User's Manual, V 1.6, August 2001

C166S V1 SubSystem

C166S V1 SubS R1

16bit

Microcontrollers

Infineon technologies

Never stop thinking.

Edition 2001-08

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

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 (see address list).

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.

C166S V1 SubSystem

C166S V1 SubS R1

Microcontrollers

Never stop thinking.

C166S V1 SubS R1

Revision History: 2001-08 V 1.6

Previous Version: V 1.5

Page	**Subjects** (*major changes since last revision*)
2-12	Periodic Wake up from Idle or Sleep Mode
2-14	Clock Generation Unit, On-chip Bootstrap Loader
3-80..3-86	Particular Pipeline Effects
6-21	CALLA Instruction description
6-38	EINIT Instruction description
6-52	JMPA Instruction description
6-78	RETI Instruction description
6-91	SRVWDT Instruction description
6-96	TRAP Instruction description
8-1...8-29	RP0H Register
8-1, 8-34	DP3, P3, ODP4, ODP6 Registers
8-22	CLKEN System Clock Enable bit
8-31	External Bus Arbitration

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:

ce.cmd@infineon.com

Table of Contents **Page**

Table of Contents **Page**

Table of Contents **Page**

Table of Contents **Page**

Table of Contents **Page**

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.

About this Manual

This manual describes the functionality of the 16-bit microcontroller-subsystem C166S_R1 of the Infineon C166 Family, the C166S-class.

1.1 The Members of the 16-bit Microcontroller Family

The microcontroller-subsystem of the Infineon 16-bit family has 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 minimum response time to external stimuli (interrupts). Intelligent peripheral subsystems have been integrated to reduce the need for CPU (Central Processing Unit) 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 an 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 I/O (Input/ Output) pins.

The Internal Bus Interface (IBI) concept opens a straight forward path for the integration of application specific peripheral modules in addition to the standard on-chip peripherals in order to build application specific derivatives.

As programs for embedded control applications become larger, high level languages are favoured by programmers, because high level language programs are easier to write, to debug and to maintain.

The 80C166-type microcontrollers were the **first generation** of the 16-bit controller family. These devices have 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 (Random Access Memory) and highly efficient management of various resources on the external bus.

The C166S-type devices are members of the **third generation** of this family. This third generation is the synthesizable version of the second generation.

Enhanced derivatives of this second/third generation provide additional features like additional internal high-speed RAM, an integrated CAN-Module (Controller Area Network), an on-chip PLL (Phase Locked Loop), etc.

Utilizing integration to design 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 so-called Internal Bus Interface, defined for the Infineon 16-bit microcontrollers (XBus second generation). This Internal Bus Interface is an internal representation of the External Bus Interface that opens and simplifies the integration of peripherals by standardizing the required interface.

1.2 Summary of Basic Features

The C166S is an improved representative of the Infineon family of full featured 16-bit single-chip CMOS (Complementary Metal Oxide Silicon) microcontrollers. It combines high CPU performance with high peripheral functionality.
Several key features contribute to the high performance of the C166S bases subsystem (the indicated timings refer to a CPU clock of 50 MHz).

High Performance 16-Bit CPU With Four-Stage Pipeline

- 40 ns minimum instruction cycle time, with most instructions executed in 1 cycle
- 200 ns multiplication (16-bit *16-bit), 400 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 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

External Bus Interface

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

16-Priority-Level Interrupt System

- Up to 112 interrupt nodes with separate interrupt vectors
- 16 priority levels and 4(8) group levels

Up to 16-Channel Peripheral Event Controller (PEC)

- Interrupt driven single cycle data transfer
- Transfer count option (std. CPU interrupt after programmable number of PEC transfers)
- Long Transfer Counter
- Channel Linking
- Eliminates overhead of saving and restoring system state for interrupt requests

Intelligent On-chip Peripherals

- General Purpose Timer Unit Timer Block 1:
  - $f_{PDBUS+/4}$ maximum resolution
  - 3 independent timers/counters
  - Timer/counters can be concatenated
  - 4 operation modes (timer, gated timer, counter, incremental)
  - Seperate interrupt lines
- General Purpose Timer Unit Timer Block 2:
  - $f_{PDBUS+/2}$ maximum resolution
  - 2 independent timers/counters
  - Timer/counters can be concatenated
  - 3 operation modes (timer, gated timer, counter)
  - Extendend capture/reload functions
  - Seperate interrupt lines
- Asynchronous/Sychronous Serial Channel (ASC0)
  with baud rate generator, parity, framing, and overrun error detection
- High Speed Synchronous Serial Cannel (SSC0)
  with baud rate generator, programmable data length and shift direction
- Watchdog Timer with programmable timer events

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 C166/C166S 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.

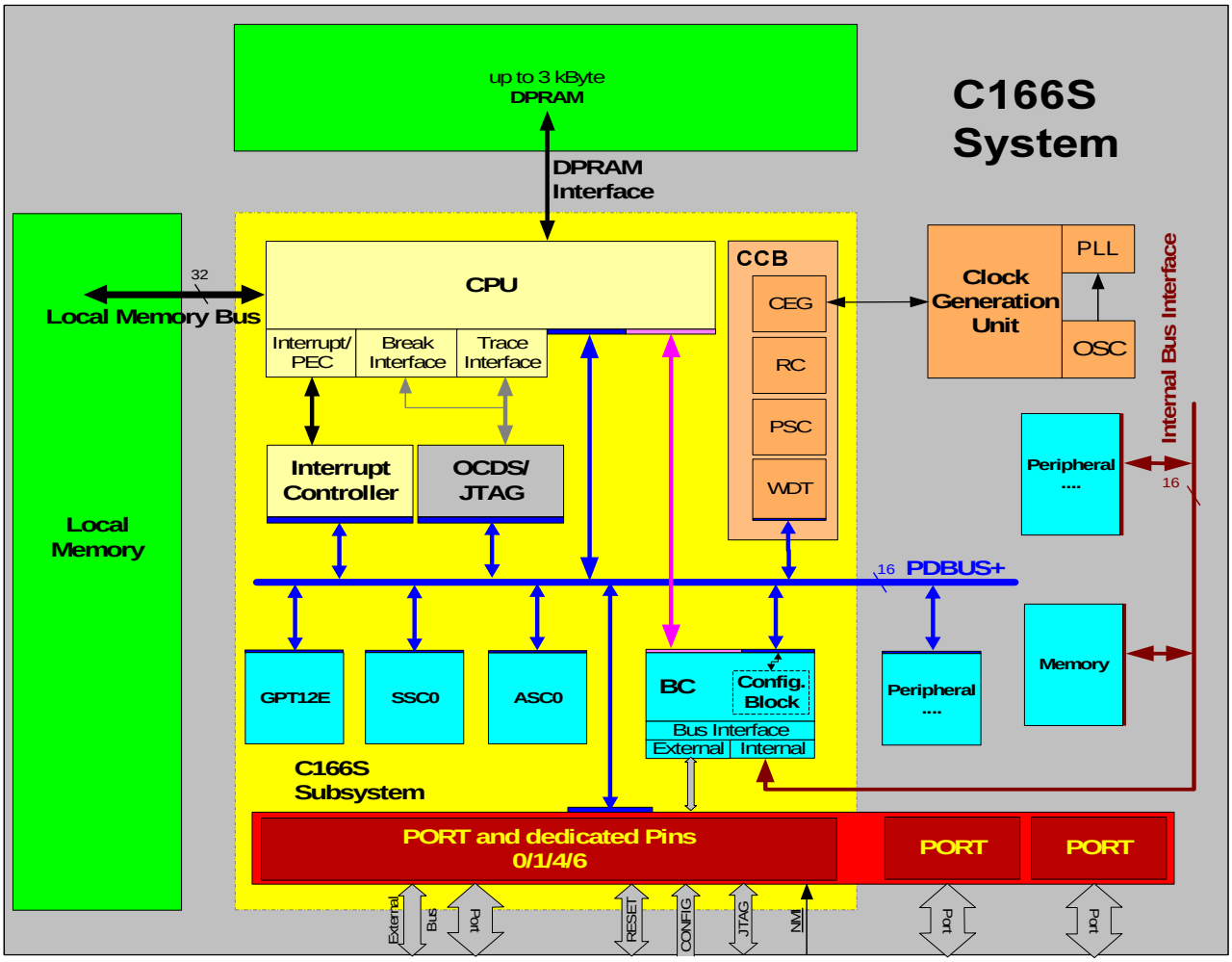
Infineon incorporates its strategic tool partners very early into the product development process, making sure embedded system developers get reliable, well-tuned tool solutions, which help them unleash the power of Infineon microcontrollers in the most effective way and with the shortest possible learning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

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

- Compilers (C, MODULA2, FORTH)
- Macro-Assemblers, Linkers, Locaters, Library Managers, Format-Converters
- Architectural Simulators
- HLL debuggers
- Real-Time operating systems
- VERILOG 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

2 System Overview

The architecture of the C166S combines the advantages of both RISC (Reduced Instruction Set Computing) and CISC (Complex Instruction Set Computing) processors in a very well-balanced way. The sum of the features which are combined results in a high performance microcontroller, which is the right choice not only for today's applications, but also for future engineering challenges. C166S based derivatives does not only integrate a powerful CPU (Central Processing Unit) core and a set of peripheral units into one chip, but also connects the units in a very efficient way. One of the four buses used concurrently on the C166S is the Internal Bus Interface, an internal representation of the external bus interface. This bus provides a standardized method of integrating application-specific peripherals to produce derivatives of the standard C166S. This manual specially describes the C166S Subsystem consists of the CPU, Interrupt Controller (ITC), Bus Controller (BC), On-Chip Debug Support (OCDS) and other system specific peripherals and modules. The following figure shows the principles of a C166S based system.



2.1 Basic CPU Concepts and Mega Core

The main core of the CPU consists of a 4-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.

To meet the demand for greater performance and flexibility, a number of areas has been optimized in the processor core. Functional blocks in the CPU core are controlled by signals from the instruction decode logic. These 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 C166S's instructions can be executed in just one machine cycle, which requires 2 CPU clock cycles T1 and T2 (2 * 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.

A 32-bit / 16-bit division takes 20 CPU clock cycles, a 16-bit $*$ 16-bit multiplication takes 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 (Read Only Memory) or RAM (Random Access Memory) or from the external memory, based on the current Instruction Pointer (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 primarily generated from PLA (Programmable Logic Array) 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 waitstates 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.

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

All standard arithmetic and logical operations are performed in a 16-bit ALU. In addition, for byte operations, signals are provided from bits six and seven of the ALU result to correctly set the condition flags. 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 (Multiply Divide Low Word) and MDH (Multiply Divide High Word), 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 also been provided 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 automatically updated in the PSW (Program Status Word) 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.

2.1.3 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 to move them 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 are also performed in a single machine cycle.

In addition, bit field instructions have been provided, which 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:

• The first solution provides two cycle branch execution after the first iteration of a loop. Thus, only one additional machine cycle is lost during the execution of the entire loop. In loops which fall through upon completion, no additional machine cycles is lost when exiting the loop. No special instructions are required to perform loops, and loops are automatically detected during execution of branch instructions.

• The second loop enhancement allows the detection of the end of a table and 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 this 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.

• The third loop enhancement provides a more flexible solution than the Decrement and Skip on Zero instruction which is found in other microcontrollers. Through the use of Compare and Increment or Decrement instructions, the user can make comparisons to any value. This allows loop counters to cover any range. This is particularly advantageous in table searching.

Saving of system state is automatically performed on the internal system stack 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.

Instructions have also been provided to support indirect branch and call instructions. This supports implementation of multiple CASE statement branching in assembler macros and high level languages.

2.1.4 Consistent and Optimized Instruction Formats

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

1. Provide powerful instructions to perform operations which currently require sequences of instructions and are frequently used. 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.
2. Avoid complex encoding schemes by placing operands in consistent fields for each instruction. Also avoid complex addressing modes which are not frequently used. This decreases the instruction decode time while also simplifying the development of compilers and assemblers.
3. Provide most frequently used instructions with one-word instruction formats. All other instructions are placed into two-word formats. This allows all instructions to be placed on word boundaries, which alleviates the need for complex alignment hardware. It also has the benefit of increasing the range for relative branching instructions.

The high performance offered by the hardware implementation of the CPU can efficiently be utilized by a programmer via the highly functional C166S 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 instruction support the conversion (extension) of bytes to words. A variety of direct, indirect or immediate addressing modes are provided to specify the required operands.

2.1.5 Programmable Multiple Priority Interrupt and PEC System

The following enhancements have been included to allow processing of a large number of interrupt sources:

1. Peripheral Event Controller (PEC): This processor is used to off-load many interrupt requests from the CPU. It avoids the overhead of entering and exiting interrupt or trap routines by performing single-cycle interrupt-driven byte or word data transfers between any two locations with an optional increment of either the PEC source or the destination pointer. Just one cycle is 'stolen' from the current CPU activity to perform a PEC service.

2. Multiple Priority Interrupt Controller (ITC): This controller allows all interrupts to be placed at any specified priority. Interrupts may also be grouped, which provides the user with the ability 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. Once having been accepted by the CPU, an interrupt service can only be interrupted by a higher prioritized service request. For standard interrupt processing, each of the possible interrupt sources has a dedicated vector location.

3. Multiple Register Banks: This feature allows the user to specify up to sixteen general purpose registers located anywhere in the internal DPRAM (Dual Port RAM). A single one-machine-cycle instruction allows to switch register banks from one task to another.

4. Interruptible Multiple Cycle Instructions: Reduced interrupt latency is provided by allowing multiple-cycle instructions (multiply, divide) to be interruptible.

5. Hardware Traps: The C166S also provides an excellent mechanism to identify and to process exceptions or error conditions that arise during run-time, so called 'Hardware Traps'. Hardware traps cause 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). Except for another higher prioritized trap service being in progress, a hardware trap will interrupt any current program execution. In turn, hardware trap services can normally not be interrupted by standard or PEC interrupts.

6. Software Traps: Software interrupts are supported by means of the 'TRAP' instruction in combination with an individual trap (interrupt) number.

2.2 The C166S System Resources

The C166S based subsystem provides 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.

2.2.1 Memory Areas

The memory space of the C166S is configured in a Von Neumann architecture which means that code memory, data memory, registers and I/O 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 additionally been made directly bit addressable.

An up to 3 KByte 16-bit wide internal DPRAM provides fast access to General Purpose Registers (GPRs), user data (variables) and system stack. The DPRAM 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 consisting 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 only restricted by the available DPRAM space. For easy parameter passing, a register bank may overlap others.

A system stack is provided as a 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, STacK OVerflow (STKOV) and STacK UNderflow (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.

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 family with enhanced functionality.

An optional Local Memory is provided for both code and data storage. This memory area is connected to the CPU via a 32-bit-wide local memory bus. Program execution from Local Memory is the fastest of all possible alternatives.

The type of the on-chip Local Memory (Flash/ROM/SRAM/DRAM/none) depends on the chosen derivative.

2.2.2 External Bus Interface

In order to meet the needs of designs where more memory is required than is provided on chip, up to 16 MBytes of external RAM and/or ROM can be connected to the microcontroller via its external bus interface. The integrated Bus Controller (BC) allows to access external memory and/or peripheral resources in a very flexible way.

It can be programmed either to Single Chip Mode when no external memory is required, or to one of four different external memory access modes, which are as follows:

- 16-/18-/20-/24-bit Addresses, 16-bit Data, Demultiplexed
- 16-/18-/20-/24-bit Addresses, 16-bit Data, Multiplexed
- 16-/18-/20-/24-bit Addresses, 8-bit Data, Multiplexed
- 16-/18-/20-/24-bit Addresses, 8-bit Data, Demultiplexed

In the demultiplexed bus modes, addresses are output on PORT1 and data is input/ output on PORT0. In the multiplexed bus modes both addresses and data use PORT0 for input/output.

Important timing characteristics of the external bus interface (Memory Cycle Time, Memory Tri-State Time, Length of ALE, Read Write Delay $\overline{CS}$ and $\overline{WR}$) have been made programmable to allow the user the adaptation of a wide range of different types of memories. In addition, different address ranges may be accessed with different bus characteristics. Up to 5 external $\overline{CS}$ signals can be generated in order to save external glue logic. Access to very slow memories is supported via a particular 'Ready' function.

For applications which require less than 16 MBytes of external memory space, this address space can be restricted to 1MByte, 256 kByte or 64 kByte. In this case Port4 outputs four, two or no address (segment) lines at all. It outputs all 8 address, if an address space of 16 MByte is used.

The on-chip Internal Bus Interface is an internal representation of the external bus and allows to access integrated application-specific peripherals/modules in the same way as external components. It provides a defined interface for these customized peripherals.

2.2.3 The 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. These built in peripherals either allow the CPU to interface with the external world, or provide functions on-chip that otherwise were to be added externally in the respective system. Each functional block processes data independently and communicates information over common buses. Individually selected clock signals are generated for each peripheral.

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 CPU and peripherals is performed through Special Function Registers (SFRs) and interrupts.

Each peripheral contains a set of Special Function Registers (SFRs), which control the functionality of the peripheral and temporarily store intermediate data results. These SFRs are located either within the standard SFR area (00'FE00$_H$-00'FFFF$_H$) or within the extended ESFR area (00'F000$_H$-00'F1FF$_H$). Each peripheral has an associated set of status flags.

Interrupt requests are generated by the peripherals based on specific events which occur during their operation (e.g. operation complete, error, etc.).

For interfacing 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 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 IO pin.

Peripheral Timing

Internal operation of 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 (f_{PDBUS+}) which is gated to the peripherals is independent from the clock signal which 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.

2.2.3.1 Asynchronous / Synchronous Serial Channel (ASC0)

Serial communication with other microcontrollers, processors, terminals or external peripheral components is provided by a Asynchronous / Synchronous Serial Channel. The ASC0 supports full-duplex asynchronous communication up to 3.125 MBaud and half-duplex synchronous communication up to 6.25 MBaud (referred to a PDBUS+ clock of 50 MHz).

A versatile baud rate generator allows to set up all standard baud rates without subsystem clock tuning. For transmission, reception, and erroneous reception three separate interrupt requests are provided.

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 to distinguish address from data bytes has been included (8-bit data + 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.

A loop back option is available for testing purposes.

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

2.2.3.2 High Speed Synchronous Serial Channel (SSC0)

Serial communication with other microcontrollers, processors, terminals or external peripheral components is provided by a High-Speed Synchronous Serial Channel.

The SSC0 allows full duplex synchronous communication up to 25 MBaud in master mode and 12.5 MBaud in slave mode (referred to a PDBUS+ clock of 50 MHz).

A dedicated baud rate generator allows to set up all standard baud rates without subsystem clock tuning. For transmission, reception, and erroneous reception three separate interrupt requests are provided.

The SSC0 transmits or receives characters of 2...16 bits length synchronously to a shift clock which can be generated by the SSC0 (master mode) or by an external master (slave mode). The SSC0 can start shifting with LSB or with MSB. Fully SPI functionality is supported. A loop back option is available for testing purposes.

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

2.2.3.3 General Purpose Timer Unit (GPT12E)

The General Purpose Timer Unit (GPT12E) represents very flexible multifunctional timer structures which may be used for timing, event counting, pulse width measurement, pulse generation, frequency multiplication, and other purposes. They incorporate five 16-bit timers that are grouped into the two timer blocks GPT1 and GPT2. Each timer in each block 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.

Block 1 contains 3 timers/counters with a maximum resolution of $f_{PDBUS+}/4$. The auxiliary timers of GPT1 may optionally be configured as reload or capture registers for the core timer.

Block 2 contains 2 timers/counters with a maximum resolution of $f_{PDBUS+}/2$. An additional CAPREL register supports capture and reload operation with extended functionality.

The following enumeration summarizes all features to be supported:

- Timer Block 1:
  - $f_{PDBUS+}/4$ maximum resolution
  - 3 independent timers/counters.
  - Timers/counters can be concatenated.
  - 4 operating modes (timer, gated timer, counter, incremental).
  - Separate interrupt request lines.

- Timer Block 2:
  - $f_{PDBUS+}/2$ maximum resolution
  - 2 independent timers/counters.
  - Timers/counters can be concatenated.
  - 3 operating modes (timer, gated timer, counter).
  - Extended capture/reload functions via 16-bit Capture/Reload register CAPREL.
  - Separate interrupt request lines.

2.2.4 Parallel Ports (PPorts)

The C166S V1 SubS R1 provides up to 48 I/O lines which are organized into four input/output ports. All port lines are bit-addressable and individually (bit-wise) programmable as inputs or outputs via direction registers. The output driver is disabled when an I/O line is configured as input. This allows true bidirectional ports which are switched to high impedance state when configured as inputs.

Further features like output driver control, input characteristic selection, temperature compensation and output mode selection (open drain or push/pull mode) are not supported by the subsystem´s port module. However, these features can be easily added by the product logic, because they are controlling the PADs directly and have no influence on the port module.

The output drivers´ enable signals are switched asynchronously to inactive level as soon as a subsystem reset occurs. In this case all pins are configured as inputs.

Most 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 A23/19/17...A16 in applications where segmentation is enabled to access more than 64 kBytes of memory.

Port 6 provides optional bus arbitration signals and chip select signals.

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

2.2.5 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.

2.2.6 OCDS and JTAG

The On-Chip Debug Support (OCDS) provides facilities to the debugger in order to emulate resources and assists in application program debug. The main features are:

– real time emulation
– extended trigger capability including: instruction pointer events, data events on address and/or value, external inputs, counters, chaining of events, timers, etc....
– software break support
– break and "break before make" (on IP events only)
– simple monitor mode or JTAG based debugging through instruction injection

The C166S OCDS is controlled by the debugger[1)] through a set of registers accessible from the JTAG interface. The OCDS also receives informations (such as IP, data, status) from the core for monitoring the activity and generating triggers. Finally, the OCDS interacts with the core through a break interface to suspend program execution, and an injection interface to allow execution of OCDS generated instructions.

2.2.7 Core Control Block (CCB)

The Core Control Block supports all central control tasks and all subsystem specific features. The following typical sub-modules are implemented in this unit:

Reset Control (RC)

The reset function is controlled by the reset control unit. The reset block resets the subsystem itself and provides three reset outputs to reset the complete c166S based system according to the reset source.

- Hardware Reset:
  The system enters the reset state immediately (asynchronous to its clock).
- Software Reset (synchronous to the CPU clock)
- Watchdog Timer Reset (synchronous to the CPU clock)

1) Debugger refers to the tool connected to the emulator, and more specifically to the OCDS via the JTAG and which manages the emulation/debugging task.

Power Saving Control (PSC)

The idle mode, and the power down mode mode are supported by the power saving control block. Periodic wakeup from Idle mode combines the drastically reduced power consumption in Idle 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.

Clock Enable Generator (CEG)

The clock enable generator module generates the clock enable signals used by the different clock gates of the subsystem. These clock gates are used for the different clock domains:

- CPU Clock
- Negative CPU Clock
- Peripheral Clock (PDBUS+ clock)

The CPU clock and the negative CPU clock shows the same frequency. However, both clocks have a phase shift of 180°. This behavior is used for running the EBC protocol state machine on two clock edges. Also the protocol of the local memory bus LM66-Bus is based on two clock edges and needs this two clock domains.

The peripheral bus clock is limited to 50 MHz. For not limiting the core to this frequency the peripherals are decoupled from the CPU by their own clock domain. The frequency of the peripheral clock domain is either equal to or half of the CPU clock domain.

For generating all this enable signals, the clock enable generator needs to be supplied by the double frequency of the CPU.

Watchdog Timer (WDT)

The watchdog timer represents one of the fail-safe mechanisms which have been implemented to prevent the controller from malfunctioning. However, the watchdog timer can only detect long term malfunctioning.

The watchdog timer is enabled automatically by setting its enable control line. It is recommended that any product enables the watchdog timer after internal chip initialization. The watchdog timer can only be disabled by software in the time interval until the EINIT (end of initialization) instruction has been executed. Thus, the application startup code is always monitored. The software has to be designed to service the watchdog timer before it overflows. If, due to hardware or software related failures, the software fails to do so, the watchdog timer overflows and generates either an internal subsystem reset, which is indicated on the subsystem boundary (general signals interface) for further product related actions, or triggers an interrupt. Which action will be triggered depends on a new control bit within the WDTCON register.

The Watchdog Timer is a 16-bit timer, which counts the PDBUS+ clock divided either by 2, 4, 128 or 256. The high byte of the Watchdog Timer register can be set to a predefined reload value (stored in WDTREL) in order 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 10 µs and 336 ms (default after reset) can be monitored (referred to a PDBUS+ clock of 50 MHz).

2.2.8 Clock Generation Unit (CGU)

The C166S clock generation unit generates the system clock based on an oscillator or crystal input. A programmable on-chip PLL adds a high flexibility on clock generation to the C166S.

2.2.9 On-chip Bootstrap Loader.

An on-chip bootstrap loader allows to move the start code into internal memories via the serial or other interfaces.

3 Central Processing Unit

The C166S Central Processing Unit (CPU) represents the synthesizable generation of the well-known C166 core family. It has many powerful enhancements while remaining compatible with the C166 family. The new architecture offers a high-performance CPU; fast and efficient access to different kinds of memories; and efficient integration with peripheral units.

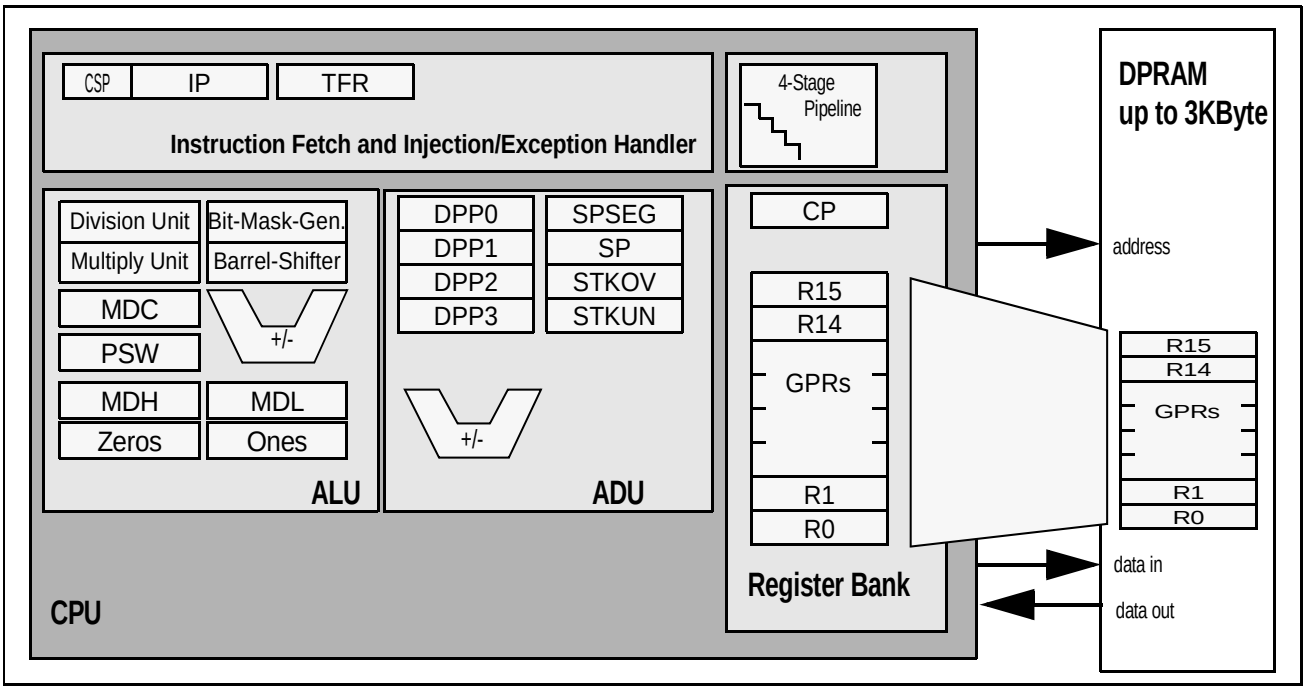


Figure 3-1 CPU architecture

The new core architecture of the C166S results in higher CPU clock frequencies compared to the C166 full custom cores.

C166S has 5 main units that are listed below. All these units have been optimized to achieve maximum performance and flexibility.

- High Performance Instruction Fetch Unit (IFU)
  - High bandwidth fetch interface
  - Instruction FIFO (First In First Out Buffer)
  - High-performance branch-, call-, and loop-processing with instruction flow prediction

- Injection/Exception Handler
  - Handling of interrupt requests
  - Handling of hardware failures

- Instruction PIPeline (IPIP)

- 4-stage execution pipeline

7. Address and Data Unit (ADU)
   - 16-bit arithmetic unit for address generation

8. Arithmetic and Logic Unit (ALU)
   - 8-bit and 16-bit arithmetic unit
   - 16-bit barrel shifter
   - Multiplication and division unit
   - 8-bit and 16-bit logic unit
   - Bit-manipulation unit

3.1 Register Description Format

The C166S contains a set of Special-Function Registers (SFRs) and Extended Special-Function Registers (ESFRs) that are described in the respective chapter of this manual. The example below shows how to interpret the format and notation that are used to describe SFRs and ESFRs.

A word register looks like this:

REG_NAME
Short Description **SFR/ESFR($A16_H$,$A8_H$)** **Reset value: *****$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	bitfield A					0	0	bit C	bit B	bit A
r	r	r	r	r	r	rwh					r	r	rw	rw	rwh

A byte register looks like this:

REG_NAME
Short Description **SFR/ESFR($A16_H$,$A8_H$)** **Reset value: **$_H$**

7	6	5	4	3	2	1	0
0	bitfield A			0	bit C	bit B	bit A
r	rwh			r	rw	rw	rwh

Field	Bits	Type	Description
bitfieldX	[m:n]	type	Description value Function off(Default) value Enable Function 1 ... ...
bitX	[n]	type	Description 0 Function off(Default) 1 Enable Function

Elements:

REG_NAME	Name of this register
bitX	Name of bit
bitfieldX	Name of bitfield
A16 / A8	Long 16-bit address / Short 8-bit address
SFR/ESFR	Register space (SFR or ESFR Register)
(* *) * *	Register contents after reset **0/1** : defined value

	U	: unchanged [undefined (**X**) after power up]
	Y	: defined by reset configuration
[n]	n	: bit number of bit
[m:n]	n	: bit number of first bit of the bitfield
	m	: bit number of last bit of the bitfield
type	**r**	: readable by software
	w	: writable by software
	h	: writable by hardware
value	**0/1**	: defined value,
	X	: undefined,
	0	: reserved for future purpose, read access delivers '0', must not be set to 1

3.2 CPU Special-Function Registers

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

The access mechanism for these CSFRs in the CPU core is identical to the access mechanism for any other SFR. Since all SFRs can be controlled by any instruction that is capable of addressing the SFR/CSFR memory space, there is no need for special system control instructions.

However, to ensure proper processor operations, certain restrictions on the user access to some CSFRs must be applied. For example, the Instruction Pointer (IP) and Code Segment Pointer (CSP) registers cannot be accessed directly at all. They can only be changed indirectly via branch instructions.

The Program Status Word (PSW), Stack Pointer (SP), and Multiply/Divide Control Register (MDC) registers can be modified explicitly by the programmer, and implicitly by the CPU during normal instruction processing.

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

Note: All (C)SFRs may be accessed word-wise, or byte-wise (some of them even bitwise). Reading bytes from word (C)SFRs is a non-critical operation. Any write operation to a single byte of an (C)SFR clears the non-addressed complementary byte within the specified (C)SFR.
Non-implemented (reserved) (C)SFR-Bits cannot be modified, and will always supply a read value of 0. Non-implemented (C)SFR will always supply a read value of $FFFF_H$.

Programming Hints

Access to SFRs

All SFRs reside in dedicated page of the memory space. The following addressing mechanisms allow to access the (C)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 word wide 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 only access 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 C166S's SFRs are marked as Reserved. User software should never write 1s to reserved bits. These bits are currently not implemented and may be used in future products to invoke new functions. In this case, the active state for these functions will be 1, and the inactive state will be 0. Therefore writing only 0s to reserved locations provides portability of the current software to future devices. After read accesses reserved bits should be ignored or masked out.

3.3 Instruction Fetch and Program Flow Control

The C166S can fetch on average one 32-bit or two 16-bit instructions via the 32-bit wide Local Memory Bus (LM-Bus) every machine cycle (which equals two clock cycles T1 and T2) to provide a continuous instruction flow. The instructions can be fetched via this new internal LM-Bus from the internal local memories (ROM, FLASH, OTP, SRAM, DRAM) every clock cycle. A waitstate mechanism allows the CPU to adapt to different kind of memories. For example, this mechanism can be used to:

- Access slower memories
- Generate a power ramp up phase for flash modules
- Stall a DRAM access during the refresh cycles.

Note: Additionally, the LM-Bus provides 16-bit read and write data accesses with and without waitstates to the internal local memory. Furthermore, read protection is provided by the CPU to protect the internal local memories against illegal data accesses.

3.3.1 Branch Target Addressing Modes

The target address and the segment of jump or call instructions can be specified by several addressing modes. The IP register may be updated using relative, absolute, or indirect modes. The CSP register can be updated only by using an absolute value. A special mode is provided to address the interrupt and trap jump vector table, which resides in the lowest portion of the code segment 0.

Table 3-1 Branch Target Addressing Modes

Mnemonic	Target Address	Target Segment	Valid Address Range
caddr	(IP) = caddr	-	caddr = 0000_H...$FFFE_H$
rel	(IP) = (IP) + 2*rel (IP) = (IP) + 2*(rel+1)	- -	rel = 00_H...$7F_H$ rel = 80_H...FF_H
[Rw]	(IP) = (Rw)	-	Rw w = 0...15
seg	-	(CSP) = seg	seg = 0...255(3)
#trap7	(IP) = 0000_H + 4*trap7	(CSP) = 0000_H	trap7 = 00_H...$7F_H$

caddr: Specifies an absolute 16-bit code address within the current segment. Branches **MAY NOT** be taken to odd code addresses. Therefore, the least significant bit of caddr must always contain a 0 or a hardware trap will occur.

rel: This mnemonic represents an 8-bit signed word offset address relative to the current IP contents, which point to the instruction after the branch instruction. Depending on the offset address range, both forward (rel= 00_H to $7F_H$) and backward (rel= 80_H to FF_H) branches are possible. The branch instruction itself is repeatedly executed, when rel = -1 (FF_H) for a word-sized branch

instruction, or rel = -2 (FE_H) for a double-word-sized branch instruction.

[Rw]: In this case, the 16-bit branch target instruction address is determined indirectly by the contents of a word GPR. In contrast to indirect data addresses, indirectly-specified code addresses are **NOT** calculated via additional pointer registers (eg. DPP registers). Branches **MAY NOT** be taken to odd code addresses. Therefore, the least significant bit of the address pointer GPR must always contain a 0 or a hardware trap would occur.

seg: Specifies an absolute code segment number. The C166S supports 256 different code segments, so only the 8 lower bits (respectively) of the 'seg' operand value are used to update the CSP register.

#trap7: Specifies a particular interrupt or trap number for branching to the corresponding interrupt or trap service routine via a jump vector table. Trap numbers from 00_H to $7F_H$ can be specified to access any double-word code location within the address range $00'0000_H$-$00'01FC_H$ in code segment 0 (i.e., the interrupt jump vector table).
For the association of trap numbers with the corresponding interrupt or trap sources, please refer to **Section 3.4** "Interrupt and Trap Functions".

3.3.2 Sequential and Non-Sequential Instruction Flow

Since passing through one pipeline stage takes at least one machine cycle (which equals two clock cycles T1 and T2), any isolated instruction takes at least four machine cycles to be completed. Pipelining, however, allows parallel (i.e., simultaneous) processing of up to four instructions. Therefore, most instructions will seem to be processed during one machine cycle as soon as the pipeline has been filled once after reset.

Pipelining increases the average instruction throughput. In this manual, any execution time specification always refers to the average instruction execution time due to pipelined parallel processing.

The execution time of a sequential and non-sequential instruction flow is mainly given by the instruction fetch from different kind of memories (number of waitstates).

The following pipeline diagram (**Table 3-2**) shows the continuous execution of instruction under the assumption of a fast Local Memory (0/1 waitstate).

Table 3-2 Sequential instruction execution (local memory, 0/1 waitstate)

Clock Cycle	T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2
FETCH	I_n		I_{n+1}		I_{n+2}		I_{n+3}		I_{n+4}		I_{n+6}	
DECODE	I_{n-1}		I_n		I_{n+1}		In+2		I_{n+3}		I_{n+4}	
EXECUTE	I_{n-2}		I_{n-1}		I_n		In+1		I_{n+2}		I_{n+3}	
WRITE BACK	I_{n-3}		I_{n-2}		I_{n-1}		I_n		I_{n+1}		I_{n+2}	
Machine Cycle	**T_m**		**T_{m+1}**		**T_{m+2}**		**T_{m+3}**		**T_{m+4}**		**T_{m+5}**	

The fetch stage fetches instructions from the Local Memory (LM) via the 32-bit LM-Bus. If 16-bit instructions are fetched from the LM-Bus, instructions can be buffered in the 3-word FIFO. The fetch stage always prefetches instructions. If the buffer is filled with instructions, LM-Bus accesses are stopped until the fetched instructions can be loaded into the buffer again.

Table 3-3 shows the standard unconditional branch (branch taken) instruction pipeline, assuming a fast local memory (0/1 waitstates)..

Table 3-3 Unconditional branches (LM-Bus, 0/1 waitstate)

Clock Cycle	T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2
LM-Address						I_{a_t}						
LM-Data 32bit						I_{d_t}	I_{d_t}					
FETCH	I_n		I_{n+1}== branch					I_t		I_{t+1}		I_{t+2}
DECODE	I_{n-1}		I_n		I_{n+1}== branch		-		I_t		I_{t+1}	
EXECUTE	I_{n-2}		I_{n-1}		I_n		I_{n+1}== branch		-		I_t	
WRITE BACK	I_{n-3}		I_{n-2}		I_{n-1}		I_n		I_{n+1}== branch		-	
Machine Cycle	**T_m**		**T_{m+1}**		**T_{m+2}**		**T_{m+3}**		**T_{m+4}**		**T_{m+5}**	

In case of a branch to a 32-bit target instruction, which is not aligned to a 32-bit address, one additional machine cycle (T1,T2) is required.

Table 3-4 shows a standard conditional branch (branch taken) instruction pipeline, assuming a fast local memory (0/1 waitstates).

Table 3-4 Conditional branches (LM-Bus, 0/1 waitstate)

Clock Cycle	T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2
Address								I_{a_t}				
Data 32bit								I_{d_t}	I_{d_t}			
FETCH		I_n		I_{n+1}== branch		(I_{n+2})				I_t		I_{t+1}
DECODE	I_{n-1}		I_n		I_{n+1}== branch		I_{n+1}== branch		-		I_t	
EXECUTE	I_{n-2}		I_{n-1}		I_n		I_{n+1}== branch		I_{n+1}== branch		-	
WRITE BACK	I_{n-3}		I_{n-2}		I_{n-1}		I_n		I_{n+1}== branch		I_{n+1}== branch	
Machine Cycle	**T_m**		**T_{m+1}**		**T_{m+2}**		**T_{m+3}**		**T_{m+4}**		**T_{m+5}**	

Cache Jump Instruction Processing

The C166S 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 (unconditional branch) or two (conditional branch) machine cycles.

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 (and 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 subsequent execution of the same cache jump instruction, the jump target instruction is not fetched from program memory but taken from the cache and immediately injected into the fetch/decode stage of the pipeline (see table below **Table 3-5**).

A time-saving jump on cache is always taken after the second and any subsequent occurrences of the same cache jump instruction, unless an instruction that has 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.

Table 3-5 shows a standard unconditional branch (branch taken and target cached) instruction pipeline, assuming a fast local memory (0/1 waitstates). .

Table 3-5 Unconditional cached branches (LM-Bus, 0/1 waitstate)

Clock Cycle	T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2
LM-Address						I_{a_t+1}						
LM-Data 32bit						I_{d_t+1}	I_{d_t+1}					
FETCH	I_n		I_{n+1}== branch			I_t		I_{t+1}		I_{t+2}		I_{t+3}
DECODE	I_{n-1}		I_n		I_{n+1}== branch		It		I_{t+1}		I_{t+2}	
EXECUTE	I_{n-2}		I_{n-1}		I_n		I_{n+1}== branch		I_t		I_{t+1}	
WRITE BACK	I_{n-3}		I_{n-2}		I_{n-1}		I_n		I_{n+1}== branch		I_t	
Machine Cycle	**T_m**		**T_{m+1}**		**T_{m+2}**		**T_{m+3}**		**T_{m+4}**		**T_{m+5}**	

Table 3-6 shows a standard conditional branch (branch taken and target cached) instruction pipeline, assuming a fast local memory (0/1 waitstates).

Table 3-6 Conditional cached branches (LM-Bus, 0/1 waitstate)

Clock Cycle	T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2	T_1	T_2
Address								I_{a_t+1}				
Data 32bit								I_{d_t+1}	I_{d_t+1}			
FETCH		I_n		$I_{n+1==branch}$		(I_{n+2})		I_t		I_{t+1}		I_{t+2}
DECODE	I_{n-1}		I_n		$I_{n+1==branch}$		$I_{n+1==branch}$		I_t		I_{t+1}	
EXECUTE	I_{n-2}		I_{n-1}		I_n		$I_{n+1==branch}$		$I_{n+1==branch}$		I_t	
WRITE BACK	I_{n-3}		I_{n-2}		I_{n-1}		I_n		$I_{n+1==branch}$		$I_{n+1==branch}$	
Machine Cycle	**T_m**		**T_{m+1}**		**T_{m+2}**		**T_{m+3}**		**T_{m+4}**		**T_{m+5}**	

3.3.3 ATOMIC and EXTended Instructions

ATOMIC and EXTended instructions (ATOMIC, EXTR, EXTP, EXTS, EXTPR, EXTSR) disable the standard and PEC interrupts and class A traps until the completion of the next sequence of instructions. The number of instructions in the sequence may vary from 1 to 4. The instruction number is coded in the 2-bit constant field #irang2 and takes values from 0 to 3. The EXTended instructions additionally change the addressing mechanism during this sequence (see instruction description).

ATOMIC and EXTended instructions become active immediately, so no additional NOP instructions are required. All instructions requiring multiple cycles or hold states for execution are considered as one instruction. ATOMIC and EXTended instructions can be used with any instruction type.

Note: If a class B trap interrupt occurs during an ATOMIC or EXTended sequence, then the sequence is terminated, an interrupt lock is removed, and the standard condition is restored before the trap routine is executed. The remaining instructions of the terminated sequence that are executed after returning from the trap routine will run under standard conditions.

Note: Certain precautions are required when using nested ATOMIC and EXTended instructions. There is only one counter to control the length of the sequence, i.e., issuing an ATOMIC or EXTended instruction within a sequence will reload the counter with the value of the new instruction.

3.3.4 Code Addressing via Code Segment and Instruction Pointer

The C166S provides a total addressable memory space of 16 MBytes. This address space is arranged as 256 segments of 64 KBytes each. A dedicated 24-bit code address pointer is used to access the memories for instruction fetches. This pointer has two parts: An 8-bit Code Segment Pointer (CSP), and a 16-bit offset Instruction Pointer (IP). The concatenation of the CSP and IP results directly in a correct 24-bit physical memory address.

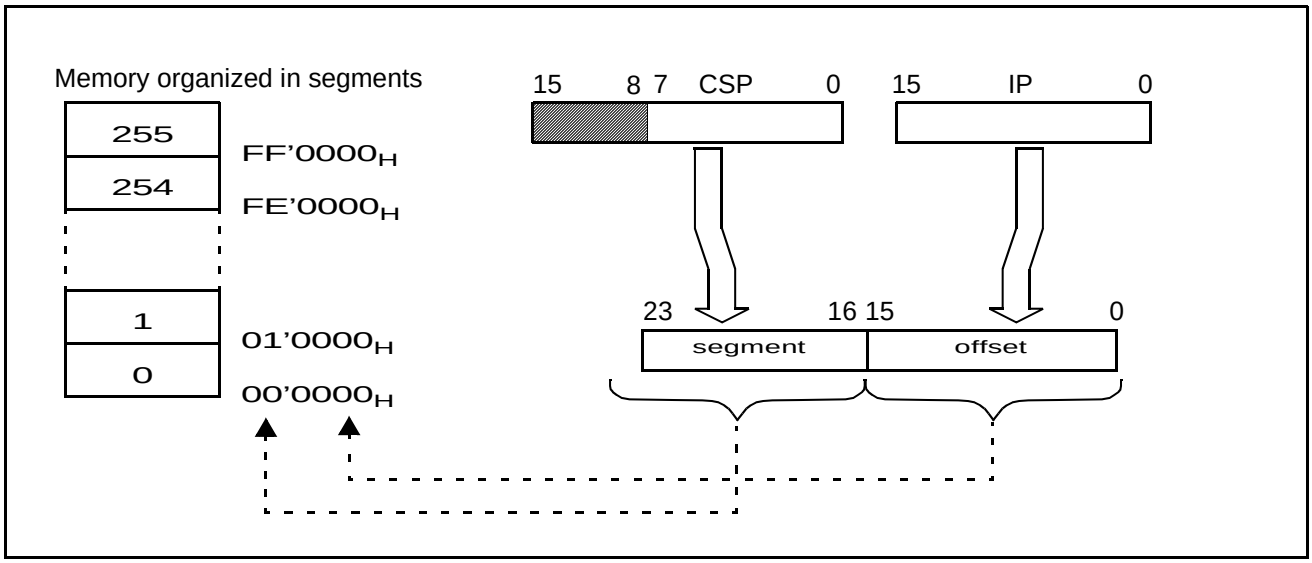
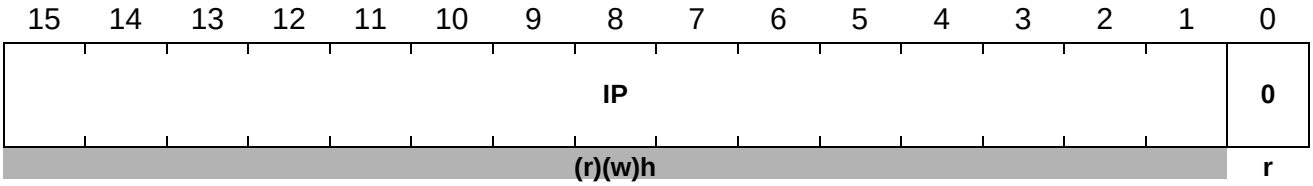


Figure 3-2 Addressing via the Code Segment- and Instruction Pointers

The 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 C166S's address space, and thus it is not directly accessible by the programmer. The IP can be modified indirectly by return instructions via the stack. The IP register is updated implicitly by the C166S for branch instructions and after instruction fetch operations.

IP
Instruction Pointer **($----_H$,$--_H$)** **Reset value: 0000_H**

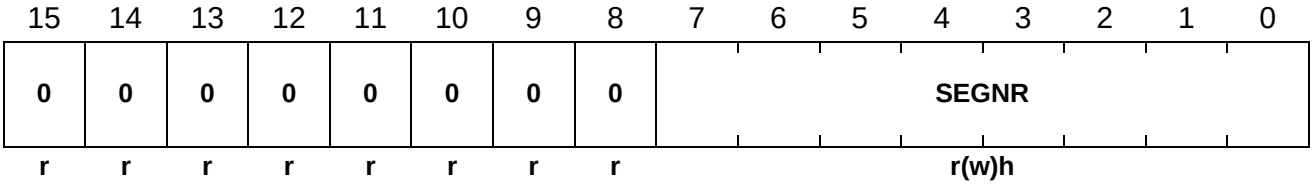


Field	Bits	Type	Description
IP	[15:1]	rwh	Specifies the intra-segment offset from which the current instruction is to be fetched; IP refers to the current segment <SEGNR>.
0	[0]	r	IP is always word-aligned

The 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 256 segments of 64 KBytes each, while the higher 8 bits are reserved for future use.

CSP
Code Segment Pointer **SFR($FE08_H$,04_H)** **Reset value: $000x^{1)}_H$**



1) The reset value of the bitfield segnr[1:0] is product-specific. With an alternate boot mode feature, the code execution can be started at different segments after reset.

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

The actual code memory address is generated by direct extension of the 16-bit contents of the IP register by the lower byte of the CSP register, as shown in **Figure 3-2**.

There are two modes: Segmented and non-segmented. The mode is selected with the **SGTDIS** bit in the SYSCON register. After reset, the segmented mode is selected.

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	SYS CON9	SYS CON8	SYS CON7	SYS CON6	SYS CON5	SYS CON4	SYS CON3	SYS CON2	SYS CON1	SYS CON0
rw			rw	rw	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
SGTDIS	11	rw	Segmentation Disable/Enable Control 0 Segmentation enabled (CSP is saved/restored during interrupt entry/exit) 1 Segmentation disabled (Only IP is saved/restored)

*Note: For summary of the SYSCON register please refer to **Section 3.3.5**.*

Segmented Mode

The CSP register can be only read and may not be written by data operations. The CSP is modified either directly by the JMPS and CALLS instructions, or indirectly via the stack by the RETS and RETI instructions. Upon the acceptance of an interrupt or the execution of a software TRAP instruction, the CSP register is automatically loaded with the segment address of the vector location.

Non-Segmented Mode

In the non-segmented mode, the CSP is fixed to segment 0. It is no longer possible to modify the CSP either directly by the JMPS or CALLS instructions, or indirectly via the stack by the RETS (RETI) instruction. For non-segmented memory mode, the contents of this register are not significant, because all code accesses are restricted automatically to segment 0.

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

3.3.5 The CPU/System Configuration Register SYSCON

This register is used to configure the C166S. It is bit-addressable and provides general system configuration and control functions. The reset value of register SYSCON depends on the state of the configuration inputs during reset.

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	SYSCON9	SYSCON8	SYSCON7	SYSCON6	SYSCON5	SYSCON4	SYSCON3	SYSCON2	SYSCON1	SYSCON0
rw			rw	rw	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
SYSCON0 ..... SYSCON9	0 9	rwh	SYStem CONfiguration
ROMEN	10	rwh	**Internal ROM ENable** (Set according to pin EA during reset) 0 Internal local memory disabled: Accesses to the Local memory area use the external bus 1 Internal local memory enabled
SGTDIS	11	rw	SeGmenTation DISable/enable control 0 Segmentation enabled (CSP is saved/restored during interrupt entry/exit) 1 Segmentation disabled (Only IP is saved/restored)
ROMS1	12	rw	Internal local memory mapping 0 Internal local memory area mapped to segment 0 ($00'0000_H$...$00'7FFF_H$) 1 Internal Local Memory area mapped to segment 1 (01'0000H...01'7FFFH)
STKSZ	[15:13]	rw	System STacK SiZe Selects the size of the system stack (in the internal DPRAM) from 32 to 1536 words

Note: The CPU SYSCON bits cannot be changed after execution of the EINIT instruction.

3.4 Interrupt and Exception Execution

An Interrupt and Exception Handler is responsible for managing all system and core exceptions. There are four different kinds of exceptions that are executed in a similar way:

- Interrupts generated by the InTerrupt Controller (ITC)
- DMA transfers issued by the Peripheral Event Controller (PEC).
- Software traps caused by the TRAP instruction
- Hardware traps issued by faults or specific system states

Normal Interrupt Processing

The CPU temporarily suspends the current program execution and branches to an Interrupt Service Routine (ISR) in order to service an interrupt-requesting device. The current program status [Instruction Pointer (IP), Processor Status Word (PSW), and in segmentation mode, the Code Segment Pointer (CSP)] is saved on the internal system stack. A prioritization scheme with 16 priority levels and with 4/8 sub-levels (4/8 group levels) specifies the order of multiple interrupt-request handling. The maximum number of interrupt requests is 112 (configured in steps of 16), wherein the lowest priority level is reserved for the CPU and cannot be used for interrupts.

Software and Hardware Traps

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 ($\overline{\text{NMI}}$) pin. Several hardware trap functions are provided for handling erroneous conditions and exceptions that arise during the program execution. Hardware traps always have highest priority and cause immediate system reaction. The software trap function is invoked by the TRAP instruction, which generates a software interrupt for a specified interrupt vector. For all types of traps, the current program status is saved in the system stack.

Interrupt Processing via the Peripheral Event Controller

A faster alternative to normal interrupt processing is servicing an interrupt requesting device by the C166S's integrated PEC. Triggered by an interrupt request, the PEC performs a single-word or byte data transfer between any two memory locations through one of up to 16 programmable PEC service channels. During a PEC transfer, the normal program execution of the CPU is halted. No internal program status information needs to be saved. The same prioritization scheme is used for PEC service as for normal interrupt processing.

3.4.1 Interrupt System Structure

The C166S provides up to 112 separate interrupt nodes that may be assigned to 128 arbitration priority levels with 16 interrupt priority groups and 4/8 priorities inside each group. 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 interrupt vector. The control register contains an interrupt request flag, Interrupt Enable (IE) bit, and interrupt priority of the associated source. Each source request is activated by one specific event, depending on the selected operating mode of the respective device. In some cases, the multi-source interrupt nodes are incorporated for efficient use of system resources. These nodes can be activated by several source requests.

The C166S provides a vectored interrupt system. This system reserves specific vector locations in the memory space 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. The reserved vector locations build a jump table in the C166S's address space.

The arbitration winner is sent to the CPU together with its priority level and action request. The CPU triggers the corresponding action, which depends on the required functionality (normal interrupt, PEC etc.) of the arbitration winner.

An action request will be accepted by the CPU if the requesting source has a higher priority than the current CPU priority level, and if interrupts are globally enabled. If the requesting source has a lower (or equal) interrupt level priority, then the requested interrupt stays pending.

3.4.2 Interrupt Arbitration

The C166S interrupt arbitration system can handle interrupt requests from up to 112 sources. Interrupt requests may be triggered either by the C166S peripherals or by external inputs. The "End of PEC" interrupt for supporting enhanced PEC functionality is connected internally to one of the interrupt request lines.

The arbitration process starts by an enabled interrupt request and stays active for as long as interrupt request is pending. If nothing is pending then the arbitration logic switches to the idle state to save power.

Each interrupt request line is controlled by its interrupt control register xxIC (here and below, 'xx' stands for the mnemonic of the respective interrupt source). An interrupt request event sets the interrupt request flag to 1 in the corresponding interrupt control register (bit xxIC.xxIR). The interrupt request can also be triggered by the software if the program sets the respective interrupt request bit. This feature is used by operating systems.

If the request bit has been set and this interrupt request is enabled by the IE bit of the same control register (bit xxIC.xxIE), then an arbitration cycle starts on the next clock

cycle. However, if an arbitration cycle is currently in progress, the new interrupt request will be delayed till the next arbitration cycle. If an interrupt request (or PEC request) is accepted by the core, the respective interrupt request flag is cleared automatically.

All interrupt requests that are pending at the beginning of a new arbitration cycle are considered simultaneously. Within the arbitration cycle, the arbitration is independent of the actual request time.

C166S uses a two-stage interrupt prioritization scheme for interrupt arbitration, as shown in **Figure 3-3**.

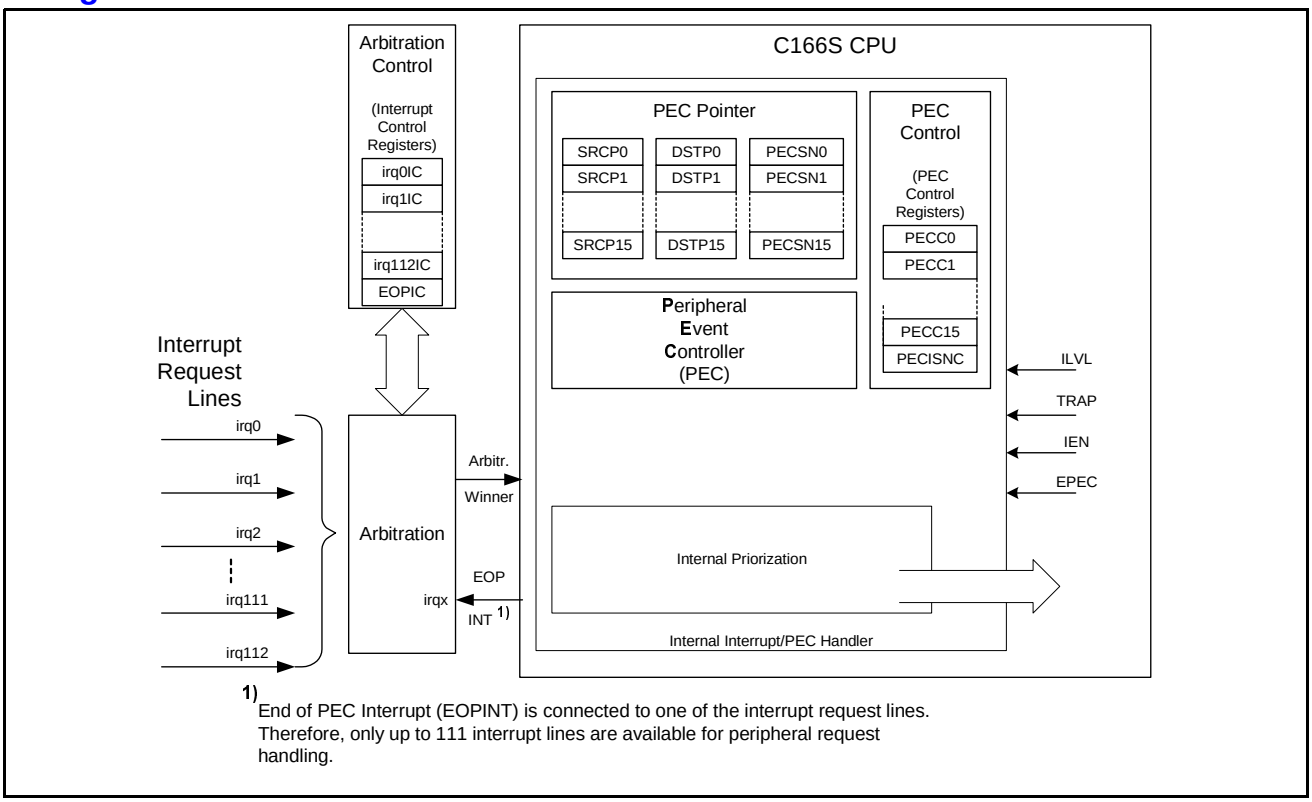


Figure 3-3 Interrupt Arbitration

The first arbitration stage compares up to 128 priority levels of interrupt request lines. The priority level of each request consists of Interrupt priority LeVeL (ILVL) and Group priority LeVeL (GLVL). An interrupt priority level is programmed for each interrupt request line by the 4-bit bitfield ILVL of the respective xxIC register. The group priority level is programmed for each interrupt request line by the 2-bit bitfield GLVL and the group extension bit xxGP of the register xxIC.

Note: All interrupt request sources that are enabled and programmed to the same ILVL must have different group priority levels. Otherwise, an incorrect interrupt vector may be generated.

In the second arbitration stage, the priority level of the first-stage winner is compared with the priority of the current CPU task. An action request will be accepted by the CPU if the requesting source has a higher priority level than the current CPU priority level (bits ILVL of the PSW register), and if interrupts are enabled globally by the global IEN flag in

PSW. The CPU denies all requests in case of a cleared IEN flag. If the requester has a lower or equal priority level than current CPU task, the request stays pending.

Note: Priority level 0000_B is the default level of the CPU. Therefore, a request on ILVL 0000_B will be arbitrated, but the CPU will never accept an action request on this level. However, every enabled interrupt request (including all denied interrupt requests - also priority level 0000_B requests) triggers a CPU wake-up from idle state independent of the setting of the global interrupt enable bit PSW.IEN.

Note: The first 16 trap numbers are reserved for the CPU traps. The first usable interrupt trap number starts with 10_H. Therefore, the number of interrupt nodes is limited to 112.

All interrupt control registers are organized identically. The lower 8 bits of an interrupt control register contain the complete interrupt control and status information of the associated source, which is required during one round of prioritization (arbitration cycle). 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. Therefore, each interrupt source can be programmed or modified with just one instruction. When reading the interrupt control registers with instructions that operate with word data types, the upper 8 bits (15...8) will return zeros. Zeros should always be written to these bit positions. The layout of the interrupt control registers shown below is applicable to all xxIC registers.

xxIC
Interrupt Control Register **bSFR($xxxx_H$,xx_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	xxGP	xxIR	xxIE	ILVL				GLVL	
r	r	r	r	r	r	r	rw	rwh	rw	rw				rw	

Field	Bits	Type	Description
xxGP	[8]	rw	Group Priority Extension Defines the value of high-order group level bit
xxIR[1)]	[7]	rwh	Interrupt Request Flag 0 No request pending 1 This source has raised an interrupt request
xxIE	[6]	rw	**Interrupt Enable Control Bit** (individually enables/disables a specific source) 0 Interrupt request is disabled 1 Interrupt request is enabled

Field	Bits	Type	Description
ILVL	[5:2]	rw	Interrupt Priority Level F_H Highest priority level ... ... 0_H Lowest priority level
GLVL	[1:0]	rw	Group Priority Level Defines the internal order for simultaneous requests of the same priority.

1) Bit xxIR supports bit-protection

The arbitration scheme allows nesting of up to 15 ISRs of different priority levels (level 0 cannot be used; see note above).

Note: When no interrupt request is active, arbitration is stopped to reduce power consumption.

3.4.3 Interrupt Vector Table

The C166S has a vectored interrupt system. This system reserves the specific vector locations in the memory space 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 vector position directly identifies the source that caused the request.

Note: The Class B hardware traps all share the same interrupt vector. The status flags in the Trap Flag Register (TFR) are used to determine which exception caused the trap. For details, see ***Section 3.4.5.2*** *"Hardware Traps".*

The reserved vector locations are assembled into a jump table that is located in the C166S's address space. The jump table contains the appropriate jump instructions that transfer control to the interrupt or trap service routines. These routines may be located anywhere within the address space. The vector table is located at the bottom in segment 0 of the address space. Each entry occupies 2 words to provide space for long jumps, except for the reset vector and the hardware trap vectors, which occupy 4 or 8 words.

Each vector location has an offset address to the segment base address ($00'0000_H$) of the vector table. The offset address can be calculated easily. The offset is the trap number shifted by 2.

3.4.4 Interrupt Control Functions in the Processor Status Word

The PSW is divided functionally into 2 parts. The lower byte of the PSW represents the arithmetic status of the CPU; the upper byte of the PSW controls the interrupt system of the C166S.

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	PSW S1	0	0	0	USR0	MUL IP	E	Z	V	C	N
rwh				rw	rwh	r	r	r	rw	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
ILVL	[15:12]	rwh	CPU Priority Level 0_H Lowest Priority ... ... F_H Highest Priority
IEN	[11]	rw	Interrupt/PEC Enable Flag (globally) 0 Interrupt/PEC requests are disabled 1 Interrupt/PEC requests are enabled

Note: For a summary of the PSW register, please refer to ***Section 3.7.6***

CPU Priority ILVL defines the current level for the CPU operation, i.e., this bit field reflects the priority level of the routine currently being executed. When the CPU enters an ISR, this bitfield is set to the priority level of the request that is being serviced. The previous PSW is saved in the system stack before entering the ISR. To be serviced, any interrupt request must have a higher priority level than the current CPU priority level. Any request of the same or a lower level will not be acknowledged. The current CPU priority level may be adjusted via software to select interrupt request sources that can be serviced.

PEC transfers do not really interrupt the CPU, but rather "steal" some CPU cycles, so PEC services do not influence the ILVL field in the PSW.

Hardware traps set the CPU level to the maximum priority (i.e., 15). Therefore, no interrupt or PEC requests will be acknowledged while an exception trap service routine is executed.

The TRAP instruction does not change the CPU level, so software trap service routines may be interrupted by higher-level requests.

Interrupt ENable flag IEN globally enables or disables interrupts and PEC operations. When IEN is cleared, no new interrupt requests are accepted by the CPU after IEN was set to 0. However, the requests that have already entered the pipeline will be completed. If IEN is set to 1, all interrupt sources are globally enabled.

Note: To generate requests, interrupt sources must be also enabled by the IEN bits in their associated control registers.

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

3.4.4.1 Saving the Status during Interrupt Service

Before an operating system or ITC can actually service a task switch request or interrupt, the CPU must save the current task status. The C166S saves the CPU status (PSW) along with the return address in the system stack. The return address defines the point where the execution of the interrupted task is to be resumed after returning from the service routine. This return address is specified by the IP and, in the case of a segmented memory mode, also by the CSP. Bit SGTDIS in the SYSCON register defines which mode is used and, therefore, controls how the return address is stored.

In the case of non-segmented mode, the system stack stores the PSW first and then the IP. In the segmented mode, PSW is followed by CSP and the IP. This order optimizes the use of the system stack if segmentation is disabled.

The CPU priority field (ILVL in PSW) is updated with the priority of the interrupt request that is to be serviced, so the CPU now executes on the new level.

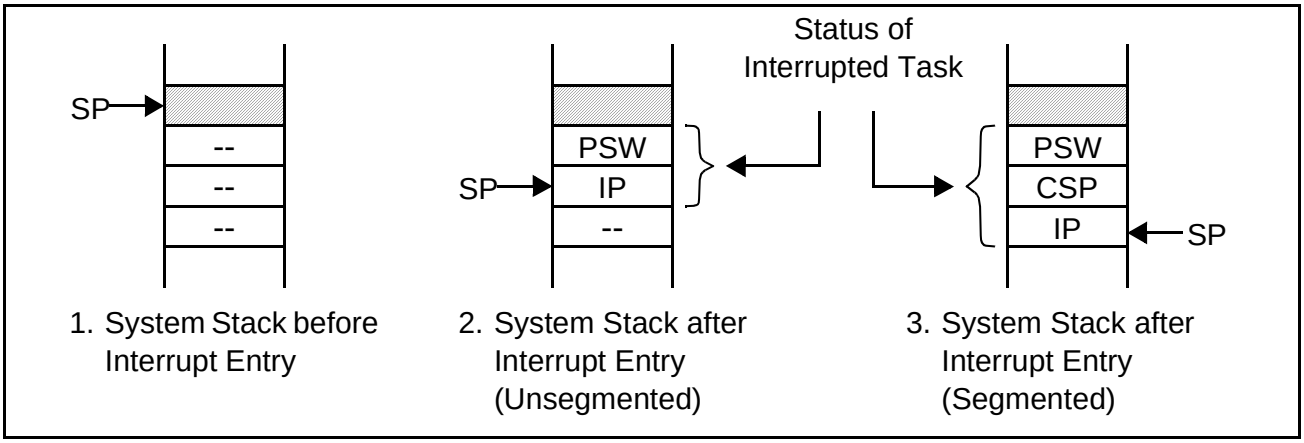


Figure 3-4 Task Status saved on the System Stack

After accepting an interrupt request, the C166S sends an acknowledgement to the ITC that the requested interrupt is being serviced. The vector associated with the requesting source is loaded into the IP and CSP, and the first instruction of the service routine is fetched. All other CPU resources such as data page pointers and the context pointer are not affected.

When the CPU returns from the ISR [RETurn from Interrupt (RETI) is executed], the status information is "popped" from the system stack in reverse order. The status information contents depend on the SGTDIS bit value.

3.4.4.2 Context Switching

An ISR 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 wasted by saving and restoring.

The C166S makes it possible to switch the complete register 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 contents of the Context Pointer (CP) into the system stack and loads the CP with the immediate value "New_Bank". The new CP value sets a new register bank. The service routine may now use its own registers. This register bank is preserved when the service routine is terminated, i.e., its contents are available for the next call. Before returning (RETI), the previous CP is simply popped from the system stack, which returns the registers to the original register bank.

Note: Resources that are used by the interrupting program must eventually be saved Pointers (DPP) and the registers of the multiply and divide unit.

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

3.4.5 Traps

3.4.5.1 Software Traps

The TRAP instruction is used to cause a software call to an ISR. The trap number that is specified in the operand field of the trap instruction determines which vector location of the vector table will be used.

The TRAP instruction's effect is similar to that of an interrupt request that uses the same vector. PSW, CSP (in segmentation mode), and IP are pushed into the system stack and then a jump is taken to the specified vector location. When a software trap is executed, the CSP for the trap service routine is loaded with segment address 0. No Interrupt Request flags are affected by the TRAP instruction. The ISR called by a TRAP instruction must be terminated with a RETI instruction to ensure correct operation.

Note: The CPU priority level is not modified by the TRAP instruction, so the service routine is executed with the same priority level as the interrupt task. Therefore, the service routine entered by the TRAP instruction can be interrupted by other traps or by higher priority interrupts, other than when triggered by a real hardware event.

3.4.5.2 Hardware Traps

Hardware traps are issued by faults or specific system states that occur during runtime (not identified at assembly time). The C166S distinguishes eight different hardware trap functions. When a hardware trap condition has been detected, the CPU branches to the trap vector location for the respective trap condition. The instruction that caused the trap event is either completed or canceled before the trap-handling routine is entered.

Hardware traps are not-maskable and always have a higher priority than any other CPU task. If several hardware trap conditions are detected within the same machine cycle, the highest-priority trap is serviced. In the case of a hardware trap, the injection unit injects a TRAP instruction into the pipeline. The TRAP instruction performs the following actions:

- Push PSW, CSP (in segmented mode) and IP onto the system stack
- Set CPU level in the PSW register to the highest possible priority level, which disables all interrupts and DMA transfers
- Branch to the trap vector location specified by the trap number of the trap condition

The eight hardware functions of the C166S are divided in two Classes.

Class A traps are:

– External NMIs
– Stack overflow
– Stack underflow

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

Class B traps are:

– Undefined opcode
– Protection fault
– Illegal word operand access
– Illegal instruction access
– Illegal external bus access

The Class B traps share the same interrupt node and interrupt vector. The bit-addressable Trap Flag Register (TFR) allows a trap service routine to identify the trap that caused the exception.

The Trap Flag Register TFR

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.

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	0	0	0	0	0	UND OPC	0	0	0	PRT FLT	ILL OPA	ILL INA	ILL BUS
rwh	rwh	rwh	r	r	r	r	r	rwh	r	r	r	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
NMI[1)]	[15]	rwh	Non-Maskable Interrupt flag 0 No non-maskable interrupt detected 1 Non-maskable interrupt detected
STKOF[1)]	[14]	rwh	STacK OverFlow flag 0 No stack overflow event detected 1 Stack overflow event detected
STKUF[1)]	[13]	rwh	STacK UnderFlow flag 0 No stack underflow event detected 1 Stack underflow event detected
UNDOPC[1)]	[7]	rwh	UNDefined OPCode 0 No undefined opcode event detected 1 Undefined opcode event detected
PRTFLT[1)]	[3]	rwh	PRoTection FauLT 0 No protection fault event detected 1 Protection fault event detected

Field	Bits	Type	Description
ILLOPA[1)]	[2]	rwh	ILLegal word OPerand Access 0 No illegal word operand access event detected 1 Ilegal word operand access event detected
ILLINA[1)]	[1]	rwh	ILLegal INstruction Access 0 No illegal instruction access detected 1 A branch to an odd address has been attempt.
ILLBUS	[0]	rwh	ILLegal External BUS Access 0 No illegal external bus access detected 1 An external access has been attempted with no bus defined.

1) This bit supports bit protection

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.

The reset functions (hardware, software, watchdog) may be also regarded as a type of trap. Reset functions have the highest trap priority (trap priority IV). The Debug trap has the second-highest trap priority (trap priority III), followed by the third-highest trap priority traps, Class A traps (trap priority II), and then by Class B traps (trap priority I). So the Debug trap can interrupt a Class A and B trap and a Class A trap can interrupt a Class B trap. The Debug trap is a special kind of interrupt-service channel for debug purposes whose priority lies between the Class A trap and the reset function. This allows the debugger to interrupt hardware traps and hardware interrupts

Exception Condition	Trap Flag	Trap Vector	Trap Number	Trap Priority
Reset Functions:				
Hardware Reset		RESET	00_H	IV
Software Reset		RESET	00_H	IV
Watchdog Timer Overflow		RESET	00_H	IV
Debug Trap	DEBUG	DEBTRAP	08_H	III

Exception Condition	Trap Flag	Trap Vector	Trap Number	Trap Priority
Class A Hardware Traps:				
Non-Maskable Interrupt	NMI	NMITRAP	02_H	II.3
STacK OverFlow	STKOF	STOTRAP	04_H	II.2
STacK UnderFlow	STKUF	STUTRAP	06_H	II.1
SOFTware BReaK	SOFTBRK	SBRKTRAP	08_H	II.0
Class B Hardware Traps:				
UNDefined OPCode	UNDOPC	BTRAP	$0A_H$	I
PRoTection FauLT	PRTFLT	BTRAP	$0A_H$	I
ILLegal word Operand Access	ILLOPA	BTRAP	$0A_H$	I
ILLegal INstruction Access	ILLINA	BTRAP	$0A_H$	I
ILLegal external BUS access	ILLBUS	BTRAP	$0A_H$	I

Class A Trap

Class A traps are generated by the high-priority system NMI or by special CPU events such as a software break, or a stack overflow or underflow event. Class A traps are not used to indicate hardware failures. After a Class A event, a dedicated service routine is called to react to the events. Each Class A trap has its own vector location in the vector table. After finishing the service routine, the remainder of the instruction flow must be executed correctly. This explains why Class A traps cannot interrupt atomic/extend sequences.

In case of an atomic/extend sequence, the execution continues until sequence completion. Upon completion, the IP of the instruction following the last executed one is pushed onto the stack.

If more than one Class A trap occurs at a same time, they are prioritized internally. The NMI trap has the highest priority, and the stack underflow trap has the lowest.

Note: When two different Class A traps occur simultaneously, both trap flags are set. The trap with the higher priority is executed. After return from the service routine, the IP is popped from the stack and immediately pushed again because of the other pending Class A trap (unless the second trap flag in TFR has been cleared by the first trap service routine).

External NMI Trap (NMI)

Whenever a high-to-low transition on the dedicated $\overline{\text{NMI}}$ 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 $\overline{\text{NMI}}$ is sampled with every CPU clock cycle to detect transitions.

STacK OverFlow Trap (STKOF)

Whenever the stack pointer SP is decremented to a value 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 first or second instruction after the instruction following the subtract instruction.

For recovery from stack overflow, there must be enough excess space on the stack for saving the current system state (PSW; IP; and, in segmented mode, the CSP) twice. Otherwise, a system reset should be generated.

STacK UnderFlow Trap (STKUF)

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 first or second instruction after the instruction following the add instruction.

Class B Trap

Class B traps are generated by unrecoverable hardware failures. In case of hardware failure, the CPU must immediately start a failure service routine. Class B traps can interrupt an atomic/extend sequence. **After finishing a Class B service routine, the interrupted instruction flow cannot be restored.**

Note: If a Class A trap and a Class B occur simultaneously, both trap flags are set. If this occurs during execution of an atomic/extend sequence, then the presence of the Class B trap breaks the protection of atomic/extend operations, and the Class A trap will be executed immediately without waiting for the sequence completion. After return from the service routine, the IP is popped from the system stack and immediately pushed again because of the other pending Class B trap. In this situation, the interrupted instruction flow cannot be restored.

All Class B traps have the same trap priority (trap priority I). When several Class B traps are active at the same time, the corresponding flags in the TFR are set, and the trap service routine is entered. Since all Class B traps have the same vector, the priority of service of Class B that occur simultaneously is determined by the software in the trap service routine.

During the execution of a Class A trap service routine, any Class B trap will not be serviced until the Class A trap service routine is exited with a RETI instruction. In this case, the Class B trap condition is stored in the TFR but the IP value of the instruction that caused this trap will be lost.

UNDefined OPCode Trap (UNDOPC)

When the instruction currently decoded by the CPU does not contain a valid C166S opcode, the UNDOPC flag is set in the 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 (PRTFLT)

DISWDT, EINIT, IDLE, PWRDN, SRST, and SRVWDT are protected instructions. Whenever one protected instruction is executed and the protection is broken, the PRTFLT flag in register TFR is set and the CPU enters the protection fault trap routine. The IP value pushed onto the system stack for the protection fault trap is the address of the instruction that caused the trap.

ILLegal word OPerand Access Trap (ILLOPA)

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 that caused the trap.

ILLegal INstruction Access Trap (ILLINA)

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 (ILLBUS)

Whenever the CPU requests an external instruction fetch or a data read or a 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 that caused the trap.

3.4.6 Peripheral Event Controller

The Peripheral Event Controller (PEC) "decides" which CPU action is required to manage an interrupt request. It may be either normal interrupt service, or fast data transfer between two memory locations. The C166S PEC controls up to sixteen[1)] fast data transfer channels.

If a normal interrupt is requested, the CPU temporarily suspends the current program execution and branches to an Interrupt Service Routine (ISR). The current program status and context must be preserved.

If a PEC channel is selected for servicing an interrupt request, a single word or byte data transfer between any two memory locations is to be performed. During a PEC transfer, the normal program execution of the CPU is halted for just 1 machine cycles. No internal program status information needs to be saved. The PEC transfer is the fastest possible interrupt response. In many cases, a PEC transfer is sufficient to service the peripheral request (serial channels, for example).

The PEC channels can perform the following actions:

- Byte or word transfer
- Continuous data transfer
- PEC channel-specific interrupt request upon data transfer completion; or for all channels a common End of PEC (EOP) interrupt for enhanced handling
- Automatic increment of source or destination pointers
- Channel linking of two PEC channels

Note: PEC transfer is executed if its priority level is higher than current CPU priority level.

1) The number of PEC channels depends on the configuration of the product. Please refer to the product User Manual.

3.4.6.1 The PEC Source and Destination Pointers

The PEC channels' source and destination pointers specify the locations between which the data is to be moved. All pointers are 24 bits wide. The 24-bit source address is stored in the internal DPRAM location SRCPx (lower 16 bits of address) and in the low byte of register PECSNx (highest 8 address bits).

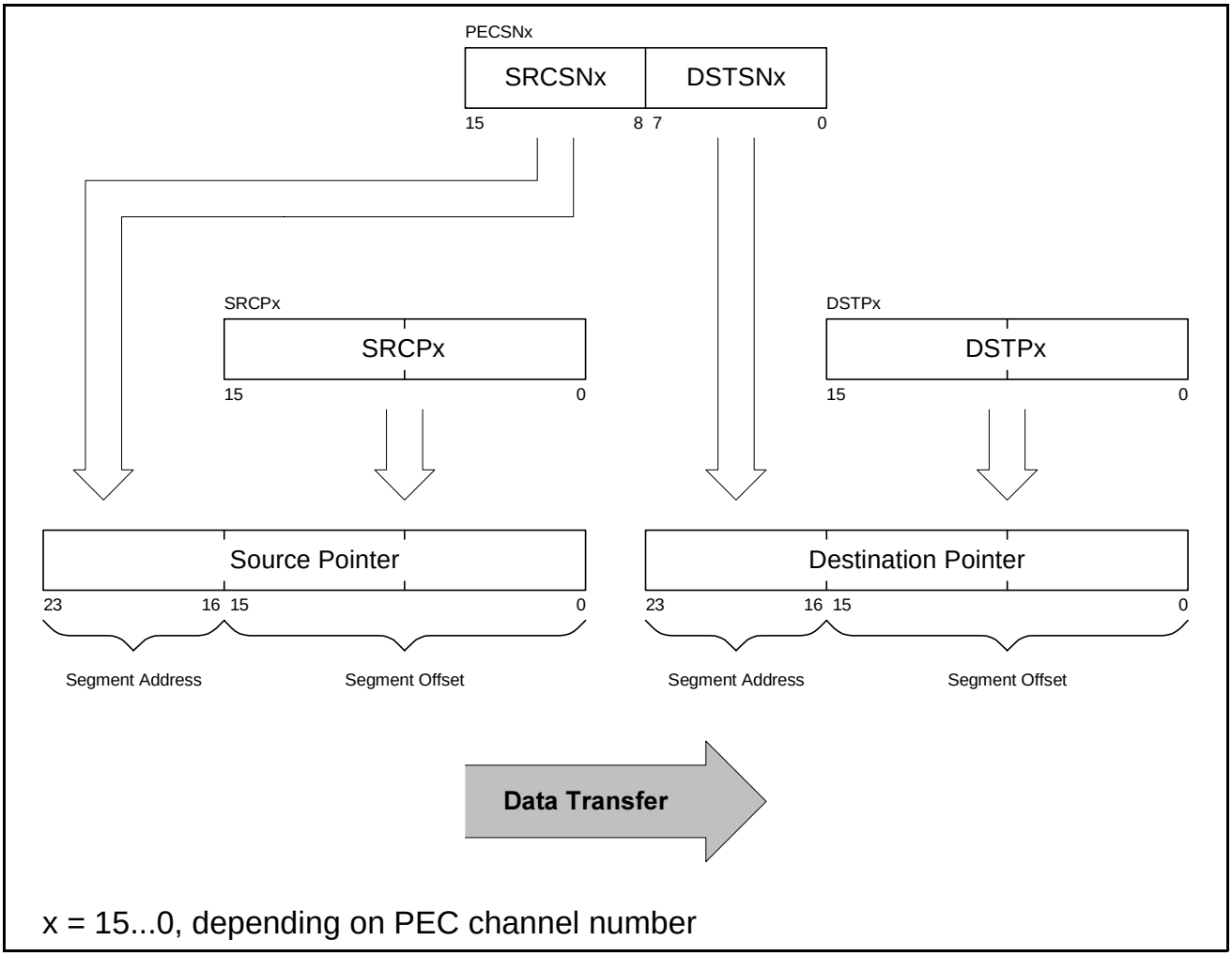
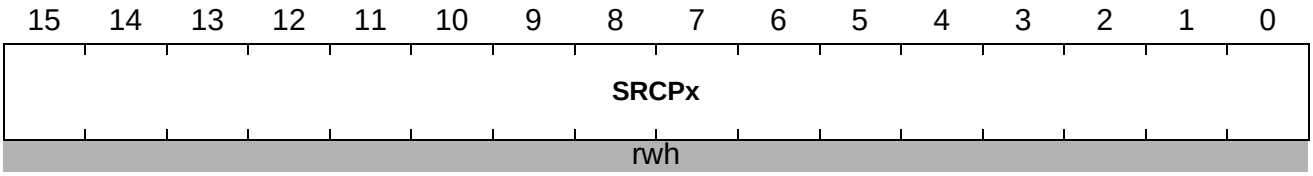


Figure 3-5 PEC Pointer Address Handling

The 24-bit destination address is stored in the DPRAM location DSTPx (lower 16 bits of address) and in the high byte of register PECSNx (highest 8 address bits). Only the lower 16 bits of the PEC address pointers (segment offset) can be modified (incremented) by the PEC transfer mechanism. The highest 8 bits, which represent the segment number, are not modified by hardware. Therefore, the PEC pointers may be incremented within the address space of one segment and may not cross the segment border. If the offset address pointer has a value of $FFFF_H$ in the case of byte transfers (BWT = 1) or $FFFE_H$ in the case of word transfers (BWT = 0), the next increment will lead to an overflow. No explicit error event is generated by the system in case of address pointer(s) overflow; therefore, the user must prevent this condition from occurring.

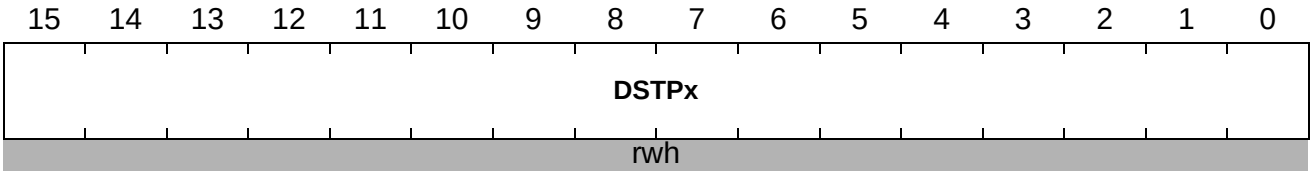
Note: If a word data transfer is selected for a specific PEC channel (i.e. BWT = 0), the respective source and destination pointers must both contain a valid word address that points to an even byte boundary. Otherwise, the Illegal Word Access trap will be invoked when this channel is used.

SRCPx
PEC Source Pointer **DPRAM($_{H,H}$)** **Reset value: 0000_H**



Field	Bits	Type	Description
SRCPx	[15:0]	rwh	Source Pointer Address of Channel x Source Address bits 15-0

DSTPx
PEC Destination Pointer **DPRAM($_{H,H}$)** **Reset value: 0000_H**



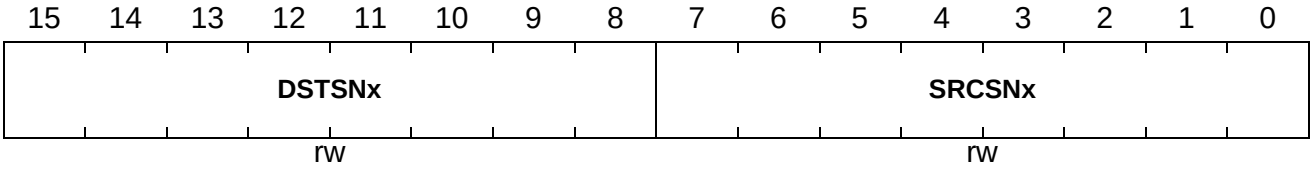
Field	Bits	Type	Description
DSTPx	[15:0]	rwh	Destination Pointer Address of Channel x Destination Address bits 15-0

Table 3-7 DPRAM Addresses of PEC Source and Destination Pointer

Pointer	Address	Pointer	Address	Pointer	Address	Pointer	Address
DSTP7	$00'FCFE_H$	SRCP7	$00'FCFC_H$	DSTP11	$00'FCDE_H$	SRCP11	$00'FCDC_H$
DSTP6	$00'FCFA_H$	SRCP6	$00'FCF8_H$	DSTP10	$00'FCDA_H$	SRCP10	$00'FCD8_H$
DSTP5	$00'FCF6_H$	SRCP5	$00'FCF4_H$	DSTP9	$00'FCD6_H$	SRCP9	$00'FCD4_H$
DSTP4	$00'FCF2_H$	SRCP4	$00'FCF0_H$	DSTP8	$00'FCD2_H$	SRCP8	$00'FCD0_H$
DSTP3	$00'FCEE_H$	SRCP3	$00'FCEC_H$	DSTP15	$00'FCCE_H$	SRCP15	$00'FCCC_H$
DSTP2	$00'FCEA_H$	SRCP2	$00'FCE8_H$	DSTP14	$00'FCCA_H$	SRCP14	$00'FCC8_H$
DSTP1	$00'FCE6_H$	SRCP1	$00'FCE4_H$	DSTP13	$00'FCC6_H$	SRCP13	$00'FCC4_H$
DSTP0	$00'FCE2_H$	SRCP0	$00'FCE0_H$	DSTP12	$00'FCC2_H$	SRCP12	$00'FCC0_H$

PECSNx

PEC Segment Pointer **SFR($_{H,H}$)** **Reset value: 0000$_H$**

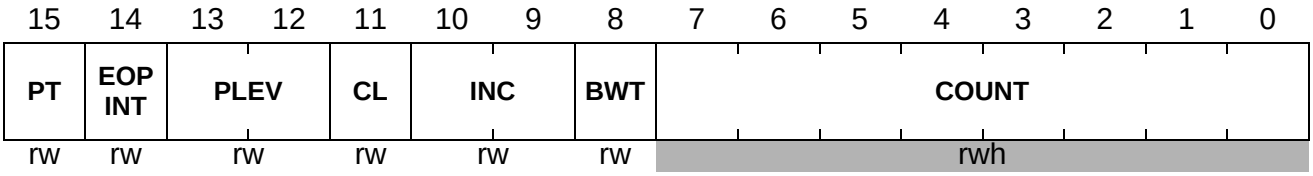


Field	Bits	Type	Description
DSTSNx	[15:8]	rw	Destination Pointer Segment Address of Channel x Destination Address bits 23-16
SRCSNx	[7:0]	rw	Source Pointer Segment Address of Channel x Source Address bits 23-16

3.4.6.2 PEC Control Registers

Each PEC channel is controlled by the respective PEC channel **C**ontrol register (PECCx) and a set of source and destination pointers (SRCPx, DSTPx and PECSNx), where "x" stands for the PEC channel number. The PECCx registers control the arbitration priority level assigned to the PEC channels and specifies the action to be performed.

PECCx
PEC Channel Control Register **SFR($_{H,H}$)** **Reset value: 0000$_H$**



Field	Bits	Type	Description
PT	[15]	rw	Transfer Mode[1)] 0 Short Transfer Mode 1 Long Transfer Mode
EOPINT	[14]	rw	End of PEC Interrupt Selection[2)] 0 EOP interrupt with the same level as the PEC transfer is triggered 1 EOP interrupt is serviced by a separate interrupt node with programmable interrupt level (EOPIC) and interrupt sharing control register (PECISNC)
PLEV	[13:12]	rw	PEC Level Selection[3)] This bitfield controls the PEC channel assignment to an arbitration priority level. (see section below)
CL	[11]	rw	Channel Link Control 0 PEC channels work independently 1 Pairs of channels are linked together
INC	[10:9]	rw	Increment Control (Modification of source and destination pointer after PEC transfer) 00 No modification 01 Increment of destination pointer DSTPx by 1 (BWT = 1) or by 2 (BWT = 0) 10 Increment of source pointer SRCPx by 1 (BWT = 1) or by 2 (BWT = 0) 11 Reserved

Field	Bits	Type	Description
BWT	[8]	rw	Byte / Word Transfer Selection 0 Transfer a word 1 Transfer a byte
COUNT	[7:0]	rwh	**PEC Transfer Count**[4] Counts PEC transfers and influences the channel´s action

1) The long transfer mode is an optional mode. If the product does not support the long transfer mode for this specific PEC channel, the PT-bit is hardwired to zero. See **Section 3.4.6.3** and **Section 3.4.6.4**

2) See **Section 3.4.6.7**

3) See **Section 3.4.6.6**

4) See **Section 3.4.6.3**

The **Byte/Word Transfer bit (BWT)** of the PECCx register selects whether a byte or a word is to be moved during a PEC service cycle, and defines an increment step size for the pointer(s) to be modified.

The **Increment Control Field (INC)** of the PECCx register defines when either one or both of the PEC pointers have to be incremented after the PEC transfer. If the pointers are not to be modified (INC=00_B), the respective channel will always move data from the same source to the same destination.

3.4.6.3 Short Transfer Mode

If the short transfer mode is enabled by the PT flag (PT = 0) in the PEC control register PECCx, the PEC Transfer Count Field (COUNT) of the PECCx controls directly the action of the respective PEC channel. The contents of the bitfield COUNT may specify a certain number of PEC transfers, unlimited transfers, or no PEC service at all.

a) If the PEC transfer counter (COUNT) value is set to 00_H, the normal interrupt requests are processed instead of PEC data transfers, and the corresponding PEC channel remains idle.

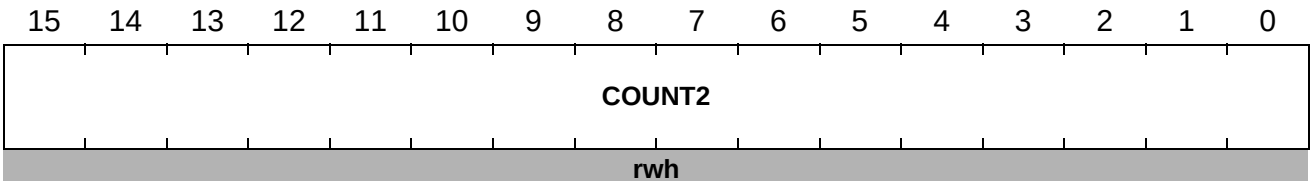
b) Continuous data transfers are selected by setting the bitfield COUNT to FF_H. In this case, COUNT is not decremented by the transfers, and the respective PEC channel can serve unlimited number of PEC requests until it is modified by the program.

c) If the bitfield COUNT is set to service a specified number of requests by the respective PEC channel, it is decremented with each PEC transfer, and the request flag is cleared to indicate that the request has been serviced. When COUNT reaches 00_H it activates the ISR that has the same priority level (EOPINT = 0), or triggers the EOP ISR with a different priority level (EOPINT = 1). When COUNT is decremented from 01_H to 00_H after a data transfer, the request flag will be cleared if EOPINT is set to 1. If EOPINT is 0, the request flag will not be cleared and another interrupt request will be generated on the same priority level. The respective PEC channel remains idle, and the associated ISR is activated instead of PEC transfer, because COUNT contains the 00_H value.

3.4.6.4 Long Transfer Mode[1)]

If the long transfer mode is enabled by the PT flag (PT = 1) in the PEC control register PECCx, the **PEC Transfer Count Field** (COUNT2) of the PECXCx register directly controls the action of the respective PEC channel.

PEC Extended Count Register
PECXC0/2/4/6 **SFR ($_{H,H}$)** **(Reset value: 0000_H)**



Field	Bits	Type	Description
COUNT2	[15:0]	rwh	PEC Extended (Long) Transfer Count PEC transfer count extension (see table below)

The long transfer mode is only available for PEC channels with an even PEC number.

Note: The channel link mode is independent of the long transfer mode. Both modes can be combined.

Note: The PEC Transfer Count Field (COUNT) of the PECCx register must be set to zero.

Note: Crossing of segment boundaries is not checked during data transfers with long transfer count, and is not supported. A wrap around occurs when reaching the segment boundary .

The contents of the bitfield COUNT2 may specify a certain number of PEC transfers or no PEC service at all. The 16-bit transfer counter permits servicing up to 65536 byte transfers or up to 32768 word transfers.

a) If the PEC transfer counter COUNT2 value is set to 0000_H , the normal interrupt requests are processed instead of PEC data transfers, and the corresponding PEC channel remains idle.

b) If the bitfield COUNT2 is set to service a specified number of requests by the respective PEC channel, it is decremented with each PEC transfer and the request flag is cleared to indicate that the request has been serviced. When COUNT2 reaches 0000_H, it activates the ISR that has the same priority level (EOPINT = 0), or triggers the EOP ISR with a different priority level (EOPINT = 1). When COUNT2 is decremented from 0001_H to 0000_H after a data transfer, the request flag will be

1) The long transfer mode is an optional mode for PEC channels with an even number. Please refer to the product User Manual.

cleared if EOPINT is set to 1. If EOPINT is 0, the request flag will not be cleared and another interrupt request will be generated on the same priority level. The respective PEC channel remains idle and the associated interrupt service routine is activated instead of PEC transfer, because COUNT2 contains the 0000_H value.

3.4.6.5 Channel Link Mode for Data Chaining

Channel linking, if enabled, links two channels together to serve the data transfer requests of one peripheral. The whole data transfer (for example a message) is divided into separately-controlled block transfers. The two PEC channels that are linked together handle chained block transfers alternately with one another. At the end of a data block transfer controlled by one PEC channel, the other (linked) PEC channel is started automatically to continue the transfer with the next data block. Channel linking and data (block) chaining are supported within pairs of PEC channels (channels 0&1, 2&3, 4&5 etc.). Each data block is controlled by one PEC channel of the channel pair.

Channel linking is enabled if the Channel Link (CL) control bits of **both** PEC channels are set to 1 in their PECCx registers . The data transfer of linked channels must always be started always with the **even** numbered channel of the channel pair.

If in channel link mode the channel's data block is completely transferred, the PEC service request processing is automatically switched to the other PEC channel of the pair. CL of the previously active PEC channel is then reset.

Every channel toggle is indicated to CPU by means of an EOP interrupt. This makes it possible to set up multiple buffers for PEC transfers by changing pointer and count values of one channel while the other channel is active. Inside the EOP interrupt, the Channel Link Control bit CL must be set again before the channel is reactivated or the channel link mode is finished.This EOP interrupt is requested, indicated, and enabled in the respective PEC Interrupt Subnode Control Register (PECISNC or PECXISNC).

Thus, all EOP interrupts are controlled with the one EOP interrupt control register EOPIC and therefore with the same interrupt priority level. This service request node requests the CPU in case of one or more pending EOP interrupt requests if the respective enable control bit(s) are set in the according subnode control register and in the interrupt control register EOPIC.

If CL of the previous PEC channel is set to zero and the count field (COUNT=0 or COUNT2=0, dependent on the mode) of the active channel is zero as well, the whole data transfer is finished and the channel link interrupt represents a termination interrupt, the End of PEC interrupt.

The channel link feature is supported for all PEC channels, including the new PEC channels 8-15. The following table shows the channels that can be linked together and the channel numbers required to start transfers via linked channels.

Table 3-8 PEC Channels That Can Be Linked Together

Linked PEC Channels		Linked PEC Start Channel	Linked PEC Channels		Linked PEC Start Channel
PEC Channel A	PEC Channel B		PEC Channel A	PEC Channel B	
channel 0	channel 1	channel 0	channel 8	channel 9	channel 8
channel 2	channel 3	channel 2	channel 10	channel 11	channel 10
channel 4	channel 5	channel 4	channel 12	channel 13	channel 12
channel 6	channel 7	channel 6	channel 14	channel 15	channel 14

The two PEC control registers of a pair are linked to one interrupt control register, whereby in this IC register only the even-numbered PEC channel is indicated with the priority/group bits.

3.4.6.6 PEC Channels Assignment and Arbitration

The PEC channels can be assigned to arbitration priority levels. All requests with interrupt priority levels 8 to 15 can be associated with the PEC functionality (up to a total of sixteen PEC channels). The following formula shows how to program the bitfield PECCx.PLEV to set up a link to a certain interrupt priority level and a group priority level.

PEC channel: x = (x.3,x.2,x.1,x.0)
linked to
Interrupt priority level: (1, ~PLEV.1, ~PLEV.0, x.2)
Group priority level: (x.3, x.1, x.0) [3-1]

The following table lists all possible combinations:

Table 3-9 PEC interrupt level control with PLEV bits in PECCx registers

Priority Level		PEC Channel Selection (x)			
Interrupt Level ILVL3-0	Group Level xxGP, GLVL1,0	PLEV[1,0]= 00	PLEV[1,0] = 01	PLEV[1,0] = 10	PLEV[1,0] = 11
15	7-4	15-12	-	-	-
15	3-0	7-4	-	-	-
14	7-4	11-8	-	-	-
14	3-0	3-0	-	-	-
13	7-4	-	15-12	-	-
13	3-0	-	7-4	-	-
12	7-4	-	11-8	-	-
12	3-0	-	3-0	-	-
11	7-4	-	-	15-12	-
11	3-0	-	-	7-4	-
10	7-4	-	-	11-8	-
10	3-0	-	-	3-0	-
9	7-4	-	-	-	15-12
9	3-0	-	-	-	7-4
8	7-4	-	-	-	11-8
8	3-0	-	-	-	3-0

All interrupt requests that are not assigned to a PEC channel go directly to the interrupt handler.

3.4.6.7 Programmable End of PEC Interrupt Level

The programmable EOP interrupt supports PEC transfers, which need a high priority level for the transfer request, and which do not need the same priority level for the termination interrupt. One dedicated service request node with a programmable interrupt level is shared among all PEC channels. This service request node is controlled by the EOPIC interrupt control register.

EOPIC
Interrupt Control Register **bSFR($xxxx_H$,xx_H)[1)]** **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	xxGP	EOP IR	EOP IE	ILVL				GLVL	
r	r	r	r	r	r	r	rw	rwh	rw	rw				rw	

1) The EOPIC register is assigned to one of the 64 interrupt control registers. The assignment is product specific.

Field	Bits	Type	Description
xxGP	[8]	rw	Group Priority Extension Defines the value of high order group level bit
EOPIR[1)]	[7]	rwh	Interrupt Request Flag 0 No request pending 1 his source has raised an interrupt request
EOPIE	[6]	rw	**Interrupt Enable Control Bit** 0 Interrupt request is disabled 1 Interrupt request is enabled
ILVL	[5:2]	rw	Interrupt Priority Level F_H Highest priority level ... ... 0_H Lowest priority level
GLVL	[1:0]	rw	Group Priority Level 3_H Highest priority level ... ... 0_H Lowest priority leve

1) Bit xxIR supports bit-protection

The Register PECISNC and PECXISN contain flags of the EOP interrupt node. This node is used when the enhanced End of PEC interrupt feature is invoked and control bit EOPINT is set to 1 in the corresponding PECCx**.**

PECISNC

PEC Interrupt Sub Node Control **SFR($_{H,H}$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C7IR	C7IE	C6IR	C6IE	C5IR	C5IE	C4IR	C4IE	C3IR	C3IE	C2IR	C2IE	C1IR	C1IE	C0IR	C0IE
rwh	rw	rwh	rw	rwh	rw	rwh	rw	rwh	rw	rwh	rw	rwh	rw	rwh	rw

PECXISNC

PEC Interrupt Sub Node Control **SFR($_{H,H}$)** **Reset value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C15 IR	C15 IE	C14 IR	C14 IE	C13 IR	C13 IE	C12 IR	C12 IE	C11 IR	C11 IE	C10 IR	C10 IE	C9 IR	C9 IE	C8 IR	C8 IE
rwh	rw	rwh	rw	rwh	rw	rwh	rw	rwh	rw	rwh	rw	rwh	rw	rwh	rw

Field	Bits	Type	Description
CxIR	15, 13, 11, 9, 7, 5, 3, 1	rwh	Interrupt Sub Node Request Flag of PEC Channel x [1) 2)] 0 No special EOP interrupt request is pending for PEC channel x 1 PEC channel x has raised an EOP interrupt request
CxIE	14, 12, 10, 8, 6, 4, 2, 0	rw	**Interrupt Sub Node Enable Control Bit of PEC Channel x** [1) 3)] (individually enables/disables a specific source) 0 EOP interrupt request of PEC channel x is disabled 1 EOP interrupt request of PEC channel x is enabled

1) x = 15...0

2) NOTE:
The EOP sub-node interrupt request flags are not cleared by hardware when entering the ISR (interrupt has been accepted by the CPU), unlike the interrupt request flags of the interrupt nodes (request flags xxIC.xxIR). The ISR has to check the request flags and to clear them before executing the RETI instruction.

3) It is recommended that you clear an interrupt request flag (CxIR) before setting the respective enable flag (CxIE). Otherwise, pending former requests will immediately trigger an interrupt request after setting the enable bit.

3.5 Using General-Purpose Registers

The C166S uses several banks of 16 dedicated General Purpose Registers (GPRs) **R0**, **R1**, **R2**... **R15** that can be accessed in one CPU cycle. The GPRs are the working registers of the Arithmetic and Logic Units (ALU) and may also serve as address pointers in indirect addressing modes.

Several banks of GPRs are memory-mapped. The banks of these GPRs are located in the DPRAM. One bank uses a block of 16 consecutive words. A Context Pointer (CP) register determines the base address of the currently selected bank.

The C166S can switch the complete GPR bank with a single instruction for time-critical tasks. After switching, the new task is executed within its own separate context.

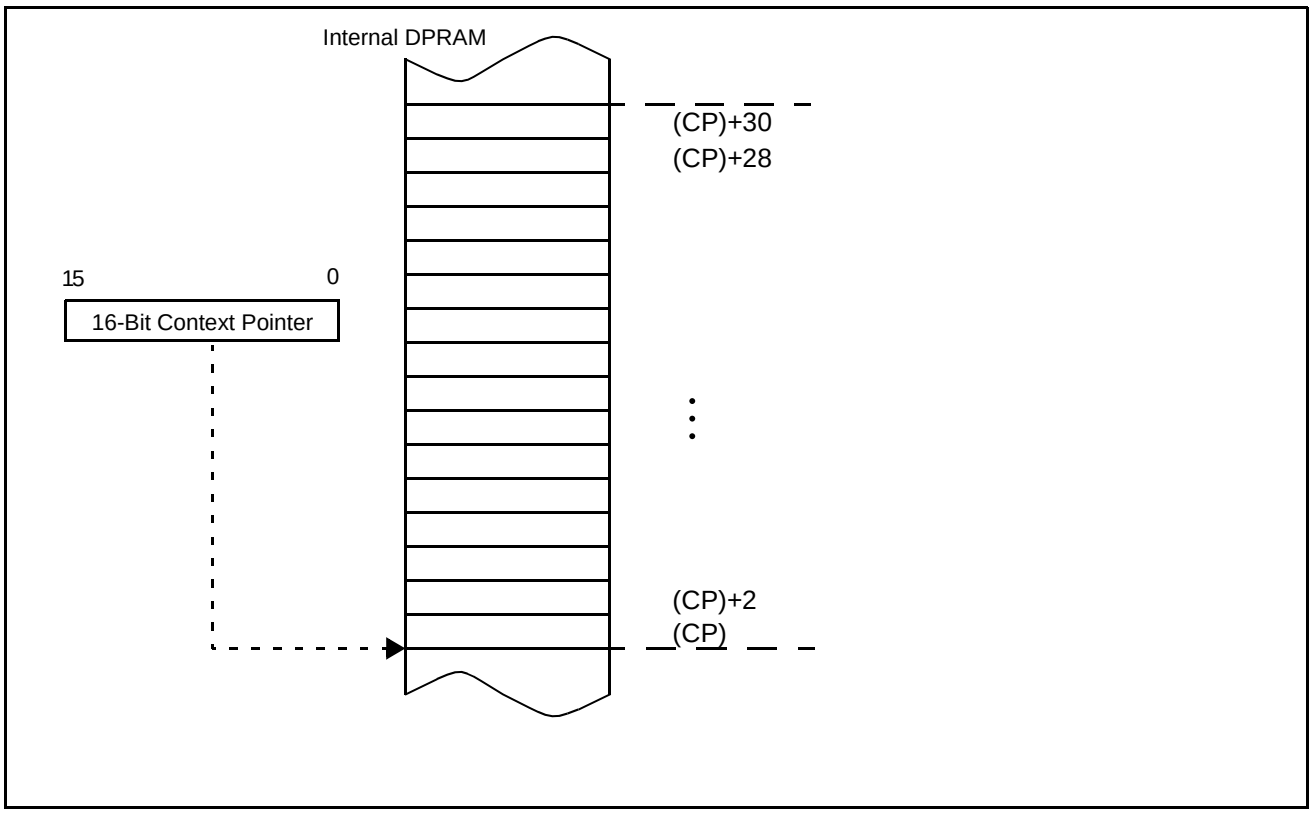


Figure 3-6 Register Bank Selection via Register CP

There are 3 different ways to access the GPRs:

Short 4-bit GPR addresses (mnemonic: Rw or Rb) specify an address relative to the memory location pointed to by the contents of the CP register, i.e., the base of contents of the current register bank. Both byte-wise and word-wise GPR accesses are possible. The short 4-bit GPR address is logically added to the contents of register CP if a byte (Rb) GPR address is specified, or multiplied by two and then added to CP if a word (Rw) GPR address is specified (see **Figure 3-7**).

Note: If GPRs are used as indirect address pointers, they are always accessed word-wise.

For some instructions, only the first 4 GPRs (R0, R1, R2 and R3) can be used as indirect address pointers. These GPRs are specified via short 2-bit GPR addresses. The physical address calculation is identical to the one 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 a short 4-bit GPR address, while the four most significant bits are ignored. The physical GPR address is calculated in a similar fashion as the short 4-bit GPR addresses. For single-bit GPR accesses, the GPR's word address is calculated in the same way. The accessed bit position within the word is specified by a separate additional 4-bit value.

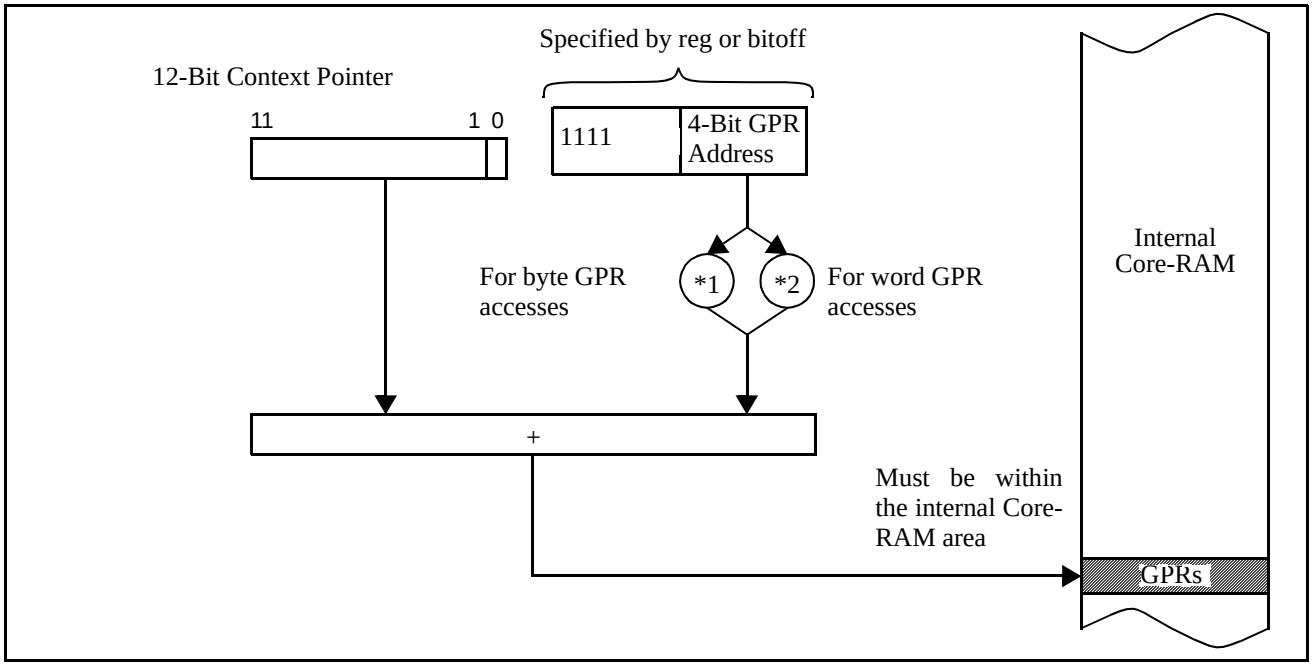


Figure 3-7 Implicit CP Use by logical Short GPR Addressing Modes

24-Bit memory addresses within a range from (CP)+0 to (CP)+30 can be used to access GPRs directly. Both byte and word GPR accesses are possible. The 24-bit memory address is generated according to the rules for long- and indirect-addressing modes (**Section 3.6.2**).

Table 3-10 Addressing modes to Access Word-GPRs

Name	Physical Address	8-Bit Address	4-Bit Address	Description	Reset Value
R0	(CP)+0	$F0_H$	0_H	General-Purpose word Register R0	$UUUU_H$
R1	(CP)+2	$F1_H$	1_H	General-Purpose word Register R1	$UUUU_H$
R2	(CP)+4	$F2_H$	2_H	General-Purpose word Register R2	$UUUU_H$
R3	(CP)+6	$F3_H$	3_H	General-Purpose word Register R3	$UUUU_H$
R4	(CP)+8	$F4_H$	4_H	General-Purpose word Register R4	$UUUU_H$
R5	(CP)+10	$F5_H$	5_H	General-Purpose word Register R5	$UUUU_H$
R6	(CP)+12	$F6_H$	6_H	General-Purpose word Register R6	$UUUU_H$
R7	(CP)+14	$F7_H$	7_H	General-Purpose word Register R7	$UUUU_H$
R8	(CP)+16	$F8_H$	8_H	General-Purpose word Register R8	$UUUU_H$
R9	(CP)+18	$F9_H$	9_H	General-Purpose word Register R9	$UUUU_H$
R10	(CP)+20	FA_H	A_H	General-Purpose word Register R10	$UUUU_H$
R11	(CP)+22	FB_H	B_H	General-Purpose word Register R11	$UUUU_H$
R12	(CP)+24	FC_H	C_H	General-Purpose word Register R12	$UUUU_H$
R13	(CP)+26	FD_H	D_H	General-Purpose word Register R13	$UUUU_H$
R14	(CP)+28	FE_H	E_H	General-Purpose word Register R14	$UUUU_H$
R15	(CP)+30	FF_H	F_H	General-Purpose word Register R15	$UUUU_H$

Note: The first 8 GPRs (R7...R0) may also be accessed byte-wise.

Note: Writing to a GPR byte does not affect the other byte of the same GPR.

Each half of the byte-wise accessible registers has a special name (see table below).

Table 3-11 Addressing modes to access Byte-GPRs

Name	Physical Address	8-Bit Address	4-Bit Address	Description	Reset Value
RL0	(CP)+0	$F0_H$	0_H	General-Purpose byte Register RL0	UU_H
RH0	(CP)+1	$F1_H$	1_H	General-Purpose byte Register RL1	UU_H
RL1	(CP)+2	$F2_H$	2_H	General-Purpose byte Register RL2	UU_H
RH1	(CP)+3	$F3_H$	3_H	General-Purpose byte Register RL3	UU_H
RL2	(CP)+4	$F4_H$	4_H	General-Purpose byte Register RL4	UU_H
RH2	(CP)+5	$F5_H$	5_H	General-Purpose byte Register RL5	UU_H
RL3	(CP)+6	$F6_H$	6_H	General-Purpose byte Register RL6	UU_H
RH3	(CP)+7	$F7_H$	7_H	General-Purpose byte Register RL7	UU_H
RL4	(CP)+8	$F8_H$	8_H	General-Purpose byte Register RL8	UU_H
RH4	(CP)+9	$F9_H$	9_H	General-Purpose byte Register RL9	UU_H
RL5	(CP)+10	FA_H	A_H	General-Purpose byte Register RL10	UU_H
RH5	(CP)+11	FB_H	B_H	General-Purpose byte Register RL11	UU_H
RL6	(CP)+12	FC_H	C_H	General-Purpose byte Register RL12	UU_H
RH6	(CP)+13	FD_H	D_H	General-Purpose byte Register RL13	UU_H
RL7	(CP)+14	FE_H	E_H	General-Purpose byte Register RL14	UU_H
RH7	(CP)+15	FF_H	F_H	General-Purpose byte Register RL15	UU_H

3.5.1 Context Switch

An Interrupt Service Routine (ISR) or a task scheduler of an operating system usually saves the contents of all used registers into the stack, and restores them before returning. The more registers a routine uses, the more time is wasted by saving and restoring.

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

The Context Pointer

The CP register is not bit-addressable. It can be updated via any instruction capable of modifying SFRs.

CP
Context Pointer **SFR($FE10_H$,08_H)** **Reset value: $FC00_H$**

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

Field	Bits	Type	Description
1	[15:12]	r	CP always points in the DPRAM
CP	[11:1]	rw	**Modifiable portion of register CP** Specifies the (word) base address of the current memory-mapped register bank. *Note: When writing a value to register CP with bits CP[11:9] = 000, bits CP[11:10] are set to 11 by hardware.*
0	[0]	r	CP is always word-aligned

Note: It is the user's responsibility to ensure that the physical GPR address specified via CP register plus short GPR address must always be an RAM location. If this condition is not met, unexpected results may occur. Do not set CP below the DPRAM start address.

Note: Due to the internal instruction pipeline, a new CP value cannot be used for GPR address calculations for the instruction immediately following the instruction updating the CP register.

The C166S switches the complete memory-mapped GPR bank with a single instruction. After switching, the service routine executes within its own separate context.

The instruction SCXT CP, #New_Bank pushes the value of the current context pointer (CP) into the system stack and loads CP with the immediate value New_Bank, which selects a new register bank. The service routine may now use its own registers. This memory register bank is preserved when the service routine terminates, i.e. its contents are available on the next call. Before returning from the service routine (RETI), the previous CP is simply popped from the system stack, which returns the registers to the original bank.

3.6 Data Addressing

The C166S provides a lot of powerful addressing modes for word-wise, byte-wise and bitwise data accesses (short, long, indirect). The different addressing modes use different formats and have different scopes.

The following major tasks are performed:

- Address generation using short-, long- and indirect-addressing modes
- Data paging or overwriting mechanism
- System stack handling

3.6.1 Short Addressing Modes

All of these addressing modes use an implicit base offset address to specify a 24-bit physical address. Short addressing modes allow access to the GPRs, SFRs, or bit-addressable memory space:

Physical Address = Base Address + Δ * Short Address

Note: Δ is 1 for byte-wise GPRs, Δ is 2 for word-wise GPRs.

Table 3-12 Short addressing modes

Mnemonic	Physical Address	Short Address Range	Scope of Access
Rw	(CP) + 2*Rw or local	Rw = 0...15	GPRs(Word)
Rb	(CP) + 1*Rb or local	Rb = 0...15	GPRs(Byte)
reg	$00'FE00_H$+ 2*reg $00'F000_H$+ 2*reg (CP)+ 2*(reg$\wedge 0F_H$) (CP)+ 1*(reg$\wedge 0F_H$)	reg = 00_H...EF_H reg = 00_H...EF_H reg = $F0_H$...FF_H reg = $F0_H$...FF_H	SFRs (Word, Low byte) ESFRs(Word, Low byte) GPRs(Word) GPRs(Bytes)
bitoff	$00'FD00_H$+ 2*bitoff $00'FF00_H$+ 2*(bitoff$\wedge 7F_H$) $00'F100_H$+ 2*(bitoff$\wedge 7F_H$) (CP) + 2*(bitoff$\wedge 0F_H$)	bitoff = 00_H...$7F_H$ bitoff = 80_H...EF_H bitoff = 80_H...EF_H bitoff = $F0_H$...FF_H	DPRAMBit word offset SFR Bit word offset ESFRBit word offset GPR Bit word offset
bitaddr	Word offset as with bitoff. immediate bit position.	bitoff = 00_H...FF_H bitpos= 0...15	Any single bit

Rw, Rb: Specifies direct access to any GPR in the currently active context. Both Rw and Rb require 4 bits in the instruction format. The base address of the global register bank is determined by the contents of register CP. Rw specifies a 4-bit word GPR address relative to the base address (CP), while Rb specifies a 4-bit byte GPR address relative to the base address (CP).

reg: Specifies direct access to any (E)SFR or GPR in the currently active context. The reg value requires 8 bits in the instruction format. Short reg addresses in the range from 00_H to EF_H always specify (E)SFRs. In that case, the factor Δ equals 2, and the base address is $00'FE00_H$ for the standard SFR area or $00'F000_H$ for the extended ESFR area. The reg accesses to the ESFR area require a preceding EXT*R instruction to switch the base address. Depending on the opcode, either the total word (for word operations) or the low byte (for byte operations) of an SFR can be addressed via reg. Note that the high byte of an SFR cannot be accessed via the reg addressing mode. Short reg addresses in the range from $F0_H$ to FF_H always specify GPRs. In that case, only the lower 4bits of reg are significant for physical address generation and, therefore, the address calculation is identical to the address generation process described for the Rb and Rw addressing modes.

bitoff: Specifies direct access to any word in the bit-addressable memory space. The bitoff value requires 8 bits in the instruction format. Depending on the specified bitoff range, different base addresses are used to generate physical addresses: Short bitoff addresses in the range from 00_H to $7F_H$ use $00'FD00_H$ as a base address to specify the 128 highest DPRAM word locations in the range from $00'FD00_H$h to $00'FDFE_H$. Short bitoff addresses in the range from 80_H to EF_H use base address $00'FF00_H$ to specify the internal SFR word locations in the range from $00'FF00_H$ to $00'FFDE_H$ or base address $00'F100_H$ to specify the internal ESFR word locations in the range from $00'F100_H$ to $00'F1DE_H$. The bitoff accesses to the ESFR area require a preceding EXT*R instruction to switch the base address. For short bitoff addresses from $F0_H$ to FF_H, only the lowest four bits are used to generate the address of the selected word GPR.

bitaddr: Any bit address is specified by a word address within the bit-addressable memory space (see bitoff), and by a bit position (bitpos) within that word. Therefore, bitaddr requires 12 bits in the instruction format.

3.6.2 Long and Indirect Addressing Modes

These addressing modes use one of the 4 DPP registers to specify a 24-bit address. Any word or byte data within the entire address space can be accessed with these modes. Any long or indirect 16-bit address contains two parts that have different meanings. Bits 13-0 specify a 14-bit data page offset, while bits 15-14 specify the **D**ata **P**age **P**ointer (DPP) (1 of 4) register, which is used to generate the full 24-bit address (see figure below).

The C166S also supports an override mechanism for the DPP addressing scheme [EXTP(R) and EXTS(R) instructions].

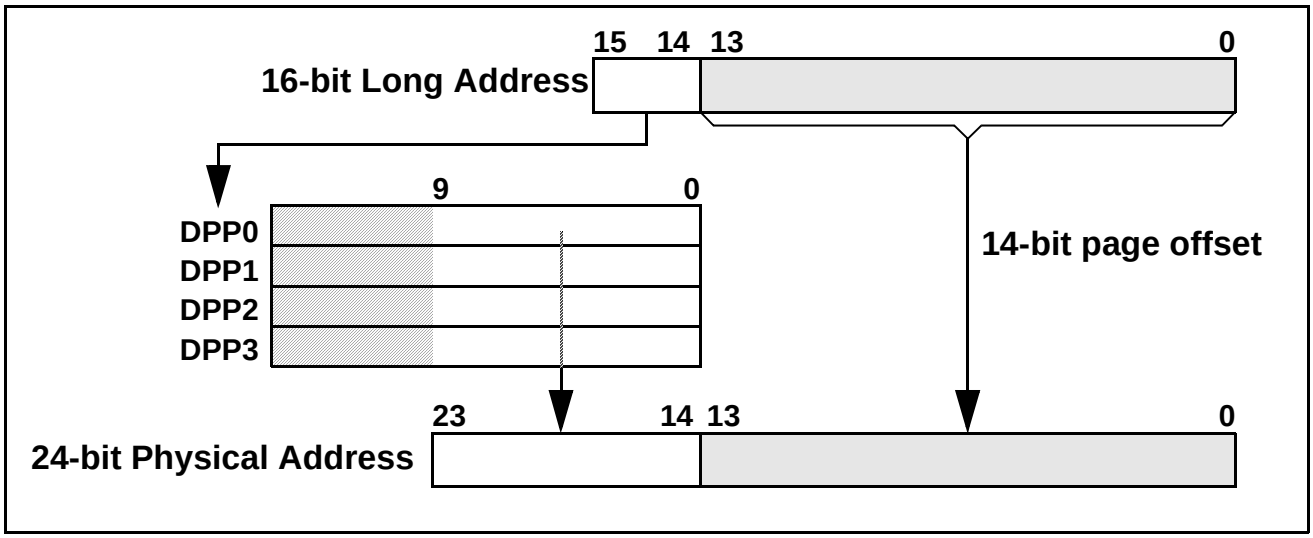


Figure 3-8 Interpretation of a 16-bit Long Address

Note: Word accesses on odd byte addresses are not executed. A hardware trap will be triggered.

3.6.2.1 Addressing via Data Page Pointer

The 4 non-bit-addressable DPP registers select up to 4 different data pages. The lower 10 bits of each DPP register select one of the 1024 possible 16-KByte data pages, while the upper 6 bits are reserved for the future use. The DPP registers provide access to the entire memory space in 16-KByte pages.

The DPP registers are used implicitly 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).

Data paging is performed by concatenating the lower 14 bits of an indirect or direct long 16-bit address with the contents of the DDP register selected by the upper 2 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.

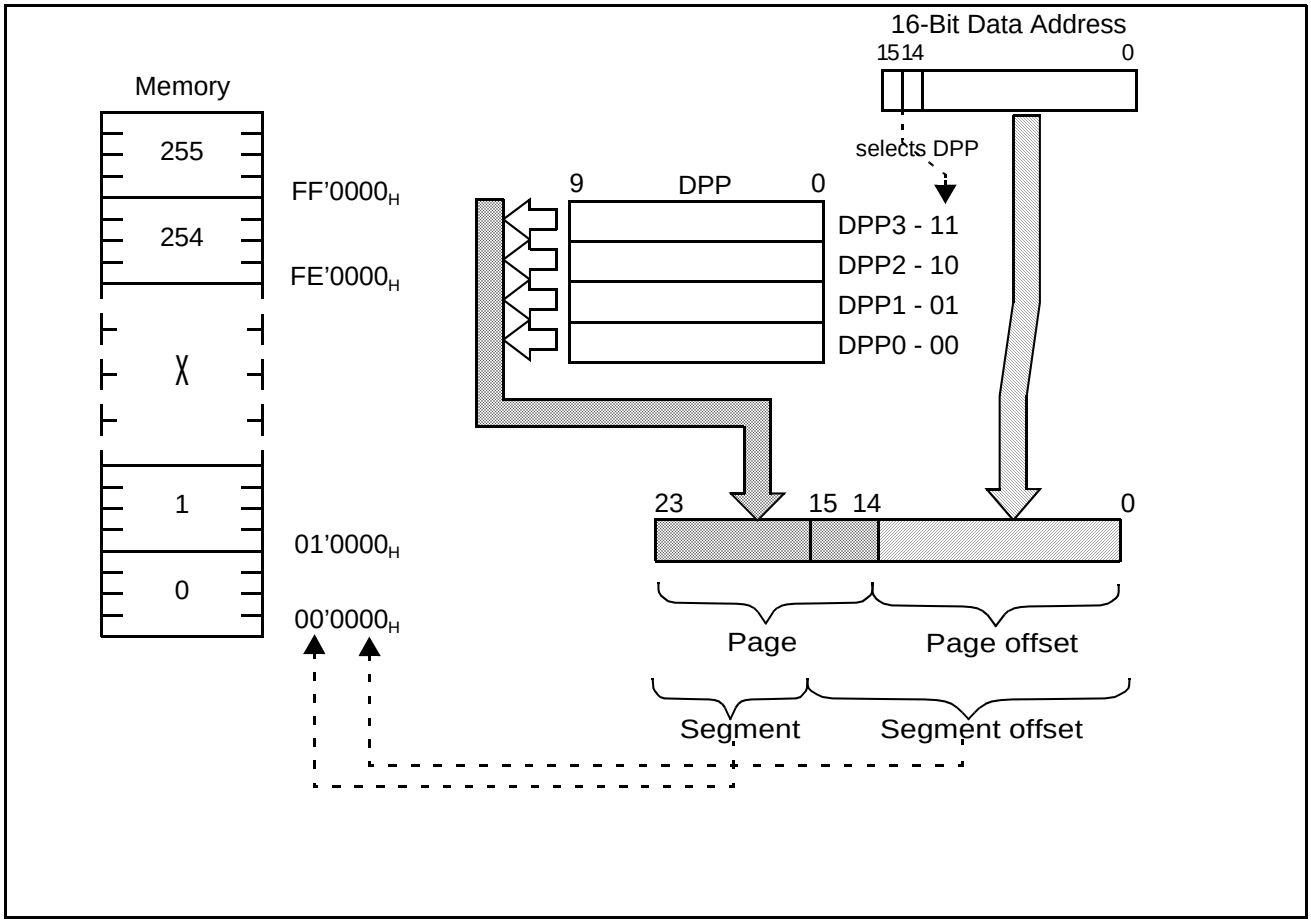


Figure 3-9 Addressing via the Data Page Pointer

After reset, the DPP registers select data pages 3-0 within segment 0. If the user does not want to use any data paging, no further action is required.

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
0	0	0	0	0	0	DPP0PN									
r	r	r	r	r	r	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
0	0	0	0	0	0	DPP1PN									
r	r	r	r	r	r	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
0	0	0	0	0	0	DPP2PN									
r	r	r	r	r	r	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
0	0	0	0	0	0	DPP3PN									
r	r	r	r	r	r	rw									

Field	Bits	Type	Description
DPPxPN	[9:0]	rw	Data Page Number of DPP Specifies the data page selected via DPP.

Note: In a non-segmented memory mode, the whole DPP register is still used for the calculation of the physical 24-bit address.

A DPP register can be updated via any instruction that is capable of modifying an SFR.

Note: Due to the internal instruction pipeline, a new DPP value is not usable for the operand address calculation of the instruction immediately following the instruction updating the DPPx register.

3.6.2.2 DPP Override Mechanism in the C166S

The C166S provides an override mechanism to temporarily bypass the DPP addressing scheme.

The EXTP(R) and EXTS(R) instructions override this addressing mechanism. Instruction EXTP(R) replaces the contents of the DPP register, while instruction EXTS(R) concatenates the complete 16-bit long address with the specified segment base address. The overriding page or segment may be specified directly as a constant (#pag, #seg) or via a word GPR (Rw).

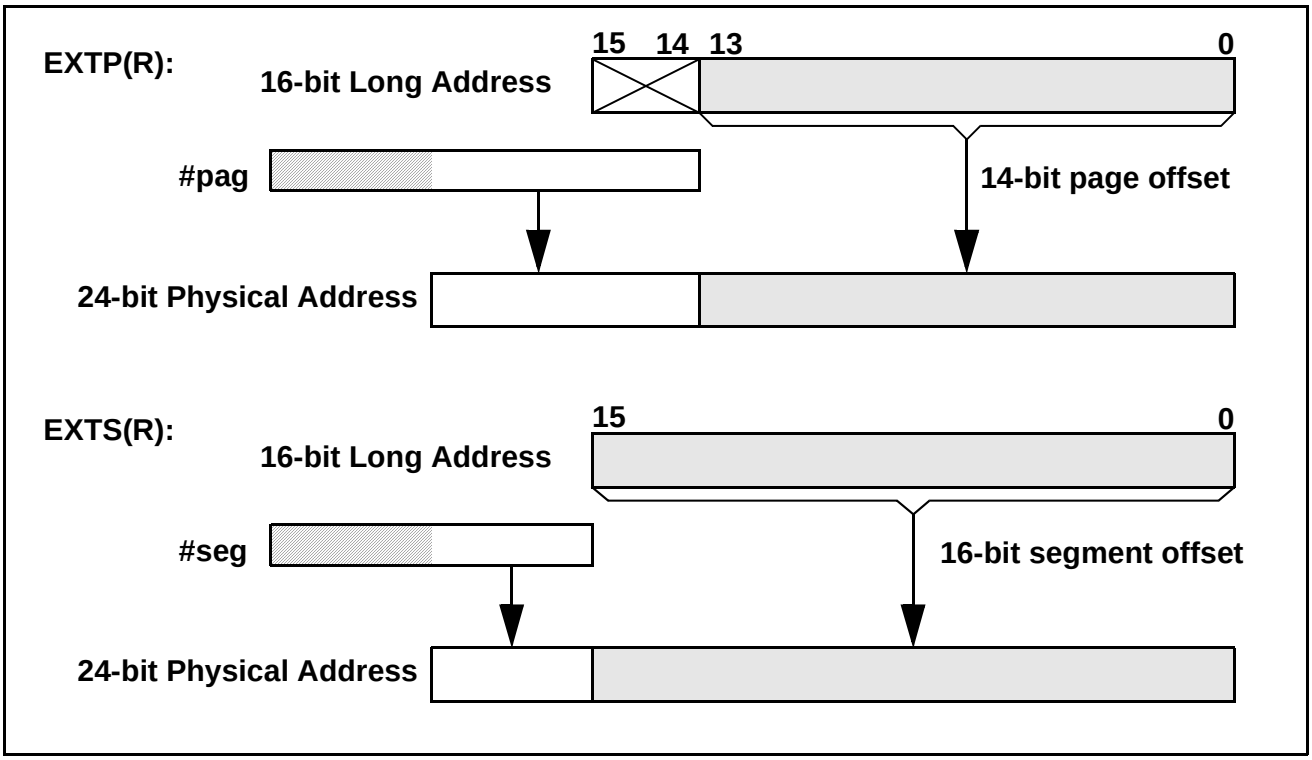


Figure 3-10 Overriding the DPP Mechanism

3.6.2.3 Long Addressing Mode

The long addressing mode uses a 16-bit constant value encoded in the instruction format which specifies the data page offset and the DPP.

The long addressing mode is referred to by the mnemonic mem.

Table 3-13 Long addressing mode

Mnemonic	Physical Address	Scope of Access
mem	(DPP0)|| mem∧$3FFF_H$ (DPP1)|| mem∧$3FFF_H$ (DPP2)|| mem∧$3FFF_H$ (DPP3)|| mem∧$3FFF_H$	any word or byte
mem	pag || mem∧$3FFF_H$	any word or byte
mem	seg || mem	any word or byte

Note: The long addressing mode may be used with the DPP overriding mechanism (EXTP(R) and EXTS(R)).

3.6.2.4 Indirect Addressing Modes

These addressing modes can be considered as a combination of short and long addressing modes. This means that a long 16-bit address is provided indirectly by the contents of a word GPR that is specified directly by a short 4-bit address (Rw = 0 to 15). There are indirect addressing modes which add a constant value to the GPR contents before the long 16-bit address is calculated. Other indirect addressing modes can decrement or increment the indirect address pointers (GPR contents) by 2 or 1 (referring to words or bytes).

In each case, one of the four DPP registers is used to specify physical 24-bit addresses. Any word or byte data within the entire memory space can be addressed indirectly.

Note: Indirect addressing may be used with the DPP overriding mechanism (EXTP(R) and EXTS(R)).

Some instructions use only the lowest 4-word GPRs (R3-R0) as indirect address pointers, which are then specified via short 2-bit addresses.

Physical addresses are generated from indirect address pointers using the following algorithm:

1) Calculate the physical address of the word GPR, which is used as indirect address pointer, using the specified short address (Rw) and

GPR Address = (CP) + 2 * Short Address

2) If required, pre-decrement indirect address pointer (-Rw) by the data-type-dependent value (Δ=1 for byte operations, Δ=2 for word operations) before the long 16-bit address is generated:

(GPR Address) = (GPR Address) - Δ ; [optional step!]

3) Calculate the long 16-bit address by adding a constant value (Rw+const16 if selected) to the contents of the indirect address pointer:

Long Address = (GPR Pointer) + Constant ; [+Constant is optional]

4) Calculate the physical 24-bit address using the resulting long address and the corresponding DPP register contents (see long mem addressing modes).

Physical Address = (DPPi) + Page offset

5) If required, post-in/decrement indirect address pointers ('Rw**±**') by the data-type-dependent value (Δ=1 for byte operations, Δ=2 for word operations).

(GPR Pointer) = (GPR Pointer) ± Δ; [optional step!]

The following indirect addressing modes are provided:

Table 3-14 Indirect addressing modes

Mnemonic	Particularities
[Rw]	Most instructions accept any GPR (R15...R0) as indirect address pointer. Some instructions accept only the lower four GPRs (R3...R0).
[Rw+]	The specified indirect address pointer is automatically post-incremented by 2 or 1 (for word or byte data operations) after the access.
[-Rw]	The specified indirect address pointer is automatically pre-decremented by 2 or 1 (for word or byte data operations) before the access.
[Rw+#data16]	The specified 16-bit constant is added to the indirect address pointer before the long address is calculated.

3.6.3 The System Stack

A system stack is provided to store return vectors, segment pointers, and processor status for procedures and interrupt routines.

The internal system stack can also be used to store data temporarily, 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 "unwanted" other byte must be disregarded.
Register SP can be loaded only with even byte addresses (The LSB of SP is always 0).

The **S**tack **P**ointer (SP) addresses the stack within the DPRAM area.

The Stack Pointer Register

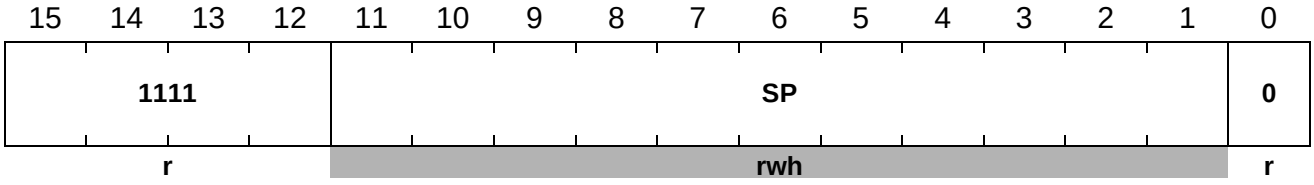
The non-bit-addressable Stack Pointer (SP) register is used to point to the Top Of the System (TOS) stack. The SP register is pre-decremented whenever data is to be pushed onto the stack, and it is post-incremented whenever data is to be popped from the stack. Therefore, the system stack grows from higher toward lower memory locations.

Since the Least Significant Bit (LSB) of register SP is tied to 0, and bits 15-12 are tied to 1 by hardware, the SP register can contain values only from $F000_H$ to $FFFE_H$. This allows access to a physical stack within the DPRAM of the C166S. A virtual stack (usually bigger) can be implemented via software. This mechanism is supported by registers STKOV and STKUN (see descriptions below (**Section 3.6.3.1**).

The SP register can be updated via any instruction that is capable of modifying a 16-bit SFR.

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

SP
Stack Pointer **SFR($FE12_H$,09_H)** **Reset value: $FC00_H$**



Field	Bits	Type	Description
1111	[15:12]	r	Fixed at 1111
SP	[11:1]	rwh	Modifiable portion of register SP Specifies the top of the system stack.
0	[0]	r	Fixed at 0

3.6.3.1 Stack Overflow and Underflow

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.

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 that 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 (see Linear Stack).

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 Circular Stack).

The STacK OVerflow Pointer Register STKOV

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

STKOV
STacK OVerflow Pointer **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
1111				**STKOV**											**0**
r				**rw**											**r**

Field	Bits	Type	Description
1111	[15:12]	r	Fixed at 1111
STKOV	[11:1]	rw	Modifiable portion of register STKOV Specifies the segment offset address of the lower limit of the system stack.
0	[0]	r	Fixed at '0'

STKOV can be updated via any instruction that is capable of modifying an SFR.

Note: When a value is MOVED into the stack pointer, NO check against the overflow/ registers is performed.

- **Fatal error indication** treats the stack overflow as a system error and executes 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.

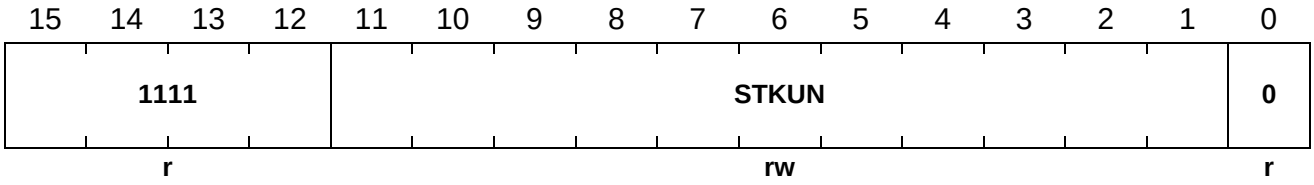
- **Automatic system stack flushing** allows the system stack to be used as a "Stack Cache" for a bigger external user stack. In this case, STKOV should be initialized to a value that represents the desired lowest Top Of Stack (TOS) address plus an offset based on the selected maximum stack size. This offset considers the worst case that will occur when a stack overflow condition is detected just during entry into an ISR, or during an ATOMIC/EXTend sequence. Under these conditions, additional stack word locations are required to push IP, PSW, and CSP for both the ISR and the hardware trap service routine.

The STacK UNderflow Pointer Register STKUN

STKUN is a non-bit-addressable register that is compared to the SP register after each operation that 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 the content of STKUN, a stack underflow hardware trap will occur.

Since the LSB of STKUN is tied to 0 and bits 15 through 12 are tied to 1 by hardware, STKUN register can only contain values from $F000_H$ to $FFFE_H$.

STKUN
STacK UNderflow Pointer **SFR($FE16_H$,$0B_H$)** **Reset value: $FC00_H$**



Field	Bits	Type	Description
1111	[15:12]	r	Fixed at 1111
STKUN	[11:1]	rw	Modifiable portion of register STKUN Specifies the segment offset address of the upper limit of the system stack.
0	[0]	r	Fixed at 0

STKUN can be updated via any instruction capable of modifying an SFR.

Note: When a value is MOVED into the stack pointer, NO check against the overflow/ registers is performed.

- **Fatal error indication** treats the stack underflow as a system error and executes associated trap service routine.

- **Automatic system stack refilling** allows the system stack to be used as a "Stack Cache" for a bigger external user stack. In this case, STKUN should be initialized to a value that represents the desired highest Bottom of Stack address.

Scope of Stack Limit Control

The stack limit control by the register pair STKOV and STKUN detects cases where SP is moved outside the defined stack area either by ADD or SUB instructions, or by PUSH or POP operations (explicit or implicit, e.g., CALL or RET instructions).

This control mechanism is not triggered and no stack trap is generated when:

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

3.6.3.2 Linear Stack

The C166S offers a linear stack option (STKSZ = 111_B) in which the system stack may use the complete DPRAM 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 DPRAM area that may be 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 DPRAM and therefore may only use the address range $00'F600_H$ to $00'FDFE_H$. It is the user's responsibility to restrict the system stack to the DPRAM range.

Note: Stack accesses below the DPRAM area (ESFR space and reserved area) and within address range 00'FE00$_H$ and 00'FFFE$_H$ (SFR space) will have unpredictable results.

3.6.3.3 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 (since the stack empties upward to higher memory locations) is reached. The 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 occurs very infrequently. If this is not true for a given program environment, this technique should not be used because of the overhead of flushing and filling.

The basic mechanism is the transformation of the addresses of a virtual stack area controlled via SP, STKOV, and STKUN to a defined physical stack area within the DPRAM via hardware. This virtual stack area covers all possible locations that SP can point to, i.e. 00'F000$_H$ through 00'FFFE$_H$. STKOV and STKUN accept the same 4-KByte address range. The size of the physical stack area within the DPRAM that is used for standard stack operations is defined via bitfield STKSZ in register SYSCON (see below).

Table 3-15 Circular Stack Address Transformation

STKSZ	Stack Size (Words)	DPRAM 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 1KByte DPRAM)	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 $_B$	1024	00'FDFE$_H$-00'FX00$_H$ (Note: No circular stack) 00'FX00$_H$ represents the lower DPRAM 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 SP (see table **Table 3-15**) 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 **Figure 3-11**).

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 using the internal system stack without any changes, provided that the 256-word area is not exceeded.

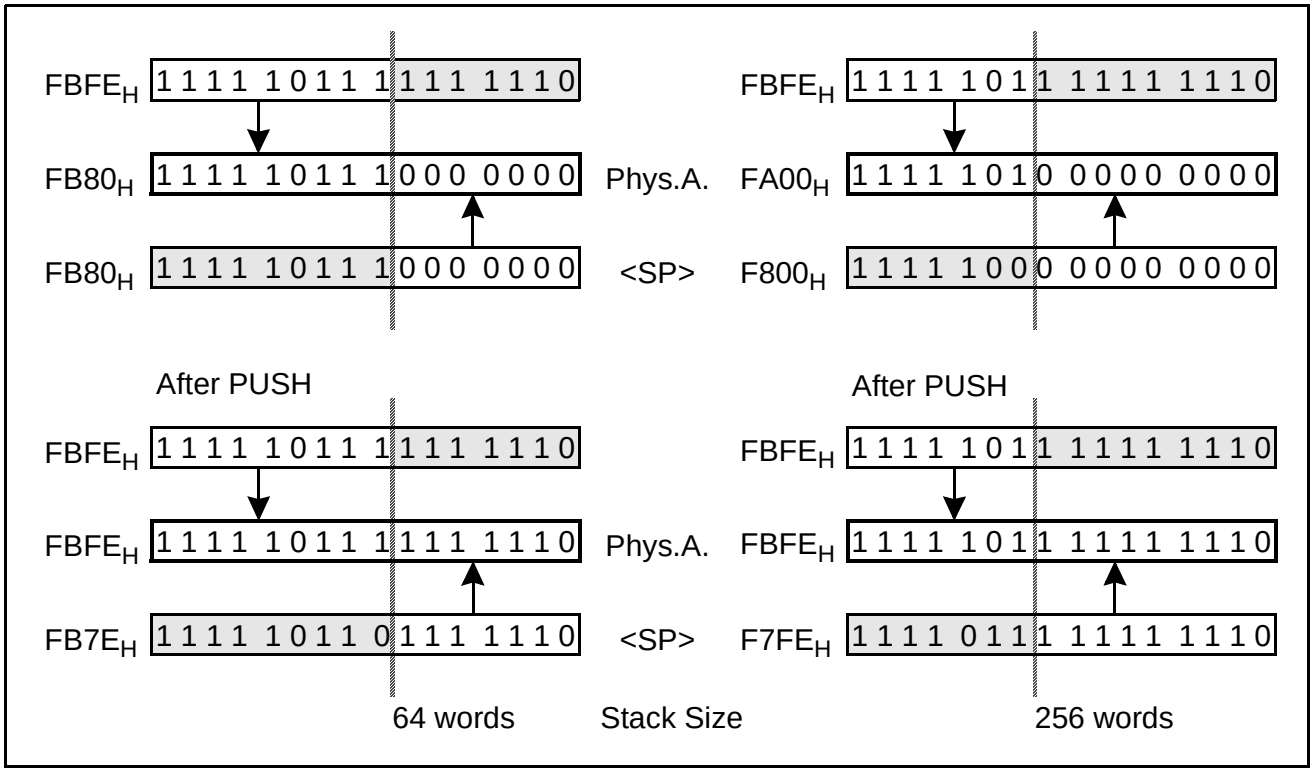


Figure 3-11 Physical Stack Address Generation

The following example demonstrates the circular stack mechanism that 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. The next instruction will push register R2 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 only requires moving that portion of stack data that is to be re-used (i.e. the upper part of the defined stack area) instead of the whole 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 DPRAM for system stack; in this case, the address transformation mechanism is deactivated.

When a boundary is reached, the stack underflow or overflow trap is entered. Inside the trap handler a predetermined portion of the internal stack is moved 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 1/4 to 1/10 the size of the internal stack. Once 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 usage of the circular stack mechanism:

1. Specify the size of the physical system stack area within the DPRAM (bitfield STKSZ in register SYSCON).
2. Define two pointers that 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.
3. Set 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 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.

3.7 Data Processing

All standard arithmetic, shift, and logical operations are performed in the 16-bit Arithmetic and Logic Unit (ALU). In addition to the standard ALU, the ALU of the C166S includes bit manipulation, and a multiply-and-divide unit. Most internal execution blocks have been optimized to perform operations on either 8-bit or 16-bit numbers. Once the pipeline has been filled, most instructions are completed in one machine cycle.

The status flags are automatically updated in the PSW register (see **Section 3.7.6**) after each ALU operation. These flags allow branching upon specific conditions. Support of both signed and unsigned arithmetic is provided by the user-selectable branch test. The status flags are also preserved automatically by the CPU upon entry into an interrupt or trap routine.

3.7.1 Data Types

The C166S supports operations on boolean/bit, bit string, character, and integer data types. Most instructions operate with specific data types, while others are useful for manipulating several data types.

The C166S data formats support all ANSI C data types. In addition, some C compilers support new types that allow the efficient use of the bit-manipulation instructions in embedded control applications.

The C166S directly supports the following data formats:

Table 3-16 CPU data formats

CPU data format	Size (bytes)	Range
BIT	1 bit	0 or 1
BYTE	1	0 to 255U or -128 to +127
WORD	2	0 to 65535U or -32768 to +32767

Table 3-17 ANSI C data types

ANSI C data types	Size (bytes)	Range	CPU data format
bit	1bit	0 or 1	BIT
sfrbit	1bit	0 or 1	BIT
esfrbit	1bit	0 or 1	BIT
signed char	1	-128 to +127	BYTE
unsigned char	1	0 to 255U	BYTE

Table 3-17 ANSI C data types

ANSI C data types	Size (bytes)	Range	CPU data format
sfr	1	0 to 65535U	WORD
esfr	1	0 to 65535U	WORD
signed short	2	-32768 to +32767	WORD
unsigned short	2	0 to 65535U	WORD
bitword	2	0 to 65535U	WORD or BIT
signed int	2	-32768 to +32767	WORD
unsigned int	2	0 to 65535U	WORD
signed long	4	-2147483648 to +2147483647	Not directly supported
unsigned long	4	0 to 4294967295UL	Not directly supported
float	4	+/-1,176E-38 to +/-3,402E+38	Not directly supported
double	8	+/- 2,225E-308 to +/- 1,797E+308	Not directly supported
long double	8	+/- 2,225E-308 to +/- 1,797E+308	Not directly supported
near pointer	2	16/14bits depending on memory model	WORD
far pointer	4	14bits (16k) in any page	Not directly supported
huge pointer	4	24bits (16M)	Not directly supported
shuge pointer	4	24bits (16M), but arithmetic is done 16-bit wide	Not directly supported

3.7.2 Constants

In addition to the powerful addressing modes the C166S instruction set also supports word-wide or byte-wide immediate constants. For an optimum utilization of the available code storage, these constants are represented in the instruction formats by either 3, 4, 8 or 16 bits. The short constants are always zero-extended, while the long constants are truncated if necessary to match the data format required for the particular operation (see table below):

Table 3-18 Constant formats

Mnemonic	Word Operation	Byte Operation
#data3	0000_H + data3	00_H + data3
#data4	0000_H + data4	00_H + data4
#data8	0000_H + data8	data8
#data16	data16	data16 $\wedge$ FF_H
#mask	0000_H + mask	mask

Note: Immediate constants are always signified by a leading #.

3.7.3 The 16-bit Adder/Subtracter, Barrel Shifter and the 16-bit Logic Unit

All standard arithmetic and logical operations are performed by a 16-bit ALU. In case of byte operations signals from bits six and seven of the ALU result are used to control the condition flags. Multiple precision arithmetic is supported by a CARRY-IN signal to the ALU from previously calculated portions of the desired operation.

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

3.7.4 Bit-manipulation Unit

The C166S offers a large number of instructions for bit processing. The special bit-manipulation unit was implemented for this purpose. The bit-manipulation instructions are for efficient control and testing of peripherals. Unlike other microcontrollers, the C166S has instructions that provide direct access to two operands in the bit-addressable space without requiring them to be moved into temporary locations.

The same logical instructions that are available for words and bytes can also be used for bits. The user can 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 require a single machine cycle. In addition, bit field instructions are able to modify multiple bits of one operand in a single instruction.

All instructions that manipulate single bits or bit groups internally use a read-modify-write sequence that accesses the whole word, which contains the specified bit(s).

This method has several consequences:

- Bits can only be modified within the internal address areas, i.e. DPRAM and SFRs. External locations cannot be used with bit instructions. The upper 256 bytes of the SFR area, the ESFR area, and the DPRAM are bit-addressable (see "Memory Organization" **Chapter 4**), i.e., those register bits located within the respective sections can be manipulated directly using bit instructions. The other SFRs must be accessed byte- or word-wise.

Note: All GPRs are bit-addressable independent of the allocation of the register bank via the context pointer CP. Even GPRs which are allocated to not bit-addressable RAM locations provide this feature.

- The read-modify-write approach may be critical with hardware-effected bits. In these cases, the hardware may change specific bits while the read-modify-write operation is in progress, where the write back would overwrite the new bit value generated by the hardware. The solution is either to use the implemented hardware protection (see below) or through special programming (see "Particular Pipeline Effects" **Section 3.8**).

Protected bits are not changed during the read-modify-write sequence, i.e., when hardware sets an interrupt request flag between the read and the write of the read-modify-write sequence, for example. 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 bit.

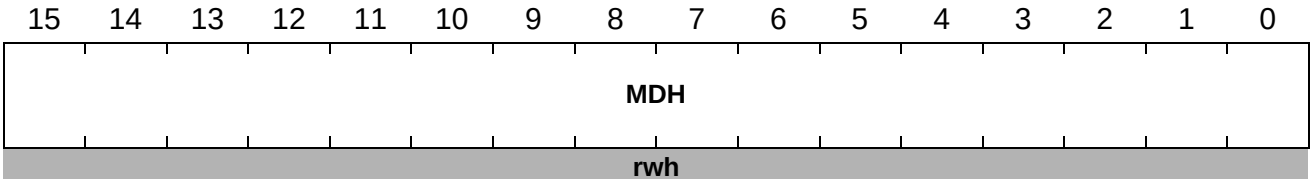
3.7.5 Multiply and Divide Unit

The C166S has a separated multiply-and-divide unit. The multiplication is executed within five machine cycles, while a division takes 20 machine cycles. The multiply-and-divide process is interruptible by an interrupt that has a higher priority level than the current CPU level.

The Multiply/Divide High Word Register (MDH)

The non-bit-addressable Multiply/Divide High word register contains the high word of the 32-bit Multiply/Divide (MD) register, which is used by the CPU when it performs a multiplication or a division using implicit addressing (DIV, DIVL, DIVLU, DIVU, MUL, MULU). After an implicitly-addressed multiplication, this register represents the high-order 16 bits of the 32-bit result. For long divisions, MDH must be loaded with the high-order 16 bits of the 32-bit dividend before the division has started. After any division, MDH represents the 16-bit remainder.

MDH
Multiply/Divide High Word **SFR(FE0C$_H$,06$_H$)** **Reset value: 0000$_H$**



Field	Bits	Type	Description
MDH	[15:0]	rwh	High part of MD The high order 16 bits of the 32-bit multiply and divide register MD.

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

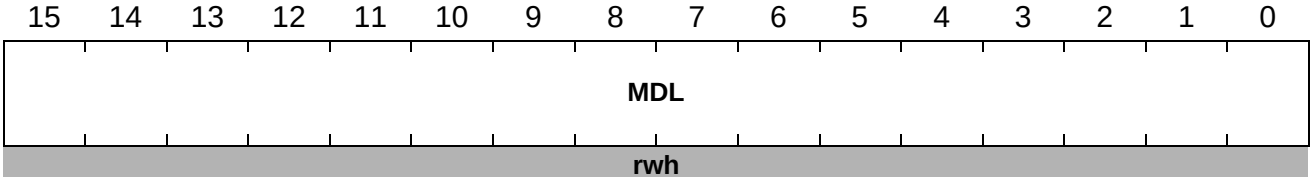
When a multiplication or division is interrupted before its completion and when a new multiply or divide operation is to be performed within the Interrupt Service Routine (ISR), the contents of MDH must be saved along with the contents of registers MDL and MDC to avoid erroneous results.

The Multiply/Divide Low Word Register (MDL)

The non-bit-addressable Multiply/Divide Low word register contains the low word of the 32-bit multiply/divide MD register which is used by the CPU when it performs a multiplication or a division using implicit addressing (DIV, DIVL, DIVLU, DIVU, MUL, MULU). After a multiplication, this 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 has started. After any division, MDL represents the 16-bit quotient.

MDL
Multiply/Divide Low Word **SFR(FE0E$_H$,07$_H$)** **Reset value: 0000$_H$**



Field	Bits	Type	Description
MDL	[15:0]	rwh	Low part of MD The low order 16 bits of the 32-bit multiply and divide register MD.

Whenever this register is updated via software, the MDRIU flag in the MDC register 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 ISR, the contents of MDL must be saved along with the contents of registers MDH and MDC to avoid erroneous results.

The Multiply/Divide Control Register (MDC)

The bit-addressable 16-bit Multiply/Divide Control 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. MDC is updated by hardware during each single cycle of a multiply or divide instruction.

MDC
Multiply/Divide Control **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
0	0	0	0	0	0	0	0	!!	!!	!!	MDRIU	!!	!!	!!	!!
r	r	r	r	r	r	r	r	rwh	rwh	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
MDRIU	[4]	rwh	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.
!!	[7],[6], [5],[3], [2],[1], [0]	rwh	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), and then it must be cleared 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 that might be of interest for 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 other than described above. Otherwise, correct continuation of an interrupted multiply or divide operation cannot be guaranteed.

Multiplication or division is performed simply 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 of 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 MD (MDH) must be moved into the register file or memory first, in order 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 above save sequence and the restore sequence after COPYL are required only if the current routine could have interrupted a previous routine that 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 about how to restart the instruction is contained in this register. Register MDC must be cleared to be initialized correctly for a subsequent multiplication or division. The old MDC contents must be popped from the stack before the RETI instruction is executed.

For a 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 MD must be loaded. The result is also stored in 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 tested implicitly 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 (e.g. scheduler switches to another task), MULIP must be set or cleared according to the context of the task that is switched to.*

3.7.6 The Processor Status Word Register (PSW)

The bit-addressable Processor Status Word register reflects the current status of the microcontroller. Two groups of bits represent the current ALU status and the current CPU interrupt status. One separate bit (USR0) within PSW is provided as a general-purpose flag.

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	S1	0	0	0	USR0	MUL IP	E	Z	V	C	N
rwh				rw	rw	r	r	r	rw	rwh	rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
ILVL	[15:12]	rwh	CPU Priority LeVeL 0_H Lowest priority ... ... F_H Highest priority
IEN	[11]	rw	Interrupt/PEC ENable Bit (globally) 0 Interrupt/PEC requests are disabled 1 Interrupt/PEC requests are enabled
S1	[10]	rw	Reserved for system
USR0	[6]	rwh	General Purpose Flag May be used by application
MULIP	[5]	r	MULtiplication/division In Progress 0 No multiplication/division in process 1 Multiplication/division has been interrupted
E	[4]	rwh	End of table Flag 0 Source operand is neither 8000_h nor 80_h 1 Source operand is 8000_h or 80_h
Z	[3]	rwh	Zero Flag 0 ALU result is not zero 1 ALU result is zero
V	[2]	rwh	OVerflow Flag 0 No overflow produced 0 Overflow produced

Field	Bits	Type	Description
C	[1]	rwh	Carry Flag 0 No carry/borrow bit produced 1 Carry/borrow bit produced
N	[0]	rwh	Negative Result 0 ALU result is not negative 1 ALU result is negative

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

The condition flags (N, C, V, Z, E) within the PSW indicate the ALU status resulting from the ALU operation last performed. They are set by the majority of instructions according to specific rules depending on the ALU operation or data movement.

After execution of an instruction that explicitly updates PSW, the condition flags may no longer represent an actual CPU status. An explicit write operation to PSW supersedes the condition flag values that are implicitly generated by the CPU. An explicit read access to PSW returns the value of PSW after execution of the previous instruction.

Note: After reset, all of the ALU status bits are cleared.

- **N-Flag:** For the majority of 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 (MSB) of the specified word or byte data type has been generated during a subtraction. Subtraction is performed by the ALU as a 2's-complement addition. 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 can not cause a carry flag to be set.
  For shift and rotate operations, the C-flag represents the value of the bit shifted out last. If a shift count of zero is specified, the C-flag will be cleared. The C-flag is also cleared for a prioritize 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:** The addition, subtraction, and 2's complement operations set the V-flag to 1 if the result exceeds the range of 16-bit signed numbers for word operations (-8000_H to $+7FFF_H$), or 8-bit signed numbers for byte operations (-80_H to $+7F_H$). Otherwise, the V-flag is cleared. The result of an integer addition, integer subtraction, or 2's complement operation 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 can not be represented in a word data type; otherwise, it is cleared. A division by zero will always cause an overflow. Unlike the division result, the result of multiplication is valid regardless of V-flag value.
  Since 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. By only using the C-flag, a rounding error caused by a shift-right operation can be estimated as up to one half of the LSB of the result. In conjunction with the V-flag, the C-flag allows the rounding error to be evaluated with a finer resolution (see table below).
  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 3-19 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 addition and subtraction with carry, the Z-flag is set to 1 only if the Z-flag already contains a 1 as a result of a previous operation, and if the result of the current ALU operation equals zero. This mechanism supports multiple precision calculations.
  For Boolean bit operations with only one operand, the Z-flag represents the logical negation of the previous state of the specified bit. For Boolean bit operations with two operands, the Z-flag represents the logical NORing of the two specified bits. For the prioritize operation, the Z-flag indicates whether or not the second operand was zero.

- **E-Flag:** End of table flag. The E-flag can be altered by instructions that perform ALU or data movement operations. The E-flag is cleared by those instructions that can not

be reasonably used for table search operations. In all other cases, the E-flag value depends on the value of the source operand to signify whether or not the end of a search table is reached. If the value of the source operand of an instruction equals the lowest negative number that depends on 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 the entrance into an ISR when a multiply or divide ALU operation was interrupted before completion. Depending on the state of the MULIP bit, the hardware decides whether a multiplication or division must be continued or not after the end of an interrupt service. The MULIP bit is overwritten with the contents of the stacked MULIP-flag when RETurn-from-Interrupt-instruction (RETI) is executed. This normally means that the MULIP-flag is cleared again after that.

Note: The MULIP flag is a part of the task environment. When the ISR does not return to the interrupted multiply/divide instruction (e.g. in case of a task scheduler that 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 (IEN) bit makes it possible to enable (IEN=1) or disable (IEN=0) interrupts globally. The four-bit LeVeL field (ILVL) specifies the priority of the current CPU activity. The priority level is updated by hardware upon entry into an ISR, but it can also be modified via software to prevent other interrupts from being acknowledged. If an priority level 15 has been assigned to the CPU, it has the highest possible priority, and thus the current CPU operation cannot be interrupted except by hardware traps or external non-maskable interrupts. For details, please refer to **Section 3.4** "Interrupt and Trap Functions" .

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

3.8 Instruction Pipeline

The instruction pipeline of the C166S partitions instruction processing into four stages. Each of these has a separate 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 local memory, DPRAM, 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 that implicitly access the system stack, the SP register is either decremented or incremented, as specified. For branch instructions, the IP and the CSP 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. 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 the instruction.

4th –>WRITE BACK: In this stage, all external operands and the remaining operands within the DPRAM space are written back.

A particularity of the C166S are the so-called "injected instructions." These injected instructions are generated internally by the machine to provide the time needed to process instructions that cannot be processed within one machine cycle. They are automatically injected 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, too. 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.

3.8.1 Particular Pipeline Effects

Since up to 4 different instructions are processed simultaneously, additional hardware has been included in the C166S to prevent a loss of performance when dealing with all causal dependencies on instructions in different pipeline stages. This extra hardware resolves most of the possible conflicts (e.g. multiple usage of buses) in a time-optimized way, preventing the pipeline dependencies from becoming noticeable to the user in most cases. However, in some rare cases, attention from the programmer is required specifically because the C166S is a pipelined machine.

3.8.1.1 General considerations

Due to pipeline, Read operation, performed by next instruction, can takes place before a Write operation, performed by the earlier instruction. While this fact can lead to some general problems, an extra CPU hardware - the forwarding mechanism - deals with the operand read/write addresses, and also controls the pipeline when needed. Especially, if there are sequential write and read operations on the same address, the read is hold until write is executed. Thus in most cases the pipeline behavior is resolved and made transparent. So the user is assured, that after writing to a location, next read from the same location will return the correct result.

However, there are write operations, which are changing some important parameters of the system: configuration, data pages, stack location, register banks etc. If they are not followed by a read from the same address, the CPU will not recognize the need of holding the pipeline while write is finished. So, the effect of this operation will not be seen immediately within next instruction.

After writing to a memory location (*MEMLOC*), non-critical instruction(s) must follow, before the first instruction, which will be affected by that write operation:

```
In     :Writing to MEMLOC
In+1   :Non-critical instruction(s)   ;MEMLOC still holds the old value
       :....
In+d   :Any instruction               ;new MEMLOC-VALUE is already effective
```

Non-critical instruction means instruction, which execution does not depend on the writing to *MEMLOC*. The most "non-critical" is NOP instruction, as doing nothing. The programmer has to be always aware not to place some critical instruction within that gap of one (or more) machine cycles, while the write operation still has no effect. Also, the delays caused by pipeline conflicts can be used for other instructions in order to optimize performance.

3.8.1.2 Specific cases with core registers

When writing to an internal core register, the time needed before the new value becomes effective depends only on the pipeline. This time is usually one, as exception two machine cycles, what means one or two non-critical instructions to be executed. Below are described some specific situations when changing important system registers.

- **Address Pointer Updating**

Indirect addressing modes use a GPR value to generate the address of the source and/or destination operand. If this GPR is updated explicitly by the preceding instruction, one NOP instruction is automatically inserted.

```
In      :ADD   R0,#0002h     ;increment address pointer GPR 0
Iinject :NOP                 ;automatically injected NOP
In+1    :MOV   R2,[R0]       ;use GPR 0 for indirect addressing
```

To improve performance, an instruction not using this new GPR as a destination operand can be inserted between an explicit GPR-changing and a subsequent instruction using an indirect addressing mode.

```
In      :ADD   R0,#0002h     ;increment address pointer GPR 0
In+1    :....                ;must not be an instruction updating GPR 0
In+2    :MOV   R2,[R0]       ;use GPR 0 for indirect addressing
```

- **Context Pointer Updating**

An instruction that calculates a physical GPR operand address via the CP register is incapable of using a new CP value that is to be updated by the preceding instruction. Thus, to make sure that the new CP value is used, at least two instructions must be inserted between an instruction that changes the CP and a subsequent instruction that uses the GPR, as shown in the following example:

```
In      :SCXT CP,#0FC00h     ;select a new context
In+1    :....                ;must not be an instruction using a GPR
In+2    :....                ;must not be an instruction using a GPR
In+3    :MOV   R0,#dataX     ;write to GPR 0 in the new context
```

- **Data Page Pointer Updating**

An instruction that calculates a physical operand address via a particular DPPn (n=0 to n=3) register is not capable of using a new DPPn register value that is to be updated by the preceding instruction. Thus, to make sure that the new DPPn register value is used, at least one instruction must be inserted between a DPPn-changing instruction and a subsequent instruction that implicitly uses DPPn via a long or indirect addressing mode, as shown in the following example:

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

- **Explicit Stack Pointer Updating**

Neither the RET nor POP instruction is capable of correctly using a new SP register value, which is to be 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 instruction that explicitly writes to SP, and any of the afore mentioned subsequent instructions that implicitly use the SP, 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
```

Furthermore none of the RETI, RETS or RETP instructions are capable of correctly using a new SP register value, which is to be updated by one or both of the two immediately preceding instructions. Thus, in order to use the new SP register value without erroneously performed stack accesses, at least two instructions must be inserted between an instruction that explicitly writes to SP, and any of the afore mentioned subsequent instructions that implicitly use the SP, 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  :....              ;must not be an instruction popping operands
                         ;from the system stack
In+3  :RETP  R0          ;return from subroutine and pop word value from
                         ;new top of stack into R0
```

Most of the potential conflicts, if a change of SP value is immediately followed by a writing to the stack (instructions PUSH, CALL, SCXT, TRAP) are solved internally by CPU logic. The only exceptions are CALLS and PCALL instructions, which require one preceding instruction not using updated SP, as shown below:

```
In    :MOV  SP,#0FA40H   ;select a new top of stack
In+1  :....              ;must not be an instruction using
                         ;the new address of system stack
In+2  :PCALL R3,sub_addr ;push R3 value and return address at the new
                         ;top of stack and call subroutine
```

- **Controlling Interrupts**

Software modifications (implicit or explicit) of the PSW are done in the execute phase of instructions. In order to maintain fast interrupt responses, however, the current interrupt prioritization round does not consider these changes, i.e., an interrupt request may be acknowledged after the instruction that disables interrupts via IEN or ILVL, or after the following instructions. Therefore, time-critical instruction sequences 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 uninterruptable 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 1 instruction also applies for enabling the interrupt system; i.e., no interrupt requests are acknowledged until the instruction following the enabling instruction.

- **External Memory Access Sequences**

The effect described here will only become noticeable when watching the external memory access sequences on the external bus (e.g., 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 diverge from the sequence of the corresponding external memory accesses performed by the EBC, due to the predefined priority of external memory accesses:

1. Write data
2. Fetch code
3. Read data

- **Initialization of Port Pins**

Modifications of the direction 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 accessing the whole port, instructions modifying 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
```

- **Changing the System Configuration**

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

Note: As a rule, instructions that change local memory mapping should be executed from DPRAM 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 external memory area.

3.8.1.3 Common portable solution

When writing to an external memory location, the time needed before the new value becomes effective depends also on the overall system performance. In general more extra cycle(s) will be needed in case of lower peripheral bus speed to cover the delay additionally caused. Special attention has to be paid, when writing to external SFRs, such as peripheral control registers, which have effect over the system behavior - for example enabling/disabling transfer, interrupts etc.

To assure that the critical write operation has been completed, before making use of it, it is recommended next to make a read operation on the same memory location. The reasons are:

- the CPU will recognize the need of holding pipeline while the write operation is completed;
- as reading from the same location, execution-time scales in respect to bus speed. So there is no need to think about how fast that bus is and one instruction is enough in all cases, to assure effectiveness of that write for the next instructions.

3.8.2 Instruction State Times

Instruction pipelining usually reduces the average instruction processing time (typically within a range of 1-4 machine cycles). However, there are some rare cases, where a particular pipeline situation causes the processing time for a single instruction to be extended either by a half or by one machine cycle. Although this additional time represents only a tiny part of the total program execution time, it might be of interest to avoid these pipeline-caused time delays in time-critical program modules.

The time to execute an instruction depends on where the instruction is fetched from, and where possible operands are read from or written to. The fastest processing mode of the C166S is to execute a program fetched from the internal code memory. In that 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 EBC, which works in parallel with the CPU. This section summarizes execution times.

The table below shows the minimum execution times required to process an instruction fetched from the internal local memory, the DPRAM, or external memory. These execution times apply to most instructions except some of the branches, the multiplication, the division, and a special move instruction. In case of internal local memory program execution, the execution time does not depend 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 3-20 Minimum Execution Times

	Instruction Fetch		Word Operand Access	
Memory Area	**Word Instruction**	**Doubleword Instruction**	**Read from**	**Write to**
Internal local memory	1[1)]	1[1)]	1[1)]	1[1)]
DPRAM	1[1)]	2	1[1)]	1[1)]
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

1) Minimum execution time for instruction fetch and operand accesses. Nevertheless the minimum execution time of an instruction remains 2 clock cycles (one machine cycle).

Execution from the DPRAM provides flexibility in terms of loadable and modifiable code. The execution time from external memory depends strongly 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 Local Memory operand accesses (same for byte and word operand accesses)
- DPRAM 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 local memory space
- Testing branch conditions immediately after PSW writes

3.9 Dedicated CSFRs

The Constant Zeros Register ZEROS

All bits of this bit-addressable register are fixed at 0 by hardware. This register is read-only. Register ZEROS can be used as a register-addressable constant of all zeros for bit manipulation or mask generation. It can be accessed via any instruction capable of accessing an SFR.

Zeros
Constant Zeros Register **SFR($FF1C_H$,$8E_H$)** **Reset value: 0000_H**

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

Field	Bits	Type	Description
0	[all]	r	Fixed at Zero

The Constant Ones Register ONES

All bits of this bit-addressable register are fixed at 1 by hardware. This register is 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 which is capable of accessing an SFR.

ONES
Constant Ones Register **SFR($FF1E_H$,$8F_H$)** **Reset value: $FFFF_H$**

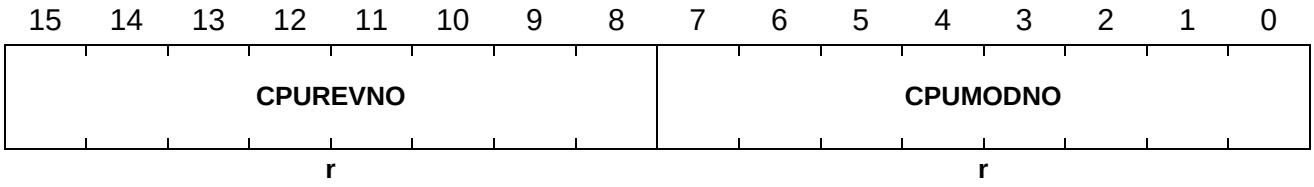
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Field	Bits	Type	Description
1	[all]	r	Fixed at 1

CPU Identification register CPUID

This 16-bit register contains the module and revision number of the implemented C166S module.

CPUID
CPU Identification Register ESFR($F00C_H$,$E\text{-}06_H$) Reset value: $04??_H$



Field	Bits	Type	Description
CPUREVNO	[15:8]	r	Module Number 04_H C166S core module number
CPUMODNO	[7:0]	r	Version Number Version Number, starts with 01_H and is incremented with every new core version

3.10 Summary of CPU Registers

This section summarizes all registers in the C166S. There are two kind of registers, the General Purpose Registers (GPR) and the CPU-Special Function Registers (CSFR). The GPRs are the working registers of the arithmetic and logic operations and may be also used as address pointers indirect addressing modes. The CSFRs are the control registers of the C166S.

For easy reference, the CSFRs are listed in this section by address (to identify a register at a given. address) and by name (to find an address of a specific register).

3.10.1 General Purpose Registers

The GPRs are the working registers of the C166S. All GPRs are bit addressable.

Table 3-21 Addressing modes to access Word-GPRs

Name	Physical Address	8-Bit Address	4-Bit Address	Description	Reset Value
R0	(CP)+0	$F0_H$	0_H	General-Purpose word Register R0	$UUUU_H$
R1	(CP)+2	$F1_H$	1_H	General-Purpose word Register R1	$UUUU_H$
R2	(CP)+4	$F2_H$	2_H	General-Purpose word Register R2	$UUUU_H$
R3	(CP)+6	$F3_H$	3_H	General-Purpose word Register R3	$UUUU_H$
R4	(CP)+8	$F4_H$	4_H	General-Purpose word Register R4	$UUUU_H$
R5	(CP)+10	$F5_H$	5_H	General-Purpose word Register R5	$UUUU_H$
R6	(CP)+12	$F6_H$	6_H	General-Purpose word Register R6	$UUUU_H$
R7	(CP)+14	$F7_H$	7_H	General-Purpose word Register R7	$UUUU_H$
R8	(CP)+16	$F8_H$	8_H	General-Purpose word Register R8	$UUUU_H$
R9	(CP)+18	$F9_H$	9_H	General-Purpose word Register R9	$UUUU_H$
R10	(CP)+20	FA_H	A_H	General-Purpose word Register R10	$UUUU_H$
R11	(CP)+22	FB_H	B_H	General-Purpose word Register R11	$UUUU_H$
R12	(CP)+24	FC_H	C_H	General-Purpose word Register R12	$UUUU_H$
R13	(CP)+26	FD_H	D_H	General-Purpose word Register R13	$UUUU_H$
R14	(CP)+28	FE_H	E_H	General-Purpose word Register R14	$UUUU_H$

The first 8 GPRs (R7...R0) may be also accessed byte-wise. Unlike SFRs, writing to a GPR byte does not affect another byte of the GPR.

The following byte-wise accessible registers have special names. .

Table 3-22 Addressing modes to access Byte-GPRs

Name	Physical Address	8-Bit Address	4-Bit Address	Description	Reset Value
RL0	(CP)+0	$F0_H$	0_H	General-Purpose byte Register RL0	UU_H
RH0	(CP)+1	$F1_H$	1_H	General-Purpose byte Register RL1	UU_H
RL1	(CP)+2	$F2_H$	2_H	General-Purpose byte Register RL2	UU_H
RH1	(CP)+3	$F3_H$	3_H	General-Purpose byte Register RL3	UU_H
RL2	(CP)+4	$F4_H$	4_H	General-Purpose byte Register RL4	UU_H
RH2	(CP)+5	$F5_H$	5_H	General-Purpose byte Register RL5	UU_H
RL3	(CP)+6	$F6_H$	6_H	General-Purpose byte Register RL6	UU_H
RH3	(CP)+7	$F7_H$	7_H	General-Purpose byte Register RL7	UU_H
RL4	(CP)+8	$F8_H$	8_H	General-Purpose byte Register RL8	UU_H
RH4	(CP)+9	$F9_H$	9_H	General-Purpose byte Register RL9	UU_H
RL5	(CP)+10	FA_H	A_H	General-Purpose byte Register RL10	UU_H
RH5	(CP)+11	FB_H	B_H	General-Purpose byte Register RL11	UU_H
RL6	(CP)+12	FC_H	C_H	General-Purpose byte Register RL12	UU_H
RH6	(CP)+13	FD_H	D_H	General-Purpose byte Register RL13	UU_H
RL7	(CP)+14	FE_H	E_H	General-Purpose byte Register RL14	UU_H
RH7	(CP)+15	FF_H	F_H	General-Purpose byte Register RL15	UU_H

3.10.2 Core Special Function Registers Ordered by Name

The following table lists all CSFRs in alphabetical order. **Bit-addressable** CSFRs are marked with the letter "**b**" in column "Name".
CSFRs within the **Extended CSFR-Space** (ECSFRs) are marked with the letter "**E**" in column "8-Bit Address".

Name	Physical Address	8-Bit Address	Description	Reset Value
CP	$FE10_H$	08_H	Context Pointer	$FC00_H$
CPUID	$F00C_H$	$E\text{-}06_H$	CPU Identification Register	$04YY_H$ 1)
CSP	$FE08_H$	04_H	Code Segment Pointer (8 bits, not directly writable)	0000_H
DPP0	$FE00_H$	00_H	Data Page Pointer 0 (10 bits)	0000_H
DPP1	$FE02_H$	01_H	Data Page Pointer 1 (10 bits)	0001_H
DPP2	$FE04_H$	02_H	Data Page Pointer 2 (10 bits)	0002_H
DPP3	$FE06_H$	03_H	Data Page Pointer 3 (10 bits)	0003_H
MDCb	$FF0E_H$	87_H	Multiply Divide Control Register	0000_H
MDH	$FE0C_H$	06_H	Multiply Divide High Word	0000_H
MDL	$FE0E_H$	07_H	Multiply Divide Low Word	0000_H
ONESb	$FF1E_H$	$8F_H$	Constant Value 1's Register (read only)	$FFFF_H$
PSWb	$FF10_H$	88_H	Program Status Word	0000_H
SP	$FE12_H$	09_H	Stack Pointer	$FC00_H$
STKOV	$FE14_H$	$0A_H$	Stack Overflow Register	$FA00_H$
STKUN	$FE16_H$	$0B_H$	Stack Underflow Register	$FC00_H$
SYSCON	$FF12_H$	89_H	System/CPU Control Register	$YYYY_H$ 2)
TFRb	$FFAC_H$	$D6_H$	Trap Flag Register	0000_H
ZEROSb	$FF1C_H$	$8E_H$	Constant Value 0's Register (read only)	0000_H

1) YY: defined by implemented CPU version

2) YYYY: defined by reset and system configuration

3.10.3 Core Special Function Registers ordered by Address

The following table lists all CSFRs which are implemented in the C166S ordered by their physical address. **Bit-addressable** CSFRs are marked with the letter "**b**" in column "Name".
CSFRs within the **Extended SFR-Space** (ESFRs) are marked with the letter "**E**" in column "8-Bit Address".

Name	Physical Address	8-Bit Address	Description	Reset Value
CPUID	$F00C_H$	$E\text{-}06_H$	CPU Identification Register	$04YY_H$ 1)
DPP0	$FE00_H$	00_H	Data Page Pointer 0 (10 bits)	0000_H
DPP1	$FE02_H$	01_H	Data Page Pointer 1 (10 bits)	0001_H
DPP2	$FE04_H$	02_H	Data Page Pointer 2 (10 bits)	0002_H
DPP3	$FE06_H$	03_H	Data Page Pointer 3 (10 bits)	0003_H
CSP	$FE08_H$	04_H	Code Segment Pointer (8 bits, not directly writable)	0000_H
MDH	$FE0C_H$	06_H	Multiply Divide High Word	0000_H
MDL	$FE0E_H$	07_H	Multiply Divide Low Word	0000_H
CP	$FE10_H$	08_H	Context Pointer	$FC00_H$
SP	$FE12_H$	09_H	Stack Pointer	$FC00_H$
STKOV	$FE14_H$	$0A_H$	Stack Overflow Register	$FA00_H$
STKUN	$FE16_H$	$0B_H$	Stack Underflow Register	$FC00_H$
MDCb	$FF0E_H$	87_H	Multiply Divide Control Register	0000_H
PSWb	$FF10_H$	88_H	Program Status Word	0000_H
SYSCON	$FF12_H$	89_H	System/CPU Control Register	$YYYY_H$ 2)
ZEROSb	$FF1C_H$	$8E_H$	Constant Value 0's Register (read only)	0000_H
ONESb	$FF1E_H$	$8F_H$	Constant Value 1's Register (read only)	$FFFF_H$
TFRb	$FFAC_H$	$D6_H$	Trap Flag Register	0000_H

1) YY: defined by implemented CPU version
2) YYYY: defined by reset and system configuration

3.10.4 Register Overview C166S Interrupt and Peripheral Event Controller

The following table lists all xSFRs which are implemented in the C166S Interrupt and Peripheral Event Controller.

Table 3-23 Register Overview - C166S Interrupt and Peripheral Event Controller

Register Name	Register Description	Module Block
PECSN0	PEC Pointer 0 Segment Address Reg.	PEC Pointer
PECSN1	PEC Pointer 1 Segment Address Reg.	PEC Pointer
PECSN...	PEC Pointer ... Segment Address Reg.	PEC Pointer
PECSN7	PEC Pointer 7 Segment Address Reg.	PEC Pointer
PECSN8[1)]	PEC Pointer 8 Segment Address Reg.	PEC Pointer
PECSN...[1)]	PEC Pointer ... Segment Address Reg.	PEC Pointer
PECSN15[1)]	PEC Pointer 15 Segment Address Reg.	PEC Pointer
PECC0	PEC Channel 0 Control Register	PEC Control
PECC...	PEC Channel ... Control Register	PEC Control
PECC7	PEC Channel 7 Control Register	PEC Control
PECC8[1)]	PEC Channel 8 Control Register	PEC Control
PECC...[1)]	PEC Channel ... Control Register	PEC Control
PECC15[1)]	PEC Channel 15 Control Register	PEC Control
IRQ0IC[1)]	Interrupt 0 Control Register	Arbitration Control
...	...	...
IRQ63IC[1)]	Interrupt 63 Control Register	Arbitration Control
IRQ64IC[1)]	Interrupt 64 Control Register	Arbitration Control
IRQ65IC[1)]	Interrupt 65 Control Register	Arbitration Control
...	...	...
IRQ111IC[1)]	Interrupt 111 Control Register	Arbitration Control

[1)] The implementation and assignment of these Interrupt/PEC Control Register is product specific.

4 Memory Organization

The memory space of the C166S has 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/DRAM (where integrated), DPRAM, the internal Special Function Register (SFR) and Extended Special Function Register (ESFR) areas, and external memory are mapped into one common address space.

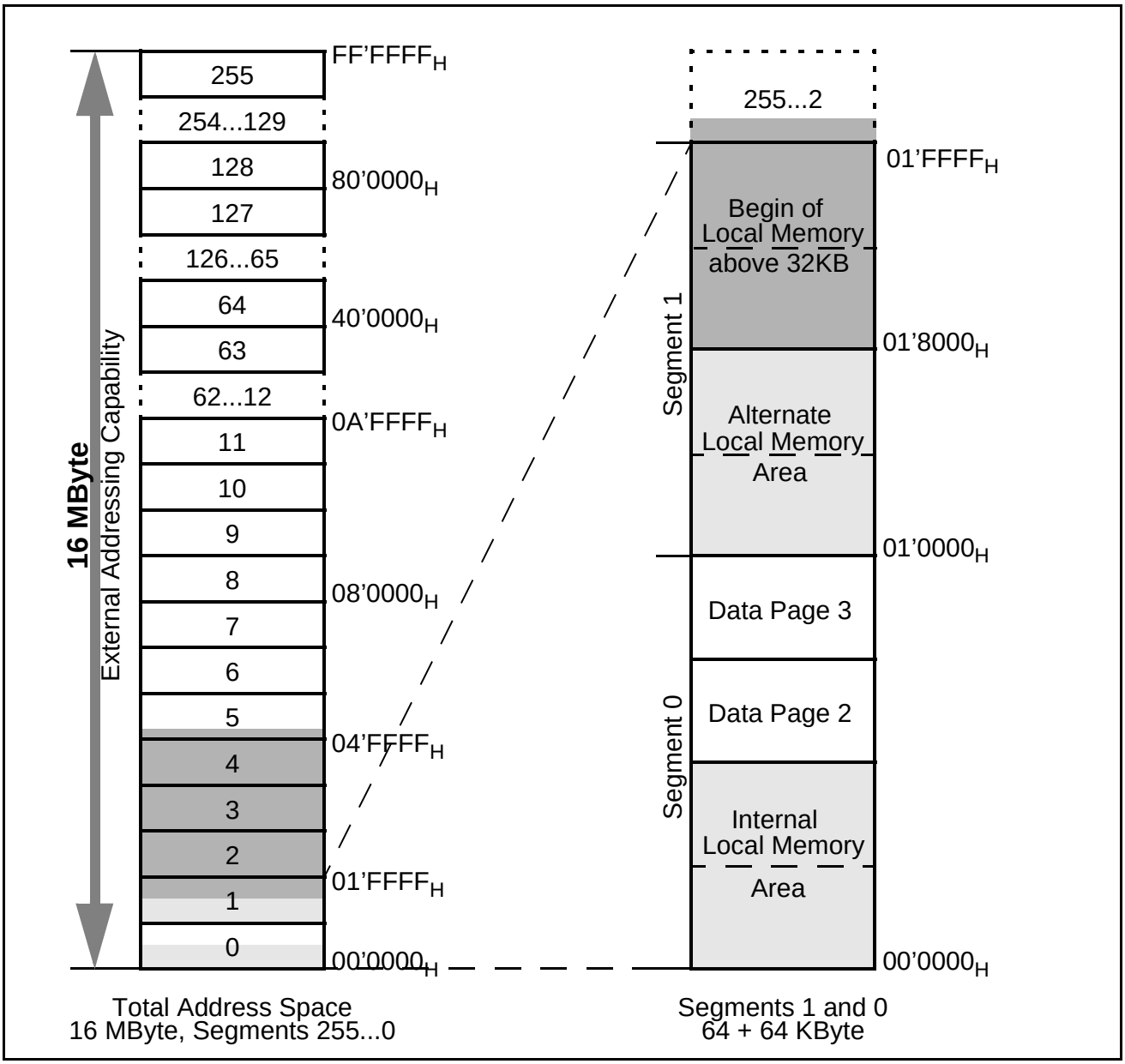


Figure 4-1 C166S Address Space Overview

The C166S has 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 4-1**).

Most internal memory areas are mirrored into segment 0, the system segment. The upper 4 KBytes of segment 0 ($00'F000_H$-$00'FFFF_H$) are the SFRs and ESFRs and the DPRAM areas. The lower 32 KByte of segment 0 (00'0000H-00'7FFFH) may be occupied by a part of the on-chip program memory and is called the internal Local Memory (LM) area. This LM 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 LM 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 LM area on devices without LM will produce unpredictable results.

4.1 Data Organization in Memory

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. Instruction double-words are stored in ascending memory locations as two subsequent words, without any restrictions (non-aligned). Single bits are always stored in the specified bit position at a word address. The memory and registers store data and instructions in little-endian byte order (the least significant bytes are at lower addresses) The byte ordering is illustrated in **Figure 4-2**. Bit position 0 is the least significant bit of the byte at an even byte address, and bit position 15 is the most significant bit of the byte at the next odd byte address. Bit addressing is supported for a part of the SFRs, a part of the DPRAM and for the General-Purpose Registers (GPRs).

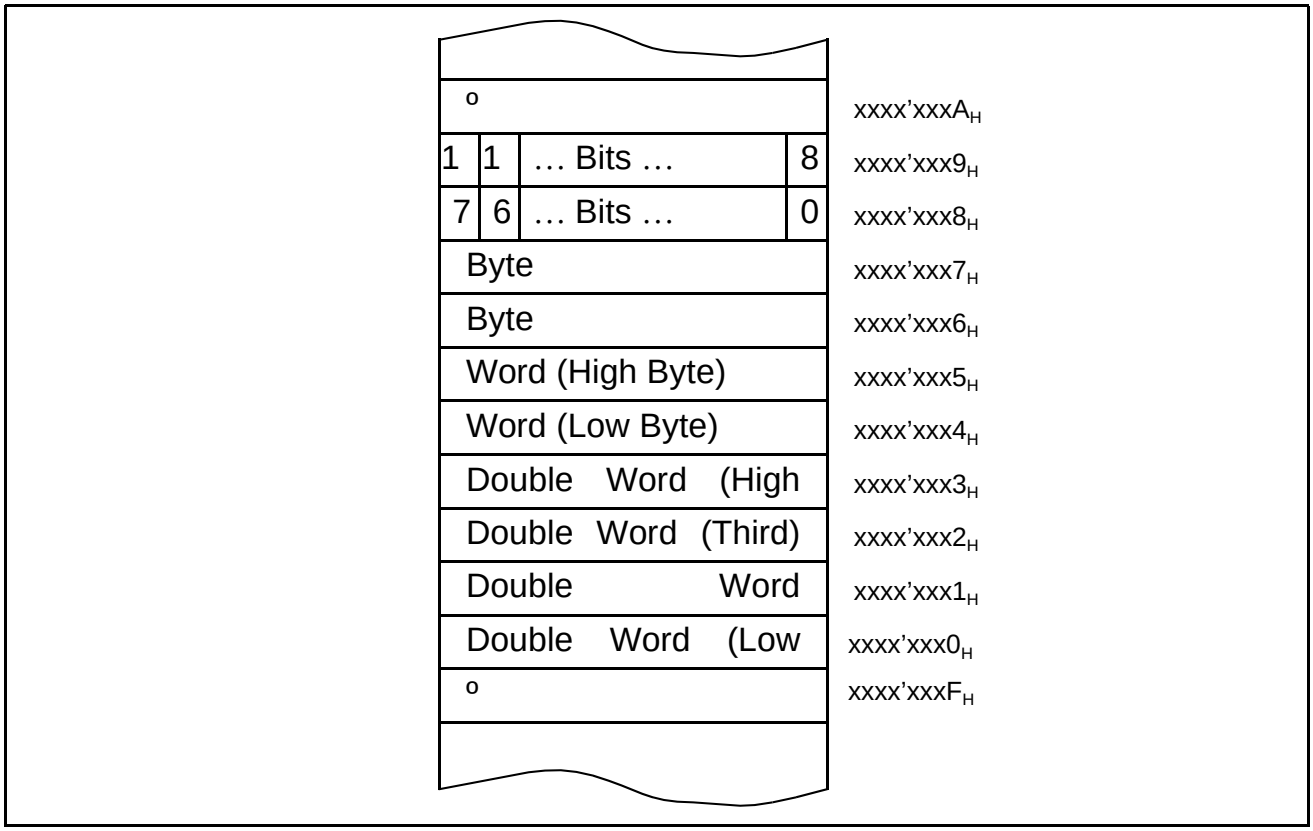


Figure 4-2 Storage of Words, Byte and Bits in a Byte Organized Memory

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

4.2 Internal Local Memory Area

The C166S reserves an address area of variable size (depending on the device configuration) for on-chip Local Memory (LM). The internal LM can be ROM, SRAM, flash or DRAM.

The internal LM may be enabled, disabled or mapped into segment 0 or segment 1 under software control.

Internal LM accesses are enabled or disabled globally via bit ROMEN in the SYSCON register. This bit is set during reset according to the level on external pin $\overline{EA}$, or may be altered via software. If enabled, the internal lower 32K of LM area occupies the lower 32 KByte of either segment 0 or segment 1. This mapping is controlled by bit ROMS1 in register SYSCON.

Note: The size of the internal LM area may be independent of the size of the actual implemented LM. Devices with less than 32 KBytes of LM or with no LM at all will have this 32-KByte area occupied if the LM is enabled. Devices with a larger LM provide the mapping option only for the internal LM area.

Devices with an LM size above 32 KByte expand the LM area from the middle of segment 1, i.e., starting at address $01'8000_H$.

The internal LM 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 LM 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 LM 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 the LM operands. Any word data access is made to an even byte address. Any double-word access is made to a modulo-4 address (even-word address). The highest possible word data storage location in the LM is $xxxx'xxFE_H$; the highest double-word location is $xxxx'xxFC_H$.

The internal LM 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 internal LM.

4.3 DPRAM and SFR-Area

The C166S differentiates between the internal data memory (DPRAM) and the internal peripheral areas. The DPRAM and the SFR areas are located within data page 3, and provide fast accesses using one dedicated Data Page Pointer (DPP) (see **Figure 4-3**).

Note: Code accesses are not possible from the SFR areas.

4.3.1 Data Memories

The DPRAM is a volatile memory available mainly for data storage. It serves for:

- GPR banks
- Variable and other data storage
- System and user stacks
- PEC source and destination pointers

A 3-KByte memory area (00'$F200_H$-00'$FE00_H$) is reserved for the DPRAM. The upper 256 Bytes of the DPRAM (00'$FD00_H$-00'$FDFF_H$) and the GPRs of the current bank are provided for single-bit storage, and thus they are bit addressable (see shaded blocks in **Figure 4-3**). Any word and byte data in the DPRAM can be accessed via indirect or long 16-bit addressing modes if the selected DPP register points to data page 3. Any word data access is made on an even byte address. The highest possible word data storage location in the DPRAM is 00'$FDFE_H$.

The highest possible code storage location in the DPRAM is either 00'$FDFE_H$ for single-word instructions, or 00'$FDFC_H$ for double-word instructions (but this is the bit-addressable area, which should not be used for code). The respective location must contain a branch instruction (unconditional), because sequential boundary crossing from DPRAM to the SFR area is not supported and causes erroneous results.

4.3.2 Special Function Register Areas

The functions of the CPU, the bus interface, the I/O ports, and the on-chip peripherals of the C166S are controlled via a number of SFRs. These SFRs are arranged within two 512-Byte areas. The first register block, the SFR area, is located in the 512 Bytes above the DPRAM (00'$FFFF_H$-00'$FE00_H$). The second register block, the ESFR area, is located in the 512 Bytes below the DPRAM (00'$F1FF_H$-00'$F000_H$).

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

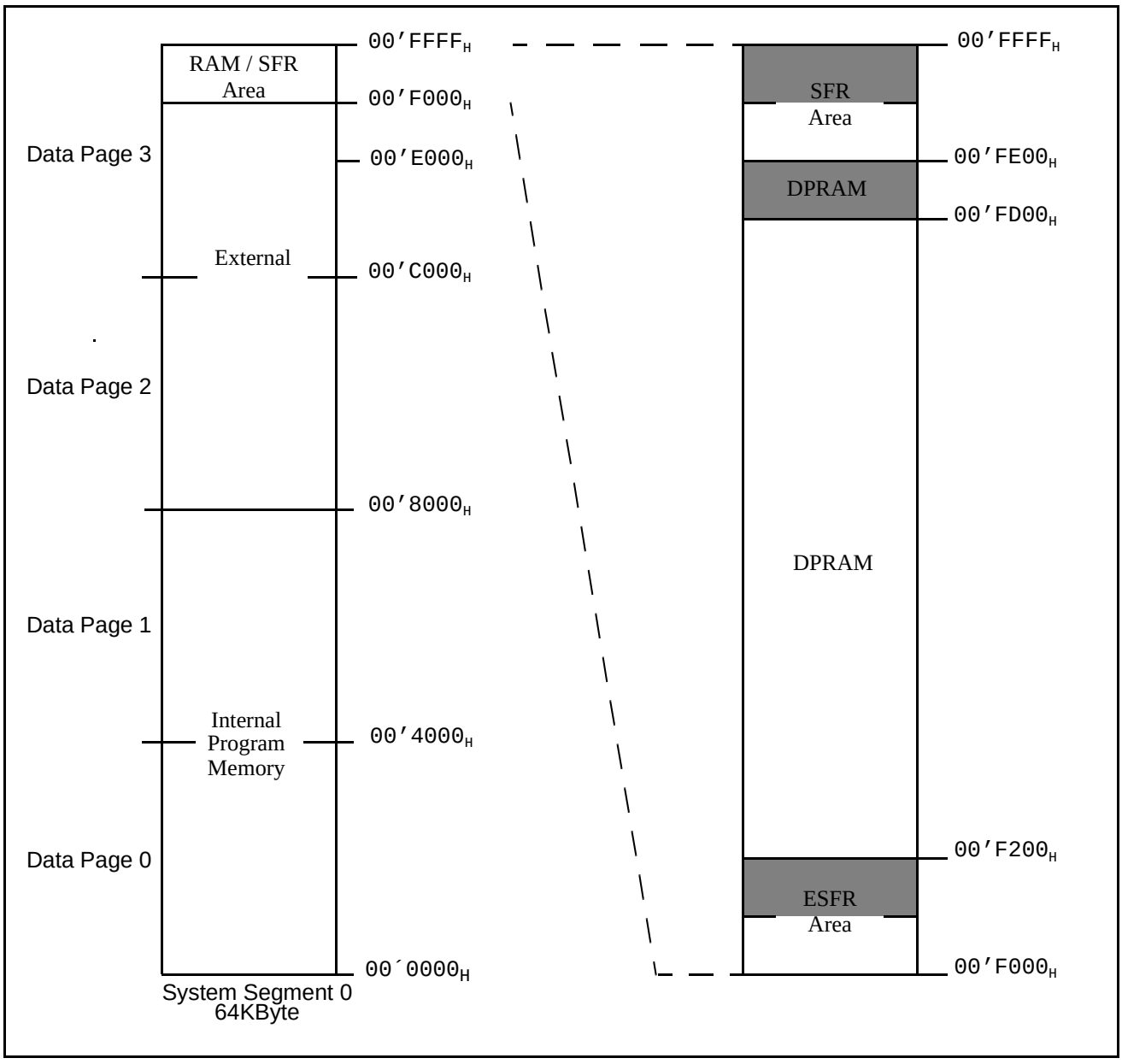


Figure 4-3 DPRAM and SFR Areas

Note: High-byte accesses of SFRs using the 8-bit offset addressing mode are not possible.

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

Note: GPRs can be accessed using the 8-bit offset addressing mode, but the GPRs are not mapped into the SFR and ESFR memory area. Using the corresponding long address instead of a GPR access executes an internal peripheral bus access.

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, the EXTend Register (EXTR) instruction is required before accessing registers in the ESFR area, in order to switch the short-addressing mechanism from the standard SFR area to the ESFR 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.

Example 1:

```
EXTR    #4                  ;Switch to ESFR area for the next 4 instructions
MOV     ODP2, #data16       ;ODP2 (ESFR register) uses 8-bit reg addressing
BFLDL   DP6, #mask, #data8  ;DP6 (ESFR register) bit addressing for bit fields
BSET    DP6.7               ;DP6 (ESFR register) bit addressing for single bits
MOV     T8REL, R1           ;T8REL uses 16-bit address, R1 is duplicated…
                            ;…and also accessible via the ESFR mode
                            ;(EXTR is not required for this access)
;-------  ;-----------------  ;The scope of the EXTR #4 instruction ends here!
MOV     T8REL, R1           ;T8REL uses 16-bit address, R1 is duplicated…
                            ;…and does not require switching
```

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

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

4.3.3 PEC Source and Destination Pointers

The 16 (24/32) word locations for the 8 (12/16) PEC channels in the DPRAM from 00'FCE0h (00'FCD0h/00'FCC0h) to 00'FCFEh (just below the bit-addressable section) are provided as source and destination address pointers for data transfers on the 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.

Whenever a PEC data transfer is performed, the pair of SRCPx and DSTPx selected by the specified PEC channel number is accessed independent of the current DPP register contents; also, the locations referred to by these pointers are accessed independently of the current DPP register contents. If a PEC channel is not used, the corresponding pointer location area is available and can be used for word or byte data storage or for instructions.

For more details about use of SRCPx and DSTPx for PEC data transfer, see "Interrupt and Exception Execution" **Section 3.4**.

4.4 External Memory Space

The C166S CPU can use an address space of up to 16 MBytes. Only parts of this address space are occupied by the internal LM, DPRAM and the IO/SFR area. All addresses not used for this kind of on-chip memory or for registers may reference external memory locations. This external memory space is accessed via the **B**us **C**ontroller (BC).

The BC is the bus-bridge between the C166S CPU and the external/internal bus interfaces. The external bus interface allows access to external (off-chip) peripherals and additional off-chip volatile and non-volatile memories. The external bus interface may further limit the amount of addressable off-chip memory. The internal bus interface provides an internal system bus that allows the on-chip integration of customer-specific peripherals, volatile and non-volatile memories. The availability of the internal and external bus interfaces depends on the functionality of the integrated BC.

4.4.1 External data accesses

External word and byte data can be accessed only 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, and double-word accesses to modulo-4 byte addresses (even word address).

External memory is not provided for single-bit storage and therefore it is not bit-addressable.

4.5 Crossing Memory Boundaries

The address space of the C166S CPU is divided implicitly 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 that represent different kinds of memory (if provided at all). These memory areas are the DPRAM, the internal IO, the internal LM (if available), and the external memory.

Accessing subsequent data locations that belong to different memory areas is not fully supported, and may therefore lead to erroneous results. There is no problem if the memory boundaries are word-aligned. However, when executing code, the different memory areas (internal memory areas and external memory) must be switched explicitly via branch instructions. Sequential boundary crossing is not supported and may lead to erroneous results.

Segments are contiguous 64-KByte blocks. 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. Larger sequential programs make sure that the highest used code location of a segment contains an unconditional branch instruction to the next following segment, to prevent the prefetcher from trying to leave the current segment.

Data Pages are contiguous 16-KByte blocks. They are referenced via the Data Page Pointers DPP3...0 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 that is used for the current access is selected via the two upper bits of the 16-bit data address. Subsequent 16-bit data addresses that cross the 16-KByte data page boundaries will use different data page pointers, while the physical locations need not be contiguous within memory.

4.6 System Stack

The system stack must be defined within the DPRAM. The size of the system stack is controlled by bitfield STKSZ in register SYSCON (see table below).

Table 4-1 System stack size

<STKSZ>	Stack Size (Words)	DPRAM 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.
$1\,1\,1_B$	1536	$00'FDFE_H$-$00'F200_H$ (Note: No circular stack)

For all system stack operations, the stack memory is accessed via the Stack Pointer (SP). The system stack implementation in the C166S is from high to low memory. The system stack grows downward as it is filled. The SP register is decremented first each time data is pushed on the system stack, and incremented after each time the data is pulled from the system stack. Only word accesses are supported to the system stack.

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

A STacK OVerflow (STKOV) register and a STacK UNderflow (STKUN) register are provided to control the lower and upper limits of the selected stack area. These two stack boundary registers can be used not only for protection against data destruction, but also to implement a circular stack with hardware-supported system stack flushing and filling (except for option STKSZ=111).

4.6.1 Data Organization in General Purpose Registers

The memory-mapped GPRs use a block of 16 consecutive words within the DPRAM Segment 0. 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 up to 16 byte GPRs (RL0, RH0, …, RL7, RH7). The 16-byte GPRs are mapped onto the first 8 word GPRs (see table below).

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 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.

The C166S supports register bank (context) switching. Multiple memory-mapped register banks can physically exist within the DPRAM at the same time. Only the register bank selected by the CP register is active at a given time, however. Selecting a new active register bank is simply done by updating the CP register.

Figure 4-4 Mapping of General-Purpose Registers to DPRAM Addresses

DPRAM 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

A particular Switch ConteXT (SCXT) instruction performs register bank switching and automatic saving of the previous context. The number of implemented register banks (arbitrary sizes) is limited only by the size of the available DPRAM.

4.7 SFR / ESFR Table

The following table lists all SFRs/ESFRs which are implemented in the C166S V1 SubS R1 ordered by their physical address.

Table 4-2 SFR/ESFR Table (ordered by physical address)

Physical Address	Name	Type[1]	8-bit Addr[2]	Description	Reset Value[3]
$F000_H$	reserved	ESFR	00_H	reserved - do not use	
$F002_H$	reserved	ESFR	01_H	reserved - do not use	
$F004_H$	reserved	ESFR	02_H	reserved - do not use	
$F006_H$	reserved	ESFR	03_H	reserved - do not use	
$F008_H$	reserved	ESFR	04_H	reserved - do not use	
$F00A_H$	reserved	ESFR	05_H	reserved - do not use	
$F00C_H$	CPUID	ESFR	06_H	CPU Identification Register	0410_H
$F00E_H$	reserved	ESFR	07_H	reserved - do not use	
$F010_H$	reserved	ESFR	08_H	reserved - do not use	
$F012_H$	reserved	ESFR	09_H	reserved - do not use	
$F014_H$	XADRS1	ESFR	$0A_H$	XBUS Address Select Register 1	0000_H
$F016_H$	XADRS2	ESFR	$0B_H$	XBUS Address Select Register 2	0000_H
$F018_H$	XADRS3	ESFR	$0C_H$	XBUS Address Select Register 3	0000_H
$F01A_H$	XADRS4	ESFR	$0D_H$	XBUS Address Select Register 4	0000_H
$F01C_H$	XADRS5	ESFR	$0E_H$	XBUS Address Select Register 5	0000_H
$F01E_H$	XADRS6	ESFR	$0F_H$	XBUS Address Select Register 6	0000_H
$F020_H$		ESFR	10_H		
$F022_H$		ESFR	11_H		
$F024_H$	XPERCON	ESFR	12_H	XBUS Peripheral Control Register	0000_H
$F026_H$		ESFR	13_H		
$F028_H$		ESFR	14_H		
$F02A_H$		ESFR	15_H		
$F02C_H$		ESFR	16_H		
$F02E_H$		ESFR	17_H		

Table 4-2 SFR/ESFR Table (ordered by physical address) (cont'd)

Physical Address	Name	Type[1)]	8-bit Addr[2)]	Description	Reset Value[3)]
$F030_H$		ESFR	18_H		
$F032_H$		ESFR	19_H		
$F034_H$		ESFR	$1A_H$		
$F036_H$		ESFR	$1B_H$		
$F038_H$		ESFR	$1C_H$		
$F03A_H$		ESFR	$1D_H$		
$F03C_H$		ESFR	$1E_H$		
$F03E_H$		ESFR	$1F_H$		
$F040_H$		ESFR	20_H		
$F042_H$		ESFR	21_H		
$F044_H$		ESFR	22_H		
$F046_H$		ESFR	23_H		
$F048_H$		ESFR	24_H		
$F04A_H$		ESFR	25_H		
$F04C_H$		ESFR	26_H		
$F04E_H$		ESFR	27_H		
$F050_H$		ESFR	28_H		
$F052_H$		ESFR	29_H		
$F054_H$		ESFR	$2A_H$		
$F056_H$		ESFR	$2B_H$		
$F058_H$		ESFR	$2C_H$		
$F05A_H$		ESFR	$2D_H$		
$F05C_H$		ESFR	$2E_H$		
$F05E_H$		ESFR	$2F_H$		
$F060_H$		ESFR	30_H		
$F062_H$		ESFR	31_H		
$F064_H$		ESFR	32_H		
$F066_H$		ESFR	33_H		

Table 4-2 SFR/ESFR Table (ordered by physical address) (cont'd)

Physical Address	Name	Type[1]	8-bit Addr[2]	Description	Reset Value[3]
$F068_H$	COMDATA	ESFR	34_H	Cerberus Communication Mode Register	0000_H
$F06A_H$	RWDATA	ESFR	35_H	Cerberus RW Mode Data Register	0000_H
$F06C_H$	IOSR	ESFR	36_H	Cerberus status register	0000_H
$F06E_H$		ESFR	37_H		
$F070_H$		ESFR	38_H		
$F072_H$		ESFR	39_H		
$F074_H$		ESFR	$3A_H$		
$F076_H$		ESFR	$3B_H$		
$F078_H$		ESFR	$3C_H$		
$F07A_H$		ESFR	$3D_H$		
$F07C_H$		ESFR	$3E_H$		
$F07E_H$		ESFR	$3F_H$		
$F080_H$		ESFR	40_H		
$F082_H$		ESFR	41_H		
$F084_H$		ESFR	42_H		
$F086_H$		ESFR	43_H		
$F088_H$		ESFR	44_H		
$F08A_H$		ESFR	45_H		
$F08C_H$		ESFR	46_H		
$F08E_H$		ESFR	47_H		
$F090_H$		ESFR	48_H		
$F092_H$		ESFR	49_H		
$F094_H$		ESFR	$4A_H$		
$F096_H$		ESFR	$4B_H$		
$F098_H$		ESFR	$4C_H$		
$F09A_H$		ESFR	$4D_H$		
$F09C_H$		ESFR	$4E_H$		

Table 4-2 SFR/ESFR Table (ordered by physical address) (cont'd)

Physical Address	Name	Type[1]	8-bit Addr[2]	Description	Reset Value[3]
$F09E_H$		ESFR	$4F_H$		
$F0A0_H$		ESFR	50_H		
$F0A2_H$		ESFR	51_H		
$F0A4_H$		ESFR	52_H		
$F0A6_H$		ESFR	53_H		
$F0A8_H$		ESFR	54_H		
$F0AA_H$		ESFR	55_H		
$F0AC_H$		ESFR	56_H		
$F0AE_H$		ESFR	57_H		
$F0B0_H$	SSC0TB	ESFR	58_H	SSC0 Transmit Buffer (WO)	0000_H
$F0B2_H$	SSC0RB	ESFR	59_H	SSC0 Receive Buffer (RO)	0000_H
$F0B4_H$	SSC0BR	ESFR	$5A_H$	SSC0 Baudrate Register	0000_H
$F0B6_H$	SSC0PISEL	ESFR	$5B_H$	SSC0 Port Input Selection Register	0000_H
$F0B8_H$	IRQ96IC	ESFR	$5C_H$	IRQ96 Interrupt Control Register	0000_H
$F0BA_H$	IRQ97IC	ESFR	$5D_H$	IRQ97 Interrupt Control Register	0000_H
$F0BC_H$	IRQ98IC	ESFR	$5E_H$	IRQ98 Interrupt Control Register	0000_H
$F0BE_H$	IRQ99IC	ESFR	$5F_H$	IRQ99 Interrupt Control Register	0000_H
$F0C0_H$	IRQ100IC	ESFR	60_H	IRQ100 Interrupt Control Register	0000_H
$F0C2_H$	IRQ101IC	ESFR	61_H	IRQ101 Interrupt Control Register	0000_H
$F0C4_H$	IRQ102IC	ESFR	62_H	IRQ102 Interrupt Control Register	0000_H
$F0C6_H$	IRQ103IC	ESFR	63_H	IRQ103 Interrupt Control Register	0000_H
$F0C8_H$	IRQ104IC	ESFR	64_H	IRQ104 Interrupt Control Register	0000_H
$F0CA_H$	IRQ105IC	ESFR	65_H	IRQ105 Interrupt Control Register	0000_H

Table 4-2 SFR/ESFR Table (ordered by physical address) (cont'd)

Physical Address	Name	Type[1]	8-bit Addr[2]	Description	Reset Value[3]
$F0CC_H$	IRQ106IC	ESFR	66_H	IRQ106 Interrupt Control Register	0000_H
$F0CE_H$	IRQ107IC	ESFR	67_H	IRQ107 Interrupt Control Register	0000_H
$F0D0_H$	IRQ108IC	ESFR	68_H	IRQ108 Interrupt Control Register	0000_H
$F0D2_H$	IRQ109IC	ESFR	69_H	IRQ109 Interrupt Control Register	0000_H
$F0D4_H$	IRQ110IC	ESFR	$6A_H$	IRQ110 Interrupt Control Register	0000_H
$F0D6_H$	IRQ111IC	ESFR	$6B_H$	IRQ111 Interrupt Control Register	0000_H
$F0D8_H$	DTIDR	ESFR	$6C_H$	Task ID register	0000_H
$F0DA_H$		ESFR	$6D_H$		
$F0DC_H$		ESFR	$6E_H$		
$F0DE_H$		ESFR	$6F_H$		
$F0E0_H$		ESFR	70_H		
$F0E2_H$		ESFR	71_H		
$F0E4_H$		ESFR	72_H		
$F0E6_H$		ESFR	73_H		
$F0E8_H$		ESFR	74_H		
$F0EA_H$		ESFR	75_H		
$F0EC_H$	DCMPSP	ESFR	76_H	Select and Programming Register for DCMPx	0000_H
$F0EE_H$	DCMPDP	ESFR	77_H	Data Programming Register for DCMPx	0000_H
$F0F0_H$	DTREVT	ESFR	78_H	Specifies hardware triggers and action	0000_H
$F0F2_H$	DEXEVT	ESFR	79_H	Specifies action if external break pin is asserted	0000_H

Table 4-2 SFR/ESFR Table (ordered by physical address) (cont'd)

Physical Address	Name	Type[1]	8-bit Addr[2]	Description	Reset Value[3]
$F0F4_H$	DSWEVT	ESFR	$7A_H$	Specifies action if DEBUG instruction is executed	0000_H
$F0F6_H$	reserved	ESFR	$7B_H$	reserved - do not use	
$F0F8_H$	DIP	ESFR	$7C_H$	Instruction pointer register	0000_H
$F0FA_H$	DIPX	ESFR	$7D_H$	Instruction pointer register extension	3000_H
$F0FC_H$	DBGSR	ESFR	$7E_H$	Debug status register	0000_H
$F0FE_H$	reserved	ESFR	$7F_H$	reserved - do not use	
$F100_H$	DP0L	ESFR-b	80_H	P0L Direction Control Register	00_H
$F102_H$	DP0H	ESFR-b	81_H	P0H Direction Control Register	00_H
$F104_H$	DP1L	ESFR-b	82_H	P1L Direction Control Register	00_H
$F106_H$	DP1H	ESFR-b	83_H	P1H Direction Control Register	00_H
$F108_H$	reserved	ESFR-b	84_H	reserved - do not use	
$F10A_H$	reserved	ESFR-b	85_H	reserved - do not use	
$F10C_H$	reserved	ESFR-b	86_H	reserved - do not use	
$F10E_H$	reserved	ESFR-b	87_H	reserved - do not use	
$F110_H$	reserved	ESFR-b	88_H	reserved - do not use	
$F112_H$	reserved	ESFR-b	89_H	reserved - do not use	
$F114_H$	XBCON1	ESFR-b	$8A_H$	XBUS Control register 1	0000_H
$F116_H$	XBCON2	ESFR-b	$8B_H$	XBUS Control register 2	0000_H
$F118_H$	XBCON3	ESFR-b	$8C_H$	XBUS Control register 3	0000_H
$F11A_H$	XBCON4	ESFR-b	$8D_H$	XBUS Control register 4	0000_H
$F11C_H$	XBCON5	ESFR-b	$8E_H$	XBUS Control register 5	0000_H
$F11E_H$	XBCON6	ESFR-b	$8F_H$	XBUS Control register 6	0000_H
$F120_H$	IRQ64IC	ESFR-b	90_H	IRQ64 Interrupt Control Register	0000_H
$F122_H$	IRQ65IC	ESFR-b	91_H	IRQ65 Interrupt Control Register	0000_H
$F124_H$	IRQ66IC	ESFR-b	92_H	IRQ66 Interrupt Control Register	0000_H
$F126_H$	IRQ67IC	ESFR-b	93_H	IRQ67 Interrupt Control Register	0000_H

Table 4-2 SFR/ESFR Table (ordered by physical address) (cont'd)

Physical Address	Name	Type[1)]	8-bit Addr[2)]	Description	Reset Value[3)]
$F128_H$	IRQ68IC	ESFR-b	94_H	IRQ68 Interrupt Control Register	0000_H
$F12A_H$	IRQ69IC	ESFR-b	95_H	IRQ69 Interrupt Control Register	0000_H
$F12C_H$	IRQ70IC	ESFR-b	96_H	IRQ70 Interrupt Control Register	0000_H
$F12E_H$	IRQ71IC	ESFR-b	97_H	IRQ71 Interrupt Control Register	0000_H
$F130_H$	IRQ72IC	ESFR-b	98_H	IRQ72 Interrupt Control Register	0000_H
$F132_H$	IRQ73IC	ESFR-b	99_H	IRQ73 Interrupt Control Register	0000_H
$F134_H$	IRQ74IC	ESFR-b	$9A_H$	IRQ74 Interrupt Control Register	0000_H
$F136_H$	IRQ75IC	ESFR-b	$9B_H$	IRQ75 Interrupt Control Register	0000_H
$F138_H$	IRQ76IC	ESFR-b	$9C_H$	IRQ76 Interrupt Control Register	0000_H
$F13A_H$	IRQ77IC	ESFR-b	$9D_H$	IRQ77 Interrupt Control Register	0000_H
$F13C_H$	IRQ78IC	ESFR-b	$9E_H$	IRQ78 Interrupt Control Register	0000_H
$F13E_H$	IRQ79IC	ESFR-b	$9F_H$	IRQ79 Interrupt Control Register	0000_H
$F140_H$	IRQ80IC	ESFR-b	$A0_H$	IRQ80 Interrupt Control Register	0000_H
$F142_H$	IRQ81IC	ESFR-b	$A1_H$	IRQ81 Interrupt Control Register	0000_H
$F144_H$	IRQ82IC	ESFR-b	$A2_H$	IRQ82 Interrupt Control Register	0000_H
$F146_H$	IRQ83IC	ESFR-b	$A3_H$	IRQ83 Interrupt Control Register	0000_H
$F148_H$	IRQ84IC	ESFR-b	$A4_H$	IRQ84 Interrupt Control Register	0000_H
$F14A_H$	IRQ85IC	ESFR-b	$A5_H$	IRQ85 Interrupt Control Register	0000_H
$F14C_H$	IRQ86IC	ESFR-b	$A6_H$	IRQ86 Interrupt Control Register	0000_H
$F14E_H$	IRQ87IC	ESFR-b	$A7_H$	IRQ87 Interrupt Control Register	0000_H
$F150_H$	IRQ88IC	ESFR-b	$A8_H$	IRQ88 Interrupt Control Register	0000_H
$F152_H$	IRQ89IC	ESFR-b	$A9_H$	IRQ89 Interrupt Control Register	0000_H
$F154_H$	IRQ90IC	ESFR-b	AA_H	IRQ90 Interrupt Control Register	0000_H
$F156_H$	IRQ91IC	ESFR-b	AB_H	IRQ91 Interrupt Control Register	0000_H
$F158_H$	IRQ92IC	ESFR-b	AC_H	IRQ92 Interrupt Control Register	0000_H
$F15A_H$	IRQ93IC	ESFR-b	AD_H	IRQ93 Interrupt Control Register	0000_H
$F15C_H$	IRQ94IC	ESFR-b	AE_H	IRQ94 Interrupt Control Register	0000_H
$F15E_H$	IRQ95IC	ESFR-b	AF_H	IRQ95 Interrupt Control Register	0000_H
$F160_H$	IRQ48IC	ESFR-b	$B0_H$	IRQ48 Interrupt Control Register	0000_H

Table 4-2 SFR/ESFR Table (ordered by physical address) (cont'd)

Physical Address	Name	Type[1]	8-bit Addr[2]	Description	Reset Value[3]
$F162_H$	IRQ49IC	ESFR-b	$B1_H$	IRQ49 Interrupt Control Register	0000_H
$F164_H$	IRQ50IC	ESFR-b	$B2_H$	IRQ50 Interrupt Control Register	0000_H
$F166_H$	IRQ51IC	ESFR-b	$B3_H$	IRQ51 Interrupt Control Register	0000_H
$F168_H$	IRQ52IC	ESFR-b	$B4_H$	IRQ52 Interrupt Control Register	0000_H
$F16A_H$	IRQ53IC	ESFR-b	$B5_H$	IRQ53 Interrupt Control Register	0000_H
$F16C_H$	IRQ54IC	ESFR-b	$B6_H$	IRQ54 Interrupt Control Register	0000_H
$F16E_H$	IRQ55IC	ESFR-b	$B7_H$	IRQ55 Interrupt Control Register	0000_H
$F170_H$	IRQ56IC	ESFR-b	$B8_H$	IRQ56 Interrupt Control Register	0000_H
$F172_H$	IRQ57IC	ESFR-b	$B9_H$	IRQ57 Interrupt Control Register	0000_H
$F174_H$	IRQ58IC	ESFR-b	BA_H	IRQ58 Interrupt Control Register	0000_H
$F176_H$	IRQ59IC	ESFR-b	BB_H	IRQ59 Interrupt Control Register	0000_H
$F178_H$	IRQ60IC	ESFR-b	BC_H	IRQ60 Interrupt Control Register	0000_H
$F17A_H$	IRQ40IC	ESFR-b	BD_H	IRQ40 Interrupt Control Register	0000_H
$F17C_H$	IRQ41IC	ESFR-b	BE_H	IRQ41 Interrupt Control Register	0000_H
$F17E_H$	IRQ15IC	ESFR-b	BF_H	IRQ15 Interrupt Control Register	0000_H
$F180_H$	EOPIC	ESFR-b	$C0_H$	End of PEC Transfer Interrupt Control Register	0000_H
$F182_H$	IRQ42IC	ESFR-b	$C1_H$	IRQ42 Interrupt Control Register	0000_H
$F184_H$	IRQ61IC	ESFR-b	$C2_H$	IRQ61 Interrupt Control Register	0000_H
$F186_H$	IRQ36IC	ESFR-b	$C3_H$	IRQ36 Interrupt Control Register	0000_H
$F188_H$	IRQ47IC	ESFR-b	$C4_H$	IRQ47 Interrupt Control Register	0000_H
$F18A_H$	IRQ43IC	ESFR-b	$C5_H$	IRQ43 Interrupt Control Register	0000_H
$F18C_H$	IRQ62IC	ESFR-b	$C6_H$	IRQ62 Interrupt Control Register	0000_H
$F18E_H$	IRQ37IC	ESFR-b	$C7_H$	IRQ37 Interrupt Control Register	0000_H
$F190_H$	IRQ45IC	ESFR-b	$C8_H$	IRQ45 Interrupt Control Register	0000_H
$F192_H$	IRQ44IC	ESFR-b	$C9_H$	IRQ44 Interrupt Control Register	0000_H
$F194_H$	IRQ63IC	ESFR-b	CA_H	IRQ63 Interrupt Control Register	0000_H
$F196_H$	IRQ38IC	ESFR-b	CB_H	IRQ38 Interrupt Control Register	0000_H
$F198_H$	IRQ46IC	ESFR-b	CC_H	IRQ46 Interrupt Control Register	0000_H

Table 4-2 SFR/ESFR Table (ordered by physical address) (cont'd)

Physical Address	Name	Type[1]	8-bit Addr[2]	Description	Reset Value[3]
$F19A_H$	WDTIC	ESFR-b	CD_H	Watchdog Timer Interrupt Control Register	0000_H
$F19C_H$	S0TBIC	ESFR-b	CE_H	ASC0 Transmit Buffer Interrupt Control Register	0000_H
$F19E_H$	IRQ39IC	ESFR-b	CF_H	IRQ39 Interrupt Control Register	0000_H
$F1A0_H$		ESFR-b	$D0_H$		
$F1A2_H$		ESFR-b	$D1_H$		
$F1A4_H$		ESFR-b	$D2_H$		
$F1A6_H$		ESFR-b	$D3_H$		
$F1A8_H$		ESFR-b	$D4_H$		
$F1AA_H$		ESFR-b	$D5_H$		
$F1AC_H$		ESFR-b	$D6_H$		
$F1AE_H$		ESFR-b	$D7_H$		
$F1B0_H$		ESFR-b	$D8_H$		
$F1B2_H$		ESFR-b	$D9_H$		
$F1B4_H$		ESFR-b	DA_H		
$F1B6_H$	ASC0PISEL	ESFR-b	DB_H	ASC0 Port Input Selection Register	0000_H
$F1B8_H$		ESFR-b	DC_H		
$F1BA_H$		ESFR-b	DD_H		
$F1BC_H$		ESFR-b	DE_H		
$F1BE_H$		ESFR-b	DF_H		
$F1C0_H$		ESFR-b	$E0_H$		
$F1C2_H$		ESFR-b	$E1_H$		
$F1C4_H$		ESFR-b	$E2_H$		
$F1C6_H$		ESFR-b	$E3_H$		
$F1C8_H$		ESFR-b	$E4_H$		
$F1CA_H$		ESFR-b	$E5_H$		
$F1CC_H$		ESFR-b	$E6_H$		

Table 4-2 SFR/ESFR Table (ordered by physical address) (cont'd)

Physical Address	Name	Type[1]	8-bit Addr[2]	Description	Reset Value[3]
$F1CE_H$		ESFR-b	$E7_H$		
$F1D0_H$		ESFR-b	$E8_H$		
$F1D2_H$		ESFR-b	$E9_H$		
$F1D4_H$		ESFR-b	EA_H		
$F1D6_H$		ESFR-b	EB_H		
$F1D8_H$		ESFR-b	EC_H		
$F1DA_H$		ESFR-b	ED_H		
$F1DC_H$		ESFR-b	EE_H		
$F1DE_H$		ESFR-b	EF_H		
$F1E0_H$	reserved			reserved - do not use	
$F1E2_H$	reserved			reserved - do not use	
$F1E4_H$	reserved			reserved - do not use	
$F1E6_H$	reserved			reserved - do not use	
$F1E8_H$	reserved			reserved - do not use	
$F1EA_H$	reserved			reserved - do not use	
$F1EC_H$	reserved			reserved - do not use	
$F1EE_H$	reserved			reserved - do not use	
$F1F0_H$	reserved			reserved - do not use	
$F1F2_H$	reserved			reserved - do not use	
$F1F4_H$	reserved			reserved - do not use	
$F1F6_H$	reserved			reserved - do not use	
$F1F8_H$	reserved			reserved - do not use	
$F1FA_H$	reserved			reserved - do not use	
$F1FC_H$	reserved			reserved - do not use	
$F1FE_H$	reserved			reserved - do not use	
$FE00_H$	DPP0	SFR	00_H	CPU Data Page Pointer 0 Register (10 bits)	0000_H
$FE02_H$	DPP1	SFR	01_H	CPU Data Page Pointer 1 Register (10 bits)	0001_H

Table 4-2 SFR/ESFR Table (ordered by physical address) (cont'd)

Physical Address	Name	Type[1]	8-bit Addr[2]	Description	Reset Value[3]
$FE04_H$	DPP2	SFR	02_H	CPU Data Page Pointer 2 Register (10 bits)	0002_H
$FE06_H$	DPP3	SFR	03_H	CPU Data Page Pointer 3 Register (10 bits)	0003_H
$FE08_H$	CSP	SFR	04_H	CPU Code Segment Pointer Register (8 bits)	0000_H
$FE0A_H$	reserved	SFR	05_H	reserved - do not use	0000_H
$FE0C_H$	MDH	SFR	06_H	CPU Multiply Divide Register - High Word	0000_H
$FE0E_H$	MDL	SFR	07_H	CPU Multiply Divide Register - Low Word	0000_H
$FE10_H$	CP	SFR	08_H	CPU Context Pointer Register	$FC00_H$
$FE12_H$	SP	SFR	09_H	CPU System Stack Pointer Register	$FC00_H$
$FE14_H$	STKOV	SFR	$0A_H$	CPU Stack Overflow Pointer Register	$FA00_H$
$FE16_H$	STKUN	SFR	$0B_H$	CPU Stack Underflow Pointer Register	$FC00_H$
$FE18_H$	ADDRSEL1	SFR	$0C_H$	Address Select Register 1	0000_H
$FE1A_H$	ADDRSEL2	SFR	$0D_H$	Address Select Register 2	0000_H
$FE1C_H$	ADDRSEL3	SFR	$0E_H$	Address Select Register 3	0000_H
$FE1E_H$	ADDRSEL4	SFR	$0F_H$	Address Select Register 4	0000_H
$FE20_H$		SFR	10_H		
$FE22_H$		SFR	11_H		
$FE24_H$		SFR	12_H		
$FE26_H$		SFR	13_H		
$FE28_H$		SFR	14_H		
$FE2A_H$		SFR	15_H		
$FE2C_H$		SFR	16_H		
$FE2E_H$		SFR	17_H		
$FE30_H$		SFR	18_H		

Table 4-2 SFR/ESFR Table (ordered by physical address) (cont'd)

Physical Address	Name	Type[1]	8-bit Addr[2]	Description	Reset Value[3]
$FE32_H$		SFR	19_H		
$FE34_H$		SFR	$1A_H$		
$FE36_H$		SFR	$1B_H$		
$FE38_H$		SFR	$1C_H$		
$FE3A_H$		SFR	$1D_H$		
$FE3C_H$		SFR	$1E_H$		
$FE3E_H$		SFR	$1F_H$		
$FE40_H$	T2	SFR	20_H	GPT Timer 2 Register	0000_H
$FE42_H$	T3	SFR	21_H	GPT Timer 3 Register	0000_H
$FE44_H$	T4	SFR	22_H	GPT Timer 4 Register	0000_H
$FE46_H$	T5	SFR	23_H	GPT Timer 5 Register	0000_H
$FE48_H$	T6	SFR	24_H	GPT Timer 6 Register	0000_H
$FE4A_H$	CAPREL	SFR	25_H	GPT Capture/Reload Register	0000_H
$FE4C_H$	GPTIPISEL	SFR	26_H	GPT Port Input Selection Register	0000_H
$FE4E_H$		SFR	27_H		
$FE50_H$		SFR	28_H		
$FE52_H$		SFR	29_H		
$FE54_H$		SFR	$2A_H$		
$FE56_H$		SFR	$2B_H$		
$FE58_H$		SFR	$2C_H$		
$FE5A_H$		SFR	$2D_H$		
$FE5C_H$	reserved	SFR	$2E_H$	reserved - do not use	
$FE5E_H$	reserved	SFR	$2F_H$	reserved - do not use	
$FE60_H$		SFR	30_H		
$FE62_H$		SFR	31_H		
$FE64_H$		SFR	32_H		
$FE66_H$		SFR	33_H		
$FE68_H$		SFR	34_H		

Table 4-2 SFR/ESFR Table (ordered by physical address) (cont'd)

Physical Address	Name	Type[1]	8-bit Addr[2]	Description	Reset Value[3]
$FE6A_H$		SFR	35_H		
$FE6C_H$		SFR	36_H		
$FE6E_H$		SFR	37_H		
$FE70_H$		SFR	38_H		
$FE72_H$		SFR	39_H		
$FE74_H$		SFR	$3A_H$		
$FE76_H$		SFR	$3B_H$		
$FE78_H$		SFR	$3C_H$		
$FE7A_H$		SFR	$3D_H$		
$FE7C_H$		SFR	$3E_H$		
$FE7E_H$		SFR	$3F_H$		
$FE80_H$		SFR	40_H		
$FE82_H$		SFR	41_H		
$FE84_H$		SFR	42_H		
$FE86_H$		SFR	43_H		
$FE88_H$		SFR	44_H		
$FE8A_H$		SFR	44_H		
$FE8C_H$		SFR	46_H		
$FE8E_H$		SFR	47_H		
$FE90_H$		SFR	48_H		
$FE92_H$		SFR	49_H		
$FE94_H$		SFR	$4A_H$		
$FE96_H$		SFR	$4B_H$		
$FE98_H$		SFR	$4C_H$		
$FE9A_H$		SFR	$4D_H$		
$FE9C_H$		SFR	$4E_H$		
$FE9E_H$		SFR	$4F_H$		
$FEA0_H$		SFR	50_H		
$FEA2_H$		SFR	51_H		

Table 4-2 SFR/ESFR Table (ordered by physical address) (cont'd)

Physical Address	Name	Type[1]	8-bit Addr[2]	Description	Reset Value[3]
$FEA4_H$		SFR	52_H		
$FEA6_H$		SFR	53_H		
$FEA8_H$		SFR	54_H		
$FEAA_H$		SFR	55_H		
$FEAC_H$		SFR	56_H		
$FEAE_H$	WDT	SFR	57_H	Watchdog Timer Register (RO)	0000_H
$FEB0_H$	S0TBUF	SFR	58_H	Serial Channel 0 Transmit Buffer Register (WO)	0000_H
$FEB2_H$	S0RBUF	SFR	59_H	Serial Channel 0 Receive Buffer Register (RO)	0000_H
$FEB4_H$	S0BG	SFR	$5A_H$	Serial Channel 0 Baud Rate Generator Reload Register	0000_H
$FEB6_H$	FDV	SFR	$5B_H$	Fractional Divider Register	0000_H
$FEB8_H$	PECSN12	SFR	$5C_H$	PEC Segment No. Register	0000_H
$FEBA_H$	PECSN13	SFR	$5D_H$	PEC Segment No. Register	0000_H
$FEBC_H$	PECSN14	SFR	$5E_H$	PEC Segment No. Register	0000_H
$FEBE_H$	PECSN15	SFR	$5F_H$	PEC Segment No. Register	0000_H
$FEC0_H$	PECC0	SFR	60_H	PEC Channel 0 Control Register	0000_H
$FEC2_H$	PECC1	SFR	61_H	PEC Channel 1 Control Register	0000_H
$FEC4_H$	PECC2	SFR	62_H	PEC Channel 2 Control Register	0000_H
$FEC6_H$	PECC3	SFR	63_H	PEC Channel 3 Control Register	0000_H
$FEC8_H$	PECC4	SFR	64_H	PEC Channel 4 Control Register	0000_H
$FECA_H$	PECC5	SFR	65_H	PEC Channel 5 Control Register	0000_H
$FECC_H$	PECC6	SFR	66_H	PEC Channel 6 Control Register	0000_H
$FECE_H$	PECC7	SFR	67_H	PEC Channel 7 Control Register	0000_H
$FED0_H$	PECSN0	SFR	68_H	PEC Segment No. Register	0000_H
$FED2_H$	PECSN1	SFR	69_H	PEC Segment No. Register	0000_H
$FED4_H$	PECSN2	SFR	$6A_H$	PEC Segment No. Register	0000_H
$FED6_H$	PECSN3	SFR	$6B_H$	PEC Segment No. Register	0000_H

Table 4-2 SFR/ESFR Table (ordered by physical address) (cont'd)

Physical Address	Name	Type[1]	8-bit Addr[2]	Description	Reset Value[3]
$FED8_H$	PECSN4	SFR	$6C_H$	PEC Segment No. Register	0000_H
$FEDA_H$	PECSN5	SFR	$6D_H$	PEC Segment No. Register	0000_H
$FEDC_H$	PECSN6	SFR	$6E_H$	PEC Segment No. Register	0000_H
$FEDE_H$	PECSN7	SFR	$6F_H$	PEC Segment No. Register	0000_H
$FEE0_H$	PECSN8	SFR	70_H	PEC Segment No. Register	0000_H
$FEE2_H$	PECSN9	SFR	71_H	PEC Segment No. Register	0000_H
$FEE4_H$	PECSN10	SFR	72_H	PEC Segment No. Register	0000_H
$FEE6_H$	PECSN11	SFR	73_H	PEC Segment No. Register	0000_H
$FEE8_H$	PECC8	SFR	74_H	PEC Channel 8 Control Register	0000_H
$FEEA_H$	PECC9	SFR	75_H	PEC Channel 9 Control Register	0000_H
$FEEC_H$	PECC10	SFR	76_H	PEC Channel 10 Control Register	0000_H
$FEEE_H$	PECC11	SFR	77_H	PEC Channel 11 Control Register	0000_H
$FEF0_H$	PECXC0	SFR	78_H	PEC Channel 0 Extended Control Register	0000_H
$FEF2_H$	PECXC2	SFR	79_H	PEC Channel 2 Extended Control Register	0000_H
$FEF4_H$	reserved	SFR	$7A_H$	reserved - do not use	
$FEF6_H$	reserved	SFR	$7B_H$	reserved - do not use	
$FEF8_H$	PECC12	SFR	$7C_H$	PEC Channel 12 Control Register	0000_H
$FEFA_H$	PECC13	SFR	$7D_H$	PEC Channel 13 Control Register	0000_H
$FEFC_H$	PECC14	SFR	$7E_H$	PEC Channel 14 Control Register	0000_H
$FEFE_H$	PECC15	SFR	$7F_H$	PEC Channel 15 Control Register	0000_H
$FF00_H$	P0L	SFR-b	80_H	Port 0 Low Register (Lower half)	00_H
$FF02_H$	P0H	SFR-b	81_H	Port 0 High Register (Upper half)	00_H
$FF04_H$	P1L	SFR-b	82_H	Port 1 Low Register (Lower half)	00_H
$FF06_H$	P1H	SFR-b	83_H	Port 1 High Register (Upper half)	00_H
$FF08_H$	reserved	SFR-b	84_H	reserved - do not use	
$FF0A_H$	reserved	SFR-b	85_H	reserved - do not use	
$FF0C_H$	BUSCON0	SFR-b	86_H	Bus Configuration Register 0	0000_H

Table 4-2 SFR/ESFR Table (ordered by physical address) (cont'd)

Physical Address	Name	Type[1]	8-bit Addr[2]	Description	Reset Value[3]
$FF0E_H$	MDC	SFR-b	87_H	CPU Multiply Divide Control Register	0000_H
$FF10_H$	PSW	SFR-b	88_H	CPU Program Status Word	0000_H
$FF12_H$	SYSCON	SFR-b	89_H	CPU System Configuration Register	$xxxx_H$
$FF14_H$	BUSCON1	SFR-b	$8A_H$	Bus Configuration Register 1	0000_H
$FF16_H$	BUSCON2	SFR-b	$8B_H$	Bus Configuration Register 2	0000_H
$FF18_H$	BUSCON3	SFR-b	$8C_H$	Bus Configuration Register 3	0000_H
$FF1A_H$	BUSCON4	SFR-b	$8D_H$	Bus Configuration Register 4	0000_H
$FF1C_H$	ZEROS	SFR-b	$8E_H$	Constant Value 0sRegister'	0000_H
$FF1E_H$	ONES	SFR-b	$8F_H$	Constant Value 1sRegister'	$FFFF_H$
$FF20_H$		SFR-b	90_H		
$FF22_H$		SFR-b	91_H		
$FF24_H$		SFR-b	92_H		
$FF26_H$		SFR-b	93_H		
$FF28_H$		SFR-b	94_H		
$FF2A_H$		SFR-b	95_H		
$FF2C_H$		SFR-b	96_H		
$FF2E_H$		SFR-b	97_H		
$FF30_H$		SFR-b	98_H		
$FF32_H$		SFR-b	99_H		
$FF34_H$		SFR-b	$9A_H$		
$FF36_H$		SFR-b	$9B_H$		
$FF38_H$		SFR-b	$9C_H$		
$FF3A_H$		SFR-b	$9D_H$		
$FF3C_H$		SFR-b	$9E_H$		
$FF3E_H$		SFR-b	$9F_H$		
$FF40_H$	T2CON	SFR-b	$A0_H$	GPT Timer 2 Control Register	0000_H
$FF42_H$	T3CON	SFR-b	$A1_H$	GPT Timer 3 Control Register	0000_H

Table 4-2 SFR/ESFR Table (ordered by physical address) (cont'd)

Physical Address	Name	Type[1]	8-bit Addr[2]	Description	Reset Value[3]
$FF44_H$	T4CON	SFR-b	$A2_H$	GPT Timer 4 Control Register	0000_H
$FF46_H$	T5CON	SFR-b	$A3_H$	GPT Timer 5 Control Register	0000_H
$FF48_H$	T6CON	SFR-b	$A4_H$	GPT Timer 6 Control Register	0000_H
$FF4A_H$		SFR-b	$A5_H$		
$FF4C_H$		SFR-b	$A6_H$		
$FF4E_H$		SFR-b	$A7_H$		
$FF50_H$		SFR-b	$A8_H$		
$FF52_H$		SFR-b	$A9_H$		
$FF54_H$		SFR-b	AA_H		
$FF56_H$		SFR-b	AB_H		
$FF58_H$		SFR-b	AC_H		
$FF5A_H$		SFR-b	AD_H		
$FF5C_H$		SFR-b	AE_H		
$FF5E_H$		SFR-b	AF_H		
$FF60_H$	T2IC	SFR-b	$B0_H$	GPT12E Timer 2 Interrupt Control Register	0000_H
$FF62_H$	T3IC	SFR-b	$B1_H$	GPT12E Timer 3 Interrupt Control Register	0000_H
$FF64_H$	T4IC	SFR-b	$B2_H$	GPT12E Timer 4 Interrupt Control Register	0000_H
$FF66_H$	T5IC	SFR-b	$B3_H$	GPT12E Timer 5 Interrupt Control Register	0000_H
$FF68_H$	T6IC	SFR-b	$B4_H$	GPT12E Timer 6 Interrupt Control Register	0000_H
$FF6A_H$	CRIC	SFR-b	$B5_H$	GPT12E Capture/Reload Interrupt Control Register	0000_H
$FF6C_H$	S0TIC	SFR-b	$B6_H$	ASC0 Transmit Interrupt Control Register	0000_H
$FF6E_H$	S0RIC	SFR-b	$B7_H$	ASC0 Receive Interrupt Control Register	0000_H

Table 4-2 SFR/ESFR Table (ordered by physical address) (cont'd)

Physical Address	Name	Type[1]	8-bit Addr[2]	Description	Reset Value[3]
$FF70_H$	S0EIC	SFR-b	$B8_H$	ASC0 Error Interrupt Control Register	0000_H
$FF72_H$	SSC0TIC	SFR-b	$B9_H$	SSC0 Transmit Interrupt Control Register	0000_H
$FF74_H$	SSC0RIC	SFR-b	BA_H	SSC0 Receive Interrupt Control Register	0000_H
$FF76_H$	SSC0EIC	SFR-b	BB_H	SSC0 Error Interrupt Control Register	0000_H
$FF78_H$	IRQ16IC	SFR-b	BC_H	IRQ16 Interrupt Control Register	0000_H
$FF7A_H$	IRQ17IC	SFR-b	BD_H	IRQ17 Interrupt Control Register	0000_H
$FF7C_H$	IRQ18IC	SFR-b	BE_H	IRQ18 Interrupt Control Register	0000_H
$FF7E_H$	IRQ19IC	SFR-b	BF_H	IRQ19 Interrupt Control Register	0000_H
$FF80_H$	IRQ20IC	SFR-b	$C0_H$	IRQ20 Interrupt Control Register	0000_H
$FF82_H$	IRQ21IC	SFR-b	$C1_H$	IRQ21 Interrupt Control Register	0000_H
$FF84_H$	IRQ22IC	SFR-b	$C2_H$	IRQ22 Interrupt Control Register	0000_H
$FF86_H$	IRQ23IC	SFR-b	$C3_H$	IRQ23 Interrupt Control Register	0000_H
$FF88_H$	IRQ24IC	SFR-b	$C4_H$	IRQ24 Interrupt Control Register	0000_H
$FF8A_H$	IRQ25IC	SFR-b	$C5_H$	IRQ25 Interrupt Control Register	0000_H
$FF8C_H$	IRQ26IC	SFR-b	$C6_H$	IRQ26 Interrupt Control Register	0000_H
$FF8E_H$	IRQ27IC	SFR-b	$C7_H$	IRQ27 Interrupt Control Register	0000_H
$FF90_H$	IRQ28IC	SFR-b	$C8_H$	IRQ28 Interrupt Control Register	0000_H
$FF92_H$	IRQ29IC	SFR-b	$C9_H$	IRQ29 Interrupt Control Register	0000_H
$FF94_H$	IRQ30IC	SFR-b	CA_H	IRQ30 Interrupt Control Register	0000_H
$FF96_H$	IRQ31IC	SFR-b	CB_H	IRQ31 Interrupt Control Register	0000_H
$FF98_H$	IRQ34IC	SFR-b	CC_H	IRQ34 Interrupt Control Register	0000_H
$FF9A_H$	IRQ35IC	SFR-b	CD_H	IRQ35 Interrupt Control Register	0000_H
$FF9C_H$	IRQ32IC	SFR-b	CE_H	IRQ32 Interrupt Control Register	0000_H
$FF9E_H$	IRQ33IC	SFR-b	CF_H	IRQ33 Interrupt Control Register	0000_H
$FFA0_H$		SFR-b	$D0_H$		
$FFA2_H$		SFR-b	$D1_H$		

Table 4-2 SFR/ESFR Table (ordered by physical address) (cont'd)

Physical Address	Name	Type[1]	8-bit Addr[2]	Description	Reset Value[3]
$FFA4_H$		SFR-b	$D2_H$		
$FFA6_H$		SFR-b	$D3_H$		
$FFA8_H$	PECISNC	SFR-b	$D4_H$	PEC Interrupt Subnode Control Register	0000_H
$FFAA_H$		SFR-b	$D5_H$		
$FFAC_H$	TFR	SFR-b	$D6_H$	Trap Flag Register	0000_H
$FFAE_H$	WDTCON	SFR-b	$D7_H$	Watchdog Timer Control Register	$008x_H$
$FFB0_H$	S0CON	SFR-b	$D8_H$	Serial Channel 0 Control Register	0000_H
$FFB2_H$	SSC0CON	SFR-b	$D9_H$	SSC0 Control Register	0000_H
$FFB4_H$		SFR-b	DA_H		
$FFB6_H$		SFR-b	DB_H		
$FFB8_H$		SFR-b	DC_H		
$FFBA_H$	PECXISNC	SFR-b	DD_H	PEC Extended Interrupt Subnode Control Register	0000_H
$FFBC_H$		SFR-b	DE_H		
$FFBE_H$		SFR-b	DF_H		
$FFC0_H$		SFR-b	$E0_H$		
$FFC2_H$		SFR-b	$E1_H$		
$FFC4_H$		SFR-b	$E2_H$		
$FFC6_H$		SFR-b	$E3_H$		
$FFC8_H$	P4	SFR-b	$E4_H$	Port 4 Register (8 bits)	00_H
$FFCA_H$	DP4	SFR-b	$E5_H$	Port 4 Direction Control Register	00_H
$FFCC_H$	P6	SFR-b	$E6_H$	Port 6 Register (8 bits)	00_H
$FFCE_H$	DP6	SFR-b	$E7_H$	Port 6 Direction Control Register	00_H
$FFD0_H$		SFR-b	$E8_H$		
$FFD2_H$		SFR-b	$E9_H$		
$FFD4_H$		SFR-b	EA_H		
$FFD6_H$		SFR-b	EB_H		
$FFD8_H$		SFR-b	EC_H		

Table 4-2 SFR/ESFR Table (ordered by physical address) (cont'd)

Physical Address	Name	Type[1]	8-bit Addr[2]	Description	Reset Value[3]
$FFDA_H$	reserved	SFR-b	ED_H	reserved - do not use	
$FFDC_H$	reserved	SFR-b	EE_H	reserved - do not use	
$FFDE_H$	reserved	SFR-b	EF_H	reserved - do not use	
$FFE0_H$	reserved			reserved - do not use	
$FFE2_H$	ASC0ID			ASC0 Identification Register	$44xx_H$
$FFE4_H$	SSC0ID			SSC0 Identification Register	$45xx_H$
$FFE6_H$	GPTID			GPT Identification Register	$58xx_H$
$FFE8_H$	reserved			reserved - do not use	0000_H
$FFEA_H$	reserved			reserved - do not use	0000_H
$FFEC_H$	reserved			reserved - do not use	0000_H
$FFEE_H$	reserved			reserved - do not use	0000_H
$FFF0_H$	reserved			reserved - do not use	0000_H
$FFF2_H$	reserved			reserved - do not use	0000_H
$FFF4_H$	reserved			reserved - do not use	0000_H
$FFF6_H$	reserved			reserved - do not use	0000_H
$FFF8_H$	reserved			reserved - do not use	0000_H
$FFFA_H$	reserved			reserved - do not use	0000_H
$FFFC_H$	reserved			reserved - do not use	0000_H
$FFFE_H$	reserved			reserved - do not use	0000_H

1) The PDBUS+ chip select depends on the register type. Chip select "pd_cs_esfr" is used for register types "ESFR" and "ESFR-b", whereas chip select "pd_cs_sfr" valid for register types "SFR" and "SFR-b"

2) This address is identical to PDBUS+ address A[8:1]. However, for address ranges $F1E0_H$ to $F1FF_H$ and $FFE0_H$ to $FFFF_H$ there is no 8-bit address, but a PDBUS+ address.

3) **NOTE:** Reserved addresses are always read as $FFFF_H$, except another reset value is explicitly documented in this column. However, for enabling future enhancements without any compatibility problems, this addresses should neither be written nor be used as read value by the software.

The following table lists all SFRs/ESFRs which are implemented in the C166S V1 SubS R1 ordered by their name.

Table 4-3 SFR/ESFR Table (ordered by name)

Name	Physical Address	Type[1)]	8-bit Addr[2)]	Description	Reset Value
ADDRSEL1	$FE18_H$	SFR	$0C_H$	Address Select Register 1	0000_H
ADDRSEL2	$FE1A_H$	SFR	$0D_H$	Address Select Register 2	0000_H
ADDRSEL3	$FE1C_H$	SFR	$0E_H$	Address Select Register 3	0000_H
ADDRSEL4	$FE1E_H$	SFR	$0F_H$	Address Select Register 4	0000_H
ASC0ID	$FFE2_H$			ASC0 Identification Register	$44xx_H$
ASC0PISEL	$F1B6_H$	SFR-b	DB_H	ASC0 Port Input Selection Register	0000_H
BUSCON0	$FF0C_H$	SFR-b	86_H	Bus Configuration Register 0	0000_H
BUSCON1	$FF14_H$	SFR-b	$8A_H$	Bus Configuration Register 1	0000_H
BUSCON2	$FF16_H$	SFR-b	$8B_H$	Bus Configuration Register 2	0000_H
BUSCON3	$FF18_H$	SFR-b	$8C_H$	Bus Configuration Register 3	0000_H
BUSCON4	$FF1A_H$	SFR-b	$8D_H$	Bus Configuration Register 4	0000_H
CAPREL	$FE4A_H$	SFR	25_H	GPT Capture/Reload Register	0000_H
COMDATA	$F068_H$	ESFR	34_H	Cerberus Communication Mode Register	0000_H
CP	$FE10_H$	SFR	08_H	CPU Context Pointer Register	$FC00_H$
CPUID	$F00C_H$	ESFR	06_H	CPU Identification Register	0410_H
CRIC	$FF6A_H$	SFR-b	$B5_H$	GPT12E Capture/Reload Interrupt Control Register	0000_H
CSP	$FE08_H$	SFR	04_H	CPU Code Segment Pointer Register (8 bits)	0000_H
DBGSR	$F0FC_H$	ESFR	$7E_H$	Debug status register	0000_H
DCMPDP	$F0EE_H$	ESFR	77_H	Data Programming Register for DCMPx	0000_H
DCMPSP	$F0EC_H$	ESFR	76_H	Select and Programming Register for DCMPx	0000_H
DEXEVT	$F0F2_H$	ESFR	79_H	Specifies action if external break pin is asserted	0000_H

Table 4-3 SFR/ESFR Table (ordered by name) (cont'd)

Name	Physical Address	Type[1)]	8-bit Addr[2)]	Description	Reset Value
DIP	$F0F8_H$	ESFR	$7C_H$	Instruction pointer register	0000_H
DIPX	$F0FA_H$	ESFR	$7D_H$	Instruction pointer register extension	3000_H
DP0H	$F102_H$	ESFR-b	81_H	P0H Direction Control Register	00_H
DP0L	$F100_H$	ESFR-b	80_H	P0L Direction Control Register	00_H
DP1H	$F106_H$	ESFR-b	83_H	P1H Direction Control Register	00_H
DP1L	$F104_H$	ESFR-b	82_H	P1L Direction Control Register	00_H
DP4	$FFCA_H$	SFR-b	$E5_H$	Port 4 Direction Control Register	00_H
DP6	$FFCE_H$	SFR-b	$E7_H$	Port 6 Direction Control Register	00_H
DPP0	$FE00_H$	SFR	00_H	CPU Data Page Pointer 0 Register (10 bits)	0000_H
DPP1	$FE02_H$	SFR	01_H	CPU Data Page Pointer 1 Register (10 bits)	0001_H
DPP2	$FE04_H$	SFR	02_H	CPU Data Page Pointer 2 Register (10 bits)	0002_H
DPP3	$FE06_H$	SFR	03_H	CPU Data Page Pointer 3 Register (10 bits)	0003_H
DSWEVT	$F0F4_H$	ESFR	$7A_H$	Specifies action if DEBUG instruction is executed	0000_H
DTIDR	$F0D8_H$	ESFR	$6C_H$	Task ID register	0000_H
DTREVT	$F0F0_H$	ESFR	78_H	Specifies hardware triggers and action	0000_H
EOPIC	$F180_H$	ESFR-b	$C0_H$	End of PEC Transfer Interrupt Control Register	0000_H
FDV	$FEB6_H$	SFR	5B	Fractional Divider Register	0000_H
GPTID	$FFE6_H$			GPT Identification Register	$58xx_H$
GPTPISEL	$FE4C_H$	SFR	26_H	GPT Port Input Selection Register	0000_H
IOSR	$F06C_H$	ESFR	36_H	Cerberus status register	0000_H
IRQ15IC	$F17E_H$	ESFR-b	BF_H	IRQ15 Interrupt Control Register	0000_H
IRQ16IC	$FF78_H$	SFR-b	BC_H	IRQ16 Interrupt Control Register	0000_H

Table 4-3 SFR/ESFR Table (ordered by name) (cont'd)

Name	Physical Address	Type[1)]	8-bit Addr[2)]	Description	Reset Value
IRQ17IC	$FF7A_H$	SFR-b	BD_H	IRQ17 Interrupt Control Register	0000_H
IRQ18IC	$FF7C_H$	SFR-b	BE_H	IRQ18 Interrupt Control Register	0000_H
IRQ19IC	$FF7E_H$	SFR-b	BF_H	IRQ19 Interrupt Control Register	0000_H
IRQ20IC	$FF80_H$	SFR-b	$C0_H$	IRQ20 Interrupt Control Register	0000_H
IRQ21IC	$FF82_H$	SFR-b	$C1_H$	IRQ21 Interrupt Control Register	0000_H
IRQ22IC	$FF84_H$	SFR-b	$C2_H$	IRQ22 Interrupt Control Register	0000_H
IRQ23IC	$FF86_H$	SFR-b	$C3_H$	IRQ23 Interrupt Control Register	0000_H
IRQ24IC	$FF88_H$	SFR-b	$C4_H$	IRQ24 Interrupt Control Register	0000_H
IRQ25IC	$FF8A_H$	SFR-b	$C5_H$	IRQ25 Interrupt Control Register	0000_H
IRQ26IC	$FF8C_H$	SFR-b	$C6_H$	IRQ26 Interrupt Control Register	0000_H
IRQ27IC	$FF8E_H$	SFR-b	$C7_H$	IRQ27 Interrupt Control Register	0000_H
IRQ28IC	$FF90_H$	SFR-b	$C8_H$	IRQ28 Interrupt Control Register	0000_H
IRQ29IC	$FF92_H$	SFR-b	$C9_H$	IRQ29 Interrupt Control Register	0000_H
IRQ30IC	$FF94_H$	SFR-b	CA_H	IRQ30 Interrupt Control Register	0000_H
IRQ31IC	$FF96_H$	SFR-b	CB_H	IRQ31 Interrupt Control Register	0000_H
IRQ32IC	$FF9C_H$	SFR-b	CE_H	IRQ32 Interrupt Control Register	0000_H
IRQ33IC	$FF9E_H$	SFR-b	CF_H	IRQ33 Interrupt Control Register	0000_H
IRQ34IC	$FF98_H$	SFR-b	CC_H	IRQ34 Interrupt Control Register	0000_H
IRQ35IC	$FF9A_H$	SFR-b	CD_H	IRQ35 Interrupt Control Register	0000_H
IRQ36IC	$F186_H$	ESFR-b	$C3_H$	IRQ36 Interrupt Control Register	0000_H
IRQ37IC	$F18E_H$	ESFR-b	$C7_H$	IRQ37 Interrupt Control Register	0000_H
IRQ38IC	$F196_H$	ESFR-b	CB_H	IRQ38 Interrupt Control Register	0000_H
IRQ39IC	$F19E_H$	ESFR-b	CF_H	IRQ39 Interrupt Control Register	0000_H
IRQ40IC	$F17A_H$	ESFR-b	BD_H	IRQ40 Interrupt Control Register	0000_H
IRQ41IC	$F17C_H$	ESFR-b	BE_H	IRQ41 Interrupt Control Register	0000_H
IRQ42IC	$F182_H$	ESFR-b	$C1_H$	IRQ42 Interrupt Control Register	0000_H
IRQ43IC	$F18A_H$	ESFR-b	$C5_H$	IRQ43 Interrupt Control Register	0000_H
IRQ44IC	$F192_H$	ESFR-b	$C9_H$	IRQ44 Interrupt Control Register	0000_H
IRQ45IC	$F190_H$	ESFR-b	$C8_H$	IRQ45 Interrupt Control Register	0000_H

Table 4-3 SFR/ESFR Table (ordered by name) (cont'd)

Name	Physical Address	Type[1)]	8-bit Addr [2)]	Description	Reset Value
IRQ46IC	$F198_H$	ESFR-b	CC_H	IRQ46 Interrupt Control Register	0000_H
IRQ47IC	$F188_H$	ESFR-b	$C4_H$	IRQ47 Interrupt Control Register	0000_H
IRQ48IC	$F160_H$	ESFR-b	$B0_H$	IRQ48 Interrupt Control Register	0000_H
IRQ49IC	$F162_H$	ESFR-b	$B1_H$	IRQ49 Interrupt Control Register	0000_H
IRQ50IC	$F164_H$	ESFR-b	$B2_H$	IRQ50 Interrupt Control Register	0000_H
IRQ51IC	$F166_H$	ESFR-b	$B3_H$	IRQ51 Interrupt Control Register	0000_H
IRQ52IC	$F168_H$	ESFR-b	$B4_H$	IRQ52 Interrupt Control Register	0000_H
IRQ53IC	$F16A_H$	ESFR-b	$B5_H$	IRQ53 Interrupt Control Register	0000_H
IRQ54IC	$F16C_H$	ESFR-b	$B6_H$	IRQ54 Interrupt Control Register	0000_H
IRQ55IC	$F16E_H$	ESFR-b	$B7_H$	IRQ55 Interrupt Control Register	0000_H
IRQ56IC	$F170_H$	ESFR-b	$B8_H$	IRQ56 Interrupt Control Register	0000_H
IRQ57IC	$F172_H$	ESFR-b	$B9_H$	IRQ57 Interrupt Control Register	0000_H
IRQ58IC	$F174_H$	ESFR-b	BA_H	IRQ58 Interrupt Control Register	0000_H
IRQ59IC	$F176_H$	ESFR-b	BB_H	IRQ59 Interrupt Control Register	0000_H
IRQ60IC	$F178_H$	ESFR-b	BC_H	IRQ60 Interrupt Control Register	0000_H
IRQ61IC	$F184_H$	ESFR-b	$C2_H$	IRQ61 Interrupt Control Register	0000_H
IRQ62IC	$F18C_H$	ESFR-b	$C6_H$	IRQ62 Interrupt Control Register	0000_H
IRQ63IC	$F194_H$	ESFR-b	CA_H	IRQ63 Interrupt Control Register	0000_H
IRQ64IC	$F120_H$	ESFR-b	90_H	IRQ64 Interrupt Control Register	0000_H
IRQ65IC	$F122_H$	ESFR-b	91_H	IRQ65 Interrupt Control Register	0000_H
IRQ66IC	$F124_H$	ESFR-b	92_H	IRQ66 Interrupt Control Register	0000_H
IRQ67IC	$F126_H$	ESFR-b	93_H	IRQ67 Interrupt Control Register	0000_H
IRQ68IC	$F128_H$	ESFR-b	94_H	IRQ68 Interrupt Control Register	0000_H
IRQ69IC	$F12A_H$	ESFR-b	95_H	IRQ69 Interrupt Control Register	0000_H
IRQ70IC	$F12C_H$	ESFR-b	96_H	IRQ70 Interrupt Control Register	0000_H
IRQ71IC	$F12E_H$	ESFR-b	97_H	IRQ71 Interrupt Control Register	0000_H
IRQ72IC	$F130_H$	ESFR-b	98_H	IRQ72 Interrupt Control Register	0000_H
IRQ73IC	$F132_H$	ESFR-b	99_H	IRQ73 Interrupt Control Register	0000_H
IRQ74IC	$F134_H$	ESFR-b	$9A_H$	IRQ74 Interrupt Control Register	0000_H

Table 4-3 SFR/ESFR Table (ordered by name) (cont'd)

Name	Physical Address	Type[1)]	8-bit Addr[2)]	Description	Reset Value
IRQ75IC	$F136_H$	ESFR-b	$9B_H$	IRQ75 Interrupt Control Register	0000_H
IRQ76IC	$F138_H$	ESFR-b	$9C_H$	IRQ76 Interrupt Control Register	0000_H
IRQ77IC	$F13A_H$	ESFR-b	$9D_H$	IRQ77 Interrupt Control Register	0000_H
IRQ78IC	$F13C_H$	ESFR-b	$9E_H$	IRQ78 Interrupt Control Register	0000_H
IRQ79IC	$F13E_H$	ESFR-b	$9F_H$	IRQ79 Interrupt Control Register	0000_H
IRQ80IC	$F140_H$	ESFR-b	$A0_H$	IRQ80 Interrupt Control Register	0000_H
IRQ81IC	$F142_H$	ESFR-b	$A1_H$	IRQ81 Interrupt Control Register	0000_H
IRQ82IC	$F144_H$	ESFR-b	$A2_H$	IRQ82 Interrupt Control Register	0000_H
IRQ83IC	$F146_H$	ESFR-b	$A3_H$	IRQ83 Interrupt Control Register	0000_H
IRQ84IC	$F148_H$	ESFR-b	$A4_H$	IRQ84 Interrupt Control Register	0000_H
IRQ85IC	$F14A_H$	ESFR-b	$A5_H$	IRQ85 Interrupt Control Register	0000_H
IRQ86IC	$F14C_H$	ESFR-b	$A6_H$	IRQ86 Interrupt Control Register	0000_H
IRQ87IC	$F14E_H$	ESFR-b	$A7_H$	IRQ87 Interrupt Control Register	0000_H
IRQ88IC	$F150_H$	ESFR-b	$A8_H$	IRQ88 Interrupt Control Register	0000_H
IRQ89IC	$F152_H$	ESFR-b	$A9_H$	IRQ89 Interrupt Control Register	0000_H
IRQ90IC	$F154_H$	ESFR-b	AA_H	IRQ90 Interrupt Control Register	0000_H
IRQ91IC	$F156_H$	ESFR-b	AB_H	IRQ91 Interrupt Control Register	0000_H
IRQ92IC	$F158_H$	ESFR-b	AC_H	IRQ92 Interrupt Control Register	0000_H
IRQ93IC	$F15A_H$	ESFR-b	AD_H	IRQ93 Interrupt Control Register	0000_H
IRQ94IC	$F15C_H$	ESFR-b	AE_H	IRQ94 Interrupt Control Register	0000_H
IRQ95IC	$F15E_H$	ESFR-b	AF_H	IRQ95 Interrupt Control Register	0000_H
IRQ96IC	$F0B8_H$	ESFR	$5C_H$	IRQ96 Interrupt Control Register	0000_H
IRQ97IC	$F0BA_H$	ESFR	$5D_H$	IRQ97 Interrupt Control Register	0000_H
IRQ98IC	$F0BC_H$	ESFR	$5E_H$	IRQ98 Interrupt Control Register	0000_H
IRQ99IC	$F0BE_H$	ESFR	$5F_H$	IRQ99 Interrupt Control Register	0000_H
IRQ100IC	$F0C0_H$	ESFR	60_H	IRQ100 Interrupt Control Register	0000_H
IRQ101IC	$F0C2_H$	ESFR	61_H	IRQ101 Interrupt Control Register	0000_H

Table 4-3 SFR/ESFR Table (ordered by name) (cont'd)

Name	Physical Address	Type[1)]	8-bit Addr[2)]	Description	Reset Value
IRQ102IC	$F0C4_H$	ESFR	62_H	IRQ102 Interrupt Control Register	0000_H
IRQ103IC	$F0C6_H$	ESFR	63_H	IRQ103 Interrupt Control Register	0000_H
IRQ104IC	$F0C8_H$	ESFR	64_H	IRQ104 Interrupt Control Register	0000_H
IRQ105IC	$F0CA_H$	ESFR	65_H	IRQ105 Interrupt Control Register	0000_H
IRQ106IC	$F0CC_H$	ESFR	66_H	IRQ106 Interrupt Control Register	0000_H
IRQ107IC	$F0CE_H$	ESFR	67_H	IRQ107 Interrupt Control Register	0000_H
IRQ108IC	$F0D0_H$	ESFR	68_H	IRQ108 Interrupt Control Register	0000_H
IRQ109IC	$F0D2_H$	ESFR	69_H	IRQ109 Interrupt Control Register	0000_H
IRQ110IC	$F0D4_H$	ESFR	$6A_H$	IRQ110 Interrupt Control Register	0000_H
IRQ111IC	$F0D6_H$	ESFR	$6B_H$	IRQ111 Interrupt Control Register	0000_H
MDC	$FF0E_H$	SFR-b	87_H	CPU Multiply Divide Control Register	0000_H
MDH	$FE0C_H$	SFR	06_H	CPU Multiply Divide Register - High Word	0000_H
MDL	$FE0E_H$	SFR	07_H	CPU Multiply Divide Register - Low Word	0000_H
ONES	$FF1E_H$	SFR-b	$8F_H$	Constant Value 1sRegister'	$FFFF_H$
P0H	$FF02_H$	SFR-b	81_H	Port 0 High Register (Upper half)	00_H
P0L	$FF00_H$	SFR-b	80_H	Port 0 Low Register (Lower half)	00_H
P1H	$FF06_H$	SFR-b	83_H	Port 1 High Register (Upper half)	00_H
P1L	$FF04_H$	SFR-b	82_H	Port 1 Low Register (Lower half)	00_H
P4	$FFC8_H$	SFR-b	$E4_H$	Port 4 Register (8 bits)	00_H

Table 4-3 SFR/ESFR Table (ordered by name) (cont'd)

Name	Physical Address	Type[1)]	8-bit Addr[2)]	Description	Reset Value
P6	$FFCC_H$	SFR-b	$E6_H$	Port 6 Register (8 bits)	00_H
PECC0	$FEC0_H$	SFR	60_H	PEC Channel 0 Control Register	0000_H
PECC1	$FEC2_H$	SFR	61_H	PEC Channel 1 Control Register	0000_H
PECC2	$FEC4_H$	SFR	62_H	PEC Channel 2 Control Register	0000_H
PECC3	$FEC6_H$	SFR	63_H	PEC Channel 3 Control Register	0000_H
PECC4	$FEC8_H$	SFR	64_H	PEC Channel 4 Control Register	0000_H
PECC5	$FECA_H$	SFR	65_H	PEC Channel 5 Control Register	0000_H
PECC6	$FECC_H$	SFR	66_H	PEC Channel 6 Control Register	0000_H
PECC7	$FECE_H$	SFR	67_H	PEC Channel 7 Control Register	0000_H
PECC8	$FEE8_H$	SFR	74_H	PEC Channel 8 Control Register	0000_H
PECC9	$FEEA_H$	SFR	75_H	PEC Channel 9 Control Register	0000_H
PECC10	$FEEC_H$	SFR	76_H	PEC Channel 10 Control Register	0000_H
PECC11	$FEEE_H$	SFR	77_H	PEC Channel 11 Control Register	0000_H
PECC12	$FEF8_H$	SFR	$7C_H$	PEC Channel 12 Control Register	0000_H
PECC13	$FEFA_H$	SFR	$7D_H$	PEC Channel 13 Control Register	0000_H
PECC14	$FEFC_H$	SFR	$7E_H$	PEC Channel 14 Control Register	0000_H
PECC15	$FEFE_H$	SFR	$7F_H$	PEC Channel 15 Control Register	0000_H
PECISNC	$FFA8_H$	SFR-b	$D4_H$	PEC Interrupt Subnode Control Register	0000_H
PECSN0	$FED0_H$	SFR	68_H	PEC Segment No. Register	0000_H
PECSN1	$FED2_H$	SFR	69_H	PEC Segment No. Register	0000_H
PECSN2	$FED4_H$	SFR	$6A_H$	PEC Segment No. Register	0000_H
PECSN3	$FED6_H$	SFR	$6B_H$	PEC Segment No. Register	0000_H
PECSN4	$FED8_H$	SFR	$6C_H$	PEC Segment No. Register	0000_H

Table 4-3 SFR/ESFR Table (ordered by name) (cont'd)

Name	Physical Address	Type[1)]	8-bit Addr[2)]	Description	Reset Value
PECSN5	$FEDA_H$	SFR	$6D_H$	PEC Segment No. Register	0000_H
PECSN6	$FEDC_H$	SFR	$6E_H$	PEC Segment No. Register	0000_H
PECSN7	$FEDE_H$	SFR	$6F_H$	PEC Segment No. Register	0000_H
PECSN8	$FEE0_H$	SFR	70_H	PEC Segment No. Register	0000_H
PECSN9	$FEE2_H$	SFR	71_H	PEC Segment No. Register	0000_H
PECSN10	$FEE4_H$	SFR	72_H	PEC Segment No. Register	0000_H
PECSN11	$FEE6_H$	SFR	73_H	PEC Segment No. Register	0000_H
PECSN12	$FEB8_H$	SFR	$5C_H$	PEC Segment No. Register	0000_H
PECSN13	$FEBA_H$	SFR	$5D_H$	PEC Segment No. Register	0000_H
PECSN14	$FEBC_H$	SFR	$5E_H$	PEC Segment No. Register	0000_H
PECSN15	$FEBE_H$	SFR	$5F_H$	PEC Segment No. Register	0000_H
PECXC0	$FEF0_H$	SFR	78_H	PEC Channel 0 Extended Control Register	0000_H
PECXC2	$FEF2_H$	SFR	79_H	PEC Channel 2 Extended Control Register	0000_H
PECXISNC	$FFBA_H$	SFR-b	DD_H	PEC Extended Interrupt Subnode Control Register	0000_H
PSW	$FF10_H$	SFR-b	88_H	CPU Program Status Word	0000_H
RWDATA	$F06A_H$	ESFR	35_H	Cerberus RW Mode Data Register	0000_H
S0BG	$FEB4_H$	SFR	$5A_H$	Serial Channel 0 Baud Rate Generator Reload Register	0000_H
S0CON	$FFB0_H$	SFR-b	$D8_H$	Serial Channel 0 Control Register	0000_H
S0EIC	$FF70_H$	SFR-b	$B8_H$	ASC0 Error Interrupt Control Register	0000_H
S0RBUF	$FEB2_H$	SFR	59_H	Serial Channel 0 Receive Buffer Register (RO)	0000_H
S0RIC	$FF6E_H$	SFR-b	$B7_H$	ASC0 Receive Interrupt Control Register	0000_H

Table 4-3 SFR/ESFR Table (ordered by name) (cont'd)

Name	Physical Address	Type[1)]	8-bit Addr[2)]	Description	Reset Value
S0TBIC	$F19C_H$	ESFR-b	CE_H	ASC0 Transmit Buffer Interrupt Control Register	0000_H
S0TBUF	$FEB0_H$	SFR	58_H	Serial Channel 0 Transmit Buffer Register (WO)	0000_H
S0TIC	$FF6C_H$	SFR-b	$B6_H$	ASC0 Transmit Interrupt Control Register	0000_H
SP	$FE12_H$	SFR	09_H	CPU System Stack Pointer Register	$FC00_H$
SSC0BR	$F0B4_H$	ESFR	$5A_H$	SSC0 Baudrate Register	0000_H
SSC0CON	$FFB2_H$	SFR-b	$D9_H$	SSC0 Control Register	0000_H
SSC0EIC	$FF76_H$	SFR-b	BB_H	SSC0 Error Interrupt Control Register	0000_H
SSC0ID	$FFE4_H$			SSC0 Identification Register	$45xx_H$
SSC0PISEL	$F0B6_H$	ESFR	$5B_H$	SSC0 Port Input Selection Register	0000_H
SSC0RB	$F0B2_H$	ESFR	59_H	SSC0 Receive Buffer (RO)	0000_H
SSC0RIC	$FF74_H$	SFR-b	BA_H	SSC0 Receive Interrupt Control Register	0000_H
SSC0TB	$F0B0_H$	ESFR	58_H	SSC0 Transmit Buffer (WO)	0000_H
SSC0TIC	$FF72_H$	SFR-b	$B9_H$	SSC0 Transmit Interrupt Control Register	0000_H
STKOV	$FE14_H$	SFR	$0A_H$	CPU Stack Overflow Pointer Register	$FA00_H$
STKUN	$FE16_H$	SFR	$0B_H$	CPU Stack Underflow Pointer Register	$FC00_H$
SYSCON	$FF12_H$	SFR-b	89_H	CPU System Configuration Register	$xxxx_H$
T2	$FE40_H$	SFR	20_H	GPT Timer 2 Register	0000_H
T2CON	$FF40_H$	SFR-b	$A0_H$	GPT Timer 2 Control Register	0000_H
T2IC	$FF60_H$	SFR-b	$B0_H$	GPT12E Timer 2 Interrupt Control Register	0000_H

Table 4-3 SFR/ESFR Table (ordered by name) (cont'd)

Name	Physical Address	Type[1)]	8-bit Addr[2)]	Description	Reset Value
T3	$FE42_H$	SFR	21_H	GPT Timer 3 Register	0000_H
T3CON	$FF42_H$	SFR-b	$A1_H$	GPT Timer 3 Control Register	0000_H
T3IC	$FF62_H$	SFR-b	$B1_H$	GPT12E Timer 3 Interrupt Control Register	0000_H
T4	$FE44_H$	SFR	22_H	GPT Timer 4 Register	0000_H
T4CON	$FF44_H$	SFR-b	$A2_H$	GPT Timer 4 Control Register	0000_H
T4IC	$FF64_H$	SFR-b	$B2_H$	GPT12E Timer 4 Interrupt Control Register	0000_H
T5	$FE46_H$	SFR	23_H	GPT Timer 5 Register	0000_H
T5CON	$FF46_H$	SFR-b	$A3_H$	GPT Timer 5 Control Register	0000_H
T5IC	$FF66_H$	SFR-b	$B3_H$	GPT12E Timer 5 Interrupt Control Register	0000_H
T6	$FE48_H$	SFR	24_H	GPT Timer 6 Register	0000_H
T6CON	$FF48_H$	SFR-b	$A4_H$	GPT Timer 6 Control Register	0000_H
T6IC	$FF68_H$	SFR-b	$B4_H$	GPT12E Timer 6 Interrupt Control Register	0000_H
TFR	$FFAC_H$	SFR-b	$D6_H$	Trap Flag Register	0000_H
WDT	$FEAE_H$	SFR	57_H	Watchdog Timer Register (RO)	0000_H
WDTCON	$FFAE_H$	SFR-b	$D7_H$	Watchdog Timer Control Register	$008x_H$
WDTIC	$F19A_H$	ESFR-b	CD_H	Watchdog Timer Interrupt Control Register	0000_H
XADRS1	$F014_H$	ESFR	$0A_H$	XBUS Address Select Register 1	0000_H
XADRS2	$F016_H$	ESFR	$0B_H$	XBUS Address Select Register 2	0000_H
XADRS3	$F018_H$	ESFR	$0C_H$	XBUS Address Select Register 3	0000_H
XADRS4	$F01A_H$	ESFR	$0D_H$	XBUS Address Select Register 4	0000_H
XADRS5	$F01C_H$	ESFR	$0E_H$	XBUS Address Select Register 5	0000_H
XADRS6	$F01E_H$	ESFR	$0F_H$	XBUS Address Select Register 6	0000_H
XBCON1	$F114_H$	ESFR-b	$8A_H$	XBUS Control register 1	0000_H
XBCON2	$F116_H$	ESFR-b	$8B_H$	XBUS Control register 2	0000_H

Table 4-3 SFR/ESFR Table (ordered by name) (cont'd)

Name	Physical Address	Type[1)]	8-bit Addr[2)]	Description	Reset Value
XBCON3	$F118_H$	ESFR-b	$8C_H$	XBUS Control register 3	0000_H
XBCON4	$F11A_H$	ESFR-b	$8D_H$	XBUS Control register 4	0000_H
XBCON5	$F11C_H$	ESFR-b	$8E_H$	XBUS Control register 5	0000_H
XBCON6	$F11E_H$	ESFR-b	$8F_H$	XBUS Control register 6	0000_H
XPERCON	$F024_H$	ESFR	12_H	XBUS Peripheral Control Register	0000_H
ZEROS	$FF1C_H$	SFR-b	$8E_H$	Constant Value 0sRegister'	0000_H

1) The PDBUS+ chip select depends on the register type. Chip select "pd_cs_esfr" is used for register types "ESFR" and "ESFR-b", whereas chip select "pd_cs_sfr" valid for register types "SFR" and "SFR-b"

2) This address is identical to PDBUS+ address A[8:1]. However, for address ranges $F1E0_H$ to $F1FF_H$ and $FFE0_H$ to $FFFF_H$ there is no 8-bit address, but a PDBUS+ address.

4.8 Interrupt Vector Table

The interrupt vector table is an instruction table. For each interrupt a 4 byte range is reserved for instructions. Up to 112 interrupt nodes and 16 trap entries are defined for a C166S V1 SubS R1 based product. The subsystem itself allocates 15 interrupt nodes. All interrupt nodes above them can be used by the product.

The table below lists all possible 112 interrupts and traps sorted by the trap number. The maximum number of interrupt nodes depends on the subsystem configuration. Depending on this configured number of interrupts not all below listed interrupts are available on product level.

Example:

PARAM_IC_NODES = 16

⇒ Only one interrupt source is available on product level (irq_n_i[15]).

PARAM_IC_NODES = 48

⇒ 23 interrupt sources are available on product level (irq_n_i[15...47]).

Table 4-4 Interrupt Vector Table (sorted by trap number)

Signal Name (Interrupt IF)	Source of Interrupt	Interrupt Control Register	Vector Location	Trap No
-	Reset	-	0000_H	00_H
-		-	0004_H	01_H
nmi_trap_n_i[1)]	Non maskable interrupt	-	0008_H	02_H
-		-	$000C_H$	03_H
-	Stack overflow	-	0010_H	04_H
-		-	0014_H	05_H
-	Stack underflow	-	0018_H	06_H
-		-	$001C_H$	07_H
-	Software Break	-	0020_H	08_H
-		-	0024_H	09_H
-	Class B Trap	-	0028_H	$0A_H$
-		-	$002C_H$	$0B_H$
-		-	0030_H	$0C_H$
-		-	0034_H	$0D_H$

Table 4-4 Interrupt Vector Table (cont'd) **(sorted by trap number)**

Signal Name (Interrupt IF)	Source of Interrupt	Interrupt Control Register	Vector Location	Trap No
-		-	0038_H	$0E_H$
-		-	$003C_H$	$0F_H$
irq_i[16]	Product Interrupt Request 16	IRQ16IC	0040_H	10_H
irq_i[17]	Product Interrupt Request 17	IRQ17IC	0044_H	11_H
irq_i[18]	Product Interrupt Request 18	IRQ18IC	0048_H	12_H
irq_i[19]	Product Interrupt Request 19	IRQ19IC	$004C_H$	13_H
irq_i[20]	Product Interrupt Request 20	IRQ20IC	0050_H	14_H
irq_i[21]	Product Interrupt Request 21	IRQ21IC	0054_H	15_H
irq_i[22]	Product Interrupt Request 22	IRQ22IC	0058_H	16_H
irq_i[23]	Product Interrupt Request 23	IRQ23IC	$005C_H$	17_H
irq_i[24]	Product Interrupt Request 24	IRQ24IC	0060_H	18_H
irq_i[25]	Product Interrupt Request 25	IRQ25IC	0064_H	19_H
irq_i[26]	Product Interrupt Request 26	IRQ26IC	0068_H	$1A_H$
irq_i[27]	Product Interrupt Request 27	IRQ27IC	$006C_H$	$1B_H$
irq_i[28]	Product Interrupt Request 28	IRQ28IC	0070_H	$1C_H$
irq_i[29]	Product Interrupt Request 29	IRQ29IC	0074_H	$1D_H$
irq_i[30]	Product Interrupt Request 30	IRQ30IC	0078_H	$1E_H$
irq_i[31]	Product Interrupt Request 31	IRQ31IC	$007C_H$	$1F_H$
irq_i[32]	Product Interrupt Request 32	IRQ32IC	0080_H	20_H
irq_i[33]	Product Interrupt Request 33	IRQ33IC	0084_H	21_H
per_irq_i[2][2)]	GPT12E Timer T2	T2IC	0088_H	22_H
per_irq_i[3][2)]	GPT12E Timer T3	T3IC	$008C_H$	23_H
per_irq_i[4][2)]	GPT12E Timer T4	T4IC	0090_H	24_H
per_irq_i[5][2)]	GPT12E Timer T5	T5IC	0094_H	25_H
per_irq_i[6][2)]	GPT12E Timer T6	T6IC	0098_H	26_H
per_irq_i[7][2)]	GPT12E Capture/Reload	CRIC	$009C_H$	27_H
irq_i[34]	Product Interrupt Request 34	IRQ34IC	$00A0_H$	28_H
irq_i[35]	Product Interrupt Request 35	IRQ35IC	$00A4_H$	29_H
per_irq_i[11][2)]	ASC0 Transmit	S0TIC	$00A8_H$	$2A_H$

Table 4-4 Interrupt Vector Table (cont'd) **(sorted by trap number)**

Signal Name (Interrupt IF)	Source of Interrupt	Interrupt Control Register	Vector Location	Trap No
per_irq_i[12][2)]	ASC0 Receive	S0RIC	$00AC_H$	$2B_H$
per_irq_i[13][2)]	ASC0 Error	S0EIC	$00B0_H$	$2C_H$
per_irq_i[8][2)]	SSC0 Transmit	SSC0TIC	$00B4_H$	$2D_H$
per_irq_i[9][2)]	SSC0 Receive	SSC0RIC	$00B8_H$	$2E_H$
per_irq_i[10][2)]	SSC0 Error	SSC0EIC	$00BC_H$	$2F_H$
irq_i[48]	Product Interrupt Request 48	IRQ48IC	$00C0_H$	30_H
irq_i[49]	Product Interrupt Request 49	IRQ49IC	$00C4_H$	31_H
irq_i[50]	Product Interrupt Request 50	IRQ50IC	$00C8_H$	32_H
irq_i[51]	Product Interrupt Request 51	IRQ51IC	$00CC_H$	33_H
irq_i[52]	Product Interrupt Request 52	IRQ52IC	$00D0_H$	34_H
irq_i[53]	Product Interrupt Request 53	IRQ53IC	$00D4_H$	35_H
irq_i[54]	Product Interrupt Request 54	IRQ54IC	$00D8_H$	36_H
irq_i[55]	Product Interrupt Request 55	IRQ55IC	$00DC_H$	37_H
irq_i[56]	Product Interrupt Request 56	IRQ56IC	$00E0_H$	38_H
irq_i[57]	Product Interrupt Request 57	IRQ57IC	$00E4_H$	39_H
irq_i[58]	Product Interrupt Request 58	IRQ58IC	$00E8_H$	$3A_H$
irq_i[59]	Product Interrupt Request 59	IRQ59IC	$00EC_H$	$3B_H$
irq_i[60]	Product Interrupt Request 60	IRQ60IC	$00F0_H$	$3C_H$
irq_i[40]	Product Interrupt Request 40	IRQ40IC	$00F4_H$	$3D_H$
irq_i[41]	Product Interrupt Request 41	IRQ41IC	$00F8_H$	$3E_H$
irq_i[15]	Product Interrupt Request 15	IRQ15IC	$00FC_H$	$3F_H$
irq_i[36]	Product Interrupt Request 36	IRQ36IC	0100_H	40_H
irq_i[37]	Product Interrupt Request 37	IRQ37IC	0104_H	41_H
irq_i[38]	Product Interrupt Request 38	IRQ38IC	0108_H	42_H
irq_i[39]	Product Interrupt Request 39	IRQ39IC	$010C_H$	43_H
irq_i[61]	Product Interrupt Request 61	IRQ61IC	0110_H	44_H
irq_i[62]	Product Interrupt Request 62	IRQ62IC	0114_H	45_H
irq_i[63]	Product Interrupt Request 63	IRQ63IC	0118_H	46_H
per_irq_i[14][2)]	ASC0 Transmit Buffer	S0TBIC	$011C_H$	47_H

Table 4-4 Interrupt Vector Table (cont'd) **(sorted by trap number)**

Signal Name (Interrupt IF)	Source of Interrupt	Interrupt Control Register	Vector Location	Trap No
irq_i[42]	Product Interrupt Request 42	IRQ42IC	0120_H	48_H
irq_i[43]	Product Interrupt Request 43	IRQ43IC	0124_H	49_H
irq_i[44]	Product Interrupt Request 44	IRQ44IC	0128_H	$4A_H$
per_irq_i[1][2)]	Watchdog Timer	WDTIC	0130_H	$4B_H$
per_irq_i[0][2)]	End of PEC Transfer	EOPIC	$012C_H$	$4C_H$
irq_i[45]	Product Interrupt Request 45	IRQ45IC	0134_H	$4D_H$
irq_i[46]	Product Interrupt Request 46	IRQ46IC	0138_H	$4E_H$
irq_i[47]	Product Interrupt Request 47	IRQ47IC	$013C_H$	$4F_H$
irq_i[64]	Product Interrupt Request 64	IRQ64IC	0140_H	50_H
irq_i[65]	Product Interrupt Request 65	IRQ65IC	0144_H	51_H
irq_i[66]	Product Interrupt Request 66	IRQ66IC	0148_H	52_H
irq_i[67]	Product Interrupt Request 67	IRQ67IC	$014C_H$	53_H
irq_i[68]	Product Interrupt Request 68	IRQ68IC	0150_H	54_H
irq_i[69]	Product Interrupt Request 69	IRQ69IC	0154_H	55_H
irq_i[70]	Product Interrupt Request 70	IRQ70IC	0158_H	56_H
irq_i[71]	Product Interrupt Request 71	IRQ71IC	$015C_H$	57_H
irq_i[72]	Product Interrupt Request 72	IRQ72IC	0160_H	58_H
irq_i[73]	Product Interrupt Request 73	IRQ73IC	0164_H	59_H
irq_i[74]	Product Interrupt Request 74	IRQ74IC	0168_H	$5A_H$
irq_i[75]	Product Interrupt Request 75	IRQ75IC	$016C_H$	$5B_H$
irq_i[76]	Product Interrupt Request 76	IRQ76IC	0170_H	$5C_H$
irq_i[77]	Product Interrupt Request 77	IRQ77IC	0174_H	$5D_H$
irq_i[78]	Product Interrupt Request 78	IRQ78IC	0178_H	$5E_H$
irq_i[79]	Product Interrupt Request 79	IRQ79IC	$017C_H$	$5F_H$
irq_i[80]	Product Interrupt Request 80	IRQ50IC	0180_H	60_H
irq_i[81]	Product Interrupt Request 81	IRQ81IC	0184_H	61_H
irq_i[82]	Product Interrupt Request 82	IRQ82IC	0188_H	62_H
irq_i[83]	Product Interrupt Request 83	IRQ83IC	$018C_H$	63_H
irq_i[84]	Product Interrupt Request 84	IRQ84IC	0190_H	64_H

Table 4-4 Interrupt Vector Table (cont'd) **(sorted by trap number)**

Signal Name (Interrupt IF)	Source of Interrupt	Interrupt Control Register	Vector Location	Trap No
irq_i[85]	Product Interrupt Request 85	IRQ85IC	0194_H	65_H
irq_i[86]	Product Interrupt Request 86	IRQ86IC	0198_H	66_H
irq_i[87]	Product Interrupt Request 87	IRQ87IC	$019C_H$	67_H
irq_i[88]	Product Interrupt Request 88	IRQ88IC	$01A0_H$	68_H
irq_i[89]	Product Interrupt Request 89	IRQ89IC	$01A4_H$	69_H
irq_i[90]	Product Interrupt Request 90	IRQ90IC	$01A8_H$	$6A_H$
irq_i[91]	Product Interrupt Request 91	IRQ91IC	$01AC_H$	$6B_H$
irq_i[92]	Product Interrupt Request 92	IRQ92IC	$01B0_H$	$6C_H$
irq_i[93]	Product Interrupt Request 93	IRQ93IC	$01B4_H$	$6D_H$
irq_i[94]	Product Interrupt Request 94	IRQ94IC	$01B8_H$	$6E_H$
irq_i[95]	Product Interrupt Request 95	IRQ95IC	$01BC_H$	$6F_H$
irq_i[96]	Product Interrupt Request 96	IRQ96IC	$01C0_H$	70_H
irq_i[97]	Product Interrupt Request 97	IRQ97IC	$01C4_H$	71_H
irq_i[98]	Product Interrupt Request 98	IRQ98IC	$01C8_H$	72_H
irq_i[99]	Product Interrupt Request 99	IRQ99IC	$01CC_H$	73_H
irq_i[100]	Product Interrupt Request 100	IRQ100IC	$01D0_H$	74_H
irq_i[101]	Product Interrupt Request 101	IRQ101IC	$01D4_H$	75_H
irq_i[102]	Product Interrupt Request 102	IRQ102IC	$01D8_H$	76_H
irq_i[103]	Product Interrupt Request 103	IRQ103IC	$01DC_H$	77_H
irq_i[104]	Product Interrupt Request 104	IRQ104IC	$01E0_H$	78_H
irq_i[105]	Product Interrupt Request 105	IRQ105IC	$01E4_H$	79_H
irq_i[106]	Product Interrupt Request 106	IRQ106IC	$01E8_H$	$7A_H$
irq_i[107]	Product Interrupt Request 107	IRQ107IC	$01EC_H$	$7B_H$
irq_i[108]	Product Interrupt Request 108	IRQ108IC	$01F0_H$	$7C_H$
irq_i[109]	Product Interrupt Request 109	IRQ109IC	$01F4_H$	$7D_H$
irq_i[110]	Product Interrupt Request 110	IRQ110IC	$01F8_H$	$7E_H$
irq_i[111]	Product Interrupt Request 111	IRQ111IC	$01FC_H$	$7F_H$

1) This signal is part of the general signals interface and is handled by the CPU directly (not handled by the interrupt controller).

2) This signals are not part of the subsystem boundary, but included in the core macro´s interrupt interface.

The following table lists all possible 97 interrupt source entries, which are available on product level (listed by the interrupt interface signal name).

Note: Depending on the number of interrupt configuration not all entries are available (see ***Page 4-43****)*

Table 4-5 Interrupt Vector Table (sorted by signal name)

Signal Name (Interrupt IF)	Source of Interrupt	Interrupt Control Register	Vector Location	Trap No
irq_i[15]	Product Interrupt Request 15	IRQ15IC	$00FC_H$	$3F_H$
irq_i[16]	Product Interrupt Request 16	IRQ16IC	0040_H	10_H
irq_i[17]	Product Interrupt Request 17	IRQ17IC	0044_H	11_H
irq_i[18]	Product Interrupt Request 18	IRQ18IC	0048_H	12_H
irq_i[19]	Product Interrupt Request 19	IRQ19IC	$004C_H$	13_H
irq_i[20]	Product Interrupt Request 20	IRQ20IC	0050_H	14_H
irq_i[21]	Product Interrupt Request 21	IRQ21IC	0054_H	15_H
irq_i[22]	Product Interrupt Request 22	IRQ22IC	0058_H	16_H
irq_i[23]	Product Interrupt Request 23	IRQ23IC	$005C_H$	17_H
irq_i[24]	Product Interrupt Request 24	IRQ24IC	0060_H	18_H
irq_i[25]	Product Interrupt Request 25	IRQ25IC	0064_H	19_H
irq_i[26]	Product Interrupt Request 26	IRQ26IC	0068_H	$1A_H$
irq_i[27]	Product Interrupt Request 27	IRQ27IC	$006C_H$	$1B_H$
irq_i[28]	Product Interrupt Request 28	IRQ28IC	0070_H	$1C_H$
irq_i[29]	Product Interrupt Request 29	IRQ29IC	0074_H	$1D_H$
irq_i[30]	Product Interrupt Request 30	IRQ30IC	0078_H	$1E_H$
irq_i[31]	Product Interrupt Request 31	IRQ31IC	$007C_H$	$1F_H$
irq_i[32]	Product Interrupt Request 32	IRQ32IC	0080_H	20_H
irq_i[33]	Product Interrupt Request 33	IRQ33IC	0084_H	21_H
irq_i[34]	Product Interrupt Request 34	IRQ34IC	$00A0_H$	28_H
irq_i[35]	Product Interrupt Request 35	IRQ35IC	$00A4_H$	29_H
irq_i[36]	Product Interrupt Request 36	IRQ36IC	0100_H	40_H
irq_i[37]	Product Interrupt Request 37	IRQ37IC	0104_H	41_H
irq_i[38]	Product Interrupt Request 38	IRQ38IC	0108_H	42_H
irq_i[39]	Product Interrupt Request 39	IRQ39IC	$010C_H$	43_H

Table 4-5 Interrupt Vector Table (cont'd) **(sorted by signal name)**

Signal Name (Interrupt IF)	Source of Interrupt	Interrupt Control Register	Vector Location	Trap No
irq_i[40]	Product Interrupt Request 40	IRQ40IC	$00F4_H$	$3D_H$
irq_i[41]	Product Interrupt Request 41	IRQ41IC	$00F8_H$	$3E_H$
irq_i[42]	Product Interrupt Request 42	IRQ42IC	0120_H	48_H
irq_i[43]	Product Interrupt Request 43	IRQ43IC	0124_H	49_H
irq_i[44]	Product Interrupt Request 44	IRQ44IC	0128_H	$4A_H$
irq_i[45]	Product Interrupt Request 45	IRQ45IC	0134_H	$4D_H$
irq_i[46]	Product Interrupt Request 46	IRQ46IC	0138_H	$4E_H$
irq_i[47]	Product Interrupt Request 47	IRQ47IC	$013C_H$	$4F_H$
irq_i[48]	Product Interrupt Request 48	IRQ48IC	$00C0_H$	30_H
irq_i[49]	Product Interrupt Request 49	IRQ49IC	$00C4_H$	31_H
irq_i[50]	Product Interrupt Request 50	IRQ50IC	$00C8_H$	32_H
irq_i[51]	Product Interrupt Request 51	IRQ51IC	$00CC_H$	33_H
irq_i[52]	Product Interrupt Request 52	IRQ52IC	$00D0_H$	34_H
irq_i[53]	Product Interrupt Request 53	IRQ53IC	$00D4_H$	35_H
irq_i[54]	Product Interrupt Request 54	IRQ54IC	$00D8_H$	36_H
irq_i[55]	Product Interrupt Request 55	IRQ55IC	$00DC_H$	37_H
irq_i[56]	Product Interrupt Request 56	IRQ56IC	$00E0_H$	38_H
irq_i[57]	Product Interrupt Request 57	IRQ57IC	$00E4_H$	39_H
irq_i[58]	Product Interrupt Request 58	IRQ58IC	$00E8_H$	$3A_H$
irq_i[59]	Product Interrupt Request 59	IRQ59IC	$00EC_H$	$3B_H$
irq_i[60]	Product Interrupt Request 60	IRQ60IC	$00F0_H$	$3C_H$
irq_i[61]	Product Interrupt Request 61	IRQ61IC	0110_H	44_H
irq_i[62]	Product Interrupt Request 62	IRQ62IC	0114_H	45_H
irq_i[63]	Product Interrupt Request 63	IRQ63IC	0118_H	46_H
irq_i[64]	Product Interrupt Request 64	IRQ64IC	0140_H	50_H
irq_i[65]	Product Interrupt Request 65	IRQ65IC	0144_H	51_H
irq_i[66]	Product Interrupt Request 66	IRQ66IC	0148_H	52_H
irq_i[67]	Product Interrupt Request 67	IRQ67IC	$014C_H$	53_H
irq_i[68]	Product Interrupt Request 68	IRQ68IC	0150_H	54_H

Table 4-5 Interrupt Vector Table (cont'd) **(sorted by signal name)**

Signal Name (Interrupt IF)	Source of Interrupt	Interrupt Control Register	Vector Location	Trap No
irq_i[69]	Product Interrupt Request 69	IRQ69IC	0154_H	55_H
irq_i[70]	Product Interrupt Request 70	IRQ70IC	0158_H	56_H
irq_i[71]	Product Interrupt Request 71	IRQ71IC	$015C_H$	57_H
irq_i[72]	Product Interrupt Request 72	IRQ72IC	0160_H	58_H
irq_i[73]	Product Interrupt Request 73	IRQ73IC	0164_H	59_H
irq_i[74]	Product Interrupt Request 74	IRQ74IC	0168_H	$5A_H$
irq_i[75]	Product Interrupt Request 75	IRQ75IC	$016C_H$	$5B_H$
irq_i[76]	Product Interrupt Request 76	IRQ76IC	0170_H	$5C_H$
irq_i[77]	Product Interrupt Request 77	IRQ77IC	0174_H	$5D_H$
irq_i[78]	Product Interrupt Request 78	IRQ78IC	0178_H	$5E_H$
irq_i[79]	Product Interrupt Request 79	IRQ79IC	$017C_H$	$5F_H$
irq_i[80]	Product Interrupt Request 80	IRQ50IC	0180_H	60_H
irq_i[81]	Product Interrupt Request 81	IRQ81IC	0184_H	61_H
irq_i[82]	Product Interrupt Request 82	IRQ82IC	0188_H	62_H
irq_i[83]	Product Interrupt Request 83	IRQ83IC	$018C_H$	63_H
irq_i[84]	Product Interrupt Request 84	IRQ84IC	0190_H	64_H
irq_i[85]	Product Interrupt Request 85	IRQ85IC	0194_H	65_H
irq_i[86]	Product Interrupt Request 86	IRQ86IC	0198_H	66_H
irq_i[87]	Product Interrupt Request 87	IRQ87IC	$019C_H$	67_H
irq_i[88]	Product Interrupt Request 88	IRQ88IC	$01A0_H$	68_H
irq_i[89]	Product Interrupt Request 89	IRQ89IC	$01A4_H$	69_H
irq_i[90]	Product Interrupt Request 90	IRQ90IC	$01A8_H$	$6A_H$
irq_i[91]	Product Interrupt Request 91	IRQ91IC	$01AC_H$	$6B_H$
irq_i[92]	Product Interrupt Request 92	IRQ92IC	$01B0_H$	$6C_H$
irq_i[93]	Product Interrupt Request 93	IRQ93IC	$01B4_H$	$6D_H$
irq_i[94]	Product Interrupt Request 94	IRQ94IC	$01B8_H$	$6E_H$
irq_i[95]	Product Interrupt Request 95	IRQ95IC	$01BC_H$	$6F_H$
irq_i[96]	Product Interrupt Request 96	IRQ96IC	$01C0_H$	70_H
irq_i[97]	Product Interrupt Request 97	IRQ97IC	$01C4_H$	71_H

Table 4-5 Interrupt Vector Table (cont'd) **(sorted by signal name)**

Signal Name (Interrupt IF)	Source of Interrupt	Interrupt Control Register	Vector Location	Trap No
irq_i[98]	Product Interrupt Request 98	IRQ98IC	$01C8_H$	72_H
irq_i[99]	Product Interrupt Request 99	IRQ99IC	$01CC_H$	73_H
irq_i[100]	Product Interrupt Request 100	IRQ100IC	$01D0_H$	74_H
irq_i[101]	Product Interrupt Request 101	IRQ101IC	$01D4_H$	75_H
irq_i[102]	Product Interrupt Request 102	IRQ102IC	$01D8_H$	76_H
irq_i[103]	Product Interrupt Request 103	IRQ103IC	$01DC_H$	77_H
irq_i[104]	Product Interrupt Request 104	IRQ104IC	$01E0_H$	78_H
irq_i[105]	Product Interrupt Request 105	IRQ105IC	$01E4_H$	79_H
irq_i[106]	Product Interrupt Request 106	IRQ106IC	$01E8_H$	$7A_H$
irq_i[107]	Product Interrupt Request 107	IRQ107IC	$01EC_H$	$7B_H$
irq_i[108]	Product Interrupt Request 108	IRQ108IC	$01F0_H$	$7C_H$
irq_i[109]	Product Interrupt Request 109	IRQ109IC	$01F4_H$	$7D_H$
irq_i[110]	Product Interrupt Request 110	IRQ110IC	$01F8_H$	$7E_H$
irq_i[111]	Product Interrupt Request 111	IRQ111IC	$01FC_H$	$7F_H$

5 Instruction Set

5.1 Short Instruction Summary

The following compressed cross-reference tables quickly identify a specific instruction and provide basic information about it. Two ordering schemes are included:

The first table (two pages) is a compressed cross-reference table that quickly identifies a specific hexadecimal opcode with the respective mnemonic.

The second table lists the instructions by their mnemonic and identifies the addressing modes that may be used with a specific instruction and the instruction length depending on the selected addressing mode. This reference helps to optimize instruction sequences in terms of code size and/or execution time.

Note: Both ordering schemes (hexadecimal opcode and mnemonic) are provided in more detailed lists in the following sections of this manual.

•	**0x**	**1x**	**2x**	**3x**	**4x**	**5x**	**6x**	**7x**
x0	ADD	ADDC	SUB	SUBC	CMP	XOR	AND	OR
x1	ADDB	ADDCB	SUBB	SUBCB	CMPB	XORB	ANDB	ORB
x2	ADD	ADDC	SUB	SUBC	CMP	XOR	AND	OR
x3	ADDB	ADDCB	SUBB	SUBCB	CMPB	XORB	ANDB	ORB
x4	ADD	ADDC	SUB	SUBC	-	XOR	AND	OR
x5	ADDB	ADDCB	SUBB	SUBCB	-	XORB	ANDB	ORB
x6	ADD	ADDC	SUB	SUBC	CMP	XOR	AND	OR
x7	ADDB	ADDCB	SUBB	SUBCB	CMPB	XORB	ANDB	ORB
x8	ADD	ADDC	SUB	SUBC	CMP	XOR	AND	OR
x9	ADDB	ADDCB	SUBB	SUBCB	CMPB	XORB	ANDB	ORB
xA	BFLDL	BFLDH	BCMP	BMOVN	BMOV	BOR	BAND	BXOR
xB	MUL	MULU	PRIOR	-	DIV	DIVU	DIVL	DIVLU
xC	ROL	ROL	ROR	ROR	SHL	SHL	SHR	SHR
xD	JMPR	JMPR	JMPR	JMPR	JMPR	JMPR	JMPR	JMPR
xE	BCLR	BCLR	BCLR	BCLR	BCLR	BCLR	BCLR	BCLR
xF	BSET	BSET	BSET	BSET	BSET	BSET	BSET	BSET

Note: Both ordering schemes (hexadecimal opcode and mnemonic) are provided in more detailed lists in the following sections of this manual.

	8x	**9x**	**Ax**	**Bx**	**Cx**	**Dx**	**Ex**	**Fx**
x0	CMPI1	CMPI2	CMPD1	CMPD2	MOVBZ	MOVBS	MOV	MOV
x1	NEG	CPL	NEGB	CPLB	-	AT/ EXTR	MOVB	MOVB
x2	CMPI1	CMPI2	CMPD1	CMPD2	MOVBZ	MOVBS	PCALL	MOV
x3	-	-	-	-	-	-	-	MOVB
x4	MOV	MOV	MOVB	MOVB	MOV	MOV	MOVB	MOVB
x5	-	-	DIS WDT	EINIT	MOVBZ	MOVBS	-	-
x6	CMPI1	CMPI2	CMPD1	CMPD2	SCXT	SCXT	MOV	MOV
x7	IDLE	PWRDN	SRV WDT	SRST	-	EXTP/S/ R	MOVB	MOVB
x8	MOV	MOV	MOV	MOV	MOV	MOV	MOV	-
x9	MOVB	MOVB	MOVB	MOVB	MOVB	MOVB	MOVB	-
xA	JB	JNB	JBC	JNBS	CALLA	CALLS	JMPA	JMPS
xB	-	TRAP	CALLI	CALLR	RET	RETS	RETP	RETI
xC	-	JMPI	ASHR	ASHR	NOP	EXTP/S/ R	PUSH	POP
xD	JMPR	JMPR	JMPR	JMPR	JMPR	JMPR	JMPR	JMPR
xE	BCLR	BCLR	BCLR	BCLR	BCLR	BCLR	BCLR	BCLR
xF	BSET	BSET	BSET	BSET	BSET	BSET	BSET	BSET

5.2 Instruction Set Summary

This chapter summarizes the instructions by listing them according to their functional class. This allows to identify the right instruction(s) for a specific required function.

The following notes apply to this summary:

Data Addressing Modes

Rw: – Word GPR (R0, R1, … , R15)

Rb: – Byte GPR (RL0, RH0, …, RL7, RH7)

reg: – SFR or GPR
(in case of a byte operation on an SFR, only the low byte can be accessed via 'reg')

mem: – Direct word or byte memory location

[…]: – Indirect word or byte memory location
(Any word GPR can be used as indirect address pointer, except for the arithmetic, logical and compare instructions, where only R0 to R3 are allowed)

bitaddr: – Direct bit in the bit-addressable memory area

bitoff: – Direct word in the bit-addressable memory area

#data: – Immediate constant
(The number of significant bits which can be specified by the user is represented by the respective appendix 'x')

#mask8: – Immediate 8-bit mask used for bit-field modifications

Multiply and Divide Operations

The MDL and MDH registers are implicit source and/or destination operands of the multiply and divide instructions.

Branch Target Addressing Modes

caddr: – Direct 16-bit jump target address (Updates the Instruction Pointer)

seg: – Direct 2-bit segment address
(Updates the Code Segment Pointer)

rel: – Signed 8-bit jump target word offset address relative to the Instruction Pointer of the following instruction

#trap7: – Immediate 7-bit trap or interrupt number.

Extension Operations

The EXT* instructions override the standard DPP addressing scheme:

#pag10: – Immediate 10-bit page address.

#seg8: – Immediate 8-bit segment address.

Branch Condition Codes

cc: Symbolically specifiable condition codes

cc_UC	–Unconditional
cc_Z	–Zero
cc_NZ	–Not Zero
cc_V	–Overflow
cc_NV	–No Overflow
cc_N	–Negative
cc_NN	–Not Negative
cc_C	–Carry
cc_NC	–No Carry
cc_EQ	–Equal
cc_NE	–Not Equal
cc_ULT	–Unsigned Less Than
cc_ULE	–Unsigned Less Than or Equal
cc_UGE	–Unsigned Greater Than or Equal
cc_UGT	–Unsigned Greater Than
cc_SLE	–Signed Less Than or Equal
cc_SGE	–Signed Greater Than or Equal
cc_SGT	–Signed Greater Than
cc_NET	–Not Equal and Not End-of-Table

Mnemonic	Addressing Modes			Bytes
ADD[B] ADDC[B] AND[B] OR[B] SUB[B] SUBC[B] XOR[B]	Rwn Rwn Rwn Rwn reg reg mem	Rwm [Rwi] [Rwi+] #data3 #data16 mem reg	1) 1) 1) 1)	2 2 2 2 4 4 4
ASHR ROL / ROR SHL / SHR	Rwn Rwn	Rwm #data4		2 2
BAND BCMP BMOV BMOVN BOR / BXOR	bitaddrZ.z	bitaddrQ.q		4
BCLR BSET	bitaddrQ.q			2
BFLDH BFLDL	bitoffQ	#mask8 #data8		4
MOV[B]	Rwn Rwn Rwn Rwn [Rwm] [-Rwm] [Rwn] [Rwn+] [Rwn] reg Rwn [Rwm+#d16] [Rwn] mem reg mem	Rwm #data4 [Rwm] [Rwm+] Rwn Rwn [Rwm] [Rwm] [Rwm+] #data16 [Rwm+#d16] Rwn mem [Rwn] mem reg	1) 1) 1) 1) 1) 1) 2) 1) 1)	2 2 2 2 2 2 2 2 2 4 4 4 4 4 4 4
MOVBS MOVBZ	Rwn reg mem	Rbm mem reg		2 4 4
EXTS EXTSR	Rwm #seg	#irang2 #irang2		2 4
NOP RET RETI RETS	-			2

Mnemonic	Addressing Modes			Bytes
CPL[B] NEG[B]	Rwn		1)	2
DIV DIVL DIVLU DIVU	Rwn			2
MUL MULU	Rwn	Rwm		2
CMPD1/2 CMPI1/2	Rwn Rwn Rwn	#data4 #data16 mem		2 4 4
CMP[B]	Rwn Rwn Rwn Rwn reg reg	Rwm [Rwi] [Rwi+] #data3 #data16 mem	1) 1) 1) 1) 2)	2 2 2 2 4 4
CALLA JMPA	cc	caddr		4
CALLI JMPI	cc	[Rwn]		2
CALLS JMPS	seg	caddr		4
CALLR	rel			2
JMPR	cc	rel		2
JB JBC JNB JNBS	bitaddrQ.q	rel		4
PCALL	reg	caddr		4
POP PUSH RETP	reg			2
SCXT	reg reg	#data16 mem		4 4
PRIOR	Rwn	Rwm		2
TRAP	#trap7			2
ATOMIC EXTR	#irang2			2
EXTP EXTPR	Rwm #pag	#irang2 #irang2		2 4
SRST/IDLE PWRDN SRVWDT DISWDT EINIT	-			4

1) Byte oriented instructions (suffix 'B') use Rb instead of Rw (not with [Rwn]!).
2) Byte oriented instructions (suffix 'B') use #data8 instead of #data16.

Instruction Set Summary

Mnemonic		Description	Bytes
Arithmetic Operations			
ADD	Rw, Rw	Add direct word GPR to direct GPR	2
ADD	Rw, [Rw]	Add indirect word memory to direct GPR	2
ADD	Rw, [Rw +]	Add indirect word memory to direct GPR and post-increment source pointer by 2	2
ADD	Rw, #data3	Add immediate word data to direct GPR	2
ADD	reg, #data16	Add immediate word data to direct register	4
ADD	reg, mem	Add direct word memory to direct register	4
ADD	mem, reg	Add direct word register to direct memory	4
ADDB	Rb, Rb	Add direct byte GPR to direct GPR	2
ADDB	Rb, [Rw]	Add indirect byte memory to direct GPR	2
ADDB	Rb, [Rw +]	Add indirect byte memory to direct GPR and post-increment source pointer by 1	2
ADDB	Rb, #data3	Add immediate byte data to direct GPR	2
ADDB	reg, #data8	Add immediate byte data to direct register	4
ADDB	reg, mem	Add direct byte memory to direct register	4
ADDB	mem, reg	Add direct byte register to direct memory	4
ADDC	Rw, Rw	Add direct word GPR to direct GPR with Carry	2
ADDC	Rw, [Rw]	Add indirect word memory to direct GPR with Carry	2
ADDC	Rw, [Rw +]	Add indirect word memory to direct GPR with Carry and post-increment source pointer by 2	2
ADDC	Rw, #data3	Add immediate word data to direct GPR with Carry	2
ADDC	reg, #data16	Add immediate word data to direct register with Carry	4
ADDC	reg, mem	Add direct word memory to direct register with Carry	4
ADDC	mem, reg	Add direct word register to direct memory with Carry	4
ADDCB	Rb, Rb	Add direct byte GPR to direct GPR with Carry	2
ADDCB	Rb, [Rw]	Add indirect byte memory to direct GPR with Carry	2
ADDCB	Rb, [Rw +]	Add indirect byte memory to direct GPR with Carry and post-increment source pointer by 1	2
ADDCB	Rb, #data3	Add immediate byte data to direct GPR with Carry	2
ADDCB	reg, #data8	Add immediate byte data to direct register with Carry	4
ADDCB	reg, mem	Add direct byte memory to direct register with Carry	4

Instruction Set Summary (cont'd)

Mnemonic		Description	Bytes
Arithmetic Operations (cont'd)			
ADDCB	mem, reg	Add direct byte register to direct memory with Carry	4
SUB	Rw, Rw	Subtract direct word GPR from direct GPR	2
SUB	Rw, [Rw]	Subtract indirect word memory from direct GPR	2
SUB	Rw, [Rw +]	Subtract indirect word memory from direct GPR and post-increment source pointer by 2	2
SUB	Rw, #data3	Subtract immediate word data from direct GPR	2
SUB	reg, #data16	Subtract immediate word data from direct register	4
SUB	reg, mem	Subtract direct word memory from direct register	4
SUB	mem, reg	Subtract direct word register from direct memory	4
SUBB	Rb, Rb	Subtract direct byte GPR from direct GPR	2
SUBB	Rb, [Rw]	Subtract indirect byte memory from direct GPR	2
SUBB	Rb, [Rw +]	Subtract indirect byte memory from direct GPR and post-increment source pointer by 1	2
SUBB	Rb, #data3	Subtract immediate byte data from direct GPR	2
SUBB	reg, #data8	Subtract immediate byte data from direct register	4
SUBB	reg, mem	Subtract direct byte memory from direct register	4
SUBB	mem, reg	Subtract direct byte register from direct memory	4
SUBC	Rw, Rw	Subtract direct word GPR from direct GPR with Carry	2
SUBC	Rw, [Rw]	Subtract indirect word memory from direct GPR with Carry	2
SUBC	Rw, [Rw +]	Subtract indirect word memory from direct GPR with Carry and post-increment source pointer by 2	2
SUBC	Rw, #data3	Subtract immediate word data from direct GPR with Carry	2
SUBC	reg, #data16	Subtract immediate word data from direct register with Carry	4
SUBC	reg, mem	Subtract direct word memory from direct register with Carry	4
SUBC	mem, reg	Subtract direct word register from direct memory with Carry	4
SUBCB	Rb, Rb	Subtract direct byte GPR from direct GPR with Carry	2
SUBCB	Rb, [Rw]	Subtract indirect byte memory from direct GPR with Carry	2
SUBCB	Rb, [Rw +]	Subtract indirect byte memory from direct GPR with Carry and post-increment source pointer by 1	2
SUBCB	Rb, #data3	Subtract immediate byte data from direct GPR with Carry	2
SUBCB	reg, #data8	Subtract immediate byte data from direct register with Carry	4

Instruction Set Summary (cont'd)

Mnemonic		Description	Bytes
Arithmetic Operations (cont'd)			
SUBCB	reg, mem	Subtract direct byte memory from direct register with Carry	4
SUBCB	mem, reg	Subtract direct byte register from direct memory with Carry	4
MUL	Rw, Rw	Signed multiply direct GPR by direct GPR (16-16-bit)	2
MULU	Rw, Rw	Unsigned multiply direct GPR by direct GPR (16-16-bit)	2
DIV	Rw	Signed divide register MDL by direct GPR (16-/16-bit)	2
DIVL	Rw	Signed long divide register MD by direct GPR (32-/16-bit)	2
DIVLU	Rw	Unsigned long divide register MD by direct GPR (32-/16-bit)	2
DIVU	Rw	Unsigned divide register MDL by direct GPR (16-/16-bit)	2
CPL	Rw	Complement direct word GPR	2
CPLB	Rb	Complement direct byte GPR	2
NEG	Rw	Negate direct word GPR	2
NEGB	Rb	Negate direct byte GPR	2
Logical Instructions			
AND	Rw, Rw	Bitwise AND direct word GPR with direct GPR	2
AND	Rw, [Rw]	Bitwise AND indirect word memory with direct GPR	2
AND	Rw, [Rw +]	Bitwise AND indirect word memory with direct GPR and post-increment source pointer by 2	2
AND	Rw, #data3	Bitwise AND immediate word data with direct GPR	2
AND	reg, #data16	Bitwise AND immediate word data with direct register	4
AND	reg, mem	Bitwise AND direct word memory with direct register	4
AND	mem, reg	Bitwise AND direct word register with direct memory	4
ANDB	Rb, Rb	Bitwise AND direct byte GPR with direct GPR	2
ANDB	Rb, [Rw]	Bitwise AND indirect byte memory with direct GPR	2
ANDB	Rb, [Rw +]	Bitwise AND indirect byte memory with direct GPR and post-increment source pointer by 1	2
ANDB	Rb, #data3	Bitwise AND immediate byte data with direct GPR	2
ANDB	reg, #data8	Bitwise AND immediate byte data with direct register	4
ANDB	reg, mem	Bitwise AND direct byte memory with direct register	4
ANDB	mem, reg	Bitwise AND direct byte register with direct memory	4

Instruction Set Summary (cont'd)

Mnemonic		Description	Bytes
Logical Instructions (cont'd)			
OR	Rw, Rw	Bitwise OR direct word GPR with direct GPR	2
OR	Rw, [Rw]	Bitwise OR indirect word memory with direct GPR	2
OR	Rw, [Rw +]	Bitwise OR indirect word memory with direct GPR and post-increment source pointer by 2	2
OR	Rw, #data3	Bitwise OR immediate word data with direct GPR	2
OR	reg, #data16	Bitwise OR immediate word data with direct register	4
OR	reg, mem	Bitwise OR direct word memory with direct register	4
OR	mem, reg	Bitwise OR direct word register with direct memory	4
ORB	Rb, Rb	Bitwise OR direct byte GPR with direct GPR	2
ORB	Rb, [Rw]	Bitwise OR indirect byte memory with direct GPR	2
ORB	Rb, [Rw +]	Bitwise OR indirect byte memory with direct GPR and post-increment source pointer by 1	2
ORB	Rb, #data3	Bitwise OR immediate byte data with direct GPR	2
ORB	reg, #data8	Bitwise OR immediate byte data with direct register	4
ORB	reg, mem	Bitwise OR direct byte memory with direct register	4
ORB	mem, reg	Bitwise OR direct byte register with direct memory	4
XOR	Rw, Rw	Bitwise XOR direct word GPR with direct GPR	2
XOR	Rw, [Rw]	Bitwise XOR indirect word memory with direct GPR	2
XOR	Rw, [Rw +]	Bitwise XOR indirect word memory with direct GPR and post-increment source pointer by 2	2
XOR	Rw, #data3	Bitwise XOR immediate word data with direct GPR	2
XOR	reg, #data16	Bitwise XOR immediate word data with direct register	4
XOR	reg, mem	Bitwise XOR direct word memory with direct register	4
XOR	mem, reg	Bitwise XOR direct word register with direct memory	4
XORB	Rb, Rb	Bitwise XOR direct byte GPR with direct GPR	2
XORB	Rb, [Rw]	Bitwise XOR indirect byte memory with direct GPR	2
XORB	Rb, [Rw +]	Bitwise XOR indirect byte memory with direct GPR and post-increment source pointer by 1	2
XORB	Rb, #data3	Bitwise XOR immediate byte data with direct GPR	2
XORB	reg, #data8	Bitwise XOR immediate byte data with direct register	4
XORB	reg, mem	Bitwise XOR direct byte memory with direct register	4
XORB	mem, reg	Bitwise XOR direct byte register with direct memory	4

Instruction Set Summary (cont'd)

Mnemonic		Description	Bytes
Boolean Bit Manipulation Operations			
BCLR	bitaddr	Clear direct bit	2
BSET	bitaddr	Set direct bit	2
BMOV	bitaddr, bitaddr	Move direct bit to direct bit	4
BMOVN	bitaddr, bitaddr	Move negated direct bit to direct bit	4
BAND	bitaddr, bitaddr	AND direct bit with direct bit	4
BOR	bitaddr, bitaddr	OR direct bit with direct bit	4
BXOR	bitaddr, bitaddr	XOR direct bit with direct bit	4
BCMP	bitaddr, bitaddr	Compare direct bit to direct bit	4
BFLDH	bitoff, #mask8, #data8	Bitwise modify masked high byte of bit-addressable direct word memory with immediate data	4
BFLDL	bitoff, #mask8, #data8	Bitwise modify masked low byte of bit-addressable direct word memory with immediate data	4
CMP	Rw, Rw	Compare direct word GPR to direct GPR	2
CMP	Rw, [Rw]	Compare indirect word memory to direct GPR	2
CMP	Rw, [Rw +]	Compare indirect word memory to direct GPR and post-increment source pointer by 2	2
CMP	Rw, #data3	Compare immediate word data to direct GPR	2
CMP	reg, #data16	Compare immediate word data to direct register	4
CMP	reg, mem	Compare direct word memory to direct register	4
CMPB	Rb, Rb	Compare direct byte GPR to direct GPR	2
CMPB	Rb, [Rw]	Compare indirect byte memory to direct GPR	2
CMPB	Rb, [Rw +]	Compare indirect byte memory to direct GPR and post-increment source pointer by 1	2
CMPB	Rb, #data3	Compare immediate byte data to direct GPR	2
CMPB	reg, #data8	Compare immediate byte data to direct register	4
CMPB	reg, mem	Compare direct byte memory to direct register	4
Compare and Loop Control Instructions			
CMPD1	Rw, #data4	Compare immediate word data to direct GPR and decrement GPR by 1	2
CMPD1	Rw, #data16	Compare immediate word data to direct GPR and decrement GPR by 1	4

Instruction Set Summary (cont'd)

Mnemonic		Description	Bytes
Compare and Loop Control Instructions (cont'd)			
CMPD1	Rw, mem	Compare direct word memory to direct GPR and decrement GPR by 1	4
CMPD2	Rw, #data4	Compare immediate word data to direct GPR and decrement GPR by 2	2
CMPD2	Rw, #data16	Compare immediate word data to direct GPR and decrement GPR by 2	4
CMPD2	Rw, mem	Compare direct word memory to direct GPR and decrement GPR by 2	4
CMPI1	Rw, #data4	Compare immediate word data to direct GPR and increment GPR by 1	2
CMPI1	Rw, #data16	Compare immediate word data to direct GPR and increment GPR by 1	4
CMPI1	Rw, mem	Compare direct word memory to direct GPR and increment GPR by 1	4
CMPI2	Rw, #data4	Compare immediate word data to direct GPR and increment GPR by 2	2
CMPI2	Rw, #data16	Compare immediate word data to direct GPR and increment GPR by 2	4
CMPI2	Rw, mem	Compare direct word memory to direct GPR and increment GPR by 2	4
Prioritize Instruction			
PRIOR	Rw, Rw	Determine number of shift cycles to normalize direct word GPR and store result in direct word GPR	2
Shift and Rotate Instructions			
SHL	Rw, Rw	Shift left direct word GPR; number of shift cycles specified by direct GPR	2
SHL	Rw, #data4	Shift left direct word GPR; number of shift cycles specified by immediate data	2
SHR	Rw, Rw	Shift right direct word GPR; number of shift cycles specified by direct GPR	2

Instruction Set Summary (cont'd)

Mnemonic		Description	Bytes
Shift and Rotate Instructions (cont'd)			
SHR	Rw, #data4	Shift right direct word GPR; number of shift cycles specified by immediate data	2
ROL	Rw, Rw	Rotate left direct word GPR; number of shift cycles specified by direct GPR	2
ROL	Rw, #data4	Rotate left direct word GPR; number of shift cycles specified by immediate data	2
ROR	Rw, Rw	Rotate right direct word GPR; number of shift cycles specified by direct GPR	2
ROR	Rw, #data4	Rotate right direct word GPR; number of shift cycles specified by immediate data	2
ASHR	Rw, Rw	Arithmetic (sign bit) shift right direct word GPR; number of shift cycles specified by direct GPR	2
ASHR	Rw, #data4	Arithmetic (sign bit) shift right direct word GPR; number of shift cycles specified by immediate data	2
Data Movement			
MOV	Rw, Rw	Move direct word GPR to direct GPR	2
MOV	Rw, #data4	Move immediate word data to direct GPR	2
MOV	reg, #data16	Move immediate word data to direct register	4
MOV	Rw, [Rw]	Move indirect word memory to direct GPR	2
MOV	Rw, [Rw +]	Move indirect word memory to direct GPR and post-increment source pointer by 2	2
MOV	[Rw], Rw	Move direct word GPR to indirect memory	2
MOV	[-Rw], Rw	Pre-decrement destination pointer by 2 and move direct word GPR to indirect memory	2
MOV	[Rw], [Rw]	Move indirect word memory to indirect memory	2
MOV	[Rw +], [Rw]	Move indirect word memory to indirect memory and post-increment destination pointer by 2	2
MOV	[Rw], [Rw +]	Move indirect word memory to indirect memory and post-increment source pointer by 2	2
MOV	Rw, [Rw + #data16]	Move indirect word memory by base plus constant to direct GPR	4
MOV	[Rw + #data16], Rw	Move direct word GPR to indirect memory by base plus constant	4

Instruction Set Summary (cont'd)

Mnemonic		Description	Bytes
Data Movement (cont'd)			
MOV	[Rw], mem	Move direct word memory to indirect memory	4
MOV	mem, [Rw]	Move indirect word memory to direct memory	4
MOV	reg, mem	Move direct word memory to direct register	4
MOV	mem, reg	Move direct word register to direct memory	4
MOVB	Rb, Rb	Move direct byte GPR to direct GPR	2
MOVB	Rb, #data4	Move immediate byte data to direct GPR	2
MOVB	reg, #data8	Move immediate byte data to direct register	4
MOVB	Rb, [Rw]	Move indirect byte memory to direct GPR	2
MOVB	Rb, [Rw +]	Move indirect byte memory to direct GPR and post-increment source pointer by 1	2
MOVB	[Rw], Rb	Move direct byte GPR to indirect memory	2
MOVB	[-Rw], Rb	Pre-decrement destination pointer by 1 and move direct byte GPR to indirect memory	2
MOVB	[Rw], [Rw]	Move indirect byte memory to indirect memory	2
MOVB	[Rw +], [Rw]	Move indirect byte memory to indirect memory and post-increment destination pointer by 1	2
MOVB	[Rw], [Rw +]	Move indirect byte memory to indirect memory and post-increment source pointer by 1	2
MOVB	Rb, [Rw + #data16]	Move indirect byte memory by base plus constant to direct GPR	4
MOVB	[Rw + #data16], Rb	Move direct byte GPR to indirect memory by base plus constant	4
MOVB	[Rw], mem	Move direct byte memory to indirect memory	4
MOVB	mem, [Rw]	Move indirect byte memory to direct memory	4
MOVB	reg, mem	Move direct byte memory to direct register	4
MOVB	mem, reg	Move direct byte register to direct memory	4
MOVBS	Rw, Rb	Move direct byte GPR with sign extension to direct word GPR	2
MOVBS	reg, mem	Move direct byte memory with sign extension to direct word register	4
MOVBS	mem, reg	Move direct byte register with sign extension to direct word memory	4

Instruction Set Summary (cont'd)

Mnemonic		Description	Bytes
Data Movement (cont'd)			
MOVBZ	Rw, Rb	Move direct byte GPR with zero extension to direct word GPR	2
MOVBZ	reg, mem	Move direct byte memory with zero extension to direct word register	4
MOVBZ	mem, reg	Move direct byte register with zero extension to direct word memory	4
Jump and Call Operations			
JMPA	cc, caddr	Jump absolute if condition is met	4
JMPI	cc, [Rw]	Jump indirect if condition is met	2
JMPR	cc, rel	Jump relative if condition is met	2
JMPS	seg, caddr	Jump absolute to a code segment	4
JB	bitaddr, rel	Jump relative if direct bit is set	4
JBC	bitaddr, rel	Jump relative and clear bit if direct bit is set	4
JNB	bitaddr, rel	Jump relative if direct bit is not set	4
JNBS	bitaddr, rel	Jump relative and set bit if direct bit is not set	4
CALLA	cc, caddr	Call absolute subroutine if condition is met	4
CALLI	cc, [Rw]	Call indirect subroutine if condition is met	2
CALLR	rel	Call relative subroutine	2
CALLS	seg, caddr	Call absolute subroutine in any code segment	4
PCALL	reg, caddr	Push direct word register onto system stack and call absolute subroutine	4
TRAP	#trap7	Call interrupt service routine via immediate trap number	2
System Stack Operations			
POP	reg	Pop direct word register from system stack	2
PUSH	reg	Push direct word register onto system stack	2
SCXT	reg, #data16	Push direct word register onto system stack und update register with immediate data	4
SCXT	reg, mem	Push direct word register onto system stack und update register with direct memory	4

Instruction Set Summary (cont'd)

Mnemonic		Description	Bytes
Return Operations			
RET		Return from intra-segment subroutine	2
RETS		Return from inter-segment subroutine	2
RETP	reg	Return from intra-segment subroutine and pop direct word register from system stack	2
RETI		Return from interrupt service subroutine	2
System Control			
SRST		Software Reset	4
IDLE		Enter Idle Mode	4
PWRDN		Enter Power Down Mode (supposes NMI-pin being low)	4
SRVWDT		Service Watchdog Timer	4
DISWDT		Disable Watchdog Timer	4
EINIT		Signify End-of-Initialization on RSTOUT-pin	4
ATOMIC	#irang2	Begin ATOMIC sequence *)	2
EXTR	#irang2	Begin EXTended Register sequence *)	2
EXTP	Rw, #irang2	Begin EXTended Page sequence *)	2
EXTP	#pag10, #irang2	Begin EXTended Page sequence *)	4
EXTPR	Rw, #irang2	Begin EXTended Page and Register sequence *)	2
EXTPR	#pag10, #irang2	Begin EXTended Page and Register sequence *)	4
EXTS	Rw, #irang2	Begin EXTended Segment sequence *)	2
EXTS	#seg8, #irang2	Begin EXTended Segment sequence *)	4
EXTSR	Rw, #irang2	Begin EXTended Segment and Register sequence *)	2
EXTSR	#seg8, #irang2	Begin EXTended Segment and Register sequence *)	4
Miscellaneous			
NOP		Null operation	2

5.3 Instruction Opcodes

The following pages list the instructions of the C166S ordered by their hexadecimal opcodes. This helps to identify specific instructions when reading executable code, ie. during the debugging phase.

Notes for Opcode Lists

1. These instructions are encoded by means of additional bits in the operand field of the instruction

$x0_H$ – $x7_H$:	Rw, #data3	or	Rb, #data3
$x8_H$ – xB_H:	Rw, [Rw]	or	Rb, [Rw]
xC_H – xF_H:	Rw, [Rw +]	or	Rb, [Rw +]

For these instructions only the lowest four GPRs, R0 to R3, can be used as indirect address pointers.

2. These instructions are encoded by means of additional bits in the operand field of the instruction

$00xx.xxxx_B$:	EXTS	or	ATOMIC
$01xx.xxxx_B$:	EXTP		
$10xx.xxxx_B$:	EXTSR	or	EXTR
$11xx.xxxx_B$:	EXTPR		

Notes on the JMPR Instructions

The condition code to be tested for the JMPR instructions is specified by the opcode. Two mnemonic representation alternatives exist for some of the condition codes.

Notes on the BCLR and BSET Instructions

The position of the bit to be set or to be cleared is specified by the opcode. The operand 'bitoff.n' (n = 0 to 15) refers to a particular bit within a bit-addressable word.

Notes on the Undefined Opcodes

A hardware trap occurs when one of the undefined opcodes signified by '----' is decoded by the CPU.

Hex-code	Number of Bytes	Mnemonic	Operands		Hex-code	Number of Bytes	Mnemonic	Operands
00	2	ADD	Rw, Rw		20	2	SUB	Rw, Rw
01	2	ADDB	Rb, Rb		21	2	SUBB	Rb, Rb
02	4	ADD	reg, mem		22	4	SUB	reg, mem
03	4	ADDB	reg, mem		23	4	SUBB	reg, mem
04	4	ADD	mem, reg		24	4	SUB	mem, reg
05	4	ADDB	mem, reg		25	4	SUBB	mem, reg
06	4	ADD	reg, #data16		26	4	SUB	reg, #data16
07	4	ADDB	reg, #data8		27	4	SUBB	reg, #data8
08	2	ADD	Rw, [Rw +] or Rw, [Rw] or Rw, #data3		28	2	SUB	Rw, [Rw +] or Rw, [Rw] or Rw, #data3
09	2	ADDB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3		29	2	SUBB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3
0A	4	BFLDL	bitoff, #mask8, #data8		2A	4	BCMP	bitaddr, bitaddr
0B	2	MUL	Rw, Rw		2B	2	PRIOR	Rw, Rw
0C	2	ROL	Rw, Rw		2C	2	ROR	Rw, Rw
0D	2	JMPR	cc_UC, rel		2D	2	JMPR	cc_EQ, rel or cc_Z, rel
0E	2	BCLR	bitoff.0		2E	2	BCLR	bitoff.2
0F	2	BSET	bitoff.0		2F	2	BSET	bitoff.2
10	2	ADDC	Rw, Rw		30	2	SUBC	Rw, Rw
11	2	ADDCB	Rb, Rb		31	2	SUBCB	Rb, Rb
12	4	ADDC	reg, mem		32	4	SUBC	reg, mem
13	4	ADDCB	reg, mem		33	4	SUBCB	reg, mem
14	4	ADDC	mem, reg		34	4	SUBC	mem, reg
15	4	ADDCB	mem, reg		35	4	SUBCB	mem, reg
16	4	ADDC	reg, #data16		36	4	SUBC	reg, #data16
17	4	ADDCB	reg, #data8		37	4	SUBCB	reg, #data8
18	2	ADDC	Rw, [Rw +] or Rw, [Rw] or Rw, #data3		38	2	SUBC	Rw, [Rw +] or Rw, [Rw] or Rw, #data3
19	2	ADDCB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3		39	2	SUBCB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3
1A	4	BFLDH	bitoff, #mask8, #data8		3A	4	BMOVN	bitaddr, bitaddr
1B	2	MULU	Rw, Rw		3B	-	-	-
1C	2	ROL	Rw, #data4		3C	2	ROR	Rw, #data4
1D	2	JMPR	cc_NET, rel		3D	2	JMPR	cc_NE, rel or cc_NZ, rel
1E	2	BCLR	bitoff.1		3E	2	BCLR	bitoff.3
1F	2	BSET	bitoff.1		3F	2	BSET	bitoff.3

Hex-code	Number of Bytes	Mnemonic	Operands		Hex-code	Number of Bytes	Mnemonic	Operands
40	2	CMP	Rw, Rw		60	2	AND	Rw, Rw
41	2	CMPB	Rb, Rb		61	2	ANDB	Rb, Rb
42	4	CMP	reg, mem		62	4	AND	reg, mem
43	4	CMPB	reg, mem		63	4	ANDB	reg, mem
44	-	-	-		64	4	AND	mem, reg
45	-	-	-		65	4	ANDB	mem, reg
46	4	CMP	reg, #data16		66	4	AND	reg, #data16
47	4	CMPB	reg, #data8		67	4	ANDB	reg, #data8
48	2	CMP	Rw, [Rw +] or Rw, [Rw] or Rw, #data3		68	2	AND	Rw, [Rw +] or Rw, [Rw] or Rw, #data3
49	2	CMPB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3		69	2	ANDB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3
4A	4	BMOV	bitaddr, bitaddr		6A	4	BAND	bitaddr, bitaddr
4B	2	DIV	Rw		6B	2	DIVL	Rw
4C	2	SHL	Rw, Rw		6C	2	SHR	Rw, Rw
4D	2	JMPR	cc_V, rel		6D	2	JMPR	cc_N, rel
4E	2	BCLR	bitoff.4		6E	2	BCLR	bitoff.6
4F	2	BSET	bitoff.4		6F	2	BSET	bitoff.6
50	2	XOR	Rw, Rw		70	2	OR	Rw, Rw
51	2	XORB	Rb, Rb		71	2	ORB	Rb, Rb
52	4	XOR	reg, mem		72	4	OR	reg, mem
53	4	XORB	reg, mem		73	4	ORB	reg, mem
54	4	XOR	mem, reg		74	4	OR	mem, reg
55	4	XORB	mem, reg		75	4	ORB	mem, reg
56	4	XOR	reg, #data16		76	4	OR	reg, #data16
57	4	XORB	reg, #data8		77	4	ORB	reg, #data8
58	2	XOR	Rw, [Rw +] or Rw, [Rw] or Rw, #data3		78	2	OR	Rw, [Rw +] or Rw, [Rw] or Rw, #data3 [1)]
59	2	XORB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3		79	2	ORB	Rb, [Rw +] or Rb, [Rw] or Rb, #data3
5A	4	BOR	bitaddr, bitaddr		7A	4	BXOR	bitaddr, bitaddr
5B	2	DIVU	Rw		7B	2	DIVLU	Rw
5C	2	SHL	Rw, #data4		7C	2	SHR	Rw, #data4
5D	2	JMPR	cc_NV, rel		7D	2	JMPR	cc_NN, rel
5E	2	BCLR	bitoff.5		7E	2	BCLR	bitoff.7
5F	2	BSET	bitoff.5		7F	2	BSET	bitoff.7

Hex-code	Number of Bytes	Mnemonic	Operands		Hex-code	Number of Bytes	Mnemonic	Operands
80	2	CMPI1	Rw, #data4		A0	2	CMPD1	Rw, #data4
81	2	NEG	Rw		A1	2	NEGB	Rb
82	4	CMPI1	Rw, mem		A2	4	CMPD1	Rw, mem
83	4	CoXXX	xx		A3	4	CoXXX	xx
84	4	MOV	[Rw], mem		A4	4	MOVB	[Rw], mem
85	-	-	-		A5	4	DISWDT	
86	4	CMPI1	Rw, #data16		A6	4	CMPD1	Rw, #data16
87	4	IDLE			A7	4	SRVWDT	
88	2	MOV	[-Rw], Rw		A8	2	MOV	Rw, [Rw]
89	2	MOVB	[-Rw], Rb		A9	2	MOVB	Rb, [Rw]
8A	4	JB	bitaddr, rel		AA	4	JBC	bitaddr, rel
8B	-	-	-		AB	2	CALLI	cc, [Rw]
8C	-	-	-		AC	2	ASHR	Rw, Rw
8D	2	JMPR	cc_C, rel or cc_ULT, rel		AD	2	JMPR	cc_SGT, rel
8E	2	BCLR	bitoff.8		AE	2	BCLR	bitoff.10
8F	2	BSET	bitoff.8		AF	2	BSET	bitoff.10
90	2	CMPI2	Rw, #data4		B0	2	CMPD2	Rw, #data4
91	2	CPL	Rw		B1	2	CPLB	Rb
92	4	CMPI2	Rw, mem		B2	4	CMPD2	Rw, mem
93	4	-	-		B3	4	-	-
94	4	MOV	mem, [Rw]		B4	4	MOVB	mem, [Rw]
95	-	-	-		B5	4	EINIT	
96	4	CMPI2	Rw, #data16		B6	4	CMPD2	Rw, #data16
97	4	PWRDN			B7	4	SRST	
98	2	MOV	Rw, [Rw+]		B8	2	MOV	[Rw], Rw
99	2	MOVB	Rb, [Rw+]		B9	2	MOVB	[Rw], Rb
9A	4	JNB	bitaddr, rel		BA	4	JNBS	bitaddr, rel
9B	2	TRAP	#trap7		BB	2	CALLR	rel
9C	2	JMPI	cc, [Rw]		BC	2	ASHR	Rw, #data4
9D	2	JMPR	cc_NC, rel or cc_UGE, rel		BD	2	JMPR	cc_SLE, rel
9E	2	BCLR	bitoff.9		BE	2	BCLR	bitoff.11
9F	2	BSET	bitoff.9		BF	2	BSET	bitoff.11

Hex-code	Num-ber of Bytes	Mnemonic	Operands		Hex-code	Num-ber of Bytes	Mnemonic	Operands
C0	2	MOVBZ	Rw, Rb		E0	2	MOV	Rw, #data4
C1	-	-	-		E1	2	MOVB	Rb, #data4
C2	4	MOVBZ	reg, mem		E2	4	PCALL	reg, caddr
C3	4	-	-		E3	-	-	-
C4	4	MOV	[Rw+#data16], Rw		E4	4	MOVB	[Rw+#data16], Rb
C5	4	MOVBZ	mem, reg		E5	-	-	-
C6	4	SCXT	reg, #data16		E6	4	MOV	reg, #data16
C7	-	-	-		E7	4	MOVB	reg, #data8
C8	2	MOV	[Rw], [Rw]		E8	2	MOV	[Rw], [Rw+]
C9	2	MOVB	[Rw], [Rw]		E9	2	MOVB	[Rw], [Rw+]
CA	4	CALLA	cc, addr		EA	4	JMPA	cc, caddr
CB	2	RET			EB	2	RETP	reg
CC	2	NOP			EC	2	PUSH	reg
CD	2	JMPR	cc_SLT, rel		ED	2	JMPR	cc_UGT, rel
CE	2	BCLR	bitoff.12		EE	2	BCLR	bitoff.14
CF	2	BSET	bitoff.12		EF	2	BSET	bitoff.14
D0	2	MOVBS	Rw, Rb		F0	2	MOV	Rw, Rw
D1	2	ATOMIC or EXTR	#irang2		F1	2	MOVB	Rb, Rb
D2	4	MOVBS	reg, mem		F2	4	MOV	reg, mem
D3	4	-	-		F3	4	MOVB	reg, mem
D4	4	MOV	Rw, [Rw + #data16]		F4	4	MOVB	Rb, [Rw + #data16]
D5	4	MOVBS	mem, reg		F5	-	-	-
D6	4	SCXT	reg, mem		F6	4	MOV	mem, reg
D7	4	EXTP(R), EXTS(R)	#pag10,#irang2 #seg8, #irang2		F7	4	MOVB	mem, reg
D8	2	MOV	[Rw+], [Rw]		F8	-	-	-
D9	2	MOVB	[Rw+], [Rw]		F9	-	-	-
DA	4	CALLS	seg, caddr		FA	4	JMPS	seg, caddr
DB	2	RETS			FB	2	RETI	
DC	2	EXTP(R), EXTS(R)	Rw, #irang2		FC	2	POP	reg
DD	2	JMPR	cc_SGE, rel		FD	2	JMPR	cc_ULE, rel
DE	2	BCLR	bitoff.13		FE	2	BCLR	bitoff.15
DF	2	BSET	bitoff.13		FF	2	BSET	bitoff.15

5.4 Instruction Description

This chapter describes each instruction in details. The instructions are listed alphabetically, and the description contains the following elements.

- **Instruction Name:** Specifies the mnemonic opcode of the instruction in oversized bold lettering for easy reference. The mnemonics have been chosen with regard to the particular operation performed by the instruction.
- **Syntax:** Specifies the mnemonic opcode and the required formal operands of the instruction as used in the following subsection 'Operation'. There are instructions with either none, one, two or three operands, which must be separated from each other by commas:

MNEMONIC {op1 {,op2 {,op3 } } }

The syntax for the actual operands of an instruction depends on the selected addressing mode. All of the available addressing modes are summarized at the end of each single instruction description. In contrast to the syntax for the instructions described in the following, the assembler provides much more flexibility in writing C166S programs (e.g. by generic instructions and by automatically selecting appropriate addressing modes whenever possible), and thus it eases the use of the instruction set. For more information about this item please refer to the Assembler manual.

- **Operation:** This part presents a logical description of the operation performed by an instruction as a symbolic formula or a high level language construct.

The following symbols are used to represent data movement, arithmetic or logical operators.

Diadic operations:	(opX)		operator (opY)
←	(opY)	is	**MOVED** into (opX)
+	(opX)	is	**ADDED** to (opY)
-	(opY)	is	**SUBTRACTED** from (opX)
*	(opX)	is	**MULTIPLIED** by (opY)
/	(opX)	is	**DIVIDED** by (opY)
∧	(opX)	is	logically **AND**ed with (opY)
∨	(opX)	is	logically **OR**ed with (opY)
⊕	(opX)	is	logically **EXCLUSIVELY OR**ed with (opY)
⇔	(opX)	is	**COMPARED** against (opY)
mod	(opX)	is	divided **MODULO** (opY)

Monadic operations:			**operator (opX)**
¬	(opX)	is	logically **COMPLEMENTED**

Parentheses indicate a method of the used operand addressing as follows:

opX	Specifies the immediate constant value of opX
(opX)	Specifies the contents of opX
(opX[n])	Specifies the contents of bit n of opX
((opX))	Specifies the contents of the contents of opX (ie. opX is used as pointer to the actual operand)

The following operands notation will also be used in the operational description:

CP	Context Pointer
CSP	Code Segment Pointer
IP	Instruction Pointer
MD	Multiply/Divide register (32 bits wide, consists of MDH and MDL)
MDL, MDH	Multiply/Divide Low and High registers (each 16 bit wide)
PSW	Program Status Word
SP	System Stack Pointer
SYSCON	SYSCON Configuration register
C	Carry condition flag in the PSW register
V	Overflow condition flag in the PSW register
SGTDIS	Segmentation Disable bit in the SYSCON register
count	Temporary variable for an intermediate storage of the number of shift or rotate cycles which remain to complete the shift or rotate operation
tmp	Temporary variable for an intermediate result
0, 1, 2,...	Constant values due to the data format of the specified operation

Data Types: This part specifies the particular data type according to the instruction. Basically, the following data types are possible:

BIT, BYTE, WORD

Only instructions which extend byte data to word change data type. Note that the data types mentioned in this subsection do not cover accesses to indirect address pointers or to the system stack. These accesses are always performed with word data. Moreover, no data type is specified for System Control Instructions and for those of the branch instructions which do not access any explicitly addressed data.

- **Description:** This part provides a brief description of the action that is executed by the respective instruction.

- **Condition Code:** The Condition code indicates that respective instruction is executed, if the specified condition exists, and is skipped, if it does not. The table below summarizes the 16 possible condition codes that can be used within Call and Branch instructions. The table shows the abbreviations, the test that is executed for a specific condition and a 4-bit number associated with condition code.

Condition Code Mnemonic cc	Test	Description	Condition Code Number c
cc_UC	1 = 1	Unconditional	0_H
cc_Z	Z = 1	Zero	2_H
cc_NZ	Z = 0	Not zero	3_H
cc_V	V = 1	Overflow	4_H
cc_NV	V = 0	No overflow	5_H
cc_N	N = 1	Negative	6_H
cc_NN	N = 0	Not negative	7_H
cc_C	C = 1	Carry	8_H
cc_NC	C = 0	No carry	9_H
cc_EQ	Z = 1	Equal	2_H
cc_NE	Z = 0	Not equal	3_H
cc_ULT	C = 1	Unsigned less than	8_H
cc_ULE	(Z∨C) = 1	Unsigned less than or equal	F_H
cc_UGE	C = 0	Unsigned greater than or equal	9_H
cc_UGT	(Z∨C) = 0	Unsigned greater than	E_H
cc_SLT	(N⊕V) = 1	Signed less than	C_H
cc_SLE	(Z∨(N⊕V)) = 1	Signed less than or equal	B_H
cc_SGE	(N⊕V) = 0	Signed greater than or equal	D_H

Condition Code Mnemonic cc	Test	Description	Condition Code Number c
cc_SGT	(Z∨(N⊕V)) = 0	Signed greater than	A_H
cc_NET	(Z∨E) = 0	Not equal AND not end of table	1_H

- **Condition Flags:** This part reflects the state of the N, C, V, Z and E flags in the PSW register which is the state after execution of the corresponding instruction, except if the PSW register itself was specified as the destination operand of that instruction (see Note).

The resulting state of the flags is represented by symbols as follows:

'*' The flag is set due to the following standard rules for the corresponding flag:

N = 1 : MSB of the result is set

N = 0 : MSB of the result is not set

C = 1 : Carry occurred during operation

C = 0 : No Carry occurred during operation

V = 1 : Arithmetic Overflow occurred during operation

V = 0 : No Arithmetic Overflow occurred during operation

Z = 1 : Result equals zero

Z = 0 : Result does not equal zero

E = 1 : Source operand represents the lowest negative number (either 8000h for word data or 80h for byte data)

E = 0 : Source operand does not represent the lowest negative number for the specified data type

'S' The flag is set due to rules which deviate from the described standard. For more details see instruction pages (below) or the ALU status flags description.

'-' The flag is not affected by the operation.

'0' The flag is cleared by the operation.

'NOR' The flag contains the logical NORing of the two specified bit operands.

'AND' The flag contains the logical ANDing of the two specified bit operands.

'OR' The flag contains the logical ORing of the two specified bit operands.

'XOR' The flag contains the logical XORing of the two specified bit operands.

'B' The flag contains the original value of the specified bit operand.

'$\overline{B}$' The flag contains the complemented value of the specified bit operand.

Note: If the PSW register was specified as the destination operand of an instruction, the condition flags can not be interpreted as just described, because the PSW register is modified depending on the data format of the instruction as follows:
For word operations, the PSW register is overwritten with the word result. For byte operations, the non-addressed byte is cleared and the addressed byte is overwritten. For bit or bit-field operations on the PSW register, only the specified bits are modified. Supposed that the condition flags were not selected as destination bits, they stay unchanged. This means that they keep the state after execution of the previous instruction.
In any case, if the PSW was the destination operand of an instruction, the PSW flags do NOT represent the condition flags of this instruction as usual.

- **Addressing Modes:** This part specifies which combinations of different addressing modes are available for the required operands. The selected addressing mode combination is mostly specified by the opcode of the corresponding instruction. However, there are some arithmetic and logical instructions where the addressing mode combination is not specified by the (identical) opcodes but by particular bits within the operand field.

The addressing mode entries are made up of three elements:

Mnemonic Shows accepted operands for the respective instruction.

Format This part specifies the format of the instructions as it is represented in the assembler listing. The **Figure 5-1** shows the relation between the instruction format representation of the assembler and the corresponding internal organization of such an instruction format (N = nibble = 4 bits).

The following symbols are used to describe the instruction formats:

00_H through FF_H: Instruction Opcodes

0, 1 : Constant Values

:.... : Each of the 4 characters immediately following a colon represents a single bit

:..ii : 2-bit short GPR address (Rwi)

SS : Code segment number

:..## : 2-bit immediate constant (#irang2)

:.### : 3-bit immediate constant (#data3)

...#:# : 5-bit immediate constant (#data5)

c : 4-bit condition code specification (cc)

n : 4-bit short GPR address (Rwn or Rbn)

m : 4-bit short GPR address (Rwm or Rbm)

q : 4-bit position of the source bit within the word specified by QQ

z : 4-bit position of the destination bit within the word specified by ZZ

: 4-bit immediate constant (#data4)

t:ttt0 : 7-bit trap number (#trap7)

QQ : 8-bit word address of the source bit (bitoff)

rr : 8-bit relative target address word offset (rel)

RR : 8-bit word address reg

ZZ : 8-bit word address of the destination bit (bitoff)

: 8-bit immediate constant (#data8)

xx : 8-bit immediate constant (represented by #data16, byte xx is not significant)

@@ : 8-bit immediate constant (#mask8)

MMMM : 16-bit address (mem or caddr; low byte, high byte)

: 16-bit immediate constant (#data16; low byte, high byte)

Number of Bytes All C166S instructions are either 2 or 4 bytes. According to the instruction size, all instructions can be classified as either single word or double word instructions.

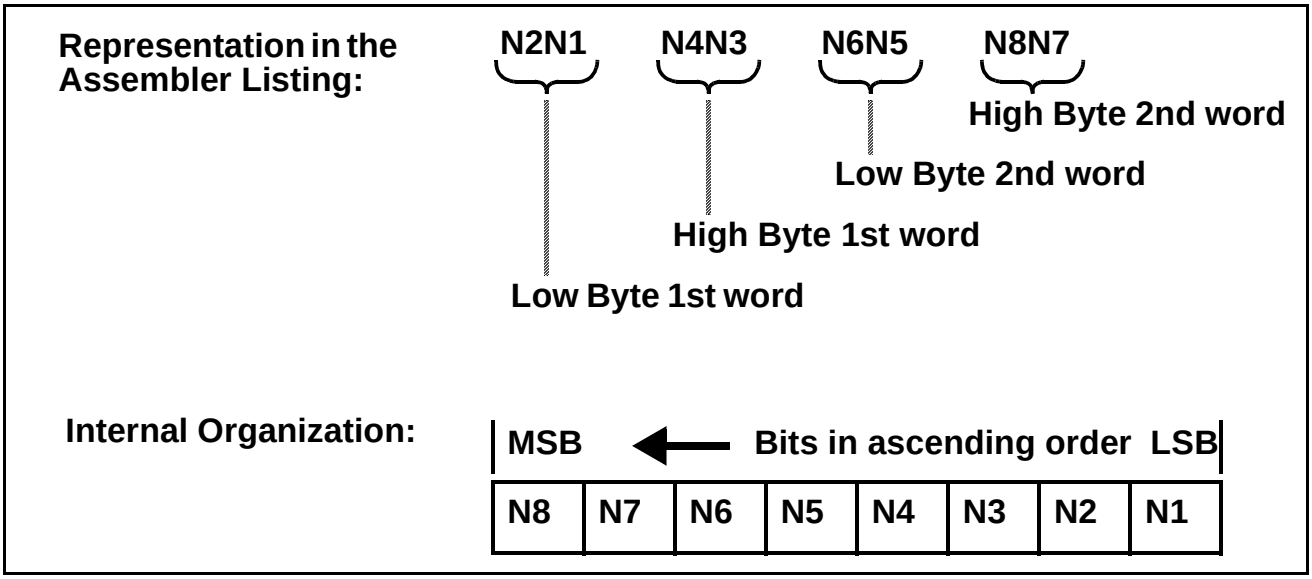


Figure 5-1 Instruction Format Representation

The following pages of this section contain a detailed description of each instruction in alphabetical order.

6 Detailed Instruction Set

The following pages of this section contain a detailed description of each instruction in alphabetical order.

ADD Integer Addition ADD

Group Arithmetic Instructions

Syntax **ADD op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ← (op1) + (op2)

Description

Performs a 2's complement binary addition of the source operand specified by op2 and the destination operand specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
*	*	*	*	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the word data type. Cleared otherwise.
- C Set if a carry is generated from the most significant bit of the word data type. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
ADD	Rw_n , #data3	08 n:0###	2
ADD	Rw_n , Rw_m	00 nm	2
ADD	Rw_n , [Rw_i+]	08 n:11ii	2
ADD	Rw_n , [Rw_i]	08 n:10ii	2
ADD	mem , reg	04 RR MM MM	4
ADD	reg , #data16	06 RR ## ##	4
ADD	reg , mem	02 RR MM MM	4

ADDB Integer Addition ADDB

Group Arithmetic Instructions

Syntax **ADDB op1, op2**

Source Operand(s) op1, op2 → BYTE

Destination Operand(s) op1 → BYTE

Operation

(op1) ← (op1) + (op2)

Description

Performs a 2's complement binary addition of the source operand specified by op2 and the destination operand specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
*	*	*	*	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the byte data type. Cleared otherwise.
- C Set if a carry is generated from the most significant bit of the byte data type. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
ADDB	Rb_n , #data3	09 n:0###	2
ADDB	Rb_n , Rb_m	01 nm	2
ADDB	Rb_n , [Rw_i+]	09 n:11ii	2
ADDB	Rb_n , [Rw_i]	09 n:10ii	2
ADDB	mem , reg	05 RR MM MM	4
ADDB	reg , #data8	07 RR ## xx	4
ADDB	reg , mem	03 RR MM MM	4

ADDC Integer Addition with Carry ADDC

Group Arithmetic Instructions

Syntax **ADDC op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ← (op1) + (op2) + (C)

Description

Performs a 2's complement binary addition of the source operand specified by op2, the destination operand specified by op1 and the previously generated carry bit. The sum is then stored in op1. This instruction can be used to perform multiple precision arithmetic.

CPU Flags

E	Z	V	C	N
*	S	*	*	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero and previous Z flag was set. Cleared otherwise.
- V Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the word data type. Cleared otherwise.
- C Set if a carry is generated from the most significant bit of the word data type. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
ADDC	Rw_n , #data3	18 n:0###	2
ADDC	Rw_n , Rw_m	10 nm	2
ADDC	Rw_n , [Rw_i+]	18 n:11ii	2
ADDC	Rw_n , [Rw_i]	18 n:10ii	2
ADDC	mem , reg	14 RR MM MM	4
ADDC	reg , #data16	16 RR ## ##	4
ADDC	reg , mem	12 RR MM MM	4

ADDCB Integer Addition with Carry ADDCB

Group Arithmetic Instructions

Syntax **ADDCB op1, op2**

Source Operand(s) op1, op2 → BYTE

Destination Operand(s) op1 → BYTE

Operation

(op1) ← (op1) + (op2) + (C)

Description

Performs a 2's complement binary addition of the source operand specified by op2, the destination operand specified by op1 and the previously generated carry bit. The sum is then stored in op1. This instruction can be used to perform multiple precision arithmetic.

CPU Flags

E	Z	V	C	N
*	S	*	*	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero and previous Z flag was set. Cleared otherwise.
- V Set if an arithmetic overflow occurred, i.e. the result cannot be represented in the byte data type. Cleared otherwise.
- C Set if a carry is generated from the most significant bit of the byte data type. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
ADDCB	Rb_n , #data3	19 n:0###	2
ADDCB	Rb_n , Rb_m	11 nm	2
ADDCB	Rb_n , [Rw_i+]	19 n:11ii	2
ADDCB	Rb_n , [Rw_i]	19 n:10ii	2
ADDCB	mem , reg	15 RR MM MM	4
ADDCB	reg , #data8	17 RR ## xx	4
ADDCB	reg , mem	13 RR MM MM	4

AND — Logical AND — AND

Group Logical Instructions

Syntax **AND op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ← (op1) ∧ (op2)

Description

Performs a bitwise logical AND of the source operand specified by op2 and the destination operand specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
*	*	0	0	*

- E — Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z — Set if result equals zero. Cleared otherwise.
- V — Always cleared.
- C — Always cleared.
- N — Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
AND	Rw_n , #data3	68 n:0###	2
AND	Rw_n , Rw_m	60 nm	2
AND	Rw_n , [Rw_i+]	68 n:11ii	2
AND	Rw_n , [Rw_i]	68 n:10ii	2
AND	mem , reg	64 RR MM MM	4
AND	reg , #data16	66 RR ## ##	4
AND	reg , mem	62 RR MM MM	4

ANDB Logical AND ANDB

Group Logical Instructions

Syntax **ANDB op1, op2**

Source Operand(s) op1, op2 → BYTE

Destination Operand(s) op1 → BYTE

Operation

(op1) ← (op1) ∧ (op2)

Description

Performs a bitwise logical AND of the source operand specified by op2 and the destination operand specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
*	*	0	0	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
ANDB	Rb_n , #data3	69 n:0###	2
ANDB	Rb_n , Rb_m	61 nm	2
ANDB	Rb_n , [Rw_i+]	69 n:11ii	2
ANDB	Rb_n , [Rw_i]	69 n:10ii	2
ANDB	mem , reg	65 RR MM MM	4
ANDB	reg , #data8	67 RR ## xx	4
ANDB	reg , mem	63 RR MM MM	4

ASHR Arithmetic Shift Right ASHR

Group Shift and Rotate Instructions

Syntax **ASHR op1, op2**

Source Operand(s) op1 → WORD
op2 → shift counter

Destination Operand(s) op1 → WORD

Operation

```
(count) ← (op2)
(V) ← 0
(C) ← 0
DO WHILE ((count) ≠ 0)
        (V) ← (C) ∨ (V)
        (C) ← (op1[0])
        (op1[n]) ← (op1[n+1]) [n=0...14]
        (count) ← (count) - 1
END WHILE
```

Description

Arithmetically shifts the destination word operand op1 right by the number of times as specified by the source operand op2. To preserve the sign of the original operand op1, the most significant bits of the result are filled with zeros if the original most significant bit was a 0 or with ones if the original most significant bit was a 1. The Overflow flag is used as a Rounding flag. The least significant bit is shifted into the Carry. Only shift values between 0 and 15 are allowed. When using a GPR as the count control, only the least significant 4 bits are used.

CPU Flags

E	Z	V	C	N
0	*	*	*	*

- E Always cleared.
- Z Set if result equals zero. Cleared otherwise.
- V Set if in any cycle of the shift operation a 1 is shifted out of the carry flag. Cleared in case of a shift count equal 0.
- C The carry flag is set according to the last least significant bit shifted out of op1. Cleared for a shift count of zero.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
ASHR	Rw_n , #data4	BC #n	2
ASHR	Rw_n , Rw_m	AC nm	2

ATOMIC Begin ATOMIC Sequence ATOMIC

Group System Control Instructions

Syntax **ATOMIC op1**

Source Operand(s) op1 → 2-bit instruction counter

Destination Operand(s) none

Operation

(count) ← (op1) [1 ≤ op1 ≤ 4]
Disable interrupts and Class A traps
DO WHILE ((count) ≠ 0 AND Class_B_Trap_Condition ≠ TRUE)
    Next Instruction
    (count) ← (count) - 1
END WHILE
(count) ← 0
Enable interrupts and traps

Description

Causes standard and PEC interrupts and class A hardware traps to be disabled for a specified number of instructions. The ATOMIC instruction becomes immediately active. No NOPs are required for normal ATOMIC execution. Depending on the value of op1, the period of validity of the ATOMIC sequence extends over the sequence of the next 1 to 4 instructions being executed after the ATOMIC instruction. All instructions requiring multiple cycles or hold states to be executed are regarded as one instruction in this sense. Any instruction type can be used with the ATOMIC instruction.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
ATOMIC	#irang2	D1 :00##-0	2

BAND Bit Logical AND BAND

Group Boolean Bit Manipulation Instructions

Syntax **BAND op1, op2**

Source Operand(s) op1, op2 → BIT

Destination Operand(s) op1 → BIT

Operation

(op1) ← (op1) ∧ (op2)

Description

Performs a single bit logical AND of the source bit specified by op2 and the destination bit specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
0	NOR	OR	AND	XOR

E Always cleared.
Z Contains the logical NOR of the two specified bits.
V Contains the logical OR of the two specified bits.
C Contains the logical AND of the two specified bits.
N Contains the logical XOR of the two specified bits.

Encoding

Mnemonic		Format	Bytes
BAND	$bitaddr_{Z.z}$, $bitaddr_{Q.q}$	6A QQ ZZ qz	4

BCLR Bit Clear BCLR

Group Boolean Bit Manipulation Instructions

Syntax **BCLR op1**

Source Operand(s) none

Destination Operand(s) op1 → BIT

Operation

(op1) ← 0

Description

Clears the bit specified by op1. This instruction is primarily used for peripheral and system control.

CPU Flags

E	Z	V	C	N
0	$\overline{B}$	0	0	B

E Always cleared.
Z Contains the logical negation of the previous state of the specified bit.
V Always cleared.
C Always cleared.
N Contains the previous state of the specified bit.

Encoding

Mnemonic		Format	Bytes
BCLR	$bitaddr_{Q.q}$	qE QQ	2

BCMP Bit to Bit Compare BCMP

Group Boolean Bit Manipulation Instructions

Syntax **BCMP op1, op2**

Source Operand(s) op1, op2 → BIT

Destination Operand(s) none

Operation

(op1) ⇔ (op2)

Description

Performs a single bit comparison of the source bit specified by op1 and the source bit specified by op2. No result is written by this instruction. Only the flags are updated.

CPU Flags

E	Z	V	C	N
0	NOR	OR	AND	XOR

E Always cleared.
Z Contains the logical NOR of the two specified bits.
V Contains the logical OR of the two specified bits.
C Contains the logical AND of the two specified bits.
N Contains the logical XOR of the two specified bits.

Encoding

Mnemonic		Format	Bytes
BCMP	$bitaddr_{Z.z}$, $bitaddr_{Q.q}$	2A QQ ZZ qz	4

BFLDH Bit Field High Byte BFLDH

Group Boolean Bit Manipulation Instructions

Syntax **BFLDH op1, op2, op3**

Source Operand(s) op1 → WORD
op2, op3 → BYTE

Destination Operand(s) op1 → WORD

Operation

```
(count) ← 0
DO WHILE ((count) <8)
        IF (op2[(count)] = 1)
            (op1[(count) + 8]) ← op3[(count)]
        ENDIF
(count) ← (count) + 1
END WHILE
```

Description
Replaces those bits in the high byte of the destination word operand op1 which are selected by an '1' in the mask specified by op2 with the bits at the corresponding positions in "op3".

CPU Flags

E	Z	V	C	N
0	*	0	0	*

E Always cleared.
Z Set if result equals zero. Cleared otherwise.
V Always cleared.
C Always cleared.
N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
BFLDH	$bitoff_Q$, #mask8 , #data8	1A QQ ## @@	4

BFLDL Bit Field Low Byte BFLDL

Group Boolean Bit Manipulation Instructions

Syntax **BFLDL op1, op2, op3**

Source Operand(s) op1 → WORD
op2, op3 → BYTE

Destination Operand(s) op1 → WORD

Operation

```
(count) ← 0
DO WHILE ((count) <8)
        IF op2[(count)] = 1
            (op1[(count)]) ← op3[(count)]
        ENDIF
(count) ← (count) + 1
END WHILE
```

Description
Replaces those bits in the low byte of the destination word operand op1 which are selected by an '1' in the mask specified by op2 with the bits at the corresponding positions in "op3".

CPU Flags

E	Z	V	C	N
0	*	0	0	*

- E Always cleared.
- Z Set if result equals zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
BFLDL	$bitoff_Q$, #mask8 , #data8	0A QQ @@ ##	4

BMOV Bit to Bit Move BMOV

Group Boolean Bit Manipulation Instructions

Syntax **BMOV op1, op2**

Source Operand(s) op2 → BIT

Destination Operand(s) op1 → BIT

Operation

(op1) ← (op2)

Description

Moves a single bit from the source operand specified by op2 into the destination operand specified by op1. The source bit is examined and the flags are updated accordingly.

CPU Flags

E	Z	V	C	N
0	$\overline{B}$	0	0	B

- E Always cleared.
- Z Contains the logical negation of the source bit.
- V Always cleared.
- C Always cleared.
- N Contains the state of the source bit.

Encoding

Mnemonic		Format	Bytes
BMOV	$bitaddr_{Z.z}$, $bitaddr_{Q.q}$	4A QQ ZZ qz	4

BMOVN Bit to Bit Move and Negate BMOVN

Group Boolean Bit Manipulation Instructions

Syntax **BMOVN op1, op2**

Source Operand(s) op2 → BIT

Destination Operand(s) op1 → BIT

Operation

(op1) ← ¬(op2)

Description

Moves the complement of a single bit from the source operand specified by op2 into the destination operand specified by op1. The source bit is examined and the flags are updated accordingly.

CPU Flags

E	Z	V	C	N
0	$\overline{B}$	0	0	B

- E Always cleared.
- Z Contains the logical negation of the source bit.
- V Always cleared.
- C Always cleared.
- N Contains the state of the source bit.

Encoding

Mnemonic		Format	Bytes
BMOVN	$bitaddr_{Z.z}$, $bitaddr_{Q.q}$	3A QQ ZZ qz	4

BOR Bit Logical OR BOR

Group Boolean Bit Manipulation Instructions

Syntax **BOR op1, op2**

Source Operand(s) op1, op2 → BIT

Destination Operand(s) op1 → BIT

Operation

(op1) ← (op1) ∨ (op2)

Description

Performs a single bit logical OR of the source bit specified by op2 and the destination bit specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
0	NOR	OR	AND	XOR

E Always cleared.
Z Contains the logical NOR of the two specified bits.
V Contains the logical OR of the two specified bits.
C Contains the logical AND of the two specified bits.
N Contains the logical XOR of the two specified bits.

Encoding

Mnemonic		Format	Bytes
BOR	$bitaddr_{Z.z}$, $bitaddr_{Q.q}$	5A QQ ZZ qz	4

BSET Bit Set BSET

Group Boolean Bit Manipulation Instructions

Syntax **BSET op1**

Source Operand(s) none

Destination Operand(s) op1 → BIT

Operation

(op1) ← 1

Description

Sets the bit specified by op1.

CPU Flags

E	Z	V	C	N
0	$\overline{B}$	0	0	B

E Always cleared.
Z Contains the logical negation of the previous state of the specified bit.
V Always cleared.
C Always cleared.
N Contains the previous state of the specified bit.

Encoding

Mnemonic		Format	Bytes
BSET	$bitaddr_{Q.q}$	qF QQ	2

BXOR Bit Logical XOR BXOR

Group Boolean Bit Manipulation Instructions

Syntax **BXOR op1, op2**

Source Operand(s) op1, op2 → BIT

Destination Operand(s) op1 → BIT

Operation

(op1) ← (op1) ⊕ (op2)

Description

Performs a single bit logical EXCLUSIVE OR of the source bit specified by op2 and the destination bit specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
0	NOR	OR	AND	XOR

E Always cleared.
Z Contains the logical NOR of the two specified bits.
V Contains the logical OR of the two specified bits.
C Contains the logical AND of the two specified bits.
N Contains the logical XOR of the two specified bits.

Encoding

Mnemonic		Format	Bytes
BXOR	$bitaddr_{Z.z}$, $bitaddr_{Q.q}$	7A QQ ZZ qz	4

CALLA Call Subroutine Absolute CALLA

Group Call Instructions

Syntax **CALLA op1, op2**

Source Operand(s) op1 → extended condition code
op2 → 16-bit address offset

Destination Operand(s) none

Operation

```
IF (op1) THEN
        (SP) ← (SP) - 2
        ((SP)) ← (IP)
        (IP) ← op2
ELSE
        next instruction
END IF
```

Description

If the condition specified by op1 is met, a branch to the absolute memory location specified by the second operand op2 is taken. The value of the instruction pointer IP is placed into the system stack. Because the IP always points to the instruction following the branch instruction, the value stored in the system stack represents the return address of the calling routine. If the condition is not met, no action is taken and the next instruction is executed normally.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
CALLA	xcc , caddr	CA d00a MM MM	4

CALLI Call Subroutine Indirect CALLI

Group Call Instructions

Syntax **CALLI op1, op2**

Source Operand(s) op1 → condition code
op2 → 16-bit address offset

Destination Operand(s) none

Operation

```
IF (op1) THEN
        (SP) ← (SP) - 2
        ((SP)) ← (IP)
        (IP) ← op2
ELSE
        next instruction
END IF
```

Description

If the condition specified by op1 is met, a branch to the location specified indirectly by the second operand op2 is taken. The value of the instruction pointer IP is placed onto the system stack. Because the IP always points to the instruction following the branch instruction, the value stored in the system stack represents the return address of the calling routine. If the condition is not met, no action is taken and the next instruction is executed normally.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
CALLI	cc , [Rw_n]	AB cn	2

CALLR Call Subroutine Relative CALLR

Group Call Instructions

Syntax **CALLR op1**

Source Operand(s) op1 → 8-bit signed displacement

Destination Operand(s) none

Operation

(SP) ← (SP) - 2
((SP)) ← (IP)
(IP) ← (IP) + 2*sign_extend(op1)

Description

A branch is taken to the location specified by the instruction pointer IP plus the relative displacement op1. The displacement is a two's complement number which is sign extended and counts the relative distance in **words**. The value of the instruction pointer (IP) is placed into the system stack. Because the IP always points to the instruction following the branch instruction, the value stored in the system stack represents the return address of the calling routine. The value of the IP used in the target address calculation is the address of the instruction following the CALLR instruction.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
CALLR	rel	BB rr	2

CALLS Call Inter-Segment Subroutine CALLS

Group Call Instructions

Syntax **CALLS op1, op2**

Source Operand(s) op1 → segment number
op2 → 16-bit address offset

Destination Operand(s) none

Operation

(SP) ← (SP) - 2
((SP)) ← (CSP)
(SP) ← (SP) - 2
((SP)) ← (IP)
(CSP) ← op1
(IP) ← op2

Description

A branch is taken to the absolute location specified by op2 within the segment specified by op1. The previous value of the CSP is placed into the system stack to insure correct return to the calling segment. The value of the instruction pointer (IP) is also placed into the system stack. Because the IP always points to the instruction following the branch instruction, the value stored on the system stack represents the return address to the calling routine.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
CALLS	seg , caddr	DA SS MM MM	4

CMP Integer Compare CMP

Group Boolean Bit Manipulation Instructions

Syntax **CMP op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) none

Operation

(op1) ⇔ (op2)

Description

The source operand specified by op1 is compared to the source operand specified by op2 by performing a 2's complement binary subtraction of op2 from op1. The flags are set according to the rules of subtraction. The operands remain unchanged.

CPU Flags

E	Z	V	C	N
*	*	*	S	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the word data type. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CMP	Rw_n , #data3	48 n:0###	2
CMP	Rw_n , Rw_m	40 nm	2
CMP	Rw_n , [Rw_i+]	48 n:11ii	2
CMP	Rw_n , [Rw_i]	48 n:10ii	2
CMP	reg , #data16	46 RR ## ##	4
CMP	reg , mem	42 RR MM MM	4

CMPB Integer Compare CMPB

Group Boolean Bit Manipulation Instructions

Syntax **CMPB op1, op2**

Source Operand(s) op1, op2 → BYTE

Destination Operand(s) none

Operation

(op1) ⇔ (op2)

Description

The source operand specified by op1 is compared to the source operand specified by op2 by performing a 2's complement binary subtraction of op2 from op1. The flags are set according to the rules of subtraction. The operands remain unchanged.

CPU Flags

E	Z	V	C	N
*	*	*	S	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the byte data type. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CMPB	Rb_n , #data3	49 n:0###	2
CMPB	Rb_n , Rb_m	41 nm	2
CMPB	Rb_n , [Rw_i+]	49 n:11ii	2
CMPB	Rb_n , [Rw_i]	49 n:10ii	2
CMPB	reg , #data8	47 RR ## xx	4
CMPB	reg , mem	43 RR MM MM	4

CMPD1 Integer Compare and Decrement by 1 CMPD1

Group Compare and Loop Control Instructions

Syntax **CMPD1 op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ⇔ (op2)
(op1) ← (op1) - 1

Description

This instruction is used to enhance the performance and flexibility of loops. The source operand specified by op1 is compared to the source operand specified by op2 by performing a 2's complement binary subtraction of op2 from op1. Operand op1 may specify ONLY GPR registers. Once the subtraction has completed, the operand op1 is decremented by one. Using the set flags, a branch instruction can then be used in conjunction with this instruction to form common high level language FOR loops of any range.

CPU Flags

E	Z	V	C	N
*	*	*	S	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the word data type. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CMPD1	Rw_n , #data16	A6 Fn ## ##	4
CMPD1	Rw_n , #data4	A0 #n	2
CMPD1	Rw_n , mem	A2 Fn MM MM	4

CMPD2 Integer Compare and Decrement by 2 CMPD2

Group Compare and Loop Control Instructions

Syntax **CMPD2 op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ⇔ (op2)
(op1) ← (op1) - 2

Description

This instruction is used to enhance the performance and flexibility of loops. The source operand specified by op1 is compared to the source operand specified by op2 by performing a 2's complement binary subtraction of op2 from op1. Operand op1 may specify ONLY GPR registers. Once the subtraction has completed, the operand op1 is decremented by two. Using the set flags, a branch instruction can then be used in conjunction with this instruction to form common high level language FOR loops of any range.

CPU Flags

E	Z	V	C	N
*	*	*	S	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the word data type. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CMPD2	Rw_n , #data16	B6 Fn ## ##	4
CMPD2	Rw_n , #data4	B0 #n	2
CMPD2	Rw_n , mem	B2 Fn MM MM	4

CMPI1 Integer Compare and Increment by 1 CMPI1

Group Compare and Loop Control Instructions

Syntax **CMPI1 op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ⇔ (op2)
(op1) ← (op1) + 1

Description

This instruction is used to enhance the performance and flexibility of loops. The source operand specified by op1 is compared to the source operand specified by op2 by performing a 2's complement binary subtraction of op2 from op1. Operand op1 may specify ONLY GPR registers. Once the subtraction has completed, the operand op1 is incremented by one. Using the set flags, a branch instruction can then be used in conjunction with this instruction to form common high level language FOR loops of any range.

CPU Flags

E	Z	V	C	N
*	*	*	S	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the word data type. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CMPI1	Rw_n , #data16	86 Fn ## ##	4
CMPI1	Rw_n , #data4	80 #n	2
CMPI1	Rw_n , mem	82 Fn MM MM	4

CMPI2 Integer Compare and Increment by 2 CMPI2

Group Compare and Loop Control Instructions

Syntax **CMPI2 op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ⇔ (op2)
(op1) ← (op1) + 2

Description

This instruction is used to enhance the performance and flexibility of loops. The source operand specified by op1 is compared to the source operand specified by op2 by performing a 2's complement binary subtraction of op2 from op1. Operand op1 may specify ONLY GPR registers. Once the subtraction has completed, the operand op1 is incremented by two. Using the set flags, a branch instruction can then be used in conjunction with this instruction to form common high level language FOR loops of any range.

CPU Flags

E	Z	V	C	N
*	*	*	S	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the word data type. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CMPI2	Rw_n , #data16	96 Fn ## ##	4
CMPI2	Rw_n , #data4	90 #n	2
CMPI2	Rw_n , mem	92 Fn MM MM	4

CPL Integer One's Complement CPL

Group Arithmetic Instructions

Syntax **CPL op1**

Source Operand(s) op1 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ← ¬(op1)

Description

Performs a 1's complement of the source operand specified by op1. The result is stored back into op1.

CPU Flags

E	Z	V	C	N
*	*	0	0	*

- E Set if the value of op1 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CPL	Rw_n	91 n0	2

CPLB Integer One's Complement CPLB

Group Arithmetic Instructions

Syntax **CPLB op1**

Source Operand(s) op1 → BYTE

Destination Operand(s) op1 → BYTE

Operation

(op1) ← ¬(op1)

Description

Performs a 1's complement of the source operand specified by op1. The result is stored back into op1.

CPU Flags

E	Z	V	C	N
*	*	0	0	*

- E Set if the value of op1 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
CPLB	Rb_n	B1 n0	2

DISWDT Disable Watchdog Timer DISWDT

Group System Control Instructions

Syntax **DISWDT**

Source Operand(s) none

Destination Operand(s) none

Operation

Disable the watchdog timer

Description

This instruction disables the watchdog timer. If the WDTCTL bit is cleared, the DISWDT instruction can be executed at any time between the Reset and the first execution of either EINIT or SRVWDT. Once either an EINIT or a SRVWDT has been executed, the DISWDT instruction will have no effect. If the WDTCTL bit is set, the DISWDT instruction can always be executed regardless of the execution of EINIT or SRVWDT. To insure that this instruction is not accidentally executed, it is implemented as a protected instruction.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic	Format	Bytes
DISWDT	A5 5A A5 A5	4

DIV 16-by-16 Signed Division DIV

Group Arithmetic Instructions

Syntax **DIV op1**

Source Operand(s) op1 → WORD
MDL → WORD

Destination Operand(s) MD → DOUBLEWORD

Operation

(MDL) ← (MDL) / (op1)
(MDH) ← (MDL) mod (op1)

Description

Performs a signed 16-bit by 16-bit division of the low order word stored in the MD register by the source word operand op1. The signed quotient is then stored in the low order word of the MD register (MDL) and the remainder is stored in the high order word of the MD register (MDH).

CPU Flags

E	Z	V	C	N
0	*	*	0	*

- E Always cleared.
- Z Set if quotient, stored in the MDL register, equals zero. Cleared otherwise. Undefined if the V flag is set.
- V Set if an arithmetic overflow occurred, i.e. the quotient cannot be represented in a word data type, or if the divisor op1 was zero. Cleared otherwise.
- C Always cleared.
- N Set if the most significant bit of the quotient, stored in the MDL register, is set. Cleared otherwise. Undefined if the V flag is set.

Encoding

Mnemonic		Format	Bytes
DIV	Rw_n	4B nn	2

DIVL 32-by-16 Signed Division DIVL

Group Arithmetic Instructions

Syntax **DIVL op1**

Source Operand(s) op1 → WORD
MD → DOUBLEWORD

Destination Operand(s) MD → DOUBLEWORD

Operation

(MDL) ← (MD) / (op1)
(MDH) ← (MD) mod (op1)

Description

Performs an extended signed 32-bit by 16-bit division of the two words stored in the MD register by the source word operand op1. The signed quotient is then stored in the low order word of the MD register (MDL) and the remainder is stored in the high order word of the MD register (MDH).

CPU Flags

E	Z	V	C	N
0	*	*	0	*

- E Always cleared.
- Z Set if quotient, stored in the MDL register, equals zero. Cleared otherwise. Undefined if the V flag is set.
- V Set if an arithmetic overflow occurred, i.e. the quotient cannot be represented in a word data type, or if the divisor op1 was zero. Cleared otherwise.
- C Always cleared.
- N Set if the most significant bit of the quotient, stored in the MDL register, is set. Cleared otherwise. Undefined if the V flag is set.

Encoding

Mnemonic		Format	Bytes
DIVL	Rw_n	6B nn	2

DIVLU 32-by-16 Unsigned Division DIVLU

Group Arithmetic Instructions

Syntax **DIVLU op1**

Source Operand(s) op1 → WORD
MD → DOUBLEWORD

Destination Operand(s) MD → DOUBLEWORD

Operation

(MDL) ← (MD) / op1
(MDH) ← (MD) mod (op1)

Description

Performs an extended unsigned 32-bit by 16-bit division of the two words stored in the MD register by the source word operand op1. The unsigned quotient is then stored in the low order word of the MD register (MDL) and the remainder is stored in the high order word of the MD register (MDH).

CPU Flags

E	Z	V	C	N
0	*	*	0	*

- E Always cleared.
- Z Set if quotient, stored in the MDL register, equals zero. Cleared otherwise. Undefined if the V flag is set.
- V Set if an arithmetic overflow occurred, i.e. the quotient cannot be represented in a word data type, or if the divisor op1 was zero. Cleared otherwise.
- C Always cleared.
- N Set if the most significant bit of the quotient, stored in the MDL register, is set. Cleared otherwise. Undefined if the V flag is set.

Encoding

Mnemonic		Format	Bytes
DIVLU	Rw_n	7B nn	2

DIVU 16-by-16 Unsigned Division DIVU

Group Arithmetic Instructions

Syntax **DIVU op1**

Source Operand(s) op1 → WORD
MDL → WORD

Destination Operand(s) MD → DOUBLEWORD

Operation

(MDL) ← (MDL) / (op1)
(MDH) ← (MDL) mod (op1)

Description

Performs an unsigned 16-bit by 16-bit division of the low order word stored in the MD register by the source word operand op1. The unsigned quotient is then stored in the low order word of the MD register (MDL) and the remainder is stored in the high order word of the MD register (MDH).

CPU Flags

E	Z	V	C	N
0	*	*	0	*

- E Always cleared.
- Z Set if quotient, stored in the MDL register, equals zero. Cleared otherwise. Undefined if the V flag is set.
- V Set if the divisor op1 was zero.
- C Always cleared.
- N Set if the most significant bit of the quotient, stored in the MDL register, is set. Cleared otherwise. Undefined if the V flag is set.

Encoding

Mnemonic		Format	Bytes
DIVU	Rw_n	5B nn	2

EINIT End of Initialization EINIT

Group System Control Instructions

Syntax **EINIT**

Source Operand(s) none

Destination Operand(s) none

Operation

End of Initialization

Description

After a reset, the reset output pin RSTOUT is pulled low. It remains low until the EINIT instruction has been executed at which time it goes high. This enables the software to signal the external circuitry that it has successfully initialized the microcontroller. Execution of the Disable Watchdog Timer (DISWDT) instruction after the execution of the EINIT instruction has no effect. To insure that this instruction is not accidentally executed, it is implemented as a protected instruction.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic	Format	Bytes
EINIT	B5 4A B5 B5	4

EXTP Begin EXTended Page Sequence EXTP

Group System Control Instructions

Syntax **EXTP op1, op2**

Source Operand(s) op1 → 10-bit page number
op2 → 2-bit instruction counter

Destination Operand(s) none

Operation

(count) ← (op2) [1 ≤ op2 ≤ 4]
Disable interrupts and Class A traps
Data_Page ← (op1)
DO WHILE ((count) ≠ 0 AND Class_B_Trap_Condition ≠ TRUE)
    Next Instruction
    (count) ← (count) - 1
END WHILE
(count) ← 0
Data_Page ← (DPPx)
Enable interrupts and traps

Description

Overrides the standard DPP addressing scheme of the long and indirect addressing modes for a specified number of instructions. During their execution, both standard and PEC interrupts and class A hardware traps are locked. The EXTP instruction becomes immediately active such that no additional NOPs are required. For any long ('mem') or indirect ([...]) address in the EXTP instruction sequence, the 10-bit page number (address bits A23-A14) is not determined by the contents of a DPP register but by the value of op1 itself. The 14-bit page offset (address bits A13-A0) is derived from the long or indirect address as usual. The value of op2 defines the length of the effected instruction sequence.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
EXTP	#pag , #irang2	D7 :01##-0 pp 0:00pp	4
EXTP	Rw_m , #irang2	DC :01##-m	2

EXTPR Begin EXTended Page and Register Sequence EXTPR

Group System Control Instructions

Syntax **EXTPR op1, op2**

Source Operand(s) op1 → 10-bit page number
op2 → 2-bit instruction counter

Destination Operand(s) none

Operation

(count) ← (op2) [1 ≤ op2 ≤ 4]
Disable interrupts and Class A traps
Data_Page ← (op1)
SFR_range ← Extended
DO WHILE ((count) ≠ 0 AND Class_B_Trap_Condition ≠ TRUE)
    Next Instruction
    (count) ← (count) - 1
END WHILE
(count) ← 0
Data_Page ← (DPPx)
SFR_range ← Standard
Enable interrupts and traps

Description

Overrides the standard DPP addressing scheme of the long and indirect addressing modes and causes all SFR or SFR bit accesses via the 'reg', 'bitoff' or 'bitaddr' addressing modes being made to the Extended SFR space for a specified number of instructions. During their execution, both standard and PEC interrupts and class A hardware traps are locked. For any long ('mem') or indirect ([...]) address in the EXTP instruction sequence, the 10-bit page number (address bits A23-A14) is not determined by the contents of a DPP register but by the value of op1 itself. The 14-bit page offset (address bits A13-A0) is derived from the long or indirect address as usual. The value of op2 defines the length of the effected instruction sequence.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.

C	Not affected.
N	Not affected.

Encoding

Mnemonic		Format	Bytes
EXTPR	#pag , #irang2	D7 :11##-0 pp 0:00pp	4
EXTPR	Rw_m , #irang2	DC :11##-m	2

EXTR Begin EXTended Register Sequence EXTR

Group System Control Instructions

Syntax **EXTR op1**

Source Operand(s) op1 → 2-bit instruction counter

Destination Operand(s) none

Operation

(count) ← (op1) [1 ≤ op1 ≤ 4]
Disable interrupts and Class A traps
SFR_range ← Extended
DO WHILE ((count) ≠ 0 AND Class_B_Trap_Condition ≠ TRUE)
    Next Instruction
    (count) ← (count) - 1
END WHILE
(count) ← 0
SFR_range ← Standard
Enable interrupts and traps

Description

Causes all SFR or SFR bit accesses via the 'reg', 'bitoff' or 'bitaddr' addressing modes being made to the Extended SFR space for a specified number of instructions. During their execution, both standard and PEC interrupts and class A hardware traps are locked. The value of op1 defines the length of the effected instruction sequence.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
EXTR	#irang2	D1 :10##-0	2

EXTS Begin EXTended Segment Sequence EXTS

Group System Control Instructions

Syntax **EXTS op1, op2**

Source Operand(s) op1 → segment number
op2 → 2-bit instruction counter

Destination Operand(s) none

Operation

(count) ← (op2) [1 ≤ op2 ≤ 4]
Disable interrupts and Class A traps
Data_Segment ← (op1)
DO WHILE ((count) ≠ 0 AND Class_B_Trap_Condition ≠ TRUE)
    Next Instruction
    (count) ← (count) - 1
END WHILE
(count) ← 0
Data_Page ← (DPPx)
Enable interrupts and traps

Description

Overrides the standard DPP addressing scheme of the long and indirect addressing modes for a specified number of instructions. During their execution, both standard and PEC interrupts and class A hardware traps are locked. The EXTS instruction becomes immediately active such that no additional NOPs are required. For any long ('mem') or indirect ([...]) address in an EXTS instruction sequence, the value of op1 determines the 8-bit segment (address bits A23-A16) valid for the corresponding data access. The long or indirect address itself represents the 16-bit segment offset (address bits A15-A0). The value of op2 defines the length of the effected instruction sequence.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
EXTS	#seg , #irang2	D7 :00##-0 ss 00	4
EXTS	Rw_m , #irang2	DC :00##-m	2

EXTSR Begin EXTended Segment and Register Sequence EXTSR

Group System Control Instructions

Syntax **EXTSR op1, op2**

Source Operand(s) op1 → segment number
op2 → 2-bit instruction counter

Destination Operand(s) none

Operation

(count) ← (op2) [1 ≤ op2 ≤ 4]
Disable interrupts and Class A traps
Data_Segment ← (op1)
SFR_range ← Extended
DO WHILE ((count) ≠ 0 AND Class_B_Trap_Condition ≠ TRUE)
    Next Instruction
    (count) ← (count) - 1
END WHILE
(count) ← 0
Data_Page ← (DPPx)
SFR_range ← Standard
Enable interrupts and traps

Description

Overrides the standard DPP addressing scheme of the long and indirect addressing modes and causes all SFR or SFR bit accesses via the 'reg', 'bitoff' or 'bitaddr' addressing modes being made to the Extended SFR space for a specified number of instructions. During their execution, both standard and PEC interrupts and class A hardware traps are locked. The EXTSR instruction becomes immediately active such that no additional NOPs are required. For any long ('mem') or indirect ([...]) address in an EXTSR instruction sequence, the value of op1 determines the 8-bit segment (address bits A23-A16) valid for the corresponding data access. The long or indirect address itself represents the 16-bit segment offset (address bits A15-A0). The value of op2 defines the length of the effected instruction sequence.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.

V	Not affected.
C	Not affected.
N	Not affected.

Encoding

Mnemonic		Format	Bytes
EXTSR	#seg , #irang2	D7 :10##-0 ss 00	4
EXTSR	Rw_m , #irang2	DC :10##-m	2

IDLE Enter Idle Mode IDLE

Group System Control Instructions

Syntax **IDLE**

Source Operand(s) none

Destination Operand(s) none

Operation

Enter Idle Mode

Description

This instruction causes the part to enter the idle mode. In this mode, the CPU is powered down while the peripherals remain running. It remains powered down until a peripheral interrupt or external interrupt occurs. To insure that this instruction is not accidentally executed, it is implemented as a protected instruction.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic	Format	Bytes
IDLE	87 78 87 87	4

JB Relative Jump if Bit Set JB

Group Jump Instructions

Syntax **JB op1, op2**

Source Operand(s) op1 → BIT
op2 → 8-bit signed displacement

Destination Operand(s) none

Operation

```
IF ((op1) = 1) THEN
        (IP) ← (IP) + 2*sign_extend(op2)
ELSE
        Next Instruction
END IF
```

Description

If the bit specified by op1 is set, program execution continues at the location of the instruction pointer IP, plus the specified displacement op2. The displacement is a two's complement number which is sign extended and counts the relative distance in **words**. The value of the IP used in the target address calculation is the address of the instruction following the JB instruction. If the specified bit is cleared, program execution continues normally with the instruction following the JB instruction.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
JB	$bitaddr_{Q.q}$, rel	8A QQ rr q0	4

JBC Relative Jump if Bit Set and Clear Bit JBC

Group Jump Instructions

Syntax **JBC op1, op2**

Source Operand(s) op1 → BIT
op2 → 8-bit signed displacement

Destination Operand(s) none

Operation

```
IF ((op1) = 1) THEN
        (op1) ← 0
        (IP) ← (IP) + 2*sign_extend(op2)
ELSE
        Next Instruction
END IF
```

Description

If the bit specified by op1 is set, program execution continues at the location of the instruction pointer IP, plus the specified displacement op2. The bit specified by op1 is cleared, allowing implementation of semaphore operations. The displacement is a two's complement number which is sign extended and counts the relative distance in **words**. The value of the IP used in the target address calculation is the address of the instruction following the JBC instruction. If the specified bit was clear, program execution continues normally with the instruction following the JBC instruction.

Note Flags are always updated by this instruction.

CPU Flags

E	Z	V	C	N
0	$\overline{B}$	0	0	B

E Always cleared.
Z Contains the logical negation of the previous state of the specified bit.
V Always cleared.
C Always cleared.
N Contains the previous state of the specified bit.

Encoding

Mnemonic		Format	Bytes
JBC	$bitaddr_{Q.q}$, rel	AA QQ rr q0	4

JMPA Absolute Conditional Jump JMPA

Group Jump Instructions

Syntax **JMPA op1, op2**

Source Operand(s) op1 → extended condition code
op2 → 16-bit address offset

Destination Operand(s) none

Operation

```
IF ((op1) = 1) THEN
        (IP) ← op2
ELSE
        Next Instruction
END IF
```

Description

If the condition specified by op1 is met, a branch to the absolute address specified by op2 is taken. If the condition is not met, no action is taken, and the instruction following the JMPA instruction is executed normally.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
JMPA	xcc , caddr	EA d0la MM MM	4

JMPI Indirect Conditional Jump JMPI

Group Jump Instructions

Syntax **JMPI op1, op2**

Source Operand(s) op1 → condition code
op2 → 16-bit address offset

Destination Operand(s) none

Operation

```
IF ((op1) = 1) THEN
        (IP) ← (op2)
ELSE
        Next Instruction
END IF
```

Description

If the condition specified by op1 is met, a branch to the absolute address specified by op2 is taken. If the condition is not met, no action is taken, and program execution continues normally with the instruction following the JMPI instruction.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
JMPI	cc , [Rw_n]	9C cn	2

JMPR Relative Conditional Jump JMPR

Group Jump Instructions

Syntax **JMPR op1, op2**

Source Operand(s) op1 → condition code
op2 → 8-bit signed displacement

Destination Operand(s) none

Operation

```
IF ((op1) = 1) THEN
        (IP) ← (IP) + 2*sign_extend(op2)
ELSE
        Next Instruction
END IF
```

Description

If the extended condition specified by op1 is met, program execution continues at the location of the instruction pointer, IP, plus the specified displacement, op2. The displacement is a two's complement number which is sign-extended and counts the relative distance in words. The value of the IP used in the target address calculation is the address of the instruction following the JMPR instruction. If the specified condition is not met, program execution continues normally with the instruction following the JMPR instruction.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
JMPR	cc , rel	cD rr	2

JMPS Absolute Inter-Segment Jump JMPS

Group Jump Instructions

Syntax **JMPS op1, op2**

Source Operand(s) op1 → segment number
op2 → 16-bit address offset

Destination Operand(s) none

Operation

(CSP) ← op1
(IP) ← op2

Description

Branches unconditionally to the absolute address specified by op2 within the segment specified by op1.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
JMPS	seg , caddr	FA SS MM MM	4

JNB Relative Jump if Bit Clear JNB

Group Jump Instructions

Syntax **JNB op1, op2**

Source Operand(s) op1 → BIT
op2 → 8-bit signed displacement

Destination Operand(s) none

Operation

```
IF ((op1) = 0) THEN
        (IP) ← (IP) + 2*sign_extend(op2)
ELSE
        Next Instruction
END IF
```

Description

If the bit specified by op1 is clear, program execution continues at the location of the instruction pointer IP, plus the specified displacement op2. The displacement is a two's complement number which is sign-extended and counts the relative distance in words. The value of the IP used in the target address calculation is the address of the instruction following the JNB instruction. If the specified bit is set, program execution continues normally with the instruction following the JNB instruction.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
JNB	$bitaddr_{Q.q}$, rel	9A QQ rr q0	4

JNBS Relative Jump if Bit Clear and Set Bit JNBS

Group Jump Instructions

Syntax **JNBS op1, op2**

Source Operand(s) op1 → BIT
op2 → 8-bit signed displacement

Destination Operand(s) none

Operation

```
IF ((op1) = 0) THEN
        (op1) ← 1
        (IP) ← (IP) + 2*sign_extend(op2)
ELSE
        Next Instruction
END IF
```

Description

If the bit specified by op1 is clear, program execution continues at the location of the instruction pointer IP, plus the specified displacement op2. The bit specified by op1 is set, allowing implementation of semaphore operations. The displacement is a two's complement number which is sign-extended and counts the relative distance in words. The value of the IP used in the target address calculation is the address of the instruction following the JNBS instruction. If the specified bit was set, program execution continues normally with the instruction following the JNBS instruction.

Note Flags are always updated by this instruction.

CPU Flags

E	Z	V	C	N
0	B	0	0	B

E Always cleared.
Z Contains the logical negation of the previous state of the specified bit.
V Always cleared.
C Always cleared.
N Contains the previous state of the specified bit.

Encoding

Mnemonic		Format	Bytes
JNBS	$bitaddr_{Q.q}$, rel	BA QQ rr q0	4

MOV Move Data MOV

Group Data Movement Instructions

Syntax **MOV op1, op2**

Source Operand(s) op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ← (op2)

Description

Moves the contents of the source operand specified by op2 to the location specified by the destination operand op1. The contents of the moved data is examined, and the flags are updated accordingly.

CPU Flags

E	Z	V	C	N
*	*	-	-	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if the value of the source operand op2 equals zero. Cleared otherwise.
- V Not affected.
- C Not affected.
- N Set if the most significant bit of the source operand op2 is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
MOV	Rw_n , #data4	E0 #n	2
MOV	Rw_n , Rw_m	F0 nm	2
MOV	Rw_n , [Rw_m+#data16]	D4 nm ## ##	4
MOV	Rw_n , [Rw_m+]	98 nm	2
MOV	Rw_n , [Rw_m]	A8 nm	2
MOV	[-Rw_m] , Rw_n	88 nm	2
MOV	[Rw_m+#data16] , Rw_n	C4 nm ## ##	4

MOV	[Rw_m] , Rw_n	B8 nm	2
MOV	[Rw_n+] , [Rw_m]	D8 nm	2
MOV	[Rw_n] , [Rw_m+]	E8 nm	2
MOV	[Rw_n] , [Rw_m]	C8 nm	2
MOV	[Rw_n] , mem	84 0n MM MM	4
MOV	mem , [Rw_n]	94 0n MM MM	4
MOV	mem , reg	F6 RR MM MM	4
MOV	reg , #data16	E6 RR ## ##	4
MOV	reg , mem	F2 RR MM MM	4

MOVB Move Data MOVB

Group Data Movement Instructions

Syntax **MOVB op1, op2**

Source Operand(s) op2 → BYTE

Destination Operand(s) op1 → BYTE

Operation

(op1) ← (op2)

Description

Moves the contents of the source operand specified by op2 to the location specified by the destination operand op1. The contents of the moved data is examined, and the flags are updated accordingly.

CPU Flags

E	Z	V	C	N
*	*	-	-	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if the value of the source operand op2 equals zero. Cleared otherwise.
- V Not affected.
- C Not affected.
- N Set if the most significant bit of the source operand op2 is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
MOVB	Rb_n , #data4	E1 #n	2
MOVB	Rb_n , Rb_m	F1 nm	2
MOVB	Rb_n , [Rw_m + #data16]	F4 nm ## ##	4
MOVB	Rb_n , [Rw_m+]	99 nm	2
MOVB	Rb_n , [Rw_m]	A9 nm	2
MOVB	[-Rw_m] , Rb_n	89 nm	2
MOVB	[Rw_m + #data16] , Rb_n	E4 nm ## ##	4

MOVB	[Rw_m] , Rb_n	B9 nm	2
MOVB	[Rw_n+] , [Rw_m]	D9 nm	2
MOVB	[Rw_n] , [Rw_m+]	E9 nm	2
MOVB	[Rw_n] , [Rw_m]	C9 nm	2
MOVB	[Rw_n] , mem	A4 0n MM MM	4
MOVB	mem , [Rw_n]	B4 0n MM MM	4
MOVB	mem , reg	F7 RR MM MM	4
MOVB	reg , #data8	E7 RR ## xx	4
MOVB	reg , mem	F3 RR MM MM	4

MOVBS Move Byte Sign Extend MOVBS

Group Data Movement Instructions

Syntax **MOVBS op1, op2**

Source Operand(s) op2 → BYTE

Destination Operand(s) op1 → WORD

Operation

```
(low byte op1) ← (op2)
IF ((op2[7]) = 1) THEN
        (high byte op1) ← FFH
ELSE
        (high byte op1) ← 00H
END IF
```

Description

Moves and sign-extends the contents of the source byte operand specified by op2 to the word location specified by the destination operand op1. The contents of the moved data is examined, and the flags are updated accordingly.

CPU Flags

E	Z	V	C	N
0	*	-	-	*

- E Always cleared.
- Z Set if the value of the source byte operand op2 equals zero. Cleared otherwise.
- V Not affected.
- C Not affected.
- N Set if the most significant bit of the source operand op2 is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
MOVBS	Rw_n , Rb_m	D0 mn	2
MOVBS	mem , reg	D5 RR MM MM	4
MOVBS	reg , mem	D2 RR MM MM	4

MOVBZ Move Byte Zero Extend MOVBZ

Group Data Movement Instructions

Syntax **MOVBZ op1, op2**

Source Operand(s) op2 → BYTE

Destination Operand(s) op1 → WORD

Operation

(low byte op1) ← (op2)
(high byte op1) ← 00H

Description

Moves and zero-extends the contents of the source byte operand specified by op2 to the word location specified by the destination operand op1. The contents of the moved data is examined, and the flags are updated accordingly.

CPU Flags

E	Z	V	C	N
0	*	-	-	0

- E Always cleared.
- Z Set if the value of the source byte operand op2 equals zero. Cleared otherwise.
- V Not affected.
- C Not affected.
- N Always cleared.

Encoding

Mnemonic		Format	Bytes
MOVBZ	Rw_n , Rb_m	C0 mn	2
MOVBZ	mem , reg	C5 RR MM MM	4
MOVBZ	reg , mem	C2 RR MM MM	4

MUL Signed Multiplication MUL

Group Arithmetic Instructions

Syntax **MUL op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) MD → DOUBLEWORD

Operation

(MD) ← (op1) * (op2)

Description

Performs a 16-bit by 16-bit signed multiplication using the two words specified by operands op1 and op2 respectively. The signed 32-bit result is placed in the MD register.

CPU Flags

E	Z	V	C	N
0	*	*	0	*

- E Always cleared.
- Z Set if result equals zero. Cleared otherwise.
- V This bit is set if the result cannot be represented in a word data type. Cleared otherwise.
- C Always cleared.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
MUL	Rw_n , Rw_m	0B nm	2

MULU Unsigned Multiplication MULU

Group Arithmetic Instructions

Syntax **MULU op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) MD → DOUBLEWORD

Operation

(MD) ← (op1) * (op2)

Description

Performs a 16-bit by 16-bit unsigned multiplication using the two words specified by operands op1 and op2 respectively. The unsigned 32-bit result is placed in the MD register.

CPU Flags

E	Z	V	C	N
0	*	*	0	*

- E Always cleared.
- Z Set if result equals zero. Cleared otherwise.
- V This bit is set if the result cannot be represented in a word data type. Cleared otherwise.
- C Always cleared.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
MULU	Rw_n , Rw_m	1B nm	2

NEG Integer Two's Complement NEG

Group Arithmetic Instructions

Syntax **NEG op1**

Source Operand(s) op1 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ← 0 - (op1)

Description

Performs a binary 2's complement of the source operand specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
*	*	*	*	*

- E Set if the value of op1 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the word data type. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
NEG	Rw_n	81 n0	2

NEGB Integer Two's Complement NEGB

Group Arithmetic Instructions

Syntax **NEGB op1**

Source Operand(s) op1 → BYTE

Destination Operand(s) op1 → BYTE

Operation

(op1) ← 0 - (op1)

Description

Performs a binary 2's complement of the source operand specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
*	*	*	*	*

- E Set if the value of op1 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the byte data type. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
NEGB	Rb_n	A1 n0	2

NOP No Operation NOP

Group Null operation

Syntax **NOP**

Source Operand(s) none

Destination Operand(s) none

Operation

No Operation

Description

This instruction causes a null operation to be performed. A null operation causes no change in the status of the flags.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic	Format	Bytes
NOP	CC 00	2

OR — Logical OR — OR

Group: Logical Instructions

Syntax **OR op1, op2**

Source Operand(s): op1, op2 → WORD

Destination Operand(s): op1 → WORD

Operation

(op1) ← (op1) ∨ (op2)

Description

Performs a bitwise logical OR of the source operand specified by op2 and the destination operand specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
*	*	0	0	*

- E — Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z — Set if result equals zero. Cleared otherwise.
- V — Always cleared.
- C — Always cleared.
- N — Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
OR	Rw_n , #data3	78 n:0###	2
OR	Rw_n , Rw_m	70 nm	2
OR	Rw_n , [Rw_i+]	78 n:11ii	2
OR	Rw_n , [Rw_i]	78 n:10ii	2
OR	mem , reg	74 RR MM MM	4
OR	reg , #data16	76 RR ## ##	4
OR	reg , mem	72 RR MM MM	4

ORB Logical OR ORB

Group Logical Instructions

Syntax **ORB op1, op2**

Source Operand(s) op1, op2 → BYTE

Destination Operand(s) op1 → BYTE

Operation

(op1) ← (op1) ∨ (op2)

Description

Performs a bitwise logical OR of the source operand specified by op2 and the destination operand specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
*	*	0	0	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
ORB	Rb_n , #data3	79 n:0###	2
ORB	Rb_n , Rb_m	71 nm	2
ORB	Rb_n , [Rw_i+]	79 n:11ii	2
ORB	Rb_n , [Rw_i]	79 n:10ii	2
ORB	mem , reg	75 RR MM MM	4
ORB	reg , #data8	77 RR ## xx	4
ORB	reg , mem	73 RR MM MM	4

PCALL Push Word and Call Subroutine Absolute PCALL

Group Call Instructions

Syntax **PCALL op1, op2**

Source Operand(s) op1 → WORD
op2 → 16-bit address offset

Destination Operand(s) none

Operation

(tmp) ← (op1)
(SP) ← (SP) - 2
((SP)) ← (tmp)
(SP) ← (SP) - 2
((SP)) ← (IP)
(IP) ← op2

Description

Pushes the word specified by operand op1 and the value of the instruction pointer, IP, onto the system stack, and branches to the absolute memory location specified by the second operand op2. Because IP always points to the instruction following the branch instruction, the value stored on the system stack represents the return address of the calling routine.

CPU Flags

E	Z	V	C	N
*	*	-	-	*

E Set if the value of the pushed operand op1 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.

Z Set if the value of the pushed operand op1 equals zero. Cleared otherwise.

V Not affected.

C Not affected.

N Set if the most significant bit of the pushed operand op1 is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
PCALL	reg , caddr	E2 RR MM MM	4

POP Pop Word from System Stack POP

Group System Stack Instructions

Syntax **POP op1**

Source Operand(s) none

Destination Operand(s) op1 → WORD

Operation

(tmp) ← ((SP))
(SP) ← (SP) + 2
(op1) ← (tmp)

Description

Pops one word from the system stack specified by the Stack Pointer into the operand specified by op1. The Stack Pointer is then incremented by two.

CPU Flags

E	Z	V	C	N
*	*	-	-	*

- E Set if the value of the popped word represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if the value of the popped word equals zero. Cleared otherwise.
- V Not affected.
- C Not affected.
- N Set if the most significant bit of the popped word is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
POP	reg	FC RR	2

PRIOR Prioritize Register PRIOR

Group Prioritize Instruction

Syntax **PRIOR op1, op2**

Source Operand(s) op2 → WORD

Destination Operand(s) op1 → WORD

Operation

```
(tmp) ← (op2)
(count) ← 0
DO WHILE (((tmp[15]) ≠ 1) AND ((op2) ≠ 0)))
        (tmp[n]) ← (tmp[n-1]) [n=15...1]
        (count) ← (count) + 1
END WHILE
(op1) ← (count)
```

Description

This instruction stores a count value in the word operand specified by op1. This count value indicates the number of single bit shifts required to normalize the word operand op2 so that its most significant bit is equal to one. If the source operand op2 equals zero, a zero is written to operand op1 and the zero flag is set. Otherwise the zero flag is cleared.

CPU Flags

E	Z	V	C	N
0	*	0	0	0

- E Always cleared.
- Z Set if the value of the source operand op2 equals zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.
- N Always cleared.

Encoding

Mnemonic		Format	Bytes
PRIOR	Rw_n , Rw_m	2B nm	2

PUSH Push Word on System Stack PUSH

Group System Stack Instructions

Syntax **PUSH op1**

Source Operand(s) op1 → WORD

Destination Operand(s) none

Operation

(tmp) ← (op1)
(SP) ← (SP) - 2
((SP)) ← (tmp)

Description

Moves the word specified by operand op1 to the location in the system stack specified by the Stack Pointer, after the Stack Pointer has been decremented by two.

CPU Flags

E	Z	V	C	N
*	*	-	-	*

- E Set if the value of the pushed operand op1 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if the value of the pushed operand op1 equals zero. Cleared otherwise.
- V Not affected.
- C Not affected.
- N Set if the most significant bit of the pushed operand op1 is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
PUSH	reg	EC RR	2

PWRDN Enter Power Down Mode PWRDN

Group System Control Instructions

Syntax **PWRDN**

Source Operand(s) none

Destination Operand(s) none

Operation

Enter Power Down Mode

Description

This instruction causes the part to enter the power down mode. In this mode, all peripherals and the CPU are powered down until the part is externally reset. To insure that this instruction is not accidentally executed, it is implemented as a protected instruction. To further control the action of this instruction, the PWRDN instruction is only enabled when the non-maskable interrupt pin (NMI) is in the low state. Otherwise, this instruction has no effect.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic	Format	Bytes
PWRDN	97 68 97 97	4

RET Return from Subroutine RET

Group Return Instructions

Syntax **RET**

Source Operand(s) none

Destination Operand(s) none

Operation

(IP) ← ((SP))
(SP) ← (SP) + 2

Description

Returns from a subroutine. The IP is popped from the system stack.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic	Format	Bytes
RET	CB 00	2

RETI Return from Interrupt Subroutine RETI

Group Return Instructions

Syntax **RETI**

Source Operand(s) none

Destination Operand(s) none

Operation

```
(IP) ← ((SP))
(SP) ← (SP) + 2
IF (SYSCON.SGTDIS = 0) THEN
        (CSP) ← ((SP))
        (SP) ← (SP) + 2
END IF
(PSW) ← ((SP))
(SP) ← (SP) + 2
```

Description

Returns from an interrupt routine. The IP, CSP, and PSW are popped off the system stack. The CSP is only popped if segmentation is enabled. This is indicated by the SGTDIS bit in the SYSCON register.

CPU Flags

E	Z	V	C	N
*	*	*	*	*

E Restored from the PSW popped from stack.
Z Restored from the PSW popped from stack.
V Restored from the PSW popped from stack.
C Restored from the PSW popped from stack.
N Restored from the PSW popped from stack.

Encoding

Mnemonic	Format	Bytes
RETI	FB 88	2

RETP Return from Subroutine and Pop Word RETP

Group Return Instructions

Syntax **RETP op1**

Source Operand(s) none

Destination Operand(s) op1 → WORD

Operation

(IP) ← ((SP))
(SP) ← (SP) + 2
(tmp) ← ((SP))
(SP) ← (SP) + 2
(op1) ← (tmp)

Description

Returns from a subroutine. First the IP is popped from the system stack and then the next word is popped from the system stack into the operand specified by op1.

CPU Flags

E	Z	V	C	N
*	*	-	-	*

- E Set if the value of the popped word represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if the value of the popped word equals zero. Cleared otherwise.
- V Not affected.
- C Not affected.
- N Set if the most significant bit of the popped word is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
RETP	reg	EB RR	2

RETS Return from Inter-Segment Subroutine RETS

Group Return Instructions

Syntax **RETS**

Source Operand(s) none

Destination Operand(s) none

Operation

(IP) ← ((SP))
(SP) ← (SP) + 2
(CSP) ← ((SP))
(SP) ← (SP) + 2

Description

Returns from an inter-segment subroutine. The IP and CSP are popped from the system stack.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic	Format	Bytes
RETS	DB 00	2

ROL — Rotate Left — ROL

Group: Shift and Rotate Instructions

Syntax **ROL op1, op2**

Source Operand(s): op1 → WORD
op2 → shift counter

Destination Operand(s): op1 → WORD

Operation

```
(count) ← (op2)
(C) ← 0
DO WHILE ((count) ≠ 0)
        (C) ← (op1[15])
        (op1[n]) ← (op1[n-1]) [n=15...1]
        (op1[0]) ← (C)
        (count) ← (count) - 1
END WHILE
```

Description

Rotates the destination word operand op1 the number of times as specified by the source operand op2. Bit 15 is rotated into Bit 0 and into the Carry. Only shift values between 0 and 15 are allowed. When using a GPR as the count control, only the least significant 4 bits are used.

CPU Flags

E	Z	V	C	N
0	*	0	S	*

- E — Always cleared.
- Z — Set if result equals zero. Cleared otherwise.
- V — Always cleared.
- C — The carry flag is set according to the last most significant bit shifted out of op1. Cleared for a shift count of zero.
- N — Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
ROL	Rw_n , #data4	1C #n	2
ROL	Rw_n , Rw_m	0C nm	2

ROR Rotate Right ROR

Group Shift and Rotate Instructions

Syntax **ROR op1, op2**

Source Operand(s) op1 → WORD
op2 → shift counter

Destination Operand(s) op1 → WORD

Operation

```
(count) ← (op2)
(C) ← 0
(V) ← 0
DO WHILE ((count) ≠ 0)
        (V) ← (V) ∨ (C)
        (C) ← (op1[0])
        (op1[n]) ← (op1[n+1]) [n=0...14]
        (op1[15]) ← (C)
        (count) ← (count) - 1
END WHILE
```

Description
Rotates the destination word operand op1 right by the number of times as specified by the source operand op2. Bit 0 is rotated into Bit 15 and into the Carry. Only shift values between 0 and 15 are allowed. When using a GPR as the count control, only the least significant 4 bits are used.

CPU Flags

E	Z	V	C	N
0	*	S	S	*

- E Always cleared.
- Z Set if result equals zero. Cleared otherwise.
- V Set if in any cycle of the rotate operation a 1 is shifted out of the carry flag. Cleared for a rotate count of zero.
- C The carry flag is set according to the last least significant bit shifted out of op1. Cleared for a shift count of zero.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
ROR	Rw_n , #data4	3C #n	2
ROR	Rw_n , Rw_m	2C nm	2

SCXT Switch Context SCXT

Group System Stack Instructions

Syntax **SCXT op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(tmp1) ← (op1)
(tmp2) ← (op2)
(SP) ← (SP) - 2
((SP)) ← (tmp1)
(op1) ← (tmp2)

Description

Switches contexts of any register. Switching context is a push and load operation. The contents of the register specified by the first operand op1, are pushed onto the stack. That register is then loaded with the value specified by the second operand, op2.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
SCXT	reg , #data16	C6 RR ## ##	4
SCXT	reg , mem	D6 RR MM MM	4

SHL Shift Left SHL

Group Shift and Rotate Instructions

Syntax **SHL op1, op2**

Source Operand(s) op1 → WORD
op2 → shift counter

Destination Operand(s) op1 → WORD

Operation

```
(count) ← (op2)
(C) ← 0
DO WHILE ((count) ≠ 0)
        (C) ← (op1[15])
        (op1[n]) ← (op1[n-1]) [n=15...1]
        (op1[0]) ← 0
        (count) ← (count) - 1
END WHILE
```

Description

Shifts the destination word operand op1 the number of times as specified by the source operand op2. The least significant bits of the result are filled with zeros accordingly. The least The most significant bit is shifted into the Carry. Only shift values between 0 and 15 are allowed. When using a GPR as the count control, only the least significant 4 bits are used.

CPU Flags

E	Z	V	C	N
0	*	0	S	*

E Always cleared.

Z Set if result equals zero. Cleared otherwise.

V Always cleared.

C The carry flag is set according to the last most significant bit shifted out of op1. Cleared for a shift count of zero.

N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
SHL	Rw_n , #data4	5C #n	2
SHL	Rw_n , Rw_m	4C nm	2

SHR — Shift Right — SHR

Group: Shift and Rotate Instructions

Syntax **SHR op1, op2**

Source Operand(s): op1 → WORD
op2 → shift counter

Destination Operand(s): op1 → WORD

Operation

```
(count) ← (op2)
(C) ← 0
(V) ← 0
DO WHILE ((count) ≠ 0)
        (V) ← (C) ∨ (V)
        (C) ← (op1[0])
        (op1[n]) ← (op1[n+1]) [n=0...14]
        (op1[15]) ← 0
        (count) ← (count) - 1
END WHILE
```

Description

Shifts the destination word operand op1 right by the number of times as specified by the source operand op2. The most significant bits of the result are filled with zeros accordingly. Since the bits shifted out effectively represent the remainder, the Overflow flag is used instead as a Rounding flag. A shift right is a division by a power of two. The overflow flag with the carry flag allows to determine whether the fractional part of the division result is greater than, less than or equal to one half (0.5 in decimal base). This allows to round the division result accordingly. Only shift values between 0 and 15 are allowed. When using a GPR as the count control, only the least significant 4 bits are used.

CPU Flags

E	Z	V	C	N
0	*	S	S	*

E Always cleared.

Z Set if result equals zero. Cleared otherwise.

V Set if in any cycle of the shift operation a 1 is shifted out of the carry flag. Cleared in case of a shift count equal 0.

C	The carry flag is set according to the last least significant bit shifted out of op1. Cleared for a shift count of zero.
N	Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
SHR	Rw_n , #data4	7C #n	2
SHR	Rw_n , Rw_m	6C nm	2

SRST Software Reset SRST

Group System Control Instructions

Syntax **SRST**

Source Operand(s) none

Destination Operand(s) none

Operation

Software Reset

Description

This instruction is used to perform a software reset. A software reset has the same effect on the microcontroller as an externally applied hardware reset. To insure that this instruction is not accidentally executed, it is implemented as a protected instruction.

CPU Flags

E	Z	V	C	N
0	0	0	0	0

E Always cleared.
Z Always cleared.
V Always cleared.
C Always cleared.
N Always cleared.

Encoding

Mnemonic	Format	Bytes
SRST	B7 48 B7 B7	4

SRVWDT Service Watchdog Timer SRVWDT

Group System Control Instructions

Syntax **SRVWDT**

Source Operand(s) none

Destination Operand(s) none

Operation

Service Watchdog Timer

Description

This instruction reloads the high order byte of the Watchdog Timer with a preset value and clears the low byte. Once this instruction has been executed, the watchdog timer cannot be disabled. To insure that this instruction is not accidentally executed, it is implemented as a protected instruction.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic	Format	Bytes
SRVWDT	A7 58 A7 A7	4

SUB Integer Subtraction SUB

Group Arithmetic Instructions

Syntax **SUB op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ← (op1) - (op2)

Description

Performs a 2's complement binary subtraction of the source operand specified by op2 and the destination operand specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
*	*	*	S	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the word data type. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
SUB	Rw_n , #data3	28 n:0###	2
SUB	Rw_n , Rw_m	20 nm	2
SUB	Rw_n , [Rw_i+]	28 n:11ii	2
SUB	Rw_n , [Rw_i]	28 n:10ii	2
SUB	mem , reg	24 RR MM MM	4
SUB	reg , #data16	26 RR ## ##	4
SUB	reg , mem	22 RR MM MM	4

SUBB Integer Subtraction SUBB

Group Arithmetic Instructions

Syntax **SUBB op1, op2**

Source Operand(s) op1, op2 → BYTE

Destination Operand(s) op1 → BYTE

Operation

(op1) ← (op1) - (op2)

Description

Performs a 2's complement binary subtraction of the source operand specified by op2 and the destination operand specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
*	*	*	S	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the word data type. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
SUBB	Rb_n , #data3	29 n:0###	2
SUBB	Rb_n , Rb_m	21 nm	2
SUBB	Rb_n , [Rw_i+]	29 n:11ii	2
SUBB	Rb_n , [Rw_i]	29 n:10ii	2
SUBB	mem , reg	25 RR MM MM	4
SUBB	reg , #data8	27 RR ## xx	4
SUBB	reg , mem	23 RR MM MM	4

SUBC Integer Subtraction with Carry SUBC

Group Arithmetic Instructions

Syntax **SUBC op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ← (op1) - (op2) - (C)

Description

Performs a 2's complement binary subtraction of the source operand specified by op2 and the previously generated carry bit from the destination operand specified by op1. The result is then stored in op1. This instruction can be used to perform multiple precision arithmetic.

CPU Flags

E	Z	V	C	N
*	S	*	S	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero and previous Z flag was set. Cleared otherwise.
- V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the word data type. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
SUBC	Rw_n , #data3	38 n:0###	2
SUBC	Rw_n , Rw_m	30 nm	2
SUBC	Rw_n , [Rw_i+]	38 n:11ii	2
SUBC	Rw_n , [Rw_i]	38 n:10ii	2
SUBC	mem , reg	34 RR MM MM	4
SUBC	reg , #data16	36 RR ## ##	4
SUBC	reg , mem	32 RR MM MM	4

SUBCB Integer Subtraction with Carry SUBCB

Group Arithmetic Instructions

Syntax **SUBCB op1, op2**

Source Operand(s) op1, op2 → BYTE

Destination Operand(s) op1 → BYTE

Operation

(op1) ← (op1) - (op2) - (C)

Description

Performs a 2's complement binary subtraction of the source operand specified by op2 and the previously generated carry bit from the destination operand specified by op1. The result is then stored in op1. This instruction can be used to perform multiple precision arithmetic.

CPU Flags

E	Z	V	C	N
*	S	*	S	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero and the previous Z flag was set. Cleared otherwise.
- V Set if an arithmetic underflow occurred, i.e. the result cannot be represented in the word data type. Cleared otherwise.
- C Set if a borrow is generated. Cleared otherwise.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
SUBCB	Rb_n , #data3	39 n:0###	2
SUBCB	Rb_n , Rb_m	31 nm	2
SUBCB	Rb_n , [Rw_i+]	39 n:11ii	2
SUBCB	Rb_n , [Rw_i]	39 n:10ii	2
SUBCB	mem , reg	35 RR MM MM	4
SUBCB	reg , #data8	37 RR ## xx	4
SUBCB	reg , mem	33 RR MM MM	4

TRAP Software Trap TRAP

Group Call Instructions

Syntax **TRAP op1**

Source Operand(s) op1 → 7-bit trap number

Destination Operand(s) none

Operation

```
(SP) ← (SP) - 2
((SP) ← (PSW)
IF (SYSCON.SGTDIS = 0) THEN
        (SP) ← (SP) - 2
        ((SP)) ← (CSP)
        (CSP) ← (0)
END IF
(SP) ← (SP) - 2
((SP)) ← (IP)
(IP) ←  zero_extended((op1) * 4)
```

Description

Invokes a trap or interrupt routine based on the specified operand op1. The invoked routine is determined by branching to the specified vector table entry point. This routine has no indication of whether it was called by software or hardware. System state is preserved identically to hardware interrupt entry except that the CPU priority level is not affected. The RETI, Return from Interrupt instruction is used to resume execution after the completion of the trap or interrupt routine. The CSP is pushed if the segmentation is enabled. This is indicated by the SGTDIS bit of the SYSCON register.

CPU Flags

E	Z	V	C	N
-	-	-	-	-

E Not affected.
Z Not affected.
V Not affected.
C Not affected.
N Not affected.

Encoding

Mnemonic		Format	Bytes
TRAP	#trap7	9B t:ttt0	2

XOR Logical Exclusive OR XOR

Group Logical Instructions

Syntax **XOR op1, op2**

Source Operand(s) op1, op2 → WORD

Destination Operand(s) op1 → WORD

Operation

(op1) ← (op1) ⊕ (op2)

Description

Performs a bitwise logical EXCLUSIVE OR of the source operand specified by op2 and the destination operand specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
*	*	0	0	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
XOR	Rw_n , #data3	58 n:0###	2
XOR	Rw_n , Rw_m	50 nm	2
XOR	Rw_n , [Rw_i+]	58 n:11ii	2
XOR	Rw_n , [Rw_i]	58 n:10ii	2
XOR	mem , reg	54 RR MM MM	4
XOR	reg , #data16	56 RR ## ##	4
XOR	reg , mem	52 RR MM MM	4

XORB Logical Exclusive OR XORB

Group Logical Instructions

Syntax **XORB op1, op2**

Source Operand(s) op1, op2 → BYTE

Destination Operand(s) op1 → BYTE

Operation

(op1) ← (op1) ⊕ (op2)

Description

Performs a bitwise logical EXCLUSIVE OR of the source operand specified by op2 and the destination operand specified by op1. The result is then stored in op1.

CPU Flags

E	Z	V	C	N
*	*	0	0	*

- E Set if the value of op2 represents the lowest possible negative number. Cleared otherwise. Used to signal the end of a table.
- Z Set if result equals zero. Cleared otherwise.
- V Always cleared.
- C Always cleared.
- N Set if the most significant bit of the result is set. Cleared otherwise.

Encoding

Mnemonic		Format	Bytes
XORB	Rb_n , #data3	59 n:0###	2
XORB	Rb_n , Rb_m	51 nm	2
XORB	Rb_n , [Rw_i+]	59 n:11ii	2
XORB	Rb_n , [Rw_i]	59 n:10ii	2
XORB	mem , reg	55 RR MM MM	4
XORB	reg , #data8	57 RR ## xx	4
XORB	reg , mem	53 RR MM MM	4

7 Parallel Ports

In order to accept or generate single external control signals or parallel data, the C166S s provides up to 48 parallel IO lines organized into six 8-bit IO ports (PORT0 made of P0H and P0L, PORT1 made of P1H and P1L, Port 4, Port 6).

These port lines may be used for general purpose Input/Output controlled via software or may be used implicitly by the C166S V1 SubS R1's integrated peripherals or 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. The IO ports are true bidirectional ports which are switched to high impedance state when configured as inputs.

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

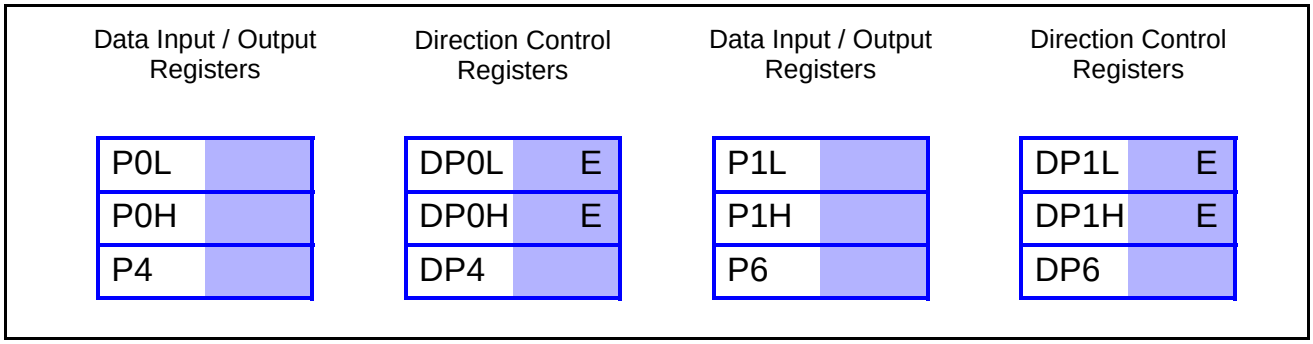


Figure 7-1 SFRs associated with the Parallel Ports

A write operation to a port pin configured as an input causes the value to be written into the port output register, while a read operation returns the registered 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 register.

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

7.1 Alternate Port Functions

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

Table 7-1 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 ext. resources	A15 ... A0,
Port 4	Selected segment address lines in systems with more than 64 KBytes of external resources	A23 ... A16
Port 6	Chip select output signals	$\overline{CS4}$ ... $\overline{CS0}$

If an **alternate output function** of a pin is to be used, the direction of this pin must be programmed for output (DPx.y='1'), except for some signals that are used directly after reset and are configured automatically. Otherwise the pin remains in the high-impedance state and is not effected by the alternate output function. The respective port register should hold a '1', because its output is combined with the alternate output data.

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 register. Thus, the alternate input function reads the value stored in the port output register. This can be used for testing purposes to allow a software trigger of an alternate input function by writing to the port output register.

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, where 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 registers check how the alternate data output is combined with the respective port register output.

There is one basic structure for all port lines with only an alternate input function. Port lines with only an alternate output function, however, have different structures due to the

way the direction of the pin is switched and depending on whether the pin is accessible by the user software or not in the alternate function mode.

All port lines that are not used for these alternate functions may be used as general purpose IO lines. When using port pins for general purpose output, the initial output value should be written to the port register prior to enabling the output drivers, in order to avoid undesired transitions on the output pins. This applies to single pins as well as to pin groups (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 3.8.1*** *"Particular Pipeline Effects" in chapter "The Central Processing Unit").*

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

7.2 PORT0

The two 8-bit ports P0H and P0L represent the higher and lower part of PORT0, respectively. Both halfs of PORT0 can be written (e.g. via a PEC transfer) without effecting the other half.
If this port is used for general purpose IO, the direction of each line can be configured via the corresponding direction registers 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).

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.

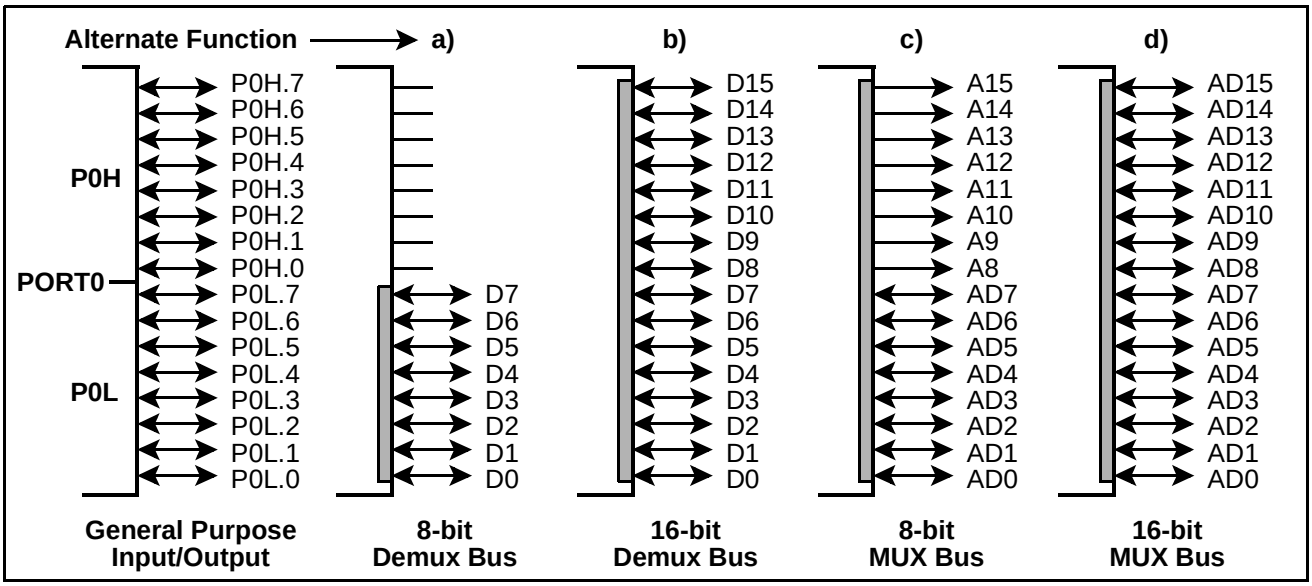


Figure 7-2 PORT0 IO and Alternate Functions

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 can be the 16-bit intrasegment address or the 8/16-bit data information. The incoming data on PORT0 is read on the line "Alternate Data Input". 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 becomes active.

The figure below shows the structure of a PORT0 pin.

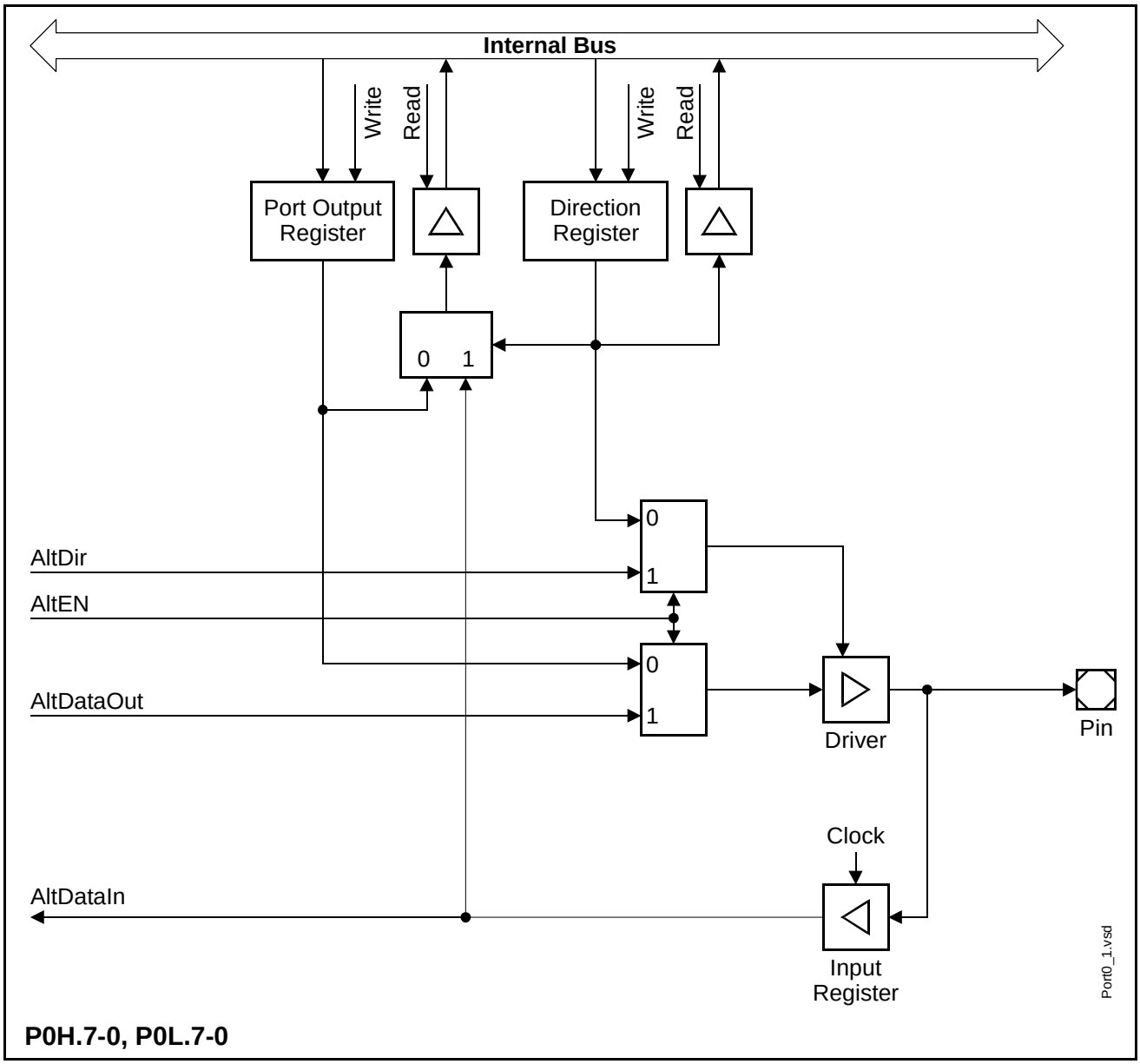


Figure 7-3 Block Diagram of a PORT0 Pin

7.3 PORT1

The two 8-bit ports P1H and P1L represent the higher and lower part of PORT1, respectively. Both halfs of PORT1 can be written (e.g. via a PEC transfer) without effecting the other half.
If this port is used for general purpose IO, the direction of each line can be configured via the corresponding direction registers DP1H and DP1L.

P1L
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
-	-	-	-	-	-	-	-	rw	rw	rw	rw	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
								DP1L.7	DP1L.6	DP1L.5	DP1L.4	DP1L.3	DP1L.2	DP1L.1	DP1L.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
								DP1H.7	DP1H.6	DP1H.5	DP1H.4	DP1H.3	DP1H.2	DP1H.1	DP1H.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.

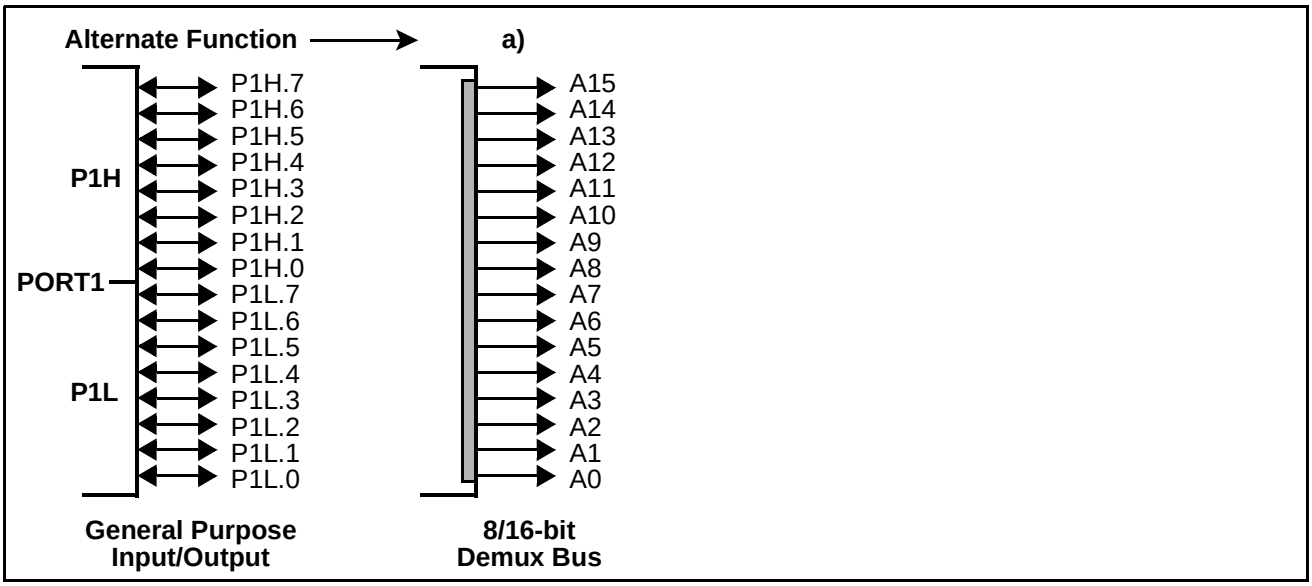


Figure 7-4 PORT1 IO and Alternate Functions

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 register are controlled by the bus controller hardware. The input of the port output register 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 register, otherwise unpredictable results may occur. When the external bus modes are disabled, the contents of the direction register last written by the user becomes active.

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 register input.

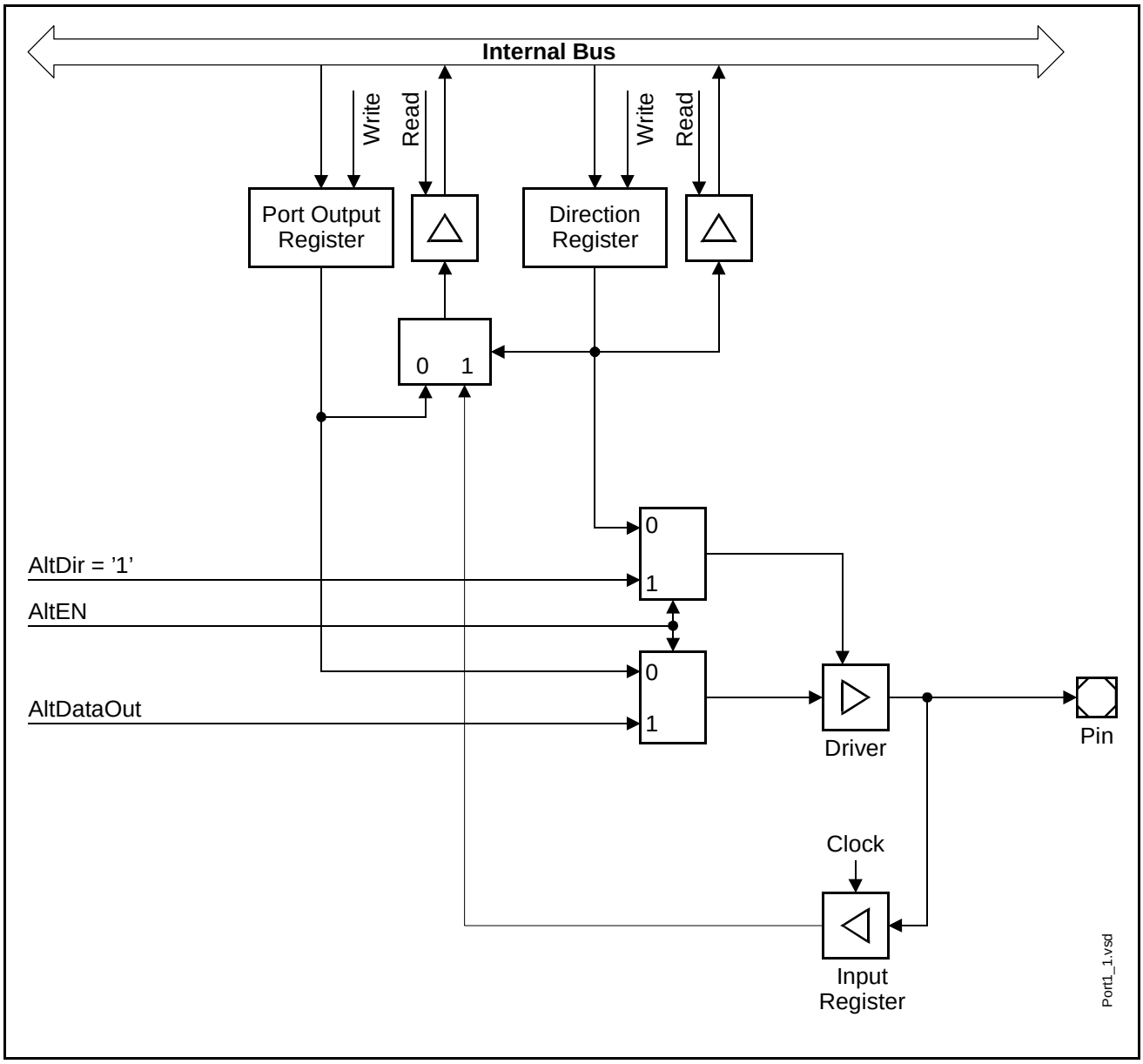


Figure 7-5 Block Diagram of a PORT1 Pin with Address Function

7.4 Port 4

If this 8-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
								P4.7	**P4.6**	**P4.5**	**P4.4**	**P4.3**	**P4.2**	**P4.1**	**P4.0**
-	-	-	-	-	-	-	-	rw	rw	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 .7	**DP4 .6**	**DP4 .5**	**DP4 .4**	**DP4 .3**	**DP4 .2**	**DP4 .1**	**DP4 .0**
-	-	-	-	-	-	-	-	rw	rw	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

Alternate Functions of Port 4

During external bus cycles that use segmentation (i.e. an address space above 64 KByte) a number of Port 4 pins may output the segment address lines. The number of pins that is used for segment address output determines the external address space which is directly accessible. The other pins of Port 4 (if any) may be used for general purpose IO.
If segment address lines are selected, the alternate function of Port 4 may be necessary to access e.g. external memory directly after reset. For this reason Port 4 will be switched to this alternate function automatically.

The number of segment address lines is selected via conf_rst_salsel_i[1:0] (SALSEL) during reset.

The table below summarizes the alternate functions of Port 4 depending on the number of selected segment address lines.

Table 7-2 Alternate Functions of Port 4

Port 4 Pin	Std. Function SALSEL=01 64 KB	Altern. Function SALSEL=11 256KB	Altern. Function SALSEL=00 1 MB	Altern. Function SALSEL=10 16 MB
P4.0	Gen. purpose IO	Seg. Address A16	Seg. Address A16	Seg. Address A16
P4.1	Gen. purpose IO	Seg. Address A17	Seg. Address A17	Seg. Address A17
P4.2	Gen. purpose IO	Gen. purpose IO	Seg. Address A18	Seg. Address A18
P4.3	Gen. purpose IO	Gen. purpose IO	Seg. Address A19	Seg. Address A19
P4.4	Gen. purpose IO	Gen. purpose IO	Gen. purpose IO	Seg. Address A20
P4.5	Gen. purpose IO	Gen. purpose IO	Gen. purpose IO	Seg. Address A21
P4.6	Gen. purpose IO	Gen. purpose IO	Gen. purpose IO	Seg. Address A22
P4.7	Gen. purpose IO	Gen. purpose IO	Gen. purpose IO	Seg. Address A23

Note: Port 4 pins that are not used for segment address output may be used for general purpose IO.

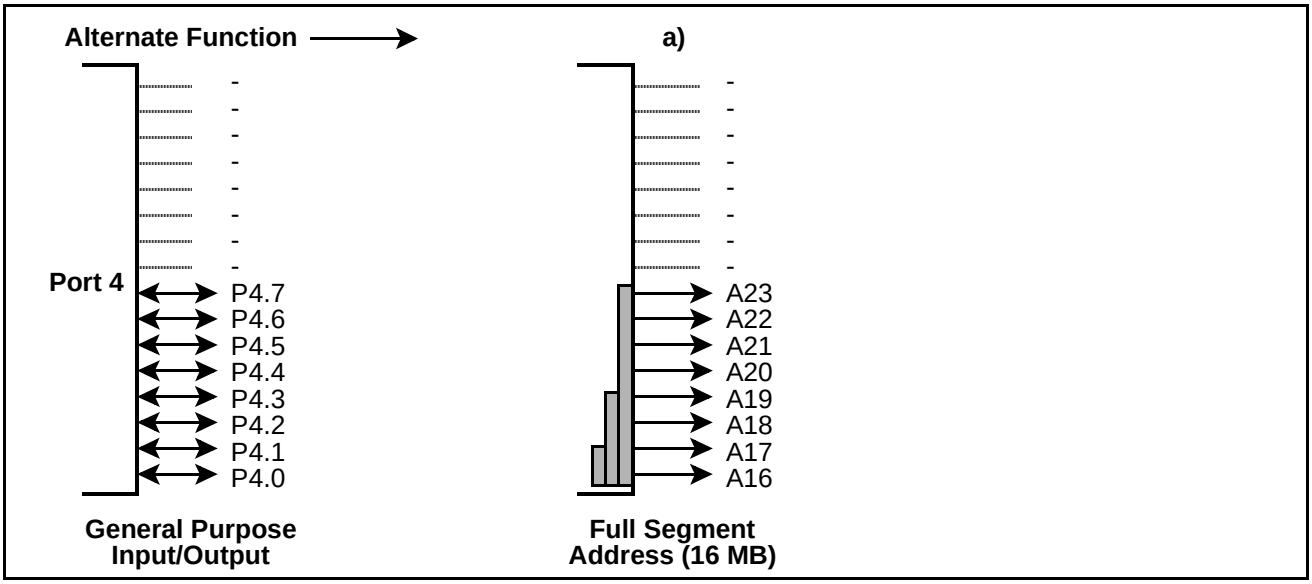


Figure 7-6 Port 4 IO and Alternate Functions

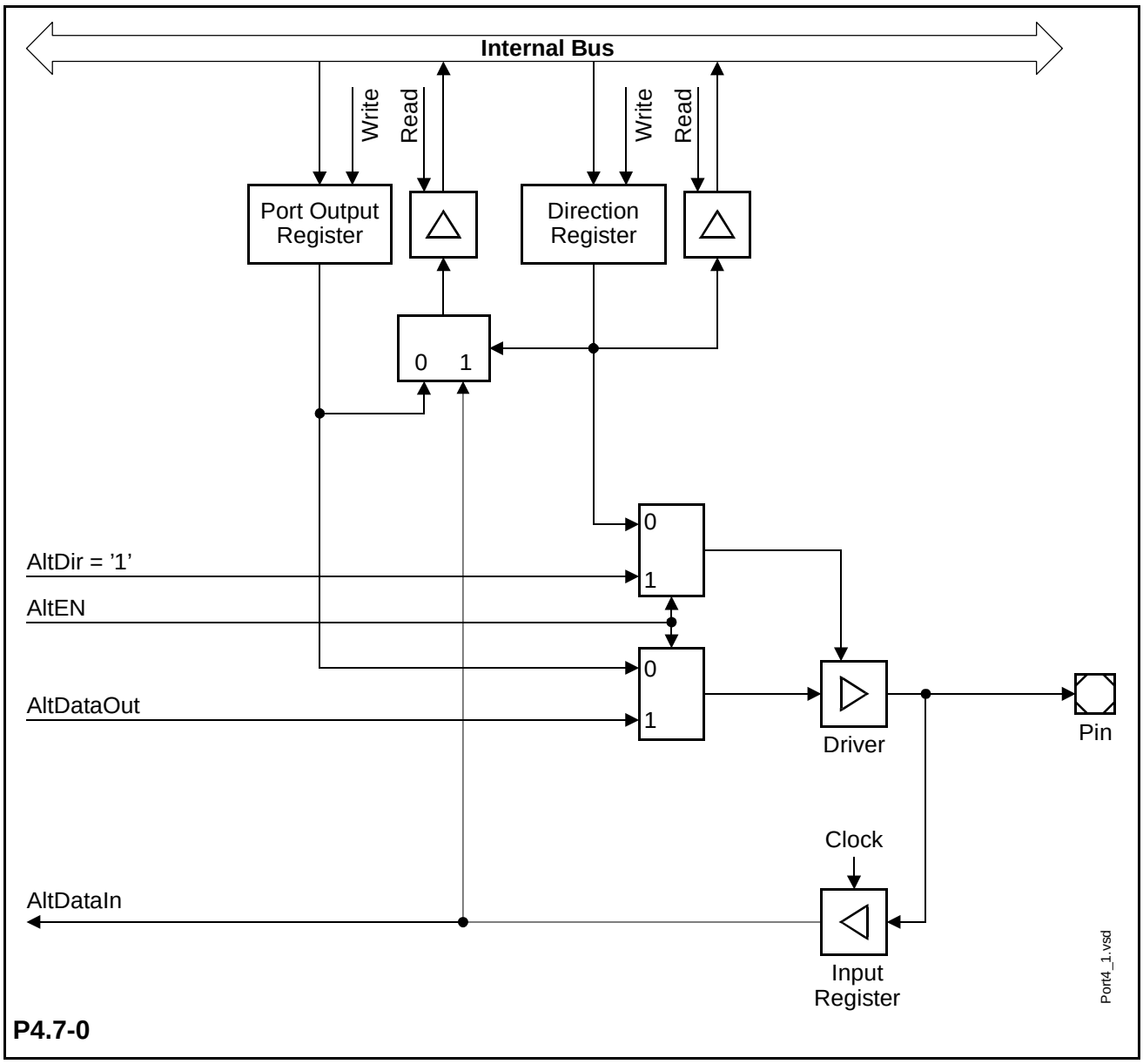


Figure 7-7 Block Diagram of a Port 4 Pin

7.5 Port 6

If this 8-bit port is used for general purpose IO, the direction of each line can be configured via the corresponding direction register DP6.

P6
Port 6 Data Register **SFR ($FFCC_H/E6_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								P6.7	**P6.6**	**P6.5**	**P6.4**	**P6.3**	**P6.2**	**P6.1**	**P6.0**
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
P6.y	**Port data register P6 bit y**

DP6
P6 Direction Ctrl. Register **SFR ($FFCE_H/E7_H$)** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								DP6 .7	**DP6 .6**	**DP6 .5**	**DP6 .4**	**DP6 .3**	**DP6 .2**	**DP6 .1**	**DP6 .0**
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
DP6.y	**Port direction register DP6 bit y** DP6.y = 0: Port line P6.y is an input (high-impedance) DP6.y = 1: Port line P6.y is an output

Alternate Functions of Port 6

A programmable number of chip select signals (CS4...CS0) derived from the bus control registers (BUSCON4...BUSCON0) can be output on 5 pins of Port 6. The number of chip select signals is selected via conf_rst_cssel_i[1:0] (CSSEL) during reset.

The table below summarizes the alternate functions of Port 6 depending on the number of selected chip select lines (coded via bitfield CSSEL).

Table 7-3 Alternate Functions of Port 6

Port 6 Pin	Altern. Function CSSEL = 10	Altern. Function CSSEL = 01	Altern. Function CSSEL = 00	Altern. Function CSSEL = 11
P6.0	Gen. purpose IO	Chip select $\overline{CS0}$	Chip select $\overline{CS0}$	Chip select $\overline{CS0}$
P6.1	Gen. purpose IO	Chip select $\overline{CS1}$	Chip select $\overline{CS1}$	Chip select $\overline{CS1}$
P6.2	Gen. purpose IO	Gen. purpose IO	Chip select $\overline{CS2}$	Chip select $\overline{CS2}$
P6.3	Gen. purpose IO	Gen. purpose IO	Gen. purpose IO	Chip select $\overline{CS3}$
P6.4	Gen. purpose IO	Gen. purpose IO	Gen. purpose IO	Chip select $\overline{CS4}$
P6.5	Gen. purpose IO	Gen. purpose IO	Gen. purpose IO	Gen. purpose IO
P6.6	Gen. purpose IO	Gen. purpose IO	Gen. purpose IO	Gen. purpose IO
P6.7	Gen. purpose IO	Gen. purpose IO	Gen. purpose IO	Gen. purpose IO

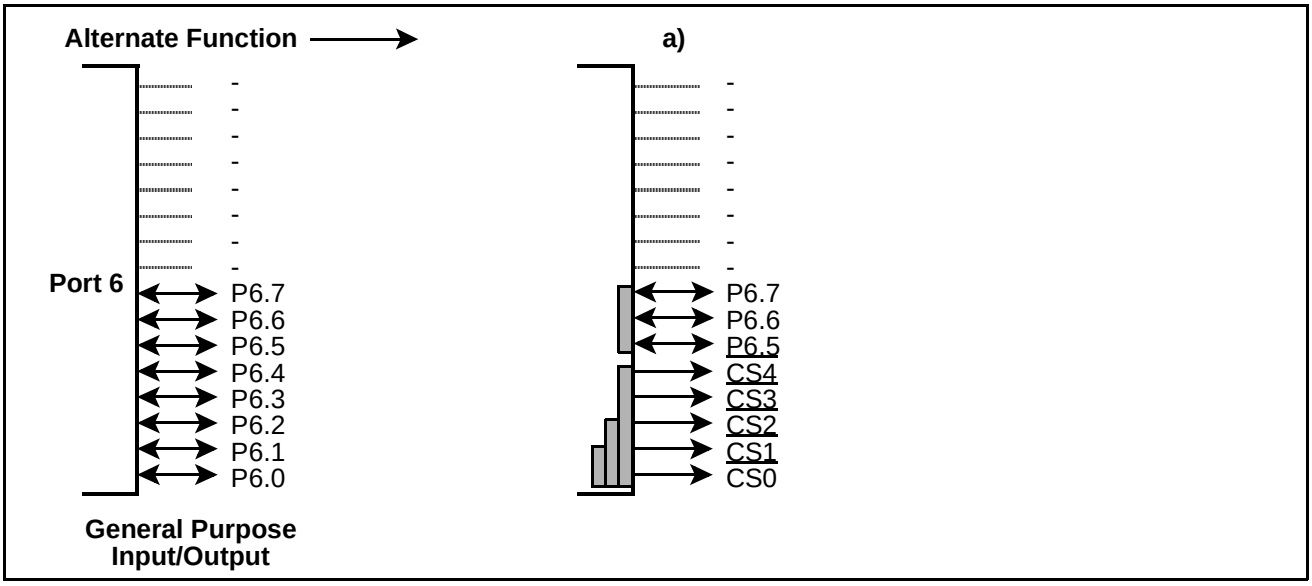


Figure 7-8 Port 6 IO and Alternate Functions

Note: The chip select lines of Port 6 should have (product specific) an internal weak pull-up device.This device should be switched on during reset for all potential $\overline{CS}$ output pins. This feature should be implemented to drive the chip select lines high during reset in order to avoid multiple chip selection.

After reset the $\overline{CS}$ function must be used, if selected so. In this case there is no possibility to program any port registers before. Thus the alternate function ($\overline{CS}$) is selected automatically in this case.

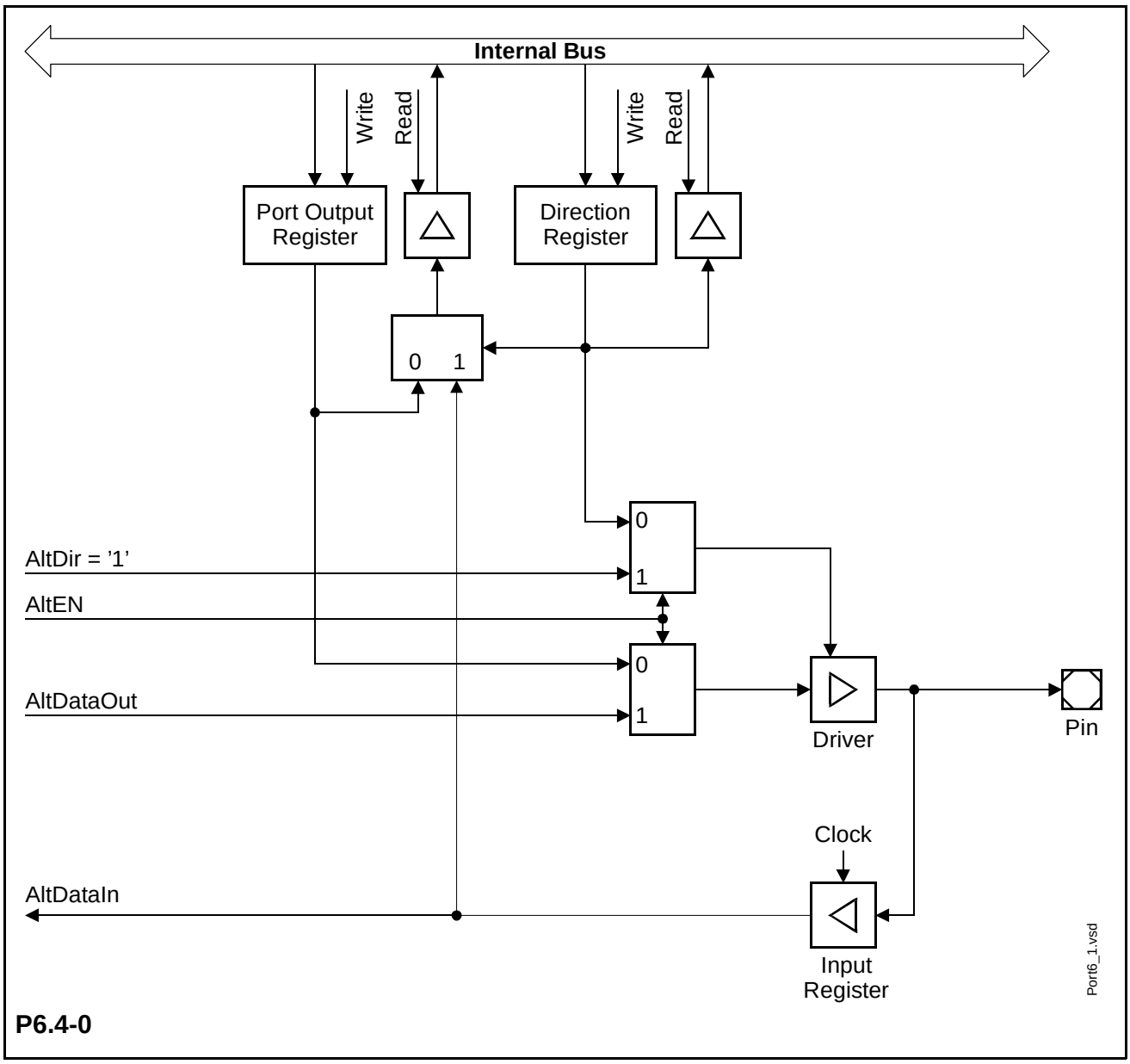


Figure 7-9 Block Diagram of Port 6 Pins with an alternate output function

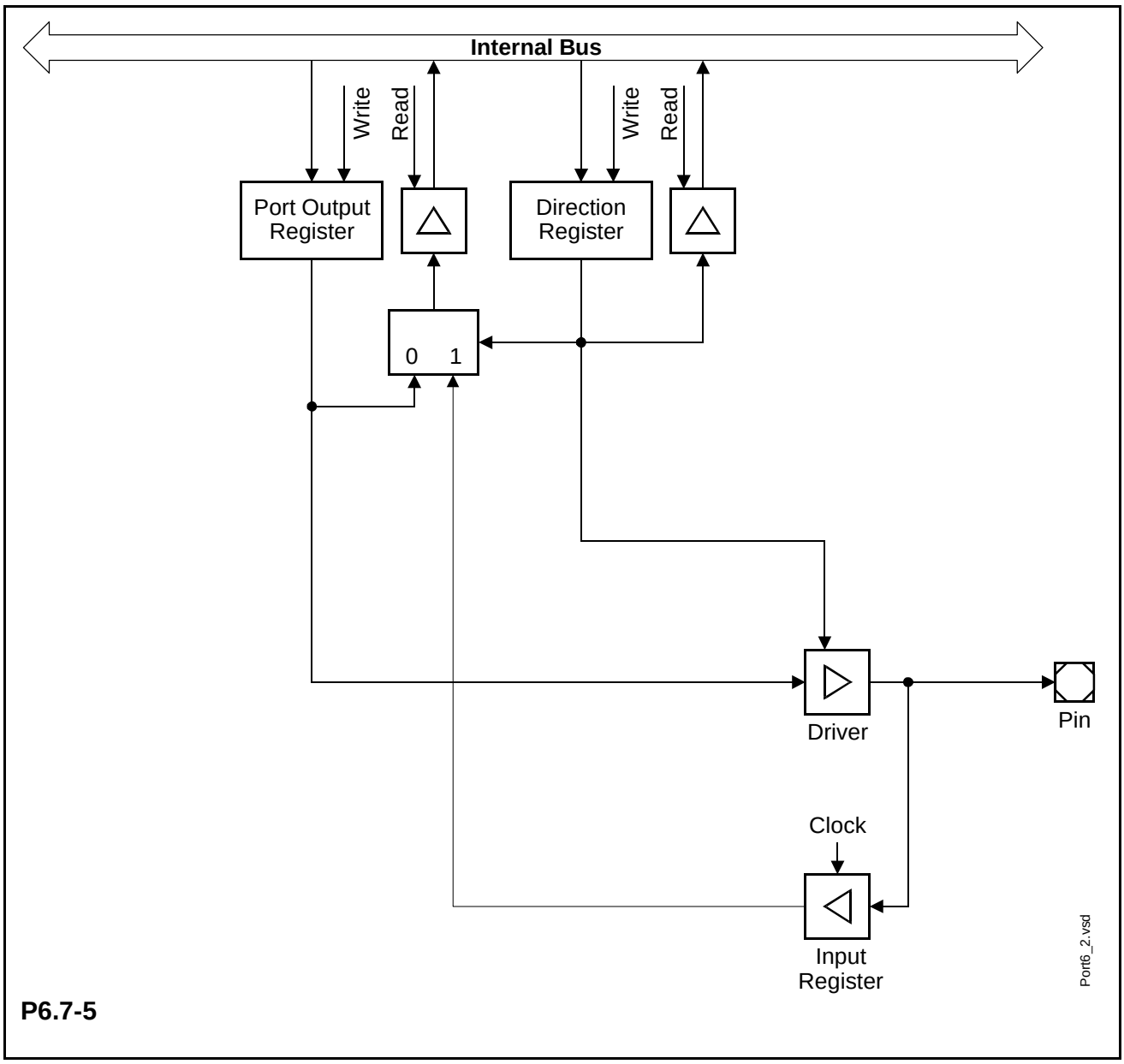


Figure 7-10 Block Diagram of Port 6 Pins without an alternate output function

8 The External Bus Interface

Although the C166S subsystem supports a powerful set of on-chip peripherals and on-chip RAM and ROM/OTP/Flash areas, these internal units cover only a small fraction of the chip's address space (up to 16 MBytes). The external bus interface allows access to external peripherals and additional volatile and non-volatile memory. The external bus interface has a number of possible configurations, so it can be tailored to fit perfectly into a given application system.

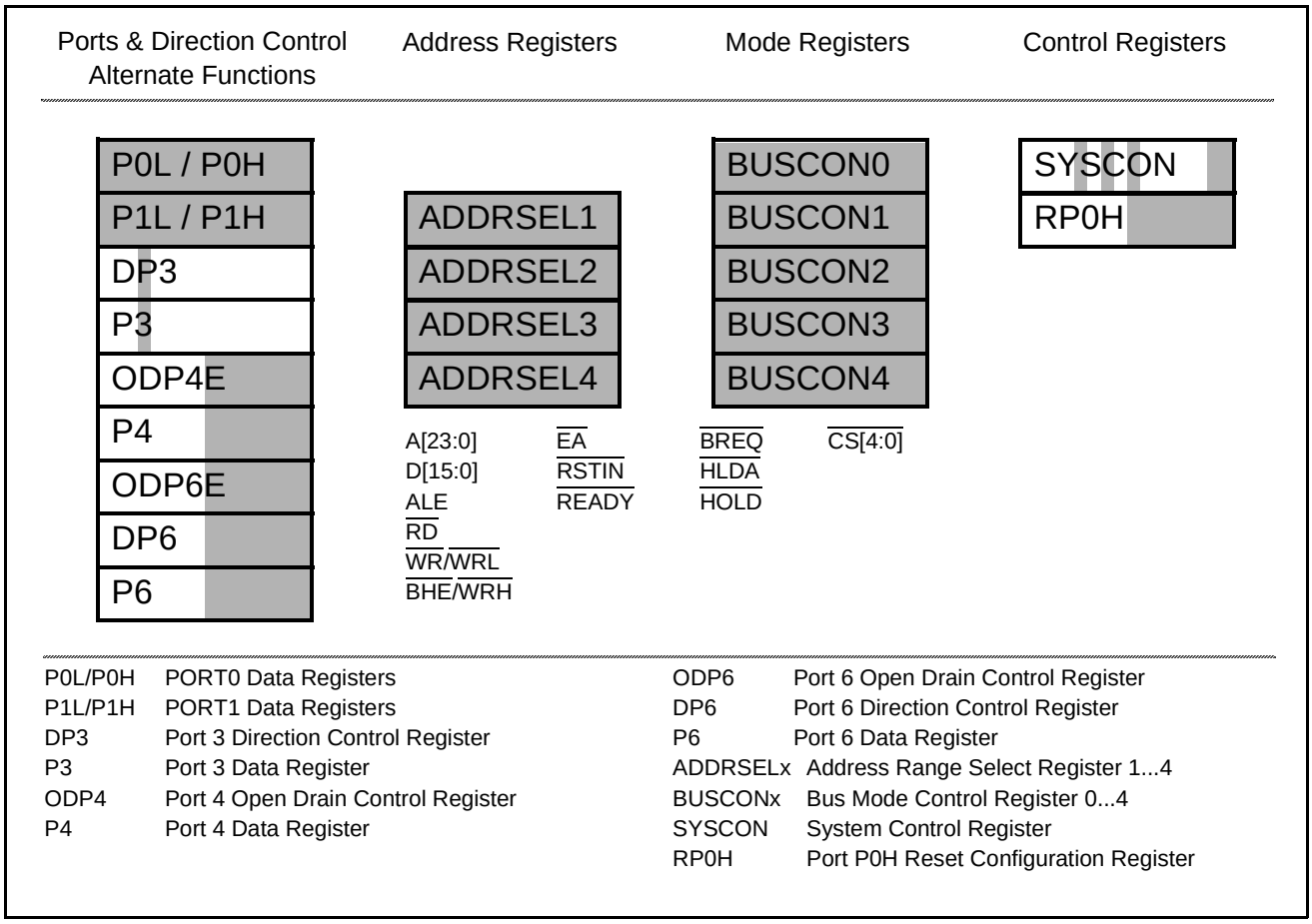


Figure 8-1 SFRs and Port Pins Associated with the External Bus Interface

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, 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 / READY control / ALE / RW delay). These parameters are used for accesses within a specific address area that is defined via the corresponding register ADDRSELx.

The four pairs BUSCON1/ADDRSEL1...BUSCON4/ADDRSEL4 make it possible to define four independent "address windows", while all external accesses outside these windows are controlled via BUSCON0.

8.1 Single-chip Mode

Single-chip mode is entered when the signal conf_start_external_n_i is high during reset. In this case, BUSCON0 is initialized with 0000_H, which also resets bit BUSACT0, so no external bus is enabled.

In single-chip mode, the C166S operates only with and out of internal resources. No external bus is configured and no external peripherals and/or memory can be accessed. No port lines are used 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 C166S to start execution out of the internal program memory (Mask-ROM, OTP, DRAM, SRAM 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 enabled explicitly by software.

8.2 External Bus Modes

When the external bus interface is enabled (bit BUSACTx=1) and configured (bitfield BTYP), the C166S uses a subset of its port lines together with some control lines to build the external bus.

Table 8-1 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 conf_rst_bustyp_i[1:0] during reset, provided that the signal conf_start_external_n_i is low during reset. Otherwise, BUSCON0 may be programmed via software just like the other BUSCON registers.

The 16-MByte address space of the C166S is divided into 256 segments of 64 KBytes each. The 16-bit intra-segment address is output on PORT0. 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 16 MByte), 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.

8.2.1 Multiplexed Bus Modes

In the multiplexed bus modes, the 16-bit intra-segment address and the data both use PORT0. The address is time-multiplexed with the data, and has to be latched externally. The width of the required latch depends on the selected data bus width. For example, 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 segment 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 (ALE) signal 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 appropriate 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 that is 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.

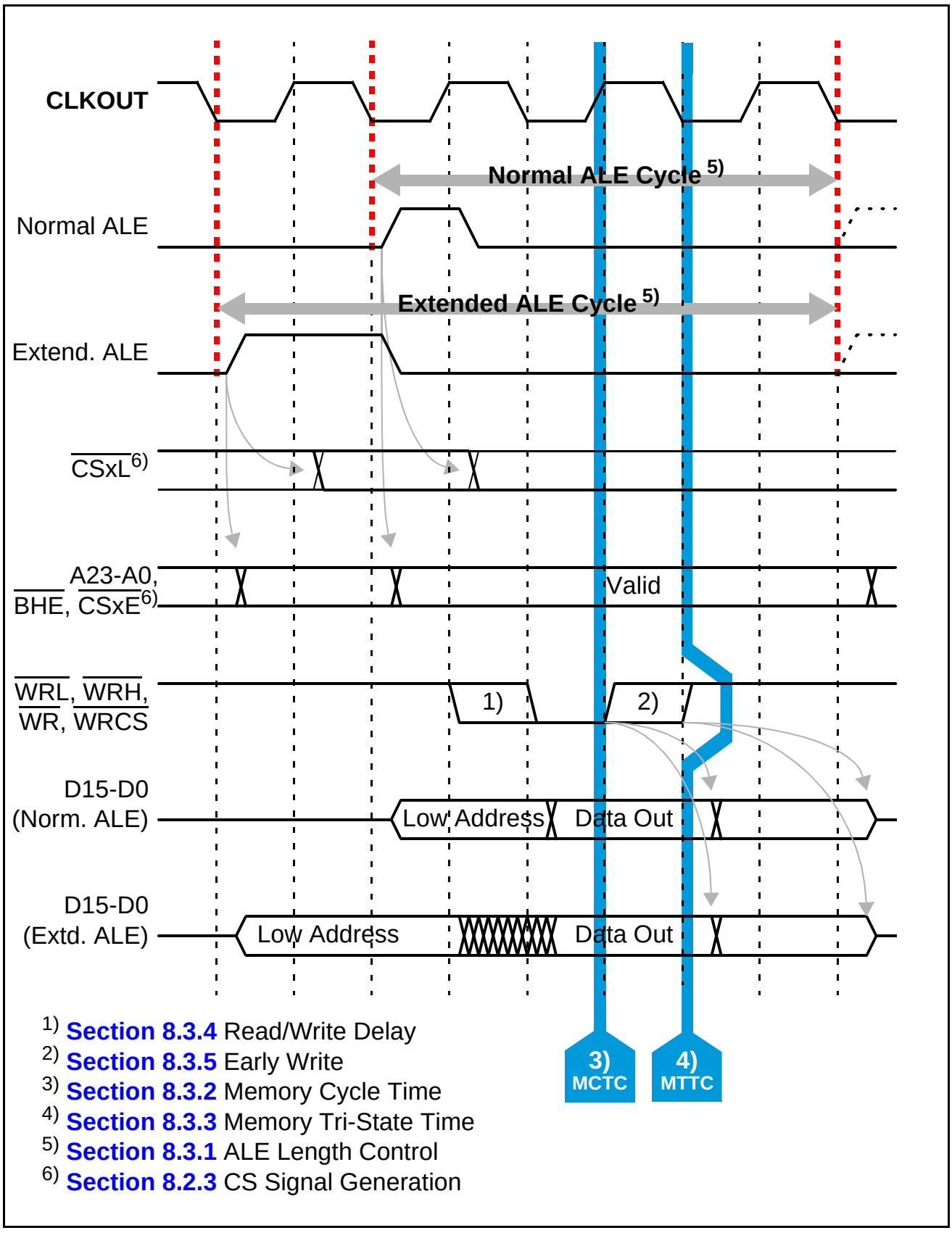


Figure 8-2 Multiplexed Bus, Write Access

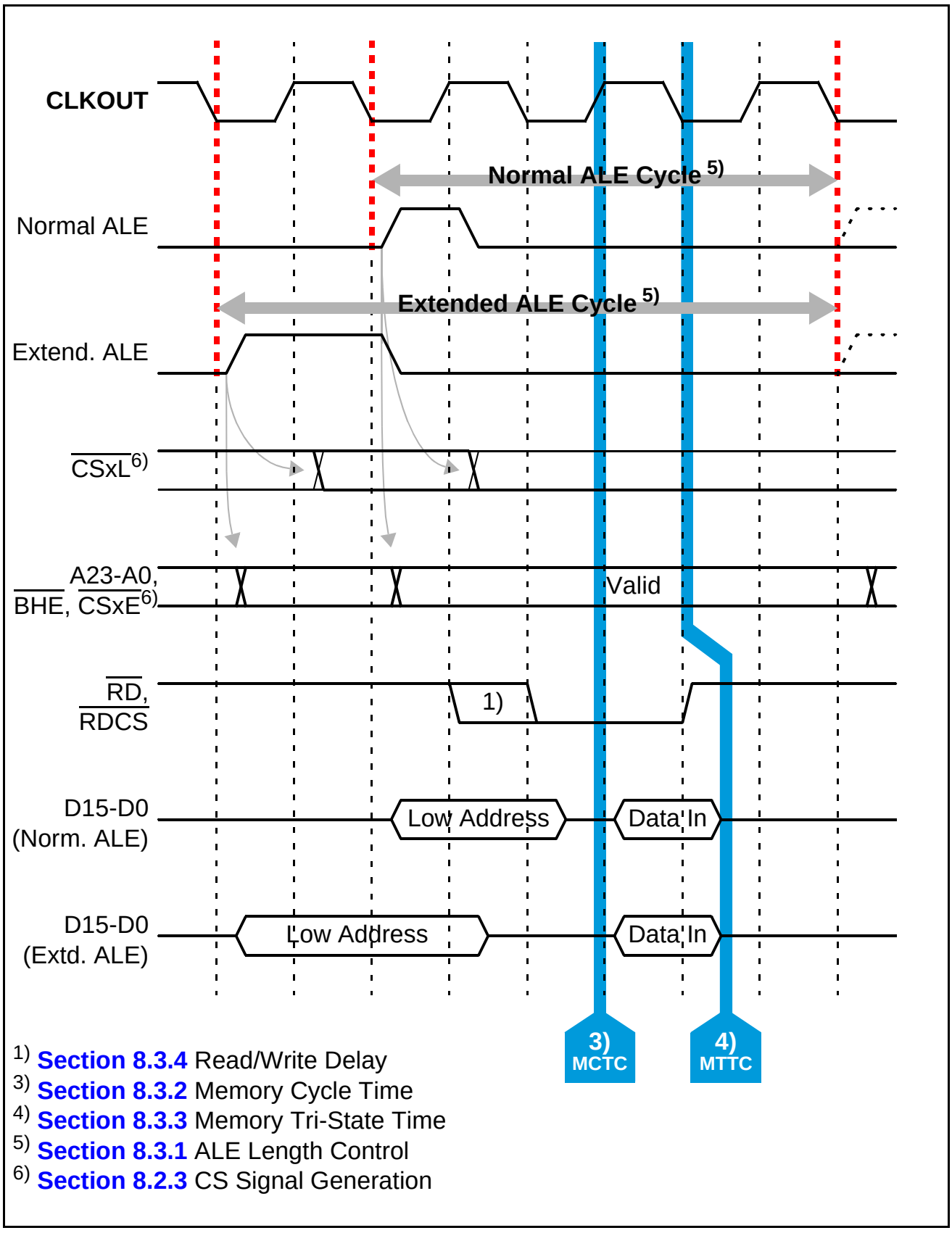


Figure 8-3 Multiplexed Bus, Read Access

8.2.2 Demultiplexed Bus Modes

In the demultiplexed bus modes, the 16-bit intra-segment address is permanently output on PORT1, while 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 appropriate 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 becomes 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 relevant address on the address bus. The data remain valid on the bus until the next external bus cycle is started.

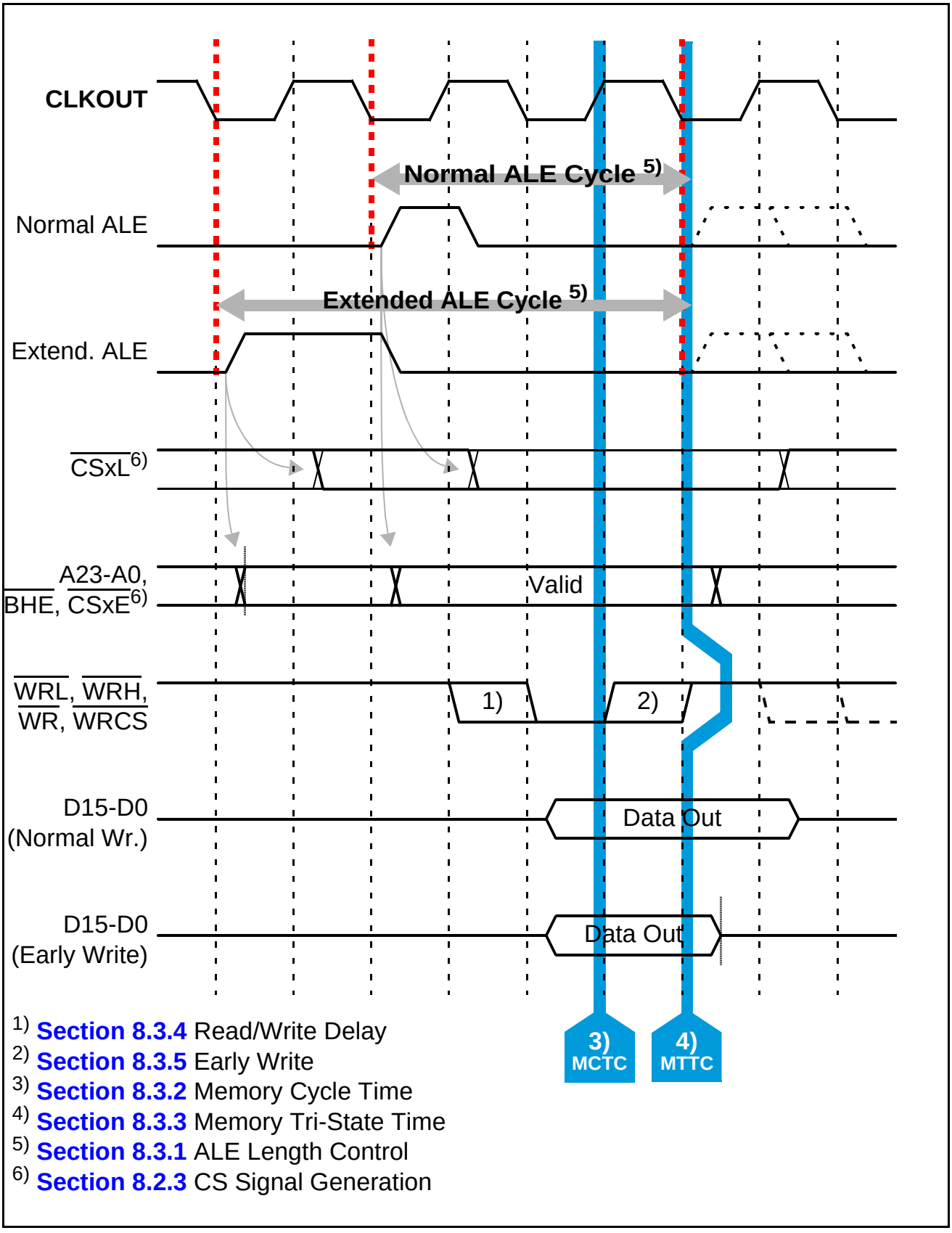


Figure 8-4 Demultiplexed Bus, Write Access

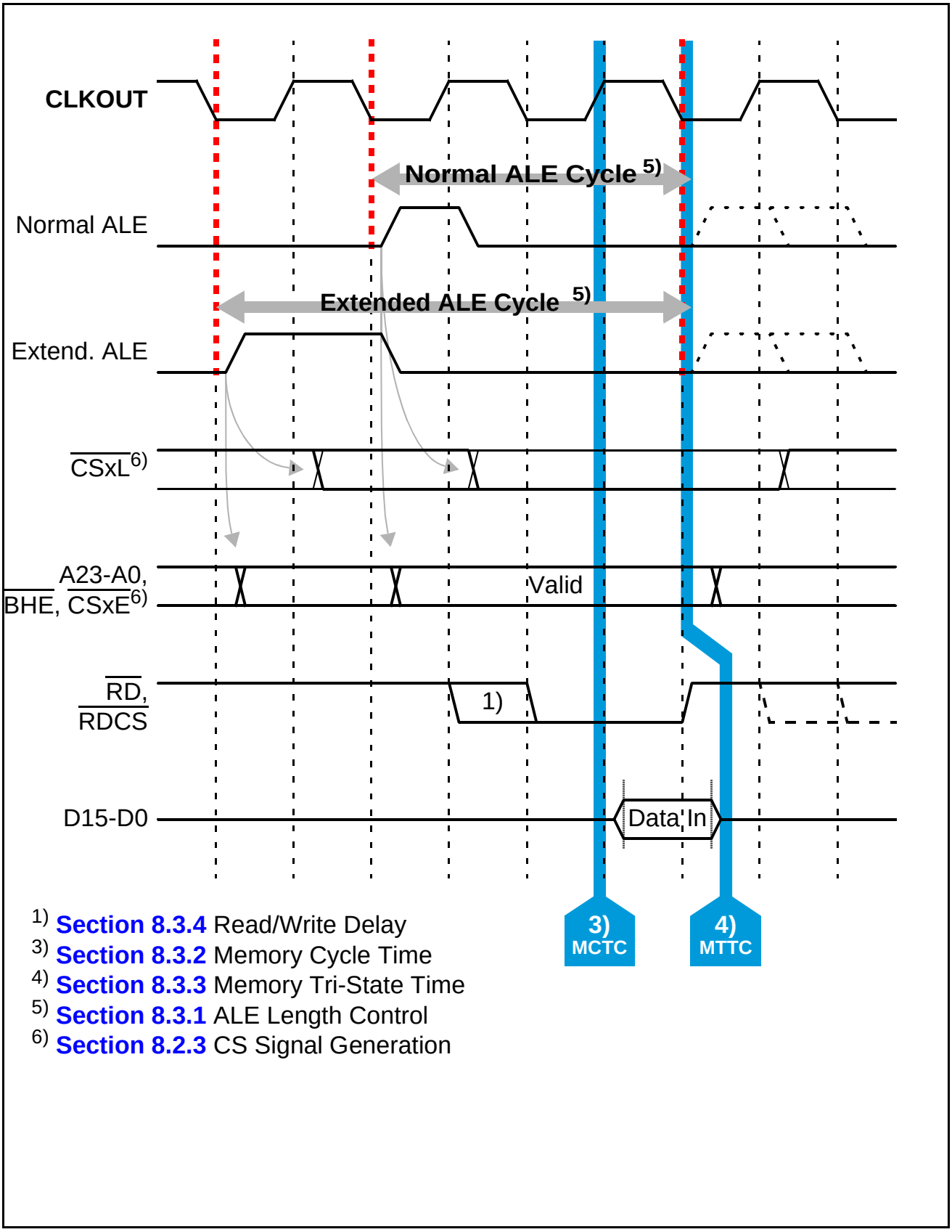


Figure 8-5 Demultiplexed Bus, Read Access

8.2.3 Switching Among the Bus Modes

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

A change of the external bus characteristics can be initiated in two different ways:

Reprogramming the BUSCON and/or ADDRSEL registers allows the bus mode to be changed for a given address window, or changing the size of an address window that uses a certain bus mode. Reprogramming makes it possible to use a great number of different address windows (more than BUSCONs are available), although there is some overhead for changing the registers and keeping appropriate tables.

Switching between predefined address windows automatically selects the bus mode that is associated with the respective window. Predefined address windows allow the use of different bus modes without any overhead, but restrict the number of windows to the number of BUSCONs. However, as BUSCON0 controls all address areas that are not covered by the other BUSCONs, there may be gaps between windows that 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 an external address decoder to be connected to PORT1 only, while using it for all kinds of bus cycles.

Note: Never change the configuration for an address area that currently supplies the instruction stream. Due to internal pipelining, it is very difficult to determine the first instruction fetch that will use the new configuration. Only change the configuration for address areas that are not currently accessed. This applies to BUSCON registers as well as to ADDRSEL registers.

The usage of the BUSCON/ADDRSEL registers is controlled via the addresses issued. When an access (code fetch or data) is initiated, the generated physical address determines whether the access is made internally, uses one of the address windows defined by ADDRSEL4...1, or 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 four address windows plus the default (BUSCON0) are to be used.

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, which delays the complete (multiplexed) bus cycle and extends the corresponding ALE signal (see **Figure 8-6**).

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.

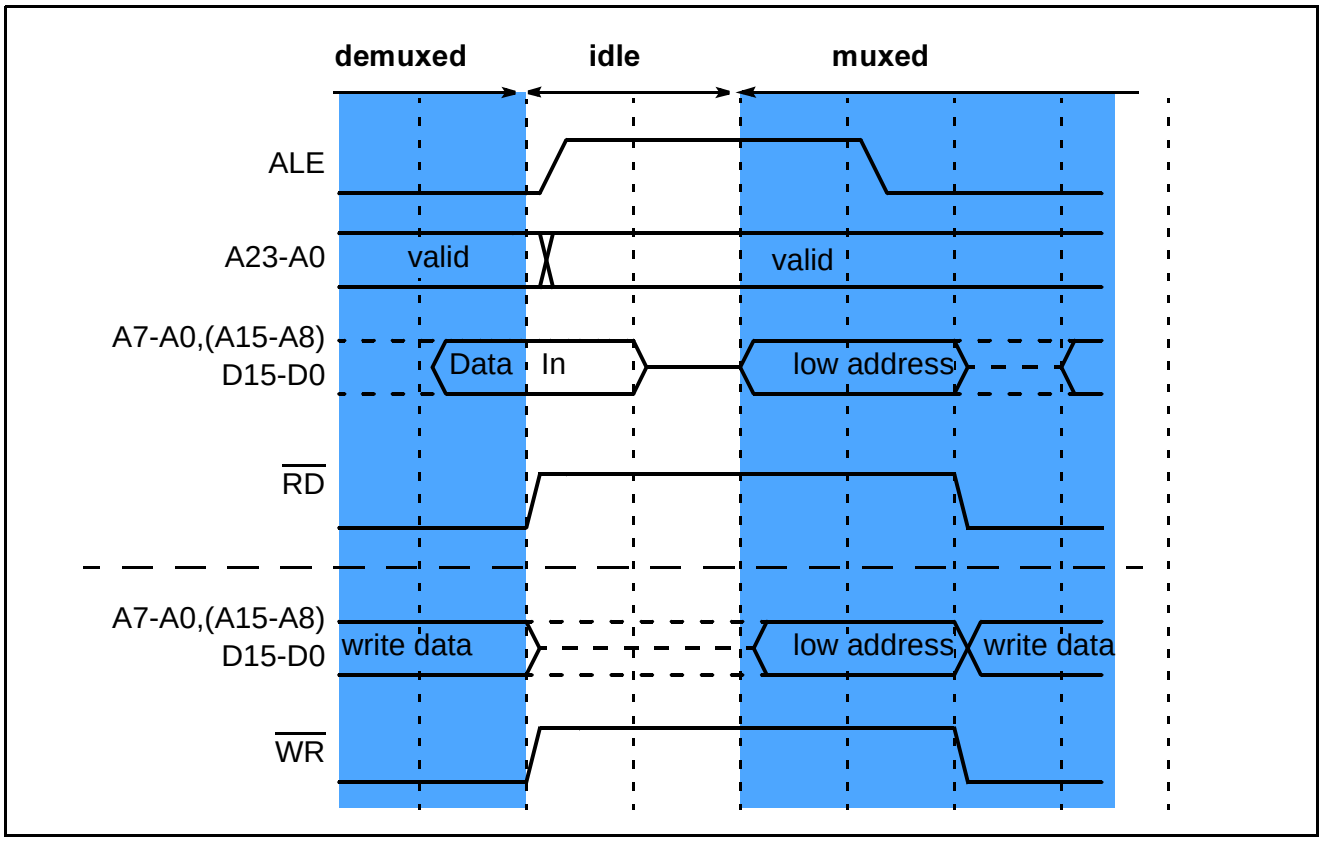


Figure 8-6 Switching from Demultiplexed to Multiplexed Bus Mode

Switching between external resources (e.g., different peripherals) may create a problem if the previously-accessed resource needs some time to switch off its output drivers (after a read), and if the resource to be accessed next switches on its output drivers very fast. In systems running on higher frequencies, this may lead to a bus conflict (switch-off delays normally are 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 8-6**).

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. If the following cycle accesses a different window, a waitstate is 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 only uses P0L, the lower byte of PORT0. This saves on address latches, bus transceivers, bus routing, and memory-related increases in 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 in which the low byte is accessed first, then the high byte. The assembly of bytes to 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 Byte High Enable $\overline{BHE}$ signal, while the lower byte is selected with the A0 signal. The two bytes of the memory can be enabled independently of each other, or together when accessing words.

When writing bytes to an external 16-bit device that has a single $\overline{CS}$ input and two $\overline{WR}$ enable inputs (for the two bytes), the EBC can generate these two write control signals directly. 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 C166S 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 8-2 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. PORT0H becomes available for general-purpose IO when only the 8-bit demultiplexed bus mode is selected.

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 a standard I/O pin. The $\overline{\text{BHE}}$ pin is used implicitly by the EBC to select one of two byte-organized memory chips, which are connected to the C166S via a word-wide external data bus. After reset, the $\overline{\text{BHE}}$ function is automatically enabled (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 if 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, which extend the 16-bit address output on PORT0 and thus increase the accessible address space. The number of segment address lines is selected via conf_rst_salsel_i[1:0] during reset, and coded in bit field SALSEL in register RP0H (see table below).

Table 8-3 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	Eight:	A23...A16	16	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 that 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. Make sure 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, which make it possible to select external peripherals or memory banks directly without requiring an external decoder. The number of $\overline{CS}$ lines is selected via conf_rst_cssel_i[1:0] during reset, and coded in bit field CSSEL in register RP0H (see **Table 8-4**).

Table 8-4 Decoding of Chip Select Lines

CSSEL	Chip Select Lines		Note
1 1	Five:	$\overline{CS4}$...$\overline{CS0}$	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 (i.e. when no external bus cycle is started), even if this area is covered by the respective ADDRSELx register. An internal bus interface access 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 may be operated in four different modes (see **Table 8-5**) that are selected via bits CSWENx and CSRENx in the respective BUSCONx register.

Table 8-5 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 ($\overline{CS}$ is renamed $\overline{WRCS}$ or $\overline{RDCS}$ in the protocol diagrams) 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 only for read cycles; write chip select is activated only for write cycles; and read/write chip select is activated for both read and write cycles (write cycles are assumed if either of the signals $\overline{WRH}$ or $\overline{WRL}$ goes active). These modes save external glue logic when accessing external devices such as latches or drivers that have 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 ($\overline{CS}$ is renamed in $\overline{CSxL}$ in the protocol diagrams) (CSCFG=0) becomes active with the falling edge of ALE and becomes inactive at the beginning of an external bus cycle that 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 internal bus interface) are accessed.

- An **early** address chip-select signal ($\overline{CS}$ is renamed in $\overline{CSxE}$ in the protocol diagrams) (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. Unselected $\overline{CS}$ lines will enter the high-impedance state, and be available for general purpose I/O.

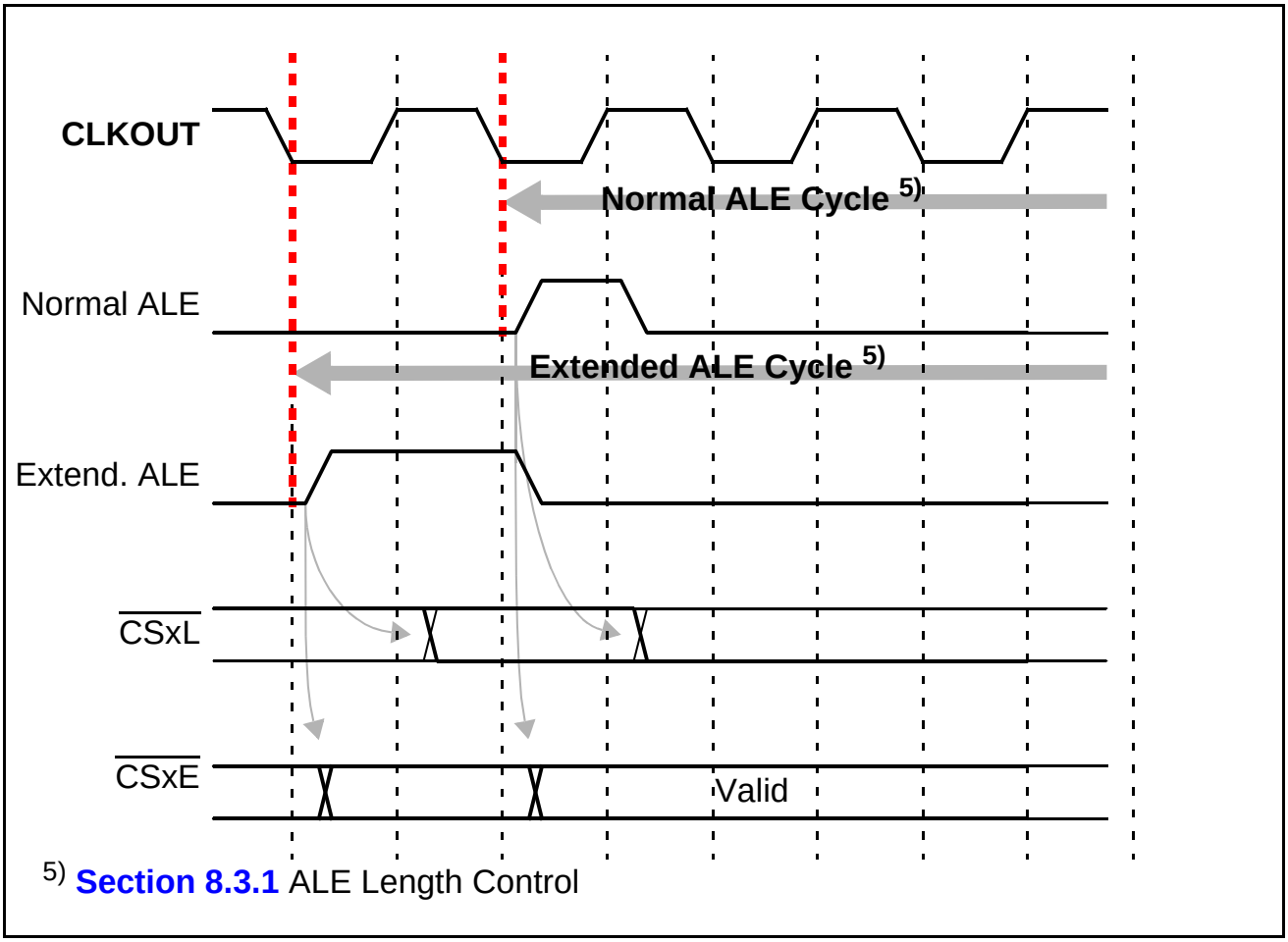


Figure 8-7 Latched and Early Chip Select

Segment Address versus Chip Select

The external bus interface of the C166S supports many configurations for the external memory. By increasing the number of segment address lines, the C166S can address a linear address space of 256 KByte, 1 MByte, 4 MByte, 8 MByte or 16 MByte. This allows implementation of a large sequential memory area, and access to a great number of external devices, using an external decoder. By increasing the number of $\overline{CS}$ lines, the C166S 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. A18...A16 **and** $\overline{CS4}$...$\overline{CS0}$), the segment address line function will take precedence.*

8.3 Programmable Bus Characteristics

Important timing characteristics of the external bus interface have been made user-programmable to adapt it to a wide range of 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: External accesses use the slowest possible bus cycle after reset. The bus cycle timing may then be optimized by the initialization software.

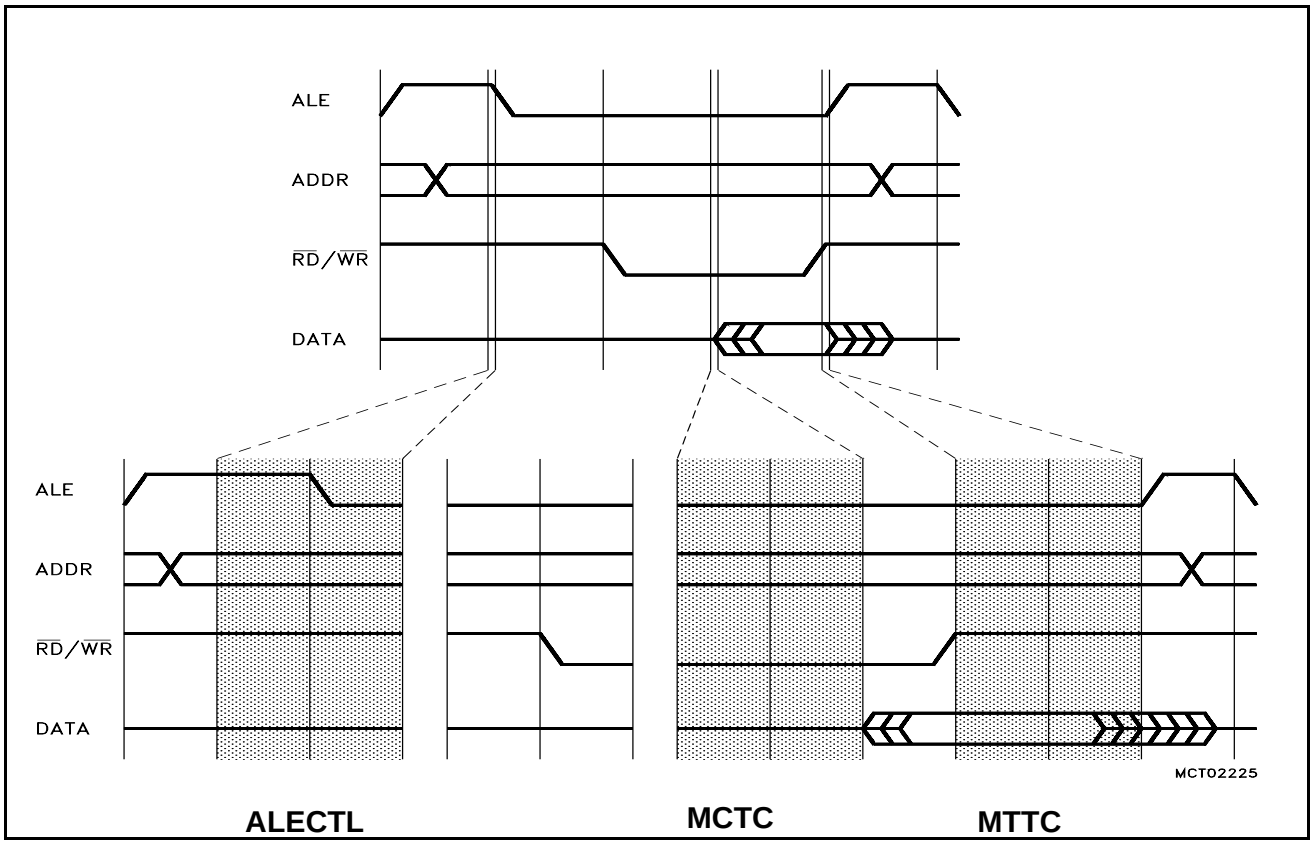


Figure 8-8 Programmable External Bus Cycle

8.3.1 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 appropriate address window will have their ALE signal 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 (i.e., 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 bits are 0 after reset.

8.3.2 Programmable Memory Cycle Time

The C166S 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.

The external bus cycles of the C166S can be extended by introducing waitstates during access (see **Figure 8-8**) to compensate for a memory or peripheral that cannot keep pace with the controller's maximum speed . During these memory cycle time waitstates, the CPU is idle if this access is required for the execution of the current instruction.

The memory cycle time waitstates can be programmed in increments of one CPU clock (2 TCL) within a range from 0 to 15 (default after reset) via the Memory Cycle Time Control (MCTC) fields of the BUSCON registers. 15-<MCTC> waitstates will be inserted.

8.3.3 Programmable Memory Tri-State Time

The C166S allows the user to adjust the time between two subsequent external accesses in order to account for the tri-state time of the external device. The tristate time defines when the external device has released the bus after deactivation of the read command ($\overline{RD}$).

The output of the next address on the external bus can be delayed by introducing a waitstate after the previous bus cycle in order to compensate for a memory or peripheral that needs more time to switch off its bus drivers, (see **Figure 8-8**). During this memory tri-state time waitstate, 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 tristate time waitstate requires one CPU clock (2 TCL) and is controlled via the Memory Tristate Time Control (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 tristate time waitstate in addition to the programmable MTTC waitstate.

8.3.4 Read/Write Signal Delay

The C166S 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) are concurrent (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 multiplexed bus modes, however, the data drivers of an external device may conflict with the C166S'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 Read Write Delay Control (RWDCx) bits in the BUSCON registers. The command(s) will be delayed if bit RWDCx is 0 (default after reset).

8.3.5 Early $\overline{WR}$

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 that operate on higher CPU clock frequencies and employ external modules (memories, peripherals, etc.) that switch on their own data drivers very rapidly in response to e.g. a chip select signal.

Conflicts between the C166S's and the external peripheral's output drivers can be avoided by selecting early $\overline{WR}$ for the C166S.

Note: Make sure that the reduced $\overline{WR}$ low time still meets the requirements of the external peripheral/memory.

Early $\overline{WR}$ deactivation is controlled via the Early Write EnNable (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, i.e. EWENx = 0).

8.3.6 $\overline{READY}$ Controlled Bus Cycles

For situations in which the programmable waitstates are not enough, or the response (access) time of a peripheral is not constant, the C166S has external bus cycles that are terminated via an asynchronous $\overline{READY}$ input signal. In this case, the C166S first inserts a programmable number of waitstates (0-7) and then monitors the $\overline{READY}$ line to determine the actual end of the current bus cycle. The external device drives $\overline{READY}$ low in order to indicate that data have been latched (write cycle) or are available (read cycle).

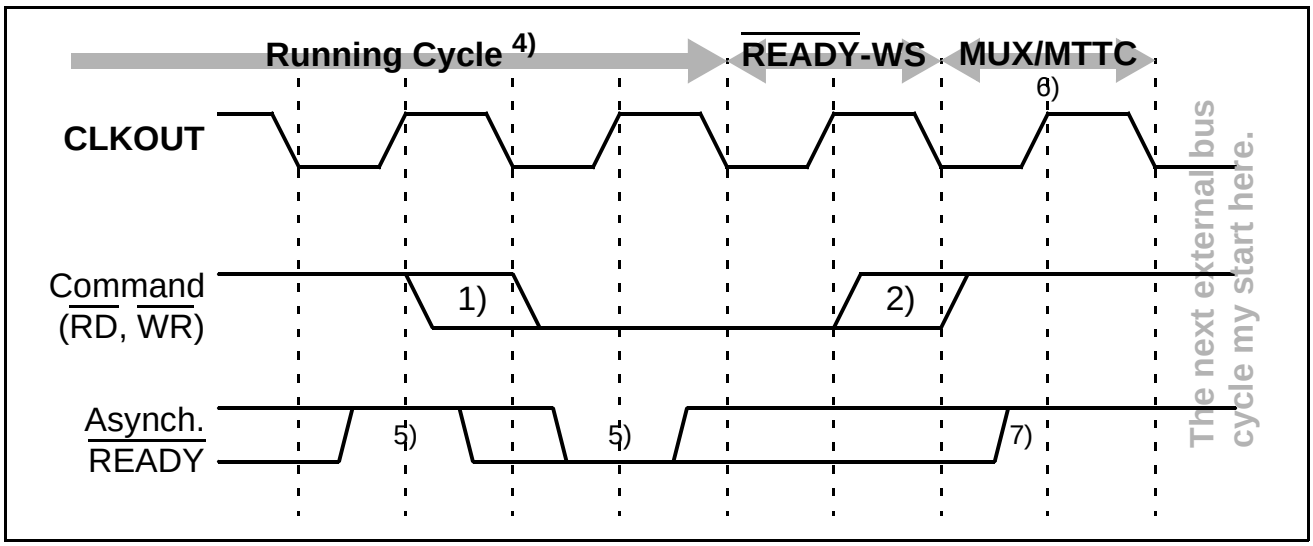


Figure 8-9 READY Controlled Bus Cycles

1) **Section 8.3.4** Read/Write Delay
2) **Section 8.3.5** Early Write
4) Cycle as programmed, including MCTC waitstates (Example shows 0 MCTC WS).
5) READY sampled HIGH at this sampling point generates a READY controlled waitstate, READY sampled LOW at this sampling point terminates the currently running bus cycle.
6) Multiplexed bus modes have a MUX waitstate added after a bus cycle, and an additional MTTC waitstate may be inserted here. For a multiplexed bus **with** MTTC waitstate, this delay is 2 CLKOUT cycles; for a demultiplexed bus **without** MTTC waitstate this delay is zero.
7) If the next following bus cycle is READY controlled, an active READY signal must be disabled before the first valid sample point for the next bus cycle. This sample point depends on the MTTC waitstate of the current cycle, and on the MCTC waitstates and the ALE mode of the next cycle. If the current cycle uses a multiplexed bus, the intrinsic MUX waitstate adds another CLKOUT cycle to the READY deactivation time.

The READY function is enabled via the ReaDY ENable (RDYENx) bits in the BUSCON registers. When this function is selected (RDYENx = 1), only the lower 3 bits of the respective MCTC bit field define the number of inserted waitstates (0-7), while the MSB of bit field MCTC is unused:

As shown in **Figure 8-9**, the asynchronous READY requires additional waitstates caused by the internal synchronization. The asynchronous READY is synchronized internally, and programmed waitstates may be necessary to provide proper bus cycles (see also notes on "normally-ready" peripherals below).

An asynchronous READY signal that has been activated by an external device may be deactivated in response to the trailing (rising) edge of the respective command (RD or WR).

Note: When the READY function is enabled for a specific address window, each bus cycle within this window must be terminated with an active READY signal. Otherwise, the controller hangs until the next reset. A time-out function is provided by the watchdog timer.

Combining the READY function with predefined waitstates is advantageous in two cases:

Memory components with a fixed access time and peripherals operating with READY may be grouped into the same address window. The (external) waitstate control logic in this case would activate READY either upon the memory's chip select or with the peripheral's READY output. After the predefined number of waitstates, the C166S will check its READY line to determine the end of the bus cycle. For a memory access, it will already be low; for a peripheral access, it may be delayed. As memories tend to be faster than peripherals, there should be no impact on system performance.

When using the READY function with so-called "normally-ready" peripherals, erroneous bus cycles may occur if the READY line is sampled too early. These peripherals pull their READY output low while they are idle. When they are accessed, they deactivate READY until the bus cycle is complete, then drive it low again. If, however, the peripheral deactivates READY after the first sample point of the C166S, the controller samples an active READY and terminates the current bus cycle too early. By inserting predefined waitstates, the first READY sample point can be shifted to an interval in which the peripheral has safely controlled the READY line.

8.4 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 memory mapping are controlled via register SYSCON. The properties of a bus cycle 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, which makes it possible to specify up to four address areas and the individual bus characteristics within these areas. All accesses that are not covered by these four areas are then controlled via BUSCON0. This allows the use of memory components or peripherals 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	SYSCON5	SYSCON4	SYSCON3	XPEN	VISIBLE	SYSCON0
rw			rw	rw	rwh	rwh	rw	rwh	rw	rwh	rwh	rwh	rw	rw	rwh

Field	Bits	Type	Description
SYSCON0	0	rwh	System Configuration Bit
VISIBLE	1	rw	**Visible Mode Control** 0: Accesses to XBUS peripherals are done internally 1: XBUS peripheral accesses are made visible on the external pins
XPEN	2	rw	**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
SYSCON3 ..... SYSCON5	3 5	rwh	System Configuration Bit

Field	Bits	Type	Description
CSCFG	6	rw	**Chip Select Configuration Control** 0: Latched CS mode. The CS signals are latched internally and driven to the (enabled) port pins synchronously. 1: Unlatched CS mode. The CS signals are directly derived from the address and driven to the (enabled) port pins.
WRCFG	7	rwh	**Write Configuration Control** (Set according to pin P0H.0 during reset) 0: Pins WR and BHE retain their normal function 1: Pin WR acts as WRL, pin BHE acts as WRH
CLKEN	8	rw	**System Clock Enable** (CLKOUT, cleard after reset) 0: CLKOUT disabled: pin may be used for general purpose IO 1: CLKOUT enabled: pin outputs the system clock signal
BYTDIS	9	rwh	**Disable/Enable Control for Pin BHE** (Set according to data bus width) 0: Pin BHE enabled 1: Pin BHE disabled, pin may be used for general purpose IO
ROMEN	10	rwh	**CPU Configuration Bit** **Internal ROM Enable** (Set according to pin EA during reset) 0 Internal Local Memory disabled: accesses to the Local Memory area use the external bus 1 Internal Local Memory enabled
SGTDIS	11	rw	**CPU Configuration Bit** **Segmentation Disable/Enable Control** 0 Segmentation enabled (CSP is saved/restored during interrupt entry/exit) 1 Segmentation disabled (Only IP is saved/restored)

Field	Bits	Type	Description
ROMS1	12	rw	**CPU Configuration Bit** **Internal Local Memory Mapping** 0 Internal Local Memory area mapped to segment 0 ($00'0000_H$...$00'7FFF_H$) 1 Internal Local Memory area mapped to segment 1 (01'0000H...01'7FFFH)
STKSZ	[15:13]	rw	**CPU Configuration Bit** **System Stack Size** Selects the size of the system stack (in the internal DPRAM) from 32 to 1024 words

Note: Register SYSCON cannot be changed after execution of the EINIT instruction.

The layout of the BUSCON registers and ADDRSEL registers is identical.

Registers BUSCON4...BUSCON1, which control the selected address windows, are completely under software control. Register BUSCON0, which is also used for the very first code access after reset, is partly controlled by hardware, i.e., it is initialized via dedicated configuration signals during the reset sequence. This hardware control allows an appropriate external bus to be defined for systems in which 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	0	RDYEN0	BSWC0	BUSACT0	ALECTL0	EWEN0	BTYP		MTTC0	RWDC0	MCTC			
rw	rw	r	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	0	RDYEN1	BSWC1	BUSACT1	ALECTL1	EWEN1	BTYP		MTTC1	RWDC1	MCTC			
rw	rw	r	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	0	RDYEN2	BSWC2	BUSACT2	ALECTL2	EWEN2	BTYP		MTTC2	RWDC2	MCTC			
rw	rw	r	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	0	RDYEN3	BSWC3	BUSACT3	ALECTL3	EWEN3	BTYP		MTTC3	RWDC3	MCTC			
rw	rw	r	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	0	RDYEN4	BSWC4	BUSACT4	ALECTL4	EWEN4	BTYP		MTTC4	RWDC4	MCTC			
rw	rw	r	rw	rw	rw	rw	rw	rw		rw	rw	rw			

Note: BUSCON0 is initialized with $00C0_H$ if conf_start_external_i is high during reset. If conf_start_external_i is low during reset, bits BUSACT0 and ALECTL0 are set (1)

and bit field BTYP is loaded with the bus configuration selected via conf_rst_bustyp_i[1:0].

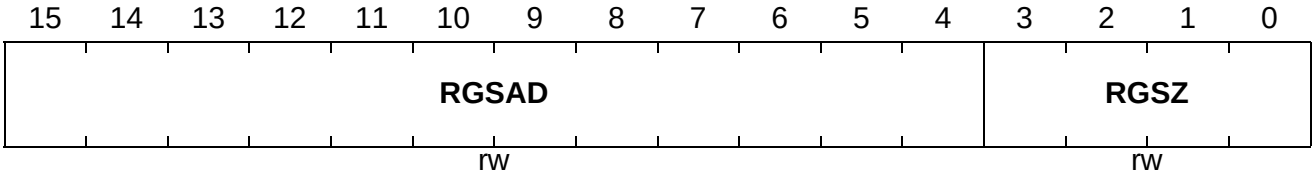
Field	Bits	Type	Description
MCTC	[3:0]	rw	**Memory Cycle Time Control** [1)](Number of memory cycle time waitstates) 0000: 15 waitstates . . . (Number = 15 - <MCTC>) 1111: No waitstates
RWDCx	4	rw	**Read/Write Delay Control for BUSCONx** 0: With rd/wr delay: activate command 1 TCL after falling edge of ALE 1: No rd/wr delay: activate command with falling edge of ALE
MTTCx	5	rw	**Memory Tristate Time Control** 0: 1 waitstate 1: No waitstate
BTYP	[7:6]	rw	**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 conf_rst_bustyp_i[1:0] during reset.*
EWENx	8	rw	**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	9	rw	**ALE Lengthening Control** 0: Normal ALE signal 1: Lengthened ALE signal
BUSACTx	10	rw	**Bus Active Control** 0: External bus disabled 1: External bus enabled within respective address window (ADDRSEL)

Field	Bits	Type	Description
BSWCx	11	rw	**BUSCON Switch Control** 0: Address windows are switched immediately 1: A tristate waitstate is inserted if the next bus cycle accesses a different window than the one controlled by this BUSCON register[2)]
RDYENx	12	rw	**READY Input Enable** 0: External bus cycle is controlled by bit field MCTC only 1: External bus cycle is controlled by the READY input signal
CSRENx	14	rw	**Read Chip Select Enable** 0: The CS signal is independent of the read command (RD) 1: The CS signal is generated for the duration of the read command
CSWENx	15	rw	**Write Chip Select Enable** 0: The CS signal is independent of the write cmd. (WR,WRL,WRH) 1: The CS signal is generated for the duration of the write command

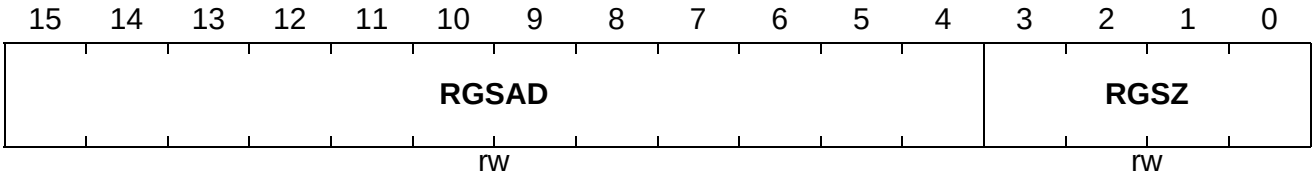
1) When the READY function is selected (RDYENx = 1), only the lower 3 bits of the respective MCTC bit field define the number of inserted waitstates (0-7), while the MSB of bit field MCTC is unused

2) A BUSCON switch waitstate is enabled by bit BUSCONx.BSWCx of the address window that is left.

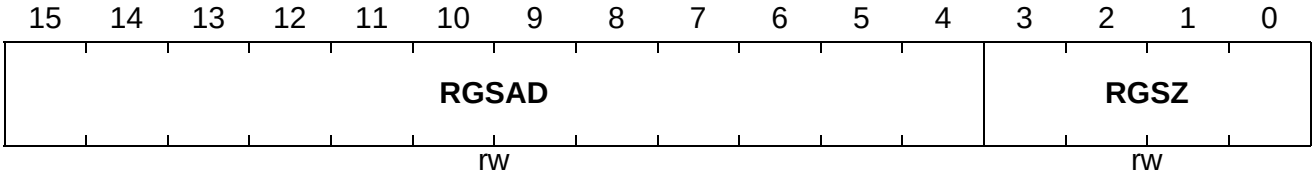
ADDRSEL1
Address Select Register 1 **SFR ($FE18_H/0C_H$)** **Reset value: 0000_H**



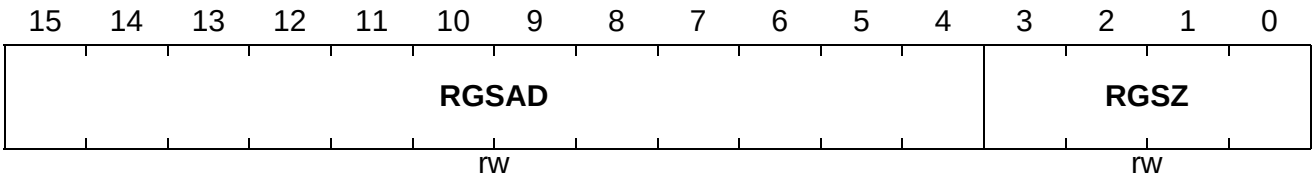
ADDRSEL2
Address Select Register 2 **SFR ($FE1A_H/0D_H$)** **Reset value: 0000_H**



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**



Field	Bits	Type	Description
RGSZ	[0:3]	rw	**Range Size Selection** Defines the size of the address area controlled by the respective BUSCONx/ADDRSELx register pair. See **Table 8-6** below.
RGSAD	[15:4]	rw	**Range Start Address** Defines the upper bits of the start address of the respective address area. See **Table 8-6** below.

Note: There is no register ADDRSEL0, as 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 4 address areas to be defined within the address space of the C166S. Within each of these address areas, external accesses can be controlled by one of the four different bus modes, independent of each other and of the bus mode specified in register BUSCON0. Each ADDRSELx register 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 below). For a given window size, only those upper address bits of the start address (marked "R") that are not implicitly used for addresses inside the window are used. The lower bits of the start address (marked "x") are disregarded.

Table 8-6 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 that can be defined within the C166S's address space may partly overlap each other. Thus small areas may be cut out of bigger windows, for example, in order to utilize external resources effectively, especially within segment 0.

For each access, the EBC compares the current address with all address-select registers (programmable ADDRSELx and hardwired/programmable XADRSx[1)]). This comparison is done in three levels. The XADRSx registers have the highest priority (priority I). The ADDRSEL registers have the second highest priority (priority II). If there is no match with any XADRSx or ADDRSELx register, the access to the external bus uses register BUSCON0 (priority III).

1) The XADR registers are the control registers of the internal XBUS interface (see **Section 8.7**).

Priority 1: The 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 of the XADRSx registers: XADR1 (priority I.1) , XADRS2 (I.2), XADRS3 (I.3), XADR4 (I.4) , XADRS5 (I.5), XADRS6 (I.6)

Priority 2: A match with one of the registers ADDRSELx directs the access to the respective external area using the corresponding BUSCONx register.
Priority of the ADDRSELx registers: ADDRSEL2 (priority II.1), ADDRSEL4 (II.2), ADDRSEL1 (II.3), ADDRSEL3 (II.4)

Priority 3: If there is no match with any XADRSx or ADDRSELx register, the access to the external bus uses BUSCON0.

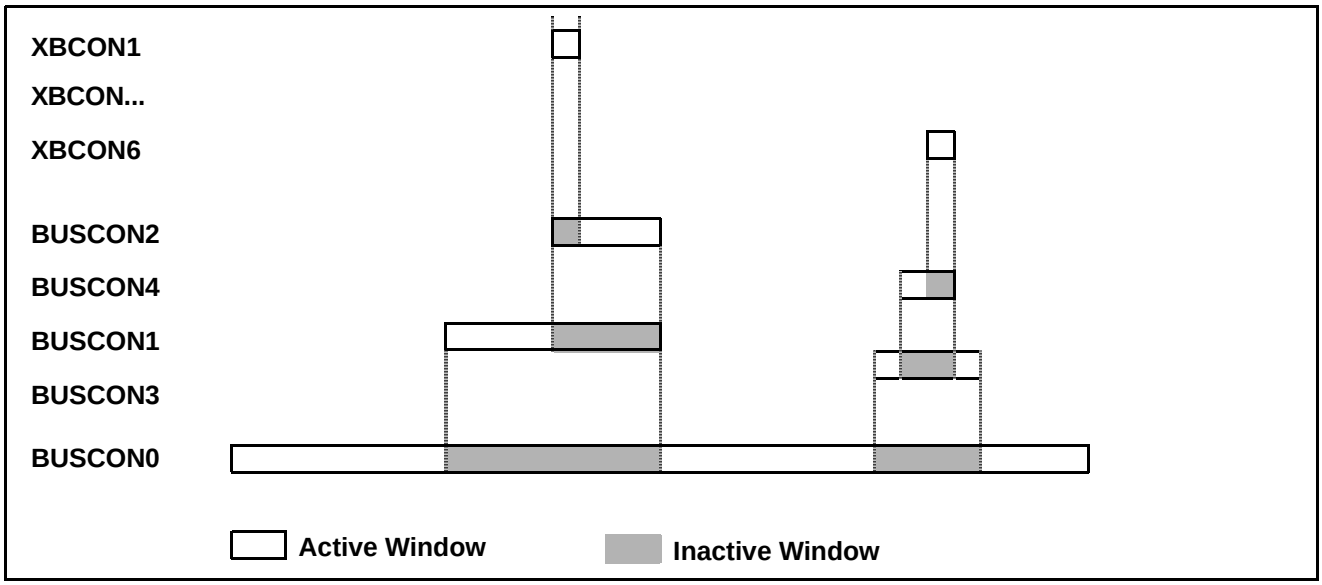


Figure 8-10 Address Window Arbitration Example

RP0H
Reset Value of P0H **SFR ($F108_H/84_H$)** **Reset value: - - XX_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								CLKCFG			SALSEL		CSSEL		WRC
											rh		rh		rh

Note: RP0H cannot be changed directly via software, but rather allows the current configuration to be checked.
Bitfields CLKCFG, SALSEL, and CSSEL may be reconfigured via register RSTCON.

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 put-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 put-downs)
CLKCFG	**Clock Generation Mode Configuration** These pins define the clock generation mode, i.e. the mechanism how the internal CPU clock is generated from the externally applied (XTAL1) input clock.

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.
- 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 8.5** EBC Idle State).

8.5 EBC Idle State

When the external bus interface is enabled, but no external access is currently being executed, the EBC is idle. As long as only internal resources (from an architecture point of view) such as DPRAM, GPRs, or SFRs, etc. are used, the external bus interface does not change (see**Table 8-7** below).

Accesses to on-chip X-Peripherals are also controlled by the EBC. However, even though an X-Peripheral appears to the controller as if it were an external peripheral, the 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 8-7** below). The "address" mentioned above includes Port 4, BHE, and ALE (which also pulses for an XBUS cycle). The external CS signals are driven inactive (high) because the EBC switches to an internal XCS signal.

The **external control signals** (RD and WR or WRL/WRH if enabled) **remain inactive** (high).

Table 8-7 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 CS signal corresponding to last used address	Inactive (high) for selected CS signals
BHE	Level corresponding to last external access	Level corresponding to last XBUS access
ALE	Inactive (low)	Pulses as defined for X-Peripheral
RD	Inactive (high)	**Inactive (high)**
WR/WRL	Inactive (high)	**Inactive (high)**
WRH	Inactive (high)	**Inactive (high)**

8.6 External Bus Arbitration

In high-performance systems, it may be efficient to share external resources such as memory banks or peripheral devices among more than one controller. The C166S supports this approach with the possibility to arbitrate the access to its external bus, i.e., to the external devices.

This bus arbitration allows an external master to request the C166S's bus via the HOLD input. The C166S acknowledges this request via the HLDA output and will float its bus lines in this case. The CS outputs provide internal pull-up devices. The new master may now access the peripheral devices or memory banks via the same interface lines as the C166S. During this time, the C166S can keep on executing, as long as it does not need

access to the external bus. All actions that just require internal resources such as instruction, data memory, or on-chip peripherals may be executed in parallel.

When the C166S needs access to its external bus while it is occupied by another bus master, it demands it via the BREQ output.

The external bus arbitration is enabled by setting bit HLDEN in register PSW to 1. The three bus arbitration pins HOLD, HLDA, and BREQ will be controlled automatically by the EBC independent of their I/O configuration. Bit HLDEN may be cleared during the execution of program sequences in which the external resources are required but cannot be shared with other bus masters. In this case, the C166S will not answer to HOLD requests from other external masters. If HLDEN is cleared while the C166S is in hold state (code execution from internal RAM/ROM), this hold state is left only after HOLD has been deactivated again. The current hold state will continue and only the next HOLD request is not answered.

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	HLD EN	0	0	0	USR0	MUL IP	E	Z	V	C	N
				rw	rw	r	r	r	rw		rwh	rwh	rwh	rwh	rwh

Field	Bits	Type	Description
HLDEN	[10]	rw	**External Bus Arbitration Control** 0_H External arbitration disabled 1_H External arbitration enabled

Connecting two C166Ss in this way would require additional logic to combine the respective output signals HLDA and BREQ. This can be avoided by switching one of the controllers into Slave Mode, in which pin HLDA is switched to input. This allows the slave controller to be connected directly to another master controller without glue logic. The Slave Mode is selected by setting bit DP6.7 to '1'. DP6.7='0' (default after reset) selects the Master Mode.

Note: The pins HOLD, HLDA and BREQ keep their alternate function (bus arbitration) even after the arbitration mechanism has been switched off by clearing HLDEN. All three pins are used for bus arbitration after bit HLDEN has been set once.

Connecting Bus Masters

When multiple C166Ss or a C166S and another bus master share external resources, some glue logic is required to define the currently active bus master, and to enable a

C166S that has surrendered its bus interface to regain control of it in case it must access the shared external resources. This glue logic is required if the other bus master does not automatically remove its hold request after having used the shared resources.

When two C166Ss are to be connected in this way, the external glue logic can be left out. One of the controllers must be operated in its Master Mode (default after reset, DP6.7=0) while the other one must be operated in its Slave Mode (selected with DP6.7=1).

In Slave Mode, the C166S inverts the direction of its HLDA pin and uses it as an input, while the master's HLDA pin remains an output. This approach does not require any additional glue logic for the bus arbitration (see **Figure 8-11** below).

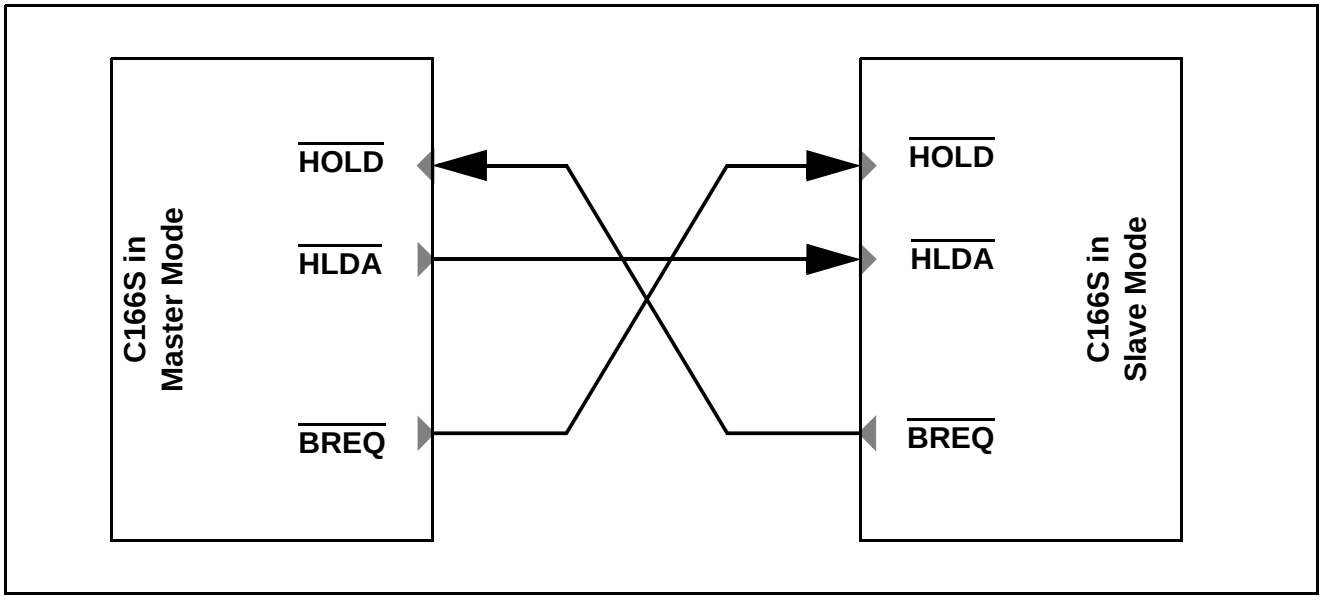


Figure 8-11 Sharing External Resources Using Slave Mode

When the bus arbitration is enabled (HLDEN=1), the three corresponding pins are controlled automatically by the EBC. Normally, the respective port direction register bits retain their reset value, which is 0. This selects Master Mode. Slave Mode is enabled by intentionally switching pin BREQ to output (DP6.7=1).

Entering the Hold State

Access to the C166S's external bus is requested by driving its HOLD input low. After synchronizing this signal, the C166S will complete a current external bus cycle (if any is active), release the external bus, and grant access to it by driving the HLDA output low. During hold state, the C166S treats the external bus interface as follows:

- Address and data bus(es) float to tristate
- ALE is pulled low by an internal pull-down device
- Command lines are pulled high by internal pull-up devices (RD, WR/WRL, BHE/WRH)
- CSx outputs are pulled high (push/pull mode) or float to tri-state (open-drain mode)

Should the C166S require access to its external bus during hold mode, it activates its bus request output BREQ to notify the arbitration circuitry. BREQ is activated only during hold mode. It will be inactive during normal operation.

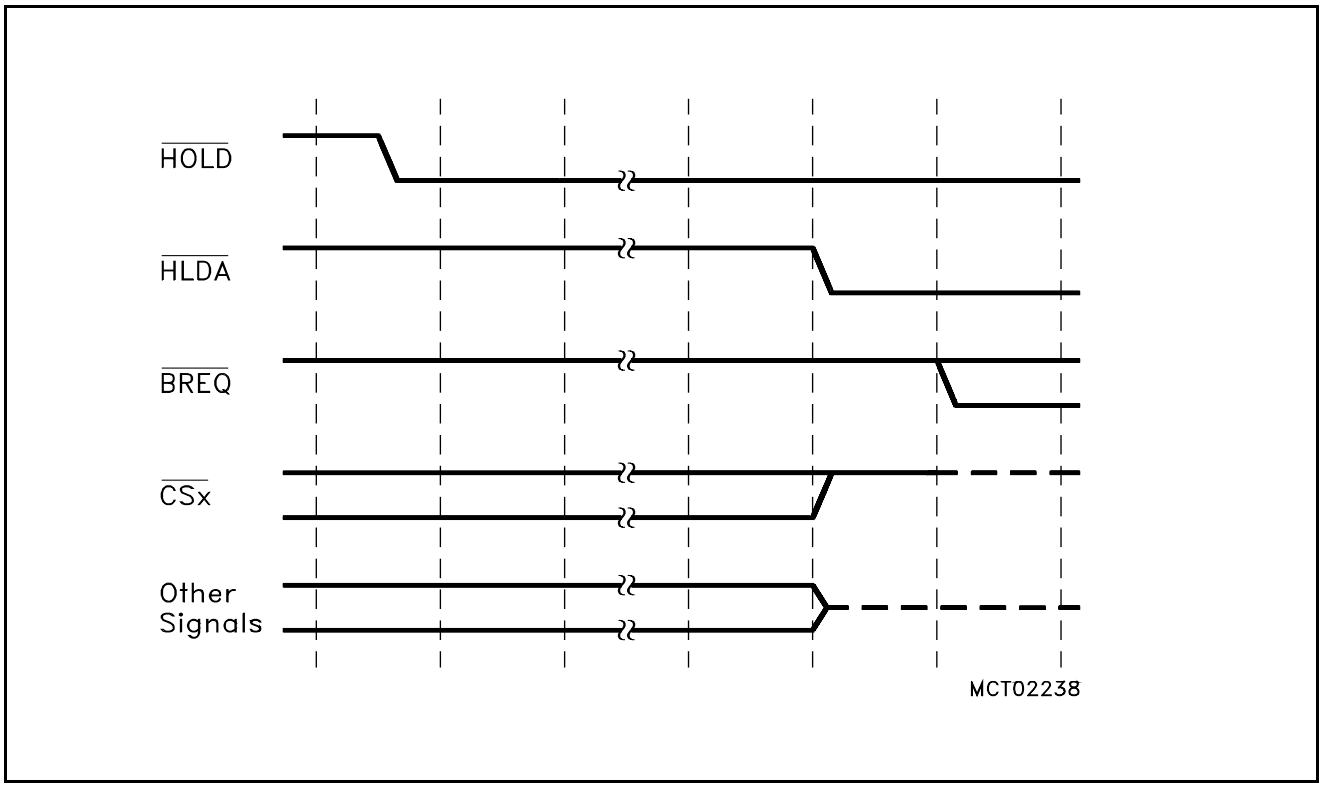


Figure 8-12 External Bus Arbitration, Releasing the Bus

*Note: The C166S will complete the bus cycle that is currently running before granting bus access, as indicated by the broken lines in **Figure 8-12**. This may delay hold acknowledge compared to this figure. The **Figure 8-12** shows the first possibility for BREQ to go active.*
During bus hold pin, P3.12 is switched back to its standard function and is then controlled by DP3.12 and P3.12. DP3.12 should be cleared and held at 0 to ensure floating in hold mode.

Exiting the Hold State

The external bus master returns the access rights to the C166S by driving the HOLD input high. After synchronizing this signal, the C166S will drive the HLDA output high, actively drive the control signals, and resume executing external bus cycles if required.

Depending on the arbitration logic, the external bus can be returned to the C166S under two circumstances:

- The external master no longer requires access to the shared resources and gives up its own access rights, or

- the C166S needs access to the shared resources and demands this by activating its BREQ output. The arbitration logic may then deactivate the other master's HLDA and so free the external bus for the C166S, depending on the priority of the different masters.

Note: The Hold State is not terminated by clearing bit HLDEN.

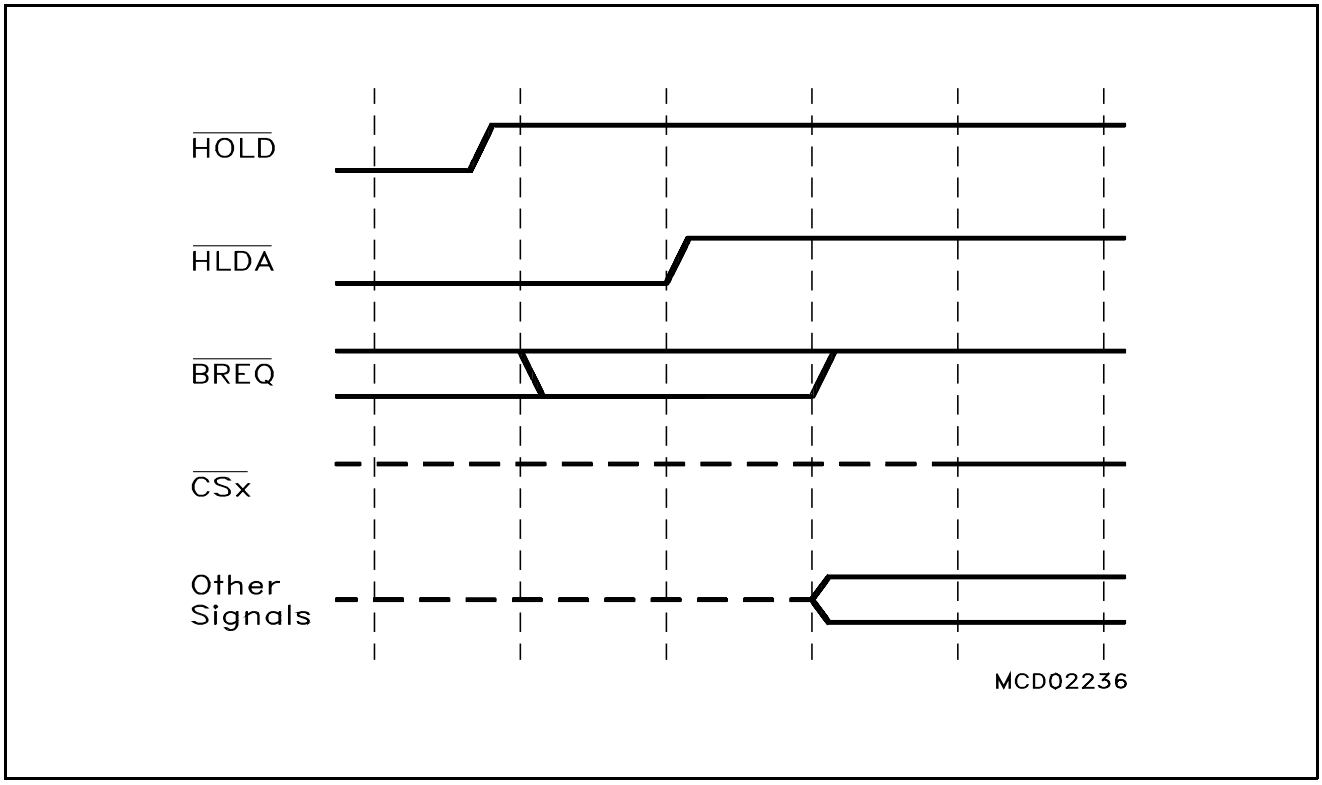


Figure 8-13 External Bus Arbitration, Regaining the Bus

Note: The falling BREQ edge indicates the last chance for BREQ to trigger the regain sequence. Even if BREQ is activated earlier, the regain sequence is initiated by HOLD going high. BREQ and HOLD are connected via an external arbitration circuitry. HOLD may also be deactivated without the C166S requesting the bus.

8.7 The XBUS Interface

The C166S provides an on-chip interface (the XBUS interface) by 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, i.e., it 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, most of the XADDRSEL registers select smaller address windows than

the standard ADDRSEL registers. As the register pairs control integrated peripherals rather than externally connected ones, most of the registers are fixed by mask programming rather than being user-programmable.

The XBUS provides byte-wide or word-wide X-Peripheral accesses. Because the on-chip connection can be very efficient, and for performance reasons, X-Peripherals are implemented only with a separate address bus, i.e., in demultiplexed bus mode. Interrupt nodes are provided for X-Peripherals to be integrated.

Enabling XBUS Peripherals

After reset, all on-chip XBUS peripherals are disabled. An XBUS peripheral cannot be used unless it has been enabled via the global enable bit XPEN in register SYSCON.

Additional to the XPEN bit, the XPERCON register define which on-chip peripherals are enabled or disabled (xpercon_o[15:0]). If a peripheral is disabled all it's addresses are no more visible.

The XPERCON register is accessible until execution of the EINIT instruction. XPER selection with the XPERCON register has to be performed before the selected X-peripherals are globally enabled via XPEREN-bit in SYSCON Register.

XPERCON
XPeripheral Control Register SFR ($F024_H/12_H$) Reset value: 0000_H

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

Field	Bits	Type	Description
PER0	[0]	rw	XPeripheral Enable Control for XCS1 0 XPeripheral disabled 1 XPeripheral enabled
PER1	[1]	rw	XPeripheral Enable Control for XCS2 0 XPeripheral disabled 1 XPeripheral enabled
PER2	[2]	rw	XPeripheral Enable Control for XCS3 0 XPeripheral disabled 1 XPeripheral enabled
PER3	[3]	rw	XPeripheral Enable Control for XCS4 0 XPeripheral disabled 1 XPeripheral enabled

Field	Bits	Type	Description
PER4	[4]	rw	XPeripheral Enable Control for XCS5 0 XPeripheral disabled 1 XPeripheral enabled
PER5	[5]	rw	XPeripheral Enable Control for XCS6 0 XPeripheral disabled 1 XPeripheral enabled
PERx	[15:6]	rw	XPeripheral Enable Control for XCSx, x=7..16 0 XPeripheral disabled 1 XPeripheral enabled

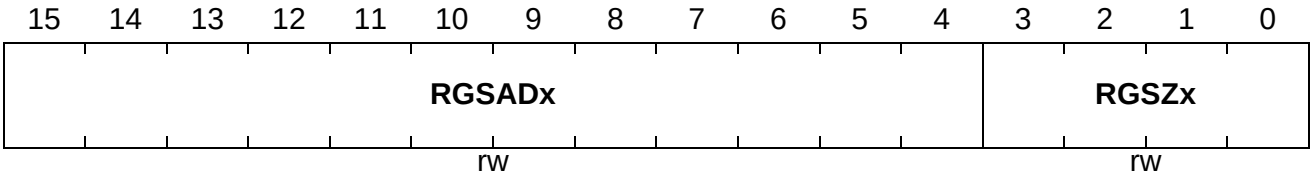
8.7.1 XBUS Access Control

In C166S up to six (configurable) address ranges with according bus definitions can be programmed for XBUS peripherals (including memories).

Address ranges and thus address mapping of memories or peripherals on XBUS are controlled with the address selection registers XADRSx. The respective bus type definitions are controlled with registers XBCONx.

The XADRS registers are defined as follows:

XADRS1(/2/3/4/5/6)
XBUS Address Selection Reg. **(Reset value: $XXXX_H$)**



Field	Bits	Type	Description
RGSADx	[15:4]	rw	**Address Range Start Address Selection**
RGSZx	[3:0]	rw	**Address Range Size Selection**

The respective SFR addresses of XADRSx registers can be found in list of SFRs.

Due to the different range size options, address mapping of XPERs is possible only within the first MByte of the total address range if XADRS1 to XADRS4 is used. The upper four address lines (A23:A20) are set to zero. Note that the range start address can be located only on boundaries specified by the selected range size.

The following table shows the definitions of range size selections and range start addresses for the address selection registers **XADRS1/2/3/4.**

The address range and address range start definition of **XADRS5 and XADRS6** registers is identical to the address selection definition for external devices (see **Address Window Definition**). It is thus possible to use the whole address range also for internal memories or peripherals.

Range Size RGSZ	Selected Address Range	Relevant(R) bits of RGSAD	Selected Range Start Address (Relevant(R) bits of RGSAD)
0000	256 Byte	RRRR RRRR RRRR	0000 RRRR RRRR RRRR 0000 0000
0001	512 Bytes	RRRR RRRR RRR0	0000 RRRR RRRR RRR0 0000 0000
0010	1 KBytes	RRRR RRRR RR00	0000 RRRR RRRR RR00 0000 0000
0011	2 KBytes	RRRR RRRR R000	0000 RRRR RRRR R000 0000 0000
0100	4 KBytes	RRRR RRRR 0000	0000 RRRR RRRR 0000 0000 0000
0101	8 KBytes	RRRR RRR0 0000	0000 RRRR RRR0 0000 0000 0000
0110	16 KBytes	RRRR RR00 0000	0000 RRRR RR00 0000 0000 0000
0111	32 KBytes	RRRR R000 0000	0000 RRRR R000 0000 0000 0000
1000	64 KBytes	RRRR 0000 0000	0000 RRRR 0000 0000 0000 0000
1001	128 KBytes	RRR0 0000 0000	0000 RRR0 0000 0000 0000 0000
1010	256 KBytes	RR00 0000 0000	0000 RR00 0000 0000 0000 0000
1011	512 KBytes	R000 0000 0000	0000 R000 0000 0000 0000 0000
11xx	- reserved		

Figure 8-14 Address Range and Address Range Start Definition of XADRS1/2/3/4 register

The XBCONx registers are defined as follows:

XBCON1 (/2/3/4/5/6)
XBUS Control Register **(Reset value: $XXXX_H$)**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	RDY ENx	BS WCx	BUS ACT x	0	0	B TYP x	0	1	1	MCTCx			
r	r	r	rw	rw	rw	r	r	rw	r	r	r	rw			

All XBCONx registers are located in the bitaddressable ESFR memory space. The respective SFR addresses of XBCON registers can be found in list of SFRs.

Field	Bits	Type	Description
MCTCx	[3:0]	rw	**Memory Cycle Time Control** [1] (number of wait states). See BUSCON
BTYPx	[7]	rw	**XBUS Type Definition** 0 8-bit Demultiplexed Bus 1 16-bit Demultiplexed Bus
BUSACTx	10	rw	**XBUS Active Control** 0 XBUS (peripheral) disabled 1 XBUS (peripheral) enabled Enables the XBUS and the according chip select $\overline{\text{XCS}}$x for the respective address window (respective XBUS peripheral), selected with XADRSx window. **Note:** Enable/Disable also is controlled with XPERCON and SYSCON registers.
BSWCx	11	rw	**BUSCON Switch Control** 0: Address windows are switched immediately 1: A tristate waitstate is inserted if the next bus cycle accesses a different window than the one controlled by this BUSCON register[2]
RDYENx	12	rw	**READY Enable** 0 The bus cycle length is controlled by bit field MCTC only 1 The bus cycle length is controlled by the peripheral using $\overline{\text{READY}}$ signal

1) When the READY function is selected (RDYENx = 1), only the lower 3 bits of the respective MCTC bit field define the number of inserted waitstates (0-7), while the MSB of bit field MCTC is unused

2) A BUSCON switch waitstate is enabled by bit BUSCONx.BSWCx of the address window that is left.

9 Watchdog Timer

To allow recovery from software or hardware failure, the User's Manual provides a Watchdog Timer [1]. If the software fails to service this timer before an overflow occurs, a watchdog timer reset can be initiated and a watchdog timer overflow can be signaled by wdtint_n_o.

When the watchdog timer reset is enabled (default) and the software has been designed to service it regularly before it overflows, the watchdog timer will supervise the program execution as it only will overflow if the program does not progress properly. The watchdog timer will also time out if a software error was due to hardware related failures. This prevents the controller from malfunctioning for longer than a user-specified time. The watchdog timer reset resets the CPU, Interruptcontroller, the External Bus Controller, the Control Block and the WDT itself.

The wdtint_n_o always shows the occurrence of a watchdog timer overflow. If the watchdog timer reset is disabled, the WDTINT signal can be used to trigger an interrupt. The wdtint_n_o signals can be directly connected to one interrupt control node. The watchdog timer can be used as a running timer and generates a periodical interrupt request with the occurrence of a timer overflow. In case of an overflow the WDT counter is automatically reloaded. Nevertheless the automatic reload is overruled in case of a WDT reset (wdt reset not disabled). The WDT can still be serviced with the execution of the srvwdt instruction.

Note: The WDT is automatically reloaded in case of a WDT overflow. In case of an enabled WDT reset, the generated reset resets the WDT and overrules the reload mechanism.

The watchdog timer provides two 16-bit registers and two subsystem signals:

- a read-only timer register that contains the current count,
- a control register for initialization and reset source detection,
- a conf_wdt_en_i subsystem signal to disable globally the WDT and
- a wdtint_o subsystem signal to signal a watchdog timer overflow.

The 16-bit watchdog timer is realized as two concatenated 8-bit timers. 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 upon each service access.

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

- by 2 (WDTIN = '0', WDTPRE = '0'), or
- by 4 (WDTIN = '0', WDTPRE = '1'), or

[1] The WDT can be globally disabled by the conf_wdt_en subsystem signal.

- by 128 (WDTIN = '1', WDTPRE = '0'), or
- by 256 (WDTIN = '1', WDTPRE = '1').

9.1 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. The 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 that indicate the source of a reset.

After any reset a globally enable watchdog timer (conf_wdt_en active) starts counting up from 0000_H with the default frequency $f_{WDT} = f_{PD}/256$. The default input frequency may be changed to another frequency ($f_{WDT} = f_{PD}/128$) by programming the prescaler (bit WDTIN).

The watchdog timer supports different modes:

- WDT Reset mode: When the watchdog timer is not disabled via instruction DISWDT it will continue counting up, even during 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 a watchdog timer reset and a wdt overflow will be signaled (wdtint) as well. But nevertheless the whole subsystem including the WDT itself will be reseted.
- WDT Interrupt mode: If the TIMEN bit is set, only the generation of watchdog timer resets is surpressed after the execution of the DISWDT instruction. A watchdog timer overflow event will still be signaled.
- WDT Disable mode: This mode is enable if the TIMer ENable TIMEN bit in the WDTCON register is not set, and the watchdog timer reset generation was stopped with the execution of the DISWDT instruction. In this case the whole WDT counter is stopped. Neither a watchdog timer reset will be generated nor a watchdog timer overflow (wdtint_o) event will be signaled.

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: The above described protection using the execution of EINIT or SRVWDT must be implemented inside the WDT block.

A watchdog reset will not 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. After being serviced the watchdog timer continues counting up from the value (<WDTREL> * 2^8).

Note: SRVWDT always triggers a timer reload independent of the execution of the EINIT and DISWDT instruction.

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

WDTCON
WDT Control Register **SFR ($FFAE_H/D7_H$)** **Reset value: $008X^{1)}_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WDTREL								WDT PRE	TIM EN	-	-	-	SW R	WDT R	WDT IN
-								rw	rw	-	-	-	rh	rh	rw

1) X=0xx1$_b$

Bit	Function
WDTIN	**Watchdog Timer Input Frequency Select** (combined with WDTPRE) Controls the input clock prescaler.
WDTR	**Watchdog Timer Reset Indication Flag**
SWR	**Software Reset Indication Flag**
TIMEN	**TIMer ENable TIMEN** If the TIMEN bit is set, only the generation of watchdog timer resets is surpressed after the execution of the DISWDT instruction. A watchdog timer overflow event will still be signaled.
WDTPRE	**Watchdog Timer Input Prescaler Control (combined with WDTIN)** Controls the input clock prescaler.
WDTREL	**Watchdog Timer Reload Value** (for the high byte of WDT)

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

- **the input frequency** to the watchdog timer can be selected via a prescaler controlled by bits WDTPRE and WDTIN in register WDTCON to be $f_{PD}/2$, $f_{PD}/4$, $f_{PD}/128$, or $f_{PD}/256$.
- **the 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:

$$P_{WDT} = \frac{2^{(1 + <WDTPRE> + <WDTIN>*6)} * (2^{16} - <WDTREL>*2^{8})}{f_{PD}}$$

Note: For safety reasons, the user is advised to rewrite WDTCON each time before the watchdog timer is serviced.

Table 9-1 Reset Indication Flag Combinations

Event	Reset Indication Flags[1)]	
	SWR	**WDTR**
Hardware Reset (HWRST)	0	0
Software Reset (SRST)	1	0
Watchdog Timer Reset (WDTRST)	0	1
HWRST and SRST	0	0
HWRST and WDTRST	0	0
HWRST, SRST and WDTRST	0	0
SRST and WDTRST	1	1

1) Description of table entries:
'1' = flag is set, '0' = flag is cleared.

Hardware Reset is indicated after a reset was triggered by a hardware event. In this case neither the SWR nor the WDTR bit is set. In case of a hardware reset, the software and watchdog timer reset are surpressed and not visible.

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

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

10 Asynchronous/Synchronous Serial Interface (ASC)

10.1 Introduction

This document describes the Asynchronous/Synchronous Serial Interface (ASC). The ASC supports a certain protocol to transfer data via a serial interconnection. It is also connected to a parallel bus of a microcontroller. The implementation is similar to the implementation in the C166 microcontrollers, however its parameters are changeable to work with parallel busses of different width and with different protocols.

Features

- Full duplex asynchronous operating modes
  - 8- or 9-bit data frames, LSB first
  - Parity bit generation/checking
  - One or two stop bits
  - Baudrate from 3.75 MBaud to 0.888 Baud (@ 60 MHz module clock f_{clk})
- Multiprocessor Mode for automatic address/data byte detection
- Loop-back capability
- Half-duplex 8-bit synchronous operating mode
  - Baudrate from 7.5 MBaud to 764.4 Baud (@ 60 MHz module clock f_{clk})
- Double buffered transmitter/receiver
- Interrupt generation
- on a transmitter buffer empty condition
- on a transmit last bit of a frame condition
- on a receiver buffer full condition
- on an error condition (frame, parity, overrun error)

Figure 10-1 shows all funtional relevant interfaces associated with the ASC Kernel.

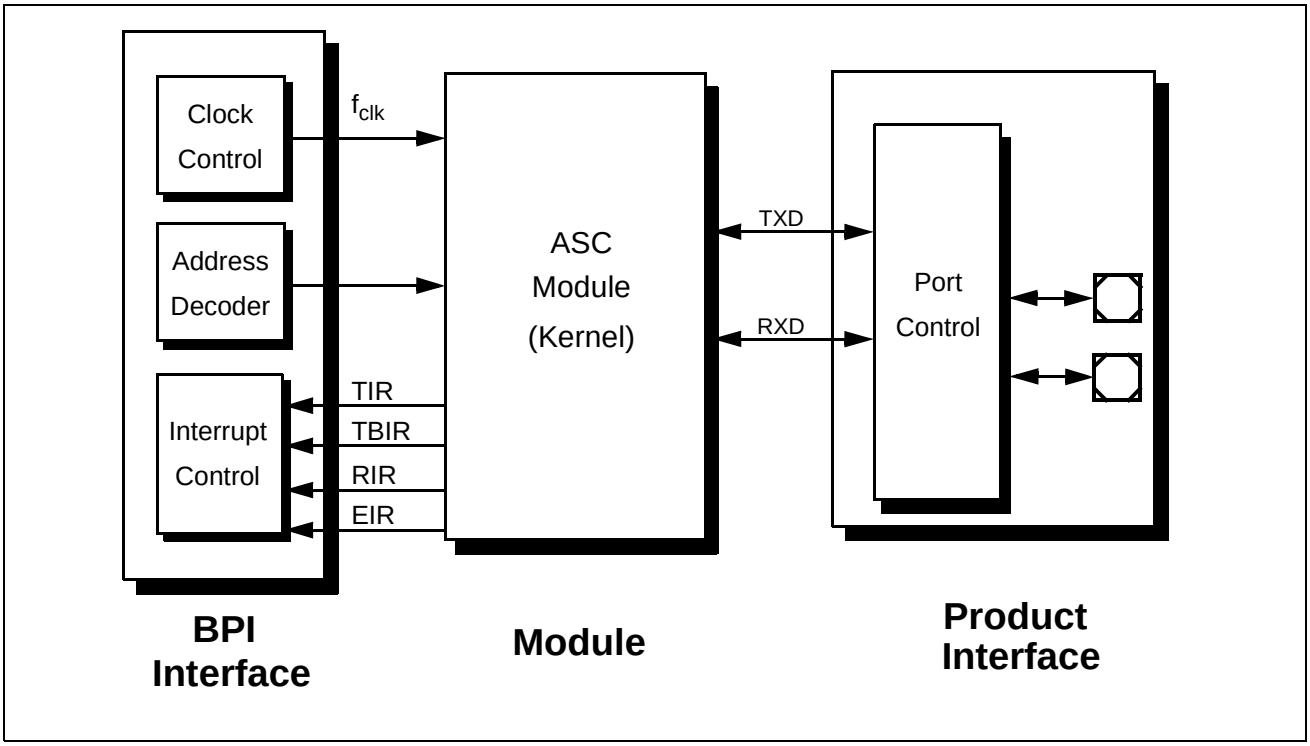


Figure 10-1 ASC Interface Diagram

Figure 10-2 shows all of the registers associated with the ASC Kernel.

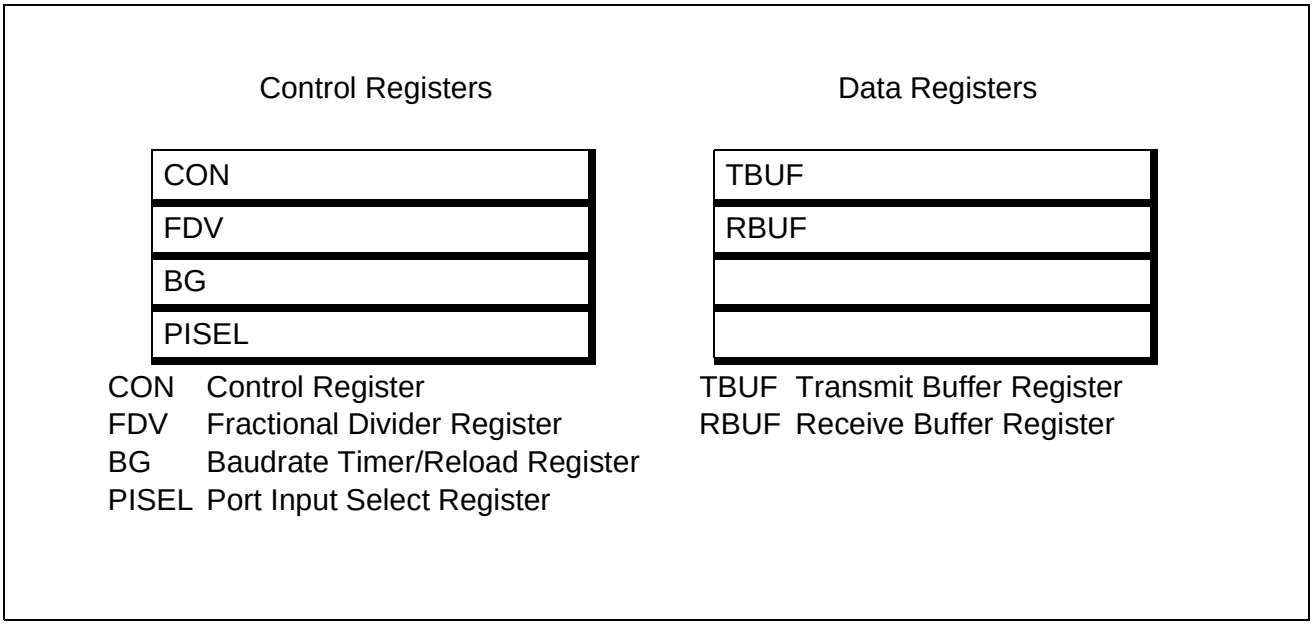


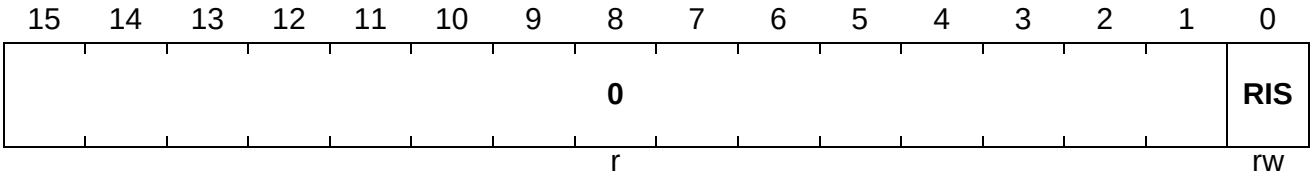
Figure 10-2 ASC Kernel Registers

All ASC registers are located in the SFR/ESFR memory space. The respective SFR addresses can be found in list of SFRs.

Port Input Select Register

The PISEL register controls the receiver input selection of the ASC module.

PISEL
Port Input Select Register **(Reset value: 0000_H)**



Field	Bits	Typ	Description
RIS	[0]	rw	**Receiver Input Select** 0 Receiver input RXDY0 selected 1 Receiver input RXDY1 selected
0	[15:1]	r	**Reserved** for future use; reading returns 0; writing to these bit positions has no effect.

10.2 Operational Overview

Figure 10-3 shows a block diagram of the ASC with its operating modes (asynchronous and synchronous mode).

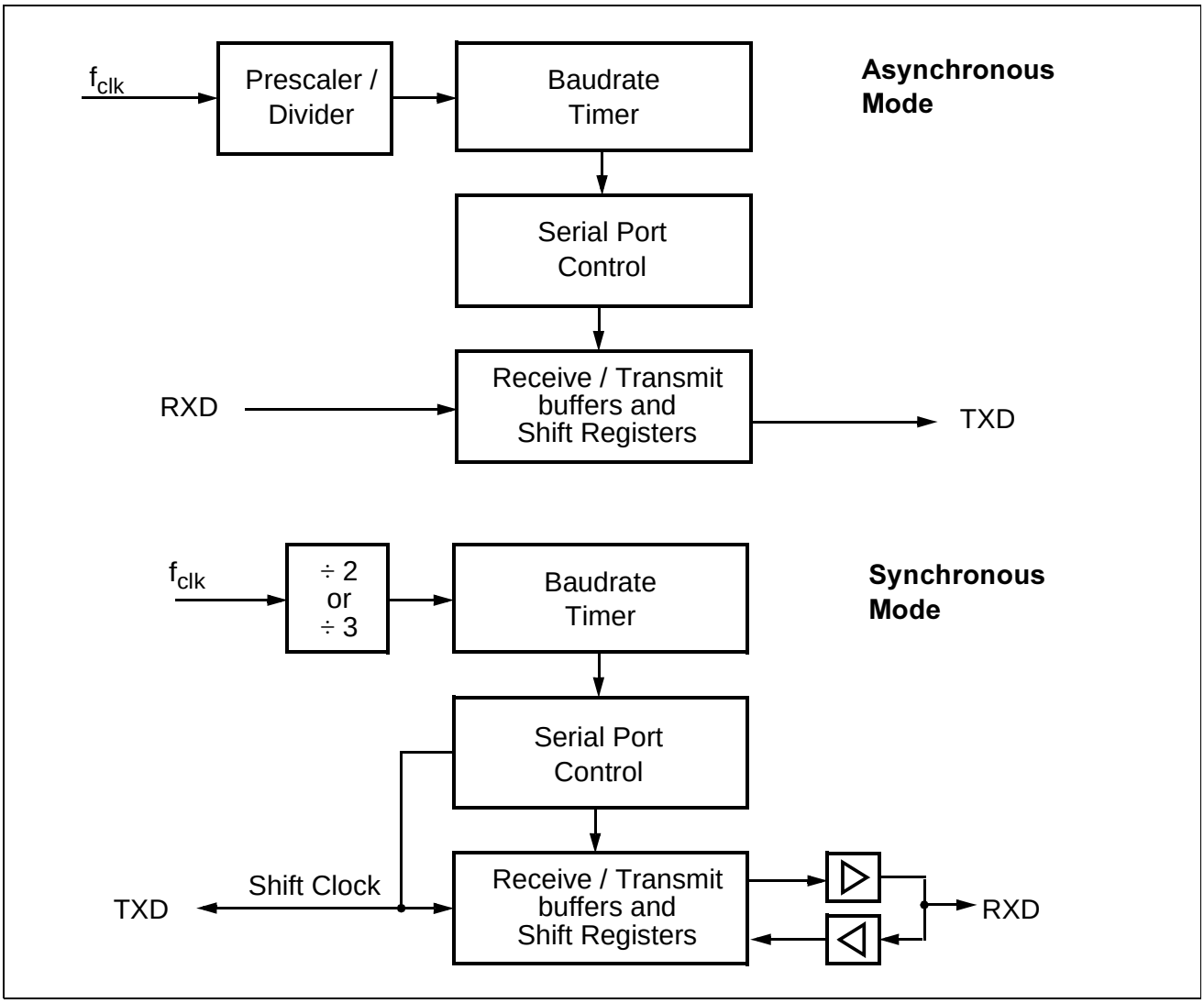


Figure 10-3 Block Diagram of the ASC

10.3 General Operation

The ASC supports full-duplex asynchronous communication up to 3.75 MBaud and half-duplex synchronous communication up to 7.5 MBaud (@ 60 MHz module clock). In Synchronous Mode, data are transmitted or received synchronous to a shift clock that is generated by the microcontroller. In Asynchronous Mode, either 8- or 9-bit data transfer, parity generation, and the number of stop bits can be selected. Parity, framing, and overrun error detection is provided to increase the reliability of data transfers. Transmission and reception of data is 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 baudrate timer with a versatile input clock divider circuitry provides the serial clock signal.

A transmission is started by writing to the Transmit Buffer register TBUF. The selected operating mode determines the number of data bits that will actually be transmitted, so that, bits written to positions 9 through 15 of register TBUF are always insignificant. Data transmission is double-buffered, so a new character may be written to 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 CON_REN. After reception of a character has been completed, the received data can be read from the (read-only) Receive Buffer register RBUF; the received parity bit can also be read if provided by the selected operating mode. Bits in the upper half of RBUF that are 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 already begin before the previously received character has been read out of the receive buffer register. In all modes, receive overrun error detection can be selected through bit CON_OEN. When enabled, the overrun error status flag CON_OE and the error interrupt request line EIR will be activated when the receive buffer register has not been read by the time reception of a ninth character is complete. The previously received character in the receive buffer is overwritten.

The Loop-Back option (selected by bit CON_LB) allows the data currently being transmitted to be received simultaneously in the receive buffer. This may be used to test serial communication routines at an early stage without having to provide an external network. In Loop-back Mode, the alternate input/output functions of the asosiacted Port pins are not necessary.

Note: Serial data transmission or reception is only possible when the Baudrate Generator Run Bit CON_R is set to 1. Otherwise, the serial interface is idle.
Do not program the mode control field COM_M to one of the reserved combinations to avoid unpredictable behavior of the serial interface.

The operating mode of the serial channel ASC is controlled by its control register CON. This register contains control bits for mode and error check selection, and status flags for error identification.

CON
Control Register **(Reset value: 0000_H)**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	LB	BRS	ODD	FDE	OE	FE	PE	OEN	FEN/ RXDI	PEN	REN	STP	M		
rw	rw	rw	rw	rw	rwh	rwh	rwh	rw	rw	rw	rwh	rw		rw	

Field	Bits	Typ	Description
M	[2:0]	rw	**Mode Control** 000 8-bit-data for synchronous operation 001 8-bit-data for asynchronous operation 010 Reserved. Do not use this combination 011 7-bit-data and parity for asynchronous operation 100 9-bit-data for asynchronous operation 101 8-bit-data and wake up bit for asynchronous operation 110 Reserved. Do not use this combination 111 8-bit-data and parity for asynchronous operation
STP	[3]	rw	**Number of Stop Bits Selection** 0 One stop bit 1 Two stop bits
REN	[4]	rwh	**Receiver Enable Bit** 0 Receiver disabled 1 Receiver enabled *Note: bit is cleared by hardware after reception of a byte in Synchronous Mode*
PEN	[5]	rw	**Parity Check Enable** All Asynchronous Modes without IrDA Mode: 0 Ignore parity 1 Check parity

Field	Bits	Typ	Description
FEN	[6]	rw	**Framing Check Enable** (Asynchronous Mode only) 0 Ignore framing errors 1 Check framing errors
OEN	[7]	rw	**Overrun Check Enable** 0 Ignore overrun errors 1 Check overrun errors
PE	[8]	rwh	**Parity Error Flag** Set by hardware on a parity error (PEN=1). Must be cleared by software.
FE	[9]	rwh	**Framing Error Flag** Set by hardware on a framing error (FEN=1). Must be cleared by software.
OE	[10]	rwh	**Overrun Error Flag** Set by hardware on an overrun error (OEN=1). Must be cleared by software.
FDE	[11]	rw	**Fractional Divider Enable** 0 Fractional divider disabled 1 Fractional divider enabled and used as perscaler for baudrate generator (bit BRS is don't care)
ODD	[12]	rw	**Parity Selection** 0 Even parity selected (parity bit of 1 is included in data stream on odd number of 1 and parity bit of 0 is included in data stream on even number of 1) 1 Odd parity selected (parity bit of 1 is included in data stream on even number of 1 and parity bit of 0 is included in data stream on odd number of 1)
BRS	[13]	rw	**Baudrate Selection** 0 Divide clock by reload-value+constant (depending on mode) 1 Additionally reduce serial clock to 2/3 *Note: BRS is don't care if FDE=1 (fractional divider selected)*

Field	Bits	Typ	Description
LB	[14]	rw	**Loopback Mode Enabled** 0 Loopback Mode disabled. Standard transmit/receive Mode 1 Loopback Mode enabled
R	[15]	rw	**Baudrate Generator Run Control Bit** 0 Baudrate generator disabled (ASC inactive) 1 Baudrate generator enabled *Note: BR_VALUE should only be written if R=0.*

10.3.1 Asynchronous Operation

Asynchronous Mode supports full-duplex communication in which both transmitter and receiver use the same data frame format and the same baudrate. Data is transmitted on line TXD and received on line RXD. **Figure 10-4** shows the block diagram of the ASC when operating in Asynchronous Mode.

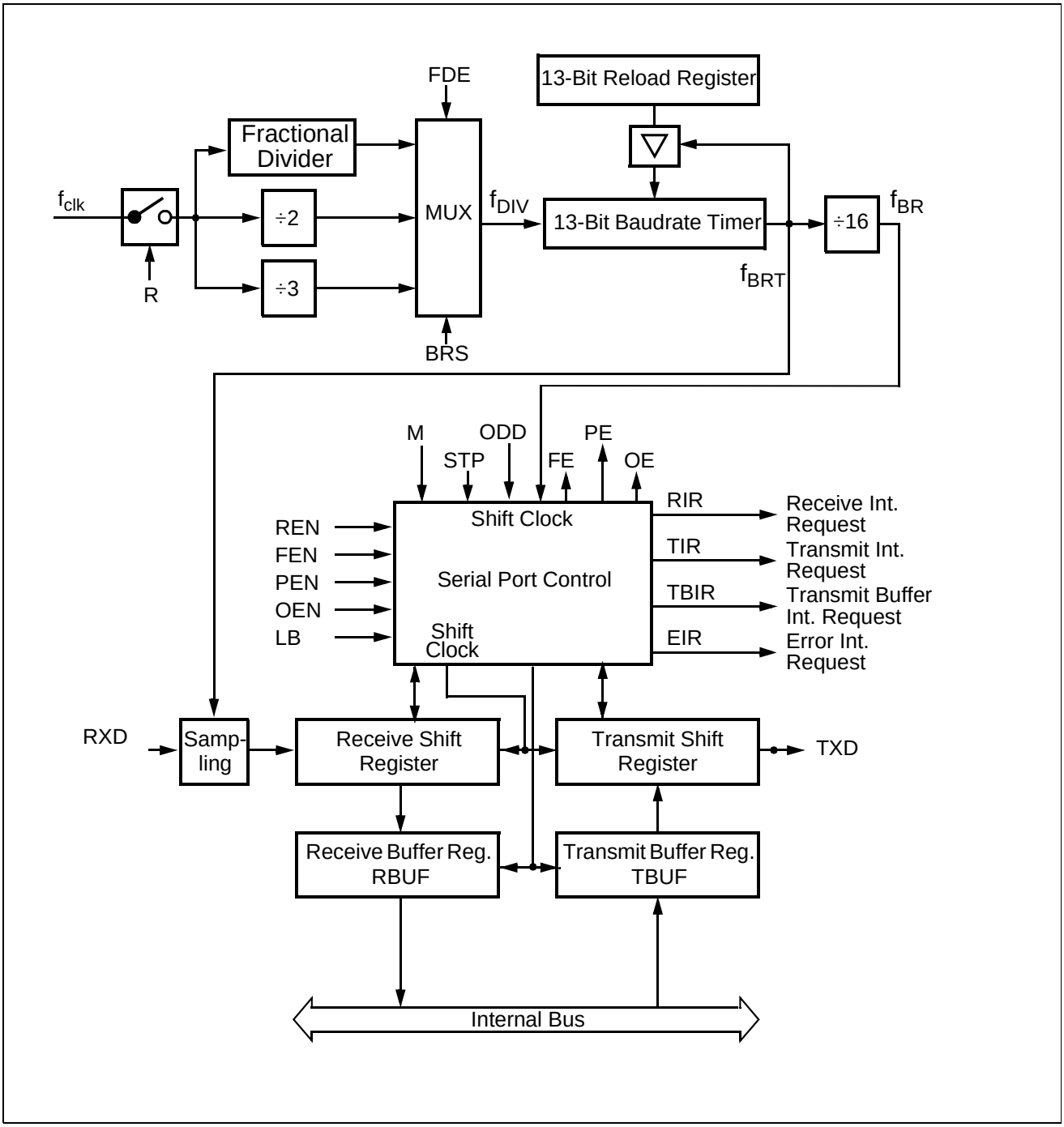


Figure 10-4 Asynchronous Mode of Serial Channel ASC

10.3.1.1 Asynchronous Data Frames

8-Bit Data Frames

8-bit data frames consist of either eight data bits D7...D0 (CON_M='001_B'), or seven data bits D6...D0 plus an automatically generated parity bit (CON_M='011_B'). Parity may be odd or even, depending on bit CON_ODD. An even parity bit will be set if the modulo-2-sum of the 7 data bits is 1. An odd parity bit will be cleared in this case. Parity checking is enabled via bit CON_PEN (always OFF in 8-bit data mode). The parity error flag CON_PE will be set, along with the error interrupt request flag, if a wrong parity bit is received. The parity bit itself will be stored in bit RBUF.7.

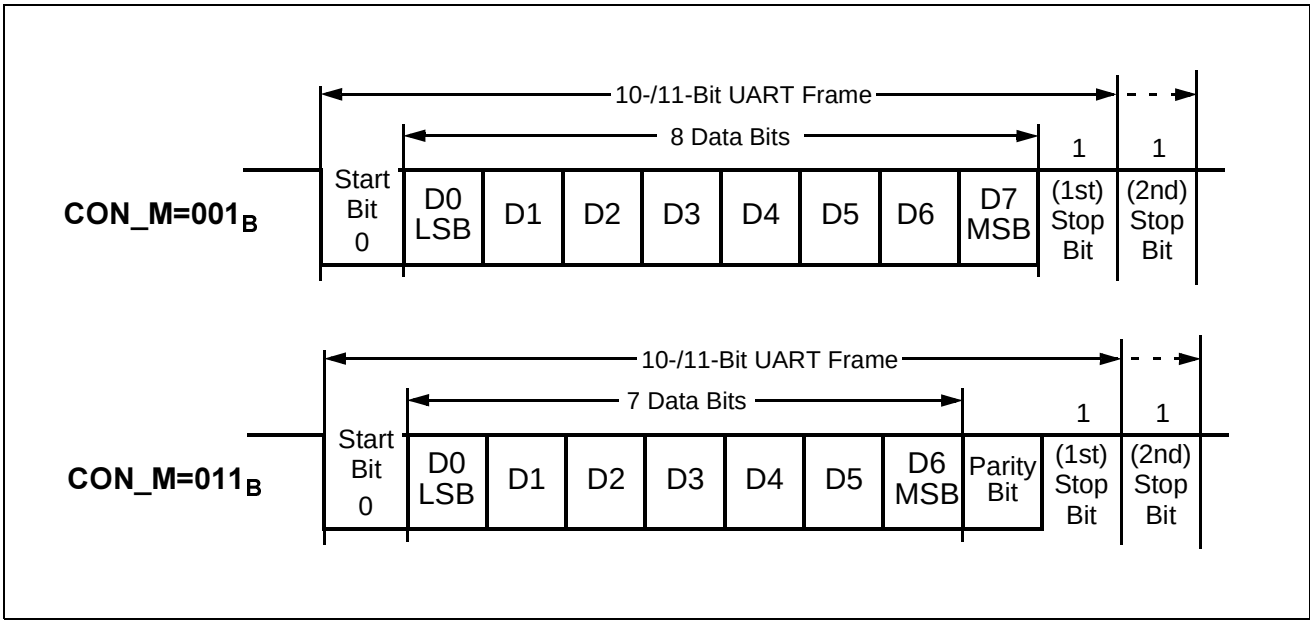


Figure 10-5 Asynchronous 8-Bit Frames

9-Bit Data Frames

9-bit data frames consist of either nine data bits D8...D0 (CON_M='100_B'), eight data bits D7...D0 plus an automatically generated parity bit (CON_M='111_B'), or eight data bits D7...D0 plus wake-up bit (CON_M='101_B'). Parity may be odd or even, depending on bit CON_ODD. An even parity bit will be set if the modulo-2-sum of the 8 data bits is 1. An odd parity bit will be cleared in this case. Parity checking is enabled via bit CON_PEN (always OFF in 9-bit data and wake-up mode). The parity error flag CON_PE will be set, along with the error interrupt request flag, if a wrong parity bit is received. The parity bit itself will be stored in bit RBUF.8.

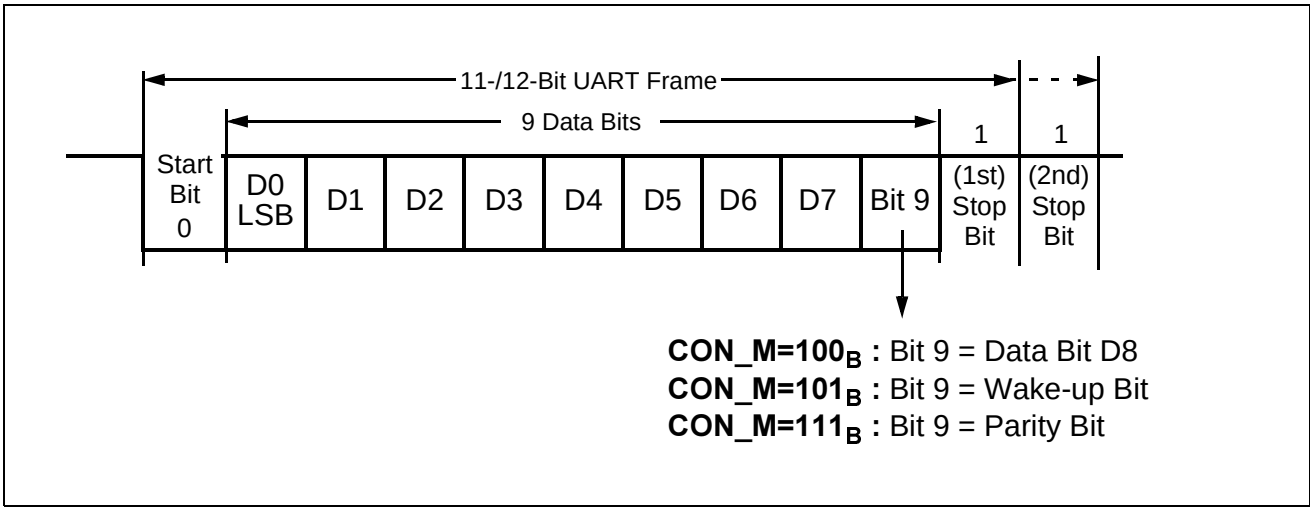


Figure 10-6 Asynchronous 9-Bit Frames

In wake-up mode, received frames are transferred to the receive buffer register only if the 9th bit (the wake-up bit) is 1. If this bit is 0, no receive interrupt request will be activated and no data will be transferred.

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 to identify the target slave. An address byte differs from a data byte in that the additional 9th bit is a 1 for an address byte, but is a 0 for a data byte; so, no slave will be interrupted by a data 'byte'. An address 'byte' will interrupt all slaves (operating in 8-bit data + wake-up bit mode), so each slave can examine the eight LSBs of the received character (the address). The addressed slave will switch to 9-bit data mode (such as by clearing bit CON_M.0), to enable 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 mode, ignoring the following data bytes.

10.3.1.2 Asynchronous Transmission

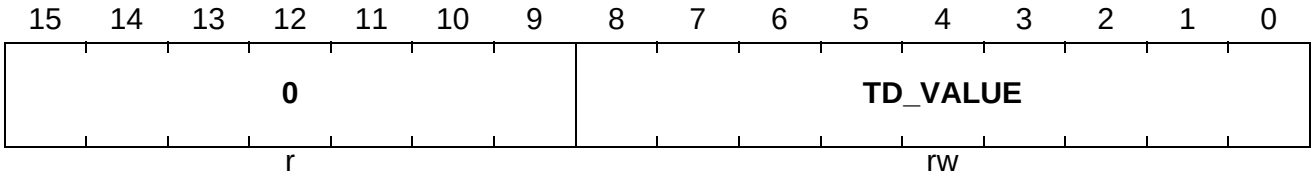
Asynchronous transmission begins at the next overflow of the divide-by-16 baudrate timer (transition of the baudrate clock f_{BR}), if bit CON_R is set and data has been loaded into TBUF. The transmitted data frame consists of three basic elements:

– Start bit
– Data field (eight or nine bits, LSB first, including a parity bit, if selected)
– Delimiter (one or two stop bits)

Data transmission is double-buffered. When the transmitter is idle, the transmit data loaded in the transmit buffer register TBUF is immediately moved to the transmit shift register, thus freeing the transmit buffer for the next data to be sent. This is indicated by the transmit Buffer interrupt request line TBIR being activated. TBUF may now be loaded with the next data, while transmission of the previous data continues.

The transmit interrupt request line TIR will be activated before the last bit of a frame is transmitted, that is, before the first or the second stop bit is shifted out of the transmit shift register.

TBUF
Transmit Buffer Register **(Reset value: 0000_H)**



Field	Bits	Typ	Description
TD_VALUE	[8:0]	rw	**Transmit Data Register Value** TBUF contains the data to be transmitted in either asynchronous or synchronous operating mode of the ASC. Data transmission is double buffered. Therefore, a new value can be written to TBUF before transmission of the previous value is completed.
0	[15:9]	r	**Reserved** for future use; reading returns 0; writing to these bit positions has no effect.

Note: The transmitter output pin TXD must be configured for alternate data output'.

10.3.1.3 Asynchronous Reception

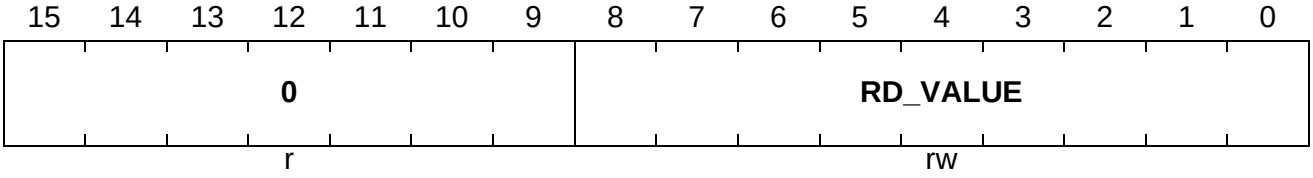
Asynchronous reception is initiated by a falling edge (1-to-0 transition) on line RXD, provided that bits CON_R and CON_REN are set. The receive data input line RXD is sampled at 16 times the rate of the selected baudrate. A majority decision of the 7th, 8th, and 9th sample determines the effective bit 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 line RXD. If the start bit proves valid, the receive circuit continues sampling and shifts the incoming data frame into the receive shift register.

When the last stop bit has been received, the content of the receive shift register are transferred to the receive data Buffer register RBUF. Simultaneously, the receive interrupt request line RIR is activated after the 9th sample in the last stop bit time slot (as programmed), regardless of whether valid stop bits have been received or not. The

receive circuit then waits for the next start bit (1-to-0 transition) at the receive data input line.

RBUF
Receive Buffer Register **Reset value: 0000_H**



Field	Bits	Typ	Description
RD_VALUE	[8:0]	rw	**Receive Data Register Value** RBUF contains the received data bits and, depending on the selected mode, the parity bit in asynchronous and synchronous operating mode of the ASC. In asynchronous operating mode with M=011 (7-bit data + parity) the received parity bit is written into RD7. In asynchronous operating mode with M=111 (8-bit data + parity) the received parity bit is written into RD8.
0	[15:9]	r	**Reserved** for future use; reading returns 0; writing to these bit positions has no effect.

Note: The receiver input pin RXD must be configured for input.

Asynchronous reception is stopped by clearing bit CON_REN. A currently received frame is completed including the generation of the receive interrupt request and an error interrupt request, if appropriate. Start bits that follow this frame will not be recognized.

Note: In wake-up mode, received frames are transferred to the receive FIFO register only if the 9th bit (the wake-up bit) is 1. If this bit is 0, no receive interrupt request will be activated and no data will be transferred.

10.3.2 Synchronous Operation

Synchronous Mode supports half-duplex communication, basically for simple I/O expansion via shift registers. Data is transmitted and received via line RXD while line TXD outputs the shift clock.

Note: These signals are alternate functions of port pins.

Synchronous Mode is selected with CON_M='000_B'.

Eight data bits are transmitted or received synchronous to a shift clock generated by the internal baudrate generator. The shift clock is active only as long as data bits are transmitted or received.

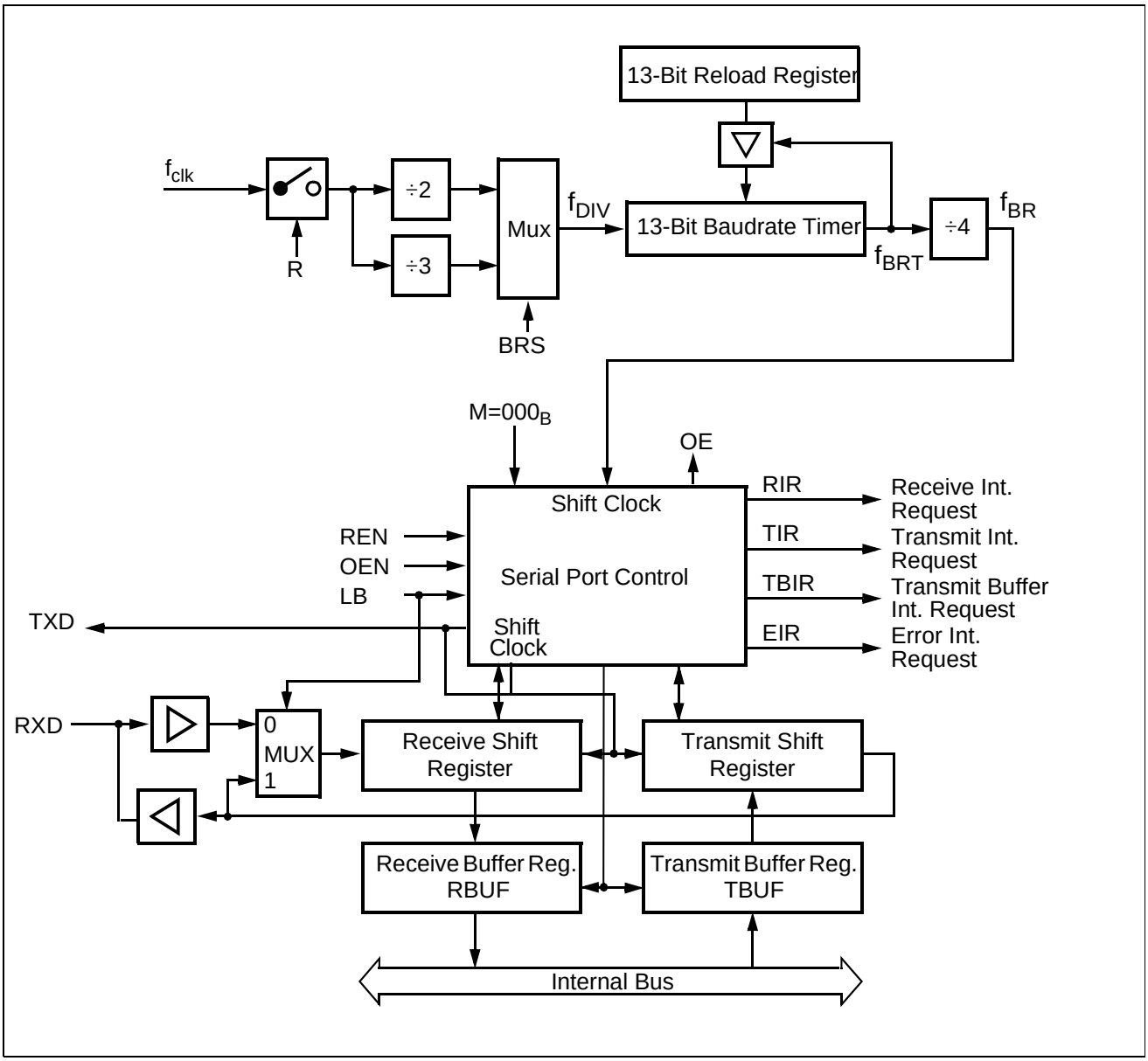


Figure 10-7 Synchronous Mode of Serial Channel ASC

10.3.2.1 Synchronous Transmission

Synchronous transmission begins within four state times after data has been loaded into TBUF, provided that CON_R is set and CON_REN is cleared (half-duplex, no reception). Exception: in loopback mode (bit CON_LB set), CON_REN must be set for reception of the transmitted byte. Data transmission is double-buffered. When the transmitter is idle, the transmit data loaded into TBUF is immediately moved to the transmit shift register, thus freeing TBUF for more data. This is indicated by the transmit Buffer interrupt request line TBIR being activated. TBUF may now be loaded with the next data, while transmission of the previous continoues. The data bits are transmitted synchronous with the shift clock. After the bit time for the eighth data bit, both the TXD and RXD lines will go high, the transmit interrupt request line TIR is activated, and serial data transmission stops.

Note: Pin TXD must be configured for alternate data output in order to provide the shift clock. Pin RXD must also be configured for output during transmission.

10.3.2.2 Synchronous Reception

Synchronous reception is initiated by setting bit CON_REN. If bit CON_R is set, the data applied at RXD is clocked into the receive shift register synchronous to the clock that is output at TXD. After the eighth bit has been shifted in, the contents of the receive shift register are transferred to the receive data buffer RBUF, the receive interrupt request line RIR is activated, the receiver enable bit CON_REN is reset, and serial data reception stops.

Note: Pin TXD must be configured for alternate data output in order to provide the shift clock. Pin RXD must be configured as alternate data input.

Synchronous reception is stopped by clearing bit CON_REN. A currently received byte is completed, including the generation of the receive interrupt request and an error interrupt request, if appropriate. Writing to the transmit buffer register while a reception is in progress has no effect on reception and will not start a transmission.

If a previously received byte has not been read out of a full receive buffer at the time the reception of the next byte is complete, both the error interrupt request line EIR and the overrun error status flag CON_OE will be activated/set, provided the overrun check has been enabled by bit CON_OEN.

10.3.2.3 Synchronous Timing

Figure 10-8 shows timing diagrams of the ASC Synchronous Mode data reception and data transmission. In idle state, the shift clock level is high. With the beginning of a synchronous transmission of a data byte, the data is shifted out at RXD with the falling edge of the shift clock. If a data byte is received through RXD, data is latched with the rising edge of the shift clock.

Between two consecutive receive or transmit data bytes, one shift clock cycle (f_{BR}) delay is inserted.

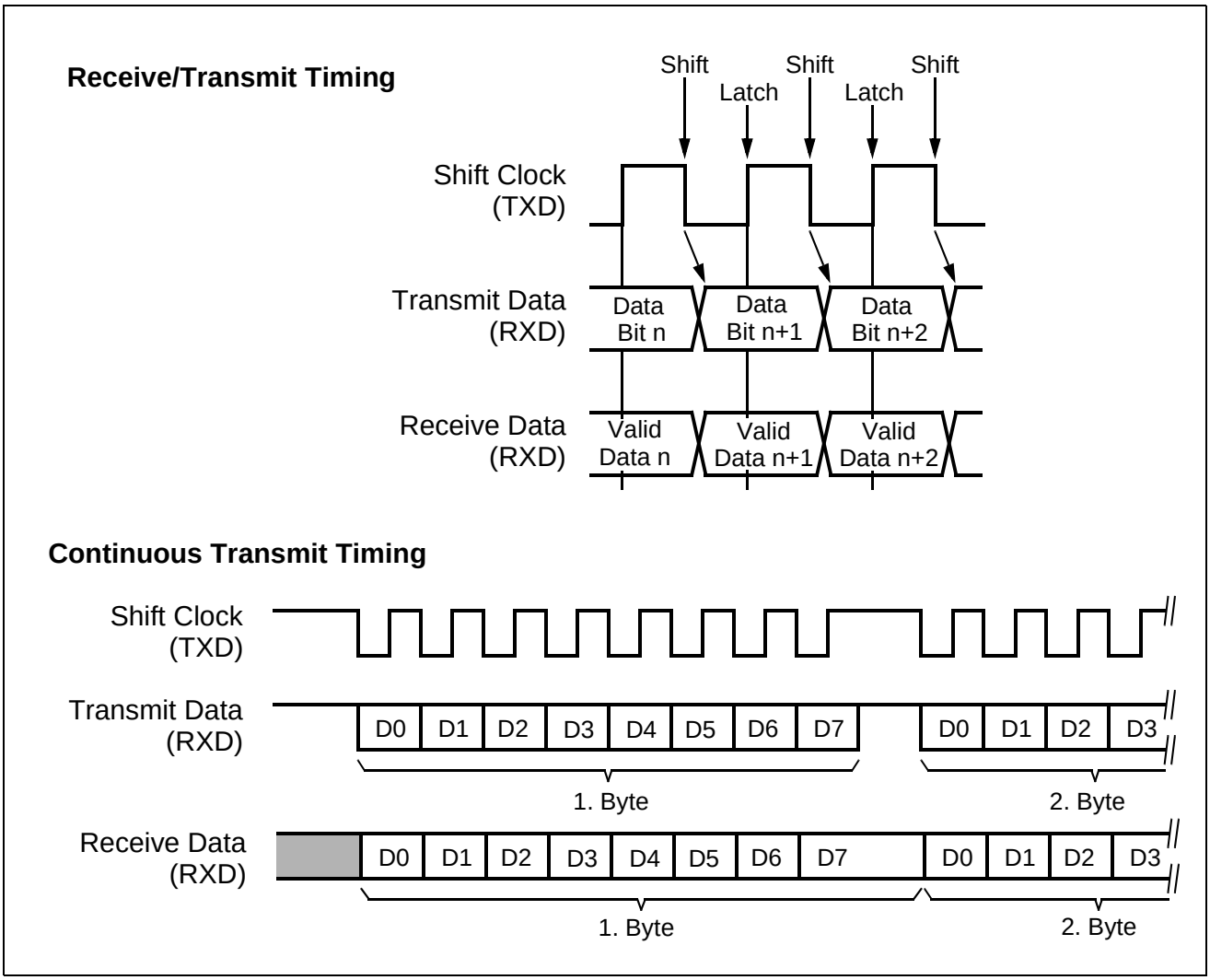


Figure 10-8 ASC Synchronous Mode Waveforms

10.3.3 Baudrate Generation

The serial channel ASC has its own dedicated 13-bit baudrate generator with reload capability, allowing baudrate generation independent of other timers.

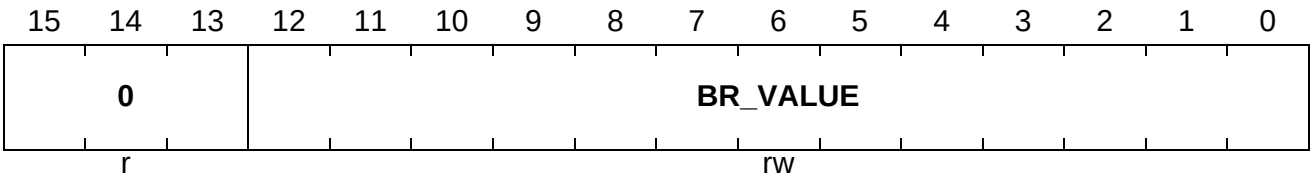
The baudrate generator is clocked with a clock (f_{DIV}) derived via a prescaler from the ASC input clock f_{clk}. The baudrate timer counts downwards and can be started or stopped through the baudrate generator run bit CON_R. Each underflow of the timer provides one clock pulse to the serial channel. The timer is reloaded with the value stored in its 13-bit reload register each time it underflows. The resulting clock f_{BRT} is again divided by a factor for the baudrate clock (±16 in asynchronous modes and ±4 in synchronous mode). The prescaler is selected by the bits CON_BRS and CON_FDE. In addition to the two fixed dividers, a fractional divider prescaler unit is available in the asynchronous operating modes that allows selection of prescaler divider ratios of n/512 with n=0...511. Therefore, the baudrate of ASC is determined by the module clock, the content of FDV, the reload value of BG, and the operating mode (asynchronous or synchronous).

Register BG is the dual-function Baudrate Generator/Reload register. Reading BG returns the contents of the timer BR_VALUE (bits 15...13 return zero), while writing to BG always updates the reload register (bits 15...13 are insignificant).

An auto-reload of the timer with the contents of the reload register is performed each time CON_BG is written to. However, if CON_R is cleared at the time a write operation to CON_BG is performed, the timer will not be reloaded until the first instruction cycle after CON_R was set. For a clean baudrate initialization, CON_BG should be written only if CON_R is reset. If CON_BG is written while CON_R is set, unpredictable behavior of the ASC may occur during running transmit or receive operations.

The ASC baudrate timer reload register BG contains the 13-bit reload value for the baudrate timer in Asynchronous and Synchronous modes.

BG
Baudrate Timer/Reload Register **Reset value: 0000_H**



10.3.3.1 Baudrate in Asynchronous Mode

For asynchronous operation, the baudrate generator provides a clock f_{BRT} with sixteen times the rate of the established baudrate. Every received bit is sampled at the 7th, 8th, and 9th cycle of this clock. The clock divider circuitry, which generates the input clock for the 13-bit baudrate timer, is extended by a fractional divider circuitry that allows adjustment for more accurate baudrate and the extension of the baudrate range.

Field	Bits	Typ	Description
BR_VALUE	[12:0]	rw	**Baudrate Timer/Reload Value** Reading returns the 13-bit contents of the baudrate timer; writing loads the baudrate timer/reload value. *Note: BG should only be written if R='0'.*
0	[15:13]	r	**Reserved** for future use; reading returns 0; writing to these bit positions has no effect.

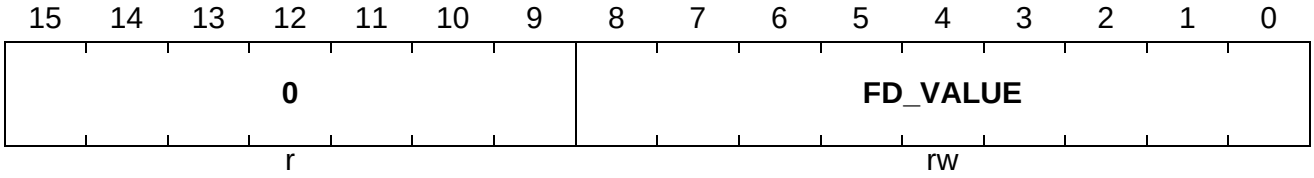
The baudrate of the baudrate generator depends on the following bits and register values:

– Input clock f_{clk}
– Selection of the baudrate timer input clock f_{DIV} by bits CON_FDE and CON_BRS
– If bit CON_FDE=1 (fractional divider): value of register CON_FDV
– Value of the 13-bit reload register BG

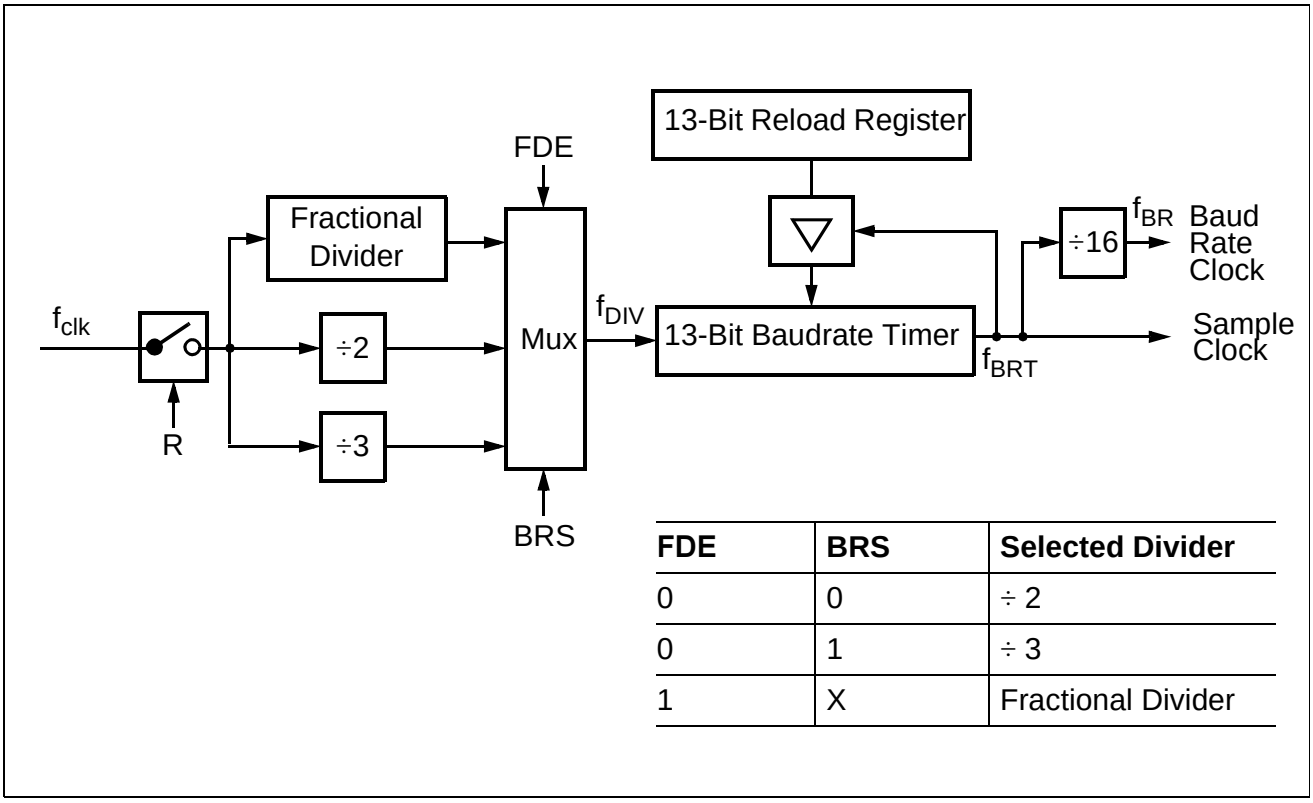
The output clock of the baudrate timer with the reload register is the sample clock in the asynchronous modes of the ASC. For baudrate calculations, this baudrate clock f_{BR} is derived from the sample clock f_{DIV} by a division by 16.

The ASC fractional divider register FDV contains the 9-bit divider value for the fractional divider (asynchronous mode only). It is also used for reference clock generation of the autobaud detection unit.

FDV
Fractional Divider Register **Reset value: 0000_H**



Field	Bits	Typ	Description
FD_VALUE	[8:0]	rw	**Fractional divider register value** FD_VALUE contains the 9-bit value of the fractional divider which defines the fractional divider ratio n/512 (n=0...511). With n=0, the fractional divider is switched off (input=output frequency, $f_{DIV} = f_{clk}$, see **Figure 10-9**).
0	[15:9]	r	**Reserved** for future use; reading returns 0; writing to these bit positions has no effect.



FDE	BRS	Selected Divider
0	0	÷ 2
0	1	÷ 3
1	X	Fractional Divider

Figure 10-9 ASC Baudrate Generator Circuitry in Asynchronous Modes

Using the fixed Input Clock Divider

The baudrate for asynchronous operation of serial channel ASC when using the fixed input clock divider ratios (CON_FDE=0) and the required reload value for a given baudrate can be determined by the following formulas:

Table 10-1 Asynchronous Baudrate Formulas using the Fixed Input Clock Dividers

FDE	BRS	BG	Formula
0	0	0 ... 8191	$\text{Baudrate} = \frac{f_{clk}}{32 \times (BG+1)}$ $BG = \frac{f_{clk}}{32 \times \text{Baudrate}} - 1$
	1		$\text{Baudrate} = \frac{f_{clk}}{48 \times (BG+1)}$ $BG = \frac{f_{kw\_clk}}{48 \times \text{Baudrate}} - 1$

BG represents the contents of the reload register BG (BR_VALUE), taken as unsigned 13-bit integer.

The maximum baudrate that can be achieved for the asynchronous modes when using the two fixed clock dividers and a module clock of 60 MHz is 1.875 MBaud. **Table 10-2** lists various commonly used baudrates together with the required reload values and the deviation errors compared to the intended baudrate.

Table 10-2 Typical Asynchronous Baudrates using the Fixed Input Clock Dividers

Baudrate	BRS = '0', f_{clk} = 60 MHz		BRS = '1', f_{clk} = 60 MHz	
	Deviation Error	Reload Value	Deviation Error	Reload Value
1.875 MBaud	---	0000_H	---	---
1.25 MBaud	---	---	---	0000_H
19.2 KBaud	+0.7 % / -0.4 %	0060_H / 0061_H	+0.2 % / -1.4 %	0040_H / 0041_H
9600 Baud	+0.2 % / -0.4 %	$00C2_H$ / $00C3_H$	+0.2 % / -0.6 %	0081_H / 0082_H
4800 Baud	+0.2 % / -0.1 %	0185_H / 0186_H	+0.2 % / -0.2 %	0104_H / 0105_H
2400 Baud	+0.0 % / -0.1 %	$030C_H$ / $030D_H$	+0.2 % / -0.0 %	0207_H / 0208_H
1200 Baud	+0.0 % / -0.0 %	0619_H / $061A_H$	+0.1 % / -0.0 %	0410_H / 0411_H

Note: CON_FDE must be 0 to achieve the baudrates in the table above. The deviation errors given in the table above are rounded. Using a baudrate crystal will provide correct baudrates without deviation errors.

Using the Fractional Divider

When the fractional divider is selected, the input clock f_{DIV} for the baudrate timer is derived from the module clock f_{clk} by a programmable divider. If CON_FDE is set, the fractional divider is activated. It divides f_{clk} by a fraction of n/512 for any value of n from 0 to 511. If n=0, the divider ratio is 1, which means that $f_{DIV}=f_{clk}$. In general, the fractional divider allows the baudrate to be programmed with much more accuracy than with the two fixed prescaler divider stages.

BG represents the contents of the reload register BG (BR_VALUE), taken as an unsigned 13-bit integer.
FDV represents the contents of the fractional divider register (FD_VALUE) taken as an unsigned 9-bit integer.

Table 10-3 Asynchronous Baudrate Formulas using the Fractional Input Clock Divider

FDE	BRS	BG	FDV	Formula
1	-	1 ... 8191	1 ... 511	$\text{Baudrate} = \frac{\text{FDV}}{512} \times \frac{f_{clk}}{16 \times (\text{BG}+1)}$
			0	$\text{Baudrate} = \frac{f_{clk}}{16 \times (\text{BG}+1)}$

Table 10-4 Typical Asynchronous Baudrates using the Fractional Input Clock Divider

f_{clk}	Desired Baudrate	BG	FDV	Resulting Baudrate	Deviation
40 MHz	115.2 kBaud	15_H	$1F7_H$	115.214 kBaud	0.01 %
	57.6 kBaud	26_H	$1F7_H$	57.607 kBaud	0.01 %
	38.4 kBaud	41_H	$1F7_H$	38.404 kBaud	0.01 %
	19.2 kBaud	83_H	$1F7_H$	19.202 kBaud	0.01 %
60 MHz	115.2 kBaud	20_H	$1F7_H$	115.214 kBaud	0.01 %
	57.6 kBaud	41_H	$1F7_H$	57.607 kBaud	0.01 %
	38.4 kBaud	61_H	$1FD_H$	38.415 kBaud	0.04 %
	19.2 kBaud	$C5_H$	$1FD_H$	19.207 kBaud	0.04 %

*Note: **ApNote AP2423** provides a program 'ASC.EXE' which allows calculation of values for the FDV and BG registers depending on f_{clk}, the requested baudrate, and the maximum deviation.*

10.3.3.2 Baudrate in Synchronous Mode

For synchronous operation, the baudrate generator provides a clock with four times the rate of the established baudrate.(see **Figure 10-10**).

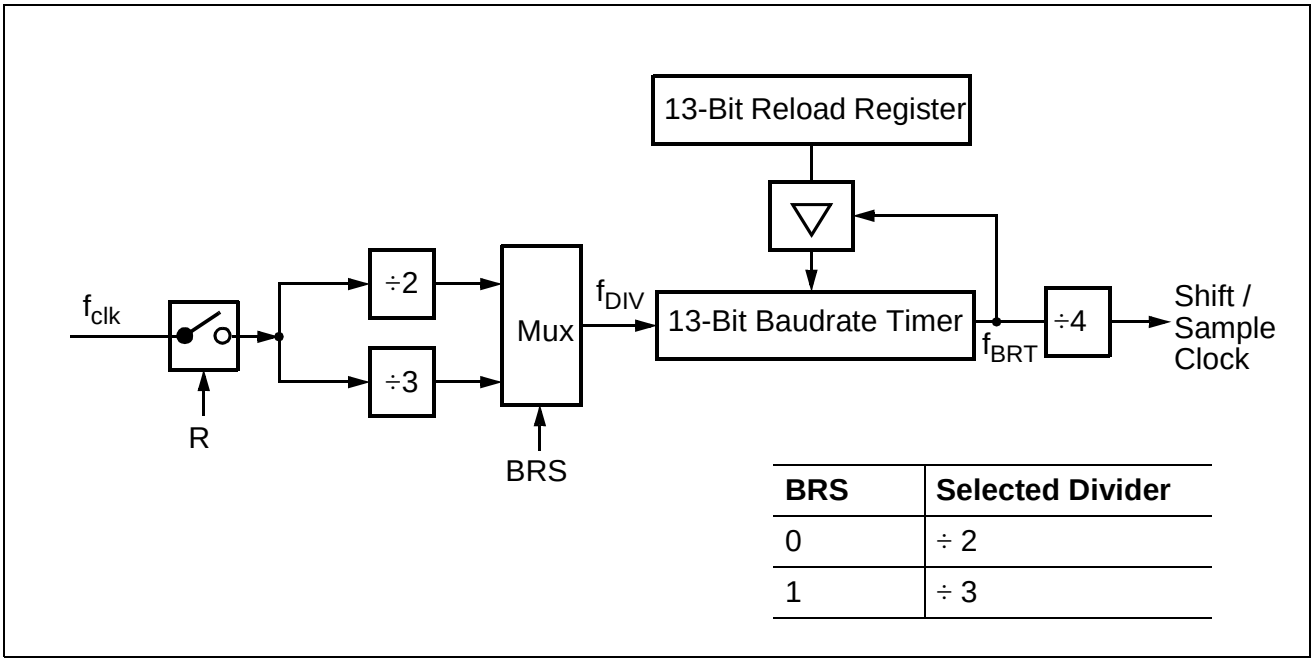


Figure 10-10 ASC Baudrate Generator Circuitry in Synchronous Mode

The baudrate for synchronous operation of serial channel ASC can be determined by the formulas as shown in **Table 10-5**.

Table 10-5 Synchronous Baudrate Formulas

SBRS	BG	Formula
0	0 ... 8191	$\text{Baudrate} = \frac{f_{clk}}{8 \times (BG+1)}$ $\quad BG = \frac{f_{clk}}{8 \times \text{Baudrate}} - 1$
1		$\text{Baudrate} = \frac{f_{clk}}{12 \times (BG+1)}$ $\quad BG = \frac{f_{clk}}{12 \times \text{Baudrate}} - 1$

BG represents the contents of the reload register (BR_VALUE), taken as an unsigned 13-bit integers.

The maximum baudrate that can be achieved in synchronous mode when using a module clock of 60 MHz is 7.5 MBaud.

10.3.4 Hardware Error Detection Capabilities

To improve the safety of serial data exchange, the serial channel ASC provides an error interrupt request flag to indicate the presence of an error, and three (selectable) error status flags in register CON to indicate which error has been detected during reception. Upon completion of a reception, the error interrupt request line EIR will be activated simultaneously with the receive interrupt request line RIR, if one or more of the following conditions are met:

- If the framing error detection enable bit CON_FEN is set and any of the expected stop bits is not high, the framing error flag CON_FE is set, indicating that the error interrupt request is due to a framing error (Asynchronous Mode only).
- If the parity error detection enable bit CON_PEN is set in the modes where a parity bit is received, and the parity check on the received data bits proves false, the parity error flag CON_PE is set, indicating that the error interrupt request is due to a parity error (Asynchronous Mode only).
- If the overrun error detection enable bit CON_OEN is set and the last character received was not read out of the receive buffer by software or by a DMA transfer at the time the reception of a new frame is complete, the overrun error flag CON_OE is set indicating that the error interrupt request is due to an overrun error (Asynchronous and Synchronous Mode).

10.3.5 Interrupts

Four interrupt sources are provided for serial channel ASC. Line TIC indicates a transmit interrupt, TBIC indicates a transmit FIFO interrupt, RIC indicates a receive interrupt, and SEIC indicates an error interrupt of the serial channel. The interrupt output lines TBIR, TIR, RIR, and EIR are activated (active state) for two periods of the module clock f_{clk}.

The cause of an error interrupt request (framing, parity, overrun error) can be identified by the error status flags FE, PE, and OE located in control register CON.

Note: In contrast to the error interrupt request line EIR, the error status flags FE/PE/OE are not reset automatically but must be cleared by software.

For normal operation (other than error interrupt) the ASC provides three interrupt requests to control data exchange via this serial channel:

- TBIR is activated when data is moved from TBUF to the transmit shift register.
- TIR is activated before the last bit of an asynchronous frame is transmitted, or after the last bit of a synchronous frame has been transmitted.
- RIR is activated when the received frame is moved to RBUF.

While the task of the receive interrupt handler is quite clear, the transmitter is serviced by two interrupt handlers. This provides advantages for the servicing software.

For single transfers, it is sufficient to use the transmitter interrupt (TIR), which indicates that all the previously loaded data has been transmitted except for the last bit of an asynchronous frame.

For multiple back-to-back transfers, it is necessary to load the following piece of data until the time the last bit of the previous frame has been transmitted. In Asynchronous Mode, this leaves just one bit-time for the handler to respond to the transmitter interrupt request; in Synchronous Mode it is completly impossible at all if no FIFO is present.

Using the transmit buffer interrupt (TBIR) to reload transmit data provides the time to transmit a complete frame for the service routine, as TBUF may be reloaded while the previous data is still being transmitted.

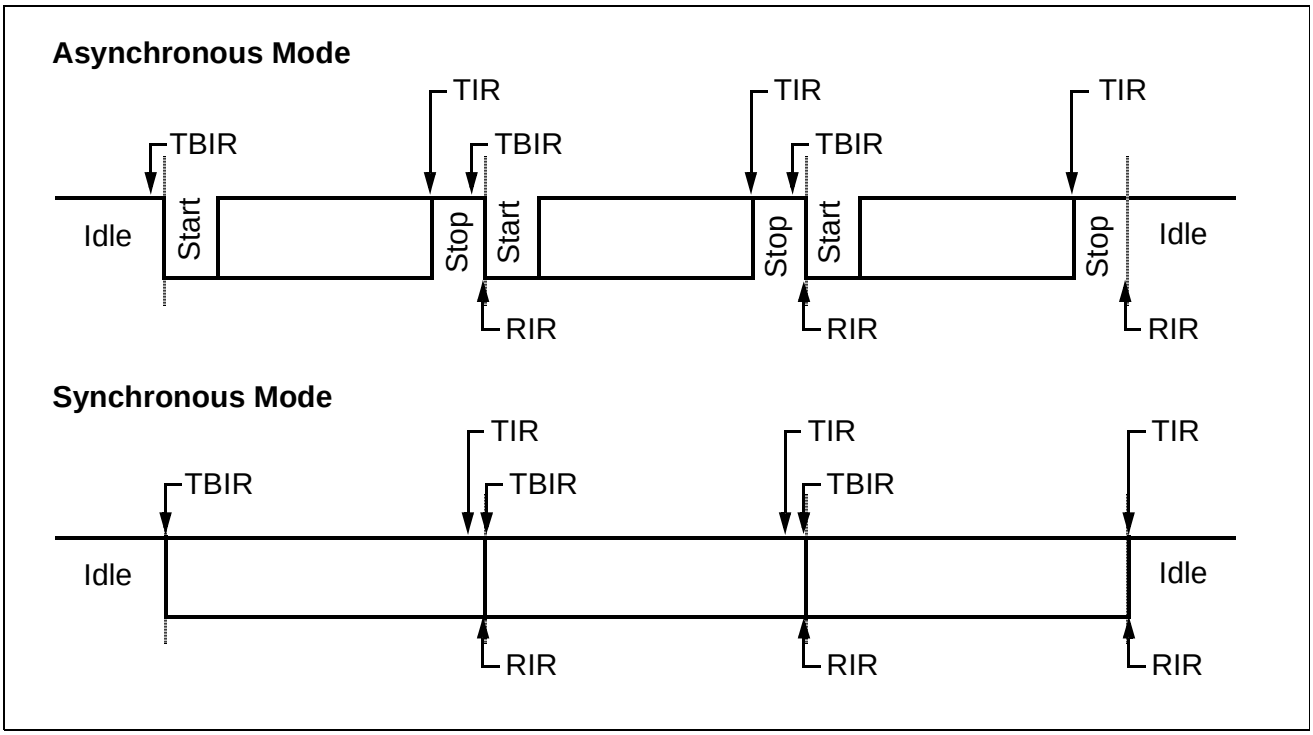


Figure 10-11 ASC Interrupt Generation

As shown in **Figure 10-11**, TBIR is an early trigger for the reload routine, while TIR indicates the completed transmission. Therefore, software using handshake should rely on TIR at the end of a data block to ensure that all data has actually been transmitted.

Note: Refer to the general Interrupt Control Register description for an explanation of the control fields

11 High-Speed Synchronous Serial Interface (SSC)

11.1 Introduction

The High-Speed Synchronous Serial Interface (SSC) supports both full-duplex and half-duplex serial synchronous communication up to 30 MBaud (@ 60 MHz module clock). The serial clock signal can be generated by the SSC itself (Master Mode) or can be received from an external master (Slave Mode). Data width, shift direction, clock polarity, and phase are programmable. This allows communication with SPI-compatible devices. Transmission and reception of data is double-buffered. A 16-bit baudrate generator provides the SSC with a separate serial clock signal.

Features:

- Master and Slave Mode operation
  - Full-duplex or half-duplex operation

- Flexible data format
  - Programmable number of data bits: 2 to 16 bits
  - Programmable shift direction: LSB or MSB shift first
  - Programmable clock polarity: idle low or high state for the shift clock
  - Programmable clock/data phase: data shift with leading or trailing edge of the shift clock

- Baudrate generation from 30 MBaud to 457.76 Baud (@ 60 MHz module clock)

- Interrupt generation
  - On a transmitter empty condition
  - On a receiver full condition
  - On an error condition (receive, phase, baudrate, transmit error)

Figure 11-1 shows all funtional relevant interfaces associated with the SSC Kernel.

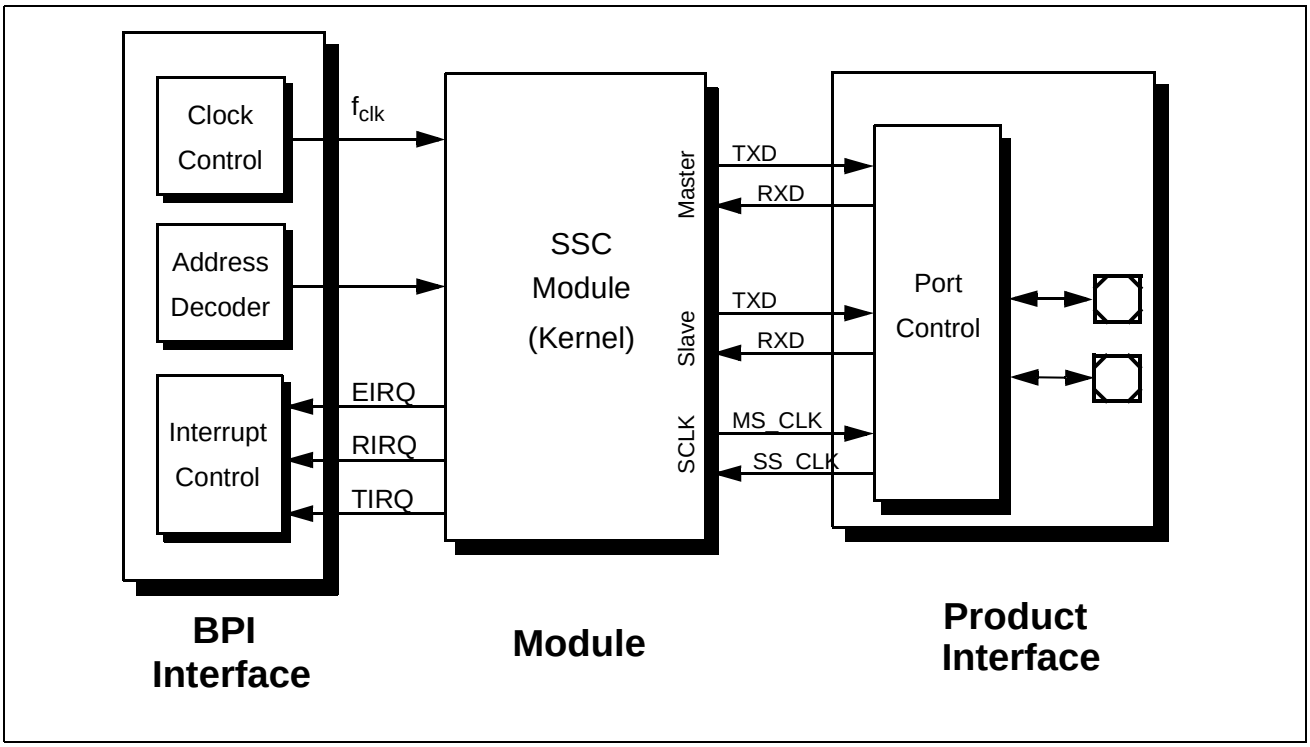


Figure 11-1 SSC Interface Diagram

Figure 11-2 shows all of the registers associated with the SSC Kernel.

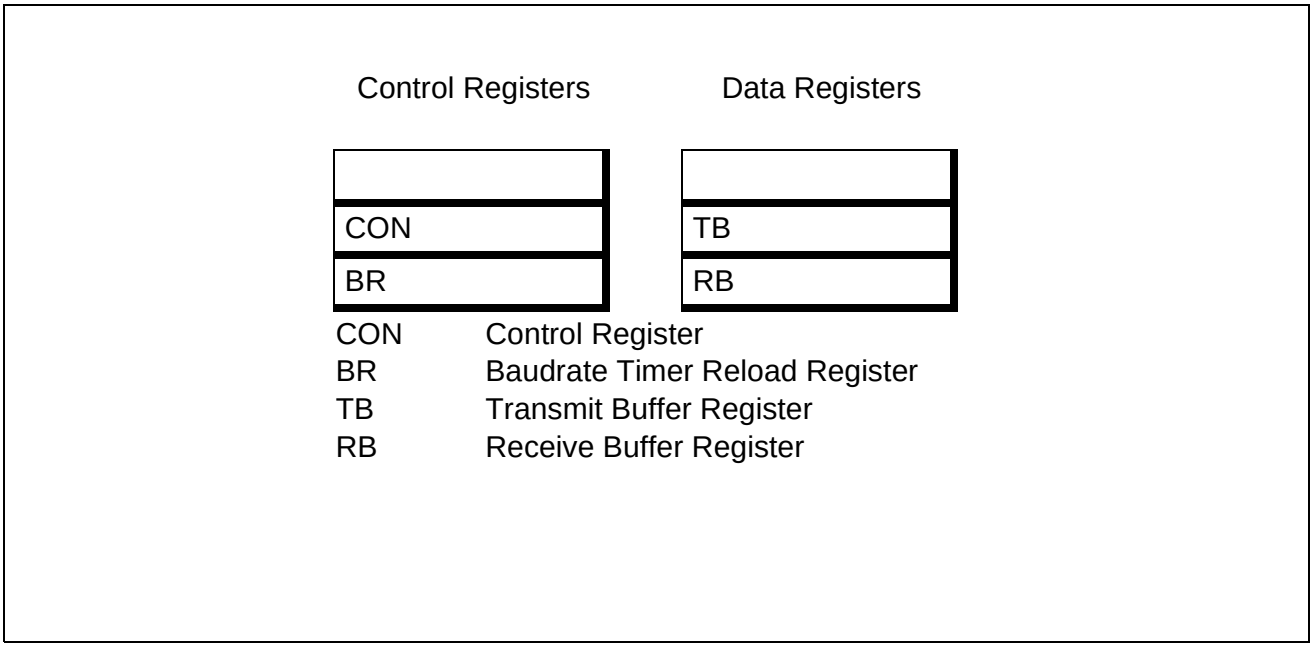


Figure 11-2 SSC Kernel Registers

All SSC registers are located in the SFR/ESFR memory space. The respective SFR addresses can be found in list of SFRs.

11.2 General Operation

The SSC supports full-duplex and half-duplex synchronous communication up to 30 MBaud (@ 60 MHz module clock). The serial clock signal can be generated by the SSC itself (Master Mode) or can be received from an external master (Slave Mode). Data width, shift direction, clock polarity, and phase are programmable. This allows communication with SPI-compatible devices. Transmission and reception of data is double-buffered. A 16-bit baudrate generator provides the SSC with a separate serial clock signal.

The high-speed synchronous serial interface can be configured in a very flexible way, so it can be used with other synchronous serial interfaces, can serve for master/slave or multimaster interconnections or can operate compatible with the popular SPI interface. Thus, the SSC can be used to communicate with shift registers (IO expansion), peripherals (e.g. EEPROMs etc.) or other controllers (networking). The SSC supports half-duplex and full-duplex communication. Data is transmitted or received on lines TXD and RXD, normally connected with pins MTSR (Master Transmit / Slave Receive) and MRST (Master Receive / Slave Transmit). The clock signal is output via line MS_CLK (Master Serial Shift Clock) or input via line SS_CLK (Slave Serial Shift Clock). Both lines are normally connoted to pin SCLK. These pins are alternate functions of port pins.

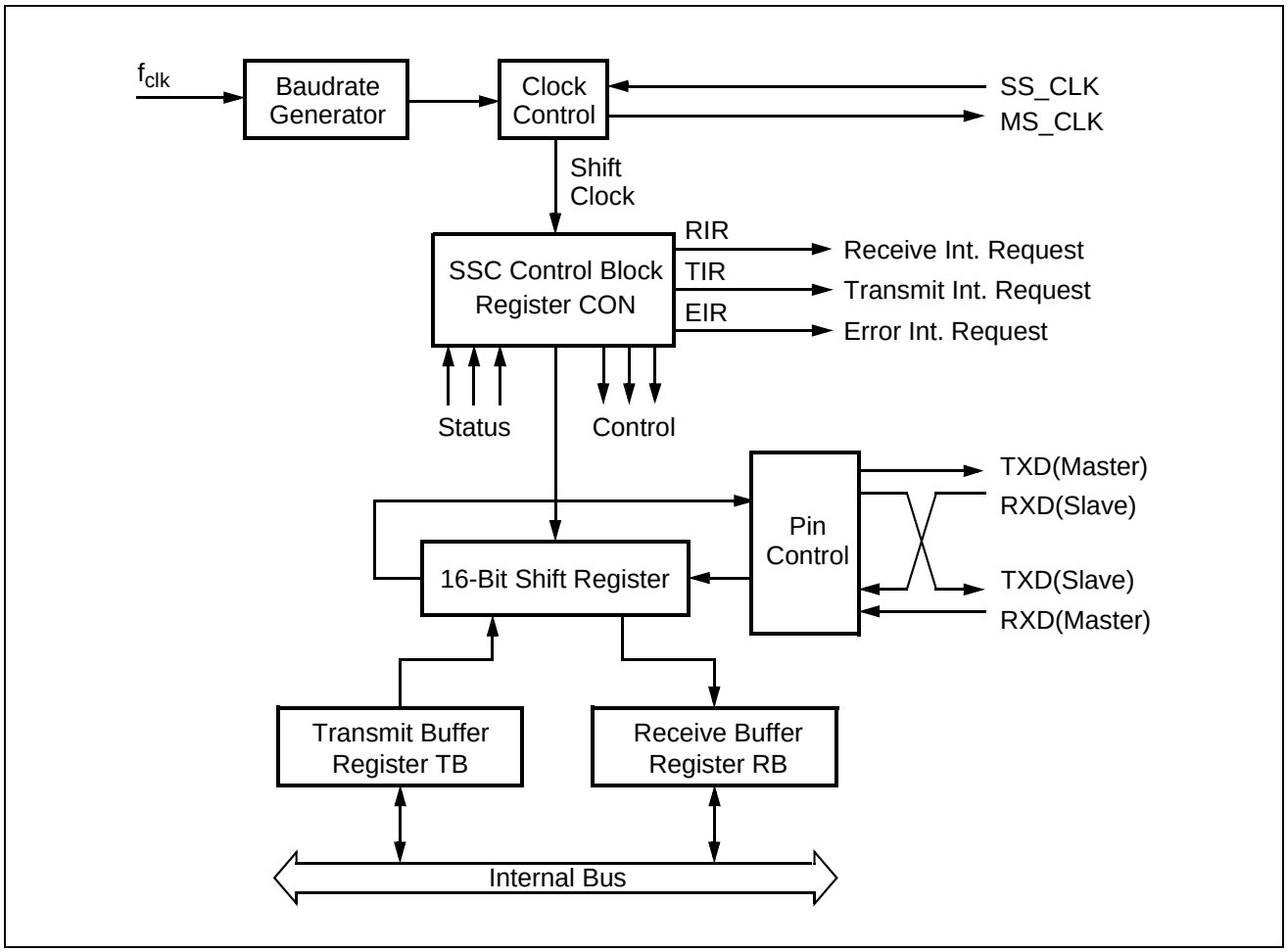


Figure 11-3 Synchronous Serial Channel SSC Block Diagram

11.2.1 Operating Mode Selection

The operating mode of the serial channel SSC is controlled by its control register CON. This register serves two purposes:

- During programming (SSC disabled by CON.EN=0), it provides access to a set of control bits
- During operation (SSC enabled by CON.EN=1), it provides access to a set of status flags.

Configuration Register

This register contains control bits for mode and error check selection, and status flags for error identification. Depending on bit EN, either control functions or status flags and master/slave control is enabled.

CON.EN = 0: Programming Mode

CON
Control Register **(Reset value: 0000_H)**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN	**MS**	**0**	**AREN**	**BEN**	**PEN**	**REN**	**TEN**	**LB**	**PO**	**PH**	**HB**	**BM**			
rw	rw	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw			

Field	Bits	Type	Description
BM	[3:0]	rw	**Data Width Selection** 0000 Reserved. Do not use this combination. 0001 - 1111 Transfer Data Width is 2...16 bit (<BM>+1)
HB	4	rw	**Heading Control** 0 Transmit/Receive LSB First 1 Transmit/Receive MSB First
PH	5	rw	**Clock Phase Control** 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
PO	6	rw	**Clock Polarity Control** 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

Field	Bits	Type	Description
LB	7	rw	**Loop Back Control** 0 Normal output 1 Receive input is connected with transmit output (half-duplex mode)
TEN	8	rw	**Transmit Error Enable** 0 Ignore transmit errors 1 Check transmit errors
REN	9	rw	**Receive Error Enable** 0 Ignore receive errors 1 Check receive errors
PEN	10	rw	**Phase Error Enable** 0 Ignore phase errors 1 Check phase errors
BEN	11	rw	**Baudrate Error Enable** 0 Ignore baudrate errors 1 Check baudrate errors
AREN	12	rw	**Automatic Reset Enable** 0 No additional action upon a baudrate error 1 The SSC is automatically reset upon a baudrate error
MS	14	rw	**Master Select** 0 Slave Mode. Operate on shift clock received via SCLK. 1 Master Mode. Generate shift clock and output it via SCLK.
EN	15	rw	**Enable Bit = '0'** Transmission and reception disabled. Access to control bits.
0	13	r	**Reserved**; returns '0' if read; should be written with '0';

CON.EN = 1: Operating Mode

CON
Control Register **(Reset value: 0000_H)**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN	**MS**	**0**	**BSY**	**BE**	**PE**	**RE**	**TE**	**–**	**–**	**–**	**–**	**BC**			
rw	rw	r	rh	rwh	rwh	rwh	rwh			r				rw	

Field	Bits	Type	Description
BC	[3:0]	rh	**Bit Count Field** 0001 - 1111 Shift counter is updated with every shifted bit. **Do not write to !!!**
TE	8	rwh	**Transmit Error Flag** 0 No error 1 Transfer starts with the slave's transmit buffer not being updated
RE	9	rwh	**Receive Error Flag** 0 No error 1 Reception completed before the receive buffer was read
PE	19	rwh	**Phase Error Flag** 0 No error 1 Received data changes around sampling clock edge
BE	11	rwh	**Baudrate Error Flag** 0 No error 1 More than factor 2 or 0.5 between slave's actual and expected baudrate
BSY	12	rh	**Busy Flag** Set while a transfer is in progress. **Do not write to!!!**
MS	14	rw	**Master Select Bit** 0 Slave Mode. Operate on shift clock received via SCLK. 1 Master Mode. Generate shift clock and output it via SCLK.
EN	15	rw	**Enable Bit = '1'** Transmission and reception enabled. Access to status flags and M/S control
–	[7:4]	-	**Reserved**:
0	13	r	**Reserved**; returns '0' if read; should be written with '0';

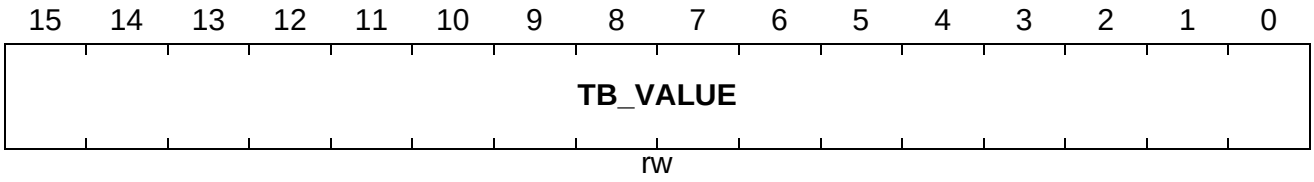
Note: The target of an access to CON (control bits or flags) is determined by the state of CON.EN prior to the access; that is, writing $C057_H$ to CON in programming mode (CON.EN=0) will initialize the SSC (CON.EN was 0) and then turn it on (CON.EN=1). When writing to CON, ensure that reserved locations receive zeros.

The shift register of the SSC is connected to both the transmit lines and the receive lines via the pin control logic (see block diagram in **Figure 11-3**). Transmission and reception of serial data are synchronized and take place at the same time, i.e. the same number of transmitted bits is also received. Transmit data is written into the Transmit Buffer (TB). It is moved to the shift register as soon as this is empty. An SSC master (CON.MS=1) immediately begins transmitting, while an SSC slave (CON.MS=0) will wait for an active shift clock. When the transfer starts, the busy flag CON.BSY is set and the Transmit Interrupt Request line TIR will be activated to indicate that register TB may be reloaded again. When the programmed number of bits (2...16) has been transferred, the contents of the shift register are moved to the Receive Buffer RB and the Receive Interrupt Request line RIR will be activated. If no further transfer is to take place (TB is empty), CON.BSY will be cleared at the same time. Software should not modify CON.BSY, as this flag is hardware controlled.

Transmitter Buffer Register

The SSC transmitter buffer register TB contains the transmit data value.

TB
Transmitter Buffer Register **(Reset value: 0000_H)**

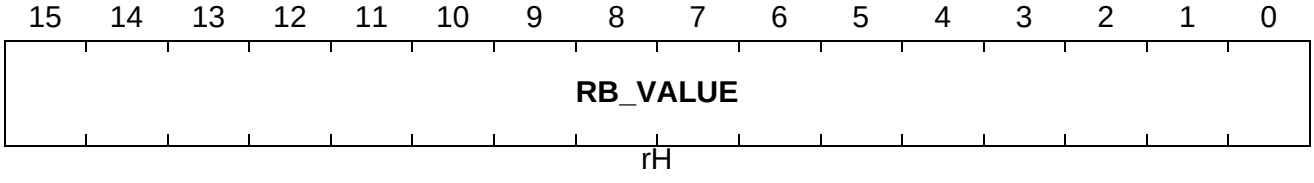


Field	Bits	Type	Description
TB_VALUE	[15:0]	rw	**Transmit Data Register Value** TB_VALUE is the data value to be transmitted. Unselected bits of TB are ignored during transmission.

Receiver Buffer Register

The SSC receiver buffer register RB contains the receive data value.

RB
Receiver Buffer Register **(Reset value: 0000_H)**



Field	Bits	Type	Description
RB_VALUE	[15:0]	rh	**Receive Data Register Value** RB contains the received data value RB_VALUE. Unselected bits of RB will be not valid and should be ignored.

Note: Only one SSC (etc.) can be master at a given time.

The transfer of serial data bits can be programmed in many respects:

- The data width can be specified from 2 bits to 16 bits
- A transfer may start with either the LSB or the MSB
- The shift clock may be idle low or idle high
- The data bits may be shifted with the leading edge or the trailing edge of the shift clock signal
- The baudrate may be set from 457.76 Baud up to 30 MBaud (@ 60 MHz module clock)
- The shift clock can be generated (MS_CLK) or can be received (SS_CLK)

These features allow the adaptation of the SSC to a wide range of applications in which serial data transfer is required.

The Data Width Selection supports the transfer of frames of any data length, from 2-bit "characters" up to 16-bit "characters". Starting with the LSB (CON.HB=0) allows communication with SSC devices in Synchronous Mode or with 8051 like serial interfaces for example. Starting with the MSB (CON.HB=1) allows operation compatible with the SPI interface.
Regardless of the data width selected and whether the MSB or the LSB is transmitted first, the transfer data is always right-aligned in registers TB and RB, with the LSB of the transfer data in bit 0 of these registers. The data bits are rearranged for transfer by the internal shift register logic. The unselected bits of TB are ignored; the unselected bits of RB will not be valid and should be ignored by the receiver service routine.

The Clock Control allows the adaptation of transmit and receive behavior of the SSC to a variety of serial interfaces. A specific shift clock edge (rising or falling) is used to shift out transmit data, while the other shift clock edge is used to latch in receive data. Bit CON.PH selects the leading edge or the trailing edge for each function. Bit CON.PO selects the level of the shift clock line in the idle state. Thus, for an idle-high clock, the leading edge is a falling one, a 1-to-0 transition (see **Figure 11-4**).

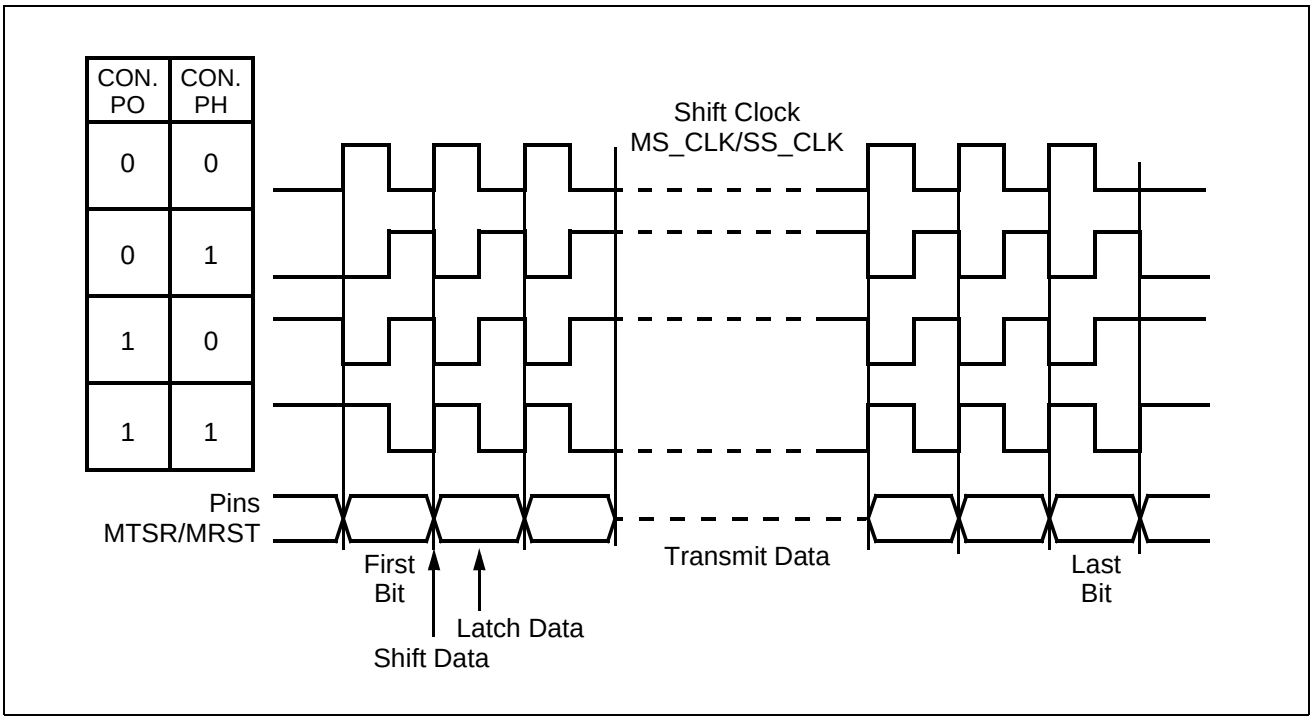


Figure 11-4 Serial Clock Phase and Polarity Options

11.2.2 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 line TXD is the transmit line; the receive line is connected to its data input line RXD; the shift clock line is either MS_CLK or SS_CLK. Only the device selected for master operation generates and outputs the shift clock on line MS_CLK. All slaves receive this clock; so, their pin SCLK must be switched to input mode. The output of the master's shift register is connected to the external transmit line, which in turn is connected to the slaves' shift register input. The output of the slaves' shift register is connected to the external receive line in order to enable the master to receive the data shifted out of the slave. The external connections are hard-wired, the function and direction of these pins is determined by the master or slave operation of the individual device.

Note: The shift direction shown in the figure applies for MSB-first operation as well as for LSB-first operation.

When initializing the devices in this configuration, one device must be selected for master operation while all other devices must be programmed for slave operation. Initialization includes the operating mode of the device's SSC and also the function of the respective port lines.

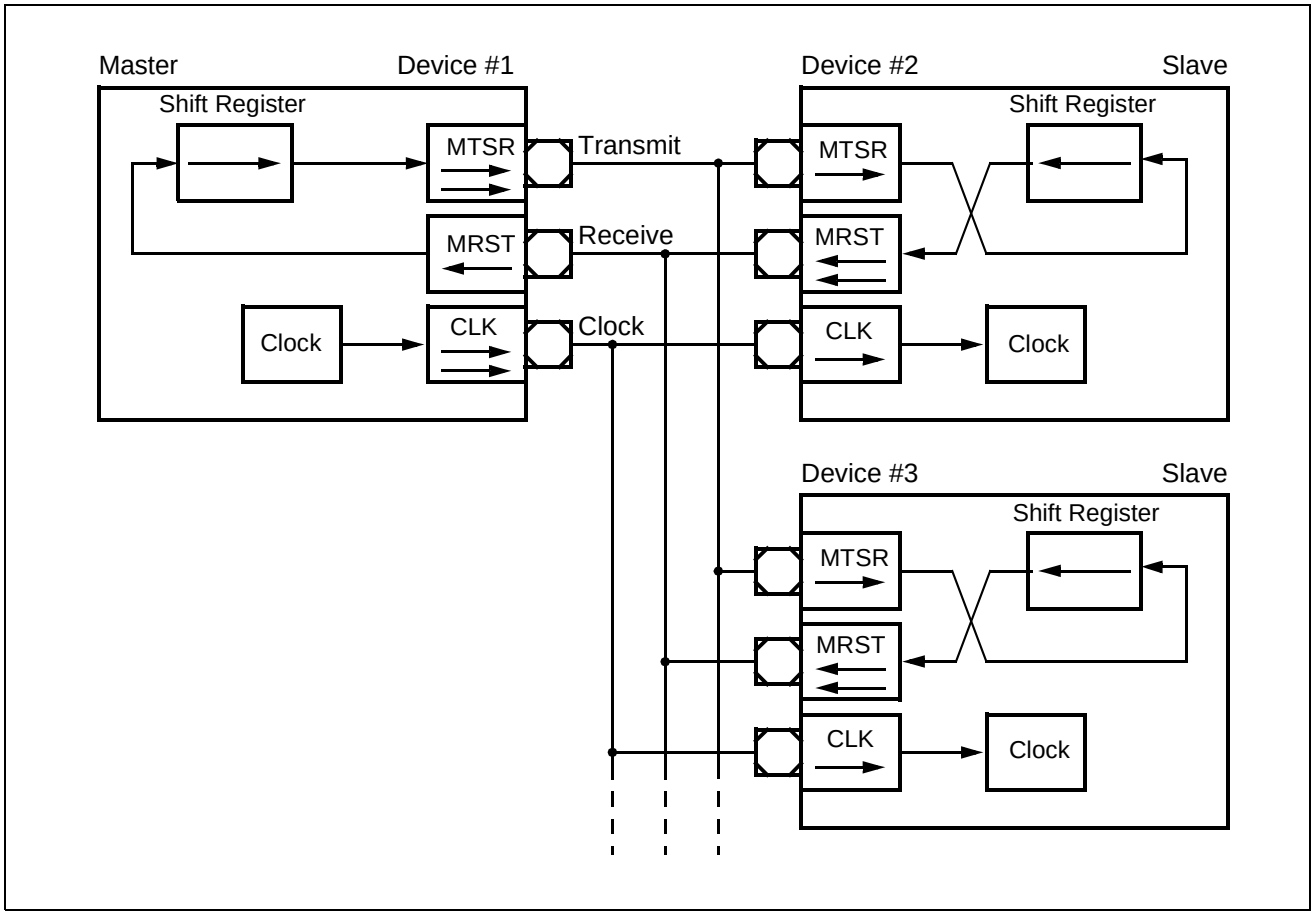


Figure 11-5 SSC Full-Duplex Configuration

The data output pins MRST of all slave devices are connected together onto the one receive line in the configuration shown in **Figure 11-5**. During a transfer each slave shifts out data from its shift register. There are two ways to avoid collisions on the receive line due to different slave data:

– Only one slave drives the line, i.e. enables the driver of its MRST pin. All the other slaves must have their MRST pins programmed as input so only one slave can put its data onto the master's receive line. Only receiving data from the master is possible. The master selects the slave device from which it expects data either by separate select lines, or by sending a special command to this slave. The selected slave then switches its MRST line to output until it gets a de-selection signal or command.

– The slaves use open drain output on MRST. This forms a wired-AND connection. The receive line needs an external pull-up in this case. Corruption of the data on the receive line sent by the selected slave is avoided when all slaves not selected for transmission to the master only send ones (1s). Because 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 the necessary initialization 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 will start. After a transfer, the alternate data line will always remain at the logic level of the last transmitted data bit.

When the serial interfaces are enabled, the master device can initiate the first data transfer by writing the transmit data into register TB. This value is copied into the shift register (assumed to be empty at this time), and the selected first bit of the transmit data will be placed onto the TXD line on the next clock from the baudrate generator (transmission starts only if CON.EN=1). Depending on the selected clock phase, a clock pulse will also be generated on the MS_CLK line. At the same time, with the opposite clock edge, the master latches and shifts in the data detected at its input line RXD. 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 (via the data width selection), the data transmitted by the master is contained in all the slaves' shift registers, while the master's shift register holds the data of the selected slave. In the master and all slaves, the content of the shift register are copied into the receive buffer RB and the receive interrupt line RIR is activated.

A slave device will immediately output the selected first bit (MSB or LSB of the transfer data) at line RXD when the contents of the transmit buffer are copied into the slave's shift register. Bit CON.BSY is not set until the first clock edge at SS_CLK appears. The slave device 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 already be used to clock in the first data bit. Thus, the slave's first data bit must already be valid at this time.

*Note: On the SSC, a transmission **and** a reception takes place at the same time, regardless of whether valid data has been transmitted or received.*

11.2.3 Half-Duplex Operation

In a Half-Duplex Mode, only one data line is necessary for both receiving **and** transmitting of data. The data exchange line is connected to both the MTSR and MRST pins of each device, the shift 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.

Similar to 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 this method, any corruptions on the common data exchange line are detected if the received data is not equal to the transmitted data.

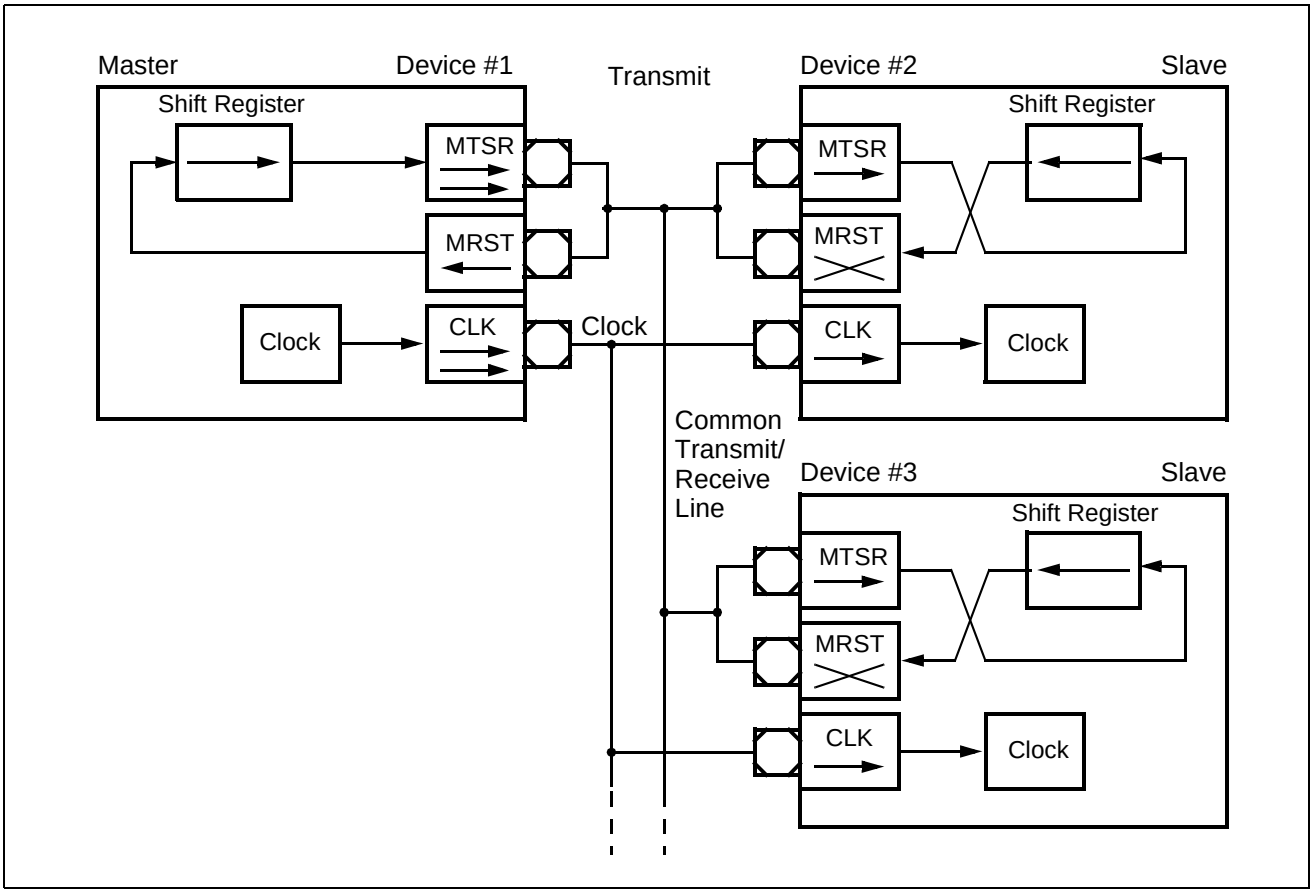


Figure 11-6 SSC Half-Duplex Configuration

11.2.4 Continuous Transfers

When the transmit interrupt request flag is set, it indicates that the transmit buffer TB is empty and ready to be loaded with the next transmit data. If TB has been reloaded by the time the current transmission is finished, the data is immediately transferred to the shift register and the next transmission 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 that can operate with or require more than 16 data bits per transfer. It is just 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 instance.

Note: Of course, this can happen only in multiples of the selected basic data width, because it would require disabling/enabling of the SSC to reprogram the basic data width on-the-fly.

11.2.5 Baudrate Generation

The serial channel SSC has its own dedicated 16-bit baudrate generator with 16-bit reload capability, allowing baudrate generation independent of the timers. **Figure 11-3** shows the baudrate generator. **Figure 11-7** shows the baudrate generator of the SSC in more detail.

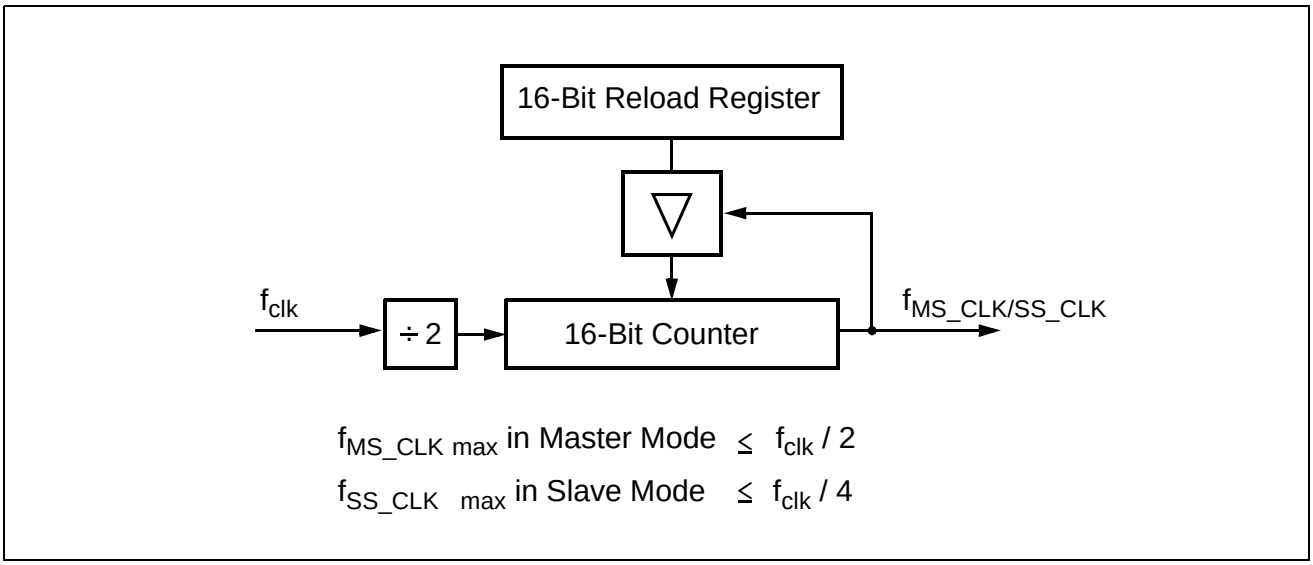


Figure 11-7 SSC Baudrate Generator

The baudrate generator is clocked with the module clock f_{clk}. The timer counts downwards. Register BR is the dual-function Baudrate Generator/Reload register. Reading BR, while the SSC is enabled, returns the content of the timer. Reading BR, while the SSC is disabled, returns the programmed reload value. In this mode the desired reload value can be written to BR.

Baudrate Timer Reload Register

The SSC baudrate timer reload register BR contains the 16-bit reload value for the baudrate timer.

BR
Baudrate Timer Reload Register **(Reset value: 0000_H)**

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

Note: Never write to BR while the SSC is enabled.

The formulas below calculate either the resulting baudrate for a given reload value, or the required reload value for a given baudrate:

$$\text{Baudrate} = \frac{f_{clk}}{2 * (
 + 1)} \qquad BR = \frac{f_{clk}}{2 * \text{Baudrate}} - 1$$

Field	Bits	Type	Description
BR_VALUE	[15:0]	rw	**Baudrate Timer/Reload Register Value** Reading BR returns the 16-bit content of the baudrate timer. Writing BR loads the baudrate timer reload register with BR_VALUE.

 represents the contents of the reload register, taken as unsigned 16-bit integer; while Baudrate is equal to f_{MS_CLK/SS_CLK} as shown in **Figure 11-7**.

The maximum baudrate that can be achieved when using a module clock of 60 MHz is 30 MBaud in Master Mode (with
 = 0000_H) or 15 MBaud in Slave Mode (with
 = 0001_H).

Table 11-1 lists some possible baudrates together with the required reload values and the resulting bit times, assuming a module clock of 60 MHz.

Table 11-1 Typical Baudrates of the SSC (f_{clk} = 60 MHz)

Reload Value	Baudrate (= f_{MS_CLK/SS_CLK})	Deviation
0000_H	30 MBaud (only in Master Mode)	0.0%
0001_H	15 MBaud	0.0%
$001D_H$	1 MBaud	0.0%
0027_H	750 kBaud	0.0%
$003B_H$	500 kBaud	0.0%
0095_H	200 kBaud	0.0%
$012B_H$	100 kBaud	0.0%
$FFFF_H$	457.76 Baud	0.0%

11.2.6 Error Detection Mechanisms

The SSC is able to detect four different error conditions. Receive Error and Phase Error are detected in all modes; Transmit Error and Baudrate Error only apply to Slave Mode. When an error is detected, the respective error flag is set and an error interrupt request will be generated by activating the EIR line (see **Figure 11-8**). 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 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.

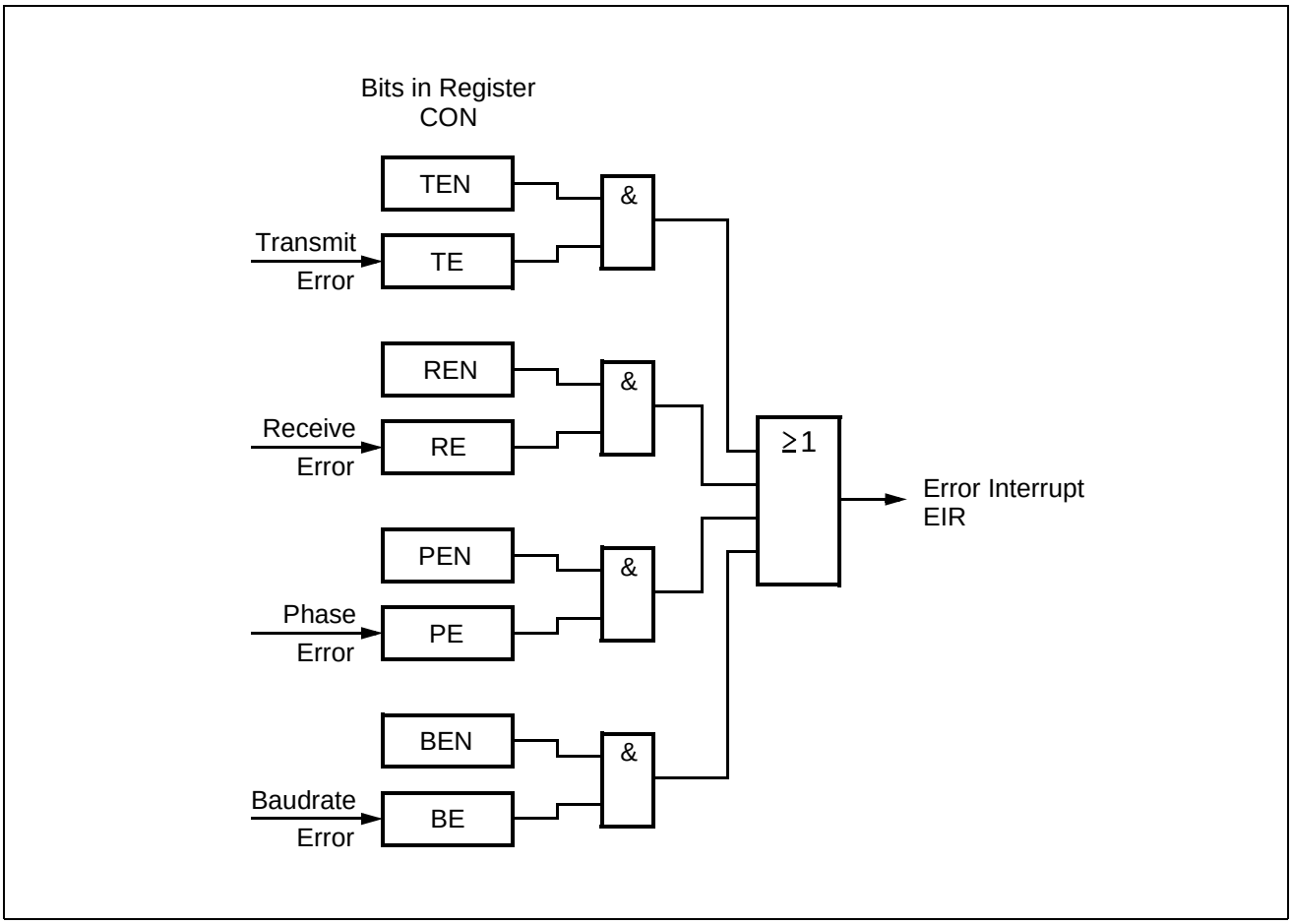


Figure 11-8 SSC Error Interrupt Control

A **Receive Error** (Master or Slave Mode) is detected when a new data frame is completely received but the previous data was not read out of the receive buffer register RB. This condition sets the error flag CON.RE and, when enabled via CON.REN, the error interrupt request line EIR. The old data in the receive buffer RB will be overwritten with the new value and is irretrievably lost.

A **Phase Error** (Master or Slave Mode) is detected when the incoming data at pin MRST (Master Mode) or MTSR (Slave Mode), sampled with the same frequency as the module clock, changes between one cycle before and two cycles after the latching edge of the shift clock signal SCLK. This condition sets the error flag CON.PE and, when enabled via CON.PEN, the error interrupt request line EIR.

A **Baudrate Error** (Slave Mode) is detected when the incoming clock signal deviates from the programmed baudrate by more than 100%, i.e. it either is more than double or less than half the expected baudrate. This condition sets the error flag CON.BE and, when enabled via CON.BEN, the error interrupt request line EIR. Using this error detection capability requires that the slave's baudrate generator is programmed to the same baudrate as the master device. This feature detects false additional, or missing pulses on the clock line (within a certain frame).

Note: If this error condition occurs and bit CON.REN=1, an automatic reset of the SSC will be performed in case of this error. This is done to re-initialize 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 (SS_CLK gets active) but the transmit buffer TB of the slave was not updated since the last transfer. This condition sets the error flag CON.TE and, when enabled via CON.TEN, the error interrupt request line EIR. 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 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; that is, their transmit buffers must be loaded with '$FFFF_H$' prior to any transfer.

Note: A slave with push/pull output drivers not selected for transmission, will normally have its output drivers switched. However, in order to avoid possible conflicts or misinterpretations, it is recommended to always load the slave's transmit buffer prior to any transfer.

The cause of an error interrupt request (receive, phase, baudrate, transmit error) can be identified by the error status flags in control register CON.

Note: In contrast to the error interrupt request line EIR, the error status flags CON.TE, CON.RE, CON.PE, and CON.BE, are not reset automatically upon entry into the error interrupt service routine, but must be cleared by software.

12 General Purpose Timer Unit

12.1 Introduction

The **G**eneral **P**urpose **T**imer Unit (GPT12E) provides very flexible multifunctional timer structures that may be used for timing, event counting, pulse width measurement, pulse generation, frequency multiplication, and other purposes. The GPT12E incorporates five 16-bit timers grouped into two timer blocks: Block 1 (GPT1) and Block 2 (GPT2). Each timer in each block can operate independently in a number of different modes, such as Gated Timer Mode or Counter Mode; or, each timer can be concatenated with another timer of the same block.

Block 1 contains three timers/counters with a maximum resolution of $f_{clk}/4$. The auxiliary timers of GPT1 may optionally be configured as reload or capture registers for the core timer.

Block 2 contains two timers/counters with a maximum resolution of $f_{clk}/2$. An additional CAPREL register supports capture and reload operation with extended functionality.

Note: Core Timer T6 may be concatenated with timers of other on chip peripherals.

The following summary identifies all features to be supported by the GPT12E:

- Timer Block 1:
  - Maximum resolution of $f_{clk}/4$
  - Three independent timers/counters
  - Concatenation of timers/counters can be done
  - Four operating modes (Timer Mode, Gated Timer Mode, Counter Mode, Incremental Interface Mode)
  - Separate interrupt lines for each timer/counter

- Timer Block 2:
  - Maximum resolution of $f_{clk}/2$
  - Two independent timers/counters
  - Concatenation of Timers/counters can be done
  - Three operating modes (Timer Mode, Gated Timer Mode, Counter Mode)
  - Extended capture/reload functions via 16-bit capture/reload register CAPREL
  - Separate interrupt lines for each timer/counter

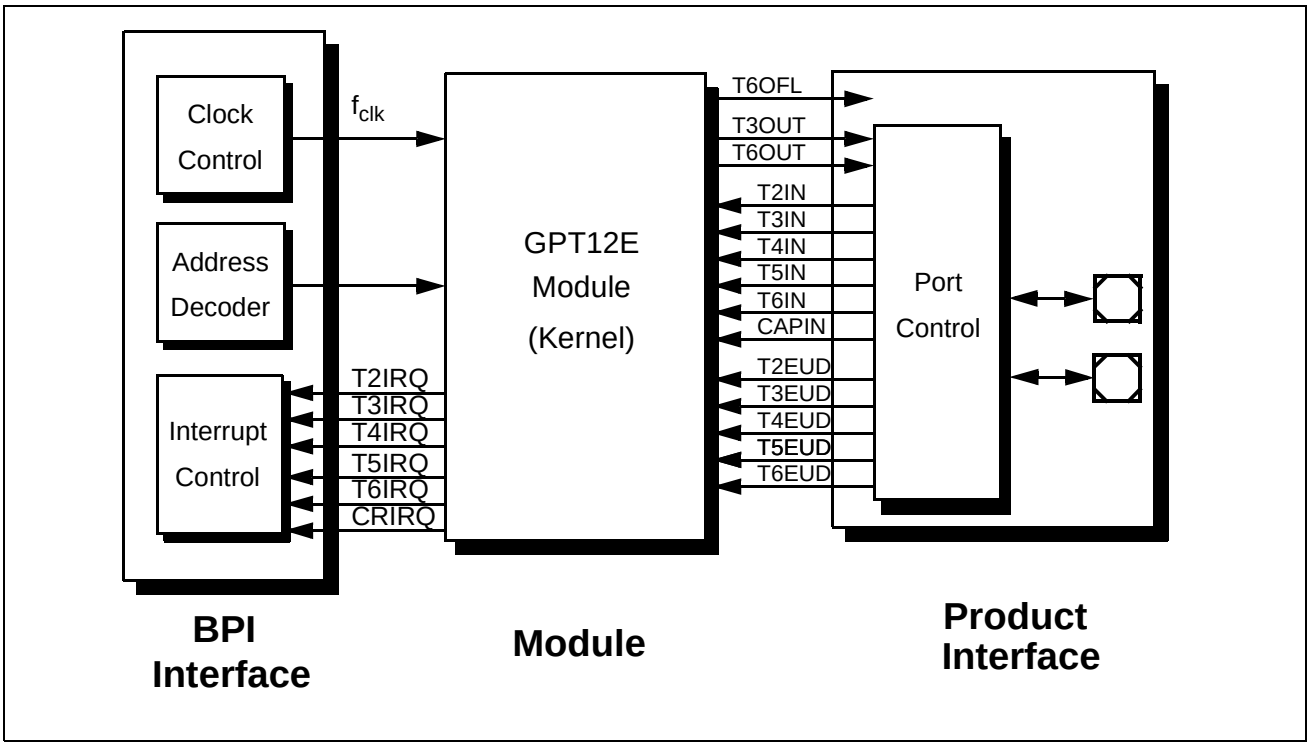


Figure 12-1 GPT12E Interface Diagram

*Note: **B**us **P**eripheral **I**nterface (BPI) is the connection to the on-chip bus system.*

12.2 Functional Description of Timer Block 1

All three timers of Block 1 (T2, T3, T4) can run in four basic modes: Timer Mode, Gated Timer Mode, Counter Mode, and Incremental Interface Mode. All timers can count up or down. Each timer of Block 1 is controlled by a separate control register TxCON.

Each timer has an input line, 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 line, **E**xternal **U**p/**D**own Control Input TxEUD. An overflow/underflow of core Timer T3 is indicated by the **O**utput **T**oggle **L**atch T3OTL whose state may be output on related signal line T3OUT. Additionally, the auxiliary timers T2 and T4 may be concatenated with core Timer T3 or may be used as capture or reload registers for core Timer T3. Concatenation of T3 with other timers is provided through line T3OTL.

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 **S**pecial **F**unction **R**egister (SFR) space. When any of the timer registers is written by the CPU in the state immediately before a timer increment, decrement, reload, or capture is to be performed, the CPU write operation has priority in order to guarantee correct results.

From a programmer's point of view, the GPT1 block is composed of a set of SFRs as summarized below. Those registers which are not part of the GPT1 block are shaded.

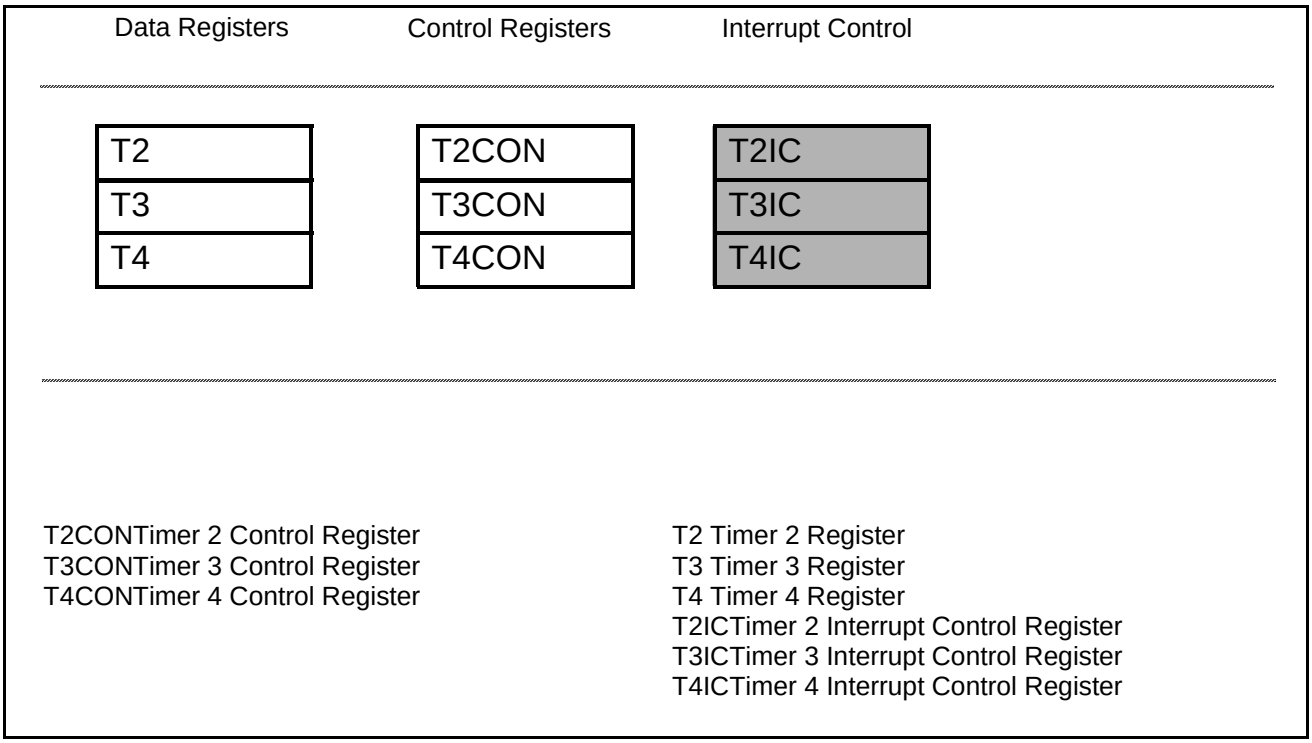


Figure 12-2 SFRs associated with Timer Block GPT1

All GPT1 registers are located in the SFR/ESFR memory space. The respective SFR addresses can be found in list of SFRs.

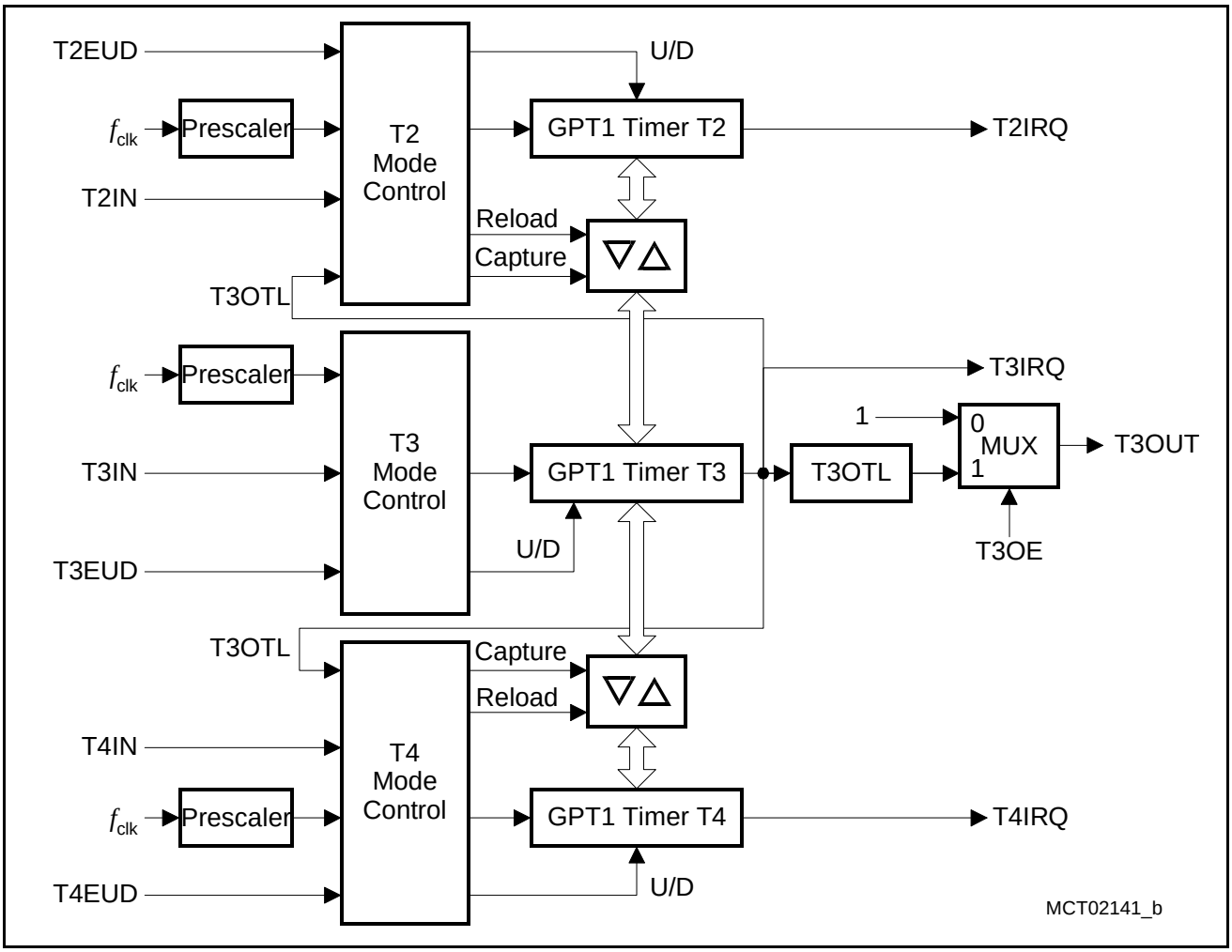
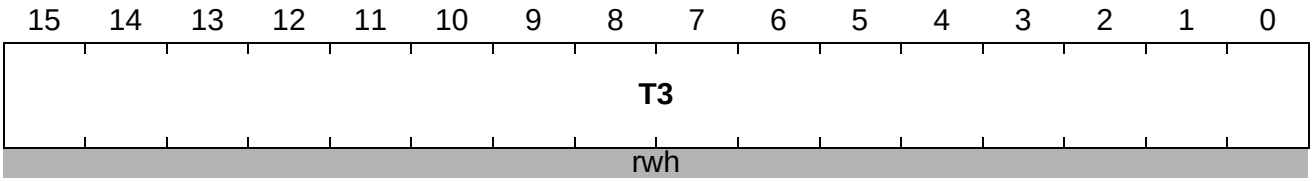


Figure 12-3 Structure of Timer Block 1

12.2.1 Core Timer T3

The operation of core Timer T3 is controlled by its bitaddressable control register T3CON.

T3
Timer 3 **(Reset value: 0000_H)**



Field	Bits	Typ	Description
T3	[15:0]	rwh	**Timer 3** Contains the current value of Timer 3.

T3CON
Timer 3 Control Register **(Reset value: 0000_H)**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T3 RDIR	T3 CH DIR	T3 EDG E	BPS1		T3 OTL	T3 OE	T3 UDE	T3 UD	T3R	T3M			T3I		
rh	rwh	rwh	rw		rwh	rw	rw	rw	rw	rw			rw		

Field	Bits	Typ	Description
T3I	[2:0]	rw	**Timer 3 Input Parameter Selection** Timer Mode: see **Table 12-2** for encoding Gated Timer Mode: see **Table 12-2** for encoding Counter Mode: see **Table 12-4** for encoding Incremental Interface Mode: see **Table 12-5** for encoding
T3M	[5:3]	rw	**Timer 3 Mode Control** 000 Timer Mode 001 Counter Mode 010 Gated Timer Mode with gate active low 011 Gated Timer Mode with gate active high 100 ***Reserved***. Do not use this combination! 101 ***Reserved***. Do not use this combination! 110 Incremental Interface Mode (Rotation Detection Mode) 111 Incremental Interface Mode (Edge Detection Mode)
T3R	[6]	rw	**Timer 3 Run Bit** 0 Stops Timer/counter 3 1 Runs Timer/counter 3
T3UD	[7]	rw	**Timer 3 Up/Down Control** (when T3UDE = 0) 0 Counts up 1 Counts down
T3UDE	[8]	rw	**Timer 3 External Up/Down Enable** 0 Counting direction is internally controlled by software 1 Counting direction is externally controlled by line T3EUD

Field	Bits	Typ	Description
T3OE	[9]	rw	**Overflow/Underflow Output Enable** 0 T3 overflow/underflow cannot be externally monitored via T3IN 1 T3 overflow/underflow may be externally monitored via T3IN
T3OTL	[10]	rwh	**Timer 3 Overflow Toggle Latch** Toggles on each overflow/underflow of T3. Can be set or reset by software.
BPS1	[12:11]	rw	**Timer Block Prescaler 1** The maximum input frequency[1)] 00 Timer Block 1 is f_{clk} / 8 01 Timer Block 1 is f_{clk} / 4 10 Timer Block 1 is f_{clk} / 32 11 Timer Block 1 is f_{clk} / 16
T3EDGE	[13]	rwh	**Timer 3 Edge Detection** The bit is set on each successful edge detection. The bit has to be reset by SW. 0 No count edge was detected 1 A count edge was detected
T3CHDIR	[14]	rwh	**Timer 3 Count Direction Change** The bit is set on a change of the count direction of Timer 3. The bit has to be reset by SW. 0 No change in count direction was detected 1 A change in count direction was detected
T3RDIR	[15]	rh	**Timer 3 Rotation Direction** 0 Timer 3 counts up 1 Timer 3 counts down

1) Additionally, the timer input frequency can be modified by T3I for Timer Mode, Gated Timer Mode and Counter Mode.

Run Control

The timer can be started or stopped by software through bit T3R. Setting bit T3R will start the timer; clearing T3R stops the timer.

In Gated Timer Mode, the timer will run only if T3R is set and the gate is active (high or low, as programmed).

Note: When bit T2RC/T4RC in timer control register T2CON/T4CON is set, T3R will also control (start and stop) auxiliary Timer T2/T4.

Count Direction Control

The count direction of the core timer can be controlled either by software or by the external input line, T3EUD. These options are selected by bits T3UD and T3UDE in control register, T3CON. When the up/down control is set by software (bit T3UDE is cleared), the count direction can be altered by setting or clearing bit T3UD. When T3UDE is set, line 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 12-1**. If T3UD is cleared and line T3EUD shows a low level, the timer is counting up. With a high level at T3EUD, the timer is counting down. If T3UD is set, a high level at line T3EUD specifies counting up, and a low level specifies counting down. The count direction can be changed whether or not the timer is running or not.

Note: When line T3EUD is used as external count direction control input, its associated port pin must be configured as input.

Table 12-1 Core Timer T3 Count Direction Control

Line T3EUD	**Bit T3UDE**	**Bit T3UD**	**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: The direction control works the same way for core Timer T3 and for auxiliary Timers T2 and T4.

Timer 3 Overflow/Underflow Monitoring

An overflow or underflow of Timer T3 will set clock bit T3OTL in control register T3CON. T3OTL can also be set or reset by software. Bit T3OE (overflow/underflow output enable) in register T3CON enables the state of T3OTL to be monitored via an external line, T3OUT. If this line is linked to an external port pin (configured as output), T3OUT can be used to control external hardware.

Additionally, T3OTL can be used in conjunction with auxiliary Timers T2 and T4. In this case, T3OTL serves as input for the counter function or as trigger source for the reload function of T2 and T4. T3OTL is internally connected for this functionality and it is not necessary to enable overflow/underflow output on T3OUT for this purpose.

Timer 3 in Timer Mode

Timer Mode for core Timer T3 is selected by setting bitfield T3M in register T3CON to '000_B'.

A block diagram of T3 in Timer Mode is shown in **Figure 12-4**.

In this mode, T3 is clocked with the module clock f_{clk} divided by a programmable prescaler block, controlled by bitfield T3I and BPS1. The input frequency f_{T3} for Timer T3 and its resolution r_{T3} are scaled linearly with lower module clock frequencies, as can be seen from the following formula:

$$f_{T3} = \frac{f_{clk}}{<BPS1> * 2^{<T3I>}} \qquad r_{T3}\ [ms] = \frac{<BPS1> * 2^{<T3I>}}{f_{clk}\ [MHz]}$$

Note: <BPS1> represents the prescaler value of the prescaler part controlled by bitfield BPS1. For the values, see the bit description in register T3CON.

Table 12-2 Timer 3 Input Parameter Selection: Timer and Gated Timer Modes

T3I	Prescaler for f_{clk} (BPS1 = 00)	Prescaler for f_{clk} (BPS1 = 01)	Prescaler for f_{clk} (BPS1 = 10)	Prescaler for f_{clk} (BPS1 = 11)
000	8	4	32	16
001	16	8	64	32
010	32	16	128	64
011	64	32	256	128
100	128	64	512	256
101	256	128	1024	512
110	512	256	2048	1024
111	1024	512	4096	2048

Table 12-3 Example for Timer 3 Frequencies and Resolutions

f_{clk} [MHz]	T3I	BPS1	f_{T3} [KHz]	r_{T3} [µs]
40	7	10	9.77	102.4
40	0	01	10000	0.1
50	0	00	6250	0.16
50	4	11	195.31	5.12
50	7	10	12.20	81.97

This formula also applies to T3 in Gated Timer Mode and to the auxiliary timers T2 and T4 in Timer Mode and Gated Timer Mode.

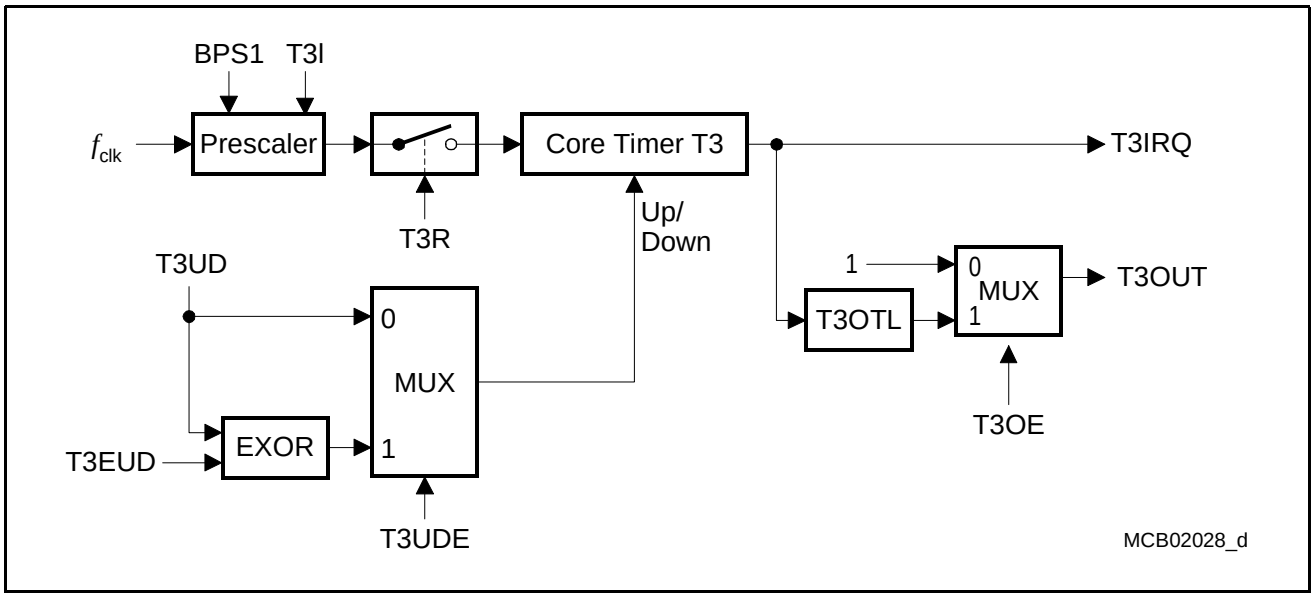


Figure 12-4 Block Diagram of Core Timer T3 in Timer Mode

Timer 3 in Gated Timer Mode

Gated Timer Mode for core Timer T3 is selected by setting bitfield 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 for the Timer Mode. However, the input clock to the timer in this mode is gated by the external input line T3IN (Timer T3 External Input); an associated port pin should be configured as input.

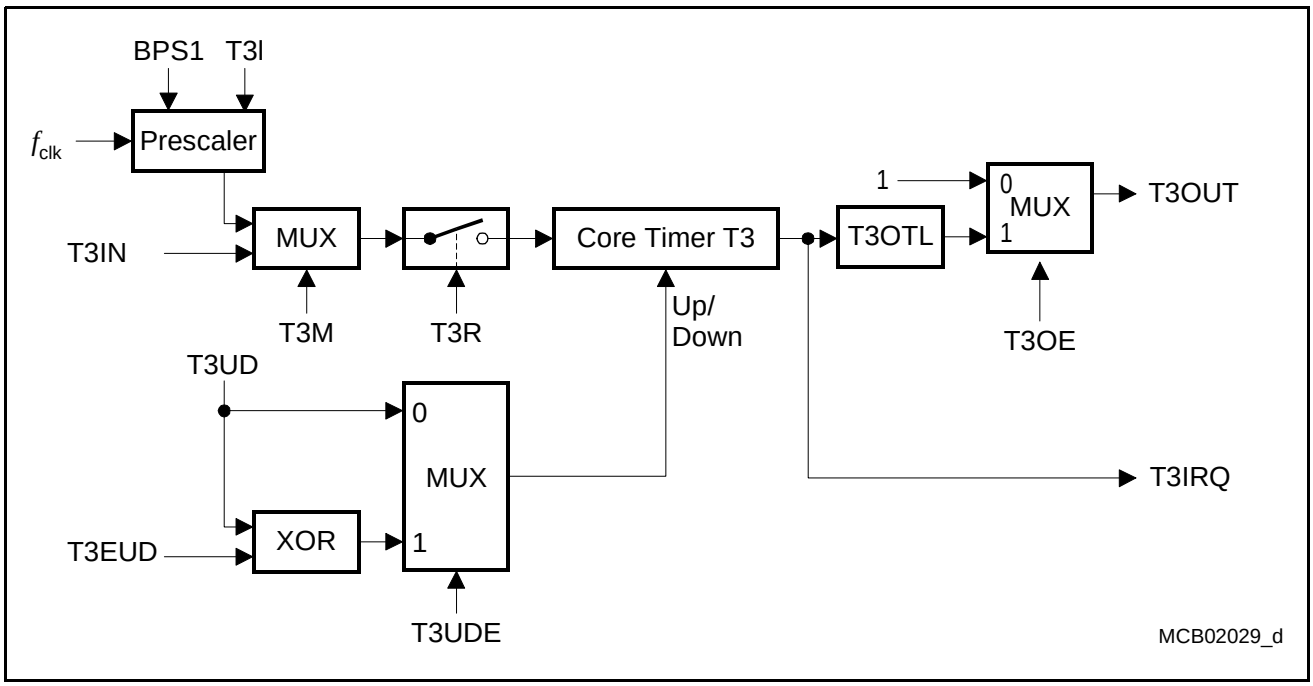


Figure 12-5 Block Diagram of Core Timer T3 in Gated Timer Mode

If T3M = '010_B', the timer is enabled when T3IN shows a low level. A high level at this line stops the timer. If T3M = '011_B', line T3IN must have a high level to enable the timer. Additionally, the timer can be turned on or off by software using bit T3R. The timer will run only if T3R is set and the gate is active. It will stop if either T3R is cleared or the gate is inactive.

Note: A transition of the gate signal at line T3IN does not cause an interrupt request.

Timer 3 in Counter Mode

Counter Mode for core Timer T3 is selected by setting bitfield T3M in register T3CON to '001_B'. In Counter Mode, Timer T3 is clocked by a transition at the external input line 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 line. Bitfield T3I in control register T3CON selects the triggering transition (see **Table 12-4**).

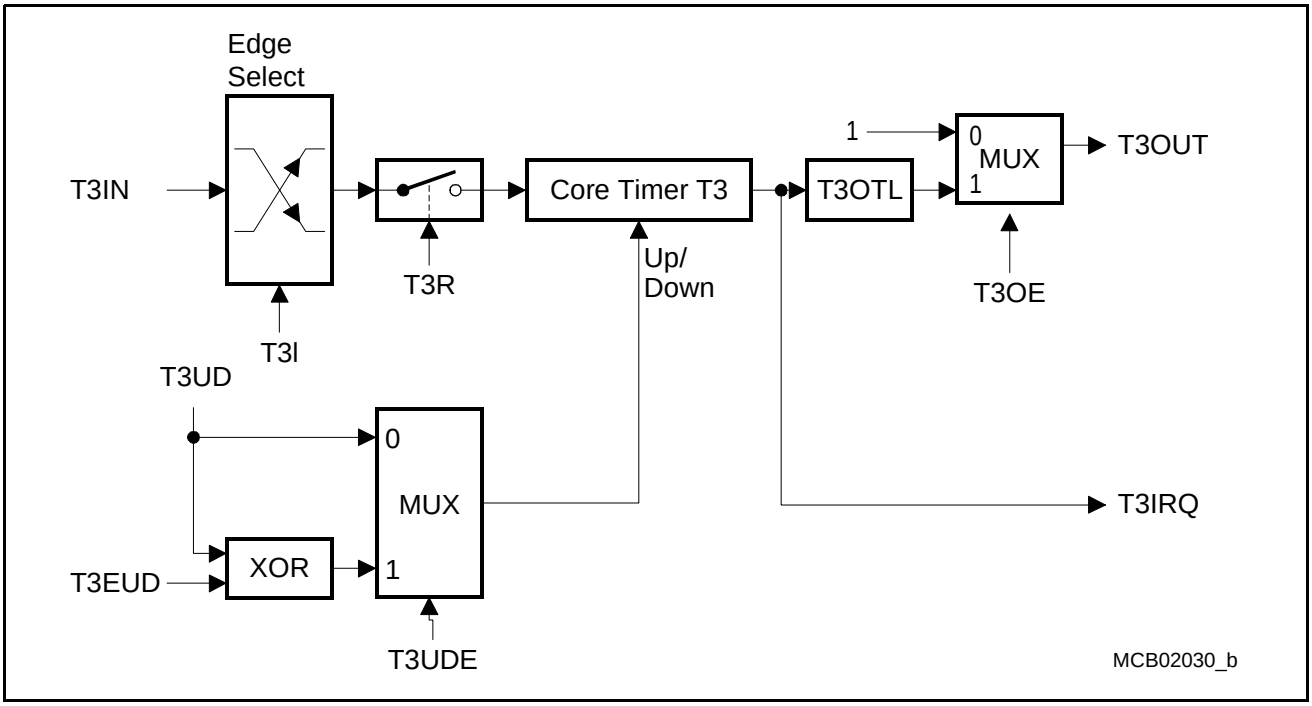


Figure 12-6 Block Diagram of Core Timer T3 in Counter Mode

Table 12-4 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 Mode operation, a port pin associated with line T3IN must be configured as input. The maximum input frequency allowed in Counter Mode is $f_{clk}/8$ (BPS1 ='01_B'). To ensure that a transition of the count input signal applied to T3IN is correctly recognized, its level should be held high or low for at least 4 f_{clk} cycles (BPS1 ='01_B') before it changes.

Timer 3 in Incremental Interface Mode

Incremental Interface Mode for core Timer T3 is selected by setting bitfield T3M in register T3CON to '110_B' or '111_B'. In Incremental Interface Mode, the two inputs associated with Timer T3 (T3IN, T3EUD) are used to interface to an external incremental encoder. T3 is clocked by each transition on one or both of the external input lines which gives 2-fold or 4-fold resolution of the encoder input.

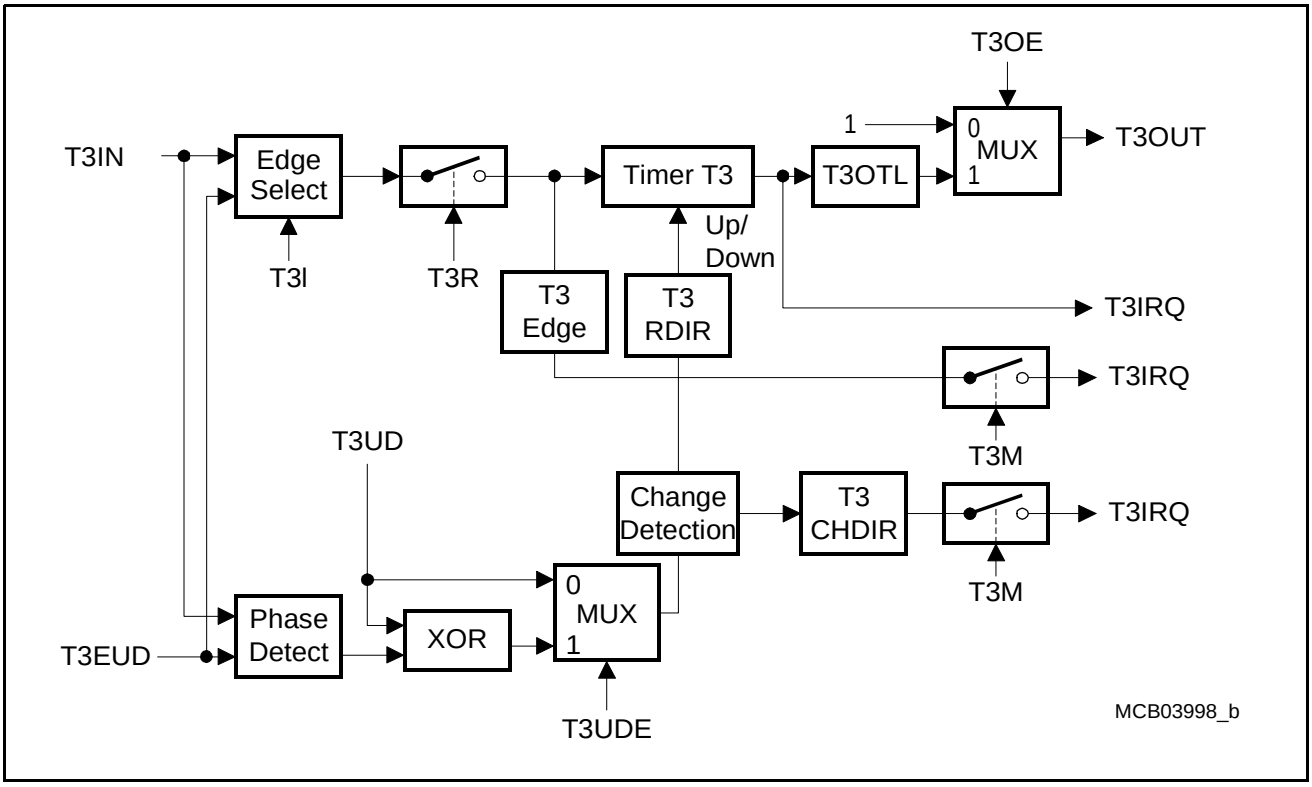


Figure 12-7 Block Diagram of Core Timer T3 in Incremental Interface Mode

Bitfield T3I in control register T3CON selects the triggering transitions (see **Table 12-5**). The sequence of the transitions of the two input signals is evaluated and generates count pulses as well as the direction signal. Depending on whether Rotation Detection Mode (T3M='110_B') or Edge Detection Mode (T3M='111_B') is chosen, an interrupt request on T3IRQ is generated. For Rotation Detection Mode, an interrupt is generated each time the count direction of Timer T3 changes. For Edge Detection Mode, an interrupt is generated each time a count action for Timer T3 occurs. Count direction, changes in the count direction, and count requests are monitored by status bits T3RDIR, T3CHDIR, and T3EDGE in register T3CON. T3 is modified automatically according to the speed and direction of the incremental encoder. Therefore, the contents of Timer T3 always represents the encoder's current position.

Table 12-5 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 microcontroller 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 in **Figure 12-8**). This greatly increases noise immunity.

Note: The third encoder output T0, which indicates the mechanical zero position, may be connected to an external interrupt input to trigger a reset of Timer T3.

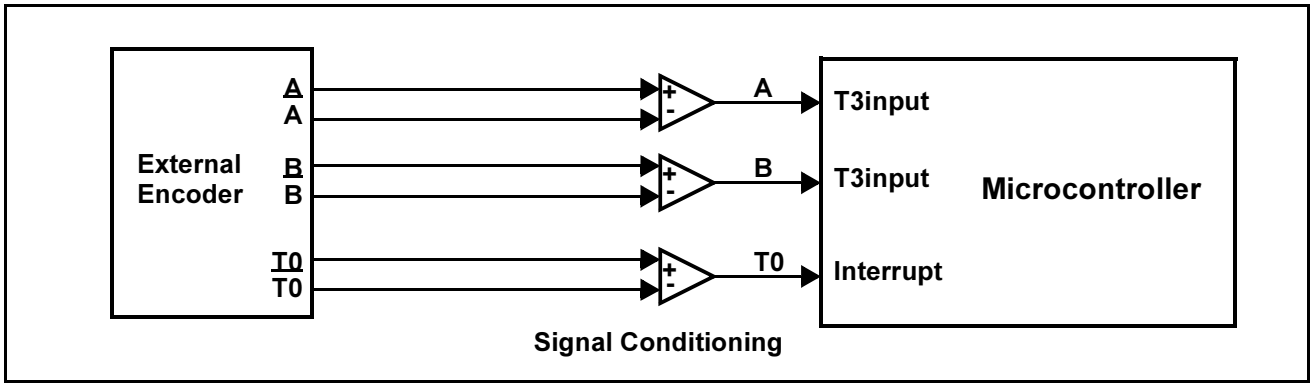


Figure 12-8 Interfacing the Encoder to the Microcontroller

The following conditions must be met for Incremental Interface Mode operation:

- Bitfield T3M must be '110_B' or '111_B'
- Pins associated with lines T3IN and T3EUD must be configured as inputs
- Bit T3UDE must be set to enable external direction control

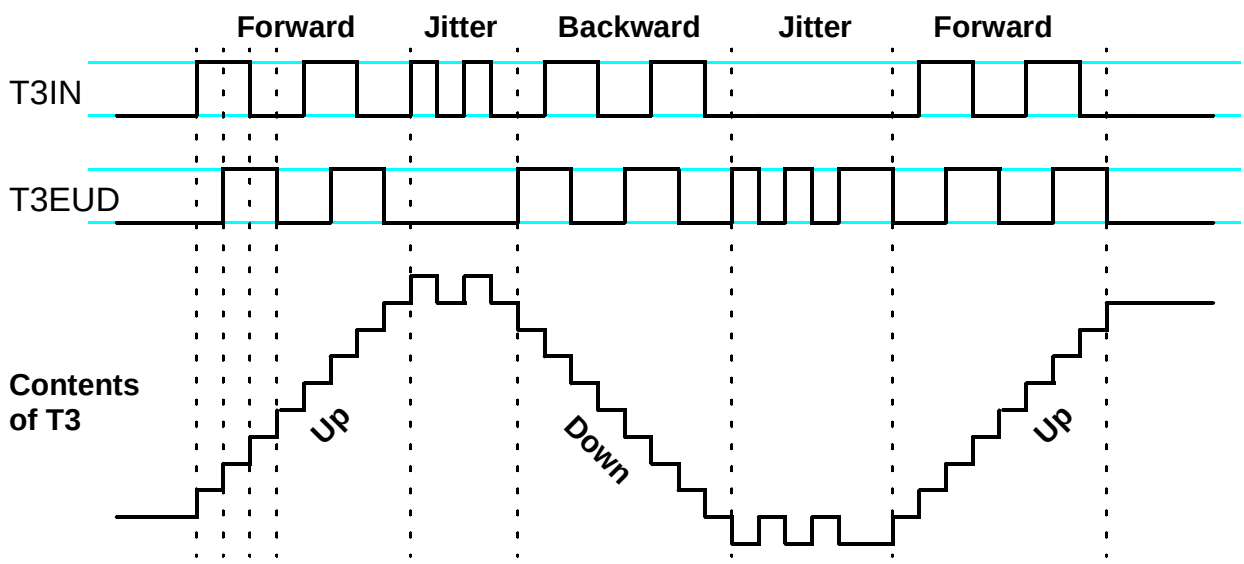
The maximum input frequency allowed in Incremental Interface Mode is $f_{clk}/8$ (T3BPS = '01_B'). To ensure that a transition of any input signal is correctly recognized, its level should be held high or low for at least 4 f_{clk} cycles (T3BPS = '01_B') before it changes.

In Incremental Interface Mode, the count direction is automatically derived from the sequence in which the input signals change, corresponding to the rotation direction of the connected sensor. **Table 12-6** summarizes the possible combinations.

Table 12-6 Core Timer T3 (Incremental Interface Mode) Count Direction

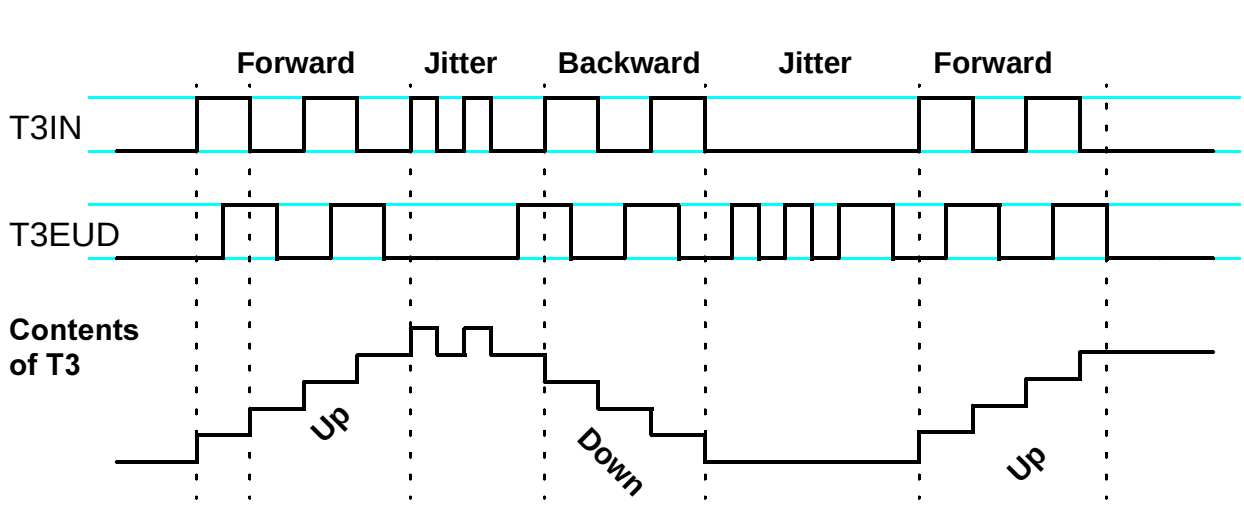
Level on respective other input	T3IN **Input**		T3EUD **Input**	
	Rising ↗	**Falling** ↘	**Rising** ↗	**Falling** ↘
High	Down	Up	Up	Down
Low	Up	Down	Down	Up

Figure 12-9 and **Figure 12-10** give examples of the operation of T3 to illustrate count signal generation and direction control. Each example also shows how input jitter, which might occur if the sensor rests near one of its switching points, is compensated.



Note: This example shows the timer behavior assuming that T3 counts upon any transition on any input, i.e. T3I = '011_B'.

Figure 12-9 Evaluation of the Incremental Encoder Signals



Note: This example shows the timer behavior assuming that T3 counts upon any transition on input T3IN, i.e. T3I = '001_B'.

Figure 12-10 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.

12.2.2 Auxiliary Timers T2 and T4

T2
Timer 2 **(Reset value: 0000_H)**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T2															
rwh															

T4
Timer 4 **(Reset value: 0000_H)**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T4															
rwh															

T2CON
Timer 2 Control Register **(Reset value: 0000_H)**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T2 RDIR	T2 CH DIR	T2 EDG E	T2 IR DIS	0		T2 RC	T2 UDE	T2 UD	T2R	T2M				T2I	
rh	rwh	rwh	rw	r		rw	rw	rw	rw	rw				rw	

T4CON
Timer 4 Control Register **(Reset value: 0000_H)**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T4 RDIR	T4 CH DIR	T4 EDG E	T4 IR DIS	0		T4 RC	T4 UDE	T4 UD	T4R	T4M				T4I	
rh	rwh	rwh	rw	r		rw	rw	rw	rw	rw				rw	

Field	Bits	Typ	Description
Tx	[15:0]	rwh	**Timer x** Contains the current value of Timer x.

Field	Bits	Typ	Description
TxI	[2:0]	rw	**Timer x Input Parameter Selection** Timer Mode: see **Table 12-7** for encoding Gated Timer Mode: see **Table 12-7** for encoding Counter Mode: see **Table 12-8** for encoding Incremental Interface Mode: see **Table 12-9** for encoding
TxM	[5:3]	rw	**Timer x Mode Control** (Basic Operating Mode) 000 Timer Mode 001 Counter Mode 010 Gated Timer Mode with gate active low 011 Gated Timer Mode with gate active high 100 Reload Mode 101 Capture Mode 110 Incremental Interface Mode (Rotation Detection Mode) 111 Incremental Interface Mode (Edge Detection Mode)
TxR	[6]	rw	**Timer x Run Bit** 0 Timer/Counter x stops 1 Timer/Counter x runs
TxUD	[7]	rw	**Timer x Up/Down Control** (when TxUDE = '0') 0 Counts Up 1 Counts Down
TxUDE	[8]	rw	**Timer x External Up/Down Enable** 0 Counting direction is internally controlled by software 1 Counting direction is externally controlled by line TxEUD
TxRC	[9]	rw	**Timer x Remote Control** 0 Timer/Counter x is controlled by its own run bit TxR 1 Timer/Counter x is controlled by the run bit of core Timer 3

Field	Bits	Typ	Description
TxIRDIS	[12]	rw	**Timer x Interrupt Disable** 0 Interrupt generation for TxCHDIR and TxEDGE interrupts in Incremental Interface Mode is enabled 1 Interrupt generation for TxCHDIR and TxEDGE interrupts in Incremental Interface Mode is disabled
TxEDGE	[13]	rwh	**Timer x Edge Detection** The bit is set on each successful edge detection. The bit has to be reset by software. 0 No count edge was detected 1 A count edge was detected
TxCHDIR	[14]	rwh	**Timer x Count Direction Change** The bit is set on a change of the count direction of timer x. The bit has to be reset by software. 0 No change in count direction was detected 1 A change in count direction was detected
TxRDIR	[15]	rh	**Timer x Rotation Direction** 0 Timer x counts up 1 Timer x counts down
0	[11:10]	r	**Reserved** for future use; reading returns 0; writing to these bit positions has no effect.

Both auxiliary timers T2 and T4 have exactly the same functionality. They can be configured for Timer Mode, Gated Timer Mode, Counter Mode, 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 which are present in all 3 timers of Timer Block 1 are controlled in the same bit positions and manner in each of the specific control registers.

Run control for auxiliary timers T2 and T4 can be handled by the associated run control bit T2R/T4R in register T2CON/T4CON. Alternatively, a remote control option (T2RC, T4RC set) may be enabled to start and stop T2/T4 via the run bit T3R of core Timer T3.

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 two exceptions:

- There is no TxOUT output line for T2 and T4.
- Overflow/underflow monitoring is not supported (no bit TxOTL in registers TxCON).

Table 12-7 Timer x Input Parameter Selection: Timer and Gated Timer Modes

TxI	Prescaler for f_{clk} (BPS1 = 00)	Prescaler for f_{clk} (BPS1 = 01)	Prescaler for f_{clk} (BPS1 = 10)	Prescaler for f_{clk} (BPS1 = 11)
000	8	4	32	16
001	16	8	64	32
010	32	16	128	64
011	64	32	256	128
100	128	64	512	256
101	256	128	1024	512
110	512	256	2048	1024
111	1024	512	4096	2048

Timers T2 and T4 in Counter Mode

In Counter Mode, Timers T2 and T4 can be clocked either by a transition at the respective external input line TxIN, or by a transition of T3OTL.

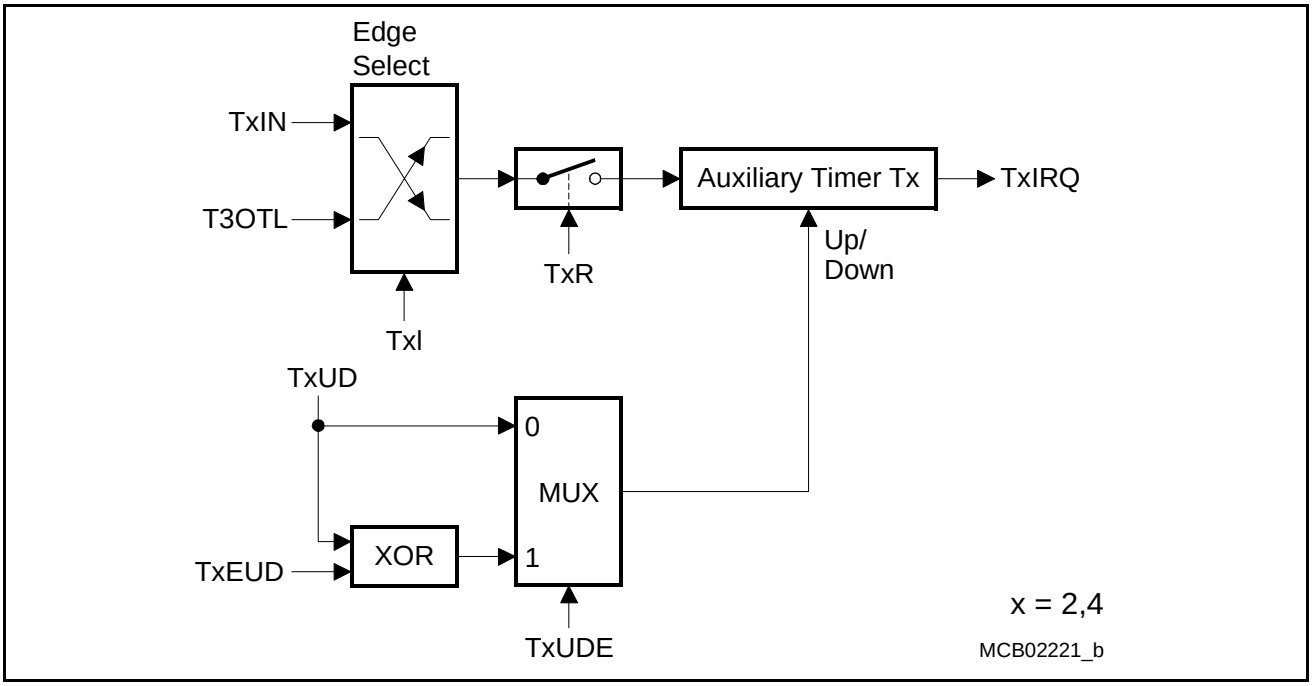


Figure 12-11 Block Diagram of an Auxiliary Timer in Counter Mode

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 line or at the output toggle latch T3OTL. Bitfield TxI in the respective control register TxCON selects the triggering transition (see **Table 12-8**).

Table 12-8 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 T3OTL
1 1 0	Negative transition (falling edge) of T3OTL
1 1 1	Any transition (rising or falling edge) of T3OTL

Note: Only state transitions of T3OTL caused by the overflow/underflow 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, an external pin associated with line TxIN must be configured as input. The maximum input frequency allowed in Counter Mode is $f_{clk}/8$ (BPS1 = '01'). To ensure that a transition of the count input signal applied to TxIN is correctly recognized, its level should be held for at least 4 f_{clk} cycles (BPS1 = '01') before it changes.

12.2.3 Timer Concatenation

Using T3OTL as a clock source for an auxiliary timer of Block 1 in Counter Mode concatenates core Timer T3 with the respective auxiliary timer. Depending on which transition of T3OTL is selected to clock the auxiliary timer, this concatenation forms a 32-bit or a 33-bit timer/counter.

- **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 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 core Timer T3. This configuration forms a 33-bit timer (16-bit core timer+T3OTL+16-bit auxiliary timer).

The count directions is not required to be the same in the two concatenated timers. This offers a wide variety of different configurations. T3 can operate in Timer Mode, Gated Timer Mode or Counter Mode in this case.

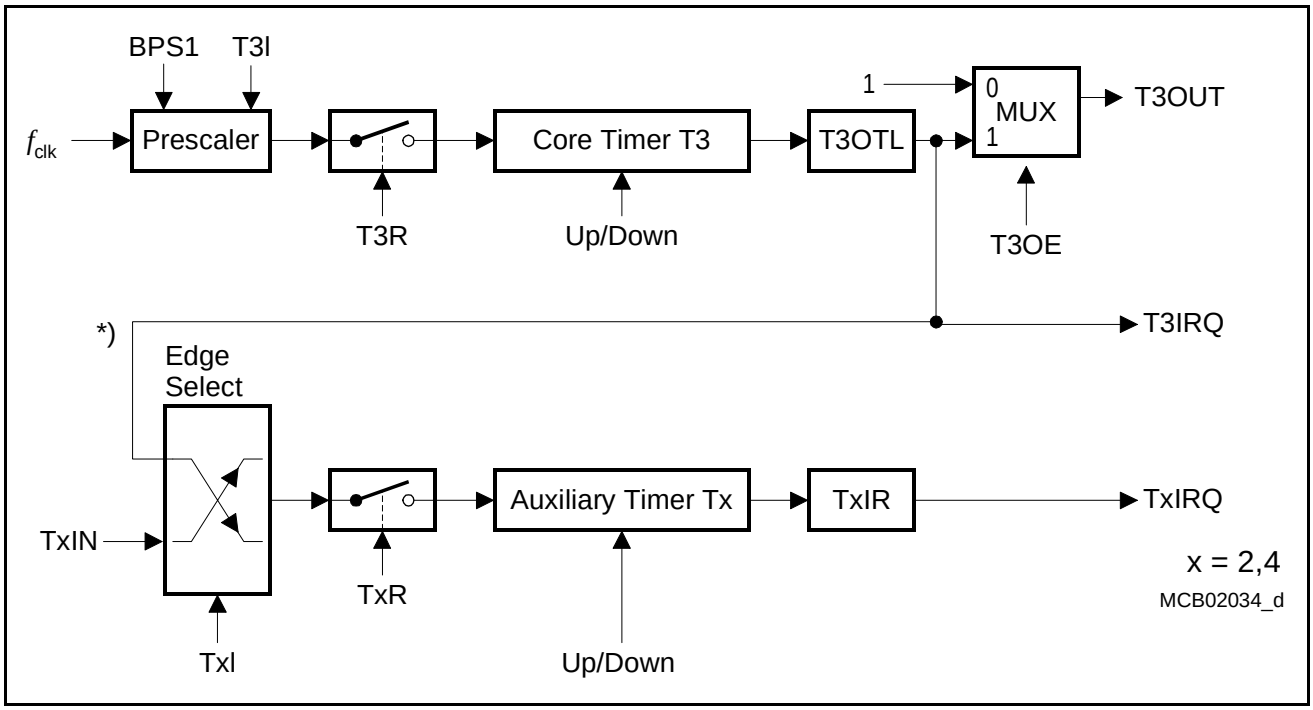


Figure 12-12 Concatenation of Core Timer T3 and an Auxiliary Timer

Note: Line ')' is affected by over/underflow of T3 only, but NOT by software modifications of T3OTL*

Auxiliary Timer in Reload Mode

Reload Mode for the auxiliary timers T2 and T4 is selected by setting bitfield TxM in the respective register TxCON to '100_B'. In Reload Mode, 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 12-8**). That is, 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.

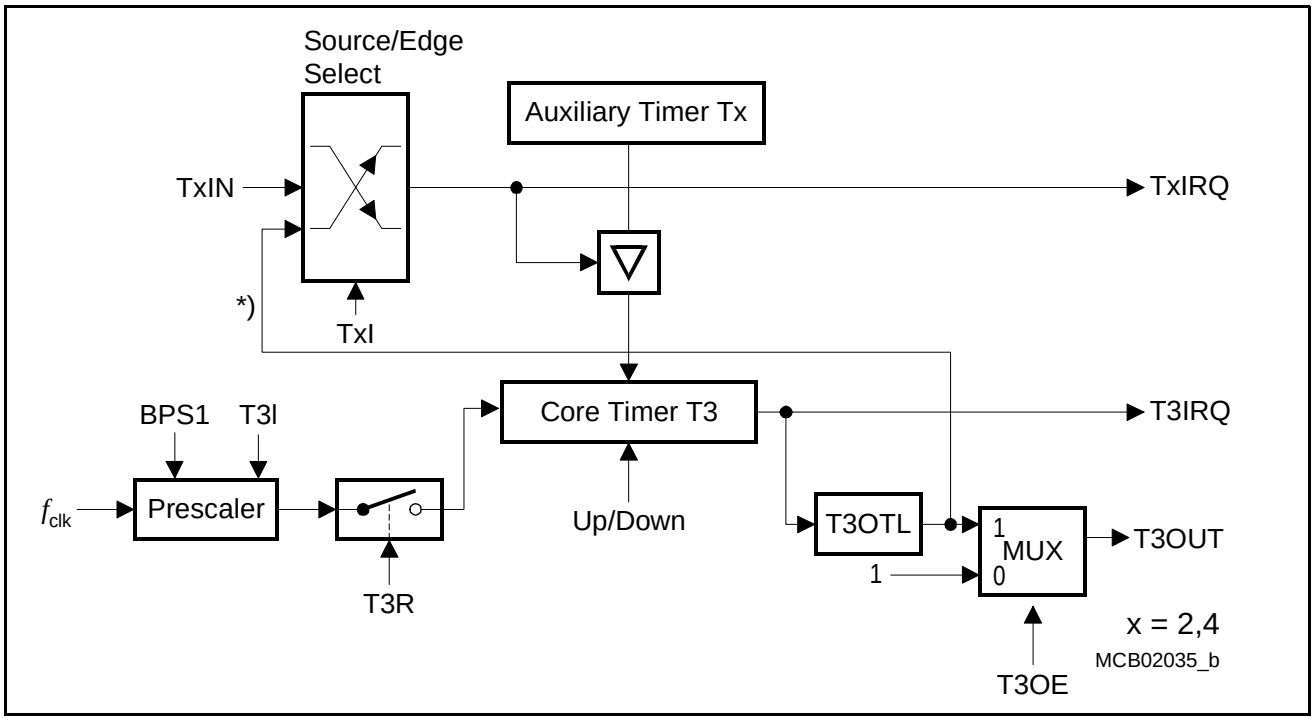


Figure 12-13 GPT1 Auxiliary Timer in Reload Mode

Note: Line ')' is affected by over/underflow of T3 only, but NOT by software modifications of T3OTL*

Upon a trigger signal, T3 is loaded with the contents of the respective timer register (T2 or T4) and T2IRQ or T4IRQ is driven high.

Note: When a T3OTL transition is selected for the trigger signal, the interrupt request flag T3IR will 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. Depending on the selected active transition, the following functions can be performed:

- 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 **P**ulse **W**idth **M**odulation (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.

The **Figure 12-14** shows an example for the generation of a PWM signal using the alternate reload mechanism. T2 defines the high time of the PWM signal (reloaded on positive transitions) and T4 defines the low time of the PWM signal (reloaded on negative transitions). The PWM signal can be output on line T3OUT if the enable bit T3OE is set. Using this method, the high and low time of the PWM signal can be varied over a wide range.

Note: T3OTL is accessible via software and may be changed, if required, to modify the PWM signal. However, this will NOT trigger the reloading of T3.

Note: An associated port pin linked to line T3OUT *should be configured as output.*

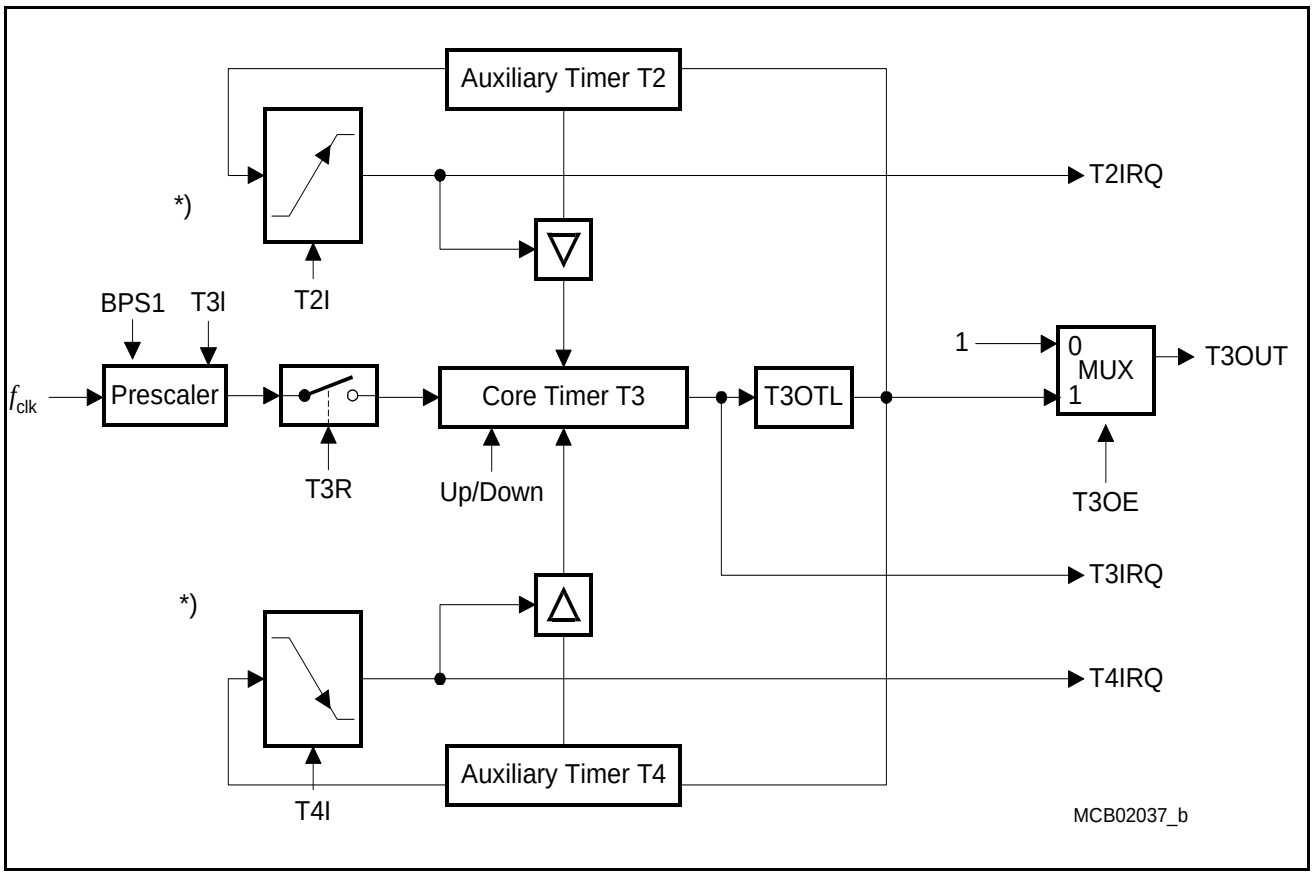


Figure 12-14 GPT1 Timer Reload Configuration for PWM Generation

Note: Line ')' is affected by over/underflow of T3 only, NOT by software modifications of T3OTL*

Note: It should be avoided to select the same reload trigger event for both auxiliary timers. In this 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 bitfield 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 line 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 bitfield TxI are used to select the active transition (see **Table 12-8**), 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.

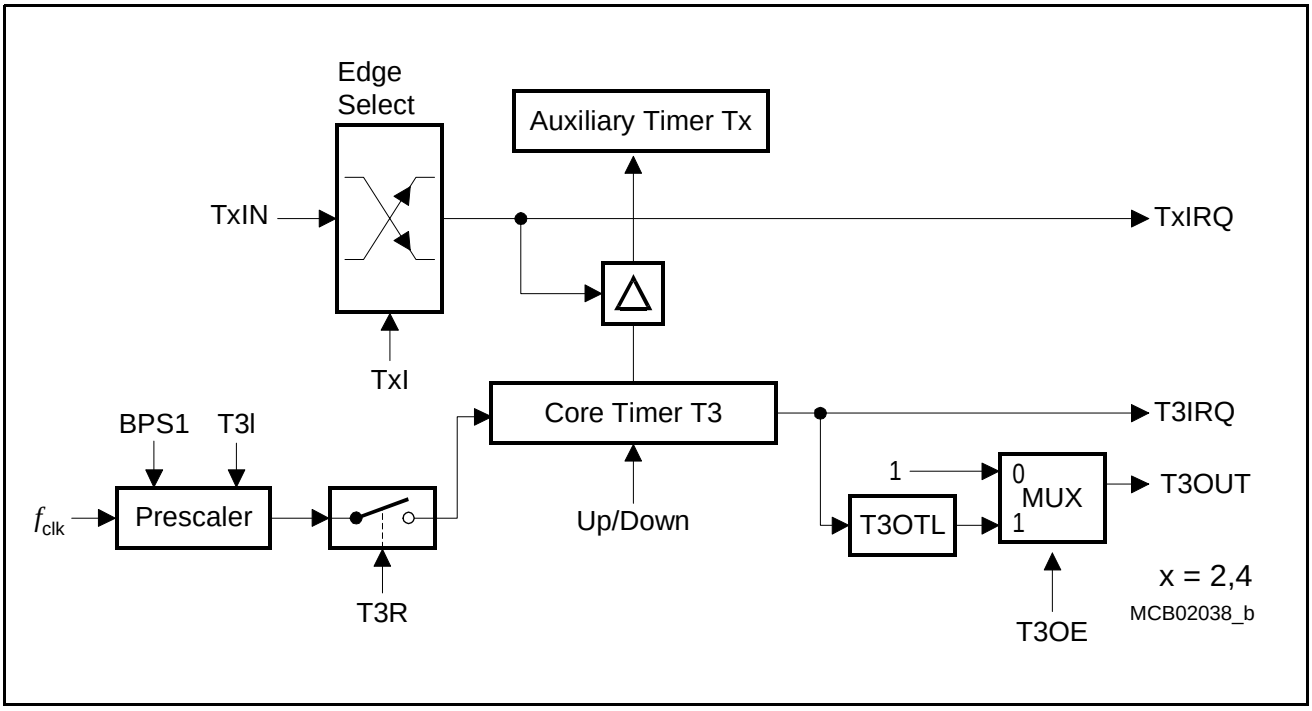


Figure 12-15 Auxiliary Timer of Timer Block 1 in Capture Mode

Upon a trigger (selected transition) at the corresponding input line TxIN, the contents of the core timer are loaded into the auxiliary timer register and the associated interrupt request line TxIRQ will be driven high.

Note: Port pins associated with T2IN and T4IN must be configured to Input, and the level of the capture trigger signal should be held high or low for at least 4 f_{clk} (BPS1 = '01') cycles before it changes to ensure correct edge detection.

Auxiliary in Incremental Interface Mode

When auxiliary Timers T2 and T4 are programmed to Incremental Interface Mode, their operation is the same as described for core Timer T3. The descriptions, figures, and tables apply accordingly with two exceptions:

- There is no TxOUT output line for T2 and T4.
- Overflow/underflow monitoring is not supported (no bit TxOTL).

Table 12-9 Timer x Input Parameter Selection for Incremental Interface Mode

TxI	Triggering Edge for Counter Update
000	None. Counter Tx stops
001	Any transition (rising or falling edge) on TxIN
010	Any transition (rising or falling edge) on TxEUD
011	Any transition (rising or falling edge) on TxIN or TxEUD
1XX	***Reserved***. Do not use this combination!

12.3 Functional Description of Timer Block 2

Timer Block 2 includes the two Timers T5 (referred to as the auxiliary timer) and T6 (referred to as the core timer), and the 16-bit capture/reload register CAPREL. Each timer of Block 2 is controlled by a separate control register, TxCON.

Each timer has an input line (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 line. An overflow/underflow of core Timer T6 is indicated by bit T6OTL whose state may be output on related line T6OUT and on line T6OFL. Core Timer T6 may be reloaded with the contents of CAPREL.

The toggle bit also supports the concatenation of T6 with auxiliary Timer T5, while concatenation of T6 with other timers is provided through line T6OUT. Triggered by an external signal, the contents of T5 can be captured into register CAPREL, and T5 may optionally be cleared. Both timer (T6 and T5) can count up or down, and the current timer value can be read or modified by the CPU in the non-bitaddressable SFRs T6 and T5.

From a programmer's point of view, the GPT2 block is composed of a set of SFRs as summarized below. Those registers which are not part of the GPT2 block are shaded.

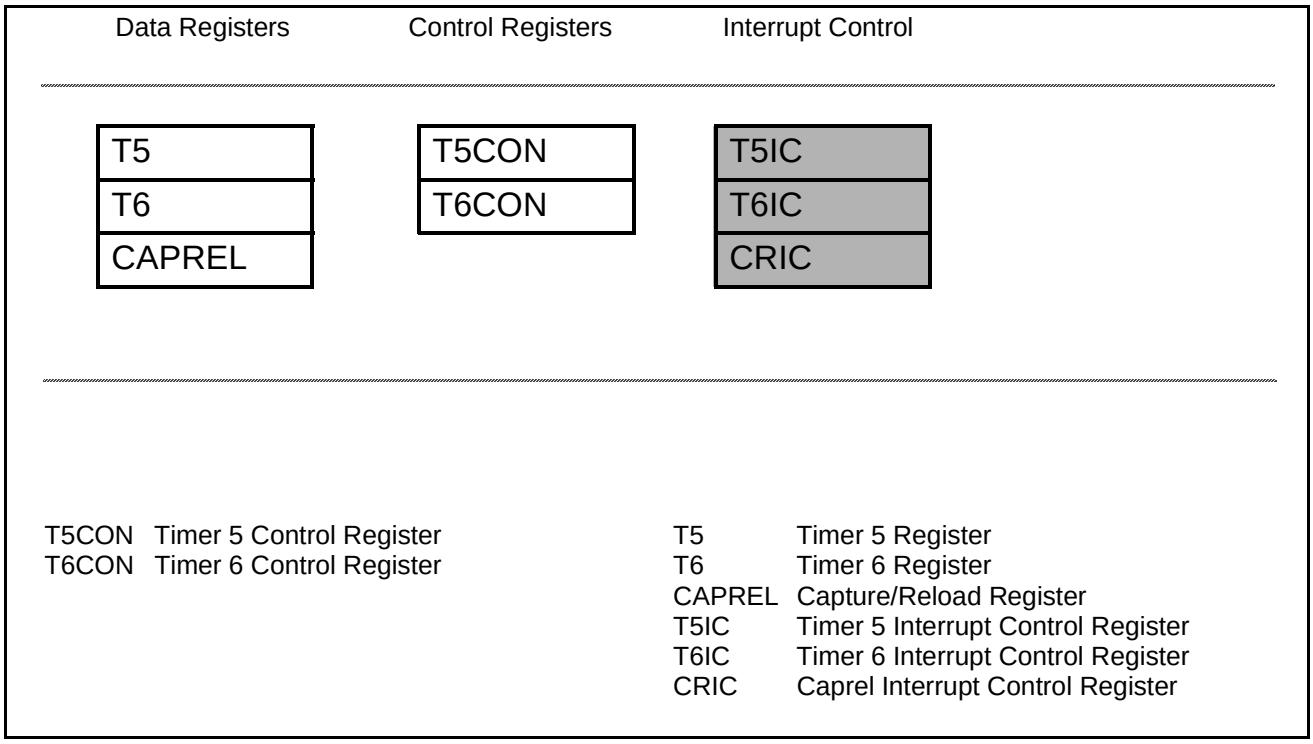


Figure 12-16 SFRs associated with Timer Block GPT2

All GPT2 registers are located in the SFR/ESFR memory space. The respective SFR addresses can be found in list of SFRs.

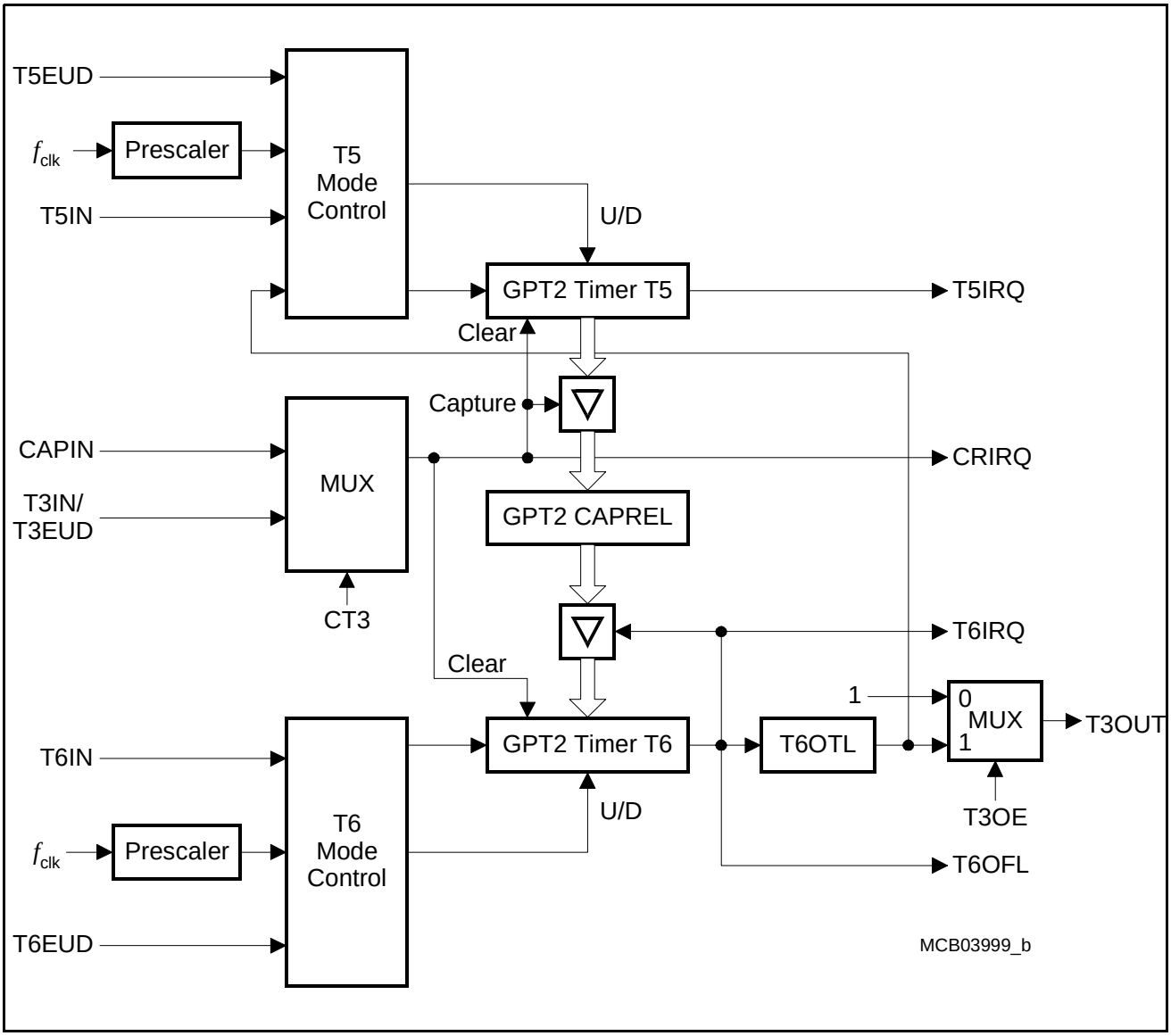
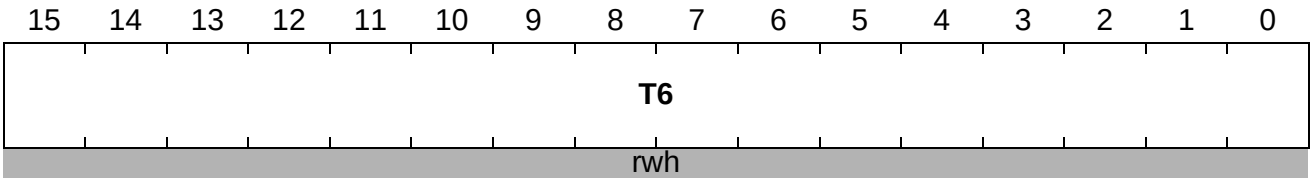


Figure 12-17 Structure of Timer Block 2

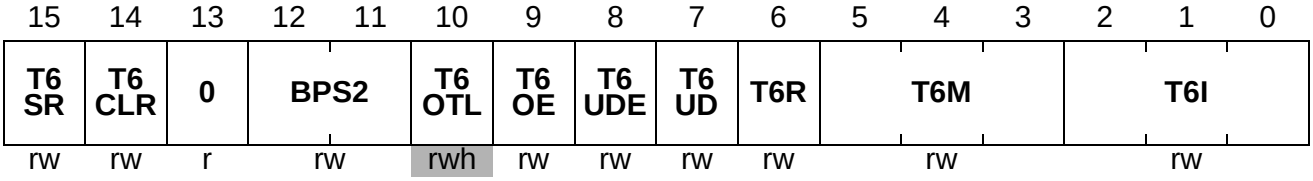
12.3.1 Core Timer T6

The operation of the core Timer T6 is controlled by its bitaddressable control register T6CON.

T6
Timer 6 **(Reset value: 0000_H)**



T6CON
Timer 6 Control Register **(Reset value: 0000_H)**



Field	Bits	Typ	Description
T6	[15:0]	rwh	**Timer 6** Contains the current value of Timer 6.

Field	Bits	Typ	Description
T6I	[2:0]	rw	**Timer 6 Input Parameter Selection** Timer Mode: see **Table 12-11** for encoding Gated Timer Mode: see **Table 12-11** for encoding Counter Mode: see **Table 12-12** for encoding
T6M	[5:3]	rw	**Timer 6 Mode Control** (Basic Operating Mode) 000 Timer Mode 001 Counter Mode 010 Gated Timer Mode with gate active low 011 Gated Timer with gate active high 1XX ***Reserved***. Do not use this combination!
T6R	[6]	rw	**Timer 6 Run Bit** 0 Timer/Counter 6 stops 1 Timer/Counter 6 runs

Field	Bits	Typ	Description
T6UD	[7]	rw	**Timer 6 Up / Down Control** (when T6UDE = '0') 0 Counts Up 1 Counts Down
T6UDE	[8]	rw	**Timer 6 External Up/Down Enable** 0 Counting direction is internally controlled by software 1 Counting direction is externally controlled by line T6EUD
T6OE	[9]	rw	**Overflow/Underflow Output Enable** 0 T6 overflow/underflow can not be externally monitored 1 T6 overflow/underflow may be externally monitored via T6OUT
T6OTL	[10]	rwh	**Timer 6 Output Toggle Latch** Toggles on each overflow/underflow of T6. Can be set or reset by software.
BPS2	[12:11]	rw	**Timer Block Prescaler 2** The maximum input frequency[1)] 00 Timer Block 2 is f_{clk} / 4 01 Timer Block 2 is f_{clk} / 2 10 Timer Block 2 is f_{clk} / 16 11 Timer Block 2 is f_{clk} / 8
T6CLR	[14]	rw	**Timer 6 Clear Bit** 0 Timer 6 is not cleared on a capture event 1 Timer 6 is cleared on a capture event
T6SR	[15]	rw	**Timer 6 Reload Mode Enable** 0 Reload from register CAPREL disabled 1 Reload from register CAPREL enabled
0	[13]	r	**Reserved** for future use; reading returns 0; writing to these bit positions has no effect.

1) Additionally, the timer input frequency can be modified by T6I for Timer Mode, Gated Timer Mode and Counter Mode.

Timer 6 Run Bit

The timer can be started or stopped by software through bit T6R (Timer T6 Run Bit). Setting bit T6R will start the timer; clearing T6R stops the timer.

In Gated Timer Mode, the timer will run only if T6R is set and the gate is active (high or low, as programmed).

Note: When bit T5RC is set, bit T6R will also control (start and stop) auxiliary Timer T5.

Count Direction Control

The count direction of the core timer can be controlled either by software or by the **E**xternal **U**p/**D**own control input line (T6EUD). These options are selected by bits T6UD and T6UDE in control register T6CON. When the up/down control is done by software (bit T6UDE is cleared), the count direction can be altered by setting or clearing bit T6UD. When T6UDE is set, line T6EUD is selected to be the controlling source of the count direction. However, bit T6UD can still be used to reverse the actual count direction, as shown in the table below. If T6UD is cleared and line T6EUD shows a low level, the timer is counting up. With a high level at T6EUD the timer is counting down. If T6UD is set, a high level at line T6EUD specifies counting up, and a low level specifies counting down. The count direction can be changed whether the timer is running or not.

Table 12-10 Core Timer T6 Count Direction Control

Line T6EUD	**Bit T6UDE**	**Bit T6UD**	**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: The direction control works the same for core Timer T6 and for auxiliary Timer T5.

Timer 6 Overflow/Underflow Monitoring

An overflow or underflow of Timer T6 will toggle T6OTL in control register T6CON. T6OTL can also be set or reset by software. Bit T6OE in register T6CON enables the state of T6OTL to be monitored via the external output line T6OUT. An associated port pin must be configured as output.

Additionally, T6OTL can be used in conjunction with the timer over/underflow as an input for the counter function of auxiliary Timer T5. For this purpose, the state of T6OTL does not have to be available at line T6OUT, because an internal connection is provided for this option.

An overflow or underflow of Timer T6 can also be used to clock other timers. For this purpose, there is the special output line T6OFL.

Timer 6 in Timer Mode

Timer Mode for core Timer T6 is selected by setting bitfield T6M in register T6CON to '000_B'. In this mode, T6 is clocked with the module clock divided by a programmable prescaler, as selected by bitfield T6I. The input frequency f_{T6} for Timer T6 and its resolution r_{T6} are scaled linearly with lower clock frequencies f_{clk}, as can be seen from the following formula:

$$f_{T6} = \frac{f_{clk}\,[MHz]}{<BPS2> * 2^{<T6I>}} \qquad r_{T6}\,[ms] = \frac{<BPS2> * 2^{<T6I>}}{f_{clk}\,[MHz]}$$

Note: <BPS2> represents the prescaler value of the prescaler part controlled by bitfield BPS2. For the values, see the bit description in register T6CON.

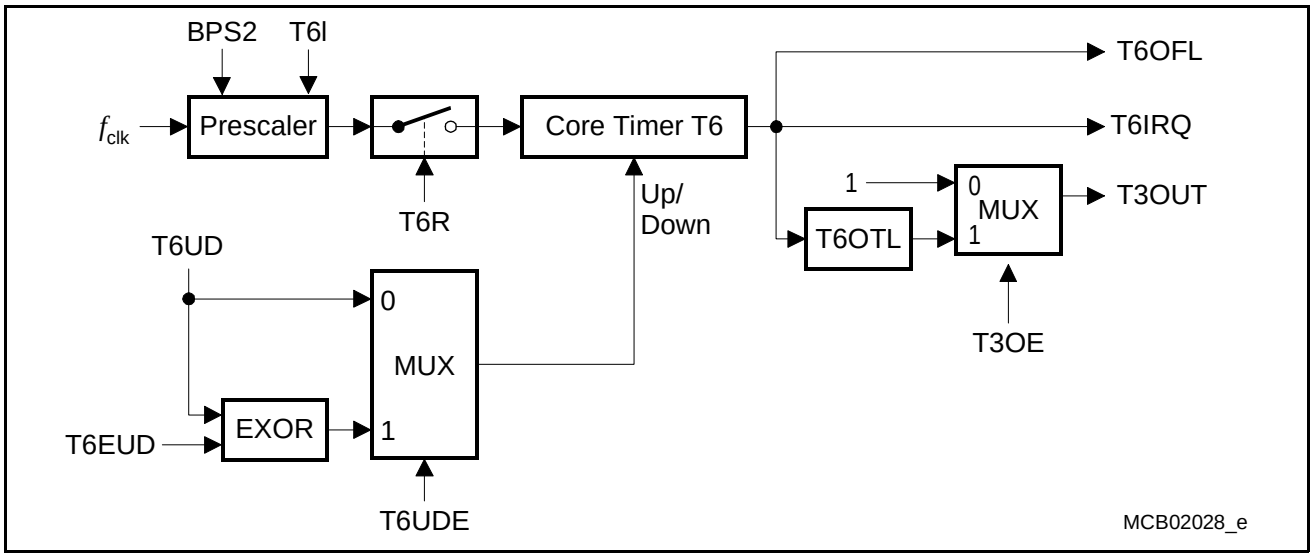


Figure 12-18 Block Diagram of Core Timer T6 in Timer Mode

Table 12-11 Timer 6 Input Parameter Selection: Timer and Gated Timer Modes

T6I	Prescaler for f_{clk} (BPS2 = 00)	Prescaler for f_{clk} (BPS2 = 01)	Prescaler for f_{clk} (BPS2 = 10)	Prescaler for f_{clk} (BPS2 = 11)
000	4	2	16	8
001	8	4	32	16
010	16	8	64	32
011	32	16	128	64

Table 12-11 Timer 6 Input Parameter Selection: Timer and Gated Timer Modes

T6I	Prescaler for f_{clk} (BPS2 = 00)	Prescaler for f_{clk} (BPS2 = 01)	Prescaler for f_{clk} (BPS2 = 10)	Prescaler for f_{clk} (BPS2 = 11)
100	64	32	256	128
101	128	64	512	256
110	256	128	1024	512
111	512	256	2048	1024

Timer 6 in Gated Timer Mode

Gated Timer Mode for core Timer T6 is selected by setting bitfield T6M in register T6CON to '010_B' or '011_B'. Bit T6M.0 (T6CON.3) selects the active level of the gate input. In Gated Timer Mode, the same options for the input frequency as for the Timer Mode are available. However, in this mode, the input clock to the timer is gated by the external input line T6IN.

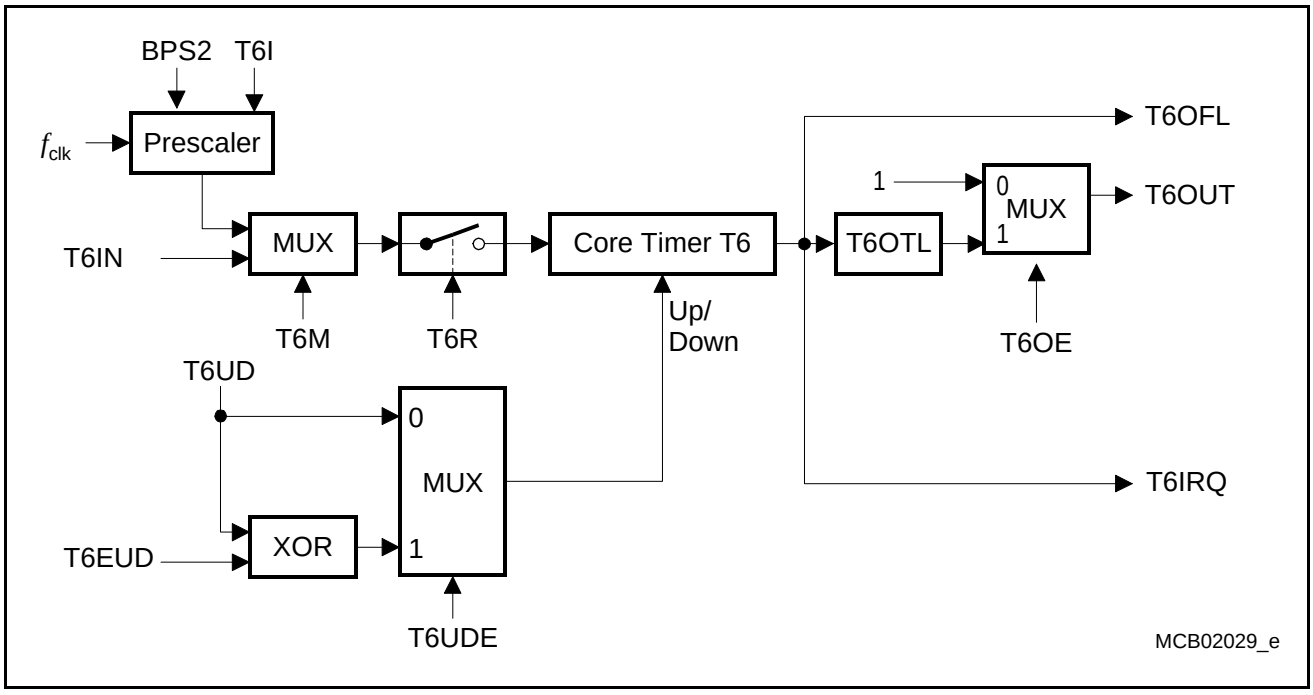


Figure 12-19 Block Diagram of Core Timer T6 in Gated Timer Mode

If T6M.0 = '0', the timer is enabled when T6IN shows a low level. A high level at this line stops the timer. If T6M.0 = '1', line T6IN must have a high level to enable the timer. Additionally, the timer can be turned on or off by software using bit T6R. The timer will run only if T6R is set and the gate is active. It will stop if either T6R is cleared or the gate is inactive.

Note: A transition of the gate signal at line T6IN *does not cause an interrupt request.*

Timer 6 in Counter Mode

Counter Mode for core Timer T6 is selected by setting bitfield T6M in register T6CON to '001_B'. In Counter Mode, Timer T6 is clocked by a transition at the external input line T6IN. 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 line. Bitfield T6I in control register T6CON selects the triggering transition (see **Table 12-12**).

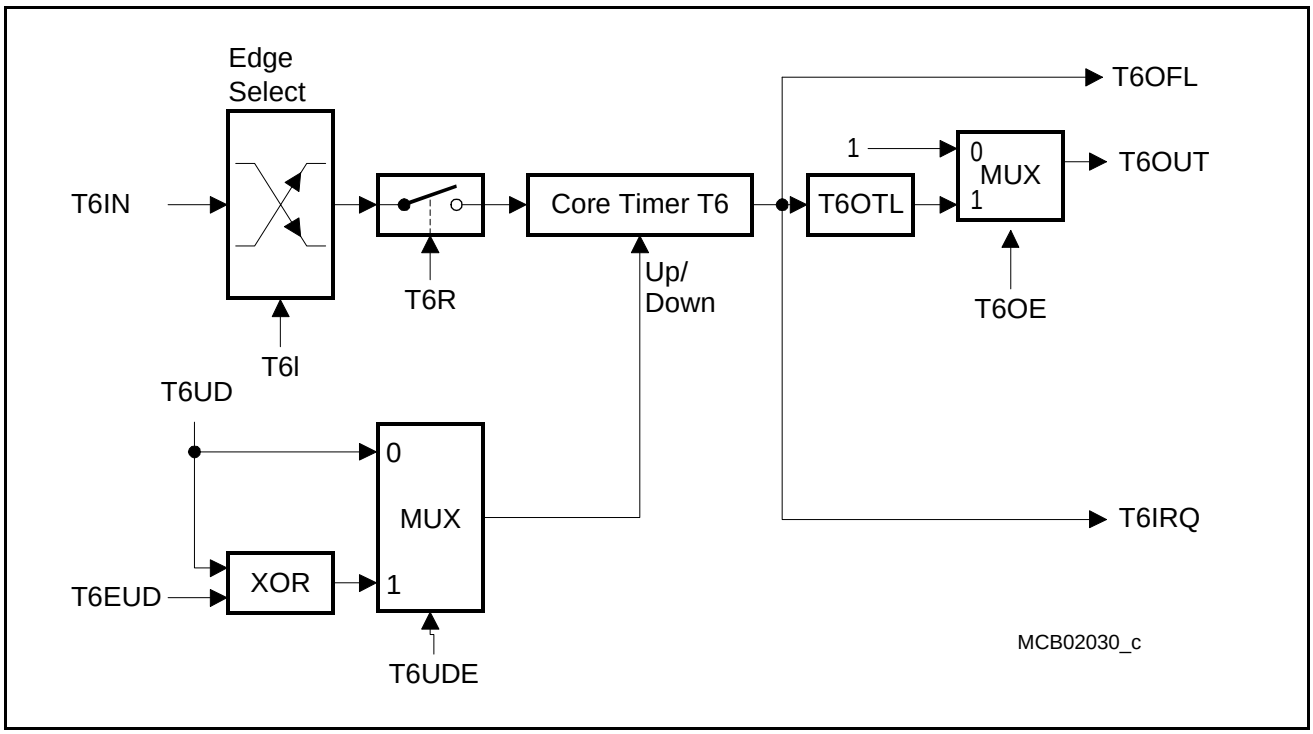


Figure 12-20 Block Diagram of Core Timer T6 in Counter Mode

Table 12-12 Core Timer T6 (Counter Mode) Input Edge Selection

T6I	Triggering Edge for Counter Increment/Decrement
0 0 0	None. Counter T6 is disabled
0 0 1	Positive transition (rising edge) on T6IN
0 1 0	Negative transition (falling edge) on T6IN
0 1 1	Any transition (rising or falling edge) on T6IN
1 X X	Reserved. Do not use this combination

The maximum input frequency allowed in Counter Mode is $f_{clk}/4$ (BPS2 = '01'). To ensure that a transition of the count input signal applied to T6IN is correctly recognized, its level should be held high or low for at least 2 f_{clk} cycles (BPS2 = '01') before it changes.

12.3.2 Auxiliary Timer T5

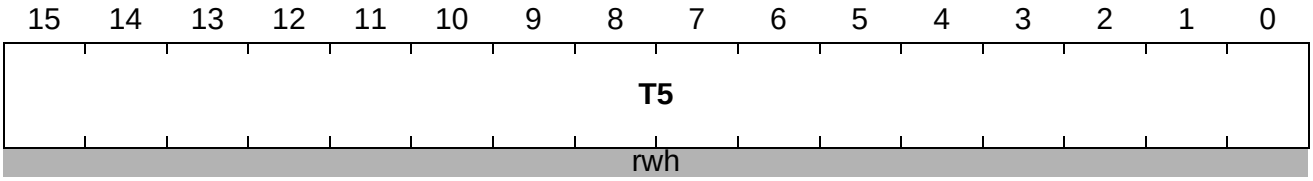
The auxiliary Timer T5 can be configured for Timer Mode, Gated Timer Mode, or Counter Mode with the same options for the timer frequencies and the count signal as core Timer T6. In addition to these three counting modes, the auxiliary timer can be concatenated with the core timer.

The individual configuration for Timer T5 is determined by its bitaddressable control register T5CON. Note that functions present in both timers of Timer Block 2 are controlled in the same bit positions and in the same manner in each of the specific control registers.

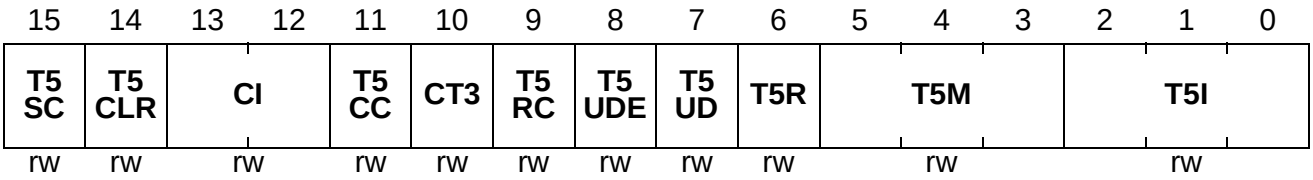
Run control for auxiliary Timer T5 can be handled by the associated Run Control Bit T5R in register T5CON. Alternatively, a remote control option (T5RC is set) may be enabled to start and stop T5 via the run bit T6R of core Timer T6.

Note: The auxiliary timer has no bit T5OTL. Therefore, an output line for overflow/ underflow monitoring is not provided.

T5
Timer 5 **(Reset value: 0000_H)**



T5CON
Timer 5 Control Register **(Reset value: 0000_H)**



Field	Bits	Typ	Description
T5	[15:0]	rwh	**Timer 5** Contains the current value of Timer 5.

Field	Bits	Typ	Description
T5I	[2:0]	rw	**Timer 5 Input Parameter Selection** Timer Mode: see **Table 12-13** for encoding Gated Timer Mode: see **Table 12-13** for encoding Counter Mode: see **Table 12-14** for encoding
T5M	[5:3]	rw	**Timer 5 Mode Control** (Basic Operating Mode) 000 Timer Mode 001 Counter Mode 010 Gated Timer Mode with gate active low 011 Gated Timer Mode with gate active high 1XX ***Reserved***. Do not use this combination!
T5R	[6]	rw	**Timer 5 Run Bit** 0 Timer/Counter 5 stops 1 Timer/Counter 5 runs
T5UD	[7]	rw	**Timer 5 Up/Down Control** (when T5UDE = '0') 0 Counts Up 1 Counts Down
T5UDE	[8]	rw	**Timer 5 External Up/Down Enable** 0 Counting direction is internally controlled by software 1 Counting direction is externally controlled by line T5EUD
T5RC	[9]	rw	**Timer 5 Remote Control** 0 Timer/counter 5 is controlled by its own run bit T5R 1 Timer/counter 5 is controlled by the run bit of core Timer 6
CT3	[10]	rw	**Timer 3 Capture Trigger Enable** 0 Capture trigger from input line CAPIN 1 Capture trigger from T3 input lines T3IN and/ or T3EUD
T5CC	[11]	rw	**Timer 5 Capture Correction** 0 T5 is just captured, without any correction 1 T5 is decremented by 1 before being captured

Field	Bits	Typ	Description
CI	[13:12]	rw	**Register CAPREL Capture Trigger Selection** (depending on bit CT3) 00 Capture disabled 01 Positive transition (rising edge) on CAPIN or any transition on T3IN 10 Negative transition (falling edge) on CAPIN or any transition on T3EUD 11 Any transition (rising or falling edge) on CAPIN or any transition on T3IN or T3EUD
T5CLR	[14]	rw	**Timer 5 Clear Bit** 0 Timer 5 is not cleared on a capture event 1 Timer 5 is cleared on a capture event
T5SC	[15]	rw	**Timer 5 Capture Mode Enable** 0 Capture into register CAPREL disabled 1 Capture into register CAPREL enabled

Count Direction Control for Auxiliary Timer

The count direction of the auxiliary timer can be controlled in the same way as for core Timer T6. The description and the table apply accordingly.

Timer T5 in Timer Mode or Gated Timer Mode

When auxiliary Timer T5 is programmed to Timer or Gated Timer Mode, its operation is the same as described for core Timer T6. The descriptions, figures, and tables apply accordingly with three exceptions:

- There is no T5OUT line for T5
- There is no T5OFL line for T5
- Overflow/underflow monitoring is not supported (no bit T5OTL).

Table 12-13 Timer 5 Input Parameter Selection: Timer and Gated Timer Modes

T5I	Prescaler for f_{clk} (BPS2 = 00)	Prescaler for f_{clk} (BPS2 = 01)	Prescaler for f_{clk} (BPS2 = 10)	Prescaler for f_{clk} (BPS2 = 11)
000	4	2	16	8
001	8	4	32	16
010	16	8	64	32
011	32	16	128	64
100	64	32	256	128
101	128	64	512	256
110	256	128	1024	512
111	512	256	2048	1024

Timer T5 in Counter Mode

Counter Mode for auxiliary Timer T5 is selected by setting bitfield T5M in register T5CON to '001_B'. In Counter Mode, Timer T5 can be clocked either by a transition at the external input line T5IN or by a transition of the output toggle latch T6OTL on Timer 6.

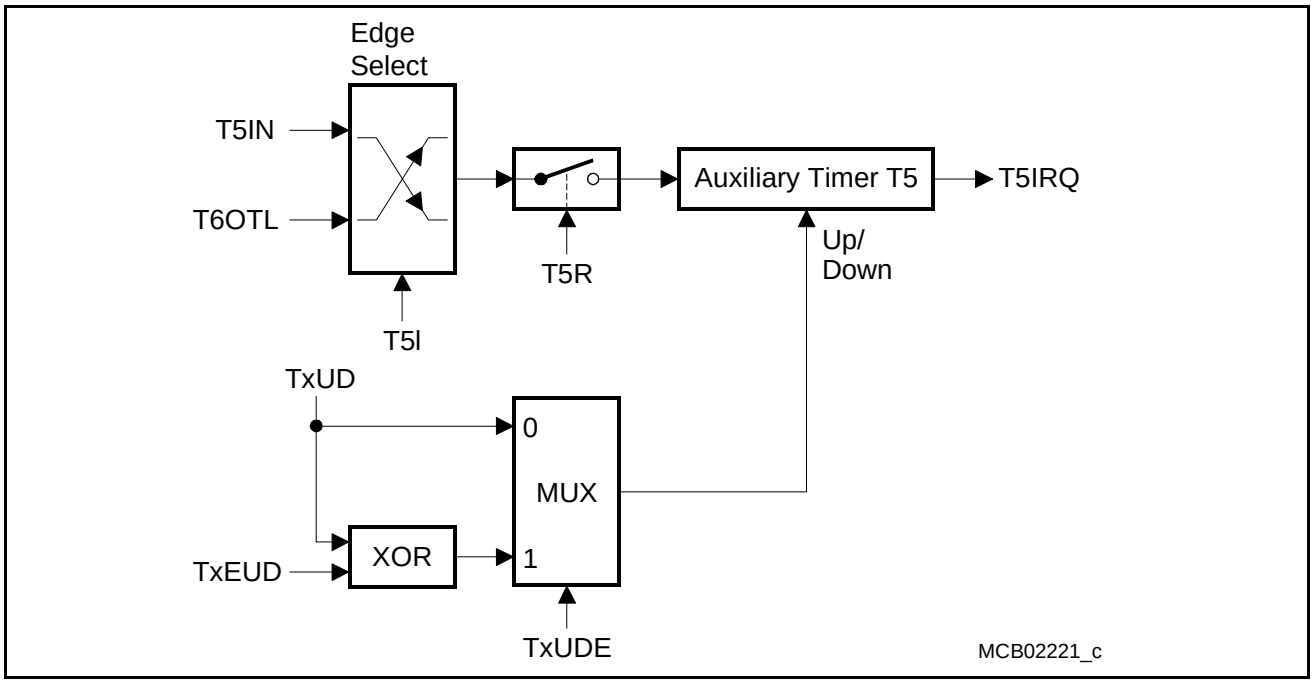


Figure 12-21 Block Diagram of Auxiliary Timer T5 in Counter Mode

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 either the input line T5IN or at the toggle latch T6OTL.

Bitfield T5P in control register T5CON selects the triggering transition (see **Table 12-14**).

Table 12-14 Auxiliary Timer (Counter Mode) Input Edge Selection

T5P	Triggering Edge for Counter Increment/Decrement
X 0 0	None. Counter T5 is disabled
0 0 1	Positive transition (rising edge) on T5IN
0 1 0	Negative transition (falling edge) on T5IN
0 1 1	Any transition (rising or falling edge) on T5IN
1 0 1	Positive transition (rising edge) on T6OTL
1 1 0	Negative transition (falling edge) on T6OTL
1 1 1	Any transition (rising or falling edge) on T6OTL

Note: Only state transitions of T6OTL caused by the overflow/underflow of T6 will trigger the counter function of T5. Modifications of T6OTL via software will NOT trigger the counter function of T5.

The maximum input frequency allowed in Counter Mode is $f_{clk}/4$ (BPS2 = '01'). To ensure that a transition of the count input signal applied to T5IN is correctly recognized, its level should be held high or low for at least 2 f_{clk} cycles (BPS2 = '01') before it changes.

12.3.3 Timer Concatenation

Using the toggle bit T6OTL as a clock source for the auxiliary Timer of Block 2 in Counter Mode concatenates core Timer T6 with auxiliary Timer T5. Depending on which transition of T6OTL is selected to clock auxiliary Timer T5, this concatenation forms a 32-bit or a 33-bit timer/counter.

- **32-bit Timer/Counter:** If both a positive and a negative transition of T6OTL is used to clock auxiliary Timer T5, this timer is clocked on every overflow/underflow of core Timer T6. Thus, the two timers form a 32-bit timer.
- **33-bit Timer/Counter:** If either a positive or a negative transition of T6OTL is selected to clock auxiliary Timer T5, this timer is clocked on every second overflow/underflow of core Timer T6. This configuration forms a 33-bit timer (16-bit core timer+T6OTL+16-bit auxiliary timer).The count directions of the two concatenated timers are not required to be the same. This offers a wide variety configurations.T6 can operate in Timer Mode, Gated Timer Mode or Counter Mode in this case.

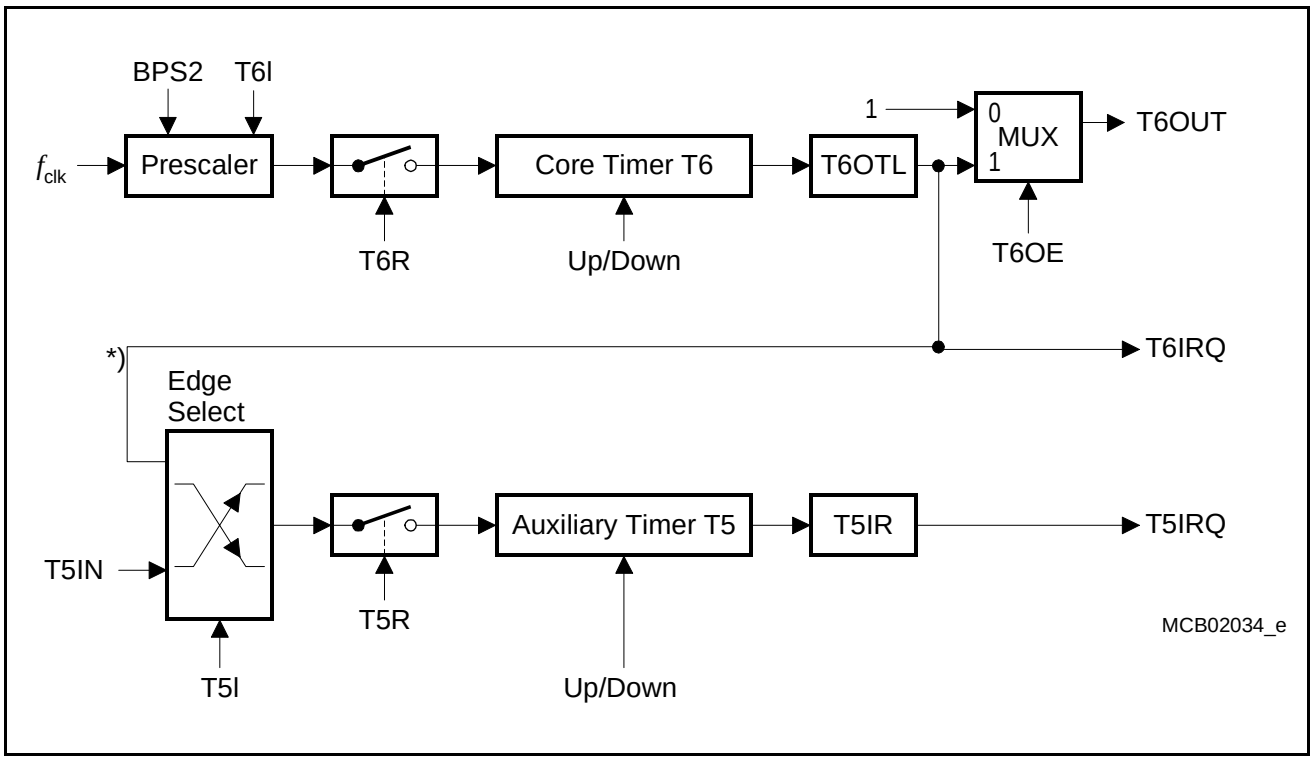
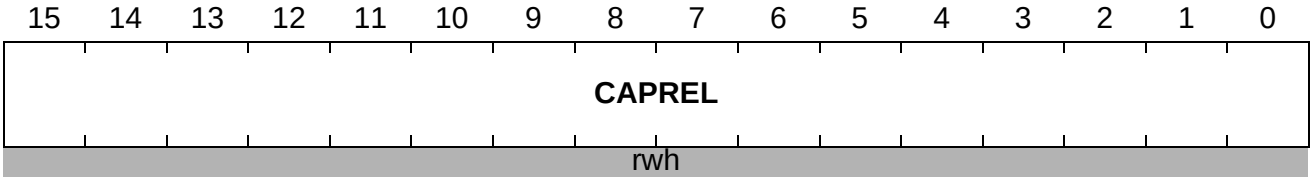


Figure 12-22 Concatenation of Core Timer T6 and Auxiliary Timer T5

Note: Line ')' is affected by over/underflow of T6 only, NOT by software modifications of T6OTL*

Capture/Reload Register CAPREL in Capture Mode

CAPREL
Capture/Reload Register **(Reset value: 0000_H)**



Field	Bits	Typ	Description
CAPREL	[15:0]	rwh	**Capture Reload Register Value** Contains the current value of the CAPREL register.

This 16-bit register can be used as a capture register for auxiliary Timer T5. This mode is selected by setting bit T5SC in control register T5CON. Bit CT3 selects the external input line (CAPIN) or the input lines (T3IN and/or T3EUD) of Timer T3 as the source for a capture trigger. Either a positive, a negative, or both a positive and a negative transition at line CAPIN can be selected to trigger the capture function, or transitions on input T3IN

or input T3EUD or both inputs T3IN and T3EUD. The active edge is controlled by bitfield CI in register T5CON.

The maximum input frequency for the capture trigger signal at CAPIN is $f_{clk}/2$ (BPS2 = '01'). To ensure that a transition of the capture trigger signal is correctly recognized, its level should be held for at least one f_{clk} cycle (BPS2 = '01') before it changes.

When Timer T3 capture trigger is enabled (CT3 is set) register CAPREL captures the contents of T5 upon transitions of the selected input(s). These values can be used to measure input signals of T3. This is useful, for example, when T3 operates in Incremental Interface Mode, to derive dynamic information (speed or acceleration) from the input signals.

When a selected transition at the external input line CAPIN is detected, the contents of auxiliary Timer T5 are latched into register CAPREL, and interrupt request line CRIRQ is driven at high level. With the same event, Timer T5 can be cleared to 0000_H. This option is controlled by bit T5CLR in register T5CON. If T5CLR is cleared, the contents of Timer T5 are not affected by a capture. If T5CLR is set, Timer T5 is cleared after the current timer value has been latched into register CAPREL.

Note: Bit T5SC only controls whether a capture is performed or not. If T5SC is cleared, the input line CAPIN *can still be used to clear Timer T5 or as an external interrupt input. This interrupt is controlled by the CAPREL interrupt control register CRIC.*

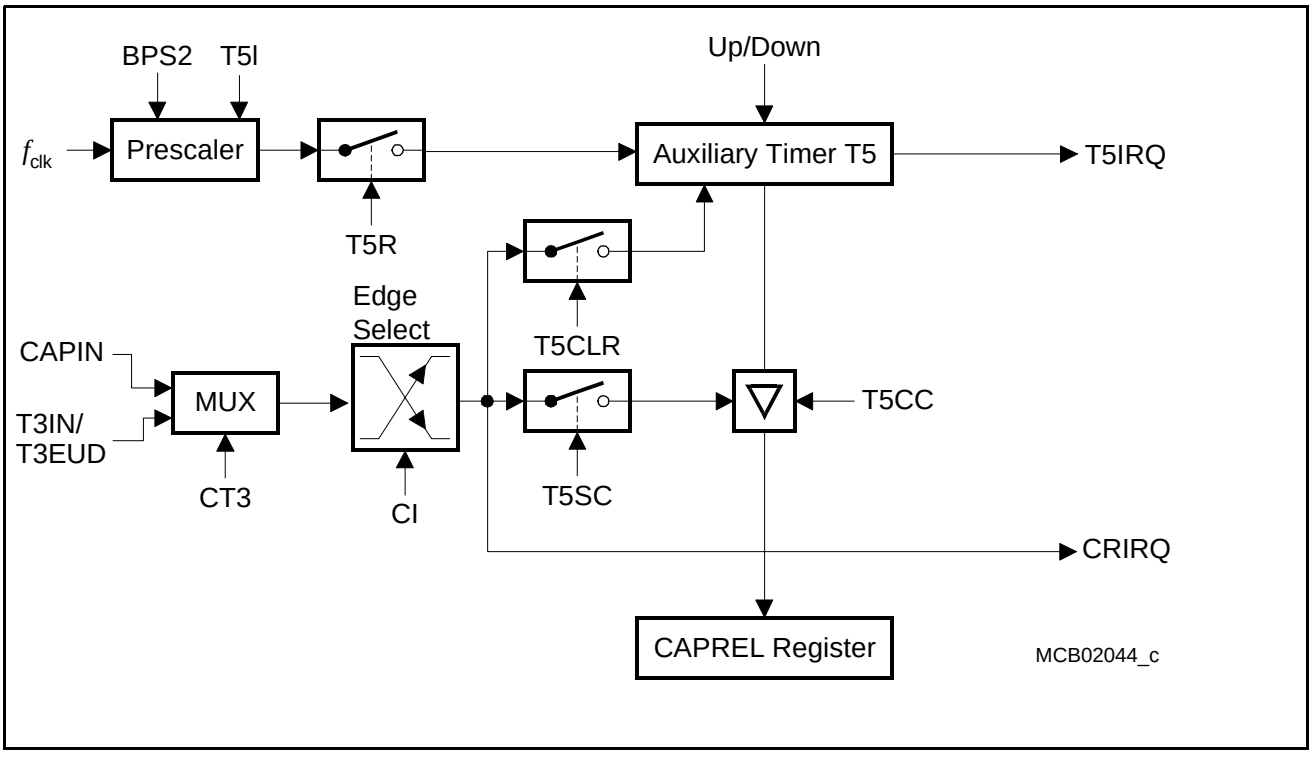


Figure 12-23 Timer Block 2 Register CAPREL in Capture Mode

Timer Block 2 Capture/Reload Register CAPREL in Reload Mode

This 16-bit register can be used as a reload register for core Timer T6. This mode is selected by setting bit T6SR in register T6CON. The event causing a reload in this mode is an overflow or underflow of core Timer T6.

When Timer T6 overflows from $FFFF_H$ to 0000_H (when counting up) or underflows from 0000_H to $FFFF_H$ (when counting down), the value stored in register CAPREL is loaded into Timer T6. This will not drive the interrupt request line CRIRQ associated with the CAPREL register. However, interrupt request line T6IRQ will be driven at high level to indicate the overflow/underflow of T6.

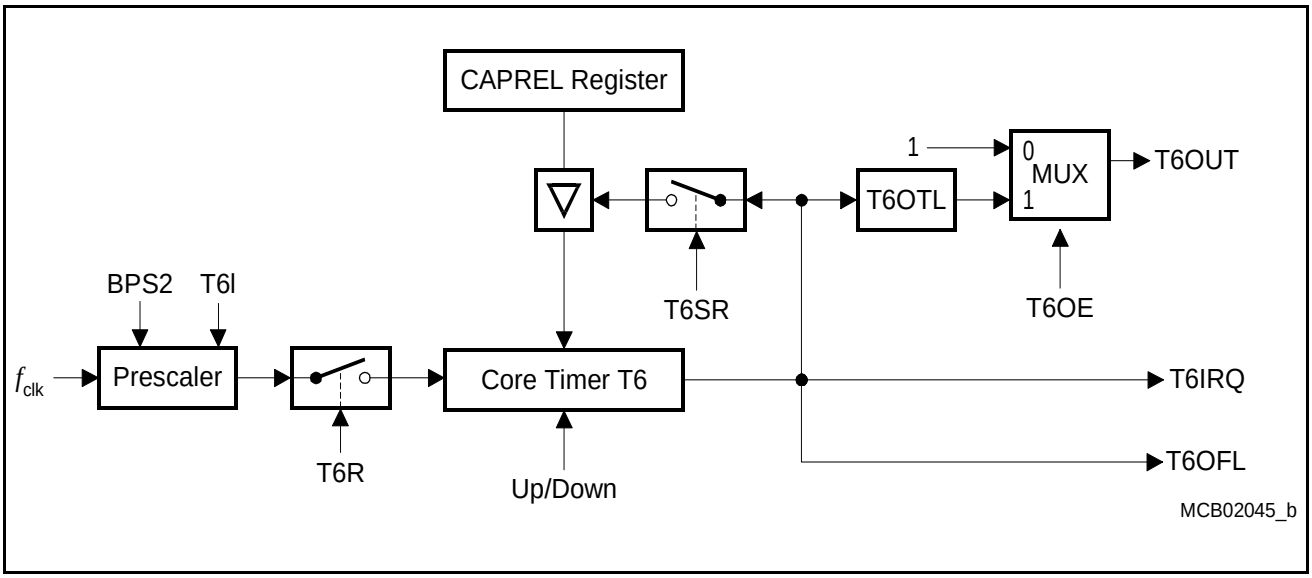


Figure 12-24 Timer Block 2 Register CAPREL in Reload Mode

Timer Block 2 Capture/Reload Register CAPREL in Capture-And-Reload Mode

Because the reload and capture functions of register CAPREL can be enabled individually by bits T5SC and T6SR, the two functions can be enabled simultaneously by setting both bits. This feature can be used to generate an output frequency that is a multiple of the input frequency.

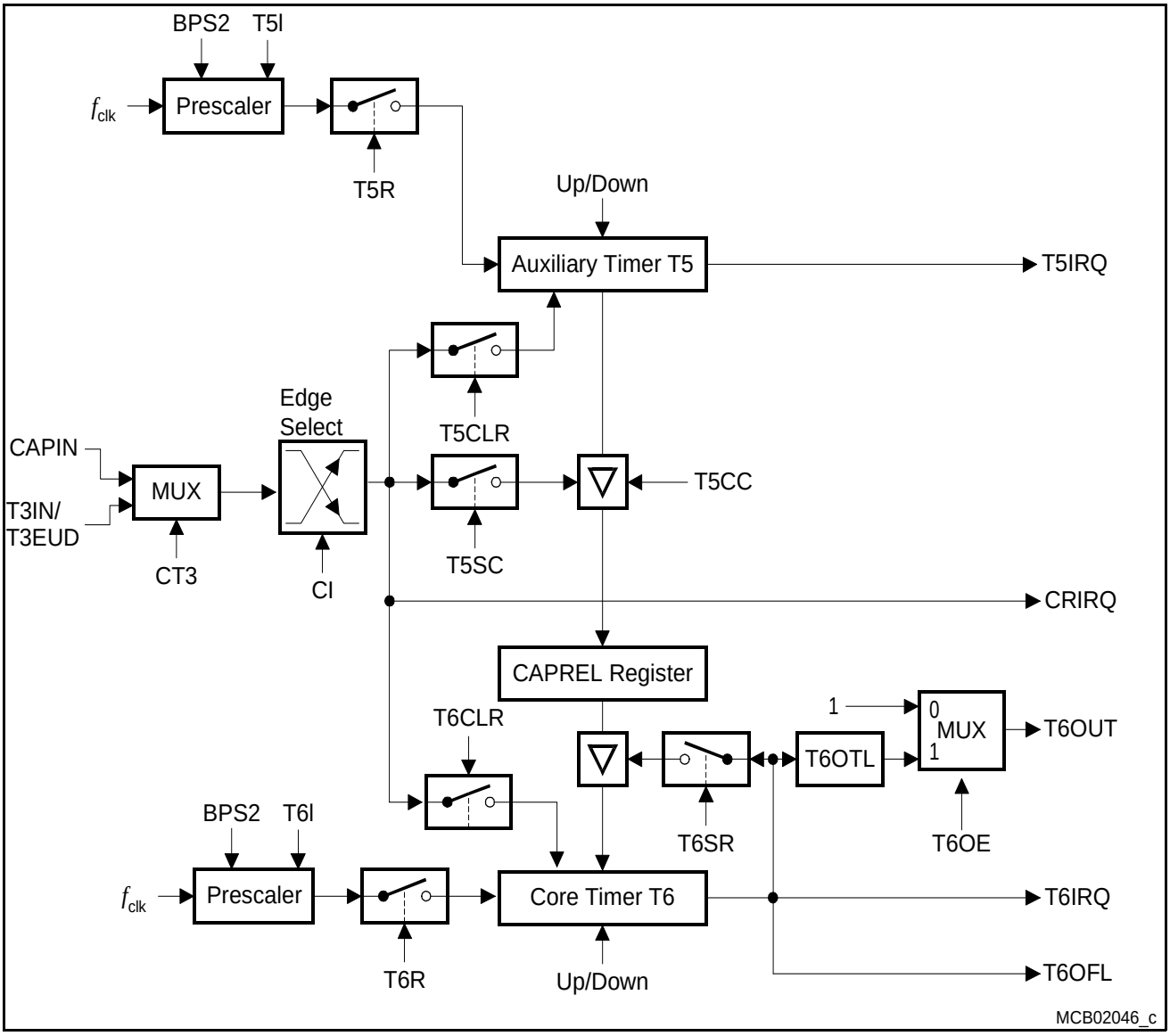


Figure 12-25 Timer Block 2 Register CAPREL in Capture-And-Reload Mode

This combined mode can be used to detect consecutive external events which may occur aperiodically, but which require a finer resolution (more 'ticks' within the time between two external events). For this purpose, the time between the external events is measured using Timer T5 and the CAPREL register. Timer T5 runs in Timer Mode counting up with a frequency of $f_{clk}/32$, for example. The external events are applied to line CAPIN. When an external event occurs, the contents of Timer T5 are latched into

register CAPREL, and Timer T5 is cleared (T5CLR cleared). Thus, register CAPREL always contains the correct time between two events, measured in Timer T5 increments. Timer T6, which runs in Timer Mode counting down with a frequency of $f_{clk}/4$, for example, uses the value in register CAPREL to perform a reload on underflow. This means that the value in register CAPREL represents the time between two underflows of Timer T6, now measured in Timer T6 increments. Because Timer T6 runs eight times faster than Timer T5, it will underflow eight times within the time between two external events. Thus, the underflow signal of Timer T6 generates eight 'ticks'. Upon each underflow, the interrupt request flag T6IR will be set and bit T6OTL will be toggled. The state of T6OTL may be output on line T6OUT. This signal has eight times more transitions than the signal applied to line CAPIN.

A certain deviation of the output frequency is generated by the fact that Timer T5 will count actual time units (for example: T5 running at 1 MHz will capture the value $64_H/100_D$ for a 10 KHz input signal) while T6OTL will only toggle upon an underflow of T6 (that is, the transition from 0000_H to $FFFF_H$). In the above mentioned example T6 would count down from 64_H so the underflow would occur after 101 T6 timing ticks. The actual output frequency then is 79.2 KHz instead of the expected 80 KHz.
This can be solved by activating the Capture Correction (T5CC is set). If capture correction is active, the content of T5 is decremented by 1 before being captured. The described deviation is eliminated (in the example, T5 would now capture $63_H/99_D$ and the output frequency would be 80 KHz).

Note: The underflow signal of Timer T6 can furthermore be used to clock one ore more timers of other timer units. This makes it possible to set compare events based on a finer resolution than that of the external events. This connection is accomplished via signal T6OFL.

13 Instruction Index

This section lists alphabetically all C166S instructions together with references to respective pages holding the detailed descriptions. This helps to quickly find the explanation of any specific core instruction.

14 Keyword Index

This section lists a number of keywords which refer to specific details of the C166S V1 SubS R1 in terms of its architecture, its functional units or functions. This helps to quickly find the answer to specific questions about the C166S V1 SubS R1.

R

S

T

V

W

X

Z

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