SIEMENS

C161RI

16-Bit CMOS Single-Chip Microcontroller

User’s Manual Version 1.0 05.98

http://www.siemens.de/
semiconductor/

05.98

Published by Siemens AG,
Bereich Halbleiter, Marketing-
Kommunikation, Balanstraße 73,
81541 München

Attention please!

As far as patents or other rights of third parties are concerned, liability is only assumed for components, not for applications, processes and circuits implemented within components or assemblies.

The information describes the type of component and shall not be considered as assured characteristics.

Terms of delivery and rights to change design reserved.

For questions on technology, delivery and prices please contact the Semiconductor Group Offices in Germany or the Siemens Companies and Representatives worldwide (see address list).

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Siemens Office, Semiconductor Group.

Siemens AG is an approved CECC manufacturer.

Packing

Please use the recycling operators known to you. We can also help you – get in touch with your nearest sales office. By agreement we will take packing material back, if it is sorted. You must bear the costs of transport.

For packing material that is returned to us unsorted or which we are not obliged to accept, we shall have to invoice you for any costs incurred.

Components used in life-support devices or systems must be expressly authorized for such purpose!

Critical components[1] of the Semiconductor Group of Siemens AG, may only be used in life-support devices or systems[2] with the express written approval of the Semiconductor Group of Siemens AG.

1 A critical component is a component used in a life-support device or system whose failure can reasonably be expected to cause the failure of that life-support device or system, or to affect its safety or effectiveness of that device or system.

2 Life support devices or systems are intended (a) to be implanted in the human body, or (b) to support and/or maintain and sustain human life. If they fail, it is reasonable to assume that the health of the user may be endangered.

C161RI **Revision History:**	**Version 1.0 05.98**
Previous Releases:	-
Page	**Subjects**

We Listen to Your Comments

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

mcdocu.comments@siemens-scg.com

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.

With the increasing complexity of embedded control applications, a significant increase in CPU performance and peripheral functionality over conventional 8-bit controllers is required from microcontrollers for high-end embedded control systems. In order to achieve this high performance goal Siemens has decided to develop its family of 16-bit CMOS microcontrollers without the constraints of backward compatibility.

Of course the architecture of the 16-bit microcontroller family pursues successful hardware and software concepts, which have been established in Siemens' popular 8-bit controller families.

About this Manual

This manual describes the functionality of the 16-bit microcontroller C161RI of the Siemens C166-family.

The descriptions in this manual refer to the following derivatives:

- **C161RI-LM**
- **C161RI-LF**

This manual is valid for the mentioned derivatives. Of course it refers to all devices of the different available temperature ranges and packages.

For simplicity all these various versions are referred to by the term **C161RI** throughout this manual. The complete pro-electron conforming designations are listed in the respective data sheets.

1.1 The Members of the 16-bit Microcontroller Family

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

The core of the 16-bit family has been developed with a modular family concept in mind. All family members execute an efficient control-optimized instruction set (additional instructions for members of the second generation). This allows 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 IO pins.

The XBUS 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 and highly efficient management of various resources on the external bus.

Enhanced derivatives of this second generation provide additional features like additional internal high-speed RAM, an integrated CAN-Module, an on-chip PLL, 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 XBUS, defined for the Siemens 16-bit microcontrollers (second generation). This XBUS is an internal representation of the external bus interface that opens and simplifies the integration of peripherals by standardizing the required interface. One representative taking advantage of this technology is the integrated CAN module.

The C165-type devices are reduced versions of the C167 which provide a smaller package and reduced power consumption at the expense of the A/D converter, the CAPCOM units and the PWM module.

The C164-type devices and some of the C161-type devices are further enhanced by a flexible power management and form the **third generation** of the 16-bit controller family. This power management mechanism provides effective means to control the power that is consumed in a certain state of the controller and thus allows the minimization of the overall power consumption with respect to a given application.

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

- mask-programmable ROM
- Flash memory
- OTP memory
- ROMless with no non-volatile memory at all.

Also there are devices with specific functional units.

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

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

Note: Not all derivatives will be offered in any temperature range, speed class, package or program memory variation.

Information about specific versions and derivatives will be made available with the devices themselves. Contact your Siemens representative for up-to-date material.

Note: As the architecture and the basic features (i.e. CPU core and built in peripherals) are identical for most of the currently offered versions of the C161RI, the descriptions within this manual that refer to the “C161RI” also apply to the other variations, unless otherwise noted.

1.2 Summary of Basic Features

The C161RI is an improved representative of the Siemens family of full featured 16-bit single-chip CMOS microcontrollers. It combines high CPU performance (up to 10 million instructions per second) with high peripheral functionality and means for power reduction.
Several key features contribute to the high performance of the C161RI (the indicated timings refer to a CPU clock of 20 MHz).

High Performance 16-Bit CPU With Four-Stage Pipeline

- 100 ns minimum instruction cycle time, with most instructions executed in 1 cycle
- 500 ns multiplication (16-bit × 16-bit), 1 µs 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

Integrated On-chip Memory

- 1 KByte internal RAM for variables, register banks, system stack and code
- 2 KByte on-chip high-speed XRAM for variables, user stack and code

External Bus Interface

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

16-Priority-Level Interrupt System

- 27 interrupt nodes with separate interrupt vectors
- 300/500 ns typical/maximum interrupt latency in case of internal program execution
- Fast external interrupts

8-Channel Peripheral Event Controller (PEC)

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

Intelligent On-chip Peripheral Subsystems

- 4-Channel 8-bit A/D Converter with programmable conversion time (7.625 µs minimum), auto scan modes, channel injection mode
- 2 Multifunctional General Purpose Timer Units
  GPT1: three 16-bit timers/ counters, maximum resolution $f_{CPU}/8$
  GPT2: two 16-bit timers/counters, maximum resolution $f_{CPU}/4$
- Asynchronous/Synchronous Serial Channel (USART)
  with baud rate generator, parity, framing, and overrun error detection
- High Speed Synchronous Serial Channel
  programmable data length and shift direction
- I^2C-Bus Module with 10-bit addressing and 400 Kbit/sec
- Real Time Clock
- Watchdog Timer with programmable time intervals
- Bootstrap Loader for flexible system initialization

76 IO Lines With Individual Bit Addressability

- Tri-stated in input mode
- Push/pull or open drain output mode

Different Temperature Ranges

- 0 to + 70 °C, – 40 to + 85 °C

Siemens CMOS Process

- Low Power CMOS Technology, including power saving Idle and Power Down modes

100-Pin Plastic Quad Flat Pack (PQFP) Packages

- P-MQFP, 14 × 20 mm body, 0.65 mm (25.6 mil) lead spacing, surface mount technology
- P-TQFP, 14 × 14 mm body, 0.5 mm (19.7 mil) lead spacing, surface mount technology

Complete Development Support

For the development tool support of its microcontrollers, Siemens 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 Siemens C500 and C166 microcontroller families, guaranteeing a remarkable variety of price-performance classes as well as early availability of high quality key tools such as compilers, assemblers, simulators, debuggers or in-circuit emulators.

Siemens 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 Siemens microcontrollers in the most effective way and with the shortest possible learning curve.

The tool environment for the Siemens 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
- VHDL chip models
- In-Circuit Emulators (based on bondout or standard chips)
- Plug-In emulators
- Emulation and Clip-Over adapters, production sockets
- Logic Analyzer disassemblers
- Starter Kits
- Evaluation Boards with monitor programs
- Industrial boards (also for CAN, FUZZY, PROFIBUS, FORTH applications)
- Network driver software (CAN, PROFIBUS)

1.3 Abbreviations

The following acronyms and termini are used within this document:

ADC	Analog Digital Converter
ALE	Address Latch Enable
ALU	Arithmetic and Logic Unit
ASC	Asynchronous/synchronous Serial Controller
CISC	Complex Instruction Set Computing
CMOS	Complementary Metal Oxide Silicon
CPU	Central Processing Unit
EBC	External Bus Controller
ESFR	Extended Special Function Register
Flash	Non-volatile memory that may be electrically erased
GPR	General Purpose Register
GPT	General Purpose Timer unit
HLL	High Level Language
I^{2}C	Inter Integrated Circuit (Bus)
IO	Input / Output
OTP	One Time Programmable memory
PEC	Peripheral Event Controller
PLA	Programmable Logic Array
PLL	Phase Locked Loop
PWM	Pulse Width Modulation
RAM	Random Access Memory
RISC	Reduced Instruction Set Computing
ROM	Read Only Memory
SDD	Slow Down Divider
SFR	Special Function Register
SSC	Synchronous Serial Controller
XBUS	Internal representation of the External Bus

2 Architectural Overview

The architecture of the C161RI combines the advantages of both RISC and CISC processors in a very well-balanced way. The sum of the features which are combined result in a high performance microcontroller, which is the right choice not only for today's applications, but also for future engineering challenges. The C161RI not only integrates a powerful CPU 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 C161RI is the XBUS, an internal representation of the external bus interface. This bus provides a standardized method of integrating application-specific peripherals to produce derivates of the standard C161RI.

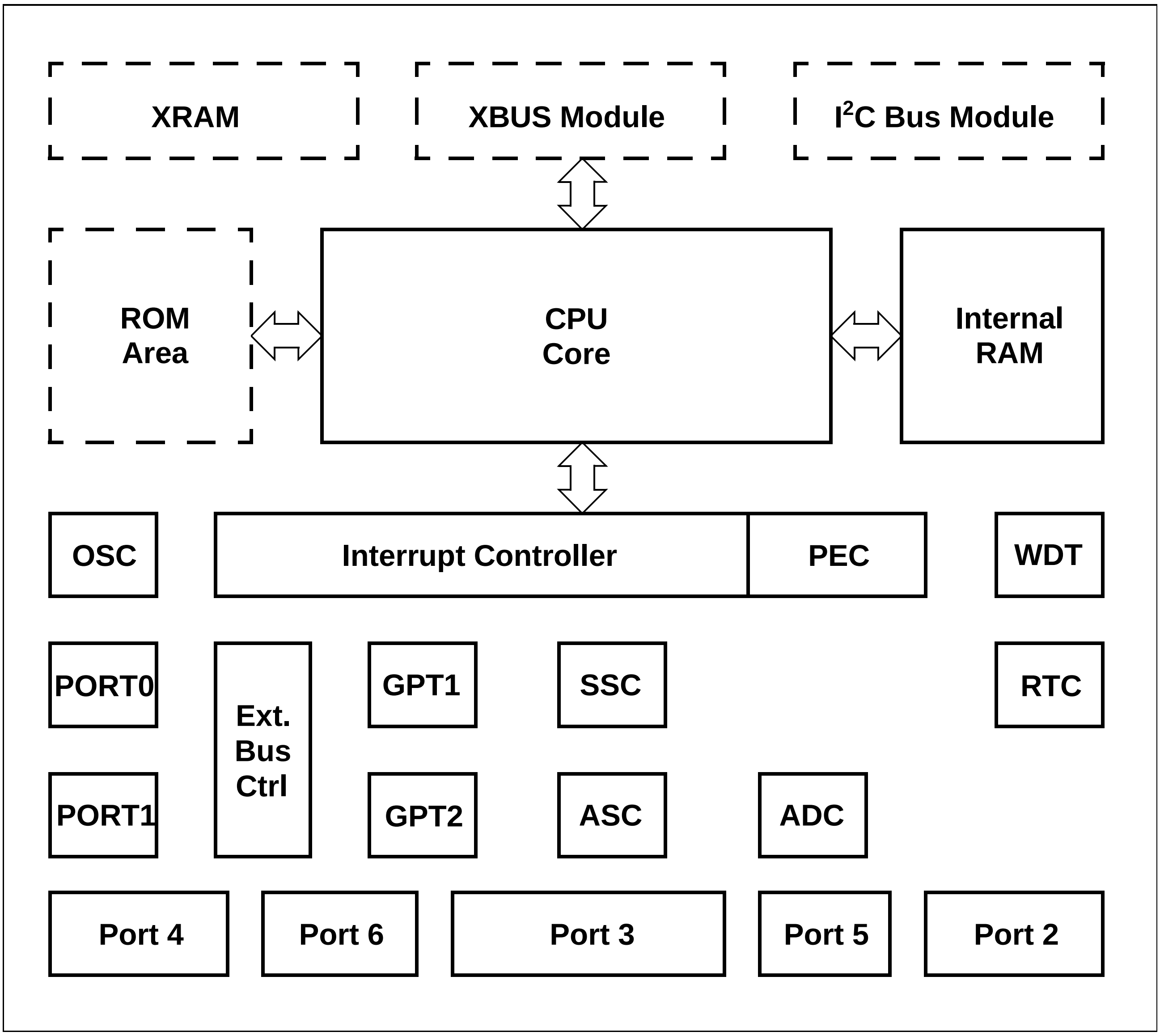


Figure 2-1
C161RI Functional Block Diagram

2.1 Basic CPU Concepts and Optimizations

The main core of the CPU consists of a 4-stage instruction pipeline, a 16-bit arithmetic and logic unit (ALU) and dedicated SFRs. Additional hardware is provided for a separate multiply and divide unit, a bit-mask generator and a barrel shifter.

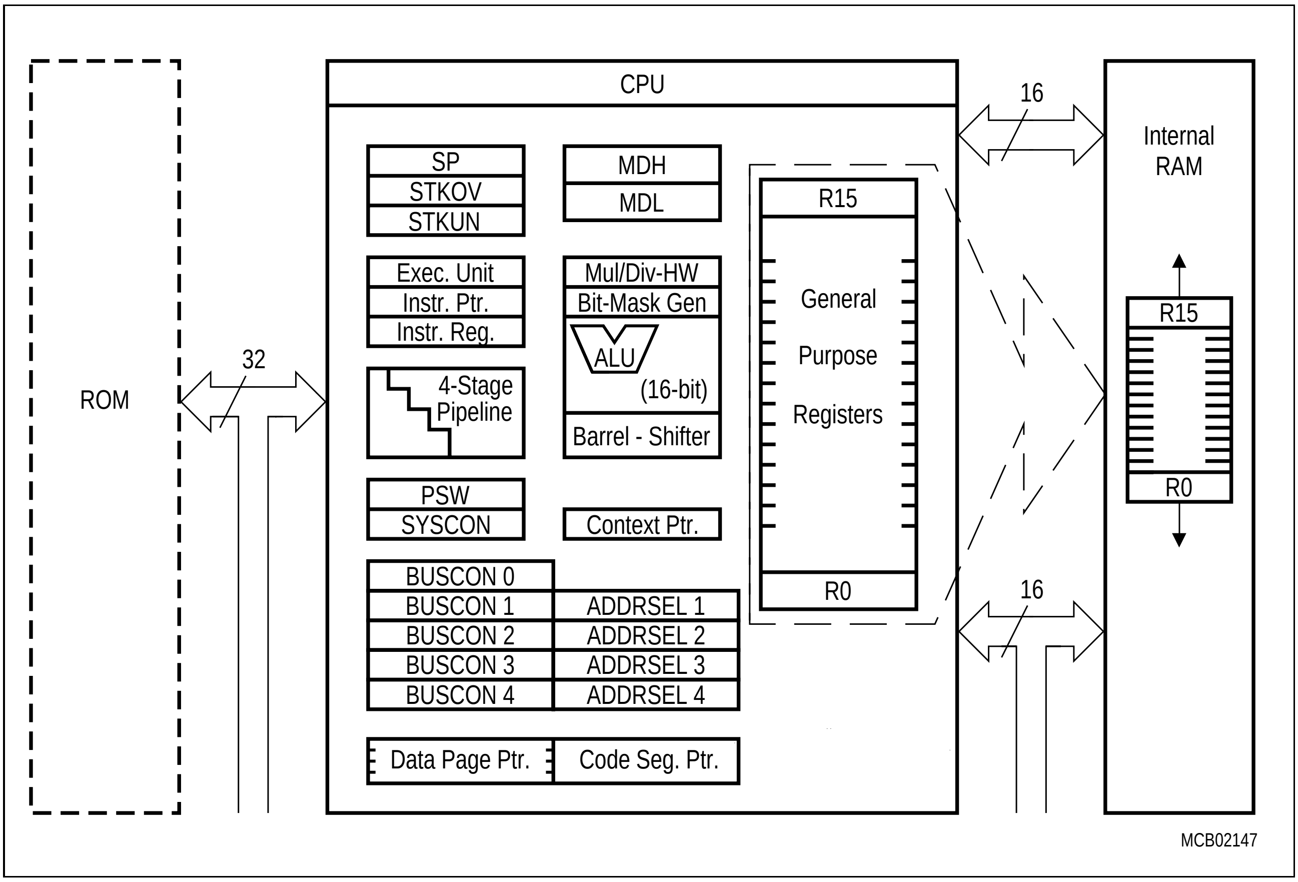


Figure 2-2
CPU Block Diagram

To meet the demand for greater performance and flexibility, a number of 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

High Instruction Bandwidth / Fast Execution

Based on the hardware provisions, most of the C161RI's instructions can be executed in just one machine cycle, which requires 2 CPU clock cycles ($2 \times 1/f_{CPU}$ = 4 TCL). For example, shift and rotate instructions are always processed within one machine cycle, independent of the number of bits to be shifted.

Branch-, multiply- and divide instructions normally take more than one machine cycle. These instructions, however, have also been optimized. For example, branch instructions only require an additional machine cycle, when a branch is taken, and most branches taken in loops require no additional machine cycles at all, due to the so-called 'Jump Cache'.
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 or RAM or from the external memory, based on the current IP value.

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

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

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

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

Instruction Decoder

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

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

All standard arithmetic and logical operations are performed in a 16-bit ALU. 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 and MDH, and require four and nine machine cycles, respectively, to perform a 16-bit by 16-bit (or 32-bit by 16-bit) calculation plus one machine cycle to setup and adjust the operands and the result. Even these longer multiply and divide instructions can be interrupted during their execution to allow for very fast interrupt response. Instructions have 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 after each arithmetic, logical, shift, or movement operation. These flags allow branching on specific conditions. Support for both signed and unsigned arithmetic is provided through user-specifiable branch tests. These flags are also preserved automatically by the CPU upon entry into an interrupt or trap routine.
All targets for branch calculations are also computed in the central ALU.

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

Extended Bit Processing and Peripheral Control

A large number of instructions has been dedicated to bit processing. These instructions provide efficient control and testing of peripherals while enhancing data manipulation. Unlike other microcontrollers, these instructions provide direct access to two operands in the bit-addressable space without requiring 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 single cycle branch execution after the first iteration of a loop. Thus, only one machine cycle is lost during the execution of the entire loop. In loops which fall through upon completion, no machine cycles are lost when exiting the loop. No special instructions are required to perform loops, and loops are automatically detected during execution of branch instructions.
- 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.

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 C161RI 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.

Programmable Multiple Priority Interrupt 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 in segment 0 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: 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 RAM. A single one-machine-cycle instruction allows to switch register banks from one task to another.

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

With an interrupt response time within a range from just 5 to 10 CPU clock cycles (in case of internal program execution), the C161RI is capable of reacting very fast on non-deterministic events.

Its fast external interrupt inputs are sampled every CPU clock cycle and allow to recognize even very short external signals.

The C161RI 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 similiar 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.

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

2.2 The On-chip System Resources

The C161RI controllers provide a number of powerful system resources designed around the CPU. The combination of CPU and these resources results in the high performance of the members of this controller family.

Peripheral Event Controller (PEC) and Interrupt Control

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

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

An individual PEC transfer counter is implicitly decremented for each PEC service except forming in the continuous transfer mode. When this counter reaches zero, a standard interrupt is performed to the vector location related to the corresponding source. PEC services are very well suited, for example, to move register contents to/from a memory table. The C161RI has 8 PEC channels each of which offers such fast interrupt-driven data transfer capabilities.

Memory Areas

The memory space of the C161RI is configured in a Von Neumann architecture which means that code memory, data memory, registers and IO ports are organized within the same linear address space which covers up to 16 MBytes. The entire memory space can be accessed bytewise or wordwise. Particular portions of the on-chip memory have additionally been made directly bit addressable.

A 1 KByte 16-bit wide internal RAM provides fast access to General Purpose Registers (GPRs), user data (variables) and system stack. The internal RAM may also be used for code. A unique decoding scheme provides flexible user register banks in the internal memory while optimizing the remaining RAM for user data.

The CPU disposes of an actual register context consisting of up to 16 wordwide and/or bytewide GPRs, 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 internal RAM space. For easy parameter passing, a register bank may overlap others.

A system stack of up to 512 words 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, STKOV and STKUN, are implicitly compared against the stack pointer value upon each stack access for the detection of a stack overflow or underflow.

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

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

For Special Function Registers 1024 Bytes of the address space are reserved. The standard Special Function Register area (SFR) uses 512 bytes, while the Extended Special Function Register area (ESFR) uses the other 512 bytes. (E)SFRs are wordwide registers which are used for controlling and monitoring functions of the different on-chip units. Unused (E)SFR addresses are reserved for future members of the C166 family with enhanced functionality.

External Bus Interface

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

- 16-/18-/20-/23-bit Addresses, 16-bit Data, Demultiplexed
- 16-/18-/20-/23-bit Addresses, 8-bit Data, Demultiplexed
- 16-/18-/20-/23-bit Addresses, 16-bit Data, Multiplexed
- 16-/18-/20-/23-bit Addresses, 8-bit Data, Multiplexed

The demultiplexed bus modes use PORT1 for addresses and PORT0 for data input/output. The multiplexed bus modes use PORT0 for both addresses and data input/output. Port 4 is used for the upper address lines (A16 …) if selected.

Important timing characteristics of the external bus interface (waitstates, ALE length and Read/Write Delay) have been made programmable to allow the user the adaption of a wide range of different types of memories and/or peripherals. Access to very slow memories or peripherals is supported via a particular ’Ready’ function.

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

The on-chip XBUS is an internal representation of the external bus 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.

The on-chip XRAM and the on-chip I^{2}C-Module are examples for these X-Peripherals.

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. Each functional block processes data independently and communicates information over common buses. Peripherals are controlled by data written to the respective Special Function Registers (SFRs). These SFRs are located either within the standard SFR area (00’FE00$_H$ … 00’FFFF$_H$) or within the extended ESFR area (00’F000$_H$ … 00’F1FF$_H$).

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

The C161RI generic peripherals are:

- Two General Purpose Timer Blocks (GPT1 and GPT2)
- Two Serial Interfaces (ASC0 and SSC)
- A Watchdog Timer
- An 8-bit Analog / Digital Converter
- A Real Time Clock
- Seven IO ports with a total of 76 IO lines

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

Peripheral Interfaces

The on-chip peripherals generally have two different types of interfaces, an interface to the CPU and an interface to external hardware. Communication between CPU and peripherals is performed through Special Function Registers (SFRs) and interrupts. The SFRs serve as control/status and data registers for the peripherals. Interrupt requests are generated by the peripherals based on specific events which occur during their operation (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 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. Further details on peripheral timing are included in the specific sections about each peripheral.

Programming Hints

Access to SFRs

All SFRs reside in data page 3 of the memory space. The following addressing mechanisms allow to access the SFRs:

- indirect or direct addressing with **16-bit (mem) addresses** must guarantee that the used data page pointer (DPP0 … DPP3) selects data page 3.
- accesses via the Peripheral Event Controller (**PEC**) use the SRCPx and DSTPx pointers instead of the data page pointers.
- **short 8-bit (reg) addresses** to the standard SFR area do not use the data page pointers but directly access the registers within this 512 Byte area.
- **short 8-bit (reg) addresses** to the extended **ESFR** area require switching to the 512 Byte extended SFR area. This is done via the EXTension instructions EXTR, EXTP(R), EXTS(R).

Byte write operations to 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 C161RI's SFRs are marked as 'Reserved'. User software should never write '1's to reserved bits. These bits are currently not implemented and may be used in future products to invoke new functions. In this case, the active state for these functions will be '1', and the inactive state will be '0'. Therefore writing only ‘0’s to reserved locations provides portability of the current software to future devices. After read accesses reserved bits should be ignored or masked out.

Parallel Ports

The C161RI provides up to 76 IO lines which are organized into six input/output ports and one input port. All port lines are bit-addressable, and all input/output lines are individually (bit-wise) programmable as inputs or outputs via direction registers. The IO ports are true bidirectional ports which are switched to high impedance state when configured as inputs. The output drivers of three IO ports can be configured (pin by pin) for push/pull operation or open-drain operation via control registers. During the internal reset, all port pins are configured as inputs.

All port lines have programmable alternate input or output functions associated with them. PORT0 and PORT1 may be used as address and data lines when accessing external memory, while Port 4 outputs the additional segment address bits A22/19/17 … A16 in systems where segmentation is used to access more than 64 KBytes of memory. Port 6 provides I^{2}C Bus lines and the chip select signals $\overline{\text{CS4}}$ … $\overline{\text{CS0}}$. Port 2 accepts the fast external interrupt inputs. Port 3 includes alternate functions of timers, serial interfaces, the optional bus control signal $\overline{\text{BHE}}$ and the system clock output (CLKOUT). Port 5 is used for timer control signals and for the analog inputs to the A/D Converter. All port lines that are not used for these alternate functions may be used as general purpose IO lines.

Serial Channels

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

The ASC0 is upward compatible with the serial ports of the Siemens 8-bit microcontroller families and supports full-duplex asynchronous communication at up to 500 KBaud and half-duplex synchronous communication at up to 2 MBaud @ 16 MHz CPU clock.
A dedicated baud rate generator allows to set up all standard baud rates without oscillator tuning. For transmission, reception and error handling 4 separate interrupt vectors 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 plus wake up bit mode).
In synchronous mode, the ASC0 transmits or receives bytes (8 bits) synchronously to a shift clock which is generated by the ASC0. The ASC0 always shifts the LSB first. 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.

The SSC supports full-duplex synchronous communication at up to 4 Mbaud @ 16 MHz CPU clock. It may be configured so it interfaces with serially linked peripheral components. A dedicated baud rate generator allows to set up all standard baud rates without oscillator tuning. For transmission, reception and error handling 3 separate interrupt vectors are provided.
The SSC transmits or receives characters of 2 … 16 bits length synchronously to a shift clock which can be generated by the SSC (master mode) or by an external master (slave mode). The SSC can start shifting with the LSB or with the MSB and allows the selection of shifting and latching clock edges as well as the clock polarity.
A number of optional hardware error detection capabilities has been included to increase the reliability of data transfers. Transmit and receive error supervise the correct handling of the data buffer. Phase and baudrate error detect incorrect serial data.

The On-chip I^{2}C Bus Module

The integrated I^{2}C Module handles the transmission and reception of frames over the two-line I^{2}C bus in accordance with the I^{2}C Bus specification. The on-chip I^{2}C Module can receive and transmit data using 7-bit or 10-bit addressing and it can operate in slave mode, in master mode or in multi-master mode.

Several physical interfaces (port pins) can be established under software control. Data can be transferred at speeds up to 400 Kbit/sec.

Two interrupt nodes dedicated to the I^{2}C module allow efficient interrupt service and also support operation via PEC transfers.

Note: The port pins associated with the I^{2}C interfaces feature open drain drivers only, as required by the I^{2}C specification.

General Purpose Timer (GPT) Unit

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

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

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

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

The core timers T3 and T6 have output toggle latches (TxOTL) which change their state on each timer over-flow/underflow. The state of these latches may be used internally to concatenate the core timers with the respective auxiliary timers resulting in 32/33-bit timers/counters for measuring long time periods with high resolution.

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

The maximum resolution of the timers in module GPT1 is 8 CPU clock cycles (= 16 TCL). With their maximum resolution of 4 CPU clock cycles (= 8 TCL) the GPT2 timers provide precise event control and time measurement.

Watchdog Timer

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

The Watchdog Timer is always enabled after a reset of the chip, and can only be disabled in the time interval until the EINIT (end of initialization) instruction has been executed. Thus, the chip's start-up procedure 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 an internal hardware reset and pulls the $\overline{\text{RSTOUT}}$ pin low in order to allow external hardware components to reset.

The Watchdog Timer is a 16-bit timer, clocked with the CPU clock divided either by 2 or by 128. The high byte of the Watchdog Timer register can be set to a prespecified 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 31 µs and 525 ms can be monitored (@ 16 MHz). The default Watchdog Timer interval after reset is 8.19 ms (@ 16 MHz).

A/D Converter

For analog signal measurement, an 8-bit A/D converter with 4 multiplexed input channels and a sample and hold circuit has been integrated on-chip. It uses the method of successive approximation. The sample time (for loading the capacitors) and the conversion time is programmable and can so be adjusted to the external circuitry.

Overrun error detection is provided for the conversion result register (ADDAT): an interrupt request will be generated when the result of a previous conversion has not been read from the result register at the time the next conversion is complete.

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

The A/D converter of the C161RI supports two different conversion modes. In the standard Single Channel conversion mode, the analog level on a specified channel is sampled once and converted to a digital result. In the Single Channel Continuous mode, the analog level on a specified channel is repeatedly sampled and converted without software intervention.

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

Real Time Clock

The C161RI contains a real time clock (RTC) which serves for different purposes:

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

The RTC module consists of a chain of 3 divider blocks, a fixed 8:1 divider, the reloadable 16-bit timer T14 and the 32-bit RTC timer (accessible via registers RTCH and RTCL). Both timers count up.

2.4 Power Management Features

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

- Flexible clock generation
- Flexible peripheral management (peripherals can be enabled/disabled separately or in groups)
- Periodic wakeup from Idle mode via RTC timer

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

Flexible Clock Generation

The flexible clock generation system combines a variety of improved mechanisms (partly user controllable) to provide the C161RI modules with clock signals. This is especially important in power sensitive modes like standby operation.

The power optimized oscillator generally reduces the amount of power which is consumed in order to generate the clock signal within the C161RI.

The clock system efficiently controls the amount of power which is consumed in order to distribute the clock signal within the C161RI.

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

Flexible Peripheral Management

The flexible peripheral management provides a mechanism to enable and disable each peripheral module separately. In each situation (e.g. several system operating modes, standby, etc.) only those peripherals may be kept running which are required for the respective functionality. All others can be switched off. It also allows the operation control of whole groups of peripherals including the power required for generating and distributing their clock input signal. Other peripherals may remain active, e.g. in order to maintain communication channels. The registers of separately disabled peripherals (not within a disabled group) can still be accessed.

Periodic wakeup from Idle Mode

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.

2.5 Protected Bits

The C161RI provides a special mechanism to protect bits which can be modified by the on-chip hardware from being changed unintentionally by software accesses to related bits (see also chapter “The Central Processing Unit”).

The following bits are protected:

Register	Bit Name	Notes
T2IC, T3IC, T4IC	**T2IR, T3IR, T4IR**	GPT1 timer interrupt request flags
T5IC, T6IC	**T5IR, T6IR**	GPT2 timer interrupt request flags
CRIC	**CRIR**	GPT2 CAPREL interrupt request flag
T3CON, T6CON	**T3OTL, T6OTL**	GPTx timer output toggle latches
S0TIC, S0TBIC	**S0TIR, S0TBIR**	ASC0 transmit(buffer) interrupt request flags
S0RIC, S0EIC	**S0RIR, S0EIR**	ASC0 receive/error interrupt request flags
S0CON	**S0REN**	ASC0 receiver enable flag
SSCTIC, SSCRIC	**SSCTIR, SSCRIR**	SSC transmit/receive interrupt request flags
SSCEIC	**SSCEIR**	SSC error interrupt request flag
SSCCON	**SSCBSY**	SSC busy flag
SSCCON	**SSCBE, SSCPE**	SSC error flags
SSCCON	**SSCRE, SSCTE**	SSC error flags
ADCIC, ADEIC	**ADCIR, ADEIR**	ADC end-of-conv./overrun intr. request flag
ADCON	**ADST**	ADC start flag request flag
CC15IC … CC8IC	**CC15IR … CC8IR**	Fast external interrupt request flags
TFR	**TFR.15,14,13**	Class A trap flags
TFR	**TFR.7,3,2,1,0**	Class B trap flags
XP3IC … XP0IC	**XP3IR … XP0IC**	X-Peripheral interrupt request flags

Σ = 44 protected bits.

3 Memory Organization

The memory space of the C161RI is configured in a “Von Neumann” architecture. This means that code and data are accessed within the same linear address space. All of the physically separated memory areas, including internal ROM/Flash/OTP (where integrated), internal RAM, the internal Special Function Register Areas (SFRs and ESFRs), the address areas for integrated XBUS peripherals and external memory are mapped into one common address space.

The C161RI provides a total addressable memory space of 16 MBytes. This address space is arranged as 256 segments of 64 KBytes each, and each segment is again subdivided into four data pages of 16 KBytes each (see figure below).

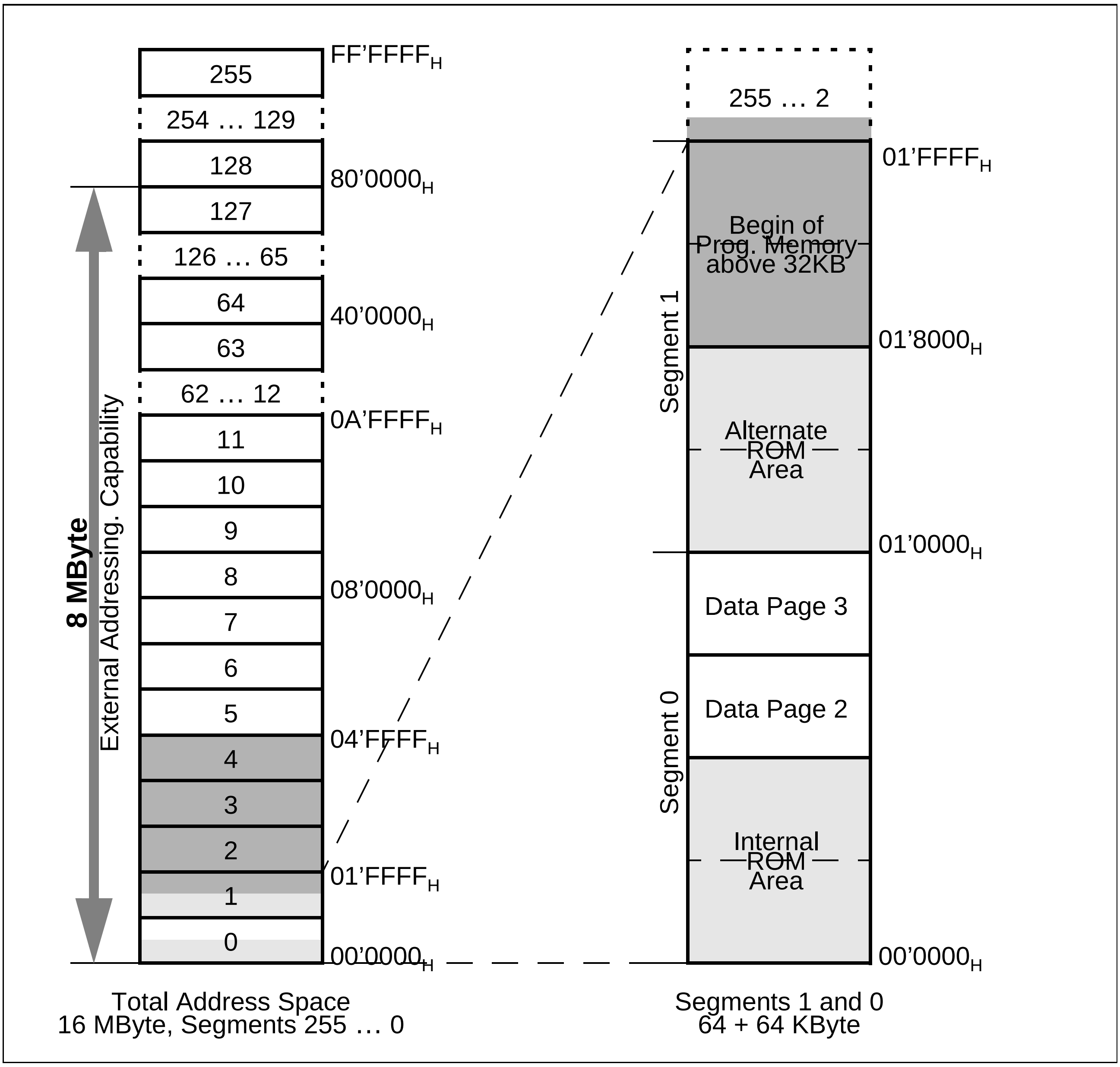


Figure 3-1
Address Space Overview

Most internal memory areas are mapped into segment 0, the system segment. The upper 4 KByte of segment 0 (00'F000$_H$ … 00'FFFF$_H$) hold the Internal RAM and Special Function Register Areas (SFR and ESFR). The lower 32 KByte of segment 0 (00'0000$_H$ … 00'7FFF$_H$) may be occupied by a part of the on-chip ROM/Flash/OTP memory and is called the Internal ROM area. This ROM area can be remapped to segment 1 (01'0000$_H$ … 01'7FFF$_H$), to enable external memory access in the lower half of segment 0, or the internal ROM may be disabled at all.

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

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

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

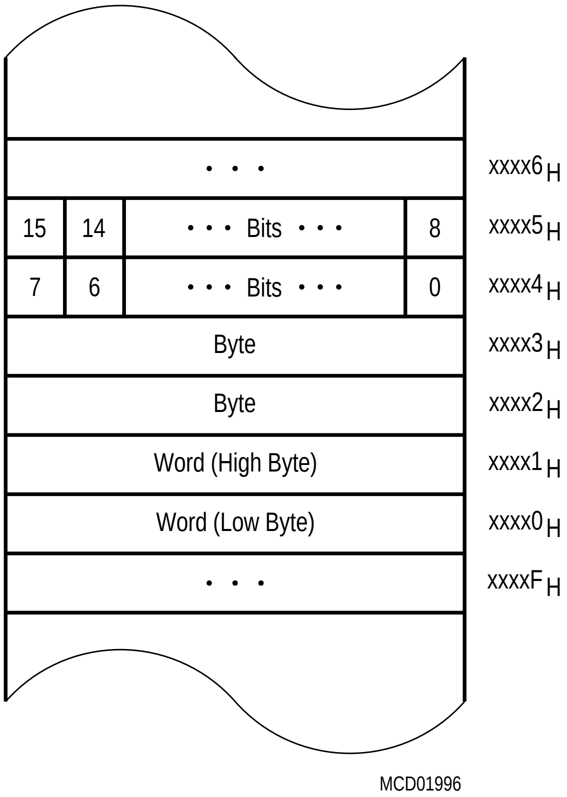


Figure 3-2
Storage of Words, Byte and Bits in a Byte Organized Memory

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

3.1 Internal ROM

The C161RI may reserve an address area of variable size (depending on the version) for on-chip mask-programmable ROM/Flash/OTP memory (organized as X × 32). The lower 32 KByte of this on-chip memory block are referred to as “Internal ROM Area”. Internal ROM accesses are globally enabled or disabled via bit ROMEN in register SYSCON. This bit is set during reset according to the level on pin $\overline{\text{EA}}$, or may be altered via software. If enabled, the internal ROM area occupies the lower 32 KByte of either segment 0 or segment 1. This ROM mapping is controlled by bit ROMS1 in register SYSCON.

Note: The size of the internal ROM area is independent of the size of the actual implemented ROM. Also devices with less than 32 KByte of ROM or with no ROM at all will have this 32 KByte area occupied, if the ROM is enabled. Devices with larger ROMs provide the mapping option only for the ROM area.

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

The internal ROM/Flash 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 ROM 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 ROM to external memory is not supported and causes erroneous results.

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

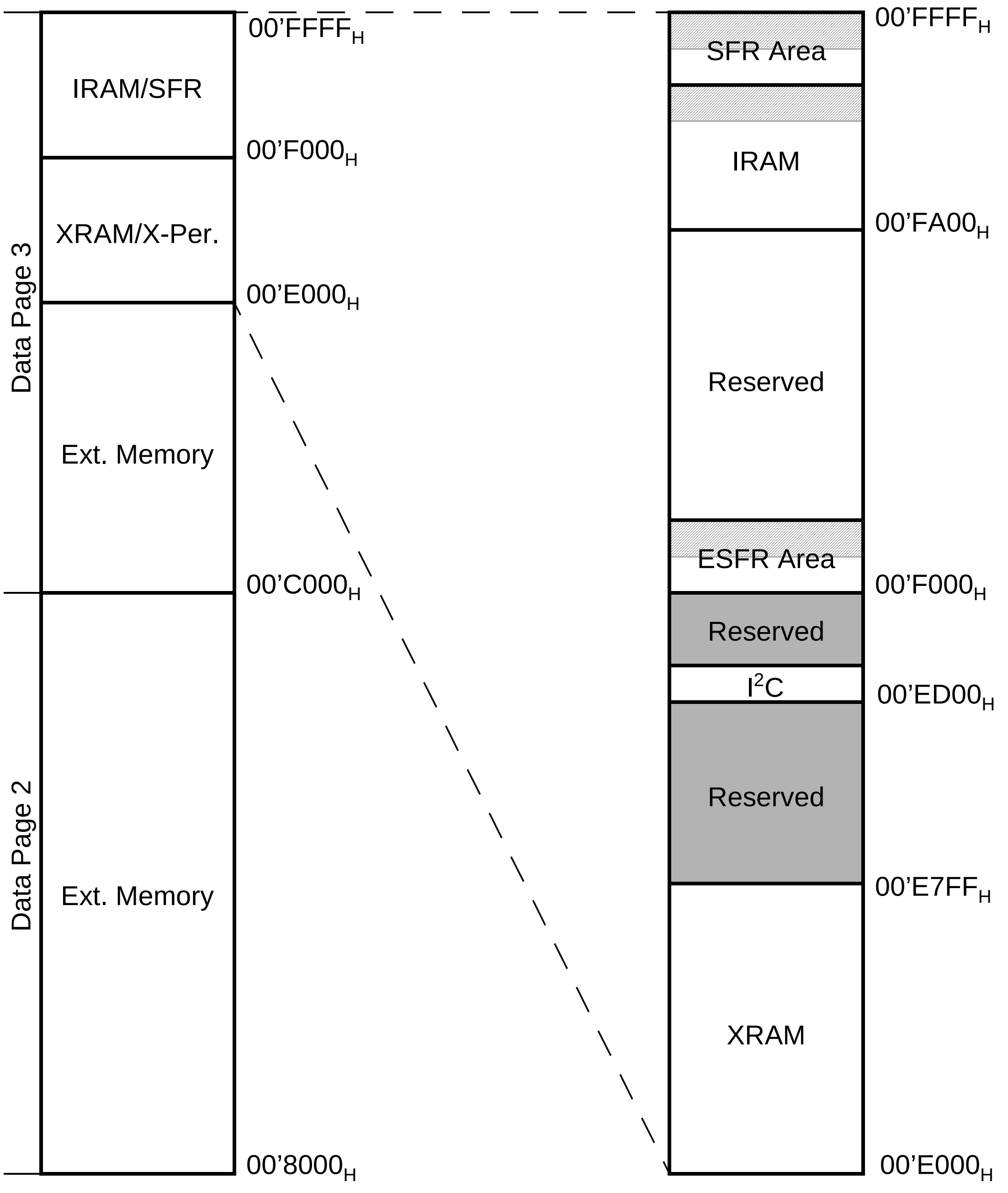
The internal ROM is not provided for single bit storage, and therefore it is not bit addressable.

Note: The ‘x’ in the locations above depend on the available ROM/Flash memory and on the mapping.

The internal ROM may be enabled, disabled or mapped into segment 0 or segment 1 under software control. Chapter “System Programming” shows how to do this and reminds of the precautions that must be taken in order to prevent the system from crashing.

3.2 Internal RAM and SFR Area

The RAM/SFR area is located within data page 3 and provides access to the internal RAM (IRAM, organized as X×16) and to two 512 Byte blocks of Special Function Registers (SFRs).
The C161RI provides 1 KByte of IRAM.



Note: New XBUS peripherals will be preferably placed into the shaded areas, which now access external memory (bus cycles executed).

Figure 3-3
System Memory Map

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

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

Any word and byte data in the internal RAM can be accessed via indirect or long 16-bit addressing modes, if the selected DPP register points to data page 3. Any word data access is made on an even byte address. The highest possible word data storage location in the internal RAM is 00’FDFE$_H$. For PEC data transfers, the internal RAM can be accessed independent of the contents of the DPP registers via the PEC source and destination pointers.

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

System Stack

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

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

For all system stack operations the on-chip RAM is accessed via the Stack Pointer (SP) register. The stack grows downward from higher towards lower RAM address locations. Only word accesses are supported to the system stack. A stack overflow (STKOV) 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 allow to implement a circular stack with hardware supported system stack flushing and filling (except for option “111”).

The technique of implementing this circular stack is described in chapter “System Programming”.

General Purpose Registers

The General Purpose Registers (GPRs) use a block of 16 consecutive words within the internal RAM. The Context Pointer (CP) register determines the base address of the currently active register bank. This register bank may consist of up to 16 word GPRs (R0, R1, …, R15) and/or of up to 16 byte GPRs (RL0, RH0, …, RL7, RH7). The sixteen byte GPRs are mapped onto the first eight word GPRs (see table below).

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

Mapping of General Purpose Registers to RAM Addresses

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

The C161RI supports fast register bank (context) switching. Multiple register banks can physically exist within the internal RAM at the same time. Only the register bank selected by the Context Pointer register (CP) is active at a given time, however. Selecting a new active register bank is simply done by updating the CP register. A particular Switch Context (SCXT) instruction performs register bank switching and an automatic saving of the previous context. The number of implemented register banks (arbitrary sizes) is only limited by the size of the available internal RAM.

Details on using, switching and overlapping register banks are described in chapter “System Programming”.

PEC Source and Destination Pointers

The 16 word locations in the internal RAM from 00’FCE0$_H$ to 00’FCFE$_H$ (just below the bit-addressable section) are provided as source and destination address pointers for data transfers on the eight PEC channels. Each channel uses a pair of pointers stored in two subsequent word locations with the source pointer (SRCPx) on the lower and the destination pointer (DSTPx) on the higher word address (x = 7 … 0).

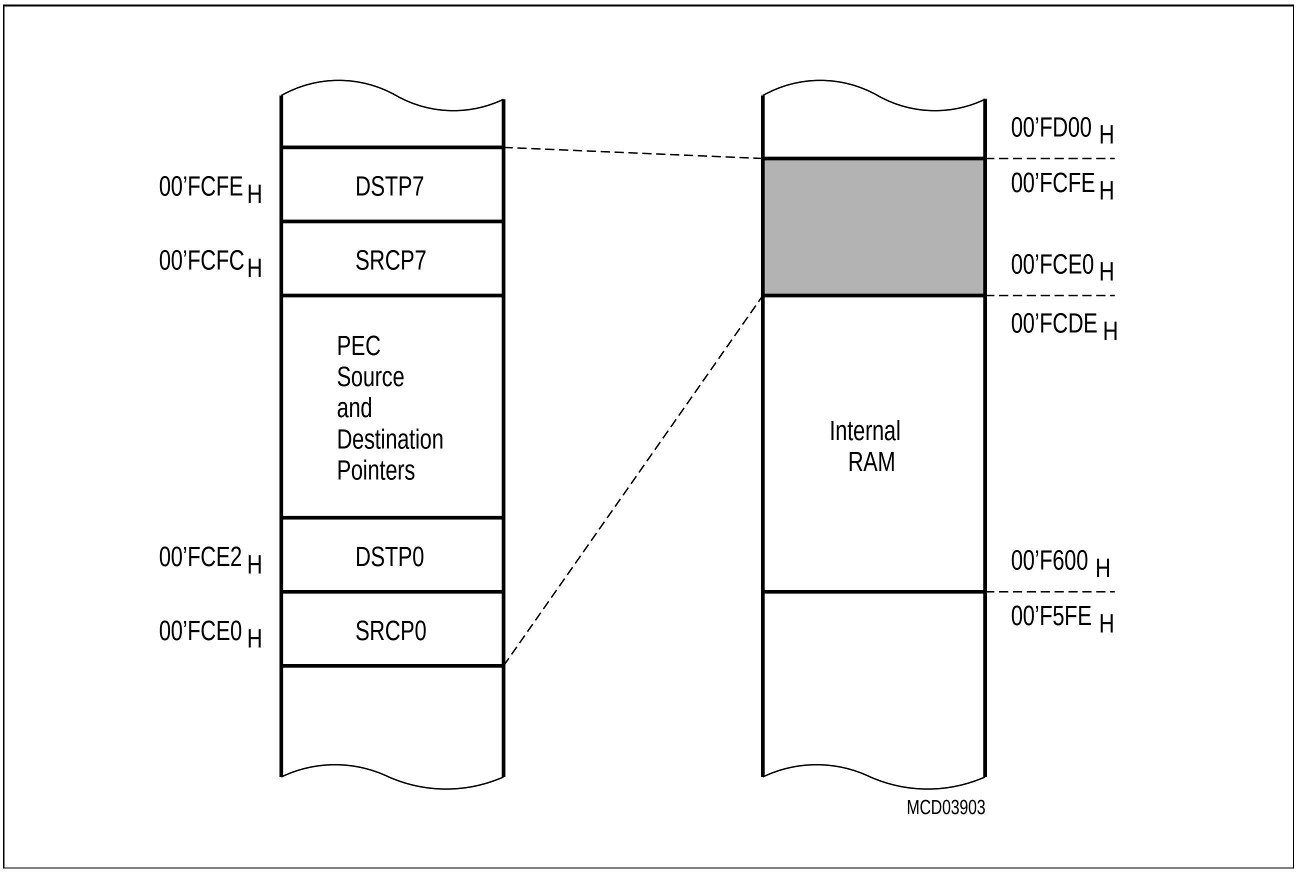


Figure 3-4
Location of the PEC Pointers

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

For more details about the use of the source and destination pointers for PEC data transfers see section “Interrupt and Trap Functions”.

Special Function Registers

The functions of the CPU, the bus interface, the IO ports and the on-chip peripherals of the C161RI are controlled via a number of so-called Special Function Registers (SFRs). These SFRs are arranged within two areas of 512 Byte size each. The first register block, the SFR area, is located in the 512 Bytes above the internal RAM (00’FFFF$_H$ … 00’FE00$_H$), the second register block, the Extended SFR (ESFR) area, is located in the 512 Bytes below the internal RAM (00’F1FF$_H$ … 00’F000$_H$).

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

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

The upper half of each register block is bit-addressable, so the respective control/status bits can directly be modified or checked using bit addressing.

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

```
ESFR_SWITCH_EXAMPLE:
EXTR   #4                       ;Switch to ESFR area for next 4 instr.
MOV    ODP2, #data16            ;ODP2 uses 8-bit reg addressing
BFLDL  DP6, #mask, #data8       ;Bit addressing for bit fields
BSET   DP1H.7                   ;Bit addressing for single bits
MOV    T8REL, R1                ;T8REL uses 16-bit mem address,
                                ;R1 is duplicated into the ESFR space
                                ;(EXTR is not required for this access)

;----  ;-------------------     ;The scope of the EXTR #4 instruction...
                                ;...ends here!

MOV    T8REL, R1                ;T8REL uses 16-bit mem address,
                                ;R1 is accessed via the SFR space
```

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

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

3.3 The On-Chip XRAM

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

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

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

Note: As the XRAM appears like external memory it cannot be used for the C161RI’s system stack or register banks. The XRAM is not provided for single bit storage and therefore is not bit addressable.

The on-chip XRAM is accessed without any waitstates using 16-bit demultiplexed bus cycles which take 125 ns (@ 16 MHz f_{CPU}). Even if the XRAM is used like external memory it does not occupy BUSCONx/ADDRSELx registers but rather is selected via additional dedicated XBCON/XADRS registers. These registers are mask-programmed and are not user accessible. With these registers the address area 00’E000$_H$ to 00’E7FF$_H$ is reserved for XRAM accesses.

XRAM Access via External Masters

When bit XPER-SHARE in register SYSCON is set the on-chip XRAM of the C161RI can be accessed by an external master during hold mode via the C161RI’s bus interface. These external accesses must use the same configuration as internally programmed, ie. demultiplexed bus, 125 ns minimum access cycle time. No waitstates are required.

Note: The configuration in register SYSCON cannot be changed after the execution of the EINIT instruction.

3.4 External Memory Space

The C161RI is capable of using an address space of up to 16 MByte. Only parts of this address space are occupied by internal memory areas. All addresses which are not used for on-chip memory (ROM/Flash/OTP or RAM) or for registers may reference external memory locations. This external memory is accessed via the C161RI’s external bus interface.

Four memory bank sizes are supported:

- Non-segmented mode: 64 KByte with A15 … A0 on PORT0
- 2-bit segmented mode: 256 KByte with A17 ... A16 on Port 4 and A15 … A0 on PORT0
- 4-bit segmented mode: 1 MByte with A19 … A16 on Port 4 and A15 … A0 on PORT0
- 7-bit segmented mode: 8 MByte with A22 … A16 on Port 4 and A15 … A0 on PORT0

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

The C161RI also supports **four different bus types**:

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

Memory model and bus mode are selected during reset by pin $\overline{\text{EA}}$ and PORT0 pins. For further details about the external bus configuration and control please refer to chapter "The External Bus Interface".

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

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

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

3.5 Crossing Memory Boundaries

The address space of the C161RI is implicitly divided into equally sized blocks of different granularity and into logical memory areas. Crossing the boundaries between these blocks (code or data) or areas requires special attention to ensure that the controller executes the desired operations.

Memory Areas are partitions of the address space that represent different kinds of memory (if provided at all). These memory areas are the internal RAM/SFR area, the internal ROM/Flash/OTP (if available), the on-chip X-Peripherals (if integrated) and the external memory.

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

Note: Changing from the external memory area to the internal RAM/SFR area takes place within segment 0.

Segments are contiguous blocks of 64 KByte each. They are referenced via the code segment pointer CSP for code fetches and via an explicit segment number for data accesses overriding the standard DPP scheme.
During code fetching segments are not changed automatically, but rather must be switched explicitly. The instructions JMPS, CALLS and RETS will do this.
In larger sequential programs make sure that the highest used code location of a segment contains an unconditional branch instruction to the respective following segment, to prevent the prefetcher from trying to leave the current segment.

Data Pages are contiguous blocks of 16 KByte each. 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 therefore will use different data page pointers, while the physical locations need not be subsequent within memory.

4 The Central Processing Unit (CPU)

Basic tasks of the CPU are to fetch and decode instructions, to supply operands for the arithmetic and logic unit (ALU), to perform operations on these operands in the ALU, and to store the previously calculated results. As the CPU is the main engine of the C161RI controller, it is also affected by certain actions of the peripheral subsystem.

Since a four stage pipeline is implemented in the C161RI, up to four instructions can be processed in parallel. Most instructions of the C161RI are executed in one machine cycle (2 CPU clock periods) due to this parallelism. This chapter describes how the pipeline works for sequential and branch instructions in general, and which hardware provisions have been made to speed the execution of jump instructions in particular. The general instruction timing is described including standard and exceptional timing.

While internal memory accesses are normally performed by the CPU itself, external peripheral or memory accesses are performed by a particular on-chip External Bus Controller (EBC), which is automatically invoked by the CPU whenever a code or data address refers to the external address space. If possible, the CPU continues operating while an external memory access is in progress. If external data are required but are not yet available, or if a new external memory access is requested by the CPU, before a previous access has been completed, the CPU will be held by the EBC until the request can be satisfied. The EBC is described in a dedicated chapter.

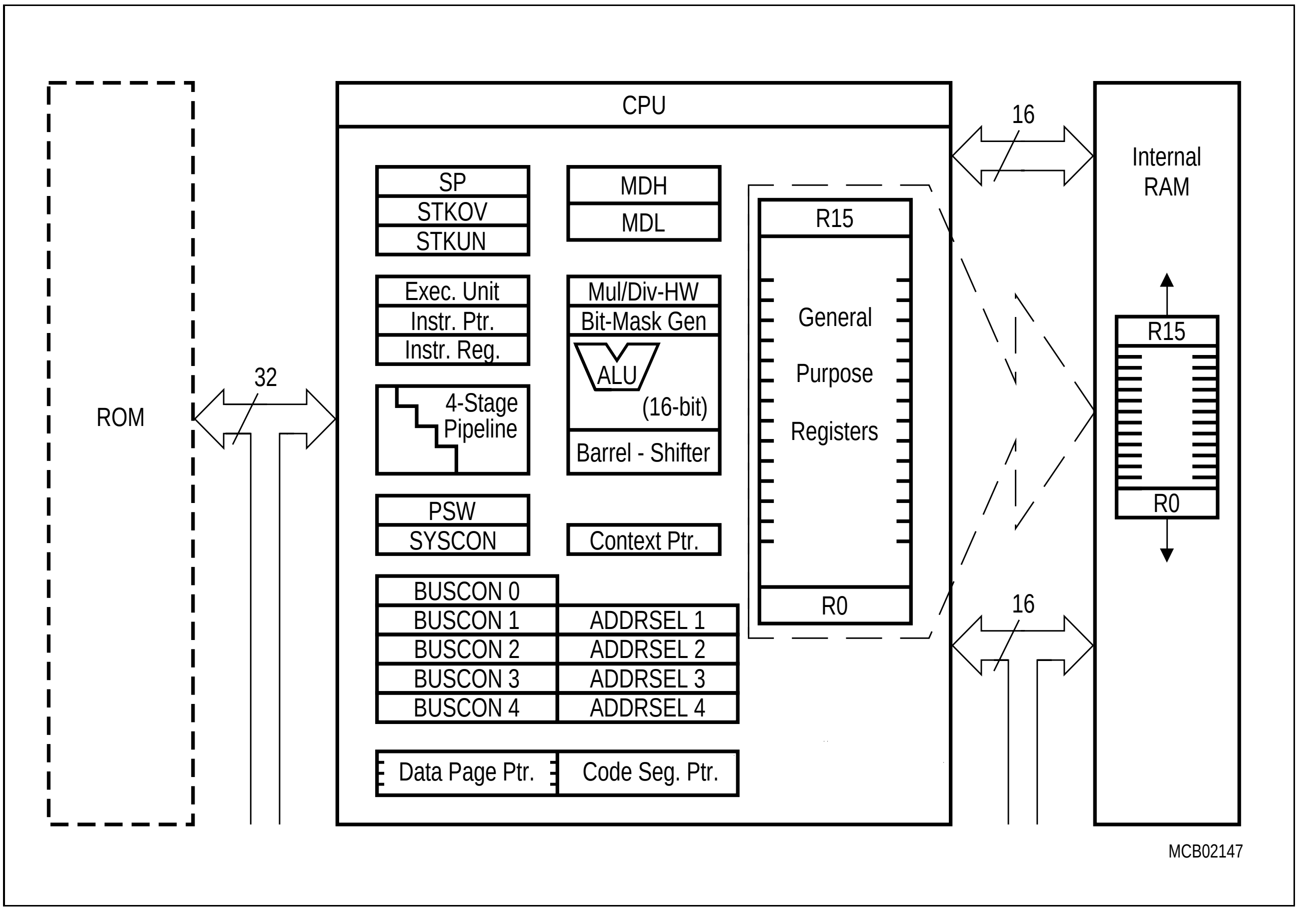


Figure 4-1
CPU Block Diagram

The on-chip peripheral units of the C161RI work nearly independent of the CPU with a separate clock generator. Data and control information is interchanged between the CPU and these peripherals via Special Function Registers (SFRs). Whenever peripherals need a non-deterministic CPU action, an on-chip Interrupt Controller compares all pending peripheral service requests against each other and prioritizes one of them. If the priority of the current CPU operation is lower than the priority of the selected peripheral request, an interrupt will occur.

Basically, there are two types of interrupt processing:

- **Standard interrupt processing** forces the CPU to save the current program status and the return address on the stack before branching to the interrupt vector jump table.

- **PEC interrupt processing** steals just one machine cycle from the current CPU activity to perform a single data transfer via the on-chip Peripheral Event Controller (PEC).

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

In contrast to other on-chip peripherals, there is a closer conjunction between the watchdog timer and the CPU. If enabled, the watchdog timer expects to be serviced by the CPU within a programmable period of time, otherwise it will reset the chip. Thus, the watchdog timer is able to prevent the CPU from going totally astray when executing erroneous code. After reset, the watchdog timer starts counting automatically, but it can be disabled via software, if desired.

Beside its normal operation there are the following particular CPU states:

- **Reset state:** Any reset (hardware, software, watchdog) forces the CPU into a predefined active state.

- **IDLE state:** The clock signal to the CPU itself is switched off, while the clocks for the on-chip peripherals keep running.

- **POWER DOWN state:** All of the on-chip clocks are switched off (RTC clock selectable).

A transition into an active CPU state is forced by an interrupt (if being IDLE) or by a reset (if being in POWER DOWN mode).
The IDLE, POWER DOWN and RESET states can be entered by particular C161RI system control instructions.

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

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

4.1 Instruction Pipelining

The instruction pipeline of the C161RI partitiones instruction processing into four stages of which each one has its individual task:

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

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

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

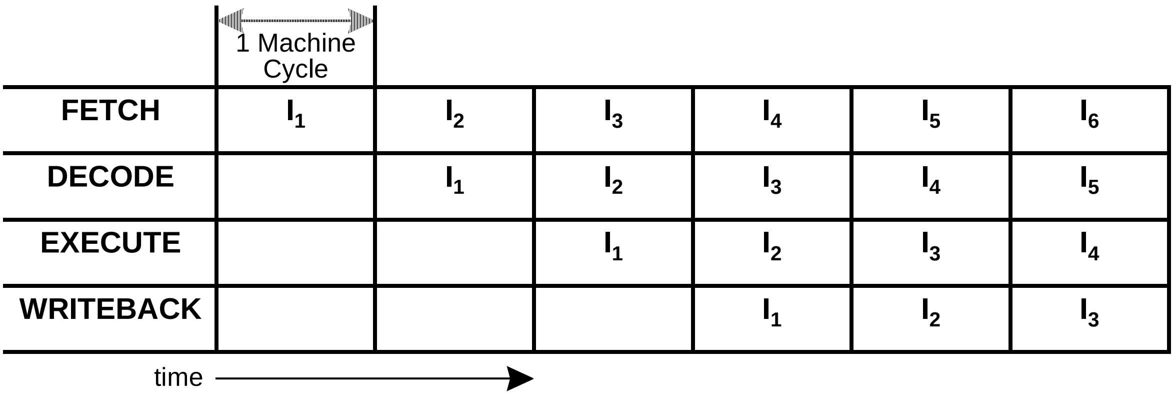
4th –>WRITE BACK:
In this stage all external operands and the remaining operands within the internal RAM space are written back.

A particularity of the C161RI are the so-called injected instructions. These injected instructions are generated internally by the machine to provide the time needed to process instructions, which 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.

Sequential Instruction Processing

Each single instruction has to pass through each of the four pipeline stages regardless of whether all possible stage operations are really performed or not. Since passing through one pipeline stage takes at least one machine cycle, any isolated instruction takes at least four machine cycles to be completed. Pipelining, however, allows parallel (i.e. simultaneous) processing of up to four instructions. Thus, most of the instructions seem to be processed during one machine cycle as soon as the pipeline has been filled once after reset (see figure below).

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

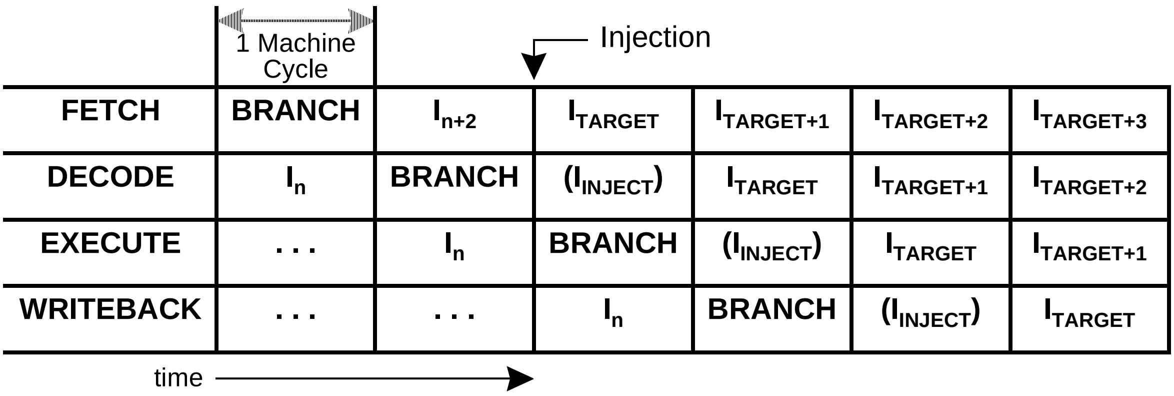


FETCH	I_1	I_2	I_3	I_4	I_5	I_6
DECODE		I_1	I_2	I_3	I_4	I_5
EXECUTE			I_1	I_2	I_3	I_4
WRITEBACK				I_1	I_2	I_3

Figure 4-2
Sequential Instruction Pipelining

Standard Branch Instruction Processing

Instruction pipelining helps to speed sequential program processing. In the case that a branch is taken, the instruction which has already been fetched providently is mostly not the instruction which must be decoded next. Thus, at least one additional machine cycle is normally required to fetch the branch target instruction. This extra machine cycle is provided by means of an injected instruction (see figure below).



FETCH	BRANCH	I_{n+2}	I_{TARGET}	$I_{TARGET+1}$	$I_{TARGET+2}$	$I_{TARGET+3}$
DECODE	I_n	BRANCH	(I_{INJECT})	I_{TARGET}	$I_{TARGET+1}$	$I_{TARGET+2}$
EXECUTE	. . .	I_n	BRANCH	(I_{INJECT})	I_{TARGET}	$I_{TARGET+1}$
WRITEBACK	. . .	. . .	I_n	BRANCH	(I_{INJECT})	I_{TARGET}

Figure 4-3
Standard Branch Instruction Pipelining

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

Cache Jump Instruction Processing

The C161RI incorporates a jump cache to optimize conditional jumps, which are processed repeatedly within a loop. Whenever a jump on cache is taken, the extra time to fetch the branch target instruction can be saved and thus the corresponding cache jump instruction in most cases takes only one machine cycle.

This performance is achieved by the following mechanism:
Whenever a cache jump instruction passes through the decode stage of the pipeline for the first time (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 repeatedly following 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 decode stage of the pipeline (see figure below).

A time saving jump on cache is always taken after the second and any further occurrence of the same cache jump instruction, unless an instruction which, 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.

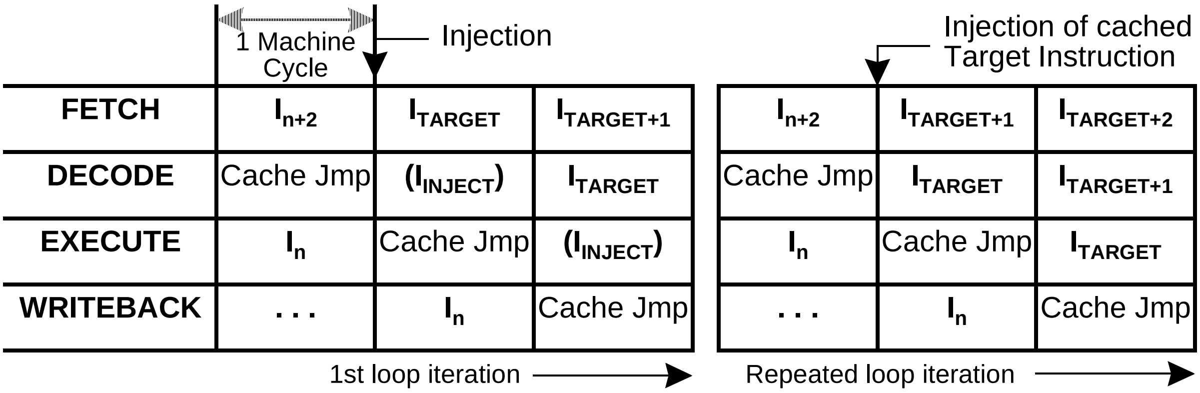


Figure 4-4
Cache Jump Instruction Pipelining

Particular Pipeline Effects

Since up to four different instructions are processed simultaneously, additional hardware has been spent in the C161RI to consider all causal dependencies which may exist on instructions in different pipeline stages without a loss of performance. This extra hardware (i.e. for 'forwarding' operand read and write values) resolves most of the possible conflicts (e.g. multiple usage of buses) in a time optimized way and thus avoids that the pipeline becomes noticeable for the user in most cases. However, there are some very rare cases, where the circumstance that the C161RI is a pipelined machine requires attention by the programmer. In these cases the delays caused by pipeline conflicts can be used for other instructions in order to optimize performance.

- **Context Pointer Updating**

An instruction, which calculates a physical GPR operand address via the CP register, is mostly not capable of using a new CP value, which is to be updated by an immediately preceding instruction. Thus, to make sure that the new CP value is used, at least one instruction must be inserted between a CP-changing and a subsequent GPR-using instruction, 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  : MOV   R0, #dataX         ; write to GPR 0 in the new context
```

- **Data Page Pointer Updating**

An instruction, which calculates a physical operand address via a particular DPPn (n = 0 to 3) register, is mostly not capable of using a new DPPn register value, which is to be updated by an immediately 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 which 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 segmentation is enabled
```

- **Explicit Stack Pointer Updating**

None of the RET, RETI, RETS, RETP or POP instructions is capable of correctly using a new SP register value, which 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 explicitly SP-writing and any subsequent of the just mentioned implicitly SP-using instructions, as shown in the following example:

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

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

- **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:

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

- **Controlling Interrupts**

Software modifications (implicit or explicit) of the PSW are done in the execute phase of the respective 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. Timecritical instruction sequences therefore should not begin directly after the instruction disabling interrupts, as shown in the following example:

```
INT_OFF:   BCLR  IEN              ; globally disable interrupts
           $I_{N-1}$              ; non-critical instruction
CRIT_1ST:  $I_N$                  ; begin of uninterruptable critical sequence
           . . .
CRIT_LAST: $I_{N+x}$              ; end of uninterruptable critical sequence
INT_ON:    BSET  IEN              ; globally re-enable interrupts
```

Note: The described delay of 1 instruction also applies for enabling the interrupts system i.e. no interrupt requests are acknowledged until the instruction following the enabling instruction.

- **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 ROM, segmentation, stack size) cannot use the new resources (e.g. ROM or stack). In these cases an instruction that does not access these resources should be inserted. Code accesses to the new ROM area are only possible after an absolute branch to this area.

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

- **BUSCON/ADDRSEL**

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

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

- **Timing**

Instruction pipelining reduces the average instruction processing time in a wide scale (from four to one machine cycles, mostly). 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.

Besides a general execution time description, the following section provides some hints on how to optimize time-critical program parts with regard to such pipeline-caused timing particularities.

4.2 Bit-Handling and Bit-Protection

The C161RI provides several mechanisms to manipulate bits. These mechanisms either manipulate software flags within the internal RAM, control on-chip peripherals via control bits in their respective SFRs or control IO functions via port pins.

The instructions BSET, BCLR, BAND, BOR, BXOR, BMOV, BMOVN explicitly set or clear specific bits. The instructions BFLDL and BFLDH allow to manipulate up to 8 bits of a specific byte at one time. The instructions JBC and JNBS implicitly clear or set the specified bit when the jump is taken. The instructions JB and JNB (also conditional jump instructions that refer to flags) evaluate the specified bit to determine if the jump is to be taken.

Note: Bit operations on undefined bit locations will always read a bit value of ‘0’, while the write access will not effect the respective bit location.

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

This method has several consequences:

- Bits can only be modified within the internal address areas, i.e. internal RAM and SFRs. External locations cannot be used with bit instructions.

The upper 256 bytes of the SFR area, the ESFR area and the internal RAM are bit-addressable (see chapter “Memory Organization”), i.e. those register bits located within the respective sections can be directly manipulated using bit instructions. The other SFRs must be accessed byte/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 writeback would overwrite the new bit value generated by the hardware. The solution is either the implemented hardware protection (see below) or realized through special programming (see “Particular Pipeline Effects”).

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

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

A summary of the protected bits implemented in the C161RI can be found at the end of chapter “Architectural Overview”.

4.3 Instruction State Times

Basically, 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 C161RI 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 C161RI's on-chip External Bus Controller (EBC), which works in parallel with the CPU.

This section summarizes the execution times in a very condensed way. A detailed description of the execution times for the various instructions and the specific exceptions can be found in the **"C16x Family Instruction Set Manual"**.

The table below shows the minimum execution times required to process a C161RI instruction fetched from the internal code memory, the internal RAM or from external memory. These execution times apply to most of the C161RI instructions - except some of the branches, the multiplication, the division and a special move instruction. In case of internal ROM program execution there is no execution time dependency on the instruction length except for some special branch situations. The numbers in the table are in units of CPU clock cycles and assume no waitstates.

Minimum Execution Times

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

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

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

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

4.4 CPU Special Function Registers

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

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

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

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

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

The System Configuration Register SYSCON

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

SYSCON (FF12$_H$ / 89$_H$) SFR **Reset Value: 0XX0$_H$**

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

Bit	Function
XPER-SHARE	**XBUS Peripheral Share Mode Control** 0: External accesses to XBUS peripherals are disabled 1: XBUS peripherals are accessible via the external bus during hold mode
VISIBLE	**Visible Mode Control** 0: Accesses to XBUS peripherals are done internally 1: XBUS peripheral accesses are made visible on the external pins

Bit	Function
XPEN	**XBUS Peripheral Enable Bit** 0: Accesses to the on-chip X-Peripherals and their functions are disabled 1: The on-chip X-Peripherals are enabled and can be accessed
BDRSTEN	**Bidirectional Reset Enable Bit** 0: Pin $\overline{\text{RSTIN}}$ is an input only. 1: Pin $\overline{\text{RSTIN}}$ is pulled low during the internal reset sequence.
OWDDIS	**Oscillator Watchdog Disable Bit** 0: The on-chip oscillator watchdog is enabled and active. 1: The on-chip oscillator watchdog is disabled and the CPU clock is always fed from the oscillator input.
CSCFG	**Chip Select Configuration Control** 0: Latched $\overline{\text{CS}}$ mode. The $\overline{\text{CS}}$ signals are latched internally and driven to the (enabled) port pins synchronously. 1: Unlatched $\overline{\text{CS}}$ mode. The $\overline{\text{CS}}$ signals are directly derived from the address and driven to the (enabled) port pins.
WRCFG	**Write Configuration Control** (Set according to pin P0H.0 during reset) 0: Pins $\overline{\text{WR}}$ and $\overline{\text{BHE}}$ retain their normal function 1: Pin $\overline{\text{WR}}$ acts as $\overline{\text{WRL}}$, pin $\overline{\text{BHE}}$ acts as $\overline{\text{WRH}}$
CLKEN	**System Clock Output Enable** (CLKOUT) 0: CLKOUT disabled: pin may be used for general purpose IO 1: CLKOUT enabled: pin outputs the system clock signal
BYTDIS	**Disable/Enable Control for Pin $\overline{\text{BHE}}$** (Set according to data bus width) 0: Pin $\overline{\text{BHE}}$ enabled 1: Pin $\overline{\text{BHE}}$ disabled, pin may be used for general purpose IO
ROMEN	**Internal ROM Enable** (Set according to pin $\overline{\text{EA}}$ during reset) 0: Internal ROM disabled: accesses to the ROM area use the external bus 1: Internal ROM enabled
SGTDIS	**Segmentation Disable/Enable Control** 0: Segmentation enabled (CSP and IP are saved/restored during interrupt entry/exit) 1: Segmentation disabled (Only IP is saved/restored)
ROMS1	**Internal ROM Mapping** 0: Internal ROM area mapped to segment 0 (00’0000$_H$ … 00’7FFF$_H$) 1: Internal ROM area mapped to segment 1 (01’0000$_H$ … 01’7FFF$_H$)
STKSZ	**System Stack Size** Selects the size of the system stack (in the internal RAM) from 32 to 1024 words

Note: Register SYSCON cannot be changed after execution of the EINIT instruction.
The function of bits XPER-SHARE, VISIBLE, WRCFG, BYTDIS, ROMEN and ROMS1 is described in more detail in chapter “The External Bus Controller”.

System Clock Output Enable (CLKEN)

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

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

Segmentation Disable/Enable Control (SGTDIS)

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

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

System Stack Size (STKSZ)

This bitfield defines the size of the physical system stack, which is located in the internal RAM of the C161RI. An area of 32 … 256 words or all of the internal RAM may be dedicated to the system stack. A so-called "circular stack" mechanism allows to use a bigger virtual stack than this dedicated RAM area.

These techniques as well as the encoding of bitfield STKSZ are described in more detail in chapter "System Programming".

The Processor Status Word PSW

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

PSW (FF10$_H$ / 88$_H$) SFR **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ILVL				**IEN**	**HLDEN**	**-**	**-**	**-**	**USR0**	**MULIP**	**E**	**Z**	**V**	**C**	**N**
rw				rw	rw	-	-	-	rw	rw	rw	rw	rw	rw	rw

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

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

The condition flags (N, C, V, Z, E) within the PSW indicate the ALU status due to the last recently performed ALU operation. They are set by most of the instructions due to specific rules, which depend on the ALU or data movement operation performed by an instruction.

After execution of an instruction which explicitly updates the PSW register, the condition flags cannot be interpreted as described in the following, because any explicit write to the PSW register supersedes the condition flag values, which are implicitly generated by the CPU. Explicitly reading the PSW register supplies a read value which represents the state of the PSW register after execution of the immediately preceding instruction.

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

• **N-Flag:** For most of the ALU operations, the N-flag is set to ’1’, if the most significant bit of the result contains a ’1’, otherwise it is cleared. In the case of integer operations the N-flag can be interpreted as the sign bit of the result (negative: N = ’1’, positive: N = ’0’). Negative numbers are always represented as the 2's complement of the corresponding positive number. The range of signed numbers extends from '– 8000_H' to '+ $7FFF_H$' for the word data type, or from '– 80_H' to '+ $7F_H$' for the byte data type. For Boolean bit operations with only one operand the N-flag represents the previous state of the specified bit. For Boolean bit operations with two operands the N-flag represents the logical XORing of the two specified bits.

• **C-Flag:** After an addition the C-flag indicates that a carry from the most significant bit of the specified word or byte data type has been generated. After a subtraction or a comparison the C-flag indicates a borrow, which represents the logical negation of a carry for the addition.
This means that the C-flag is set to ’1’, if **no** carry from the most significant bit of the specified word or byte data type has been generated during a subtraction, which is performed internally by the ALU as a 2’s complement addition, and the C-flag is cleared when this complement addition caused a carry.
The C-flag is always cleared for logical, multiply and divide ALU operations, because these operations cannot cause a carry anyhow.
For shift and rotate operations the C-flag represents the value of the bit shifted out last. If a shift count of zero is specified, the C-flag will be cleared. The C-flag is also cleared for a prioritize ALU operation, because a ’1’ is never shifted out of the MSB during the normalization of an operand.
For Boolean bit operations with only one operand the C-flag is always cleared. For Boolean bit operations with two operands the C-flag represents the logical ANDing of the two specified bits.

• **V-Flag:** For addition, subtraction and 2’s complementation the V-flag is always set to ’1’, if the result overflows the maximum range of signed numbers, which are representable by either 16 bits for word operations ('– 8000_H' to '+ $7FFF_H$'), or by 8 bits for byte operations ('– 80_H' to '+ $7F_H$'), otherwise the V-flag is cleared. Note that the result of an integer addition, integer subtraction, or 2's complement is not valid, if the V-flag indicates an arithmetic overflow.
For multiplication and division the V-flag is set to '1', if the result cannot be represented in a word data type, otherwise it is cleared. Note that a division by zero will always cause an overflow. In contrast to the result of a division, the result of a multiplication is valid regardless of whether the V-flag is set to '1' or not.
Since logical ALU operations cannot produce an invalid result, the V-flag is cleared by these operations.

The V-flag is also used as 'Sticky Bit' for rotate right and shift right operations. With only using the C-flag, a rounding error caused by a shift right operation can be estimated up to a quantity of one half of the LSB of the result. In conjunction with the V-flag, the C-flag allows evaluating the rounding error 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.

Shift Right Rounding Error Evaluation

C-Flag	V-Flag	Rounding Error Quantity
0	0	- No rounding error -
0	1	0 < Rounding error $< {}^{1}/_{2}$ LSB
1	0	Rounding error $= {}^{1}/_{2}$ LSB
1	1	Rounding error $> {}^{1}/_{2}$ LSB

- **Z-Flag:** The Z-flag is normally set to ’1’, if the result of an ALU operation equals zero, otherwise it is cleared.
For the addition and subtraction with carry the Z-flag is only set to ’1’, if the Z-flag already contains a ’1’ and the result of the current ALU operation additionally equals zero. This mechanism is provided for the support of multiple precision calculations.
For Boolean bit operations with only one operand the Z-flag represents the logical negation of the previous state of the specified bit. For Boolean bit operations with two operands the Z-flag represents the logical NORing of the two specified bits. For the prioritize ALU operation the Z-flag indicates, if the second operand was zero or not.

- **E-Flag:** The E-flag can be altered by instructions, which perform ALU or data movement operations. The E-flag is cleared by those instructions which cannot be reasonably used for table search operations. In all other cases the E-flag is set depending on the value of the source operand to signify whether the end of a search table is reached or not. If the value of the source operand of an instruction equals the lowest negative number, which is representable by the data format of the corresponding instruction (’8000$_H$’ for the word data type, or ’80$_H$’ for the byte data type), the E-flag is set to ’1’, otherwise it is cleared.

- **MULIP-Flag:** The MULIP-flag will be set to ’1’ by hardware upon the entrance into an interrupt service routine, when a multiply or divide ALU operation was interrupted before completion. Depending on the state of the MULIP bit, the hardware decides whether 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 the 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 interrupting service routine does not return to the interrupted multiply/divide instruction (i.e. 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 bit allows to globally enable (IEN = ’1’) or disable (IEN = ’0’) interrupts. The four-bit Interrupt Level field (ILVL) specifies the priority of the current CPU activity. The interrupt level is updated by hardware upon entry into an interrupt service routine, but it can also be modified via software to prevent other interrupts from being acknowledged. In case an interrupt 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 chapter “Interrupt and Trap Functions”.

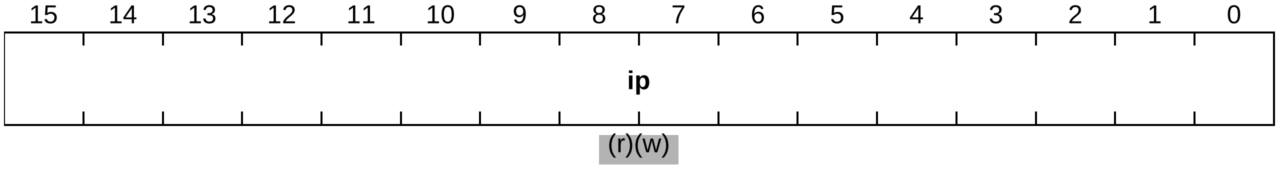
After reset all interrupts are globally disabled, and the lowest priority (ILVL = 0) is assigned to the initial CPU activity.

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 C161RI's address space, and thus it is not directly accessable by the programmer. The IP can, however, be modified indirectly via the stack by means of a return instruction.

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

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



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

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

CSP (FE08$_H$ / 04$_H$) SFR **Reset Value: 0000$_H$**

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

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

Code memory addresses are generated by directly extending the 16-bit contents of the IP register by the contents of the CSP register as shown in the figure below.

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

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

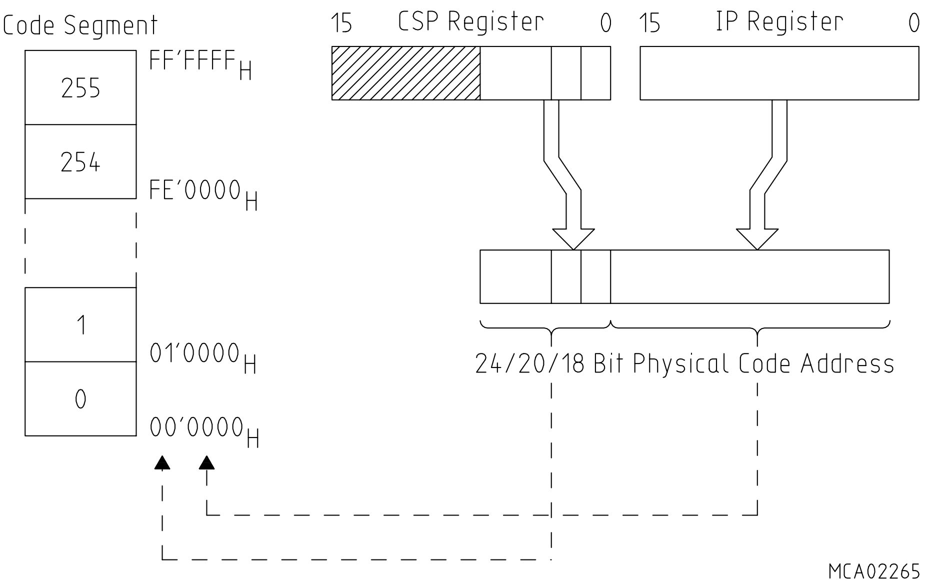


Figure 4-5
Addressing via the Code Segment Pointer

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

The Data Page Pointers DPP0, DPP1, DPP2, DPP3

These four non-bit addressable registers select up to four different data pages being active simultaneously at run-time. 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 future use. The DPP registers allow to access the entire memory space in pages of 16 Kbytes each.

The DPP registers are implicitly used, whenever data accesses to any memory location are made via indirect or direct long 16-bit addressing modes (except for override accesses via EXTended instructions and PEC data transfers). After reset, the Data Page Pointers are initialized in a way that all indirect or direct long 16-bit addresses result in identical 18-bit addresses. This allows to access data pages 3 … 0 within segment 0 as shown in the figure below. If the user does not want to use any data paging, no further action is required.

DPP0 (FE00$_H$ / 00$_H$) SFR **Reset Value: 0000$_H$**

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

DPP1 (FE02$_H$ / 01$_H$) SFR **Reset Value: 0001$_H$**

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

DPP2 (FE04$_H$ / 02$_H$) SFR **Reset Value: 0002$_H$**

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

DPP3 (FE06$_H$ / 03$_H$) SFR **Reset Value: 0003$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	**-**	**-**	**-**	**-**	**-**	**DPP3PN**									
-	-	-	-	-	-	rw									

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

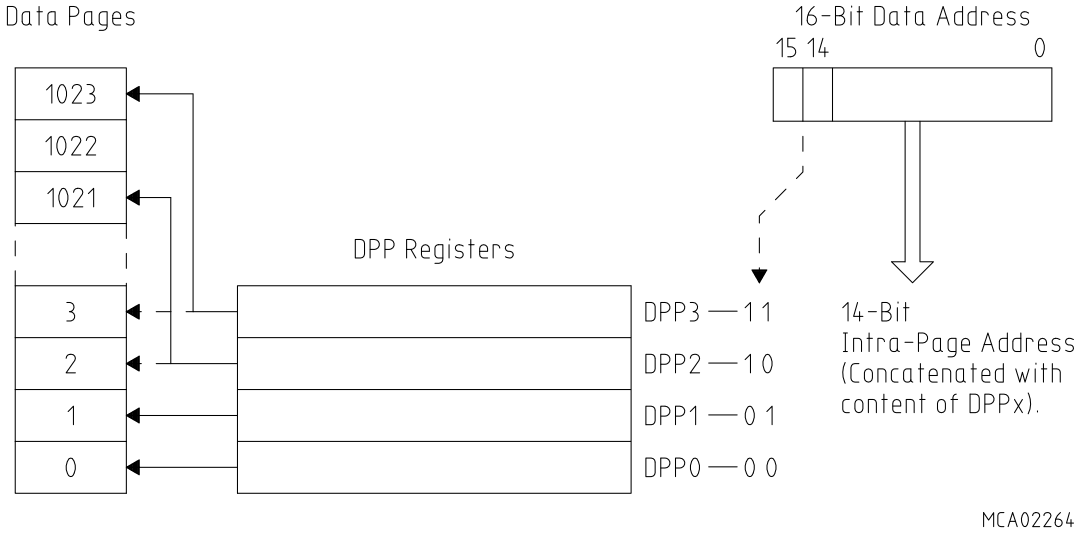
Data paging is performed by concatenating the lower 14 bits of an indirect or direct long 16-bit address with the contents of the DPP register selected by the upper two bits of the 16-bit address. The contents of the selected DPP register specify one of the 1024 possible data pages. This data page base address together with the 14-bit page offset forms the physical 24-bit address (selectable part is driven to the address pins).

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

In case of the segmented memory mode the selected number of segment address bits (via bitfield SALSEL) of the respective DPP register is output on the respective segment address pins of Port 4 for all external data accesses.

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

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



After reset or with segmentation disabled the DPP registers select data pages 3 … 0.
All of the internal memory is accessible in these cases.

Figure 4-6
Addressing via the Data Page Pointers

The Context Pointer CP

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

CP (FE10$_H$ / 08$_H$) SFR **Reset Value: FC00$_H$**

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

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

Note: It is the user’s responsibility that the physical GPR address specified via CP register plus short GPR address must always be an internal RAM location. If this condition is not met, unexpected results may occur.

- Do not set CP below the IRAM start address, i.e. 00’FA00$_H$/00’F600$_H$/00’F200$_H$ (1/2/3KB)
- Do not set CP above 00’FDFE$_H$
- Be careful using the upper GPRs with CP above 00’FDE0$_H$

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

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

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

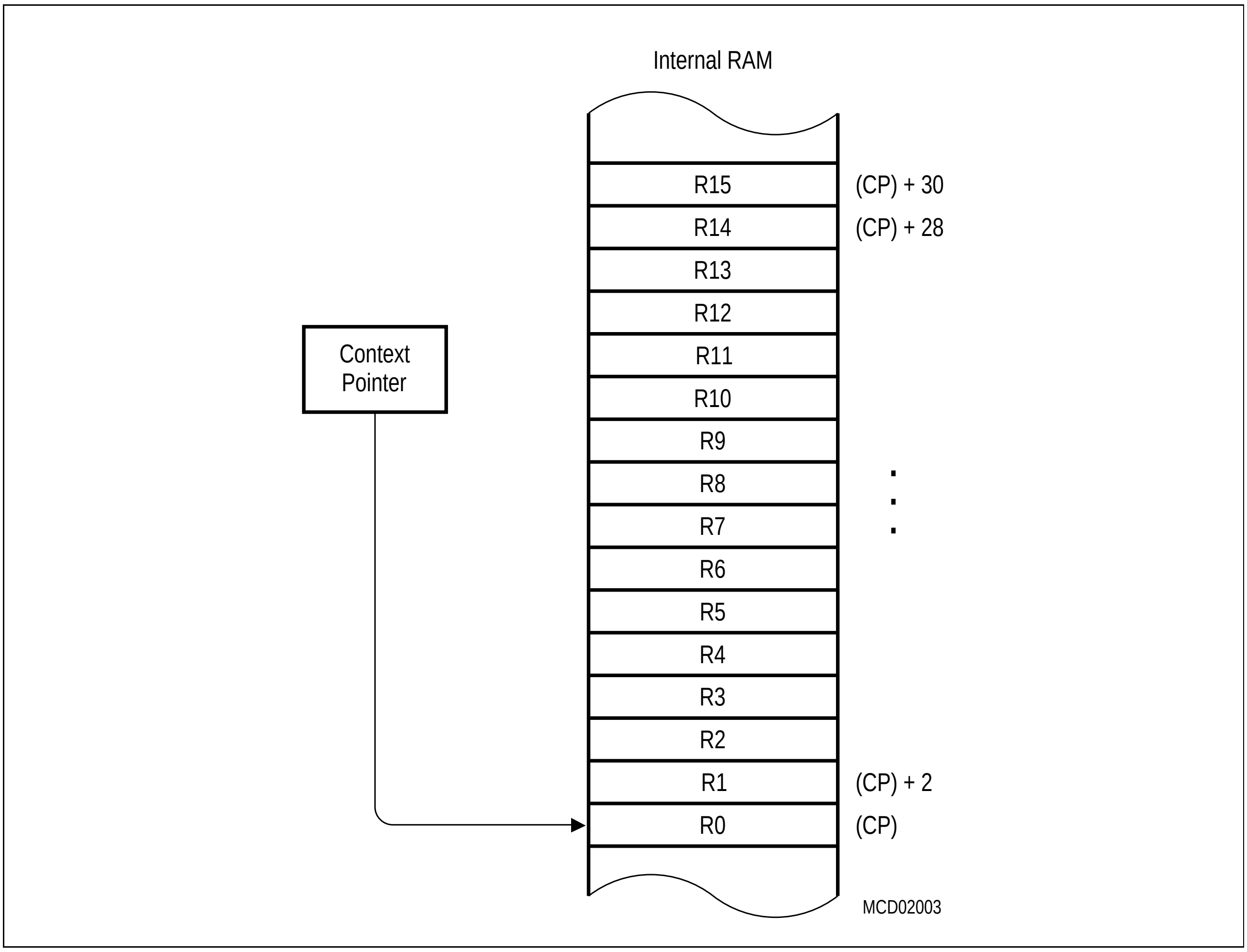


Figure 4-7
Register Bank Selection via Register CP

Several addressing modes use register CP implicitly for address calculations. The addressing modes mentioned below are described in chapter “Instruction Set Summary”.

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

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

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

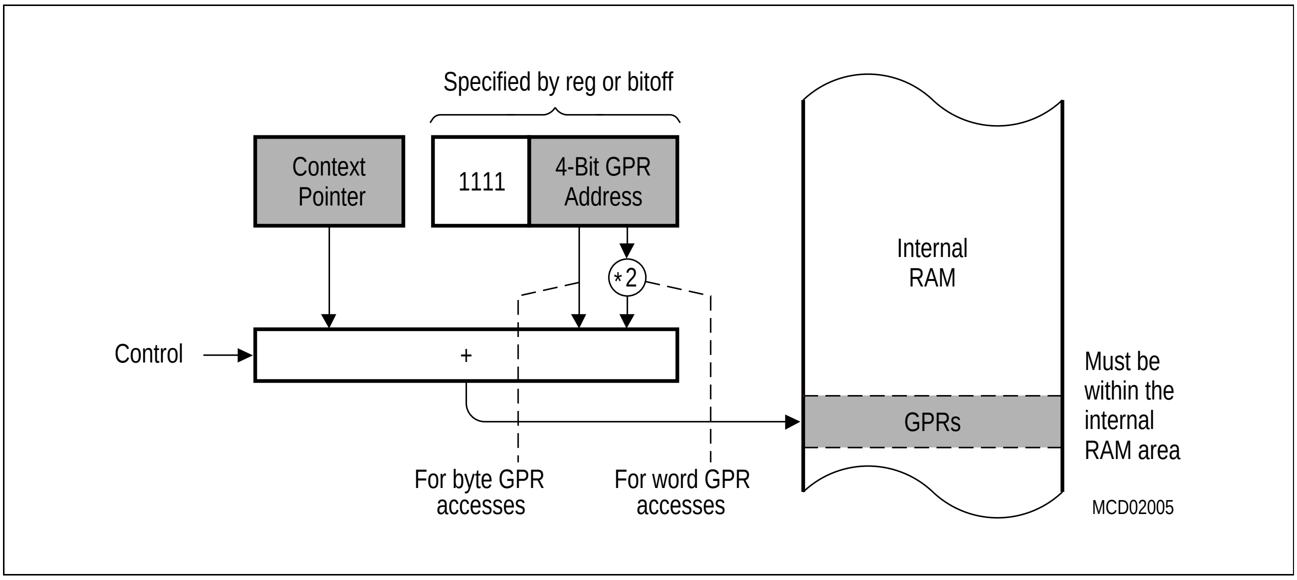


Figure 4-8
Implicit CP Use by Short GPR Addressing Modes

The Stack Pointer SP

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

Since the least significant bit of register SP is tied to ’0’ and bits 15 through 12 are tied to ’1’ by hardware, the SP register can only contain values from F000$_H$ to FFFE$_H$. This allows to access a physical stack within the internal RAM of the C161RI. A virtual stack (usually bigger) can be realized via software. This mechanism is supported by registers STKOV and STKUN (see respective descriptions below).

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

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

SP (FE12$_H$ / 09$_H$) SFR **Reset Value: FC00$_H$**

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

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

The Stack Overflow Pointer STKOV

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

Since the least significant bit of register STKOV is tied to ’0’ and bits 15 through 12 are tied to ’1’ by hardware, the STKOV register can only contain values from F000$_H$ to FFFE$_H$.

STKOV (FE14$_H$ / 0A$_H$) SFR **Reset Value: FA00$_H$**

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

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

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

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

More details about the stack overflow trap service routine and virtual stack management are given in chapter “System Programming”.

The Stack Underflow Pointer STKUN

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

Since the least significant bit of register STKUN is tied to ’0’ and bits 15 through 12 are tied to ’1’ by hardware, the STKUN register can only contain values from F000$_H$ to FFFE$_H$.

STKUN (FE16$_H$ / 0B$_H$) SFR **Reset Value: FC00$_H$**

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

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

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

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

More details about the stack underflow trap service routine and virtual stack management are given in chapter “System Programming”.

Scope of Stack Limit Control

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

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

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

The Multiply/Divide High Register MDH

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

MDH (FE0C$_H$ / 06$_H$) SFR **Reset Value: 0000$_H$**

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

Bit	Function
mdh	Specifies 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 register (MDC) is set to ’1’.

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

A detailed description of how to use the MDH register for programming multiply and divide algorithms can be found in chapter “System Programming”.

The Multiply/Divide Low Register MDL

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

MDL (FE0E$_H$ / 07$_H$) SFR **Reset Value: 0000$_H$**

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

Bit	Function
mdl	Specifies the low 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 register (MDC) is set to ’1’. The MDRIU flag is cleared, whenever the MDL register is read via software.

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

A detailed description of how to use the MDL register for programming multiply and divide algorithms can be found in chapter “System Programming”.

The Multiply/Divide Control Register MDC

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

MDC (FF0E$_H$ / 87$_H$) SFR **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**!!**	**!!**	**!!**	**MDRIU**	**!!**	**!!**	**!!**	**!!**
-	-	-	-	-	-	-	-	r(w)	r(w)	r(w)	r(w)	r(w)	r(w)	r(w)	r(w)

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

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

The MDRIU flag is the only portion of the MDC register which might be of interest 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 in another way than described above. Otherwise, a correct continuation of an interrupted multiply or divide operation cannot be guaranteed.

A detailed description of how to use the MDC register for programming multiply and divide algorithms can be found in chapter “System Programming”.

The Constant Zeros Register ZEROS

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

ZEROS (FF1C$_H$ / 8E$_H$) SFR **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

The Constant Ones Register ONES

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

ONES (FF1E$_H$ / 8F$_H$) SFR **Reset Value: FFFF$_H$**

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

5 Interrupt and Trap Functions

The architecture of the C161RI supports several mechanisms for fast and flexible response to service requests that can be generated from various sources internal or external to the microcontroller.

These mechanisms include:

Normal Interrupt Processing

The CPU temporarily suspends the current program execution and branches to an interrupt service routine in order to service an interrupt requesting device. The current program status (IP, PSW, in segmentation mode also CSP) is saved on the internal system stack. A prioritization scheme with 16 priority levels allows the user to specify the order in which multiple interrupt requests are to be handled.

Interrupt Processing via the Peripheral Event Controller (PEC)

A faster alternative to normal software controlled interrupt processing is servicing an interrupt requesting device with the C161RI's integrated Peripheral Event Controller (PEC). Triggered by an interrupt request, the PEC performs a single word or byte data transfer between any two locations in segment 0 (data pages 0 through 3) through one of eight programmable PEC Service Channels. During a PEC transfer the normal program execution of the CPU is halted for just 1 instruction cycle. No internal program status information needs to be saved. The same prioritization scheme is used for PEC service as for normal interrupt processing. PEC transfers share the 2 highest priority levels.

Trap Functions

Trap functions are activated in response to special conditions that occur during the execution of instructions. A trap can also be caused externally by the Non-Maskable Interrupt pin $\overline{\text{NMI}}$. Several hardware trap functions are provided for handling erroneous conditions and exceptions that arise during the execution of an instruction. 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 on the system stack.

External Interrupt Processing

Although the C161RI does not provide dedicated interrupt pins, it allows to connect external interrupt sources and provides several mechanisms to react on external events, including standard inputs, non-maskable interrupts and fast external interrupts. These interrupt functions are alternate port functions, except for the non-maskable interrupt and the reset input.

5.1 Interrupt System Structure

The C161RI provides 27 separate interrupt nodes that may be assigned to 16 priority levels. In order to support modular and consistent software design techniques, most sources of an interrupt or PEC request are supplied with a separate interrupt control register and interrupt vector. The control register contains the interrupt request flag, the interrupt enable bit, and the interrupt priority of the associated source. Each source request is then activated by one specific event, depending on the selected operating mode of the respective device. For efficient usage of the resources also multi-source interrupt nodes are incorporated. These nodes can be activated by several source requests, e.g. as different kinds of errors in the serial interfaces. However, specific status flags which identify the type of error are implemented in the serial channels' control registers. Additional sharing of interrupt nodes is supported via the interrupt subnode control register ISNC (see description below).

The C161RI provides a vectored interrupt system. In this system specific vector locations in the memory space are reserved for the reset, trap, and interrupt service functions. Whenever a request occurs, the CPU branches to the location that is associated with the respective interrupt source. This allows direct identification of the source that caused the request. The only exceptions are the class B hardware traps, which all share the same interrupt vector. The status flags in the Trap Flag Register (TFR) can then be used to determine which exception caused the trap. For the special software TRAP instruction, the vector address is specified by the operand field of the instruction, which is a seven bit trap number.

The reserved vector locations build a jump table in the low end of the C161RI's address space (segment 0). The jump table is made up of the appropriate jump instructions that transfer control to the interrupt or trap service routines, which may be located anywhere within the address space. The entries of the jump table are located at the lowest addresses in code segment 0 of the address space. Each entry occupies 2 words, except for the reset vector and the hardware trap vectors, which occupy 4 or 8 words.

The table below lists all sources that are capable of requesting interrupt or PEC service in the C161RI, the associated interrupt vectors, their locations and the associated trap numbers. It also lists the mnemonics of the affected Interrupt Request flags and their corresponding Interrupt Enable flags. The mnemonics are composed of a part that specifies the respective source, followed by a part that specifies their function (IR = Interrupt Request flag, IE = Interrupt Enable flag).

Note: Each entry of the interrupt vector table provides room for two word instructions or one doubleword instruction. The respective vector location results from multiplying the trap number by 4 (4 bytes per entry).
All interrupt nodes that are currently not used by their associated modules or are not connected to a module in the actual derivative may be used to generate software controlled interrupt requests by setting the respective IR flag.

Source of Interrupt or PEC Service Request	**Request Flag**	**Enable Flag**	**Interrupt Vector**	**Vector Location**	**Trap Number**
Fast External Interrupt 0	CC8IR	CC8IE	CC8INT	$00'0060_H$	18_H / 24_D
Fast External Interrupt 1	CC9IR	CC9IE	CC9INT	$00'0064_H$	19_H / 25_D
Fast External Interrupt 2	CC10IR	CC10IE	CC10INT	$00'0068_H$	$1A_H$ / 26_D
Fast External Interrupt 3	CC11IR	CC11IE	CC11INT	$00'006C_H$	$1B_H$ / 27_D
Fast External Interrupt 4	CC12IR	CC12IE	CC12INT	$00'0070_H$	$1C_H$ / 28_D
Fast External Interrupt 5	CC13IR	CC13IE	CC13INT	$00'0074_H$	$1D_H$ / 29_D
Fast External Interrupt 6	CC14IR	CC14IE	CC14INT	$00'0078_H$	$1E_H$ / 30_D
Fast External Interrupt 7	CC15IR	CC15IE	CC15INT	$00'007C_H$	$1F_H$ / 31_D
GPT1 Timer 2	T2IR	T2IE	T2INT	$00'0088_H$	22_H / 34_D
GPT1 Timer 3	T3IR	T3IE	T3INT	$00'008C_H$	23_H / 35_D
GPT1 Timer 4	T4IR	T4IE	T4INT	$00'0090_H$	24_H / 36_D
GPT2 Timer 5	T5IR	T5IE	T5INT	$00'0094_H$	25_H / 37_D
GPT2 Timer 6	T6IR	T6IE	T6INT	$00'0098_H$	26_H / 38_D
GPT2 CAPREL Register	CRIR	CRIE	CRINT	$00'009C_H$	27_H / 39_D
A/D Conversion Complete	ADCIR	ADCIE	ADCINT	$00'00A0_H$	28_H / 40_D
A/D Overrun Error	ADEIR	ADEIE	ADEINT	$00'00A4_H$	29_H / 41_D
ASC0 Transmit	S0TIR	S0TIE	S0TINT	$00'00A8_H$	$2A_H$ / 42_D
ASC0 Transmit Buffer	S0TBIR	S0TBIE	S0TBINT	$00'011C_H$	47_H / 71_D
ASC0 Receive	S0RIR	S0RIE	S0RINT	$00'00AC_H$	$2B_H$ / 43_D
ASC0 Error	S0EIR	S0EIE	S0EINT	$00'00B0_H$	$2C_H$ / 44_D
SSC Transmit	SCTIR	SCTIE	SCTINT	$00'00B4_H$	$2D_H$ / 45_D
SSC Receive	SCRIR	SCRIE	SCRINT	$00'00B8_H$	$2E_H$ / 46_D
SSC Error	SCEIR	SCEIE	SCEINT	$00'00BC_H$	$2F_H$ / 47_D
I²C Data Transfer Event	XP0IR	XP0IE	XP0INT	$00'0100_H$	40_H / 64_D
I²C Protocol Event	XP1IR	XP1IE	XP1INT	$00'0104_H$	41_H / 65_D
X-Peripheral Node 2	XP2IR	XP2IE	XP2INT	$00'0108_H$	42_H / 66_D
RTC (via ISNC)	XP3IR	XP3IE	XP3INT	$00'010C_H$	43_H / 67_D

The table below lists the vector locations for hardware traps and the corresponding status flags in register TFR. It also lists the priorities of trap service for cases, where more than one trap condition might be detected within the same instruction. After any reset (hardware reset, software reset instruction SRST, or reset by watchdog timer overflow) program execution starts at the reset vector at location 00’0000$_H$. Reset conditions have priority over every other system activity and therefore have the highest priority (trap priority III).

Software traps may be initiated to any vector location between 00’0000$_H$ and 00’01FC$_H$. A service routine entered via a software TRAP instruction is always executed on the current CPU priority level which is indicated in bit field ILVL in register PSW. This means that routines entered via the software TRAP instruction can be interrupted by all hardware traps or higher level interrupt requests.

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

Normal Interrupt Processing and PEC Service

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

Interrupt System Register Description

Interrupt processing is controlled globally by register PSW through a general interrupt enable bit (IEN) and the CPU priority field (ILVL). Additionally the different interrupt sources are controlled individually by their specific interrupt control registers (… IC). Thus, the acceptance of requests by the CPU is determined by both the individual interrupt control registers and the PSW. PEC services are controlled by the respective PECCx register and the source and destination pointers, which specify the task of the respective PEC service channel.

Interrupt Control Registers

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

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

xxIC ($yyyy_H$ / zz_H) <SFR area> **Reset Value: - - 00_H**

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

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

The **Interrupt Request Flag** is set by hardware whenever a service request from the respective source occurs. It is cleared automatically upon entry into the interrupt service routine or upon a PEC service. In the case of PEC service the Interrupt Request flag remains set, if the COUNT field in register PECCx of the selected PEC channel decrements to zero. This allows a normal CPU interrupt to respond to a completed PEC block transfer.

Note: Modifying the Interrupt Request flag via software causes the same effects as if it had been set or cleared by hardware.

Interrupt Priority Level and Group Level

The four bits of bit field ILVL specify the priority level of a service request for the arbitration of simultaneous requests. The priority increases with the numerical value of ILVL, so 0000_B is the lowest and 1111_B is the highest priority level.

When more than one interrupt request on a specific level gets active at the same time, the values in the respective bit fields GLVL are used for second level arbitration to select one request for being serviced. Again the group priority increases with the numerical value of GLVL, so 00_B is the lowest and 11_B is the highest group priority.

Note: All interrupt request sources that are enabled and programmed to the same priority level must always be programmed to different group priorities. Otherwise an incorrect interrupt vector will be generated.

Upon entry into the interrupt service routine, the priority level of the source that won the arbitration and who's priority level is higher than the current CPU level, is copied into bit field ILVL of register PSW after pushing the old PSW contents on the stack.

The interrupt system of the C161RI allows nesting of up to 15 interrupt service routines of different priority levels (level 0 cannot be arbitrated).

Interrupt requests that are programmed to priority levels 15 or 14 (i.e., ILVL = $111X_B$) will be serviced by the PEC, unless the COUNT field of the associated PECC register contains zero. In this case the request will instead be serviced by normal interrupt processing. Interrupt requests that are programmed to priority levels 13 through 1 will always be serviced by normal interrupt processing.

Note: Priority level 0000_B is the default level of the CPU. Therefore a request on level 0 will never be serviced, because it can never interrupt the CPU. However, an enabled interrupt request on level 0000_B will terminate the C161RI's Idle mode and reactivate the CPU.

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

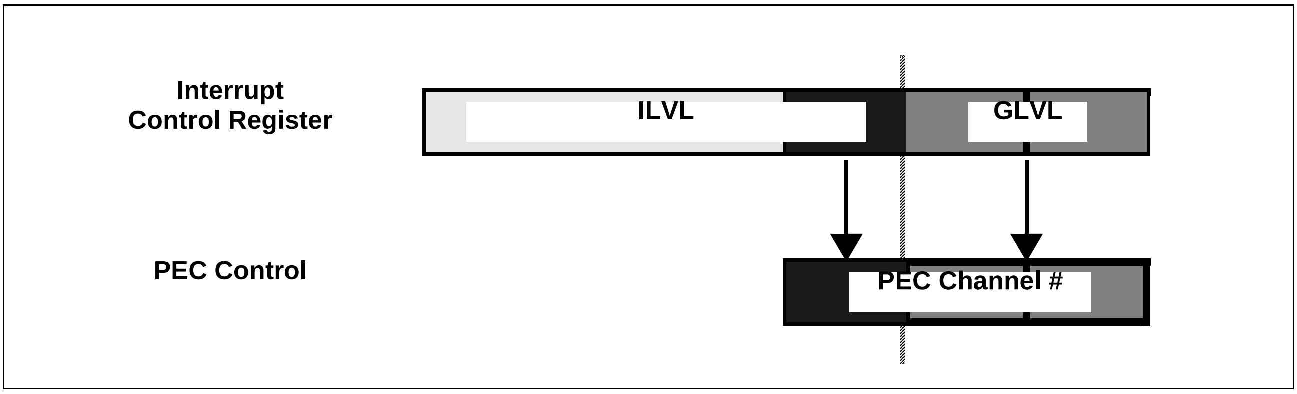


Figure 5-1
Priority Levels and PEC Channels

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

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

The table below shows in a few examples, which action is executed with a given programming of an interrupt control register.

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

Note: All requests on levels 13 … 1 cannot initiate PEC transfers. They are always serviced by an interrupt service routine. No PECC register is associated and no COUNT field is checked.

Interrupt Control Functions in the PSW

The Processor Status Word (PSW) is functionally divided into 2 parts: the lower byte of the PSW basically represents the arithmetic status of the CPU, the upper byte of the PSW controls the interrupt system of the C161RI and the arbitration mechanism for the external bus interface.

Note: Pipeline effects have to be considered when enabling/disabling interrupt requests via modifications of register PSW (see chapter "The Central Processing Unit").

PSW (FF10$_H$ / 88$_H$) SFR **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ILVL				**IEN**	**HLD EN**	**-**	**-**	**-**	**USR0**	**MUL IP**	**E**	**Z**	**V**	**C**	**N**
rw				rw	rw	-	-	-	rw	rw	rw	rw	rw	rw	rw

Bit	Function
N, C, V, Z, E, MULIP, USR0	**CPU status flags** (Described in section “The Central Processing Unit”) Define the current status of the CPU (ALU, multiplication unit).
HLDEN	**HOLD Enable** (Enables External Bus Arbitration) 0: Bus arbitration disabled, P6.7 … P6.5 may be used for general purpose IO 1: Bus arbitration enabled, P6.7 … P6.5 serve as $\overline{\text{BREQ}}$, $\overline{\text{HLDA}}$, $\overline{\text{HOLD}}$, resp.
ILVL	**CPU Priority Level** Defines the current priority level for the CPU F$_H$: Highest priority level 0$_H$: Lowest priority level
IEN	**Interrupt Enable Control Bit** (globally enables/disables interrupt requests) ‘0’: Interrupt requests are disabled ‘1’: Interrupt requests are enabled

CPU Priority ILVL defines the current level for the operation of the CPU. This bit field reflects the priority level of the routine that is currently executed. Upon the entry into an interrupt service routine this bit field is updated with the priority level of the request that is being serviced. The PSW is saved on the system stack before. The CPU level determines the minimum interrupt priority level that will be serviced. Any request on the same or a lower level will not be acknowledged.
The current CPU priority level may be adjusted via software to control which interrupt request sources will be acknowledged.

PEC transfers do not really interrupt the CPU, but rather “steal” a single cycle, so PEC services do not influence the ILVL field in the PSW.

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

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

Interrupt Enable bit IEN globally enables or disables PEC operation and the acceptance of interrupts by the CPU. When IEN is cleared, no new interrupt requests are accepted by the CPU. Requests that already have entered the pipeline at that time will process, however. When IEN is set to '1', all interrupt sources, which have been individually enabled by the interrupt enable bits in their associated control registers, are globally enabled.

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

5.2 Operation of the PEC Channels

The C161RI's Peripheral Event Controller (PEC) provides 8 PEC service channels, which move a single byte or word between two locations in segment 0 (data pages 3 … 0). This is the fastest possible interrupt response and in many cases is sufficient to service the respective peripheral request (e.g. serial channels, etc.). Each channel is controlled by a dedicated PEC Channel Counter/Control register (PECCx) and a pair of pointers for source (SRCPx) and destination (DSTPx) of the data transfer.

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

PECCx (FECy$_H$ / 6z$_H$, see table) SFR **Reset Value: 0000$_H$**

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

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

PEC Control Register Addresses

Register	Address	Reg. Space	Register	Address	Reg. Space
PECC0	FEC0$_H$ / 60$_H$	SFR	PECC4	FEC8$_H$ / 64$_H$	SFR
PECC1	FEC2$_H$ / 61$_H$	SFR	PECC5	FECA$_H$ / 65$_H$	SFR
PECC2	FEC4$_H$ / 62$_H$	SFR	PECC6	FECC$_H$ / 66$_H$	SFR
PECC3	FEC6$_H$ / 63$_H$	SFR	PECC7	FECE$_H$ / 67$_H$	SFR

Byte/Word Transfer bit BWT controls, if a byte or a word is moved during a PEC service cycle. This selection controls the transferred data size and the increment step for the modified pointer.

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

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

The PEC Transfer Count Field COUNT controls the action of a respective PEC channel, where the content of bit field COUNT at the time the request is activated selects the action. COUNT may allow a specified number of PEC transfers, unlimited transfers or no PEC service at all.

The table below summarizes, how the COUNT field itself, the interrupt requests flag IR and the PEC channel action depends on the previous content of COUNT.

Previous COUNT	Modified COUNT	IR after PEC service	Action of PEC Channel and Comments
FF_H	FF_H	‘0’	Move a Byte / Word Continuous transfer mode, i.e. COUNT is not modified
$FE_H..02_H$	$FD_H..01_H$	‘0’	Move a Byte / Word and decrement COUNT
01_H	00_H	‘1’	Move a Byte / Word Leave request flag set, which triggers another request
00_H	00_H	(‘1’)	**No action!** Activate interrupt service routine rather than PEC channel.

The PEC transfer counter allows to service a specified number of requests by the respective PEC channel, and then (when COUNT reaches 00_H) activate the interrupt service routine, which is associated with the priority level. After each PEC transfer the COUNT field is decremented and the request flag is cleared to indicate that the request has been serviced.

Continuous transfers are selected by the value FF_H in bit field COUNT. In this case COUNT is not modified and the respective PEC channel services any request until it is disabled again.

When COUNT is decremented from 01_H to 00_H after a transfer, the request flag is not cleared, which generates another request from the same source. When COUNT already contains the value 00_H, the respective PEC channel remains idle and the associated interrupt service routine is activated instead. This allows to choose, if a level 15 or 14 request is to be serviced by the PEC or by the interrupt service routine.

Note: PEC transfers are only executed, if their priority level is higher than the CPU level, i.e. only PEC channels 7 … 4 are processed, while the CPU executes on level 14.
All interrupt request sources that are enabled and programmed for PEC service should use different channels. Otherwise only one transfer will be performed for all simultaneous requests. When COUNT is decremented to 00_H, and the CPU is to be interrupted, an incorrect interrupt vector will be generated.

The source and destination pointers specify the locations between which the data is to be moved. A pair of pointers (SRCPx and DSTPx) is associated with each of the 8 PEC channels. These pointers do not reside in specific SFRs, but are mapped into the internal RAM of the C161RI just below the bit-addressable area (see figure below).

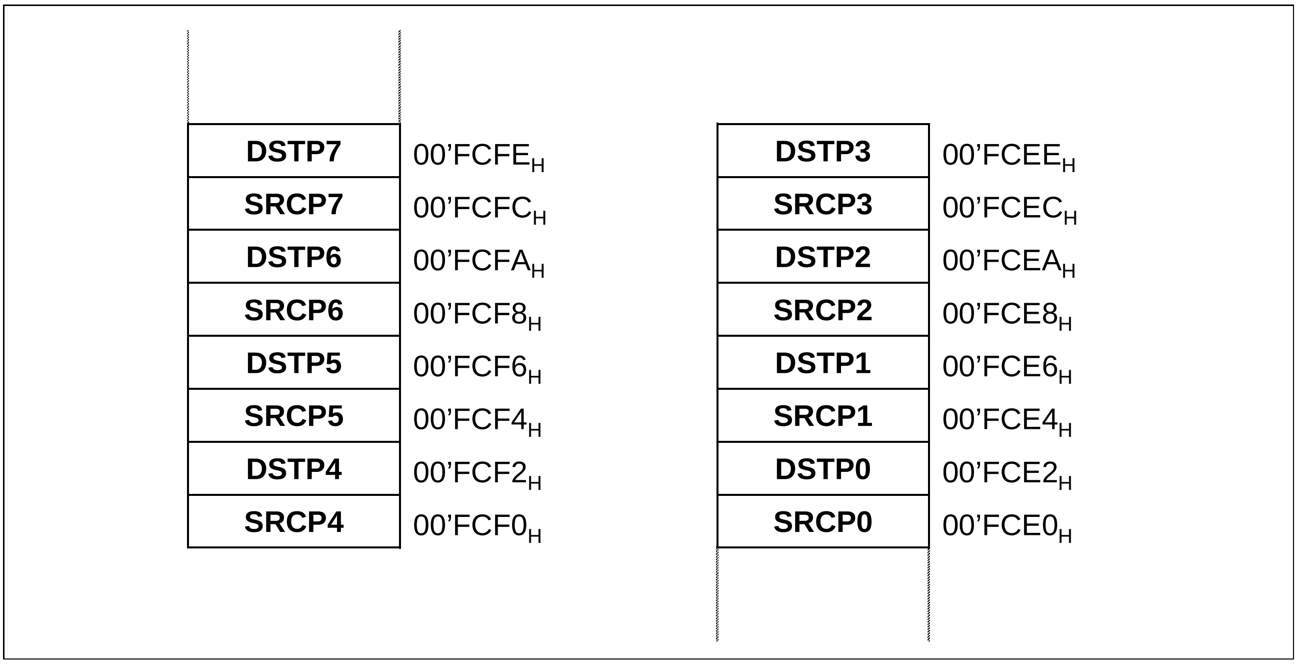


Figure 5-2
Mapping of PEC Pointers into the Internal RAM

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

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

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

5.3 Prioritization of Interrupt and PEC Service Requests

Interrupt and PEC service requests from all sources can be enabled, so they are arbitrated and serviced (if they win), or they may be disabled, so their requests are disregarded and not serviced.

Enabling and disabling interrupt requests may be done via three mechanisms:

Control Bits allow to switch each individual source “ON” or “OFF”, so it may generate a request or not. The control bits (xxIE) are located in the respective interrupt control registers. All interrupt requests may be enabled or disabled generally via bit IEN in register PSW. This control bit is the “main switch” that selects, if requests from any source are accepted or not.
For a specific request to be arbitrated the respective source’s enable bit and the global enable bit must both be set.

The Priority Level automatically selects a certain group of interrupt requests that will be acknowledged, disclosing all other requests. The priority level of the source that won the arbitration is compared against the CPU’s current level and the source is only serviced, if its level is higher than the current CPU level. Changing the CPU level to a specific value via software blocks all requests on the same or a lower level. An interrupt source that is assigned to level 0 will be disabled and never be serviced.

The ATOMIC and EXTend instructions automatically disable all interrupt requests for the duration of the following 1 … 4 instructions. This is useful e.g. for semaphore handling and does not require to re-enable the interrupt system after the unseparable instruction sequence (see chapter “System Programming”).

Interrupt Class Management

An interrupt class covers a set of interrupt sources with the same importance, i.e. the same priority from the system’s viewpoint. Interrupts of the same class must not interrupt each other. The C161RI supports this function with two features:

Classes with up to 4 members can be established by using the same interrupt priority (ILVL) and assigning a dedicated group level (GLVL) to each member. This functionality is built-in and handled automatically by the interrupt controller.

Classes with more than 4 members can be established by using a number of adjacent interrupt priorities (ILVL) and the respective group levels (4 per ILVL). Each interrupt service routine within this class sets the CPU level to the highest interrupt priority within the class. All requests from the same or any lower level are blocked now, i.e. no request of this class will be accepted.

The example below establishes 3 interrupt classes which cover 2 or 3 interrupt priorities, depending on the number of members in a class. A level 6 interrupt disables all other sources in class 2 by changing the current CPU level to 8, which is the highest priority (ILVL) in class 2. Class 1 requests or PEC requests are still serviced in this case.

The 24 interrupt sources (excluding PEC requests) are so assigned to 3 classes of priority rather than to 7 different levels, as the hardware support would do.

Software controlled Interrupt Classes (Example)

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

5.4 Saving the Status during Interrupt Service

Before an interrupt request that has been arbitrated is actually serviced, the status of the current task is automatically saved on the system stack. The CPU status (PSW) is saved along with the location, where the execution of the interrupted task is to be resumed after returning from the service routine. This return location is specified through the Instruction Pointer (IP) and, in case of a segmented memory model, the Code Segment Pointer (CSP). Bit SGTDIS in register SYSCON controls, how the return location is stored.

The system stack receives the PSW first, followed by the IP (unsegmented) or followed by CSP and then IP (segmented mode). This optimizes the usage of the system stack, if segmentation is disabled.

The CPU priority field (ILVL in PSW) is updated with the priority of the interrupt request that is to be serviced, so the CPU now executes on the new level. If a multiplication or division was in progress at the time the interrupt request was acknowledged, bit MULIP in register PSW is set to ‘1’. In this case the return location that is saved on the stack is not the next instruction in the instruction flow, but rather the multiply or divide instruction itself, as this instruction has been interrupted and will be completed after returning from the service routine.

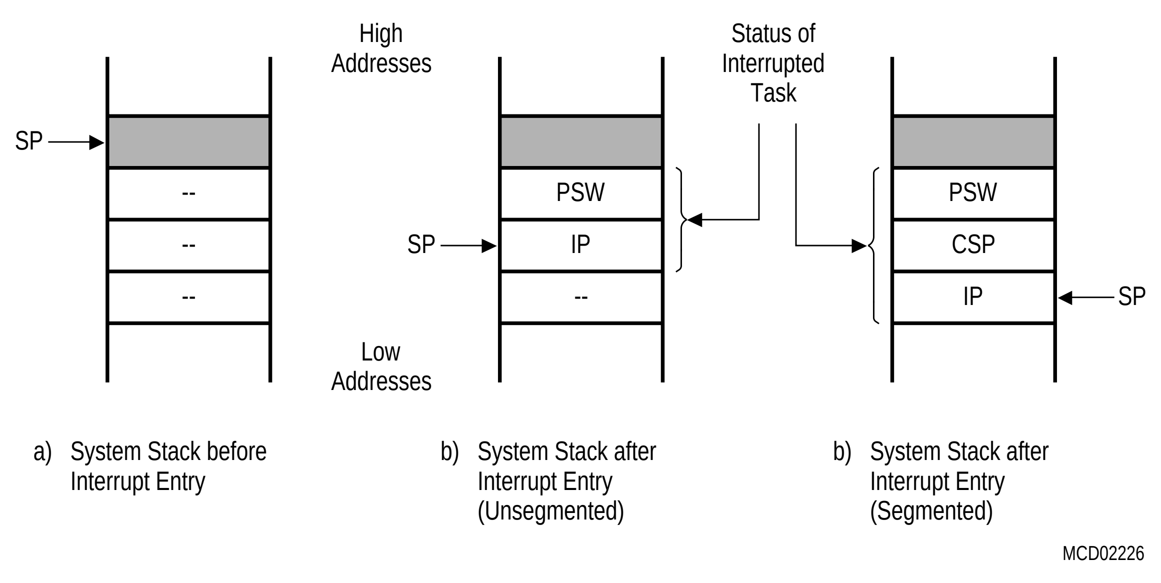


Figure 5-3
Task Status saved on the System Stack

The interrupt request flag of the source that is being serviced is cleared. The IP is loaded with the vector associated with the requesting source (the CSP is cleared in case of segmentation) and the first instruction of the service routine is fetched from the respective vector location, which is expected to branch to the service routine itself. The data page pointers and the context pointer are not affected.

When the interrupt service routine is left (RETI is executed), the status information is popped from the system stack in the reverse order, taking into account the value of bit SGTDIS.

Context Switching

An interrupt service routine usually saves all the registers it uses on the stack, and restores them before returning. The more registers a routine uses, the more time is wasted with saving and restoring. The C161RI allows to switch the complete bank of CPU registers (GPRs) with a single instruction, so the service routine executes within its own, separate context.

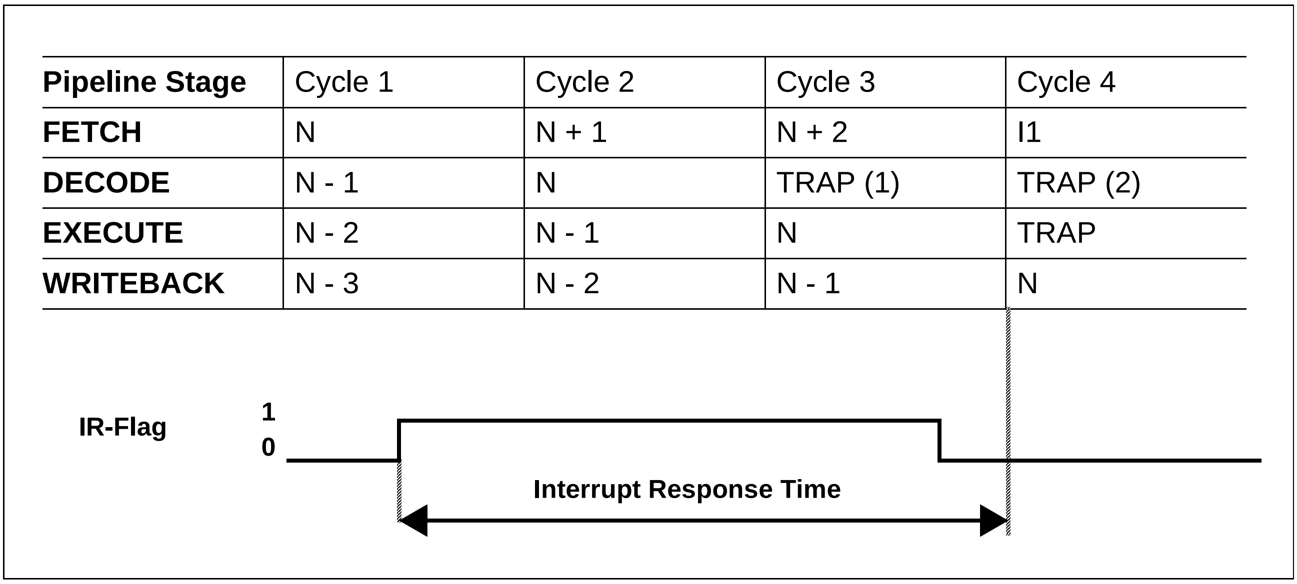
The instruction “SCXT CP, #New_Bank” pushes the content of the context pointer (CP) on the system stack and loads CP with the immediate value “New_Bank”, which selects a new register bank. The service routine may now use its “own registers”. This register bank is preserved, when the service routine terminates,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

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

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

5.5 Interrupt Response Times

The interrupt response time defines the time from an interrupt request flag of an enabled interrupt source being set until the first instruction (I1) being fetched from the interrupt vector location. The basic interrupt response time for the C161RI is 3 instruction cycles.



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

Figure 5-4
Pipeline Diagram for Interrupt Response Time

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

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

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

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

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

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

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

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

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

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

A few examples illustrate these delays:

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

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

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

PEC Response Times

The PEC response time defines the time from an interrupt request flag of an enabled interrupt source being set until the PEC data transfer being started. The basic PEC response time for the C161RI is 2 instruction cycles.

Pipeline Stage	Cycle 1	Cycle 2	Cycle 3	Cycle 4
FETCH	N	N + 1	N + 2	N + 2
DECODE	N - 1	N	PEC	N + 1
EXECUTE	N - 2	N - 1	N	PEC
WRITEBACK	N - 3	N - 2	N - 1	N

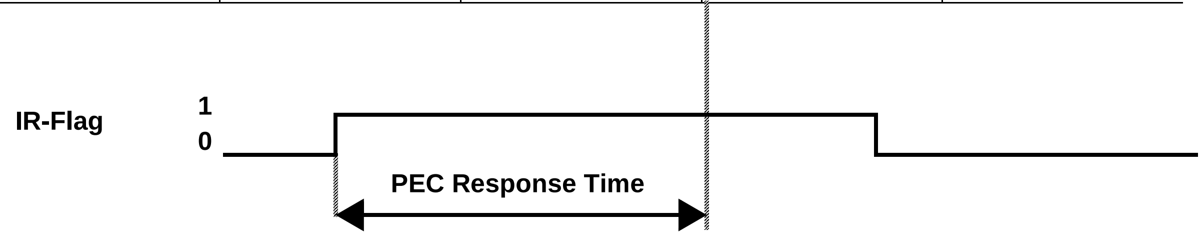


Figure 5-5
Pipeline Diagram for PEC Response Time

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

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

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

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

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

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

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

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

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

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

A few examples illustrate these delays:

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

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

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

Interrupt Node Sharing

Interrupt nodes may be shared between several module requests either if the requests are generated mutually exclusive or if the requests are generated at a low rate. If more than one source is enabled in this case the interrupt handler will first have to determine the requesting source. However, this overhead is not critical for low rate requests.

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

ISNC (F1DE$_H$ / EF$_H$) **ESFR** **Reset Value: 0000$_H$**

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

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

Sub-node Control Bit Allocation

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

Note: In order to ensure compatibility with other derivatives application software should never set reserved bits within register ISNC.
Bits PLLIE and PLLIR are not implemented, but reserved for compatibility with devices providing a PLL.

5.6 External Interrupts

Although the C161RI has no dedicated INTR input pins, it provides many possibilities to react on external asynchronous events by using a number of IO lines for interrupt input. The interrupt function may either be combined with the pin's main function or may be used instead of it, i.e. if the main pin function is not required.

Interrupt signals may be connected to:

- EX7IN … EX0IN, the fast external interrupt input pins,
- T4IN, T2IN, the timer input pins
- CAPIN, the capture input of GPT2

For each of these pins either a positive, a negative, or both a positive and a negative external transition can be selected to cause an interrupt or PEC service request. The edge selection is performed in the control register of the peripheral device associated with the respective port pin. The peripheral must be programmed to a specific operating mode to allow generation of an interrupt by the external signal. The priority of the interrupt request is determined by the interrupt control register of the respective peripheral interrupt source, and the interrupt vector of this source will be used to service the external interrupt request.

Note: In order to use any of the listed pins as external interrupt input, it must be switched to input mode via its direction control bit DPx.y in the respective port direction control register DPx.

Pins to be used as External Interrupt Inputs

Port Pin	Original Function	Control Register
P2.15-8/EX7-0IN	Fast external interrupt input pin	EXICON
P3.7/T2IN	Auxiliary timer T2 input pin	T2CON
P3.5/T4IN	Auxiliary timer T4 input pin	T4CON
P3.2/CAPIN	GPT2 capture input pin	T5CON

Pins T2IN or T4IN can be used as external interrupt input pins when the associated auxiliary timer T2 or T4 in block GPT1 is configured for capture mode. This mode is selected by programming the mode control fields T2M or T4M in control registers T2CON or T4CON to 101_B. The active edge of the external input signal is determined by bit fields T2I or T4I. When these fields are programmed to $X01_B$, interrupt request flags T2IR or T4IR in registers T2IC or T4IC will be set on a positive external transition at pins T2IN or T4IN, respectively. When T2I or T4I are programmed to $X10_B$, then a negative external transition will set the corresponding request flag. When T2I or T4I are programmed to $X11_B$, both a positive and a negative transition will set the request flag. In all three cases, the contents of the core timer T3 will be captured into the auxiliary timer registers T2 or T4 based on the transition at pins T2IN or T4IN. When the interrupt enable bits T2IE or T4IE are set, a PEC request or an interrupt request for vector T2INT or T4INT will be generated.

Pin CAPIN differs slightly from the timer input pins as it can be used as external interrupt input pin without affecting peripheral functions. When the capture mode enable bit T5SC in register T5CON is cleared to ’0’, signal transitions on pin CAPIN will only set the interrupt request flag CRIR in register CRIC, and the capture function of register CAPREL is not activated.

So register CAPREL can still be used as reload register for GPT2 timer T5, while pin CAPIN serves as external interrupt input. Bit field CI in register T5CON selects the effective transition of the external interrupt input signal. When CI is programmed to 01_B, a positive external transition will set the interrupt request flag. CI = 10_B selects a negative transition to set the interrupt request flag, and with CI = 11_B, both a positive and a negative transition will set the request flag. When the interrupt enable bit CRIE is set, an interrupt request for vector CRINT or a PEC request will be generated.

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

Fast External Interrupts

The input pins that may be used for external interrupts are sampled every 16 TCL, i.e. external events are scanned and detected in timeframes of 16 TCL. The C161RI provides 8 interrupt inputs that are sampled every 2 TCL, so external events are captured faster than with standard interrupt inputs.

The8 pins of Port 2 (P2.15-P2.8) can individually be programmed to this fast interrupt mode, where also the trigger transition (rising, falling or both) can be selected. The External Interrupt Control register EXICON controls this feature for all 8 pins.

EXICON (F1C0$_H$ / E0$_H$) **ESFR** **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXI7ES		EXI6ES		EXI5ES		EXI4ES		EXI3ES		EXI2ES		EXI1ES		EXI0ES	
rw		rw		rw		rw		rw		rw		rw		rw	

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

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

The interrupt control registers listed below (CC15IC … CC8IC) control the fast external interrupts of the C161RI. These fast external interrupt nodes and vectors are named according to the C167's CAPCOM channels CC15 … CC8, so interrupt nodes receive equal names throughout the architecture. See register description below.

CCxIC (See Table) SFR **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								CCx IR	**CCx IE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

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

Fast External Interrupt Control Register Addresses

Register	Address	External Interrupt
CC8IC	FF88$_H$ / C4$_H$	EX0IN
CC9IC	FF8A$_H$ / C5$_H$	EX1IN
CC10IC	FF8C$_H$ / C6$_H$	EX2IN
CC11IC	FF8E$_H$ / C7$_H$	EX3IN
CC12IC	FF90$_H$ / C8$_H$	EX4IN
CC13IC	FF92$_H$ / C9$_H$	EX5IN
CC14IC	FF94$_H$ / CA$_H$	EX6IN
CC15IC	FF96$_H$ / CB$_H$	EX7IN

5.7 Trap Functions

Traps interrupt the current execution similar to standard interrupts. However, trap functions offer the possibility to bypass the interrupt system's prioritization process in cases where immediate system reaction is required. Trap functions are not maskable and always have priority over interrupt requests on any priority level.

The C161RI provides two different kinds of trapping mechanisms. **Hardware traps** are triggered by events that occur during program execution (e.g. illegal access or undefined opcode), **software traps** are initiated via an instruction within the current execution flow.

Software Traps

The TRAP instruction is used to cause a software call to an interrupt service routine. The trap number that is specified in the operand field of the trap instruction determines which vector location in the address range from $00'0000_H$ through $00'01FC_H$ will be branched to.

Executing a TRAP instruction causes a similar effect as if an interrupt at the same vector had occurred. PSW, CSP (in segmentation mode), and IP are pushed on the internal system stack and a jump is taken to the specified vector location. When segmentation is enabled and a trap is executed, the CSP for the trap service routine is set to code segment 0. No Interrupt Request flags are affected by the TRAP instruction. The interrupt service routine called by a TRAP instruction must be terminated with a RETI (return from interrupt) instruction to ensure correct operation.

Note: The CPU level in register PSW is not modified by the TRAP instruction, so the service routine is executed on the same priority level from which it was invoked. Therefore, the service routine entered by the TRAP instruction can be interrupted by other traps or higher priority interrupts, other than when triggered by a hardware trap.

Hardware Traps

Hardware traps are issued by faults or specific system states that occur during runtime of a program (not identified at assembly time). A hardware trap may also be triggered intentionally, e.g. to emulate additional instructions by generating an Illegal Opcode trap. The C161RI distinguishes eight different hardware trap functions. When a hardware trap condition has been detected, the CPU branches to the trap vector location for the respective trap condition. Depending on the trap condition, the instruction which caused the trap is either completed or cancelled (i.e. it has no effect on the system state) before the trap handling routine is entered.

Hardware traps are non-maskable and always have priority over every other CPU activity. If several hardware trap conditions are detected within the same instruction cycle, the highest priority trap is serviced (see table in section "Interrupt System Structure").

PSW, CSP (in segmentation mode), and IP are pushed on the internal system stack and the CPU level in register PSW is set to the highest possible priority level (i.e. level 15), disabling all interrupts. The CSP is set to code segment zero, if segmentation is enabled. A trap service routine must be terminated with the RETI instruction.

The eight hardware trap functions of the C161RI are divided into two classes:

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

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

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

These traps share the same trap priority, and the same vector address.

The bit-addressable Trap Flag Register (TFR) allows a trap service routine to identify the kind of trap which caused the exception. Each trap function is indicated by a separate request flag. When a hardware trap occurs, the corresponding request flag in register TFR is set to '1'.

TFR (FFAC_H / D6_H) SFR **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NMI	**STK OF**	**STK UF**	**-**	**-**	**-**	**-**	**-**	**UND OPC**	**-**	**-**	**-**	**PRT FLT**	**ILL OPA**	**ILL INA**	**ILL BUS**
rw	rw	rw	-	-	-	-	-	rw	-	-	-	rw	rw	rw	rw

Bit	Function
ILLBUS	**Illegal External Bus Access Flag** An external access has been attempted with no external bus defined.
ILLINA	**Illegal Instruction Access Flag** A branch to an odd address has been attempted.
ILLOPA	**Illegal Word Operand Access Flag** A word operand access (read or write) to an odd address has been attempted.
PRTFLT	**Protection Fault Flag** A protected instruction with an illegal format has been detected.
UNDOPC	**Undefined Opcode Flag** The currently decoded instruction has no valid C161RI opcode.
STKUF	**Stack Underflow Flag** The current stack pointer value exceeds the content of register STKUN.
STKOF	**Stack Overflow Flag** The current stack pointer value falls below the content of register STKOV.
NMI	**Non Maskable Interrupt Flag** A negative transition (falling edge) has been detected on pin $\overline{\text{NMI}}$.

Note: The trap service routine must clear the respective trap flag, otherwise a new trap will be requested after exiting the service routine. Setting a trap request flag by software causes the same effects as if it had been set by hardware.

The reset functions (hardware, software, watchdog) may be regarded as a type of trap. Reset functions have the highest system priority (trap priority III).

Class A traps have the second highest priority (trap priority II), on the 3rd rank are class B traps, so a class A trap can interrupt a class B trap. If more than one class A trap occur at a time, they are prioritized internally, with the NMI trap on the highest and the stack underflow trap on the lowest priority.

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

A class A trap occurring during the execution of a class B trap service routine will be serviced immediately. During the execution of a class A trap service routine, however, any class B trap occurring will not be serviced until the class A trap service routine is exited with a RETI instruction. In this case, the occurrence of the class B trap condition is stored in the TFR register, but the IP value of the instruction which caused this trap is lost.

In the case where e.g. an Undefined Opcode trap (class B) occurs simultaneously with an NMI trap (class A), both the NMI and the UNDOPC flag is set, the IP of the instruction with the undefined opcode is pushed onto the system stack, but the NMI trap is executed. After return from the NMI service routine, the IP is popped from the stack and immediately pushed again because of the pending UNDOPC trap.

External NMI Trap

Whenever a high to low transition on the dedicated external $\overline{\text{NMI}}$ pin (Non-Maskable Interrupt) is detected, the NMI flag in register TFR is set and the CPU will enter the NMI trap routine. The IP value pushed on the system stack is the address of the instruction following the one after which normal processing was interrupted by the NMI trap.

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

Stack Overflow Trap

Whenever the stack pointer is decremented to a value which is less than the value in the stack overflow register STKOV, the STKOF flag in register TFR is set and the CPU will enter the stack overflow trap routine. Which IP value will be pushed onto the system stack depends on which operation caused the decrement of the SP. When an implicit decrement of the SP is made through a PUSH or CALL instruction, or upon interrupt or trap entry, the IP value pushed is the address of the following instruction. When the SP is decremented by a subtract instruction, the IP value pushed represents the address of the instruction after the instruction following the subtract instruction.

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

Stack Underflow Trap

Whenever the stack pointer is incremented to a value which is greater than the value in the stack underflow register STKUN, the STKUF flag is set in register TFR and the CPU will enter the stack underflow trap routine. Again, which IP value will be pushed onto the system stack depends on which operation caused the increment of the SP. When an implicit increment of the SP is made through a POP or return instruction, the IP value pushed is the address of the following instruction. When the SP is incremented by an add instruction, the pushed IP value represents the address of the instruction after the instruction following the add instruction.

Undefined Opcode Trap

When the instruction currently decoded by the CPU does not contain a valid C161RI opcode, the UNDOPC flag is set in register TFR and the CPU enters the undefined opcode trap routine. The IP value pushed onto the system stack is the address of the instruction that caused the trap.

This can be used to emulate unimplemented instructions. The trap service routine can examine the faulting instruction to decode operands for unimplemented opcodes based on the stacked IP. In order to resume processing, the stacked IP value must be incremented by the size of the undefined instruction, which is determined by the user, before a RETI instruction is executed.

Protection Fault Trap

Whenever one of the special protected instructions is executed where the opcode of that instruction is not repeated twice in the second word of the instruction and the byte following the opcode is not the complement of the opcode, the PRTFLT flag in register TFR is set and the CPU enters the protection fault trap routine. The protected instructions include DISWDT, EINIT, IDLE, PWRDN, SRST, and SRVWDT. The IP value pushed onto the system stack for the protection fault trap is the address of the instruction that caused the trap.

Illegal Word Operand Access Trap

Whenever a word operand read or write access is attempted to an odd byte address, the ILLOPA flag in register TFR is set and the CPU enters the illegal word operand access trap routine. The IP value pushed onto the system stack is the address of the instruction following the one which caused the trap.

Illegal Instruction Access Trap

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

Illegal External Bus Access Trap

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

6 Clock Generation

All activities of the C161RI’s controller hardware and its on-chip peripherals are controlled via the system clock signal f_{CPU}.
This reference clock is generated in three stages (see also figure below):

- **Oscillator**
The on-chip Pierce oscillator can either run with an external crystal and appropriate oscillator circuitry or it can be driven by an external oscillator.

- **Frequency Control**
The input clock signal feeds the controller hardware …
… directly, providing phase coupled operation on not too high input frequency
… divided by 2 in order to get 50% duty cycle clock signal
… via the Slow Down Divider (SDD) in order to reduce the power consumption.
The resulting internal clock signal is referred to as “CPU clock” f_{CPU}.

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

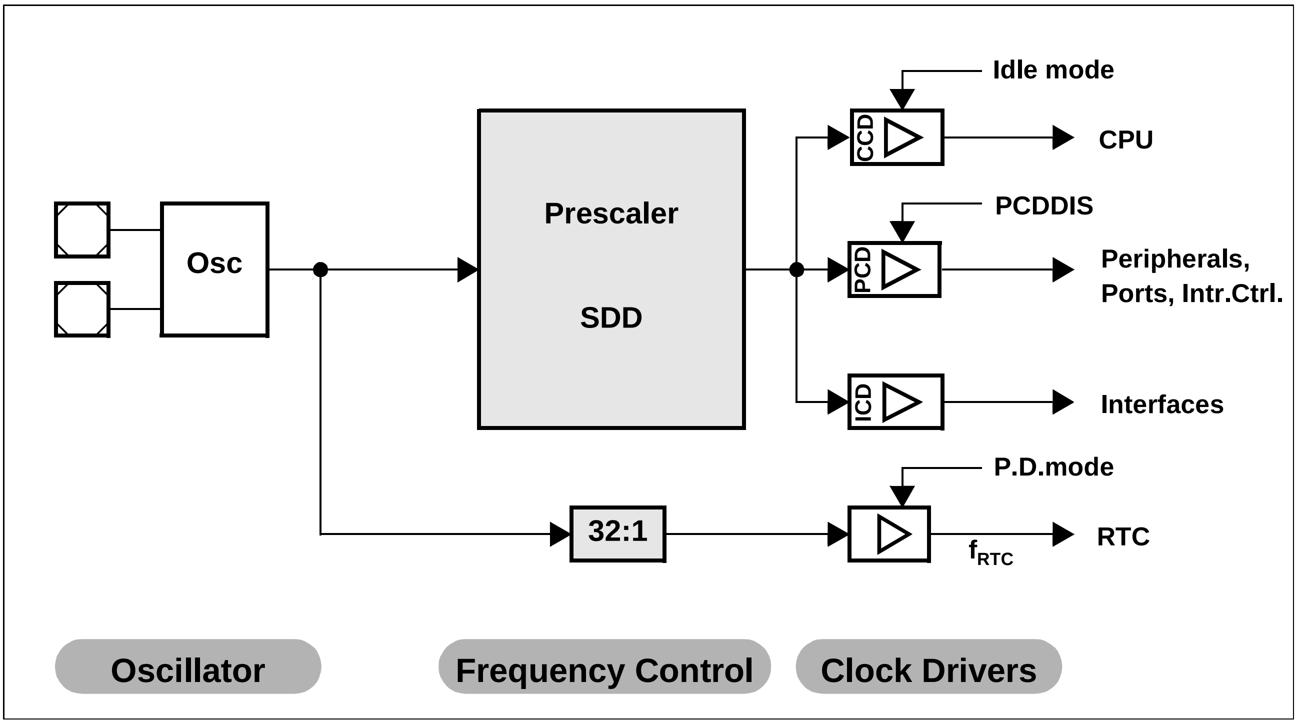


Figure 6-1
CPU Clock Generation Stages

6.1 Oscillator

The main oscillator of the C161RI is a power optimized Pierce oscillator providing an inverter and a feedback element. Pins XTAL1 and XTAL2 connect the inverter to the external crystal. The standard external oscillator circuitry (see figure below) comprises the crystal, two low end capacitors and series resistor (Rx2) to limit the current through the crystal. The additional LC combination is only required for 3rd overtone crystals to suppress oscillation in the fundamental mode. A test resistor (R_Q) may be temporarily inserted to measure the oscillation allowance of the oscillator circuitry.

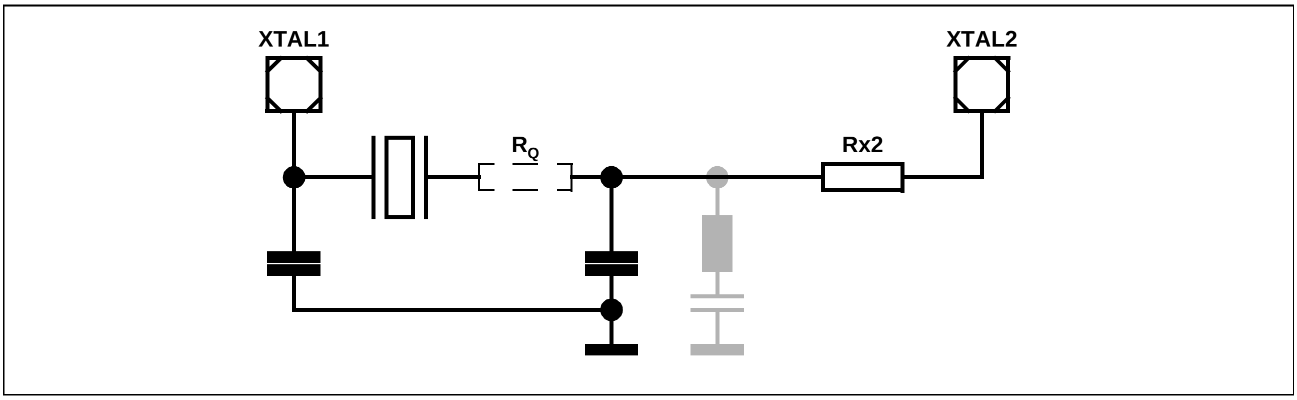


Figure 6-2
External Oscillator Circuitry

The on-chip oscillator is optimized for an input frequency range of 1 to 16 MHz.

An external clock signal (e.g. from an external oscillator or from a master device) may be fed to the input XTAL1. The Pierce oscillator then is not required to support the oscillation itself but is rather driven by the input signal. In this case the input frequency range may be 0 to 50 MHz (please note that the maximum applicable input frequency is limited by the device's maximum CPU frequency). For input frequencies above 25 … 30 MHz the oscillator's output should be terminated as shown in the figure below, at lower frequencies it may be left open. This termination improves the operation of the oscillator by filtering out frequencies above the intended oscillator frequency.

Figure 6-3
Oscillator Output Termination

Note: It is strongly recommended to measure the oscillation allowance (or margin) in the final target system (layout) to determine the optimum parameters for the oscillator operation.
The external circuitry is different from that required by previous derivatives.

6.2 Frequency Control

The CPU clock is generated from the oscillator clock in either of two software selectable ways:

The basic clock is the standard operating clock for the C161RI and is required to deliver the intended maximum performance. The configuration via PORT0 (CLKCFG) after a long hardware reset determines one of two possible basic clock generation modes:

- Direct Drive: the oscillator clock is directly fed to the controller hardware.
- Prescaler: the oscillator clock is divided by 2 to achieve a 50% duty cycle.

The Slow Down clock is the oscillator clock divided by a programmable factor of 1 … 32. This alternate possibility runs the C161RI at a lower frequency (depending on the programmed slow down factor) and thus greatly reduces its power consumption.

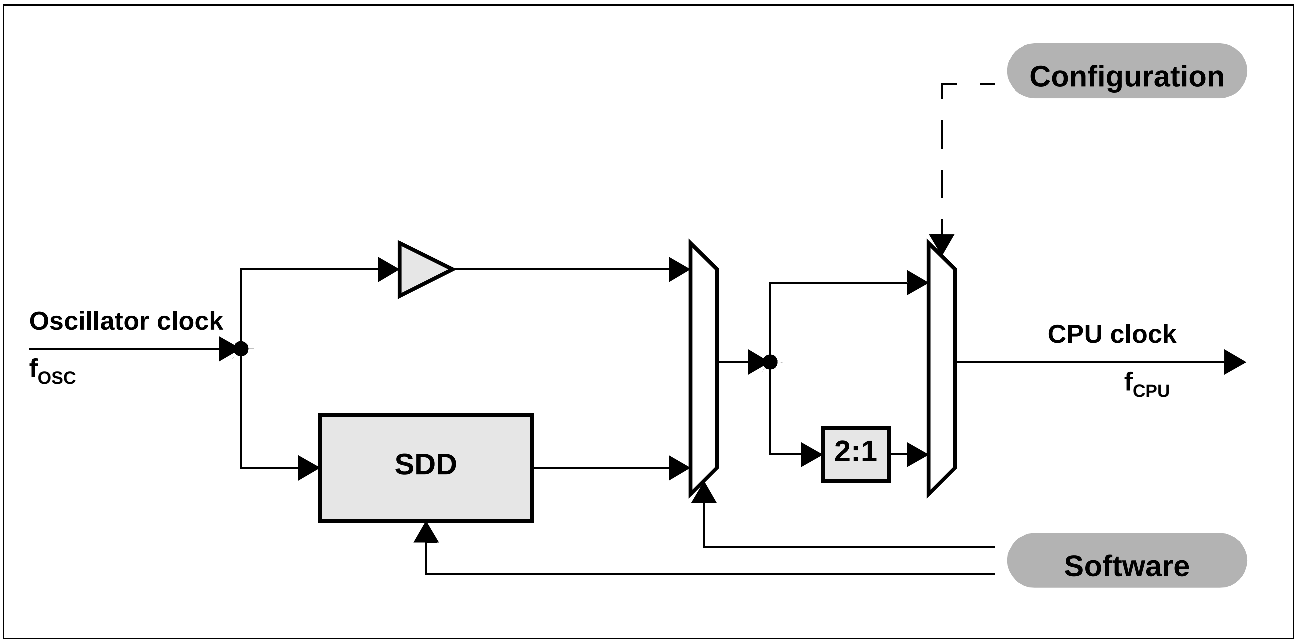


Figure 6-4
Frequency Control Paths

The internal operation of the C161RI is controlled by the internal CPU clock f_{CPU}. Both edges of the CPU clock can trigger internal (e.g. pipeline) or external (e.g. bus cycles) operations (see figure below).

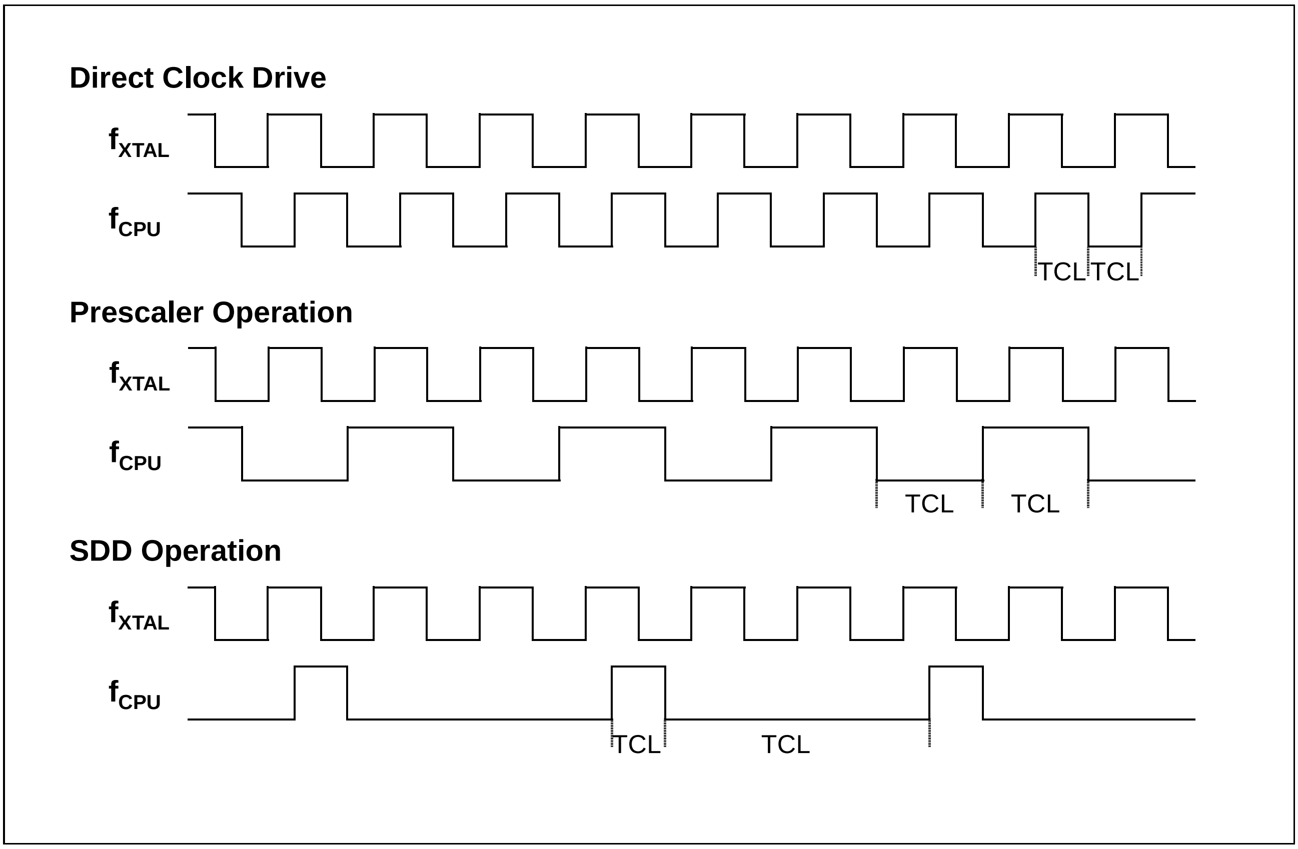


Figure 6-5
Generation Mechanisms for the CPU Clock

Direct Drive

When direct drive is configured (CLKCFG = ’011’) the C161RI’s clock system is directly fed from the external clock input, i.e. $f_{CPU} = f_{OSC}$. This allows operation of the C161RI with a reasonably small fundamental mode crystal. The specified minimum values for the CPU clock phases (TCLs) must be respected. Therefore the maximum input clock frequency depends on the clock signal’s duty cycle.

Prescaler Operation

When prescaler operation is configured (CLKCFG = ’001’) the C161RI’s input clock is divided by 2 to generate then CPU clock signal, i.e. $f_{CPU} = f_{OSC}/2$. This requires the oscillator (or input clock) to run on 2 times the intended operating frequency but guarantees a 50% duty cycle for the internal clock system independent of the input clock signal’s waveform.

The table below lists the possible selections.

C161RI Clock Generation Modes

P0.15-13 (P0H.7-5)	CPU Frequency $f_{CPU} = f_{OSC} \times F$	External Clock Input Range	Notes
1 1 1	*Reserved*		Default configuration
1 1 0	*Reserved*		
1 0 1	*Reserved*		
1 0 0	*Reserved*		
0 1 1	$f_{XTAL} \times 1$	1 to 20 MHz	Direct drive [1]
0 1 0	*Reserved*		
0 0 1	f_{XTAL} / 2	2 to 40 MHz	Prescaler operation
0 0 0	*Reserved*		

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

6.3 Clock Drivers

The operating clock signal f_{CPU} is distributed to the controller hardware via several clock drivers which are disabled under certain circumstances. The real time clock RTC is clocked via a separate clock driver which delivers the prescaled oscillator clock (contrary to the other clock drivers). The table below summarizes the different clock drivers and their function, especially in power reduction modes:

Clock Drivers Description

Clock Driver	Clock Signal	Active mode	Idle mode	P. Down mode	Connected Circuitry
CCD CPU Clock Driver	f_{CPU}	ON	Off	Off	CPU, memory modules
ICD Interface Clock Driver	f_{CPU}	ON	ON	Off	ASC0, WDT, SSC, interrupt detection circuitry
PCD Peripheral Clock Driver	f_{CPU}	Control via PCDDIS	Control via PCDDIS	Off	(X)Peripherals (timers, etc.) except ICD, interrupt controller, ports
RCD RTC Clock Driver	f_{OSC}	ON	ON	Control via PDCON	Realtime clock

Note: Disabling PCD by setting bit PCDDIS stops the clock signal for all connected modules. Make sure that all these modules are in a safe state before stopping their clock signal.
The port input and output values will not change while PCD is disabled,
CLKOUT will be high if enabled.
Please also respect the hints given in section “Flexible Peripheral Management” of chapter “Power Management”.

7 Parallel Ports

In order to accept or generate single external control signals or parallel data, the C161RI provides up to 76 parallel IO lines organized into six 8-bit IO ports (PORT0 made of P0H and P0L, PORT1 made of P1H and P1L, Port 2, Port 6), one 15-bit IO port (Port 3), one 7-bit IO port (Port 4) and one 6-bit input port (Port 5).

These port lines may be used for general purpose Input/Output controlled via software or may be used implicitly by C161RI'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 (except Port 5, of course). The IO ports are true bidirectional ports which are switched to high impedance state when configured as inputs. The output drivers of three IO ports (2, 3, 6) can be configured (pin by pin) for push/pull operation or open-drain operation via control registers.

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

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

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

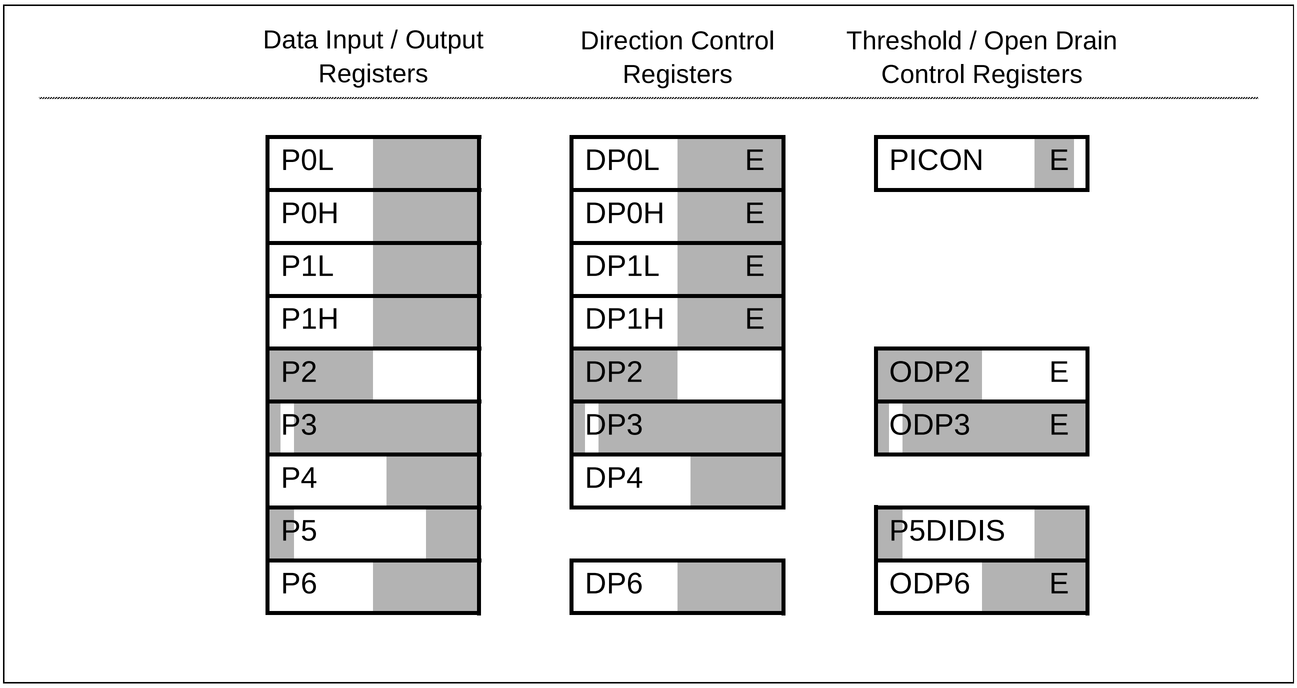


Figure 7-1
SFRs and Pins associated with the Parallel Ports

Open Drain Mode

In the C161RI certain ports provide Open Drain Control, which allows to switch the output driver of a port pin from a push/pull configuration to an open drain configuration. In push/pull mode a port output driver has an upper and a lower transistor, thus it can actively drive the line either to a high or a low level. In open drain mode the upper transistor is always switched off, and the output driver can only actively drive the line to a low level. When writing a ‘1’ to the port latch, the lower transistor is switched off and the output enters a high-impedance state. The high level must then be provided by an external pullup device. With this feature, it is possible to connect several port pins together to a Wired-AND configuration, saving external glue logic and/or additional software overhead for enabling/disabling output signals.

This feature is controlled through the respective Open Drain Control Registers ODPx which are provided for each port that has this feature implemented. These registers allow the individual bit-wise selection of the open drain mode for each port line. If the respective control bit ODPx.y is ‘0’ (default after reset), the output driver is in the push/pull mode. If ODPx.y is ‘1’, the open drain configuration is selected. Note that all ODPx registers are located in the ESFR space.

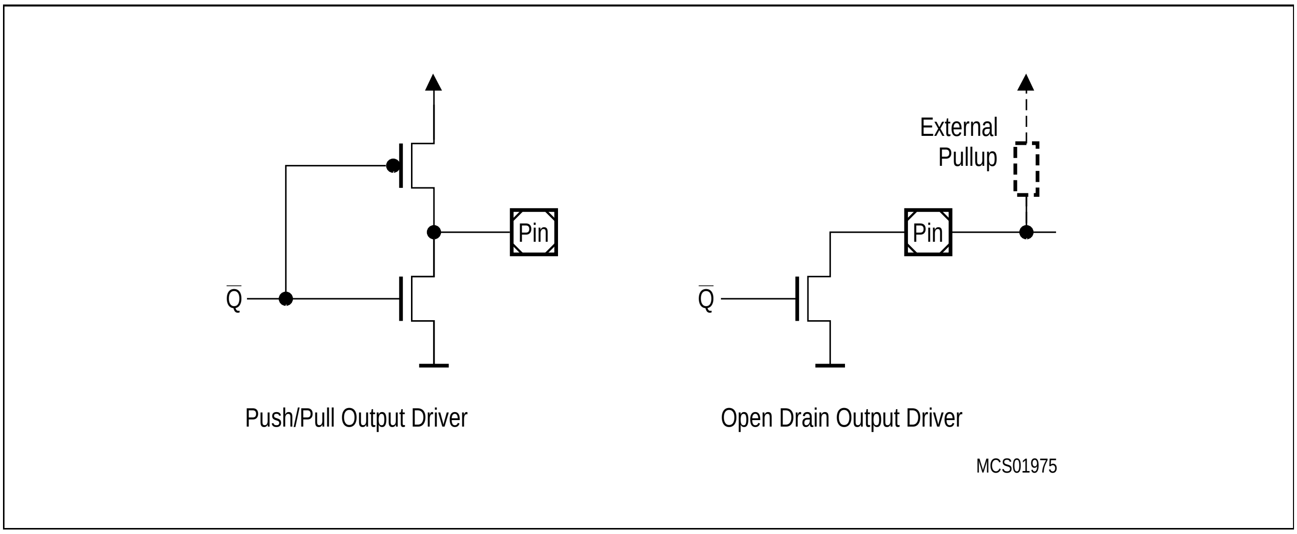


Figure 7-2
Output Drivers in Push/Pull Mode and in Open Drain Mode

Input Threshold Control

The standard inputs of the C161RI determine the status of input signals according to TTL levels. In order to accept and recognize noisy signals, CMOS-like input thresholds can be selected instead of the standard TTL thresholds for all pins of specific ports. These special thresholds are defined above the TTL thresholds and feature a defined hysteresis to prevent the inputs from toggling while the respective input signal level is near the thresholds.

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

PICON (F1C4$_H$ / E2$_H$) **ESFR** **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								-	-	-	-	P3HIN	P3LIN	P2HIN	-
-	-	-	-	-	-	-	-	-	-	-	-	rw	rw	rw	-

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

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

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

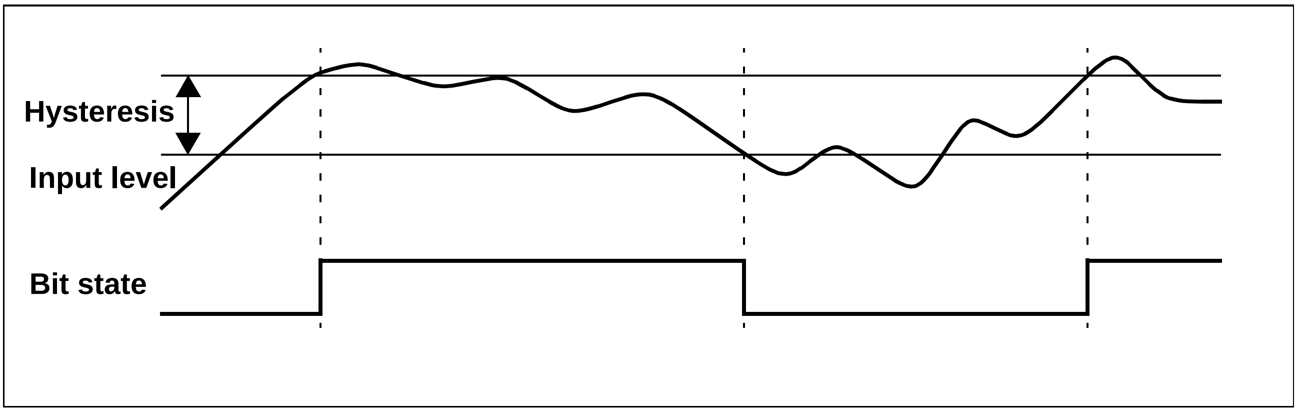


Figure 7-3
Hysteresis for Special Input Thresholds

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.

Alternate Port Functions

Port	Alternate Function(s)
PORT0	Address and data lines when accessing external resources (e.g. memory)
PORT1	Address lines when accessing external resources (e.g. memory)
Port 2	Fast external interrupt inputs
Port 3	Input/output functions of timers, serial interfaces Optional bus control signal $\overline{\text{BHE}}$/$\overline{\text{WRH}}$ and system clock output (CLKOUT)
Port 4	Additional selected segment address bits Axx … A16 in systems where more than 64 KBytes of memory are to be accessed directly
Port 5	Analog input channels to the A/D converter Timer control signal inputs
Port 6	Chip select output signals and I²C interface lines

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 latch 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 latch. Thus, the alternate input function reads the value stored in the port output latch. This can be used for testing purposes to allow a software trigger of an alternate input function by writing to the port output latch.

On most of the port lines, the user software is responsible for setting the proper direction when using an alternate input or output function of a pin. This is done by setting or clearing the direction control bit DPx.y of the pin before enabling the alternate function. There are port lines, however, 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 latches check how the alternate data output is combined with the respective port latch 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 latch prior to enabling the output drivers, in order to avoid undesired transitions on the output pins. This applies to single pins as well as to pin groups (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 “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.

7.1 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 (FF00$_H$ / 80$_H$) SFR **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 (FF02$_H$ / 81$_H$) SFR **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 (F100$_H$ / 80$_H$) **ESFR** **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 (F102$_H$ / 81$_H$) **ESFR** **Reset Value: - - 00$_H$**

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

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

Alternate Functions of PORT0

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

PORT0 is also used to select the system startup configuration. During reset, PORT0 is configured to input, and each line is held high through an internal pullup device. Each line can now be individually pulled to a low level (see DC-level specifications in the respective Data Sheets) through an external pulldown device. A default configuration is selected when the respective PORT0 lines are at a high level. Through pulling individual lines to a low level, this default can be changed according to the needs of the applications.
The internal pullup devices are designed such that an external pulldown resistors (see Data Sheet specification) can be used to apply a correct low level. These external pulldown resistors can remain connected to the PORT0 pins also during normal operation, however, care has to be taken such that they do not disturb the normal function of PORT0 (this might be the case, for example, if the external resistor is too strong).
With the end of reset, the selected bus configuration will be written to the BUSCON0 register. The configuration of the high byte of PORT0 will be copied into the special register RP0H. This read-only register holds the selection for the number of chip selects and segment addresses. Software can read this register in order to react according to the selected configuration, if required.
When the reset is terminated, the internal pullup devices are switched off, and PORT0 will be switched to the appropriate operating mode.

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

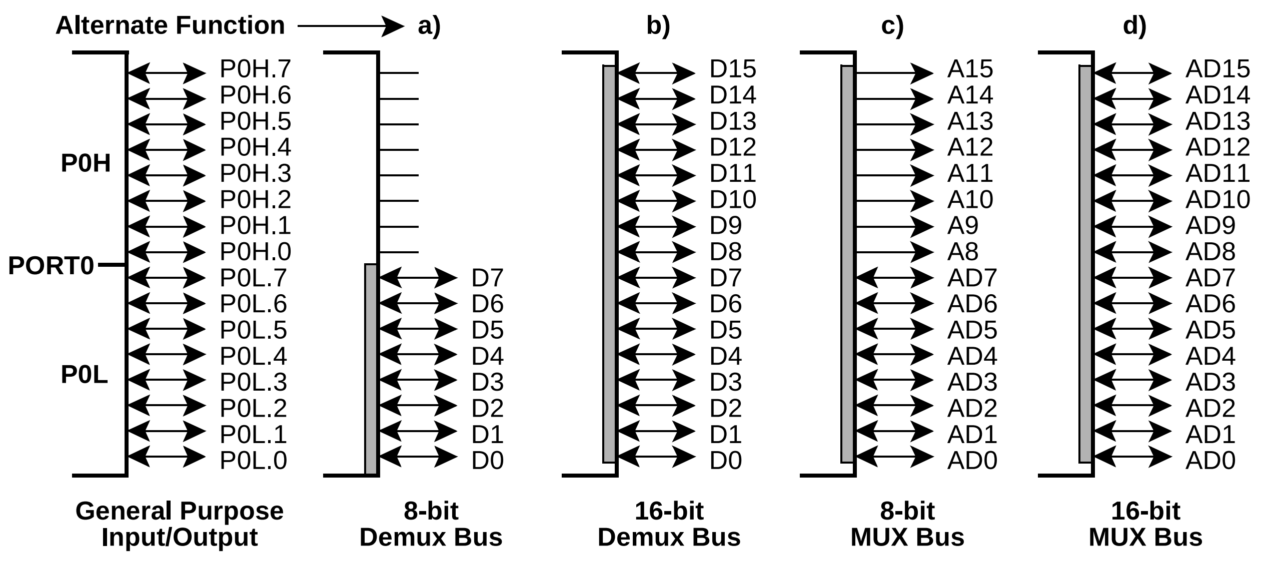


Figure 7-4
PORT0 IO and Alternate Functions

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.

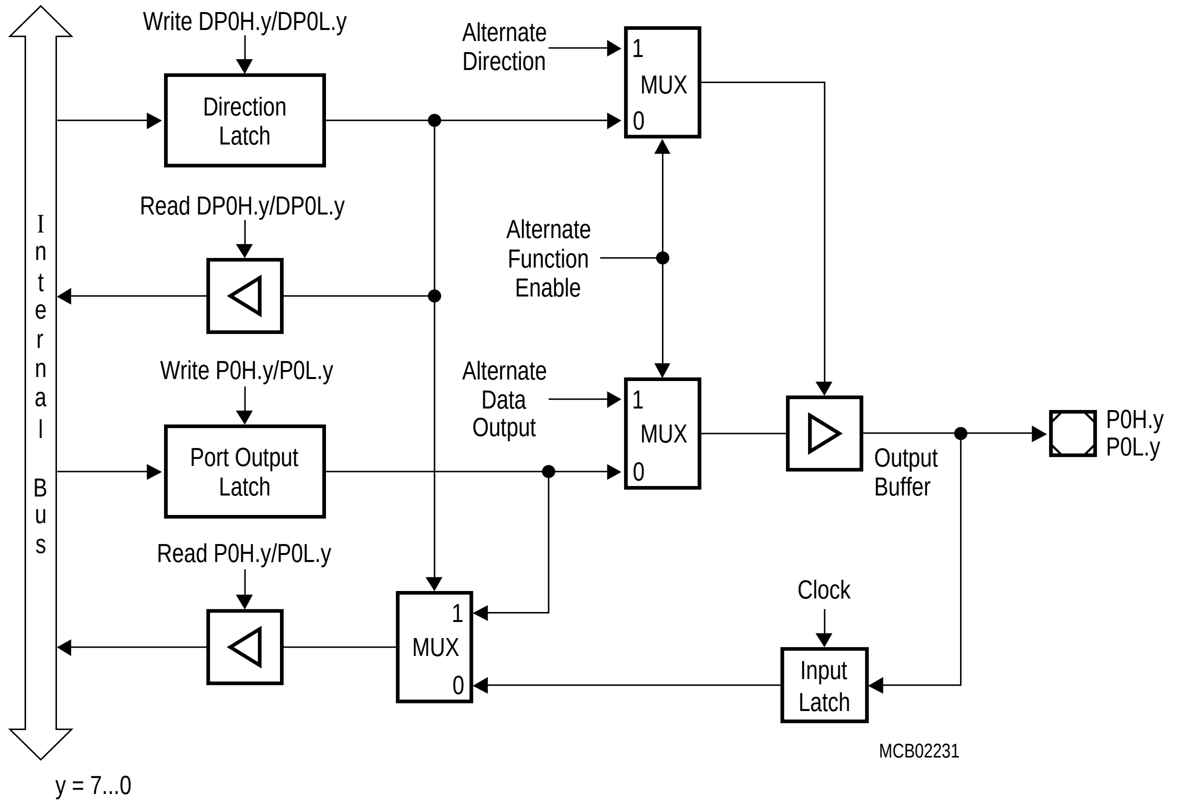


Figure 7-5
Block Diagram of a PORT0 Pin

7.2 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 (FF04$_H$ / 82$_H$) SFR **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 (FF06$_H$ / 83$_H$) SFR **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 (F104$_H$ / 82$_H$) **ESFR** **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 (F106$_H$ / 83$_H$) **ESFR** **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.

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.

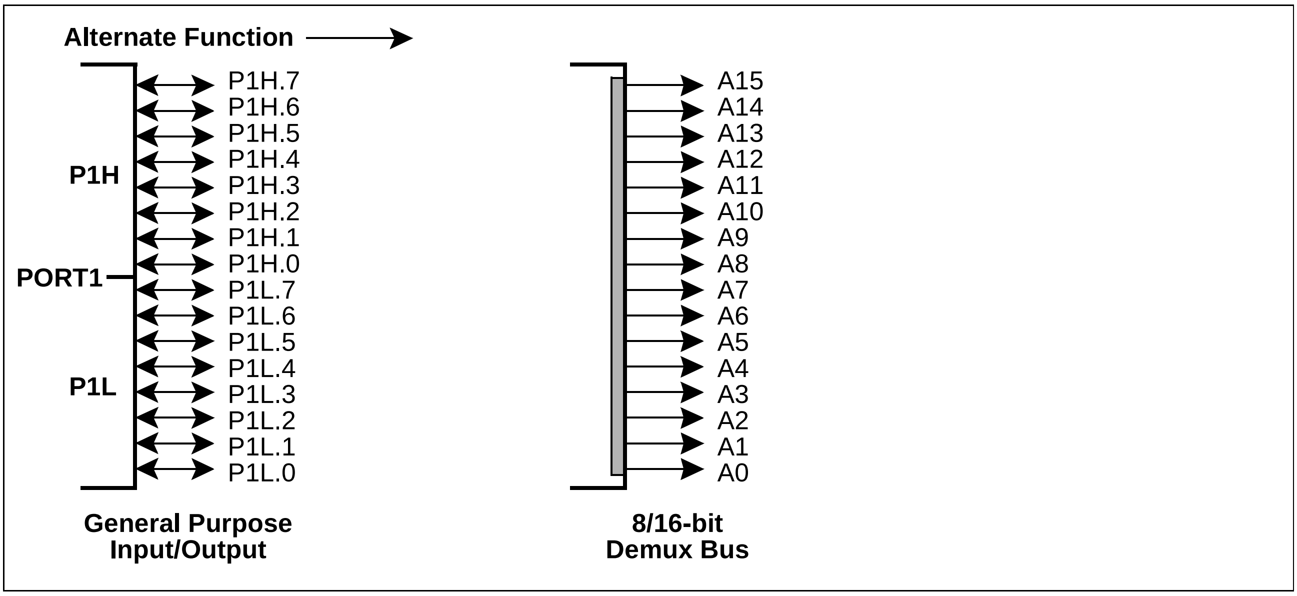


Figure 7-6
PORT1 IO and Alternate Functions

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 is the 16-bit intrasegment address. While an external bus mode is enabled, the user software should not write to the port output latch, otherwise unpredictable results may occur. When the external bus modes are disabled, the contents of the direction register last written by the user becomes active.

The figure below shows the structure of a PORT1 pin.

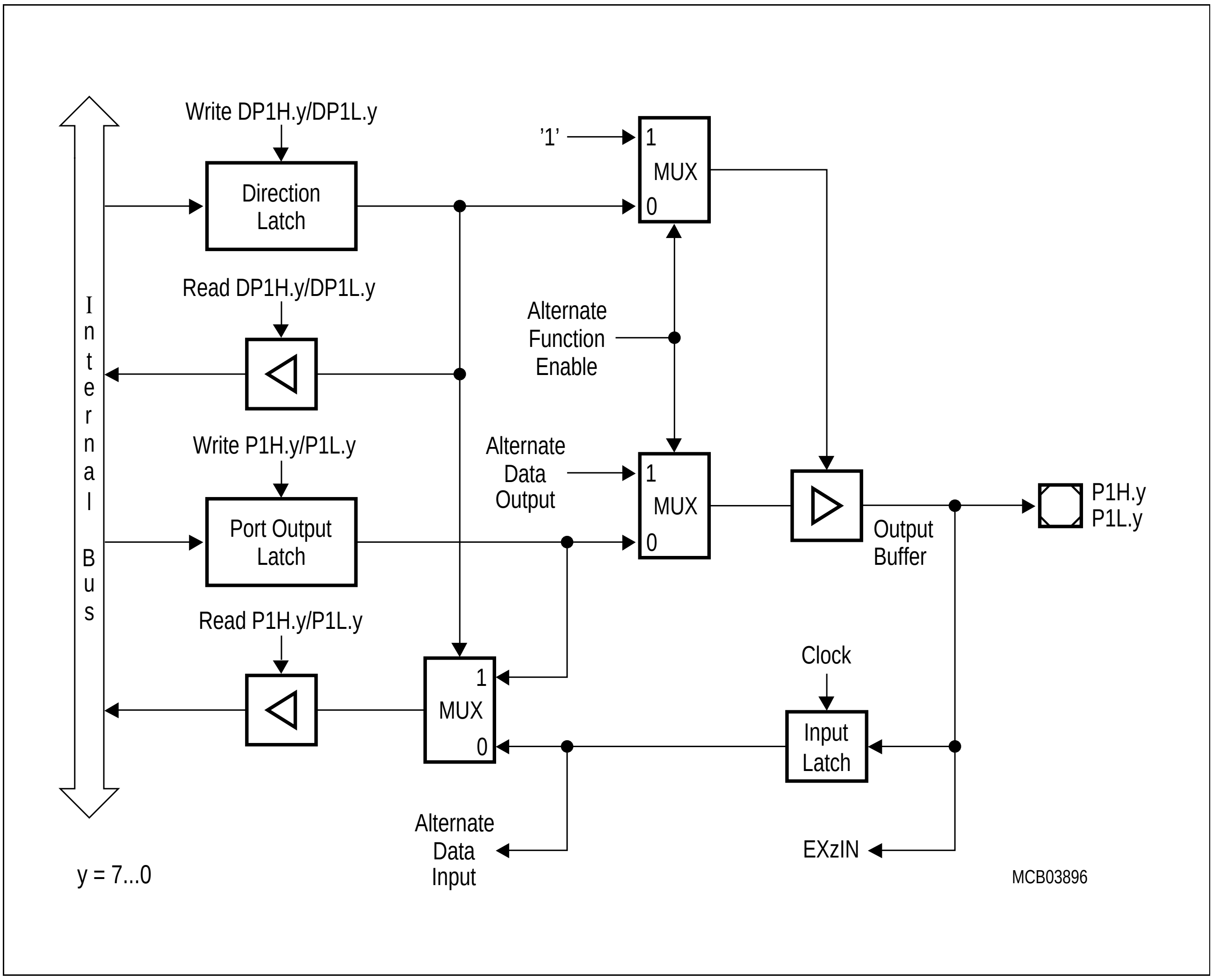


Figure 7-7
Block Diagram of a PORT1 Pin

7.3 Port 2

If this 8-bit port is used for general purpose IO, the direction of each line can be configured via the corresponding direction register DP2. Each port line can be switched into push/pull or open drain mode via the open drain control register ODP2.

P2 (FFC0$_H$ / E0$_H$) SFR **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P2.15	**P2.14**	**P2.13**	**P2.12**	**P2.11**	**P2.10**	**P2.9**	**P2.8**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**
rw	rw	rw	rw	rw	rw	rw	rw	-	-	-	-	-	-	-	-

Bit	Function
P2.y	**Port data register P2 bit y**

DP2 (FFC2$_H$ / E1$_H$) SFR **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DP2 .15	**DP2 .14**	**DP2 .13**	**DP2 .12**	**DP2 .11**	**DP2 .10**	**DP2 .9**	**DP2 .8**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**
rw	rw	rw	rw	rw	rw	rw	rw	-	-	-	-	-	-	-	-

Bit	Function
DP2.y	**Port direction register DP2 bit y** DP2.y = 0: Port line P2.y is an input (high-impedance) DP2.y = 1: Port line P2.y is an output

ODP2 (F1C2$_H$ / E1$_H$) **ESFR** **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODP2 .15	**ODP2 .14**	**ODP2 .13**	**ODP2 .12**	**ODP2 .11**	**ODP2 .10**	**ODP2 .9**	**ODP2 .8**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**
rw	rw	rw	rw	rw	rw	rw	rw	-	-	-	-	-	-	-	-

Bit	Function
ODP2.y	**Port 2 Open Drain control register bit y** ODP2.y = 0: Port line P2.y output driver in push/pull mode ODP2.y = 1: Port line P2.y output driver in open drain mode

Alternate Functions of Port 2

All Port 2 lines (P2.15 … P2.8) serve as external interrupt inputs EX7IN … EX0IN (16 TCL sample rate).

The table below summarizes the alternate functions of Port 2.

Port 2 Pin	Alternate Function b)
P2.8	EX0IN Fast External Interrupt 0 Input
P2.9	EX1IN Fast External Interrupt 1 Input
P2.10	EX2IN Fast External Interrupt 2 Input
P2.11	EX3IN Fast External Interrupt 3 Input
P2.12	EX4IN Fast External Interrupt 4 Input
P2.13	EX5IN Fast External Interrupt 5 Input
P2.14	EX6IN Fast External Interrupt 6 Input
P2.15	EX7IN Fast External Interrupt 7 Input

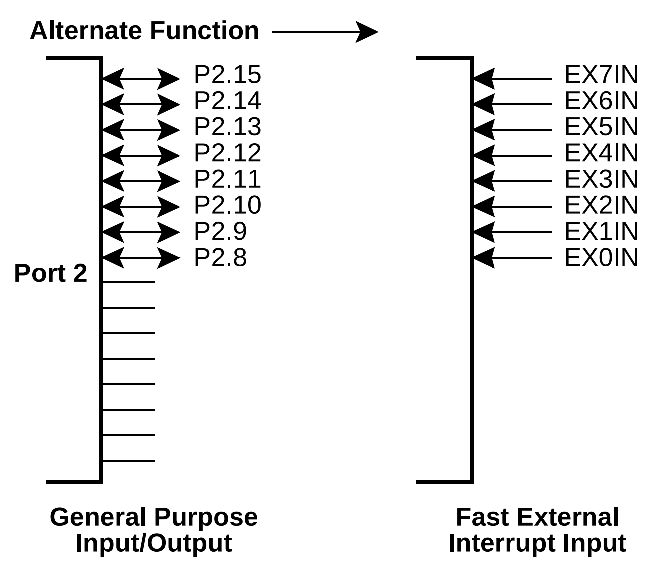


Figure 7-8
Port 2 IO and Alternate Functions

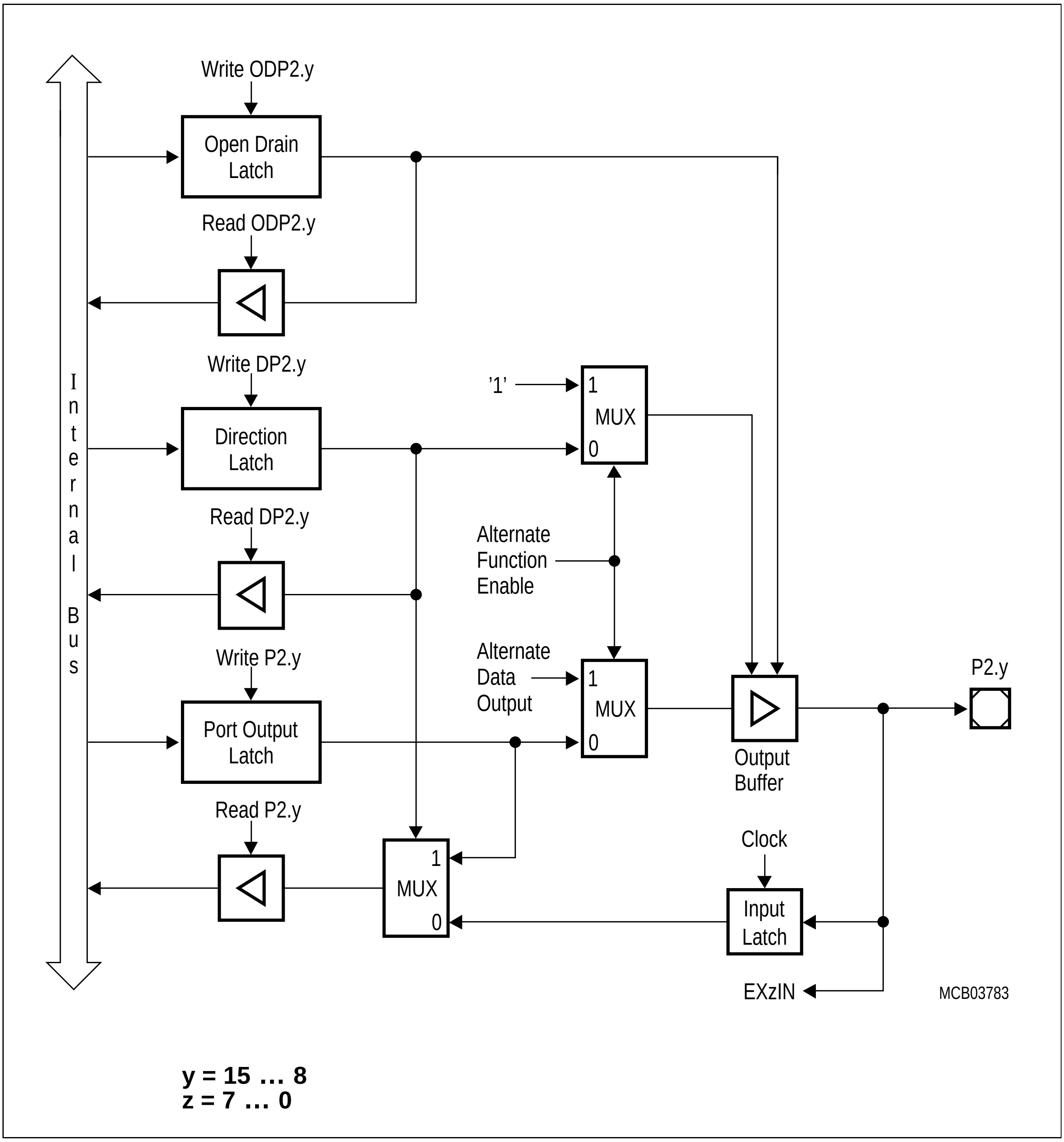


Figure 7-9
Block Diagram of a Port 2 Pin

7.4 Port 3

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

Due to pin limitations register bit P3.14 is not connected to an output pin.

P3 (FFC4$_H$ / E2$_H$) SFR **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P3.15	**-**	**P3.13**	**P3.12**	**P3.11**	**P3.10**	**P3.9**	**P3.8**	**P3.7**	**P3.6**	**P3.5**	**P3.4**	**P3.3**	**P3.2**	**P3.1**	**P3.0**
rw	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
P3.y	**Port data register P3 bit y**

Note: Register bit P3.14 is not connected to an IO pin.

DP3 (FFC6$_H$ / E3$_H$) SFR **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DP3 .15	**-**	**DP3 .13**	**DP3 .12**	**DP3 .11**	**DP3 .10**	**DP3 .9**	**DP3 .8**	**DP3 .7**	**DP3 .6**	**DP3 .5**	**DP3 .4**	**DP3 .3**	**DP3 .2**	**DP3 .1**	**DP3 .0**
rw	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
DP3.y	**Port direction register DP3 bit y** DP3.y = 0: Port line P3.y is an input (high-impedance) DP3.y = 1: Port line P3.y is an output

ODP3 (F1C6$_H$ / E3$_H$) **ESFR** **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	**-**	**ODP3 .13**	**-**	**ODP3 .11**	**ODP3 .10**	**ODP3 .9**	**ODP3 .8**	**ODP3 .7**	**ODP3 .6**	**ODP3 .5**	**ODP3 .4**	**ODP3 .3**	**ODP3 .2**	**ODP3 .1**	**ODP3 .0**
-	-	rw	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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

Alternate Functions of Port 3

The pins of Port 3 serve for various functions which include external timer control lines, the two serial interfaces, one I²C Bus interface and the control lines $\overline{\text{BHE}}$/$\overline{\text{WRH}}$ and CLKOUT.

The table below summarizes the alternate functions of Port 3.

Port 3 Pin	Alternate Function		
P3.0	SCL0	I²C Bus Clock Line 0	(open drain only)
P3.1	SDA0	I²C Bus Data Line 0	(open drain only)
P3.2	CAPIN	GPT2 Capture Input	
P3.3	T3OUT	Timer 3 Toggle Latch Output	
P3.4	T3EUD	Timer 3 External Up/Down Input	
P3.5	T4IN	Timer 4 Count Input	
P3.6	T3IN	Timer 3 Count Input	
P3.7	T2IN	Timer 2 Count Input	
P3.8	MRST	SSC Master Receive / Slave Transmit	
P3.9	MTSR	SSC Master Transmit / Slave Receive	
P3.10	TxD0	ASC0 Transmit Data Output	
P3.11	RxD0	ASC0 Receive Data Input	
P3.12	$\overline{\text{BHE}}$/$\overline{\text{WRH}}$	Byte High Enable / Write High Output	
P3.13	SCLK	SSC Shift Clock Input/Output	
-	-		
P3.15	CLKOUT	System Clock Output	

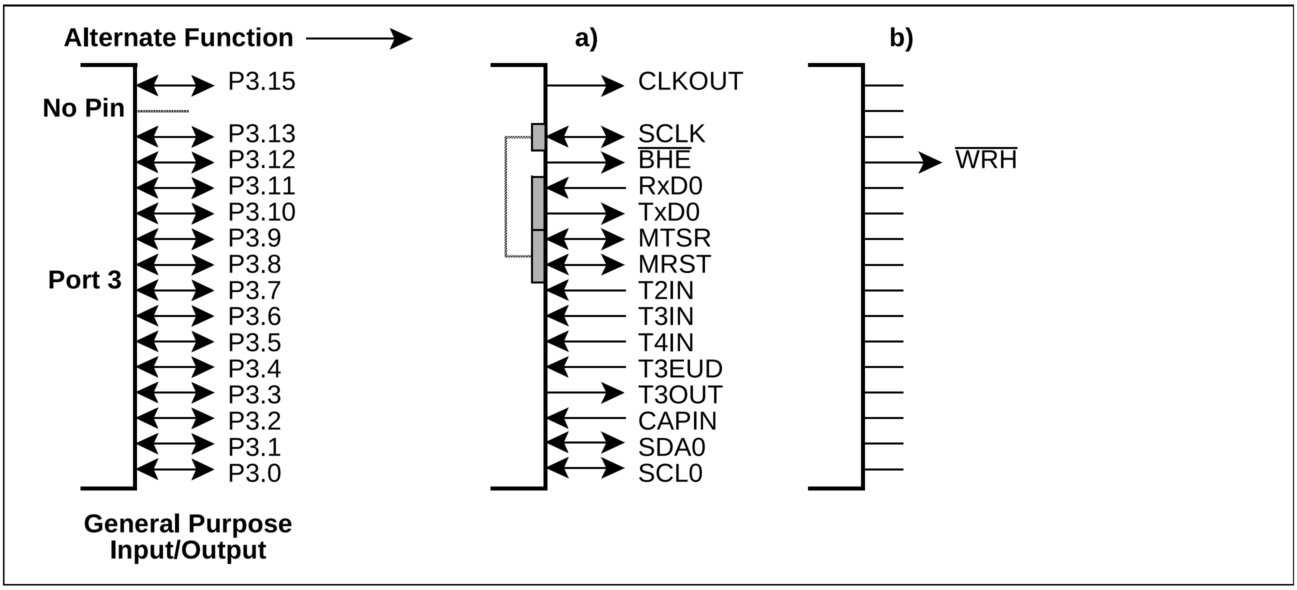


Figure 7-10
Port 3 IO and Alternate Functions

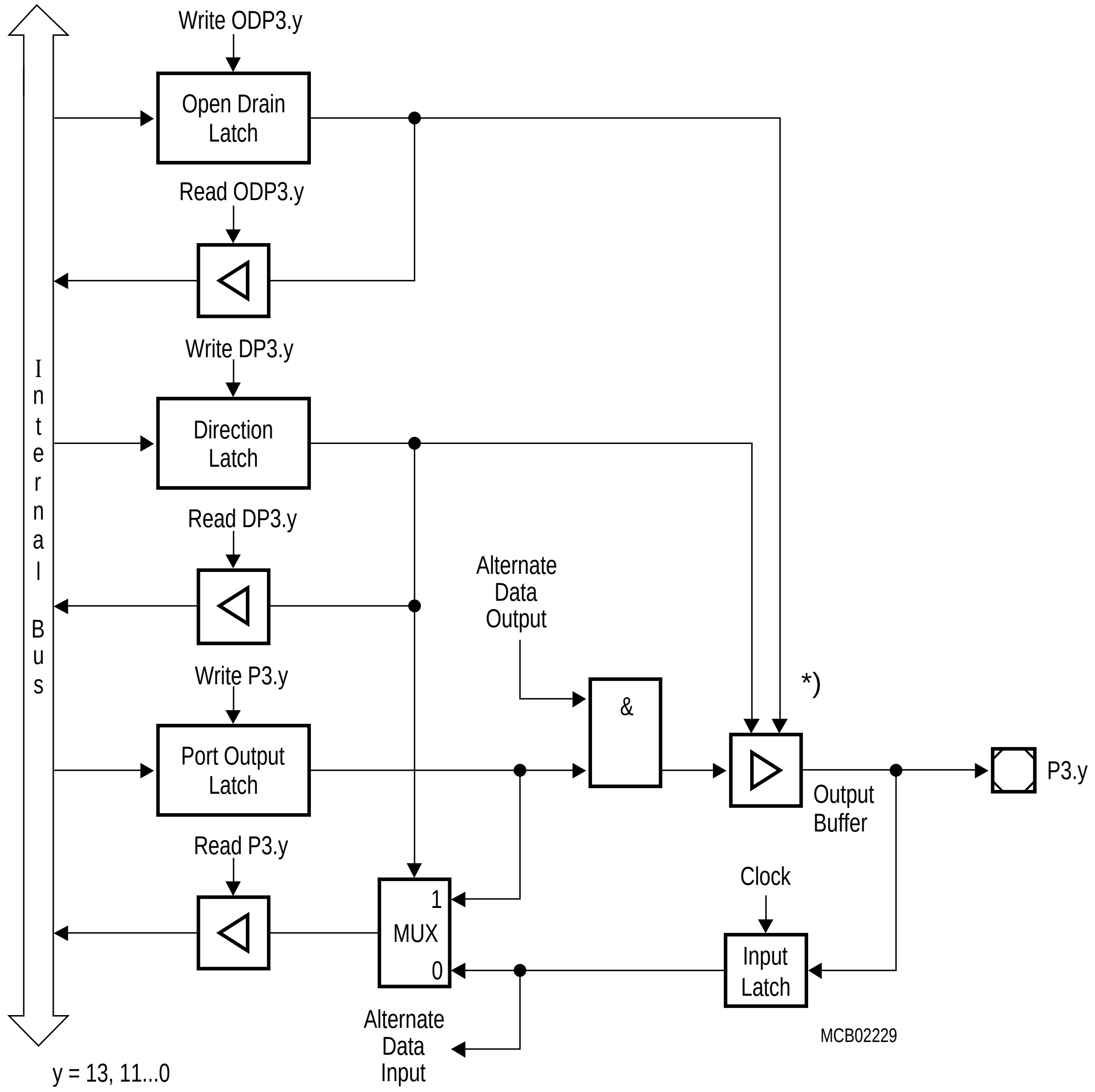
The port structure of the Port 3 pins depends on their alternate function (see figure below).

When the on-chip peripheral associated with a Port 3 pin is configured to use the alternate input function, it reads the input latch, which represents the state of the pin, via the line labeled “Alternate Data Input”. Port 3 pins with alternate input functions are:
T2IN, T3IN, T4IN, T3EUD and CAPIN.

When the on-chip peripheral associated with a Port 3 pin is configured to use the alternate output function, its “Alternate Data Output” line is ANDed with the port output latch line. When using these alternate functions, the user must set the direction of the port line to output (DP3.y = 1) and must set the port output latch (P3.y = 1). Otherwise the pin is in its high-impedance state (when configured as input) or the pin is stuck at '0' (when the port output latch is cleared). When the alternate output functions are not used, the “Alternate Data Output” line is in its inactive state, which is a high level ('1'). Port 3 pins with alternate output functions are:
T3OUT, TxD0 and CLKOUT.

When the on-chip peripheral associated with a Port 3 pin is configured to use both the alternate input and output function, the descriptions above apply to the respective current operating mode. The direction must be set accordingly. Port 3 pins with alternate input/output functions are:
SCL0, SDA0, MTSR, MRST, RxD0 and SCLK.

Note: Enabling the CLKOUT function automatically enables the P3.15 output driver. Setting bit DP3.15 = ’1’ is not required.
The CLKOUT function is automatically enabled in emulation mode.
Pins P3.0 and P3.1 provide open drain output drivers only in order to be compatible with the I^2C Bus specification.



***) Open drain output drivers on pins P3.0 and P3.1.**

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

Pin P3.12 ($\overline{\text{BHE}}$/$\overline{\text{WRH}}$) is one more pin with an alternate output function. However, its structure is slightly different (see figure below), because after reset the $\overline{\text{BHE}}$ or $\overline{\text{WRH}}$ function must be used depending on the system startup configuration. In these cases there is no possibility to program any port latches before. Thus the appropriate alternate function is selected automatically. If $\overline{\text{BHE}}$/$\overline{\text{WRH}}$ is not used in the system, this pin can be used for general purpose IO by disabling the alternate function (BYTDIS = ‘1’ / WRCFG = ‘0’).

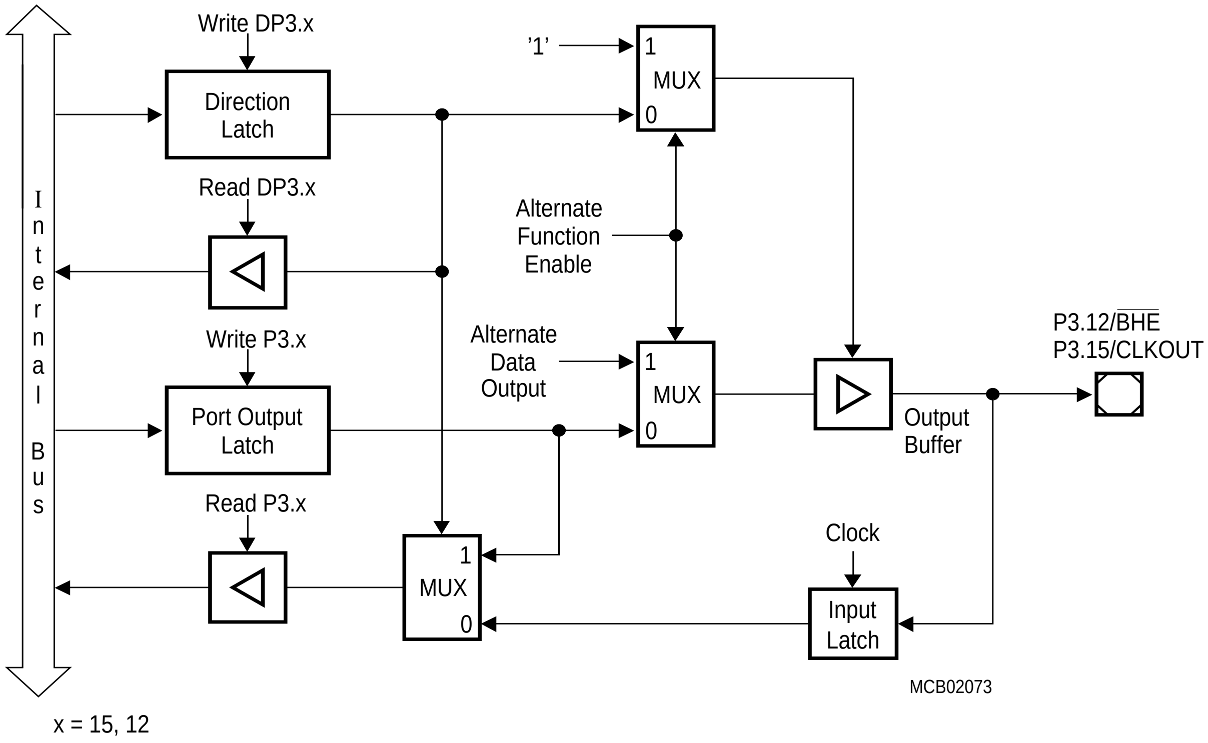


Figure 7-12
Block Diagram of Pins P3.15 (CLKOUT) and P3.12 ($\overline{\text{BHE}}$/$\overline{\text{WRH}}$)

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

7.5 Port 4

If this 7-bit port is used for general purpose IO, the direction of each line can be configured via the corresponding direction register DP4.

P4 (FFC8$_H$ / E4$_H$) SFR **Reset Value: - - 00$_H$**

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

Bit	Function
P4.y	**Port data register P4 bit y**

DP4 (FFCA$_H$ / E5$_H$) SFR **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								-	**DP4.6**	**DP4.5**	**DP4.4**	**DP4.3**	**DP4.2**	**DP4.1**	**DP4.0**
-	-	-	-	-	-	-	-	-	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 PORT0 during reset. The selected value can be read from bitfield SALSEL in register RP0H (read only) e.g. in order to check the configuration during run time.

The table below summarizes the alternate functions of Port 4 depending on the number of selected segment address lines (coded via bitfield SALSEL).

Port 4 Pin	**Std. Function** SALSEL = 01 64 KB	**Altern. Function** SALSEL = 11256KB	**Altern. Function** SALSEL = 00 1 MB	**Altern. Function** SALSEL = 10 4 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
-	-	-	-	-

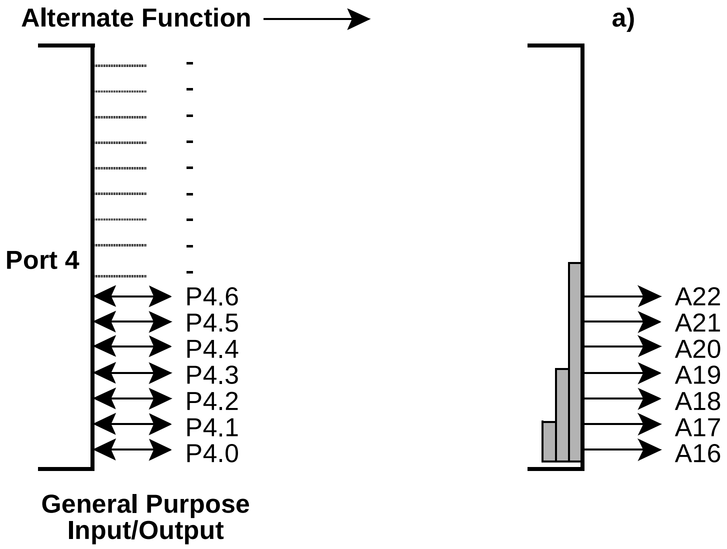


Figure 7-13
Port 4 IO and Alternate Functions

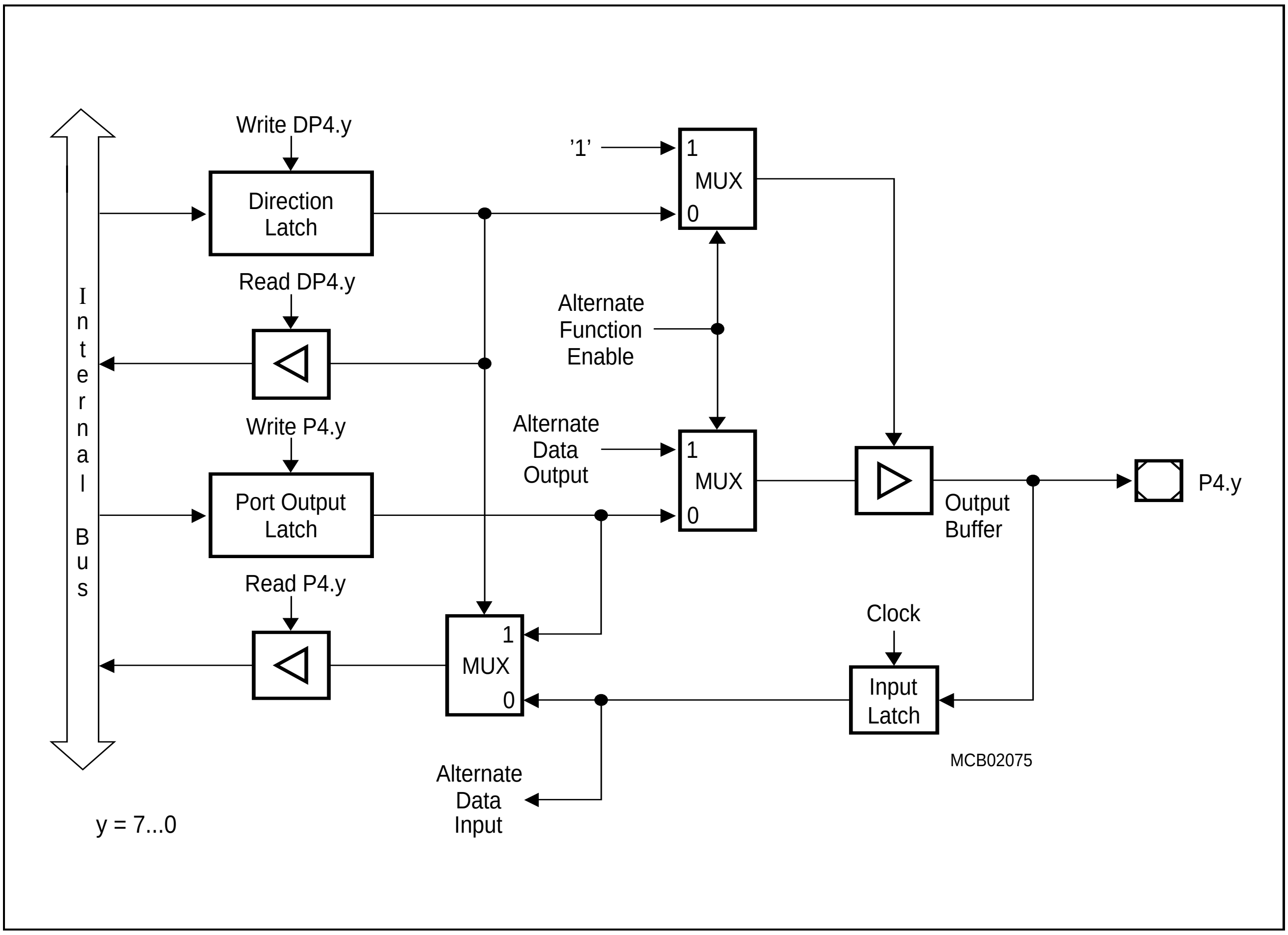


Figure 7-14
Block Diagram of a Port 4 Pin

7.6 Port 5

This 6-bit input port can only read data. There is no output latch and no direction register. Data written to P5 will be lost.

P5 (FFA2$_H$ / D1$_H$) SFR **Reset Value: XXXX$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P5.15	**P5.14**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**P5.3**	**P5.2**	**P5.1**	**P5.0**
r	r	-	-	-	-	-	-	-	-	-	-	r	r	r	r

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

Alternate Functions of Port 5

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

The table below summarizes the alternate functions of Port 5.

Port 5 Pin	Alternate Function a)	Alternate Function b)	
P5.0	Analog Input AN0	-	
P5.1	Analog Input AN1	-	
P5.2	Analog Input AN2	-	
P5.3	Analog Input AN3	-	
P5.14	-	T4EUD	Timer 4 ext. Up/Down Input
P5.15	-	T3EUD	Timer 2 ext. Up/Down Input

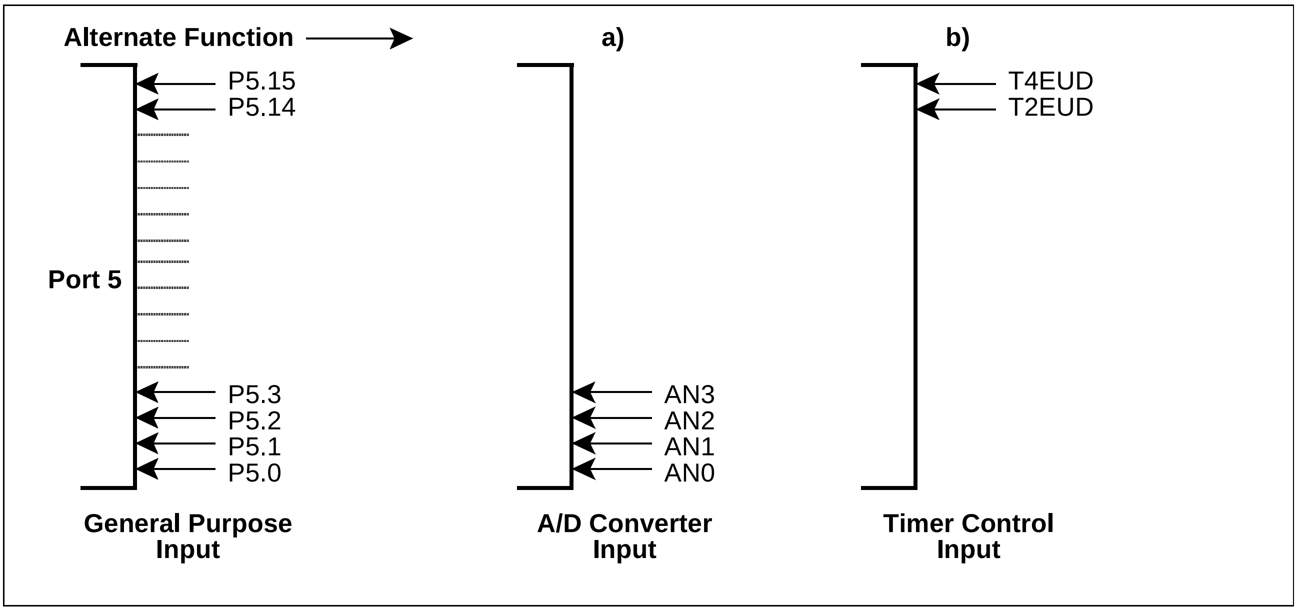


Figure 7-15
Port 5 IO and Alternate Functions

Port 5 Digital Input Control

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

After reset all digital inputs are enabled.

P5DIDIS (FFA4$_H$ / D2$_H$) SFR **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	**P5D.3**	**P5D.2**	**P5D.1**	**P5D.0**
-	-	-	-	-	-	-	-	-	-	-	-	rw	rw	rw	rw

Bit	Function
P5D.y	**Port P5 Bit y Digital Input Control** P5D.y = 0: Digital input stage connected to port line P5.y P5D.y = 1: Digital input stage disconnected from port line P5.y

Port 5 pins have a special port structure (see figure below), first because it is an input only port, and second because the analog input channels are directly connected to the pins rather than to the input latches.

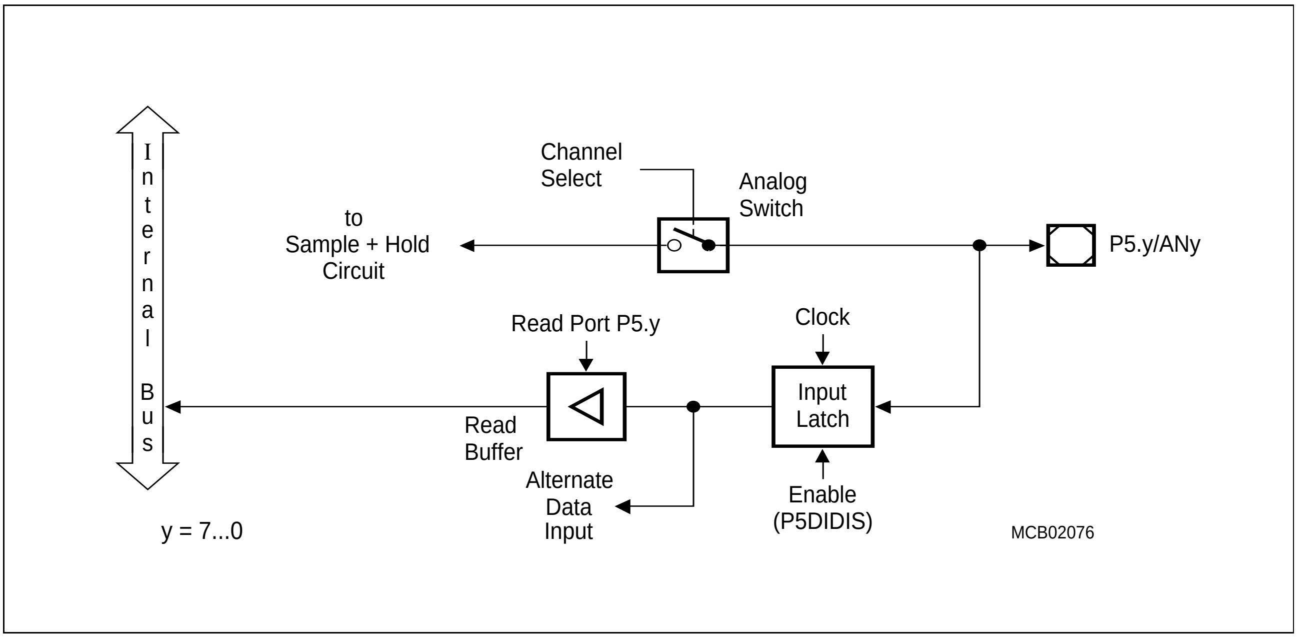


Figure 7-16
Block Diagram of a Port 5 Pin

7.7 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. Each port line can be switched into push/pull or open drain mode via the open drain control register ODP6.

P6 ($FFCC_H$ / $E6_H$) SFR **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 ($FFCE_H$ / $E7_H$) SFR **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

ODP6 ($F1CE_H$ / $E7_H$) **ESFR** **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								-	**-**	**-**	**ODP6.4**	**ODP6.3**	**ODP6.2**	**ODP6.1**	**ODP6.0**
-	-	-	-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw

Bit	Function
ODP6.y	**Port 6 Open Drain control register bit y** ODP6.y = 0: Port line P6.y output driver in push/pull mode ODP6.y = 1: Port line P6.y output driver in open drain mode

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 other 3 pins may be used for I²C Bus interface lines.
The number of chip select signals is selected via PORT0 during reset. The selected value can be read from bitfield CSSEL in register RP0H (read only) e.g. in order to check the configuration during run time.

The table below summarizes the alternate functions of Port 6 depending on the number of selected chip select lines (coded via bitfield CSSEL).

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 $\overline{\text{CS0}}$	Chip select $\overline{\text{CS0}}$	Chip select $\overline{\text{CS0}}$
P6.1	Gen. purpose IO	Chip select $\overline{\text{CS1}}$	Chip select $\overline{\text{CS1}}$	Chip select $\overline{\text{CS1}}$
P6.2	Gen. purpose IO	Gen. purpose IO	Chip select $\overline{\text{CS2}}$	Chip select $\overline{\text{CS2}}$
P6.3	Gen. purpose IO	Gen. purpose IO	Gen. purpose IO	Chip select $\overline{\text{CS3}}$
P6.4	Gen. purpose IO	Gen. purpose IO	Gen. purpose IO	Chip select $\overline{\text{CS4}}$
P6.5	SDA1 I²C bus data line 1			
P6.6	SCL1 I²C bus clock line 1			
P6.7	SDA2 I²C bus data line 2			

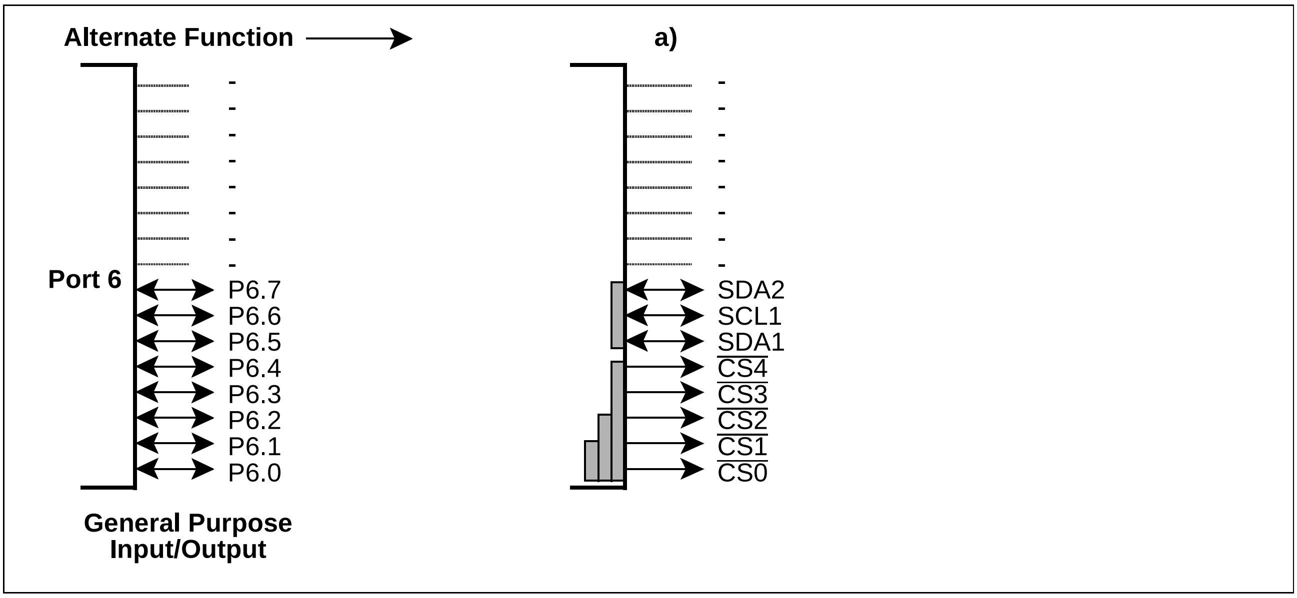
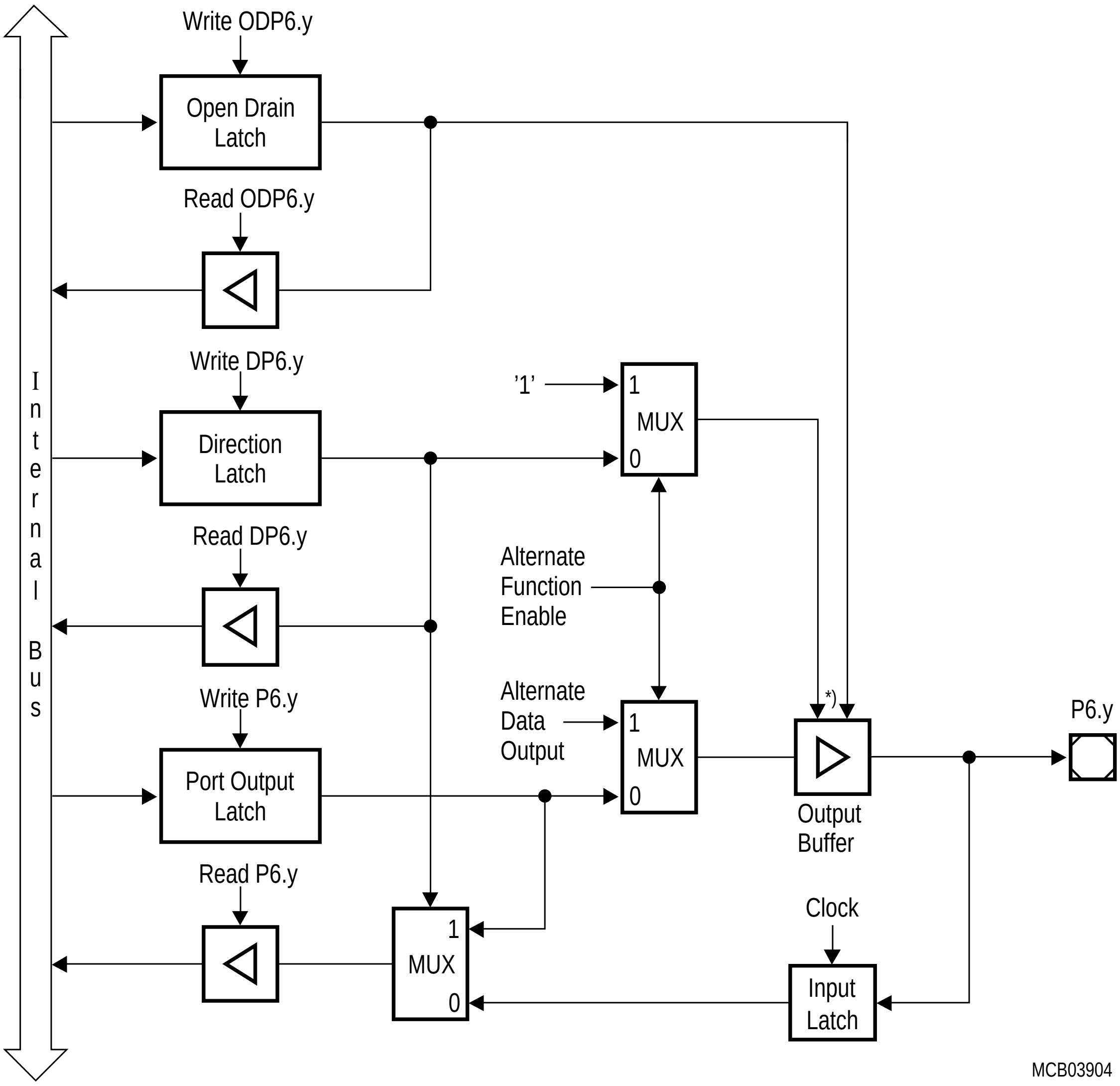


Figure 7-17
Port 6 IO and Alternate Functions

The chip select lines of Port 6 additionally have an internal weak pullup device. This device is switched on always during reset for all potential $\overline{CS}$ output pins. This feature is 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 latches before. Thus the alternate function ($\overline{CS}$) is selected automatically in this case.

Note: The open drain output option can only be selected via software earliest during the initialization routine; the configured chip select lines (via CSSEL) will be in push/pull output driver mode directly after reset.



*) Open drain output drivers on pins P6.5 - P6.7

Figure 7-18
Block Diagram of Port 6 Pins with an Alternate Output Function

8 Dedicated Pins

Most of the input/output or control signals of the functional the C161RI are realized as alternate functions of pins of the parallel ports. There is, however, a number of signals that use separate pins, including the oscillator, special control signals and, of course, the power supply.
The table below summarizes the 21 dedicated pins of the C161RI.

Pin(s)	Function
ALE	Address Latch Enable
$\overline{\mathrm{RD}}$	External Read Strobe
$\overline{\mathrm{WR}}/\overline{\mathrm{WRL}}$	External Write/Write Low Strobe
$\overline{\mathrm{READY}}$	Ready Input
$\overline{\mathrm{EA}}$	External Access Enable
$\overline{\mathrm{NMI}}$	Non-Maskable Interrupt Input
XTAL1, XTAL2	Oscillator Input/Output
$\overline{\mathrm{RSTIN}}$	Reset Input
$\overline{\mathrm{RSTOUT}}$	Reset Output
VAREF, VAGND	Power Supply for Analog/Digital Converter
VDD, VSS	Digital Power Supply and Ground (6 pins each)

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

The External Read Strobe $\overline{\mathrm{RD}}$ controls the output drivers of external memory or peripherals when the C161RI reads data from these external devices. During accesses to on-chip X-Peripherals $\overline{\mathrm{RD}}$ remains inactive (high).
During reset an internal pullup ensures an inactive (high) level on the $\overline{\mathrm{RD}}$ output.

The External Write Strobe $\overline{\mathrm{WR}}/\overline{\mathrm{WRL}}$ controls the data transfer from the C161RI to an external memory or peripheral device. This pin may either provide an general $\overline{\mathrm{WR}}$ signal activated for both byte and word write accesses, or specifically control the low byte of an external 16-bit device ($\overline{\mathrm{WRL}}$) together with the signal $\overline{\mathrm{WRH}}$ (alternate function of P3.12/$\overline{\mathrm{BHE}}$). During accesses to on-chip X-Peripherals $\overline{\mathrm{WR}}/\overline{\mathrm{WRL}}$ remains inactive (high).
During reset an internal pullup ensures an inactive (high) level on the $\overline{\mathrm{WR}}/\overline{\mathrm{WRL}}$ output.

The Ready Input $\overline{\text{READY}}$ receives a control signal from an external memory or peripheral device that is used to terminate an external bus cycle, provided that this function is enabled for the current bus cycle. $\overline{\text{READY}}$ may be used as synchronous $\overline{\text{READY}}$ or may be evaluated asynchronously. When waitstates are defined for a $\overline{\text{READY}}$ controlled address window the $\overline{\text{READY}}$ input is not evaluated during these waitstates.
An internal pullup ensures an inactive (high) level on the $\overline{\text{READY}}$ input.

The External Access Enable Pin $\overline{\text{EA}}$ determines if the C161RI after reset starts fetching code from the internal ROM area ($\overline{\text{EA}}$ = ’1’) or via the external bus interface ($\overline{\text{EA}}$ = ’0’). Be sure to hold this input low for ROMless devices. At the end of the internal reset sequence the $\overline{\text{EA}}$ signal is latched together with the PORT0 configuration.

The Non-Maskable Interrupt Input $\overline{\text{NMI}}$ allows to trigger a high priority trap via an external signal (e.g. a power-fail signal). It also serves to validate the PWRDN instruction that switches the C161RI into Power-Down mode. The $\overline{\text{NMI}}$ pin is sampled with every CPU clock cycle to detect transitions.

The Oscillator Input XTAL1 and Output XTAL2 connect the internal Pierce oscillator to the external crystal. The oscillator provides an inverter and a feedback element. The standard external oscillator circuitry (see chapter “Clock Generation”) comprises the crystal, two low end capacitors and series resistor to limit the current through the crystal.

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

The Reset Input $\overline{\text{RSTIN}}$ allows to put the C161RI into the well defined reset condition either at power-up or external events like a hardware failure or manual reset. The input voltage threshold of the $\overline{\text{RSTIN}}$ pin is raised compared to the standard pins in order to minimize the noise sensitivity of the reset input.

In bidirectional reset mode the C161RI’s line $\overline{\text{RSTIN}}$ may be driven active by the chip logic e.g. in order to support external equipment which is required for startup (e.g. flash memory).

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

The bidirectional reset function is useful in applications where external devices require a defined reset signal but cannot be connected to the C161RI’s $\overline{\text{RSTOUT}}$ signal, e.g. an external flash memory which must come out of reset and deliver code well before $\overline{\text{RSTOUT}}$ can be deactivated via EINIT.

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

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

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

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

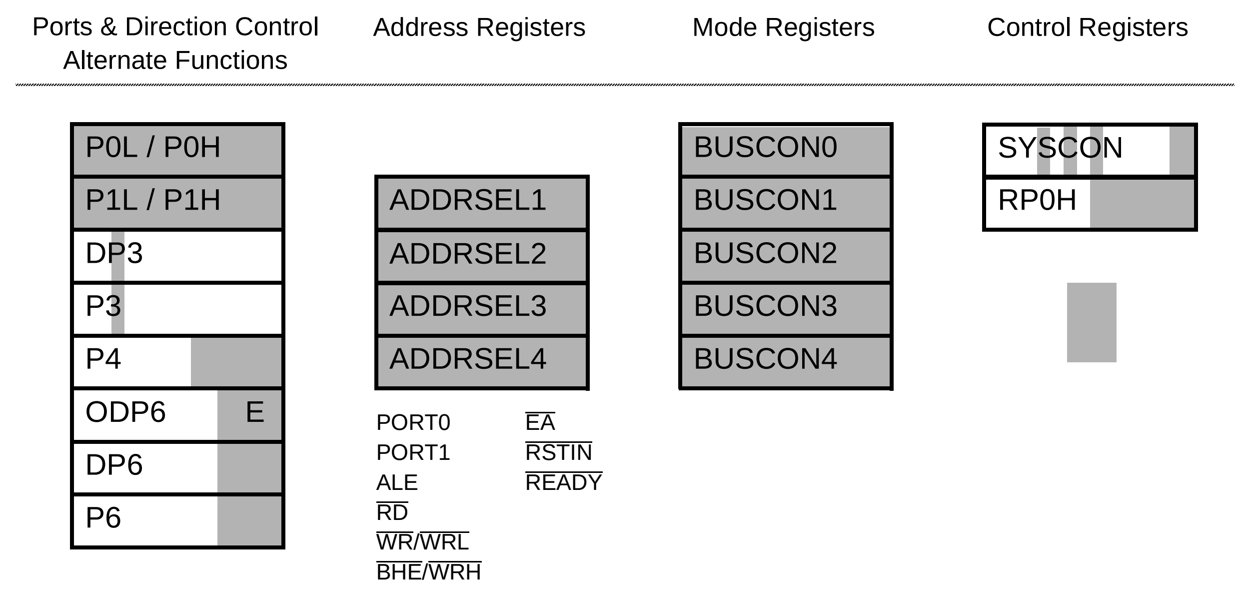
The Power Supply pins for the Analog/Digital Converter VAREF and VAGND provide a separate power supply for the on-chip ADC. This reduces the noise that is coupled to the analog input signals from the digital logic sections and so improves the stability of the conversion results, when VAREF and VAGND are properly discoupled from VDD and VSS.

The Power Supply pins VDD and VSS provide the power supply for the digital logic of the C161RI. The respective VDD/VSS pairs should be decoupled as close to the pins as possible. For best results it is recommended to implement two-level decoupling, e.g. (the widely used) 100 nF in parallel with 30 … 40 pF capacitors which deliver the peak currents.

Note: All VDD pins and all VSS pins must be connected to the power supply and ground, respectively.

9 The External Bus Interface

Although the C161RI provides a powerful set of on-chip peripherals and on-chip RAM and ROM/OTP/Flash (except for ROMless versions) areas, these internal units only cover a small fraction of its address space of up to 16 MByte. The external bus interface allows to access external peripherals and additional volatile and non-volatile memory. The external bus interface provides a number of configurations, so it can be taylored to fit perfectly into a given application system.



P0L/P0H	PORT0 Data Registers	ADDRSELx	Address Range Select Register 1...4
P1L/P1H	PORT1 Data Registers	BUSCONx	Bus Mode Control Register 0...4
DP3	Port 3 Direction Control Register	SYSCON	System Control Register
P3	Port 3 Data Register	RP0H	Port P0H Reset Configuration Register
P4	Port 4 Data Register		
ODP6	Port 6 Open Drain Control Register		
DP6	Port 6 Direction Control Register		
P6	Port 6 Data Register		

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

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

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

Single Chip Mode

Single chip mode is entered, when pin $\overline{EA}$ is high during reset. In this case register BUSCON0 is initialized with 0000_H, which also resets bit BUSACT0, so no external bus is enabled.

In single chip mode the C161RI operates only with and out of internal resources. No external bus is configured and no external peripherals and/or memory can be accessed. Also no port lines are occupied for the bus interface. When running in single chip mode, however, external access may be enabled by configuring an external bus under software control. Single chip mode allows the C161RI to start execution out of the internal program memory (Mask-ROM, OTP or Flash memory).

Note: Any attempt to access a location in the external memory space in single chip mode results in the hardware trap ILLBUS.

9.1 External Bus Modes

When the external bus interface is enabled (bit BUSACTx = ‘1’) and configured (bitfield BTYP), the C161RI uses a subset of its port lines together with some control lines to build the external bus.

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

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

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

The 16 MByte address space of the C161RI is divided into 256 segments of 64 KByte each. The 16-bit intra-segment address is output on PORT0 for multiplexed bus modes or on PORT1 for demultiplexed bus modes. When segmentation is disabled, only one 64 KByte segment can be used and accessed. Otherwise additional address lines may be output on Port 4 (addressing up to 8 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.

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

Multiplexed Bus Modes

In the multiplexed bus modes the 16-bit intra-segment address as well as the data 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, i.e. an 8-bit data bus requires a byte latch (the address bits A15 … A8 on P0H do not change, while P0L multiplexes address and data), a 16-bit data bus requires a word latch (the least significant address line A0 is not relevant for word accesses).
The upper address lines (An … A16) are permanently output on Port 4 (if segmentation is enabled) and do not require latches.

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

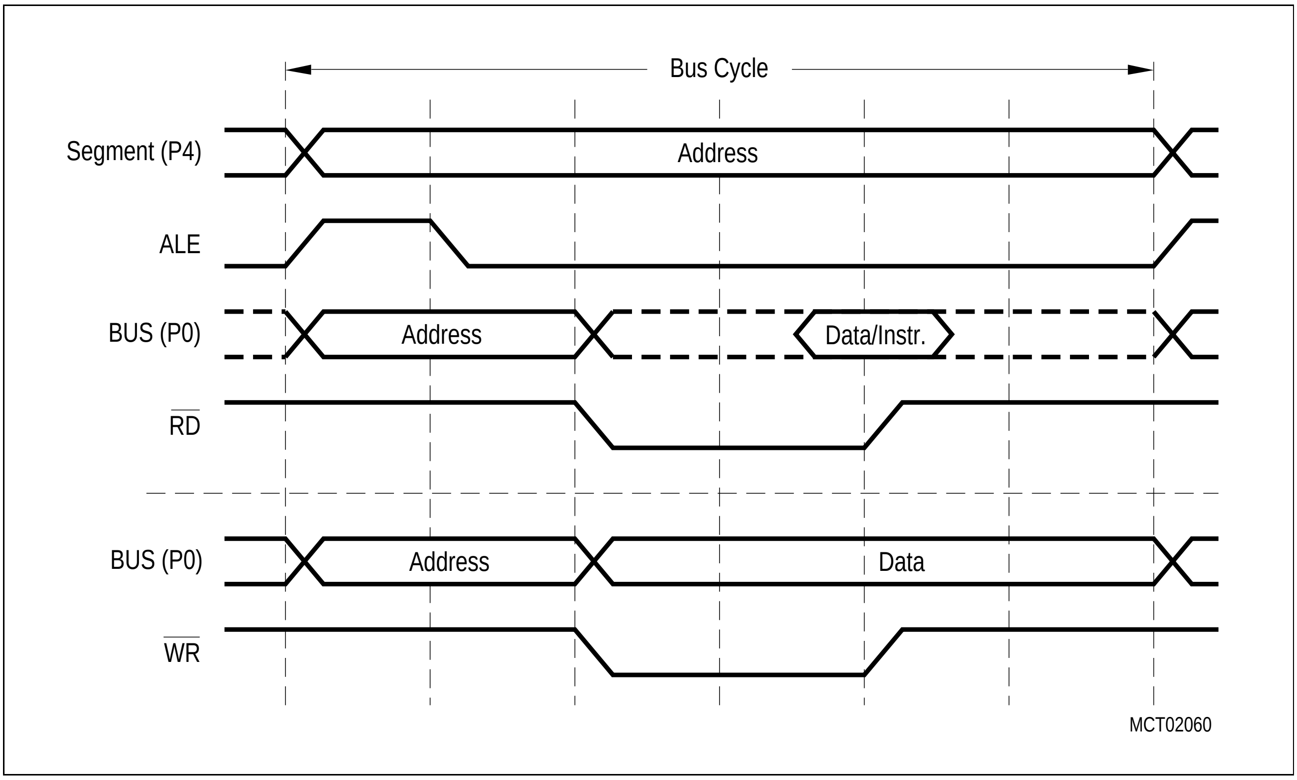


Figure 9-2
Multiplexed Bus Cycle

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 respective command signal ($\overline{\text{RD}}$, $\overline{\text{WR}}$, $\overline{\text{WRL}}$, $\overline{\text{WRH}}$). 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, which 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 data bus which is then tri-stated again.

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

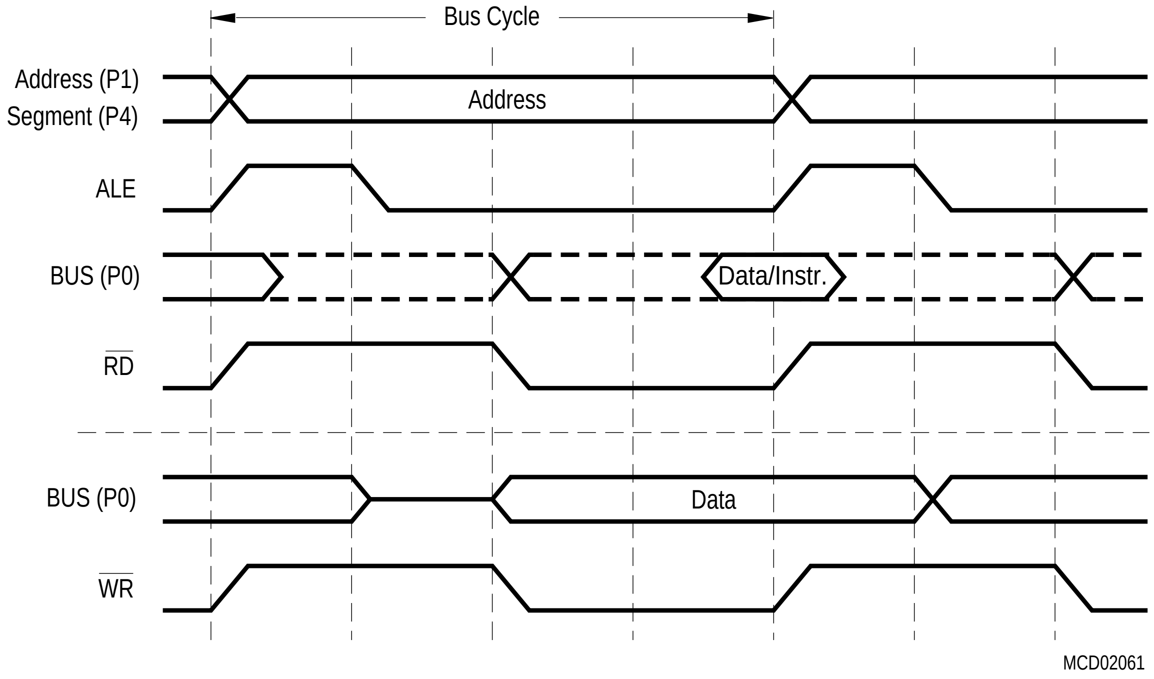


Figure 9-3
Demultiplexed Bus Cycle

Switching between the Bus Modes

The EBC allows to switch between different bus modes dynamically, i.e. subsequent external bus cycles may be executed in different ways. Certain address areas may use multiplexed or demultiplexed buses or use $\overline{\text{READY}}$ control 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 to either change the bus mode for a given address window, or change the size of an address window that uses a certain bus mode. Reprogramming allows to use a great number of different address windows (more than BUSCONs are available) on the expense of the 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 to use different bus modes without any overhead, but restrict their number to the number of BUSCONs. However, as BUSCON0 controls all address areas, which are not covered by the other BUSCONs, this allows to have gaps between these windows, which use the bus mode of BUSCON0.

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

Note: Never change the configuration for an address area that currently supplies the instruction stream. Due to the 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 issued addresses. When an access (code fetch or data) is initiated, the respective generated physical address defines, if 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 the four address windows plus the default is 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 below).

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

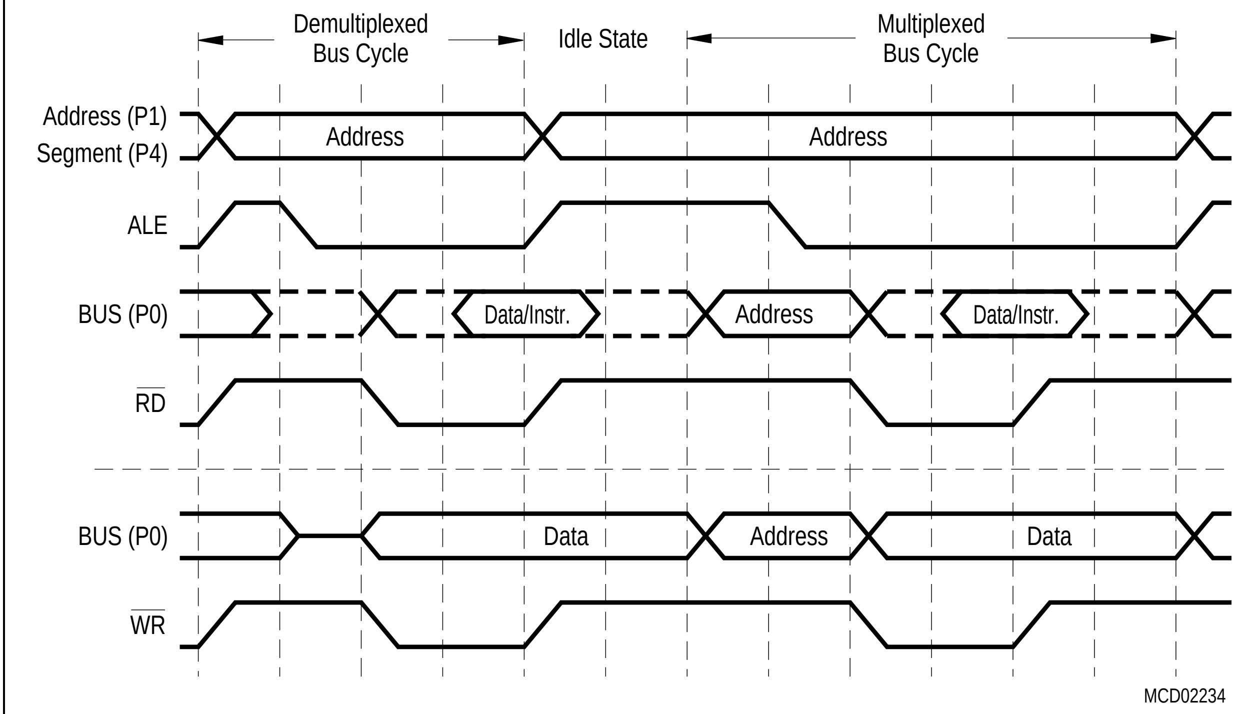


Figure 9-4
Switching from Demultiplexed to Multiplexed Bus Mode

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 cost on the expense of transfer time. The EBC can control word accesses on an 8-bit data bus as well as byte accesses on a 16-bit data bus.

Word accesses on an 8-bit data bus are automatically split into two subsequent byte accesses, where the low byte is accessed first, then the high byte. The assembly of bytes 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 $\overline{\text{BHE}}$ signal, while the lower byte is selected with the A0 signal. So the two bytes of the memory can be enabled independent from each other, or together when accessing words.

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

When reading bytes from an external 16-bit device, whole words may be read and the C161RI 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, like FIFOs, interrupt status registers, etc. In this case individual bytes should be selected using $\overline{\text{BHE}}$ and A0.

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 gets available for general purpose IO, when none of the BUSCON registers selects a demultiplexed bus mode.

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

Bit BYTDIS is provided for controlling the active low Byte High Enable ($\overline{\text{BHE}}$) pin. The function of the $\overline{\text{BHE}}$ pin is enabled, if the BYTDIS bit contains a '0'. Otherwise, it is disabled and the pin can be used as standard IO pin. The $\overline{\text{BHE}}$ pin is implicitly used by the External Bus Controller to select one of two byte-organized memory chips, which are connected to the C161RI 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 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 or PORT1 and so increase the accessible address space. The number of segment address lines is selected during reset and coded in bit field SALSEL in register RP0H (see table below).

SALSEL	Segment Address Lines	Directly accessible Address Space
1 1	Two: A17 … A16	256 KByte (Default without pull-downs)
1 0	Seven: A22 … A16	8 MByte (Maximum)
0 1	None	64 KByte (Minimum)
0 0	Four: A19 … A16	1 MByte

$\overline{\text{CS}}$ Signal Generation

During external accesses the EBC can generate a (programmable) number of $\overline{\text{CS}}$ lines on Port 6, which allow to directly select external peripherals or memory banks without requiring an external decoder. The number of $\overline{\text{CS}}$ lines is selected during reset and coded in bit field CSSEL in register RP0H (see table below).

CSSEL	Chip Select Lines	Note
1 1	Five: $\overline{\text{CS4}}$ … $\overline{\text{CS0}}$	Default without pull-downs
1 0	None	Port 6 pins free for IO
0 1	Two: $\overline{\text{CS1}}$ … $\overline{\text{CS0}}$	
0 0	Three: $\overline{\text{CS2}}$ … $\overline{\text{CS0}}$	

The $\overline{\text{CSx}}$ 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 $\overline{\text{CSx}}$ signal will go inactive (high). At the beginning of each external bus cycle the corresponding valid $\overline{\text{CS}}$ signal is determined and activated. All other $\overline{\text{CS}}$ lines are deactivated (driven high) at the same time.

Note: The $\overline{\text{CSx}}$ 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 access to an on-chip X-Peripheral deactivates all external $\overline{\text{CS}}$ signals.
Upon accesses to address windows without a selected $\overline{\text{CS}}$ line all selected $\overline{\text{CS}}$ lines are deactivated.

The chip select signals allow to be operated in four different modes (see table below) which are selected via bits CSWENx and CSRENx in the respective BUSCONx register.

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

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

Address Chip Select signals remain active during the complete bus cycle. For address chip select signals two generation modes can be selected via bit CSCFG in register SYSCON:
- A **latched** address chip select signal (CSCFG = ’0’) becomes active with the falling edge of ALE and becomes inactive at the beginning of an external bus cycle 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 X-Peripherals and XRAM) are accessed.
- An **early** address chip select signal (CSCFG = ’1’) becomes active together with the address and 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: CS0 provides a latched address chip select directly after reset (except for single chip mode) when the first instruction is fetched.

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

Segment Address versus Chip Select

The external bus interface of the C161RI supports many configurations for the external memory. By increasing the number of segment address lines the C161RI can address a linear address space of 256 KByte, 1 MByte or 8 MByte. This allows to implement a large sequential memory area, and also allows to access a great number of external devices, using an external decoder. By increasing the number of CS lines the C161RI can access memory banks or peripherals without external glue logic. These two features may be combined to optimize the overall system performance.

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

9.2 Programmable Bus Characteristics

Important timing characteristics of the external bus interface have been made user programmable to allow to adapt it to a wide range of different external bus and memory configurations with different types of memories and/or peripherals.

The following parameters of an external bus cycle are programmable:

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

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

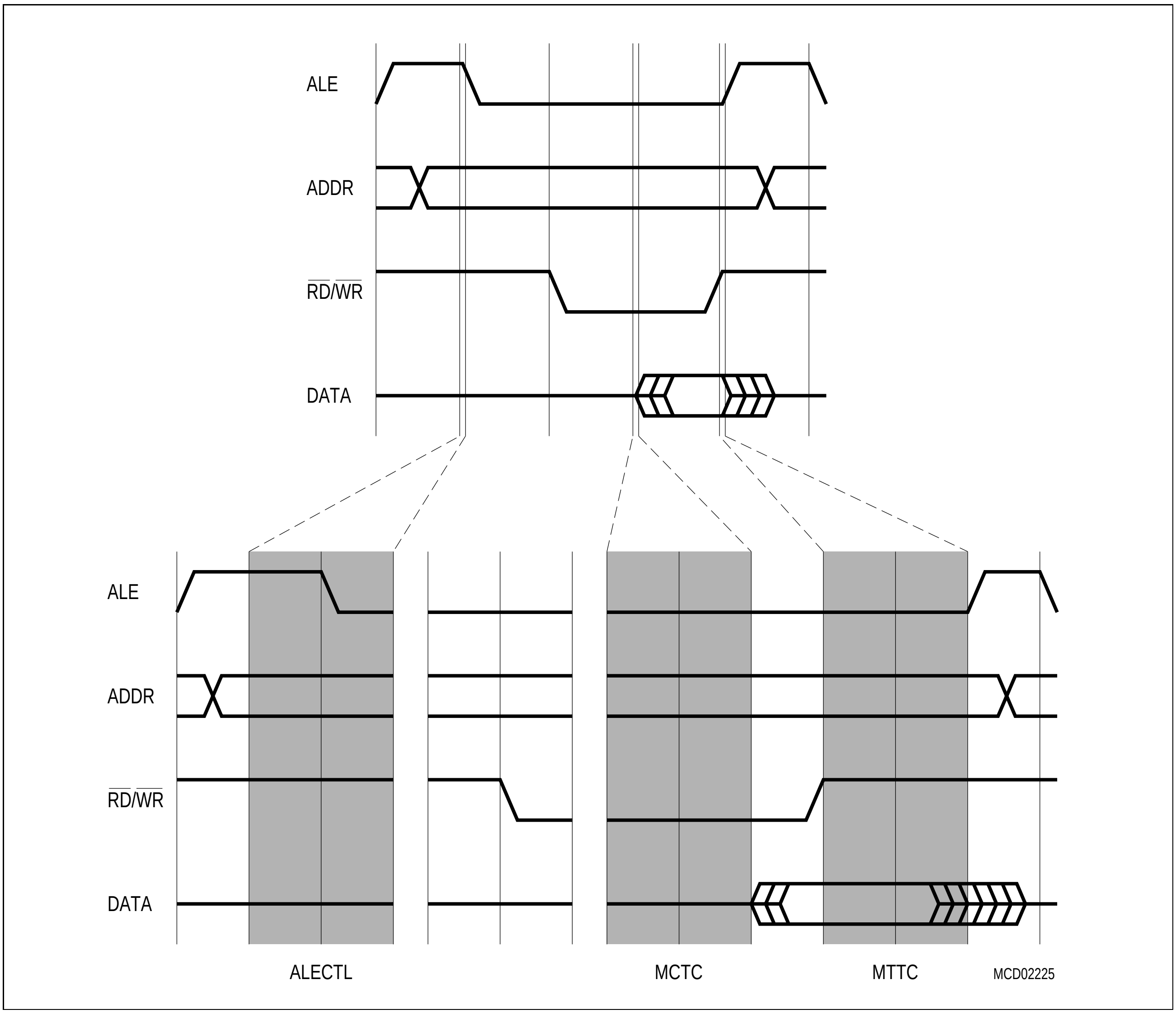


Figure 9-5
Programmable External Bus Cycle

ALE Length Control

The length of the ALE signal and the address hold time after its falling edge are controlled by the ALECTLx bits in the BUSCON registers. When bit ALECTL is set to '1', external bus cycles accessing the respective address window will have their ALE signal prolonged by half a CPU clock (1 TCL). Also the address hold time after the falling edge of ALE 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 are '0' after reset.

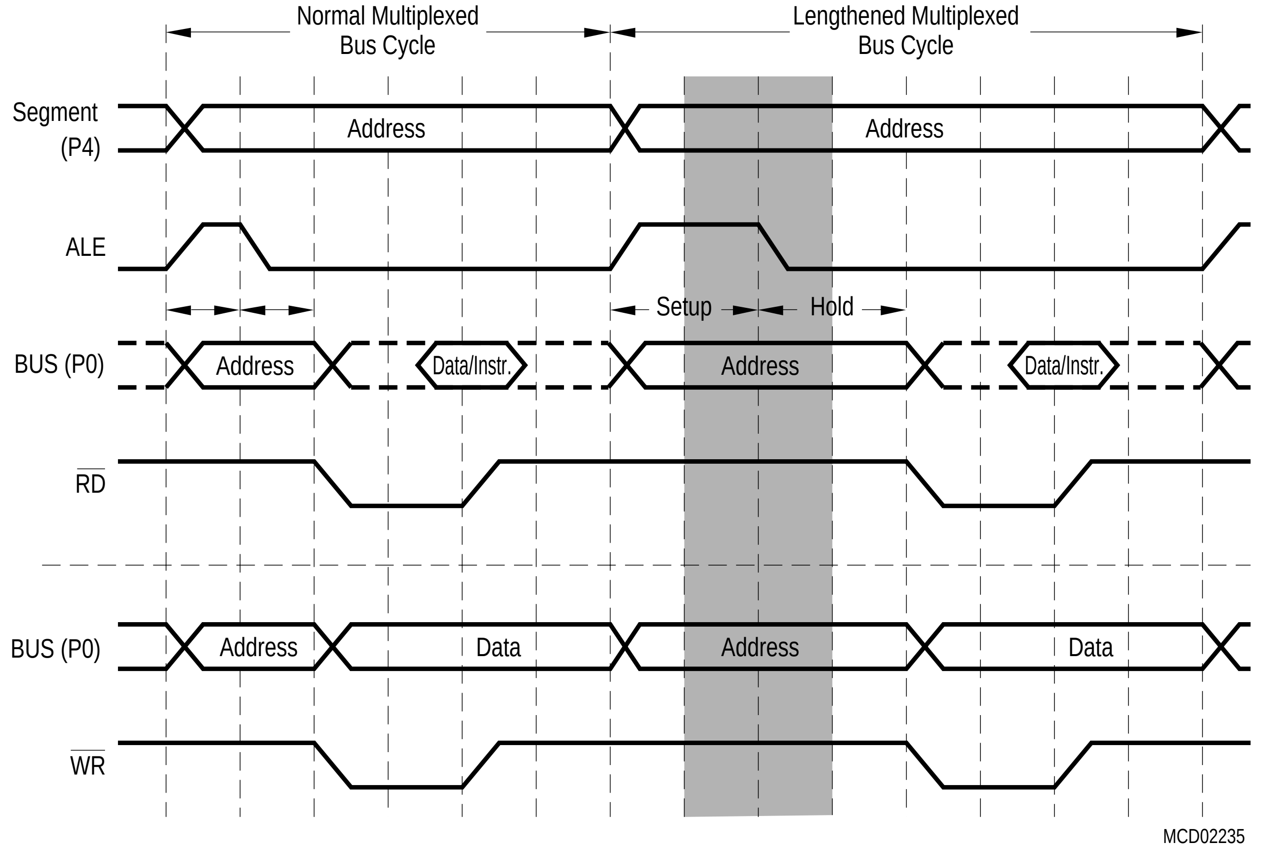


Figure 9-6
ALE Length Control

Programmable Memory Cycle Time

The C161RI allows the user to adjust the controller's external bus cycles to the access time of the respective memory or peripheral. This access time is the total time required to move the data to the destination. It represents the period of time during which the controller's signals do not change.

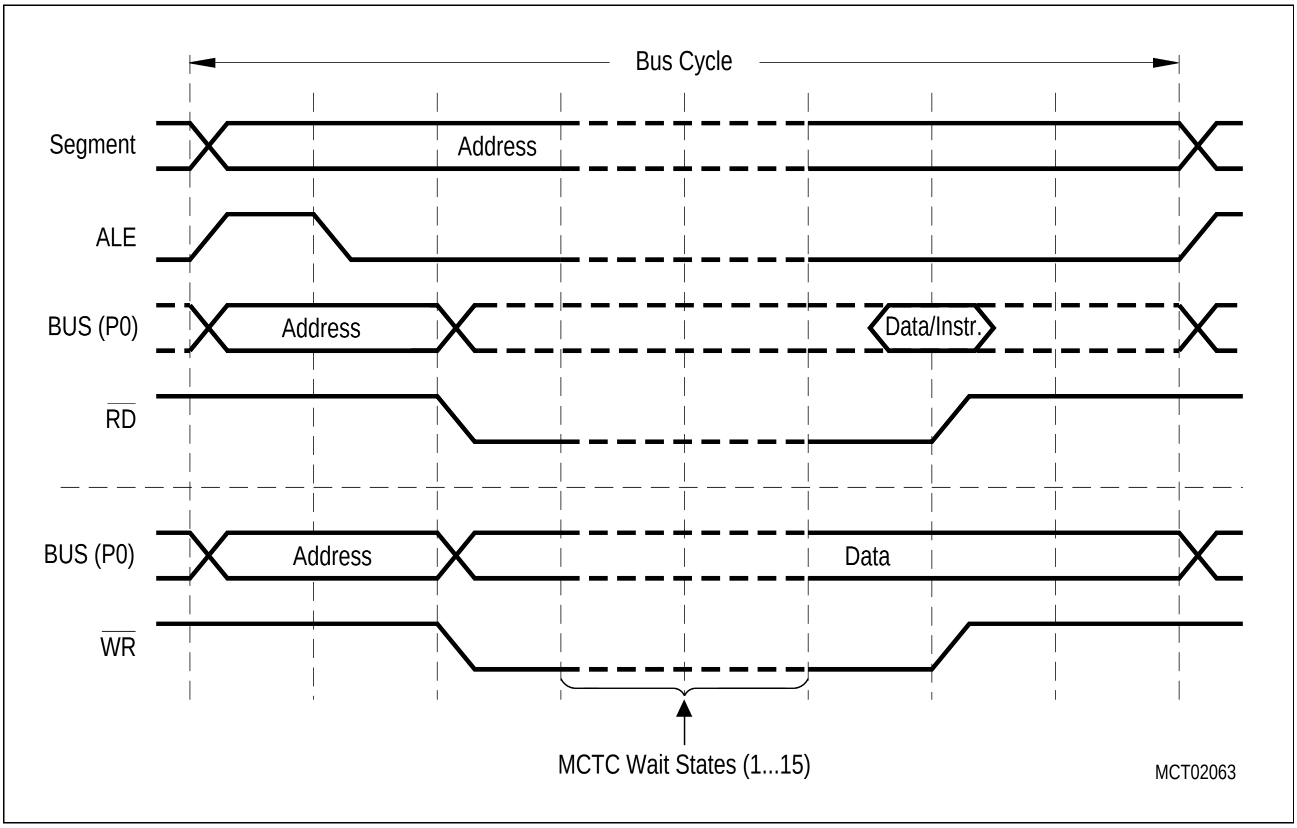


Figure 9-7
Memory Cycle Time

The external bus cycles of the C161RI can be extended for a memory or peripheral, which cannot keep pace with the controller's maximum speed, by introducing wait states during the access (see figure above). During these memory cycle time wait states, the CPU is idle, if this access is required for the execution of the current instruction.

The memory cycle time wait states can be programmed in increments of one CPU clock (2 TCL) within a range from 0 to 15 (default after reset) via the MCTC fields of the BUSCON registers. 15-<MCTC> waitstates will be inserted.

Programmable Memory Tri-State Time

The C161RI allows the user to adjust the time between two subsequent external accesses to account for the tri-state time of the external device. The tri-state time defines, when the external device has released the bus after deactivation of the read command ($\overline{RD}$).

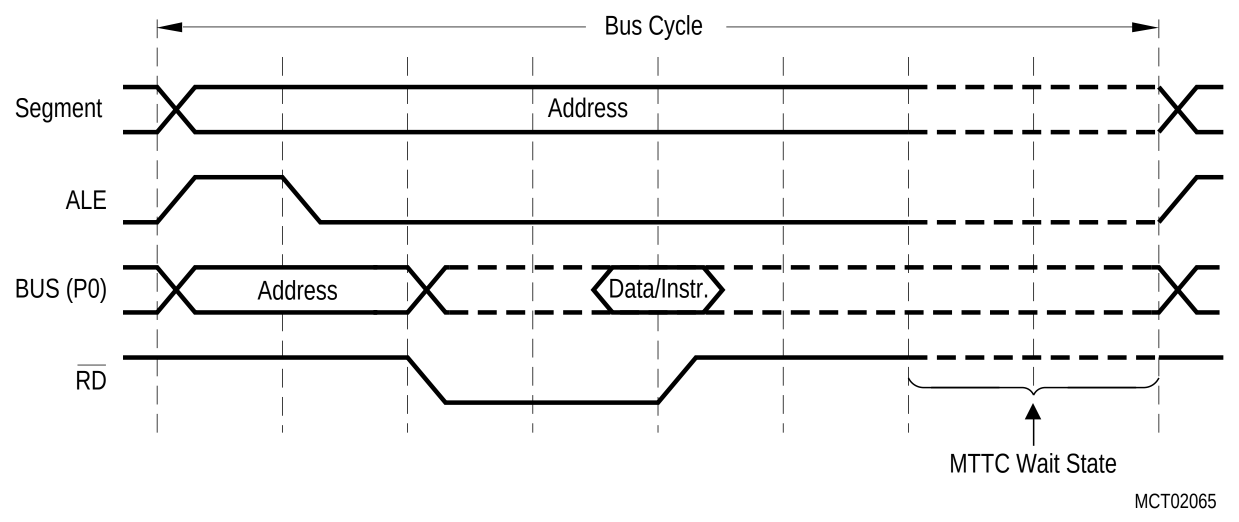


Figure 9-8
Memory Tri-State Time

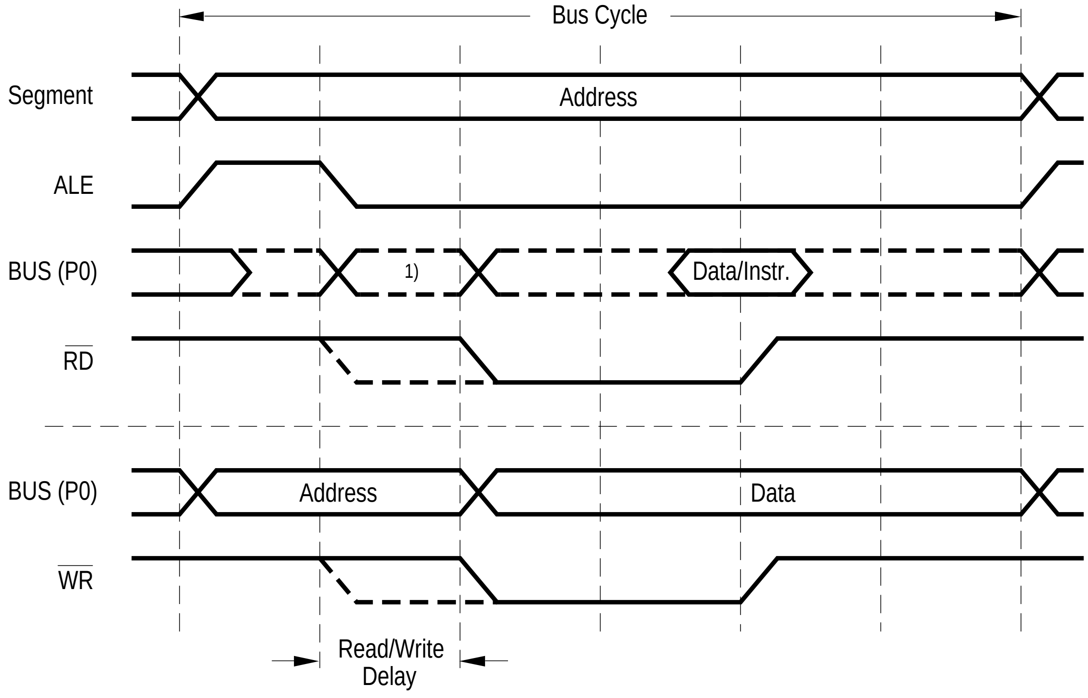
The output of the next address on the external bus can be delayed for a memory or peripheral, which needs more time to switch off its bus drivers, by introducing a wait state after the previous bus cycle (see figure above). During this memory tri-state time wait state, the CPU is not idle, so CPU operations will only be slowed down if a subsequent external instruction or data fetch operation is required during the next instruction cycle.

The memory tri-state time waitstate requires one CPU clock (2 TCL) and is controlled via the MTTCx bits of the BUSCON registers. A waitstate will be inserted, if bit MTTCx is ‘0’ (default after reset).

Note: External bus cycles in multiplexed bus modes implicitly add one tri-state time waitstate in addition to the programmable MTTC waitstate.

Read/Write Signal Delay

The C161RI 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 coincident (except for propagation delays). With the delay enabled, the command(s) become active half a CPU clock (1 TCL) after the falling edge of ALE.
The read/write delay does not extend the memory cycle time, and does not slow down the controller in general. In multiplexed bus modes, however, the data drivers of an external device may conflict with the C161RI's address, when the early $\overline{RD}$ signal is used. Therefore multiplexed bus cycles should always be programmed with read/write delay.



[1] The Data drivers from the previous bus cycle should be disabled when the $\overline{RD}$ signal becomes active.

MCT02066

Figure 9-9
Read/Write Delay

The read/write delay is controlled via the RWDCx bits in the BUSCON registers. The command(s) will be delayed, if bit RWDCx is '0' (default after reset).

9.3 $\overline{\text{READY}}$ Controlled Bus Cycles

For situations, where the programmable waitstates are not enough, or where the response (access) time of a peripheral is not constant, the C161RI provides external bus cycles that are terminated via a $\overline{\text{READY}}$ input signal (synchronous or asynchronous). In this case the C161RI first inserts a programmable number of waitstates (0 … 7) and then monitors the $\overline{\text{READY}}$ line to determine the actual end of the current bus cycle. The external device drives $\overline{\text{READY}}$ low in order to indicate that data have been latched (write cycle) or are available (read cycle).

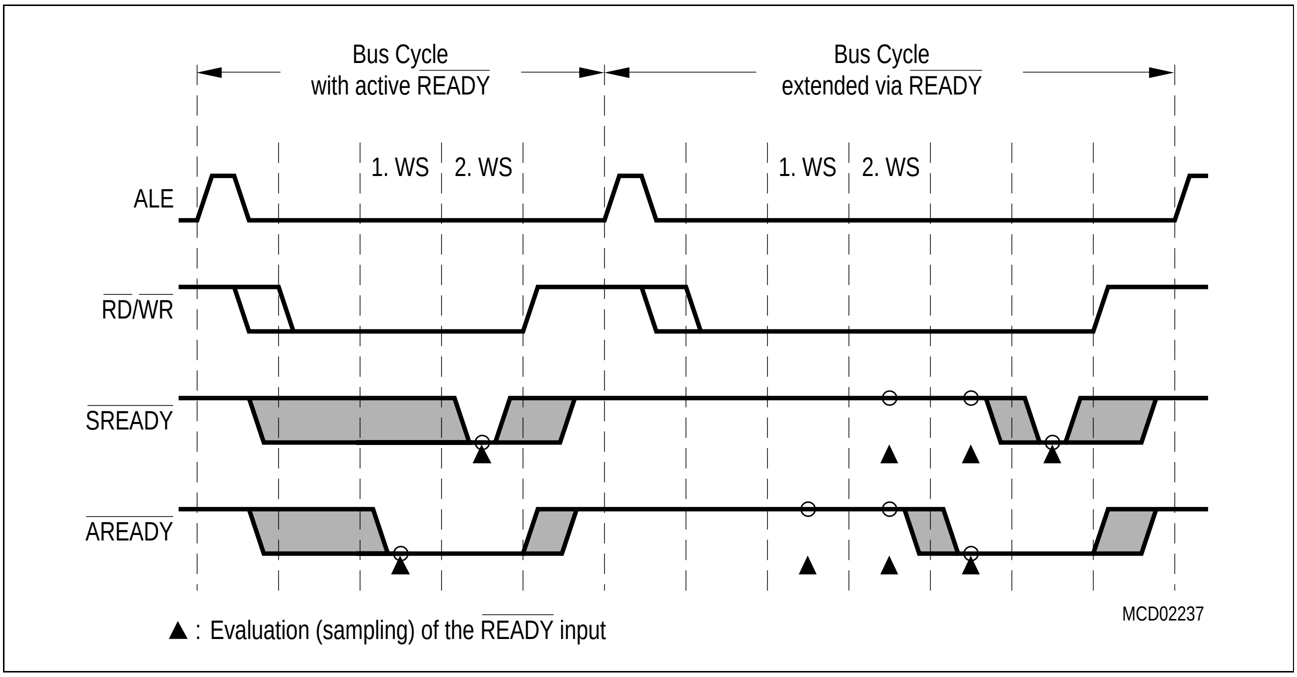


Figure 9-10
$\overline{\text{READY}}$ Controlled Bus Cycles

The $\overline{\text{READY}}$ function is enabled via the 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 selects the $\overline{\text{READY}}$ operation:

MCTC.3 = '0': Synchronous $\overline{\text{READY}}$, i.e. the $\overline{\text{READY}}$ signal must meet setup and hold times.
MCTC.3 = '1': Asynchronous $\overline{\text{READY}}$, i.e. the $\overline{\text{READY}}$ signal is synchronized internally.

The Synchronous $\overline{\text{READY}}$ provides the fastest bus cycles, but requires setup and hold times to be met. The CLKOUT signal **should be enabled** and may be used by the peripheral logic to control the $\overline{\text{READY}}$ timing in this case.

The Asynchronous $\overline{\text{READY}}$ is less restrictive, but requires additional waitstates caused by the internal synchronization. As the asynchronous $\overline{\text{READY}}$ is sampled earlier (see figure above) programmed waitstates may be necessary to provide proper bus cycles (see also notes on "normally-ready" peripherals below).

A $\overline{\text{READY}}$ signal (especially asynchronous $\overline{\text{READY}}$) that has been activated by an external device may be deactivated in response to the trailing (rising) edge of the respective command ($\overline{\text{RD}}$ or $\overline{\text{WR}}$).

Note: When the $\overline{\text{READY}}$ function is enabled for a specific address window, each bus cycle within this window must be terminated with an active $\overline{\text{READY}}$ signal. Otherwise the controller hangs until the next reset. A timeout function is only 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 $\overline{\text{READY}}$ may be grouped into the same address window. The (external) waitstate control logic in this case would activate $\overline{\text{READY}}$ either upon the memory's chip select or with the peripheral's $\overline{\text{READY}}$ output. After the predefined number of waitstates the C161RI will check its $\overline{\text{READY}}$ line to determine the end of the bus cycle. For a memory access it will be low already (see example a) in the figure above), for a peripheral access it may be delayed (see example b) in the figure above). As memories tend to be faster than peripherals, there should be no impact on system performance.

When using the $\overline{\text{READY}}$ function with so-called “normally-ready” peripherals, it may lead to erroneous bus cycles, if the $\overline{\text{READY}}$ line is sampled too early. These peripherals pull their $\overline{\text{READY}}$ output low, while they are idle. When they are accessed, they deactivate $\overline{\text{READY}}$ until the bus cycle is complete, then drive it low again. If, however, the peripheral deactivates $\overline{\text{READY}}$ **after** the first sample point of the C161RI, the controller samples an active $\overline{\text{READY}}$ and terminates the current bus cycle, which, of course, is too early. By inserting predefined waitstates the first $\overline{\text{READY}}$ sample point can be shifted to a time, where the peripheral has safely controlled the $\overline{\text{READY}}$ line (e.g. after 2 waitstates in the figure above).

9.4 Controlling the External Bus Controller

A set of registers controls the functions of the EBC. General features like the usage of interface pins ($\overline{\text{WR}}$, $\overline{\text{BHE}}$), segmentation and internal ROM mapping are controlled via register SYSCON. The properties of a bus cycle like 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 allows 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 to use memory components or peripherals with different interfaces within the same system, while optimizing accesses to each of them.

SYSCON (FF12$_H$ / 89$_H$) SFR **Reset Value: 0XX0$_H$**

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

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

Bit	Function
CLKEN	**System Clock Output Enable** (CLKOUT) 0: CLKOUT disabled: pin may be used for general purpose IO 1: CLKOUT enabled: pin outputs the system clock signal
BYTDIS	**Disable/Enable Control for Pin $\overline{\text{BHE}}$** (Set according to data bus width) 0: Pin $\overline{\text{BHE}}$ enabled 1: Pin $\overline{\text{BHE}}$ disabled, pin may be used for general purpose IO
ROMEN	**Internal ROM Enable** (Set according to pin $\overline{\text{EA}}$ during reset) 0: Internal ROM disabled: accesses to the ROM area use the external bus 1: Internal ROM enabled
SGTDIS	**Segmentation Disable/Enable Control** 0: Segmentation enabled (CSP is saved/restored during interrupt entry/exit) 1: Segmentation disabled (Only IP is saved/restored)
ROMS1	**Internal ROM Mapping** 0: Internal ROM area mapped to segment 0 (00’0000$_H$ … 00’7FFF$_H$) 1: Internal ROM area mapped to segment 1 (01’0000$_H$ … 01’7FFF$_H$)
STKSZ	**System Stack Size** Selects the size of the system stack (in the internal RAM) from 32 to 1024 words

Note: Register SYSCON cannot be changed after execution of the EINIT instruction.
Bit SGTDIS controls the correct stack operation (push/pop of CSP or not) during traps and interrupts.

The layout of the five BUSCON registers is identical. Registers BUSCON4 … BUSCON1, which control the selected address windows, are completely under software control, while register BUSCON0, which e.g. is also used for the very first code access after reset, is partly controlled by hardware, i.e. it is initialized via PORT0 during the reset sequence. This hardware control allows to define an appropriate external bus for systems, where no internal program memory is provided.

BUSCON0 (FF0C$_H$ / 86$_H$) SFR **Reset Value: 0XX0$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSWEN0	**CSREN0**	**-**	**RDYEN0**	**-**	**BUSACT0**	**ALECTL0**	**-**	**BTYP**		**MTTC0**	**RWDC0**	**MCTC**			
rw	rw	-	rw	-	rw	rw	-	rw		rw	rw	rw			

BUSCON1 (FF14$_H$ / 8A$_H$) SFR **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSWEN1	**CSREN1**	**-**	**RDYEN1**	**-**	**BUSACT1**	**ALECTL1**	**-**	**BTYP**		**MTTC1**	**RWDC1**	**MCTC**			
rw	rw	-	rw	-	rw	rw	-	rw		rw	rw	rw			

BUSCON2 (FF16$_H$ / 8B$_H$) SFR **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSWEN2	**CSREN2**	**-**	**RDYEN2**	**-**	**BUSACT2**	**ALECTL2**	**-**	**BTYP**		**MTTC2**	**RWDC2**	**MCTC**			
rw	rw	-	rw	-	rw	rw	-	rw		rw	rw	rw			

BUSCON3 (FF18$_H$ / 8C$_H$) SFR **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSWEN3	**CSREN3**	**-**	**RDYEN3**	**-**	**BUSACT3**	**ALECTL3**	**-**	**BTYP**		**MTTC3**	**RWDC3**	**MCTC**			
rw	rw	-	rw	-	rw	rw	-	rw		rw	rw	rw			

BUSCON4 (FF1A$_H$ / 8D$_H$) SFR **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSWEN4	**CSREN4**	**-**	**RDYEN4**	**-**	**BUSACT4**	**ALECTL4**	**-**	**BTYP**		**MTTC4**	**RWDC4**	**MCTC**			
rw	rw	-	rw	-	rw	rw	-	rw		rw	rw	rw			

Note: BUSCON0 is initialized with 0000$_H$, if pin $\overline{\text{EA}}$ is high during reset. If pin $\overline{\text{EA}}$ is low during reset, bits BUSACT0 and ALECTL0 are set (‘1’) and bit field BTYP is loaded with the bus configuration selected via PORT0.

Bit	Function
MCTC	**Memory Cycle Time Control** (Number of memory cycle time wait states) 0000: 15 waitstates (Number = 15 - <MCTC>) . . . 1111: No waitstates
RWDCx	**Read/Write Delay Control for BUSCONx** 0: With read/write delay: activate command 1 TCL after falling edge of ALE 1: No read/write delay: activate command with falling edge of ALE
MTTCx	**Memory Tristate Time Control** 0: 1 waitstate 1: No waitstate
BTYP	**External Bus Configuration** 00: 8-bit Demultiplexed Bus 01: 8-bit Multiplexed Bus 10: 16-bit Demultiplexed Bus 11: 16-bit Multiplexed Bus **Note:** For BUSCON0 BTYP is defined via PORT0 during reset.
ALECTLx	**ALE Lengthening Control** 0: Normal ALE signal 1: Lengthened ALE signal
BUSACTx	**Bus Active Control** 0: External bus disabled 1: External bus enabled (within the respective address window, see ADDRSEL)
RDYENx	**READY Input Enable** 0: External bus cycle is controlled by bit field MCTC only 1: External bus cycle is controlled by the $\overline{\text{READY}}$ input signal
CSRENx	**Read Chip Select Enable** 0: The $\overline{\text{CS}}$ signal is independent of the read command ($\overline{\text{RD}}$) 1: The $\overline{\text{CS}}$ signal is generated for the duration of the read command
CSWENx	**Write Chip Select Enable** 0: The $\overline{\text{CS}}$ signal is independent of the write command ($\overline{\text{WR}}$,$\overline{\text{WRL}}$,$\overline{\text{WRH}}$) 1: The $\overline{\text{CS}}$ signal is generated for the duration of the write command

ADDRSEL1 (FE18$_H$ / 0C$_H$) SFR **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RGSAD												**RGSZ**			
rw												rw			

ADDRSEL2 (FE1A$_H$ / 0D$_H$) SFR **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RGSAD												**RGSZ**			
rw												rw			

ADDRSEL3(FE1C$_H$ / 0E$_H$) SFR **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RGSAD												**RGSZ**			
rw												rw			

ADDRSEL4 (FE1E$_H$ / 0F$_H$) SFR **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RGSAD												**RGSZ**			
rw												rw			

Bit	Function
RGSZ	**Range Size Selection** Defines the size of the address area controlled by the respective BUSCONx/ ADDRSELx register pair. See table below.
RGSAD	**Range Start Address** Defines the upper bits of the start address (A23 …) of the respective address area. See table 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 to define 4 separate address areas within the address space of the C161RI. 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 in a way 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 are used (marked "R"), which are not implicitly used for addresses inside the window. The lower bits of the start address (marked "x") are disregarded.

Bit field RGSZ	Resulting Window Size	Relevant Bits (R) of Start Address (A23 … 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 C161RI's address space may partly overlap each other. Thus e.g. small areas may be cut out of bigger windows in order to effectively utilize external resources, especially within segment 0.

For each access the EBC compares the current address with all address select registers (programmable ADDRSELx and hardwired XADRSx). This comparison is done in four levels.

Priority 1: The hardwired XADRSx registers are evaluated first. A match with one of these registers directs the access to the respective X-Peripheral using the corresponding XBCONx register and ignoring all other ADDRSELx registers.

Priority 2: Registers ADDRSEL2 and ADDRSEL4 are evaluated before ADDRSEL1 and ADDRSEL3, respectively. A match with one of these registers directs the access to the respective external area using the corresponding BUSCONx register and ignoring registers ADDRSEL1/3 (see figure below).

Priority 3: A match with registers ADDRSEL1 or ADDRSEL3 directs the access to the respective external area using the corresponding BUSCONx register.

Priority 4: If there is no match with any XADRSx or ADDRSELx register the access to the external bus uses register BUSCON0.

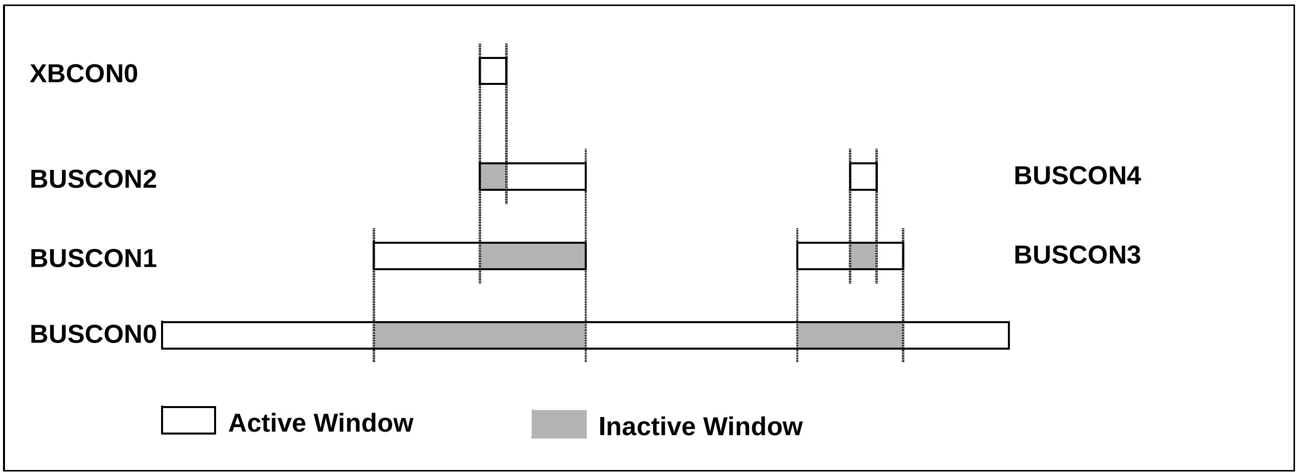


Figure 9-11
Address Window Arbitration

Note: Only the indicated overlaps are defined. All other overlaps lead to erroneous bus cycles. E.g. ADDRSEL4 may not overlap ADDRSEL2 or ADDRSEL1. The hardwired XADRSx registers are defined non-overlapping.

RP0H (F108$_H$ / 84$_H$) SFR **Reset Value: - - XX$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								CLKCFG			**SALSEL**		**CSSEL**		**WRC**
-	-	-	-	-	-	-	-	r			r		r		r

Bit	Function
WRC	**Write Configuration** 0: Pins $\overline{\text{WR}}$ and $\overline{\text{BHE}}$ operate as $\overline{\text{WRL}}$ and $\overline{\text{WRH}}$ signals 1: Pins $\overline{\text{WR}}$ and $\overline{\text{BHE}}$ operate as $\overline{\text{WR}}$ and $\overline{\text{BHE}}$ signals
CSSEL	**Chip Select Line Selection** (Number of active $\overline{\text{CS}}$ outputs) 00: 3 $\overline{\text{CS}}$ lines: $\overline{\text{CS2}}$ … $\overline{\text{CS0}}$ 01: 2 $\overline{\text{CS}}$ lines: $\overline{\text{CS1}}$ … $\overline{\text{CS0}}$ 10: No $\overline{\text{CS}}$ lines at all 11: 5 $\overline{\text{CS}}$ lines: $\overline{\text{CS4}}$ … $\overline{\text{CS0}}$ (Default without pulldowns)
SALSEL	**Segment Address Line Selection** (Number of active segment address 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 pulldowns)
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.

Note: RP0H cannot be changed via software, but rather allows to check the current configuration.

Precautions and Hints

- The external bus interface is enabled as long as at least one of the BUSCON registers has its BUSACT bit set.
- PORT1 will output the intra-segment address as long as at least one of the BUSCON registers selects a demultiplexed external bus, even for multiplexed bus cycles.
- Not all address windows defined via registers ADDRSELx may overlap each other. The operation of the EBC will be unpredictable in such a case. See chapter “Address Window Arbitration”.
- 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 EBC Idle State).

9.5 EBC Idle State

When the external bus interface is enabled, but no external access is currently executed, the EBC is idle. As long as only internal resources (from an architecture point of view) like IRAM, GPRs or SFRs, etc. are used the external bus interface does not change (see table below).

Accesses to on-chip X-Peripherals are also controlled by the EBC. However, even though an X-Peripheral appears like an external peripheral to the controller, the respective accesses do not generate valid external bus cycles.

Due to timing constraints address and write data of an XBUS cycle are reflected on the external bus interface (see table below). The “address” mentioned above includes PORT1, Port 4, $\overline{\text{BHE}}$ and ALE which also pulses for an XBUS cycle. The external $\overline{\text{CS}}$ signals on Port 6 are driven inactive (high) because the EBC switches to an internal $\overline{\text{XCS}}$ signal.

The **external control signals** ($\overline{\text{RD}}$ and $\overline{\text{WR}}$ or $\overline{\text{WRL}}$/$\overline{\text{WRH}}$ if enabled) **remain inactive** (high).

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)
Port 6	Active external $\overline{\text{CS}}$ signal corresponding to last used address	Inactive (high) for selected $\overline{\text{CS}}$ signals
$\overline{\text{BHE}}$	Level corresponding to last external access	Level corresponding to last XBUS access
ALE	Inactive (low)	Pulses as defined for X-Peripheral
$\overline{\text{RD}}$	Inactive (high)	**Inactive (high)**
$\overline{\text{WR}}$/$\overline{\text{WRL}}$	Inactive (high)	**Inactive (high)**
$\overline{\text{WRH}}$	Inactive (high)	**Inactive (high)**

9.6 The XBUS Interface

The C161RI provides an on-chip interface (the XBUS interface), which allows to connect integrated customer/application specific peripherals 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 hardwired register pair similar to registers BUSCON and ADDRSEL. As an interface to a peripheral in many cases is represented by just a few registers, the registers partly select smaller address windows than the standard ADDRSEL registers. As the register pairs control integrated peripherals rather than externally connected ones, they are fixed by mask programming rather than being user programmable.

X-Peripheral accesses provide the same choices as external accesses, so these peripherals may be bytewide or wordwide, with or without a separate address bus. Interrupt nodes are provided for X-Peripherals to be integrated.

Note: If you plan to develop a peripheral of your own to be integrated into a C161RI device to create a customer specific version, please ask for the specification of the XBUS interface and for further support.

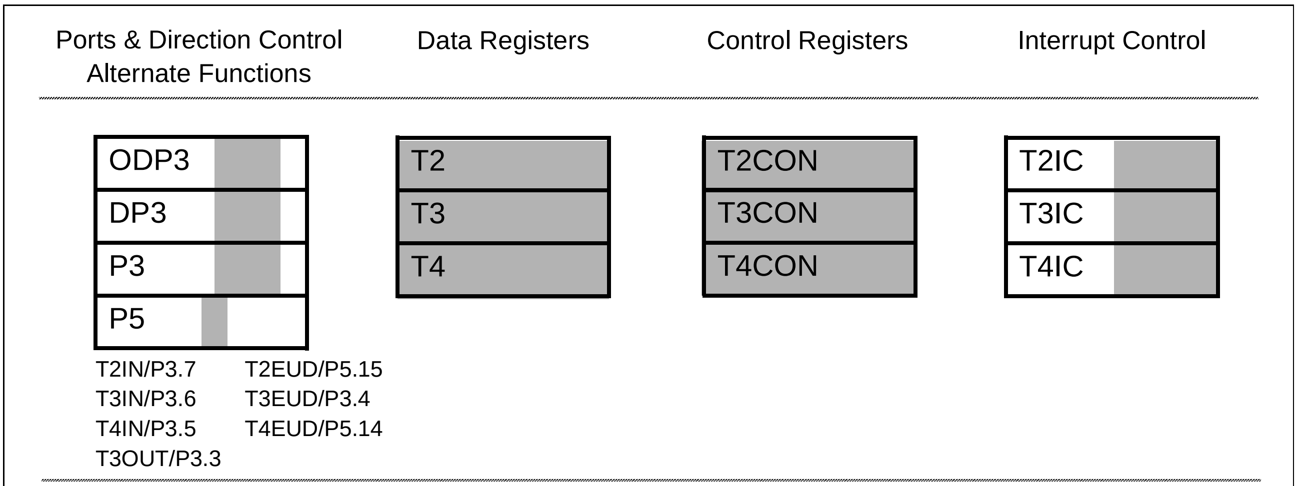
10 The General Purpose Timer Units

The General Purpose Timer Units GPT1 and GPT2 represent 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.

Block GPT1 contains 3 timers/counters with a maximum resolution of 16 TCL, while block GPT2 contains 2 timers/counters with a maximum resolution of 8 TCL and a 16-bit Capture/Reload register (CAPREL). 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. The auxiliary timers of GPT1 may optionally be configured as reload or capture registers for the core timer. In the GPT2 block, the additional CAPREL register supports capture and reload operation with extended functionality, and its core timer T6 may be concatenated with timers of the CAPCOM units (T0, T1, T7 and T8). Each block has alternate input/output functions and specific interrupts associated with it.

10.1 Timer Block GPT1

From a programmer's point of view, the GPT1 block is composed of a set of SFRs as summarized below. Those portions of port and direction registers which are used for alternate functions by the GPT1 block are shaded.



ODP3	Port 3 Open Drain Control Register	T2	GPT1 Timer 2 Register
DP3	Port 3 Direction Control Register	T3	GPT1 Timer 3 Register
P3	Port 3 Data Register	T4	GPT1 Timer 4 Register
T2CON	GPT1 Timer 2 Control Register	T2IC	GPT1 Timer 2 Interrupt Control Register
T3CON	GPT1 Timer 3 Control Register	T3IC	GPT1 Timer 3 Interrupt Control Register
T4CON	GPT1 Timer 4 Control Register	T4IC	GPT1 Timer 4 Interrupt Control Register

Figure 10-1
SFRs and Port Pins Associated with Timer Block GPT1

All three timers of block GPT1 (T2, T3, T4) can run in 4 basic modes, which are timer, gated timer, counter and incremental interface mode, and all timers can either count up or down. Each timer has an alternate input function pin (TxIN) associated with it which serves as the gate control in gated timer mode, or as the count input in counter mode. The count direction (Up / Down) may be programmed via software or may be dynamically altered by a signal at an external control input pin. Each overflow/underflow of core timer T3 is latched in the toggle FlipFlop T3OTL and may be indicated on an alternate output function pin. The auxiliary timers T2 and T4 may additionally be concatenated with the core timer, or used as capture or reload registers for the core timer.

The current contents of each timer can be read or modified by the CPU by accessing the corresponding timer registers T2, T3, or T4, which are located in the non-bitaddressable SFR space. When any of the timer registers is written to 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.

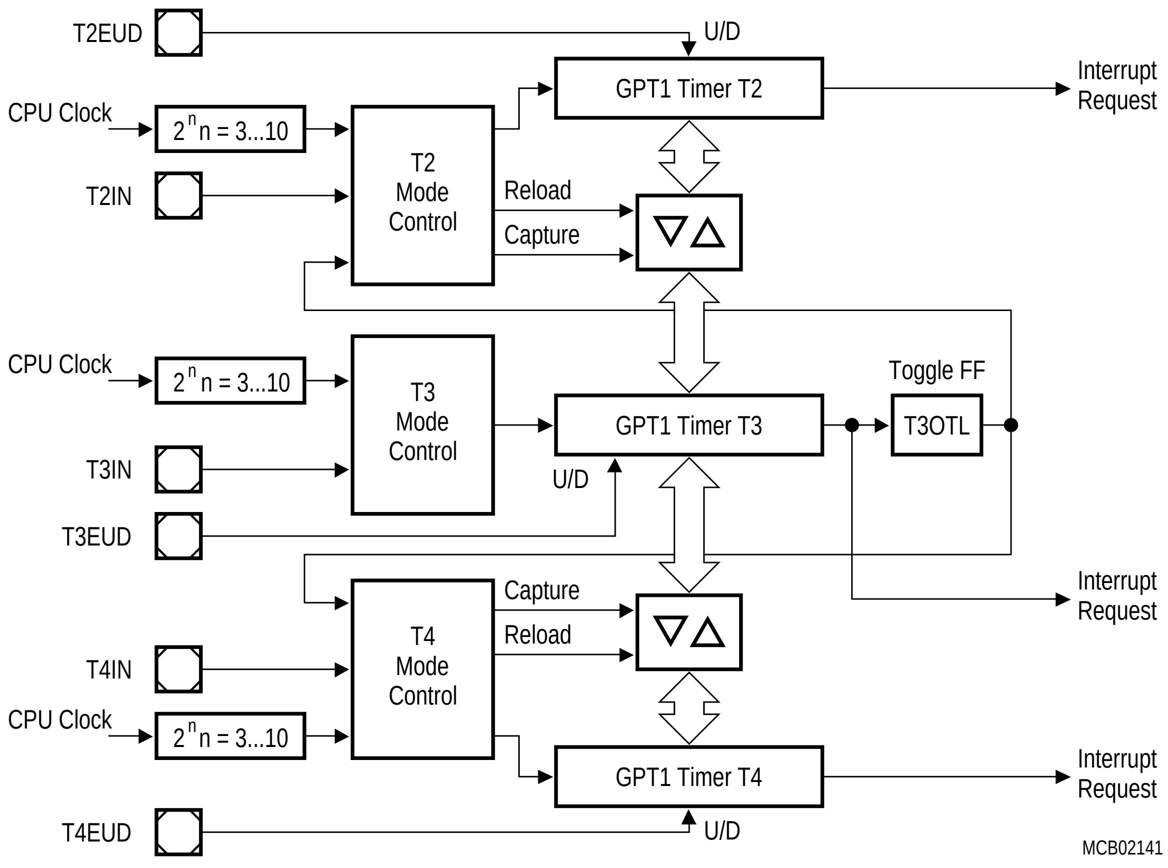


Figure 10-2
GPT1 Block Diagram

GPT1 Core Timer T3

The core timer T3 is configured and controlled via its bitaddressable control register T3CON.

T3CON (FF42$_H$ / A1$_H$) SFR **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	**-**	**-**	**-**	**-**	**T3 OTL**	**T3OE**	**T3 UDE**	**T3UD**	**T3R**	**T3M**			**T3I**		
-	-	-	-	-	rw	rw	rw	rw	rw	rw			rw		

Bit	Function
T3I	**Timer 3 Input Selection** Depends on the operating mode, see respective sections.
T3M	**Timer 3 Mode Control** (Basic Operating Mode) 0 0 0 : Timer Mode 0 0 1 : Counter Mode 0 1 0 : Gated Timer with Gate active low 0 1 1 : Gated Timer with Gate active high 1 0 0 : *Reserved*. Do not use this combination. 1 0 1 : *Reserved*. Do not use this combination. 1 1 0 : Incremental Interface Mode 1 1 1 : *Reserved*. Do not use this combination.
T3R	**Timer 3 Run Bit** T3R = ‘0’: Timer / Counter 3 stops T3R = ‘1’: Timer / Counter 3 runs
T3UD	**Timer 3 Up / Down Control** *)
T3UDE	**Timer 3 External Up/Down Enable** *)
T3OE	**Alternate Output Function Enable** T3OE = ‘0’: Alternate Output Function Disabled T3OE = ‘1’: Alternate Output Function Enabled
T3OTL	**Timer 3 Output Toggle Latch** Toggles on each overflow / underflow of T3. Can be set or reset by software.

*) For the effects of bits T3UD and T3UDE refer to the direction table below.

Timer 3 Run Bit

The timer can be started or stopped by software through bit T3R (Timer T3 Run Bit). If T3R = ‘0’, the timer stops. Setting T3R to ‘1’ will start the timer.
In gated timer mode, the timer will only run if T3R = ‘1’ and the gate is active (high or low, as programmed).

Count Direction Control

The count direction of the core timer can be controlled either by software or by the external input pin T3EUD (Timer T3 External Up/Down Control Input), which is the alternate input function of port pin P3.4. These options are selected by bits T3UD and T3UDE in control register T3CON. When the up/down control is done by software (bit T3UDE = '0'), the count direction can be altered by setting or clearing bit T3UD. When T3UDE = '1', pin T3EUD is selected to be the controlling source of the count direction. However, bit T3UD can still be used to reverse the actual count direction, as shown in the table below. If T3UD = '0' and pin T3EUD shows a low level, the timer is counting up. With a high level at T3EUD the timer is counting down. If T3UD = '1', a high level at pin T3EUD specifies counting up, and a low level specifies counting down. The count direction can be changed regardless of whether the timer is running or not.

When pin T3EUD/P3.4 is used as external count direction control input, it must be configured as input, i.e. its corresponding direction control bit DP3.4 must be set to '0'.

GPT1 Core Timer T3 Count Direction Control

Pin TxEUD	Bit TxUDE	Bit TxUD	Count Direction
X	0	0	Count Up
X	0	1	Count Down
0	1	0	Count Up
1	1	0	Count Down
0	1	1	Count Down
1	1	1	Count Up

Note: The direction control works the same for core timer T3 and for auxiliary timers T2 and T4. Therefore the pins and bits are named Tx …

Timer 3 Output Toggle Latch

An overflow or underflow of timer T3 will clock the toggle bit T3OTL in control register T3CON. T3OTL can also be set or reset by software. Bit T3OE (Alternate Output Function Enable) in register T3CON enables the state of T3OTL to be an alternate function of the external output pin T3OUT. For that purpose, a '1' must be written into the respective port data latch and pin T3OUT must be configured as output by setting the corresponding direction control bit to '1'. If T3OE = '1', pin T3OUT then outputs the state of T3OTL. If T3OE = '0', pin T3OUT can be used as general purpose IO pin.

In addition, T3OTL can be used in conjunction with the timer over/underflows as an input for the counter function or as a trigger source for the reload function of the auxiliary timers T2 and T4. For this purpose, the state of T3OTL does not have to be available at pin T3OUT, because an internal connection is provided for this option.

Timer 3 in Timer Mode

Timer mode for the core timer T3 is selected by setting bit field T3M in register T3CON to ‘000_B’. In this mode, T3 is clocked with the internal system clock (CPU clock) divided by a programmable prescaler, which is selected by bit field T3I. The input frequency f_{T3} for timer T3 and its resolution r_{T3} are scaled linearly with lower clock frequencies f_{CPU}, as can be seen from the following formula:

$$f_{T3} = \frac{f_{CPU}}{8 \times 2^{<T3I>}} \qquad r_{T3}\,[\mu s] = \frac{8 \times 2^{<T3I>}}{f_{CPU}\,[MHz]}$$

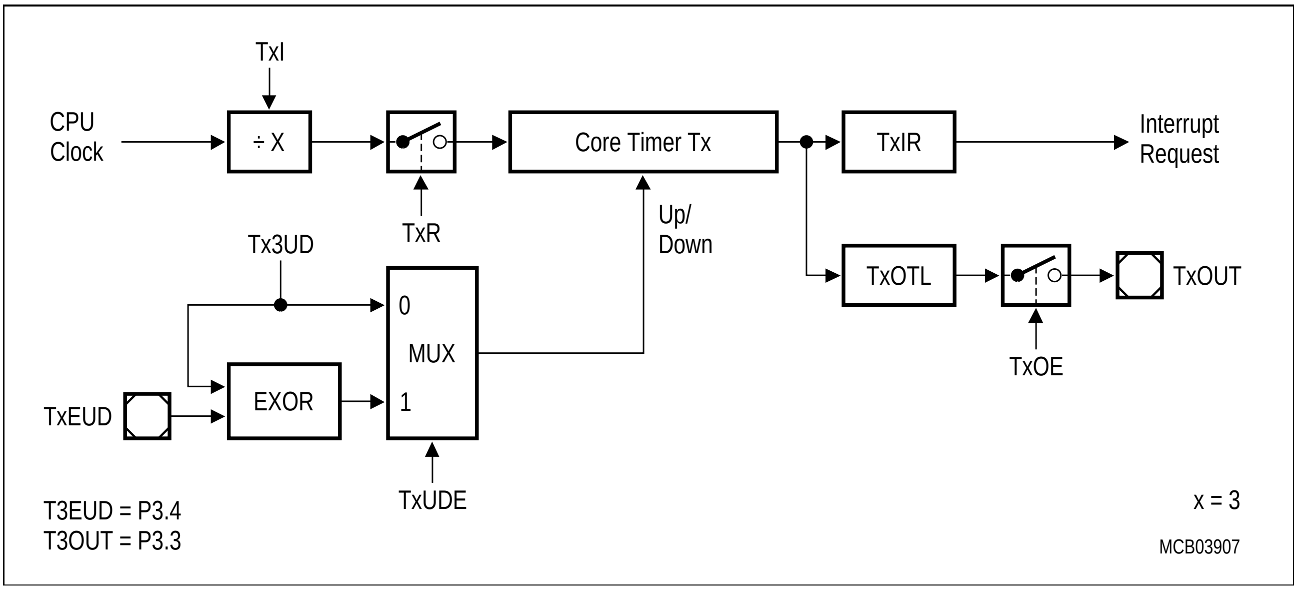


Figure 10-3
Block Diagram of Core Timer T3 in Timer Mode

The timer input frequencies, resolution and periods which result from the selected prescaler option are listed in the table below. This table also applies to the Gated Timer Mode of T3 and to the auxiliary timers T2 and T4 in timer and gated timer mode. Note that some numbers may be rounded to 3 significant digits.

GPT1 Timer Input Frequencies, Resolution and Periods

f_{CPU} = 20 MHz	Timer Input Selection T2I / T3I / T4I							
	000_B	001_B	010_B	011_B	100_B	101_B	110_B	111_B
Prescaler factor	8	16	32	64	128	256	512	1024
Input Frequency	2.5 MHz	1.25 MHz	625 kHz	312.5 kHz	156.25 kHz	78.125 kHz	39.06 kHz	19.53 kHz
Resolution	400 ns	800 ns	1.6 µs	3.2 µs	6.4 µs	12.8 µs	25.6 µs	51.2 µs
Period	26 ms	52.5 ms	105 ms	210 ms	420 ms	840 ms	1.68 s	3.36 s

Timer 3 in Gated Timer Mode

Gated timer mode for the core timer T3 is selected by setting bit field T3M in register T3CON to '010_B' or '011_B'. Bit T3M.0 (T3CON.3) selects the active level of the gate input. In gated timer mode the same options for the input frequency as for the timer mode are available. However, the input clock to the timer in this mode is gated by the external input pin T3IN (Timer T3 External Input).
To enable this operation pin T3IN must be configured as input, i.e. the corresponding direction control bit must contain '0'.

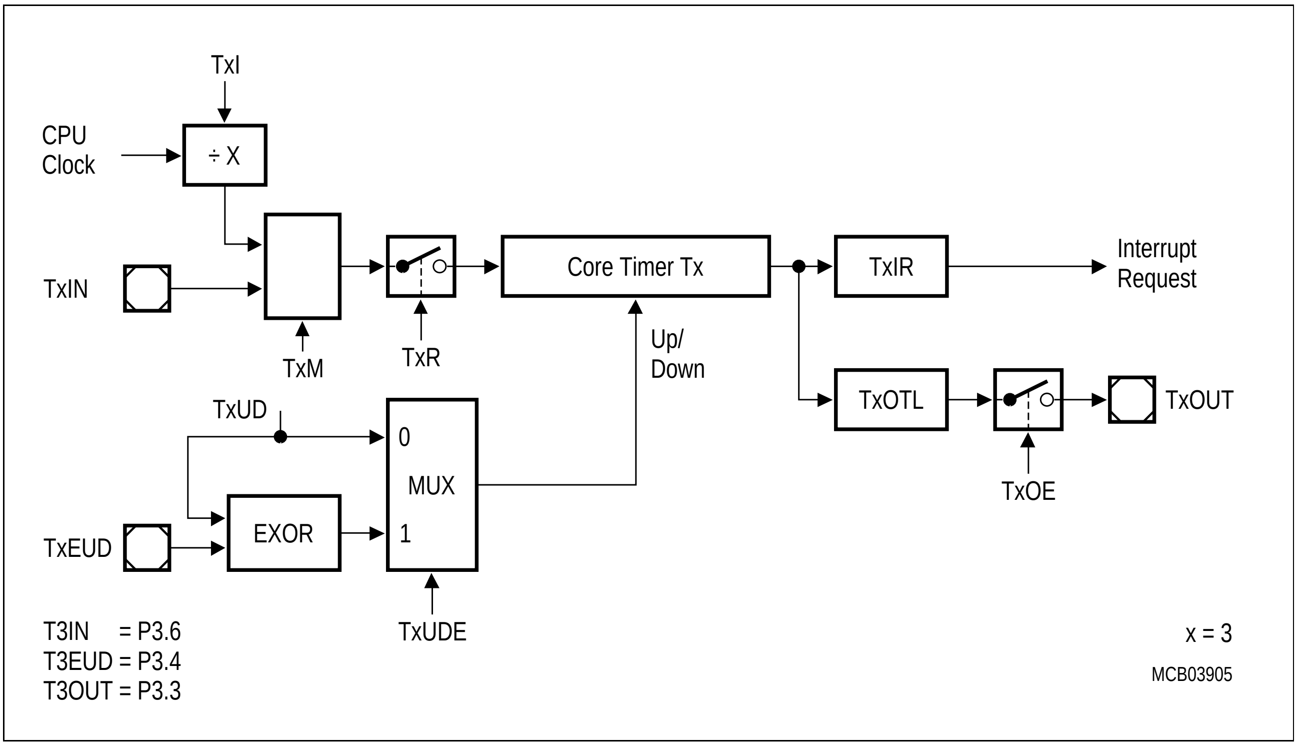


Figure 10-4
Block Diagram of Core Timer T3 in Gated Timer Mode

If T3M.0 = '0', the timer is enabled when T3IN shows a low level. A high level at this pin stops the timer. If T3M.0 = '1', pin T3IN must have a high level in order to enable the timer. In addition, the timer can be turned on or off by software using bit T3R. The timer will only run, if T3R = '1' and the gate is active. It will stop, if either T3R = '0' or the gate is inactive.

Note: A transition of the gate signal at pin T3IN does not cause an interrupt request.

Timer 3 in Counter Mode

Counter mode for the core timer T3 is selected by setting bit field T3M in register T3CON to ‘001_B’. In counter mode timer T3 is clocked by a transition at the external input pin T3IN. The event causing an increment or decrement of the timer can be a positive, a negative, or both a positive and a negative transition at this pin. Bit field T3I in control register T3CON selects the triggering transition (see table below).

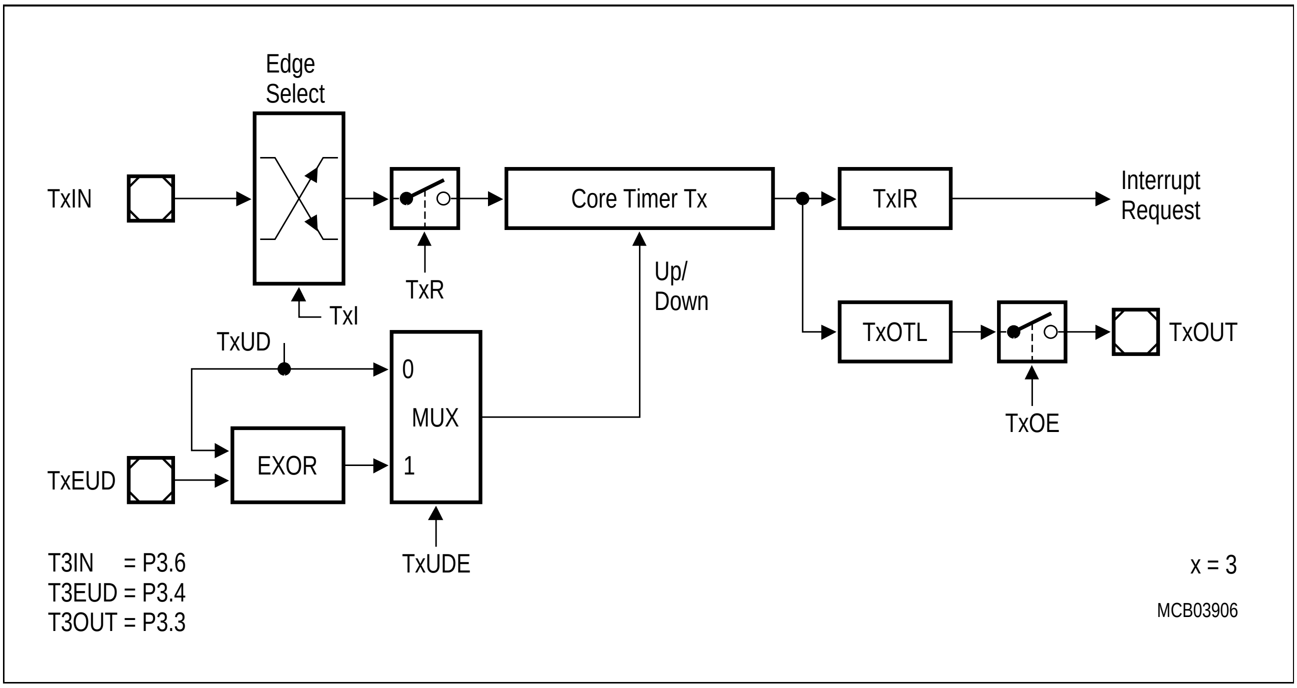


Figure 10-5
Block Diagram of Core Timer T3 in Counter Mode

GPT1 Core Timer T3 (Counter Mode) Input Edge Selection

T3I	Triggering Edge for Counter Increment / Decrement
0 0 0	None. Counter T3 is disabled
0 0 1	Positive transition (rising edge) on T3IN
0 1 0	Negative transition (falling edge) on T3IN
0 1 1	Any transition (rising or falling edge) on T3IN
1 X X	Reserved. Do not use this combination

For counter operation, pin T3IN must be configured as input, i.e. the respective direction control bit DPx.y must be ‘0’. The maximum input frequency which is allowed in counter mode is $f_{CPU}/16$. To ensure that a transition of the count input signal which is applied to T3IN is correctly recognized, its level should be held high or low for at least 8 f_{CPU} cycles before it changes.

Timer 3 in Incremental Interface Mode

Incremental Interface mode for the core timer T3 is selected by setting bit field T3M in register T3CON to '110_B'. In incremental interface mode the two inputs associated with timer T3 (T3IN, T3EUD) are used to interface to an incremental encoder. T3 is clocked by each transition on one or both of the external input pins which gives 2-fold or 4-fold resolution of the encoder input.

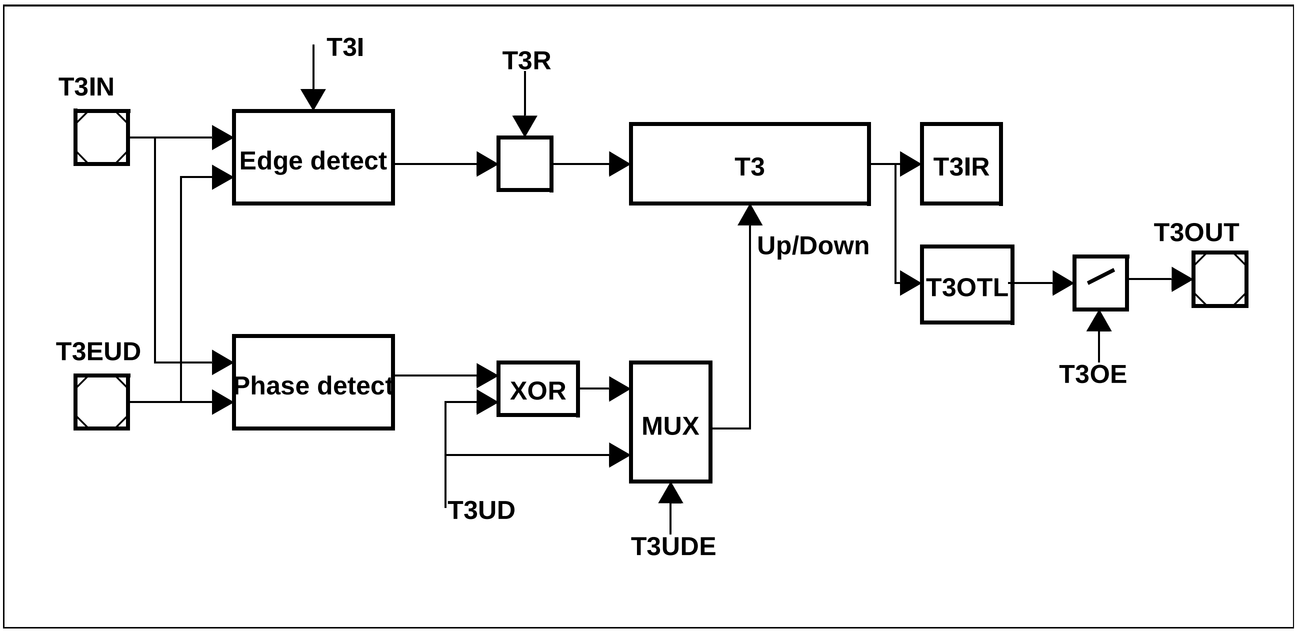


Figure 10-6
Block Diagram of Core Timer T3 in Incremental Interface Mode

Bitfield T3I in control register T3CON selects the triggering transitions (see table below). In this mode the sequence of the transitions of the two input signals is evaluated and generates count pulses as well as the direction signal. So T3 is modified automatically according to the speed and the direction of the incremental encoder and its contents therefore always represent the encoder's current position.

GPT1 Core Timer T3 (Incremental Interface Mode) Input Edge Selection

T3I	Triggering Edge for Counter Increment / Decrement
0 0 0	None. Counter T3 stops.
0 0 1	Any transition (rising or falling edge) on T3IN.
0 1 0	Any transition (rising or falling edge) on T3EUD.
0 1 1	Any transition (rising or falling edge) on any T3 input (T3IN or T3EUD).
1 X X	Reserved. Do not use this combination

The incremental encoder can be connected directly to the C161RI without external interface logic. In a standard system, however, comparators will be employed to convert the encoder's differential outputs (e.g. A, $\overline{A}$) to digital signals (e.g. A). This greatly increases noise immunity.

Note: The third encoder output Top0, which indicates the mechanical zero position, may be connected to an external interrupt input and trigger a reset of timer T3 (e.g. via PEC transfer from ZEROS).

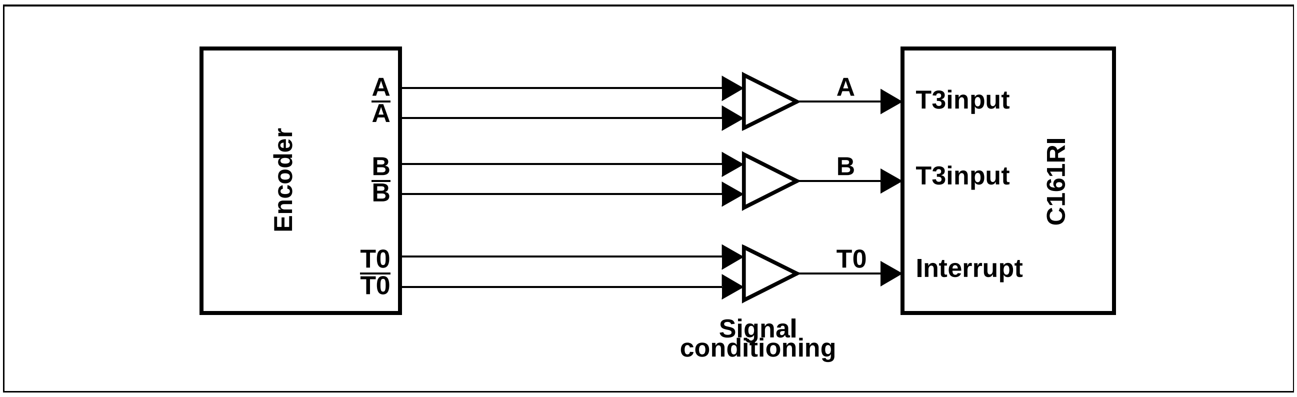


Figure 10-7
Connection of the Encoder to the C161RI

For incremental interface operation the following conditions must be met:

- Bitfield T3M must be '110_B'.
- Both pins T3IN and T3EUD must be configured as input, i.e. the respective direction control bits must be '0'.
- Bt be '1' to enable automatic direction control.

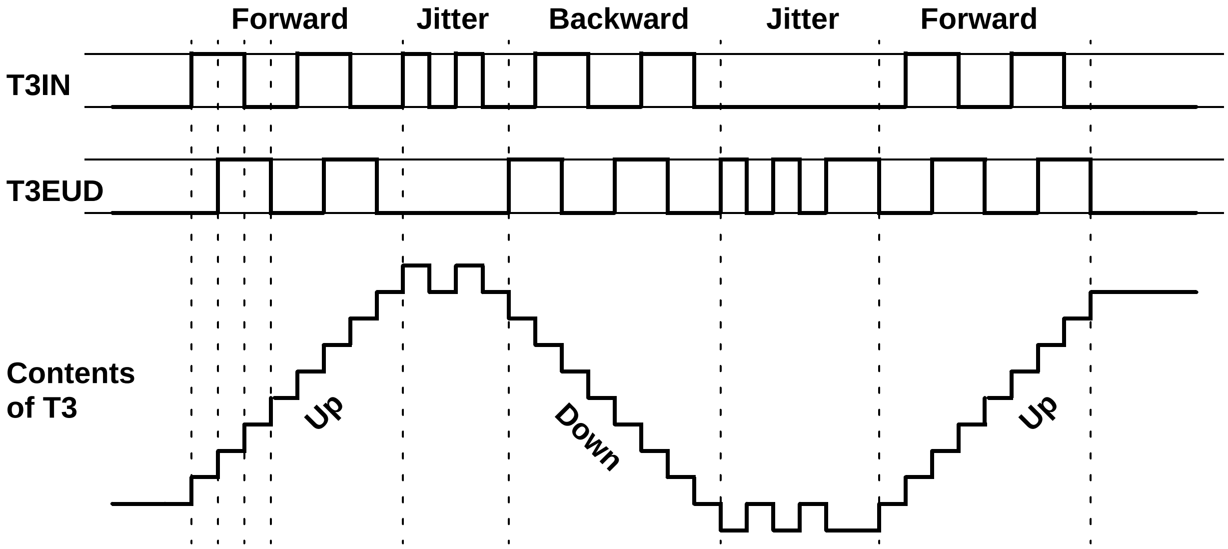
The maximum input frequency which is allowed in incremental interface mode is $f_{CPU}/16$. To ensure that a transition of any input signal is correctly recognized, its level should be held high or low for at least 8 f_{CPU} cycles before it changes.

In Incremental Interface Mode the count direction is automatically derived from the sequence in which the input signals change, which corresponds to the rotation direction of the connected sensor. The table below summarizes the possible combinations.

GPT1 Core Timer T3 (Incremental Interface Mode) Count Direction

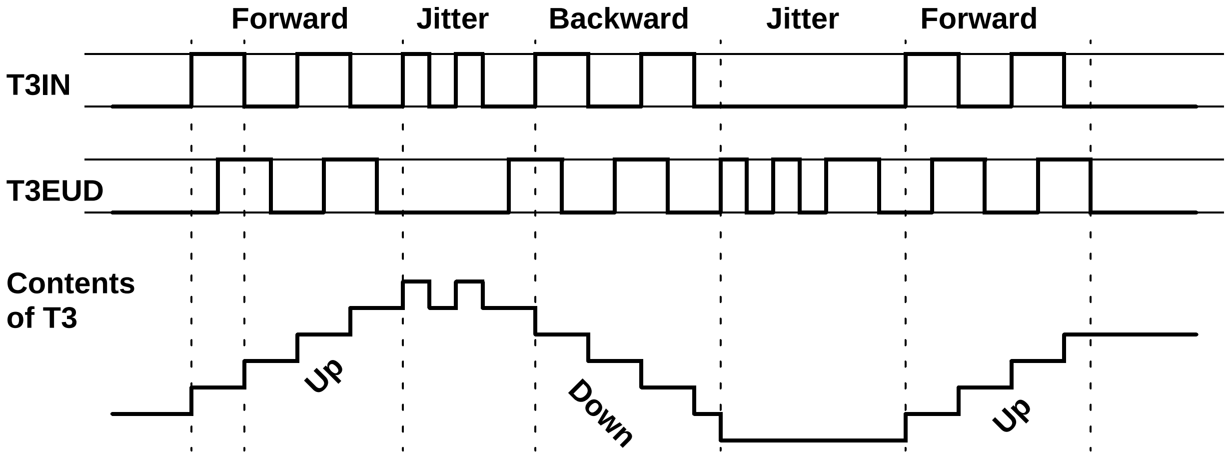
Level on respective other input	T3IN Input		T3EUD Input	
	Rising ↗	Falling ↘	Rising ↗	Falling ↘
High	Down	Up	Up	Down
Low	Up	Down	Down	Up

The figures below give examples of T3's operation, visualizing count signal generation and direction control. It also shows how input jitter is compensated which might occur if the sensor rests near to one of its switching points.



Note: This example shows the timer behaviour assuming that T3 counts upon any transition on any input, ie. T3I = ’011_B’.

Figure 10-8
Evaluation of the Incremental Encoder Signals



Note: This example shows the timer behaviour assuming that T3 counts upon any transition on input T3IN, ie. T3I = ’001_B’.

Figure 10-9
Evaluation of the Incremental Encoder Signals

Note: Timer T3 operating in incremental interface mode automatically provides information on the sensor’s current position. Dynamic information (speed, acceleration, deceleration) may be obtained by measuring the incoming signal periods. This is facilitated by an additional special capture mode for timer T5.

GPT1 Auxiliary Timers T2 and T4

Both auxiliary timers T2 and T4 have exactly the same functionality. They can be configured for timer, gated timer, counter, or incremental interface mode with the same options for the timer frequencies and the count signal as the core timer T3. In addition to these 4 counting modes, the auxiliary timers can be concatenated with the core timer, or they may be used as reload or capture registers in conjunction with the core timer.

The individual configuration for timers T2 and T4 is determined by their bitaddressable control registers T2CON and T4CON, which are both organized identically. Note that functions which are present in all 3 timers of block GPT1 are controlled in the same bit positions and in the same manner in each of the specific control registers.

T2CON (FF40$_H$ / A0$_H$) SFR **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	T2 UDE	T2UD	T2R	T2M			T2I		
-	-	-	-	-	-	-	rw	rw	rw	rw			rw		

T4CON (FF44$_H$ / A2$_H$) SFR **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	T4 UDE	T4UD	T4R	T4M			T4I		
-	-	-	-	-	-	-	rw	rw	rw	rw			rw		

Bit	Function
TxI	**Timer x Input Selection** Depends on the Operating Mode, see respective sections.
TxM	**Timer x Mode Control** (Basic Operating Mode) 0 0 0 : Timer Mode 0 0 1 : Counter Mode 0 1 0 : Gated Timer with Gate active low 0 1 1 : Gated Timer with Gate active high 1 0 0 : Reload Mode 1 0 1 : Capture Mode 1 1 0 : Incremental Interface Mode 1 1 1 : *Reserved*. Do not use this combination.
TxR	**Timer x Run Bit** TxR = ‘0’: Timer / Counter x stops TxR = ‘1’: Timer / Counter x runs
TxUD	**Timer x Up / Down Control** *)
TxUDE	**Timer x External Up/Down Enable** *)

*) For the effects of bits TxUD and TxUDE refer to the direction table (see T3 section).

Note: The auxiliary timers have no output toggle latch and no alternate output function.

Count Direction Control for Auxiliary Timers

The count direction of the auxiliary timers can be controlled in the same way as for the core timer T3. The description and the table apply accordingly.

Timers T2 and T4 in Timer Mode or Gated Timer Mode

When the auxiliary timers T2 and T4 are programmed to timer mode or gated timer mode, their operation is the same as described for the core timer T3. The descriptions, figures and tables apply accordingly with one exception:

- There is no output toggle latch for T2 and T4.

Timers T2 and T4 in Incremental Interface Mode

When the auxiliary timers T2 and T4 are programmed to incremental interface mode, their operation is the same as described for the core timer T3. The descriptions, figures and tables apply accordingly.

Timers T2 and T4 in Counter Mode

Counter mode for the auxiliary timers T2 and T4 is selected by setting bit field TxM in the respective register TxCON to '001_B'. In counter mode timers T2 and T4 can be clocked either by a transition at the respective external input pin TxIN, or by a transition of timer T3's output toggle latch T3OTL.

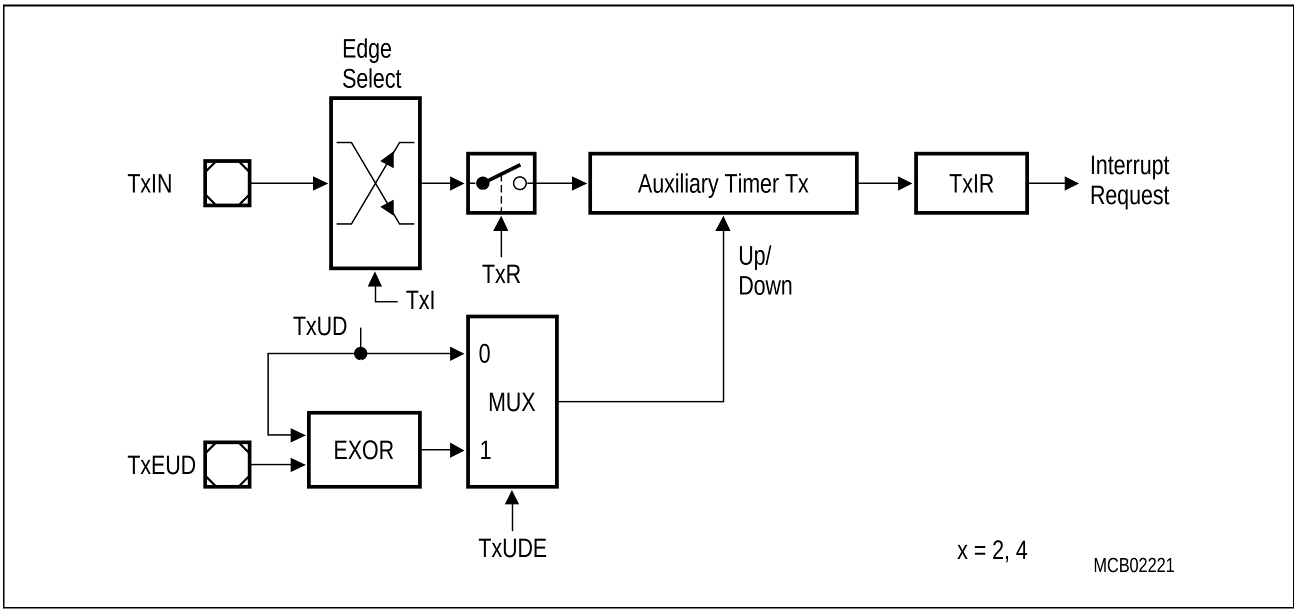


Figure 10-10
Block Diagram of an Auxiliary Timer in Counter Mode

The event causing an increment or decrement of a timer can be a positive, a negative, or both a positive and a negative transition at either the respective input pin, or at the toggle latch T3OTL. Bit field TxI in the respective control register TxCON selects the triggering transition (see table below).

GPT1 Auxiliary Timer (Counter Mode) Input Edge Selection

T2I / T4I	Triggering Edge for Counter Increment / Decrement
X 0 0	None. Counter Tx is disabled
0 0 1	Positive transition (rising edge) on TxIN
0 1 0	Negative transition (falling edge) on TxIN
0 1 1	Any transition (rising or falling edge) on TxIN
1 0 1	Positive transition (rising edge) of output toggle latch T3OTL
1 1 0	Negative transition (falling edge) of output toggle latch T3OTL
1 1 1	Any transition (rising or falling edge) of output toggle latch T3OTL

Note: Only state transitions of T3OTL which are caused by the overflows/underflows of T3 will trigger the counter function of T2/T4. Modifications of T3OTL via software will NOT trigger the counter function of T2/T4.

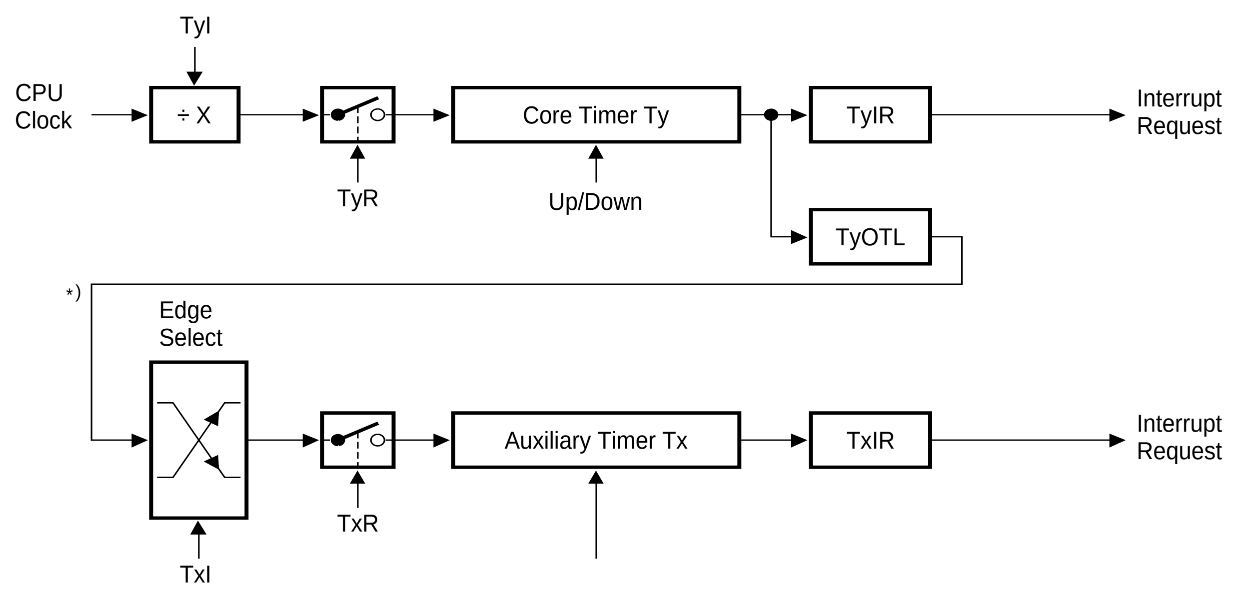
For counter operation, pin TxIN must be configured as input, i.e. the respective direction control bit must be ‘0’. The maximum input frequency which is allowed in counter mode is $f_{CPU}/16$. To ensure that a transition of the count input signal which is applied to TxIN is correctly recognized, its level should be held for at least 8 f_{CPU} cycles before it changes.

Timer Concatenation

Using the toggle bit T3OTL as a clock source for an auxiliary timer in counter mode concatenates the core timer T3 with the respective auxiliary timer. 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 is used to clock the auxiliary timer, this timer is clocked on every overflow/underflow of the core timer T3. Thus, the two timers form a 32-bit timer.
- **33-bit Timer/Counter**: If either a positive or a negative transition of T3OTL is selected to clock the auxiliary timer, this timer is clocked on every second overflow/underflow of the core timer T3. This configuration forms a 33-bit timer (16-bit core timer+T3OTL+16-bit auxiliary timer).

The count directions of the two concatenated timers are not required to be the same. This offers a wide variety of different configurations.
T3 can operate in timer, gated timer or counter mode in this case.



*) Line only affected by over/underflows of T3, but NOT by software modifications of T3OTL.

T3OUT = P3.3

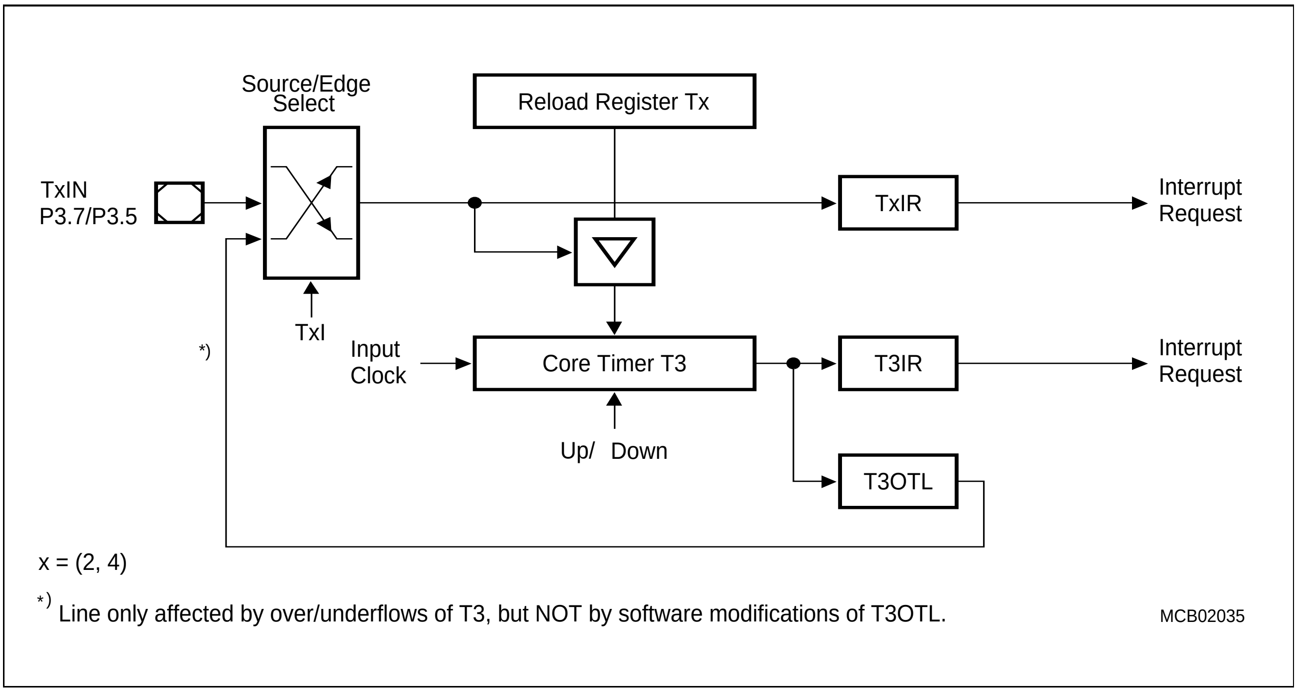
x = 2,4 y = 3

Figure 10-11
Concatenation of Core Timer T3 and an Auxiliary Timer

Auxiliary Timer in Reload Mode

Reload mode for the auxiliary timers T2 and T4 is selected by setting bit field TxM in the respective register TxCON to ‘100_B’. In reload mode the core timer T3 is reloaded with the contents of an auxiliary timer register, triggered by one of two different signals. The trigger signal is selected the same way as the clock source for counter mode (see table above), i.e. a transition of the auxiliary timer’s input or the output toggle latch T3OTL may trigger the reload.

Note: When programmed for reload mode, the respective auxiliary timer (T2 or T4) stops independent of its run flag T2R or T4R.



*) Line only affected by over/underflows of T3, but NOT by software modifications of T3OTL.

Figure 10-12
GPT1 Auxiliary Timer in Reload Mode

Upon a trigger signal T3 is loaded with the contents of the respective timer register (T2 or T4) and the interrupt request flag (T2IR or T4IR) is set.

Note: When a T3OTL transition is selected for the trigger signal, also 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.

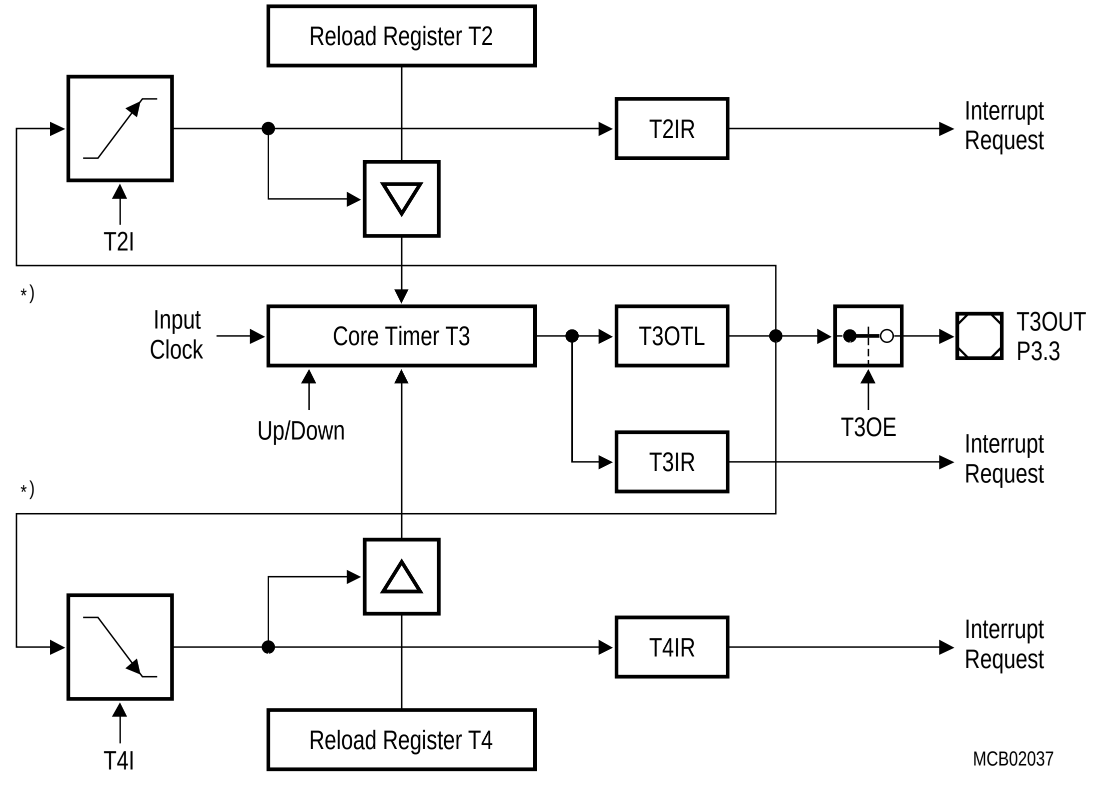
The reload mode triggered by T3OTL can be used in a number of different configurations. Depending on the selected active transition the following functions can be performed:

- If both a positive and a negative transition of T3OTL is 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 to perform very flexible pulse width modulation (PWM). One of the auxiliary timers is programmed to reload the core timer on a positive transition of T3OTL, the other is programmed for a reload on a negative transition of T3OTL. With this combination the core timer is alternately reloaded from the two auxiliary timers.

The figure below 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 T3OUT with T3OE = ‘1’, port latch = ‘1’ and direction bit = ‘1’. With this method the high and low time of the PWM signal can be varied in a wide range.

Note: The output toggle latch 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: Lines only affected by over/underflows of T3, but NOT by software modifications of T3OTL.

Figure 10-13
GPT1 Timer Reload Configuration for PWM Generation

Note: Although it is possible, 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 bit field TxM in the respective register TxCON to ‘101_B’. In capture mode the contents of the core timer are latched into an auxiliary timer register in response to a signal transition at the respective auxiliary timer's external input pin TxIN. The capture trigger signal can be a positive, a negative, or both a positive and a negative transition.

The two least significant bits of bit field TxI are used to select the active transition (see table in the counter mode section), 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 independent of its run flag T2R or T4R.

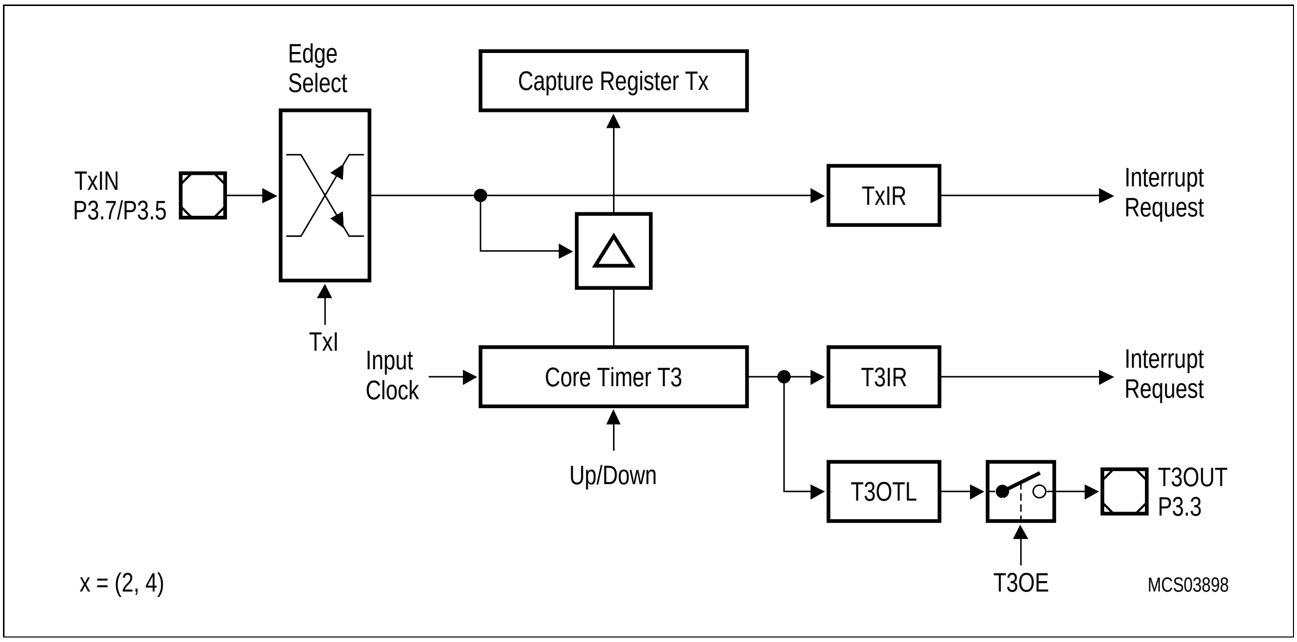


Figure 10-14
GPT1 Auxiliary Timer in Capture Mode

Upon a trigger (selected transition) at the corresponding input pin TxIN the contents of the core timer are loaded into the auxiliary timer register and the associated interrupt request flag TxIR will be set.

Note: The direction control bits for T2IN and T4IN must be set to '0', and the level of the capture trigger signal should be held high or low for at least 8 f_{CPU} cycles before it changes to ensure correct edge detection.

Interrupt Control for GPT1 Timers

When a timer overflows from FFFF$_H$ to 0000$_H$ (when counting up), or when it underflows from 0000$_H$ to FFFF$_H$ (when counting down), its interrupt request flag (T2IR, T3IR or T4IR) in register TxIC will be set. This will cause an interrupt to the respective timer interrupt vector (T2INT, T3INT or T4INT) or trigger a PEC service, if the respective interrupt enable bit (T2IE, T3IE or T4IE in register TxIC) is set. There is an interrupt control register for each of the three timers.

T2IC (FF60$_H$ / B0$_H$) SFR **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								T2IR	**T2IE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

T3IC (FF62$_H$ / B1$_H$) SFR **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								T3IR	**T3IE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

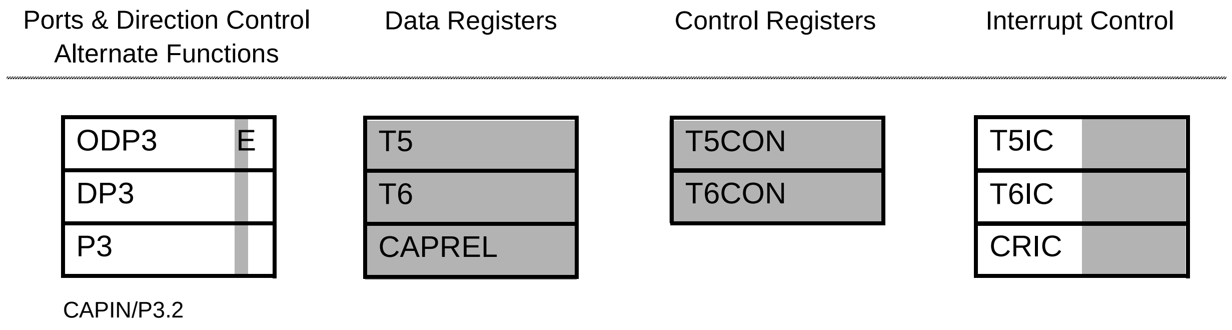
T4IC (FF64$_H$ / B2$_H$) SFR **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								T4IR	**T4IE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

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

10.2 Timer Block GPT2

From a programmer's point of view, the GPT2 block is represented by a set of SFRs as summarized below. Those portions of port and direction registers which are used for alternate functions by the GPT2 block are shaded.



ODP3	Port 3 Open Drain Control Register	T5	GPT2 Timer 5 Register
DP3	Port 3 Direction Control Register	T6	GPT2 Timer 6 Register
P3	Port 3 Data Register	CAPREL	GPT2 Capture/Reload Register
T5CON	GPT2 Timer 5 Control Register	T5IC	GPT2 Timer 5 Interrupt Control Register
T6CON	GPT2 Timer 6 Control Register	T6IC	GPT2 Timer 6 Interrupt Control Register
		CRIC	GPT2 CAPREL Interrupt Control Register

Figure 10-15
SFRs and Port Pins Associated with Timer Block GPT2

Timer block GPT2 supports high precision event control with a maximum resolution of 8 TCL. It includes the two timers T5 and T6, and the 16-bit capture/reload register CAPREL. Timer T6 is referred to as the core timer, and T5 is referred to as the auxiliary timer of GPT2.

The count direction (Up / Down) may be programmed via software. An overflow/underflow of T6 is indicated by the output toggle bit T6OTL. In addition, T6 may be reloaded with the contents of CAPREL.

The toggle bit also supports the concatenation of T6 with auxiliary timer T5. 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 T5 and T6.

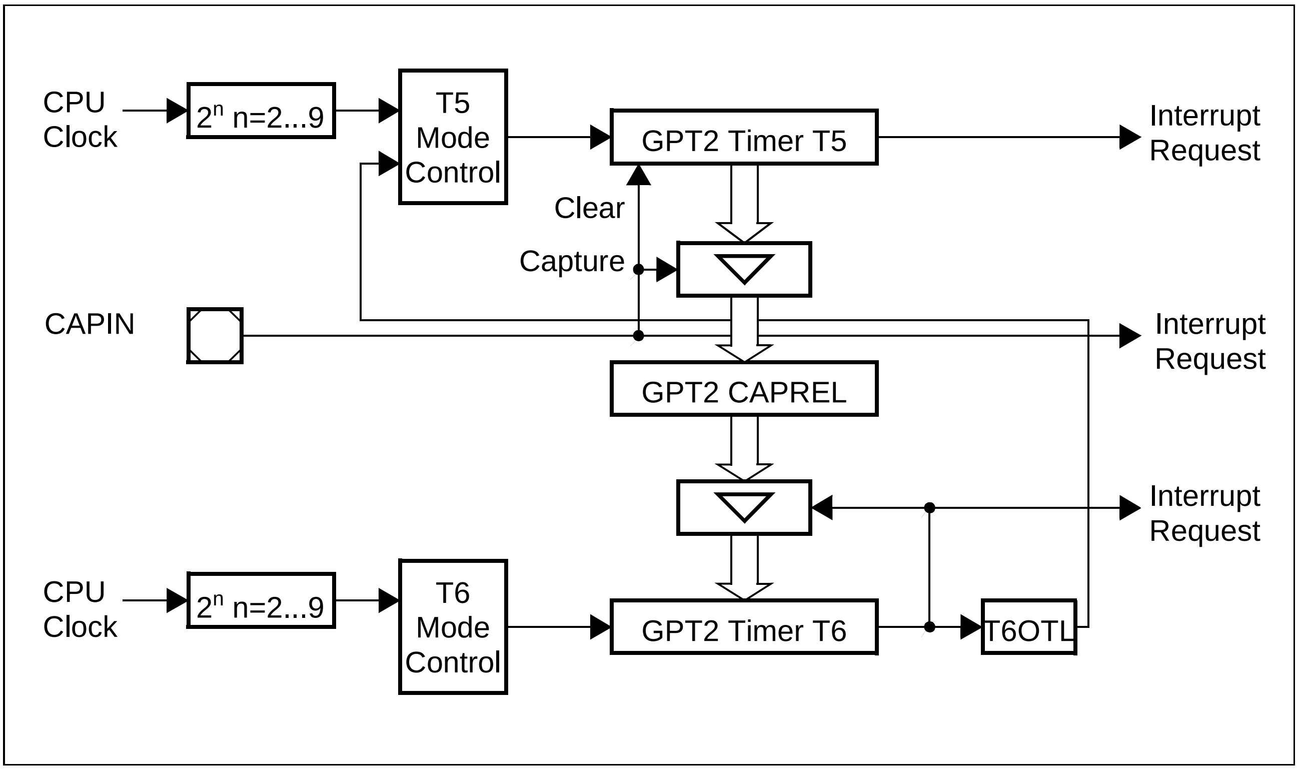


Figure 10-16
GPT2 Block Diagram

GPT2 Core Timer T6

The operation of the core timer T6 is controlled by its bitaddressable control register T6CON.

T6CON ($FF48_H$ / $A4_H$) SFR **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T6SR	**-**	**-**	**-**	**-**	**T6 OTL**	**-**	**-**	**T6UD**	**T6R**	**T6M**			**T6I**		
rw	-	-	-	-	rw	-	-	rw	rw	rw			rw		

Bit	Function
T6I	**Timer 6 Input Selection** Depends on the Operating Mode, see respective sections.
T6M	**Timer 6 Mode Control** (Basic Operating Mode) 0 0 0 : Timer Mode 0 0 1 : Reserved. Do not use this combination. 0 1 0 : Reserved. Do not use this combination. 0 1 1 : Reserved. Do not use this combination. 1 X X : Reserved. Do not use this combination.
T6R	**Timer 6 Run Bit** T6R = ‘0’: Timer / Counter 6 stops T6R = ‘1’: Timer / Counter 6 runs
T6UD	**Timer 6 Up / Down Control** T6UD = ‘0’: Timer / Counter 6 counts up T6UD = ‘1’: Timer / Counter 6 counts down
T6OTL	**Timer 6 Output Toggle Latch** Toggles on each overflow / underflow of T6. Can be set or reset by software.
T6SR	**Timer 6 Reload Mode Enable** T6SR = ‘0’: Reload from register CAPREL Disabled T6SR = ‘1’: Reload from register CAPREL Enabled

Timer 6 Run Bit

The timer can be started or stopped by software through bit T6R (Timer T6 Run Bit). If T6R = ‘0’, the timer stops. Setting T6R to ‘1’ will start the timer.
In gated timer mode, the timer will only run if T6R = ‘1’ and the gate is active (high or low, as programmed).

Timer 6 Output Toggle Latch

An overflow or underflow of timer T6 will clock the toggle bit T6OTL in control register T6CON. T6OTL can also be set or reset by software. T6OTL can be used in conjunction with the timer over/underflows as an input for the counter function of the auxiliary timer T5.

Timer 6 in Timer Mode

Timer mode for the core timer T6 is selected by setting bitfield T6M in register T6CON to '000_B'. In this mode, T6 is clocked with the internal system clock divided by a programmable prescaler, which is selected by bit field T6I. The input frequency f_{T6} for timer T6 and its resolution r_{T6} are scaled linearly with lower clock frequencies f_{CPU}, as can be seen from the following formula:

$$f_{T6} = \frac{f_{CPU}}{4 \times 2^{<T6I>}} \qquad r_{T6}\ [\mu s] = \frac{4 \times 2^{<T6I>}}{f_{CPU}\ [MHz]}$$

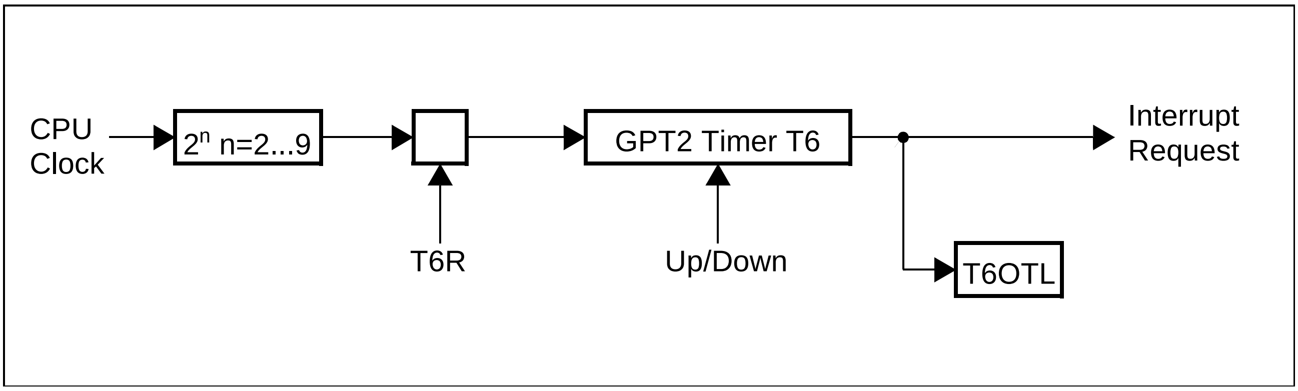


Figure 10-17
Block Diagram of Core Timer T6 in Timer Mode

The timer input frequencies, resolution and periods which result from the selected prescaler option are listed in the table below. This table also applies to the auxiliary timer T5 in timer mode. Note that some numbers may be rounded to 3 significant digits.

GPT2 Timer Input Frequencies, Resolution and Periods

f_{CPU} = 20 MHz	Timer Input Selection T5I / T6I							
	000_B	001_B	010_B	011_B	100_B	101_B	110_B	111_B
Prescaler factor	4	8	16	32	64	128	256	512
Input Frequency	5 MHz	2.5 MHz	1.25 MHz	625 kHz	312.5 kHz	156.25 kHz	78.125 kHz	39.06 kHz
Resolution	200 ns	400 ns	800 ns	1.6 µs	3.2 µs	6.4 µs	12.8 µs	25.6 µs
Period	13 ms	26 ms	52.5 ms	105 ms	210 ms	420 ms	840 ms	1.68 s

Note: Bitfield T6M in register T6CON will be '000_B' after reset. Do not modify this bitfield to any other value.

GPT2 Auxiliary Timer T5

The auxiliary timer T5 can be configured for timer mode with the same options for the timer frequencies as the core timer T6. In addition the auxiliary timer can be concatenated with the core timer (operation in counter mode). Its contents may be captured to register CAPREL upon a selectable trigger.

The individual configuration for timer T5 is determined by its bitaddressable control register T5CON. Note that functions which are present in both timers of block GPT2 are controlled in the same bit positions and in the same manner in each of the specific control registers.

Note: The auxiliary timer has no output toggle latch and no alternate output function.

T5CON ($FF46_H$ / $A3_H$) SFR **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T5SC	**T5 CLR**	**CI**		**-**	**CT3**	**-**	**T5 UDE**	**T5UD**	**T5R**	**-**	**T5M**		**T5I**		
rw	rw	rw		-	rw	-	rw	rw	rw	-	rw		rw		

Bit	Function
T5I	**Timer 5 Input Selection** Depends on the Operating Mode, see respective sections.
T5M	**Timer 5 Mode Control** (Basic Operating Mode) 0 0 : Timer Mode 0 1 : Counter Mode 1 0 : Reserved. Do not use this combination. 1 1 : Reserved. Do not use this combination.
T5R	**Timer 5 Run Bit** 0: Timer / Counter 5 stops 1: Timer / Counter 5 runs
T5UD	**Timer 5 Up / Down Control** T5UD = ‘0’: Timer / Counter 5 counts up T5UD = ‘1’: Timer / Counter 5 counts down
CT3	**Timer 3 Capture Trigger Enable** 0 : Capture trigger from pin CAPIN 1 : Capture trigger from T3 input pins
CI	**Register CAPREL Capture Trigger Selection** (depending on bit CT3) 0 0 : Capture disabled 0 1 : Positive transition (rising edge) on CAPIN or any transition on T3IN 1 0 : Negative transition (falling edge) on CAPIN or any transition on T3EUD 1 1 : Any transition (rising or falling edge) on CAPIN or any transition on T3IN or T3EUD

Bit	Function
T5CLR	**Timer 5 Clear Bit** 0: Timer 5 not cleared on a capture 1: Timer 5 is cleared on a capture
T5SC	**Timer 5 Capture Mode Enable** 0: Capture into register CAPREL disabled 1: Capture into register CAPREL enabled

Timer T5 in Counter Mode

Counter mode for the auxiliary timer T5 is selected by setting bit field T5M in register T5CON to '001_B'. In counter mode timer T5 can be clocked by a transition of timer T6's output toggle latch T6OTL (i.e. timer concatenation).

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 the toggle latch T6OTL.
Bit field T5I in control register T5CON selects the triggering transition (see table below).

GPT2 Auxiliary Timer (Counter Mode) Input Edge Selection

T5I	Triggering Edge for Counter Increment / Decrement
X 0 0	None. Counter T5 is disabled
0 0 1	Reserved. Do not use this combination.
0 1 0	Reserved. Do not use this combination.
0 1 1	Reserved. Do not use this combination.
1 0 1	Positive transition (rising edge) of output toggle latch T6OTL
1 1 0	Negative transition (falling edge) of output toggle latch T6OTL
1 1 1	Any transition (rising or falling edge) of output toggle latch T6OTL

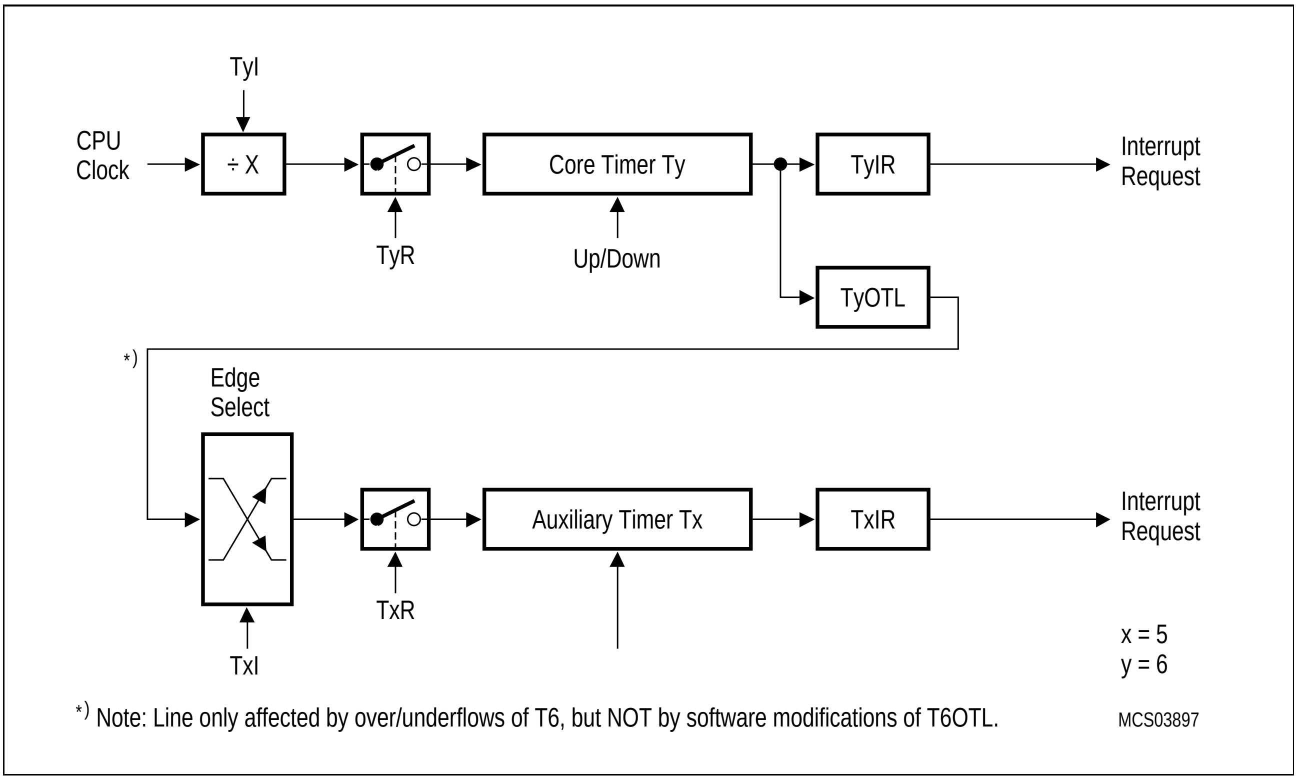
Note: Only state transitions of T6OTL which are caused by the overflows/underflows of T6 will trigger the counter function of T5. Modifications of T6OTL via software will NOT trigger the counter function of T5.

Timer Concatenation

Using the toggle bit T6OTL as a clock source for the auxiliary timer in counter mode concatenates the core timer T6 with the auxiliary timer. Depending on which transition of T6OTL 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 T6OTL is used to clock the auxiliary timer, this timer is clocked on every overflow/underflow of the 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 the auxiliary timer, this timer is clocked on every second overflow/underflow of the 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 of different configurations.



*) Note: Line only affected by over/underflows of T6, but NOT by software modifications of T6OTL.

Figure 10-18
Concatenation of Core Timer T6 and Auxiliary Timer T5

GPT2 Capture/Reload Register CAPREL in Capture Mode

This 16-bit register can be used as a capture register for the auxiliary timer T5. This mode is selected by setting bit T5SC = ‘1’ in control register T5CON. Bit CT3 selects the external input pin CAPIN or the input pins of timer T3 as the source for a capture trigger. Either a positive, a negative, or both a positive and a negative transition at pin 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 bit field CI in register T5CON.

The maximum input frequency for the capture trigger signal at CAPIN is $f_{CPU}/4$. To ensure that a transition of the capture trigger signal is correctly recognized, its level should be held for at least 4 f_{CPU} cycles before it changes.

When the timer T3 capture trigger is enabled (CT3 = ’1’) register CAPREL captures the contents of T5 upon transitions of the selected input(s). These values can be used to measure T3’s input signals. This is useful e.g. when T3 operates in incremental interface mode, in order to derive dynamic information (speed acceleration) from the input signals.

When a selected transition at the selected input pin(s) (CAPIN, T3IN, T3EUD) is detected, the contents of the auxiliary timer T5 are latched into register CAPREL, and interrupt request flag CRIR is set. With the same event, timer T5 can be cleared to 0000_H. This option is controlled by bit T5CLR in register T5CON. If T5CLR = ‘0’, the contents of timer T5 are not affected by a capture. If T5CLR = ‘1’, 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 = ‘0’, the selected trigger event can still be used to clear timer T5 or to generate an interrupt request. This interrupt is controlled by the CAPREL interrupt control register CRIC.

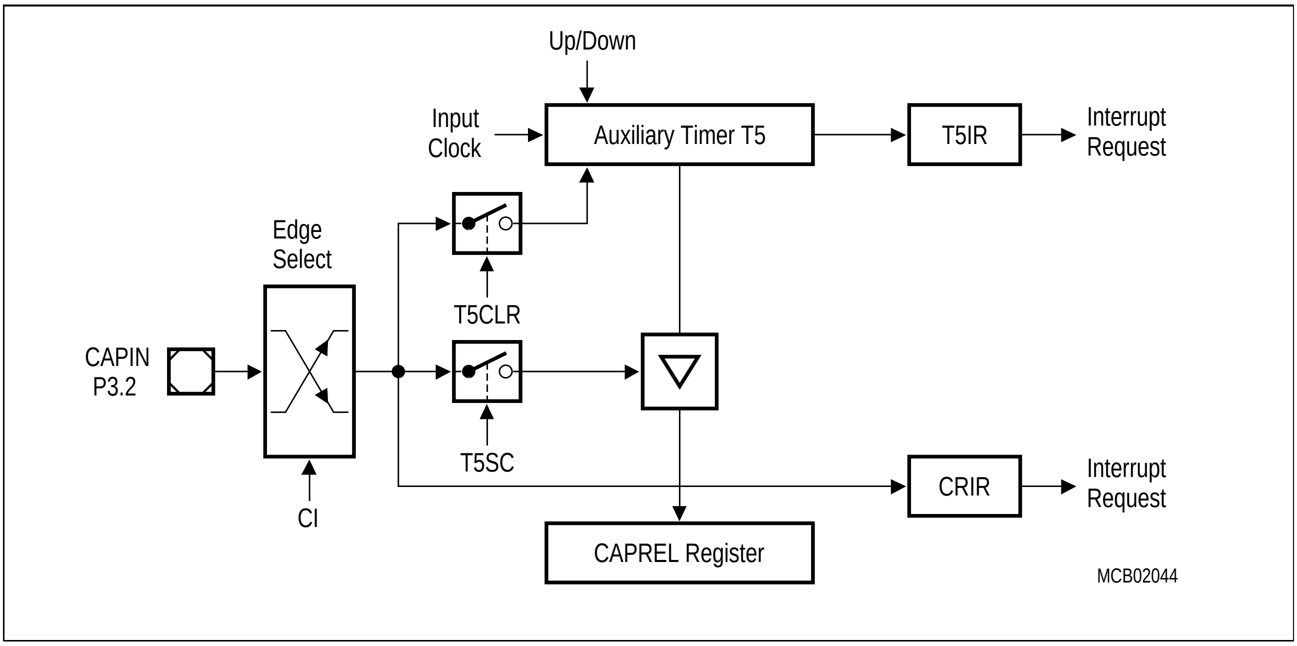


Figure 10-19
GPT2 Register CAPREL in Capture Mode

GPT2 Capture/Reload Register CAPREL in Reload Mode

This 16-bit register can be used as a reload register for the core timer T6. This mode is selected by setting bit T6SR = ‘1’ in register T6CON. The event causing a reload in this mode is an overflow or underflow of the core timer T6.

When timer T6 overflows from FFFF$_H$ to 0000$_H$ (when counting up) or when it underflows from 0000$_H$ to FFFF$_H$ (when counting down), the value stored in register CAPREL is loaded into timer T6. This will not set the interrupt request flag CRIR associated with the CAPREL register. However, interrupt request flag T6IR will be set indicating the overflow/underflow of T6.

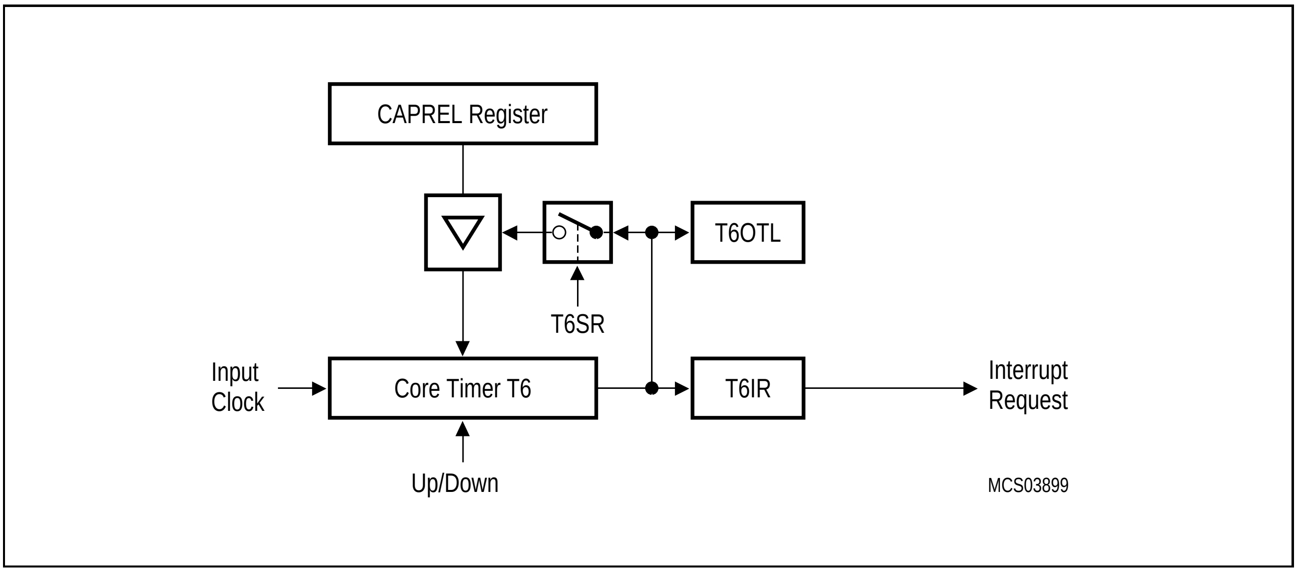


Figure 10-20
GPT2 Register CAPREL in Reload Mode

GPT2 Capture/Reload Register CAPREL in Capture-And-Reload Mode

Since the reload function and the capture function 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.

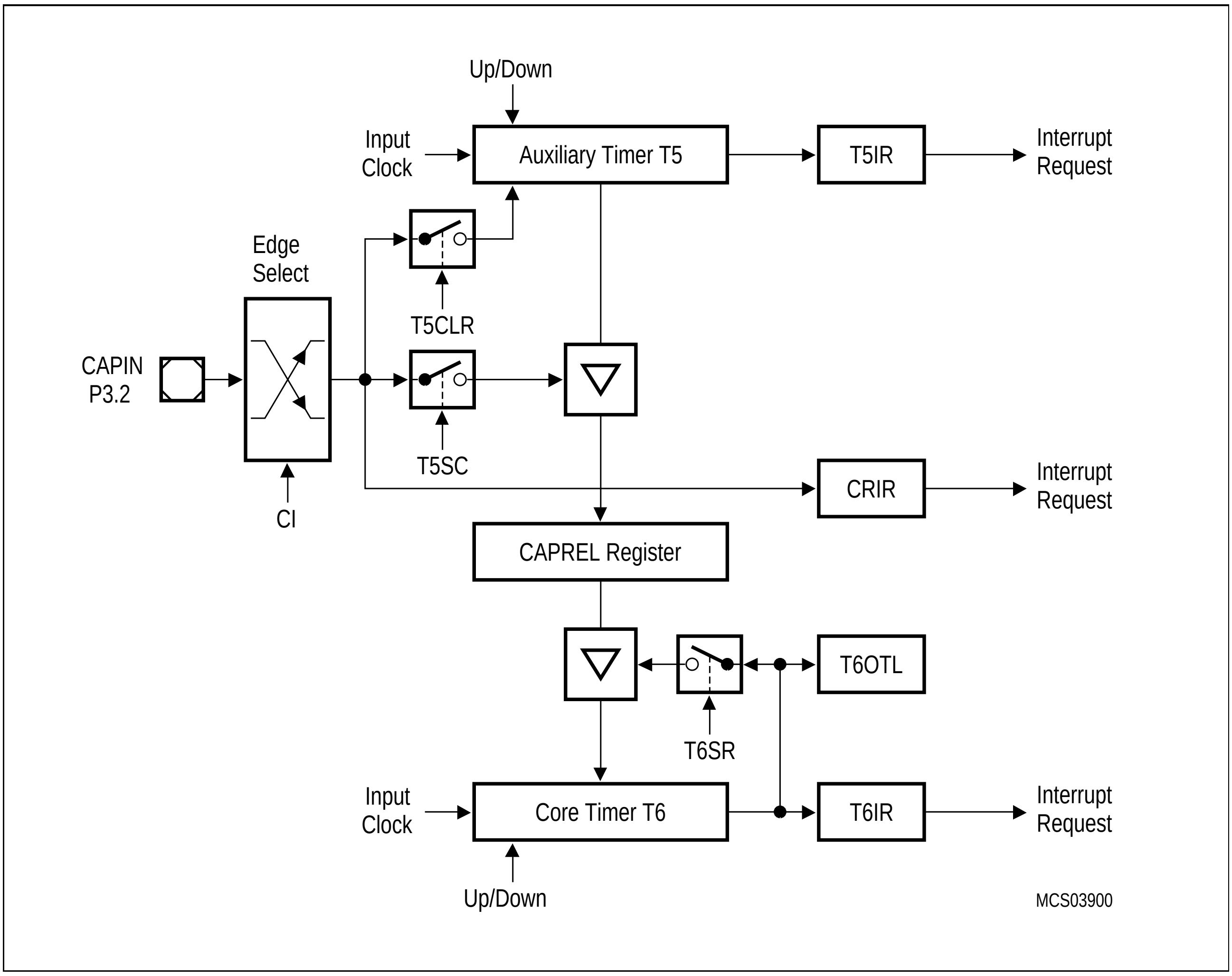


Figure 10-21
GPT2 Register CAPREL in Capture-And-Reload Mode

This combined mode can be used to detect consecutive external events which may occur aperiodically, but where a finer resolution, that means, more 'ticks' within the time between two external events is required.

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 e.g. f_{CPU}/32. The external events are applied to pin CAPIN. When an external event occurs, the timer T5 contents are latched into register CAPREL, and timer T5 is cleared (T5CLR = '1'). 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 e.g. f_{CPU}/4, uses the value in register CAPREL to perform a reload on underflow. This means, the value in register CAPREL represents the time between two underflows of timer T6, now measured in timer T6 increments. Since timer T6 runs 8 times faster than timer T5, it will underflow 8 times within the time between two external events. Thus, the underflow signal of timer T6 generates 8 'ticks'. Upon each underflow, the interrupt request flag T6IR will be set and bit T6OTL will be toggled.

Interrupt Control for GPT2 Timers and CAPREL

When a timer overflows from FFFF$_H$ to 0000$_H$ (when counting up), or when it underflows from 0000$_H$ to FFFF$_H$ (when counting down), its interrupt request flag (T5IR or T6IR) in register TxIC will be set. Whenever a transition according to the selection in bit field CI is detected at pin CAPIN, interrupt request flag CRIR in register CRIC is set. Setting any request flag will cause an interrupt to the respective timer or CAPREL interrupt vector (T5INT, T6INT or CRINT) or trigger a PEC service, if the respective interrupt enable bit (T5IE or T6IE in register TxIC, CRIE in register CRIC) is set. There is an interrupt control register for each of the two timers and for the CAPREL register.

T5IC (FF66$_H$ / B3$_H$) SFR **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								T5IR	**T5IE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

T6IC (FF68$_H$ / B4$_H$) SFR **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								T6IR	**T6IE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

CRIC (FF6A$_H$ / B5$_H$) SFR **Reset Value: - - 00$_H$**

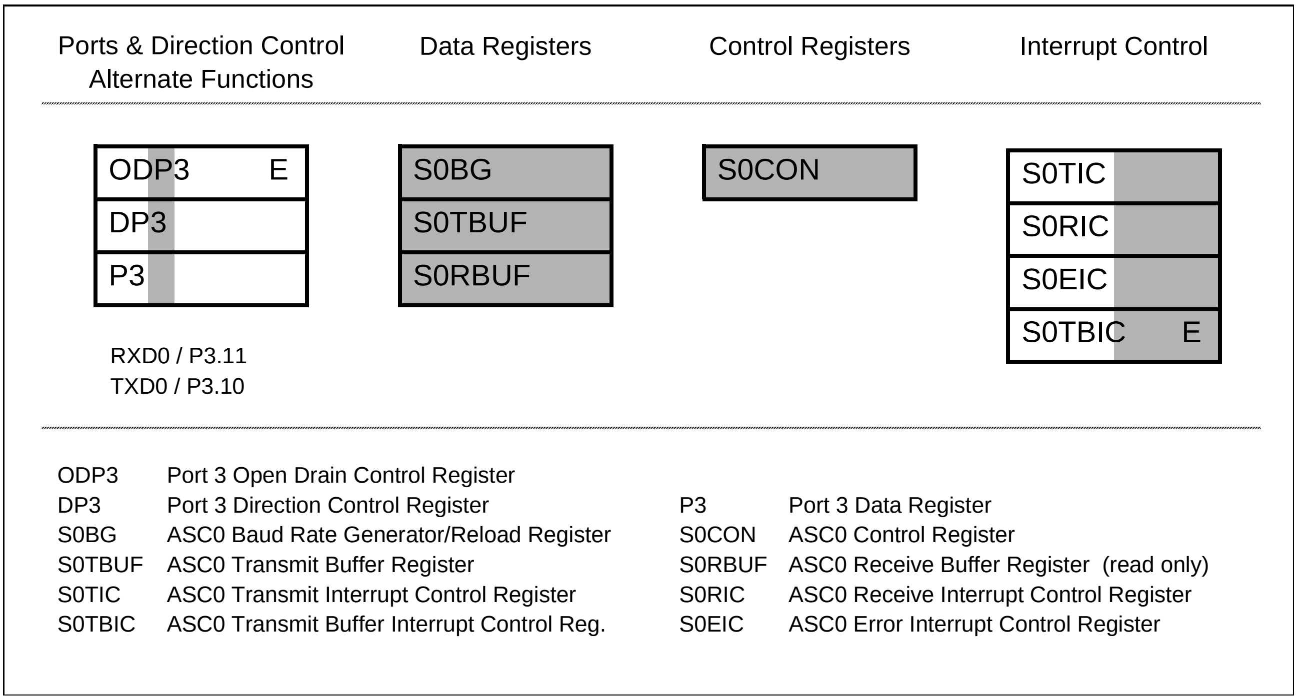
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								CRIR	**CRIE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

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

11 The Asynchronous/Synchronous Serial Interface

The Asynchronous/Synchronous Serial Interface ASC0 provides serial communication between the C161RI and other microcontrollers, microprocessors or external peripherals.

The ASC0 supports full-duplex asynchronous communication up to 500 KBaud and half-duplex synchronous communication up to 2 MBaud (@ 16 MHz CPU clock). In synchronous mode, data are transmitted or received synchronous to a shift clock which is generated by the C161RI. In asynchronous mode, 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 to distinguish address from data bytes is included. Testing is supported by a loop-back option. A 13-bit baud rate generator provides the ASC0 with a separate serial clock signal.



ODP3	Port 3 Open Drain Control Register
DP3	Port 3 Direction Control Register
S0BG	ASC0 Baud Rate Generator/Reload Register
S0TBUF	ASC0 Transmit Buffer Register
S0TIC	ASC0 Transmit Interrupt Control Register
S0TBIC	ASC0 Transmit Buffer Interrupt Control Reg.
P3	Port 3 Data Register
S0CON	ASC0 Control Register
S0RBUF	ASC0 Receive Buffer Register (read only)
S0RIC	ASC0 Receive Interrupt Control Register
S0EIC	ASC0 Error Interrupt Control Register

Figure 11-1
SFRs and Port Pins associated with ASC0

The operating mode of the serial channel ASC0 is controlled by its bitaddressable control register S0CON. This register contains control bits for mode and error check selection, and status flags for error identification.

S0CON (FFB0$_H$ / D8$_H$) SFR **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S0R	**S0LB**	**S0 BRS**	**S0 ODD**	**-**	**S0OE**	**S0FE**	**S0PE**	**S0 OEN**	**S0 FEN**	**S0 PEN**	**S0 REN**	**S0 STP**	**S0M**		
rw	rw	rw	rw	-	rw	rw	rw	rw	rw	rw	rw	rw	rw		

Bit	Function
S0M	**ASC0 Mode Control** 0 0 0 : 8-bit data synchronous operation 0 0 1 : 8-bit data async. operation 0 1 0 : Reserved. Do not use this combination! 0 1 1 : 7-bit data + parity async. operation 1 0 0 : 9-bit data async. operation 1 0 1 : 8-bit data + wake up bit async. operation 1 1 0 : Reserved. Do not use this combination! 1 1 1 : 8-bit data + parity async. operation
S0STP	**Number of Stop Bits Selection** async. operation 0 : One stop bit 1 : Two stop bits
S0REN	**Receiver Enable Bit** 0 : Receiver disabled 1 : Receiver enabled (Reset by hardware after reception of byte in synchronous mode)
S0PEN	**Parity Check Enable Bit** async. operation 0 : Ignore parity 1 : Check parity
S0FEN	**Framing Check Enable Bit** async. operation 0 : Ignore framing errors 1 : Check framing errors
S0OEN	**Overrun Check Enable Bit** 0 : Ignore overrun errors 1 : Check overrun errors
S0PE	**Parity Error Flag** Set by hardware on a parity error (S0PEN = ’1’). Must be reset by software.
S0FE	**Framing Error Flag** Set by hardware on a framing error (S0FEN = ’1’). Must be reset by software.
S0OE	**Overrun Error Flag** Set by hardware on an overrun error (S0OEN = ’1’). Must be reset by software.
S0ODD	**Parity Selection Bit** 0 : Even parity (parity bit set on odd number of ‘1’s in data) 1 : Odd parity (parity bit set on even number of ‘1’s in data)

Bit	Function
S0BRS	**Baudrate Selection Bit** 0 : Divide clock by reload-value + constant (depending on mode) 1 : Additionally reduce serial clock to 2/3rd
S0LB	**LoopBack Mode Enable Bit** 0 : Standard transmit/receive mode 1 : Loopback mode enabled
S0R	**Baudrate Generator Run Bit** 0 : Baudrate generator disabled (ASC0 inactive) 1 : Baudrate generator enabled

A transmission is started by writing to the Transmit Buffer register S0TBUF (via an instruction or a PEC data transfer). Only the number of data bits which is determined by the selected operating mode will actually be transmitted, i.e. bits written to positions 9 through 15 of register S0TBUF are always insignificant. After a transmission has been completed, the transmit buffer register is cleared to 0000_H.
Data transmission is double-buffered, so a new character may be written to the transmit buffer register, before the transmission of the previous character is complete. This allows the transmission of characters back-to-back without gaps.

Data reception is enabled by the Receiver Enable Bit S0REN. After reception of a character has been completed, the received data and, if provided by the selected operating mode, the received parity bit can be read from the (read-only) Receive Buffer register S0RBUF. Bits in the upper half of S0RBUF which 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 buffer overrun error detection can be selected through bit S0OEN. When enabled, the overrun error status flag S0OE and the error interrupt request flag S0EIR will be set when the receive buffer register has not been read by the time reception of a second character is complete. The previously received character in the receive buffer is overwritten.

The Loop-Back option (selected by bit S0LB) allows the data currently being transmitted to be received simultaneously in the receive buffer. This may be used to test serial communication routines at an early stage without having to provide an external network. In loop-back mode the alternate input/output functions of the Port 3 pins are not necessary.

Note: Serial data transmission or reception is only possible when the Baud Rate Generator Run Bit S0R is set to ‘1’. Otherwise the serial interface is idle.
Do not program the mode control field S0M in register S0CON to one of the reserved combinations to avoid unpredictable behavior of the serial interface.

11.1 Asynchronous Operation

Asynchronous mode supports full-duplex communication, where both transmitter and receiver use the same data frame format and the same baud rate. Data is transmitted on pin TXD0/P3.10 and received on pin RXD0/P3.11. These signals are alternate functions of Port 3 pins.

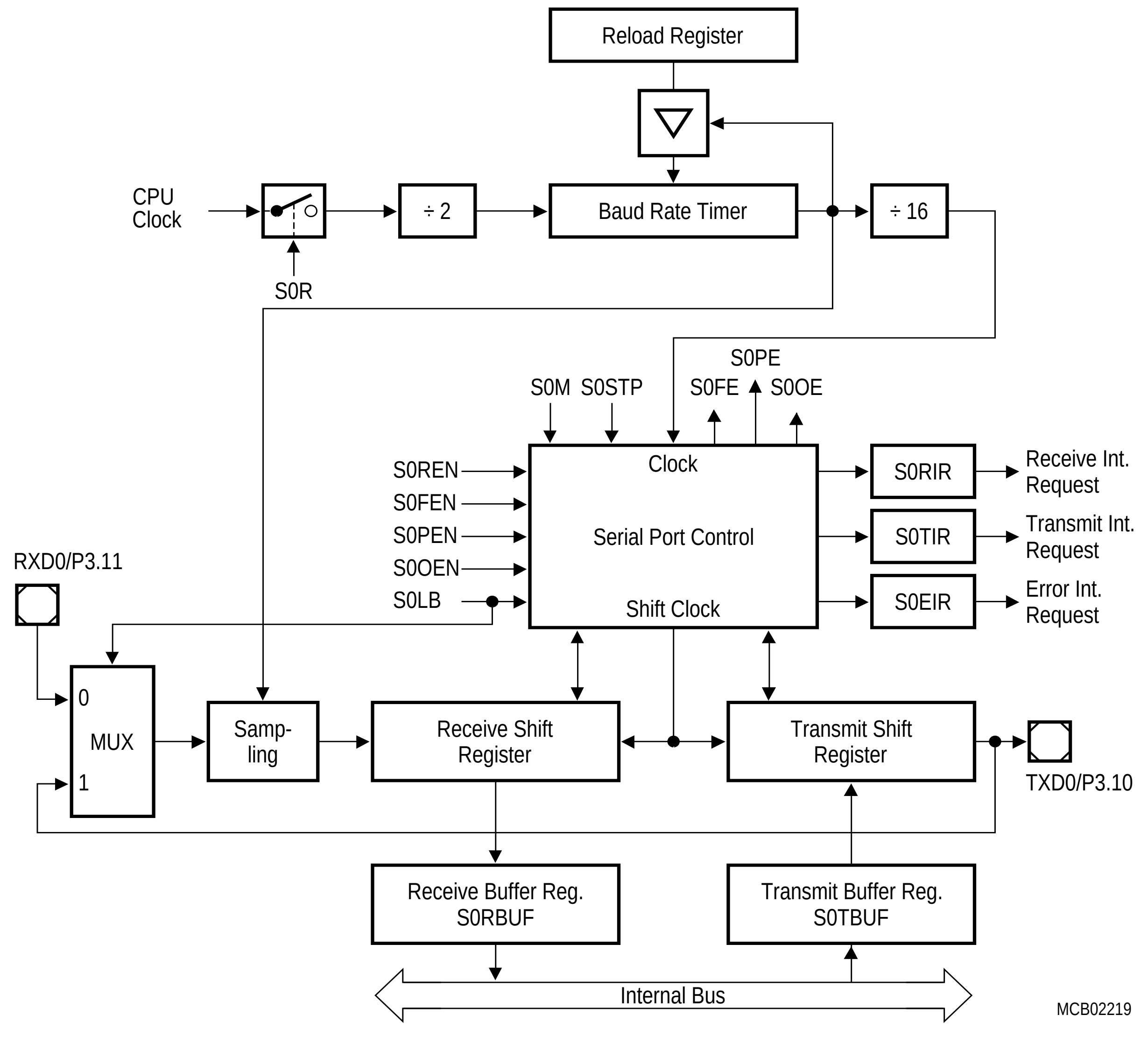


Figure 11-2
Asynchronous Mode of Serial Channel ASC0

Asynchronous Data Frames

8-bit data frames either consist of 8 data bits D7 … D0 (S0M = ’001_B’), or of 7 data bits D6 … D0 plus an automatically generated parity bit (S0M = ’011_B’). Parity may be odd or even, depending on bit S0ODD in register S0CON. An even parity bit will be set, if the modulo-2-sum of the 7 data bits is ‘1’. An odd parity bit will be cleared in this case. Parity checking is enabled via bit S0PEN (always OFF in 8-bit data mode). The parity error flag S0PE will be set along with the error interrupt request flag, if a wrong parity bit is received. The parity bit itself will be stored in bit S0RBUF.7.

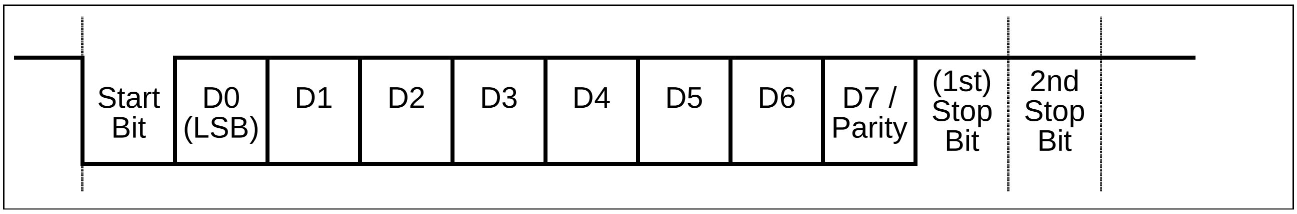


Figure 11-3
Asynchronous 8-bit Data Frames

9-bit data frames either consist of 9 data bits D8 … D0 (S0M = ’100_B’), of 8 data bits D7 … D0 plus an automatically generated parity bit (S0M = ’111_B’) or of 8 data bits D7 … D0 plus wake-up bit (S0M = ’101_B’). Parity may be odd or even, depending on bit S0ODD in register S0CON. An even parity bit will be set, if the modulo-2-sum of the 8 data bits is ‘1’. An odd parity bit will be cleared in this case. Parity checking is enabled via bit S0PEN (always OFF in 9-bit data and wake-up mode). The parity error flag S0PE will be set along with the error interrupt request flag, if a wrong parity bit is received. The parity bit itself will be stored in bit S0RBUF.8.

In wake-up mode received frames are only transferred to the receive buffer register, 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 multi-processor system:
When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address byte which identifies the target slave. An address byte differs from a data byte in that the additional 9th bit is a '1' for an address byte and 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 8 LSBs of the received character (the address). The addressed slave will switch to 9-bit data mode (e.g. by clearing bit S0M.0), which enables it to also receive the data bytes that will be coming (having the wake-up bit cleared). The slaves that were not being addressed remain in 8-bit data + wake-up bit mode, ignoring the following data bytes.

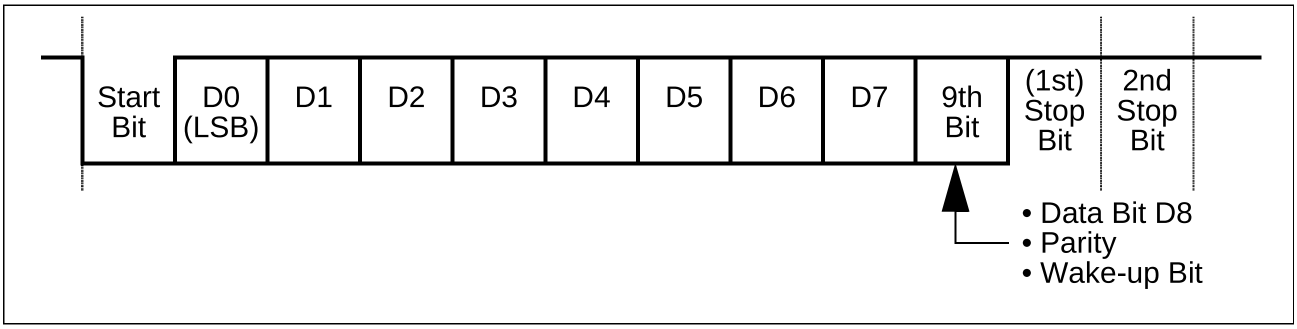


Figure 11-4
Asynchronous 9-bit Data Frames

Asynchronous transmission begins at the next overflow of the divide-by-16 counter (see figure above), provided that S0R is set and data has been loaded into S0TBUF. The transmitted data frame consists of three basic elements:

- the start bit
- the data field (8 or 9 bits, LSB first, including a parity bit, if selected)
- the delimiter (1 or 2 stop bits)

Data transmission is double buffered. When the transmitter is idle, the transmit data loaded into S0TBUF is immediately moved to the transmit shift register thus freeing S0TBUF for the next data to be sent. This is indicated by the transmit buffer interrupt request flag S0TBIR being set. S0TBUF may now be loaded with the next data, while transmission of the previous one is still going on.

The transmit interrupt request flag S0TIR will be set before the last bit of a frame is transmitted, i.e. before the first or the second stop bit is shifted out of the transmit shift register.
The transmitter output pin TXD0/P3.10 must be configured for alternate data output, i.e. P3.10 = ’1’ and DP3.10 = ’1’.

Asynchronous reception is initiated by a falling edge (1-to-0 transition) on pin RXD0, provided that bits S0R and S0REN are set. The receive data input pin RXD0 is sampled at 16 times the rate of the selected baud rate. A majority decision of the 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 pin RXD0. If the start bit proves valid, the receive circuit continues sampling and shifts the incoming data frame into the receive shift register.

When the last stop bit has been received, the content of the receive shift register is transferred to the receive data buffer register S0RBUF. Simultaneously, the receive interrupt request flag S0RIR is set after the 9th sample in the last stop bit time slot (as programmed), regardless whether valid stop bits have been received or not. The receive circuit then waits for the next start bit (1-to-0 transition) at the receive data input pin.

The receiver input pin RXD0/P3.11 must be configured for input, i.e. DP3.11 = ’0’.

Asynchronous reception is stopped by clearing bit S0REN. 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 only transferred to the receive buffer register, 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.

11.2 Synchronous Operation

Synchronous mode supports half-duplex communication, basically for simple IO expansion via shift registers. Data is transmitted and received via pin RXD0/P3.11, while pin TXD0/P3.10 outputs the shift clock. These signals are alternate functions of Port 3 pins. Synchronous mode is selected with S0M = ’000_B’.

8 data bits are transmitted or received synchronous to a shift clock generated by the internal baud rate generator. The shift clock is only active as long as data bits are transmitted or received.

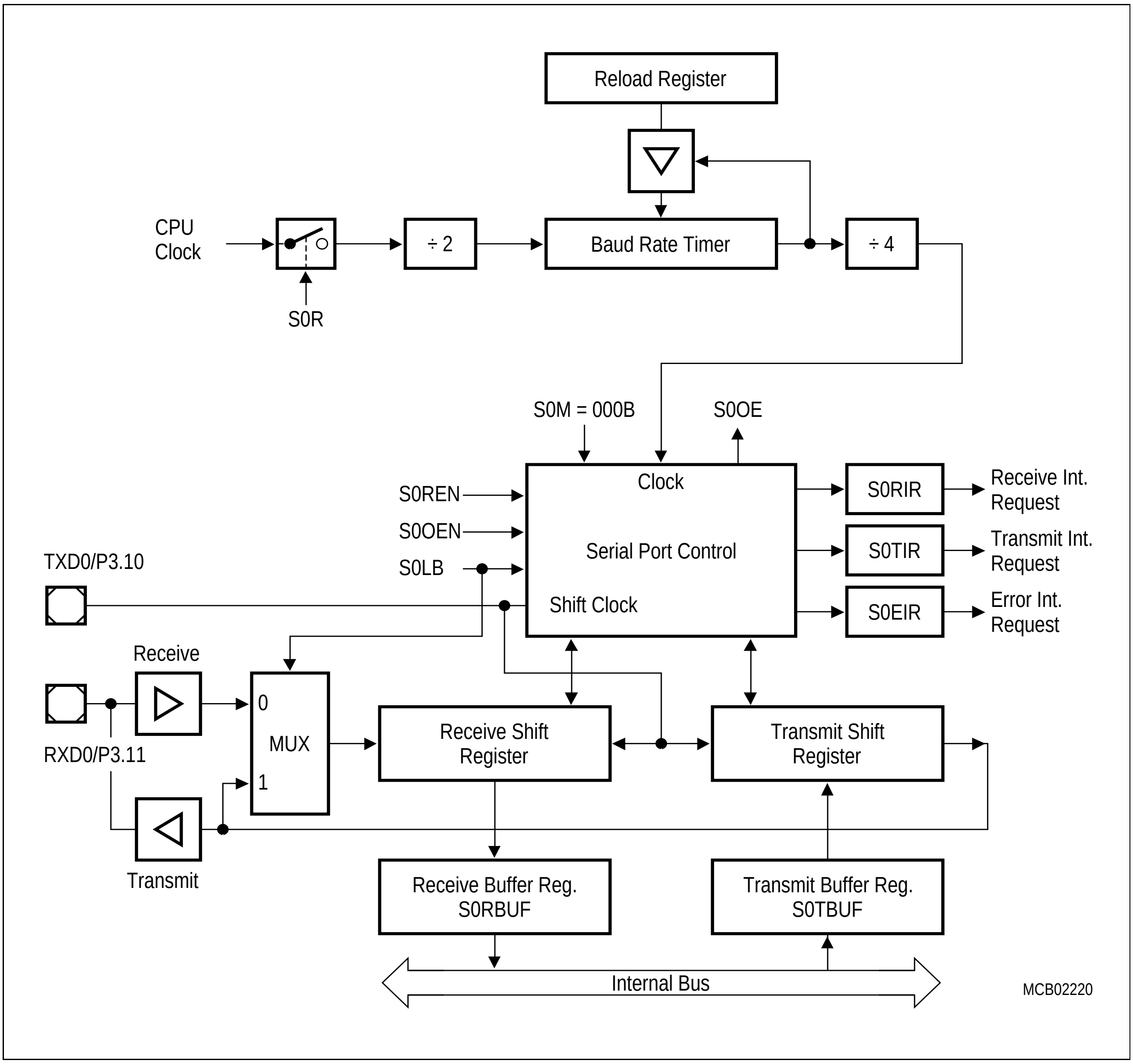


Figure 11-5
Synchronous Mode of Serial Channel ASC0

Synchronous transmission begins within 4 state times after data has been loaded into S0TBUF, provided that S0R is set and S0REN = ’0’ (half-duplex, no reception). Data transmission is double buffered. When the transmitter is idle, the transmit data loaded into S0TBUF is immediately moved to the transmit shift register thus freeing S0TBUF for the next data to be sent. This is indicated by the transmit buffer interrupt request flag S0TBIR being set. S0TBUF may now be loaded with the next data, while transmission of the previous one is still going on. The data bits are transmitted synchronous with the shift clock. After the bit time for the 8th data bit, both pins TXD0 and RXD0 will go high, the transmit interrupt request flag S0TIR is set, and serial data transmission stops.

Pin TXD0/P3.10 must be configured for alternate data output, i.e. P3.10 = ’1’ and DP3.10 = ’1’, in order to provide the shift clock. Pin RXD0/P3.11 must also be configured for output (P3.11 = ’1’ and DP3.11 = ’1’) during transmission.

Synchronous reception is initiated by setting bit S0REN = ’1’. If bit S0R = 1, the data applied at pin RXD0 are clocked into the receive shift register synchronous to the clock which is output at pin TXD0. After the 8th bit has been shifted in, the content of the receive shift register is transferred to the receive data buffer S0RBUF, the receive interrupt request flag S0RIR is set, the receiver enable bit S0REN is reset, and serial data reception stops.

Pin TXD0/P3.10 must be configured for alternate data output, i.e. P3.10 = ’1’ and DP3.10 = ’1’, in order to provide the shift clock. Pin RXD0/P3.11 must be configured as alternate data input (DP3.11 = ’0’).

Synchronous reception is stopped by clearing bit S0REN. 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 the receive buffer register at the time the reception of the next byte is complete, both the error interrupt request flag S0EIR and the overrun error status flag S0OE will be set, provided the overrun check has been enabled by bit S0OEN.

11.3 Hardware Error Detection Capabilities

To improve the safety of serial data exchange, the serial channel ASC0 provides an error interrupt request flag, which indicates the presence of an error, and three (selectable) error status flags in register S0CON, which indicate which error has been detected during reception. Upon completion of a reception, the error interrupt request flag S0EIR will be set simultaneously with the receive interrupt request flag S0RIR, if one or more of the following conditions are met:

- If the framing error detection enable bit S0FEN is set and any of the expected stop bits is not high, the framing error flag S0FE is set, indicating that the error interrupt request is due to a framing error (Asynchronous mode only).
- If the parity error detection enable bit S0PEN is set in the modes where a parity bit is received, and the parity check on the received data bits proves false, the parity error flag S0PE is set, indicating that the error interrupt request is due to a parity error (Asynchronous mode only).
- If the overrun error detection enable bit S0OEN is set and the last character received was not read out of the receive buffer by software or PEC transfer at the time the reception of a new frame is complete, the overrun error flag S0OE is set indicating that the error interrupt request is due to an overrun error (Asynchronous and synchronous mode).

11.4 ASC0 Baud Rate Generation

The serial channel ASC0 has its own dedicated 13-bit baud rate generator with 13-bit reload capability, allowing baud rate generation independent of the GPT timers.

The baud rate generator is clocked with the CPU clock divided by 2 ($f_{CPU}/2$). The timer is counting downwards and can be started or stopped through the Baud Rate Generator Run Bit S0R in register S0CON. Each underflow of the timer provides one clock pulse to the serial channel. The timer is reloaded with the value stored in its 13-bit reload register each time it underflows. The resulting clock is again divided according to the operating mode and controlled by the Baudrate Selection Bit S0BRS. If S0BRS = ’1’, the clock signal is additionally divided to 2/3rd of its frequency (see formulas and table). So the baud rate of ASC0 is determined by the CPU clock, the reload value, the value of S0BRS and the operating mode (asynchronous or synchronous).

Register S0BG is the dual-function Baud Rate Generator/Reload register. Reading S0BG returns the content of the timer (bits 15 … 13 return zero), while writing to S0BG always updates the reload register (bits 15 … 13 are insignificant).

An auto-reload of the timer with the content of the reload register is performed each time S0BG is written to. However, if S0R = ’0’ at the time the write operation to S0BG is performed, the timer will not be reloaded until the first instruction cycle after S0R = ’1’.

Asynchronous Mode Baud Rates

For asynchronous operation, the baud rate generator provides a clock with 16 times the rate of the established baud rate. Every received bit is sampled at the 7th, 8th and 9th cycle of this clock. The baud rate for asynchronous operation of serial channel ASC0 and the required reload value for a given baudrate can be determined by the following formulas:

$$B_{Async} = \frac{f_{CPU}}{16 \times (2 + \text{<S0BRS>}) \times (\text{<S0BRL>} + 1)} \qquad \text{S0BRL} = \left(\frac{f_{CPU}}{16 \times (2 + \text{<S0BRS>}) \times B_{Async}}\right) - 1$$

<S0BRL> represents the content of the reload register, taken as unsigned 13-bit integer,
<S0BRS> represents the value of bit S0BRS (i.e. ‘0’ or ‘1’), taken as integer.

The maximum baud rate that can be achieved for the asynchronous modes when using a CPU clock of 16 MHz is 500 KBaud. The table below lists various commonly used baud rates together with the required reload values and the deviation errors compared to the intended baudrate.

ASC0 Baudrate Generation

Baud Rate		**S0BRS = ‘0’, f_{CPU} = 16 MHz**		**S0BRS = ‘1’, f_{CPU} = 16 MHz**	
		Deviation Error	**Reload Value**	**Deviation Error**	**Reload Value**
500	KBaud	± 0.0%	0000_H	---	---
19.2	KBaud	+ 0.2% / – 3.5%	0019_H / $001A_H$	+ 2.1% / – 3.5%	0010_H / 0011_H
9600	Baud	+ 0.2% / – 1.7%	0033_H / 0034_H	+ 2.1% / – 0.8%	0021_H / 0022_H
4800	Baud	+ 0.2% / – 0.8%	0067_H / 0068_H	+ 0.6% / – 0.8%	0044_H / 0045_H
2400	Baud	+ 0.2% / – 0.3%	$00CF_H$ / $00D0_H$	+ 0.6% / – 0.1%	0089_H / $008A_H$
1200	Baud	+ 0.4% / – 0.1%	$019F_H$ / $01A0_H$	+ 0.3% / – 0.1%	0114_H / 0115_H
600	Baud	+ 0.0% / – 0.1%	0340_H / 0341_H	+ 0.1% / – 0.1%	$022A_H$ / $022B_H$
61	Baud	+ 0.1%	$1FFF_H$	+ 0.0% / – 0.0%	$115B_H$ / $115C_H$

Note: The deviation errors given in the table above are rounded. Using a baudrate crystal (e.g. 18.432 MHz) will provide correct baudrates without deviation errors.

Synchronous Mode Baud Rates

For synchronous operation, the baud rate generator provides a clock with 4 times the rate of the established baud rate. The baud rate for synchronous operation of serial channel ASC0 can be determined by the following formula:

$$B_{Sync} = \frac{f_{CPU}}{4 \times (2 + \text{<S0BRS>}) \times (\text{<S0BRL>} + 1)} \qquad \text{S0BRL} = \left(\frac{f_{CPU}}{4 \times (2 + \text{<S0BRS>}) \times B_{Sync}}\right) - 1$$

<S0BRL> represents the content of the reload register, taken as unsigned 13-bit integers,
<S0BRS> represents the value of bit S0BRS (i.e. ‘0’ or ‘1’), taken as integer.

The maximum baud rate that can be achieved in synchronous mode when using a CPU clock of 16 MHz is 2 MBaud.

11.5 ASC0 Interrupt Control

Four bit addressable interrupt control registers are provided for serial channel ASC0. Register S0TIC controls the transmit interrupt, S0TBIC controls the transmit buffer interrupt, S0RIC controls the receive interrupt and S0EIC controls the error interrupt of serial channel ASC0. Each interrupt source also has its own dedicated interrupt vector. S0TINT is the transmit interrupt vector, S0TBINT is the transmit interrupt vector, S0RINT is the receive interrupt vector, and S0EINT is the error interrupt vector.

The cause of an error interrupt request (framing, parity, overrun error) can be identified by the error status flags in control register S0CON.

Note: In contrary to the error interrupt request flag S0EIR, the error status flags S0FE/S0PE/S0OE are not reset automatically upon entry into the error interrupt service routine, but must be cleared by software.

S0TIC (FF6C$_H$ / B6$_H$) SFR **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								S0TIR	**S0TIE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

S0RIC (FF6E$_H$ / B7$_H$) SFR **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								S0RIR	**S0RIE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

S0EIC (FF70$_H$ / B8$_H$) SFR **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								S0EIR	**S0EIE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

S0TBIC (F19C$_H$ / CE$_H$) **ESFR** **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								S0 TBIR	**S0 TBIE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

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

Using the ASC0 Interrupts

For normal operation (i.e. besides the error interrupt) the ASC0 provides three interrupt requests to control data exchange via this serial channel:

• S0TBIR	is activated when data is moved from S0TBUF to the transmit shift register.
• S0TIR	is activated before the last bit of an asynchronous frame is transmitted, or after the last bit of a synchronous frame has been transmitted.
• S0RIR	is activated when the received frame is moved to S0RBUF.

While the task of the receive interrupt handler is quite clear, the transmitter is serviced by two interrupt handlers. This provides advantages for the servicing software.

For single transfers is sufficient to use the transmitter interrupt (S0TIR), which indicates that the previously loaded data has been transmitted, except for the last bit of an asynchronous frame.

For multiple back-to-back transfers it is necessary to load the following piece of data at last 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 impossible at all.
Using the transmit buffer interrupt (S0TBIR) to reload transmit data gives the time to transmit a complete frame for the service routine, as S0TBUF may be reloaded while the previous data is still being transmitted.

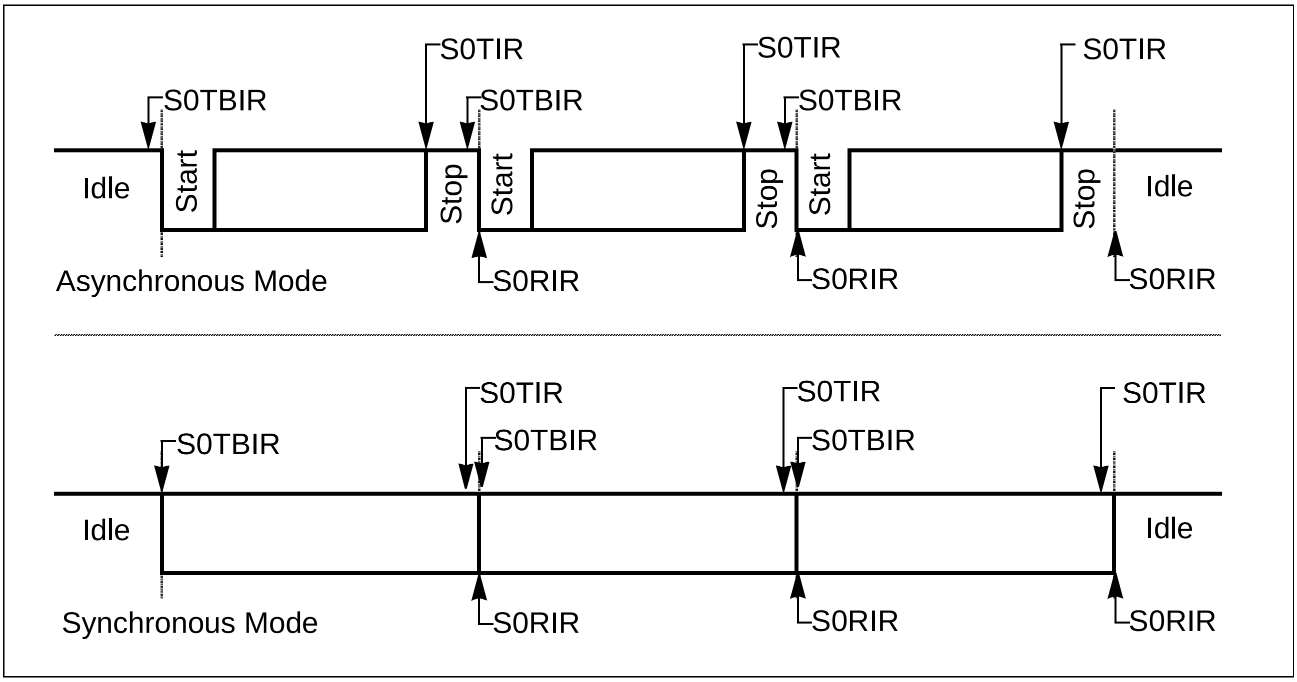


Figure 11-6
ASC0 Interrupt Generation

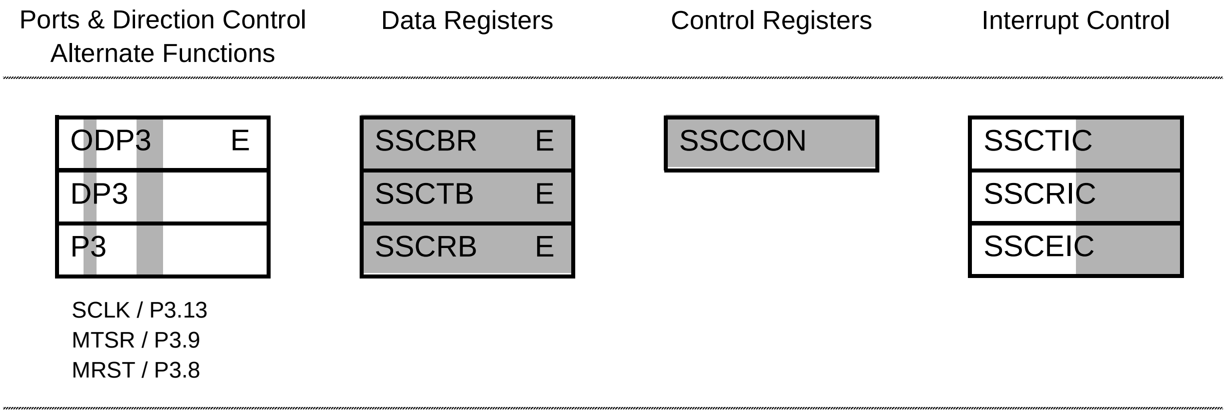
As shown in the figure above, S0TBIR is an early trigger for the reload routine, while S0TIR indicates the completed transmission. Software using handshake therefore should rely on S0TIR at the end of a data block to make sure that all data has really been transmitted.

12 The High-Speed Synchronous Serial Interface

The High-Speed Synchronous Serial Interface SSC provides flexible high-speed serial communication between the C161RI and other microcontrollers, microprocessors or external peripherals.

The SSC supports full-duplex and half-duplex synchronous communication up to 4 MBaud (@ 16 MHz CPU clock). The serial clock signal can be generated by the SSC itself (master mode) or be received from an external master (slave mode). Data width, shift direction, clock polarity and phase are programmable. This allows communication with SPI-compatible devices. Transmission and reception of data is double-buffered. A 16-bit baud rate 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 (e.g. the ASC0 in synchronous mode), serve for master/slave or multimaster interconnections or operate compatible with the popular SPI interface. So it 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 pins MTSR/P3.9 (Master Transmit / Slave Receive) and MRST/P3.8 (Master Receive / Slave Transmit). The clock signal is output or input on pin SCLK/P3.13. These pins are alternate functions of Port 3 pins.



ODP3	Port 3 Open Drain Control Register		
DP3	Port 3 Direction Control Register	P3	Port 3 Data Register
SSCBR	SSC Baud Rate Generator/Reload Register	SSCCON	SSC Control Register
SSCTB	SSC Transmit Buffer Register	SSCRB	SSC Receive Buffer Register
SSCTIC	SSC Transmit Interrupt Control Register	SSCRIC	SSC Receive Interrupt Control Register
		SSCEIC	SSC Error Interrupt Control Register

Figure 12-1
SFRs and Port Pins associated with the SSC

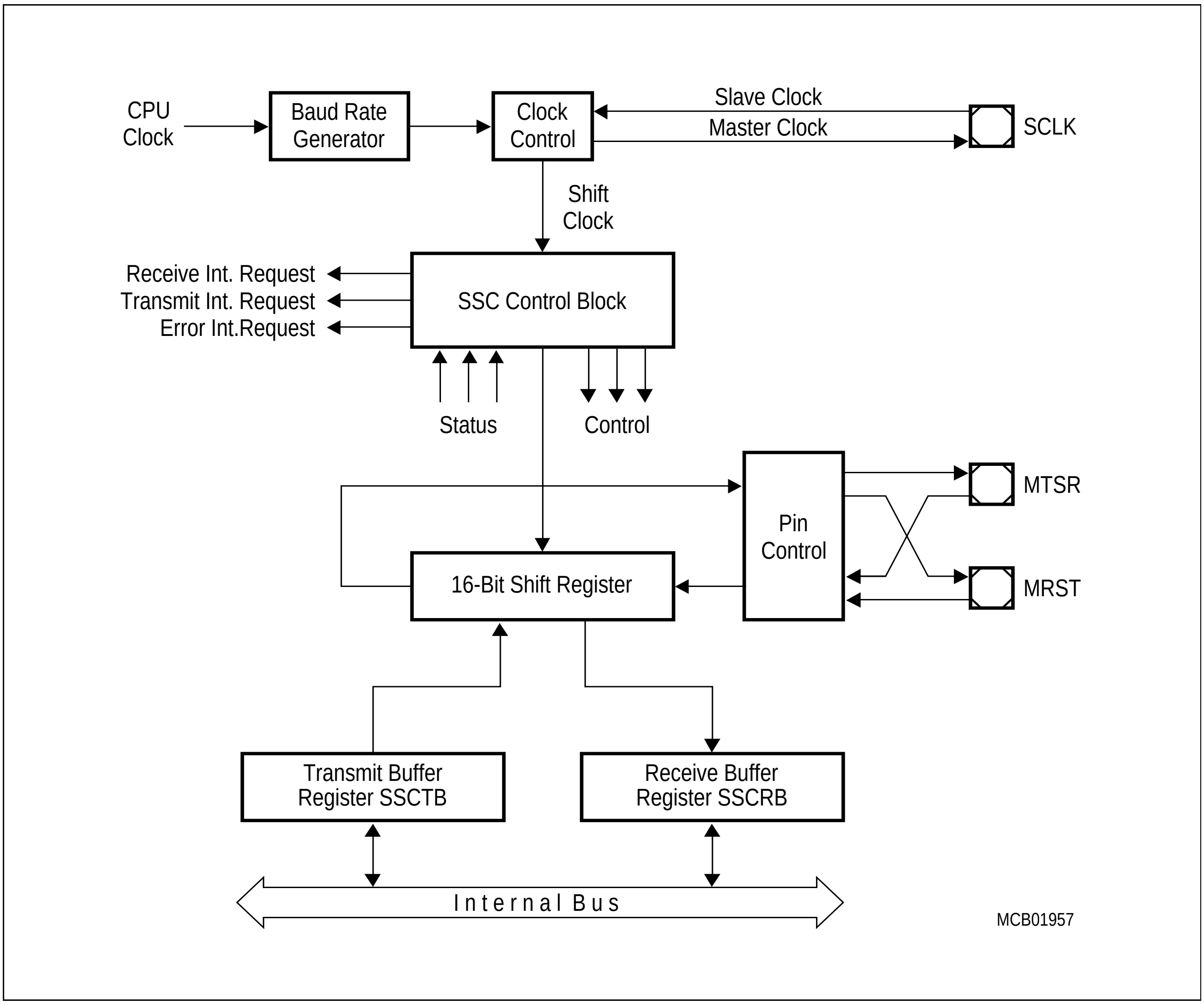


Figure 12-2
Synchronous Serial Channel SSC Block Diagram

The operating mode of the serial channel SSC is controlled by its bit-addressable control register SSCCON. This register serves for two purposes:

- during programming (SSC disabled by SSCEN = ’0’) it provides access to a set of control bits,
- during operation (SSC enabled by SSCEN = ’1’) it provides access to a set of status flags.

Register SSCCON is shown below in each of the two modes.

SSCCON (FFB2$_H$ / D9$_H$) SFR **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SSC EN=0	**SSC MS**	**-**	**SSC AREN**	**SSC BEN**	**SSC PEN**	**SSC REN**	**SSC TEN**	**-**	**SSC PO**	**SSC PH**	**SSC HB**	**SSCBM**			
rw	rw	-	rw	rw	rw	rw	rw	-	rw	rw	rw	rw			

Bit	Function (Programming Mode, SSCEN = ‘0’)
SSCBM	**SSC Data Width Selection** 0 : Reserved. Do not use this combination. 1 … 15 : Transfer Data Width is 2 … 16 bit (<SSCBM> + 1)
SSCHB	**SSC Heading Control Bit** 0 : Transmit/Receive LSB First 1 : Transmit/Receive MSB First
SSCPH	**SSC Clock Phase Control Bit** 0 : Shift transmit data on the leading clock edge, latch on trailing edge 1 : Latch receive data on leading clock edge, shift on trailing edge
SSCPO	**SSC Clock Polarity Control Bit** 0 : Idle clock line is low, leading clock edge is low-to-high transition 1 : Idle clock line is high, leading clock edge is high-to-low transition
SSCTEN	**SSC Transmit Error Enable Bit** 0 : Ignore transmit errors 1 : Check transmit errors
SSCREN	**SSC Receive Error Enable Bit** 0 : Ignore receive errors 1 : Check receive errors
SSCPEN	**SSC Phase Error Enable Bit** 0 : Ignore phase errors 1 : Check phase errors
SSCBEN	**SSC Baudrate Error Enable Bit** 0 : Ignore baudrate errors 1 : Check baudrate errors
SSCAREN	**SSC Automatic Reset Enable Bit** 0 : No additional action upon a baudrate error 1 : The SSC is automatically reset upon a baudrate error
SSCMS	**SSC Master Select Bit** 0 : Slave Mode. Operate on shift clock received via SCLK. 1 : Master Mode. Generate shift clock and output it via SCLK.
SSCEN	**SSC Enable Bit = ‘0’** Transmission and reception disabled. Access to control bits.

SSCCON (FFB2$_H$ / D9$_H$) SFR **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SSC EN=1	**SSC MS**	**-**	**SSC BSY**	**SSC BE**	**SSC PE**	**SSC RE**	**SSC TE**	**-**	**-**	**-**	**-**	**SSCBC**			
rw	rw	-	rw	rw	rw	rw	rw	-	-	-	-	r			

Bit	Function (Operating Mode, SSCEN = ‘1’)
SSCBC	**SSC Bit Count Field** Shift counter is updated with every shifted bit. **Do not write to!!!**
SSCTE	**SSC Transmit Error Flag** 1 : Transfer starts with the slave’s transmit buffer not being updated
SSCRE	**SSC Receive Error Flag** 1 : Reception completed before the receive buffer was read
SSCPE	**SSC Phase Error Flag** 1 : Received data changes around sampling clock edge
SSCBE	**SSC Baudrate Error Flag** 1 : More than factor 2 or 0.5 between Slave’s actual and expected baudrate
SSCBSY	**SSC Busy Flag** Set while a transfer is in progress. **Do not write to!!!**
SSCMS	**SSC Master Select Bit** 0 : Slave Mode. Operate on shift clock received via SCLK. 1 : Master Mode. Generate shift clock and output it via SCLK.
SSCEN	**SSC Enable Bit = ‘1’** Transmission and reception enabled. Access to status flags and M/S control.

Note: • The target of an access to SSCCON (control bits or flags) is determined by the state of SSCEN prior to the access, i.e. writing C057$_H$ to SSCCON in programming mode (SSCEN = ’0’) will initialize the SSC (SSCEN was ‘0’) and then turn it on (SSCEN = ’1’).
• When writing to SSCCON, make sure that reserved locations receive zeros.

The shift register of the SSC is connected to both the transmit pin and the receive pin via the pin control logic (see block diagram). Transmission and reception of serial data is synchronized and takes place at the same time, i.e. the same number of transmitted bits is also received. Transmit data is written into the Transmit Buffer SSCTB. It is moved to the shift register as soon as this is empty. An SSC-master (SSCMS = ’1’) immediately begins transmitting, while an SSC-slave (SSCMS = ’0’) will wait for an active shift clock. When the transfer starts, the busy flag SSCBSY is set and a transmit interrupt request (SSCTIR) will be generated to indicate that SSCTB 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 SSCRB and a receive interrupt request (SSCRIR) will be generated. If no further transfer is to take place (SSCTB is empty), SSCBSY will be cleared at the same time. Software should not modify SSCBSY, as this flag is hardware controlled.

Note: Only one SSC (etc.) can be master at a given time.

The transfer of serial data bits can be programmed in many respects:

- the data width can be chosen from 2 bits to 16 bits
- transfer may start with the LSB or the MSB
- the shift clock may be idle low or idle high
- data bits may be shifted with the leading or trailing edge of the clock signal
- the baudrate may be set from 122 Bd up to 4 MBd (@ 16 MHz CPU clock)
- the shift clock can be generated (master) or received (slave)

This allows the adaptation of the SSC to a wide range of applications, where serial data transfer is required.

The Data Width Selection supports the transfer of frames of any length, from 2-bit “characters” up to 16-bit “characters”. Starting with the LSB (SSCHB = ’0’) allows communication e.g. with ASC0 devices in synchronous mode (C166 family) or 8051 like serial interfaces. Starting with the MSB (SSCHB = ’1’) allows operation compatible with the SPI interface.
Regardless which data width is selected and whether the MSB or the LSB is transmitted first, the transfer data is always right aligned in registers SSCTB and SSCRB, 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 SSCTB are ignored, the unselected bits of SSCRB will be not valid and should be ignored by the receiver service routine.

The Clock Control allows the adaptation of transmit and receive behavior of the SSC to a variety of serial interfaces. A specific clock edge (rising or falling) is used to shift out transmit data, while the other clock edge is used to latch in receive data. Bit SSCPH selects the leading edge or the trailing edge for each function. Bit SSCPO selects the level of the clock line in the idle state. So for an idle-high clock the leading edge is a falling one, a 1-to-0 transition. The figure below is a summary.

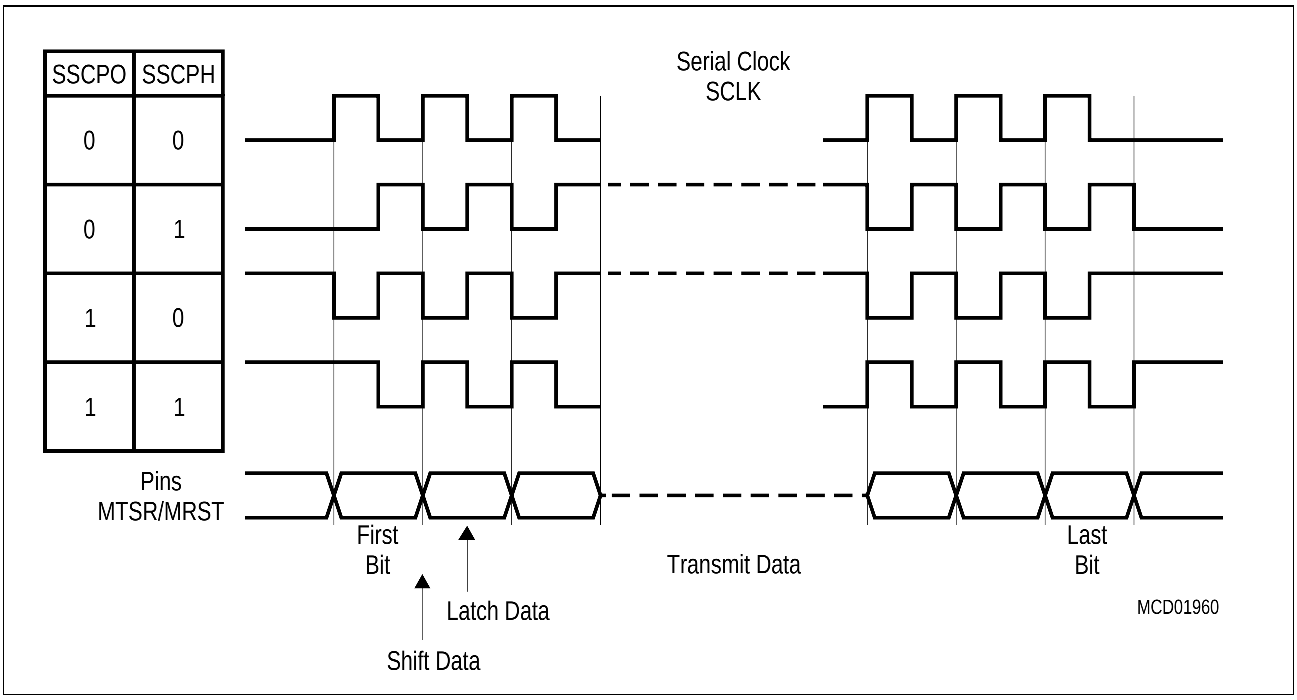


Figure 12-3
Serial Clock Phase and Polarity Options

12.1 Full-Duplex Operation

The different devices are connected through three lines. The definition of these lines is always determined by the master: The line connected to the master's data output pin MTSR is the transmit line, the receive line is connected to its data input line MRST, and the clock line is connected to pin SCLK. Only the device selected for master operation generates and outputs the serial clock on pin SCLK. All slaves receive this clock, so their pin SCLK must be switched to input mode (DP3.13 = '0'). The output of the master's shift register is connected to the external transmit line, which in turn is connected to the slaves' shift register input. The output of the slaves' shift register is connected to the external receive line in order to enable the master to receive the data shifted out of the slave. The external connections are hard-wired, the function and direction of these pins 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, select one device for master operation (SSCMS = '1'), all others must be programmed for slave operation (SSCMS = '0'). Initialization includes the operating mode of the device's SSC and also the function of the respective port lines (see "Port Control").

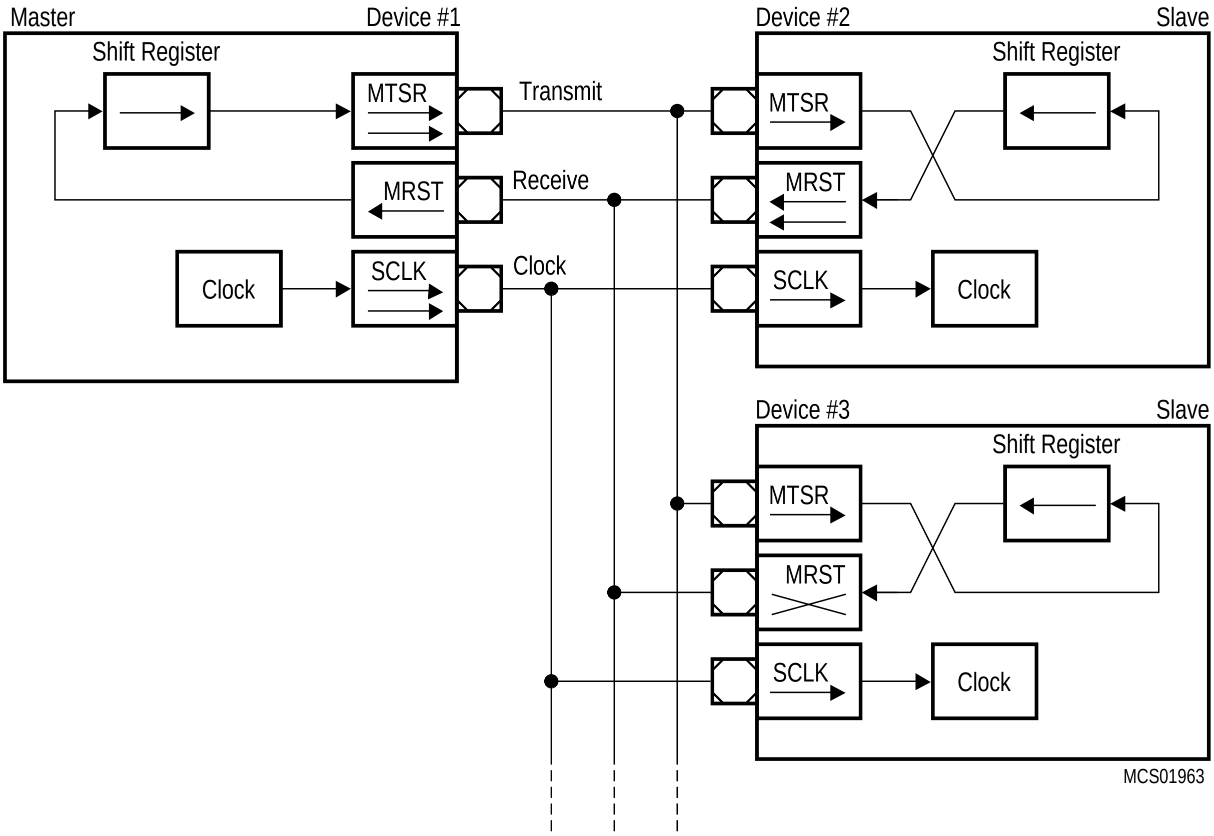


Figure 12-4
SSC Full Duplex Configuration

The data output pins MRST of all slave devices are connected together onto the one receive line in this configuration. During a transfer each slave shifts out data from its shift register. There are two ways to avoid collisions on the receive line due to different slave data:

Only one slave drives the line, i.e. enables the driver of its MRST pin. All the other slaves have to program there MRST pins to input. So only one slave can put its data onto the master's receive line. Only receiving of data from the master is possible. The master selects the slave device from which it expects data either by separate select lines, or by sending a special command to this slave. The selected slave then switches its MRST line to output, until it gets a deselection signal or command.

The slaves use open drain output on MRST. This forms a Wired-AND connection. The receive line needs an external pullup in this case. Corruption of the data on the receive line sent by the selected slave is avoided, when all slaves which are not selected for transmission to the master only send ones (‘1’). Since this high level is not actively driven onto the line, but only held through the pullup device, the selected slave can pull this line actively to a low level when transmitting a zero bit. The master selects the slave device from which it expects data either by separate select lines, or by sending a special command to this slave.

After performing all necessary initializations of the SSC, the serial interfaces can be enabled. For a master device, the alternate clock line will now go to its programmed polarity. The alternate data line will go to either '0' or '1', until the first transfer 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 SSCTB. This value is copied into the shift register (which is assumed to be empty at this time), and the selected first bit of the transmit data will be placed onto the MTSR line on the next clock from the baudrate generator (transmission only starts, if SSCEN = ’1’). Depending on the selected clock phase, also a clock pulse will be generated on the SCLK line. With the opposite clock edge the master at the same time latches and shifts in the data detected at its input line MRST. This “exchanges” the transmit data with the receive data. Since 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 slaves’ shift registers, while the master's shift register holds the data of the selected slave. In the master and all slaves the content of the shift register is copied into the receive buffer SSCRB and the receive interrupt flag SSCRIR is set.

A slave device will immediately output the selected first bit (MSB or LSB of the transfer data) at pin MRST, when the content of the transmit buffer is copied into the slave's shift register. It will not wait for the next clock from the baudrate generator, as the master does. The reason for this is that, depending on the selected clock phase, the first clock edge generated by the master may be already used to clock in the first data bit. So the slave's first data bit must already be valid at this time.

Note: On the SSC always a transmission **and** a reception takes place at the same time, regardless whether valid data has been transmitted or received. This is different e.g. from asynchronous reception on ASC0.

The initialization of the SCLK pin on the master requires some attention in order to avoid undesired clock transitions, which may disturb the other receivers. The state of the internal alternate output lines is ’1’ as long as the SSC is disabled. This alternate output signal is ANDed with the respective port line output latch. Enabling the SSC with an idle-low clock (SSCPO = ’0’) will drive the alternate data output and (via the AND) the port pin SCLK immediately low. To avoid this, use the following sequence:

- select the clock idle level (SSCPO = ’x’)
- load the port output latch with the desired clock idle level (P3.13 = ’x’)
- switch the pin to output (DP3.13 = ’1’)
- enable the SSC (SSCEN = ’1’)
- if SSCPO = ’0’: enable alternate data output (P3.13 = ’1’)

The same mechanism as for selecting a slave for transmission (separate select lines or special commands) may also be used to move the role of the master to another device in the network. In this case the previous master and the future master (previous slave) will have to toggle their operating mode (SSCMS) and the direction of their port pins (see description above).

12.2 Half Duplex Operation

In a half duplex configuration only one data line is necessary for both receiving **and** transmitting of data. The data exchange line is connected to both pins MTSR and MRST of each device, the clock line is connected to the SCLK pin.

The master device controls the data transfer by generating the shift clock, while the slave devices receive it. Due to the fact that all transmit and receive pins are connected to the one data exchange line, serial data may be moved between arbitrary stations.

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 only send ones.

Since the data inputs and outputs are connected together, a transmitting device will clock in its own data at the input pin (MRST for a master device, MTSR for a slave). By these means any corruptions on the common data exchange line are detected, where the received data is not equal to the transmitted data.

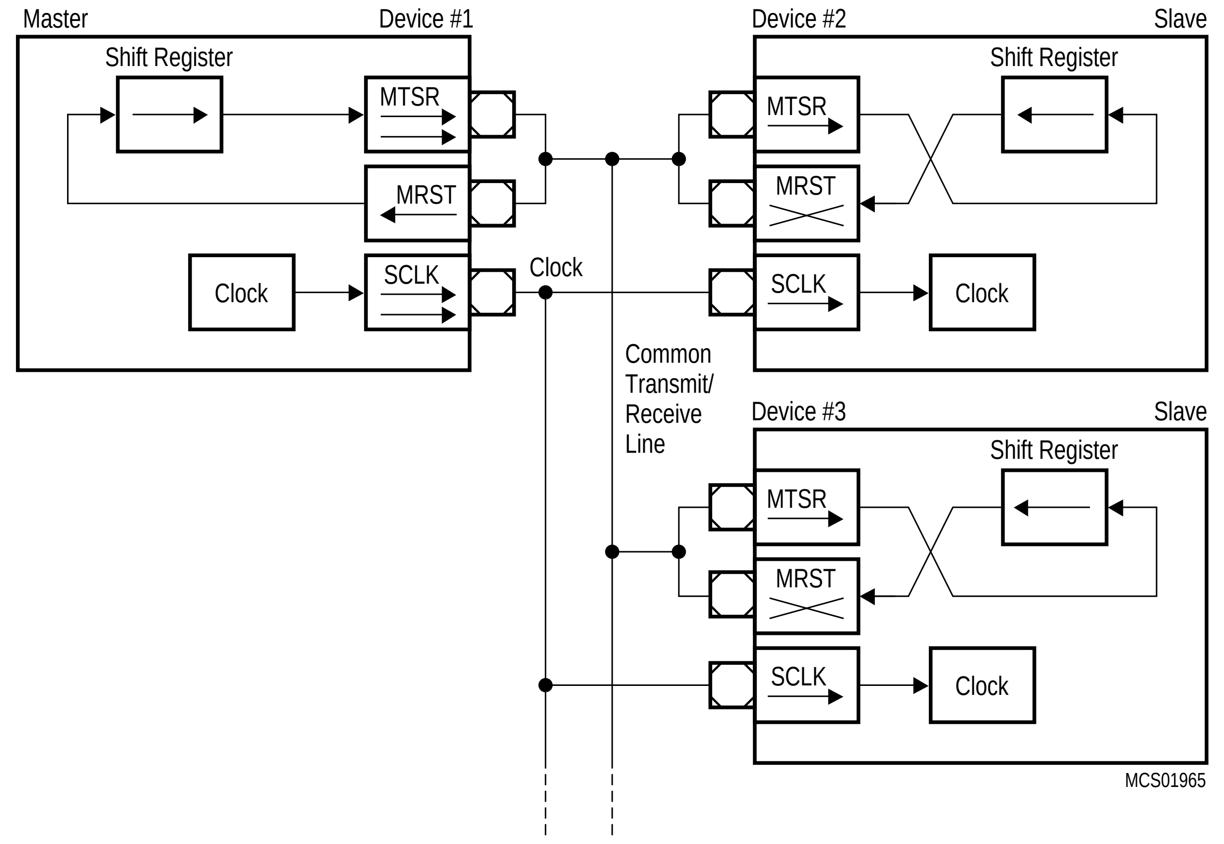


Figure 12-5
SSC Half Duplex Configuration

Continuous Transfers

When the transmit interrupt request flag is set, it indicates that the transmit buffer SSCTB is empty and ready to be loaded with the next transmit data. If SSCTB has been reloaded by the time the current transmission is finished, the data is immediately transferred to the shift register and the next transmission will start without any additional delay. On the data line there is no gap between the two successive frames. E.g. two byte transfers would look the same as one word transfer. This feature can be used to interface with devices which can operate with or require more than 16 data bits per transfer. It is just a matter of software, how long a total data frame length can be. This option can also be used e.g. to interface to byte-wide and word-wide devices on the same serial bus.

Note: Of course, this can only happen in multiples of the selected basic data width, since it would require disabling/enabling of the SSC to reprogram the basic data width on-the-fly.

Port Control

The SSC uses three pins of Port 3 to communicate with the external world. Pin P3.13/SCLK serves as the clock line, while pins P3.8/MRST (Master Receive / Slave Transmit) and P3.9/MTSR (Master Transmit / Slave Receive) serve as the serial data input/output lines.

The operation of these pins depends on the selected operating mode (master or slave). In order to enable the alternate output functions of these pins instead of the general purpose IO operation, the respective port latches have to be set to ’1’, since the port latch outputs and the alternate output lines are ANDed. When an alternate data output line is not used (function disabled), it is held at a high level, allowing IO operations via the port latch. The direction of the port lines depends on the operating mode. The SSC will automatically use the correct alternate input or output line of the ports when switching modes. The direction of the pins, however, must be programmed by the user, as shown in the tables. Using the open drain output feature helps to avoid bus contention problems and reduces the need for hardwired hand-shaking or slave select lines. In this case it is not always necessary to switch the direction of a port pin. The table below summarizes the required values for the different modes and pins.

SSC Port Control

Pin	Master Mode			Slave Mode		
	Function	**Port Latch**	**Direction**	**Function**	**Port Latch**	**Direction**
SCLK	Serial Clock Output	P3.13 = ’1’	DP3.13 = ’1’	Serial Clock Input	P3.13 = ’x’	DP3.13 = ’0’
MTSR	Serial Data Output	P3.9 = ’1’	DP3.9 = ’1’	Serial Data Input	P3.9 = ’x’	DP3.9 = ’0’
MRST	Serial Data Input	P3.8 = ’x’	DP3.8 = ’0’	Serial Data Output	P3.8 = ’1’	DP3.8 = ’1’

Note: In the table above, an ’x’ means that the actual value is irrelevant in the respective mode, however, it is recommended to set these bits to ’1’, so they are already in the correct state when switching between master and slave mode.

12.3 Baud Rate Generation

The serial channel SSC has its own dedicated 16-bit baud rate generator with 16-bit reload capability, allowing baud rate generation independent from the timers.

The baud rate generator is clocked with the CPU clock divided by 2 ($f_{CPU}/2$). The timer is counting downwards and can be started or stopped through the global enable bit SSCEN in register SSCCON. Register SSCBR is the dual-function Baud Rate Generator/Reload register. Reading SSCBR, while the SSC is enabled, returns the content of the timer. Reading SSCBR, while the SSC is disabled, returns the programmed reload value. In this mode the desired reload value can be written to SSCBR.

Note: Never write to SSCBR, while the SSC is enabled.

The formulas below calculate either the resulting baud rate for a given reload value, or the required reload value for a given baudrate:

$$B_{SSC} = \frac{f_{CPU}}{2 \times (\text{<SSCBR>} + 1)} \qquad \text{SSCBR} = \left(\frac{f_{CPU}}{2 \times \text{Baudrate}_{SSC}} \right) - 1$$

<SSCBR> represents the content of the reload register, taken as unsigned 16-bit integer.

The maximum baud rate that can be achieved when using a CPU clock of 16 MHz is 4 MBaud. The table below lists some possible baud rates together with the required reload values and the resulting bit times, assuming a CPU clock of 16 MHz.

Baud Rate		**Bit Time**		**Reload Value**
Reserved. Use a reload value > 0.		---	---	0000_H
4.0	MBaud	250	ns	0001_H
2.67	MBaud	375	ns	0002_H
2.0	MBaud	500	ns	0003_H
1.6	MBaud	625	ns	0004_H
1.0	MBaud	1	µs	0007_H
100	KBaud	10	µs	$004F_H$
10	KBaud	100	µs	$031F_H$
1.0	KBaud	1	ms	$1F3F_H$
122.1	Baud	8.2	ms	$FFFF_H$

Note: The contents of SSCBR must be > 0.

12.4 Error Detection Mechanisms

The SSC is able to detect four different error conditions. Receive Error and Phase Error are detected in all modes, while Transmit Error and Baudrate Error only apply to slave mode. When an error is detected, the respective error flag is set. When the corresponding Error Enable Bit is set, also an error interrupt request will be generated by setting SSCEIR (see figure below). The error interrupt handler may then check the error flags to determine the cause of the error interrupt. The error flags are not reset automatically (like SSCEIR), but rather must be cleared by software after servicing. This allows servicing of some error conditions via interrupt, while the others may be polled by software.

Note: The error interrupt handler must clear the associated (enabled) errorflag(s) to prevent repeated interrupt requests.

A **Receive Error** (Master or Slave mode) is detected, when a new data frame is completely received, but the previous data was not read out of the receive buffer register SSCRB. This condition sets the error flag SSCRE and, when enabled via SSCREN, the error interrupt request flag SSCEIR. The old data in the receive buffer SSCRB will be overwritten with the new value and is unretrievably lost.

A **Phase Error** (Master or Slave mode) is detected, when the incoming data at pin MRST (master mode) or MTSR (slave mode), sampled with the same frequency as the CPU clock, changes between one sample before and two samples after the latching edge of the clock signal (see “Clock Control”). This condition sets the error flag SSCPE and, when enabled via SSCPEN, the error interrupt request flag SSCEIR.

A **Baud Rate Error** (Slave mode) is detected, when the incoming clock signal deviates from the programmed baud rate by more than 100%, i.e. it either is more than double or less than half the expected baud rate. This condition sets the error flag SSCBE and, when enabled via SSCBEN, the error interrupt request flag SSCEIR. Using this error detection capability requires that the slave's baud rate generator is programmed to the same baud rate 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 SSCAREN = ’1’, an automatic reset of the SSC will be performed in case of this error. This is done to reinitialize the SSC, if too few or too many clock pulses have been detected.

A **Transmit Error** (Slave mode) is detected, when a transfer was initiated by the master (shift clock gets active), but the transmit buffer SSCTB of the slave was not updated since the last transfer. This condition sets the error flag SSCTE and, when enabled via SSCTEN, the error interrupt request flag SSCEIR. If a transfer starts while the transmit buffer is not updated, the slave will shift out the 'old' contents of the shift register, which normally is the data received during the last transfer.
This may lead to the corruption of the data on the transmit/receive line in half-duplex mode (open drain configuration), if this slave is not selected for transmission. This mode requires that slaves not selected for transmission only shift out ones, i.e. their transmit buffers must be loaded with 'FFFF$_H$' prior to any transfer.

Note: A slave with push/pull output drivers, which is not selected for transmission, will normally have its output drivers switched. However, in order to avoid possible conflicts or misinterpretations, it is recommended to always load the slave's transmit buffer prior to any transfer.

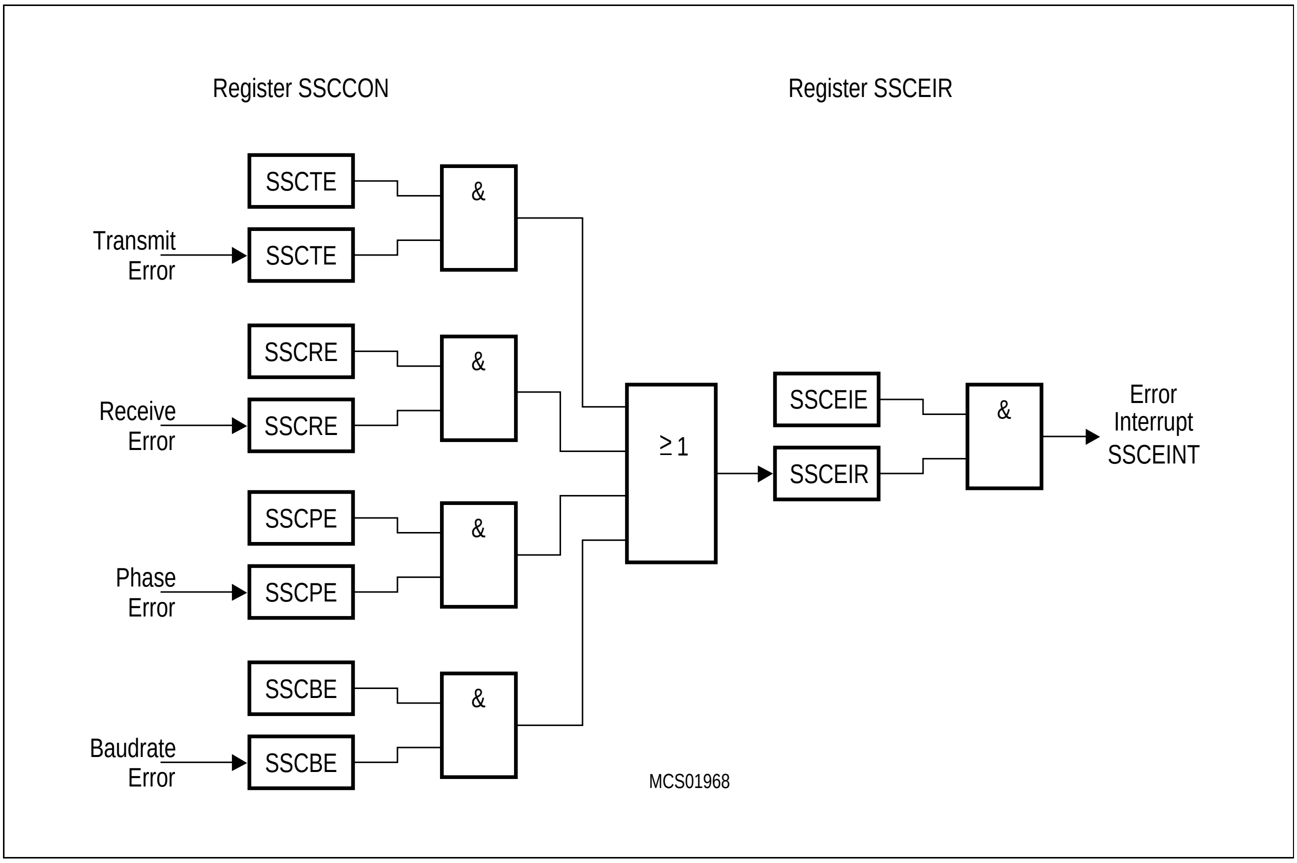


Figure 12-6
SSC Error Interrupt Control

12.5 SSC Interrupt Control

Three bit addressable interrupt control registers are provided for serial channel SSC. Register SSCTIC controls the transmit interrupt, SSCRIC controls the receive interrupt and SSCEIC controls the error interrupt of serial channel SSC. Each interrupt source also has its own dedicated interrupt vector. SCTINT is the transmit interrupt vector, SCRINT is the receive interrupt vector, and SCEINT is the error interrupt vector.

The cause of an error interrupt request (receive, phase, baudrate, transmit error) can be identified by the error status flags in control register SSCCON.

Note: In contrary to the error interrupt request flag SSCEIR, the error status flags SSCxE are not reset automatically upon entry into the error interrupt service routine, but must be cleared by software.

SSCTIC (FF72$_H$ / B9$_H$) SFR **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								SSC TIR	**SSC TIE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

SSCRIC (FF74$_H$ / BA$_H$) SFR **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								SSC RIR	**SSC RIE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

SSCEIC (FF76$_H$ / BB$_H$) SFR **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								SSC EIR	**SSC EIE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

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

13 The Watchdog Timer (WDT)

To allow recovery from software or hardware failure, the C161RI provides a Watchdog Timer. If the software fails to service this timer before an overflow occurs, an internal reset sequence will be initiated. This internal reset will also pull the $\overline{\text{RSTOUT}}$ pin low, which also resets the peripheral hardware which might be the cause for the malfunction. When the watchdog timer is enabled and the software has been designed to service it regularly before it overflows, the watchdog timer will supervise the program execution 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.

Note: When the bidirectional reset is enabled also pin $\overline{\text{RSTIN}}$ will be pulled low for the duration of the internal reset sequence upon a watchdog timer reset.

The watchdog timer provides two registers: a read-only timer register that contains the current count, and a control register for initialization and reset source detection.

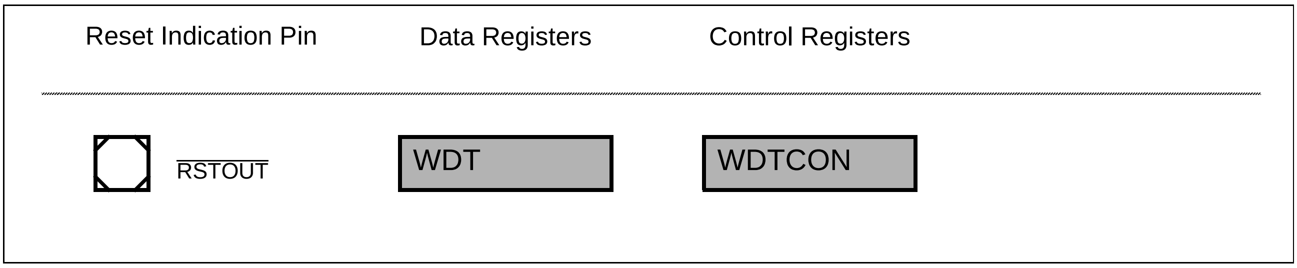


Figure 13-1
SFRs and Port Pins associated with the Watchdog Timer

The watchdog timer is a 16-bit up counter which can be clocked with the CPU clock (f_{CPU}) either divided by 2 or divided by 128. This 16-bit timer is realized as two concatenated 8-bit timers (see figure below). 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.

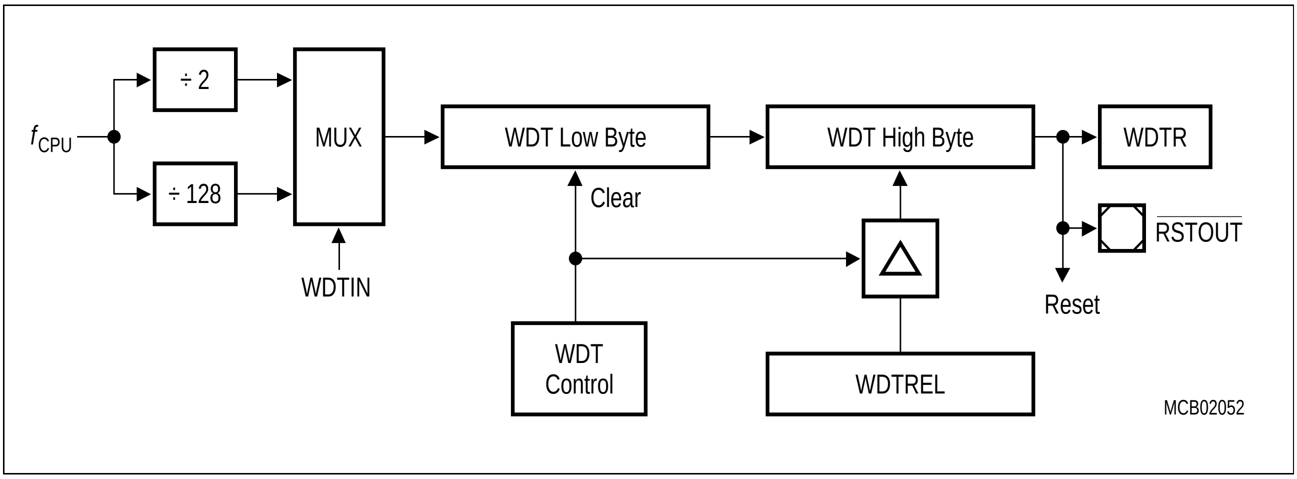


Figure 13-2
Watchdog Timer Block Diagram

13.1 Operation of the Watchdog Timer

The current count value of the Watchdog Timer is contained in the Watchdog Timer Register WDT which is a non-bitaddressable read-only register. 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 provides flags that indicate the source of a reset.

WDTCON (FFAE$_H$ / D7$_H$) SFR **Reset Value: 00XX$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WDTREL								**-**	**-**	**-**	**LHWR**	**SHWR**	**SWR**	**WDTR**	**WDTIN**
rw								-	-	-	r	r	r	r	rw

Bit	Function
WDTIN	**Watchdog Timer Input Frequency Selection** 0: Input frequency is f_{CPU} / 2 1: Input frequency is f_{CPU} / 128
WDTR	**Watchdog Timer Reset Indication Flag** Cleared by a hardware reset or by the SRVWDT instruction.
SWR	**Software Reset Indication Flag**
SHWR	**Short Hardware Reset Indication Flag**
LHWR	**Long Hardware Reset Indication Flag**
WDTREL	**Watchdog Timer Reload Value** (for the high byte of WDT)

Note: The reset value depends on the reset source (see description below).
The execution of EINIT clears the reset indication flags.

After any software reset, external hardware reset (see note), or watchdog timer reset, the watchdog timer is enabled and starts counting up from 0000_H with the frequency $f_{CPU}/2$. The input frequency may be switched to $f_{CPU}/128$ by setting bit WDTIN. The watchdog timer can be disabled via the instruction DISWDT (Disable Watchdog Timer). Instruction DISWDT is a protected 32-bit instruction which will ONLY be executed during the time between a reset and execution of either the EINIT (End of Initialization) or the SRVWDT (Service Watchdog Timer) instruction. Either one of these instructions disables the execution of DISWDT.

Note: After a hardware reset that activates the Bootstrap Loader the watchdog timer will be disabled.

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 an internal reset. This reset will pull the external reset indication pin $\overline{\text{RSTOUT}}$ low (and $\overline{\text{RSTIN}}$ in bidirectional reset mode). The Watchdog Timer Reset Indication Flag (WDTR) in register WDTCON will be set in this case.

A watchdog reset will also complete a running external bus cycle before starting the internal reset sequence if this bus cycle does not use $\overline{\text{READY}}$ or samples $\overline{\text{READY}}$ active (low) after the programmed waitstates. Otherwise the external bus cycle will be aborted.

To prevent the watchdog timer from overflowing it must be serviced periodically by the user software. The watchdog timer is serviced with the instruction SRVWDT which is a protected 32-bit instruction. Servicing the watchdog timer clears the low byte and reloads the high byte of the watchdog timer register WDT with the preset value from bitfield WDTREL which is the high byte of register WDTCON. Servicing the watchdog timer will also reset bit WDTR. After being serviced the watchdog timer continues counting up from the value ($\text{<WDTREL>} \times 2^8$). 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.

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 bit WDTIN in register WDTCON to be either $f_{CPU}/2$ or $f_{CPU}/128$.
- **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:

$$P_{WDT} = \frac{2^{(1 + \text{<WDTIN>} \times 6)} \times (2^{16} - \text{<WDTREL>} \times 2^{8})}{f_{CPU}}$$

The table below marks the possible ranges for the watchdog time which can be achieved using a certain CPU clock. Some numbers are rounded to 3 significant digits.

Watchdog Time Ranges

Reload Value in WDTREL	**Prescaler for f_{CPU}**					
	2 (WDTIN = '0')			**128 (WDTIN = '1')**		
	20 MHz	**16 MHz**	**12 MHz**	**20 MHz**	**16 MHz**	**12 MHz**
FF_H	25.6 µs	32.0 µs	42.67 µs	1.64 ms	2.05 ms	2.73 ms
$7F_H$	3.3 ms	4.13 ms	5.5 ms	211 ms	264 ms	352 ms
00_H	6.55 ms	8.19 ms	10.92 ms	419 ms	524 ms	699 ms

Note: For safety reasons, the user is advised to rewrite WDTCON each time before the watchdog timer is serviced.

13.2 Reset Source Indication

The reset indication flags in register WDTCON provide information on the source for the last reset. As the C161RI starts executing from location 00’0000$_H$ after any possible reset event the initialization software may check these flags in order to determine if the recent reset event was triggered by an external hardware signal (via $\overline{\text{RSTIN}}$), by software itself or by an overflow of the watchdog timer. The initialization (and also the further operation) of the microcontroller system can thus be adapted to the respective circumstances, e.g. a special routine may verify the software integrity after a watchdog timer reset.

The reset indication flags are not mutually exclusive, i.e. more than one flag may be set after reset depending on its source. The table below summarizes the possible combinations:

Reset Indication Flag Combinations

	Reset Indication Flags			
Reset Source	**LHWR**	**SHWR**	**SWR**	**WDTR**
Long Hardware Reset	X	X	X	
Short Hardware Reset	*	X	X	
Software Reset	*	*	X	
Watchdog Timer Reset	*	*	X	X

*) When the bidirectional reset mode is enabled, the indicated flags are also set in the respective reset case. The WDTCON reset value will then be different from the table value.

Note: The listed reset values for WDTCON assume the reserved bits as ‘0’.

Long Hardware Reset is indicated when the $\overline{\text{RSTIN}}$ input is still sampled low (active) at the end of a hardware triggered internal reset sequence.

Short Hardware Reset is indicated when the $\overline{\text{RSTIN}}$ input is sampled high (inactive) at the end of a hardware triggered internal reset sequence.

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

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

Note: When bidirectional reset is enabled the $\overline{\text{RSTIN}}$ pin is pulled low for the duration of the internal reset sequence upon any sort of reset.
Therefore a long hardware reset (LHWR) will be recognized in any case.

14 The Real Time Clock

The Real Time Clock (RTC) module of the C161RI basically is an independent timer chain which is clocked directly with the oscillator clock and serves for different purposes:

- System clock to determine the current time and date
- Cyclic time based interrupt
- 48-bit timer for long term measurements

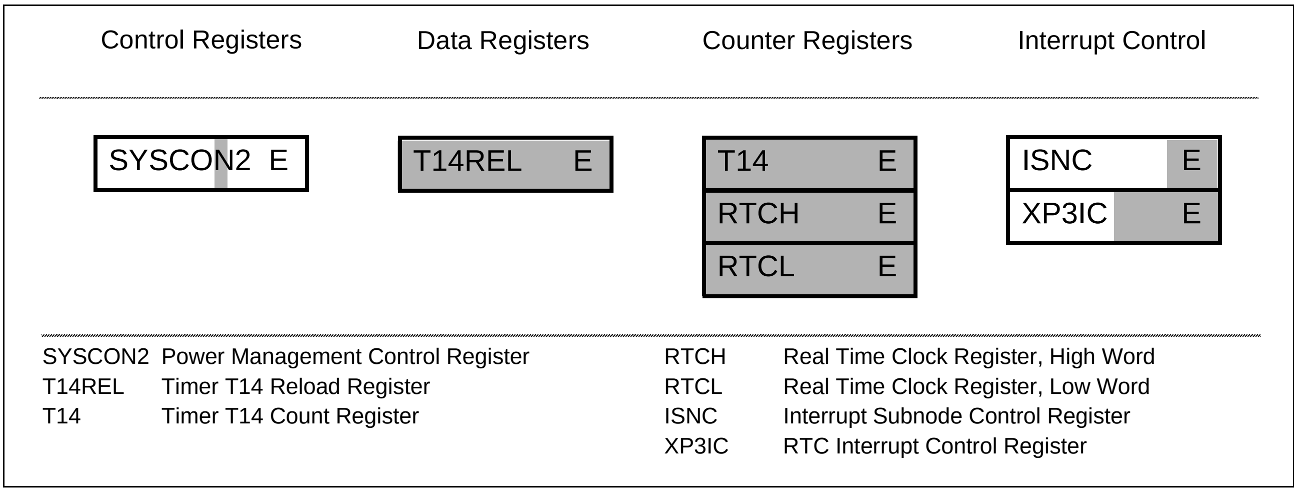


Figure 14-1
SFRs Associated with the RTC Module

The RTC module consists of a chain of 3 divider blocks, a fixed 8:1 divider, the reloadable 16-bit timer T14 and the 32-bit RTC timer (accessible via registers RTCH and RTCL). Both timers count up.

The clock signal for the RTC module is directly derived from the on-chip oscillator frequency (not from the CPU clock) and fed through a separate clock driver. It is therefore independent from the selected clock generation mode of the C161RI and is controlled by the clock generation circuitry.

RTC Register Location within the ESFR Space

Register Name	Long/Short Address	Reset Value	Notes
T14	F0D2$_H$ / 69$_H$	UUUU$_H$	Prescaler timer, generates input clock for RTC register and periodic interrupt
T14REL	F0D0$_H$ / 68$_H$	UUUU$_H$	Timer reload register
RTCH	F0D6$_H$ / 6B$_H$	UUUU$_H$	High word of RTC register
RTCL	F0D4$_H$ / 6A$_H$	UUUU$_H$	Low word of RTC register

Note: The RTC registers are not affected by a reset. After a power on reset, however, they are undefined.

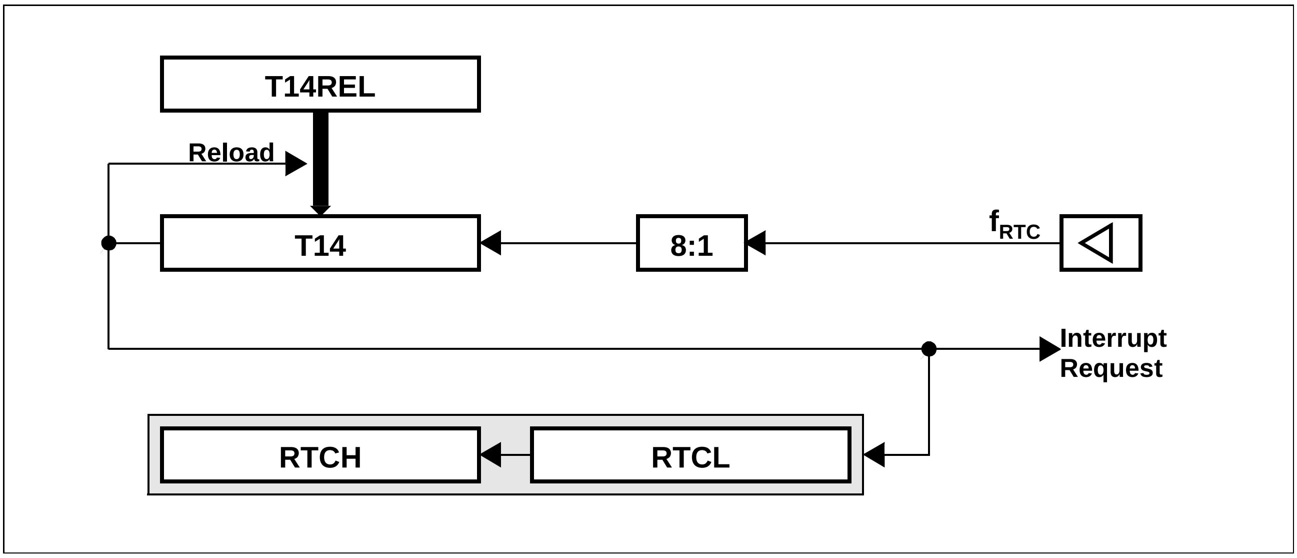


Figure 14-2
RTC Block Diagram

System Clock Operation

A real time system clock can be maintained that keeps on running also during idle mode and power down mode (optionally) and represents the current time and date. This is possible as the RTC module is not effected by a reset.
The maximum resolution (minimum stepwidth) for this clock information is determined by timer T14's input clock. The maximum usable timespan is achieved when T14REL is loaded with 0000_H and so T14 divides by 2^{16}.

Cyclic Interrupt Generation

The RTC module can generate an interrupt request whenever timer T14 overflows and is reloaded. This interrupt request may e.g. be used to provide a system time tick independent of the CPU frequency without loading the general purpose timers, or to wake up regularly from idle mode. The interrupt cycle time can be adjusted via the timer T14 reload register T14REL. Please refer to "RTC Interrupt Generation" below for more details.

48-bit Timer Operation

The concatenation of the 16-bit reload timer T14 and the 32-bit RTC timer can be regarded as a 48-bit timer which is clocked with the RTC input frequency divided by the fixed prescaler. The reload register T14REL should be cleared to get a 48-bit binary timer. However, any other reload value may be used.
The maximum usable timespan is 2^{48} ($\approx 10^{14}$) T14 input clocks, which would equal more than 100 years at an oscillator frequency of 20 MHz.

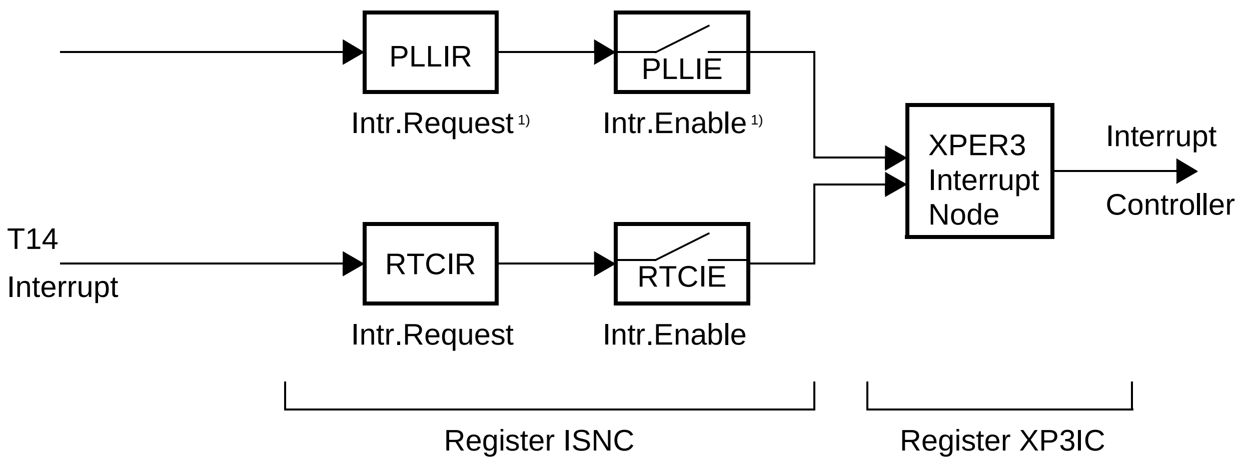
RTC Register Access

The actual value of the RTC is represented by the 3 registers T14, RTCL and RTCH. As these registers are concatenated to build the RTC counter chain, internal overflows occur while the RTC is running. When reading or writing the RTC value make sure to account for such internal overflows in order to avoid reading/writing corrupted values. When reading/writing e.g. 0000_H to RTCH and then accessing RTCL will produce a corrupted value as RTCL may overflow before it can be accessed. In this case, however, RTCH would be 0001_H. The same precautions must be taken for T14 and T14REL.

14.1 RTC Interrupt Generation

The RTC interrupt shares the XPER3 interrupt node with the PLL/OWD interrupt (if available). This is controlled by the interrupt subnode control register ISNC. The interrupt handler can determine the source of an interrupt request via the separate interrupt request and enable flags (see figure below) provided in register ISNC.

Note: If only one source is enabled no additional software check is required, of course.



Note: 1) Only available if PLL is implemented

Figure 14-3
RTC Interrupt Logic

If T14 interrupts are to be used both stages, the interrupt node (XP3IE = ’1’) and the RTC subnode (RTCIE = ’1’) must be enabled.

Please note that the node request bit XP3IR is automatically cleared when the interrupt handler is vectored to, while the subnode request bit T14IR must be cleared by software.

Defining the RTC Time Base

The reload timer T14 determines the input frequency of the RTC timer, i.e. the RTC time base, as well as the T14 interrupt cycle time. The table below lists the interrupt period range and the T14 reload values (for a time base of 1 s and 1 ms) for several oscillator frequencies:

RTC Interrupt Periods and Reload Values

Oscillator Frequency		RTC Interrupt Period		Reload Value A		Reload Value B	
		Minimum	Maximum	T14REL	Base	T14REL	Base
32.768 KHz	Aux.	244.14 µs	16.0 s	F000$_H$	1.000 s	FFFC$_H$	0.977 ms
32 KHz	Aux.	250 µs	16.38 s	F060$_H$	1.000 s	FFFC$_H$	1.000 ms
32 KHz	Main	8000 µs	524.29 s	FF83$_H$	1.000 s	----	----
4 MHz	Main	64.0 µs	4.19 s	C2F7$_H$	1.000 s	FFF0$_H$	1.024 ms
5 MHz	Main	51.2 µs	3.35 s	B3B5$_H$	0.999 s	FFEC$_H$	1.024 ms
8 MHz	Main	32.0 µs	2.10 s	85EE$_H$	1.000 s	FFE1$_H$	0.992 ms
10 MHz	Main	25.6 µs	1.68 s	676A$_H$	0.999 s	FFD9$_H$	0.998 ms
12 MHz	Main	21.3 µs	1.40 s	48E5$_H$	1.000 s	FFD2$_H$	1.003 ms
16 MHz	Main	16.0 µs	1.05 s	0BDC$_H$	1.000 s	FFC2$_H$	0.992 ms

Increased RTC Accuracy through Software Correction

The accuracy of the C161RI’s RTC is determined by the oscillator frequency and by the respective prescaling factor (excluding or including T14). The accuracy limit generated by the prescaler is due to the quantization of a binary counter (where the average is zero), while the accuracy limit generated by the oscillator frequency is due to the difference between ideal and real frequency (and therefore accumulates over time). The total accuracy of the RTC can be further increased via software for specific applications that demand a high time accuracy.

The key to the improved accuracy is the knowledge of the exact oscillator frequency. The relation of this frequency to the expected ideal frequency is a measure for the RTC’s deviation. The number N of cycles after which this deviation causes an error of ± 1 cycle can be easily computed. So the only action is to correct the count by ± 1 after each series of N cycles.

This correction may be applied to the RTC register as well as to T14.

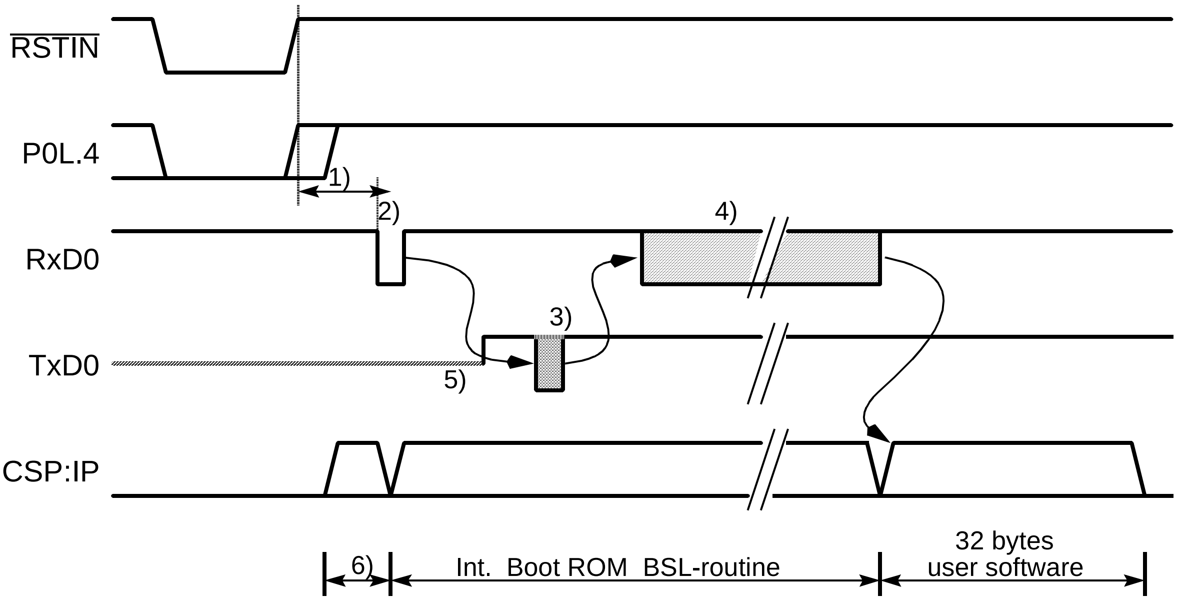
Also the correction may be done cyclic, e.g. within T14’s interrupt service routine, or by evaluating a formula when the RTC registers are read (for this the respective “last” RTC value must be available somewhere).

Note: For the majority of applications, however, the standard accuracy provided by the RTC’s structure will be more than sufficient.

15 The Bootstrap Loader

The built-in bootstrap loader of the C161RI provides a mechanism to load the startup program, which is executed after reset, via the serial interface. In this case no external memory or an internal ROM/OTP/Flash is required for the initialization code starting at location $00'0000_H$.

The bootstrap loader moves code/data into the internal RAM, but it is also possible to transfer data via the serial interface into an external RAM using a second level loader routine. ROM memory (internal or external) is not necessary. However, it may be used to provide lookup tables or may provide “core-code”, i.e. a set of general purpose subroutines, e.g. for IO operations, number crunching, system initialization, etc.



1) BSL initialization time, > 2 µs @ f_{CPU} = 20 MHz.
2) Zero byte (1 start bit, eight ‘0’ data bits, 1 stop bit), sent by host.
3) Identification byte, sent by C161RI.
4) 32 bytes of code / data, sent by host.
5) Caution: TxD0 is only driven a certain time after reception of the zero byte (2.5 µs @ f_{CPU} = 20 MHz).
6) Internal Boot ROM.

Figure 15-1
Bootstrap Loader Sequence

The Bootstrap Loader may be used to load the complete application software into ROMless systems, it may load temporary software into complete systems for testing or calibration, it may also be used to load a programming routine for Flash devices.

The BSL mechanism may be used for standard system startup as well as only for special occasions like system maintenance (firmware update) or end-of-line programming or testing.

Entering the Bootstrap Loader

The C161RI enters BSL mode if pin P0L.4 is sampled low at the end of a hardware reset. In this case the built-in bootstrap loader is activated independent of the selected bus mode. The bootstrap loader code is stored in a special Boot-ROM, no part of the standard mask ROM, OTP or Flash memory area is required for this.

After entering BSL mode and the respective initialization the C161RI scans the RXD0 line to receive a zero byte, i.e. one start bit, eight '0' data bits and one stop bit. From the duration of this zero byte it calculates the corresponding baudrate factor with respect to the current CPU clock, initializes the serial interface ASC0 accordingly and switches pin TxD0 to output. Using this baudrate, an identification byte is returned to the host that provides the loaded data.

This identification byte identifies the device to be bootet. The following codes are defined:

55_H:	8xC166.
$A5_H$:	Previous versions of the C167 (obsolete).
$B5_H$:	C165.
$C5_H$:	C167 derivatives.
$D5_H$:	All devices equipped with identification registers.

Note: The identification byte $D5_H$ does not directly identify a specific derivative. This information can in this case be obtained from the identification registers.

When the C161RI has entered BSL mode, the following configuration is automatically set (values that deviate from the normal reset values, are ***marked***):

Watchdog Timer:	***Disabled***	Register STKUN:	$FA40_H$
Context Pointer CP:	$FA00_H$	Register STKOV:	$FA0C_H$ 0<->C
Stack Pointer SP:	$FA40_H$	Register BUSCON0:	acc. to startup config.
Register S0CON:	***8011_H***	P3.10 / TXD0:	'***1***'
Register S0BG:	acc. to '00' byte	DP3.10:	'***1***'

Other than after a normal reset the watchdog timer is disabled, so the bootstrap loading sequence is not time limited. Pin TXD0 is configured as output, so the C161RI can return the identification byte.

Note: Even if the internal ROM/OTP/Flash is enabled, no code can be executed out of it.

The hardware that activates the BSL during reset may be a simple pull-down resistor on P0L.4 for systems that use this feature upon every hardware reset. You may want to use a switchable solution (via jumper or an external signal) for systems that only temporarily use the bootstrap loader.

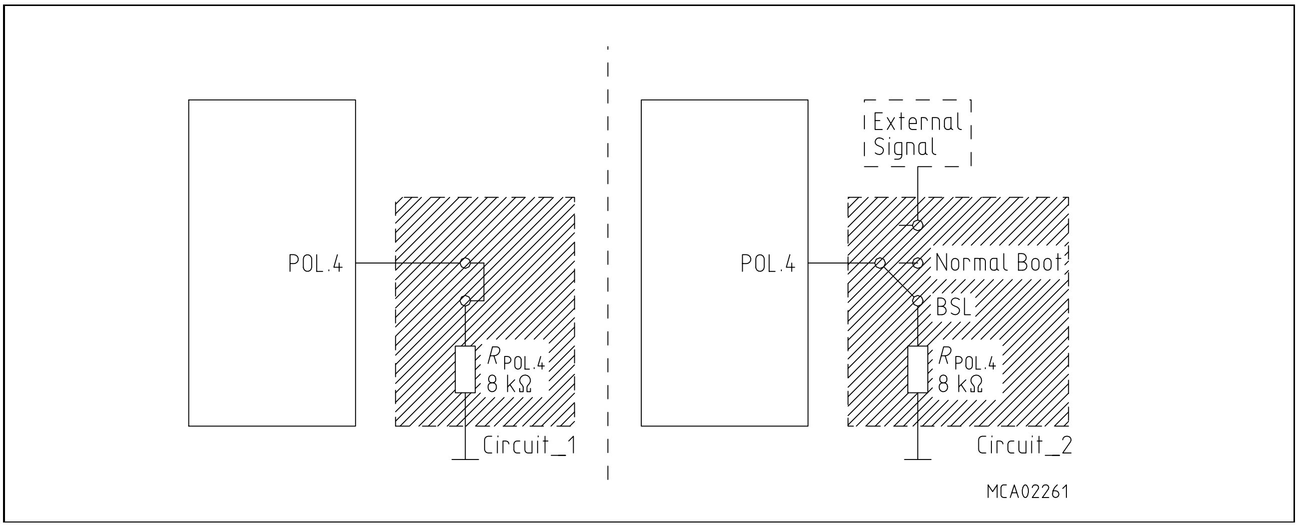


Figure 15-2
Hardware Provisions to Activate the BSL

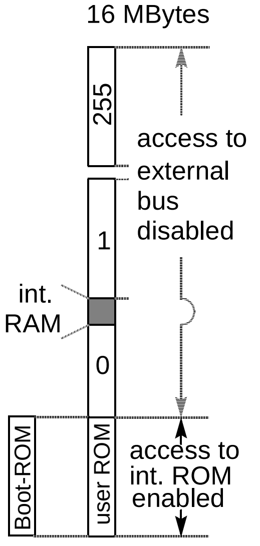
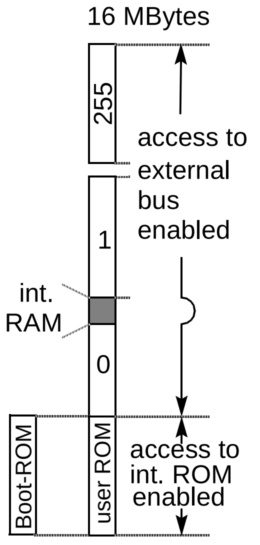
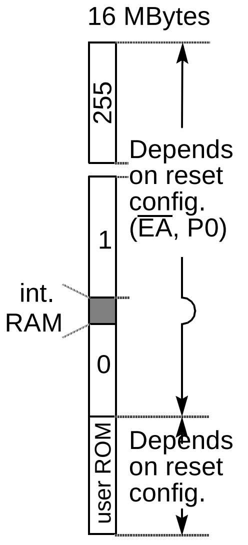
After sending the identification byte the ASC0 receiver is enabled and is ready to receive the initial 32 bytes from the host. A half duplex connection is therefore sufficient to feed the BSL.

Note: In order to properly enter BSL mode it is not only required to pull P0L.4 low, but also pins P0L.2, P0L.3, P0L.5 must receive defined levels. This is described in chapter “System Reset”.

Memory Configuration after Reset

The configuration (i.e. the accessibility) of the C161RI's memory areas after reset in Bootstrap-Loader mode differs from the standard case. Pin $\overline{EA}$ is not evaluated when BSL mode is selected, and accesses to the internal code memory are partly redirected, while the C161RI is in BSL mode (see table below). All code fetches are made from the special Boot-ROM, while data accesses read from the internal code memory. Data accesses will return undefined values on ROMless devices.

Note: The code in the Boot-ROM is not an invariant feature of the C161RI. User software should not try to execute code from the internal ROM area while the BSL mode is still active, as these fetches will be redirected to the Boot-ROM.
The Boot-ROM will also "move" to segment 1, when the internal ROM area is mapped to segment 1.

			
BSL mode active	**Yes** (P0L.4 = '0')	**Yes** (P0L.4 = '0')	**No** (P0L.4 = '1')
$\overline{EA}$ pin	high	low	acc. to application
Code fetch from internal ROM area	Boot-ROM access	Boot-ROM access	User ROM access
Data fetch from internal ROM area	User ROM access	User ROM access	User ROM access

Loading the Startup Code

After sending the identification byte the BSL enters a loop to receive 32 bytes via ASC0. These bytes are stored sequentially into locations $00'FA40_H$ through $00'FA5F_H$ of the internal RAM. So up to 16 instructions may be placed into the RAM area. To execute the loaded code the BSL then jumps to location $00'FA40_H$, i.e. the first loaded instruction. The bootstrap loading sequence is now terminated, the C161RI remains in BSL mode, however. Most probably the initially loaded routine will load additional code or data, as an average application is likely to require substantially more than 16 instructions. This second receive loop may directly use the pre-initialized interface ASC0 to receive data and store it to arbitrary user-defined locations.

This second level of loaded code may be the final application code. It may also be another, more sophisticated, loader routine that adds a transmission protocol to enhance the integrity of the loaded code or data. It may also contain a code sequence to change the system configuration and enable the bus interface to store the received data into external memory.

This process may go through several iterations or may directly execute the final application. In all cases the C161RI will still run in BSL mode, i.e. with the watchdog timer disabled and limited access to the internal code memory. All code fetches from the internal ROM area ($00'0000_H$ … $00'7FFF_H$ or $01'0000_H$ … $01'7FFF_H$, if mapped to segment 1) are redirected to the special Boot-ROM. Data fetches access will access the internal code memory of the C161RI, if any is available, but will return undefined data on ROMless devices.

Exiting Bootstrap Loader Mode

In order to execute a program in normal mode, the BSL mode must be terminated first. The C161RI exits BSL mode upon a software reset (ignores the level on P0L.4) or a hardware reset (P0L.4 must be high then!). After a reset the C161RI will start executing from location $00'0000_H$ of the internal ROM or the external memory, as programmed via pin $\overline{EA}$.

Choosing the Baudrate for the BSL

The calculation of the serial baudrate for ASC0 from the length of the first zero byte that is received, allows the operation of the bootstrap loader of the C161RI with a wide range of baudrates. However, the upper and lower limits have to be kept, in order to insure proper data transfer.

$$\mathbf{B_{C161RI}} = \frac{f_{CPU}}{32 \cdot (S0BRL + 1)}$$

The C161RI uses timer T6 to measure the length of the initial zero byte. The quantization uncertainty of this measurement implies the first deviation from the real baudrate, the next deviation is implied by the computation of the S0BRL reload value from the timer contents. The formula below shows the association:

$$S0BRL = \frac{T6 - 36}{72} \quad , \qquad T6 = \frac{9}{4} \cdot \frac{f_{CPU}}{B_{Host}}$$

For a correct data transfer from the host to the C161RI the maximum deviation between the internal initialized baudrate for ASC0 and the real baudrate of the host should be below 2.5%. The deviation (F_B, in percent) between host baudrate and C161RI baudrate can be calculated via the formula below:

$$F_B = \left| \frac{B_{Contr} - B_{Host}}{B_{Contr}} \right| \cdot 100 \quad \% \quad , \qquad F_B \leq 2{,}5 \quad \%$$

Note: **F**unction (F_B) does not consider the tolerances of oscillators and other devices supporting the serial communication.

This baudrate deviation is a nonlinear function depending on the CPU clock and the baudrate of the host. The maxima of the function (F_B) increase with the host baudrate due to the smaller baudrate prescaler factors and the implied higher quantization error (see figure below).

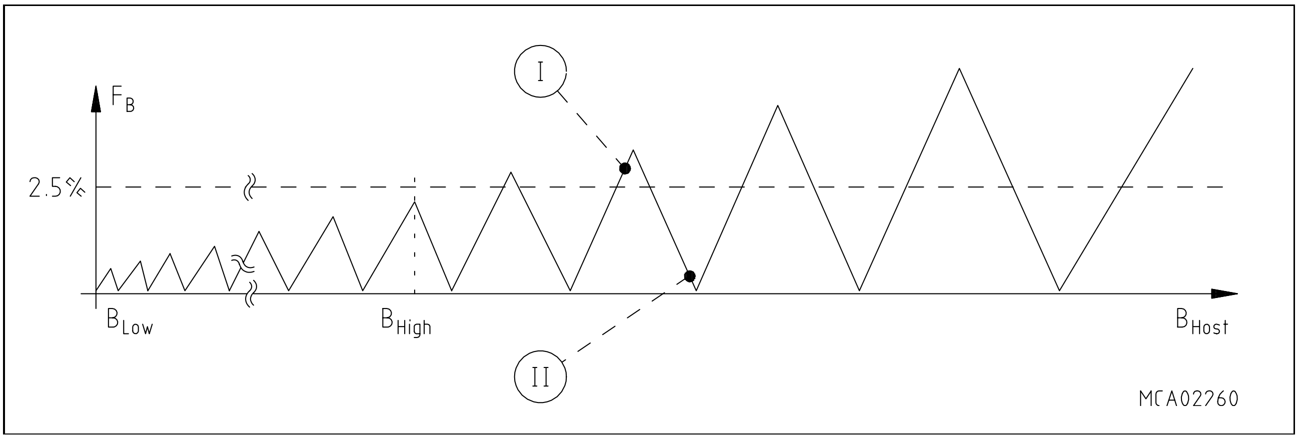


Figure 15-3
Baudrate deviation between host and C161RI

The minimum baudrate (B_{Low} in the figure above) is determined by the maximum count capacity of timer T6, when measuring the zero byte, i.e. it depends on the CPU clock. Using the maximum T6 count 2^{16} in the formula the minimum baudrate for f_{CPU} = 16 MHz is 549 Baud. The lowest standard baudrate in this case would be 600 Baud. Baudrates below B_{Low} would cause T6 to overflow. In this case ASC0 cannot be initialized properly.

The maximum baudrate (B_{High} in the figure above) is the highest baudrate where the deviation still does not exceed the limit, i.e. all baudrates between B_{Low} and B_{High} are below the deviation limit. The maximum standard baudrate that fulfills this requirement is 19200 Baud.

Higher baudrates, however, may be used as long as the actual deviation does not exceed the limit. A certain baudrate (marked I) in the figure) may e.g. violate the deviation limit, while an even higher baudrate (marked II) in the figure) stays very well below it. This depends on the host interface.

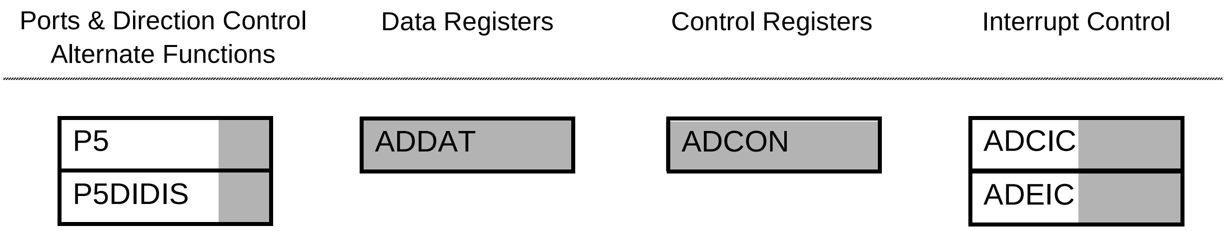
16 The Analog / Digital Converter

The C161RI provides an Analog / Digital Converter with 8-bit resolution and a sample & hold circuit on-chip. A multiplexer selects between up to 4 analog input channels (alternate functions of Port 5).

The ADC supports the following conversion modes:

- **Fixed Channel Single Conversion**
  produces just one result from the selected channel
- **Fixed Channel Continuous Conversion**
  repeatedly converts the selected channel

A set of SFRs and port pins provide access to control functions and results of the ADC.



P5	Port 5 Analog Input Port: AN0/P5.0...AN3/P5.3	ADCIC	A/D Converter Interrupt Control Register
P5DIDIS	Port 5 Digital Input Disable Register		(End of Conversion)
ADDAT	A/D Converter Result Register	ADEIC	A/D Converter Interrupt Control Register
ADCON	A/D Converter Control Register		(Overrun Error)

Figure 16-1
SFRs and Port Pins associated with the A/D Converter

The external analog reference voltages V_{AREF} and V_{AGND} are fixed. The separate supply for the ADC reduces the interference with other digital signals.

The conversion time is programmable, so the ADC can be adjusted to the internal resistances of the analog sources and/or the analog reference voltage supply.

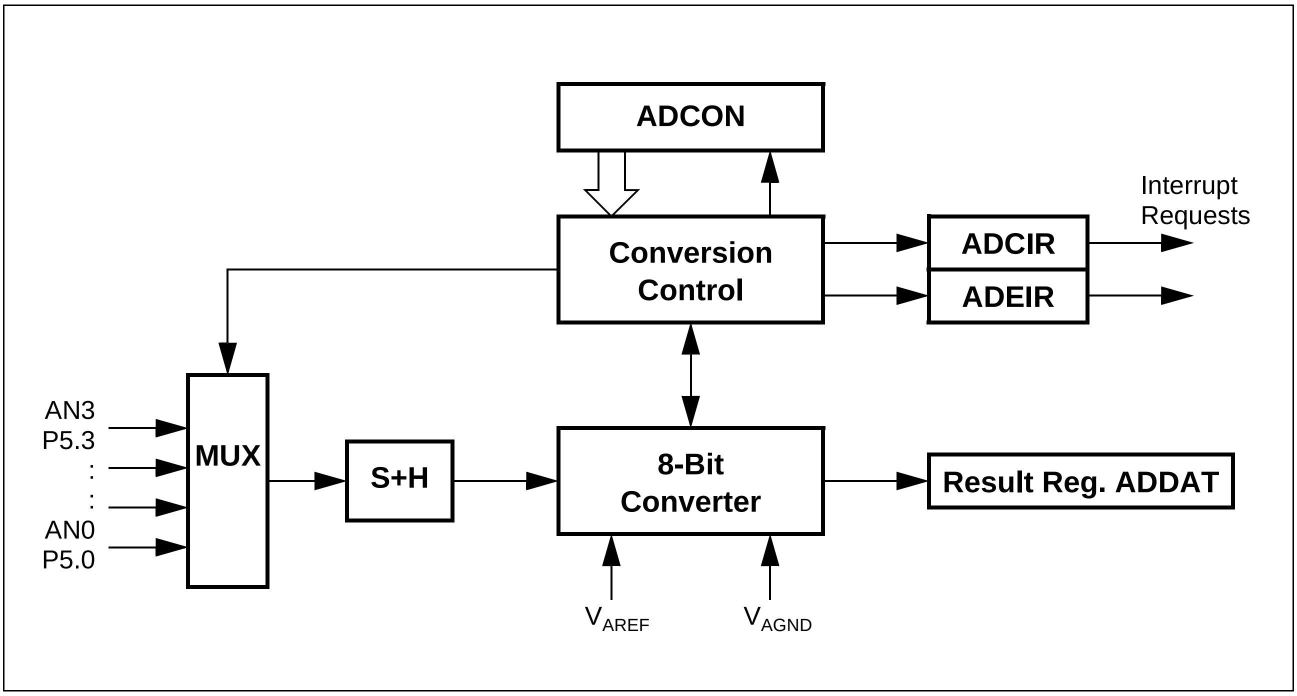


Figure 16-2
Analog / Digital Converter Block Diagram

16.1 Mode Selection and Operation

The analog input channels AN0 … AN3 are alternate functions of Port 5 which is an input-only port. The Port 5 lines may either be used as analog or digital inputs. For pins that shall be used as analog inputs it is recommended to disable the digital input stage via register P5DIDIS. This avoids undesired cross currents and switching noise while the (analog) input signal level is between V_{IL} and V_{IH}.

The functions of the A/D converter are controlled by the bit-addressable A/D Converter Control Register ADCON. Its bitfields specify the analog channel to be acted upon, the conversion mode, and also reflect the status of the converter.

ADCON (FFA0$_{\mathrm{H}}$ / D0$_{\mathrm{H}}$) SFR **Reset Value: 0000$_{\mathrm{H}}$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADCTC		**-**	**-**	**-**	**-**	**-**	**AD BSY**	**AD ST**	**AD RP**	**-**	**ADM**	**-**	**-**	**ADCH**	
rw		-	-	-	-	-	r	rw	rw	-	rw	-	-	rw	

Bit	Function
ADCH	**ADC Analog Channel Input Selection** Selects the ADC channel which is to be converted.
ADM	**ADC Mode Selection** 0: Fixed Channel Single Conversion 1: Fixed Channel Continuous Conversion
ADRP	**ADC Result Position** 0: 8-bit result is written to ADDAT.2 … ADDAT.9 1: 8-bit result is written to ADDAT.0 … ADDAT.7
ADST	**ADC Start Bit** 0: Stop a running conversion 1: Start conversion(s)
ADBSY	**ADC Busy Flag** 0: ADC is idle 1: A conversion is active.
ADCTC	**ADC Conversion Time Control** (Defines the ADC basic conversion clock f_{BC}) 00: $f_{\mathrm{BC}} = f_{\mathrm{CPU}} / 2$ 01: $f_{\mathrm{BC}} = f_{\mathrm{CPU}} / 4$ 10: $f_{\mathrm{BC}} = f_{\mathrm{CPU}} / 8$ 11: $f_{\mathrm{BC}} = f_{\mathrm{CPU}} / 16$

Bit field ADCH specifies the analog input channel which is to be converted. Bitfield ADM selects the operating mode of the A/D converter. A conversion (or a sequence) is then started by setting bit ADST. Clearing ADST stops the A/D converter after a certain operation which depends on the selected operating mode.

The busy flag (read-only) ADBSY is set, as long as a conversion is in progress.

The result of a conversion is stored in the result register ADDAT.

ADDAT (FEA0_H / 50_H) SFR **Reset Value: 0000_H**

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

Bit	Function
ADRES	**A/D Conversion Result** The 8-bit result is stored right- or left-adjusted according to bit ADRP. ADRP = ’0’: Result in ADRES.9-2 (left-adjusted, ADRES.1-0 = ’00’) ADRP = ’1’: Result in ADRES.7-0 (right-adjusted, ADRES.9-8 = ’00’)

A conversion is started by setting bit ADST = ‘1’. The busy flag ADBSY will be set and the converter then selects and samples the input channel, which is specified by the channel selection field ADCH in register ADCON. The sampled level will then be held internally during the conversion. When the conversion of this channel is complete, the 8-bit result is transferred into the result register ADDAT and the interrupt request flag ADCIR is set. The conversion result is placed into bitfield ADRES of register ADDAT left- or right-adjusted, depending on bit ADRP in register ADCON.

If bit ADST is reset via software, while a conversion is in progress, the A/D converter will stop after the current conversion.

Setting bit ADST while a conversion is running, will abort this conversion and start a new conversion with the parameters specified in ADCON.

Note: Abortion and restart (see above) are triggered by bit ADST changing from ‘0’ to ‘1’, i.e. ADST must be ‘0’ before being set.

While a conversion is in progress, the mode selection field ADM and the channel selection field ADCH may be changed. These bitfields will be evaluated after the current conversion.

Fixed Channel Conversion Modes

These modes are selected by programming the mode selection bitfield ADM in register ADCON to ‘0_B’ (single conversion) or to ‘1_B’ (continuous conversion). After starting the converter through bit ADST the busy flag ADBSY will be set and the channel specified in bit field ADCH will be converted. After the conversion is complete, the interrupt request flag ADCIR will be set.

In Single Conversion Mode the converter will automatically stop and reset bits ADBSY and ADST.

In Continuous Conversion Mode the converter will automatically start a new conversion of the channel specified in ADCH. ADCIR will be set after each completed conversion.

When bit ADST is reset by software, while a conversion is in progress, the converter will complete the current conversion and then stop and reset bit ADBSY.

16.2 Conversion Timing Control

When a conversion is started, first the capacitances of the converter are loaded via the respective analog input pin to the current analog input voltage. The time to load the capacitances is referred to as sample time. Next the sampled voltage is converted to a digital value in successive steps, which correspond to the resolution of the ADC. During these phases (except for the sample time) the internal capacitances are repeatedly charged and discharged via pins V_{AREF} and V_{AGND}.

The current that has to be drawn from the sources for sampling and changing charges depends on the time that each respective step takes, because the capacitors must reach their final voltage level within the given time, at least with a certain approximation. The maximum current, however, that a source can deliver, depends on its internal resistance.

The time that the two different actions during conversion take (sampling, and converting) can be programmed within a certain range in the C161RI relative to the CPU clock. The absolute time that is consumed by the different conversion steps therefore is independent from the general speed of the controller. This allows adjusting the A/D converter of the C161RI to the properties of the system:

Fast Conversion can be achieved by programming the respective times to their absolute possible minimum. This is preferable for scanning high frequency signals. The internal resistance of analog source and analog supply must be sufficiently low, however.

High Internal Resistance can be achieved by programming the respective times to a higher value, or the possible maximum. This is preferable when using analog sources and supply with a high internal resistance in order to keep the current as low as possible. The conversion rate in this case may be considerably lower, however.

The conversion time is programmed via the upper two bits of register ADCON. Bitfield ADCTC (conversion time control) selects the basic conversion clock, used for the operation of the A/D converter. The sample time is derived from this conversion clock. The table below lists the possible combinations. The timings refer to CPU clock cycles, where $t_{CPU} = 1 / f_{CPU}$.

The limit values for f_{BC} (see data sheet) must not be exceeded when selecting ADCTC and f_{CPU}.

ADCON.15|14 (ADCTC)	A/D Converter Basic Clock f_{BC}	Sample Time t_S	Conversion Time t_C
00	f_{CPU} / 2	$t_{CPU} \times 12$	$t_{CPU} \times 62$
01	f_{CPU} / 4	$t_{CPU} \times 24$	$t_{CPU} \times 122$
10	f_{CPU} / 8	$t_{CPU} \times 48$	$t_{CPU} \times 242$
11	f_{CPU} / 16	$t_{CPU} \times 96$	$t_{CPU} \times 482$

The time for a complete conversion includes the sample time (6 t_{BC}), the conversion itself and the time required to transfer the digital value to the result register (2 t_{CPU}) as shown in the example below.

Converter Timing Example

Assumptions: f_{CPU} = 16 MHz (i.e. t_{CPU} = 62.5 ns), ADCTC = '01'.

Basic clock f_{BC} = f_{CPU} / 4 = 4 MHz, i.e. t_{BC} = 250 ns.
Sample time t_S = $t_{BC} \times 6$ = 1500 ns.
Conversion time t_C = 30 t_{BC} + 2 t_{CPU} = (7500 + 125) ns = 7.625 µs.

Note: For the exact specification please refer to the data sheet of the selected derivative.

16.3 A/D Converter Interrupt Control

At the end of each conversion, interrupt request flag ADCIR in interrupt control register ADCIC is set. This end-of-conversion interrupt request may cause an interrupt to vector ADCINT, or it may trigger a PEC data transfer which reads the conversion result from register ADDAT e.g. to store it into a table in the internal RAM for later evaluation.

The interrupt request flag ADEIR in register ADEIC will be set if a conversion result overwrites a previous value in register ADDAT (error interrupt in standard mode). This interrupt request may be used to cause an interrupt to vector ADEINT.

ADCIC ($FF98_H$ / CC_H) SFR **Reset Value: - - 00_H**

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

ADEIC ($FF9A_H$ / CD_H) SFR **Reset Value: - - 00_H**

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

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

17 The I^{2}C-Bus Module

The on-chip I^{2}C-bus module (Inter Integrated Circuit) connects the C161RI to other external controllers and/or peripherals via the two-line serial I^{2}C interface. The I^{2}C-Bus module provides communication at data rates of up to 400 Kbit/s in master and/or slave mode and features 7-bit addressing as well as 10-bit addressing.

Note: The I^{2}C-Bus module is an XBUS peripheral and therefore requires bit XPEN in register SYSCON to be set in order to be operable.

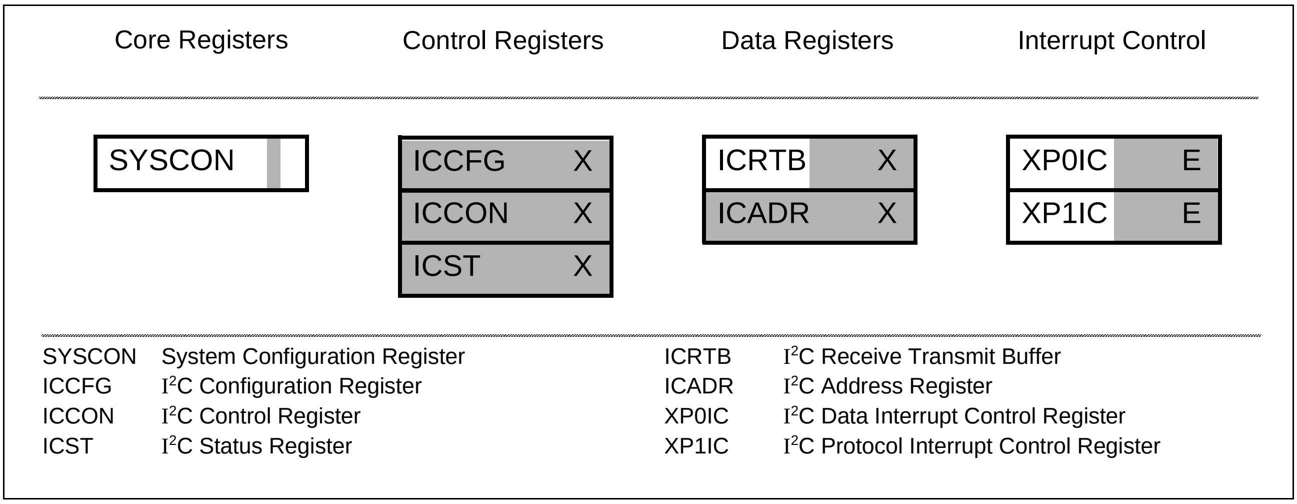


Figure 17-1
SFRs Associated with the I^{2}C-Bus Module

The module can operate in three different modes:

- **Master mode**, where the C161RI controls the bus transactions and provides the clock signal.
- **Slave mode**, where an external master controls the bus transactions and provides the clock signal.
- **Multimaster mode**, where several masters can be connected to the bus, i.e. the C161RI can be master or slave.

The on-chip I^{2}C-bus module allows efficient communication over the common I^{2}C bus. The module unloads the CPU of the C161RI of low level tasks like

- (De)Serialization of bus data
- Generation of start and stop conditions
- Monitoring the bus lines in slave mode
- Evaluation of the device address in slave mode
- Bus access arbitration in multimaster mode

17.1 I²C-Bus Conditions

Data is transferred over the 2-line I²C bus (SDA, SCL) using a protocol that ensures reliable and efficient transfers. This protocol clearly distinguishes regular data transfers from defined control signals which control the data transfers.

The following bus conditions are defined:

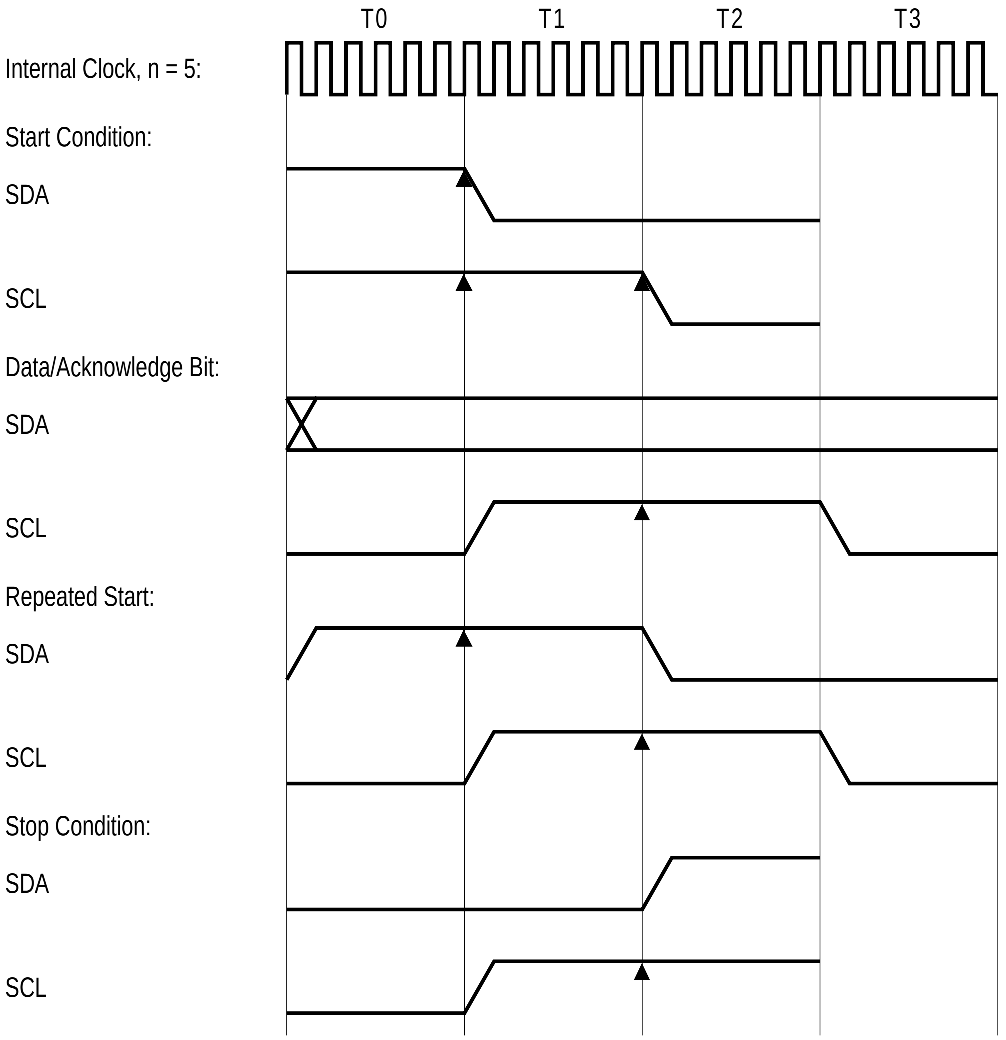
Bus Idle: SDA and SCL remain high. The I²C bus is currently not used.

Data Valid: SDA stable during the high phase of SCL. SDA then represents the transferred bit. There is one clock pulse for each transferred bit of data.
During data transfers SDA may only change while SCL is low (see below)!

Start Transfer: A falling edge on SDA (↘) while SCL is high indicates a start condition.
This start condition initiates a data transfer over the I²C bus.

Stop Transfer: A rising edge on SDA (↗) while SCL is high indicates a stop condition.
This stop condition terminates a data transfer. Between a start condition and a stop condition an arbitrary number of bytes may be transferred.

The figure below gives examples for these bus conditions.



↑ The high level of the signal is verified.
If the signal is low, the previous state (Ti) is repeated.
The length of each state is 1 ... 256 CPU clock cycles, as defined by bitfield BRP in register ICCFG (in the above axample n = 5, ie. BRP = 04_H)

Figure 17-2
I²C-Bus Conditions

17.2 The Physical I²C-Bus Interface

Communication via the I²C Bus uses two bidirectional lines, the serial data line SDA and the serial clock line SCL. These two generic interface lines can each be connected to a number of IO port lines of the C161RI (see figure below). These connections can be established and released under software control.

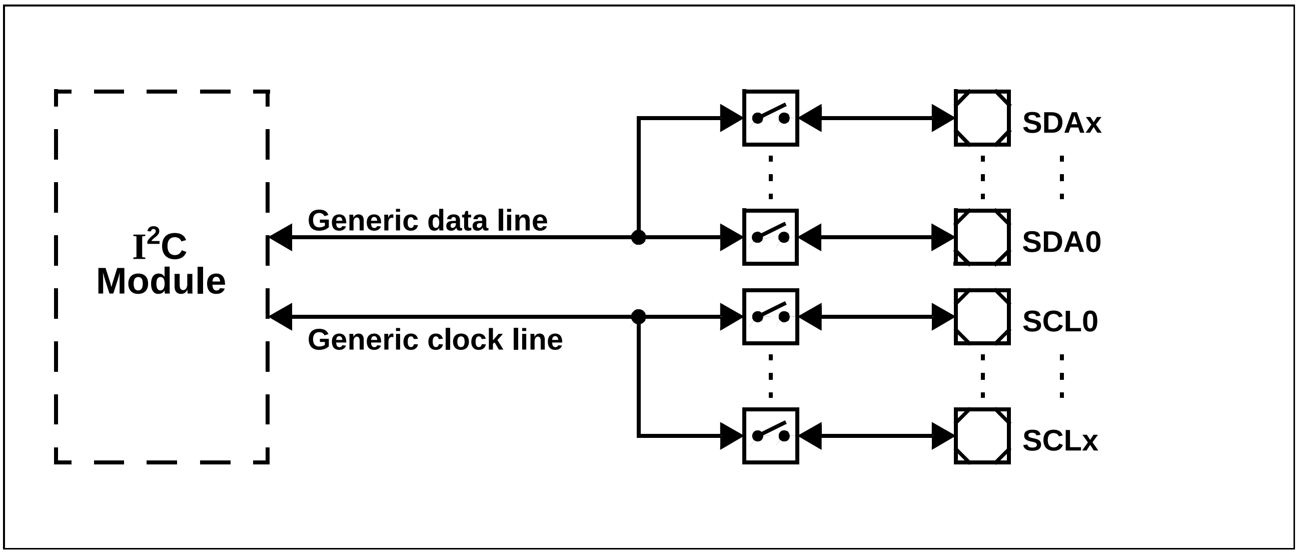


Figure 17-3
I²C-Bus Line Connections

This mechanism allows a number of configurations of the physical I²C-bus interface:

Channel switching: The I²C module can be connected to a specific pair of pins (e.g. SDA0 and SCL0) which then forms a separate I²C channel to the external system. The channel can be dynamically switched by connecting the module to another pair of pins (e.g. SDA1 and SCL1). This establishes physically separate interface channels.

Broadcasting: Connecting the module to more than one pair of pins (e.g. SDA0/1 and SCL0/1) allows the transmission of messages over multiple physical channels at the same time. Please note that this configuration is critical when the C161RI is a slave or receives data.

Note: Never change the physical bus interface configuration while a transfer is in progress.

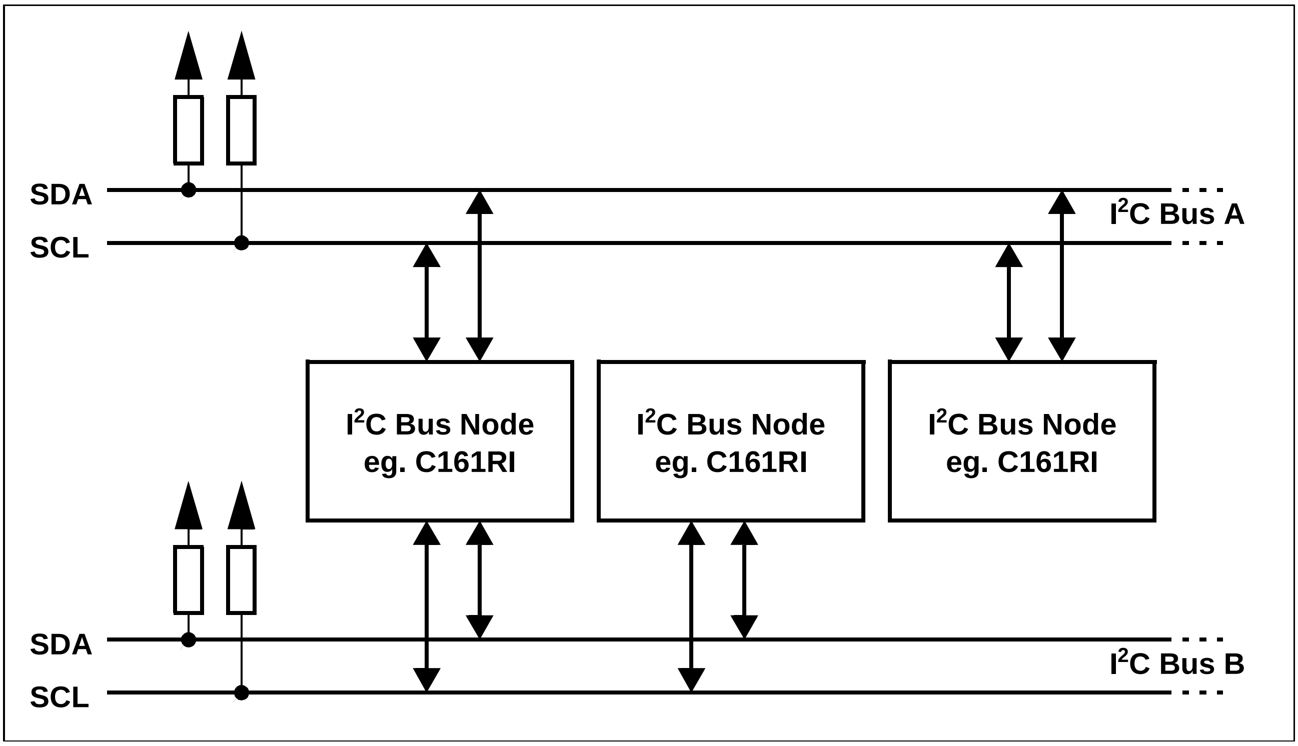


Figure 17-4
Physical Bus Configuration Example

Output Pin Configuration

The pin drivers that are assigned to the I²C channel(s) provide open drain outputs (i.e. no upper transistor). This ensures that the I²C module does not put any load on the I²C-bus lines while the C161RI is not powered. The I²C-bus lines therefore require external pullup resistors (approx. 10 KΩ for operation at 100 KBaud, 2 KΩ for operation at 400 KBaud).

Note: If the pins that are assigned to the I²C channel(s) are to be used as general purpose IO they must be used for open drain outputs or as inputs.

All pins of the C161RI that are to be used for I²C-bus communication must be switched to output and their alternate function must be enabled (by setting the respective port output latch to ‘1’), before any communication can be established.

If not driven by the I²C module (i.e. the corresponding enable bit in register ICCFG is ‘0’) they then switch off their drivers (i.e. driving ‘1’ to an open drain output). Due to the external pullup devices the respective bus levels will then be ‘1’ which is idle.

The I²C module features digital input filters in order to improve the rejection of noise from the external bus lines.

17.3 Operating the I²C Bus

The on-chip I²C-bus module of the C161RI can be operated in variety of operating modes.

Master or Slave operation can be selected, so theI²C module can control the external bus (master) or can be controlled via the bus (slave) by a remote master.

7-bit or 10-bit addressing can be selected, so the I²C module can communicate with standard 7-bit devices as well as with more sophisticated 10-bit devices.

100 KBd or 400 KBd transfer speed can be selected, so the I²C module can communicate with slow devices conforming to the standard I²C-bus specification as well as with fast devices conforming to the extended specification.

Physical channels can be selected, so the I²C module can use electrically separated channels or increase the addressing range by using more data lines.

Note: Baudrate and physical channels should never be changed (via ICCFG) during a transfer.

Operation in Master Mode

If the on-chip I²C module shall control the I²C bus (i.e. be bus master) master mode must be selected via bitfield MOD in register ICCON. The physical channel is configured by a control word written to register ICCFG, defining the active interface pins and the used baudrate. More than one SDA and/or SCL line may be active at a time. The address of the remote slave that is to be accessed is written to ICRTB. The bus is claimed by setting bit BUM in register ICCON. This generates a start condition on the bus and automatically starts the transmission of the address in ICRTB. Bit TRX in register ICCON defines the transfer direction (TRX = '1', i.e. transmit, for the slave address). A repeated start condition is generated by setting bit RSC in register ICCON, which automatically starts the transmission of the address previously written to ICRTB. This may be used to change the transfer direction. RSC is cleared automatically after the repeated start condition has been generated.

The bus is released by clearing bit BUM in register ICCON. This generates a stop condition on the bus.

Operation in Multimaster Mode

If multimaster mode is selected via bitfield MOD in register ICCON the on-chip I²C module can operate concurrently as a bus master or as a slave. The descriptions of these modes apply accordingly.

Multimaster mode implies that several masters are connected to the same bus. As more than one master may try to claim the bus at a given time an arbitration is done on the SDA line. When a master device detects a mismatch between the data bit to be sent and the actual level on the SDA (bus) line it looses the arbitration and automatically switches to slave mode (leaving the other device as the remaining master). This loss of arbitration is indicated by bit AL in register ICST which must be checked by the driver software when operating in multimaster mode. Lost arbitration is also indicated when the software tries to claim the bus (by setting bit BUM) while the I²C module is operating in slave mode (indicated by bit BB = '1').

Bit AL must be cleared via software.

Operation in Slave Mode

If the on-chip I²C module shall be controlled via the I²C bus by a remote master (i.e. be a bus slave) slave mode must be selected via bitfield MOD in register ICCON. The physical channel is configured by a control word written to register ICCFG, defining the active interface pins and the used baudrate. It is recommended to have only one SDA and SCL line active at a time when operating in slave mode. The address by which the slave module can be selected is written to register ICADR.

The I²C module is selected by another master when it receives (after a start condition) either its own device address (stored in ICADR) or the general call address (00_H). In this case an interrupt is generated and bit SLA in register ICST is set indicating the valid selection. The desired transfer mode is then selected via bit TRX (TRX = ’0’ for reception, TRX = ’1’ for transmission).

For a transmission the respective data byte is placed into the buffer ICRTB (which automatically sets bit TRX) and the acknowledge behavior is selected via bit ACKDIS.

For a reception the respective data byte is fetched from the buffer ICRTB after IRQD has been activated.

In both cases the data transfer itself is enabled by clearing bit IRQP which releases the SCL line.

When a stop condition is detected bit SLA is cleared.

The I²C-bus configuration register ICCFG selects the bus baudrate as well as the activation of SDA and SCL lines. So an external I²C channel can be established (baudrate and physical lines) with one single register access.

Systems that utilize several I²C channels can prepare a set of control words which configure the respective channels. By writing one of these control words to ICCFG the respective channel is selected. Different channels may use different baudrates. Also different operating modes can be selected, e.g. enabling all physical interfaces for a broadcast transmission.

Note: See also section “The Physical I²C-Bus Interface“.

ICCFG ($ED00_H$) **XReg** **Reset Value: $XX00_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BRP								**-**	**-**	**SCL SEL1**	**SCL SEL0**	**-**	**SDA SEL2**	**SDA SEL1**	**SDA SEL0**
rw								-	-	rw	rw	-	rw	rw	rw

Bit	Function
SDASELx	**SDA Pin Selection** These bits determine to which pins the I²C data line is connected. 0: SDA pin x is disconnected. 1: SDA pin x is connected with I²C data line.
SCLSELx	**SCL Pin Selection** These bits determine to which pins the I²C clock line is connected. 0: SCL pin x is disconnected. 1: SCL pin x is connected with I²C clock line.
BRP	**Baudrate Prescaler** Determines the baudrate for the active I²C channel(s). The resulting baudrate is $B_{I2C} = f_{CPU} / (4 \times (BRP + 1))$. See table below.

I²C-Bus Baudrate Selection

CPU Frequency f_{CPU}	Reload Value for BRP	
	100 KBd	**400 KBd**
20 MHz	31_H	$0B_H$ or $0C_H$
16 MHz	27_H	09_H
12 MHz	$1D_H$	06_H or 07_H
10 MHz	18_H	05_H
1 MHz	01_H or 02_H	*Not possible.*

ICCON (ED02$_H$) **XReg** **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	TRX	AIR DIS	ACK DIS	BUM	MOD		RSC	M10
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw		rw	rw

Bit	Function
M10	**Address Mode** 0: 7-bit addressing using ICA.7-1. 1: 10-bit addressing using ICA.9-0.
RSC	**Repeated Start Condition** 0: No operation. RSC is cleared automatically after the repeated start condition has been sent. 1: Generate a repeated start condition in (multi)master mode. RSC cannot be set in slave mode.
MOD	**Basic Operating Mode** 00: I^2C module is disabled and initialized. 01: Slave mode. 10: Master mode. 11: Multi-Master mode.
BUM	**Busy Master** 0: Clearing bit BUM (↘) generates a stop condition. 1: Setting bit BUM generates a start condition in (multi)master mode. **Note**: Setting BUM (↗) while BB = '1' generates an arbitration lost situation. In this case BUM is cleared and bit AL is set. BUM cannot be set in slave mode.
ACKDIS	**Acknowledge Pulse Disable** 0: An acknowledge pulse is generated for each received frame. 1: No acknowledge pulse is generated.
AIRDIS	**Auto Interrupt Reset Disable** 0: IRQD is cleared automatically upon a read/write access to ICRTB. (Advantageous if data are read/written via PEC transfers) 1: IRQD must explicitly be cleared via software. (Allows to trigger a stop condition after the last data transfer before the bus is released by clearing IRQD.)
TRX	**Transmit Select** 0: Data is received from the I^2C bus. 1: Data is transmitted to the I^2C bus. **Note**: TRX is set automatically when writing to the transmit buffer ICRTB.

ICST (ED04$_H$) **XReg** **Reset Value: 000X$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**IRQP**	**IRQD**	**BB**	**LRB**	**SLA**	**AL**	**ADR**
-	-	-	-	-	-	-	-	-	rw	rw	r	r	r	rw	r

Bit	Function
ADR	**Address** Bit ADR is set after a start condition in slave mode until the address has been received (1 byte in 7-bit address mode, 2 bytes in 10-bit address mode).
AL	**Arbitration Lost** Bit AL is set when the I^{2}C module has tried to become master on the bus but has lost the arbitration. Operation is continued until the 9th clock pulse. Bit IRQP is set along with bit AL. Bit AL must be cleared via software.
SLA	**Slave** 0: The I^{2}C bus is not busy, or the module is in master mode. 1: The I^{2}C module has been selected as a slave (device address received).
LRB	**Last Received Bit** (undefined after reset) Bit LRB represents the last bit (ie. the acknowledge bit) of the last transmitted or received frame.
BB	**Bus Busy** 0: The I^{2}C bus is idle, ie. a stop condition has occurred. 1: The I^{2}C bus is active, ie. a start condition has occurred. **Note**:Bit BB is always '0' while the I^{2}C module is disabled.
IRQD	**I^{2}C Interrupt Request Bit for Data Transfer Events 1)** 0: No interrupt request pending. 1: A data transfer event interrupt request is pending. IRQD is set after the acknowledge bit of a byte has been received or transmitted, and is cleared automatically upon a read or write access to the buffer ICRTB if bit AIRDIS = '0'. IRQD must be cleared via software if bit AIRDIS = '1'.
IRQP	**I^{2}C Interrupt Request Bit for Protocol Events 1)** 0: No interrupt request pending. 1: A protocol event interrupt request is pending. IRQP is set when bit SLA or bit AL is set (↗), and must be cleared via software.

1) While either IRQD or IRQP is set and the I^{2}C module is in master mode or has been selected as a slave, the I^{2}C clock line is held low which prevents further transfers on the I^{2}C bus.
The clock line (i.e. the I^{2}C bus) is released when both IRQD **and** IRQP are cleared. Only in this case the next I^{2}C-bus action can take place.
Note that IRQD is cleared automatically upon a read or write access to register ICRTB if bit AIRDIS is not set.
Both interrupt request bits may be set or cleared via software, e.g. to control the I^{2}C bus.

The I^{2}C address register ICADR stores the device address (ICA) which identifies the I^{2}C node when operating in slave mode. Bit M10 in register ICCON determines which part of ICADR is valid and used.

ICADR (ED06$_H$) **XReg** **Reset Value: 0XXX$_H$**

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

Bit	Function
ICA.7-1	**Address in 7-bit mode** (ICA.9, ICA.8, ICA.0 disregarded).
ICA.0-9	**Address in 10-bit mode** (all bits used).

The I^{2}C Receive/Transmit Buffer (ICRTB) accepts bytes to be transmitted and provides received bytes.

ICRTB (ED08$_H$) **XReg** **Reset Value: - - XX$_H$**

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

Bit	Function
ICData	**Transmit and shift data** This field accepts the byte to be transmitted or provides the received byte. **Note**:A data transfer event interrupt request (IRQD) is cleared automatically when reading from or writing to ICRTB, if bit AIRDIS = ’0’. If AIRDIS = ’1’ the request flag IRQD must be cleared via software.

Note: It is recommended not to access the receive/transmit buffer while a data transfer is in progress.

17.4 I²C Interrupt Control

The bit addressable interrupt control registers XP0IC and XP1IC are assigned to the I²C module. The occurrence of an interrupt request sets the respective interrupt request bit XP0IR/XP1IR. If this interrupt node is enabled (XPxEN = ’1’) a CPU interrupt is generated and arbitrated. These interrupt requests may be serviced via a standard service routine or with PEC transfers (see below). If polling of bits XP0IR and XP1IR is used please note that these request bits must be cleared via software.

Data transfer event interrupts are indicated by bit IRQD and allocated to vector **XP0INT**.
A data transfer event occurs after the acknowledge bit for a byte has been received or transmitted.

Protocol transfer event interrupts are indicated by bit IRQP and allocated to vector **XP1INT**.
A protocol transfer event occurs when bit SLA is set, i.e. a slave address is received, or when bit AL is set, i.e. the bus arbitration has been lost.

As long as either interrupt request flag (IRQD or IRQP) of the I²C-bus module is set the selected clock line(s) SCLx is/are held low. This disables any further transfer on the I²C bus and enables the driver software to react on the recent event. When both request bits are cleared the clock line(s) is/are released again and subsequent bus transfers can take place.

Note: The interrupt node request bits XP0IR and XP1IR are cleared automatically when the CPU services the respective interrupt (not in case of polling!).
The I²C-bus module interrupt request bit IRQP must be cleared via the driver software.
The I²C-bus module interrupt request bit IRQD is cleared automatically upon a read/write access to buffer ICRTB if bit AIRDIS = ’0’, otherwise it must be cleared via the driver software.

XP0IC (F186$_H$ / C3$_H$) **ESFR** **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								XP0IR	**XP0IE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

XP1IC (F18E$_H$ / C7$_H$) **ESFR** **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								XP1IR	**XP1IE**	**ILVL**				**GLVL**	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

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

17.5 Programming Example

The sample program below illustrates an I²C communication between the C161RI and an NVRAM (such as SDA2526 or SLA24C04). It uses 7-bit addressing with a slave address of 50_H which is concatenated with the Read/Write bit. This program does not use interrupts, but polls the corresponding I²C interrupt request flags.

The master (C161RI) starts in master transmitter mode and first sends the slave address ($A0_H = 50_H//0_B$) followed by the subaddress (00_H). The C161RI changes to master receiver mode, repeats the slave address ($A1_H = 50_H//1_B$) and then receives two bytes. The first byte is acknowledged (ACK = '0') by the master, the second byte is not acknowledged (ACK = '1'). The transfer is finished with a STOP condition by the master.

The following figure shows the waveforms for the described transfer. A programming example in "C" illustrates how the operation could be realized.

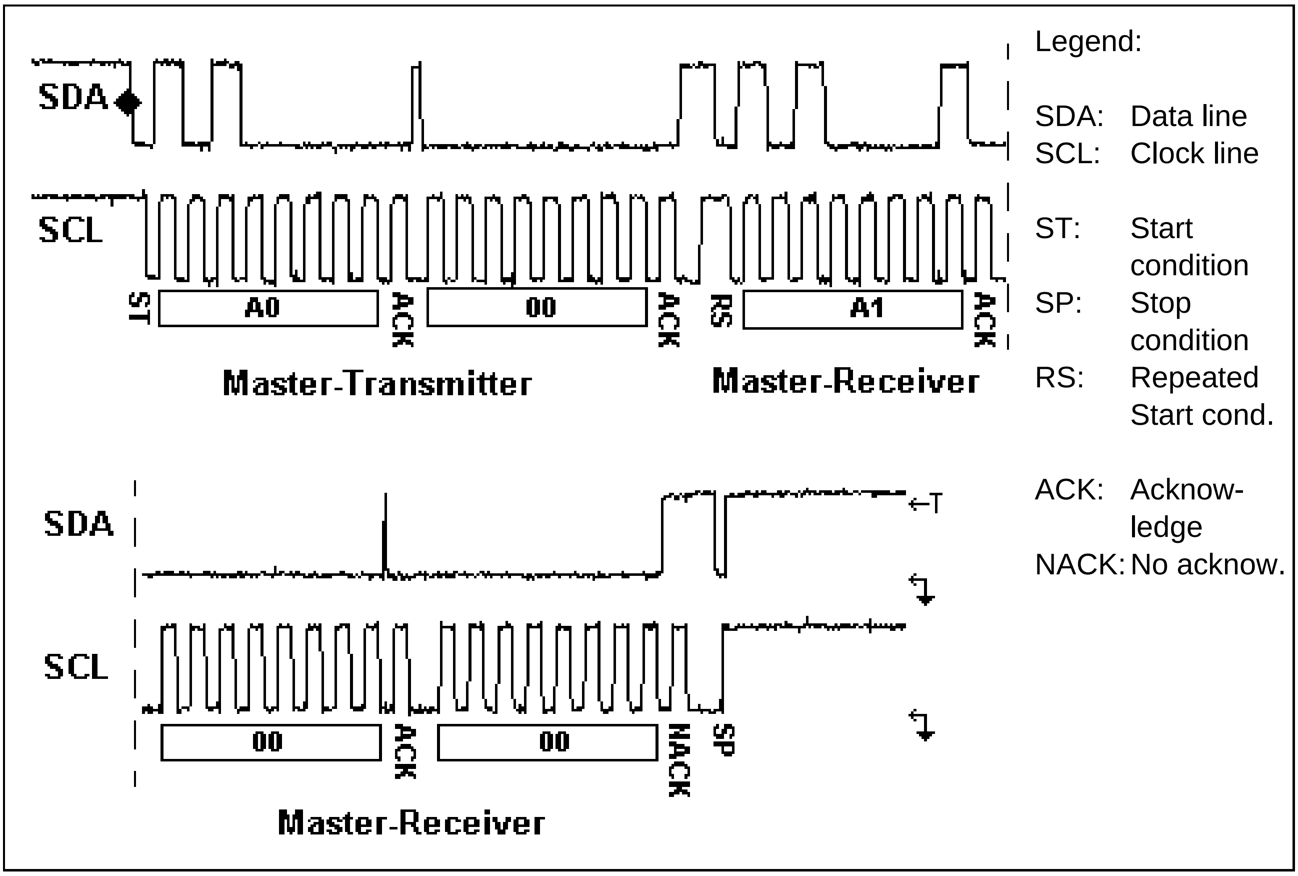


Figure 17-5
I²C-Bus Programming Example Waveforms

```
/*----------------------------------------------------------------*\
| Programming example to read 2 bytes from an NVRAM              |
| via the I2C bus.                                                |
\*----------------------------------------------------------------*/

void main() {

// X-peripheral enable:
SYSCON |= 0x0004;               // set XPEN, before EINIT-instr.!!!

// I2C control register configuration:
ICCON = 0x0008;                 // master mode
ICST  = 0x0000;                 // reset status register
ICCFG = 0x2711;                 // 100kHz @ 16MHz, SDA0, SCL0
XP0IC = 0x0000;                 // disable interrupt IRQD, use polling
XP1IC = 0x0000;                 // disable interrupt IRQP, use polling

// P3 configuration (provide external pullups on I2C-lines!):
_bfld_ (P3, 0x0003, 0x0003);    // enable alternate function on P3.0-1
_bfld_ (DP3, 0x0003, 0x0003);   // switch i2c pins to output

// slave address
ICRTB = 0x0000 | 0xA0;          // write transmit buffer
ICCON |= BUM;                   // BUM=1: start cond. + send slave addr.
while((ICST & IRQD) == 0x0000); // waiting for end of transmission
if (ICST & LRB)                 // ACK?
       {
       ICST &= ~AL;             // Clear bit AL
       ICST &= ~IRQP;           // Clear bit IRQP
       }

// sub address
ICRTB = 0x0000|0x0000;          // send sub-address
while((ICST & IRQD)== 0x0000);  // waiting for end of transmission
if (ICST & LRB)                 // ACK?
       {
       ICST &= ~AL;             // Clear bit AL
       ICST &= ~IRQP;           // Clear bit IRQP
       }

// switch to master-receiver, send slave address with a repeated start:
ICCON |= RSC;                   // repeated start condition
ICRTB = 0x0000|0xA1;            // write to transmit buffer
while((ICST & IRQD)== 0x0000);  // waiting for end of transmission
if (ICST & LRB)                 // ACK?
       {
       ICST &= ~AL;             // Clear bit AL
       ICST &= ~IRQP;           // Clear bit IRQP
       }
```

```
// drive clock (SCL) for first byte, give ack:
ICCON &= ~ACKDIS;                // acknowledge from master
ICCON &= ~TRX;                   // TRX = 0 for master receiver
dummy = ICRTB;                   // start clock to receive the 1st byte
while((ICST & IRQD)==0x0000);    // waiting for end of 1st byte

// read first byte, drive clock for second, give no ack:
ICCON |= ACKDIS;                 // no acknowledge from master
array[0] = ICRTB;                // read ICRTB (1st byte),
                                 // start clock to receive the 2nd byte
while((ICST & IRQD)==0x0000);    // waiting for end of 2nd byte

// read 2nd byte without automatic clear of IRQD, generate STOP:
ICCON |= AIRDIS;                 // AIRDIS=1: read ICRTB, send no clock
array[1] = ICRTB;                // read ICRTB (2nd byte)
ICCON &= ~BUM;                   // BUM=0: initiate stop condition
ICST  &= ~IRQD;                  // Clear bit IRQD

}
```

18 System Reset

The internal system reset function provides initialization of the C161RI into a defined default state and is invoked either by asserting a hardware reset signal on pin $\overline{\text{RSTIN}}$ (Hardware Reset Input), upon the execution of the SRST instruction (Software Reset) or by an overflow of the watchdog timer.

Whenever one of these conditions occurs, the microcontroller is reset into its predefined default state through an internal reset procedure. When a reset is initiated, pending internal hold states are cancelled and the current internal access cycle (if any) is completed. An external bus cycle is aborted, except for a watchdog reset (see description). After that the bus pin drivers and the IO pin drivers are switched off (tristate).

The internal reset procedure requires 516 CPU clock cycles in order to perform a complete reset sequence. This 516 cycle reset sequence is started upon a watchdog timer overflow, a SRST instruction or when the reset input signal $\overline{\text{RSTIN}}$ is latched low (hardware reset). The internal reset condition is active at least for the duration of the reset sequence and then until the $\overline{\text{RSTIN}}$ input is inactive. When this internal reset condition is removed (reset sequence complete, $\overline{\text{RSTIN}}$ inactive) the reset configuration is latched from PORT0 and $\overline{\text{RD}}$, and then pins ALE, $\overline{\text{RD}}$ and $\overline{\text{WR}}$ are driven to their inactive levels.

Note: Bit ADP which selects the Adapt mode is latched with the rising edge of $\overline{\text{RSTIN}}$.

After the internal reset condition is removed, the microcontroller will start program execution from memory location 00’0000$_H$ in code segment zero. This start location will typically hold a branch instruction to the start of a software initialization routine for the application specific configuration of peripherals and CPU Special Function Registers.

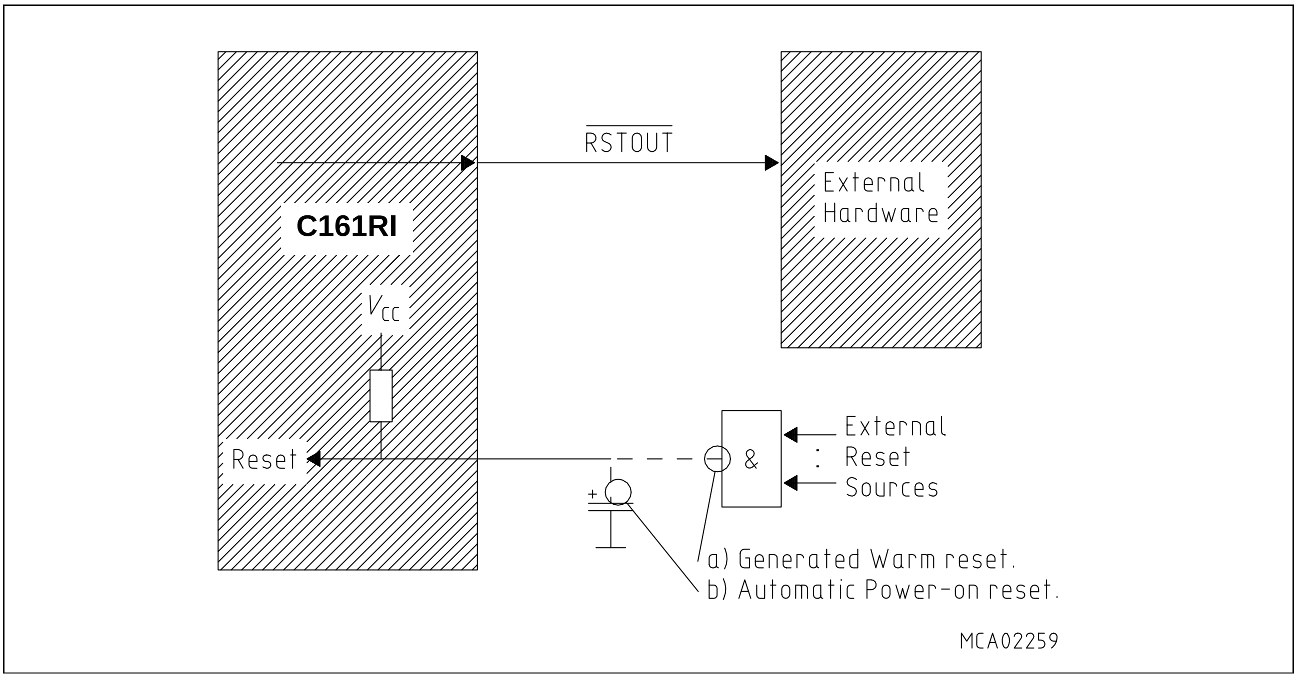


Figure 18-1
External Reset Circuitry

Hardware Reset

A hardware reset is triggered when the reset input signal $\overline{\text{RSTIN}}$ is latched low. To ensure the recognition of the $\overline{\text{RSTIN}}$ signal (latching), it must be held low for at least 2 CPU clock cycles. Also shorter $\overline{\text{RSTIN}}$ pulses may trigger a hardware reset, if they coincide with the latch's sample point. However, it is recommended to keep $\overline{\text{RSTIN}}$ low for ca. 1 ms. After the reset sequence has been completed, the $\overline{\text{RSTIN}}$ input is sampled. When the reset input signal is active at that time the internal reset condition is prolonged until $\overline{\text{RSTIN}}$ gets inactive.

During a hardware reset the PORT0 inputs for the reset configuration need some time to settle on the required levels, especially if the hardware reset aborts a read operation from an external peripheral. During this settling time the configuration may intermittently be wrong.

The input $\overline{\text{RSTIN}}$ provides an internal pullup device equalling a resistor of 50 KΩ to 150 KΩ (the minimum reset time must be determined by the lowest value). Simply connecting an external capacitor is sufficient for an automatic power-on reset (see b) in figure above). $\overline{\text{RSTIN}}$ may also be connected to the output of other logic gates (see a) in figure above). See also section “Bidirectional Reset” in this case).

Note: A power-on reset requires an active time of two reset sequences (1036 CPU clock cycles) after a stable clock signal is available (about 10 … 50 ms to allow the on-chip oscillator to stabilize).

Software Reset

The reset sequence can be triggered at any time via the protected instruction SRST (Software Reset). This instruction can be executed deliberately within a program, e.g. to leave bootstrap loader mode, or upon a hardware trap that reveals a system failure.

Note: A software reset disregards the configuration of P0L.5 … P0L.0.

Watchdog Timer Reset

When the watchdog timer is not disabled during the initialization or serviced regularly during program execution is will overflow and trigger the reset sequence. Other than hardware and software reset the watchdog reset completes a running external bus cycle if this bus cycle either does not use $\overline{\text{READY}}$ at all, or if $\overline{\text{READY}}$ is sampled active (low) after the programmed waitstates. When $\overline{\text{READY}}$ is sampled inactive (high) after the programmed waitstates the running external bus cycle is aborted. Then the internal reset sequence is started.

Note: A watchdog reset disregards the configuration of P0L.5 … P0L.0.
The watchdog reset cannot occur while the C161RI is in bootstrap loader mode!

Bidirectional Reset

In a special mode (Bidirectional reset) the C161RI's line $\overline{\text{RSTIN}}$ (normally an input) may be driven active by the chip logic e.g. in order to support external equipment which is required for startup (e.g. flash memory).

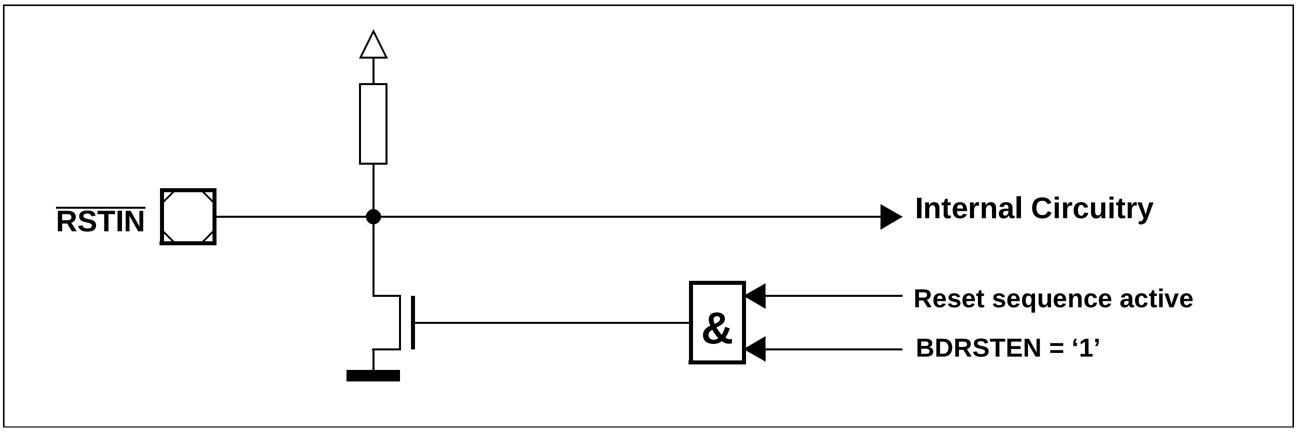


Figure 18-2
Bidirectional Reset Operation

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

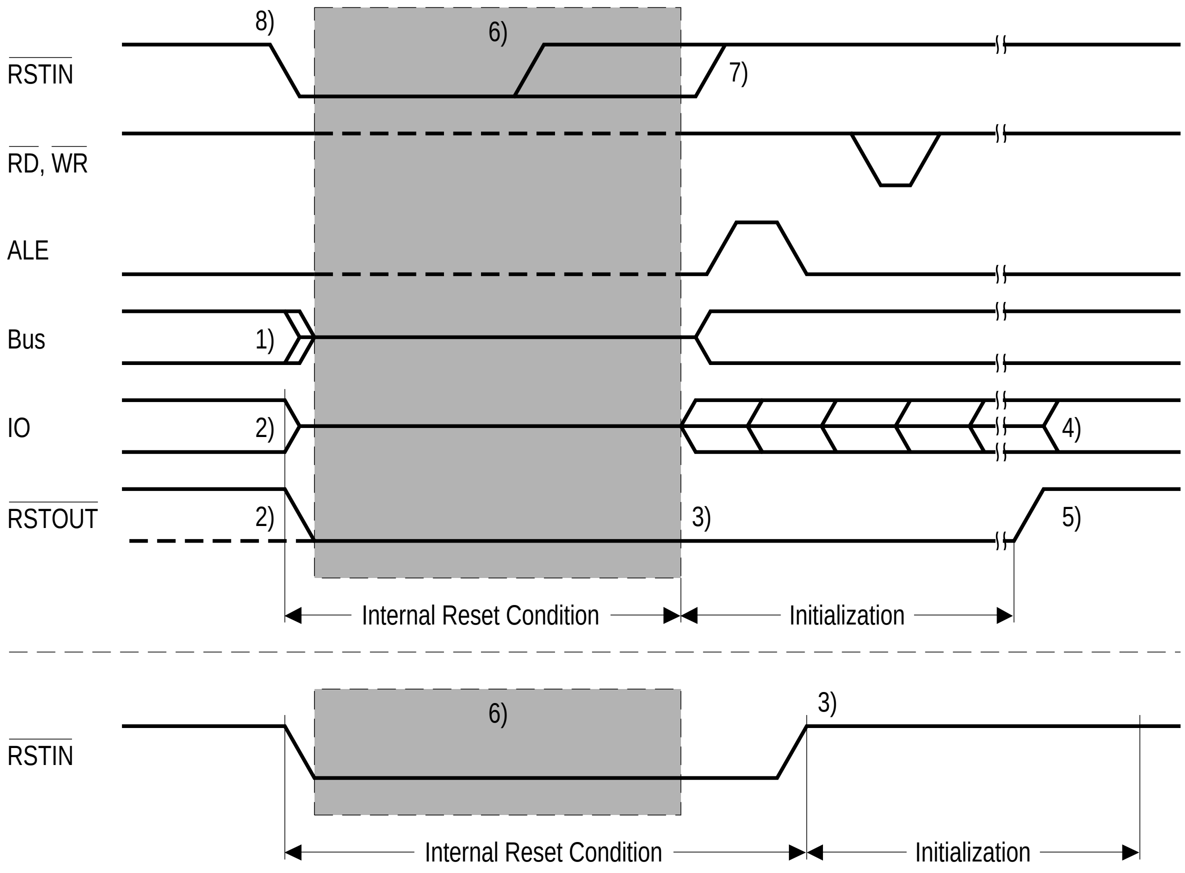
The Bidirectional reset function is useful in applications where external devices require a defined reset signal but cannot be connected to the C161RI's $\overline{\text{RSTOUT}}$ signal, e.g. an external flash memory which must come out of reset and deliver code well before $\overline{\text{RSTOUT}}$ can be deactivated via EINIT.

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

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

The C161RI's Pins after Reset

After the reset sequence the different groups of pins of the C161RI are activated in different ways depending on their function. Bus and control signals are activated immediately after the reset sequence according to the configuration latched from PORT0, so either external accesses can takes place or the external control signals are inactive. The general purpose IO pins remain in input mode (high impedance) until reprogrammed via software (see figure below). The $\overline{\text{RSTOUT}}$ pin remains active (low) until the end of the initialization routine (see description).



When the internal reset condition is extended by $\overline{\text{RSTIN}}$, the activation of the output signals is delayed until the end of the internal reset condition.

1) Current bus cycle is completed or aborted.
2) Switches asynchronously with $\overline{\text{RSTIN}}$, synchronously upon software or watchdog reset.
3) The reset condition ends here. The C161RI starts program execution.
4) Activation of the IO pins is controlled by software.
5) Execution of the EINIT instruction.
6) The shaded area designates the internal reset sequence, which starts after synchronization of $\overline{\text{RSTIN}}$.
7) A short hardware reset is extended until the end of the reset sequence in Bidirectional reset mode.
8) A software or WDT reset activates the $\overline{\text{RSTIN}}$ line in Bidirectional reset mode.

MCD03902

Figure 18-3
Reset Input and Output Signals

Reset Output Pin

The $\overline{\text{RSTOUT}}$ pin is dedicated to generate a reset signal for the system components besides the controller itself. $\overline{\text{RSTOUT}}$ will be driven active (low) at the begin of any reset sequence (triggered by hardware, the SRST instruction or a watchdog timer overflow). $\overline{\text{RSTOUT}}$ stays active (low) beyond the end of the internal reset sequence until the protected EINIT (End of Initialization) instruction is executed (see figure above). This allows the complete configuration of the controller including its on-chip peripheral units before releasing the reset signal for the external peripherals of the system.

Note: $\overline{\text{RSTOUT}}$ will float as long as pins P0L.0 and P0L.1 select emulation mode or adapt mode.

Watchdog Timer Operation after Reset

The watchdog timer starts running after the internal reset has completed. It will be clocked with the internal system clock divided by 2 (f_{CPU} / 2), and its default reload value is 00_H, so a watchdog timer overflow will occur 131072 CPU clock cycles ($2 * 2^{16}$) after completion of the internal reset, unless it is disabled, serviced or reprogrammed meanwhile. When the system reset was caused by a watchdog timer overflow, the WDTR (Watchdog Timer Reset Indication) flag in register WDTCON will be set to ’1’. This indicates the cause of the internal reset to the software initialization routine. WDTR is reset to ’0’ by an external hardware reset, by servicing the watchdog timer or after EINIT. After the internal reset has completed, the operation of the watchdog timer can be disabled by the DISWDT (Disable Watchdog Timer) instruction. This instruction has been implemented as a protected instruction. For further security, its execution is only enabled in the time period after a reset until either the SRVWDT (Service Watchdog Timer) or the EINIT instruction has been executed. Thereafter the DISWDT instruction will have no effect.

Reset Values for the C161RI Registers

During the reset sequence the registers of the C161RI are preset with a default value. Most SFRs, including system registers and peripheral control and data registers, are cleared to zero, so all peripherals and the interrupt system are off or idle after reset. A few exceptions to this rule provide a first pre-initialization, which is either fixed or controlled by input pins.

DPP1: 0001_H (points to data page 1)
DPP2: 0002_H (points to data page 2)
DPP3: 0003_H (points to data page 3)
CP: $FC00_H$
STKUN: $FC00_H$
STKOV: $FA00_H$
SP: $FC00_H$
WDTCON: $00XX_H$ (value depends on the reset source)
S0RBUF: XX_H (undefined)
SSCRB: $XXXX_H$ (undefined)
SYSCON: $0XX0_H$ (set according to reset configuration)
BUSCON0: $0XX0_H$ (set according to reset configuration)
RP0H: XX_H (reset levels of P0H)
ONES: $FFFF_H$ (fixed value)

The Internal RAM after Reset

The contents of the internal RAM are not affected by a system reset. However, after a power-on reset, the contents of the internal RAM are undefined. This implies that the GPRs (R15 … R0) and the PEC source and destination pointers (SRCP7 … SRCP0, DSTP7 … DSTP0) which are mapped into the internal RAM are also unchanged after a warm reset, software reset or watchdog reset, but are undefined after a power-on reset.

Ports and External Bus Configuration during Reset

During the internal reset sequence all of the C161RI's port pins are configured as inputs by clearing the associated direction registers, and their pin drivers are switched to the high impedance state. This ensures that the C161RI and external devices will not try to drive the same pin to different levels. Pin ALE is held low through an internal pulldown, and pins $\overline{\text{RD}}$, $\overline{\text{WR}}$ and $\overline{\text{READY}}$ are held high through internal pullups. Also the pins selected for $\overline{\text{CS}}$ output will be pulled high.

The registers SYSCON and BUSCON0 are initialized according to the configuration selected via PORT0.

When an external start is selected (pin $\overline{\text{EA}}$ = '0'):
- the Bus Type field (BTYP) in register BUSCON0 is initialized according to P0L.7 and P0L.6
- bit BUSACT0 in register BUSCON0 is set to '1'
- bit ALECTL0 in register BUSCON0 is set to '1'
- bit ROMEN in register SYSCON will be cleared to '0'
- bit BYTDIS in register SYSCON is set according to the data bus width
- bit WRCFG in register SYSCON is set according to pin P0H.0 (WRC)

When an internal start is selected (pin $\overline{\text{EA}}$ = '1'):
- register BUSCON0 is cleared to 0000_H
- bit ROMEN in register SYSCON will be set to '1'
- bit BYTDIS in register SYSCON is cleared, i.e. $\overline{\text{BHE}}/\overline{\text{WRH}}$ is enabled
- bit WRCFG in register SYSCON is set according to pin P0H.0 (WRC)

The other bits of register BUSCON0, and the other BUSCON registers are cleared. This default initialization selects the slowest possible external accesses using the configured bus type.

When the internal reset has completed, the configuration of PORT0, PORT1, Port 4 Port 6 and of the $\overline{\text{BHE}}$ signal (High Byte Enable, alternate function of P3.12) depends on the bus type which was selected during reset. When any of the external bus modes was selected during reset, PORT0 will operate in the selected bus mode. Port 4 will output the selected number of segment address lines (all zero after reset) and Port 6 will drive the selected number of $\overline{\text{CS}}$ lines ($\overline{\text{CS0}}$ will be '0', while the other active $\overline{\text{CS}}$ lines will be '1'). When no memory accesses above 64 K are to be performed, segmentation may be disabled.

When the on-chip bootstrap loader was activated during reset, pin TxD0 (alternate function of P3.10) will be switched to output mode after the reception of the zero byte.

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

Application-Specific Initialization Routine

After the internal reset condition is removed the C161RI fetches the first instruction from location 00'0000$_H$, which is the first vector in the trap/interrupt vector table, the reset vector. 4 words (locations 00'0000$_H$ through 00'0006$_H$) are provided in this table to start the initialization after reset. As a rule, this location holds a branch instruction to the actual initialization routine that may be located anywhere in the address space.

Note: When the Bootstrap Loader Mode was activated during a hardware reset the C161RI does not fetch instructions from location 00'0000$_H$ but rather expects data via serial interface ASC0.

If single chip mode is selected during reset, the first instruction is fetched from the internal ROM/OTP/Flash. Otherwise it is fetched from external memory. When internal ROM access is enabled after reset in single chip mode (bit ROMEN = '1' in register SYSCON), the software initialization routine may enable and configure the external bus interface before the execution of the EINIT instruction. When external access is enabled after reset, it may be desirable to reconfigure the external bus characteristics, because the BUSCON0 register is initialized during reset to the slowest possible memory configuration.

To decrease the number of instructions required to initialize the C161RI, each peripheral is programmed to a default configuration upon reset, but is disabled from operation. These default configurations can be found in the descriptions of the individual peripherals.

During the software design phase, portions of the internal memory space must be assigned to register banks and system stack. When initializing the stack pointer (SP) and the context pointer (CP) it must be ensured that these registers are initialized before any GPR or stack operation is performed. This includes interrupt processing, which is disabled upon completion of the internal reset and should remain disabled until the SP is initialized.

Note: Traps (incl. $\overline{\text{NMI}}$) may occur, even though the interrupt system is still disabled.

In addition, the stack overflow (STKOV) and the stack underflow (STKUN) registers should be initialized. After reset, the CP, SP, and STKUN registers all contain the same reset value 00'FC00$_H$, while the STKOV register contains 00'FA00$_H$. With the default reset initialization, 256 words of system stack are available, where the system stack selected by the SP grows downwards from 00'FBFE$_H$, while the register bank selected by the CP grows upwards from 00'FC00$_H$.

Based on the application, the user may wish to initialize portions of the internal memory before normal program operation. Once the register bank has been selected by programming the CP register, the desired portions of the internal memory can easily be initialized via indirect addressing.

At the end of the initialization, the interrupt system may be globally enabled by setting bit IEN in register PSW. Care must be taken not to enable the interrupt system before the initialization is complete in order to avoid e.g. the corruption of internal memory locations by stack operations using an uninitialized stack pointer.

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

The execution of the EINIT instruction …

- disables the action of the DISWDT instruction,
- disables write accesses to register SYSCON (all configurations regarding register SYSCON (enable CLKOUT, stacksize, etc.) must be selected before the execution of EINIT),
- disables write accesses to registers SYSCON2 and SYSCON3 (further write accesses to SYSCON2 and SYSCON3 can be executed only using a special unlock mechanism),
- clears the reset source detection bits in register WDTCON,
- causes the $\overline{\text{RSTOUT}}$ pin to go high (this signal can be used to indicate the end of the initialization routine and the proper operation of the microcontroller to external hardware).

18.1 System Startup Configuration

Although most of the programmable features of the C161RI are either selected during the initialization phase or repeatedly during program execution, there are some features that must be selected earlier, because they are used for the first access of the program execution (e.g. internal or external start selected via $\overline{\text{EA}}$).

These selections are made during reset via the pins of PORT0, which are read at the end of the internal reset sequence. During reset internal pullup devices are active on the PORT0 lines, so their input level is high, if the respective pin is left open, or is low, if the respective pin is connected to an external pulldown device. With the coding of the selections, as shown below, in many cases the default option, i.e. high level, can be used.

The value on the upper byte of PORT0 (P0H) is latched into register RP0H upon reset, the value on the lower byte (P0L) directly influences the BUSCON0 register (bus mode) or the internal control logic of the C161RI.

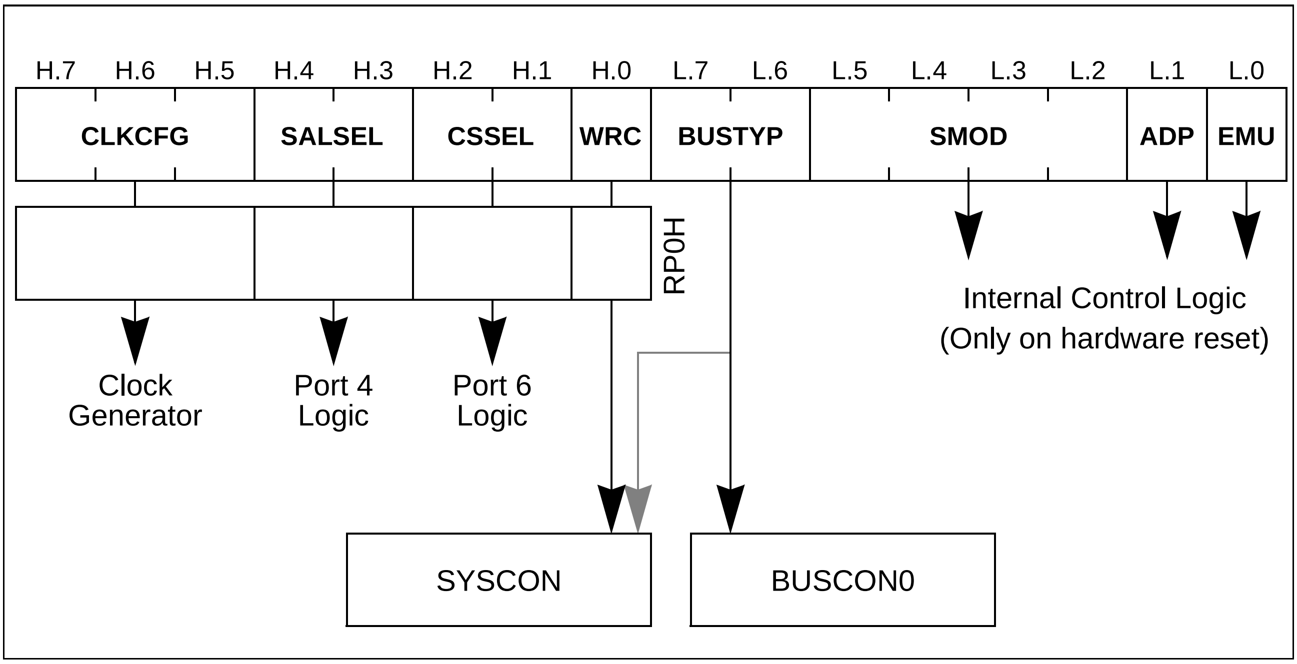


Figure 18-4
PORT0 Configuration during Reset

The pins that control the operation of the internal control logic and the reserved pins are evaluated only during a hardware triggered reset sequence. The pins that influence the configuration of the C161RI are evaluated during any reset sequence, i.e. also during software and watchdog timer triggered resets.

The configuration via P0H is latched in register RP0H for subsequent evaluation by software. Register RP0H is described in chapter "The External Bus Interface".

Note: The load on those pins that shall be latched as '1' must be small enough for the internal pullup device to keep their level high, or external pullup devices must ensure the high level. Those pins that shall be latched as '0' must be pulled low externally.
Make sure that the valid target levels are reached until the end of the reset sequence.
There is a specific application note to illustrate this.

The following describes the different selections that are offered for reset configuration. The default modes refer to pins at high level, i.e. without external pulldown devices connected.
Please also consider the note above.

Emulation Mode

Pin P0L.0 (EMU) selects the Emulation Mode, when low during reset. This mode allows the access to integrated XBUS peripherals via the external bus interface pins in application specific versions of the C161RI. In addition also the $\overline{\text{RSTOUT}}$ pin floats to tristate rather than be driven low. When the emulation mode has been latched the CLKOUT output is automatically enabled.

This mode is used for special emulator purposes and is of no use in basic C161RI devices, so P0L.0 should be held high.

Default: Emulation Mode is off.

Note: In emulation mode the direct drive clock option is selected with P0.15 (P0H.7) = '1'.

Adapt Mode

Pin P0L.1 (ADP) selects the Adapt Mode, when low during reset. In this mode the C161RI goes into a passive state, which is similar to its state during reset. The pins of the C161RI float to tristate or are deactivated via internal pullup/pulldown devices, as described for the reset state. In addition also the $\overline{\text{RSTOUT}}$ pin floats to tristate rather than be driven low, and the on-chip oscillator is switched off.

This mode allows switching a C161RI that is mounted to a board virtually off, so an emulator may control the board's circuitry, even though the original C161RI remains in its place. The original C161RI also may resume to control the board after a reset sequence with P0L.1 high. Please note that adapt mode overrides any other configuration via PORT0.

Default: Adapt Mode is off.

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

Special Operation Modes

Pins P0L.5 to P0L.2 (SMOD) select special operation modes of the C161RI during reset (see table below). Make sure to only select valid configurations in order to ensure proper operation of the C161RI.

Definition of Special Modes for Reset Configuration

P0.5-2 (P0L.5-2)	Special Mode	Notes
1 1 1 1	Normal Start	Default configuration. Begin of execution as defined via pin $\overline{EA}$.
1 1 1 0	*Reserved*	Do not select this configuration!
1 1 0 1	*Reserved*	Do not select this configuration!
1 1 0 0	*Reserved*	Do not select this configuration!
1 0 1 1	Bootstrap Loader	Load boot routine via ASC0.
1 0 1 0	*Reserved*	Do not select this configuration!
1 0 0 1	*Reserved*	Do not select this configuration!
1 0 0 0	*Reserved*	Do not select this configuration!
0 1 1 1	*Reserved*	Do not select this configuration!
0 1 1 0	*Reserved*	Do not select this configuration!
0 1 0 1	*Reserved*	Do not select this configuration!
0 1 0 0	*Reserved*	Do not select this configuration!
0 0 X X	*Reserved*	Do not select this configuration!

The on-chip Bootstrap Loader allows moving the start code into the internal RAM of the C161RI via the serial interface ASC0. The C161RI will remain in bootstrap loader mode until a hardware reset not selecting BSL mode or a software reset.

Default: The C161RI starts fetching code from location 00’0000$_H$, the bootstrap loader is off.

External Bus Type

Pins P0L.7 and P0L.6 (BUSTYP) select the external bus type during reset, if an external start is selected via pin $\overline{\text{EA}}$. This allows the configuration of the external bus interface of the C161RI even for the first code fetch after reset. The two bits are copied into bit field BTYP of register BUSCON0. P0L.7 controls the data bus width, while P0L.6 controls the address output (multiplexed or demultiplexed). This bit field may be changed via software after reset, if required.

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

PORT0 and PORT1 are automatically switched to the selected bus mode. In multiplexed bus modes PORT0 drives both the 16-bit intra-segment address and the output data, while PORT1 remains in high impedance state as long as no demultiplexed bus is selected via one of the BUSCON registers. In demultiplexed bus modes PORT1 drives the 16-bit intra-segment address, while PORT0 or P0L (according to the selected data bus width) drives the output data.
For a 16-bit data bus $\overline{\text{BHE}}$ is automatically enabled, for an 8-bit data bus $\overline{\text{BHE}}$ is disabled via bit BYTDIS in register SYSCON.

Default: 16-bit data bus with multiplexed addresses.

Note: If an internal start is selected via pin $\overline{\text{EA}}$, these two pins are disregarded and bit field BTYP of register BUSCON0 is cleared.

Write Configuration

Pin P0H.0 (WRC) selects the initial operation of the control pins $\overline{\text{WR}}$ and $\overline{\text{BHE}}$ during reset. When high, this pin selects the standard function, i.e. $\overline{\text{WR}}$ control and $\overline{\text{BHE}}$. When low, it selects the alternate configuration, i.e. $\overline{\text{WRH}}$ and $\overline{\text{WRL}}$. Thus even the first access after a reset can go to a memory controlled via $\overline{\text{WRH}}$ and $\overline{\text{WRL}}$. This bit is latched in register RP0H and its inverted value is copied into bit WRCFG in register SYSCON.

Default: Standard function ($\overline{\text{WR}}$ control and $\overline{\text{BHE}}$).

Chip Select Lines

Pins P0H.2 and P0H.1 (CSSEL) define the number of active chip select signals during reset. This allows the selection which pins of Port 6 drive external $\overline{\text{CS}}$ signals and which are used for general purpose IO. The two bits are latched in register RP0H.

Default: All 5 chip select lines active ($\overline{\text{CS4}}$ … $\overline{\text{CS0}}$).

CSSEL	Chip Select Lines		Note
1 1	Five:	$\overline{\text{CS4}}$ … $\overline{\text{CS0}}$	Default without pull-downs
1 0	None		Port 6 pins free for IO
0 1	Two:	$\overline{\text{CS1}}$ … $\overline{\text{CS0}}$	
0 0	Three:	$\overline{\text{CS2}}$ … $\overline{\text{CS0}}$	

Note: The selected number of $\overline{\text{CS}}$ signals cannot be changed via software after reset.

Segment Address Lines

Pins P0H.4 and P0H.3 (SALSEL) define the number of active segment address lines during reset. This allows the selection which pins of Port 4 drive address lines and which are used for general purpose IO. The two bits are latched in register RP0H. Depending on the system architecture the required address space is chosen and accessible right from the start, so the initialization routine can directly access all locations without prior programming. The required pins of Port 4 are automatically switched to address output mode.

SALSEL	Segment Address Lines		Directly accessible Address Space	
1 1	Two:	A17 … A16	256	KByte (Default without pull-downs)
1 0	Seven:	A22 … A16	8	MByte (Maximum)
0 1	None		64	KByte (Minimum)
0 0	Four:	A19 … A16	1	MByte

Even if not all segment address lines are enabled on Port 4, the C161RI internally uses its complete 24-bit addressing mechanism. This allows the restriction of the width of the effective address bus, while still deriving $\overline{\text{CS}}$ signals from the complete addresses.

Default: 2-bit segment address (A17 … A16) allowing access to 256 KByte.

Note: The selected number of segment address lines cannot be changed via software after reset.

Clock Generation Control

Pins P0H.7, P0H.6 and P0H.5 (CLKCFG) select the basic clock generation mode during reset. The oscillator clock either directly feeds the CPU and peripherals (direct drive) or it is divided by 2. These bits are latched in register RP0H.

C161RI Clock Generation Modes

P0.15-13 (P0H.7-5)	CPU Frequency $f_{CPU} = f_{OSC} \times F$	External Clock Input Range	Notes
1 1 1	*Reserved*		Default configuration
1 1 0	*Reserved*		
1 0 1	*Reserved*		
1 0 0	*Reserved*		
0 1 1	$f_{XTAL} \times 1$	1 to 20 MHz	Direct drive [1]
0 1 0	*Reserved*		
0 0 1	$f_{XTAL} / 2$	2 to 40 MHz	Prescaler operation
0 0 0	*Reserved*		

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

Default: Reserved combination. Select a valid configuration in any case!

Note: Watch the different requirements for frequency and duty cycle of the oscillator input clock for the possible selections.

19 Power Management

For an increasing number of microcontroller based systems it is an important objective to reduce the power consumption of the system as much as possible. A contradictory objective is, however, to reach a certain level of system performance. Besides optimization of design and technology a microcontroller's power consumption can generally be reduced by lowering its operating frequency and/or by reducing the circuitry that is clocked. The architecture of the C161RI provides three major means of reducing its power consumption (see figure below) under software control:

- Reduction of the CPU frequency for Slow Down operation (Flexible Clock Gen. Management)
- Selection of the active peripheral modules (Flexible Peripheral Management)
- Special operating modes to deactivate CPU, port drivers and control logic (Idle, Power Down)

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

Intermittend operation (i.e. alternating phases of high performance and power saving) is supported by the cyclic interrupt generation mode of the on-chip RTC (real time clock).

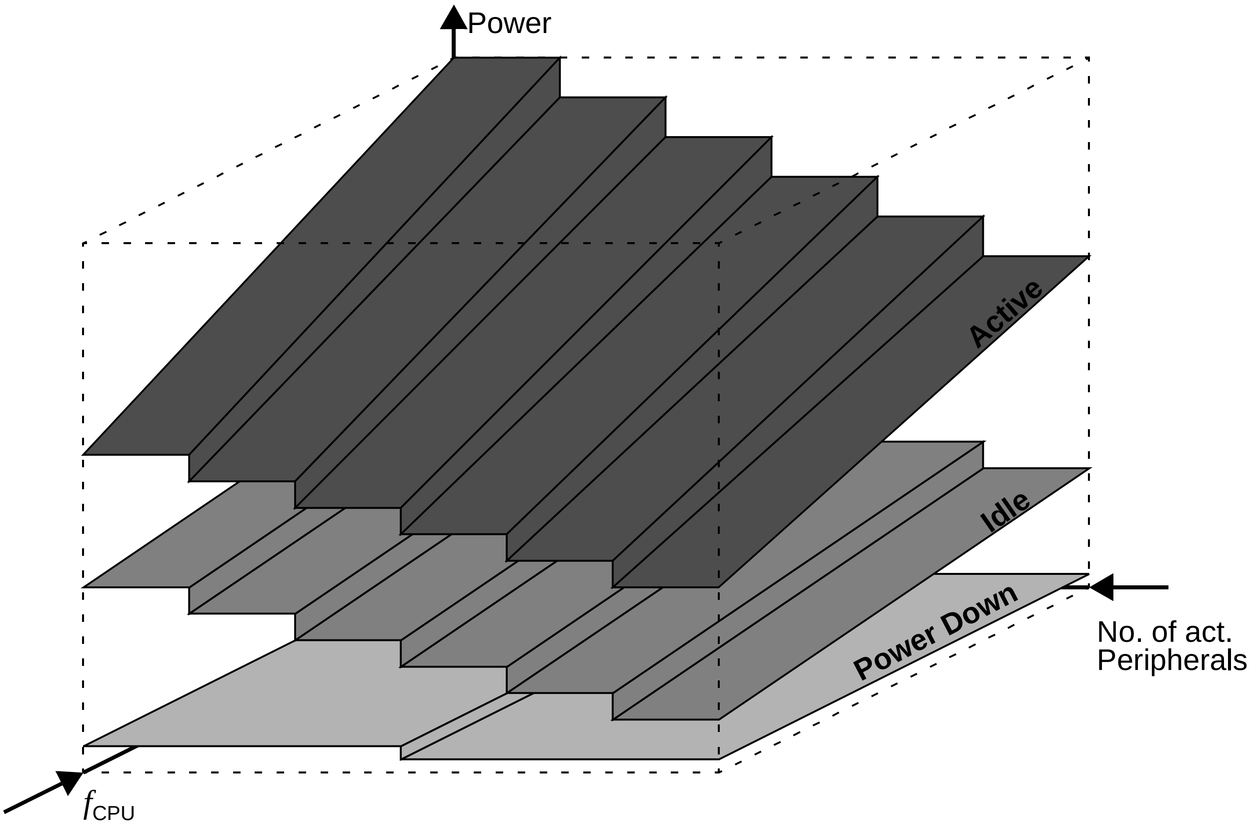


Figure 19-1
Power Reduction Possibilities

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

For the basic power reduction modes (Idle, Power Down) there are dedicated instructions, while special registers control clock generation (SYSCON2) and peripheral management (SYSCON3).

Two different general power reduction modes with different levels of power reduction have been implemented in the C161RI, which may be entered under software control.

In **Idle mode** the CPU is stopped, while the (enabled) peripherals continue their operation. Idle mode can be terminated by any reset or interrupt request.

In **Power Down mode** both the CPU and the peripherals are stopped. The real time clock and its selected oscillator may optionally be kept running. Power Down mode can only be terminated by a hardware reset.

Note: All external bus actions are completed before Idle or Power Down mode is entered. However, Idle or Power Down mode is **not** entered if $\overline{\text{READY}}$ is enabled, but has not been activated (driven low) during the last bus access.

In addition the power management selects the current CPU frequency and controls which peripherals are active.

During **Slow Down operation** the basic clock generation path is bypassed and the CPU clock is generated via the programmable Slow Down Divider (SDD) from the selected oscillator clock signal.

Peripheral Management disables and enables the on-chip peripheral modules independently, reducing the amount of clocked circuitry including the respective clock drivers.

19.1 Idle Mode

The power consumption of the C161RI microcontroller can be decreased by entering Idle mode. In this mode all enabled peripherals, **including** the watchdog timer, continue to operate normally, only the CPU operation is halted and the on-chip memory modules are disabled.

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

Idle mode is entered after the IDLE instruction has been executed and the instruction before the IDLE instruction has been completed. To prevent unintentional entry into Idle mode, the IDLE instruction has been implemented as a protected 32-bit instruction.

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

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

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

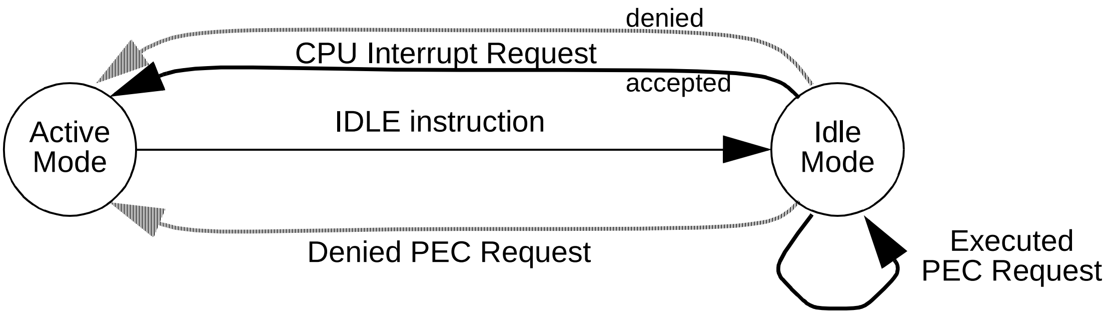


Figure 19-2
Transitions between Idle mode and Active Mode

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

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

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

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

19.2 Power Down Mode

To further reduce the power consumption the microcontroller can be switched to Power Down mode. Clocking of all internal blocks is stopped (RTC and selected oscillator optionally), the contents of the internal RAM, however, are preserved through the voltage supplied via the V_{DD} pins. The watchdog timer is stopped in Power Down mode. This mode can only be terminated by an external hardware reset, i.e. by asserting a low level on the $\overline{\text{RSTIN}}$ pin. This reset will initialize all SFRs and ports to their default state, but will not change the contents of the internal RAM.

There are two levels of protection against unintentionally entering Power Down mode. First, the PWRDN (Power Down) instruction which is used to enter this mode has been implemented as a protected 32-bit instruction. Second, this instruction is effective **only** if the $\overline{\text{NMI}}$ (Non Maskable Interrupt) pin is externally pulled low while the PWRDN instruction is executed. The microcontroller will enter Power Down mode after the PWRDN instruction has completed.

This feature can be used in conjunction with an external power failure signal which pulls the $\overline{\text{NMI}}$ pin low when a power failure is imminent. The microcontroller will enter the NMI trap routine which can save the internal state into RAM. After the internal state has been saved, the trap routine may then execute the PWRDN instruction. If the $\overline{\text{NMI}}$ pin is still low at this time, Power Down mode will be entered, otherwise program execution continues.

The initialization routine (executed upon reset) can check the reset identification flags in register WDTCON to determine whether the controller was initially switched on, or whether it was properly restarted from Power Down mode.

The realtime clock (RTC) can be kept running in Power Down mode in order to maintain a valid system time as long as the supply voltage is applied. This enables a system to determine the current time and the duration of the period while it was down (by comparing the current time with a timestamp stored when Power Down mode was entered). The supply current in this case remains well below 1 mA.

During power down the voltage at the V_{DD} pins can be lowered to 2.7 V while the RTC and its selected oscillator will still keep on running and the contents of the internal RAM will still be preserved.
When the RTC (and oscillator) is disabled the internal RAM is preserved down to a voltage of 2.5 V.

Note: When the RTC remains active in Power Down mode also the oscillator which generates the RTC clock signal will keep on running, of course.
If the supply voltage is reduced the specified maximum CPU clock frequency for this case must be respected.

The total power consumption in Power Down mode depends on the active circuitry (i.e. RTC on or off) and on the current that flows through the port drivers. To minimize the consumed current the RTC and/or all pin drivers can be disabled (pins switched to tristate) via a central control bitfield in register SYSCON2. If an application requires one or more port drivers to remain active even in Power Down mode also individual port drivers can be disabled simply by configuring them for input.

The bus interface pins can be separately disabled by releasing the external bus (disable all address windows by clearing the BUSACT bits) and switching the ports to input (if necessary). Of course the required software in this case must be executed from internal memory.

Status of Output Pins during Power Reduction Modes

During Idle mode the CPU clocks are turned off, while all peripherals continue their operation in the normal way. Therefore all ports pins, which are configured as general purpose output pins, output the last data value which was written to their port output latches. If the alternate output function of a port pin is used by a peripheral, the state of the pin is determined by the operation of the peripheral.

Port pins which are used for bus control functions go into that state which represents the inactive state of the respective function (e.g. $\overline{\text{WR}}$), or to a defined state which is based on the last bus access (e.g. $\overline{\text{BHE}}$). Port pins which are used as external address/data bus hold the address/data which was output during the last external memory access before entry into Idle mode under the following conditions:

P0H outputs the high byte of the last address if a multiplexed bus mode with 8-bit data bus is used, otherwise P0H is floating. P0L is always floating in Idle mode.

PORT1 outputs the lower 16 bits of the last address if a demultiplexed bus mode is used, otherwise the output pins of PORT1 represent the port latch data.

Port 4 outputs the segment address for the last access on those pins that were selected during reset, otherwise the output pins of Port 4 represent the port latch data.

During Power Down mode the oscillator (except for RTC operation) and the clocks to the CPU and to the peripherals are turned off. Like in Idle mode, all port pins which are configured as general purpose output pins output the last data value which was written to their port output latches.

When the alternate output function of a port pin is used by a peripheral the state of this pin is determined by the last action of the peripheral before the clocks were switched off.

Note: All pin drivers can be switched off by selecting the general port disable function prior to entering Power Down mode.

State of C161RI Output Pins during Idle and Power Down Mode

C161RI Output Pin(s)	**Idle Mode**		**Power Down Mode** (if pin drivers are generally enabled)	
	No external bus	**External bus enabled**	**No external bus**	**External bus enabled**
ALE	Low	Low	Low	Low
$\overline{\text{RD}}$, $\overline{\text{WR}}$	High	High	High	High
CLKOUT	Active	Active	High	High
$\overline{\text{RSTOUT}}$	1)	1)	1)	1)
P0L	Port Latch Data	Floating	Port Latch Data	Floating
P0H	Port Latch Data	A15 … A8 2)/Float	Port Latch Data	A15 … A8 2)/Float
PORT1	Port Latch Data	Last Address 3) / Port Latch Data	Port Latch Data	Last Address 3) / Port Latch Data
Port 4	Port Latch Data	Port Latch Data/ Last segment	Port Latch Data	Port Latch Data/ Last segment
$\overline{\text{BHE}}$	Port Latch Data	Last value	Port Latch Data	Last value
$\overline{\text{CSx}}$	Port Latch Data	Last value 4)	Port Latch Data	Last value 4)
Other Port Output Pins	Port Latch Data / Alternate Function	Port Latch Data / Alternate Function	Port Latch Data / Alternate Function	Port Latch Data / Alternate Function

Note:

1): High if EINIT was executed before entering Idle or Power Down mode, Low otherwise.

2): For multiplexed buses with 8-bit data bus.

3): For demultiplexed buses.

4): The CS signal that corresponds to the last address remains active (low), all other enabled CS signals remain inactive (high). By accessing an on-chip X-Periperal prior to entering a power save mode all external CS signals can be deactivated.

19.3 Slow Down Operation

A separate clock path can be selected for Slow Down operation bypassing the basic clock path used for standard operation. The programmable Slow Down Divider (SDD) divides the oscillator frequency by a factor of 1 … 32 which is specified via bitfield CLKREL in register SYSCON2. When bitfield CLKREL is written during SDD operation the reload counter will output one more clock pulse with the “old” frequency in order to resynchronize internally before generating the “new” frequency.

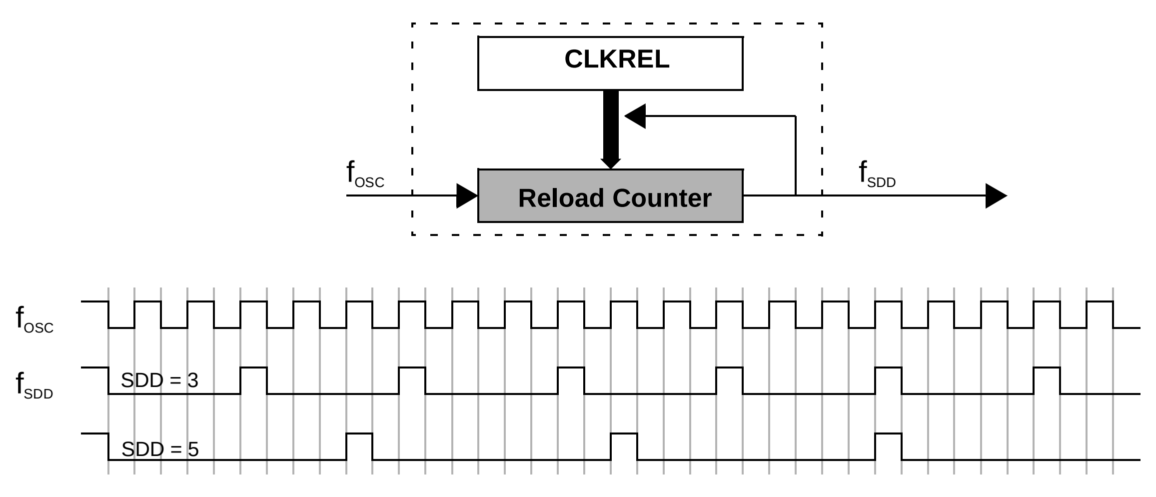


Figure 19-3
Slow Down Divider Operation

Using e.g. a 5 MHz input clock the on-chip logic may be run at a frequency down to 156.25 KHz without an external hardware change.

Note: During Slow Down operation the whole device (including bus interface and generation of signal CLKOUT) is clocked with the asymmetrical SDD clock (see figure above).

All these clock options are selected via bitfield CLKCON in register SYSCON2. A state machine controls the switching mechanism itself and ensures a continuous and glitch-free clock signal to the on-chip logic.

Switching to Slow Down operation affects frequency sensitive peripherals like serial interfaces, timers, PWM, etc. If these units are to be operated in Slow Down mode their precalers or reload values must be adapted. Please note that the reduced CPU frequency decreases e.g. timer resolution and increases the step width e.g. for baudrate generation. The oscillator frequency in such a case should be chosen to accommodate the required resolutions and/or baudrates.

SYSCON2 (F1D0_H / E8_H) **ESFR** **Reset Value: 00X0_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CLK LOCK	**CLKREL**					**CLKCON**		**SCS**	**RCS**	**PDCON**		**SYSRLS**			
r	rw					rw		rw	rw	rw		rw			

Bit	Function
SYSRLS	**SYSCON Release Function** (Unlock field) Must be written in a defined way in order to execute the unlock sequence. See separate description
PDCON	**Power Down Control** (during power down mode) 00: RTC = On, Ports = On (default after reset). 01: RTC = On, Ports = Off. 10: RTC = Off, Ports = On. 11: RTC = Off, Ports = Off.
RCS	**RTC Clock Source** (not affected by a reset) 0: Main oscillator. 1: Reserved.
SCS	**SDD Clock Source** (not affected by a reset) 0: Main oscillator. 1: Reserved.
CLKCON	**Clock State Control** 00: Running on configured basic frequency. 01: Running on slow down frequency. 10: Reserved. Do not use this combination. 11: Reserved. Do not use this combination.
CLKREL	**Reload Counter Value for Slowdown Divider** (SDD factor = CLKREL + 1)
CLKLOCK	**Clock Signal Status Bit** 0: Main oscillator is unstable. 1: Main oscillator is stable. Any running software requires an active clock signal and so never will see CLKLOCK = '0'.

Note: SYSCON2 (except for bitfield SYSRLS, of course) is write protected after the execution of EINIT unless it is released via the unlock sequence.

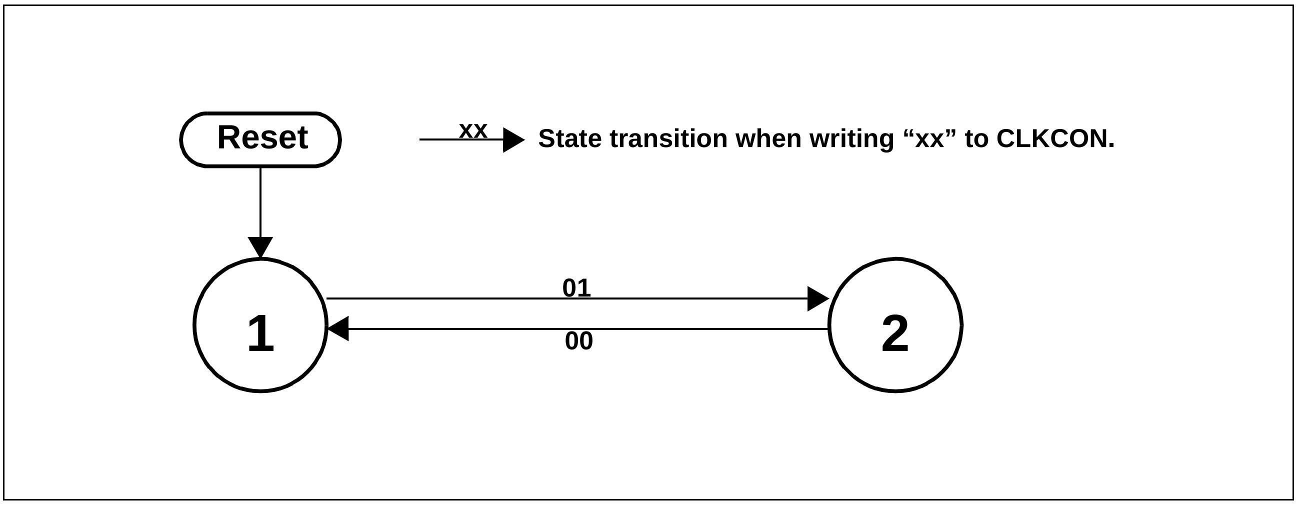


Figure 19-4
Clock Switching State Machine

Clock Switching State Description

State Number	f_{CPU} **Source**	**CLK CON**	**Note**
1	Basic	00	Standard operation on basic clock frequency.
2	SDD	01	SDD operation. Manual switch back to basic clock frequency.

19.4 Flexible Peripheral Management

The power consumed by the C161RI also depends on the amount of active logic. Peripheral management enables the system designer to deactivate those on-chip peripherals that are not required in a given system status (e.g. a certain interface mode or standby). All modules that remain active, however, will still deliver their usual performance. If all modules that are fed by the peripheral clock driver (PCD) are disabled and also the other functions fed by the PCD are not required, this clock driver itself may also be disabled to save additional power.

This flexibility is realized by distributing the CPU clock via several clock drivers which can be separately controlled, and may also be smaller.

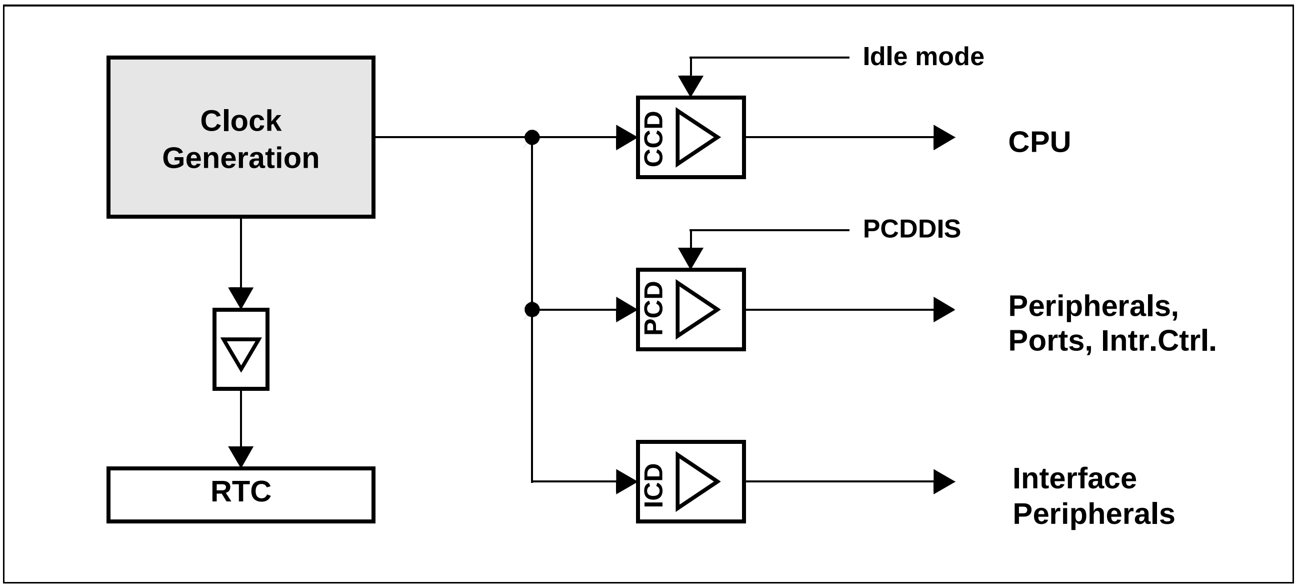


Figure 19-5
CPU Clock Distribution

Note: The Real Time Clock (RTC) is fed by a separate clock driver, so it can be kept running even in Power Down mode while still all the other circuitry is disconnected from the clock.

The registers of the generic peripherals can be accessed even while the respective module is disabled, as long as PCD is running (the registers of peripherals which are connected to ICD can be accessed even in this case, of course). The registers of X-peripherals cannot be accessed while the respective module is disabled by any means.

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

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

SYSCON3 (F1D4$_H$ / EA$_H$) **ESFR** **Reset Value: 0000$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PCD DIS	**-**	**-**	**-**	**I²C DIS**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**GPT DIS**	**SSC DIS**	**ASC0 DIS**	**ADC DIS**
rw			-	rw	-	-	-	-	-	-	-	rw	rw	rw	rw

Bit	**Function** (associated peripheral module)
ADCDIS	**Analog/Digital Converter**
ASC0DIS	**USART ASC0**
SSCDIS	**Synchronous Serial Channel SSC**
GPTDIS	**General Purpose Timer Blocks**
I2CDIS	**On-chip I²C-Bus Module**
PCDDIS	**Peripheral Clock Driver** (also X-Peripherals)

Note: The allocation of peripheral disable bits within register SYSCON3 is device specific and may be different in other derivatives than the C161RI.
SYSCON3 is write protected after the execution of EINIT unless it is released via the unlock sequence.

When disabling the peripheral clock driver (PCD), the following details should be respected:

- The clock signal for all connected peripherals is stopped. Make sure that all peripherals enter a safe state before disabling PCD.
- The output signal CLKOUT will remain high (‘1’).
- Interrupt requests will still be recognized even while PCD is disabled.
- No new output values are gated from the port output latches to the output port pins and no new input values are latched from the input port pins.

Security Mechanism

The power management control registers SYSCON2 and SYSCON3 control functions and modes which are critical for the C161RI's operation. For this reason they are locked (except for bitfield SYSRLS in register SYSCON2) after the execution of EINIT (like register SYSCON) so these vital system functions cannot be changed inadvertently e.g. by software errors. However, as these registers control the power management they need to be accessed during operation to select the appropriate mode. The system control software gets this access via a special unlock sequence which allows **one single** write access to either SYSCON2 or SYSCON3 when executed properly. This provides a maximum of security.

Note: Of course SYSCON2 and SYSCON3 may be read at any time without restrictions.

The unlock sequence is executed by writing defined values to bitfield SYSRLS using defined instructions (see table below). The instructions of the unlock sequence (including the intended write access) must be secured with an EXTR instruction (switch to ESFR space and lock interrupts).

Note: The unlock sequence provides no write access to register SYSCON.

SYSCON2/SYSCON3 Unlock Sequence

Step	SYSRLS	Instruction	Notes
---	0000_B 1)	---	Status before release sequence
1	1001_B	BFLDL, OR, ORB 2), XOR, XORB 2)	Read-Modify-Write access
2	0011_B	MOV, MOVB 2), MOVBS 2), MOVBZ 2)	Write access
3	0111_B	BSET, BMOV 2), BMOVN 2), BOR 2), BXOR 1)	Read-Modify-Write access, bit instruction
4	---	---	Single (read-modify-)write access to SYSCON2 or SYSCON3.
---	0000_B 3)	---	Status after release sequence

Note: 1) SYSRLS must be set to 0000_B before the first step, if any OR command is used.
2) Usually byte accesses should not be used for special function registers.
3) SYSRLS is cleared by hardware if unlock sequence and write access were successful. SYSRLS shows the last value written otherwise.

The code examples below show how an access to SYSCON2/SYSCON3 can be accomplished in an application.

```
ENTER_SLOWDOWN:                     ;Currently running on basic clock frequ.
EXTR   #4H                          ;Switch to ESFR space and lock sequence
BFLDL  SYSCON2,#0FH,#09H            ;Unlock sequence, step 1 (1001B)
MOV    SYSCON2,#0003H               ;Unlock sequence, step 2 (0011B)
BSET   SYSCON2.2                    ;Unlock sequence, step 3 (0111B)
                                    ;Single access to SYSCON2/SYSCON3
BFLDH  SYSCON2,#03H,#01H            ;CLKCON=01B --> SDD frequency
```

```
EXIT_SLOWDOWN:                      ;Currently running on SDD frequency
EXTR   #4H                          ;Switch to ESFR space and lock sequence
BFLDL  SYSCON2,#0FH,#09H            ;Unlock sequence, step 1 (1001B)
MOV    SYSCON2,#0003H               ;Unlock sequence, step 2 (0011B)
BSET   SYSCON2.2                    ;Unlock sequence, step 3 (0111B)
                                    ;Single access to SYSCON2/SYSCON3
BFLDH  SYSCON2,#03H,#00H            ;CLKCON=00B --> basic frequency
```

20 System Programming

To aid in software development, a number of features has been incorporated into the instruction set of the C161RI, including constructs for modularity, loops, and context switching. In many cases commonly used instruction sequences have been simplified while providing greater flexibility. The following programming features help to fully utilize this instruction set.

Instructions Provided as Subsets of Instructions

In many cases, instructions found in other microcontrollers are provided as subsets of more powerful instructions in the C161RI. This allows the same functionality to be provided while decreasing the hardware required and decreasing decode complexity. In order to aid assembly programming, these instructions, familiar from other microcontrollers, can be built in macros, thus providing the same names.

Directly Substitutable Instructions are instructions known from other microcontrollers that can be replaced by the following instructions of the C161RI:

Substituted Instruction		C161RI Instruction		Function
CLR	Rn	AND	Rn, #0_H	Clear register
CPLB	Bit	BMOVN	Bit, Bit	Complement bit
DEC	Rn	SUB	Rn, #1_H	Decrement register
INC	Rn	ADD	Rn, #1_H	Increment register
SWAPB	Rn	ROR	Rn, #8_H	Swap bytes within word

Modification of System Flags is performed using bit set or bit clear instructions (BSET, BCLR). All bit and word instructions can access the PSW register, so no instructions like CLEAR CARRY or ENABLE INTERRUPTS are required.

External Memory Data Access does not require special instructions to load data pointers or explicitly load and store external data. The C161RI provides a Von-Neumann memory architecture and its on-chip hardware automatically detects accesses to internal RAM, GPRs, and SFRs.

Multiplication and Division

Multiplication and division of words and double words is provided through multiple cycle instructions implementing a Booth algorithm. Each instruction implicitly uses the 32-bit register MD (MDL = lower 16 bits, MDH = upper 16 bits). The MDRIU flag (Multiply or Divide Register In Use) in register MDC is set whenever either half of this register is written to or when a multiply/divide instruction is started. It is cleared whenever the MDL register is read. Because an interrupt can be acknowledged before the contents of register MD are saved, this flag is required to alert interrupt routines, which require the use of the multiply/divide hardware, so they can preserve register MD. This register, however, only needs to be saved when an interrupt routine requires use of the MD register and a previous task has not saved the current result. This flag is easily tested by the Jump-on-Bit instructions.

Multiplication or division is simply performed by specifying the correct (signed or unsigned) version of the multiply or divide instruction. The result is then stored in register MD. The overflow flag (V) is set if the result from a multiply or divide instruction is greater than 16 bits. This flag can be used to determine whether both word halfs must be transferred from register MD. The high portion of register 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 only required if the current routine could have interrupted a previous routine which contained a MUL or DIV instruction. Register MDC is also saved because it is possible that a previous routine’s Multiply or Divide instruction was interrupted while in progress. In this case the information about how to restart the instruction is contained in this register. Register MDC must be cleared to be correctly initialized for a subsequent multiplication or division. The old MDC contents must be popped from the stack before the RETI instruction is executed.

For 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 register MD must be loaded. The result is also stored into register MD. The low portion (MDL) contains the integer result of the division, while the high portion (MDH) contains the remainder.

The following instruction sequence performs a 32 by 16-bit division:

```
MOV    MDH, R1                  ;Move dividend to MD register. Sets MDRIU
MOV    MDL, R2                  ;Move low portion to MD
DIV    R3                       ;Divide 32/16 signed, R3 holds divisor
JMPR   cc_V, ERROR              ;Test for divide overflow
MOV    R3, MDH                  ;Move remainder to R3
MOV    R4, MDL                  ;Move integer result to R4. Clears MDRIU
```

Whenever a multiply or divide instruction is interrupted while in progress, the address of the interrupted instruction is pushed onto the stack and the MULIP flag in the PSW of the interrupting routine is set. When the interrupt routine is exited with the RETI instruction, this bit is implicitly tested before the old PSW is popped from the stack. If MULIP='1' the multiply/divide instruction is re-read from the location popped from the stack (return address) and will be completed after the RETI instruction has been executed.

Note: The MULIP flag is part of the **context of the interrupted task**. When the interrupting routine does not return to the interrupted task (e.g. scheduler switches to another task) the MULIP flag must be set or cleared according to the context of the task that is switched to.

BCD Calculations

No direct support for BCD calculations is provided in the C161RI. BCD calculations are performed by converting BCD data to binary data, performing the desired calculations using standard data types, and converting the result back to BCD data. Due to the enhanced performance of division instructions binary data is quickly converted to BCD data through division by 10_D. Conversion from BCD data to binary data is enhanced by multiple bit shift instructions. This provides similar performance compared to instructions directly supporting BCD data types, while no additional hardware is required.

20.1 Stack Operations

The C161RI supports two types of stacks. The system stack is used implicitly by the controller and is located in the internal RAM. The user stack provides stack access to the user in either the internal or external memory. Both stack types grow from high memory addresses to low memory addresses.

Internal System Stack

A system stack is provided to store return vectors, segment pointers, and processor status for procedures and interrupt routines. A system register, SP, points to the top of the stack. This pointer is decremented when data is pushed onto the stack, and incremented when data is popped.

The internal system stack can also be used to temporarily store data or pass it between subroutines or tasks. Instructions are provided to push or pop registers on/from the system stack. However, in most cases the register banking scheme provides the best performance for passing data between multiple tasks.

Note: The system stack allows the storage of words only. Bytes must either be converted to words or the respective other byte must be disregarded.
Register SP can only be loaded with even byte addresses (The LSB of SP is always ’0’).

Detection of stack overflow/underflow is supported by two registers, STKOV (Stack Overflow Pointer) and STKUN (Stack Underflow Pointer). Specific system traps (Stack Overflow trap, Stack Underflow trap) will be entered whenever the SP reaches either boundary specified in these registers.

The contents of the stack pointer are compared to the contents of the overflow register, whenever the SP is DECREMENTED either by a CALL, PUSH or SUB instruction. An overflow trap will be entered, when the SP value is less than the value in the stack overflow register.

The contents of the stack pointer are compared to the contents of the underflow register, whenever the SP is INCREMENTED either by a RET, POP or ADD instruction. An underflow trap will be entered, when the SP value is greater than the value in the stack underflow register.

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

In many cases the user will place a software reset instruction (SRST) into the stack underflow and overflow trap service routines. This is an easy approach, which does not require special programming. However, this approach assumes that the defined internal stack is sufficient for the current software and that exceeding its upper or lower boundary represents a fatal error.

It is also possible to use the stack underflow and stack overflow traps to cache portions of a larger external stack. Only the portion of the system stack currently being used is placed into the internal memory, thus allowing a greater portion of the internal RAM to be used for program, data or register banking. This approach assumes no error but requires a set of control routines (see below).

Circular (virtual) Stack

This basic technique allows pushing until the overflow boundary of the internal stack is reached. At this point a portion of the stacked data must be saved into external memory to create space for further stack pushes. This is called “stack flushing”. When executing a number of return or pop instructions, the upper boundary (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 registers SP, STKOV and STKUN, to a defined physical stack area within the internal RAM 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 internal RAM that effectively is used for standard stack operations is defined via bitfield STKSZ in register SYSCON (see below).

<STKSZ>	Stack Size (Words)	Internal RAM Addresses (Words) of Physical Stack	Significant Bits of Stack Pointer 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 IRAM)	SP.9 … SP.0
1 0 1 $_B$	---	Reserved. Do not use this combination.	---
1 1 0 $_B$	---	Reserved. Do not use this combination.	---
1 1 1 $_B$	1024	00’FDFE$_H$ … 00’FX00$_H$ (Note: No circular stack) 00’FX00$_H$ represents the lower IRAM limit, i.e. 1 KB: 00’FA00$_H$, 2 KB: 00’F600$_H$, 3 KB: 00’F200$_H$	SP.11 … SP.0

The virtual stack addresses are transformed to physical stack addresses by concatenating the significant bits of the stack pointer register SP (see table) 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 below).

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.

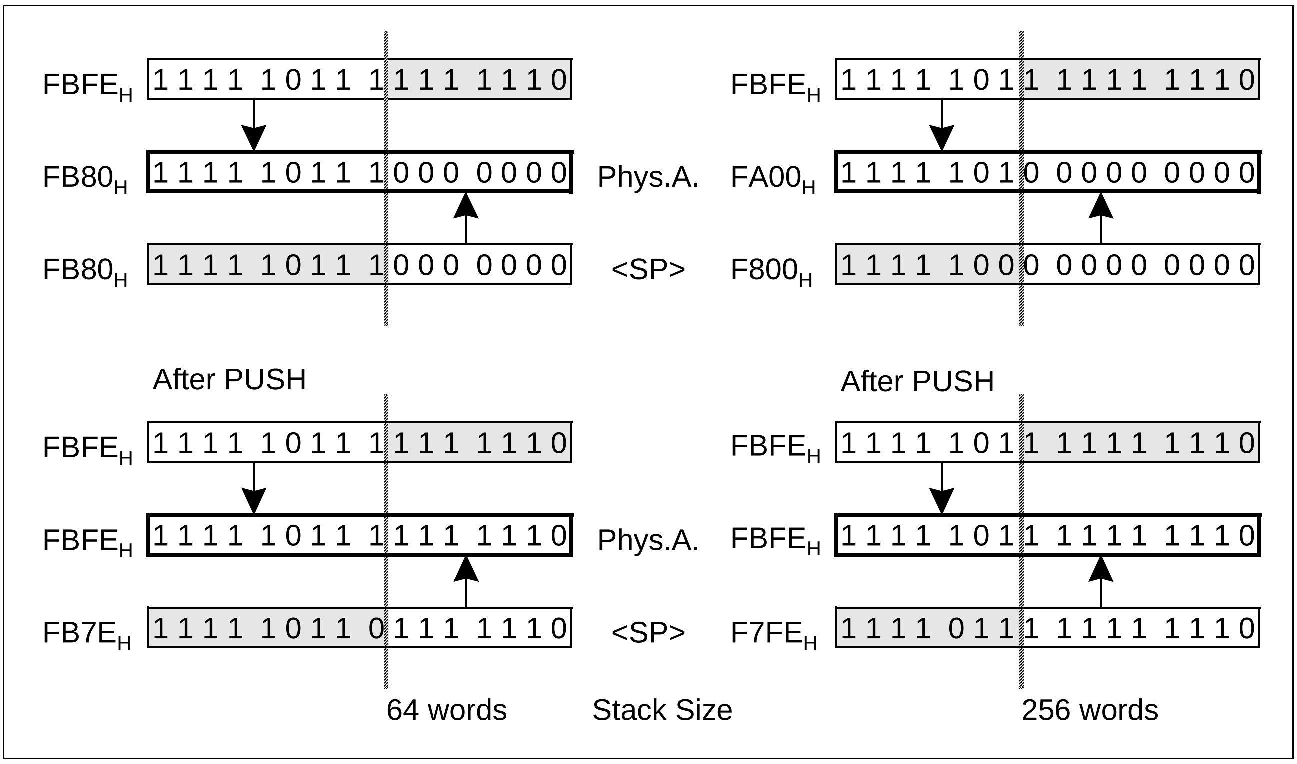


Figure 20-1
Physical Stack Address Generation

The following example demonstrates the circular stack mechanism which is also an effect of this virtual stack mapping: First, register R1 is pushed onto the lowest physical stack location according to the selected maximum stack size. With the following instruction, register R2 will be pushed onto the highest physical stack location although the SP is decremented by 2 as for the previous push operation.

```
MOV     SP, #0F802H              ;Set SP before last entry …
                                 ;… of physical stack of 256 words
…                                ;(SP)=F802H: Physical stack addr.=FA02H
PUSH    R1                       ;(SP)=F800H: Physical stack addr.=FA00H
PUSH    R2                       ;(SP)=F7FEH: Physical stack addr.=FBFEH
```

The effect of the address transformation is that the physical stack addresses wrap around from the end of the defined area to its beginning. When flushing and filling the internal stack, this circular stack mechanism only requires to move that portion of stack data which is really 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 internal RAM for system stack.
In the latter case the address transformation mechanism is deactivated.

When a boundary is reached, the stack underflow or overflow trap is entered, where the user moves a predetermined portion of the internal stack to or from the external stack. The amount of data transferred is determined by the average stack space required by routines and the frequency of calls, traps, interrupts and returns. In most cases this will be approximately one quarter to one tenth the size of the internal stack. 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:

- Specify the size of the physical system stack area within the internal RAM (bitfield STKSZ in register SYSCON).
- Define two pointers, which specify the upper and lower boundary of the external stack. These values are then tested in the stack underflow and overflow trap routines when moving data.
- Set the stack overflow pointer (STKOV) to the limit of the defined internal stack area plus six words (for the reserved space to store two interrupt entries).

The internal stack will now fill until the overflow pointer is reached. After entry into the overflow trap procedure, the top of the stack will be copied to the external memory. The internal pointers will then be modified to reflect the newly allocated space. After exiting from the trap procedure, the internal stack will wrap around to the top of the internal stack, and continue to grow until the new value of the stack overflow pointer is reached.

When the underflow pointer is reached while the stack is emptied the bottom of stack is reloaded from the external memory and the internal pointers are adjusted accordingly.

Linear Stack

The C161RI also offers a linear stack option (STKSZ = ‘111_B’), where the system stack may use the complete internal RAM area. This provides a large system stack without requiring procedures to handle data transfers for a circular stack. However, this method also leaves less RAM space for variables or code. The RAM area that may effectively 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 internal RAM 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 internal RAM range.

Note: Avoid stack accesses below the IRAM area (ESFR space and reserved area) and within address range 00’$FE00_H$ and 00’$FFFE_H$ (SFR space).
Otherwise unpredictable results will occur.

User Stacks

User stacks provide the ability to create task specific data stacks and to off-load data from the system stack. The user may push both bytes and words onto a user stack, but is responsible for using the appropriate instructions when popping data from the specific user stack. No hardware detection of overflow or underflow of a user stack is provided. The following addressing modes allow implementation of user stacks:

[– Rw], Rb or [– Rw], Rw: Pre-decrement Indirect Addressing.
Used to push one byte or word onto a user stack. This mode is only available for MOV instructions and can specify any GPR as the user stack pointer.

Rb, [Rw$_i$+] or Rw, [Rw$_i$+]: Post-increment Index Register Indirect Addressing.
Used to pop one byte or word from a user stack. This mode is available to most instructions, but only GPRs R0-R3 can be specified as the user stack pointer.

Rb, [Rw+] or Rw, [Rw+]: Post-increment Indirect Addressing.
Used to pop one byte or word from a user stack. This mode is only available for MOV instructions and can specify any GPR as the user stack pointer.

20.2 Register Banking

Register banking provides the user with an extremely fast method to switch user context. A single machine cycle instruction saves the old bank and enters a new register bank. Each register bank may assign up to 16 registers. Each register bank should be allocated during coding based on the needs of each task. Once the internal memory has been partitioned into a register bank space, internal stack space and a global internal memory area, each bank pointer is then assigned. Thus, upon entry into a new task, the appropriate bank pointer is used as the operand for the SCXT (switch context) instruction. Upon exit from a task a simple POP instruction to the context pointer (CP) restores the previous task’s register bank.

20.3 Procedure Call Entry and Exit

To support modular programming a procedure mechanism is provided to allow coding of frequently used portions of code into subroutines. The CALL and RET instructions store and restore the value of the instruction pointer (IP) on the system stack before and after a subroutine is executed.

Procedures may be called conditionally with instructions CALLA or CALLI, or be called unconditionally using instructions CALLR or CALLS.

Note: Any data pushed onto the system stack during execution of the subroutine must be popped before the RET instruction is executed.

Passing Parameters on the System Stack

Parameters may be passed via the system stack through PUSH instructions before the subroutine is called, and POP instructions during execution of the subroutine. Base plus offset indirect addressing also permits access to parameters without popping these parameters from the stack during execution of the subroutine. Indirect addressing provides a mechanism of accessing data referenced by data pointers, which are passed to the subroutine.

In addition, two instructions have been implemented to allow one parameter to be passed on the system stack without additional software overhead.

The PCALL (push and call) instruction first pushes the 'reg' operand and the IP contents onto the system stack and then passes control to the subroutine specified by the 'caddr' operand.

When exiting from the subroutine, the RETP (return and pop) instruction first pops the IP and then the 'reg' operand from the system stack and returns to the calling program.

Cross Segment Subroutine Calls

Calls to subroutines in different segments require the use of the CALLS (call inter-segment subroutine) instruction. This instruction preserves both the CSP (code segment pointer) and IP on the system stack.

Upon return from the subroutine, a RETS (return from inter-segment subroutine) instruction must be used to restore both the CSP and IP. This ensures that the next instruction after the CALLS instruction is fetched from the correct segment.

Note: It is possible to use CALLS within the same segment, but still two words of the stack are used to store both the IP and CSP.

Providing Local Registers for Subroutines

For subroutines which require local storage, the following methods are provided:

Alternate Bank of Registers: Upon entry into a subroutine, it is possible to specify a new set of local registers by executing the SCXT (switch context) instruction. This mechanism does not provide a method to recursively call a subroutine.

Saving and Restoring of Registers: To provide local registers, the contents of the registers which are required for use by the subroutine can be pushed onto the stack and the previous values be popped before returning to the calling routine. This is the most common technique used today and it does provide a mechanism to support recursive procedures. This method, however, requires two machine cycles per register stored on the system stack (one cycle to PUSH the register, and one to POP the register).

Use of the System Stack for Local Registers: It is possible to use the SP and CP to set up local subroutine register frames. This enables subroutines to dynamically allocate local variables as needed within two machine cycles. A local frame is allocated by simply subtracting the number of required local registers from the SP, and then moving the value of the new SP to the CP.

This operation is supported through the SCXT (switch context) instruction with the addressing mode 'reg, mem'. Using this instruction saves the old contents of the CP on the system stack and moves the value of the SP into CP (see example below). Each local register is then accessed as if it was a normal register. Upon exit from the subroutine, first the old CP must be restored by popping it from the stack and then the number of used local registers must be added to the SP to restore the allocated local space back to the system stack.

Note: The system stack is growing downwards, while the register bank is growing upwards.

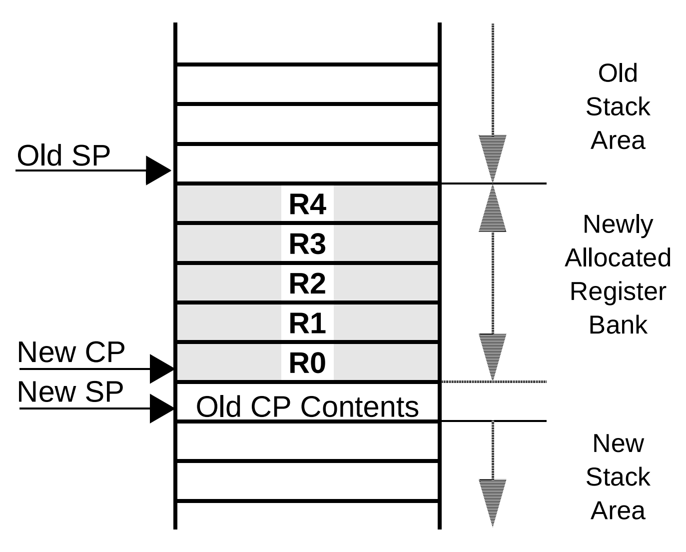


Figure 20-2
Local Registers

The software to provide the local register bank for the example above is very compact:

After entering the subroutine:

```
SUB    SP, #10D                  ;Free 5 words in the current system stack
SCXT   CP, SP                    ;Set the new register bank pointer
```

Before exiting the subroutine:

```
POP    CP                        ;Restore the old register bank
ADD    SP, #10D                  ;Release the 5 words…
                                 ;… of the current system stack
```

20.4 Table Searching

A number of features have been included to decrease the execution time required to search tables. First, branch delays are eliminated by the branch target cache after the first iteration of the loop. Second, in non-sequentially searched tables, the enhanced performance of the ALU allows more complicated hash algorithms to be processed to obtain better table distribution. For sequentially searched tables, the auto-increment indirect addressing mode and the E (end of table) flag stored in the PSW decrease the number of overhead instructions executed in the loop.

The two examples below illustrate searching ordered tables and non-ordered tables, respectively:

```
MOV    R0, #BASE                 ;Move table base into R0
LOOP:
CMP    R1, [R0+]                 ;Compare target to table entry
JMPR   cc_SGT, LOOP              ;Test whether target has not been found
```

Note: The last entry in the table must be greater than the largest possible target.

```
MOV    R0, #BASE                 ;Move table base into R0
LOOP:
CMP    R1, [R0+]                 ;Compare target to table entry
JMPR   cc_NET, LOOP              ;Test whether target is not found AND..
                                 ;..the end of table has not been reached.
```

Note: The last entry in the table must be equal to the lowest signed integer (8000_H).

20.5 Peripheral Control and Interface

All communication between peripherals and the CPU is performed either by PEC transfers to and from internal memory, or by explicitly addressing the SFRs associated with the specific peripherals. After resetting the C161RI all peripherals (except the watchdog timer) are disabled and initialized to default values. A desired configuration of a specific peripheral is programmed using MOV instructions of either constants or memory values to specific SFRs. Specific control flags may also be altered via bit instructions.

Once in operation, the peripheral operates autonomously until an end condition is reached at which time it requests a PEC transfer or requests CPU servicing through an interrupt routine. Information may also be polled from peripherals through read accesses to SFRs or bit operations including branch tests on specific control bits in SFRs. To ensure proper allocation of peripherals among multiple tasks, a portion of the internal memory has been made bit addressable to allow user semaphores. Instructions have also been provided to lock out tasks via software by setting or clearing user specific bits and conditionally branching based on these specific bits.

It is recommended that bit fields in control SFRs are updated using the BFLDH and BFLDL instructions or a MOV instruction to avoid undesired intermediate modes of operation which can occur, when BCLR/BSET or AND/OR instruction sequences are used.

20.6 Floating Point Support

All floating point operations are performed using software. Standard multiple precision instructions are used to perform calculations on data types that exceed the size of the ALU. Multiple bit rotate and logic instructions allow easy masking and extracting of portions of floating point numbers.

To decrease the time required to perform floating point operations, two hardware features have been implemented in the CPU core. First, the PRIOR instruction aids in normalizing floating point numbers by indicating the position of the first set bit in a GPR. This result can the be used to rotate the floating point result accordingly. The second feature aids in properly rounding the result of normalized floating point numbers through the overflow (V) flag in the PSW. This flag is set when a one is shifted out of the carry bit during shift right operations. The overflow flag and the carry flag are then used to round the floating point result based on the desired rounding algorithm.

20.7 Trap/Interrupt Entry and Exit

Interrupt routines are entered when a requesting interrupt has a priority higher than the current CPU priority level. Traps are entered regardless of the current CPU priority. When either a trap or interrupt routine is entered, the state of the machine is preserved on the system stack and a branch to the appropriate trap/interrupt vector is made.

All trap and interrupt routines require the use of the RETI (return from interrupt) instruction to exit from the called routine. This instruction restores the system state from the system stack and then branches back to the location where the trap or interrupt occurred.

20.8 Unseparable Instruction Sequences

The instructions of the C161RI are very efficient (most instructions execute in one machine cycle) and even the multiplication and division are interruptable in order to minimize the response latency to interrupt requests (internal and external). In many microcontroller applications this is vital.

Some special occasions, however, require certain code sequences (e.g. semaphore handling) to be uninterruptable to function properly. This can be provided by inhibiting interrupts during the respective code sequence by disabling and enabling them before and after the sequence. The necessary overhead may be reduced by means of the ATOMIC instruction which allows locking 1 … 4 instructions to an unseparable code sequence, during which the interrupt system (standard interrupts and PEC requests) **and Class A Traps** ($\overline{\text{NMI}}$, stack overflow/underflow) are disabled. A **Class B Trap** (illegal opcode, illegal bus access, etc.), however, will interrupt the atomic sequence, since it indicates a severe hardware problem. The interrupt inhibit caused by an ATOMIC instruction gets active immediately, i.e. no other instruction will enter the pipeline except the one that follows the ATOMIC instruction, and no interrupt request will be serviced in between. All instructions requiring multiple cycles or hold states are regarded as one instruction in this sense (e.g. MUL is one instruction). Any instruction type can be used within an unseparable code sequence.

```
ATOMIC #3                   ;The next 3 instr. are locked (No NOP requ.)
MOV    R0, #1234H           ;Instr. 1 (no other instr. enters pipeline!)
MOV    R1, #5678H           ;Instr. 2
MUL    R0, R1               ;Instr. 3: MUL regarded as one instruction
MOV    R2, MDL              ;This instruction is out of the scope …
                            ;… of the ATOMIC instruction sequence
```

20.9 Overriding the DPP Addressing Mechanism

The standard mechanism to access data locations uses one of the four data page pointers (DPPx), which selects a 16 KByte data page, and a 14-bit offset within this data page. The four DPPs allow immediate access to up to 64 KByte of data. In applications with big data arrays, especially in HLL applications using large memory models, this may require frequent reloading of the DPPs, even for single accesses.

The EXTP (extend page) instruction allows switching to an arbitrary data page for 1 … 4 instructions without having to change the current DPPs.

```
EXTP   R15, #1             ;The override page number is stored in R15
MOV    R0, [R14]           ;The (14-bit) page offset is stored in R14
MOV    R1, [R13]           ;This instruction uses the std. DPP scheme!
```

The EXTS (extend segment) instruction allows switching to a 64 KByte segment oriented data access scheme for 1 … 4 instructions without having to change the current DPPs. In this case all 16 bits of the operand address are used as segment offset, with the segment taken from the EXTS instruction. This greatly simplifies address calculation with continuous data like huge arrays in “C”.

```
EXTS   #15, #1             ;The override seg. is 15 (0F’0000H..0F’FFFFH)
MOV    R0, [R14]           ;The (16-bit) segment offset is stored in R14
MOV    R1, [R13]           ;This instruction uses the std. DPP scheme!
```

Note: Instructions EXTP and EXTS inhibit interrupts the same way as ATOMIC.

Short Addressing in the Extended SFR (ESFR) Space

The short addressing modes of the C161RI (REG or BITOFF) implicitly access the SFR space. The additional ESFR space would have to be accessed via long addressing modes (MEM or [Rw]). The EXTR (extend register) instruction redirects accesses in short addressing modes to the ESFR space for 1 … 4 instructions, so the additional registers can be accessed this way, too.

The EXTPR and EXTSR instructions combine the DPP override mechanism with the redirection to the ESFR space using a single instruction.

Note: Instructions EXTR, EXTPR and EXTSR inhibit interrupts the same way as ATOMIC.
The switching to the ESFR area and data page overriding is checked by the development tools or handled automatically.

Nested Locked Sequences

Each of the described extension instruction and the ATOMIC instruction starts an internal “extension counter” counting the effected instructions. When another extension or ATOMIC instruction is contained in the current locked sequence this counter is restarted with the value of the new instruction. This allows the construction of locked sequences longer than 4 instructions.

Note: • Interrupt latencies may be increased when using locked code sequences.
• PEC requests are not serviced during idle mode, if the IDLE instruction is part of a locked sequence.

20.10 Handling the Internal Code Memory

The Mask-ROM/OTP/Flash versions of the C161RI provide on-chip code memory that may store code as well as data. The lower 32 KByte of this code memory are referred to as the „internal ROM area“. Access to this internal ROM area is controlled during the reset configuration and via software. The ROM area may be mapped to segment 0, to segment 1 or the code memory may be disabled at all.

Note: The internal ROM area always occupies an address area of 32 KByte, even if the implemented mask ROM/OTP/Flash memory is smaller than that (e.g. 8 KByte).
Of course the total implemented memory may exceed 32 KBytes.

Code Memory Configuration during Reset

The control input pin $\overline{\text{EA}}$ (External Access) enables the user to define the address area from which the first instructions after reset are fetched. When $\overline{\text{EA}}$ is low (‘0’) during reset, the internal code memory is disabled and the first instructions are fetched from external memory. When $\overline{\text{EA}}$ is high (‘1’) during reset, the internal code memory is globally enabled and the first instructions are fetched from the internal memory.

Note: Be sure not to select internal memory access after reset on ROMless devices.

Mapping the Internal ROM Area

After reset the internal ROM area is mapped into segment 0, the “system segment” (00’0000$_H$ … 00’7FFF$_H$) as a default. This is necessary to allow the first instructions to be fetched from locations 00’0000$_H$ ff. The ROM area may be mapped to segment 1 (01’0000$_H$ … 01’7FFF$_H$) by setting bit ROMS1 in register SYSCON. The internal ROM area may now be accessed through the lower half of segment 1, while accesses to segment 0 will now be made to external memory. This adds flexibility to the system software. The interrupt/trap vector table, which uses locations 00’0000$_H$ through 00’01FF$_H$, is now part of the external memory and may therefore be modified, i.e. the system software may now change interrupt/trap handlers according to the current condition of the system. The internal code memory can still be used for fixed software routines like IO drivers, math libraries, application specific invariant routines, tables, etc. This combines the advantage of an integrated non-volatile memory with the advantage of a flexible, adaptable software system.

Enabling and Disabling the Internal Code Memory After Reset

If the internal code memory does not contain an appropriate startup code, the system may be booted from external memory, while the internal memory is enabled afterwards to provide access to library routines, tables, etc.

If the internal code memory only contains the startup code and/or test software, the system may be booted from internal memory, which may then be disabled, after the software has switched to executing from (e.g.) external memory, in order to free the address space occupied by the internal code memory, which is now unnecessary.

20.11 Pits, Traps and Mines

Although handling the internal code memory provides powerful means to enhance the overall performance and flexibility of a system, extreme care must be taken in order to avoid a system crash. Instruction memory is the most crucial resource for the C161RI and it must be made sure that it never runs out of it. The following precautions help to take advantage of the methods mentioned above without jeopardizing system security.

Internal code memory access after reset: When the first instructions are to be fetched from internal memory (EA=‘1’), the device must contain code memory, and this must contain a valid reset vector and valid code at its destination.

Mapping the internal ROM area to segment 1: Due to instruction pipelining, any new ROM mapping will at the earliest become valid for the second instruction after the instruction which has changed the ROM mapping. To enable accesses to the ROM area after mapping a branch to the newly selected ROM area (JMPS) and reloading of all data page pointers is required.
This also applies to re-mapping the internal ROM area to segment 0.

Enabling the internal code memory after reset: When enabling the internal code memory after having booted the system from external memory, note that the C161RI will then access the internal memory using the current segment offset, rather than accessing external memory.

Disabling the internal code memory after reset: When disabling the internal code memory after having booted the system from there, note that the C161RI will not access external memory before a jump to segment 0 (in this case) is executed.

General Rules

When mapping the code memory no instruction or data accesses should be made to the internal memory, otherwise unpredictable results may occur.

To avoid these problems, the instructions that configure the internal code memory should be executed from external memory or from the on-chip RAM.

Whenever the internal code memory is disabled, enabled or remapped the DPPs must be explicitly (re)loaded to enable correct data accesses to the internal and/or external memory.

21 The Register Set

This section summarizes all registers, which are implemented in the C161RI and explains the description format which is used in the chapters describing the function and layout of the SFRs. For easy reference the registers are ordered according to two different keys (except for GPRs):

- Ordered by address, to check which register a given address references,
- Ordered by register name, to find the location of a specific register.

21.1 Register Description Format

In the respective chapters the function and the layout of the SFRs is described in a specific format which provides a number of details about the described special function register. The example below shows how to interpret these details.

A word register looks like this:

***REG_NAME* (*A16*$_H$ / *A8*$_H$)** **E**/SFR **Reset Value: * * * *$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
res.	**res.**	**res.**	**res.**	**res.**	**write only**	**hw bit**	**read only**	**std bit**	**hw bit**	**bitfield**			**bitfield**		
-	-	-	-	-	w	rw	r	rw	rw	rw			rw		

Bit	Function
bit(field)name	***Explanation of bit(field)name*** *Description of the functions controlled by this bit(field).*

A byte register looks like this:

***REG_NAME* (*A16*$_H$ / *A8*$_H$)** **E**/SFR **Reset Value: - - * *$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								std bit	**hw bit**	**bitfield**				**bitfield**	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

Elements:

REG_NAME	Name of this register
A16 / A8	Long 16-bit address / Short 8-bit address
SFR/**ESFR**/**XReg**	Register space (SFR, ESFR or External/XBUS Register)
(* *) * *	Register contents after reset **0/1**: defined value, '**X**': undefined, '**U**': unchanged (undefined ('**X**') after power up)
hwbit	Bits that are set/cleared by hardware are marked with a shaded access box

21.2 CPU General Purpose Registers (GPRs)

The GPRs form the register bank that the CPU works with. This register bank may be located anywhere within the internal RAM via the Context Pointer (CP). Due to the addressing mechanism, GPR banks can only reside within the internal RAM. All GPRs are bit-addressable.

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

The first 8 GPRs (R7±R0) may also be accessed bytewise. Other than with SFRs, writing to a GPR byte does not affect the other byte of the respective GPR.
The respective halves of the byte-accessible registers receive special names:

Name	Physical Address	8-Bit Address	Description	Reset Value
RL0	(CP) + 0	$F0_H$	CPU General Purpose (Byte) Register RL0	UU_H
RH0	(CP) + 1	$F1_H$	CPU General Purpose (Byte) Register RH0	UU_H
RL1	(CP) + 2	$F2_H$	CPU General Purpose (Byte) Register RL1	UU_H
RH1	(CP) + 3	$F3_H$	CPU General Purpose (Byte) Register RH1	UU_H
RL2	(CP) + 4	$F4_H$	CPU General Purpose (Byte) Register RL2	UU_H
RH2	(CP) + 5	$F5_H$	CPU General Purpose (Byte) Register RH2	UU_H
RL3	(CP) + 6	$F6_H$	CPU General Purpose (Byte) Register RL3	UU_H
RH3	(CP) + 7	$F7_H$	CPU General Purpose (Byte) Register RH3	UU_H
RL4	(CP) + 8	$F8_H$	CPU General Purpose (Byte) Register RL4	UU_H
RH4	(CP) + 9	$F9_H$	CPU General Purpose (Byte) Register RH4	UU_H
RL5	(CP) + 10	FA_H	CPU General Purpose (Byte) Register RL5	UU_H
RH5	(CP) + 11	FB_H	CPU General Purpose (Byte) Register RH5	UU_H
RL6	(CP) + 12	FC_H	CPU General Purpose (Byte) Register RL6	UU_H
RH6	(CP) + 13	FD_H	CPU General Purpose (Byte) Register RH6	UU_H
RL7	(CP) + 14	FE_H	CPU General Purpose (Byte) Register RL7	UU_H
RH7	(CP) + 14	FF_H	CPU General Purpose (Byte) Register RH7	UU_H

21.3 Special Function Registers ordered by Name

The following table lists all SFRs which are implemented in the C161RI in alphabetical order.
Bit-addressable SFRs are marked with the letter “**b**” in column “Name”.
SFRs within the **Extended SFR-Space** (ESFRs) are marked with the letter “**E**” in column “Physical Address”. Registers within on-chip X-Peripherals are marked with the letter “**X**” in column “Physical Address”.

Name		Physical Address	8-Bit Address	Description	Reset Value
ADCIC	**b**	$FF98_H$	CC_H	A/D Converter End of Conversion Interrupt Control Register	0000_H
ADCON	**b**	$FFA0_H$	$D0_H$	A/D Converter Control Register	0000_H
ADDAT		$FEA0_H$	50_H	A/D Converter Result Register	0000_H
ADDRSEL1		$FE18_H$	$0C_H$	Address Select Register 1	0000_H
ADDRSEL2		$FE1A_H$	$0D_H$	Address Select Register 2	0000_H
ADDRSEL3		$FE1C_H$	$0E_H$	Address Select Register 3	0000_H
ADDRSEL4		$FE1E_H$	$0F_H$	Address Select Register 4	0000_H
ADEIC	**b**	$FF9A_H$	CD_H	A/D Converter Overrun Error Interrupt Control Register	0000_H
BUSCON0	**b**	$FF0C_H$	86_H	Bus Configuration Register 0	0000_H
BUSCON1	**b**	$FF14_H$	$8A_H$	Bus Configuration Register 1	0000_H
BUSCON2	**b**	$FF16_H$	$8B_H$	Bus Configuration Register 2	0000_H
BUSCON3	**b**	$FF18_H$	$8C_H$	Bus Configuration Register 3	0000_H
BUSCON4	**b**	$FF1A_H$	$8D_H$	Bus Configuration Register 4	0000_H
CAPREL		$FE4A_H$	25_H	GPT2 Capture/Reload Register	0000_H
CC8IC	**b**	$FF88_H$	$C4_H$	External Interrupt 0 Control Register	0000_H
CC9IC	**b**	$FF8A_H$	$C5_H$	External Interrupt 1 Control Register	0000_H
CC10IC	**b**	$FF8C_H$	$C6_H$	External Interrupt 2 Control Register	0000_H
CC11IC	**b**	$FF8E_H$	$C7_H$	External Interrupt 3 Control Register	0000_H
CC12IC	**b**	$FF90_H$	$C8_H$	External Interrupt 4 Control Register	0000_H
CC13IC	**b**	$FF92_H$	$C9_H$	External Interrupt 5 Control Register	0000_H
CC14IC	**b**	$FF94_H$	CA_H	External Interrupt 6 Control Register	0000_H
CC15IC	**b**	$FF96_H$	CB_H	External Interrupt 7 Control Register	0000_H
CP		$FE10_H$	08_H	CPU Context Pointer Register	$FC00_H$
CRIC	**b**	$FF6A_H$	$B5_H$	GPT2 CAPREL Interrupt Control Register	0000_H
CSP		$FE08_H$	04_H	CPU Code Segment Pointer Register (8 bits, not directly writeable)	0000_H

Name	Physical Address	8-Bit Address	Description	Reset Value
DP0L **b**	F100$_H$ **E**	80$_H$	P0L Direction Control Register	00$_H$
DP0H **b**	F102$_H$ **E**	81$_H$	P0H Direction Control Register	00$_H$
DP1L **b**	F104$_H$ **E**	82$_H$	P1L Direction Control Register	00$_H$
DP1H **b**	F106$_H$ **E**	83$_H$	P1H Direction Control Register	00$_H$
DP2 **b**	FFC2$_H$	E1$_H$	Port 2 Direction Control Register	0000$_H$
DP3 **b**	FFC6$_H$	E3$_H$	Port 3 Direction Control Register	0000$_H$
DP4 **b**	FFCA$_H$	E5$_H$	Port 4 Direction Control Register	00$_H$
DP6 **b**	FFCE$_H$	E7$_H$	Port 6 Direction Control Register	00$_H$
DPP0	FE00$_H$	00$_H$	CPU Data Page Pointer 0 Register (10 bits)	0000$_H$
DPP1	FE02$_H$	01$_H$	CPU Data Page Pointer 1 Register (10 bits)	0001$_H$
DPP2	FE04$_H$	02$_H$	CPU Data Page Pointer 2 Register (10 bits)	0002$_H$
DPP3	FE06$_H$	03$_H$	CPU Data Page Pointer 3 Register (10 bits)	0003$_H$
EXICON **b**	F1C0$_H$ **E**	E0$_H$	External Interrupt Control Register	0000$_H$
ICADR	ED06$_H$ **X**	---	I^{2}C Address Register	0XXX$_H$
ICCFG	ED00$_H$ **X**	---	I^{2}C Configuration Register	XX00$_H$
ICCON	ED02$_H$ **X**	---	I^{2}C Control Register	0000$_H$
ICRTB	ED08$_H$ **X**	---	I^{2}C Receive/Transmit Buffer	XX$_H$
ICST	ED04$_H$ **X**	---	I^{2}C Status Register	0000$_H$
IDCHIP	F07C$_H$ **E**	3E$_H$	Identifier	09XX$_H$
IDMANUF	F07E$_H$ **E**	3F$_H$	Identifier	1820$_H$
IDMEM	F07A$_H$ **E**	3D$_H$	Identifier	0000$_H$
IDPROG	F078$_H$ **E**	3C$_H$	Identifier	0000$_H$
ISNC **b**	F1DE$_H$ **E**	EF$_H$	Interrupt Subnode Control Register	0000$_H$
MDC **b**	FF0E$_H$	87$_H$	CPU Multiply Divide Control Register	0000$_H$
MDH	FE0C$_H$	06$_H$	CPU Multiply Divide Register – High Word	0000$_H$
MDL	FE0E$_H$	07$_H$	CPU Multiply Divide Register – Low Word	0000$_H$
ODP2 **b**	F1C2$_H$ **E**	E1$_H$	Port 2 Open Drain Control Register	0000$_H$
ODP3 **b**	F1C6$_H$ **E**	E3$_H$	Port 3 Open Drain Control Register	0000$_H$
ODP6 **b**	F1CE$_H$ **E**	E7$_H$	Port 6 Open Drain Control Register	00$_H$
ONES **b**	FF1E$_H$	8F$_H$	Constant Value 1’s Register (read only)	FFFF$_H$
P0L **b**	FF00$_H$	80$_H$	Port 0 Low Register (Lower half of PORT0)	00$_H$
P0H **b**	FF02$_H$	81$_H$	Port 0 High Register (Upper half of PORT0)	00$_H$
P1L **b**	FF04$_H$	82$_H$	Port 1 Low Register (Lower half of PORT1)	00$_H$

Name	Physical Address	8-Bit Address	Description	Reset Value
P1H **b**	FF06$_H$	83$_H$	Port 1 High Register (Upper half of PORT1)	00$_H$
P2 **b**	FFC0$_H$	E0$_H$	Port 2 Register	0000$_H$
P3 **b**	FFC4$_H$	E2$_H$	Port 3 Register	0000$_H$
P4 **b**	FFC8$_H$	E4$_H$	Port 4 Register (7 bits)	00$_H$
P5 **b**	FFA2$_H$	D1$_H$	Port 5 Register (read only)	XXXX$_H$
P5DIDIS **b**	FFA4$_H$	D2$_H$	Port 5 Digital Input Disable Register	0000$_H$
P6 **b**	FFCC$_H$	E6$_H$	Port 6 Register (8 bits)	00$_H$
PECC0	FEC0$_H$	60$_H$	PEC Channel 0 Control Register	0000$_H$
PECC1	FEC2$_H$	61$_H$	PEC Channel 1 Control Register	0000$_H$
PECC2	FEC4$_H$	62$_H$	PEC Channel 2 Control Register	0000$_H$
PECC3	FEC6$_H$	63$_H$	PEC Channel 3 Control Register	0000$_H$
PECC4	FEC8$_H$	64$_H$	PEC Channel 4 Control Register	0000$_H$
PECC5	FECA$_H$	65$_H$	PEC Channel 5 Control Register	0000$_H$
PECC6	FECC$_H$	66$_H$	PEC Channel 6 Control Register	0000$_H$
PECC7	FECE$_H$	67$_H$	PEC Channel 7 Control Register	0000$_H$
PICON	F1C4$_H$ **E**	E2$_H$	Port Input Threshold Control Register	0000$_H$
PSW **b**	FF10$_H$	88$_H$	CPU Program Status Word	0000$_H$
RP0H **b**	F108$_H$ **E**	84$_H$	System Startup Configuration Register (Rd. only)	XX$_H$
RTCH	F0D6$_H$ **E**	6B$_H$	RTC High Register	no
RTCL	F0D4$_H$ **E**	6A$_H$	RTC Low Register	no
S0BG	FEB4$_H$	5A$_H$	Serial Channel 0 Baud Rate Generator Reload Register	0000$_H$
S0CON **b**	FFB0$_H$	D8$_H$	Serial Channel 0 Control Register	0000$_H$
S0EIC **b**	FF70$_H$	B8$_H$	Serial Channel 0 Error Interrupt Control Register	0000$_H$
S0RBUF	FEB2$_H$	59$_H$	Serial Channel 0 Receive Buffer Register (read only)	XXXX$_H$
S0RIC **b**	FF6E$_H$	B7$_H$	Serial Channel 0 Receive Interrupt Control Register	0000$_H$
S0TBIC **b**	F19C$_H$ **E**	CE$_H$	Serial Channel 0 Transmit Buffer Interrupt Control Register	0000$_H$
S0TBUF	FEB0$_H$	58$_H$	Serial Channel 0 Transmit Buffer Register	0000$_H$
S0TIC **b**	FF6C$_H$	B6$_H$	Serial Channel 0 Transmit Interrupt Control Register	0000$_H$
SP	FE12$_H$	09$_H$	CPU System Stack Pointer Register	FC00$_H$

Name	Physical Address	8-Bit Address	Description	Reset Value
SSCBR	F0B4$_H$ **E**	5A$_H$	SSC Baudrate Register	0000$_H$
SSCCON **b**	FFB2$_H$	D9$_H$	SSC Control Register	0000$_H$
SSCEIC **b**	FF76$_H$	BB$_H$	SSC Error Interrupt Control Register	0000$_H$
SSCRB	F0B2$_H$ **E**	59$_H$	SSC Receive Buffer (read only)	XXXX$_H$
SSCRIC **b**	FF74$_H$	BA$_H$	SSC Receive Interrupt Control Register	0000$_H$
SSCTB	F0B0$_H$ **E**	58$_H$	SSC Transmit Buffer (write only)	0000$_H$
SSCTIC **b**	FF72$_H$	B9$_H$	SSC Transmit Interrupt Control Register	0000$_H$
STKOV	FE14$_H$	0A$_H$	CPU Stack Overflow Pointer Register	FA00$_H$
STKUN	FE16$_H$	0B$_H$	CPU Stack Underflow Pointer Register	FC00$_H$
SYSCON **b**	FF12$_H$	89$_H$	CPU System Configuration Register	0XX0$_H$ 1)
SYSCON2 **b**	F1D0$_H$	E8$_H$	CPU System Configuration Register 2	0000$_H$
SYSCON3 **b**	F1D4$_H$	EA$_H$	CPU System Configuration Register 3	0000$_H$
T14	F0D2$_H$ **E**	69$_H$	RTC Timer 14 Register	no
T14REL	F0D0$_H$ **E**	68$_H$	RTC Timer 14 Reload Register	no
T2	FE40$_H$	20$_H$	GPT1 Timer 2 Register	0000$_H$
T2CON **b**	FF40$_H$	A0$_H$	GPT1 Timer 2 Control Register	0000$_H$
T2IC **b**	FF60$_H$	B0$_H$	GPT1 Timer 2 Interrupt Control Register	0000$_H$
T3	FE42$_H$	21$_H$	GPT1 Timer 3 Register	0000$_H$
T3CON **b**	FF42$_H$	A1$_H$	GPT1 Timer 3 Control Register	0000$_H$
T3IC **b**	FF62$_H$	B1$_H$	GPT1 Timer 3 Interrupt Control Register	0000$_H$
T4	FE44$_H$	22$_H$	GPT1 Timer 4 Register	0000$_H$
T4CON **b**	FF44$_H$	A2$_H$	GPT1 Timer 4 Control Register	0000$_H$
T4IC **b**	FF64$_H$	B2$_H$	GPT1 Timer 4 Interrupt Control Register	0000$_H$
T5	FE46$_H$	23$_H$	GPT2 Timer 5 Register	0000$_H$
T5CON **b**	FF46$_H$	A3$_H$	GPT2 Timer 5 Control Register	0000$_H$
T5IC **b**	FF66$_H$	B3$_H$	GPT2 Timer 5 Interrupt Control Register	0000$_H$
T6	FE48$_H$	24$_H$	GPT2 Timer 6 Register	0000$_H$
T6CON **b**	FF48$_H$	A4$_H$	GPT2 Timer 6 Control Register	0000$_H$
T6IC **b**	FF68$_H$	B4$_H$	GPT2 Timer 6 Interrupt Control Register	0000$_H$
TFR **b**	FFAC$_H$	D6$_H$	Trap Flag Register	0000$_H$
WDT	FEAE$_H$	57$_H$	Watchdog Timer Register (read only)	0000$_H$
WDTCON **b**	FFAE$_H$	D7$_H$	Watchdog Timer Control Register	00XX$_H$ 2)
XP0IC **b**	F186$_H$ **E**	C3$_H$	I^{2}C Data Interrupt Control Register	0000$_H$

Name	Physical Address	8-Bit Address	Description	Reset Value
XP1IC **b**	F18E$_H$ **E**	C7$_H$	I^{2}C Protocol Interrupt Control Register	0000$_H$
XP2IC **b**	F196$_H$ **E**	CB$_H$	X-Peripheral 2 Interrupt Control Register	0000$_H$
XP3IC **b**	F19E$_H$ **E**	CF$_H$	RTC Interrupt Control Register	0000$_H$
ZEROS **b**	FF1C$_H$	8E$_H$	Constant Value 0’s Register (read only)	0000$_H$

1) The system configuration is selected during reset.

2) The reset value depends on the indicated reset source.

21.4 Registers ordered by Address

The following table lists all SFRs which are implemented in the C161RI ordered by their physical address. **Bit-addressable** SFRs are marked with the letter “**b**” in column “Name”.
SFRs within the **Extended SFR-Space** (ESFRs) are marked with the letter “**E**” in column “Physical Address”. Registers within on-chip X-Peripherals are marked with the letter “**X**” in column “Physical Address”.

Name	Physical Address	8-Bit Address	Description	Reset Value
ICCFG	ED00$_H$ **X**	---	I^{2}C Configuration Register	XX$_H$
ICCON	ED02$_H$ **X**	---	I^{2}C Control Register	UUUU$_H$
ICST	ED04$_H$ **X**	---	I^{2}C Status Register	UUUU$_H$
ICADR	ED06$_H$ **X**	---	I^{2}C Address Register	XX01$_H$
ICRTB	ED08$_H$ **X**	---	I^{2}C Receive/Transmit Buffer	UFUU$_H$
IDPROG	F078$_H$ **E**	3C$_H$	Identifier	0000$_H$
IDMEM	F07A$_H$ **E**	3D$_H$	Identifier	0000$_H$
IDCHIP	F07C$_H$ **E**	3E$_H$	Identifier	09XX$_H$
IDMANUF	F07E$_H$ **E**	3F$_H$	Identifier	1820$_H$
SSCTB	F0B0$_H$ **E**	58$_H$	SSC Transmit Buffer (write only)	0000$_H$
SSCRB	F0B2$_H$ **E**	59$_H$	SSC Receive Buffer (read only)	XXXX$_H$
SSCBR	F0B4$_H$ **E**	5A$_H$	SSC Baudrate Register	0000$_H$
T14REL	F0D0$_H$ **E**	68$_H$	RTC Timer 14 Reload Register	XXXX$_H$
T14	F0D2$_H$ **E**	69$_H$	RTC Timer 14 Register	XXXX$_H$
RTCL	F0D4$_H$ **E**	6A$_H$	RTC Low Register	XXXX$_H$
RTCH	F0D6$_H$ **E**	6B$_H$	RTC High Register	XXXX$_H$
DP0L b	F100$_H$ **E**	80$_H$	P0L Direction Control Register	00$_H$
DP0H b	F102$_H$ **E**	81$_H$	P0H Direction Control Register	00$_H$
DP1L b	F104$_H$ **E**	82$_H$	P1L Direction Control Register	00$_H$
DP1H b	F106$_H$ **E**	83$_H$	P1H Direction Control Register	00$_H$
RP0H b	F108$_H$ **E**	84$_H$	System Startup Configuration Register (Rd. only)	XX$_H$
XP0IC b	F186$_H$ **E**	C3$_H$	X-Peripheral 0 Interrupt Control Register	0000$_H$
XP1IC b	F18E$_H$ **E**	C7$_H$	I^{2}C Protocol Interrupt Control Register	0000$_H$
XP2IC b	F196$_H$ **E**	CB$_H$	X-Peripheral 2 Interrupt Control Register	0000$_H$
S0TBIC b	F19C$_H$ **E**	CE$_H$	Serial Channel 0 Transmit Buffer Interrupt Control Register	0000$_H$
XP3IC b	F19E$_H$ **E**	CF$_H$	X-Peripheral 3 Interrupt Control Register	0000$_H$

Name	Physical Address	8-Bit Address	Description	Reset Value
EXICON b	F1C0$_H$ **E**	E0$_H$	External Interrupt Control Register	0000$_H$
ODP2 b	F1C2$_H$ **E**	E1$_H$	Port 2 Open Drain Control Register	0000$_H$
PICON	F1C4$_H$ **E**	E2$_H$	Port Input Threshold Control Register	0000$_H$
ODP3 b	F1C6$_H$ **E**	E3$_H$	Port 3 Open Drain Control Register	0000$_H$
ODP6 b	F1CE$_H$ **E**	E7$_H$	Port 6 Open Drain Control Register	00$_H$
SYSCON2 b	F1D0$_H$ **E**	E8$_H$	CPU System Configuration Register 2	0000$_H$
SYSCON3 b	F1D4$_H$ **E**	EA$_H$	CPU System Configuration Register 3	0000$_H$
ISNC b	F1DE$_H$ **E**	EF$_H$	Interrupt Subnode Control Register	0000$_H$
DPP0	FE00$_H$	00$_H$	CPU Data Page Pointer 0 Register (10 bits)	0000$_H$
DPP1	FE02$_H$	01$_H$	CPU Data Page Pointer 1 Register (10 bits)	0001$_H$
DPP2	FE04$_H$	02$_H$	CPU Data Page Pointer 2 Register (10 bits)	0002$_H$
DPP3	FE06$_H$	03$_H$	CPU Data Page Pointer 3 Register (10 bits)	0003$_H$
CSP	FE08$_H$	04$_H$	CPU Code Segment Pointer Register (8 bits, not directly writeable)	0000$_H$
MDH	FE0C$_H$	06$_H$	CPU Multiply Divide Register – High Word	0000$_H$
MDL	FE0E$_H$	07$_H$	CPU Multiply Divide Register – Low Word	0000$_H$
CP	FE10$_H$	08$_H$	CPU Context Pointer Register	FC00$_H$
SP	FE12$_H$	09$_H$	CPU System Stack Pointer Register	FC00$_H$
STKOV	FE14$_H$	0A$_H$	CPU Stack Overflow Pointer Register	FA00$_H$
STKUN	FE16$_H$	0B$_H$	CPU Stack Underflow Pointer Register	FC00$_H$
ADDRSEL1	FE18$_H$	0C$_H$	Address Select Register 1	0000$_H$
ADDRSEL2	FE1A$_H$	0D$_H$	Address Select Register 2	0000$_H$
ADDRSEL3	FE1C$_H$	0E$_H$	Address Select Register 3	0000$_H$
ADDRSEL4	FE1E$_H$	0F$_H$	Address Select Register 4	0000$_H$
T2	FE40$_H$	20$_H$	GPT1 Timer 2 Register	0000$_H$
T3	FE42$_H$	21$_H$	GPT1 Timer 3 Register	0000$_H$
T4	FE44$_H$	22$_H$	GPT1 Timer 4 Register	0000$_H$
T5	FE46$_H$	23$_H$	GPT2 Timer 5 Register	0000$_H$
T6	FE48$_H$	24$_H$	GPT2 Timer 6 Register	0000$_H$
CAPREL	FE4A$_H$	25$_H$	GPT2 Capture/Reload Register	0000$_H$
ADDAT	FEA0$_H$	50$_H$	A/D Converter Result Register	0000$_H$
WDT	FEAE$_H$	57$_H$	Watchdog Timer Register (read only)	0000$_H$
S0TBUF	FEB0$_H$	58$_H$	Serial Channel 0 Transmit Buffer Register	0000$_H$

Name	Physical Address	8-Bit Address	Description	Reset Value
S0RBUF	FEB2$_H$	59$_H$	Serial Channel 0 Receive Buffer Register (read only)	XXXX$_H$
S0BG	FEB4$_H$	5A$_H$	Serial Channel 0 Baud Rate Generator Reload Register	0000$_H$
PECC0	FEC0$_H$	60$_H$	PEC Channel 0 Control Register	0000$_H$
PECC1	FEC2$_H$	61$_H$	PEC Channel 1 Control Register	0000$_H$
PECC2	FEC4$_H$	62$_H$	PEC Channel 2 Control Register	0000$_H$
PECC3	FEC6$_H$	63$_H$	PEC Channel 3 Control Register	0000$_H$
PECC4	FEC8$_H$	64$_H$	PEC Channel 4 Control Register	0000$_H$
PECC5	FECA$_H$	65$_H$	PEC Channel 5 Control Register	0000$_H$
PECC6	FECC$_H$	66$_H$	PEC Channel 6 Control Register	0000$_H$
PECC7	FECE$_H$	67$_H$	PEC Channel 7 Control Register	0000$_H$
P0L b	FF00$_H$	80$_H$	Port 0 Low Register (Lower half of PORT0)	00$_H$
P0H b	FF02$_H$	81$_H$	Port 0 High Register (Upper half of PORT0)	00$_H$
P1L b	FF04$_H$	82$_H$	Port 1 Low Register (Lower half of PORT1)	00$_H$
P1H b	FF06$_H$	83$_H$	Port 1 High Register (Upper half of PORT1)	00$_H$
BUSCON0 b	FF0C$_H$	86$_H$	Bus Configuration Register 0	0000$_H$
MDC b	FF0E$_H$	87$_H$	CPU Multiply Divide Control Register	0000$_H$
PSW b	FF10$_H$	88$_H$	CPU Program Status Word	0000$_H$
SYSCON b	FF12$_H$	89$_H$	CPU System Configuration Register	0XX0$_H$1)
BUSCON1 b	FF14$_H$	8A$_H$	Bus Configuration Register 1	0000$_H$
BUSCON2 b	FF16$_H$	8B$_H$	Bus Configuration Register 2	0000$_H$
BUSCON3 b	FF18$_H$	8C$_H$	Bus Configuration Register 3	0000$_H$
BUSCON4 b	FF1A$_H$	8D$_H$	Bus Configuration Register 4	0000$_H$
ZEROS b	FF1C$_H$	8E$_H$	Constant Value 0's Register (read only)	0000$_H$
ONES b	FF1E$_H$	8F$_H$	Constant Value 1's Register (read only)	FFFF$_H$
T2CON b	FF40$_H$	A0$_H$	GPT1 Timer 2 Control Register	0000$_H$
T3CON b	FF42$_H$	A1$_H$	GPT1 Timer 3 Control Register	0000$_H$
T4CON b	FF44$_H$	A2$_H$	GPT1 Timer 4 Control Register	0000$_H$
T5CON b	FF46$_H$	A3$_H$	GPT2 Timer 5 Control Register	0000$_H$
T6CON b	FF48$_H$	A4$_H$	GPT2 Timer 6 Control Register	0000$_H$
T2IC b	FF60$_H$	B0$_H$	GPT1 Timer 2 Interrupt Control Register	0000$_H$
T3IC b	FF62$_H$	B1$_H$	GPT1 Timer 3 Interrupt Control Register	0000$_H$

Name		Physical Address	8-Bit Address	Description	Reset Value
T4IC	**b**	FF64$_H$	B2$_H$	GPT1 Timer 4 Interrupt Control Register	0000$_H$
T5IC	**b**	FF66$_H$	B3$_H$	GPT2 Timer 5 Interrupt Control Register	0000$_H$
T6IC	**b**	FF68$_H$	B4$_H$	GPT2 Timer 6 Interrupt Control Register	0000$_H$
CRIC	**b**	FF6A$_H$	B5$_H$	GPT2 CAPREL Interrupt Control Register	0000$_H$
S0TIC	**b**	FF6C$_H$	B6$_H$	Serial Channel 0 Transmit Interrupt Control Register	0000$_H$
S0RIC	**b**	FF6E$_H$	B7$_H$	Serial Channel 0 Receive Interrupt Control Register	0000$_H$
S0EIC	**b**	FF70$_H$	B8$_H$	Serial Channel 0 Error Interrupt Control Register	0000$_H$
SSCTIC	**b**	FF72$_H$	B9$_H$	SSC Transmit Interrupt Control Register	0000$_H$
SSCRIC	**b**	FF74$_H$	BA$_H$	SSC Receive Interrupt Control Register	0000$_H$
SSCEIC	**b**	FF76$_H$	BB$_H$	SSC Error Interrupt Control Register	0000$_H$
CC8IC	**b**	FF88$_H$	C4$_H$	CAPCOM Register 8 Interrupt Control Register	0000$_H$
CC9IC	**b**	FF8A$_H$	C5$_H$	CAPCOM Register 9 Interrupt Control Register	0000$_H$
CC10IC	**b**	FF8C$_H$	C6$_H$	CAPCOM Register 10 Interrupt Control Register	0000$_H$
CC11IC	**b**	FF8E$_H$	C7$_H$	CAPCOM Register 11 Interrupt Control Register	0000$_H$
CC12IC	**b**	FF90$_H$	C8$_H$	External Interrupt 4 Control Register	0000$_H$
CC13IC	**b**	FF92$_H$	C9$_H$	External Interrupt 5 Control Register	0000$_H$
CC14IC	**b**	FF94$_H$	CA$_H$	External Interrupt 6 Control Register	0000$_H$
CC15IC	**b**	FF96$_H$	CB$_H$	External Interrupt 7 Control Register	0000$_H$
ADCIC	**b**	FF98$_H$	CC$_H$	A/D Converter End of Conversion Interrupt Control Register	0000$_H$
ADEIC	**b**	FF9A$_H$	CD$_H$	A/D Converter Overrun Error Interrupt Control Register	0000$_H$
ADCON	**b**	FFA0$_H$	D0$_H$	A/D Converter Control Register	0000$_H$
P5	**b**	FFA2$_H$	D1$_H$	Port 5 Register (read only)	XXXX$_H$
P5DIDIS	**b**	FFA4$_H$	D2$_H$	Port 5 Digital Input Disable Register	0000$_H$
TFR	**b**	FFAC$_H$	D6$_H$	Trap Flag Register	0000$_H$
WDTCON	**b**	FFAE$_H$	D7$_H$	Watchdog Timer Control Register	00XX$_H$ 2)
S0CON	**b**	FFB0$_H$	D8$_H$	Serial Channel 0 Control Register	0000$_H$
SSCCON	**b**	FFB2$_H$	D9$_H$	SSC Control Register	0000$_H$
P2	**b**	FFC0$_H$	E0$_H$	Port 2 Register	0000$_H$
DP2	**b**	FFC2$_H$	E1$_H$	Port 2 Direction Control Register	0000$_H$
P3	**b**	FFC4$_H$	E2$_H$	Port 3 Register	0000$_H$

Name		Physical Address	8-Bit Address	Description	Reset Value
DP3	**b**	FFC6$_H$	E3$_H$	Port 3 Direction Control Register	0000$_H$
P4	**b**	FFC8$_H$	E4$_H$	Port 4 Register (8 bits)	00$_H$
DP4	**b**	FFCA$_H$	E5$_H$	Port 4 Direction Control Register	00$_H$
P6	**b**	FFCC$_H$	E6$_H$	Port 6 Register (8 bits)	00$_H$
DP6	**b**	FFCE$_H$	E7$_H$	Port 6 Direction Control Register	00$_H$

1) The system configuration is selected during reset.

2) The reset value depends on the indicated reset source.

21.5 Special Notes

PEC Pointer Registers

The source and destination pointers for the peripheral event controller are mapped to a special area within the internal RAM. Pointers that are not occupied by the PEC may therefore be used like normal RAM. During Power Down mode or any warm reset the PEC pointers are preserved.

The PEC and its registers are described in chapter “Interrupt and Trap Functions”.

GPR Access in the ESFR Area

The locations 00’F000H … 00’F01EH within the ESFR area are reserved and allow to access the current register bank via short register addressing modes. The GPRs are mirrored to the ESFR area which allows access to the current register bank even after switching register spaces (see example below).

```
MOV    R5, DP3                 ;GPR access via SFR area
EXTR   #1
MOV    R5, ODP3                ;GPR access via ESFR area
```

Writing Bytes to SFRs

All special function registers may be accessed wordwise or bytewise (some of them even bitwise). Reading bytes from word SFRs is a non-critical operation. However, when writing bytes to word SFRs the complementary byte of the respective SFR is cleared with the write operation.

22 Instruction Set Summary

This chapter briefly summarizes the C161RI's instructions ordered by instruction classes. This provides a basic understanding of the C161RI's instruction set, the power and versatility of the instructions and their general usage.

A detailed description of each single instruction, including its operand data type, condition flag settings, addressing modes, length (number of bytes) and object code format is provided in the **"Instruction Set Manual"** for the C166 Family. This manual also provides tables ordering the instructions according to various criteria, to allow quick references.

Summary of Instruction Classes

Grouping the various instruction into classes aids in identifying similar instructions (e.g. SHR, ROR) and variations of certain instructions (e.g. ADD, ADDB). This provides an easy access to the possibilities and the power of the instructions of the C161RI.

Note: The used mnemonics refer to the detailed description.

Arithmetic Instructions

• Addition of two words or bytes:	ADD	ADDB
• Addition with Carry of two words or bytes:	ADDC	ADDCB
• Subtraction of two words or bytes:	SUB	SUBB
• Subtraction with Carry of two words or bytes:	SUBC	SUBCB
• 16∗16 bit signed or unsigned multiplication:	MUL	MULU
• 16/16 bit signed or unsigned division:	DIV	DIVU
• 32/16 bit signed or unsigned division:	DIVL	DIVLU
• 1's complement of a word or byte:	CPL	CPLB
• 2's complement (negation) of a word or byte:	NEG	NEGB

Logical Instructions

• Bitwise ANDing of two words or bytes:	AND	ANDB
• Bitwise ORing of two words or bytes:	OR	ORB
• Bitwise XORing of two words or bytes:	XOR	XORB

Compare and Loop Control Instructions

• Comparison of two words or bytes:	CMP	CMPB
• Comparison of two words with post-increment by either 1 or 2:	CMPI1	CMPI2
• Comparison of two words with post-decrement by either 1 or 2:	CMPD1	CMPD2

Boolean Bit Manipulation Instructions

- Manipulation of a maskable bit field in either the high or the low byte of a word: BFLDH BFLDL
- Setting a single bit (to ‘1’): BSET
- Clearing a single bit (to ‘0’): BCLR
- Movement of a single bit: BMOV
- Movement of a negated bit: BMOVN
- ANDing of two bits: BAND
- ORing of two bits: BOR
- XORing of two bits: BXOR
- Comparison of two bits: BCMP

Shift and Rotate Instructions

- Shifting right of a word: SHR
- Shifting left of a word: SHL
- Rotating right of a word: ROR
- Rotating left of a word: ROL
- Arithmetic shifting right of a word (sign bit shifting): ASHR

Prioritize Instruction

- Determination of the number of shift cycles required to normalize a word operand (floating point support): PRIOR

Data Movement Instructions

- Standard data movement of a word or byte: MOV MOVB
- Data movement of a byte to a word location with either sign or zero byte extension: MOVBS MOVBZ

Note: The data movement instructions can be used with a big number of different addressing modes including indirect addressing and automatic pointer in-/decrementing.

System Stack Instructions

- Pushing of a word onto the system stack: PUSH
- Popping of a word from the system stack: POP
- Saving of a word on the system stack, and then updating the old word with a new value (provided for register bank switching): SCXT

Jump Instructions

• Conditional jumping to an either absolutely, indirectly, or relatively addressed target instruction within the current code segment:	JMPA	JMPI	JMPR
• Unconditional jumping to an absolutely addressed target instruction within any code segment:	JMPS		
• Conditional jumping to a relatively addressed target instruction within the current code segment depending on the state of a selectable bit:	JB	JNB	
• Conditional jumping to a relatively addressed target instruction within the current code segment depending on the state of a selectable bit with a post-inversion of the tested bit in case of jump taken (semaphore support):	JBC	JNBS	

Call Instructions

• Conditional calling of an either absolutely or indirectly addressed subroutine within the current code segment:	CALLA	CALLI
• Unconditional calling of a relatively addressed subroutine within the current code segment:	CALLR	
• Unconditional calling of an absolutely addressed subroutine within any code segment:	CALLS	
• Unconditional calling of an absolutely addressed subroutine within the current code segment plus an additional pushing of a selectable register onto the system stack:	PCALL	
• Unconditional branching to the interrupt or trap vector jump table in code segment 0:	TRAP	

Return Instructions

• Returning from a subroutine within the current code segment:	RET
• Returning from a subroutine within any code segment:	RETS
• Returning from a subroutine within the current code segment plus an additional popping of a selectable register from the system stack:	RETP
• Returning from an interrupt service routine:	RETI

System Control Instructions

- Resetting the C161RI via software: SRST
- Entering the Idle mode: IDLE
- Entering the Power Down mode: PWRDN
- Servicing the Watchdog Timer: SRVWDT
- Disabling the Watchdog Timer: DISWDT
- Signifying the end of the initialization routine (pulls pin $\overline{\text{RSTOUT}}$ high, and disables the effect of any later execution of a DISWDT instruction): EINIT

Miscellaneous

- Null operation which requires 2 bytes of storage and the minimum time for execution: NOP
- Definition of an unseparable instruction sequence: ATOMIC
- Switch ‘reg’, ‘bitoff’ and ‘bitaddr’ addressing modes to the Extended SFR space: EXTR
- Override the DPP addressing scheme using a specific data page instead of the DPPs, and optionally switch to ESFR space: EXTP EXTPR
- Override the DPP addressing scheme using a specific segment instead of the DPPs, and optionally switch to ESFR space: EXTS EXTSR

Note: The ATOMIC and EXT* instructions provide support for uninterruptable code sequences e.g. for semaphore operations. They also support data addressing beyond the limits of the current DPPs (except ATOMIC), which is advantageous for bigger memory models in high level languages. Refer to chapter “System Programming” for examples.

Protected Instructions

Some instructions of the C161RI which are critical for the functionality of the controller are implemented as so-called Protected Instructions. These protected instructions use the maximum instruction format of 32 bits for decoding, while the regular instructions only use a part of it (e.g. the lower 8 bits) with the other bits providing additional information like involved registers. Decoding all 32 bits of a protected doubleword instruction increases the security in cases of data distortion during instruction fetching. Critical operations like a software reset are therefore only executed if the complete instruction is decoded without an error. This enhances the safety and reliability of a microcontroller system.

23 Device Specification

The device specification describes the electrical parameters of the device. It lists DC characteristics like input, output or supply voltages or currents, and AC characteristics like timing characteristics and requirements.

Other than the architecture, the instruction set or the basic functions of the C161RI core and its peripherals, these DC and AC characteristics are subject to changes due to device improvements or specific derivatives of the standard device.

Therefore these characteristics are not contained in this manual, but rather provided in a separate Data Sheet, which can be updated more frequently.

Please refer to the current version of the Data Sheet of the respective device for all electrical parameters.

Note: In any case the specific characteristics of a device should be verified, before a new design is started. This ensures that the used information is up to date.

The figures below show the pin diagrams of the C161RI. They shows the location of the different supply and IO pins. A detailed description of all the pins is also found in the Data Sheet.

Note: Not all alternate functions shown in the figure below are supported by all derivatives.
Please refer to the corresponding descriptions in the data sheets.

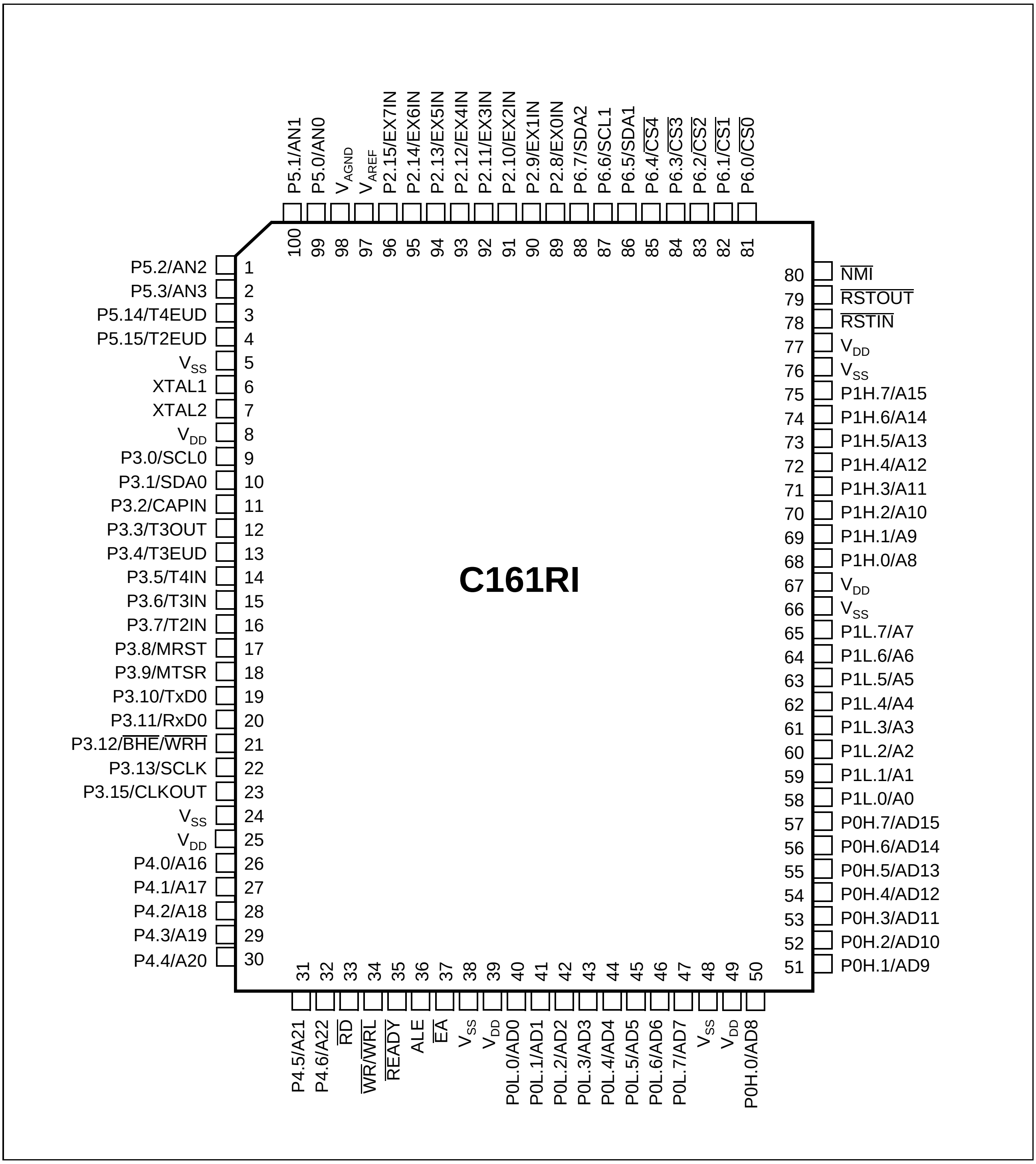


Figure 23-1
Pin Description for C161RI, P-MQFP-100 Package

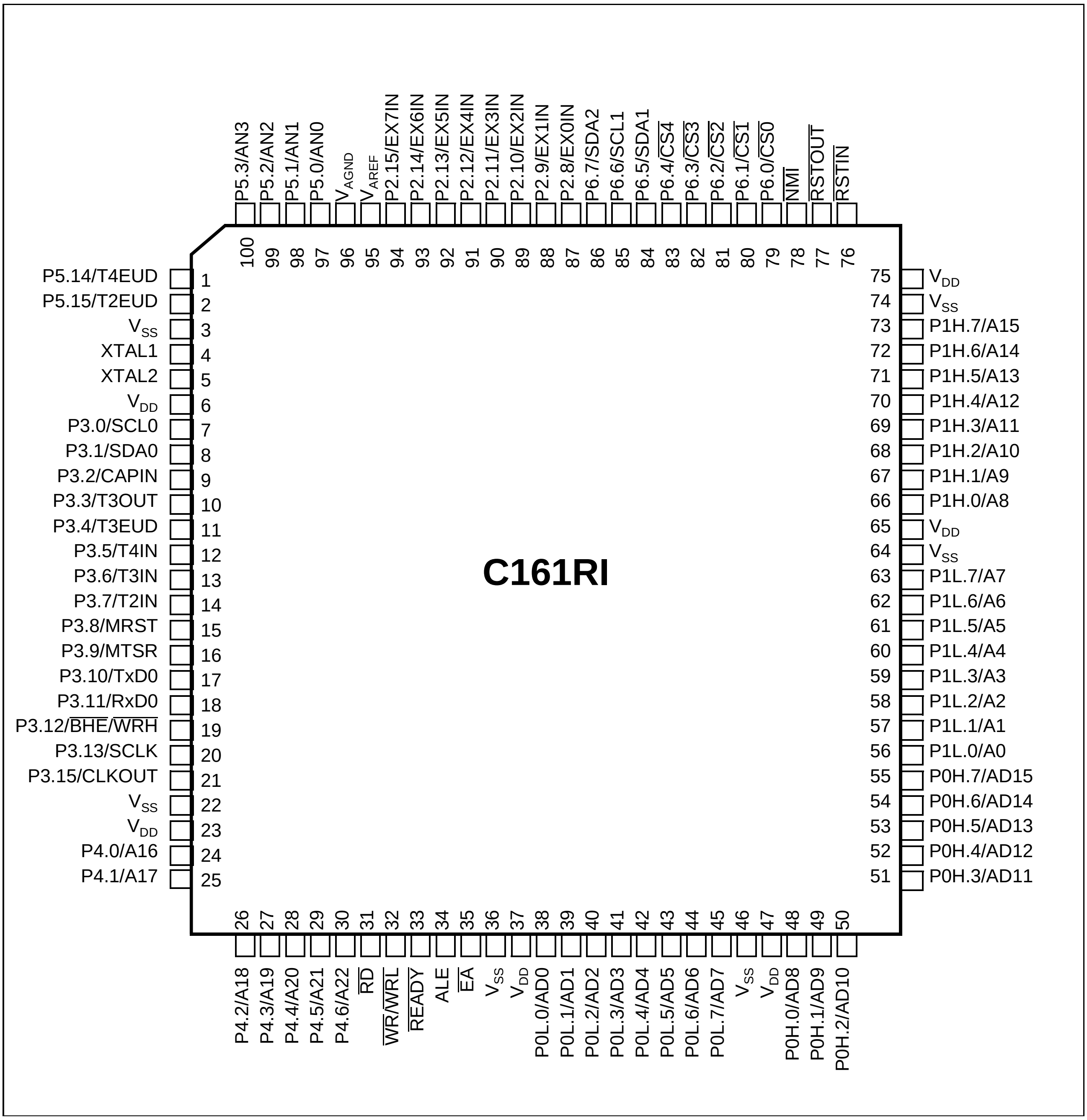


Figure 23-2
Pin Description for C161RI, P-TQFP-100 Package

24 Keyword Index

This section lists a number of keywords which refer to specific details of the C161RI in terms of its architecture, its functional units or functions. This helps to quickly find the answer to specific questions about the C161RI.