTR	$EF04_H$	**X**	–	CAN1 Bit Timing Register	$UUUU_H$
C1GMS	$EF06_H$	**X**	–	CAN1 Global Mask Short	$UFUU_H$
C1UGML	$EF08_H$	**X**	–	CAN Upper Global Mask Long	$UUUU_H$
C1LGML	$EF0A_H$	**X**	–	CAN Lower Global Mask Long	$UUUU_H$
C1UMLM	$EF0C_H$	**X**	–	CAN Upper Mask of Last Message	$UUUU_H$
C1LMLM	$EF0E_H$	**X**	–	CAN Lower Mask of Last Message	$UUUU_H$
C1MCRn	$EFn0_H$	**X**	–	CAN Message Control Register (msg. **n**)	$UUUU_H$
C1UARn	$EFn2_H$	**X**	–	CAN Upper Arbitration Register (msg. **n**)	$UUUU_H$
C1LARn	$EFn4_H$	**X**	–	CAN Lower Arbitration Register (msg. **n**)	$UUUU_H$
C1MCFGn	$EFn6_H$	**X**	–	CAN Message Configuration Register (msg. **n**)	UU_H
T12P	$F030_H$	**E**	18_H	CAPCOM 6 Timer 12 Period Register	0000_H
T13P	$F032_H$	**E**	19_H	CAPCOM 6 Timer 13 Period Register	0000_H
T12OF	$F034_H$	**E**	$1A_H$	CAPCOM 6 Timer 12 Offset Register	0000_H
CC6MSEL	$F036_H$	**E**	$1B_H$	CAPCOM 6 Mode Select Register	0000_H
T7	$F050_H$	**E**	28_H	CAPCOM Timer 7 Register	0000_H
T8	$F052_H$	**E**	29_H	CAPCOM Timer 8 Register	0000_H
T7REL	$F054_H$	**E**	$2A_H$	CAPCOM Timer 7 Reload Register	0000_H

Table 23-4 C164CI Registers Ordered by Address (cont'd)

Name	Physical Address		8-Bit Addr.	Description	Reset Value
T8REL	$F056_H$	**E**	$2B_H$	CAPCOM Timer 8 Reload Register	0000_H
IDMEM2	$F076_H$	**E**	$3B_H$	Identifier	$XXXX_H$
IDPROG	$F078_H$	**E**	$3C_H$	Identifier	$XXXX_H$
IDMEM	$F07A_H$	**E**	$3D_H$	Identifier	$XXXX_H$
IDCHIP	$F07C_H$	**E**	$3E_H$	Identifier	$XXXX_H$
IDMANUF	$F07E_H$	**E**	$3F_H$	Identifier	1820_H
POCON0L	$F080_H$	**E**	40_H	Port P0L Output Control Register	0000_H
POCON0H	$F082_H$	**E**	41_H	Port P0H Output Control Register	0000_H
POCON1L	$F084_H$	**E**	42_H	Port P1L Output Control Register	0000_H
POCON1H	$F086_H$	**E**	43_H	Port P1H Output Control Register	0000_H
POCON3	$F08A_H$	**E**	45_H	Port P3 Output Control Register	0000_H
POCON4	$F08C_H$	**E**	46_H	Port P4 Output Control Register	0000_H
POCON8	$F092_H$	**E**	49_H	Port P8 Output Control Register	0000_H
ADDAT2	$F0A0_H$	**E**	50_H	A/D Converter 2 Result Register	0000_H
POCON20	$F0AA_H$	**E**	55_H	Dedicated Pin Output Control Register	0000_H
PTCR	$F0AE_H$	**E**	57_H	Port Temperature Compensation Reg.	0000_H
SSCTB	$F0B0_H$	**E**	58_H	SSC Transmit Buffer	0000_H
SSCRB	$F0B2_H$	**E**	59_H	SSC Receive Buffer	$XXXX_H$
SSCBR	$F0B4_H$	**E**	$5A_H$	SSC Baudrate Register	0000_H
T14REL	$F0D0_H$	**E**	68_H	RTC Timer 14 Reload Register	no
T14	$F0D2_H$	**E**	69_H	RTC Timer 14 Register	no
RTCL	$F0D4_H$	**E**	$6A_H$	RTC Low Register	no
RTCH	$F0D6_H$	**E**	$6B_H$	RTC High Register	no
DP0L **b**	$F100_H$	**E**	80_H	P0L Direction Control Register	00_H
DP0H **b**	$F102_H$	**E**	81_H	P0H Direction Control Register	00_H
DP1L **b**	$F104_H$	**E**	82_H	P1L Direction Control Register	00_H
DP1H **b**	$F106_H$	**E**	83_H	P1H Direction Control Register	00_H
RP0H **b**	$F108_H$	**E**	84_H	System Startup Config. Reg. (Rd. only)	XX_H
CC16IC **b**	$F160_H$	**E**	$B0_H$	CAPCOM Reg. 16 Interrupt Ctrl. Reg.	0000_H
CC17IC **b**	$F162_H$	**E**	$B1_H$	CAPCOM Reg. 17 Interrupt Ctrl. Reg.	0000_H

Table 23-4 C164CI Registers Ordered by Address (cont'd)

Name		Physical Address		8-Bit Addr.	Description	Reset Value
CC18IC	**b**	$F164_H$	**E**	$B2_H$	CAPCOM Reg. 18 Interrupt Ctrl. Reg.	0000_H
CC19IC	**b**	$F166_H$	**E**	$B3_H$	CAPCOM Reg. 19 Interrupt Ctrl. Reg.	0000_H
CC24IC	**b**	$F170_H$	**E**	$B8_H$	CAPCOM Reg. 24 Interrupt Ctrl. Reg.	0000_H
CC25IC	**b**	$F172_H$	**E**	$B9_H$	CAPCOM Reg. 25 Interrupt Ctrl. Reg.	0000_H
CC26IC	**b**	$F174_H$	**E**	BA_H	CAPCOM Reg. 26 Interrupt Ctrl. Reg.	0000_H
CC27IC	**b**	$F176_H$	**E**	BB_H	CAPCOM Reg. 27 Interrupt Ctrl. Reg.	0000_H
T7IC	**b**	$F17A_H$	**E**	BD_H	CAPCOM Timer 7 Interrupt Ctrl. Reg.	0000_H
T8IC	**b**	$F17C_H$	**E**	BE_H	CAPCOM Timer 8 Interrupt Ctrl. Reg.	0000_H
CC6CIC	**b**	$F17E_H$	**E**	BF_H	CAPCOM 6 Interrupt Control Register	0000_H
XP0IC	**b**	$F186_H$	**E**	$C3_H$	CAN1 Module Interrupt Control Register	0000_H
CC6EIC	**b**	$F188_H$	**E**	$C4_H$	CAPCOM 6 Emergency Interrupt Control Register	0000_H
T12IC	**b**	$F190_H$	**E**	$C8_H$	CAPCOM 6 Timer 12 Interrupt Ctrl. Reg.	0000_H
T13IC	**b**	$F198_H$	**E**	CC_H	CAPCOM 6 Timer 13 Interrupt Ctrl. Reg.	0000_H
S0TBIC	**b**	$F19C_H$	**E**	CE_H	Serial Channel 0 Transmit Buffer Interrupt Control Register	0000_H
XP3IC	**b**	$F19E_H$	**E**	CF_H	PLL/RTC Interrupt Control Register	0000_H
EXICON	**b**	$F1C0_H$	**E**	$E0_H$	External Interrupt Control Register	0000_H
PICON	**b**	$F1C4_H$	**E**	$E2_H$	Port Input Threshold Control Register	0000_H
ODP3	**b**	$F1C6_H$	**E**	$E3_H$	Port 3 Open Drain Control Register	0000_H
ODP4	**b**	$F1CA_H$	**E**	$E5_H$	Port 4 Open Drain Control Register	00_H
SYSCON2	**b**	$F1D0_H$	**E**	$E8_H$	CPU System Configuration Register 2	0000_H
SYSCON3	**b**	$F1D4_H$	**E**	EA_H	CPU System Configuration Register 3	0000_H
ODP8	**b**	$F1D6_H$	**E**	EB_H	Port 8 Open Drain Control Register	00_H
EXISEL	**b**	$F1DA_H$	**E**	ED_H	External Interrupt Source Select Reg.	0000_H
SYSCON1	**b**	$F1DC_H$	**E**	EE_H	CPU System Configuration Register 1	0000_H
ISNC	**b**	$F1DE_H$	**E**	EF_H	Interrupt Subnode Control Register	0000_H
RSTCON	**b**	$F1E0_H$	**m**	–	Reset Control Register	$00XX_H$
DPP0		$FE00_H$		00_H	CPU Data Page Pointer 0 Reg. (10 bits)	0000_H
DPP1		$FE02_H$		01_H	CPU Data Page Pointer 1 Reg. (10 bits)	0001_H

Table 23-4 C164CI Registers Ordered by Address (cont'd)

Name	Physical Address	8-Bit Addr.	Description	Reset Value
DPP2	$FE04_H$	02_H	CPU Data Page Pointer 2 Reg. (10 bits)	0002_H
DPP3	$FE06_H$	03_H	CPU Data Page Pointer 3 Reg. (10 bits)	0003_H
CSP	$FE08_H$	04_H	CPU Code Segment Pointer Register (8 bits, not directly writable)	0000_H
MDH	$FE0C_H$	06_H	CPU Multiply Divide Reg. – High Word	0000_H
MDL	$FE0E_H$	07_H	CPU Multiply Divide Reg. – Low Word	0000_H
CP	$FE10_H$	08_H	CPU Context Pointer Register	$FC00_H$
SP	$FE12_H$	09_H	CPU System Stack Pointer Register	$FC00_H$
STKOV	$FE14_H$	$0A_H$	CPU Stack Overflow Pointer Register	$FA00_H$
STKUN	$FE16_H$	$0B_H$	CPU Stack Underflow Pointer Register	$FC00_H$
ADDRSEL1	$FE18_H$	$0C_H$	Address Select Register 1	0000_H
ADDRSEL2	$FE1A_H$	$0D_H$	Address Select Register 2	0000_H
ADDRSEL3	$FE1C_H$	$0E_H$	Address Select Register 3	0000_H
ADDRSEL4	$FE1E_H$	$0F_H$	Address Select Register 4	0000_H
CC60	$FE30_H$	18_H	CAPCOM 6 Register 0	0000_H
CC61	$FE32_H$	19_H	CAPCOM 6 Register 1	0000_H
CC62	$FE34_H$	$1A_H$	CAPCOM 6 Register 2	0000_H
CMP13	$FE36_H$	$1B_H$	CAPCOM 6 Timer 13 Compare Reg.	0000_H
T2	$FE40_H$	20_H	GPT1 Timer 2 Register	0000_H
T3	$FE42_H$	21_H	GPT1 Timer 3 Register	0000_H
T4	$FE44_H$	22_H	GPT1 Timer 4 Register	0000_H
CC16	$FE60_H$	30_H	CAPCOM Register 16	0000_H
CC17	$FE62_H$	31_H	CAPCOM Register 17	0000_H
CC18	$FE64_H$	32_H	CAPCOM Register 18	0000_H
CC19	$FE66_H$	33_H	CAPCOM Register 19	0000_H
CC20	$FE68_H$	34_H	CAPCOM Register 20	0000_H
CC21	$FE6A_H$	35_H	CAPCOM Register 21	0000_H
CC22	$FE6C_H$	36_H	CAPCOM Register 22	0000_H
CC23	$FE6E_H$	37_H	CAPCOM Register 23	0000_H
CC24	$FE70_H$	38_H	CAPCOM Register 24	0000_H

Table 23-4 C164CI Registers Ordered by Address (cont'd)

Name	Physical Address	8-Bit Addr.	Description	Reset Value
CC25	$FE72_H$	39_H	CAPCOM Register 25	0000_H
CC26	$FE74_H$	$3A_H$	CAPCOM Register 26	0000_H
CC27	$FE76_H$	$3B_H$	CAPCOM Register 27	0000_H
CC28	$FE78_H$	$3C_H$	CAPCOM Register 28	0000_H
CC29	$FE7A_H$	$3D_H$	CAPCOM Register 29	0000_H
CC30	$FE7C_H$	$3E_H$	CAPCOM Register 30	0000_H
CC31	$FE7E_H$	$3F_H$	CAPCOM Register 31	0000_H
ADDAT	$FEA0_H$	50_H	A/D Converter Result Register	0000_H
WDT	$FEAE_H$	57_H	Watchdog Timer Register (read only)	0000_H
S0TBUF	$FEB0_H$	58_H	Serial Channel 0 Transmit Buffer Reg. (write only)	0000_H
S0RBUF	$FEB2_H$	59_H	Serial Channel 0 Receive Buffer Reg. (read only)	$XXXX_H$
S0BG	$FEB4_H$	$5A_H$	Serial Channel 0 Baud Rate Generator Reload Register	0000_H
PECC0	$FEC0_H$	60_H	PEC Channel 0 Control Register	0000_H
PECC1	$FEC2_H$	61_H	PEC Channel 1 Control Register	0000_H
PECC2	$FEC4_H$	62_H	PEC Channel 2 Control Register	0000_H
PECC3	$FEC6_H$	63_H	PEC Channel 3 Control Register	0000_H
PECC4	$FEC8_H$	64_H	PEC Channel 4 Control Register	0000_H
PECC5	$FECA_H$	65_H	PEC Channel 5 Control Register	0000_H
PECC6	$FECC_H$	66_H	PEC Channel 6 Control Register	0000_H
PECC7	$FECE_H$	67_H	PEC Channel 7 Control Register	0000_H
P0L **b**	$FF00_H$	80_H	Port 0 Low Reg. (Lower half of PORT0)	00_H
P0H **b**	$FF02_H$	81_H	Port 0 High Reg. (Upper half of PORT0)	00_H
P1L **b**	$FF04_H$	82_H	Port 1 Low Reg. (Lower half of PORT1)	00_H
P1H **b**	$FF06_H$	83_H	Port 1 High Reg. (Upper half of PORT1)	00_H
BUSCON0 **b**	$FF0C_H$	86_H	Bus Configuration Register 0	0000_H
MDC **b**	$FF0E_H$	87_H	CPU Multiply Divide Control Register	0000_H
PSW **b**	$FF10_H$	88_H	CPU Program Status Word	0000_H

Table 23-4 C164CI Registers Ordered by Address (cont'd)

Name	Physical Address	8-Bit Addr.	Description	Reset Value
SYSCON b	$FF12_H$	89_H	CPU System Configuration Register	[1]$0xx0_H$
BUSCON1 b	$FF14_H$	$8A_H$	Bus Configuration Register 1	0000_H
BUSCON2 b	$FF16_H$	$8B_H$	Bus Configuration Register 2	0000_H
BUSCON3 b	$FF18_H$	$8C_H$	Bus Configuration Register 3	0000_H
BUSCON4 b	$FF1A_H$	$8D_H$	Bus Configuration Register 4	0000_H
ZEROS b	$FF1C_H$	$8E_H$	Constant Value 0's Register (read only)	0000_H
ONES b	$FF1E_H$	$8F_H$	Constant Value 1's Register (read only)	$FFFF_H$
T78CON b	$FF20_H$	90_H	CAPCOM Timer 7 and 8 Ctrl. Reg.	0000_H
CCM4 b	$FF22_H$	91_H	CAPCOM Mode Control Register 4	0000_H
CCM5 b	$FF24_H$	92_H	CAPCOM Mode Control Register 5	0000_H
CCM6 b	$FF26_H$	93_H	CAPCOM Mode Control Register 6	0000_H
CCM7 b	$FF28_H$	94_H	CAPCOM Mode Control Register 7	0000_H
CTCON b	$FF30_H$	98_H	CAPCOM 6 Compare Timer Ctrl. Reg.	1010_H
CC6MCON b	$FF32_H$	99_H	CAPCOM 6 Mode Control Register	$00FF_H$
TRCON b	$FF34_H$	$9A_H$	CAPCOM 6 Trap Enable Ctrl. Reg.	$00XX_H$
CC6MIC b	$FF36_H$	$9B_H$	CAPCOM 6 Mode Interrupt Ctrl. Reg.	0000_H
T2CON b	$FF40_H$	$A0_H$	GPT1 Timer 2 Control Register	0000_H
T3CON b	$FF42_H$	$A1_H$	GPT1 Timer 3 Control Register	0000_H
T4CON b	$FF44_H$	$A2_H$	GPT1 Timer 4 Control Register	0000_H
T2IC b	$FF60_H$	$B0_H$	GPT1 Timer 2 Interrupt Control Register	0000_H
T3IC b	$FF62_H$	$B1_H$	GPT1 Timer 3 Interrupt Control Register	0000_H
T4IC b	$FF64_H$	$B2_H$	GPT1 Timer 4 Interrupt Control Register	0000_H
S0TIC b	$FF6C_H$	$B6_H$	Serial Channel 0 Transmit Interrupt Control Register	0000_H
S0RIC b	$FF6E_H$	$B7_H$	Serial Channel 0 Receive Interrupt Control Register	0000_H
S0EIC b	$FF70_H$	$B8_H$	Serial Channel 0 Error Interrupt Ctrl. Reg.	0000_H
SSCTIC b	$FF72_H$	$B9_H$	SSC Transmit Interrupt Control Register	0000_H
SSCRIC b	$FF74_H$	BA_H	SSC Receive Interrupt Control Register	0000_H

Table 23-4 C164CI Registers Ordered by Address (cont'd)

Name		Physical Address	8-Bit Addr.	Description	Reset Value
SSCEIC	**b**	$FF76_H$	BB_H	SSC Error Interrupt Control Register	0000_H
CC8IC	**b**	$FF88_H$	$C4_H$	External Interrupt 0 Control Register	0000_H
CC9IC	**b**	$FF8A_H$	$C5_H$	External Interrupt 1 Control Register	0000_H
CC10IC	**b**	$FF8C_H$	$C6_H$	External Interrupt 2 Control Register	0000_H
CC11IC	**b**	$FF8E_H$	$C7_H$	External Interrupt 3 Control Register	0000_H
ADCIC	**b**	$FF98_H$	CC_H	A/D Converter End of Conversion Interrupt Control Register	0000_H
ADEIC	**b**	$FF9A_H$	CD_H	A/D Converter Overrun Error Interrupt Control Register	0000_H
ADCON	**b**	$FFA0_H$	$D0_H$	A/D Converter Control Register	0000_H
P5	**b**	$FFA2_H$	$D1_H$	Port 5 Register (read only)	$XXXX_H$
P5DIDIS	**b**	$FFA4_H$	$D2_H$	Port 5 Digital Input Disable Register	0000_H
FOCON	**b**	$FFAA_H$	$D5_H$	Frequency Output Control Register	0000_H
TFR	**b**	$FFAC_H$	$D6_H$	Trap Flag Register	0000_H
WDTCON		$FFAE_H$	$D7_H$	Watchdog Timer Control Register	[2] $00xx_H$
S0CON	**b**	$FFB0_H$	$D8_H$	Serial Channel 0 Control Register	0000_H
SSCCON	**b**	$FFB2_H$	$D9_H$	SSC Control Register	0000_H
P3	**b**	$FFC4_H$	$E2_H$	Port 3 Register	0000_H
DP3	**b**	$FFC6_H$	$E3_H$	Port 3 Direction Control Register	0000_H
P4	**b**	$FFC8_H$	$E4_H$	Port 4 Register (7 bits)	00_H
DP4	**b**	$FFCA_H$	$E5_H$	Port 4 Direction Control Register	00_H
P8	**b**	$FFD4_H$	EA_H	Port 8 Register (8 bits)	00_H
DP8	**b**	$FFD6_H$	EB_H	Port 8 Direction Control Register	00_H

[1] The system configuration is selected during reset.

[2] The reset value depends on the indicated reset source.

23.5 Special Notes

PEC Pointer Registers

The source and destination pointers for the Peripheral Event Controller (PEC) are mapped to a special area within the internal RAM. Pointers not occupied by the PEC may be used like normal RAM. During Power Down mode or any warm reset, the PEC pointers are preserved.

The PEC and its registers are described in **Chapter 5**.

GPR Access in the ESFR Area

The locations $00'F000_H$ … $00'F01E_H$ within the ESFR area are reserved and allow to access the current register bank via short register addressing modes. The GPRs are mirrored to the ESFR area to allow access to the current register bank even after switching register spaces (see example below).

```
MOV     R5, DP3             ;GPR access via SFR area
EXTR    #1
MOV     R5, ODP3            ;GPR access via ESFR area
```

Writing Bytes to SFRs

All Special Function Registers may be accessed wordwise or bytewise (some of them even bitwise). Reading bytes from word SFRs is a non-critical operation. However, when writing bytes to word SFRs, the complementary byte of the respective SFR is cleared with the write operation.

24 Instruction Set Summary

This chapter briefly summarizes the C164CI's instructions by instruction classes. This provides a basic description of the C164CI's instruction set, the power and versatility of the instructions, and their general usage.

Detailed descriptions of each individual instruction, including its operand data type, condition flag settings, addressing modes, length (number of bytes), and object code format are provided in the **"Instruction Set Manual"** for the C166 Family. This manual also provides tables listing the instructions according to various criteria to facilitate quick information access.

Summary of Instruction Classes

Grouping the various instructions into classes assists in identifying similar instructions (such as SHR, ROR) and variations of instructions (such as ADD, ADDB). This provides an easy access to the possibilities and power of the instructions of the C164CI.

Note: The used mnemonics refer to the detailed description.

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

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

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

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

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

Prioritize Instruction

- Determination of the number of shift cycles required to normalize a word operand (floating point support): PRIOR

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 variety of different addressing modes including indirect addressing and automatic pointer in-/decrementing.

System Stack Instructions

- Pushing a word onto the system stack: PUSH
- Popping a word from the system stack: POP
- Saving a word on the system stack, and then updating the old word with a new value (provided for register bank switching): SCXT

Jump Instructions

• Conditional jumping to either an 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 the case of a jump taken (semaphore support):	JBC	JNBS	

Call Instructions

• Conditional calling of either an 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 0:	TRAP	

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

System Control Instructions

- Resetting the C164CI via software: SRST
- Entering the Idle mode: IDLE
- Entering the Power Down mode: PWRDN
- Servicing the Watchdog Timer: SRVWDT
- Disabling the Watchdog Timer: DISWDT
- Signifying the end of the initialization routine (pulls pin $\overline{\text{RSTOUT}}$ high, and disables the effect of any later execution of a DISWDT instruction): EINIT

Miscellaneous

- Null operation which requires two bytes of storage and the minimum time for execution: NOP
- Definition of an inseparable 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). This is advantageous for larger memory models in high level languages. Refer to* ***Chapter 22*** *for examples.*

Protected Instructions

Some instructions of the C164CI 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. Regular instructions use only a portion of it (such as the lower 8 bits), with the other bits providing additional information such as indicating the involved registers. Decoding all 32 bits of a protected doubleword instruction increases security in cases of data distortion during instruction fetching. Critical operations such as a software reset are, therefore, executed only if the complete instruction is decoded without an error. This enhances the safety and reliability of a microcontroller system.

25 Device Specification

The device specification describes the electrical parameters of the device. It lists DC characteristics such as input/output voltages, supply voltages, input/output/supply currents, as well as AC characteristics such as timing characteristics and requirements.

Other than the architecture, the instruction set, or the basic functions of the C164CI core and its peripherals, these DC and AC characteristics are subject to change due to device improvements or development of specific derivatives of the standard device.

Therefore, these characteristics are not contained in this User's Manual, but are provided in device-specific Data Sheets, 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: The specific characteristics of a device should always be verified before a new design is started to ensure use of the most current information.

Figure 25-1 shows the pin diagram of the C164CI. It shows the location of the various supply and IO pins. Detailed descriptions of all pins are also found in the Data Sheet.

*Note: Not all alternate functions shown in **Figure 25-1** are supported by all derivatives. Please refer to the corresponding descriptions in their data sheets.*

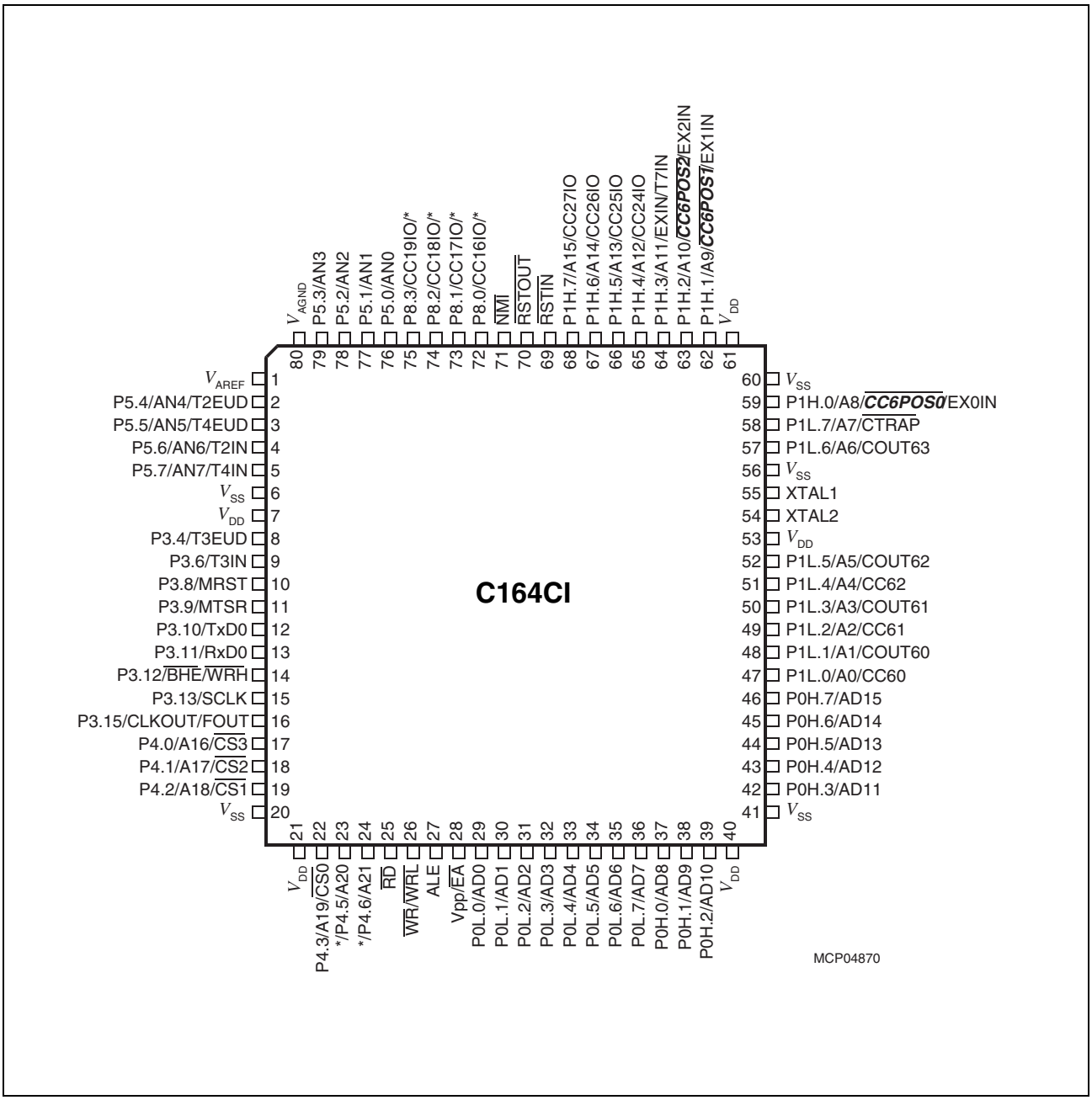


Figure 25-1 Pin Configuration for C164CI, P-MQFP-80 Package

***)** Port 4 and Port 8 pins indicated with (*) can have CAN interface lines assigned to them. **Chapter 19** lists the possible assignments.

The ***marked input signals*** are available only in devices with a full-function CAPCOM6. They are not available in devices with a reduced-function CAPCOM6.

26 Keyword Index

This section lists a number of keywords which refer to specific details of the C164CI in terms of its architecture, its functional units or functions. This helps to quickly find the answer to specific questions about the C164CI.

A

Access to X-Peripherals 9-30
Acronyms 1-8
Adapt Mode 20-15
ADC 2-15, 18-1
ADCIC, ADEIC 18-15
ADCON 18-3
ADDAT, ADDAT2 18-5
Address
  Arbitration 9-24
  Area Definition 9-23
  Boundaries 3-11
  Segment 9-9, 20-18
ADDRSELx 9-20, 9-24
ALE length 9-13
Alternate signals 7-10
ALU 4-16
Analog/Digital Converter 2-15, 18-1
Arbitration
  Address 9-24
ASC0 11-1
  Asynchronous mode 11-5
  Baudrate 11-11
  Error Detection 11-10
  Interrupts 11-15
  Synchronous mode 11-8
Asynchronous Serial Interface (->ASC0) 11-1
Auto Scan conversion 18-7

B

Baudrate
  ASC0 11-11
  Bootstrap Loader 15-6
  SSC 12-13
BHE 7-25, 9-9
Bidirectional reset 20-4
Bit
  addressable memory 3-4
  Handling 4-10
  Manipulation Instructions 24-2
  protected 2-20, 4-10
  reserved 2-12
Block Commutation Mode 17-16
Bootstrap Loader 15-1, 20-16
Boundaries 3-11
BTR 19-12
Burst Mode
  CAPCOM6 17-10
Bus
  CAN 2-14, 19-1, 19-36
  Demultiplexed 9-5
  Idle State 9-27
  Mode Configuration 9-3, 20-17
  Multiplexed 9-4
BUSCONx 9-19, 9-24

C

CAN Interface 2-14, 19-1
  activation 19-31
  port control 19-37
CAPCOM 2-17
  interrupt 16-22
  timer 16-4
  Trap Function 17-18
  unit 16-1, 17-1
Capture Mode
  CAPCOM2 16-13
  CAPCOM6 17-11
  GPT1 10-19
Capture/Compare unit 16-1, 17-1

CC6IC, CC6EIC 17-33
CC6MCON 17-26
CC6MIC 17-30
CC6MSEL 17-29
CCM4, CCM5, CCM6, CCM7 16-10
CCxIC 5-28, 16-22
Center Aligned Mode
  CAPCOM6 17-7
Chip Select
  Configuration 9-10, 20-18
  Latched/Early 9-11
Clock
  distribution 6-2, 21-14
  generator modes 6-8, 20-19
  output signal 21-16
Code memory handling 22-16
Compare modes 16-14
  double register 16-19
Concatenation of Timers 10-16
Configuration
  Address 9-9, 20-18
  Bus Mode 9-3, 20-17
  Chip Select 9-10, 20-18
  default 20-20
  of pins 25-2
  PLL 6-8, 20-19
  Reset 20-7, 20-12
  special modes 20-16
  Write Control 20-17
Context
  Pointer 4-25
  Switching 5-18
Conversion
  Analog/Digital 18-1
  Auto Scan 18-7
  timing control 18-13
Count direction 10-4
Counter 10-8, 10-14
CP 4-25
CPU 2-2, 4-1
  Host Mode (CHM) 3-15
CSP 4-20
CSR 19-7
CTCON 17-23

D

Data Page 4-22, 22-14
  boundaries 3-11
Default startup configuration 20-20
Delay
  Read/Write 9-16
Demultiplexed Bus 9-5
Development Support 1-7
Direct Drive 6-7
Direction
  count 10-4
Disable
  Interrupt 5-15
  Peripheral 21-14
  Segmentation 4-15
Division 4-30, 22-1
Double-Register compare 16-19
DP0L, DP0H 7-13
DP1L, DP1H 7-17
DP3 7-21
DP4 7-26
DP8 7-33
DPP 4-22, 22-14
Driver characteristic (ports) 7-5

E

Early chip select 9-11
Early WR control 9-16
Edge Aligned Mode
  CAPCOM6 17-5
Edge characteristic (ports) 7-5
Emulation Mode 20-14
Enable
  Interrupt 5-15
  Peripheral 21-14
  Segmentation 4-15
  XBUS peripherals 9-29
Error Detection
  ASC0 11-10
  CAN 19-4
  SSC 12-15

EXICON 5-27
EXISEL 5-29
External
  Bus 2-10
  Bus Characteristics 9-12–9-17
  Bus Idle State 9-27
  Bus Modes 9-3–9-8
  Fast interrupts 5-27
  Host Mode (EHM) 3-16, 20-16
  Interrupt source control 5-29
  Interrupts 5-25
  Interrupts during sleep mode 5-30
  startup configuration 20-13

F

Fast external interrupts 5-27
Flags 4-16–4-19
FOCON 21-17
Frequency output signal 21-16
Full-Duplex 12-7

G

GMS 19-15
GPR 3-6, 4-25, 23-2
GPT 2-16
GPT1 10-1

H

Half-Duplex 12-10
Hardware
  Reset 20-2
  Traps 5-31

I

Idle
  Mode 21-3
  State (Bus) 9-27
Incremental Interface 10-9
Indication of reset source 13-6
Input threshold 7-2
Inseparable instructions 22-14
Instruction 22-1, 24-1
  Bit Manipulation 24-2
  Branch 4-4
  inseparable 22-14
  Pipeline 4-3
  protected 24-4
  Timing 4-11
Interface
  CAN 2-14, 19-1
  External Bus 9-1
  serial async. (->ASC0) 11-1
  serial sync. (->SSC) 12-1
Internal RAM (->IRAM) 3-4
Interrupt
  CAPCOM 16-22
  during sleep mode 5-30
  Enable/Disable 5-15
  External 5-25
  Fast external 5-27
  Handling CAN 19-9
  Node Sharing 5-24
  Priority 5-7
  Processing 5-1, 5-5
  Response Times 5-19
  RTC 14-3
  source control 5-29
  Sources 5-2
  System 2-7, 5-2
  Vectors 5-2
IP 4-19
IRAM 3-4
  status after reset 20-8
ISNC 5-24

L

LARn 19-21
Latched chip select 9-11
LGML 19-16
LMLM 19-17

M

Management
  Peripheral 21-14
  Power 21-1
MCFGn 19-22

MCRn 19-19
MDC 4-32
MDH 4-30
MDL 4-31
Memory 2-8
  bit-addressable 3-4
  Code memory handling 22-16
  Cycle Time 9-14
  External 3-10
  OTP 3-13
  RAM/SFR 3-4
  ROM area 3-3
  Tri-state time 9-15
  XRAM 3-9
Multi-Channel Modes (CAPCOM6) 17-12
Multiplexed Bus 9-4
Multiplication 4-30, 22-1

N

NMI 5-1, 5-34
Noise filter (Ext. Interrupts) 5-30

O

ODP3 7-22
ODP4 7-27
ODP8 7-34
ONES 4-33
OPCTRL 3-17
Open Drain Mode 7-4
Oscillator
  circuitry 6-3
  measurement 6-3
  Watchdog 6-9, 20-19
OTP
  Programming 3-13

P

P0L, P0H 7-12
P1L, P1H 7-16
P3 7-21
P4 7-26
P5 7-30
P5DIDIS 7-32
P8 7-33
PCIR 19-10
PEC 2-8, 3-7, 5-11
  Response Times 5-22
PECCx 5-11
Peripheral
  enable on XBUS 9-29
  Enable/Disable 21-14
  Management 21-14
  Summary 2-11
Phase Locked Loop (->PLL) 6-1
Phase Sequences 17-14
PICON 7-2
Pins 8-1
  configuration 25-2
  in Idle and Power Down mode 21-9
Pipeline 4-3
  Effects 4-6
PLL 6-1, 20-19
POCON* 7-6
Port 2-14
  driver characteristic 7-5
  edge characteristic 7-5
  input threshold 7-2
  Temperature compensation 7-8
Power Down Mode 21-6
Power Management 2-18, 21-1
Prescaler 6-7
Programming
  CPU Host Mode 3-15
  External Host Mode 3-16
  OTP 3-13
Protected
  Bits 2-20, 4-10
  instruction 24-4
PSW 4-16, 5-9
PTCR 7-9

R

RAM
  extension 3-9
  internal 3-4
Read/Write Delay 9-16

Real Time Clock (->RTC) 14-1
Registers 23-1
  sorted by address 23-11
  sorted by name 23-4
Reserved bits 2-12
Reset 20-1
  Bidirectional 20-4
  Configuration 20-7, 20-12
  Hardware 20-2
  Output 20-8
  Software 20-3
  Source indication 13-6
  Values 20-5
  Watchdog Timer 20-3
RP0H 9-25
RSTCON 20-23
RTC 2-15, 14-1

S

S0BG 11-11
S0CON 11-2
S0EIC, S0RIC, S0TIC, S0TBIC 11-15
S0RBUF 11-7, 11-9
S0TBUF 11-7, 11-9
Security Mechanism 21-21
Segment
  Address 9-9, 20-18
  boundaries 3-11
Segmentation 4-20
  Enable/Disable 4-15
Serial Interface 2-13, 11-1
  Asynchronous (->ASC0) 11-5
  CAN 2-14, 19-1
  Synchronous 11-8
  Synchronous (->SSC) 12-1
SFR 3-8, 23-4, 23-11
Sharing
  Interrupt Nodes 5-24
Single Chip Mode 9-2
  startup configuration 20-20
Sleep Mode 21-5
Slow Down Mode 21-10
Software
  Reset 20-3
  system configuration 20-22
  Traps 5-31
Source
  Interrupt 5-2
  Reset 13-6
SP 4-27
Special operation modes (config.) 20-16
SSC 12-1
  Baudrate generation 12-13
  Error Detection 12-15
  Full-Duplex 12-7
  Half-Duplex 12-10
SSCBR 12-13
SSCCON 12-2, 12-4
SSCEIC, SSCRIC, SSCTIC 12-17
SSCRB, SSCTB 12-8
Stack 3-5, 4-27, 22-4
Startup Configuration 20-7, 20-12
  external reset 20-13
  single-chip 20-20
  via software 20-22
STKOV 4-28
STKUN 4-29
Subroutine 22-10
Synchronous Serial Interface (->SSC) 12-1
SYSCON 4-13, 9-18
SYSCON1 21-6
SYSCON2 21-12
SYSCON3 21-15

T

T12IC, T13IC 17-33
T2CON 10-12
T2IC, T3IC, T4IC 10-20
T3CON 10-3
T4CON 10-12
T78CON 16-5
T7IC 16-9
T8IC 16-9
Temperature compensation 7-8
TFR 5-33

Threshold 7-2
Timer 2-16, 10-1
  Auxiliary Timer 10-12
  CAPCOM2 16-4
  CAPCOM6 17-3
  Concatenation 10-16
  Core Timer 10-3
Tools 1-7
Trap Function (CAPCOM6) 17-18
Traps 5-31
TRCON 17-25
Tri-State Time 9-15

U

UARn 19-21
UGML 19-16
UMLM 19-17
Unlock Sequence 21-21

V

Visible mode 9-29

W

Waitstate
  Memory Cycle 9-14
  Tri-State 9-15
  XBUS peripheral 9-29
Watchdog 2-18, 13-1
  after reset 20-5
  Oscillator 6-9, 20-19
  Reset 20-3
WDT 13-2
WDTCON 13-4

X

XBUS 2-10, 9-28
  enable peripherals 9-29
  external access 9-30
  waitstates 9-29
XRAM
  on-chip 3-9
  status after reset 20-8

Z

ZEROS 4-33

Infineon goes for Business Excellence

“Business excellence means intelligent approaches and clearly defined processes, which are both constantly under review and ultimately lead to good operating results.
Better operating results and business excellence mean less idleness and wastefulness for all of us, more professional success, more accurate information, a better overview and, thereby, less frustration and more satisfaction.”

Dr. Ulrich Schumacher

http://www.infineon.com

Published by Infineon Technologies AG