Data Sheet, DS 2, April 2001

C161U

Embedded C166 with USB, USART and SSC

Version 1.3

Wired Communications

Never stop thinking.

Edition 2001-04-5

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

Attention please!

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

Terms of delivery and rights to technical change reserved.

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

Infineon Technologies is an approved CECC manufacturer.

Information

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

Warnings

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

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

Data Sheet, DS 2, April 2001

C161U

Embedded C166 with USB,USART and SSC

Version 1.3

Wired Communications

Never stop thinking.

C161U

Revision History: **2001-04-5** DS 2

Previous Version:	Preliminary Data Sheet 02.2000, this is the first non-preliminary version.[1)]

Page	Subjects (major changes since last revision)
89	Correction of the PEC Control Register: The correct channel numbers are PEC channels 0 and 2. The name of the corresponding register is PECXC2 and not PECXC1.
101	Correction of EPEC_CTRL_REGx description
333	Correction of USB description: AS2 was removed
459	Correction of the Parameter t14.
	Improved formatting (text, figures, tables)

[1)] All previous distributed versions are *preliminary*. They have been replaced by this version.

For questions on technology, delivery and prices please contact the Infineon Technologies Offices in Germany or the Infineon Technologies Companies and Representatives worldwide: see our webpage at http://www.infineon.com

Table of Contents

Page

Table of Contents

Page

Table of Contents Page

Table of Contents Page

Table of Contents

Page

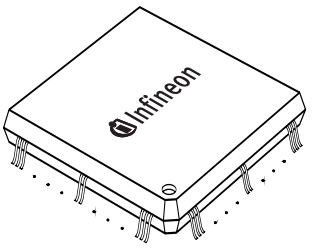
Table of Contents **Page**

Embedded C166 with USB, USART and SSC
C161U

Version 1.3 **CMOS**

1 Overview

C161U is a new low cost member of the Infineon Communication Controller family using low power CMOS technology. The device combines the successful Infineon C166 16-bit full-static core with a full-speed Universal Serial Bus (USB) interface and 3-Kbyte of Dual-Port on-chip RAM.



P-TQFP-100

C161U adresses Intelligent NT or SOHO PBX designs, offering up to 18 MIPS along with legacy peripherals such as USART, SCI and Timers.

The USB device core has a built-in DMA, which provides maximum flexibility and performance. Off-loading the CPU in such a manner allows the user to implement value add software features enabling product differentiation.

C161U provides:

- On-Chip full-static C166 Core supporting a 16- or 8-bit C16x Family System running up to 36 MHz
- 12 Mbit/s Full-Speed USB Interface Vers. 1.1 compliant
- Support for Audio, Data and Communication Device Classes
- USART Interface with AutoBaud Support (1.2 kbit/s - 230.4 kbit/s)
- AT-Command sensitive AutoBaud Detection

1.1 Key Features

C161U is a new low-cost member of the Infineon Communication Controller family. The device has the following features:

- C166 Static Core with Peripherals including:
  – Full-static core up to 18 MIPS (@ 36 MHz)

Type	Package
C161U	P-TQFP-100

  - Peripheral Event Controller (PEC) for 8 independent DMA channels
  - 16 Dynamically Programmable Priority-Level Interrupt System
  - Two External Interrupts
  - Up to 56 SW-configurative Input/Output (I/O) Ports, some with Interrupt Capabilities
  - 8-bit or 16-bit External Data Bus
  - Multiplexed or Demultiplexed Address/Data Bus
  - Up to 2-Mbyte Linear Address Space for Code and Data
  - Four Programmable Chip-Select Lines with Wait-State Generator Logic
  - On-Chip 3,072-Byte Dual-Port SRAM for user applications
  - On-Chip 1,024-Byte Special Function Register Area
  - On-Chip PLL with Output Clock Signal
  - Five Multimode General Purpose Timers
  - On-Chip Programmable Watchdog Timer
  - Glueless Interface to EPROM, Flash EPROM and SRAM
  - Low-Power Management Supporting Idle-, Power-Down- and Sleep-Mode and additional CPU clock slow-down mode with mode control for each peripheral
  - USART interface with Auto Baud Rate detection up to 230,400 kbit/s
  - USART Baud Rate generation in asynchronous mode up to 2.25 MBaud @ 36 MHz
  - USART Baud Rate generation in synchronous mode up to 4.5 MBaud @ 36 MHz
  - USART standard Baud Rates generation with very small deviation (230.4 kBaud < 0.01%, 460.8 kBaud < 0.15 %, 691.2 kBaud < 0.04 %, 921.6 kBaud < 0.15 %) @ 36 MHz
  - High speed Serial Synchronous Channel Interface (SSC) with ALIS-3.0 and AC97 compatibility up to 18 MBaud in SSC Master Mode and up to 9 MBaud in SSC Slave Mode @ 36 MHz
- USB Interface including:
  - USB Specification 1.1 Compliant
  - 12 Mbit/s Full-Speed Mode
  - 7 SW-configurable Endpoints, in addition to the bi-directional Control Endpoint 0
  - 3 Configurations with 3 alternate settings and 4 interfaces supported
  - Each non-Control Endpoint can be either Isochronous, Bulk or Interrupt
  - Autonomous DMA Transfer by on-chip DMA for 8 USB endpoints
- On-Chip PLL for CPU and USB clock generation
- External crystal and direct driven input clock of 8 MHz when USB interface is used. In applications without USB, the input clock frequency can vary between 4 and 20 MHz dependent on the CPU target clock frequency.
- Single and variable crystal clock input frequency (using USB 8 MHz only)
- Bootstrap Loader support via USART interface
- On-Chip Debug Support (OCDS)
- JTAG Boundary Scan Test Support according to IEEE 1149.1
- 3.3 V Single Supply Voltage, 5 V (TTL-) Tolerant I/Os
- -40 °C to +85 °C operating temperature range
- C161U is available in a 100-Pin P-TQFP package

Power Management

Besides the basic power-save (power-reduction) modes Idle mode and Power down mode, the C161U offers a number of additional power management features, which can be selectively used for effective power reduction. Refer to **Table 1**.

Table 1 Overview of Power Management Modes

Mode	Description	CPU Wake-up
Running mode	The system is fully operational. All clocks and peripherals are set and enabled, as determined by software. Full power consumption.	----
Slow down mode	The CPU runs slower. The oscillator runs at a lower frequency; the clock is divided by a programmable factor (1...32). Peripherals management is possible; incl. PLL On/Off. Refer to register SYSCON 2.	Controlled by software.
Idle mode	When the processor has no active tasks to perform, it enters Idle mode by the IDLE command. All peripherals remain powered and clocked, however, peripherals management is possible. For detailed description see Chapter 19.1, "Idle Mode".	• Any interrupt • Reset
Sleep mode	The program stops execution and turns off the clocks for: • almost the entire chip, but RTC, or • the entire chip. The whole clock system is stopped. Refer to register SYSCON 1.	• All enabled external interrupts • NMI • RTC timer (in asynchronous mode) • PEC requests • ASC interface • SSC interface
Power down mode	The program stops execution (instruction PWRDN) and turns off the clocks for the CPU and for all peripherals; ports optionally.	• Reset

Note: Peripherals Management enables the user to control (via software) the clock of selected peripherals. Refer to register SYSCON 3.

C161U power requirement in individual modes is described in **DC Characteristics, Table 94**

1.2 Logic Symbol

C161U logic symbol is shown in **Figure 1** below.

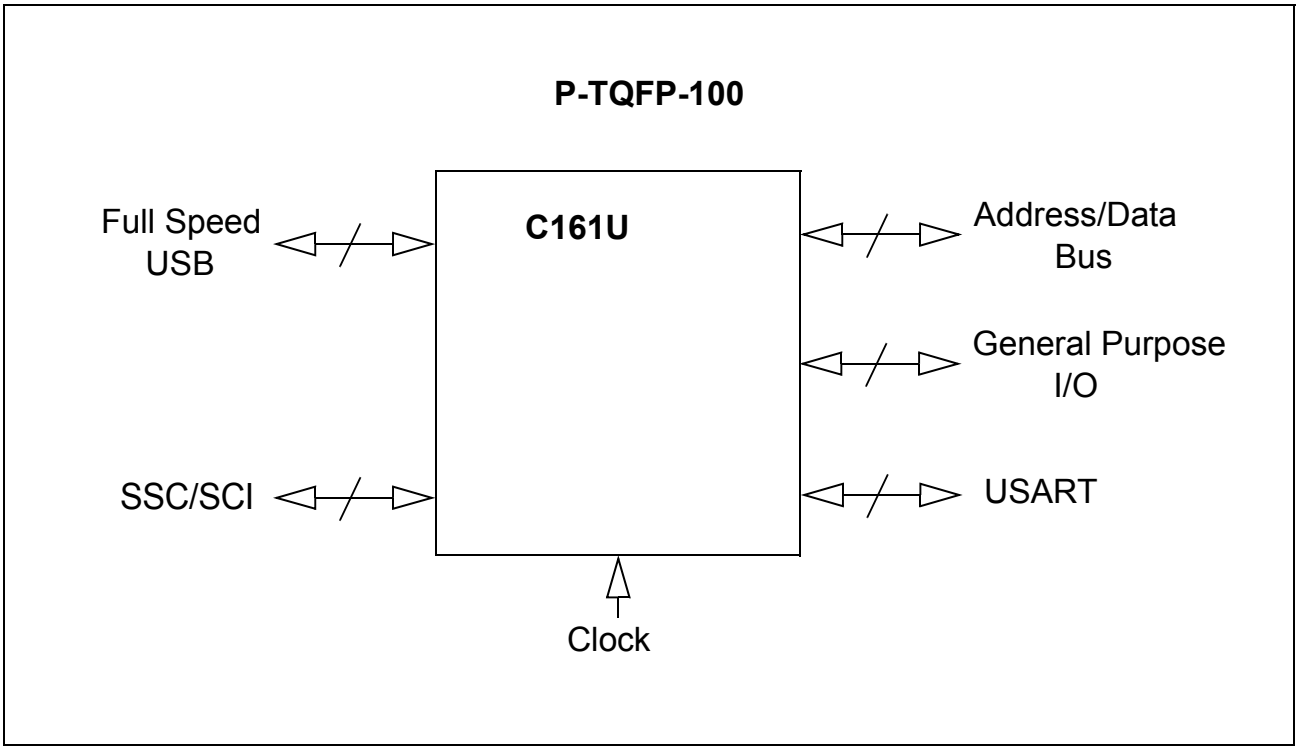


Figure 1 C161U Logic Symbol

1.3 Pinning Diagram

Figure 2 shows the pinning diagram of the C161U.

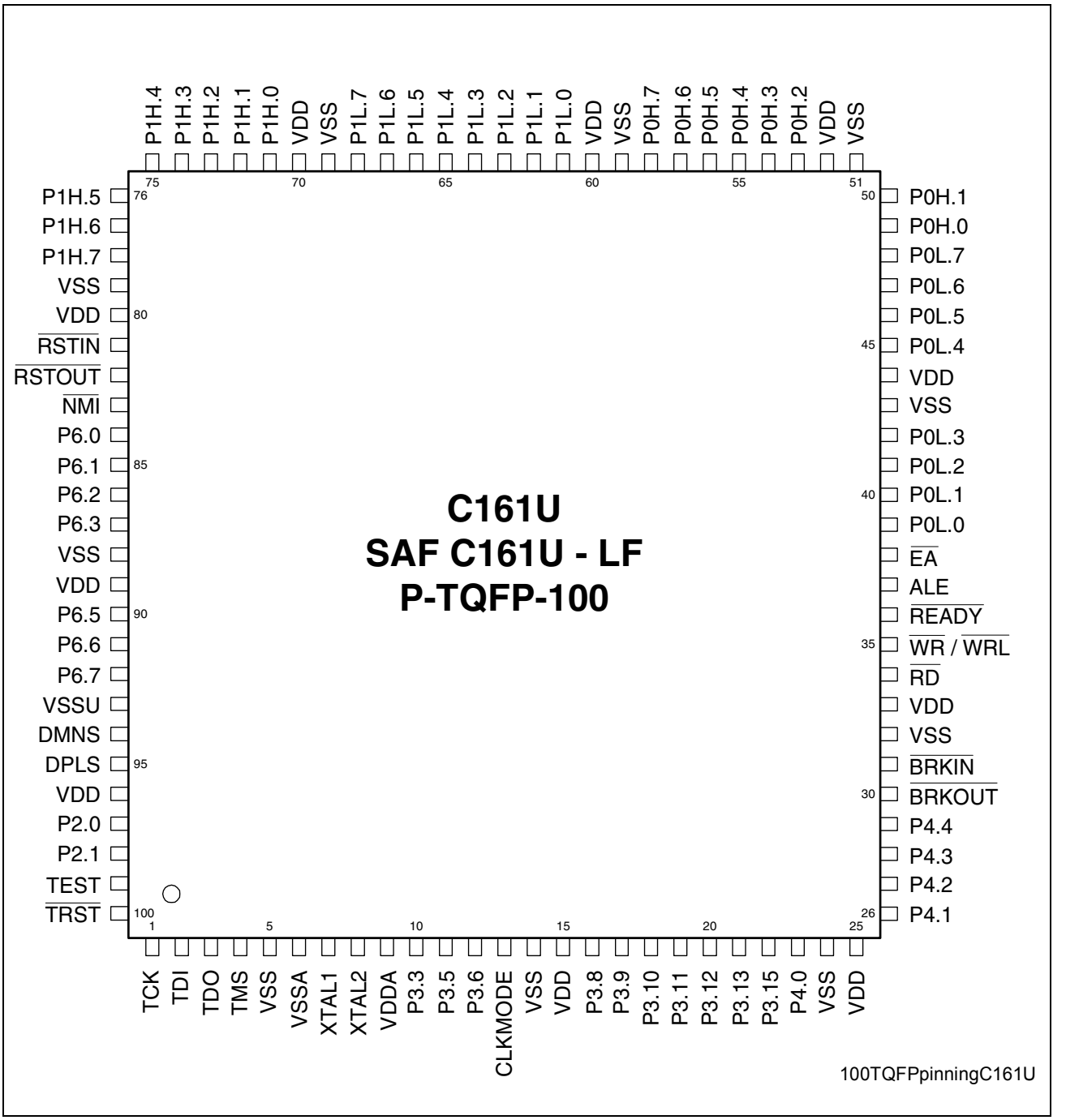


Figure 2 Pinning Diagram of the C161U

1.4 Typical Applications

1.4.1 Personal Computer (PC) Peripherals Applications

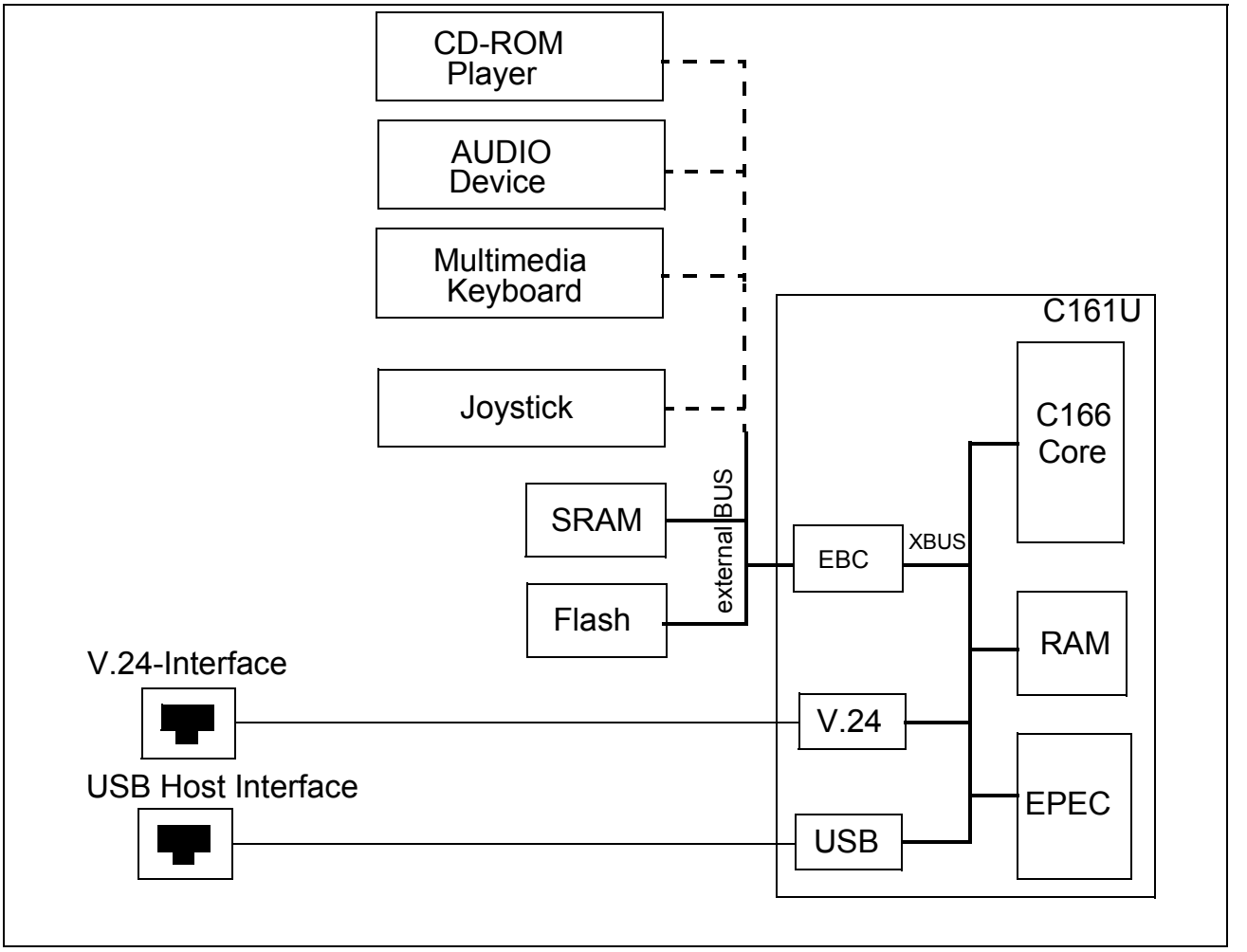


Figure 3 C161U in Personal Computer Peripherals

2 Pin Descriptions

2.1 C161U Pin Diagram

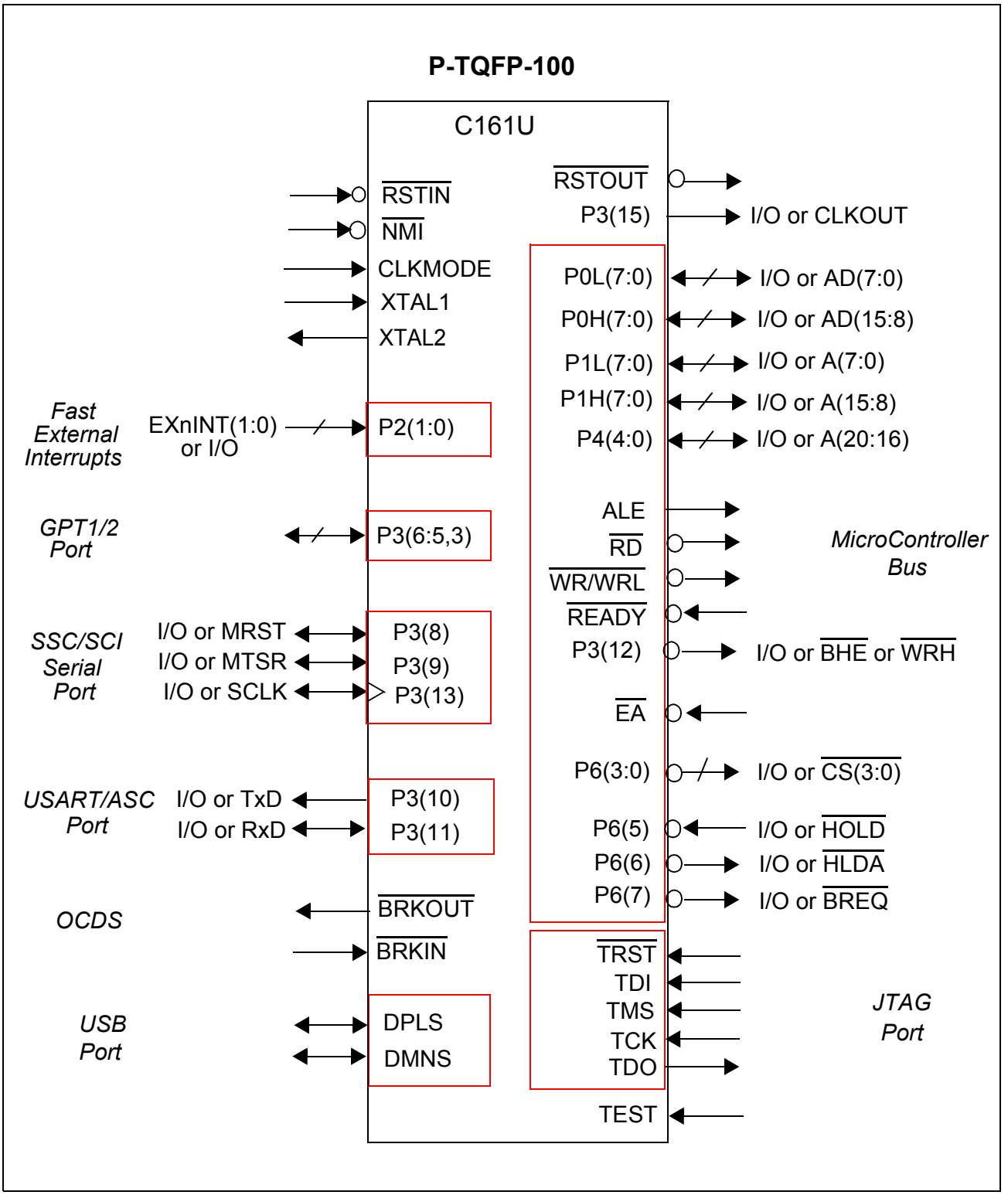


Figure 4 C161U Pin Configuration

2.2 C161U Pin Definitions and Functions

Table 2 Microprocessor Bus and Control Signals

Pin No.	Symbol	Input (I) Output (O)	Function
39-42, 45-48, 49-50, 53-58	PORT0: P0L0-P0L7, P0H0-P0H7	I/O	PORT0 consists of the two 8-bit bidirectional I/O ports P0L and P0H. It is bitwise programmable for input or output via direction bits. For a pin configured as input, the output driver is put into high-impedance. In case of an external bus configuration, PORT0 serves as the address (A) and address/data (AD) bus in demultiplexed bus modes. **Demultiplexed bus modes:** Data Path Width: 8-bit 16-bit P0L0-P0L7: D0-D7 D0-D7 P0H0-P0H7: I/O D8-D15 **Multiplexed bus modes:** Data Path Width: 8-bit 16-bit P0L0-P0L7: AD0-AD7 AD0-AD7 P0H0-P0H7: A8-A15 AD8-AD15
61-68, 71-78	PORT1: P1L0-P1L7, P1H0-P1H7	I/O	PORT1 consists of the two 8-bit bidirectional I/O ports P1L and P1H. It is bitwise programmable for input or output via direction bits. For a pin configured as input, the output driver is put into high-impedance. PORT1 is used as the 16-bit address bus (A) in demultiplexed bus modes and also after switching from a demultiplexed bus mode to a multiplexed bus mode (see Chapter 8.3).

Table 2 Microprocessor Bus and Control Signals (cont'd)

Pin No.	Symbol	Input (I) Output (O)	Function
23, 26-29	P4.0 - P4.4	I/O	PORT4 is an 5-bit bidirectional I/O port. It is bit-wise programmable for input or output via direction bits. For a pin configured as input, the output driver is put into high-impedance state. In case of an external bus configuration, Port4 can be used to output the segment address lines:
		O	P40 A16 Least Significant Segment Address Line
			... ...
		O	P4.4 A20 Most Significant Segment Address Line
81	$\overline{\text{RSTIN}}$	I	Reset Input with Schmitt-Trigger characteristics. A low level at this pin for a specified duration while the oscillator is running resets the device. An internal pull-up resistor permits power-on reset using only a capacitor connected to V_{SS}.
82	$\overline{\text{RSTOUT}}$	O	Internal Reset Indication Output. This pin is set to a low level when the C161U is executing either a hardware-, software- or a watchdog timer reset. $\overline{\text{RSTOUT}}$ remains low until the C161U has initialized itself.
83	$\overline{\text{NMI}}$	I	Non-Maskable Interrupt Input. A high to low transition at this pin causes the CPU to vector to the NMI trap routine. When the PWRDN (power down) instruction is executed, the $\overline{\text{NMI}}$ pin must be low in order to force the CPU to go into power down mode. If $\overline{\text{NMI}}$ is high, when PWRDN is executed, the device will continue to run in normal mode. If not used, pin $\overline{\text{NMI}}$ should be pulled high externally.

Table 2 Microprocessor Bus and Control Signals (cont'd)

Pin No.	Symbol	Input (I) Output (O)	Function
84-87, 90-92	P6.0- P6.3, P6.5- P6.7	O I/O O ... O I O O	Port6 is an 7-bit bidirectional I/O port. It is bit-wise programmable for input or output via direction bits. For a pin configured as input, the output driver is put into high-impedance state. Port6 outputs can be configured as push/pull or open-drain drivers. P6.0 $\overline{CS0}$ Chip Select 0 Output ... ... P6.3 $\overline{CS3}$ Chip Select 3 Output P6.5 $\overline{HOLD}$ External Master Hold Request Input P6.6 $\overline{HLDA}$ Hold Acknowledge Output P6.7 $\overline{BREQ}$ Bus Request Output
97-98	P2.0- P2.1	I/O I I	PORT2 is an 2-bit bidirectional I/O port. It is bit-wise programmable for input or output via direction bits. For a pin configured as input, the output driver is put into high-impedance state. Port2 outputs can be configured as push/pull or open-drain drivers. P2.0 EX0IN Fast External Interrupt 0 Input P2.1 EX1IN Fast External Interrupt 1 Input
34	$\overline{RD}$	O	External Memory Read Strobe. $\overline{RD}$ is activated for every external instruction or data read access.
35	$\overline{WR}$/$\overline{WRL}$	O	External Memory Write Strobe. In $\overline{WR}$ mode this pin is activated for every external data write access. In $\overline{WRL}$ mode this pin is activated for low byte data write accesses on a 16-bit bus, and for every data write access on an 8-bit bus. See WRCFG in register SYSCON for mode selection.
37	ALE	O	Address Latch Enable Output. Can be used for latching the address into external memory or an address latch in the multiplexed bus modes.

Table 2 Microprocessor Bus and Control Signals (cont'd)

Pin No.	Symbol	Input (I) Output (O)	Function
36	$\overline{\text{READY}}$	I	Ready Input. When the ready function is enabled, a high level at this pin during an external memory access will force the insertion of memory cycle time waitstates until the pin returns to an low level.
38	$\overline{\text{EA}}$	I	External Access Enable pin. A low level at this pin during and after Reset forces the CPU to begin instruction execution out of external memory. **Note:** This pin must always be set to '0'.

Table 3 General Purpose I/O and Control Signals

Pin No.	Symbol	Input (I) Output (O)	Function
10-12, 16-22	P3.3, P3.5- P3.6, P3.8- P3.13, P3.15	I/O I/O I/O	PORT3 is a 10-bit bidirectional I/O port. It is bit-wise programmable for input or output via direction bits. For a pin configured as input, the output driver is put into high-impedance state. Port3 outputs can be configured as push/pull or open-drain drivers. The following PORT3 pins also serve for alternate functions:
		O	P3.3 T3OUT GPT1 Timer T3 Toggle Latch Output
		I	P3.5 T4IN GPT1 Timer T4 Input for Count/Gate/Reload/Capture
		I	Input for Timer 3 T3EUD Input for Timer 2 T2EUD
		I	P3.6 T3IN GPT1 Timer T3 Count/Gate/ Input
		I/O	P3.7 T2IN GPT1 Timer T2 Input for Count/Gate/Reload/Capture
		I/O	P3.8 MRST SSC Master-Rec./Slave-Transmit I/O
		O	P3.9 MTSR SSC Master-Transmit/Slave-Rec. O/I
		I/O	P3.10 TxD0 ASC Clock/Data Output (Async./Sync.)
		O	P3.11 RxD0 ASC Data Input (Async.) or I/O (Sync.)
		O	P3.12 $\overline{\text{BHE}}$ External Memory High Byte Enable Signal
		I/O	$\overline{\text{WRH}}$ External Memory High Byte Write Strobe
		O	P3.13 SCLK SSC Master Clock Output/ Slave Clock Input (CPU Clock)
			P3.15 CLKOUT System Clock Output (CPU Clock)

Table 4 USB Interface Signals

Pin No.	Symbol	Input (I) Output (O)	Function
95	DPLS	I/O	USB Data+ input/output signal.
94	DMNS	I/O	USB Data- input/output signal.

Table 5 Clock Interface Signals

Pin No.	Symbol	Input (I) Output (O)	Function
7	XTAL1	I	External crystal input to the on-chip oscillator. Clock input for direct driven clock without using an external crystal. Function is determined by the CLKMODE pin.
8	XTAL2	O	Output from the oscillator amplifier circuit. To clock the C161U from an external source, drive XTAL1, while XTAL2 leaving unconnected. Minimum and maximum high/ low and rise/fall times specified in the AC characteristics must be observed.
13	CLKMODE	I	Clock Mode Select pin. CLKMODE must be set to LOW if an external crystal is used. Set to HIGH signal enables the direct clock input path and switches the internal oscillator in power down mode.

Table 6 Boundary Scan / JTAG / Test Interface Signals/OCDS

Pin No.	Symbol	Input (I) Output (O)	Function
1	TCK	I	Boundary Scan Test Clock Input. There is no internal pull device implemented. During normal operation, it is recommended to connect TCK to VSS.
2	TDI	I	Boundary Scan Test Data Input. An internal pull-up device is connected to TDI. During normal operation, TDI can be left open.

Table 6 Boundary Scan / JTAG / Test Interface Signals/OCDS

Pin No.	Symbol	Input (I) Output (O)	Function
3	TDO	O	Boundary Scan Test Data Output. During normal operation, the output TDO can be left open.
4	TMS	I	Boundary Scan Test Mode Select Input, internal pull-up.
100	$\overline{\text{TRST}}$	I	Boundary Scan Test Reset. There is an internal pull-up device implemented. $\overline{\text{TRST}}$ is low active, which means the boundary scan tap controller resets while $\overline{\text{TRST}}$ = ’0’. For normal operation, • $\overline{\text{TRST}}$ can be connected to a LOW signal (using ’0’ signal or external pull-down device) to keep the tap controller in reset mode, or • $\overline{\text{TRST}}$ can be left open. In this case, the reset is performed using the TMS/TCK signals according to IEEE 1149.1 In boundary scan test mode, $\overline{\text{TRST}}$ can be left open, since the internal pull-up device provides the necessary HIGH signal.
99	TEST	I	Test Mode Enable Pin. HIGH signal enables the chip internal test mode. **Note:** In normal operation, TEST must be connected to VSS (LOW signal) since no internal Pull-Down resistor is provided.
31	$\overline{\text{BRKIN}}$	I	In OCDS mode, a falling edge from HIGH to LOW signal on $\overline{\text{brkin}}$ forces the system to stop. An internal pull-up resistor is provided.
30	$\overline{\text{BRKOUT}}$	O	In OCDS mode, a falling edge on $\overline{\text{brkout}}$ indicates the trigger of a pre-selected OCDS event.

Table 7 Power/Ground Signals

Pin No.	Symbol	Input (I) Output (O)	Function
15, 25, 33, 44, 52, 60, 70, 80, 89, 96	VDD	-	**Digital Supply Voltage** **Note:** All pins must be connected to VDD.
5, 14, 24, 32, 43, 51, 59, 69, 79, 88	VSS	-	**Digital Ground** **Note:** All pins must be connected to VSS.
9	VDDAX	-	**Analog Supply Voltage:** VDDAX supplies the oscillator circuitry only, and is internal not connected to VDD in order to separate possible noise influence from the noise sensitive part. External, on board level, the VDDAX can be connected to the same power supply as the VDD.
6	VSSAX	-	**Analog Ground** VSSAX is connected to the oscillator circuitry Gound only, in order to separate possible noise influence. External, on board level, the VSSA can be connected to the same Ground as the VSS.
93	VSSU	-	**Digital Ground for USB Transceiver** VSSU is connected to the USB transceiver only and is internally not connected to the common ground in order to separate possible noise influence. External, on board level, the VSSU can be connected to the same ground as VSS.

3 Architectural Overview

The architecture of the C161U 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 C161U 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 C161U 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 C161U.

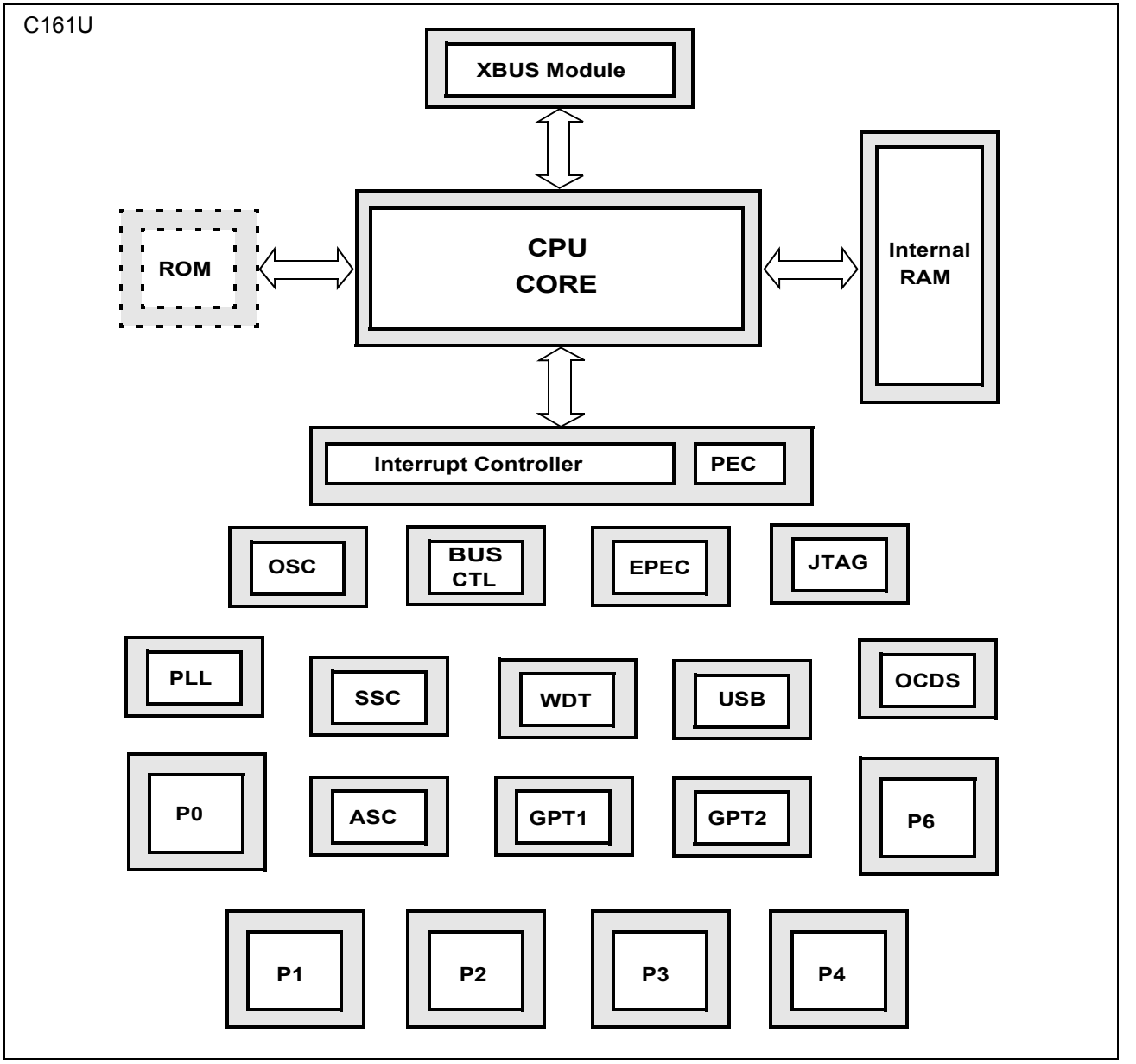


Figure 5 C161U Functional Block Diagram

3.1 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.

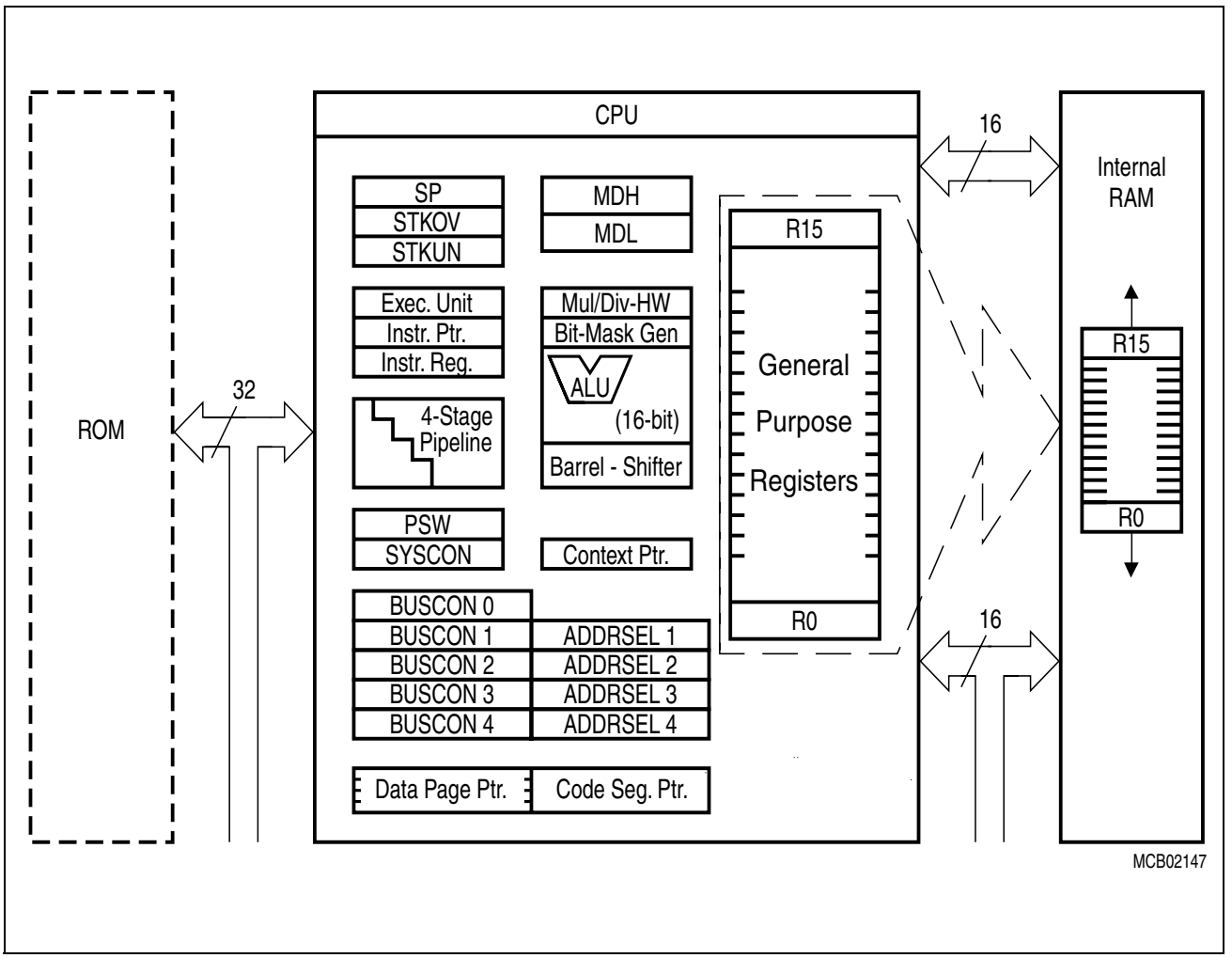


Figure 6 CPU Block Diagram

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 C161U's instructions can be executed in just one machine cycle, which requires 55.6 ns at 36 MHz CPU clock. 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 1µs, a 16-bit * 16-bit multiplication takes 0.5 µs.

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 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 C161U 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 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 140 ns to 280 ns (in case of internal program execution), the C161U is capable of reacting very fast on non-deterministic events.

Its fast external interrupt inputs are sampled every 28 ns and allow to recognize even very short external signals.

C161U 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.

3.2 On-Chip System Resources

C161U 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.

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. C161U has 8 PEC channels each of which offers such fast interrupt-driven data transfer capabilities.

Memory Areas

The memory space of the C161U is configured in a Von Neumann architecture which means that code memory, data memory, registers and I/O ports are organized within the same linear address space which covers up to 2 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 16-bit wide internal RAM (IRAM) 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 size of the internal RAM is 3 KByte.

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 1024 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.

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 C161U 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 2 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 four different external access modes:

- 16-/18-/20-/24-bit Addresses, 16-bit Data, Demultiplexed
- 16-/18-/20-/24-bit Addresses, 8-bit Data, Demultiplexed
- 16-/18-/20-/24-bit Addresses, 16-bit Data, Multiplexed
- 16-/18-/20-/24-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. All modes use Port 4 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 (A20/A19/A17...A16), as is the case when using the segmented memory model.

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.

3.3 Clock Generation Concept

The on-chip clock generator provides the C161U with its basic clock signal that controls all activities of the controller hardware. Its oscillator can either run with an external crystal and appropriate oscillator circuitry (see also recommendations in chapter „Dedicated Pins“) or it can be driven by an external oscillator. The oscillator either directly feeds the external clock signal to the controller hardware (through buffers), divides the external clock frequency by 2 or 4, or feeds an on-chip phase locked loop (PLL) which multiplies the input frequency by a selectable factor **F**. This resulting internal clock signal is also referred to as “CPU clock”. Two separated clock signals are generated for the CPU itself and the peripheral part of the chip. While the CPU clock is stopped during the idle mode, the peripheral clock keeps running. Both clocks are switched off, when the power down mode is entered.

Note: Pin13 CLKMODE must be connected to LOW signal if an external crystal is used. Pin cockmode connected to HIGH signal enables the direct input path and switches the oscillator circuit in power down mode.

The on-chip PLL circuit allows operation of the C161U on a low frequency external clock while still providing maximum performance. The PLL generates a CPU clock signal with 50% duty cycle. The PLL also provides fail safe mechanisms which allow the detection of frequency deviations and the execution of emergency actions in case of an external clock failure.

In addition to the CPU clock, the PLL generates the USB clock which is used for the USB module only. shows the general clock generation concept of the C161U.

Note: If the USB interface of the C161U is used, an USB clock of 48 MHz is mandatory. In addition, a CPU clock equal or greater than 20 MHz is required in order to guarantee the full USB functionality. According to **Table 8**, the only possible input clock frequency when operating the USB interface is 8 MHz, either using an external crystal or an direct input clock.

The following constrains must be taken into account when considering the clock concept:

1. The USB clock must be 48 MHz, see the note above. Since there is a fixed PLL prescaler of 1/6, the XTAL1 frequency must be 8 MHz.
2. If running the USB interface, the CPU clock must be equal or greater than 20 MHz
3. The maximum CPU clock frequency is 36 MHz (18 MIPS).

4. All AC and DC specifications described in Chapter 24, "AC/DC Characteristics" must be fulfilled.
5. The input frequency of the internal oscillator circuit is 4 MHz up to 20 MHz. This applies only, if an external crystal is used.
6. In direct drive mode, the internal oscillator circuit is bypassed. The clock input frequency range is 4 MHz up to 36 MHz.

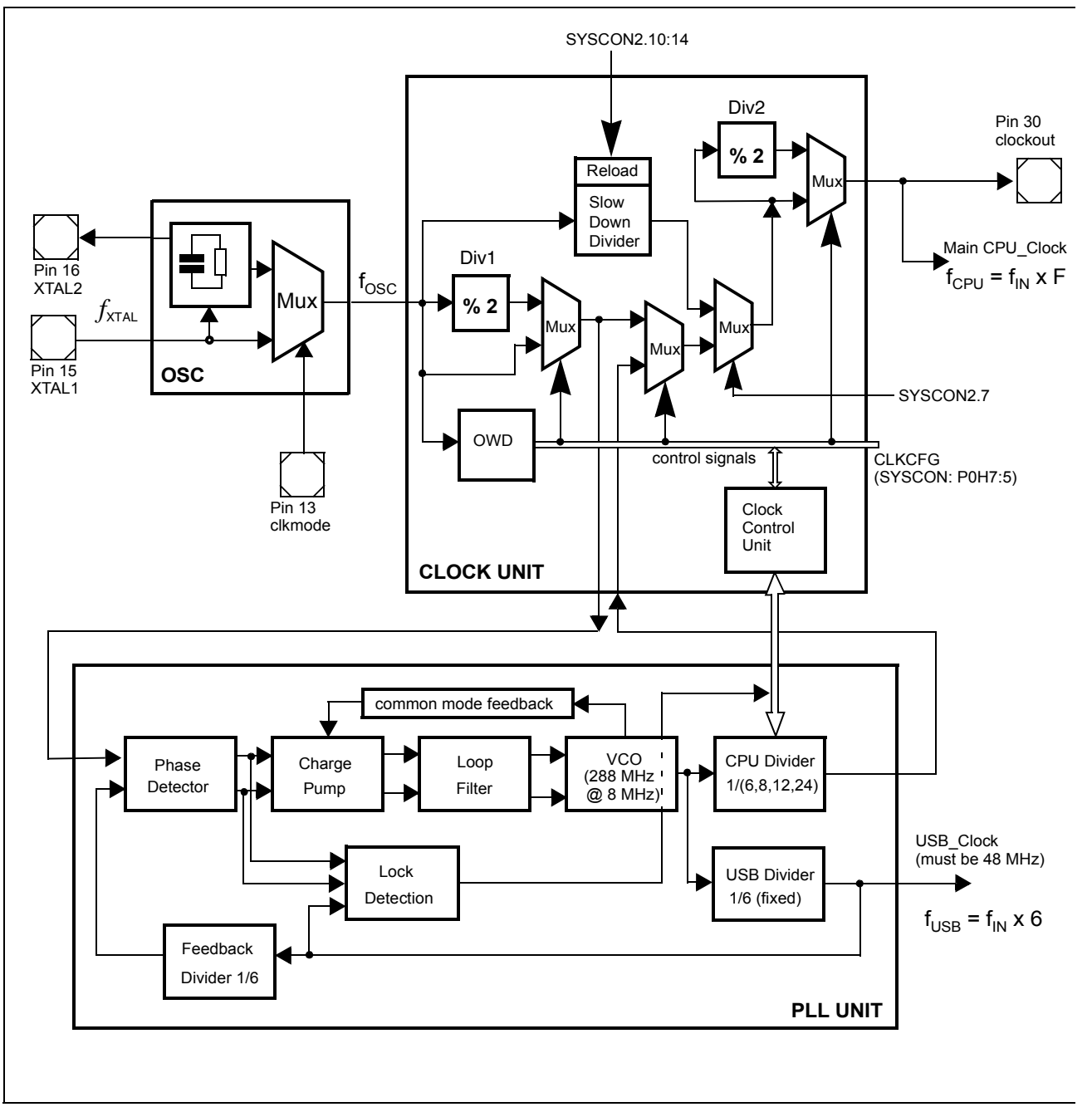


Figure 7 C161U General Clock Concept

Note: All supported clock modes for the C161U are shown in **Table 8**. Because of the limited size of the register, there are not all combinations adjustable, which can be derived theoretical from .

Table 8 C161U Clock Generation Modes

P0H.7-P0H.5	Frequency	Divider Activation
USB Interface is NOT used		
0 0 1	f_{XTAL} * 0.5	direct drive, D1 not active, D2 active, PLL free running (2..5 MHz) **Note:** The PLL can be switched off completely by setting bit PLLDIS = '1' (SYSCON3.13, see page 393).
0 1 0	f_{XTAL} * 1.5	D1 not active, D2 not active, F = 1.5
0 1 1	f_{XTAL} * 1.0	direct drive, D1 not active, D2 not active, PLL free running (2..5 MHz) **Note:** The PLL can be switched off completely by setting bit PLLDIS = '1' (SYSCON3.13, see page 393).
1 0 0	f_{XTAL} * 6.0	D1 not active, D2 not active, F = 6.0
1 0 1	f_{XTAL} * 1.125	D1 active, D2 active, F = 1.125
1 1 0	f_{XTAL} * 3.0	D1 not active, D2 not active, F = 3.0
1 1 1	**f_{XTAL} * 4.5**	**D1 not active, D2 not active, F = 4.5, Default Mode**
0 0 0	f_{XTAL} * 0.375	D1 active, D2 active, F = 0.375
USB Interface is used (USB clock must be 48 MHz)		
1 1 0	f_{XTAL} * 3	D1 not active, D2 not active, F = 3.0 f_{USB} =$_{def}$ 48 MHz f_{XTAL} = 8 MHz f_{CPU} = 24 MHz
1 1 1	**f_{XTAL} * 4.5**	**D1 not active, D2 not active, F = 4.5, Default Mode** **f_{USB} =$_{def}$ 48 MHz f_{XTAL} = 8 MHz f_{CPU} = 36 MHz**

PLL Operation

On power-up the PLL provides a stable clock signal within ca. 1 ms after VDD has reached 3.3 V±10%, even if there is no external clock signal (in this case the PLL will run on its basic frequency of 2...5 MHz). The PLL starts synchronizing with the external clock signal as soon as it is available. Within ca. 1 ms after stable oscillations of the external clock within the specified frequency range the PLL will be synchronous with this clock at a frequency of **F** * f_{OSC}, ie. the PLL locks to the external clock.

Note: If the C161U is required to operate on the desired CPU clock directly after reset make sure that RSTIN remains active until the PLL has locked (ca. 1 ms).

When PLL operation is selected the CPU clock is a selectable multiple of the oscillator frequency, ie. the input frequency. The table above lists the possible selections.

The PLL constantly synchronizes to the external clock signal. Due to the fact that the external frequency is 1/**F'**th of the PLL output frequency the output frequency may be slightly higher or lower than the desired frequency. This jitter is irrelevant for longer time periods. For short periods (1...4 CPU clock cycles) it remains below 4%.

When the PLL detects a missing input clock signal it generates an interrupt request. This warning interrupt indicates that the PLL frequency is no more locked, ie. no more stable. This occurs when the input clock is unstable and especially when the input clock fails completely, eg. due to a broken crystal. In this case the synchronization mechanism will reduce the PLL output frequency down to the PLL's basic frequency (2...5 MHz). The basic frequency is still generated and allows the CPU to execute emergency actions in case of a loss of the external clock.

Prescaler Operation

When pins P0.15-13 (P0H.7-5) are equal '001' during reset the CPU clock is derived from the internal oscillator (input clock signal) by a 2:1 prescaler (see **Table 8**).

The frequency of f_{CPU} is half the frequency of f_{XTAL} and the high and low time of f_{CPU} (ie. the duration of an individual TCL) is defined by the period of the input clock f_{XTAL}.

The timings listed in the 'AC Characteristics' of the data sheet that refer to TCLs therefore can be calculated using the period of f_{XTAL} for any TCL.

Direct Drive

When pins P0.15-13 (P0H.7-5) equal '011' during reset the clock system is directly driven from the internal oscillator with the input clock signal, ie. $f_{OSC} = f_{CPU}$.

The maximum input clock frequency depends on the clock signal's duty cycle, because the minimum values for the clock phases (TCLs) must be respected.

Oscillator Watchdog

The C161U provides an Oscillator Watchdog (OWD) which monitors the clock signal generated by the on-chip oscillator (either with a crystal or via external clock drive) in prescaler or direct drive mode. For this operation the PLL provides a clock signal which is used to supervise transitions on the oscillator clock. This PLL clock is independent from the XTAL1 clock. When the expected oscillator clock transitions are missing the OWD activates the PLL Unlock / OWD interrupt node and supplies the CPU with the PLL clock signal. Under these circumstances the PLL will oscillate with its basic frequency.

The OWD's interrupt output can be disabled by setting bit OSCENBL = '0' (default after reset) in SYSCON register. In this case, no oscillator watchdog interrupt request is generated and the CPU clock signal is derived from the oscillator clock in any case.

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

3.4 On-Chip Peripheral Blocks

C161U 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.

C161U peripherals are:

- Two General Purpose Timer Blocks (GPT1 and GPT2)
- An Asynchronous/Synchronous Serial Interface (ASC)
- A High-Speed Synchronous Serial Interface (SSC)
- An Universal Serial Bus Interface (USB)
- A Watchdog Timer (WDT)
- Six I/O ports with a total of 56 I/O lines

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

Peripheral Interfaces

The on-chip peripherals generally have two different types of interfaces, an interface to the CPU and an interface to external hardware. Communication between 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 (eg. 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 I/O 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** it must be guaranteed 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 C161U'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. Read accesses to reserved bits return '0's.

Parallel Ports

C161U provides up to 72 I/O lines which are organized into seven input/output ports. All port lines are bit-addressable, and all input/output lines are individually (bit-wise) programmable as inputs or outputs via direction registers. The I/O ports are true bidirectional ports which are switched to high impedance state when configured as inputs. The output drivers of three I/O ports can be configured (pin by pin) for push/pull operation or open-drain operation via control registers. During the internal reset, all port pins are configured as inputs.

All port lines have programmable alternate input or output functions associated with them. PORT0 and PORT1 may be used as address and data lines when accessing external memory, while Port 4 outputs the additional segment address bits A20/A19/ A17...A16 in systems where segmentation is used to access more than 64 KBytes of memory. Port 6 provides optional bus arbitration signals ($\overline{\text{BREQ}}$, $\overline{\text{HLDA}}$, $\overline{\text{HOLD}}$) and chip select signals. 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 7 is used for general purpose I/Os. All port lines that are not used for these alternate functions may be used as general purpose I/O 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 (ASC) and a High-Speed Synchronous Serial Channel (SSC).

ASC is upward compatible with the serial ports of the Infineon 8-bit microcontroller families and supports full-duplex asynchronous communication at up to 2.25 MBaud and half-duplex synchronous communication at up to 4.5 MBaud @ 36 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 ASC transmits or receives bytes (8 bits) synchronously to a shift clock which is generated by the ASC. The ASC 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.

SSC supports full-duplex synchronous communication at up to 18 Mbaud @ 36 MHz CPU clock in SSC master mode and up to 9 MBaud @ 36 MHz in SSC slave mode. 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.

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.

The five 16-bit timers are organized in two separate modules, GPT1 and GPT2. Each timer in each module 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 three basic modes of operation, which are Timer, Gated Timer, and Counter Mode. In Timer Mode the input clock for a timer is derived from the internal CPU clock divided by a programmable prescaler, while Counter Mode allows a timer to be clocked in reference to external events (via TxIN).
Pulse width or duty cycle measurement is supported in Gated Timer Mode where the operation of a timer is controlled by the ‘gate’ level on its external input pin TxIN.

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 eg. 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 output on port pins (TxOUT) or 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.

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 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.

3.5 Protected Bits

C161U 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	ASC transmit(buffer) interrupt request flags
S0RIC, S0EIC	S0RIR, S0EIR	ASC receive/error interrupt request flags
S0CON	S0REN	ASC 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
TFR	TFR.15,14,13	Class A trap flags
TFR	TFR.7,3,2,1,0	Class B trap flags
XPyIC (y=3...0)	XPyIR (y=3...0)	X-Peripheral y interrupt request flag

Σ = 33 protected bits in the C161U

4 Memory Organization

The memory space of the C161U 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, 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.

C161U provides a total addressable memory space of 2 MBytes. This address space is arranged as 32 segments of 64 KBytes each, and each segment is again subdivided into four data pages of 16 KBytes each (see **Figure 8**).

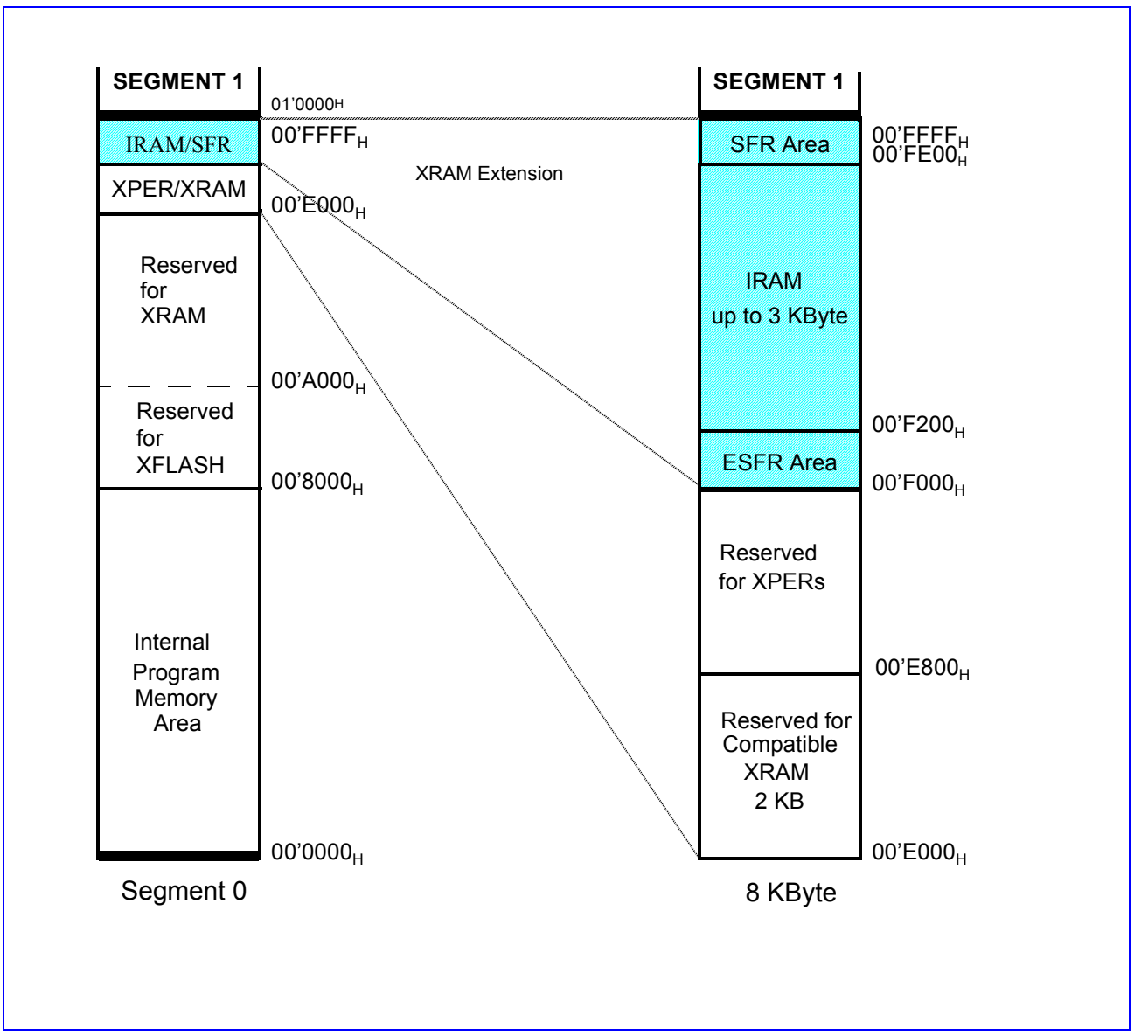


Figure 8 Memory Areas and Address Space

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.

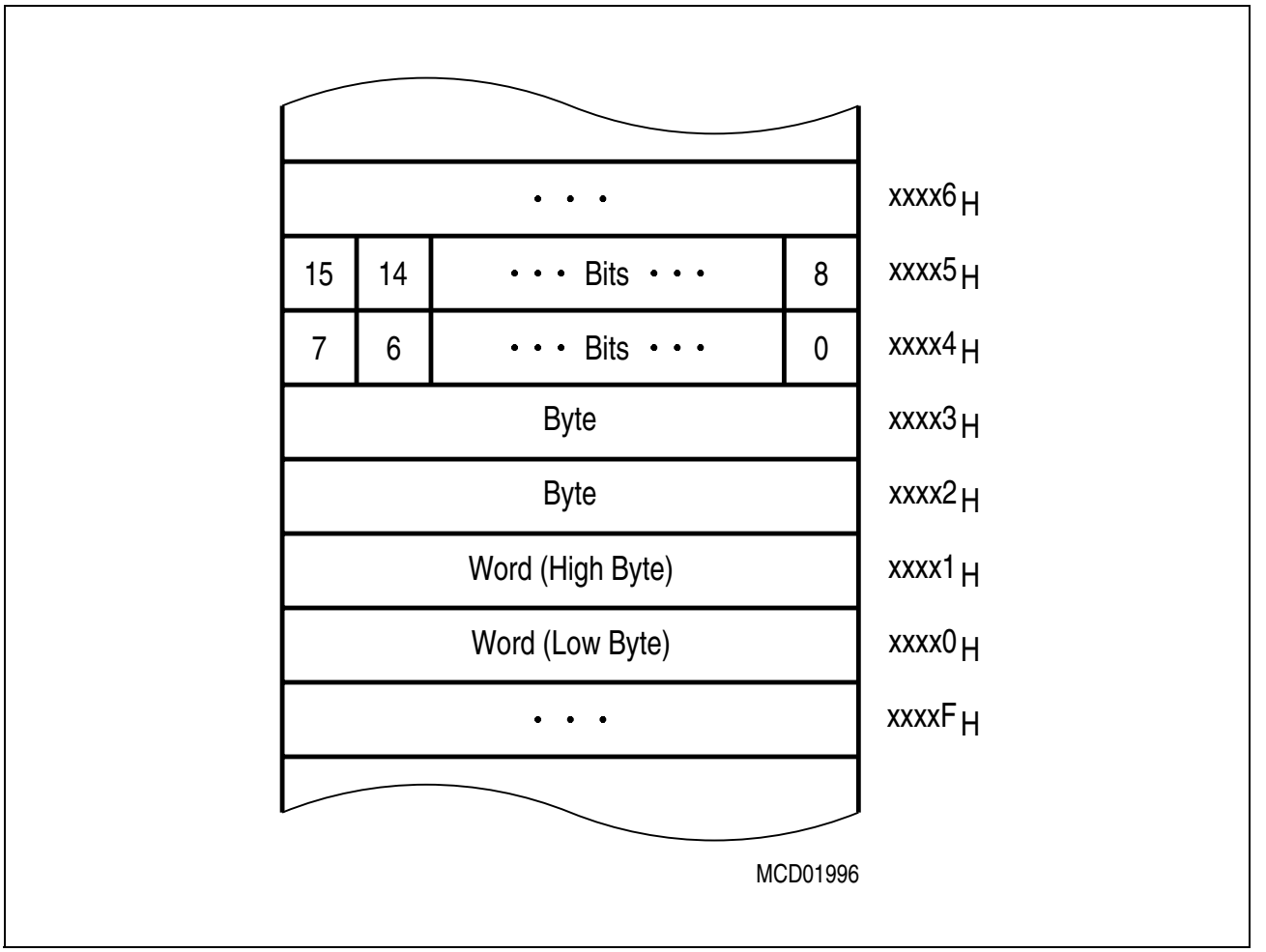


Figure 9 Storage of Words, Byte and Bits in a Byte Organized Memory

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

4.1 Internal RAM and SFR Area

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). C161U provides 3 KByte of IRAM, see **Figure 10**.

The internal RAM serves for several purposes:

- System Stack (programmable size)
- General Purpose Register Banks (GPRs)
- Source and destination pointers for the Peripheral Event Controller (PEC)
- Variable and other data storage, or
- Code storage.
-

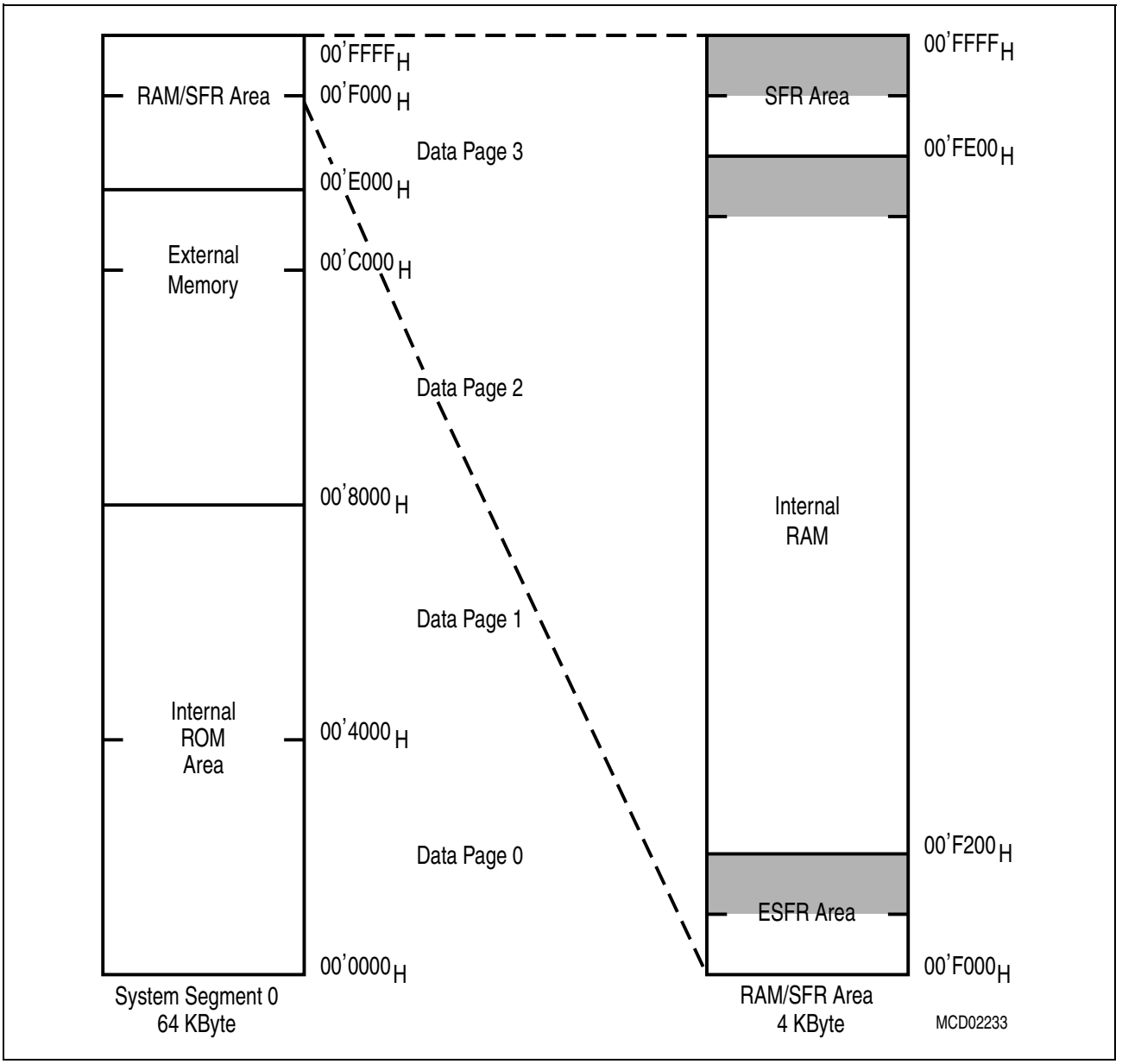


Figure 10 Internal RAM Area and SFR Areas

Note: The upper 256 Bytes of SFR area, ESFR area and internal RAM are bit-addressable (see shaded blocks in **Figure 10**).

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 $_B$	512	00'FBFE$_H$...00'F800$_H$
1 0 1 $_B$	---	Reserved. Do not use this combination.
1 1 0 $_B$	---	Reserved. Do not use this combination.
1 1 1 $_B$	1024	00'FDFE$_H$...00'F600$_H$ (Note: No circular stack)

For all system stack operations the on-chip RAM is accessed via the Stack Pointer (SP) register. The stack grows downward from higher towards lower RAM address locations. Only word accesses are supported to the system stack. A stack overflow (STKOV) 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

C161U 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).

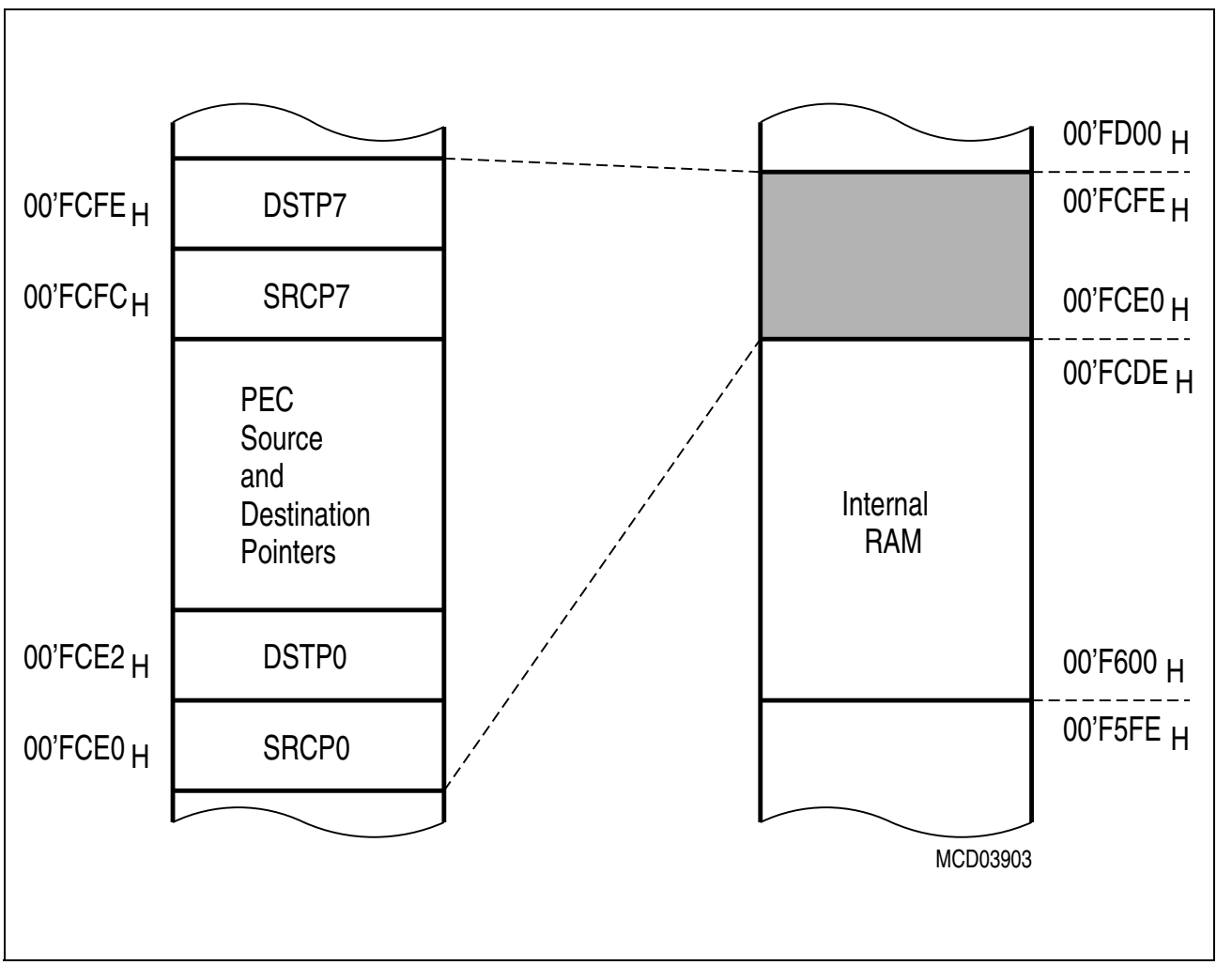


Figure 11 Location of the PEC Pointers

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 I/O ports and the on-chip peripherals of the C161U 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.

4.2 External Memory Space

C161U is capable of using an address space of up to 2 MByte. Only parts of this address space are occupied by internal memory areas. All addresses which are not used for on-chip memory (RAM) or for registers may reference external memory locations. This external memory is accessed via the C161U's external bus interface.

Four memory bank sizes are supported:

- Non-segmented mode: 64 KByte — with A15...A0 on PORT0 or PORT1
- 2-bit segmented mode: 256 KByte — with A17...A16 on Port 4 and A15...A0 on PORT0 or PORT1
- 4-bit segmented mode: 1 MByte — with A19...A16 on Port 4 and A15...A0 on PORT0 or PORT1
- 8-bit segmented mode: 2 MByte — with A20...A16 on Port 4 and A15...A0 on PORT0 or PORT1

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

C161U also supports **four different bus types**:

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

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

4.3 Crossing Memory Boundaries

The address space of the C161U 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 on-chip X-Peripherals 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.

5 Central Processor Unit

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 C161U controller, it is also affected by certain actions of the peripheral subsystem.

Since a four stage pipeline is implemented in the C161U, up to four instructions can be processed in parallel. Most instructions of the C161U are executed in one machine cycle (2 CPU clock cycles) 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.

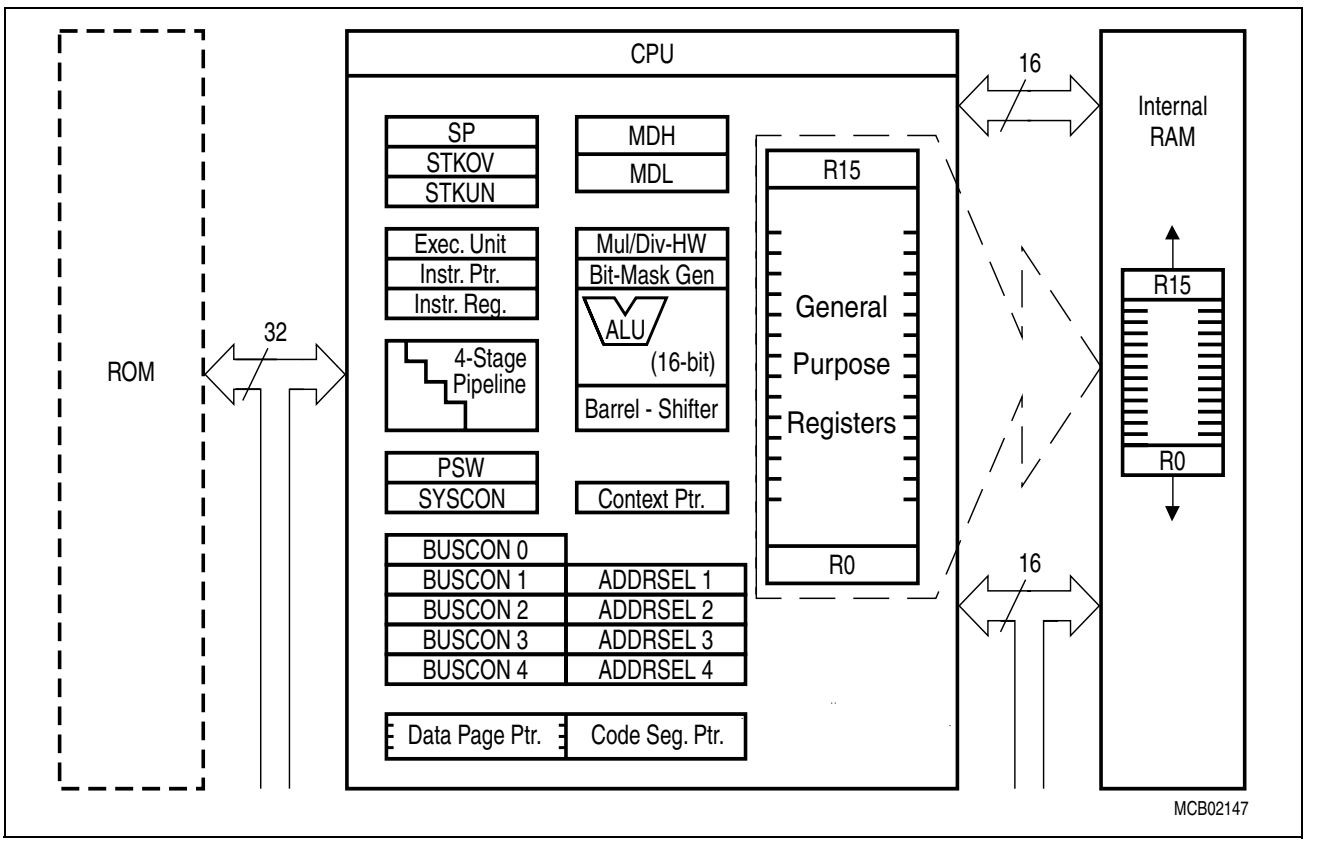


Figure 12 CPU Block Diagram

The on-chip peripheral units of the C161U 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.

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 C161U 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

5.1 Instruction Pipelining

The instruction pipeline of the C161U 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 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 C161U 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 (ie. 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.

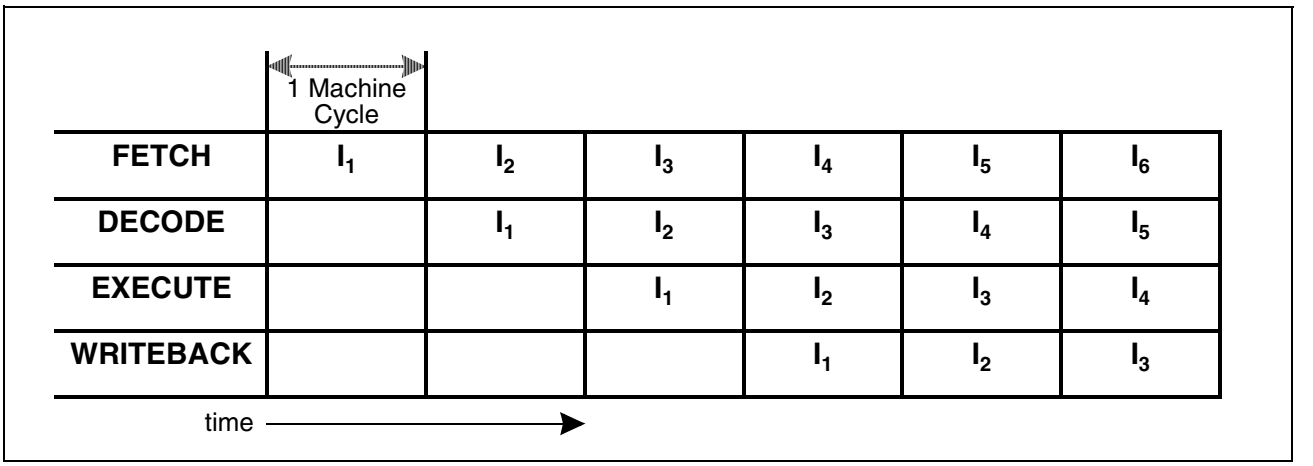


Figure 13 Sequential Instruction Pipelining

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 14**).

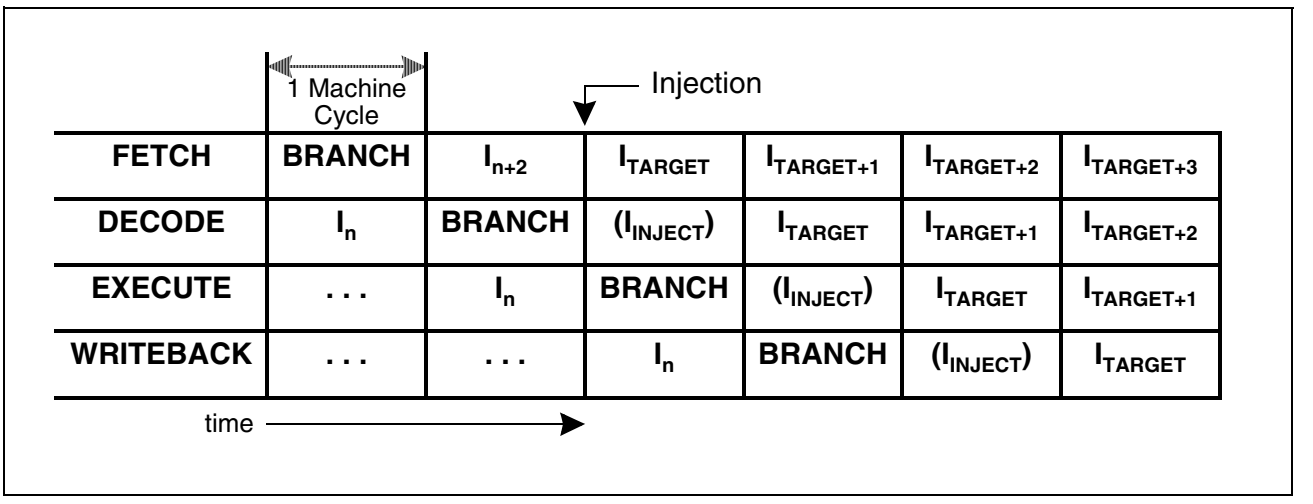


Figure 14 Standard Branch Instruction Pipelining

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

C161U 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 progam memory but taken from the cache and immediatly injected into the decode stage of the pipeline (see **Figure 15**).

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.

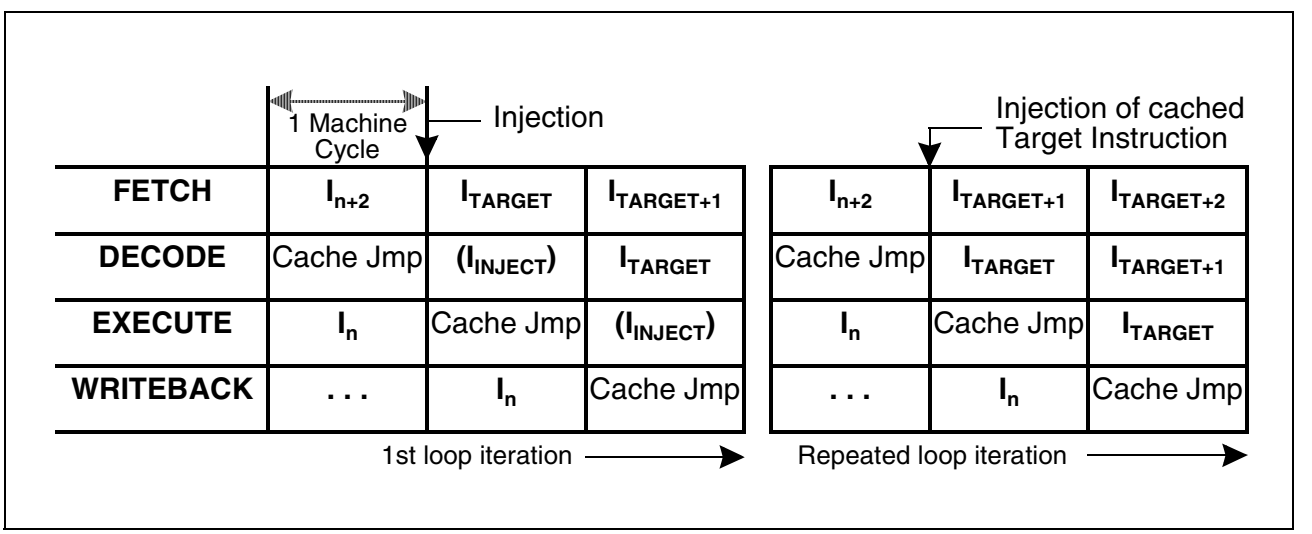


Figure 15 Cache Jump Instruction Pipelining

Particular Pipeline Effects

Since up to four different instructions are processed simultaneously, additional hardware has been spent in the C161U to consider all causal dependencies which may exist on instructions in different pipeline stages without a loss of performance. This extra hardware (ie. for 'forwarding' operand read and write values) resolves most of the possible conflicts (eg. 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 C161U 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:

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

- **Data Page Pointer Updating**

An instruction, which calculates a physical operand address via a particular 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:

```
I_n     : MOV   DPP0, #4         ; select data page 4 via DPP0
I_n+1   : ....                   ; must not be an instruction using DPP0
I_n+2   : MOV   DPP0:0000H, R1   ; move contents of R1 to address location 01'0000_H
                                 ; (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 (eg. 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, ie. 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
                IN-1            ; non-critical instruction
CRIT_1ST:       IN              ; begin of uninterruptable critical sequence
                . . .
CRIT_LAST:      IN+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 ie. 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).

```
WRONG:  BSET    DP3.13   ; change direction of P3.13 to output
        BSET    P3.5     ; P3.13 is still input, the rd-mod-wr reads pin P3.13

RIGHT:  BSET    DP3.13   ; change direction of P3.13 to output
        NOP              ; any instruction not accessing port 3
        BSET    P3.5     ; P3.13 is now output,
                         ; the rd-mod-wr reads the P3.13 output latch
```

- **Changing the System Configuration**

The instruction following an instruction that changes the system configuration via register SYSCON (eg. segmentation, stack size) cannot use the new resources (eg. stack). In these cases an instruction that does not access these resources should be inserted.

- **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.

5.2 Bit-Handling and Bit-Protection

C161U 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 I/O 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, ie. 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”), ie. 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, ie. when hardware sets eg. 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 C161U can be found at the end of chapter “Architectural Overview”.

5.3 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 C161U 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 C161U'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 detailled 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 C161U instruction fetched from the internal RAM or from external memory. These execution times apply to most of the C161U instructions - except some of the branches, the multiplication, the division and a special move instruction. The numbers in the table are in units of [ns], refer to a CPU clock of 20 MHz and assume no waitstates.

Table 9 Minimum Execution Times

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

Execution from the internal RAM provides flexibility in terms of loadable and modifyable code on the account of execution time.
Execution from external memory 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 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

5.4 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'.

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	-	OSC ENBL	-	XPEN	VISI BLE	XPER-SHARE
rw			rw	rw	rw	rw	rw	rw	rw	-	rw	-	rw	rw	rw

Bit	Function
XPER-SHARE	Reserved The XPER-SHARE mode, known from other C16x Infineon derivatives, is **not** supported in the C161U. This bit must be set to ’0’ signal.
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 **Note:** This bit is valid only for derivatives that contain X-Peripherals.
OSCENBL	Oscillator Watchdog Enable Bit ‘0’: The oscillator watchdog is disabled. Default configuration. ‘1’: The oscillator watchdog is enabled.
CSCFG	Chip Select Configuration Control ‘0’: Latched $\overline{CS}$ mode. The $\overline{CS}$ signals are latched internally and driven to the enabled port pins synchronously. ‘1’: Unlatched $\overline{CS}$ mode. The $\overline{CS}$ signals are directly derived from the address and driven to the enabled port pins.
WRCFG	**Write Configuration Control** (Set according to pin P0H.0 during reset) ’0’: Pins $\overline{WR}$ and $\overline{BHE}$ retain their normal function ’1’: Pin $\overline{WR}$ acts as $\overline{WRL}$, pin $\overline{BHE}$ acts as $\overline{WRH}$
CLKEN	**System Clock Output Enable** (CLKOUT) ’0’: CLKOUT disabled: pin may be used for general purpose I/O ’1’: CLKOUT enabled: pin outputs the system clock signal
BYTDIS	**Disable/Enable Control for Pin BHE** (Set according to data bus width) ’0’: Pin $\overline{BHE}$ enabled ’1’: Pin $\overline{BHE}$ disabled, pin may be used for general purpose I/O

Bit	Function
ROMEN	Internal Boot-ROM Enable '0': Internal Boot-ROM is disabled. Access of the lower 32k address space will be linked to external memory. During normal operation, bit ROMEN must always be set to '0' signal '1': Internal Boot-ROM is enabled. This bit is only set in BSL mode. During BSL mode, if the lowest 32k of external memory needs to be programmed, bit ROMEN must be set to '0' signal. After BSL mode, make sure that bit ROMEN is cleared.
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	Reserved The ROMS1, known from other C16x Infineon derivatives, is **not** supported in the C161U. This bit must be set to '0' signal.
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.

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 whose frequency equals the CPU operating frequency ($f_{OUT} = f_{CPU}$).

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

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 C161U. An area of 32...512 words or all of the internal RAM may be dedicated to the system stack. A so-called “circular stack” mechanism allows 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”.

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	HLD EN	-	-	-	USR0	MUL IP	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.

Table 10 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 (ie. 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”.

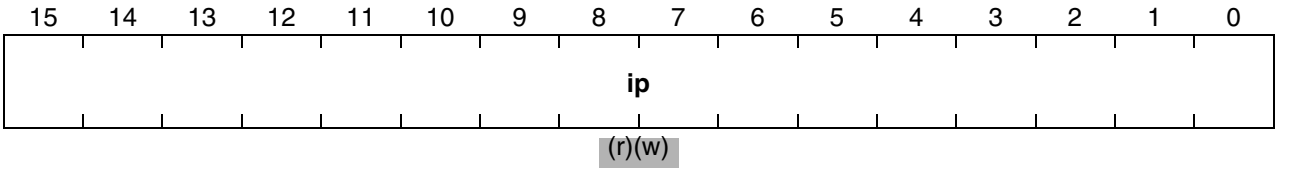
After reset all interrupts are globally disabled, and the lowest priority (ILVL=0) is assigned to the initial CPU activity.

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 C161U'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>.

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.

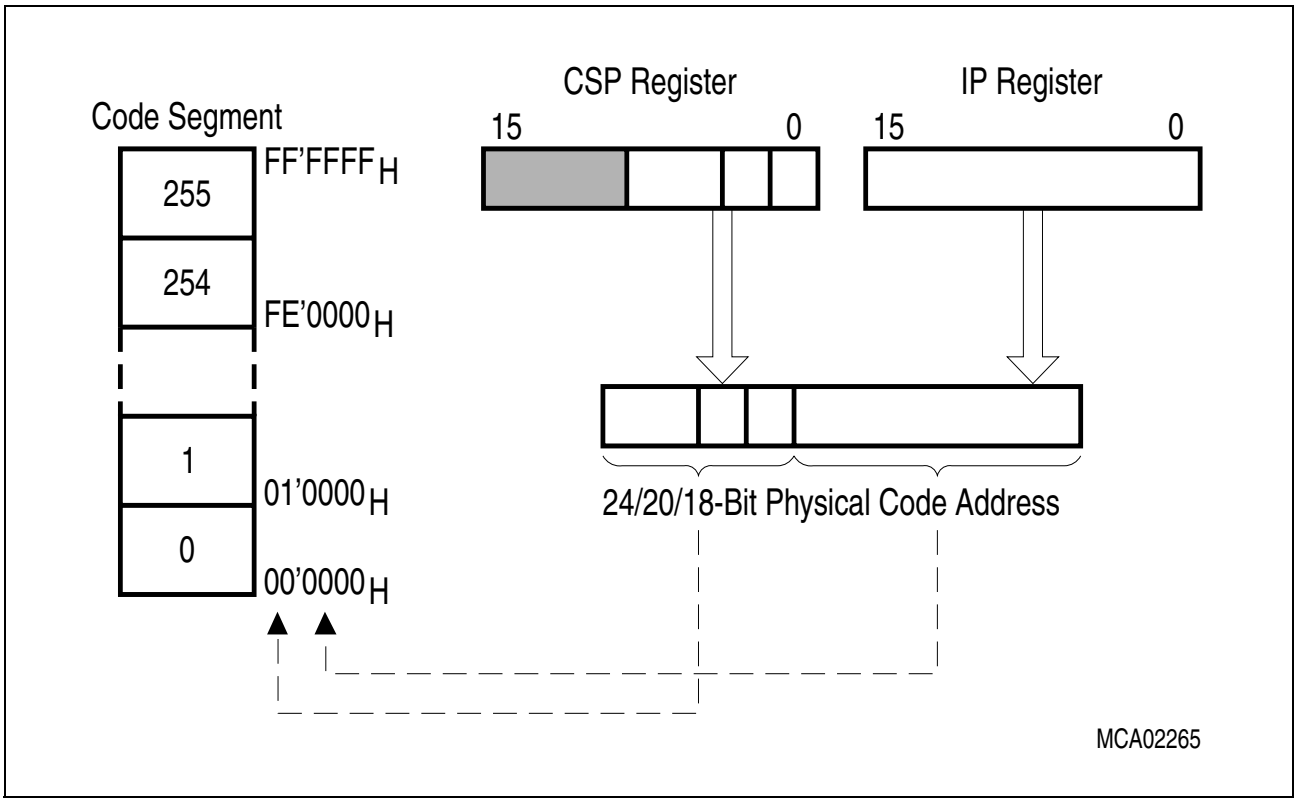


Figure 16 Addressing via the Code Segment Pointer

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

Data Page Pointers DPP0, DPP1, DPP2, DPP3

These four non-bit addressable registers select up to four different data pages 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 content of the selected DPP register specifies 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.

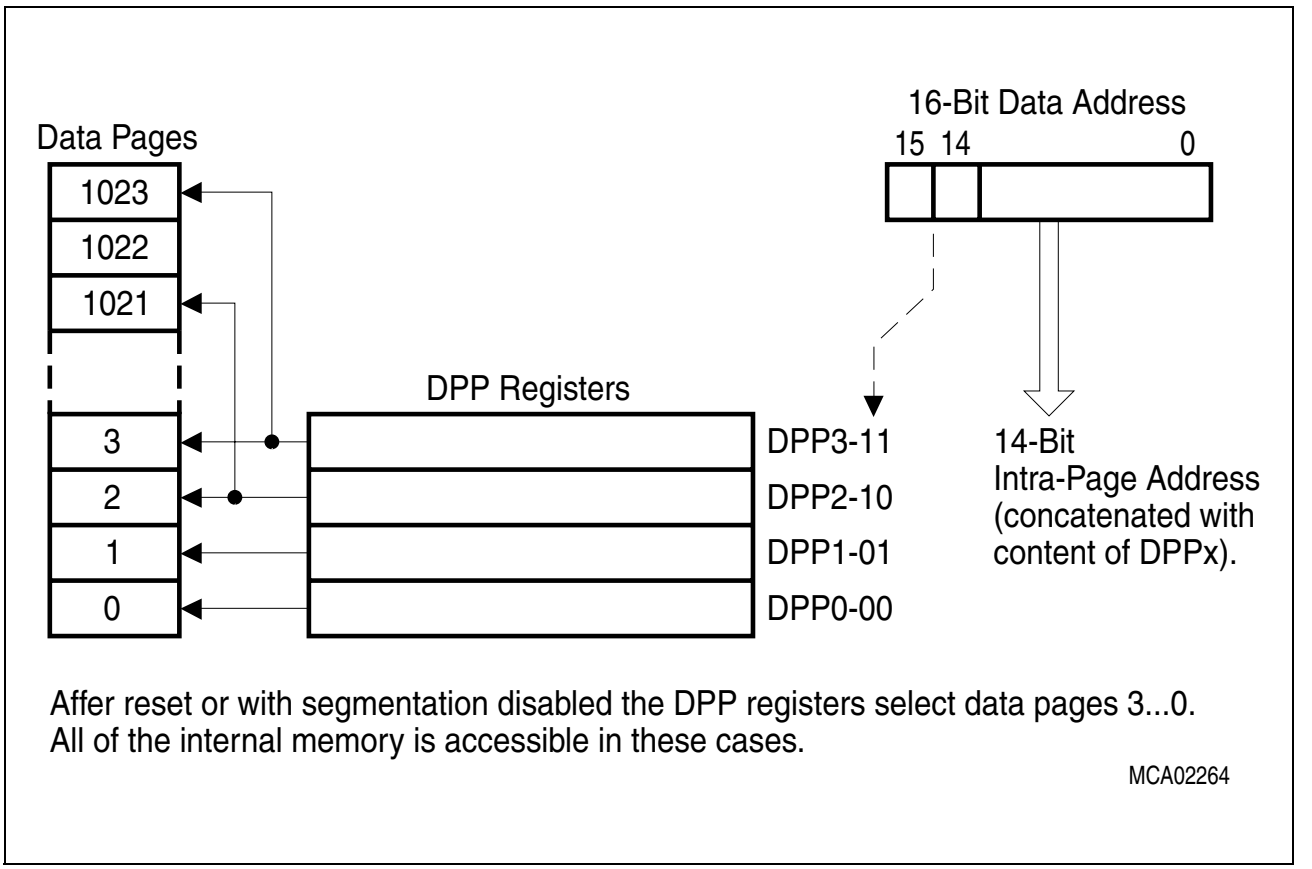


Figure 17 Addressing via the Data Page Pointers

Context Pointer CP

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

CP ($FE10_H$ / 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, ie. $00'FA00_H$/$00'F600_H$/$00'F200_H$ (1/2/3 KB)
- 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.

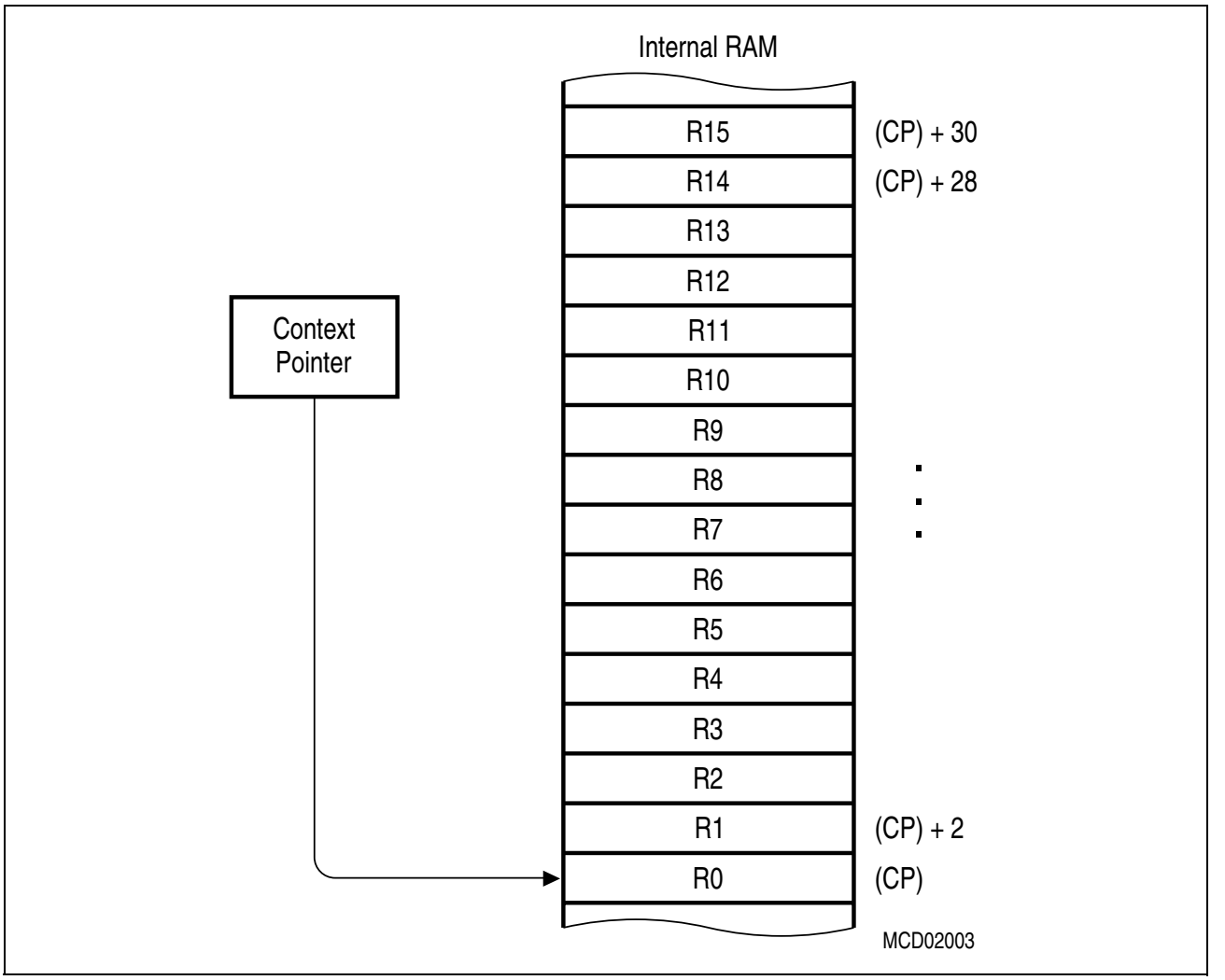


Figure 18 Register Bank Selection via Register CP

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, ie. 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.

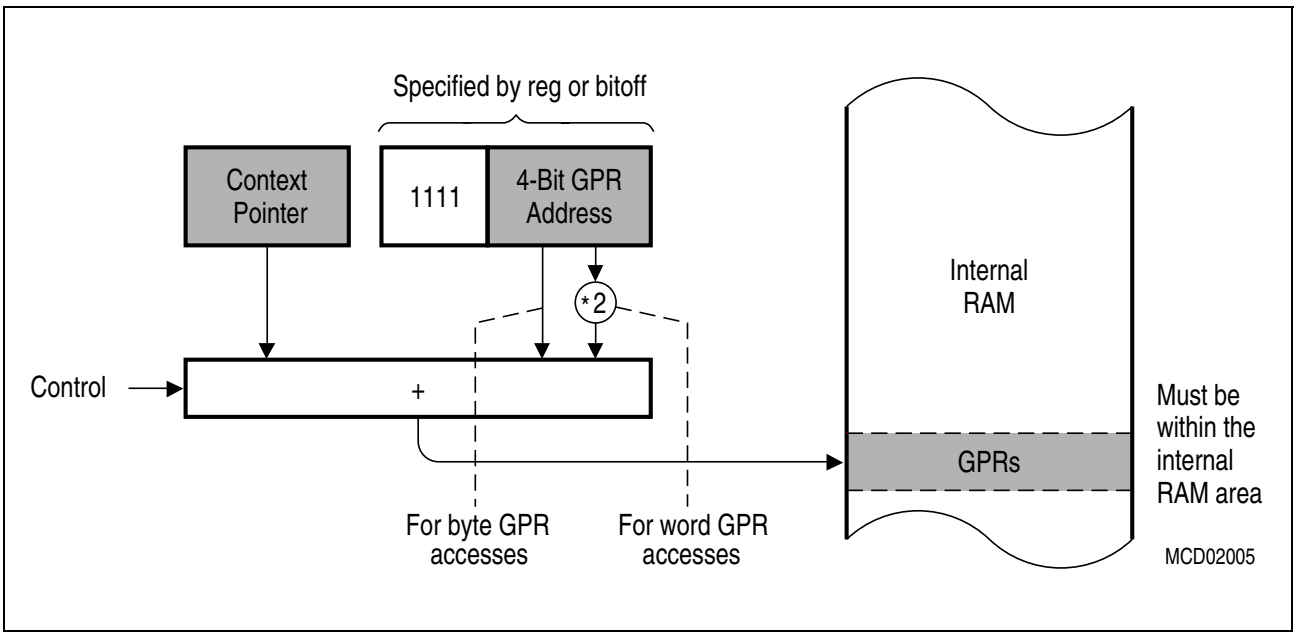


Figure 19 Implicit CP Use by Short GPR Addressing Modes

Stack Pointer SP

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

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 C161U. 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.

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 (eg. 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”.

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 (eg. POP and RET instructions) and after each addition to the SP register. If the content of the SP register is greater than the the content of 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, ie. CALL or RET instructions).

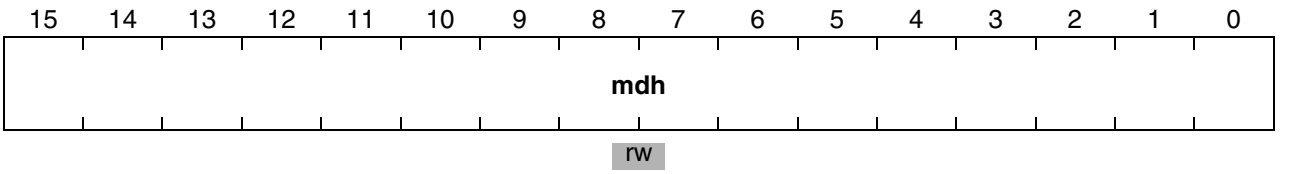
This control mechanism is not triggered, ie. no stack trap is generated, when

- 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.

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**



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'.

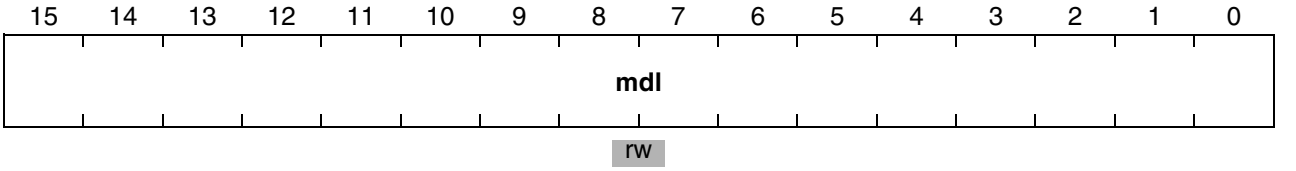
When a multiplication or division is interrupted before its completion and when a new multiply or divide operation is to be performed within the interrupt service routine, register MDH must be saved along with registers MDL and MDC to avoid erroneous results.

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

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**



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”.

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".

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, ie. 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

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, ie. 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

5.5 PEC - Extension of Functionality

Introduction

Compared to existing C16x architecture, the PEC transfer function is enhanced by extended functionality. The extended PEC function is a further step into DMA control functionality. It especially supports integrated system design with XBUS as system bus.

Note: The device address decoding structure is always based on 24-bit addresses. But due to the limited number of port P4 pins, only the address bits A20:A16 can be made visible on the external X-Bus interface.

The extended PEC functions are defined as follows:

– Source pointer and destination pointer are extended to 24-bit pointer, thus enabling PEC controlled data transfer between any two locations within the total address space. Both 8-bit segment numbers of every source/destination pointer pair are defined in one 16-bit SFR register; thus, 8 PEC segment number registers are available for the 8 PEC channels.
– Two of the PEC channels are expanded by additional 16-bit transfer count registers; when enabled, the original 8-bit bytecount in the control register serves as package length count, thus defining the amount of bytes or words to be transferred with one request. In C161U the package size is always limited to one transfer.
– For always two channels a chaining feature is provided. When enabled in the PEC control register, a termination interrupt of one channel will automatically switch transfer control to the other channel of the channel pair.

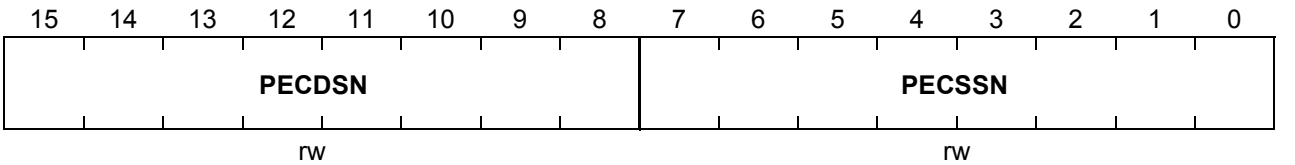
24-bit Extension of Source and Destination Pointers

The source and destination pointers specify the locations between which the data is to be moved. For each of the eight PEC channels the source and destination pointers are specified by one SFR register and two IRAM memory locations. One SFR register stores the 8-bit segment number of the source (PECSSN) and the 8-bit segment number of the destination (PECDSN) location in a respective 16-bit PEC Segment Number register (PECSNx). The respective segment offset of source and destination are stored in IRAM memory location identical to the IRAM locations of SRCPx and DSTPx pointers of Full-Custom C16x standard PEC channels - thus the extension is fully compatible. With the segment number extension of source and destination, data can be transferred by a PEC transfer between any two locations within the 2 MByte address space of the C161U.

Note: The segment number extension of source and destination is provided for all 8 PEC channels. After reset, all 8 segment number registers PECSNx are cleared, providing full compatibility to FC-C16x PEC channels.

The PEC segment number registers PECSNx are defined as follows:

PECSNx (Addresses see table) **SFR** **Reset Value: 0000_H**



Bit	Function
PECSSN	PEC Source Segment Number 8-bit Segment Number used for addressing the source of the respective PEC transfer. Bits 4:0 can be used externally (address bits A20:A16). Due to the limited number of pins, the upper bits 7:5 can not be used externally but can still be used for chip select (CS) generation.
PECDSN	PEC Destination Segment Number 8-bit Segment Number used for addressing the destination of the respective PEC transfer. Bits 12:8 can be used externally (address bits A20:A16). Due to the limited number of pins, the upper bits 15:13 can not be used externally but can still be used for chip select (CS) generation.

Table 11 PEC Segment Number Register Addresses

Register	Address	Reg. Space	Register	Address	Reg. Space
PECSN0	$FED0_H$ / 68_H	SFR	PECSN4	$FED8_H$ / $6C_H$	SFR
PECSN1	$FED2_H$ / 69_H	SFR	PECSN5	$FEDA_H$ / $6D_H$	SFR
PECSN2	$FED4_H$ / $6A_H$	SFR	PECSN6	$FEDC_H$ / $6E_H$	SFR
PECSN3	$FED6_H$ / $6B_H$	SFR	PECSN7	$FEDE_H$ / $6F_H$	SFR

Extended PEC Channel Control

The PEC control registers with the extended functionality and their application for new PEC control are defined as follows:

PECCx (Addresses: see table) **SFR** **Reset Value: 0000_H**

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

Bit	Function
COUNT	PEC Transfer Count Counts PEC transfers (bytes or words) and influences the channel's action
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)
CL	Channel Link Control 0: PEC channels work independent 1: Pairs of channels are linked together
CLT	Channel Link Toggle State 0: Even numbered PEC channel of linked channels active 1: Odd numbered PEC channel of linked channels active
PT	Package Transfer 0: Single Transfer; extended Count2 not enabled 1: Package Transfer; extended Count2 enabled (only for channels 0 and 2) Package Transfer is only supported in PECC0 and PECC2

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, ie. 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, ie. 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.

Channel Link control bit CL controls the channel link mode. In this mode PEC channels work by pair (channels 0 and 1, 2 and 3, 4 and 5, 6 and 7). The channel link mode is enabled for one pair when the CL bit is set in any of the 2 PECCx registers. In this case, the 2 channels handle PEC requests alternative to each other. The whole data transfer is divided into several block transfers where each block is controlled by a PEC channel. When a block transfer is completed, a channel link interrupt is generated and the request processing is switched to the other PEC channel of the pair. This mechanism allows to set up shadow and multiple buffers for PEC transfers by changing pointers and count values of one channel when the other channel is active.

The very first transfer is always initiated with the even channel (called channel A, that is channel 0, 2, 4 or 6). When the associated count field reaches 0 (COUNT or COUNT2 depending on the selected mode), the request service is transfered to the odd channel (channel B, that is channel 1, 3, 5 or 7). If the CL bit of the "linked" channel is set and the count field is different from 0, the next PEC requests will be serviced by this channel.

The channel link interrupts share one common interrupt node (Trap number $4C_H$ - vector location $00'0130_H$). This node is controlled by the Channel Link Interrupt Sub-Node Control (CLISNC) register. It raises an interrupt request in case of one or more channel link request flag and the respective enable control bit is set in CLISNC register. These flags signal a PEC condition of the PEC linked channels which requires an action from the CPU. The following conditions are possible:

1. In single transfer mode, a COUNT value change from 01_H to 00_H in a linked PEC channel and the CL flag is set in the respective PEC control register,

2. In packet transfer mode, a COUNT2 value change from 0001_H to 0000_H in a linked PEC channel and the CL flag is set in the respective PEC control register.

In these cases the CPU is requested to update the PEC control and pointer registers while the next block transfer is executed. The last block transfer is determined by the missing link bit in the linked PEC control register. If a service request hits a linked channel with a COUNT field equal to 00H and the channel link flag disabled, a standard interrupt is performed as known from standard PEC channels.

CLISNC ($FFA8_H$ / $D4_H$) SFR **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	C6 IR	C6 IE	-	-	C4 IR	C4 IE	-	-	C2 IR	C2 IE	-	-	C0 IR	C0 IE
-	-	rw	rw	-	-	rw	rw	-	-	rw	rw	-	-	rw	rw

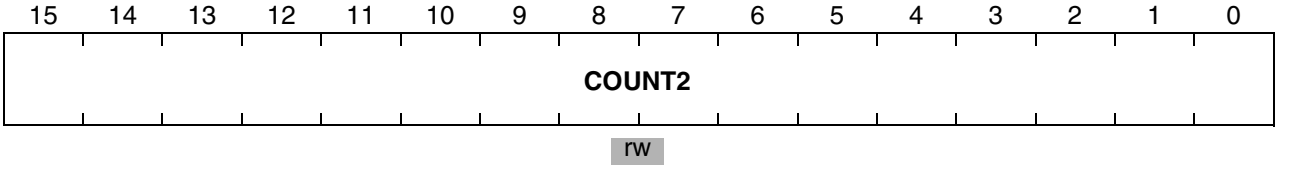
Bit	Function
xxIE	Channel Link Interrupt Enable Bit (individual for each pair of linked channels) '0': Interrupt request disabled '1': Interrupt request enabled
xxIR	Channel Service Request Flag '0': No channel link service request pending '1': The channel pair has raised a request to service a PEC channel after channel link

Packet Transfer control bit PT is implemented only in PECC0 and PECC2. When set to '1', this bit enables the Packet Transfer mode. In this mode, each service request initiates the transfer of an entire data packet of a fixed size. The COUNT field in the PECCx register is used to define the size of the packet (in number of bytes or words depending on the value of BWT). Therefore packets up to 256 bytes/words may be transfered.

The register PECXC0/2 is then used to specify the number of requests to be serviced by a PEC packet transfer before activating the interrupt service routine, which is associated with the priority level. After each PEC packet transfer, the COUNT2 field is decremented and the request flag is cleared, and then when COUNT2 reaches 0000_H, an interrupt request is generated to the corresponding interrupt vector.

Note: In the C161U, the packet size is limited to 1. Packet transfers are not supported, but the extended transfer count COUNT2 is used when PT bit is set.

PECXCx (FEFy$_H$ / 7z$_H$, see table) SFR **Reset Value: 0000$_H$**



Bit	Function
COUNT2	Extended PEC Transfer Count Counts PEC transfers and influences the channel's action in Packet transfer mode

PEC Extended Control Register Addresses

Register	Address	Reg. Space	Register	Address	Reg. Space
PECXC0	FEF0$_H$ / 78$_H$	SFR	PECXC2	FEF2$_H$ / 79$_H$	SFR

Source and destination pointers specifiy the locations between which the data is to be moved. For PEC transfer description refer to **Chapter 7.3**, page 115, where the PEC operation is descriped more in detail.

Channel Link Mode for Data Chaining

Data chaining with linked PEC channels is enabled, if the Channel Link Control Bit in PECCx register is set to '1', either in one or both PEC channel control registers of a channel pair. In this case, two PEC channels are linked together and handle chained block transfers alternatively to each other. The whole data transfer is divided into several block transfers where each block is controlled by one PEC channel of a channel pair. When a data block is completely transferred a **channel link interrupt** is generated and the PEC service request processing is automatically switched to the 'other' PEC channel of the channel-pair. Thus, PEC service requests addressed to a linked PEC channel are either handled by linked PEC channel A or by linked PEC channel B. This channel toggle allows to set up shadow and multiple buffers for PEC transfers by changing pointer and count values of one channel while the other channel is active. The following table list the channels that can be linked together and the channel numbers to address the linked channels.

For each pair of linked channels, an internal channel flag, the Channel Link Toggle flag

Table 12 PEC Channels which could be linked together

Linked PEC Channels		Linked PEC Channel
PEC Channel A	PEC Channel B	
channel 0	channel 1	channel 0
channel 2	channel 3	channel 2
channel 4	channel 5	channel 4
channel 6	channel 7	channel 6

CLT identifies which of the two PEC channels will serve the next PEC request. The CLT flag is indicated in both PECCx registers of two linked PEC channels, where the CLT bit in channel B always is inverse to the CLT bit in channel A. The very first transfer is always started with the channel A if the CLT bit was not programmed otherwise before. The CLT bit is only valid in case of linked PEC channels, indicated by the CL bits of linked channels. If linking is not enabled, the CLT bit of both channels is always zero (compatibility!).

The internal channel link flag CLT toggles, and the other channel begins service with the next request if the "old" channel stops the service (COUNT=0 or COUNT2=0, dependent on the mode), and if the new channel has in its PEC control register the CL flag enabled and its transfer count is more than zero. Note: With the last transfer of a block transfer (COUNT=0 or COUNT2=0), the channel link control flag CL of that channel is cleared in its PECCx register. If the channel link flag CL of the new (chained) PEC control register is found to be zero the whole data transfer is finished and the channel link interrupt is coincidently a termination interrupt. The channel link mode is finished and the internal channel toggle flag is cleared after the last transfer of the block, if the CL flags of both pair channels are cleared.

6 DMA - External PEC (EPEC)

EPEC provides fast and easy means to transfer single data between any memory location within the address space by using the XBUS. The advantages are reduced XBUS protocol handling and capability of addressing all system resources including internal RAM and SFR.

6.1 EPEC Functionality

EPEC provides a DMA controller for the USB device core to provide fast and flexible data tranfer capability. EPEC is implemented as a 16 channel controller with a 24-bit source pointer, a 24-bit destination pointer and a 10-bit Transmit Byte Length Counter with auto-increment support of two bytes (one word) per channel with Terminal Count (TC) indication (Interrupt pulse valid for one clock cycle). After TC is reached, the counter stops itself.

EPEC is connected to the XBUS and to a proprietary 24-bit bus connected directly to the C166 CBC. EPEC has the highest priority among other interrupts and PECs and does not participate in the interrupt priorization round. In case of an DPEC/EPEC collision, the DPEC will get priority and one instruction cycle later the EPEC is processed. EPEC provides DMA like functionality by injecting a memory transfer instruction (mov [dest], [src]) into the decode stage of the pipeline and thus only needs one additional instruction cycle. Even in IDLE mode, the EPEC will be processed waking up the CPU for one instruction cycle and immediately going back to IDLE state.

6.2 EPEC Implementation

EPEC control block is located in the CBC core with its main purpose to synchronize the external EPEC request to the internal T1-T4 states of the CPU and the priorization between DPEC and EPEC. It also drives the externally provided 24-bit source and destination pointer values on the internal memory address bus, thus controlling the whole timing with respect to the CPU.

The EPEC Block diagram is shown in **Figure 20** below.

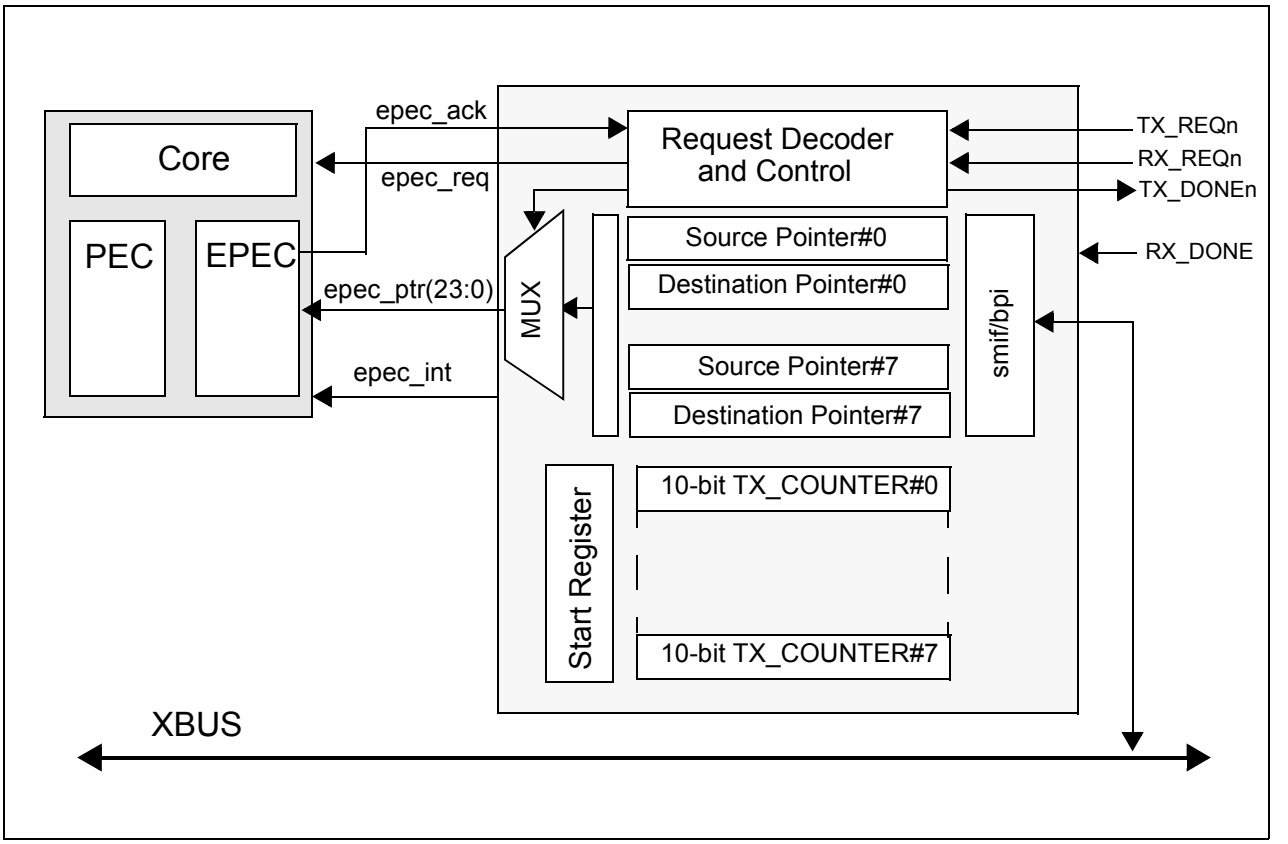


Figure 20 DMA/EPEC Block Diagram

The TX_REQn and RX_REQn shown in **Figure 20** will be generated by the USBD to request a word transfer over the XBUS from/to the FIFOs.

Besides the transfer request interrupts the EPEC provides a 10-bit transmit counter register per channel, which will be written by SW. After the terminal count value is reached, the counter stops and generates and TX_DONE pulse to the USBD. For each endpoint an EPEC_CTRL register is provided to control the endpoint.

6.3 EPEC Register Description

The EPEC register description below shows the individual channel assignments between requesting USB source interrupts and each individual EPEC channel. The EPEC Register Base address is **$00ED00_H$**.

The detailed register description is shown below.

Table 13 EPEC Register Summary

$00ED00_H$+	**Name**	**Function**
00_H	EPECCLC	EPEC Clock Control Register
08_H	EPECID	EPEC Identification Register
10_H	EPEC_SPTR_IN_R00	16 LSBs of USB endpoint#0 source pointer IN
12_H	EPEC_SPTR_IN_R01	8 MSBs of USB endpoint#0 source pointer IN
14_H	EPEC_SPTR_OUT_R00	16 LSBs of USB endpoint#0 source pointer OUT
16_H	EPEC_SPTR_OUT_R01	8 MSBs of USB endpoint#0 source pointer OUT
18_H	EPEC_SPTR_REG10	16 LSBs of USB endpoint#1 source pointer
$1A_H$	EPEC_SPTR_REG11	8 MSBs of USB endpoint#1 source pointer
$1C_H$	EPEC_SPTR_REG20	16 LSBs of USB endpoint#2 source pointer
$1E_H$	EPEC_SPTR_REG21	8 MSBs of USB endpoint#2 source pointer
20_H	EPEC_SPTR_REG30	16 LSBs of USB endpoint#3 source pointer
22_H	EPEC_SPTR_REG31	8 MSBs of USB endpoint#3 source pointer
24_H	EPEC_SPTR_REG40	16 LSBs of USB endpoint#4 source pointer
26_H	EPEC_SPTR_REG41	8 MSBs of USB endpoint#4 source pointer
28_H	EPEC_SPTR_REG50	16 LSBs of USB endpoint#5 source pointer
$2A_H$	EPEC_SPTR_REG51	8 MSBs of USB endpoint#5 source pointer
$2C_H$	EPEC_SPTR_REG60	16 LSBs of USB endpoint#6 source pointer
$2E_H$	EPEC_SPTR_REG61	8 MSBs of USB endpoint#6 source pointer
30_H	EPEC_SPTR_REG70	16 LSBs of USB endpoint#7 source pointer
32_H	EPEC_SPTR_REG71	8 MSBs of USB endpoint#7 source pointer
34_H	EPEC_DPTR_IN_R00	16 LSBs of USB endpoint#0 destination pointer IN
36_H	EPEC_DPTR_IN_R01	8 MSBs of USB endpoint#0 destination pointer IN

Table 13 EPEC Register Summary (cont'd)

$00ED00_H$+	Name	Function
38_H	EPEC_DPTR_OUT_R00	16 LSBs of USB endpoint#0 destination pointer OUT
$3A_H$	EPEC_DPTR_OUT_R01	8 MSBs of USB endpoint#0 destination pointer OUT
$3C_H$	EPEC_DPTR_REG10	16 LSBs of USB endpoint#1 destination pointer
$3E_H$	EPEC_DPTR_REG11	8 MSBs of USB endpoint#1 destination pointer
40_H	EPEC_DPTR_REG20	16 LSBs of USB endpoint#2 destination pointer
42_H	EPEC_DPTR_REG21	8 MSBs of USB endpoint#2 destination pointer
44_H	EPEC_DPTR_REG30	16 LSBs of USB endpoint#3 destination pointer
46_H	EPEC_DPTR_REG31	8 MSBs of USB endpoint#3 destination pointer
48_H	EPEC_DPTR_REG40	16 LSBs of USB endpoint#4 destination pointer
$4A_H$	EPEC_DPTR_REG41	8 MSBs of USB endpoint#4 destination pointer
$4C_H$	EPEC_DPTR_REG50	16 LSBs of USB endpoint#5 destination pointer
$4E_H$	EPEC_DPTR_REG51	8 MSBs of USB endpoint#5 destination pointer
50_H	EPEC_DPTR_REG60	16 LSBs of USB endpoint#6 destination pointer
52_H	EPEC_DPTR_REG61	8 MSBs of USB endpoint#6 destination pointer
54_H	EPEC_DPTR_REG70	16 LSBs of USB endpoint#7 destination pointer
56_H	EPEC_DPTR_REG71	8 MSBs of USB endpoint#7 destination pointer
58_H	EPEC_CTRL_IN_R0	Control and Status register for USB endpoint#0 IN
$5A_H$	EPEC_CTRL_OUT_R0	Control and Status register for USB endpoint#0 OUT
$5C_H$	EPEC_CTRL_REG1	Control and Status register for USB endpoint#1
$5E_H$	EPEC_CTRL_REG2	Control and Status register for USB endpoint#2

Table 13 EPEC Register Summary (cont'd)

$00ED00_H$+	**Name**	**Function**
60_H	EPEC_CTRL_REG3	Control and Status register for USB endpoint#3
62_H	EPEC_CTRL_REG4	Control and Status register for USB endpoint#4
64_H	EPEC_CTRL_REG5	Control and Status register for USB endpoint#5
66_H	EPEC_CTRL_REG6	Control and Status register for USB endpoint#6
68_H	EPEC_CTRL_REG7	Control and Status register for USB endpoint#7
$6A_H$	EPEC_INT_REG	EPEC Interrupt
$6C_H$	EPEC_INTMSK_REG	EPEC Interrupt Mask Register
$6E..FF_H$		These registers are all reserved

EPEC Clock Control Register

Address: $ED00_H$
Name: **EPECCLC**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												EPEC EX_DIS	EPEC GPSEN	EPEC DIS	EPEC DISR

Field	Bits	Type	Value	Description
EPECEX_DIS	3	R/W	0	**EPEC Controller Clock Disable** 0: The clock of the EPEC interface controller can be switched off using the SYSCON register. 1: The clock of the EPEC interface controller can NOT be switched off using the SYSCON registers.
EPECGPSEN	2	R/W	0	**EPEC Controller Clock OCDS Disable** 0: The clock of the EPEC interface controller is enabled, normal operation. 1: The clock of the EPEC interface controller is disabled during debugging mode (OCDS)
EPECDIS	1	R	0	**EPEC Controller Clock Status** 0: The status of the EPEC interface controller clock is ’enabled’. 1: The status of the EPEC interface controller clock is ’disabled’.
EPECDISR	0	R/W	0	**EPEC Controller Clock Disable** 0: The clock of the EPEC interface controller is enabled, normal operation. 1: The clock of the EPEC interface controller is disabled.
RESERVED	15:4	-	0	These bits are reserved

The register EPECCLC is clocked with the bus clock to be able to switch the EPEC interface controller clock on again, if it was off. If required, switching off the clock can be prevented by the EPEC controller.

The state of the EPEC interface controller clock is controlled by the register bit EPECDISR. The actual clock state will be shown by the state bit EPECDIS.

For on chip debugging support (OCDS) an additional bit EPECGPSEN is introduced to stop the peripheral clock for arbitrary lengths of time during debugging if this function is enabled. If debugging mode is active, the peripheral core rejects write access to registers connected to the peripheral clock.

To be compatible with previous C16x products an EPECEX_DISR signal is provided to disable the peripheral clock.

EPEC Identification Register

Address: $ED08_H$
Name: EPECID

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

Field	Bits	Type	Value	Description
ID	15:0	R	0	EPEC Identification Register. **Note:** The value of the EPECID register is hardwired to ZERO in the C161U.

EPEC_SPTR_REGx0 (x=7..0) Reset Value: 0000_H

Table 14 EPEC_SPTR_REGx0 Source Pointer Register

Bit No.	Name	Function
15:0	SPTRx(15:0)	16 LSBs of USB endpoint#x source pointer

The EPEC source pointer registers (x0) provide the least significant 16-bits of the 24-bit source pointer address for USB endpoints

EPEC_SPTR_REGx1 (x=7..0) Reset Value: 0000_H

Table 15 EPEC_SPTR_REGx1 Source Pointer Register

Bit No.	Name	Function
15:8	reserved	always 00_H
7:0	SPTRx(23:16)	8 MSBs of USB endpoint#x source pointer

The EPEC source pointer registers (x1) provide the most significant 8-bits of the 24-bit source pointer for USB endpoints.

EPEC_DPTR_REGx0 (x=7..0) Reset Value: 0000_H

Table 16 EPEC_DPTR_REGx0 Destination Pointer Register

Bit No.	Name	Function
15:0	DPTRx(15:0)	16 LSBs of USB endpoint#x destination pointer

The EPEC destination pointer registers (x0) provide the least significant 16-bits of the 24-bit destination Pointer address for USB endpoints.

EPEC_DPTR_REGx1 (x=7..0) Reset Value: 0000_H

Table 17 EPEC_DPTR_REG01 Destination Pointer Register

Bit No.	Name	Function
15:8	reserved	always 00_H
7:0	DPTRx(23:16)	8 MSBs of USB endpoint#x source pointer

The EPEC destination pointer registers (x1) provide the most significant 8-bits of the 24-bit destination pointer for USB endpoints.

EPEC_CTRL_REGx (x=7..0) Reset Value: 0000_H

Table 18 EPEC_CTRL_REGx Source Pointer Register

Bit No.	Name	Function
15	TXR_ENAx	Transfer / Receive Enable control bit, set by SW and cleared by EPEC after transfer complete ’1’: Transmitter / Receiver enabled ’0’: Transmitter / Receiver disabled
14	EXT_SRC	External Source ’1’: Reserved (program memory) ’0’: External source is selected EXT_SRC must be set to ’0’

Table 18 EPEC_CTRL_REGx Source Pointer Register

Bit No.	Name	Function
13:12	REQ_SRC	EPEC request sourceRX/TX Fifo ’10’: EPEC is connected to TX Fifo Request (IN) ’01’: EPEC is connected to RX Fifo Request (OUT) ’11’: EPEC is connected to RX and TX Fifo Request (BI) for Control endpoint 0. ’00’: reserved
11	CNT_UP_DN	Byte Counter direction select ’1’: Rx ’0’: Tx
10	CLR	Clear EPEC channel ’1’: Clears EPEC channel settings into idle mode ’0’: no action
9:0	BYTE_CNT	Number of bytes to be transmitted

The EPEC Transmit Byte Length registers (x) provide the 10-bit Transmit bytes length of the actual packet to be send to the USB endpoint#x and the related bit. Each EPEC channel can be cleared by SW, if necessary.

All USB source and destination pointers will be used in either receive or transmit direction, since the USB endpoint’s direction of data is SW-configurable as IN, OUT or bidirectional (except endpoint 0) after USB device controller reset.

EPEC_INT_REG Reset Value: 0000_H

Table 19 EPEC_INT_REG Interrupt Register

Bit No.	Name	Function
15:8	RxTxSTART	Rx / Tx Start Signal ’1’ indicates channel has started transfering data.
7:0	TXDONE_INTx (x=7..0)	TX packet transfer completed by EPEC ’1’: transfer complete ’0’: busy or idle

The EPEC interrupt register indicates the end of an TX-packet transfer for an USB endpoint.

EPEC_INTMSK_REG Reset Value: 0000_H

Table 20 EPEC_INTMSK_REG Interrupt Register

Bit No.	Name	Function
15:8	RxTxSTARTMSK	Rx / Tx Start Mask '1': masked '0': not masked
7:0	TXDONE_INTMSKx (x=7..0)	Mask interrupt TX packet transfer completed by EPEC '1': masked '0': not masked

The EPEC interrupt mask register masks out the end of an TX-packet transfer interrupt for an USB endpoint.

6.4 EPEC Transfer Example

The EPEC (external peripheral event controller, external in the sense that it is external to the CPU block) controls the transfer of data between the USB block and the external or internal RAM.

Note: If the EPEC is not enabled, then no transfer is possible.

The sequence of operations is as follows:

1. The USB block will generate FIFO request signals as soon as the system reset was deasserted. It thus signals to the EPEC that the USB FIFOs are ready to receive data.
2. Now the USB has to be configured. The EPEC channel that serves USB endpoint_0_IN is setup with source and destination pointer, the EPEC control register is programmed with the number of bytes that need to be transfered and the bit for external/ internal source has to be set according to the application.
3. Now the EPEC channel for endpoint_0_IN can be activated by setting the enable bit in EPEC control register for endpoint_0_IN.
4. The transfer of configuration data to the USB FIFO for endpoint_0_IN starts because the USB FIFO has signaled that there is space available and the EPEC channel has been enabled.
5. After the EPEC byte counter has reached the number of bytes that have to be transfered, the EPEC channel disables itself and indicates the transmit end of data condition in the EPEC interrupt register.
6. The indication of the transmit end of data condition triggers the generation of the EPEC interrupt pulse to the CPU on the irq(40) line.

7. After the interrupt generation unit has generated the interrupt pulse it waits for a write to the interrupt register. It then is ready to generate the next interrupt pulse to the CPU. If no write to the interrupt register takes place, no new interrupt pulse can be asserted.

6.5 Implementation of EPEC Interrupt Generation Unit

Currently the EPEC interrupt controller implements a clear on write functionality.

This implies the following for the interrupt routine:

1. One of two conditions generates an entry into the EPEC interrupt register: either a channel start or a channel transmit end of data (for endpoint 0 this functionality is reduced to start for direction in, EPEC transmit; and end for direction out, EPEC receive; otherwise we would have more than 16 interrupts). The generation of the entries into the EPEC interrupt register can be controled by programming the EPEC interrupt mask register.
2. The interrupt routine is triggered by the interrupt pulse that the EPEC interrupt controller gernerates on irq(40).
3. The routine should then read the EPEC interrupt register to determine the source of the interrupt.
4. The interrupt has to be acknowledged by writing a ‘1’ to the position of the interrupt source in the EPEC interrupt register.
5. Now the interrupt controller is ready to generate the next interrupt pulse.

7 Interrupt and Trap Functions

The architecture of the C161U 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 C161U'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 C161U 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.

7.1 Interrupt System Structure

C161U provides up to 64 separate interrupt nodes that may be assigned to 16 priority levels. The 4 lowest nodes are reserved for the CPU - thus, up to 60 nodes are available for all interrupts. In order to support modular and consistent software design techniques, each source of an interrupt or PEC request is 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 activated by one specific event, depending on the selected operating mode of the respective device. The only exceptions are the two serial channels of the table, where an error interrupt request can be generated by different kinds of error, and the two subnode interrupts controlled by the ISNC and CLISNC registers (see Interrupt and PEC descriptions). However, specific status flags which identify the type of error are implemented in the serial channels' control registers.

The C161U 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 C161U'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 C161U, the associated interrupt vectors, their locations, their trap numbers and the SFR addresses of associated interrupt control registers. It also lists the mnemonics of the corresponding Interrupt Enable flags. The mnemonics are composed of a part that specifies the respective source, followed by a part that specifies their function (IE=Interrupt Enable flag). The same composition is used for the mnemonics of according interrupt request flags (IR=Interrupt Request flag; example: CC0IR belongs to interrupt source CC0INT) and for the names of according interrupt control registers (IC=Interrupt Control; example: CC0IC) which are not included in **Table 21**.

Table 21 C161U Interrupts and PEC Service Requests

Nr.	Source of Interrupt or PEC Service Request	Interrupt Name	Enable Flag	Vector Location	Trap Number	SFR hex Address
irq(0)	GPT Timer 2	T2INT	T2IE	$00'0088_H$	22_H / 34_D	FF60
irq(1)	GPT Timer 3	T3INT	T3IE	$00'008C_H$	23_H / 35_D	FF62
irq(2)	GPT Timer 4	T4INT	T4IE	$00'0090_H$	24_H / 36_D	FF64
irq(3)	GPT Timer 5	T5INT	T5IE	$00'0094_H$	25_H / 37_D	FF66
irq(4)	GPT Timer 6	T6INT	T6IE	$00'0098_H$	26_H / 38_D	FF68
irq(5)	GPT CAPREL Register	CRINT	CRIE	$00'009C_H$	27_H / 39_D	FF6A
irq(6)	ASC Transmit	S0TINT	S0TIE	$00'00A8_H$	$2A_H$ / 42_D	FF6C
irq(7)	ASC Receive	S0RINT	S0RIE	$00'00AC_H$	$2B_H$ / 43_D	FF6E
irq(8)	ASC Error	S0EINT	S0EIE	$00'00B0_H$	$2C_H$ / 44_D	FF70
irq(9)	ASC Transmit Buffer	S0TBINT	S0TBIE	$00'011C_H$	47_H / 71_D	F19C
irq(10)	SSC Transmit	SSCTINT	SSCTIE	$00'00B4_H$	$2D_H$ / 45_D	FF72
irq(11)	SSC Receive	SSCRINT	SSCRIE	$00'00B8_H$	$2E_H$ / 46_D	FF74
irq(12)	SSC Error	SSCEINT	SSCEIE	$00'00BC_H$	$2F_H$ / 47_D	FF76
irq(13)	ASC Autobaud Start	ABSTINT	ABSTIE	$00'0118_H$	46_H / 70_D	F194
irq(14)	ASC Autobaud End	ABENDINT	ABENDIE	$00'0114_H$	45_H / 69_D	F18C
irq(15)	rRTC Interrupt	RTC_INT	RTCIE	$00'0110_H$	44_H / 68_D	F184
irq(16)	UDC SETUP	USETINT	USETIE	$00'00F0_H$	$3C_H$ / 60_D	F178
irq(17)	UDC Load Config Done	ULCDINT	ULCDIE	$00'00EC_H$	$3B_H$ / 59_D	F176
irq(18)	UDC Suspend	USSINT	USSIE	$00'00E8_H$	$3A_H$ / 58_D	F174
irq(19)	UDC Suspend off	USSOINT	USSOIE	$00'00E4_H$	39_H / 57_D	F172
irq(20)	UDC Start of Frame	USOFINT	USOFIE	$00'00E0_H$	38_H / 56_D	F170
irq(21)	UDC Config Val	UCFGVINT	UCFGVIE	$00'00DC_H$	37_H / 55_D	F16E
irq(22)	UDC TXWR	UTXRINT	UTXRIE	$00'00D8_H$	36_H / 54_D	F16C
irq(23)	UDC RXRR	URXRINT	URXRIE	$00'00D4_H$	35_H / 53_D	F16A
irq(24)	UDC TX Done7	UTD7INT	UTD7IE	$00'00D0_H$	34_H / 52_D	F168
irq(25)	UDC TX Done6	UTD6INT	UTD6IE	$00'00CC_H$	33_H / 51_D	F166

Nr.	Source of Interrupt or PEC Service Request	Interrupt Name	Enable Flag	Vector Location	Trap Number	SFR hex Address
irq(26)	UDC TX Done5	UTD5INT	UTD5IE	$00'00C8_H$	32_H / 50_D	F164
irq(27)	UDC TX Done4	UTD4INT	UTD4IE	00'00C4H	31H / 49D	F162
irq(28)	UDC TX Done3	UTD3INT	UTD3IE	00'00C0H	30H / 48D	F160
irq(29)	UDC TX Done2	UTD2INT	UTD2IE	00'005CH	17H / 23D	FF86
irq(30)	UDC TX Done1	UTD1INT	UTD1IE	00'0058H	16H / 22D	FF84
irq(31)	UDC TX Done0	UTD0INT	UTD0IE	00'0054H	15H / 21D	FF82
irq(32)	UDC RX Done7	URD7INT	URD7IE	00'0050H	14H / 20D	FF80
irq(33)	UDC RX Done6	URD6INT	URD6IE	00'004CH	13H / 19D	FF7E
irq(34)	UDC RX Done5	URD5INT	URD5IE	00'0048H	12H / 18D	FF7C
irq(35)	UDC RX Done4	URD4INT	URD4IE	00'0044H	11H / 17D	FF7A
irq(36)	UDC RX Done3	URD3INT	URD3IE	00'0040H	10H / 16D	FF78
irq(37)	UDC RX Done2	URD2INT	URD2IE	00'0080H	20H / 32D	FF9C
irq(38)	UDC RX Done1	URD1INT	URD1IE	00'0084H	21H / 33D	FF9E
irq(39)	UDC RX Done0	URD0INT	URD0IE	00'00F4H	3DH / 61D	F17A
irq(40)	EPEC	EPECINT	EPECIE	00'00F8H	3EH / 62D	F17C
irq(41)	reserved			$00'00A0_H$	28_H / 40_D	FF98
firq(0)	Fast ext. Interrupt	EX0INT	EX0IE	$00'0060_H$	18_H / 24_D	FF88
firq(1)	Fast ext. Interrupt	EX1INT	EX1IE	$00'0064_H$	19_H / 25_D	FF8A
firq(2)	Fast ext. Interrupt	EX2INT	EX2IE	$00'0068_H$	$1A_H$ / 26_D	FF8C
firq(3)	Fast ext. Interrupt	EX3INT	EX3IE	$00'006C_H$	$1B_H$ / 27_D	FF8E
firq(4)	Fast ext. Interrupt	EX4INT	EX4IE	$00'0070_H$	$1C_H$ / 28_D	FF90
firq(5)	Fast ext. Interrupt	EX5INT	EX5IE	$00'0074_H$	$1D_H$ / 29_D	FF92
firq(6)	Fast ext. Interrupt	EX6INT	EX6IE	$00'0078_H$	$1E_H$ / 30_D	FF94
firq(7)	Fast ext. Interrupt	EX7INT	EX7IE	$00'007C_H$	$1F_H$ / 31_D	FF96
xb(0)	UDC TXWR	UTXRINT	UTXRIE	$00'0100_H$	40_H / 64_D	F186
xb(1)	EPEC	EPECINT	EPECIE	$00'0104_H$	41_H / 65_D	F18E
xb(3)	internal PLL Lock / RTC	XP3INT	XP3IE	$00'010C_H$	43_H / 67_D	F19E
	CLISN Interrupt	CLISNINT	CLISNIE	$00'0130_H$	$4C_H$ / 76_D	FFA8

Note: The X-Bus interrupts xb(0) and xb(1), known from C16x device's, are connected to the main interrupt node of the respective X-Bus peripheral: UTXRINT (xb(0) and irq(22)) and EPECINT (xb(1) and irq(40).

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

Note: One interrupt control register is provided for each interrupt node. All IC registers of the C161U can be found in the SFR list.

Table 22 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.

Table 22 Hardware Traps and Vector Locations

Exception Condition	Trap Flag	Trap Vector	Vector Location	Trap Number	Trap Prio.
Reset Functions:					
Hardware Reset		RESET	$00'0000_H$	00_H	III
Software Reset		RESET	$00'0000_H$	00_H	III
Watchdog Timer Overflow		RESET	$00'0000_H$	00_H	III
Class A Hardware Traps:					
Non-Maskable Interrupt	NMI	NMITRAP	$00'0008_H$	02_H	II
Stack Overflow	STKOF	STOTRAP	$00'0010_H$	04_H	II
Stack Underflow	STKUF	STUTRAP	$00'0018_H$	06_H	II
Debug Trap	DEBUG	DEBTRAP	$00'0020_H$	08_H	II

Exception Condition	Trap Flag	Trap Vector	Vector Location	Trap Number	Trap Prio.
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 Prio.

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.

7.2 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

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 C161U 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 (ie, 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 C161U'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.

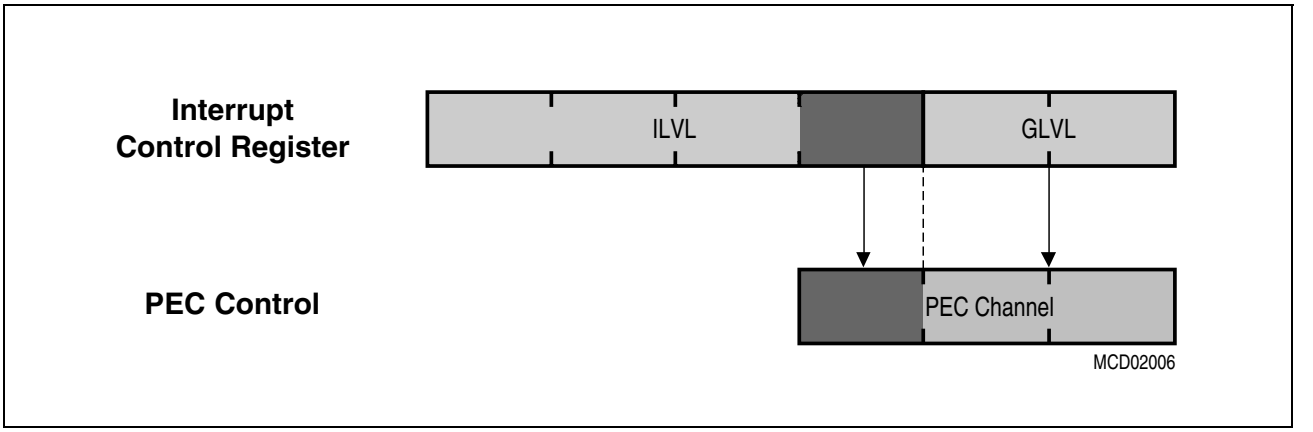


Figure 21 Priority Levels and PEC Channels

Simultaneous requests for PEC channels are prioritized according to the PEC channel number, where channel 0 has lowest and channel 7 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.

Table 23 Programming Example

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 C161U 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 I/O 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 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 (ie. 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 interrupt requests are accepted by the CPU. 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.

7.3 Operation of the PEC Channels

C161U's Peripheral Event Controller (PEC) provides 8 PEC service channels, which move a single byte or word. This is the fastest possible interrupt response and in many cases is sufficient to service the respective peripheral request (eg. 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.

Note: For the PECCx register description, please also refer to page 88 of Sub-Chapter "Extended PEC Channel Control".

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, ie. 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, ie. 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.

Source and destination pointers specifiy 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 C161U just below the bit-addressable area (see figure below).

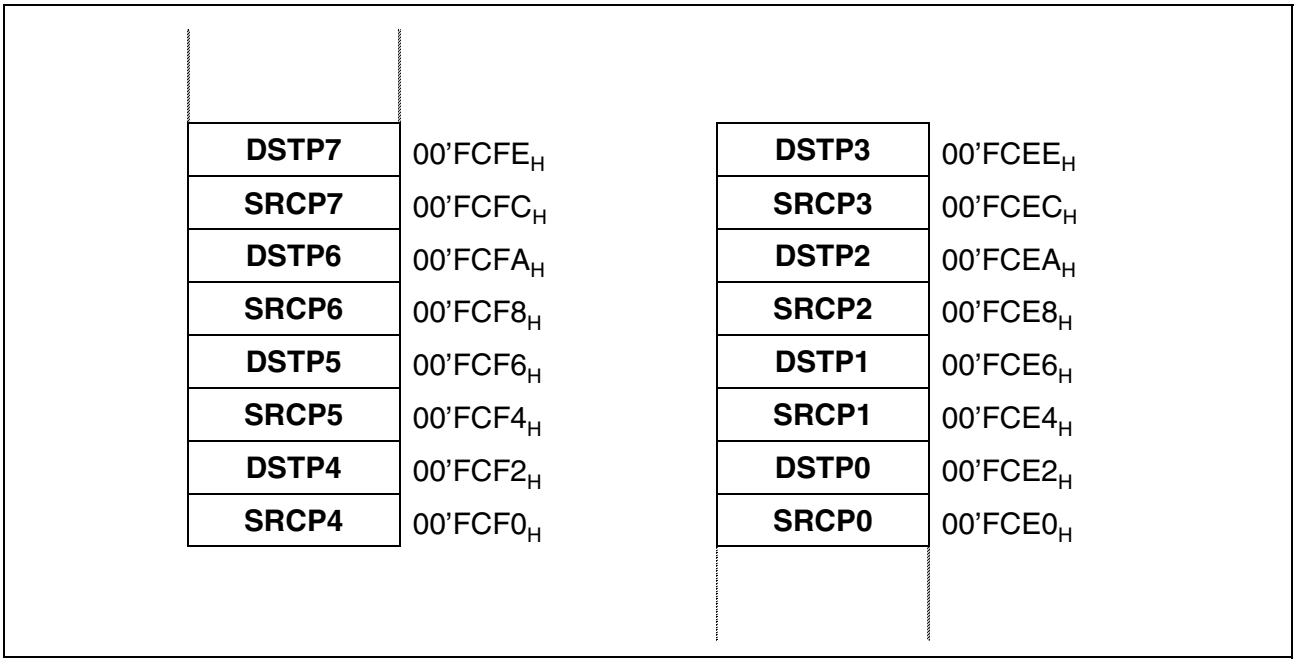


Figure 22 Mapping of PEC Pointers into the Internal RAM

PEC data transfers do not use the data page pointers DPP3...DPP0, see also Chapter 5.5, "PEC - Extension of Functionality". 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 (ie. 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.

7.4 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.

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.

ATOMIC and EXTend instructions automatically disable all interrupt requests for the duration of the following 1...4 instructions. This is useful eg. 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, ie. the same priority from the system’s viewpoint. Interrupts of the same class must not interrupt each other. C161U 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, ie. 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 19 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.

Table 24 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!

7.5 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.

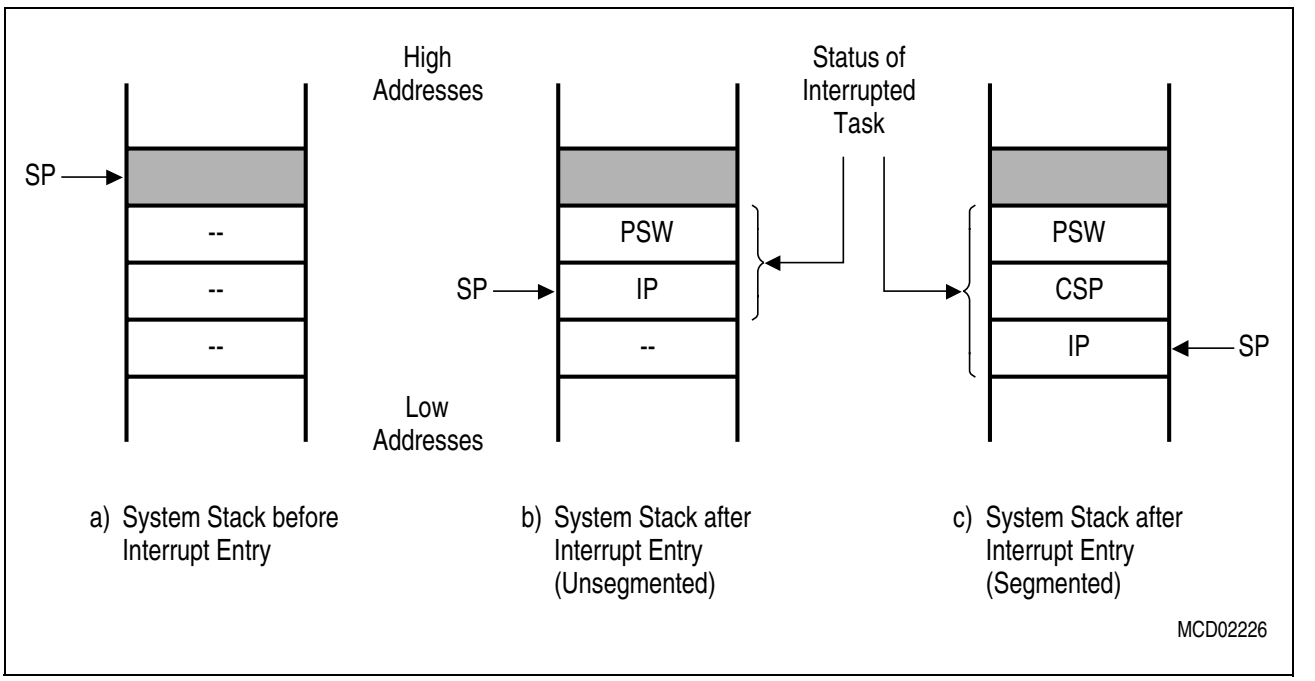


Figure 23 Task Status saved on the System Stack

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. C161U allows to switch the complete bank of CPU registers (GPRs) with a single instruction, so the service routine executes within its own, separate context.

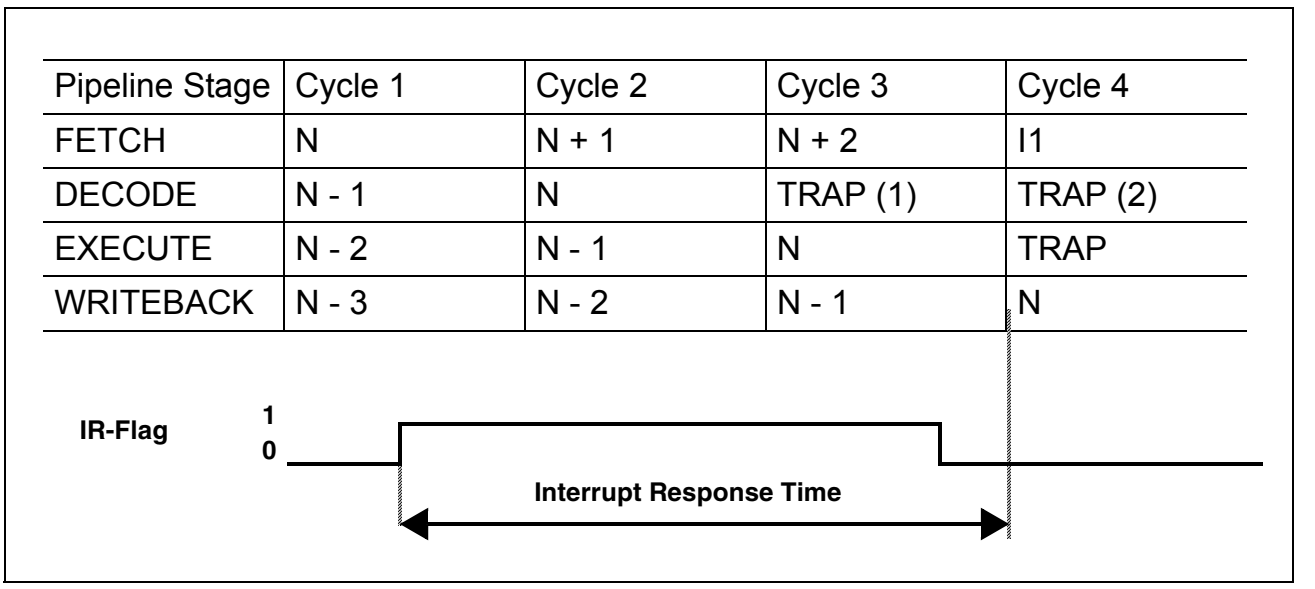
The instruction “SCXT CP, #New_Bank” pushes the content of the context pointer (CP) on the system stack and loads CP with the immediate value “New_Bank”, which selects a new register bank. The service routine may now use its “own registers”. This register bank is preserved, when the service routine terminates, ie. 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, eg. the DPPs and the registers of the MUL/DIV unit.

7.6 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 C161U 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 24 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 (eg. 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.

7.7 PEC Response Times

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 C161U 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 25 Pipeline Diagram for PEC Response Time

In **Figure 25** 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.

7.8 External Interrupts

Although the C161U has no dedicated INTR input pins, it provides many possibilities to react on external asynchronous events by using a number of I/O lines for interrupt input. The interrupt function may either be combined with the pin's main function or may be used instead of it, ie. if the main pin function is not required.

Interrupt signals may be connected to:

- EX1IN...EX0IN, the fast external interrupt input pins,
- T4IN, the timer input pin,

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.

Table 25 Pins to be used as External Interrupt Inputs

Port Pin	Original Function	Control Register
P2.1-0/EX1-0IN	Fast external interrupt input pin	EXICON
P3.5/T4IN	Auxiliary timer T4 input pin	T4CON

Pin T4IN can be used as external interrupt input pin when the associated auxiliary timer T4 in block GPT1 is configured for capture mode. This mode is selected by programming the mode control field T4M in control register T4CON to 101_B. The active edge of the external input signal is determined by bit field T4I. When these field are programmed to $X01_B$, interrupt request flag T4IR in register T4IC will be set on a positive external transition at pin T4IN. When T4I is programmed to $X10_B$, then a negative external transition will set the corresponding request flag. When T4I is programmed to $X11_B$, both a positive and a negative transition will set the request flag. In all three cases, the contents of the core timer T3 will be captured into the auxiliary timer register T4 based on the transition at pin T4IN. When the interrupt enable bit T4IE are set, a PEC request or an interrupt request for vector T4INT 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.

7.8.1 Fast External Interrupts

The input pins that may be used for external interrupts are sampled every 16 TCL, ie. external events are scanned and detected in timeframes of 16 TCL. C161U provides 2 external interrupt inputs that are sampled every 2 TCL, so external events are captured faster than with standard interrupt inputs.

The pins of Port 2 (P2.1...P2.0) 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:

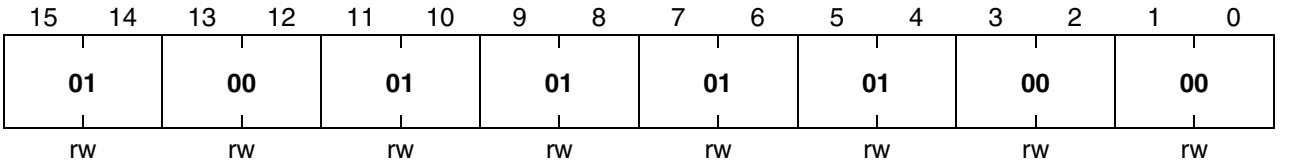
1. Although the C161U provides only two pins of Port 2 which can be used for fast external interrupts, bit fields EXI7ES..EXI2ES must be set in order to use alternate sources as fast external interrupts. For alternate sources, refer to register EXISEL on page 128.
2. The fast external interrupt inputs are sampled every 2 TCL. The interrupt request arbitration and processing, however, is executed every 8 TCL.
3. In Sleep mode, no clock is available. Therefore sampling is performed with asynchronous structures.
4. In Sleep mode fast external interrupts as well as the NMI input are controlled for spike suppression in the System Control Block. Input signals shorter than 10 ns are suppressed, detection is guaranteed for minimum 150 ns input signals.

7.8.2 External Interrupt Source Control

Fast external interrupts may also have interrupt sources selected from other peripherals. This function is very advantageous in Slow Down mode or in Sleep mode, if for example the SSC interface shall be used to wake-up the system. The register EXISEL is used to switch the receive inputs of the serial interfaces to the fast external interrupts, in order to detect incomming messages in case of disabled serial interface modules.

EXISEL register is defined as follows:

EXISEL ($F1DA_H$ / ED_H) **ESFR-bReset Value: 0000_H**

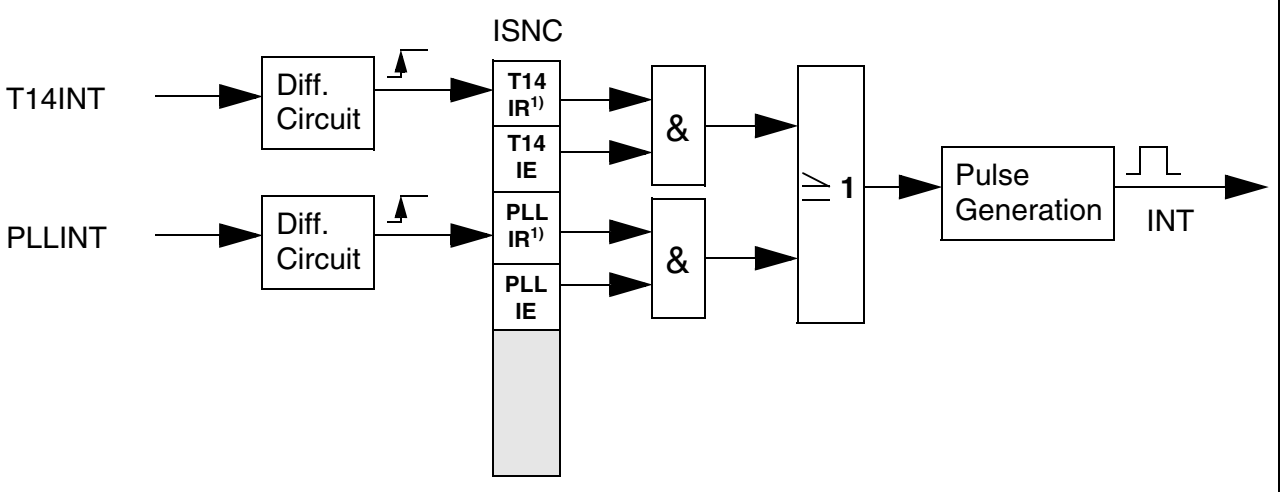


Bit	Function
EXI0SS	0 0: Must be set to ’00’. 0 1: Not allowed. 1 0: Not allowed. 1 1: Not allowed.
EXI1SS	0 0: Must be set to ’00’. 0 1: Not allowed. 1 0: Not allowed. 1 1: Not allowed.
EXI2SS	0 0: Not allowed. 0 1: Input from source ASC_RxD @ P3.11. 1 0: Not allowed. 1 1: Not allowed.
EXI3SS	0 0: Not allowed. 0 1: Input from source SSC_RxD @ P3.9. 1 0: Not allowed. 1 1: Not allowed.
EXI4SS	0 0: Not allowed. 0 1: Input from source SSC_SCLK @ P3.13. 1 0: Not allowed. 1 1: Not allowed.
EXI5SS	0 0: Not allowed. 0 1: Input from source USB_suspend interrupt. 1 0: Not allowed. 1 1: Not allowed.

Bit	Function
EXI6SS	0 0: Must be set to ’00’. 0 1: Not allowed. 1 0: Not allowed. 1 1: Not allowed.
EXI7SS	0 0: Not allowed. 0 1: Input from source RTC_INT. 1 0: Not allowed. 1 1: Not allowed.

7.8.3 Interrupt Subnode Control

The Real Time Clock (RTC) interrupt T14INT and the PLL/OWD interrupt share one interrupt node, the XPER3 interrupt node. In order to enable the interrupt handler to determine the source of that shared interrupt request, the subnode interrupt control register ISNC is provided. The separate interrupt request and enable flags of register ISNC (see below) for the PLL (PLLIR, PLLIE) as well as for the RTC (T14IR, T14IE) are used as shown in **Figure 26**.



Note: The Interrupt Service Routine must clear the IR flag in register ISNC manually. Otherwise no further interrupts can be detected.
All request flags are bit protected.

Figure 26 Interrupt Subnode Control for PLL / RTC Interrupts

ISNC register is defined as follows:

ISNC ($F1DE_H$ / EF_H) **ESFR-bReset Value : 0000_H**

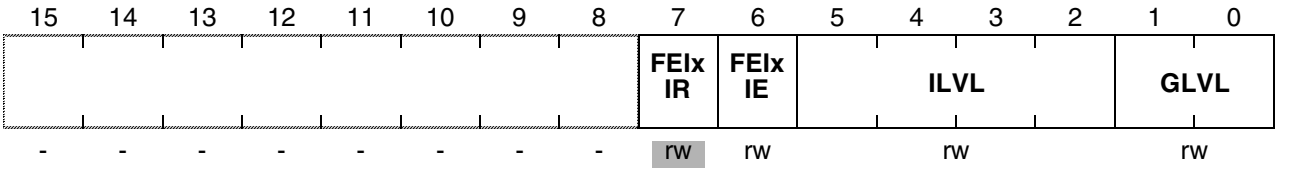
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
												PLL IE	**PLL IR**	**RTC T14 IE**	**RTC T14 IR**

Bit	Function
T14IR	T14 Overflow Interrupt Request Flag ‘0’: No request pending ‘1’: This source has raised an interrupt request
T14IE	T14 Overflow Interrupt Enable Control Bit ‘0’: Interrupt request is disabled ‘1’: Interrupt request is enabled
PLLIR	PLL Interrupt Request Flag ‘0’: No request pending ‘1’: This source has raised an interrupt request
PLLIE	PLL Interrupt Enable Control Bit ‘0’: Interrupt request is disabled ‘1’: Interrupt request is enabled

7.8.4 Interrupt Control Register

The interrupt control registers listed below (FEI1IC..FEI0IC) control the fast external interrupts of the C161U.

FEIxIC (See Table) SFR **Reset Value: - - 00_{HH}**



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

Table 26 Fast External Interrupt Control Register Addresses

Register	Address	External Interrupt
FEI0IC	$FF88_H$ / $C4_H$	EX0IN
FEI1IC	$FF8A_H$ / $C5_H$	EX1IN

7.9 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.

C161U provides two different kinds of trapping mechanisms. **Hardware traps** are triggered by events that occur during program execution (eg. illegal access or undefined opcode), **software traps** are initiated via an instruction within the current execution flow.

Software Traps

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, eg. to emulate additional instructions by generating an Illegal Opcode trap. C161U 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 (ie. 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 (ie. 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 C161U 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
- Ilegal 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 C161U 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.

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 C161U 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.

8 Parallel Ports

In order to accept or generate single external control signals or parallel data, the C161U provides up to 56 parallel I/O lines. C161U features **Port 0** (inculdes 8 bit P0H and 8 bit P0L), **Port 1** (8 bit P1H and 8 bit P1L), **Port 2** (2 bit), **Port 3** (10 bit), **Port 4** (5 bit) and **Port 6** (7 bit).

These port lines may be used for general purpose Input/Output controlled via software or may be used implicitly by C161U's integrated peripherals or the External Bus Controller.

All port lines are bit addressable, and all input/output lines are individually (bit-wise) programmable as inputs or outputs via direction registers. The I/O ports are true bidirectional ports which are switched to high impedance state when configured as inputs. The output drivers of the I/O ports (P0H, P1, P2, P3, P4, P6) 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 (DPx.y = '0') 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.

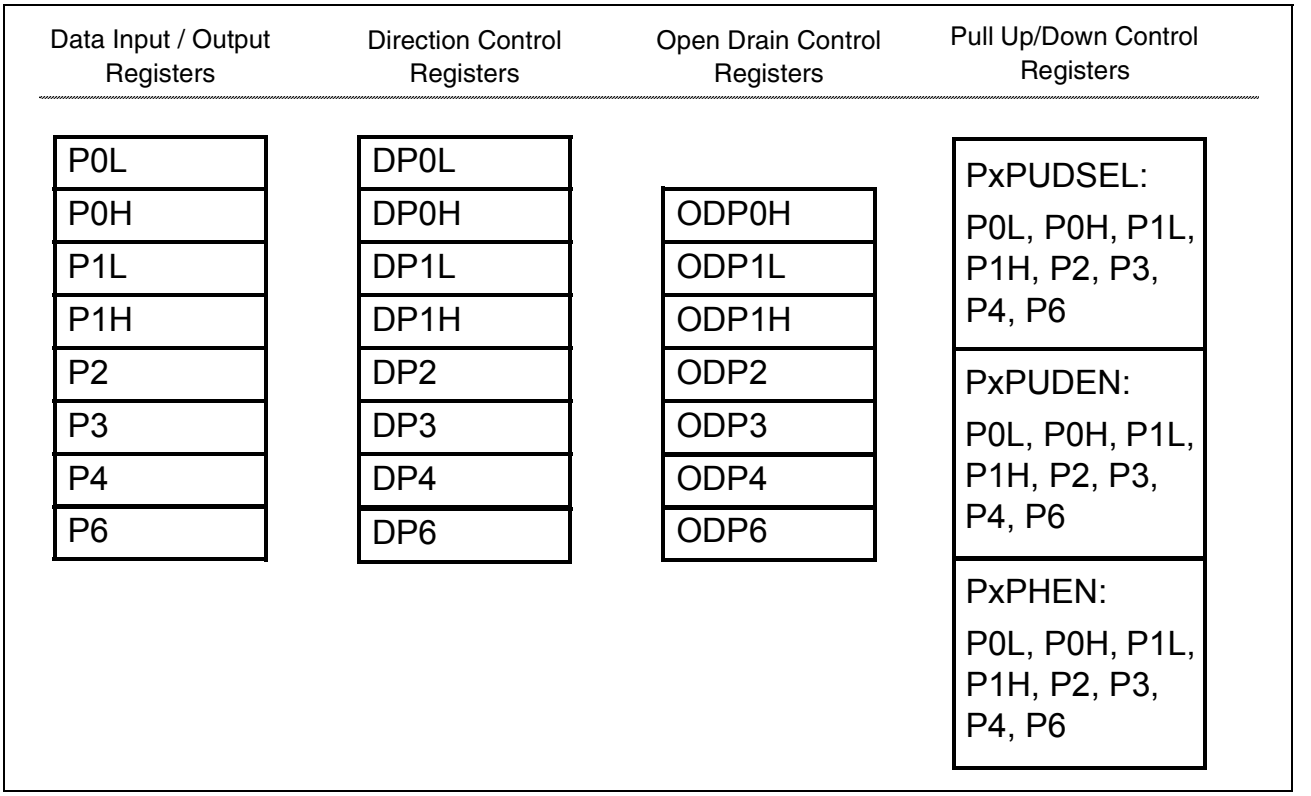


Figure 27 SFRs and Pins associated with the Parallel Ports

In the C161U 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 implemented for all ports except P0L, and is controlled through the respective Open Drain Control Registers ODPx. 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.

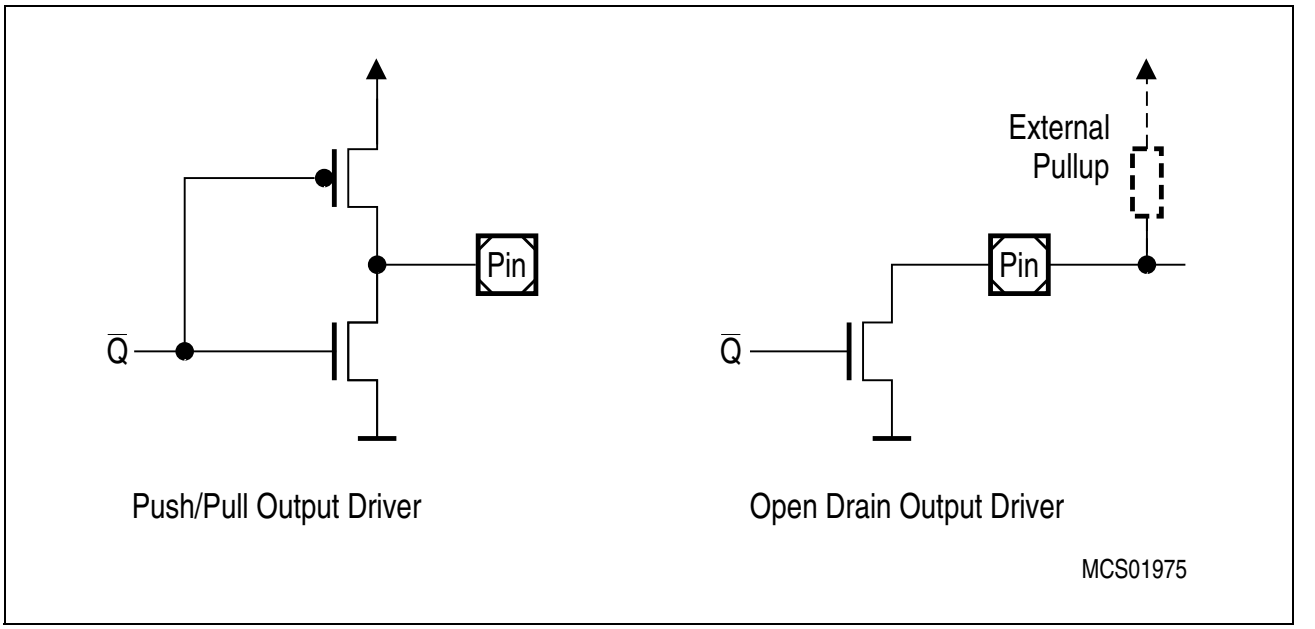


Figure 28 Output Drivers in Push/Pull Mode and in Open Drain Mode

Alternate Port Functions

Each port line has one programmable alternate input or output function associated.

PORT0 and PORT1 may be used as the address and data lines when accessing external memory.

Port 2 is used for fast external interrupt inputs.

Port 3 includes alternate input/output functions of timers, serial interfaces, the optional bus control signal $\overline{BHE}$/WRH and the system clock output (CLKOUT).

Port 4 outputs the additional segment address bits A20/A19/A17...A16 in systems where more than 64 KBytes of memory are to be accessed directly.

Port 6 provides the optional chip select outputs and the bus arbitration 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 ANDed with the alternate output data.

Note: DP0L and, if a 16 bit external XBus data bus is used, also DP0H must be '0' as long as the XBUS is active.

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.

Note: In this case, make sure DP0 is set to ’0’ signal.

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 I/O 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
```

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

8.1 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 (eg. via a PEC transfer) without effecting the other half.

If this port is used for general purpose I/O, the direction of each line can be configured via the corresponding direction registers DP0H and DP0L.

Each port line of PORT0H can be switched into push/pull or open drain mode via the open drain control register ODP0H.

For port pins configured as input (via DP0x or alternate function), an internal pull transistor is connected to the pad if register P0xPUDEN = ’1’, no matter whether the C161U is in normal operation mode or in power down mode. Either pulldown transistor or pullup transistor will be selected via P0xPUDSEL.

For port pins configured as output, the internal pull transistors are always disabled. The output driver is disabled in power down mode unless P0xPHEN = ’1’.

After reset, P0xPUDEN and P0xPUDSEL are set to HIGH signal, thereby providing the default reset configuration 1111_H to the C161U during reset.

Note: While this feature allows the user to start the C161U after reset in default configuration without external pull devices, the default configuration may be overwritten by stronger external pulldown devices. In this case, Software should isable the internal pull’s after reset (see also next chapter ’Alternate Function’).

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

ODP0H ($FE22_H$ / 11_H) SFR **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								ODP0H.7	ODP0H.6	ODP0H.5	ODP0H.4	ODP0H.3	ODP0H.2	ODP0H.1	ODP0H.0
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
ODP0H.y	Port0H Open Drain control register bit y ODP0H.y = 0: Port line P0H.y output driver in push/pull mode ODP0H.y = 1: Port line P0H.y output driver in open drain mode

P0LPUDSEL ($FE60_H$ / 30_H) SFR **Reset Value: - - FF_H**

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

P0HPUDSEL ($FE62_H$ / 31_H) SFR **Reset Value: - - FF_H**

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

Bit	Function
P0xPUDSEL.y	Pulldown/Pullup Selection 0: internal programmable pulldown transistor is selected 1: internal programmable pullup transistor is selected

P0LPUDEN ($FE64_H$ / 32_H) SFR **Reset Value: - - FF_H**

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

P0HPUDEN ($FE66_H$ / 33_H) SFR **Reset Value: - - FF_H**

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

Bit	Function
P0xPUDEN.y	Pulldown/Pullup Enable 0: internal programmable pull transistor is disabled 1: internal programmable pull transistor is enabled

P0LPHEN ($FE68_H$ / 34_H) SFR **Reset Value: - - 00_H**

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

P0HPHEN ($FE6A_H$ / 35_H) SFR **Reset Value: - - 00_H**

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

Bit	Function
P0xPHEN.y	Output Driver Enable in Power Down Mode 0: output driver is disabled in power down mode 1: output driver is enabled in power down mode

8.1.1 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 I/O (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 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.

Note: When the reset is terminated, the internal pullup devices must be switched off by Software 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.

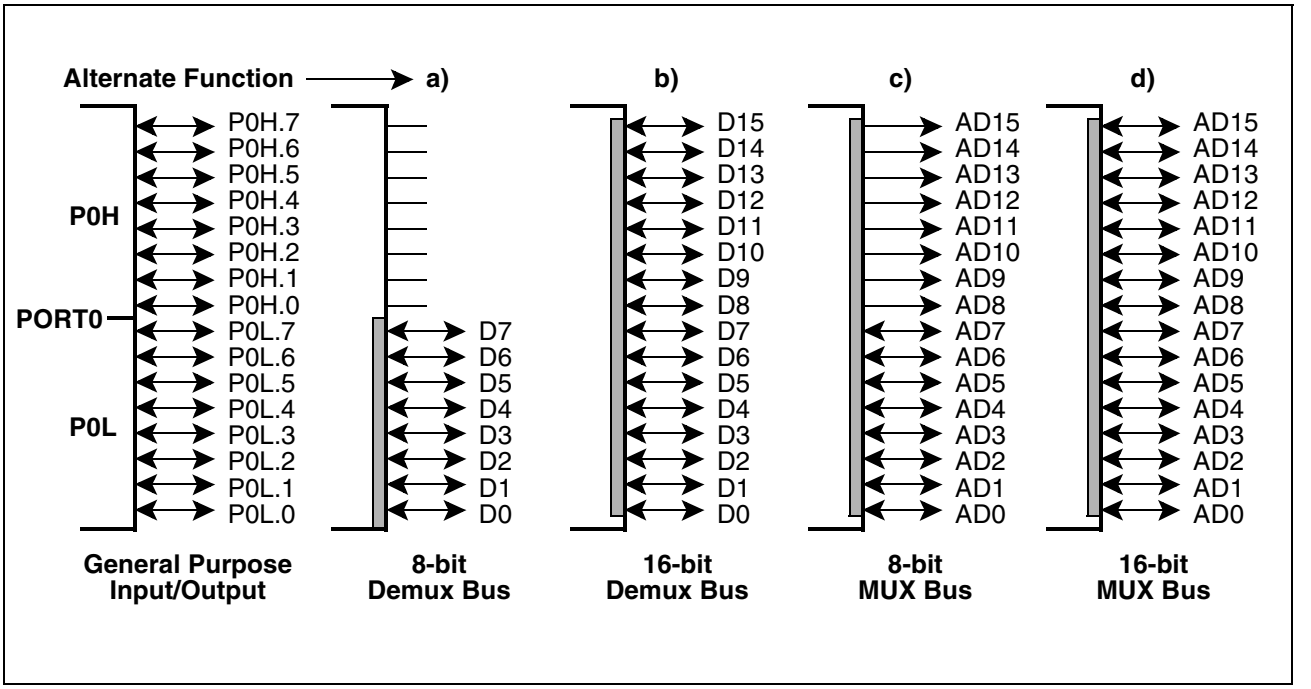


Figure 29 PORT0 I/O and Alternate Functions

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.

Figure 30 shows the structure of a PORT0 pin.

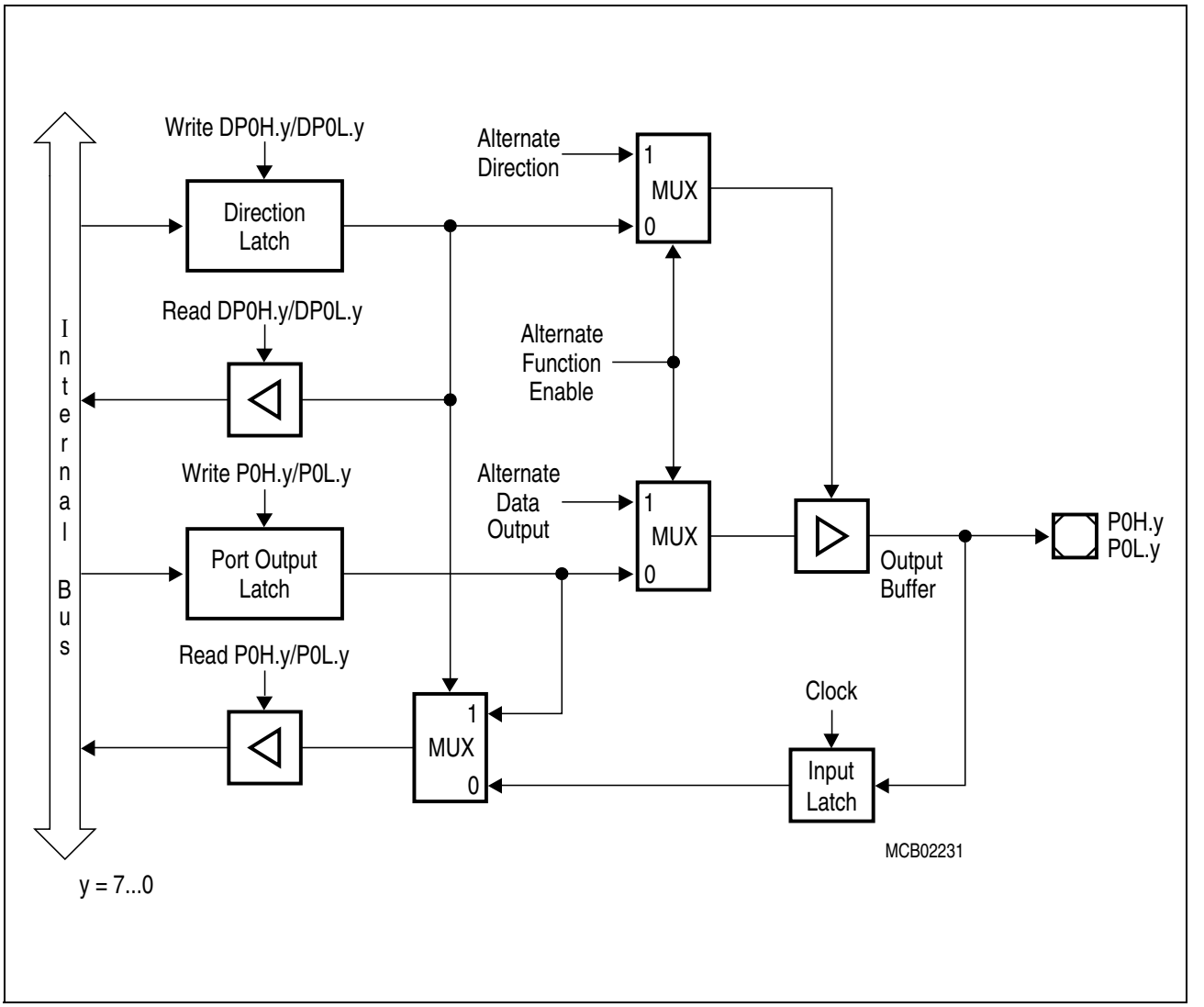


Figure 30 Block Diagram of a PORT0 Pin

8.2 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 (eg. via a PEC transfer) without effecting the other half.

If this port is used for general purpose I/O, the direction of each line can be configured via the corresponding direction registers DP1H and DP1L.

Each port line can be switched into push/pull or open drain mode via the open drain control register ODP1L and ODP1H.

For port pins configured as input (via DP1x or alternate function), an internal pull transistor is connected to the pad if register P1xPUDEN = ’1’, no matter wheter the C161U is in normal operation mode or in power down mode. Either pulldown transistor or pullup transistor will be selected via P1xPUDSEL.

For port pins configured as output, the internal pull transistors are always disabled. The output driver is disabled in power down mode unless P1xPHEN = ’1’.

After reset, P1xPUDEN and P1xPUDSEL are set to LOW signal.

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

ODP1L ($FE24_H$ / 12_H) SFR **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								ODP1L.7	ODP1L.6	ODP1L.5	ODP1L.4	ODP1L.3	ODP1L.2	ODP1L.1	ODP1L.0
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

ODP1H (FE26$_H$ / 13$_H$) SFR **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								ODP1H.7	ODP1H.6	ODP1H.5	ODP1H.4	ODP1H.3	ODP1H.2	ODP1H.1	ODP1H.0
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
ODP1x.y	Port1x Open Drain control register bit y ODP1x.y = 0: Port line P1x.y output driver in push/pull mode ODP1x.y = 1: Port line P1x.y output driver in open drain mode

P1LPUDSEL (FE6C$_H$ / 36$_H$) SFR **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								P1LPUDSEL.7	P1LPUDSEL.6	P1LPUDSEL.5	P1LPUDSEL.4	P1LPUDSEL.3	P1LPUDSEL.2	P1LPUDSEL.1	P1LPUDSEL.0
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

P1HPUDSEL (FE6E$_H$ / 37$_H$) SFR **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								P1HPUDSEL.7	P1HPUDSEL.6	P1HPUDSEL.5	P1HPUDSEL.4	P1HPUDSEL.3	P1HPUDSEL.2	P1HPUDSEL.1	P1HPUDSEL.0
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
P1xPUDSEL.y	Pulldown/Pullup Selection 0: internal programmable pulldown transistor is selected 1: internal programmable pullup transistor is selected

P1LPUDEN ($FE70_H$ / 38_H) SFR **Reset Value: - - 00_H**

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

P1HPUDEN ($FE72_H$ / 39_H) SFR **Reset Value: - - 00_H**

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

Bit	Function
P1xPUDEN.y	Pulldown/Pullup Enable P1xPUDEN.y = 0: internal programmable pull transistor is disabled P1xPUDEN.y = 1: internal programmable pull transistor is enabled

P1LPHEN ($FE74_H$ / $3A_H$) SFR **Reset Value: - - 00_H**

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

P1HPHEN ($FE76_H$ / $3B_H$) SFR **Reset Value: - - 00_H**

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

Bit	Function
P1xPHEN.y	Output Driver Enable in Power Down Mode P1xPHEN.y = 0: output driver is disabled in power down mode P1xPHEN.y = 1: output driver is enabled in power down mode

8.2.1 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 I/O.

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 I/O.

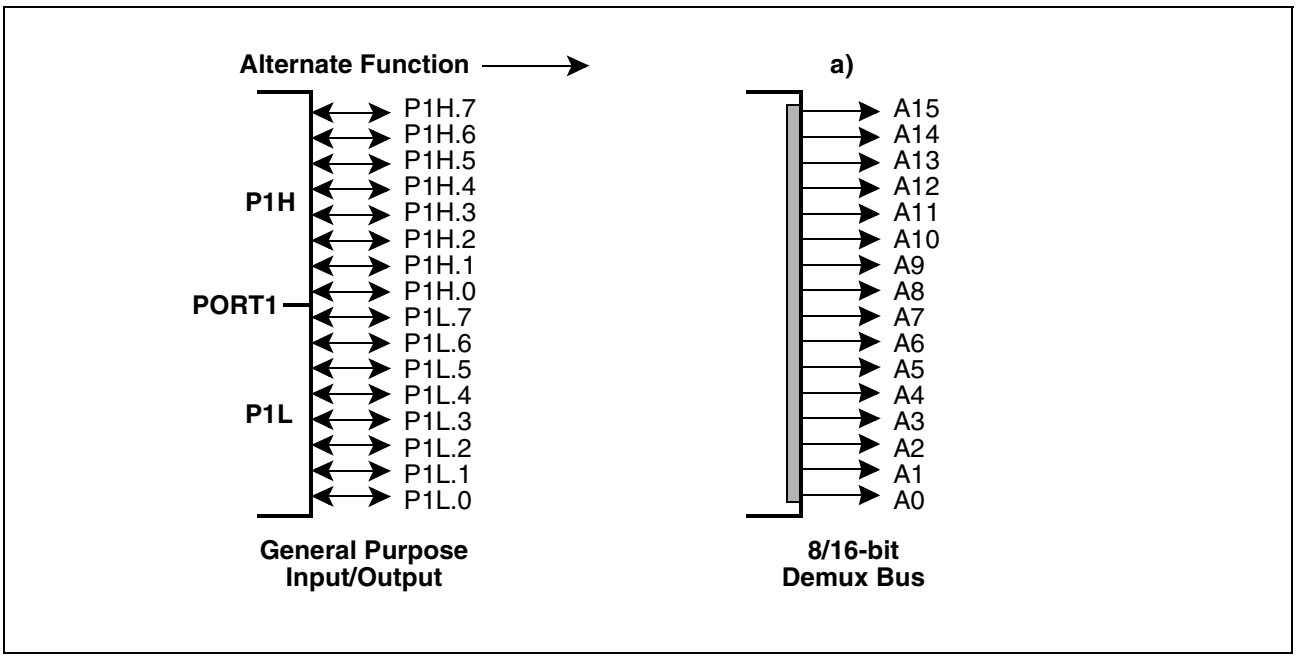


Figure 31 PORT1 I/O and Alternate Functions

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.

Figure 32 shows the structure of a PORT1 pin.

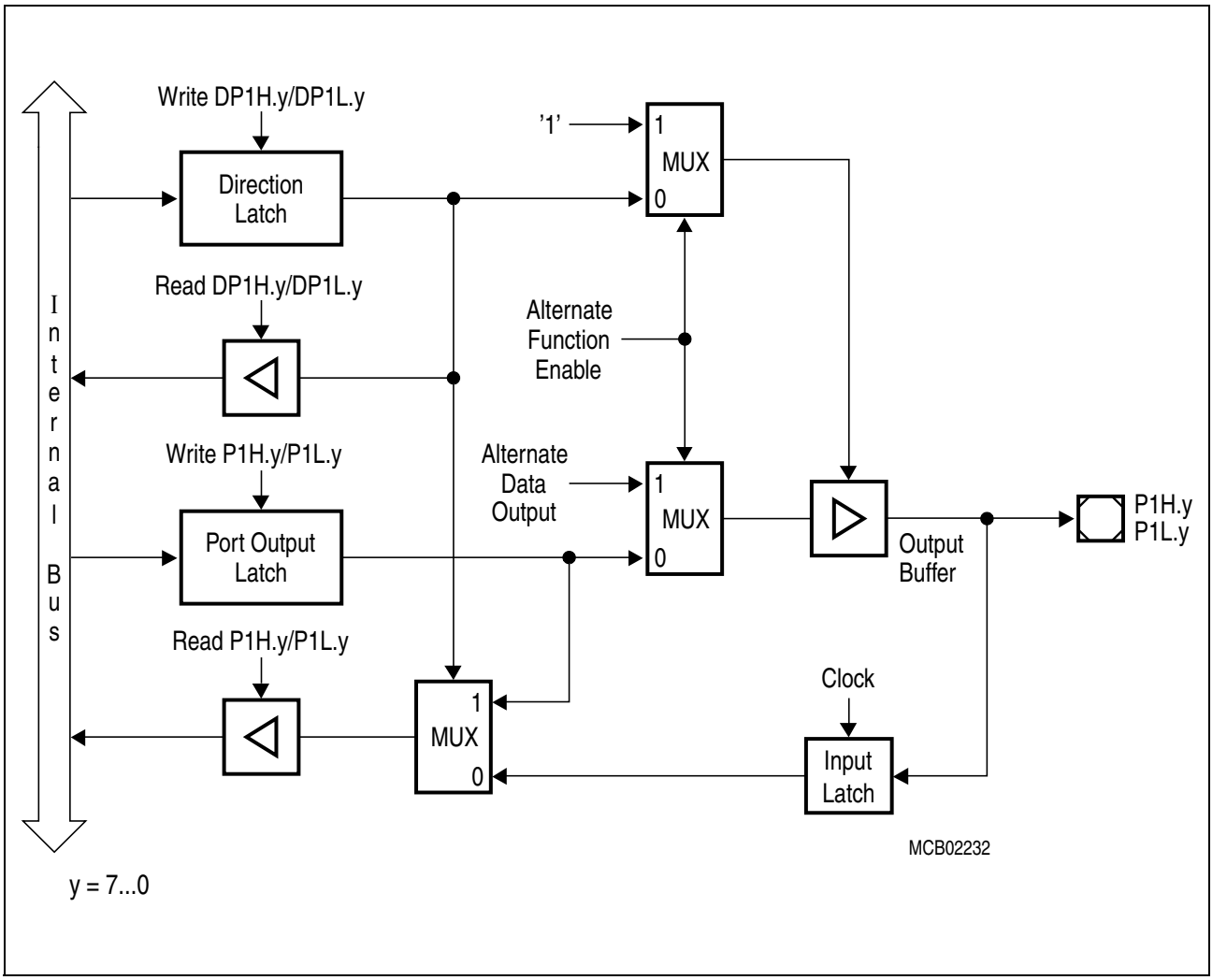


Figure 32 Block Diagram of a PORT1 Pin

8.3 PORT2

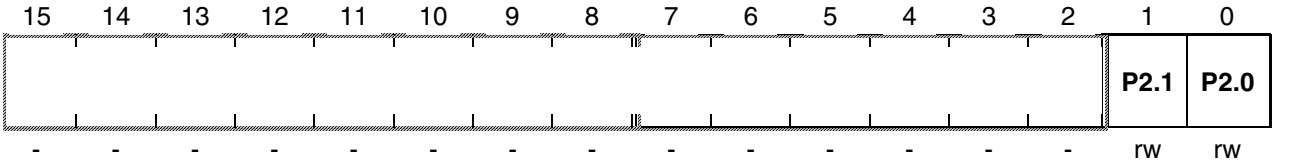
In the C161U Port 2 is an 2 -bit port. If Port 2 is used for general purpose I/O, the direction of each line can be configured via the corresponding direction register DP2. Each port line can be switched into push/pull or open drain mode via the open drain control register ODP2.

For port pins configured as input (via DP2 or alternate function), an internal pull transistor is connected to the pad if register P2PUDEN = '1', no matter wheter the C161U is in normal operation mode or in power down mode. Either pulldown transistor or pullup transistor will be selected via P2PUDSEL.

For port pins configured as output, the internal pull transistors are always disabled. The output driver is disabled in power down mode unless P2PHEN = '1'.

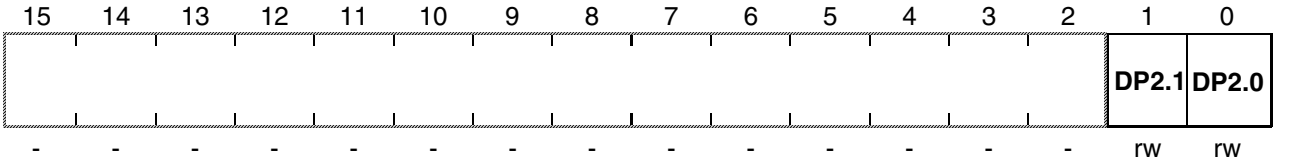
After reset, P2PUDEN and P2PUDSEL are set to LOW signal.

P2 ($FFC0_H$ / $E0_H$) SFR **Reset Value: - - 00_H**



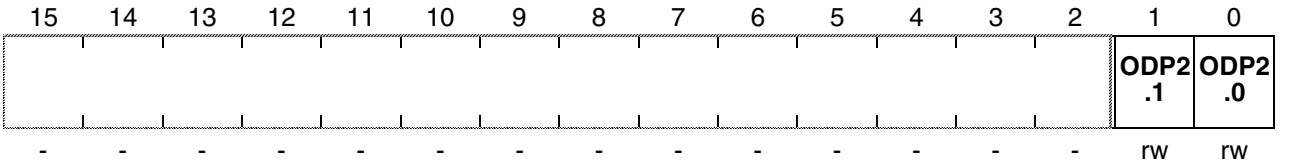
Bit	Function
P2.y	Port data register P2 bit y

DP2 ($FFC2_H$ / $E1_H$) SFR **Reset Value: - - 00_H**



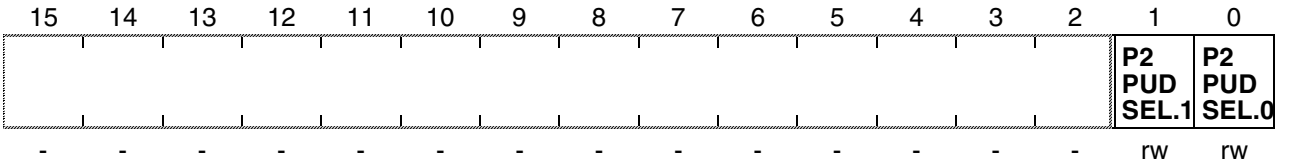
Bit	Function
DP2.y	Port direction register DP2 bit y DP2.y = 0: Port line P2.y is an input (high-impedance) DP2.y = 1: Port line P2.y is an output

ODP2 ($F1C2_H$ / $E1_H$) ESFR **Reset Value: - - 00_H**



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

P2PUDSEL ($FE78_H$ / $3C_H$) SFR **Reset Value: - - 00_H**



Bit	Function
P2PUDSEL.y	Pulldown/Pullup Selection P2PUDSEL.y = 0: internal programmable pulldown transistor is selected P2PUDSEL.y = 1: internal programmable pullup transistor is selected

P2PUDEN ($FE7A_H$ / $3D_H$) SFR **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
														P2PUDEN.1	P2PUDEN.0
-	-	-	-	-	-	-	-	-	-	-	-	-	-	rw	rw

Bit	Function
P2PUDEN.y	Pulldown/Pullup Enable P2PUDEN.y = 0: internal programmable pull transistor is disabled P2PUDEN.y = 1: internal programmable pull transistor is enabled

P2PHEN ($FE7C_H$ / $3E_H$) SFR **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
														P2PHEN.1	P2PHEN.0
-	-	-	-	-	-	-	-	-	-	-	-	-	-	rw	rw

Bit	Function
P2PHEN.y	Output Driver Enable in Power Down Mode P2PHEN.y = 0: output driver is disabled in power down mode P2PHEN.y = 1: output driver is enabled in power down mode

8.3.1 Alternate Functions of PORT2

All Port 2 lines (P2.1..P2.0) can serve as Fast External Interrupt inputs (EX1IN...EX0IN).

Table 27 summarizes the alternate functions of Port 2.

Table 27 Port 2 Alternate Functions: Fast External Interrupts

Port 2 Pin	Alternate Function
P2.0	EX0IN Fast External Interrupt 0 Input
P2.1	EX1IN Fast External Interrupt 1 Input

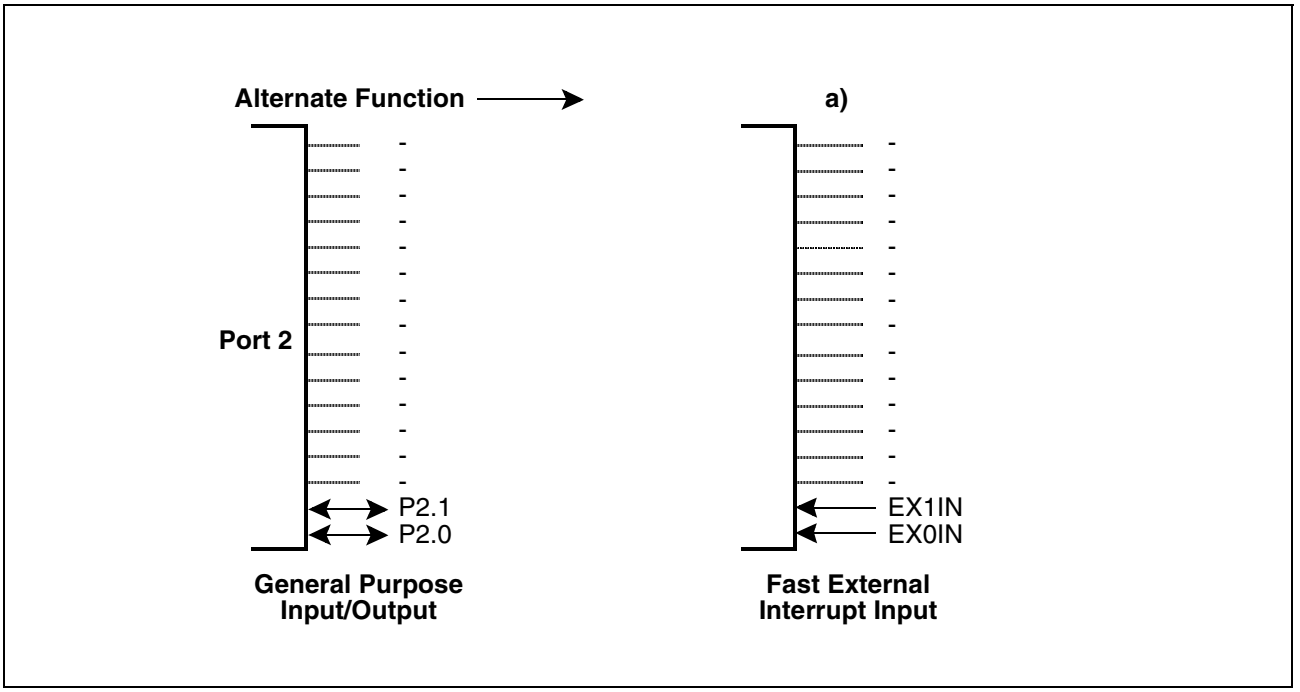


Figure 33 Port 2 I/O and Alternate Functions

The pins of Port 2 combine internal bus data and alternate data output before the port latch input.

Note: As opposed to the C161U, in other existing Infineon C16x devices EX0IN is assigned to P2.8 and EX1IN is assigned to P2.9 using the higher byte of Port 2 instead of using the lower byte of Port 2.

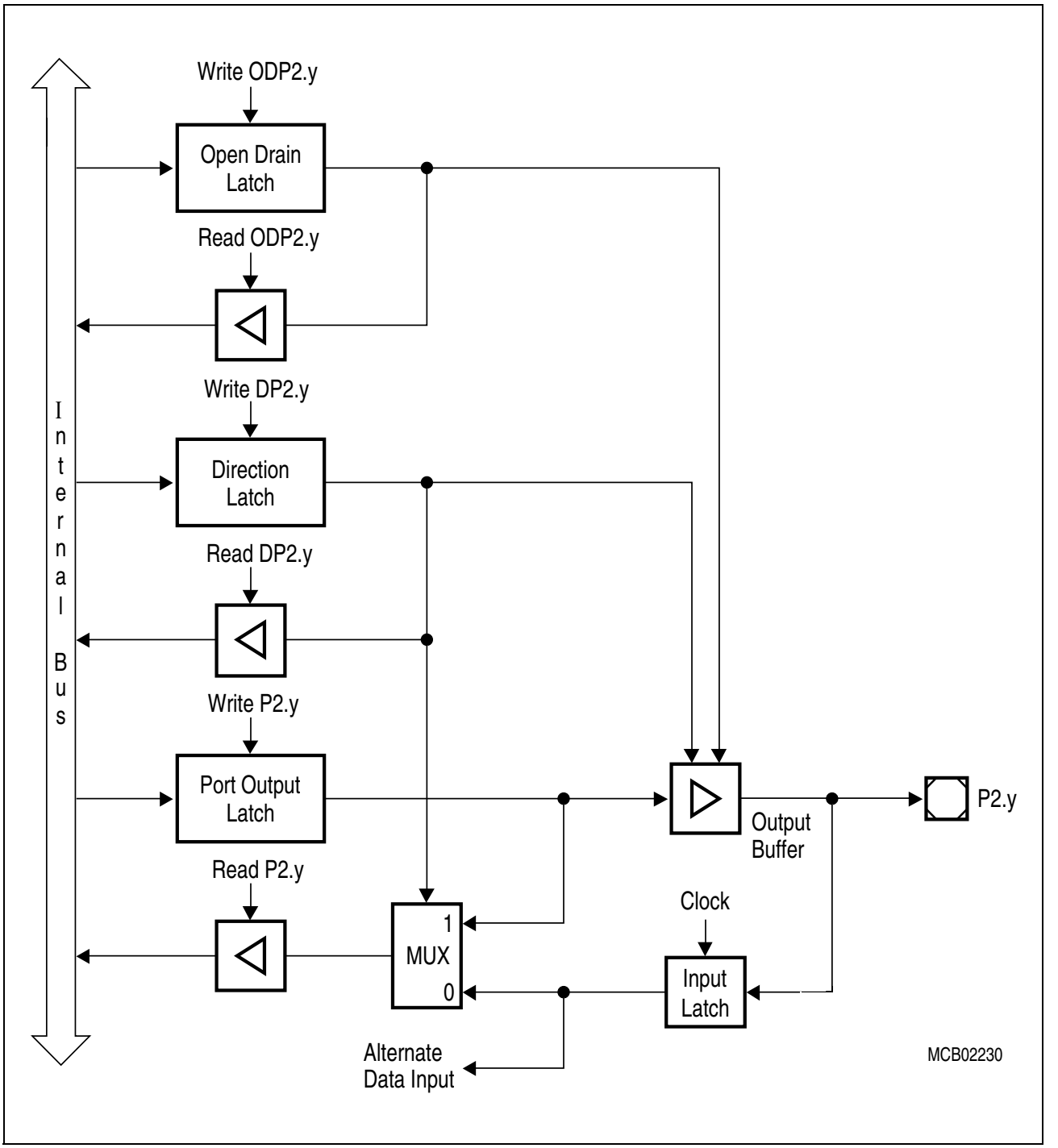


Figure 34 Block Diagram of a Port 2 Pin (y = 1...0)

8.4 PORT3

If this 10-bit port is used for general purpose I/O, the direction of each line can be configured via the corresponding direction register DP3. Each port lines can be switched into push/pull or open drain mode via the open drain control register ODP3.

Note: Due to pin limitations register bit P3.0..P3.2, P3.4, P3.7 and P3.14 is not connected to an output pin. The Port 3 bit-assignment is not consecutive to for compatibility with other C16x devices.

For port pins configured as input (via DP3 or alternate function), an internal pull transistor is connected to the pad if register P3PUDEN = ’1’, no matter wheter the C161U is in normal operation mode or in power down mode. Either pulldown transistor or pullup transistor will be selected via P3PUDSEL.

For port pins configured as output, the internal pull transistors are always disabled. The output driver is disabled in power down mode unless P3PHEN = ’1’.

After reset, P3PUDEN and P3PUDSEL are set to LOW signal.

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**	**-**	**P3.6**	**P3.5**	**-**	**P3.3**	**-**	**-**	**-**
rw	-	rw	rw	rw	rw	rw	rw	-	rw	rw	-	rw	-	-	-

Bit	Function
P3.y	Port data register P3 bit y

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 .6**	**DP3 .5**	**-**	**DP3 .3**	**-**	**-**	**-**
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 .15	**-**	**ODP3 .13**	**ODP3 .12**	**ODP3 .11**	**ODP3 .10**	**ODP3 .9**	**ODP3 .8**	**-**	**ODP3 .6**	**ODP3 .5**	**-**	**ODP3 .3**	**-**	**-**	**-**
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

P3PUDSEL ($FE7E_H$ / $3F_H$) SFR **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P3PUDSEL.15	**-**	**P3PUDSEL.13**	**P3PUDSEL.12**	**P3PUDSEL.11**	**P3PUDSEL.10**	**P3PUDSEL.9**	**P3PUDSEL.8**	**-**	**P3PUDSEL.6**	**P3PUDSEL.5**	**-**	**P3PUDSEL.3**	**-**	**-**	**-**
rw	-	rw	rw	rw	rw	rw	rw	-	rw	rw	-	rw	-	-	-

Bit	Function
P3PUDSEL.y	Pulldown/Pullup Selection P3PUDSEL.y = 0: internal programmable pulldown transistor is selected P3PUDSEL.y = 1: internal programmable pullup transistor is selected

P3PUDEN ($FE80_H$ / 40_H) SFR **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P3PUDEN.15	-	P3PUDEN.13	P3PUDEN.12	P3PUDEN.11	P3PUDEN.10	P3PUDEN.9	P3PUDEN.8	-	P3PUDEN.6	P3PUDEN.5	-	P3PUDEN.3	-	-	-
rw	-	rw	rw	rw	rw	rw	rw	-	rw	rw	-	rw	-	-	-

Bit	Function
P3PUDEN.y	Pulldown/Pullup Enable P3PUDEN.y = 0: internal programmable pull transistor is disabled P3PUDEN.y = 1: internal programmable pull transistor is enabled

P3PHEN ($FE82_H$ / 41_H) SFR **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P3PHEN.15	-	P3PHEN.13	P3PHEN.12	P3PHEN.11	P3PHEN.10	P3PHEN.9	P3PHEN.8	-	P3PHEN.6	P3PHEN.5	-	P3PHEN.3	-	-	-
rw	-	rw	rw	rw	rw	rw	rw	rw	rw	rw	-	rw	-	-	-

Bit	Function
P3PHEN.y	Output Driver Enable in Power Down Mode P3PHEN.y = 0: output driver is disabled in power down mode P3PHEN.y = 1: output driver is enabled in power down mode

8.4.1 Alternate Functions of PORT3

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

Table 28 summarizes the alternate functions of Port 3.

Table 28 Alternate Functions of Port 3

Port 3 Pin	Alternate Function	
P3.0	---	No pin assigned
P3.1	---	No pin assigned
P3.2	---	No pin assigned
P3.3	T3OUT	Timer 3 Toggle Output
P3.4	---	No pin assigned!
P3.5	T4IN	Timer 4 Count Input (T3EUD Input, T2EUD Input)
P3.6	T3IN	Timer 3 Count Input
P3.7	---	No pin assigned
P3.8	MRST	SSC Master Receive / Slave Transmit
P3.9	MTSR	SSC Master Transmit / Slave Receive
P3.10	TxD0	ASC Transmit Data Output
P3.11	RxD0	ASC Receive Data Input
P3.12	$\overline{BHE}$/$\overline{WRH}$	Byte High Enable / Write High Output
P3.13	SCLK	SSC Shift Clock Input/Output
P3.14	---	No pin assigned
P3.15	CLKOUT	System Clock Output

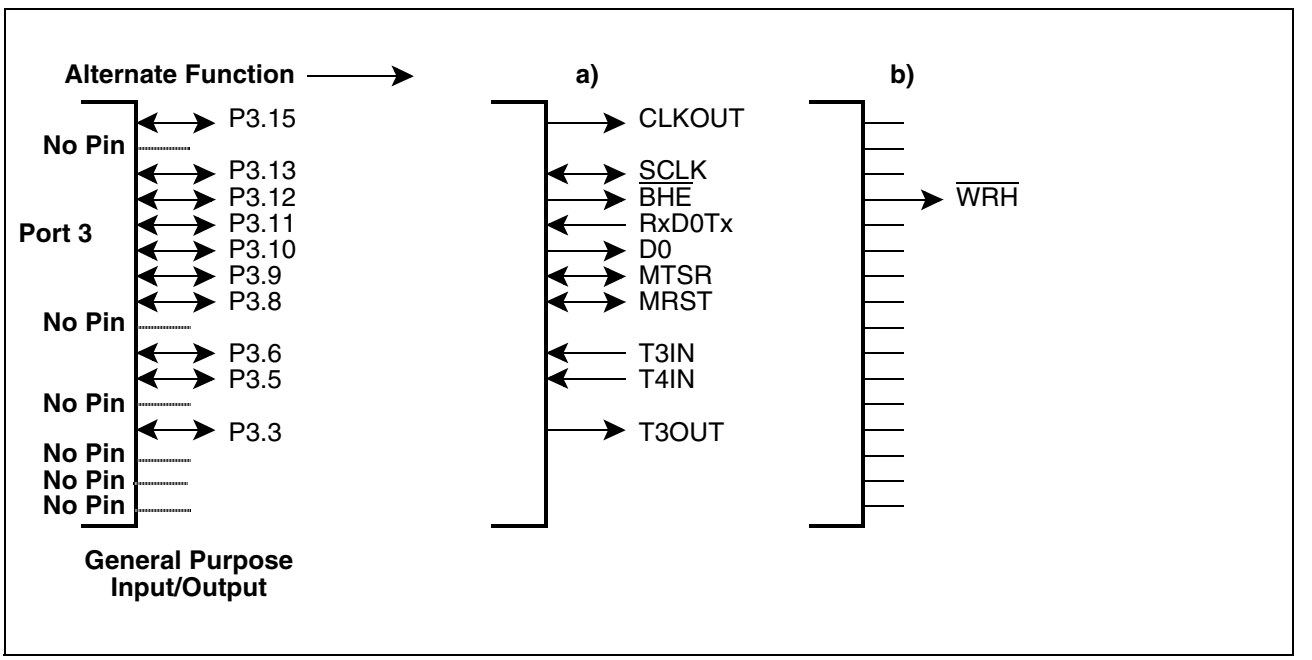


Figure 35 Port 3 I/O and Alternate Functions

The port structure of the Port 3 pins depends on their alternate function (see figures 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:
T3IN and T4IN/T3EUD/T2EUD.

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:
T6OUT, 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:
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.

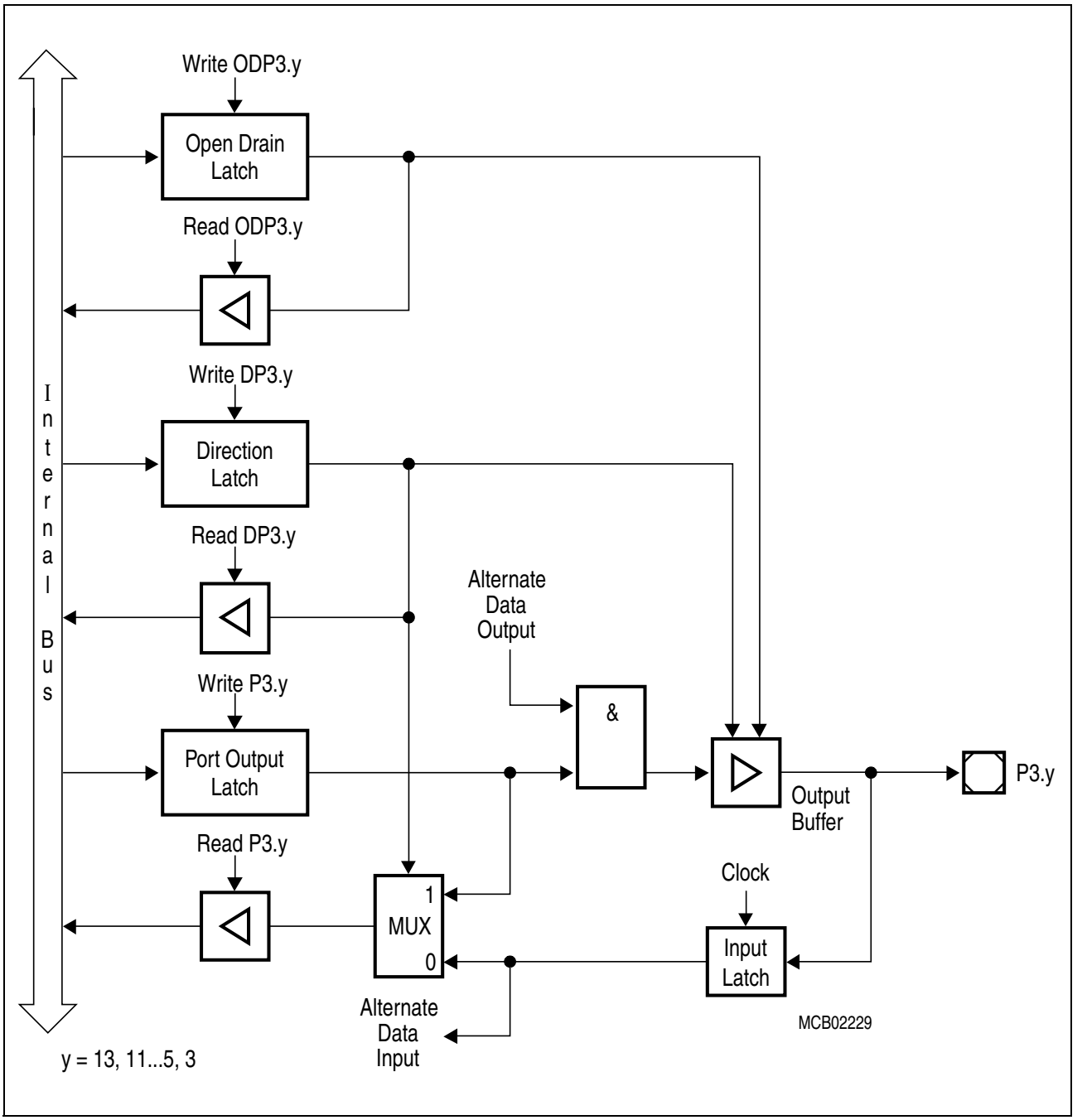


Figure 36 Block Diagram of a Port 3 Pin with Alternate Input or Alternate Output Function (y = 13, 11...8, 6, 5, 3)

Pin P3.12 (BHE/WRH) is one more pin with an alternate output function. However, its structure is slightly different (see figure below), because after reset the BHE or 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 BHE/WRH is not used in the system, this pin can be used for general purpose I/O by disabling the alternate function (BYTDIS = '1' / WRCFG='0').

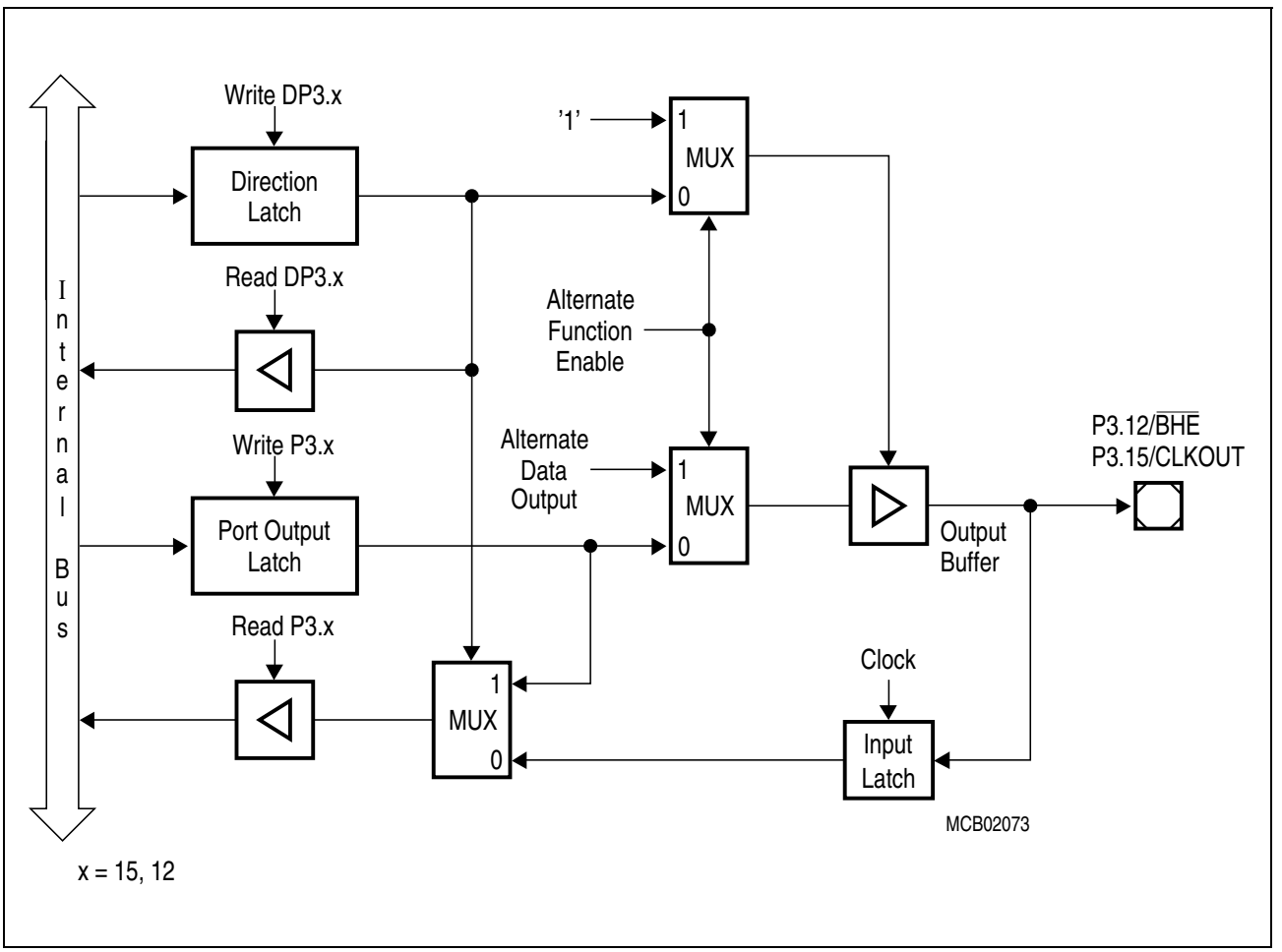


Figure 37 Block Diagram of Pins P3.15 (CLKOUT) and P3.12 (BHE/WRH)

Note: Enabling the BHE or WRH function automatically enables the P3.12 output driver. Setting bit DP3.12='1' is not required.
During bus hold pin P3.12 is switched back to its standard function and is then controlled by DP3.12 and P3.12. Keep DP3.12 = '0' in this case to ensure floating in hold mode.

8.5 PORT4

If this 5-bit port is used for general purpose I/O, the direction of each line can be configured via the corresponding direction register DP4.

Each port line can be switched into push/pull or open drain mode via the open drain control register ODP4.

For port pins configured as input (via DP4 or alternate function), an internal pull transistor is connected to the pad if register P4PUDEN = ’1’, no matter wheter the C161U is in normal operation mode or in power down mode. Either pulldown transistor or pullup transistor will be selected via P4PUDSEL.

For port pins configured as output, the internal pull transistors are always disabled. The output driver is disabled in power down mode unless P4PHEN = ’1’.

After reset, P4PUDEN and P4PUDSEL are set to LOW signal.

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
											P4.4	**P4.3**	**P4.2**	**P4.1**	**P4.0**
-	-	-	-	-	-	-	-	-	-	-	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
											DP4.4	DP4.3	DP4.2	DP4.1	DP4.0
-	-	-	-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw

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

ODP4 ($F1CA_H$ / $E5_H$) **ESFR** **Reset Value: - - 00_H**

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

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

P4PUDSEL ($FE84_H$ / 42_H) SFR **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
											P4PUDSEL.4	P4PUDSEL.3	P4PUDSEL.2	P4PUDSEL.1	P4PUDSEL.0
-	-	-	-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw

Bit	Function
P4PUDSEL.y	Pulldown/Pullup Selection P4PUDSEL.y = 0: internal programmable pulldown transistor is selected P4PUDSEL.y = 1: internal programmable pullup transistor is selected

P4PUDEN (FE86$_H$ / 43$_H$) SFR **Reset Value: - - 00$_H$**

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

Bit	Function
P4PUDEN.y	Pulldown/Pullup Enable P4PUDEN.y = 0: internal programmable pull transistor is disabled P4PUDEN.y = 1: internal programmable pull transistor is enabled

P4PHEN (FE88$_H$ / 44$_H$) SFR **Reset Value: - - 00$_H$**

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

Bit	Function
P4PHEN.y	Output Driver Enable in Power Down Mode P4PHEN.y = 0: output driver is disabled in power down mode P4PHEN.y = 1: output driver is enabled in power down mode

8.5.1 Alternate Functions of PORT4

During external bus cycles that use segmentation (ie. 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 I/O. If segment address lines are selected, the alternate function of Port 4 may be necessary to access eg. external memory directly after reset. For this reason Port 4 will be switched to its 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) eg. in order to check the configuration during run time.

Table 29 summarizes the alternate functions of Port 4 depending on the number of selected segment address lines (coded via bitfield SALSEL).

Table 29 Alternate Functions of Port 4

Port 4 Pin	Std. Function SALSEL=0164 KB	Altern. Function SALSEL=11 256KB	Altern. Function SALSEL=001 MB	Altern. Function SALSEL=102 MB
P4.0	Gen. purpose I/O	Seg. Address A16	Seg. Address A16	Seg. Address A16
P4.1	Gen. purpose I/O	Seg. Address A17	Seg. Address A17	Seg. Address A17
P4.2	Gen. purpose I/O	Gen. purpose I/O	Seg. Address A18	Seg. Address A18
P4.3	Gen. purpose I/O	Gen. purpose I/O	Seg. Address A19	Seg. Address A19
P4.4	Gen. purpose I/O	Gen. purpose I/O	Gen. purpose I/O	Seg. Address A20

396

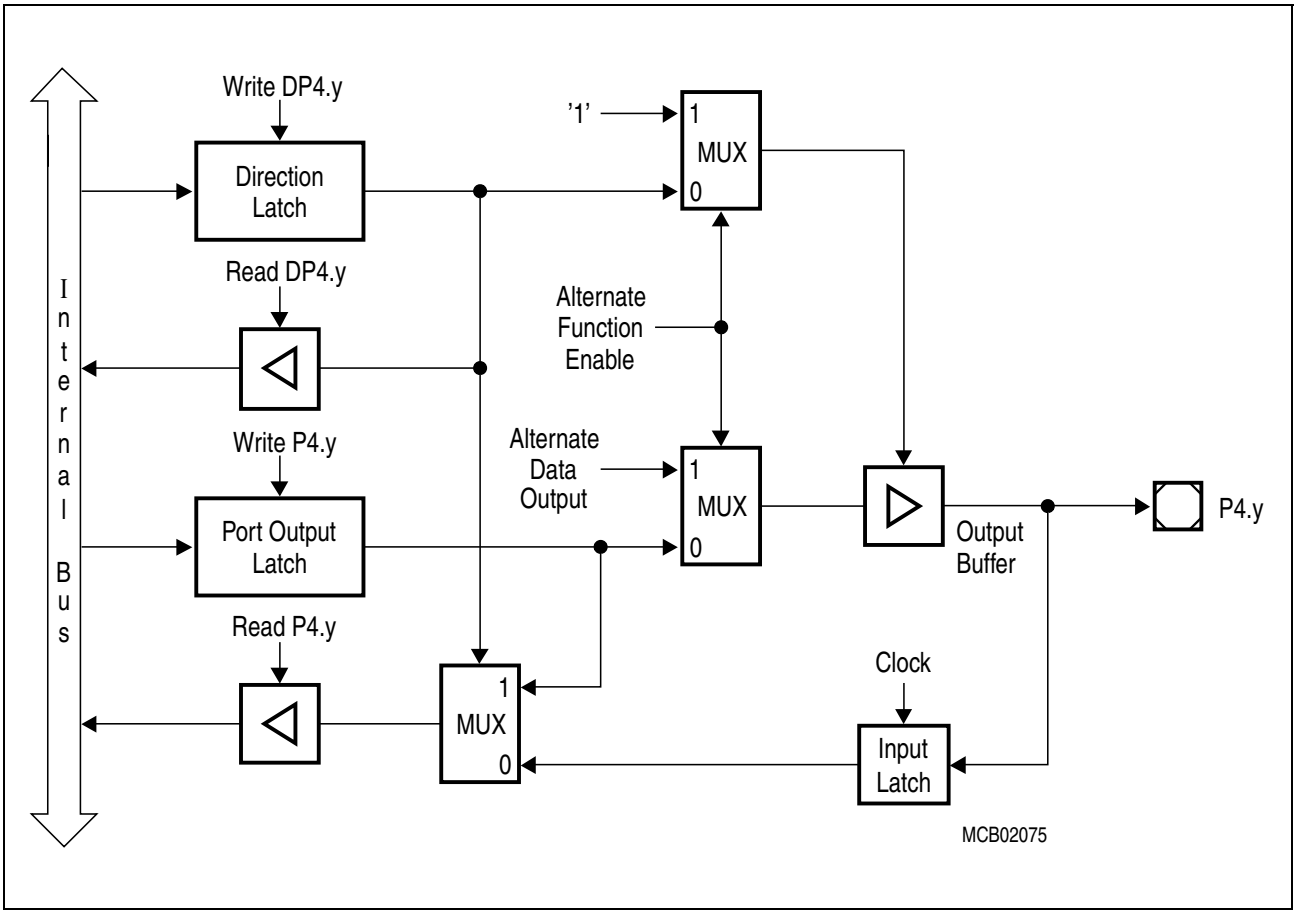


Figure 38 Block Diagram of a Port 4 Pin (y = 4...0)

8.6 PORT6

If this 7-bit port is used for general purpose I/O, 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.

For port pins configured as input (via DP6 or alternate function), an internal pull transistor is connected to the pad if register P6PUDEN = ’1’, no matter wheter the C161U is in normal operation mode or in power down mode. Either pulldown transistor or pullup transistor will be selected via P6PUDSEL.

For port pins configured as output, the internal pull transistors are always disabled. The output driver is disabled in power down mode unless P6PHEN = ’1’.

After reset, P6PUDEN and P6PUDSEL are set to LOW signal.

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	-	P6.3	P6.2	P6.1	P6.0
-	-	-	-	-	-	-	-	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	-	DP6.3	DP6.2	DP6.1	DP6.0
-	-	-	-	-	-	-	-	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
								ODP6.7	ODP6.6	ODP6.5	-	ODP6.3	ODP6.2	ODP6.1	ODP6.0
-	-	-	-	-	-	-	-	rw	rw	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

P6PUDSEL ($FE90_H$ / 48_H) SFR **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								P6 PUD SEL.7	P6 PUD SEL.6	P6 PUD SEL.5	-	P6 PUD SEL.3	P6 PUD SEL.2	P6 PUD SEL.1	P6 PUD SEL.0
-	-	-	-	-	-	-	-	rw	rw	rw	-	rw	rw	rw	rw

Bit	Function
P6PUDSEL.y	Pulldown/Pullup Selection P6PUDSEL.y = 0: internal programmable pulldown transistor is selected P6PUDSEL.y = 1: internal programmable pullup transistor is selected

P6PUDEN ($FE92_H$ / 49_H) SFR **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								P6 PUD EN.7	P6 PUD EN.6	P6 PUD EN.5	-	P6 PUD EN.3	P6 PUD EN.2	P6 PUD EN.1	P6 PUD EN.0
-	-	-	-	-	-	-	-	rw	rw	rw	-	rw	rw	rw	rw

Bit	Function
P6PUDEN.y	Pulldown/Pullup Enable P6PUDEN.y = 0: internal programmable pull transistor is disabled P6PUDEN.y = 1: internal programmable pull transistor is enabled

P6PHEN ($FE94_H$ / $4A_H$) SFR **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								P6 PHEN .7	P6 PHEN .6	P6 PHEN .5	-	P6 PHEN .3	P6 PHEN .2	P6 PHEN .1	P6 PHEN .0
-	-	-	-	-	-	-	-	rw	rw	rw	-	rw	rw	rw	rw

Bit	Function
P6PHEN.y	Output Driver Enable in Power Down Mode P6PHEN.y = 0: output driver is disabled in power down mode P6PHEN.y = 1: output driver is enabled in power down mode

8.6.1 Alternate Functions of PORT6

A programmable number of chip select signals (CS3...CS0) derived from the bus control registers (BUSCON3...BUSCON0) can be output on 4 pins of Port 6. The other 3 pins may be used for bus arbitration to accomodate additional masters in a C161U system. 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) eg. in order to check the configuration during run time.

Table 30 summarizes the alternate functions of Port 6 depending on the number of selected chip select lines (coded via bitfield CSSEL).

Table 30 Alternate Functions of Port 6

Port 6 Pin	Altern. Function CSSEL = 10	Altern. Function CSSEL = 01	Altern. Function CSSEL = 00	Altern. Function CSSEL = 11
P6.0	Gen. purpose I/O	Chip select $\overline{CS0}$	Chip select $\overline{CS0}$	Chip select $\overline{CS0}$
P6.1	Gen. purpose I/O	Chip select $\overline{CS1}$	Chip select $\overline{CS1}$	Chip select $\overline{CS1}$
P6.2	Gen. purpose I/O	Gen. purpose I/O	Chip select $\overline{CS2}$	Chip select $\overline{CS2}$
P6.3	Gen. purpose I/O	Gen. purpose I/O	Gen. purpose I/O	Chip select $\overline{CS3}$
P6.5	$\overline{HOLD}$ External hold request input			
P6.6	$\overline{HLDA}$ Hold acknowledge output			
P6.7	BREQ Bus request output			

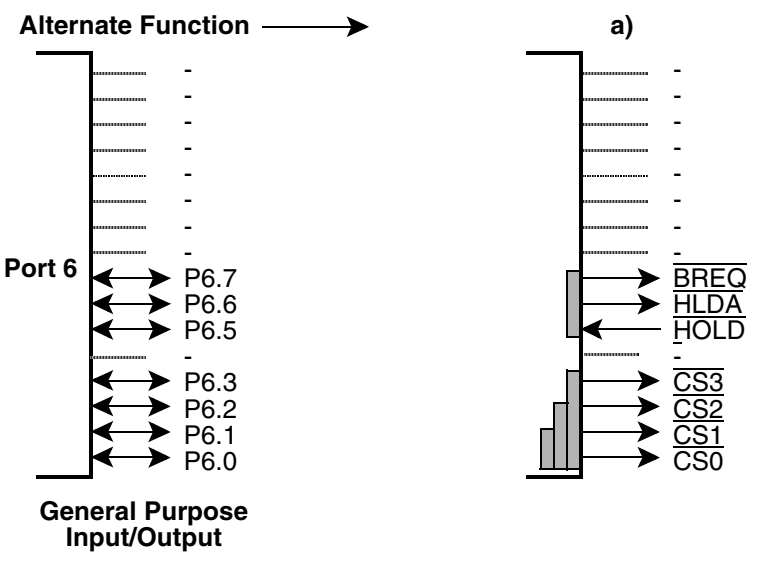


Figure 39 Port 6 I/O and Alternate Functions

The chip select lines of Port 6 additionally have an internal weak pullup device. This device is switched on under the following conditions:

- Always during reset
- If the Port 6 line is used as a chip select output, and the C161U is in Hold mode (invoked through $\overline{HOLD}$), and the respective pin driver is in push/pull mode (ODP6.x = ‘0’).

This feature is implemented to drive the chip select lines high during reset in order to avoid multiple chip selection, and to allow another master to access the external memory via the same chip select lines (Wired-AND), while the C161U is in Hold mode.

With ODP6.x = ‘1’ (open drain output selected), the internal pullup device will not be active during Hold mode; external pullup devices must be used in this case.
When entering Hold mode the $\overline{CS}$ lines are actively driven high for one clock phase, then the output level is controlled by the pullup devices (if activated).

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; at least signal $\overline{CS0}$ will be in push/pull output driver mode directly after reset.

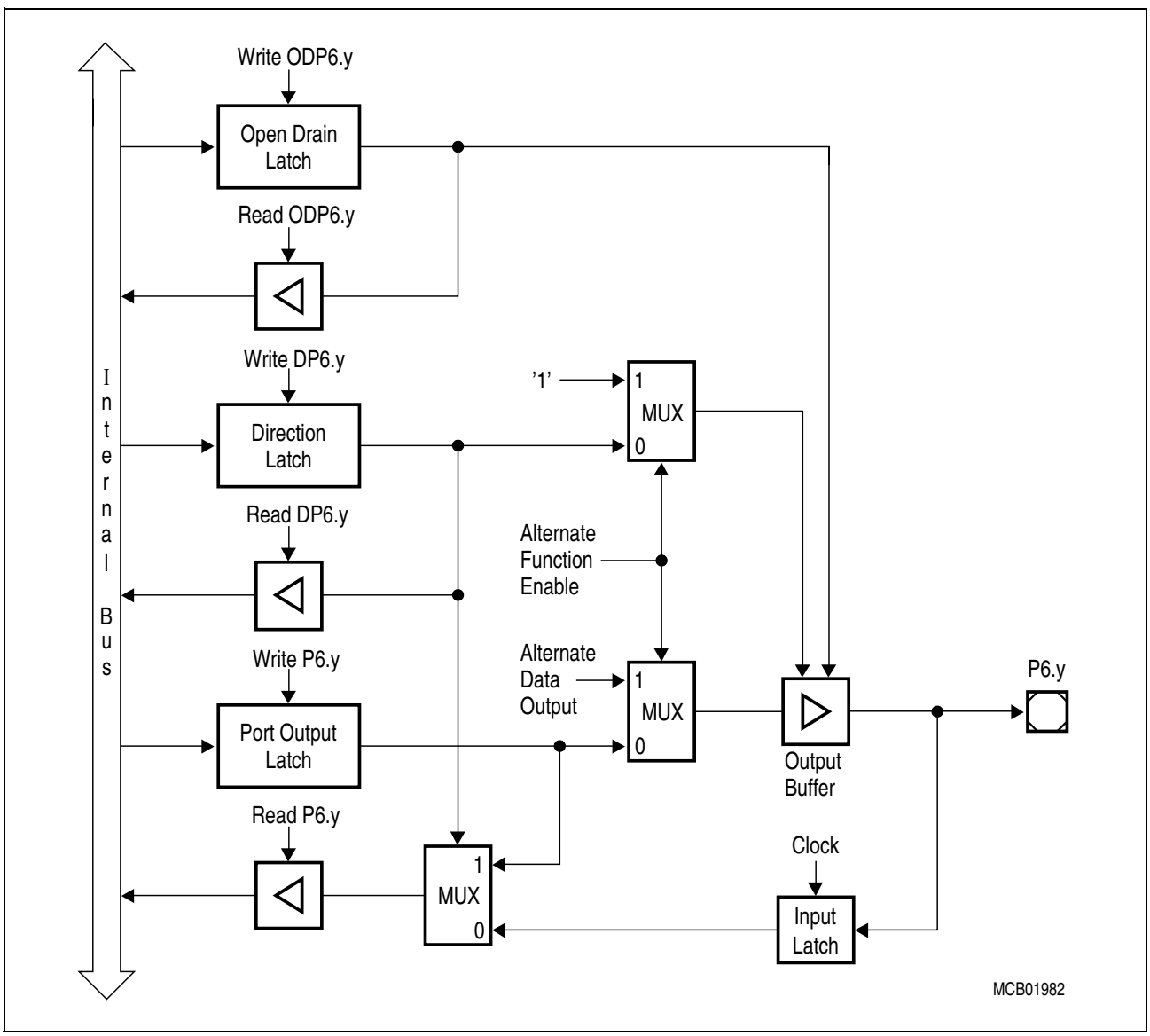


Figure 40 Block Diagram of Port 6 Pins with an alternate output function

The bus arbitration signals $\overline{\text{HOLD}}$, $\overline{\text{HLDA}}$ and $\overline{\text{BREQ}}$ are selected with bit HLDEN in register PSW. When the bus arbitration signals are enabled via HLDEN, also these pins are switched automatically to the appropriate direction. Note that the pin drivers for $\overline{\text{HLDA}}$ and $\overline{\text{BREQ}}$ are automatically enabled, while the pin driver for $\overline{\text{HOLD}}$ is automatically disabled.

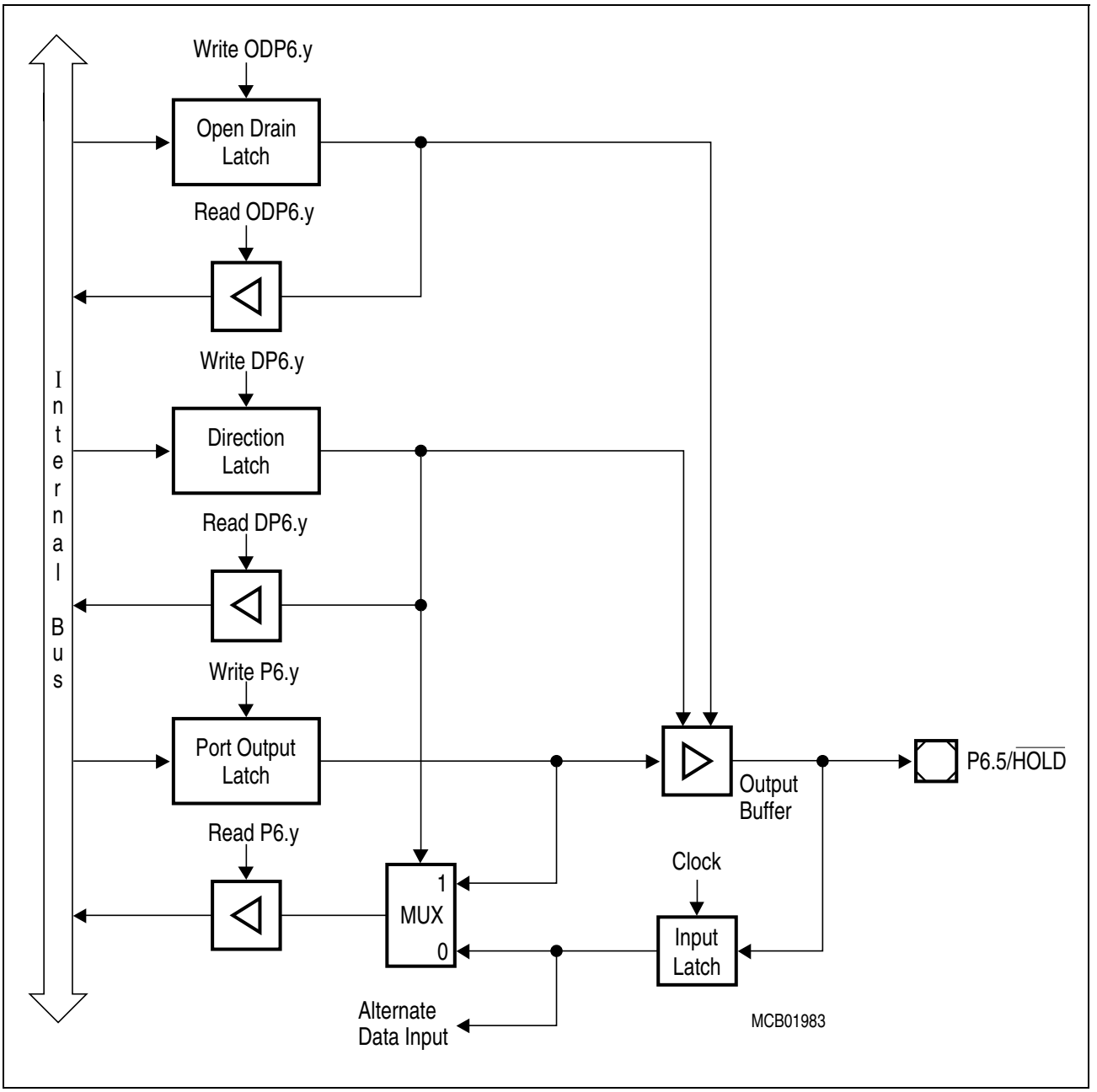


Figure 41 Block Diagram of Pin P6.5 ($\overline{\text{HOLD}}$)

9 Dedicated Pins

Most of the input/output or control signals of the functional the C161U are realized as alternate functions of pins of the parallel ports. There is, however, a number of signals that use separate pins, including the USB interface, the oscillator, special control signals and the power supply. **Table 31** summarizes all dedicated pins of the C161U.

Table 31 Dedicated Pins

Pin(s)	Function
ALE	Address Latch Enable
RD	External Read Strobe
$\overline{WR}$/$\overline{WRL}$	External Write/Write Low Strobe
READY	Ready Input
EA	External Access Enable
NMI	Non-Maskable Interrupt Input
RSTIN	Reset Input
RSTOUT	Reset Output
XTAL1, XTAL2	Oscillator Input/Output
CLKMODE	Oscillator Clock Input Mode Select
DMNS, DPLS	USB
$\overline{BRKIN}$, $\overline{BRKOUT}$	OCDS
TDI, TDO, TCK, TMS, TRST	JTAG Interface
TEST	Test Mode Enable
VDD, GND	Power Supply and Ground (11 pins VDD, 12 pins GND)

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, ie. 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, ie. the internal RAM and the special function registers. In single chip mode, ie. when no external bus is enabled (no BUSACT bit set), ALE will also remain inactive for X-Peripheral accesses.

External Read Strobe $\overline{\text{RD}}$ controls the output drivers of external memory or peripherals when the C161U reads data from these external devices. During reset and during Hold mode an internal pullup ensures an inactive (high) level on the $\overline{\text{RD}}$ output.

External Write Strobe $\overline{\text{WR}}$/$\overline{\text{WRL}}$ controls the data transfer from the C161U to an external memory or peripheral device. This pin may either provide an general $\overline{\text{WR}}$ signal activated for both byte and word write accesses, or specifically control the low byte of an external 16-bit device ($\overline{\text{WRL}}$) together with the signal $\overline{\text{WRH}}$ (alternate function of P3.12/ $\overline{\text{BHE}}$). During reset and during Hold mode an internal pullup ensures an inactive (high) level on the $\overline{\text{WR}}$/$\overline{\text{WRL}}$ output.

Note: Whether **$\overline{\text{RD}}$** and **$\overline{\text{WR}}$/$\overline{\text{WRL}}$** remain idle during X-peripheral accesses depends on the value of bit VISIBLE of register SYSCON.

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.

External Access Enable Pin $\overline{\text{EA}}$ is dedicated for on-chip ROM derivates. In this case it determines, if the chip after reset starts fetching code from the internal ROM area ($\overline{\text{EA}}$='1') or via the external bus interface ($\overline{\text{EA}}$='0'). **For the ROM-less C161U be sure to hold this input low.**

Non-Maskable Interrupt Input $\overline{\text{NMI}}$ allows to trigger a high priority trap via an external signal (eg. a power-fail signal). It also serves to validate the PWRDN instruction that switches the C161U into Power-Down mode. The $\overline{\text{NMI}}$ pin is sampled with every CPU clock cycle to detect transitions.

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 figure below) comprises the crystal, two low end capacitors and series resistor 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.
An external clock signal may be fed to the input XTAL1, leaving XTAL2 open.

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 following starting configuration is recommended to be used for the C161U:
Quarz: C_L = 30 pF (max.), R_S = 70 Ohm (max.), Accuracy: 96ppm or better
External: Circuitry: $C_A = C_B$ = 47 pF (max.), no serial resistor (Rx2 = 0)

Note: Please check the Infineon Application Notes in addition to this recommendation.

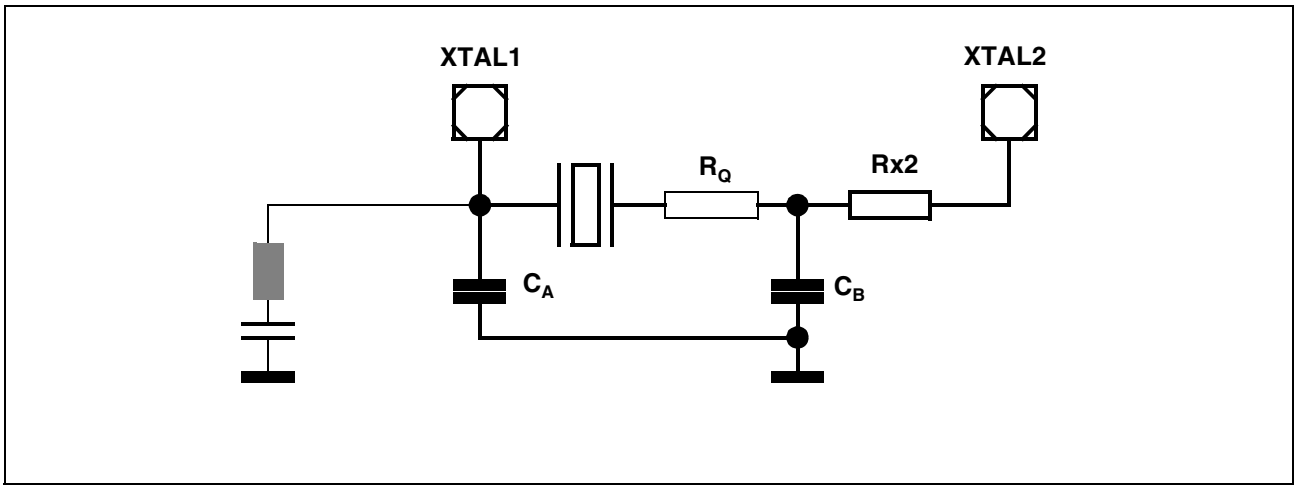


Figure 42 External Oscillator Circuitry

Clock Mode Select CLKMODE CLKMODE must be LOW if an external crystal is used. HIGH signal enables the direct clock input path and switches the internal oscillator in power down mode..

Reset Input RSTIN allows to put the C161U 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 RSTIN pin is raised compared to the standard pins in order to minimize the noise sensitivity of the reset input.

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

Power Supply pins VDD and GND provide the power supply for the digital logic of the C161U. The respective VCC/VSS pairs should be decoupled as close to the pins as possible. For best results it is recommended to implement two-level decoupling, eg. (the widely used) 100 nF in parallel with 30...40 pF capacitors which deliver the peak currents.

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

10 External Bus Interface

Although the C161U provides a powerful set of on-chip peripherals and on-chip RAM areas, these internal units only cover a small fraction of its address space of up to 2 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.

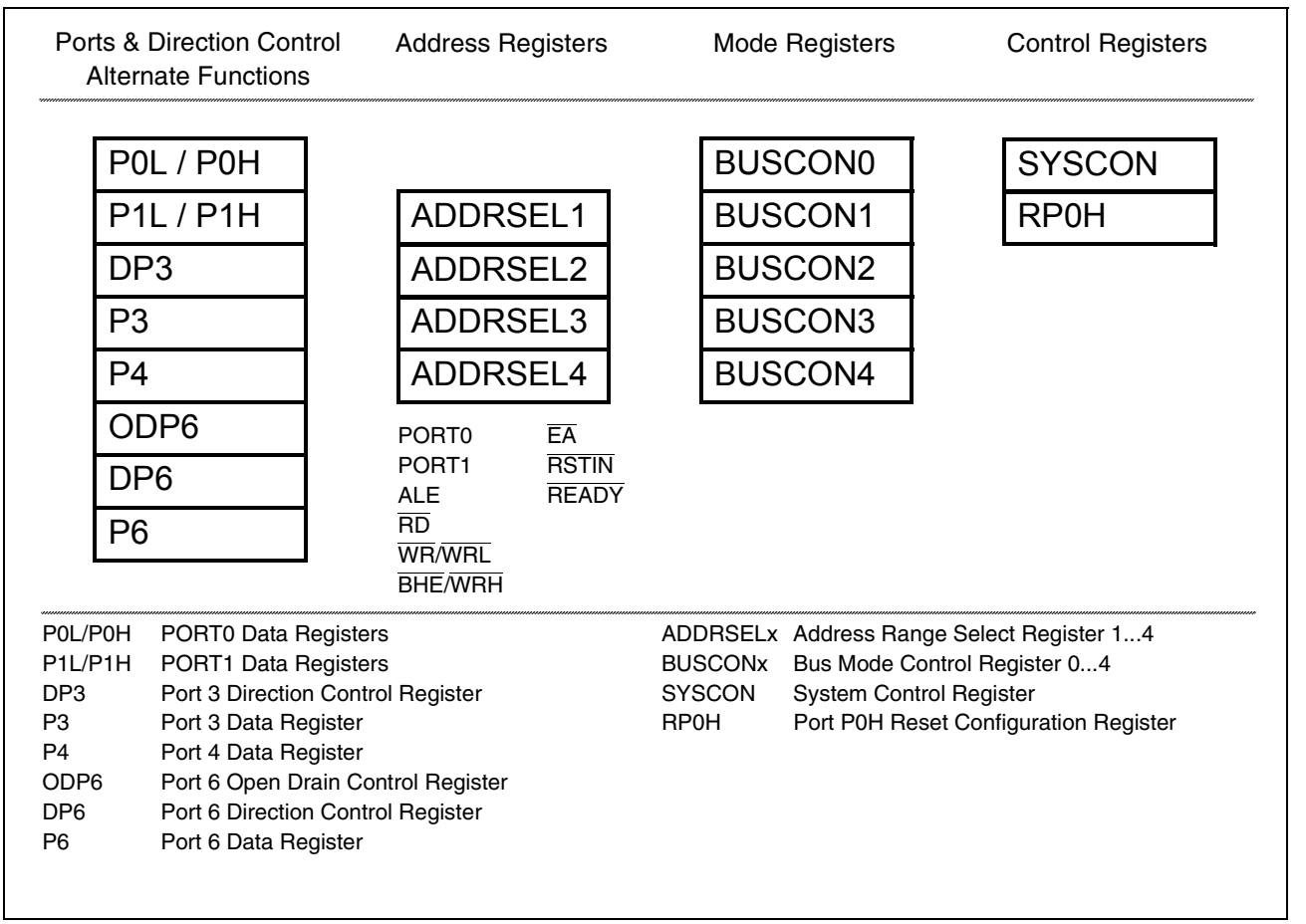


Figure 43 SFRs and Port Pins Associated with the External Bus Interface

Accesses to external memory or peripherals are executed by the integrated External Bus Controller (EBC). The function of the EBC is controlled via the SYSCON register and the BUSCONx and ADDRSELx registers. The BUSCONx registers specify the external bus cycles in terms of address (mux/demux), data (16-bit/8-bit), chip selects and length (waitstates / $\overline{\text{READY}}$ control / 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 C161U 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.

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

10.1 External Bus Modes

When the external bus interface is enabled (bit BUSACTx='1') and configured (bitfield BTYP), the C161U 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 C161U 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, 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, ie. 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.

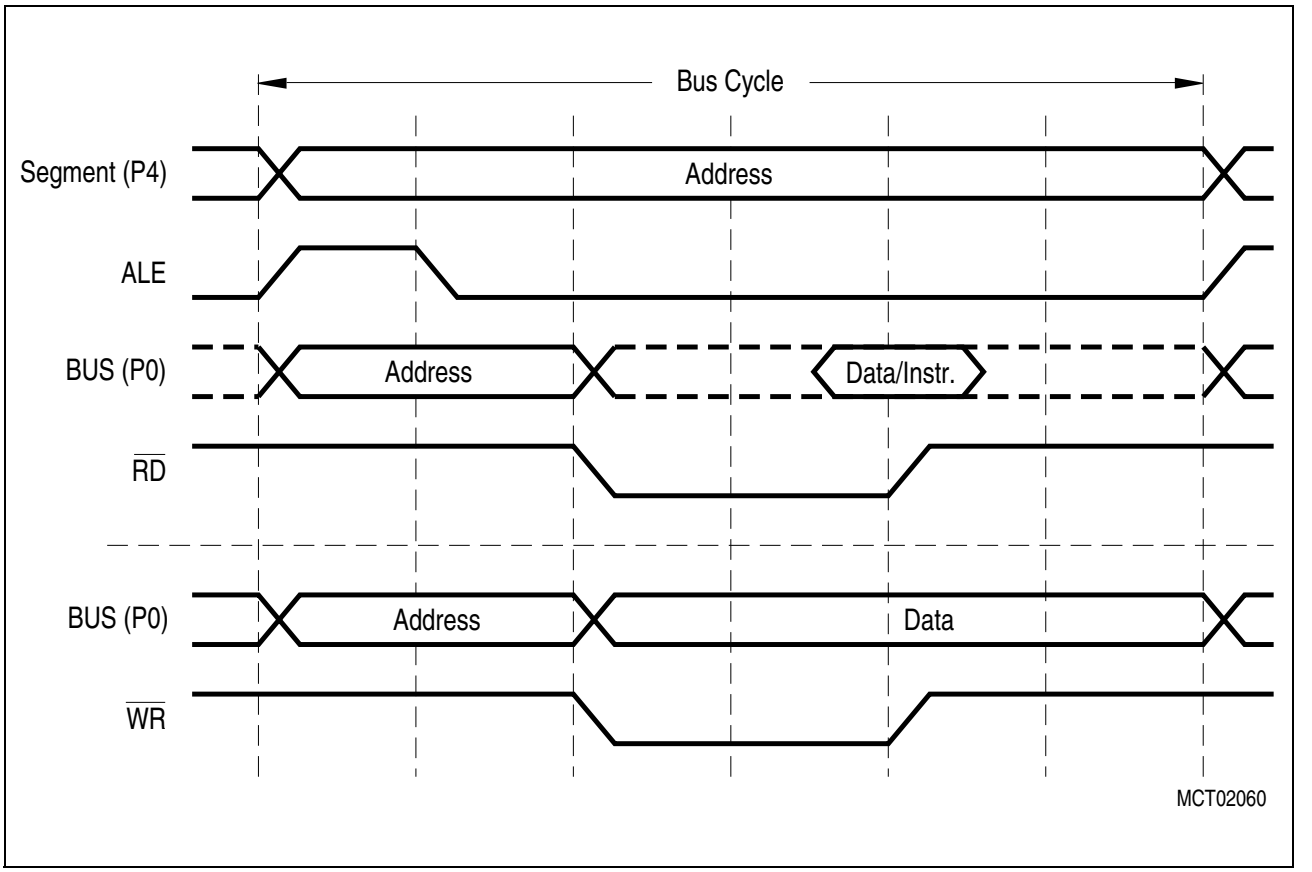


Figure 44 Multiplexed Bus Cycle

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 ($\overline{RD}$, $\overline{WR}$, $\overline{WRL}$, $\overline{WRH}$). Data is driven onto the data bus either by the EBC (for write cycles) or by the external memory/peripheral (for read cycles). After a period of time, 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.

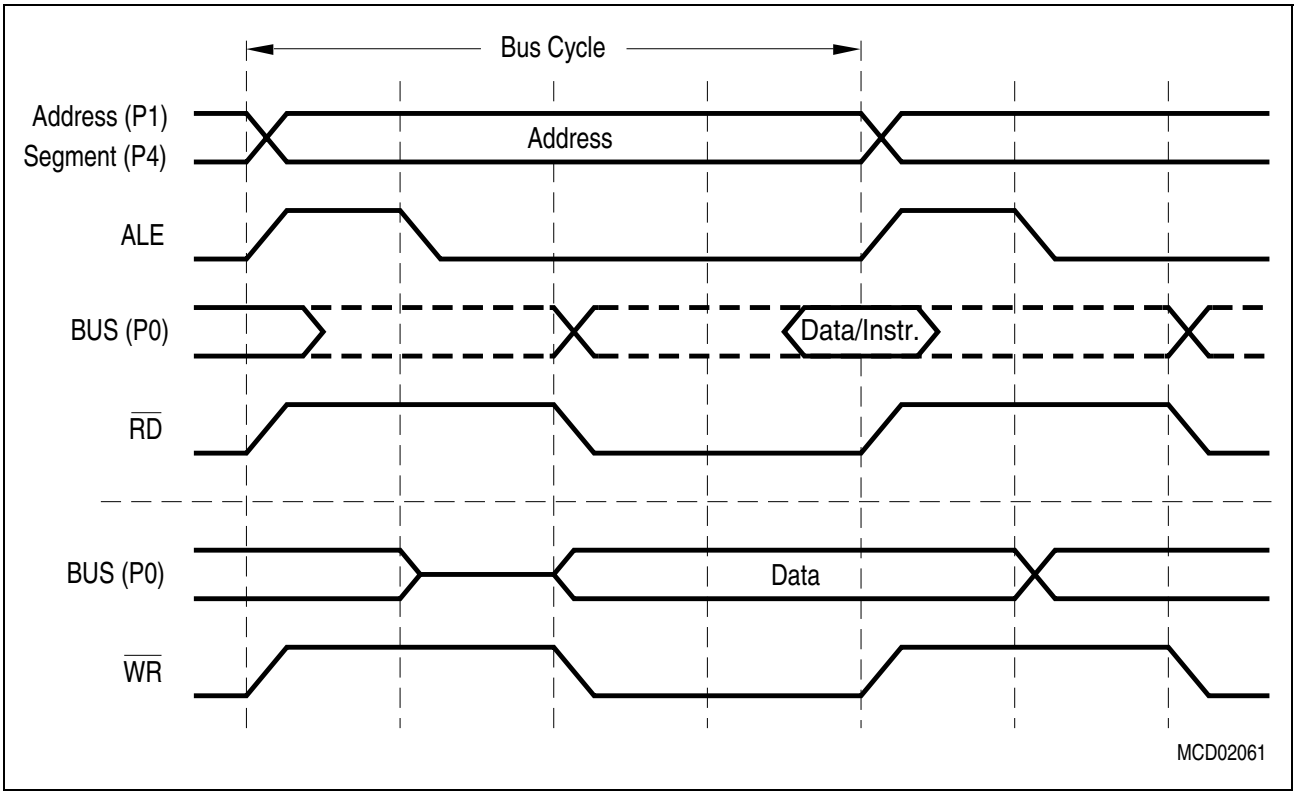


Figure 45 Demultiplexed Bus Cycle

Switching between the Bus Modes

The EBC allows to switch between different bus modes dynamically, ie. subsequent external bus cycles may be executed in different ways. Certain address areas may use multiplexed or demultiplexed buses or use 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.

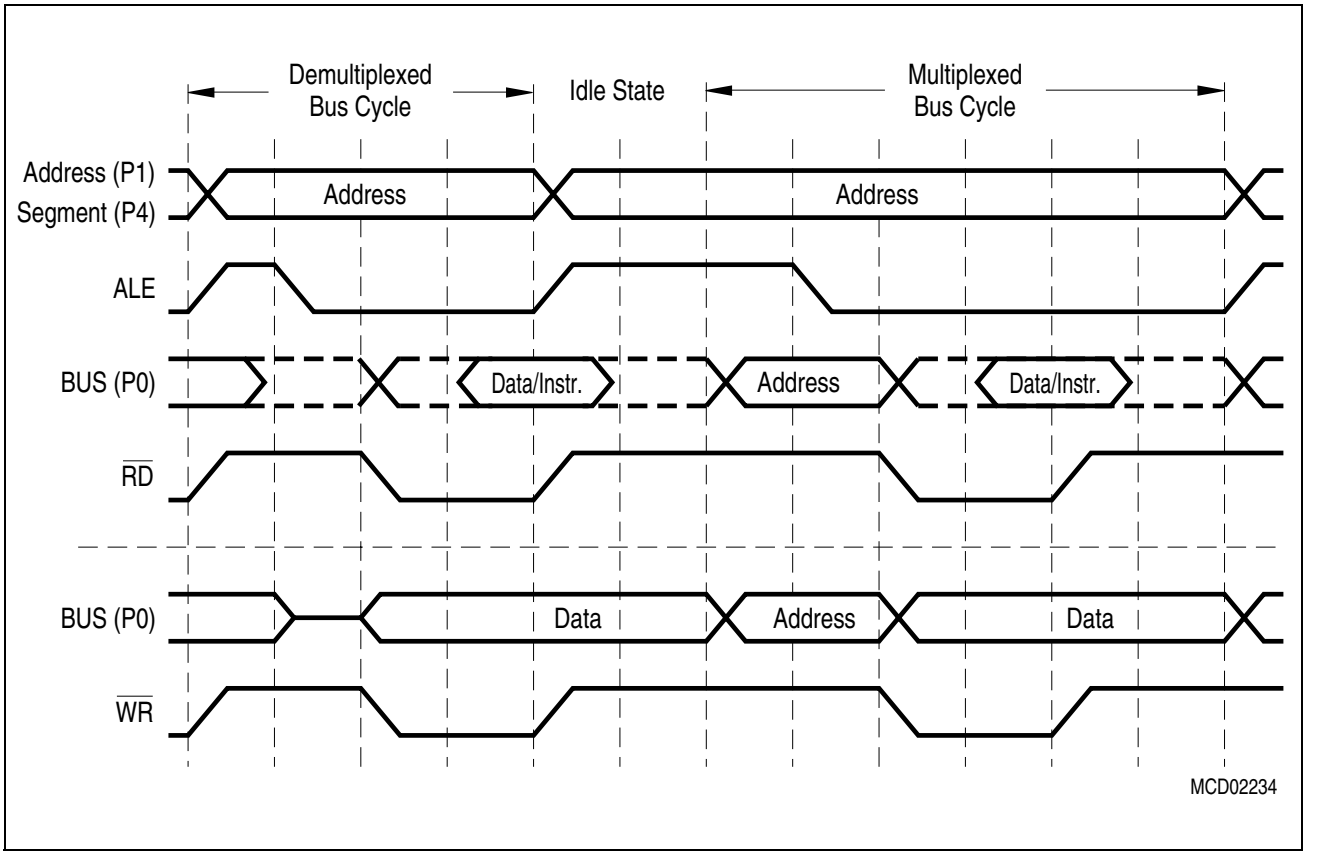


Figure 46 Switching from Demultiplexed to Multiplexed Bus Mode

External Data Bus Width

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. 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 $\overline{BHE}$ 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 $\overline{CS}$ input, but two $\overline{WR}$ enable inputs (for the two bytes), the EBC can directly generate these two write control signals. This saves the external combination of the $\overline{WR}$ signal with A0 or $\overline{BHE}$. In this case pin $\overline{WR}$ serves as $\overline{WRL}$ (write low byte) and pin $\overline{BHE}$ serves as $\overline{WRH}$ (write high

byte). Bit WRCFG in register SYSCON selects the operating mode for pins $\overline{WR}$ and $\overline{BHE}$. The respective byte will be written on both data bus halfs.

When reading bytes from an external 16-bit device, whole words may be read and the C161U 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 $\overline{BHE}$ and A0.

Bus Mode	**Transfer Rate (**Speed factor for byte/word/dword access**)**	**System Requirements**	**Free I/O 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 I/O, when none of the BUSCON registers selects a demultiplexed bus mode.

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

Bit BYTDIS is provided for controlling the active low Byte High Enable ($\overline{BHE}$) pin. The function of the $\overline{BHE}$ pin is enabled, if the BYTDIS bit contains a '0'. Otherwise, it is disabled and the pin can be used as standard I/O pin. The $\overline{BHE}$ pin is implicitly used by the External Bus Controller to select one of two byte-organized memory chips, which are connected to the C161U via a word-wide external data bus. After reset the $\overline{BHE}$ 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 $\overline{BHE}$ 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).

Note: The total accessible address space may be increased by accessing several banks which are distinguished by individual chip select signals.

SALSEL	Segment Address Lines	Directly accessible Address Space
1 1	Two: A17...A16	256 KByte (Default without pull-downs)
1 0	Seven: A20...A16	2 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	Four: $\overline{\text{CS3}}$...$\overline{\text{CS0}}$	Default without pull-downs
1 0	None	Port 6 pins free for I/O
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 (ie. 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, 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, mode for $\overline{\text{CS0}}$)
0	1	Read Chip Select

CSWENx	CSRENx	Chip Select Mode
1	0	Write Chip Select
1	1	Read/Write Chip Select

Address Chip Select signals remain active until an access to another address window. An address chip select becomes active with the falling edge of ALE and becomes inactive with the falling edge of ALE of an external bus cycle that accesses a different address area. No spikes will be generated on the chip select lines.

Read or Write Chip Select signals remain active only as long as the associated control signal ($\overline{RD}$ or $\overline{WR}$) is active. This also includes the programmable read/write delay. Read chip select is 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 $\overline{WRH}$ or $\overline{WRL}$ gets active). These modes save external glue logic, when accessing external devices like latches or drivers that only provide a single enable input.

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

Internal pullup devices hold all $\overline{CS}$ 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 $\overline{CS}$ lines. Not selected $\overline{CS}$ lines will enter the high-impedance state and are available for general purpose I/O.

The pullup devices are also active during bus hold on the selected $\overline{CS}$ lines, while $\overline{HLDA}$ is active and the respective pin is switched to push/pull mode. Open drain outputs will float during bus hold. In this case external pullup devices are required or the new bus master is responsible for driving appropriate levels on the $\overline{CS}$ lines.

Segment Address versus Chip Select

The external bus interface of the C161U supports many configurations for the external memory. By increasing the number of segment address lines the C161U can address a linear address space of 256 KByte, 1 MByte or 2 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 $\overline{CS}$ lines the C161U can access memory banks or peripherals without external glue logic. These two features may be combined to optimize the overall system performance. Enabling 4 segment address lines and 4 chip select lines eg. allows to access four memory banks of 2 MByte each. So the available address space is 8 MByte (without glue logic).

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

10.2 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:

- LE 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 acceses use the slowest possible bus cycle after reset. The bus cycle timing may then be optimized by the initialization software.

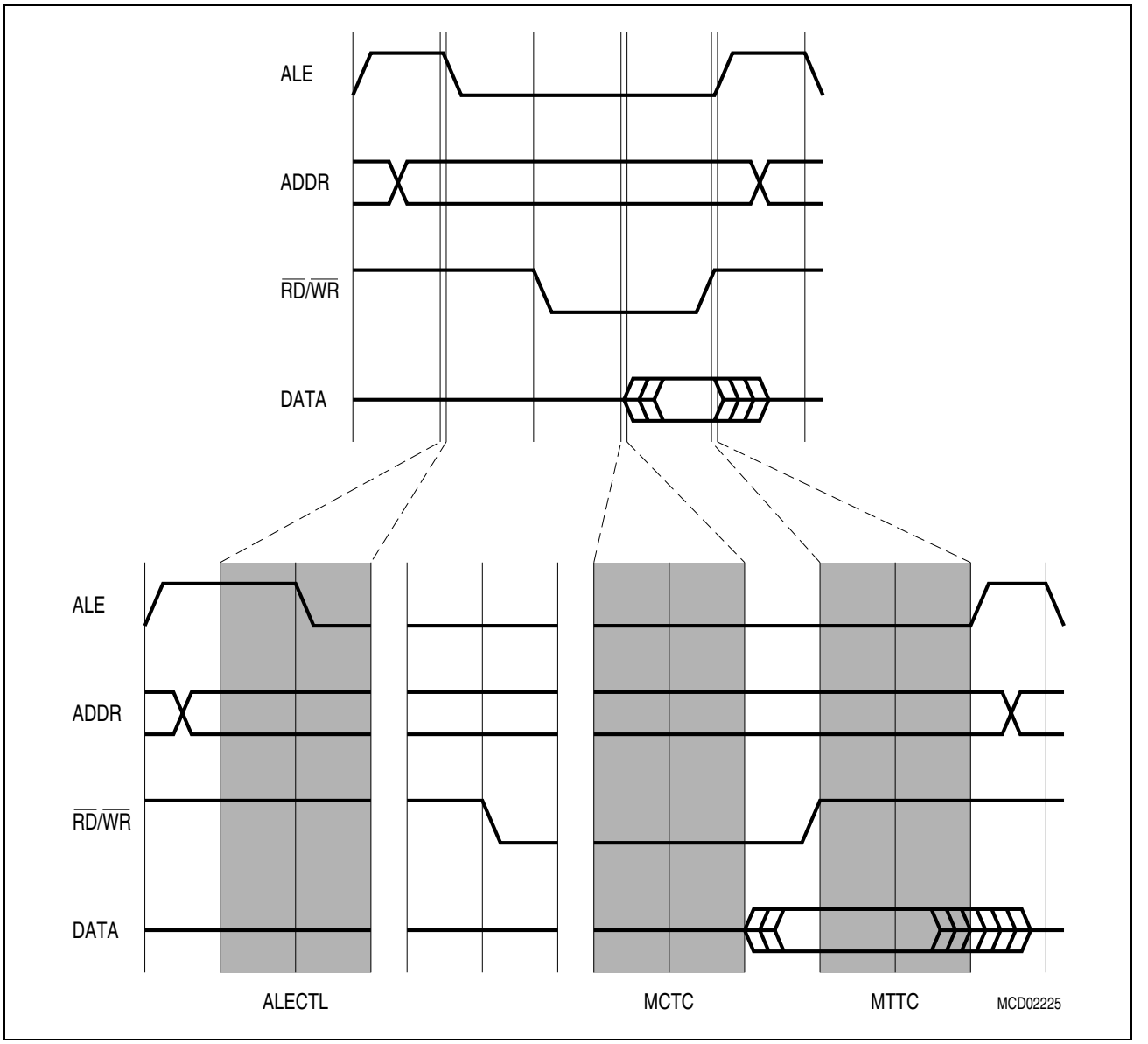


Figure 47 Programmable External Bus Cycle

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. Also the address hold time after the falling edge of ALE (on a multiplexed bus) will be prolonged by half a CPU clock, so the data transfer within a bus cycle refers to the same CLKOUT edges as usual (ie. 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.

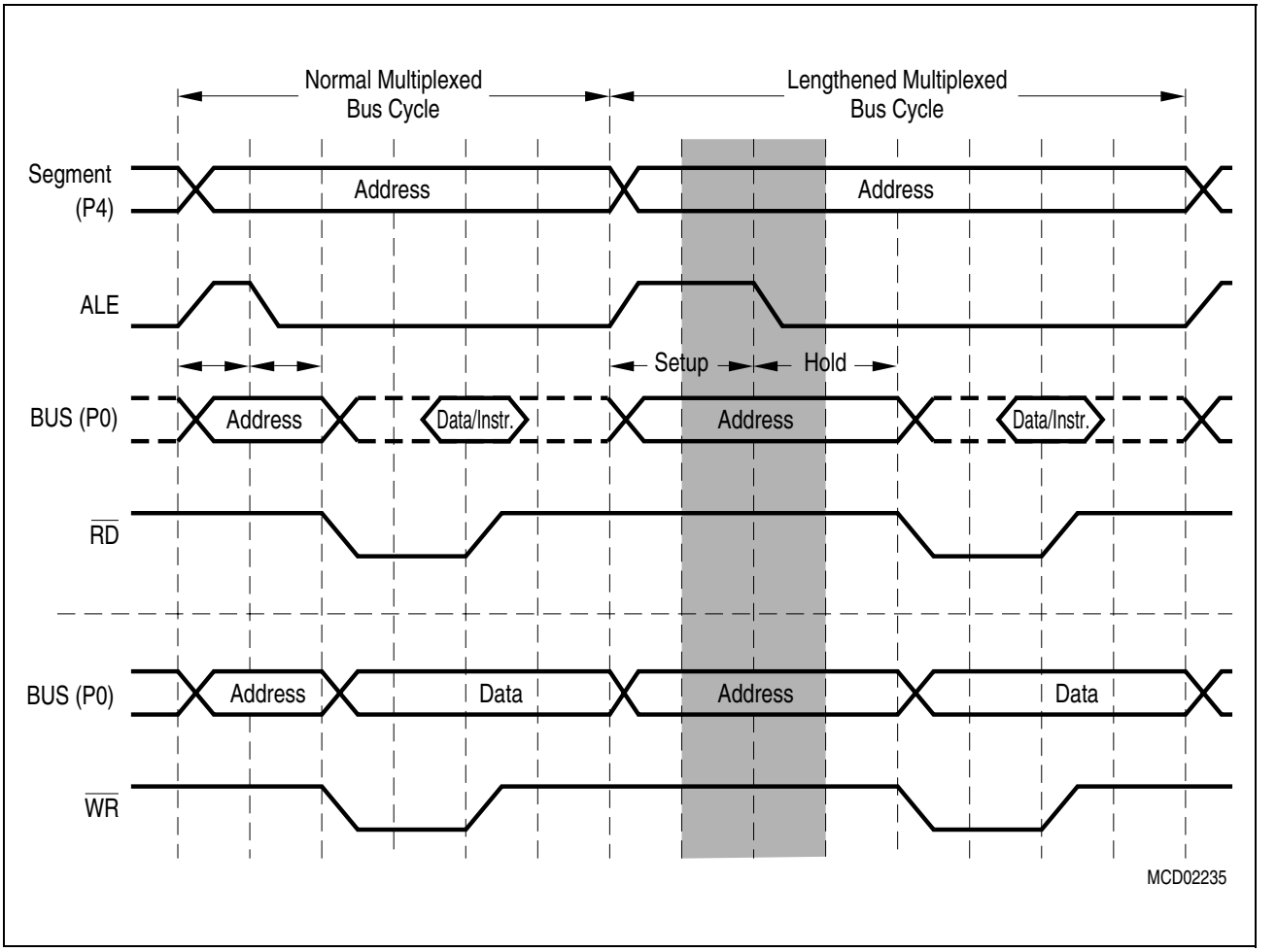


Figure 48 ALE Length Control

Programmable Memory Cycle Time

C161U 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.

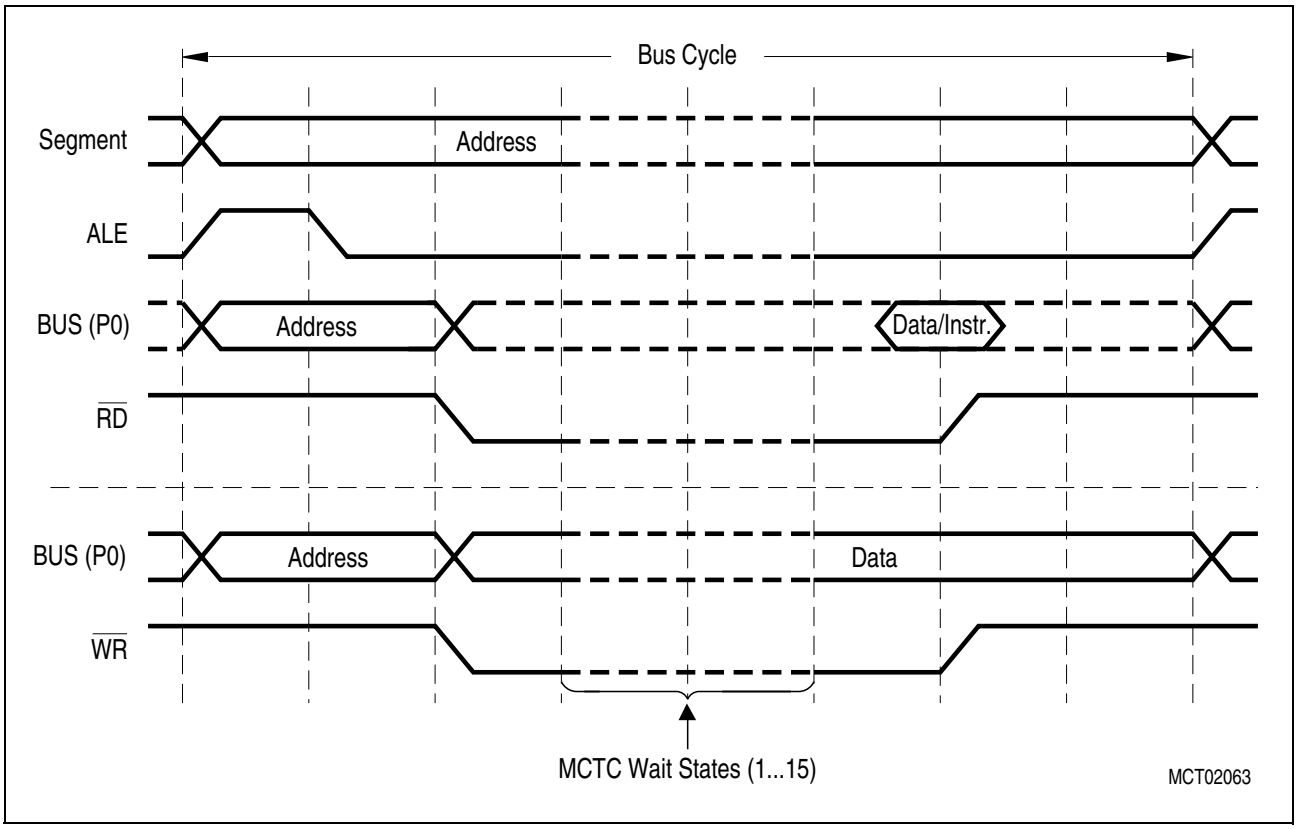


Figure 49 Memory Cycle Time

The external bus cycles of the C161U 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 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

C161U 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 ($\overline{RD}$).

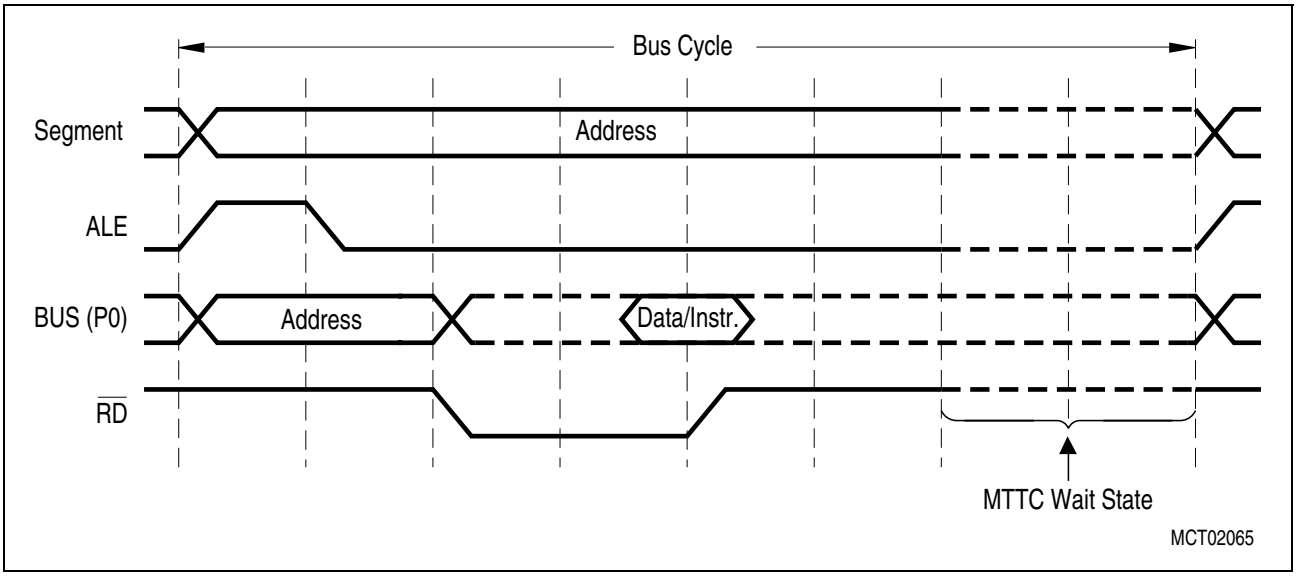


Figure 50 Memory Tri-State Time

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 (28 ns at f_{CPU} = 36 MHz) 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

C161U 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 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 C161U’s address, when the early $\overline{RD}$ signal is used. Therefore multiplexed bus cycles should always be programmed with read/write delay.

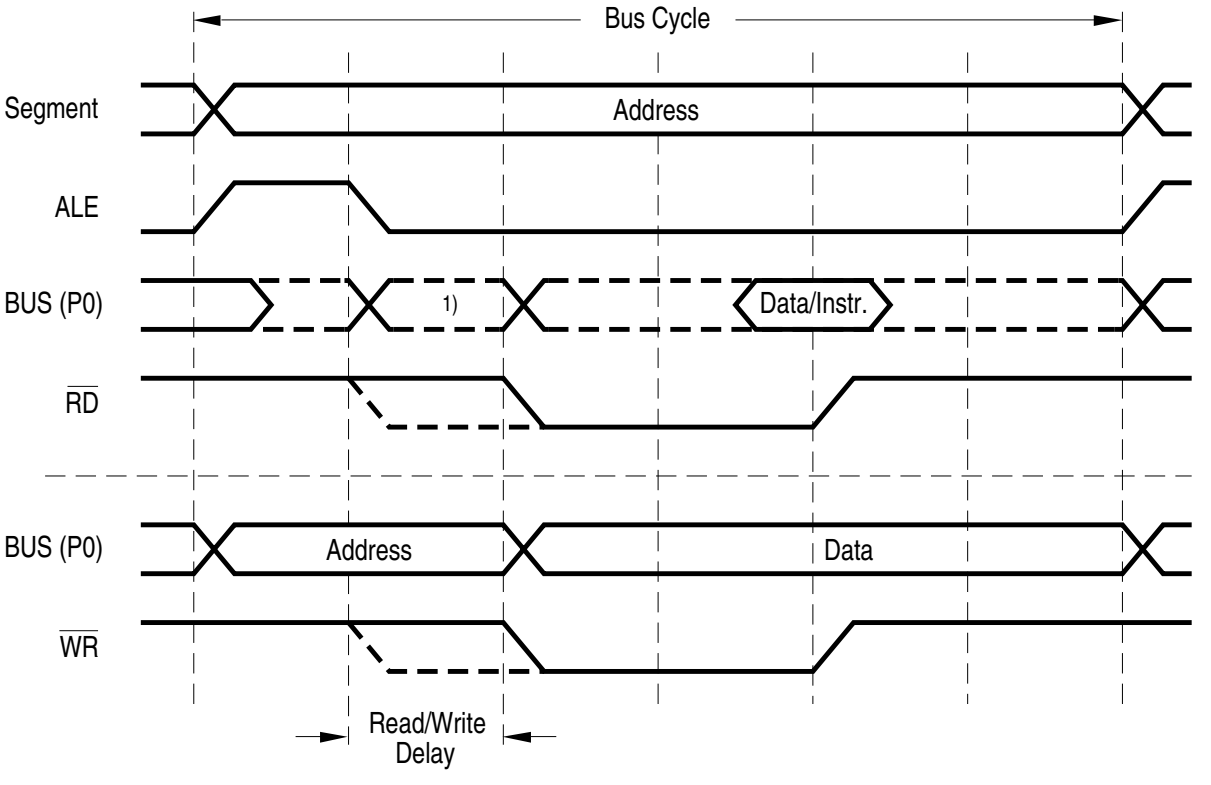


Figure 51 Read/Write Delay

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

Early $\overline{WR}$ Signal Deactivation

The duration of an external write access can be shortened by one TCL. The $\overline{WR}$ signal is activated (driven low) in the standard way, but can be deactivated (driven high) one TCL earlier than defined in the standard timing. In this case, also the data output drivers will be deactivated one TCL earlier.

This is especially useful in systems which operate on higher CPU clock frequencies and employ external modules (memories, peripherals, etc.) which switch on their own data drivers very fast in response to e.g. a chip select signal.

Conflicts between the C161U and the external peripheral’s output drivers can be avoided then by selecting early WR for the C161U.

Note: Make sure that the reduced $\overline{WR}$ low time then still matches the requirements of the external peripheral/memory.

Early $\overline{WR}$ deactivation is controlled via the EWENx bits in the BUSCON registers (see page 196). The $\overline{WR}$ signal will be shortened if bit EWENx is set to ’1’ signal. Default after reset is a standard $\overline{WR}$ signal (EWENx = ’0’).

10.3 $\overline{READY}$ 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 C161U provides external bus cycles that are terminated via a $\overline{READY}$ input signal (synchronous or asynchronous). In this case the C161U first inserts a programmable number of waitstates (0...7) and then monitors the $\overline{READY}$ line to determine the actual end of the current bus cycle. The external device drives $\overline{READY}$ low in order to indicate that data have been latched (write cycle) or are available (read cycle).

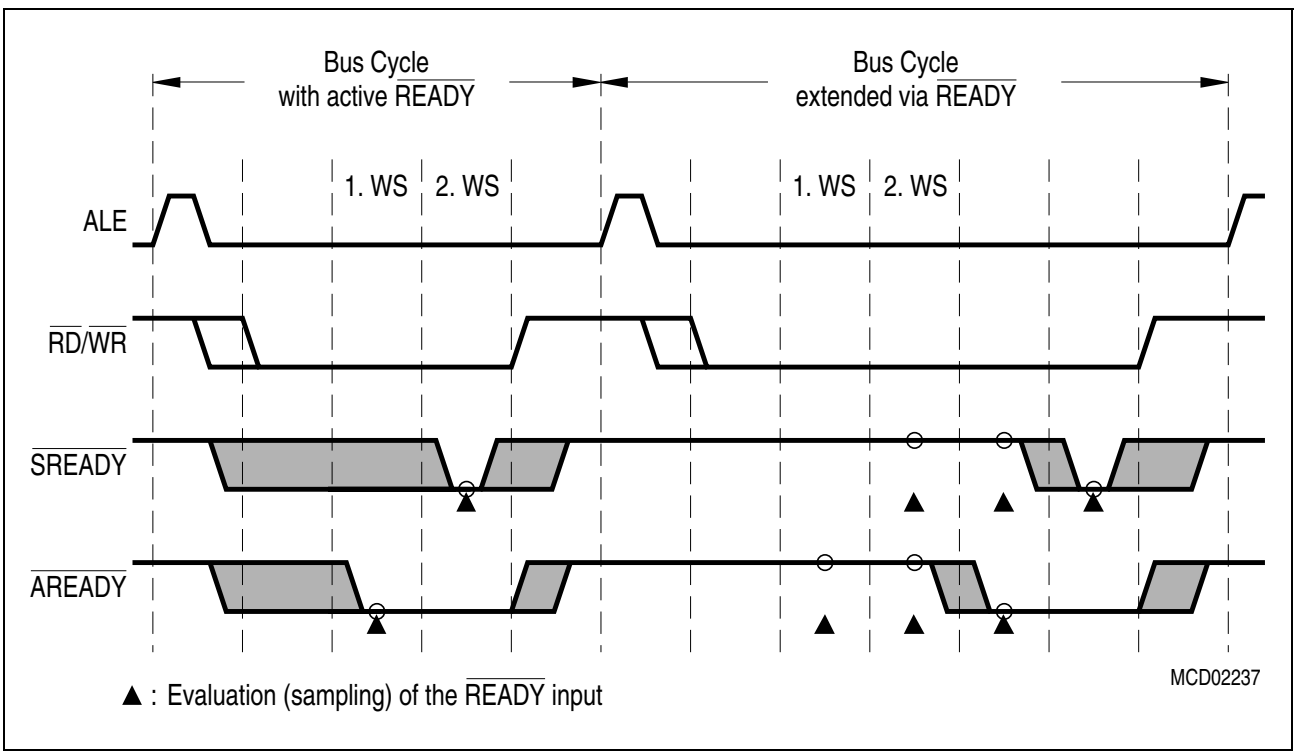


Figure 52 $\overline{READY}$ Controlled Bus Cycles

The $\overline{READY}$ 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 $\overline{READY}$ operation:

MCTC.3 = ‘0’: Synchronous $\overline{READY}$, ie. the $\overline{READY}$ signal must meet setup and hold times.
MCTC.3 = ‘1’: Asynchronous $\overline{READY}$, ie. the $\overline{READY}$ signal is synchronized internally.

Synchronous 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 READY timing in this case.

Asynchronous READY is less restrictive, but requires additional waitstates caused by the internal synchronization. As the asynchronous 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 READY signal (especially asynchronous READY) that has been activated by an external device may be deactivated in response to the trailing (rising) edge of the respective command (RD or WR).

Note: When the READY function is enabled for a specific address window, each bus cycle within this window must be terminated with an active READY signal. Otherwise the controller hangs until the next reset. A 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 READY may be grouped into the same address window. The (external) waitstate control logic in this case would activate READY either upon the memory’s chip select or with the peripheral’s READY output. After the predefined number of waitstates the C161U will check its 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 READY function with so-called “normally-ready” peripherals, it may lead to erroneous bus cycles, if the READY line is sampled too early. These peripherals pull their READY output low, while they are idle. When they are accessed, they deactivate READY until the bus cycle is complete, then drive it low again. If, however, the peripheral deactivates READY **after** the first sample point of the C161U, the controller samples an active READY and terminates the current bus cycle, which, of course, is too early. By inserting predefined waitstates the first READY sample point can be shifted to a time, where the peripheral has safely controlled the READY line (eg. after 2 waitstates in the figure above).

10.4 Controlling the External Bus Controller

A set of registers controls the functions of the EBC. General features like the usage of interface pins ($\overline{WR}$, $\overline{BHE}$), segmentation are controlled via register SYSCON.

Note: For SYSCON register description, refer to page 66.

The properties of a bus cycle like chip select mode, usage of $\overline{READY}$, 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.

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 eg. is also used for the very first code access after reset, is partly controlled by hardware, ie. 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: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSW EN0	CSR EN0	-	RDY EN0	-	BUS ACT0	ALE CTL0	EW EN0	BTYP		MTT C0	RWD C0	MCTC			
rw	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
CSW EN1	CSR EN1	-	RDY EN1	-	BUS ACT1	ALE CTL1	EW EN1	BTYP		MTT C1	RWD C1	MCTC			
rw	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
CSW EN2	CSR EN2	-	RDY EN2	-	BUS ACT2	ALE CTL2	EW EN2	BTYP		MTT C2	RWD C2	MCTC			
rw	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
CSW EN3	CSR EN3	-	RDY EN3	-	BUS ACT3	ALE CTL3	EW EN3	BTYP		MTT C3	RWD C3	MCTC			
rw	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
CSW EN4	CSR EN4	-	RDY EN4	-	BUS ACT4	ALE CTL4	EW EN4	BTYP		MTT C4	RWD C4	MCTC			
rw	rw	-	rw	-	rw	rw	rw	rw		rw	rw	rw			

Bit	Function
MCTCx	**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

Bit	Function
MTTCx	Memory Tristate Time Control ‘0’: 1 waitstate ‘1’: No waitstate
EWENx	Early Write Enable Bit ‘0’: Normal write ‘1’: Early write is enabled. The write signal turns off one TCL earlier. In order to have no overlapping with the following ALE signal, the write control signal is shortened by one TCL by setting bit EWEN.
BTYPx	External Bus Configuration 0 0 : 8-bit Demultiplexed Bus 0 1 : 8-bit Multiplexed Bus 1 0 : 16-bit Demultiplexed Bus 1 1 : 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

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.

Bus Access Control

CPU accesses to internal and external busses, e.g. to internal or external memories or peripherals, are controlled with the respective address ranges. These address ranges are supported by 'chip select' functions for XBUS resources or for external off-chip resources. In the C161U six address ranges with according bus definitions can be programmed for XBUS peripherals (including memories) and additionally five ranges for external bus peripherals.

Note: In contrast to previous Infineon devices the XADRS/XBCON registers are not hardwired but fully programmable.

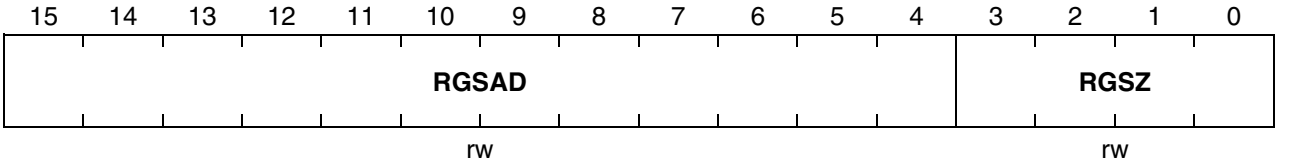
Address ranges and address mapping of memories or peripherals on XBUS or external bus are controlled with the address selection registers XADRSx for XBUS and ADDRSELx for external bus. The respective bus type definitions are controlled with registers XBCONx and BUSCONx.

In comparison to previous devices, C161U has 3 more address selection registers for XBUS:

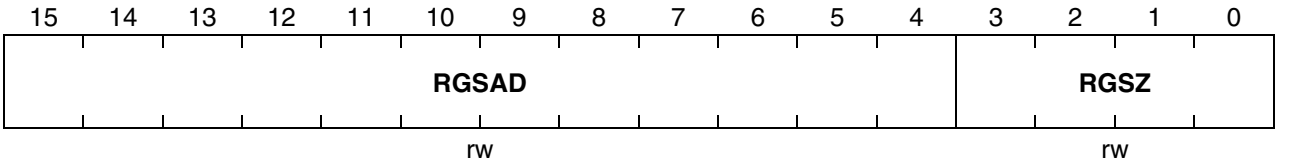
- The new register pair XADRS4 / XBCON4 use the same standard scheme of address selection and $\overline{\text{XCS}}$ control as the XADRS1-3 registers; smallest possible address range is 256 bytes.
- The new register pairs XADRS5 / XBCON5 and XADRS6 / XBCON6 control address selections as defined for external peripherals (as contolled by ADDRSEL); thus, mapping of XPER addresses to the total address space is provided, with smallest possible address range of 4 KBytes. XBCON5/6 and $\overline{\text{XCS}}$5/6 control are identical to the standard XBUS address ranges.

After reset, no address selection register is selected; thus the default address range is enabled and controlled with BUSCON0 and additionally the chip select output $\overline{\text{CS0}}$ is activated (as in standard C16x architecture).

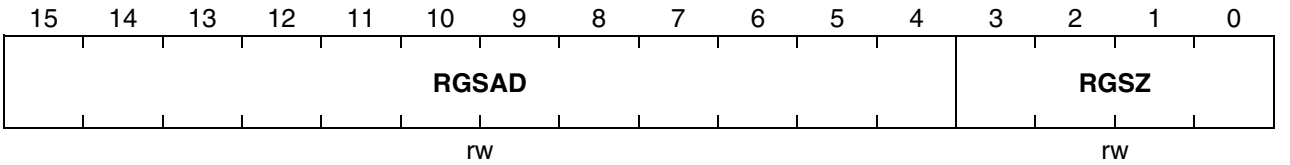
ADDRSEL1 ($FE18_H$ / $0C_H$) SFR **Reset Value: 0000_H**



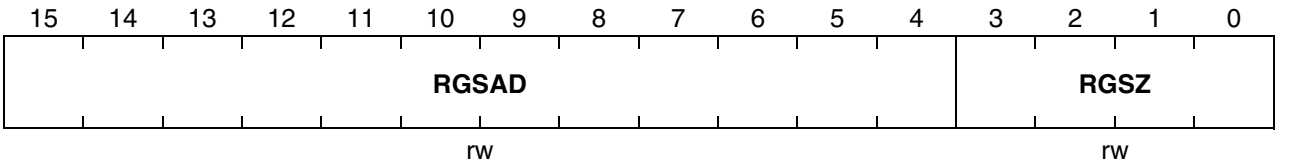
ADDRSEL2 ($FE1A_H$ / $0D_H$) SFR **Reset Value: 0000_H**



ADDRSEL3($FE1C_H$ / $0E_H$) SFR **Reset Value: 0000_H**



ADDRSEL4 ($FE1E_H$ / $0F_H$) SFR **Reset Value: 0000_H**



Bit	Function
RGSZ	Range Size Selection
RGSAD	Range Start Address

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.

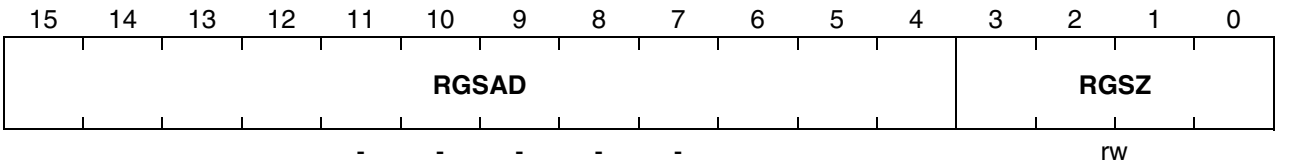
Definition of Address Areas

The four register pairs BUSCON4/ADDRSEL4...BUSCON1/ADDRSEL1 allow to define 4 separate address areas within the address space of the C161U. 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 (A20...A12)
0 0 0 0	4 KBytes	R R R R R R R R R
0 0 0 1	8 KBytes	R R R R R R R R x
0 0 1 0	16 KBytes	R R R R R R R x x
0 0 1 1	32 KBytes	R R R R R R x x x
0 1 0 0	64 KBytes	R R R R R x x x x
0 1 0 1	128 KBytes	R R R R x x x x x
0 1 1 0	256 KBytes	R R R x x x x x x
0 1 1 1	512 KBytes	R R x x x x x x x
1 0 0 0	1 MBytes	R x x x x x x x x
1 0 0 1	2 MBytes	x x x x x x x x x
1 0 1 0	Reserved	
1 0 1 1	Reserved	
1 1 x x	Reserved.	

XADRS1(2/3/4/5/6) (F014$_H$ / 0A$_H$) **ESFRx** **Reset Value: 0000$_H$**



Bit	Function
RGSAD	Address Range Start Address Selection
RGSZ	Address Range Size Selection

The respective SFR addresses of XADRS registers can be found in list of SFRs.

Due to the different range size options, address mapping of XPERs is possible only within the first MByte of the total address range if XADRS1 to XADRS4 is used. The upper four address lines (A23:A20) are set to zero. Note that the range start address can be only on boundaries specified by the selected range size.

The following tables show the different definitions of range size selections and range start addresses for the two types of address selections:

Range Size RGSZ	Selected Address Range	Relvant(R) bits of RGSAD	Selected Range Start Address (Relevant(R) bits of RGSAD)
0000	256 Byte	RRRR RRRR RRRR	0000 RRRR RRRR RRRR 0000 0000
0001	512 Bytes	RRRR RRRR RRR0	0000 RRRR RRRR RRR0 0000 0000
0010	1 KB	RRRR RRRR RR00	0000 RRRR RRRR RR00 0000 0000
0011	2 KB	RRRR RRRR R000	0000 RRRR RRRR R000 0000 0000
0100	4 KB	RRRR RRRR 0000	0000 RRRR RRRR 0000 0000 0000
0101	8 KB	RRRR RRR0 0000	0000 RRRR RRR0 0000 0000 0000
0110	16 KB	RRRR RR00 0000	0000 RRRR RR00 0000 0000 0000
0111	32 KB	RRRR R000 0000	0000 RRRR R000 0000 0000 0000
1000	64 KB	RRRR 0000 0000	0000 RRRR 0000 0000 0000 0000
1001	128 KB	RRR0 0000 0000	0000 RRR0 0000 0000 0000 0000
1010	256 KB	RR00 0000 0000	0000 RR00 0000 0000 0000 0000
1011	512 KB	R000 0000 0000	0000 R000 0000 0000 0000 0000
11xx	- reserved		

Table 32 Address Range and Address Range Start Definition of XADRS1/2/3/4 register

Range Size RGSZ	Selected Address Range	Relvant(R) bits of RGSAD	Selected Range Start Address (Relevant(R) bits of RGSAD)
0000	4 KB	RRRR RRRR RRRR	RRRR RRRR RRRR 0000 0000 0000
0001	8 KB	RRRR RRRR RRR0	RRRR RRRR RRR 0 0000 0000 0000
0010	16 KB	RRRR RRRR RR00	RRRR RRRR RR 00 0000 0000 0000
0011	32 KB	RRRR RRRR R000	RRRR RRRR R 000 0000 0000 0000
0100	64 KB	RRRR RRRR 0000	RRRR RRRR 0000 0000 0000 0000
0101	128 KB	RRRR RRR0 0000	RRRR RRR 0 0000 0000 0000 0000
0110	256 KB	RRRR RR00 0000	RRRR RR 00 0000 0000 0000 0000
0111	512 KB	RRRR R000 0000	RRRR R 000 0000 0000 0000 0000
1000	1 MB	RRRR 0000 0000	RRRR 0000 0000 0000 0000 0000
1001	2 MB	RRR0 0000 0000	RRR 0 0000 0000 0000 0000 0000
1010	4 MB	RR00 0000 0000	RR 00 0000 0000 0000 0000 0000
1011	8 MB	R000 0000 0000	R 000 0000 0000 0000 0000 0000
11xx	- reserved		

Table 33 Address Range and Address Range Start Definition of XADRS5/6 register

The XBCONx registers are defined as follows:

XBCON/2/3 ($F114_H$ / $8A_H$) **ESFR-b** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	RDY ENx	BS WCx	BUS ACTx	ALE CTLx	EW ENx	BTYPx		MT TCx	RW DCx	MCTCx			
-	-	-	rw	rw	rw	rw	rw	rw		rw	rw	rw			

Bit	Function
MCTCx	**Memory Cycle Time Control** (see BUSCON)
RWDCx	**READ/WRITE Delay Control** (see BUSCON)
MTTCx	**Memory Tri-state Time Control** (see BUSCON)
BTYPx	**Bus Type Selection**; only demultiplexed busses are supported on XBUS; '00': 8 bit bus '10': 16 bit bus; 'x1': reserved.
EWENx	Early Write Enable '0': Standard write enable signal control '1': Write active state is disabled one TCL earlier
ALECTLx	**ALE Lengthening Control Bit** (see BUSCON)
BUSACTx*	Bus Active Control '0': XBUS (peripheral) disabled '1': XBUS (peripheral) enabled Enables the XBUS and the according chip select $\overline{\text{XCSx}}$ for the respective address window (respective XBUS peripheral), selected with according XADRSx window; after reset, all address windows on XBUS are disabled. *not used in FC-Cores, where XBCON is hardwired.
BSWCx	BUSCON Switch Control '0': Standard switch of bustype (switch of XBCON) '1': A bus wait state (Tri-state cycle) is included after execution of last old-bustype cycle and before the first new-bustype cycle after switch of XBCON or BUSCON; the BSWC bit is indicated in the old-bustype XBCON/BUSCON.
RDYENx	READY Enable '0': The bus cycle length is controlled by the bus controller using MCTC '1': The bus cycle length is controlled by the peripheral using READY

Note: The 'BUSCON switch control' BSWC is a new function, which is necessary due to the execution with higher frequencies, to avoid bus collisions on data bus in case of peripheral change (see BUSCON).

Note: All XADRSx/ADDRSELx registers as well as XBCONx/BUSCONx registers are user programmable SFR registers. All BUSCONx registers are mapped into the bitaddressable SFR memory space, all XBCONx registers are located in the bitaddressable ESFR memory space. Although they are free programmable, programming should be performed during the initialisation phase before the first accesses are controlled with XBCONx or BUSCONx.

Note: The respective SFR addresses of XBCON registers can be found in list of SFRs.

Note: Within the C161U, register XBCON2 is related to the USB module and register XBCON2 is related to the EPEC module. For configuration, please also refer to Chapter 10.8, "Initialization of the C161U's X-peripherals" on page 212.

Address Window Arbitration

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 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.

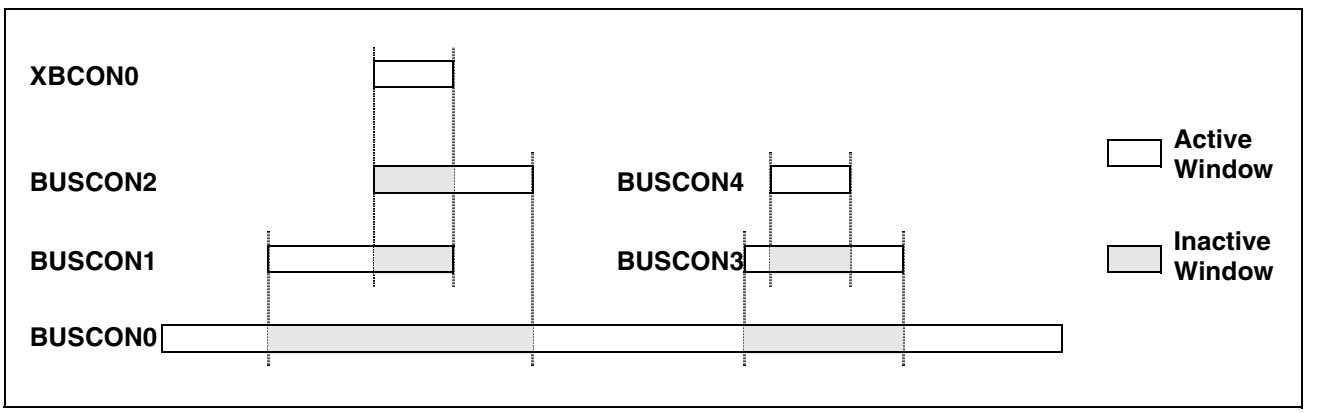


Figure 53 Address Window Arbitration

Note: Only the indicated overlaps are defined. All other overlaps lead to erroneous bus cycles. Eg. 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 WR and BHE operate as WRL and WRH signals 1: Pins WR and BHE operate as WR and BHE signals
CSSEL	**Chip Select Line Selection** (Number of active CS outputs) 0 0: 3 CS lines: CS2...CS0 0 1: 2 CS lines: CS1...CS0 1 0: No CS lines at all 1 1: 4 CS lines: CS3...CS0 (Default without pulldowns)
SALSEL	**Segment Address Line Selection** (Number of active segment address outputs) 0 0: 4-bit segment address: A19...A16 0 1: No segment address lines at all 1 0: 5-bit segment address: A20...A16 1 1: 2-bit segment address: A17...A16 (Default without pulldowns)
CLKCFG	Clock Generation Mode Configuration These pins define the clock generation mode, ie. the mechanism how the 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 areas 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 areas 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).

10.5 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, $\overline{BHE}$ and ALE which also pulses for an XBUS cycle. The external $\overline{CS}$ signals on Port 6 are driven inactive (high) because the EBC switches to an internal $\overline{XCS}$ signal.

The **external control signals** ($\overline{RD}$ and $\overline{WR}$ or $\overline{WRL}$/$\overline{WRH}$ if enabled) **remain inactive** (high).

Table 34 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 $\overline{CS}$ signal corresponding to last used address	Inactive (high) for selected $\overline{CS}$ signals
BHE	Level corresponding to last external access	Level corresponding to last XBUS access
ALE	Inactive (low)	Pulses as defined for X-Peripheral
RD	Inactive (high)	Inactive (high)
WR/WRL	Inactive (high)	Inactive (high)
WRH	Inactive (high)	Inactive (high)

10.6 External Bus Arbitration

In high performance systems it may be efficient to share external resources like memory banks or peripheral devices among more than one controller. C161U supports this approach with the possibility to arbitrate the access to its external bus, ie. to the external devices.

This bus arbitration allows an external master to request the C161U's bus via the HOLD input. C161U acknowledges this request via the HLDA output and will float its bus lines in this case. The CS outputs provide internal pullup devices. The new master may now access the peripheral devices or memory banks via the same interface lines as the C161U. During this time the C161U can keep on executing, as long as it does not need access to the external bus. All actions that just require internal resources like instruction or data memory and on-chip peripherals, may be executed in parallel.

When the C161U needs access to its external bus while it is occupied by another bus master, it demands it via the BREQ output.

The external bus arbitration is enabled by setting bit HLDEN in register PSW to '1'. In this case the three bus arbitration pins HOLD, HLDA and BREQ are automatically controlled by the EBC independent of their I/O configuration. Bit HLDEN may be cleared during the execution of program sequences, where the external resources are required but cannot be shared with other bus masters. In this case the C161U will not answer to HOLD requests from other external masters. If HLDEN is cleared while the C161U is in Hold State (code execution from internal RAM) this Hold State is left only after HOLD has been deactivated again. Ie. in this case the current Hold State continues and only the next HOLD request is not answered.

Connecting eg. two C161Us in this way would require additional logic to combine the respective output signals HLDA and BREQ. This can be avoided by switching one of the controllers into Slave Mode where pin HLDA is switched to input. This allows to directly connect the slave controller to another master controller without glue logic. The Slave Mode is selected by setting bit DP6.7 to '1'. DP6.7='0' (default after reset) selects the Master Mode.

Note: The pins HOLD, HLDA and BREQ keep their alternate function (bus arbitration) even after the arbitration mechanism has been switched off by clearing HLDEN. All three pins are used for bus arbitration after bit HLDEN was set once.

Connecting Bus Masters

When multiple C161Us or a C161U and another bus master shall share external resources some glue logic is required that defines the currently active bus master and also enables a C161U which has surrendered its bus interface to regain control of it in case it must access the shared external resources. This glue logic is required if the „other" bus master does not automatically remove its hold request after having used the shared resources.

When two C161Us are to be connected in this way the external glue logic can be left out. In this case one of the controllers must be operated in its Master Mode (default after reset, DP6.7=’0’) while the other one must be operated in its Slave Mode (selected with DP6.7=’1’).

In Slave Mode the C161U inverts the direction of its HLDA pin and uses it as an input, while the master’s HLDA pin remains an output. This approach does not require any additional glue logic for the bus arbitration (see figure below).

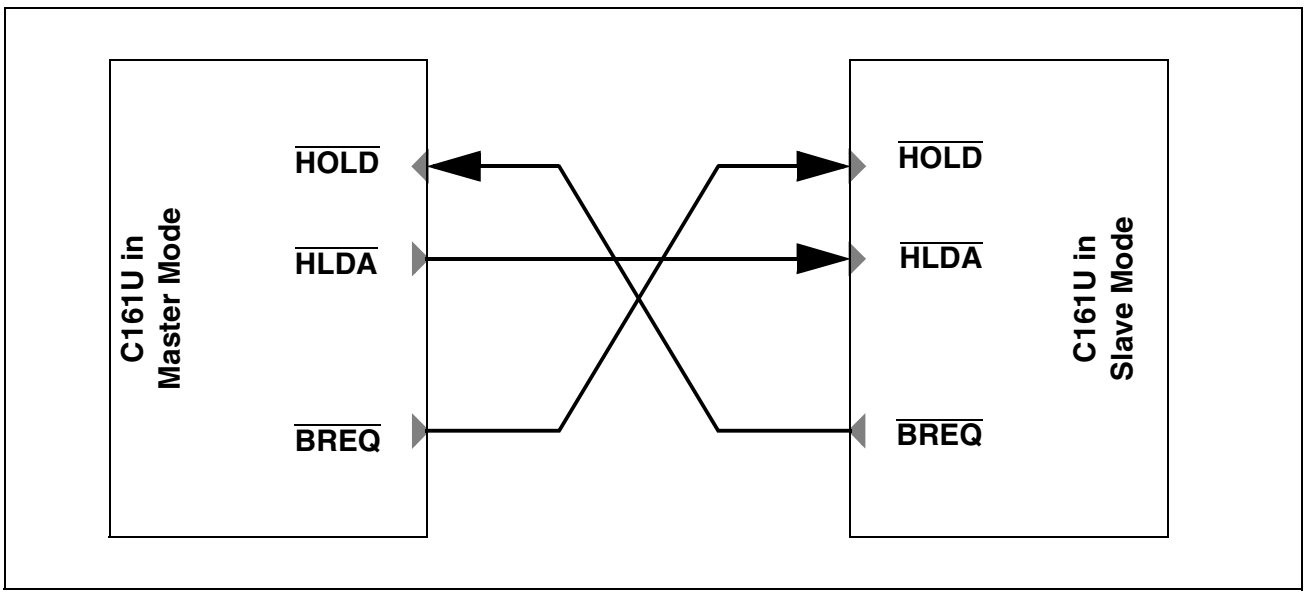


Figure 54 Sharing External Resources using Slave Mode

When the bus arbitration is enabled (HLDEN=’1’) the three corresponding pins are automatically controlled by the EBC. Normally the respective port direction register bits retain their reset value which is ’0’. This selects Master Mode where the device operates compatible with earlier versions. Slave Mode is enabled by intentionally switching pin BREQ to output (DP6.7=’1’) which is neither required for Master Mode nor for earlier devices.

Entering the Hold State

Access to the C161U’s external bus is requested by driving its HOLD input low. After synchronizing this signal the C161U will complete a current external bus cycle (if any is active), release the external bus and grant access to it by driving the HLDA output low. During hold state the C161U treats the external bus interface as follows:

- Address and data bus(es) float to tri-state
- ALE is pulled low by an internal pulldown device
- Command lines are pulled high by internal pullup devices (RD, WR/WRL, BHE/WRH)
- CSx outputs are pulled high (push/pull mode) or float to tri-state (open drain mode)

Should the C161U require access to its external bus during hold mode, it activates its bus request output $\overline{\text{BREQ}}$ to notify the arbitration circuitry. $\overline{\text{BREQ}}$ is activated only during hold mode. It will be inactive during normal operation.

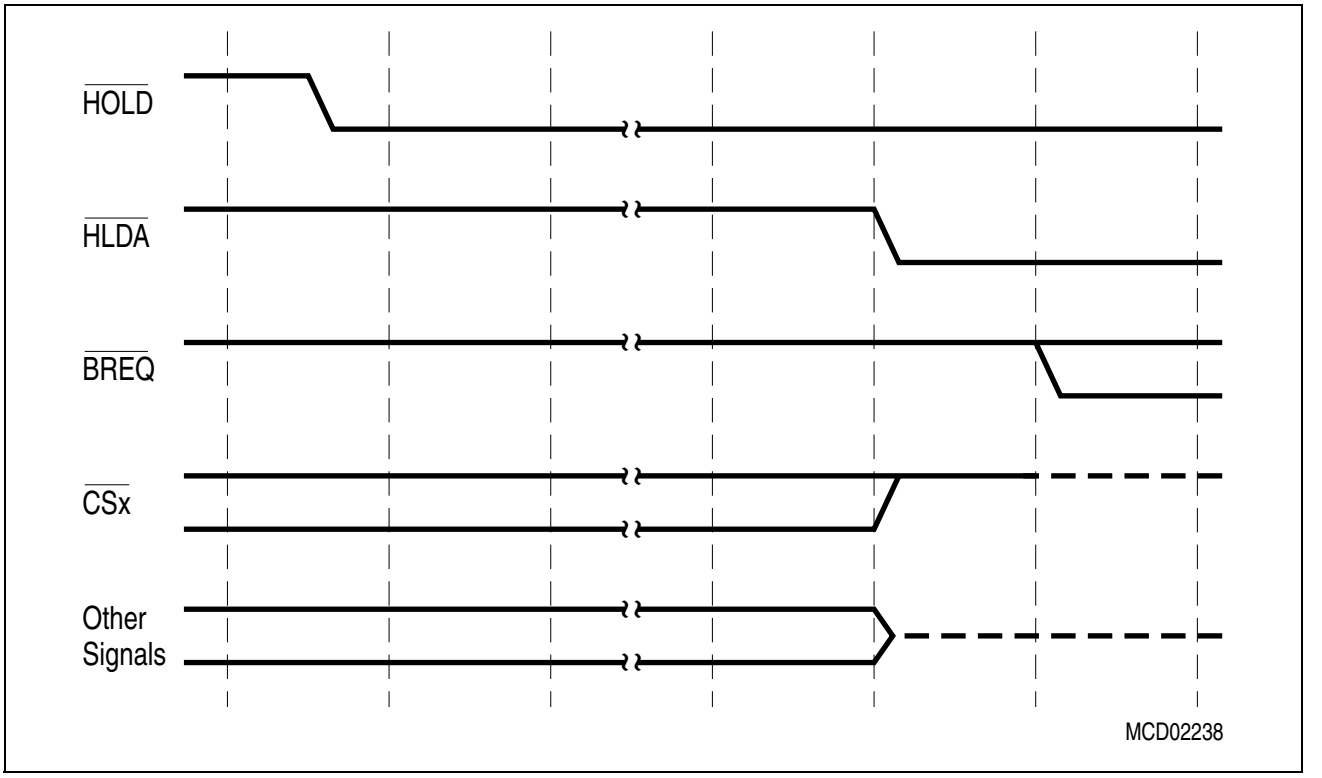


Figure 55 External Bus Arbitration, Releasing the Bus

Note: C161U will complete the currently running bus cycle before granting bus access as indicated by the broken lines. This may delay hold acknowledge compared to this figure.
The figure above shows the first possibility for $\overline{\text{BREQ}}$ to get active.
During bus hold pin P3.12 is switched back to its standard function and is then controlled by DP3.12 and P3.12. Keep DP3.12 = ’0’ in this case to ensure floating in hold mode.

Exiting the Hold State

The external bus master returns the access rights to the C161U by driving the $\overline{\text{HOLD}}$ input high. After synchronizing this signal the C161U will drive the $\overline{\text{HLDA}}$ output high, actively drive the control signals and resume executing external bus cycles if required.

Depending on the arbitration logic, the external bus can be returned to the C161U under two circumstances:

• The external master does no more require access to the shared resources and gives up its own access rights, or

• C161U needs access to the shared resources and demands this by activating its $\overline{\text{BREQ}}$ output. The arbitration logic may then deactivate the other master's $\overline{\text{HLDA}}$ and so free the external bus for the C161U, depending on the priority of the different masters.

Note: The Hold State is not terminated by clearing bit HLDEN.

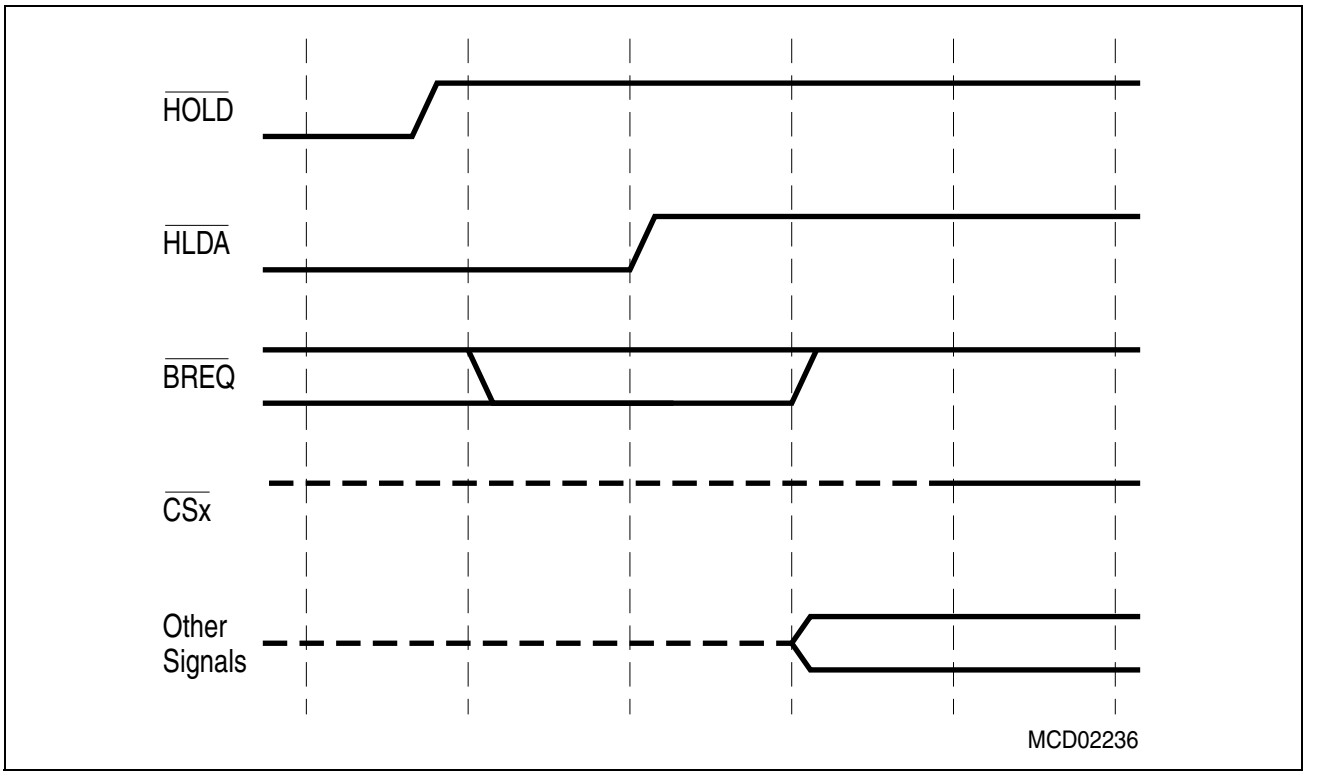


Figure 56 External Bus Arbitration, (Regaining the Bus)

Note: The falling $\overline{\text{BREQ}}$ edge shows the last chance for $\overline{\text{BREQ}}$ to trigger the indicated regain-sequence. Even if $\overline{\text{BREQ}}$ is activated earlier the regain-sequence is initiated by $\overline{\text{HOLD}}$ going high. $\overline{\text{BREQ}}$ and $\overline{\text{HOLD}}$ are connected via an external arbitration circuitry. Please note that $\overline{\text{HOLD}}$ may also be deactivated without the C161U requesting the bus.

10.7 XBUS Interface

C161U 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, ie. it is operated in the same way.

The current XBUS interface is prepared to support up to 3 X-Peripherals.

For each peripheral on the XBUS (X-Peripheral) there is a separate address window controlled by an XBCON and an XADRS register. As an interface to a peripheral in many cases is represented by just a few registers, the XADRS registers 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 and configuration pins (on PORT0) are provided for X-Peripherals to be integrated.

10.8 Initialization of the C161U's X-peripherals

The following registers must be set for initialization of the C161U X-peripherals:

XPERCON-Register (Addr. F024, default: 0000):

Bit 6:	'1': USB module active	'0': USB module switched-off
Bit 7:	'1': EPEC active	'0': EPEC switched-off

SYSCON-Register (Addr. FF12, default: 0xx0):

Bit 2:	'1': X-Peripherals enable	'0': X-Peripherals disable
Bit 1:	'1': X-Per accesses visible at externalXBUS	'0': X-Per accesses not visible

SYSCON3-Register (Addr. F1D4, default: 0000):

Bit 15:	'1': All peripheral clocks disabled	'0': Individual disable control by bits 14 thru 0
Bit 12:11:	'00': USB transceiver in normal operation '01': Suspend mode, differential USB receiver switched off '10': Reserved, do not use this combination '11': USB transceiver switched off - full power down mode	
Bit 8:	'1': Disable EPEC clock	'0': Enable EPEC clock
Bit 7:	'1': Disable USB clock	'0': Enable USB clock
Bit 3:	'1': Disable GPT12 clock	'0': Enable GPT12 clock
Bit 2:	'1': Disable SSC clock	'0': Enable SSC clock
Bit 1:	'1': Disable ASC clock	'0': Enable ASC clock
Bit 0:	'1': Disable RTC clock	'0': Enable RTC clock

XBCON1-Register (Addr.F114, default: 0000_H):

This register is not used. Must be set to '0000_H').

XBCON2-Register (Addr.F116, default: 0000_H):

Definition of the USB bus protocol.
Must be set to '$048x_H$'. Recommended using 0 waitstates: '$048F_H$'.

XBCON3-Register (Addr.F118, default: 0000_H):

Definition of the EPEC bus protocol.
Must be set to '$048x_H$'. Recommended using 0 waitstates: '$048F_H$'.

XADRS2-Register (Addr.F016, default: UUUU):
Definition of the USB address space.
Should be initialized with ’$0EE0_H$’ before writing to XBCON2.

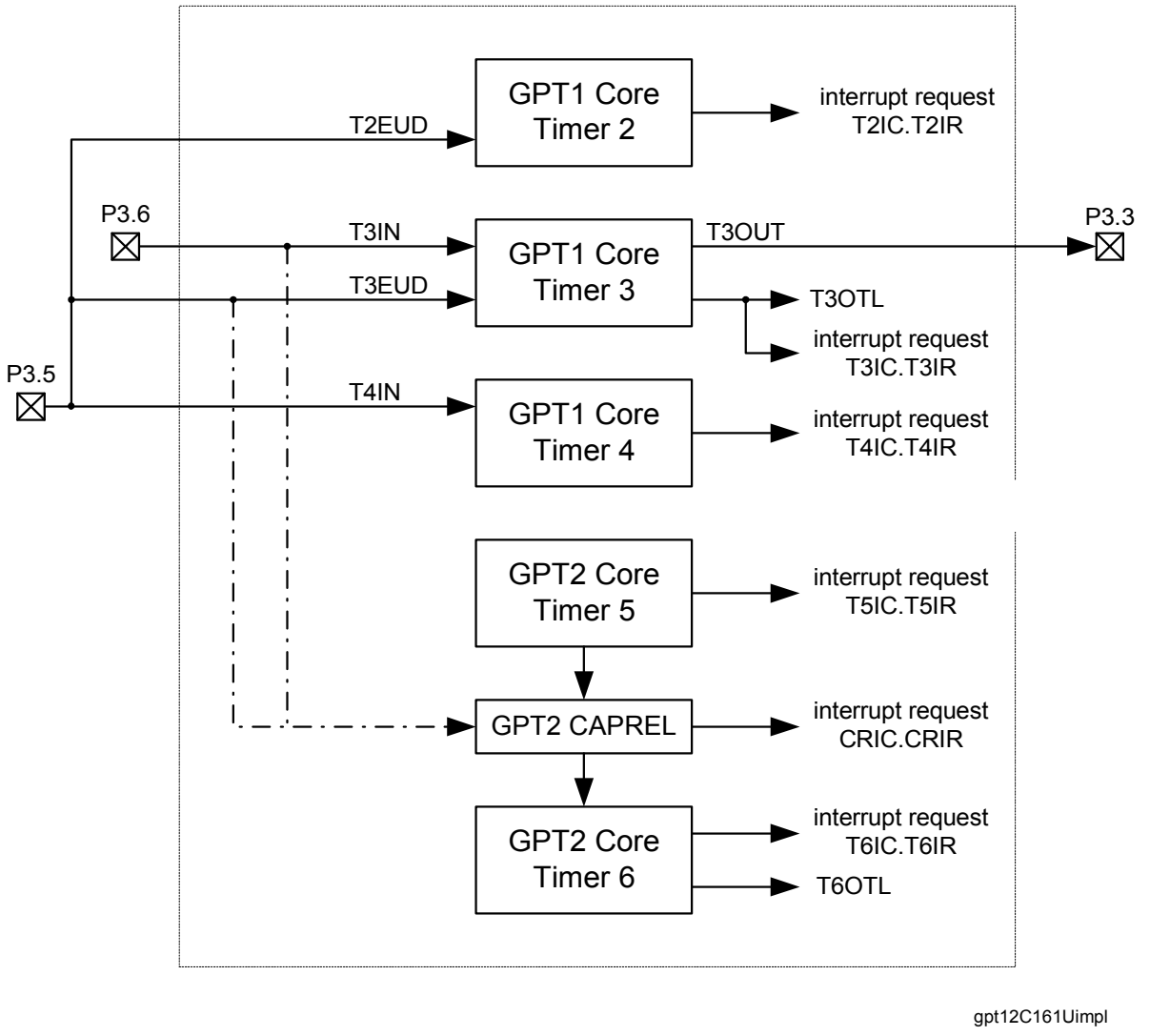
XADRS3-Register (Addr.F018, default: UUUU):
Definition of the EPEC address space.
Should be initialized with ’$0ED0_H$’ before writing to XBCON3.

Note: Bits (12:11) of register SYSCON3 are related to the analog USB transceiver only and not to the digital USB module.

11 General Purpose Timer Unit

General Purpose Timer Unit (GPT) represents very flexible multifunctional timer structures which may be used for timing, event counting, pulse width measurement, pulse generation, frequency multiplication, and other purposes.

In the C161U, there are following alternate function pins available: P3.3/T3OUT, P3.5/ T4IN/T3EUD/T2EUD and P3.6/T3IN, as shown in **Figure 57**.



Note: In the C161U, there are no external pins connected to Timer 5 and Timer 6. Additional, there is no second external connection to Timer 4 (T4EUD), no second external connection to Timer 2 (T2IN) and no external connection to the register GPT2 CAPREL.

Figure 57 GPT module external pins

GPT incorporate five 16-bit timers that are grouped into the two timer blocks GPT1 and GPT2. Each timer in each block may operate independently in a number of different modes such as gated timer or counter mode, or may be concatenated with another timer of the same block.

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

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

The following enumeration summarizes all features to be supported:

Timer Block 1:

$f_{Timer}/4$ maximum resolution.

3 independent timers/counters.

Timers/counters can be concatenated.

4 operating modes (timer, gated timer, counter, incremental).

Separate interrupt nodes.

Timer Block 2:

$f_{Timer}/2$ maximum resolution.

2 independent timers/counters.

Timers/counters can be concatenated.

2 operating modes (timer, counter).

Capture/reload functions via 16-bit Capture/Reload register CAPREL.

Separate interrupt nodes.

11.1 Kernel Description

11.1.1 Functional Description of Timer Block 1

All three timers of block 1 (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.

The input line (TxIN) associated with it which serves as the gate control in gated timer mode, or as the count input in counter mode. The count direction (Up / Down) may be programmed via software or may be dynamically altered by a signal at an external control input line. An overflow/underflow of core timer T3 is indicated by the output toggle latch T3OTL whose state may be output on related line T3OUT.

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.

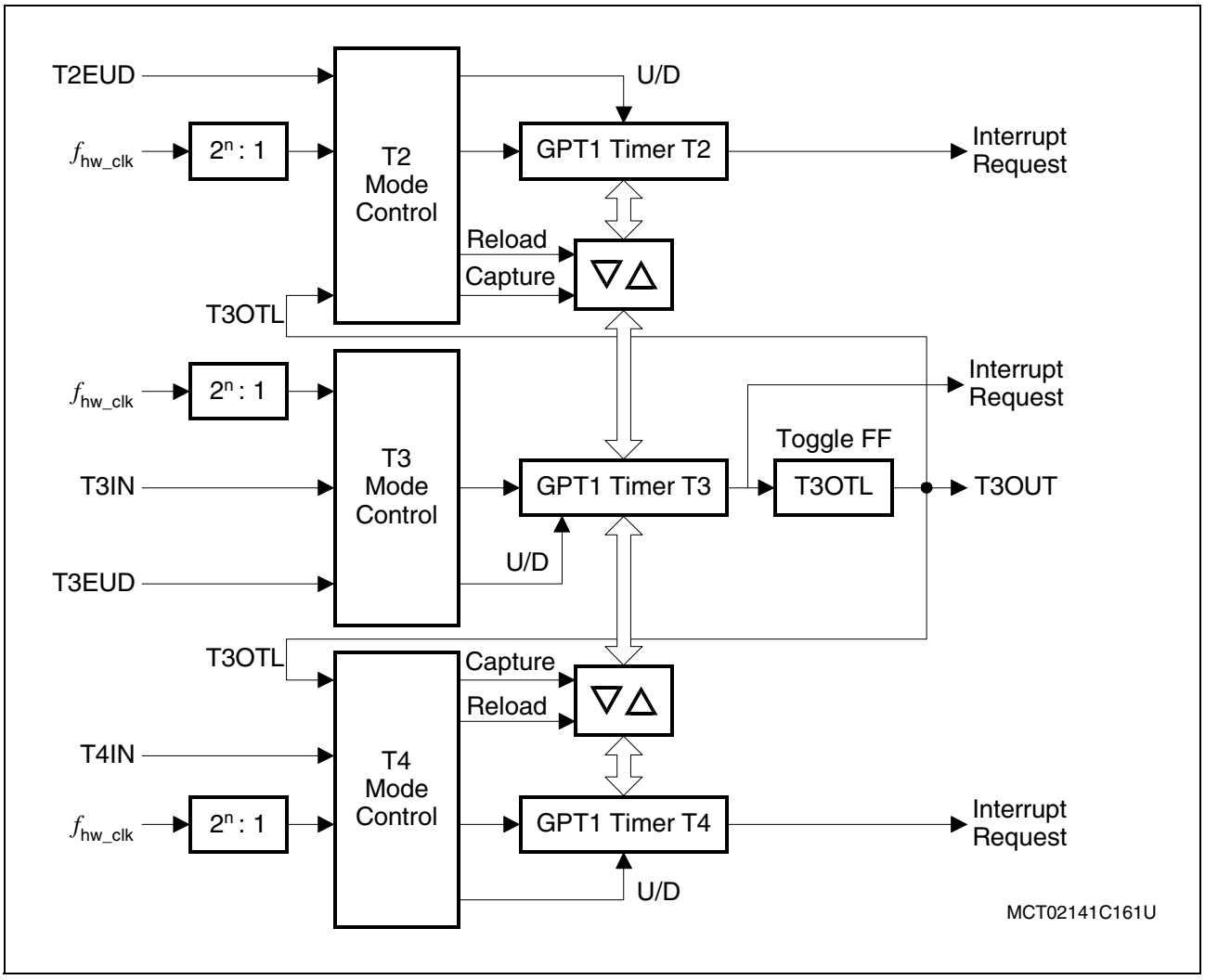


Figure 58 Structure of Timer Block 1

11.1.1.1 Core Timer T3

The operation of the core timer T3 is controlled by its bitaddressable control register T3CON.

Run Control

The timer can be started or stopped by software through bit T3R (Timer T3 Run Bit). Setting bit T3R to ‘1’ will start the timer, clearing T3R stops the timer.

In gated timer mode, the timer will only run if T3R = ‘1’ and the gate is active (high or low, as programmed).

Note: When bit T2RC/T4RC in timer control register T2CON/T4CON is set to ’1’, T3R will also control (start and stop) auxiliary timer T2/T4.

Count Direction Control

The count direction of the core timer can be controlled either by software or by the external input line T3EUD (Timer T3 External Up/Down Control Input). 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’, line T3EUD is selected to be the controlling source of the count direction. However, bit T3UD can still be used to reverse the actual count direction, as shown in the table below. If T3UD = ‘0’ and line T3EUD shows a low level, the timer is counting up. With a high level at T3EUD the timer is counting down. If T3UD = ‘1’, a high level at line 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 line T3EUD is used as external count direction control input, its associated port pin must be configured as input.

Table 35 GPT1 Core Timer T3 Count Direction Control

Line 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

Timer 2 and Timer 4 can be controlled by software only, since no second timer input exist, see also **Figure 57**.

Timer 3 Overflow/Underflow Monitoring

An overflow or underflow of timer T3 will clock the overflow toggle latch T3OTL in control register T3CON. T3OTL can also be set or reset by software. Bit T3OE (Overflow/ Underflow Output Enable) in register T3CON enables the state of T3OTL to be monitored via an external line T3OUT. If this line is linked to an external port pin, which has to be configured as output, T3OTL can be used to control external HW.

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 any port pin, 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 module clock f_{Timer} divided by a programmable prescaler, which is controlled by bit field T3I and bit FM1. The input frequency f_{T3} for timer T3 and its resolution r_{T3} are scaled linearly with lower module clock frequencies, as can be seen from the following formula:

$$f_{T3} = \frac{f_{Timer}}{8 * 2^{<T3I - FM1>}} \qquad r_{T3}\,[\mu s] = \frac{8 * 2^{<T3I - FM1>}}{f_{Timer}\,[MHz]}$$

Table 36 Example for Timer 3 Frequencies and Resolutions

f_{Timer} [MHz]	T3I	FM1	f_{T3} [KHz]	r_{T3} [µs]
24	'111'	0	23.44	42.67
24	'000'	1	6000.0	0.17
36	'000'	0	4500.0	0.22
36	'100'	0	281.25	3.55
36	'111'	1	70.31	14.22

This formula also applies to the Gated Timer Mode of T3 and to the auxiliary timers T2 and T4 in timer and gated timer mode, where applicable.

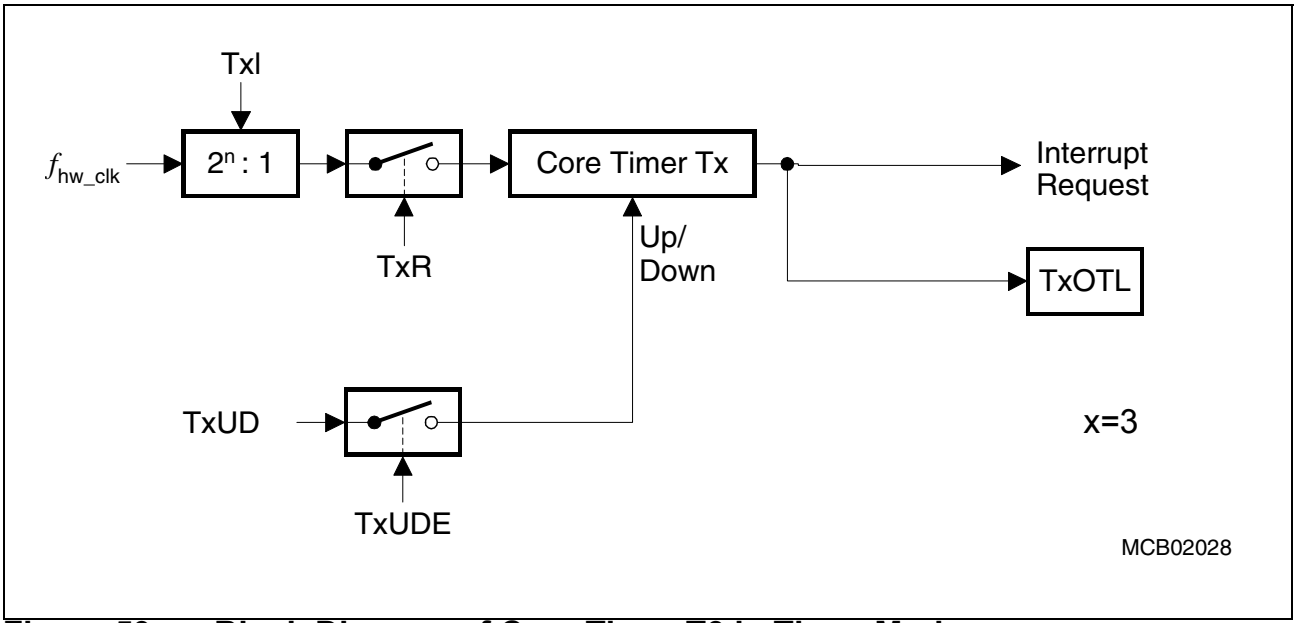


Figure 59 Block Diagram of Core Timer T3 in Timer Mode

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 line T3IN (Timer T3 External Input), which is an alternate function of P3.6.

To enable this operation pin P3.6/T3IN must be configured as input, ie. direction control bit DP3.6 must contain '0'.

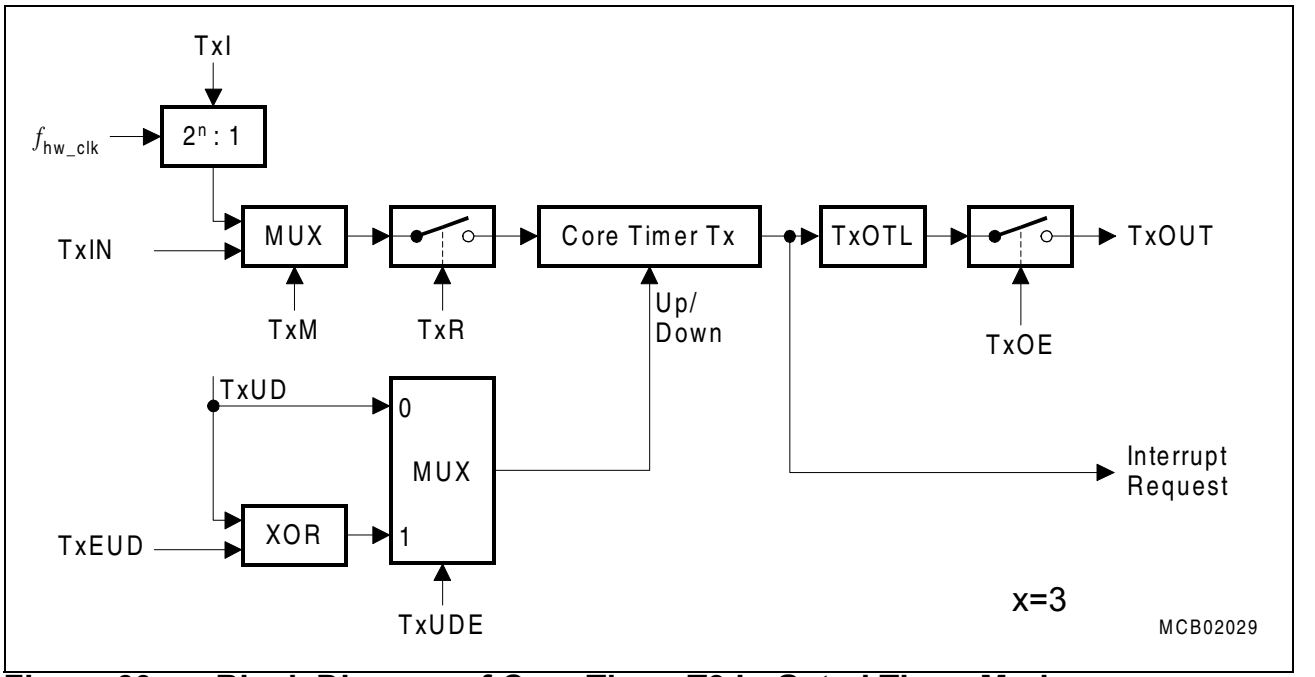


Figure 60 Block Diagram of Core Timer T3 in Gated Timer Mode

If T3M = '010_B', the timer is enabled when T3IN shows a low level. A high level at this line stops the timer. If T3M = '011_B', line T3IN must have a high level 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 line 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, which is an alternate function of P3.6. The event causing an increment or decrement of the timer can be a positive, a negative, or both a positive and a negative transition at this line. Bit field T3I in control register T3CON selects the triggering transition (see **Table 37** below).

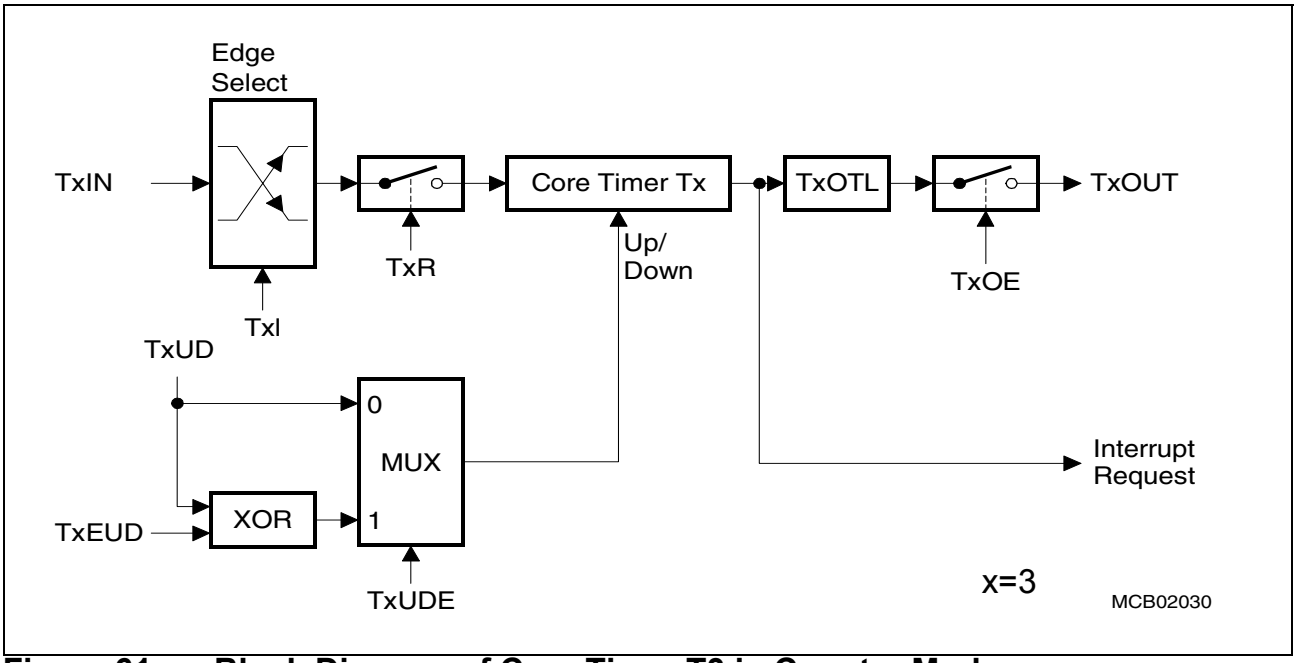


Figure 61 Block Diagram of Core Timer T3 in Counter Mode

Table 37 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, a port pin P3.6/T3IN must be configured as input. The maximum input frequency which is allowed in counter mode is $f_{Timer}/8$ (FM1 = ’1’). 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 4 f_{Timer} cycles (FM1 = ’1’) 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’ or ‘111_B’. In incremental interface mode pin P3.6/T3IN (configured as timer input T3IN) and pin P3.5/T3EUD (configured as timer input) are used to interface to an incremental encoder.

Note: In the C161U, the T3EUD timer input is connected to P3.5. In this case, Timer 4 input T4IN can be used by Software only.

T3 is clocked by each transition on one or both of the external input lines which gives 2-fold or 4-fold resolution of the encoder input.

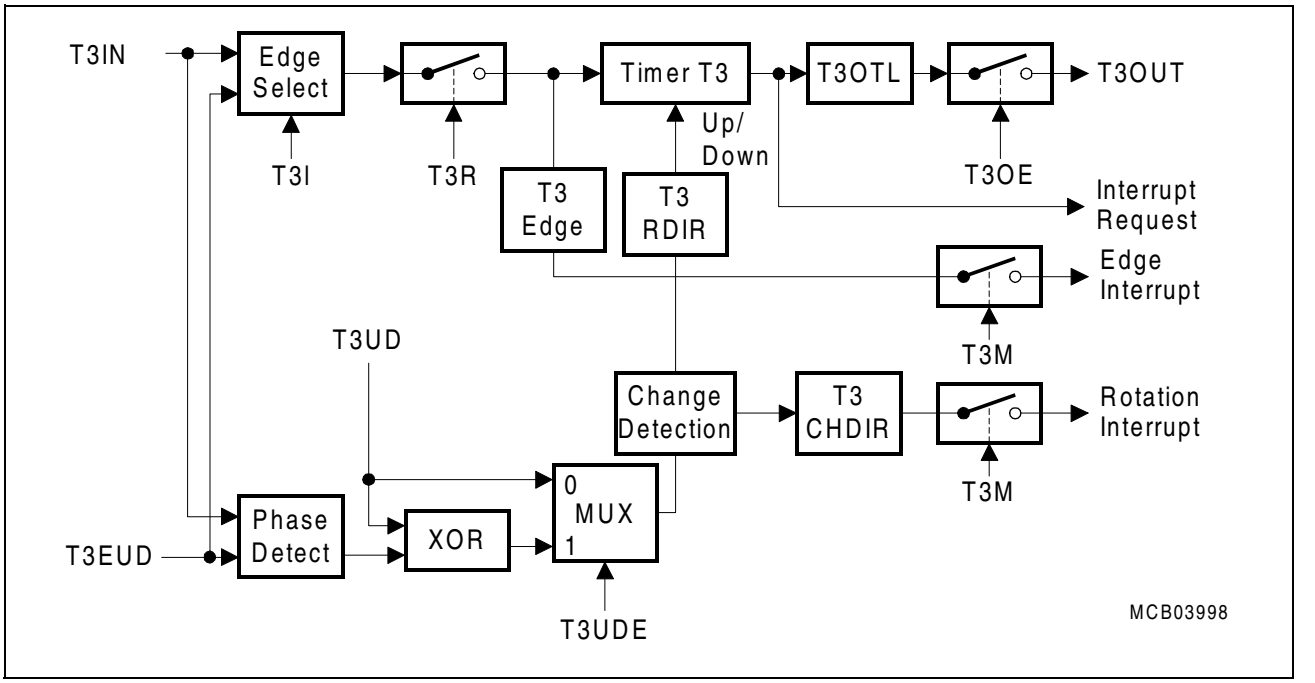


Figure 62 Block Diagram of Core Timer T3 in Incremental Interface Mode

Bit field 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. Depending on the chosen Incremental Intrerface Mode, Rotation detection '110_B' or Edge Detection '111_B', an interrupt can be generated. This interrupt is only generated if it's enabled by setting bit T3IREN in register T3CON. For the Rotation detection an interrupt will be generated each time the count direction of timer 3 changes. For the Edge detection an interrupt will be generated each time a count action for timer 3 occurs. Count direction, changes in the count direction and count requests are monitored through the status bits T3RDIR, T3CHDIR and T3EDGE in register T3CON. T3 is modified automatically according to the speed and the direction of the incremental encoder. Therefore, the contents of timer T3 always represents the encoder's current position.

Table 38 Core Timer T3 (Incremental Interface Mode) Input Edge Selection

T3I	Triggering Edge for Counter Increment / Decrement
0 0 0	None. Counter T3 stops.
0 0 1	Any transition (rising or falling edge) on T3IN.
0 1 0	Any transition (rising or falling edge) on T3EUD.
0 1 1	Any transition (rising or falling edge) on any T3 input (T3IN or T3EUD).
1 X X	Reserved. Do not use this combination

The incremental encoder can be connected directly to the microcontroller without external interface logic. In a standard system, however, comparators will be employed to convert the encoder's differential outputs (e.g. A, $\overline{A}$) to digital signals (e.g. A). This greatly increases noise immunity.

Note: The third encoder output T0, which indicates the mechanical zero position, may be connected to an external interrupt input and trigger a reset of timer T3.

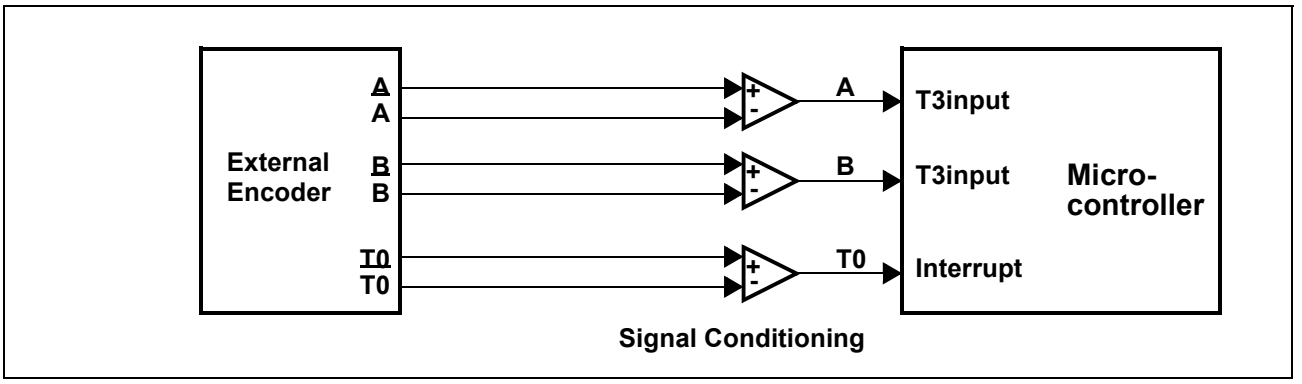


Figure 63 Interfacing the Encoder to the Microcontroller

For incremental interface operation the following conditions must be met:

Bitfield T3M must be '110_B' or '111_B'.

Pins associated to lines T3IN and T3EUD must be configured as input.

Bit T3UDE must be '1' to enable automatic direction control.

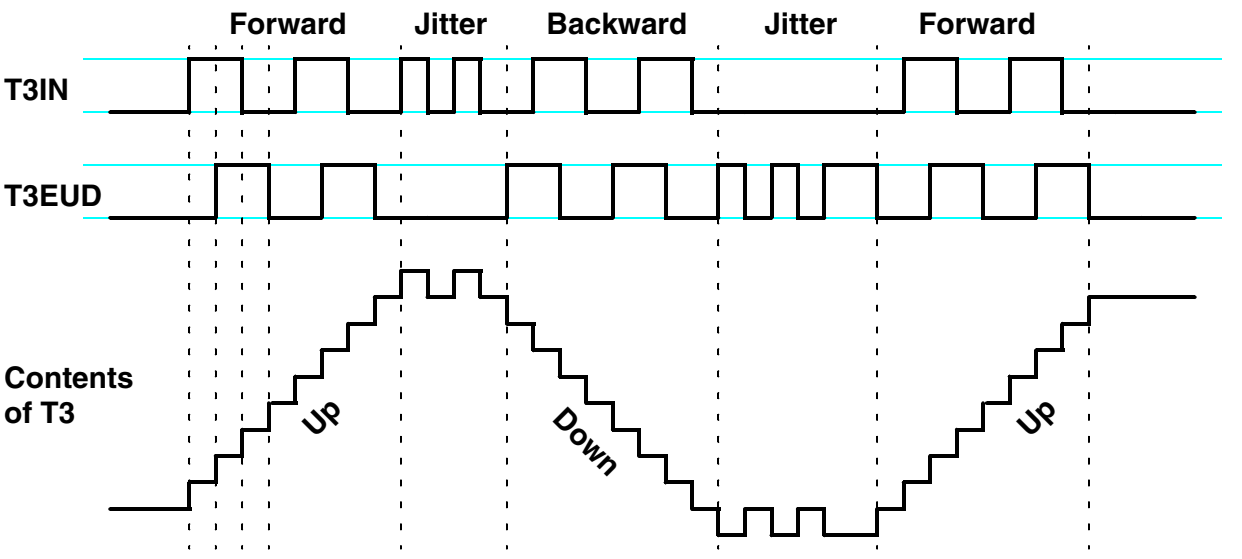
The maximum input frequency which is allowed in incremental interface mode is $f_{Timer}/8$ (FM = 1). To ensure that a transition of any input signal is correctly recognized, its level should be held high or low for at least 4 f_{Timer} cycles (FM = 1) before it changes.

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.

Table 39 Core Timer T3 (Incremental Interface Mode) Count Direction

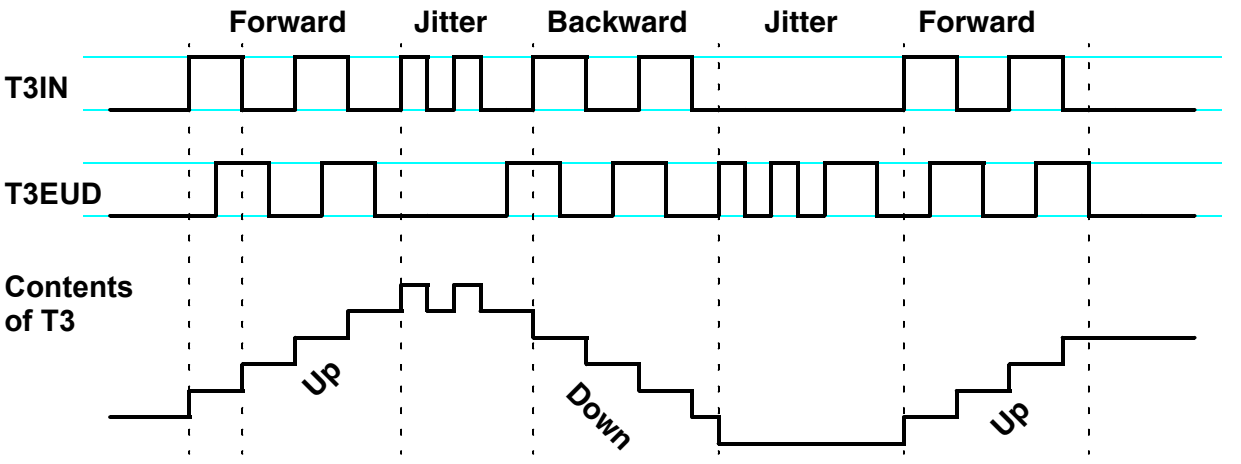
Level on respective other input	T3IN Input		T3EUD Input	
	Rising ↗	**Falling** ↘	**Rising** ↗	**Falling** ↘
High	Down	Up	Up	Down
Low	Up	Down	Down	Up

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 behavior assuming that T3 counts upon any transition on any input, i.e. T3I = '011_B'.

Figure 64 Evaluation of the Incremental Encoder Signals



Note: This example shows the timer behavior assuming that T3 counts upon any transition on input T3IN, i.e. T3I = '001_B'.

Figure 65 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.

11.1.1.2 Auxiliary Timers T2 and T4

Note: For the external pin connection of the timer T2 and timer T4, please refer to **Figure 57**, page 214.

Both auxiliary timers T2 and T4 have exactly the same functionality with the only restriction of the availibility of external pins connected to each timer. Timer T2 can be configured for timer, gated timer, counter, or incremental interface mode. Timer T4 can be configured for timer, gated timer and counter mode. In addition, 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.

Note: Timer 2 input T2IN is not connected to any external pin because of the limited number of pins of the C161U.

Note: Timer 2 input T2EUD is connected to P3.5. When the external input for T2EUD is used (P3.5 configured as input), T4IN and T3EUD can be used by Software (internal) only.

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 timer block 1 are controlled in the same bit positions and in the same manner in each of the specific control registers.

Run control for auxiliary timers T2 and T4 can be handled by the associated Run Control Bit T2R, T4R in register T2CON/T4CON. Alternatively, a remote control option (T2RC, T4RC = ’1’) may be enabled to start and stop T2/T4 via the run bit T3R of core timer T3.

Timers T2 and T4 in Timer Mode or Gated Timer Mode

When the auxiliary timers T2 and T4 are programmed to timer mode or gated timer mode, their operation is the same as described for the core timer T3. The descriptions, figures and tables apply accordingly with two exceptions:

There is no TxOUT output line for T2 and T4.

There is no T4EUD input, and no T2IN input. Therefore Software must be programmed accordingly.

Overflow/Underflow Monitoring is not supported (no output toggle latch).

Timer T4 in Counter Mode

In counter mode timer T4 can be clocked either by a transition at the respective external input line T4IN, or by a transition of timer T3’s output toggle latch T3OTL.

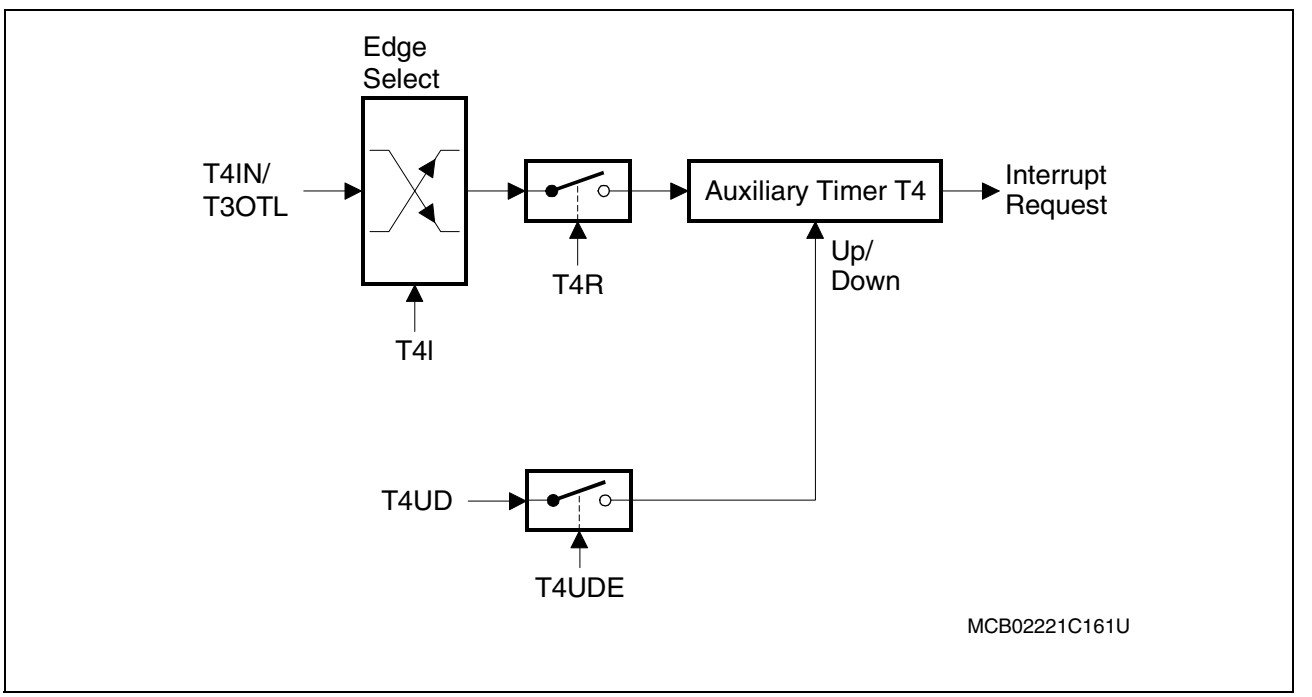


Figure 66 Block Diagram of the Auxiliary Timer T4 in Counter Mode

The event causing an increment or decrement of the timer T4 can be a positive, a negative, or both a positive and a negative transition at either the input line, or at the output toggle latch T3OTL.

Bit field T4I in the respective control register T4CON selects the triggering transition (see table below).

Table 40 Auxiliary Timer T4 (Counter Mode) Input Edge Selection

T4I	Triggering Edge for Counter Increment / Decrement
X 0 0	None. Counter T4 is disabled
0 0 1	Positive transition (rising edge) on T4IN
0 1 0	Negative transition (falling edge) on T4IN
0 1 1	Any transition (rising or falling edge) on T4IN
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 T4. Modifications of T3OTL via software will NOT trigger the counter function of T4.

For counter operation, an external pin associated to line T4IN must be configured as input. The maximum input frequency which is allowed in counter mode is $f_{Timer}/8$ (FM1 = ’1’). To ensure that a transition of the count input signal which is applied to T4IN is correctly recognized, its level should be held for at least 4 f_{Timer} cycles (FM1 = ’1’) before it changes.

Timer T2 in Counter Mode

The operation of Timer T2 in counter mode is the same as described above for Timer T4 in Chapter "Timer T4 in Counter Mode", with the following exeption:

- For Timer T2 there is no external input T2IN. Therefore, the counter function of T2 can only be triggered by state transitions of T3OTL, which are caused by the overflows/ underflows of T3. Modifications of T3OTL via software will NOT trigger the counter function of T2.

11.1.1.3 Timer Concatenation

Using the output toggle latch 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.

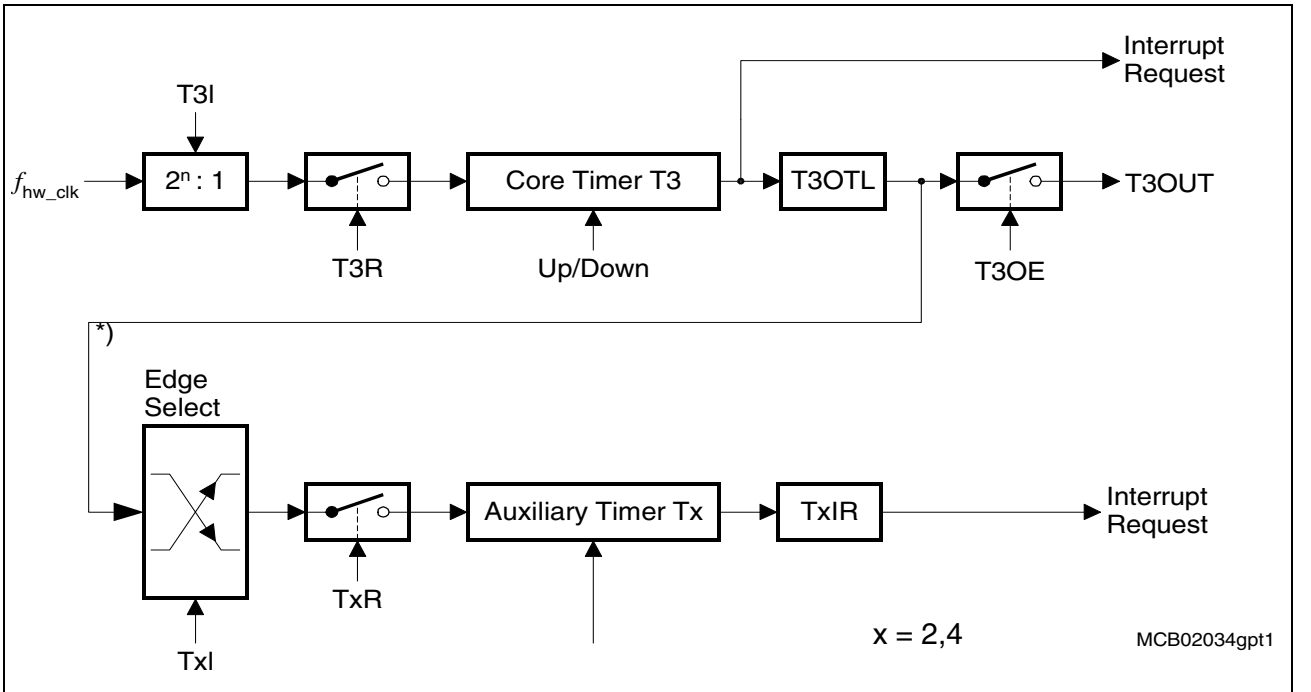
32-bit Timer/Counter: If both a positive and a negative transition of T3OTL is used to clock the auxiliary timer, this timer is clocked on every overflow/underflow of the core timer T3. Thus, the two timers form a 32-bit timer.

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.



Note: Line '*' only affected by over/underflows of T3, but NOT by software modifications of T3OTL.

Figure 67 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 for Timer T4 and by one signal (T3OFL/T3OTL) for Timer T2. 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: As opposed to Timer T4, for Timer T2 there is no input signal T2IN. Therefore, the reload mode for Timer T2 can only be triggered by Timer T3 signal T3OFL (which will set the toggle latch T3OTL as well).

Note: When programmed for reload mode, the respective auxiliary timer (T2 or T4) stops independent of its run flag T2R or T4R.

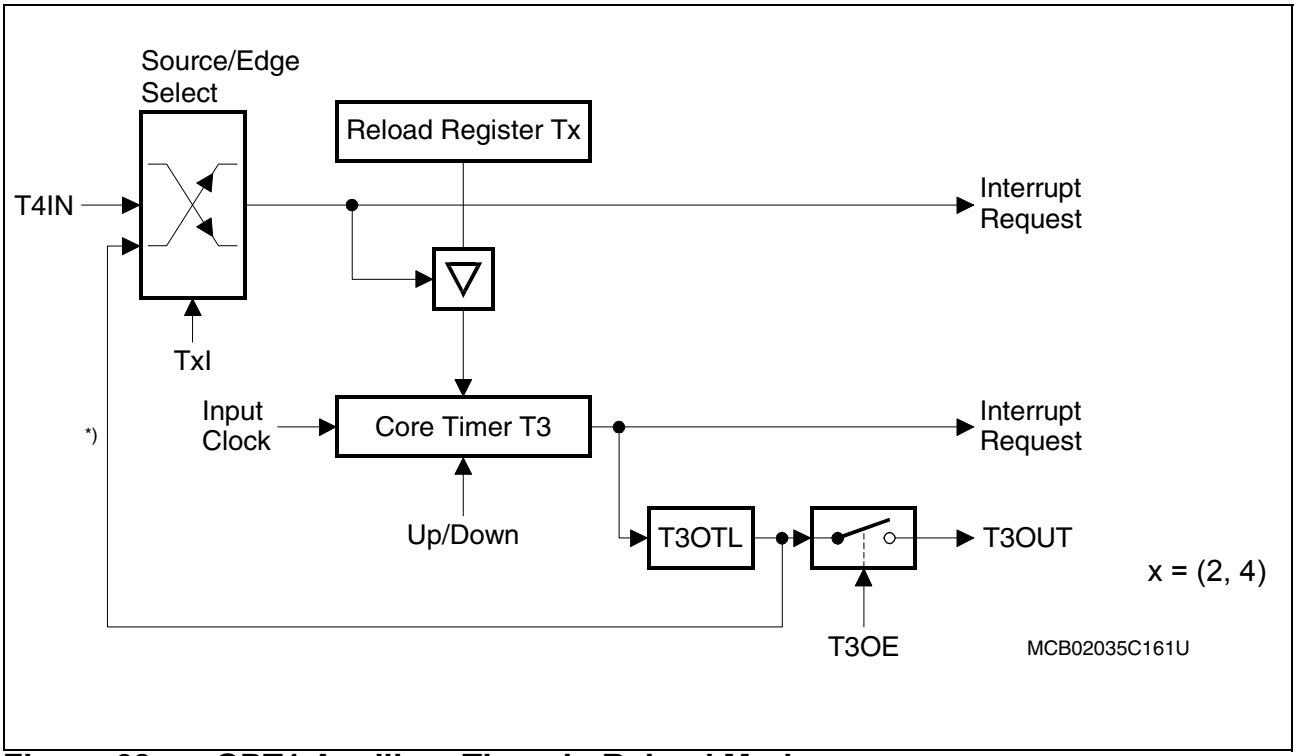


Figure 68 GPT1 Auxiliary Timer in Reload Mode

Note: Line '*' only affected by over/underflows of T3, but NOT by software modifications of T3OTL.

Upon a trigger signal T3 is loaded with the contents of the respective timer register (T2 or T4) and 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).

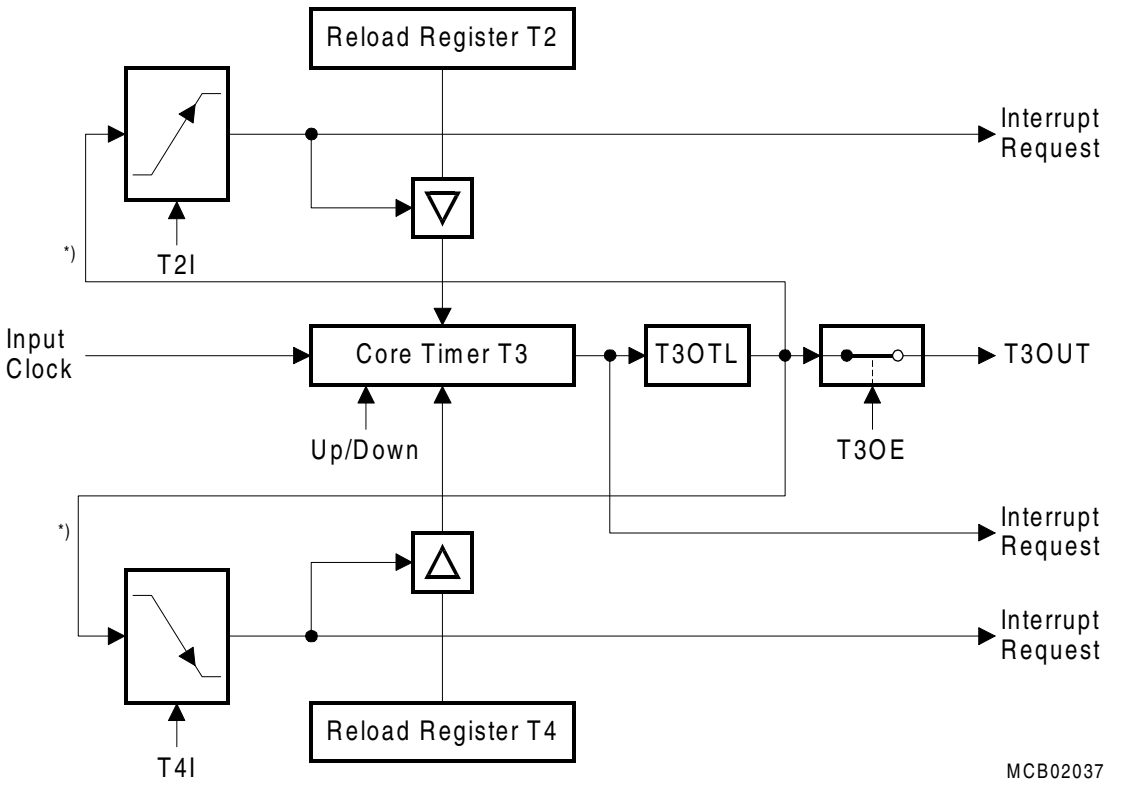
If either a positive or a negative transition of T3OTL is selected to trigger a reload, the core timer will be reloaded with the contents of the auxiliary timer on every second overflow or underflow.

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 line T3OUT if the control bit T3OE is set to ‘1’. With this method the high and low time of the PWM signal can be varied in a wide range.

Note:

1. 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.
2. An associated port pin linked to line T3OUT should be configured as output.



Note: Lines ’*’ only affected by over/underflows of T3, but NOT by software modifications of T3OTL.

Figure 69 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 T4 in Capture Mode

Capture mode for the auxiliary timer T4 is selected by setting bit field T4M in the register T4CON to '101_B'. In capture mode the contents of the core timer are latched into the auxiliary timer register in response to a signal transition at the auxiliary timer's external input line T4IN. 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 T4I are used to select the active transition (see table in the counter mode section), while the most significant bit T4I.2 is irrelevant for capture mode. It is recommended to keep this bit cleared (T4I.2 = '0').

Note: When programmed for capture mode, the auxiliary timer T4 stops independent of its run flag T4R.

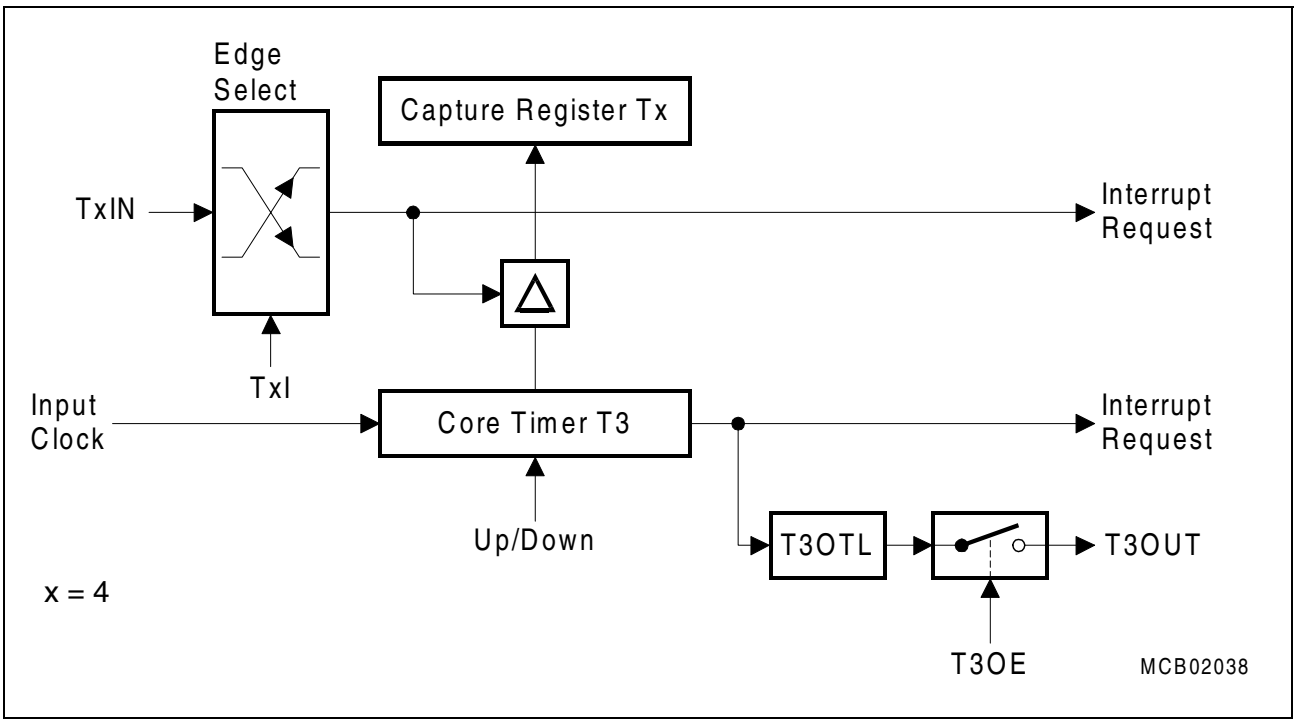


Figure 70 Auxiliary Timer T4 of Timer Block 1 in Capture Mode

Upon a trigger (selected transition) at the input line T4IN the contents of the core timer is loaded into the auxiliary timer register and the associated interrupt request flag T4IR will be set.

Note: The direction control for T4IN must be set to 'Input', and the level of the capture trigger signal should be held high or low for at least 4 f_{Timer} (FM1 = '1') cycles before it changes to ensure correct edge detection.

11.1.2 Functional Description of Timer Block 2

Timer block 2 includes the two timers T5 (referred to as the auxiliary timer) and T6 (referred to as the core timer), and the 16-bit capture/reload register CAPREL.

Note: The block 2 Timer T5 and Timer T6 can be used by Software only. There exist no external pin for timer T5 and T6. Additional, there is no external pin connected to register CAPREL, see also **Figure 57**, page 214.

The count direction (Up / Down) must be programmed via software. An overflow/ underflow of core timer T6 is indicated by the output toggle latch T6OTL whose state may be output on line T6OFL. The auxiliary timer T6 may be reloaded with the contents of CAPREL.

The toggle bit also supports the concatenation of T6 with auxiliary timer T5, while concatenation of T6 with other timers is provided through line T6OFL. Triggered by an external signal, T3IN or T3EUD, the contents of timer 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.

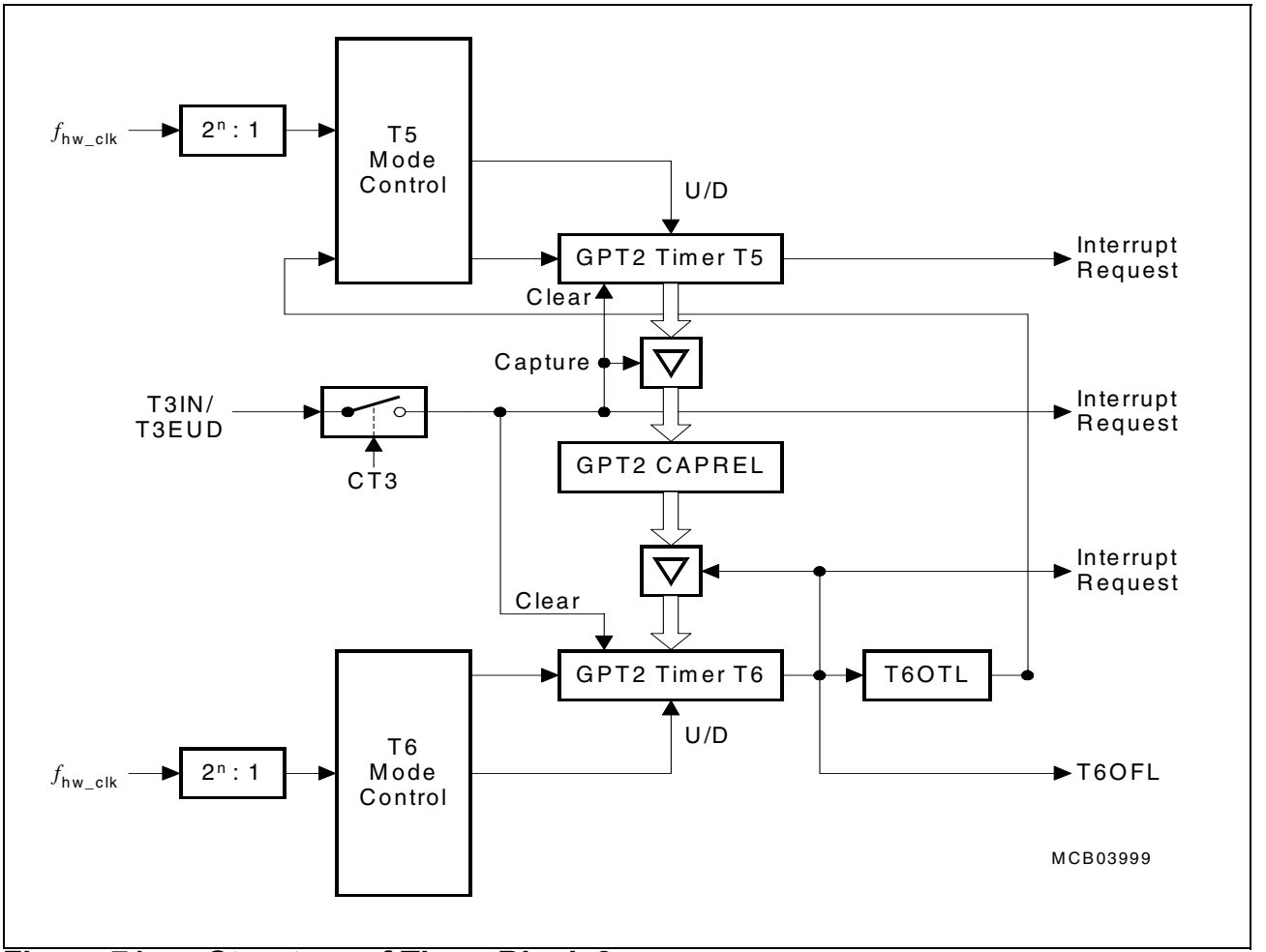


Figure 71 Structure of Timer Block 2

11.1.2.1 Core Timer T6

The operation of the core timer T6 is controlled by its bitaddressable control register T6CON.

Timer 6 Run Bit

The timer can be started or stopped by software through bit T6R (Timer T6 Run Bit). Setting bit T6R to ‘1’ will start the timer, clearing T6R stops the timer.

Count Direction Control

The count direction of the core timer can be controlled by software only. The count direction can be altered by setting or clearing bit T6UD.

Note: Bit T6UDE of register T6CON must be always set to ’0’.

The count direction can be changed regardless of whether the timer is running or not.

Table 41 Core Timer T6 Count Direction Control

Bit TxUDE	Bit TxUD	Count Direction
0	0	Count Up
0	1	Count Down

Note: The direction control works the same for core timer T6 and for auxiliary timer T5. Therefore the lines and bits are named Tx...

Timer 6 Overflow/Underflow Monitoring

An overflow or underflow of timer T6 will clock the toggle latch T6OTL in control register T6CON. T6OTL can also be set or reset by software.

In addition, T6OTL can be used in conjunction with the timer over/underflows as an input for the counter function of the auxiliary timer T5. For this purpose, an internal connection is provided for this option.

An overflow or underflow of timer T6 can also be used to clock other timers. For this purpose, there is the special output line T6OFL.

Timer 6 in Timer Mode

Timer mode for the core timer T6 is selected by setting bit field T6M in register T6CON to '000_B'. In this mode, T6 is clocked with the module 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_{Timer}, as can be seen from the following formula:

$$f_{T6} = \frac{f_{Timer}}{4 * 2^{<T6I - FM2>}} \qquad r_{T6}\,[\mu s] = \frac{4 * 2^{<T6I - FM2>}}{f_{Timer}\,[MHz]}$$

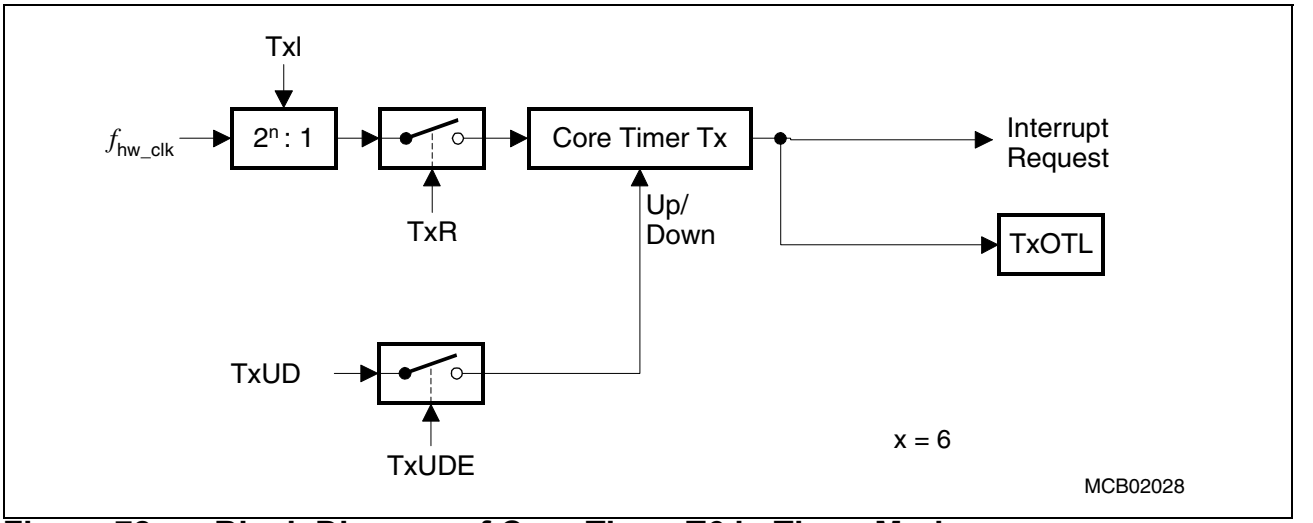


Figure 72 Block Diagram of Core Timer T6 in Timer Mode

11.1.2.2 Auxiliary Timer T5

The auxiliary timer T5 can be configured for timer mode with the same options for the timer frequencies and the count signal as the core timer T6. In addition, the auxiliary timer can be concatenated with the core timer.

The individual configuration for timer T5 is determined by its bitaddressable control register T5CON. Note that functions which are present in both timers of timer block 2 are controlled in the same bit positions and in the same manner in each of the specific control registers.

Run control for auxiliary timer T5 can be handled by the associated Run Control Bit T5R in register T5CON. Alternatively, a remote control option (T5RC = ’1’) may be enabled to start and stop T5 via the run bit T6R of core timer T6.

Note: The auxiliary timer has no overflow/underflow toggle latch. Therefore, an output line for Overflow/Underflow Monitoring is not provided.

Count Direction Control for Auxiliary Timer

The count direction of the auxiliary timer can be controlled in the same way as for the core timer T6. The description and the table apply accordingly.

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 signal T6OFL only.

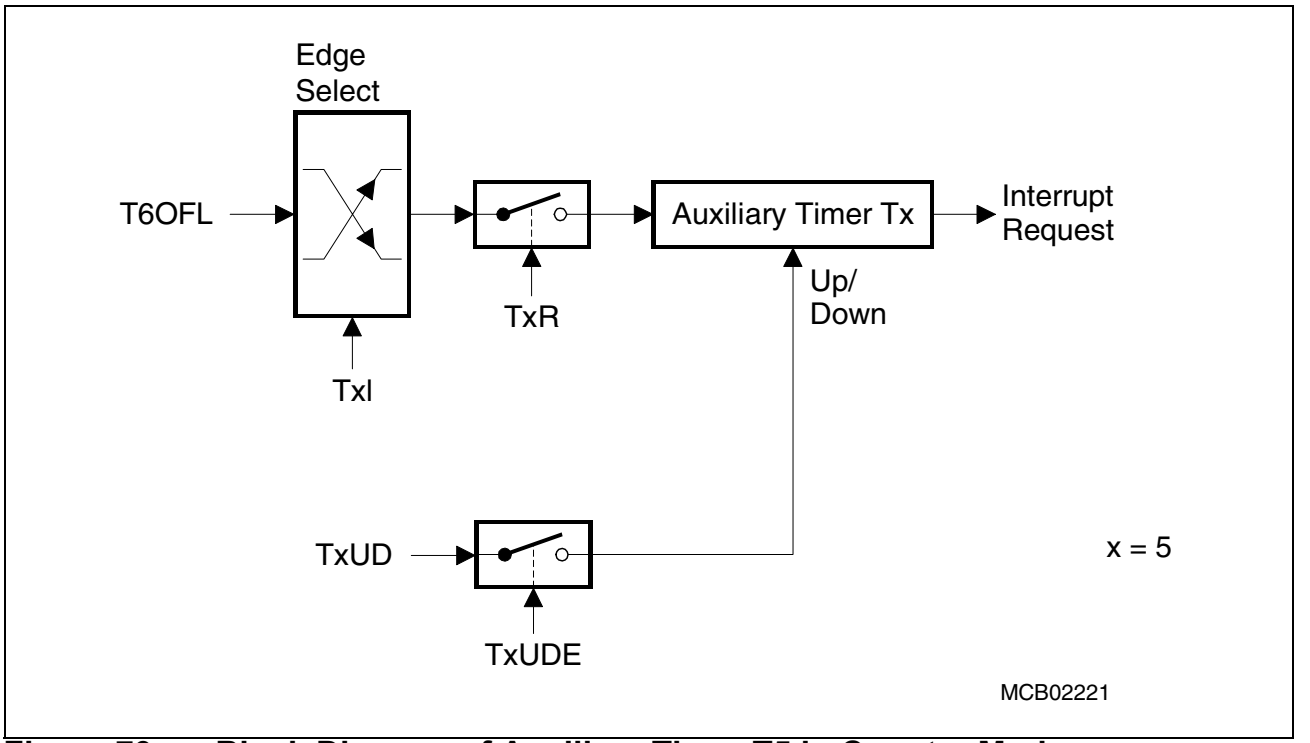


Figure 73 Block Diagram of Auxiliary Timer T5 in Counter Mode

The event causing an increment or decrement of the timer can be a positive, a negative, or both a positive and a negative transition at signal T6OFL (toggle latch T6OTL).

Bit field T5P in control register T5CON selects the triggering transition (see table below).

Table 42 Auxiliary Timer (Counter Mode) Input Edge Selection

T5P	Triggering Edge for Counter Increment / Decrement
X 0 0	None. Counter T5 is disabled
0 0 1	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.

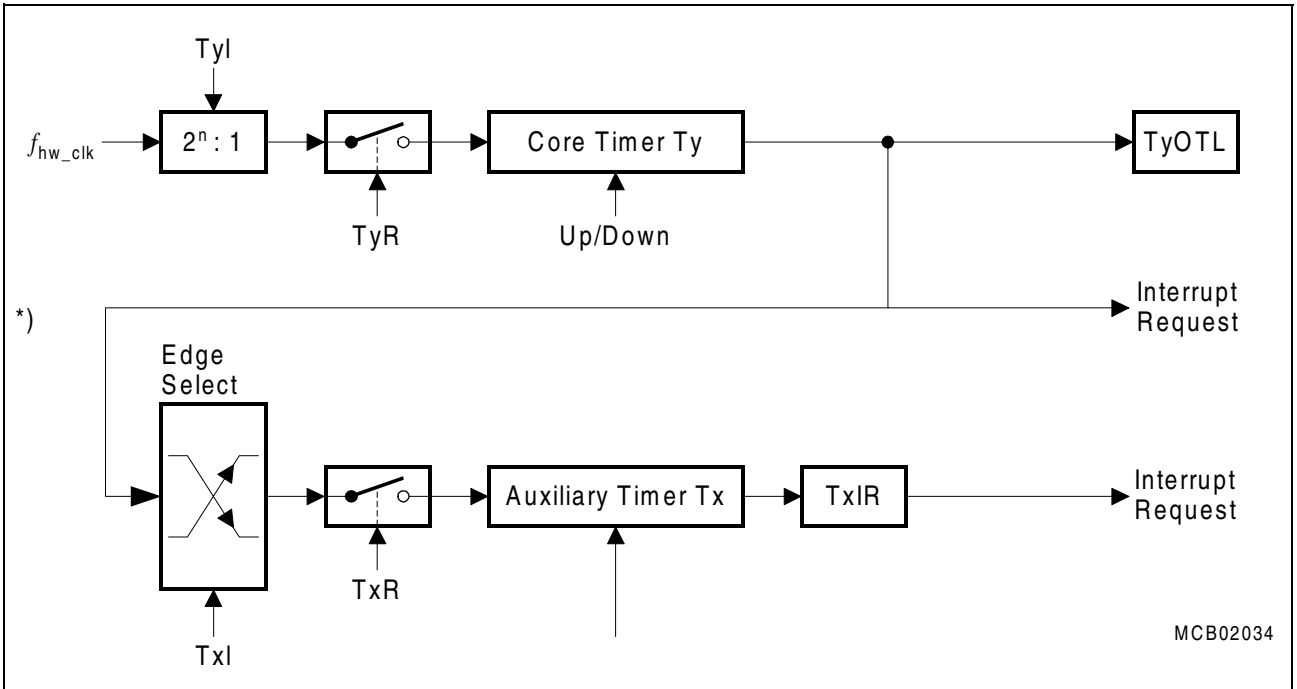
11.1.2.3 Timer Concatenation

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 74 Concatenation of Core Timer T6 and Auxiliary Timer T5

Timer Block 2 Capture/Reload Register CAPREL in Reload Mode

The 16-bit capture/reload register CAPREL 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.

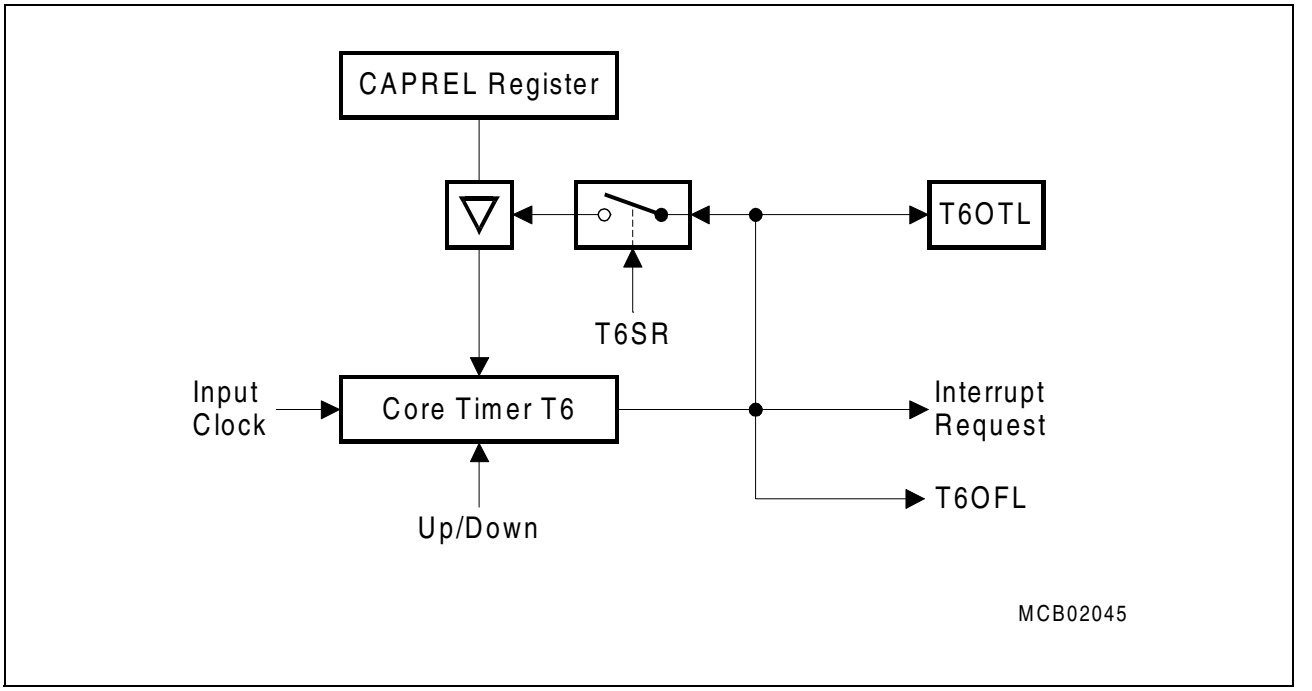


Figure 75 Timer Block 2 Register CAPREL in Reload Mode

11.1.3 GPT Register Set

All GPT12 related registers are summarized in the overview table below.

Table 43 GPT Register Overview

Name	Description	Address	Reset Value
GPTCLC	GPT Clock Control Register	$FE4C_H$	0000_H
T2CON	Timer 2 Function Control Register	$FF40_H$	0000_H
T3CON	Timer 3 Function Control Register	$FF42_H$	0000_H
T4CON	Timer 4 Function Control Register	$FF44_H$	0000_H
T5CON	Timer 5 Function Control Register	$FF46_H$	0000_H
T6CON	Timer 6 Function Control Register	$FF48_H$	0000_H
T2	GPT1 Timer 2 Register	$FE40_H$	0000_H
T3	GPT1 Timer 3 Register	$FE42_H$	0000_H
T4	GPT1 Timer 4 Register	$FE44_H$	0000_H
T5	GPT2 Timer 5 Register	$FE46_H$	0000_H
T6	GPT2 Timer 6 Register	$FE48_H$	0000_H
CAPREL	Capture Reload Register	$FE4A_H$	0000_H
T2IC[1)]	GPT1 Timer 2 Interrupt Control Register	$FF60_H$	0000_H
T3IC[1)]	GPT1 Timer 3 Interrupt Control Register	$FF62_H$	0000_H
T4IC[1)]	GPT1 Timer 4 Interrupt Control Register	$FF64_H$	0000_H
T5IC[1)]	GPT2 Timer 5 Interrupt Control Register	$FF66_H$	0000_H
T6IC[1)]	GPT2 Timer 6 Interrupt Control Register	$FF68_H$	0000_H
CRIC[1)]	GPT2 CAPREL Interrupt Control Register	$FF6A_H$	0000_H

1) For the Interrupt Control Register description, please refer to Chapter 7.2, page 111.

GPT Clock Control Register

GPTCLC ($FE4C_H$) **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	EXDISR	SUSPEN	GPTDISS	GPTDISR
												rw	rw	r	rw

Bit	Function
GPTDISR	GPT Disable Request Bit GPTDISR = ‘0’: GPT clock disable not requested GPTDISR = ‘1’: GPT clock disable requested
GPTDISS	GPT Disable Status Bit GPTDISS = ‘0’: GPT clock enabled GPTDISS = ‘1’: GPT clock disabled
SUSPEN	Peripheral Suspend Enable Bit for OCDS SUSPEN = ‘0’: Peripheral suspend disabled SUSPEN = ‘1’: Peripheral suspend enabled
EXDISR	External Disable Request EXRDIS = ‘0’: External clock disable Request is enabled EXRDIS = ‘1’: External clock disable Request is disabled

Function Control Registers

The operating mode of the core timer T3 is configured and controlled via its bitaddressable control register T3CON.

T3CON
Timer 3 Control Register **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T3IREN	T3RDIR	T3CHDIR	T3EDGE	FM1	T3OTL	T3OE	T3UDE	T3UD	T3R	T3M			T3I		

Field	Bits	Type	Value	Description
T3I	[2:0]	rw		Timer 3 Input Parameter Selection Timer mode see **Table 44** for encoding Gated Timer see **Table 44** for encoding Counter mode see **Table 45** for encoding Incremental Interface mode see **Table 46** for encoding

Field	Bits	Type	Value	Description
T3M	[5:3]	rw		Timer 3 Mode Control
			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 (Rotation detection)
			1 1 1	Incremental Interface Mode (Edge detection)
T3R	6	rw		Timer 3 Run Bit
			0	Timer / Counter 3 stops
			1	Timer / Counter 3 runs
T3UD	7	rw		**Timer 3 Up / Down Control**
				(when T3UDE = ’0)
			0	Counting ’Up’
			1	Counting ’Down’
T3UDE	8	rw		**Timer 3 External Up/Down Enable**
			0	Counting direction is internally controlled by SW
			1	Counting direction is externally controlled by line T3EUD
T3OE	9	rw		Overflow/Underflow Output Enable
			0	T3 overflow/underflow can not be externally monitored
			1	T3 overflow/underflow may be externally monitored via T3OUT
T3OTL	10	rw		Timer 3 Output Toggle Latch
			0 / 1	Toggles on each overflow / underflow of T3. Can be set or reset by software.
FM1	11	rw		Fast Mode for Timer Block 1
			0	The maximum input frequency for Timer 2/3/4 is f_{Timer} / 8.
			1	The maximum input frequency for Timer 2/3/4 is f_{Timer} / 4.
T3EDGE	12	rw		Timer 3 Edge Detection
				The bit is set on each successful edge detection.
				The bit has to be reset by SW.
			0	No count edge was detected
			1	A count edge was detected

Field	Bits	Type	Value	Description
T3CHDIR	13	rw		Timer 3 Count Direction Change The bit is set on a change of the countdirection of timer 3. The bit has to be reset by SW.
			0	No change in count direction was detected
			1	A change in count direction was detected
T3RDIR	14	r		Timer 3 Rotation Direction
			0	Timer 3 counts up.
			1	Timer 3 counts down.
T3IREN	15	rw		Timer 3 Interrupt Enable
			0	Interrupt generation for T3CHDIR and T3EDGE is disabled.
			1	Interrupt generation for T3CHDIR and T3EDGE is enabled.

Table 44 Timer 3 Input Parameter Selection for Timer mode and Gated mode

T3I	Prescaler for f_{Timer} (FM1 = 0)	Prescaler for f_{Timer} (FM1 = 1)
000	8	4
001	16	8
010	32	16
011	64	32
100	128	64
101	256	128
110	512	256
111	1014	512

Table 45 Timer 3 Input Parameter Selection for Counter mode

T3I	Triggering Edge for Counter Update
000	None. Counter T3 is disabled
001	Positive transition (raising edge) on T3IN
010	Negative transition (falling edge) on T3IN
011	Any transition (raising or falling edge) on T3IN
1XX	***Reserved***. Do not use this combination!

Table 46 Timer 3 Input Parameter Selection for Incremental Interface mode

T3I	Triggering Edge for Counter Update
000	None. Counter T3 stops
001	Any transition (raising or falling edge) on T3IN
010	Any transition (raising or falling edge) on T3EUD
011	Any transition (raising or falling edge) on T3IN or T3EUD
1XX	***Reserved***. Do not use this combination!

T2CON
Timer 2 Control Register **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T2 IREN	T2 RDIR	T2CH DIR	T2 EDG E	'0'		T2RC	T2 UDE	T2UD	T2R	T2M			T2I		

Field	Bits	Type	Value	Description
T2I	[2:0]	rw		Timer 2 Input Parameter Selection Timer mode see **Table 47** for encoding Gated Timer see **Table 47** for encoding Counter mode see **Table 48** for encoding Incremental Interface mode see **Table 49** for encoding
T2M	[5:3]	rw	 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1 1 1 0 1 1 1	**Timer 2 Mode Control** (Basic Operating Mode) Timer Mode Counter Mode Gated Timer with Gate active low Gated Timer with Gate active high Reload Mode Capture Mode Incremental Interface Mode (Rotation detection) Incermental Interface Mode (Edge detection)
T2R	6	rw	 0 1	Timer 2 Run Bit Timer / Counter 2 stops Timer / Counter 2 runs

Field	Bits	Type	Value	Description
T2UD	7	rw		Timer 2 Up / Down Control (when T2UDE = ’0)
			0	Counting ’Up’
			1	Counting ’Down’
T2UDE	8	rw		Timer 2 External Up/Down Enable
			0	Counting direction is internally controlled by SW
			1	Counting direction is externally controlled by line T2EUD Pin P3.5 connected to T2EUD is also connected to T4IN and to T3EUD.
T2RC	9	rw		Timer 2 Remote Control
			0	Timer / Counter 2 is controlled by its own run bit T2R
			1	Timer / Counter 2 is controlled by the run bit of core timer 3
0	[11:10]	r		reserved for future use; reading returns 0; writing to these bit positions has no effect.
T2EDGE	12	rw		Timer 2 Edge Detection The bit is set on each successful edge detection. The bit has to be reset by SW.
			0	No count edge was detected
			1	A count edge was detected
T2CHDIR	13	rw		Timer 2 Count Direction Change The bit is set on a change of the countdirection of timer 2. The bit has to be reset by SW.
			0	No change in count direction was detected
			1	A change in count direction was detected
T2RDIR	14	r		Timer 2 Rotation Direction
			0	Timer 2 counts up.
			1	Timer 2 counts down.
T2IREN	15	rw		Timer 2 Interrupt Enable
			0	Interrupt generation for T2CHDIR and T2EDGE is disabled.
			1	Interrupt generation for T2CHDIR and T2EDGE is enabled.

Table 47 Timer 2 Input Parameter Selection for Timer mode and Gated mode

T2I	Prescaler for f_{Timer} (FM1 = 0)	Prescaler for f_{Timer} (FM1 = 1)
000	8	4
001	16	8
010	32	16
011	64	32
100	128	64
101	256	128
110	512	256
111	1014	512

Table 48 Timer 2 Input Parameter Selection for Counter mode

T2I	Triggering Edge for Counter Update
X 0 0	None. Counter T2 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 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

Table 49 Timer 2 Input Parameter Selection for Incremental Interface mode

T2I	Triggering Edge for Counter Update
000	None. Counter T2 stops
001	***Reserved***. Do not use this combination!
010	Any transition (raising or falling edge) on T2EUD
011	***Reserved***. Do not use this combination!
1XX	***Reserved***. Do not use this combination!

T4CON
Timer 4 Control Register **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T4 IREN	T4 RDIR	T4CH DIR	T4 EDG E	'0'		T4RC	'0' (T4 UDE)	T4UD	T4R	T4M			T4I		

Field	Bits	Type	Value	Description
T4I	[2:0]	rw		Timer 4 Input Parameter Selection Timer mode see **Table 50** for encoding Gated Timer see **Table 50** for encoding Counter mode see **Table 51** for encoding
T4M	[5:3]	rw	 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1 1 1 0 1 1 1	**Timer 4 Mode Control** (Basic Operating Mode) Timer Mode Counter Mode Gated Timer with Gate active low Gated Timer with Gate active high Reload Mode Capture Mode Reserved. Do not use this combination. Reserved. Do not use this combination.
T4R	6	rw	 0 1	Timer 4 Run Bit Timer / Counter 4 stops Timer / Counter 4 runs
T4UD	7	rw	 0 1	**Timer 4 Up / Down Control** Counting 'Up' Counting 'Down'
'0' (T4UDE-bit)	8	rw	 0	Timer 4 External Up/Down Enable This bit must be set to '0' signal.
T4RC	9	rw	 0 1	Timer 4 Remote Control Timer / Counter 4 is controlled by its own run bit T4R Timer / Counter 4 is controlled by the run bit of core timer 3
'0'	[11:10]	r		reserved for future use; reading returns 0; writing to these bit positions has no effect.

Field	Bits	Type	Value	Description
T4EDGE	12	rw		Timer 4 Edge Detection The bit is set on each successful edge detection. The bit has to be reset by SW.
			0	No count edge was detected
			1	A count edge was detected
T4CHDIR	13	rw		Timer 4 Count Direction Change The bit is set on a change of the countdirection of timer 4. The bit has to be reset by SW.
			0	No change in count direction was detected
			1	A change in count direction was detected
T4RDIR	14	r		Timer 4 Rotation Direction
			0	Timer 4 counts up.
			1	Timer 4 counts down.
T4IREN	15	rw		Timer 4 Interrupt Enable
			0	Interrupt generation for T4CHDIR and T4EDGE is disabled.
			1	Interrupt generation for T4CHDIR and T4EDGE is enabled.

Table 50 Timer 4 Input Parameter Selection for Timer mode and Gated mode

T4I	Prescaler for f_{Timer} (FM1 = 0)	Prescaler for f_{Timer} (FM1 = 1)
000	8	4
001	16	8
010	32	16
011	64	32
100	128	64
101	256	128
110	512	256
111	1014	512

Table 51 Timer 4 Input Parameter Selection for Counter mode

T4I	Triggering Edge for Counter Update
X 0 0	None. Counter T4 is disabled
0 0 1	Positive transition (rising edge) on T4IN
0 1 0	Negative transition (falling edge) on T4IN
0 1 1	Any transition (rising or falling edge) on T4IN
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

T6CON
Timer 6 Control Register **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T6SR	T6 CLR	'0'		FM2	T6 OTL	'0' (T6 OE)	'0' (T6 UDE)	T6UD	T6R	T6M			T6I		

Field	Bits	Type	Value	Description
T6I	[2:0]	rw		Timer 6 Input Parameter Selection Timer mode see **Table 52** for encoding
T6M	[5:3]	rw	 0 0 0 0 0 1 0 1 0 0 1 1 1 x x	Timer 6 Mode Control (Basic Operating Mode) Timer Mode ***Reserved***. Do not use this combination! ***Reserved***. Do not use this combination! ***Reserved***. Do not use this combination! ***Reserved***. Do not use this combination!
T6R	6	rw	 0 1	Timer 6 Run Bit Timer / Counter 6 stops Timer / Counter 6 runs
T6UD	7	rw	 0 1	**Timer 6 Up / Down Control** Counting 'Up' Counting 'Down'
'0' (T6UDE)	8	rw	 0	Timer 6 External Up/Down Enable This bit must be set to '0' signal

Field	Bits	Type	Value	Description
’0’ (T6OE)	9	rw	0	Overflow/Underflow Output Enable This bit must be set to ’0’ signal
T6OTL	10	rw	0 / 1	Timer 6 Output Toggle Latch Toggles on each overflow / underflow of T6. Can be set or reset by software.
FM2	11	rw	0 1	Fast Mode for Timer Block 2 The maximum input frequency for Timer 5/6 is f_{Timer} / 4. The maximum input frequency for Timer 5/6 is f_{Timer} / 2.
’0’	[13:12]	r		reserved for future use; reading returns 0; writing to these bit positions has no effect.
T6CLR	14	rw	0 1	Timer 6 Clear Bit Timer 6 is not cleared on a capture event Timer 6 is cleared on a capture event
T6SR	15	rw	0 1	Timer 6 Reload Mode Enable Reload from register CAPREL Disabled Reload from register CAPREL Enabled

Table 52 Timer 6 Input Parameter Selection for Timer mode and Gated mode

T6I	Prescaler for f_{Timer} (FM2 = 0)	Prescaler for f_{Timer} (FM2 = 1)
000	4	2
001	8	4
010	16	8
011	32	16
100	64	32
101	128	64
110	256	128
111	512	256

T5CON
Timer 5 Control Register **Reset Value: 0000_H**

15	14	13 12	11	10	9	8	7	6	5	4 3	2 1 0
T5SC	T5 CLR	CI	CC	'1' (CT3)	T5RC	'0' (T5 UDE)	T5UD	T5R	'0'	T5M	T5I

Field	Bits	Type	Value	Description
T5I	[2:0]	rw		Timer 5 Input Parameter Selection Timer mode see **Table 53** for encoding Counter mode see **Table 54** for encoding
T5M	[4:3]	rw	0 0 0 1 1 0 1 1	Timer 5 Mode Control (Basic Operating Mode) Timer Mode Counter Mode Reserved. Do not use this configuration Reserved. Do not use this configuration
'0'	5	r		reserved for future use; reading returns 0; writing to these bit positions has no effect.
T5R	6	rw	0 1	Timer 5 Run Bit Timer / Counter 5 stops Timer / Counter 5 runs
T5UD	7	rw	0 1	**Timer 5 Up / Down Control** Counting 'Up' Counting 'Down'
'0' (T5UDE)	8	rw	0	Timer 5 External Up/Down Enable This bit must be set to '0' signal
T5RC	9	rw	0 1	Timer 5 Remote Control Timer / Counter x is controlled by its own run bit T5R Timer / Counter 5 is controlled by the run bit of core timer 6 (T6R)
'1' (CT3)	10	rw	0	Timer 3 Capture Trigger Enable This bit must be set to '1' signal
CC	11	rw	0 1	Capture Correction T5 is just captured T5 is decremented by 1 before being captured

Field	Bits	Type	Value	Description
CI	[13:12]	rw		**Register CAPREL Capture Trigger Selection**
			0 0	Capture disabled
			0 1	Any transition on T3IN
			1 0	Any transition on T3EUD
			1 1	Any transition on T3IN or T3EUD
T5CLR	14	rw		Timer 5 Clear Bit
			0	Timer 5 not cleared on a capture
			1	Timer 5 is cleared on a capture
T5SC	15	rw		Timer 5 Capture Mode Enable
			0	Capture into register CAPREL Disabled
			1	Capture into register CAPREL Enabled

Table 53 Timer 5 Input Parameter Selection for Timer mode

T5I	Prescaler for f_{Timer} (FM2 = 0)	Prescaler for f_{Timer} (FM2 = 1)
000	4	2
001	8	4
010	16	8
011	32	16
100	64	32
101	128	64
110	256	128
111	512	256

Table 54 Timer 5 Input Parameter Selection for Counter mode

T5I	Triggering Edge for Counter Update
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

Table 54 Timer 5 Input Parameter Selection for Counter mode

T5I	Triggering Edge for Counter Update
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

T2/T3/T4/T5/T6
Timer Tx Register **Reset Value: 0000_H**

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

Field	Bits	Type	Value	Description
value	[15:0]	rw		Timer Tx Register 16 bit register contains the actual timer value of the respective Timer Tx.

CAPREL
GPT2 CAPREL Register **Reset Value: 0000_H**

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

Field	Bits	Type	Value	Description
value	[15:0]	rw		GPT2 CAPREL Register 16 bit register contains the actual value of the CAPREL register.

12 Asynchronous/Synchr. Serial Interface

Asynchronous/Synchronous Serial Interface (ASC) provides serial communication between the C161U and other microcontrollers, microprocessors or external peripherals. The ASC supports a certain protocol to transfer data via a serial interconnection.

12.1 Functional Description

ASC supports full-duplex asynchronous communication up to 2.25 MBaud and half-duplex synchronous communication up to 4.5 MBaud (@ 36 MHz CPU clock which is equal to the ASC module clock). In synchronous mode, data are transmitted or received synchronous to a shift clock which is generated by the microcontroller. In asynchronous mode, 8- or 9-bit data transfer, parity generation, and the number of stop bits can be selected.

12.1.1 Features

- Full duplex asynchronous operating modes
  - 8- or 9-bit data frames, LSB first
  - Parity bit generation/checking
  - One or two stop bits
  - Baudrate from 2.25 MBaud to 0.5364 Baud (@ 36 MHz module clock = CPU clock)
  - Multiprocessor mode for automatic address/data byte detection
  - Loop-back capability
  - Support for IrDA data transmission/reception up to max. 115.2 kBaud

- Autobaud detection unit for asynchronous operating modes
  - Detection of standard baudrates
    1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400 Baud
  - Detection of non-standard baudrates
  - Detection of asynchronous modes
    7-bit, even parity; 7-bit, odd parity;
    8-bit, even parity; 8-bit, odd parity; 8-bit, no parity
  - Automatic initialization of control bits and baudrate generator after detection
  - Detection of a serial two-byte ASCII character frame

- Recently introduced fractional divider
  - The fractional divider drastically improves the accuracy of the adjustment for baudrates
  - Standard Baud Rates generation with very small deviation (230.4 kBaud < 0.01%, 460.8 kBaud < 0.15 %, 691.2 kBaud < 0.04 %, 921.6 kBaud < 0.15 %) @ 36 MHz

- Half-duplex 8-bit synchronous operating mode
  – Baudrate from 4.5 MBaud to 366.21 Baud (@ 36 MHz module clock = CPU clock)

- Double buffered transmitter/receiver

- Interrupt generation
  – on a transmitter buffer empty condition
  – on a transmit last bit of a frame condition
  – on a receiver buffer full condition
  – on an error condition (frame, parity, overrun error)
  – on the start and the end of a autobaud detection

12.1.2 Overview

Figure 76 shows a block diagram of the ASC with its operating modes (asynchronous and synchronous mode.).

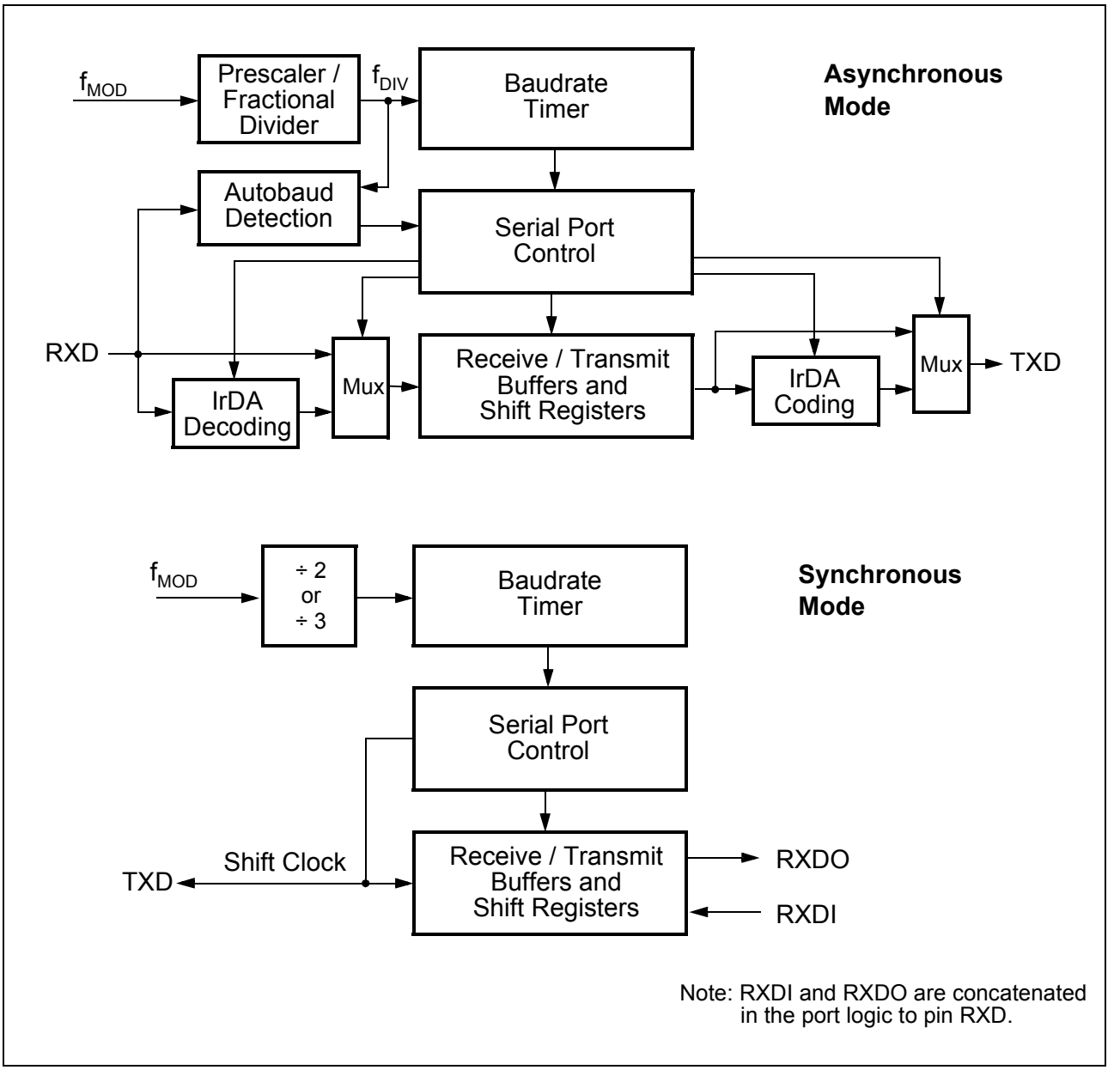


Figure 76 Block Diagram of the ASC

12.1.3 Register Description

ASC registers can be basically divided into four types of registers as shown in **Figure 77**.

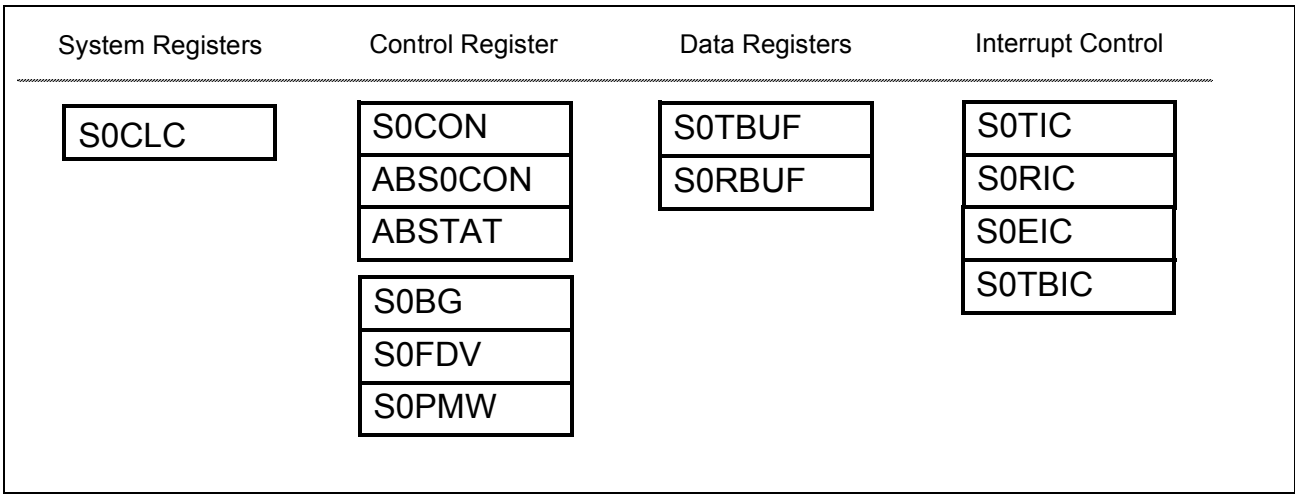


Figure 77 SFRs associated with ASC

S0CLC	Clock Control Register
S0ID	Identification Register
S0CON	Control Register
ABS0CON	Autobaud Control Register
ABSTAT	Autobaud Status Register
S0TIC	ASC Transmit Interrupt Control Register
S0RIC	ASC Receive Interrupt Control Registe
S0BG	Baudrate Timer Reload Register
S0FDV	Fractional Divider Register
S0PMW	IrDA Pulse Mode and Width Register
S0TBUF	Transmit Buffer Register
S0RBUF	Receive Buffer Register (read only)
S0EIC	ASC Error Interrupt Control Register
S0TBIC	ASC Transmit Buffer Interrupt Control Register

Table 55 ASC Register Summary

Name	Address	Reset Value	Type	Description
S0CLC	$FFBA_H$	0000_H	rw / r	ASC Clock Control Register
S0CON	$FFB0_H$	0000_H	rwh	Control Register
ABS0CON	$FEF8_H$	0000_H	rwh	Autobaud Control Register
ABSTAT	$FEFE_H$	0000_H	rwh	Autobaud Status Register
S0BG	$FEB4_H$	0000_H	rw	Baudrate Timer Reload Register
S0FDV	$FEB6_H$	0000_H	rw	Fractional Divider Register
S0PMW	$FEAA_H$	0000_H	rw	IrDA Pulse Mode and Width Register
S0TBUF	$FEB0_H$	0000_H	rw	Transmit Buffer Register
S0RBUF	$FEB2_H$	0000_H	r	Receive Buffer Register
S0TIC	$FF6C_H$	0000_H	rw	Serial Channel 0 Transmit Interrupt Control Register
S0RIC	$FF6E_H$	0000_H	rw	Serial Channel 0 Receive Interrupt Control Register
S0EIC	$FF70_H$	0000_H	rw	Serial Channel 0 Error Interrupt Control Register
S0TBIC	$F19C_H$	0000_H	rw	Serial Channel 0 Transmit Buffer IC Register

ASC Clock Control Register

S0CLC ($FFBA_H$) **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	EX DISR	SUS PEN	S0 DISS	S0 DISR
												rw	rw	r	rw

Bit	Function
S0DISR	ASC Disable Request Bit S0DISR = ‘0’: ASC clock disable not requested S0DISR = ‘1’: ASC clock disable requested
S0DISS	ASC Disable Status Bit S0DISS = ‘0’: ASC clock enabled S0DISS = ‘1’: ASC clock disabled
SUSPEN	Peripheral Suspend Enable Bit for OCDS SUSPEN = ‘0’: Peripheral suspend disabled SUSPEN = ‘1’: Peripheral suspend enabled
EXDISR	External Disable Request EXRDIS = ‘0’: External clock disable Request is enabled EXRDIS = ‘1’: External clock disable Request is disabled

The serial operating modes of the ASC module are controlled by its control register S0CON. This register contains control bits for mode and error check selection, and status flags for error identification.

S0CON
Control Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	LB	BRS	ODD	FDE	OE	FE	PE	OEN	FEN	PEN/ RXDI	REN	STP	M		

Note: Serial data transmission or reception is only possible when the run bit S0CON.R is set to ‘1’. Otherwise the serial interface is idle.
Do not program the mode control field S0CON.M to one of the reserved combinations to avoid unpredictable behaviour of the serial interface.

Field	Bits	Type	Value	Description
M	2-0	rwh		Mode Selection
			0 0 0	8-bit data synchronous operation
			0 0 1	8-bit data async. operation
			0 1 0	IrDA mode, 8-bit data async. operation
			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 bitasync. operation
			1 1 0	Reserved. Do not use this combination!
			1 1 1	8-bit data + parity async. operation
				Bits are set/cleared by hardware after a successfull autobaud detection operation. In synchronous operation (M=’000’), the Fractional Divider is always disabled.
STP	3	rw		Number of Stop Bit Selection
			0	One stop bit
			1	Two stop bits
REN	4	rwh		Receiver Enable Control
			0	Receiver diabled
			1	Receiver enabled
				Bit can be affected during autobaud detection operation when bit ABEN_AUREN is set. Bit is reset by hardware after reception of byte in synchronous mode.
PEN RXDI	5	rw		Parity Check Enable / IrDA Input Inverter Enable
				All asynchronous modes without IrDA mode:
			0	Ignore parity
			1	Check parity
				Only in IrDA mode (M=010):
			0	RXD input is not inverted
			1	RXD input is inverted
FEN	6	rw		**Framing Check Enable** (async. modes only)
			0	Ignore framing errors
			1	Check framing errors
OEN	7	rw		Overrun Check Enable
			0	Ignore overrun errors
			1	Check overrun errors

Field	Bits	Type	Value	Description
PE	8	rwh		Parity Error Flag Set by hardware on a parity error (PEN='1'). Must be reset by software.
FE	9	rwh		Framing Error Flag Set by hardware on a framing error (FEN='1'). Must be reset by software.
OE	10	rwh		Overrun Error Flag Set by hardware on an overrun error (OEN='1'). Must be reset by software.
FDE	11	rw	0 1	Fractional Divider Enable Fractional divider disabled Fractional divider is enabled and used as prescaler for baudrate timer (bit BRS is don't care)
ODD	12	rwh	0 1	Parity Selection Even parity selected (parity bit set on odd number of '1's in data) Odd parity selected (parity bit set on even number of '1's in data) Bit is be set/cleared by hardware after a successfull autobaud detection operation.
BRS	13	rw	0 1	Baudrate Selection Baudrate timer prescaler divide-by-2 selected Baudrate timer prescaler divide-by-3 selected BRS is don't care if FDE=1 (fractional divider enabled)
LB	14	rw	0 1	Loopback Mode Enable Loopback mode disabled Loopback mode enabled
R	15	rw	0 1	Baudrate Generator Run Control Baudrate generator disabled (**ASC_P** inactive) Baudrate generator enabled BG should only be written if R='0'.

The autobaud control register ABS0CON of the ASC module is used to control the autobaud detection operation. It contains its general enable bit, the interrupt enable control bits, and data path control bits.

ABS0CON
Autobaud Control Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**RX INV**	**TX INV**	**ABEM**		**0**	**0**	**0**	**FC DET EN**	**AB DET EN**	**ABST EN**	**AUR EN**	**AB EN**

Field	Bits	Type	Value	Description
ABEN	0	rwh	 0 1	Autobaud Detection Enable Autobaud detection is disabled Autobaud detection is enabled ABEN is reset by hardware after a successful autobaud detection; (with the stop bit detection of the second character). Resetting ABEN by software if it was set aborts the autobaud detection.
AUREN	1	rw	 0 1	Automatic Autobaud Control of CON_REN CON_REN is not affected during autobaud detection CON_REN is cleared (receiver disabled) when ABEN and AUREN are set together. CON_REN is set (receiver enabled) after a successful autobaud detection (with the stop bit detection of the second character).
ABSTEN	2	rw	 0 1	Start of Autobaud Detection Interrupt Enable Start of autobaud detection interrupt disabled Start of autobaud detection interrupt enabled
ABDETEN	3	rw	 0 1	Autobaud Detection Interrupt Enable Autobaud detection interrupt disabled Autobaud detection interrupt enabled

Field	Bits	Type	Value	Description
FCDETEN	4	rw		First Character of Two-Byte Frame Detected Enable
			0	Autobaud detection interrupt ABDETIR becomes active after the two-byte frame recognition
			1	Autobaud detection interrupt ABDETIR becomes active after detection of the first and second byte of the two-byte frame.
ABEM	8-9	rw		Autobaud Echo Mode Enable In echo mode the serial data at RXD is switched to TXD output.
			0 0	Echo mode disabled
			0 1	Echo mode is enabled during autobaud detection
			1 0	Echo mode is always enabled
			1 1	reserved;
TXINV	10	rw		Transmit Inverter Enable
			0	Transmit inverter disabled
			1	Transmit inverter enabled
RXINV	11	rw		Receive Inverter Enable
			0	Receive inverter disabled
			1	Receive inverter enabled
–	7-5, 15-12	0	all	reserved

The autobaud status register ABSTAT of the ASC module indicates the status of the autobaud detection operation.

ABSTAT
Autobaud Status Register

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**	**DET WAIT**	**SCC DET**	**SCS DET**	**FCC DET**	**FCS DET**

Note: SCSDET or SCCDET are set when the second character has been recognized. CON_ABEN is reset and ABDETIR set after SCSDET or SCCDET have seen set.

Field	Bits	Type	Value	Description
FCSDET	0	rwh	0 1	First Character with Small Letter Detected no small ’a’ character detected small ’a’ character detected Bit is cleared by hardware when ABCON_ABEN is set or if FCCDET or SCSDET or SCCDET is set. Bit can be also cleared by software.
FCCDET	1	rwh	0 1	First Character with Capital Letter Detected no capital ’A’ character detected capital ’A’ character detected Bit is cleared by hardware when ABCON_ABEN is set or if FCSDET or SCSDET or SCCDET is set. Bit can be also cleared by software.
SCSDET	2	rwh	0 1	Second Character with Small Letter Detected no small ’t’ character detected small ’t’ character detected Bit is cleared by hardware when ABCON_ABEN is set or if FCSDET or FCCDET or SCCDET is set. Bit can be also cleared by software.
SCCDET	3	rwh	0 1	Second Character with Capital Letter Detected no capital ’T’ character detected capital ’T’ character detected Bit is cleared by hardware when ABCON_ABEN is set or if FCSDET or FCCDET or SCSDET is set. Bit can be also cleared by software.
DETWAIT	4	rwh	0 1	Autobaud Detection is Waiting Either character ’a’, ’A’, ’t’, or ’T’ has been detected. The autobaud detection unit waits for the first ’a’ or ’A’ Bit is cleared when either FCSDET or FCCDET is set (’a’ or ’A’ detected). Bit can be also cleared by software. DETWAIT is set by hardware when ABCON_ABEN is set.
–	15-5	0	all	reserved

The baudrate timer reload register S0BG of the ASC module contains the 13-bit reload value for the baudrate timer in asynchronous and sychronous mode.

S0BG
Baudrate Timer/Reload Register

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

Field	Bits	Type	Value	Description
BR_VALUE	12-0	rw	all	Baudrate Timer/Reload Register Value Reading BG returns the 13-bit content of the baudrate timer (bits 15....13 return 0); writing BG loads the baudrate timer reload register (bits 15....13 are don't care). BG should only be written if CON_R='0'.
–	15-13	0	all	reserved

The fractional divider register S0FDV of the ASC module contains the 9-bit divider value for the fractional divider (asynchronous mode only). It is also used for reference clock generation of the autobaud detection unit.

S0FDV
Fractional Divider Register

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

Field	Bits	Type	Value	Description
FD_VALUE	8-0	rw	all	Fractional Divider Register Value FDV contains the 9-bit value n of the fractional divider which defines the fractional divider ratio: n/512 n=0-511). With n=0, the fractional divider is switched off (input=output frequency, $f_{DIV} = f_{MOD}$, see **Figure 86**).
–	15-9	0	all	reserved

The IrDA pulse mode and width register S0PMW of the ASC module contains the 8-bit IrDA pulse width value and the IrDA pulse width mode select bit. This register is only required in the IrDA operating mode.

S0PMW
IrDA Pulse Mode/Width Register

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

Field	Bits	Type	Value	Description
PW_VALUE	7-0	rw	all	IrDA Pulse Width Value PW_VALUE is the 8-bit value n, which defines the variable pulse width of an IrDA pulse. Depending on the **ASC_P** input frequency f_{MOD}, this value can be used to adjust the IrDA pulse width to value which is not equal 3/16 bit time (e.g. 1.6 µs).
IRPW	8	rw	 0 1	IrDA Pulse Width Mode Control IrDA pulse width is 3/16 of the bit time IrDA pulse width is defined by PW_VALUE
–	15-9	0	all	reserved

The transmitter buffer register S0TBUF of the ASC module contains the transmit data value in asynchronous and synchronous modes.

S0TBUF
Transmitter Buffer Register

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

Field	Bits	Type	Value	Description
TD_VALUE	8-0	rw	all	Transmit Data Register Value TBUF contains the data to be transmitted in asynchronous and synchronous operating mode of the ASC. Data transmission is double buffered, Therefore, a new value can be written to TBUF before the transmission of the previous value is complete.
–	15-9	0	all	reserved

The receiver buffer register S0RBUF of the ASC module contains the receive data value in asynchronous and synchronous modes.

S0RBUF
Receive Buffer Register

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

Field	Bits	Type	Value	Description
RD_VALUE	8-0	rw	all	Receive Data Register Value S0RBUF contains the reveived data bits and, depending on the selected mode, the parity bit in asynchronous and synchronous operating mode of the ASC. In asynchronous operating mode with M=011 (7-bit data + parity) the received parity bit is written into RD7. In asynchronous operating mode with M=111 (8-bit data + parity) the received parity bit is written into RD8.
–	15-9	0	all	reserved

S0TIC SFR **Reset Value: - - 00_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								S0 TIR	S0 TIE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

S0RIC SFR **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								S0 RIR	S0 RIE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

S0EIC SFR **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								S0 EIR	S0 EIE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

S0TBIC SFR **Reset Value: - - 00$_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								S0 EIR	S0 EIE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

Note: Please refer to the general Interrupt Control Register description on page 111 for an explanation of the control fields.

12.1.4 General Operation

ASC supports full-duplex asynchronous communication up to 2.25 MBaud and half-duplex synchronous communication up to 4.5 MBaud (@ 36 MHz CPU clock which is equal to the ASC module clock). In synchronous mode, data are transmitted or received synchronous to a shift clock which is generated by the microcontroller. 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 baudrate timer with a versatile input clock divider circuitry provides the ASC with the serial clock signal. In a special asynchronous mode, the ASC supports IrDA data transmission up to 115.2 kBaud with fixed or programmable IrDA pulse width. A autobaud detection unit allows to detect asynchronous data frames with its baudrate and mode with automatic initialization of the baudrate generator and the mode controll bits.

A transmission is started by writing to the Transmit Buffer register S0TBUF. Only the number of data bits which is determined by the selected operating mode will actually be transmitted, ie. bits written to positions 9 through 15 of register S0TBUF are always insignificant.

Data transmission is double-buffered, so a new character may be written to the transmit buffer register, before the transmission of the previous character is complete. This allows the transmission of characters back-to-back without gaps.

Data reception is enabled by the Receiver Enable Bit CON_REN. After reception of a character has been completed, the received data 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 CON_OEN. When enabled, the overrun error status flag CON_OE and the error interrupt request line EIR will be acitvated 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 CON_LB) allows the data currently being transmitted to be received simultaneously in the receive buffer. This may be used to test serial communication routines at an early stage without having to provide an external network. In loop-back mode the alternate input/output function of port pins is not required.

Note: Serial data transmission or reception is only possible when the Baudrate Generator Run Bit CON_R is set to ‘1’. Otherwise the serial interface is idle.
Do not program the mode control field COM_M to one of the reserved combinations to avoid unpredictable behaviour of the serial interface

12.1.5 Asynchronous Operation

Asynchronous mode supports full-duplex communication, where both transmitter and receiver use the same data frame format and the same baudrate. Data is transmitted on pin P3.10/TXD and received on pin P3.11/RXD. IrDA data transmission/reception is supported up to 115.2 KBit/s. **Figure 78** shows the block diagram of the ASC when operating in asynchronous mode.

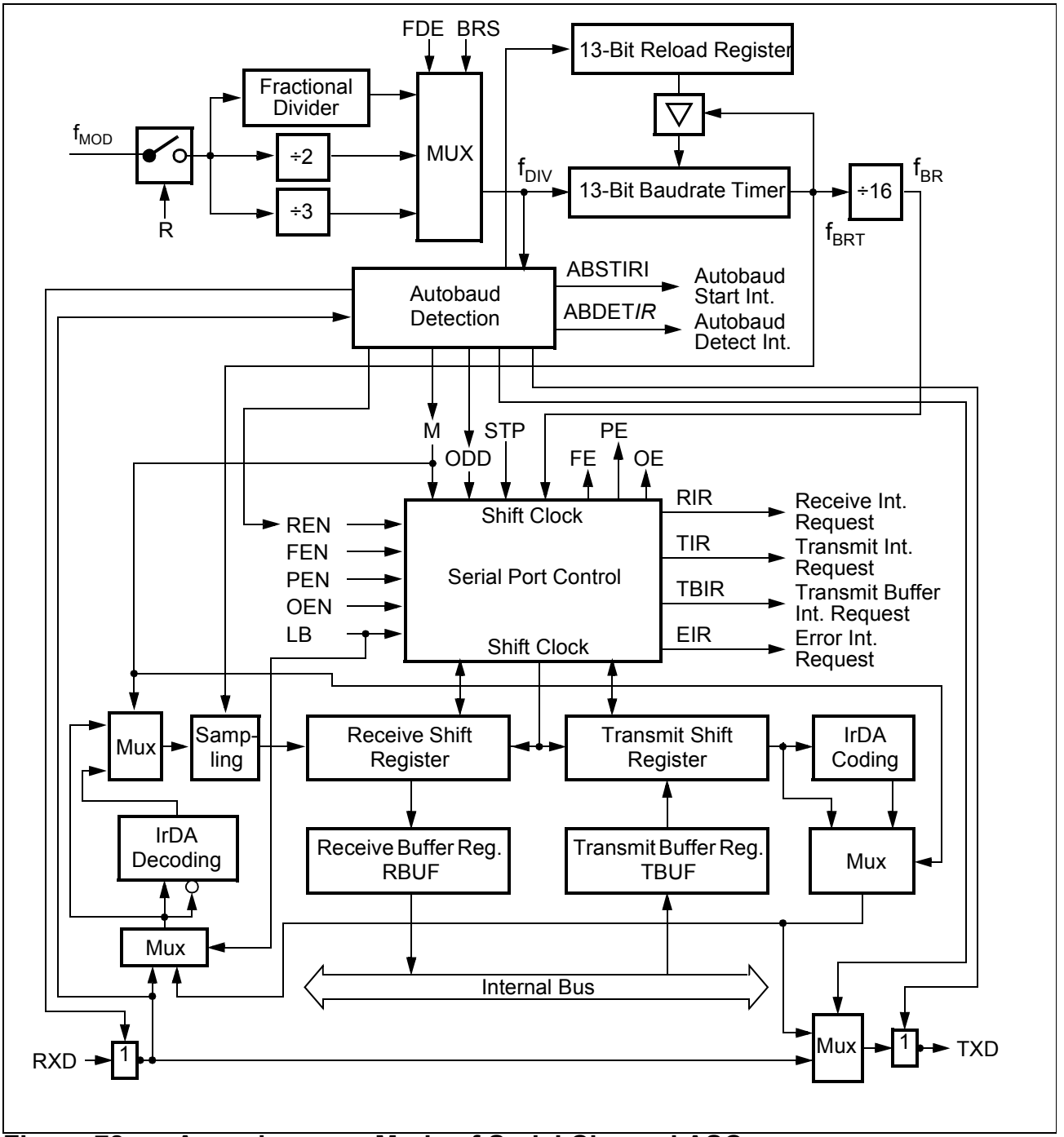


Figure 78 Asynchronous Mode of Serial Channel ASC

12.1.5.1 Asynchronous Data Frames

8-Bit Data Frames

8-bit data frames either consist of 8 data bits D7...D0 (CON_M='001_B'), or of 7 data bits D6...D0 plus an automatically generated parity bit (CON_M='011_B'). Parity may be odd or even, depending on bit CON_ODD. An even parity bit will be set, if the modulo-2-sum of the 7 data bits is '1'. An odd parity bit will be cleared in this case. Parity checking is enabled via bit CON_PEN (always OFF in 8-bit data mode). The parity error flag CON_PE will be set along with the error interrupt request flag, if a wrong parity bit is received. The parity bit itself will be stored in bit RBUF.7.

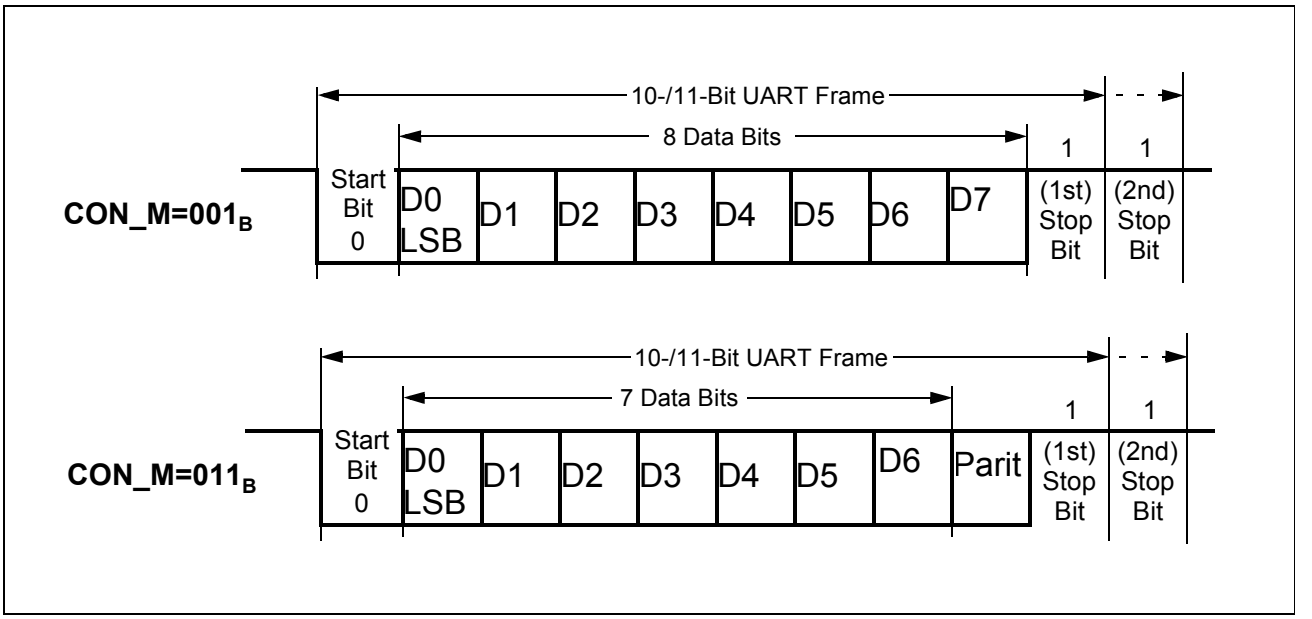


Figure 79 Asynchronous 8-Bit Frames

9-Bit Data Frames

9-bit data frames either consist of 9 data bits D8...D0 (CON_M='100_B'), of 8 data bits D7...D0 plus an automatically generated parity bit (CON_M='111_B') or of 8 data bits D7...D0 plus wake-up bit (CON_M='101_B'). Parity may be odd or even, depending on bit CON_ODD. An even parity bit will be set, if the modulo-2-sum of the 8 data bits is '1'. An odd parity bit will be cleared in this case. Parity checking is enabled via bit CON_PEN (always OFF in 9-bit data and wake-up mode). The parity error flag CON_PE will be set along with the error interrupt request flag, if a wrong parity bit is received. The parity bit itself will be stored in bit RBUF.8.

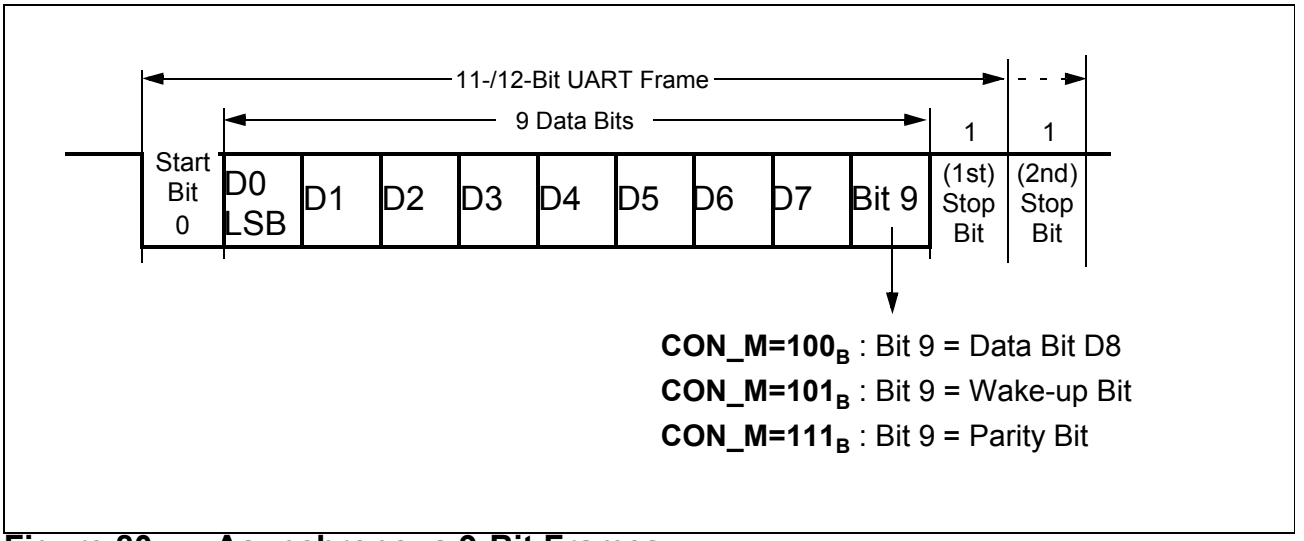


Figure 80 Asynchronous 9-Bit Frames

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 (eg. by clearing bit CON_M.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

IrDA Frames

The modulation schemes of IrDA is based on standard asynchronous data transmission frames. The asynchronous data format in IrDA mode (CON_M=010_B) is defined as follows :

1 start bit / 8 data bits / 1 stop bit

The coding/decoding of/to the asynchronous data frames is shown in **Figure 81**. In general, during the IrDA transmissions UART frames are encoded into IR frames and vice cersa. A low level on the IR frame indicates a “LED off“ state. A high level on the IR frame indicates a “LED on“ state.

For a “0“ bit in the UART frame a high pulse is generated. For a “1“ bit in the UART frame no pulse is generated. The high pulse starts in the middle of a bit cell and has a fixed

width of 3/16 of the bit time. The ASC also allows to program the length of the IrDA high pulse. Further, the polarity of the received IrDA pulse cane be inverted in IrAD mode. **Figure 81** shows the non-inverted IrDA pulse scheme.

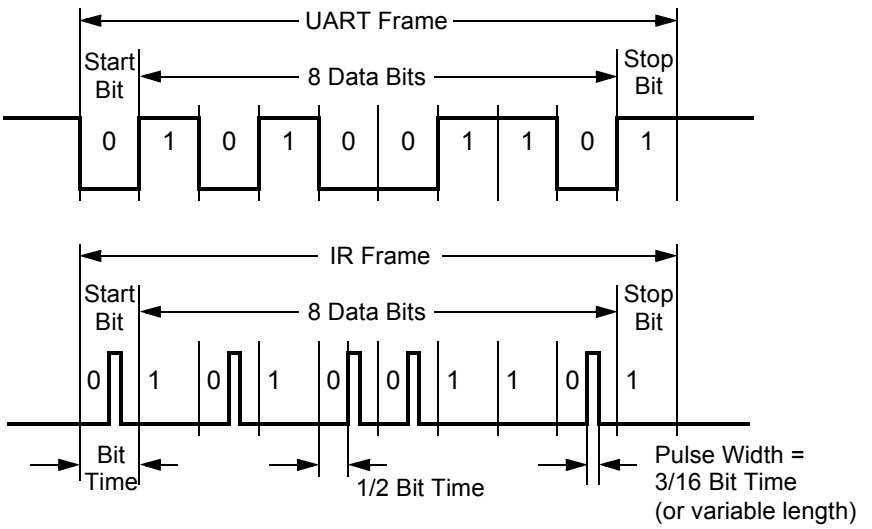


Figure 81 IrDA Frame Encoding/Decoding

12.1.5.2 Asynchronous Transmission

Asynchronous transmission begins at the next overflow of the divide-by-16 baudrate timer (transition of the baudrate clock f_{BR}), if bit S0CON.R 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 register 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 line TBIR being activated. S0TBUF may now be loaded with the next data, while transmission of the previous one is still going on.

The transmit interrupt request line TIR will be activated before the last bit of a frame is transmitted, ie. before the first or the second stop bit is shifted out of the transmit shift register.

The transmitter output pin P3.10/TXD must be configured for alternate data output’.

12.1.5.3 Asynchronous Reception

Asynchronous reception is initiated by a falling edge (1-to-0 transition) on pin P3.11/ RXD, provided that bits CON_R and CON_REN are set. The receive data input pin P3.11/RXD is sampled at 16 times the rate of the selected baudrate. A majority decision of the 7th, 8th and 9th sample determines the effective bit value. This avoids erroneous results that may be caused by noise.

If the detected value is not a '0' when the start bit is sampled, the receive circuit is reset and waits for the next 1-to-0 transition at pin P3.11/RXD. If the start bit proves valid, the receive circuit continues sampling and shifts the incoming data frame into the receive shift register.

When the last stop bit has been received, the content of the receive shift register is transferred to the receive data buffer register S0RBUF. Simultaneously, the receive interrupt request line RIR is activated 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 P3.11/RXD must be configured for input.

Asynchronous reception is stopped by clearing bit CON_REN. A currently received frame is completed including the generation of the receive interrupt request and an error interrupt request, if appropriate. Start bits that follow this frame will not be recognized.

Note: In wake-up mode received frames are 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.

12.1.5.4 IrDA Mode

The duration of the IrDA pulse is normally 3/16 of a bit period. The IrDA standard also allows the pulse duration being independent of the baudrate or bit period. In this case the transmitted pulse has always the width corresponding to the 3/16 pulse width at 115.2 kBaud which is 1.627 µs. Both, bit period dependend or fixed IrDA pulse width generation can be selected. The IrDA pulse width mode is selected by bit PMW_IRPW.

In case of fixed IrDA pulse width generation, the lower 8 bits in register PMW are used to adapt the IrDA pulse width to a fixed value of e.g. 1.627 µs. The fixed IrDA pulse width is generated by a programmable timer as shown in **Figure 82**.

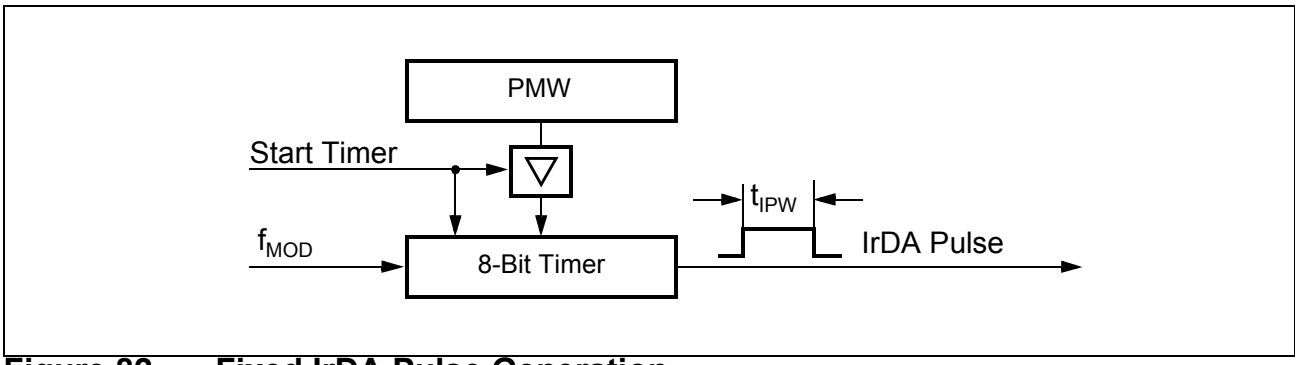


Figure 82 Fixed IrDA Pulse Generation

The IrDA pulse width can be calculated according the formulas given in **Table 56**.

Table 56 Formulas for the IrDA Pulse Width Calculation

PMW	PMW_IRPW	Formulas	
1 ... 255	0	$t_{IPW} = \frac{3}{16 \times Baudrate}$	$t_{IPW\ min} = \frac{(PMW >> 1)}{f_{MOD}}$
	1	$t_{IPW} = \frac{PMW}{f_{MOD}}$	

The name PMW in the formulas of **Table 56** represents the content of the reload register PMW (PW_VALUE), taken as unsigned 8-bit integer.

The content of PMW further defines the minimum IrDA pulse width ($t_{IPW\ min}$) which is still recognized during a receive operation as a valid IrDA pulse. This function is independent of the selected IrDA pulse width mode (fixed or variable) which is defined by bit PMW_IRPW. The minimum IrDA pulse width is calculated by a shift right operation of PMW bit 7-0 by one bit divided by the module clock f_{MOD}.

Note: If PMW_IRPW=0 (fixed IrDA pulse width), PW_VALUE must be a value which assures that $t_{IPW} > t_{IPW\ min}$.

Table 57 gives two examples for typical frequencies of the C161U: 36 MHz and 24 MHz..

Table 57 IrDA Pulse Width Adaption to 1.627 µs

f_{MOD}	PMW	t_{IPW}	Error	$t_{IPW\ min}$
24 MHz	39	1.625 µs	- 0.12 %	0.79 µs
36 MHz	59	1.639 µs	+ 0.74 %	0.81 µs

12.1.5.5 RXD/TXD Data Path Selection in Asynchronous Modes

The data paths for the serial input and output data of the ASC in asynchronous modes are affected by several control bits in the registers CON and ABCON as shown in **Figure 83**. The synchronous mode operation is not affected by these data path selection capabilities.

The input signal from RXD passes an inverter which is controlled by bit ABCON_RXINV. The output signal of this inverter is used for the autobaud detection and may bypass the ASC logic in the echo mode (controlled by bit ABCON_ABEM). Further, two multiplexers are in the RXD input signal path for providing the loopback mode capability (controlled by bit CON_LB) and the IrDA receive pulse inversion capability (controlled by bit CON_RXDI).

Depending on the asynchronous operating mode (controlled by bitfield CON_M), the ASC output signal or the RXD input signal in echo mode (controlled by bit ABCON_ABEM) is switched to the TXD output via an inverter (controlled by bit ABCON_TXINV).

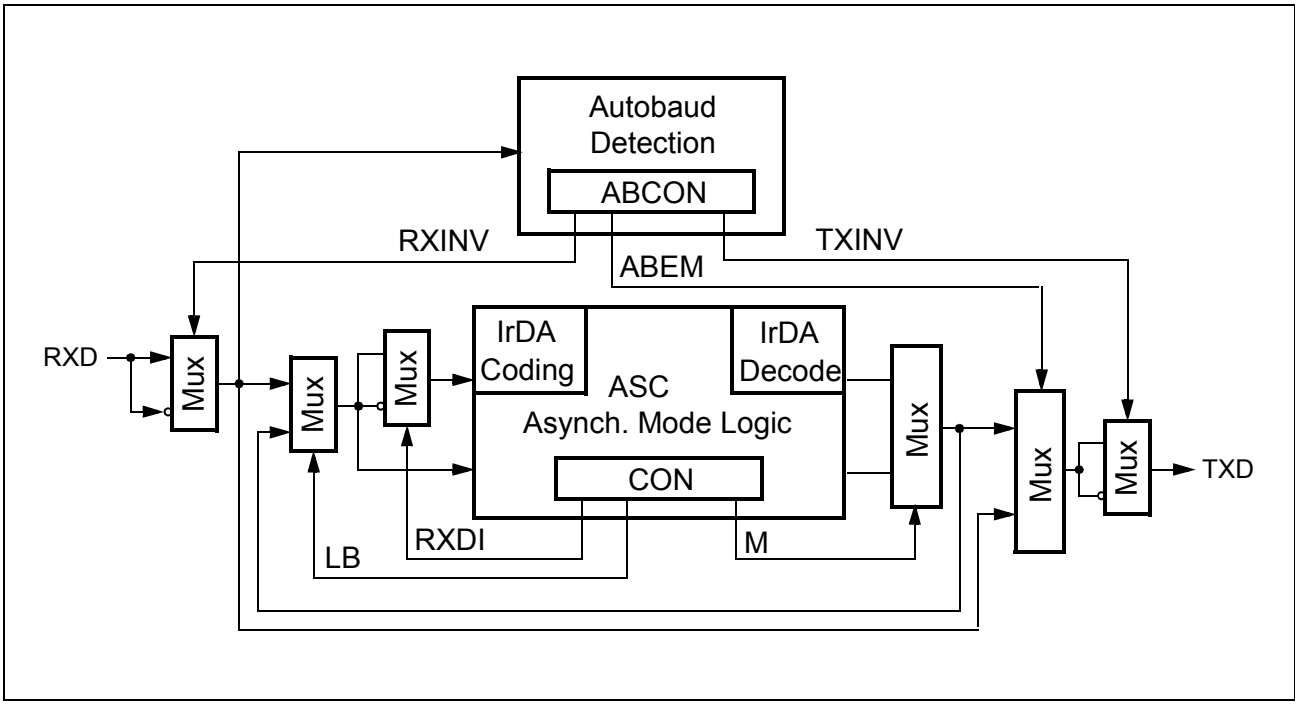


Figure 83 RXD/TXD Data Path in Asynchronous Modes (ASC)

Note: In echo mode the transmit output signal of the ASC logic is blocked by the echo mode output multiplexer. **Figure 83** also shows that it is not possible to use an IrDA coded receiver input signal for autobaud detection.

12.1.6 Synchronous Operation

Synchronous mode supports half-duplex communication, basically for simple I/O expansion via shift registers. Data is transmitted and received via pin RXD while pin TXD outputs the shift clock. These signals are alternate functions of port pins P3.11 and P3.10. Synchronous mode is selected with CON_M='000_B'.

Eight data bits are transmitted or received synchronous to a shift clock generated by the internal baudrate generator. The shift clock is only active as long as data bits are transmitted or received.

Note: The lines RXDI and RXDO are concatenated in the port logic to pin RXD.

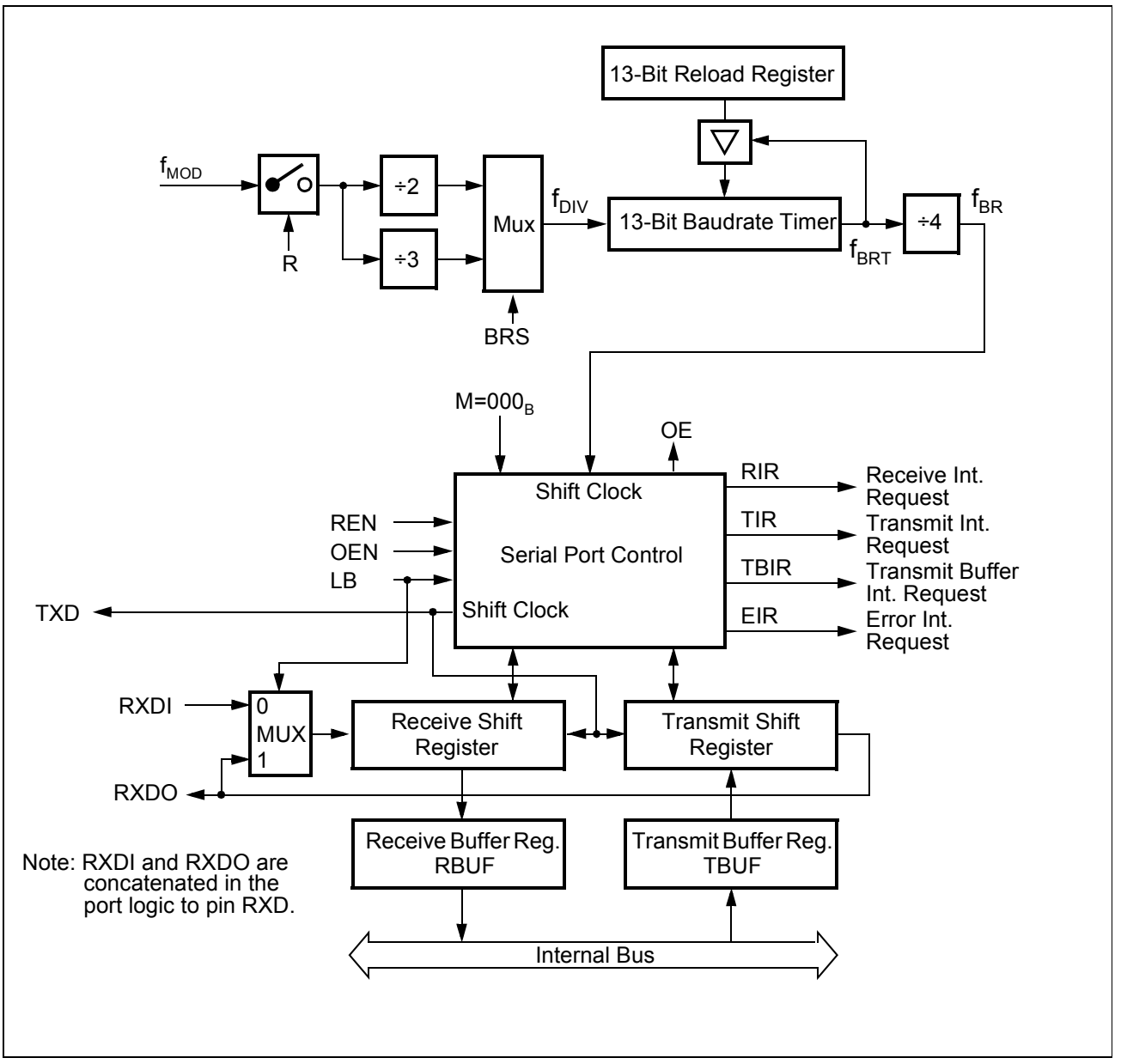


Figure 84 Synchronous Mode of Serial Channel ASC_P

12.1.6.1 Synchronous Transmission

Synchronous transmission begins within 4 state times after data has been loaded into S0TBUF provided that CON_R is set and CON_REN='0' (half-duplex, no reception). Exception : in loopback mode (bit CON_LB set), CON_REN must be set for reception of the transmitted byte. Data transmission is double buffered. When the transmitter is idle, the transmit data loaded into 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 line TBIR being activated. 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 TXD and RXD will go high, the transmit interrupt request line TIR is activated, and serial data transmission stops.

Pin P3.10/TXD must be configured for alternate data output in order to provide the shift clock. Pin P3.11/RXD must also be configured for output during transmission.

12.1.6.2 Synchronous Reception

Synchronous reception is initiated by setting bit CON_REN='1'. If bit CON_R=1, the data applied at RXD is clocked into the receive shift register synchronous to the clock which is output at pin TXD. After the 8th bit has been shifted in, the content of the receive shift register is transferred to the receive data buffer RBUF, the receive interrupt request line RIR is activated, the receiver enable bit CON_REN is reset, and serial data reception stops.

Pin P3.10/TXD must be configured for alternate data output in order to provide the shift clock. Pin P3.11/RXD must be configured as alternate data input.

Synchronous reception is stopped by clearing bit CON_REN. A currently received byte is completed including the generation of the receive interrupt request and an error interrupt request, if appropriate. Writing to the transmit buffer register while a reception is in progress has no effect on reception and will not start a transmission.

If a previously received byte has not been read out of the receive buffer register at the time the reception of the next byte is complete, both the error interrupt request line EIR and the overrun error status flag CON_OE will be activated/set, provided the overrun check has been enabled by bit CON_OEN.

12.1.6.3 Synchronous Timing

Figure 85 shows timing diagrams of the ASC synchronous mode data reception and data transmission. In idle state the shift clock is at high level. With the beginning of a synchronous transmission of a data byte the data is shifted out at RXD with the falling edge of the shift clock. If a data byte is received through RXD data is latched with the rising edge of the shift clock.

Between two consecutive receive or transmit data bytes one shift clock cycle (f_{BR}) delay is inserted.

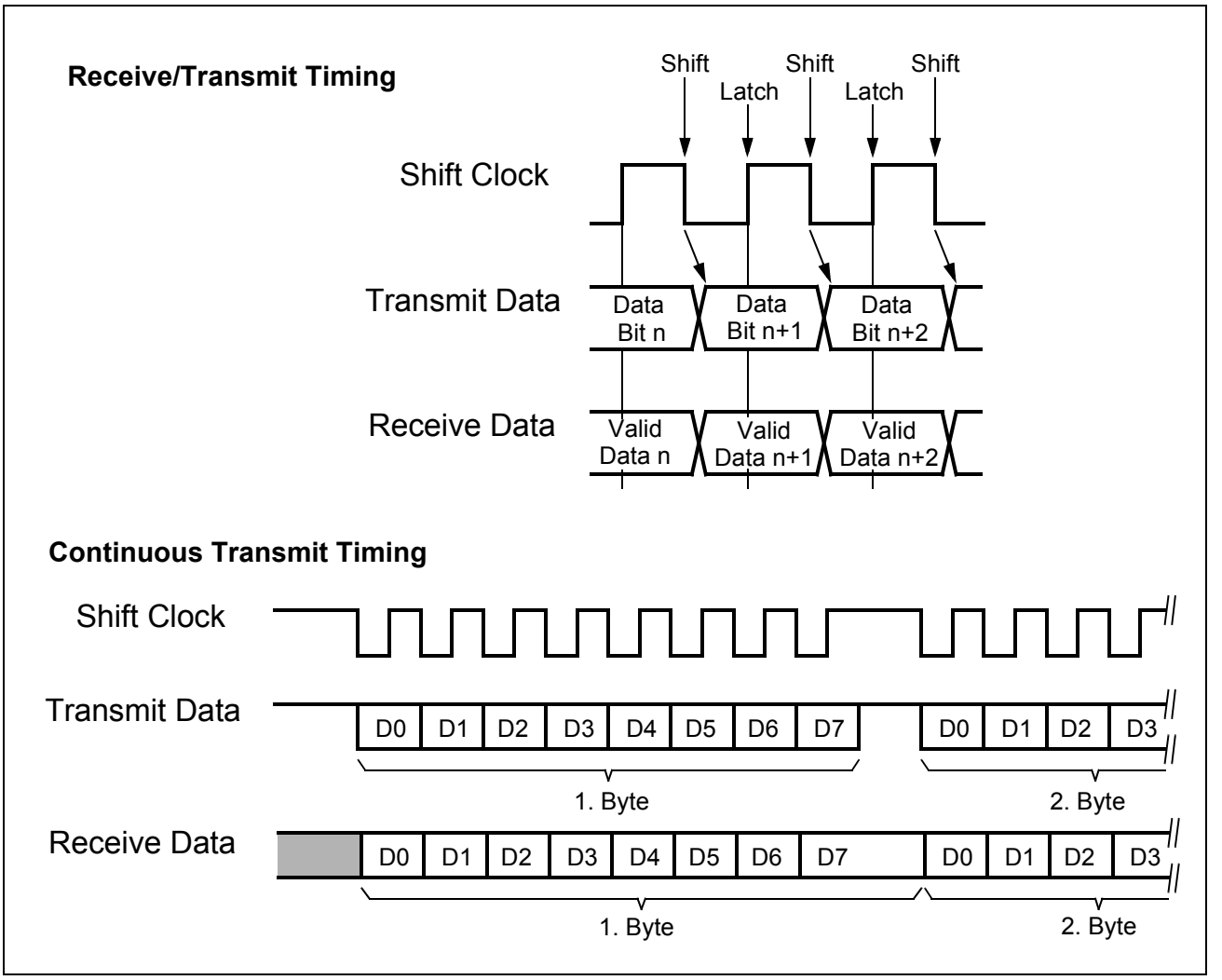


Figure 85 ASC_P3 Synchronous Mode Waveforms

12.1.7 Baudrate Generation

The serial channel ASC has its own dedicated 13-bit baudrate generator with 13-bit reload capability, allowing baudrate generation independent of the GPT timers.

The baudrate generator is clocked with a clock (f_{DIV}) which is derived via a prescaler from the ASC input clock f_{MOD}, e.g. 36 MHz. The baudrate timer is counting downwards and can be started or stopped through the baudrate generator run bit CON_R. Each underflow of the timer provides one clock pulse to the serial channel. The timer is reloaded with the value stored in its 13-bit reload register each time it underflows. The resulting clock f_{BRT} is again divided by a factor for the baudrate clock (± 16 in asynchronous modes and ± 4 in synchronous mode). The prescaler is selected by the bits CON_BRS and CON_FDE. In the asynchronous operating modes, additionally to the two fixed dividers a fractional divider prescaler unit is available which allows to select prescaler divider ratios of n/512 with n=0-511. Therefore, the baudrate of ASC is determined by the module clock, the content of S0FDV, the reload value of S0BG and the operating mode (asynchronous or synchronous).

Register S0BG is the dual-function Baudrate Generator/Reload register. Reading BG returns the content of the timer BR_VALUE (bits 15...13 return zero), while writing to S0BG always updates the reload register (bits 15...13 are insiginificant).

An auto-reload of the timer with the content of the reload register is performed each time CON_BG is written to. However, if CON_R=’0’ at the time the write operation to BG is performed, the timer will not be reloaded until the first instruction cycle after CON_R=’1’. For a clean baudrate initialization S0BG should only be written if CON_R=’0’. If S0BG is written with CON_R=’1’, an unpredicted behaviour of the ASC may occur during running transmit or receive operations.

12.1.7.1 Baudrates in Asynchronous Mode

For asynchronous operation, the baudrate generator provides a clock f_{BRT} with 16 times the rate of the established baudrate. Every received bit is sampled at the 7th, 8th and 9th cycle of this clock. The clock divider circuitry, which generates the input clock for the 13-bit baudrate timer, is extended by a fracxtional divider circuitry, which allows the adjustment of more accurate baudrates and the extension of the baudrate range.

The baudrate of the baudrate generator depends on the following input clock, bits and register values :

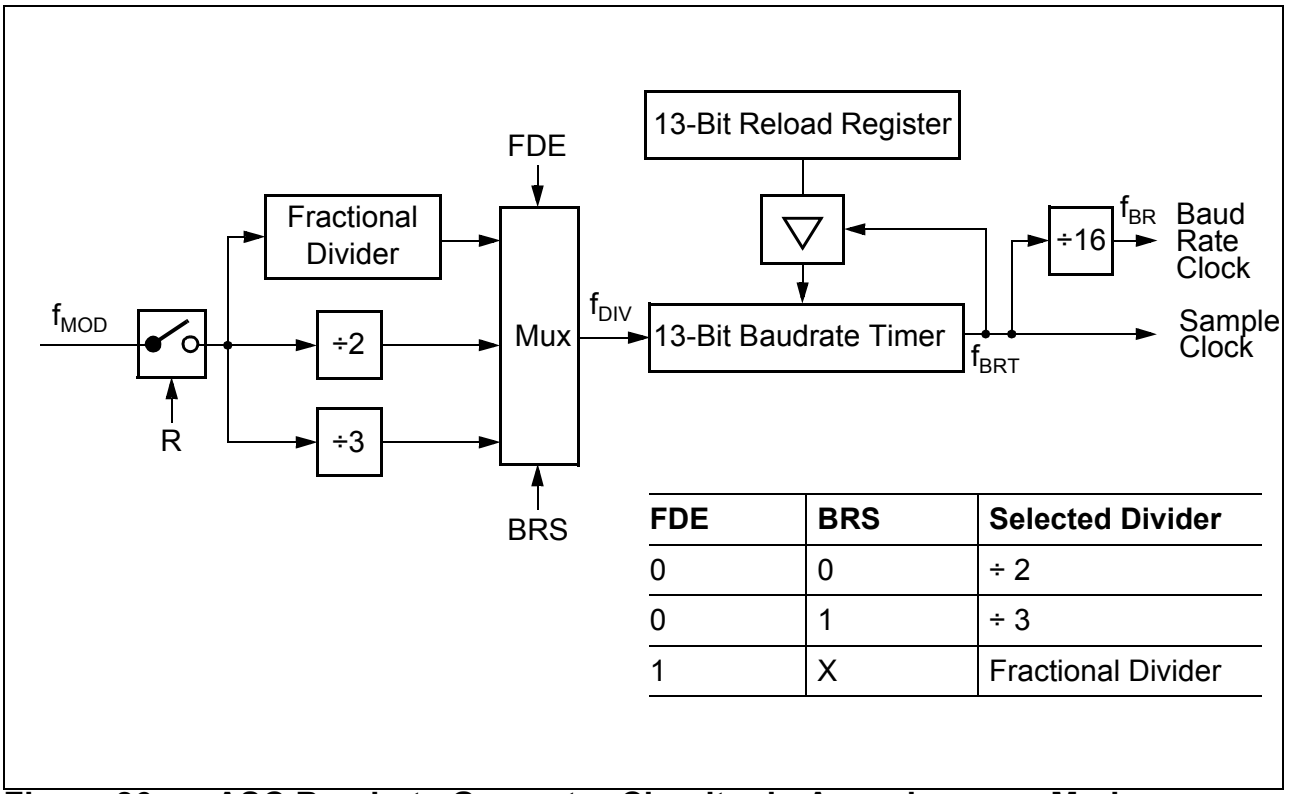
Input clock f_{MOD}

Selection of the baudrate timer input clock f_{DIV} by bits CON_FDE and CON_BRS

If bit CON_FDE=1 (fractional divider) : value of register CON_FDV

value of the 13-bit reload register S0BG

The output clock of the baudrate timer with the reload register is the sample clock in the asynchronous modes of the ASC. For baudrate calculations, this baudrate clock f_{BR} is derived from the sample clock f_{DIV} by a division by 16.



FDE	BRS	Selected Divider
0	0	÷ 2
0	1	÷ 3
1	X	Fractional Divider

Figure 86 ASC Baudrate Generator Circuitry in Asynchronous Modes

Using the fixed Input Clock Divider

The baudrate for asynchronous operation of serial channel ASC when using the fixed input clock divider ratios (CON_FDE=0) and the required reload value for a given baudrate can be determined by the following formulas :

Table 58 Asynchronous Baudrate Formulas using the Fixed Input Clock Dividers

FDE	BRS	BG	Formula
0	0	0 ... 8191	$\text{Baudrate} = \frac{f_{MOD}}{32 \times (BG+1)}$ $BG = \frac{f_{MOD}}{32 \times \text{Baudrate}} - 1$
	1		$\text{Baudrate} = \frac{f_{MOD}}{48 \times (BG+1)}$ $BG = \frac{f_{MOD}}{48 \times \text{Baudrate}} - 1$

BG represents the contents of the reload register S0BG (BR_VALUE), taken as unsigned 13-bit integer.

The maximum baudrate that can be achieved for the asynchronous modes when using the two fixed clock dividers and a module clock of 36 MHz is 1.125 MBaud. **Table 59** below lists various commonly used baudrates together with the required reload values and the deviation errors compared to the intended baudrate.

Table 59 Typical Asynchronous Baudrates using the Fixed Input Clock Dividers

Baudrate		BRS = '0', f_{MOD} = 36 MHz		BRS = '1', f_{MOD} = 36 MHz	
		Deviation Error	Reload Value	Deviation Error	Reload Value
1.125	MBaud	---	0000_H	---	---
750.0	kBaud	---	---	---	0000_H
19.2	kBaud	- 0.69 %	$003A_H$	+ 0.16 %	0026_H
9600	Baud	+ 0.16 %	0074_H	+ 0.16 %	$004D_H$
4800	Baud	+ 0.16 %	$00E9_H$	+ 0.16 %	$009B_H$
2400	Baud	+ 0.16 %	$01D3_H$	+ 0.16 %	0137_H
1200	Baud	+ 0.05 %	$03A8_H$	+/- 0.0 %	0270_H

Note: CON_FDE must be 0 to achieve the baudrates in the table above. The deviation errors given in the table above are rounded. Using a baudrate crystal will provide correct baudrates without deviation errors.

Using the Fractional Divider

When the fractional divider is selected, the input clock f_{DIV} for the baudrate timer is derived from the module clock f_{MOD} by a programmable divider. If CON_FDE=1, the fractional divider is activated, It divides f_{MOD} by a fraction of n/512 for any value of n from 0 to 511. If n=0, the divider ratio is 1 which means that $f_{DIV}=f_{MOD}$. In general, the fractional divider allows to program the baudrrate with a much better accuracy than with the two fixed prescaler divider stages.

BG represents the content of the reload register S0BG (BR_VALUE), taken as unsigned 13-bit integer. FDV represents the content of the fractional divider register S0FDV (FD_VALUE) taken as unsigned 9-bit integer. For example, typical asynchronous baudrates are shown in **Table 61**.

Using the fractional divider and a module clock of 36 MHz (equal to the C161U CPU clock) the available baudrate range is 2.25 MBaud down to 0.5364 Baud.

.

Table 60 Asynchronous Baudrate Formulas using the Fractional Input Clock Divider

FDE	BRS	BG	FDV	Formula
1	X	1 ... 8191	1 ... 511	$\text{Baudrate} = \frac{\text{FDV}}{512} \times \frac{f_{MOD}}{16 \times (BG+1)}$
			0	$\text{Baudrate} = \frac{f_{MOD}}{16 \times (BG+1)}$

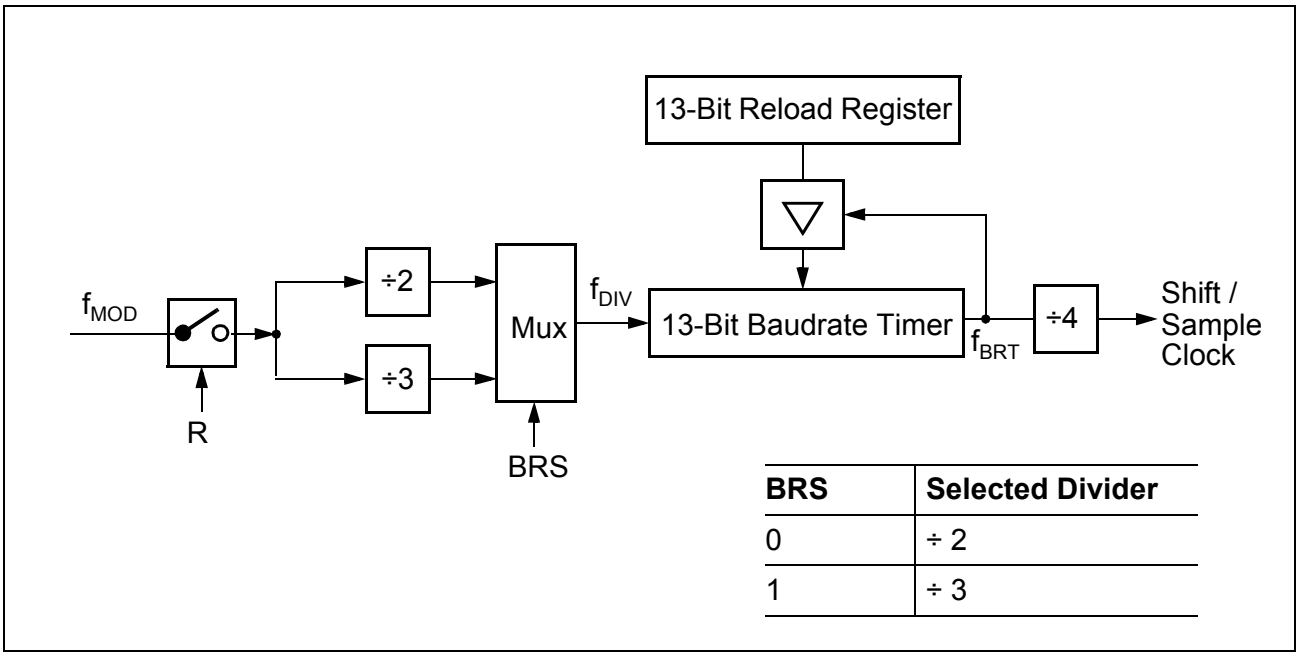
Table 61 Typical Asynchronous Baudrates using the Fractional Input Clock Divider

f_{MOD}	Desired Baudrate	BG	FDV	Resulting Baudrate	Deviation
36 MHz	max. Baudrate	0	0	2.25 MBaud	0 %
	230.4 kBaud	6	367	230.399 kBaud	< 0.01 %
	115.2 kBaud	13	367	115.199 kBaud	< 0.01 %
	57.6 kBaud	27	367	57.5997 kBaud	< 0.01 %
	38.4 kBaud	41	367	38.3998 kBaud	< 0.01 %
	19.2 kBaud	83	367	19.1999 kBaud	< 0.01 %
	min. Baudrate	8191	1	0.53644 Baud	0 %

Note: ApNote AP2423 provides a program 'ASC.EXE' which allows to calculate values for the S0FDV and S0BG registers depending on f_{MOD}, the requested baudrate, and the maximum deviation. Please contact your Infineon Technologies representative.

12.1.7.2 Baudrates in Synchronous Mode

For synchronous operation, the baudrate generator provides a clock with 4 times the rate of the established baudrate.(see **Figure 87**).



BRS	Selected Divider
0	÷ 2
1	÷ 3

Figure 87 ASC Baudrate Generator Circuitry in Synchronous Mode

The baudrate for synchronous operation of serial channel ASC can be determined by the formulas as shown in **Table 62**.

Table 62 Synchronous Baudrate Formulas

BRS	BG	Formula
0	0 ... 8191	$\text{Baudrate} = \frac{f_{MOD}}{8 \times (BG+1)}$ $\quad BG = \frac{f_{MOD}}{8 \times \text{Baudrate}} - 1$
1		$\text{Baudrate} = \frac{f_{MOD}}{12 \times (BG+1)}$ $\quad BG = \frac{f_{MOD}}{12 \times \text{Baudrate}} - 1$

BG represents the content of the reload register S0BR (BR_VALUE), taken as unsigned 13-bit integers.

The maximum baudrate that can be achieved in synchronous mode when using a module clock of 36 MHz is 4.5 MBaud.

12.1.8 Autobaud Detection

12.1.8.1 General Operation

The autobaud detection unit in the ASC provides a capability to recognize the mode and the baudrate of an asynchronous input signal at RXD. Generally, the baudrates to be recognized must be known by the application. With this knowledge always a set of nine baudrates can be detected. The autobaud detection unit is not designed to calculate a baudrate of an unknown asynchronous frame.

Figure 88 shows how the autobaud detection unit of the ASC is integrated into its asynchronous mode configuration. The RXD data line is an input to the autobaud detection unit. The clock f_{DIV} which is generated by the fractional divider is used by the autobaud detection unit as time base. After successful recognition of baudrate and asynchronous operating mode of the RXD data input signal, bits in the S0CON register and the value of the S0BG register in the baudrate timer are set to the appropriate values, and the ASC can start immediately with the reception of serial input data.

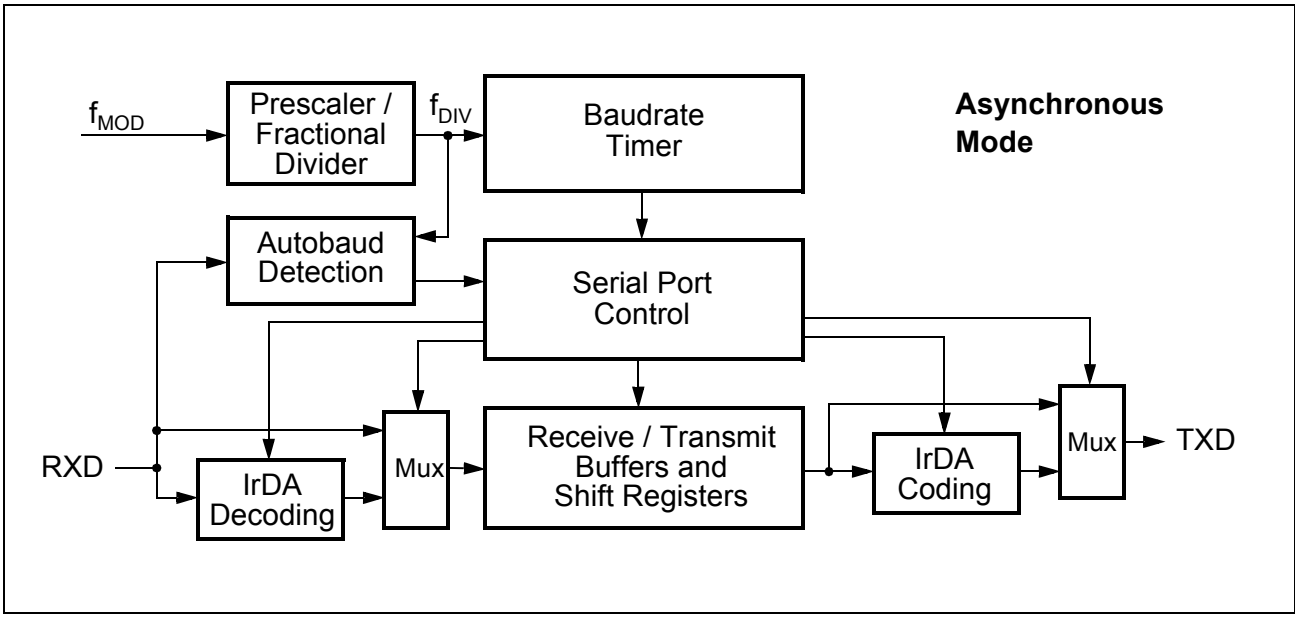


Figure 88 ASC Asynchronous Mode Block Diagram

The following sequence must be generally executed to start the autobaud detection unit for operation :

Definition of the baudrates to be detected : standard or non-standard baudrates

Programming of the Prescaler/Fractional Divider to select a specific value of f_{DIV}

Starting the Prescaler/Fractional Divider (setting CON_R)

Preparing the interrupt system of the CPU

Enabling the autobaud detection (setting ABCON_ EN and the interrupt enable bits in ABCON for interrupt generation, if required)

Polling interrupt request flag or waiting for the autobaud detection interrupt

12.1.8.2 Serial Frames for Autobaud Detection

The autobaud detection of the ASC is based on the serial reception of a specific two-byte serial frame. This serial frame is build up by the two ASCII bytes "at" or "AT" ("aT" or "At" are not allowed). Both byte combinations can be detected in five types of asynchronous frames. **Figure 89** and **Figure 90** show the serial frames which are detected at least.

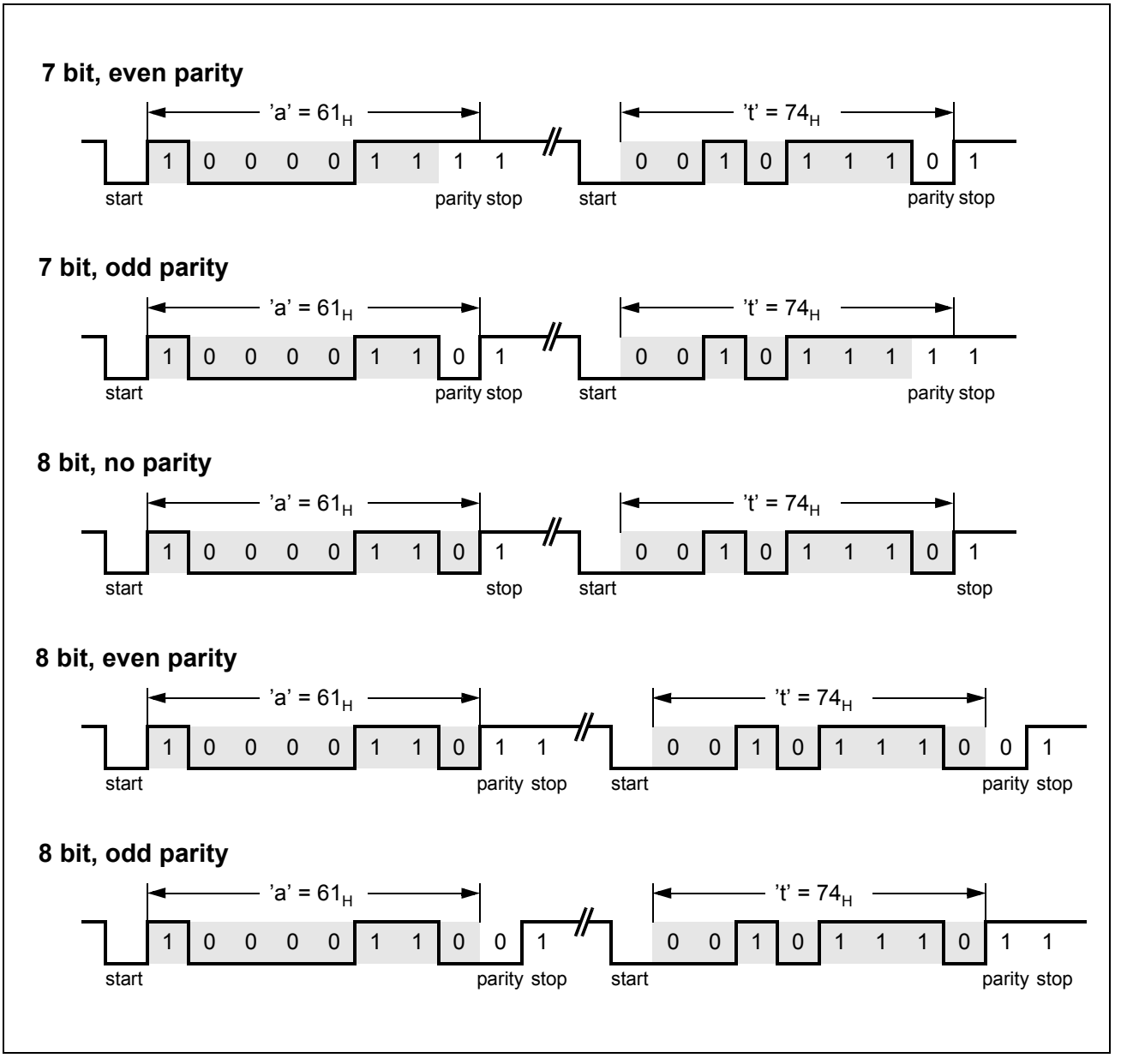


Figure 89 Two-Byte Serial Frames with ASCII 'at'

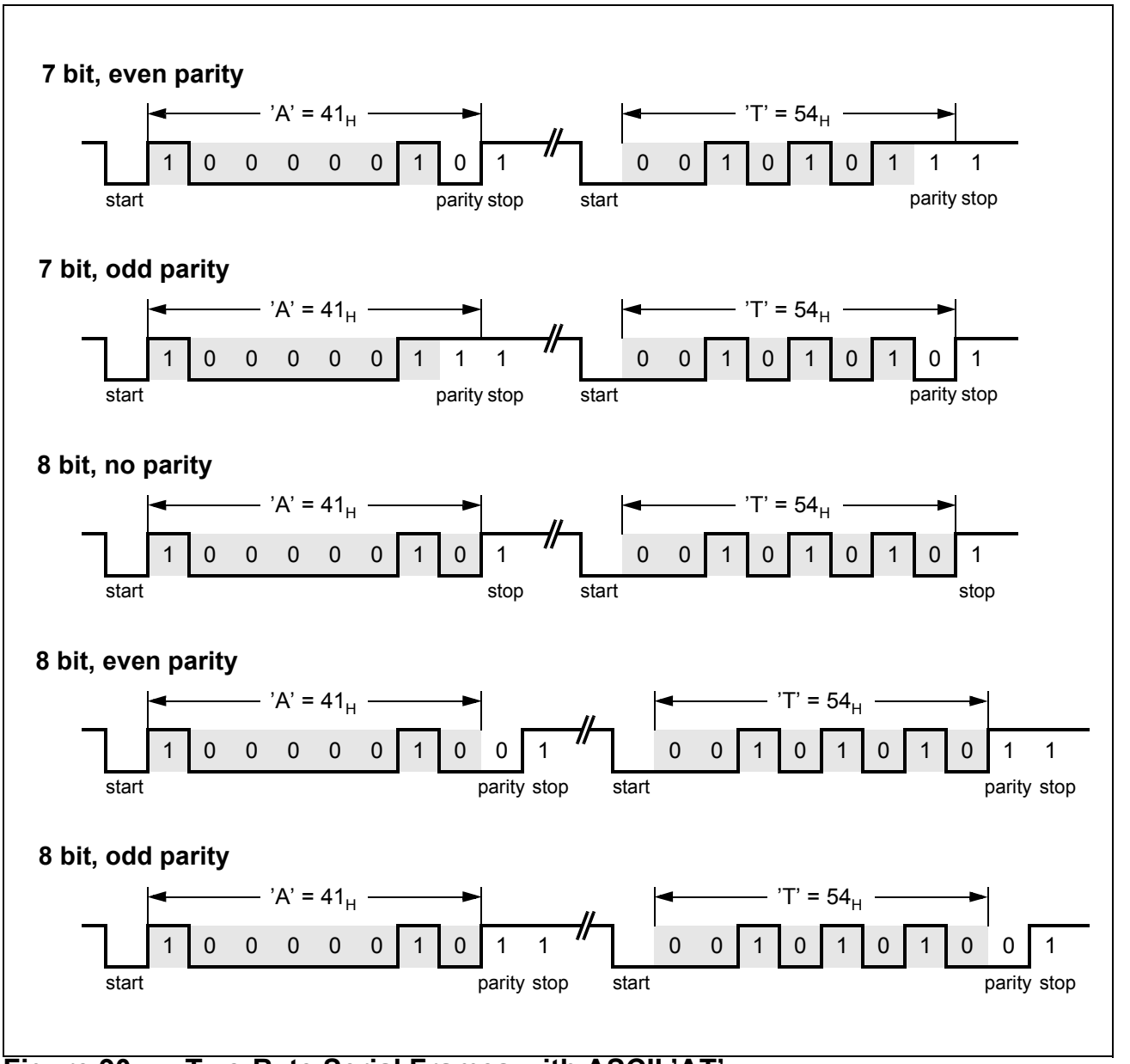


Figure 90 Two-Byte Serial Frames with ASCII 'AT'

12.1.8.3 Baudrate Selection and Calculation

The autobaud detection requires some calculations concerning the programming of the baudrate generator and the baudrates to be detected. Two steps must be considered :

Defining the baudrate(s) to be detected

Programming of the baudrate timer prescaler - setup of the clock rate of f_{DIV}

In general, the baudrate generator of the ASC in asynchronous mode is build up by two parts (see also **Figure 86**) :

the clock prescaler part which derives f_{DIV} from f_{MOD}

the baudrate timer part which generates the sample clock f_{BRT} and the baudrate clock f_{BR}

Prior to an autobaud detection the prescaler part has to be setup by the CPU while the baudrate timer (register BG) is initialized with a 13-bit value (BR_VALUE) automatically after a successfull autobaud detection. For the following calculations, the fractional divider is used (CON_FDE = 1).

Note: It is also possible to use the fixed divide-by-2 or divide-by-3 prescaler. But the fractional divider allows to adapt f_{DIV} much more precise to the required value.

Standard Baudrates

For standard baudrate detection the baudrates as shown in **Table 63** can be e.g. detected. Therefore, the output frequency f_{DIV} of the ASC baudrate generator must be set to a frequency derived from the module clock f_{MOD} in a way that it is equal to 11.0592 MHz. The value to be written into register FDV is the nearest integer value which is calculated according the following formula :

$$FDV = \frac{512 \times 11.0592\ MHz}{f_{MOD}}$$

Table 63 defines the nine standard baudrates (Br0 - Br8) which can be detected for f_{DIV}=11.0592 MHz.

Table 63 Autobaud Detection using Standard Baudrates (f_{DIV} = 11.0592 MHz)

Baudrate Numbering	Detectable Standard Baudrate	Divide Factor d_f	BG is loaded after detection with value
Br0	230.400 kBaud	48	2 = 002_H
Br1	115.200 kBaud	96	5 = 005_H
Br2	57.600 kBaud	192	11 = $00B_H$
Br3	38.400 kBaud	288	17 = 011_H
Br4	19.200 kBaud	576	35 = 023_H
Br5	9600 Baud	1152	71 = 047_H
Br6	4800 Baud	2304	143 = $08F_H$
Br7	2400 Baud	4608	287 = $11F_H$
Br8	1200 Baud	9216	**575= $23F_H$**

According **Table 63** a baudrate of 9600 Baud is achieved when register BG is loaded with a value of 047_H, assuming that f_{DIV} has been set to 11.0592 MHz.
Table 63 also lists a divide factor d_f which is defined with the following formula :

$$Baudrate = \frac{f_{DIV}}{d_f}$$

This divide factor d_f defines a fixed relationship between the prescaler output frequency f_{DIV} and the baudrate to be detected during the autobaud detection operation. This means, changing f_{DIV} results in a totally different baudrate table in means of baudrate values. For the baudrates to be detected, the following relations are always valid :

– $Br0 = f_{DIV} / 48_D$, $Br1 = f_{DIV} / 96_D$, ... up to $Br8 = f_{DIV} / 9216_D$,

A requirement for detecting standard baudrates up to 230.400 kBaud is the f_{DIV} minimum value of 11.0592 MHz. With the value FD_VALUE in register FDV the fractional divider f_{DIV} is adapted to the module clock frequency f_{MOD}. **Table 64** defines the deviation of the standard baudrates when using autobaud detection depending on the module clock f_{MOD}.

Table 64 Standard Baudrates - Deviations and Errors for Autobaud Detection

f_{MOD}	FDV	Error in f_{DIV}
10 MHz	not possible	
12 MHz	472	+ 0.03 %
13 MHz	436	+ 0.1 %
16 MHz	354	+ 0.03 %
18 MHz	315	+ 0.14 %
18.432 MHz	307	- 0.07 %
20 MHz	283	- 0.04 %
24 MHz	236	+ 0.03 %
25 MHz	226	- 0.22 %
30 MHz	189	+ 0.14 %
33 MHz	172	+ 0.24 %
36 MHz	157	+ 0.18 %

Note: If the deviation of the baudrate after autobaud detection is to high, the baudrate generator (fractional divider FDV and reload register BG) can be reprogrammed if required to get a more precise baudrate with less error.

Non-Standard Baudrates

Due to the relationship between Br0 to Br8 in **Table 63** concerning the divide factor d_f other baudrates than the standard baudrates can be also selected. E.g. if a baudrate of 50 kBaud has to be detected, Br2 is e.g. defined as baudrate for the 50 kBaud setection. This further results in :

– f_{DIV} = 50 kBaud x d_f@Br2 = 50 kBaud x 192 = 9.6 MHz

Therefore, depending on the module clock frequency f_{MOD}, the value of the fractional divider (register S0FDV must be set in this example according the formula :

$$FDV = \frac{512 \times f_{DIV}}{f_{MOD}} \quad \text{with } f_{DIV} = 9.6 \text{ MHz}$$

Using this selection (f_{DIV} = 9.6 MHz), the detectable baudrates start at 200 kBaud (Br0) down to 1042 Baud (Br8). **Table 65** shows the baudrate table for this example.

Table 65 Autobaud Detection using Non-Standard Baudrates (f_{DIV} = 9.6 MHz)

Baudrate Numbering	Detectable Non-Standard Baudrates	Divide Factor d_f	BG is loaded after detection with value
Br0	200.000 kBaud	48	2 = 002_H
Br1	100.000 kBaud	96	5 = 005_H
Br2	50 kBaud	192	11 = $00B_H$
Br3	33.333 kBaud	288	17 = 011_H
Br4	16.667 kBaud	576	35 = 023_H
Br5	8333 Baud	1152	71 = 047_H
Br6	4167 Baud	2304	143 = $08F_H$
Br7	2083 Baud	4608	287 = $11F_H$
Br8	1047 Baud	9216	575 = $23F_H$

12.1.8.4 Overwriting Registers on Successful Autobaud Detection

With a successfull autobaud detection some bits in register S0CON and S0BG are automatically set to a value which corresponds to the mode and baudrate of the detected serial frame conditions (see **Table 66**). In control register S0CON the mode control bits

CON_M and the parity select bit CON_ODD are overwritten. Register S0BG is loaded with the 13-bit reload value for the baudrate timer.

Table 66 Autobaud Detection Overwrite Values for the S0CON Register

Detected Parameters		CON_M	CON_ODD	BG_BR_VALUE
Operating Mode	7 bit, even parity 7 bit, odd parity 8 bit, even parity 8 bit, odd parity 8 bit, no parity	0 1 1 0 1 1 1 1 1 1 1 1 0 0 1	0 1 0 1 0	-
Baudrate	Br0 Br1 Br2 Br3 Br4 Br5 Br6 Br7 Br8	-	-	2 = 002_H 5 = 005_H 11 = $00B_H$ 17 = 011_H 35 = 023_H 71 = 047_H 143 = $08F_H$ 287 = $11F_H$ 575 = $23F_H$

Note: The autobaud detection interrupts are described in Chapter 12.1.10.

12.1.9 Hardware Error Detection Capabilities

To improve the safety of serial data exchange, the serial channel ASC 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 line EIR will be activated simultaneously with the receive interrupt request line RIR, if one or more of the following conditions are met :

the framing error detection enable bit CON_FEN is set and any of the expected stop bits is not high, the framing error flag CON_FE is set, indicating that the error interrupt request is due to a framing error (Asynchronous mode only).

If the parity error detection enable bit CON_PEN is set in the modes where a parity bit is received, and the parity check on the received data bits proves false, the parity error flag CON_PE is set, indicating that the error interrupt request is due to a parity error (Asynchronous mode only).

If the overrun error detection enable bit CON_OEN is set and the last character received was not read out of the receive buffer by software or DMA transfer at the time the reception of a new frame is complete, the overrun error flag CON_OE is set indicating that the error interrupt request is due to an overrun error (Asynchronous and synchronous mode).

12.1.10 Interrupts

Six interrupt sources are provided for serial channel ASC. Line TIR indicates a transmit interrupt, TBIR indicates a transmit buffer interrupt, RIR indicates a receive interrupt and EIR indicates an error interrupt of the serial channel. The autobaud detection unit provides two additional interrupts, the ABSTIR start of autobaud operation interrupt and the ABDETIR autobaud detected interrupt. The interrupt output lines TBIR, TIR, RIR, EIR, ABSTIR, and ABDETIR are activated (active state) for two periods of the module clock f_{MOD}four.

The cause of an error interrupt request (framing, parity, overrun error) can be identified by the error status flags FE, PE, and OE which are located in control register CON. For the two autobaud detection interrupts register ABSTAT provides status information.

Note: In contrary to the error interrupt request line EIR, the error status flags FE/PE/OE are not reset automatically but must be cleared by software.

For normal operation (ie. besides the error interrupt) the ASC provides three interrupt requests to control data exchange via this serial channel:

- TBIR is activated when data is moved from TBUF to the transmit shift register.
- TIR is activated before the last bit of an asynchronous frame is transmitted, or after the last bit of a synchronous frame has been transmitted.
- RIR is activated when the received frame is moved to RBUF.

The transmitter is serviced by two interrupt handlers. This provides advantages for the servicing software.

For single transfers, it is sufficient to use the transmitter interrupt (TIR), which indicates that 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 (TBIR) to reload transmit data gives the time to transmit a complete frame for the service routine, as TBUF may be reloaded while the previous data is still being transmitted.

The ABSTIR start of autobaud operation interrupt is generated whenever the autobaud detection unit is enabled (ABEN and ABDETEN and ABSTEN set), and a start bit has been detected at RXD. In this case ABSTIR is generated during autobaud detection whenever a start bit is detected.

The ABDETIR autobaud detected interrupt is always generated after recognition of the second character of the two-byte frame, this means after a successful autobaud detection. If ABCON_FCDETEN is set the ABDETIR autobaud detected interrupt is also generated after the recognition of the first character of the two-byte frame.

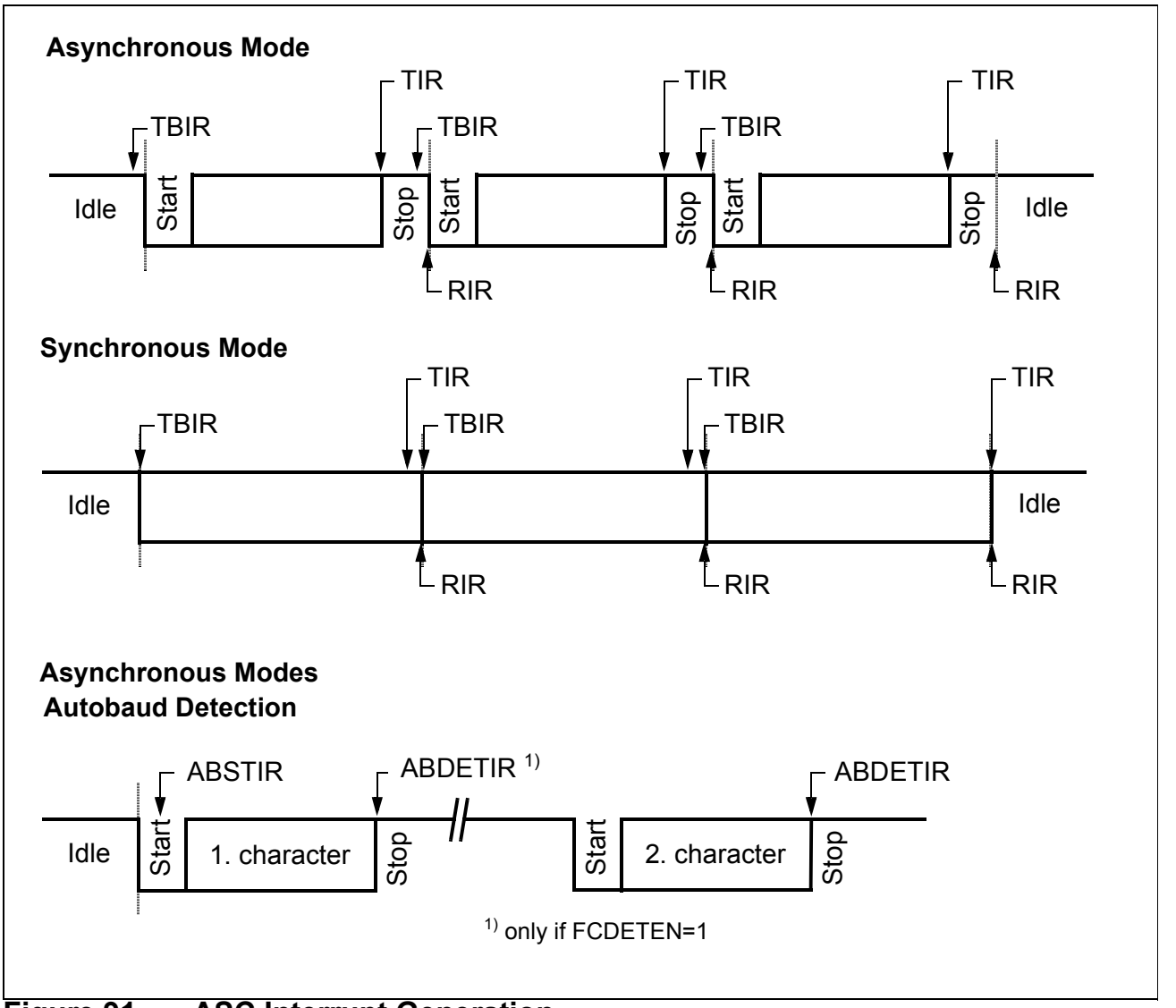


Figure 91 ASC Interrupt Generation

As shown in **Figure 91**, TBIR is an early trigger for the reload routine, while TIR indicates the completed transmission. Software using handshake therefore should rely on TIR at the end of a data block to make sure that all data has really been transmitted.

13 Real Time Clock (RTC)

13.1 Introduction

Real Time Clock (RTC) module of the C161U basically is an independent timer chain and counts time ticks. The base frequency of the RTC can be programmed via a reload counter. The RTC can work fully asynchronous to the system frequency and is optimized on low power consumption.

13.1.1 Features

The RTC serves for different purposes:

– System clock to determine the current time and date
– Cyclic time based interrupt
– Alarm interrupt for wake up on a defined time
– 48-bit timer for long term measurements

13.1.2 Overview

The real time clock module provides three different types of registers: two control registers for controlling the RTC´s functionality, three data registers for setting the clock divider for RTC base frequency programming and for flexible interrupt generation, and three counter registers they contain the actual time and date. The interrupts are programmed via one interrupt sub node register.

13.2 Function Description

RTC module consists of a chain of 2 divider blocks, the reloadable 16-bit timer T14 and the 32-bit RTC timer (accessible via registers RTCH and RTCL). Both timers count up. Timer T14 is reloaded with the value of register T14REL on every timer T14 overflow.

The count input of the RTC module (RTC_REF_CLK) can be optional divided by a prescaler with factor 8, see **Figure 93**.

The RTC module operates in two different modes, an asynchronous and a synchronous mode. In synchronous mode the RTC module is clocked with a synchronous clock referring to the CPU clock (RTC clock). In asynchronous mode the RTC module is clocked with the asynchronous counting input clock (RTC_REF_CLK). The asynchronous mode is necessary in case of a very low or disabled CPU clock (eg. sleep mode).

The mode control (asynchronous / synchronous) of the RTC is described as follows:

- The RTC is switched to asynchronous mode if SYSCON2.RCS = '1'
- The RTC is switched to asynchronous mode if device is in sleep mode.

- RTC is switched to asynchronous mode if the system is not in slowdown mode and the system clock is not locked (observe selected CLK_CFG as set during Reset).

This means that the mode is controlled by bit RCS of register SYSCON2 (see page 390) unless the system's clock setting indicates that no or an unreliable CPU clock is available. In the latter case, synchronous mode is not working and thus the RTC is forced into asynchronous mode.

As long as the CPU clock is four times faster than the RTC_REF_CLK, the RTC module can be operated in synchronous mode. Otherwise the asynchronous mode has to be selected by software.

In asynchronous mode no writing but only asynchronous reading to the registers via the internal bus is possible.

13.2.1 RTC Block Diagram

Figure 92 shows the RTC block diagram:

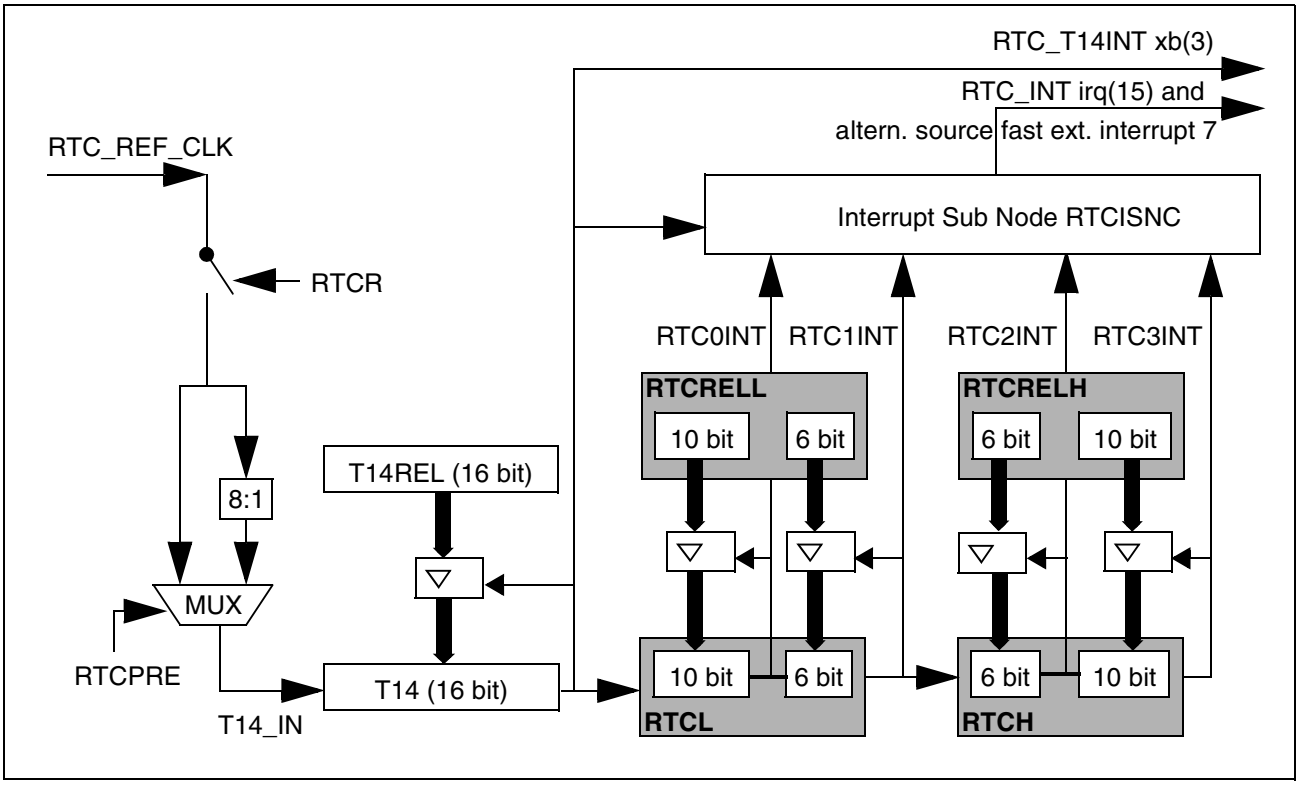


Figure 92 RTC Block Diagram

13.2.2 RTC Control

The operating behaviour of the RTC module is controlled by the RTCCON register. The RTC starts counting by setting the run bit RTCR. After reset the run bit is set and the RTC automatically starts operation. The bit RTCPRE selects a prescaler which divides the

counting clock by factor 8. Activating the prescaler reduces the resolution of the reload counter T14. If the prescaler is not activated, the RTC may be lose counting clocks on switching from asynchronous to synchronous mode and back. This effect can be avoided by activating the prescaler.

Setting the bits T14DEC or T14INC decrements or increments the T14 timer with the next count event. If at the next count event a reload has to be executed, then an increment operation is delayed until the next count event occurs. The in/decrement function can only be used if register T14REL is not equal to $FFFF_H$. These bits are cleared by hardware after the decrement/increment operation.

13.2.3 System Clock Operation

A real time system clock can be maintained that represents the current time and date. If the RTC module is not effected by a system reset, it keeps running also during idle mode and power down mode.
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}.

13.2.4 Cyclic Interrupt Generation

RTC module can generate an interrupt request RTC_T14INT whenever timer T14 overflows and is reloaded. This interrupt request may eg. be used to provide a system time tick independent of the CPU clock frequency without loading the general purpose timers, or to wake up regularly from idle mode. The T14 overflow interrupt (RTC_T14INT) cycle time can be adjusted via the timer T14 reload register T14REL. This interrupt request is also ored with all other interrupts of the RTC via the RTC interrupt sub node RTCISN.

13.2.5 Alarm Interrupt Generation

RTC module can also provide an alarm interrupt. For an easier programming of this interrupt, the RTCL and RTCH timer can be divided into smaller reloadable timers. Each sub-timer can be programmed for an overflow on different time bases (e.g. second, hour, minute, day). With each timer overflow a RTC interrupt is generated. All these RTC interrupts are ored via the interrupt sub node RTCISNC to one interrupt request RTC_INT. Additionally the RTC_T14INT is connected to this interrupt sub node.

13.2.6 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 counts with the RTC count input frequency (RTC_REF_CLK) divided by the fixed prescaler, if the prescaler is selected. The reload registers T14REL, RTCRELL and RTCRELH should be cleared to get a 48-bit binary

timer. However, any other reload values may be used.
The maximum usable timespan is 2^{48} ($\approx 10^{14}$) T14 input clocks, which equals more than 100 years (referring to a RTC count input frequency RTC_REF_CLK of 625 KHz, which is equal to a 20 MHz Oscillator divided by 32, and an activated prescaler).

13.2.7 Defining the RTC Time Base

The count input of the RTC module (RTC_REF_CLK) is connected as shown in **Figure 93**. The RTC timer base is equal to timer T14 overflow and depends on the selectable prescaler and on the reload counter T14.

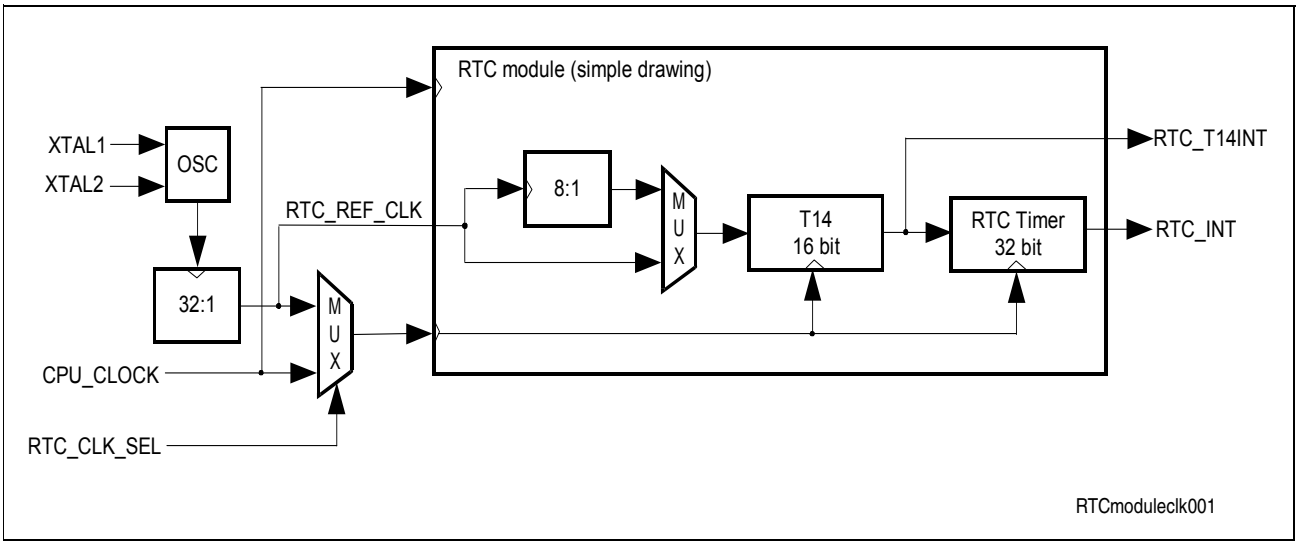


Figure 93 RTC module clocking scheme

The table below lists the RTC_T14INT interrupt period range for several RTC count input frequencies:

Table 67 RTC Interrupt Periods

Oscillator Frequency	Divider Factor	RTC Input Frequency	Prescaler Factor	RTC_T14INT Period	
				Minimum	Maximum
32 KHz	1	32 KHz		31.25 µs	2.048 s
1 MHz	32	312.5 KHz	8	256.0 µs	16.77 s
4 MHz	32	125 KHz	8	64.0 µs	4.194 s
5 MHz	32	19.531 KHz	8	51.2 µs	3.355 s
8 MHz	32	156.25 KHz	8	32.0 µs	2.097 s
10 MHz	32	312.5 KHz	8	25.6 µs	1.678 s
12 MHz	32	375 KHz	8	21.3 µs	1.398 s

Table 67 RTC Interrupt Periods (cont'd)

Oscillator Frequency	Divider Factor	RTC Input Frequency	Prescaler Factor	RTC_T14INT Period	
				Minimum	Maximum
16 MHz	32	500 KHz	8	16.0 µs	1.049 s
20 MHz	32	625 KHz	8	12.8 µs	0.839 s
24 MHz	32	750 KHz	8	10.67 µs	0.699 s
25 MHz	32	781.25 KHz	8	10.24 µs	0.671 s
32 MHz	32	1 MHz	8	8.0 µs	0.524 s
50 MHz	32	1.56 MHz	8	5.12 µs	0.336 s

Table 68 lists the T14 reload values for a time base of 1 s (A), 100 ms (B) and 1 ms (C) and several RTC input frequencies:

Table 68 RTC Reload Values

RTC Input Frequency	Reload Value A		Reload Value B		Reload Value C	
	T14REL	Base	T14REL	Base	T14REL	Base
32 KHz	8300_H	1.000 s	$F380_H$	100.0 ms	$FFE0_H$	1.000 ms
312.5 KHz	$F0BE_H$	0.999 s	$FE79_H$	100.1 ms	$FFFC_H$	1.024 ms
125 KHz	$C2F7_H$	1.000 s	$F9E5_H$	100.0 ms	$FFF0_H$	1.024 ms
19.531 KHz	$B3B5_H$	0.999 s	$F85F_H$	99.9 ms	$FFEC_H$	1.024 ms
156.25 KHz	$85EE_H$	1.000 s	$F3CB_H$	100.0 ms	$FFE1_H$	0.992 ms
312.5 KHz	6769_H	1.000 s	$F0BE_H$	99.9 ms	$FFD9_H$	0.998 ms
375 KHz	$48E5_H$	1.000 s	$EDB0_H$	100.0 ms	$FFD1_H$	1.003 ms
500 KHz	$0BDC_H$	1.000 s	$E796_H$	100.0 ms	$FFC1_H$	1.008 ms
625 KHz			$E17B_H$	100.0 ms	$FFB2_H$	0.998 ms
750 KHz			$DB61_H$	100.0 ms	$FFA2_H$	1.003 ms
781.25 KHz			$D9DA_H$	100.0 ms	$FF9E_H$	1.004 ms
1 MHz			$CF2C_H$	100.0 ms	$FF83_H$	1.000 ms
1.56 MHz			$B3B5_H$	99.9 ms	$FF3D_H$	0.998 ms

13.2.8 Increased RTC Accuracy through Software Correction

The accuracy of the C161U's RTC is determined by the oscillator frequency and by the respective prescaling factor (excluding or including T14 and the selectable Prescaler). 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, eg. 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). T14 can be adjusted by a write access or better by using the in/ decrement function provided by the RTCCON register.

Note: For the majority of applications, however, the standard accuracy provided by the RTC's structure will be more than sufficient.

13.2.9 Hardware dependend RTC Accuracy

The RTC has different counting accuracies, depending on the operating mode (with or without prescaler). There is only an impact on the counting accuracy when switching the RTC from synchronous mode to asynchronous mode and back.

.

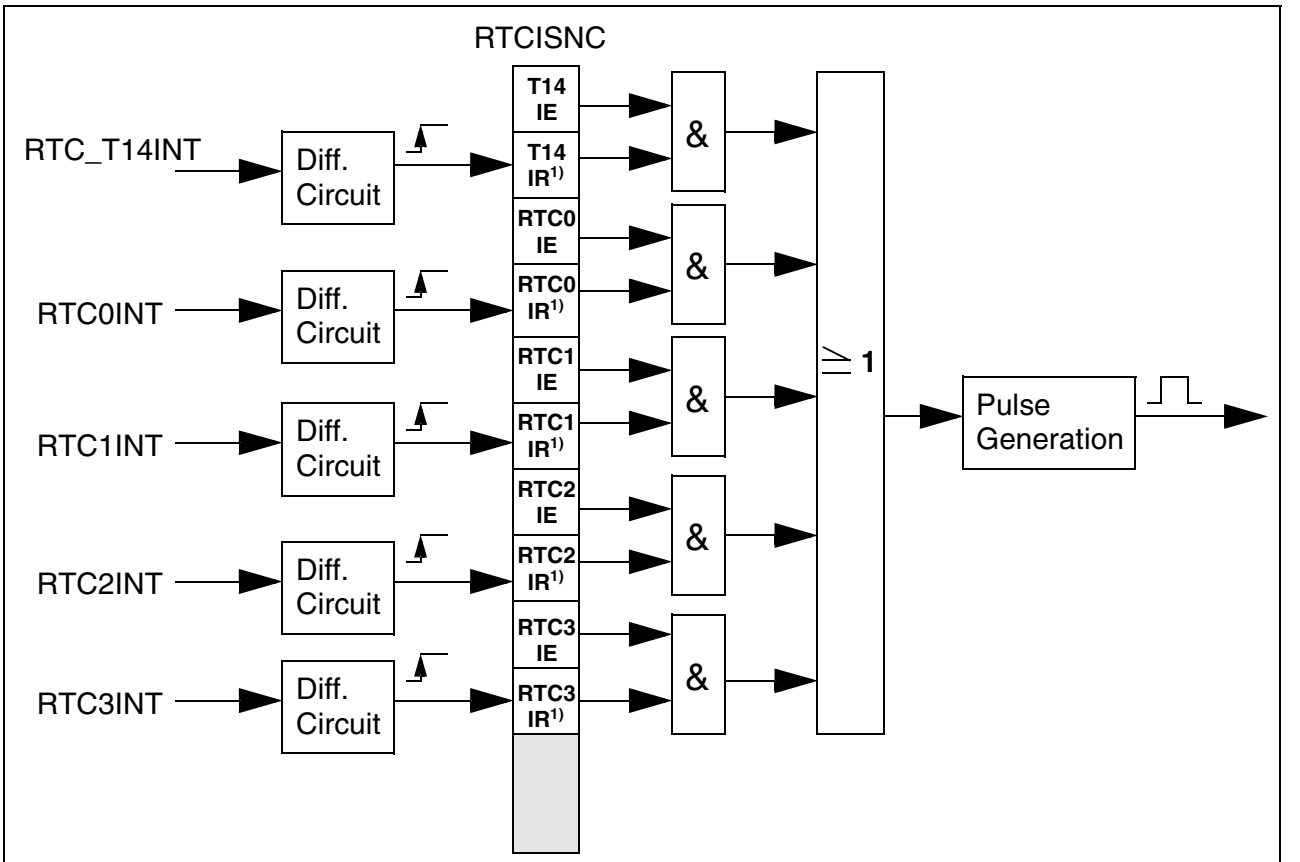
Table 69 Impact on counting accuracy

Operating mode	Inaccuracy in T14 counting ticks
without prescaler	+0 / -0.5
with prescaler	+0 / -0

13.2.10 Interrupt Sub Node RTCISNC

All RTC interrupts are connected to one interrupt node via an interrupt sub node. For this interrupt sharing each interrupt source has additionally to the node enable and request flag its own enable and request flag located in register RTCISNC. After a RTC interrupt (RTC_INT) is arbitrated, the interrupt service routine has to check the request flags of all enabled sources and run the respective software routine. The request flags have to be deleted by software before leaving the interrupt service routine.

The following figure shows the additional necessary differentiating circuits for the interrupt logic:



Note: 1)The ISR (Interrupt Service Routine) must delete the IR flag in register RTCISNC manually. Otherwise no further interrupts can be detected.

Figure 94 Differentiating Circuits for RTC interrupt

13.2.11 RTC Disable Functionality

The Peripheral Kernel of the RTC can be disabled, if the RTC functionality is not used. In this case only the bus interface is enabled. Disabling the RTC module reduces the power consumption and the generated noise of the complete system. The disable request can be set in two different ways. The bit RTCDIS (if implemented in C161U) of the central peripheral control register SYSCON3 controls the disable request of the RTC module. Additional the request can be set with bit RTCDISR inside the RTC Clock Control register (RTCCLC). The central and the local disable requests are ored. Clearing the respective disable request flag enables the RTC module again. Note that both request flags have to be cleared for enabling the RTC module. An activated request flag sets directly the disabling status flag RTCDISS inside the RTCCLC register and disables the clock within the Clock Gating Module. Since the RTCCLC register is clocked with the

bus clock, the disabling status of the RTC can be read and the RTC can be enabled again.

The following figure shows the disabling mechanism of the RTC module:

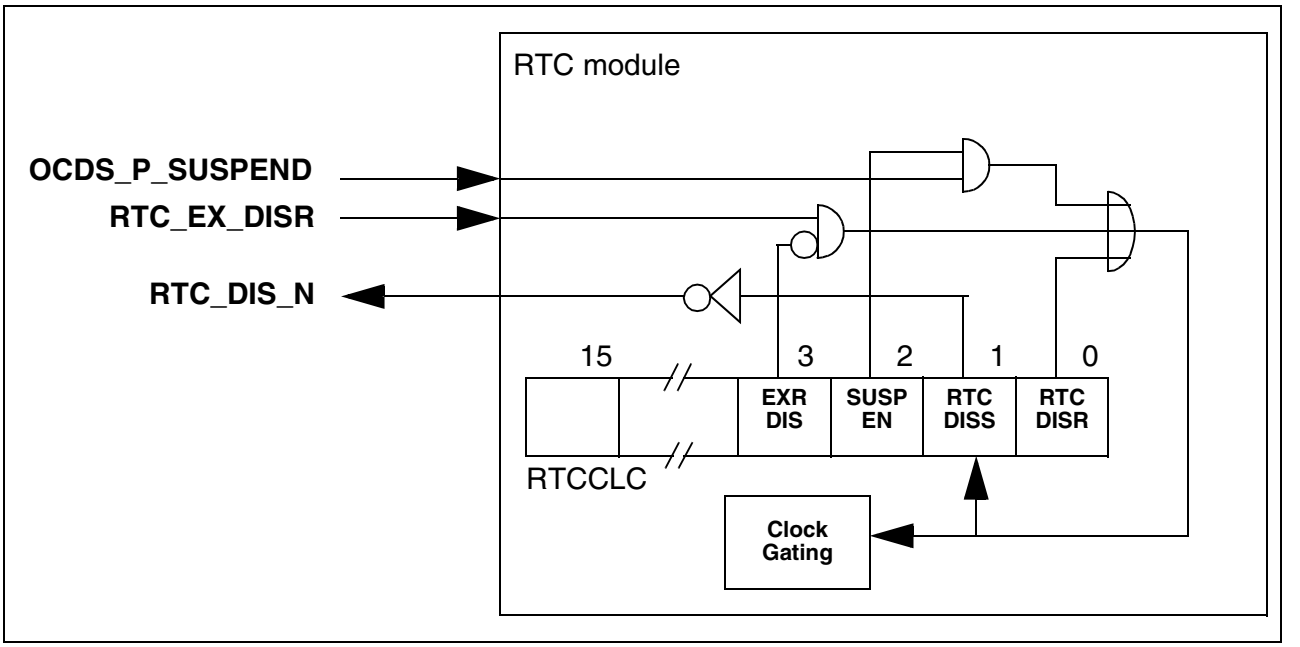


Figure 95 Disabling Mechanism of the RTC

In disable state no write access to RTC registers is possible. The only exeption is the RTCCLC register.

13.2.12 Register Definition of RTC module

The following table shows the register addresses map:

Table 70 Address Map Overview

SFR Address	b/p	Register Name
$F0C8_H$		RTCCLC
$F1CC_H$	b	RTCCON
$F0D0_H$		T14REL
$F0D2_H$		T14
$F0D4_H$		RTCL
$F0D6_H$		RTCH
$F0CC_H$		RTCRELL

SFR Address	b/p	Register Name
$F0CE_H$		RTCRELH
$F1C8_H$	b/p	RTCISNC

b: **b**itaddressable **p**: bit **p**rotected

RTC Clock Control Register

RTCCLC ($F0C8_H$) **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	EX DISR	SUS PEN	RTC DISS	RTC DISR
												rw	rw	r	rw

Bit	Function
RTCDISR	RTC Disable Request Bit RTCDISR = ‘0’: RTC clock disable not requested RTCDISR = ‘1’: RTC clock disable requested
RTCDISS	RTC Disable Status Bit RTCDISS = ‘0’: RTC clock enabled RTCDISS = ‘1’: RTC clock disabled
SUSPEN	Peripheral Suspend Enable Bit for OCDS SUSPEN = ‘0’: Peripheral suspend disabled SUSPEN = ‘1’: Peripheral suspend enabled
EXDISR	External Disable Request EXRDIS = ‘0’: External clock disable Request is enabled EXRDIS = ‘1’: External clock disable Request is disabled

RTC Control Register

RTCCON ($F1CC_H$) **bitaddressable** **Reset Value:0003_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC POS	0	0	0	0	0	0	0	0	0	0	0	T14 INC	T14 DEC	RTC PRE	RTC R
r												rw	rw	rw	rw

Bit	Function
RTCR	RTC Run Bit RTCR = ‘0’: RTC stops RTCR = ‘1’: RTC runs
RTCPRE	RTC Input Source Prescaler enable RTCPRE = ‘0’: Input Prescaler disabled RTCPRE = ‘1’: Input Prescaler enabled
T14DEC	Decrement T14 Timer Value Setting this bit to 1 effects a decrement of the T14 timer value. The bit is cleared by hardware after decrementation.
T14INC	Increment T14 Timer Value Setting this bit to 1 effects an increment of the T14 timer value. The bit is cleared by hardware after incrementation.
ACCPOS	RTC register access possible This bit indicates that a synchronous read / write access to RTC registers is possible. The Clock Control register RTCCLC can allways be accessed. ACCPOS = ‘0’: No write access is possible, only asynchronous reads. ACCPOS = ‘1’: Read / Write access is possible

Note: For compatibility reasons the bits RTCR and RTCPRE are set on asynchronous HW reset (power on reset).

Note: Prescaler Timer T14. Timer T14 generates the input clock for the RTC register and the periodic interrupt

T14 ($F0D2_H$) **Reset Value:0000_H**

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

Timer T14 Reload Register

T14REL ($F0D0_H$) **Reset Value:0000_H**

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

RTC Count Register Low Word

RTCL ($F0D4_H$) **Reset Value:0000_H**

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

RTC Count Register High Word

RTCH ($F0D6_H$) **Reset Value:0000_H**

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

RTC Reload Register Low Word

RTCRELL ($F0CC_H$) **Reset Value:0000_H**

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

RTC Reload Register High Word

RTCRELH ($F0CE_H$) **Reset Value:0000_H**

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

RTC Interrupt Sub Node Control

RTCISNC ($F1C8_H$) **bitaddressable** **Reset Value: $UUUU_H$**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	RTC 3 IR	RTC 3IE	RTC 2 IR	RTC 2IE	RTC 1 IR	RTC 1IE	RTC 0 IR	RTC 0IE	T14 IR	T14 IE
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
T14IE	T14 Overflow Interrupt Enable Control Bit ‘0’: Interrupt request is disabled ‘1’: Interrupt request is enabled
T14IR	T14 Overflow Interrupt Request Flag (bit protected) ‘0’: No request pending ‘1’: This source has raised an interrupt request
RTCxIE	RTCx Interrupt Enable Control Bit ‘0’: Interrupt request is disabled ‘1’: Interrupt request is enabled
RTCxIR	RTCx Interrupt Request Flag (bit protected) ‘0’: No request pending ‘1’: This source has raised an interrupt request

Note: The interrupt request flags of the RTC interrupt sub node have to be cleared by software inside the interrupt service routine.

14 High-Speed Synchronous Serial Interface

The High-Speed Synchronous Serial Interface SSC provides flexible high-speed serial communication between the C161U and other microcontrollers, microprocessors or external peripherals.

SSC supports full-duplex and half-duplex synchronous communication up to 18 MBaud in SSC Master Mode and 9 MBaud in SSC Slave Mode (@ 36 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 (eg. the ASC 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 (eg. EEPROMs etc.) or other controllers (networking). The SSC supports half-duplex and full-duplex communication. Data is transmitted or received on pins MTSR/P3.9 (Master Transmit / Slave Receive) and MRST/P3.8 (Master Receive / Slave Transmit). The clock signal is output or input on pin SCLK/P3.13. These pins are alternate functions of Port 3 pins.

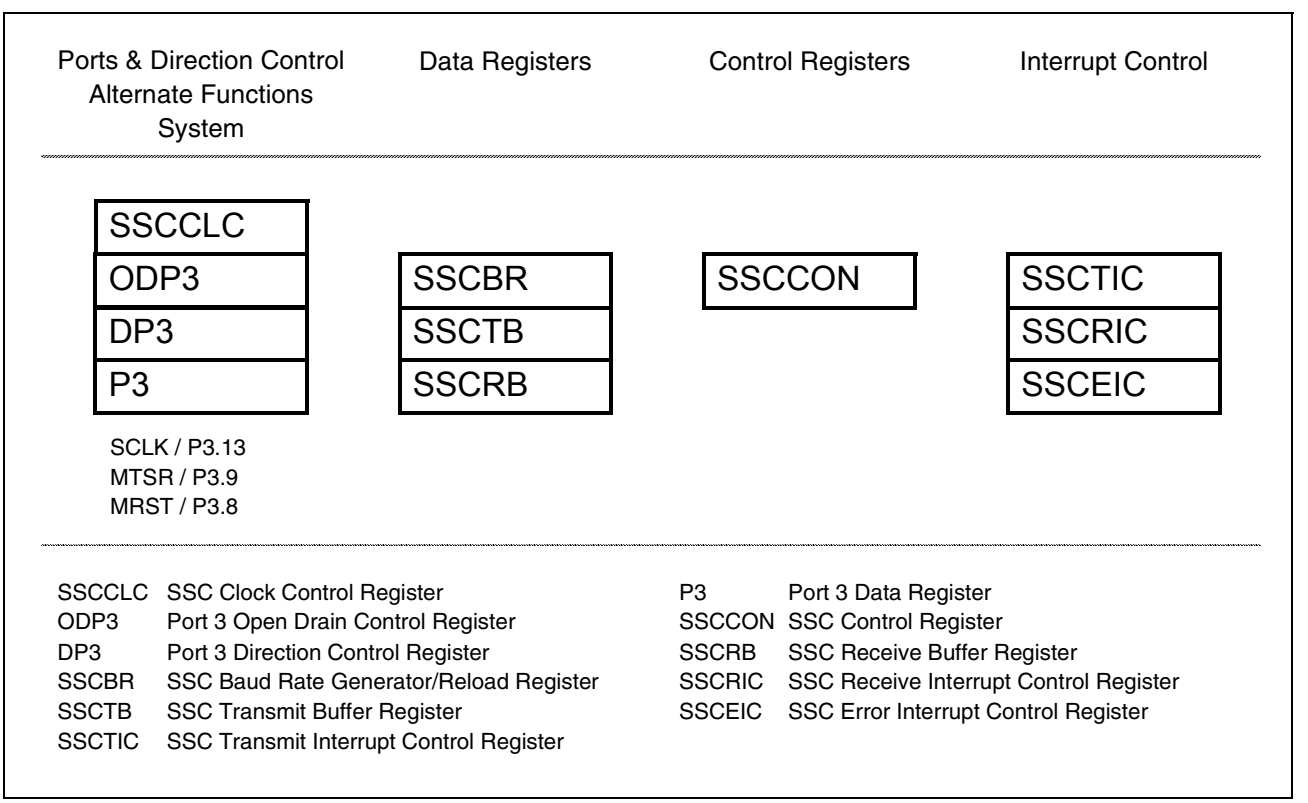


Figure 96 SFRs and Port Pins associated with the SSC

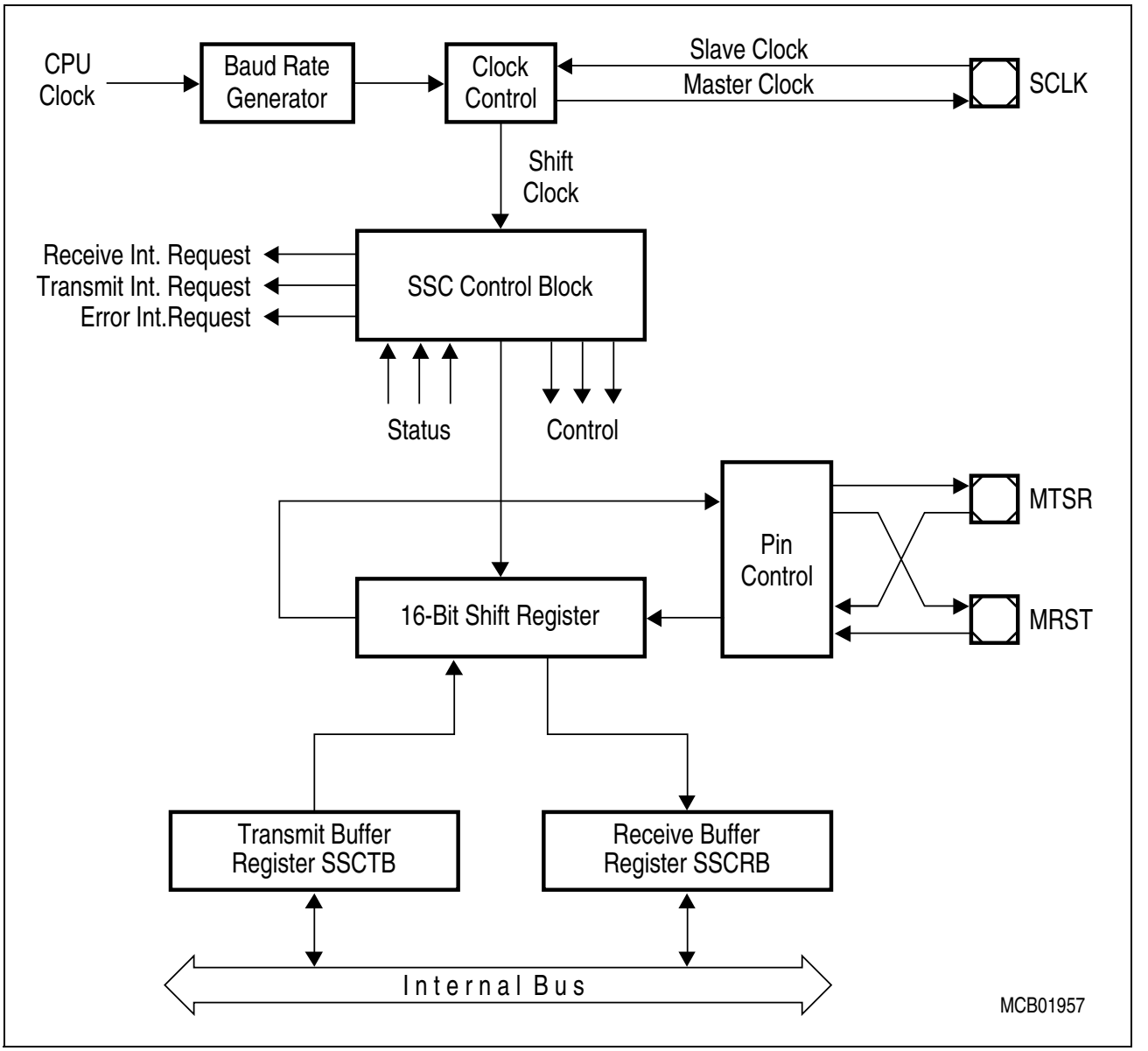


Figure 97 Synchronous Serial Channel SSC Block Diagram

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, ie. 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, ie. 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 signa
- the baudrate may be set from 274.7 Baud up to 18 MBaud (@ 36 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.

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 eg. with ASC 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.

Clock Control allows the adaptation of transmit and receive behaviour 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.

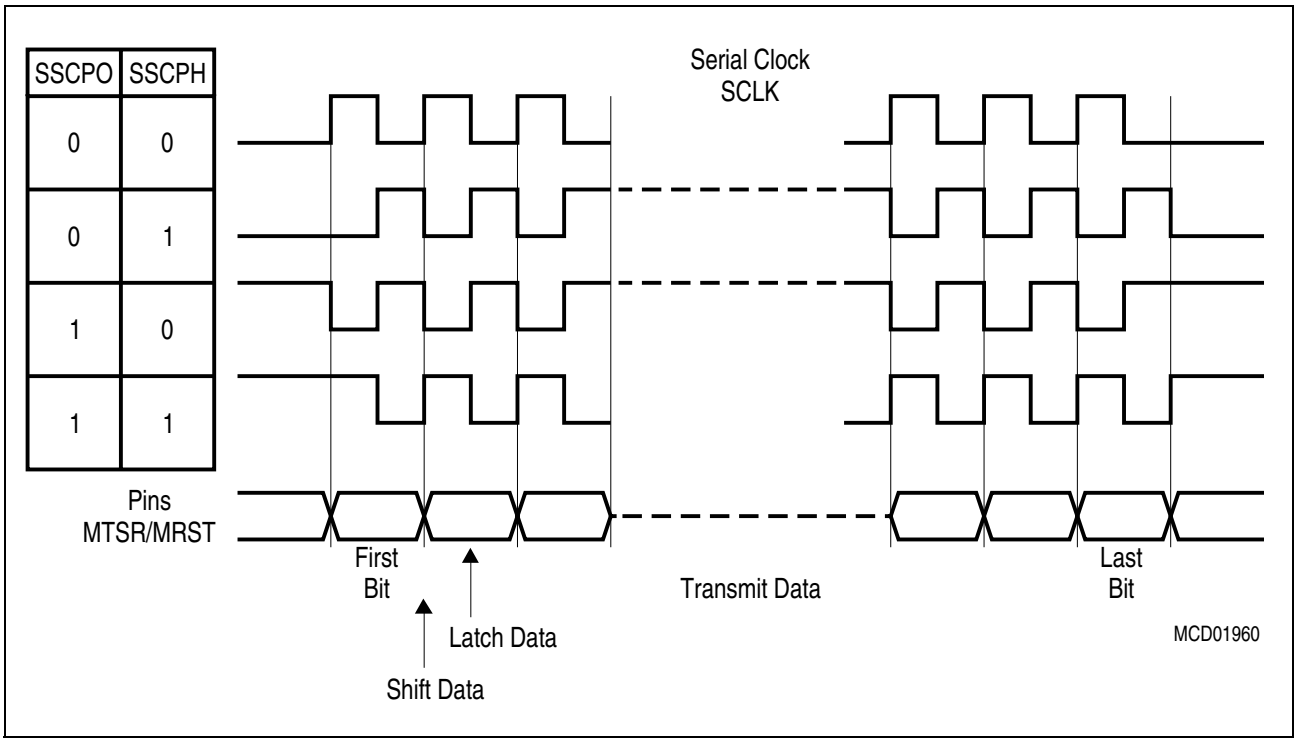


Figure 98 Serial Clock Phase and Polarity Options

14.1 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”).

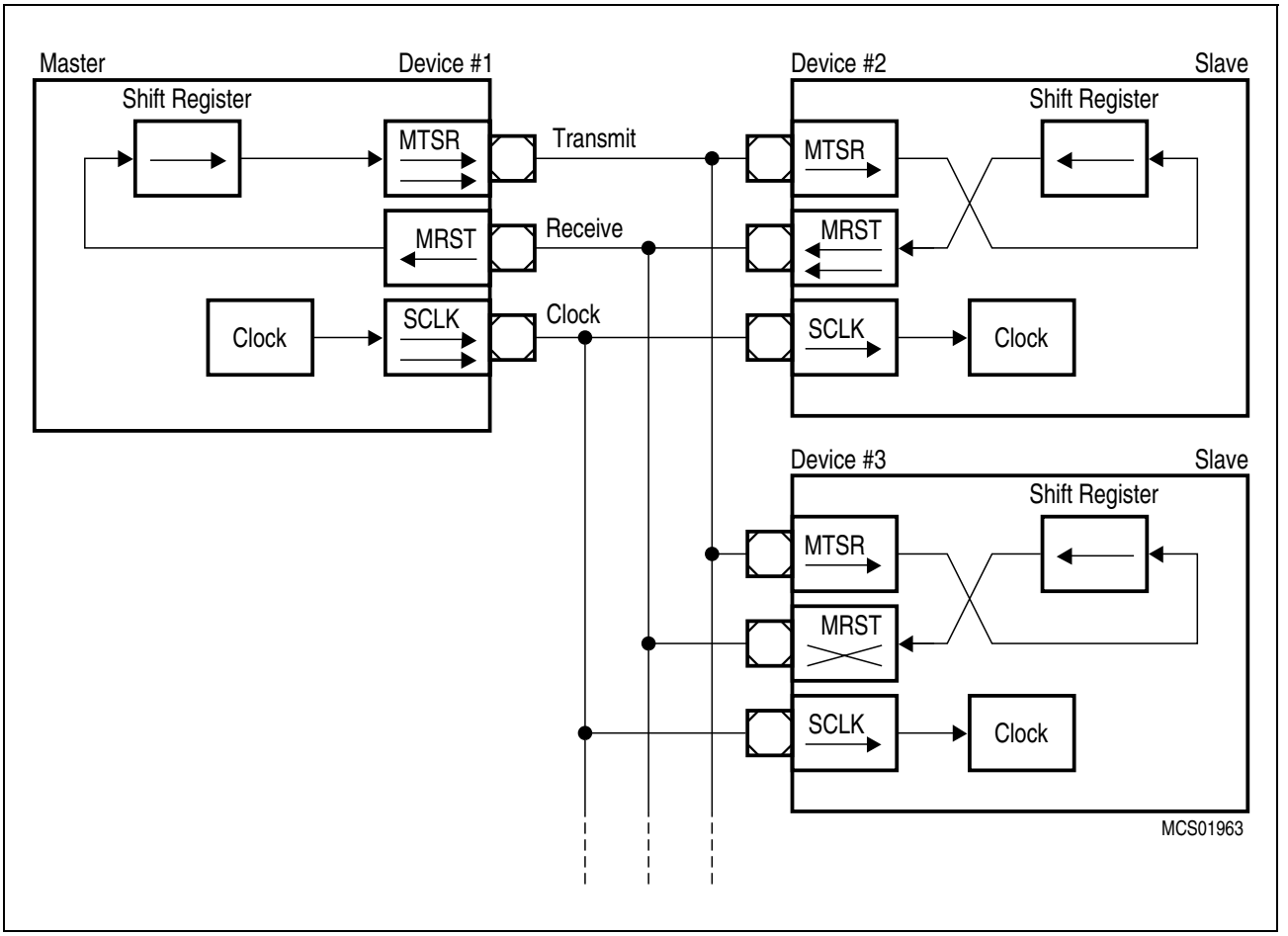


Figure 99 SSC Full Duplex Configuration

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, ie. 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.

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 eg. from asynchronous reception on ASC.

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).

14.2 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.

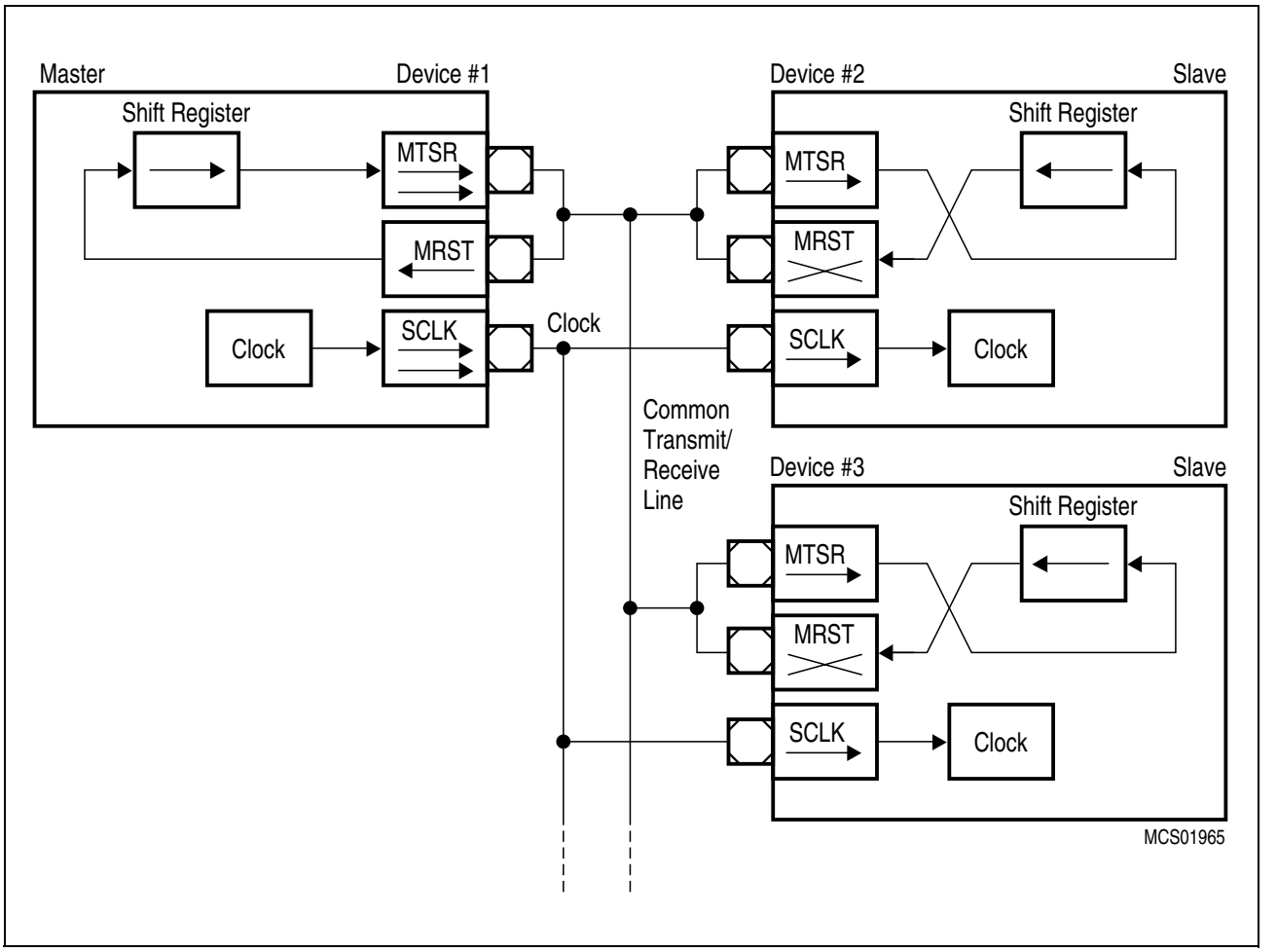


Figure 100 SSC Half Duplex Configuration

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. Eg. 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 eg. 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

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 I/O 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 I/O 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.

14.3 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:

$$B_{SSC} = \frac{f_{CPU}}{2 * (<SSCBR> + 1)} \qquad SSCBR = \left(\frac{f_{CPU}}{2 * Baudrate_{SSC}} \right) - 1$$

<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 36 MHz is 18 MBaud in SSC Master Mode (<SSCBR>= '0_d'), while in SSC Slave Mode the maximum baud rate is 9 MBaud (<SSCBR>= '1_d' since <SSCBR>='0_d' is not allowed in Slave Mode). The minimum baud rate is 274.66 Baud (<SSCBR> = '$FFFF_H$' = '65535_D').

14.4 Error Detection Mechanisms

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%, ie. 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, ie. 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.

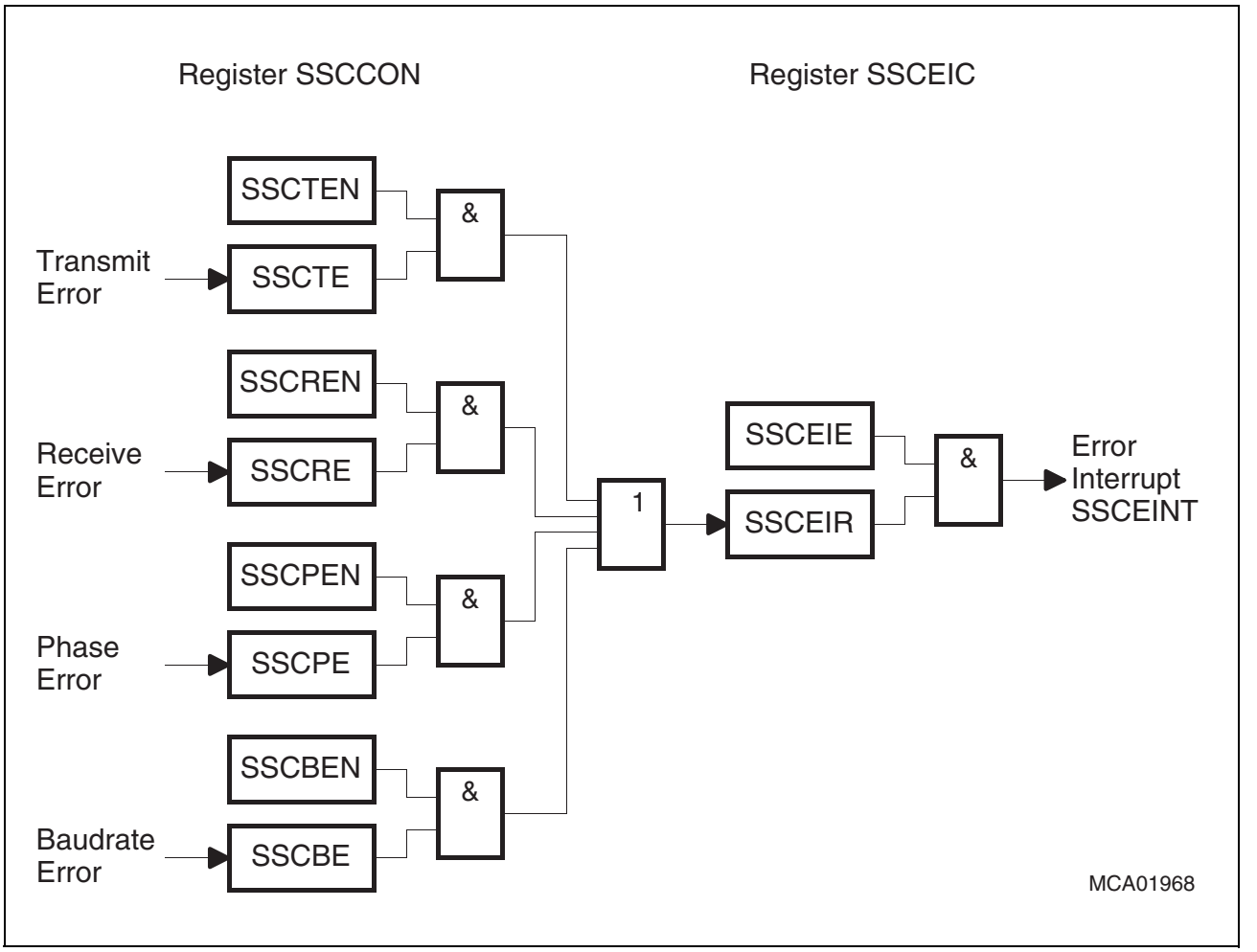


Figure 101 SSC Error Interrupt Control

14.5 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 on page 111 for an explanation of the control fields.

SSC Clock Control Register

SSCCLC ($F0B6_H$) **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	EX DISR	SUS PEN	SSC DISS	SSC DISR
												rw	rw	r	rw

Bit	Function
SSCDISR	SSC Disable Request Bit SSCDISR = ‘0’:SSC clock disable not requested SSCDISR = ‘1’:SSC clock disable requested
SSCDISS	SSC Disable Status Bit SSCDISS = ‘0’:SSC clock enabled SSCDISS = ‘1’:SSC clock disabled

Bit	Function
SUSPEN	Peripheral Suspend Enable Bit for OCDS SUSPEN = ‘0’: Peripheral suspend disabled SUSPEN = ‘1’: Peripheral suspend enabled
EXDISR	External Disable Request EXRDIS = ‘0’: External clock disable Request is enabled EXRDIS = ‘1’: External clock disable Request is disabled

15 USB Interface Controller

15.1 USB Features

All USB data transfers will be initiated by the USB host. The host is also suspending the device in order to save power and controls the remote wakeup. The device itself may resume from suspend mode. The electrical interface and the protocol is compliant with USB specification 1.1.

C161U provides eight endpoints: in addition to Endpoint Zero as the Default Control Pipe, seven Endpoints can be configured. Each of these endpoints can be of type isochronous, bulk or interrupt. Two configurations with alternate settings for multiple modes of device operation will be supported.

C161U is designed to provide support for all USB device classes, including Communication Device, Audio and HID Class.

The maximum packet length supported will be 1023 bytes.

Acknowledged (ACK) and Not Acknowledged (NACK) handshake will be generated by Hardware (HW) within the UDC, whereas each individual endpoint can be STALLED by Software (SW).

15.2 USB Protocol

USB protocol is packet based and consists of the following key elements: tokens, packets, transactions and transfers.

Tokens

Tokens will identify the type of packet in the Packet Identifier field (PID) of the token packet. Four PIDs are defined: SOF (start of frame), SETUP (for device control), IN and OUT (for control and other data).

Packets

Four types of packets are defined: SOF, token, data and handshake. Each packet begins with a SYNC byte followed by the Packet Identifier (PID).

A *SOF packet* consists of the following fields:

- SYNC byte (8 bits)
- Packet Identifier (8 bits)
- Frame Number (11 bits)
- CRC-5 (5 bits)

A *token packet* consists of the following fields:

- SYNC byte (8 bits)
- Packet Identifier (8 bits)

– Address (7 bits)
– Endpoint (4 bits)
– CRC-5 (5 bits)

A *data packet* consists of the following fields:

– SYNC byte (8 bits)
– Packet Identifier (8 bits)
– Data (n bytes)
– CRC-5 (5 bits)

A *handshake packet* consists of the following fields:

– SYNC byte (8 bits)
– Packet Identifier (8 bits)

Transaction

A transaction consists of a token packet, optional data packets and a handshake packet.

Transfer

A transfer consists of one or more transactions. Control transfers consists of a setup transaction, optional data transactions and a handshake/status transaction.

15.3 USB Endpoints

Data transactions will be handled by the UDC core as a bus master via the application bus interface. The 8-byte SETUP control data will be written into the USBD_SETUP registers only, whereas subsequent data transactions will use endpoint 0 or another endpoint specified by the SETUP packet.

In addition, the USBD FIFO/Control block provides eight (i.e. per endpoint) 8-byte Transmit FIFOs for IN transfers and eight 8-byte Receive FIFOs for OUT transfers. The Transmit FIFO is accessed by the USBD_TXWRn register and the Receive FIFO by the USBD_RXRDn register from the CPU via XBUS. Each Receive and Transmit FIFO provides a Packet-Complete indication register and interrupt USBD_TXDONEn/ USBD_RXDONEn, indicating a complete packet transfer from/to the CPU.

The basic FIFO structure is shown in **Figure 102** below. Each endpoint FIFO pair generates two transfer request interrupt to the EPEC. The USBD_TXREQn interrupt indicates that the Transmit FIFO is able to accept words from the XBUS. The USBD_RXREQn interrupt indicates an valid word in the Receive FIFO which can be read.

The USBD control logic multiplexes the active FIFO input/output on the unidirectional DEV_DATA or UDC_DATA bus.

A 10-bit receive byte counter counts the length of the currently receiving packet. This length information together with an endpoint/packet status will be provided in the length register after the packet is received completely.

For SW syncronization purposes, an 1 ms Start-of-Frame (SOF) interrupt will be generated periodically by the USB host. The frame number will be captured for each received frame.

No data available for an Interrupt, Bulk or Control endpoint for the dedicated Transmit FIFO will result in a NACK, generated by the UDC. No Receive FIFO space available will result also in a NACK. Additionally, for receive function the RX byte count register has to be read. Otherwise the next packet will be NACKed.

In transmit direction, the software has to read the STATUS bit from the USBD_STATUS_REG1 register after transferring the packet in order to determine whether the transfer was ACKed or NACKed by the host.

The SW can STALL or resume the device from suspend mode by using the command register USBD_CMD_REG.

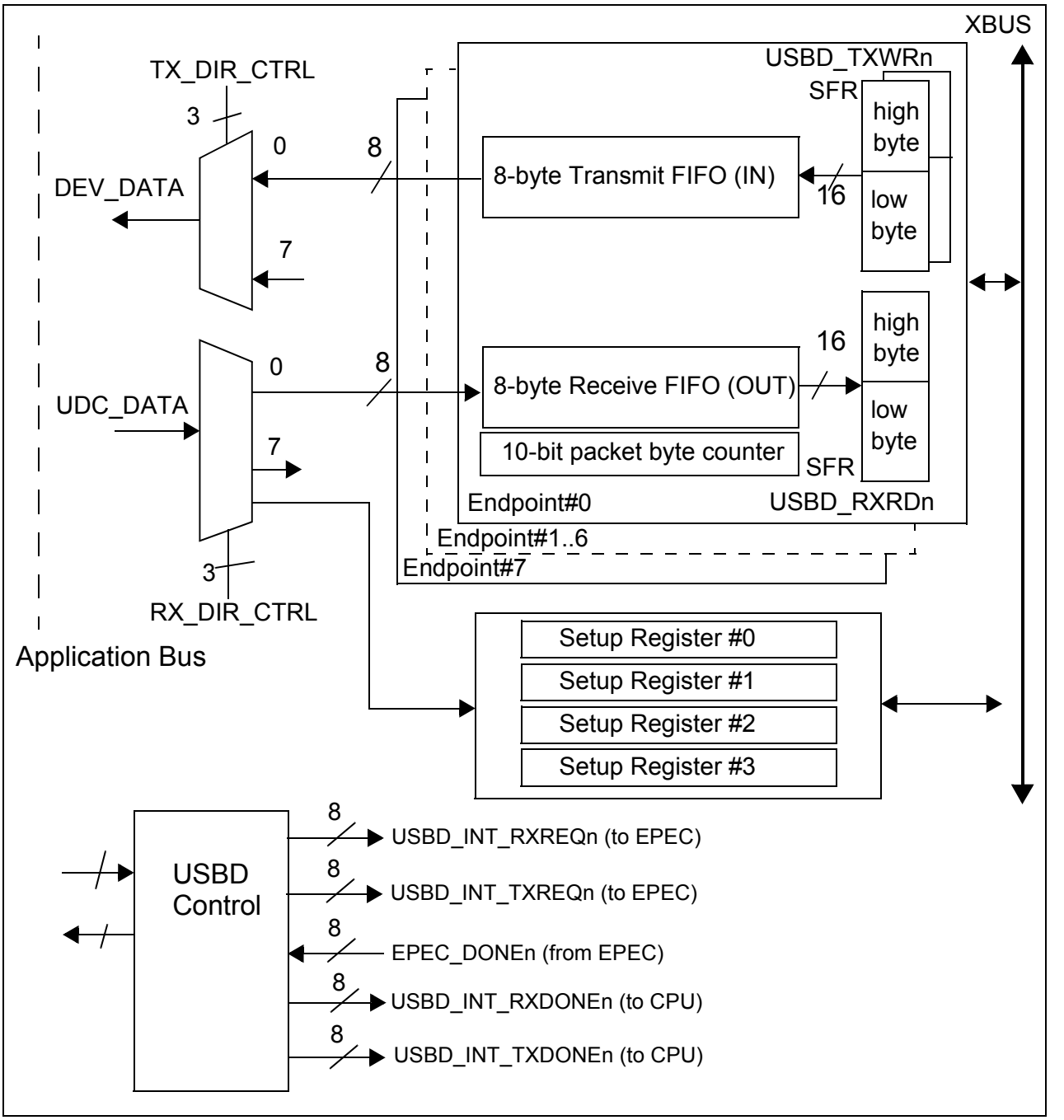


Figure 102 USBD basic FIFO/Control structure

Data requested by Transmit FIFOs (including control IN packets following SETUP packets) will be transferred from an internal/external memory location to the endpoint FIFO by the EPEC injecting MOV-instructions into the decode stage of the instruction pipeline.

Data available by Receive FIFOs (including control OUT packets following SETUP packets) will be transferred from the endpoint FIFO to an internal/external memory

location by the EPEC injecting MOV-instructions into the decode stage of the instruction pipeline.

In both directions the SW will control the EPEC providing DMA-like source and destination pointers on a per-packet basis. This provides a maximum degee of flexibility for the application SW, e.g. a linked list structure or circular FIFOs implementation with buffer pool elements can be used.

IN-Transactions (Device to Host: TX)

The host requests device data by using the IN transaction. The IN transaction mechanism handled by the device HW/SW is shown in **Figure 103** below.

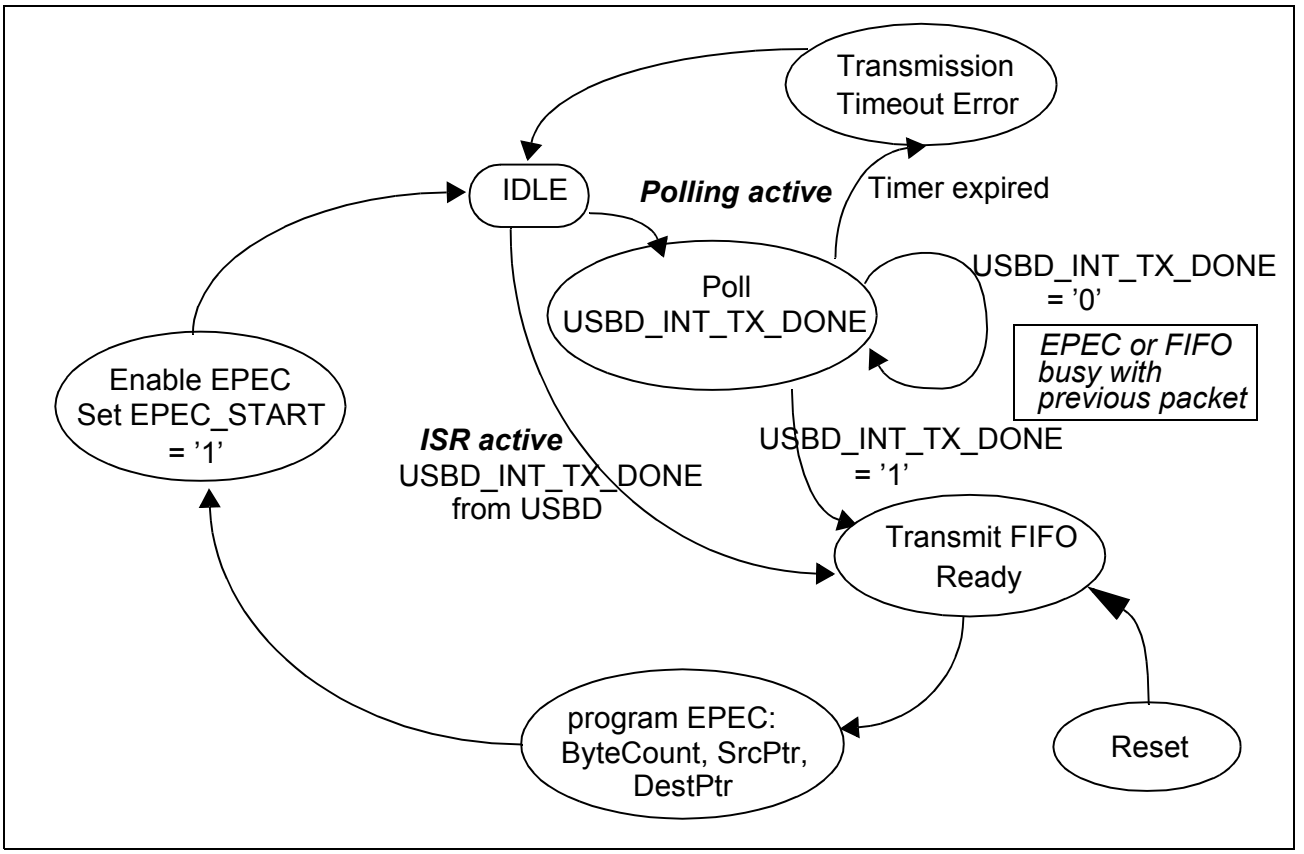


Figure 103 USB IN-Transfer controlled by SW

Since the UDC requests data handshake from the application within 1UDC clock cycle (@12MHz) the SW has to provide the next data packet to the Transmit FIFO in time, i.e. before the host is requesting data, in order to prevent the FIFO from starvation resulting in a NACK to the host. This can be achieved by either polling for the USBD_TXDONEn for bit being asserted, indicating a free FIFO or by providing the next packet as soon as the previous packet has been transmitted completely using the interrupt USBD_INT_TXDONE directly.

Zero-data-length packets can be sent from SW to the host by writing into the TxEOD register.

The device SW provides a new packet by setting up the new source and destination pointer within the EPEC. Since the packet length of the last packet is either 0 or less than the maximum defined packet length, the EPEC provides a 10-bit byte counter. After reaching its terminal count value, the EPEC stops transferring data, sets the EPEC_DONEn pulse signalizing to the USB core the end of packet and sets the EPEC interrupt. The counter will be loaded by SW for each packet to be transmitted.

The new packet transfer is started by setting the EPEC_STARTn bit in the EPEC register EPEC_CMD. This bit will enable the word transfer request USBD_INT_TXREQ generated by the Transmit FIFO.

Whether the last EPEC transfer was a word or byte transfer will be handled by the EPEC for transmit data.

By this mechanism only a minimum of SW transaction will be required which guarantees minimum latency times for the UDC handshake procedure.

OUT Transactions (Host to Device: RX)

The host transmits device data by using the OUT transaction. The OUT transaction mechanism handled by the device HW/SW is shown in **Figure 104**.

Since the UDC requests data handshake from the application within 4 UDC clock cycles (@12MHz) the SW has to provide the next empty receive buffer to the Receive FIFO in time, in order to prevent the FIFO from overflow resulting in a NACK to the host. This can be achieved by either polling for the USBD_RXDONEn bit for being asserted, indicating an empty Receive FIFO or by providing the next receive buffer as soon as the previous packet has been received completely using the interrupt USBD_INT_RXDONE directly.

The USBD provides a 10-bit byte counter USBD_RXBYTECNTn for each endpoint Receive FIFO counting the data strobe pulses asserted by the UDC. A completed packet transfer over the application bus (either XferAck or XferNack) will stop the byte counter and the counter value will be copied into the USBD_RXBYTECNTn register along with a packet status information (e.g. packet valid). As soon as the complete packet has been transferred by the EPEC, an endpoint receive interrupt USBD_INT_RXDONE will be generated.

A new packet can only be received if the RX FIFO is empty, i.e. the EPEC has transfered the last byte/word. The next packet cannot be accepted by the very same Receive FIFO until the counter value has been cleared by a CPU read access. Therefore, once the counter is read and cleared the EPEC is enabled again, the SW has to load the EPEC with a new source and destination pointers **before** reading the counter register. This will enable the word transfer request USBD_INT_RXREQ generated by the Receive FIFO and EPEC starts transferring words.

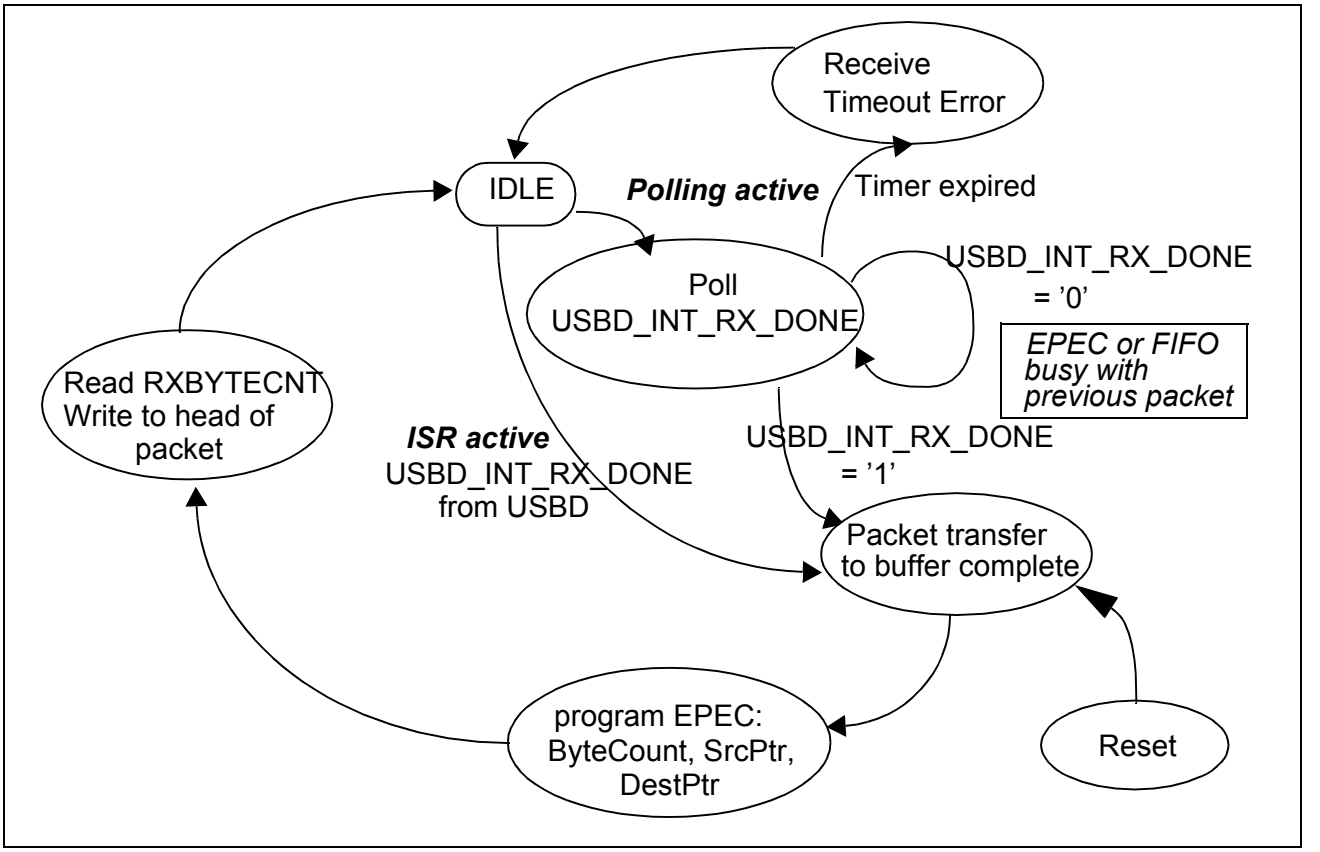


Figure 104 USB OUT-Transfer controlled by SW

Whether the last EPEC transfer was a word or byte transfer will be indicated by a USBD_RXBYTECNT value, which will be read by SW.

By this mechanism only a minimum of SW transaction will be required which guarantees minimum latency times for the UDC handshake procedure.

Control Endpoint 0

The default control pipe is reserved to Endpoint 0. The host communicates over the control endpoint in order to retrieve device information, device configurations, interfaces, alternate settings and other device endpoint characteristics. Control endpoints are bidirectional and the data delivery will be guaranteed.

The hosts starts with an 8-bytes SETUP packet, which will be written to the SETUP register bank USBD_Setup(3:0). A SETUP interrupt request will inform the CPU in order to process the SETUP packet if the packet was received without any errors. A pending/ processing SETUP request, which will be overwritten by the host, has to be retired by the device SW.

In the bidirectional data stage, optional data packets can be read/written by the host through the 8-bytes bidirectional Endpoint 0 FIFO.

The following handshake/status stage will be generated by the UDC core itself. The application interface has to check for the zero length data IN/OUT token in reverse direction of the actual transfer sent by the host at the end of the data stage. I.e. the control read (IN token) will be terminated by an OUT token with zero length data transfer and the control write by an IN token.

The status stage will be NACK'd by the device until the SW has processed the request. A STALL will indicate to the host that the request cannot be completed successfully by the device.

Isochronous, Bulk and Interrupt Endpoints

All these types of transfers are handled by the FIFO/EPEC mechanism described for IN and OUT transactions above.

Standard Device Requests

Most of the standard device requests will be handled by the UDC internally, including SETUP stage, optional DATA stage and STATUS stage. Only GET_DESCRIPTOR, SET_DESCRIPTOR, SYNC_FRAME will be forwarded to the application bus and handled by SW.

The 8-byte SETUP request will be captured in the SETUP registers. IN/OUT transactions from the host will be NACK'd until the SW is ready. The SETUP registers cannot be overwritten by the host until all 4 registers have been read.

The USB specification allows an early termination of control transfers. For control requests with data transmission from device to host, this means that host can send a status out packet even thought it had requested more bytes than actually were sent during this transfer. If Software has already set up the next packet in the FIFO's, this data must not be transfered with the next control in request. The data can be flushed by Software after receiving the empty status out packet or automatically this can be done by setting the Auto Flash Enable of the CMD register.

Requests to device, interface or endpoints with no DATA stage have to be successfully completed (incl. STATUS stage) within 2 ms. Requests with DATA stage require the first data packet within 10 ms and all subsequent packets within 5 ms. The STATUS stage must then be completed within 2 ms.

The UDC performs a zero data transfer write/read transaction on the application bus. The application itself, i.e. USBD control has to identify this STATUS stage of the Control Read/Write by a change of read/write direction and has to respond to this transaction by proper handshake.

Vendor/Class Requests

All Vendor/Class requests will be forwarded to the application bus and handled by SW.

The 8-byte SETUP request will be captured in the SETUP registers. IN/OUT transactions from the host will be NACK'd until the SW is ready. The SETUP registers cannot be overwritten by the host until all 4 registers have been read.

Requests to device, interface or endpoints with no DATA stage have to be successfully completed (incl. STATUS stage) within 2 ms. Requests with DATA stage require the first data packet within 10 ms and all subsequent packets within 5 ms. The STATUS stage must then be completed within 2 ms.

The UDC performs a zero data transfer write/read transaction on the application bus. The application itself, i.e. USBD control has to identify this STATUS stage of the Control Read/Write by a change of read/write direction and has to respond to this transaction by proper handshake.

Load Configuration Data

After power-up the UDC has to be loaded by the SW via control endpoint#0 Transmit FIFO. The only buffers the UDC maintains will be the 17 EndPtBufs one for each physical endpoint. The number of bytes written by the SW then equals 5x17, i.e. 85 bytes.

The first strobe will load the first byte into the MSB byte of EndPtBuf0(39:32) of endpoint#0 and so on.

Latency Considerations

The latency time is considered to be the minimum/maximum accumulated time between two EPEC transfers. Since the EPEC transfer is injected into the decode pipeline, the latency time is determined by the previous commands which have already entered the FETCH and DECODE stage of the pipeline.

USB data rate for a packet to be processed by the application is 12 Mbit/s. Since the UDC samples 8 bits before the byte is transferred over the application bus, the FIFOs are either read or written every 1.333 µs to provide a full word access to/from the XBUS interface. The first EPEC transfer at the start of a packet transfer over the XBUS, i.e. its initial latency time, may be delayed by up to 4 * 1.333 µs (i.e. 5.33 µs) since each FIFO is 8-byte deep.

Worst case: The worst case EPEC 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 EPEC response time is the time to perform 7 word bus accesses.

with f_{CPU} @24 MHz (TCL=21 ns) and 1 external waitstate:

= 7 * ext. bus accesses with 1 waitstate + 2 * states + 1 * ext. access for src/dest. pointer

= 7 * (4 * TCL + 2 * TCL) + 2 * (2* TCL) + (4 * TCL + 2 * TCL)

= 52 * TCL = 1.092 µs

with f_{CPU} @36 MHz (TCL=13.89 ns) and 1 external waitstate:

= 7 * ext. bus accesses with 1 waitstate + 2 * states + 1 * ext. access for src/dest. pointer

= 7 * (4 * TCL + 2 * TCL) + 2 * (2 * TCL) + (4 * TCL + 2 * TCL)

= 52 * TCL = 722.3 ns

Best case: 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 EPEC response time is the time to execute 1 word bus access plus 2 state times (2*TCL).

with f_{CPU} @24 MHz (TCL=21ns) and 1 external waitstate:

= 1* ext. bus accesses with 1 waitstate + 2 * states

= 1* (4 * TCL + 2 * TCL) + 2 * (2* TCL)

= 10 * TCL = 210 ns

with f_{CPU} @36 MHz (TCL=13.89 ns) and 1 external waitstate:

= 1* ext. bus accesses with 1 waitstate + 2 * states

= 1* (4 * TCL + 2 * TCL) + 2 * (2 * TCL)

= 10* TCL = 138.9 ns

Once a request for EPEC 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 RAM or external memory and to write the destination over the external bus in an external program environment.

A bus access in this context also includes delays by an external READY signal or by bus arbitration (HOLD mode).

Suspend and Host Resume Support

UDC has built in counters, to count 6 ms idle time on the USB, which detects a Suspend and 3 ms counter to send the RemoteWakeup sequence to the host. The UDC_Suspend_Set interrupt will gate the indicate the Suspend mode to the application. In response to this signal Software is supposed to turn off the clock of all the modules which are not used at that moment in order to support a low power consumption. The USB interface module clock can be switched off by SW using the USBCLC register.

The RemoteWakeup detection will deassert the UDC_Suspend for at least 20 ms, which will generate a second interrupt UDC_Suspend_Off. The application will restart the UDC and XBus clock supply as soon as possible.

Note: The normal suspend_off interrupt can not be generated if the USB clock is switched off. In this case, the fast external interrupt EX5INT (firq(5)) must be used instead.

Suspend and Remote Resume Support

After entering the Suspend state, the UDC may start a RemoteWakeup sequence on the USB, after the device has enabled the UDC clock supply. After the PLL VCO frequency has been stabilized the DEV_Resume signal can be asserted which causes the UDC to drive a resume (K-state) to the USB for 3 ms.

UDC provides a DEVICE_REMOTE_WAKEUP feature bit in the EndpointInfo block of the UDC core. At power-on reset this bit is disabled and on a SetFeature command to the device, this bit can be set by the host. The state of this bit is reflected on the STATUS2 register.

15.4 USB Interface Controller (USBD) Architecture

USB Interface Controller architecture is shown in **Figure 105** below. UDC provides the 48 MHz PLL for full-speed clock & data recovery, Serial Interface Engine (SIE) and the USB Bridge Layer Block. The UDC is connected with its Application Bus Interface to the Standard Module Interface (SMIF), which itself is hooked onto the BUS Peripheral interface (BPI).

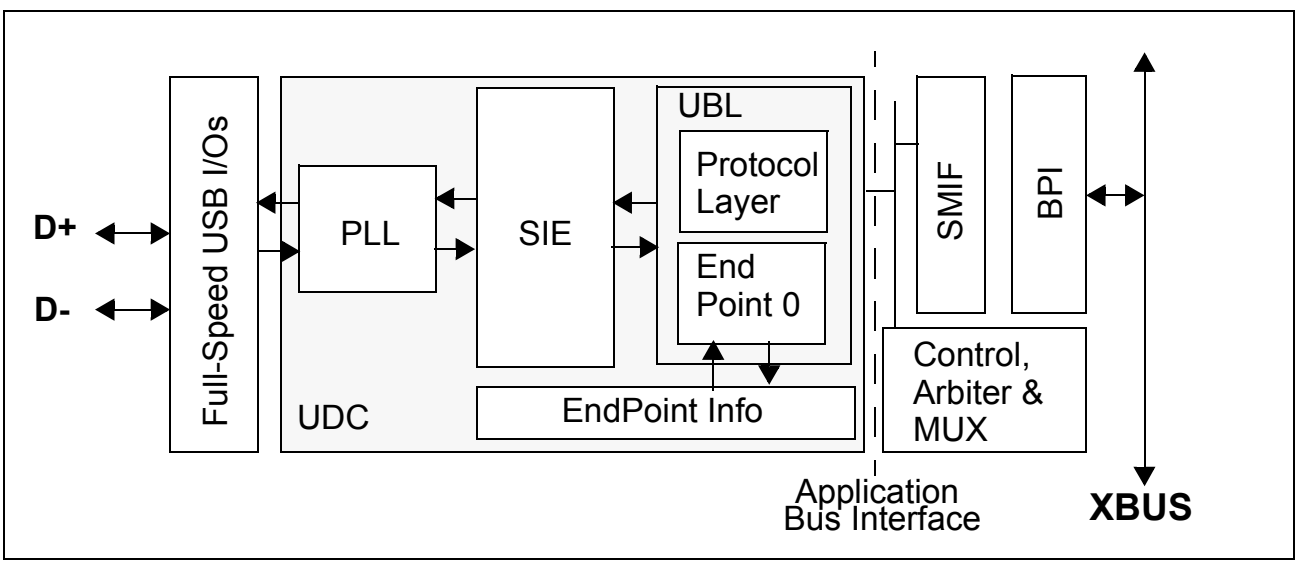


Figure 105 USB Interface Controller Architecture

15.5 Endpoint Info Block

EPINFO Block is a configurable block which will be programmed at compile time . Since the GetDescriptor command is forwarded to the Application, the EPINFO provides only the registers used by the Application to download application specific configuration data. The EPINFO block features in summary are:

- DATA0/DATA1 Synchronization
- EndPtStalled Bit support

– Two different Configurations, with
  - Configuration #1 with 4 Interfaces (three interfaces with 3 alternate settings, one interface with 2 alternate settings) and each interface with maximal seven unique endpoints where each endpoint can be assigned to one interface only.
  - Configuration #2 with 2 Interfaces, each Interface with 2 alternate settings and 2 Endpoints for each setting
– Address pointer and length support for 2 strings
– EndPtBuf information for each of the 6 currently active non-Control Endpoints

All these registers are loaded under SW control as part of the Configuration Process at Power-On time.

Examples for Configurations, Alternate Settings and Endpoints supported are shown below.

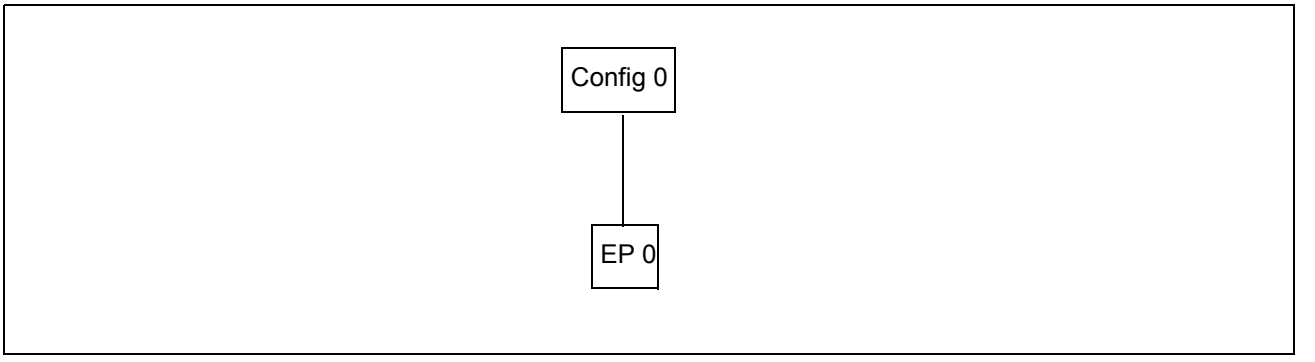


Figure 106 USB Configuration0 with Control Endpoint 0 supported.

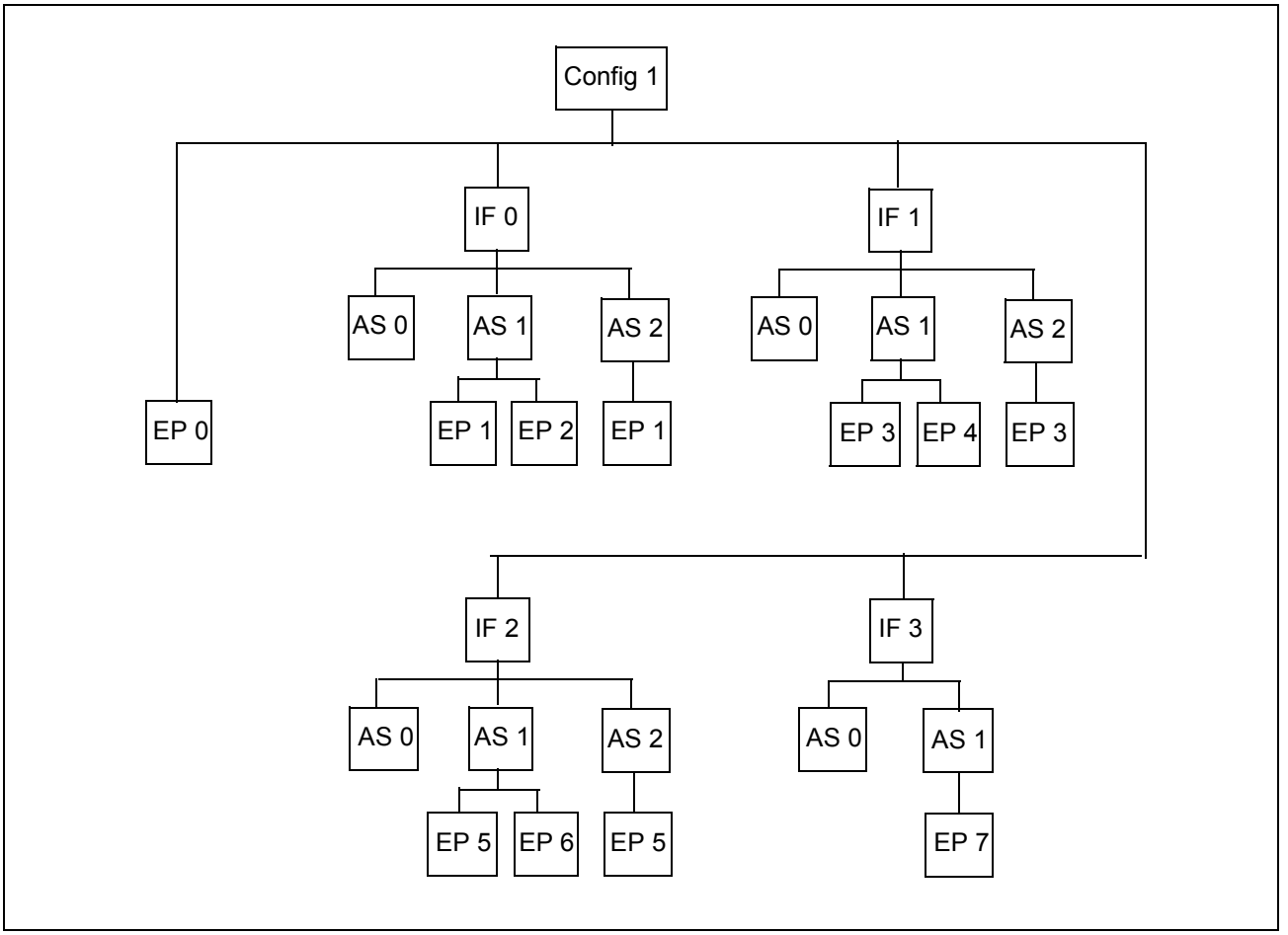


Figure 107 Example for USB Configuration1 with Alternate Settings and Endpoints

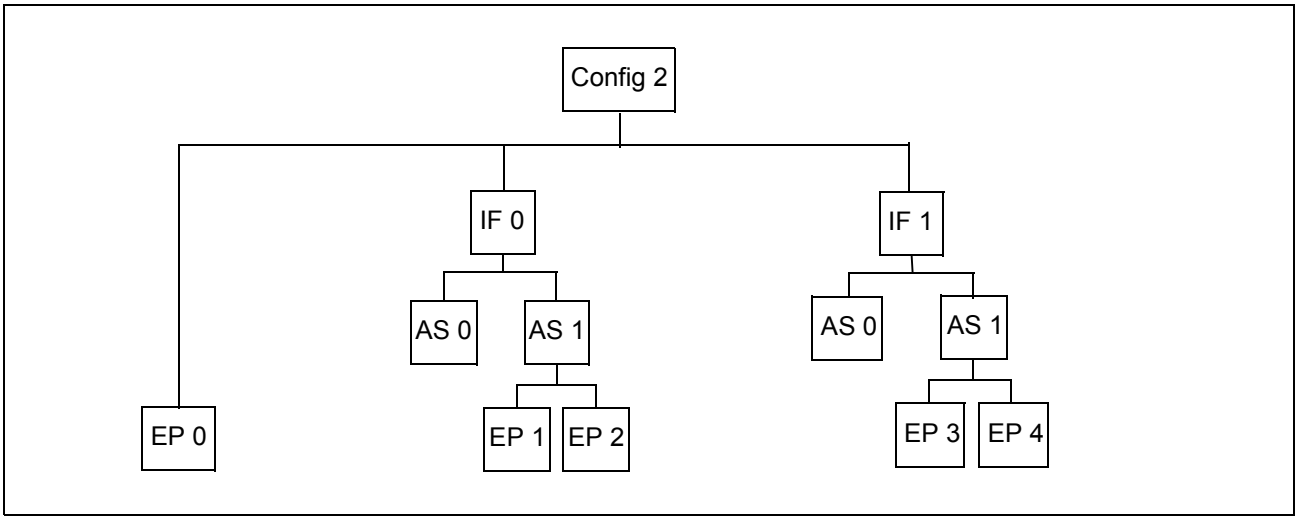


Figure 108 Example for USB Configuration2 with Alternate Settings and Endpoints

EndPtBuf's

The endpoint configuration data file structure shown in the figures above, which will be downloaded into the USB endpoint info block is summarized in **Table 71**.

Table 71 Endpoint, Configuration and Interface Structure

PhyEP	EpNum	Ep_Config	Ep_Interface	Ep_AltSetting	Ep_Type	Ep_Dir	MaxPktSize
1	0	0	0	0	Control	I/O	64 bytes
2	1	1	0	1	**	I/O	1023 bytes
3	2	1	0	1	**	I/O	1023 bytes
4	1	1	0	2	**	I/O	1023 bytes
5	3	1	1	1	**	I/O	1023 bytes
6	4	1	1	1	**	I/O	1023 bytes
7	3	1	1	2	**	I/O	1023 bytes
8	5	1	2	1	**	I/O	1023 bytes
9	6	1	2	1	**	I/O	1023 bytes
10	5	1	2	2	**	I/O	1023 bytes
11	7	1	3	1	**	I/O	1023 bytes
12	0	2	0	1	**	I/O	1023 bytes
13	1	2	0	1	**	I/O	1023 bytes
14	2	2	0	2	**	I/O	1023 bytes
15	1	2	1	1	**	I/O	1023 bytes
16	3	2	1	1	**	I/O	1023 bytes
17	4	2	1	2	**	I/O	1023 bytes
18	3	2	1	2	**	I/O	1023 bytes

The EndPtBuf's format is shown in **Table 72.**.

Table 72 EndPtBuf Format

Bit Field	Type	Description
(39:36)	EpNum	Logical Endpoint number (Endpoint number on host side)
(35:34)	Ep_Config	Configuration number related to this Endpoint
(33:32)	Ep_Interface	Interface number related to this Endpoint
(31:30)	Ep_AltSetting	Alternate Setting related to this Endpoint

Table 72 EndPtBuf Format (cont'd)

Bit Field	Type	Description
(29:28)	Ep_Type	Type of EndPoint '00' - Control '01' - Isochronous '10' - Bulk '11' - Interrupt
(27)	Ep_Dir	Direction of Endpoint (O: out, I: in)
(26:17)	Ep_MaxPktSize	Maximum packet size for this endpoint in bytes
(16)	reserved	Reserved.
(15:13)	Ep_BufAdrPtr	Reflects as UDC_BufAdrPtr to the Application Bus and indicates the endpoint number.
(12:0)	reserved	Reserved. Must be set to '0'.

ConfigBuf's

Since the UDC is forwarding the GetDescriptor Command to the Application, the UDC does not maintain Configuration Buffer space within the UDC core.

StringBuf's

Since the UDC is forwarding the GetDescriptor Command to the Application, the UDC does not maintain String Buffer space within the UDC core.

15.6 USB Microprocessor Registers

USB register set provides Special Function Registers (SFR's) for eight receiving Endpoints and eight transmitting Endpoints. In addition, Interrupt and Status SFRs will be provided for fast data processing. Each individual endpoint's Transmit and Receive FIFO can be accessed via XBUS. Transmit and Receive FIFO interrupts will be generated for each endpoint.

Note: Please note, that the SETUPxx and RXRR registers are one time read-only registers. Multiple read operations without the approriate handshake request will provide unconsistent data.

The USBD register set is shown in **Table 73.** The Base Address is $00EE00_H$.

Table 73 USBD Register Set

$00EE00_H$ + ...	**Register**	**Function**
00_H	USBCLC	USB clock control register.
08_H	USBD_ID	USB peripheral identification register, set to ZERO in the current version.
10_H	USBD_CMD_REG	Command register
12_H	USBD_STATUS_REG0	USB endpoint FIFO status
14_H	USBD_STATUS_REG1	USB endpoint FIFO handshake control
16_H	USBD_STATUS_REG2	USB Device Remote Wake-Up Feature Status
18_H	reserved	
$1A_H$	reserved	
$1C_H$	reserved	
$1E_H$	reserved	
20_H	reserved	
22_H	reserved	
24_H	USBD_TIME_REG	USB timestamp info: Frame number of the transmitted frame
26_H	USBD_SETUP_REG01	USB setup bytes (1:0)
28_H	USBD_SETUP_REG23	USB setup bytes (3:2)
$2A_H$	USBD_SETUP_REG45	USB setup bytes (5:4)
$2C_H$	USBD_SETUP_REG67	USB setup bytes (7:6)
$2E_H$	USBD_TXWR0	USB Transmit FIFO data register
30_H	USBD_TXEOD0	EPEC/SW End-of-packet indication for USBD.
32_H	USBD_RXRR0	USB Receive FIFO data register
34_H	USBD_RX_BYTECNT0	USB receive packet length in bytes
36_H	USBD_TXWR1	USB Transmit FIFO data register
38_H	USBD_TXEOD1	EPEC/SW End-of-packet indication for USBD.
$3A_H$	USBD_RXRR1	USB Receive FIFO data register
$3C_H$	USBD_RX_BYTECNT1	USB receive packet length in bytes
$3E_H$	USBD_TXWR2	USB Transmit FIFO data register

Table 73 USBD Register Set (cont'd)

$00EE00_H$ + ...	Register	Function
40_H	USBD_TXEOD2	EPEC/SW End-of-packet indication for USBD.
42_H	USBD_RXRR2	USB Receive FIFO data register
44_H	USBD_RX_BYTECNT2	USB receive packet length in bytes
46_H	USBD_TXWR3	USB Transmit FIFO data register
48_H	USBD_TXEOD3	EPEC/SW End-of-packet indication for USBD.
$4A_H$	USBD_RXRR3	USB Receive FIFO data register
$4C_H$	USBD_RX_BYTECNT3	USB receive packet length in bytes
$4E_H$	USBD_TXWR4	USB Transmit FIFO data register
50_H	USBD_TXEOD4	EPEC/SW End-of-packet indication for USBD.
52_H	USBD_RXRR4	USB Receive FIFO data register
54_H	USBD_RX_BYTECNT4	USB receive packet length in bytes
56_H	USBD_TXWR5	USB Transmit FIFO data register
58_H	USBD_TXEOD5	EPEC/SW End-of-packet indication for USBD.
$5A_H$	USBD_RXRR5	USB Receive FIFO data register
$5C_H$	USBD_RX_BYTECNT5	USB receive packet length in bytes
$5E_H$	USBD_TXWR6	USB Transmit FIFO data register
60_H	USBD_TXEOD6	EPEC/SW End-of-packet indication for USBD.
62_H	USBD_RXRR6	USB Receive FIFO data register
64_H	USBD_RX_BYTECNT6	USB receive packet length in bytes
66_H	USBD_TXWR7	USB Transmit FIFO data register
68_H	USBD_TXEOD7	EPEC/SW End-of-packet indication for USBD.
$6A_H$	USBD_RXRR7	USB Receive FIFO data register
$6C_H$	USBD_RX_BYTECNT7	USB receive packet length in bytes
$6E_H$	USBD_CFGVAL	Current Configuration & Alternate Setting selected by Host
70_H	USBC_CMD_RESET	USB Block Reset

The detailed register description is shown below.

USBCLC Reset Value: 0000_H

Table 74 USBCLC Register

Bit No.	Bit	Function
(15:4)		reserved. Always ’0’
3	USBEX_DIS	USB Controller Clock Disable 0: The kernel clock of the USB interface controller is enabled, normal operation. 1: The kernel clock of the USB interface controller is disabled.
2	USBGPSEN	USB Controller Clock OCDS Disable 0: The kernel clock of the USB interface controller is enabled, normal operation. 1: The kernel clock of the USB interface controller is disabled during debugging mode (OCDS)
1	USBDIS	USB Controller Clock Status 0: The status of the USB interface controller kernel clock is ’enabled’. 1: The status of the USB interface controller clock is ’disabled’.
0	USBDISR	USB Controller Clock Disable 0: The kernel clock of the USB interface controller is enabled, normal operation. 1: The kernel clock of the USB interface controller is disabled.

The state of the USB interface kernel clock is controlled by the register bit USBDISR.

The actual USB kernel clock state will be shown by the state bit USBDIS.

The register USBCLC is clocked with the bus clock to be able to switch the USB interface controller clock on again, if it was off. If required, switching off the clock can be prevented by the USB controller.

For on chip debugging support (OCDS) an additional bit USBGPSEN is introduced to stop the peripheral clock for arbitrary lengths of time during debugging if this function is enabled. If debugging mode is active, the peripheral core rejects write access to registers connected to the peripheral clock.

To be compatible with previous C16x products an USBEX_DISR signal is provided to disable the peripheral clock.

USBD_CMD_REG Reset Value: 0000_H

Table 75 USBD_CMD_REG Command Register

Bit No.	Bit	Function
15	Autoflush_Enable	Enable HW controlled flushing of TX channel, if IN transfer is not completed by host, i.e. if a SETUP transfer is immediately followed by a Status-OUT transfer. This also clears EPEC channel 0. (For Control endpoint 0 only. Will not be reset by HW). The Autoflush_Enable feature can be used @ 36 MHz CPU clock only and is NOT supported @ 24 MHz CPU clock.
14:12	Flush_TX_Channel_Select	Select the channel to be flushed 'xxx': binary coded 7 downto 0
11	Flush_TX _Channel	Flush the TX channel, i.e. clear FIFO and control state machine 1: Flush Command active, will be reset by HW 0: no action
10	UDC_Suspend	1: Device is in Suspend Mode 0: Normal Mode: device has initiated Device Resume proc. (read-only)
9	DEV_Resume	1: Start RemoteWakeup procedure 0: normal operation
8	USB_TXProtect	1: Enables protection feature of the USB transceiver in case of short circuit between DPLS/DMNS and VDDU/VSSU 0: short circuit protection disabled
(7:0)	STALL_EP(7:0)	1: Set Endpoint N (N=7..0) to STALLED 0: normal operation

The USBD command register provides set STALL capability per Endpoint, RemoteWakeup, Resume and Suspend control for the device.

USBD_STATUS_REG0 Reset Value: 0000_H

Table 76 USBD_STATUS_REG0 Status Register

Bit No.	Bit	Function
15:8	RX_EMPTY(7:0)	1: Indicates RX Fifo Endpoint N (N=7..0) is empty 0: not empty The reset value is ’0’, but after reset the value changes immediately.
7:0	TX_FULL(7:0)	1: Indicates TX Fifo Endpoint N (N=7..0) is full 0: normal operation

The USBD status register 0 provides the SW with status indication for RX and TX FIFOs and provides handshake control for both FIFOs. This register is read-only.

USBD_STATUS_REG1 Reset Value: 0000_H

Table 77 USBD_STATUS_REG1 Status Register

Bit No.	Bit	Function
15:8	RX_XFR_ACK(7:0)	1: Indicates RX Transfer Acknowledge for USBD FIFO Endpoint N (N=7..0) 0: idle/busy
7:0	TX_XFR_ACK(7:0)	1: Indicates TX Transfer Acknowledge for USBD FIFO Endpoint N (N=7..0) 0: idle/busy

The USBD status register 1 provides the SW with internal handshake indication for RX and TX FIFOs. This register is read/write (writing ’1’ resets the bit).

Note: RX_XFR_ACK are reflected in the corresponding RX_BYTECNT registers.

USBD_STATUS_REG2 Reset Value: 0000_H

Table 78 USBD_STATUS_REG2 Status Register

Bit No.	Bit	Function
0	DEV_REM_WAKEUP_FEAT	1: Indicates that the DeviceRemoteWakeup Feature has been set by the host 0: DeviceRemoteWakeup Feature not allowed
15:1	Reserved	reserved

This register is read-only.

USBD_TIME_REG Reset Value: 0000_H

Table 79 USBD_TIME_REG Register

Bit No.	Bit	Function
(15:11)		reserved. Always ’0’
(10:0)	FRAME_NUMBER	Actual frame number

The USB Time register provides the actual 11-bit frame number information of the actual received start of frame packet.

USBD_Setup_REG01 Reset Value: 0000_H

Table 80 USBD_Setup_REG01 Register

Bit No.	Bit	Function
(15:8)	Setup_Byte1	Byte 1 of SETUP command
(7:0)	Setup_Byte0	Byte 0 of SETUP command

The USB Setup_REG01 register provides setup byte(1:0) of the SETUP command from the host. The register can not be overwritten by the host until the last setup register (67) is read by the device.

USBD_Setup_REG23 Reset Value: 0000_H

Table 81 USBD_Setup_REG23 Register

Bit No.	Bit	Function
(15:8)	Setup_Byte3	Byte 3 of SETUP command
(7:0)	Setup_Byte2	Byte 2 of SETUP command

The USB Setup_REG23 register provides setup byte(3:2) of the SETUP command from the host. The register can not be overwritten by the host until the last setup register (67) is read by the device.

USBD_Setup_REG45 Reset Value: 0000_H

Table 82 USBD_Setup_REG45 Register

Bit No.	Bit	Function
(15:8)	Setup_Byte5	Byte 5 of SETUP command
(7:0)	Setup_Byte4	Byte 4 of SETUP command

The USB Setup_REG45 register provides setup byte(5:4) of the SETUP command from the host. The register can not be overwritten by the host until the last setup register (67) is read by the device.

USBD_Setup_REG67 Reset Value: 0000_H

Table 83 USBD_Setup_REG67 Register

Bit No.	Bit	Function
(15:8)	Setup_Byte7	Byte 7 of SETUP command
(7:0)	Setup_Byte6	Byte 6 of SETUP command

The USB Setup_REG67 register provides setup byte(7:6) of the SETUP command from the host. The register can not be overwritten by the host until the last setup register (67) is read by the device.

Note: The Transmit/Receive handling by SW and EPEC is controlled by register groups per endpoint in address range 24_H to $6E_H$. The register group description is shown in detail for one endpoint and will be repetitive for the others.

USBD_TXWRn (n=7..0) Reset Value: 0000_H

Table 84 USBD_TXWRn Transmit Data FIFO EP#n Register

Bit No.	Bit	Function
(15:0)	TXWRn	16-bit word for Transmit FIFO endpoint#n (n=7..0)

The USB TXWR0 register provides 16-bit write access to the Endpoint#n Transmit FIFO.

USBD_TXEODn (n=7..0) Reset Value: 0000_H

Table 85 USBD_TXEODn Transmit End-of-packet Register

Bit No.	Bit	Function
(15:1)		reserved
0	TXEOD	’1’: Indicates last byte of packet transferred to Transmit FIFO endpoint#n (n=7..0) ’0’: EPEC/SW is idle/busy

The USB TXEODn register provides 16-bit read/write access. Indicates that the EPEC/SW has sent the last valid data byte to the Endpoint’s#n Transmit FIFO. Writing ’1’ to the register makes the next request for a data packet from Host answered by C161U with an empty data packet.

As long as a ’1’ is stored in the register, the packet has not been transfered to the host. If transferred, the bit is cleared.

Note: If bit TXEOD is written at the time an IN request from the host - due to missing transmit data - is answered by a NACK packet, bit TXEOD is cleared and the tx_done interrupt is asserted although the NACK packet has been sent instead of the empty packet.
Therefore, prior to setting the TXEOD bit, Software must reset the endpoint’s acknowledge-bit TX_XFR_ACKn in register USBD_STATUS_REG1. In reaction to endpoint’s tx_done interrupt, Software always should check the value of TX_XFR_ACKn to verify that the previous data transfer succeeded. If TX_XFR_ACKn indicates an unsuccessful data transmission, Software must repeat the transfer.

USBD_RXRRn (n=7..0) Reset Value: 0000_H

Table 86 USBD_RXRRn Receive Data FIFO EP#n Register

Bit No.	Bit	Function
(15:0)	RXRRn	16-bit word for Receive FIFO endpoint#n (n=7..0)

The USB RXRR0 register provides 16-bit read access to the Endpoint#n Receive FIFO.

USBD_RX_BYTECNTn (n=7..0) Reset Value: 0000_H

Table 87 USBD_RX_BYTECNTn Byte Counter Receive FIFO EP#n Register

Bit No.	Bit	Function
15	RX_STATUS	packet status indication '1': status ok '0': packet error
(14:10)	reserved	always '0'
(9:0)	RX_BYTECNT	10-bit byte counter for received packet in endpoint#n (n=7..0)

The USB BYTE_CNTn register provides 16-bit read access to the Endpoint#n Receive FIFO's 10-bit byte counter after a complete package is available. The status of the packet is indicated by bit 10 (see USB_STATUS_REG1).

USBD_CFGVAL_REG Reset Value: 0000_H

Table 88 USBD_CFGVAL_REG Command Register

Bit No.	Bit	Function
15:10		Reserved (always '0')
9:8	AS_IF3	Alternate Setting selected for Interface 3: 'xx': Alternate Setting 3-0 binary coded. (read-only)
7:6	AS_IF2	Alternate Setting selected for Interface 2: 'xx': Alternate Setting 3-0 binary coded. (read-only)

Table 88 USBD_CFGVAL_REG Command Register

Bit No.	Bit	Function
5:4	AS_IF1	Alternate Setting selected for Interface 1: 'xx': Alternate Setting 3-0 binary coded. (read-only)
3:2	AS_IF0	Alternate Setting selected for Interface 0: 'xx': Alternate Setting 3-0 binary coded. (read-only)
1:0	CFG	Configuration selected by host 'xx':Configuration 2-0 binary coded. (read-only)

The USBD CFGVAL register provides the current selection of the configuration and alternate setting done by the host. The SET_CONFIGURATION resets all settings to Alternate Setting 0, i.e. the Control Endpoint 0.

USBC_CMD_RESET Reset Value: 0000_H

Table 89 USBC_CMD_RESET USB RESET REGISTER

Bit No.	Bit	Function
(15:1)	Reserved	reserved
(0)	USBC_RST	Resets the USB block including the transmit and receive logic. The USB RESET does not reset the USBCLC register. The USB reset must be active for at least 10 clock cycles. The USB reset must always be activated after the USB device is connected to the USB

15.7 Programmers Guidlines: Using USB and EPEC

For normal functionality, the following interrupts must be enabled:

all udc_tx_done- and udc_rx_done-interrupts of those endpoints and in the direction in which they are used

udc_setup, udc_suspend, udc_suspendoff, udc_start of frame, udc_configval

optional interrupts are the EPEC-interrupt (this interrupt might only be used if SW needs to have complete control over the contents of the fifos and always wants to keep track of the status of the transmission) and the UDC_txwr- and UDC_rxrr-interrupts (which are generated every time, a fifo in the USB-block is ready to accept data or receive-data

might be read from the fifo; this interrupt is connected to the EPEC which in nomal functionality does this transfer).

By default, the bits 8 and 15 of the CMD register (auto flush enable and tx_protect) should be set.

15.7.1 Writing the configuration-value

After a system-reset, the USB-block expects the configurator of the UDC to be transferred via the tx-fifo(0) like a normal in-transfer. Best is using the EPEC for this endpoint. The source-pointer must be programmed to a memory-block containing the 85 bytes of the configurator, destination to usbd_txwr-register(0) and the length-register to 85 bytes. After starting the EPEC, it will transfer the configurator to the USB-block and after having transferred all the data, the EPEC-interrupt is generated and if the configurator went through the fifo, the udc_tx_done-interrupt(0) is also generated.

In order to allow generation of a next interrupt-pulse of the EPEC, SW must read the EPEC-interrupt-register and clear the interrupt-bit of endpoint 0 by writing a ‘1’ to it.

15.7.2 In-Transfer (Transmit)

An In-transfer means that the device will send data to the host by using endpointX. This transfer is started by a bulk_in, interrupt, control_in or iso_in request from the host.

- SW sets up source-pointer (to a memory-block containing the data to be sent), a destination-pointer (usbd_txwr-registerX) and the packetlenght (usually maxpacketlength) of the EPEC for the endpoint on which the transfer is to be done; the packetlength must always be the maxpacketlength for this endpoint for interrupt-, bulk_in or control_in-transfers except for the last packet to be transferred, for isochronous_in-endpoints this endpoint must be according to the sequence of packetlengths and the data which is available to be sent.
  If SW wants to send a packet of zero bytes, it must not use the EPEC, but must write ‘1’ to the corresponding usbd_tx_eod-register of the USB
- SW writes EPEC-start, EPEC transferres the data from the memory to the USB-fifoX; when host requests for the data, it is transferred through the fifo to the host; each time, the USB-block has space in the fifo and may accept a write into the usbd_txwr-register, the udc_txwr-interrupt is generated (for normal functionality this can be ignored as the EPEC uses this as a handshake for the next transfer)
- when EPEC has transferred all the data, it generates the EPEC-interrupt; SW must read the EPEC-interrupt-register and clear the (to the endpoint) corresponding bit by writing a ‘1’ to it
- when the transfer over the USB is finished, the udc_tx_done-interruptX is generated and SW can check the corresponding bit in the Status-Register, wheather the transfer was successful or not; this is for bulk, interrupt and control-Endpoints only, not for Isochonous Endpoints (where no ACK will be sent as Handshake)

- for non-Iso-transfers: if fhe transfer was ACK'd, the next packet can be set up for transmission, otherwise, host expects the same data to be resent
- SW must set up again the EPEC source-, destination-pointer and packetlength and start the transfer

If SW has already set up data in a tx-fifo and now, e.g. host changes the configuration or interfaces, SW can use a write into the command-register to flush the fifo of the corresponding endpoint. Before doing this, the EPEC-channel must be disabled or reprorgammed, otherwise the next pending bytes will be transferred into the tx-fifo.

15.7.3 Out-Transfer (Receive)

During an Out-transfer, host is transferring data to the device. SW must provide an free memory-block for each endpoint and set up the EPEC for moving arriving data from the USB-block to a free memory-location.

- SW provides an free memory-block and sets up source- (usbd_rxrr-registerX), destination-pointer (free memory-block) and packet lenght of the EPEC and sets the TXR_ENAx bit (refer to Table 18, "EPEC_CTRL_REGx Source Pointer Register," on page 101); the packetlength supported by the EPEC must always be an even number of bytes (in receive-direction the EPEC only does word-transfers) and have at least space for the maxpacketlength of the endpoint.
- when host sends data, it is forwarded through the fifo's and with every transferred word that can be read from usbd_rxrr-registerX the usbd_rxrr-interruptX is set (for normal functionality this can be ignored); EPEC transferes this data into the memory
- EPEC generates an EPEC-interrupt when it has transferred all the data into the memory (this interrupt is generated shortly after the USB-block has generated the udc_rx_done-interrupt); SW must read the EPEC-interrupt-register and clear the (to the endpoint) corresponding bit in this register by writing a '1' into it
- when USB-block has finished the whole transfer, it generates the udc_rx_done-interrupt; SW must then read the usbd_rx_bytecnt-register, in order to determine the number of bytes of the received packet and to release the interlocking of the fifo for the next transfer; the most significant bit in this register contains also the status-bit of the status-register and shows whether this packet had transmission-errors or not
  if host has sent a packet of zero lenght, no rxrr-interrupt is generated but only a rx_done-interrupt; here also SW must read the rxbytecountX-register
- in order to prepare a receive on this endpoint again SW must provide a new free memory-block, set up the source-, destination-pointer and packetlength of the EPEC and write the start-bit

15.7.4 Reading out Setup-Packets

Setup-packets are treated without the EPEC. If host sends a setup-packet which is forwarded to the CPU (there are only three commands: get_descriptor, set_descriptor and synch_frame all the other ones are treated internally), and the packet is valid, the

udc_setup-interrupt is generated and SW must read all four setup-registers. By reading the last one (setup 61, an overwrite-protection for those registers will be released and the device will accept the next setup-packet. As long as this register is not read, each setup-packet on the USB will be NAK'd.

15.7.5 Special case: Setup-Transfer

The endpoint zero has an additional feature in transmit-direction in order to handle an early end of a setup-transaction. If, as an example, host requests for a device-desciptor (by sending a get_descriptor(device_descriptor)) with a length of 12 bytes and the maxpacketlength of endpoint zeo is eight, host must request for two control_in-packets and then accnwowledge them with an status_out-packet. According to the USB-spec, host can also request only one packet or even none before it sends out the status-packet. This is called an early end of the setup-transaction.

If SW has already set up the data in the tx-fifo after the get_descriptor-command and host will immediately after this request for different data with another setup-packet, the old data, already in the fifo would be sent. By enabling the AutoFlushEnable-feature, with every setup-packet which is visible for the CPU will flush the tx-fifo for endpoint zero. This will avoid wrong, old data to be sent over control-enpoint zero.

This flush-mechanism could also be done by flushing the usb with a write into the command-register of the USB-block but the flush initiated by SW might happen too late.

Note: The AutoFlushEnable-feature described above is only available if the C161U CPU is running with 36 MHz. This feature can not be used if the device is running with 24 MHz. In this case, pending data has to be flushed explicitly by Software via register USBD_CMD_REG on reception of the OUT packet that ends setup transfer, indicated by the rx_done interrupt of endpoint 0.

15.7.6 Setting of configuration and alternate settings of interfaces

Each time the host send a valid set_configuration- or set_interface-command, this will show up for the CPU as a configval-interrupt. In order to determine the actual confitguration and the alternate setting of an interface, SW must read the configval-register and set up the endpoints which are actually enabled.

There is no overwirte-protection on this register, the value is always updated if a valid set_configuration or set_interface-packet is received.

15.7.7 Stalling Endpoints

All transmit-endpoints can be stalled by writing into the command-register, where it makes no sense to stall an isochronous endpoint (for a isochronous packet there is no stall-handshake, so the host will never notice that the endpoint is stalled and thus will never try to abolish the stall-condition). The stall will be kept as long the bit in the command-register is set and the host did not send a clear_feature-command.

If an endpoint was stalled during a in-transfer, this transfer will be finished and the next request by the host will return a stall-handshake. If there is data in the fifo for transission, the data will be kept and sent, if the stall-condition is abolished and host requests for this data.

A stall-condition during an out-transfer will finish first and the next request will return a stall-handshake.

Endpoint zero has only one stall-bit for both directions, for in and out. A stall on one direction will also lead to a stall in the other transfer-direction.

15.7.8 Start of Frame

Host sends a Start of Frame-(SOF)-signal every ms. A SOF-interrupt is generated and the value of the actual frame-number is stored in the SOF-register. There is no overwirte-protection on this register, the value is always updated if a valid SOF-packid is received.

15.7.9 Suspend and Suspendoff

In normal operation, there is the UDC-clock of 48 MHz enabled and the normal CPU-clock which may vary according to the divider in the clock-generation-unit.

If the host is sending a suspend-request (by driving an idle-state for more than 6 ms), after 6 ms the suspend-interrupt will be generated. This must cause SW to go in low-power mode. There are different modes in which the chip can be set. According to the mode the wakeup initiated by the host, must be detected differently:

- Using the bit 0 in clc register of every peripheral to turn off the clock. The suspendoff interrupt is generated even though the rest of the usbblk is turned off.
- Using the SYSCON3 register to turn off the clock of xbus and pdbus peripherals (peripheral disable only). The SYSCON3 register is a write protected register and SW first must go into low protected mode to be able to do this (see page 393 for SYSCON3 register description). In this mode also, the suspendoff interrupt is generated.
- Using the SYSCON3 register group disable (msb of the register) to turn off all the xbus and pdbus peripherals. In this mode, the normal suspendoff interrupt is not generated, wakeup must be done with the falling edge of the fast external interrupt alternate function firq_alt(5).
- Going into sleep mode which stops program execution and turns off the clock for most of the entire chip. In this mode the fast external interrupt alternate function firq_alt(5) is also generated and will wakeup the cpu. Program execution will start with the interrupt procedure of the interrupt, or, if SW was in an interrupt routine with a higher priority before, program execution will continue at the point, it was stopped.

If SW wants to send a device-wakeup this feature must have been enabled by the host. Whether this feature was enabled or not, is reflected in the STATUS3 register. If this feature is enabled, and SW wants to wake up the USB, it must turn on the clocks and

write the resume-bit of the command-register. This will drive the non-idle-state on the USB for 3 ms, host will start a wakeup-procedure.

15.7.10 Device disconnecting

Either our device is bus- or self-powered. In the case of being bus-powered, every time the device is disconnected from the bus, the power supply will break down and a re-plugging will restart with a reset of the entire chip.

In case of a self-powered device, if there is a disconnection from the USB, the logic of the usbblk must be reset. There must be external logic added to provide the detection of a disconnection. If SW detects a dis- and reconnection, it must disable all the epec channels and reset the whole usbblk by writing into the usbd_cmd_reset register. After de-asserting the reset, the whole configuration process (with writing of the configurator) must be redone.

16 Watchdog Timer (WDT)

To allow recovery from software or hardware failure, the C161U 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.

The watchdog timer provides two registers: a read-only timer register that contains the current count, and a control register for initialization.

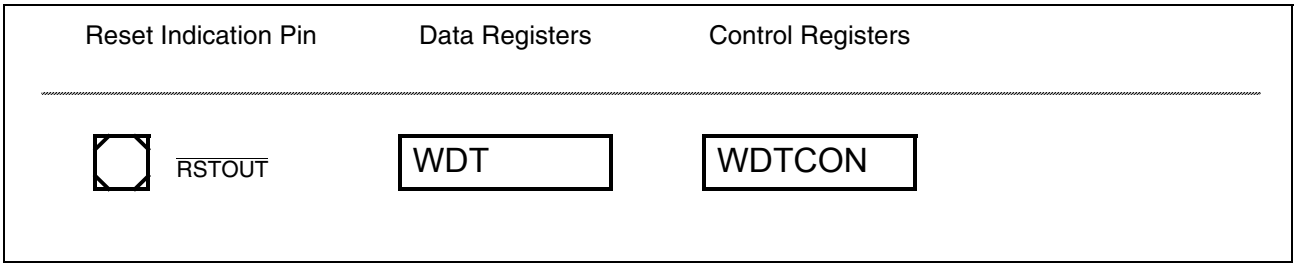


Figure 109 SFRs and Port Pins associated with the Watchdog Timer

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 on each service access.

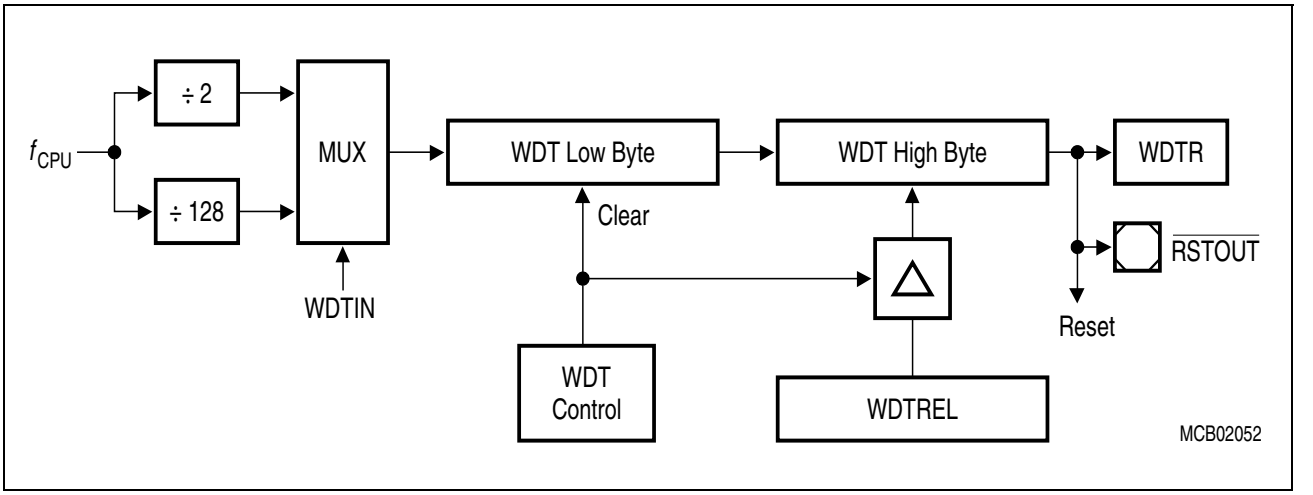


Figure 110 Watchdog Timer Block Diagram

16.1 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 a flag that indicates a watchdog timer overflow.

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								RESERVED			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 Set by the watchdog timer on an overflow. Cleared by the SRVWDT instruction.	
SWR	Software Reset Set by the command SRST	
SHWR	Short Hardware Reset Set by the input RSTIN	**Note**: C161U does not distinguish between short and long hardware reset.
LHWR	Long Hardware Reset Set by the input RSTIN	
reserved	Reserved These bits are reserved	
WDTREL	**Watchdog Timer Reload Value** (for the high byte)	

The reset sources supported by the C161U are summarized in **Table 90**.

Note: Differentiation between long and short hardware reset, known from other Infineon C16x devices, is not supported.

Note: An internal power-on detection circuitry, also known from other C16x devices, is not implemented. Therefore, bit WDTCON.5 (in other devices called PONR - power-on reset) is reserved.

Table 90 WDTCON Register: Reset Source Identification

Type of Reset	WDTCON Reset Value	WDTCON Flags being set
Hardware reset via pin $\overline{\text{RSTIN}}$	$001C_H$	LHWR, SHWR, SWR
Software reset via command SRST	0004_H	SWR
Watchdog Timer reset	0006_H	SWR, WDTR

Note: The WDTCON register bits [7, 6, 5] 4, 3, 2 and 1 are cleared by the EINIT command.

After any software reset, external hardware reset, 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.

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. It differs from a software or external hardware reset in that bit WDTR (Watchdog Timer Reset Indication Flag) of register WDTCON will be set. A hardware reset or the SRVWDT instruction will clear this bit. Bit WDTR can be examined by software in order to determine the cause of the reset.

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.

Note: After a hardware reset that activates the Bootstrap Loader the watchdog timer will be disabled.

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 (<WDTREL> * 2^8). Instruction SRVWDT has been encoded in such a way that the chance of unintentionally servicing the watchdog timer (eg. 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:

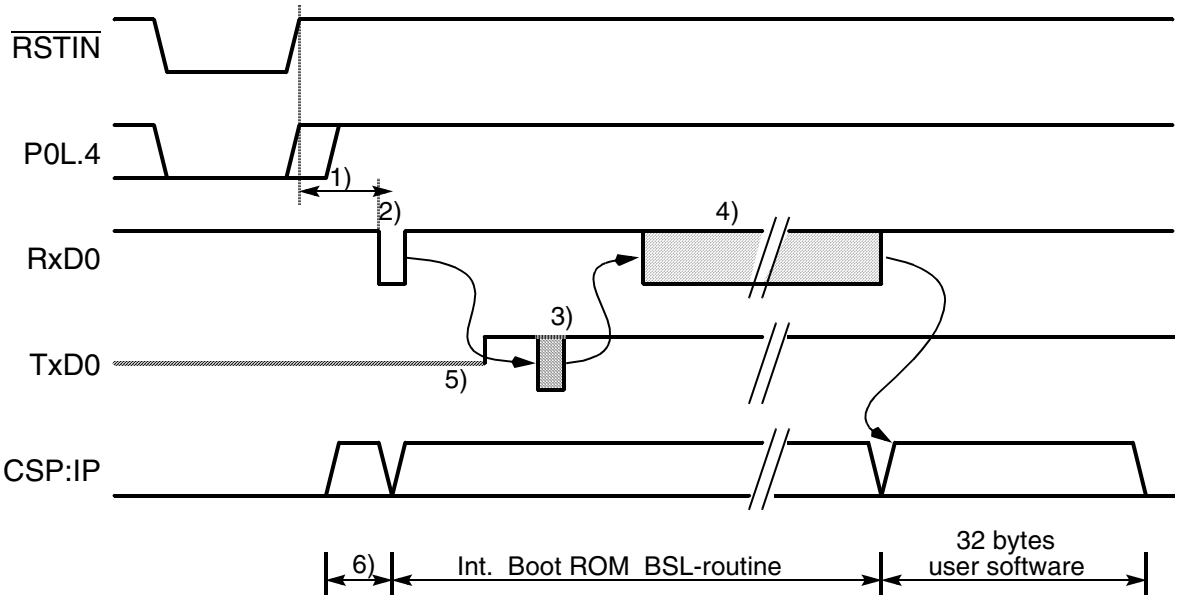
$$P_{WDT} = \frac{2^{(1 + <WDTIN>*6)} * (2^{16} - <WDTREL> * 2^8)}{f_{CPU}}$$

Note: For safety reasons, the user is advised to rewrite WDTCON each time before the watchdog timer is serviced.

17 Bootstrap Loader

The built-in bootstrap loader of the C161U provides a mechanism to load the startup program, which is executed after reset, via the serial interface.

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. It may be used to provide lookup tables or may provide “core-code”, ie. a set of general purpose subroutines, eg. for I/O operations, number crunching, system initialization, etc.



1) BSL initialization time, > 2µs @ 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 C161U.

4) 32 bytes of code / data, sent by host.

5) Caution: TxD0 is only driven a certain time after reception of the zero byte (2.5µs @ f_{CPU} = 20 MHz).

6) Internal Boot ROM.

Figure 111 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.

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

C161U 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.

After entering BSL mode and the respective initialization the C161U scans the RXD0 line to receive a zero byte, ie. 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 ASC accordingly and switches pin TxD0 to output. Using this baudrate, an identification byte is returned to the host that provides the loaded data.

This identification byte identifies the device to be booted. The following codes are defined for Infineon Technologies microcontrollers:

55_H: 8xC166.
$A5_H$: Previous versions of the C167 (obsolete).
$B5_H$: C165.
$C5_H$: C167 derivatives.
$D5_H$: C161U (and all other 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 C161U has entered BSL mode, the following configuration is automatically set (values that deviate from the normal reset values, are ***marked***):

Watchdog Timer:	***Disabled***	Register SYSCON:	$0E00_H$
Context Pointer CP:	$FA00_H$	Register STKUN:	$FA40_H$
Stack Pointer SP:	$FA40_H$	Register STKOV:	$FA0C_H$ 0<->C
Register S0CON:	$\boldsymbol{8011_H}$	Register BUSCON0:	acc. to startup config.
Register S0BG:	acc. to '00' byte	P3.10 / TXD0:	'***1***'
		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 C161U can return the identification byte.

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.

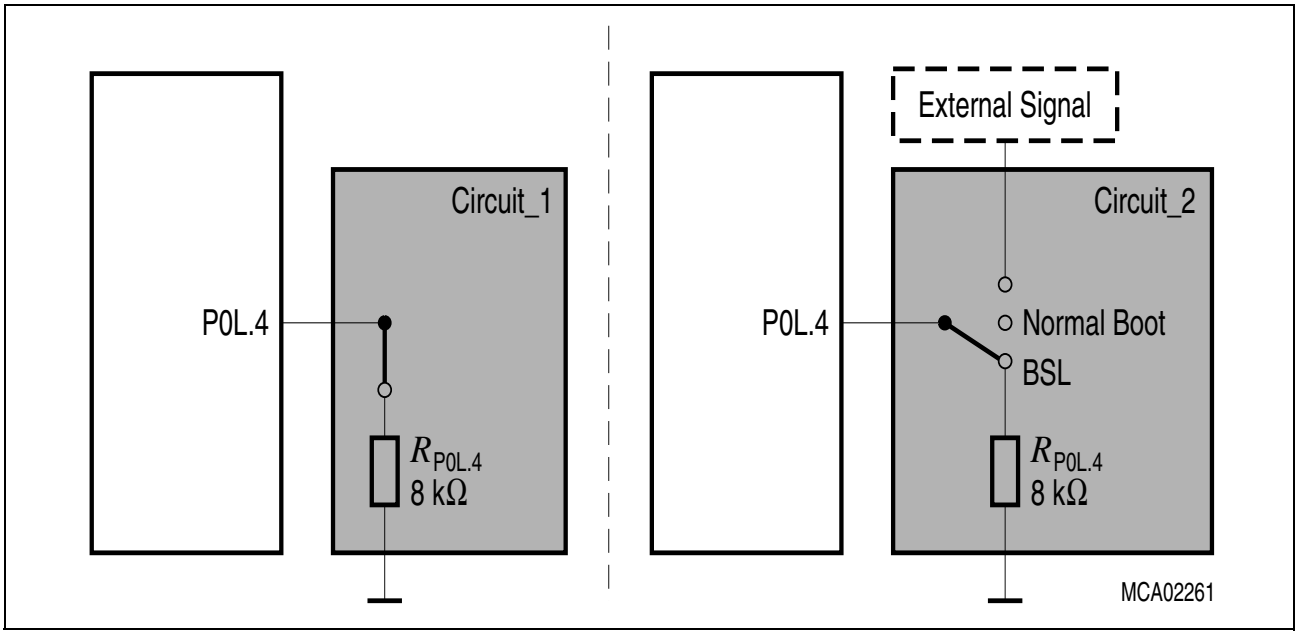


Figure 112 Hardware Provisions to Activate the BSL

After sending the identification byte the ASC 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“.

Loading the Startup Code

After sending the identification byte the BSL enters a loop to receive 32 bytes via ASC. 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$_H$, ie. the first loaded instruction. The bootstrap loading sequence is now terminated, the C161U 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 ASC 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 C161U will still run in BSL mode, ie. with the watchdog timer disabled and limited access to the internal code memory.

Exiting Bootstrap Loader Mode

In order to execute a program in normal mode, the BSL mode must be terminated first. C161U 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 C161U will start executing from location $00'0000_H$ of the external memory (make sure, pin $\overline{EA}$ is tied to '0' signal).

Choosing the Baudrate for the BSL

The calculation of the serial baudrate for ASC from the length of the first zero byte that is received, allows the operation of the bootstrap loader of the C161U with a wide range of baudrates. However, the upper and lower limits have to be kept, in order to insure proper data transfer.

$$B_{C161U} \quad \frac{f_{CPU}}{32 \cdot (S0BRL + 1)}$$

C161U 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:

$$S0BRL = \frac{T6 - 36}{72} \quad , \quad T6 = \frac{9}{4} \cdot \frac{f_{CPU}}{B_{Host}}$$

For a correct data transfer from the host to the C161U the maximum deviation between the internal initialized baudrate for ASC and the real baudrate of the host should be below 2.5%. The deviation (F_B, in percent) between host baudrate and C161U baudrate can be calculated via the formula below:

$$F_B = \left| \frac{B_{Contr} - B_{Host}}{B_{Contr}} \right| \cdot 100 \ \% \quad , \quad F_B \leq 2{,}5 \ \%$$

Note: Function (F_B) does not consider the tolerances of oscillators and other devices supporting the serial communication.

This baudrate deviation is a nonlinear function depending on the CPU clock and the baudrate of the host. The maxima of the function (F_B) increase with the host baudrate due to the smaller baudrate prescaler factors and the implied higher quantization error (see figure below).

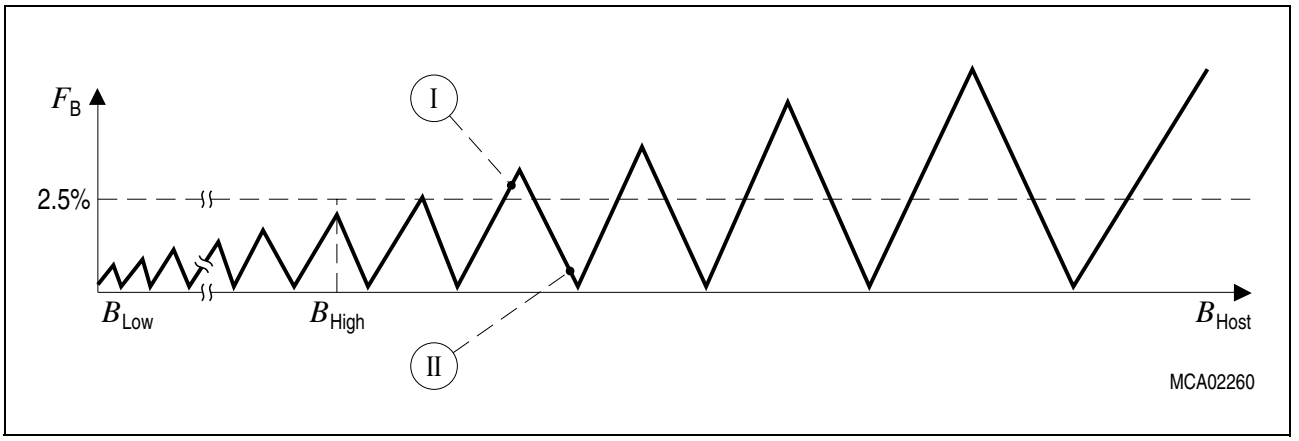


Figure 113 Baudrate deviation between host and C161U

Minimum baudrate (B_{Low} in the figure above) is determined by the maximum count capacity of timer T6, when measuring the zero byte, ie. it depends on the CPU clock. Using the maximum T6 count 2^{16} in the formula the minimum baudrate for f_{CPU}=20 MHz is 687 Baud. The lowest standard baudrate in this case would be 1200 Baud. Baudrates below B_{Low} would cause T6 to overflow. In this case ASC cannot be initialized properly.

Maximum baudrate (B_{High} in the figure above) is the highest baudrate where the deviation still does not exceed the limit, ie. 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 eg. 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.

18 System Reset

The internal system reset function provides initialization of the C161U into a defined default state and is invoked either by asserting a hardware reset signal on pin RSTIN (Hardware Reset Input), upon the execution of the SRST instruction (Software Reset) or by an overflow of the watchdog timer (WDT).

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 I/O pin drivers are switched off (tristate). RSTOUT is activated depending on the reset source.

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 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 RSTIN input is inactive. When this internal reset condition is removed (reset sequence complete and RSTIN inactive), the reset configuration is latched from PORT0, and pins ALE, RD and WR are driven to their inactive levels.

Note: Bit ADP, which selects the Adapt mode during low RSTIN signal, is latched with the rising edge of 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.

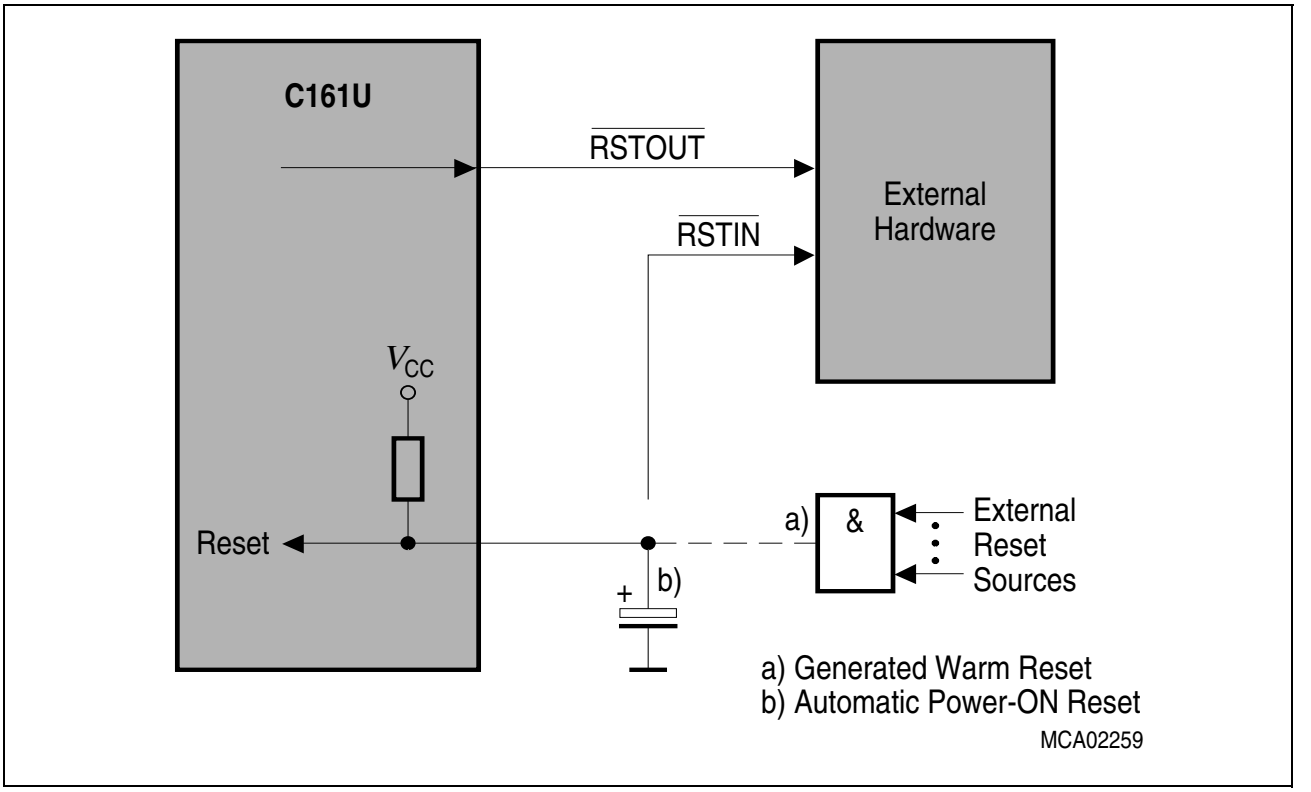


Figure 114 External Reset Circuitry

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 8 CPU clock cycles.

Note: During reset, the CPU is clocked with the free-running PLL clock which may run as slow as < 1 MHz.

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 form an external peripheral. During this settling time the configuration may intermittently be wrong. In such a case also the PLL clock selection may be wrong.

Note: To ensure a glitch free start-up of the C161U, it is strongly recommended to provide an external reset pulse of ca. 1 ms in order to allow the PLL to settle on the desired CPU clock frequency.

The input RSTIN provides an internal pullup device equalling a resistor of 100 KΩ to 660 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 114**. RSTIN may also be connected to the output of other logic gates, see a) same figure.

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, eg. to leave bootstrap loader mode, or upon a hardware trap that reveals a system failure. C161U's latched in reset configuration on software reset is shown in **Figure 116**, page 368.

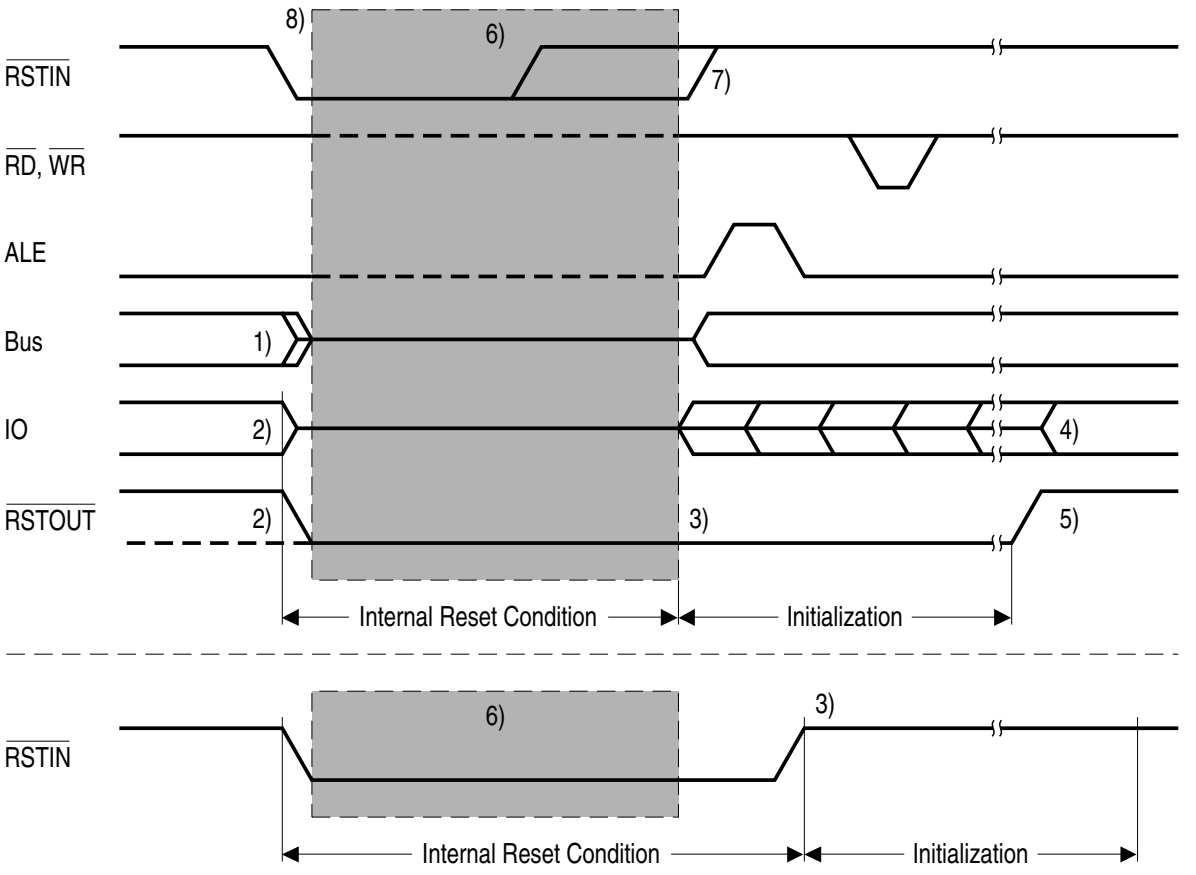
Watchdog Timer Reset

When the watchdog timer is not disabled during the initialization or serviced regularly during program execution it 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 READY at all, or if READY is sampled active (low) after the programmed waitstates. When READY is sampled inactive (high) after the programmed waitstates the running external bus cycle is aborted. Then the internal reset sequence is started.

Note: For latched in watchdog reset configuration, refer to **Figure 116**, page 368.
The watchdog reset cannot occur while the C161U is in bootstrap loader mode!

C161U's Pins after Reset

After the reset sequence the different groups of pins of the C161U are activated in different ways depending on their function. Bus and control signals are activated immediately after the reset sequence according to the configuration latched from PORT0, so either external accesses can take place or the external control signals are inactive. The general purpose I/O pins remain in input mode (high impedance) until reprogrammed via software (see figure below). The RSTOUT pin remains active (low) until the end of the initialization routine (see description).



When the internal reset condition is extended by 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 asinchronously with RSTIN, sinchronously upon software or watchdog reset.
3) The reset condition ends here. The C 167CR 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 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 RSTIN line in Bidirectional reset mode.

MCS02258

Figure 115 Reset Input and Output Signals

Reset Output Pin

The RSTOUT pin is dedicated to generate a reset signal for the system components besides the controller itself. RSTOUT will be driven active (low) at the begin of any reset sequence (triggered by hardware, the SRST instruction or a watchdog timer overflow). 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.

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 (18 MHz @ f_{CPU}=36 MHz), and its default reload value is 00_H, so a watchdog timer overflow will occur 131072 CPU clock cycles (3.64 ms @ f_{CPU}=36 MHz) 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 or by servicing the watchdog timer. 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.

Note: For a complete description of register WDTCON, refer to **Chapter 16.1**, page 352.

Reset Values for the C161U Registers

During the reset sequence the registers of the C161U 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 configuration)
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)

Internal RAM after Reset

The contents of the internal RAM are not affected by a system reset. However, after power-on 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 hardware reset, software reset or watchdog reset, but are undefined after power-on.

Ports and External Bus Configuration during Reset

During the internal reset sequence all of the C161U'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 C161U 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 $\overline{RD}$ and $\overline{WR}$ are held high through internal pullups. Also the pins selected for $\overline{CS}$ output will be pulled high.

The registers SYSCON and BUSCON0 are initialized according to the configuration selected via PORT0:

- 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

Note: In the C161U, pin $\overline{EA}$ must always be set to ’0’. The "internal start" ($\overline{EA}$=’1’), known from other Infineon C16x devices is not supported.

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. The Ready function is disabled at the end of the internal system reset.

When the internal reset has completed, the configuration of PORT0, PORT1, Port 4, Port 6 and of the $\overline{BHE}$ signal (High Byte Enable, alternate function of P3.12) depends on the bus type which was selected during reset. When any of the external bus modes was selected during reset, PORT0 (and PORT1) 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 $\overline{CS}$ lines ($\overline{CS0}$ will be ‘0’, while the other active $\overline{CS}$ 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 C161U 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 $00'0007_H$) 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 C161U does not fetch instructions from location $00'0000_H$ but rather expects data via serial interface ASC.

The first instruction is fetched from external memory. To decrease the number of instructions required to initialize the C161U, 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 initializating 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.

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

- disables the action of the DISWDT instruction,
- disables write accesses to register SYSCON,

Note:All configurations regarding register SYSCON (enable CLKOUT, stacksize, etc.) must be selected before the execution of EINIT.

- disables write access 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 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 C161U 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.

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, ie. 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 C161U.

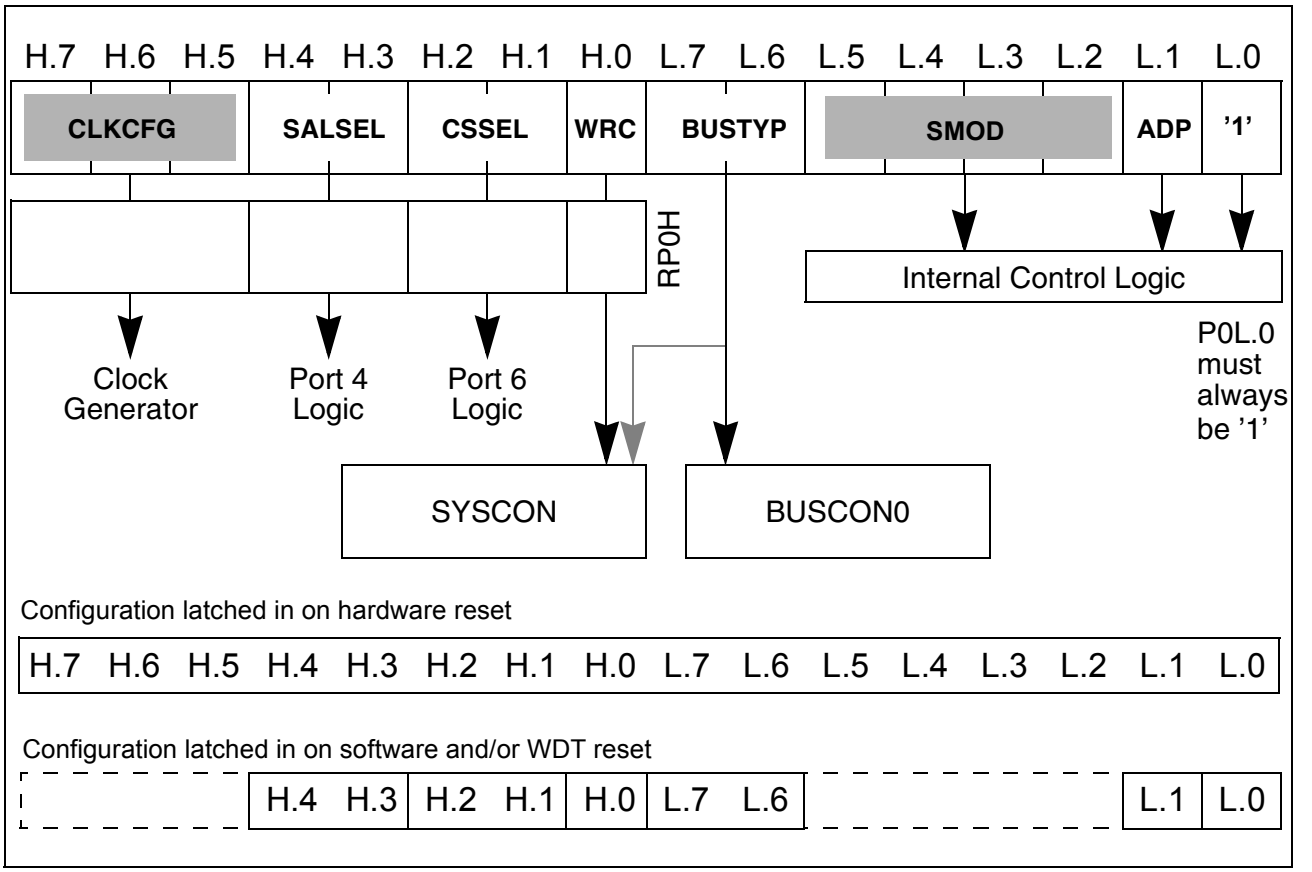


Figure 116 PORT0 Configuration during Reset

Note: The configuration on pins P0H.7:P0H.5 (CLKCFG) and P0L5:P0L.2 (SMOD) is latched in on a hardware triggered reset only and will not be evaluated by the C161U on a software and/or WDT reset.

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 reset configuration needs to be held on port P0 throughout the start-up phase until the C161U takes over control of the external XBUS. This is first indicated by driving the XBUS output lines ALE, $\overline{RD}$, $\overline{WR}$/$\overline{WRL}$, $\overline{CS}$, P4, P1H and P1L. Since it might prove infeasible to detect the change from tristate to a strongly driven value, the first rising edge of ALE can be used for indication of the end of the reset configuration hold time. The first rising edge of ALE occurs 4 CPU cycles after taking control of the external bus.

The following describes the different selections that are offered for reset configuration. The default modes refer to pins at high level, ie. without external pulldown devices connected. **Table 91** shows a summary of all modes, supported by the C161U.

Note: The Emulation Mode, known from other C16x Infineon devices, is not supported by the C161U. Make sure, on pin P0L.0 a HIGH signal is always latched in. HIGH on P0L.0 is the default configuration and is supported by the internal pull-up device.

Adapt Mode

Table 91 C161U's Supported Modes and Related Reset Configurations

P0L.5 : P0L.2 (SMOD)	P0L.1 (ADP)	Selected Mode
x x x x	0	Adapt Mode
1 1 1 1	1	Normal Mode
0 0 0 1	1	Internal Boot-ROM Read-Out
1 0 1 1	1	Bootstrap-Loader Mode
1 1 0 1	1	Selftest

Pin P0L.1 (ADP) selects the Adapt Mode when low during reset. It is latched with the rising edge of RSTIN. In this mode the C161U goes into a passive state, which is similar to its state during reset. The pins of the C161U float to tristate or are deactivated via internal pullup/pulldown devices, as described for the reset state. In addition also the RSTOUT pin floats to tristate rather than be driven low, and the on-chip oscillator is switched off.

This mode allows switching a C161U that is mounted to a board virtually off, so an emulator may control the board's circuitry, even though the original C161U remains in its place. The original C161U also may resume to control the board after a reset sequence with P0L.1 high.

Default: Adapt Mode is off.

Bootstrap Loader Mode

Pin P0L.4 (BSL) activates the on-chip bootstrap loader, when low during reset. The bootstrap loader allows moving the start code into the internal RAM of the C161U via the serial interface ASC. The C161U will remain in bootstrap loader mode until a hardware reset with P0L.4 high or a software reset.

Default: The C161U 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 EA. This allows the configuration of the external bus interface of

the C161U 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, ie. $\overline{\text{WR}}$ control and $\overline{\text{BHE}}$. When low, it selects the alternate configuration, ie. $\overline{\text{WRH}}$ 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 4 chip select lines active ($\overline{\text{CS3}}$...$\overline{\text{CS0}}$).

Note: The selected number of $\overline{\text{CS}}$ signals cannot be changed via software after reset.

CSSEL	Chip Select Lines	Note
1 1	Four: $\overline{CS3}$...$\overline{CS0}$	Default without pull-downs
1 0	None	Port 6 pins free for I/O
0 1	Two: $\overline{CS1}$...$\overline{CS0}$	P6.4..P6.2 free for GPI/O
0 0	Three: $\overline{CS2}$...$\overline{CS0}$	P6.4..P6.3 free for GPI/O

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	Eight: A20...A16	2 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 C161U internally uses its complete 24-bit addressing mechanism. This allows the restriction of the width of the effective address bus, while still deriving $\overline{CS}$ signals from the complete addresses.

Default: 2-bit segment address (A17...A16) allowing access to 256 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 clock generation mode (on-chip PLL) during reset. Please refer to Chapter 3.3, "Clock Generation Concept".

19 Power Reduction Modes

Two different power reduction modes with different levels of power reduction have been implemented in the C161U, which may be entered under software control.

In **Idle mode** the CPU is stopped, while the 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. 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.

19.1 Idle Mode

The power consumption of the C161U microcontroller can be decreased by entering Idle mode. In this mode all peripherals, **including** the watchdog timer, continue to operate normally, only the CPU operation is halted.

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.

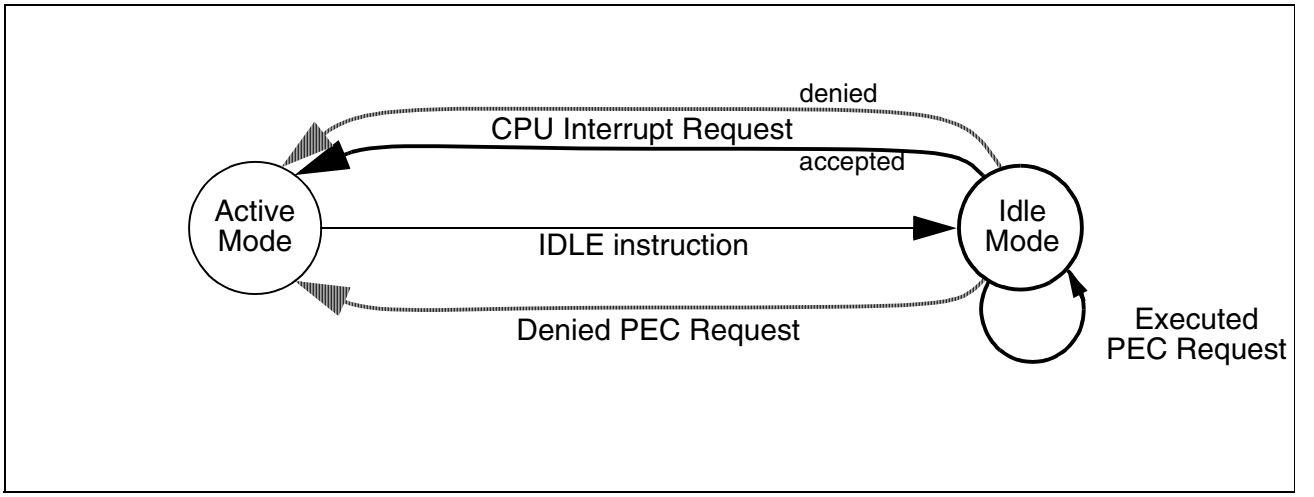


Figure 117 Transitions between Idle mode and active mode

Idle mode can also be terminated by a Non-Maskable Interrupt, ie. 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.,

The standard Idle mode can be additionally configured by programming the SYSCON3 register, using the flexible peripheral management functions. This is especially advantages, because it is thus possible to activate only these peripherals also in Idle mode which are really required for standby operation or for wakeup, reducing power consumption to the absolute minimum for a specific peripheral operation during Idle mode.

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, the contents of the internal RAM, however, are preserved through the voltage supplied via the V_{CC} pins. The watchdog timer is stopped in Power Down mode. This mode can only be terminated by an external hardware reset, ie. by asserting a low level on the 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 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 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 set a flag or write a certain bit pattern into specific RAM locations, and then execute the PWRDN instruction. If the NMI pin is still low at this time, Power Down mode will be entered, otherwise program execution continues. During power down the voltage at the V_{CC} pins can be lowered to 2.5 V while the contents of the internal RAM will still be preserved.

The initialization routine (executed upon reset) can check the identification flag or bit pattern within RAM to determine whether the controller was initially switched on, or whether it was properly restarted from Power Down mode.

19.3 Status of Output Pins during Idle and Power Down Mode

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 (eg. WR), or to a defined state which is based on the last bus access (eg. 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 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.

The table below summarizes the state of all C161U output pins during Idle and Power Down mode.

C161U Output Pin(s)	Idle Mode		Power Down Mode	
	No external bus	External bus enabled	No external bus	External bus enabled
ALE	Low	Low	Low	Low
RD, WR	High	High	High	High
CLKOUT	Active	Active	High	High
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
BHE	Port Latch Data	Last value	Port Latch Data	Last value
HLDA	Port Latch Data	Last value	Port Latch Data	Last value
BREQ	Port Latch Data	High	Port Latch Data	High
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 $\overline{CS}$ signal that corresponds to the last address remains active (low), all other enabled $\overline{CS}$ signals remain inactive (high). By accessing an on-chip X-Periperal prior to entering a power save mode all external $\overline{CS}$ signals can be deactivated.

19.4 Extended Power Management

Infineon Technologies C16x's well known basic power reduction modes (Idle and Power Down) are enhanced by a number of additional power management features. These features can be combined or selectively used to reduce the controller's power consumption to the respective application's possible minimum. According to the sense of platform modularity, the extended power management functions are controlled by different submodules and registers, as follows::

Sub Module	Control Register
Extended Power Management /Sleep Mode Control	SYSCON1
Flexible Clock Generation Management	SYSCON2
Flexible Peripheral Management	SYSCON3

C161U's power management functions can be supplemented by the Real Time Clock (RTC) timer with optional periodic wakeup from Sleep or Idle mode. The periodic wakeup combines the drastically reduced power consumption in power reduction modes (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. The RTC is fully controlled by the C161U's power reduction submodules.

The Extended Power Management Module controls the Sleep mode. The Sleep mode is a new power management function which represents and is equal to a Power Down mode but with exit/wakeup handling as in Idle mode. Wakeup out of Sleep state is possible with all external interrupts (including alternate sources e.g. from ASC interface), with $\overline{NMI}$ and with RTC interrupts. As in Idle mode also PEC requests are executed in Sleep mode, resulting in an interruption and resumption of Sleep mode. The watchdog timer is stopped in Sleep mode. The contents of internal RAM and of CBC's registers are preserved through the voltage supplied via the VDD pins.

As in Power Down mode, the Sleep mode may also be combined with a running real time clock RTC. In Sleep mode the oscillators (RTC and selected oscillator optionally), the

PLL as well as the whole clock system is stopped as in power down state. This implies - contrary to Idle mode - , that after wakeup the exit of Sleep mode and thus the start of any CPU operation is normally delayed by the ramp-up time of the clock system (oscillator, PLL). Only when the PLL clock is locked on configured frequency, the system clock is started and the following processing is identical to wakeup from Idle mode.

For description of Idle mode and its possibilities of configuration see Chapter 19.1, "Idle Mode".

Register in Extended Power Management Module :

Register	Description
SYSCON1	System configuration control register for sleep management

Note: SYSCON1 is a protected register; its security level is automatically set to full write protection after execution of EINIT instruction.

The power reduction modes *Idle* and *power down* are extended by the Infineons C16x devices newly introduced sleep mode.

19.4.1 Sleep Mode

The Sleep mode is a new power management function which represents and is equal to a Power Down mode but with exit/wakeup handling as in Idle mode. Wakeup from Sleep state is possible with all external interrupts (including alternate sources e.g. from SSC interface), with $\overline{\text{NMI}}$ and with RTC interrupts. As in Idle mode also PEC requests are executed in Sleep mode, resulting in an interruption and resumption of Sleep mode. The watchdog timer is stopped in Sleep mode. The contents of internal RAM and registers are preserved through the voltage supplied via the VDD pins.

Generally, the external bus and the XBUS are released during Sleep mode if enabled by the Hold Enable bit HLDEN in the last Program Status Word PSW. If enabled, the signal HLDA is active as long as the Sleep mode (as well as the Idle or Power Down mode) is active. Only when the clock is available again after wakeup, the HOLD request signal is sampled and the HLDA state continued until HOLD is deactivated.

As in Power Down mode, the Sleep mode may also be combined with a running real time clock RTC. In Sleep mode the oscillators (RTC and selected oscillator optionally), the PLL as well as the whole clock system is stopped as in Power Down state. This implies, that after wakeup the exit of Sleep mode normally is delayed by the ramp-up time of the clock system (oscillator, PLL).

Sleep mode is entered after the standard IDLE instruction (protected 32 bit instruction) has been executed and the instruction before the IDLE instruction has been completed. The selection between standard Idle mode and Sleep mode is controlled with the new register SYSCON1 (see below).

Note: Sleep mode cannot be entered in Slow Down mode - the start of sleep mode and wakeup is only possible in the normal clocking mode (PLL or direct drive) as defined with the startup configuration on port P0. If Sleep mode shall be entered during Slow Down mode, automatically the standard Idle mode is selected as configured with SYSCON3 register.

The Sleep mode is controlled by bitfield SLEEPCON within register SYSCON1.

SYSCON1 ($F1DC_H$ / EE_H) **ESFR-b** **Reset Value: 0000_H**

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

Note: SYSCON1 is write protected after the execution of EINIT unless it is released via the unlock sequence.

General description of SYSCON1 bits:

Bit	Function
SLEEPCON	SLEEP Mode Configuration ‘0 0’: normal IDLE mode ‘0 1’: SLEEP mode with running RTC ‘1 0’: reserved ‘1 1’: SLEEP mode with stopped RTC and stopped OSC

Before entering Sleep mode with the IDLE instruction, the continuation of instruction processing **after** termination of Sleep mode must be prepared as known from standard Idle mode. For wakeup with interrupt, four general possibilities of continuation can be selected, which are controlled (prepared) as follows:

- **Continuation with first instruction after the IDLE instruction** will be enabled if
  - interrupts are globally disabled with the Interrupt Enable bit in PSW, **or**
  - the interrupt is enabled by global (PSW) and by individual (interrupt control register) enable bit, but the current CPU priority level (in PSW) of IDLE instruction is higher than the interrupt level.
- **Continuation with first instruction of dedicated interrupt service routine** will be selected if
  the interrupt is enabled by global (in PSW) and by individual (interrupt control register) enable bit, and the CPU priority level of IDLE instruction is lower than the interrupt level, thus the enabled interrupt has highest priority. Additionally, PEC Transfer for this interrupt is not enabled. The continuation with the dedicated service routine is **always** performed in case of NMI hardware traps, independently of any enable bit or CPU priority level.

- **Execution of one PEC Transfer and resumption of Sleep mode** will be selected if the interrupt is enabled by global (in PSW) and by individual (interrupt control register) enable bit, and the CPU priority level of IDLE instruction is lower than the interrupt level, thus the enabled interrupt has highest priority. Additionally, PEC Transfer for this interrupt is enabled.
- **Continuation with standard Idle mode as configured with register SYSCON3** if the interrupt is not enabled with the individual Interrupt Enable flag in its interrupt control register. Note: In standard Idle mode the watchdog timer has to be serviced. For description of SYSCON3 see ´description of Peripheral Management Module.

As wakeup from Idle mode, wakeup from Sleep mode is performed with any enabled interrupt request. Sleep mode is terminated and the before selected (and above described) continuation of processing is executed, if one of the following interrupts occur:

- **Fast External Interrupts (EXxINT).** All fast external interrupts can be selected for wakeup from Sleep mode by defining the related trigger transitions (edges) in the EXICON register. All transition types are allowed also in Sleep mode.
- **Alternate sources for Fast External Interrupts (EXxINT)** as defined by the EXISEL register. If selected, transitions on receive lines of serial interface controllers (ASC, SSC, USB) can be used for wakeup from Sleep mode.
- **RTC Timer T14 cyclic interrupt.** For waking up from Sleep mode via RTC T14 interrupt, the RTC operation during Sleep mode must be selected in bitfield SLEEPCON within register SYSCON1. Additionally, the RTC interrupt must be enabled in the Interrupt Subnode Control register ISNC.
- **RTC interrupt(s)**. With new real time clock, additional RTC interrupts can be enabled via the Interrupt Subnode Control register RTCISNC. For wakeup, RTC operation during Sleep mode must be selected in SYSCON1. This function is not supported in C167CS.
- **Non-Maskable Interrupt NMI**. A high-to-low transition on $\overline{\text{NMI}}$ pin always terminates the Sleep mode. The $\overline{\text{NMI}}$ input is filtered for spike suppression. (Planned: Input signals shorter than 10ns are suppressed, detection is guaranteed for minimum 150ns $\overline{\text{NMI}}$ signal).

Setup Lengthening Control (Start Delay)

Contrary to Idle mode, after wakup from Sleep mode at first the ramp-up of clock system (oscillator and PLL) has to be controlled before any CPU operation can be started. Only when the clock system is locked on configured frequency, the following processing is identical to wakeup from Idle mode.

Note: This setup lengthening function is very similar to the start delay after HW-reset because of reset lengthening conditions (see reset section). Setup lengthening uses the same lengthening control signals as reset lengthening, but after setup the program start is controlled with the trailing edge of a setup-active signal (contrary to the RST signal in case of reset lengthening) which is provided to the core to delay the execution of first instruction. The system hold state during setup is controlled by start delay of clock distribution.

20 System Control Unit (CSCU)

20.1 Introduction

System Control Unit CSCU is used to control system specific tasks such as reset control or power management (see previous Chapter "Power Reduction Modes") within an on-chip system build around the Infineons Cell-Based Core C166. The power management features of the CSCU provide effective means to realize standby conditions for the system with an optimum balance between power reduction, peripheral operation and system functionality. Additionally, the CSCU controls the modes and operation of Real Time Clock RTC.

Summary of Features and Functions

CSCU is characterized by the following functions:

- Central Control of system operation
- External interrupt and frequency output control
- Protection management for system control registers
- General XBUS peripherals control
- Control of visibility of XPERs
- Power management additional to the standard Idle and Power Down modes
- Sleep mode with wakeup from Power Down state by external interrupts
- Peripheral Management with individual clock and power control of peripherals
- Control of power down state of Flash modules during Idle
- Flexible clock generation management
- Programmable system Slow Down control with or without PLL
- Control interface for Clock Generation Unit
- Identification register block for chip and CSCU identification
- Device, revision, manufacturer
- CSCU identification register

20.2 Operational Overview

20.2.1 Overview of CSCU submodules

In the following paragraphs a functional overview of the different blocks and submodules of the System Control Unit is presented.

XBUS Peripheral Configuration Block

In the C161U, XBUS peripherals can be separately switched on or off by programming the XPERCON register. If switched off, the respective peripheral is not visible, meaning, that its address space and its functional pins are not occupied.

Note:

1. In parallel to the XPER control with XPERCON register, the visibility of XPER address spaces also is controlled with the BUSACT bits in respective XBCON registers (in the C166 core)
2. The XPER configuration is additionally controlled by means of flexible peripheral management control (see Peripheral Management Module below) for power reduction.

Register in XPER Configuration Block:

Register	Description
XPERCON	XBUS peripheral control of XPER visibility

System Control Block

This block has several system management functions.

System Control Block controls the system register write protection, introduced for the system control registers SYSCON1-3.

Note: The new register write protection especially supports modularity of design, and is therefore not compatible with the previously known C16x release function, using the release bitfield in SYSCON2 for write protection.

Additional control functions of the System Control Block:

- Control of fast external interrupt inputs
- Control of external interrupt source selection
- Control of interrupt subnode for PLL and realtime clock interrupts
- Control of spike suppression for fast external interrupts and NMI in Sleep mode
- Clock output frequency control

The System Control Block provides the following registers:

Register	Description
SCUSLC	SCU security level command register
SCUSLS	SCU security level status and password register
EXICON	External interrupt control register (see Chapter 7.8.1, page 127)
EXISEL	External interrupt source selection control register (see Chapter 7.8.2, page 128)
ISNC	Interrupt subnode control register (see Chapter 7.8.3, page 129)
FOCON	Frequency output control register

Identification Register Block

All new derivatives of Infineons C16x microcontroller family provide a set of min. four identification registers (expandable to eight). These registers offer information on the chip manufacturer, the chip type and its memory properties.

Identification registers in the ID Block:

Register	Description
IDMANUF	Manufacturer and department
IDCHIP	Identification of device and revision code
IDMEM	Identification of on-chip program memory (type, size)
IDPROG	Identification of programming/erasing voltage of on-chip program memory
IDMEM2	Identification of additional EEPROM, OTP, DRAM or Flash memory

20.3 XBUS Peripheral Configuration Block

The XBUS peripherals can be separately selected for being visible to the user by means of corresponding selection bits in the XPERCON register. If not selected and therefore not enabled (not activated with XPERCON bit), the peripheral's address space including SFR addresses and port pins are not occupied by the peripheral, thus the peripheral is not visible and not available. To make an XBUS peripheral visible, its related bit in XPERCON register must be set before the XPERs are globally enabled with XPEN-bit in SYSCON register (during system initialization before EINIT instruction).

Note: After reset, **no** XBUS peripheral is selected in XPERCON register.

XPERCON register is defined as follows:

XPERCON ($F024_H$ / 12_H) **ESFRReset Value : 0000_H**

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

Bit Field	Bits	Type	Value	Description
reserved	15..8	rw	0	These bits are reserved and must be set to '0'.
XPER7	7	rw	0 1	**EPEC** module is not visible EPEC is selected and visible
XPER6	6	rw	0 1	**USB** module is not visible USB is selected and visible
XPER5	5	rw	0	This bit is reserved and should be set to '0'.

Bit Field	Bits	Type	Value	Description
reserved	4..0	rw	0	These bits are reserved and must be set to Zero

Note: CSCU provides per XPERCON bit one enable signal for XPER visibility control. These enable signals are routed to the core to be combined with selectable (per module pins) BUSACT functions of XBCON registers.

20.4 System Control Block

20.4.1 Register Write Protection

System Control Unit CSCU provides two different protection types of registers:

- Unprotected Registers
- Protectable Registers

The unprotected registers allow reading and writing (if not read-only) of register values without any restrictions. However, the write access of the protectable registers (security registers) can be programmed for three different modes of security level, whereas the read access is always unprotected:

- Write Protected Mode
- Low Protected Mode
- Unprotected Mode

In write protected mode the registers can not be accessed by a write command. However in low protected mode the registers can be written with a special command sequence (see desription below). If the registers are set to unprotected mode, all write accesses are possible.

Some register controlled functions and modes which are critical for the C161U's operation are locked after the execution of EINIT, so these vital system functions cannot be changed inadvertently eg. by software errors. However, as these security registers control also the power management they need to be accessed during operation to select the appropriate mode.

The switching between the different security levels is controlled by a state machine. Via a password and a command sequence the security levels can be changed. After reset always the unprotected mode is automatically selected. The EINIT command switches the security level automatically to protected mode.

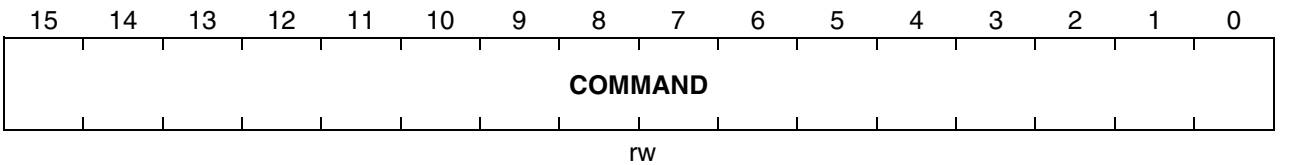
The low protected mode is especially important for a standby state of the application. This mode allows fast accesses within two commands to the protected registers without removing the protection completely.

Security Level Switching

Two registers are provided for switching the security level, the security level command register SCUSLC and the security level status register SCUSLS . The security level command register SCUSLC is used to control the state machine for switching the security level. SCUSLC register is loaded with the different commands of the command sequence necessary to control a change of the security level. It is also used for the one unlock command, which is necessary in the low protected mode to access one protected register. The commands of the (unlock) command sequence are characterized by certain pattern words (as $AAAA_H$) or by patterns combined with an 8-bit password. For command definition see the following state diagram (figure below). The new password is defined with command 3 and stored in the according 8-bit field in the SCUSLS register.

SCUSLC register is defined as follow

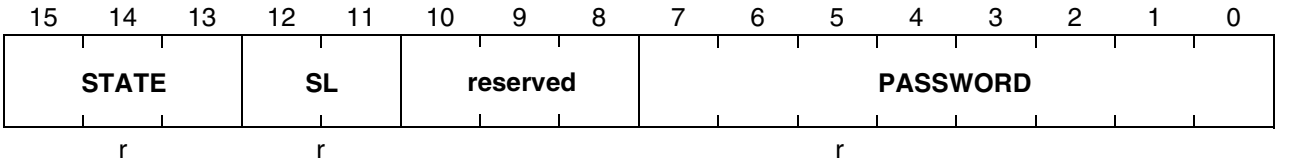
SCUSLC ($F0C0_H$ / 60_H) **ESFRReset Value: 0000_H**



The command definition is described in **Figure 118**.

The security level status register SCUSLS is a read only register which shows the current password, the actual security level and the state of the switching statemachine. The SCUSLS is defined as followes:

SCUSLS ($F0C2_H$ / 61_H) **ESFR Reset Value: 0000_H**

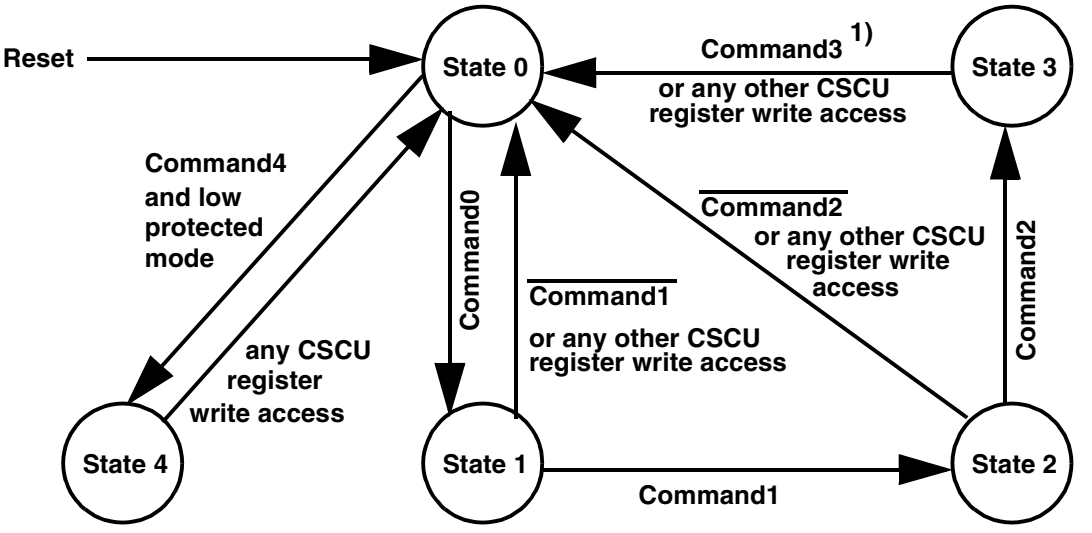


Bit	Function
PASSWORD	Current Password
SL	Security Level 0 0: Unprotected write mode 0 1: Low protected mode 1 0: Reserved 1 1: Write protected mode
STATE	Actual State 0 0 0: Wait for first command (command 0) 0 0 1: Wait for command 1 0 1 0: Wait for command 2 0 1 1: Wait for new security level and for new password (command 3) 1 0 0: Security registers are unlocked; access to one register is possible (only in low protected mode) 1 0 1: Reserved 1 1 0: Reserved 1 1 1: Reserved

The following registers are defined as protected (security) registers:

- SYSCON1
- SYSCON2
- SYSCON3

The following state diagram, **Figure 118**, shows the state machine for security level switching and for unlock command execution in low protected mode:



Command Number	Command
0	$AAAA_h$
1	5554_h
2	96_h + inverse (old) password
3	000_b + new level + 000_b + new password
4	$8E_h$ + inverse (new) password: Unlock security register in low protected mode

Note : 1)Only if the security level command register is accessed, the new security level and the new password is valid.

Figure 118 Statemachine for Security Level Switching

Write Access in Low Protected Mode

The write access in low protected mode is also done via a command sequence. First the specific command 4 (see figure above) with the current password has to be written to register SCUSLC. After this command all security registers are unlocked until the next write access to any CSCU register is done. Read access is always possible to all registers of the CSCU and will not influence the command sequences. In register SCUSCS the actual status of the command state machine can always be read.

It is recommendet to use an atomic sequence for all command sequences.

20.4.2 Clock Output Frequency Control

A clock output signal with programmable frequency (f_{OUT}) can be output via pin FOUT. This clock signal is generated via a reload counter, so the output frequency can be selected in small steps. An optional toggle latch provides a clock signal with a 50% duty cycle.

Signal f_{OUT} always provides complete output periods (see Signal Waveforms below):

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

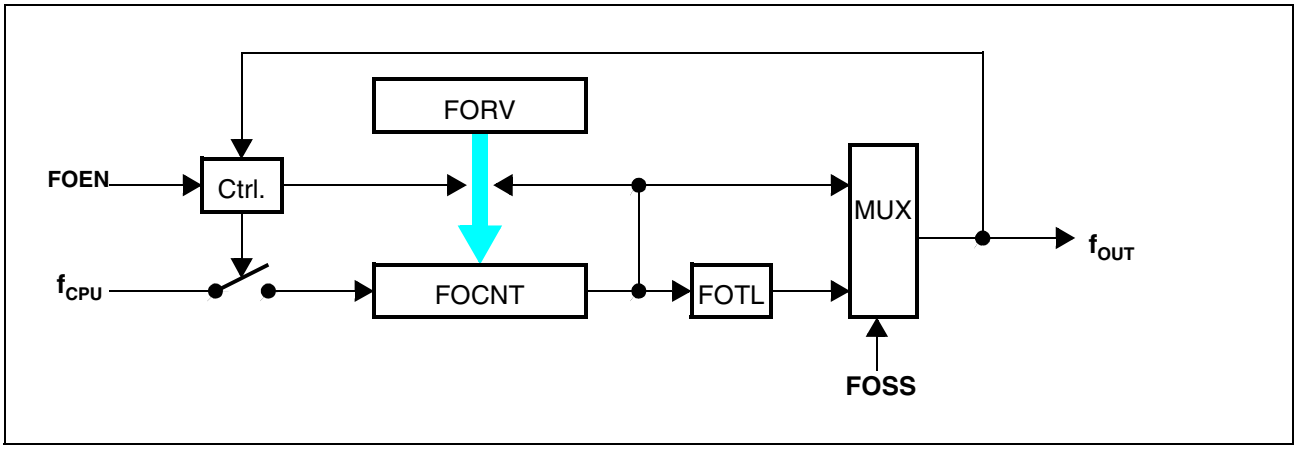


Figure 119 Clock Output Signal Generation

Register FOCON provides control over the output signal generation.

FOCON ($FFAA_H$ / $D5_H$) **SFR-b** **Reset Value: 0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FOEN	FOSS	FORV						-	FOTL	FOCNT					
rw	rw	rw						-	rw	rw					

Bit	Function
FOCNT	Frequency Output Counter
FOTL	Frequency Output Toggle Latch Is toggled upon each underflow of FOCNT.
FORV	Frequency Output Reload Value Is copied to FOCNT upon each underflow of FOCNT.

Bit	Function
FOSS	Frequency Output Signal Select 0: Output of the toggle latch: DC=50%. 1: Output of the reload counter: DC depends on FORV.
FOEN	Frequency Output Enable 0: Frequency output generation stops when signal f_{OUT} is/gets low. 1: FOCNT is running, f_{OUT} is gated to pin. 1st reload after 0-1 transition.

Note: It is not recommended to write to any part of bitfield FOCNT, especially not while the counter is running. Writing to FOCNT prior to starting the counter is obsolete because it will immediatley be reloaded from FORV. Writing to FOCNT during operation may produce uninteded counter values.

Output Frequency Calculation

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

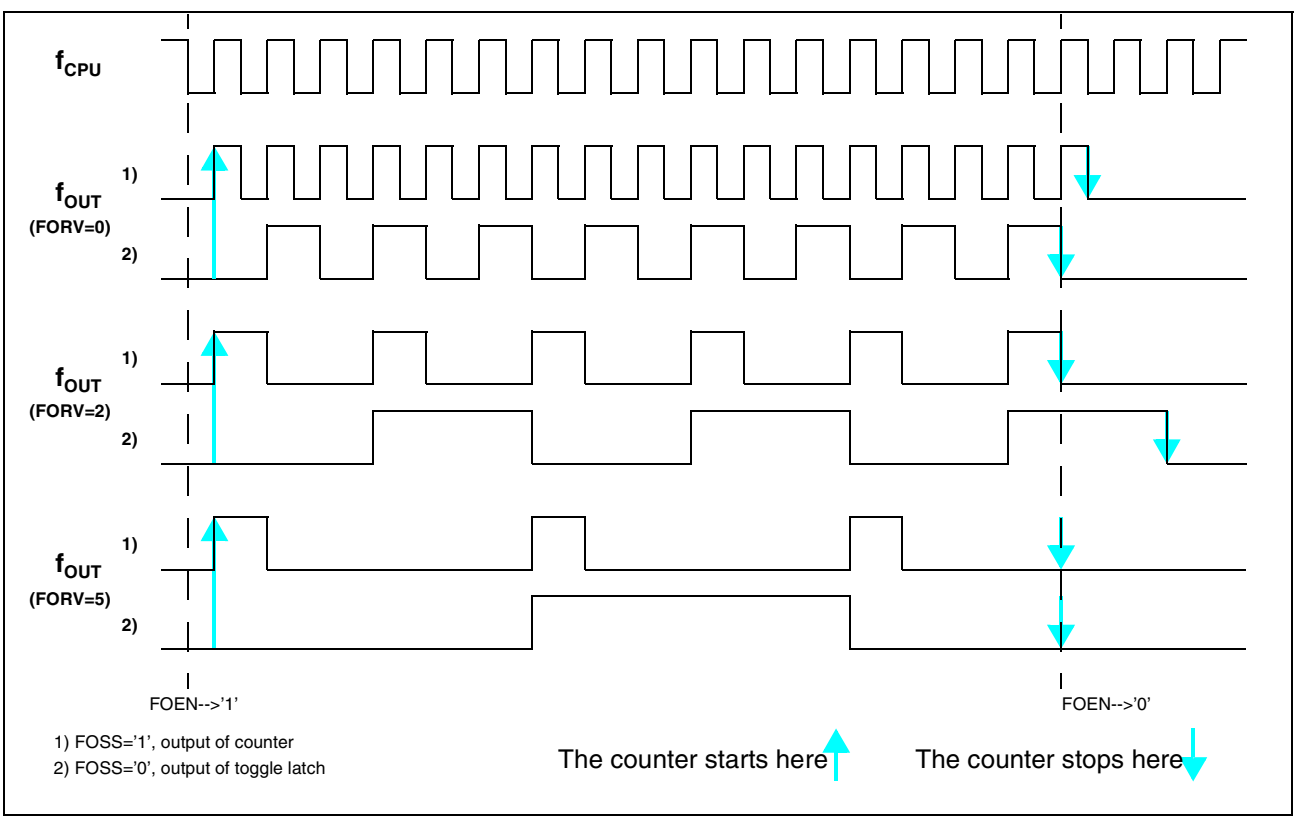


Figure 120 Signal Waveforms

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

Connection to Output

Signal f_{OUT} in the C161U is an alternate function of pin P3.15/CLKOUT/FOUT.
The priority ranking is: P3.15 < FOUT < CLKOUT.

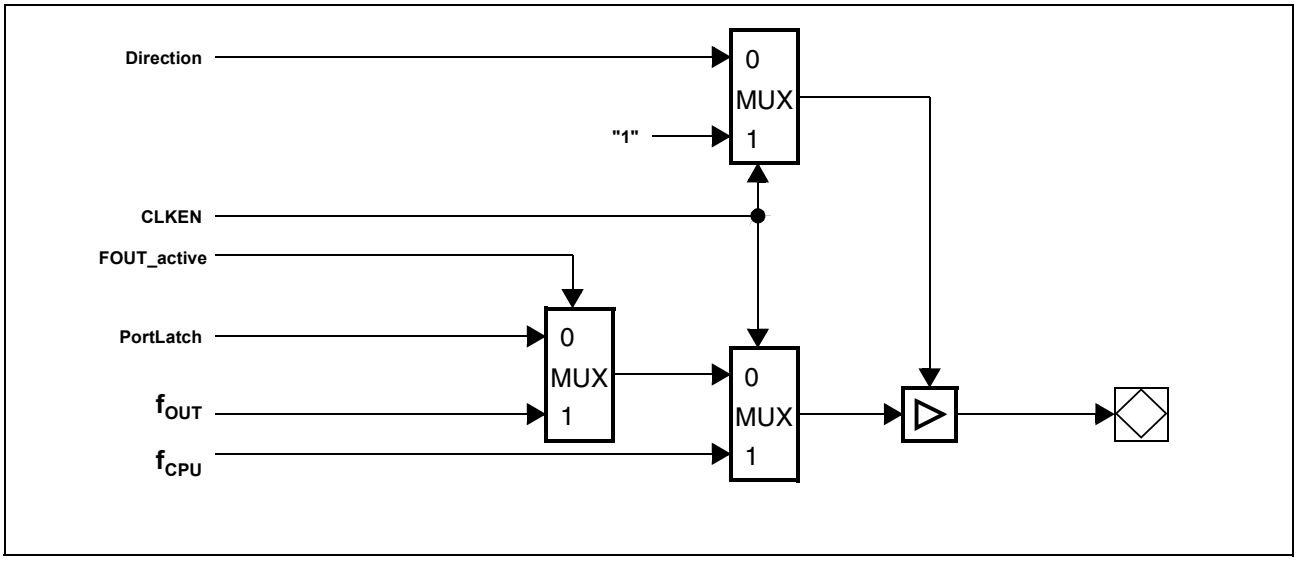


Figure 121 Connection to Port Logic (Functional Approach)

Note: For the generation of f_{OUT} pin FOUT must be switched to output, ie. DP3.15='1'. While f_{OUT} is disabled the pin is controlled by the port latch (see figure above). The port latch P3.15 must be '0' in order to maintain the fOUT inactive level on the pin. Clock Management Module

Flexible Clock Management

This module especially serves for power management support. Flexible clock management includes programmable system slow down with additional control of power down and optional real time clock. The 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. The PLL may be completely switched off in this mode.

This module also controls the oscillator selection (main or auxiliary) for Real Time Clock and for Slow Down Divider. During Power Down mode, this block controls the operation of RTC and ports.

The clock generation is controlled via register SYSCON2.

SYSCON2 ($F1D0_H$ / $E8_H$) ESFR-b Reset Value: $0000.0000.UU00.0000_B$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CLK LOCK	CLKREL					CLKCON		SCS	RCS	PDCON		reserved			
r	rw					rw		rw	rw	rw		rw			

Bit	Function
PDCON	Power Down Control (during power down mode) x0: RTC = Off, Ports = On (default after reset). x1: RTC = Off, Ports = Off. In power down mode, the RTC of the C161U is always off. Bit 5 of SYSCON2 is don't care.
RCS	RTC Clock Source (not affected by a reset) 0: RTC is switched to synchronous mode. The input is derived from the CPU clock. 1: RTC is switched to asynchronous mode. The input is derived from the RTC_REF_CLK (oscillator clock).
SCS	SDD Clock Source (not affected by a reset) Has to be set to '0'.
CLKCON	Clock State Control 00: Running on configured basic frequency. 01: Running on slow down frequency, PLL ON. 10: Running on slow down frequency, PLL OFF. 11: Reserved. Do not use this combination.
CLKREL	Reload Counter Value for Slowdown Divider
CLKLOCK	Clock Signal Status Bit 0: Main oscillator is unstable or PLL is unlocked. 1: Main oscillator is stable **and** PLL is locked.

Note: SYSCON2 is a security register. The security level is automatically set to write protectection after execution of EINIT.

Note: To be **compatibile to Infineon's C167CR / 167CS**, the Power Down Control PDCON must be programmed to '10' (RTC = Off, Ports = On) during the initialisation phase before the execution of EINIT instruction. The initial state after reset is so defined that a reset does not interrupt the real time clock.

20.5 Peripheral Management Module

This module especially serves for power management support, controlling dynamically the operation and thus the power consumption of the different peripherals on PD Bus and XBUS. In each situation (eg. 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.

Peripheral's operation is disabled or enabled by controlling the specific clock input. This function also is supported in idle and/or slow down mode.

The Real Time Clock (RTC) may be fed by a separate clock driver, so it can be kept running even in power down mode.

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

Note: In contrast to the peripheral management of Infineon's 16x family the registers of a **disabled** module are not accessable. Only the clock control register of the platform peripheral is accessable. Note, the register access is not compatible to the C167CS.

The user gets access to the flexible operation control of peripherals via the SYSCON3 register. This register is defined as follows:

SYSCON3 ($F1D4_H$ / EA_H) **ESFR-b** **Reset Value:0000_H**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GRP DIS	re-served	PLL DIS	USBTDIS		reserved		PER DIS8	PER DIS7	PER DIS6	reserved		PER DIS3	PER DIS2	PER DIS1	PER DIS0
rw	-	rw	rw		-	-	rw	rw	rw	-	-	rw	rw	rw	rw

.

Bit	Function
PERDISx	Peripheral Disable Flag 0 - 14 ‘0’: Module is enabled; the peripheral is supplied with the clock signal ‘1’: Module is disabled; the clock input of peripheral is disabled
GRPDIS	Peripheral Group Disable Flag (PD-Bus and X-Bus Peripherals) ‘0’: Peripheral clock driver for peripheral group is enabled ‘1’: Peripheral clock driver for peripheral group is disabled
USBTDIS	USB Transceiver Disable Flag ONLY IF BIT XPERCON.6 = ’1’ ‘00’: Normal operation, USB transceiver enabled ‘01’: Suspend mode, differencial transceiver switched off ‘10’: Reserved, do not use this combination. ‘11’: Full power down. If bit 6 of register XPERCON set to ’0’, the USB transceiver is always switched off (power down mode), independently of bit USBTDIS.
PLLDIS	PLL Disable Flag (additional power savings / noise reduction feature) ‘0’: The PLL of the C161U is switched on. This is the **default** configuration. ‘1’: The PLL is completely switched off. The free running feature and the oscillator watchdog will not work, since there is no PLL clock at all. It makes sense to switch off the PLL in **direct drive** clock mode only.

Note: Please refer to Chapter 10.8, "Initialization of the C161U’s X-peripherals", for complete register initialization.

Note: SYSCON3 is an security register. The security level is automatically set to write protection after execution of EINIT

PERDISx	Module Type	Module	**Function** (examples for associated peripheral modules)
0	PD-Bus Unit	RTC	Real Time Clock
1	PD-Bus Unit	ASC	USART

PERDISx	Module Type	Module	**Function** (examples for associated peripheral modules)
2	PD-Bus Unit	SSC	Synchronous Serial Channel
3	PD-Bus Unit	GPT12	General Purpose Timer Block
4..5	reserved	-	Reserved, has to be set to ’0’.
6	reserved	-	Reserved, has to be set to ’0’.
7	X-Bus Unit	USB	Universal Serial Bus Interface
8	X-Bus Unit	EPEC	Extended PEC
9..14	reserved	-	Reserved, has to be set to ’0’.

20.6 Identification Registers

20.6.1 Introduction

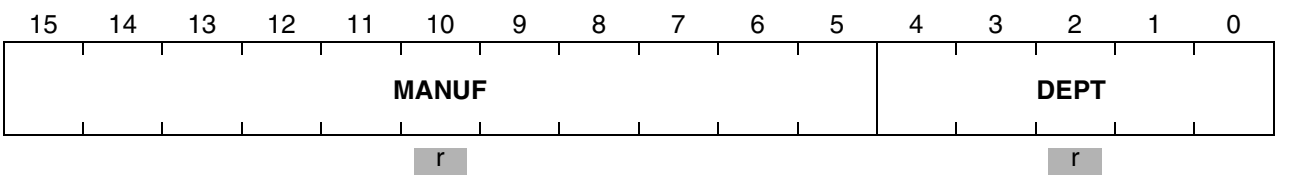
The C161U provides a set of 5 identification registers that offer information on the chip, as manufacturer, chip type and its memory properties.

The ID registers are read only registers. A device that incorporates ID registers shall return $D5_H$ as its Bootstrap Loader identification byte. A standardized routine may then be downloaded which sends the ID registers to the serial interface, so the host gets exact information about its partner.

20.6.2 ID Register Description

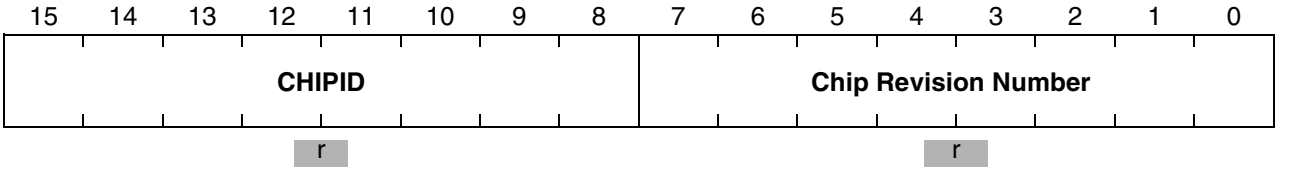
The ID registers are placed in the extended SFR area.

IDMANUF ($F07E_H$ / $3F_H$) **ESFR**



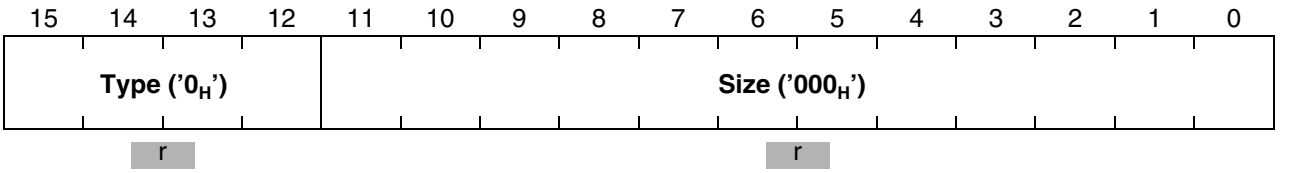
Bit	Function
MANUF	Manufacturer $0C1_H$: Infineon Technologies JEDEC normalized manufacturer code
DEPT	Department 04_H: Infineon’s Datacom Department

IDCHIP ($F07C_H$ / $3E_H$) **ESFR**



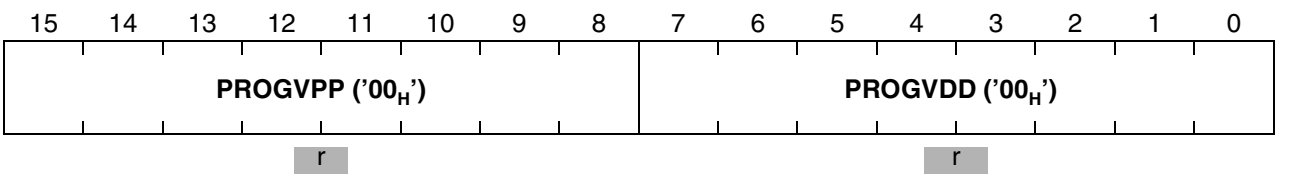
Bit	Function
Revision	Device Revision Code 03_H: actual device revision code
CHIPID	Device Identification 06_H: Infineons C16x device identification

IDMEM ($F07A_H$ / $3D_H$) **ESFR**



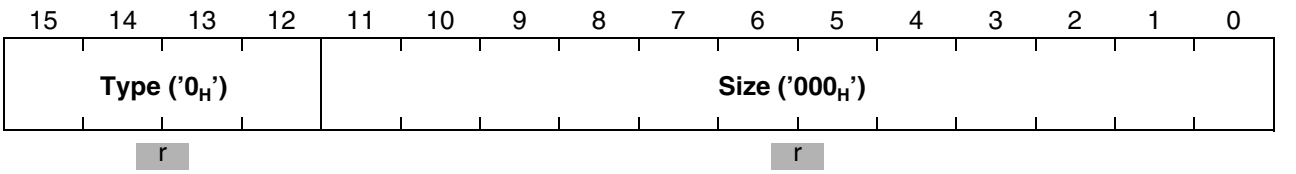
Bit	Function
Size	Size of on-chip Program Memory The size of the implemented program memory in terms of 4 K blocks, i.e. Memory-size = <Size>*4 KByte. **000_H: No program memory on the C161U**.
Type	Type of on-chip Program Memory Identifies the memory type on this silicon. **0_H: ROMless** 1_H: Mask ROM 2_H: EPROM 3_H: Flash 4_H: OTP 5_H: EEPROM 6_H: DRAM/SRAM

IDPROG ($F078_H$ / $3C_H$) **ESFR**



Bit	Function
PROGVDD	Programming V_{DD} Voltage The voltage of the standard power supply pins required when programming or erasing (if applicable) the on-chip program memory. Formula: V_{DD} = 20*<PROGVDD> / 256 [V] **00_H: No program memory on the C161U**.
PROGVPP	Programming V_{PP} Voltage The voltage of the special programming power supply (if existent) required to program or erase (if applicable) the on-chip program memory. Formula: V_{PP} = 20*<PROGVPP> / 256 [V] **00_H: No program memory on the C161U**.

IDMEM2 ($F076_H$ / $3B_H$) **ESFR**



Note: IDMEM2 describes the second block of (program) memory. This register is dedicated to other Infineon devices containing Flash, EEPROM or DRAM sections. Static RAM modules are not described with ID registers. Since there is no program memory on the C161U, IDMEM2 is set to '0000_H'.

21 System Programming

To aid in software development, a number of features has been incorporated into the instruction set of the C161U, 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 C161U. 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 C161U:

Substituted Instruction		C161U 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. C161U 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 (eg. 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 C161U. 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.

21.1 Stack Operations

C161U 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, ie. $00'F000_H$ 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, ie. 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.

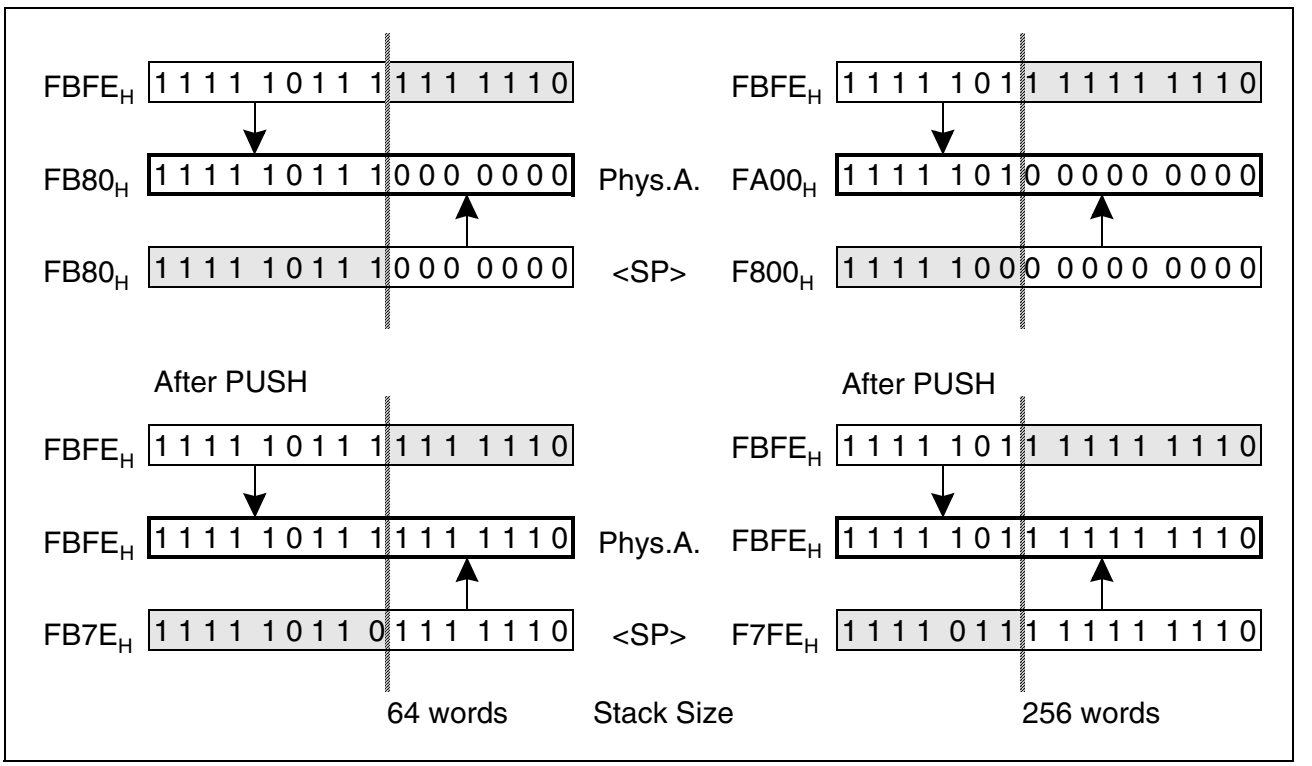


Figure 122 Physical Stack Address Generation

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 (ie. 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 meptied the bottom of stack is reloaded from the external memory and the internal pointers are adjusted accordingly.

Linear Stack

C161U 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+] or Rw, [Rw+]: 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.

21.2 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.

21.3 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.

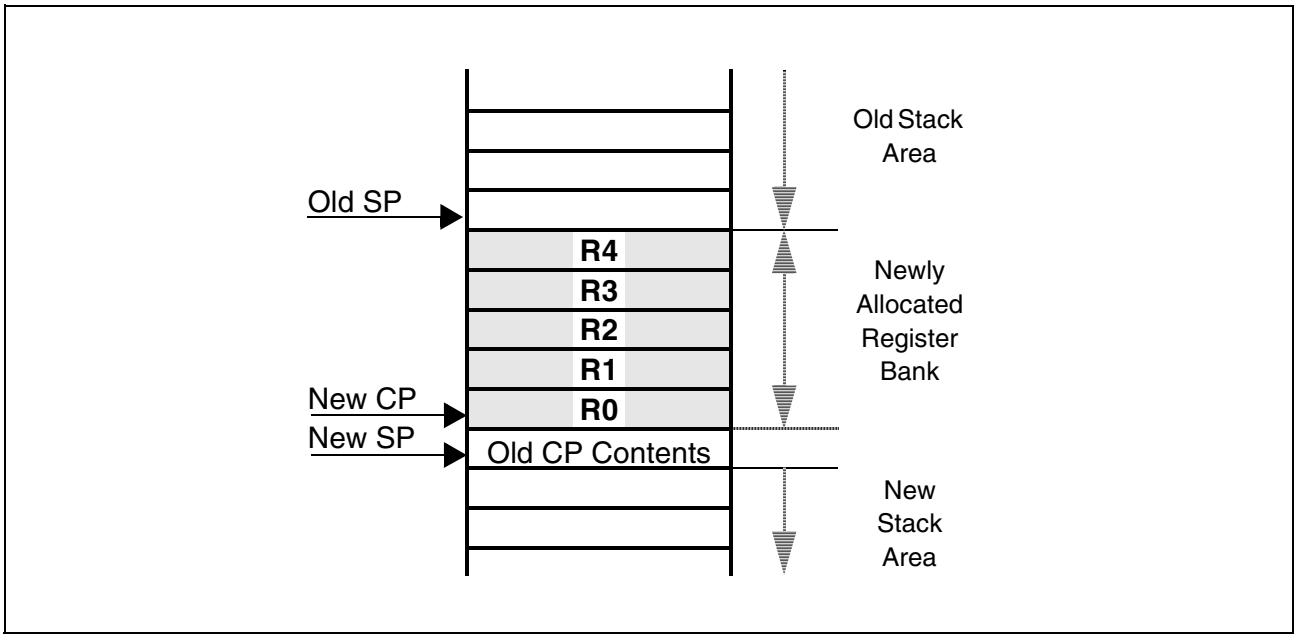


Figure 123 Local Registers

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
```

21.4 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
```

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).

21.5 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 C161U 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.

21.6 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.

21.7 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.

21.8 Unseparable Instruction Sequences

The instructions of the C161U 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 (eg. 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, ie. 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 (eg. 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
```

21.9 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.

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!
```

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 C161U (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.

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.

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.

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 (eg.) external memory, in order to free the address space occupied by the internal code memory, which is now unnecessary.

21.10 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 C161U 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.

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.

22 Register Set

This section summarizes all registers, which are implemented in the C161U 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.

22.1 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

22.2 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	Physic. Addr.	8-Bit Addr.	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	Physic. Addr.	8-Bit Addr.	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

22.3 Special Function Registers ordered by Address

The following table lists all SFRs which are implemented in the C161U ordered by physical address. Bit-addressable SFRs are marked with the letter “**b**” in column “Type”. SFRs within the Extended SFR-Space (ESFRs) are marked with the letter “**E**” in column “Type”.

Table 92 Registers ordered by Address

Physi. Addr	Register Name	Type	8-bit Addr	Description	Reset Value
	R0	SFR-b	$F0_H$	General Purpose Register 0	$UUUU_H$
	R0	ESFR-b	$F0_H$	General Purpose Register 0	$UUUU_H$
	R1	SFR-b	$F1_H$	General Purpose Register 1	$UUUU_H$
	R1	ESFR-b	$F1_H$	General Purpose Register 1	$UUUU_H$
	R10	ESFR-b	FA_H	General Purpose Register 10	$UUUU_H$
	R10	SFR-b	FA_H	General Purpose Register 10	$UUUU_H$
	R11	SFR-b	FB_H	General Purpose Register 11	$UUUU_H$
	R11	ESFR-b	FB_H	General Purpose Register 11	$UUUU_H$
	R12	SFR-b	FC_H	General Purpose Register 12	$UUUU_H$
	R12	ESFR-b	FC_H	General Purpose Register 12	$UUUU_H$
	R13	SFR-b	FD_H	General Purpose Register 13	$UUUU_H$
	R13	ESFR-b	FD_H	General Purpose Register 13	$UUUU_H$
	R14	SFR-b	FE_H	General Purpose Register 14	$UUUU_H$
	R14	ESFR-b	FE_H	General Purpose Register 14	$UUUU_H$
	R15	ESFR-b	FF_H	General Purpose Register 15	$UUUU_H$
	R15	SFR-b	FF_H	General Purpose Register 15	$UUUU_H$
	R2	SFR-b	$F2_H$	General Purpose Register 2	$UUUU_H$
	R2	ESFR-b	$F2_H$	General Purpose Register 2	$UUUU_H$
	R3	SFR-b	$F3_H$	General Purpose Register 3	$UUUU_H$
	R3	ESFR-b	$F3_H$	General Purpose Register 3	$UUUU_H$
	R4	ESFR-b	$F4_H$	General Purpose Register 4	$UUUU_H$
	R4	SFR-b	$F4_H$	General Purpose Register 4	$UUUU_H$
	R5	ESFR-b	$F5_H$	General Purpose Register 5	$UUUU_H$
	R5	SFR-b	$F5_H$	General Purpose Register 5	$UUUU_H$

Physi. Addr	Register Name	Type	8-bit Addr	Description	Reset Value
	R6	ESFR-b	$F6_H$	General Purpose Register 6	$UUUU_H$
	R6	SFR-b	$F6_H$	General Purpose Register 6	$UUUU_H$
	R7	ESFR-b	$F7_H$	General Purpose Register 7	$UUUU_H$
	R7	SFR-b	$F7_H$	General Purpose Register 7	$UUUU_H$
	R8	SFR-b	$F8_H$	General Purpose Register 8	$UUUU_H$
	R8	ESFR-b	$F8_H$	General Purpose Register 8	$UUUU_H$
	R9	SFR-b	$F9_H$	General Purpose Register 9	$UUUU_H$
	R9	ESFR-b	$F9_H$	General Purpose Register 9	$UUUU_H$
$F014_H$	XADRS1	ESFR	$0A_H$	XBUS Address Select Register 1	
$F016_H$	XADRS2	ESFR	$0B_H$	XBUS Address Select Register 2	
$F018_H$	XADRS3	ESFR	$0C_H$	XBUS Address Select Register 3	
$F01A_H$	XADRS4	ESFR	$0D_H$	XBUS Address Select Register 4	
$F01C_H$	XADRS5	ESFR	$0E_H$	XBUS Address Select Register 5	
$F01E_H$	XADRS6	ESFR	$0F_H$	XBUS Address Select Register 6	
$F024_H$	XPERCON	ESFR	12_H	XBUS Peripheral Control Register	0401_H
$F076_H$	IDMEM2	ESFR	$3B_H$	Identifier	0000_H
$F078_H$	IDPROG	ESFR	$3C_H$	Identifier	0000_H
$F07A_H$	IDMEM	ESFR	$3D_H$	Identifier	0000_H
$F07C_H$	IDCHIP	ESFR	$3E_H$	Identifier	0603_H
$F07E_H$	IDMANUF	ESFR	$3F_H$	Identifier	1824_H
$F0B0_H$	SSCTB	ESFR	58_H	SSC Transmit Buffer (WO)	0000_H
$F0B2_H$	SSCRB	ESFR	59_H	SSC Receive Buffer (RO)	$xxxx_H$
$F0B4_H$	SSCBR	ESFR	$5A_H$	SSC Baudrate Register	0000_H
$F0B6_H$	SSCCLC	ESFR	$5B_H$	SSC Clock Control Register	0000_H
$F0C0_H$	SCUSLC	ESFR	60_H	Security Level Control Register	0000_H
$F0C2_H$	SCUSLS	ESFR	61_H	Security Level Status Register	0000_H
$F0C8_H$	RTCCLC	ESFR	64_H	RTC Clock Control Register	0000_H
$F0CC_H$	RTCRELL	ESFR	66_H	RTC Timer Reload Register Low	0000_H
$F0CE_H$	RTCRELH	ESFR	67_H	RTC Timer Reload Register High	0000_H
$F0D0_H$	T14REL	ESFR	68_H	Timer 14 Reload Register	n_H

Physi. Addr	Register Name	Type	8-bit Addr	Description	Reset Value
$F0D2_H$	T14	ESFR	69_H	Timer 14 Register	n_H
$F0D4_H$	RTCL	ESFR	$6A_H$	RTC Timer Register Low	n_H
$F0D6_H$	RTCH	ESFR	$6B_H$	RTC Timer Register High	n_H
$F0D8_H$	DTIDR	ESFR	$6C_H$	Task ID register[1)]	0000_H
$F100_H$	DP0L	ESFR-b	80_H	P0L Direction Control Register	00_H
$F102_H$	DP0H	ESFR-b	81_H	P0H Direction Control Register	00_H
$F104_H$	DP1L	ESFR-b	82_H	P1L Direction Control Register	00_H
$F106_H$	DP1H	ESFR-b	83_H	P1H Direction Control Register	00_H
$F108_H$	RP0H	ESFR-b	84_H	System Startup Configuration Register (RO)	xx_H
$F114_H$	XBCON1	ESFR-b	$8A_H$	XBUS Control register 1: reserved	0000_H
$F116_H$	XBCON2	ESFR-b	$8B_H$	XBUS Control register 2: USB module	0000_H
$F118_H$	XBCON3	ESFR-b	$8C_H$	XBUS Control register 3: EPEC module	0000_H
$F11A_H$	XBCON4	ESFR-b	$8D_H$	XBUS Control register 4: reserved	0000_H
$F11C_H$	XBCON5	ESFR-b	$8E_H$	XBUS Control register 5: reserved	0000_H
$F11E_H$	XBCON6	ESFR-b	$8F_H$	XBUS Control register 6: reserved	0000_H
$F160_H$	UTD3IC	ESFR-b	$B0_H$	UDC TX Done3 Interrupt Control Register	0000_H
$F162_H$	UTD4IC	ESFR-b	$B1_H$	UDC TX Done4 Interrupt Control Register	0000_H
$F164_H$	UTD5IC	ESFR-b	$B2_H$	UDC TX Done5 Interrupt Control Register	0000_H
$F166_H$	UTD6IC	ESFR-b	$B3_H$	UDC TX Done6 Interrupt Control Register	0000_H
$F168_H$	UTD7IC	ESFR-b	$B4_H$	UDC TX Done7 Interrupt Control Register	0000_H
$F16A_H$	URXRIC	ESFR-b	$B5_H$	UDC RXRR Interrupt Control Register	0000_H
$F16C_H$	UTXRIC	ESFR-b	$B6_H$	UDC TXWR Interrupt Control Register	0000_H

Physi. Addr	Register Name	Type	8-bit Addr	Description	Reset Value
$F16E_H$	UCFGVIC	ESFR-b	$B7_H$	UDC Config Val Interrupt Control Register	0000_H
$F170_H$	USOFIC	ESFR-b	$B8_H$	UDC Start of Frame Interrupt Control Register	0000_H
$F172_H$	USSOIC	ESFR-b	$B9_H$	UDC Suspend off Interrupt Control Register	0000_H
$F174_H$	USSIC	ESFR-b	BA_H	UDC Suspend Interrupt Control Register	0000_H
$F176_H$	ULCDIC	ESFR-b	BB_H	UDC Load Config Done Interrupt Control Register	0000_H
$F178_H$	USETIC	ESFR-b	BC_H	UDC SETUP Interrupt Control Register	0000_H
$F17A_H$	URD0IC	ESFR-b	BD_H	UDC RX Done0 Interrupt Control Register	0000_H
$F17C_H$	EPECIC	ESFR-b	BE_H	EPEC Interrupt	0000_H
$F180_H$	PECCLIC	ESFR-b	$C0_H$	PEC Channel Link Interrupt Control Register	0000_H
$F184_H$	RTC_INTIC	ESFR-b	$C2_H$	RTC_INT Sub Node Interrupt Register	0000_H
$F186_H$	XP0IC	ESFR-b	$C3_H$	X-Bus Peripheral 0 UDC TXWR Interrupt Control Register	0000_H
$F18C_H$	ABENDIC	ESFR-b	$C6_H$	ASC Autobaud End Interrupt Control Register	0000_H
$F18E_H$	XP1IC	ESFR-b	$C7_H$	X-Bus Peripheral 1 EPEC Interrupt Control Register	0000_H
$F194_H$	ABSTIC	ESFR-b	CA_H	ASC Autobaud Start Interrupt Control Register	0000_H
$F19A_H$	RES6IC	ESFR-b	CD_H	reserved	0000_H
$F19C_H$	S0TBIC	ESFR-b	CE_H	Serial Channel 0 Transmit Buffer IC Register	0000_H
$F19E_H$	XP3IC	ESFR-b	CF_H	X-Bus Peripheral 3 PLL/RTC Interrupt Control Register	0000_H
$F1C0_H$	EXICON	ESFR-b	$E0_H$	External Interrupt Control Register	0000_H
$F1C2_H$	ODP2	ESFR-b	$E1_H$	Port 2 Open Drain Control Register	0000_H

Physi. Addr	Register Name	Type	8-bit Addr	Description	Reset Value
$F1C6_H$	ODP3	ESFR-b	$E3_H$	Port 3 Open Drain Control Register	0000_H
$F1C8_H$	RTCISNC	ESFR-b	$E4_H$	RTC Interrupt Sub Node Control Register	0000_H
$F1CA_H$	ODP4	ESFR-b	$E5_H$	Port 4 Open Drain Control Register	00_H
$F1CC_H$	RTCCON	ESFR-b	$E6_H$	RTC Control Register	00_H
$F1CE_H$	ODP6	ESFR-b	$E7_H$	Port 6 Open Drain Control Register	00_H
$F1D0_H$	SYSCON2	ESFR-b	$E8_H$	System Configuration Register 2/ Clock Control	0000_H
$F1D4_H$	SYSCON3	ESFR-b	EA_H	System Configuration Register 3/ Periph. Managem.	0000_H
$F1D6_H$	reserved	ESFR-b	EB_H	reserved - do not use	0000_H
$F1D8_H$	reserved	ESFR-b	EC_H	reserved - do not use	0000_H
$F1DA_H$	EXISEL	ESFR-b	ED_H	External Interrupt Select Register	0000_H
$F1DC_H$	SYSCON1	ESFR-b	EE_H	System Configuration Register 1/ Sleep Mode	0000_H
$F1DE_H$	ISNC	ESFR-b	EF_H	Interrupt Sub Node Control Register	0000_H
$FE00_H$	DPP0	SFR	00_H	CPU Data Page Pointer 0 Register (10 bits)	0000_H
$FE02_H$	DPP1	SFR	01_H	CPU Data Page Pointer 1 Register (10 bits)	0001_H
$FE04_H$	DPP2	SFR	02_H	CPU Data Page Pointer 2 Register (10 bits)	0002_H
$FE06_H$	DPP3	SFR	03_H	CPU Data Page Pointer 3 Register (10 bits)	0003_H
$FE08_H$	CSP	SFR	04_H	CPU Code Segment Pointer Register (8 bits)	0000_H
$FE0A_H$	EMUCON	SFR	05_H	Emulation Control Register[2)]	$xxxx_H$
$FE0C_H$	MDH	SFR	06_H	CPU Multiply Divide Register - High Word	0000_H
$FE0E_H$	MDL	SFR	07_H	CPU Multiply Divide Register - Low Word	0000_H
$FE10_H$	CP	SFR	08_H	CPU Context Pointer Register	$FC00_H$

Physi. Addr	Register Name	Type	8-bit Addr	Description	Reset Value
$FE12_H$	SP	SFR	09_H	CPU System Stack Pointer Register	$FC00_H$
$FE14_H$	STKOV	SFR	$0A_H$	CPU Stack Overflow Pointer Register	$FA00_H$
$FE16_H$	STKUN	SFR	$0B_H$	CPU Stack Underflow Pointer Register	$FC00_H$
$FE18_H$	ADDRSEL1	SFR	$0C_H$	Address Select Register 1	0000_H
$FE1A_H$	ADDRSEL2	SFR	$0D_H$	Address Select Register 2	0000_H
$FE1C_H$	ADDRSEL3	SFR	$0E_H$	Address Select Register 3	0000_H
$FE1E_H$	ADDRSEL4	SFR	$0F_H$	Address Select Register 4	0000_H
$FE22_H$	ODP0H	SFR	11_H	Port 0 Open Drain Control Register High	0000_H
$FE24_H$	ODP1L	SFR	12_H	Port 1 Open Drain Control Register Low	0000_H
$FE26_H$	ODP1H	SFR	13_H	Port 1 Open Drain Control Register High	0000_H
$FE40_H$	T2	SFR	20_H	GPT1 Timer 2 Register	0000_H
$FE42_H$	T3	SFR	21_H	GPT1 Timer 3 Register	0000_H
$FE44_H$	T4	SFR	22_H	GPT1 Timer 4 Register	0000_H
$FE46_H$	T5	SFR	23_H	GPT2 Timer 5 Register	0000_H
$FE48_H$	T6	SFR	24_H	GPT2 Timer 6 Register	0000_H
$FE4A_H$	CAPREL	SFR	25_H	GPT1/2 Capture / Reload Register	0000_H
$FE4C_H$	GPTCLC	SFR	26_H	GPT1/2 Clock Control Register	0000_H
$FE60_H$	P0LPUDSEL	SFR	30_H	Port 0 Low Pull-Up/Down Select Register	$xxFF_H$
$FE62_H$	P0HPUDSEL	SFR	31_H	Port 0 High Pull-Up/Down Select Register	$xxFF_H$
$FE64_H$	P0LPUDEN	SFR	32_H	Port 0 Low Pull Switch On/Off Register	xxFF
$FE66_H$	P0HPUDEN	SFR	33_H	Port 0 High Pull Switch On/Off Register	$xxFF_H$
$FE68_H$	P0LPHEN	SFR	34_H	Port 0 Low Pin Hold Enable Register	0000_H

Physi. Addr	Register Name	Type	8-bit Addr	Description	Reset Value
$FE6A_H$	P0HPHEN	SFR	35_H	Port 0 High Pin Hold Enable Register	0000_H
$FE6C_H$	P1LPUDSEL	SFR	36_H	Port 1 Low Pull-Up/Down Select Register	0000_H
$FE6E_H$	P1HPUDSEL	SFR	37_H	Port 1 High Pull-Up/Down Select Register	0000_H
$FE70_H$	P1LPUDEN	SFR	38_H	Port 1 Low Pull Switch On/Off Register	0000_H
$FE72_H$	P1HPUDEN	SFR	39_H	Port 1 High Pull Switch On/Off Register	0000_H
$FE74_H$	P1LPHEN	SFR	$3A_H$	Port 1 Low Pin Hold Enable Register	0000_H
$FE76_H$	P1HPHEN	SFR	$3B_H$	Port 1 High Pin Hold Enable Register	0000_H
$FE78_H$	P2PUDSEL	SFR	$3C_H$	Port 2 Pull-Up/Down Select Register	0000_H
$FE7A_H$	P2PUDEN	SFR	$3D_H$	Port 2 Pull Switch On/Off Register	0000_H
$FE7C_H$	P2PHEN	SFR	$3E_H$	Port 2 Pin Hold Enable Register	0000_H
$FE7E_H$	P3PUDSEL	SFR	$3F_H$	Port 3 Pull-Up/Down Select Register	0000_H
$FE80_H$	P3PUDEN	SFR	40_H	Port 3 Pull Switch On/Off Register	0000_H
$FE82_H$	P3PHEN	SFR	41_H	Port 3 Pin Hold Enable Register	0000_H
$FE84_H$	P4PUDSEL	SFR	42_H	Port 4 Pull-Up/Down Select Register	0000_H
$FE86_H$	P4PUDEN	SFR	43_H	Port 4 Pull Switch On/Off Register	0000_H
$FE88_H$	P4PHEN	SFR	44_H	Port 4 Pin Hold Enable Register	0000_H
$FE90_H$	P6PUDSEL	SFR	48_H	Port 6 Pull-Up/Down Select Register	0000_H
$FE92_H$	P6PUDEN	SFR	49_H	Port 6 Pull Switch On/Off Register	0000_H
$FE94_H$	P6PHEN	SFR	$4A_H$	Port 6 Pin Hold Enable Register	0000_H
$FEAA_H$	S0PMW	SFR	55_H	ASC IrDA PMW Control Register	0000_H
$FEAE_H$	WDT	SFR	57_H	Watchdog Timer Register (RO)	0000_H

Physi. Addr	Register Name	Type	8-bit Addr	Description	Reset Value
$FEB0_H$	S0TBUF	SFR	58_H	Serial Channel 0 Transmit Buffer Register (WO)	0000_H
$FEB2_H$	S0RBUF	SFR	59_H	Serial Channel 0 Receive Buffer Register (RO)	$xxxx_H$
$FEB4_H$	S0BG	SFR	$5A_H$	Serial Channel 0 Baud Rate Generator Reload Register	0000_H
$FEB6_H$	S0FDV	SFR	$5B_H$	ASC Fractional Divide Register	0000_H
$FEC0_H$	PECC0	SFR	60_H	PEC Channel 0 Control Register	0000_H
$FEC2_H$	PECC1	SFR	61_H	PEC Channel 1 Control Register	0000_H
$FEC4_H$	PECC2	SFR	62_H	PEC Channel 2 Control Register	0000_H
$FEC6_H$	PECC3	SFR	63_H	PEC Channel 3 Control Register	0000_H
$FEC8_H$	PECC4	SFR	64_H	PEC Channel 4 Control Register	0000_H
$FECA_H$	PECC5	SFR	65_H	PEC Channel 5 Control Register	0000_H
$FECC_H$	PECC6	SFR	66_H	PEC Channel 6 Control Register	0000_H
$FECE_H$	PECC7	SFR	67_H	PEC Channel 7 Control Register	0000_H
$FED0_H$	PECSN0	SFR	68_H	PEC Segment No Register	
$FED2_H$	PECSN1	SFR	69_H	PEC Segment No Register	
$FED4_H$	PECSN2	SFR	$6A_H$	PEC Segment No Register	
$FED6_H$	PECSN3	SFR	$6B_H$	PEC Segment No Register	
$FED8_H$	PECSN4	SFR	$6C_H$	PEC Segment No Register	
$FEDA_H$	PECSN5	SFR	$6D_H$	PEC Segment No Register	
$FEDC_H$	PECSN6	SFR	$6E_H$	PEC Segment No Register	
$FEDE_H$	PECSN7	SFR	$6F_H$	PEC Segment No Register	
$FEF0_H$	PECXC0	SFR	78_H	PEC Channel 0 Extended Control Register	
$FEF2_H$	PECXC2	SFR	79_H	PEC Channel 2 Extended Control Register	
$FEF8_H$	ABS0CON	SFR	$7C_H$	ASC Autobaud Control Register	0000_H
$FEFE_H$	ABSTAT	SFR	$7F_H$	ASC Autobaud Status Register	0000_H
$FF00_H$	P0L	SFR-b	80_H	Port 0 Low Register (Lower half)	00_H
$FF02_H$	P0H	SFR-b	81_H	Port 0 High Register (Upper half)	00_H
$FF04_H$	P1L	SFR-b	82_H	Port 1 Low Register (Lower half)	00_H

Physi. Addr	Register Name	Type	8-bit Addr	Description	Reset Value
$FF06_H$	P1H	SFR-b	83_H	Port 1 High Register (Upper half)	00_H
$FF0C_H$	BUSCON0	SFR-b	86_H	Bus Configuration Register 0	0000_H
$FF0E_H$	MDC	SFR-b	87_H	CPU Multiply Divide Control Register	0000_H
$FF10_H$	PSW	SFR-b	88_H	CPU Program Status Word	0000_H
$FF12_H$	SYSCON	SFR-b	89_H	CPU System Configuration Register	$0xx0_H$
$FF14_H$	BUSCON1	SFR-b	$8A_H$	Bus Configuration Register 1	0000_H
$FF16_H$	BUSCON2	SFR-b	$8B_H$	Bus Configuration Register 2	0000_H
$FF18_H$	BUSCON3	SFR-b	$8C_H$	Bus Configuration Register 3	0000_H
$FF1A_H$	BUSCON4	SFR-b	$8D_H$	Bus Configuration Register 4	0000_H
$FF1C_H$	ZEROS	SFR-b	$8E_H$	Constant Value 0sRegister'	0000_H
$FF1E_H$	ONES	SFR-b	$8F_H$	Constant Value 1sRegister'	$FFFF_H$
$FF40_H$	T2CON	SFR-b	$A0_H$	GPT1 Timer 2 Control Register	0000_H
$FF42_H$	T3CON	SFR-b	$A1_H$	GPT1 Timer 3 Control Register	0000_H
$FF44_H$	T4CON	SFR-b	$A2_H$	GPT1 Timer 4 Control Register	0000_H
$FF46_H$	T5CON	SFR-b	$A3_H$	GPT2 Timer 5 Control Register	0000_H
$FF48_H$	T6CON	SFR-b	$A4_H$	GPT2 Timer 6 Control Register	0000_H
$FF60_H$	T2IC	SFR-b	$B0_H$	GPT1 Timer 2 Interrupt Control Register	0000_H
$FF62_H$	T3IC	SFR-b	$B1_H$	GPT1 Timer 3 Interrupt Control Register	0000_H
$FF64_H$	T4IC	SFR-b	$B2_H$	GPT1 Timer 4 Interrupt Control Register	0000_H
$FF66_H$	T5IC	SFR-b	$B3_H$	GPT2 Timer 5 Interrupt Control Register	0000_H
$FF68_H$	T6IC	SFR-b	$B4_H$	GPT2 Timer 6 Interrupt Control Register	0000_H
$FF6A_H$	CRIC	SFR-b	$B5_H$	GPT2 CAPREL Interrupt Control Register	0000_H
$FF6C_H$	S0TIC	SFR-b	$B6_H$	Serial Channel 0 Transmit Interrupt Control Register	0000_H

Physi. Addr	Register Name	Type	8-bit Addr	Description	Reset Value
$FF6E_H$	S0RIC	SFR-b	$B7_H$	Serial Channel 0 Receive Interrupt Control Register	0000_H
$FF70_H$	S0EIC	SFR-b	$B8_H$	Serial Channel 0 Error Interrupt Control Register	0000_H
$FF72_H$	SSCTIC	SFR-b	$B9_H$	SSC Transmit Interrupt Control Register	0000_H
$FF74_H$	SSCRIC	SFR-b	BA_H	SSC Receive Interrupt Control Register	0000_H
$FF76_H$	SSCEIC	SFR-b	BB_H	SSC Error Interrupt Control Register	0000_H
$FF78_H$	URD3IC	SFR-b	BC_H	UDC RX Done3 Interrupt Control Register	0000_H
$FF7A_H$	URD4IC	SFR-b	BD_H	UDC RX Done4 Interrupt Control Register	0000_H
$FF7C_H$	URD5IC	SFR-b	BE_H	UDC RX Done5 Interrupt Control Register	0000_H
$FF7E_H$	URD6IC	SFR-b	BF_H	UDC RX Done6 Interrupt Control Register	0000_H
$FF80_H$	URD7IC	SFR-b	$C0_H$	UDC RX Done7 Interrupt Control Register	0000_H
$FF82_H$	UTD0IC	SFR-b	$C1_H$	UDC TX Done0 Interrupt Control Register	0000_H
$FF84_H$	UTD1IC	SFR-b	$C2_H$	UDC TX Done1 Interrupt Control Register	0000_H
$FF86_H$	UTD2IC	SFR-b	$C3_H$	UDC TX Done2 Interrupt Control Register	0000_H
$FF88_H$	FEI0IC	SFR-b	$C4_H$	Fast External Interrupt 0 Control Register	0000_H
$FF8A_H$	FEI1IC	SFR-b	$C5_H$	Fast External Interrupt 1 Control Register	0000_H
$FF98_H$	RES4IC	SFR-b	CB_H	reserved	0000_H
$FF9C_H$	URD2IC	SFR-b	CE_H	UDC RX Done2 Interrupt Control Register	0000_H

Physi. Addr	Register Name	Type	8-bit Addr	Description	Reset Value
$FF9E_H$	URD1IC	SFR-b	CF_H	UDC RX Done1 Interrupt Control Register	0000_H
$FFA8_H$	CLISNC	SFR-b	$D4_H$	The channel link interrupt subnode register	0000_H
$FFAA_H$	FOCON	SFR-b	$D5_H$	Frequency Output Control Register	0000_H
$FFAC_H$	TFR	SFR-b	$D6_H$	Trap Flag Register	0000_H
$FFAE_H$	WDTCON	SFR-b	$D7_H$	Watchdog Timer Control Register	$000x_H$
$FFB0_H$	S0CON	SFR-b	$D8_H$	Serial Channel 0 Control Register	0000_H
$FFB2_H$	SSCCON	SFR-b	$D9_H$	SSC Control Register	0000_H
$FFBA_H$	S0CLC	SFR-b	DD_H	ASC Clock Control Register	0000_H
$FFC0_H$	P2	SFR-b	$E0_H$	Port 2 Register	0000_H
$FFC2_H$	DP2	SFR-b	$E1_H$	Port 2 Direction Control Register	0000_H
$FFC4_H$	P3	SFR-b	$E2_H$	Port 3 Register	0000_H
$FFC6_H$	DP3	SFR-b	$E3_H$	Port 3 Direction Control Register	0000_H
$FFC8_H$	P4	SFR-b	$E4_H$	Port 4 Register (8 bits)	00_H
$FFCA_H$	DP4	SFR-b	$E5_H$	Port 4 Direction Control Register	00_H
$FFCC_H$	P6	SFR-b	$E6_H$	Port 6 Register (8 bits)	00_H
$FFCE_H$	DP6	SFR-b	$E7_H$	Port 6 Direction Control Register	00_H

1) The DTIDR register is a data register which is used by advanced real time operating systems to store the task ID of the active task. It is used for hardware trigger events in the OCDS.

2) The EMUCON register is a reserved test register and is not to be used by other software.

22.4 Special Function Registers ordered by Name

The following table lists all SFRs which are implemented in the C161U ordered by their name. **Bit-addressable** SFRs are marked with the letter "**b**" in column "Type".
SFRs within the **Extended SFR-Space** (ESFRs) are marked with the letter "**E**" in column "Type".

Table 93 Registers ordered by Name

Register Name	Physi. Addr	Type	8-bit Addr	Description	Reset Value
ABENDIC	$F18C_H$	ESFR-b	$C6_H$	ASC Autobaud End Interrupt Control Register	0000_H
ABS0CON	$FEF8_H$	SFR	$7C_H$	ASC Autobaud Control Register	0000_H
ABSTAT	$FEFE_H$	SFR	$7F_H$	ASC Autobaud Status Register	0000_H
ABSTIC	$F194_H$	ESFR-b	CA_H	ASC Autobaud Start Interrupt Control Register	0000_H
ADDRSEL1	$FE18_H$	SFR	$0C_H$	Address Select Register 1	0000_H
ADDRSEL2	$FE1A_H$	SFR	$0D_H$	Address Select Register 2	0000_H
ADDRSEL3	$FE1C_H$	SFR	$0E_H$	Address Select Register 3	0000_H
ADDRSEL4	$FE1E_H$	SFR	$0F_H$	Address Select Register 4	0000_H
BUSCON0	$FF0C_H$	SFR-b	86_H	Bus Configuration Register 0	0000_H
BUSCON1	$FF14_H$	SFR-b	$8A_H$	Bus Configuration Register 1	0000_H
BUSCON2	$FF16_H$	SFR-b	$8B_H$	Bus Configuration Register 2	0000_H
BUSCON3	$FF18_H$	SFR-b	$8C_H$	Bus Configuration Register 3	0000_H
BUSCON4	$FF1A_H$	SFR-b	$8D_H$	Bus Configuration Register 4	0000_H
CAPREL	$FE4A_H$	SFR	25_H	GPT1/2 Capture / Reload Register	0000_H
CLISNC	$FFA8_H$	SFR-b	$D4_H$	The channel link interrupt subnode register	0000_H
CP	$FE10_H$	SFR	08_H	CPU Context Pointer Register	$FC00_H$
CRIC	$FF6A_H$	SFR-b	$B5_H$	GPT2 CAPREL Interrupt Control Register	0000_H
CSP	$FE08_H$	SFR	04_H	CPU Code Segment Pointer Register (8 bits)	0000_H
DP0H	$F102_H$	ESFR-b	81_H	P0H Direction Control Register	00_H
DP0L	$F100_H$	ESFR-b	80_H	P0L Direction Control Register	00_H
DP1H	$F106_H$	ESFR-b	83_H	P1H Direction Control Register	00_H

Register Name	Physi. Addr	Type	8-bit Addr	Description	Reset Value
DP1L	$F104_H$	ESFR-b	82_H	P1L Direction Control Register	00_H
DP2	$FFC2_H$	SFR-b	$E1_H$	Port 2 Direction Control Register	0000_H
DP3	$FFC6_H$	SFR-b	$E3_H$	Port 3 Direction Control Register	0000_H
DP4	$FFCA_H$	SFR-b	$E5_H$	Port 4 Direction Control Register	00_H
DP6	$FFCE_H$	SFR-b	$E7_H$	Port 6 Direction Control Register	00_H
DPP0	$FE00_H$	SFR	00_H	CPU Data Page Pointer 0 Register (10 bits)	0000_H
DPP1	$FE02_H$	SFR	01_H	CPU Data Page Pointer 1 Register (10 bits)	0001_H
DPP2	$FE04_H$	SFR	02_H	CPU Data Page Pointer 2 Register (10 bits)	0002_H
DPP3	$FE06_H$	SFR	03_H	CPU Data Page Pointer 3 Register (10 bits)	0003_H
DTIDR	$F0D8_H$	ESFR	$6C_H$	Task ID register	0000_H
EMUCON	$FE0A_H$	SFR	05_H	Emulation Control Register	$xxxx_H$
EPECIC	$F17C_H$	ESFR-b	BE_H	EPEC Interrupt	0000_H
EXICON	$F1C0_H$	ESFR-b	$E0_H$	External Interrupt Control Register	0000_H
EXISEL	$F1DA_H$	ESFR-b	ED_H	External Interrupt Select Register	0000_H
FEI0IC	$FF88_H$	SFR-b	$C4_H$	Fast External Interrupt 0 Control Register	0000_H
FEI1IC	$FF8A_H$	SFR-b	$C5_H$	Fast External Interrupt 1 Control Register	0000_H
FOCON	$FFAA_H$	SFR-b	$D5_H$	Frequency Output Control Register	0000_H
GPTCLC	$FE4C_H$	SFR	26_H	GPT1/2 Clock Control Register	0000_H
IDCHIP	$F07C_H$	ESFR	$3E_H$	Identifier	0603_H
IDMANUF	$F07E_H$	ESFR	$3F_H$	Identifier	1824_H
IDMEM	$F07A_H$	ESFR	$3D_H$	Identifier	0000_H
IDMEM2	$F076_H$	ESFR	$3B_H$	Identifier	0000_H
IDPROG	$F078_H$	ESFR	$3C_H$	Identifier	0000_H
ISNC	$F1DE_H$	ESFR-b	EF_H	Interrupt Sub Node Control Register	0000_H
MDC	$FF0E_H$	SFR-b	87_H	CPU Multiply Divide Control Register	0000_H

Register Name	Physi. Addr	Type	8-bit Addr	Description	Reset Value
MDH	$FE0C_H$	SFR	06_H	CPU Multiply Divide Register - High Word	0000_H
MDL	$FE0E_H$	SFR	07_H	CPU Multiply Divide Register - Low Word	0000_H
ODP0H	$FE22_H$	SFR	11_H	Port 0 Open Drain Control Register High	0000_H
ODP1H	$FE26_H$	SFR	13_H	Port 1 Open Drain Control Register High	0000_H
ODP1L	$FE24_H$	SFR	12_H	Port 1 Open Drain Control Register Low	0000_H
ODP2	$F1C2_H$	ESFR-b	$E1_H$	Port 2 Open Drain Control Register	0000_H
ODP3	$F1C6_H$	ESFR-b	$E3_H$	Port 3 Open Drain Control Register	0000_H
ODP4	$F1CA_H$	ESFR-b	$E5_H$	Port 4 Open Drain Control Register	00_H
ODP6	$F1CE_H$	ESFR-b	$E7_H$	Port 6 Open Drain Control Register	00_H
ONES	$FF1E_H$	SFR-b	$8F_H$	Constant Value 1sRegister'	$FFFF_H$
P0H	$FF02_H$	SFR-b	81_H	Port 0 High Register (Upper half)	00_H
P0HPHEN	$FE6A_H$	SFR	35_H	Port 0 High Pin Hold Enable Register	0000_H
P0HPUDEN	$FE66_H$	SFR	33_H	Port 0 High Pull Switch On/Off Register	$xxFF_H$
P0HPUDSEL	$FE62_H$	SFR	31_H	Port 0 High Pull-Up/Down Select Register	$xxFF_H$
P0L	$FF00_H$	SFR-b	80_H	Port 0 Low Register (Lower half)	00_H
P0LPHEN	$FE68_H$	SFR	34_H	Port 0 Low Pin Hold Enable Register	0000_H
P0LPUDEN	$FE64_H$	SFR	32_H	Port 0 Low Pull Switch On/Off Register	$xxFF_H$
P0LPUDSEL	$FE60_H$	SFR	30_H	Port 0 Low Pull-Up/Down Select Register	$xxFF_H$
P1H	$FF06_H$	SFR-b	83_H	Port 1 High Register (Upper half)	00_H
P1HPHEN	$FE76_H$	SFR	$3B_H$	Port 1 High Pin Hold Enable Register	0000_H
P1HPUDEN	$FE72_H$	SFR	39_H	Port 1 High Pull Switch On/Off Register	0000_H
P1HPUDSEL	$FE6E_H$	SFR	37_H	Port 1 High Pull-Up/Down Select Register	0000_H

Register Name	Physi. Addr	Type	8-bit Addr	Description	Reset Value
P1L	$FF04_H$	SFR-b	82_H	Port 1 Low Register (Lower half)	00_H
P1LPHEN	$FE74_H$	SFR	$3A_H$	Port 1 Low Pin Hold Enable Register	0000_H
P1LPUDEN	$FE70_H$	SFR	38_H	Port 1 Low Pull Switch On/Off Register	0000_H
P1LPUDSEL	$FE6C_H$	SFR	36_H	Port 1 Low Pull-Up/Down Select Register	0000_H
P2	$FFC0_H$	SFR-b	$E0_H$	Port 2 Register	0000_H
P2PHEN	$FE7C_H$	SFR	$3E_H$	Port 2 Pin Hold Enable Register	0000_H
P2PUDEN	$FE7A_H$	SFR	$3D_H$	Port 2 Pull Switch On/Off Register	0000_H
P2PUDSEL	$FE78_H$	SFR	$3C_H$	Port 2 Pull-Up/Down Select Register	0000_H
P3	$FFC4_H$	SFR-b	$E2_H$	Port 3 Register	0000_H
P3PHEN	$FE82_H$	SFR	41_H	Port 3 Pin Hold Enable Register	0000_H
P3PUDEN	$FE80_H$	SFR	40_H	Port 3 Pull Switch On/Off Register	0000_H
P3PUDSEL	$FE7E_H$	SFR	$3F_H$	Port 3 Pull-Up/Down Select Register	0000_H
P4	$FFC8_H$	SFR-b	$E4_H$	Port 4 Register (8 bits)	00_H
P4PHEN	$FE88_H$	SFR	44_H	Port 4 Pin Hold Enable Register	0000_H
P4PUDEN	$FE86_H$	SFR	43_H	Port 4 Pull Switch On/Off Register	0000_H
P4PUDSEL	$FE84_H$	SFR	42_H	Port 4 Pull-Up/Down Select Register	0000_H
P6	$FFCC_H$	SFR-b	$E6_H$	Port 6 Register (8 bits)	00_H
P6PHEN	$FE94_H$	SFR	$4A_H$	Port 6 Pin Hold Enable Register	0000_H
P6PUDEN	$FE92_H$	SFR	49_H	Port 6 Pull Switch On/Off Register	0000_H
P6PUDSEL	$FE90_H$	SFR	48_H	Port 6 Pull-Up/Down Select Register	0000_H
PECC0	$FEC0_H$	SFR	60_H	PEC Channel 0 Control Register	0000_H
PECC1	$FEC2_H$	SFR	61_H	PEC Channel 1 Control Register	0000_H
PECC2	$FEC4_H$	SFR	62_H	PEC Channel 2 Control Register	0000_H
PECC3	$FEC6_H$	SFR	63_H	PEC Channel 3 Control Register	0000_H
PECC4	$FEC8_H$	SFR	64_H	PEC Channel 4 Control Register	0000_H
PECC5	$FECA_H$	SFR	65_H	PEC Channel 5 Control Register	0000_H
PECC6	$FECC_H$	SFR	66_H	PEC Channel 6 Control Register	0000_H
PECC7	$FECE_H$	SFR	67_H	PEC Channel 7 Control Register	0000_H

Register Name	Physi. Addr	Type	8-bit Addr	Description	Reset Value
PECCLIC	$F180_H$	ESFR-b	$C0_H$	PEC Channel Link Interrupt Control Register	0000_H
PECSN0	$FED0_H$	SFR	68_H	PEC Segment No Register	
PECSN1	$FED2_H$	SFR	69_H	PEC Segment No Register	
PECSN2	$FED4_H$	SFR	$6A_H$	PEC Segment No Register	
PECSN3	$FED6_H$	SFR	$6B_H$	PEC Segment No Register	
PECSN4	$FED8_H$	SFR	$6C_H$	PEC Segment No Register	
PECSN5	$FEDA_H$	SFR	$6D_H$	PEC Segment No Register	
PECSN6	$FEDC_H$	SFR	$6E_H$	PEC Segment No Register	
PECSN7	$FEDE_H$	SFR	$6F_H$	PEC Segment No Register	
PECXC0	$FEF0_H$	SFR	78_H	PEC Channel 0 Extended Control Register	
PECXC2	$FEF2_H$	SFR	79_H	PEC Channel 2 Extended Control Register	
PSW	$FF10_H$	SFR-b	88_H	CPU Program Status Word	0000_H
R0		SFR-b	$F0_H$	General Purpose Register 0	$UUUU_H$
R0		ESFR-b	$F0_H$	General Purpose Register 0	$UUUU_H$
R1		SFR-b	$F1_H$	General Purpose Register 1	$UUUU_H$
R1		ESFR-b	$F1_H$	General Purpose Register 1	$UUUU_H$
R10		ESFR-b	FA_H	General Purpose Register 10	$UUUU_H$
R10		SFR-b	FA_H	General Purpose Register 10	$UUUU_H$
R11		SFR-b	FB_H	General Purpose Register 11	$UUUU_H$
R11		ESFR-b	FB_H	General Purpose Register 11	$UUUU_H$
R12		SFR-b	FC_H	General Purpose Register 12	$UUUU_H$
R12		ESFR-b	FC_H	General Purpose Register 12	$UUUU_H$
R13		SFR-b	FD_H	General Purpose Register 13	$UUUU_H$
R13		ESFR-b	FD_H	General Purpose Register 13	$UUUU_H$
R14		SFR-b	FE_H	General Purpose Register 14	$UUUU_H$
R14		ESFR-b	FE_H	General Purpose Register 14	$UUUU_H$
R15		ESFR-b	FF_H	General Purpose Register 15	$UUUU_H$
R15		SFR-b	FF_H	General Purpose Register 15	$UUUU_H$

Register Name	Physi. Addr	Type	8-bit Addr	Description	Reset Value
R2		SFR-b	$F2_H$	General Purpose Register 2	$UUUU_H$
R2		ESFR-b	$F2_H$	General Purpose Register 2	$UUUU_H$
R3		SFR-b	$F3_H$	General Purpose Register 3	$UUUU_H$
R3		ESFR-b	$F3_H$	General Purpose Register 3	$UUUU_H$
R4		ESFR-b	$F4_H$	General Purpose Register 4	$UUUU_H$
R4		SFR-b	$F4_H$	General Purpose Register 4	$UUUU_H$
R5		ESFR-b	$F5_H$	General Purpose Register 5	$UUUU_H$
R5		SFR-b	$F5_H$	General Purpose Register 5	$UUUU_H$
R6		ESFR-b	$F6_H$	General Purpose Register 6	$UUUU_H$
R6		SFR-b	$F6_H$	General Purpose Register 6	$UUUU_H$
R7		ESFR-b	$F7_H$	General Purpose Register 7	$UUUU_H$
R7		SFR-b	$F7_H$	General Purpose Register 7	$UUUU_H$
R8		SFR-b	$F8_H$	General Purpose Register 8	$UUUU_H$
R8		ESFR-b	$F8_H$	General Purpose Register 8	$UUUU_H$
R9		SFR-b	$F9_H$	General Purpose Register 9	$UUUU_H$
R9		ESFR-b	$F9_H$	General Purpose Register 9	$UUUU_H$
RES4IC	$FF98_H$	SFR-b	CB_H	reserved	0000_H
RES6IC	$F19A_H$	ESFR-b	CD_H	reserved	0000_H
reserved	$F1D6_H$	ESFR-b	EB_H	reserved - do not use	0000_H
reserved	$F1D8_H$	ESFR-b	EC_H	reserved - do not use	0000_H
RP0H	$F108_H$	ESFR-b	84_H	System Startup Configuration Register (RO)	xx_H
RTC_INTIC	$F184_H$	ESFR-b	$C2_H$	RTC_INT Sub Node Interrupt Register	0000_H
RTCCLC	$F0C8_H$	ESFR	64_H	RTC Clock Control Register	0000_H
RTCCON	$F1CC_H$	ESFR-b	$E6_H$	RTC Control Register	00_H
RTCH	$F0D6_H$	ESFR	$6B_H$	RTC Timer Register High	n_H
RTCISNC	$F1C8_H$	ESFR-b	$E4_H$	RTC Interrupt Sub Node Control Register	0000_H
RTCL	$F0D4_H$	ESFR	$6A_H$	RTC Timer Register Low	n_H
RTCRELH	$F0CE_H$	ESFR	67_H	RTC Timer Reload Register High	0000_H

Register Name	Physi. Addr	Type	8-bit Addr	Description	Reset Value
RTCRELL	$F0CC_H$	ESFR	66_H	RTC Timer Reload Register Low	0000_H
S0BG	$FEB4_H$	SFR	$5A_H$	Serial Channel 0 Baud Rate Generator Reload Register	0000_H
S0CLC	$FFBA_H$	SFR-b	DD_H	ASC Clock Control Register	0000_H
S0CON	$FFB0_H$	SFR-b	$D8_H$	Serial Channel 0 Control Register	0000_H
S0EIC	$FF70_H$	SFR-b	$B8_H$	Serial Channel 0 Error Interrupt Control Register	0000_H
S0FDV	$FEB6_H$	SFR	$5B_H$	ASC Fractional Divide Register	0000_H
S0PMW	$FEAA_H$	SFR	55_H	ASC IrDA PMW Control Register	0000_H
S0RBUF	$FEB2_H$	SFR	59_H	Serial Channel 0 Receive Buffer Register (RO)	$xxxx_H$
S0RIC	$FF6E_H$	SFR-b	$B7_H$	Serial Channel 0 Receive Interrupt Control Register	0000_H
S0TBIC	$F19C_H$	ESFR-b	CE_H	Serial Channel 0 Transmit Buffer IC Register	0000_H
S0TBUF	$FEB0_H$	SFR	58_H	Serial Channel 0 Transmit Buffer Register (WO)	0000_H
S0TIC	$FF6C_H$	SFR-b	$B6_H$	Serial Channel 0 Transmit Interrupt Control Register	0000_H
SCUSLC	$F0C0_H$	ESFR	60_H	Security Level Control Register	
SCUSLS	$F0C2_H$	ESFR	61_H	Security Level Status Register	
SP	$FE12_H$	SFR	09_H	CPU System Stack Pointer Register	$FC00_H$
SSCBR	$F0B4_H$	ESFR	$5A_H$	SSC Baudrate Register	0000_H
SSCCLC	$F0B6_H$	ESFR	$5B_H$	SSC Clock Control Register	0000_H
SSCCON	$FFB2_H$	SFR-b	$D9_H$	SSC Control Register	0000_H
SSCEIC	$FF76_H$	SFR-b	BB_H	SSC Error Interrupt Control Register	0000_H
SSCRB	$F0B2_H$	ESFR	59_H	SSC Receive Buffer (RO)	$xxxx_H$
SSCRIC	$FF74_H$	SFR-b	BA_H	SSC Receive Interrupt Control Register	0000_H
SSCTB	$F0B0_H$	ESFR	58_H	SSC Transmit Buffer (WO)	0000_H
SSCTIC	$FF72_H$	SFR-b	$B9_H$	SSC Transmit Interrupt Control Register	0000_H

Register Name	Physi. Addr	Type	8-bit Addr	Description	Reset Value
STKOV	$FE14_H$	SFR	$0A_H$	CPU Stack Overflow Pointer Register	$FA00_H$
STKUN	$FE16_H$	SFR	$0B_H$	CPU Stack Underflow Pointer Register	$FC00_H$
SYSCON	$FF12_H$	SFR-b	89_H	CPU System Configuration Register	$0xx0_H$
SYSCON1	$F1DC_H$	ESFR-b	EE_H	System Configuration Register 1/ Sleep Mode	0000_H
SYSCON2	$F1D0_H$	ESFR-b	$E8_H$	System Configuration Register 2/ Clock Control	0000_H
SYSCON3	$F1D4_H$	ESFR-b	EA_H	System Configuration Register 3/ Periph. Managem.	0000_H
T14	$F0D2_H$	ESFR	69_H	Timer 14 Register	n_H
T14REL	$F0D0_H$	ESFR	68_H	Timer 14 Reload Register	n_H
T2	$FE40_H$	SFR	20_H	GPT1 Timer 2 Register	0000_H
T2CON	$FF40_H$	SFR-b	$A0_H$	GPT1 Timer 2 Control Register	0000_H
T2IC	$FF60_H$	SFR-b	$B0_H$	GPT1 Timer 2 Interrupt Control Register	0000_H
T3	$FE42_H$	SFR	21_H	GPT1 Timer 3 Register	0000_H
T3CON	$FF42_H$	SFR-b	$A1_H$	GPT1 Timer 3 Control Register	0000_H
T3IC	$FF62_H$	SFR-b	$B1_H$	GPT1 Timer 3 Interrupt Control Register	0000_H
T4	$FE44_H$	SFR	22_H	GPT1 Timer 4 Register	0000_H
T4CON	$FF44_H$	SFR-b	$A2_H$	GPT1 Timer 4 Control Register	0000_H
T4IC	$FF64_H$	SFR-b	$B2_H$	GPT1 Timer 4 Interrupt Control Register	0000_H
T5	$FE46_H$	SFR	23_H	GPT2 Timer 5 Register	0000_H
T5CON	$FF46_H$	SFR-b	$A3_H$	GPT2 Timer 5 Control Register	0000_H
T5IC	$FF66_H$	SFR-b	$B3_H$	GPT2 Timer 5 Interrupt Control Register	0000_H
T6	$FE48_H$	SFR	24_H	GPT2 Timer 6 Register	0000_H
T6CON	$FF48_H$	SFR-b	$A4_H$	GPT2 Timer 6 Control Register	0000_H
T6IC	$FF68_H$	SFR-b	$B4_H$	GPT2 Timer 6 Interrupt Control Register	0000_H

Register Name	Physi. Addr	Type	8-bit Addr	Description	Reset Value
UCFGVIC	$F16E_H$	ESFR-b	$B7_H$	UDC Config Val Interrupt Control Register	0000_H
ULCDIC	$F176_H$	ESFR-b	BB_H	UDC Load Config Done Interrupt Control Register	0000_H
URD0IC	$F17A_H$	ESFR-b	BD_H	UDC RX Done0 Interrupt Control Register	0000_H
URD1IC	$FF9E_H$	SFR-b	CF_H	UDC RX Done1 Interrupt Control Register	0000_H
URD2IC	$FF9C_H$	SFR-b	CE_H	UDC RX Done2 Interrupt Control Register	0000_H
URD3IC	$FF78_H$	SFR-b	BC_H	UDC RX Done3 Interrupt Control Register	0000_H
URD4IC	$FF7A_H$	SFR-b	BD_H	UDC RX Done4 Interrupt Control Register	0000_H
URD5IC	$FF7C_H$	SFR-b	BE_H	UDC RX Done5 Interrupt Control Register	0000_H
URD6IC	$FF7E_H$	SFR-b	BF_H	UDC RX Done6 Interrupt Control Register	0000_H
URD7IC	$FF80_H$	SFR-b	$C0_H$	UDC RX Done7 Interrupt Control Register	0000_H
URXRIC	$F16A_H$	ESFR-b	$B5_H$	UDC RXRR Interrupt Control Register	0000_H
USETIC	$F178_H$	ESFR-b	BC_H	UDC SETUP Interrupt Control Register	0000_H
USOFIC	$F170_H$	ESFR-b	$B8_H$	UDC Start of Frame Interrupt Control Register	0000_H
USSIC	$F174_H$	ESFR-b	BA_H	UDC Suspend Interrupt Control Register	0000_H
USSOIC	$F172_H$	ESFR-b	$B9_H$	UDC Suspend off Interrupt Control Register	0000_H
UTD0IC	$FF82_H$	SFR-b	$C1_H$	UDC TX Done0 Interrupt Control Register	0000_H
UTD1IC	$FF84_H$	SFR-b	$C2_H$	UDC TX Done1 Interrupt Control Register	0000_H

Register Name	Physi. Addr	Type	8-bit Addr	Description	Reset Value
UTD2IC	$FF86_H$	SFR-b	$C3_H$	UDC TX Done2 Interrupt Control Register	0000_H
UTD3IC	$F160_H$	ESFR-b	$B0_H$	UDC TX Done3 Interrupt Control Register	0000_H
UTD4IC	$F162_H$	ESFR-b	$B1_H$	UDC TX Done4 Interrupt Control Register	0000_H
UTD5IC	$F164_H$	ESFR-b	$B2_H$	UDC TX Done5 Interrupt Control Register	0000_H
UTD6IC	$F166_H$	ESFR-b	$B3_H$	UDC TX Done6 Interrupt Control Register	0000_H
UTD7IC	$F168_H$	ESFR-b	$B4_H$	UDC TX Done7 Interrupt Control Register	0000_H
UTXRIC	$F16C_H$	ESFR-b	$B6_H$	UDC TXWR Interrupt Control Register	0000_H
TFR	$FFAC_H$	SFR-b	$D6_H$	Trap Flag Register	0000_H
UCFGVIC	$F16E_H$	ESFR-b	$B7_H$	UDC Config Val Interrupt Control Register	0000_H
ULCDIC	$F176_H$	ESFR-b	BB_H	UDC Load Config Done Interrupt Control Register	0000_H
WDT	$FEAE_H$	SFR	57_H	Watchdog Timer Register (RO)	0000_H
WDTCON	$FFAE_H$	SFR-b	$D7_H$	Watchdog Timer Control Register	$000x_H$
XADRS1	$F014_H$	ESFR	$0A_H$	XBUS Address Select Register 1	
XADRS2	$F016_H$	ESFR	$0B_H$	XBUS Address Select Register 2	
XADRS3	$F018_H$	ESFR	$0C_H$	XBUS Address Select Register 3	
XADRS4	$F01A_H$	ESFR	$0D_H$	XBUS Address Select Register 4	
XADRS5	$F01C_H$	ESFR	$0E_H$	XBUS Address Select Register 5	
XADRS6	$F01E_H$	ESFR	$0F_H$	XBUS Address Select Register 6	
XBCON1	$F114_H$	ESFR-b	$8A_H$	XBUS Control register 1: reserved	0000_H
XBCON2	$F116_H$	ESFR-b	$8B_H$	XBUS Control register 2: USB module	0000_H
XBCON3	$F118_H$	ESFR-b	$8C_H$	XBUS Control register 3: EPEC module	0000_H
XBCON4	$F11A_H$	ESFR-b	$8D_H$	XBUS Control register 4: reserved	0000_H

Register Name	Physi. Addr	Type	8-bit Addr	Description	Reset Value
XBCON5	$F11C_H$	ESFR-b	$8E_H$	XBUS Control register 5: reserved	0000_H
XBCON6	$F11E_H$	ESFR-b	$8F_H$	XBUS Control register 6: reserved	
XP0IC	$F186_H$	ESFR-b	$C3_H$	X-Bus Peripheral 0 UDC TXWR Interrupt Control Register	0000_H
XP1IC	$F18E_H$	ESFR-b	$C7_H$	X-Bus Peripheral 1 EPEC Interrupt Control Register	0000_H
XP3IC	$F19E_H$	ESFR-b	CF_H	X-Bus Peripheral 3 PLL/RTC Interrupt Control Register	0000_H
XPERCON	$F024_H$	ESFR	12_H	XBUS Peripheral Control Register	0401_H
ZEROS	$FF1C_H$	SFR-b	$8E_H$	Constant Value 0sRegister'	0000_H

22.5 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.

23 Instruction Set Summary

This chapter briefly summarizes the C161U's instructions ordered by instruction classes. This provides a basic understanding of the C161U'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 C16x 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 (eg. SHR, ROR) and variations of certain instructions (eg. ADD, ADDB). This provides an easy access to the possibilities and the power of the instructions of the C161U.

Note: The used mnemonics refer to the detailled 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 ontothe 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: ETS
- 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 C161U via software: SRST
- Entering the Idle mode: DLE
- 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 eg. 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 C161U 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 (eg. 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.

24 AC/DC Characteristics

24.1 Absolute Maximum Ratings

- Storage temperature (T_{ST}) -65 to +150 °C
- Voltage on V_{DD} pins with respect to ground (V_{SS}) -0.5 to + 4.0 V
- Voltage on any pin with respect to ground (V_{SS}) (except V_{DD}, V_{SS}, XTAL and USB pins) -0.5 to 5.5 V
- Absolute maximum total I/O current 250 mA
- Absolute maximum current on any pin, sink or source 10 mA

Stresses beyond those listed above may cause permanent damage to the device. This is a stress rating only, and functional operation of the C161U is not implied at these or any other conditions above those indicated in the operational sections of this specification. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

24.2 Recommended Operating Conditions

The following conditions are to be met for correct operation of the device.

- Ambient temperature under bias (T_A): -40 to +85 °C
- Load capacitance (C_L): $\leq$100 pF

24.3 DC Characteristics

The parameters listed below partly represent the characteristics of the C161U and partly its demands on the system. To aid in interpreting the table correctly when evaluating parameters for a design, the following notation is used in the column “Symbol”:

CC (**C**ontroller **C**haracteristics):
The logic of the C161U will provide signals with the respective timing characteristics.

SR (**S**ystem **R**equirement):
The external system must provide signals with the respective timing characteristics to the C161U.

V_{DD} = 3.3 V ± 10%; V_{SS} = 0 V
T_A = -40 to 85° C, nom = 25° C

Table 94 DC Characteristics

Parameter	Symbol	Limit Values			Unit	Test Condition
		min.	nom.	max.		
Power source current, normal operation	I_{CC}	–	95	150	mA	36 MHz system frequency [6]
Power source current, idle mode	I_{ID}	–	–	t.b.d.	mA	–
Power source current, sleep mode	I_{PD}	–	140	–	µA	–
Power source current, power-down mode	I_{PD}	–	25	–	µA	–
Input low voltage	V_{ILSR}	-0.5	–	0.8	V	–
Input high voltage	V_{IHSR}	2.2	–	5.5	V	V_{DD}= 3.6 V
	V_{IHSR}	2.0	–		V	V_{DD}= 3.3 V
	V_{IHSR}	1.8	–		V	V_{DD}= 3.0 V
Output low voltage	V_{OLCC}	–	–	0.4	V	I_{OL} = 3.2 mA
Output high voltage	V_{OHCC}	2.4	–	–	V	I_{OH} = -3.2 mA
Input leakage current	I_{OZ2CC}	–	–	±1	µA	0 V < V_{IN} < V_{DD}
$\overline{\text{RSTIN}}$ pullup resistor	R_{RSTCC}	100	–	660	kΩ	at V_{DD} = 3.3 V
Read/Write inactive current [4]	I_{RWH} [2]	–	–	-40	µA	V_{OUT} = 2.4 V
Read/Write active current [4]	I_{RWL} [3]	-100	–	–	µA	V_{OUT} = V_{OLmax}
ALE inactive current [4]	I_{ALEL} [2]	–	–	40	µA	V_{OUT} = V_{OLmax}
ALE active current [4]	I_{ALEH} [3]	100	–	–	µA	V_{OUT} = 2.4 V
Port 6 inactive current [4]	I_{P6H} [2]	–	–	-40	µA	V_{OUT} = 2.4 V
Port 6 active current [4]	I_{P6L} [3]	-100	–	–	µA	V_{OUT} = V_{OLmax}

Table 94 DC Characteristics (cont'd)

Parameter	Symbol	Limit Values			Unit	Test Condition
		min.	nom.	max.		
Port 0 configuration current [4]	I_{P0H} [2]	–	–	-10	µA	$V_{IN} = V_{IHmin}$
	I_{P0L} [3]	-100	–	–	µA	$V_{IN} = V_{ILmax}$
XTAL1 input current	I_{ILCC}	–	–	±20	µA	$0\ V < V_{IN} < V_{DD}$
XTAL1 max input voltage[5]	V_{IH2}	1	–	V_{DD} + 0.3	V	–
Pin capacitance[1] (digital inputs/outputs)	C_{IOCC}	–	–	9	pF	f = 1 MHz; T_A = 25 °C

1) Not tested; guaranteed by design characterization.

2) The maximum current may be drawn while the respective signal line remains inactive.

3) The minimum current must be drawn in order to drive the respective signal line active.

4) This specification is only valid during Reset or during Adapt-mode. Port 6 pins are affected only if they are used for $\overline{CS}$ output and the open drain function is not enabled.

5) Not 5-V tolerant.

6) At a lower system frequency, the power consumption decreases accordingly.

Note:

1. The sum of power from all port pins may not exceed 1 W; total I/O current may not exceed 250 mA.
2. The strength of output drivers (I_{OL} and I_{OH}) is 7.5 mA.

24.4 USB Full-speed (12 Mbit/s) Driver Characteristics

C161U is compliant to the characterization of the USB interface according to "*Universal Serial Bus Specification, Revision 1.1, September 23, 1998*". More specific, the Driver Characteristics can be found on pp. 108 of this specification.

Note: C161U meets all values at **25 grad Celsius** as specified in the USB Spec 1.1 For higher temperature, the C161U values deviate up to 3.5 % worse than "full-speed buffer V/I characteristics" according to USB Spec 1.1, page 109.

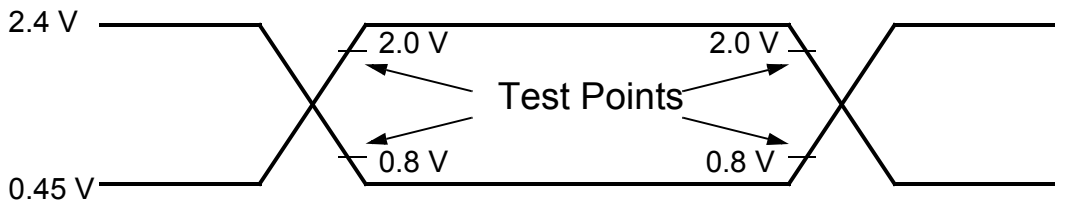
24.5 Failsafe operation

C161U I/O pins may be exposed up to a 5.5 V level generated by the other system components. That may happen during operation in the normal power range as well as during power-up/down transitions when the value of VDD may be anywhere in the range from 0 V to 3.63 V. The following table specifies 5.5 V failsafe conditions for the different ranges of VDD.

Table 95 Failsafe conditions

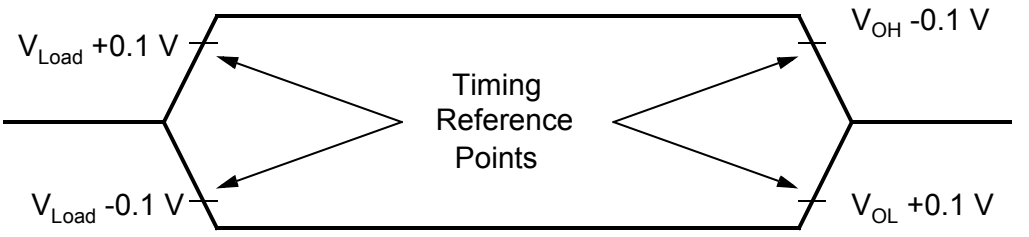
	VDD	I/O Status	Safe condition with 5.5 V applied to I/O pin	Note
Power-up	Not connected	Undetermined	Not to exceed 10 mA on any pin, 250 mA total	_
	0 V - 2.97 V	Undetermined	Not to exceed 10 mA on any pin, 250 mA total	_
Normal Power range	2.97 V - 3.63 V	Determined	Not to exceed 10 mA on any pin, 250 mA total	_
Power-down	2.97 V - 2.25 V	Determined	Not to exceed 10 mA on any pin, 250 mA total	At 2.5 V ± 10% (Power down mode) I/Os are active and preserve the status
	2.25 V - 0	Undetermined	Not to exceed 10 mA on any pin, 250 mA total	_

24.6 Testing Waveforms



AC inputs during testing are driven at 2.4 V for a logic ‘1’ and 0.45 V for a logic ‘0’.
Timing measurements are made at V_{IH} min for a logic ‘1’ and V_{IL} max for a logic ‘0’.

Figure 124 Input Output Waveforms



For timing purposes a port pin is no longer floating when a 100 mV change from load voltage occurs, but begins to float when a 100 mV change from the loaded V_{OH}/V_{OL} level occurs (I_{OH}/I_{OL} = 20 mA).

Figure 125 Float Waveforms

24.7 AC Characteristics

The parameters in this chapter partly represent the characteristics of the C161U and partly its demands on the system. To aid in interpreting the table correctly when evaluating parameters for a design, the following notation is used in the column “Symbol”:

CC (**C**ontroller **C**haracteristics):
The logic of the C161U will provide signals with the respective timing characteristics.

SR (**S**ystem **R**equirement):
The external system must provide signals with the respective timing characteristics to the C161U.

24.7.1 Definition of Internal Timing

The internal operation of the C161U is controlled by the internal CPU clock f_{CPU}. Both edges of the CPU clock can trigger internal (eg. pipeline) or external (eg. bus cycles) operations. The specification of the external timing (AC Characteristics) therefore depends on the time between two consecutive edges of the CPU clock, called “TCL” (see **Figure 126**).

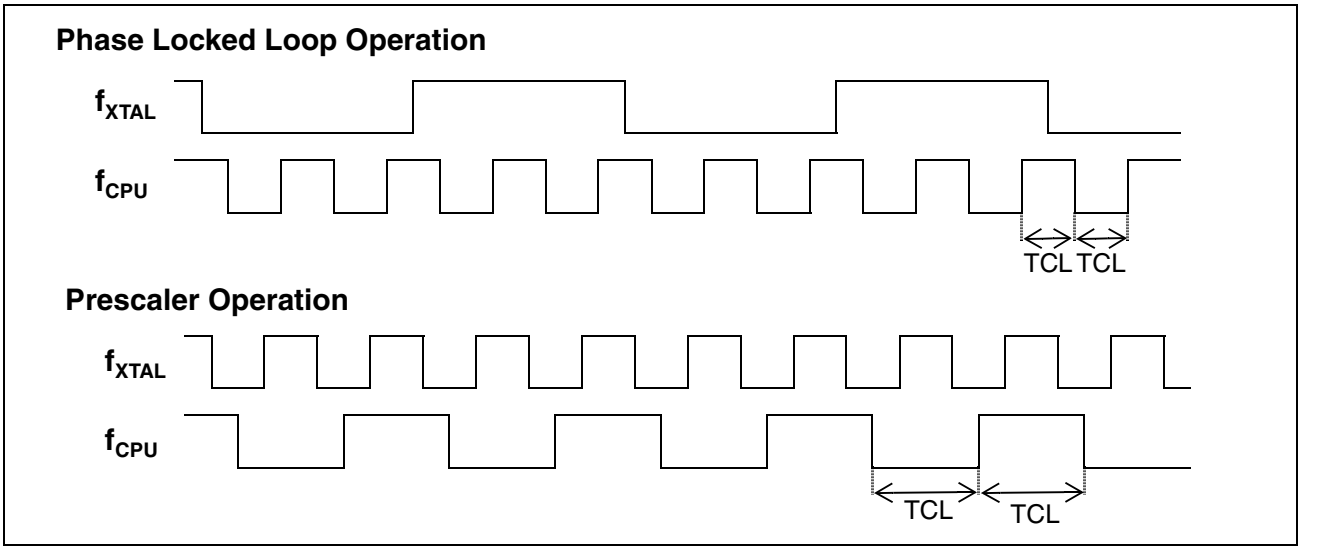


Figure 126 Generation mechanisms for the CPU Clock

The CPU clock signal can be generated via different mechanisms. The mechanism used to generate the CPU clock is selected during reset via the logic levels on pins P0.15-13 (P0H.7-5) and is described in detail in **Chapter 3.3**, page 35. The duration of TCLs and their variation (and also the derived external timing) depends on the mechanism used to generate f_{CPU}. This influence must be regarded when calculating the timings for the C161U.

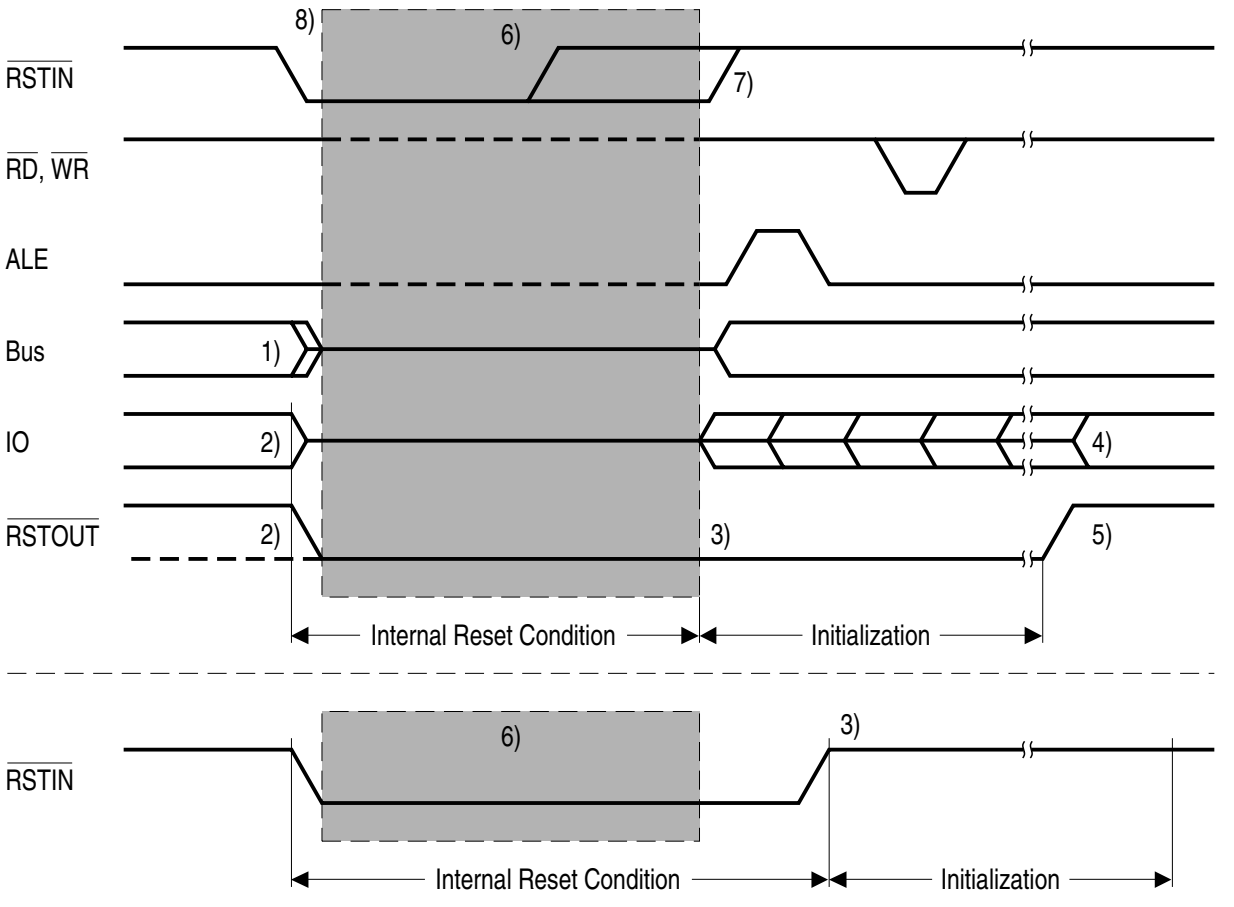
Note: The example for PLL operation shown in **Figure 126** refers to a PLL factor of 4.

The PLL multiplies the input frequency by the factor **F** which is selected via the combination of pins P0.15-13 (ie. $f_{CPU} = f_{XTAL} * \mathbf{F}$). With every **F**'th transition of f_{XTAL} the PLL circuit synchronizes the CPU clock to the input clock. This synchronization is done smoothly, i.e. the CPU clock frequency does not change abruptly.

Due to this adaptation to the input clock, the frequency of f_{CPU} is constantly adjusted so it is locked to f_{XTAL}. The slight variation causes a jitter of f_{CPU} which also affects the duration of individual TCLs.

The timings listed in the AC Characteristics that refer to TCLs therefore must be calculated using the minimum TCL that is possible under the respective circumstances. The actual minimum value for TCL depends on the jitter of the PLL. As the PLL is constantly adjusting its output frequency so it corresponds to the applied input frequency (crystal or oscillator), the relative deviation for periods of more than one TCL is lower than for a single TCL. This is especially important for bus cycles using wait-states and for the operation of timers, serial interfaces, etc. For all slower operations and longer periods (eg. pulse train generation or measurement, lower baudrates, etc.), the deviation caused by the PLL jitter is negligible.

24.7.2 System Reset



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 asinchronously with $\overline{\text{RSTIN}}$, sinchronously upon software or watchdog reset.
3) The reset condition ends here. The C 167CR starts program execution.
4) Activation of the IO pins is controlled by software.
5) Execution of the EINIT instruction.
6) The shaded area designates the internal reset sequence, which starts after synchronization of $\overline{\text{RSTIN}}$.
7) A short hardware reset is extended until the end of the reset sequence in Bidirectional reset mode.
8) A software or WDT reset activates the $\overline{\text{RSTIN}}$ line in Bidirectional reset mode.

MCS02258

Figure 127 Reset Input and Output Signals

Note: Minimum reset time after power on is 1 ms after voltage reaches V_{DD} minimum.

24.7.3 External Clock Drive XTAL1

V_{DD} = 3.3 V ± 10%; V_{SS} = 0 V
T_A = -40 to +85 °C

Table 96 External Clock Drive XTAL 1

Parameter	Symbol	External crystal: 4-20 MHz (internal oscillator "on", PLL running)		Direct drive: 4-36 MHz (internal oscillator by-passed, PLL "free running" or "off")		Unit
		min	max	min	max	
Oscillator period	t_{OSCSR}	50	250	27.8	250	ns
Duty Cycle		-	-	50		%
High time	t_{1SR}	-	-	13.9	125	ns
Low time	t_{2SR}	-	-	13.9	125	ns
Rise time	t_{3SR}	-	-	-	3 @ 36 MHz	ns
Fall time	t_{4SR}	-	-	-	3 @ 36 MHz	ns

Note: Special requirements for the external crystal must be observed: The accuracy of the crystal must be 96ppm or better. Please note, the implemented low swing crystal oscillator has a signal amplitude of only about 1 V peak-to-peak. More detailed information can also be found in the appropriated application note.

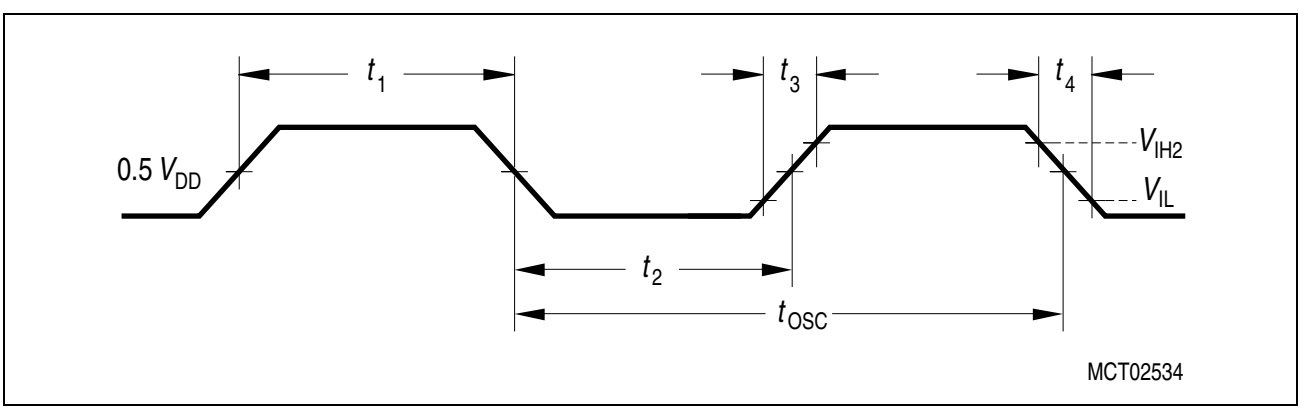


Figure 128 External Clock Drive XTAL1

24.7.4 JTAG Interface Timing

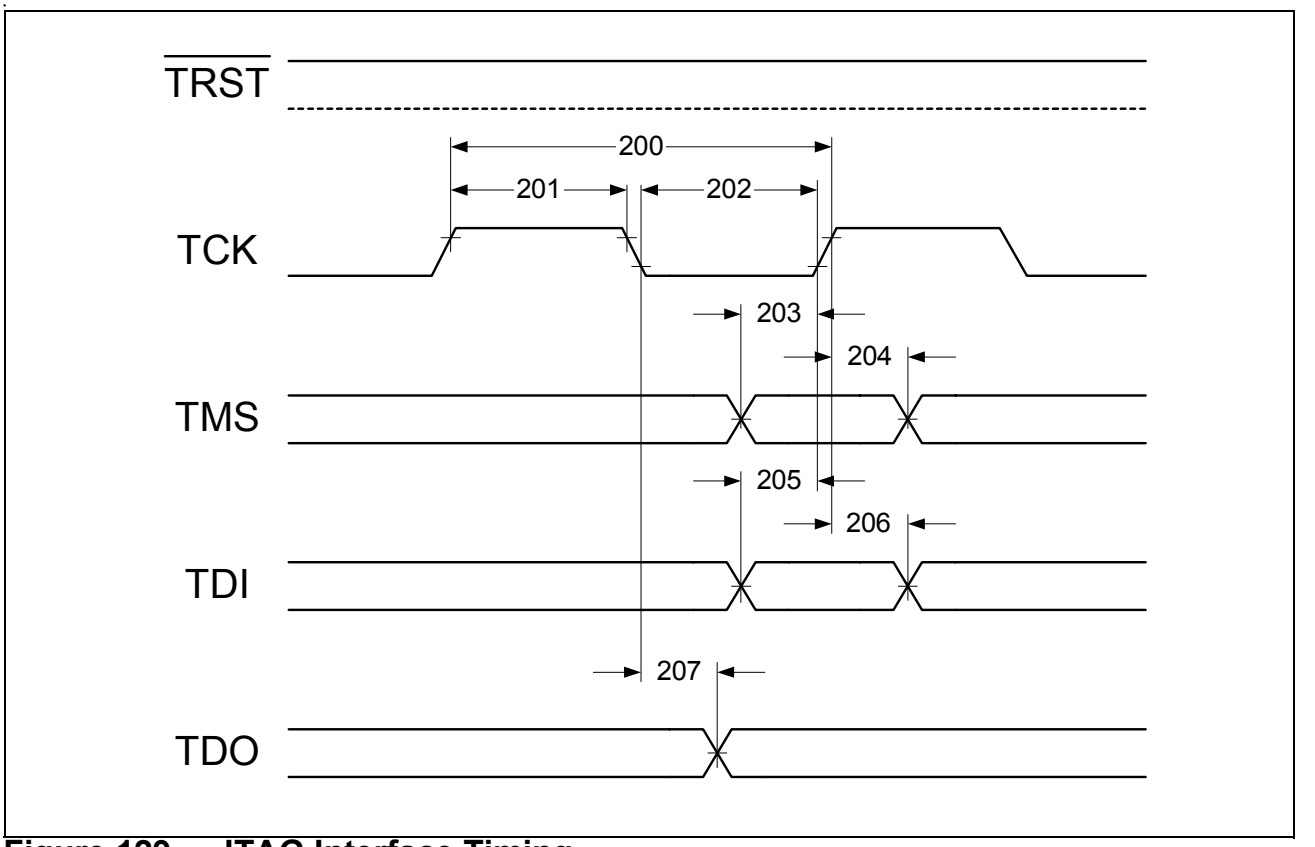


Figure 129 JTAG Interface Timing

Table 97 JTAG Interface Timing

No.	Parameter	Limit Values		Unit
		min.	max.	
200	TCK period	120		ns
201	TCK high time	60		ns
202	TCK low time	60		ns
203	TMS setup time	20		ns
204	TMS hold time	20		ns
205	TDI setup time	20		ns
206	TDI hold time	20		ns
207	TDO valid time	50		ns

24.8 Asynchronous Bus Timing

This term means that timing is defined with respect to ALE (as opposed to CLKOUT). The following configurations are typical :

Table 98 Asynchronous Bus Timing

RDY	ALE	WR	MTTC	RD/WR	MCTC	cycles	Application	BUSCON
no	normal	early	no	normal	no	2	SRAMS demuxed bus	0A3F (8) or 0ABF(16)
no	normal	-	no	normal	1	3	fast EPROMS demuxed bus	0A3E / 0ABE
no	normal	-	1	normal	2	5	slow FLASH demuxed bus	0A1D / 0A9D
no	normal	normal	no	delayed	no	2+1	SRAMS muxed bus	04EF / 046F
no	normal	-	no	delayed	1	3+1	fast EPROMS muxed bus	04EE / 046E
no	normal	-	no	delayed	2	4+1	slow FLASH muxed bus	04ED / 046D

24.8.1 Memory Cycle Variables

The timing tables below use 4 variables which are derived from the BUSCONx registers and represent the special characteristics of the programmed memory cycle. The following table describes, how these variables are to be computed.

Table 99 Memory Cycle Variables

Description	Symbol	Values
ALE extension	t_A	TCL * <ALECTL>
memory cycle time waitstates	t_C	2TCL * (15 - <MCTC>)
memory tristate time	t_F	2TCL * (1 - <MTTC>)
early write	t_W	TCL * <EWEN>

24.8.1.1 AC Characteristics, Multiplexed Bus

V_{DD} = 3.3 V ± 10 %; V_{SS} = 0 V
T_A = -40 to +85 °C
C_L (for PORT0, PORT1, Port 4, ALE, $\overline{RD}$, $\overline{WR}$, $\overline{BHE}$, CLKOUT) = 100 pF
C_L (for Port 6, $\overline{CS}$) = 100 pF
ALE cycle time = 6 TCL + 2t_A + t_C + t_F (83.3 ns at 36 MHz CPU clock without wait states)

Table 100 AC Characteristics, Multiplexed Bus

Parameter	Symbol	Max. CPU Clock 36 MHz (TCL=14ns)		Variable CPU Clock 5 to 36 MHz		Unit
		min.	max.	min.	max.	
ALE high time	t_{5}CC	4 + t_A	–	TCL - 10 + t_A	–	ns
Address, $\overline{CSx}$[2] setup to ALE	t6CC	-6 + t_A	–	TCL - 20 + t_A	–	ns
Address hold after ALE	t7CC	4 + t_A		TCL - 10 + t_A		ns
ALE falling edge to $\overline{RD}$, $\overline{WR}$ (with RW-delay)	t8CC	4 + t_A	–	TCL - 10 + t_A	–	ns
ALE falling edge to $\overline{RD}$, $\overline{WR}$ (no RW-delay)	t9CC	-10 + t_A	–	-10 + t_A	–	ns
Address float after RD, WR (with RW-delay)	t10CC	–	15	–	15	ns
Address float after $\overline{RD}$, $\overline{WR}$ (no RW-delay)	t11CC	–	29	–	TCL + 15	ns
$\overline{RD}$, $\overline{WR}$ low time (with RW-delay) [3]	t_{12}CC	17 + t_C - t_W	–	2TCL - 11 + t_C - t_W	–	ns
$\overline{RD}$, $\overline{WR}$ low time (no RW-delay) [3]	t_{13}CC	31 + t_C - t_W	–	3TCL - 11 + t_C - t_W	–	ns
$\overline{RD}$ to valid data in (with RW-delay)	t_{14}SR	–	0 + t_C	–	2TCL - 28 + t_C	ns
$\overline{RD}$ to valid data in (no RW-delay)	t_{15}SR	–	13 + t_C	–	3TCL - 29 + t_C	ns
ALE low to valid data in	t_{16}SR	–	13 + t_A + t_C	–	3TCL - 29 + t_A + t_C	ns
Address, $\overline{CSx}$[2] to valid data in	t_{17}SR	–	18 + 2t_A + t_C	–	4TCL - 38 + 2t_A + t_C	ns
Data hold after $\overline{RD}$ rising edge	t_{18}SR	0	–	0	–	ns
Data float after $\overline{RD}$	t_{19}SR	–	13 + t_F	–	2TCL - 15 + t_F	ns
Data valid to $\overline{WR}$	t_{22}CC	13 + t_C - t_W	–	2TCL - 15 + t_C - t_W	–	ns

Table 100 AC Characteristics, Multiplexed Bus (cont'd)

Parameter	Symbol	Max. CPU Clock 36 MHz (TCL=14ns)		Variable CPU Clock 5 to 36 MHz		Unit
		min.	max.	min.	max.	
Data hold after $\overline{WR}$	t_{23}CC	13 + t_F + t_W	–	2TCL - 15 + t_F + t_W	–	ns
ALE rising edge after $\overline{RD}$, $\overline{WR}$ [3]	t_{25}CC	13 + t_F + t_W	–	2TCL - 15 + t_F + t_W	–	ns
Address hold after $\overline{RD}$, $\overline{WR}$ [3]	t_{27}CC	13 + t_F + t_W	–	2TCL - 15 + t_F + t_W	–	ns
ALE falling edge to $\overline{CSx}$ [1]	t_{38}CC	-9 - t_A	13 - t_A	-9 - t_A	13 - t_A	ns
$\overline{CSx}$ [1] low to Valid Data In	t_{39}SR	–	12 + t_C + 2t_A	–	3TCL - 30 + t_C+2t_A	ns
$\overline{CSx}$ [1] hold after $\overline{RD}$, $\overline{WR}$ [3]	t_{40}CC	27 + t_F + t_W	–	3TCL - 15 + t_F + t_W	–	ns
ALE falling edge to $\overline{RdCS}$, $\overline{WrCS}$ (with RW delay)	t_{42}CC	5 + t_A	–	TCL - 9 + t_A	–	ns
ALE falling edge to $\overline{RdCS}$, $\overline{WrCS}$ (no RW delay)	t_{43}CC	-9 + t_A	–	-9 + t_A	–	ns
Address float after $\overline{RdCS}$, $\overline{WrCS}$ (with RW delay)	t_{44}CC	–	13	–	13	ns
Address float after $\overline{RdCS}$, $\overline{WrCS}$ (no RW delay)	t_{45}CC	–	27	–	TCL + 13	ns
$\overline{RdCS}$ to Valid Data In (with RW delay)	t_{46}SR	–	-4 + t_C	–	2TCL - 32 + t_C	ns
$\overline{RdCS}$ to Valid Data In (no RW delay)	t_{47}SR	–	10 + t_C	–	3TCL - 32 + t_C	ns
$\overline{RdCS}$, $\overline{WrCS}$ Low Time (with RW delay) [3]	t_{48}CC	14 + t_C - t_W	–	2TCL - 14 + t_C - t_W	–	ns
$\overline{RdCS}$, $\overline{WrCS}$ Low Time (no RW delay) [3]	t_{49}CC	30 + t_C - t_W	–	3TCL - 12 + t_C - t_W	–	ns
Data valid to $\overline{WrCS}$	t_{50}CC	13 + t_C - t_W	–	2TCL - 15 + t_C - t_W	–	ns

Table 100 AC Characteristics, Multiplexed Bus (cont'd)

Parameter	Symbol	Max. CPU Clock 36 MHz (TCL=14ns)		Variable CPU Clock 5 to 36 MHz		Unit
		min.	max.	min.	max.	
Data hold after $\overline{\text{RdCS}}$	t_{51}SR	0	–	0	–	ns
Data float after $\overline{\text{RdCS}}$	t_{52}SR	–	7 + t_F	–	2TCL - 21 + t_F	ns
Address hold after $\overline{\text{RdCS}}$, $\overline{\text{WrCS}}$ [3]	t_{54}CC	8 + t_F + t_W	–	2TCL - 20 + t_F + t_W	–	ns
Data hold after $\overline{\text{WrCS}}$	t_{56}CC	8 + t_F + t_W	–	2TCL - 20 + t_F + t_W	–	ns

[1] Normal (latched) $\overline{\text{CS}}$: bit CSCFG, register SYSCON.6, is set to '0', latched mode is selcted.

[2] Early (unlatched) $\overline{\text{CS}}$; (bit CSCFG = '1') while address bus is changing spikes may occur on $\overline{\text{CS}}$ in this mode.

[3] The memory cycle variable t_W applies only for write accesses; for read accesses this variable is always zero.

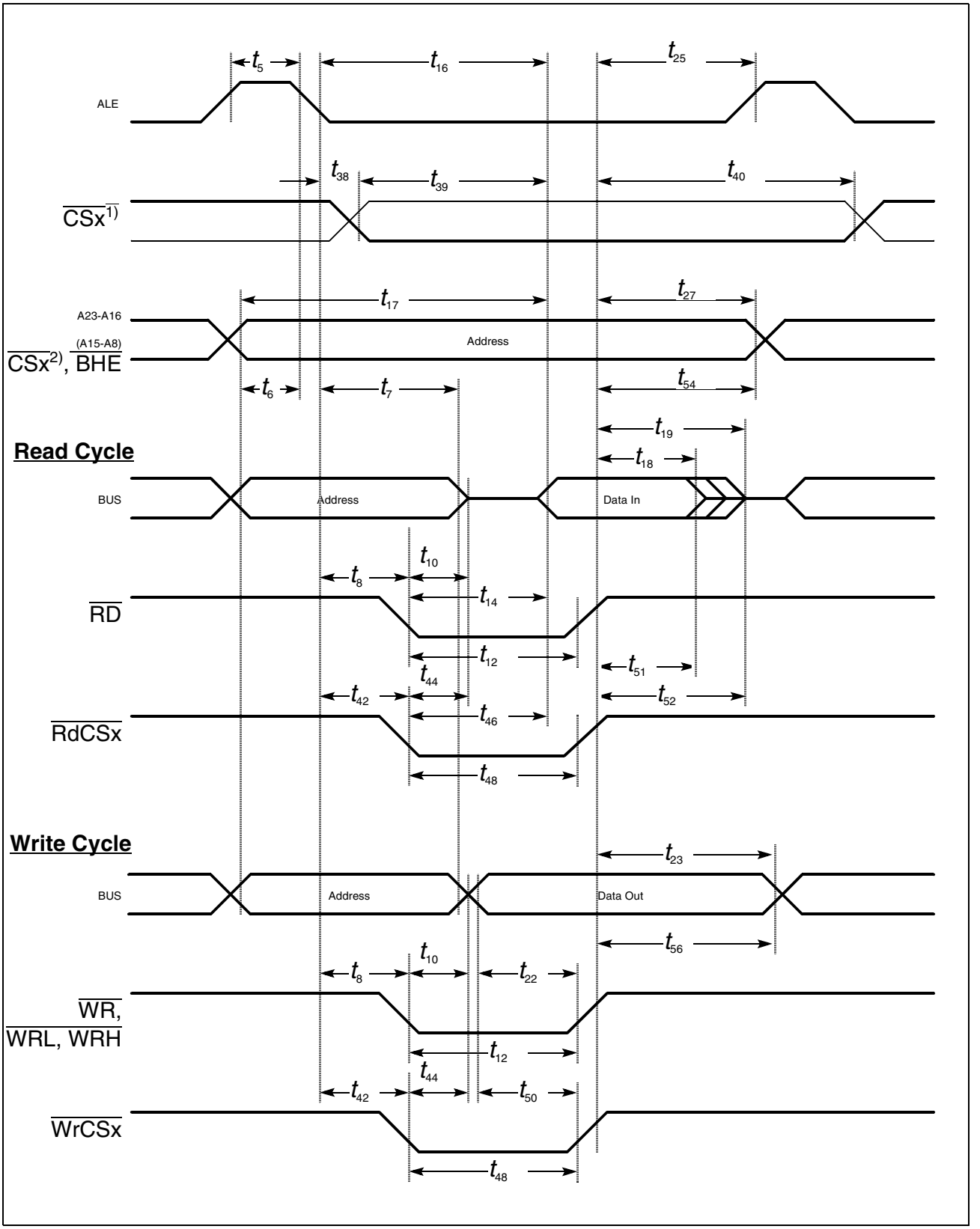


Figure 130 External Memory Cycle: Multiplexed Bus, With Read/Write Delay, Normal ALE

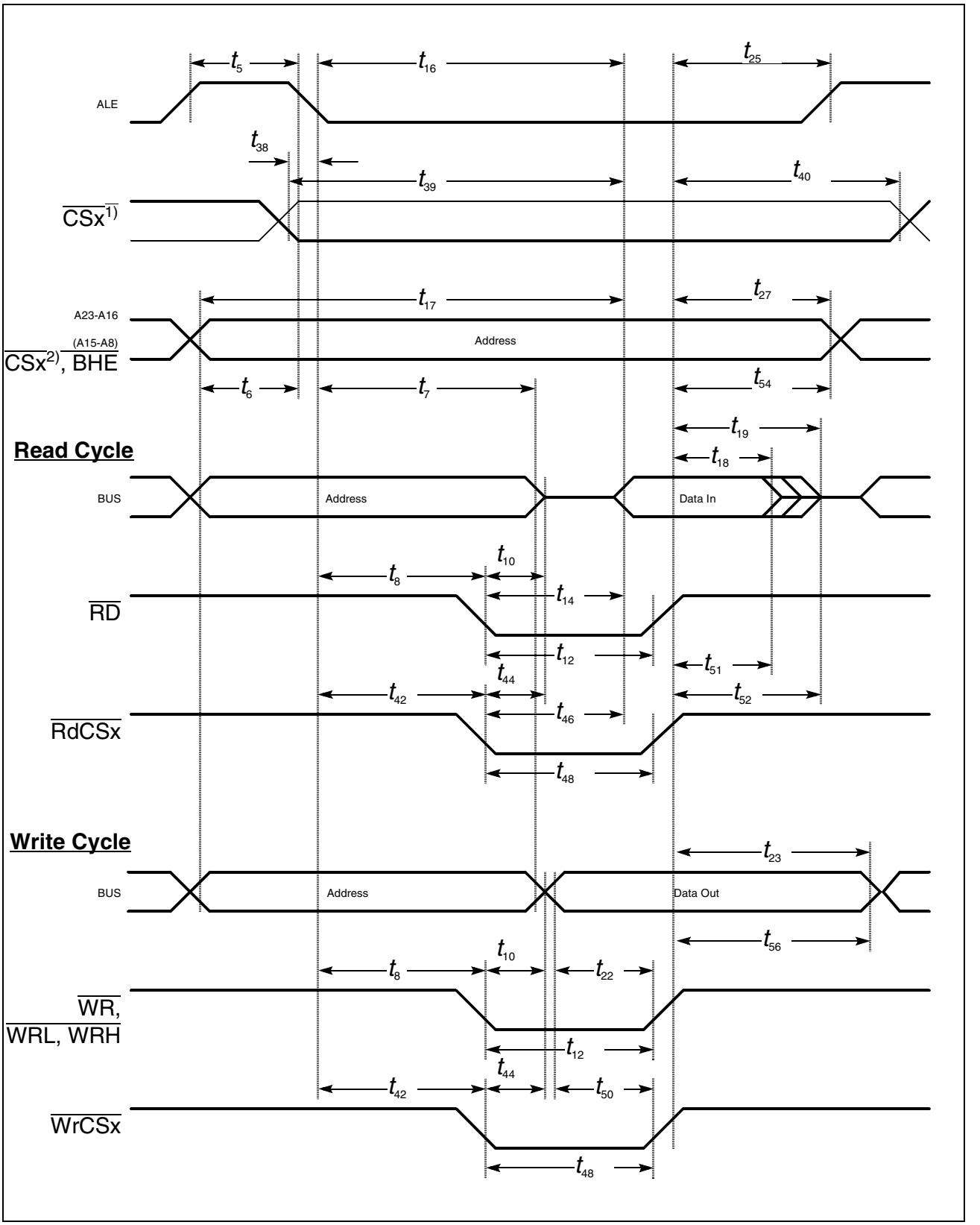


Figure 131 External Memory Cycle: Multiplexed Bus, With Read/Write Delay, Extended ALE

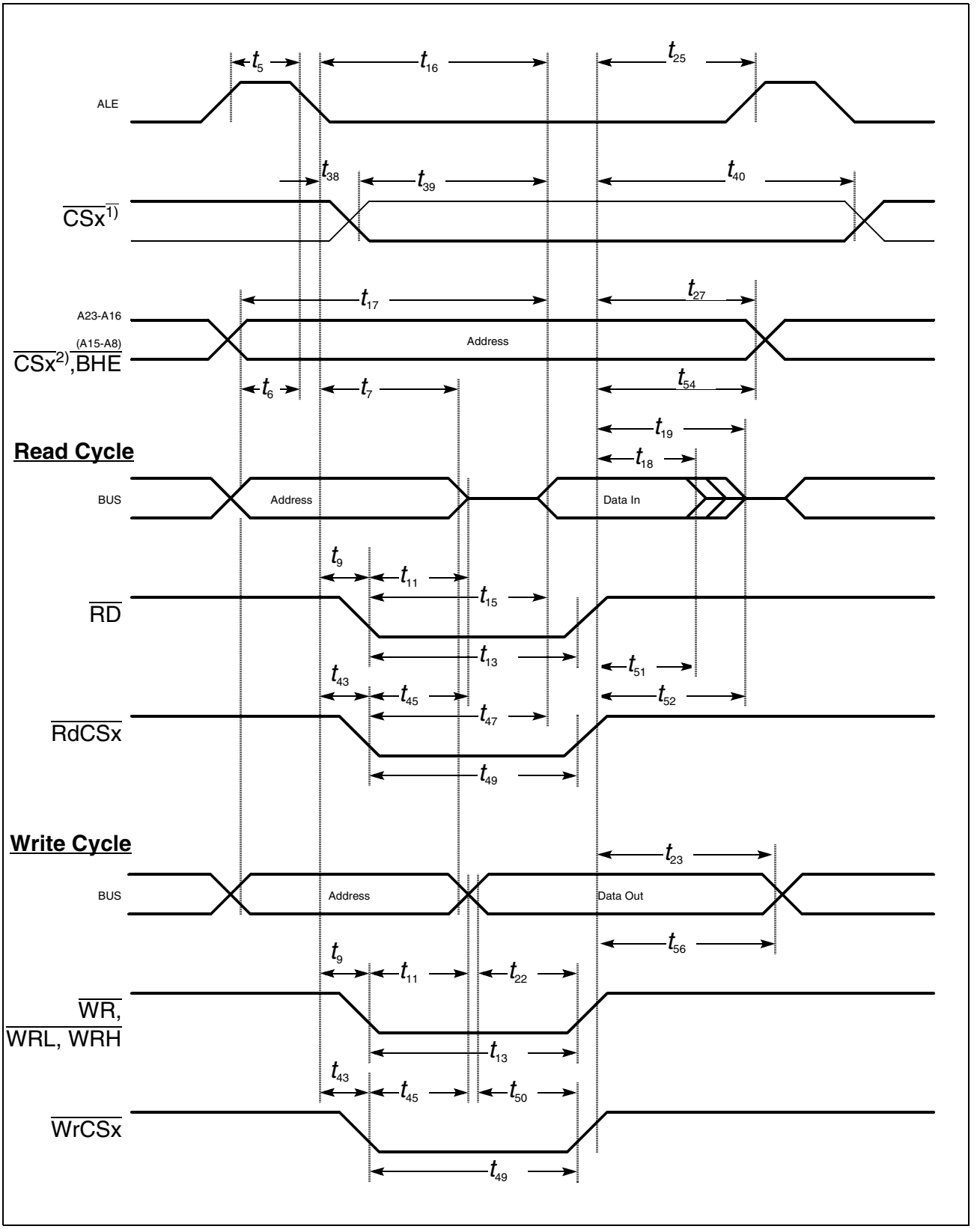


Figure 132 External Memory Cycle: Multiplexed Bus, No Read/Write Delay, Normal ALE

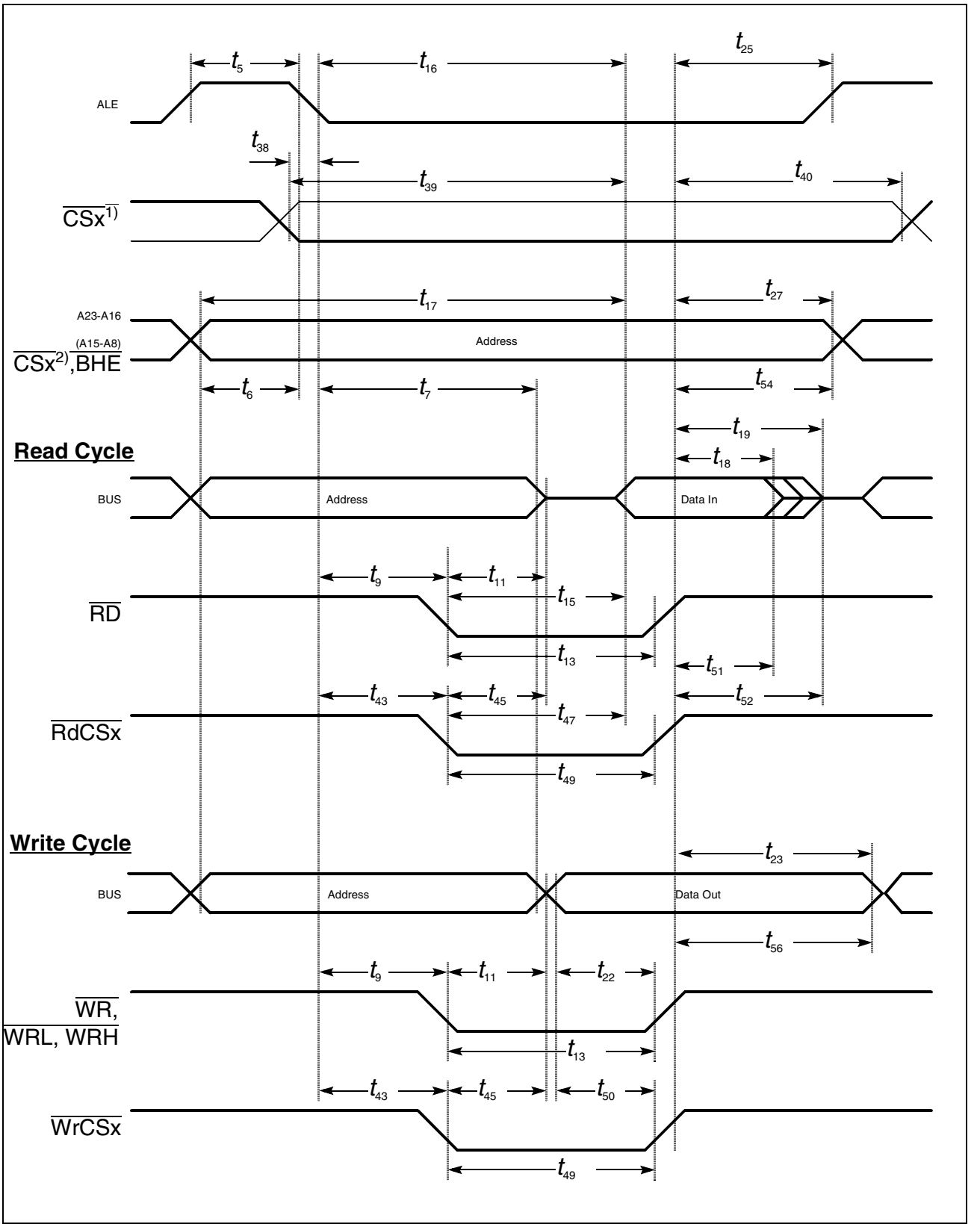


Figure 133 External Memory Cycle: Multiplexed Bus, No Read/Write Delay, Extended ALE

24.8.1.2 AC Characteristics, Demultiplexed Bus

V_{DD} = 3.3 V ± 10 %; V_{SS} = 0 V
T_A = -40 to +85 °C
C_L (for PORT0, PORT1, Port 4, ALE, $\overline{RD}$, $\overline{WR}$, $\overline{BHE}$, CLKOUT) = 100 pF
C_L (for Port 6, $\overline{CS}$) = 100 pF
ALE cycle time = 4 TCL + 2t_A + t_C + t_F (55.5 ns at 36 MHz CPU clock without waitstates)

Table 101 AC Characteristics, Demultiplexed Bus

Parameter	Symbol	Max. CPU Clock = 36 MHz (TCL=14ns)		Variable CPU Clock 5 to 36 MHz		Unit
		min.	max.	min.	max.	
ALE high time	t_{5}CC	4 + t_A	–	TCL - 10 + t_A	–	ns
Address, $\overline{CSx}$[4] setup to ALE	t_{6}CC	-6 + t_A	–	TCL - 20 + t_A	–	ns
ALE falling edge to $\overline{RD}$, $\overline{WR}$ (with RW-delay)	t_{8}CC	4 + t_A	–	TCL - 10 + t_A	–	ns
ALE falling edge to $\overline{RD}$, $\overline{WR}$ (no RW-delay)	t_{9}CC	-10 + t_A	–	-10 + t_A	–	ns
$\overline{RD}$, $\overline{WR}$ low time (with RW-delay)[6]	t_{12}CC	17 + t_C - t_W	–	2TCL - 11 + t_C - t_W	–	ns
$\overline{RD}$, $\overline{WR}$ low time (no RW-delay)[6]	t_{13}CC	31 + t_C - t_W	–	3TCL - 11 + t_C - t_W	–	ns
$\overline{RD}$ to valid data in (with RW-delay)	t_{14}SR	–	0 + t_C	–	2TCL - 28 + t_C	ns
$\overline{RD}$ to valid data in (no RW-delay)	t_{15}SR	–	13 + t_C	–	3TCL - 29 + t_C	ns
ALE low to valid data in	t_{16}SR	–	13 + t_A + t_C	–	3TCL - 29 + t_A + t_C	ns
Address, $\overline{CSx}$[4] to valid data in	t_{17}SR	–	16 + 2t_A + t_C	–	4TCL - 40 + 2t_A + t_C	ns
Data hold after $\overline{RD}$ rising edge	t_{18}SR	0	–	0	–	ns
Data float after $\overline{RD}$ rising edge (with RW-delay [1])	t_{20}SR	–	14 + t_F	–	2TCL - 14 + 2t_A + t_F [1]	ns

Table 101 AC Characteristics, Demultiplexed Bus (cont'd)

Parameter	Symbol	Max. CPU Clock = 36 MHz (TCL=14ns)		Variable CPU Clock 5 to 36 MHz		Unit
		min.	max.	min.	max.	
Data float after $\overline{RD}$ rising edge (no RW-delay [1])	t_{21}SR	–	4 + t_F	–	TCL - 10 + 2t_A + t_F [1]	ns
Data valid to $\overline{WR}$	t_{22}CC	13 + t_C - t_W	–	2TCL - 15 + t_C - t_W	–	ns
Data hold after $\overline{WR}$	t_{24}CC	4 + t_F + t_W	–	TCL - 10 + t_F + t_W	–	ns
ALE rising edge after $\overline{RD}$, $\overline{WR}$ [6]	t_{26}CC	0 + t_F + t_W	–	0 + t_F + t_W	–	ns
Address, $\overline{CSx}$ [4] hold after $\overline{WR}$ [2] [5]	t_{28}CC	0 + t_F + t_W	–	0 + t_F + t_W	–	ns
ALE falling edge to $\overline{CSx}$ [3]	t_{38}CC	-9 - t_A	13 - t_A	-9 - t_A	13 - t_A	ns
$\overline{CSx}$ [3] low to Valid Data In	t_{39}SR	–	12 + t_C + 2t_A	–	3TCL - 30 + t_C+2t_A	ns
$\overline{CSx}$ [3] hold after $\overline{RD}$, $\overline{WR}$ [6]	t_{41}CC	0 + t_F + t_W	–	TCL - 14 + t_F + t_W	–	ns
ALE falling edge to $\overline{RdCS}$, $\overline{WrCS}$ (with RW-delay)	t_{42}CC	5 + t_A	–	TCL - 9 + t_A	–	ns
ALE falling edge to $\overline{RdCS}$, $\overline{WrCS}$ (no RW-delay)	t_{43}CC	-9 + t_A	–	-9 + t_A	–	ns
$\overline{RdCS}$ to Valid Data In (with RW-delay)	t_{46}SR	–	-4 + t_C	–	2TCL - 32 + t_C	ns
$\overline{RdCS}$ to Valid Data In (no RW-delay)	t_{47}SR	–	10 + t_C	–	3TCL - 32 + t_C	ns
$\overline{RdCS}$, $\overline{WrCS}$ Low Time (with RW-delay) [6]	t_{48}CC	14 + t_C - t_W	–	2TCL - 14 + t_C - t_W	–	ns
$\overline{RdCS}$, $\overline{WrCS}$ Low Time (no RW-delay) [6]	t_{49}CC	30 + t_C - t_W	–	3TCL - 12 + t_C - t_W	–	ns

Table 101 AC Characteristics, Demultiplexed Bus (cont'd)

Parameter	Symbol	Max. CPU Clock = 36 MHz (TCL=14ns)		Variable CPU Clock 5 to 36 MHz		Unit
		min.	max.	min.	max.	
Data valid to $\overline{WrCS}$	t_{50}CC	13 + t_C - t_W	–	2TCL - 15 + t_C - t_W	–	ns
Data hold after $\overline{RdCS}$	t_{51}SR	0	–	0	–	ns
Data float after $\overline{RdCS}$ (with RW-delay [1])	t_{53}SR	–	7 + t_F	–	2TCL - 21 + t_F	ns
Data float after $\overline{RdCS}$ (no RW-delay [1])	t_{68}SR	–	0 + t_F	–	TCL - 14 + t_F	ns
Address hold after $\overline{WrCS}$ [2]	t_{55}CC	-14 + t_F + t_W	–	- 14 + t_F + t_W	–	ns
Data hold after $\overline{WrCS}$	t_{57}CC	0 + t_F + t_W	–	TCL - 14 + t_F + t_W	–	ns

[1] RW-delay and t_A refer to the next following bus cycle.

[2] Read data are latched with the same clock edge that triggers the address change and the rising $\overline{RD}$, $\overline{RDCS}$ edge. Therefore address changes before the end of $\overline{RD}$, $\overline{RDCS}$ have no impact on read cycles.

[3] Normal (latched) $\overline{CS}$: **bit CSCFG, register SYSCON.6, is set to '0', latched mode is selcted.**

[4] Early (unlatched) $\overline{CS}$ **(bit CSCFG = '1')**; while address bus is changing spikes may occur on $\overline{CS}$ in this mode.

[5] Demultiplexed cycles: In case of early (unlatched) $\overline{CS}$ together with normal write (not early write) $\overline{CS}$ may go inactive 3 ns before the rising edge of $\overline{WR}$.

[6] The memory cycle variable t_W applies only for write accesses; for read accesses this variable is always zero.

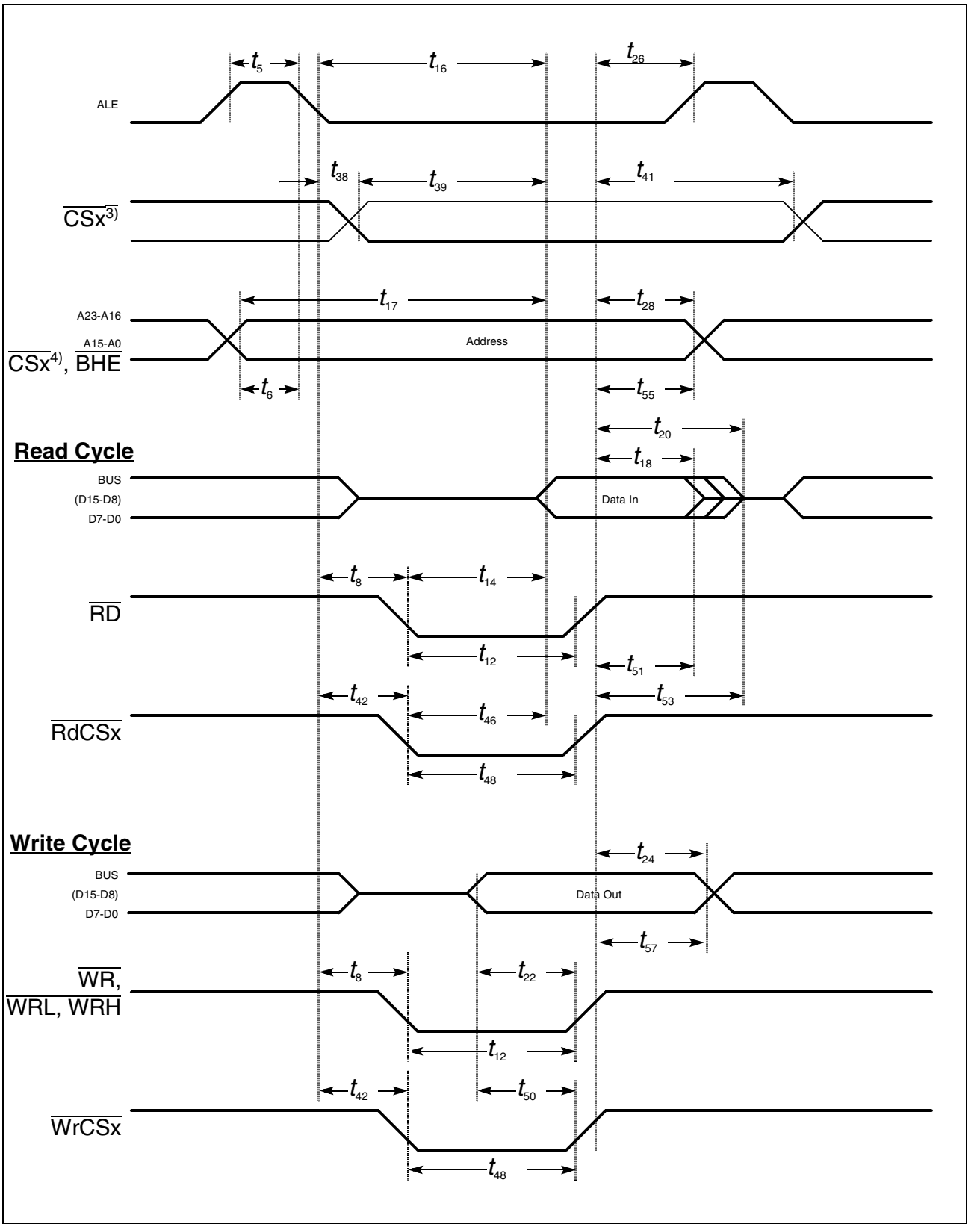


Figure 134 External Memory Cycle: Demultiplexed Bus, With Read/Write Delay, Normal ALE

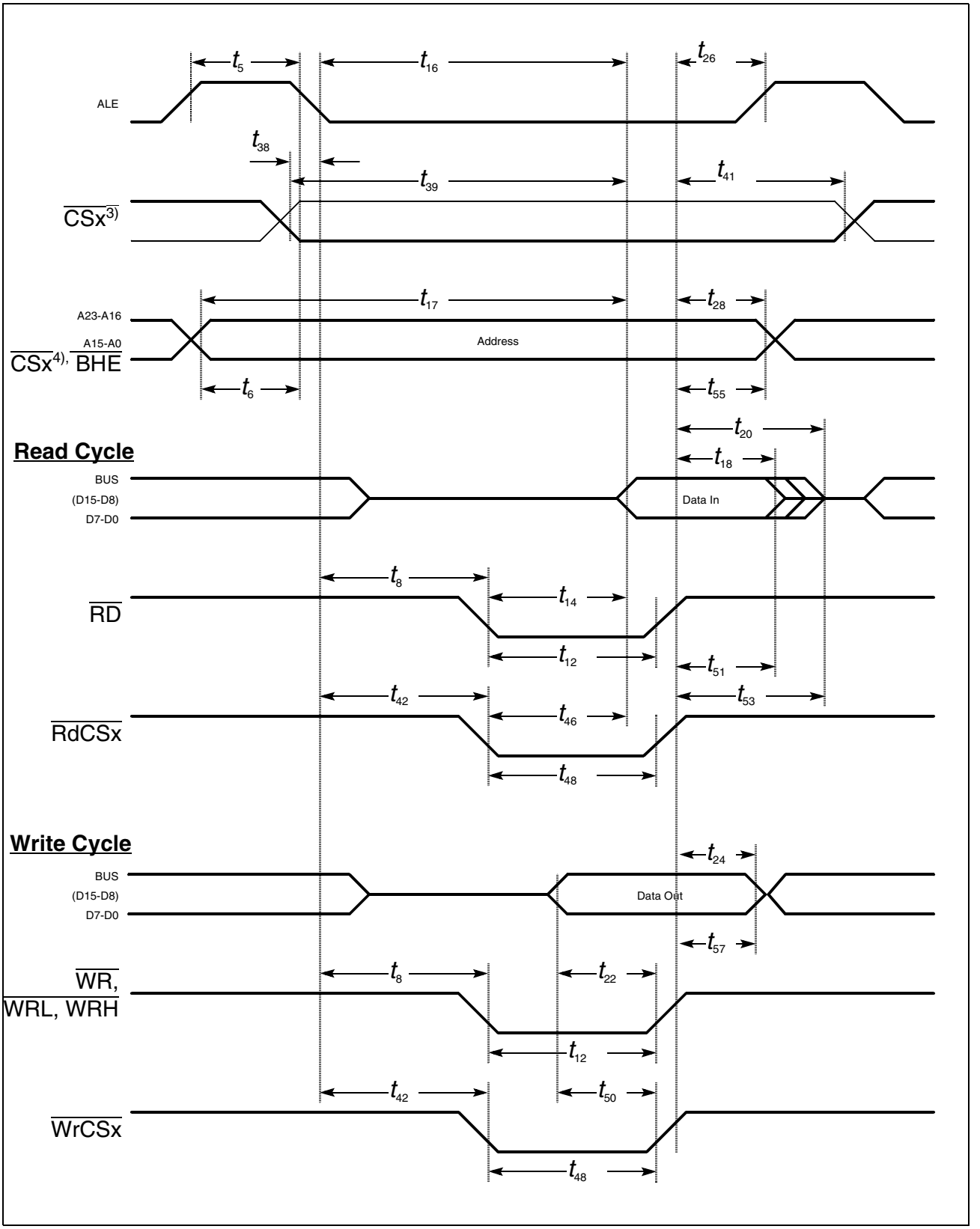


Figure 135 External Memory Cycle: Demultiplexed Bus, With Read/Write Delay, Extended ALE

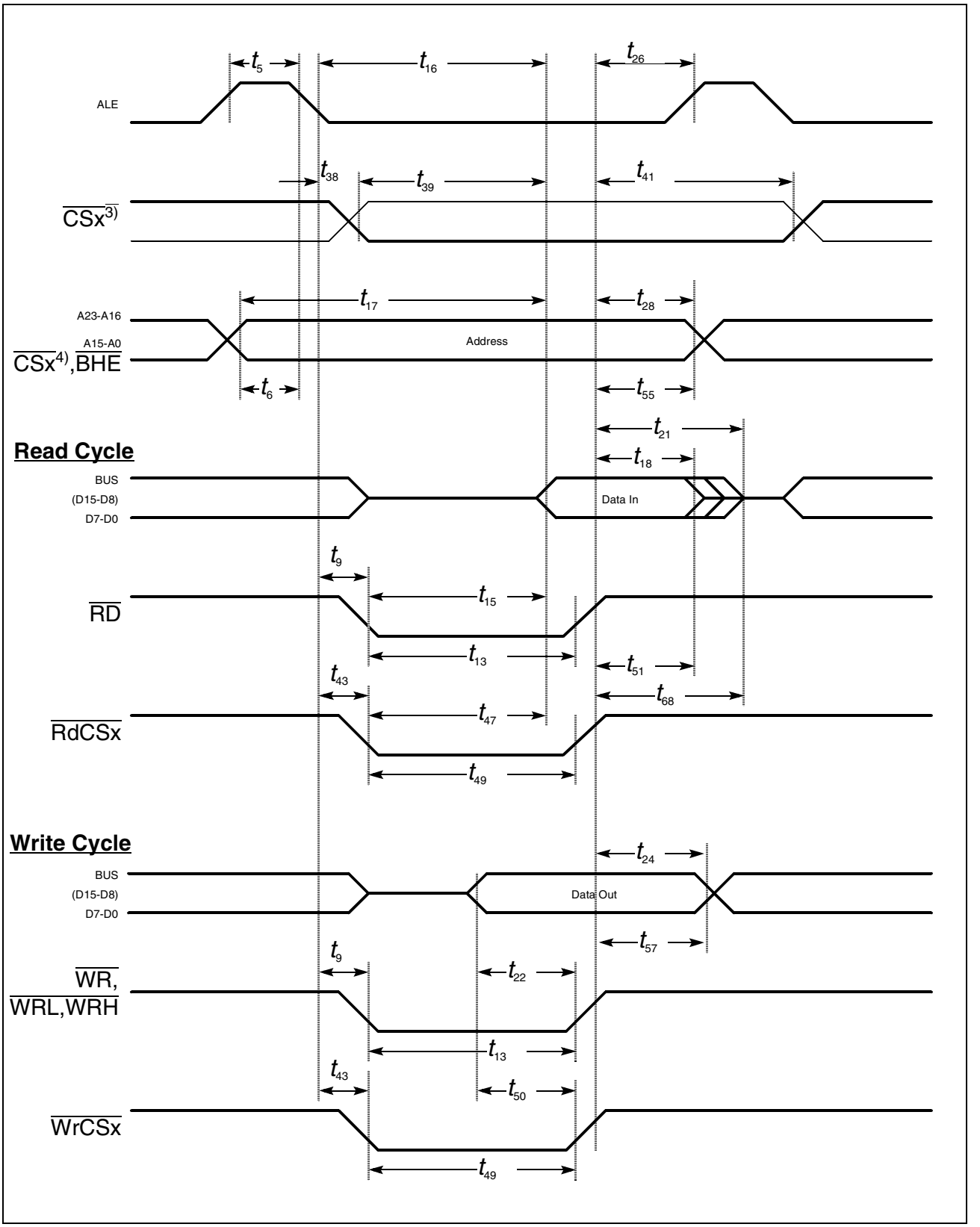


Figure 136 External Memory Cycle: Demultiplexed Bus, No Read/Write Delay, Normal ALE

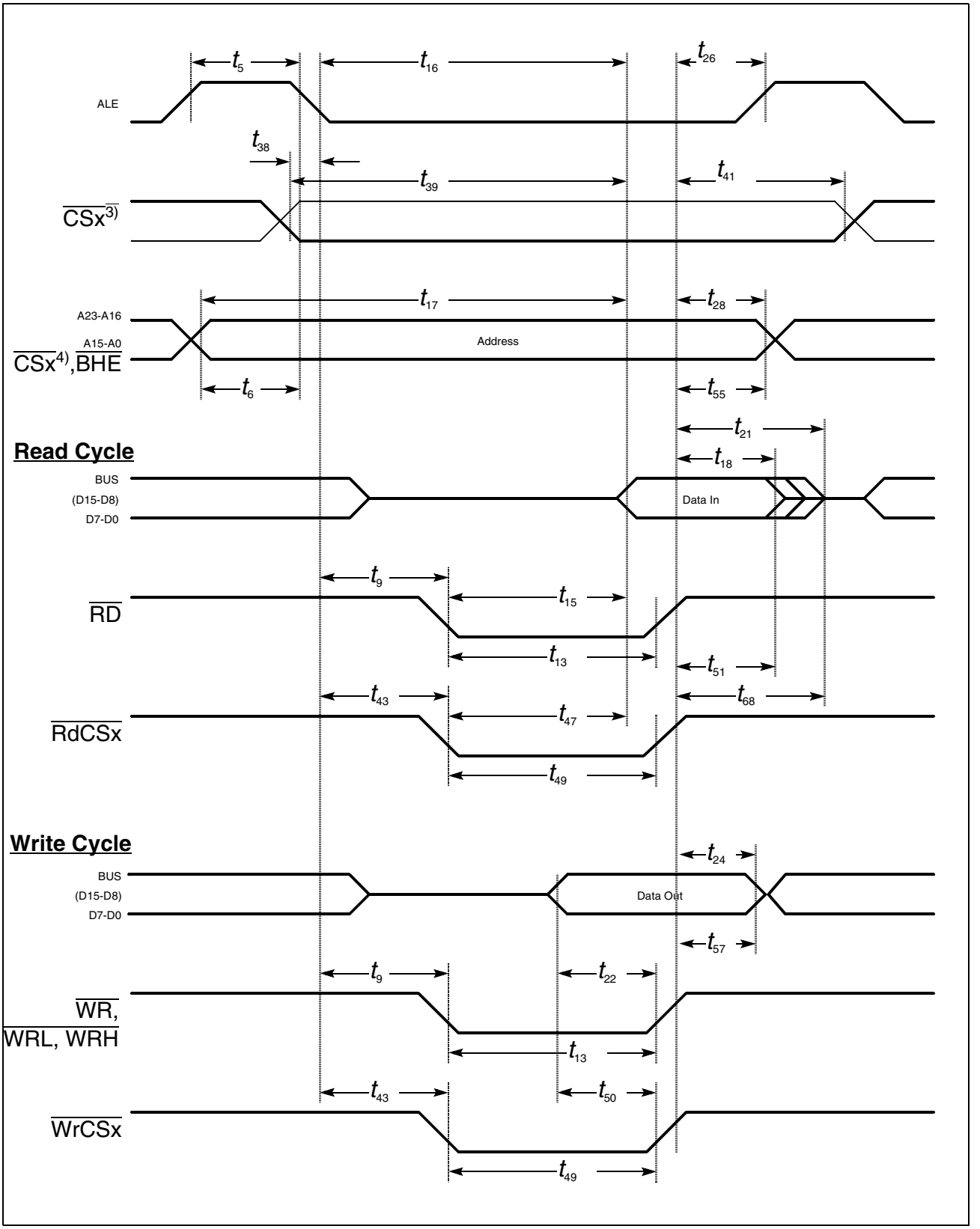


Figure 137 External Memory Cycle: Demultiplexed Bus, No Read/Write Delay, Extended ALE

24.8.1.3 AC Characteristics, CLKOUT and $\overline{\text{READY}}$

V_{DD} = 3.3 V ± 10 %; V_{SS} = 0 V
T_A = -40 to +85 °C
C_L (for PORT0, PORT1, Port 4, ALE, $\overline{\text{RD}}$, $\overline{\text{WR}}$, $\overline{\text{BHE}}$, CLKOUT, $\overline{\text{READY}}$) = 100 pF
C_L (for Port 6, $\overline{\text{CS}}$) = 100 pF

Note: Timing parameters for 36-MHz clock are based on the assumption that the oscillator is running at 8 MHz and PLL with F = 4.5.

Table 102 AC Characteristics, CLKOUT and $\overline{\text{READY}}$

Parameter	Symbol	CPU Clock 36 MHz (TCL = 14 ns)		Variable CPU Clock 5 to 36 MHz		Unit
		min.	max.	min.	max.	
CLKOUT cycle time	t_{29CC}	27	29	2TCL - 1	2TCL + 1	ns
CLKOUT high time	t_{30CC}	8	–	TCL – 6	–	ns
CLKOUT low time	t_{31CC}	4	–	TCL – 10	–	ns
CLKOUT rise time	t_{32CC}	–	4	–	4	ns
CLKOUT fall time	t_{33CC}	–	4	–	4	ns
CLKOUT rising edge to ALE falling edge	t_{34CC}	0 + t_A	16 + t_A	0 + t_A	16 + t_A	ns
Synchronous $\overline{\text{READY}}$ setup time to CLKOUT	t_{35SR}	18	–	18	–	ns
Synchronous $\overline{\text{READY}}$ hold time after CLKOUT	t_{36SR}	0	–	0	–	ns
Asynchronous $\overline{\text{READY}}$ low time	t_{37SR}	45	–	2TCL + 17	–	ns
Asynchronous $\overline{\text{READY}}$ setup time [1]	t_{58SR}	18	–	18	–	ns
Asynchronous $\overline{\text{READY}}$ hold time [1]	t_{59SR}	0	–	0	–	ns
Async. $\overline{\text{READY}}$ hold time after $\overline{\text{RD}}$, $\overline{\text{WR}}$ high (Demultiplexed Bus) [2]	t_{60SR}	0	-10 + 2t_A + t_C + t_F [2]	0	TCL - 24 + 2t_A + t_C + t_F [2]	ns

[1] These timings are given for test purposes only, in order to assure recognition at a specific clock edge.

2) Demultiplexed bus is the worst case. For multiplexed bus, 2 TCL is to be added to the maximum values. This adds even more time for deactivating READY.

Note: The $2t_A$ and t_C refer to the next following bus cycle, t_F refers to the current bus cycle.

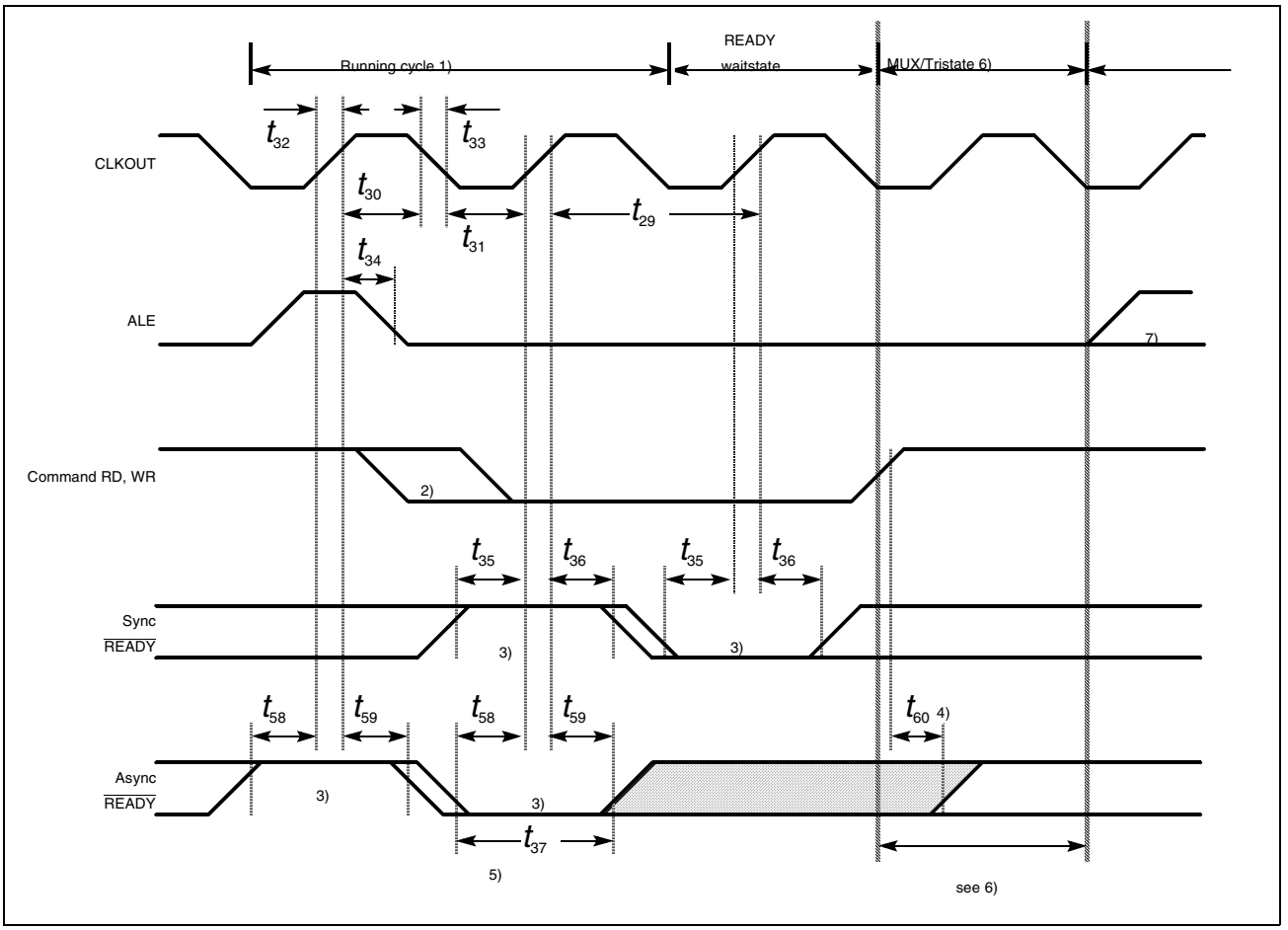


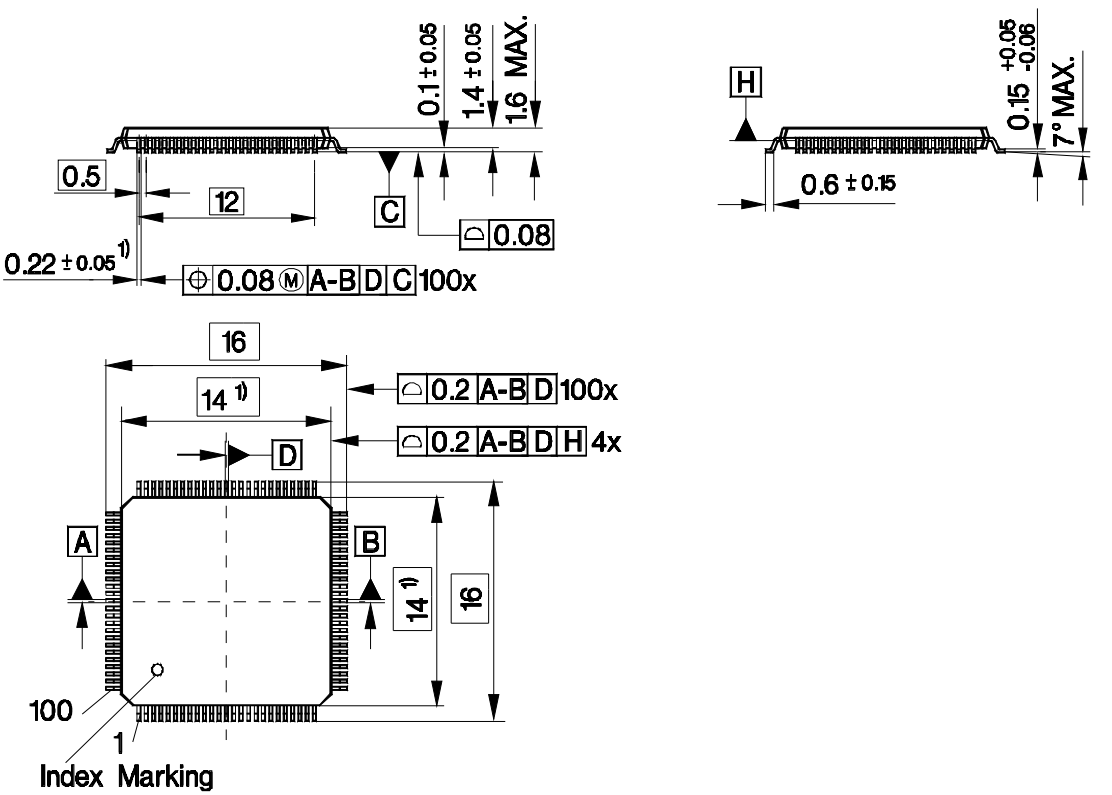
Figure 138 CLKOUT and READY

1) Cycle as programmed, including MCTC wait states (Example shows 0 MCTC WS).
2) The leading edge of the respective command depends on RW-delay.
3) READY sampled HIGH at this sampling point generates a READY controlled wait state. READY sampled LOW at this sampling point terminates the bus cycle currently running.
4) READY may be deactivated in response to the trailing (rising) edge of the corresponding command (RD or WR).
5) If the Asynchronous READY signal does not fulfill the indicated setup and hold times with respect to CLKOUT (eg. because CLKOUT is not enabled), it must fulfill t_{37} in order to be safely synchronized. This is guaranteed if READY is removed in reponse to the command (see Note 4)).
6) Multiplexed bus modes have a MUX wait state added after a bus cycle, and an additional MTTC wait state may be inserted here. For a multiplexed bus with MTTC wait state, this delay is 2 CLKOUT cycles, for a demultiplexed bus without MTTC wait state this delay is zero.
7) The next external bus cycle may start here.

25 Package Outline

P-TQFP-100

(Plastic Thin Quad Flat Package)



1) Does not include plastic or metal protrusion of 0.25 max. per side

2) Does not include dambar protrusion of 0.08 max. per side at max. material condition

Infineon goes for Business Excellence

“Business excellence means intelligent approaches and clearly defined processes, which are both constantly under review and ultimately lead to good operating results.
Better operating results and business excellence mean less idleness and wastefulness for all of us, more professional success, more accurate information, a better overview and, thereby, less frustration and more satisfaction.”

Dr. Ulrich Schumacher

http://www.infineon.com

Published by Infineon Technologies AG