ST10R272L

USER'S MANUAL

Release 1.0

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

1 INTRODUCTION

The ST10R272L is a new member of the ST10 family designed to give high performance in applications where low-power operation is important.

The low-power features of the ST10R272L include:

- **3.3V operation:** Half the power consumption of an equivalent 5V device (250mW at 25MHz).
- **Idle Mode:** Consumes less than 100mW at 25MHz.
- **Power-Down Mode:** Less than 170uW.
- **Static Design:** Operates at a lower clock speed for lower power consumption.

The Performance Features of the ST10R272Linclude:

- **Operation up to 50MHz:** Double the performance of existing ST10/C166 devices
- **On-chip DSP co-processor:** Up to 25Mips DSP performance

The ST10R272L is fully software compatible with other members of the ST10 and C166 families. Most designs using R163 or R165 type devices can be adapted for the R272L with minimal change, giving the immediate benefits of lower power consumption and higher performance.

This manual describes the functionality of the ST10R272L. The architectural overview gives a general description of the ST10R272L device. It describes in brief, the CPU performance, the on-chip system resources, the clock generator, the peripheral blocks and the protected bits.

The detailed operation of the CPU, the MAC and the on-chip peripherals - parallel ports, external bus interface, peripheral event controller, general purpose timers, serial interfaces, watchdog timer, bootstrap loader, capcom units, pulse width module and the analog-digital converter - are contained in the subsequent chapters. The operating modes - system reset, power reduction, interrupt handling and system programming - are described in detail, in separate chapters.The special function registers are listed by both name and hexadecimal address. The instruction set is covered in the 'ST10 Family Programming Manual' and is not described in this manual.

The DC and AC electrical characteristics and the pin description for each available package are covered in the device Data Sheet and Errata Sheet, and are not described in this manual. Before starting on a new design, verify the device characteristics with an up-to-date copy of the device Data Sheet and Errata Sheet. An index of registers is contained at the back of the manual.

2 ARCHITECTURAL OVERVIEW

The architecture of the ST10R272L combines the advantages of both RISC and CISC processors with an advanced peripheral subsystem. The following block diagram shows the different on-chip components and the advanced - high bandwidth - internal bus structure of the ST10R272L.

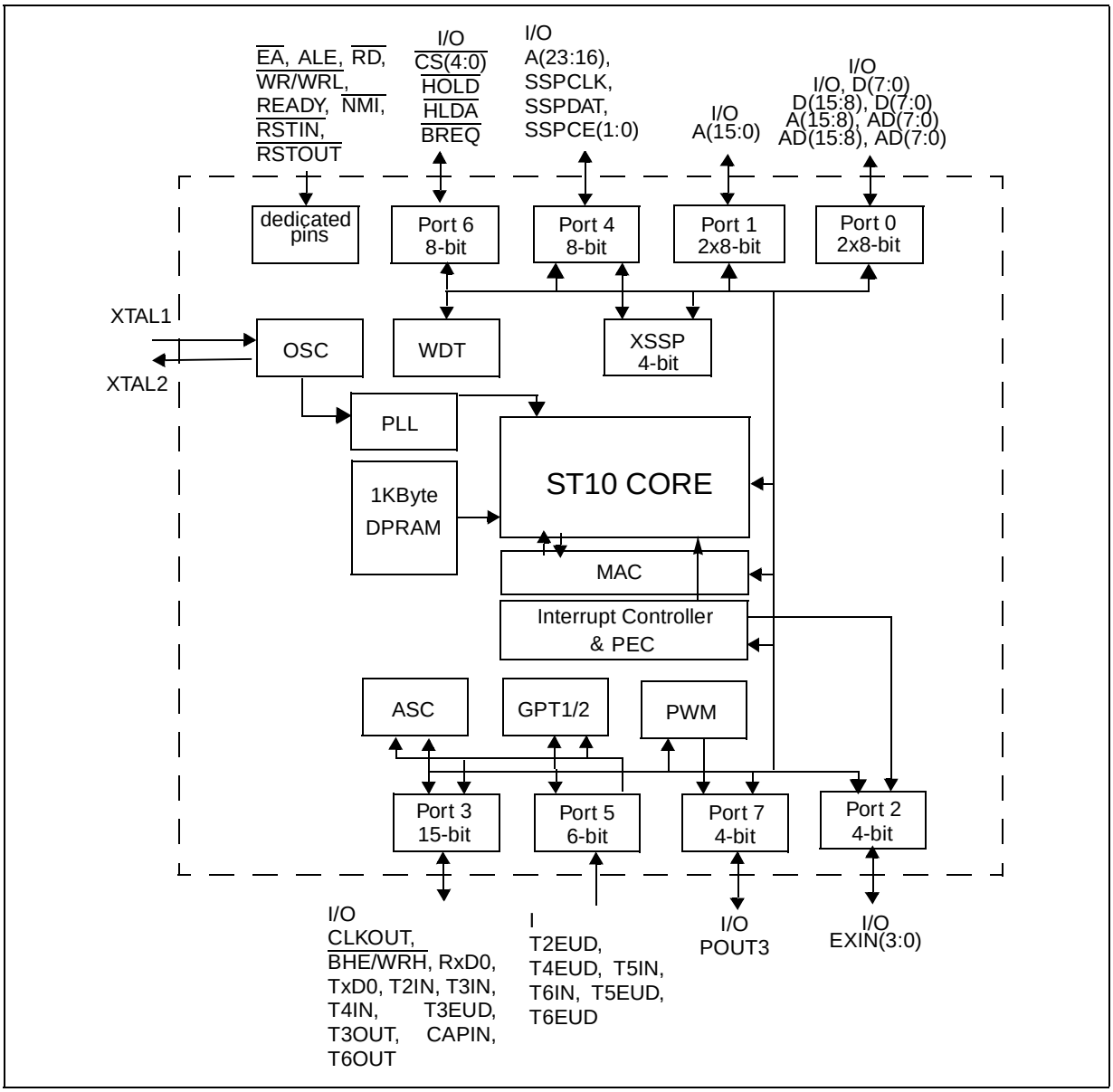


Figure 1 ST10R272L block diagram

2.1 Basic CPU concepts

The main core of the CPU contains a 4-stage instruction pipeline, a MAC multiply-accumulation unit, a separate multiply and divide unit, a bit-mask generator and a barrel shifter. Most instructions can be executed in one CPU clock cycle.

The CPU includes an actual register context, consisting of 16 wordwide GPRs physically located in 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. The number of register banks is only restricted by the available internal RAM space. For easy parameter passing, one register bank may overlap others.

A system stack of up to 1024 bytes stores temporary data. The system stack is allocated in the on-chip RAM area, and is accessed by the CPU, via the stack pointer (SP) register. Two separate SFRs, STKOV and STKUN, are compared against the stack pointer value during each stack access, to detect stack overflow or underflow.

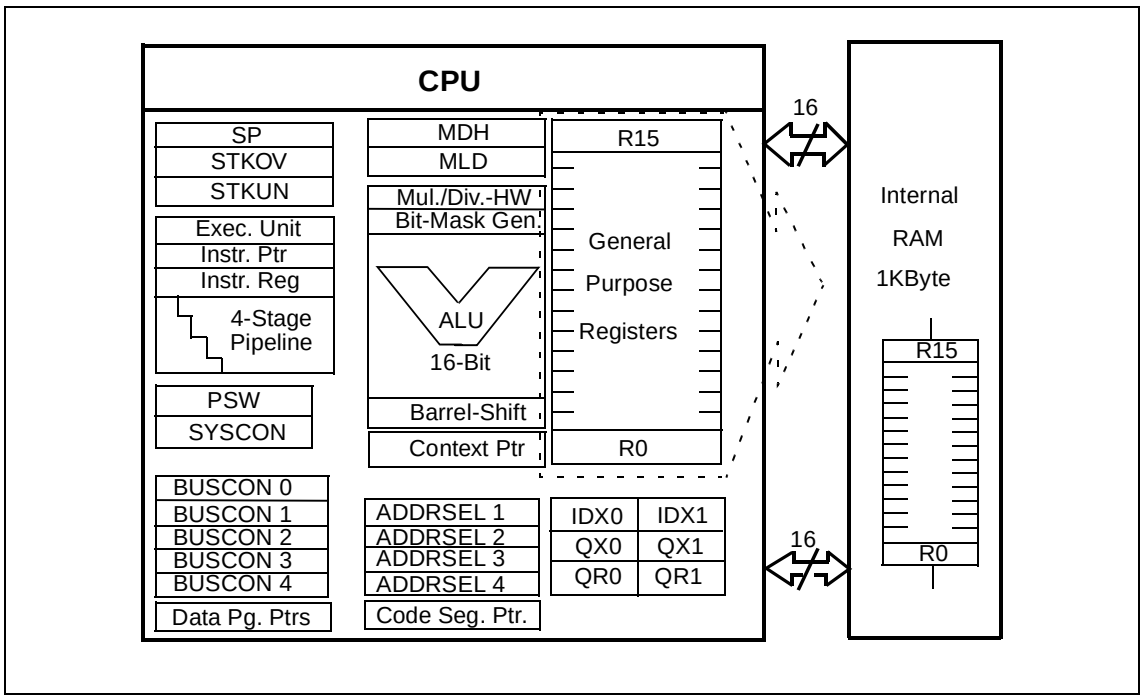


Figure 2 CPU block diagram

The MAC improves the performance of signal processing algorithms. It provides non-pipelined single-cycle instructions - including 32-bit signed arithmetic (addition, subtraction, shift,...), 16 by 16 multiplication, and multiplication with cumulative subtraction/ addition.

2.1.1 High instruction bandwidth / fast execution

Most of the ST10R272L's instructions can be executed in just one instruction cycle - shift and rotate instructions (irrespective of the number of bits to be shifted). Branch, multiply and divide instructions normally take more than one instruction cycle, but have also been optimized. For example, branch instructions require only one additional machine cycle when a branch is taken, and because of the 'Jump Cache', most branches taken in loops require no additional machine cycles.

The four-stage pipeline improves CPU processing speed:

fetch: an instruction is fetched from the internal ROM or RAM or from the external memory, based on the current IP value

decode: the previously fetched instruction is decoded and the required operands are fetched

execute: the specified operation is performed on the previously fetched operands

write back: the result is written to the specified location

If this technique were not used, each instruction would require four machine cycles.

2.1.2 High function 8-bit and 16-bit arithmetic and logic unit

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 a signal to be held in the control registers. Multiple-cycle instructions are performed through instruction injection and simple internal state machines which modify the required control signals.

All standard arithmetic and logical operations are performed in the 16-bit ALU. For byte operations, signals are provided from bits six and seven of the ALU result, and are used to 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 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 allows four bits to be multiplied and two bits to be divided per machine cycle. Therefore, 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. The longer multiply and divide instructions can be interrupted during their execution, to permit fast interrupt response. The Instruction Set contains instructions for byte packing in memory, and byte and word sign extension for wide arithmetic operations.

A set of consistent flags is automatically updated in the PSW after each arithmetic, logical, shift, or movement operation. These flags cause branching on specific conditions. User-specifiable branch tests give support for signed and unsigned arithmetic. These flags are preserved automatically by the CPU on entry into an interrupt or trap routine. All targets for branch calculations are computed in the central ALU.

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

2.1.3 Extended bit processing and peripheral control

A large number of instructions have been dedicated to bit processing, to provide efficient control and testing of peripherals while enhancing data manipulation. These instructions provide direct access to two operands in the bit-addressable space, without using temporary flags.

The same logical instructions that are available for words and bytes, are also supported for bits. A peripheral control bit can be compared and modified in one instruction. Multiple bit shift instructions avoid long instruction streams of single bit shift operations. These are also performed in a single machine cycle.

Bit field instructions can modify multiple bits from one operand, in a single instruction.

2.1.4 High performance branch, call, and loop processing

Branch instructions only require one extra machine cycle when a branch is taken, because the target address is pre-calculated while decoding the instruction. To decrease loop execution overhead, three enhancements have been provided:

- Single cycle branch execution is perfomed after the first iteration of a loop. Therefore, only one machine cycle is lost during the execution of the entire loop. In loops which fall through on 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.
- End-of-table detection avoids the use of two compare instructions embedded in loops. The lowest negative number is placed at the end of the table and branching is specified if neither this value nor the compared value have been found. The loop is terminated if either condition is 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 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, which is useful in table searching.

The system state is automatically saved on the internal system stack . Instructions are not required 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 been added to support indirect branch and call instructions, supporting the implementation of multiple CASE statement branching in assembler macros and high level languages.

2.1.5 Instruction formats

The ST10 Family instruction set has two parts: a standard instruction set which is used for all ST10 Family products, and a MAC instruction set which is used for products containing the MAC. Code compiled for the non-MAC ST10/C166 processors, will run on the ST10R272L. However, MAC instructions run on non-MAC ST10/C166 processors will be trapped because undefined opcodes are used. The instruction set is described in the **ST10 Family Programming Manual**.

The standard instruction set has different addressing modes to access word, byte and bit data. Specific instructions support the conversion (extension) of bytes to words. A variety of direct, indirect or immediate addressing modes specify operands. ATOMIC and EXTended instructions disable standard and PEC interrupts and class A traps.

The MAC instruction set uses some standard ST10 addressing modes such as GPR or #data4 for immediate shift value. New MAC instruction addressing modes supply the MAC with up to 2 new operands per instruction cycle. These allow indirect addressing with address pointer post-modification. Double indirect addressing requires 2 pointers, one can be supplied by any GPR, the other is provided by one of two new specific SFRs IDX0 and IDX1. Two pairs of offset registers QR0/QR1 and QX0/QX1 are associated with each pointer (GPR or IDX_i). The GPR pointer gives access to the entire memory space, whereas IDX_i are limited to the internal Dual-Port RAM, except for the CoMOV instruction.

2.1.6 Programmable multiple priority interrupt system

There are several response mechanisms to service requests generated from various sources, internal or external to the microcontroller. Any of these interrupt requests can be programmed to be serviced, either by the Interrupt Controller, or by the Peripheral Event Controller (PEC).

Fast external-interrupt inputs service external-interrupts with high-precision requirements. These fast interrupt inputs, feature programmable edge detection (rising edge, falling edge or both edges).

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

Hardware-traps cause immediate non-maskable system reaction, similar to a standard-interrupt service (branching to a dedicated vector table location). The occurrence

of a hardware trap is, additionally, signified by an individual bit in the trap flag register (TFR). Except when another higher-priority trap service is in progress, a hardware-trap will interrupt any actual program execution. In turn, hardware-trap services can not, normally be interrupted by a standard-interrupt or PEC-interrupt.

2.2 On-chip system resources

The ST10R272L provides a number of powerful system resources designed around the CPU.

2.2.1 Peripheral event controller (PEC) and interrupt control

The Peripheral Event Controller responds to an interrupt request with a single data-transfer (word or byte) which consumes one instruction cycle, and does not require a ‘save and restore machine-status’. Each interrupt source is prioritized in every machine cycle, in the interrupt control block. If a PEC service is selected, a PEC transfer is started. If a 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 machine-state is saved on the internal system-stack, and the CPU branches to the system specific vector for the peripheral.

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

An individual PEC-transfer counter is implicitly decremented for each PEC service, except when performing in the continuous-transfer mode. When the 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 or from a memory table. The ST10R272L has 8 PEC channels, each of which offers fast interrupt-driven data transfer capabilities.

2.2.2 Memory areas

The memory space of the ST10R272L is configured in a Von Neumann architecture which means that code memory, data memory, registers and IO ports are organized within the same linear address spaces. The entire memory space can be accessed bytewise or wordwise. Particular portions of the on-chip memory are directly bit addressable.

A 1KByte 16-bit wide internal RAM is used for variables, register banks, and the system stack. It also contains the PEC pointers and the bit-addressable space.

The CPU has 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 accessed by the CPU. The

number of register banks is only restricted by the available internal RAM space. For easy parameter passing, one register bank may overlap another.

A system stack of up to 512 words stores temporary data. It is located in the on-chip RAM area, and 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, on 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. It 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 address space are reserved. The standard Special Function Register area (SFR) uses 512 bytes, and the Extended Special Function Register area (ESFR) uses the other 512 bytes. (E)SFRs are word-wide registers used for controlling and monitoring the on-chip units. Unused (E)SFR addresses are reserved for future development.

2.2.3 External bus interface

All external memory accesses are performed by the External Bus Controller (EBC). It can be programmed either to Single-Chip Mode when no external memory is required, or to one of four different external memory access modes:

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

In the demultiplexed bus modes, addresses are output on PORT1, and data is input/output on PORT0 or P0L, respectively. In the multiplexed bus modes, both addresses and data use PORT0 for input/output.

Important timing characteristics of the external bus interface (memory cycle time, memory tri-state time, length of ale and read write delay) have been made programmable, to give the choice of a wide range of different types of memories and external peripherals. In addition, up to 4 independent address windows may be defined (via register pairs ADDRSELx / BUSCONx) to access different resources with different bus characteristics. These address windows are arranged hierarchically where BUSCON4 overrides BUSCON3 and BUSCON2 overrides BUSCON1. All accesses to locations not covered by these 4 address windows are controlled by BUSCON0. Up to 5 external $\overline{CS}$ signals (4 windows plus default) can be generated to save external glue logic. Access to very slow memories is supported by the ‘Ready’ function.

For applications which require less than 16MBytes of external memory space, the address space can be restricted to 1MByte, 256KByte or to 64KByte. Port 4 outputs four, two or no address lines. If an address space of 16MBytes is used, Port 4 outputs all 8 address lines.

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

2.3 System clock generator

The on-chip clock generator provides the basic clock signal that controls the activities of the controller hardware. Its oscillator can run with an external crystal and appropriate oscillator circuitry (refer to “DEDICATED PINS” on page 143) or can be driven by an external oscillator. The oscillator can directly feed the external clock signal to the controller hardware (through buffers) and divides the external clock frequency by 2, or feeds an on-chip phase locked loop (PLL) which multiplies the input frequency by a selectable factor **F**. The resulting internal clock signal is referred to as the ‘CPU clock’. Two separate clock signals are generated for the CPU and the peripherals. While the CPU clock is stopped during idle mode, the peripheral clock keeps running. Both clocks are switched off when power-down mode is entered.

The on-chip PLL circuit allows operation of the ST10R272L on a low frequency external clock while still providing maximum performance. The PLL multiplies the external clock frequency by a selectable factor of 1:F and generates a CPU clock signal with 50% duty cycle. The PLL also provides fail safe mechanisms which detect frequency deviations and the execution of emergency actions in case of an external clock failure. (Refer to “WATCHDOG TIMER” on page 255.)

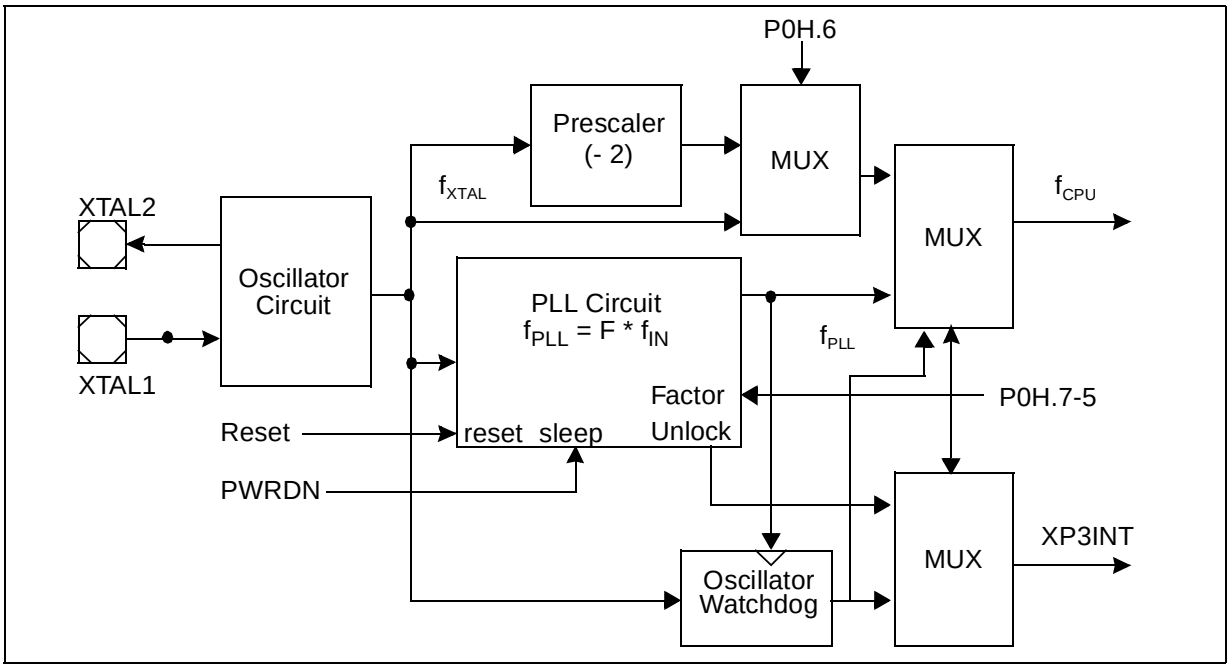


Figure 3 PLL block diagram

The table below lists all the possible selections for the on-chip clock generator. Refer to the device datasheet for the specific External Clock Input Range.

P0.15-13 (P0H.7-5)			CPU frequency f_{CPU} = f_{XTAL} * F	Notes
1	1	1	F_{XTAL} * 4	Default configuration
1	1	0	F_{XTAL} * 3	
1	0	1	F_{XTAL} * 2	
1	0	0	F_{XTAL} * 5	
0	1	1	F_{XTAL} * 1	Direct drive[1]
0	1	0	F_{XTAL} * 1.5	
0	0	1	F_{XTAL} / 2	CPU clock via 2:1 prescaler
0	0	0	F_{XTAL} * 2.5	

Table 1 CPU clock generation mechanisms

1. The maximum depends on the duty cycle of the external clock signal. The maximum input frequency is 25 MHz when using an external crystal oscillator, however, higher frequencies can be applied with an external clock source.

2.3.1 PLL operation

The PLL is enabled except when P0H.[7..5] = '011' or '001' during reset. On power-up, the PLL provides a stable clock-signal within ca. 1 ms after V_{CC} has reached 5V±10%, even if there is no external clock-signal (in this case, the PLL will run at its basic frequency of 2...5 MHz). The PLL will start synchronizing with the external clock signal as soon as it is available, but only during a hardware reset. The PLL is synchronous with this clock at a frequency of **F** * f_{XTAL}, i.e. the PLL locks to the external clock.

Note *If the ST10R272L is required to operate on the desired CPU clock directly after reset, make sure that $\overline{RSTIN}$ remains active until the PLL has locked (ca. 1 ms).*

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 long time periods. For short periods (1...4 CPU clock cycles), it remains below 4%.

When the PLL detects that it is no longer locked, i.e. no longer stable, it generates an interrupt request (on PLL Unlock XP3INT interrupt node). This occurs when the input clock is unstable and especially when the input clock fails completely, e.g. due to a broken crystal. In this case, the synchronization mechanism reduces the PLL output frequency down to the PLL's basic frequency (2...5 MHz). The basic frequency is still generated and the CPU can execute emergency actions.

2.3.2 Prescaler operation

When pins P0H.[7..5] = '001' during reset, the CPU clock is derived from the internal oscillator (input clock signal) by a 2:1 prescaler.

The frequency of f_{CPU} is half the frequency of f_{XTAL}.

The PLL runs on a basic frequency of 2...5 MHz, and delivers the clock signal for the oscillator watchdog.

2.3.3 Direct drive

When pins P0H.[7..5] = '011' during reset, the CPU clock is directly driven from the internal oscillator with the input clock signal, i.e. $f_{CPU} = f_{OSC}$. The maximum input clock frequency depends on the clock signal's duty cycle, because the minimum values for the clock phases (TCLs) must be reselected.

The PLL runs on its basic frequency of 2...5 MHz, and delivers the clock signal for the oscillator watchdog.

2.3.4 Oscillator watchdog (OWD)

The Oscillator Watchdog provides a fail safe mechanism for the loss of the external clock, for direct drive or direct drive with prescaler clock options. The oscillator watchdog is enabled by default after reset. To disable the OWD, set bit OWDDIS of the SYSCON register. When the OWD is enabled, the PLL runs on its free-running frequency, and increments the Oscillator Watchdog counter. On each transition of XTAL1 pin, the Oscillator Watchdog is cleared.

If an external clock failure occurs, then the Oscillator Watchdog counter overflows (after 16 PLL clock cycles). The CPU clock signal is switched to the PLL clock signal (the PLL will runs on its basic frequency of 2...5 MHz), and an Oscillator Watchdog interrupt request (XP3INT) is flagged. The CPU clock will not switch back to the external clock, even if a valid external clock exists on the XTAL1 pin. Only a hardware reset can switch the CPU clock source back to external clock input.

When the OWD is disabled, the CPU clock is always fed from the oscillator input and the PLL is switched off to decrease power supply current.

2.4 On-chip peripheral blocks

The ST10 family separates its peripherals from the core, allowing peripherals to be added or removed 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 Special Function Registers (SFRs). The SFRs are located in the standard SFR area or the extended ESFR area.

ST10R272L peripherals are:

- Two general purpose timer blocks (GPT1 and GPT2).
- One serial interface (ASC0).
- Watchdog Timer.
- Seven I/O ports with up to 77 I/O lines.
- Integrated application-specific synchronous serial port (X-peripheral number 0).

Each peripheral contains a set of Special Function Registers (SFRs) which control the functionality of the peripheral and temporarily store intermediate data. 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.

2.4.1 Peripheral interfaces

The on-chip peripherals, generally have two different types of interfaces - an interface to the CPU - and an interface to external hardware. Communication between the CPU and peripherals is performed through the Special Function Registers (SFRs) and interrupts. The SFRs serve as control/status and data registers for the peripherals. Interrupt requests are

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

Specific parallel port pins interface with external hardware when an input or output function has been selected for a peripheral. During this time, the port pins are controlled by the peripheral (when used as outputs) or by the external hardware which controls the peripheral (when used as inputs). This is called the 'alternate (input or output) function' of a port pin, in contrast to its function as a general purpose IO pin.

2.4.2 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. In Idle Mode the CPU clock is stopped while the peripherals continue their operation. Peripheral SFRs may be accessed by the CPU. 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.

2.4.3 Programming hints

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

- Indirect or direct addressing with 16-bit (mem) addresses. The used data page pointer (DPP0...DPP3) must select 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 512Byte area.
- Short 8-bit (reg) addresses to the extended ESFR area. Switch to the 512 Byte extended SFR area be using 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. Therefore, it is better 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.

2.4.4 Reserved bits

Some SFRs are 'reserved'. **Never write '1's to reserved bits.** These bits are not currently implemented and may be used in future products. The active state for these functions will be '1', and the inactive state will be '0'. Therefore, writing only '0's to reserved locations give upgradability of current software to future devices. Read accesses to reserved bits return '0's.

2.4.5 Parallel ports

The ST10R272L has up to 77 I/O lines, organized into six input/output ports and one input port. All port lines are bit-addressable, and all input/output lines are individually (bit-wise) programmable as inputs or outputs via direction registers. The 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 via the control registers for open-drain operation. During internal reset, all port pins are configured as inputs.

All port lines have associated, programmable, alternate, input or output functions. PORT0 and PORT1 may be used as address and data lines for external memory access. Port 4 outputs the additional segment address bits A23/19/17...A16 in systems where segmentation is enabled to access more than 64 KBytes of memory. Port 6 provides optional bus-arbitration signals ($\overline{BREQ}$, $\overline{HLDA}$, $\overline{HOLD}$) and chip-select signals. Port 3 includes alternate functions of timers, serial interfaces, the optional bus control signal $\overline{BHE}$ and the system clock output (CLKOUT). Port 5 is used for timer control signals. All port lines that are not used for these alternate functions may be used as general purpose I/O lines.

2.4.6 Serial channels

Serial communication with other microcontrollers, processors, terminals or external peripheral components is provided by two serial interfaces - an Asynchronous/synchronous Serial Channel (ASC0) and a Synchronous Serial Port (SSP).

ASCO supports full-duplex asynchronous communication. A dedicated baud rate generator sets up standard baud rates without oscillator tuning. For transmission, reception, and erroneous reception, 3 separate interrupt vectors are provided for ASC0. 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. There is a multiprocessor communication mechanism to distinguish address from data bytes (8-bit data + wake-up bit mode). In synchronous mode, the ASC0 transmits or receives bytes (8 bits) synchronously to a shift clock which is generated by the ASC0.

The SSP can be configured to interface with serially-linked peripheral components. There is one general interrupt vector for the SSP.

2.4.7 General purpose timer (GPT)

The GPT unit is a flexible multifunctional timer/counter structure 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 GPT unit incorporates five 16-bit timers, 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.

2.4.8 Watchdog timer

The Watchdog Timer is a fail-safe mechanism which limits the maximum malfunction time of the controller. The Watchdog Timer is always enabled after device reset, and can only be disabled in the time interval until the EINIT (end of initialization) instruction has been executed. In this way, the chip's start-up procedure is always monitored. The software must be designed to service the Watchdog Timer before it overflows. If, due to hardware or software related failures, the software fails to maintain the watchdog timer, it will overflow generating an internal hardware reset and pulling the $\overline{\text{RSTOUT}}$ pin low to reset external hardware components.

The Watchdog Timer is a 16-bit timer, clocked with the system clock divided either by 2 or 128. The high byte of the Watchdog Timer register can be set to a pre-specified reload value 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.

2.5 Protected bits

The ST10R272L provides a special bit protection mechanism. Bits which can be modified by the on-chip hardware, cannot be unintentionally changed by software accesses to related bits ("Bit-handling and bit-protection" on page 45).

The following bits are protected:

Register	Bit Name	Notes
T2IC, T3IC, T4IC	T2IR, T3IR, T4IR	GPT1 timer interrupt request flags
T5IC, T6IC	T5IR, T6IR	GPT2 timer interrupt request flags
CRIC	CRIR	GPT2 CAPREL interrupt request flag
T3CON, T6CON	T3OTL, T6OTL	GPTx timer output toggle latches
S0TIC, S0TBIC	S0TIR, S0TBIR	ASC0 transmit(buffer) interrupt request flags

Table 2 Protected bits

Register	Bit Name	Notes
S0RIC, S0EIC	S0RIR, S0EIR	ASC0 receive/error interrupt request flags
S0CON	S0REN	ASC0 receiver enable flag
TFR	TFR.15,14,13	Class A trap flags
TFR	TFR.7, 6, 3, 2,1,0	Class B trap flags
XPyIC (y=1, 3)	XPyIR (y=1, 3)	X-Peripheral y interrupt request flag

Table 2 Protected bits

Σ = 24 protected bits.

3 MEMORY ORGANIZATION

The memory space of theST10R272L is configured in a “Von Neumann” architecture. This means that code and data are accessed within the same linear address space. All of the physically separated memory areas, including internal RAM, the internal Special Function Register Areas (SFRs and ESFRs), the address areas for integrated XBUS peripherals and external memory are mapped into one common address space.

The ST10R272L provides a total addressable memory space of 16MBytes. This address space is arranged as 256 segments of 64KBytes each, and each segment is subdivided into four data pages of 16 KBytes each (see figure below).

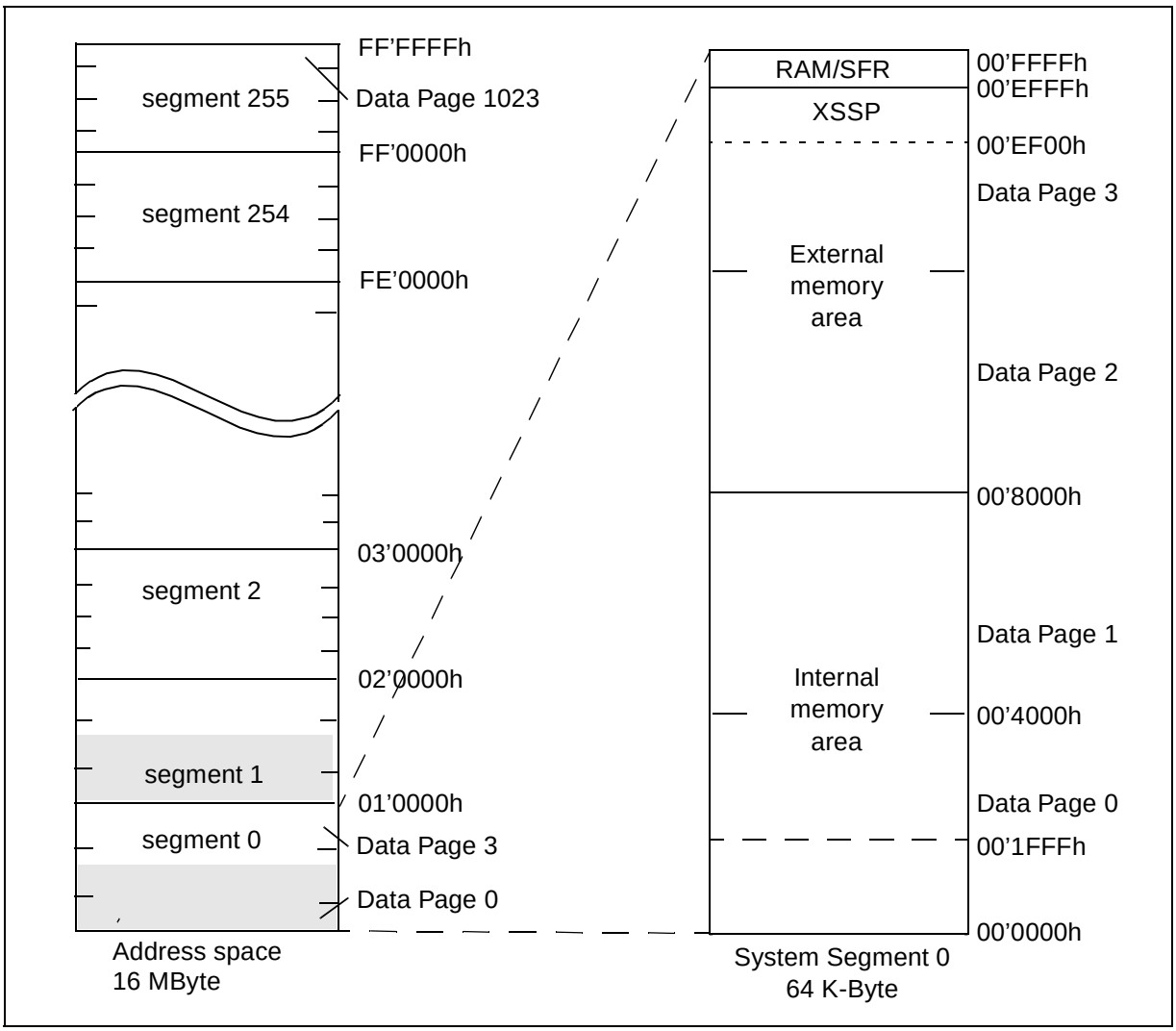


Figure 4 Memory areas and address

Most internal memory areas are mapped into segment 0, the system segment. The upper 4 KByte of segment 0 (00'F000h...00'FFFFh) hold the Internal RAM and Special Function Register Areas (SFR and ESFR).

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

The ST10R272L is a ROMless device: program ROM must be in external memory.

Bytes are stored at even or odd byte addresses. Words are stored in ascending memory locations with the low byte at an even byte address, followed by the high byte at the next odd byte address. 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.

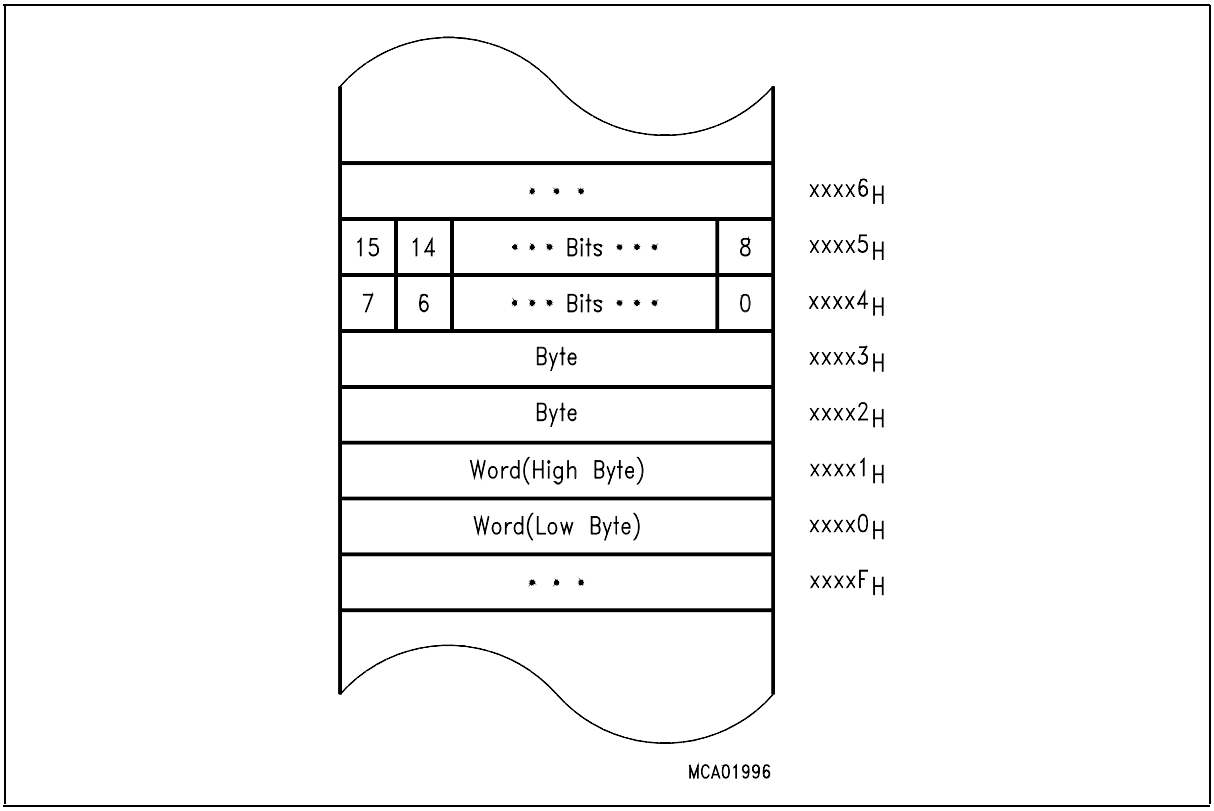


Figure 5 Storage of words, bytes and bits in a byte organized memory

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

3.1 Internal RAM and SFR area

The ST10R272L has 1 KByte of internal RAM in the address range 00'FA00h - 00'FDFFh. It is used for variables, the register banks, and the system stack. It contains the PEC pointers (address range 00'FCE0h - 00'FCFFh) and the bit-addressable space (00'FD00h - 00'FDFFh).

The internal special function registers (SFR) are in the address range 00'FE00h - 00'FFFFh, and extended special function registers (ESFR) are in the address range 00'F000h - 00'F200h.

The RAM/SFR area is located in data page 3 and provides access to 1KByte of on-chip RAM (organized as 512*16) and to two 512-byte blocks of Special Function Registers (SFRs).

The internal RAM serves 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.
- Code storage.

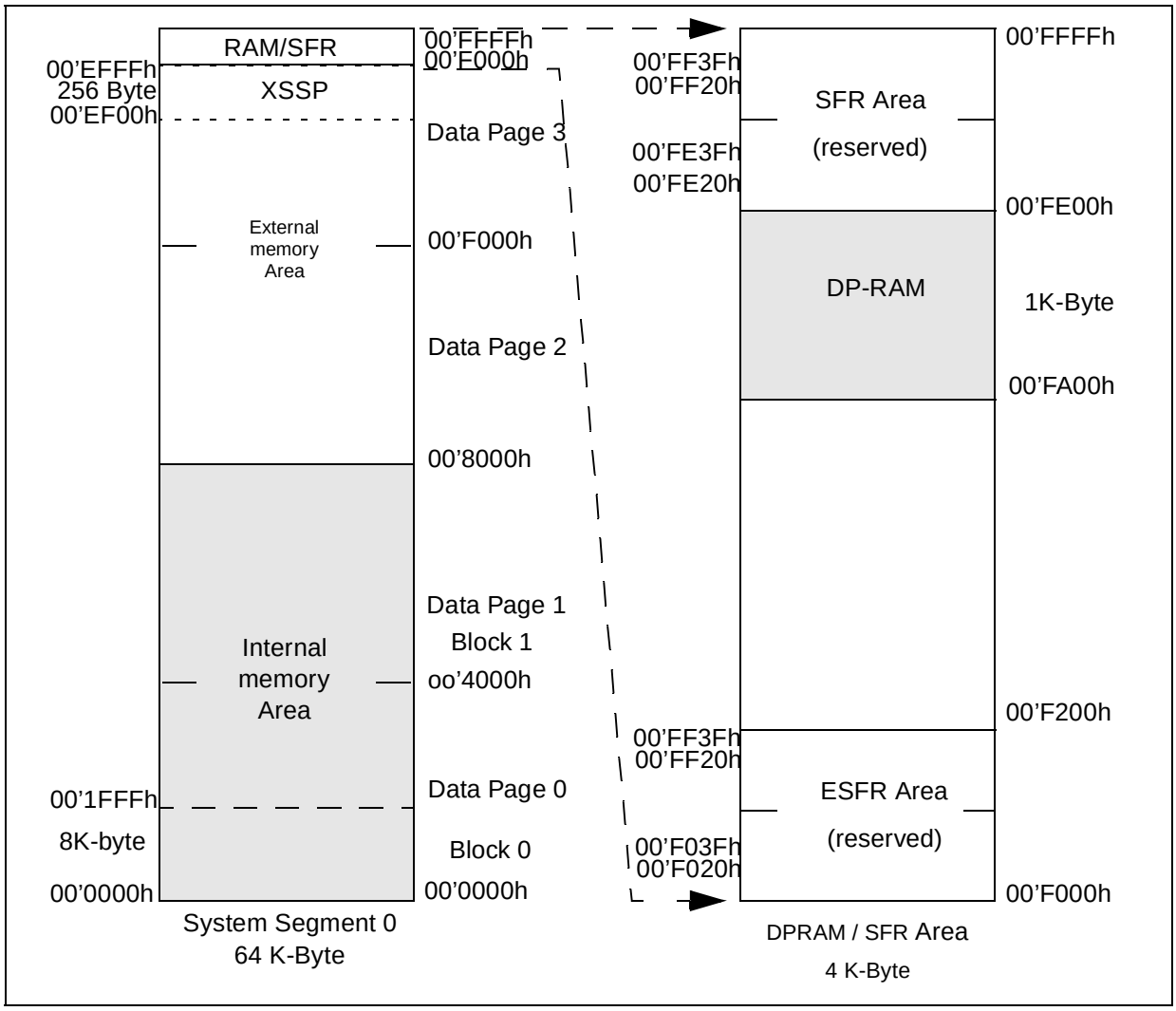


Figure 6 Internal RAM and SFR areas

Note *The upper 256 bytes of SFR area, ESFR area and internal RAM are bit-addressable.*

Code accesses are always made on EVEN byte addresses. The highest code storage location in the internal RAM is, either 00'FDFEh for single-word instructions, or 00'FDFCh 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 by 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'FDFEh. For PEC data transfers, the internal RAM can be accessed, independent of the contents of the DPP registers, via the PEC source and destination pointers.

The upper 256 bytes of the internal RAM (00'FD00h through 00'FDFFh) and the GPRs of the current bank, are provided for single bit storage and are bit addressable.

3.1.1 System stack

The system stack can be defined in the internal RAM. The system stack size is controlled by the bit-field STKSZ in the SYSCON register (see table below).

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 by the system stack. A stack overflow (STKOV) and a stack underflow (STKUN) register control the lower and upper limits of the selected stack area. These two stack boundary registers can be used, not only for protection against data destruction, but also to implement a circular stack, with hardware-supported system-stack flushing and filling (except for the 2KByte stack option).

The technique of implementing this circular stack is described in "Circular (virtual) Stack" on page 305.

<STKSZ>	Stack Size (Words)	Internal RAM Addresses (Words)
0 0 0 b	256	00'FBFEh...00'FA00h (Default after Reset)
0 0 1 b	128	00'FBFEh...00'FB00h
0 1 0 b	64	00'FBFEh...00'FB80h
0 1 1 b	32	00'FBFEh...00'FBC0h
1 0 0 b	512	00'FBFEh...00'F800h
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'FDFEh...00'F600h (Note: No circular stack)

Table 3 STKSZ bitfield in the SYSCON register

3.1.2 General purpose registers

The General Purpose Registers (GPRs) use a block of 16 consecutive words in 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, up to 16 byte GPRs (RL0, RH0, ..., RL7, RH7). The 16 byte GPRs are mapped onto the first 8 word GPRs (see table below).

In contrast to the system stack, a register bank grows from lower towards higher address locations and occupies a maximum space of 32 bytes. The GPRs are accessed via short 2-, 4- or 8-bit addressing modes, using the 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.

The ST10R272L supports fast register-bank (context) switching. Multiple register-banks can physically exist within the internal RAM at the same time. However, only the register-bank selected by the Context Pointer register (CP) is active at a given time. A new active-register-bank is selected by updating the CP register. The 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 “SYSTEM PROGRAMMING” on page 303.

Internal RAM Address	Byte Registers		Word Register
<CP> + 1Eh	---		R15
<CP> + 1Ch	---		R14
<CP> + 1Ah	---		R13
<CP> + 18h	---		R12
<CP> + 16h	---		R11
<CP> + 14h	---		R10
<CP> + 12h	---		R9
<CP> + 10h	---		R8
<CP> + 0Eh	RH7	RL7	R7
<CP> + 0Ch	RH6	RL6	R6

Table 4 Mapping of general purpose registers to RAM addresses

Internal RAM Address	Byte Registers		Word Register
<CP> + 0Ah	RH5	RL5	R5
<CP> + 08h	RH4	RL4	R4
<CP> + 06h	RH3	RL3	R3
<CP> + 04h	RH2	RL2	R2
<CP> + 02h	RH1	RL1	R1
<CP> + 00h	RH0	RL0	R0

Table 4 Mapping of general purpose registers to RAM addresses

3.1.3 PEC source and destination pointers

The 16 word-locations in the internal RAM from 00’FCE0h to 00’FCFFh (just below the bit-addressable space, 00’FD00h - 00’FDFEh) provide 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).

Whenever a PEC data-transfer is performed, the pair of source and destination pointers, selected by the specified PEC channel number, is accessed independently of the current DPP register contents. Also, the locations referred to by these pointers are accessed independently of the current DPP register contents. If a PEC channel is not used, the corresponding pointer locates the available area which 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 “INTERRUPT AND TRAP FUNCTIONS” on page 83.

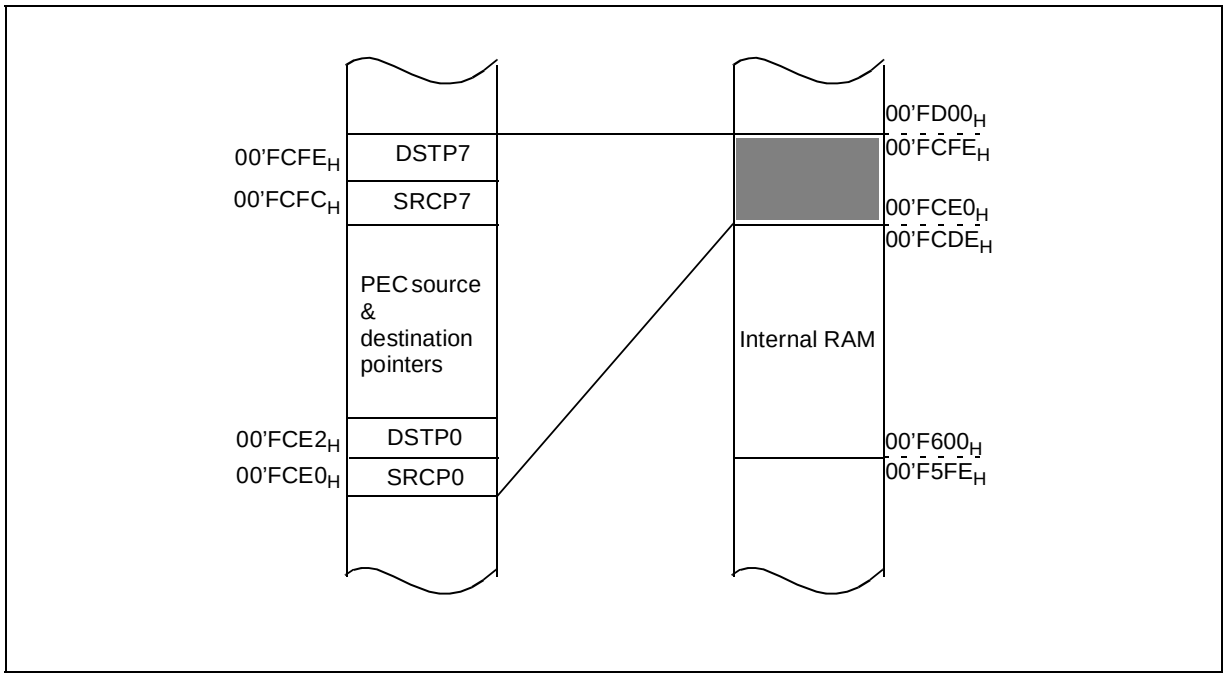


Figure 7 Location of the PEC pointers

3.1.4 Special function registers

The functions of the CPU, the bus interface, the I/O ports and the on-chip peripherals are controlled by Special Function Registers (SFRs). The SFRs are arranged in two areas of 512 bytes each. The first register block - the SFR area - is located in the 512 Bytes above the internal RAM (00'FFFFh...00'FE00h), the second register block - the Extended SFR (ESFR) area - is located in the 512 Bytes below the internal RAM (00'F1FFh...00'F000h).

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, makes it possible to address word SFRs and their respective low bytes. However, this does not work for the respective high bytes!

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

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

Identification registers are held in the SFR area.

When accessing registers in the ESFR area - using 8-bit addresses or direct bit addressing - an extend register (EXTR) instruction is required to switch the short addressing mechanism from the standard SFR area to the extended SFR area. This is not required for 16-bit and

indirect addresses. The GPRs R15...R0 are duplicated, i.e. they are accessible within both register blocks via short 2-, 4- or 8-bit addresses, without switching.

```
EXTR      #4                    ;Switch to ESFR area for the next 4 instructions
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 address, R1 is duplicated...
                                ;...and also accessible via the ESFR mode
                                ;(EXTR is not required for this access)
;------   ;-------------------  ;The scope of the EXTR #4 instruction ends here!
MOV       T8REL, R1             ;T8REL uses 16-bit address, R1 is duplicated...
                                ;...and does not require switching
```

Table 5 Example of the EXTR instruction

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

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

3.2 External memory space

The ST10R272L uses an address space of up to 16 MByte. Only parts of this address space are occupied by internal memory areas. All addresses which are not used for on-chip RAM, registers or internal Xperipherals, may reference external memory locations through the External Bus Interface.

Four memory bank sizes are supported:

Non-segmented mode:	64KByte 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.

Non-segmented mode:	64KByte with A15...A0 on PORT0 or PORT1.
8-bit segmented mode:	16 MByte with A23...A16 on Port 4 and A15...A0 on PORT0 or PORT1.

Each bank can be directly addressed via the address bus, while 'Programmable Chip Select Signals' can be used to select various memory banks.

Four different bus types are supported:

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 refer to "EXTERNAL BUS INTERFACE" on page 145.

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

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

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

3.3 Crossing memory boundaries

An address 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.

Memory Areas are partitions of the address space that represent different kinds of memory. These memory areas are: the internal RAM/SFR area 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 in segment 0.*

Segments are contiguous blocks of 64KBytes. They are referenced via the code segment pointer CSP for code fetches, and via an explicit segment number for data accesses when overriding the standard DPP scheme.
During code fetching, segments are not changed automatically, but must be switched explicitly. Instructions JMPS, CALLS and RETS will do this.

Note *To prevent the prefetcher from trying to leave the current segment (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.*

Data Pages are contiguous blocks of 16KByte each. They are referenced via the data page pointers DPP3...0 and via an explicit data page number for data accesses when overriding the standard DPP scheme. Each DPP register can select one of the possible 1024 data pages. The DPP register, 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 16KByte data page boundaries, use different data page pointers while the physical locations need not be subsequent within memory.

4 CENTRAL PROCESSING UNIT

The main core of the CPU includes a 4-stage instruction pipeline, a separate multiply and divide unit, a bit-mask generator and a barrel shifter. Most ST10R272L instructions can be executed in one machine cycle.

The MAC performs multiply-accumulate operations. It has an enhanced instruction set for 32-bit arithmetic, computation and data moves. The MAC is described in “MULTIPLY-ACCUMULATE UNIT (MAC)” on page 67.

The CPU includes an actual register context. This consists of 16 wordwide 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. The number of register banks is only restricted by the available internal RAM space. For easy parameter passing, one register bank may overlap others.

A system stack of up to 1024 bytes is provided as a storage for temporary data. The system stack is allocated in 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 compared against the stack pointer value during each stack access, to detect stack overflow or underflow.

While internal memory accesses are normally performed by the CPU, external peripheral or memory accesses are performed by the on-chip External Bus Controller (EBC). This 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 “EXTERNAL BUS INTERFACE” on page 145.

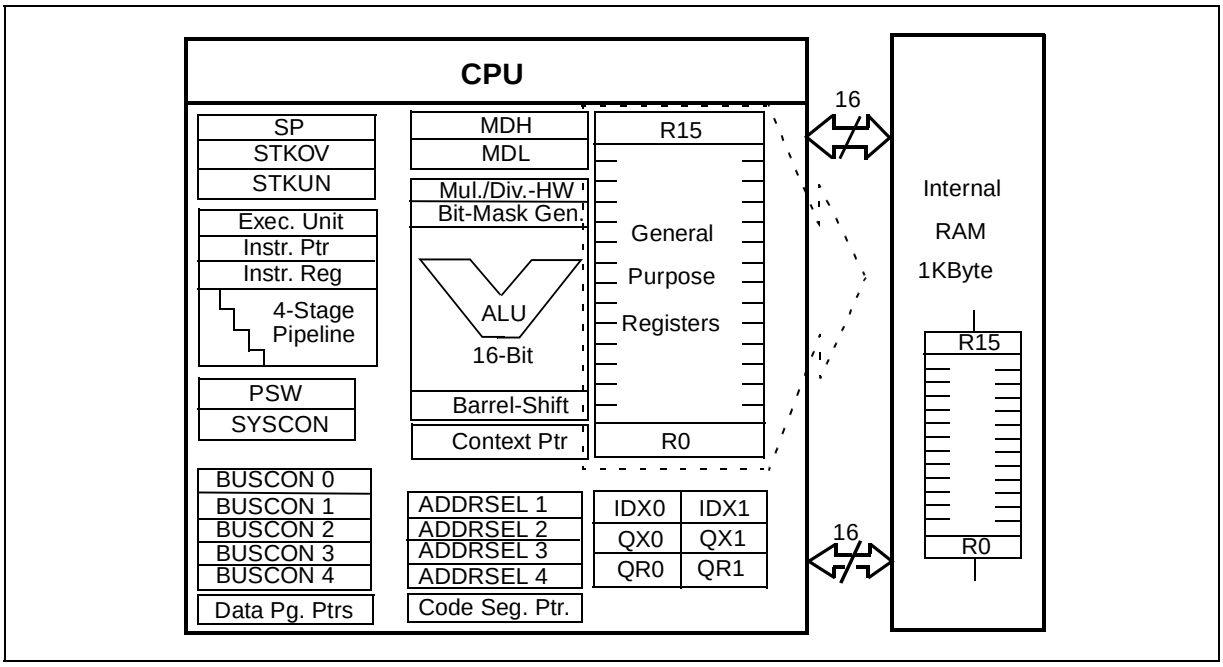


Figure 8 CPU block diagram

The on-chip peripheral units of the ST10R272L work almost independently 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.

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

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

In contrast to other on-chip peripherals, there is a close conjunction between the Watchdog Timer and the CPU. If enabled, the Watchdog Timer must be serviced by the CPU within a programmable period of time, otherwise it will reset the chip. In this way, the Watchdog Timer

is able to prevent the CPU malfunctioning for long periods of time. After reset, the Watchdog Timer starts counting automatically, but it can be disabled via software.

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

- **Reset:** Any reset (hardware, software, watchdog) forces the CPU into a predefined active state.
- **Idle:** The clock signal to the CPU itself is switched off but the clocks for the on-chip peripherals keep running.
- **Power Down:** 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 the use of ST10R272L system control instructions.

4.1 Instruction pipelining

The instruction pipeline partitions instruction processing into four stages:

Fetch: The instruction selected by the Instruction Pointer (IP), and the Code Segment Pointer (CSP) is fetched from either the internal RAM or external memory.

Decode: The instructions are decoded and, if required, the operand addresses are calculated and respective operands are fetched. For all instructions which implicitly access the system stack, the SP register is either decremented or incremented. 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).

Execute: 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.

Write back: All external operands and the remaining operands in the internal RAM space are written back.

4.1.1 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 allows simultaneous processing of up to four

instructions. Therefore, most instructions seem to be processed in one machine cycle as soon as the pipeline has been filled (see figure below).

Instruction pipelining increases the average instruction throughput over a certain period of time. Specification of instruction execution time always refers to the average execution time of pipelined parallel instruction processing.

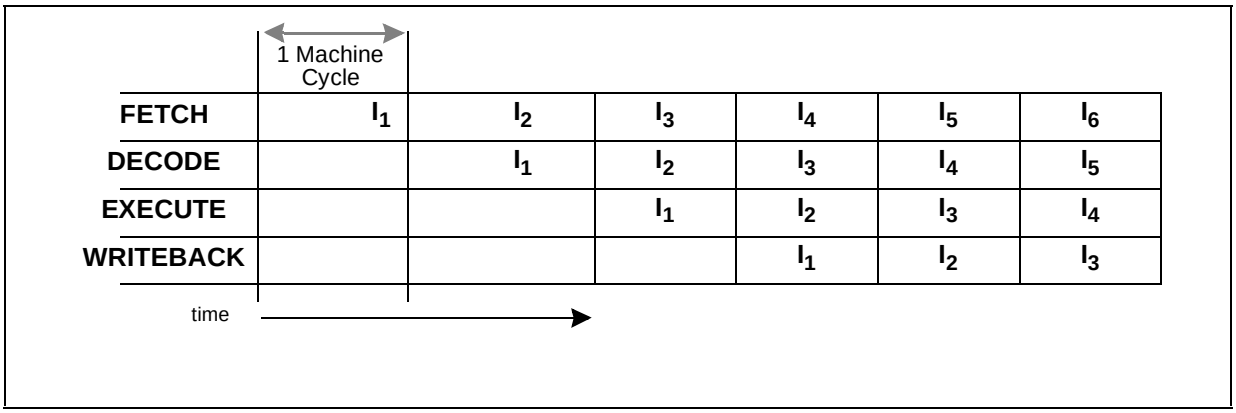


Figure 9 Sequential instruction pipelining

4.1.2 Standard branch instruction processing

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

If a conditional branch is not taken, there is no deviation from the sequential program flow and 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.

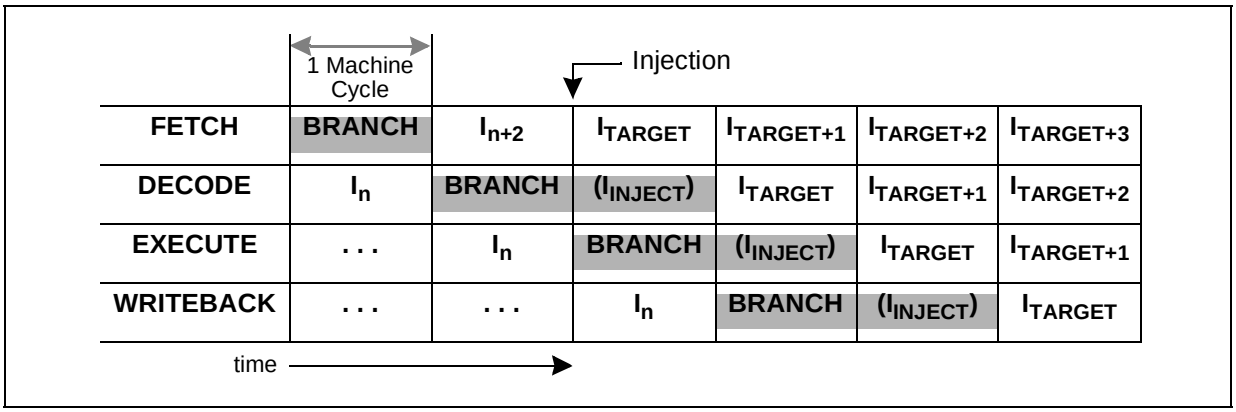


Figure 10 Standard branch instruction pipelining

4.1.3 Jump cache instruction processing

A jump cache optimizes conditional jumps that are processed repeatedly in a loop. Whenever a jump on cache is taken, the extra time to fetch the branch target instruction can be saved and the corresponding cache jump instruction - in most cases - takes only one machine cycle.

This performance is achieved by the following mechanism:
Whenever a jump cache 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. However, in contrast to standard branch instructions, the target instruction of a jump cache instruction (JMPA, JMPR, JB, JBC, JNB, JNBS) is also stored in the cache.

For subsequent execution of the same jump cache instruction, the jump target instruction is not fetched from program memory, but is taken from the cache and immediately injected into the decode stage of the pipeline (see figure below).

A jump on cache is always taken after the second and any further occurrence of the same cache jump instruction, unless:

- it changes the CSP register contents (JMPS, CALLS, RETS, TRAP, RETI),
- or a standard interrupt has been processed between two occurrences of the same cache jump instruction.

.

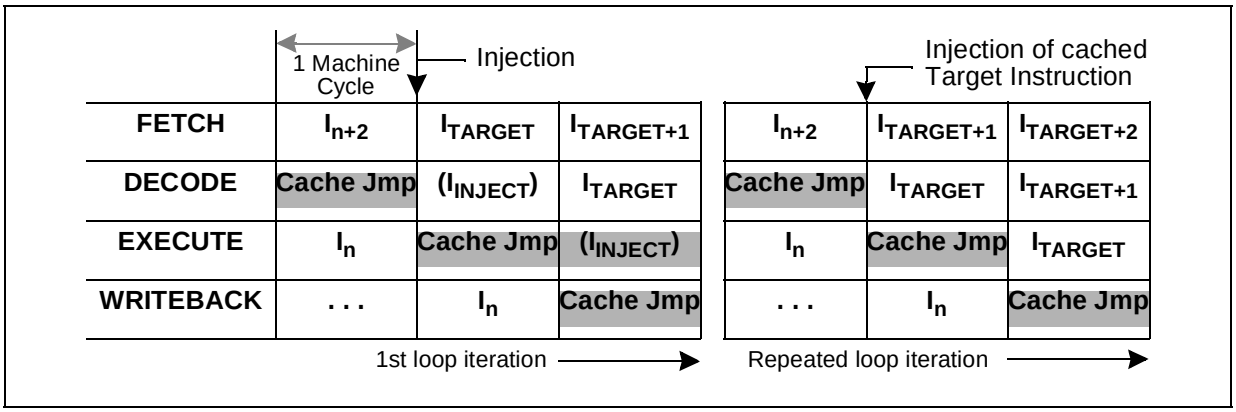


Figure 11 Cache jump instruction pipelining

4.1.4 Pipeline effects

Additional hardware takes account of any causal dependencies on instructions in different pipeline stages. This hardware (i.e. for 'forwarding' operand read and write values) resolves most of the conflicts (e.g. multiple usage of buses) and prevents the pipeline from becoming noticeable to the user. However, there are some cases where the fact that the ST10R272L is a pipelined machine, requires attention by the programmer. In these cases, the delays caused by pipeline conflicts can be used by other instructions, to optimize performance. These cases are outlined below:

Context Pointer Updating

An instruction which calculates a physical GPR operand address via the CP register, is not usually capable of using a new CP value which is to be updated by an immediately preceding instruction. Therefore, to make sure that the new CP value is used, at least one instruction must be inserted between a CP-changing and a subsequent GPR-using instruction, as shown in the following example:

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

Data Page Pointer Updating

An instruction, which calculates a physical operand address via a particular DPPn (n=0 to 3) register, is not capable of using a new DPPn register value which is to be updated by an immediately preceding instruction. 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, for example:

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

Explicit stack pointer updating

None of the RET, RETI, RETS, RETP or POP instructions are capable of correctly using a new SP register value which is to be updated by an immediately preceding instruction. Therefore, 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 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
```

External memory access sequences

This effect only becomes noticeable when watching the external memory access sequences on the external bus (e.g. by means of a Logic Analyzer). Different pipeline stages can, simultaneously, put a request on the External Bus Controller (EBC). The sequence of instructions processed by the CPU may diverge from the sequence of 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. However, to maintain fast interrupt responses, the current interrupt-prioritization round does not consider these changes - i.e. an interrupt request may be acknowledged after the instruction that disables interrupts via IEN or ILVL, or after the following instructions. Therefore, time critical instruction sequences should not begin directly after the instruction disabling interrupts, as shown in the following example:

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

Note *A delay of one instruction also applies to the enabling of the interrupt system, i.e. no interrupt requests are acknowledged until the instruction following the enabling instruction is FETCHED.*

Initialization of port pins

Port pins direction modifications (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, for example:

```
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 (e.g. segmentation, stack size), cannot use the new resources (e.g. 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.*

Updating the stack pointer

An instruction that changes the contents of the stack pointer SP (MOV, ADD, SUB), may not be followed directly by an instruction that implicitly uses the SP, i.e. a POP or RETURN instruction. To ensure proper operation, an instruction should be inserted that does not use the stack pointer.

Timing

Instruction pipelining reduces the average instruction processing time from four to one machine cycles. However, there are some cases, where the pipeline causes the a single instruction processing to be extended, either by a half or by one machine cycle.

The following section gives some hints on optimizing time-critical programs, with regard to the pipeline.

4.2 Bit-handling and bit-protection

The ST10R272L provides several bit manipulation mechanisms which manipulate software flags within the internal RAM, control on-chip peripherals via control bits in their respective SFRs, or control IO functions via port pins.

The instructions BSET, BCLR, BAND, BOR, BXOR, BMOV and BMOVN explicitly set or clear specific bits. The instructions BFLDL and BFLDH 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 affect the respective bit location.*

All instructions that manipulate single bits or bit groups internally, use a read-modify-write sequence that accesses the whole word. This method has several consequences:

- Bits can only be modified within the internal address areas, i.e. internal RAM and SFRs. External locations cannot be used for bit instructions.The upper 256 bytes of the SFR area, the ESFR area and the internal RAM are bit-addressable (refer to "MEMORY ORGANIZATION" on page 25) i.e. those register bits located within the respective sections can be directly manipulated using bit instructions. The other SFRs must be accessed byte/word wise. Note: all GPRs are bit-addressable, independent of the allocation of the register bank via the context pointer CP. Even GPRs which are allocated to non-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 overwrites the new bit value generated by the hardware. Hardware protection (see below) or special programming(see Section 4.1.4) is required .

Protected bits are not changed during the read-modify-write sequence, i.e. when hardware sets, for example, an interrupt request flag between the read and the write of the read-modify-write sequence. The hardware protection logic guarantees that only the intended bit(s) is/are affected by the write-back operation. Refer to "Protected bits" on page 23 for a summary of the protected bits implemented on this device.

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

4.3 Instruction execution time

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 is to execute a program fetched from fast external memory (no wait states), using a 16-bit demultiplexed bus. Then, most instructions can be processed in one machine cycle.

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

A detailed description of the execution times for the various instructions and the specific exceptions can be found in the **"ST10 Family Programming Manual"**.

The table below shows the minimum execution times required to process an instruction fetched from the internal RAM or from external memory. These execution times apply to most of the ST10R272L instructions - except for some branches, the multiplication and the division instructions and a special move instruction.

Memory Area	Instruction Fetch		Word Operand Access	
	Word Instruction (CPU clock cycles)	Doubleword Instruction (CPU clock cycles)	Read from	Write to
Internal RAM	6	8	0/50	0
16-bit Demux Bus	2	4	100	100
16-bit Mux Bus	3	6	150	150
8-bit Demux Bus	4	8	200	200
8-bit Mux Bus	6	12	300	300

Table 6 Minimum execution times without 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.
- Testing Branch Conditions immediately after PSW writes.

4.4 CPU special function registers

Special Function Registers (SFRs) maintain the system state information, supply the ALU with register-addressable constants, control the system and bus configuration, multiply and divide ALU operations, code memory segmentation, data memory paging, and access the general purpose registers and the system stack.

Since all SFRs can be controlled by 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, 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, supercedes 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'.

4.4.1 The system configuration register SYSCON

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

SYSCON (FF12h / 89h) **SFR** **Reset Value: 0XX0h**)

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	PWD CFG	OWD DIS	BDR STEN	SSP EN	VISI BLE	XPER-SHARE
rw			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
XPER-SHARE	**XBUS Peripheral Share Mode Control** '0': External accesses to XBUS peripherals are disabled '1': XBUS peripherals are accessible via the external bus during hold mode.

Bit	Function
VISIBLE	**Visible Mode Control** '0': Accesses to XBUS peripherals are done internally '1': XBUS peripheral accesses are made visible on the external pins.
SSPEN	**Xperipheral SSP Enable Control** '0': SSP is disabled. Pins P4.[7..4] are gen. purpose I/Os or segment address lines '1': SSP is enabled. Pins P4.[7..4] are SSP IOs or segment address lines
BDRSTEN	**Bidirectional Reset Enable** '0': $\overline{\text{RSTIN}}$ pin is an input pin only. SW Reset or WDT Reset has no effect on this pin. '1': $\overline{\text{RSTIN}}$ pin is a bidirectional pin. This pin is pulled low during 1024 TCL during rest sequence.
OWDDIS	**Oscillator Watchdog disable control** '0': Oscillator Watchdog (OWD) is enabled. If PLL is bypassed, the OWD monitors XTAL1 activity. If there is no activity on XTAL1 for at least 1 ms, the CPU clock is switched automatically to PLL's base frequency (around 5 MHz). '1': OWD is disabled. If the PLL is bypassed, the CPU clock is always driven by XTAL1 signal. The PLL is turned off to reduce the power supply current.
PWDCFG	**Power Down Mode Configuration Control** '0': Power Down Mode can only be entered during PWRDN instruction execution if $\overline{\text{NMI}}$ pin is low, otherwise the instruction has no effect. To exit Power Down Mode, an external reset must occurs by asserting the $\overline{\text{RSTIN}}$ pin. '1': Power Down Mode can only be entered during PWRDN instruction execution if all enabled fast external interrupt EXxIN pins are in their inactive level. Exiting this mode can be done by asserting one enabled EXxIN pin.
CSCFG	**Chip Select Configuration Control** '0': Latched Chip Select lines: CSx change 1 TCL after rising edge of ALE '1': Unlatched Chip Select lines: CSx change with rising edge of ALE
WRCFG	**Write Configuration Control** (Set according to pin $\overline{\text{P0H.0}}$ during reset) '0': Pins $\overline{\text{WR}}$ and $\overline{\text{BHE}}$ retain their normal function '1': Pin $\overline{\text{WR}}$ acts as $\overline{\text{WRL}}$, pin $\overline{\text{BHE}}$ acts as $\overline{\text{WRH}}$
CLKEN	**System Clock Output Enable** (CLKOUT) '0': CLKOUT disabled: pin may be used for general purpose IO '1': CLKOUT enabled: pin outputs the system clock signal
BYTDIS	**Disable/Enable Control for Pin BHE** (Set according to data bus width) '0': Pin $\overline{\text{BHE}}$ enabled '1': Pin $\overline{\text{BHE}}$ disabled, pin may be used for general purpose IO

Bit	Function
ROMEN	**Internal ROM Enable** (Set according to pin $\overline{EA}$ during reset) '0': Internal ROM disabled: accesses to the ROM area use the external bus '1': Internal ROM enabled This bit is not relevant on the ST10R262 since it does not include internal ROM. It should be kept to 0
SGTDIS	**Segmentation Disable/Enable Control** '0': Segmentation enabled (CSP is saved/restored during interrupt entry/exit) '1': Segmentation disabled (Only IP is saved/restored)
ROMS1	**Internal ROM Mapping** '0': Internal ROM area mapped to segment 0 (00'0000h...00'7FFFh) '1': Internal ROM area mapped to segment 1 (01'0000h...01'7FFFh) This bit is not relevant on the ST10R262 since it does not include internal ROM. It should be kept to 0.
STKSZ	**System Stack Size** Selects the size of the system stack (in the internal RAM) from 32 to 1024 words

Note SYSCON register cannot be changed after execution of the EINIT instruction. The function of bits XPER-SHARE, VISIBLE, WRCFG, BYTDIS, ROMEN and ROMS1 is described in more detail in "EXTERNAL BUS INTERFACE" on page 145.

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 selects 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 therefore, 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 whether the CSP register is pushed onto the system stack (in addition to the IP and PSW registers) before an interrupt service routine is entered, and whether it is re-popped when the interrupt service routine is left again.*

System stack size (STKSZ)

This bitfield defines the size of the physical system stack located in the internal RAM. An area of 32...512 words, or all of the internal RAM, may be dedicated to the system stack. A ‘circular stack’ mechanism makes it possible 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 “SYSTEM PROGRAMMING” on page 303.

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) in the PSW register provides a general-purpose user flag.

PSW (FF10h / 88h) **SFR** **Reset Value: 0000h**

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.

Bit	Function
E	**End of Table Flag** Set, when the source operand of an instruction is 8000h or 80h.
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 PSW condition flags (N, C, V, Z, E) show the ALU status from the last ALU operation. They are set by 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 here, because any explicit write to the PSW register supersedes the condition flag values 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.

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

N-Flag: For most 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 '–8000h' to '+7FFFh' for the word data-type, or from '–80h' to '+7Fh' 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 performed as a 2's complement addition, and the C-flag is cleared when this complement addition caused a

carry.
The C-flag is always cleared for logical, multiply and divide ALU operations because these operations cannot cause a carry.
For shift and rotate operations, the C-flag represents the value of the bit shifted out last. If a shift count of zero is specified, the C-flag will be cleared. The C-flag is also cleared for a prioritize ALU operation, because a ’1’ is never shifted out of the MSB during the normalization of an operand.
For Boolean bit operations with only one operand, the C-flag is always cleared. For Boolean bit operations with two operands, the C-flag represents the logical ANDing of the two specified bits.

V-Flag: For addition, subtraction and 2’s complement, the V-flag is always set to ’1’ if the result overflows the maximum range of signed numbers (16 bits for word operations ('–8000h' to '+7FFFh'), or by 8 bits for byte operations ('–80h' to '+7Fh')), 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. Using the C-flag only, gives a rounding error (caused by a shift right operation) of one half of the LSB of the result. Using the V-flag, the C-flag together gives finer resolution on the rounding error (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.

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

Table 7 Shift right rounding

Z-Flag: The Z-flag is 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 whether 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 by 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 (’8000h’ for the word data-type, or ’80h’ for the byte data-type), the E-flag is set to ’1’, otherwise it is cleared.

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

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

CPU interrupt status (IEN, ILVL)

The Interrupt Enable bit enables (IEN=’1’) or disables (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 on entry into an interrupt service routine, but it can also be modified by software to prevent other interrupts from being acknowledged. An interrupt level of '15' has the highest possible priority and the current CPU operation cannot be interrupted except by hardware traps or external non-maskable interrupts. For details refer to “INTERRUPT AND TRAP FUNCTIONS” on page 83.

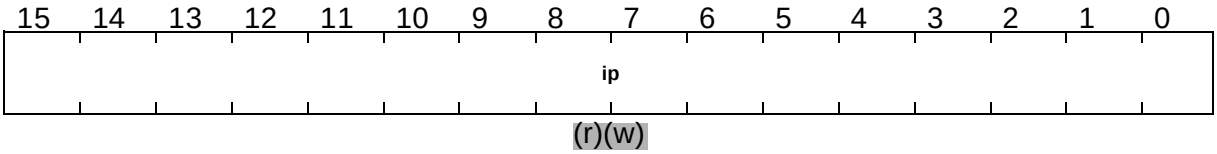
After reset all interrupts are globally disabled and the lowest priority (ILVL=0) is assigned to the initial CPU activity.

The instruction pointer IP

This register determines the 16-bit intra-segment address of the instruction in the code segment selected by the CSP register. The IP register is not mapped into the ST10R272L's address space, and cannot be directly accessed by the programmer. The IP can, however, be modified indirectly via the stack, with a return instruction.

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

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



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

The Code segment pointer CSP

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

Code memory addresses are generated by directly extending the 16-bit content of the IP register by the contents of the CSP register as shown in Figure 12.

In Segmented Memory Mode, the selected number of segment address bits (7...0, 3...0 or 1...0) of the CSP register is output on the segment address pins A23/A19/A17...A16 of Port 4 for all external code accesses. For Non-segmented Memory Mode or Single Chip Mode, the content of this register is not significant because all code accesses are automatically restricted to segment 0.

The CSP register can be read, but not written to by data operations. However, it can be modified either directly with the JMPS and CALLS instructions, or indirectly via the stack with

the RETS and RETI instructions. On the acceptance of an interrupt, or the execution of a software TRAP instruction, the CSP register is automatically set to zero

CSP (FE08h / 04h) **SFR** **Reset Value: 0000h**

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.

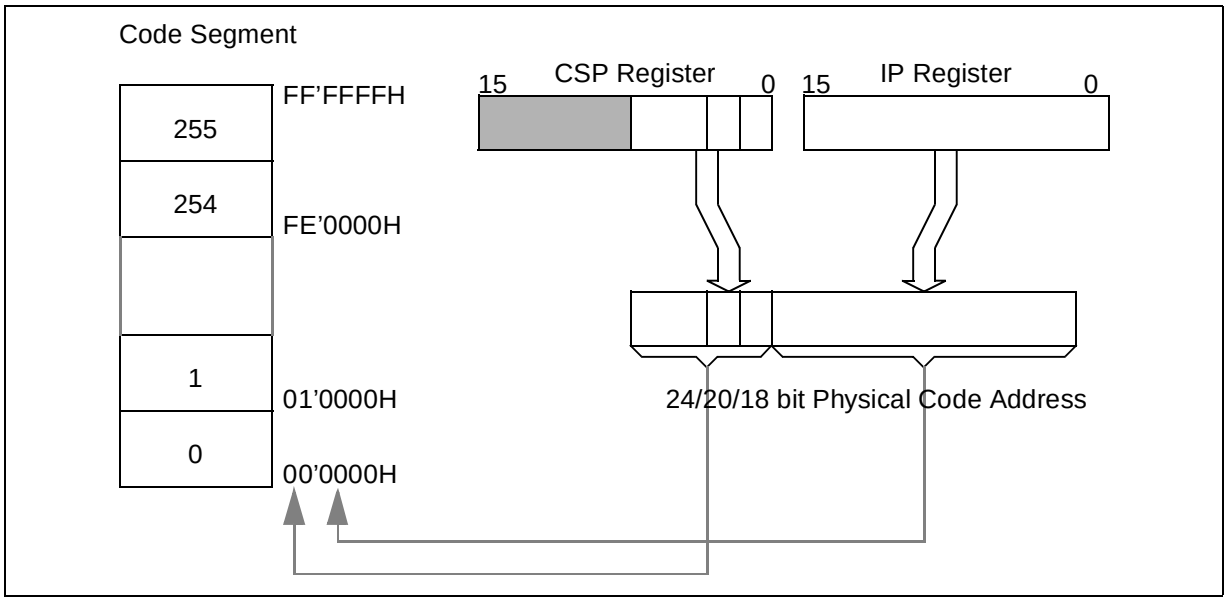


Figure 12 Addressing via the code segment pointer

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

The data page pointers DPP0, DPP1, DPP2, DPP3

These four non bit-addressable registers select up to four different data pages being active simultaneously at run-time. The lower 10 bits of each DPP register select one of the 1024 possible 16-Kbyte data pages, and the upper 6 bits are reserved for future use. The DPP registers give access to the entire memory space - in pages of 16Kbytes 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 makes it possible to access data pages 3...0 within segment 0, as shown in below. If data paging is not required, no further action is required.

DPP0 (FE00h / 00h) **SFR** **Reset Value: 0000h**

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

DPP1 (FE02h / 01h) **SFR** **Reset Value: 0001h**

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

DPP2 (FE04h / 02h) **SFR** **Reset Value: 0002h**

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

DPP3 (FE06h / 03h) **SFR** **Reset Value: 0003h**

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 DDP register (selected by the upper two bits of the 16-bit address). The 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 24/20/18-bit address.

For Non-segmented Memory Mode, only the two least significant bits of the implicitly selected DPP register are used to generate the physical address. **Extreme care should be taken when changing the content of a DPP register if a non-segmented memory model is selected, otherwise unexpected results can occur.**

In Segmented Memory Mode the selected number of segment address bits (9...2, 5...2 or 3...2) of the respective DPP register is output on the segment address pins (A23/A19/ A17...A16) of Port 4, for all external data accesses.

A DPP register can be updated by any instruction which is capable of modifying an SFR. Due to the Internal Instruction Pipeline, a new DPP value can not be used for the operand address calculation of an instruction immediately following the instruction which updates the DPP register.

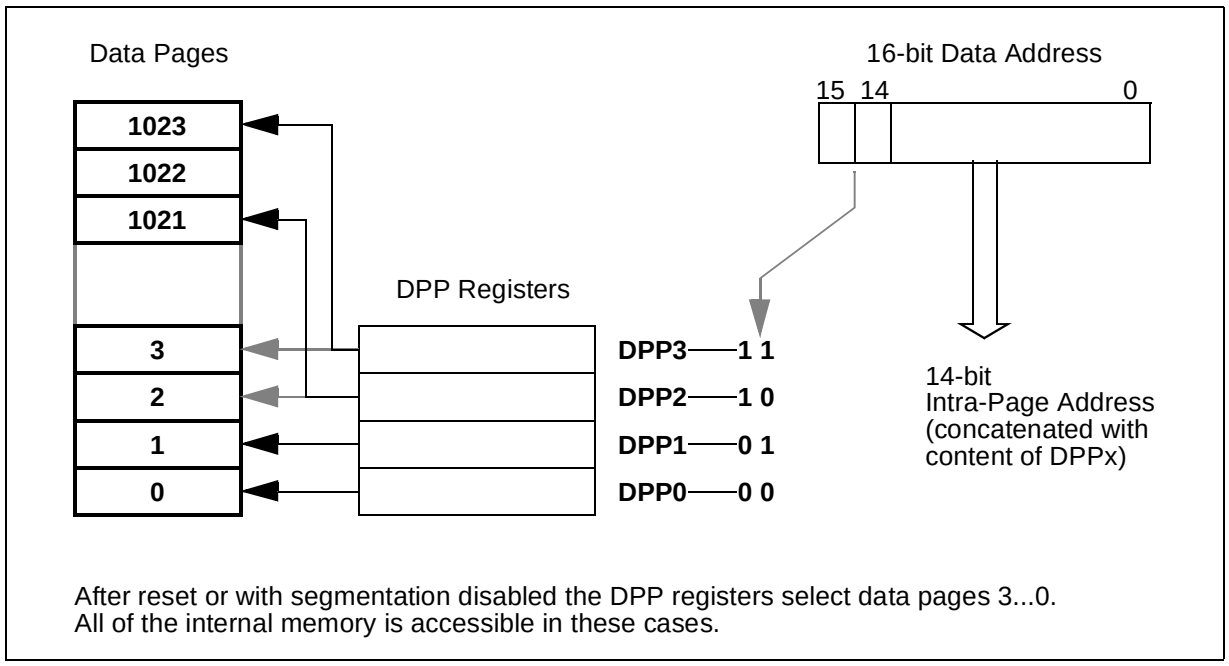


Figure 13 Addressing via the data page pointers

The context pointer CP

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

Note ***Ensure that the physical GPR address specified by the CP register and short GPR address is always at an internal RAM location. Otherwise, unexpected results may occur.***

- Do not set CP below 00'F600h or above 00'FDFEh.
- Be careful using the upper GPRs with CP above 00'FDE0h.

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

Due to the internal instruction pipeline, a new CP value can not be used for GPR address calculation of the instruction immediately following the instruction updating the CP register.

The switch context instruction (SCXT) saves the content of the CPregister on the stack and updates it with a new value in one machine cycle.

CP (FE10h / 08h) **SFR** **Reset Value: FC00h**

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.

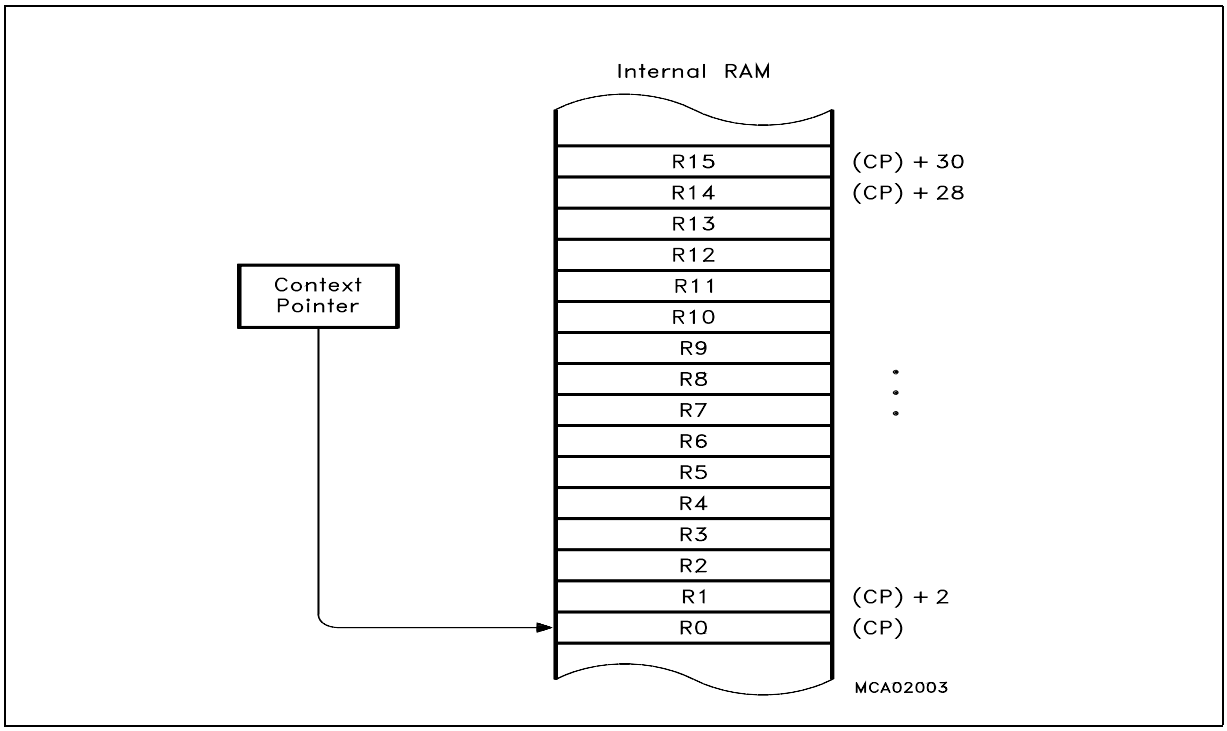


Figure 14 Register bank selection via the CP register

Several addressing modes use the CP register for address calculations.

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

GPRs used as indirect address pointers are always accessed wordwise. For some instructions, only the first four GPRs can be used as indirect address pointers. These GPRs are specified by 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 F0h to FFh interpret the four least significant bits as a short 4-bit GPR address. The four most significant bits are ignored. The respective physical GPR address calculation is identical to that for the short 4-bit GPR addresses. For single bit accesses on a GPR, the GPR's word address is

calculated as described, but the position of the bit within the word is specified by a separate additional 4-bit value.

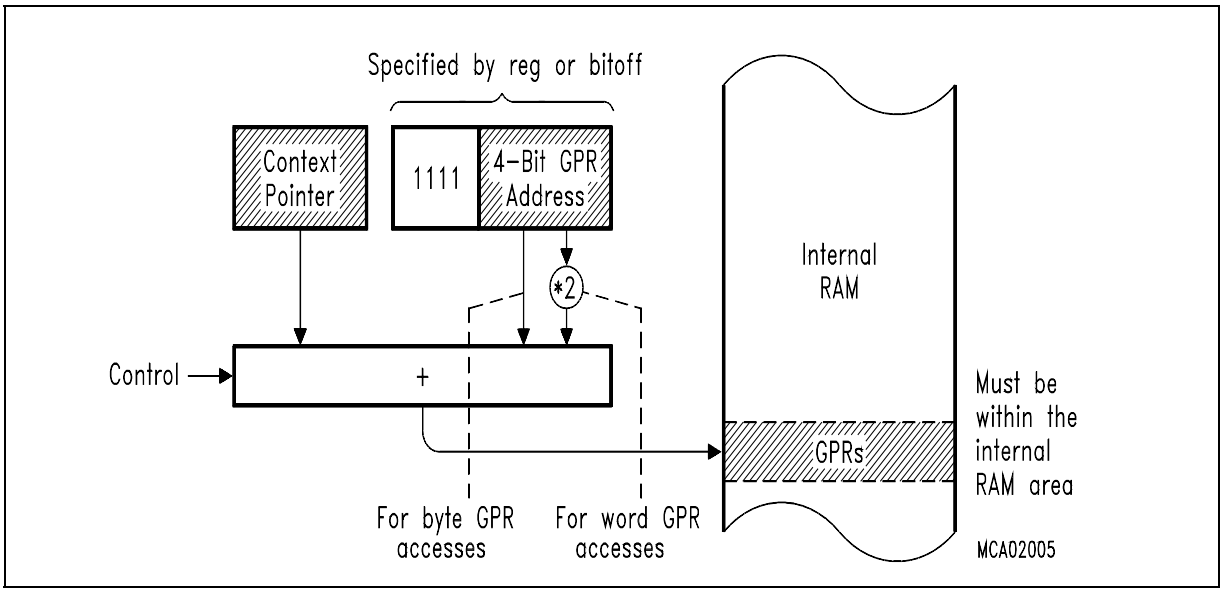


Figure 15 Implicit CP used by short GPR addressing modes

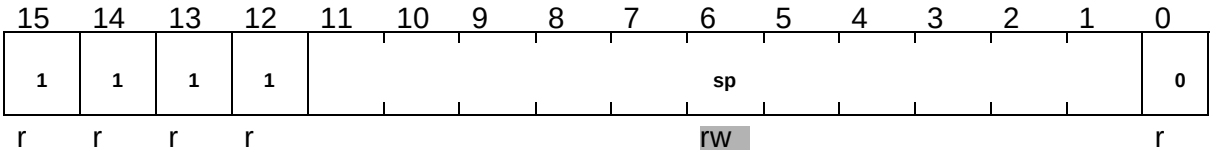
The stack pointer SP

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

Since the least significant bit the SPregister is tied to ’0’ and bits 15 through 12 are tied to ’1’ by hardware, the SP register can only contain values from F000h to FFFEh. This makes it possible to access a physical stack within the internal RAM of the ST10R272L. A virtual stack (usually bigger) can be realized via software. This mechanism is supported by registers STKOV and STKUN (see STKOV and STKUN descriptions below).

The SP register can be updated by any instruction which is capable of modifying an SFR. Due to the internal instruction pipeline, a POP or RETURN instruction must NOT immediately follow an instruction that updates the SP register.

SP (FE12h / 09h) **SFR** **Reset Value: FC00h**



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

The stack overflow pointer STKOV

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

Since the least significant bit of register STKOV is tied to ’0’ and bits 15 through 12 are tied to ’1’ by hardware, the STKOV register can only contain values from F000h to FFFEh.

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. Data in the bottom of the stack may have been overwritten by the status information stacked on servicing the stack overflow trap.
- **Automatic system stack flushing** uses the system stack as a ’Stack Cache’ for a bigger external user stack. In this case, the STKOV register 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 takes into account the worst possible case: when a stack overflow condition is detected 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.

STKOV (FE14h / 0Ah) **SFR** **Reset Value: FA00h**

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 underflow pointer STKUN

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

Since the least -significant-bit of the STKUN register is tied to ’0’ and bits 15 through 12 are tied to ’1’ by hardware, the STKUN register can only contain values from F000h to FFFEh.

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.

Scope of stack limit control

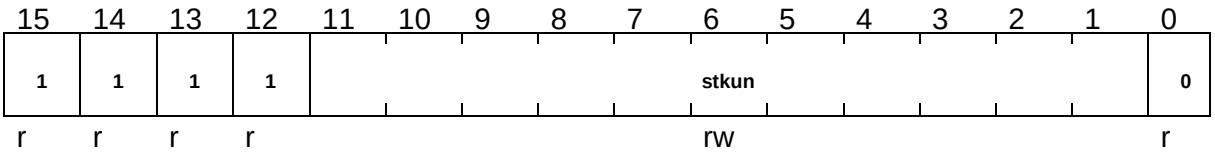
STKOV and STKUN detects when the stack pointer SP is moved outside the defined stack area either by ADD or SUB instructions, or by PUSH or POP operations (explicit or implicit, i.e. CALL or RET instructions).

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

- the stack pointer SP is directly updated via MOV instructions,

- the limits of the stack area (STKOV, STKUN) are changed, so that SP is outside of the new limits.

STKUN (FE16h / 0Bh) **SFR** **Reset Value: FC00h**



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

The multiply/divide high register MDH

This register is a part of the 32-bit multiply/divide register - used by the CPU for multiplication or 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, the MDHregister contains the 16-bit remainder.

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, registers MDH, MDL and MDC must be saved, to avoid erroneous results.

A detailed description of how to use the MDH register for programming multiply and divide algorithms can be found in “SYSTEM PROGRAMMING” on page 303.

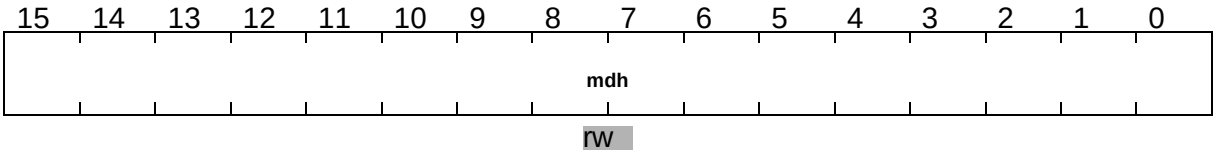
The multiply/divide low register MDL

This register is a part of the 32-bit multiply/divide register used by the CPU for multiplication or 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.

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.

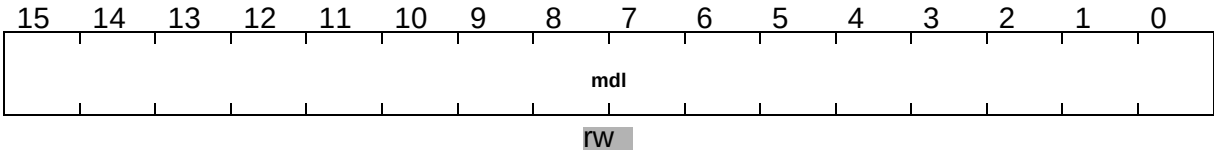
When a multiplication or division is interrupted before its completion, and when a new multiply or divide operation is to be performed within the interrupt service routine, registers MDL, MDH and MDC must be saved, to avoid erroneous results.

MDH (FE0Ch / 06h) **SFR** **Reset Value: 0000h**



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

MDL (FE0Eh/ 07h) **SFR** **Reset Value: 0000h**



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

The multiply/divide control register MDC

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

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

Note ***The MDRIU flag is the only portion of the MDC register which can be modified. The other parts of the MDC register are reserved for hardware use***

and should never be user modified. Otherwise, a correct continuation of an interrupted multiply or divide operation cannot be guaranteed.

MDC (FF0Eh / 87h) **SFR** **Reset Value: 0000h**

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.

The constant zeros register ZEROS

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

The constant ones register ONES

All bits of this bit-addressable register are fixed to '1', by hardware. This register is read-only. Register ONES can be used as a register-addressable constant of all ones, i.e. for bit

manipulation or mask generation. It can be accessed via any instruction capable of addressing an SFR.

ZEROS (FF1Ch / 8Eh) **SFR** **Reset Value: 0000h**

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

ONES (FF1Eh / 8Fh) **SFR** **Reset Value: FFFFh**

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 MULTIPLY-ACCUMULATE UNIT (MAC)

The MAC is a specialized co-processor added to the ST10R272L CPU core to improve the performance of signal processing algorithms. It includes:

- a multiply-accumulate unit
- an address generation unit, able to feed the MAC unit with 2 operands per cycle
- a repeat unit, to execute a series of multiply-accumulate instructions

New addressing capabilities enable the CPU to supply the MAC with up to 2 operands per instruction cycle. MAC instructions: multiply, multiply-accumulate, 32-bit signed arithmetic operations and the CoMOV transfer instruction have been added to the standard instruction set. Full details are provided in the ‘ST10 Family Programming Manual’.

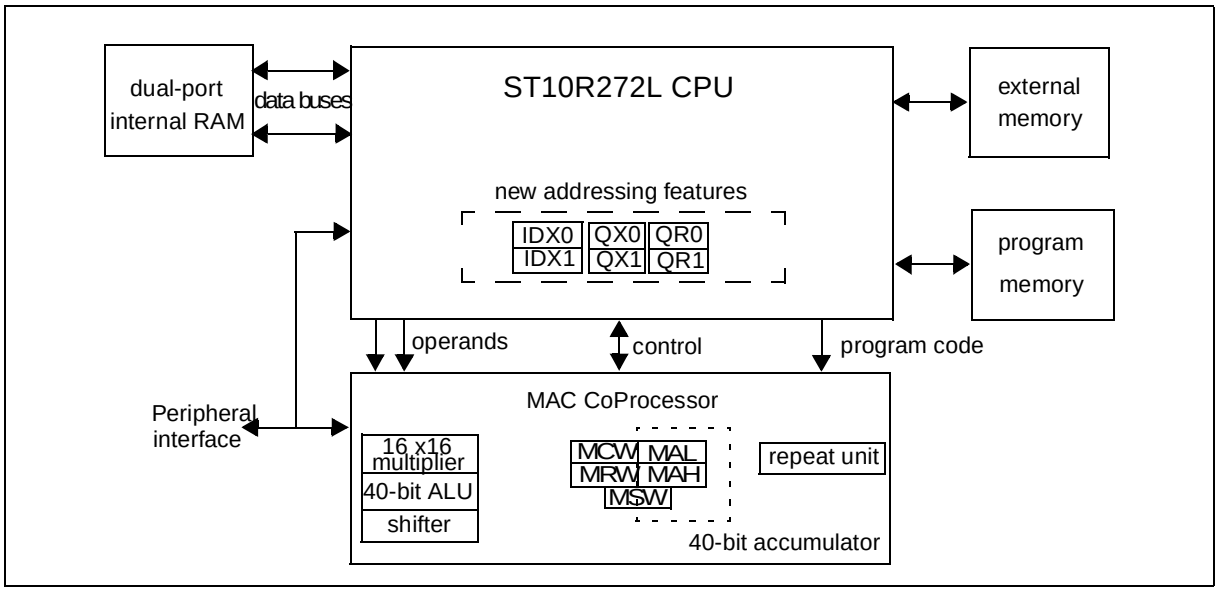


Figure 16 MAC architecture

5.1 MAC features

Enhanced addressing capabilities

- Double indirect addressing mode with pointer post-modification.
- Parallel Data Move allows one operand move during Multiply-Accumulate instructions without penalty.
- CoSTORE instruction (for fast access to the MAC SFRs) and CoMOV (for fast memory to memory table transfer).

General

- Two-cycle execution for all MAC operations.
- 16 x 16 signed/unsigned parallel multiplier.
- 40-bit signed arithmetic unit with automatic saturation mode.
- 40-bit accumulator.
- 8-bit left/right shifter.
- Scaler (one-bit left shifter)
- Data limiter
- Full instruction set with multiply and multiply-accumulate, 32-bit signed arithmetic and compare instructions.
- Three 16-bit status and control registers: MSW: MAC Status Word, MCW: MAC Control Word, MRW: MAC Repeat Word.

Program control

- Repeat Unit allows some MAC co-processor instructions to be repeated up to 8192 times. Repeated instructions may be interrupted.
- MAC interrupt (Class B Trap) on MAC condition flags.

5.2 MAC operation

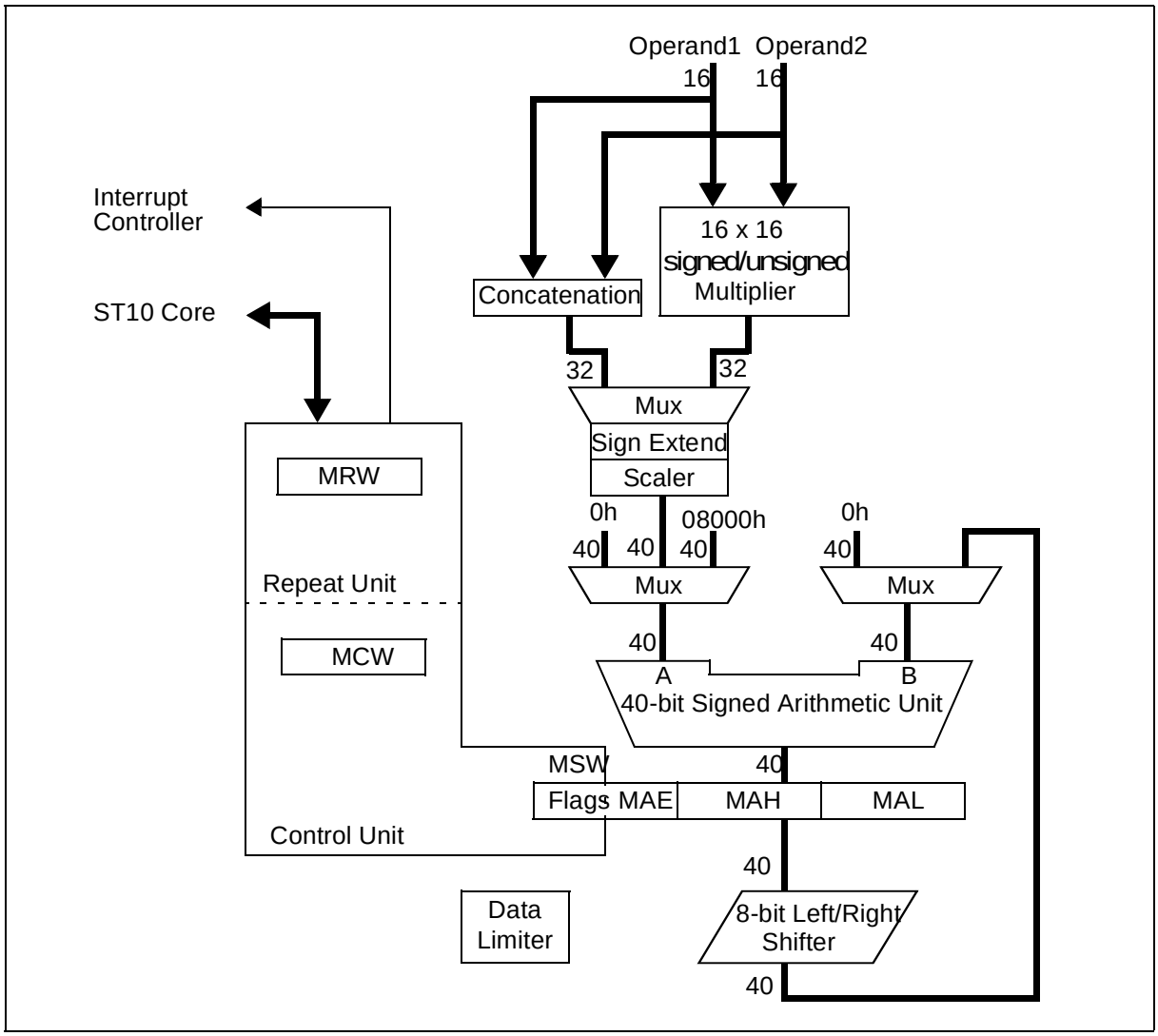


Figure 17 MAC architecture

5.2.1 Instruction pipelining

All MAC instructions use the 4-stage pipeline. During each stage the following tasks are performed:

- FETCH: All new instructions are double-word instructions.
- DECODE: If required, operand addresses are calculated and the resulting operands are fetched. IDX and GPR pointers are post-modified if necessary.

- EXECUTE: Performs the MAC operation. At the end of the cycle, the Accumulator and the MAC condition flags are updated if required. Modified GPR pointers are written-back during this stage, if required.
- WRITEBACK: Operand write-back in the case of parallel data move.

Note *At least one instruction which does not use the MAC must be inserted between two instructions that read from a MAC register. This is because the Accumulator and the status of the MAC are modified during the Execute stage. The CoSTORE instruction has been added to allow access to the MAC registers immediately after a MAC operation.*

5.2.2 Address generation

MAC instructions can use some standard ST10 addressing modes such as GPR direct or #data4 for immediate shift value.

New addressing modes have been added to supply the MAC with two new operands per instruction cycle. These allow indirect addressing with address pointer post-modification.

Double indirect addressing requires two pointers. Any GPR can be used for one pointer, the other pointer is provided by one of two specific SFRs IDX0 and IDX1. Two pairs of offset registers QR0/QR1 and QX0/QX1 are associated with each pointer (GPR or IDX_i). The GPR pointer allows access to the entire memory space, but IDX_i are limited to the internal Dual-Port RAM, except for the CoMOV instruction.

The following table shows the various combinations of pointer post-modification for each of these 2 new addressing modes. In this document the symbols "[$Rw_n\otimes$]" and "[$IDX_i\otimes$]" refer to these addressing modes.

Symbol	Mnemonic	Address Pointer Operation
"[$IDX_i\otimes$]" stands for	[IDX_i]	(IDX_i) ← (IDX_i) (no-op)
	[IDX_i+]	(IDX_i) ← (IDX_i) +2 (i=0,1)
	[IDX_i -]	(IDX_i) ← (IDX_i) -2 (i=0,1)
	[IDX_i + QX_j]	(IDX_i) ← (IDX_i) + (QX_j) (i, j =0,1)
	[IDX_i - QX_j]	(IDX_i) ← (IDX_i) - (QX_j) (i, j =0,1)

Table 8 Pointer post-modification combinations for IDXi and Rwn

Symbol	Mnemonic	Address Pointer Operation
"[Rw_n⊗]" stands for	[Rwn]	(Rwn) ← (Rwn) (no-op)
	[Rwn+]	(Rwn) ← (Rwn) +2 (n=0-15)
	[Rwn-]	(Rwn) ← (Rwn) -2 (k=0-15)
	[Rwn+QR_j]	(Rwn) ← (Rwn) + (QR_j) (n=0-15;j =0,1)
	[Rwn - QR_j]	(Rwn) ← (Rwn) - (QR_j) (n=0-15; j =0,1)

Table 8 Pointer post-modification combinations for IDXi and Rwn (Continued)

For the CoMACM class of instruction, Parallel Data Move mechanism is implemented. This class of instruction is only available with double indirect addressing mode. Parallel Data Move allows the operand pointed by IDX_i to be moved to a new location in parallel with the MAC operation. The write-back address of Parallel Data Move is calculated depending on the post-modification of IDX_i. It is obtained by the reverse operation than the one used to calculate the new value of IDX_i. The following table shows these rules.

Instruction	Writeback Address
CoMACM [IDX_i+],...	<IDX_i-2>
CoMACM [IDX_i-],...	<IDX_i+2>
CoMACM [IDX_i+QX_j],...	<IDX_i-QX_j>
CoMACM [IDX_i-QX_j],...	<IDX_i+QX_j>

Table 9 Parallel data move addressing

The Parallel Data Move shifts a table of operands in parallel with a computation on those operands. Its specific use is for signal processing algorithms like filter computation. The following figure gives an example of Parallel Data Move with CoMACM instruction.

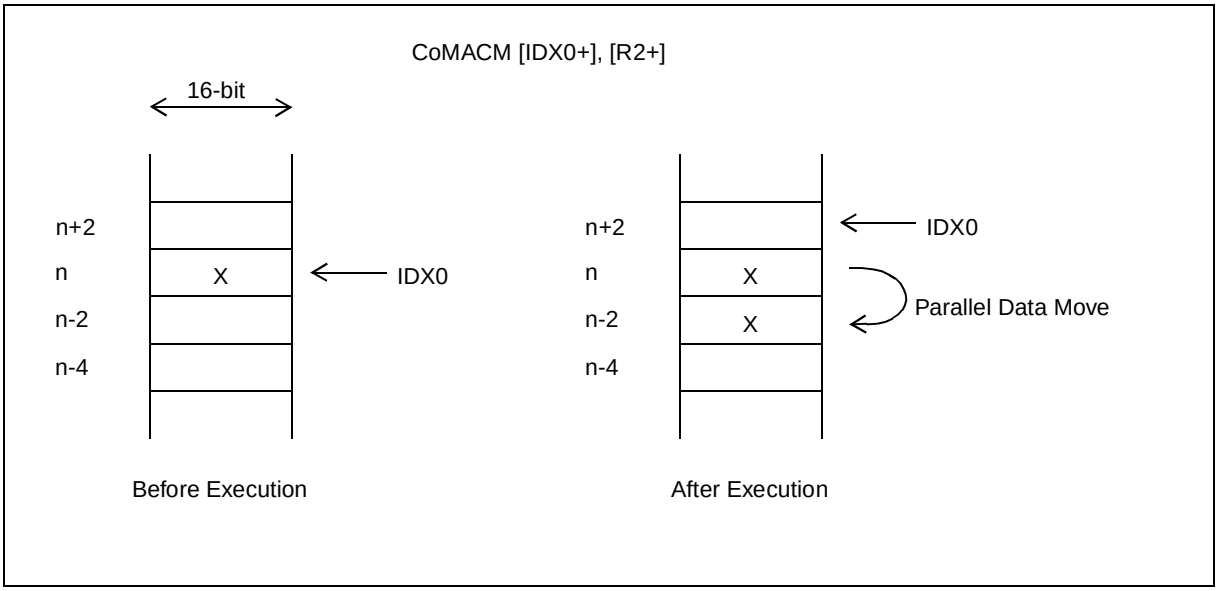


Figure 18 Example of parallel data move

5.2.3 16 x 16 signed/unsigned parallel multiplier

The multiplier executes 16 x 16-bit parallel signed/unsigned fractional and integer multiplies. The multiplier has two 16-bit input ports, and a 32-bit product output port. The input ports can accept data from the MA-bus and from the MB-bus. The output is sign-extended and then feeds a scaler that shifts the multiplier output according to the shift mode bit MP specified in the co-processor Control Word (MCW). The product can be shifted one bit left to compensate for the extra sign bit gained in multiplying two 16-bit signed (2's complement) fractional numbers if bit MP is set.

5.2.4 40-bit signed arithmetic unit

The arithmetic unit over 32 bits wide to allow intermediate overflow in a series of multiply/ accumulate operations. The extension flag E, contained in the most significant byte of MSW, is set when the Accumulator has overflowed beyond the 32-bit boundary, that is, when there are significant (non-sign) bits in the top eight (signed arithmetic) bits of the Accumulator.

The 40-bit arithmetic unit has two 40-bit input ports A and B. The A-input port accepts data from 4 possible sources: 00,0000,0000h, 00,0000,8000h (round), the sign-extended product, or the sign-extended data conveyed by the 32-bit bus resulting from the concatenation of MA- and MB-buses. Product and Concatenation can be shifted left by one according to MP for the multiplier or to the instruction for the concatenation. The B-input port

is fed either by the 40-bit shifted/not shifted and inverted/not inverted accumulator or by 00,0000,0000h. A-input and B-input ports can receive 00,0000,0000h to allow direct transfers from the B-source and A-source, respectively, to the Accumulator (case of Multiplication, Shift.). The output of the arithmetic unit goes to the Accumulator.

It is also possible to saturate the Accumulator on a 32-bit value, automatically after every accumulation. Automatic saturation is enabled by setting the saturation bit MS in the MCW register. When the Accumulator is in the saturation mode and an 32-bit overflow occurs, the accumulator is loaded with either the most positive or the most negative value representable in a 32-bit value, depending on the direction of the overflow. The value of the Accumulator upon saturation is 00,7fff,ffffh (positive) or ff,8000,0000h (negative) in signed arithmetic. Automatic saturation sets the SL flag MSW. This flag is a sticky flag which means it stays set until it is explicitly reset by the user.

40-bit overflow of the Accumulator sets the SV flag in MSW. This flag is also a sticky flag.

5.2.5 40-bit accumulator register

The 40-bit Accumulator consists of three SFR registers MAH, MAL and MAE. MAH and MAL are 16-bit wide. MAE is 8-bit wide and is contained within the least significant byte of MSW. Most co-processor operations specify the 40-bit Accumulator register as source and/or destination operand.

5.2.6 Data limiter

Saturation arithmetic is also provided to selectively limit overflow, when reading the accumulator by means of a CoSTORE <destination> MAS instruction. Limiting is performed on the MAC Accumulator. If the contents of the Accumulator can be represented in the destination operand size without overflow, the data limiter is disabled and the operand is not modified. If the contents of the accumulator cannot be represented without overflow in the destination operand size, the limiter will substitute a ‘limited’ data as explained in the following table.

Register	E bit	N bit	Output of the Limiter
x	0	x	unchanged
MAS	1	0	7fffh
MAS	1	1	8000h

Table 10 Data Limit Values

Note *In this case, the accumulator and the status register are not affected. MAS readable from a CoSTORE instruction.*

5.2.7 Accumulator shifter

The Accumulator shifter is a parallel shifter with a 40-bit input and a 40-bit output. The source operand of the shifter is the Accumulator and the possible shifting operations are:

- No shift (Unmodified)
- Up to 8-bit Arithmetic Left Shift
- Up to 8-bit Arithmetic Right Shift

E, SV and SL bits from MSW are affected by Left shifts, therefore if the saturation mechanism is enabled (MS), the behavior is similar to the one of the arithmetic unit. The carry flag C is also affected by left shifts.

5.2.8 Repeat unit

The MAC includes a repeat unit allowing the repetition of some co-processor instructions up to 2^{13} (8192) times. The repeat count may be specified either by an immediate value (up to 31 times) or by the content of the Repeat Count (bits 12 to 0) in the MAC Repeat Word (MRW). If the Repeat Count equals “N” the instruction will be executed “N+1” times. At each iteration of a cumulative instruction the Repeat Count is tested for zero. If it is zero the instruction is terminated else the Repeat Count is decremented and the instruction is repeated. During such a repeat sequence, the Repeat Flag in MRW is set until the last execution of the repeated instruction.

The syntax of repeated instructions is shown in the following examples:

```
1          Repeat #24 times
           CoMAC[IDX0+],[R0+]        ; repeated 24 times
```

In example 1, the instruction is repeated according to a 5-bit immediate value. The Repeat Count in MRW is automatically loaded with this value minus one (MRW=23).

```
1          MOV MRW, #00FFh           ; load MRW
           NOP                       ; instruction latency
           Repeat MRW times
           CoMACM [IDX1-],[R2+]      ; repeated 256 times
```

In this example, the instruction is repeated according to the Repeat Count in MRW. Notice that due to the pipeline processing at least one instruction should be inserted between the write of MRW and the next repeated instruction.

Repeat sequences may be interrupted. When an interrupt occurs during a repeat sequence, the sequence is stopped and the interrupt routine is executed. The repeat sequence resumes at the end of the interrupt routine. During the interrupt, MR remains set, indicating that a repeated instruction has been interrupted and the Repeat Count holds the number

(minus 1) of repetition that remains to complete the sequence. If the Repeat Unit is used in the interrupt routine, MRW must be saved by the user and restored before the end of the interrupt routine.

Note *The Repeat Count should be used with caution. In this case MR should be written as 0. In general MR should not be set by the user otherwise correct instruction processing can not be guaranteed.*

5.2.9 MAC interrupt

The MAC can generate an interrupt according to the value of the status flags C (carry), SV (overflow), E (extension) or SL (limit) of the MSW. The MAC interrupt is globally enabled when the MIE flag in MCW is set. When it is enabled the flags C, SV, E or SL can triggered a MAC interrupt when they are set provided that the corresponding mask flag CM, VM, EM or LM in MCW is also set. A MAC interrupt request set the MIR flag in MSW, this flag must be reset by the user during the interrupt routine otherwise the interrupt processing restarts when returning from the interrupt routine.

The MAC interrupt is implemented as a Class B hardware trap (trap number Ah - trap priority I). The associated Trap Flag in the TFR register is MACTRP, bit #6 of the TFR (Remember that this flag must also be reset by the user in the case of an MAC interrupt request).

As the MAC status flags are updated (or eventually written by software) during the Execute stage of the pipeline, the response time of a MAC interrupt request is 3 instruction cycles (see Figure 3). It is the number of instruction cycles required between the time the request is sent and the time the first instruction located at the interrupt vector location enters the pipeline. Note that the IP value stacked after a MAC interrupt does not point to the instruction that triggers the interrupt.

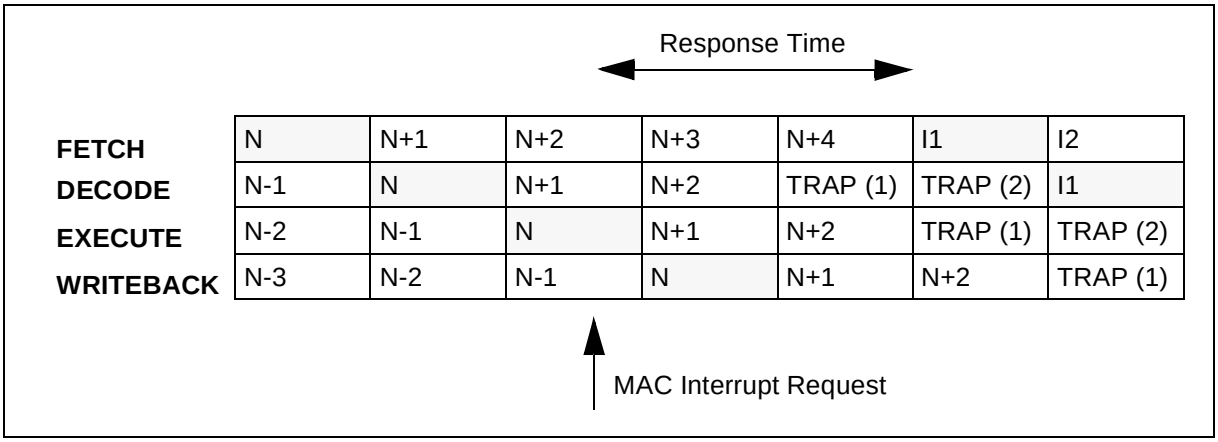


Figure 19 Pipeline diagram for MAC interrupt response time

5.2.10 Number representation & rounding

The MAC supports the two's-complement representation of binary numbers. In this format, the sign bit is the MSB of the binary word. This is set to zero for positive numbers and set to one for negative numbers. Unsigned numbers are supported only by multiply/ multiply-accumulate instructions which specifies whether each operand is signed or unsigned.

In two's complement fractional format, the N-bit operand is represented using the 1.[N-1] format (1 signed bit, N-1 fractional bits). Such a format can represent numbers between -1 and $+1-2^{-[N-1]}$. This format is supported when MP of MCW is set.

The MAC implements 'two's complement rounding'. With this rounding type, one is added to the bit to the right of the rounding point (bit 15 of MAL), before truncation (MAL is cleared).

5.3 MAC register set

5.3.1 Address registers

The new addressing modes require new (E)SFRs: 2 address pointers IDX0 / IDX1 and 4 offset registers QX0 / QX1 and QR0 / QR1.

IDX0 (FF08h / 84h) **SFR** **Reset Value: 0000h**

IDX1 (FF0Ah / 85h) **SFR** **Reset Value: 0000h**

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

Bit	Function
IDXy	16-bit IDXy address

QX0 (F000h / 00h) **ESFR** **Reset Value: 0000h**

QX1 (F002h / 01h) **ESFR** **Reset Value: 0000h**

QR0 (F004h / 02h) **ESFR** **Reset Value: 0000h**

QR1 (F006h / 03h) **ESFR** **Reset Value: 0000h**

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

Bit	Function
QRz/QXz	**16-bit address offset for IDXy pointers (QXz) or GPR pointers (QRz).** As MAC instructions handle word operands, bit 0 of these offset registers is hardwired to '0'.

5.3.2 Accumulator & control registers

The MAC unit SFRs include the 40-bit Accumulator (MAL, MAH and the low byte of MSW) and 3 control registers: the status word MSW, the control word MCW and the repeat word MRW.

MAH and MAL are located in the non bit-addressable SFR space.

MAH (FE5Eh / 2Fh) **SFR** **Reset Value: 0000h**

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

Bit	Function
MAH	MAC Unit Accumulator High (bits [31..16])

MAL (FE5Ch / 2Eh) **SFR** **Reset Value: 0000h**

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

Bit	Function
MAL	MAC Unit Accumulator Low (bits [15..0])

MSW (FFDEh / EFh) **SFR** **Reset Value: 0200h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MIR	-	SL	E	SV	C	Z	N	MAE							
r	-	rw	rw	rw	rw	rw	rw	rw							

Bit	Function
MIR	**MAC Interrupt Request** Set when the MAC Unit generates an interrupt request.
SL	**Sticky Limit Flag** Set when the result of a MAC operation is automatically saturated. Also used for CoMIN, CoMAX instructions to indicate that the Accumulator has changed. It remains set until it is explicitly reset by software.
E	**Extension Flag** Set when MAE contains significant bits at the end of a MAC operation

Bit	Function
SV	**Sticky Overflow Flag** Set when a MAC operation produces a 40-bit arithmetic overflow. It remains set until it is explicitly reset by software.
C	**Carry Flag** Set when a MAC operation produces a carry or a borrow bit.
Z	**Zero Flag** Set when the Accumulator is zero at the end of a MAC operation.
N	**Negative Flag** Set when the Accumulator is negative at the end of a MAC operation.
MAE	**Accumulator Extension (bits [39:32])**

Note *The MAC condition flags are evaluated if required by the instruction being executed. In particular they are not affected by any instruction of the regular instruction set. In consequence, their values may not be consistent with the Accumulator content. For example, loading the Accumulator with MOV instructions will not modify the condition flags*

MCW (FFDCH / EEh **SFR** **Reset Value: 0000h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MIE	LM	EM	VM	CM	MP	MS	-								
rw	rw	rw	rw	rw	rw	rw	-	-	-	-	-	-	-	-	-

Bit	Function
MIE	**MAC Interrupt Enable** '0': MAC interrupt globally disabled, '1': MAC interrupt globally enabled.
LM	**SL Mask** When set, the SL Flag can generate a MAC interrupt request.
EM	**E Mask** When set, the E Flag can generate a MAC interrupt request.
VM	**SV Mask** When set, the SV Flag can generate a MAC interrupt request.
CM	**C Mask** When set, the C Flag can generate a MAC interrupt request.
MP	**Product Shift Mode** When set, enables the one-bit left shift of the multiplier output in case of a signed-signed multiplication.
MS	**Saturation Mode** When set, enables automatic 32-bit saturation of the result of a MAC operation.

MRW (FFDAh / EDh) **SFR** **Reset Value: 0000h**

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

Bit	Function
MR	**Repeat Flag** Set when a repeated instruction is executed

Bit	Function
Repeat Count	**13-bit unsigned integer value** Indicates the number of time minus one a repeated instruction must be executed.

Note *As for the CPU Core SFRs, any write operation with the regular instruction set to a single byte of a MAC SFR clears the non-addressed complementary byte within the specified SFR. Non-implemented SFR bits cannot be modified and will always supply a read value of'0'.*

These registers are mapped in the SFR space and can addressed by the regular instruction set like any SFR. As mentioned previously, they can also be addressed by the new instruction CoSTORE. This instruction allows the user to access the MAC registers without any pipeline side effect. CoSTORE uses a specific 5-bit addressing mode called CoReg. The following table gives the address of the MAC registers in this CoReg addressing mode.

Registers	Description	Address
MSW	MAC-Unit Status Word	00000b
MAH	MAC-Unit Accumulator High	00001b
MAS	"limited" MAH /signed	00010b
MAL	MAC-Unit Accumulator Low	00100b
MCW	MAC-Unit Control Word	00101b
MRW	MAC-Unit Repeat Word	00110b

Table 11 MAC register address in CoReg addressing mode

5.4 MAC instruction set summary

The following table gives an overview of the MAC instruction set. All the mnemonics are listed with the addressing modes that can be used with each instruction.
For each combination of mnemonic and addressing mode this table indicates if it is repeatable or not

For full details of the MAC instruction set, refer to the 'ST10 Family Programming Manual'.

Mnemonic	Addressing Modes	Rep
CoMUL	Rw_n, Rw_m	No
CoMULu	[IDX_i⊗], [Rw_m⊗]	No
CoMULus	Rw_n, [Rw_m⊗]	No
CoMULsu		
CoMUL-		
CoMULu-		
CoMULus-		
CoMULsu-		
CoMUL, rnd		
CoMULu, rnd		
CoMULus, rnd		
CoMULsu, rnd		
CoMAC	Rw_n, Rw_m	No
CoMACu	[IDX_i⊗], [Rw_m⊗]	Yes
CoMACus	Rw_n, [Rw_m⊗]	Yes
CoMACsu		
CoMAC-		
CoMACu-		
CoMACus-		
CoMACsu-		
CoMAC, rnd		
CoMACu, rnd		
CoMACus, rnd		
CoMACsu, rnd		
CoMACR		
CoMACRu		
CoMACRus		
CoMACRsu		
CoMACR, rnd		
CoMACRu, rnd		
CoMACRus, rnd		
CoMACRsu, rnd		
CoNOP	[Rw_m⊗]	Yes
	[IDX_i⊗]	Yes
	[IDX_i⊗], [Rw_m⊗]	Yes
CoNEG	-	No
CoNEG, rnd		
CoRND		
CoSTORE	Rw_n, CoReg	No
	[Rw_n⊗], Coreg	Yes
CoMOV	[IDX_i⊗], [Rw_m⊗]	Yes
CoMACM	[IDX_i⊗], [Rw_m⊗]	Yes
CoMACMu		
CoMACMus		
CoMACMsu		
CoMACM-		
CoMACMu-		
CoMACMus-		
CoMACMsu-		
CoMACM, rnd		
CoMACMu, rnd		
CoMACMus, rnd		
CoMACMsu, rnd		
CoMACMR		
CoMACMRu		
CoMACMRus		
CoMACMRsu		
CoMACMR, rnd		
CoMACMRu, rnd		
CoMACMRus, rnd		
CoMACMRsu, rnd		
CoADD	Rw_n, Rw_m	No
CoADD2	[IDX_i⊗], [Rw_m⊗]	Yes
CoSUB	Rw_n, [Rw_m⊗]	Yes
CoSUB2		
CoSUBR		
CoSUB2R		
CoMAX		
CoMIN		
CoLOAD	Rw_n, Rw_m	No
CoLOAD-	[IDX_i⊗], [Rw_m⊗]	No
CoLOAD2	Rw_n, [Rw_m⊗]	No
CoLOAD2-		
CoCMP		
CoSHL	Rw_m	Yes
CoSHR	#data4	No
CoASHR	[Rw_m⊗]	Yes
CoASHR, rnd		
CoABS	-	No
	Rw_n, Rw_m	No
	[IDX_i⊗], [Rw_m⊗]	No
	Rw_n, [Rw_m⊗]	No

Table 12 MAC instruction set summary

6 INTERRUPT AND TRAP FUNCTIONS

The ST10R272L architecture supports several mechanisms for fast and flexible response to the service requests that can be generated from various sources, internal or external to the microcontroller. Any of these interrupt requests can be programmed to be serviced, either by the Interrupt Controller or by the Peripheral Event Controller (PEC).

Normal Interrupt Processing: In a standard interrupt service, program execution is suspended and a branch to the interrupt service routine is performed. 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): For a PEC service, just one cycle is 'stolen' from the current CPU activity. A PEC service is a single, byte or word data transfer between any two memory locations, with an additional increment of either the PEC source or the destination pointer. A PEC-transfer counter is decremented for each PEC service, except in the continuous transfer mode. When this counter reaches zero, a standard interrupt is performed to the corresponding source-related vector location. PEC services are very well suited, for example, to the transmission or reception of blocks of data. The ST10R272L has 8 PEC channels, each of which offers fast interrupt-driven data transfer capabilities.

Trap Functions: Software interrupts are supported by means of the 'TRAP' instruction in combination with an individual trap (interrupt) number. 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: Fast external interrupt inputs service external interrupts with high precision requirements. These fast interrupt inputs include programmable edge detection (rising edge, falling edge or both edges).

6.1 Interrupt system structure

The ST10R272L provides 20 separate interrupt nodes that may be assigned to 16 priority levels. 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.

A control register contains an interrupt request flag, an interrupt enable flag and an interrupt priority bitfield for each of the possible interrupt sources. Via its related register, each source

can be programmed to one of sixteen interrupt priority levels. Once having been accepted by the CPU, an interrupt service can only be interrupted by a higher priority service request. For standard interrupt processing, each of the possible interrupt sources has a dedicated vector location.

Each interrupt service request is activated by one specific event. The only exceptions are the two serial channels, where an error interrupt request can be generated by different kinds of error. However, specific status flags which identify the type of error are implemented in the serial channels' control registers.

The ST10R272L 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 ST10R272L's address space (segment 0). The Jump Table is made up of jump instructions that transfer control to the interrupt or trap service routines which may be located anywhere in the address space. The entries to 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.

6.1.1 Interrupt sources

The table below lists all sources that are capable of requesting interrupt or PEC services in the ST10R272L, the associated interrupt vectors, their locations and the associated trap numbers. It also lists the Interrupt Request Flags mnemonic and the Interrupt Enable flags. The mnemonics has two parts: a source part and a function part (IR=Interrupt Request flag, IE=Interrupt Enable flag).

Source of Interrupt or PEC Service Request	Request Flag	Enable Flag	Interrupt Vector	Vector Location	Trap Number
External Interrupt 0	CC8IR	CC8IE	CC8INT	60h	18h
External Interrupt 1	CC9IR	CC9IE	CC9INT	64h	19h
External Interrupt 2	CC10IR	CC10IE	CC10INT	68h	1Ah

Table 13 List of possible interrupt sources, flags, vector and trap numbers

Source of Interrupt or PEC Service Request	Request Flag	Enable Flag	Interrupt Vector	Vector Location	Trap Number
External Interrupt 3	CC11IR	CC11IE	CC11INT	6Ch	1Bh
GPT1 Timer 2	T2IR	T2IE	T2INT	88h	22h
GPT1 Timer 3	T3IR	T3IE	T3INT	8Ch	23h
GPT1 Timer 4	T4IR	T4IE	T4INT	90h	24h
GPT2 Timer 5	T5IR	T5IE	T5INT	94h	25h
GPT2 Timer 6	T6IR	T6IE	T6INT	98h	26h
GPT2 CAPREL Register	CRIR	CRIE	CRINT	9Ch	27h
ASC0 Transmit	S0TIR	S0TIE	S0TINT	A8h	2Ah
ASC0 Transmit Buffer	S0TBIR	S0TBIE	S0TBINT	11Ch	47h
ASC0 Receive	S0RIR	S0RIE	S0RINT	ACh	2Bh
ASC0 Error	S0EIR	S0EIE	S0EINT	B0h	2Ch
PWM Channel 3	PWMIR	PWMIE	PWMINT	FCh	3Fh
SSP Interrupt	XP1IR	XP1IE	XP1INT	104h	41h
PLL Unlock	XP3IR	XP3IE	XP3INT	10Ch	43h

Table 13 List of possible interrupt sources, flags, vector and trap numbers

6.1.2 Normal interrupt processing and PEC service

During each instruction cycle, one of the sources which require PEC or interrupt processing is selected according to its interrupt priority. Interrupts and PEC request priority is programed in two levels.

- CPU interrupt priority level - defines the priority level for the arbitration of requests.
- Group priority - defines the internal order for simultaneous requests of the same priority.

At the end of each instruction cycle, the interrupt system decides which source request has the highest priority. This source request is then serviced if its priority is higher than the current CPU priority in the PSW register.

6.1.3 Interrupt system registers

Interrupt processing is controlled globally by the PSW register through the 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). Therefore, the CPU accepts requests based on 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 PEC service channel task.

6.1.4 Interrupt control registers

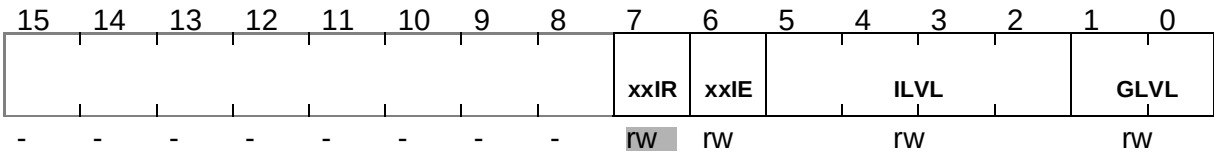
All interrupt control registers are organized identically.

- The lower 8 bits contain the source interrupt status information; this is required during one round of prioritization,
- The upper 8 bits are reserved.
- They are bit addressable and all bits can be read or written to by software - interrupt sources can be programmed or modified with one instruction.

When accessing interrupt control registers through instructions which operate on word data-types, their upper 8 bits (15...8) return zeros and discard written data.

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

xxIC (yyyyh / zzh) **SFR** **Reset Value: - - 00h**



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. Fh: Highest priority level 0h: 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 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 0000b is the lowest and 1111b is the highest priority level.

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

The **Interrupt request flag** is set by hardware whenever a service request from the respective source occurs. It is cleared automatically on entry into the interrupt service routine or on a PEC service. For PEC services, the interrupt request flag remains set if the COUNT field in the PECCx register of the selected PEC channel decrements to zero. Therefore, a normal CPU interrupt can respond to a completed PEC block transfer.

Note Modifying the interrupt request flag by software causes the same effects as if it had been set or cleared by hardware.

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.

On 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 (see "Interrupt control functions in the PSW" on page 89).

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

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

Priority level 0000b 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 0000b will terminate 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 ILVL (LSB) and GLVL (see figure below). So programming a source to priority level 15 (ILVL=1111b) selects the PEC channel group 7...4, and programming a source to priority level 14 (ILVL=1110b) selects the PEC channel group 3...0. The actual PEC channel number is then determined by the group priority field GLVL.

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

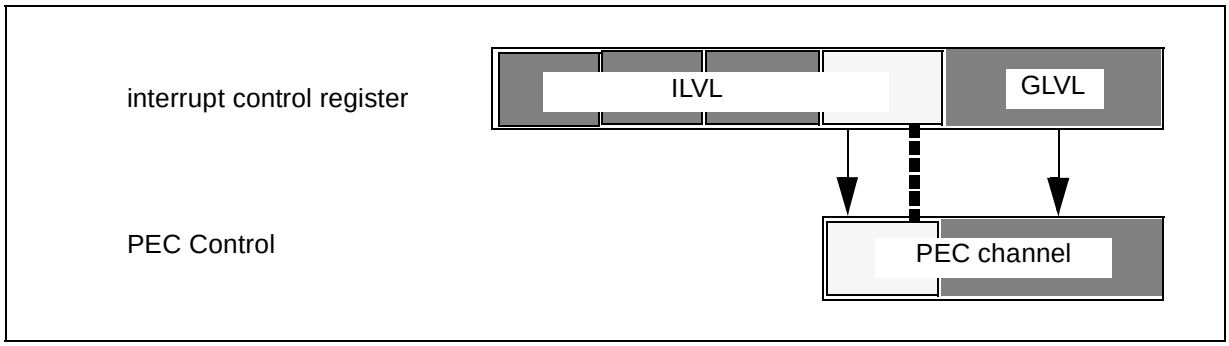


Figure 20 Priority levels and PEC channels

The following table shows - in a few examples - where an action is triggered by the programming of an interrupt control register.

Priority Level		Type of Service	
ILVL	**GLVL**	**COUNT = 00h**	**COUNT ≠ 00h**
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!

Table 14 Triggering an action with the interrupt control register (examples)

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.*

6.1.5 Interrupt control functions in the PSW

The Processor Status Word (PSW) is divided into 2 functional parts: the lower byte of the PSW represents the arithmetic status of the CPU, the upper byte of the PSW controls the

interrupt system of the ST10R272L and the arbitration mechanism for the external bus interface.

Note *Pipeline effects must be considered when enabling/disabling interrupt requests by modifying the PSW register (ref to “Processor status word PSW” on page 50).*

PSW (FF10h / 88h) **SFR** **Reset Value: 0000h**

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** (see “Saving the status during interrupt service” on page 96) Define the current status of the CPU (ALU, multiplication unit).
HLDEN	**HOLD Enable** (Enables External Bus Arbitration) 0: Bus arbitration disabled, P6.7...P6.5 may be used for general purpose IO 1: Bus arbitration enabled, P6.7...P6.5 serve as $\overline{\text{BREQ}}$, $\overline{\text{HLDA}}$, $\overline{\text{HOLD}}$, resp.
ILVL	**CPU Priority Level** Defines the current priority level for the CPU Fh: Highest priority level 0h: 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 CPU priority level. It shows the priority level of the routine that is currently being executed. On the entry into an interrupt service routine, this bit field is updated with the priority level of the request that is being serviced. The PSW is saved on the system stack. 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 changed by software to control which interrupt request sources will be acknowledged.

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

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

Note The TRAP instruction does not change the CPU level, therefore, 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 are enabled by the interrupt enable bits in the control registers, are globally enabled.

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

6.2 Operation of the PEC channels

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

The PECC registers control the PEC channel.

PECCx (FECyh/6Zh Table 15) **SFR** **Reset Value: 0000h**

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

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

Register	Address	Reg. Space	Register	Address	Reg. Space
PECC0	FEC0h / 60h	SFR	PECC4	FEC8h / 64h	SFR
PECC1	FEC2h / 61h	SFR	PECC5	FECAh / 65h	SFR
PECC2	FEC4h / 62h	SFR	PECC6	FECCh / 66h	SFR
PECC3	FEC6h / 63h	SFR	PECC7	FECEh / 67h	SFR

Table 15 PEC control register addresses

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

Increment Control Field INC specifies whether one of the PEC pointers is incremented after the PEC transfer. It is not possible to increment both pointers. If the pointers are not modified (INC=00), the 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 PEC channel. The content of bit field COUNT at the time the request is activated, specifies the action. COUNT may specify a number of PEC transfers, unlimited transfers or no PEC service at all.

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

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

COUNT	COUNT[1]	IR after PEC service	Action of PEC Channel and Comments
FFh	FFh	‘0’	Move a Byte / Word Continuous transfer mode, i.e. COUNT is not modified
FEh..02h	FDh..01h	‘0’	Move a Byte / Word and decrement COUNT

Table 16 Dependency on COUNT of the count field, IR and PEC channel

COUNT	COUNT[1]	IR after PEC service	Action of PEC Channel and Comments
01h	00h	'1'	Move a Byte / Word Leave request flag set, which triggers another request
00h	00h	'1'	No action! Activate interrupt service routine rather than PEC channel.

Table 16 Dependency on COUNT of the count field, IR and PEC channel

1. After PEC service

Continuous transfers are selected by the value 'FFh' in bit field COUNT. COUNT is not modified and the PEC channel services any request until it is disabled again.

When COUNT is decremented from 01h to 00h after a transfer, the request flag is not cleared, which generates another request from the same source. When COUNT already contains the value 00h, the PEC channel remains idle and the associated interrupt service routine is activated instead. Therefore, you can choose whether level 15 or 14 requests are to be serviced by the PEC or by the interrupt service routine.

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

The source and destination pointers specify the locations between which the data is to be moved. A pair of pointers (SRCPx and DSTPx) is associated with each of the 8 PEC channels. These pointers do not reside in specific SFRs, but are mapped into the internal RAM of the ST10R272L just below the bit-addressable area (see figure below).

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

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

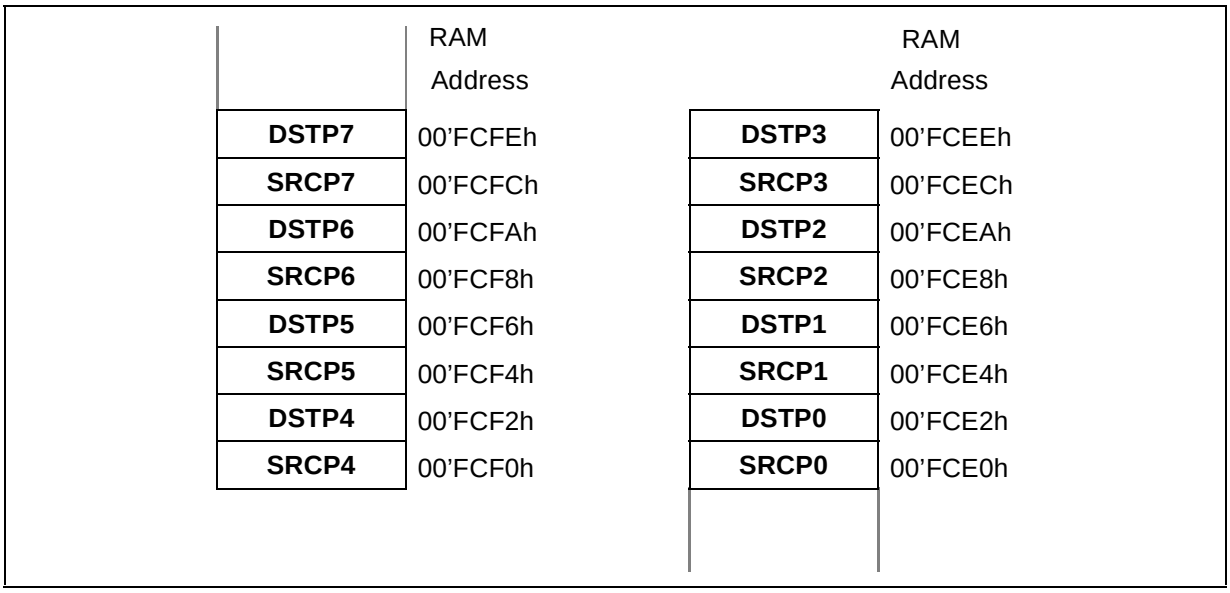


Figure 21 Mapping of PEC pointers into the internal RAM

6.3 Prioritizing interrupt and PEC service requests

Interrupt and PEC service requests from all sources can be enabled for arbitration and service (if chosen), or they may be disabled.

Interrupt requests can be enabled and disabled in three ways:

- **control bits** are used to switch each individual source 'ON' or 'OFF'. The control bits (xxIE) are located in the interrupt control registers. All interrupt requests may be enabled or disabled generally by bit IEN in the PSW register. This control bit is the 'main switch' that determines whether a selected is selected.
  Before a request can be arbitrated, its source's enable bit and the global enable bit must both be set.
- **The priority level** automatically selects a group of interrupt requests to be acknowledged, disregarding all other requests. The priority level of the source that wins the arbitration is compared against the CPU's current level. The source is only serviced if its level is higher than the current CPU level. Changing the CPU level to a specific value by software, blocks all requests on the same or lower level. An interrupt source that is assigned to level 0 will be disabled and never serviced.
- **ATOMIC and EXTend instructions** automatically disable all interrupt requests for the duration of the following 1...4 instructions. This is useful e.g. for semaphore handling and

does not require to re-enable the interrupt system after the inseparable instruction sequence.

6.4 Interrupt class management

An interrupt class covers a set of interrupt sources with the same priority. Interrupts of the same class must not interrupt each other. This is supported 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 will be blocked, i.e. no request from that 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 - the highest priority (ILVL) in class-2. Class-1 requests or PEC requests are still serviced in this case.

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

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: 8 sources on 2 levels
11	X	X	X	X	
10					
9					

Table 17 Software controlled interrupt classes (example)

ILVL (Priority)	GLVL				Interpretation
	3	2	1	0	
8	X	X	X	X	Interrupt class-2: 10 sources on 3 levels
7	X	X	X	X	
6	X	X			
5	X	X	X	X	Interrupt class-3: 6 sources on 2 levels
4	X	X			
3					
2					
1					
0					No service!

Table 17 Software controlled interrupt classes (example)

6.5 Saving the status during interrupt service

Before an arbitrated interrupt request is serviced, the status of the current task is automatically saved on the system stack. The CPU status (PSW) and the return location for the current instruction are saved. The return location is specified by the Instruction Pointer (IP) and, for a segmented memory model, the Code Segment Pointer (CSP). Bit SGTDIS in the SYSCON register 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 system stack use, 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 executes on the new level. If a multiplication or division is in progress when the interrupt request is acknowledged, bit MULIP in the PSW register 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.

The interrupt request flag of the source that is being serviced is cleared. The IP is loaded with a vector associated to 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.

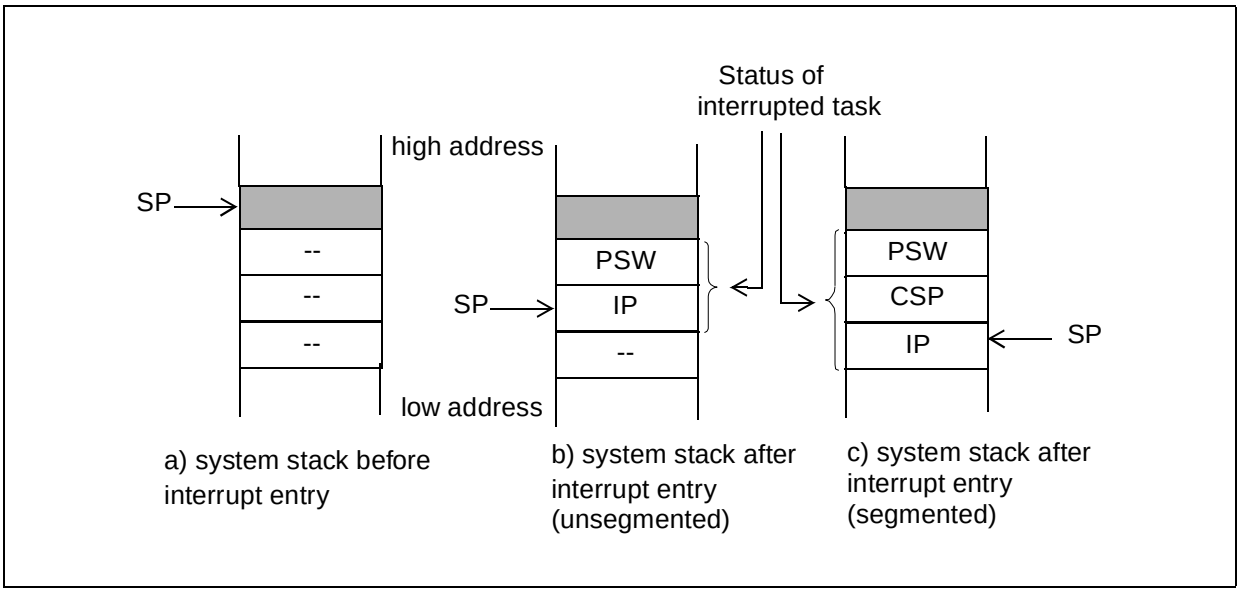


Figure 22 Task status saved on the system stack

6.5.1 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 taken by saving and restoring. The ST10R272L is able 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 can now use its ‘own registers’. This register bank is preserved when the service routine terminates, i.e. its contents are available on the next call. Before returning (RETI) the previous CP is simply POPped from the system stack, which returns the registers to the original bank.

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

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

6.6 Interrupt response times

The interrupt response time defines the time between the setting of an interrupt request flag to the first instruction fetch from the interrupt vector location (I1). The basic interrupt response time is 3 instruction cycles.

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

Cycle 1: The respective interrupt request flag is set (fetching of instruction N).

Cycle 2: The indicated source wins the prioritization round.

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: Saves PSW, IP (and CSP if segmented mode), and fetches the first instruction (I1) from the respective vector location

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

The minimum interrupt response time is 5 CPU clock cycles with,

- program execution with the fastest bus configuration (16-bit, demultiplexed, no wait states),
- no external operand read requests,
- setting the interrupt request flag during the last CPU clock cycle of an instruction.

When the interrupt request flag is set during the first CPU clock cycle of an instruction, the minimum interrupt response time is 6 CPU clock cycles.

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

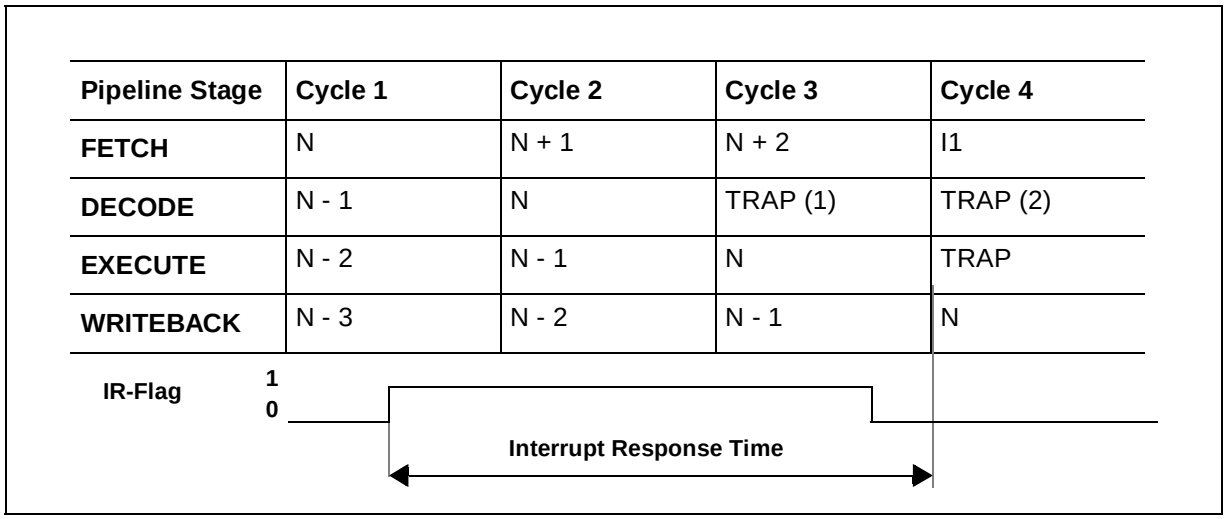


Figure 23 Pipeline diagram for interrupt response time

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 SP, the minimum interrupt response time may be extended by 1 CPU clock.

Where instruction N reads the PSW, and instruction N-1 has an effect on the condition flags, the interrupt response time may be extended by 2 CPU clock cycles.

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 instruction source and destination operands are located, there are a number of combinations. Note, however, that only access conflicts contribute to the delay. The following examples illustrate these delays:

- The worst case interrupt response time including external accesses 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 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 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.

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 in the program that was interrupted. In most cases two instructions will be executed during this time. Typically, only one instruction will be executed if the first instruction following the RETI instruction is a branch instruction (without cache hit), or if it is executed out of the internal RAM.

Note *A bus access in this context also includes delays caused by an external $\overline{READY}$ signal or by bus arbitration (HOLD mode).*

6.7 PEC response times

The PEC response time is the time between the setting of the interrupt request flag of an enabled interrupt source - to - the start of the PEC data transfer. This is 2 instruction cycles for the ST10R272L.

The PEC response pipeline has 4 cycles:

Cycle 1: The interrupt request flag is set (fetching of instruction N).

Cycle 2: A source wins the prioritization round.

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: The injected PEC transfer is completed and execution of instruction N+1 is resumed.

All instructions that enter the pipeline after the interrupt request flag (N+1, N+2) is set, are executed after the PEC data transfer.

Note *When instruction N reads any of the PEC control registers PECC7...PECC0 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 CPU clock cycles. This requires: program execution with the fastest bus configuration (16-bit, demultiplexed, no wait states) - no external operand read requests - setting the interrupt request flag during the last CPU clock cycle of an instruction.

When the interrupt request flag is set during the first CPU clock cycle of an instruction, the minimum PEC response time is 4 CPU clock 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**

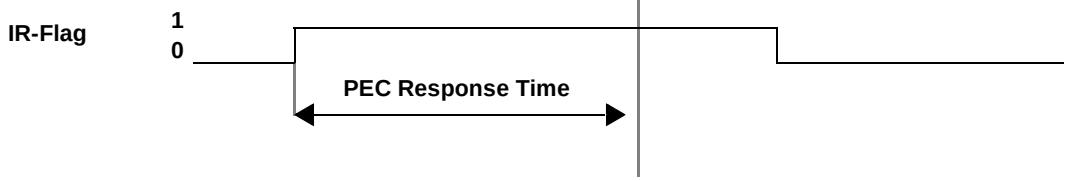


Figure 24 Pipeline diagram for PEC response time

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 CPU clock cycle for each of these conditions.
- Where 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 CPU clock cycles.

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

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

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

Once a request for PEC service has been acknowledged by the CPU, the execution of the next instruction is delayed by 2 CPU clock cycles, plus any additional time it takes to fetch the source operand from external memory and to write the destination operand over the external bus in an external program environment.

Note *A bus access in this context also includes delays caused by an external $\overline{READY}$ signal or by bus arbitration (HOLD mode).*

6.8 External interrupts

Although the ST10R272L has no dedicated INTR input pins, it can react to external asynchronous events by using the IO lines for interrupt input. The interrupt function can be combined with the pin's main function or can be used instead of it, i.e. if the main pin function is not required.

Interrupt signals may be connected to:

- T4IN, T2IN, the timer input pins
- CAPIN, the capture input of GPT2

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

Note *To use a listed pin as external interrupt input, it must be switched to input mode via its direction control bit DPx.y in the port direction control register DPx.*

Pins T2IN or T4IN can be used as external interrupt input pins when the associated auxiliary timer T2 or T4 in block GPT1 is configured for capture mode. This mode is selected by programming the mode control fields T2M or T4M in control registers T2CON or T4CON to 101b. The active edge of the external input signal is determined by bit fields T2I or T4I. When these fields are programmed to X01b, Interrupt Request Flags T2IR or T4IR in registers T2IC or T4IC will be set on a positive external transition at pins T2IN or T4IN, respectively. When T2I or T4I are programmed to X10b, then a negative external transition will set the corresponding request flag. When T2I or T4I are programmed to X11b, both a

positive and a negative transition will set the request flag. In all three cases, the contents of the core timer T3 will be captured into the auxiliary timer registers T2 or T4 based on the transition at pins T2IN or T4IN. When the interrupt enable bits T2IE or T4IE are set, a PEC request or an interrupt request for vector T2INT or T4INT is generated.

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

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

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

Port Pin	Original Function	Control Register
P3.7/T2IN	Auxiliary timer T2 input pin	T2CON
P3.5/T4IN	Auxiliary timer T4 input pin	T4CON
P3.2/CAPIN	GPT2 capture input pin	T5CON

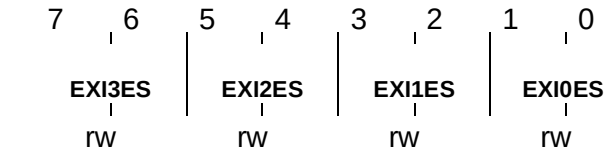
Table 18 IO pins that can be used as external interrupt inputs

6.8.1 Fast external interrupts

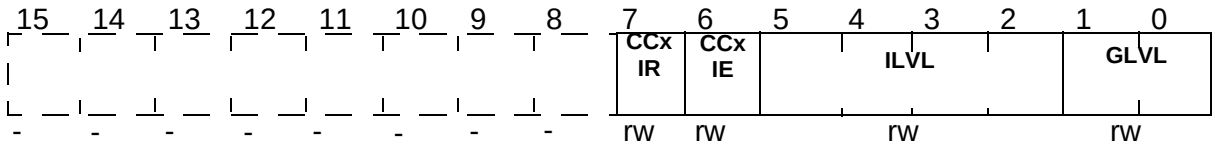
The input pins that may be used for external interrupts are sampled every 8 CPU clock cycles, i.e. external events are scanned and detected in time-frames of 8 CPU clock cycles. The ST10R272L provides 4 interrupt inputs that are sampled every CPU clock cycle, so external events are captured faster than with standard interrupt inputs.

The pins of Port 2 (EX0IN-EX3IN on P2.8-P2.11) can individually be programmed to this fast interrupt mode where the trigger transition (rising, falling or both) can be selected. The External interrupt control register EXICON controls this feature for all 4 pins.

EXICON (F1C0h / E0h) **ESFR** **Reset Value: 0000h**



CCxIC (seeTable 19) **SFR** **Reset Value: --00h**



Bit	Function
EXIxES	**External Interrupt x Edge Selection Field (x=3...0)** 0 0: Fast external interrupts disabled: standard mode 0 1: Interrupt on positive edge (rising) 1 0: Interrupt on negative edge (falling) 1 1: Interrupt on any edge (rising or falling)

Note The fast external interrupt inputs are sampled every CPU clock cycle. The interrupt request arbitration and processing, however, is executed every 4 CPU clock cycles.

Register	Address	Reg. Space
CC8IC	$FF88_H/C4_H$	SFR
CC9IC	$FF8A_H/C5_H$	SFR
CC10IC	$FF8C_H/C6_H$	SFR
CC11IC	$FF8E_H/C7_H$	SFR
CC12IC	$FF90_H/C8_H$	SFR
CC13IC	$FF92_H/C9_H$	SFR
CC14IC	$FF94_H/CA_H$	SFR
CC15IC	$FF96_H/CB_H$	SFR

Table 19 External interrupt control registers

Note *Refer to “Interrupt control registers” on page 86 for an explanation of the control fields.*

6.9 Traps

6.9.1 Software traps

Software Traps can be performed from any vector location between 00’0000h and 00’01FCh. A service routine entered via a software TRAP instruction, is always executed on the current CPU priority level (indicated in bit field ILVL in register PSW). This means that routines entered by the software TRAP instruction can be interrupted by all Hardware Traps or higher level interrupt requests.

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

Note *The CPU level in the PSW register 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.*

6.9.2 Hardware traps

Exceptions or error conditions that arise during run-time are called Hardware Traps. Hardware Traps cause immediate non-maskable system reaction, similar to a standard interrupt service (branching to a dedicated vector table location). The occurrence of a Hardware Trap is additionally signified by an individual bit in the trap flag register (TFR). Except when another higher prioritized trap service is in progress, a Hardware Trap will interrupt any actual program execution. In turn, Hardware Trap services can not normally be interrupted by standard or PEC interrupts. The following table shows all of the possible exceptions or error conditions that can arise during run-time:

Exception Condition	Trap Flag	Trap Vector	Vector Location	Trap Number	Trap Priority
Reset Functions:					

Table 20 Exceptions or error conditions

Exception Condition	Trap Flag	Trap Vector	Vector Location	Trap Number	Trap Priority
Hardware Reset		RESET	00'0000h	00h	III
Software Reset		RESET	00'0000h	00h	III
Watchdog Timer Overflow		RESET	00'0000h	00h	III
class-A Hardware Traps:					
Non-Maskable Interrupt	NMI	NMITRAP	00'0008h	02h	II
Stack Overflow	STKOF	STOTRAP	00'0010h	04h	II
Stack Underflow	STKUF	STUTRAP	00'0018h	06h	II
class-B Hardware Traps:					
Undefined opcode	UNDOPC	BTRAP	00'0028h	0Ah	I
Protected instruction fault	PRTFLT	BTRAP	00'0028h	0Ah	I
Illegal word operand access	ILLOPA	BTRAP	00'0028h	0Ah	I
Illegal instruction access	ILLINA	BTRAP	00'0028h	0Ah	I
Illegal external bus access	ILLBUS	BTRAP	00'0028h	0Ah	I
MAC trap	MACTRP	BTRAP	00'0028h	0Ah	I
Reserved			[2Ch – 3Ch]	[0Bh – 0Fh]	
Software Traps					
TRAP Instruction			Any [00'0000h – 00'01FCh] steps of 4h	Any [00h – 7Fh]	Current CPU Priority

Table 20 Exceptions or error conditions

6.9.3 Trap flag register

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 (FFACh / D6h) **SFR** **Reset Value: 0000h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NMI	STK OF	STK UF	-	-	-	-	-	UND OPC	MAC TFR	-	-	PRT FLT	ILL OPA	ILL INA	ILL BUS
rw	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 ST10R272L 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 occurs 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 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.

6.9.4 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 enters 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.

6.9.5 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 enters the stack overflow trap routine. The IP value 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 on interrupt or trap entry, the IP value is the address of the following instruction. When the SP is decremented by a subtract instruction, the IP value represents the address of the instruction after the instruction following the subtract instruction.

For recovery from stack overflow, ensure that there is enough excess space on the stack to save the current system state twice (PSW, IP, in segmented mode also CSP). Otherwise, a system reset will be generated.

6.9.6 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. The IP value 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 is the address of the following

instruction. When the SP is incremented by an add instruction, the IP value represents the address of the instruction after the instruction following the add instruction.

6.9.7 Undefined opcode trap

When the instruction currently decoded by the CPU does not contain a valid ST10R272L 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 non-implemented opcodes based on the stacked IP. To resume processing, the stacked IP value must be incremented by the size of the undefined instruction (determined by the user), before a RETI instruction is executed.

6.9.8 Protection fault trap

When a protected instruction is executed without its opcode being 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 the TFR register 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.

6.9.9 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.

6.9.10 Illegal instruction access trap

Whenever a branch is made to an odd byte address, the ILLINA flag in the TFR register 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.

6.9.11 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. However, because the ST10R262 is a romless microcontroller, an external bus must always be defined and this trap should never occur.

6.9.12 MAC interrupt on condition

User defined in the MCW register.

7 PARALLEL PORTS

The ST10R272L provides up to 77 parallel I/O lines organized into six 8-bit I/O ports (POH, POL, P1H, P1L, P4, P6), two 4-bit I/O ports (P2, P7), one 15 bit I/O port (P3) and one 6-bit input port (P5). The port lines can be used for general purpose input/output, controlled by software, or may be used implicitly by the integrated peripherals or the external bus controller.

For low power devices:

- Port 5 pins are 5 V tolerant and fail-safe (voltage max. with respect to Vss is -0.5 to 5.5, even if the chip is non powered)
- XTAL1 and XTAL2 are 3 V tolerant (voltage max. respect to Vss is -0.5 to V_{DD} + 0.5)
- All other pins are 5 V tolerant (voltage max. respect to Vss is -0.5 to 5.5 only if chip is powered)

For more information on this, refer to the ST10R272L Data Sheet.

Using the port as a general purpose I/O line

All port lines are bit addressable, and all input/output lines are individually (bit-wise) programmable as inputs or outputs via direction registers (except Port 5). 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 (2, 3, 6) can be configured (pin by pin) for push/pull operation or open-drain operation via control registers. The logic level of a pin is clocked into the input latch once per CPU clock cycle, regardless whether the port is configured for input or output.

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

Writing to a pin configured as an output (DPx.y=‘1’) causes the output latch and the pin to have the written value, since the output buffer is enabled. Reading this pin returns the value

of the output latch. A read-modify-write operation reads the value of the output latch, modifies it, and writes it back to the output latch, thus also modifying the level at the pin.

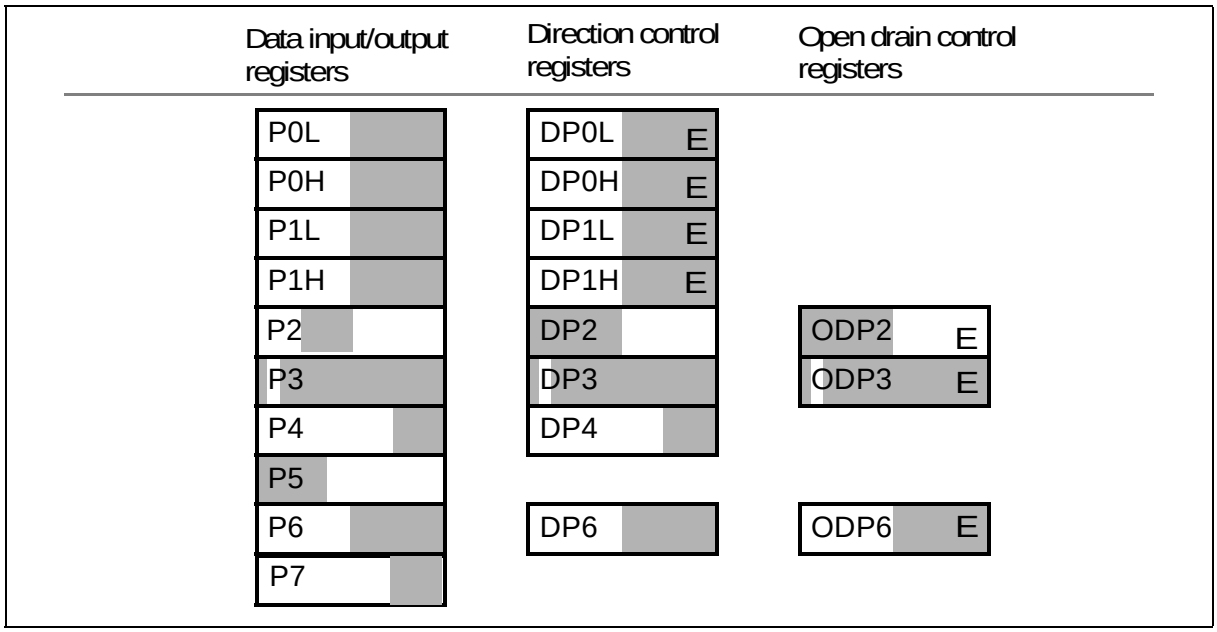


Figure 25 SFRs and pins associated with the parallel ports

Note E denotes an ESFR located in the ESFR space

In the ST10R272L 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 ports P2, P3 and P6 (see respective sections), 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.

- Each port line has one programmable alternate input or output function associated with it. 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 interface, the optional bus control signal $\overline{BHE}$ and the system clock output (CLKOUT).
- Port 4 outputs the additional segment address bits A23/19/17...A16 in systems where more than 64 KBytes of memory are to be accessed directly.
- Port 5 is used for timer control signals.

Port 6 provides the optional chip select outputs and the bus arbitration lines.

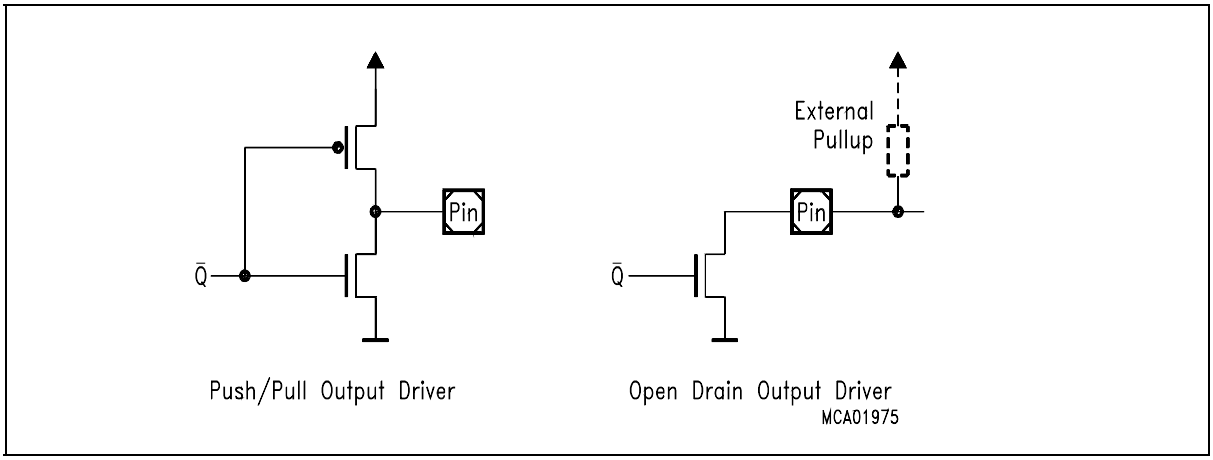


Figure 26 Output drivers in push/pull and open drain mode

Alternate input or output function of port

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 affected by the alternate output function. The respective port latch should hold a '1', because its output is ANDed with the alternate output data.

If an alternate input function of a pin is used, the direction of the pin must be programmed for input (DPx.y='0') if an external device is driving the pin. The input direction is the default after reset. If no external device is connected to the pin, one can also set the direction for this pin to output. In this case, the pin reflects the state of the port output latch. Thus, the alternate input function reads the value stored in the port output latch. This can be used for testing

purposes to allow a software trigger of an alternate input function by writing to the port output latch.

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

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

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

```
SINGLE_BIT:     BSET        P4.7
                ; Initial output level is “high”
                BSET        DP4.7
                ; Switch on the output driver
BIT_GROUP:      BFLDH       P4, #24H, #24H
                ; Initial output level is “high”
                BFLDH       DP4, #24H, #24H
                ; Switch on the output drivers
```

Note *When using several BSET pairs to control more pins of one port, these pairs must be separated by instructions, which do not reference the respective port (refer to “Pipeline effects” on page 41).*

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

7.1 Port 0

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

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

P0L (FF00h / 80h) **SFR** **Reset Value: - - 00h**

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 (FF02h / 81h) **SFR** **Reset Value: - - 00h**

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 (F100h / 80h) **ESFR** **Reset Value: - - 00h**

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 (F102h / 81h) **ESFR** **Reset Value: - - 00h**

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

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

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

outputting the address. During external accesses in demultiplexed bus modes PORT0 reads the incoming instruction or data word or outputs the data byte or word.

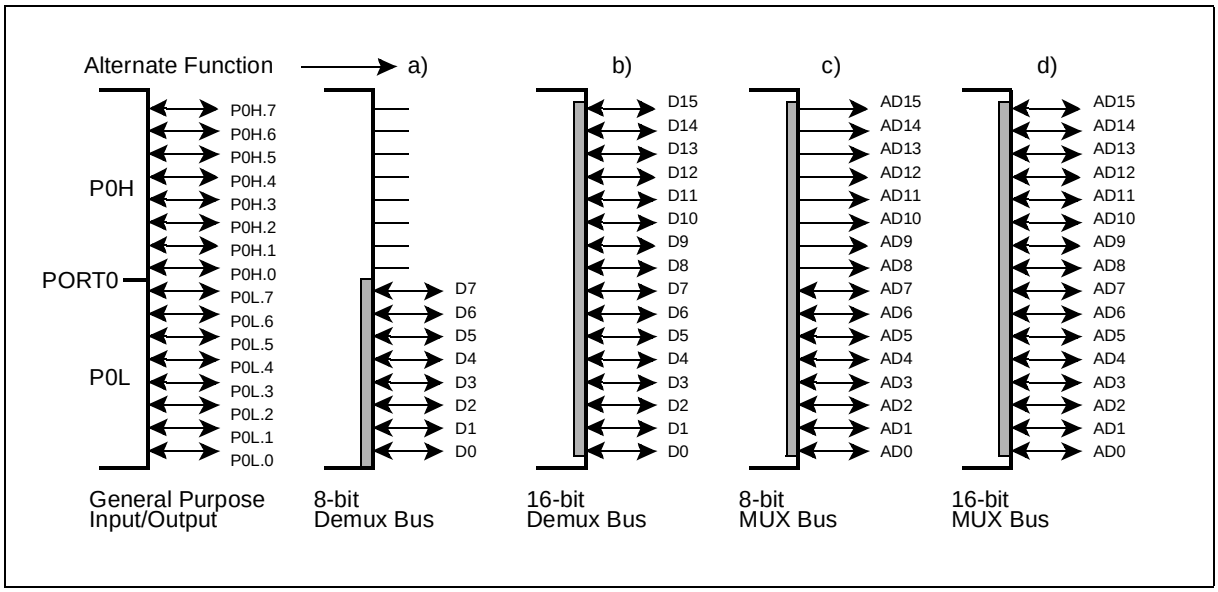


Figure 27 PORT0 I/O and alternate functions

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.

The figure below shows the structure of a PORT0 pin.

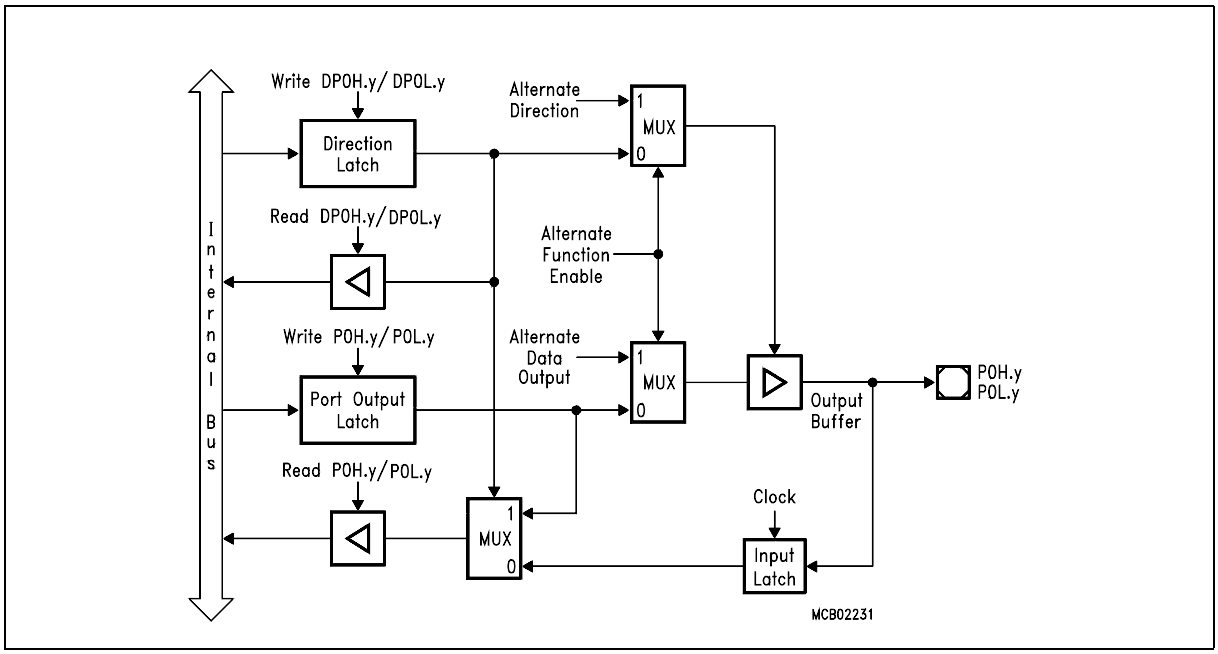


Figure 28 Block diagram of a PORT0 pin

7.2 Port 1

The two 8-bit ports P1H and P1L represent the higher and lower part of PORT1, respectively. Both halves of PORT1 can be written (e.g. via a PEC transfer) without affecting

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

P1L (FF04h / 82h) **SFR** **Reset Value: - - 00h**

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 (FF06h/ 83h) **SFR** **Reset Value: - - 00h**

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 (F104h / 82h) **ESFR** **Reset Value: - - 00h**

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 (F106h / 83h) **ESFR** **Reset Value: - - 00h**

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

7.2.1 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.

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

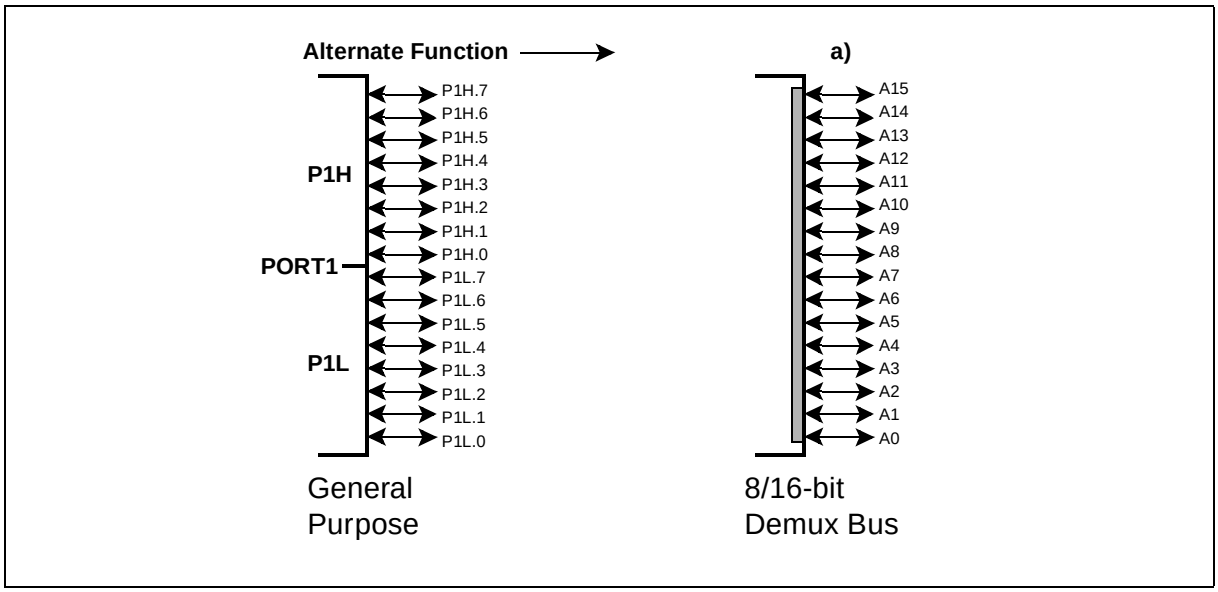


Figure 29 PORT 1 I/O and alternate functions

The figure below shows the structure of a PORT1 pin.

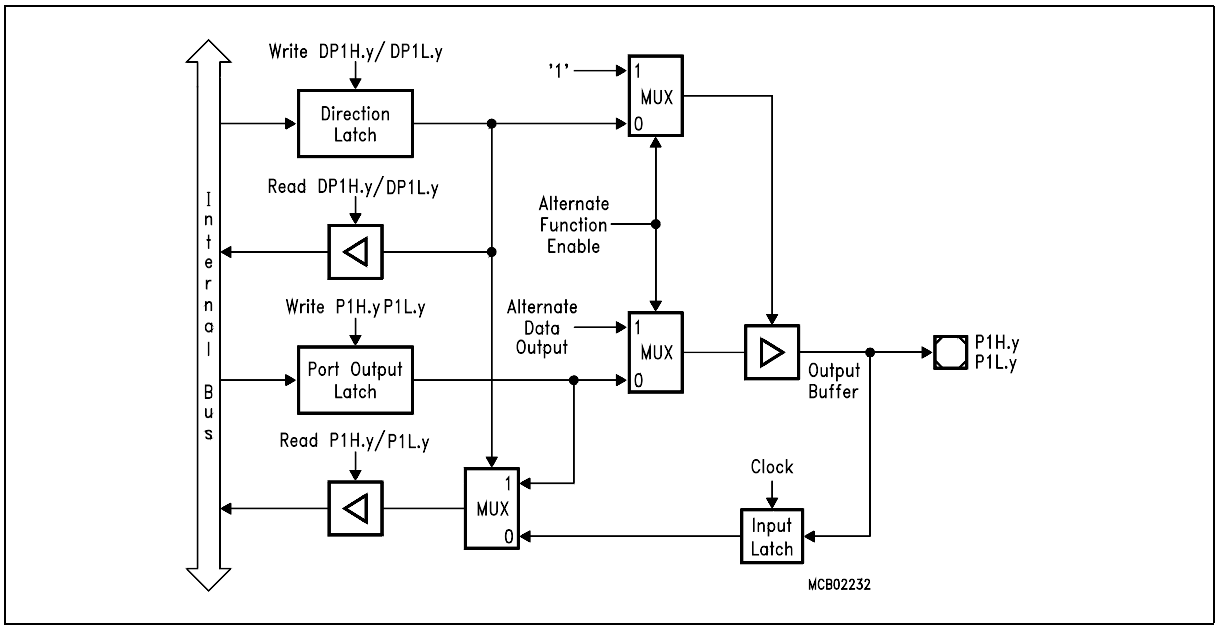


Figure 30 Block diagram of a PORT1 pin

7.3 Port 2

Port 2 is a 4-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.

7.3.1 Alternate functions of port 2

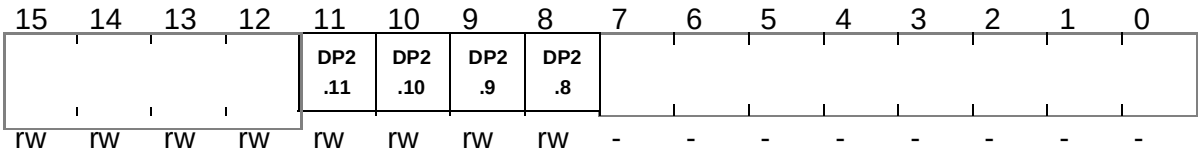
Port 2 lines P2.11...P2.8 can serve as Fast External Interrupt inputs (EX3IN...EX0IN).

P2 (FFC0h / E0h) **SFR** **Reset Value: 00 - -h**

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

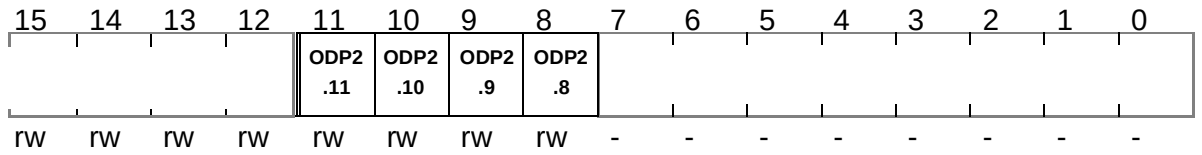
Bit	Function
P2.y	Port data register P2 bit y

DP2 (FFC2h/ E1h) **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 (F1C2h / E1h) **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

The table below summarizes the alternate functions of Port 2. .

Port 2 Pin	Alternate Function	
P2.8	EX0IN	Fast External Interrupt 0 Input
P2.9	EX1IN	Fast External Interrupt 1 Input
P2.10	EX2IN	Fast External Interrupt 2 Input
P2.11	EX3IN	Fast External Interrupt 3 Input

Table 21 Port 2 functions

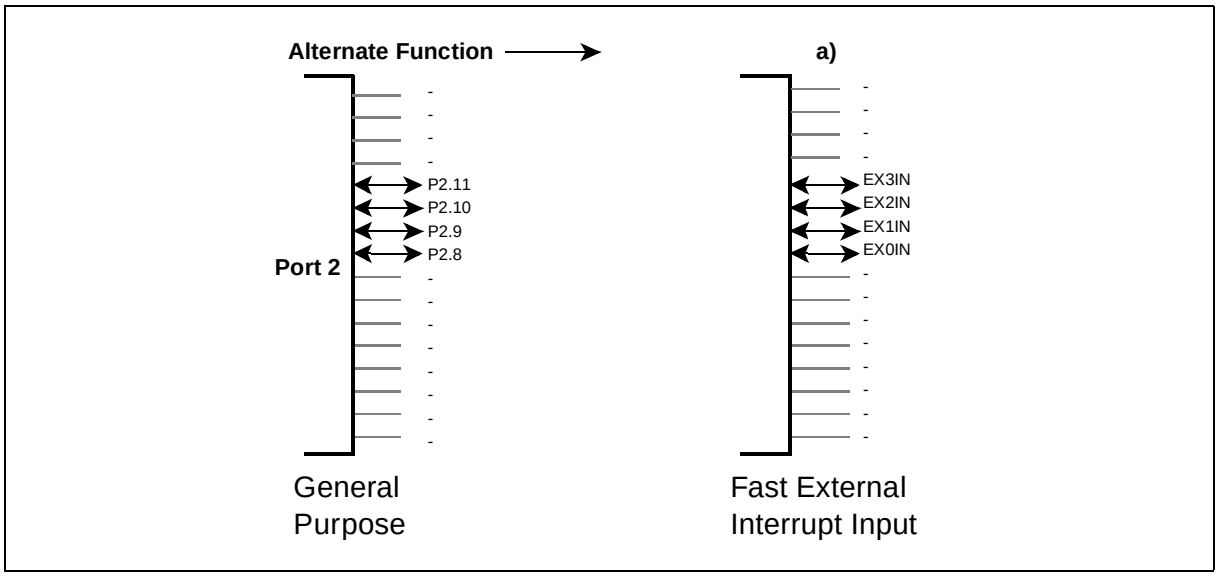


Figure 31 Port 2 I/O and alternate functions

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

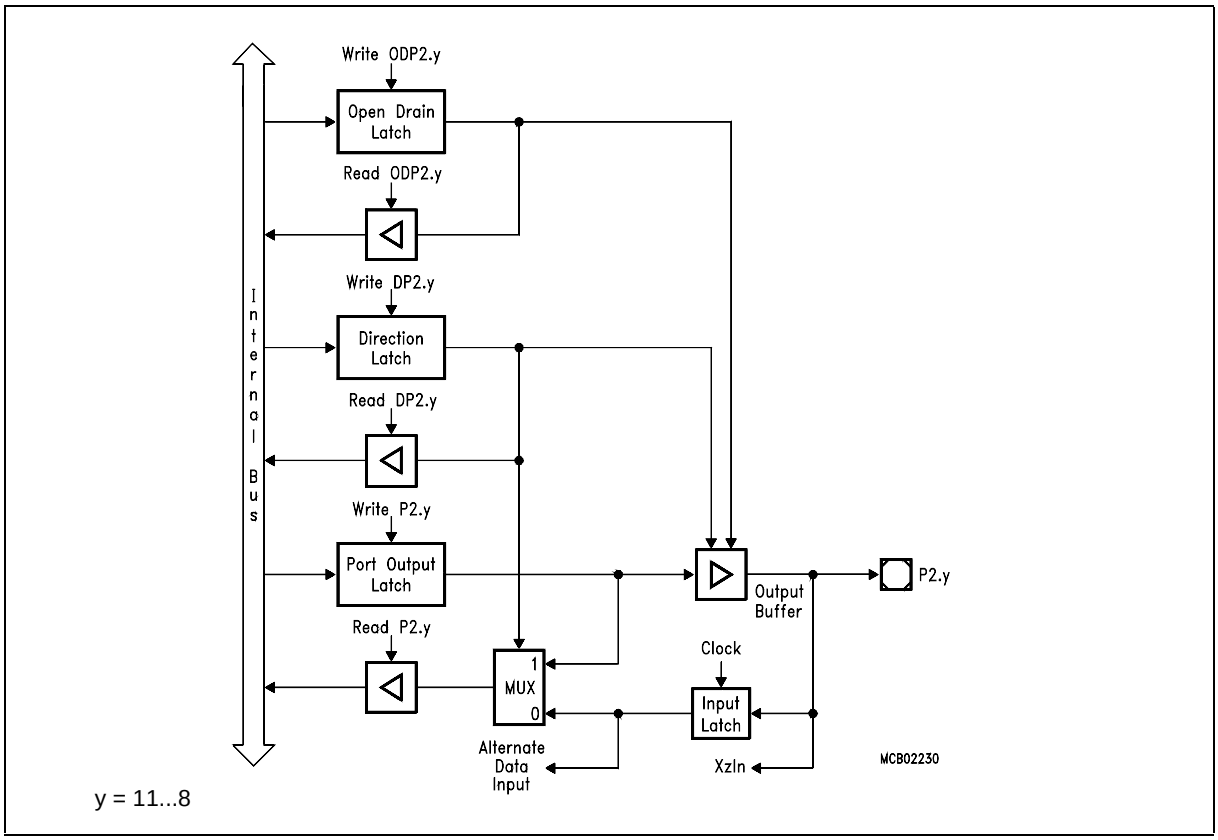


Figure 32 Block diagram of a port 2 pin

7.4 Port 3

If this 15-bit port is used for general purpose I/O, the direction of each line can be configured via the corresponding direction register DP3. Most port lines can be switched into push/pull or open drain mode via the open drain control register ODP3 (pins P3.15, P3.14 and P3.12 do not support open drain mode!).

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

P3 (FFC4h / E2h) **SFR** **Reset Value: 0000h**

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

Bit	Function
P3.y	Port data register P3 bit y

Note Register bit P3.14 is not connected to an I/O pin.

DP3 (FFC6h / E3h) **SFR** **Reset Value: 0000h**

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

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

ODP3 (F1C6h / E3h) **ESFR** **Reset Value: 0000h**

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

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

7.4.1 Alternate functions of Port 3

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.

The table below summarizes the alternate functions of Port 3. .

Port 3 Pin	Alternate Function	
P3.0	-	
P3.1	T6OUT	Timer 6 Toggle Output
P3.2	CAPIN	GPT2 Capture Input
P3.3	T3OUT	Timer 3 Toggle Output
P3.4	T3EUD	Timer 3 External Up/Down Control Input
P3.5	T4IN	Timer 4 Count Input
P3.6	T3IN	Timer 3 Count Input
P3.7	T2IN	Timer 2 Count Input
P3.10	TxD0	ASC0 Transmit Data Output
P3.11	RxD0	ASC0 Receive Data Input
P3.12	$\overline{BHE}$/$\overline{WRH}$	Byte High Enable / Write High Output
P3.14	---	No pin assigned!
P3.15	CLKOUT	System Clock Output

Figure 33 Port 3 alternate functions

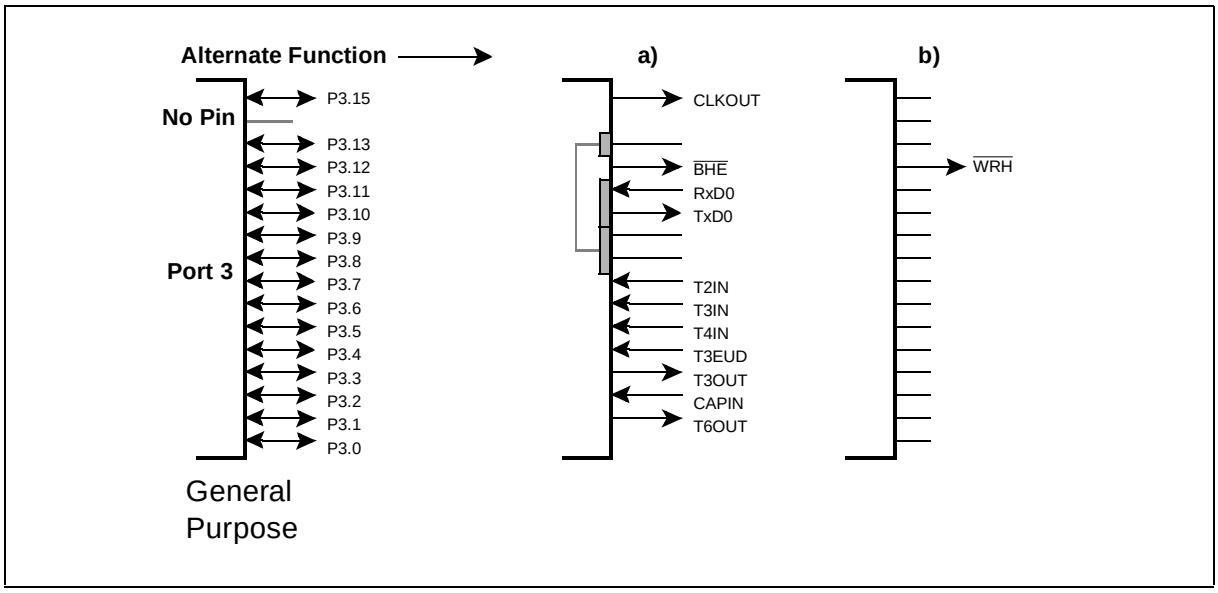


Figure 34 Port 3 I/O and alternate functions

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

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

When the on-chip peripheral associated with a Port 3 pin is configured to use the alternate output function, its “Alternate Data Output” line is ANDed with the port output latch line. When using these alternate functions, the user must set the direction of the port line to output (DP3.y=1) and must set the port output latch (P3.y=1). Otherwise the pin is in its high-impedance state (when configured as input) or the pin is stuck at '0' (when the port output latch is cleared). When the alternate output functions are not used, the “Alternate Data Output” line is in its inactive state, which is a high level ('1'). Port 3 pins with alternate output functions are:
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 pin with alternate input/output functions is: RxD0.

Enabling the CLKOUT function automatically enables the P3.15 output driver. Setting bit DP3.15='1' is not required.

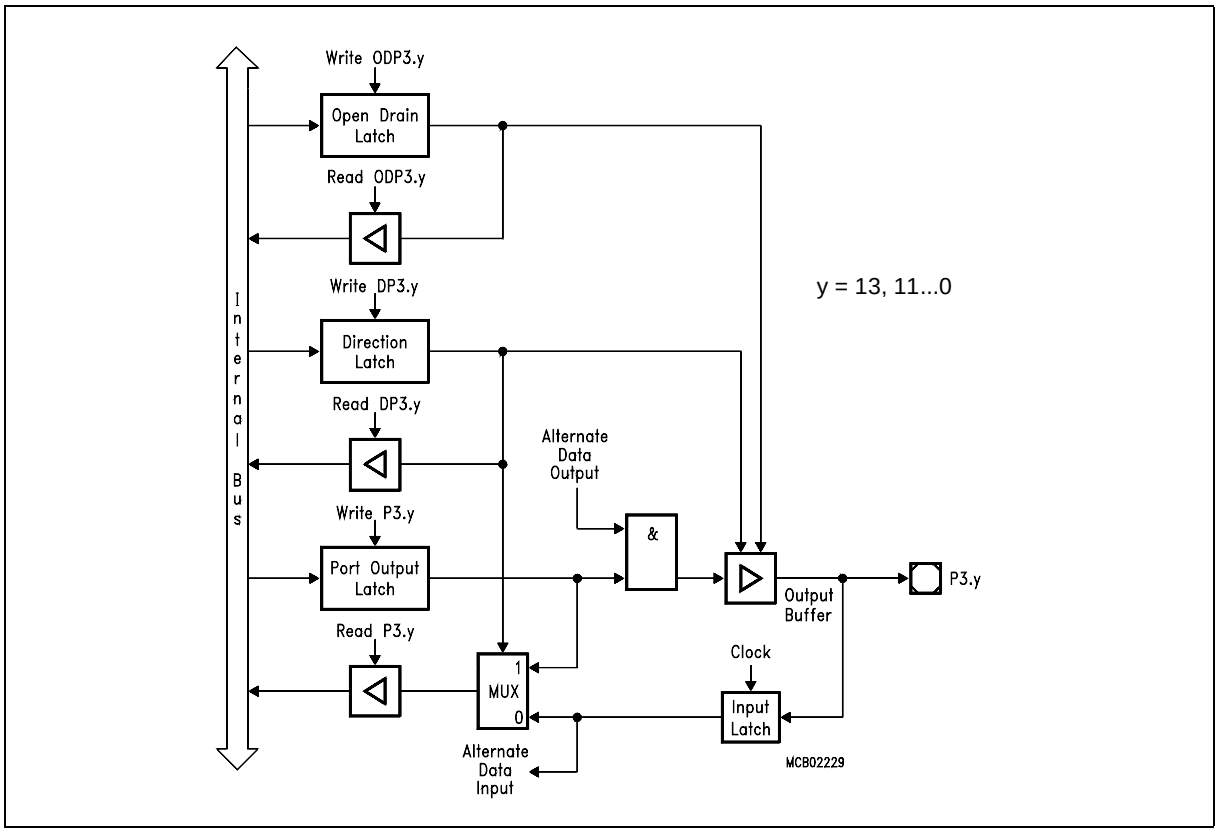


Figure 35 Block diagram of port 3 pin with alternate input or alternate output function

Pin P3.12 ($\overline{BHE}$/$\overline{WRH}$) is one more pin with an alternate output function. However, its structure is slightly different (see figure below), because after reset the $\overline{BHE}$ or $\overline{WRH}$ function must be used depending on the system start-up configuration. In these cases there is no possibility to program any port latches before. Thus the appropriate alternate function

is selected automatically. If $\overline{BHE}$/$\overline{WRH}$ is not used in the system, this pin can be used for general purpose I/O by disabling the alternate function (BYTDIS = '1' / WRCFG='0').

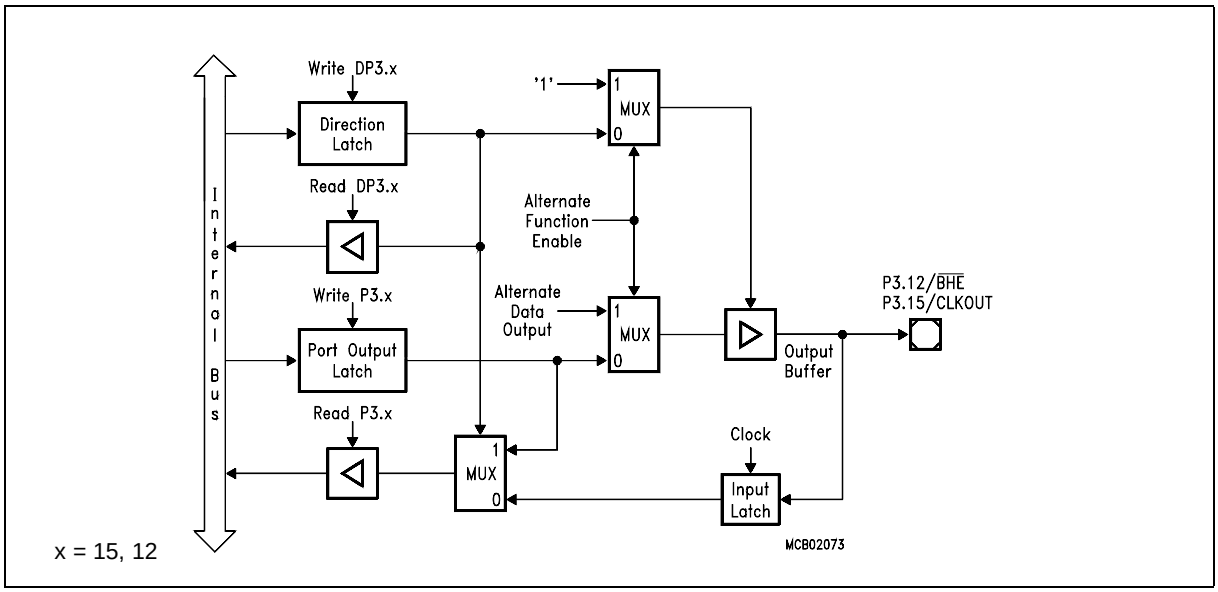


Figure 36 Block diagram of pins P3.15 (CLKOUT) and P3.12 ($\overline{BHE}$/$\overline{WRH}$)

7.5 Port 4

In the ST10R272L, the Port 4 is an 8-bit port. If the SSP is disabled (bit SSPEN cleared in SYSCON register), Port 4 is used for general purpose I/O (the direction of each line can be configured via the corresponding direction register DP4).
If the SSP is enabled (bit SSPEN set), the 4 upper pins are used for the Synchronous Serial Port (SSP) dedicated I/O. The bits in the Port 4 Data Register (P4) and the Port 4 Direction

Control Register (DP4) that correspond to the pins SSPCLK, SSPDATA, SSPDEN0 and SSPDEN1 are no influence on these pins.

P4 (FFC8h / E4h) **SFR** **Reset Value: - - 00h**

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

Bit	Function
P4.y	Port data register P4 bit y

DP4 (FFCAh / E5h) **SFR** **Reset Value: - - 00h**

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

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

7.5.1 Alternate functions of Port 4

During external bus cycles that use segmentation (i.e. an address space above 64 KByte) a number of Port 4 pins may output the segment address lines. The number of pins that is used for segment address output determines the external address space which is directly accessible. The other pins of Port 4 (if any) may be used for general purpose I/O. If segment address lines are selected, the alternate function of Port 4 may be necessary to access e.g. external memory directly after reset. For this reason Port 4 will be switched to 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) e.g. in order to check the configuration during run time.

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

Port 4 Pin	Std. Function SALSEL=01 64 KB	Altern. Function SALSEL=11 256KB	Altern. Function SALSEL=00 1 MB	Altern. Function SALSEL=10 16 MB
P4.0	Gen. purpose 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. I/O / SSPCE1	Gen. I/O / SSPCE1	Gen. I/O / SSPCE1	Seg. Address A20
P4.5	Gen. I/O / SSPCE0	Gen. I/O / SSPCE0	Gen. I/O / SSPCE0	Seg. Address A21
P4.6	Gen. I/O / SSPDAT	Gen. I/O / SSPDAT	Gen. I/O / SSPDAT	Seg. Address A22
P4.7	Gen. I/O / SSPCLK	Gen. I/O / SSPCLK	Gen. I/O / SSPCLK	Seg. Address A23

Figure 37 Port 4 alternate functions

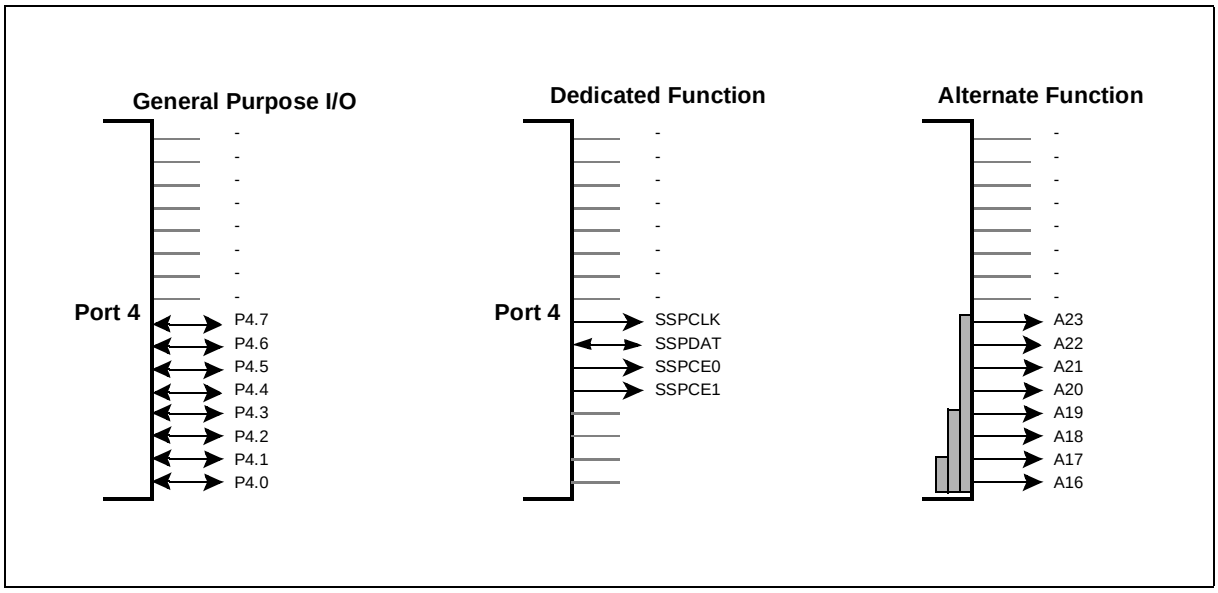


Figure 38 Port 4 I/O and alternate functions

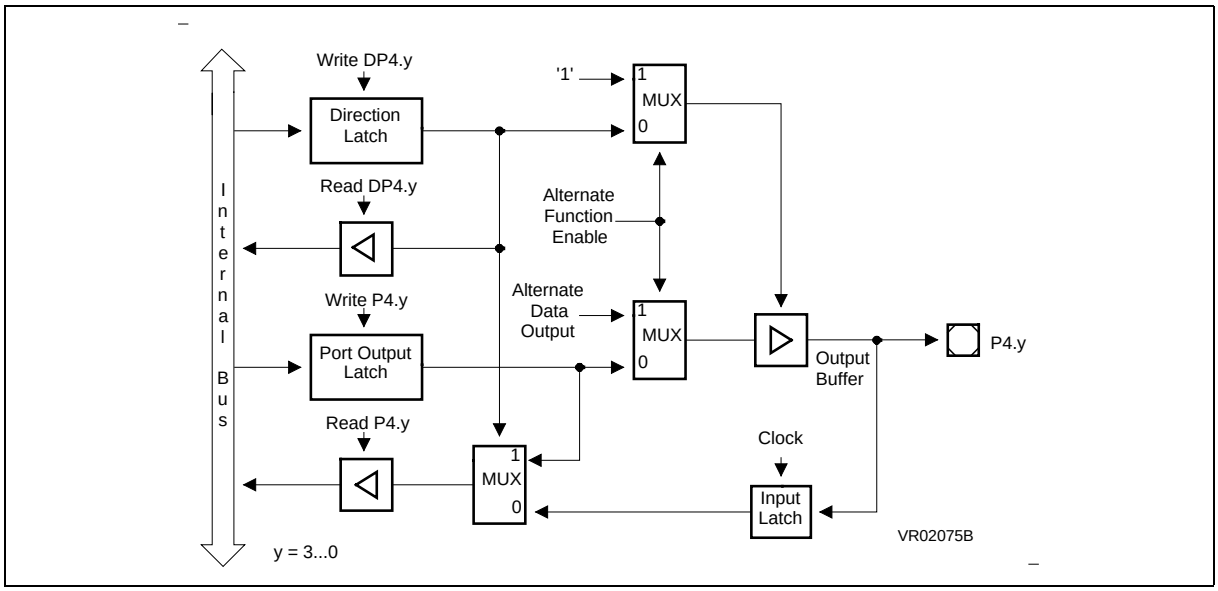


Figure 39 Block diagram of port 4 pin P4.0 to P4.3

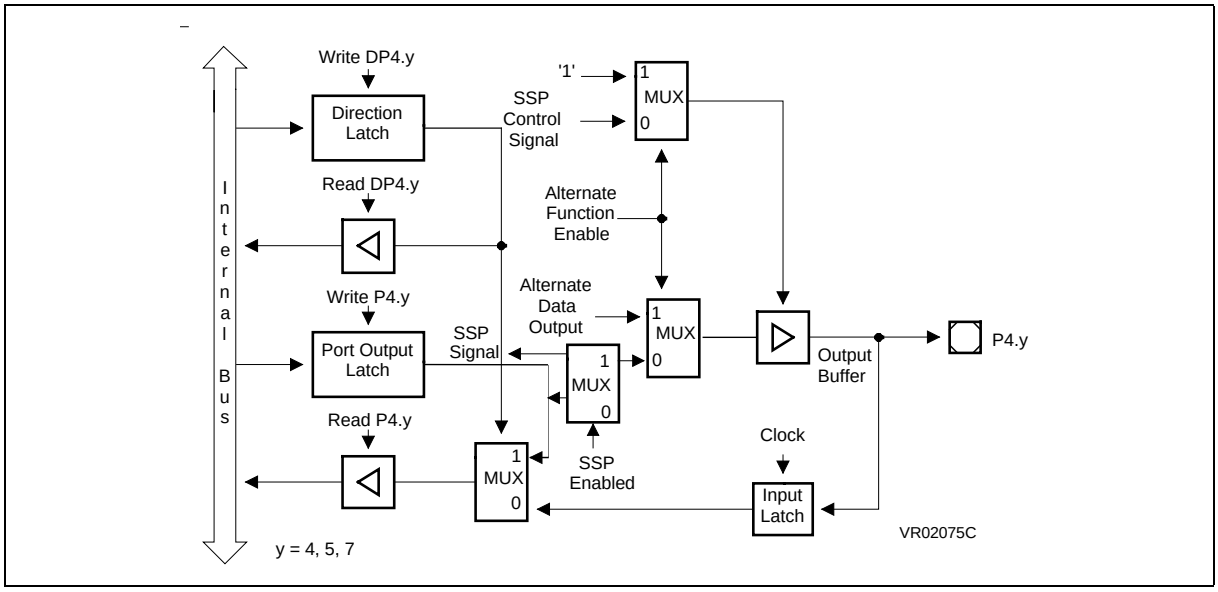


Figure 40 Block diagram of port 4 pins SSPCE1/ A20, SSPCE0 / A21, SSPCLK / A2

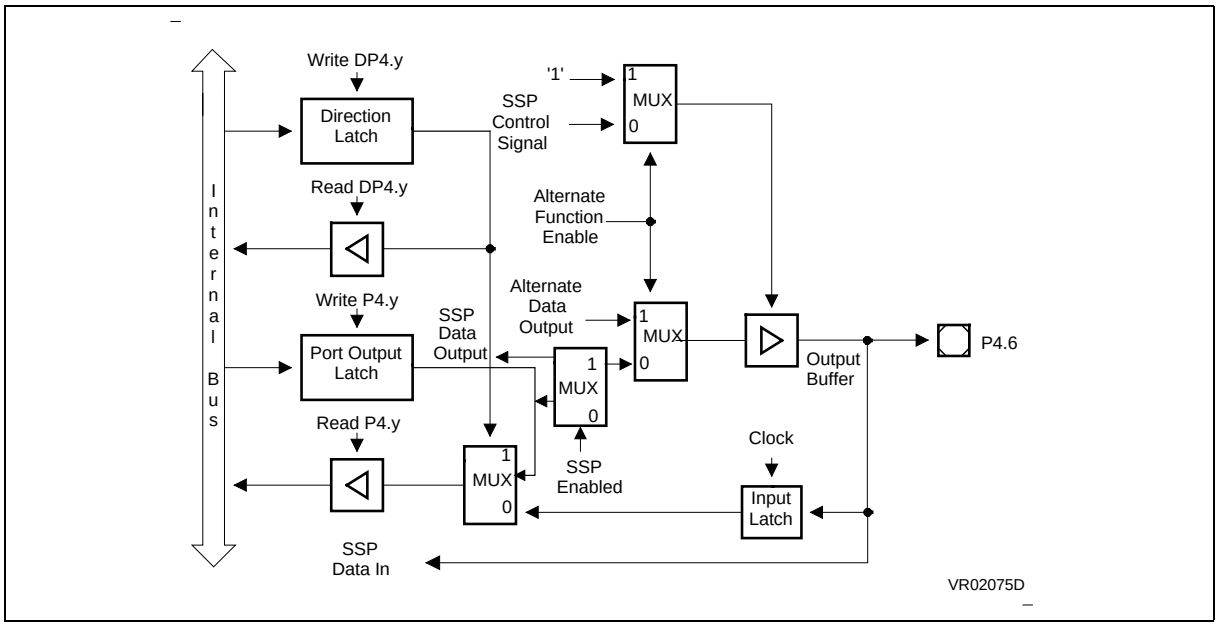


Figure 41 Block diagram of port 4 pin SSPDAT/ A22

7.6 Port 5

Port 5 pins are 5 V tolerant and fail-safe (voltage max. with respect to Vss is -0.5 to 5.5, even if the chip is non powered). For more information on this refer to the ST10R272L Data Sheet.

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

P5 (FFA2h / D1h) **SFR** **Reset Value: XX - -h**

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

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

7.6.1 Alternate functions of port 5

Each line of Port 5 serves as external timer control line for GPT1 and GPT2. The table below summarizes the alternate functions of Port 5. .

Port 5 Pin	Alternate Function
P5.10	T6EUDTimer 6 external Up/Down Control Input
P5.11	T5EUDTimer 5 external Up/Down Control Input
P5.12	T6INTimer 6 Count Input
P5.13	T5INTimer 5 Count Input
P5.14	T4EUDTimer 4 external Up/Down Control Input
P5.15	T2EUDTimer 2 external Up/Down Control Input

Table 22 Port 5 alternate functions

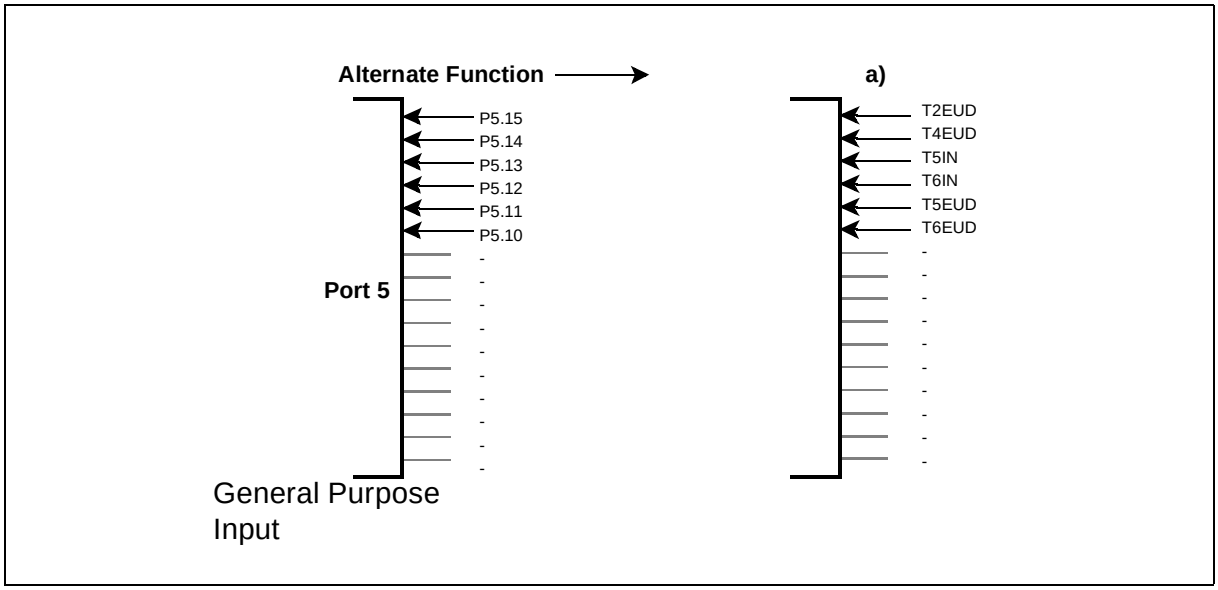


Figure 42 Port 5 I/O and alternate functions

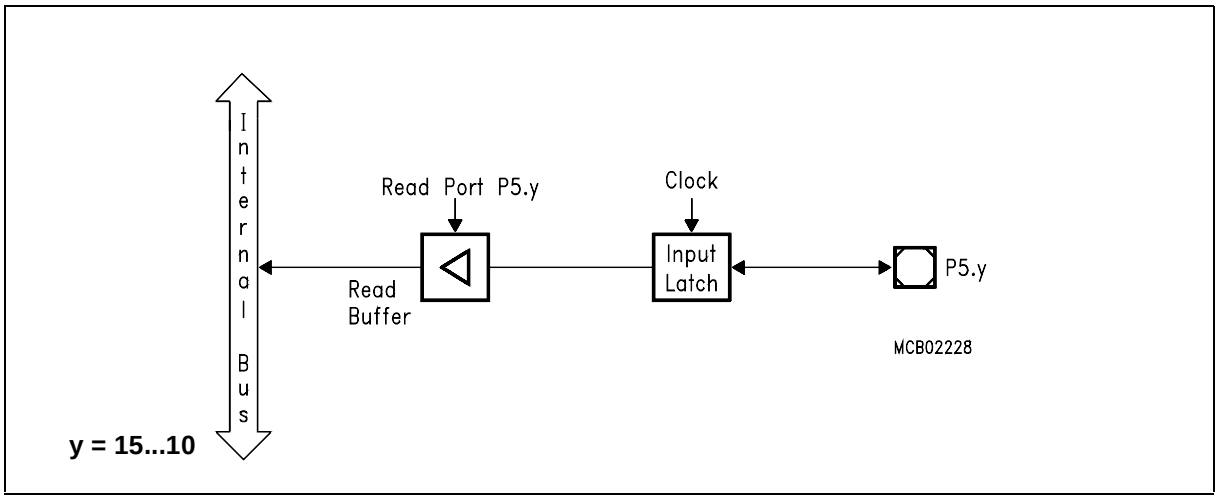


Figure 43 Block diagram of a port 5 pin

7.7 Port 6

Port 6 is an 8-bit port that can be 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.

P6 (FFCCh / E6h) **SFR** **Reset Value: - - 00h**

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

Bit	Function
P6.y	Port data register P6 bit y

DP6 (FFCEh / E7h) **SFR** **Reset Value: - - 00h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								DP6.7	DP6.6	DP6.5	DP6.4	DP6.3	DP6.2	DP6.1	DP6.0
-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw	rw	rw	rw

Bit	Function
DP6.y	**Port direction register DP6 bit y** DP6.y = 0: Port line P6.y is an input (high-impedance) DP6.y = 1: Port line P6.y is an output

ODP6 (F1CEh / E7h) **ESFR** **Reset Value: - - 00h**

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

7.7.1 Alternate functions of Port 6

A programmable number of chip select signals (CS4...CS0) derived from the bus control registers (BUSCON4...BUSCON0) can be output on 5 pins of Port 6. The other 3 pins may be used for bus arbitration to accommodate additional masters in a ST10R272L 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) e.g. in order to check the configuration during run time.

The table below summarizes the alternate functions of Port 6, as a function of the number of selected chip select lines (coded via bitfield CSSEL).

Port 6 Pin	Altern. Function CSSEL = 10	Altern. Function CSSEL = 01	Altern. Function CSSEL = 00	Altern. Function CSSEL = 11
P6.0	Gen. purpose 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 $\overline{CS3}$
P6.4	Gen. purpose I/O	Gen. purpose I/O	Gen. purpose I/O	Chip select $\overline{CS4}$
P6.5	$\overline{HOLD}$: External hold request input			
P6.6	$\overline{HLDA}$: Hold acknowledge output			
P6.7	$\overline{BREQ}$: Bus request output			

Figure 44 Port 6 alternate functions

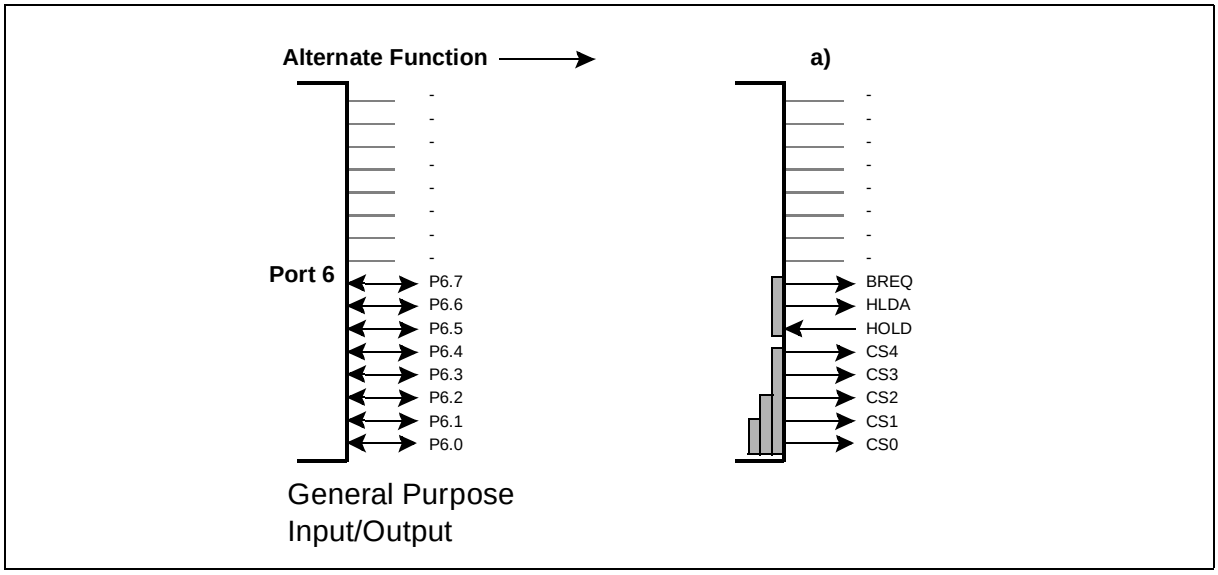


Figure 45 Port 6 I/O and alternate function

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 ST10R272L 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 ST10R272L 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 CPU clock cycle, 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.

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.

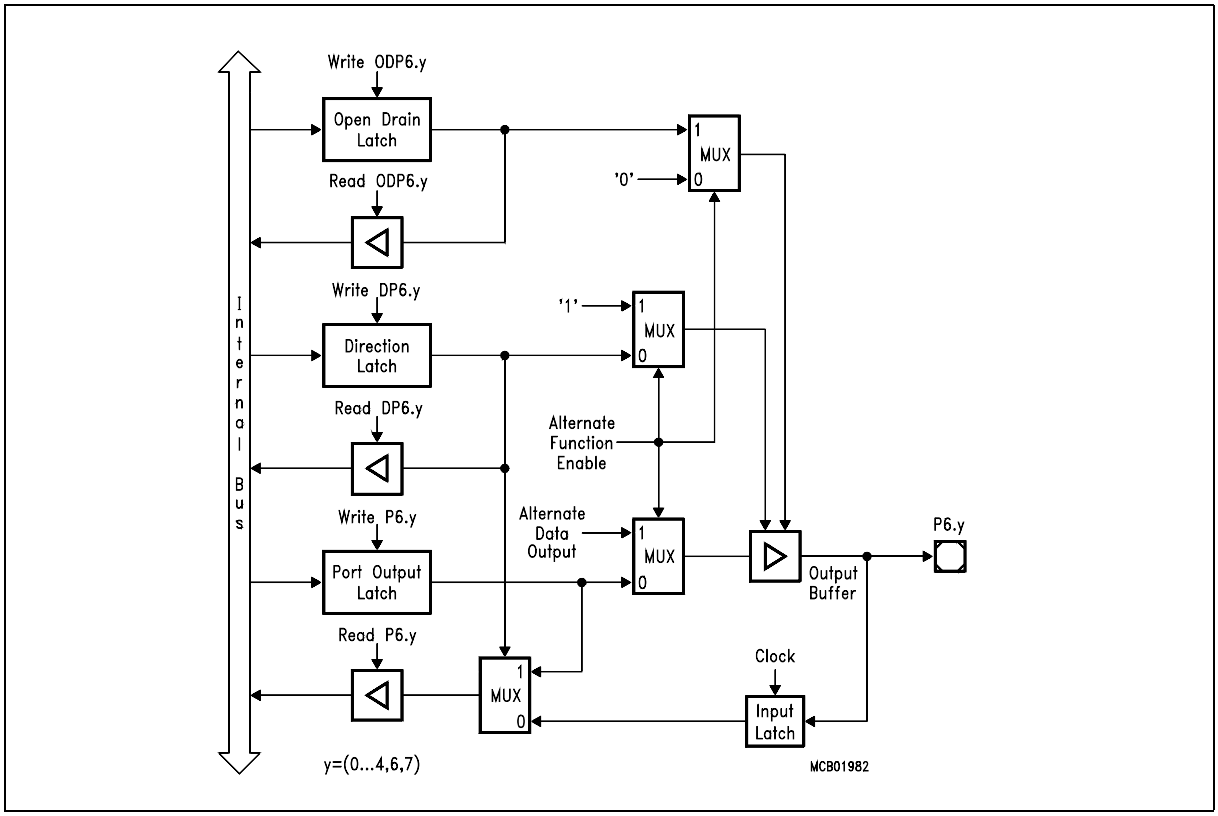


Figure 46 Block diagram of port 6 pins with an alternate output function

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.

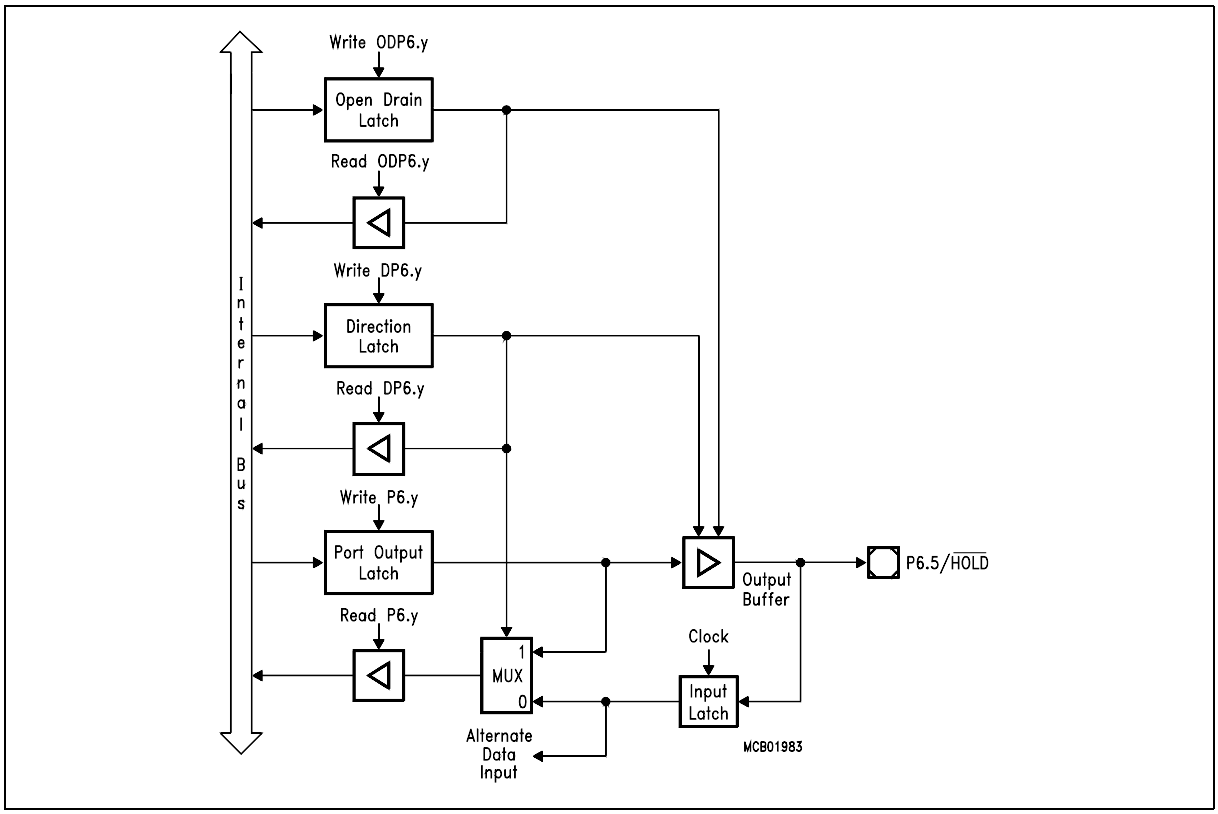


Figure 47 Block diagram of pin P6.5 (HOLD)

7.8 Port 7

Port7 is a 4-bit port. If Port 7 is used for general purpose I/O, the direction of each line can be configured via the corresponding direction register DP7. Each port line can be switched into push/pull or open drain mode via the open drain control register ODP7.

P7 (FFD0h / E8h) **SFR** **Reset Value: - - -0h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								-	-	-	-	P7.3	P7.2	P7.1	P7.0
-	-	-	-	-	-	-	-	-	-	-	-	rw	rw	rw	rw

Bit	Function
P7.y	Port data register P7 bit y

DP7 (FFD2h / E9h) **SFR** **Reset Value: - - -0h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								-	-	-	-	DP7.3	DP7.2	DP7.1	DP7.0

Bit	Function
DP7.y	**Port direction register DP7 bit y** DP7.y = 0: Port line P7.y is an input (high-impedance) DP7.y = 1: Port line P7.y is an output

ODP7 (F1D2h / E9h) **ESFR** **Reset Value: - - -0h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								-	-	-	-	ODP7.3	ODP7.2	ODP7.1	ODP7.0
-	-	-	-	-	-	-	-	-	-	-	-	rw	rw	rw	rw

Bit	Function
ODP7.y	**Port 7Open Drain control register bit y** ODP7.y = 0: Port line P7.y output driver in push/pull mode ODP7.y = 1: Port line P7.y output driver in open drain mode

7.8.1 Alternate functions of Port 7

The table below summarizes the alternate functions of Port 7.

Port 7Pin	Alternate Function
P7.3	POUT3 PWM (Channel 3) Output

Table 23 Port 7 alternate functions

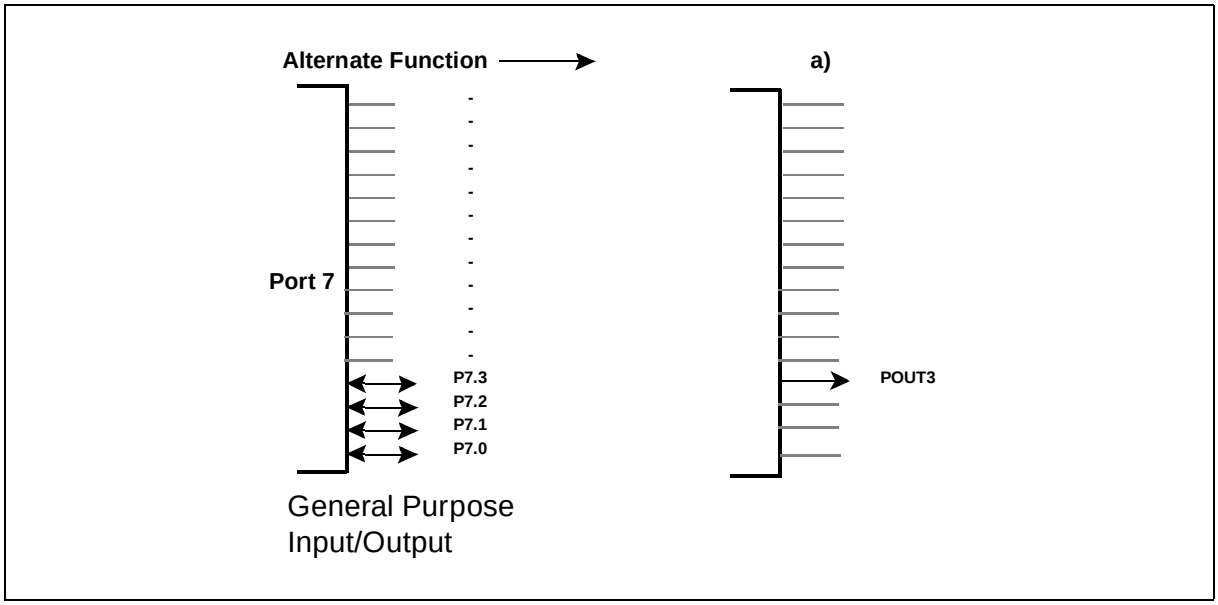


Figure 48 Port 7 I/O and Alternate Functions

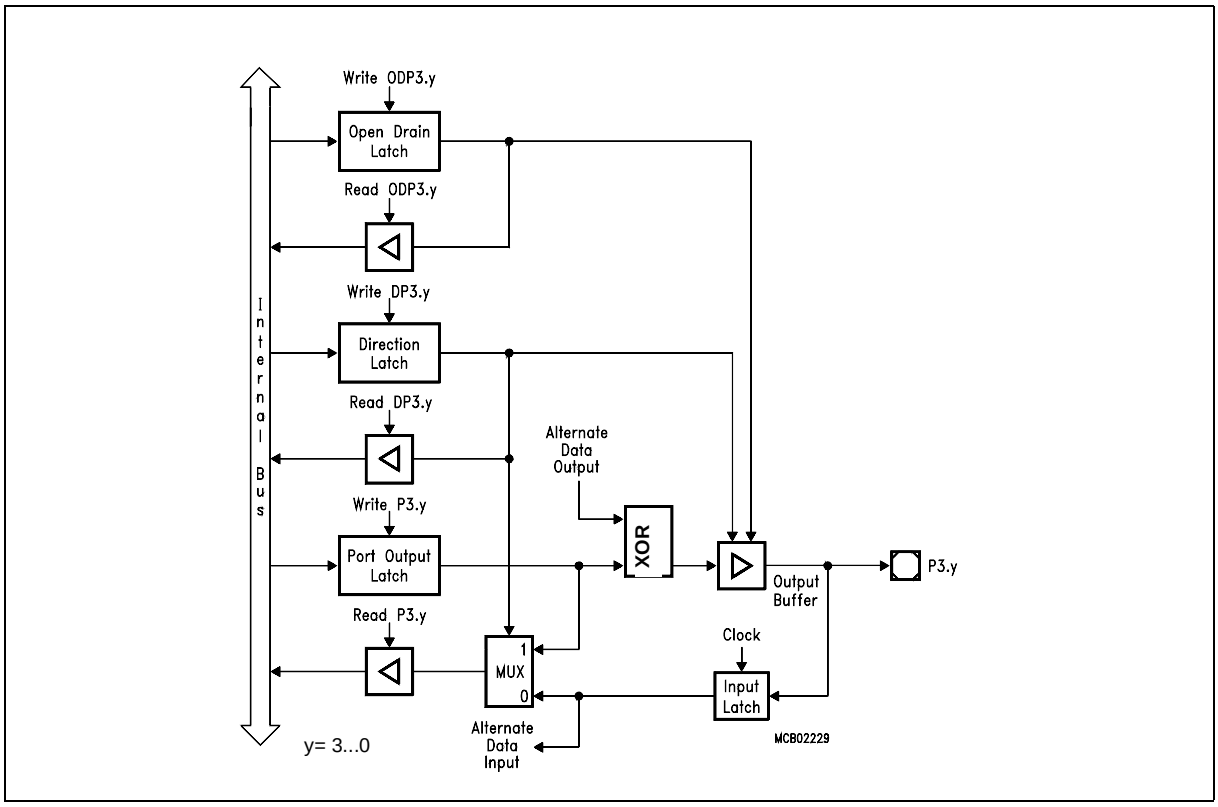


Figure 49 Block diagram of a port 7 pin (P7.3..P7.0)

The port structure of pins p7.0 through P7.3 is similar to the structure of Port3 pins with an alternate output function (e.g. T3OUT, T6OUT, etc.), as it is described in “Alternate functions of Port 3” on page 125. The exception is, however, that the port output latch value and the alternate data output are not ANDed, but EXORed. This feature inverts the alternate output by writing a ‘1’ into the respective output latch. With a ‘0’ in the port latch, the alternate output is not inverted. With this option, however, separate alternate output enable control bit must be provided in PWM unit (i.e. PEN0 bit in PWMCON1).

8 DEDICATED PINS

Most of the input/output or control signals of the functional the ST10R272L are realized as alternate functions of pins of the parallel ports. There is, however, a number of signals that use separate pins, including the oscillator, special control signals and the power supply.

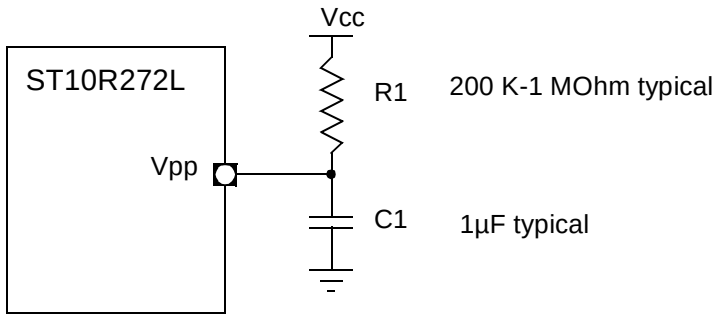
The table below summarizes the dedicated pins of the ST10R272L.

Pin(s)	Function
ALE	Address Latch Enable: controls external address latches that provide a stable address in multiplexed bus modes. ALE is activated / not activated
RD	External Read Strobe: controls the output drivers of external memory or peripherals when the ST10R272L reads data from these external devices. During reset and during Hold mode an internal pullup ensures an inactive (high) level on the $\overline{RD}$ output.
$\overline{WR}$/$\overline{WRL}$	External Write/Write Low Strobe: controls the data transfer from the ST10R272L to an external memory or peripheral device. This pin may either provide an general $\overline{WR}$ signal activated for both byte and word write accesses, or specifically control the low byte of an external 16-bit device ($\overline{WRL}$) together with the signal $\overline{WRH}$ (alternate function of P3.12/$\overline{BHE}$). During reset and during Hold mode an internal pullup ensures an inactive (high) level on the $\overline{WR}$/$\overline{WRL}$ output.
$\overline{READY}$/READY	Ready Input: 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. $\overline{READY}$/READY may be used as synchronous $\overline{READY}$/READY or may be evaluated asynchronously.
EA	External Access Enable: after reset, this pin sets code access either from the internal ROM ($\overline{EA}$='1') or from the external bus interface ($\overline{EA}$='0').
NMI	Non-Maskable Interrupt Input: triggers a high priority trap via an external signal (e.g. a power-fail signal). It also serves to validate the PWRDN instruction that switches the ST10R272L into protected power-down mode.
RSTIN	Reset Input: puts the ST10R272L into the well defined reset condition either at power-up or external events like a hardware failure or manual reset. The input voltage threshold of the $\overline{RSTIN}$ pin is raised compared to the standard pins in order to minimize the noise sensitivity of the reset input. An internal pullup resistor permits power on reset using only a capacitor connected to Vss. In bi-directional reset mode, the $\overline{RSTIN}$ line is pulled low for the duration of the internal reset sequence.

Table 24 Summary of dedicated pins

Pin(s)	Function
RSTOUT	Reset Output: provides a special reset signal for external circuitry. $\overline{\text{RSTOUT}}$ is activated at the beginning of the reset sequence, triggered via $\overline{\text{RSTIN}}$, a watchdog timer overflow or by the SRST instruction. $\overline{\text{RSTOUT}}$ remains active (low) until the EINIT instruction is executed. This allows to initialize the controller before the external circuitry is activated.
XTAL1, XTAL2	Oscillator Input/Output: connect the internal clock oscillator to the external crystal. An external clock signal may be fed to the input XTAL1, leaving XTAL2 open.
V_{CC}, V_{SS}	Digital Power Supply and Ground (6 pins each): provides the power supply for the digital logic of the ST10R272L. All V_{CC} pins and all V_{SS} pins must be connected to the power supply and ground, respectively.
V_{PP}/RPD	Flash Programming Voltage for ST10F262 or Exit From Powerdown for all derivtives If a Fast External Interrupt pin (EX3IN..EX0IN) is used to Exit From Powerdown mode, an external RC circuit should be connected to the Vpp/RPD pin. The discharging of the external capacitor causes a delay that allows the oscillator and PLL circuits to stabilize before the clock signal is delivered to the CPU and peripherals, refer to the figure below. For more information on exiting power down mode refer to“POWER REDUCTION MODES” on page 293.

Table 24 Summary of dedicated pins (Continued)



Vpp external RC circuit for exiting powerdown mode with external interrupt and for power-up asynchronous reset

Figure 50 Vpp external RC ciruit

9 EXTERNAL BUS INTERFACE

Although the ST10R272L 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 16 MByte. The external bus interface is used to access external peripherals and additional volatile and non-volatile memory.

Accesses to external memory or peripherals are executed by the integrated External Bus Controller (EBC). It can be programmed either to Single Chip Mode when no external memory is required, or to one of four different external memory access modes. 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 are used to define four independent address windows, while all external accesses outside these windows are controlled by the BUSCON0 register.

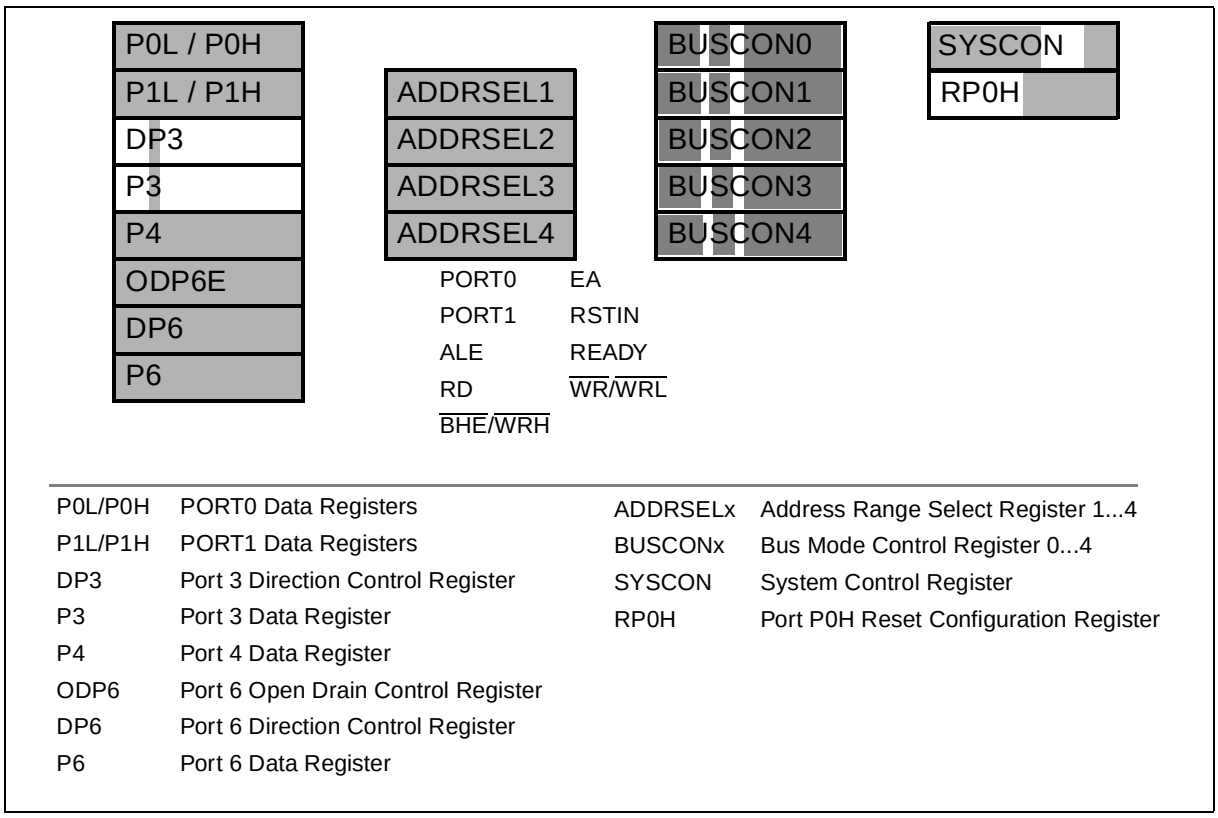


Figure 51 SFRs and port pins associated with the external bus interface

9.1 Single chip mode

Single chip mode is entered when pin $\overline{EA}$ is high during reset. Register BUSCON0 is cleared (except bit ALECTL0 and bits BTYP0[1:0] = P0L.[7:6]), which resets bit BUSACT0 of BUSCON0 register. No external bus is enabled.

In single chip mode the ST10R272L operates from internal resources. No external bus is configured and no external peripherals and/or memory can be accessed, and no port lines are occupied for the bus interface. The ROMless ST10R262 cannot operate in single chip mode, and the $\overline{EA}$ pin must be forced at 0 during reset.

9.2 External bus modes

All external memory accesses are performed by the on-chip External Bus Controller which can be programmed either to single chip mode when no external memory is required, or to the following external memory access modes:

16-bit data	demultiplexed	16-/18-/20-/24-bit addresses
16-bit data	multiplexed	16-/18-/20-/24-bit addresses
8-bit data	multiplexed	16-/18-/20-/24-bit addresses
8-bit data	demultiplexed	16-/18-/20-/24-bit addresses

In the demultiplexed bus modes, addresses are output on PORT1 and data is input/output on PORT0/P0L, respectively. In the multiplexed bus modes both addresses and data use PORT0 for input/output.

When the external bus interface is enabled and configured (bit BUSACTx='1' and bitfield BTYP), the ST10R272L uses a subset of its port lines together with some control lines to build the external bus.

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. Afterwards, BUSCON0 may be modified via software just like the other BUSCON registers.

The 16 MByte address space of the ST10R272L is divided into 256 segments of 64 KByte. 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. For applications which require less than 16 MBytes of external memory space, this address space can be restricted to 1 MByte, 256 KByte or to 64 KByte.

In this case Port 4 outputs four, two or no address lines at all. It outputs all 8 address lines, if an address space of 16 MBytes is used.

Note *When the on-chip SSP Module is to be used the segment address output on Port 4 must be limited to 4 bits (i.e. A19...A16) in order to enable the alternate function of the SSP interface pins.*

Bit SGTDIS of SYSCON register defines whether or not the CSP register is saved during interrupt entry.

9.2.1 Multiplexed bus modes

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

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

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

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

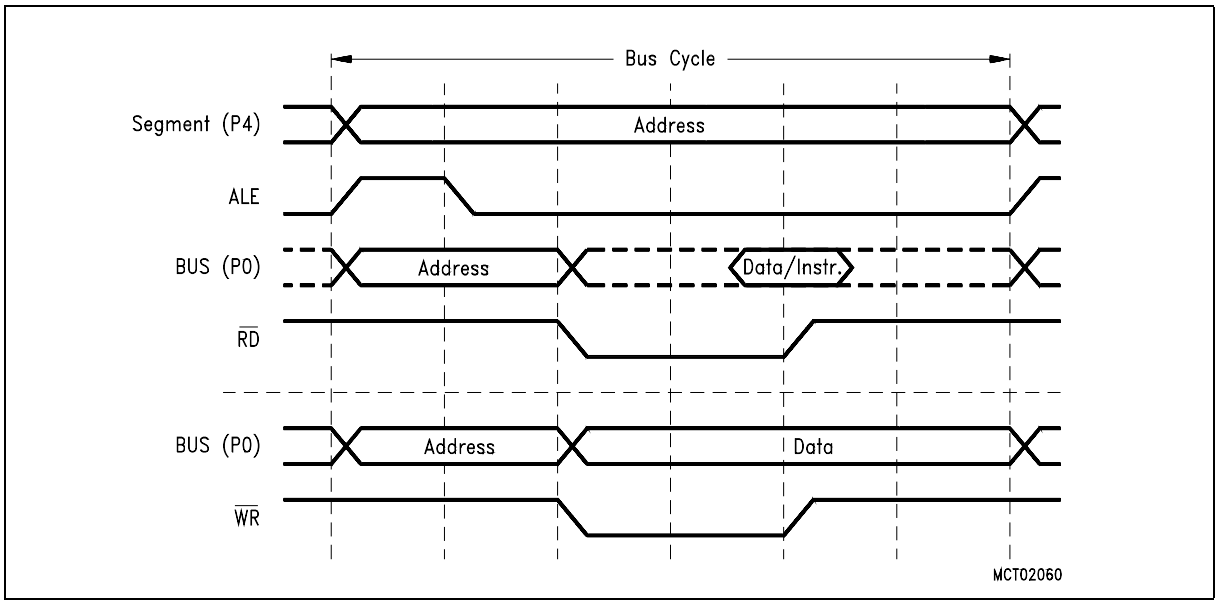


Figure 52 Multiplexed bus cycle

9.2.2 Demultiplexed bus modes

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

The EBC initiates an external access by placing an address on the address bus. After a programmable period of time the EBC activates the 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.

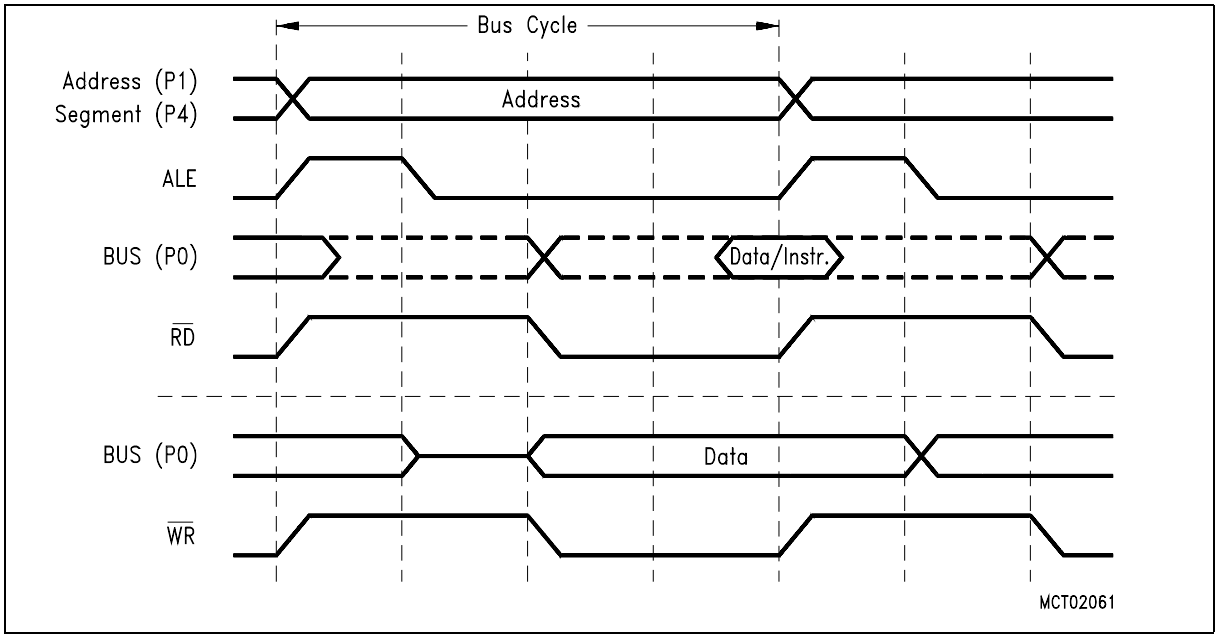


Figure 53 Demultiplexed bus cycle

9.2.3 Switching between bus modes

The EBC can be used to switch between different bus modes dynamically, i.e. subsequent external bus cycles may be executed in different ways. Certain address areas may use multiplexed or demultiplexed buses or use $\overline{\text{READY}}$ control or predefined waitstates.

A change of the external bus characteristics can be initiated in two different ways:

Reprogramming the BUSCON and/or ADDRSEL registers allows to either change the bus mode for a given address window, or change the size of an address window that uses a certain bus mode. Reprogramming allows to use a great number of different address windows (more than BUSCONs are available) on the expense of the overhead for changing the registers and keeping appropriate tables.

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

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

Note: Never change the configuration for an address area that currently supplies the instruction stream. Due to the internal pipelining it is very difficult to determine the first instruction fetch that will use the new configuration. Only change the configuration for address areas that are not currently accessed. This applies to BUSCON registers as well as to ADDRSEL registers.

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

Switching from demultiplexed to multiplexed bus mode represents a special case. The bus cycle is started by activating ALE and driving the address to Port 4 and PORT1 as usual, if another BUSCON register selects a demultiplexed bus. However, in the multiplexed bus modes the address is also required on PORT0. In this special case the address on PORT0 is delayed by one CPU clock cycle, which delays the complete (multiplexed) bus cycle and extends the corresponding ALE signal (see figure below).

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

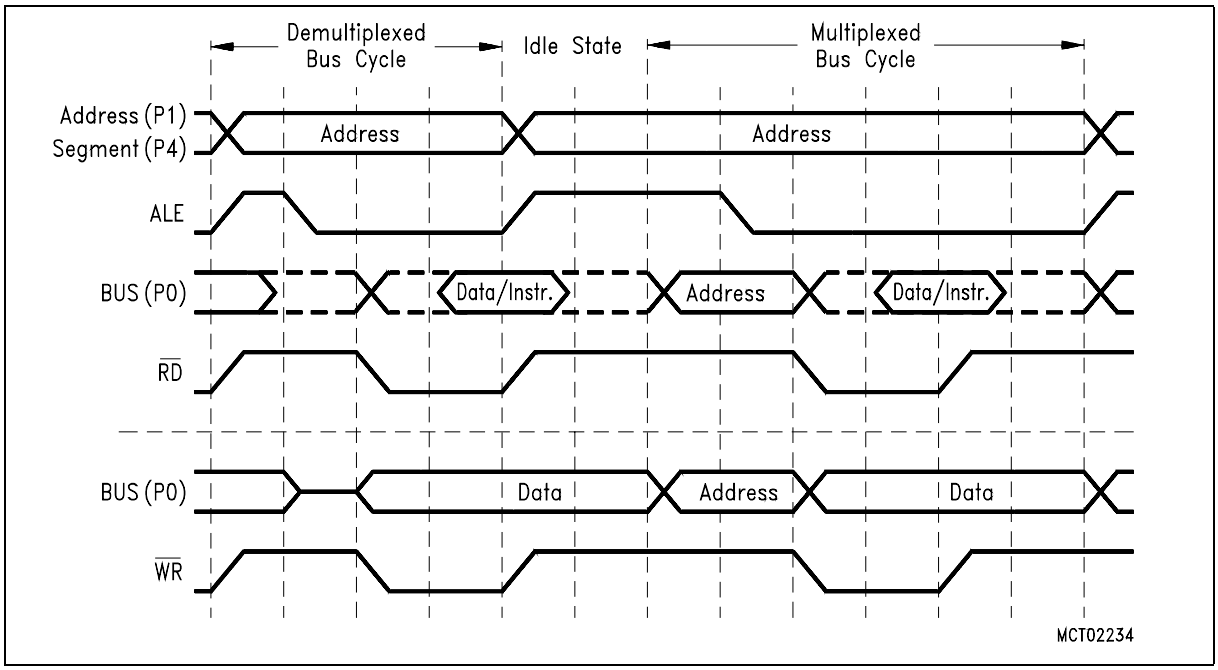


Figure 54 Switching from demultiplexed to multiplexed bus mode

9.2.4 External data bus width

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

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

Byte accesses on a 16-bit data bus require that the upper and lower half of the memory can be accessed individually. In this case the upper byte is selected with the $\overline{BHE}$ 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 halves.

When reading bytes from an external 16-bit device, whole words may be read and the ST10R272L 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.

PORT1 gets available for general purpose I/O, when none of the BUSCON registers selects a demultiplexed bus mode.

9.2.5 Disable/enable control for pin $\overline{BHE}$ (BYTDIS)

Bit BYTDIS of SYSCON register controlls the active-low Byte High Enable ($\overline{BHE}$) pin. The $\overline{BHE}$ pin function 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 ST10R272L via a word-wide external data bus. After reset, the $\overline{BHE}$ function is automatically enabled if a 16-bit data bus is selected during reset (BYTDIS = '0'), 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.

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 Demultiplexed	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 Demultiplexed	Very high	(1 / 1 / 2)	Low (no latch, word bus)	---

9.2.6 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 pulldowns)
1 0	Eight:	A23...A16	16	MByte (max.)
0 1	None		64	KByte (min.)
0 0	Four:	A19...A16	1	MByte

Table 25 Address segmentation

9.2.7 $\overline{CS}$ signal generation

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

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

Table 26 Bitfield CSEL in RPOH register

The $\overline{CSx}$ outputs are associated with the BUSCONx registers and are driven active (low) for any access within the address area defined for the respective BUSCON register. For any access outside this defined address area the respective $\overline{CSx}$ signal will go inactive (high).

Note No $\overline{CSx}$ signal will be generated for an access to any internal address area, even if this area is covered by the respective ADDRSELx register.

The chip select signals can be used for four operating modes, selected via bits CSWENx and CSRENx in the respective BUSCONx register.

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

Table 27 Chip select mode

Address chip select signals remain active for the whole external bus cycle. 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.

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

Internal pullup devices hold the selected $\overline{CS}$ 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, 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.

9.2.8 Segment address versus chip select

The external bus interface of the ST10R272L supports many configurations for the external memory. By increasing the number of segment address lines the ST10R272L can address a linear address space of 256 KByte, 1 MByte or 16 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 ST10R272L 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 5 chip select lines e.g. allows to access five memory banks of 1 MByte each. So the available address space is 5 MByte (without glue logic).

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

9.3 Programmable bus characteristics

Important timing characteristics of the external bus interface (Memory Cycle Time, Memory Tri-State Time, Length of ALE and Read Write Delay) have been made programmable to allow the user the use of a wide range of different types of memories and external peripherals. In addition, up to 4 independent address windows may be defined (via register pairs ADDRSELx / BUSCONx) which allow to access different resources with different bus characteristics. These address windows are arranged hierarchically where BUSCON4 overrides BUSCON3 and BUSCON2 overrides BUSCON1. All accesses to locations not covered by these 4 address windows are controlled by BUSCON0. Up to 5 external $\overline{CS}$ signals (4 windows plus default) can be generated in order to save external glue logic. Access to very slow memories is supported via a particular ‘Ready’ function.

The following parameters of an external bus cycle are programmable:

ALE control:	Defines the ALE signal length and the address hold time after its falling edge.
Memory cycle time:	Defines the allowable access time (extendable wit h 1...15 waitstates).
Memory tri-state time:	Defines the time for a data driver to float (extendable with 1 waitstate).
Read/write delay time:	Defines when a command is activated after falling edge of ALE.
READY:	Polarity is programmable.
Chip select timing control:	Adjusts the position of the CSx lines.

Table 28 ProgrammableEBI parameters

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

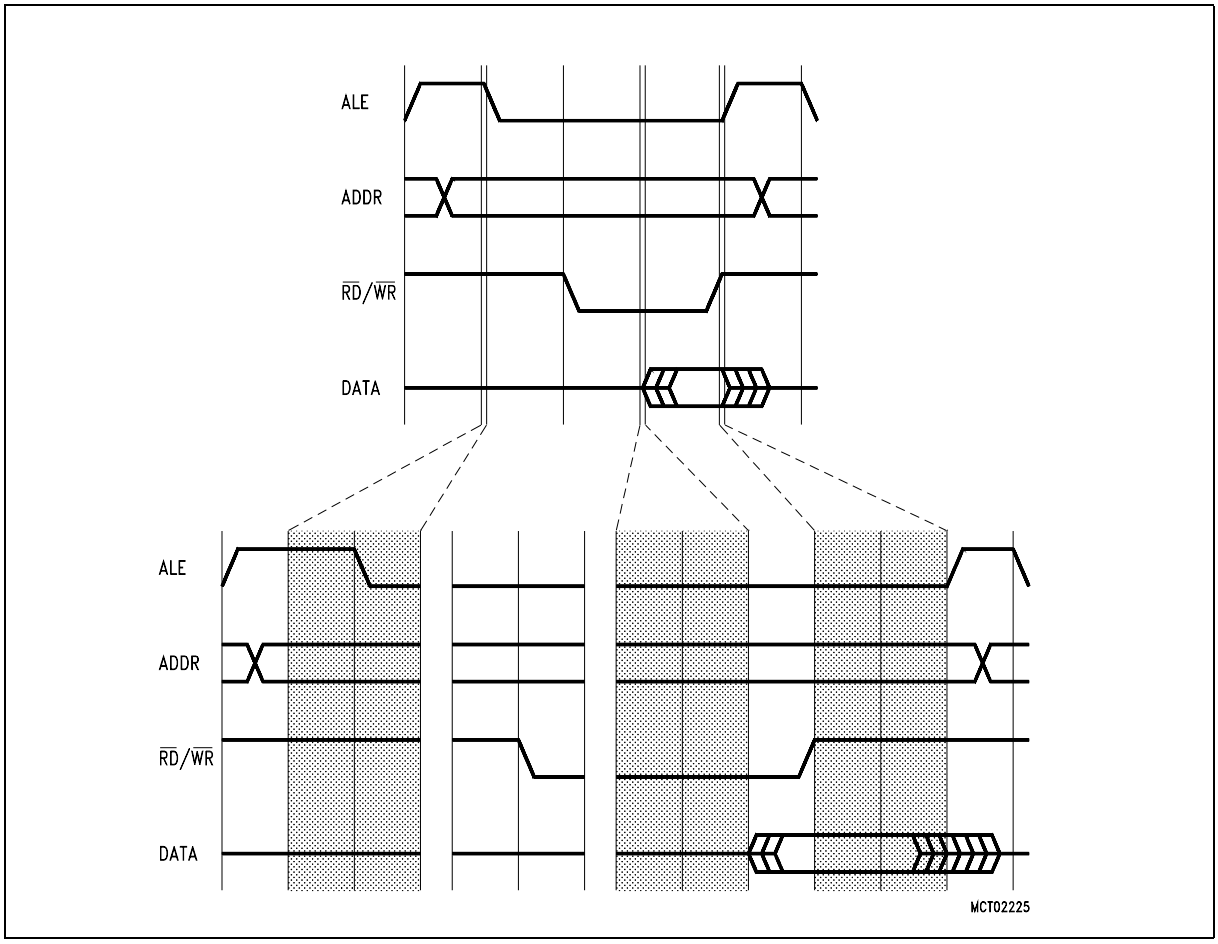


Figure 55 Programmable external bus cycle

9.3.1 ALE length control

The length of the ALE signal and the address hold time after its falling edge are controlled by the ALECTLx bits in the BUSCON registers. When bit ALECTL is set to '1', external bus cycles accessing the 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 (i.e. the data transfer is delayed by one CPU clock). This allows more time for the address to be latched.

ALECTL0 is '1' after reset to select the slowest possible bus cycle, the other ALECTLx are '0' after reset.

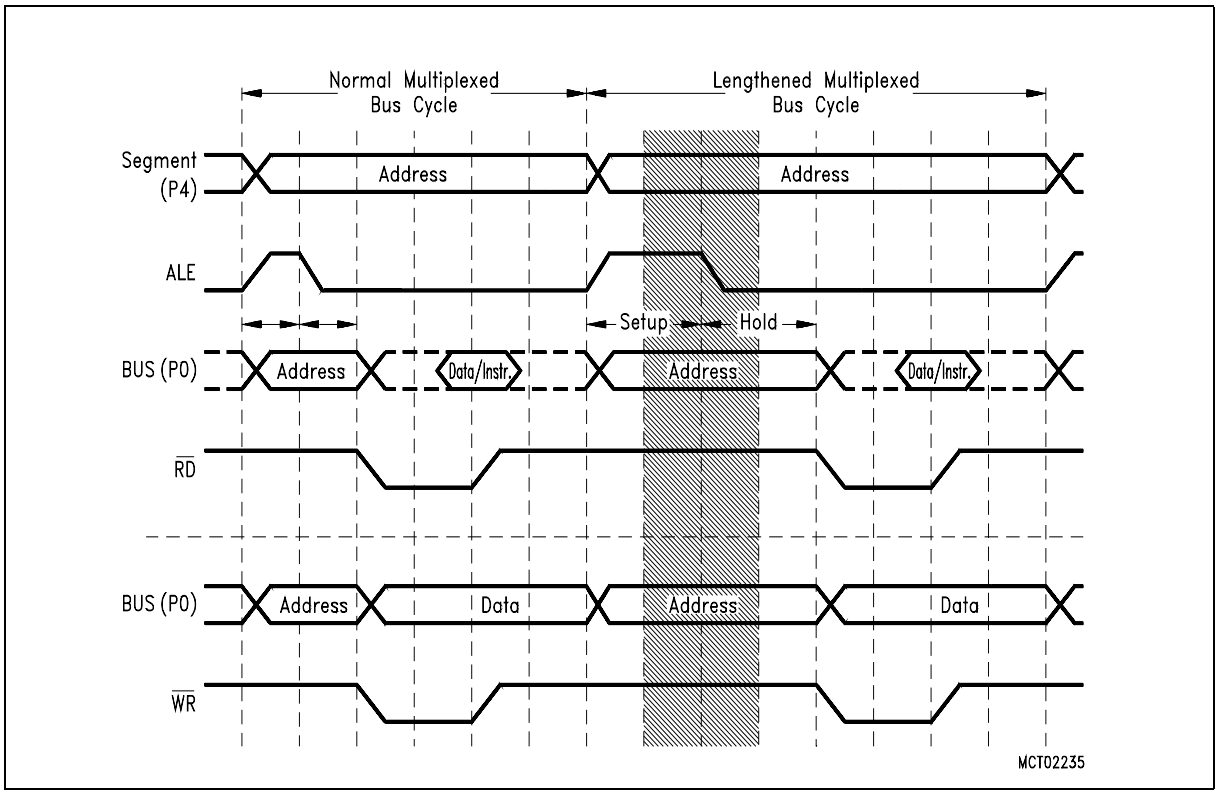


Figure 56 ALE length control

9.3.2 Programmable memory cycle time

The ST10R272L allows the user to adjust the controller's external bus cycles to the access time of the respective memory or peripheral. This access time is the total time required to move the data to the destination. It represents the period of time during which the controller's signals do not change.

The external bus cycles of the ST10R272L can be extended for a memory or peripherals 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.

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.

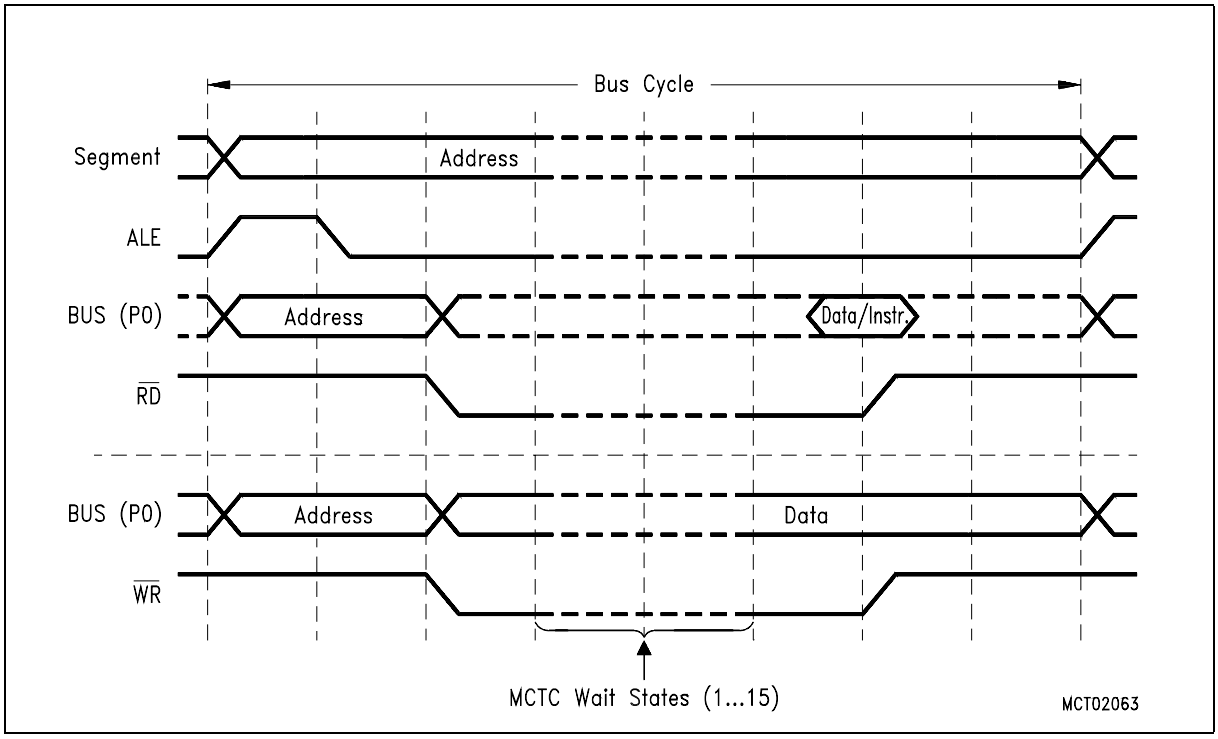


Figure 57 Memory cycle time

9.3.3 Programmable memory tri-state time

The time between two subsequent external accesses can be adjusted 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 ($\overline{RD}$).

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 waitstate 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 and is controlled by the MTTCx bits of the BUSCON registers. A waitstate will be inserted, if bit MTTCx is ‘0’ (default after reset).

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

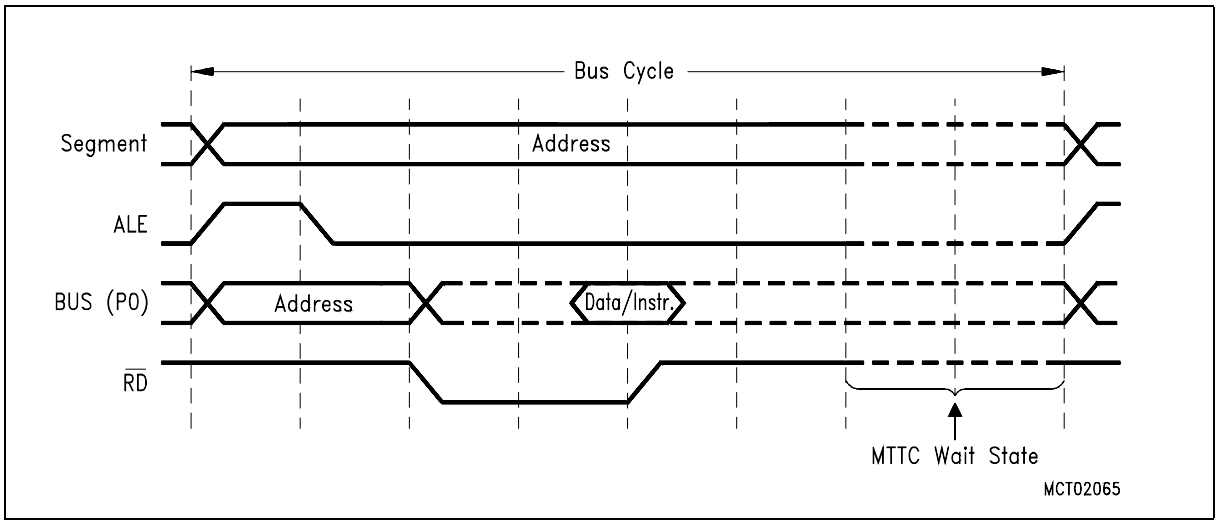


Figure 58 Memory tri-state time

9.3.4 Read/write delay time

The timing of the read and write commands can be adjusted to take account of the 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 ST10R272L's address, when the early $\overline{RD}$ signal is used. Therefore multiplexed bus cycles should always be programmed with read/write delay.

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

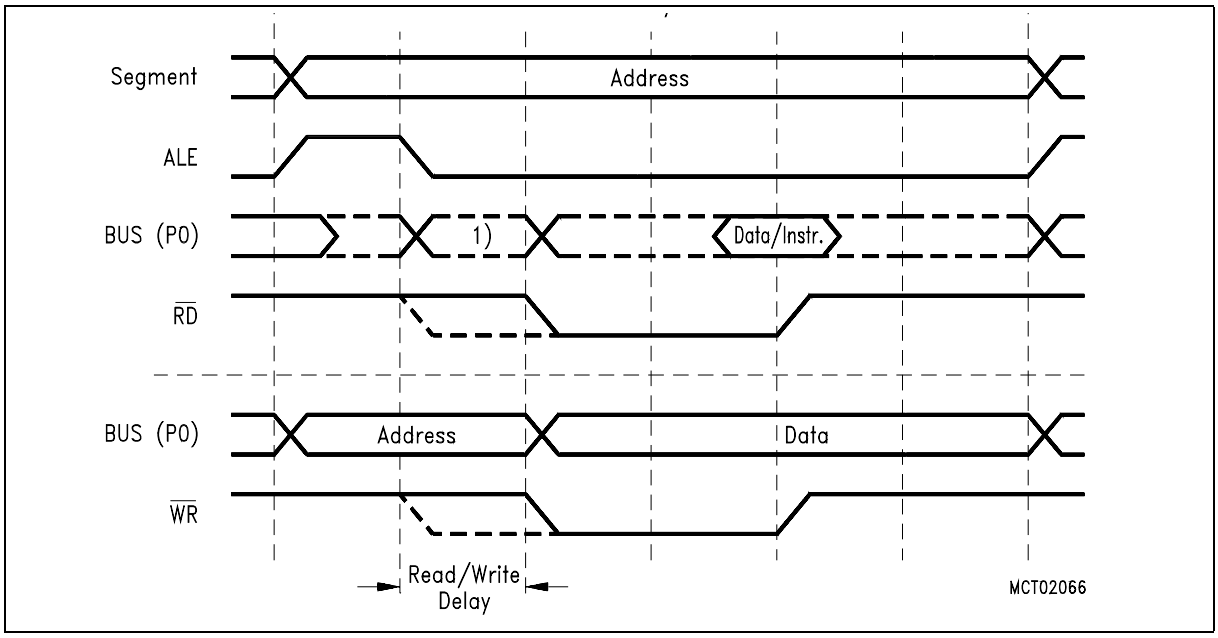


Figure 59 Read/write delay

9.3.5 READY/$\overline{\text{READY}}$ controlled bus cycles

The active level of the ready pin can be set to READY or $\overline{\text{READY}}$ by the RDYPOL bit 13 in the BUSCON register.

Where the programmable waitstates are not enough, or where the response (access) time of a peripheral is not constant, the ST10R272L provides external bus cycles that are terminated by a READY or $\overline{\text{READY}}$ input signal (synchronous or asynchronous). In this case the ST10R272L first inserts a programmable number of waitstates (0...7) and then monitors the READY or $\overline{\text{READY}}$ line to determine the actual end of the current bus cycle. The external device drives READY or $\overline{\text{READY}}$ low in order to indicate that data has been latched (write cycle) or are available (read cycle).

When the READY or $\overline{\text{READY}}$ function is enabled for a specific address window, each bus cycle in this window must be terminated with the active level defined by the RDYPOL bit 13 in the associated BUSCON register.

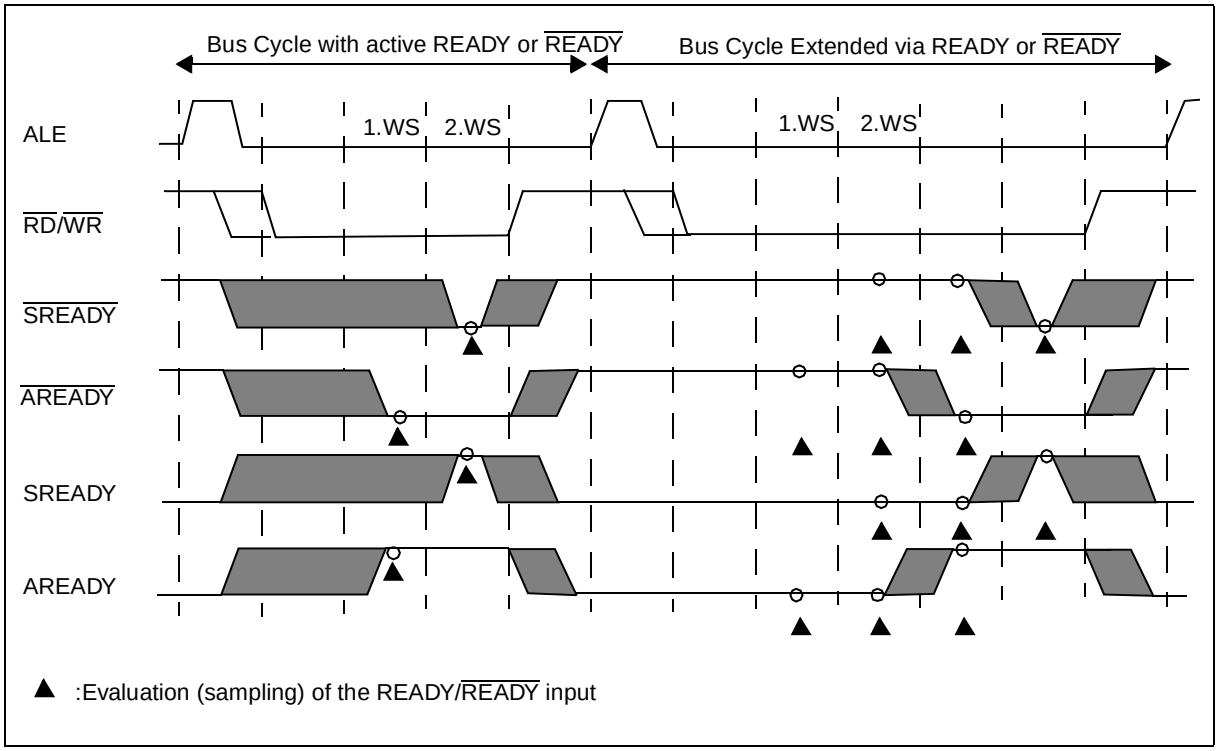


Figure 60 READY/$\overline{\text{READY}}$ controlled bus cycles

The READY/$\overline{\text{READY}}$ function is enabled by the RDYENx bits in the BUSCON registers. When this function is selected (RDYENx = '1'), only the lower 3 bits of the respective MCTC bit field define the number of inserted waitstates (0...7), while the MSB of bit field MCTC selects the $\overline{\text{READY}}$ operation:

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

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

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

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

Note *When the READY/$\overline{\text{READY}}$ function is enabled for a specific address window, each bus cycle within this window must be terminated with an active READY/$\overline{\text{READY}}$ signal. Otherwise the controller hangs until the next reset. A time-out 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/$\overline{\text{READY}}$ may be grouped into the same address window. The (external) waitstate control logic in this case would activate READY/$\overline{\text{READY}}$ either upon the memory's chip select or with the peripheral's READY/$\overline{\text{READY}}$ output. After the predefined number of waitstates the ST10R272L will check its READY/$\overline{\text{READY}}$ line to determine the end of the bus cycle. For a memory access it will be low already (see example a in the figure above), for a peripheral access it may be delayed (see example b in the figure above). As memories tend to be faster than peripherals, there should be no impact on system performance.
- When using the READY/$\overline{\text{READY}}$ function with so-called "normally-ready" peripherals, it may lead to erroneous bus cycles, if the READY/$\overline{\text{READY}}$ line is sampled too early. These peripherals pull their READY/$\overline{\text{READY}}$ output low, while they are idle. When they are accessed, they deactivate READY/$\overline{\text{READY}}$ until the bus cycle is complete, then drive it low again. If, however, the peripheral deactivates READY/$\overline{\text{READY}}$ after the first sample point of the ST10R272L, the controller samples an active READY/$\overline{\text{READY}}$ and terminates the current bus cycle, which, of course, is too early. By inserting predefined waitstates the first READY/$\overline{\text{READY}}$ sample point can be shifted to a time, where the peripheral has safely controlled the READY/$\overline{\text{READY}}$ line (e.g. after 2 waitstates in the figure above).

9.3.6 Programmable chip select timing control

The position of the CSx lines can be changed. By default (after reset), the CSx lines change half a CPU clock cycle after the rising edge of ALE. With the CSCFG bit set in the SYSCON register ("Registers" on page 164), the CSx lines change with the rising edge of ALE, therefore the CSx lines change at the same time as the address lines.

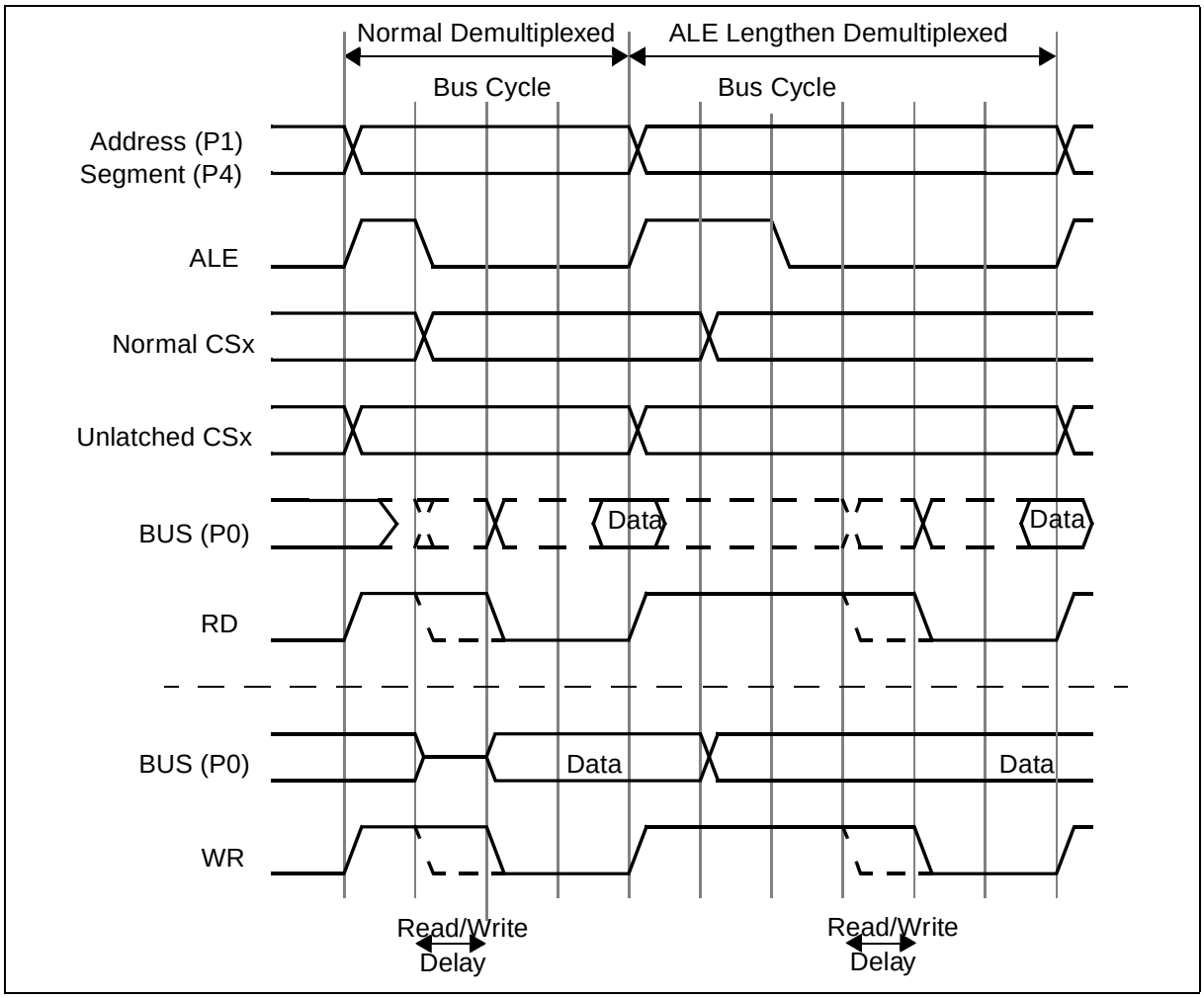


Figure 61 Chip select delay

9.4 Controlling the external bus controller

A set of registers controls the EBC. General features like the use of interface pins ($\overline{WR}$, $\overline{BHE}$), segmentation and internal ROM mapping are controlled by the SYSCON register. The bus cycle properties such as chip select mode, the use 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 associated address select register (ADDRSEL4...ADDRSEL1) which specifies up to four address areas and the individual bus characteristics within these areas. All accesses that are not covered by these four areas are controlled via BUSCON0. This makes it possible to use memory components or peripherals with different interfaces within the same system, with optimized access.

9.4.1 Registers

SYSCON (FF12h / 89h) **SFR** **Reset Value: 0XX0h**)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STKSZ			ROMS1	SGTDIS	ROMEN	BYTDIS	CLKEN	WRCFG	CSCFG	PWDCFG	OWDDIS	-	SSPEN	VISIBLE	XPER-SHARE
rw			rw	rw	rw	rw	rw	rw	rw	rw	rw	-	rw	rw	rw

Bit	Function
XPER-SHARE	**XBUS Peripheral Share Mode Control** ‘0’: External accesses to XBUS peripherals are disabled ‘1’: XBUS peripherals are accessible via the external bus during hold mode.
VISIBLE	**Visible Mode Control** ‘0’: Accesses to XBUS peripherals are done internally ‘1’: XBUS peripheral accesses are made visible on the external pins.
SSPEN	**Xperipheral SSP Enable Control** ‘0’: SSP is disabled. Pins P4.[7..4] are general purpose I/Os or segment address lines ‘1’: SSP is enabled. Pins P4.[7..4] are SSP IOs or segment address lines
OWDDIS	**Oscillator Watchdog Disable Control** ‘0’: Oscillator Watchdog (OWD) is enabled. If PLL is bypassed, the OWD monitors XTAL1 activity. If there is no activity on XTAL1 for at least 1 ms, the CPU clock is switched automatically to PLL’s base frequency (around 5 MHz). ‘1’: OWD is disabled. If the PLL is bypassed, the CPU clock is always driven by XTAL1 signal. The PLL is turned off to reduce the power supply current.
PWDCFG	**Power Down Mode Configuration Control** ‘0’: Power Down Mode can only be entered during PWRDN instruction execution if $\overline{\text{NMI}}$ pin is low, otherwise the instruction has no effect. To exit Power Down Mode, an external reset must occurs by asserting the $\overline{\text{RSTIN}}$ pin. ‘1’: Power Down Mode can only be entered during PWRDN instruction execution if all enabled fast external interrupt EXxIN pins are in their inactive level. Exiting this mode can be done by asserting one enabled EXxIN pin.
CSCFG	**Chip Select Configuration Control** ‘0’: Latched Chip Select lines: CSx change 1 TCL after rising edge of ALE ‘1’: Unlatched Chip Select lines: CSx change with rising edge of ALE
WRCFG	**Write Configuration Control** (Set according to pin $\overline{\text{P0H.0}}$ during reset) ‘0’: Pins $\overline{\text{WR}}$ and $\overline{\text{BHE}}$ retain their normal function ‘1’: Pin $\overline{\text{WR}}$ acts as $\overline{\text{WRL}}$, pin $\overline{\text{BHE}}$ acts as $\overline{\text{WRH}}$

Bit	Function
CLKEN	**System Clock Output Enable** (CLKOUT) '0': CLKOUT disabled: pin may be used for general purpose IO '1': CLKOUT enabled: pin outputs the system clock signal
BYTDIS	**Disable/Enable Control for Pin BHE** (Set according to data bus width) '0': Pin $\overline{\text{BHE}}$ enabled '1': Pin $\overline{\text{BHE}}$ disabled, pin may be used for general purpose IO
ROMEN	**Internal ROM Enable** (Set according to pin $\overline{\text{EA}}$ during reset) '0': Internal ROM disabled: accesses to the ROM area use the external bus '1': Internal ROM enabled This bit is not relevant on the ST10R262 since it does not include internal ROM. It should be kept to 0
SGTDIS	**Segmentation Disable/Enable Control** '0': Segmentation enabled (CSP is saved/restored during interrupt entry/exit) '1': Segmentation disabled (Only IP is saved/restored)
ROMS1	**Internal ROM Mapping** '0': Internal ROM area mapped to segment 0 (00'0000h...00'7FFFh) '1': Internal ROM area mapped to segment 1 (01'0000h...01'7FFFh) This bit is not relevant on the ST10R262 since it does not include internal ROM. It should be kept to 0.
STKSZ	**System Stack Size** Selects the size of the system stack (in the internal RAM) from 32 to 1024 words

Note SYSCON register cannot be changed after execution of the EINIT instruction. Bit SGTDIS controls the correct stack operation (push/pop of CSP or not) during traps and interrupts. Bits marked with "-" must be kept at 0.

The layout of the five BUSCON registers is identical. Registers BUSCON4...BUSCON1, which control the selected address windows, are completely under software control, while register BUSCON0, which e.g. is also used for the very first code access after reset, is partly controlled by hardware, i.e. it is initialized via PORT0 during the reset sequence. This hardware control allows to define an appropriate external bus for systems, where no internal program memory is provided. The active level of the READY pin can be selected by software via the RDYPOL bit. When the READY function is enabled for a specific address window, each bus cycle within this window must be terminated with the active level defined by the RDYPOL bit in the associated BUSCON register

BUSCON0 (FF0Ch / 86h) **SFR** **Reset Value: 0XX0h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSW EN0	CSR EN0	RDY POL0	RDY EN0	-	BUS ACT0	ALE CTL0	-	BTYP		MTT C0	RWD C0	MCTC			
rw	rw	rw	rw	-	rw	rw	-	rw		rw	rw	rw			

BUSCON1 (FF14h / 8Ah) **SFR** **Reset Value: 0000h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSW EN1	CSR EN1	RDY POL1	RDY EN1	-	BUS ACT1	ALE CTL1	-	BTYP		MTT C1	RWD C1	MCTC			
rw	rw	rw	rw	-	rw	rw	-	rw		rw	rw	rw			

BUSCON2 (FF16h / 8Bh) **SFR** **Reset Value: 0000h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSW EN2	CSR EN2	RDY POL2	RDY EN2	-	BUS ACT2	ALE CTL2	-	BTYP		MTT C2	RWD C2	MCTC			
rw	rw	rw	rw	-	rw	rw	-	rw		rw	rw	rw			

BUSCON2 (FF18h / 8Bh) **SFR** **Reset Value: 0000h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSW EN3	CSR EN3	RDY POL3	RDY EN3	-	BUS ACT3	ALE CTL3	-	BTYP		MTT C3	RWD C3	MCTC			
rw	rw	rw	rw	-	rw	rw	-	rw		rw	rw	rw			

BUSCON4 (FF1Ah / 8Dh) **SFR** **Reset Value: 0000h**

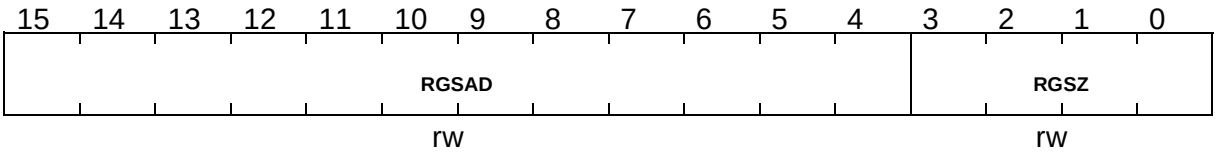
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSW EN4	CSR EN4	RDY POL4	RDY EN4	-	BUS ACT4	ALE CTL4	-	BTYP		MTT C4	RWD C4	MCTC			
rw	rw	rw	rw	-	rw	rw	-	rw		rw	rw	rw			

Bit	Function
MCTCx	**Memory Cycle Time Control** (Number of memory cycle time wait states) 0 0 0 0: 15 waitstates (Number = 15 - <MCTC>) . . . 1 1 1 1: 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

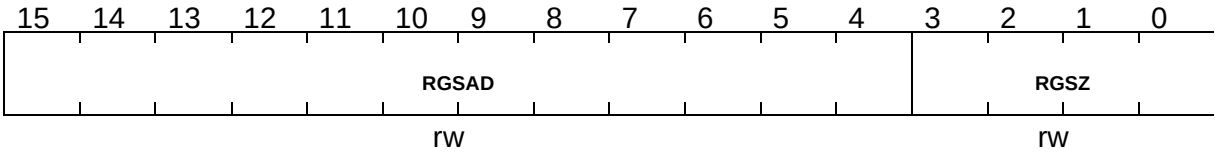
Bit	Function
MTTCx	**Memory Tristate Time Control** ‘0’: 1 waitstate ‘1’: No waitstate
BTYPx	**External Bus Configuration** 0 0: 8-bit Demultiplexed Bus 0 1: 8-bit Multiplexed Bus 1 0: 16-bit Demultiplexed Bus 1 1: 16-bit Multiplexed Bus **Note:** For BUSCON0 BTYP is defined via PORT0 during reset.
ALECTLx	**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
RDYPOLx	**Ready Active Level Control** ‘0’: The active level on the READY pin is low, bus cycle terminates with a ‘0’ on READY pin, ‘1’: The active level on the READY pin is high, bus cycle terminates with a ‘1’ on READY pin.
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 *BUSACT0 is initialized with 0, if pin $\overline{EA}$ is high during reset. If pin $\overline{EA}$ is low during reset, bit BUSACT0 is set. ALECTL0 is set (‘1’) and bit field BTYP is loaded with the bus configuration selected via PORT0.*

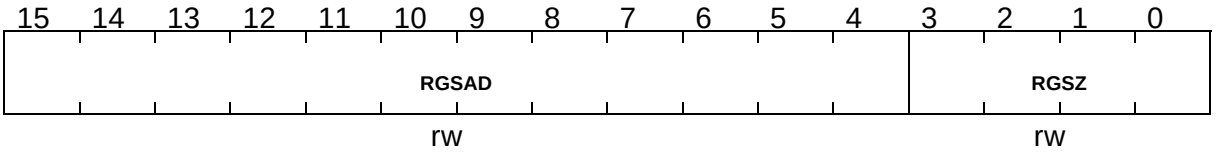
ADDRSEL1 (FE18h / 0Ch) **SFR** **Reset Value: 0000h**



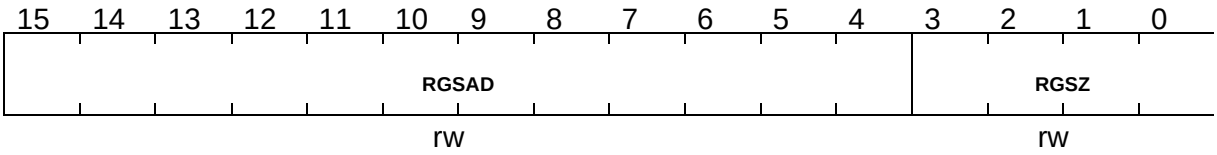
ADDRSEL2 (FE1Ah / 0Dh) **SFR** **Reset Value: 0000h**



ADDRSEL3(FE1Ch / 0Eh) **SFR** **Reset Value: 0000h**



ADDRSEL4 (FE1Eh / 0Fh) **SFR** **Reset Value: 0000h**



Bit	Function
RGSZ	**Range Size Selection** Defines the size of the address area controlled by the respective BUSCONx/ ADDRSELx register pair. See table below.
RGSAD	**Range Start Address** Defines the upper bits of the start address (A23...) of the respective address area. See table below.

Note *There is no register ADDRSEL0, as register BUSCON0 controls all external accesses outside the four address windows of BUSCON4...BUSCON1 within the complete address space.*

9.4.2 Defining address areas

The four register pairs BUSCON4/AD-DRSEL4...BUSCON1/ADDRSEL1 are used to define 4 separate address areas within the address space of the ST10R272L. Within each of these address areas external accesses can be controlled by one of the four different bus modes, independent of each other and of the bus mode specified in register BUSCON0. Each ADDRSELx register in a way cuts out an address window, within which the parameters in register BUSCONx are used to control external accesses. The range start address of such a window, defines the upper address bits, which are not used within the address window of the specified size (see table below). For a given window size only those upper address bits of the start address are used (marked “R”), which are not implicitly used for addresses inside the window. The lower bits of the start address (marked “x”) are disregarded.

Bit field RGSZ	Resulting Window Size	Relevant Bits (R) of Start Address (A23...A12)											
0 0 0 0	4 KByte	R	R	R	R	R	R	R	R	R	R	R	R
0 0 0 1	8 KByte	R	R	R	R	R	R	R	R	R	R	R	X
0 0 1 0	16 KByte	R	R	R	R	R	R	R	R	R	R	X	X
0 0 1 1	32 KByte	R	R	R	R	R	R	R	R	R	X	X	X
0 1 0 0	64 KByte	R	R	R	R	R	R	R	R	X	X	X	X
0 1 0 1	128 KByte	R	R	R	R	R	R	R	X	X	X	X	X
0 1 1 0	256 KByte	R	R	R	R	R	R	X	X	X	X	X	X
0 1 1 1	512 KByte	R	R	R	R	R	X	X	X	X	X	X	X
1 0 0 0	1 MByte	R	R	R	R	X	X	X	X	X	X	X	X
1 0 0 1	2 MByte	R	R	R	X	X	X	X	X	X	X	X	X
1 0 1 0	4 MByte	R	R	X	X	X	X	X	X	X	X	X	X
1 0 1 1	8 MByte	R	X	X	X	X	X	X	X	X	X	X	X
1 1 x x	Reserved.												

Table 29 Address area definition

9.4.3 Prioritizing address areas

A prioritizing scheme is used to allow overlapping of address areas. The ADDRSELs registers are split into two groups:

- The first group with ADDRSEL1 and ADDRSEL2, gives to ADDRSEL2 a higher priority than ADDRSEL1. Thus, an overlapping among address areas defined via registers ADDRSEL1 and 2 is allowed.

- The other group with ADDRSEL3 and ADDRSEL4, gives to ADDRSEL4 a higher priority than ADDRSEL3. Thus, an overlapping among address areas defined via registers ADDRSEL3 and 4 is allowed.

The BUSCON0 register always has the lowest priority, and its address area can be overlapped by any of the other ADDRSELs.

The Xperipheral address areas defined via XADRs registers always have higher priority than address areas defined via ADDRSELs registers.

RP0H (F108h / 84h) **SFR** **Reset Value: - - XXh**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								CLKSEL			SALSEL		CSSEL		WRCFG
-	-	-	-	-	-	-	-	r			r		r		r

Bit	Function	
WRCFG	**Write Configuration Control**	
	'0':	Pins $\overline{WR}$ and $\overline{BHE}$ retain their normal function
	'1':	Pins $\overline{WR}$ acts as $\overline{WRL}$, pin $\overline{BHE}$ acts as $\overline{WRH}$
CSSEL	**Chip Select Line Selection** (Number of active $\overline{CS}$ outputs)	
	0 0:	3 $\overline{CS}$ lines: $\overline{CS}$2...$\overline{CS}$0
	0 1:	2 $\overline{CS}$ lines: $\overline{CS}$1...$\overline{CS}$0
	1 0:	No $\overline{CS}$ lines at all
	1 1:	5 $\overline{CS}$ lines: $\overline{CS}$4...$\overline{CS}$0 (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:	8-bit segment address: A23...A16
	1 1:	2-bit segment address: A17...A16 (Default without pulldowns)
CLKSEL	**System Clock Selection**	
	000:	f_{CPU} = 2.5 * f_{OSC}
	001:	f_{CPU} = 0.5 * f_{OSC}
	010:	f_{CPU} = 1.5 * f_{OSC}
	011:	f_{CPU} = f_{OSC}
	100:	f_{CPU} = 5 * f_{OSC}
	101:	f_{CPU} = 2 * f_{OSC}
	110:	f_{CPU} = 3 * f_{OSC}
	111:	f_{CPU} = 4 * f_{OSC}

Note *RP0H cannot be changed via software, but rather allows to check the current configuration.*

9.4.4 Precautions and hints

- The external bus interface is enabled as long as at least one of the BUSCON registers has its BUSACT bit set.
- PORT1 will output the intra-segment address as long as at least one of the BUSCON registers selects a demultiplexed external bus, even for multiplexed bus cycles.
- The address areas defined via registers ADDRSEL1 and ADDRSEL2 may not overlap address areas defined via registers ADDRSEL3 and ADDRESEL4. The operation of the EBC will be unpredictable in such a case.
- 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).

9.5 EBC idle state

When the external bus interface is enabled, but no external access is currently executed, the EBC is idle. During this idle state the external interface appears in the following way:

- PORT0 goes into high impedance (floating)
- PORT1 (if used for the bus interface) drives the address used last
- Port 4 (the activated pins) drives the segment address used last
- Port 6 drives the $\overline{CS}$ signal corresponding to the address (see above), if enabled
- ALE remains inactive (low)
- $\overline{RD}$/$\overline{WR}$ remain inactive (high)

9.6 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. The ST10R272L supports this approach with the possibility to arbitrate the access to its external bus, i.e. to the external devices.

This bus arbitration allows an external master to request the ST10R272L's bus via the $\overline{HOLD}$ input. The ST10R272L acknowledges this request via the $\overline{HLDA}$ output and will float its bus lines in this case. The $\overline{CS}$ outputs may provide internal pullup devices. The new master may now access the peripheral devices or memory banks via the same interface lines as the ST10R272L. During this time the ST10R272L 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 ST10R272L needs access to its external bus while it is occupied by another bus master, it demands it via the $\overline{\text{BREQ}}$ output.

The external bus arbitration is enabled by setting bit HLDEN in register PSW to '1'. This bit 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 ST10R272L will not answer to $\overline{\text{HOLD}}$ requests from other external masters.

The pins $\overline{\text{HOLD}}$, $\overline{\text{HLDA}}$ and $\overline{\text{BREQ}}$ 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.

9.6.1 Entering the hold state

Access to the ST10R272L's external bus is requested by driving its $\overline{\text{HOLD}}$ input low. After synchronizing this signal the ST10R272L will complete a current external bus cycle (if any is active), release the external bus and grant access to it by driving the $\overline{\text{HLDA}}$ output low. During hold state the ST10R272L 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 ($\overline{\text{RD}}$, $\overline{\text{WR}}$/$\overline{\text{WRL}}$, $\overline{\text{BHE}}$/$\overline{\text{WRH}}$)
- $\overline{\text{CSx}}$ outputs are pulled high (push/pull mode) or float to tri-state (open drain mode)

Should the ST10R272L require access to its external bus during hold mode, it activates its bus request output $\overline{BREQ}$ to notify the arbitration circuitry. $\overline{BREQ}$ is activated only during hold mode. It will be inactive during normal operation.

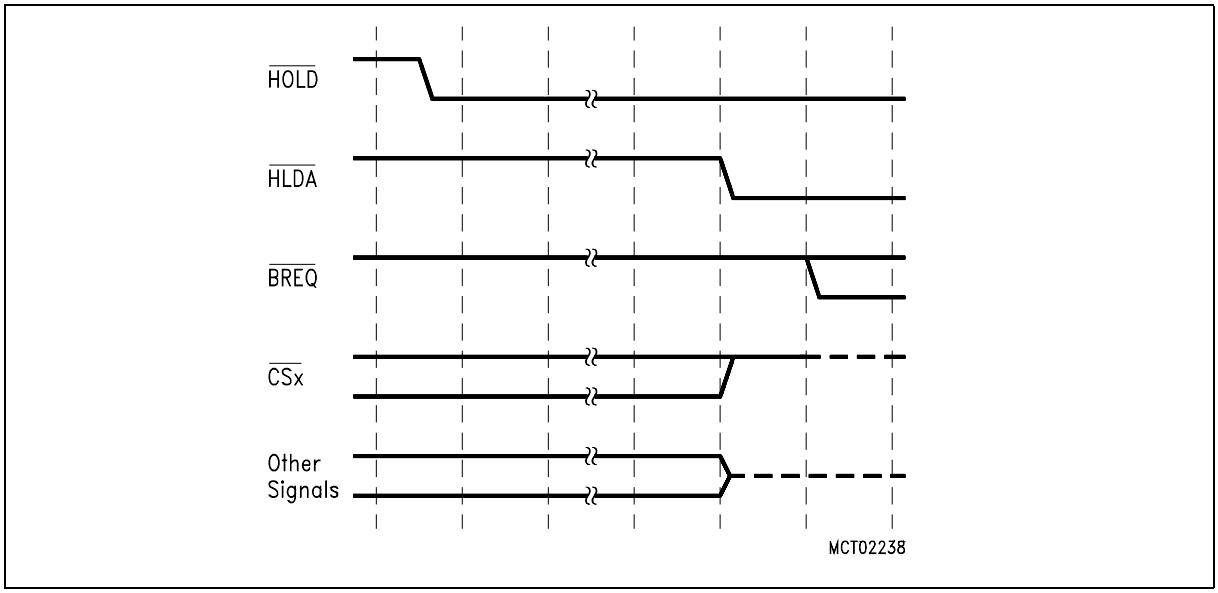


Figure 62 External bus arbitration - releasing the bus

Note *The ST10R272L completes the 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 chance that $\overline{BREQ}$ has to become active.

9.6.2 Exiting the hold state

The external bus master returns the access rights to the ST10R272L by driving the $\overline{HOLD}$ input high. After synchronizing this signal the ST10R272L will drive the $\overline{HLDA}$ 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 ST10R272L under two circumstances:

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

the ST10R272L needs access to the shared resources and demands this by activating its $\overline{BREQ}$ output. The arbitration logic may then deactivate the other master's $\overline{HLDA}$ and so free the external bus for the ST10R272L, depending on the priority of the different masters.

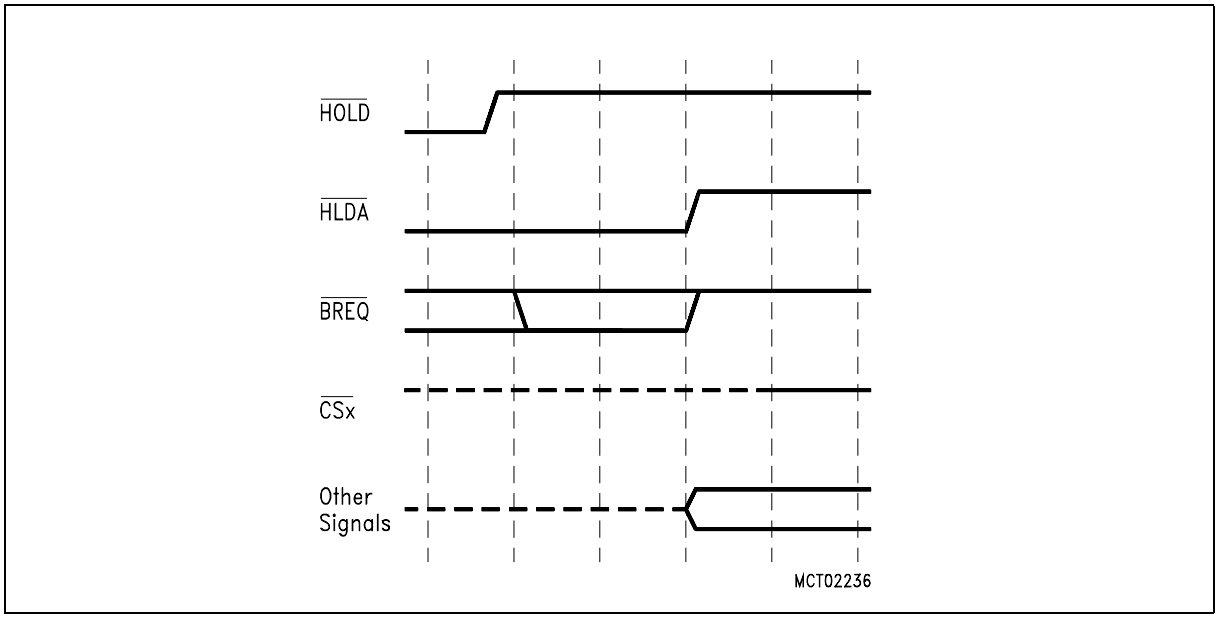


Figure 63 External bus arbitration - regaining the bus

Note *The falling $\overline{BREQ}$ edge shows the last chance for $\overline{BREQ}$ to trigger the indicated regain-sequence. Even if $\overline{BREQ}$ is activated earlier the regain-sequence is initiated by $\overline{HOLD}$ going high. $\overline{BREQ}$ and $\overline{HOLD}$ are connected via an external arbitration circuitry. Please note that $\overline{HOLD}$ may also be deactivated without the ST10R272L requesting the bus.*

9.7 The XBUS interface

The ST10R272L provides an on-chip interface (the XBUS interface), which allows to connect integrated customer/application specific peripherals to the standard controller core. The XBUS is an internal representation of the external bus interface, i.e. it is operated in the same way.

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.

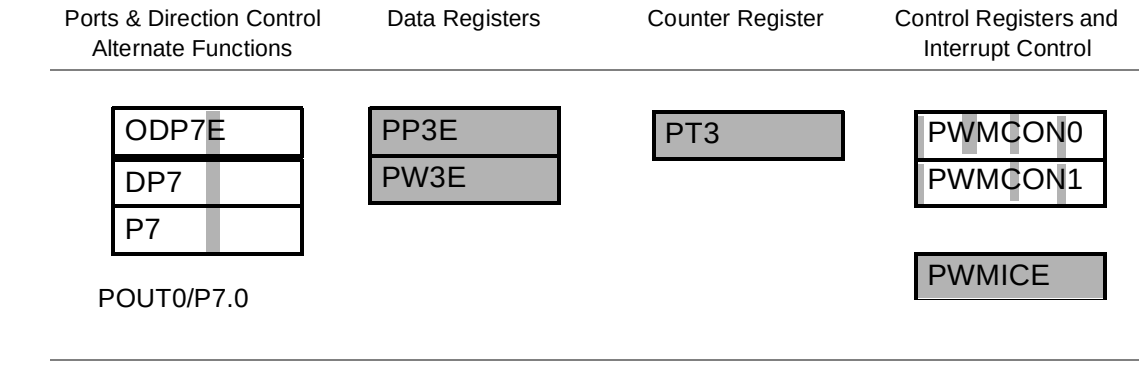
X-Peripheral accesses provide the same choices as external accesses, so these peripherals may be bytewide or wordwide, with or without a separate address bus. Interrupt nodes and configuration pins (on PORT0) are provided for X-Peripherals to be integrated.

Note *If you plan to develop your own peripheral for integration into a ST10R272L device, contact the Technical Support group for the further specification of the XBUS.*

10 PWM MODULE

A 1-channel pulse width modulation (PWM) module is implemented in the ST10R272L. The minimum PWM signal frequency depends on the width (16 bits) and the resolution (CLK/1 or CLK/64) of the PWM timer. The maximum PWM signal frequency assumes that the PWM output signal changes with every cycle of the timer. In a real applications the maximum PWM frequency depends on the required resolution of the PWM output signal. For a list of PWM frequencies refer to Table The following table summarizes the PWM frequencies that result from various combinations of operating mode, counter resolution (input clock) and pulse width resolution..

Figure 0.1 SFRs and port pins associated with PWM unit



ODP7	Port 7Open Drain Control Register	PWMCON0	PWM Control Register 0
DP7	Port 7Direction Control Register	PWCON1	PWM Control Register 1
P7	Port 7 Data Register	PWMIC	PWM Interrupt Control Register
PT3	PWM Channel 3 Timer Register		
PP3	PWM Channel 3 Period Register		
PW3	PWM Channel 3 Pulse Width Register		

The pulse width modulation module operates on channel 3 of the PWM module. This channel has a 16-bit up/down counter PT3, a 16-bit period register PP3, a 16-bit pulse width register PW3 with a shadow latch, two comparators, and the necessary control logic. The

operation of channel 3 is controlled by the PWMCON0 and PWMCON1registers, and the interrupt control and status is handled by the PWMIC interrupt control register.

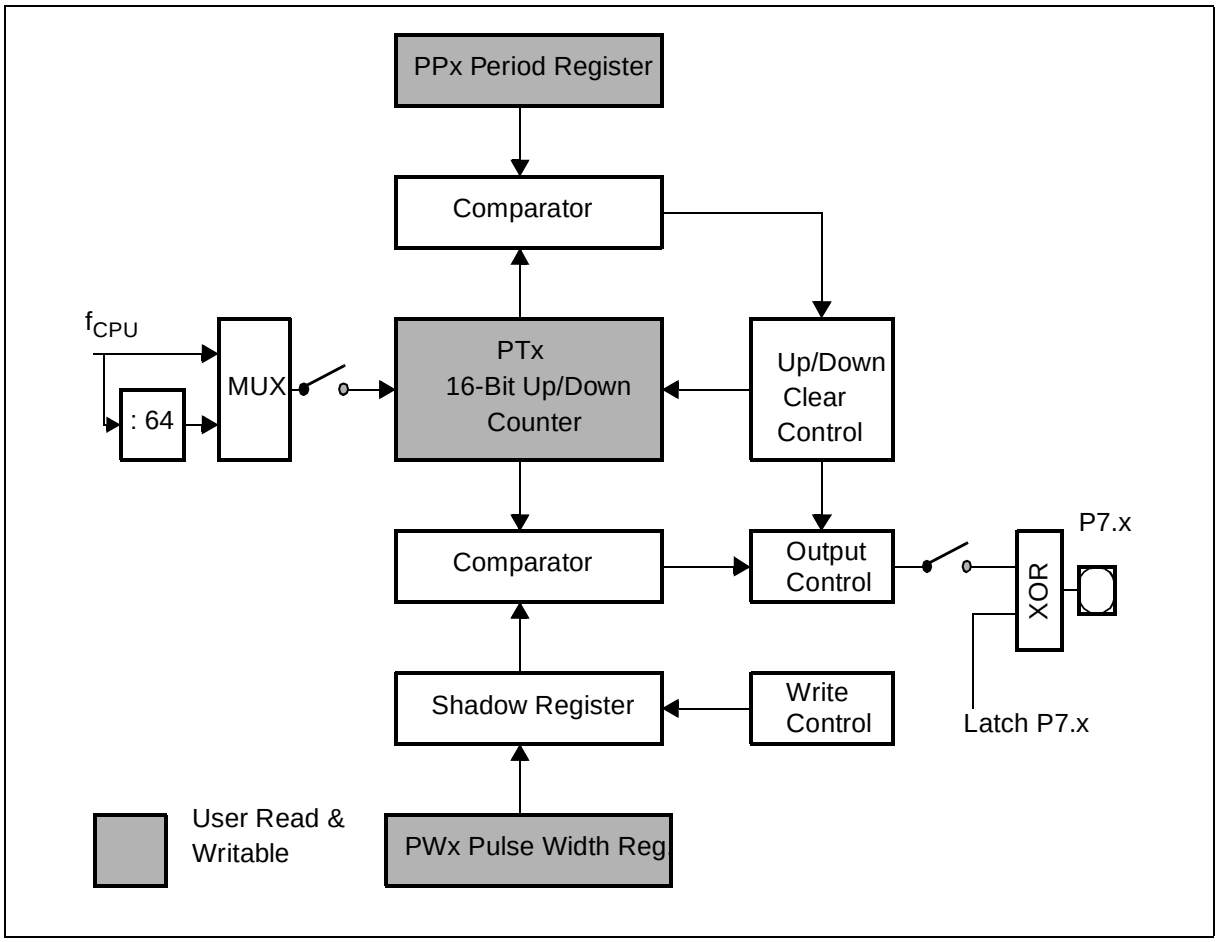


Figure 64 PWM channel block diagram

Note *For the following descriptions the comparison of the timer contents and the PWM value is performed through a 'greater than or equal to' comparison: PWM Output Signal = [PT3] ≥ [PW3]*

10.1 Operating Modes

The PWM module provides three different operating modes:

- Mode 0 - standard PWM generation (edge aligned PWM),
- Mode 1 - symmetrical PWM generation (center aligned PWM),
- Single shot mode.

Note *The output of the PWM module is EXORed with the output of the port output latch (P7.3). After reset, this latch is cleared, so the PWM signal is directly driven to the port pin. By setting the port output latch to '1', the PWM signal is inverted (XORed with '1') before being driven to the port pin. The descriptions below refer to the standard case after reset, i.e. direct driving.*

10.1.1 Mode 0 - standard PWM generation (edge aligned PWM)

Mode 0 is selected by clearing bit PM3 in register PWMCON1 to '0'. In this mode, the PWM timer PT3 is always counting up until it reaches the value in the period register PP3. On the next count pulse, the timer is reset to 0000h and continues counting up with subsequent count pulses. The PWM output signal is switched to a high level when the timer contents are equal to or greater than the contents of the pulse width shadow register. The signal is switched back to a low level when the timer is reset to 0000h, i.e. below the pulse width shadow register. The period of the resulting PWM signal is determine by the value of the PP3 register plus 1, counted in units of the timer resolution.

$$PWM_{Period\ Mode\ 0} = [PP3] + 1$$

The duty cycle of the PWM output signal is controlled by the value in the pulse width shadow register. This mechanism allows to select duty cycles from 0% to 100%, including the boundaries. For a PW3 value of 0000h, the output will remain at a high level, representing a duty cycle of 100%. For a PW3 value higher than the value in the period register PP3, the output will remain at a low level, which corresponds to a duty cycle of 0%.

Figure 65 illustrates the operation and output waveforms of the PWM channel in Mode 0 for different values in the pulse width register PW3. This mode is referred to as Edge Aligned PWM, because the value in the pulse width (shadow) register only affects the positive edge

of the output signal. The negative edge is always fixed and related to the clearing of the timer.

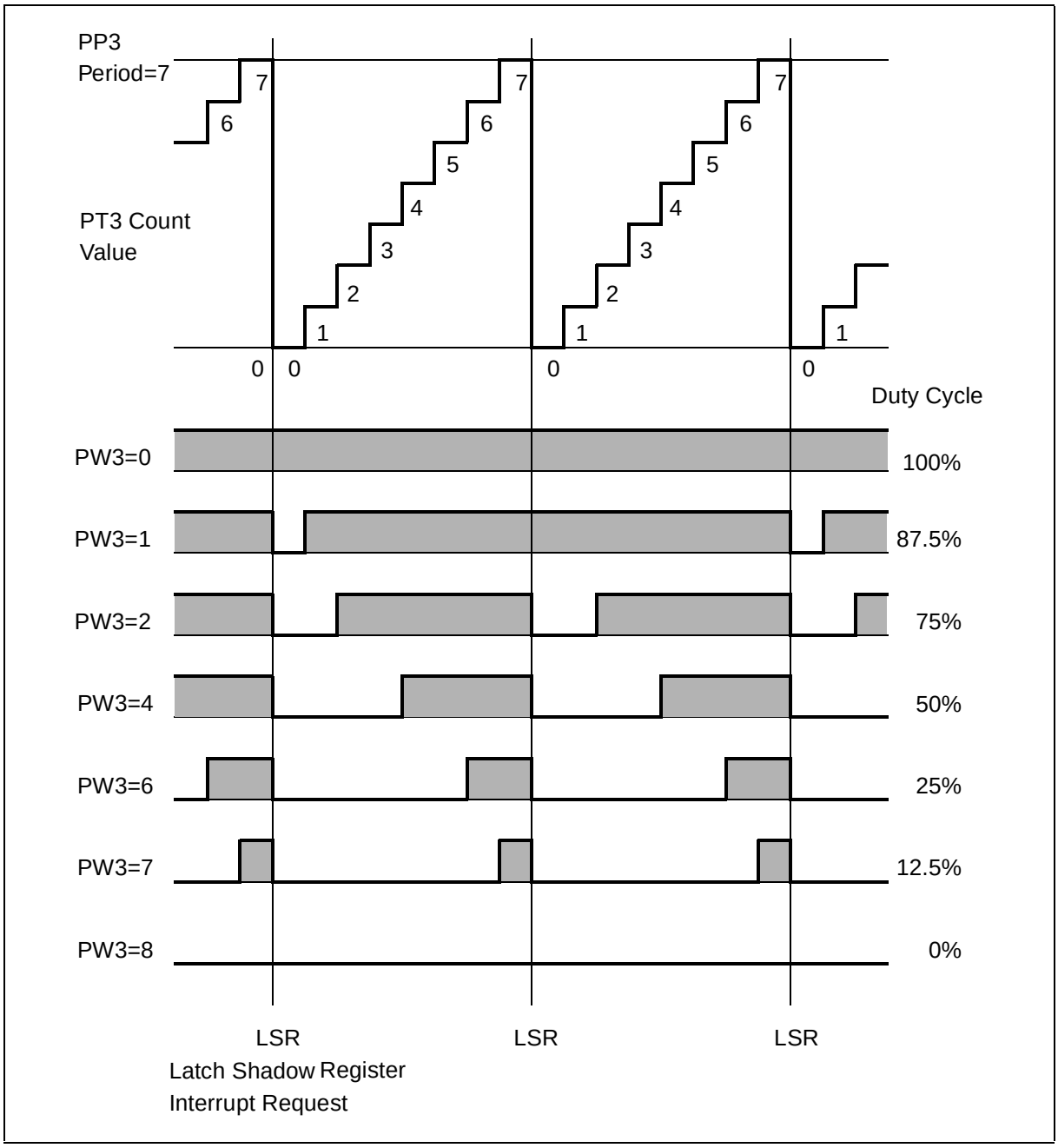


Figure 65 Operation and output waveform in Mode 0

10.1.2 Mode 1 - symmetrical PWM generation (center aligned)

Mode 1 is selected by setting bit PM3 in register PWMCON1 to '1'. In this mode, the PWM timer PT3 is counts up until it reaches the value in the period register PP3. On the next count pulse, the count direction is reversed and the timer starts counting down with subsequent count pulses until it reaches the value 0000h. On the next count pulse, the count direction is reversed again and the count cycle is repeated with the following count pulses.

The PWM output signal is switched to a high level when the timer contents are equal to or greater than the contents of the pulse width shadow register while the timer is counting up. The signal is switched back to a low level when the respective timer has counted down to a value below the contents of the pulse width shadow register. So in mode 1, the value of PW3 register controls both edges of the output signal.

Note that in this mode, the period of the PWM signal is twice the period of the timer:

$$PWM_{Period\ Mode\ 1} = 2 * ([PP3] + 1)$$

Figure 66 illustrates the operation and output waveforms of the PWM channel in Mode 1 for different values in the pulse width register PW3. This mode is referred to as Center Aligned

PWM, because the value in the pulse width (shadow) register affects both edges of the output signal symmetrically.

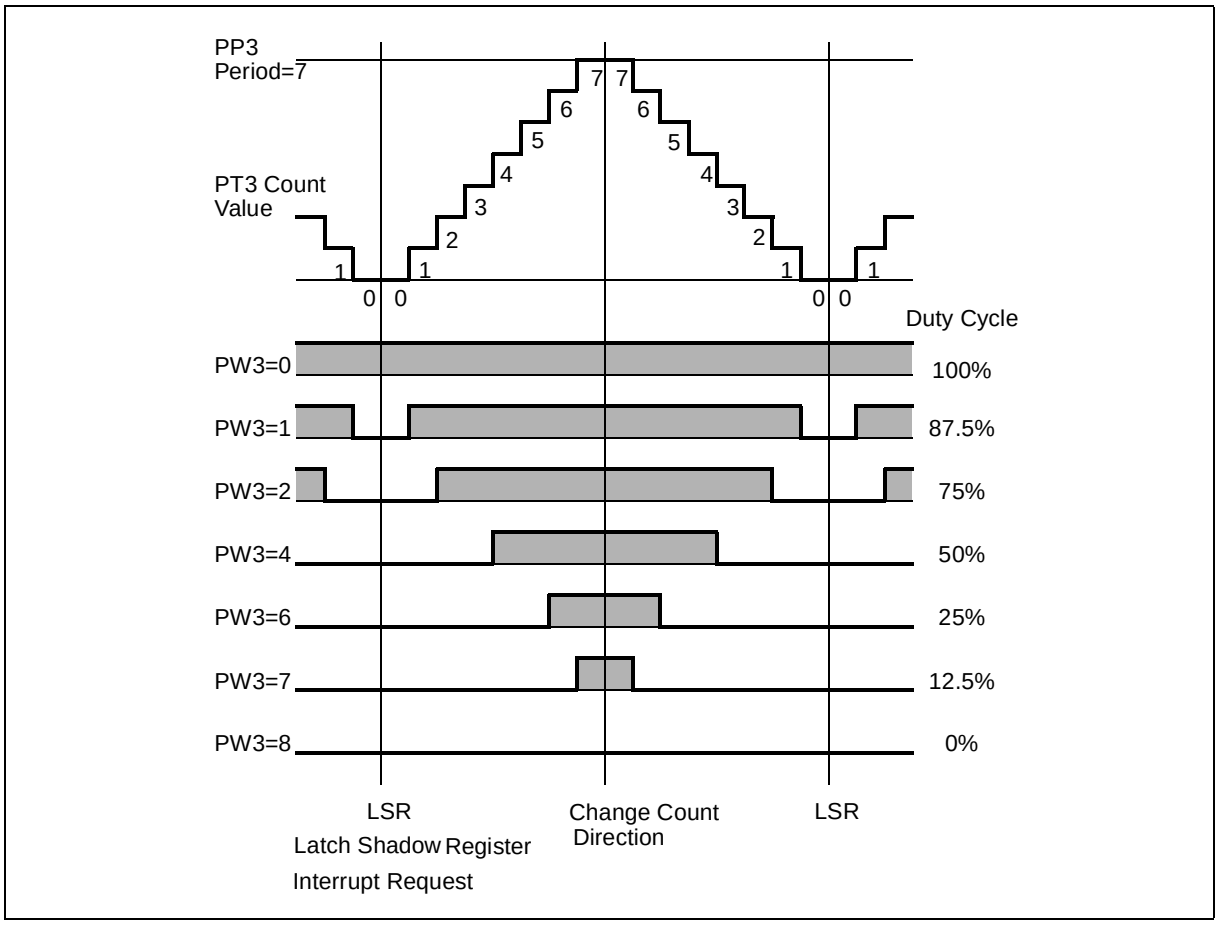


Figure 66 Operation and output waveform in mode 1

10.1.3 Single shot mode

Single shot mode is selected by setting the bit PS3 in register PWMCON1 to ‘1’. In this mode, the PWM timer PT3 is started via software and is counting up until it reaches the value in the period register PP3. Upon the next count pulse, the timer is cleared to 0000h and stopped via hardware, i.e. the PTR3 bit is cleared.

The PWM output signal is switched to a high level when the timer contents are equal to or greater than the contents of the pulse width shadow register. The signal is switched back to a low level when the timer is cleared, i.e. is below the pulse width shadow register. Thus starting the PWM timer in single shot mode produces one single pulse on the port pin, provided that the pulse width value is between 0000h and the period value. In order to generate a further pulse, the timer has to be started again via software by setting bit PTR3.

After starting the timer (i.e. PTR3 = ‘1’), the output pulse may be modified via software. Writing to timer PT3 changes the positive and/or negative edge of the output signal, depending on whether the pulse has already started (i.e. the output is high) or not (i.e. the output is still low). This (multiple) retriggering is always possible while the timer is running, i.e. after the pulse has started and before the timer is stopped.

Loading counter PT3 directly with the value in the PP3 register will abort the current PWM pulse upon the next clock pulse (counter is cleared and stopped by hardware).

By setting the period (PP3), the timer start value (PT3) and the pulse width value (PW3) appropriately, the pulse width (tw) and the optional pulse delay (td) may varied in a wide range.

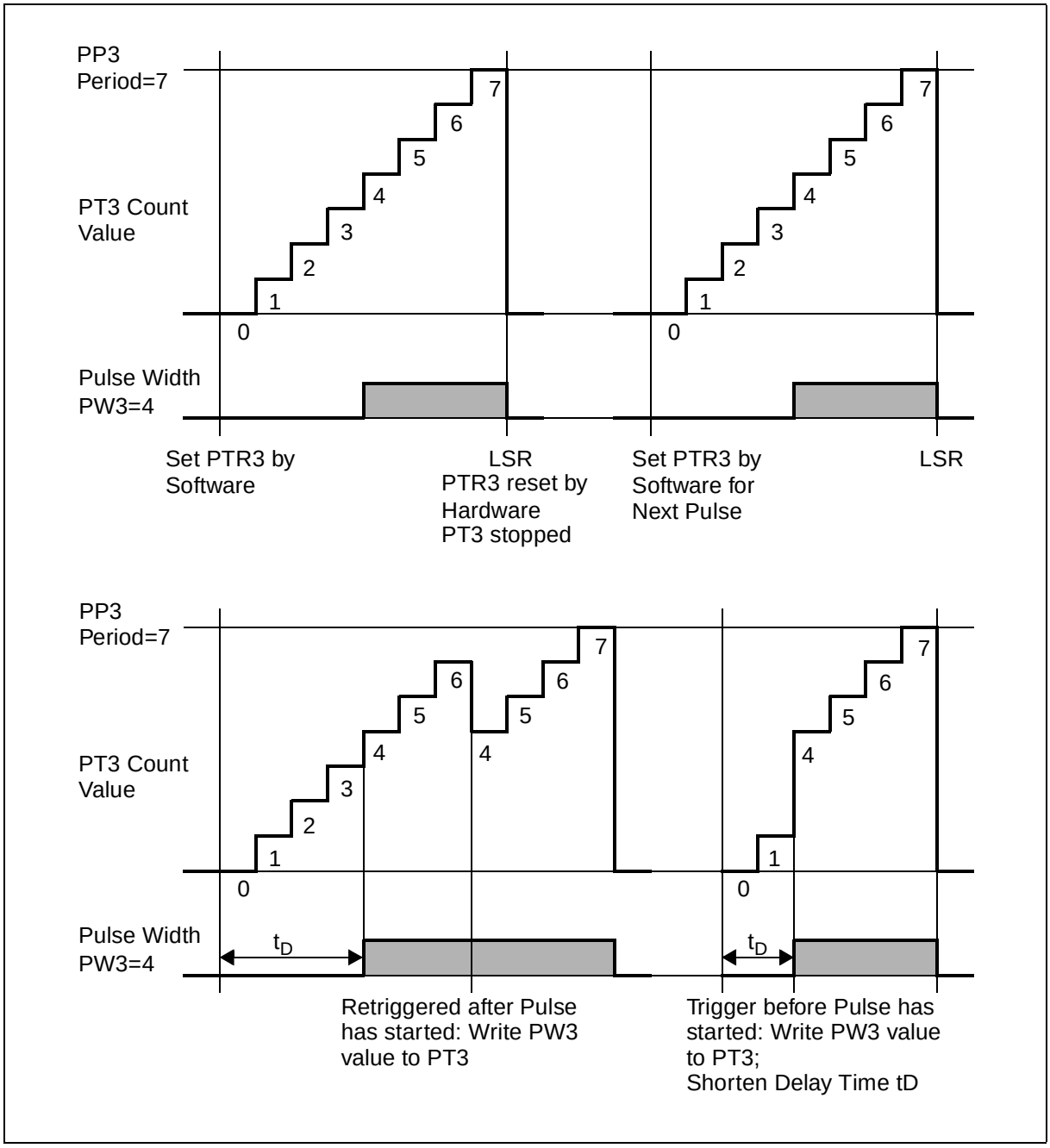


Figure 67 Operation and output waveform in single shot mode

10.2 PWM module registers

The PWM module is controlled by two sets of registers. The waveforms are selected by the timer register PT3, the period register PP3 and the pulse width register PW3. Three registers control the operating mode and the general functions (PWMCON0 and PWMCON1) of the PWM module and the interrupt behavior (PWMIC).

10.2.1 Up/down counter PT3

The counter PT3 of the PWM channel 3 is clocked by either the CPU clock or the CPU clock divided by 64. Bit PTI3 in register PWMCON0 selects the clock source. The PWM counter counts up or down (controlled by hardware), while its run control bit PTR3 is set. A timer is started (PTR3 = '1') via software and is stopped (PTR3 = '0') either by hardware or software, depending on its operating mode. Control bit PTR3 enables or disables the clock input of counter PT3 rather than controlling the PWM output signal.

PT3 (F036h / 1Bh) **ESFR** **Reset Value: 0000h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PT3															
rw															

The following table summarizes the PWM frequencies that result from various combinations of operating mode, counter resolution (input clock) and pulse width resolution.

Input Clock and Mode (Counter resolution)		8-bit PWM resolution	10-bit PWM resolution	12-bit PWM resolution	14-bit PWM resolution	16-bit PWM resolution
f_{CPU}	Mode 0	$f_{cpu}/2^{8}$	$f_{cpu}/2^{10}$	$f_{cpu}/2^{12}$	$f_{cpu}/2^{14}$	$f_{cpu}/2^{16}$
f_{CPU} / 64	Mode 0	$f_{cpu}/64x2^{8}$	$f_{cpu}/64x2^{10}$	$f_{cpu}/64x2^{12}$	$f_{cpu}/64x2^{14}$	$f_{cpu}/64x2^{16}$
f_{CPU}	Mode 1	$f_{cpu}/2x2^{8}$	$f_{cpu}/2x2^{10}$	$f_{cpu}/2x2^{12}$	$f_{cpu}/2x2^{14}$	$f_{cpu}/2x2^{16}$
f_{CPU} / 64	Mode 1	$f_{cpu}/2x64x2^{8}$	$f_{cpu}/2x64x2^{10}$	$f_{cpu}/2x64x2^{12}$	$f_{cpu}/2x64x2^{14}$	$f_{cpu}/2x64x2^{16}$

Table 30 PWM frequency by operating mode, counter and pulse width resolution

10.2.2 Period register PP3

The 16-bit period register PP3 of PWM channel 3 determines the period of a PWM cycle,i.e. the frequency of the PWM signal. The hardware compares the contents of the PP3 register

with the contents of the associated counter PT3. When a match is found between the counter and PP3 register, the counter is either reset to 0000h, or the count direction is switched from counting up to counting down, depending on the selected operating mode of that PWM channel.

PP3 (F03Eh / 1Fh) **ESFR** **Reset Value: 0000h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PP3															
rw															

10.2.3 Pulse width register PW3

This 16-bit register holds the actual PWM pulse width value, which corresponds to the duty cycle of the PWM signal. This register is connected to a 16-bit shadow register. The CPU accesses the PW3 register while the hardware compares the contents of the shadow register with the content of the counter PT3. The shadow register is loaded from the PW3 register at the beginning of every new PWM cycle, or upon a write access to PW3, while the timer is stopped.

When the counter value is greater than or equal to the shadow register value, the PWM signal is set, otherwise it is reset. The output of the comparator may be described by the Boolean formula:

$$PWM_{output\ signal} = ([PT3] \geq [PW3\ shadow\ register]$$

PW3 (F036h / 1Bh) **SFR** **Reset Value: 0000h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PW3															
rw															

10.2.4 PWM control register PWMCON0

This 16-bit control register controls the function of the timers of the PWM channel, and holds the individual interrupt enable and request flags.

PWMCON0 (FF30h / 98h) **SFR** **Reset Value: 0000h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PIR3	-	-	-	PIE3	-	-	-	PT3	-	-	-	PTR3	-	-	-
rw	-	-	-	rw	-	-	-	rw	-	-	-	rw	-	-	-

Bit	Function
PTR3	**PWM Timer PT3 Run Control** ‘0’: Timer PT3 is disconnected from its input clock ‘1’: Timer PT3 is running
PTI3	**PWM Timer PT3 Input Clock Control** ‘0’: Timer PT3 is clocked with CPU clock ‘1’: Timer PT3 is clocked with CPU clock / 64
PIE3	**PWM Channel 3 Individual Interrupt Enable Control** ‘0’: Interrupt from channel 3 disabled ‘1’: Interrupt from channel 3 enabled
PIR3	**PWM Channel 3 Individual Interrupt Request Flag** ‘0’: No request pending from PT3 ‘1’: PT3 has raised an interrupt request

10.2.5 PWM control register PWMCON1

Register PWMCON1 controls the operating mode and the output of the PWM channel. The basic operating mode (standard = edge aligned, or symmetrical = center aligned PWM mode), is selected by the mode bit PM3. Single shot mode is selected by separate control bit PS3. The output signal of the PWM channel is enabled by bit PEN3. If the output is not enabled, the pin can be used for general purpose I/O, and the PWM signal can only be used to generate an interrupt request.

PWMCON1 (FF32h / 99h) **SFR** **Reset Value: 0000h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PS3	-	-	-	-	-	-	-	PM3	-	-	-	PEN3	-	-	-
rw	-	-	-	-	-	-	-	rw	-	-	-	rw	-	-	-

Bit	Function
PEN3	**PWM Channel 3 Output Enable Control** ‘0’: Channel 3 output signal disabled, generate interrupt only ‘1’: Channel 3 output signal enabled
PM3	**PWM Channel 3 Mode Control** ‘0’: Channel 3 operates in mode 0 (edge aligned PWM) ‘1’: Channel 3 operates in mode 1 (center aligned PWM)
PS3	**PWM Channel 3 Single Shot Mode Control Bit** ‘0’: Channel 3 works in respective standard mode ‘1’: Channel 3 operates in single shot mode

10.3 Interrupt request generation

The term ’channel interrupt’ refers to an interrupt request that can be generated by each of the four channels on a multi-channel PWM module. The term ’module interrupt’ refers to the interrupt vector that is assigned to all four channels.

The PWMCON0 register has individual interrupt enable and interrupt request flags for each channel. When the individual enable flag PIEx of a channel is set, then the interrupt request flag PIRx of that channel is set with the same signal that loads the shadow register with the value from register PWx. This indicates that the newest PWM value was transferred to the shadow latch for being compared to the timer contents, and that register PWx is now ’empty’ to receive the next value.

The module interrupt for all channels is controlled by the PWM Module Interrupt Control register PWMIC. This register is organized like any other standard interrupt control register, shown hereafter. If the module interrupt enable bit PWMIE is set, then the interrupt request flag PWMIR is set if any of the channel interrupt request flags PIRx is set (provided this interrupt is enabled through the respective PIEx bit). Software is used to then poll the channel interrupt request flags to determine which channel(s) caused the interrupt.

PWMIC (F17Eh / BFh) **ESFR** **Reset Value: - - 00h**

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

Note *Refer to “Interrupt control registers” on page 86 for an explanation of the control fields.*
The channel interrupt request flags PIR3 (in register PWMCON0) will not be automatically cleared by hardware upon entry into the interrupt service routine, so they must be cleared by software. The module interrupt request flag PWMIR is cleared by hardware upon entry into the interrupt service routine, regardless of how many channel interrupts were active. However, it will be set again if during execution of the service routine a new channel interrupt request is generated.

10.4 PWM output signals

In the ST10R272L, the output signal of the PWM channel 3 (POUT3) is alternate output function on Port 7 pin 3 (P7.3). The output signal of PWM channel 3 is enabled by control bit PEN3 in register PWMCON1.

The PWM signal is EXORed with the respective port latch outputs before being driven to the port pin. This allows to drive the PWM signal directly to the port pin (P7.3=’0’) or drive the inverted PWM signal (P7.3=’1’).

11 GENERAL PURPOSE TIMER UNITS

The GPT unit is a multi-functional timer/counter structure, used for many different time related tasks such as event timing and counting, pulse width and duty cycle measurement, pulse generation and pulse multiplication.
The GPT unit incorporates five 16-bit timers organized into two separate modules, GPT1 and GPT2. Each timer in each module can operate independently in a number of different modes, or can be concatenated with another timer of the same module.

11.1 Timer block GPT1

For programming, the GPT1 block is composed of a set of SFRs as summarized below. The portions of port and direction registers which are used by the GPT1for alternate functions, are shaded.

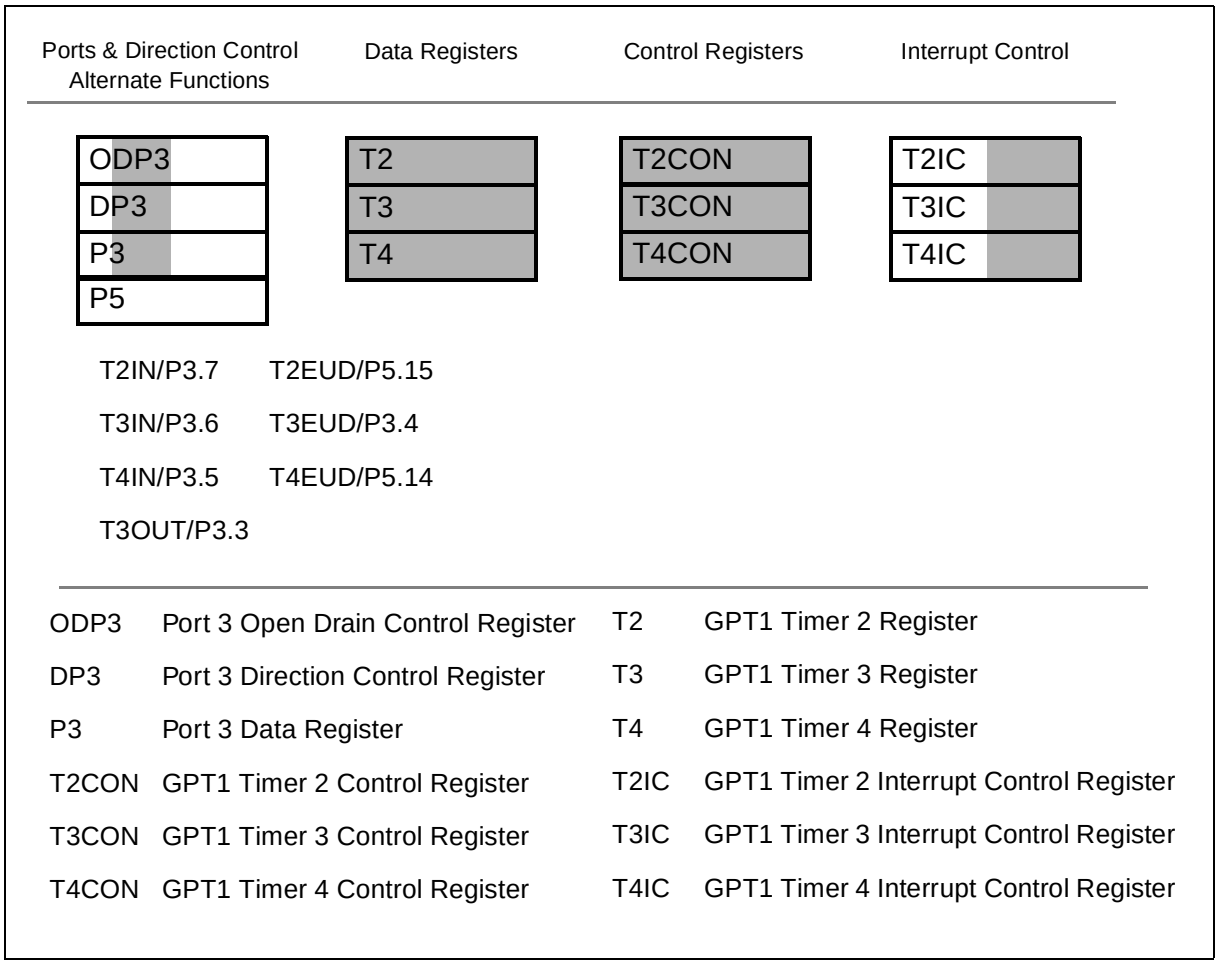


Figure 68 SFRs and port pins associated with timer block GPT1

All three timers of block GPT1 (T2, T3, T4) can run in 3 basic modes: timer, gated timer, and counter mode, and all timers can count either up or down. Each timer has an associated alternate input function pin on Port 3, which serves as the gate control in gated timer mode, or as the count input in counter mode. The count direction (Up / Down) can be programmed by software or can be dynamically altered by a signal at an external control-input pin. Each overflow/underflow of core timer T3 can be indicated on an alternate output function pin. The auxiliary timers T2 and T4 can, additionally, be concatenated with the core timer, or used as capture or reload registers for the core timer.

In incremental interface mode, the GPT1 timers (T2, T3, T4) can be directly connected to the incremental position sensor signals A and B by their respective inputs TxIN and TxEUD. Direction and count signals are internally derived from these two input signals - so the contents of the respective timer Tx corresponds to the sensor position. The third position sensor signal TOP0 can be connected to an interrupt input.

The current contents of each timer can be read or modified by the CPU by accessing the corresponding timer registers T2, T3, or T4 located in the non-bitaddressable SFR space. When any of the timer registers is written to by the CPU in the state immediately before a timer increment, decrement, reload, or capture, the CPU write operation has priority. This is to guarantee correct results.

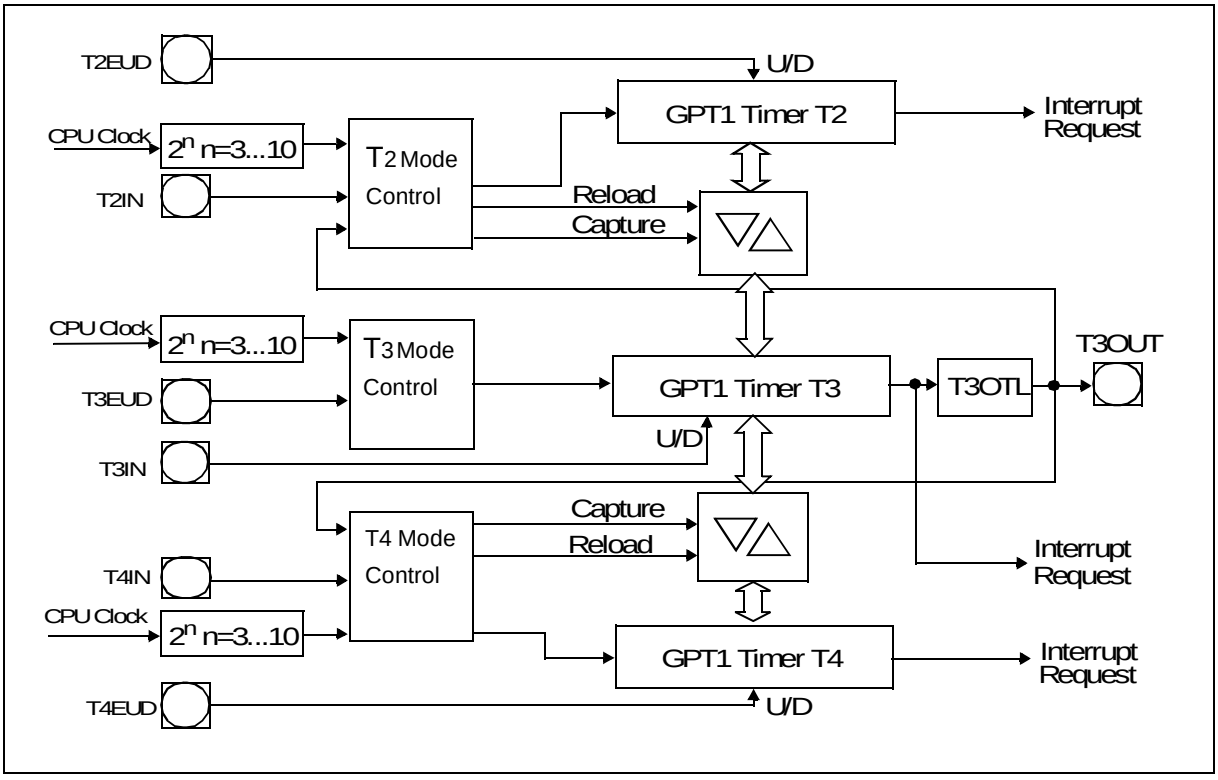


Figure 69 GPT1 block diagram

11.1.1 GPT1 core timer T3

The core timer T3 is configured and controlled by its bit addressable control register T3CON.

T3CON (FF42h / A1h) **SFR** **Reset Value: 0000h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	T3 OTL	T3OE	T3 UDE	T3UD	T3R	T3M			T3I		
-	-	-	-	-	rw	rw	rw	rw	rw	rw			rw		

Bit	Function
T3I	**Timer 3 Input Selection** Depends on the operating mode, see respective sections.
T3M	**Timer 3 Mode Control (Basic Operating Mode)** 0 0 0: Timer Mode 0 0 1: Counter Mode 0 1 0: Gated Timer with Gate active low 0 1 1: Gated Timer with Gate active high 1 X X: Reserved. Do not use this combination.
T3R	**Timer 3 Run Bit** T3R = '0': Timer / Counter 3 stops T3R = '1': Timer / Counter 3 runs
T3UD	**Timer 3 Up / Down Control** [1]
T3UDE	**Timer 3 External Up/Down Enable** [1]
T3OE	**Alternate Output Function Enable** T3OE ='0' Alternate Output Function Disable T3OE = '1': Alternate Output Function Enabled
T3OTL	**Timer 3 Output Toggle Latch** Toggles on each overflow / underflow of T3. Can be set or reset by software.

1. For bits T3UD and T3UDE refer to Table 31 GPT1 core timer T3 count direction control below.

Timer 3 run bit

The timer can be started or stopped by software through bit T3R (Timer T3 Run Bit). If T3R='0', the timer stops, T3R= '1' starts the timer.
In gated timer mode, the timer will only run if T3R='1' and the gate is active (high or low, as programmed).

Count direction control

The count direction of the core timer can be controlled either by software or by the external input pin T3EUD the alternate input function of port pin P3.4. Bits T3UD and T3UDE in control register T3CON set the count direction. When the count direction is controlled by software (bit T3UDE='0'), it can be altered by setting or clearing bit T3UD. When T3UDE='1', pin T3EUD is count direction controller. However, bit T3UD can still be used to reverse the actual count direction as shown in the table below. The count direction can be changed regardless of whether the timer is running or not.

When pin T3EUD/P3.4 is used as external count direction control input, it must be configured as input, i.e. its corresponding direction control bit DP3.4 must be set to '0'.

Pin TxEUD	Bit TxUDE	Bit TxUD	Count Direction
X	0	0	Count Up
X	0	1	Count Down
0	1	0	Count Up
1	1	0	Count Down
0	1	1	Count Down
1	1	1	Count Up

Table 31 GPT1 core timer T3 count direction control

Note *The direction control works the same for core timer T3 and for auxiliary timers T2 and T4. Therefore the pins and bits are named Tx...*

Timer 3 output toggle latch

An overflow or underflow of timer T3 clocks the toggle bit T3OTL in the T3CON control register. T3OTL can also be set or reset by software. Bit T3OE (Alternate Output Function Enable) in the T3CON register enables the state of T3OTL to be an alternate function of the external output pin T3OUT/P3.3. For that purpose, a '1' must be written into port data latch P3.3, and pin T3OUT/P3.3 must be configured as an output by setting direction control bit

DP3.3 to '1'. If T3OE='1', pin T3OUT outputs the state of T3OTL. If T3OE='0', pin T3OUT can be used as general purpose IO pin.

T3OTL can also be used in conjunction with the timer over/underflows as an input for the counter function or as a trigger source for the reload function of the auxiliary timers T2 and T4. For this purpose, the state of T3OTL does not have to be available at pin T3OUT, because an internal connection is provided for this option.

Timer 3 in timer mode

Timer mode for the core timer T3 is selected by setting bit field T3M in the T3CON register to '000_B'. In this mode, T3 is clocked with the internal system clock (CPU clock) divided by a programmable pre-scaler selected by bit field T3I. The input frequency f_{T3} for timer T3 and its resolution r_{T3} are scaled linearly with lower clock frequencies f_{CPU}, as can be seen from the following formula:

$$f_{T3} = f_{CPU} / 8 \times 2^{\langle T3I \rangle} \qquad r_{T3[\mu S]} = 8 \times 2^{\langle T3I \rangle} / f_{CPU}[MHz]$$

The timer resolutions which result from the selected pre-scaler option are listed in the table below. This table also applies to the Gated Timer Mode of T3 and to the auxiliary timers T2 and T4 in timer and gated timer mode.

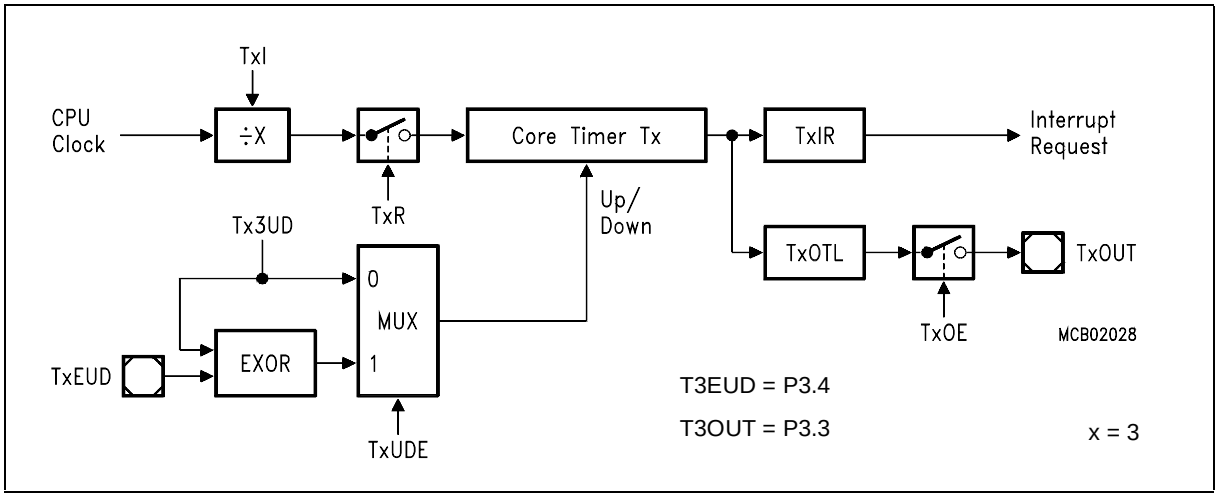


Figure 70 Block diagram of core timer T3 in timer mode

	Timer Input Selection T2I / T3I / T4I							
	000_B	001_B	010_B	011_B	100_B	101_B	110_B	111_B

Table 32 GPT1 timer resolutions

	Timer Input Selection T2I / T3I / T4I							
Pre-scaler factor	8	16	32	64	128	256	512	1024
Resolution in CPU clock cycles	8	16	32	64	128	256	512	1024

Table 32 GPT1 timer resolutions

Timer 3 in gated timer mode

Gated timer mode for the core timer T3 is selected by setting bit field T3M in the T3CON register to '010_B' or '011_B'. Bit T3M.0 (T3CON.3) selects the active level of the gate input. In gated timer mode the same options for the input frequency as for the timer mode are available. However, the input clock to the timer in this mode is gated by the external input pin T3IN (Timer T3 External Input), which is an alternate function of P3.6.
To enable this operation pin T3IN/P3.6 must be configured as input, i.e. direction control bit DP3.6 must contain '0'.

If T3M.0='0', the timer is enabled when T3IN shows a low level. A high level at this pin stops the timer. If T3M.0='1', pin T3IN must have a high level in order to enable the timer. In addition, the timer can be turned on or off by software using bit T3R. The timer will only run, if T3R='1' and the gate is active. It will stop, if either T3R='0' or the gate is inactive.

Note A transition of the gate signal at pin T3IN does not cause an interrupt request.

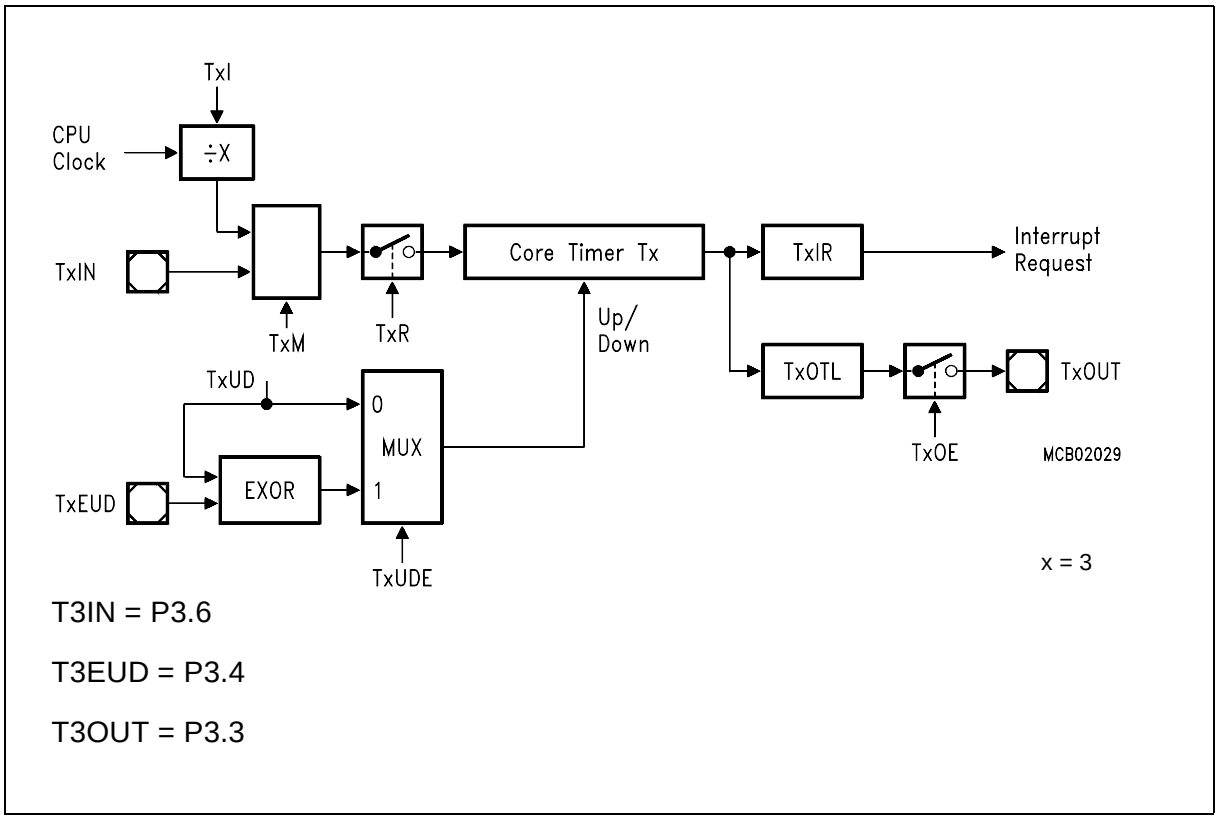


Figure 71 Block diagram of core timer T3 in gated timer mode

Timer 3 in counter mode

Counter mode for the core timer T3 is selected by setting bit field T3M in the T3CON register 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 pin. Bit field T3I in control the T3CON register selects the triggering transition (see table below).

For counter operation, pin T3IN/P3.6 must be configured as input, i.e. direction control bit DP3.6 must be '0'. The maximum input frequency which is allowed in counter mode is f_{CPU}/ 16. To ensure that a transition of the count input signal which is applied to T3IN is correctly recognized, its level should be held high or low for at least 8 CPU clock cycles before it changes.

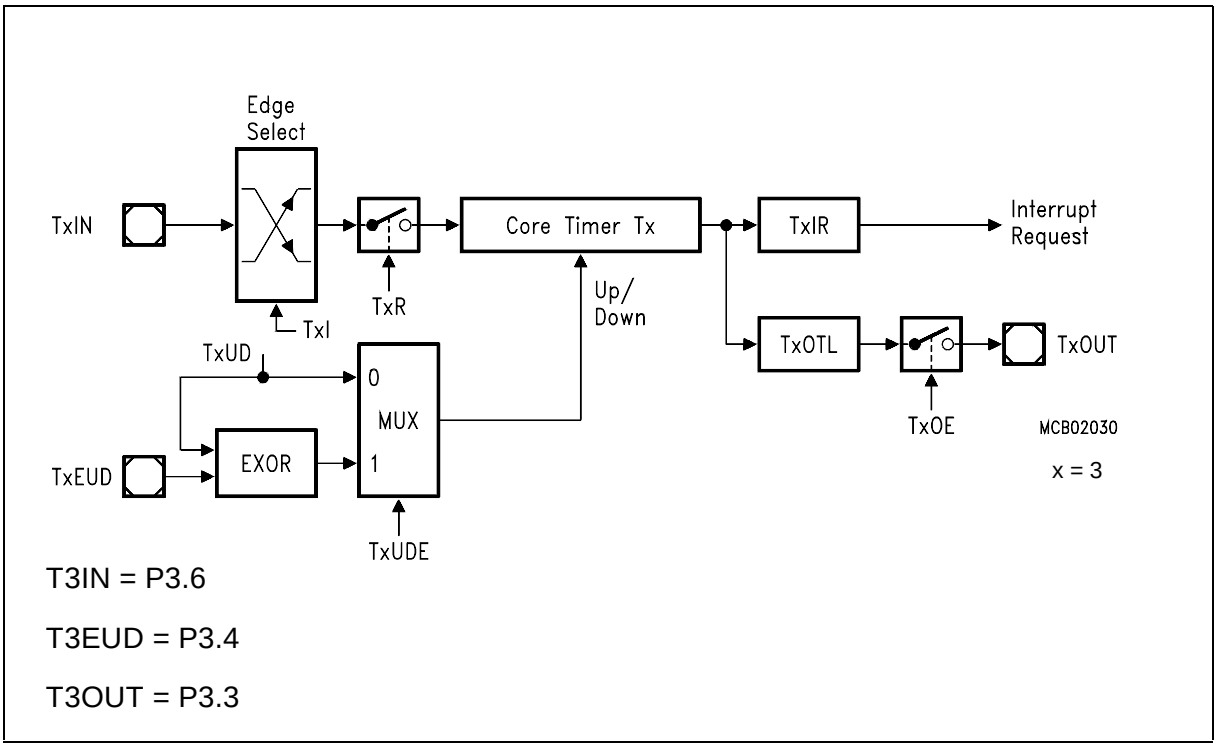


Figure 72 Block diagram of core timer T3 in counter mode

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

Table 33 GPT1 core timer T3 (counter mode) input edge selection

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 ‘110B’. In incremental interface mode the two inputs associated with timer T3 (T3IN T3EUD) are used to interface to an incremental encoder. T3 is clocked by

each transition on one or both of the external input pins which gives 2-fold or 4-fold resolution to the encoder input.

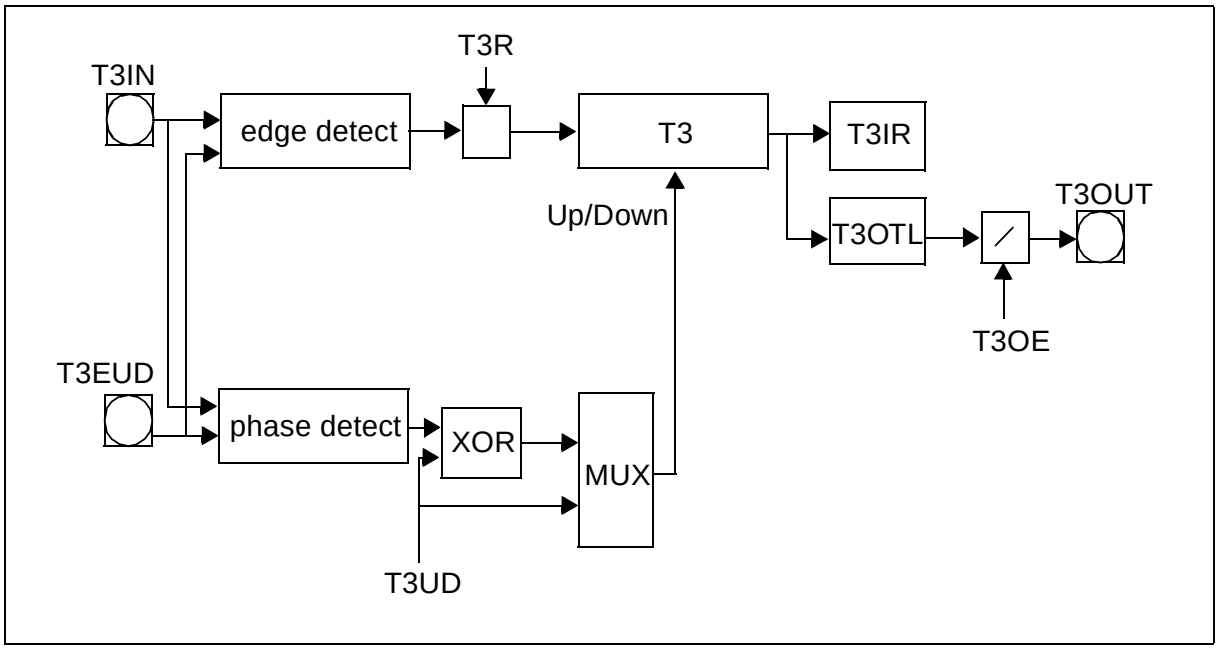


Figure 73 Core timer T3 in incremental interface mode

Bitfield T3I in the T3CON control register selects the triggering transitions (see table below). In this mode, the sequence of the transitions of the two input signals is evaluated, and generates count pulses as well as the direction signal. So, T3 is modified automatically according to the speed and the direction of the incremental encoder, its contents therefore always represent the encoder's current position.

T3I	Triggering Edge for Counter Increment/Decrement
000	None. Counter stops
001	Any transition (rising or falling edge) on T3IN
010	Any transition (rising or falling edge) on T3EUD
011	Any transition (rising or falling edge) on T3 input (T3IN or T3EUD)
1XX	Reserved. Do not use this combination

Table 34 GPT1 core timer T3 (incremental interface mode) input edge selection

The incremental encoder can be connected directly to the ST10R272L without external interface logic. However, in a standard system comparators are employed to convert the encoder's differential outputs (e.g. A, $\overline{A}$) to digital signals (e.g. A). this greatly increases noise immunity.

Note *The third encoder output Top0 which indicates the mechanical zero position, may be connected to an external interrupt input and trigger a reset timer T3 (e.g.via PEC transfer from ZEROS).*

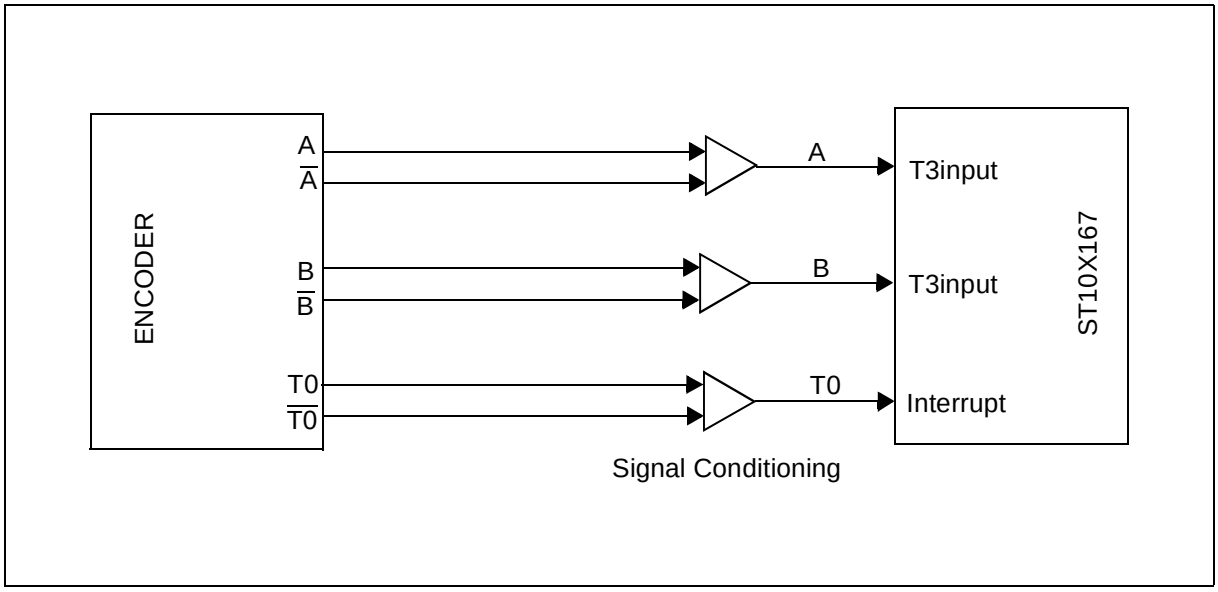


Figure 74 Encoder - ST10R272L connection

For incremental interface operation the following conditions must be met

- Bitfield T3M must be '110_B'
- Both pins T3IN and T3EUD must be configured as input, i.e. the respective direction control bits must be '0'.
- Bit T3EUD must be '1' to enable automatic direction control.

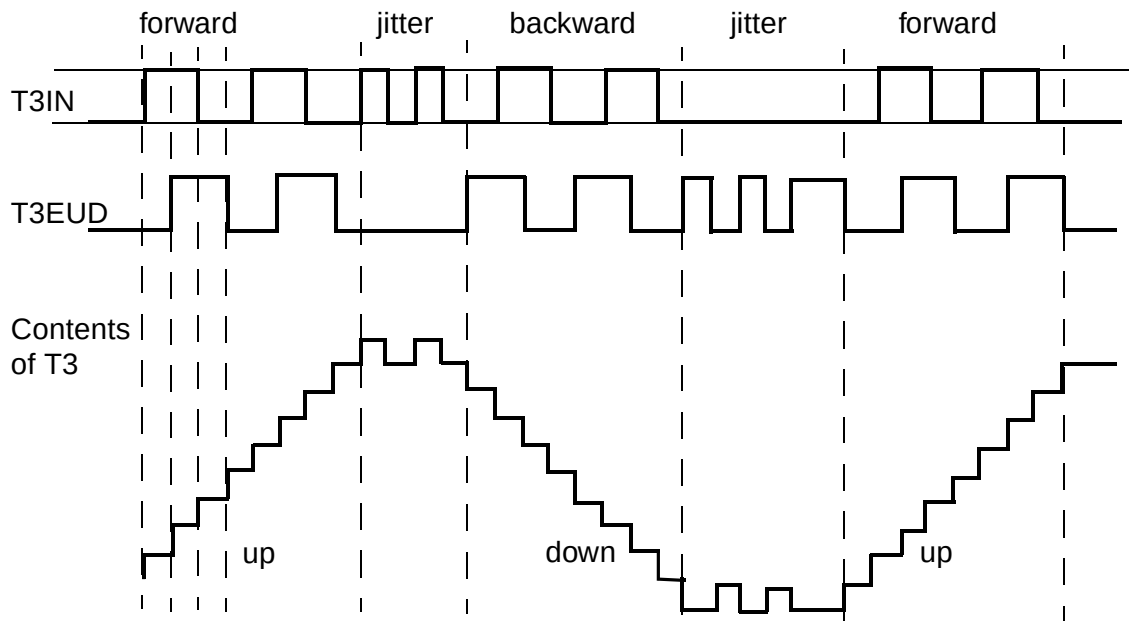
The maximum allowed input frequency in incremental interface mode is $f_{CPU}/16$. To ensure correct recognition of the transition of any input signal, its level should be held high or low for at least 8 CPU clock cycles.

In incremental interface mode, the count direction is automatically derived from the sequence in which the input signals change. This corresponds to the rotation direction of the connected sensor. The table below summarizes the possible combinations.

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

Table 35 GPT1 core Timer T3 (incremental interface mode) count direction

The figures below show of T3's operation, visualizing count signal generation, direction control and input jitter is compensation. Input jitter is compensation might occur if the sensor rests near to one of the 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 75 Evaluation of the incremental encoder signals

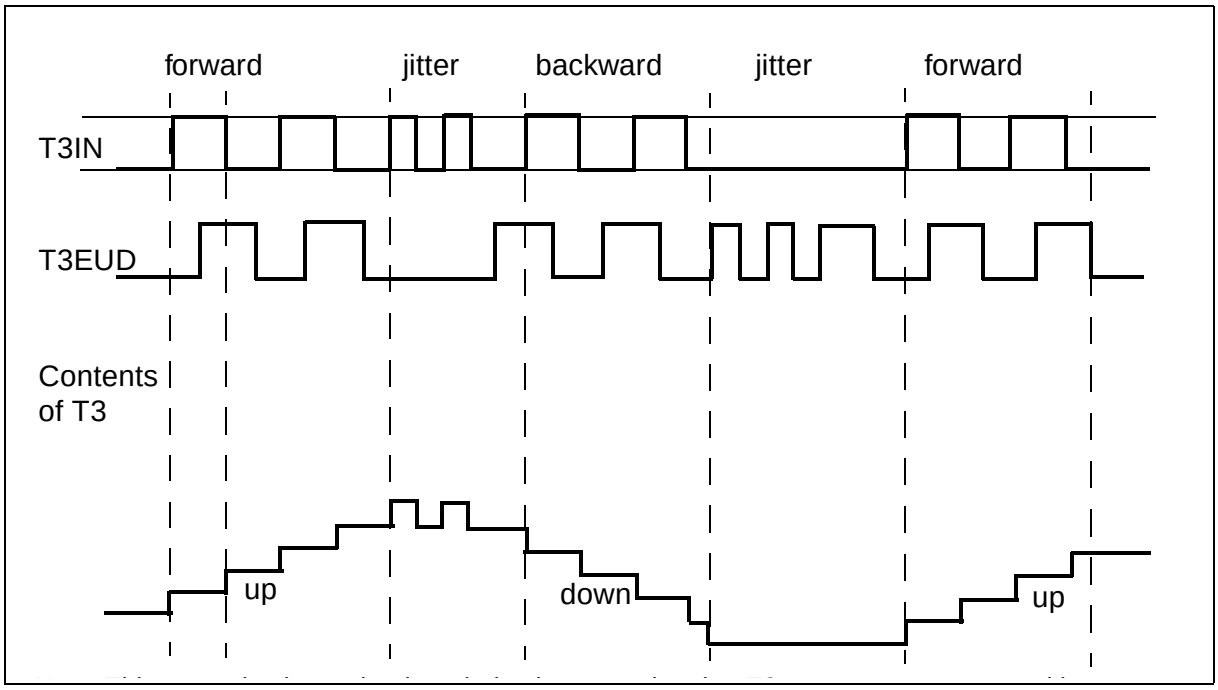


Figure 76 Evaluation of the incremental encoder signals

Note *Timer 3 operating in incremental interface mode, automatically provides information on the sensor's current position. Dynamic information (speed, acceleration, deceleration) may be obtained by measuring the incoming signal periods. This is facilitated by a special capture mode for timer T5.*

11.1.2 GPT1 auxiliary timers T2 and T4

Both auxiliary timers T2 and T4 have exactly the same functionality. They can be configured for timer, gated timer, or counter mode with the same options for the timer frequencies and the count signal as the core timer T3. In addition to these 3 counting modes, the auxiliary timers can be concatenated with the core timer, or they may be used as reload or capture registers in conjunction with the core timer.

Note *The auxiliary timers have no output toggle latch and no alternate output function.*

The individual configuration for timers T2 and T4 is determined by their bit addressable control registers T2CON and T4CON, which are both organized identically. Note that

functions which are present in all 3 timers of block GPT1 are controlled in the same bit positions and in the same manner in each of the specific control registers.

T2CON (FF40h / A0h) **SFR** **Reset Value: 0000h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	T2 UDE	T2UD	T2R	T2M			T2I		
-	-	-	-	-	-	-	rw	rw	rw	rw			rw		

T4CON (FF44h / A2h) **SFR** **Reset Value: 0000h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	T4 UDE	T4UD	T4R	T4M			T4I		
-	-	-	-	-	-	-	rw	rw	rw	rw			rw		

Bit	Function
TxI	**Timer x Input Selection** Depends on the Operating Mode.
TxM	**Timer x Mode Control (Basic Operating Mode)** 0 0 0: Timer Mode 0 0 1: Counter Mode 0 1 0: Gated Timer with Gate active low 0 1 1: Gated Timer with Gate active high 1 0 0: Reload Mode 1 0 1: Capture Mode 1 1 0: Incremental interface mode 1 1 X: Reserved. Do not use this combination
TxR	**Timer x Run Bit** TxR = '0': Timer / Counter x stops TxR = '1': Timer / Counter x runs
TxUD	**Timer x Up / Down Control**[1]

Bit	Function
TxUDE	Timer x External Up/Down Enable [1]

1. For the effects of bits TxUD and TxUDE refer to Table 31 GPT1 core timer T3 count direction control.

Count direction control for auxiliary timers

The count direction of the auxiliary timers can be controlled in the same way as for the core timer T3. The description and the table apply accordingly.

Timers T2 and T4 in timer mode or gated timer mode

When the auxiliary timers T2 and T4 are programmed to timer mode or gated timer mode, their operation is the same as described for the core timer T3. The descriptions, figures and tables apply accordingly with one exception: There is no output toggle latch and no alternate output pin for T2 and T4.

Timers T2 and T4 in counter mode

Counter mode for the auxiliary timers T2 and T4 is selected by setting bit field TxM in the respective TxCON register to '001_B'. In counter mode, timers T2 and T4 can be clocked either by a transition at the respective external input pin TxIN, or by a transition of timer T3's output toggle latch T3OTL.

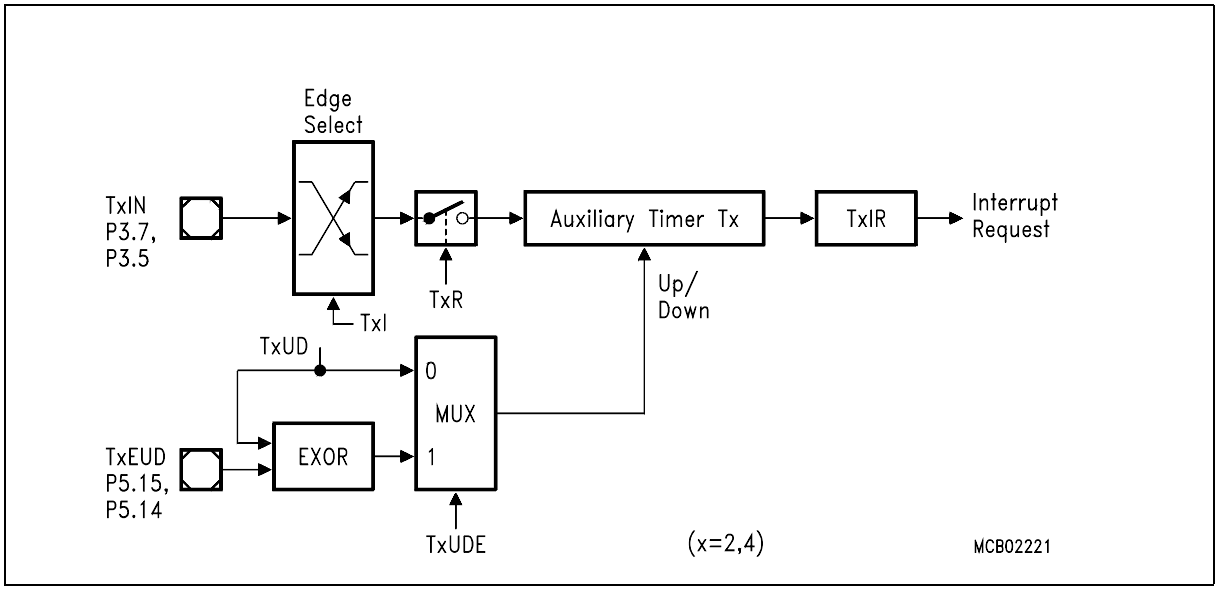


Figure 77 Block diagram of an auxiliary timer in counter mode

The event causing an increment or decrement of a timer can be a positive, a negative, or both a positive and a negative transition at either the respective input pin, or at the toggle latch T3OTL. Bit field TX in the respective control register TxCON selects the triggering transition (see table below).

T2I / T4I	Triggering Edge for Counter Increment / Decrement
X 0 0	None. Counter Tx is disabled
0 0 1	Positive transition (rising edge) on TxIN
0 1 0	Negative transition (falling edge) on TxIN
0 1 1	Any transition (rising or falling edge) on TxIN
1 0 1	Positive transition (rising edge) of output toggle latch T3OTL
1 1 0	Negative transition (falling edge) of output toggle latch T3OTL
1 1 1	Any transition (rising or falling edge) of output toggle latch T3OTL

Table 36 GPT1 auxiliary timer (counter mode) input edge selection

Note *Only state transitions of T3OTL which are caused by the overflows/underflows of T3 will trigger the counter function of T2/T4. Modifications of T3OTL via software will NOT trigger the counter function of T2/T4.*

For counter operation, pin TxIN must be configured as input, i.e. the respective direction control bit must be '0'. The maximum input frequency which is allowed in counter mode is $f_{CPU}/8$. To ensure that a transition of the count input signal which is applied to TxIN is correctly recognized, its level should be held for at least 8 CPU clock cycles before it changes.

Timer concatenation

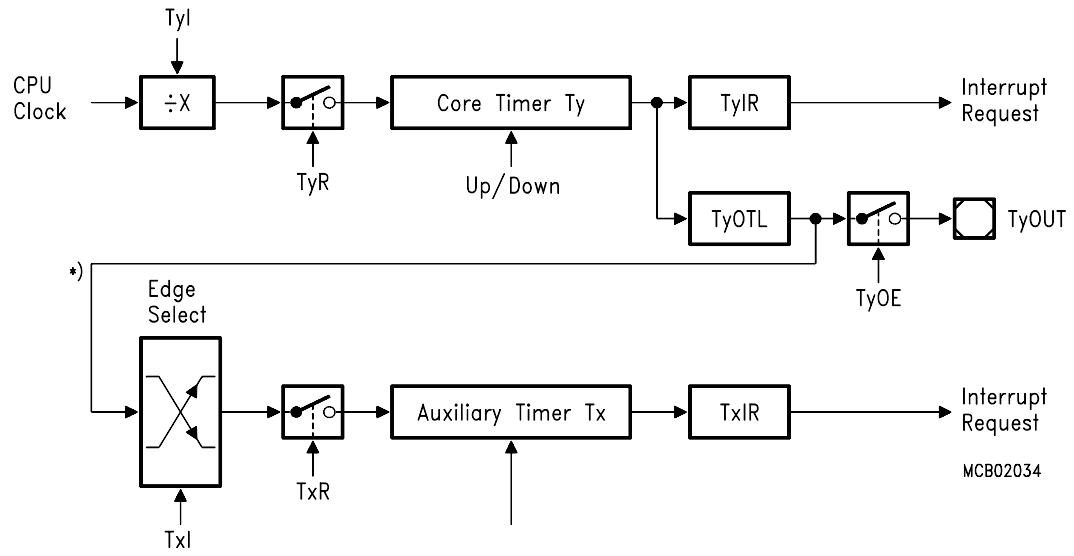
Using the toggle bit T3OTL as a clock source for an auxiliary timer in counter mode, concatenates the core timer T3 with the respective auxiliary timer. Depending on which transition of T3OTL is selected to clock the auxiliary timer, this concatenation forms a 32-bit or a 33-bit timer/counter.

- **32-bit timer/counter**: If both a positive and a negative transition of T3OTL is used to clock the auxiliary timer, this timer is clocked on every overflow/underflow of the core timer T3. Thus, the two timers form a 32-bit timer.
- **33-bit timer/counter**: If either a positive or a negative transition of T3OTL is selected to clock the auxiliary timer, this timer is clocked on every second overflow/underflow of the

core timer T3. This configuration forms a 33-bit timer (16-bit core timer+T3OTL+16-bit auxiliary timer).

The count directions of the two concatenated timers are not required to be the same. This offers a wide variety of different configurations.
T3 can operate in timer, gated timer or counter mode in this case.

.



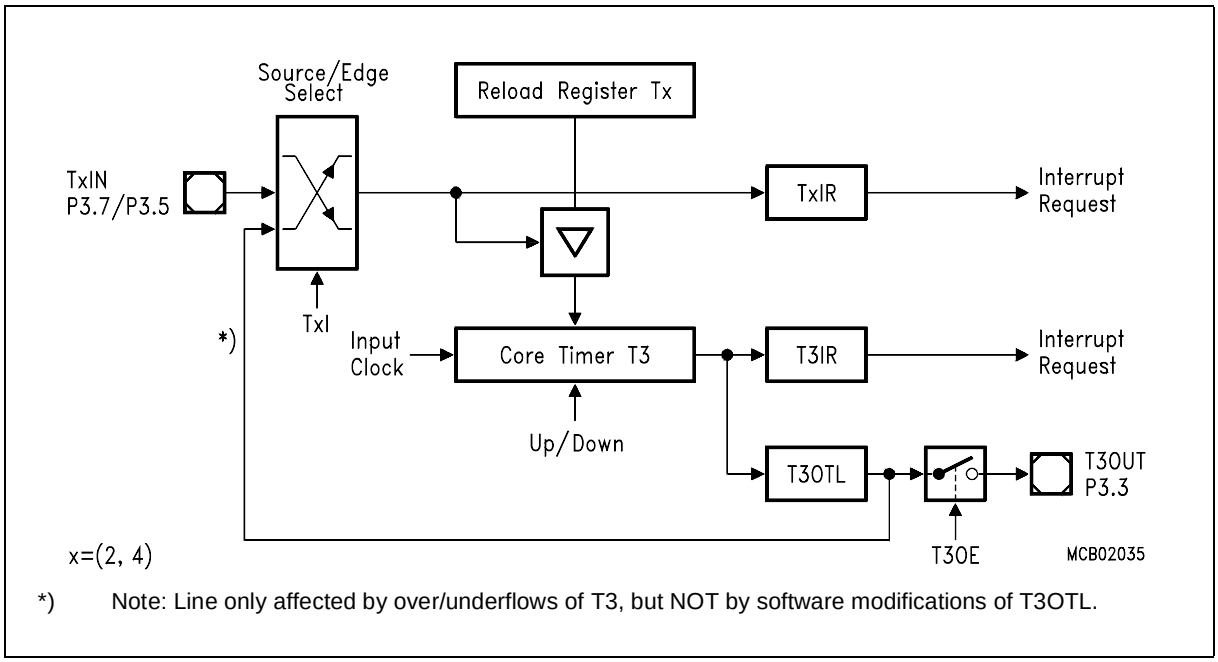
*) Note: Line only affected by over/underflows of T3, but NOT by software modifications of T3OTL.

Figure 78 Concatenation of core timer T3 and an auxiliary timer

Auxiliary timer in reload mode

Reload mode for the auxiliary timers T2 and T4 is selected by setting bit field TxM in the respective register TxCON to '100_B'. In reload mode the core timer T3 is reloaded with the contents of an auxiliary timer register, triggered by one of two different signals. The trigger signal is selected the same way as the clock source for counter mode (see table above), i.e. a transition of the auxiliary timer's input or the output toggle latch T3OTL may trigger the reload.

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



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

Figure 79 GPT1 auxiliary timer in reload mode

When a trigger signal T3 is loaded with the contents of the respective timer register (T2 or T4) and the interrupt request flag (T2IR or T4IR) is set.

Note *When a T3OTL transition is selected for the trigger signal, also the interrupt request flag T3IR will be set upon a trigger, indicating T3's overflow or underflow. Modifications of T3OTL via software will NOT trigger the counter function of T2/T4.*

The reload mode triggered by T3OTL can be used in a number of different configurations. Depending on the selected active transition the following functions can be performed:

- If both a positive and a negative transition of T3OTL is selected to trigger a reload, the core timer will be reloaded with the contents of the auxiliary timer each time it overflows or underflows. This is the standard reload mode (reload on overflow/underflow).
- If either a positive or a negative transition of T3OTL is selected to trigger a reload, the core timer will be reloaded with the contents of the auxiliary timer on every second overflow or underflow.
- Using this "single-transition" mode for both auxiliary timers allows to perform very flexible pulse width modulation (PWM). One of the auxiliary timers is programmed to reload the core timer on a positive transition of T3OTL, the other is programmed for a reload on a

negative transition of T3OTL. With this combination the core timer is alternately reloaded from the two auxiliary timers.

The figure below shows an example for the generation of a PWM signal using the alternate reload mechanism. T2 defines the high time of the PWM signal (reloaded on positive transitions) and T4 defines the low time of the PWM signal (reloaded on negative transitions). The PWM signal can be output on T3OUT with T3OE='1', P3.3='1' and DP3.3='1'. With this method the high and low time of the PWM signal can be varied in a wide range.

Note *The output toggle latch T3OTL is software accessible and may be changed, if required, to modify the PWM signal. However, this will NOT trigger the reload of T3.*

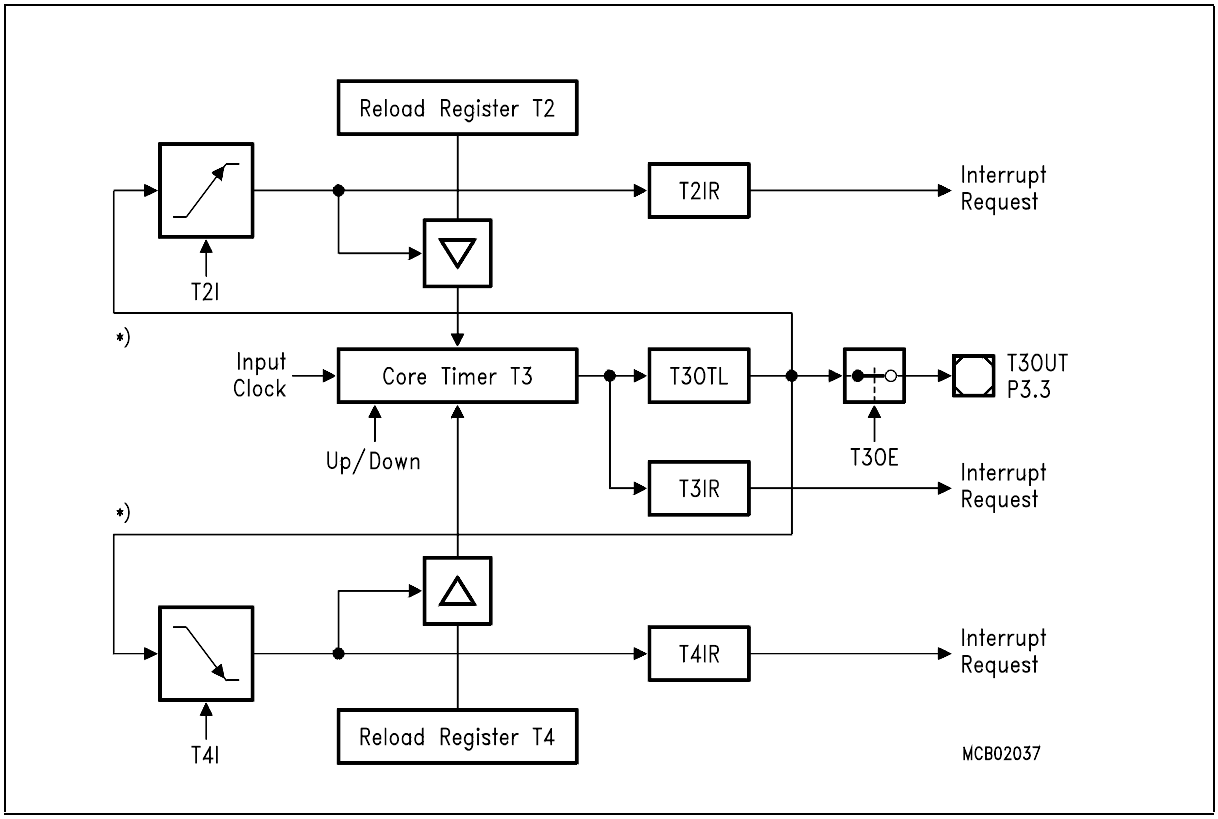


Figure 80 GPT1 timer reload configuration for PWM generation

Note *Avoid selecting the same reload trigger event for both auxiliary timers as both reload registers would try to load the core timer at the same time. If this combination is selected, T2 is disregarded and the contents of T4 is reloaded.*

Auxiliary timer in capture mode

Capture mode for the auxiliary timers T2 and T4 is selected by setting bit field TxM in the respective register TxCON to '101_B'. In capture mode the contents of the core timer are latched into an auxiliary timer register in response to a signal transition at the respective auxiliary timer's external input pin TxIN. The capture trigger signal can be a positive, a negative, or both a positive and a negative transition.

The two least significant bits of bit field TxI are used to select the active transition (see table in the counter mode section), while the most significant bit TxI.2 is irrelevant for capture mode. It is recommended to keep this bit cleared (TxI.2 = '0').

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

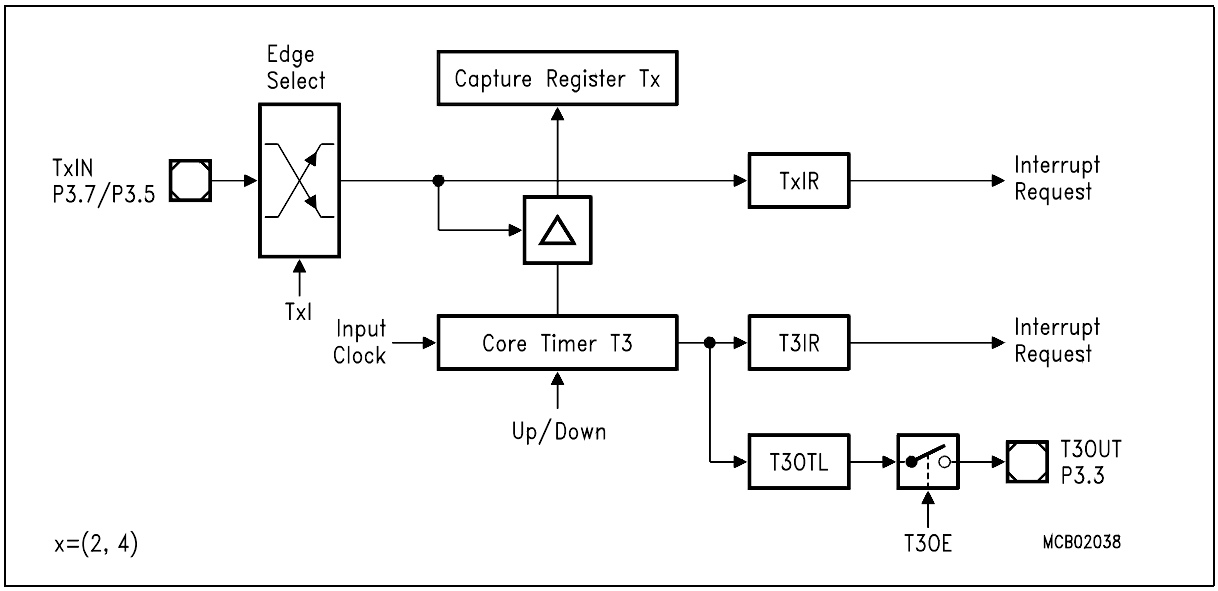


Figure 81 GPT1 auxiliary timer in capture mode

Upon a trigger (selected transition) at the corresponding input pin TxIN the contents of the core timer are loaded into the auxiliary timer register and the associated interrupt request flag TxIR will be set.

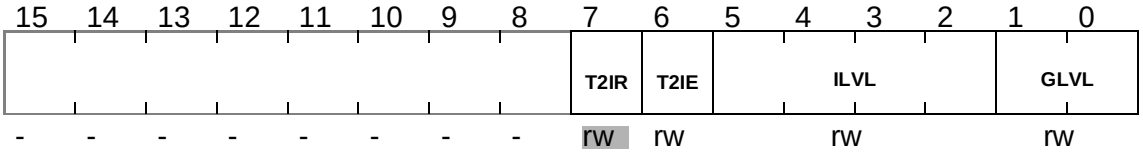
Note *The direction control bits DP3.7 (for T2IN) and DP3.5 (for T4IN) must be set to '0', and the level of the capture trigger signal should be held high or low for at least 8 CPU clock cycles before it changes to ensure correct edge detection.*

11.1.3 Interrupt control for GPT1 timers

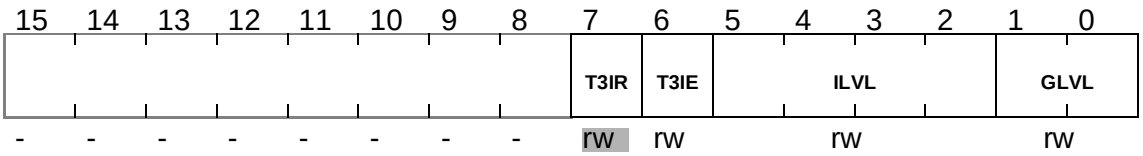
When a timer overflows from $FFFF_H$ to 0000_H (when counting up), or when it underflows from 0000_H to $FFFF_H$ (when counting down), its interrupt request flag (T2IR, T3IR or T4IR)

in register TxIC will be set. This will cause an interrupt to the respective timer interrupt vector (T2INT, T3INT or T4INT) or trigger a PEC service, if the respective interrupt enable bit (T2IE, T3IE or T4IE in register TxIC) is set. There is an interrupt control register for each of the three timers.

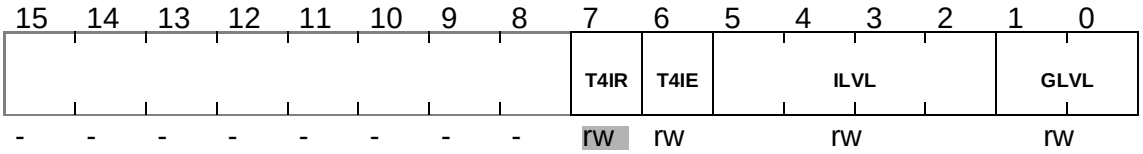
T2IC (FF60h / B0h) **SFR** **Reset Value: - - 00h**



T3IC (FF62h / B1h) **SFR** **Reset Value: - - 00h**



T4IC (FF64h / B2h) **SFR** **Reset Value: - - 00h**



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

11.2 Timer block GPT2

From a programmer's point of view, the GPT2 block is represented by a set of SFRs as summarized below. Those portions of port and direction registers which are used for alternate functions by the GPT2 block are shaded.

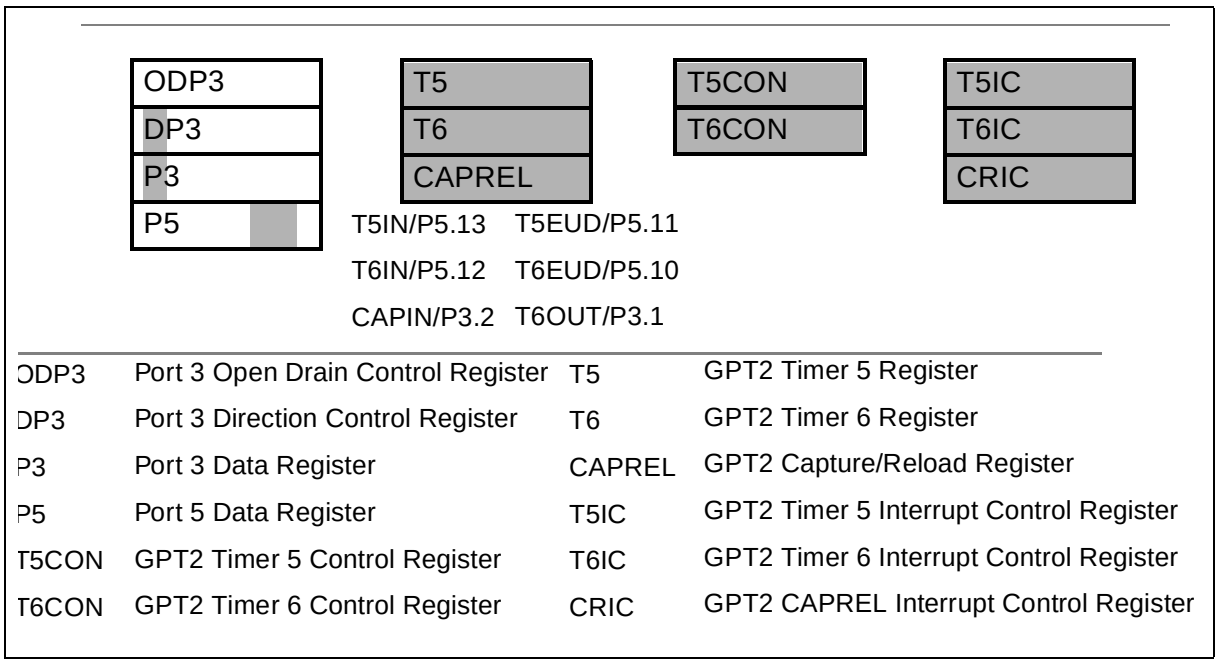


Figure 82 SFRs and port pins associated with timer block GPT2

Timer block GPT2 supports high precision event control with a maximum resolution of 4 CPU clock cycles. It includes the two timers T5 and T6, and the 16-bit capture/reload register CAPREL. Timer T6 is referred to as the core timer, and T5 is referred to as the auxiliary timer of GPT2.

Each timer has an alternate input function pin associated with it which serves as the gate control in gated timer mode, or as the count input in counter mode. The count direction (Up / Down) may be programmed via software or may be dynamically altered by a signal at an external control input pin. An overflow/underflow of T6 is indicated by the output toggle bit T6OTL whose state may be output on an alternate function port pin. In addition, 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 the timers of the CAPCOM units is provided through a direct connection. Triggered by an external signal, the contents of T5 can be captured into register CAPREL, and T5 may optionally be cleared. Both timer T6 and T5 can count up or down, and the current timer value can be read or modified by the CPU in the non-bitaddressable SFRs T5 and T6.

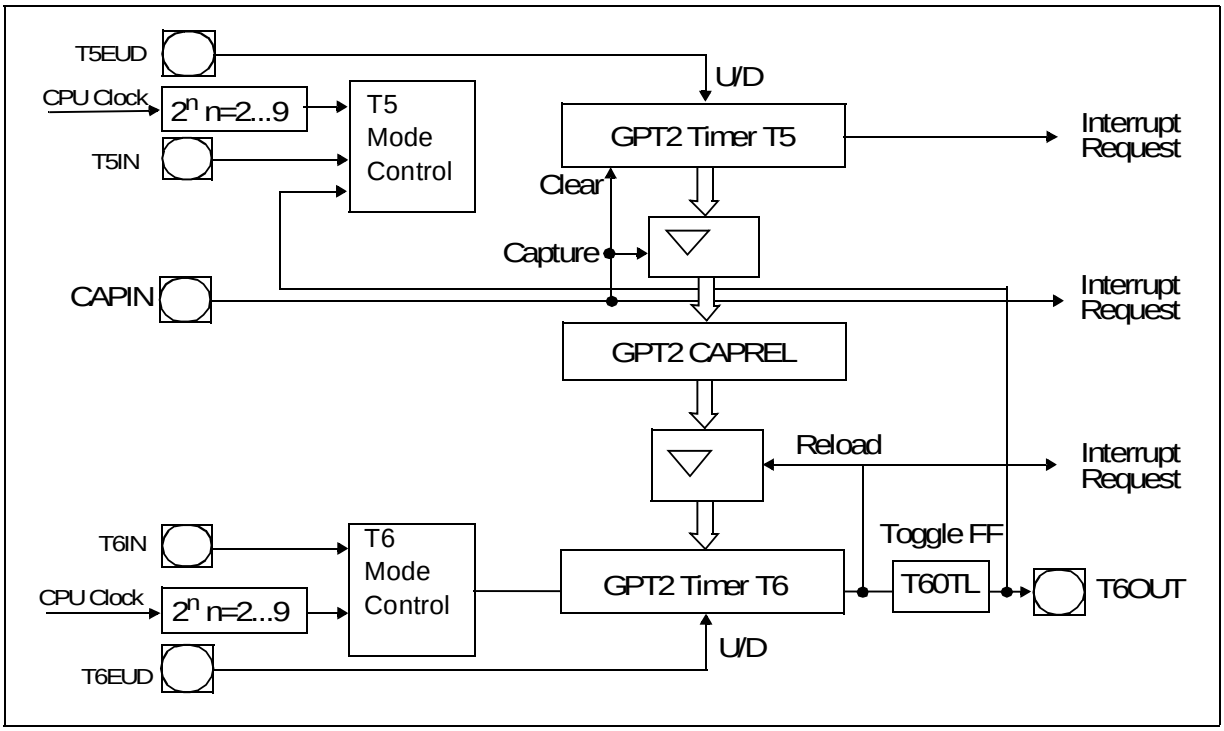


Figure 83 GPT2 block diagram

11.2.1 GPT2 core timer T6

The operation of the core timer T6 is controlled by its bit-addressable control register T6CON.

T6CON (FF48h / A4h) **SFR** **Reset Value: 0000h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T6SR	-	-	-	-	T6 OTL	T6OE	T6 UDE	T6UD	T6R	T6M			T6I		
rw	-	-	-	-	rw	rw	rw	rw	rw	rw			rw		

Bit	Function
T6I	**Timer 6 Input Selection** Depends on the Operating Mode, see respective sections.

Bit	Function
T6M	**Timer 6 Mode Control (Basic Operating Mode)** 0 0 0: Timer Mode 0 0 1: Counter Mode 0 1 0: Gated Timer with Gate active low 0 1 1: Gated Timer with Gate active high 1 X X: Reserved. Do not use this combination.
T6R	**Timer 6 Run Bit** T6R = '0': Timer / Counter 6 stops T6R = '1': Timer / Counter 6 runs
T6UD	**Timer 6 Up / Down Control** [1]
T6UDE	**Timer 6 External Up/Down Enable** [1]
T6OE	**Alternate Output Function Enable** T6OE = '0': Alternate Output Function Disabled T6OE = '1': Alternate Output Function Enabled
T6OTL	**Timer 6 Output Toggle Latch** Toggles on each overflow / underflow of T6. Can be set or reset by software.
T6SR	**Timer 6 Reload Mode Enable** T6SR = '0': Reload from register CAPREL Disabled T6SR = '1': Reload from register CAPREL Enabled

1. For the effects of bits T6UD and T6UDE refer to Table 37 GPT2 core timer T6 count direction control below.

Timer 6 run bit

The timer can be started or stopped by software through bit T6R (Timer T6 Run Bit). If T6R='0', the timer stops. Setting T6R to '1' will start the timer.
In gated timer mode, the timer will only run if T6R='1' and the gate is active (high or low, as programmed).

Count direction control

The count direction of the core timer can be controlled either by software, or by the external input pin T6EUD (Timer T6 External Up/Down Control Input), which is the alternate input function of port pin P5.10. These options are selected by bits T6UD and T6UDE in control register T6CON. When the up/down control is done by software (bit T6UDE='0'), the count direction can be altered by setting or clearing bit T6UD. When T6UDE='1', pin T6EUD is selected to be the controlling source of the count direction. However, bit T6UD can still be used to reverse the actual count direction, as shown in the table below. If T6UD='0' and pin T6EUD shows a low level, the timer is counting up. With a high level at T6EUD the timer is counting down. If T6UD='1', a high level at pin T6EUD 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.

Pin TxEUD	Bit TxUDE	Bit TxUD	Count Direction
X	0	0	Count Up
X	0	1	Count Down
0	1	0	Count Up
1	1	0	Count Down
0	1	1	Count Down
1	1	1	Count Up

Table 37 GPT2 core timer T6 count direction control

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

Timer 6 output toggle latch

An overflow or underflow of timer T6 will clock the toggle bit T6OTL in control register T6CON. T6OTL can also be set or reset by software. Bit T6OE (Alternate Output Function Enable) in register T6CON enables the state of T6OTL to be an alternate function of the external output pin T6OUT/P3.1. For that purpose, a '1' must be written into port data latch P3.1 and pin T6OUT/P3.1 must be configured as output by setting direction control bit DP3.1 to '1'. If T6OE='1', pin T6OUT then outputs the state of T6OTL. If T6OE='0', pin T6OUT can be used as general purpose IO pin.

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, the state of T6OTL does not have to be available at pin T6OUT, because an internal connection is provided for this option.

An overflow or underflow of timer T6 can also be used to clock the timers in the CAPCOM units. For this purpose, there is a direct internal connection between timer T6 and the CAPCOM timers.

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 internal system clock divided by a programmable pre-scaler, which is selected by bit field T6I. The input frequency f_{T6} for timer T6 and its

resolution r_{T6} are scaled linearly with lower clock frequencies f_{CPU}, as can be seen from the following formula:.

$$f_{T6} = f_{CPU} / 4 \times 2^{\langle T6I \rangle} \qquad r_{T6[\mu S]} = (4 \times 2^{\langle T6I \rangle}) / f_{CPU}[MHz]$$

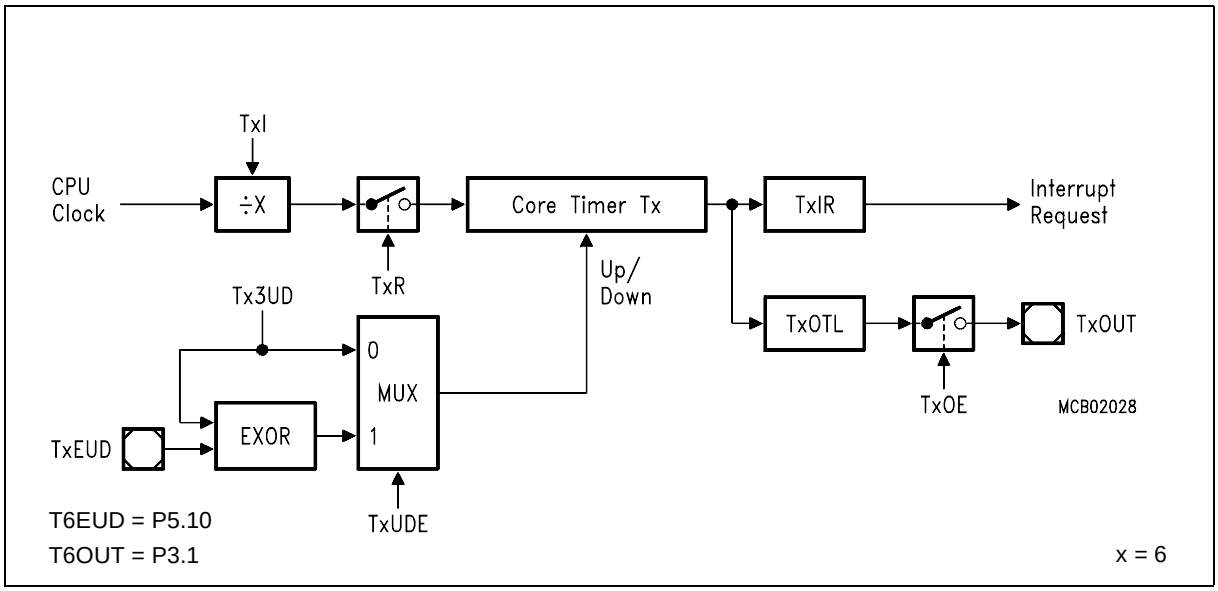


Figure 84 Block diagram of core timer T6 in timer mode

The timer resolutions which result from the selected pre-scaler option are listed in the table below. This table also applies to the Gated Timer Mode of T6 and to the auxiliary timer T5 in timer and gated timer mode.

	Timer Input Selection T5I / T6I							
	000_B	001_B	010_B	011_B	100_B	101_B	110_B	111_B
Pre-scaler factor	4	8	16	32	64	128	256	512
Resolution in CPU clock cycles	4	8	16	32	64	128	256	512

Table 38 GPT2 timer resolutions

Refer to the device datasheet for a table of timer input frequencies, resolution and periods for the range of pre-scaler options.

Timer 6 in gated timer mode

Gated timer mode for the core timer T6 is selected by setting bit field T6M in register T6CON to '010_B' or '011_B'. Bit T6M.0 (T6CON.3) selects the active level of the gate input. In gated timer mode the same options for the input frequency as for the timer mode are available. However, the input clock to the timer in this mode is gated by the external input pin T6IN (Timer T6 External Input), which is an alternate function of P5.12.

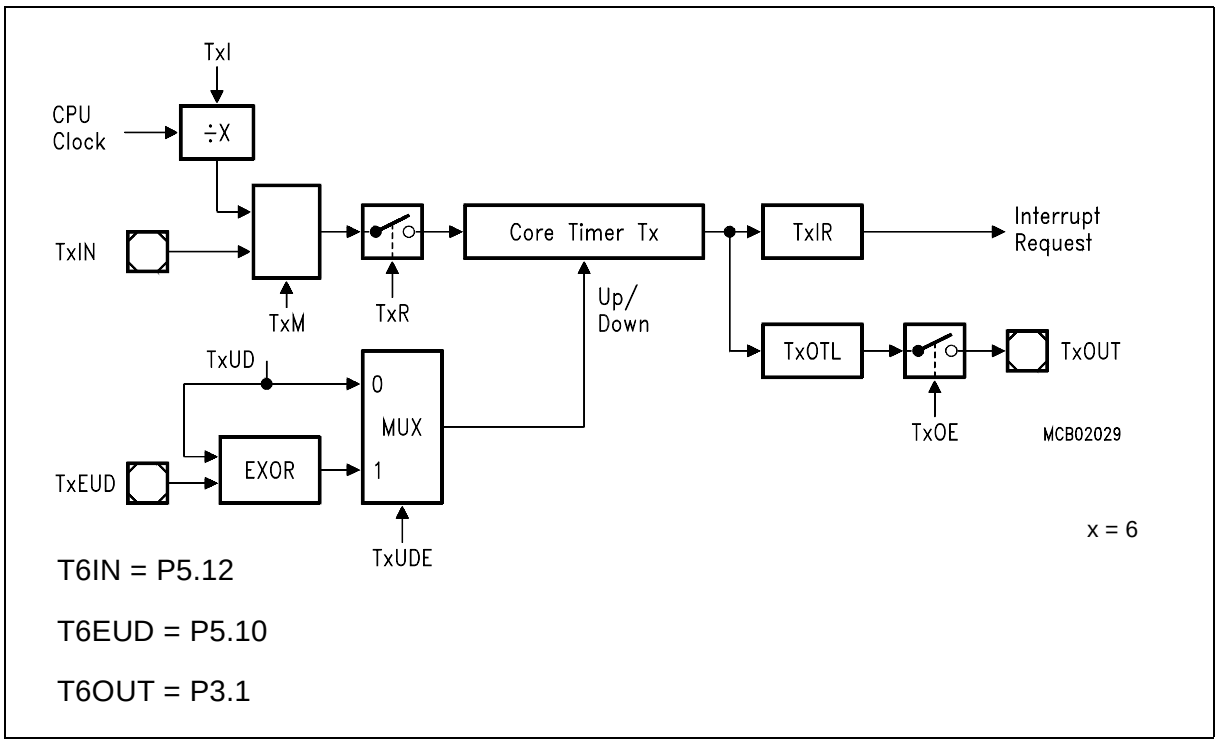


Figure 85 Block diagram of core timer T6 in gated timer mode

If T6M.0='0', the timer is enabled when T6IN shows a low level. A high level at this pin stops the timer. If T6M.0='1', pin T6IN 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 T6R. The timer will only run, if T6R='1' and the gate is active. It will stop, if either T6R='0' or the gate is inactive.

Note A transition of the gate signal at pin T6IN does not cause an interrupt request.

Timer 6 in counter mode

Counter mode for the core timer T6 is selected by setting bit field T6M in register T6CON to '001_B'. In counter mode timer T6 is clocked by a transition at the external input pin T6IN, which is an alternate function of P5.12. The event causing an increment or decrement of the

timer can be a positive, a negative, or both a positive and a negative transition at this pin. Bit field T6I in control register T6CON selects the triggering transition (see table below). The maximum input frequency which is allowed in counter mode is $f_{CPU}/8$. To ensure that a transition of the count input signal which is applied to T6IN is correctly recognized, its level should be held high or low for at least 4 CPU clock cycles before it changes.

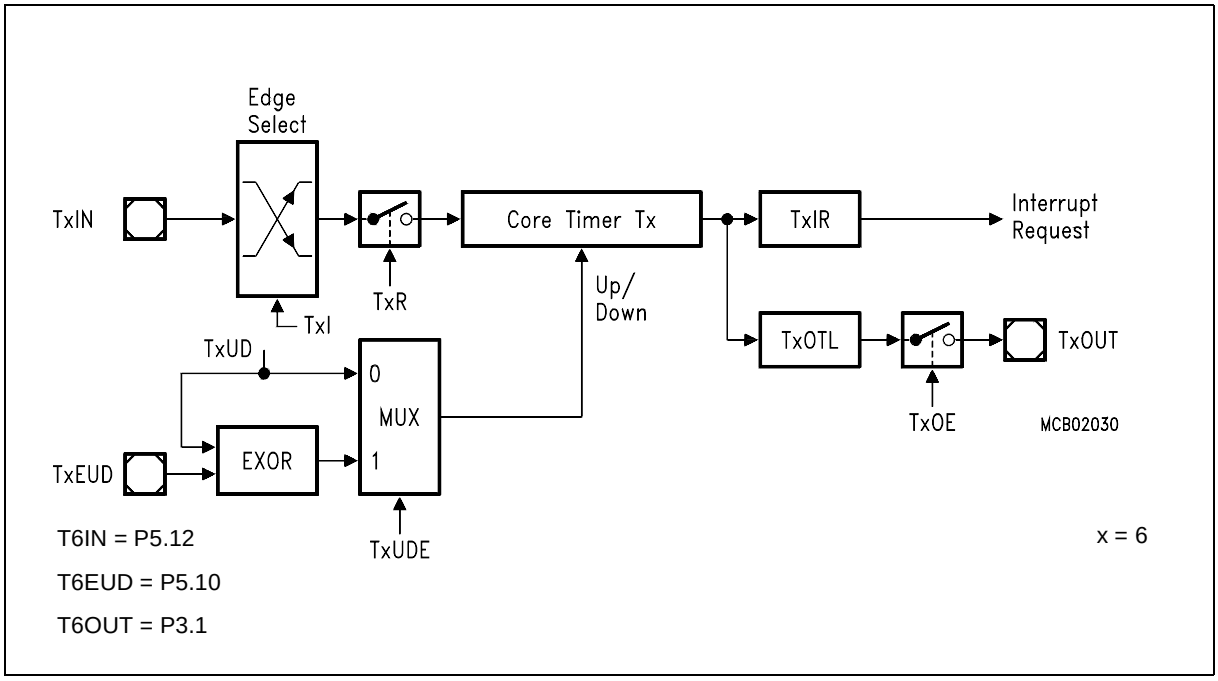


Figure 86 Block diagram of core timer T6 in counter mode

T6I	Triggering Edge for Counter Increment / Decrement
0 0 0	None. Counter T6 is disabled
0 0 1	Positive transition (rising edge) on T6IN
0 1 0	Negative transition (falling edge) on T6IN
0 1 1	Any transition (rising or falling edge) on T6IN
1 X X	Reserved. Do not use this combination

Table 39 GPT2 core timer T6 (counter mode) input edge selection

11.2.2 GPT2 Auxiliary Timer T5

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

to these 3 counting modes, the auxiliary timer can be concatenated with the core timer. The auxiliary timer has no output toggle latch and no alternate output function.

The individual configuration for timer T5 is determined by its bitaddressable control register T5CON. Note that functions which are present in both timers of block GPT2 are controlled in the same bit positions and in the same manner in each of the specific control registers.

T5CON (FF46h / A3h) **SFR** **Reset Value: 0000h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T5SC	T5 CLR	CI		-	-	CT3	T5 UDE	T5UD	T5R	-	T5M		T5I		
rw	rw	rw		-	-	-	rw	rw	rw	-	rw		rw		

Bit	Function
T5I	**Timer 5 Input Selection** Depends on the Operating Mode, see respective sections.
T5M	**Timer 5 Mode Control (Basic Operating Mode)** 0 0: Timer Mode 0 1: Counter Mode 1 0: Gated Timer with Gate active low 1 1: Gated Timer with Gate active high
T5R	**Timer 5 Run Bit** T5R = ‘0’: Timer / Counter 5 stops T5R = ‘1’: Timer / Counter 5 runs
T5UD	**Timer 5 Up / Down Control** [1]
T5UDE	**Timer 5 External Up/Down Enable**[1]
CI	**Register CAPREL Input Selection** 0 0: Capture disabled 0 1: Positive transition (rising edge) on CAPIN 1 0: Negative transition (falling edge) on CAPIN 1 1: Any transition (rising or falling edge) on CAPIN
T5CLR	**Timer 5 Clear Bit** T5CLR = ‘0’: Timer 5 not cleared on a capture T5CLR = ‘1’: Timer 5 is cleared on a capture

Bit	Function
T5SC	Timer 5 Capture Mode Enable T5SC = '0': Capture into register CAPREL Disabled T5SC = '1': Capture into register CAPREL Enabled

1. *For the effects of bits TxUD and TxUDE refer to Table 37 GPT2 core timer T6 count direction control.*

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 timer mode or gated timer mode

When the auxiliary timer T5 is programmed to timer mode or gated timer mode, its operation is the same as described for the core timer T6. The descriptions, figures and tables apply accordingly with one exception: There is no output toggle latch and no alternate output pin for T5.

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 either by a transition at the external input pin T5IN, or by a transition of timer T6's output toggle latch T6OTL.

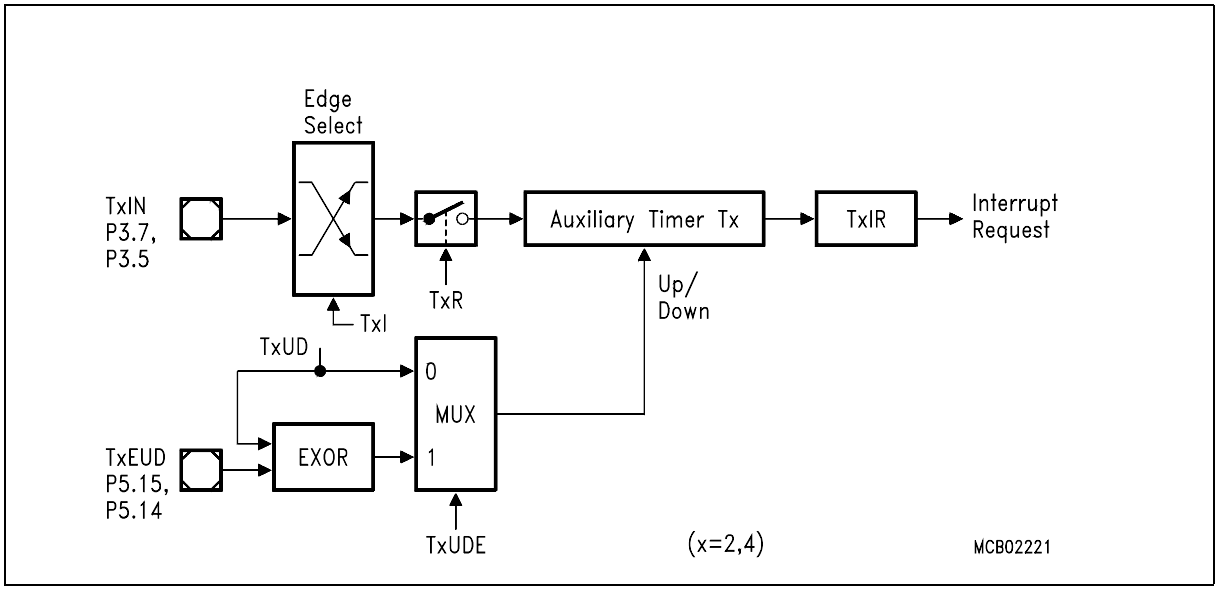


Figure 87 Block diagram of auxiliary timer T5 in counter mode

The event causing an increment or decrement of the timer can be a positive, a negative, or both a positive and a negative transition at either the input pin, or at the toggle latch T6OTL. Bit field T5I in control register T5CON selects the triggering transition (see table below).

T5I	Triggering Edge for Counter Increment / Decrement
X 0 0	None. Counter T5 is disabled
0 0 1	Positive transition (rising edge) on T5IN
0 1 0	Negative transition (falling edge) on T5IN
0 1 1	Any transition (rising or falling edge) on T5IN
1 0 1	Positive transition (rising edge) 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

Table 40 GPT2 auxiliary timer (counter mode) input edge selection

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.*

The maximum input frequency which is allowed in counter mode is $f_{CPU}/4$. To ensure that a transition of the count input signal which is applied to T5IN is correctly recognized, its level should be held high or low for at least 4 f_{CPU} cycles before it changes.

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 33bit timer (16bit 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.

T6 can operate in timer, gated timer or counter mode in this case.

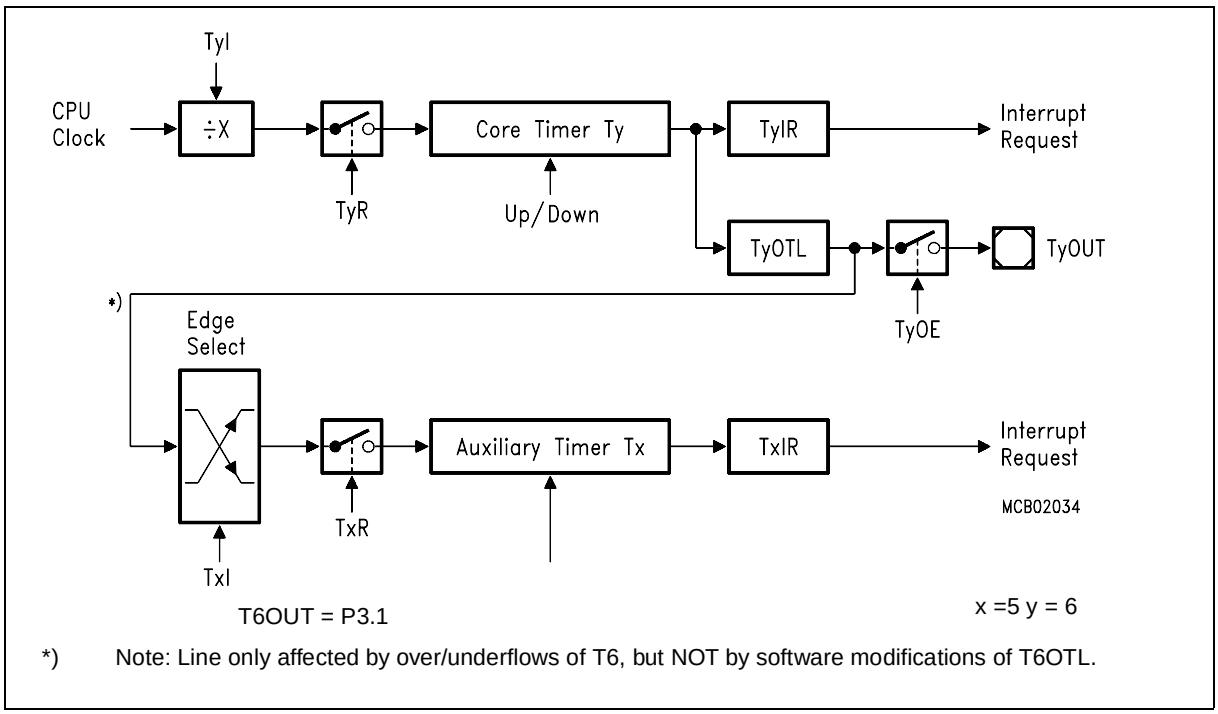


Figure 88 Concatenation of core timer T6 and auxiliary timer T5

GPT2 capture/reload register CAPREL in capture mode

This 16-bit register can be used as a capture register for the auxiliary timer T5. This mode is selected by setting bit T5SC='1' in control register T5CON. Bit CT3 selects the external input pin CAPIN or the input pins of timer T3 as the source for a capture trigger. Either a positive, a negative, or both a positive and a negative transition at this pin can be selected to trigger the capture function or transitions on input T3IN or input T3EUD or both inputs T3IN and T3EUD. The active edge is controlled by bit field CI in register T5CON.

The maximum input frequency for the capture trigger signal at CAPIN is $f_{CPU}/4$. To ensure that a transition of the capture trigger signal is correctly recognized, its level should be held for at least 4 CPU clock cycles before it changes.

When the timer T3 capture trigger is enabled (CT3='1') the CAPREL register captures the contents of T5 upon transitions of the selected input(s). These values can be used to measure T3's input signals. This is useful e.g. when T3 operates in incremental interface mode, in order to derive dynamic information (speed, acceleration, deceleration) from the input signals.

When a selected transition at the external input pin (CAPIN, T3IN, T3EUD) is detected, the contents of the auxiliary timer T5 are latched into register CAPREL, and interrupt request flag CRIR is set. With the same event, timer T5 can be cleared to 0000_H. This option is controlled by bit T5CLR in register T5CON. If T5CLR='0', the contents of timer T5 are not affected by a capture. If T5CLR='1', timer T5 is cleared after the current timer value has been latched into register CAPREL.

Note *Bit T5SC only controls whether a capture is performed or not. If T5SC='0', the input pin CAPIN can still be used to clear timer T5 or as an external interrupt input. This interrupt is controlled by the CAPREL interrupt control register CRIC.*

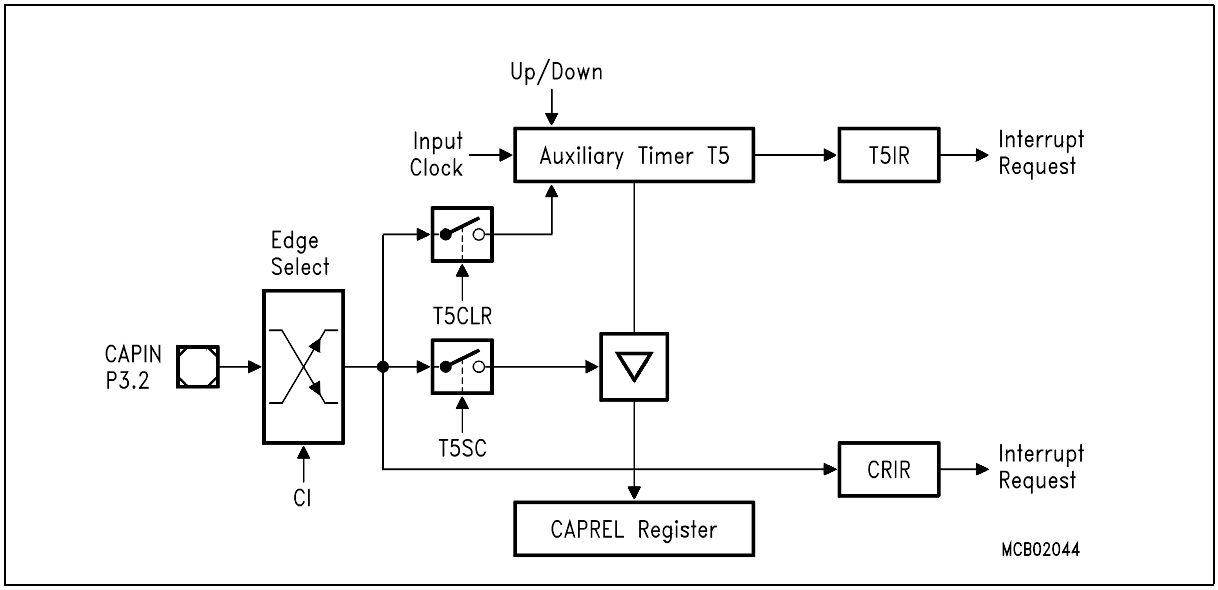


Figure 89 GPT2 register CAPREL in capture mode

GPT2 capture/reload register CAPREL in reload mode

This 16-bit register can be used as a reload register for the core timer T6. This mode is selected by setting bit T6SR='1' in register T6CON. The event causing a reload in this mode is an overflow or underflow of the core timer T6.

When timer T6 overflows from FFFFh to 0000h (when counting up) or when it underflows from 0000h to FFFFh (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.

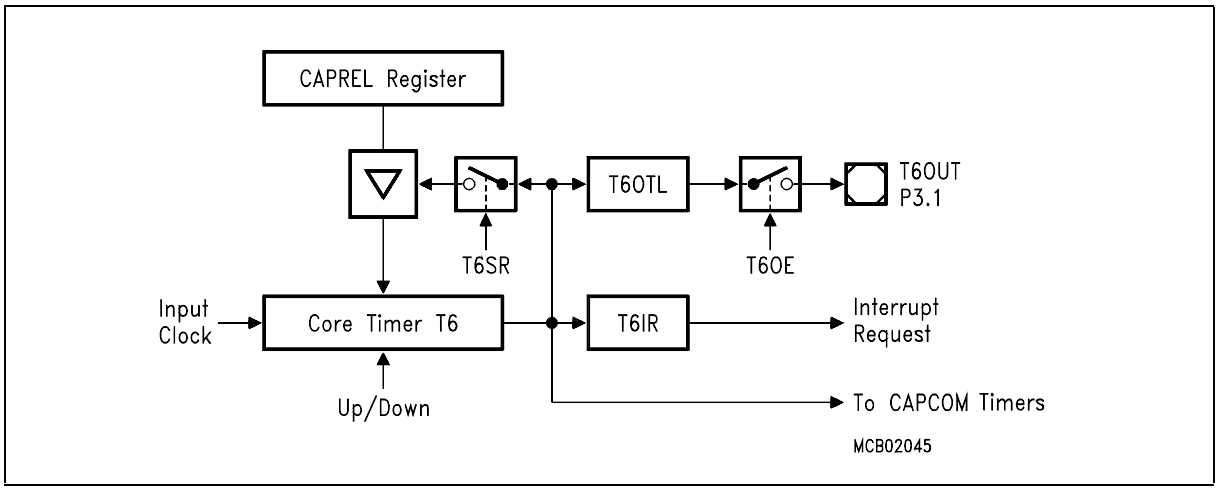


Figure 90 GPT2 register CAPREL in reload mode

GPT2 capture/reload register CAPREL in capture-and-reload mode

Since the reload function and the capture function of register CAPREL can be enabled individually by bits T5SC and T6SR, the two functions can be enabled simultaneously by setting both bits. This feature can be used to generate an output frequency that is a multiple of the input frequency.

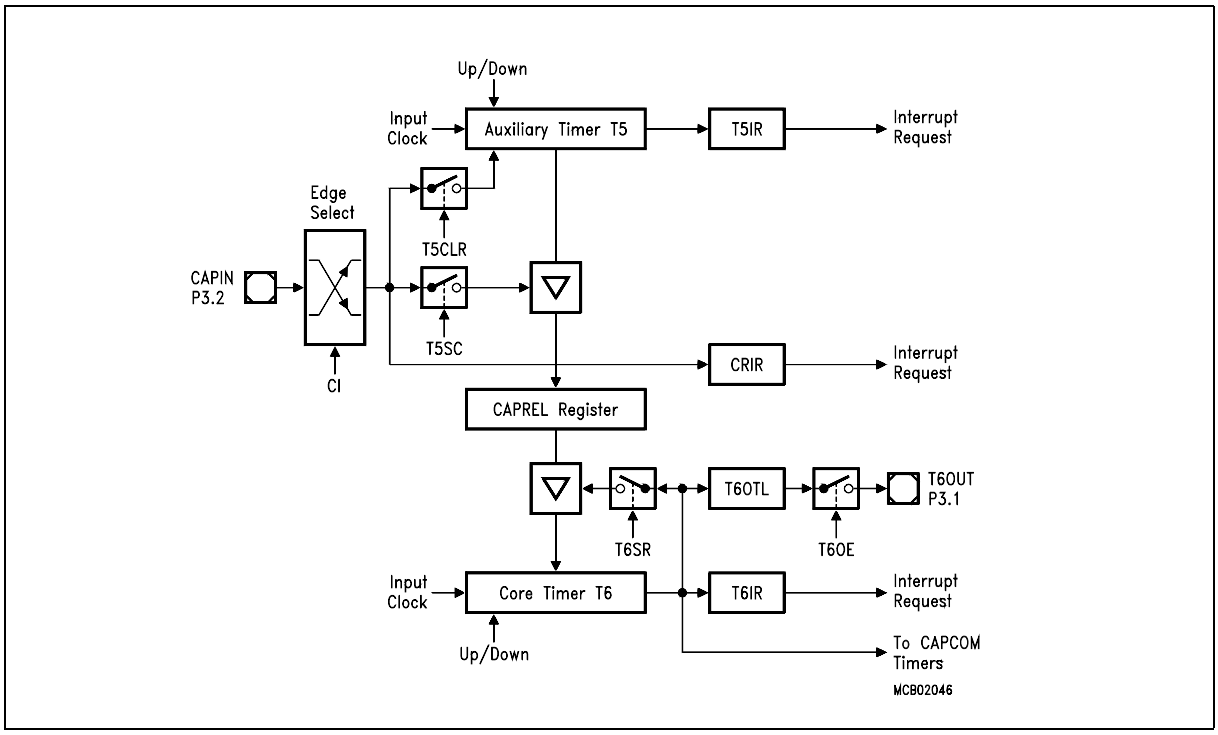


Figure 91 GPT2 register CAPREL in capture-and-reload mode

This combined mode can be used to detect consecutive external events which may occur periodically, but where a finer resolution, that means, more 'ticks' within the time between two external events is required.

For this purpose, the time between the external events is measured using timer T5 and the CAPREL register. Timer T5 runs in timer mode counting up with a frequency of e.g. $f_{CPU}/32$. The external events are applied to pin CAPIN. When an external event occurs, the timer T5 contents are latched into register CAPREL, and timer T5 is cleared (T5CLR='1'). Thus, register CAPREL always contains the correct time between two events, measured in timer T5 increments. Timer T6, which runs in timer mode counting down with a frequency of e.g. $f_{CPU}/4$, uses the value in register CAPREL to perform a reload on underflow. This means, the value in register CAPREL represents the time between two underflows of timer T6, now measured in timer T6 increments. Since timer T6 runs 8 times faster than timer T5, it will underflow 8 times within the time between two external events. Thus, the underflow signal of timer T6 generates 8 'ticks'. Upon each underflow, the interrupt request flag T6IR will be set and bit T6OTL will be toggled. The state of T6OTL may be output on pin T6OUT. This signal has 8 times more transitions than the signal which is applied to pin CAPIN.

The underflow signal of timer T6 can furthermore be used to clock one or more of the timers of the CAPCOM units, which gives the user the possibility to set compare events based on a finer resolution than that of the external events.

11.2.3 Interrupt control for GPT2 timers and CAPREL

When a timer overflows from $FFFF_H$ to 0000_H (when counting up), or when it underflows from 0000_H to $FFFF_H$ (when counting down), its interrupt request flag (T5IR or T6IR) in register TxIC will be set. Whenever a transition according to the selection in bit field CI is detected at pin CAPIN, interrupt request flag CRIR in register CRIC is set. Setting any request flag will cause an interrupt to the respective timer or CAPREL interrupt vector (T5INT, T6INT or CRINT) or trigger a PEC service, if the respective interrupt enable bit (T5IE or T6IE in register TxIC, CRIE in register CRIC) is set. There is an interrupt control register for each of the two timers and for the CAPREL register.

T5IC (FF66h / B3h) **SFR** **Reset Value: - - 00h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								T5IR	T5IE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

T6IC (FF68h / B4h) **SFR** **Reset Value: - - 00h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								T6IR	T6IE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

CRIC (FF6Ah / B5h) **SFR** **Reset Value: - - 00h**

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

Note *Refer to “Interrupt control registers” on page 86 for an explanation of the control fields.*

12 ASYNCHRONOUS/SYNCHRONOUS SERIAL INTERFACE

The asynchronous/synchronous serial interface ASC0 provides serial communication between the ST10R272L and other microcontrollers, microprocessors or external peripherals.

A dedicated baud rate generator sets up all standard baud rates without oscillator tuning. 3 separate interrupt vectors are provided for transmission, reception and erroneous reception. 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, there is a mechanism to distinguish address from data bytes (8-bit data + wake up bit mode).

In synchronous mode, the ASC0 transmits or receives bytes (8 bits) synchronously to a shift clock which is generated by the ASC0. The ASC0 always shifts the LSB first. A loop back option is available for testing purposes.
A number of optional hardware error detection capabilities increase the reliability of data transfers. A parity bit can automatically be generated on transmission or be checked on reception. Framing error detection recognizes 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 by the time a new character is received.

The operating mode of the serial channel ASC0 is controlled by its bit addressable control register S0CON. This register contains control bits for mode and error check selection, and status flags for error identification.

A transmission is started by writing to the (write-only) transmit buffer register S0TBUF (by an instruction or a PEC data transfer). Only the number of data bits which is determined by the selected operating mode will actually be transmitted, i.e. bits written to positions 9 through 15 of register S0TBUF are always insignificant. After a transmission has been completed, the transmit buffer register is cleared to 0000h.
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 to send characters back-to-back without gaps.

Data reception is enabled by the Receiver Enable Bit S0REN. After reception of a character has been completed, the received data and, if provided by the selected operating mode, the received parity bit can be read from the (read-only) Receive Buffer register S0RBUF. Bits in the upper half of S0RBUF which are not valid in the selected operating mode will be read as zeros.
Data reception is double-buffered, so that reception of a second character may already begin before the previously received character has been read out of the receive buffer register. In all modes, receive buffer overrun error detection can be selected through bit S0OEN. When enabled, the overrun error status flag S0OE and the error interrupt request flag S0EIR will be set when the receive buffer register has not been read by the time reception of a second character is complete. The previously received character in the receive buffer is overwritten.

The Loop-Back option (selected by bit S0LB) simultaneously receives the data currently being transmitted. This may be used to test serial communication routines at an early stage without having to provide an external network. In loop-back mode the alternate input/output functions of the Port 3 pins are not necessary.

Note *Serial data transmission or reception is only possible when the Baud Rate Generator Run Bit S0R is set to '1'. Otherwise the serial interface is idle. Do not program the mode control field S0M in register S0CON to one of the reserved combinations to avoid unpredictable behavior of the serial interface.*

S0CON (FFB0h / D8h) **SFR** **Reset Value: 0000h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S0R	S0LB	S0 BRS	S0 ODD	-	S0OE	S0FE	S0PE	S0 OEN	S0 FEN	S0 PEN	S0 REN	S0 STP	S0M		
rw	rw	rw	rw	-	rw	rw	rw	rw	rw	rw	rw	rw	rw		

Bit	Function		
S0M	**ASC0 Mode Control**		
	0 0 0:	8-bit data	synchronous operation
	0 0 1:	8-bit data	async. operation
	0 1 0:	Reserved	Do not use this combination!
	0 1 1:	7-bit data + parity	async. operation
	1 0 0:	9-bit data	async. operation
	1 0 1:	8-bit data + wake up bit	async. operation
	1 1 0:	Reserved.	Do not use this combination!
	1 1 1:	8-bit data + parity	async. operation
S0STP	**Number of Stop Bits Selection**		**async. operation**
	0:	One stop bit	
	1:	Two stop bits	
S0REN	**Receiver Enable Bit**		
	0:	Receiver disabled	
	1:	Receiver enabled	
	(Reset by hardware after reception of byte in synchronous mode)		
S0PEN	**Parity Check Enable Bitasync. operation**		
	0:	Ignore parity	
	1:	Check parity	

Bit	Function	
S0FEN	**Framing Check Enable Bitasync. operation** 0: Ignore framing errors 1: Check framing errors	
S0OEN	**Overrun Check Enable Bit** 0: Ignore overrun errors 1: Check overrun errors	
S0PE	**Parity Error Flag** Set by hardware on a parity error (S0PEN='1'). Must be reset by software.	
S0FE	**Framing Error Flag** Set by hardware on a framing error (S0FEN='1'). Must be reset by software.	
S0OE	**Overrun Error Flag** Set by hardware on an overrun error (S0OEN='1'). Must be reset by software.	
S0ODD	**Parity Selection Bit** 0: Even parity (parity bit set on odd number of '1's in data) 1: Odd parity (parity bit set on even number of '1's in data)	
S0BRS	**Baudrate Selection Bit** 0: Divide clock by reload-value + constant (depending on mode) 1: Additionally reduce serial clock to 2/3rd	
S0LB	**LoopBack Mode Enable Bit** 0: Standard transmit/receive mode 1: Loopback mode enabled	
S0R	**Baudrate Generator Run Bit** 0: Baudrate generator disabled (ASC0 inactive) 1: Baudrate generator enabled	

12.1 Asynchronous operation

Asynchronous mode supports full-duplex communication, where both transmitter and receiver use the same data frame format and the same baud rate. Data is transmitted on pin

TXD0/P3.10 and received on pin RXD0/P3.11. These signals are alternate functions of Port 3 pins.

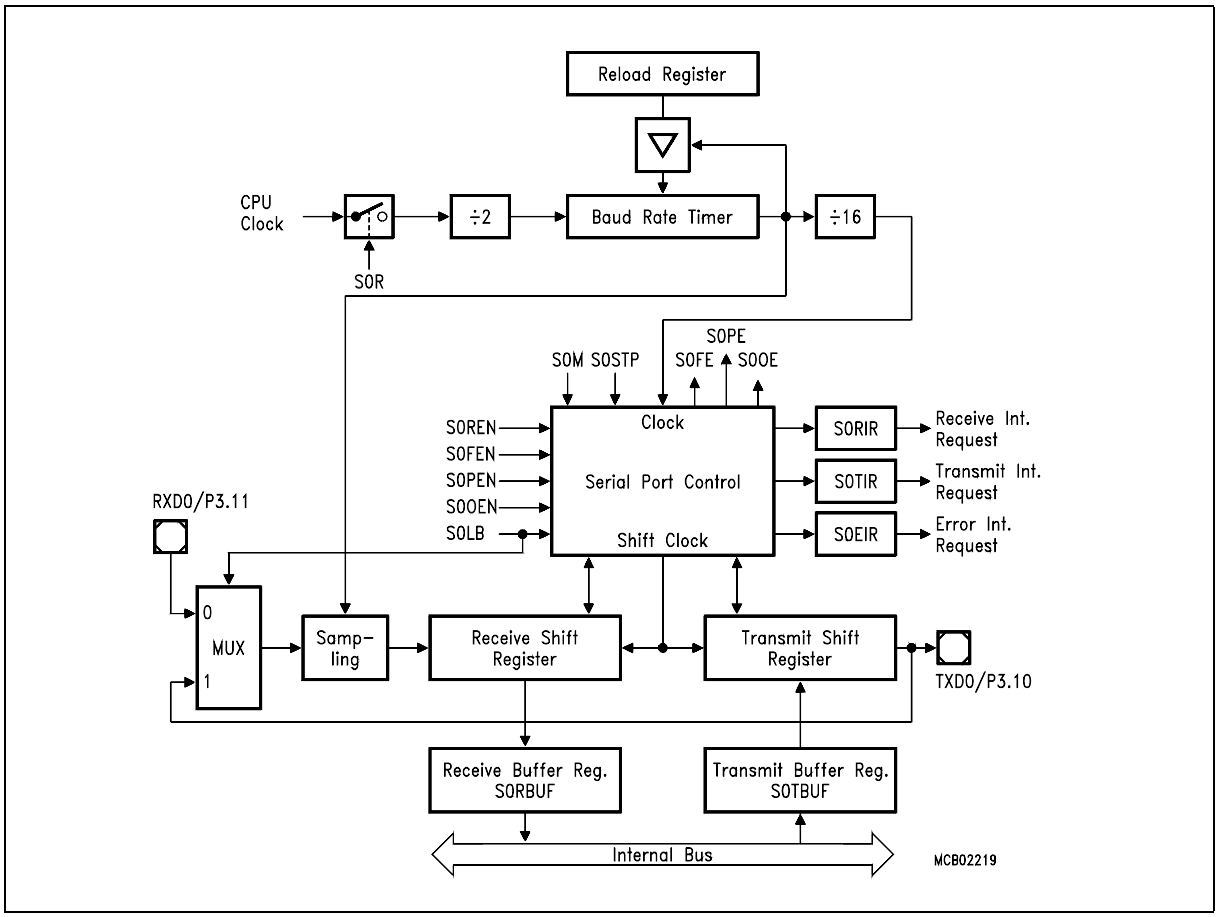


Figure 92 Asynchronous mode of serial channel ASC0

8-bit data frames either consist of 8 data-bits D7...D0 (S0M='001b'), or of 7 data-bits D6...D0 plus an automatically generated parity bit (S0M='011b'). Parity may be odd or even, depending on bit S0ODD in register S0CON. An even parity bit will be set, if the modulo-2-sum of the 7 data bits is '1'. An odd parity bit will be cleared in this case. Parity checking is enabled via bit S0PEN (always OFF in 8-bit data mode). The parity error flag S0PE will be set along with the error interrupt request flag, if a wrong parity bit is received. The parity bit itself will be stored in bit S0RBUF.7.

9-bit data frames consist of either, 9 data-bits D8...D0 (S0M='100b'), of 8 data-bits D7...D0 plus an automatically generated parity bit (S0M='111b'), or of 8 data-bits D7...D0 plus wake-up bit (S0M='101b'). Parity may be odd or even, depending on bit S0ODD in register S0CON. An even parity bit will be set, if the modulo-2-sum of the 8 data bits is '1'. An odd parity bit will be cleared in this case. Parity checking is enabled via bit S0PEN (always OFF in 9-bit data and wake-up mode). The parity error flag S0PE will be set along with the error

interrupt request flag, if a wrong parity bit is received. The parity bit itself will be stored in bit S0RBUF.8.

In wake-up mode, received frames are only transferred to the receive buffer register if the 9th bit (the wake-up bit) is ‘1’. If this bit is ‘0’, no receive interrupt request will be activated and no data will be transferred.

This feature may be used to control communication in multi-processor system:
When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address byte which identifies the target slave. An address byte differs from a data byte in that the additional 9th bit is a '1' for an address byte and a '0' for a data byte, so no slave will be interrupted by a data 'byte'. An address 'byte' will interrupt all slaves (operating in 8-bit data + wake-up bit mode), so each slave can examine the 8 LSBs of the received character (the address). The addressed slave will switch to 9-bit data mode (e.g. by clearing bit S0M.0), which enables it to also receive the data bytes that will be coming (having the wake-up bit cleared). The slaves that were not being addressed, remain in 8-bit data + wake-up bit mode, ignoring the following data bytes.

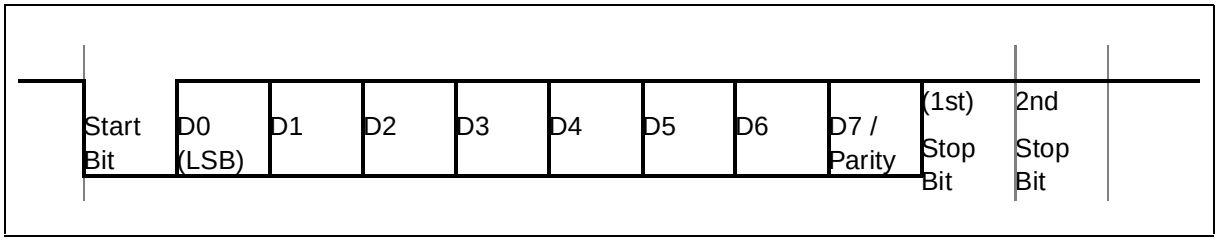


Figure 93 Asynchronous 8-bit data frames

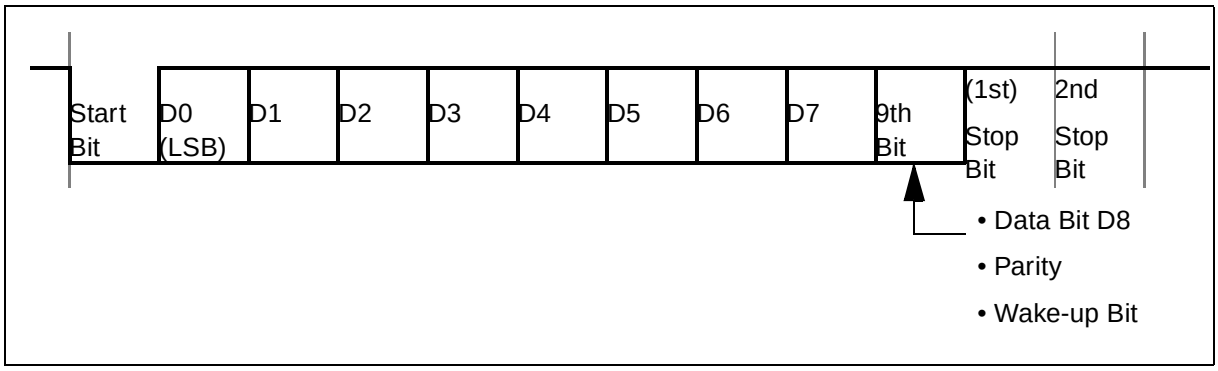


Figure 94 Asynchronous 9-bit data frames

Asynchronous transmission begins at the next overflow of the divide-by-16 counter (see figure above), provided that S0R is set and data has been loaded into S0TBUF. The transmitted data frame consists of three basic elements:

- the start bit

- the data field (8 or 9 bits, LSB first, including a parity bit, if selected)
- the delimiter (1 or 2 stop bits)

Data transmission is double buffered. When the transmitter is idle, the transmit data loaded into S0TBUF is immediately moved to the transmit shift register thus freeing S0TBUF for the next data to be sent. This is indicated by the transmit buffer interrupt request flag S0TBIR being set. S0TBUF may now be loaded with the next data, while transmission of the previous one is still going on.

The transmit interrupt request flag S0TIR will be set before the last bit of a frame is transmitted, i.e. before the first or the second stop bit is shifted out of the transmit shift register.
The transmitter output pin TXD0/P3.10 must be configured for alternate data output, i.e. P3.10='1' and DP3.10='1'.

Asynchronous reception is initiated by a falling edge (1-to-0 transition) on pin RXD0, provided that bits S0R and S0REN are set. The receive data input pin RXD0 is sampled at 16 times the rate of the selected baud rate. A majority decision of the 7th, 8th and 9th sample determines the effective bit value. This avoids erroneous results that may be caused by noise.

If the detected value is not a '0' when the start bit is sampled, the receive circuit is reset and waits for the next 1-to-0 transition at pin RXD0. If the start bit proves valid, the receive circuit continues sampling and shifts the incoming data frame into the receive shift register.

When the last stop bit has been received, the content of the receive shift register is transferred to the receive data buffer register S0RBUF. Simultaneously, the receive interrupt request flag S0RIR is set after the 9th sample in the last stop bit time slot (as programmed), regardless whether valid stop bits have been received or not. The receive circuit then waits for the next start bit (1-to-0 transition) at the receive data input pin.

The receiver input pin RXD0/P3.11 must be configured for input, i.e. DP3.11='0'.

Asynchronous reception is stopped by clearing bit S0REN. A currently received frame is completed including the generation of the receive interrupt request and an error interrupt request, if appropriate. Start bits that follow this frame will not be recognized.

Note *In wake-up mode, received frames are only transferred to the receive buffer register, if the 9th bit (the wake-up bit) is '1'. If this bit is '0', no receive interrupt request will be activated and no data will be transferred.*

12.2 Synchronous operation

Synchronous mode supports half-duplex communication, basically for simple IO expansion via shift registers. Data is transmitted and received via pin RXD0/P3.11, while pin TXD0/

P3.10 outputs the shift clock. These signals are alternate functions of Port 3 pins. Synchronous mode is selected with S0M='000b'.

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

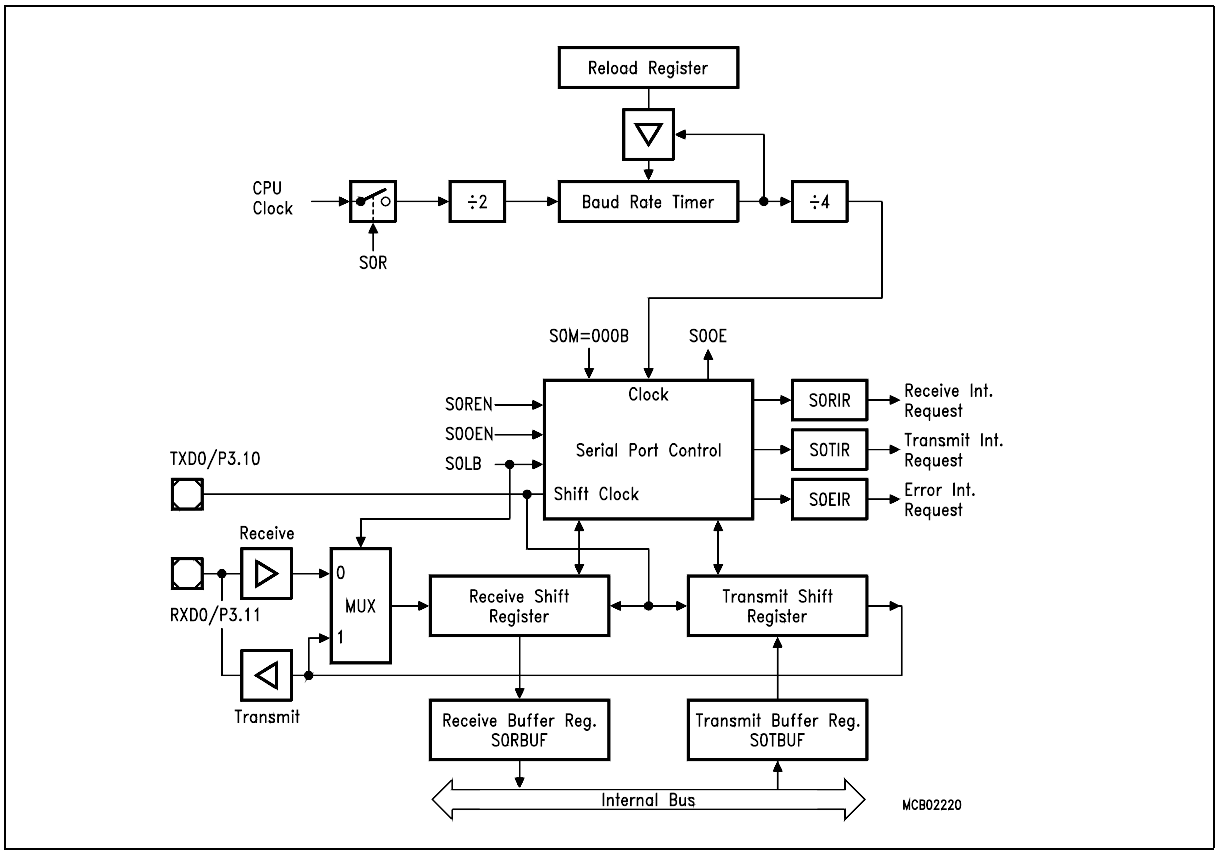


Figure 95 Synchronous mode of serial channel ASC0

Synchronous transmission begins within 4 CPU clock cycles after data has been loaded into S0TBUF, provided that S0R is set and S0REN='0' (half-duplex, no reception). Data transmission is double buffered. When the transmitter is idle, the transmit data loaded into S0TBUF is immediately moved to the transmit shift register, thus freeing S0TBUF for the next data to be sent. This is indicated by the transmit buffer interrupt request flag S0TBIR being set. S0TBUF may now be loaded with the next data while transmission of the previous one is still going on. The data bits are transmitted synchronously with the shift clock. After the bit time for the 8th data bit, both pins TXD0 and RXD0 will go high, the transmit interrupt request flag S0TIR is set, and serial data transmission stops.

Pin TXD0/P3.10 must be configured for alternate data output, i.e. P3.10='1' and DP3.10='1', in order to provide the shift clock. Pin RXD0/P3.11 must also be configured for output (P3.11='1' and DP3.11='1') during transmission.

Synchronous reception is initiated by setting bit S0REN='1'. If bit S0R=1, the data applied at pin RXD0 are clocked into the receive shift register synchronous to the clock which is output at pin TXD0. After the 8th bit has been shifted in, the content of the receive shift register is transferred to the receive data buffer S0RBUF, the receive interrupt request flag S0RIR is set, the receiver enable bit S0REN is reset, and serial data reception stops.

Pin TXD0/P3.10 must be configured for alternate data output, i.e. P3.10='1' and DP3.10='1', in order to provide the shift clock. Pin RXD0/P3.11 must be configured as alternate data input (DP3.11='0').

Synchronous reception is stopped by clearing bit S0REN. A currently received byte is completed including the generation of the receive interrupt request and an error interrupt request, if appropriate. Writing to the transmit buffer register while a reception is in progress has no effect on reception and will not start a transmission.

If a previously received byte has not been read out of the receive buffer register at the time the reception of the next byte is complete, both the error interrupt request flag S0EIR and the overrun error status flag S0OE will be set, provided the overrun check has been enabled by bit S0OEN.

12.3 Hardware error detection capabilities

To improve the safety of serial data exchange, the serial channel ASC0 provides an error interrupt request flag, which indicates the presence of an error, and three (selectable) error status flags in register S0CON, which indicate which error has been detected during reception. Upon completion of a reception, the error interrupt request flag S0EIR will be set simultaneously with the receive interrupt request flag S0RIR, if one or more of the following conditions are met:

- If the framing error detection enable bit S0FEN is set and any of the expected stop bits is not high, the framing error flag S0FE is set, indicating that the error interrupt request is due to a framing error (Asynchronous mode only).
- If the parity error detection enable bit S0PEN is set in the modes where a parity bit is received, and the parity check on the received data bits proves false, the parity error flag S0PE is set, indicating that the error interrupt request is due to a parity error (Asynchronous mode only).
- If the overrun error detection enable bit S0OEN is set and the last character received was not read out of the receive buffer by software or PEC transfer at the time the reception of a new frame is complete, the overrun error flag S0OE is set indicating that

the error interrupt request is due to an overrun error (Asynchronous and synchronous mode).

12.4 ASC0 baud rate generation

The serial channel ASC0 has its own dedicated 13-bit baud rate generator with 13-bit reload capability, allowing baud rate generation independent from the timers.

The baud rate generator is clocked with the CPU clock divided by 2. The timer is counting downwards and can be started or stopped through the Baud Rate Generator Run Bit S0R in register S0CON. Each underflow of the timer provides one clock pulse to the serial channel. The timer is reloaded with the value stored in its 13-bit reload register each time it underflows. The resulting clock is again divided according to the operating mode and controlled by the Baudrate Selection Bit S0BRS. If S0BRS='1', the clock signal is additionally divided to 2/3rd of its frequency (see formulas and table). So the baud rate of ASC0 is determined by the CPU clock, the reload value, the value of S0BRS and the operating mode (asynchronous or synchronous).

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

An auto-reload of the timer with the content of the reload register is performed each time S0BG is written to. However, if S0R='0' at the time the write operation to S0BG is performed, the timer will not be reloaded until the first instruction cycle after S0R='1'.

12.4.1 Asynchronous mode baud rates

For asynchronous operation, the baud rate generator provides a clock with 16 times the rate of the established baud rate. Every received bit is sampled at the 7th, 8th and 9th cycle of this clock. The baud rate for asynchronous operation of serial channel ASC0 and the required reload value for a given baudrate can be determined by the following formulas:

$$B_{Async} = \frac{f_{CPU}}{(32 + 16*\langle S0BRS\rangle) * (\langle S0BRL\rangle + 1)}$$

$$S0BRL = \left(\frac{f_{CPU}}{(32 + 16*\langle S0BRS\rangle) * B_{Async}}\right) - 1$$

<S0BRL> is the content of the reload register, taken as unsigned 13-bit integer, <S0BRS> is the value of bit S0BRS (i.e. '0' or '1'), taken as integer.

Using the above equation, the maximum baud rate can be calculated for any given clock speed. The device datasheet gives a table of values for baud rate vs reload register value for SOBRS=0 and SOBRS=1.

12.4.2 Synchronous mode baud rates

For synchronous operation, the baud rate generator provides a clock with 4 times the rate of the established baud rate. The baud rate for synchronous operation of serial channel ASC0 can be determined by the following formula:

$$B_{Sync} = \frac{f_{CPU}}{(8 + 4*<S0BRS>) * (<S0BRL> + 1)}$$

$$S0BRL = \left(\frac{f_{CPU}}{(8 + 4*<S0BRS>) * B_{Sync}}\right) - 1$$

<S0BRL> is the content of the reload register, taken as unsigned 13-bit integers, <S0BRS> is the value of bit S0BRS (i.e. ‘0’ or ‘1’), taken as integer.

Using the above equation, the maximum baud rate can be calculated for any given clock speed.

12.5 ASC0 interrupt control

Four bit addressable interrupt control registers are provided for serial channel ASC0. Register S0TIC controls the transmit interrupt, S0TBIC controls the transmit buffer interrupt, S0RIC controls the receive interrupt and S0EIC controls the error interrupt of serial channel ASC0. Each interrupt source also has its own dedicated interrupt vector. S0TINT is the transmit interrupt vector, S0TBINT is the transmit interrupt vector, S0RINT is the receive interrupt vector, and S0EINT is the error interrupt vector.

The cause of an error interrupt request (framing, parity, overrun error) can be identified by the error status flags in control register S0CON.

Note *In contrast to the error interrupt request flag S0EIR, the error status flags S0FE/ S0PE/S0OE are not reset automatically on entry into the error interrupt service routine, but must be cleared by software.*

S0TIC (FF6Ch / B6h) **SFR** **Reset Value: - - 00h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								S0TIR	S0TIE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

S0RIC (FF6Eh / B7h) **SFR** **Reset Value: - - 00h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								S0RIR	S0RIE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

S0EIC (FF70h / B8h) **SFR** **Reset Value: - - 00h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								S0EIR	S0EIE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

S0TBIC (F19Ch / CEh) **ESFR** **Reset Value: - - 00h**

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

Note *Refer to “Interrupt control registers” on page 86 for an explanation of the control fields.*

12.6 Using the ASC0 interrupts

For normal operation (i.e. apart from the error interrupt) the ASC0 provides three interrupt requests to control data exchange via this serial channel:

- S0TBIR: activated when data is moved from S0TBUF to the transmit shift register.
- S0TIR: activated before the last bit of an asynchronous frame is transmitted, or after the last bit of a synchronous frame has been transmitted.
- S0RIR: activated when the received frame is moved to S0RBUF.

While the task of the receive interrupt handler is clear, the transmitter is serviced by two interrupt handlers. This provides advantages for the servicing software.

For single transfers it is sufficient to use the transmitter interrupt (S0TIR), which indicates that the previously loaded data has been transmitted, except for the last bit of an asynchronous frame.

For multiple back-to-back transfers it is necessary to load the following data at least until the time the last bit of the previous frame is being 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 not possible at all.
Using the transmit buffer interrupt (S0TBIR) to reload transmit data allows the time to transmit a complete frame for the service routine, as S0TBUF may be reloaded while the previous data is still being transmitted.

As shown in the figure below, S0TBIR is an early trigger for the reload routine, and S0TIR indicates the completed transmission. Therefore,a software-using handshake should rely on S0TIR at the end of a data block to make sure that all data has really been transmitted.

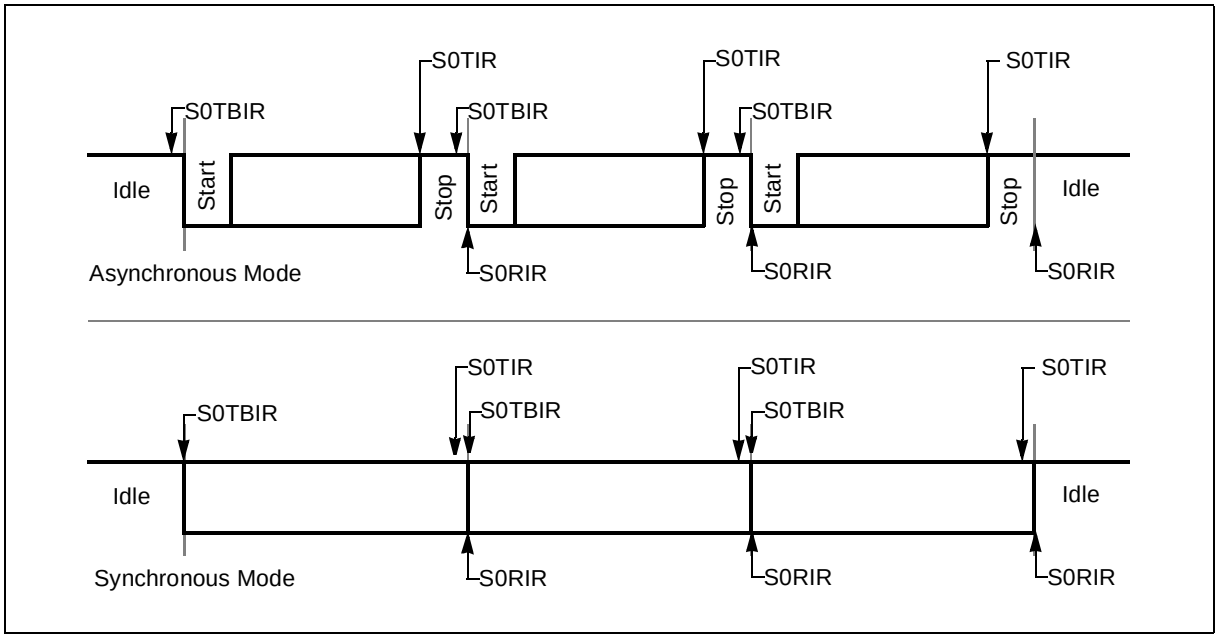


Figure 96 ASC0 interrupt generation

13 SYNCHRONOUS SERIAL PORT

The Synchronous Serial Port SSP provides high-speed serial communication with external slave devices such as EEPROM, via a three-wire interface similar to the SPI protocol. The interface lines are:

- SSPCLK: Serial clock output. Driven by the ST10R272L (master) to the peripheral (slave) which is selected for transfer.
- SSPDAT: Bi-directional serial data line. Data is transferred between the ST10R272L and the peripheral at up to 10 MBit/s.
- SSPCE0,1: Serial peripheral chip enable signals 0 and 1. These signals select one of he slaves connected to the SSP for transfer.

The SSP transmits 1...3 bytes or receives 1 byte after sending 1...3 bytes synchronously to a shift clock which is generated by the SSP. The SSP can start shifting with the LSB or with the MSB and allows to select shifting and latching clock edges as well as the clock polarity. Up to two chip select lines may be activated in order to direct data transfers to one or both of two peripheral devices.

One general interrupt vector is provided for the SSP.

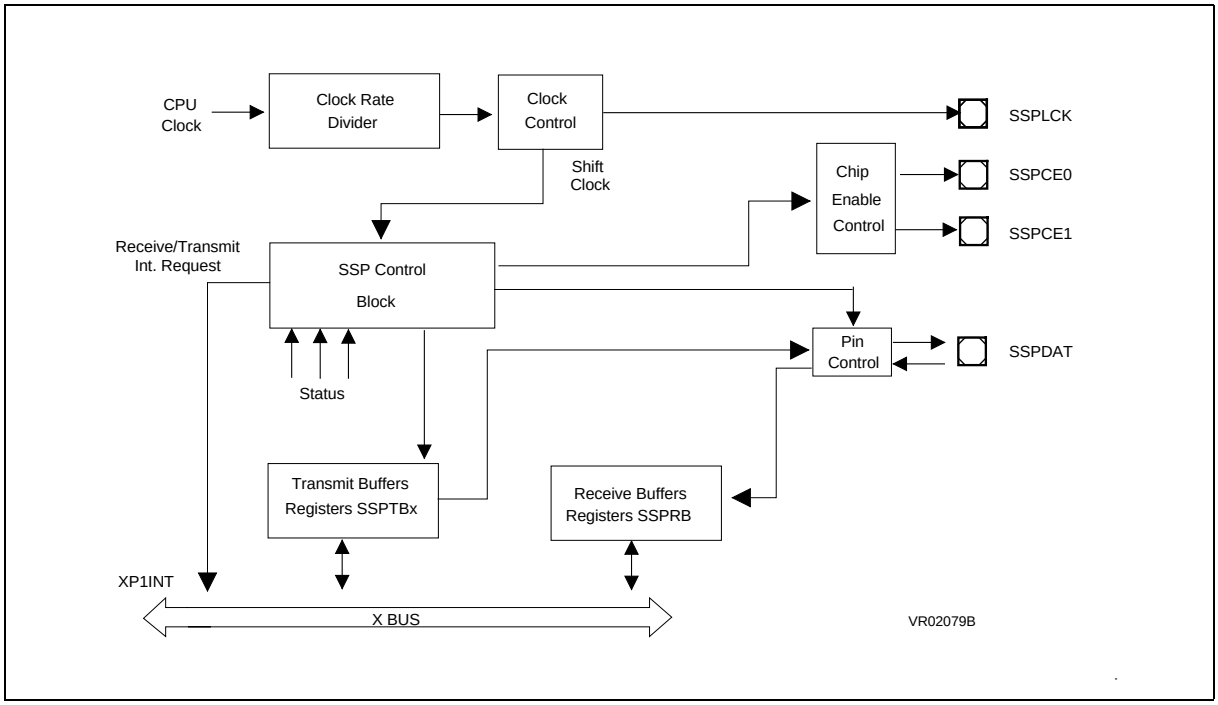


Figure 97 Synchronous serial port SSP block diagram

13.1 XBUS implementation

The SSP is implemented as an XPERipheral onto the XBUS in the address range 00EF00h - 00EFFFh, a 256 Byte range (10 byte addresses used). It is connected via a 16-bit demultiplexed bus, without waitstates, allowing word and byte accesses via the CPU or the PEC.
From a user's point of view, the XBUS can be regarded as an internal representation of the external bus in the 16-bit demultiplexed mode, allowing the fastest possible word and byte accesses by the CPU or the PEC, to the SSP registers. The SSP registers are in a special SSP address area of 256 bytes which is mapped into segment 0 and uses addresses EF00h through EFFFh (10 bytes addresses used).

13.2 SSP registers

Bit 2 of control register SYSCON serves as an enable/disable control for the SSP module. This bit is named XSSPEN (Xperipheral SSP ENable Control). After reset, XSSPEN is set to '0', and the SSP is disabled.

The four upper pins of Port4 can be used for segment address lines or general purpose I/Os, (refer to "Port 4" on page 129). In order to use the SSP, bit XSSPEN must first be set to 1. Note that SYSCON register can only be written to during the initialization phase after a reset until the EINIT instruction is executed. After the EINIT instruction, the SYSCON register is locked against modifications until the next reset.
When the SSP is enabled, the four upper pins of Port4 can not be used as general purpose I/O. Note that the segment address selection done via the system start-up configuration during reset has priority and overrides the SSP functions on these pins.

The SSP registers are organized as five 16-bit registers located on word addresses. However, all registers may be accessed bytewise in order to select special actions without

effecting other mechanisms. The figure below shows all control and data registers of the SSP.

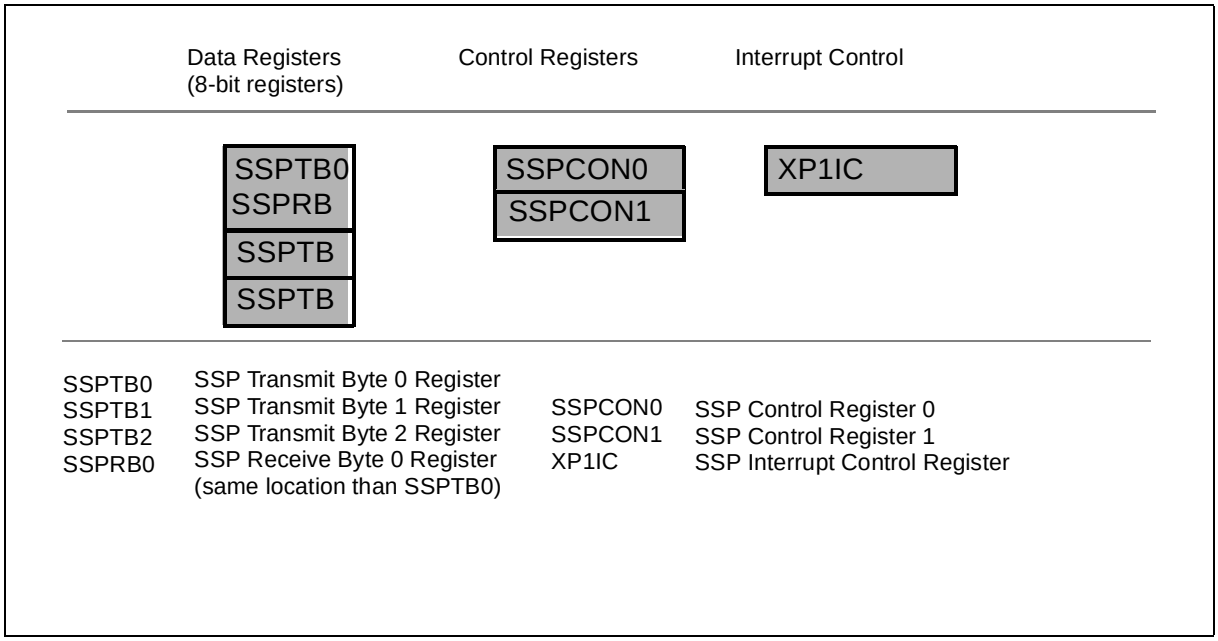


Figure 98 Synchronous serial port register

13.2.1 SSP control register 0 - SSPCON0

This register contains all bits which are required to setup a read or write operation for the SSP.

SSPCCON0 (00'EF00h) **Reset Value: 0000h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SSP BSY	-	-	-	-	-	SSP CKP	SSP CKE	SSP RW	SSP HB	SSP CM	SSPSEL			SSPCKS	
-	-	-	-	-	-	rw	rw	rw	rw	rw	rw			rw	

Bit	Function
SSPCKS	**SSP Clock Rate Selection Baud Rate** 000: SSP clock = CPU clock divided by 2 MBit/s. 001: SSP clock = CPU clock divided by 4 MBit/s. 010: SSP clock = CPU clock divided by 8 MBit/s. 011: SSP clock = CPU clock divided by 16 MBit 100: SSP clock = CPU clock divided by 32 KBit/s. 101: SSP clock = CPU clock divided by 64KBit/s. KBit/s. 110: SSP clock = CPU clock divided by 128 KBit/s. 111: SSP clock = CPU clock divided by 256 KBit/s.
SSPSEL	**SSP Chip Enable Selection** 00: No chip enable line selected. 01: Chip enable line 0 (SSPCE0) selected. 10: Chip enable line 1 (SSPCE1) selected. 11: Both chip enable lines selected. Can be used for broadcast messages. Improper use for read operations may cause line conflicts among several selected slaves.
SSPCM	**SSP Continuous Mode Selection** 0: Single Transfer Mode. Chip enable line deactivated after end of transfer. 1: Continuous Mode. Chip enable line remains active between transfers.
SSPHB	**SSP Heading Control Bit** 0: Transmit/Receive LSB First 1: Transmit/Receive MSB First
SSPRW	**SSP Read/Write Control Bit** 0: Write Operation selected 1: Read Operation selected.
SSPCKE	**SSP Clock Edge 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.
SSPCKP	**SSP Clock Polarity Control Bit** 0: Idle clock is high, leading clock edge is high-to-low transition. 1: Idle clock is low, leading clock edge is low-to-high transition.

Bit	Function
SSPBSY	**SSP Busy Flag** 0: SSP is idle 1: SSP is busy. The Busy flag is set with the first write into one of the transmit buffers. It is automatically cleared after the last bit has been transferred and selected chip select line is switched inactive.

The clock control adapts transmit and receive behavior of the SSP 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 SSPKE selects the leading edge or the trailing edge for each function. Bit SSPKP 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.

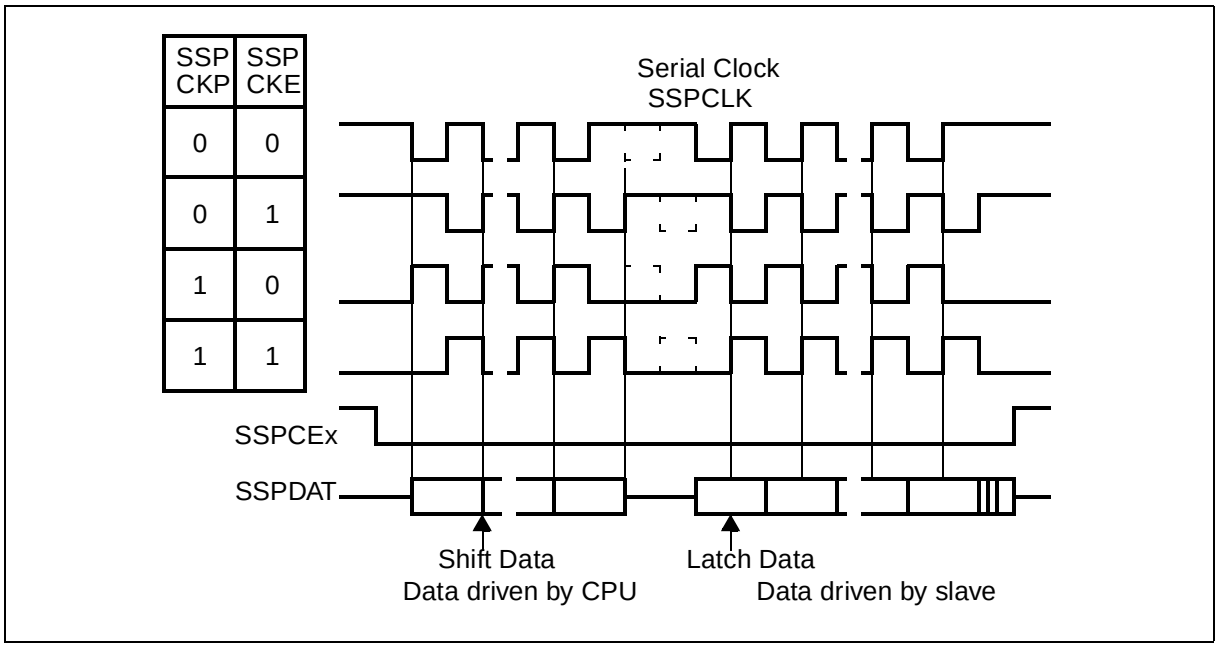


Figure 99 Serial clock phase and polarity options

13.2.2 SSP Control Register 1 - SSPCON1

This register contains all bits which are required to configure the output lines of the SSP. It contains control bits which are normally written once during the initialization of the system.

SSPCON1 (00'EF02h) **Reset Value: 0000h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	SSP CEO1	SSP CEO0	SSP CKO	SSP CEP1	SSP CEP0
-	-	-	-	-	-	-	-	-	-	-	rw	rw	rw	rw	rw

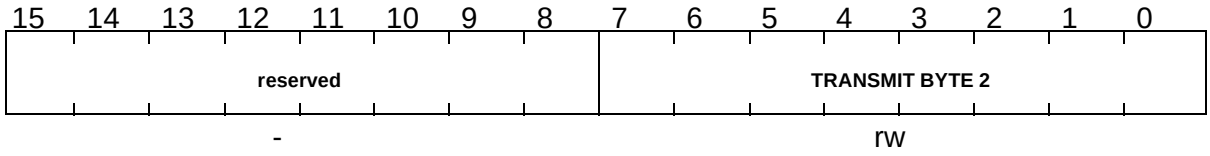
Bit	Function (Operating Mode, SSCEN = '1')
SSPCEP0	**SSP Chip Enable Line 0 (SSPCEN0) Polarity Control Bit** 0: Inactive Chip Enable line is low, active level is high. 1: Inactive Chip Enable line is high, active level is low.
SSPCEP1	**SSP Chip Enable Line 1 (SSPCEN1) Polarity Control Bit** 0: Inactive Chip Enable line is low, active level is high. 1: Inactive Chip Enable line is high, active level is low.
SSPCKO	**SSP Clock Line Output (SSPCLK) Control Bit** 0: Clock output pin state is high-impedance. 1: Clock output pin is configured to push/pull output.
SSPCEO0	**SSP Chip Enable Line 0 (SSPCEN0) Output Control Bit** 0: Chip Enable 0 output pin state is high-impedance 1: Chip Enable 0 output pin is configured to push/pull output.
SSPCEO1	**SSP Chip Enable Line 1 (SSPCEN1) Output Control Bit** 0: Chip Enable 1 output pin state is high-impedance. 1: Chip Enable 1 output pin is configured to push/pull output.

13.2.3 SSP transmit buffer registers - SSPTBx

Three 8-bit transmit registers are provided with the SSP, organized as follows:
Note that the same register is used for the transmit buffer byte 0 and the receive buffer.

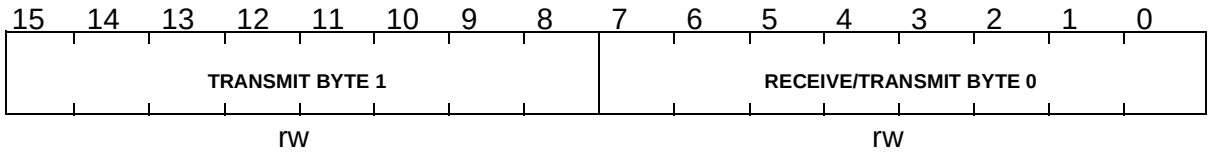
Reserved Byte (00'EF07h) **Reset Value: xxh**

SSPTB2 (00'EF06h) **Reset Value: xxh**



SSPTB1 (00'EF05h) **Reset Value: xxh**

SSPTB0 (00'EF04h) **Reset Value: xxh**



13.2.4 Initialization

After reset, all SSP I/O lines are in high-impedance state. The SSPDAT line is controlled automatically by the SSP, according to the performed read or write operation. The SSPCLK, SSPCE0 and SSPCE1 lines, however, have individual output control bits. This allows the user to first program the desired polarity of these lines before switching them to output. With this, it is possible to pull the lines to the desired initial polarity already during and after rest until they are switched to push/pull outputs by connecting external pullup or pulldown resistors to pins.

While the polarity of the chip enables lines is programmed via register SSPCON1, the polarity of the clock line is controlled through a bit in register SSPCON0. The reason for this is that the chip enable polarity normally only needs to be selected during initialization, while the clock line polarity and active edge might be switched between transfers to different peripheral slaves. This can be handled by a write to only one control register, SSPCON0, together with other necessary selections for the transfer.

13.2.5 Starting a transfer

Prior to any transfer, all required selections for this transfer should be made. This is performed through programming the control bits in SSPCON0 register to the desired values. The SSPCKS0..2 bits determine the baudrate of the transfer. With the SSPCKS1..0 bits, the appropriate chip enable line(s) is (are) selected. If a continuous transfer is desired, bit SSPCM must be set. The heading control bit SSPHB selects whether each byte is transferred with LSB or MSB first. The type of operation, a read or write operation, is controlled through bit SSPRW. If set, a read operation will take place. In order to communicate with several peripherals with different clocking requirements, the control bits SSPCKP and SSPCKE allow to set the polarity and relevant shift/latch edges of the clock.

When the programming of the control register SSPCON0 is complete, the transfer is started with a write to transmit buffer SSPTB0, regardless of the type of operation (read or write operation).

13.2.6 Performing a Write Operation

If the SPRW bit in register SSPCON0 is reset, a write operation is selected. During a write operation, information is only transferred from the CPU (master) to the slave peripheral. The transmit buffers SSPTB2...SSPTB0 are written by the CPU with the information data to be transferred. Writing to SSPTB0 will start the transfer. The following figure shows the basic waveforms for a write operation of the SSP.

The length of the transfer is determined through which transmit buffers were written to prior to the transfer. Internal flags, 'TBx_Full', are used for this purpose. These flags are set through a write to the corresponding transmit buffer register. SSPTB0 must be written in any case in order to start the transfer. Writing only to SSPTB0 will perform an 8-bit transfer of the content of SSPTB0. When the last bit is shifted out completely, the TB0_Full flag is reset. To start the next transfer, SSPTB0 must be written through software again, even if the same information data as for previous transfer should be transferred.

Content of SSPTBx registers are undefined after a write operation.

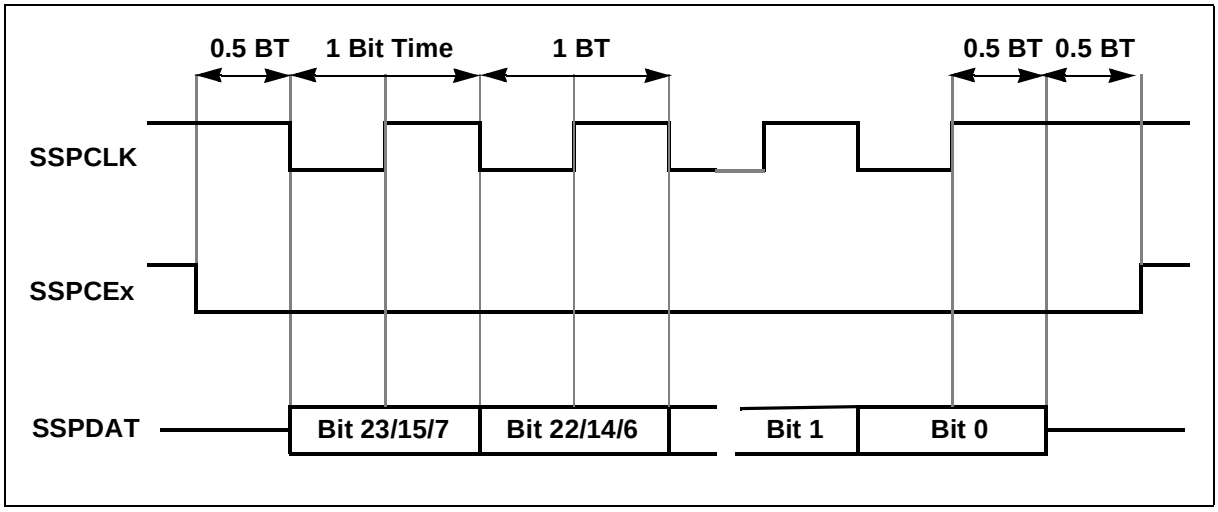


Figure 100 Write operation waveforms

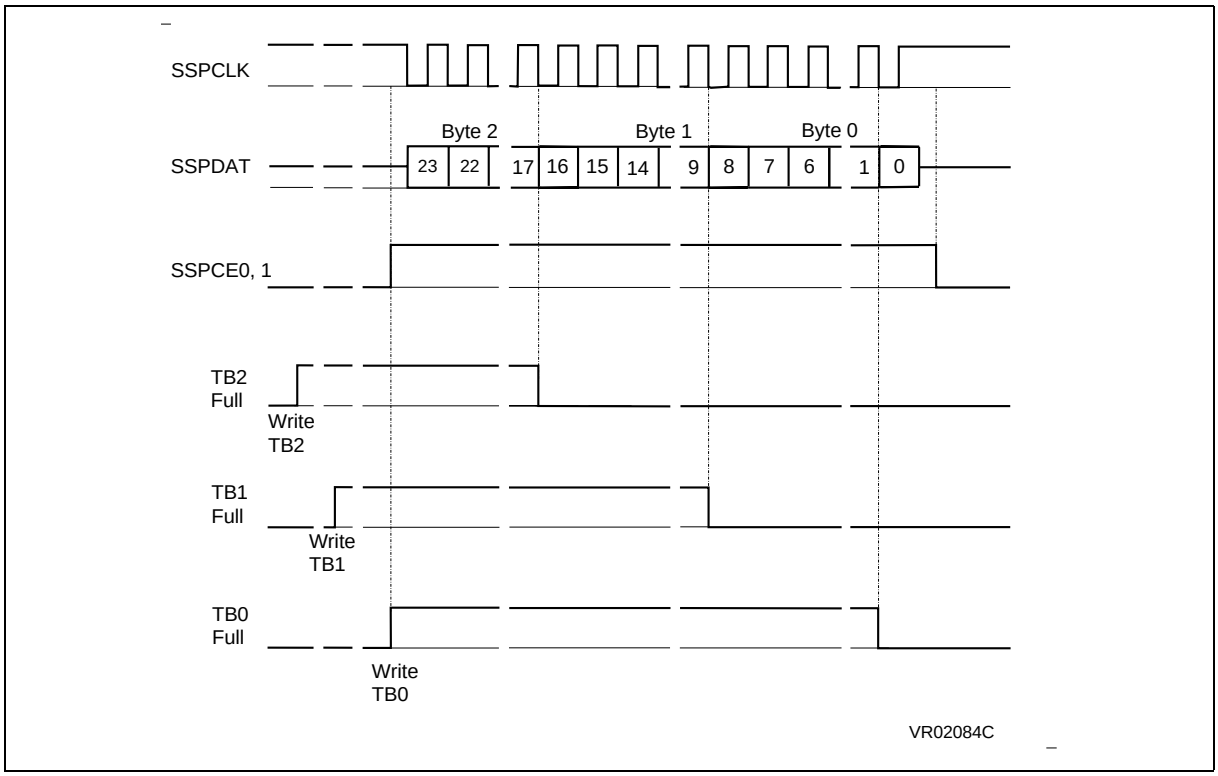


Figure 101 Write operation controlled through transmit buffer full signal

While SSPTB0 must be the last transmit buffer register written, there are no sequence requirment for writing to the other two SSPTBx registers. SSPTB2 can be written prior to SSPTB1 or vice versa. The following table shows the SSP operation as a function of which transmit buffer registers are written, prior to a transfer.

Transmit Buffers written	Transfer length	Transfer sequence
SSPTB2, SSPTB1, SSPTB0	24-bit (3 byte) transfer	SSPTB2, SSPTB1, SSPTB0
SSPTB1, SSPTB2, SSPTB0	24-bit (3 Byte) transfer	SSPTB2, SSPTB1, SSPTB0
SSPTB1, SSPTB0	16-bit (2 byte) transfer	SSPTB1, SSPTB0
SSPTB2, SSPTB0	illegal option	-
SSPTB0	8-bit (1 byte) transfer	SSPTB0

Table 41 SSP operation as a function of transmit buffer registers

Performing a read operation

If the SPRW bit in register SSPCON0 is set, a read operation is selected. During a read operation, first information (command or address information) is transferred from the CPU (master) to the slave peripheral. Then the transfer direction is switched, and information is transferred from the slave to the master. As for the write operation, the transmit buffers SSPTB2...SSPTB0 are written by the CPU with the information data to be first transferred. Writing to SSPTB0 will start the transfer. The following figure shows the basic waveforms for a read operation of the SSP.

For the first part of a read operation (transfer of SSPTBx from master to slave), the same rules as for the write operation are applied concerning the relation between the transfer length and writes in the Transmit Buffers.After writing to SSPTB0, first the content of the transmit buffers are shifted out. Then the data line SSPDAT is switched to input (high-impedance). After a gap of one bit clock, the data present at the SSPDAT pin is latched in with the next 8 clock edges. When the last bit is clocked in, the chip enable line is deactivated one half bit time later.

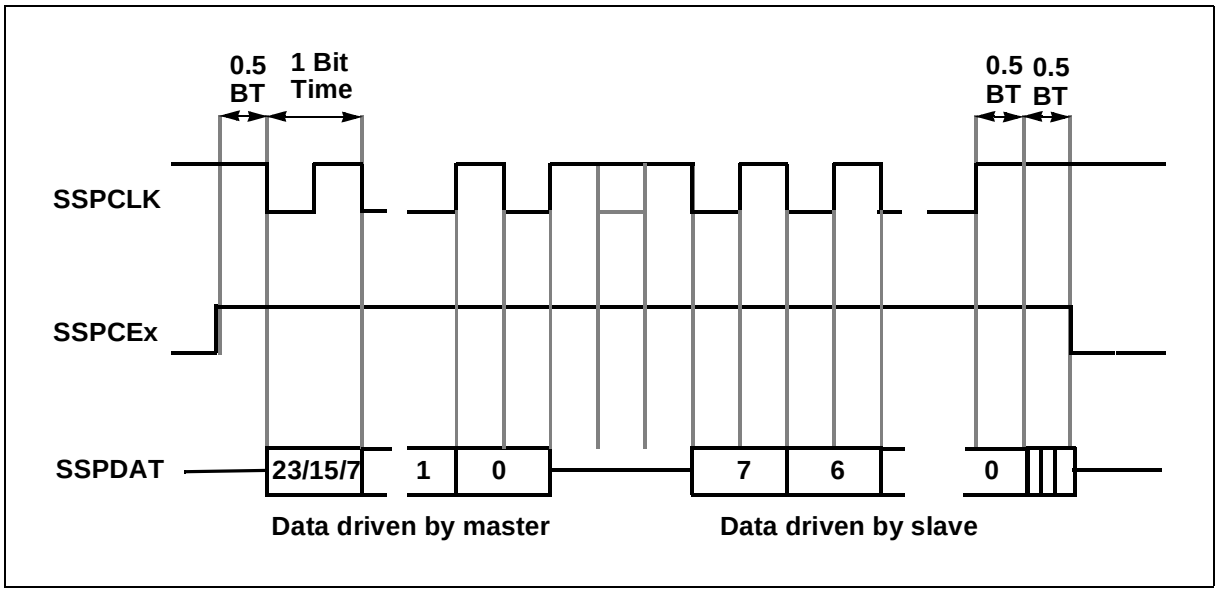


Figure 102 Read operation waveforms

13.2.7 Chip enable lines

Two chip enable lines are provided by the SSP which are automatically activated at the beginning of a transfer and deactivated again after the transfer is completed. As shown in the previous figures, activation of a chip enable line always begins one half bit time before the first data bit is output at the SSPDAT pin, and the deactivation (except for continuous transfers) is performed one half bit time after the last bit of the transfer has been completely transmitted/received.

The chip enable lines are selected through the control bits SSPSEL0 and SSPSEL1. Note that these automatic chip enable lines can be extended through normal IO pins, which, however, must be activated and deactivated through software. To avoid conflicts with the automatic chip enable lines, the combination ‘00’ for SSPSEL1..0 disables these lines, and should be programmed when using IO lines as additional chip enable signals.

It is also possible to activate both chip enable lines with the combination ‘11’ for SSPSEL1..0. This can be used if the same information has to be transferred to several slaves simultaneously (message broadcast), providing that all slave peripherals use the same clock configuration. Note that this option should only be used for write operation, since for read operations, a conflict on the data line SSPDAT could occur if several slave peripherals try to drive this line.

13.2.8 Using the SSP chip enable and clock lines for output functions

Note that the polarity and output control bits directly influence the lines SSPCLK, SSPCE0 and SSPCE1, regardless whether a transfer is in progress or not. It is recommended not to reprogram these control bits while a transfer is in progress to avoid false operation of the addressed slave peripheral(s).

However, if no transfer is in progress or the SSP is not used at all, the polarity and output control bit can be used to perform general purpose output functions on the pins SSPCLK, SSPCE0 and SSPCE1. The possible options are:

- Output a low level.
- Output a high level.
- High-impedance (tri-state). The pin can be used as a general purpose I/O through Port 4 P4 register if this port line is configured as an input through DP4 register.

13.2.9 Continuous transfer modes

In order to simplify communication with some standard slave devices such as EEPROMs, a continuous transfer mode is implemented in the SSP. This mode is distinguished from the normal mode in that the chip enable line is not deactivated automatically after a transfer is completed, but instead remains active until the mode is switched back from continuous mode (SSPCM = ‘1’) to normal mode (SSPCM = ‘0’). Thus, consecutive transfers can be performed while holding the chip enable line active during the entire procedure. This condition keeps most peripheral slave devices in the operational mode initiated through the first command transfer. EEPROMs, for instance, are placed into a read mode through first transferring a read command to it. This command is followed by the start address of the location to be read. After this, the EEPROM transfers the data stored in the specified address to the master. If now the chip enable line is deactivated, the EEPROM cancels the read mode and returns to idle mode. A new read operation must again start with the read

command followed by an address, and then the data byte at this address is returned to the master.

However, if the chip enable line is not deactivated after the EEPROM has transferred the first data byte, the EEPROM automatically increments to the next address location. If now subsequent clock pulses are applied to the device, the content of this next location is transferred to the master, and so on. In this mode, the EEPROM can be continuously read without having to issue the read command and the start address again.

A similar operation is true for writing to such an EEPROM. These devices allow a certain number of data bytes to be written after an initial write command followed by a start address for the writes.

The figure below shows the write operation in continuous mode. The TB0_Full flag is again used to start subsequent writes to the slave device. The gap between the transfers is application dependent, since the CPU first has to react on the interrupt request at the end of one transfer and rewrite the transmit buffer registers before the next transfer will start. This procedure continues until the mode is switched from continuous mode to normal mode. The chip enable line will then be deactivated immediately if no transfer is in progress or after the current transfer is completed.

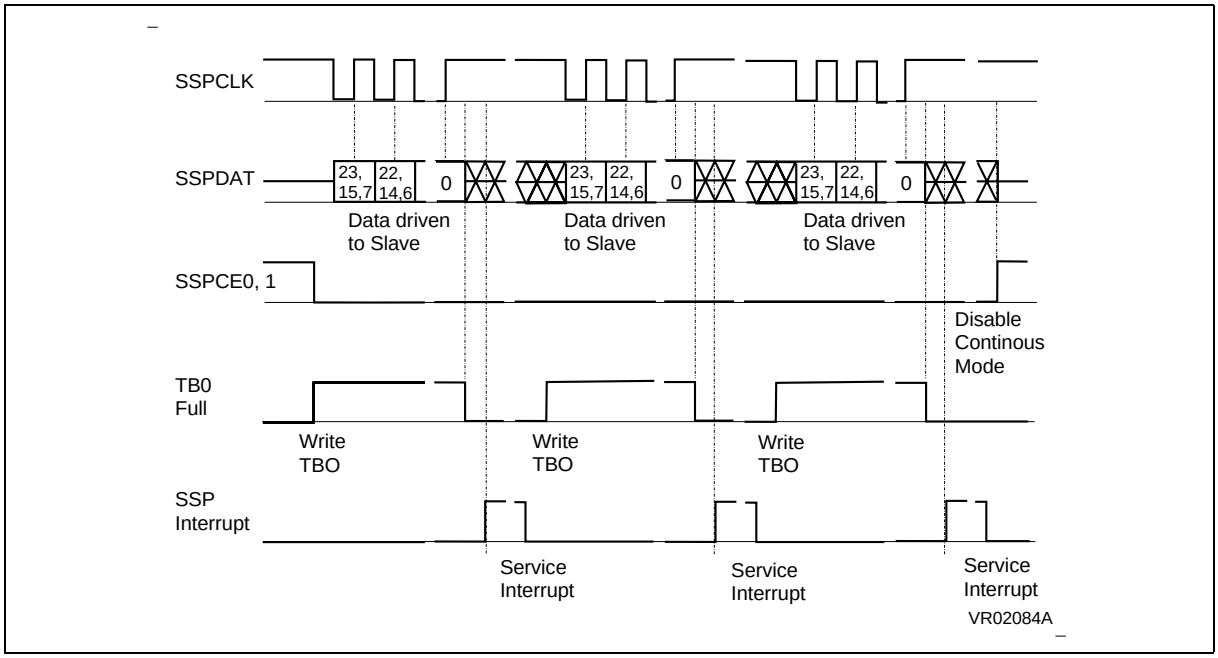


Figure 103 Write operation waveforms in continuous mode

The following figure shows the read operation in continuous mode. Note that the diagram shown starts at the point where the master has written the initial information to the slave and switches the data line SSPDAT from output to input. At the end of a transfer, the byte

received from the slave is stored in register SSPRB0, and an internal flag RB0_Full is set. Reading SSPRB0 through software clears this flag, and issues the next 8 clock pulses to receive the next byte from the slave device. The time between the transfers depends again on the application; the CPU has to react on the interrupt request and read register SSPRB0 in order to start the next transfer. This procedure continues until the mode is switched from continuous mode back to normal mode. The chip enable lines will then be deactivated immediately if no transfer is in progress or after the current transfer is completed.

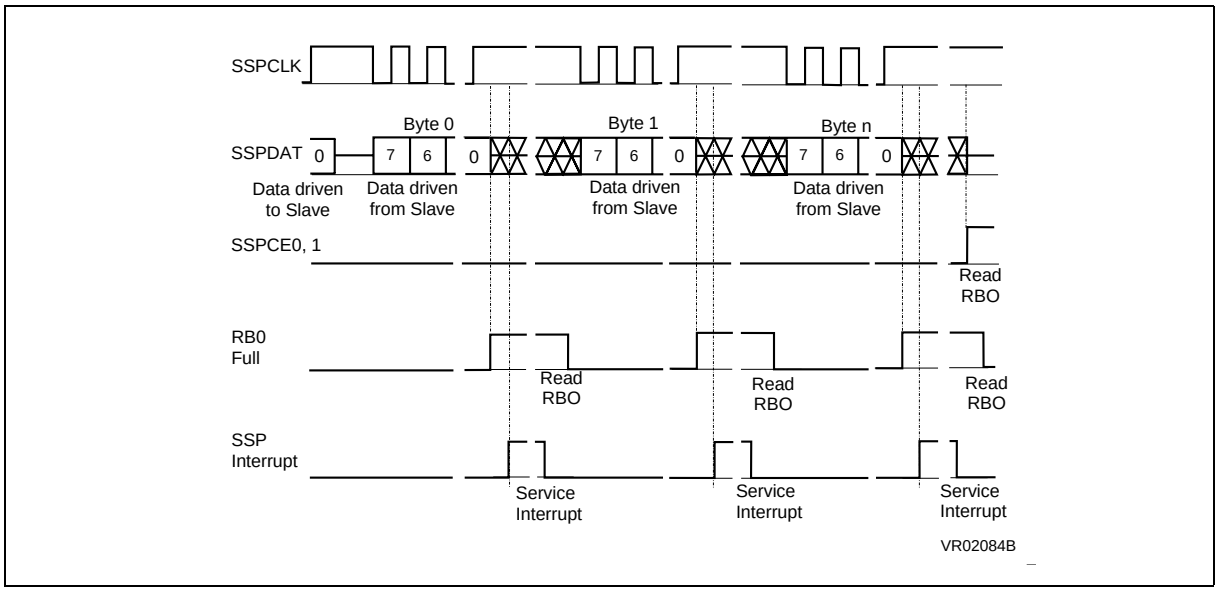


Figure 104 Read operation waveforms in continuous mode

13.2.10 Interrupt control for the SSP

At the end of a transfer in a read or write operation, the interrupt request flag XP1IR in register XP1IC is set. This can cause the following effects:

- an interrupt to the XP1INT interrupt vector,
- trigger a PEC service if the interrupt enable bit XP1IE in register XP1IC is set,
- poll the XP1IR flag by software. Note that when using polling technique, the software must clear the XP1IR flag.

The timing for the interrupt request generation is such that the request bit is set one half bit time after completion of the last data bit time. In reference to the normal mode, this is the same time point where the chip enable line is deactivated. Note that the distinction between a write or a read operation interrupt must be performed through software.

XP1IC (F18Eh) **ESFR** **Reset Value: - - 00h**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								XP1IR	XP1IE	ILVL				GLVL	
-	-	-	-	-	-	-	-	rw	rw	rw				rw	

Note *Refer to “Interrupt control registers” on page 86 for an explanation of the control fields.*

13.2.11 SSP input/output pins

The SSP is connected to the external world by the following four signals on Port 4:

SSP Signal	Port Pin	Function
SSPCLK	P4.7	Clock Line of the SSP.
SSPDAT	P4.6	Data Input/Output Line of the SSP.
SSPCE0	P4.5	Chip Enable Line 0 of the SSP.
SSPCE1	P4.4	Chip Enable Line 1 of the SSP.

Table 42 SSP input/output pins

Note *These SSP signals are only available on the Port 4 pins, if Port 4 is not programmed to output all 8 segment address lines. Select 0, 2 or 4 segment address lines (at start-up configuration during reset) if the SSP is to be used.*

13.2.12 Accessing the on-chip SSP

The SSP is accessed like an external memory or peripheral. That means that the registers of the SSP can be read and written using 16-bit or 8-bit direct or indirect MEM addressing modes. Since the XBUS, to witch the SSP is connected, also represents the external bus, SSP accesses follow the same rules and procedures as accesses to the external bus. SSP accesses cannot be executed in parallel to external instruction fetches or data read/writes, but are arbitrated and inserted into the external bus access stream.

However, the on-chip SSP is accessed via the 16-bit demultiplexed bus mode exclusively. This provides the user the fastest access to the SSP registers.

When accessing the on-chip SSP, the bus has to be switched to the appropriate bus mode within the SSP address range. This is done via a specific register-pair, XBCON1 and XADRS1, which is similar to the BUSCONx/ADDRSELx register-pairs. With the XBCON1 register (XBUS Bus Configuration Register 1), the bus mode, number of waitstates, etc., for accessing the SSP are controlled. The XADRS1 register (XBUS Address Select Register 1)

is used to specify the SSP address area. Contrary to the BUSCONx/ADDRSELx registers, the XBCON1/XADRS1 registers are mask-programmed, i.e. they are not software programmable.

The mask-programming of these registers is:

XBCON1:	04BFh
XADRS1:	0EF0h

The XBCON1 register is organized like the BUSCONx registers except that there is no option for read/write chip selects (bits 14 and 15). With the mask-programmed value shown above, the following options are selected:

XRDYEN:	0	READY disabled
XBUSACT:	1	XBUS active
XALECTL:	0	No ALE Lengthening
XBTYP:	10	16-bit DEMUX Bus
XMTTC:	1	No Tri-State Waitstate
XRWDC:	1	No Read/Write Delay
XMTRC:	1101	0 Waitstate

Note *The XBUSACT bit of register XBCON1 only enables accesses to the SSP, it does not control the **external** bus (as the other BUSACT bits in the BUSCONs).*

The XADRS0 register is organized like the other ADDRSELx registers except that it uses the reduced address ranges, which are defined for XBUS Peripherals. With the mask-programmed value shown above, the following options are selected:

ADDR:	00’EF00h	SSP address area starts at EF00h in segment 0
GSZ:	0000	SSP address area covers 256 bytes

Fixing the SSP address area to address EF00h in segment 0, has the advantage that the SSP is accessible from data-page 3, the ‘system’ data page. This data-page is usually accessed through the ‘system’ data-page pointer DPP3. In this way, the internal addresses, such as SFRs, internal RAM, and the SSP registers, are all located into the same data-page, and form a contiguous address space.

13.2.13 Visibility of accesses to the SSP

An access to the SSP can be made fully visible externally or not. This option is controlled by the bit VISIBLE in register SYSCON. The grade of visibility depends on several conditions described in the following.

13.2.14 Single chip mode

This description does not apply to the ST10R272L ROMless device, and only applies to ST10 devices with internal Flash or ROM.

Single chip mode is entered during reset with pin EA tied to a logic high level. The chip will start running in single chip mode without an external bus.

When bit VISIBLE in register SYSCON is cleared, SSP accesses will be completely hidden. Although the SSP is connected to the XBUS, which is an internal implementation of the external bus, it can be accessed in single chip mode without any restriction. No external bus signal will be generated for an access to the SSP address range, because the XBUSACT bit in register XBCON1 only controls accesses to the SSP via the XBUS, it does **not** control the **external** bus (i.e. Port 0, Port 1, Port 4 and Port 6 can be used for general purpose I/O).

When bit VISIBLE in register SYSCON is set, then accesses to the SSP can be made visible to the external world. To do so, one of the BUSCON registers has to enable a 16-bit demultiplexed external bus (Port 0 and Port 1 cannot be used for general purpose I/O in this case). All accesses to the SSP can be monitored externally, and read/write strobes are generated. The visibility of the segment address depends on the number of segment address lines selected for Port 4.

13.2.15 External bus mode

If an external bus is enabled through one or more of the BUSCON registers, PORT0, PORT1, Port 4 and Port 6 (or parts of them) may be used to control the external bus.

When bit VISIBLE in register SYSCON is cleared, accesses to the SSP will be partly reflected on the external bus. Due to severe timing constraints, it is not possible to hide SSP accesses completely, when an external bus is enabled. SSP accesses will be reflected on the external bus in the following manner:

- The ALE signal will be generated in any case.
- No read or write Signals will be generated.
- The data of a read access cannot be seen.
- The visibility of the segment address depends on the number of segment address lines selected for Port 4.

- The SSP is connected to the bus controller via a 16-bit demultiplexed bus. However, address and data information of a SSP access will be only on those ports operating in the currently selected bus mode, as shown in the table below.

When bit VISIBLE in register SYSCON is set, full visibility of all SSP accesses is provided **if** a 16-bit demultiplexed bus mode is enabled through one of the BUSCON registers (PORT0 and PORT1 cannot be used for general purpose I/O in this case). All SSP accesses can be monitored externally and read/write strobes are generated. The visibility of the segment address depends on the number of segment address lines selected for Port 4.

Currently Selected Bus Mode	Port	Visible Information of SSP Access
8-bit Multiplexed	P0L	Low Byte of SSP Write Data
	P0H	High Byte of SSP Write Data
8-bit Demultiplexed	P0L	Low Byte of SSP Write Data (High byte not visible)
	PORT1	16-bit SSP Address
16-bit Multiplexed	PORT0	16-bit SSP Write Data
16-bit Demultiplexed	PORT0	16-bit SSP Write Data
	PORT1	16-bit SSP Address

Table 43 SSP visibilty

13.2.16 Accessing the SSP in hold mode

When the ST10R272L is placed into hold mode by an external HOLD request, accesses to external memory or peripherals have to wait until the HOLD request is deactivated. SSP accesses, however, can be executed in this mode if bit VISIBLE in register SYSCON is cleared. In this case, an access to the SSP is completely invisible to the external world. If bit VISIBLE is set, then SSP accesses have to wait until the external HOLD request is removed.

Note *SSP accesses during HOLD mode are blocked after any attempt to access an external location, even if the VISIBLE bit is cleared. The initiated external access must be finished first, which requires the HOLD condition to be removed.*

13.2.17 Power down mode

When the ST10R272L enters Power Down Mode, the XCLK (XBUS Clock) signal is turned off and SSP operation stops. Any transfer operation in progress is interrupted.

After returning from Power Down Mode via hardware reset, the SSP must be reconfigured.

14 WATCHDOG TIMER

The Watchdog Timer is a fail-safe mechanisms which prevents the controller from malfunctioning over a long period of time.

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

The Watchdog Timer is a 16-bit timer, clocked with the system clock divided either by 2 or by 128. The high byte of the watchdog timer register can be set to a pre-specified 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. The lower byte of the watchdog timer register is reset on each service access.

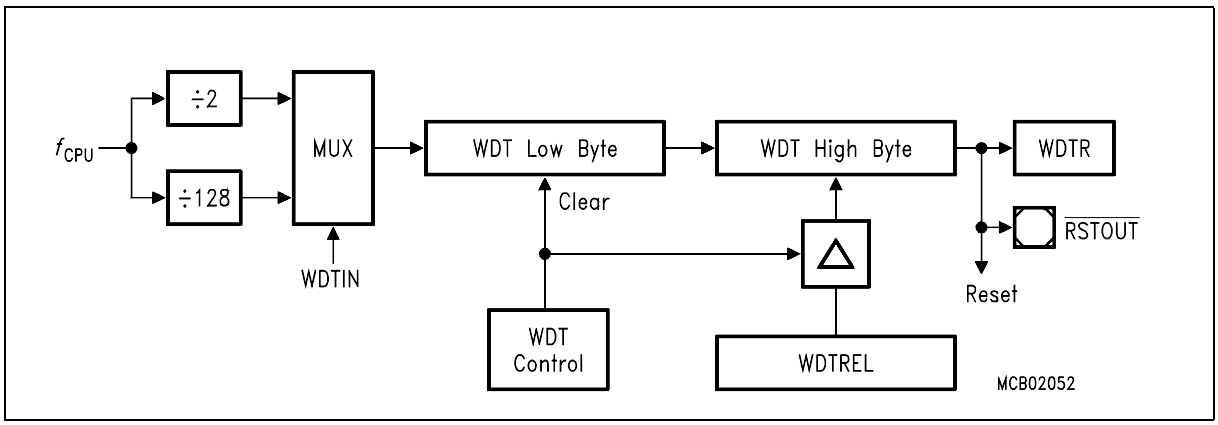


Figure 105 Watchdog timer block diagram

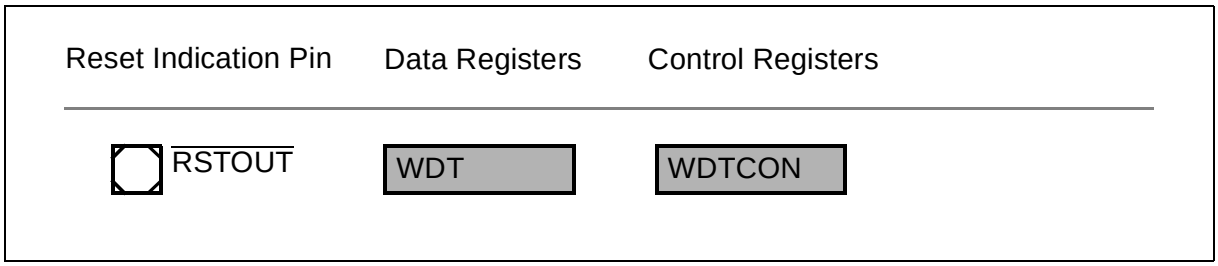


Figure 106 SFRs and port pins associated with the watchdog timer

14.1 Watchdog timer operation

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.

After any software reset, external hardware reset (see note), or watchdog timer reset, the watchdog timer is enabled and starts counting up from 0000h 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 by the instruction SRVWDT by the time the count reaches FFFFh 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.

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 time register WDT with the preset value in bit field 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.

WDTCON (FFAEh / D7h) **SFR** **Reset Value: 000Xh**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WDTREL								-	-	-	-	-	-	WDTR	WDTIN
rw								-	-	-	-	-	-	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 a hardware reset or by the SRVWDT instruction.
WDTREL	**Watchdog Timer Reload Value (for the high byte)**

Note *The reset value will be 0002h, if the reset was triggered by the watchdog timer (overflow). It will be 0000h otherwise.*

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

- The watchdog timer input frequency can be selected by 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:

$$P_{WDT} = \frac{2^{(1 + <WDTIN>*6)} * (2^{16} - <WDTREL> * 2^{8})}{f_{CPU}}$$

Refer to the device datasheet for a table of watchdog timer ranges.

15 SYSTEM RESET

The internal system reset function initializes the ST10R272L into a defined default state. It is invoked either by asserting a hardware reset signal on pin $\overline{\text{RSTIN}}$ (hardware reset input), upon the execution of the SRST instruction (software reset) or by an overflow of the watchdog timer.

Whenever one of these conditions occurs, the microcontroller is reset into its predefined default state through an internal reset procedure (warm hardware reset, software and watchdog resets) or asynchronously with $\overline{\text{RSTIN}}$ (asynchronous reset).

The asynchronous reset condition is defined by a low level on RPD/Vpp pin while $\overline{\text{RSTIN}}$ pin is asserted. Warm hardware reset (synchronized to the CPU clock) is defined by a high level on RPD/Vpp pin. As long as the $\overline{\text{RSTIN}}$ pin is asserted, a weak internal pull-down is turned on the RPD/Vpp pin.

When an asynchronous reset is initiated, the microcontroller is immediately (asynchronously) reset into its predefined default state, and does not require a stabilized clock signal on the XTAL1 pin. When this asynchronous reset condition is removed, the microcontroller starts program execution from the memory location 00’0000h in code segment zero.

When a reset other than asynchronous reset is initiated, pending internal hold states are cancelled and the current internal access cycle (if any) is completed. External bus cycles are aborted, except for a watchdog reset. Then the internal reset sequence starts, the bus pin drivers and the I/O pin drivers are switched off (tristate), and the PORT0 pins are internally pulled high. The $\overline{\text{RSTIN}}$ pin is driven low for 512 CPU clock cycles which is the internal reset sequence duration. $\overline{\text{RSTOUT}}$ is activated and remains active until the execution of the EINIT instruction. The CPU and peripherals are set in their predefined default state.

The internal reset condition is active at least for the duration of the reset sequence and then until the $\overline{\text{RSTIN}}$ input is inactive. When this internal reset condition is removed, the reset configuration is latched from PORT0, and pins ALE, $\overline{\text{RD}}$ and $\overline{\text{WR}}$ are driven to their inactive levels.

After the internal reset condition is removed, the microcontroller starts program execution from memory location 00’0000h 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.

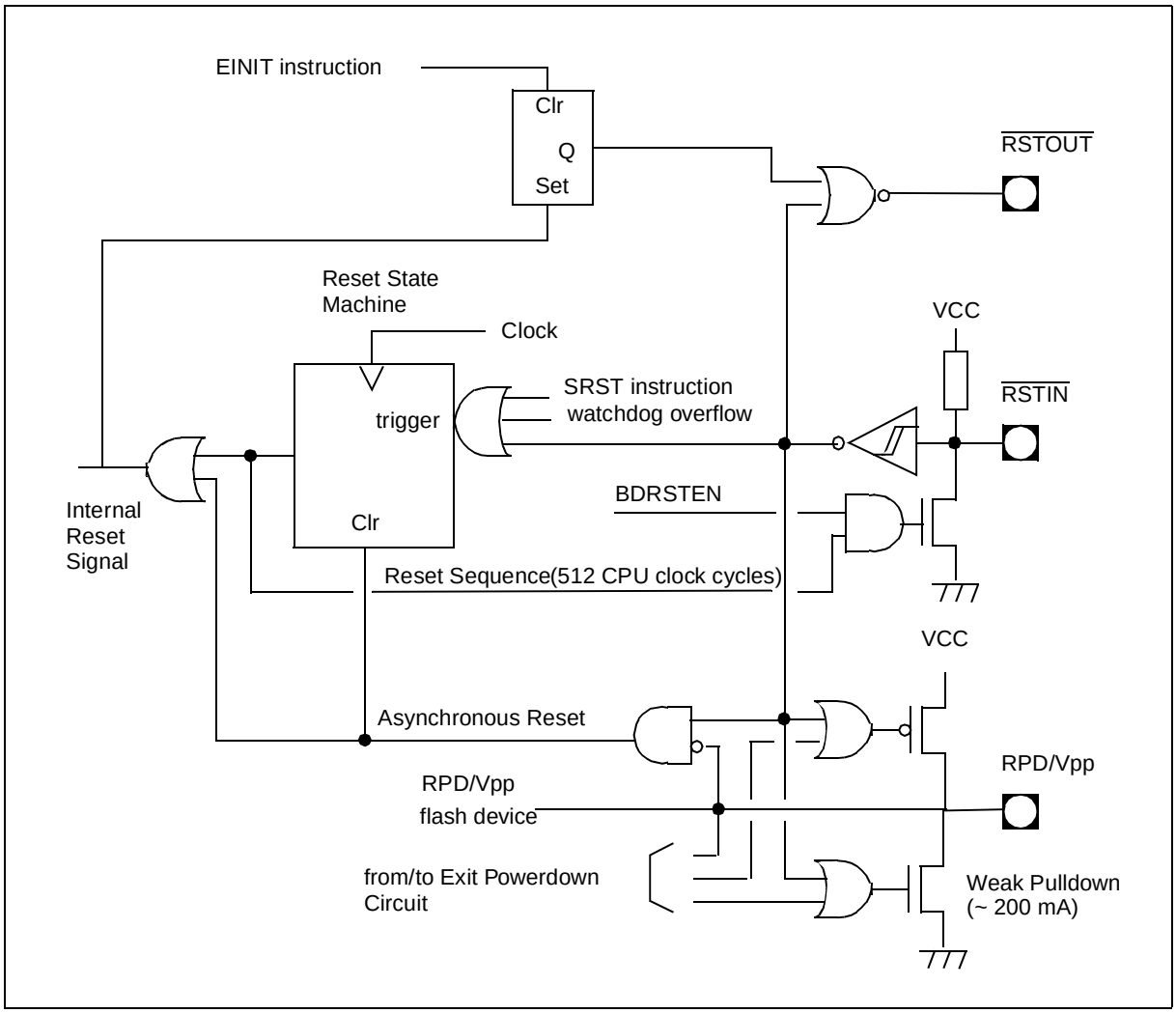


Figure 107 Internal reset circuitry

15.1 Asynchronous hardware reset

Asynchronous reset does not required a stabilized clock signal on XTAL1 as it is not internally resynchronized but immediately resets the microcontroller into its default reset state. This asynchronous reset is required on power-up of the chip and may be used during catastrophic situations. Note that rising edge of $\overline{\text{RSTIN}}$ pin is internally resynchronized before exiting the reset condition. Therefore, only the entry of the this hardware reset is asynchronous.

An asynchronous hardware reset is triggered when a low logic level on $\overline{\text{RSTIN}}$ and RPD/Vpp pins is detected. To ensure a correct power-on reset, the $\overline{\text{RSTIN}}$ and RPD/Vpp pins must be

held low until the CPU clock signal is available (about 10...50 ms to allow the on-chip oscillator to stabilize).

The input $\overline{\text{RSTIN}}$ provides an internal pullup device equalling a resistor of 50 Kohms to 150 KOhms (the minimum reset time must be determined by the lowest value). The RPD/Vpp pin provides an internal pulldown device that causes the external capacitor C2 to discharge at a typical rate of 200 µA. Simply connecting two external capacitors, one to $\overline{\text{RSTIN}}$ pin and the other to RPD/Vpp pin is sufficient for an automatic power-on reset. $\overline{\text{RSTIN}}$ may also be connected to the output of other logic gates. When reset sequence is finished, the RPD/Vpp capacitor will be charged by external pulldown or by internal pullup device if the bit PWDCFG is set in SYSCON register.

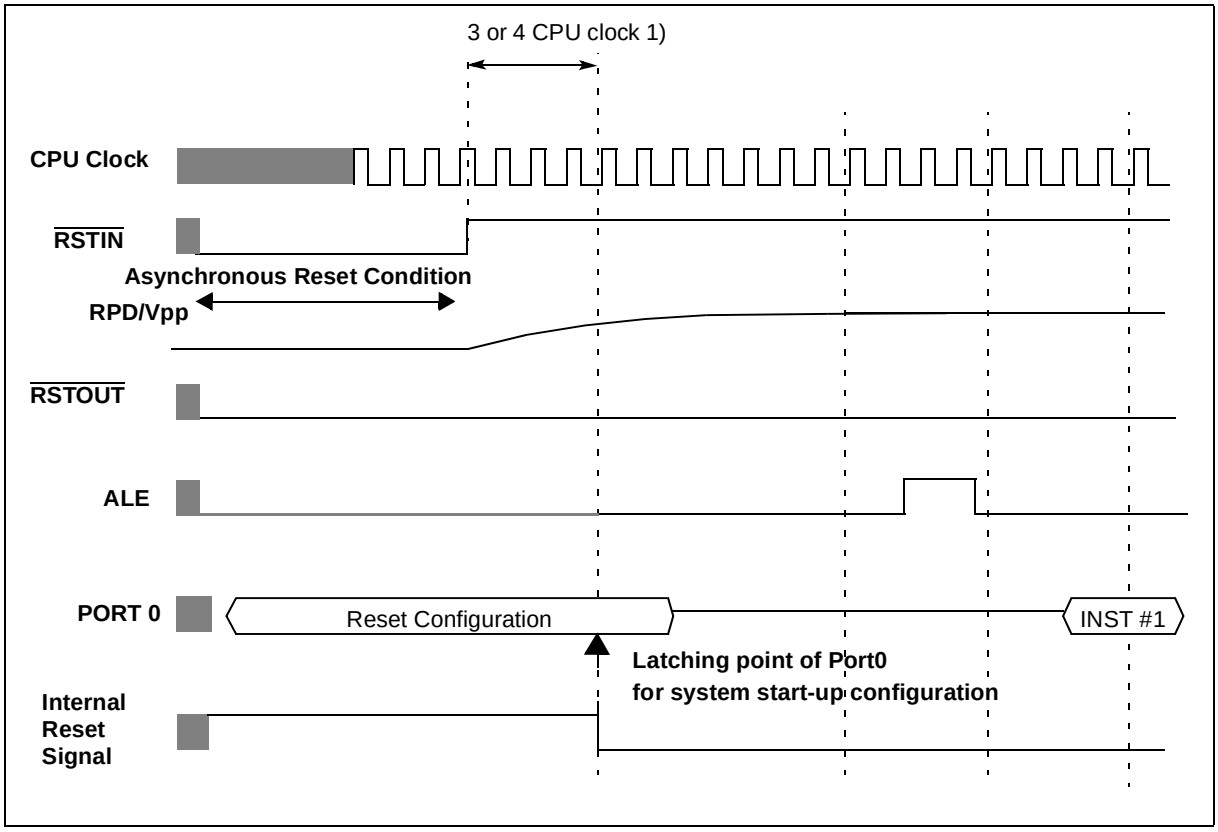


Figure 108 Asynchronous reset timing

1 $\overline{\text{RSTIN}}$ rising edge to internal latch of Port0 is 3 CPU clock cycles if the PLL is bypassed and the prescaler is on ($f_{CPU} = f_{XTAL} / 2$), else it is 4 CPU clock cycles

15.2 Synchronous hardware reset (warm reset)

A warm synchronous hardware reset is triggered when the reset input signal $\overline{\text{RSTIN}}$ is latched low and the RPD/Vpp pin is high. To ensure the recognition of the $\overline{\text{RSTIN}}$ signal (latching), it must be held low for at least 2 CPU clock cycles. Shorter $\overline{\text{RSTIN}}$ pulses may trigger a hardware reset, if they coincide with the latch's sample point.

The I/Os are immediately (asynchronously) set in high impedance, $\overline{\text{RSTOUT}}$ is driven

After $\overline{\text{RSTIN}}$ negation is detected, a short transition period (approximately 6 CPU clock cycles) elapses, during which pending internal hold states are cancelled and the current internal access cycle (if any) is completed.

External bus cycle is aborted. Then, the internal reset sequence starts for 512 CPU clock cycles. During this reset sequence, $\overline{\text{RSTIN}}$ pin is driven low and internal reset signal is asserted to reset the microcontroller in its default state.

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

.

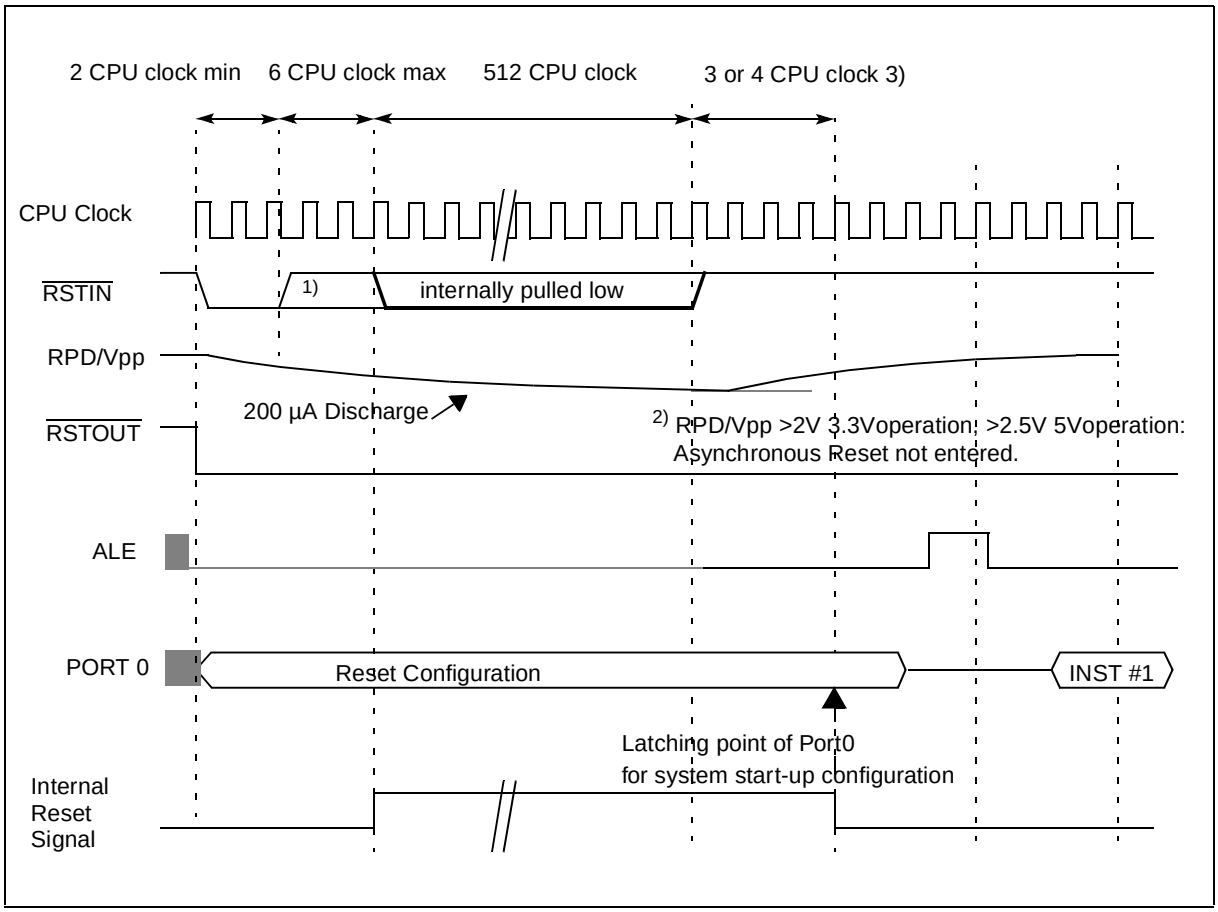


Figure 109 Synchronous warm reset:
external low pulse on $\overline{\text{RSTIN}}$ shorter than reset sequence

1 $\overline{\text{RSTIN}}$ assertion can be released there.

2 If during the reset condition ($\overline{\text{RSTIN}}$ low), RPD/Vpp voltage drops below the threshold voltage (about 2.5V for 5V operation and 2.0V for 3.3V operation), the asynchronous reset is then immediately entered.

$\overline{\text{RSTIN}}$ rising edge to internal latch of Port0 is 3CPU clock cycles if the PLL is bypassed and the prescaler is on ($f_{CPU} = f_{XTAL} / 2$), else it is 4 CPU clock cycles.

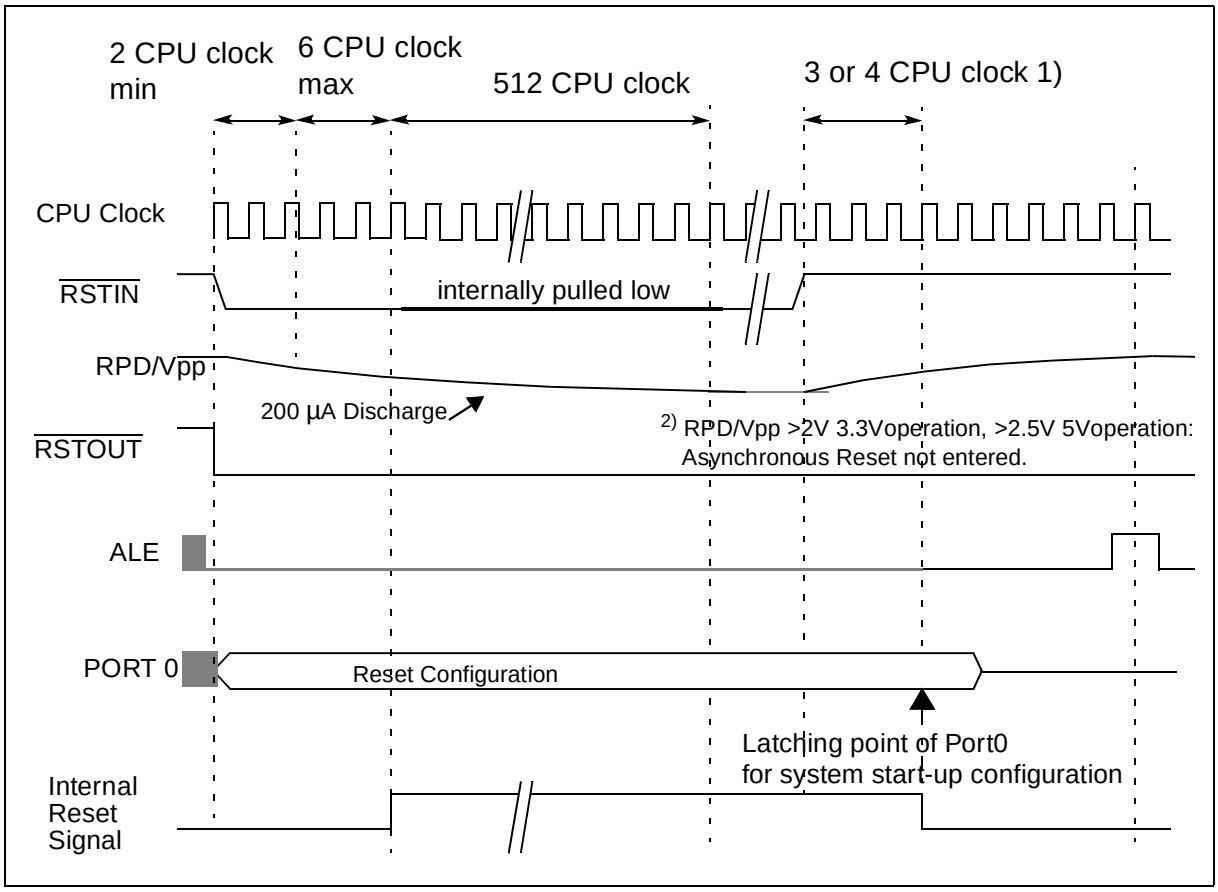


Figure 110 Synchronous warm reset: external low pulse on $\overline{\text{RSTIN}}$ longer than reset sequence

1 $\overline{\text{RSTIN}}$ rising edge to internal latch of Port0 is 3CPU clock cycles if the PLL is bypassed and the prescaler is on ($f_{CPU} = f_{XTAL} / 2$), else it is 4 CPU clock cycles

If during the reset condition ($\overline{\text{RSTIN}}$ low), RPD/Vpp voltage drops below the threshold voltage (about 2.5V for 5V operation and 2.0V for 3.3V operation), the asynchronous reset is then immediately entered.

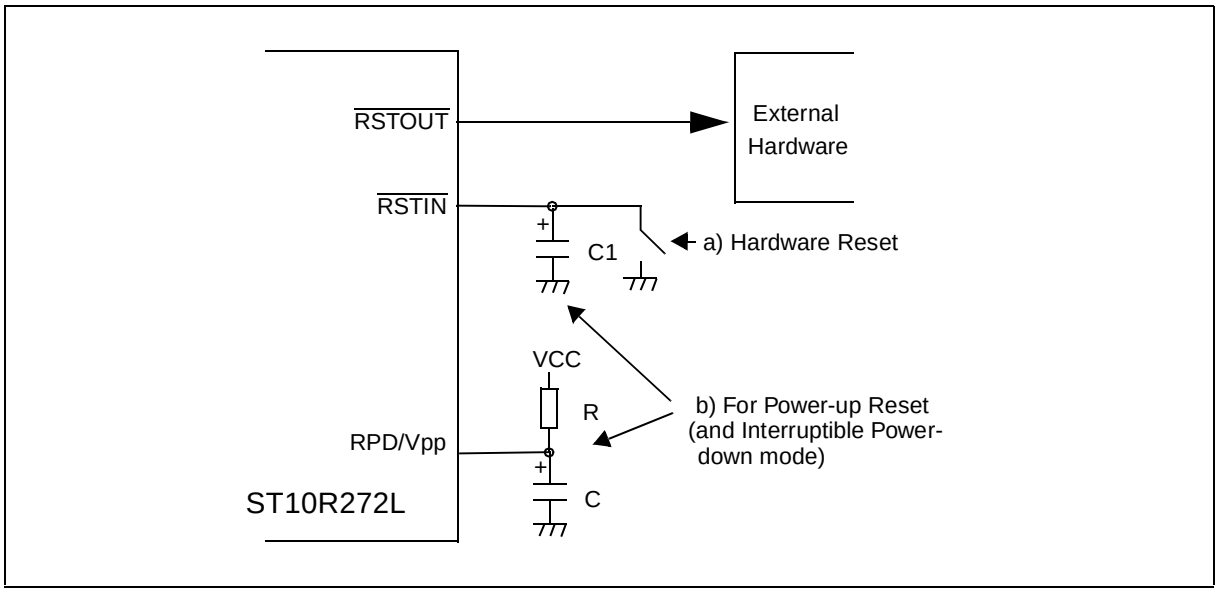


Figure 111 Minimum external reset circuitry

The simplest way to reset the ST10R272L, is to insert a capacitor between $\overline{\text{RSTIN}}$ pin and Vss (C1 in *Figure 107*), plus a capacitor between RPD/Vpp pin and Vss (C2) and a pull-up between RPD/Vpp pin and Vcc. The input $\overline{\text{RSTIN}}$ provides an internal pullup device equalling a resistor of 50-250 kOhms (the minimum reset time must be determined by the lowest value). Select C1 that produce a sufficient discharge time to permit the internal or external oscillator and/or internal PLL to stabilize.

An asynchronous hardware reset is required during power-up, to guarantee correct power-up reset with controlled supply current consumption (especially if the clock signal requires a long stabilization time). It is recommended to connect the external RC circuit shown in *Figure 111* to the RPD/Vpp pin. On power-up, the logical low level on RPD/Vpp pin forces an asynchronous hardware reset when $\overline{\text{RSTIN}}$ is asserted. The external pull-up will then charge the capacitor C. Note that an internal pull-down device on RPD/Vpp pin is turned on when $\overline{\text{RSTIN}}$ pin is low, and causes the external capacitor (C) to begin discharging at a typical rate of 100µA to 200 µA. With this mechanism, after power-up reset, short low pulses applied on $\overline{\text{RSTIN}}$ produce synchronous hardware reset, but if $\overline{\text{RSTIN}}$ is asserted longer than the time needed for C to be discharged by the internal pull-down device, then the hardware reset starts in a synchronous manner if a clock is applied, but is then forced in an asynchronous manner. This mechanism insures recovery from very catastrophic failure.

The external C capacitor on RPD/Vpp pin is also used to exit from Powerdown mode with external interrupt. In this case, select components that produce a sufficient discharge time to permit the internal or external oscillator circuitry to stabilize. If exiting powerdown mode is only done by hardware reset, the capacitor C must be selected by determining the

worst-case discharge time needed for the internal or external oscillator to stabilize. See “Selecting R and C values” on page 300 for the formula to calculate an appropriate value for C. The value for R resistor must not interfere with the discharge current of the internal pulldown device on RPD/Vpp pin. An R value between 200 kohms and 1Mohms should be suitable.

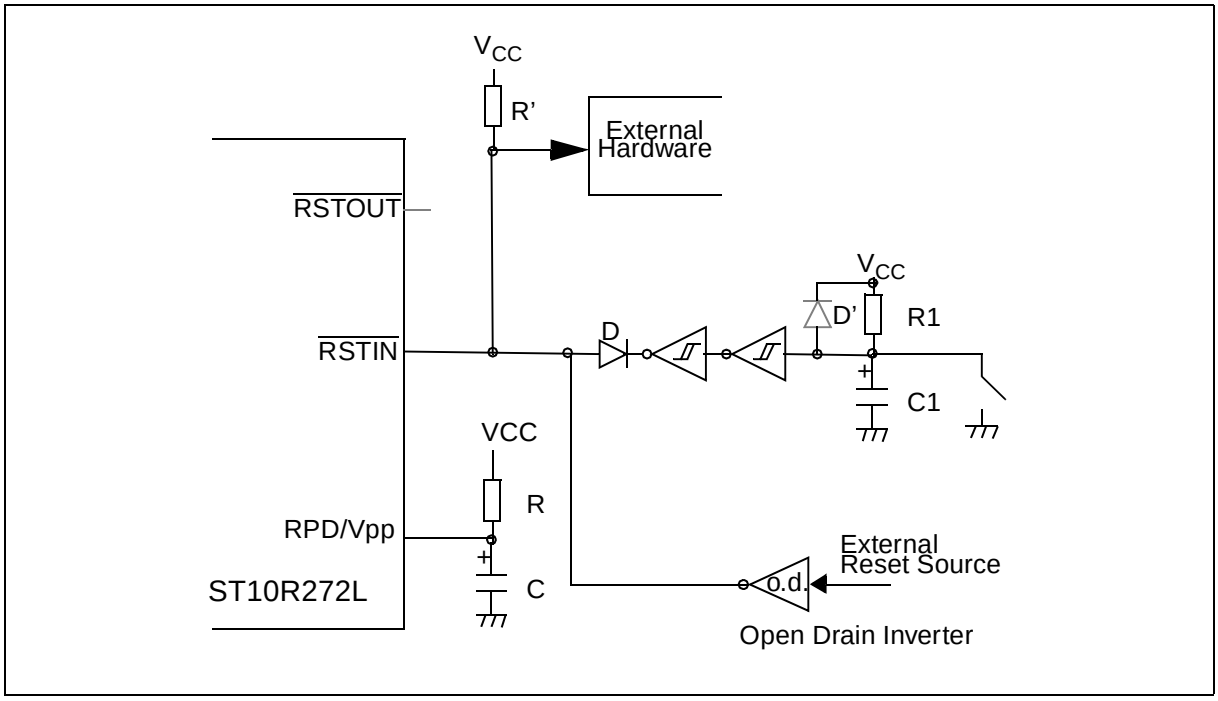


Figure 112 System reset circuitry (example)

The minimum reset circuit of *Figure 111* is not adequate when the $\overline{\text{RSTIN}}$ pin is driven from an external device, because of the capacitor C1 that will keep the voltage above V_{IL} for short low pulses applied on $\overline{\text{RSTIN}}$ pin.

Figure 10-5 shows an example of a system-reset circuit. In this example, R1C1 external circuit is only used to generate power-up or manual reset, and RC circuit on RPD/Vpp is used for power-up reset and to exit from powerdown mode. Diode D creates a wired-OR gate connection to the reset pin and may be replaced by open-collector schmitt trigger buffer. Diode D’ provides a faster cycle time for repetitive power-on resets.

R’ is an optional pull-up for faster recovery and correct biasing of TTL Open Collector drivers.

15.3 Software reset

The reset sequence can be triggered at any time via the protected instruction SRST (Software Reset). This instruction can be executed deliberately within a program, e.g. to leave bootstrap loader mode, or upon a hardware trap that reveals a system failure.

As for a Synchronous (warm) Hardware Reset, the reset sequence lasts 512 CPU clock cycles, and drives low the $\overline{\text{RSTIN}}$ pin.

Note *A software reset disregards the configuration of P0L.5...P0L.0, and does not reload RP0H register with PORT0 values.*

15.4 Watchdog timer reset

When the watchdog timer is not disabled during the initialization or serviced regularly during program execution is will overflow and trigger the reset sequence. Other than hardware and software reset the watchdog reset completes a running external bus cycle if this bus cycle either does not use $\overline{\text{READY}}$ at all, or if $\overline{\text{READY}}$ is sampled active (low) after the programmed waitstates. When $\overline{\text{READY}}$ is sampled inactive (high) after the programmed waitstates the running external bus cycle is aborted. Then the internal reset sequence is started.

Note *A watchdog reset disregards the configuration of P0L.5...P0L.0, and does not reload RP0H register with PORT0 values*
The watchdog reset cannot occur while the ST10R272L is in bootstrap loader mode!

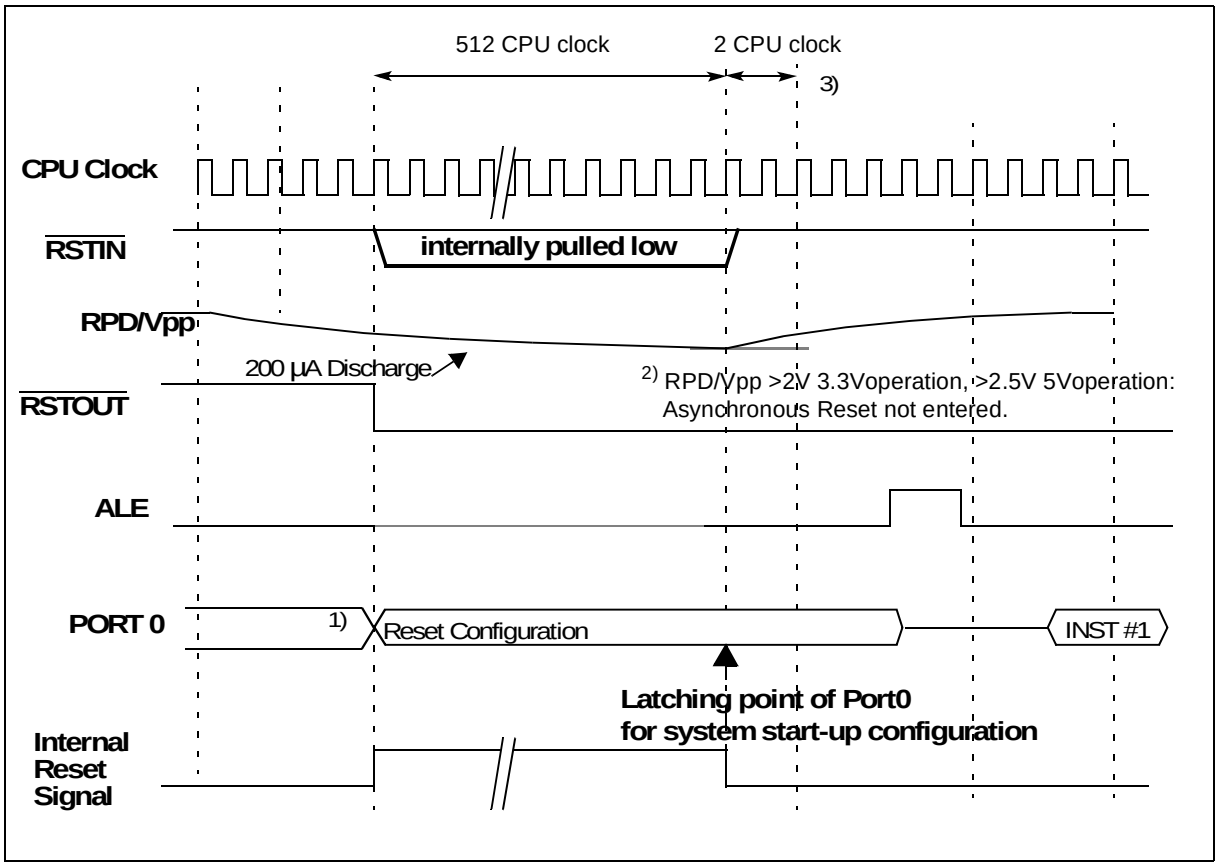


Figure 113 Software or watchdog reset

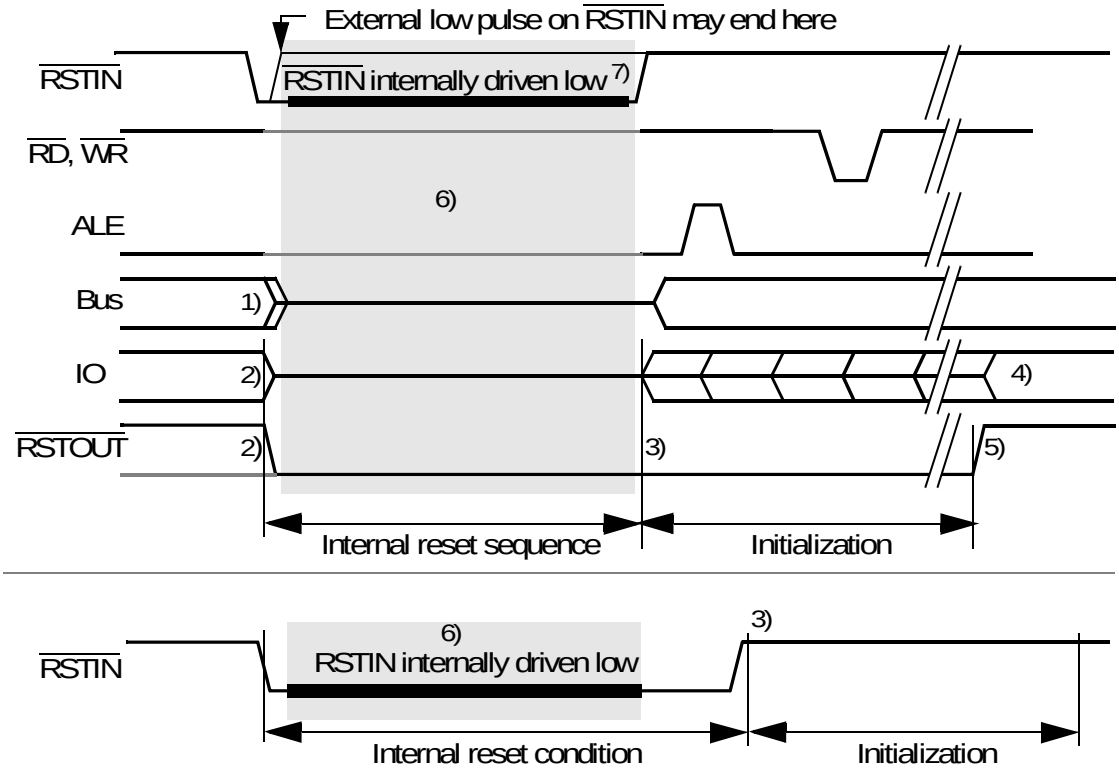
1 Current external bus cycle is completed for Watchdog reset only.

2 If during the reset condition ($\overline{\text{RSTIN}}$ low), RPD/Vpp voltage drops below the threshold voltage (about 2.5V for 5V operation and 2.0V for 3.3V operation), the asynchronous reset is then immediately entered.

3 RSTIN must be high at this point or a hardware reset sequence will be triggered.

15.5 Pins after reset

After the reset sequence the different groups of pins of the ST10R272L are activated in different ways depending on their function. Bus and control signals are activated immediately after the reset sequence according to the configuration latched from PORT0, so either external accesses can takes place or the external control signals are inactive. The general purpose IO pins remain in input mode (high impedance) until reprogrammed via software (see figure below). The $\overline{\text{RSTOUT}}$ pin remains active (low) until the end of the initialization routine.



When the internal reset condition is prolonged 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 asynchronously with RSTIN, synchronously upon software or watchdog reset.
3. The reset condition ends here. The ST10R272L 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. The RSTIN pin is not internally pulled low during asynchronous hardware reset.

Figure 114 Reset input and output signals

15.5.1 RESET input pin

The $\overline{\text{RSTIN}}$ pin is a bi-directional pin. To reset the microcontroller, the $\overline{\text{RSTIN}}$ pin must be held low for at least 2 CPU clock cycles for a warm (synchronous) hardware reset. After $\overline{\text{RSTIN}}$ negation is internally resynchronized, a short transition period (approximately 6 CPU clock cycles) elapses, then the microcontroller turns on a pull-down transistor on $\overline{\text{RSTIN}}$ pin

for the 512 CPU clock cycles of the internal reset sequence. After the reset sequence has been completed, the RSTIN input is sampled. When the reset input signal is active at that time, the internal reset signal is prolonged until RSTIN gets inactive, but the reset sequence is not re-triggered.

During reset, internal pullup devices are active on the PORT0 lines, so that external pulldown devices can be used to defined system start-up configuration. The values of PORT0 lines are latched when RSTIN is internally latched high.

15.5.2 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 to completely configure the controller including its on-chip peripheral units before releasing the reset signal for the external peripherals of the system.

15.6 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, and its default reload value is 00h, so a watchdog timer overflow will occur 131072 CPU clock cycles 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.

15.7 Registers reset values

During the reset sequence the registers 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:	0001h (points to data page 1)
DPP2:	0002h (points to data page 2)

DPP1:	0001h (points to data page 1)
DPP3:	0003h (points to data page 3)
CP:	FC00h
STKUN:	FC00h
STKOV:	FA00h
SP:	FC00h
WDTCON:	0002h, if reset was triggered by a watchdog timer overflow, 0000h otherwise
S0RBUF:	XXh (undefined)
SSCRB:	XXXXh (undefined)
SYSCON:	0XX0h (set according to reset configuration)
BUSCON0:	0XX0h (set according to reset configuration)
RP0H:	XXh (reset levels of P0H)
ONES:	FFFFh (fixed value)

15.8 Internal RAM after reset

The contents of the internal RAM are not affected by a system resynchronized reset. However, after a power-on reset, the contents of the internal RAM are undefined. This implies that the GPRs (R15...R0) and the PEC source and destination pointers (SRCP7...SRCP0, DSTP7...DSTP0) which are mapped into the internal RAM are also unchanged after a warm reset, software reset or watchdog reset, but are undefined after a power-on reset.

Note Content of the internal RAM may be affected by a warm asynchronous reset.

15.9 Ports and external bus configuration during reset

During the internal reset sequence all of the ST10R272L'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 ST10R272L 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.Pin $\overline{EA}$ must be held at '0' level.

- 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

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.

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

15.10 Application specific initialization routine

After the internal reset condition is removed the ST10R272L fetches the first instruction from location 00’0000h, which is the first vector in the trap/interrupt vector table, the reset vector. 4 words (locations 00’0000h through 00’0007h) 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.

After reset, it may be desirable to reconfigure the external bus characteristics, because the SYSCON register is initialized during reset to the slowest possible memory configuration.

To decrease the number of instructions required to initialize the ST10R272L, each peripheral is programmed to a default configuration upon reset, but is disabled from operation. These default configurations can be found in the descriptions of the individual peripherals.

During the software design phase, portions of the internal memory space must be assigned to register banks and system stack. When initializing the stack pointer (SP) and the context pointer (CP), it must be ensured that these registers are initialized before any GPR or stack operation is performed. This includes interrupt processing, which is disabled upon completion of the internal reset, and should remain disabled until the SP is initialized.

Note Traps (incl. $\overline{NMI}$) 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’FC00h, while the STKOV register contains 00’FA00h. With the default reset initialization,

256 words of system stack are available, where the system stack selected by the SP grows downwards from 00'FBFEh, while the register bank selected by the CP grows upwards from 00'FC00h.

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 (see note) and causes the $\overline{\text{RSTOUT}}$ pin to go high. This signal can be used to indicate the end of the initialization routine and the proper operation of the microcontroller to external hardware.

Note *All configurations regarding register SYSCON (enable CLKOUT, stacksize, etc.) must be selected before the execution of EINIT.*

15.10.1 System start-up configuration

Although most of the programmable features of the ST10R272L are either selected during the initialization phase or repeatedly during program execution, there are some features that must be selected earlier, because they are used for the first access of the program execution (e.g. external bus configuration).

These selections are made during reset via the pins of PORT0, which are read during 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. The coding of the selections, as shown below, allows in many cases to use the default option, i.e. high level.

The value on the upper byte of PORT0 (P0H) is latched into register RP0H upon hardware reset, the value on the lower byte (P0L) in conjunction with the value of $\overline{\text{EA}}$ pin directly influences the SYSCON and BUSCON0 registers (bus mode) or the internal control logic of the ST10R272L.

The pins that control the operation of the internal control logic (P0L.0, P0L.1) and the reserved pins are evaluated only during a hardware triggered reset sequence. The pins that influence the configuration of the ST10R272L (P0L.6, P0L.7 and P0H.0) are evaluated during any reset sequence, i.e. also during software and watchdog timer triggered resets.

The configuration via P0H is latched only during hardware reset in register RP0H for subsequent evaluation by software. Register RP0H is described in chapter “The External Bus Interface”.

Note *The reserved pins (marked “**R**”) must remain high during reset in order to ensure proper operation of the ST10R272L. The load on those pins must be small enough for the internal pullup device to keep their level high, or external pullup devices must ensure the high level.*
The pins marked “X” should be left open for ST10R272L devices that do not use them.

The following sections describe the different reset configuration options. The default modes refer to pins at high level, i.e. without external pulldown devices connected. The above note on reserved pins remains applicable.

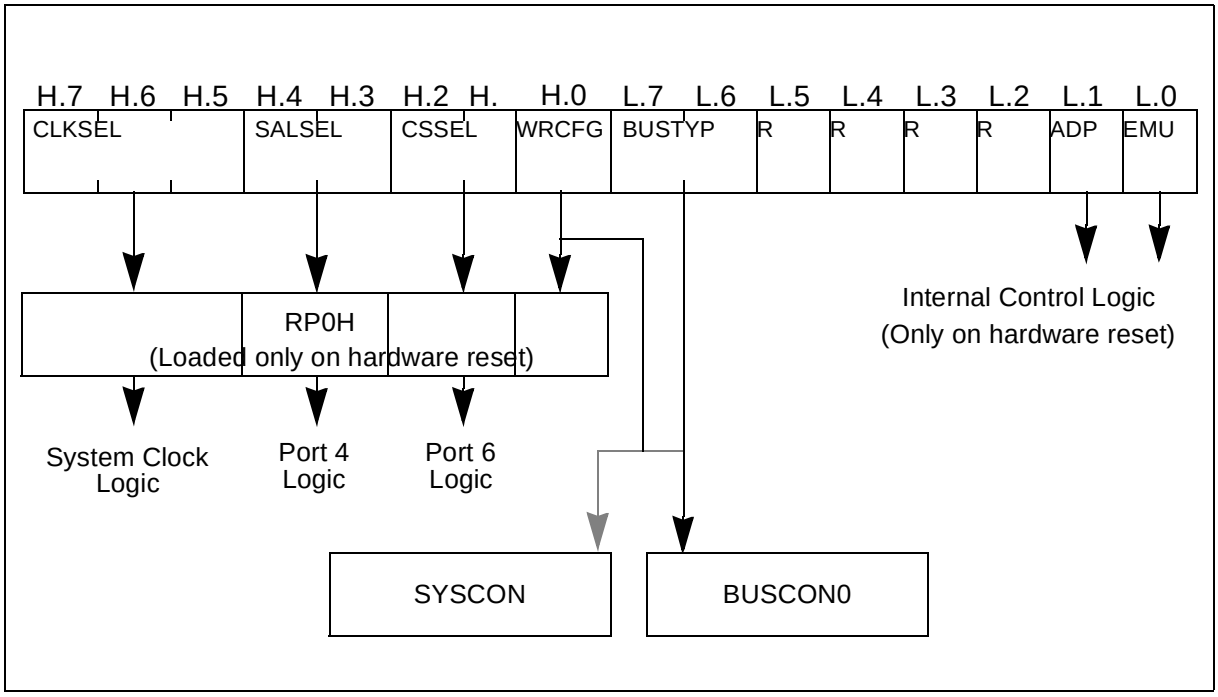


Figure 115 PORT0 configuration during reset

15.10.2 Emulation mode

When low during reset, Pin P0L.0 (EMU) selects emulation mode. This mode accesses the integrated XBUS peripherals by the external bus interface pins in application specific versions of the ST10R272L.

This mode is used for special emulator purposes and is of no use in basic ST10R272L devices, so in this case P0L.0 should be held high.

Default: Emulation mode is off.

15.10.3 Adapt mode

When low during reset, Pin P0L.1 (ADP) selects the adapt mode. In this mode the ST10R272L goes into a passive state, which is similar to its state during reset. The pins of the ST10R272L float to tristate or are deactivated via internal pullup/pulldown devices, as described for the reset state. In addition also the $\overline{\text{RSTOUT}}$ pin floats to tristate rather than be driven low, and the on-chip oscillator is switched off.

This mode allows to switch a ST10R272L that is mounted to a board virtually off, so an emulator may control the board's circuitry, even though the original ST10R272L remains in its place. The original ST10R272L also may resume to control the board after a reset sequence with P0L.1 high.

Default: Adapt mode is off.

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

15.10.4 System clock configuration

Pins P0H.7 to P0H.5 (CLKSEL) selects the system clock configuration at reset. The system clock (CPU Clock) can be selected to be 0.5, 1, 2, 2.5, 3, 4 or 5 times the externally applied frequency at the XTAL-pins. The required system configuration setups are described in "System clock generator" on page 17.

15.10.5 External bus type

Pins P0L.7 and P0L.6 (BUSTYP) select the external bus type during reset. This allows to configure the external bus interface of the ST10R272L 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.

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 ***F****or a ROMless device, pin $\overline{EA}$ must be connected to ground (external start).*

15.10.6 Chip select lines

Pins P0H.2 and P0H.1 (CSSEL) define the number of active chip select signals during reset. This allows to control which pins of Port 6 drive external $\overline{CS}$ signals and which are used for general purpose IO. The two bits are latched in register RP0H.

Default: All 5 chip select lines active ($\overline{CS4}$...$\overline{CS0}$).

Note *The selected number of $\overline{CS}$ signals cannot be changed via software after reset.*

15.10.7 Segment address lines

Pins P0H.4 and P0H.3 (SALSEL) define the number of active segment address lines during reset. This allows to control 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.

Even if not all segment address lines are enabled on Port 4, the ST10R272L internally uses its complete 24-bit addressing mechanism. This allows to restrict 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.*

15.10.8 $\overline{BHE}$ pin configuration

Pin P0H.0 defines the write configuration control (bit WRCFG of SYSCON register). If P0H.0 is pulled down during reset, the bit WRCFG is set to ‘1’ and the pin BHE is configured as $\overline{WRH}$ (Write High Byte) while pin $\overline{WR}$ is configured as WRL (Write Low Byte). Default: pins $\overline{WR}$ and $\overline{BHE}$ retain their normal functions.

16 REGISTER SET

This section summarizes the ST10R272L registers, 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: both by name and hexadecimal address (except for GPRs).

16.1 Register description format

In the following chapters the function and the layout of the SFRs is described in a specific format. The example below shows how to interpret the format.

A word register looks like this:

Elements:

REG_NAME	Name of this register
A16 / A8	Long 16-bit address / Short 8-bit address
E/SFR	Register space (SFR or ESFR)
(* *) * *	Register contents after reset
	0/1: defined value, X: undefined, U: unchanged (undefined after power up)
hwbit	Bits that are set/cleared by hardware are marked with a shaded access box

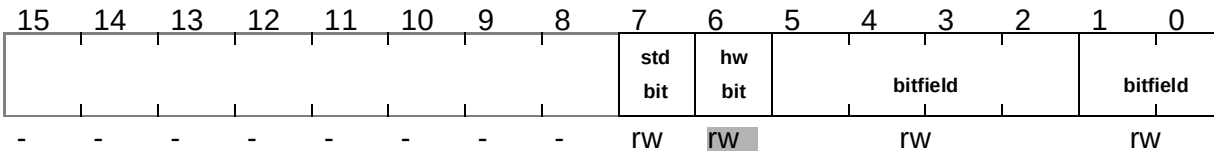
REG_NAME (A16h / A8h) **E/SFR** **Reset Value: * * * *h**

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 (A16h / A8h) **E/SFR** **Reset Value: - - * *h**



16.2 General purpose registers (GPRS)

The GPRs form the register bank that the CPU works with. This register bank may be located anywhere within the internal RAM via the Context Pointer (CP). Due to the addressing mechanism, GPR banks can only reside within the internal RAM. All GPRs are bit-addressable.

Name	Physical Address	8-Bit Address	Description	Reset Value
R0	(CP) + 0	F0h	CPU General Purpose (Word) Register R0	UUUUh
R1	(CP) + 2	F1h	CPU General Purpose (Word) Register R1	UUUUh
R2	(CP) + 4	F2h	CPU General Purpose (Word) Register R2	UUUUh
R3	(CP) + 6	F3h	CPU General Purpose (Word) Register R3	UUUUh
R4	(CP) + 8	F4h	CPU General Purpose (Word) Register R4	UUUUh
R5	(CP) + 10	F5h	CPU General Purpose (Word) Register R5	UUUUh
R6	(CP) + 12	F6h	CPU General Purpose (Word) Register R6	UUUUh
R7	(CP) + 14	F7h	CPU General Purpose (Word) Register R7	UUUUh
R8	(CP) + 16	F8h	CPU General Purpose (Word) Register R8	UUUUh
R9	(CP) + 18	F9h	CPU General Purpose (Word) Register R9	UUUUh
R10	(CP) + 20	FAh	CPU General Purpose (Word) Register R10	UUUUh
R11	(CP) + 22	FBh	CPU General Purpose (Word) Register R11	UUUUh
R12	(CP) + 24	FCh	CPU General Purpose (Word) Register R12	UUUUh

Table 44 General purpose registers

Name	Physical Address	8-Bit Address	Description	Reset Value
R13	(CP) + 26	FDh	CPU General Purpose (Word) Register R13	UUUUh
R14	(CP) + 28	FEh	CPU General Purpose (Word) Register R14	UUUUh
R15	(CP) + 30	FFh	CPU General Purpose (Word) Register R15	UUUUh

Table 44 General purpose registers

The first 8 GPRs (R7...R0) may also be accessed bytewise. Other than with SFRs, writing to a GPR byte does not affect the other byte of the respective GPR.
The respective halves of the byte-accessible registers receive special names:

Name	Physical Address	8-Bit Address	Description	Reset Value
RL0	(CP) + 0	F0h	CPU General Purpose (Byte) Register RL0	UUh
RH0	(CP) + 1	F1h	CPU General Purpose (Byte) Register Rh0	UUh
RL1	(CP) + 2	F2h	CPU General Purpose (Byte) Register RL1	UUh
RH1	(CP) + 3	F3h	CPU General Purpose (Byte) Register RH1	UUh
RL2	(CP) + 4	F4h	CPU General Purpose (Byte) Register RL2	UUh
RH2	(CP) + 5	F5h	CPU General Purpose (Byte) Register RH2	UUh
RL3	(CP) + 6	F6h	CPU General Purpose (Byte) Register RL3	UUh
RH3	(CP) + 7	F7h	CPU General Purpose (Byte) Register RH3	UUh
RL4	(CP) + 8	F8h	CPU General Purpose (Byte) Register RL4	UUh
RH4	(CP) + 9	F9h	CPU General Purpose (Byte) Register RH4	UUh
RL5	(CP) + 10	FAh	CPU General Purpose (Byte) Register RL5	UUh
RH5	(CP) + 11	FBh	CPU General Purpose (Byte) Register RH5	UUh

Table 45 General purpose registers ordered by bytewise name

Name	Physical Address	8-Bit Address	Description	Reset Value
RL6	(CP) + 12	FCh	CPU General Purpose (Byte) Register RL6	UUh
RH6	(CP) + 13	FDh	CPU General Purpose (Byte) Register RH6	UUh
RL7	(CP) + 14	FEh	CPU General Purpose (Byte) Register RL7	UUh
RH7	(CP) + 14	FFh	CPU General Purpose (Byte) Register RH7	UUh

Table 45 General purpose registers ordered by bytewise name

16.3 Special function registers ordered by name

The following table lists all ST10R272L SFRs in alphabetical order.
Bit-addressable SFRs are marked with the letter “b” in column “Name”. SFRs within the Extended SFR-Space (ESFRs) are marked with the letter “E” in column “Physical Address”.

An SFR can be specified by its individual mnemonic name. Depending on the selected addressing mode, an SFR can be accessed by its physical address (using the Data Page Pointers), or by its short 8-bit address (without using the Data Page Pointers).

Name	Physical Address	8-Bit Address	Description	Reset Value
ADDRSEL1	FE18h	0Ch	Address Select Register 1	0000h
ADDRSEL2	FE1Ah	0Dh	Address Select Register 2	0000h
ADDRSEL3	FE1Ch	0Eh	Address Select Register 3	0000h
ADDRSEL4	FE1Eh	0Fh	Address Select Register 4	0000h
BUSCON0 b	FF0Ch	86h	Bus Configuration Register 0	0XX0h
BUSCON1 b	FF14h	8Ah	Bus Configuration Register 1	0000h
BUSCON2 b	FF16h	8Bh	Bus Configuration Register 2	0000h
BUSCON3 b	FF18h	8Ch	Bus Configuration Register 3	0000h
BUSCON4 b	FF1Ah	8Dh	Bus Configuration Register 4	0000h
CAPREL	FE4Ah	25h	GPT2 Capture/Reload Register	0000h

Table 46 Special functional registers ordered by name

Name		Physical Address		8-Bit Address	Description	Reset Value
CC8IC	b	FF88h		C4h	EX0IN Interrupt Control Register	0000h
CC9IC	b	FF8Ah		C5h	EX1IN Interrupt Control Register	0000h
CC10IC	b	FF8Ch		C6h	EX2IN Interrupt Control Register	0000h
CC11IC	b	FF8Eh		C7h	EX3IN Interrupt Control Register	0000h
CP		FE10h		08h	CPU Context Pointer Register	FC00h
CRIC	b	FF6Ah		B5h	GPT2 CAPREL Interrupt Control Register	0000h
CSP		FE08h		04h	CPU Code Segment Pointer Register (read only)	0000h
DP0L	b	F100h	E	80h	P0L Direction Control Register	00h
DP0H	b	F102h	E	81h	P0h Direction Control Register	00h
DP1L	b	F104h	E	82h	P1L Direction Control Register	00h
DP1H	b	F106h	E	83h	P1h Direction Control Register	00h
DP2	b	FFC2h		E1h	Port 2 Direction Control Register	-0--h
DP3	b	FFC6h		E3h	Port 3 Direction Control Register	0000h
DP4	b	FFCAh		E5h	Port 4 Direction Control Register	00h
DP6	b	FFCEh		E7h	Port 6 Direction Control Register	00h
DP7	b	FFD2h		E9h	Port 7 Direction Control Register	-0h
DPP0		FE00h		00h	CPU Data Page Pointer 0 Register (10 bits)	0000h
DPP1		FE02h		01h	CPU Data Page Pointer 1 Register (10 bits)	0001h
DPP2		FE04h		02h	CPU Data Page Pointer 2 Register (10 bits)	0002h
DPP3		FE06h		03h	CPU Data Page Pointer 3 Register (10 bits)	0003h
EBUSCON	b	F10Eh	E	87H	Extended BUSCON register	0000h
EXICON	b	F1C0h	E	E0h	External Interrupt Control Register	0000h

Table 46 Special functional registers ordered by name

Name	Physical Address	8-Bit Address	Description	Reset Value
IDCHIP	F07Ch E	3Eh	Device Identifier Register	Refer to Data Sheet or Errata Sheet for values
IDMANUF	F07Eh E	3Fh	Manufacturer/Process Identifier Register	
IDMEM	F07Ah E	3Dh	On-chip Memory Identifier Register	
IDPROG	F078h E	3Ch	Programming Voltage Identifier Register	
IDX0 b	FF08h	84h	MAC Unit Address Pointer 0	0000h
IDX1 b	FF0Ah	85h	MAC Unit Address Pointer 1	0000h
MAH	FE5Eh	2Fh	MAC Unit Accumulator - High Word	0000h
MAL	FE5Ch	2Eh	MAC Unit Accumulator - Low Word	0000h
MCW	FFDCh	EEh	MAC Unit Control Word	0000h
MDC b	FF0Eh	87h	CPU Multiply Divide Control Register	0000h
MDH	FE0Ch	06h	CPU Multiply Divide Register – High Word	0000h
MDL	FE0Eh	07h	CPU Multiply Divide Register – Low Word	0000h
MRW b	FFDAh	EDh	MAC Unit Repeat Word	0000h
MSW b	FFDEh	EFh	MAC Unit Status Word	0200h
ODP2 b	F1C2h E	E1h	Port 2 Open Drain Control Register	-0--h
ODP3 b	F1C6h E	E3h	Port 3 Open Drain Control Register	0000h
ODP6 b	F1CEh E	E7h	Port 6 Open Drain Control Register	00h
ODP7 b	F1D2h E	E9h	Port 7 Open Drain Control Register	-0h
ONES	FF1Eh	8Fh	Constant Value 1's Register (read only)	FFFFh
P0L b	FF00h	80h	Port 0 Low Register (Lower half of PORT0)	00h
P0H b	FF02h	81h	Port 0 High Register (Upper half of PORT0)	00h
P1L b	FF04h	82h	Port 1 Low Register (Lower half of PORT1)	00h
P1H b	FF06h	83h	Port 1 High Register (Upper half of PORT1)	00h

Table 46 Special functional registers ordered by name

Name	Physical Address	8-Bit Address	Description	Reset Value
P2 b	FFC0h	E0h	Port 2 Register (4 bits)	-0--h
P3 b	FFC4h	E2h	Port 3 Register	0000h
P4 b	FFC8h	E4h	Port 4 Register (8 bits)	00h
P5 b	FFA2h	D1h	Port 5 Register (read only)	XXXXh
P6 b	FFCCh	E6h	Port 6 Register (8 bits)	00h
P7 b	FFD0h	E8h	Port 7Register (4 bits)	-0h
PECC0	FEC0h	60h	PEC Channel 0 Control Register	0000h
PECC1	FEC2h	61h	PEC Channel 1 Control Register	0000h
PECC2	FEC4h	62h	PEC Channel 2 Control Register	0000h
PECC3	FEC6h	63h	PEC Channel 3 Control Register	0000h
PECC4	FEC8h	64h	PEC Channel 4 Control Register	0000h
PECC5	FECAh	65h	PEC Channel 5 Control Register	0000h
PECC6	FECCh	66h	PEC Channel 6 Control Register	0000h
PECC7	FECEh	67h	PEC Channel 7 Control Register	0000h
PP3	F03Eh E	1Fh	PWM Module Period Register 3	0000h
PSW b	FF10h	88h	CPU Program Status Word	0000h
PW3	FE36h	1Bh	PWM Module Pulse Width Register 3	0000h
PWMCON0 b	FF30h	98h	PWM Module Control Register 0	0000h
PWMCON1 b	FF32h	99h	PWM Module Control Register 1	0000h
PWMIC b	F17Eh E	BFh	PWM Module Interrupt Control Register	0000h
QR0	F004h E	02h	MAC Unit Offset Register R0 (8 bits)	00h
QR1	F006h E	03h	MAC Unit Offset Register R1 (8 bits)	00h
QX0	F000h E	00h	MAC Unit Offset Register X0 (8 bits)	00h
QX1	F002h E	01h	MAC Unit Offset Register X1 (8 bits)	00h

Table 46 Special functional registers ordered by name

Name	Physical Address	8-Bit Address	Description	Reset Value
RP0H b	F108h E	84h	System Start-up Configuration Register (Rd. only)	XXh
S0BG	FEB4h	5Ah	Serial Channel 0 baud rate generator reload reg	0000h
S0CON b	FFB0h	D8h	Serial Channel 0 Control Register	0000h
S0EIC b	FF70h	B8h	Serial Channel 0 Error Interrupt Control Register	0000h
S0RBUF	FEB2h	59h	Serial Channel 0 receive buffer reg. (rd only)	XXh
S0RIC b	FF6Eh	B7h	Serial Channel 0 Receive Interrupt Control Reg.	0000h
S0TBIC b	F19Ch E	CEh	Serial Channel 0 transmit buffer interrupt control reg	0000h
S0TBUF	FEB0h	58h	Serial Channel 0 transmit buffer register (wr only)	00h
S0TIC b	FF6Ch	B6h	Serial Channel 0 Transmit Interrupt Control Register	0000h
SP	FE12h	09h	CPU System Stack Pointer Register	FC00h
SSPCON0	EF00h X	---	SSP Control Register 0	0000h
SSPCON1	EF02h X	---	SSP Control Register 1	0000h
SSPRTB	EF04h X	---	SSP Receive/Transmit Buffer	XXXXh
SSPTBH	EF06h X	---	SSP Transmit Buffer High	XXXXh
STKOV	FE14h	0Ah	CPU Stack Overflow Pointer Register	FA00h
STKUN	FE16h	0Bh	CPU Stack Underflow Pointer Register	FC00h
SYSCON b	FF12h	89h	CPU System Configuration Register	0xx0h[1]
T2	FE40h	20h	GPT1 Timer 2 Register	0000h
T2CON b	FF40h	A0h	GPT1 Timer 2 Control Register	0000h
T2IC b	FF60h	B0h	GPT1 Timer 2 Interrupt Control Register	0000h
T3	FE42h	21h	GPT1 Timer 3 Register	0000h
T3CON b	FF42h	A1h	GPT1 Timer 3 Control Register	0000h

Table 46 Special functional registers ordered by name

Name		Physical Address		8-Bit Address	Description	Reset Value
T3IC	b	FF62h		B1h	GPT1 Timer 3 Interrupt Control Register	0000h
T4		FE44h		22h	GPT1 Timer 4 Register	0000h
T4CON	b	FF44h		A2h	GPT1 Timer 4 Control Register	0000h
T4IC	b	FF64h		B2h	GPT1 Timer 4 Interrupt Control Register	0000h
T5		FE46h		23h	GPT2 Timer 5 Register	0000h
T5CON	b	FF46h		A3h	GPT2 Timer 5 Control Register	0000h
T5IC	b	FF66h		B3h	GPT2 Timer 5 Interrupt Control Register	0000h
T6		FE48h		24h	GPT2 Timer 6 Register	0000h
T6CON	b	FF48h		A4h	GPT2 Timer 6 Control Register	0000h
T6IC	b	FF68h		B4h	GPT2 Timer 6 Interrupt Control Register	0000h
TFR	b	FFACh		D6h	Trap Flag Register	0000h
WDT		FEAEh		57h	Watchdog Timer Register (read only)	0000h
WDTCON		FFAEh		D7h	Watchdog Timer Control Register	000xh[2]
XP1IC	b	F18Eh	E	C7h	SSP Interrupt Control Register	0000h
XP3IC	b	F19Eh	E	CFh	PLL unlock Interrupt Control Register	0000h
ZEROS	b	FF1Ch		8Eh	Constant Value 0's Register (read only)	0000h

Table 46 Special functional registers ordered by name

1. The system configuration is selected during reset.
2. Bit WDTR indicates a watchdog timer triggered reset.

16.4 Special function registers ordered by address

The following table lists all SFRs which are implemented in the ST10R272L ordered by their physical address. Bit-addressable SFRs are marked with the letter “b” in column “Name”.

SFRs within the Extended SFR-Space (ESFRs) are marked with the letter “E” in column “Physical Address”..

Physical Address		Name		8-Bit Address	Description	Reset Value
EF00h	X	SSPCON0		---	SSP Control Register 0	0000h
EF02h	X	SSPCON1		---	SSP Control Register 1	0000h
EF04h	X	SSPRTB		---	SSP Receive/Transmit Buffer	XXXXh
EF06h	X	SSPTBH		---	SSP Transmit Buffer High	XXXXh
F000h	E	QX0		00h	MAC Unit Offset Register X0 (8 bits)	00h
F002h	E	QX1		01h	MAC Unit Offset Register X1 (8 bits)	00h
F004h	E	QR0		02h	MAC Unit Offset Register R0 (8 bits)	00h
F006h	E	QR1		03h	MAC Unit Offset Register R1 (8 bits)	00h
F03Eh	E	PP3		1Fh	PWM Module Period Register 3	0000h
F078h	E	IDPROG		3Ch	Programming Voltage Identifier Register	Refer to Data Sheet or Errata Sheet for values
F07Ah	E	IDMEM		3Dh	On-chip Memory Identifier Register	
F07Ch	E	IDCHIP		3Eh	Device Identifier Register	
F07Eh	E	IDMANUF		3Fh	Manufacturer/Process Identifier Register	
F100h	E	DP0L	b	80h	P0L Direction Control Register	00h
F102h	E	DP0H	b	81h	P0h Direction Control Register	00h
F104h	E	DP1L	b	82h	P1L Direction Control Register	00h
F106h	E	DP1H	b	83h	P1h Direction Control Register	00h
F108h	E	RP0H	b	84h	System Start-up Configuration Register (Rd. only)	XXh
F10Eh	E	EBUSCON	b	87H	Extended BUSCON register	0000h
F17Eh	E	PWMIC	b	BFh	PWM Module Interrupt Control Register	0000h
F18Eh	E	XP1IC	b	C7h	SSP Interrupt Control Register	0000h

Table 47 Special functional registers ordered by address

Physical Address	Name	8-Bit Address	Description	Reset Value
F19Ch E	S0TBIC b	CEh	Serial Channel 0 transmit buffer interrupt control reg	0000h
F19Eh E	XP3IC b	CFh	PLL unlock Interrupt Control Register	0000h
F1C0h E	EXICON b	E0h	External Interrupt Control Register	0000h
F1C2h E	ODP2 b	E1h	Port 2 Open Drain Control Register	-0--h
F1C6h E	ODP3 b	E3h	Port 3 Open Drain Control Register	0000h
F1CEh E	ODP6 b	E7h	Port 6 Open Drain Control Register	00h
F1D2h E	ODP7 b	E9h	Port 7 Open Drain Control Register	-0h
FE00h	DPP0	00h	CPU Data Page Pointer 0 Register (10 bits)	0000h
FE02h	DPP1	01h	CPU Data Page Pointer 1 Register (10 bits)	0001h
FE04h	DPP2	02h	CPU Data Page Pointer 2 Register (10 bits)	0002h
FE06h	DPP3	03h	CPU Data Page Pointer 3 Register (10 bits)	0003h
FE08h	CSP	04h	CPU Code Segment Pointer Register (read only)	0000h
FE0Ch	MDH	06h	CPU Multiply Divide Register – High Word	0000h
FE0Eh	MDL	07h	CPU Multiply Divide Register – Low Word	0000h
FE10h	CP	08h	CPU Context Pointer Register	FC00h
FE12h	SP	09h	CPU System Stack Pointer Register	FC00h
FE14h	STKOV	0Ah	CPU Stack Overflow Pointer Register	FA00h
FE16h	STKUN	0Bh	CPU Stack Underflow Pointer Register	FC00h
FE18h	ADDRSEL1	0Ch	Address Select Register 1	0000h
FE1Ah	ADDRSEL2	0Dh	Address Select Register 2	0000h
FE1Ch	ADDRSEL3	0Eh	Address Select Register 3	0000h
FE1Eh	ADDRSEL4	0Fh	Address Select Register 4	0000h
FE36h	PW3	1Bh	PWM Module Pulse Width Register 3	0000h

Table 47 Special functional registers ordered by address

Physical Address	Name	8-Bit Address	Description	Reset Value
FE40h	T2	20h	GPT1 Timer 2 Register	0000h
FE42h	T3	21h	GPT1 Timer 3 Register	0000h
FE44h	T4	22h	GPT1 Timer 4 Register	0000h
FE46h	T5	23h	GPT2 Timer 5 Register	0000h
FE48h	T6	24h	GPT2 Timer 6 Register	0000h
FE4Ah	CAPREL	25h	GPT2 Capture/Reload Register	0000h
FE5Ch	MAL	2Eh	MAC Unit Accumulator - Low Word	0000h
FE5Eh	MAH	2Fh	MAC Unit Accumulator - High Word	0000h
FEAEh	WDT	57h	Watchdog Timer Register (read only)	0000h
FEB0h	S0TBUF	58h	Serial Channel 0 transmit buffer register (wr only)	00h
FEB2h	S0RBUF	59h	Serial Channel 0 receive buffer reg. (rd only)	XXh
FEB4h	S0BG	5Ah	Serial Channel 0 baud rate generator reload reg	0000h
FEC0h	PECC0	60h	PEC Channel 0 Control Register	0000h
FEC2h	PECC1	61h	PEC Channel 1 Control Register	0000h
FEC4h	PECC2	62h	PEC Channel 2 Control Register	0000h
FEC6h	PECC3	63h	PEC Channel 3 Control Register	0000h
FEC8h	PECC4	64h	PEC Channel 4 Control Register	0000h
FECAh	PECC5	65h	PEC Channel 5 Control Register	0000h
FECCh	PECC6	66h	PEC Channel 6 Control Register	0000h
FECEh	PECC7	67h	PEC Channel 7 Control Register	0000h
FF00h	P0L b	80h	Port 0 Low Register (Lower half of PORT0)	00h
FF02h	P0H b	81h	Port 0 High Register (Upper half of PORT0)	00h
FF04h	P1L b	82h	Port 1 Low Register (Lower half of PORT1)	00h
FF06h	P1H b	83h	Port 1 High Register (Upper half of PORT1)	00h

Table 47 Special functional registers ordered by address

Physical Address	Name		8-Bit Address	Description	Reset Value
FF08h	IDX0	b	84h	MAC Unit Address Pointer 0	0000h
FF0Ah	IDX1	b	85h	MAC Unit Address Pointer 1	0000h
FF0Ch	BUSCON0	b	86h	Bus Configuration Register 0	0XX0h
FF0Eh	MDC	b	87h	CPU Multiply Divide Control Register	0000h
FF10h	PSW	b	88h	CPU Program Status Word	0000h
FF12h	SYSCON	b	89h	CPU System Configuration Register	0xx0h[1]
FF14h	BUSCON1	b	8Ah	Bus Configuration Register 1	0000h
FF16h	BUSCON2	b	8Bh	Bus Configuration Register 2	0000h
FF18h	BUSCON3	b	8Ch	Bus Configuration Register 3	0000h
FF1Ah	BUSCON4	b	8Dh	Bus Configuration Register 4	0000h
FF1Ch	ZEROS	b	8Eh	Constant Value 0's Register (read only)	0000h
FF1Eh	ONES		8Fh	Constant Value 1's Register (read only)	FFFFh
FF30h	PWMCON0	b	98h	PWM Module Control Register 0	0000h
FF32h	PWMCON1	b	99h	PWM Module Control Register 1	0000h
FF40h	T2CON	b	A0h	GPT1 Timer 2 Control Register	0000h
FF42h	T3CON	b	A1h	GPT1 Timer 3 Control Register	0000h
FF44h	T4CON	b	A2h	GPT1 Timer 4 Control Register	0000h
FF46h	T5CON	b	A3h	GPT2 Timer 5 Control Register	0000h
FF48h	T6CON	b	A4h	GPT2 Timer 6 Control Register	0000h
FF60h	T2IC	b	B0h	GPT1 Timer 2 Interrupt Control Register	0000h
FF62h	T3IC	b	B1h	GPT1 Timer 3 Interrupt Control Register	0000h
FF64h	T4IC	b	B2h	GPT1 Timer 4 Interrupt Control Register	0000h
FF66h	T5IC	b	B3h	GPT2 Timer 5 Interrupt Control Register	0000h
FF68h	T6IC	b	B4h	GPT2 Timer 6 Interrupt Control Register	0000h

Table 47 Special functional registers ordered by address

Physical Address	Name		8-Bit Address	Description	Reset Value
FF6Ah	CRIC	b	B5h	GPT2 CAPREL Interrupt Control Register	0000h
FF6Ch	S0TIC	b	B6h	Serial Channel 0 Transmit Interrupt Control Register	0000h
FF6Eh	S0RIC	b	B7h	Serial Channel 0 Receive Interrupt Control Reg.	0000h
FF70h	S0EIC	b	B8h	Serial Channel 0 Error Interrupt Control Register	0000h
FF88h	CC8IC	b	C4h	EX0IN Interrupt Control Register	0000h
FF8Ah	CC9IC	b	C5h	EX1IN Interrupt Control Register	0000h
FF8Ch	CC10IC	b	C6h	EX2IN Interrupt Control Register	0000h
FF8Eh	CC11IC	b	C7h	EX3IN Interrupt Control Register	0000h
FFA2h	P5	b	D1h	Port 5 Register (read only)	XXXXh
FFACh	TFR	b	D6h	Trap Flag Register	0000h
FFAEh	WDTCON		D7h	Watchdog Timer Control Register	000xh[2]
FFB0h	S0CON	b	D8h	Serial Channel 0 Control Register	0000h
FFC0h	P2	b	E0h	Port 2 Register (4 bits)	-0--h
FFC2h	DP2	b	E1h	Port 2 Direction Control Register	-0--h
FFC4h	P3	b	E2h	Port 3 Register	0000h
FFC6h	DP3	b	E3h	Port 3 Direction Control Register	0000h
FFC8h	P4	b	E4h	Port 4 Register (8 bits)	00h
FFCAh	DP4	b	E5h	Port 4 Direction Control Register	00h
FFCCh	P6	b	E6h	Port 6 Register (8 bits)	00h
FFCEh	DP6	b	E7h	Port 6 Direction Control Register	00h
FFD0h	P7	b	E8h	Port 7Register (4 bits)	-0h
FFD2h	DP7	b	E9h	Port 7 Direction Control Register	-0h
FFDAh	MRW	b	EDh	MAC Unit Repeat Word	0000h

Table 47 Special functional registers ordered by address

Physical Address	Name	8-Bit Address	Description	Reset Value
FFDCh	MCW	EEh	MAC Unit Control Word	0000h
FFDEh	MSW b	EFh	MAC Unit Status Word	0200h

Table 47 Special functional registers ordered by address

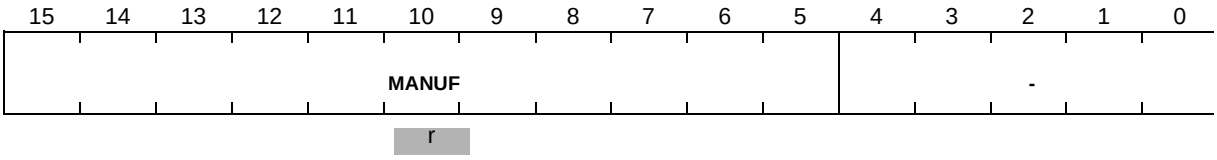
1. The system configuration is selected during reset.
2. Bit WDTR indicates a watchdog timer triggered reset.

16.5 Identification registers

The ST10R272L has four Identification registers, mapped in ESFR space. These register contain:

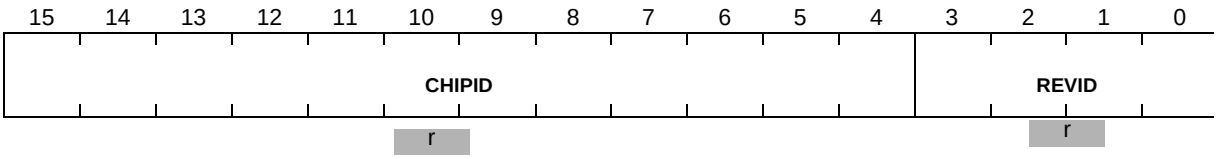
- a manufacturer identifier
- a chip identifier, with its revision
- a internal memory and size identifier
- programming voltage description

IDMANUF (F07Eh / 3Fh) **ESFR**



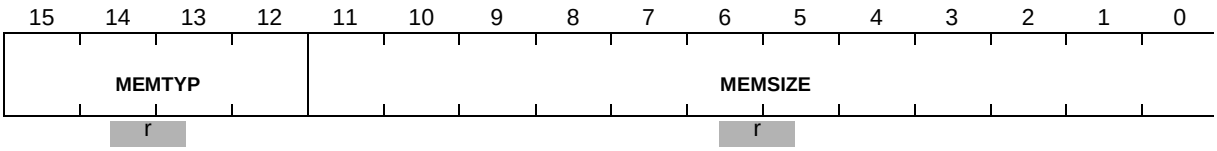
Bit	Function
MANUF	**Manufacturer Identifier** 020h: SGS-Thomson Manufacturer (JTAG worldwide normalization).

IDCHIP (F07Ch / 3Eh) **ESFR**



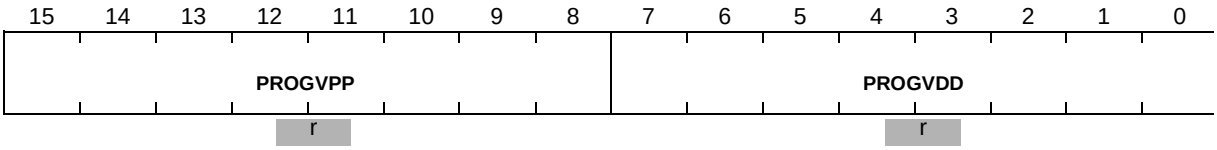
Bit	Function
REVID	**Device Revision Identifier** Refer to datasheet for values.
CHIPID	**Device Identifier** Refer to datasheet for values.

IDMEM (F07Ah / 3Dh) **ESFR**



Bit	Function
MEMSIZE	**Internal Memory Size** Refer to datasheet for values.
MEMTYP	**Internal Memory Type** Refer to datasheet for values.

IDPROG (F078h / 3Ch) **ESFR**



Bit	Function
PROGVDD	**Programming Vdd Voltage** Vdd voltage for FLASH devices is calculated using the formula: Vdd = 20 * <PROGVDD> / 256 [V] Refer to datasheet for values.
PROGVPP	**Programming Vpp Voltage** Vpp voltage for FLASH devices is calculated using the formula: Vpp = 20 * <PROGVDD> / 256 [V] Refer to datasheet for values.

17 POWER REDUCTION MODES

The ST10R272L has two power reduction modes - 'Idle Mode' and' Power Down mode'.

- **Idle Mode:** the CPU is stopped, but the peripherals continue operation. Idle Mode can be terminated by any reset or interrupt request.
- **Power down mode:** both the CPU and the peripherals are stopped. Power Down mode can only be terminated by a hardware reset in protected mode, or by an external interrupt.

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 READY/$\overline{\text{READY}}$ is enabled, but has not been activated (driven high/low) during the last bus access.*

17.1 Idle Mode

The power consumption of the ST10R272L 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 IDLE has completed all stages of the pipeline. This guarantees that all instructions before the IDLE instruction are completed before the CPU is stopped.

The IDLE instruction is a protected 32-bit instruction. This prevents unintentional entry into Idle Mode.

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 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 executes the instruction following the IDLE instruction. If the interrupt request cannot be serviced (if the priority is too low or if the interrupt system

interrupt system is globally disabled) the CPU immediately resumes normal program execution with the instruction following the IDLE instruction.

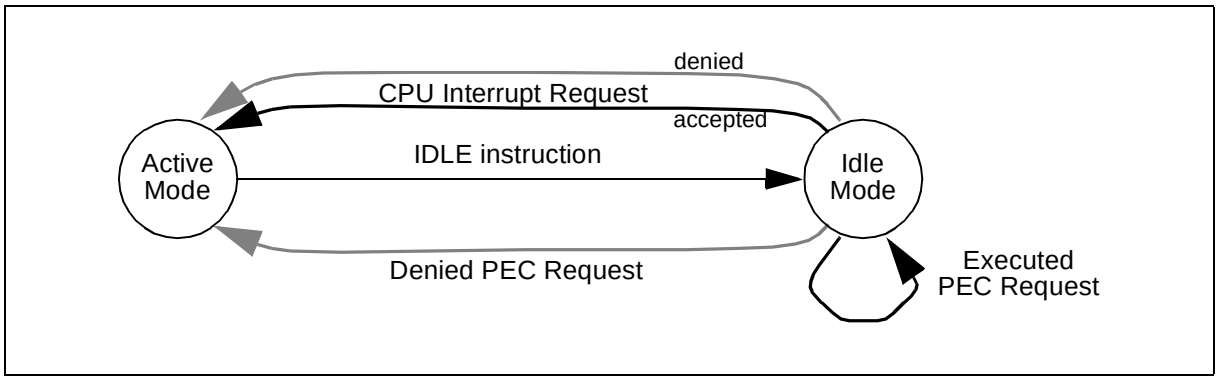


Figure 116 Transitions between Idle Mode and active mode

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

Idle Mode can also be terminated by a Non-Maskable Interrupt, i.e. a high to low transition on the $\overline{\text{NMI}}$ pin. After Idle Mode has been terminated by an interrupt or NMI request, the interrupt 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 is set before Idle Mode is entered, terminates 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. However, he associated interrupt vector will not be accessed.*

The Watchdog Timer can monitor Idle Mode: an internal reset is generated if no interrupt or NMI request occurs before the Watchdog Timer overflows. To prevent Watchdog Timer overflow during Idle Mode, it must be programmed with a reasonable time interval - before Idle Mode is entered.

17.2 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, but the contents of the internal RAM are preserved through the V_{CC} pin supply voltage. The Watchdog Timer is stopped in Power Down mode. The ST10R272L provides two different operating Power Down modes:

- Protected Power Down mode,
- Interruptible Power Down mode.

The Power Down operating mode is selected by the bit PWDCFG in SYSCON register:

SYSCON (FF12h / 89h) **SFR** **Reset Value: 0XX0h**

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	PWD CFG	OWD DIS	-	SSP EN	VISI BLE	XPER-SHARE
rw			rw	rw	rw	rw	rw	rw	rw	rw	rw	-	rw	rw	rw

Bit	Function
PWDCFG	**Power Down Mode Configuration Control** '0': Power Down Mode can only be entered during PWRDN instruction execution if $\overline{NMI}$ pin is low, otherwise the instruction has no effect. To exit Power Down Mode, an external reset must occurs by asserting the $\overline{RSTIN}$ pin. '1': Power Down Mode can only be entered during PWRDN instruction execution if all enabled fast external interrupt EXxIN pins are in their inactive level. Exiting this mode can be done by asserting one enabled EXxIN pin.

17.2.1 Protected power down mode

This mode is selected by setting bit PWDCFG in the SYSCON register to '0'. This mode can only be terminated by an external hardware reset:

In Protected power down mode, there are two levels of protection against unintentional entery.

- The PWRDN (Power Down) instruction is a protected 32-bit instruction.
- This PWRDN instruction is effective **only** if the $\overline{NMI}$ (Non Maskable Interrupt) pin is externally pulled low while PWRDN is executed. The microcontroller enters Power Down mode after the PWRDN instruction has completed.

Protected power down mode can be used in conjunction with an external-power failure signal which pulls the $\overline{NMI}$ pin low when a power failure is imminent. The microcontroller enters the NMI trap routine which saves the internal state into RAM. After the internal state

has been saved, the trap routine can set a flag or write a certain bit pattern into specific RAM locations, and then execute the PWRDN instruction. If the $\overline{NMI}$ 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 2.0V for the 3.3V device (2.5V for the 5V device) while the contents of the internal RAM are still be preserved.

Exiting protected power down mode: In this mode, the only way to exit Power Down mode is with an external hardware reset, i.e. by asserting a low level on the $\overline{RSTIN}$ pin. This reset initializes all SFRs and ports to their default state, but does not change the contents of the internal RAM.

The initialization routine (executed upon reset) checks 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.

17.2.2 Interruptible power down mode

Interruptible power down mode is selected by setting the bit PWDCFG in register SYSCON to ‘1’.

Entering interruptible power down mode: In this mode, there are two levels of protection against unintentionally entering Power Down mode.

- The PWRDN (Power Down) instruction is a protected 32-bit instruction.
- PWRDN is effective **only** if all enabled Fast External Interrupt pins (EXxIN pins, alternate functions of Port 2 pins) are in their inactive level. This inactive level is configured with the EXIxES bit field in the EXICON register, as follow:

EXICON (F1C0h / E0h) **ESFR** **Reset Value: 0000h**

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

Bit	Function	
EXIxES	**External Interrupt x Edge Selection Field (x=3...0)**	
	0 0:	Fast external interrupts disabled: standard mode
		EXxIN pin not taken in account for entering/exiting Power Down mode.
	0 1:	Interrupt on positive edge (rising)
		Enter Power Down mode if EXiIN = ‘0’, exit if EXxIN = ‘1’ (‘high’ active level)
	1 0:	Interrupt on negative edge (falling)
		Enter Power Down mode if EXiIN = ‘1’, exit if EXxIN = ‘0’ (‘low’ active level)
	1 1:	Interrupt on any edge (rising or falling)
		Always enter Power Down mode, exit if EXxIN level changed.

17.2.3 Exiting interruptible power down mode

When ‘interruptible power down mode’ is entered, the CPU and peripheral clocks are frozen, and the oscillator and PLL are stopped. Interruptible power down mode can be exited by asserting $\overline{\text{RSTIN}}$ or one of the enabled EXxIN pin (Fast External Interrupt).

$\overline{\text{RSTIN}}$ must be held low until the oscillator and PLL have stabilized.

EXxIN inputs are normally sampled interrupt inputs. However, the Power Down mode circuitry uses them as level-sensitive inputs. An EXxIN Interrupt Enable bit (bit CCxIE in the respective CCxIC register) need not to be set to bring the device out of Power Down mode. An external RC circuit must be connected on the Vpp/RPD pin, as shown in the following figure:

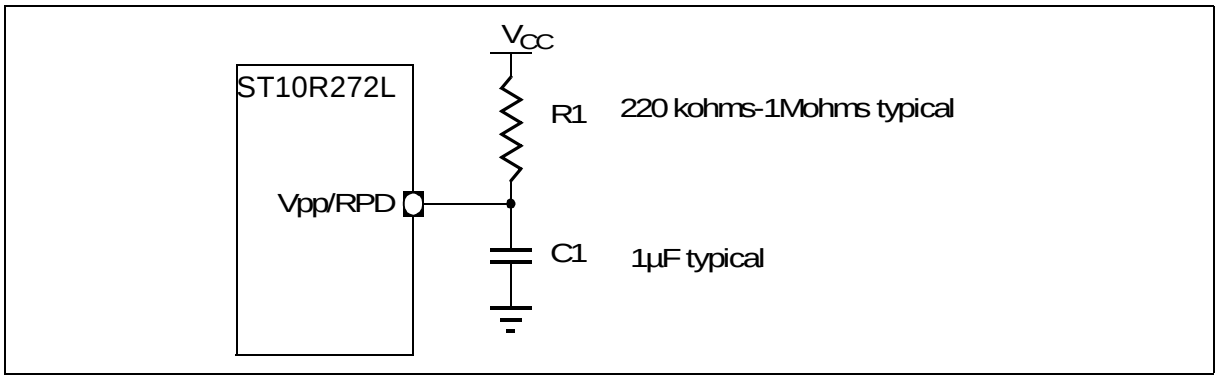


Figure 117 External RC Circuit on Vpp/RPD pin for exiting powerdown mode with external interrupt

To exit Power Down mode with external interrupt, an EXxIN pin must be asserted for at least 40 ns. This signal enables the internal oscillator and PLL circuitry and also turns on an internal weak pull-down on the Vpp/RPD pin to discharge the capacitor C1. The discharging of the external capacitor provides a delay for the oscillator and PLL circuits to stabilize, before the internal CPU and Peripheral clocks are enabled.

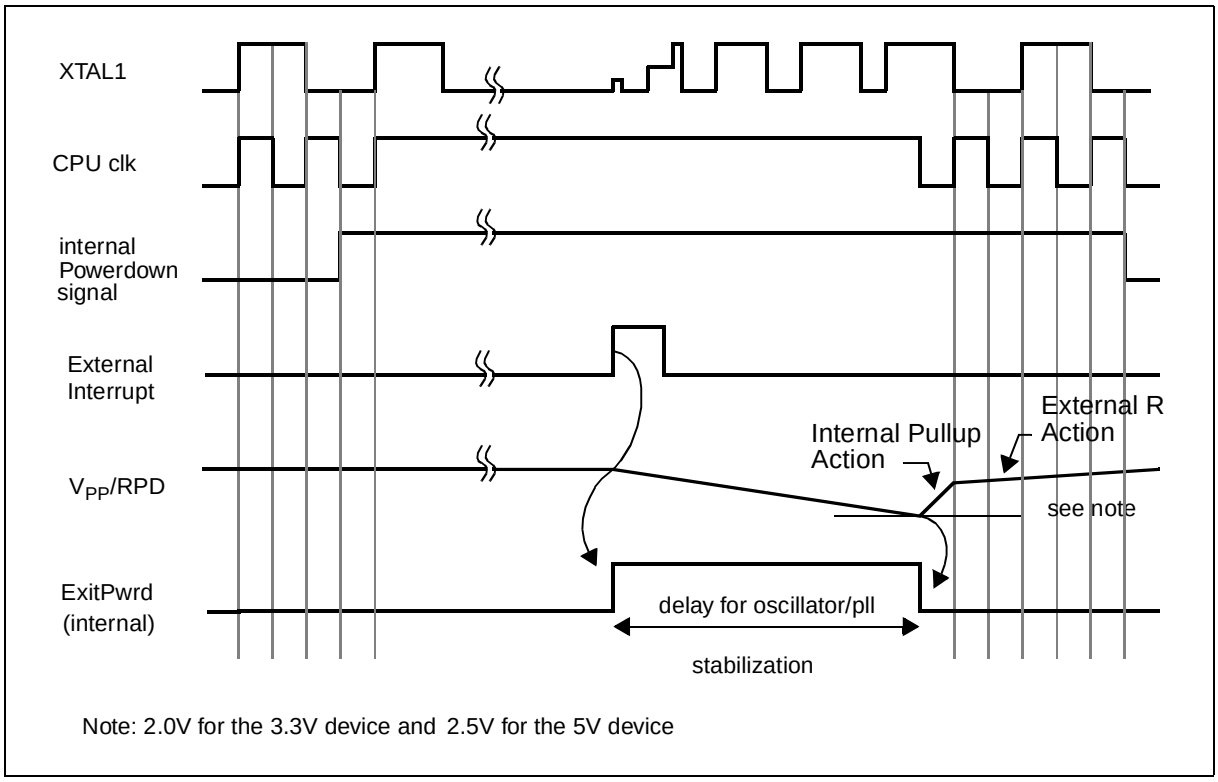


Figure 118 Powerdown exit sequence when using an external interrupt (PLL multiply factor of 2)

When the RPD/Vpp voltage drops below the threshold voltage (2.0V for the 3.3V device and 2.5V for the 5V device), the Schmitt trigger clears Q2 flip-flop (see *Figure 119*). This enables the CPU and Peripheral clocks. The device resumes code execution.

If the Interrupt was enabled (bit CCxIE='1' in the respective CCxIC register) before entering Power Down mode, the device executes the interrupt service routine, and then resumes execution after the PWRDN instruction (see note below). If the interrupt was disabled, the device executes the instruction following PWRDN instruction, and the Interrupt Request Flag (bit CCxIR in the respective CCxIC register) remains set until it is cleared by software.

Note *Due to internal pipeline, the instruction that follows the PWRDN instruction is executed **before** the CPU performs a call of the interrupt service routine.*

.

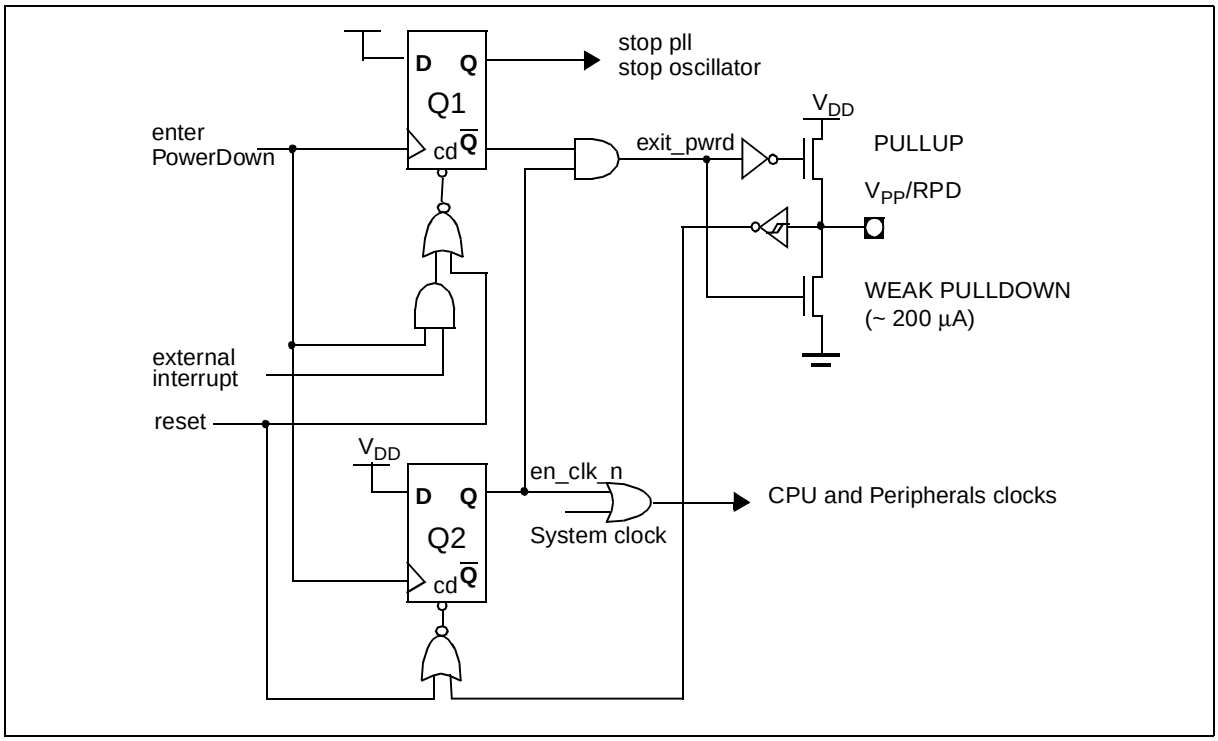


Figure 119 Simplified Powerdown Exit Circuitry

17.2.4 Selecting R and C values

Select an external capacitor C which will produce a sufficient discharge time (in conjunction with internal weak pulldown device on the Vpp/RPD pin) for the internal clock signal to stabilize.

First, determine the worst-case clock signal stabilization time, then use the following formula to calculate the C value: where:

$$C = T_{DIS} \times 1/(5 - V_{TH})$$

C	is the external capacitor on Vpp/RPD, in µF
T_{DIS}	is the worst-case discharge time, in s
I	is the discharge current, in µA
V_{TH}	is the Vpp/RPD threshold voltage, in V

Examples:

- For the 5 volt device, The oscillator needs at least 27.5 ms to stabilize (T_{DIS} = 0.0275 s), V_{TH} is 2.5 V, and I is 200µA. The minimum capacitor value for C is then 2.2 µF $C = 0{,}0275 \times 200 / (5 - 2{,}5) = 2{,}2\mu F$ When using an external oscillator, the value of C can be very small for fast recovery from Power Down mode. For example, a 1 nF capacitor will produce a discharge time of 12.5 µs.
- For the 3.3 volt device, The oscillator needs at least 27.5 ms to stabilize (T_{DIS} = 0.0275 s), V_{TH} is 2.0V, and I is 200µA. The minimum capacitor value for C is then 1.83 µF $C = 0{,}0275 \times 200 / (3{,}3 - 2{,}0) = 4{,}23\mu F$ When using an external oscillator, the value of C can be very small for fast recovery from Power Down mode. For example, a 1 nF capacitor will produce a discharge time of 6.5µs.

The external R resistor value must not interfere with the discharge current. A value between 200 kohms and 1 Mohms should be adequate.

Note The RC circuit is also used for asynchronous Power Up reset.

17.3 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 as normal. Therefore, all ports pins which are configured as general purpose output pins, output the last data value which was written to their port output latches. If the alternate output function of a port pin is used by a peripheral, the state of the pin is determined by the operation of the peripheral.

Port pins which are used for bus control functions, go into that state which represents the inactive state of the respective function (e.g. $\overline{WR}$), or to a defined state which is based on the last bus access (e.g. $\overline{BHE}$). 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 always floats 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, CPU and peripheral clocks are turned off. As for 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 ST10R272L output pins during idle and power down mode.

Output Pin(s)	Idle Mode	Power Down Mode
ALE	Low	Low
RD, WR	High	High
CLKOUT	Active	High
RSTOUT	1	1
P0L	Floating	Floating
P0H	A15...A8 [2] / Float	A15...A8 [2] / Float
PORT1	Last Address [3]/Port Latch Data	Last Address [3]/Port Latch Data
Port 4	Port Latch Data/Last segment	Port Latch Data/Last segment
BHE	Last value	Last value
HLDA	Last value	Last value
BREQ	High	High
CSx	Last value [4]	Last value[4]
Other Port Output Pins	Port Latch Data / Alternate Function	Port Latch Data / Alternate Function

Table 48 Output pin status during idle and power down modes

1. High if EINIT was executed before entering Idle or Power Down mode, Low otherwise.
2. For multiplexed buses with 8-bit data bus.
3. For demultiplexed buses.
4. The **CS** signal that corresponds to the last address remains active (low), all other enabled **CS** signals remain inactive (high).

18 SYSTEM PROGRAMMING

For software development, constructs for modularity, loops, and context switching have been included in the instruction set. Commonly used instruction sequences have been simplified to provide greater flexibility. The following programming features have been designed to optimize the use of the instruction set.

18.1 Instructions provided as subsets of instructions

Instructions used by other micro-controllers can be provided as subsets of more powerful instructions in the ST10R272L. This gives the same functionality, while decreasing the hardware overhead and decode complexity. These instructions can be built into macros.

Directly Substitutable Instructions are instructions known from other micro-controllers that can be replaced by the following instructions of the ST10R272L:

Modification of System Flags is performed using bit set or bit clear instructions (BSET, BCLR). All bit and word instructions can access the PSW register, so instructions such as CLEAR CARRY or ENABLE INTERRUPTS are NOT required.

External Memory Data Access does not require special instructions, to load data pointers or explicitly load and store external data. The ST10R272L provides a Von-Neumann memory architecture and its on-chip hardware automatically detects accesses to internal RAM, GPRs, and SFRs.

Substituted Instruction		ST10R272L Instruction		Function
CLR	Rn	AND	Rn, #0h	Clear register
CPLB	Bit	BMOVN	Bit, Bit	Complement bit
DEC	Rn	SUB	Rn, #1h	Decrement register
INC	Rn	ADD	Rn, #1h	Increment register
SWAPB	Rn	ROR	Rn, #8h	Swap bytes within word

Table 49 Instructions provided as subsets of instructions

18.2 BCD calculations

No direct support for BCD calculations is provided in the ST10R272L. 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 10d. 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.

18.3 Stack operations

The ST10R272L 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.

18.3.1 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.

The system stack allows to store 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).

18.3.2 Circular (virtual) Stack

This basic technique allows data to be pushed until the overflow boundary of the internal stack is reached. At this point a portion of the stacked data must be saved into external memory to create space for further stack pushes. This is called “stack flushing”. When executing a number of return or pop instructions, the upper boundary (since the stack empties upward to higher memory locations) is reached. The entries that have been previously saved in external memory must now be restored. This is called “stack filling”. Because procedure call instructions do not continue to nest infinitely and call and return instructions alternate, flushing and filling normally occurs very infrequently. If this is not true for a given program environment, this technique should not be used because of the overhead of flushing and filling.

The basic mechanism is the transformation of the addresses of a virtual stack area, controlled via registers SP, STKOV and STKUN, to a defined physical stack area within the internal RAM via hardware. This virtual stack area covers all possible locations that SP can point to, i.e. 00’F000h through 00’FFFEh. 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).

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’FBFEh). This transformation is done via hardware (see figure below).

The reset values (STKOV=FA00h, STKUN=FC00h, SP=FC00h, STKSZ=000b) map the virtual stack area directly to the physical stack area and allow to use internal system stack without any changes, provided that the 256 word area is not exceeded.

<STKSZ>	Stack Size (Words)	Internal RAM Addresses (Words) of Physical Stack	Significant Bits of Stack Pointer SP
0 0 0 b	256	00’FBFEh...00’FA00h (Default after Reset)	SP.8...SP.0
0 0 1 b	128	00’FBFEh...00’FB00h	SP.7...SP.0

<STKSZ>	Stack Size (Words)	Internal RAM Addresses (Words) of Physical Stack	Significant Bits of Stack Pointer SP
0 1 0 b	64	00'FBFEh...00'FB80h	SP.6...SP.0
0 1 1 b	32	00'FBFEh...00'FBC0h	SP.5...SP.0
1 0 0 b	---	Reserved. Do not use this combination.	---
1 0 1 b	---	Reserved. Do not use this combination.	---
1 1 0 b	---	Reserved. Do not use this combination.	---
1 1 1 b	1024	00'FDFEh...00'F600h (Note: No circular stack)	SP.11...SP.0

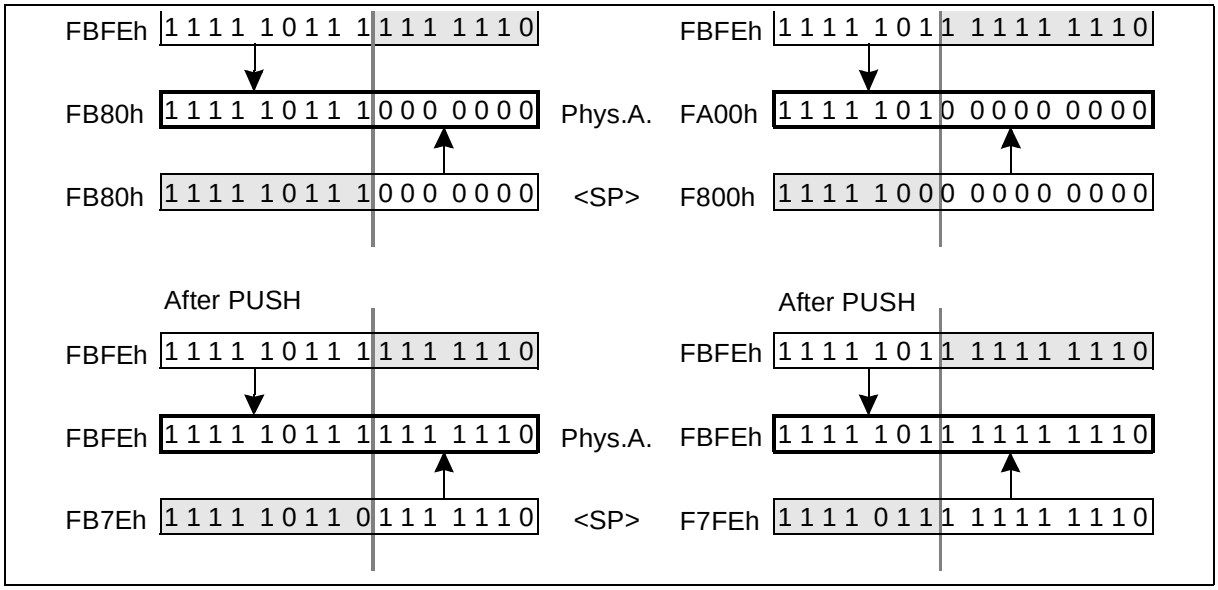


Figure 120 Physical stack address generation

The following example demonstrates the circular stack mechanism

Register R1 is pushed onto the lowest physical stack location according to the selected maximum stack size. With the following instruction, register R2 is 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 address = FA02h
```

```
MOV SP, #0F802h  ;Set SP before last entry of physical stack of 256 words
PUSH R1          ;(SP) =           F800h: Physical stack address = FA00h
PUSH R2          ;(SP) =          F7FEh: Physical stack address = FBFEh
```

The effect of the address transformation, is that the physical stack addresses *wraps around* - from the end of the defined area to its beginning. When flushing and filling the internal stack, the Circular Stack Mechanism requires only the portion of stack data which is really to be re-used, to be moved (i.e. the upper part of the defined stack area). It does not require the whole stack area to be moved. Stack data that remains 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 256 words (STKSZ = '000b' to '0116b'), it does not work with option STKSZ = '111b', which uses the complete internal RAM for system stack.*

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 transferred data 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 of 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. User programs are only affected by the execution time required by the trap routines.

The following procedure initializes the controller to use of the circular stack mechanism:

- Specify the size of the physical system stack area within the internal RAM (bitfield STKSZ in register SYSCON).
- Define two pointers, which specify the upper and lower boundary of the external stack. These values are then tested in the stack underflow and overflow trap routines when moving data.
- Set the stack overflow pointer (STKOV) to the limit of the defined internal stack area plus six words (for the reserved space to store two interrupt entries).

The internal stack fills until the overflow pointer is reached. After entry into the overflow trap procedure, the top of the stack is copied to the external memory. The internal pointers are modified to reflect the newly allocated space. After exiting from the trap procedure, the internal stack wraps around to the top of the internal stack, and continues to grow until the new value of the stack overflow pointer is reached.

When the underflow pointer is reached while the stack is emptied the bottom of stack is reloaded from the external memory and the internal pointers are adjusted accordingly.

18.3.3 Linear stack

The ST10R272L offers a linear stack option (STKSZ = '111b'), where the system stack may use the complete internal RAM area. This provides a large system stack, without requiring extra 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 is available for the system stack is defined by the STKUN and STKOV pointers. The underflow and overflow traps, in this case, serve as 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'F000h up to 00'FFFEh the (physical) system stack must be located within the internal RAM, and therefore, may only use the address range 00'F600h to 00'FDFEh. It is the user's responsibility to restrict the system stack to the internal RAM range.

Note *Avoid stack accesses within address range 00'F000h to 00'F5FEh (ESFR space and reserved area) and within address range 00'FE00h and 00'FFFEh (SFR space). Otherwise unpredictable results will occur.*

18.3.4 User stacks

User stacks can be used 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.

18.4 Register banking

Register banking provides an extremely fast method to switch user context. A single machine-cycle instruction, saves the old register 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. Therefore, on entry into a new task, the appropriate bank pointer is used as the operand for the SCXT (switch context) instruction. On exit from a task, a simple POP instruction to the context pointer (CP) restores the previous task's register bank.

18.5 CALL procedure entry-and-exit

To support modular programming, a procedure mechanism is provided for 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.

Note *Procedures may be called conditionally with instructions CALLA or CALLI, or be called unconditionally using instructions CALLR or CALLS.*

Any data pushed onto the system stack during execution of the subroutine, must be popped before the RET instruction is executed.

18.5.1 Passing parameters on the system stack

Parameters may be passed through the system stack by PUSH instructions (before the subroutine is called) and POP instructions (during execution of the subroutine). Base-plus-offset indirect addressing, permits access to parameters without popping the parameters from the stack during subroutine execution. Indirect addressing provides a mechanism to access data referenced by data pointers.

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.

18.5.2 Cross segment subroutine calls

Calls to subroutines in different segments require the CALLS instruction (call inter-segment subroutine). This instruction preserves both the CSP (code segment pointer) and IP on the system stack.

On 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.*

18.5.3 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 allows 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 grows downwards and the register bank grows upwards.*

After entering the subroutine:

```
SUB         SP, #10     ;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, #10     ;Release the 5 word of the current system stack
```

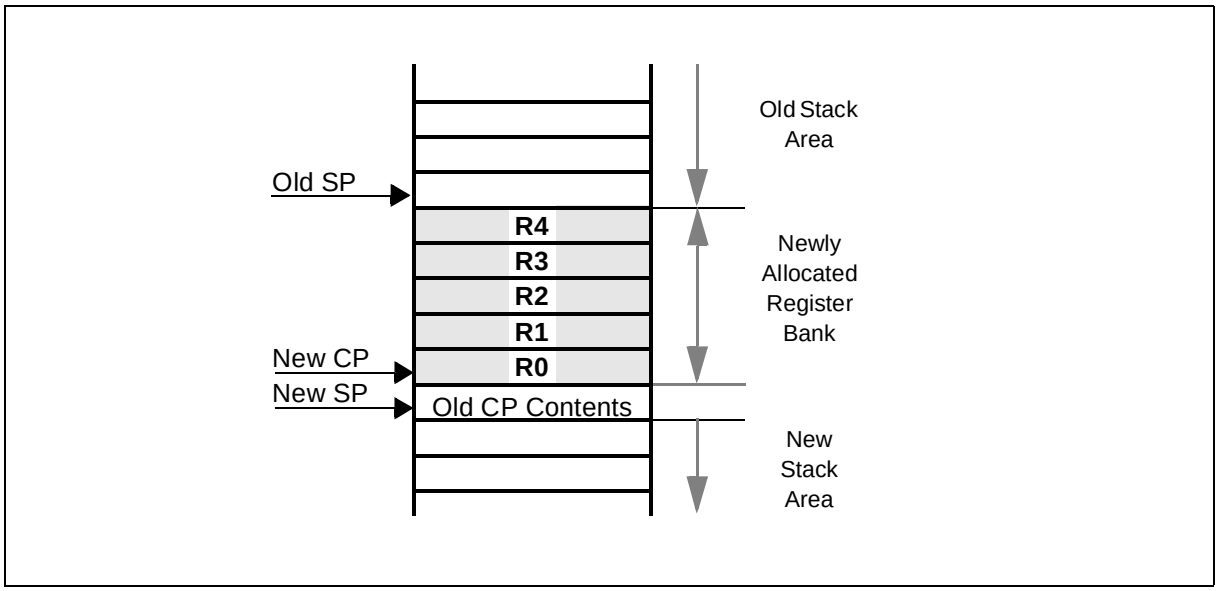


Figure 121 Local registers

18.6 Table searching

The following features decrease the execution time for table searches.

- Branch delays are eliminated by the branch target cache after the first iteration of the loop.
- In non-sequentially searched tables, the enhanced performance of the ALU allows more-complicated hash algorithms to be processed for 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, LOO                 ;Test whether target has not been found
```

Note The last entry in the table must be greater than the largest possible target.

```
MOV     R0, #BASE                   ;Move table base into R0
LOOP:   CMP           R1, [R0+]     ;Compare target to table entry
```

```
MOV      R0, #BASE                ;Move table base into R0
JMPR     cc_NET, LOO              ; 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 (8000h).

18.7 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 ST10R272L, all peripherals (except the watchdog timer) are disabled and initialized to default values. The required configuration of a specific peripheral is programmed using MOV instructions of either constants or memory values to specific SFRs. Specific control flags may be altered by bit instructions.

Once in operation, the peripheral operates autonomously until an end condition is reached, then it requests a PEC transfer or a CPU service through an interrupt routine. Information may 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 is bit addressable to allow user semaphores. Instructions provide 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.

18.8 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 can be used for masking and extracting of portions of floating point numbers.

The following features decrease the floating point operation time.

The PRIOR instruction helps to normalize 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.

An overflow (V) flag in the PSW helps to round the result of normalized floating point. This flag is set when a ‘1’ 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 required rounding algorithm.

18.9 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 machine state is preserved on the system stack, and a branch to the appropriate trap/interrupt vector is made.

All trap and interrupt routines use the RETI (return from interrupt) instruction to exit from the routine. This instruction restores the system state from the system stack and then branches back to the location where the trap or interrupt occurred.

18.10 Inseparable instruction sequences

Some instructions, such as semaphore handling, require uninterruptable code sequences. This is achieved by inhibiting interrupts during the respective code sequence - by disabling and enabling them before and after the sequence. The ATOMIC instruction locks 1...4 instructions to an inseparable code sequence during which the interrupt system **and Class A Traps** 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 becomes active immediately, i.e. no other instruction will enter the pipeline except the one that follows the ATOMIC instruction, and no interrupt request will be serviced in between. In this way, all instructions requiring multiple cycles or hold states are regarded as one instruction (e.g. MUL is one instruction). Any instruction type can be used in an inseparable code sequence. For example:

ATOMIC #3

The following 3 instructions are locked (No NOP required)

```
MOV     R0, #1234h      ;Instruction 1 (no other instr. enters the
                        pipeline!)
MOV     R1, #5678h      ;Instruction 2
MUL     R0, R1          ;Instruction 3: MUL regarded as one instruction
MOV     R2, MDL         ;This instruction is out of the scope of ATOMIC
                        ;instruction sequence
```

18.11 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 64KByte of data. In applications with big data arrays, especially in HLL applications using large memory models, this may require frequent reloading of the DPPs, even for single accesses.

The EXTP (extend page) instruction allows to switch to an arbitrary data page for 1...4 instructions without having to change the current DPPs.

Example:

```
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 standard DPP scheme!
```

The EXTS (extend segment) instruction switches to a 64KByte 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”.

Example:

```
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 standard DPP scheme!
```

Note *Instructions EXTP and EXTS inhibit interrupts the same way as ATOMIC.*

18.12 Short addressing in the extended SFR (ESFR) space

The short addressing modes of the ST10R272L (REG or BITOFF) implicitly access the SFR space. Additional ESFR space would have to be accessed by 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. Additional registers can also be accessed this way.

The EXTPR and EXTSR instructions combine the DPP override mechanism with re-direction to the ESFR space, using a single instruction.

Instructions EXTR, EXTPR and EXTSR inhibit interrupts the same way as ATOMIC.
The switch to the ESFR area, and data-page overriding is checked by the development tools or handled automatically.

Each of the described Extension instructions 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 re-started with the value of the new instruction. This can be used to construct 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.

19 INDEX OF REGISTERS

Notes: