ST10F280

USER'S MANUAL

Release 2.0

TABLE OF CONTENTS PAGE

ST10F280

USER'S MANUAL

1 - INTRODUCTION

This manual describes the functionality of the ST10F280 device.

An architectural overview describes the CPU performance, the on-chip system resources, the on-chip clock generator, the on-chip peripheral blocks and the protected bits.

The operation of the CPU and the on-chip peripherals, and the different operating modes - such as system reset, power reduction modes, interrupt handling, and system programming - are described in individual chapters.

The explanation of memory configuration has been restricted to that of the internal addressable memory space. The ST10F280 flash configurations are not discussed in this manual. Refer to the ST10F280 datasheet for detailed information.

The Special Functional Registers are listed both by name and hexadecimal address. The instruction set is covered in full in the ST10 Family Programming Manual and is, therefore, not discussed in this manual. However, software programming feature - including constructs for modularity, loops, and context switching - are described in Chapter 22 - System Programming.

The DC and AC electrical specifications of the device and the pin description for each available package, are not covered in this manual but are listed in the specific device Data Sheets.

Before starting on a new design, verify the device characteristics and pinout with an up-to-date copy of the device Data Sheet.

The ST10F280 software and hardware development tools include:

– Compilers (C, C++), Macro-Assemblers, Linkers, Locators, Library Managers, Format-Converters from Tasking & Keil
– HLL debuggers
– Real-Time operating systems
– In-Circuit Emulators (based on bondout ST chips) from Hitex, Lauterbach, Nohau
– Logic Analyzer disassemblers
– Evaluation Boards with monitor programs from FORTH
– Industrial embedded flash programming software from PLS
– Network driver software (CAN)

1.1 - Abbreviations

The following abbreviations and acronyms are used in this User's Manual:

ADC	Analog Digital Converter
ALE	Address Latch Enable
ALU	Arithmetic and Logic Unit
ASC	Asynchronous/synchronous Serial Controller
BRG	Baud Rate Generator
CAN	Controller Area Network (License Bosch)
CAPCOM	CAPture and COMpare unit
CISC	Complex Instruction Set Computing
CMOS	Complementary Metal Oxide Silicon
CPU	Central Processing Unit
EBC	External Bus Controller
ESFR	Extended Special Function Register
Flash	Non-volatile memory that may be electrically erased
GPR	General Purpose Register
GPT	General Purpose Timer unit
HLL	High Level Language
IRAM	On-chip Internal RAM
I/O	Input / Output
PEC	Peripheral Event Controller
PLA	Programmable Logic Array
PLL	Phase Locked Loop
PWM	Pulse Width Modulation
RAM	Random Access Memory
RISC	Reduced Instruction Set Computing
ROM	Read Only Memory
SFR	Special Function Register
SSC	Synchronous Serial Controller
XBUS	Internal representation of the External Bus
XRAM	On-chip extension RAM

2 - ARCHITECTURAL OVERVIEW

ST10F280 architecture combines the advantages of both RISC and CISC processors with an advanced peripheral subsystem. The following block diagram gives an overview of the different on-chip components and of the advanced, high bandwidth internal bus structure of the ST10F280. (see Figure 1).

2.1 - Basic CPU Concepts and Optimizations

The main core of the CPU includes a 4-stage instruction pipeline, a 16-bit arithmetic and logic unit (ALU) and dedicated SFRs.

Additional hardware is provided for a separate multiply and divide unit, a bit-mask generator and a barrel shifter (See Figure 2).

Several areas of the processor core have been optimized for performance and flexibility. Functional blocks in the CPU core are controlled by signals from the instruction decode logic. The core improvements are summarized below, and described in detail in the following sections:

1 High instruction bandwidth / fast execution.
2 High function 8-bit and 16-bit arithmetic and logic unit.
3 Extended bit processing and peripheral control.
4 High performance branch, call, and loop processing.
5 Consistent and optimized instruction formats.
6 Programmable multiple priority interrupt structure.

Figure 1 : ST10F280 functional block diagram

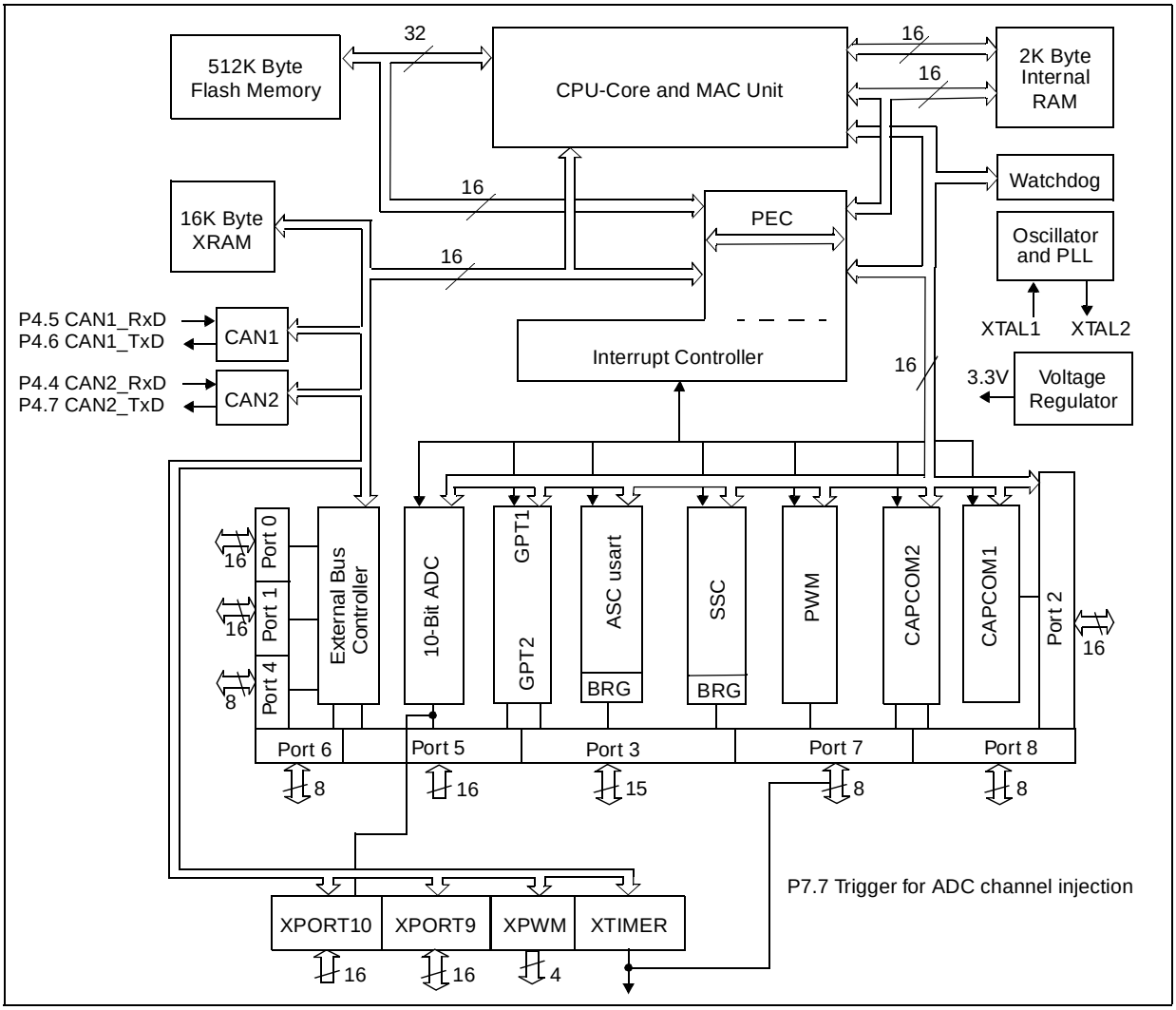
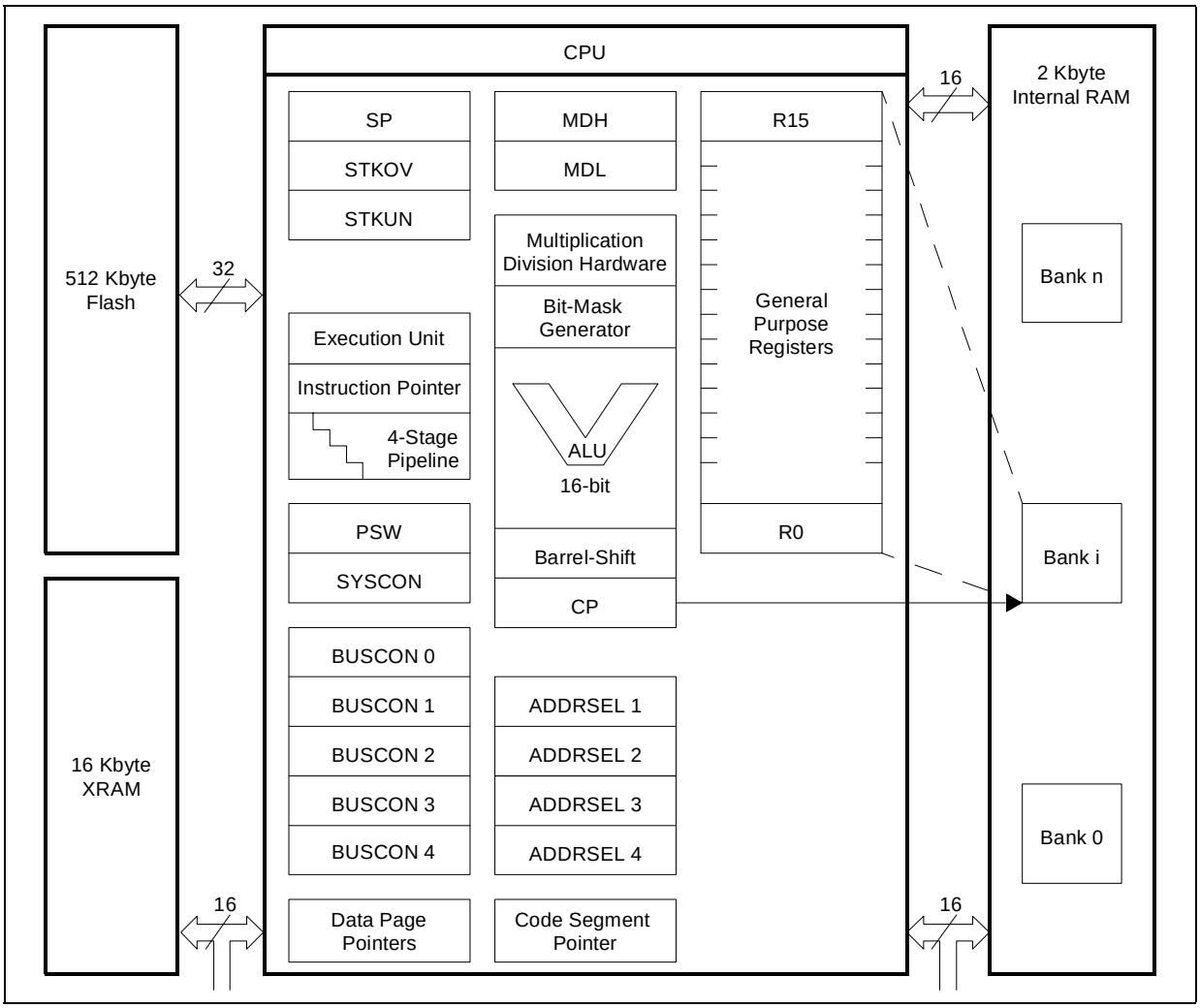


Figure 2 : CPU block diagram



2.1.1 - High Instruction Bandwidth / Fast Execution

Most of the ST10F280's instructions are executed in one instruction cycle. For example, shift and rotate instructions are processed in one instruction cycle independent of the number of bits to be shifted. Multiple-cycle instructions have been optimized: branches are carried out in 2 instruction cycles, 16 × 16 bit multiplication in 5 instruction cycles and a 32/16-bit division in 10 instruction cycles. The jump cache reduces the execution time of repeatedly performed jumps in a loop, from 2 instruction cycles to 1 instruction cycle.

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

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

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

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

– Write back: In this stage, the result is written to the specified location.

If this technique is not used, each instruction would require four instruction cycles. Pipelining offers increased performance.

2.2 - High Function 8-bit and 16-bit ALU

All standard arithmetic and logical operations are performed in a 16-bit ALU. In addition, the condition flags for byte operations are provided from the sixth and seventh bit of the ALU result.

Multiple precision arithmetic is provided through a 'CARRY-IN' signal to the ALU, from previously calculated portions of the desired operation. Most of the internal execution blocks have been optimized to perform operations on either 8-bit or 16-bit data.

Once the pipeline has been filled, one instruction is completed per instruction cycle, except for multiply and divide. An advanced Booth algorithm has been incorporated to allow 4 bits to be multiplied and 2 bits to be divided per instruction cycle. Thus, these operations use two coupled 16-bit registers, MDL and MDH, and require four and nine instruction cycles, respectively, to perform a 16-bit by 16-bit (or 32-bit by 16-bit) calculation plus one instruction cycle to setup and adjust the operands and the result.

Even these longer multiply and divide instructions can be interrupted during their execution to allow very fast interrupt response.

Instructions have also been provided to allow byte packing in memory while providing sign extension of byte for word wide arithmetic operations.

The internal bus structure also allows transfers of byte or words to or from peripherals based on the peripheral requirements.

A set of consistent flags is automatically updated in the PSW after each arithmetic, logical, shift, or movement operation.

These flags allow branching on specific conditions. Support for both signed and unsigned arithmetic is provided through user-specifiable branch tests. These flags are also preserved automatically by the CPU upon entry into an interrupt or trap routine.

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

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

2.2.1 - Extended bit Processing and Peripheral Control

A large number of instructions are dedicated to bit processing. These instructions provide efficient control and testing of peripherals and they enhance data manipulation. Unlike other microcontrollers, these instructions provide direct access to two operands in the bit-addressable space, without the need to move them into temporary flags.

The same logical instructions available for words and byte, are also supported for bit. This allows the user to compare and modify a control bit for a peripheral, in one instruction.

Multiple bit shift instructions have been included to avoid long instruction streams of single bit shift operations. These are also performed in a single instruction cycle. In addition, bit field instructions have been provided to allow the modification of multiple bit from one operand in a single instruction.

2.2.2 - High Performance Branch, Call, and Loop Processing

Due to the high percentage of branching in controller applications, branch instructions have been optimized to require one extra instruction cycle only when a branch is taken. This is implemented by pre-calculating the target address while decoding the instruction.

To decrease loop execution overhead, three enhancements have been provided:

1 Single cycle branch execution is provided after the first iteration of a loop. Therefore, only one instruction cycle is lost during the execution of the entire loop. In loops which fall through upon completion, no instruction cycle is lost when exiting the loop. No special instruction is required to perform loops, and loops are automatically detected during execution of branch instructions.

2 Detection of the end of a table avoids the use of two compare instructions embedded in loops. One simply places the lowest negative number at the end of the specific table, and specifies branching if neither this value nor the compared value have been found. Otherwise the loop is terminated if either condition has been met. The terminating condition can then be tested.

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

Saving of system state is automatically performed on the internal system stack avoiding the use of instructions to preserve state upon entry and exit of interrupt or trap routines. Call instructions push the value of the IP on the system stack, and require the same execution time as branch instructions.

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

2.2.3 - Consistent and Optimized Instruction Formats

To obtain optimum performance in a pipeline design, an instruction set has been designed using concepts of Reduced Instruction Set Computing (RISC).

These concepts primarily allow fast decoding of the instructions and operands, while reducing pipeline holds. These concepts, however, do not preclude the use of complex instructions, which are required by microcontroller users.

The following goals were used to design the instruction set:

– To provide powerful instructions to perform operations which currently require sequences of instructions and which are frequently used. To avoid transfer into and out of temporary registers such as accumulators and carry bit. To perform tasks in parallel such as saving state upon entry into interrupt routines or subroutines.

– To avoid complex encoding schemes by placing operands in consistent fields for each instruction. Also to avoid complex addressing modes which are not frequently used. This decreases the instruction decode time while also simplifying the development of compilers and assemblers.

– To provide most frequently used instructions with one-word instruction formats. All other instructions are placed into two-word formats. This allows all instructions to be placed on word boundaries, which alleviates the need for complex alignment hardware. It also has the benefit of increasing the range for relative branching instructions.

The high performance offered by the hardware implementation of the CPU can efficiently be used by a programmer via the highly functional ST10F280 instruction set. Possible operand types are bits, bytes and words. Specific instruction support the conversion (extension) of bytes to words. A variety of direct, indirect or immediate addressing modes are provided to specify the required operands.

2.2.4 - Programmable Multiple Priority Interrupt System

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

– Peripheral Event Controller (PEC): This processor is used to off-load many interrupt requests from the CPU. It avoids the overhead of entering and exiting interrupt or trap routines by performing single cycle interrupt-driven byte or word data transfers between any two locations in segment 0 with an optional increment of either the PEC source or the destination pointer. Just one cycle is 'stolen' from the current CPU activity to perform a PEC service.

– Multiple Priority Interrupt Controller: This controller allows all interrupts to be placed at any specified priority. Interrupts may also be grouped, which provides the user with the ability to prevent similar priority tasks from interrupting each other. For each of the possible interrupt sources there is a separate control register, which contains an interrupt request flag, an interrupt enable flag and an interrupt priority bit-field. Once having been accepted by the CPU, an interrupt service can only be interrupted by a higher prioritized service request. For standard interrupt processing, each of the possible interrupt sources has a dedicated vector location.

– Multiple Register Banks: This feature allows the user to specify up to sixteen general purpose registers located anywhere in the internal RAM. A single “one instruction cycle” instruction is used to switch register banks from one task to another.

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

With an interrupt response time within a range from just 150ns to 250ns (in case of internal program execution), the ST10F280 is capable of fast reaction to non-deterministic events.

The ST10F280 also provides an excellent mechanism to identify and to process exceptions or error conditions that arise during run-time, so called 'Hardware Traps'. Hardware traps cause an immediate non-maskable system reaction which is similar to a standard interrupt service (branching to a dedicated vector table location).

The occurrence of a hardware trap is additionally signified by an individual bit in the trap flag register (TFR).

Except for another higher prioritized trap service being in progress, a hardware trap will interrupt any current program execution. In turn, hardware trap services can normally not be interrupted by standard or PEC interrupts.

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

2.3 - On-chip System Resources

The ST10F280 controllers provide a number of powerful system resources designed around the CPU. The combination of CPU and these resources results in the high performance of the members of this controller family.

2.3.1 - Peripheral Event Control and Interrupt Control

The Peripheral Event Controller makes it possible to respond to an interrupt request with a single data transfer (word or byte) which only consumes one instruction cycle and does not require a save and restore of the machine status.

Each interrupt source is prioritized in every instruction cycle in the interrupt control block. If a PEC service is selected, a PEC transfer is started. If CPU interrupt service is requested, the current CPU priority level stored in the PSW register is tested to determine whether a higher priority interrupt is currently being serviced.

When an interrupt is acknowledged, the current state of the machine is saved on the internal system stack and the CPU branches to the system specific vector for the peripheral.

The PEC contains a set of SFRs which store the count value and control bit 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 similarly to any other peripheral through SFRs containing the desired configuration of each channel.

An individual PEC transfer counter is implicitly decremented for each PEC service except forming in the continuous transfer mode. When this counter reaches zero, a standard interrupt is performed to the vector location related to the corresponding source. PEC services are very well suited, for example, to move register contents to/from a memory table. The ST10F280 has 8 PEC channels each of which offers such fast interrupt-driven data transfer capabilities.

2.3.2 - Memory Areas

The memory space of the ST10F280 is organized as a unified memory which means that code memory, data memory, registers and I/O ports are organized within the same linear address space which covers up to 16 Mbytes. The entire memory space can be accessed byte wise or word wise. Particular portions of the on-chip memory have additionally been made directly bit addressable.

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

The CPU contains an actual register context, consisting of up to 16 word wide and/or byte wide GPRs which are physically located within the on-chip RAM area.

A Context Pointer (CP) register determines the base address of the active register bank to be accessed by the CPU at a time. The number of register banks is only restricted by the available internal RAM space. For easy parameter passing, one register bank may overlap others.

A system stack of up to 1024 words is provided as a storage for temporary data. The system stack is also located within the on-chip RAM area, and it is accessed by the CPU via the stack pointer (SP) register. Two separate SFRs, STKOV and STKUN, are implicitly compared against the stack pointer value upon each stack access for the detection of a stack overflow or underflow.

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

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

A 512 Kbyte internal Flash provides for both code and constant data storage. This memory area is connected to the CPU via a 32-bit wide bus. Thus, an entire double-word instruction can be fetched in just one instruction cycle. Program execution from the on-chip Flash is the fastest of all possible alternatives.

For Special Function Registers 1024 bytes of the address space are reserved. The standard Special Function Register area (SFR) uses 512 bytes, while the Extended Special Function Register area (ESFR) uses the other 512 bytes. (E)SFRs are word wide registers which are used for controlling and monitoring functions of the different on-chip units. Unused ESFR addresses are reserved for future members of the ST10 family.

2.3.3 - External Bus Interface

In order to meet the needs of designs where more memory is required than is provided on chip, up to 16 Mbytes of external memory can be connected to the microcontroller via its external bus interface.

The integrated External Bus Controller (EBC) allows flexible access to external memory and/or peripheral resources. For up to five address areas the bus mode (multiplexed / de-multiplexed), the data bus width (8-bit / 16-bit) and even the length of a bus cycle (wait-states, signal delays) can be selected independently.

This allows access to a variety of memory and peripheral components, directly and with maximum efficiency. If the device does not run in Single Chip Mode, where no external memory is required, the EBC can control external accesses in one of the following four different external access modes:

– 16-/18-/20-/24-bit Addresses, and 16-bit data, de-multiplexed.

– 16-/18-/20-/24-bit Addresses, and 8-bit data, de-multiplexed.

– 16-/18-/20-/24-bit Addresses, and 16-bit data, multiplexed.

– 16-/18-/20-/24-bit Addresses, and 8-bit data, multiplexed.

The de-multiplexed bus modes use PORT1 for addresses and PORT0 for data input/output. The multiplexed bus modes use PORT0 for both addresses and data input/output. All modes use Port 4 for the upper address lines (A16...) if selected.

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

For applications which require less than 64 Kbytes of address space, a non-segmented memory model can be selected, where all locations can be addressed by 16-bit. Port4 is not needed for the upper address bit (A23/A19/A17...A16), as is the case when using the segmented memory model.

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

The on-chip XRAM and the on-chip CAN-Modules are examples for these X-Peripherals.

2.4 - Clock Generator

The on-chip clock generator provides the ST10F280 with its basic clock signal that controls the activities of the controller hardware. Its oscillator can run with an external crystal and appropriate oscillator circuitry (see Chapter 8 - Dedicated Pins), 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 also referred to as "CPU clock". Two separated clock signals are generated for the CPU itself and the peripheral part of the chip.

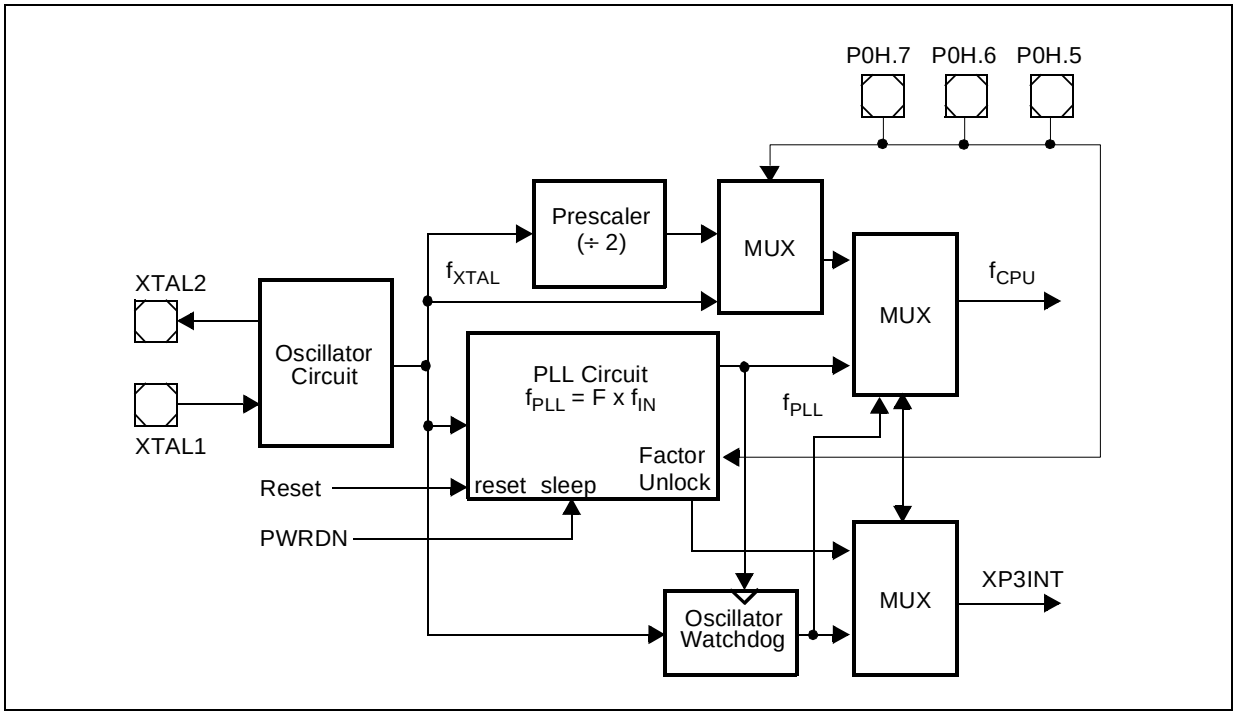
While the CPU clock is stopped during idle mode, the peripheral clock keeps running. Both clocks are switched off when the power-down mode is entered.

The on-chip PLL circuit allows operation of the ST10F280 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 allows the detection of frequency deviations and the execution of emergency actions in case of an external clock failure even when PLL is by passed (see Chapter 13 - Watchdog Timer).

Figure 3 : PLL block diagram



The Table 1 lists all the possible selections for the on-chip clock generator.

Table 1 : On-chip clock generator selections

P0H.7	P0H.6	P0H.5	CPU Frequency $f_{CPU} = f_{XTAL} \times F$	External Clock Input Range[1]	Notes
1	1	1	F_{XTAL} x 4	2.5 to 10 MHz	Default configuration
1	1	0	F_{XTAL} x 3	3.33 to 13.33 MHz	
1	0	1	F_{XTAL} x 2	5 to 20 MHz	
1	0	0	F_{XTAL} x 5	2 to 8 MHz	
0	1	1	F_{XTAL} x 1	1 to 40 MHz	Direct drive[2 4]
0	1	0	F_{XTAL} x 10	1 to 4 MHz	
0	0	1	F_{XTAL} x 0.5	2 to 80 MHz	CPU clock via prescaler[3]
0	0	0	F_{XTAL} x 2.5	4 to 16 MHz	

Notes: 1. The external clock input range refers to a CPU clock range of 1...40 MHz.
2. The maximum depends on the duty cycle of the external clock signal.
3. The maximum input frequency is 25 MHz when using an external crystal with the internal oscillator; providing that internal serial resistance of the crystal is less than 40 Ω. However, higher frequencies can be applied with an external clock source, but in this case, the input clock signal must reach the defined levels V_{IL} and V_{IH2}.
4. The PLL free-running frequency is from 2 to 10 MHz.

2.4.1 - 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 1ms after V_{DD} has reached 5V±10%, even if there is no external clock signal (in this case, the PLL will run on its basic frequency of 2 to 10 MHz).

The PLL starts synchronizing with the external clock signal as soon as it is available. Within 1ms after stable oscillations of the external clock within the specified frequency range, the PLL will be synchronous with this clock at a frequency of F x f_{XTAL}, and the PLL locks to the external clock.

Note If the ST10F280 is required to operate on the desired CPU clock directly after reset, make sure that RSTIN remains active until the PLL has locked (approx 1ms).

The PLL constantly synchronizes to the external clock signal. Due to the fact that the external frequency is 1/F'th of the PLL output frequency, the output frequency may be slightly higher or lower than the desired frequency.

This jitter is irrelevant for longer time periods. For short periods (1...4 CPU clock cycles), it remains below 4%.

When the PLL detects that it is no longer locked (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 (for example due to a broken crystal). In this case, the synchronization mechanism will reduce the PLL output frequency down to the PLL's basic frequency (2 to 10 MHz). The basic frequency is still generated and allows the CPU to execute emergency actions in case of a loss of the external clock.

2.4.2 - 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 is still running on its basic frequency of 2 to 10 MHz, and delivers the clock signal for the Oscillator Watchdog, except if bit OWDDIS is set the PLL is switched off.

2.4.3 - 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 (this means $f_{CPU} = f_{OSC}$). 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 to 10 MHz, and delivers the clock signal for the Oscillator Watchdog.

2.4.4 - Oscillator Watchdog (OWD)

In order to provide a fail safe mechanism for the instance of a loss of the external clock, an oscillator watchdog is implemented when the selected clock option is direct drive or direct drive with prescaler.

The oscillator watchdog operates as follows:

– The Oscillator WatchDog (OWD) 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 will be switched to the PLL clock signal (in this case, the PLL will run on its basic frequency of 2 to 10 MHz), and the Oscillator Watchdog Interrupt Request (XP3INT) is flagged.

The CPU clock will not switch back to the external clock even if a valid external clock exits on 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.

2.5 - On-chip Peripheral Blocks

The ST10 family of devices separates peripherals from the core. This allows peripherals to be added or deleted without modifications to the core. Each functional block processes data independently and communicates information over common buses. Peripherals are controlled by data written to the respective Special Function Registers (SFRs). These SFRs are located either within the standard SFR area (00'FE00h...00'FFFFh), or within the extended ESFR area (00'F000h...00'F1FFh).

The built in peripherals are used for interfacing the CPU to the external world, or to provide on-chip functions. The ST10F280 generic peripherals are:

– Two General Purpose Timer Blocks (GPT1 and GPT2),

– Two Serial Interfaces (ASC0 and SSC),

– A Watchdog Timer,

– Two 16-channel Capture / Compare units (CAPCOM1 and CAPCOM2),

– A 8-channel Pulse Width Modulation unit,

– Two 10-bit Analog / Digital Converters,

– Eleven I/O ports with a total of 143 I/O lines.

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

2.5.1 - 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 CPU and peripherals is performed through Special Function Registers (SFRs) and interrupts. The SFRs serve as control/status and data registers for the peripherals. Interrupt requests are generated by the peripherals based on specific events which occur during their operation like end of task, new event, error...

Specific pins of the parallel ports are used for interfacing 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 I/O pin.

2.5.2 - 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. During Idle mode the CPU's clock is stopped while the peripherals continue their operation. Peripheral SFRs may be accessed by the CPU once per state.

When an SFR is written to by software in the same state where it is also to be modified by the peripheral, the software write operation has priority. Further details on peripheral timing are included in the specific sections about each peripheral.

2.5.3 - Programming Hints

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

– Indirect or direct addressing with **16-bit (mem) addresses** it must be guaranteed that the used data page pointer (DPP0...DPP3) selects data in memory space page 3.

– Accesses via the Peripheral Event Controller (**PEC**) use the SRCPx and DSTPx pointers instead of the data page pointers.

– **short 8-bit (reg) addresses** to the standard **SFR** area do not use the data page pointers but directly access the registers within this 512 byte area.

– **short 8-bit (reg) addresses** to the extended **ESFR** area require switching to the 512 byte extended SFR area. This is done via the EXTension instructions EXTR, EXTP(R), EXTS(R).

Byte write operations to word wide SFRs via indirect or direct 16-bit (mem) addressing or byte transfers via the PEC force zeros in the non-addressed byte. Byte write operations via short 8-bit (reg) addressing can only access the low byte of an SFR and force zeros in the high byte. It is therefore recommended, to use the bit-field instructions (BFLDL and BFLDH) to write to any number of bit in either byte of an SFR without disturbing the non-addressed byte and the unselected bit.

Reserved bit, some of the bit which are contained in the ST10F280's SFRs are marked as 'Reserved'. User software must write '0's to reserved bit.

These bit are currently not implemented and may be used in future products to invoke new functions. In this case, the active state for these functions will be '1', and the inactive state will be '0'. Therefore writing only '0's to reserved locations provides portability of the current software to future devices. Read accesses to reserved bit return '0's.

2.5.4 - Parallel Ports

The ST10F280 provides up to 143 I/O lines which are organized into nine input/output ports and two input ports. All port lines are bit-addressable, and all input/output lines are individually (bit wise) programmable as inputs or outputs via direction registers. The I/O ports are true bidirectional ports which are switched to high impedance state when configured as inputs.

The output drivers of six I/O ports can be configured (pin by pin) for push-pull operation or open-drain operation via control registers. During the internal reset, all port pins are configured as inputs.

All pins of I/O ports also support an alternate programmable function:

– PORT0 supports the 8/16 data lines of the external bus in demultiplexed mode and the 8/16 address data lines of the external bus in multiplexed mode.

– PORT1 supports the 8/16 address line of the external bus in demultiplexed mode and 4 inputs (CC27...24) of the capture compare unit 2.

– Port2, accepts the fast external interrupt inputs and provides inputs/outputs for CAPCOM1 unit.
– Port3 includes the alternate functions of timers, serial interfaces, the optional bus control signal $\overline{BHE}$ and the system clock output (CLKOUT).
– Port4 outputs the additional segment address bit A16 to A23 in systems where segmentation is enabled to access more than 64 Kbytes of memory and the CANs lines also.
– Port5 is used as analog input channels of the A/D converter or as timer control signals.
– Port6 provides optional bus arbitration signals ($\overline{BREQ}$, $\overline{HLDA}$, $\overline{HOLD}$) and chip select signals.
– Port7 provides the output signals from the PWM unit and inputs/outputs for the CAPCOM2 unit.
– Port8 provides inputs/outputs for the CAPCOM2 unit. Four pins of PORT1 may also be used as inputs only for the CAPCOM2 unit.

All port lines that are not used for alternate functions may be used as general purpose I/O lines.

– Port XP9 is a standard I/O port without alternate function.
– Port XP10 is a general purpose input port and also supports the 16 multiplexed A/D (AN31...16) inputs.

2.5.5 - Serial Channels

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

They support full-duplex asynchronous communication and half-duplex synchronous communication. The SSC may be configured so it interfaces with serially linked peripheral components. Two dedicated Baud rate generators allow to set up all standard Baud rates without oscillator tuning. For transmission, reception and error handling 3 separate interrupt vectors are provided on channel SSC, 4 vectors are provided on channel 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. For multiprocessor communication, a mechanism to distinguish address from data byte has been included (8-bit data plus wake up bit mode).

In synchronous mode, the ASC0 transmits or receives byte (8-bit) synchronously to a shift clock which is generated by the ASC0. The SSC transmits or receives characters of 2...16-bit length synchronously to a shift clock which can be generated by the SSC (master mode) or by an external master (slave mode). The SSC can start shifting with the LSB or with the MSB, while the ASC0 always shifts the LSB first. A loop back option is available for testing purposes.

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

2.5.6 - The on-chip CAN Modules

The integrated CAN Modules handle the completely autonomous transmission and reception of CAN frames in accordance with the CAN specification V2.0 part B (active). A on-chip CAN Module can receive and transmit standard frames with 11-bit identifiers as well as extended frames with 29-bit identifiers.

The module provides Full CAN functionality on up to 15 message objects. Message object 15 may be configured for Basic CAN functionality.

Both modes provide separate masks for acceptance filtering which allows to accept a number of identifiers in Full CAN mode and also allows to disregard a number of identifiers in Basic CAN mode. All message objects can be updated independent from the other objects and are equipped for the maximum message length of 8 bytes. The bit timing is derived from the XCLK (which is the same than the CPU clock) and is programmable up to a data rate of 1 MBaud. The CAN Module uses two pins to interface to a bus transceiver.

2.5.7 - General Purpose Timer (GPT) Unit

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

The five 16-bit timers are organized into two separate modules, GPT1 and GPT2. Each timer in each module may operate independently in a number of different modes, or may be concatenated with another timer of the same module.

Each timer can be configured individually for one of three basic modes of operation, which are Timer, Gated Timer, and Counter Mode. In Timer Mode the input clock for a timer is derived from the internal CPU clock divided by a programmable prescaler, while Counter Mode allows a timer to be clocked in reference to external events (via TxIN). Pulse width or duty cycle measurement is supported in Gated Timer Mode where the operation of a timer is controlled by the 'gate' level on its external input pin TxIN.

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

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

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

2.5.8 - Watchdog Timer

The Watchdog Timer is a fail-safe mechanism. It limits the maximum malfunction time of the controller

– 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. 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 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.

2.5.9 - Capture / Compare (CAPCOM) Units

The two CAPCOM units support generation and control of timing sequences on up to 32 channels. The CAPCOM units are typically used to handle high speed I/O tasks such as pulse and waveform generation, pulse width modulation (PMW), Digital to Analog (D/A) conversion, software timing, or time recording relative to external events.

Four 16-bit timers (T0/T1, T7/T8) with reload registers, provide two independent time bases for the capture/compare register array.

The input clock for the timers is programmable to several pre-scaled values of the internal system clock, or may be derived from an overflow/underflow of timer T6 in module GPT2.

This provides a wide range of variation for the timer period and resolution and allows precise adjustments to the application specific requirements. In addition, external count inputs for CAPCOM timers T0 and T7 allow event scheduling for the capture/compare registers relative to external events.

Both of the two capture/compare register arrays contain 16 dual purpose capture/compare registers, each of which may be individually allocated to either CAPCOM timer T0 or T1 (T7 or T8, respectively), and programmed for capture or compare function.

Each register has one port pin associated with it which is an input pin for triggering the capture function, or is an output pin (except for CC24...CC27) to indicate the occurrence of a compare event.

When a capture/compare register has been selected for capture mode, the current contents of the allocated timer will be latched (captured) into the capture/compare register in response to an external event at the port pin which is associated with this register.

In addition, a specific interrupt request for this capture/compare register is generated. Either a positive, a negative, or both a positive and a negative external signal transition at the pin can be selected as the triggering event.

The contents of all registers which have been selected for one of the five compare modes are continuously compared with the contents of the allocated timers.

When a match occurs between the timer value and the value in a capture/compare register, specific actions will be taken, based on the selected compare mode.

2.5.10 - Pulse Width Modulation Unit

The Pulse Width Modulation Module can generate up to four PWM output signals using edge-aligned or centre-aligned PWM. In addition the PWM module can generate PWM burst signals and single shot outputs.

In Burst Mode two channels can be combined with their output signals ANDed, where one channel gates the output signal of the other channel. In Single Shot Mode, a single output pulse is generated (retriggerable) under software control.

Each PWM channel is controlled by an up/down counter with associated reload and compare registers. The polarity of the PWM output signals may be controlled via the respective port output latch (combination via EXOR).

2.5.11 - A/D Converter

A 10-bit A/D converter with 2 x 16 multiplexed input channels and a sample and hold circuit has been integrated on-chip for analog signal measurement.

The 16 A/D inputs from Port5 and the 16 A/D inputs from XPort10 are selected thanks to the XADMUX register.

The A/D uses a successive approximation method. The sample time (for loading the capacitors) and conversion time is programmable and can be modified for the external circuitry.

Overrun error detection/protection is provided for the conversion result register (ADDAT). When the result of a previous conversion has not been read from the result register at the time the next conversion is complete, either an interrupt request is generated, or the next conversion is suspended, until the previous result has been read.

For applications which require less than 16 analog input channels, the remaining channel inputs can be used as digital input port pins.

The A/D converter of the ST10F280 supports four different conversion modes:

– Standard Single Channel conversion mode, the analog level on a specified channel is sampled once and converted to a digital result.
– Single Channel Continuous mode, the analog level on a specified channel is repeatedly sampled and converted without software intervention.
– For the Auto Scan mode, the analog levels on a pre-specified number of channels are sequentially sampled and converted.
– In the Auto Scan Continuous mode, the number of pre-specified channels is repeatedly sampled and converted.
– In addition, the conversion of a specific channel can be inserted (injected) into a running sequence without disturbing this sequence. This is called Channel Injection Mode. The Peripheral Event Controller (PEC) may be used to automatically store the conversion results into a table in memory for later evaluation, without the overhead of interrupt routines for each data transfer.

2.6 - Protected bits

The ST10F280 MCU provide 106 protected bits. These bits are modified by the on-chip hardware during special events like power-on reset, power failure, application hardware, etc. These bits cannot be modified by some wrong software accesses.

Table 2 : Protected bit

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
T0IC, T1IC	T0IR, T1IR	CAPCOM1 timer interrupt request flags
T7IC, T8IC	T7IR, T8IR	CAPCOM2 timer interrupt request flags
S0TIC, S0TBIC	S0TIR, S0TBIR	ASC0 transmit(buffer) interrupt request flags
S0RIC, S0EIC	S0RIR, S0EIR	ASC0 receive/error interrupt request flags
S0CON	S0REN	ASC0 receiver enable flag
SSCTIC, SSCRIC	SSCTIR, SSCRIR	SSC transmit/receive interrupt request flags
SSCEIC	SSCEIR	SSC error interrupt request flag
SSCCON	SSCBSY	SSC busy flag
SSCCON	SSCBE, SSCPE	SSC error flags
SSCCON	SSCRE, SSCTE	SSC error flags
ADCIC, ADEIC	ADCIR, ADEIR	ADC end-of-conversion/overrun interrupt request flag
ADCON	ADST, ADCRQ	ADC start flag / injection request flag
CC31IC...CC16IC	CC31IR...CC16IR	CAPCOM2 interrupt request flags
CC15IC...CC0IC	CC15IR...CC0IR	CAPCOM1 interrupt request flags
PWMIC	PWMIR	PWM module interrupt request flag
TFR	TFR.15,14,13	Class A trap flags
TFR	TFR.7,3,2,1,0	Class B trap flags
P2	P2.15...P2.0	All bit of Port2
P7	P7.7...P7.0	All bit of Port7
P8	P8.7...P8.0	All bit of Port8
XPyIC (y=3...0)	XPyIR (y=3...0)	X-Peripheral y interrupt request flag

Note : Σ = 106 protected bit.

3 - MEMORY ORGANIZATION

The memory space of the ST10F280 is organized as a unified memory. Code memory, data memory, registers and I/O ports are organized within the same linear address space.

All of the physically separated memory areas, including internal Flash, internal RAM, the internal Special Function Register Areas (SFRs and ESFRs), the address areas for integrated XBUS peripherals like XRAM or CAN module and external memory are mapped into one common address space.

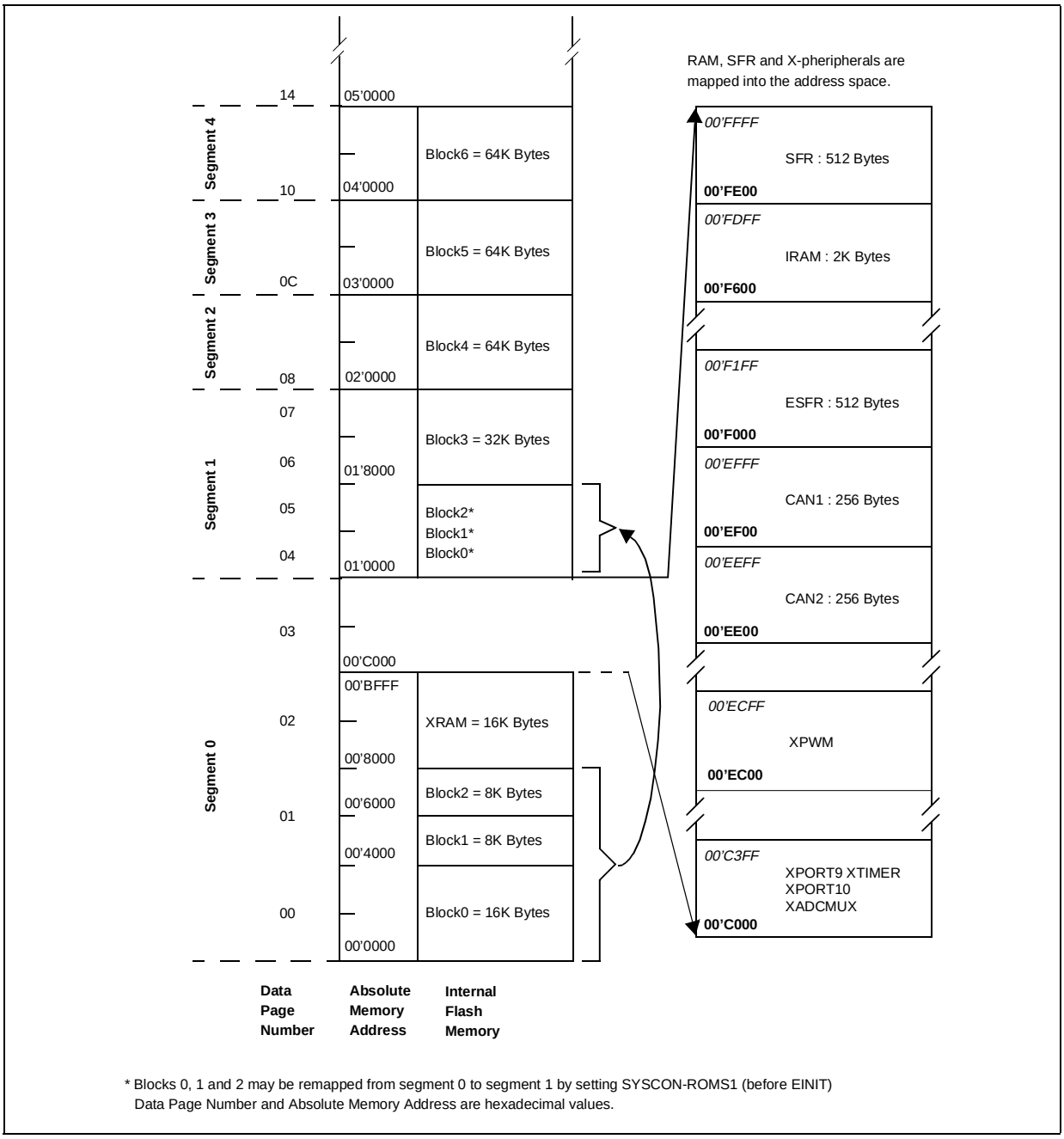
The ST10F280 provides a total addressable memory space of 16 Mbytes. This address space is arranged as 256 segments of 64 Kbytes each, and each segment is again subdivided into four data pages of 16 Kbytes each (see Figure 4).

Most of the internal memory areas are mapped into segment 0, named system segment. The upper 4 Kbytes of segment 0 (00'F000h...00'FFFFh) hold the Internal RAM and Special Function Register Areas (SFR and ESFR). The lower 32 Kbytes of segment 0 (00'0000h...00'7FFFh) can be occupied by a part of the on-chip Flash Memory and is called the internal memory area.

This Flash area can be remapped to segment 1 (01'0000h...01'7FFFh), to enable external memory access in the lower half of segment 0, or the internal Flash may be disabled.

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

Figure 4 : Memory areas and address space



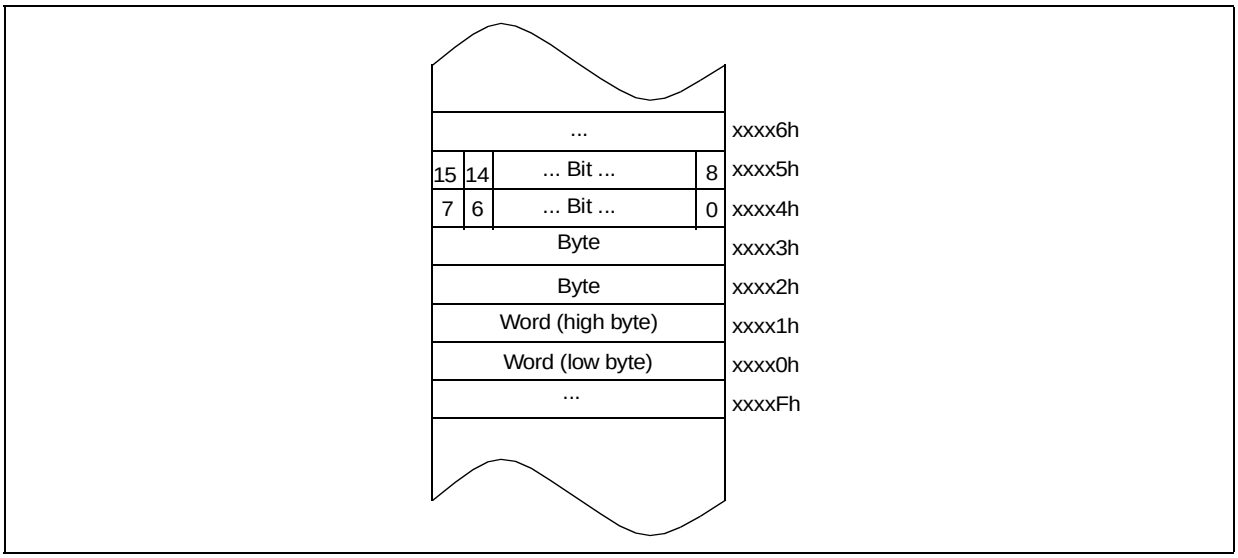
* Blocks 0, 1 and 2 may be remapped from segment 0 to segment 1 by setting SYSCON-ROMS1 (before EINIT)
Data Page Number and Absolute Memory Address are hexadecimal values.

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

Double words (code only) are stored in ascending memory locations as two subsequent words. Single bit 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, byte and bit 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, Flash, RAM) and organizational (page, segment) memory area.

3.1 - Internal Flash

The ST10F280 reserves an address area of 512 Kbytes for on-chip Flash memory.

In standard mode (the normal operating mode) the Flash appears like an on-chip ROM with the same timing and functionality. The Flash module offers a fast access time, allowing zero wait-state access with CPU frequency up to 40 MHz. Instruction fetches and data operand reads are performed with all addressing modes of the ST10F280 instruction set.

In order to optimize the programming time of the internal Flash, blocks of 8 Kbytes, 16 Kbytes, 32 Kbytes, 64 Kbytes can be used. But the size of the blocks does not apply to the whole memory space, see details in Table 3.

Table 3 : 512 Kbyte Flash memory block organization

Block	Addresses (Segment 0)	Addresses (Segment 1)	Size (K Byte)
0	00'0000h to 00'3FFFh	01'0000h to 01'3FFFh	16
1	00'4000h to 00'5FFFh	01'4000h to 01'5FFFh	8
2	00'6000h to 00'7FFFh	01'6000h to 01'7FFFh	8
3	01'8000h to 01'FFFFh	01'8000h to 01'FFFFh	32
4	02'0000h to 02'FFFFh	02'0000h to 02'FFFFh	64
5	03'0000h to 03'FFFFh	03'0000h to 03'FFFFh	64
6	04'0000h to 04'FFFFh	04'0000h to 04'FFFFh	64
7	05'0000h to 05'FFFFh	05'0000h to 05'FFFFh	64
8	06'0000h to 06'FFFFh	06'0000h to 06'FFFFh	64
9	07'0000h to 07'FFFFh	07'0000h to 07'FFFFh	64
10	08'0000h to 08'FFFFh	08'0000h to 08'FFFFh	64

Code fetches are always made on even byte addresses. The highest possible code storage location in the internal Flash is either xx'xxFEh for single word instructions, or xx'xxFCh for double word instructions.

The respective location must contain a branch instruction (unconditional), because sequential boundary crossing from internal Flash to external memory is not supported and causes erroneous results.

Any word and byte data read accesses may use the indirect or long 16-bit addressing modes. There is no short addressing mode for internal Flash operands. Any word data access is made to an even byte address.

The highest possible word data storage location in the internal Flash is xx'xxFEh. For PEC data transfers the internal Flash can be accessed independently of the contents of the DPP registers via the PEC source and destination pointers.

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

The internal Flash may be mapped into segment 0 or segment 1 under software control. "Handling the internal Flash" Section 22.10 - Handling the Internal Flash describes the mapping procedures and precautions.

3.2 - Internal RAM and SFR Area

The RAM/SFR area is located within data page 3 and provides access to the 2 Kbyte Internal RAM (organized as 1K x 16) and to two 512 byte blocks of Special Function Registers (SFRs). The internal RAM is used as:

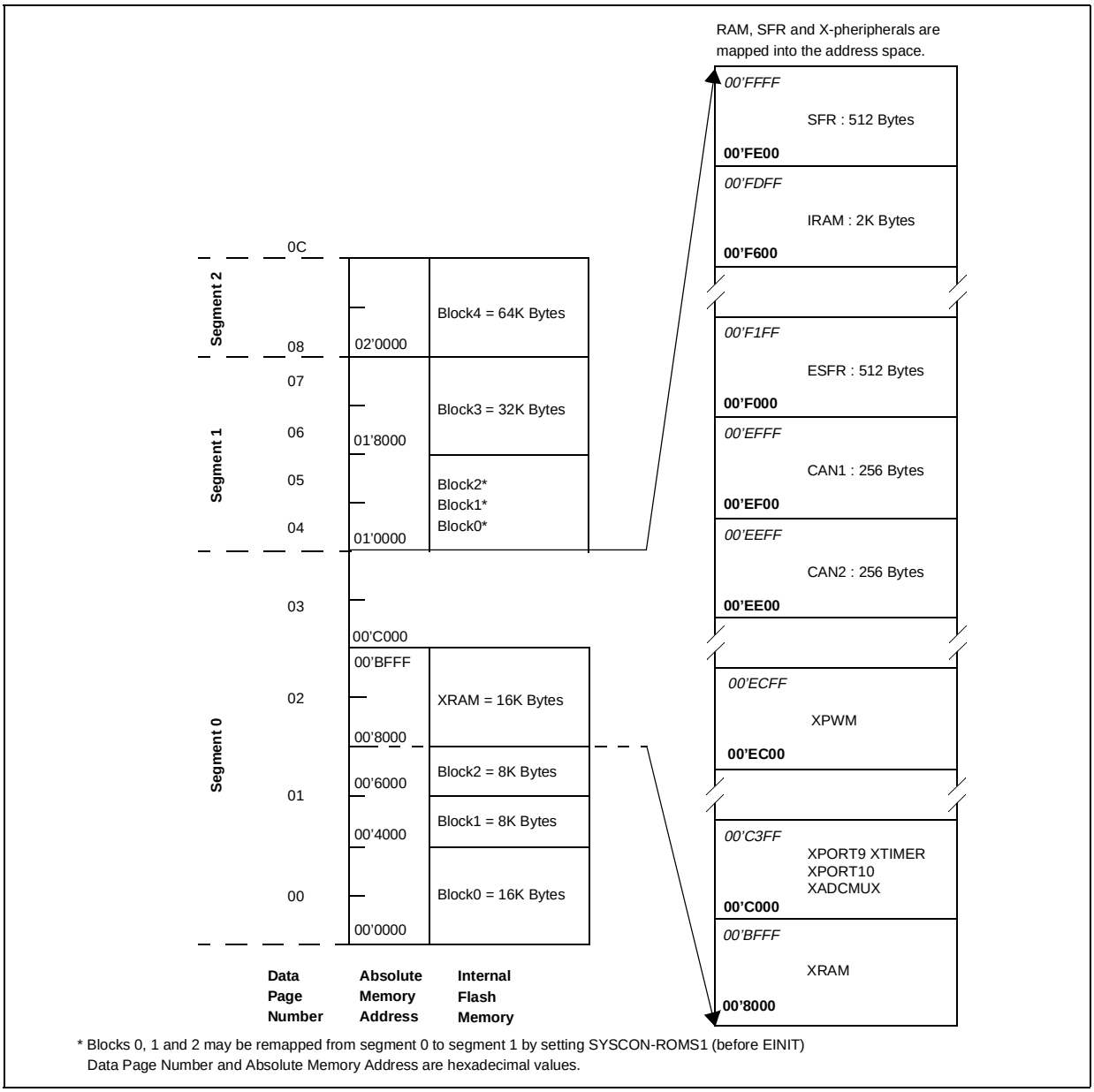
– System Stack (programmable size),

– General Purpose Register Banks (GPRs),

– Source and destination pointers for the Peripheral Event Controller (PEC),

– Variable and other data storage, or Code storage.

Figure 6 : Internal RAM and SFR/ESFR areas



* Blocks 0, 1 and 2 may be remapped from segment 0 to segment 1 by setting SYSCON-ROMS1 (before EINIT)
Data Page Number and Absolute Memory Address are hexadecimal values.

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

2. Read or write access in reserved locations may cause unexpected behaviour.

Code accesses are always made on even byte addresses. The highest possible 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 can causes erroneous results.

Any word and byte data in the internal RAM can be accessed via indirect or long 16-bit addressing modes, if the selected DPP register points to data page 3. Any word data access is made on an even byte address. The highest possible word data storage location in the internal RAM is 00'FDFEh. For PEC data transfers, the internal RAM can be accessed independently 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 therefore, they are bit addressable (see Figure 6).

3.2.1 - System Stack

The system stack may be defined within the internal RAM. The size of the system stack is controlled by bit-field STKSZ in the SYSCON register (see Table 4).

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 are provided to control the lower and upper limits of the selected stack area.

These two stack boundary registers can be used, not only for protection against data destruction, but also allow to implement a circular stack with hardware supported system stack flushing and filling (except for the 2 Kbyte stack option).

The technique of implementing this circular stack is described in Section 22.1 - Stack Operations Circular (virtual) Stack.

Table 4 : Stack size

(STKSZ)	Stack Size (words)	Internal RAM Addresses (words)
0 0 0b	256	00'FBFEh...00'FA00h (Default after Reset)
0 0 1b	128	00'FBFEh...00'FB00h
0 1 0b	64	00'FBFEh...00'FB80h
0 1 1b	32	00'FBFEh...00'FBC0h
1 0 0b	512	00'FBFEh...00'F800h
1 0 1b	---	Reserved. Do not use this combination.
1 1 0b	---	Reserved. Do not use this combination.
1 1 1b	1024	00'FDFEh...00'F600h (Note: No circular stack)

3.2.2 - General Purpose Registers

The general purpose registers (GPRs) use a block of 16 consecutive words within the internal RAM. The Context Pointer (CP) register determines the base address of the currently active register bank. This register bank may consist of up to 16 word GPRs (R0, R1, ..., R15) and/or of up to 16 byte GPRs (RL0, RH0, ..., RL7, RH7) and 8 word registers R8-R15. The sixteen byte GPRs are mapped onto the first eight word GPRs (see Table 5).

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 ST10F280 supports fast register bank (context) switching. Multiple register banks can physically exist within the internal RAM at the same time. Only the register bank selected by the Context Pointer register (CP) is active at a given time. Selecting a new active register bank is simply done by updating the CP register.

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

Details on using, switching and overlapping register banks are described in Register Banking Section 22.2 - Register Banking.

Table 5 : Mapping of general purpose registers to RAM addresses

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
(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

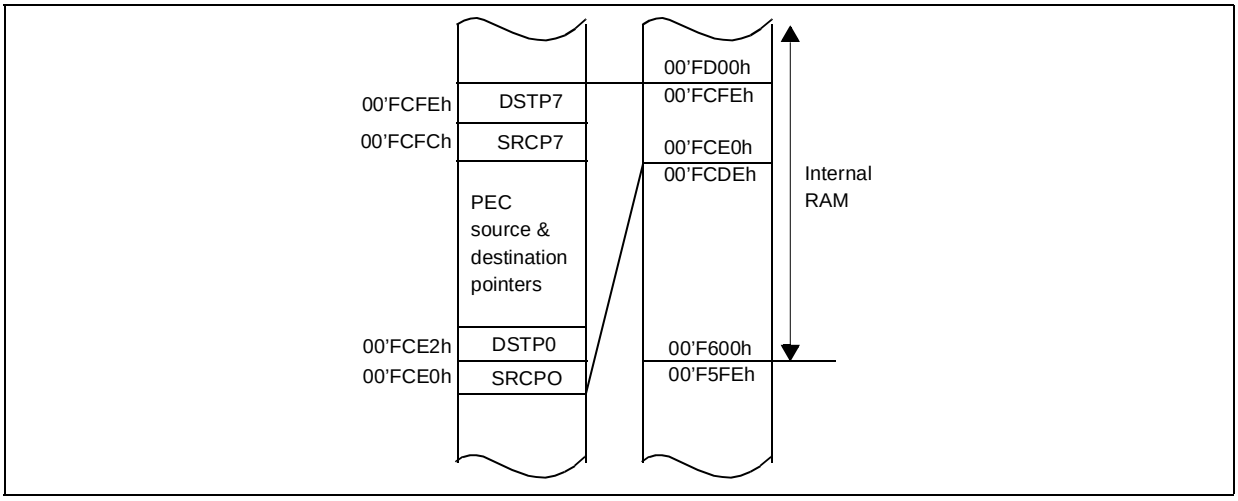
3.2.3 - PEC Source and Destination Pointers

The 16 word locations in the internal RAM from 00'FCE0h to 00'FCFEh are provided as source and destination address pointers for data transfers on the eight PEC channels. Each channel uses a pair of pointers stored in two subsequent word locations with the source pointer (SRCPx) on the lower and the destination pointer (DSTPx) on the higher word address (x = 7...0) (see Figure 7).

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

For more details about the use of the source and destination pointers for PEC data transfers see Chapter 22 - System Programming.

Figure 7 : Location of the PEC pointers



3.2.4 - Special Function Registers

The functions of the CPU, the bus interface, the I/O ports and the on-chip peripherals of the ST10F280 are controlled via a number of so-called Special Function Registers (SFRs).

These SFRs are arranged within two areas, each of 512 byte size. The first register block, is called the SFR area, and is located in the 512 byte above the internal RAM (00'FFFFh...00'FE00h), the second register block, the Extended SFR (ESFR) area, is located in the 512 byte 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 byte. This **does not work** for the respective high byte!

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

The upper half of each register block is bit-addressable, so the respective control/status bit can be directly modified or checked by using bit addressing. When accessing registers in the ESFR area using 8-bit addresses or direct bit addressing, an Extend Register (EXTR) instruction is required before, to switch the short addressing mechanism from the standard SFR area to the Extended SFR area.

This is not required for 16-bit and indirect addresses. The GPRs R15...R0 are duplicated, and they are accessible within both register blocks via short 2-, 4- or 8-bit addresses without switching.

Example:

```
EXTR    #4                  ;Switch to ESFR area for the next 4 instructions
MOV     ODP2,       #data16 ;ODP2 uses 8 Bit reg addressing
BFLDL   DP6,        #data8  ;Bit addressing for Bit fields
                    #mask
BSET    DP1h.7              ;Bit addressing for single Bit
MOV     T8REL, R1           ;T8REL uses 16 Bit address, R1 is duplicated and
                            ;also accessible
                            ;via the ESFR mode (EXTR is not required for
                            ;this access)
;-----  ;------             ;The scope of the EXTR #4 instruction ends here!
MOV     T8REL, R1           ;T8REL uses 16 Bit address, R1 is duplicated and
                            ;does not require switching
```

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

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

3.3 - The On-chip XRAM

The 16 Kbytes of on-chip extension RAM (single port XRAM) is provided as a storage for data, and code. It is located within data page 2.

XRAM: 16K Bytes of on-chip extension RAM (single port XRAM) is provided as a storage for data, user stack and code. The XRAM is a single bank, connected to the internal XBUS and are accessed like an external memory in 16-bit demultiplexed bus-mode without waitstate or read/write delay (50ns access at 40MHz CPU clock). Byte and word access is allowed.

The XRAM address range is 00'8000h 00'BFFFh if enabled (XPEN set bit 2 of SYSCON register-, and XRAMEN set bit 2 of XPERCON register-). If bit XRAMEN or XPEN is cleared, then any access in the address range 00'8000h 00'BFFFh will be directed to external memory interface, using the BUSCONx register corresponding to address matching ADDRSELx register.
As the XRAM appears like external memory, it cannot be used for the ST10F280's system stack or register banks. The XRAM is not provided for single bit storage and therefore is not bit addressable.

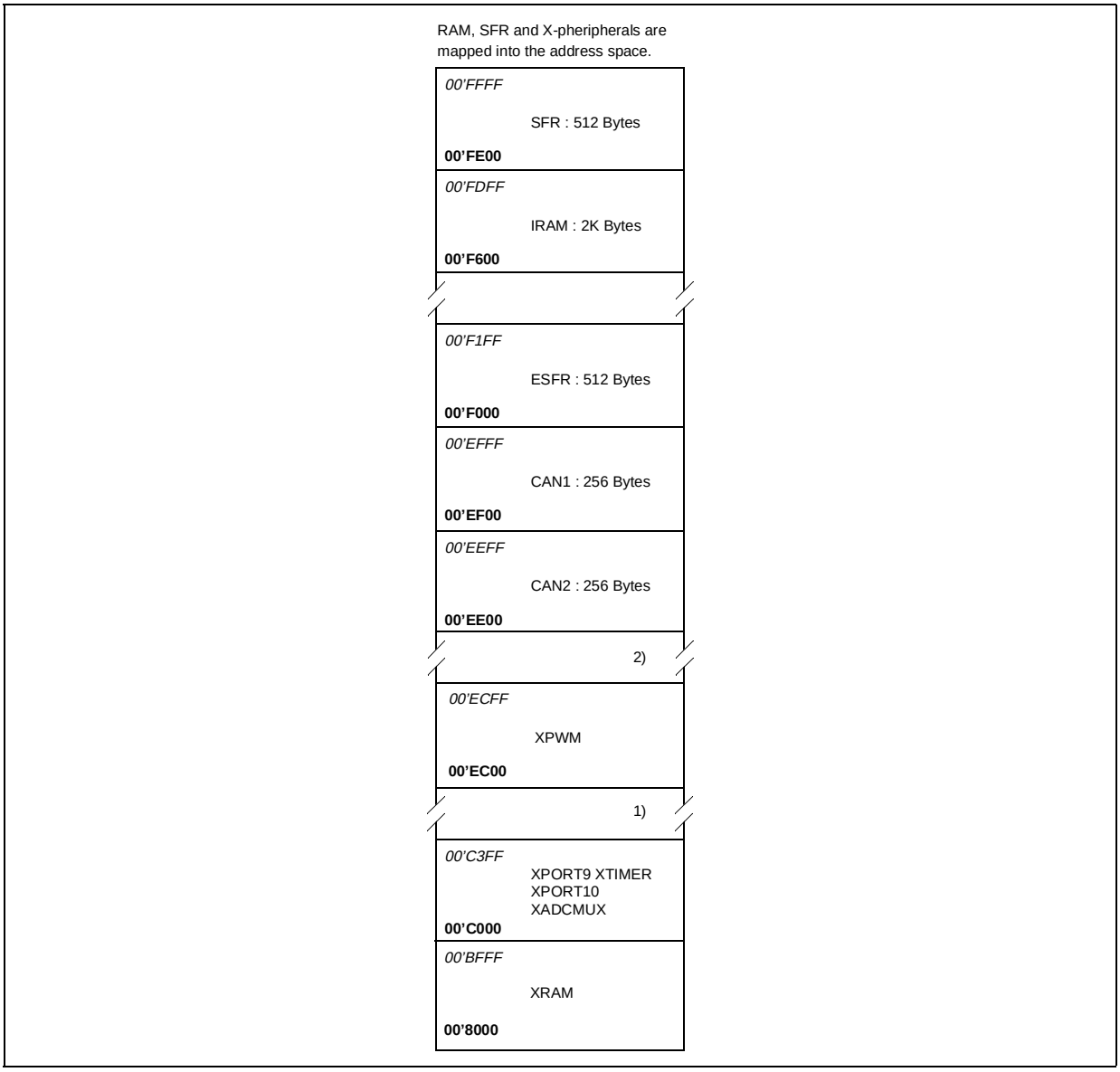
Any word and byte data read accesses may use the indirect or long 16-bit addressing modes. There is no short addressing mode for XRAM operands. Any word data access is made to an even byte address. The highest possible word data storage location in the XRAM is 00'BFFEh. For PEC data transfers the XRAM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

3.3.1 - XRAM Access Via External Masters

When bit XPER-SHARE in register SYSCON is set the on-chip XRAM of the ST10F280 can be accessed by an external master during hold mode, via the ST10F280's bus interface. These external accesses must use the same configuration as the internally programmed. No wait-states are required.

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

Figure 8 : On-chip XRAM area



Note 1. The address area 00'C400h to 00'EBFFh is mapped to external memory but should be reserved for reasons of upward compatibility.

2. The address area 00'ED00h to 00'EDFFh is mapped to external memory only when XPWM, CAN1 and CAN2 are disabled.

3.4 - External Memory Space

The ST10F280 is capable of using an address space of up to 16 Mbytes. Only parts of this address space are occupied by internal memory areas. All addresses which are not used for on-chip memory (Flash) or for registers, may refer to external memory locations. This external memory is accessed via the ST10F280's external bus interface.

Four memory bank sizes are supported:

– Non-segmented mode: 64 Kbytes with A15...A0 on PORT0 or PORT1

– 2-bit segmented mode: 256 Kbytes with A17...A16 on Port4 and A15...A0 on PORT0 or PORT1

– 4-bit segmented mode: 1 Mbyte with A19...A16 on Port4 and A15...A0 on PORT0 or PORT1

– 8-bit segmented mode: 16 Mbytes with A23...A16 on Port4 and A15...A0 on PORT0 or PORT1

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

The ST10F280 also supports **four different bus types**:

– Multiplexed 16-bit Bus with address and data on PORT0 (Default after Reset)

– Multiplexed 8-bit Bus with address and data on PORT0/P0L

– De-multiplexed 16-bit Bus with address on PORT1 and data on PORT0

– De-multiplexed 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, (See Chapter 9 - The External Bus Interface).

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

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

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

3.5 - Crossing Memory Boundaries

The address space of the ST10F280 is implicitly divided into equally sized blocks of different granularity and into logical memory areas. Crossing the boundaries between these blocks (code or data) or areas requires special attention to ensure that the controller executes the desired operations.

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

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

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

Segments are contiguous blocks of 64 Kbytes each. They are referenced via the code segment pointer CSP for code fetches and via an explicit segment number for data accesses overriding the standard DPP scheme.

During code fetching segments are not changed automatically, but rather must be switched explicitly. The instructions JMPS, CALLS and RETS will do this.

In larger sequential programs make sure that the highest used code location of a segment contains an unconditional branch instruction to the respective following segment, to prevent the prefetcher from trying to leave the current segment.

Data Pages are contiguous blocks of 16 Kbytes each. They are referenced via the data page pointers DPP3...0 and via an explicit data page number for data accesses overriding the standard DPP scheme. Each DPP register can select one of the possible 1024 data pages. The DPP register that is used for the current access is selected via the two upper bit of the 16-bit data address. Subsequent 16-bit data addresses that cross the 16 Kbytes data page boundaries therefore will use different data page pointers, while the physical locations need not be subsequent within memory.

4 - THE CENTRAL PROCESSING UNIT (CPU)

The CPU is used to fetch and decode instructions, to supply operands for the arithmetic and logic unit (ALU), to perform operations on these operands in the ALU, and to store the previously calculated results.

A four stage pipeline is implemented, where up to four instructions can be processed in parallel. Most instructions of the ST10F280 are executed in one instruction cycle due to this parallelism.

This chapter describes how the pipeline works for sequential and branch instructions in general, and which hardware provisions have been made to speed the execution of jump instructions in particular. The general instruction timing is described, including standard and exceptional timing.

While internal memory accesses are normally performed by the CPU itself, external peripheral or memory accesses are performed by a particular on-chip External Bus Controller (EBC), which is automatically invoked by the CPU whenever a code or data address refers to the external address space.

If possible, the CPU continues to operate 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 The External Bus Interface (see Chapter 9 - The External Bus Interface).

The on-chip peripheral units of the ST10F280 are almost independent of the CPU, with a separate clock generator.

Data and control information is interchanged between the CPU and these peripherals via Special Function Registers (SFRs).

Whenever peripherals need a non-deterministic CPU action, an on-chip Interrupt Controller compares all pending peripheral interrupt requests 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 service will occur. There are two types of interrupt processing:

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

2 **PEC interrupt processing** steals just one instruction cycle from the current CPU activity to perform a single data transfer via the on-chip PEC.

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

There is a close conjunction between the watchdog timer and the CPU. If enabled, the watchdog timer expects to be serviced by the CPU within a programmable period of time, otherwise it will reset the chip.

Therefore, the watchdog timer is able to prevent the CPU from going totally astray when executing erroneous code. After reset, the watchdog timer starts counting automatically, but if necessary it can be disabled via software.

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

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

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

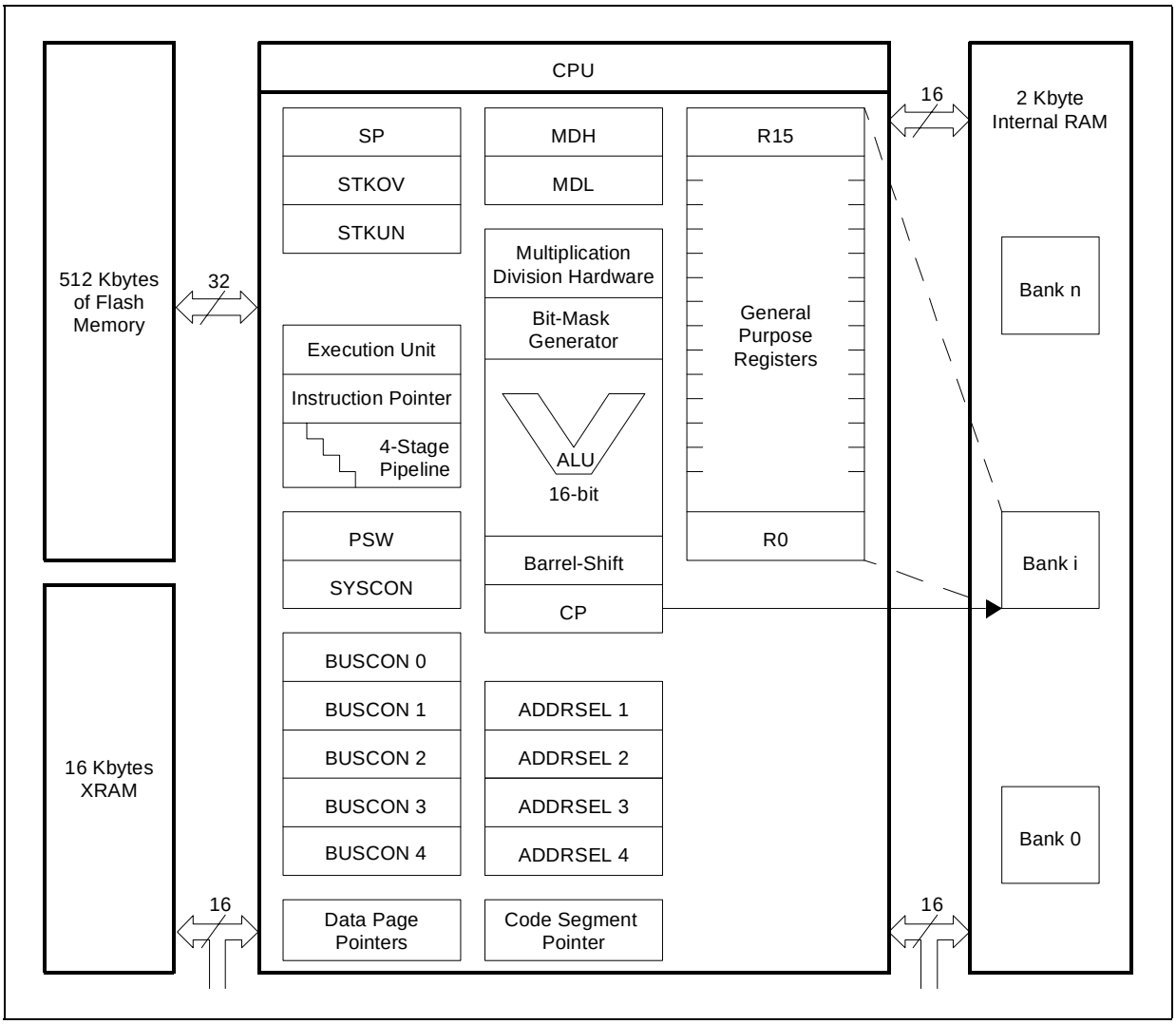
– **POWER DOWN state:** All of the on-chip clocks are switched off.

A transition into an active CPU state is forced by an interrupt (if being IDLE) or by a reset (if being in POWER DOWN mode).

The IDLE, POWER DOWN and RESET states can be entered by particular ST10F280 system control instructions. A set of Special Function Registers is dedicated to the functions of the CPU core:

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

Figure 9 : CPU block diagram



4.1 - Instruction Pipelines

The instruction pipeline breaks down CPU processing into the four following stages:

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

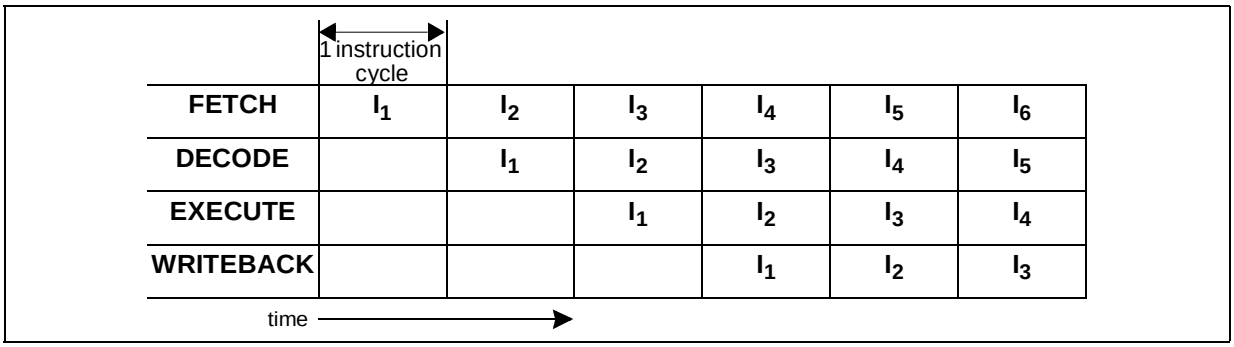
– Decode: Instructions are decoded and, if required, the operand addresses are calculated and the respective operands are fetched.
For all instructions, which implicitly access the system stack, the SP register is either decremented or incremented, as specified.
For branch instructions the Instruction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

– 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, too.

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

Injected instructions are generated internally by the machine to provide extra time for instructions that require more than one instruction cycle. Instructions are automatically injected into the decode stage of the pipeline, they pass through the remaining stages like every standard instruction. Program interrupts are performed by the same method of injecting instructions.

Figure 10 : Sequential instruction pipelining



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 instruction cycle, any isolated instruction takes at least four instruction cycles to be completed. Pipelining, however, allows parallel (simultaneous) processing of up to four instructions. Therefore, as soon as the pipeline has been filled, most instructions appear to be processed during one instruction cycle (see Figure 10).

Specification of instruction execution time always refers to the average execution time for pipelined parallel instruction processing (see Figure 10).

4.1.2 - Standard Branch Instruction Processing

When a branch is taken, it is necessary to perform the branched target instruction, before the current instruction in the pipeline. Therefore, at least one additional instruction cycle is required to fetch the branch target instruction.

This extra instruction cycle is provided by means of an injected instruction (see Figure 11). If a conditional branch is not taken, there is no deviation from the sequential program flow, and thus no extra time is required. In this case the instruction after the branch instruction will enter the decode stage of the pipeline at the beginning of the next instruction cycle after decode of the conditional branch instruction.

4.1.3 - Cache Jump Instruction Processing

The ST10F280 incorporates a jump cache. This minimizes the time taken for conditional jumps which are repeatedly processed in a loop. Whenever a cache jump instruction passes through the decode stage of the pipeline for the first time (provided that the jump condition is met), the jump target instruction is fetched as usual, causing a time delay of one instruction cycle.

If the instruction is repeated in a loop, the target instruction (JMPA, JMPR, JB, JBC, JNB, JNBS) is additionally stored in the cache. For execution of the repeated cache jump instruction, the jump target instruction is not fetched from program memory but taken from the cache and immediately injected into the decode stage of the pipeline (see Figure 12).

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

Figure 11 : Standard branch instruction pipelining

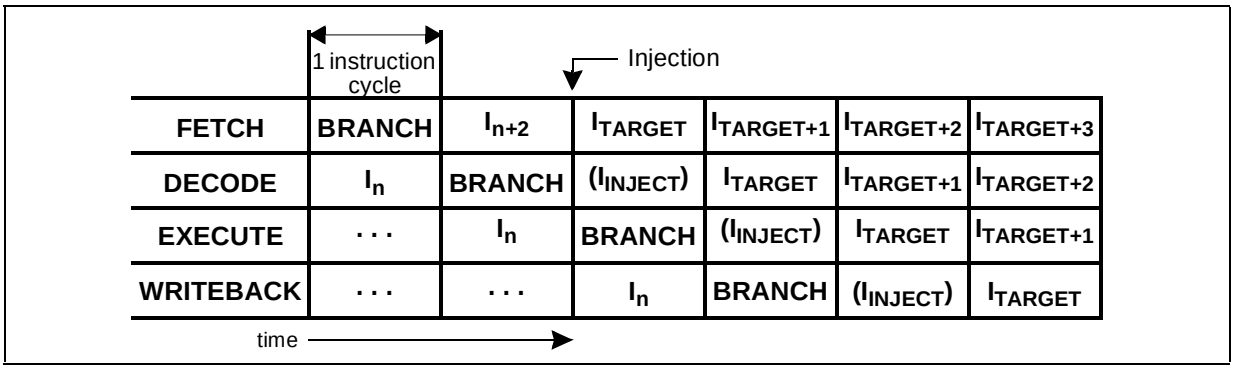
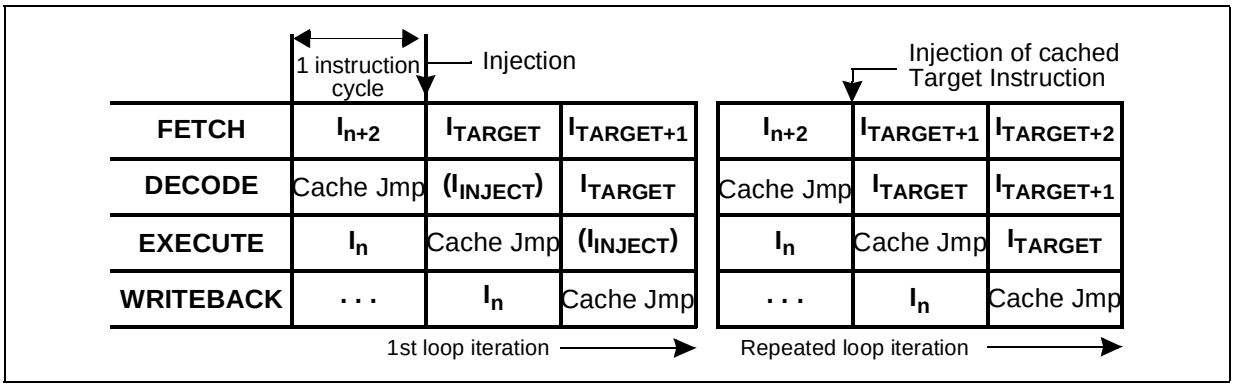


Figure 12 : Cache jump instruction pipelining



4.1.4 - Particular Pipeline Effects

Since up to four different instructions are processed simultaneously, additional hardware has been included in the ST10F280 to take into account dependencies between instructions in different stages of the pipeline.

This extra hardware like a forwarding operand read and write values, resolves most of the possible conflicts like multiple usage of buses.

This prevents delays that would cause the pipeline to become noticeable to the user. However, there are some cases where allowances must be made by the programmer, for the pipeline architecture of the ST10F280.

In these cases the delays caused by pipeline conflicts can be used for other instructions in order to optimize performance.

Context Pointer Updating

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

```
In          : MOV   DPP0, #4          ; select data page 4 via DPP0

In+1        : ....                    ; must not be an instr using DPP0

In+2        : MOV   DPP0:0000H, R1    ; move contents of R1 to address loc
                                      ; 01'0000h

                                      ; (in dp 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, in order to use the new SP register value without erroneously performed stack accesses, at least one instruction must be inserted between an explicit SP writing and any subsequent of the just mentioned implicitly SP using instructions, as shown in the 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

The effect described here will only become noticeable, when watching the external memory access sequences on the external bus by means of a Logic Analyzer. Different pipeline stages can simultaneously put a request on the External Bus Controller (EBC).

The sequence of instructions processed by the CPU may diverge from the sequence of the corresponding external memory accesses performed by the EBC, due to the predefined priority of external memory accesses.

```
1st         Write Data

2nd         Fetch Code

3rd         Read Data.
```

Controlling Interrupts

Software modifications (implicit or explicit) of the PSW are done in the execute phase of the respective instructions. In order to maintain fast interrupt responses, however, the current interrupt prioritization round does not consider these changes. For example an interrupt request may be acknowledged after the instruction that disables interrupts via IEN or ILVL or after the following instructions.

Time critical instruction sequences, therefore, should not begin directly after the instruction disabling interrupts, as shown in the example.

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

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

Initialization of Port Pins

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

```
WRONG:       BSET DP3.13       ; change direction of P3.13 to output
             BSET P3.5         ; P3.13 is still input, the read-modify-write
                               ; reads pin P3.13
RIGHT:       BSET DP3.13       ; change direction of P3.13 to output
             NOP               ; any instruction not accessing  Port3
             BSET P3.5         ; P3.13 is now output,
                               ; the read-modify-write reads the P3.13 output
                               ; latch
```

Changing the System Configuration

The instruction following an instruction that changes the system configuration via register SYSCON (like the mapping of the internal memory, like segmentation like stack size), cannot use the new resources (Memory or stack). This instruction must not access the new ressources.

Code accesses to the new Memory area are only possible after an absolute branch to this area. As a rule, instructions that change Memory mapping must be executed from internal RAM or external memory.

BUSCON/ADDRSEL

The (I_{n+1}) instruction following an (I_n) instruction that changes the properties of an external address area, cannot access operands within the new area.

This instruction (I_{n+1}) must not access this memory area. Code accesses to the new address area must be made after an absolute branch to this area.

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

Timing

Pipeline architecture drastically reduces the average instruction processing time. The mean ratio is about four to one instruction cycle. Some peculiar cases of pipeline configuration extend the instruction processing time by half or by one cycle.

These cases have to be taken in account for the time critical software routines. Besides a general execution time description, the following section provides some hints on how to optimize time-critical program parts with regard to such pipeline-caused timing particularities.

4.2 - Bit-handling and bit-protection

The ST10F280 provides several mechanisms for bit manipulation. These mechanisms, either handle software flags within the internal RAM, control on-chip peripherals via control bit in their respective SFRs, or control I/O functions via port pins.

The instructions BSET, BCLR, BAND, BOR, BXOR, BMOV and BMOVN, explicitly set or clear specific bit. The instructions BFLDL and BFLDH make it possible to change up to 8-bit of a specific byte at a time.

The instructions JBC and JNBS implicitly clear or set the specified bit when the jump is taken. The instructions JB and JNB (also conditional jump instructions that refer to flags) evaluate the specified bit to determine if the jump is to be taken.

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

All instructions that change single bit or bit groups internally use a read-modify-write sequence that accesses the whole word containing the specified bit(s). This method has several consequences:

– Bit can only be modified within the internal specific address areas (IRAM, SFRs...). External locations cannot be used with bit instructions.

– The upper 256 bytes of the SFR area, the ESFR area and the internal RAM are bit-addressable (see Chapter 3 - Memory Organization). Those register bits located within the respective sections can be directly manipulated using bit instructions. The other SFRs must be accessed byte or word wise.

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

– The read-modify-write approach may be critical with hardware-effected bit. In these cases the hardware may change specific bit while the read-modify-write operation is in progress, where the writeback would overwrite the new bit value generated by the hardware. The solution is either the implemented hardware protection (see below) or realized through special programming (see Section 4.1.4 - Particular Pipeline Effects).

Protected bit: As mentioned in Section 2.6 - Protected bits (hardware set) are not modified during a read-modify-write sequence, even if an interrupt request rises between read and write time. The hardware protection logic guarantees that only the intended bit(s) is/are effected by the write-back operation.

Note If a conflict occurs between a bit manipulation generated by hardware and an intended software access the software access has priority and determines the final value of the respective bit (See Section 2.6 - Protected bits).

4.3 - Instruction Execution Times

Instruction execution time depends on where the instruction is fetched from and where operands are read from or written to. When a program is fetched from internal memory, most of the instructions can be processed in one instruction cycle. All external memory accesses are performed by the on-chip External Bus Controller (EBC) which works in parallel with the CPU. This section summarizes the execution times. A detailed description of the execution times for the various instructions and the specific exceptions can be found in the **"ST10 Family Programming Manual"**. Table 6 shows the minimum execution times required to process a ST10F280 instruction fetched from the internal Flash, the internal RAM, or from external memory. The values are in CPU clock cycles and assume no wait-states. Two CPU clock cycles are equal to one instruction cycle.

These execution times apply to most of the ST10F280 instructions except some of the branches, the multiplication, the division and a special move instruction. In case of internal Memory program execution, there is no execution time dependency on the instruction length, except for some special branch situations. Because of the short execution time, execution from internal RAM is flexible for loadable and modifiable code. Execution from external memory depends on the selected bus mode and the programming of the bus cycles (wait-states). The operand and instruction accesses listed below can extend the execution time of an instruction:

– Internal Flash Memory operand reads (same for byte and word operand reads),

– Internal RAM operand reads via indirect addressing modes,

– Internal SFR operand reads immediately after writing,

– External operand reads,

– External operand writes,

– Jumps to non-aligned double word instructions in the internal Flash Memory space,

– Testing Branch Conditions immediately after PSW writes.

4.4 - CPU Special Function Registers

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

The access mechanism for these SFRs in the CPU core is identical to the access mechanism for any other SFR. Since all SFRs can be controlled by means of any instruction which is able to address the SFR memory space, a lot of flexibility has been gained without creating 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.

Notes : 1. Note that any explicit write request (via software) to an SFR supersedes a simultaneous modification of the same register, by hardware.

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

Table 6 : Minimum execution times

Memory Area	Instruction Fetch	
	Word Instruction (CPU clock cycles)	Doubleword Instruction (CPU clock cycles)
Internal Memory	2	2
Internal RAM	6	8
16-bit De-multiplex Bus	2	4
16-bit Multiplexed Bus	3	6
8-bit De-multiplex Bus	4	8
8-bit Multiplexed Bus	6	12

4.4.1 - The System Configuration Register SYSCON

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

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	XPEN	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
VISIBLE	**Visible Mode Control** '0': Accesses to XBUS peripherals are done internally '1': XBUS peripheral accesses are made visible on the external pins
XPEN	**XBUS Peripheral Enable bit** '0': Accesses to the on-chip X-peripherals and XRAM are disabled. '1': The on-chip X-peripherals are enabled.
BDRSTEN	**Bidirectional Reset Enable** '0': RSTIN pin is an input pin only. SW Reset or WDT Reset have no effect on this pin '1': RSTIN pin is a bidirectional pin. This pin is pulled low during 1024 TCL during reset 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 µs, the CPU clock is switched automatically to PLL's base frequency (from 2 to 10MHz). '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 power supply current.
PWDCFG	**Power Down Mode Configuration Control** '0': Power Down Mode can only be entered during PWRDN instruction execution if NMI pin is low, otherwise the instruction has no effect. To exit Power Down Mode, an external reset must occurs by asserting the 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 or with external reset.
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** (Inverted copy of WRC bit of RPOH) '0': Pins WR and BHE retain their normal function '1': Pin WR acts as WRL, pin BHE acts as WRH
CLKEN	**System Clock Output Enable** (CLKOUT) '0': CLKOUT disabled, pin may be used for general purpose I/O '1': CLKOUT enabled, pin outputs the system clock signal
BYTDIS	**Disable/Enable Control for Pin BHE** (Set according to data bus width) '0': Pin BHE enabled '1': Pin BHE disabled, pin may be used for general purpose I/O
ROMEN	**Internal Memory Enable** (Set according to pin EA during reset) '0': Internal memory disabled: accesses to the Flash Memory area use the external bus '1': Internal memory enabled

Bit	Function
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 Memory Mapping** '0': Internal memory area mapped to segment 0 (00'0000h...00'7FFFh) '1': Internal memory area mapped to segment 1 (01'0000h...01'7FFFh)
STKSZ	**System Stack Size** Selects the size of the system stack (in the internal RAM) from 32 to 1024 words

Note Register SYSCON cannot be changed after execution of the EINIT instruction.
The function of bit XPER-SHARE, VISIBLE, WRCFG, BYTDIS, ROMEN and ROMS1 is described in more detail in Section 9.4 - Controlling the External Bus Controller.

System Clock Output Enable (CLKEN)

The system clock output function is enabled by setting bit CLKEN in register SYSCON to '1'. If enabled, port pin P3.15 takes on its alternate function as CLKOUT output pin. The clock output is a 50 % duty cycle clock whose frequency equals the CPU operating frequency ($f_{OUT} = f_{CPU}$).

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

Segmentation Disable/enable Control (**SGTDIS)**

Bit SGTDIS allows to select either the segmented or non-segmented memory mode.

In non-segmented memory mode (SGTDIS='1') it is assumed that the code address space is restricted to 64 Kbytes (segment 0) and thus 16-bit are sufficient to represent all code addresses.

For implicit stack operations (CALL or RET) the CSP register is totally ignored and only the IP is saved to and restored from the stack.

In segmented memory mode (SGTDIS='0') it is assumed that the whole address space is available for instructions. For implicit stack operations (CALL or RET) the CSP register and the IP are saved to and restored from the stack. After reset the segmented memory mode is selected.

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

System Stack Size (STKSZ)

This bit-field defines the size of the physical system stack, which is located in the internal RAM of the ST10F280. An area of 32...1024 words or all of the internal RAM may be dedicated to the system stack. A so-called "circular stack" mechanism allows to use a bigger virtual stack than this dedicated RAM area. These techniques as well as the encoding of bit-field STKSZ are described in more detail in Stack Operations (see Section 22.1 - Stack Operations).

4.4.2 - X Peripherals Control Register (XPERCON)

XPERCON (F024h / 12h) ESFR Reset Value: - - 05h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	XPWMEN	XPERCONEN3	XRAMEN	CAN2EN	CAN1EN
											RW	RW	RW	RW	RW

Bit	Function
CAN1EN	**CAN1 Enable Bit** 0 Accesses to the on-chip CAN1 XPeripheral and its functions are disabled. P4.5 and P4.6 pins can be used as general purpose I/Os. Address range 00'EF00h-00'EFFFh is only directed to external memory if CAN2EN and XPWM bits are cleared also. 1 The on-chip CAN1 XPeripheral is enabled and can be accessed.
CAN2EN	**CAN2 Enable Bit** 0 Accesses to the on-chip CAN2 XPeripheral and its functions are disabled. P4.4 and P4.7 pins can be used as general purpose I/Os. Address range 00'EE00h-00'EEFFh is only directed to external memory if CAN1EN and XPWM bits are cleared also. 1 The on-chip CAN2 XPeripheral is enabled and can be accessed.
XRAMEN	**XRAM Enable Bit** 0 Accesses to the on-chip 16K Byte XRAM are disabled, external access performed. 1 The on-chip 16K Byte XRAM is enabled and can be accessed.
XPERCONEN3	**XPORT9, XTIMER, XPORT10, XADCMUX Enable Bit** 0 Accesses to the XPORT9, XTIMER, XPORT10, XADCMUX peripherals are disabled, external access performed. 1 The on-chip XPORT9, XTIMER, XPORT10, XADCMUX peripherals are enabled and can be accessed.
XPWMEN	**XPWM Enable Bit** 0 Accesses to the on-chip XPWM are disabled, external access performed. Address range 00'EC00h-00'ECFFh is only directed to external memory if CAN1EN and CAN2EN are '0' also 1 The on-chip XPWM is enabled and can be accessed.

Note: - When both CAN and XPWM are disabled via XPERCON setting, then any access in the address range 00'EC00h 00'EFFFh will be directed to external memory interface, using the BUSCONx register corresponding to address matching ADDRSELx register. P4.4 and P4.7 can be used as General Purpose I/O when CAN2 is not enabled, and P4.5 and P4.6 can be used as General Purpose I/O when CAN1 is not enabled.

- The default XPER selection after Reset is : XCAN1 is enabled, XCAN2 is disabled, XRAM is enabled, XPORT9, XTIMER, XPORT10, XPWM, XADCMUX are disabled.

- Register XPERCON cannot be changed after the global enabling of XPeripherals, i.e. after setting of bit XPEN in SYSCON register.

In EMUlation mode, all the XPERipherals are enabled (XPERCON bit are all set).

4.4.3 - The Processor Status word PSW

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

PSW (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.
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 Chapter 6 - Interrupt and Trap Functions)

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

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

After execution of an instruction which explicitly updates the PSW register, the condition flags cannot be interpreted as described in the following, because any explicit write to the PSW register supersedes the condition flag values, which are implicitly generated by the CPU.

Explicitly reading the PSW register supplies a read value which represents the state of the PSW register after execution of the immediately preceding instruction.

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

N-Flag: For most of the ALU operations, the N-flag is set to '1', if the most significant bit of the result contains a '1', otherwise it is cleared. In the case of integer operations the N-flag can be interpreted as the sign bit of the result (negative: N='1', positive: N='0'). Negative numbers are always represented as the 2's complement of the corresponding positive number. The range of signed numbers extends from '–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 XOR of the two specified bit.

C-Flag: After an addition the C-flag indicates that a carry from the most significant bit of the specified word or byte data type has been generated. After a subtraction or a comparison the C-flag indicates a borrow, which represents the logical negation of a carry for the addition.

This means that the C-flag is set to '1', if **no** carry from the most significant bit of the specified word or byte data type has been generated during a subtraction, which is performed internally by the ALU as a 2's complement addition, and the C-flag is cleared when this complement addition caused a carry. The C-flag is always cleared for logical, multiply and divide ALU operations, because these operations cannot cause a carry anyhow.

For shift and rotate operations the C-flag represents the value of the bit shifted out last. If a shift count of zero is specified, the C-flag will be cleared. The C-flag is also cleared for a prioritize ALU operation, because a '1' is never shifted out of the MSB during the normalization of an operand. For Boolean bit operations with only one operand the C-flag is always cleared. For Boolean bit operations with two operands the C-flag represents the logical ANDing of the two specified bit.

V-Flag: For addition, subtraction and 2's complementation the V-flag is always set to '1', if the result overflows the maximum range of signed numbers, which are representable by either 16-bit for word operations ('–8000h' to '+7FFFh'), or by 8-bit for byte operations ('–80h' to '+7Fh'), otherwise the V-flag is cleared. 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. A division by zero will always cause an overflow. In contrast to the result of a division, the result of a multiplication is valid regardless of whether the V-flag is set to '1' or not. Since logical ALU operations cannot produce an invalid result, the V-flag is cleared by these operations.

The V-flag is also used as 'Sticky bit' for rotate right and shift right operations. With only using the C-flag, a rounding error caused by a shift right operation can be estimated up to a quantity of one half of the LSB of the result. In conjunction with the V-flag, the C-flag allows evaluating the rounding error with a finer resolution (see Figure 13). 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 bit.

Figure 13 : Shift right rounding error evaluation

C-Flag	V-Flag	Rounding Error Quantity
0	0	No rounding error
0	1	0 < Rounding error< $^1/_2$ LSB
1	0	Rounding error= $^1/_2$ LSB
1	1	Rounding error> $^1/_2$ LSB

Z-Flag: The Z-flag is normally set to '1', if the result of an ALU operation equals zero, otherwise it is cleared. For the addition and subtraction with carry the Z-flag is only set to '1', if the Z-flag already contains a '1' and the result of the current ALU operation additionally equals zero. This mechanism is provided for the support of multiple precision calculations.

For Boolean bit operations with only one operand the Z-flag represents the logical negation of the previous state of the specified bit. For Boolean bit operations with two operands the Z-flag represents the logical NORing of the two specified bit. For the prioritize ALU operation the Z-flag indicates, if the second operand was zero or not.

E-Flag: The E-flag can be altered by instructions, which perform ALU or data movement operations. The E-flag is cleared by those instructions which cannot be reasonably used for table search operations. In all other cases the E-flag is set depending on the value of the source operand to signify whether the end of a search table is reached or not.

If the value of the source operand of an instruction equals the lowest negative number, which is representable by the data format of the corresponding instruction ('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 will be set to '1' by hardware upon the entrance into an interrupt service routine, when a multiply or divide ALU operation was interrupted before completion. Depending on the state of the MULIP bit, the hardware decides whether a multiplication or division must be continued or not after the end of an interrupt service. The MULIP bit is overwritten with the contents of the stacked MULIP-flag when the return-from-interrupt-instruction (RETI) is executed. This normally means that the MULIP-flag is cleared again after that.

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

CPU Interrupt Status (IEN, ILVL)

The Interrupt Enable bit allows to globally enable (IEN='1') or disable (IEN='0') interrupts. The four bit Interrupt Level field (ILVL) specifies the priority of the current CPU activity.

The interrupt level is updated by hardware upon entry into an interrupt service routine, but it can also be modified via software to prevent other interrupts from being acknowledged. In case an interrupt level '15' has been assigned to the CPU, it has the highest possible priority, and thus the current CPU operation cannot be interrupted except by hardware traps or external non-maskable interrupts. For details please refer to Chapter 6 - Interrupt and Trap Functions.

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

4.4.4 - The Instruction Pointer IP

This register determines the 16-bit intra-segment address of the currently fetched instruction within the code segment selected by the CSP register.

The IP register is not mapped into the MCU address space, and thus it is not directly accessible by the programmer. The IP can, however, be modified indirectly via the stack by means of a return instruction.

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

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IP															
(R)(W)															

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

4.4.5 - The Code Segment Pointer CSP

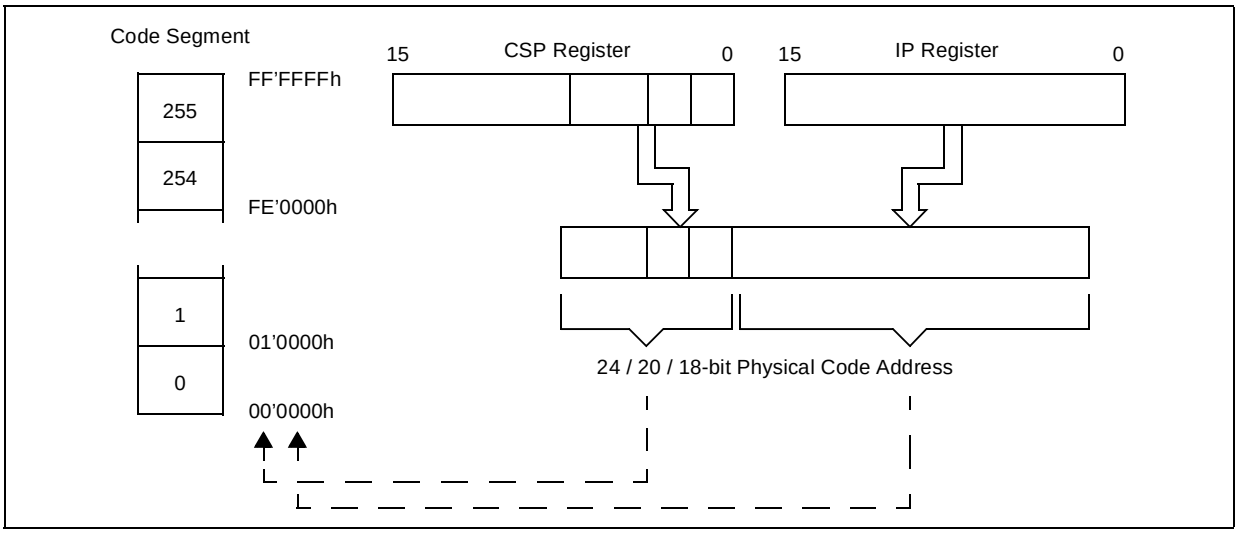
This non bit addressable register selects the code segment being used at run-time to access instructions. The lower 8 bits of register CSP select one of up to 256 segments of 64 Kbytes each, while the upper 8 bits are reserved for future use.

CSP (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 14 : Addressing Via the Code Segment Pointer



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

Code memory addresses are generated by directly extending the 16-bit contents of the IP register by the contents of the CSP register as shown in the Figure 14.

In case of the segmented memory mode the selected number of segment address bit (7...0, 3...0 or 1...0) of register CSP is output on the segment address pins A23...A16 of Port4 for all external code accesses. For non-segmented memory mode the content of this register is not significant, because all code accesses are automatically restricted to segment 0.

The CSP register can only be read but not written by data operations. It is, however, modified either directly by means of the JMPS and CALLS instructions, or indirectly via the stack by means of the RETS and RETI instructions.

Upon the acceptance of an interrupt or the execution of a software TRAP instruction, the CSP register is automatically set to zero.

4.4.6 - The Data Page Pointers DPP0, DPP1, DPP2, DPP3

These four non bit addressable registers select up to four different data pages being active simultaneously at run-time. The lower 10 bits of each DPP register select one of the 1024 possible 16 Kbyte data pages while the upper 6 bits are reserved for future use. The DPP registers make it possible to access the entire memory space, in pages of 16 Kbytes each.

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

DPP0 (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									

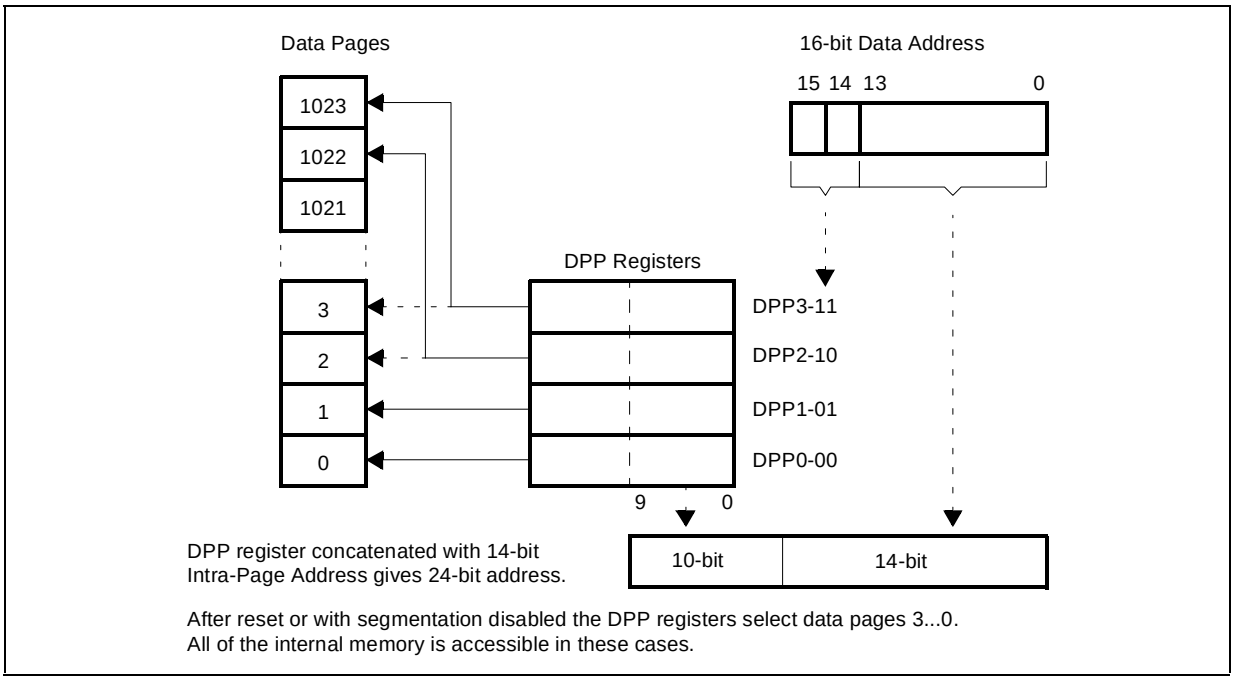
Bit	Function
DPPxPN	**Data Page Number of DPPx:** Specifies the data page selected via DPPx. Only the 2 least significant bits of DPPx are used when segmentation is disabled.

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 content of the selected DPP register specifies one of the 1024 possible data pages. This data page base address together with the 14-bit page offset forms the physical 24-/20-/18-bit address. In case of non-segmented memory mode, only the two least significant bit of the implicitly selected DPP register are used to generate the physical address. Thus, extreme care should be taken when changing the content of a DPP register, if a non-segmented memory model is selected, because otherwise unexpected results could occur.

In case of the segmented memory mode the selected number of segment address bit (9...2, 5...2 or 3...2) of the respective DPP register is output on the segment address pins A23/A19/A17/A16 of Port4 for all external data accesses. A DPP register can be updated via any instruction, which is capable of modifying an SFR.

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

Figure 15 : Addressing via the data page pointers



4.4.7 - The Context Pointer CP

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

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 bit CP.11...CP.9 = '000', bit CP.11...CP.10 are set to '11' by hardware, in all other cases all bit of bit-field "cp" receive the written value.

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

– Do not set CP below the IRAM start address, 00'F600h (2 Kbytes).

– Do not set CP above 00'FDFEh.

– Be careful using the upper GPRs with CP above 00'FDE0h.

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

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

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

Several addressing modes use register CP implicitly 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, which is the base of the current register bank.

Depending on whether a relative word (Rw) or byte (Rb) GPR address is specified, the short 4-bit GPR address is either multiplied by two or not before it is added to the content of register CP (see Figure 17).

Thus, both byte and word GPR accesses are possible in this way. GPRs used as indirect address pointers are always accessed word wise.

For some instructions only the first four GPRs can be used as indirect address pointers.

These GPRs are specified via short 2-bit GPR addresses. The respective physical address calculation is identical to that for the short 4-bit GPR addresses.

Short 8-bit register addresses (mnemonic: reg or bitoff) within a range from F0h to FFh interpret the four least significant bit as short 4-bit GPR address, while the four most significant bit are ignored.

The respective physical GPR address calculation is identical to that for the short 4-bit GPR addresses. For single bit accesses on a GPR, the GPR's word address is calculated as just described, but the position of the bit within the word is specified by a separate additional 4-bit value.

Figure 16 : Register bank selection via register CP

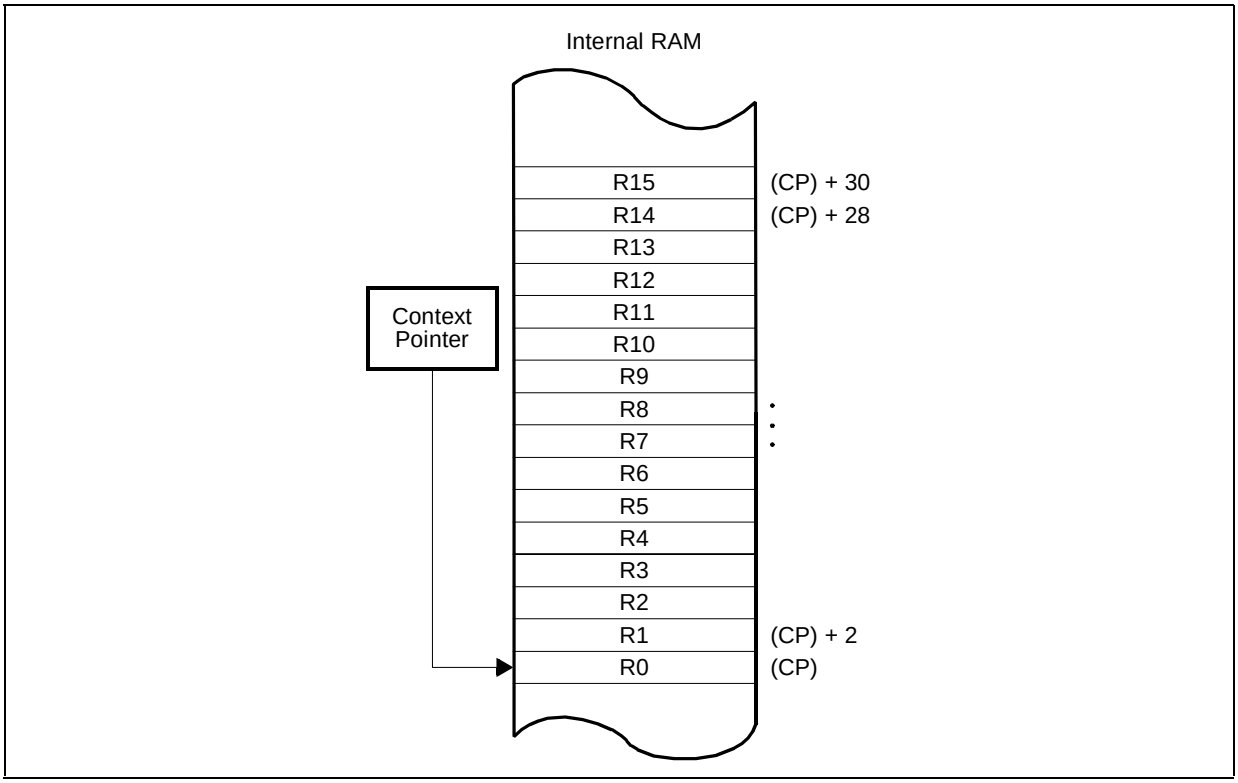
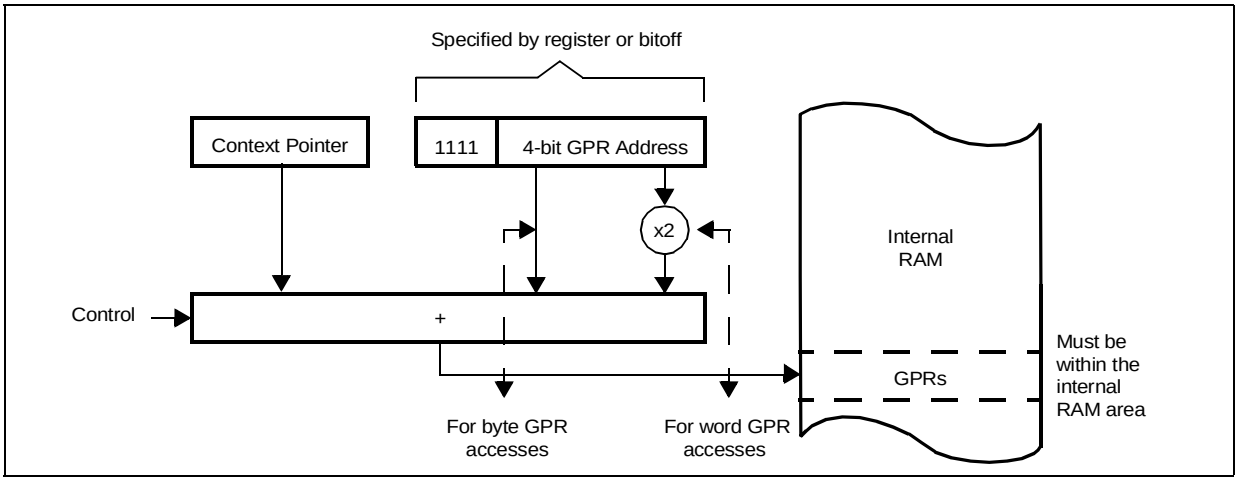


Figure 17 : Implicit CP use by short GPR addressing modes



4.4.8 - The Stack Pointer SP

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

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

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

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

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

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

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

4.4.9 - 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 (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 bit 15 through 12 are tied to '1' by hardware, the STKOV register can only contain values from F000h to FFFEh.

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

Fatal error indication treats the stack overflow as a system error through the associated trap service routine. Under these circumstances data in the bottom of the stack may have been overwritten by the status information stacked upon servicing the stack overflow trap.

Automatic system stack flushing allows to use the system stack as a 'Stack Cache' for a bigger external user stack. In this case register STKOV should be initialized to a value, which represents the desired lowest Top of Stack address plus 12 according to the selected maximum stack size. This considers the worst case that will occur, when a stack overflow condition is detected just during entry into an interrupt service routine. Then, six additional stack word locations are required to push IP, PSW, and CSP for both the interrupt service routine and the hardware trap service routine.

More details about the stack overflow trap service routine and virtual stack management are given in Chapter 22 - System Programming.

4.4.10 - The Stack Underflow Pointer STKUN

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

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

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

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

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

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

– **Fatal error indication** treats the stack underflow as a system error through the associated trap service routine.

– **Automatic system stack refilling** allows to use the system stack as a 'Stack Cache' for a bigger external user stack. In this case register STKUN should be initialized to a value, which represents the desired highest Bottom of Stack address.

More details about the stack underflow trap service routine and virtual stack management are given in Chapter 22 - System Programming.

Scope of stack limit control

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

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

– The stack pointer SP is directly updated via MOV instructions.

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

4.4.11 - The Multiply / Divide High Register MDH

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

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

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

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

Whenever this register is updated via software, the Multiply/Divide Register In Use (MDRIU) flag in the Multiply/Divide Control register (MDC) is set to '1'. When a multiplication or division is interrupted before its completion and when a new multiply or divide operation is to be performed within the interrupt service routine, register MDH must be saved along with registers MDL and MDC to avoid erroneous results.

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

4.4.12 - The Multiply / Divide Low Register MDL

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

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

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

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

Whenever this register is updated via software, the Multiply/Divide Register In Use (MDRIU) flag in the Multiply/Divide Control register (MDC) is set to '1'. The MDRIU flag is cleared, whenever the MDL register is read via software. When a multiplication or division is interrupted before its completion and when a new multiply or divide operation is to be performed within the interrupt service routine, register MDL must be saved along with registers MDH and MDC to avoid erroneous results.

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

4.4.13 - The Multiply / Divide Control Register MDC

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

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	**MS**	**MS**	**MS**	**MDRIU**	**MS**	**MS**	**MS**	**MS**
								RW	RW	RW	RW	RW	RW	RW	RW

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.
MS	**Internal Machine Status** The multiply/divide unit uses these bit to control internal operations. Never modify these bit without saving and restoring register MDC.

When a division or multiplication was interrupted before its completion and the multiply/divide unit is required, the MDC register must first be saved along with registers MDH and MDL (to be able to restart the interrupted operation later), and then it must be cleared prepare it for the new calculation. After completion of the new division or multiplication, the state of the interrupted multiply or divide operation must be restored. The MDRIU flag is the only portion of the MDC register which might be of interest for the user. The remaining portions of the MDC register are reserved for dedicated use by the hardware, and should never be modified by the user in another way than described above. Otherwise, a correct continuation of an interrupted multiply or divide operation cannot be guaranteed.

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

4.4.14 - The Constant Zeros Register ZEROS

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

4.4.15 - The Constant Ones Register ONES

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

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

4.4.16 - Example

Mask for FFFFh values use to increment or decrement memory

```
sub     mem, ones   ;mem=mem+1
                    ;increments the memory location in one
                    ;instruction instead of three, as described
                    ;below
mov     [R13], mem  ;mem -->R13
add     [R13], #1   ;R13 + 1;
mov     mem, [R13]  ;R13 --> mem
```

5 - MULTIPLY-ACCUMULATE UNIT (MAC)

The MAC is a specialized co-processor added to the ST10F280 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'.

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

– One instruction cycle execution for all MAC operations (2 CPU cycles).

– 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.

The working register of the MAC Unit is a dedicated 40-bit wide Accumulator register. A set of consistent flags is automatically updated in the MSW register (see Section 5.3.2 - Accumulator & Control Registers) after each MAC operation. These flags allow branching on specific conditions. Unlike the PSW flags, these flags are not preserved automatically by the CPU upon entry into an interrupt or trap routine. **All dedicated MAC registers must be saved on the stack if the MAC unit is shared between different tasks and interrupts**.

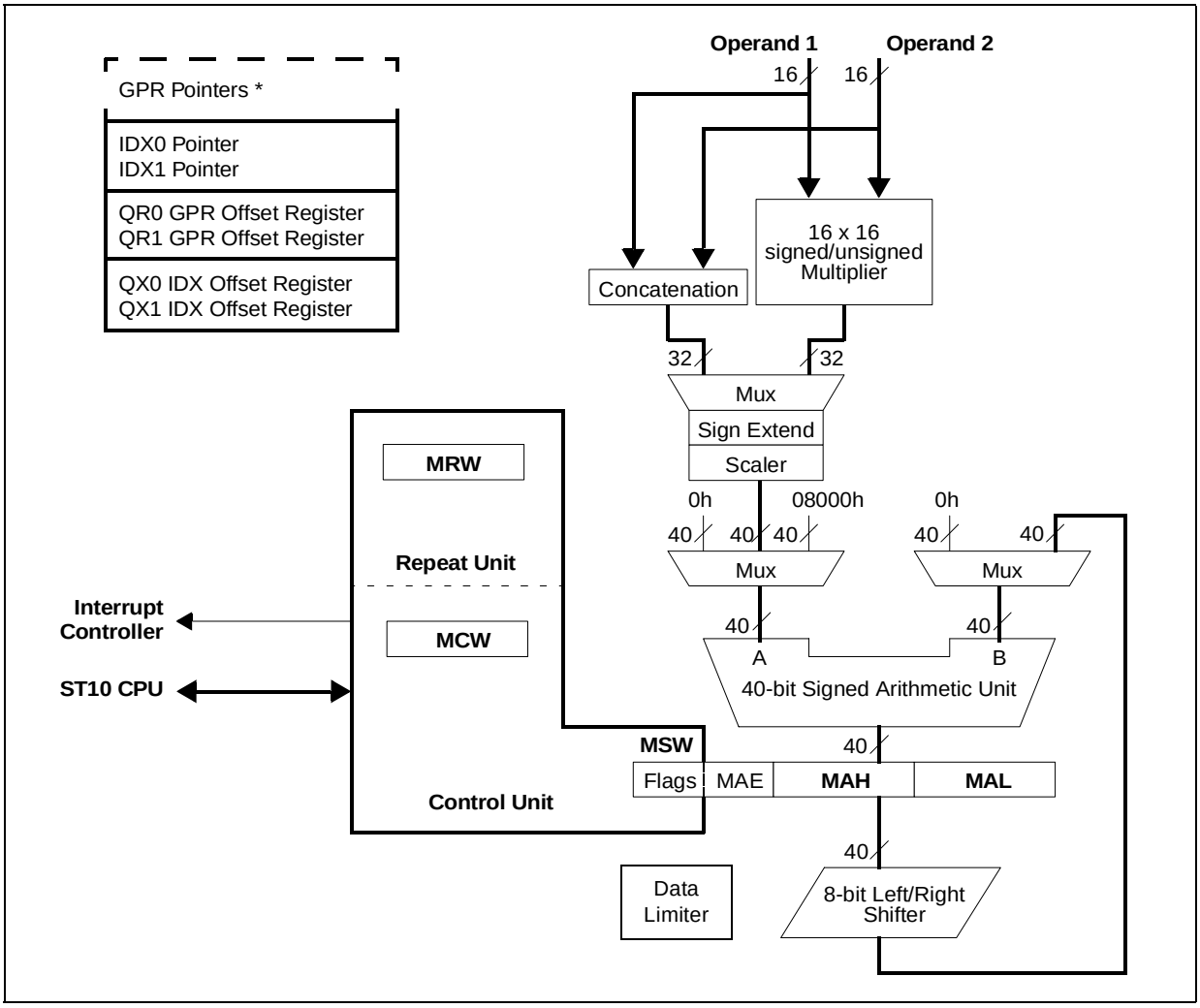
Program control

– 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 18 : MAC architecture



** Shared with standard ALU.*

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.

5.2.2 - Particular Pipeline Effects with the MAC Unit

Because the registers used by the MAC are shared with the standard ALU and because of the MAC instructions pipelining, some care must be taken when switching from the "standard instruction set" to the "MAC instruction set".

Initialization of the Pointers and Offset Registers

The new MAC instructions which use IDXi pointer is mostly not capable of using a new IDXi register value, which is to be updated by an immediately preceding instruction. Thus, to make sure that the new IDXi register value is used, at least one instruction must be inserted between a IDXi-changing instruction and one MAC instruction which explicitly uses IDXi in its addressing mode as shown in the following example:

```
I_n:   MOV IDX0, #0F200h          ; update IDX0 register
I_n+1: ...                        ; must not be an CoXXX [IDX0⊗], [Rw_m⊗] instruction
I_n+2: CoXXX [IDX0+QX1], [R2]     ; first operand read at (IDX0) address to provide the MAC function
                                  ; parallel data move to (((IDX0))-((QX1))) address (if CoXXX is CoMACM)
                                  ; move (R2) content to (IDX0) address (if CoXXX is CoMOV)
                                  ; (IDX0) <-- (IDX0) + (QX1) post modification of the pointer
```

Same requirements between the update of one of the offset reg. QXi & QRi and their next use.

Read Access to MAC registers (CoReg)

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.

Example 1

Code		MSW (before)	MSW (after)	Comment
MOV	MSW, #0	-	0000h	
MOV	R0, #0	-	-	
CoADD	R0, R0	0000h	0200h	MSW.Z set at execute
BFLDL	MSW, #FFh, #FFh	0200h	00FFh	Error !

In this example, the BFLDL instruction performs a read access to the MSW during the decode stage while the MSW.Z flag is only set at the end of the execute stage of the CoADD.

5.2.3 - 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⊗]" and "[IDX_i⊗]" refer to these addressing modes.

Table 7 : Pointer post-modification combinations for IDXi and Rwn

Symbol	Mnemonic	Address Pointer Operation
"[IDX_i⊗]" 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)
"[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)

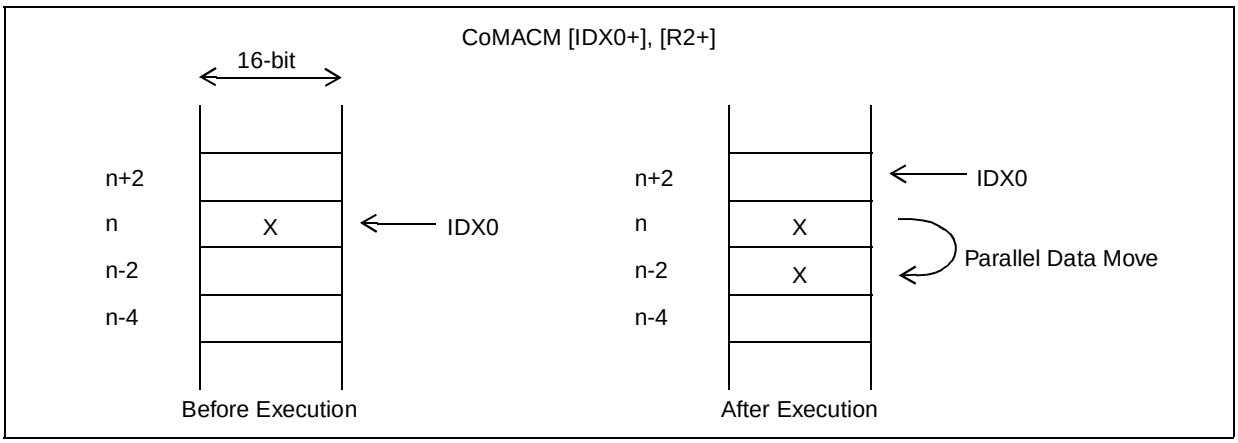
For the CoMACM class of instruction, Parallel Data Move mechanism is implemented. This class of instruction is only available with double indirect addressing mode. Parallel Data Move allows the operand pointed by IDX_i to be moved to a new location in parallel with the MAC operation. The write-back address 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.

Table 8 : Parallel data move addressing

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>

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 19 : Example of parallel data move



5.2.4 - 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.

5.2.5 - 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.

5.2.6 - The 40-bit Adder/Subtracter

The 40-bit Adder/Subtracter allows intermediate overflows in a series of multiply/accumulate operations. The Adder/Subtracter has two input ports. One input is the feedback of the 40-bit Signed Accumulator output through the ACCU-Shifter.The second input is the 32-bit operand coming from the One-bit Scaler. The 32-bit operands are sign-extended to 40-bit before the addition/subtraction is performed.

The output of the Adder/Subtracter goes to the 40-bit Signed Accumulator. It is also possible to round and to saturate the result to 32-bit automatically after every accumulation before to be loaded into the accumulator. The round operation is performed by adding 00'00008000h to the result. Automatic saturation is enabled by setting the MCW.MS saturation bit.

When the 40-bit Signed Accumulator is in the overflow saturation mode and an overflow occurs, the accumulator is loaded with either the most positive or the most negative possible 32-bit value, depending on the direction of the overflow as well as the arithmetic used. The value of the accumulator upon saturation is 00'7FFF FFFFh (positive) or FF'8000'0000h (negative).

5.2.7 - 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.

Table 9 : Limiter output using CoSTORE instruction

ME-flag	MN-flag	MAS value (saturated MAH value) [3]
0	x	Unchanged [1]
1	0	7FFFh [2]
1	1	8000h [2]

Note: 1) When data limiter is disabled, a reading with "CoSTORE <destination>, <MAH> instruction" or "CoSTORE <destination>, <MAS> instruction" gives the same result.

2) If data limiter is activated, a read with "CoSTORE <destination>, <MAH> instruction" or "CoSTORE <destination>, <MAS> instruction" gives different results. MAS gives the saturated value of MAH. The reading of MAL and MSW (MAE) are not saturated.

5.2.8 - The Accumulator Shifter

The accumulator shifter is a parallel shifter with a 40-bit input and a 40-bit output. The source accumulator shifting operation are:

– No shift (Unmodified)

– Up to 8-bit Arithmetic Left Shift

– Up to 8-bit Arithmetic Right Shift

Notice that MSW.ME, MSW.MSV and MSW.MSL bits (See MSW register description) are affected by left shifts, therefore, if the saturation detection is enabled (MCW.MS bit is set), the behavior is similar to the one of the Adder/Subtracter.

Some precautions are required in case of left shift with enabled saturation. If MSW.MAE bit-field (most significant byte of the 40-bit Signed Accumulator) contains significant bits, then the 32-bit value in the accumulator is generally saturated. However, it is possible that a left shift may move out of the Accumulator some significant bits. The 40-bit result will be misinterpreted and will be either not saturated or saturated wrong. There is a chance that the result of a left shift may produce a result which can saturate an original positive number to the minimum negative value, or vice versa.

5.2.9 - 40-bit Signed Accumulator Register

The 40-bit Accumulator consists of three smaller registers, MAL, MAH, and MAE. MAH and MAL are 16-bit wide, MAE is 8-bit wide. MAE is the Most Significant Byte of the 40-bit accumulator, MAE is accessed as the Least Significant Byte of MSW and performs guarding function.

On MAH write the value of the accumulator is automatically adjusted to signed extended 40-bit format. That means MAE is automatically loaded by zeros for the positive number (MAH has 0 in the most significant bit). In case of negative number (MAH has 1 in the most significant bit) the MAE is loaded with ones, representing the extended 40-bit negative number in 2's complement notation. Then the extended 40-bit value is equal to 32-bit value without extension. In other words, after this extension MAE does not contain significant bits. Generally, this condition is present when the highest 9 bits of the 40-bit signed result are the same.

During the 40-bit accumulator operations the result may be greater than 32 bits and the MAE content changes. The MSW.ME extension flag is set because the signed result of the 40-bit accumulator has overflown the 32-bit boundary. This condition is right when the highest 9 bits of the 40-bit signed result are not the same, this means also that MAE contains significant bits.

Note: Most of the CoXXX operations specify the 40-bit accumulator register as a source and/or a destination operand. Operand loaded in 32-bits format is extended to 40-bit signed numbers with MAE equal to 00h (for positive numbers) or FFh (for negative numbers).

CoMOV and CoSTORE instructions can be used to store any part of the 40-bit accumulator (MAL, MAH, MSW-MAE) into other word locations.

Because writing to MAH forces zero value in MAL and sign extention in MAE, MAH must be written first and MAL in second. Some care must be taken in the order these registers are handled, for example in saving status stacking as shown in the following example:

```
PUSH MAL
PUSH MAH        ; Last one because later impact on MAE, MAL

POP MAH         ; First one because impact on MAE, MAL
POP MAL
```

5.2.10 - 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.

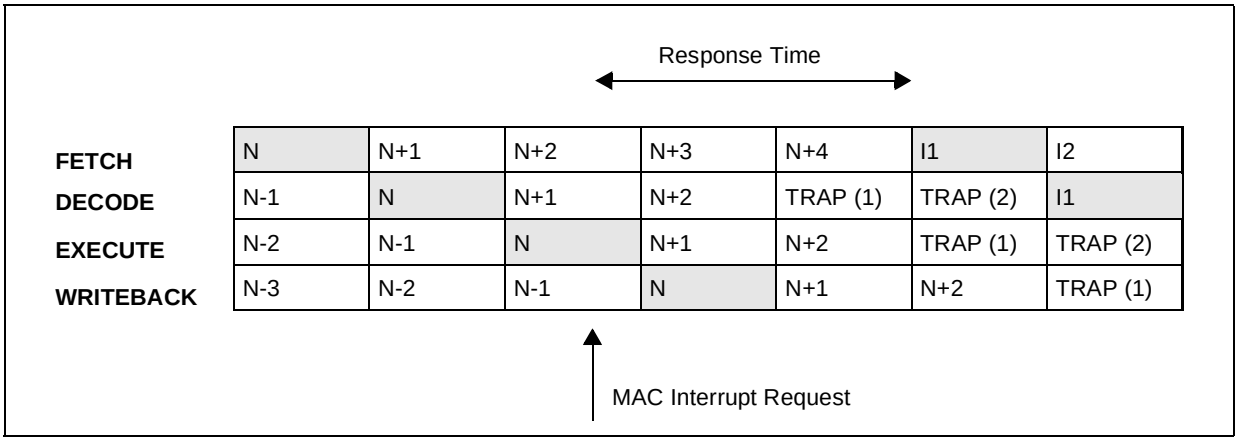
5.2.11 - MAC interrupt

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 20 : Pipeline diagram for MAC interrupt response time



5.2.12 - 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
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
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.

Table 10 : MAC register address in 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

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'.

Table 11 : MAC instruction set summary

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

Mnemonic	Addressing Modes	Rep
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

6 - INTERRUPT AND TRAP FUNCTIONS

The architecture of the ST10F280 supports several mechanisms for fast and flexible response to service requests that can be generated from various sources internal or external to the microcontroller. These mechanisms include:

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

– **Interrupt processing via the peripheral event controller (PEC)**: A faster alternative to normal software controlled interrupt processing is servicing an interrupt requesting device with the ST10F280's integrated Peripheral Event Controller (PEC). Triggered by an interrupt request, the PEC performs a single word or byte data transfer between any two locations in segment 0 (data pages 0 through 3) through one of eight programmable PEC Service Channels. During a PEC transfer the normal program execution of the CPU is halted for just 1 instruction cycle. No internal program status information needs to be saved. The same prioritization scheme is used for PEC service as for normal interrupt processing. PEC transfers share the 2 highest priority levels.

– **Trap functions**: Trap functions are activated in response to special conditions that occur during the execution of instructions. A trap can also be caused externally by the Non-Maskable Interrupt pin NMI. Several hardware trap functions are provided for handling erroneous conditions and exceptions that arise during the execution of an instruction. Hardware traps always have highest priority and cause immediate system reaction. The software trap function is invoked by the TRAP instruction, which generates a software interrupt for a specified interrupt vector. For all types of traps the current program status is saved on the system stack.

– **External interrupt processing**: Although the ST10F280 does not provide dedicated interrupt pins, it allows to connect external interrupt sources and provides several mechanisms to react on external events, including standard inputs, non-maskable interrupts and fast external interrupts. These interrupt functions are alternate port functions, except for the non-maskable interrupt and the reset input.

6.1 - Interrupt System Structure

The ST10F280 provides 56 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.

The control register contains the interrupt request flag, the interrupt enable bit, and the interrupt priority of the associated source. Each source request is activated by one specific event, depending on the selected operating mode of the respective device.

The only exceptions are the two serial channels of the ST10F280, 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 ST10F280 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 ST10F280's address space (segment 0).

The jump table is made up of the appropriate jump instructions that transfer control to the interrupt or trap service routines, which may be located anywhere within the address space.

The entries of the jump table are located at the lowest addresses in code segment 0 of the address space. Each entry occupies 2 words, except for the reset vector and the hardware trap vectors, which occupy 4 or 8 words.

The Table 12 lists all sources that are capable of requesting interrupt or PEC service in the ST10F280, the associated interrupt vectors, their locations and the associated trap numbers. It also lists the mnemonics of the affected Interrupt Request flags and their corresponding Interrupt Enable flags. The mnemonics are composed of a part that specifies the respective source, followed by a part that specifies their function (IR=Interrupt Request flag, IE=Interrupt Enable flag).

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

Table 12 : Interrupt and PEC service request sources

Source of Interrupt or PEC Service Request	Request Flag	Enable Flag	Interrupt Vector	Vector Location	Trap Number
CAPCOM Register 0	CC0IR	CC0IE	CC0INT	00'0040h	10h
CAPCOM Register 1	CC1IR	CC1IE	CC1INT	00'0044h	11h
CAPCOM Register 2	CC2IR	CC2IE	CC2INT	00'0048h	12h
CAPCOM Register 3	CC3IR	CC3IE	CC3INT	00'004Ch	13h
CAPCOM Register 4	CC4IR	CC4IE	CC4INT	00'0050h	14h
CAPCOM Register 5	CC5IR	CC5IE	CC5INT	00'0054h	15h
CAPCOM Register 6	CC6IR	CC6IE	CC6INT	00'0058h	16h
CAPCOM Register 7	CC7IR	CC7IE	CC7INT	00'005Ch	17h
CAPCOM Register 8	CC8IR	CC8IE	CC8INT	00'0060h	18h
CAPCOM Register 9	CC9IR	CC9IE	CC9INT	00'0064h	19h
CAPCOM Register 10	CC10IR	CC10IE	CC10INT	00'0068h	1Ah
CAPCOM Register 11	CC11IR	CC11IE	CC11INT	00'006Ch	1Bh
CAPCOM Register 12	CC12IR	CC12IE	CC12INT	00'0070h	1Ch
CAPCOM Register 13	CC13IR	CC13IE	CC13INT	00'0074h	1Dh
CAPCOM Register 14	CC14IR	CC14IE	CC14INT	00'0078h	1Eh
CAPCOM Register 15	CC15IR	CC15IE	CC15INT	00'007Ch	1Fh
CAPCOM Register 16	CC16IR	CC16IE	CC16INT	00'00C0h	30h
CAPCOM Register 17	CC17IR	CC17IE	CC17INT	00'00C4h	31h
CAPCOM Register 18	CC18IR	CC18IE	CC18INT	00'00C8h	32h
CAPCOM Register 19	CC19IR	CC19IE	CC19INT	00'00CCh	33h
CAPCOM Register 20	CC20IR	CC20IE	CC20INT	00'00D0h	34h
CAPCOM Register 21	CC21IR	CC21IE	CC21INT	00'00D4h	35h
CAPCOM Register 22	CC22IR	CC22IE	CC22INT	00'00D8h	36h
CAPCOM Register 23	CC23IR	CC23IE	CC23INT	00'00DCh	37h
CAPCOM Register 24	CC24IR	CC24IE	CC24INT	00'00E0h	38h
CAPCOM Register 25	CC25IR	CC25IE	CC25INT	00'00E4h	39h
CAPCOM Register 26	CC26IR	CC26IE	CC26INT	00'00E8h	3Ah

Table 12 : Interrupt and PEC service request sources (continued)

Source of Interrupt or PEC Service Request	Request Flag	Enable Flag	Interrupt Vector	Vector Location	Trap Number
CAPCOM Register 27	CC27IR	CC27IE	CC27INT	00'00ECh	3Bh
CAPCOM Register 28	CC28IR	CC28IE	CC28INT	00'00E0h	3Ch
CAPCOM Register 29	CC29IR	CC29IE	CC29INT	00'0110h	44h
CAPCOM Register 30	CC30IR	CC30IE	CC30INT	00'0114h	45h
CAPCOM Register 31	CC31IR	CC31IE	CC31INT	00'0118h	46h
CAPCOM Timer 0	T0IR	T0IE	T0INT	00'0080h	20h
CAPCOM Timer 1	T1IR	T1IE	T1INT	00'0084h	21h
CAPCOM Timer 7	T7IR	T7IE	T7INT	00'00F4h	3Dh
CAPCOM Timer 8	T8IR	T8IE	T8INT	00'00F8h	3Eh
GPT1 Timer 2	T2IR	T2IE	T2INT	00'0088h	22h
GPT1 Timer 3	T3IR	T3IE	T3INT	00'008Ch	23h
GPT1 Timer 4	T4IR	T4IE	T4INT	00'0090h	24h
GPT2 Timer 5	T5IR	T5IE	T5INT	00'0094h	25h
GPT2 Timer 6	T6IR	T6IE	T6INT	00'0098h	26h
GPT2 CAPREL Register	CRIR	CRIE	CRINT	00'009Ch	27h
A/D Conversion Complete	ADCIR	ADCIE	ADCINT	00'00A0h	28h
A/D Overrun Error	ADEIR	ADEIE	ADEINT	00'00A4h	29h
ASC0 Transmit	S0TIR	S0TIE	S0TINT	00'00A8h	2Ah
ASC0 Transmit Buffer	S0TBIR	S0TBIE	S0TBINT	00'011Ch	47h
ASC0 Receive	S0RIR	S0RIE	S0RINT	00'00ACh	2Bh
ASC0 Error	S0EIR	S0EIE	S0EINT	00'00B0h	2Ch
SSC Transmit	SSCTIR	SSCTIE	SSCTINT	00'00B4h	2Dh
SSC Receive	SSCRIR	SSCRIE	SSCRINT	00'00B8h	2Eh
SSC Error	SSCEIR	SSCEIE	SSCEINT	00'00BCh	2Fh
PWM Channel 0...3	PWMIR	PWMIE	PWMINT	00'00FCh	3Fh
CAN1 Interface	XP0IR	XP0IE	XP0INT	00'0100h	40h
CAN2 Interface	XP1IR	XP1IE	XP1INT	00'0104h	41h
XPWM	XP2IR	XP2IE	XP2INT	00'0108h	42h
PLL Unlock	XP3IR	XP3IE	XP3INT	00'010Ch	43h

Table 13 : Vector locations and status for hardware traps

Exception Condition	Trap Flag	Trap Vector	Vector Location	Trap Number	Trap Priority
Reset Functions:					MAXIMAL
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
MAC Interrupt	MACTRAP	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 MINIMAL
Reserved			[2Ch – 3Ch]	[0Bh – 0Fh]	
Software Traps TRAP Instruction			Any [00'0000h – 00'01FCh] in steps of 4h	Any [00h – 7Fh]	Current CPU Priority

The Table 13 lists the vector locations for hardware traps and the corresponding status flags in register TFR.

It also lists the priorities of trap service for cases, where more than one trap condition might be detected within the same instruction.

After any reset (hardware reset, software reset instruction SRST, or reset by watchdog timer overflow) program execution starts at the reset vector at location 00'0000h.

Reset conditions have priority over every other system activity and therefore have the highest priority (trap priority III).

Software traps may be initiated to any vector location between 00'0000h and 00'01FCh. A service routine entered via a software TRAP instruction is always executed on the current CPU priority level which is indicated in bit-field ILVL in register PSW.

This means that routines entered via the software TRAP instruction can be interrupted by all hardware traps or higher level interrupt requests.

6.1.1 - Normal Interrupt Processing and PEC Service

At each instruction cycle, among all the sources, which require a PEC or an interrupt processing, only the one with the highest priority is selected. The priority of interrupts and PEC requests is programmable in two levels. Each requesting source can be assigned to a specific priority.

A second level (called "group priority") allows to specify an internal order for simultaneous requests from a group of different sources on the same priority level.

At the end of each instruction cycle the request with the highest current priority will be determined by the interrupt system. The request will be serviced. If its priority is higher than the current CPU priority which is stored in the register PSW.

6.1.2 - Interrupt System Register Description

Interrupt processing is globally controlled by register PSW through a general interrupt enable bit (IEN) and the CPU priority field (ILVL). Additionally the different interrupt sources are individually controlled by their specific interrupt control registers (...IC).

Thus, the acceptance of requests by the CPU is determined by both the individual interrupt control registers and the PSW. PEC services are controlled by the respective PECCx register and the source and destination pointers, which specify the task of the respective PEC service channel.

6.1.3 - Interrupt Control Registers

All interrupt control registers are identically organized. The lower 8 bits of an interrupt control register contain the complete interrupt status information of the associated source, which is required during one round of prioritization, the upper 8 bits of the respective register are reserved. All interrupt control registers are bit-addressable and all bits can be read or written via software.

This allows each interrupt source to be programmed or modified with just one instruction. When accessing interrupt control registers through instructions which operate on word data types, their upper 8 bits (15...8) will return zeros, when read, and will discard written data.

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

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

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

Bit	Function
GLVL	**Group Level** Defines the internal order for simultaneous requests of the same priority. 3: Highest group priority 0: Lowest group priority
ILVL	**Interrupt Priority Level** Defines the priority level for the arbitration of requests. 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

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

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

6.1.4 - Interrupt Priority Level and Group Level

The four bits of ILVL bit-field 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 gets active at the same time, the values in the respective bit fields GLVL are used for second level arbitration to select one request for being serviced. Again the group priority increases with the numerical value of GLVL, so 00b is the lowest and 11b is the highest group priority.

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

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

The interrupt system of the ST10F280 allows nesting of up to 15 interrupt service routines of different priority levels (level 0 cannot be arbitrated).

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

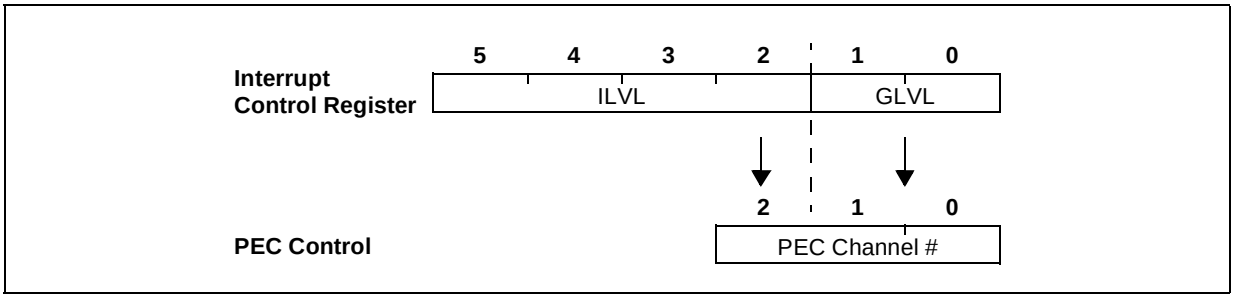
Note Priority level 0000b is the default level of the CPU. Therefore a request on level 0 will never be serviced, because it can never interrupt the CPU. However, an enabled interrupt request on level 0000b will terminate the ST10F280's Idle mode and reactivate the CPU.

For interrupt requests which are to be serviced by the PEC, the associated PEC channel number is derived from the respective ILVL (LSB) and GLVL (see Figure 21). So programming a source to priority level 15 (ILVL=1111b) selects the PEC channel group 7...4, 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 (see Figure 21).

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

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

Figure 21 : Priority levels and PEC channels



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

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

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 functionally divided into 2 parts: the lower byte of the PSW basically represents the arithmetic status of the CPU, the upper byte of the PSW controls the interrupt system of the ST10F280 and the arbitration mechanism for the external bus interface.

Note Pipeline effects have to be considered when enabling/disabling interrupt requests via modifications of register PSW (see Chapter 4 - The Central Processing Unit (CPU)).

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** (Described in section "The Central Processing Unit") Define the current status of the CPU (ALU, multiplication unit).
HLDEN	**HOLD Enable** (Enables External Bus Arbitration) 0: Bus arbitration disabled, P6.7...P6.5 may be used for general purpose I/O 1: Bus arbitration enabled, P6.7...P6.5 serve as BREQ, HLDA, HOLD, respectively
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 current level for the operation of the CPU. This bit field reflects the priority level of the routine that is currently executed. Upon the entry into an interrupt service routine this bit field is updated with the priority level of the request that is being serviced. The PSW is saved on the system stack before. The CPU level determines the minimum interrupt priority level that will be serviced. Any request on the same or a lower level will not be acknowledged.

The current CPU priority level may be adjusted via software to control which interrupt request sources will be acknowledged.

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

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

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

Interrupt Enable bit IEN globally enables or disables PEC operation and the acceptance of interrupts by the CPU. When IEN is cleared, no interrupt requests are accepted by the CPU. When IEN is set to '1', all interrupt sources, which have been individually enabled by the interrupt enable bit in their associated control registers, are globally enabled.

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

6.2 - Operation of the PEC Channels

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

PECCx (FECyh / 6zh, see Table 14) 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)

Table 14 : PEC control register addresses

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

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

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

Note The reserved combination '11' is changed to '10' by hardware. Do not to use this combination.

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

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

Previous COUNT	Modified COUNT	IR after PEC service	Action of PEC Channel and Comments
FFh	FFh	'0'	Move a byte / word Continuous transfer mode, COUNT is not modified
FEh..02h	FDh..01h	'0'	Move a byte / word and decrement COUNT
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.

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

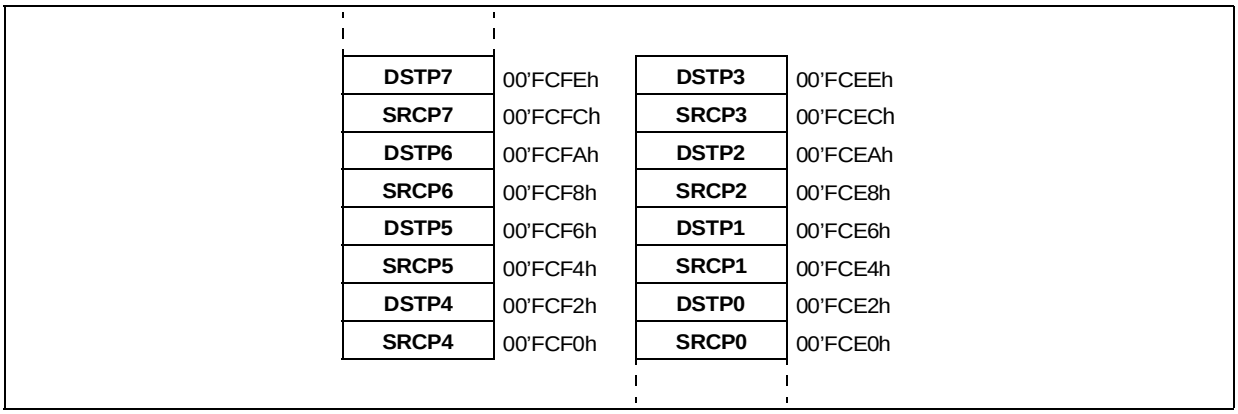
Continuous transfers are selected by the value FFh in bit-field COUNT. In this case COUNT is not modified and the respective PEC channel services any request until it is disabled again.

When COUNT is decremented from 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 respective PEC channel remains idle and the associated interrupt service routine is activated instead. This allows to choose, if a level 15 or 14 request is to be serviced by the PEC or by the interrupt service routine.

Note PEC transfers are only executed, if their priority level is higher than the CPU level, for example 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 ST10F280 just below the bit-addressable area (see Figure 22).

Figure 22 : Mapping of PEC pointers into the internal RAM



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

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

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

6.3 - Prioritizing Interrupt & PEC Service Requests

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

6.3.1 - Enabling and Disabling Interrupt Requests

This may be done in three ways:

– **Control bit** allow to switch each individual source "ON" or "OFF", so it may generate a request or not. The control bit (xxIE) are located in the respective interrupt control registers. All interrupt requests may be enabled or disabled generally via bit IEN in register PSW. This control bit is the "main switch" that selects, if requests from any source are accepted or not.
In order to be arbitrated, both dedicated and global enable bit of the interrupt source must be set.

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

– **The ATOMIC and EXTend instructions** automatically disable all interrupt requests for the duration of the following 1...4 instructions. This is useful for semaphore handling and does not require to re-enable the interrupt system after the inseparable instruction sequence (see Chapter 22 - System Programming).

6.3.2 - Interrupt Class Management

An interrupt class covers a set of interrupt sources with the same priority from the system's viewpoint. Interrupts of the same class must not interrupt each other. The ST10F280 supports this function with two features:

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

Classes with more than 4 members can be established by using a number of adjacent interrupt priorities (ILVL) and the respective group levels (4 per ILVL).

Each interrupt service routine within this class sets the CPU level to the highest interrupt priority within the class. All requests from the same or any lower level are blocked now, and no request of this class will be accepted.

The example below establishes 3 interrupt classes which cover 2 or 3 interrupt priorities, depending on the number of members in a class.

A level 6 interrupt disables all other sources in class 2 by changing the current CPU level to 8, which is the highest priority (ILVL) in class 2. Class 1 requests or PEC requests are still serviced in this case.

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

Table 15 : Example software controlled interrupt classes

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					
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!

6.4 - Saving the Status During Interrupt Service

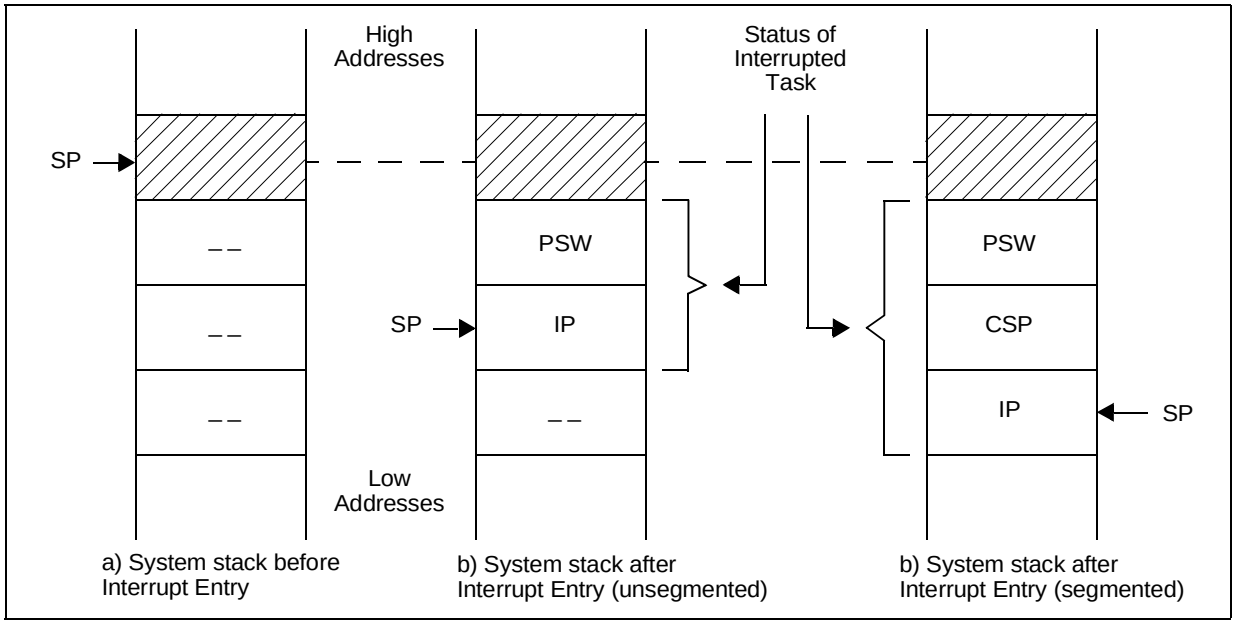
Before an interrupt request that has been arbitrated is actually serviced, the status of the current task is automatically saved on the system stack. The CPU status (PSW) is saved along with the location, where the execution of the interrupted task is to be resumed after returning from the service routine.

This return location is specified through the Instruction Pointer (IP) and, in case of a segmented memory model, the Code Segment Pointer (CSP). Bit SGTDIS in register SYSCON control, how the return location is stored.

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

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

Figure 23 : Task status saved on the system stack



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

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

6.4.1 - 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 wasted with saving and restoring. The ST10F280 allows to switch the complete bank of CPU registers (GPRs) with a single instruction, so the service routine executes within its own, separate context.

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

6.5 - Interrupt Response Times

The interrupt response time defines the time from an interrupt request flag of an enabled interrupt source being set until the first instruction (I1) being fetched from the interrupt vector location. The basic interrupt response time for the ST10F280 is 3 instruction cycles (see Figure 24).

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 (wait-states) therefore influences the interrupt response time.

In the Figure 24 the respective interrupt request flag is set in cycle 1 (fetching of instruction N). The indicated source wins the prioritization round (during cycle 2). In cycle 3 a TRAP instruction is injected into the decode stage of the pipeline, replacing instruction N+1 and clearing the source's interrupt request flag to '0'. Cycle 4 completes the injected TRAP instruction (save PSW, IP and CSP, if segmented mode) and fetches the first instruction (I1) from the respective vector location.

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

Figure 24 : Pipeline diagram for interrupt response time

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

The minimum interrupt response time is 5 CPU clock cycles. This requires program execution from the internal Flash, no external operand read requests and 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 under these conditions is 6 CPU clock cycles.

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

– When internal hold conditions between instruction pairs N-2/N-1 or N-1/N occur, or instruction N explicitly writes to the PSW or the SP, the minimum interrupt response time may be extended by 1 CPU clock cycle for each of these conditions.

– When instruction N reads an operand from the internal memory, or when N is a CALL, RETURN, TRAP, or MOV Rn, [Rm+ #data16] instruction, the minimum interrupt response time may additionally be extended by 2 CPU clock cycles during internal Flash program execution.

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

The worst case interrupt response time during internal Flash program execution adds to 12 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 instructions, source and destination operands are located, there are a number of combinations. Note, however, that only access conflicts contribute to the delay.

A few examples illustrate these delays:

– The worst case interrupt response time including external accesses, occurs when instructions N, N+1 and N+2 are executed from external memory, instructions N-1 and N require external operand read accesses, instructions N-3 to N write back external operands, and the interrupt vector also points to an external location. In this case the interrupt response time is the time to perform 9 word bus accesses, because instruction I1 cannot be fetched via the external bus until all write, fetch and read requests of preceding instructions in the pipeline are terminated.

– When the above example has the interrupt vector pointing into the internal Flash, the interrupt response time is 7 word bus accesses plus 2 CPU clock cycles, because fetching of instruction I1 from internal Flash can start earlier.

– When instructions N, N+1 and N+2 are executed out of external memory and the interrupt vector also points to an external location, but all operands for instructions N-3 through N are in internal memory, then the interrupt response time is the time to perform 3 word bus accesses.

– When the above example has the interrupt vector pointing into the internal Flash, the interrupt response time is 1 word bus access plus 4 CPU clock cycles.

After an interrupt service routine has been terminated by executing the RETI instruction, and if further interrupts are pending, the next interrupt service routine will not be entered until at least two instruction cycles have been executed of the program that was interrupted.

In most cases two instructions will be executed during this time. Only one instruction will typically be executed, if the first instruction following the RETI instruction is a branch instruction (without cache hit), or if it reads an operand from internal Flash, or if it is executed out of the internal RAM.

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

6.5.1 - PEC Response Times

The PEC response time defines the time from an interrupt request flag of an enabled interrupt source being set until the PEC data transfer being started. The basic PEC response time for the ST10F280 is 2 instruction cycles.

Figure 25 : Pipeline diagram for PEC response time

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**

In the Figure the respective interrupt request flag is set in cycle 1 (fetching of instruction N). The indicated source wins the prioritization round (during cycle 2). In cycle 3 a PEC transfer "instruction" is injected into the decode stage of the pipeline, suspending instruction N+1 and clearing the source's interrupt request flag to '0'. Cycle 4 completes the injected PEC transfer and resumes the execution of instruction N+1. All instructions that entered the pipeline after setting of the interrupt request flag (N+1, N+2) will be executed after the PEC data transfer.

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

The minimum PEC response time is 3 CPU clock cycles. This requires program execution from the internal Flash, no external operand read requests and 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 under these conditions is 4 CPU clock cycles. 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.
– When instruction N reads an operand from the internal Flash, or when N is a CALL, RETURN, TRAP, or MOV Rn, [Rm+ #data16] instruction, the minimum PEC response time may additionally be extended by 2 CPU clock cycles during internal Flash program execution.
– In case instruction N reads the PSW and instruction N-1 has an effect on the condition flags, the PEC response time may additionally be extended by 2 CPU clock cycles.

The worst case PEC response time during internal Flash program execution adds to 9 CPU clock cycles. Any reference to external locations increases the PEC response time due to pipeline related access priorities. The following conditions have to be considered:

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

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

A few examples illustrate these delays:

– The worst case interrupt response time including external accesses will occur, when instructions N and N+1 are executed out of external memory, instructions N-1 and N require external operand read accesses and instructions N-3, N-2 and N-1 write back external operands. In this case the PEC response time is the time to perform 7 word bus accesses.
– When instructions N and N+1 are executed out of external memory, but all operands for instructions N-3 through N-1 are in internal memory, then the PEC response time is the time to perform 1 word bus access plus 2 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 the additional time it might take to fetch the source operand from internal Flash or external memory and to write the destination operand over the external bus in an external program environment.

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

6.6 - External Interrupts

Although the ST10F280 has no dedicated interrupt input pins, it provides many possibilities to react on external asynchronous events by using a number of I/O lines for interrupt input. The interrupt function may either be combined with the pin's main function or may be used instead of it, if the main pin function is not required. Interrupt signals may be connected to:

– CC31IO...CC0IO, the capture input / compare output lines of the CAPCOM units,
– T4IN, T2IN, the timer input pins,
– CAPIN, the capture input of GPT2.

For each of these pins either a positive, a negative, or both a positive and a negative external transition can be selected to cause an interrupt or PEC service request. The edge selection is performed in the control register of the peripheral device associated with the respective port pin.

The peripheral must be programmed to a specific operating mode to allow generation of an interrupt by the external signal. The priority of the interrupt request is determined by the interrupt control register of the respective peripheral interrupt source, and the interrupt vector of this source will be used to service the external interrupt request.

Note In order to use any of the listed pins as external interrupt input, it must be switched to input mode via its direction control bit DPx.y in the respective port direction control register DPx (see Table 16).

When port pins CCxI/O are used as external interrupt input pins, bit field CCMODx in the control register of the corresponding capture/compare register CCx must select capture mode.

When CCMODx is programmed to 001b, the interrupt request flag CCxIR in register CCxIC will be set on a positive external transition at pin CCxI/O.

When CCMODx is programmed to 010b, a negative external transition will set the interrupt request flag. When CCMODx=011b, both a positive and a negative transition will set the request flag. In all three cases, the contents of the allocated CAPCOM timer will be latched into capture register CCx, independent whether the timer is running or not. When the interrupt enable bit CCxIE is set, a PEC request or an interrupt request for vector CCxINT will be generated (see Table 16).

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.

Table 16 : Pins to be used as external interrupt inputs

Port Pin	Original Function	Control Register
P2.0-15/CC0-15I/O	CAPCOM Register 0-15 Capture Input	CC0-CC15
P8.0-7/CC16-23I/O	CAPCOM Register 16-23 Capture Input	CC16-CC23
P1H.4-7/CC24-27I/O	CAPCOM Register 24-27 Capture Input	CC24-CC27
P7.4-7/CC28-31I/O	CAPCOM Register 28-31 Capture Input	CC28-CC31
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

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 bit T2IE or T4IE are set, a PEC request or an interrupt request for vector T2INT or T4INT will be generated.

Pin CAPIN differs slightly from the timer input pins as it can be used as external interrupt input pin without affecting peripheral functions.

When the capture mode enable bit T5SC in register T5CON is cleared to '0', signal transitions on pin CAPIN will only set the interrupt request flag CRIR in register CRIC, and the capture function of register CAPREL is not activated.

So register CAPREL can still be used as reload register for GPT2 timer T5, while pin CAPIN serves as external interrupt input. Bit field CI in register T5CON selects the effective transition of the external interrupt input signal.

When CI is programmed to 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 $\overline{\text{NMI}}$ and the reset input $\overline{\text{RSTIN}}$ provide another possibility for the CPU to react on an external input signal. $\overline{\text{NMI}}$ and $\overline{\text{RSTIN}}$ are dedicated input pins, which cause hardware traps.

6.6.1 - Fast External Interrupts

The input pins that may be used for external interrupts are sampled every 8 CPU clock cycles this means that the external events are scanned and detected in timeframes of 8 CPU clock cycles.

The ST10F280 provides 8 interrupt inputs that are sampled every CPU clock cycle so external events are captured faster than with standard interrupt inputs.

The upper 8 pins of Port2 (CC8-15 I/O on P2.8-P2.15) can individually be programmed to this fast interrupt mode. In this mode the trigger transition (rising, falling or both) can also be selected. The External Interrupt Control register EXICON controls this feature for all 8 pins. In addition, these fast interrupt inputs feature programmable edge detection (rising edge, falling edge or both edges).

Fast external interrupts may also have interrupt sources selected from other peripherals; for example the CANx controller receive signal (CANx_RxD) can be used to interrupt the system. This new function is controlled using the 'External Interrupt Source Selection' register EXISEL.

The EXxIN pins can also be used to exit power down mode if bit PWDCFG in the SYSCON register is set. Power reduction modes are detailled in Chapter 20 - Power Reduction Modes.

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

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

Bit	Function
EXIxES (x=7...0)	**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' (ref as 'high' active level) 1 0: Interrupt on negative edge (falling) Enter Power Down mode if EXiIN = '1', exit if EXxIN = '0' (ref as 'low' active level) 1 1: Interrupt on any edge (rising or falling) Always enter Power Down mode, exit if EXxIN level changed.

These fast external interrupts use the interrupt nodes and vectors of the CAPCOM channels CC8-CC15, so the capture/compare function cannot be used on the respective Port2 pins (with EXIxES ≠ 00b). However, general purpose I/O is possible in all cases.

Note The fast external interrupt inputs are sampled every 8 CPU clock cycles. The interrupt request arbitration and processing is executed every 4 CPU clock cycles.

EXISEL (F1DAh / EDh) ESFR Reset Value: 0000h

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
EXI7SS	EXI6SS	EXI5SS	EXI4SS	EXI3SS	EXI2SS	EXI1SS	EXI0SS
RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
EXIxSS	**External Interrupt x Source Selection (x=7...0)** '00': Input from associated Port 2 pin. '01': Input from "alternate source". '10': Input from Port 2 pin ORed with "alternate source". '11': Input from Port 2 pin ANDed with "alternate source".

EXIxSS	Port 2 pin	Alternate Source
0	P2.8	CAN1_RxD
1	P2.9	CAN2_RxD
2...7	P2.10...15	Not used (zero)

6.7 - Trap Functions

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

The ST10F280 provides two different kinds of trap mechanisms. **Hardware traps** are triggered by events that occur during program execution (like illegal access or undefined opcode), **software traps** are initiated via an instruction within the current execution flow.

The Traps priorities are summarized in Table 17.

Table 17 : Trap priorities

Exception Condition	Trap Flag	Trap Vector	Vector Location	Trap Number	Trap Priority
Reset Functions:					MAXIMAL
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
MAC Interrupt	MACTRAP	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
					MINIMAL

6.7.1 - Software Traps

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

Executing a TRAP instruction causes the same effect as servicing the interrupt at the same vector. PSW, CSP (in segmentation mode), and IP are pushed on the internal system stack and a jump is taken to the specified vector location.

When segmentation is enabled and a trap is executed, the CSP for the trap service routine is set to code segment 0. No Interrupt Request flags are affected by the TRAP instruction.

The interrupt service routine called by a TRAP instruction must be terminated with a RETI (return from interrupt) instruction to ensure correct operation.

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

6.7.2 - Hardware Traps

Hardware traps are issued by faults or specific system states that occur during the runtime of a program (not identified at assembly time). A hardware trap may also be triggered intentionally, for example to emulate additional instructions by generating an Illegal Opcode trap.

The ST10F280 distinguishes eight different hardware trap functions. When a hardware trap condition has been detected, the CPU branches to the trap vector location for the respective trap condition.

Depending on the trap condition, the instruction which caused the trap is either completed or cancelled (it has no effect on the system state) before the trap handling routine is entered.

Hardware traps are non-maskable and always have priority over every other CPU activity. If several hardware trap conditions are detected within the same instruction cycle, the highest priority trap is serviced (see Section 6.1 - Interrupt System Structure).

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

The eight hardware trap functions of the ST10F280 are divided into two classes:

– **Class A traps**: These traps share the same trap priority, but have an individual vector address.
  – External Non-Maskable Interrupt ($\overline{\text{NMI}}$)
  – Stack Overflow
  – Stack Underflow trap
– **Class B traps**: These traps share the same trap priority, and the same vector address.
  – Undefined Opcode
  – Protection Fault
  – MAC interrupt
  – Illegal word Operand Access
  – Illegal Instruction Access
  – Illegal External Bus Access Trap

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	**STKOF**	**STKUF**	-	-	-	-	-	**UNDOPC**	**MACTRP**	-	-	**PRTFLT**	**ILLOPA**	**ILLINA**	**ILLBUS**
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.
MACTRP	**MAC Interrupt Flag** The MAC coprocessor has generated an interruption.
UNDOPC	**Undefined Opcode Flag** The currently decoded instruction has no valid ST10F280 opcode.
STKUF	**Stack Underflow Flag** The current stack pointer value exceeds the content of register STKUN.
STKOF	**Stack Overflow Flag** The current stack pointer value falls below the content of register STKOV.
NMI	**Non Maskable Interrupt Flag** A negative transition (falling edge) has been detected on pin $\overline{\text{NMI}}$.

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

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

Class A traps have the second highest priority (Trap Priority II), on the 3rd rank are class B traps, so a class A trap can interrupt a class B trap. If more than one class A trap occur at a time, they are prioritized internally, with the NMI trap on the highest priority followed by the stack overflow trap and the stack underflow trap has 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 to service simultaneous class B traps is determined by the software in the trap service routine.

If a class A trap occurs during the execution of a class B trap service routine, class A trap will be serviced immediately. During the execution of a class A trap service routine, no class B trap will 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.

If 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.

6.7.3 - External NMI Trap

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

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

6.7.4 - Stack Overflow Trap

Whenever the stack pointer is decremented to a value which is less than the value in the stack overflow register STKOV, the STKOF flag in register TFR is set and the CPU will enter the stack overflow trap routine. Which IP value will be pushed onto the system stack depends on which operation caused the decrement of the SP.

When an implicit decrement of the SP is made through a PUSH or CALL instruction, or upon interrupt or trap entry, the IP value pushed is the address of the following instruction. When the SP is decremented by a subtract instruction, the IP value pushed represents the address of the instruction after the instruction following the subtract instruction.

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

6.7.5 - Stack Underflow Trap

Whenever the stack pointer is incremented to a value which is greater than the value in the stack underflow register STKUN, the STKUF flag is set in register TFR and the CPU will enter the stack underflow trap routine. Again, 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 pushed is the address of the following instruction.

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

6.7.6 - Undefined Opcode Trap

When the instruction currently decoded by the CPU does not contain a valid ST10F280 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 non-implemented instructions. The trap service routine can examine the faulting instruction to decode operands for non-implemented opcodes based on the stacked IP. In order to resume processing, the stacked IP value must be incremented by the size of the undefined instruction, which is determined by the user, before a RETI instruction is executed.

6.7.7 - Protection Fault Trap

The format of the protected instructions is 4 byte wide. Byte 1 and 2 are complementary values. Byte 3 and 4 are identical to byte 1. For example the format of SRST instruction is B7h 48h B7h B7h. If the format of a protected instruction going to be executed does not fulfill this coding, the PRTFLT flag in register TFR is set and the CPU enters the protection fault trap routine. The protected instructions include DISWDT, EINIT, IDLE, PWRDN, SRST, and SRVWDT. When the protection fault trap occurs, the IP value pushed onto the system stack is the address of the faulty instruction.

6.7.8 - 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 MCV 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.

6.7.9 - 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.

6.7.10 - Illegal Instruction Access Trap

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

6.7.11 - 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.

7 - PARALLEL PORTS

7.1 - Introduction

The ST10F280 has up to 143 parallel I/O lines, organized into,

– Eight 8-bit I/O ports (PORT0 made of P0H and P0L, PORT1 made of P1H and P1L, Port4, Port 6, Port 7, Port8),

– One 15-bit I/O port (Port3),

– Two 16-bit input ports (Port5 and XPort10),

– Two 16-bit I/O ports (Port2 and XPort9).

These port lines may be used for general purpose Input/Output, controlled via software, or may be used implicitly by ST10F280's integrated peripherals or the External Bus Controller.

All port lines are bit addressable, and all input/output lines are individually (bit wise) programmable as inputs or outputs via direction registers (except Port5 and XPort10). The I/O ports are true bidirectional ports which are switched to high impedance state when configured as inputs.

The output drivers of seven I/O ports (2, 3, 4, 6, 7, 8, 9) 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.

7.1.1 - Open Drain Mode

Some of I/O Ports of ST10F280 provide Open Drain Control. It is used 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 pull-up device. With this feature, it is possible to connect several port pins together to a AND-wired configuration, saving external glue logic and/or additional software overhead for enabling/disabling output signals.

This feature is implemented for ports P2, P3, P6, P7, P8, XP9 and P4 partially 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 (see Figure 27).

Figure 26 : SFRs and pins associated with the parallel ports

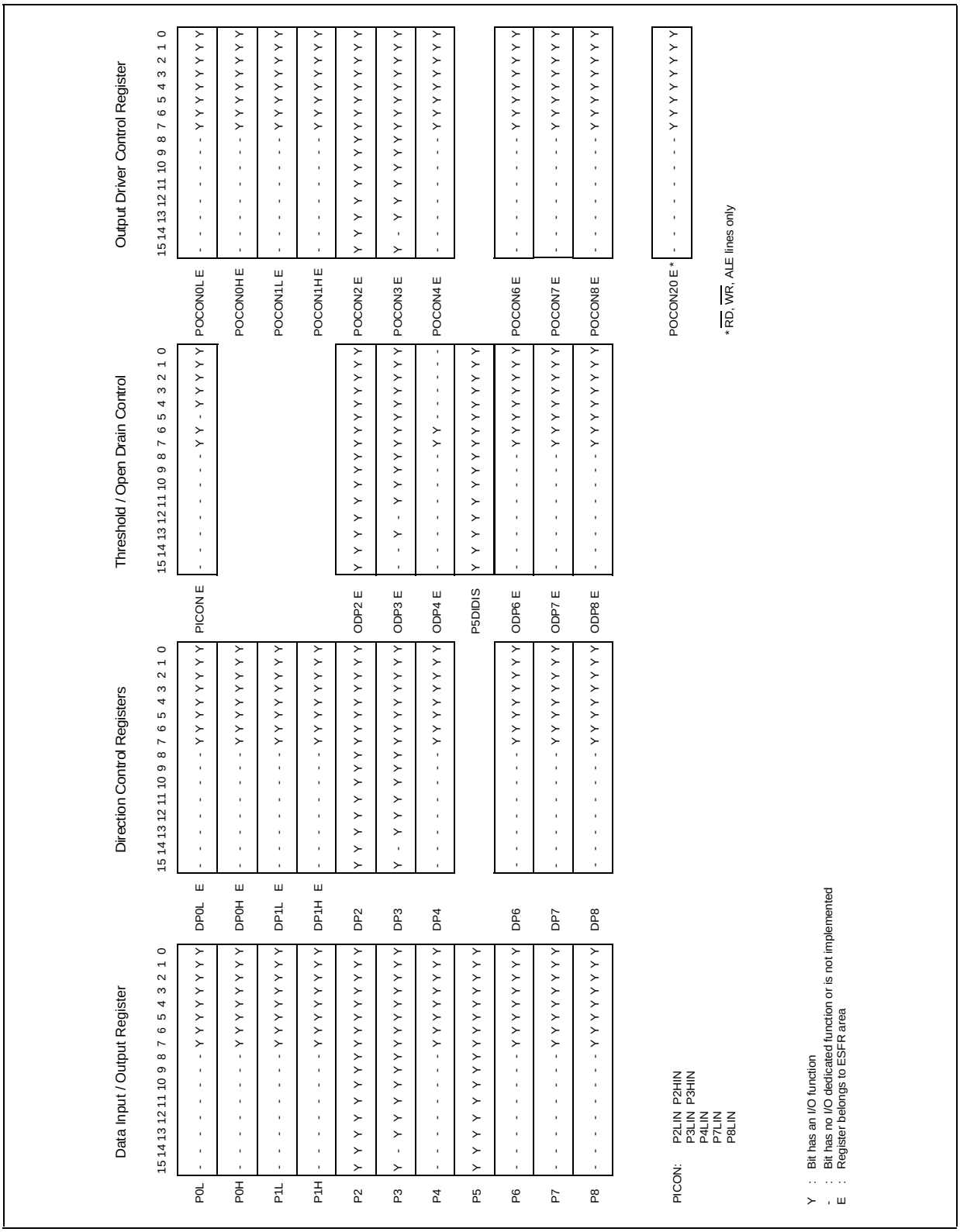
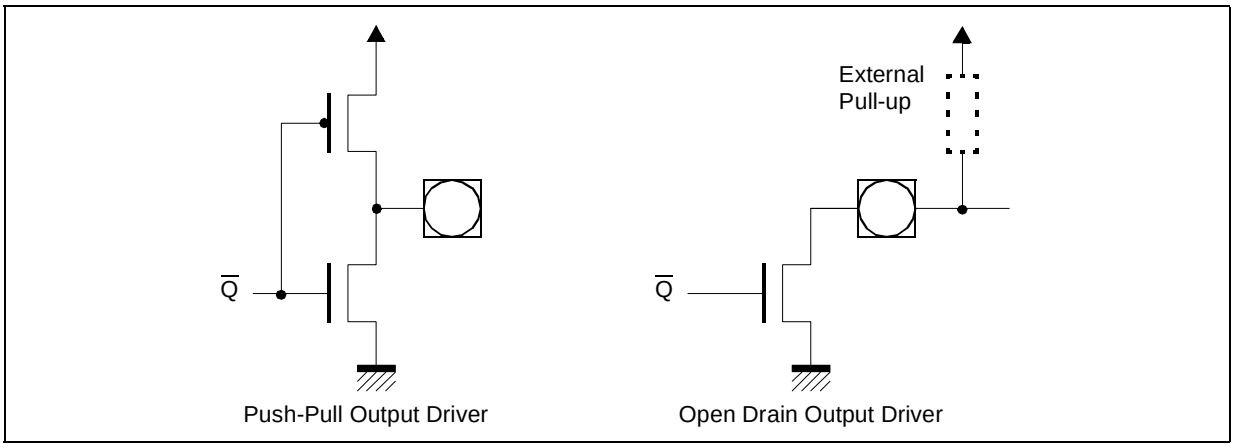


Figure 27 : Output drivers in push-pull mode and in open drain mode



7.1.2 - Input Threshold Control

The standard inputs of the ST10F280 determine the status of input signals according to TTL levels.

In order to accept and recognize noisy signals, CMOS-like input thresholds can be selected instead of the standard TTL thresholds for all pins of Port2, Port3, Port4, Port7 and Port8. These special thresholds are defined above the TTL thresholds and feature a defined hysteresis to prevent the inputs from toggling while the respective input signal level is near the thresholds.

The Port Input Control register PICON is used to select these thresholds for each byte of the indicated ports, the 8-bit ports P4, P7 and P8 are controlled by one bit each while ports P2 and P3 are controlled by two bits each.

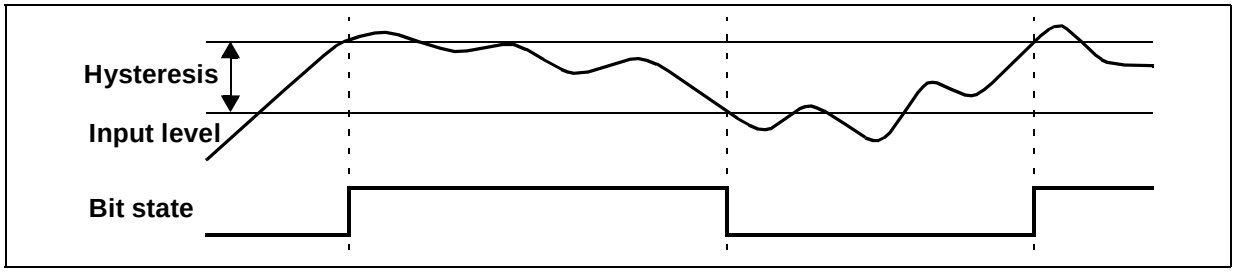
PICON (F1C4h / E2h) ESFR Reset Value: - - 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	P8LIN	P7LIN	-	P4LIN	P3HIN	P3LIN	P2HIN	P2LIN
								RW	RW			RW	RW	RW	RW

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

All options for individual direction and output mode control are available for each pin, independent of the selected input threshold. The input hysteresis provides stable inputs from noisy or slowly changing external signals.

Figure 28 : Hysteresis for special input thresholds



7.1.3 - Alternate Port Functions

Each port line has one associated programmable alternate input or output function. PORT0 and PORT1 may be used as the address and data lines when accessing external memory.

Port4 outputs the additional segment address bit A23/19/17...A16 in systems where more than 64K bytes of memory are to be accessed directly.

Port6 provides the optional chip select outputs and the bus arbitration lines.

Port2, Port7 and Port8 are associated with the capture inputs or compare outputs of the CAPCOM units and/or with the outputs of the PWM module.
Port2 is also used for fast external interrupt inputs and for timer 7 input.

Port3 includes alternate input/output functions of timers, serial interfaces, the optional bus control signal BHE/WRH and the system clock output (CLKOUT). Port5 is used for the analog input channels to the A/D converter or timer control signals.

If the alternate output function of a pin is to be used, the direction of this pin must be programmed for output (DPx.y='1'), except for some signals that are used directly after reset and are configured automatically. Otherwise the pin remains in the high-impedance state and is not effected by the alternate output function. The respective port latch should hold a '1', because its output is ANDed with the alternate output data (except for PWM output signals).

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

On most of the port lines, the user software is responsible for setting the proper direction when using an alternate input or output function of a pin.

This is done by setting or clearing the direction control bit DPx.y of the pin before enabling the alternate function.

There are port lines, however, where the direction of the port line is switched automatically.

For instance, in the multiplexed external bus modes of PORT0, the direction must be switched several times for an instruction fetch in order to output the addresses and to input the data.

Obviously, this cannot be done through instructions. In these cases, the direction of the port line is switched automatically by hardware if the alternate function of such a pin is enabled.

To determine the appropriate level of the port output latches check how the alternate data output is combined with the respective port latch output.

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

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

Note When using several BSET pairs to control more pins of one port, these pairs must be separated by instructions, which do not reference the respective port (see "Particular Pipeline Effects" in chapter "The Central Processing Unit").

```
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
```

7.1.4 - Output Driver Control

The port output control registers POCONx allow to select the port output driver characteristics of a port. The aim of these selections is to adapt the output drivers to the application's requirements, and to improve the EMI behaviour of the device. Two characteristics may be selected:

Edge characteristic defines the rise/fall time for the respective output. Slow edges reduce the peak currents that are sinked/sourced when changing the voltage level of an external capacitive load. For a bus interface or pins that are changing at frequency higher than 1MHz, however, fast edges may still be required.

Driver characteristic defines either the general driving capability of the respective driver, or if the driver strength is reduced after the target output level has been reached or not. Reducing the driver strength increases the output's internal resistance, which attenuates noise that is imported via the output line. For driving LEDs or power transistors, however, a stable high output current may still be required as described below.

This rise / fall time of 4 I/O pads (a nibble) is selected using 2-bit named **PNxEC**. That means **P**ort **N**ibble (**x** = nibble number, it could be 3 as for Port 2.15 to 2.12) **E**dge **C**haracteristic.

The sink / source capability of the same 4 I/O pads is selected using 2-bit named **PNxDC**. That means **P**ort **N**ibble (**x** = nibble number) **D**rive **C**haracteristic (See Table 18).

POCONx (F0yyh / zzh) for 8-bit Ports ESFR Reset Value: - - 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	PN1DC		PN1EC		PN0DC		PN0EC	
								RW		RW		RW		RW	

POCONx (F0yyh / zzh) for 16-bit Ports ESFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PN3DC		PN3EC		PN2DC		PN2EC		PN1DC		PN1EC		PN0DC		PN0EC	
RW		RW		RW		RW		RW		RW		RW		RW	

Bit	Function
PNxEC	**Port Nibble x Edge Characteristic (rise/fall time)** 00: Fast edge mode, rise/fall times depend on the size of the driver. 01: Slow edge mode, rise/fall times ~60 ns 10: Reserved 11: Reserved
PNxDC	**Port Nibble x Driver Characteristic (output current)** 00: High Current mode: Driver always operates with maximum strength. 01: Dynamic Current mode: Driver strength is reduced after the target level has been reached. 10: Low Current mode: Driver always operates with reduced strength. 11: Reserved

Note: In case of reading an 8 bit P0CONx register, high byte (bit 15..8) is read as 00h

The table lists the defined POCON registers and the allocation of control bit fields and port pins.

Table 18 : Port control register allocation

Control Register	Physical Address	8-bit Address	Controlled Port Nibble			
			3	2	1	0
POCON0L	F080h	40h			P0L.7...4	P0L.3...0
POCON0H	F082h	41h			P0H.7...4	P0H.3...0
POCON1L	F084h	42h			P1L.7...4	P1L.3...0
POCON1H	F086h	43h			P1H.7...4	P1H.3...0
POCON2	F088h	44h	P2.15...12	P2.11...8	P2.7...4	P2.3...0
POCON3	F08Ah	45h	P3.15, 3.13, 3.12	P3.11...8	P3.7...4	P3.3...0
POCON4	F08Ch	46h			P4.7...4	P4.3...0
POCON6	F08Eh	47h			P6.7...4	P6.3...0
POCON7	F090h	48h			P7.7...4	P7.3...0
POCON8	F092h	49h			P8.7...4	P8.3...0

Dedicated Pins Output Control

Programmable pad drivers also are supported for the dedicated pins ALE, $\overline{RD}$ and $\overline{WR}$. For these pads, a special POCON20 register is provided.

POCON20 (F0AAh / 55h) ESFR Reset Value: - - 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	PN1DC		PN1EC		PN0DC		PN0EC	
								RW		RW		RW		RW	

Bit	Function
PN0EC	**$\overline{RD}$, $\overline{WR}$ Edge Characteristic (rise/fall time)** 00: Fast edge mode, rise/fall times depend on the size of the driver. 01: Slow edge mode, rise/fall times ~60 ns 10: Reserved 11: Reserved
PN0DC	**$\overline{RD}$, $\overline{WR}$ Driver Characteristic (output current)** 00: High Current mode: Driver always operates with maximum strength. 01: Dynamic Current mode: Driver strength is reduced after the target level has been reached. 10: Low Current mode: Driver always operates with reduced strength. 11: Reserved
PN1EC	**ALE Edge Characteristic (rise/fall time)** 00: Fast edge mode, rise/fall times depend on the size of the driver. 01: Slow edge mode, rise/fall times ~60 ns 10: Reserved 11: Reserved
PN1DC	**ALE Driver Characteristic (output current)** 00: High Current mode: Driver always operates with maximum strength. 01: Dynamic Current mode: Driver strength is reduced after the target level has been reached. 10: Low Current mode: Driver always operates with reduced strength. 11: Reserved

7.2 - Port0

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

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

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

7.2.1 - 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 de-multiplexed 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 pull-up device.

Each line can now be individually pulled to a low level (see DC-level specifications in the respective Data Sheets) through an external pull-down device. A default configuration is selected when the respective PORT0 lines are at a high level. Through pulling individual lines to a low level, this default can be changed according to the needs of the applications.

The internal pull-up devices are designed such that an external pull-down resistors (see Data Sheet specification) can be used to apply a correct low level.

These external pull-down resistors can remain connected to the PORT0 pins also during normal operation, however, care 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.

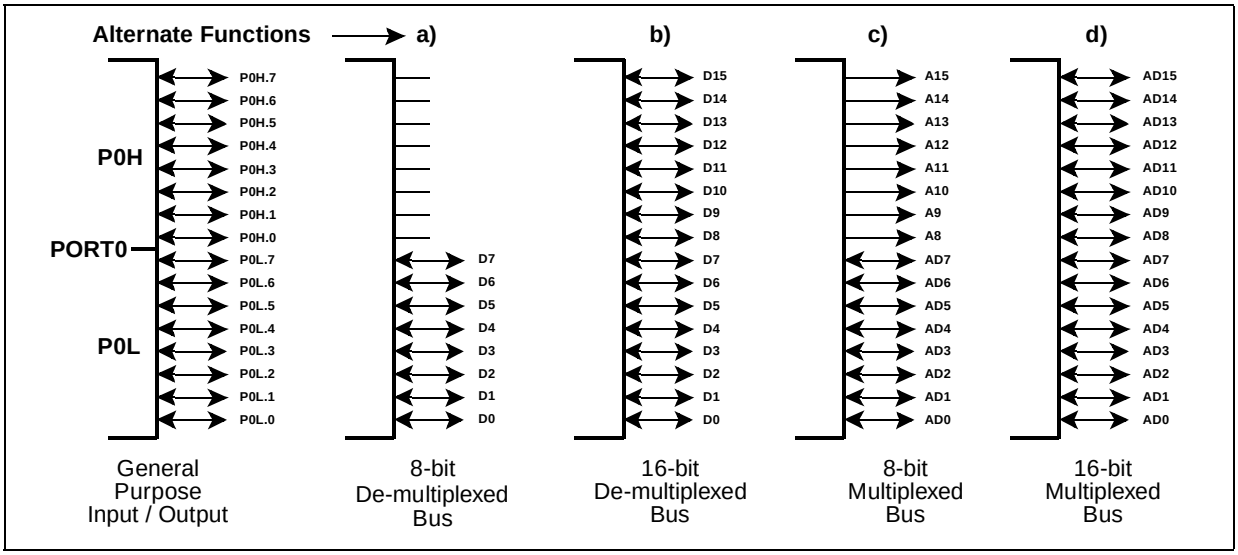
This read-only register holds the selection for the number of chip selects and segment addresses. Software can read this register in order to react according to the selected configuration, if required. When the reset is terminated, the internal pull-up devices are switched off, and PORT0 will be switched to the appropriate operating mode.

During external accesses in multiplexed bus modes PORT0 first outputs the 16-bit intra-segment address as an alternate output function. PORT0 is then switched to high-impedance input mode to read the incoming instruction or data.

In 8-bit data bus mode, two memory cycles are required for word accesses, the first for the low byte and the second for the high byte of the word. During write cycles PORT0 outputs the data byte or word after outputting the address. During external accesses in de-multiplexed bus modes PORT0 reads the incoming instruction or data word or outputs the data byte or word (see Figure 29).

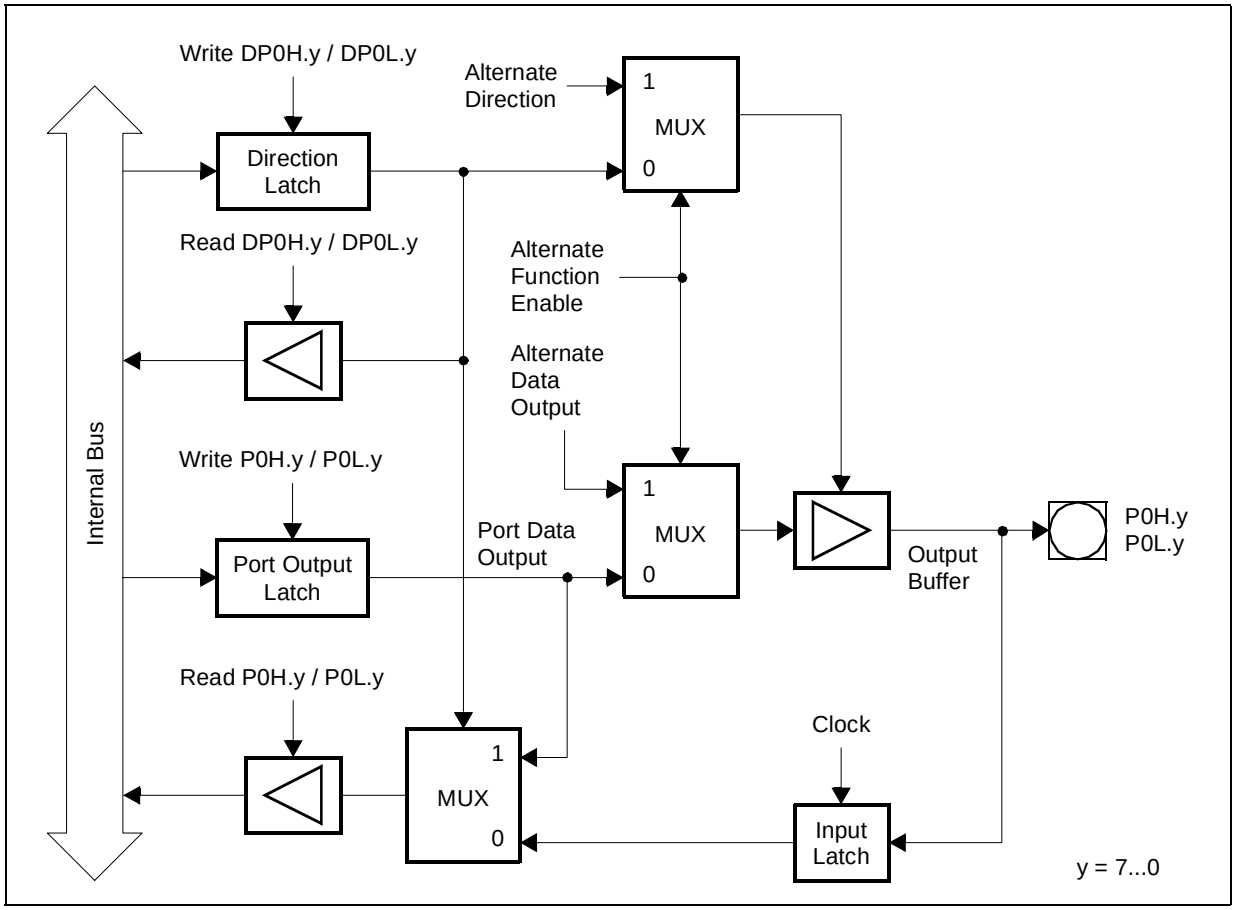
When an external bus mode is enabled, the direction of the port pin and the loading of data into the port output latch are controlled by the bus controller hardware. The input of the port output latch is disconnected from the internal bus and is switched to the line labeled "Alternate Data Output" via a multiplexer. The alternate data can be the 16-bit intra-segment address or the 8-/16-bit data information. The incoming data on PORT0 is read on the line "Alternate Data Input". While an external bus mode is enabled, the user software should not write to the port output latch, otherwise unpredictable results may occur. When the external bus modes are disabled, the contents of the direction register last written by the user becomes active.

Figure 29 : PORT0 I/O and alternate functions



The Figure 30 shows the structure of a PORT0 pin.

Figure 30 : Block diagram of a PORT0 pin



7.3 - Port1

The two 8-bit ports P1H and P1L represent the higher and lower part of PORT1, respectively. Both halves of PORT1 can be written (for example via a PEC transfer) without effecting the other half. If this port is used for general purpose I/O, the direction of each line can be configured via the corresponding direction registers DP1H and DP1L.

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

7.3.1 - Alternate Functions of PORT1

When a de-multiplexed external bus is enabled, PORT1 is used as address bus.
Note that de-multiplexed bus modes use PORT1 as a 16-bit port. Otherwise all 16 port lines can be used for general purpose I/O. The upper four pins of PORT1 (P1H.7...P1H.4) also are capture input lines for the CAPCOM2 unit (CC27-24 I/O).

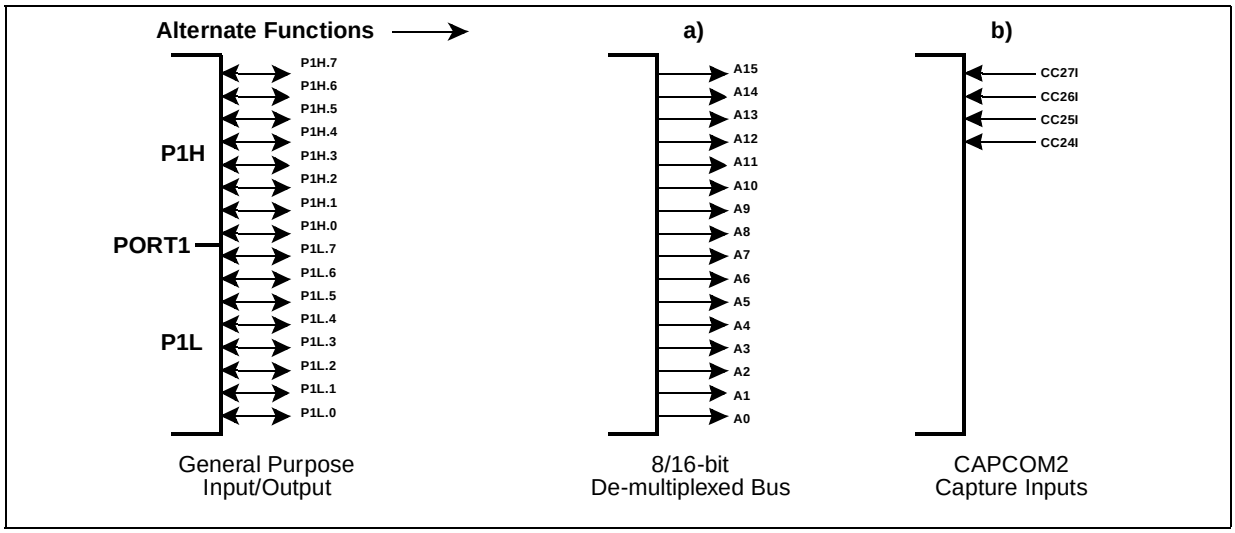
As all other capture inputs, the capture input function of pins P1H.7...P1H.4 can also be used as external interrupt inputs with a sample rate of 8 CPU clock cycles.

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

During external accesses in de-multiplexed 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 de-multiplexed bus mode, PORT1 is not used and is available for general purpose I/O.

Figure 31 : PORT1 I/O and alternate functions

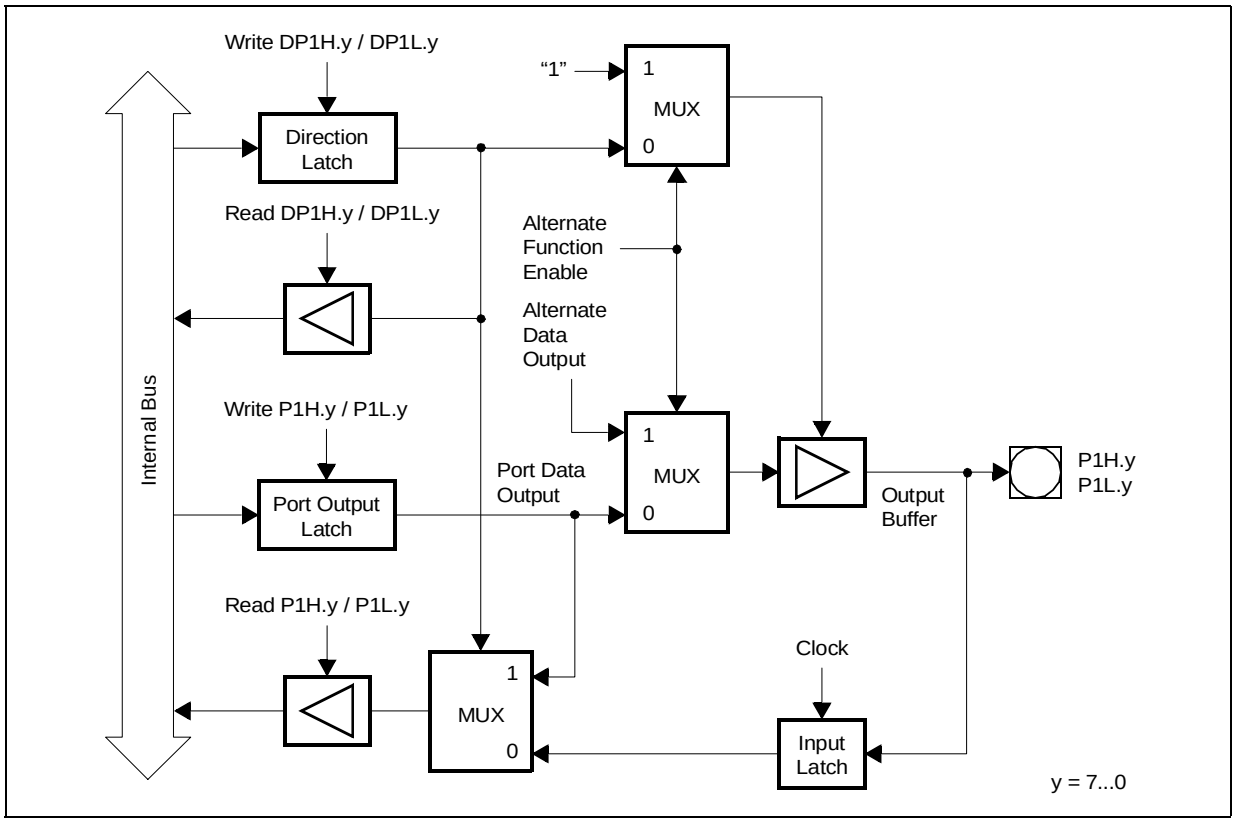


When an external bus mode is enabled, the direction of the port pin and the loading of data into the port output latch are controlled by the bus controller hardware. The input of the port output latch is disconnected from the internal bus and is switched to the line labeled "Alternate Data Output" via a multiplexer. The alternate data is the 16-bit intra-segment address.

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

The Figure 32 shows the structure of a PORT1 pin.

Figure 32 : Block diagram of a PORT1 pin



7.4 - Port2

If this 16-bit port 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.

P2 (FFC0h / E0h) SFR Reset Value: 0000h

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

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

DP2 (FFC2h / E1h) SFR Reset Value: 0000h

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

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

ODP2 (F1C2h / E1h) ESFR Reset Value: 0000h

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

Bit	Function
ODP2.y	**Port2 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

7.4.1 - Alternate Functions of Port2

All Port2 lines (P2.15...P2.0) can be configured capture inputs or compare outputs (CC15IO...CC0IO) for the CAPCOM1 unit.

When a Port2 line is used as a capture input, the state of the input latch, which represents the state of the port pin, is directed to the CAPCOM unit via the line "Alternate Pin Data Input". If an external capture trigger signal is used, the direction of the respective pin must be set to input. If the direction is set to output, the state of the port output latch will be read since the pin represents the state of the output latch. This can be used to trigger a capture event through software by setting or clearing the port latch. Note that in the output configuration, no external device may drive the pin, otherwise conflicts would occur.

When a Port2 line is used as a compare output (compare modes 1 and 3), the compare event (or the timer overflow in compare mode 3) directly effects the port output latch. In compare mode 1, when a valid compare match occurs, the state of the port output latch is read by the CAPCOM control hardware via the line "Alternate Latch Data Input", inverted, and written back to the latch via the line "Alternate Data Output". The port output latch is clocked by the signal "Compare Trigger" which is generated by the CAPCOM unit. In compare mode 3, when a match occurs, the value '1' is written to the port output latch via the line "Alternate Data Output". When an overflow of the corresponding timer occurs, a '0' is written to the port output latch. In both cases, the output latch is clocked by the signal "Compare Trigger". The direction of the pin should be set to output by the user, otherwise the pin will be in the high-impedance state and will not reflect the state of the output latch.

As can be seen from the port structure (Figure 34), the user software always has free access to the port pin even when it is used as a compare output. This is useful for setting up the initial level of the pin when using compare mode 1 or the double-register mode. In these modes, unlike in compare mode 3, the pin is not set to a specific value when a compare match occurs, but is toggled instead.

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

As all other capture inputs, the capture input function of pins P2.17...P2.0 can also be used as external interrupt inputs with a sample rate of 8 CPU clock cycles.

For pins 2.15 to 2.8, the sampling rate is 8 CPU clock cycle. When used as capture input and 1 CPU clock cycle if used as fast external input.

7.4.2 - External Interrupts

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

Fast external interrupts may also have interrupt sources selected from other peripherals; for example the CANx controller receive signal (CANx_RxD) can be used to interrupt the system. This new function is controlled using the 'External Interrupt Source Selection' register EXISEL.

EXISEL (F1DAh / EDh) ESFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXI7SS		EXI6SS		EXI5SS		EXI4SS		EXI3SS		EXI2SS		EXI1SS		EXI0SS	
RW		RW		RW		RW		RW		RW		RW		RW	

Bit	Function
EXIxSS	**External Interrupt x Source Selection (x=7...0)** '00': Input from associated Port 2 pin. '01': Input from "alternate source". '10': Input from Port 2 pin ORed with "alternate source". '11': Input from Port 2 pin ANDed with "alternate source".

EXIxSS	Port 2 pin	Alternate Source
0	P2.8	CAN1_RxD
1	P2.9	CAN2_RxD
2...7	P2.10...15	Not used (zero)

The upper eight Port2 lines (P2.15...P2.8) also support Fast External Interrupt inputs (EX7IN...EX0IN).

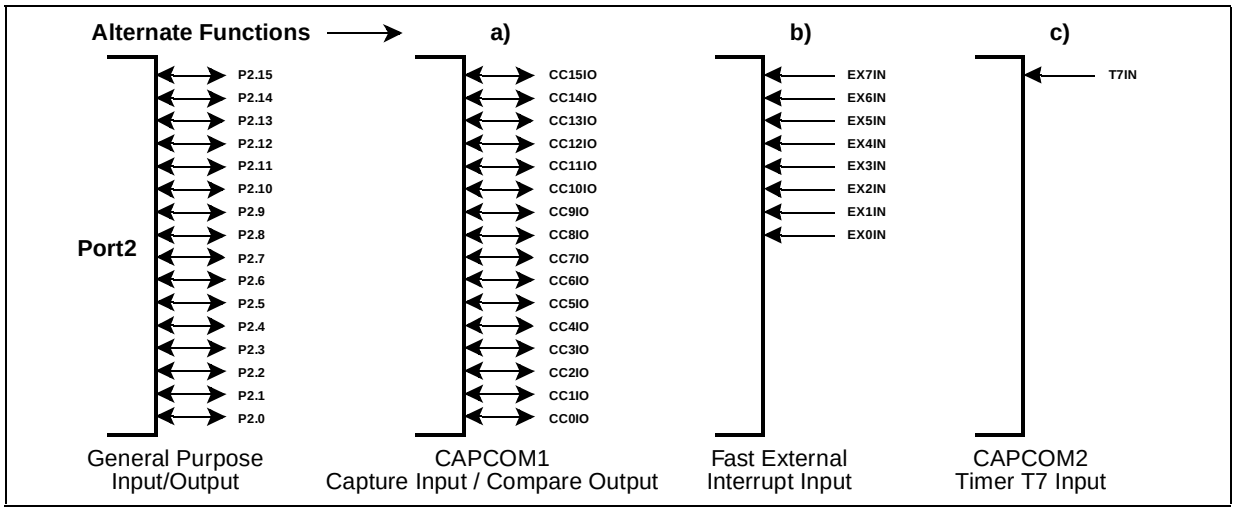
P2.15 in addition is the input for CAPCOM2 timer T7 (T7IN).

The Table 19 summarizes the alternate functions of Port2.

Table 19 : Port2 alternate functions

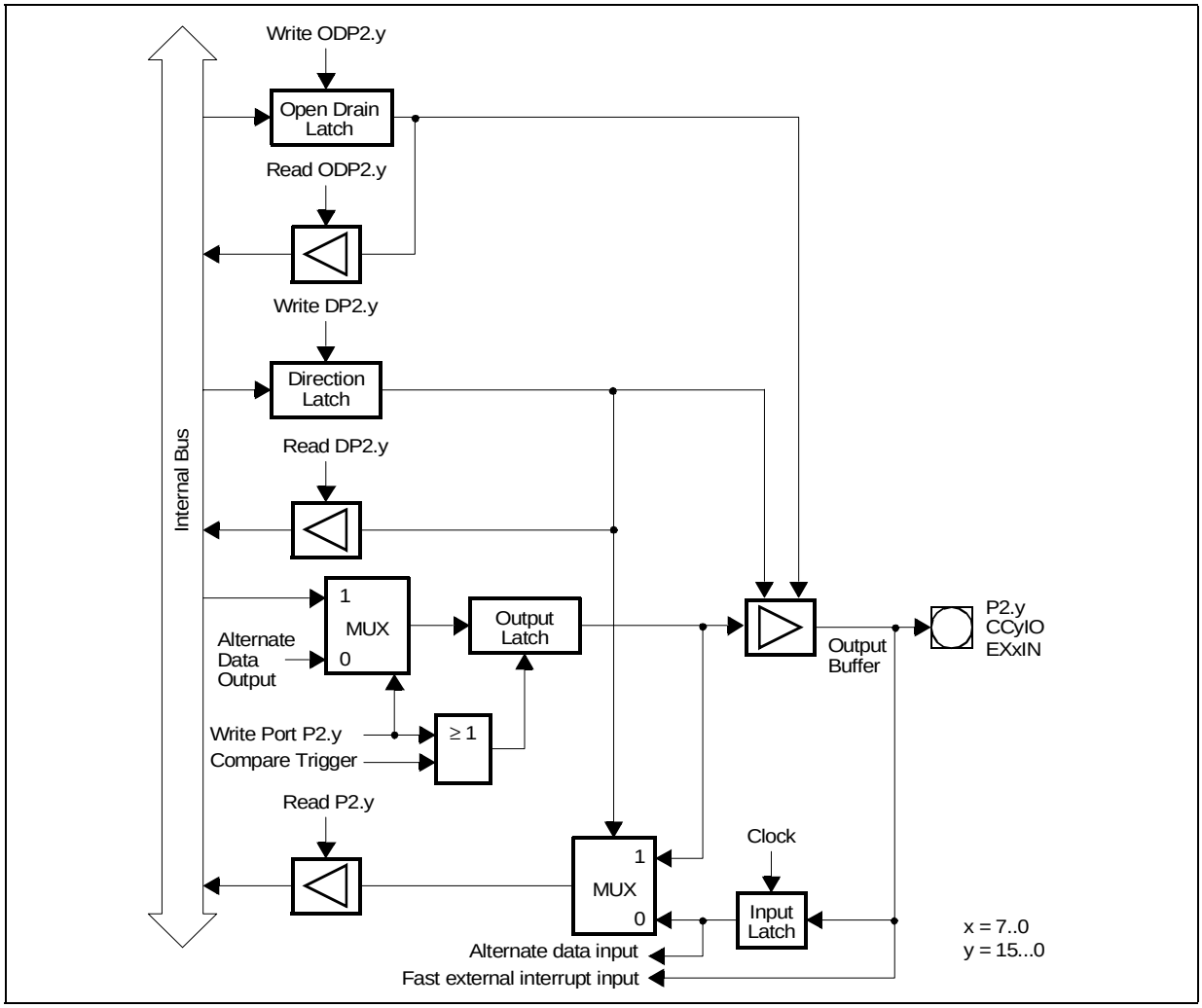
P.2 Pin	Alt Function a)	Alternate Function b)		Alternate Function c)
P2.0	CC0IO	-		-
P2.1	CC1IO	-		-
P2.2	CC2IO	-		-
P2.3	CC3IO	-		-
P2.4	CC4IO	-		-
P2.5	CC5IO	-		-
P2.6	CC6IO	-		-
P2.7	CC7IO	-		-
P2.8	CC8IO	EX0IN	Fast External Interrupt 0 Input	-
P2.9	CC9IO	EX1IN	Fast External Interrupt 1 Input	-
P2.10	CC10IO	EX2IN	Fast External Interrupt 2 Input	-
P2.11	CC11IO	EX3IN	Fast External Interrupt 3 Input	-
P2.12	CC12IO	EX4IN	Fast External Interrupt 4 Input	-
P2.13	CC13IO	EX5IN	Fast External Interrupt 5 Input	-
P2.14	CC14IO	EX6IN	Fast External Interrupt 6 Input	-
P2.15	CC15IO	EX7IN	Fast External Interrupt 7 Input	T7IN Timer T7 External Count Input

Figure 33 : Port2 I/O and alternate functions



The pins of Port2 combine internal capture input bus data with compare output alternate data output before the port latch input.

Figure 34 : Block diagram of a Port2 pin



7.5 - Port3

If this 15-bit port is used for general purpose I/O, the direction of each line can be configured by the corresponding direction register DP3. Most port lines can be switched into push-pull or open-drain mode by 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**

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

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

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

ODP3 (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	**Port3 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

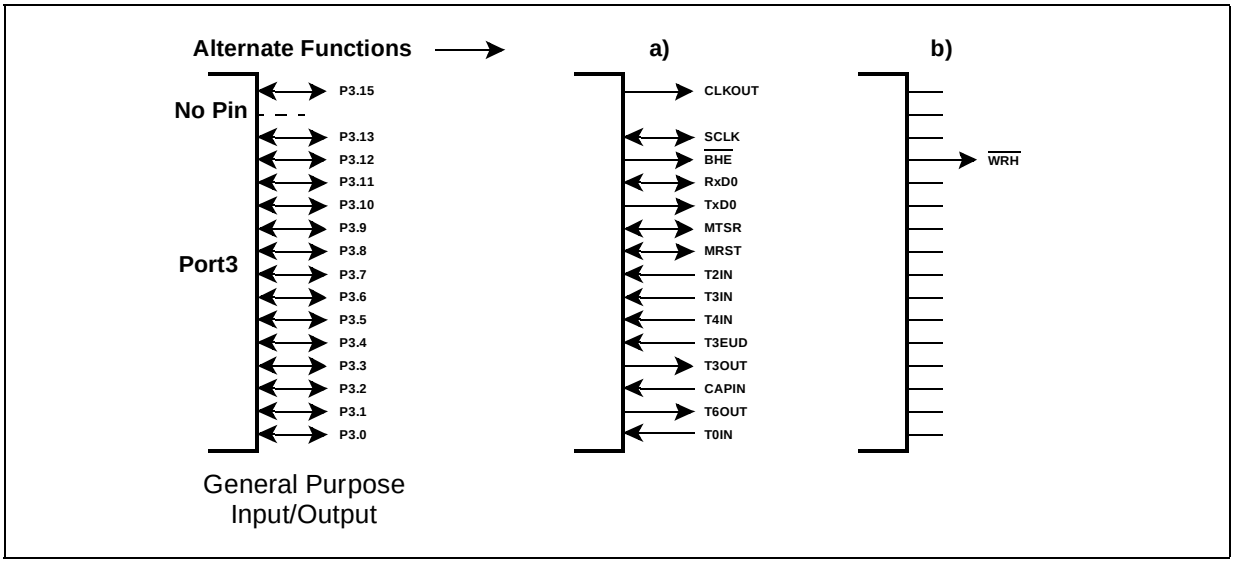
7.5.1 - Alternate Functions of Port3

The pins of Port3 are used for various functions which include external timer control lines, the two serial interfaces and the control lines BHE / WRH and CLKOUT.

Table 20 : Port3 alternative functions

Port3 Pin	Alternate Function	
P3.0	T0IN	CAPCOM1 Timer 0 Count Input
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 Input
P3.5	T4IN	Timer 4 Count Input
P3.6	T3IN	Timer 3 Count Input
P3.7	T2IN	Timer 2 Count Input
P3.8	MRST	SSC Master Receive / Slave Transmit
P3.9	MTSR	SSC Master Transmit / Slave Receive
P3.10	TxD0	ASC0 Transmit Data Output
P3.11	RxD0	ASC0 Receive Data Input (/ Output in synchronous mode)
P3.12	$\overline{BHE}$/$\overline{WRH}$	Byte High Enable / Write High Output
P3.13	SCLK	SSC Shift Clock Input/Output
P3.14	---	No pin assigned
P3.15	CLKOUT	System Clock Output

Figure 35 : Port3 I/O and alternate functions



The structure of the Port3 pins depends on their alternate function (see Figure 36).

When the on-chip peripheral associated with a Port3 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". Port3 pins with alternate input functions are:

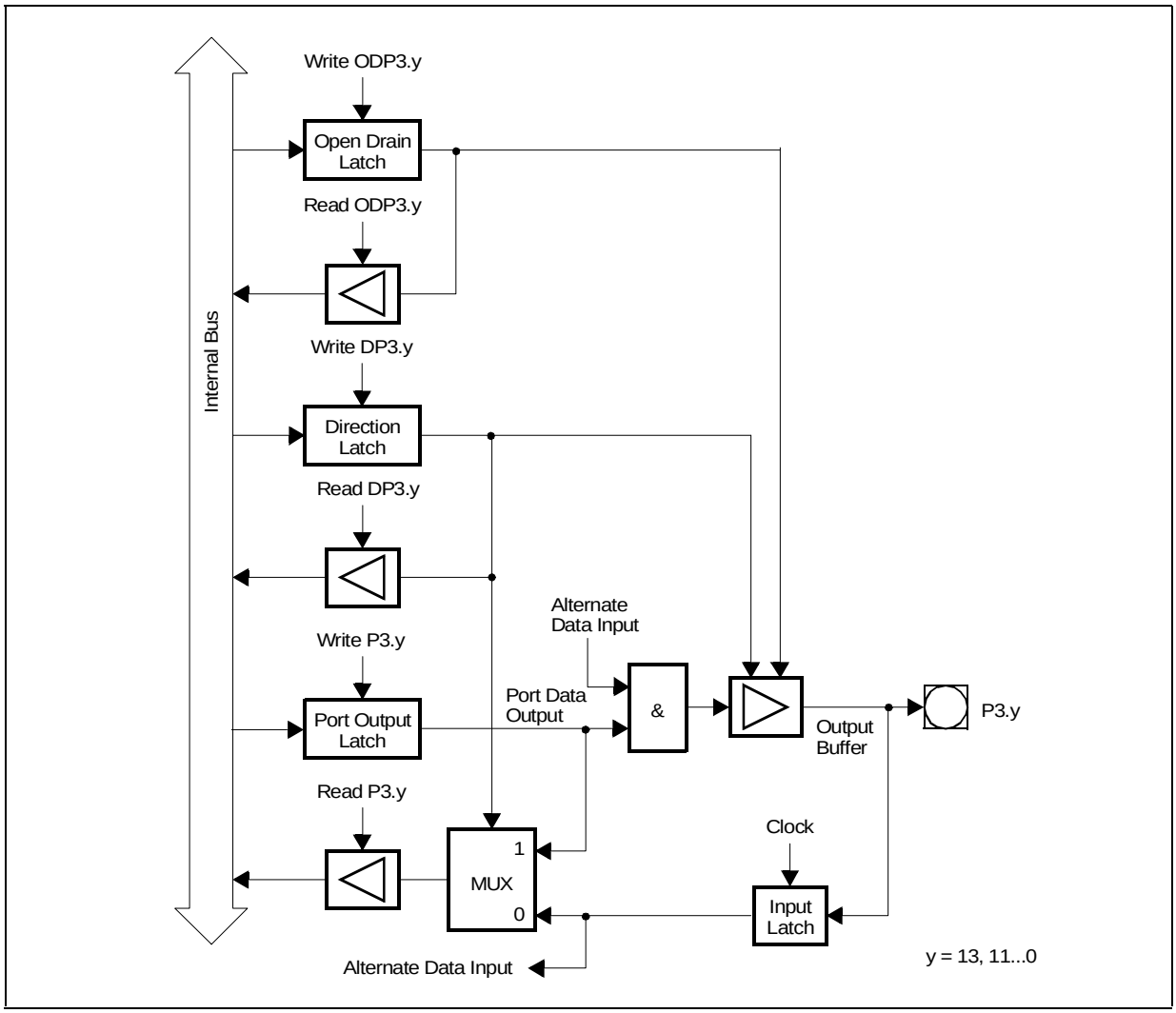
T0IN, T2IN, T3IN, T4IN, T3EUD and CAPIN.

When the on-chip peripheral associated with a Port3 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'). Port3 pins with alternate output functions are: T6OUT, T3OUT, TxD0, BHE and CLKOUT.

When the on-chip peripheral associated with a Port3 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. Port3 pins with alternate input/output functions are: MTSR, MRST, RxD0 and SCLK.

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

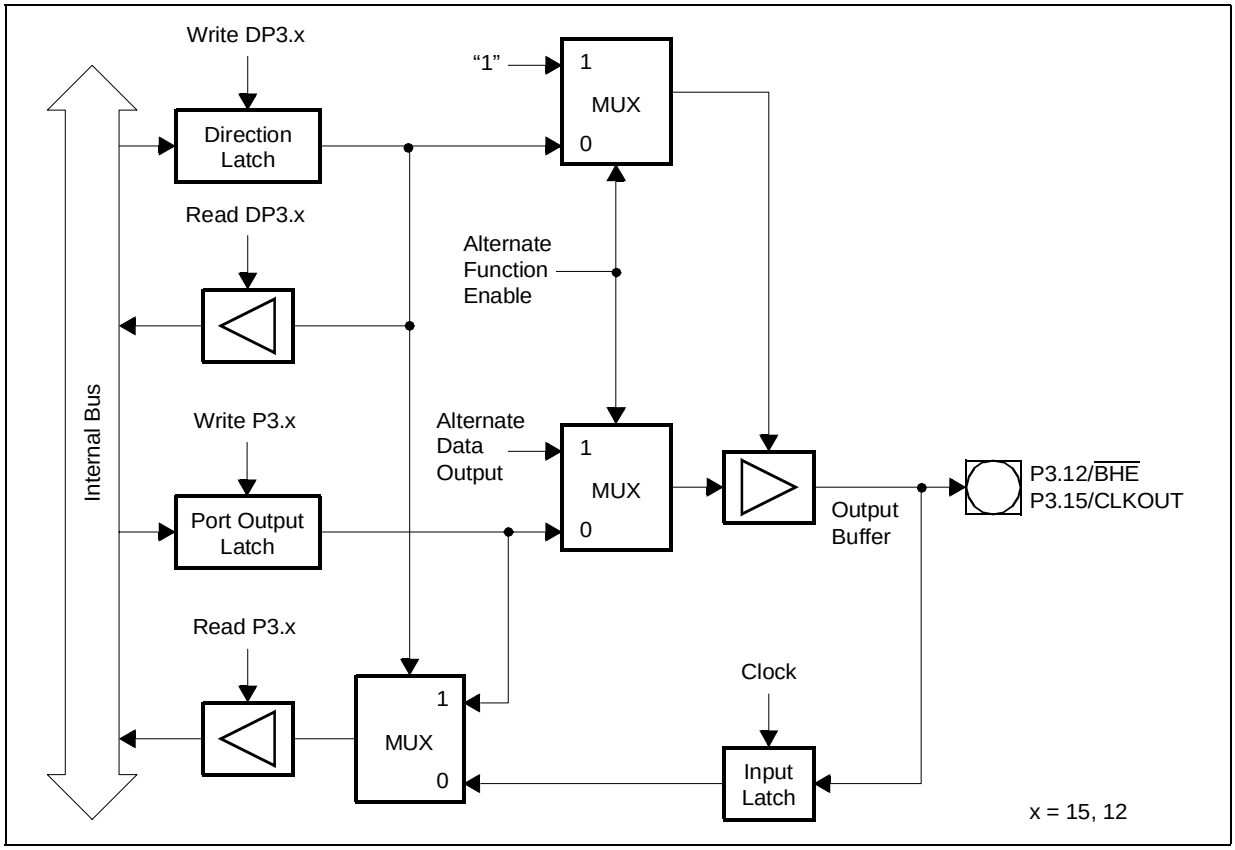
Figure 36 : Block diagram of Port3 pins



Pin P3.12 (BHE/WRH) is another pin with an alternate output function, however, its structure is slightly different (see Figure 37). After reset the BHE or WRH function must be used depending on the system start-up configuration. In either of these cases, there is no possibility to program any port latches before. Thus, the appropriate alternate function is selected automatically. If BHE/WRH is not used in the system, this pin can be used for general purpose I/O by disabling the alternate function (BYTDIS = '1' / WRCFG='0').

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

Figure 37 : Block diagram of P3.15 (CLKOUT) and P3.12 (BHE/WRH) pins



7.6 - Port4

If this 8-bit port is used for general purpose I/O, the direction of each line can be configured via the corresponding direction register DP4.

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

ODP4 (F1CAh / E5h) ESFR Reset Value: - - 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	ODP4.7	ODP4.6	-	-	-	-	-	-
								RW	RW						

Bit	Function
ODP4.y	**Port 4 Open drain control register bit y** ODP4.y = 0: Port line P4.y output driver in push/pull mode ODP4.y = 1: Port line P4.y output driver in open drain mode if P4.y is not a segment address line output

Only bit 6 and 7 are implemented, all other bit will be read as "0".

7.6.1 - Alternate Functions of Port4

During external bus cycles that use segmentation (for address space above 64 Kbytes) a number of Port4 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 Port4 (if any) may be used for general purpose I/O. If segment address lines are selected, the alternate function of Port4 may be necessary to access for external memory directly after reset. For this reason Port4 will be switched to this alternate function automatically.

The number of segment address lines is selected via PORT0 during reset. The selected value can be read from bit-field SALSEL in register RP0H (read only) in order to check the configuration during run time.

The CAN interfaces use 2 or 4 pins of Port4 to interface the CAN modules to the external CAN transceiver. In this case the number of possible segment address lines is reduced.

The Table 21 summarizes the alternate functions of Port4 depending on the number of selected segment address lines (coded via bit-field SALSEL).

Table 21 : Port4 alternate functions

Port 4	Standard Function SALSEL = 01 64 Kbytes	Alternate Function SALSEL = 11 256 Kbytes	Alternate Function SALSEL = 00 1 Mbyte	Alternate Function SALSEL = 10 16 Mbytes
P4.0	GPIO	Segment Address A16	Segment. Address A16	Segment Address A16
P4.1	GPIO	Segment Address A17	Segment Address A17	Segment Address A17
P4.2	GPIO	GPIO	Segment Address A18	Segment Address A18
P4.3	GPIO	GPIO	Segment Address A19	Segment Address A19
P4.4	GPIO/CAN2_RxD	GPIO/CAN2_RxD	GPIO/CAN2_RxD	Segment Address A20
P4.5	GPIO/CAN1_RxD	GPIO/CAN1_RxD	GPIO/CAN1_RxD	Segment Address A21
P4.6	GPIO/CAN1_TxD	GPIO/CAN1_TxD	GPIO/CAN1_TxD	Segment Address A22
P4.7	GPIO/CAN2_TxD	GPIO/CAN2_TxD	GPIO/CAN2_TxD	Segment Address A23

Figure 38 : Port4 I/O and alternate functions

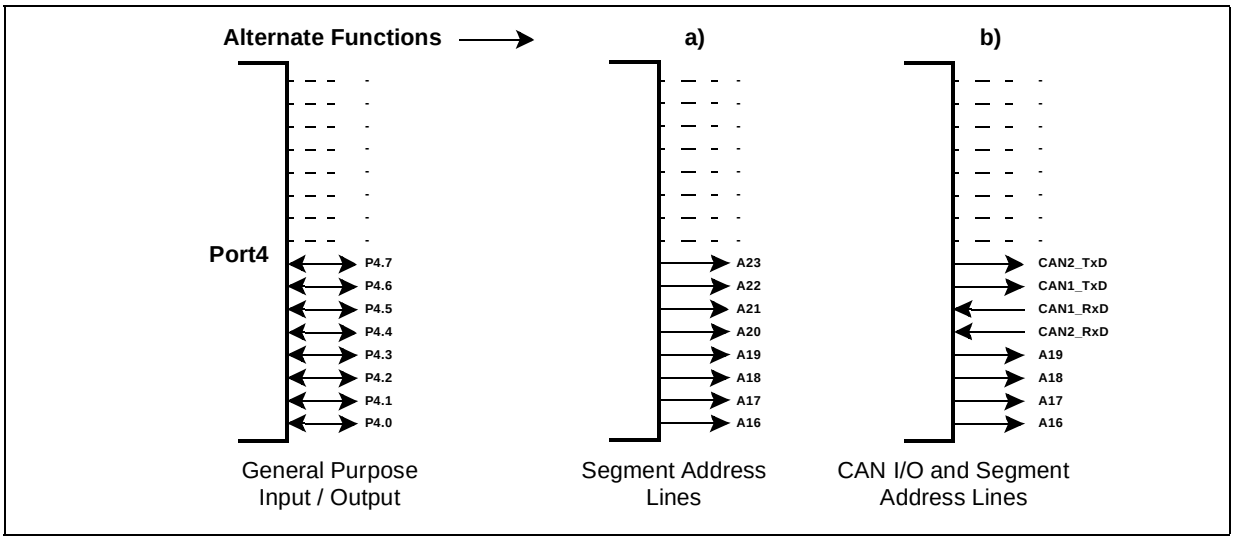


Figure 39 : Block diagram of a Port4 pin

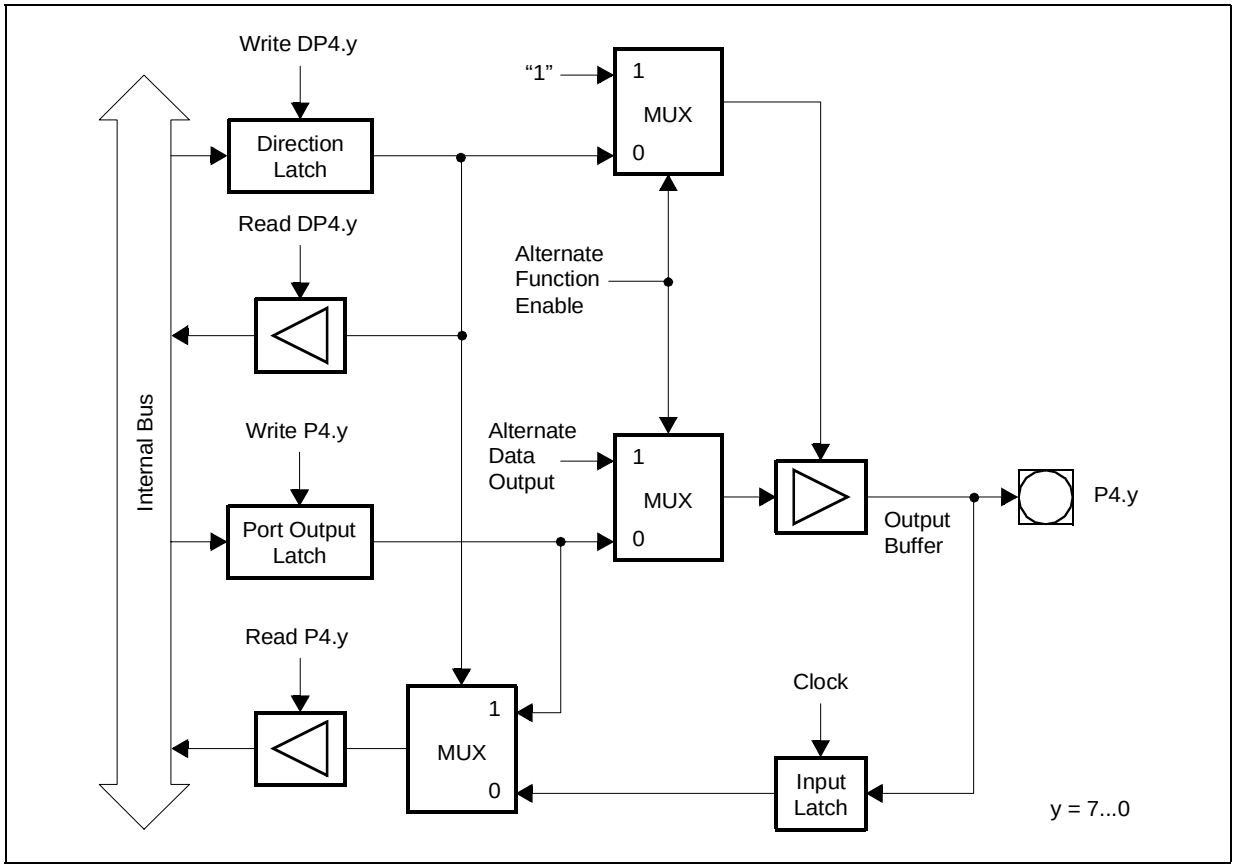


Figure 40 : Block diagram of P4.4 and P4.5 pins

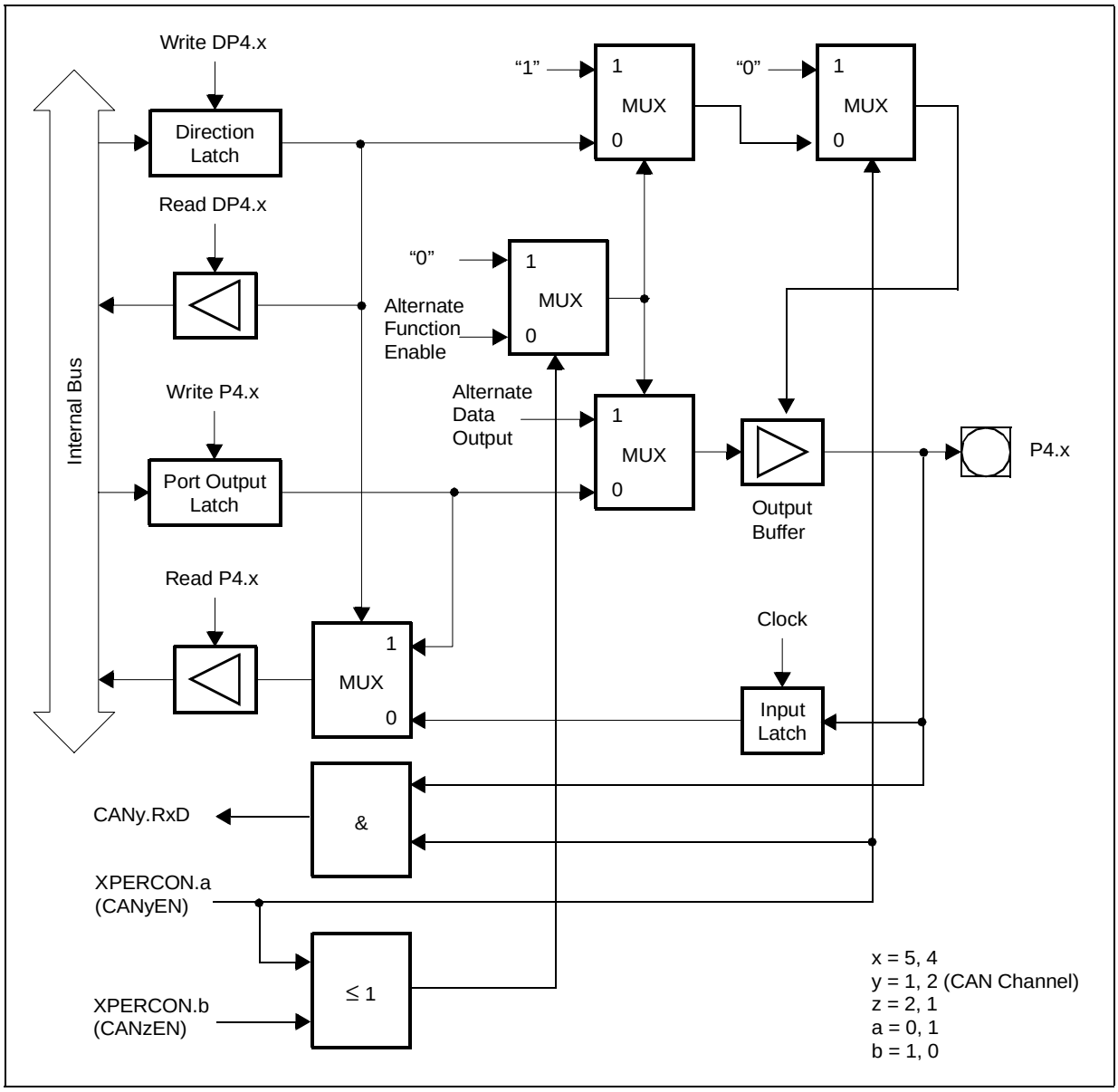
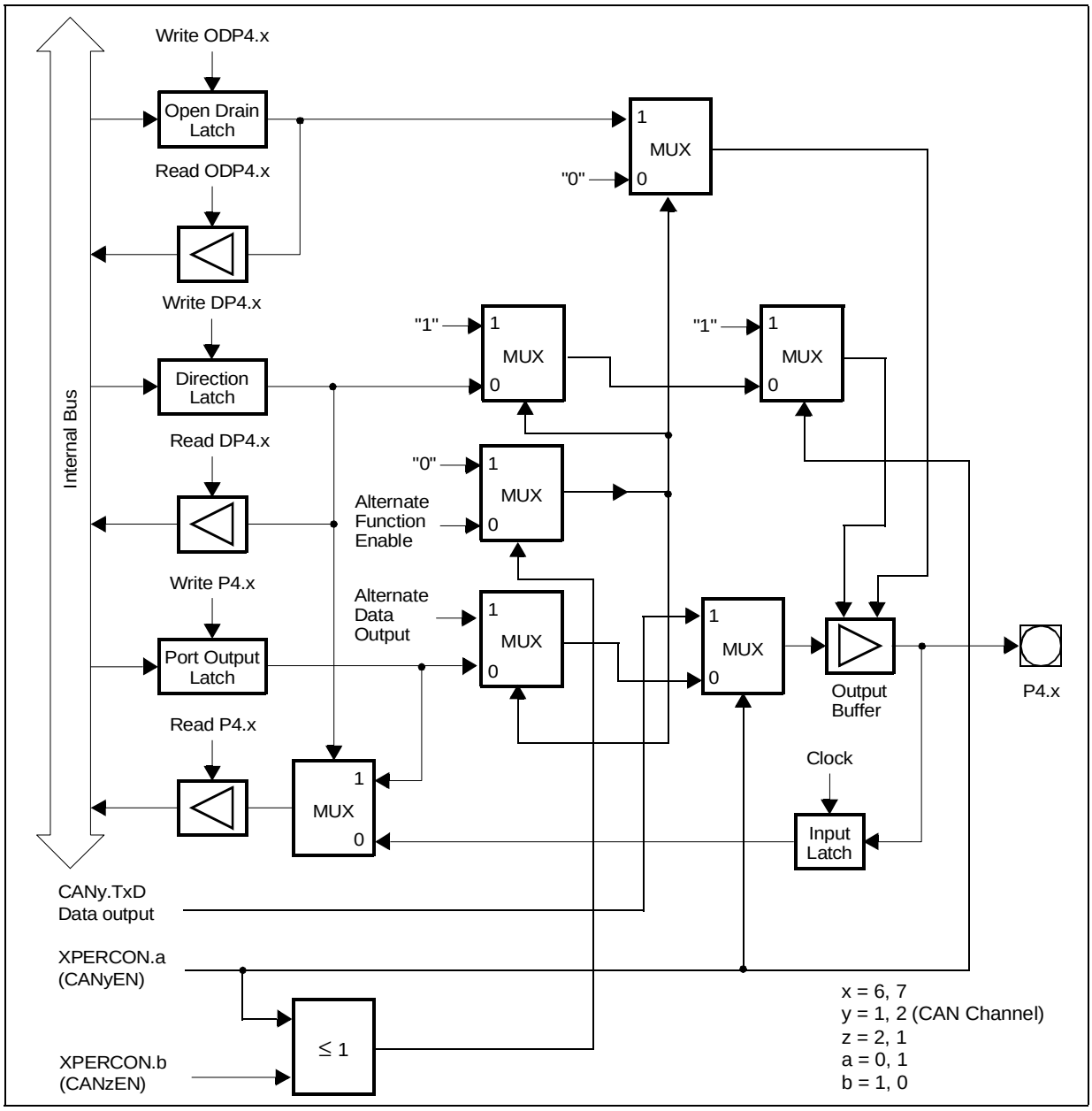


Figure 41 : Block diagram of P4.6 and P4.7 pins



7.7 - Port5

This 16-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: XXXXh

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	P5.9	P5.8	P5.7	P5.6	P5.5	P5.4	P5.3	P5.2	P5.1	P5.0
R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

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

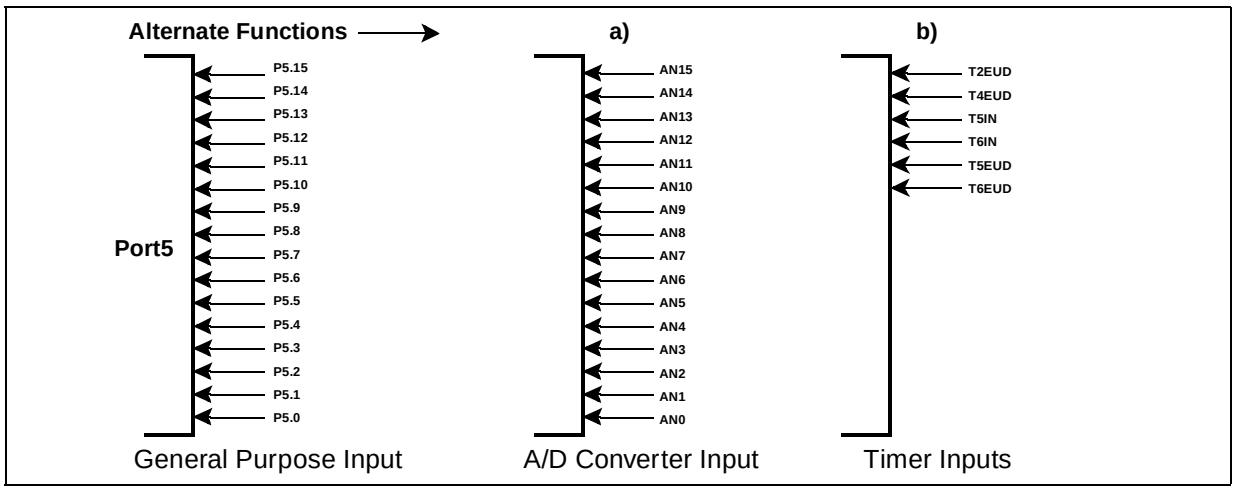
7.7.1 - Alternate Functions of Port5

Each line of Port5 is also connected to the input multiplexer of the Analog/Digital Converter. All port lines (P5.15...P5.0) can accept analog signals (AN15...AN0) that can be converted by the ADC. No special programming is required for pins that shall be used as analog inputs. The upper 6 pins of Port5 also serve as external timer control lines for GPT1 and GPT2. The Table 22 summarizes the alternate functions of Port5.

Table 22 : Port5 alternate functions

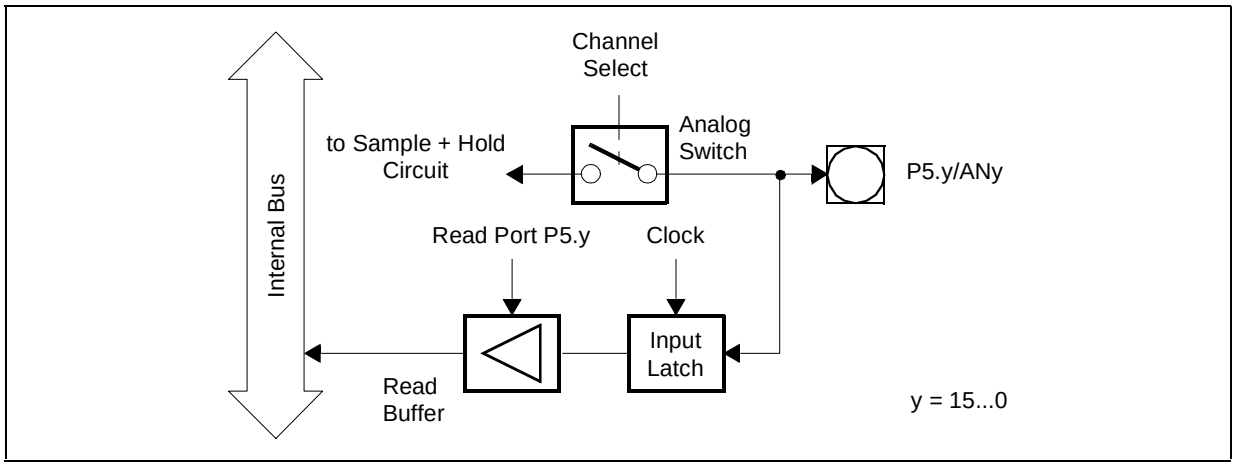
Port5 Pin	Alternate Function a)	Alternate Function b)
P5.0	Analog Input AN0	-
P5.1	Analog Input AN1	-
P5.2	Analog Input AN2	-
P5.3	Analog Input AN3	-
P5.4	Analog Input AN4	-
P5.5	Analog Input AN5	-
P5.6	Analog Input AN6	-
P5.7	Analog Input AN7	-
P5.8	Analog Input AN8	-
P5.9	Analog Input AN9	-
P5.10	Analog Input AN10	T6EUD Timer 6 external Up/Down Input
P5.11	Analog Input AN11	T5EUD Timer 5 external Up/Down Input
P5.12	Analog Input AN12	T6IN Timer 6 Count Input
P5.13	Analog Input AN13	T5IN Timer 5 Count Input
P5.14	Analog Input AN14	T4EUD Timer 4 external Up/Down Input
P5.15	Analog Input AN15	T2EUD Timer 2 external Up/Down Input

Figure 42 : Port5 I/O and alternate functions



Port5 pins have a special port structure (see Figure 43), first because it is an input only port, and second because the analog input channels are directly connected to the pins rather than to the input latches.

Figure 43 : Block diagram of a Port5 pin



7.7.2 - Port 5 Schmitt Trigger Analog Inputs

A Schmitt trigger protection can be activated on each pin of Port 5 by setting the dedicated bit of register P5DIDIS.

P5DIDIS (FFA4h / D2h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
P5DI DIS.15	P5DI DIS.14	P5DI DIS.13	P5DI DIS.12	P5DI DIS.11	P5DI DIS.10	P5DI DIS.9	P5DI DIS.8	P5DI DIS.7	P5DI DIS.6	P5DI DIS.5	P5DI DIS.4	P5DI DIS.3	P5DI DIS.2	P5DI DIS.1	P5DI DIS.0
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
P5DIDIS.y	**Port 5 Digital Disablel register bit y** P5DIDIS.y = 0: Port line P5.y digital input is enabled (Schmitt trigger enabled) P5DIDIS.y = 1: Port line P5.y digital input is disabled (Schmitt trigger disabled, necessary for input leakage current reduction)

7.8 - Port6

If this 8-bit port is used for general purpose I/O, the direction of each line can be configured via the corresponding direction register DP6. Each port line can be switched into push-pull or open-drain mode via the open-drain control register ODP6.

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	**Port6 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

7.8.1 - Alternate Functions of Port6

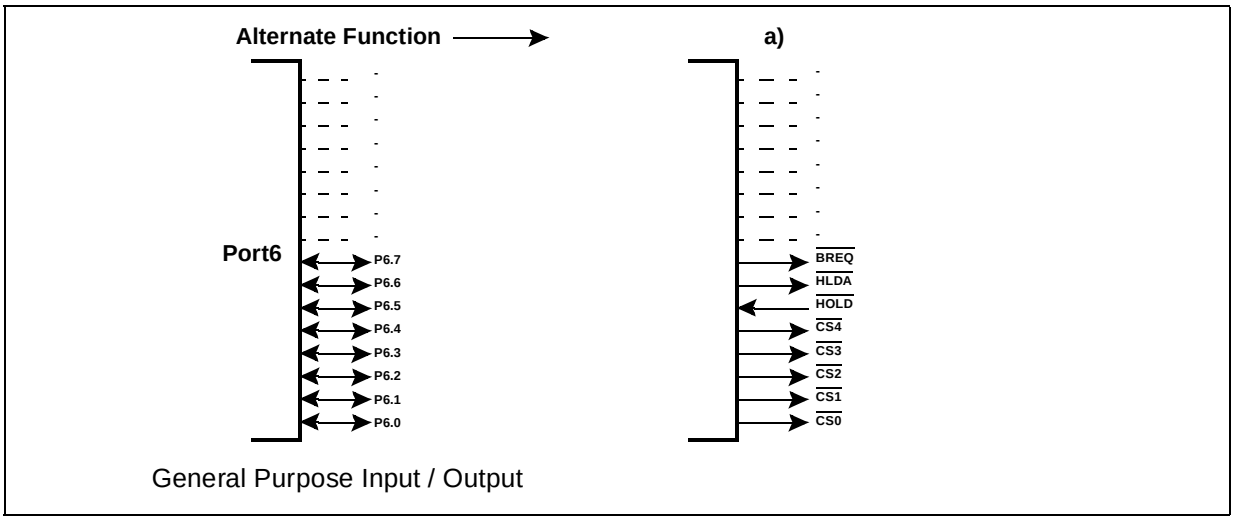
A programmable number of chip select signals (CS4...CS0) derived from the bus control registers (BUSCON4...BUSCON0) can be output on 5 pins of Port6. The other 3 pins may be used for bus arbitration to accommodate additional masters in a ST10F280 system.
The number of chip select signals is selected via PORT0 during reset. The selected value can be read from bit-field CSSEL in register RP0H (read only) in order to check the configuration during run time.

The Table 23 summarizes the alternate functions of Port6 depending on the number of selected chip select lines (coded via bit-field CSSEL).

Table 23 : Port6 alternate functions

Port6 Pin	Alternate Function CSSEL = 10	Alternate Function CSSEL = 01	Alternate Function CSSEL = 00	Alternate Function CSSEL = 11
P6.0	General purpose I/O	Chip select $\overline{CS0}$	Chip select $\overline{CS0}$	Chip select $\overline{CS0}$
P6.1	General purpose I/O	Chip select $\overline{CS1}$	Chip select $\overline{CS1}$	Chip select $\overline{CS1}$
P6.2	General purpose I/O	General purpose I/O	Chip select $\overline{CS2}$	Chip select $\overline{CS2}$
P6.3	General purpose I/O	General purpose I/O	General purpose I/O	Chip select $\overline{CS3}$
P6.4	General purpose I/O	General purpose I/O	General 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 : Port6 I/O and alternate functions



The chip select lines of Port6 have an internal weak pull-up device. This device is switched on under the following conditions:

– Always during reset

– If the Port6 line is used as a chip select output, and the ST10F280 is in Hold mode (invoked through 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 (AND-wired), while the ST10F280 is in Hold mode.

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

After reset the 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 (CS) is selected automatically in this case.

Note The open drain output option can only be selected via software earliest during the initialization routine; at least signal CS0 will be in push-pull output driver mode directly after reset (see Figure 45).

The bus arbitration signals HOLD, HLDA and 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 HLDA and BREQ are automatically enabled, while the pin driver for HOLD is automatically disabled (see Figure 46).

Figure 45 : Block diagram of Port6 pins

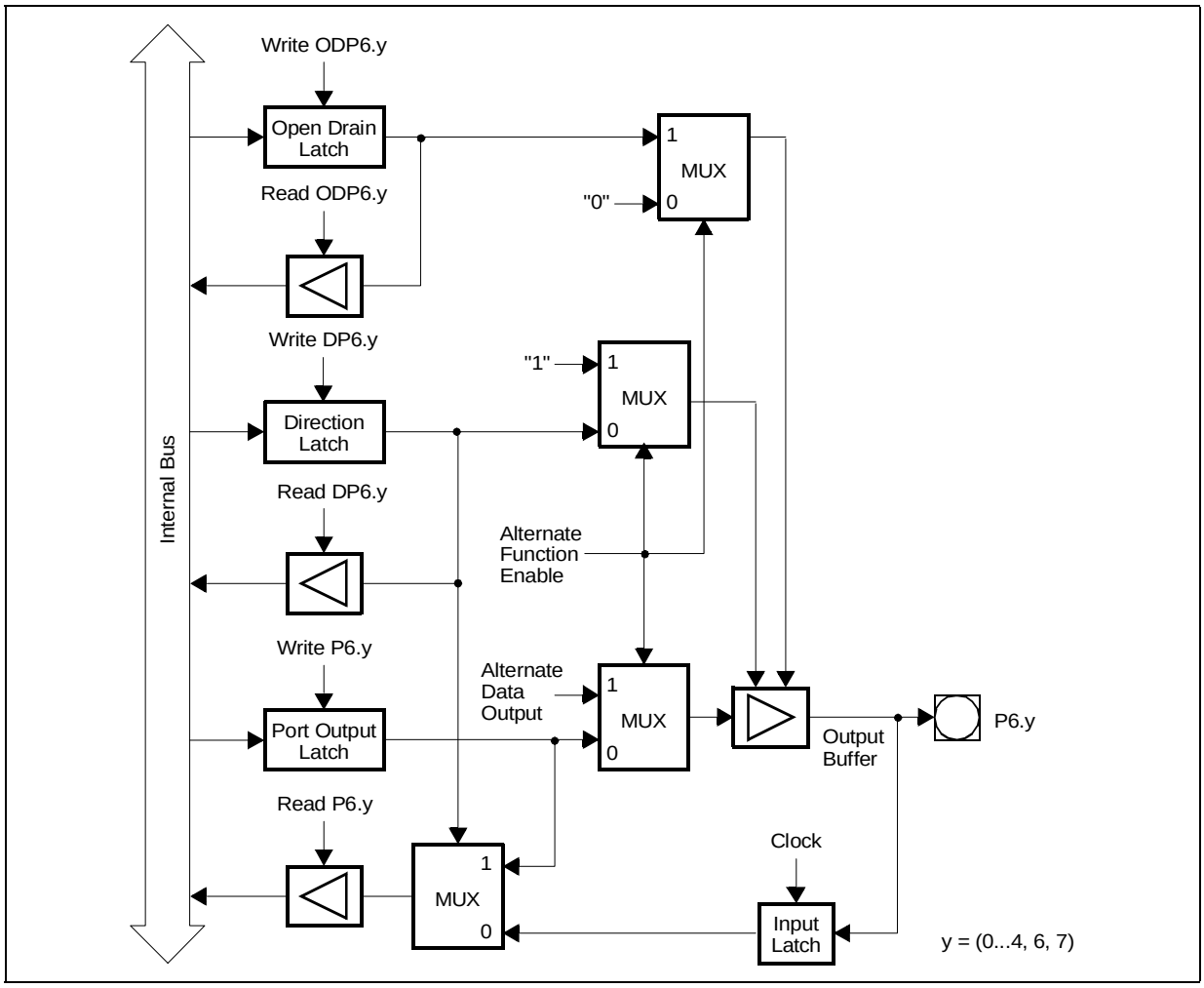
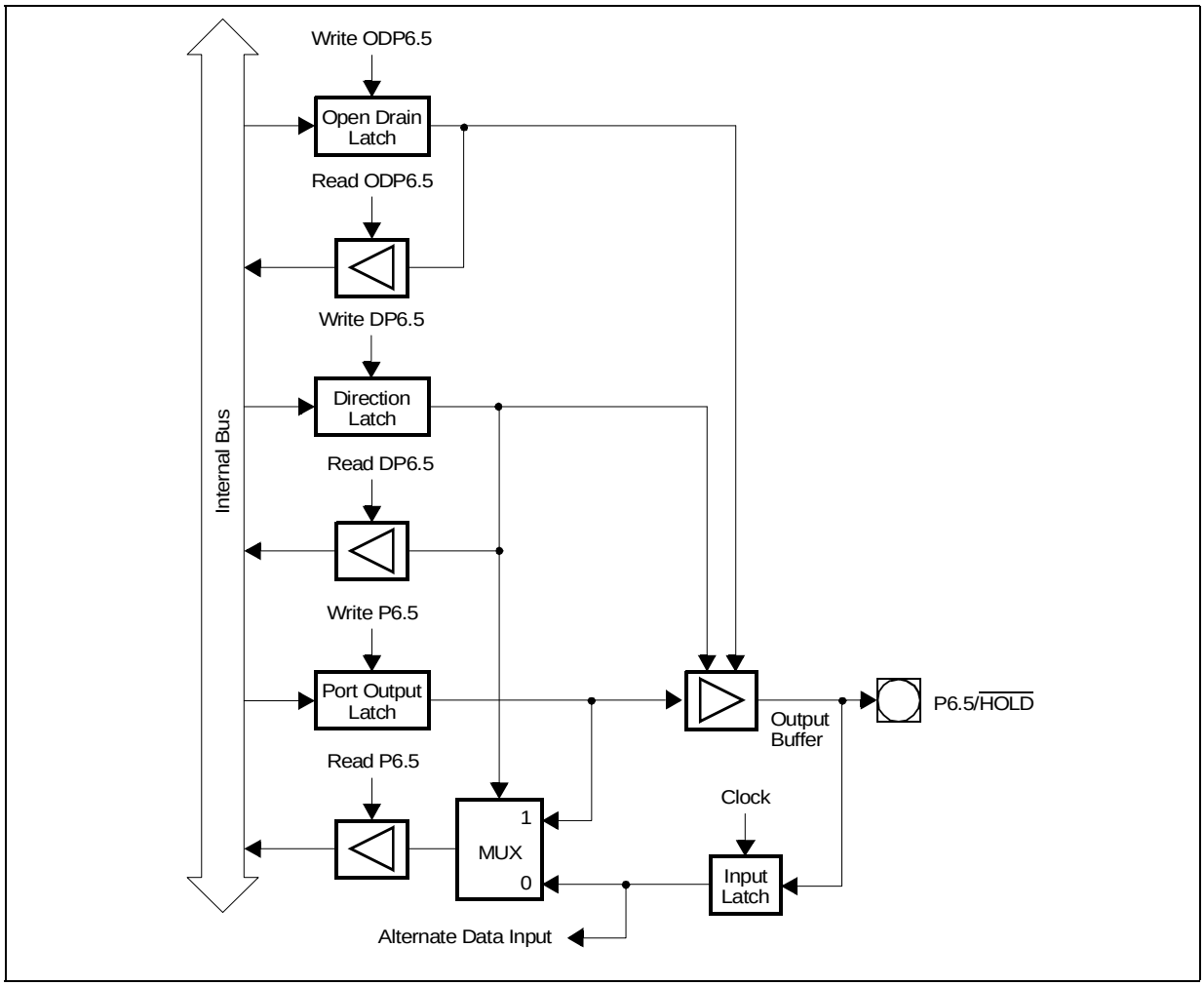


Figure 46 : Block diagram of P6.5 pin ($\overline{\text{HOLD}}$)



7.9 - Port7

If this 8-bit port 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: - - 00h

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

Bit	Function
P7.y	**Port data register P7 bit y**

DP7 (FFD2h / E9h) SFR Reset Value: - - 00h

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

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: - - 00h

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

Bit	Function
ODP7.y	**Port7 Open 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

7.9.1 - Alternate Functions of Port7

The upper 4 lines of Port7 (P7.7...P7.4) are used as capture inputs or compare outputs (CC31IO...CC28IO) for the CAPCOM2 unit.

How CAPCOM2 unit is connected to Port7 lines and how to handle them by software is similar to the Port2 lines description.

As all other capture inputs, the capture input function of pins P7.7...P7.4 can also be used as external interrupt inputs with a sample rate of 8 CPU clock cycles.

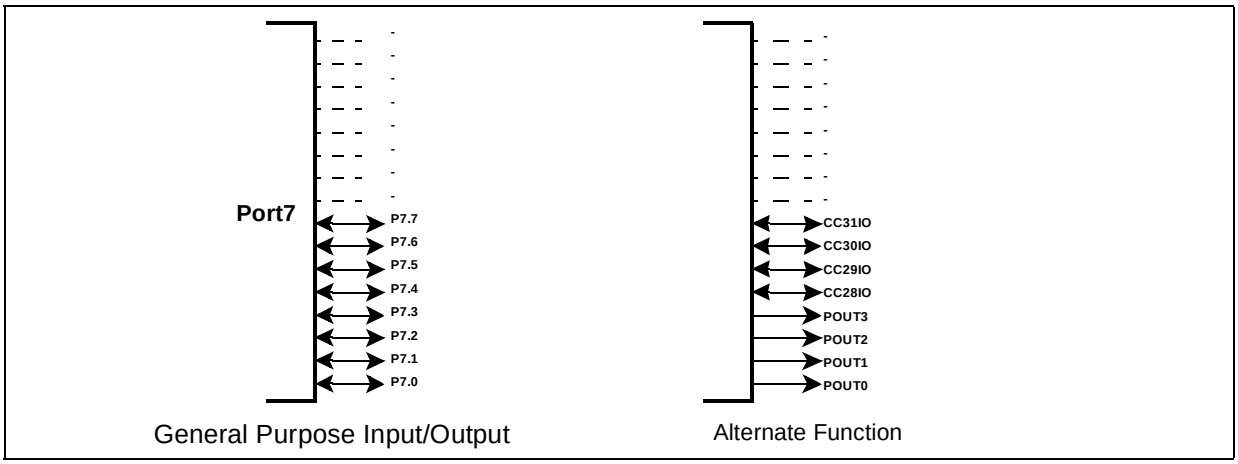
The lower 4 lines of Port7 (P7.3...P7.0) supports outputs of the PWM module (POUT3...POUT0). At these pins the value of the respective port output latch is XORed with the value of the PWM output rather than ANDed, as the other pins do. This allows to use the alternate output value either as it is (port latch holds a '0') or invert its level at the pin (port latch holds a '1').
Note that the PWM outputs must be enabled via the respective PENx bit in PWMCON1.

The Table 24 summarizes the alternate functions of Port7.

Table 24 : Port7 alternate functions

Port7 Pin	Alternate Function	
P7.0	POUT0	PWM mode channel 0 output
P7.1	POUT1	PWM mode channel 1 output
P7.2	POUT2	PWM mode channel 2 output
P7.3	POUT3	PWM mode channel 3 output
P7.4	CC28 I/O	Capture input / compare output channel 28
P7.5	CC29 I/O	Capture input / compare output channel 29
P7.6	CC30 I/O	Capture input / compare output channel 30
P7.7	CC31 I/O	Capture input / compare output channel 31

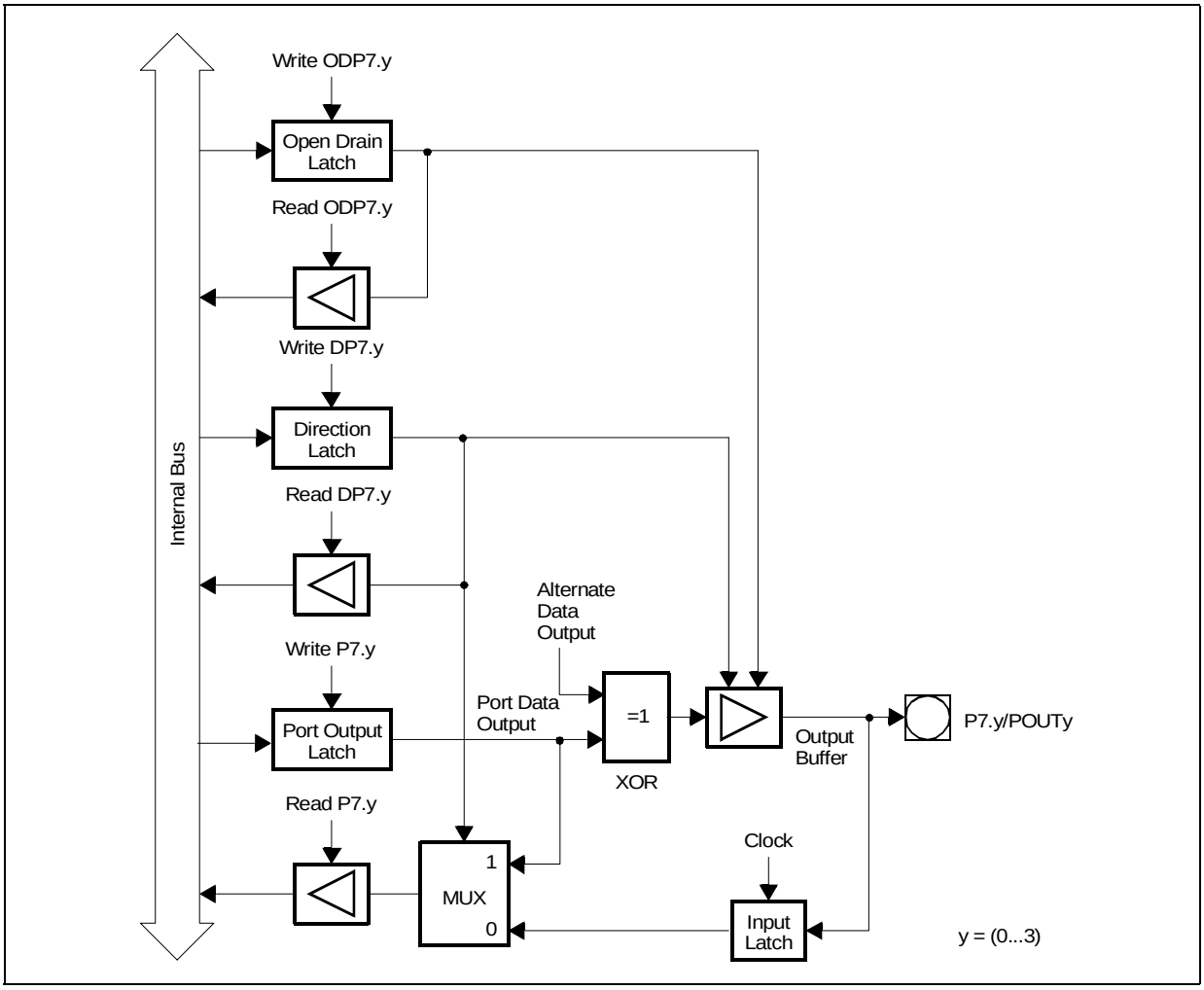
Figure 47 : Port7 I/O and alternate functions



The structure of Port7 differ in the way the output latches are connected to the internal bus and to the pin driver (see Figure 48 and Figure 49).

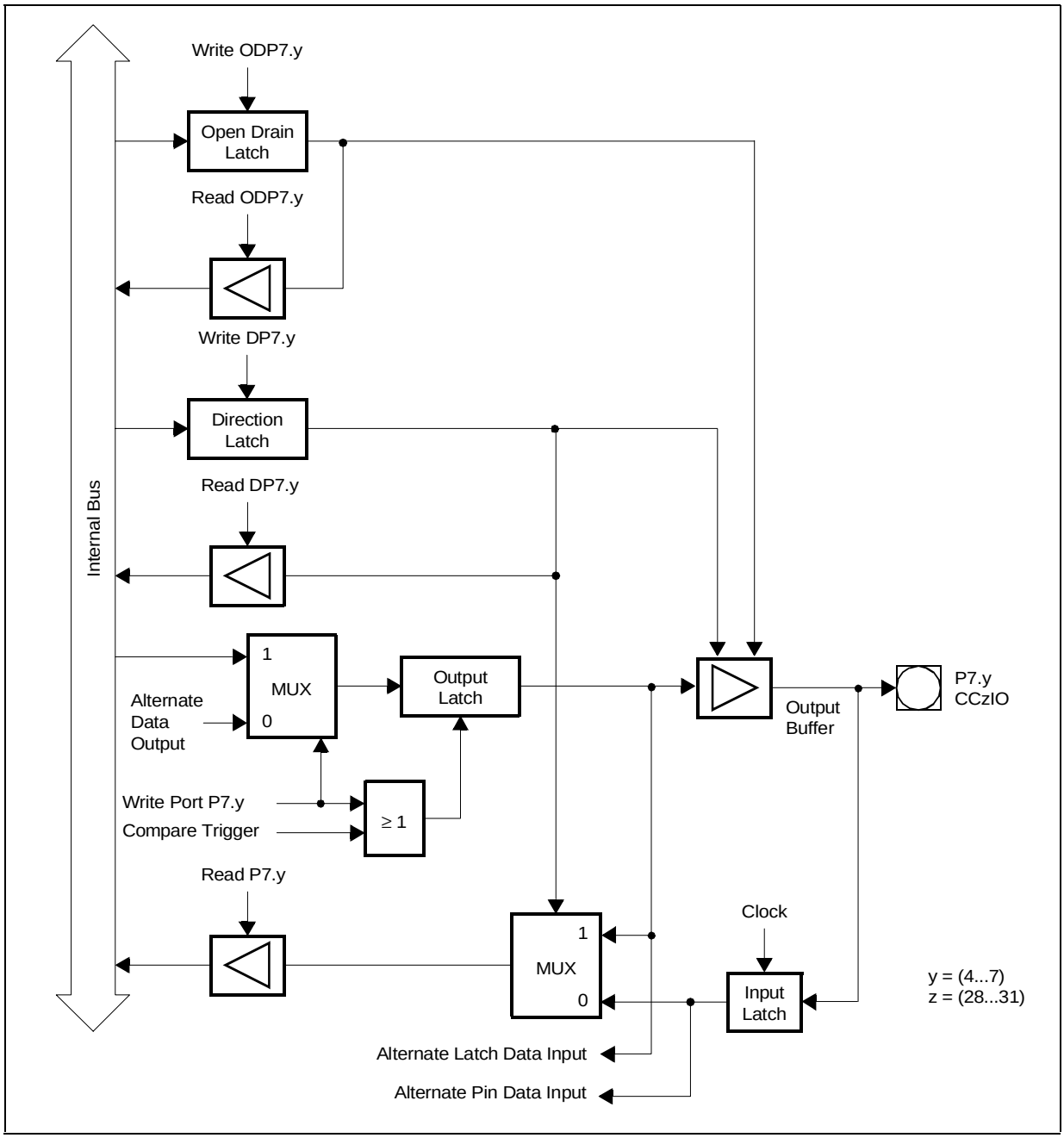
Pins P7.3...P7.0 (POUT3...POUT0) XOR the alternate data output with the port latch output, which allows to use the alternate data directly or inverted at the pin driver.

Figure 48 : Block diagram of Port7 pins (P7.3...P7.0)



Pins P7.7...P7.4 (CC31IO...CC28IO) combine internal bus data and alternate data output before the port latch input, as do the Port2 pins.

Figure 49 : Block diagram of Port7 pins (P7.7...P7.4)



7.10 - Port8

If this 8-bit port is used for general purpose I/O, the direction of each line can be configured via the corresponding direction register DP8. Each port line can be switched into push-pull or open-drain mode via the open-drain control register ODP8.

P8 (FFD4h / EAh) SFR Reset Value: - - 00h

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

Bit	Function
P8.y	**Port data register P8 bit y**

DP8 (FFD6h / EBh) SFR Reset Value: - - 00h

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

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

ODP8 (F1D6h / EBh) ESFR Reset Value: - - 00h

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

Bit	Function
ODP8.y	**Port8 Open Drain control register bit y** ODP8.y = 0: Port line P8.y output driver in push-pull mode ODP8.y = 1: Port line P8.y output driver in open drain mode

7.10.1 - Alternate Functions of Port8

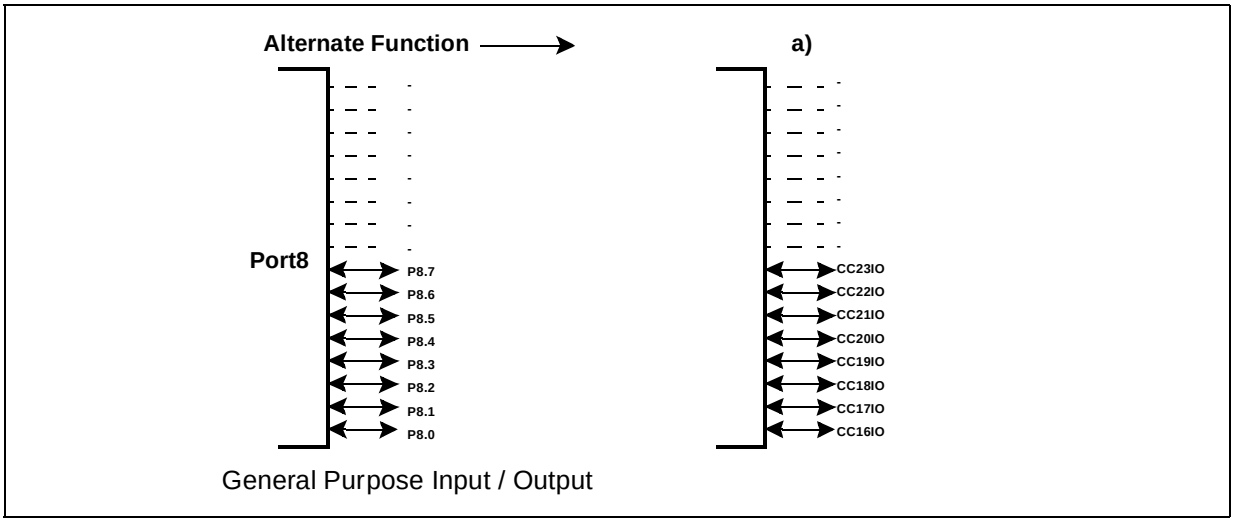
All Port8 lines (P8.7...P8.0) support capture inputs or compare outputs (CC23IO...CC16IO) for the CAPCOM2 unit (see Table 25). The use of the port lines by the CAPCOM unit, its accessibility via software and the precautions are the same as described for the Port2 lines.

As all other capture inputs, the capture input function of pins P8.7...P8.0 can also be used as external interrupt inputs with a sample rate of 8 CPU clock cycles.

Table 25 : Port8 alternate functions

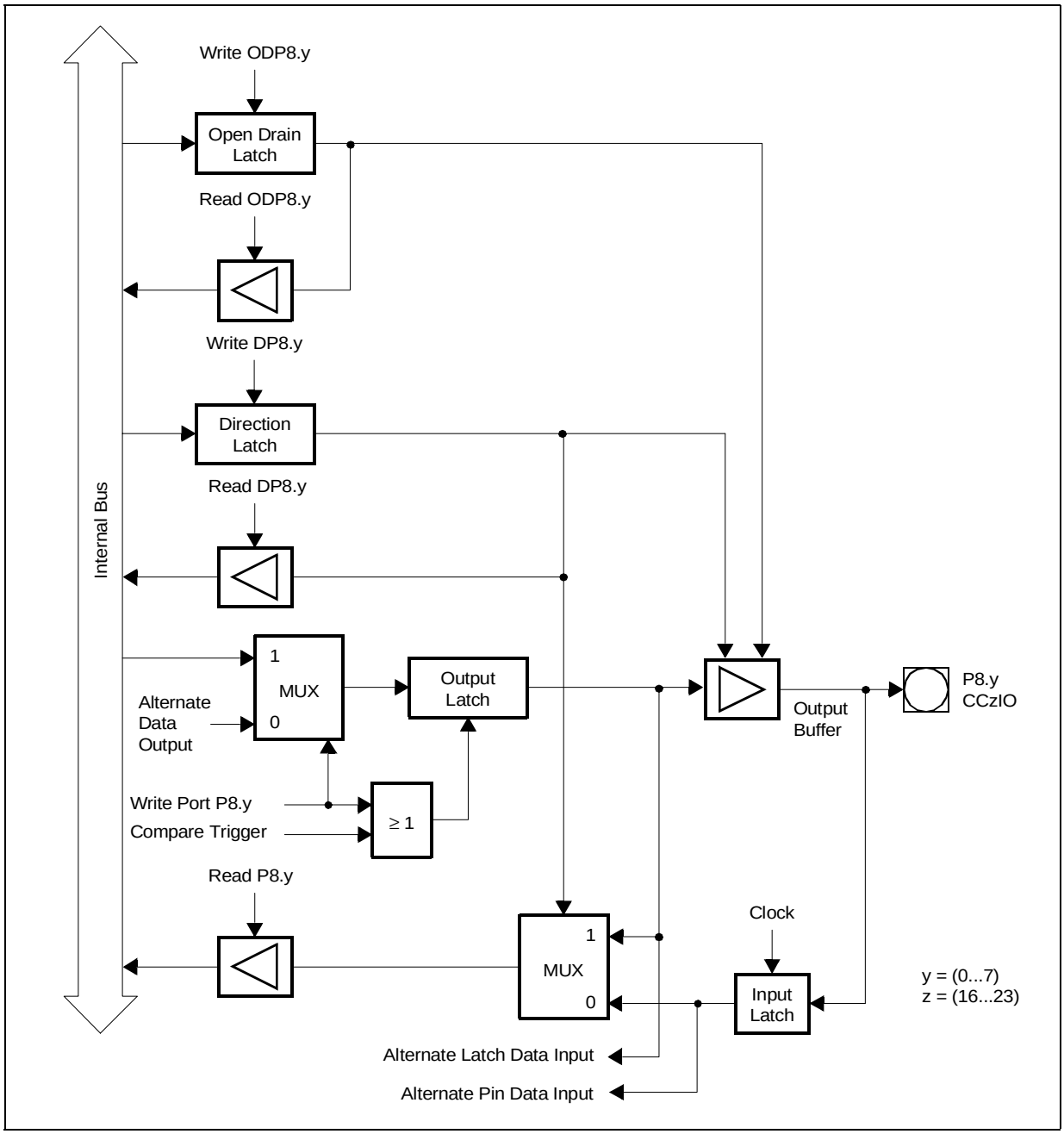
Port8 Pin	Alternate Function	
P8.0	CC16IO	Capture input / compare output channel 16
P8.1	CC17IO	Capture input / compare output channel 17
P8.2	CC18IO	Capture input / compare output channel 18
P8.3	CC19IO	Capture input / compare output channel 19
P8.4	CC20IO	Capture input / compare output channel 20
P8.5	CC21IO	Capture input / compare output channel 21
P8.6	CC22IO	Capture input / compare output channel 22
P8.7	CC23IO	Capture input / compare output channel 23

Figure 50 : Port8 I/O and alternate functions



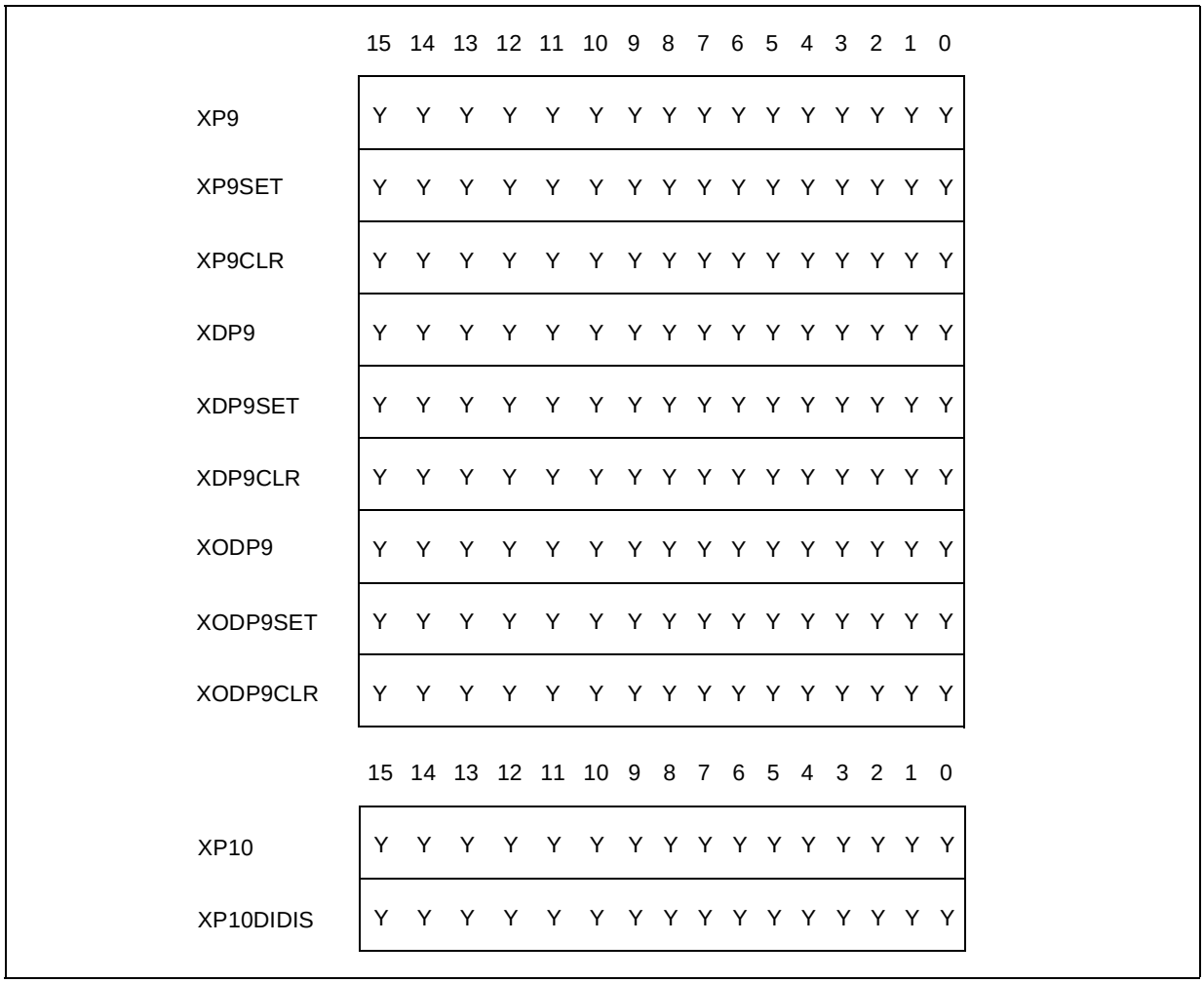
The pins of Port8 combine internal bus data and alternate data output before the port latch input, as do the Port2 pins.

Figure 51 : Block diagram of Port8 pins



7.11 - XPort 9

Figure 52 : XPorts Registers



The XPort9 is enabled by setting XPEN bit 2 of the SYSCON register and XPERCONEN3 of the new XPERCON register. On the XBUS interface, the register are not bit-addressable.

This 16-bit port is used for general purpose I/O, the direction of each line can be configured via the corresponding direction register XDP9. Each port line can be switched into push/pull or open drain mode via the open drain control register XODP9.

All port lines can be individually (bit-wise) programmed. The "bit-addressable" feature is available via specific "Set" and "Clear" registers: XP9SET, XP9CLR, XDP9SET, XDP9CLR, XODP9SET, XODP9CLR.

XP9 (C100h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XP9.15	**XP9.14**	**XP9.13**	**XP9.12**	**XP9.11**	**XP9.10**	**XP9.9**	**XP9.8**	**XP9.7**	**XP9.6**	**XP9.5**	**XP9.4**	**XP9.3**	**XP9.2**	**XP9.1**	**XP9.0**
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
XP9.y	**Port data register XP9 bit y**

XP9SET (C102h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XP9SET.15	XP9SET.14	XP9SET.13	XP9SET.12	XP9SET.11	XP9SET.10	XP9SET.9	XP9SET.8	XP9SET.7	XP9SET.6	XP9SET.5	XP9SET.4	XP9SET.3	XP9SET.2	XP9SET.1	XP9SET.0
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Bit	Function
XP9SET.y	Writing a '1' will set the corresponding bit in XP9 register, Writing a '0' has no effect.

XP9CLR (C104h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XP9CLR.15	XP9CLR.14	XP9CLR.13	XP9CLR.12	XP9CLR.11	XP9CLR.10	XP9CLR.9	XP9CLR.8	XP9CLR.7	XP9CLR.6	XP9CLR.5	XP9CLR.4	XP9CLR.3	XP9CLR.2	XP9CLR.1	XP9CLR.0
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Bit	Function
XP9CLR.y	Writing a '1' will clear the corresponding bit in XP9 register, Writing a '0' has no effect.

XDP9 (C200h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XDP9.15	**XDP9.14**	**XDP9.13**	**XDP9.12**	**XDP9.11**	**XDP9.10**	**XDP9.9**	**XDP9.8**	**XDP9.7**	**XDP9.6**	**XDP9.5**	**XDP9.4**	**XDP9.3**	**XDP9.2**	**XDP9.1**	**XDP9.0**
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
XDP9.y	**Port direction register XDP9 bit y** XDP9.y = 0: Port line XP9.y is an input (high-impedance) XDP9.y = 1: Port lineX P9.y is an output

XDP9SET (C202h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XDP9SET.15	XDP9SET.14	XDP9SET.13	XDP9SET.12	XDP9SET.11	XDP9SET.10	XDP9SET.9	XDP9SET.8	XDP9SET.7	XDP9SET.6	XDP9SET.5	XDP9SET.4	XDP9SET.3	XDP9SET.2	XDP9SET.1	XDP9SET.0
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Bit	Function
XDP9SET.y	Writing a '1' will set the corresponding bit in XDP9 register, Writing a '0' has no effect.

XDP9CLR (C204h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XDP9CLR.15	XDP9CLR.14	XDP9CLR.13	XDP9CLR.12	XDP9CLR.11	XDP9CLR.10	XDP9CLR.9	XDP9CLR.8	XDP9CLR.7	XDP9CLR.6	XDP9CLR.5	XDP9CLR.4	XDP9CLR.3	XDP9CLR.2	XDP9CLR.1	XDP9CLR.0
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Bit	Function
XDP9CLR.y	Writing a '1' will clear the corresponding bit in XDP9 register, Writing a '0' has no effect.

XODP9 (C300h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XODP9.15	**XODP9.14**	**XODP9.13**	**XODP9.12**	**XODP9.11**	**XODP9.10**	**XODP9.9**	**XODP9.8**	**XODP9.7**	**XODP9.6**	**XODP9.5**	**XODP9.4**	**XODP9.3**	**XODP9.2**	**XODP9.1**	**XODP9.0**
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
XODP9.y	**Port 9 Open Drain control register bit y** XODP9.y = 0: Port line XP9.y output driver in push/pull mode XODP9.y = 1: Port line XP9.y output driver in open drain mode

XODP9SET (C302h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XODP9 SET.15	XODP9 SET.14	XODP9 SET.13	XODP9 SET.12	XODP9 SET.11	XODP9 SET.10	XODP9 SET.9	XODP9 SET.8	XODP9 SET.7	XODP9 SET.6	XODP9 SET.5	XODP9 SET.4	XODP9 SET.3	XODP9 SET.2	XODP9 SET.1	XODP9 SET.0
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Bit	Function
XODP9SET.y	Writing a '1' will set the corresponding bit in XODP9 register, Writing a '0' has no effect.

XODP9CLR (C304h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XODP9 CLR.15	XODP9 CLR.14	XODP9 CLR.13	XODP9 CLR.12	XODP9 CLR.11	XODP9 CLR.10	XODP9 CLR.9	XODP9 CLR.8	XODP9 CLR.7	XODP9 CLR.6	XODP9 CLR.5	XODP9 CLR.4	XODP9 CLR.3	XODP9 CLR.2	XODP9 CLR.1	XODP9 CLR.0
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W

Bit	Function
XODP9CLR.y	Writing a '1' will clear the corresponding bit in XODP9 register, Writing a '0' has no effect.

7.12 - XPort 10

The XPort10 is enabled by setting XPEN bit 2 of the SYSCON register and bit 3 of the new XPERCON register. On the XBUS interface, the register are not bit-addressable. This 16-bit input port can only read data. There is no output latch and no direction register. Data written to XP10 will be lost.

XP10 (C380h) XBUS Reset Value: XXXXh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XP10.15	XP10.14	XP10.13	XP10.12	XP10.11	XP10.10	XP10.9	XP10.8	XP10.7	XP10.6	XP10.5	XP10.4	XP10.3	XP10.2	XP10.1	XP10.0
R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R

Bit	Function
XP10.y	**Port data register XP10 bit y (Read only)**

XP10DIDIS (C382h) Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
XP10 DIDIS.15	XP10 DIDIS.14	XP10 DIDIS.13	XP10 DIDIS.12	XP10 DIDIS.11	XP10 DIDIS.10	XP10 DIDIS.9	XP10 DIDIS.8	XP10 DIDIS.7	XP10 DIDIS.6	XP10 DIDIS.5	XP10 DIDIS.4	XP10 DIDIS.3	XP10 DIDIS.2	XP10 DIDIS.1	XP10 DIDIS.0
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
XP10DIDIS.y	**XPort 10 Digital Disable register bit y** 0 Port line XP10.y digital input is enabled (Schmitt trigger enabled) 1 Port line XP10.y digital input is disabled (Schmitt trigger disabled, necessary for input leakage current reduction)

New Disturb Protection on Analog Inputs

A new register is provided for additional disturb protection support on analog inputs for Port XP10.

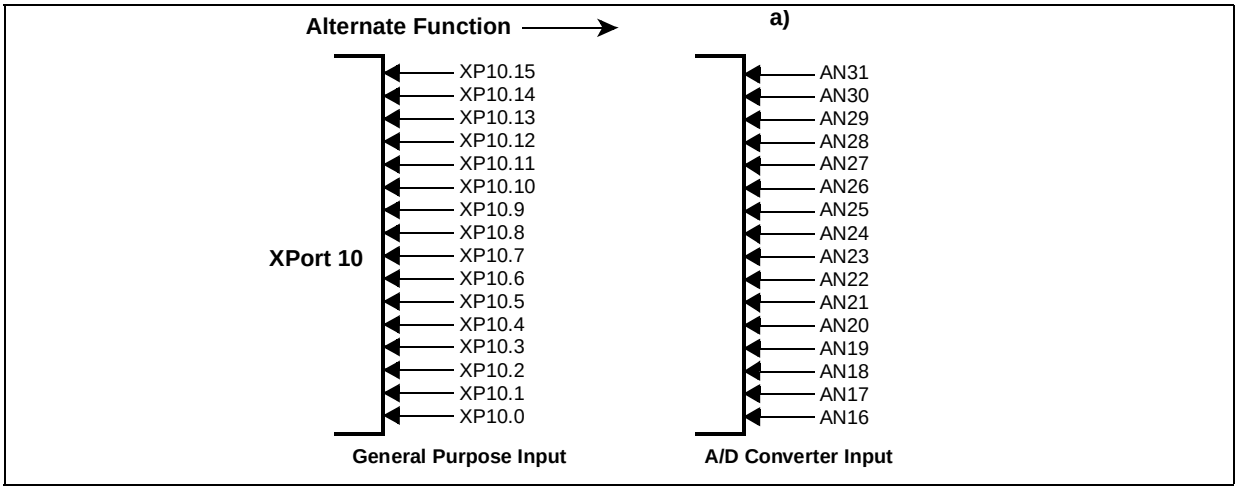
7.12.1 - Alternate Functions of XPort 10

Each line of XPort 10 is also connected to one of the multiplexer of the Analog/Digital Converter. All port lines (XP10.15...XP10.0) can accept analog signals (AN31...AN16) that can be converted by the ADC. No special programming is required for pins that shall be used as analog inputs. The Table 26 summarizes the alternate functions of XPort 10.

Table 26 : XPort 10 Alternate Functions

XPort 10 Pin	Alternate Function a)
P10.0	Analog Input AN16
P10.1	Analog Input AN17
P10.2	Analog Input AN18
P10.3	Analog Input AN19
P10.4	Analog Input AN20
P10.5	Analog Input AN21
P10.6	Analog Input AN22
P10.7	Analog Input AN23
P10.8	Analog Input AN24
P10.9	Analog Input AN25
P10.10	Analog Input AN26
P10.11	Analog Input AN27
P10.12	Analog Input AN28
P10.13	Analog Input AN29
P10.14	Analog Input AN30
P10.15	Analog Input AN31

Figure 53 : PORT10 I/O and Alternate Functions



8 - DEDICATED PINS

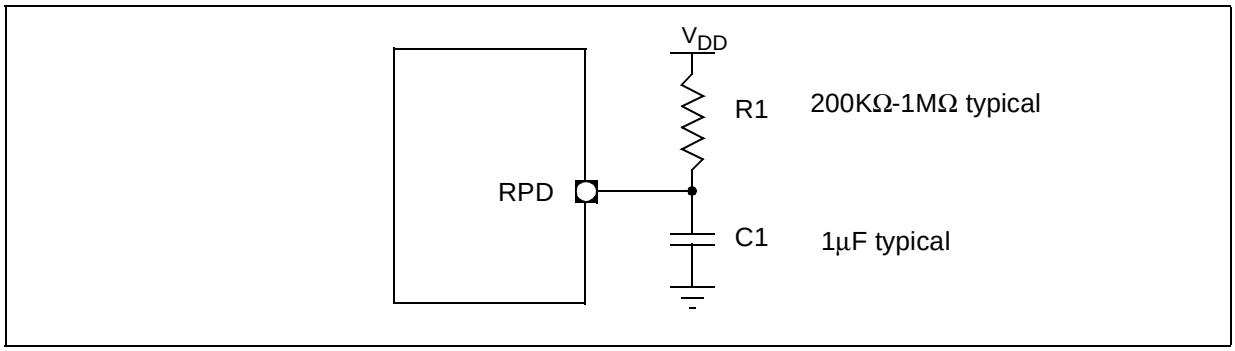
Most of the input/output or control signals of the ST10F280 are realized as alternate functions of pins of the parallel ports. There is, however, a number of signals that use separate pins, including the oscillator, special control signals and the power supply.

The Table 27 summarizes the dedicated pins of the ST10F280.

Table 27 : Summary of dedicated pins

Pin(s)	Function
ALE	**Address Latch Enable:** controls external address latches that provide a stable address in multiplexed bus modes. ALE is activated for every external bus cycle independent of the selected bus mode. It is also activated for bus cycles with a de-multiplexed address bus. When an external bus is enabled (if one or more of the BUSACT bit is set) also X-Peripheral accesses will generate an active ALE signal. ALE is not activated for internal accesses, like accesses to Flash, to the internal RAM and to the special function registers. In single chip mode, when no external bus is enabled (no BUSACT bit set), ALE will also remain inactive for X-Peripheral accesses
$\overline{RD}$	**External Read Strobe:** controls the output drivers of external memory or peripherals when the ST10F280 reads data from these external devices. During reset and during Hold mode an internal pull-up 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 ST10F280 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/BHE). During reset and during Hold mode an internal pull-up 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. For the ST10F280 the polarity can be set to $\overline{READY}$ or READY by setting bit 13 in the BUSCON register.
$\overline{EA}$	**External Access Enable:** determines, if the ST10F280 after reset starts fetching code from the internal Memory area ($\overline{EA}$='1') or via the external bus interface ($\overline{EA}$='0').
$\overline{NMI}$	**Non-Maskable Interrupt Input:** allows to trigger a high priority trap via an external signal. It can be used as power fail input or to validate the PWRDN instruction that switches the ST10F280 into power-down mode.
$\overline{RSTIN}$	**Reset Input:** puts the ST10F280 into the reset default configuration 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.
$\overline{RSTOUT}$	**Reset Output:** provides a special reset signal for external circuitry. $\overline{RSTOUT}$ is activated at the beginning of the reset sequence, triggered via $\overline{RSTIN}$, a watchdog timer overflow or by the SRST instruction. $\overline{RSTOUT}$ 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.
XADCINJ	Output trigger for ADC channel injection.
V_{DD}, V_{SS}	**Digital Power Supply and Ground (6 pins each):** provides the power supply for the digital logic of the ST10F280. All V_{DD} pins and all V_{SS} pins must be connected to the power supply and ground, respectively.
RPD	**Exit from powerdown:** If a Fast External Interrupt pin (EX3IN..EX0IN) is used to exit from Power Down mode, an external RC circuit should be connected to the 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 see Figure 54. For more information on exiting power down mode refer to Chapter 20.
DC1 DC2	3.3V Decoupling pin: a decoupling capacitor of ≥ 330 nF must be connected between this pin and nearest V_{SS} pin.

Figure 54 : RPD external RC circuit



RPD external RC circuit is used for exiting power-down mode with external interrupt and for power-up asynchronous reset.

9 - THE EXTERNAL BUS INTERFACE

The on-chip peripherals and on-chip RAM and Flash Memory only cover a small fraction of the ST10F280 address space. The external bus interface gives access to external peripherals and additional volatile and non-volatile memory. It provides a number of configurations and can be tailored to fit perfectly into a given application system (see Figure 55).

Accesses to external memory or peripherals are executed by the integrated External Bus Controller (EBC). The function of the EBC is controlled via the SYSCON register and the BUSCONx and ADDRSELx registers. The BUSCONx registers specify the external bus cycles in terms of address (mux/demux), data (16-bit/8-bit), chip selects and length (wait-states / READY control / ALE / RW delay). These parameters are used for accesses within a specific address area which is defined via the corresponding register ADDRSELx. The four pairs BUSCON1/ADDRSEL1...BUSCON4/ADDRSEL4 allow to define four independent "address windows", while all external accesses outside these windows are controlled via register BUSCON0.

9.1 - Single Chip Mode

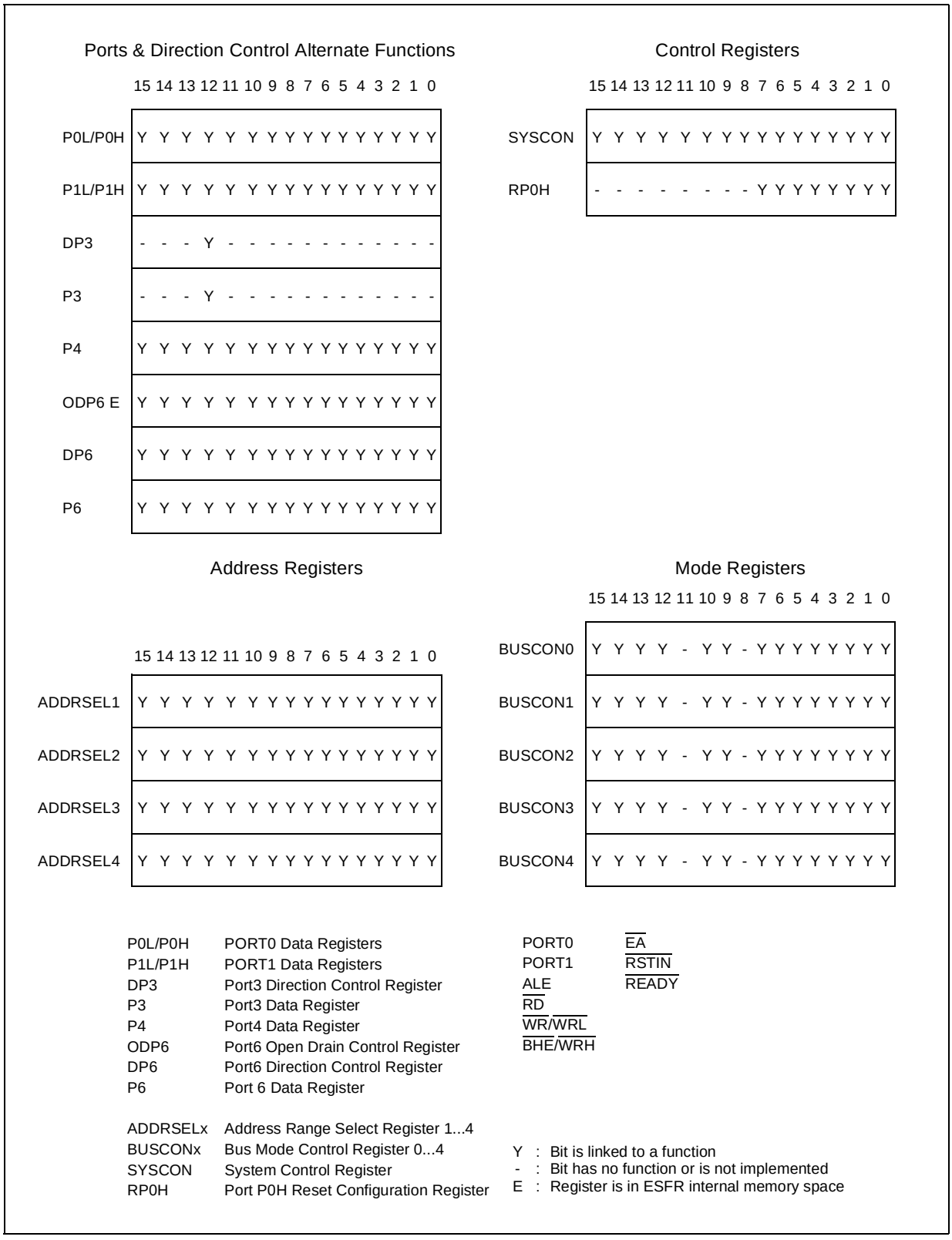
Single chip mode is entered, when pin $\overline{EA}$ is high during reset. In this case register BUSCON0 is initialized with 0000h, which also resets bit BUSACT0, so no external bus is enabled.

In single chip mode the ST10F280 operates only with and out of internal resources. No external bus is configured and no external peripherals and/or memory can be accessed. Also no port lines are occupied for the bus interface.

When running in single chip mode, however, external access may be enabled by configuring an external bus under software control. Single chip mode allows the ST10F280 to start execution out of the internal Flash program memory.

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

Figure 55 : SFRs and port pins associated with the external bus interface



9.2 - External Bus Modes

When the external bus interface is enabled (bit BUSACTx='1' of BUSCONx register) and configured (bit-field BTYP), the ST10F280 uses a subset of its port lines together with some control lines to build the external bus.

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

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

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

The 16 Mbyte address space of the ST10F280 is divided into 256 segments of 64 Kbytes each. The 16-bit intra-segment address is output on PORT0 for multiplexed bus modes or on PORT1 for de-multiplexed bus modes.

When segmentation is disabled, only one 64 Kbyte segment can be used and accessed. Otherwise, additional address lines may be output on Port4, and/or several chip select lines may be used to select different memory banks or peripherals. These functions are selected during reset via bit-fields SALSEL and CSSEL of register RP0H, respectively.

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

9.2.1 - Multiplexed Bus Modes

In the multiplexed bus modes the 16-bit intra-segment address and 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, an 8-bit data bus requires a byte latch (the address bit 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 Port4 (if segmentation is enabled) and do not require latches.

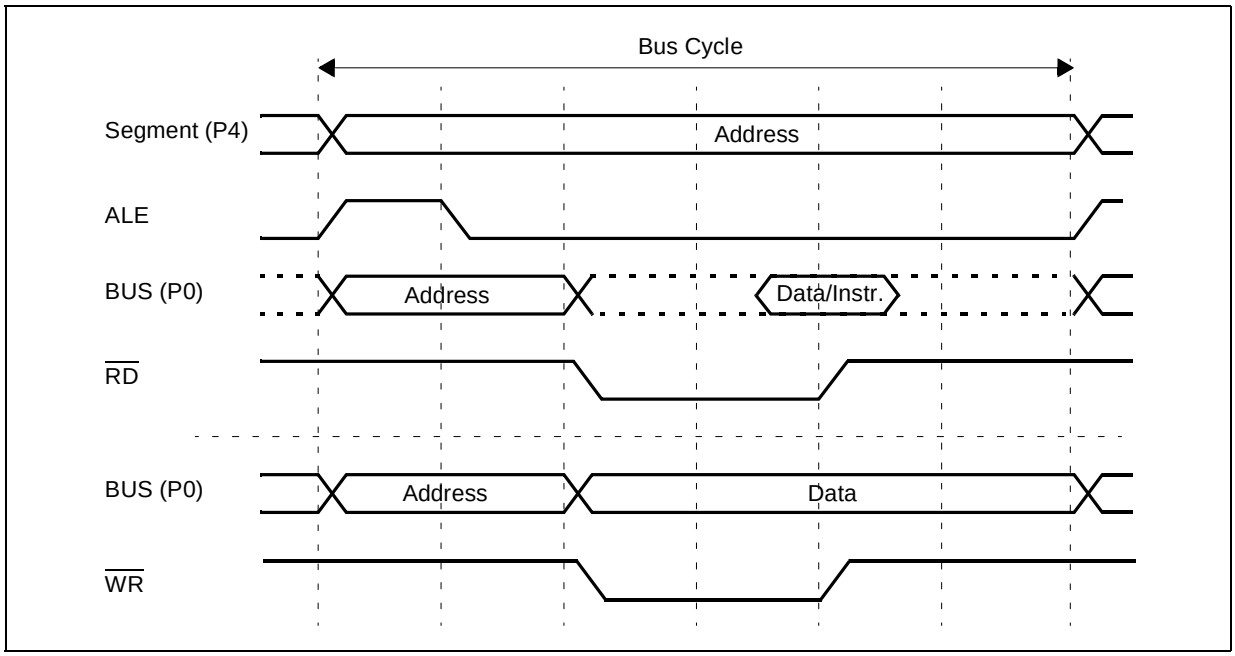
The EBC initiates an external access by generating the Address Latch Enable signal (ALE) and then placing an address on the bus. The falling edge of ALE triggers an external latch to capture the address.

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 56 : Multiplexed bus cycle



9.2.2 - De-multiplexed Bus Modes

In the de-multiplexed 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 Port4 (if selected via SALSEL during reset). No address latches are required.

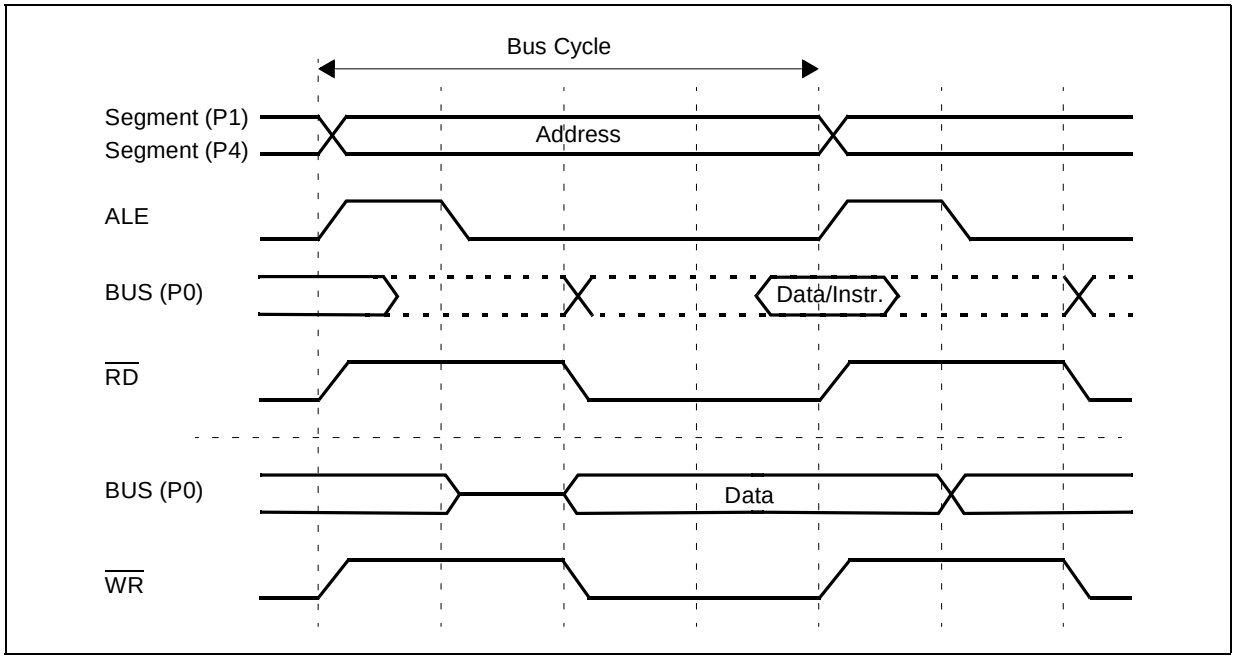
The EBC initiates an external access by placing an address on the address bus. After a programmable period of time the EBC activates the respective command signal ($\overline{RD}$, $\overline{WR}$, $\overline{WRL}$, $\overline{WRH}$).

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 57 : De-multiplexed bus cycle



9.2.3 - Switching Between the Bus Modes

The EBC allows dynamic switching between different bus modes, this means that subsequent external bus cycles may be executed in different ways. Certain address areas may use multiplexed or de-multiplexed buses or use READY control or predefined wait-states.

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 de-multiplexed bus mode, even if the current bus cycle uses a multiplexed bus mode. This means that an external address decoder can be connected to PORT1 only, while using it for all kinds of bus cycles.

Note Never change the configuration for an address area that currently supplies the instruction stream. Due to the internal pipelines 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 use 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 de-multiplexed to multiplexed bus mode represents a special case. The bus cycle is started by activating ALE and driving the address to Port4 and PORT1 as usual, if another BUSCON register selects a de-multiplexed 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 58).

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

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.

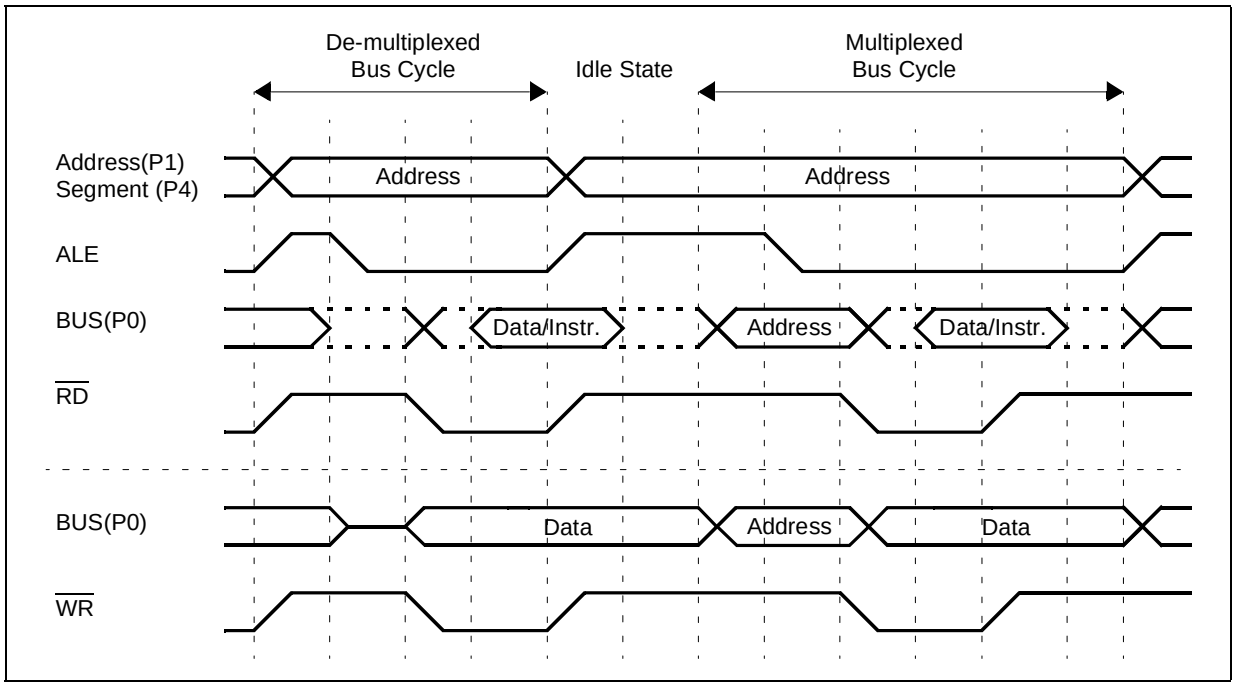
Word accesses on an 8-bit data bus are automatically split into two subsequent byte accesses, where the low byte is accessed first, then the high byte. The assembly of byte to words and the disassembly of words into byte is handled by the EBC and is transparent to the CPU and the programmer.

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 byte 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 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 byte from an external 16-bit device, whole words may be read and the ST10F280 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 byte should be selected using $\overline{BHE}$ and A0.

Figure 58 : Switching from de-multiplexed to multiplexed bus mode



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

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

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

Bit BYTDIS is provided for controlling the active low Byte High Enable ($\overline{BHE}$) pin. The function of the $\overline{BHE}$ pin is enabled, if the BYTDIS bit contains a '0'. Otherwise, it is disabled and the pin can be used as standard I/O pin. The $\overline{BHE}$ pin is implicitly used by the External Bus Controller to select one of two byte-organized memory chips, which are connected to the ST10F280 via a word-wide external data bus. After reset the $\overline{BHE}$ function is automatically enabled (BYTDIS = '0'), if a 16-bit data bus is selected during reset, otherwise it is disabled (BYTDIS='1'). It may be disabled, if byte access to 16-bit memory is not required, and the $\overline{BHE}$ signal is not used.

9.2.6 - Segment Address Generation

During external accesses the EBC generates a (programmable) number of address lines on Port4, which extend the 16-bit address output on PORT0 or PORT1, and so increase the accessible address space. The number of segment address lines is selected during reset and coded in bit-field SALSEL in register RP0H (see table below).

SALSEL	Segment address lines	Directly accessible address space
1 1	Two: A17...A16	256 Kbytes (Default without pull-downs)
1 0	Eight: A23...A16	16 Mbytes (Maximum)
0 1	None	64 Kbytes (Minimum)
0 0	Four: A19...A16	1 Mbyte

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

9.2.7 - $\overline{CS}$ Signal Generation

During external accesses the EBC can generate a (programmable) number of $\overline{CS}$ lines on Port6, which allows to directly select external peripherals or memory banks without requiring an external decoder. The number of $\overline{CS}$ lines is selected during reset and coded in bit field CSSEL in register RP0H (see table below).

CSSEL	Chip Select Lines	Note
1 1	Five: $\overline{CS4}$...$\overline{CS0}$	Default without pull-downs
1 0	None	Port6 pins free for I/O
0 1	Two: $\overline{CS1}$...$\overline{CS0}$	
0 0	Three: $\overline{CS2}$...$\overline{CS0}$	

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

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

The chip select signals allow to operate in four different modes, which are selected via bit CSWENx and CSRENx in the respective BUSCONx register.

CSWENx	CSRENx	Chip Select Mode
0	0	Address Chip Select (Default after Reset, mode for $\overline{CS0}$)
0	1	Read Chip Select
1	0	Write Chip Select
1	1	Read/Write Chip Select

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

Read or write chip select signals remain active only as long as the associated control signal ($\overline{RD}$ or $\overline{WR}$) is active.

This also includes the programmable read/write delay. Read chip select is only activated for read cycles, write chip select is only activated for write cycles, read/write chip select is activated for both read and write cycles (write cycles are assumed, if any of the signals $\overline{WRH}$ or $\overline{WRL}$ becomes active).

These modes save external glue logic, when accessing external devices like latches or drivers that only provide a single enable input.

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

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

The pull-up devices are also active during bus hold on the selected $\overline{CS}$ lines, while $\overline{HLDA}$ is active and the respective pin is switched to push-pull mode. Open drain outputs will float during bus hold. In this case external pull-up 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 supports many configurations for the external memory. By increasing the number of segment address lines, a linear address space of 256 Kbytes, 1 Mbyte or 16 Mbytes can be addressed.

It is possible to implement a large memory area and to access a great number of external devices using an external decoder. By increasing the number of $\overline{CS}$ line, accesses can be made to 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 to give access to five memory banks of 1 Mbyte each, so the available address space is 5 Mbytes (without glue logic).

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

9.3 - Programmable Bus Characteristics

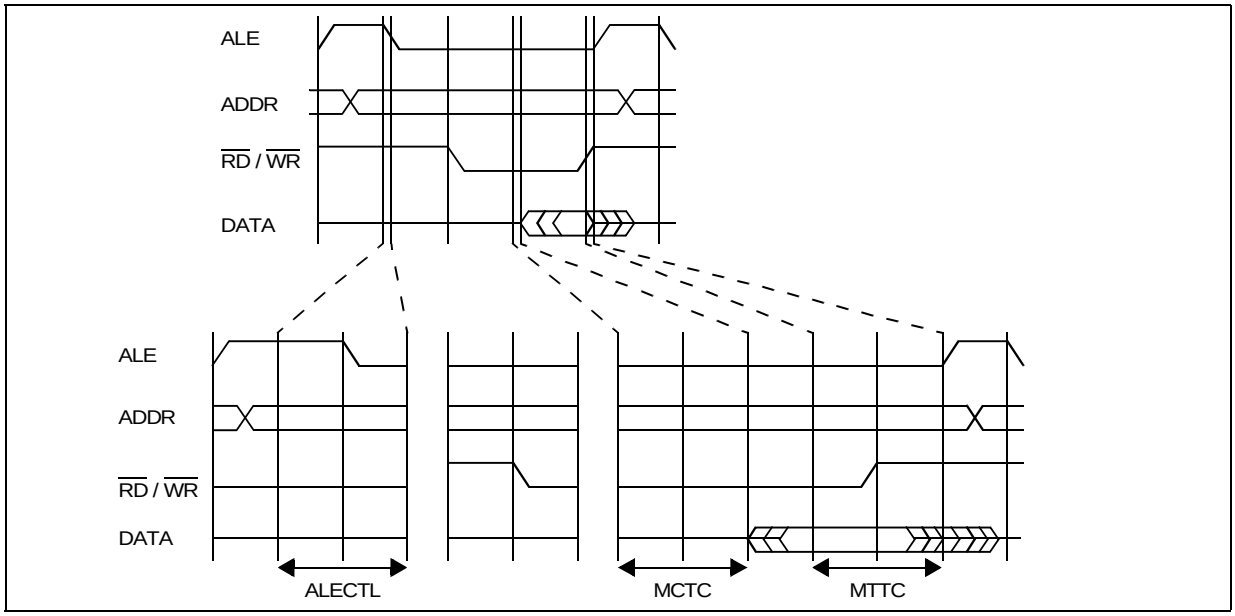
Important timing characteristics of the external bus interface have been made user programmable to allow to adapt it to a wide range of different external bus and memory configurations with different types of memories and/or peripherals.

The following parameters of an external bus cycle are programmable:

– **ALE control** defines the ALE signal length and the address hold time after its falling edge
– **Memory cycle time** (extendable with 1...15 wait-states) defines the allowable access time
– **Memory tri-state time** (extendable with 1 wait-state) defines the time for a data driver to float
– **Read/write delay time** defines when a command is activated after the falling edge of ALE
– **READY polarity** is programmable
– **READY control** defines, if a bus cycle is terminated internally or externally
– **Programmable chip select timing control**

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

Figure 59 : Programmable external bus cycle

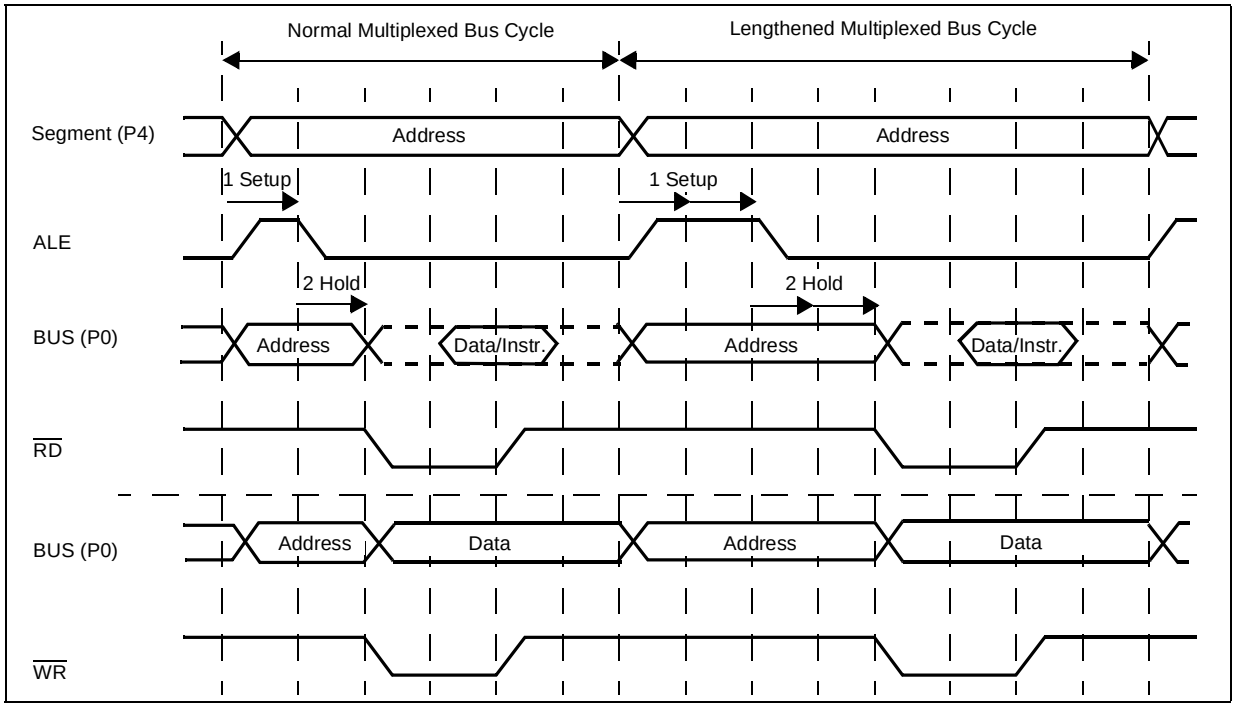


9.3.1 - ALE Length Control

The length of the ALE signal and the address hold time after its falling edge are controlled by the ALECTLx bit 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 cycle. Also the address hold time after the falling edge of ALE (on a multiplexed bus) will be prolonged by half a CPU clock, so the data transfer within a bus cycle refers to the same CLKOUT edges as usual (the data transfer is delayed by one CPU clock cycle). This allows more time for the address to be latched.

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

Figure 60 : ALE length control

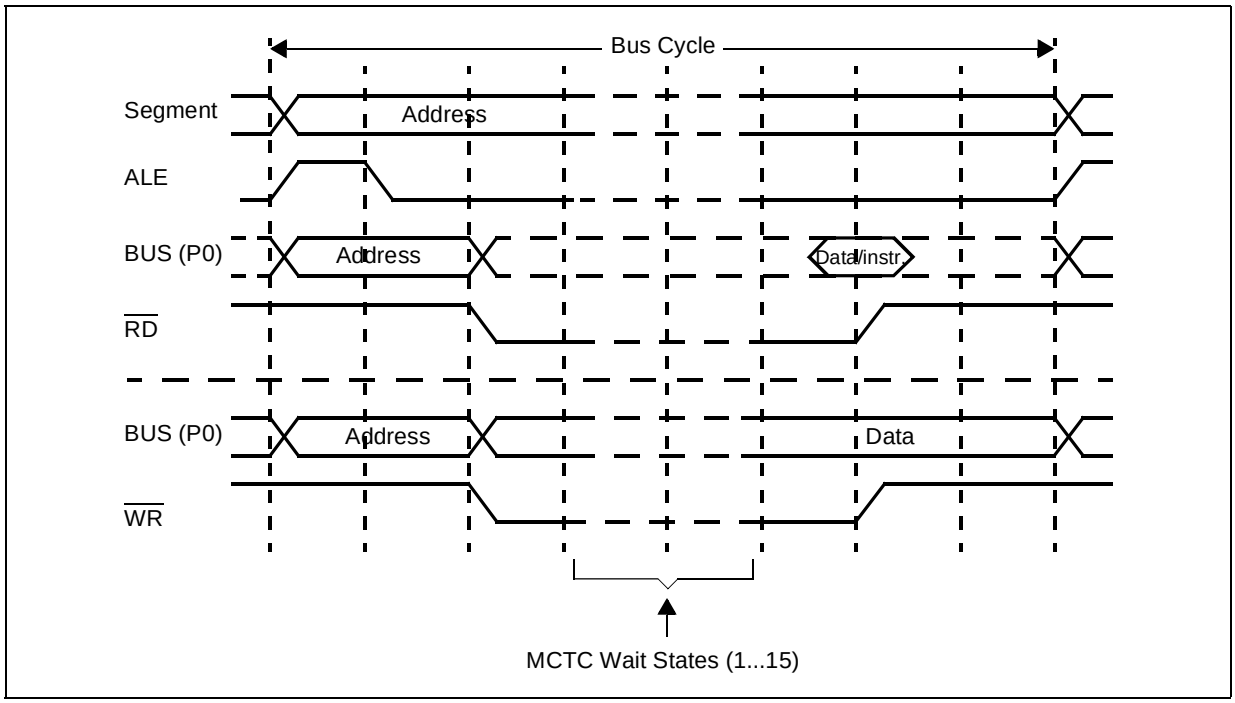


9.3.2 - Programmable Memory Cycle Time

The ST10F280 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 ST10F280 can be extended for a memory or a peripheral, which cannot keep pace with the controller's maximum speed some wait-states are introduced during the access (see Figure 61). During these memory cycle time wait-states, the CPU is idle, if this access is required for the execution of the current instruction. The memory cycle time wait-states can be programmed in increments of one CPU clock within a range from 0 to 15 (default after reset) via the MCTC fields of the BUSCON registers. 15-(MCTC) wait-states will be inserted.

Figure 61 : Memory cycle time



9.3.3 - Programmable Memory Tri-state Time

The ST10F280 allows the user to adjust the time between two subsequent external accesses to address slow external device. The tri-state time MTTC starts, when the external device has released the bus after deactivation of the read command ($\overline{RD}$).

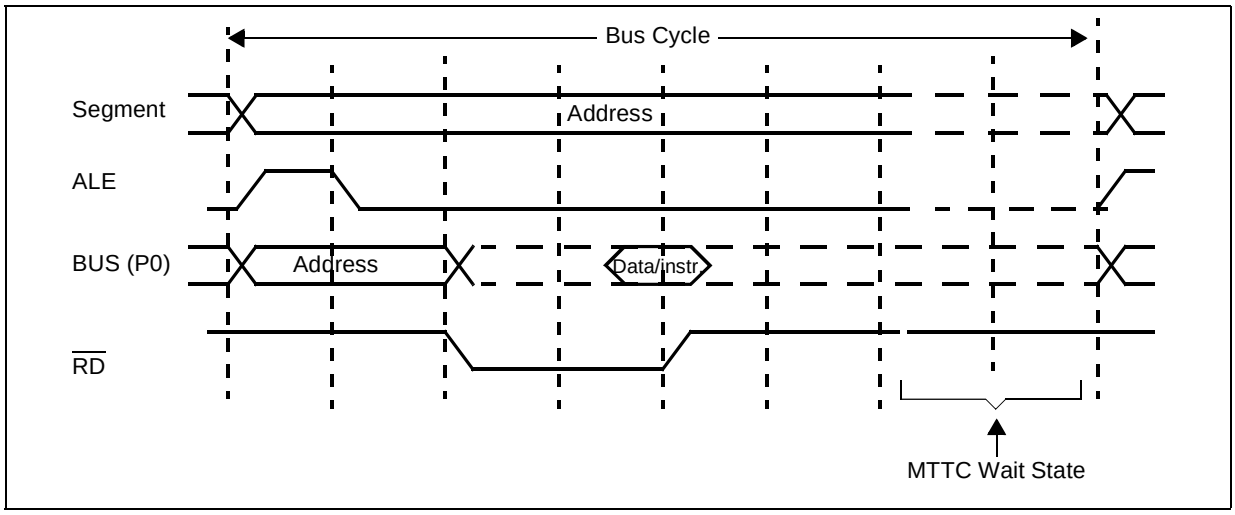
The output of the next address on the external bus can be delayed for a memory or peripheral, which needs more time to switch off its bus drivers, by introducing a wait-state after the previous bus cycle (see Figure 62).

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 wait-state requires one CPU clock and is controlled via the MTTCx bit of the BUSCON registers. A wait-state will be inserted, if bit MTTCx is '0' (default after reset).

External bus cycles in multiplexed bus modes implicitly add one tri-state time wait-state in addition to the programmable MTTC wait-state.
Any MTTC wait-states are applicable to both read and write cycles.

Figure 62 : Memory tri-state time



9.3.4 - Read / Write Signal Delay

The ST10F280 allows the user to adjust the timing of the read and write commands to account for timing requirements of external peripherals.

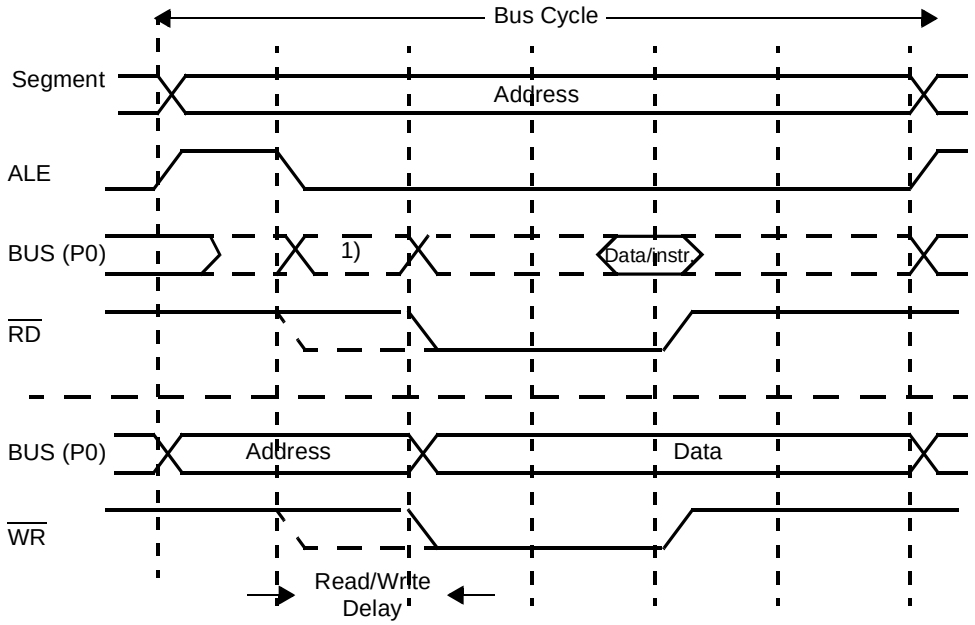
The read/write delay controls the time between the falling edge of ALE and the falling edge of the command. Without read/write delay the falling edges of ALE and command(s) are coincident (except for propagation delays). With the delay enabled, the command(s) become active half a CPU clock cycle after 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 ST10F280'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 bit in the BUSCON registers. The command(s) will be delayed, if bit RWDCx is '0' (default after reset).

Figure 63 : Read/write delay



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

9.3.5 - READY Polarity

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

9.3.6 - READY / $\overline{READY}$ Controlled Bus Cycles

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

For situations where the programmable wait-states are not enough, or where the response (access) time of a peripheral is not constant, the ST10F280 provides external bus cycles that are terminated by a READY or $\overline{READY}$ input signal (synchronous or asynchronous). In this case the ST10F280 first inserts a programmable number of wait-states (0...7) and then monitors the READY or $\overline{READY}$ line to determine the actual end of the current bus cycle. The external device drives READY or $\overline{READY}$ 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 in the associated BUSCON register (see Figure 64).

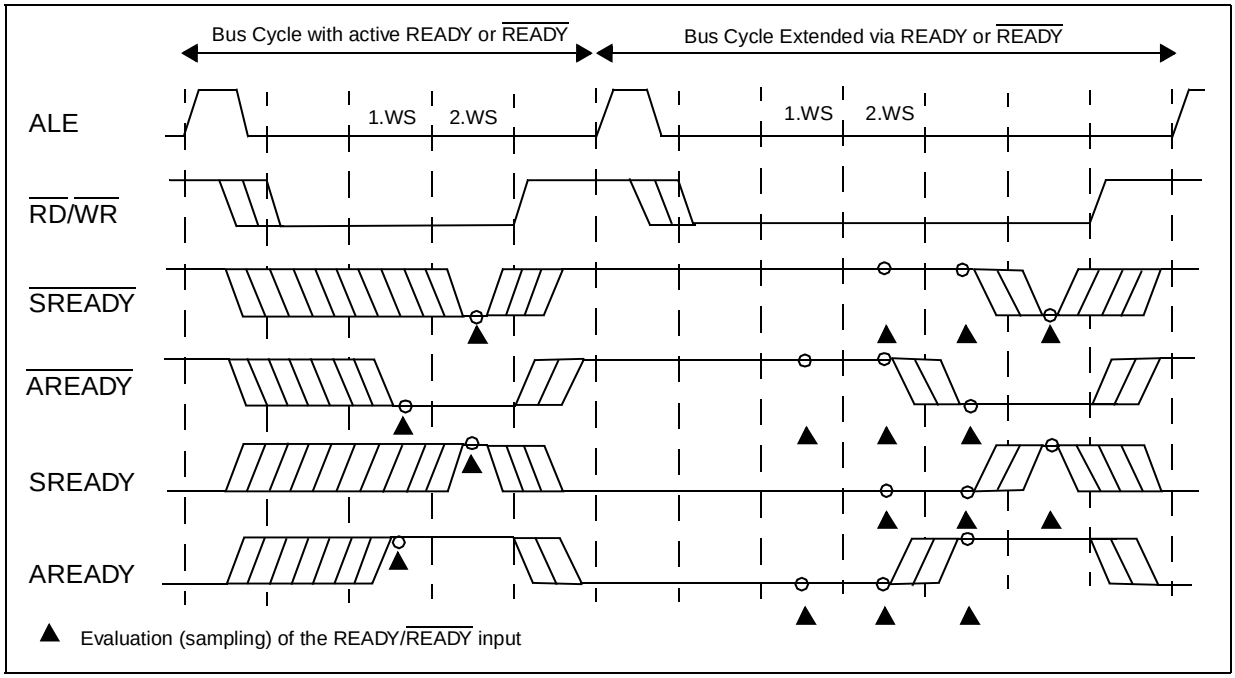
The READY/$\overline{\text{READY}}$ function is enabled by the RDYENx bit in the BUSCON registers. When this function is selected (RDYENx = '1'), only the lower 3 bits of the respective MCTC bit-field define the number of inserted wait-states (0...7), while the MSB of bit-field MCTC selects the $\overline{\text{READY}}$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

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

The synchronous READY/$\overline{\text{READY}}$ (SREADY / $\overline{\text{SREADY}}$) 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/$\overline{\text{READY}}$ (AREADY / $\overline{\text{AREADY}}$) is less restrictive, but requires additional wait-states caused by the internal synchronization. As the asynchronous READY/$\overline{\text{READY}}$ is sampled earlier (see Figure 64) programmed wait-states may be necessary to provide proper bus cycles (see also notes on "normally-ready" peripherals below).

Figure 64 : READY/$\overline{\text{READY}}$ controlled bus cycles



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 wait-states is advantageous in two cases:

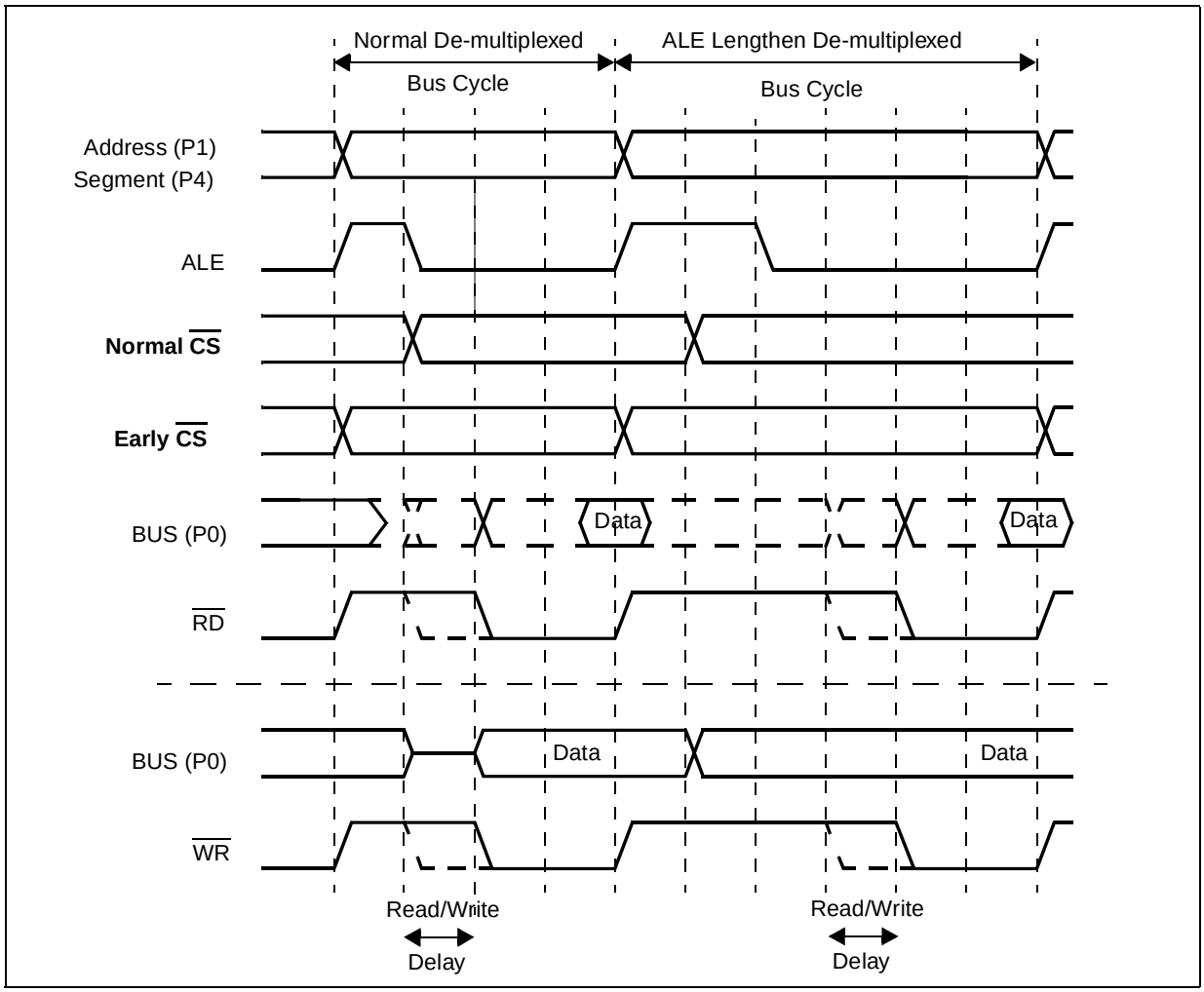
– Memory components with a fixed access time and peripherals operating with READY/$\overline{\text{READY}}$ may be grouped into the same address window. The (external) wait-state control logic in this case would activate READY/READY either upon the memory's chip select or with the peripheral's READY/READY output. After the predefined number of wait-states the ST10F280 will check its READY/$\overline{\text{READY}}$ line to determine the end of the bus cycle. For a memory access it will below already (see Figure 64), for a peripheral access it may be delayed. As memories tend to be faster than peripherals, there should be no impact on system performance.

– When using the READY/$\overline{\text{READY}}$ 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 ST10F280, the controller samples an active READY/$\overline{\text{READY}}$ and terminates the current bus cycle, which, of course, is too early. By inserting predefined wait-states 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 (after 2 wait-states in the Figure 64).

9.3.7 - Programmable Chip Select Timing Control

The position of the $\overline{\text{CS}}$ lines can be changed. By default (after reset), the $\overline{\text{CS}}$ lines change half a CPU clock cycle after the rising edge of ALE. With the CSCFG bit set in the SYSCON register, the $\overline{\text{CS}}$ lines change with the rising edge of ALE, therefore the $\overline{\text{CS}}$ lines change at the same time that the address lines are changed.

Figure 65 : Chip select delay



9.4 - Controlling the External Bus Controller

A set of registers controls the functions of the EBC. General features like the usage of interface pins (WR, BHE), segmentation and internal Memory mapping are controlled by the SYSCON register.

The properties of a bus cycle like chip select mode, usage of READY, length of ALE, external bus mode, read/write delay and wait-states are controlled by BUSCON4...BUSCON0 registers. Four of these registers (BUSCON4...BUSCON1) have an associated address select register (ADDRSEL4...ADDRSEL1) which allows to specify up to four address areas and the individual bus characteristics within these areas. All accesses that are not covered by these four areas are then controlled via BUSCON0. This allows to use memory components or peripherals with different interfaces within the same system, while optimizing accesses to each of them.

Note BUSCON4...BUSCON0 bit SGTDIS controls the correct stack operation (push/pop of CSP or not) during traps and interrupts.

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	XPEN	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
VISIBLE	**Visible Mode Control** '0': Accesses to XBUS peripherals are done internally '1': XBUS peripheral accesses are made visible on the external pins
XPEN	**XBUS Peripheral Enable bit** '0': Accesses to the on-chip XRAM are disabled, external bus cycles instead. '1': External bus cycles are executed for accesses to the XRAM area.
BDRSTEN	**Bidirectional Reset Enable** '0': RSTIN pin is an input pin only. SW Reset or WDT Reset have no effect on this pin '1': RSTIN pin is a bidirectional pin. This pin is pulled low during 1024 TCL during reset 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 µs, the CPU clock is switched automatically to PLL's base frequency (around 5MHz). '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 power supply current.
PWDCFG	**Power Down Mode Configuration Control** '0': Power Down Mode can only be entered during PWRDN instruction execution if NMI pin is low, otherwise the instruction has no effect. To exit Power Down Mode, an external reset must occurs by asserting the 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 or with external reset.
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** (Inverted copy of WRC bit of RPOH) '0': Pins WR and BHE retain their normal function '1': Pin WR acts as WRL, pin BHE acts as WRH
CLKEN	**System Clock Output Enable** (CLKOUT) '0': CLKOUT disabled, pin may be used for general purpose I/O '1': CLKOUT enabled, pin outputs the system clock signal

Bit	Function
BYTDIS	**Disable/Enable Control for Pin BHE** (Set according to data bus width) '0': Pin $\overline{BHE}$ enabled '1': Pin $\overline{BHE}$ disabled, pin may be used for general purpose I/O
ROMEN	**Internal Memory Enable** (Set according to pin $\overline{EA}$ during reset) '0': Internal memory disabled: accesses to the Flash Memory area use the external bus '1': Internal memory enabled
SGTDIS	**Segmentation Disable/Enable Control** '0': Segmentation enabled (CSP is saved/restored during interrupt entry/exit) '1': Segmentation disabled (Only IP is saved/restored)
ROMS1	**Internal Memory Mapping** '0': Internal memory area mapped to segment 0 (00'0000h...00'7FFFh) '1': Internal memory area mapped to segment 1 (01'0000h...01'7FFFh)
STKSZ	**System Stack Size** Selects the size of the system stack (in the internal RAM) from 32 to 1024 words

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 is also used for the very first code access after reset, is partly controlled by hardware, and it is initialized via PORT0 during the reset sequence.

This hardware control allows to define an appropriate external bus for systems, where no internal program memory is provided.

BUSCON0 (FF0Ch / 86h) SFR Reset Value: 0xx0h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSWEN0	CSREN0	RDYPOL0	RDYEN0	-	**BUSACT0**	**ALECTL0**	-	**BTYP**		MTTC0	RWDC0	MCTC			
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
CSWEN1	CSREN1	RDYPOL1	RDYEN1	-	BUSACT1	ALECTL1	-	BTYP		MTTC1	RWDC1	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
CSWEN2	CSREN2	RDYPOL2	RDYEN2	-	BUSACT2	ALECTL2	-	BTYP		MTTC2	RWDC2	MCTC			
RW	RW		RW		RW	RW		RW		RW	RW	RW			

BUSCON3 (FF18h / 8Ch) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CSWEN3	CSREN3	RDYPOL3	RDYEN3	-	BUSACT3	ALECTL3	-	BTYP		MTTC3	RWDC3	MCTC			
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
CSWEN4	CSREN4	RDYPOL4	RDYEN4	-	BUSACT4	ALECTL4	-	BTYP		MTTC4	RWDC4	MCTC			
RW	RW		RW		RW	RW		RW		RW	RW	RW			

Bit	Function
MCTC	**Memory Cycle Time Control** (Number of memory cycle time wait-states) 0 0 0 0: 15 wait-states (Number of wait-states = 15 - [MCTC]) . . . 1 1 1 1: No wait-states
RWDCx	**Read/Write Delay Control for BUSCONx** '0': With read/write delay, the CPU inserts 1 TCL after falling edge of ALE '1': No read/write delay, $\overline{RW}$ is activated after falling edge of ALE
MTTCx	**Memory Tristate Time Control** '0': 1 wait-state '1': No wait-state
BTYP	**External Bus Configuration** 0 0: 8-bit De-multiplexed Bus 0 1: 8-bit Multiplexed Bus 1 0: 16-bit De-multiplexed Bus 1 1: 16-bit Multiplexed Bus **Note:** For BUSCON0 BTYP bit-field 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 $\overline{READY}$ input signal
RDYPOLx	**Ready Active Level Control** '0': Active level on the READY pin is low, bus cycle terminates with a '0' on READY pin, '1': Active level on the READY pin is high, bus cycle terminates with a '1' on READY pin.
CSRENx	**Read Chip Select Enable** '0': The $\overline{CS}$ signal is independent of the read command ($\overline{RD}$) '1': The $\overline{CS}$ signal is generated for the duration of the read command
CSWENx	**Write Chip Select Enable** '0': The $\overline{CS}$ signal is independent of the write command ($\overline{WR}$,$\overline{WRL}$,$\overline{WRH}$) '1': The $\overline{CS}$ signal is generated for the duration of the write command

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

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

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

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

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

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

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

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

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

Bit	Function
RGSZ	**Range Size Selection** Defines the size of the address area controlled by the respective BUSCONx/ADDRSELx register pair. See Table 28.
RGSAD	**Range Start Address** Defines the upper bit of the start address (A23...) of the respective address area. See Table 28.

Note Register BUSCON0 controls the complete external address space, except for the 4 windows supported by BUSCON1 to BUSCON4. So there is no need of ADDRSEL0 register.

9.4.1 - Definition of Address Areas

The four register pairs BUSCON4/ADDRSEL4...BUSCON1/ADDRSEL1 allow to define 4 separate address areas within the address space of the ST10F280. 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 bit, which are not used within the address window of the specified size (see Table 28).

For a given window size, only those upper address bit of the start address are used (marked "R"), which are not implicitly used for addresses inside the window. The lower bit of the start address (marked "x") are disregarded.

Table 28 : Definition of address areas

Bit-field RGSZ	Resulting Window Size	Relevant bit (R) of Start Address (A23...A12)											
		A23											A12
0 0 0 0	4 Kbytes	R	R	R	R	R	R	R	R	R	R	R	R
0 0 0 1	8 Kbytes	R	R	R	R	R	R	R	R	R	R	R	x
0 0 1 0	16 Kbytes	R	R	R	R	R	R	R	R	R	R	x	x
0 0 1 1	32 Kbytes	R	R	R	R	R	R	R	R	R	x	x	x
0 1 0 0	64 Kbytes	R	R	R	R	R	R	R	R	x	x	x	x
0 1 0 1	128 Kbytes	R	R	R	R	R	R	R	x	x	x	x	x
0 1 1 0	256 Kbytes	R	R	R	R	R	R	x	x	x	x	x	x
0 1 1 1	512 Kbytes	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 Mbytes	R	R	R	x	x	x	x	x	x	x	x	x
1 0 1 0	4 Mbytes	R	R	x	x	x	x	x	x	x	x	x	x
1 0 1 1	8 Mbytes	R	x	x	x	x	x	x	x	x	x	x	x
1 1 x x	Reserved												

9.4.2 - Address Window Arbitration

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

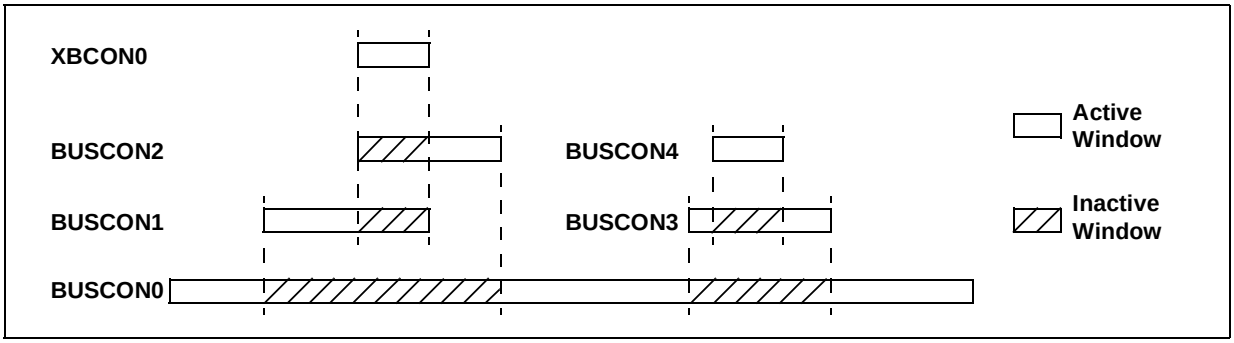
– The hardwired XADRSx registers are evaluated first. A match with one of these registers directs the access to the respective X-Peripheral using the corresponding XBCONx register and ignoring all other ADDRSELx registers.

– Registers ADDRSEL2 and ADDRSEL4 are evaluated before ADDRSEL1 and ADDRSEL3, respectively. A match with one of these registers directs the access to the respective external area using the corresponding BUSCONx register and ignoring registers ADDRSEL1/3 (see Figure 66).

– A match with registers ADDRSEL1 or ADDRSEL3 directs the access to the respective external area using the corresponding XBCONx register.

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

Figure 66 : Address window arbitration



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

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	**CLKCFG**			**SALSEL**		**CSSEL**		**WRC**
								R			R		R		R

Bit	Function
WRC [1]	**Write Configuration Control** (Set according to pin P0H.0 during reset) '0': Pins $\overline{WR}$ acts as $\overline{WRL}$, pin $\overline{BHE}$ acts as $\overline{WRH}$ '1': Pins $\overline{WR}$ and $\overline{BHE}$ retain their normal function
CSSEL [1]	**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 pull-downs)
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 pull-downs)
CLKCFG	P0H.7-5 / CPU Frequency $f_{CPU} = f_{XTAL} \times F$ / Notes 111 / $f_{XTAL} \times 4$ / Default configuration [1] 110 / $f_{XTAL} \times 3$ 101 / $f_{XTAL} \times 2$ 100 / $f_{XTAL} \times 5$ 011 / $f_{XTAL} \times 1$ / Direct drive 010 / $f_{XTAL} \times 10$ 001 / $f_{XTAL} \times 0.5$ / CPU clock via prescaler [2] 000 / $f_{XTAL} \times 2.5$

Notes 1. In ST10F280, RP0H.[7..0] bit are loaded only during a long hardware reset.

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

9.4.3 - 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 de-multiplexed external bus, even for multiplexed bus cycles.

– Not all address areas defined via registers ADDRSELx may overlap each other. The operation of the EBC will be unpredictable in such a case. See Section 9.4.2 - Address Window Arbitration.

– The address areas defined via registers ADDRSELx may overlap internal address areas. Internal accesses will be executed in this case.

– For any access to an internal address area the EBC will remain inactive (see EBC Idle State).

9.5 - EBC Idle State

When the external bus interface is enabled, but no external access is currently executed, the EBC is idle. As long as only internal resources (from an architecture point of view) like IRAM, GPRs or SFRs, etc. are used the external bus interface does not change (see Table 29).

Accesses to on-chip X-Peripherals are also controlled by the EBC. However, even though an X-Peripheral appears like an external peripheral to the controller, the respective accesses do not generate valid external bus cycles.

Due to timing constraints address and write data of an XBUS cycle are reflected on the external bus interface (see Table 29). The address mentioned above includes Port1, Port 4, BHE and ALE which also pulses for an XBUS cycle. The external CS signals on Port 6 are driven inactive (high) because the EBC switches to an internal XCS signal.

The **external control signals** (RD and WR or WRL/WRH if enabled) **remain inactive** (high) (see Table 29).

Table 29 : Status of the external bus interface during EBC idle state

Pins	Internal accesses only	XBUS accesses
PORT0	Tristate (floating)	Tristate (floating) for read accesses XBUS write data for write accesses
PORT1	Last used external address (if used for the bus interface)	Last used XBUS address (if used for the bus interface)
Port 4	Last used external segment address (on selected pins)	Last used XBUS segment address (on selected pins)
Port 6	Active external CS signal corresponding to last used address	Inactive (high) for selected CS signals
BHE	Level corresponding to last external access	Level corresponding to last XBUS access
ALE	Inactive (low)	Pulses as defined for X-Peripheral
RD	Inactive (high)	Inactive (high)
WR/WRL	Inactive (high)	Inactive (high)
WRH	Inactive (high)	Inactive (high)

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 ST10F280 supports this approach with the possibility to arbitrate the access to its external bus, and to the external devices.

This bus arbitration allows an external master to request the ST10F280's bus via the HOLD input. The ST10F280 acknowledges this request via the HLDA output and will float its bus lines in this case. The CS outputs provide internal pull-up devices.

The new master may now access the peripheral devices or memory banks via the same interface lines as the ST10F280. During this time the ST10F280 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 ST10F280 needs access to its external bus while it is occupied by another bus master, it demands it via the BREQ output.

The external bus arbitration is enabled by setting bit HLDEN in register PSW to '1'. In this case the three bus arbitration pins HOLD, HLDA and BREQ are automatically controlled by the EBC independent of their I/O configuration. Bit HLDEN may be cleared during the execution of program sequences, where the external resources are required but cannot be shared with other bus masters. In this case the ST10F280 will not answer to HOLD requests from other external masters. If HLDEN is cleared while the ST10F280 is in hold state (code execution from internal RAM/Flash) this hold state is left only after HOLD has been deactivated again. In this case the current hold state continues and only the next HOLD request is not answered.

Connecting two ST10F280's in this way would require additional logic to combine the respective output signals HLDA and BREQ. This can be avoided by switching one of the controllers into slave mode where pin HLDA is switched to input.

This allows to directly connect the slave controller to another master controller without glue logic. The slave mode is selected by setting bit DP6.7 to'1'. DP6.7='0' (default after reset) selects the Master Mode.

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

9.6.1 - Connecting Bus Masters

When multiple ST10F280's or a ST10F280 and another bus master shall share external resources some glue logic is required that defines the currently active bus master and also enables a ST10F280 which has surrendered its bus interface to regain control of it in case it must access the shared external resources.

This glue logic is required if the other bus master does not automatically remove its hold request after having used the shared resources.

When two ST10F280 are connected in this way the external glue logic can be left out. In this case one of the controllers must be operated in its master mode (default after reset, DP6.7='0') while the other one must be operated in its slave mode (selected with DP6.7='1').

In slave mode the ST10F280 inverts the direction of its HLDA pin and uses it as an input, while the master's HLDA pin remains an output. This approach does not require any additional glue logic for the bus arbitration (see Figure 67).

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

9.6.2 - Entering the Hold State

Access to the ST10F280's external bus is requested by driving its HOLD input low. After synchronizing this signal the ST10F280 will complete a current external bus cycle (if any is active), release the external bus and grant access to it by driving the HLDA output low. During hold state the ST10F280 treats the external bus interface as follows:

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

Should the ST10F280 require access to its external bus during hold mode, it activates its bus request output BREQ to notify the arbitration circuitry. BREQ is activated only during hold mode. It will be inactive during normal operation (see Figure 68).

Figure 67 : Sharing external resources using slave mode

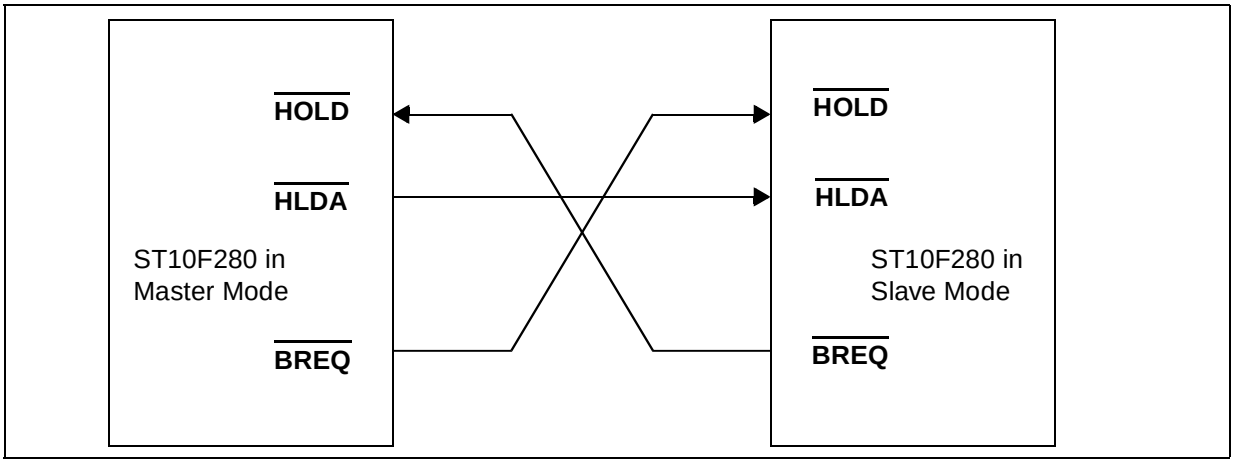
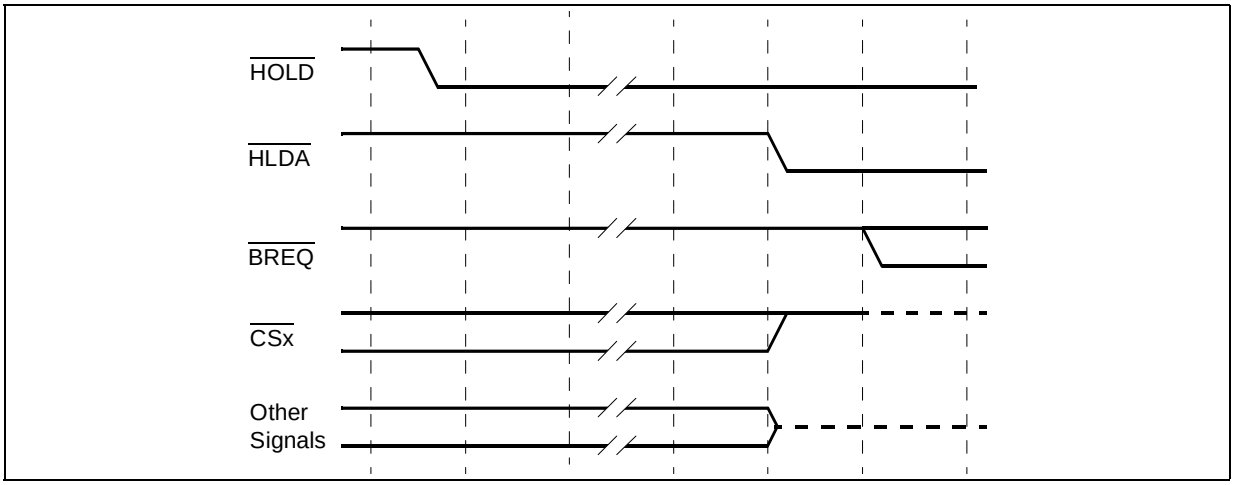


Figure 68 : External bus arbitration, releasing the bus



Note The ST10F280 will complete the currently running bus cycle before granting bus access as indicated by the broken lines. This may delay hold acknowledge compared to this figure. The figure above shows the first possibility for BREQ to get active. During bus hold pin P3.12 is switched back to its standard function and is then controlled by DP3.12 and P3.12. Keep DP3.12 = '0' in this case to ensure floating in hold mode.

9.6.3 - Exiting the Hold State

The external bus master returns the access rights to the ST10F280 by driving the HOLD input high. After synchronizing this signal the ST10F280 will drive the HLDA output high, actively drive the control signals and resume executing external bus cycles if required. Depending on the arbitration logic, the external bus can be returned to the ST10F280 under two circumstances:

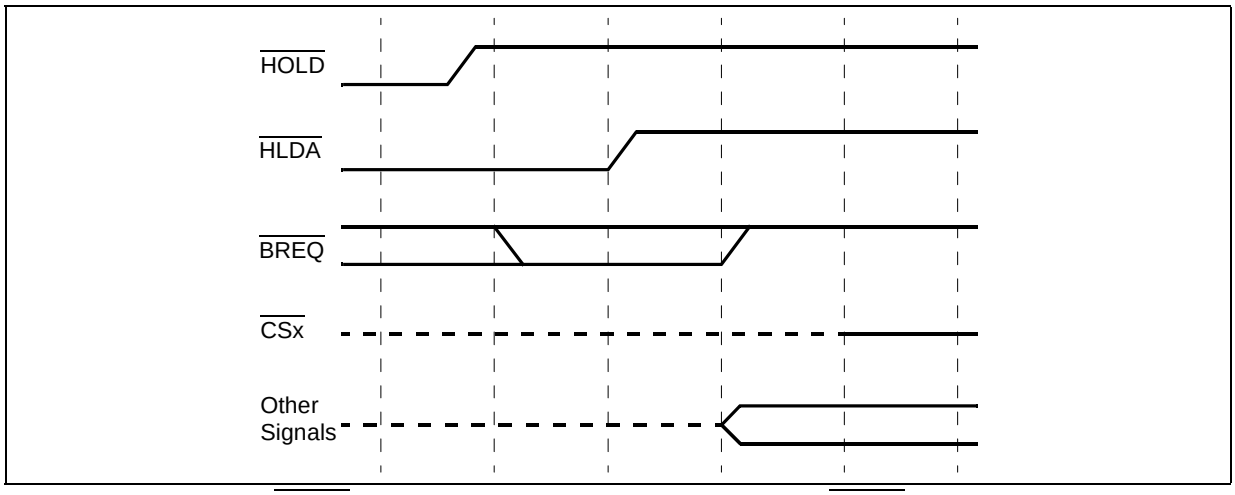
– The external master does no more require access to the shared resources and gives up its own access rights.

– The ST10F280 needs access to the shared resources and demands this by activating its BREQ output.

The arbitration logic may then deactivate the other master's HLDA and so free the external bus for the ST10F280, depending on the priority of the different masters.

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

Figure 69 : External bus arbitration, (regaining the bus)



Note The falling $\overline{\text{BREQ}}$ edge shows the last chance for $\overline{\text{BREQ}}$ to trigger the indicated regain-sequence. Even if $\overline{\text{BREQ}}$ is activated earlier the regain-sequence is initiated by HOLD going high. $\overline{\text{BREQ}}$ and $\overline{\text{HOLD}}$ are connected via an external arbitration circuitry. Please note that $\overline{\text{HOLD}}$ may also be deactivated without the ST10F280 requesting the bus.

9.7 - The XBUS Interface

The ST10F280 provides an on-chip interface (the XBUS interface), which allows to connect integrated costumer / application specific peripherals to the standard controller core.

The XBUS is an internal representation of the external bus interface, it works 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 byte wide or word wide, with or without a separate address bus.

Visibility of XBUS Peripherals

In order to keep the ST10F280 compatible with the ST10C167 and with the ST10F167, the XBUS peripherals can be selected to be visible and / or accessible on the external address / data bus. CAN1EN and CAN2EN bit of XPERCON register must be set. If these bits are cleared before the global enabling with XPEN-bit in SYSCON register, the corresponding address space, port pins and interrupts are not occupied by the peripheral, thus the peripheral is not visible and not available. Refer to Chapter 21 - Register Set.

XPERCON (F024h / 12h) ESFR Reset Value: - - 05h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	XPWMEN	XPERCONEN3	XRAMEN	CAN2EN	CAN1EN
											RW	RW	RW	RW	RW

Bit	Function
CAN1EN	**CAN1 Enable Bit** 0 Accesses to the on-chip CAN1 XPeripheral and its functions are disabled. P4.5 and P4.6 pins can be used as general purpose I/Os. Address range 00'EF00h-00'EFFFh is only directed to external memory if CAN2EN and XPWM bits are cleared also. 1 The on-chip CAN1 XPeripheral is enabled and can be accessed.
CAN2EN	**CAN2 Enable Bit** 0 Accesses to the on-chip CAN2 XPeripheral and its functions are disabled. P4.4 and P4.7 pins can be used as general purpose I/Os. Address range 00'EE00h-00'EEFFh is only directed to external memory if CAN1EN and XPWM bits are cleared also. 1 The on-chip CAN2 XPeripheral is enabled and can be accessed.
XRAMEN	**XRAM Enable Bit** 0 Accesses to the on-chip 16K Byte XRAM are disabled, external access performed. 1 The on-chip 16K Byte XRAM is enabled and can be accessed.
XPERCONEN3	**XPORT9, XTIMER, XPORT10, XADCMUX Enable Bit** 0 Accesses to the XPORT9, XTIMER, XPORT10, XADCMUX peripherals are disabled, external access performed. 1 The on-chip XPORT9, XTIMER, XPORT10, XADCMUX peripherals are enabled and can be accessed.
XPWMEN	**XPWM Enable Bit** 0 Accesses to the on-chip XPWM are disabled, external access performed. Address range 00'EC00h-00'ECFFh is only directed to external memory if CAN1EN and CAN2EN are '0' also 1 The on-chip XPWM is enabled and can be accessed.

Note: - When both CAN and XPWM are disabled via XPERCON setting, then any access in the address range 00'EC00h 00'EFFFh will be directed to external memory interface, using the BUSCONx register corresponding to address matching ADDRSELx register. P4.4 and P4.7 can be used as General Purpose I/O when CAN2 is not enabled, and P4.5 and P4.6 can be used as General Purpose I/O when CAN1 is not enabled.

- The default XPER selection after Reset is : XCAN1 is enabled, XCAN2 is disabled, XRAM is enabled, XPORT9, XTIMER, XPORT10, XPWM, XADCMUX are disabled.

- Register XPERCON cannot be changed after the global enabling of XPeripherals, i.e. after setting of bit XPEN in SYSCON register.

10 - THE GENERAL PURPOSE TIMER UNITS

The general purpose timer units GPT1 and GPT2 are flexible multifunctional timer structures which may be used for timing, event counting, pulse width measurement, pulse generation, frequency multiplication, and other purposes. They incorporate five 16-bit timers that are grouped into the two timer blocks GPT1 and GPT2.

Block GPT1 contains 3 timers/counters with a maximum resolution of 8 CPU clock cycles, while block GPT2 contains 2 timers/counters with a maximum resolution of 4 CPU clock cycles and a 16-bit Capture/ Reload register (CAPREL). Each timer in each block may operate independently in a number of different modes such as gated timer or counter mode, or may be concatenated with another timer of the same block.

The auxiliary timers of GPT1 may optionally be configured as reload or as capture registers for the core timer. In the GPT2 block, the additional CAPREL register supports capture and reload operation with extended functionality, and its core timer T6 may be concatenated with timers of the CAPCOM units (T0, T1, T7 and T8). Each block has alternate input/output functions and specific interrupts associated with it.

10.1 - Timer Block GPT1

From a programmer's point of view, the GPT1 block is composed of a set of SFRs. Those portions of port and direction registers which are used for alternate functions by the GPT1 block are named by "Y" in Figure 70.

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 Port3, 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 bit-addressable 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 70 : SFRs and port pins associated with timer block GPT1

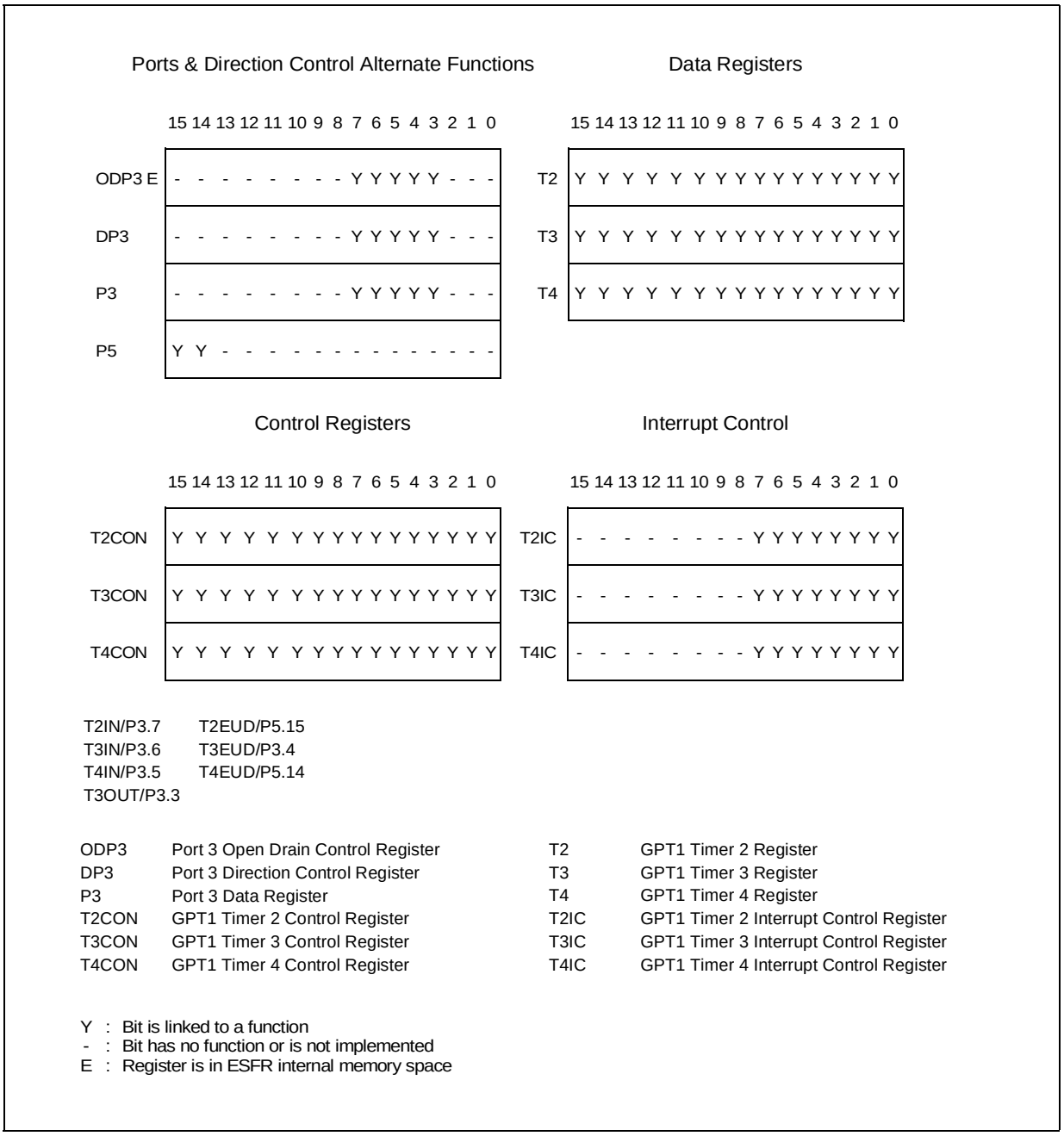
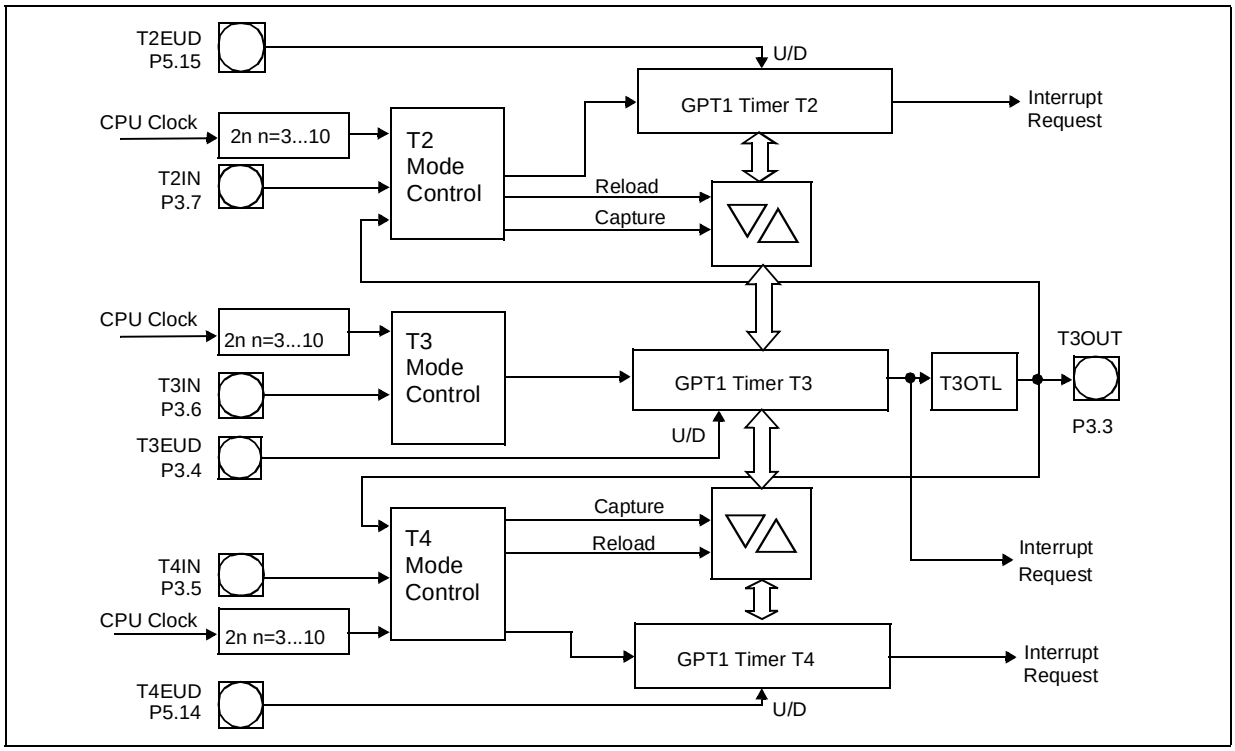


Figure 71 : GPT1 block diagram



10.1.1 - GPT1 Core Timer T3

The core timer T3 is configured and controlled via 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
-	-	-	-	-	**T3OTL**	T3OE	T3UDE	T3UD	T3R	T3M			T3I		
					RW	RW	RW	RW	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 1 0: Incremental interface mode 1 X X: Reserved. Do not use other combination (except 110).
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 Disabled - T3OE = '1':Alternate Output Function Enabled
T3OTL	**Timer 3 Output Toggle Latch** - Toggles on each overflow / underflow of T3. Can be set or reset by software.

Note 1. For the effects of bit T3UD and T3UDE refer to the direction Table 30.

Timer 3 Run bit

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

Count Direction Control

The count direction of the core timer can be controlled either by software or by the external input pin T3EUD (Timer T3 External Up/Down Control Input), which is the alternate input function of port pin P3.4.

These options are selected by bit T3UD and T3UDE in control register T3CON. When the up/down control is done by software (bit T3UDE='0'), the count direction can be altered by setting or clearing bit T3UD.

When T3UDE='1', pin T3EUD is selected to be the controlling source of the count direction. However, bit T3UD can still be used to reverse the actual count direction, as shown in the Table 30.

If T3UD='0' and pin T3EUD is at low level, the timer is counting up. With a high level at T3EUD the timer is counting down.

If T3UD='1', a high level at pin T3EUD specifies counting up, and a low level specifies counting down. The count direction can be changed regardless of whether the timer is running or not.

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

Table 30 : GPT1 core timer T3 count direction control

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

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

Timer 3 Output Toggle Latch

An overflow or underflow of timer T3 will clock the toggle bit T3OTL in control register T3CON. T3OTL can also be set or reset by software.

Bit T3OE (Alternate Output Function Enable) in register T3CON enables the state of T3OTL to be an alternate function of the external output pin T3OUT/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 output by setting direction control bit DP3.3 to '1'. If T3OE='1', pin T3OUT then outputs the state of T3OTL. If T3OE='0', pin T3OUT can be used as general purpose I/O pin.

In addition, T3OTL can be used in conjunction with the timer over/underflows as an input for the counter function or as a trigger source for the reload function of the auxiliary timers T2 and T4.

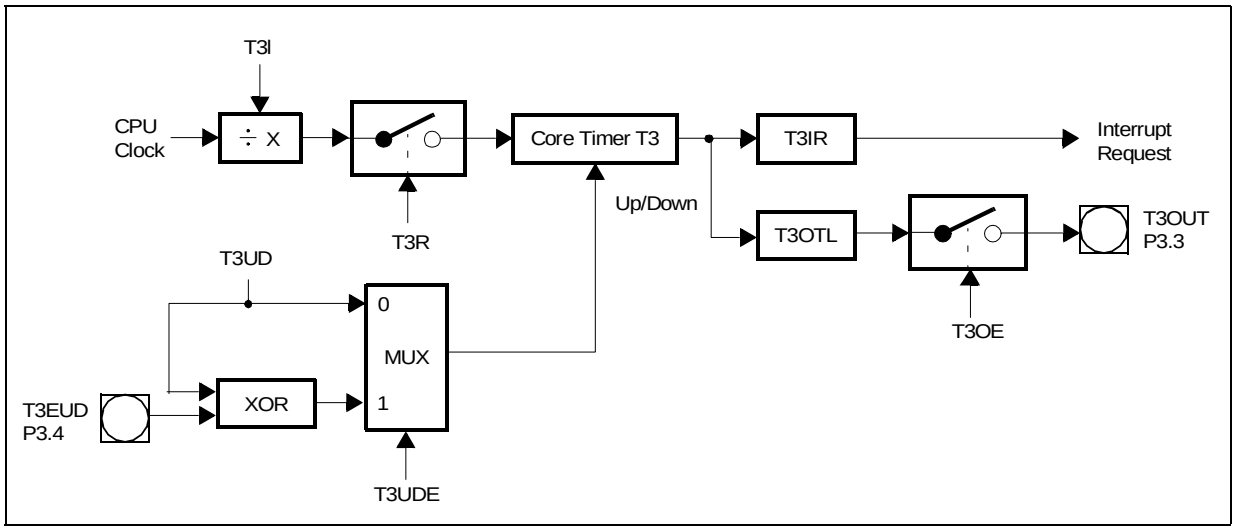
For this purpose, the state of T3OTL does not have to be available at pin T3OUT, because an internal connection is provided for this option.

Timer 3 in Timer Mode

Timer mode for the core timer T3 is selected by setting bit-field T3M in register T3CON to '000b'. In this mode, T3 is clocked with the internal system clock (CPU clock) divided by a programmable pre-scaler, which is selected by bit-field T3I.

The input frequency f_{T3} for timer T3 and its resolution r_{T3} are scaled linearly with lower clock frequencies f_{CPU}, as can be seen from the following formula:

Figure 72 : Core timer T3 in timer mode



$$f_{T3} = \frac{f_{CPU}}{8 \times 2^{(T3I)}}$$

$$r_{T3}\ [\mu s] = \frac{8 \times 2^{(T3I)}}{f_{CPU}\ [MHz]}$$

The timer resolutions which result from the selected pre-scaler option are listed in the Table 31. 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.

Table 31 : GPT1 timer resolutions

	Timer Input Selection T2I / T3I / T4I							
	000b	001b	010b	011b	100b	101b	110b	111b
Pre-scaler factor	8	16	32	64	128	256	512	1024
Resolution in CPU clock cycles	8	16	32	64	128	256	512	1024

Refer to the device datasheet for a table of timer input frequencies, resolution and periods for the range of pre-scaler options.

Timer 3 in Gated Timer Mode

Gated timer mode for the core timer T3 is selected by setting bit-field T3M in register T3CON to '010b' or '011b'. 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, and direction control bit DP3.6 must contain '0' (see Figure 73). If T3M.0='0', the timer is enabled when T3IN shows a low level. A high level at this pin stops the timer. If T3M.0='1', pin T3IN must have a high level in order to enable the timer. In addition, the timer can be turned on or off by software using bit T3R. The timer will only run, if T3R='1' and the gate is active. It will stop, if either T3R='0' or the gate is inactive.

Note A transition of the gate signal at pin T3IN does not cause an interrupt request.

Timer 3 in Counter Mode

Counter mode for the core timer T3 is selected by setting bit-field T3M in register T3CON to '001b'. 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 register T3CON selects the triggering transition (see Table 32).

Figure 73 : Core timer T3 in gated timer mode

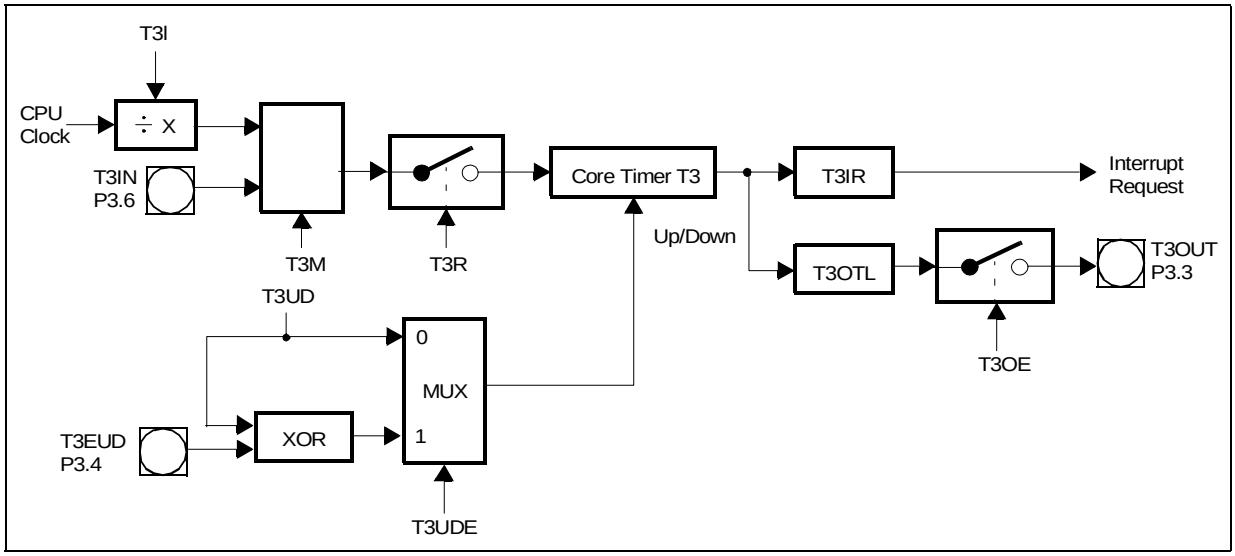


Figure 74 : Core timer T3 in counter mode

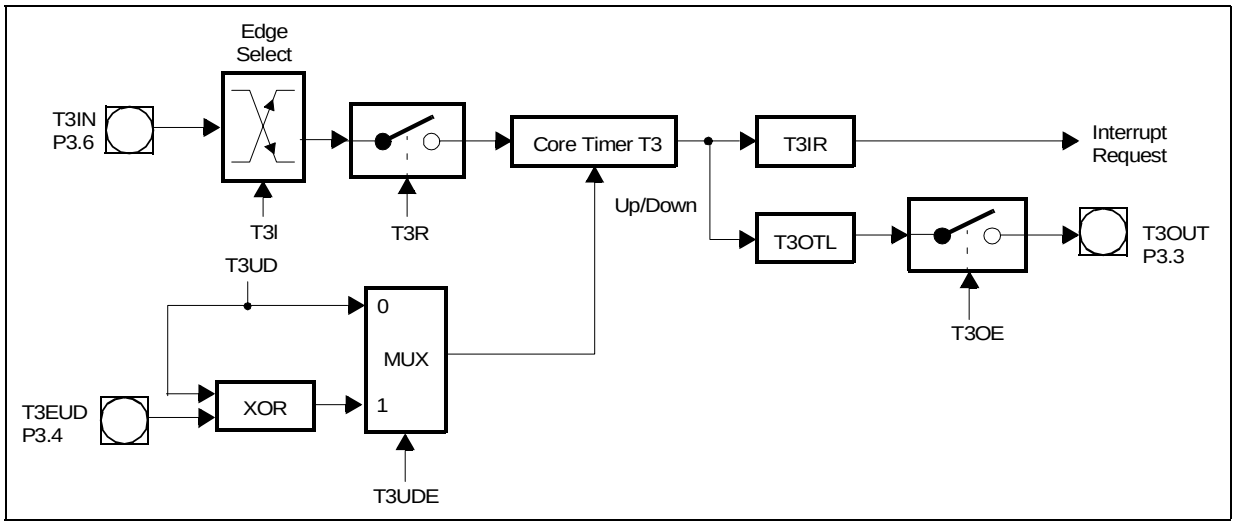


Table 32 : GPT1 core timer T3 (counter mode) input edge selection

T3I	Triggering Edge for Counter Increment / Decrement
0 0 0	None. Counter T3 is disabled
0 0 1	Positive transition (rising edge) on T3IN
0 1 0	Negative transition (falling edge) on T3IN
0 1 1	Any transition (rising or falling edge) on T3IN
1 X X	Reserved. Do not use this combination

For counter operation, pin T3IN/P3.6 must be configured as input, and 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.

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 (see Figure 75).

Bit-field T3I in control register T3CON selects the triggering transitions (see Table 33). In this mode the sequence of the transitions of the two input signals is evaluated and generates count pulses as well as the direction signal.

So T3 is modified automatically according to the speed and the direction of the incremental encoder and its contents, therefore, always represent the encoder's current position.

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

The third encoder output "T0 $\overline{T0}$" which indicates the mechanical zero position, may be connected to an external interrupt input and trigger a reset timer T3 (for example, via PEC transfer from ZEROS) (see Figure 76).

Figure 75 : Core timer T3 in incremental interface mode

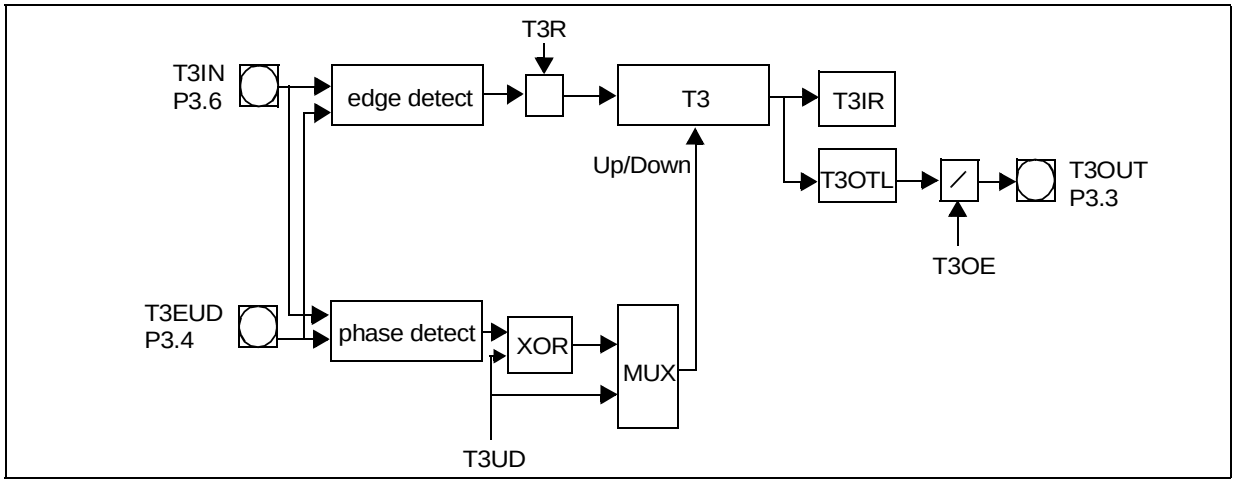
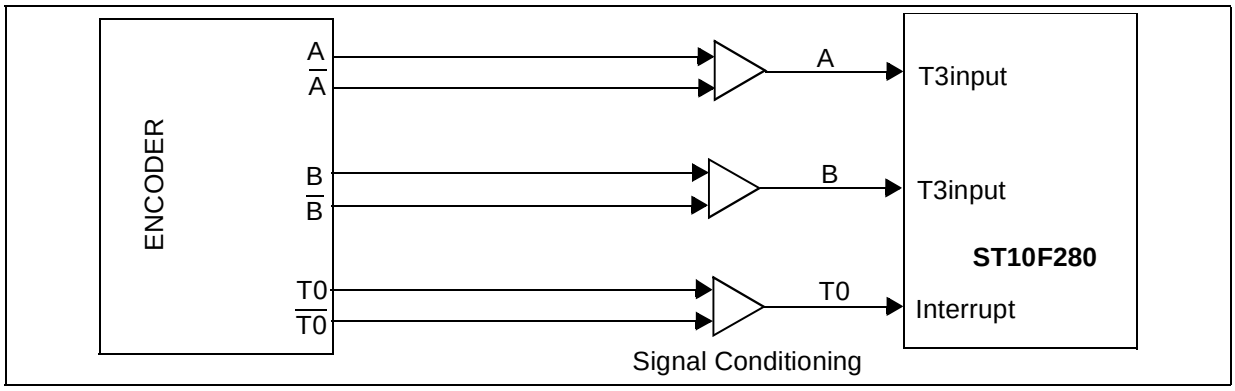


Table 33 : GPT1 core timer T3 (incremental interface mode) input edge selection

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

Figure 76 : Connection of the encoder to the ST10F280



For incremental interface operation the following conditions must be met

– Bit-field T3M must be '110b'

– Both pins T3IN and T3EUD must be configured as input, at the respective direction control bit with '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.

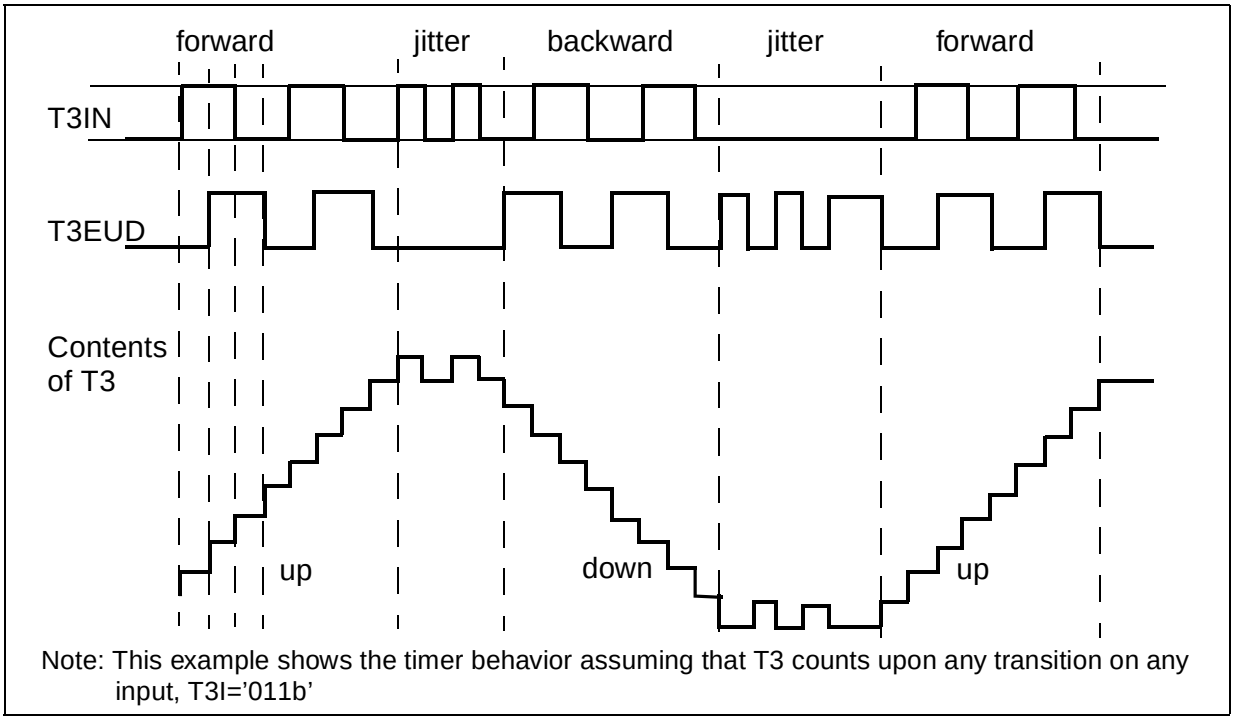
In incremental interface mode, the count direction is automatically derived from the sequence in which the input signals change.

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

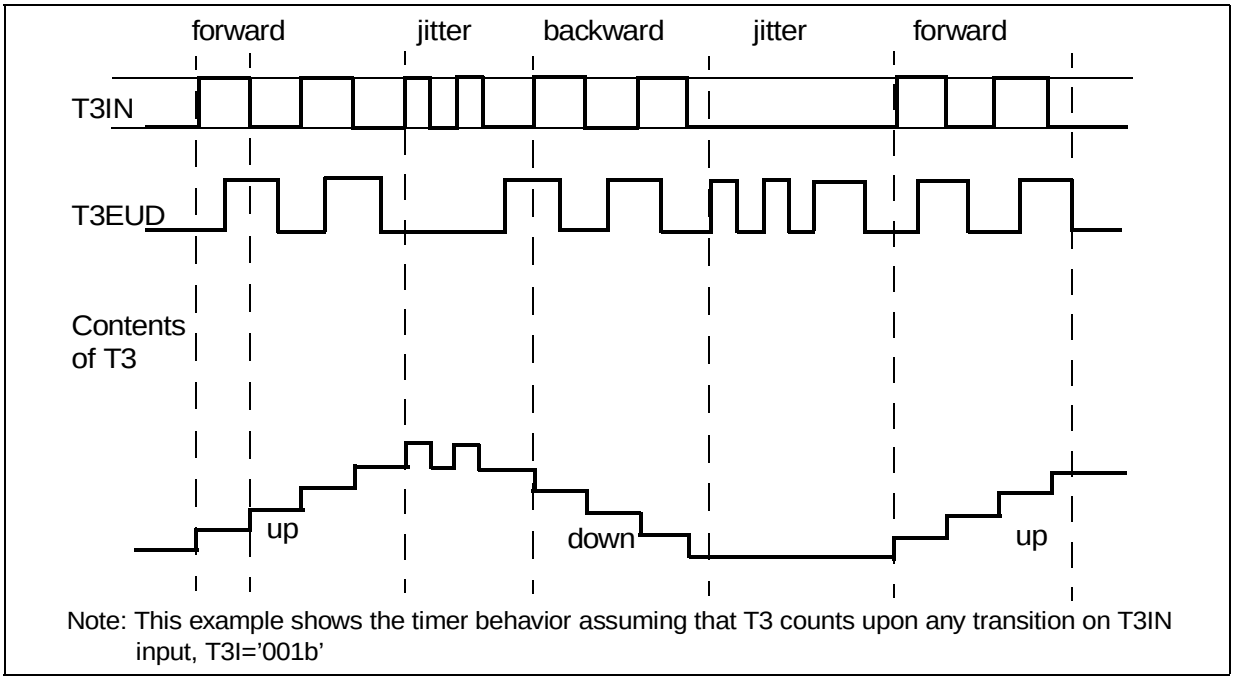
The Figure 77 gives examples of T3's operation, visualizing count signal generation and direction control. It also shows how input jitter is compensated. This might occur if the sensor stays near to one of the switching points.

Figure 77 : Evaluation of the incremental encoder signals



Note: This example shows the timer behavior assuming that T3 counts upon any transition on any input, T3I='011b'

Figure 78 : Evaluation of the incremental encoder signals



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

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 an additional special capture mode for timer T5.

10.1.2 - GPT1 Auxiliary Timers T2 and T4

Both auxiliary timers T2 and T4 have exactly the same functionality. They can be configured like 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. 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 the 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
-	-	-	-	-	-	-	T2UDE	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
-	-	-	-	-	-	-	T4UDE	T4UD	T4R	T4M			T4I		
							RW	RW	RW	RW			RW		

Bit	Function
TxI	**Timer x Input Selection** Depends on the Operating Mode, see respective sections.
TxM	**Timer x Mode Control (Basic Operating Mode)** 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]
TxUDE	**Timer x External Up/Down Enable** [1]

Note 1. For th*e effects of bit TxUD and TxUDE refer to the direction Table 33 in T3 section.*

Count Direction Control for Auxiliary Timers

The count direction of the auxiliary timers can be controlled in the same way as for the core timer T3. The description and the table apply accordingly.

Timers T2 and T4 in Timer Mode or Gated Timer Mode

When the auxiliary timers T2 and T4 are programmed to timer mode or gated timer mode, their operation is the same as described for the core timer T3. The descriptions, figures and tables apply accordingly with one exception: There is no output toggle latch 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 register TxCON to '001b'. 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 (see Figure 79).

The event causing an increment or decrement of a timer can be a positive, a negative, or both a positive and a negative transition at either the respective input pin, or at the toggle latch T3OTL. Bit-field TxI in the respective control register TxCON selects the triggering transition (see Table 34).

Note Only 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, 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.

Figure 79 : Auxiliary timer in counter mode

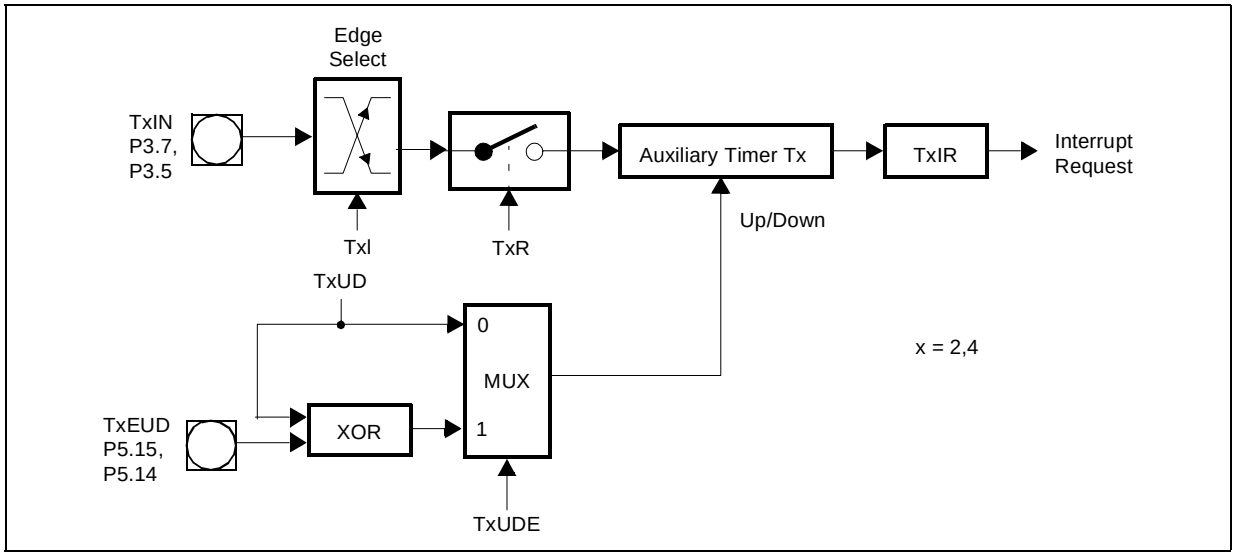


Table 34 : GPT1 auxiliary timer (counter mode) input edge selection

T2I / T4I	Triggering Edge for Counter Increment / Decrement
X 0 0	None. Counter Tx is disabled
0 0 1	Positive transition (rising edge) on TxIN
0 1 0	Negative transition (falling edge) on TxIN
0 1 1	Any transition (rising or falling edge) on TxIN
1 0 1	Positive transition (rising edge) of output toggle latch T3OTL
1 1 0	Negative transition (falling edge) of output toggle latch T3OTL
1 1 1	Any transition (rising or falling edge) of output toggle latch T3OTL

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.

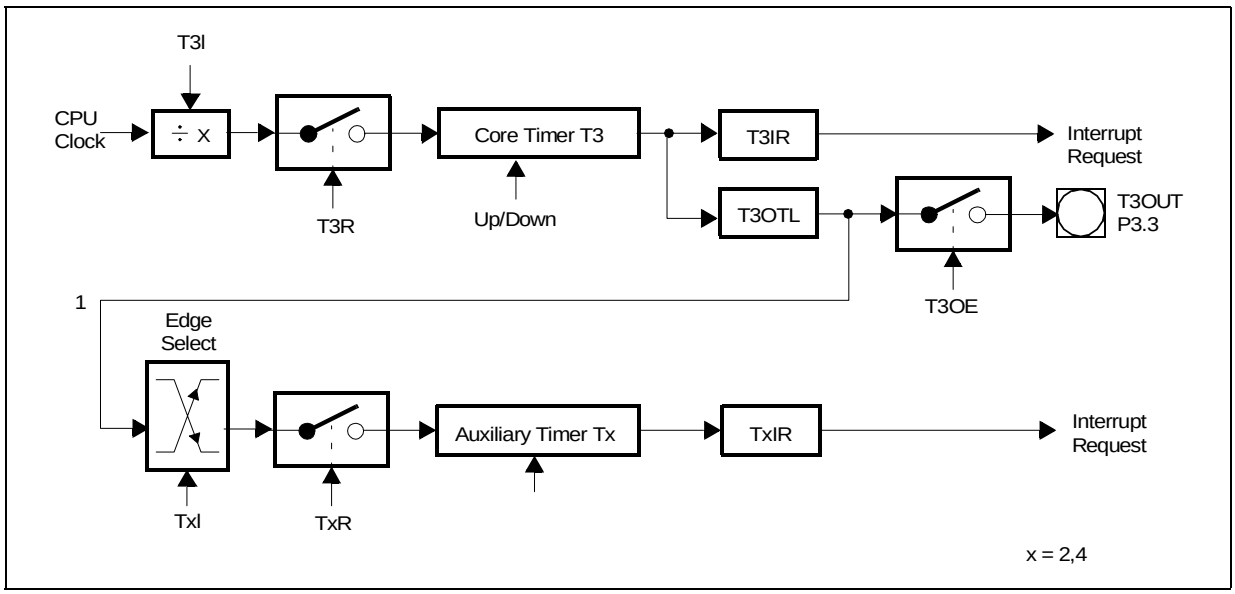
T3 can operate in timer, gated timer or counter mode in this case (see Figure 80).

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 '100b'. 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 34). A transition of the auxiliary timer's input or the output toggle latch T3OTL may trigger the reload.

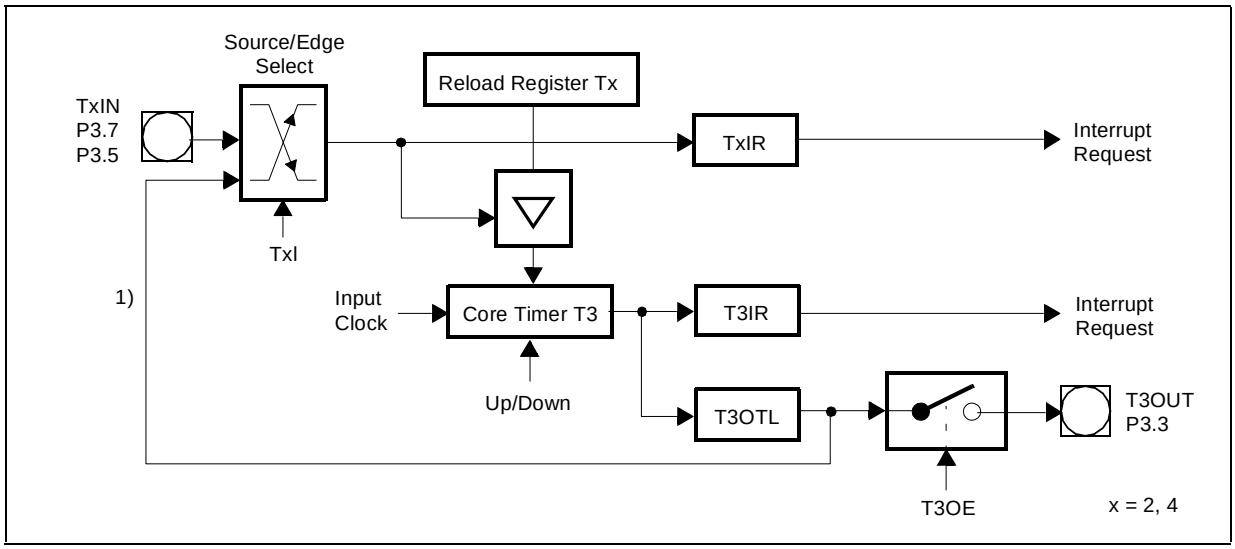
Note When programmed for reload mode, the respective auxiliary timer (T2 or T4) stops independent of its run flag T2R or T4R.

Figure 80 : Concatenation of core timer T3 and an auxiliary timer



Note 1. Line only affected by over/underflows of T3, but NOT by software modifications of T3OTL.

Figure 81 : GPT1 auxiliary timer in reload mode



Note 1. Line only affected by over/underflows of T3, but NOT by software modifications of T3OTL.

Upon a trigger signal T3 is loaded with the contents of the respective timer register (T2 or T4) and the interrupt request flag (T2IR or T4IR) is set.

Note When a T3OTL transition is selected for the trigger signal, also the interrupt request flag T3IR will be set upon a trigger, indicating T3's overflow or underflow.
Modifications of T3OTL via software will NOT trigger the counter function of T2/T4.

The reload mode triggered by T3OTL can be used in a number of different configurations. Depending on the selected active transition the following functions can be performed:

– If both a positive and a negative transition of T3OTL is selected to trigger a reload, the core timer will be reloaded with the contents of the auxiliary timer each time it overflows or underflows. This is the standard reload mode (reload on overflow/underflow).
– If either a positive or a negative transition of T3OTL is selected to trigger a reload, the core timer will be reloaded with the contents of the auxiliary timer on every second overflow or underflow.
– Using this "single-transition" mode for both auxiliary timers allows to perform very flexible pulse width modulation (PWM). One of the auxiliary timers is programmed to reload the core timer on a positive transition of T3OTL, the other is programmed for a reload on a negative transition of T3OTL. With this combination the core timer is alternately reloaded from the two auxiliary timers.

Figure 82 shows an example for the generation of a PWM signal using the alternate reload mechanism.

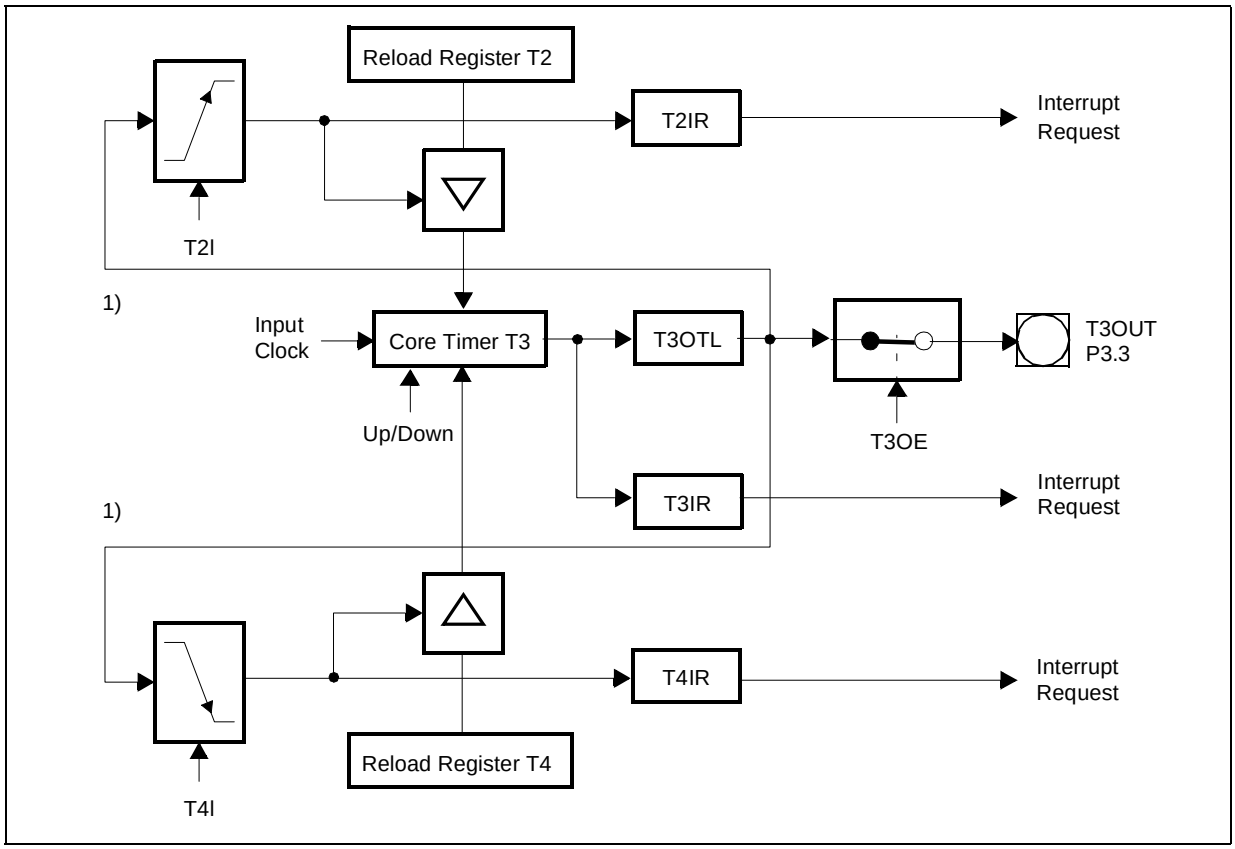
T2 defines the high time of the PWM signal (reloaded on positive transitions) and T4 defines the low time of the PWM signal (reloaded on negative transitions).

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.

Avoid selecting the same reload trigger event for both auxiliary timers as both reload registers will try to load the core timer at the same time. If this happens, T2 is disregarded and the contents of T4 is reloaded.

Figure 82 : GPT1 timer reload configuration for PWM generation



Note 1) Lines only affected by over/underflows of T3, but NOT by software modifications of T3OTL.

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 '101b'.

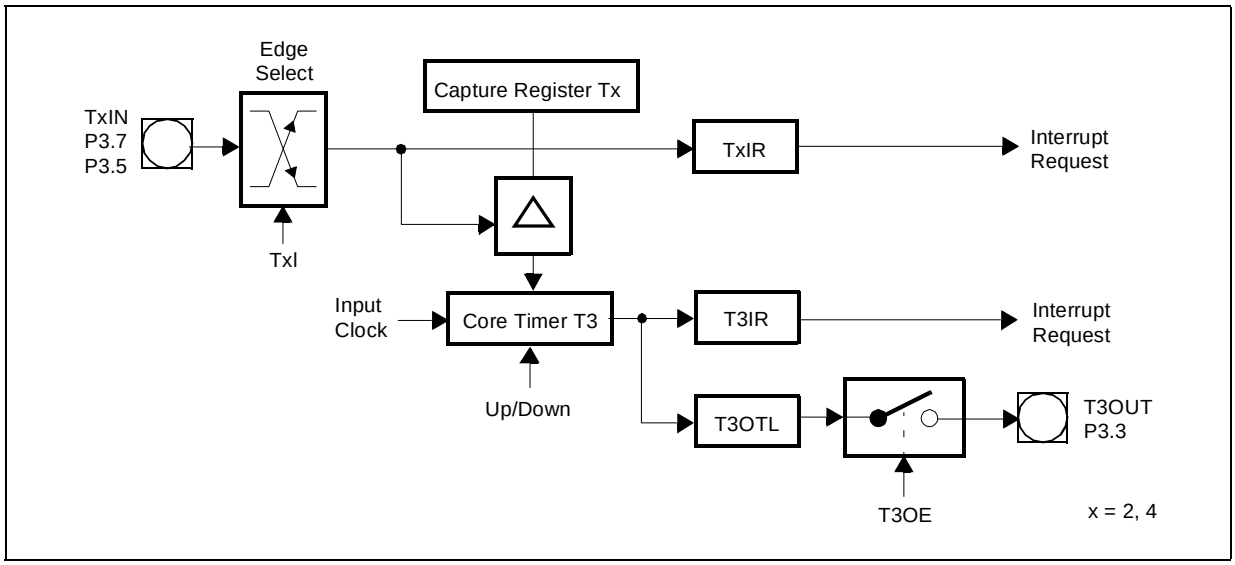
In capture mode the contents of the core timer are latched into an auxiliary timer register in response to a signal transition at the respective auxiliary timer's external input pin TxIN.

The capture trigger signal can be a positive, a negative, or both a positive and a negative transition.

The two least significant bit 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 83 : 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 bit 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.

10.1.3 - Interrupt Control for GPT1 Timers

When a timer overflows from FFFFh to 0000h (when counting up), or when it underflows from 0000h to FFFFh (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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	**T2IR**	T2IE	ILVL				GLVL	
								RW	RW	RW				RW	

T3IC (FF62h / B1h) SFR Reset Value: - - 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	**T3IR**	T3IE	ILVL				GLVL	
								RW	RW	RW				RW	

T4IC (FF64h / B2h) SFR Reset Value: - - 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	**T4IR**	T4IE	ILVL				GLVL	
								RW	RW	RW				RW	

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

10.2 - Timer Block GPT2

From a programmer's point of view, the GPT2 block is represented by a set of SFRs. The I/O of port and direction registers which are used for alternate functions by the GPT2 block are noted "Y" in Figure 84.

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 associated input pin 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 bit-addressable SFRs T5 and T6.

Figure 84 : SFRs and port pins associated with timer block GPT2

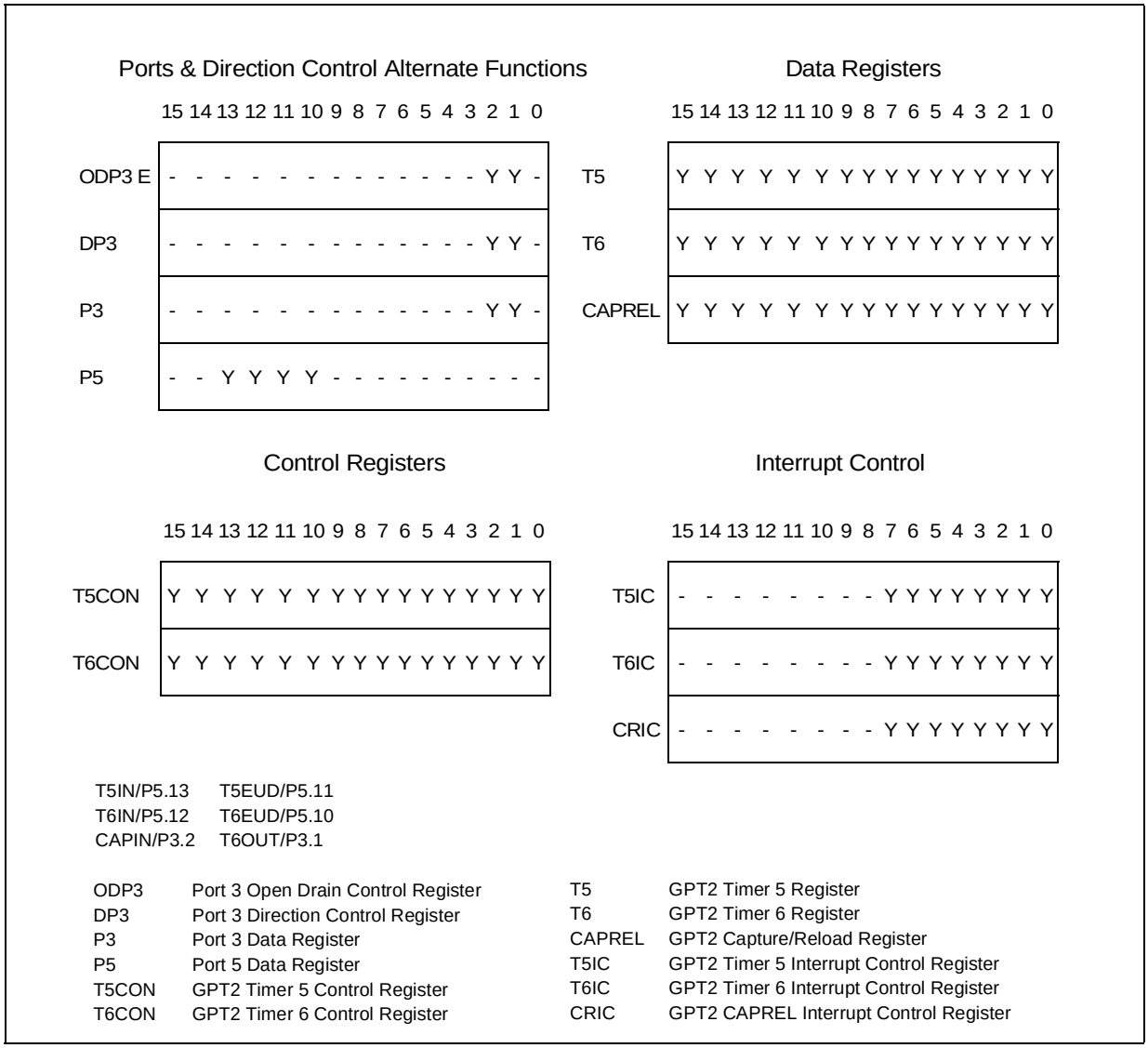
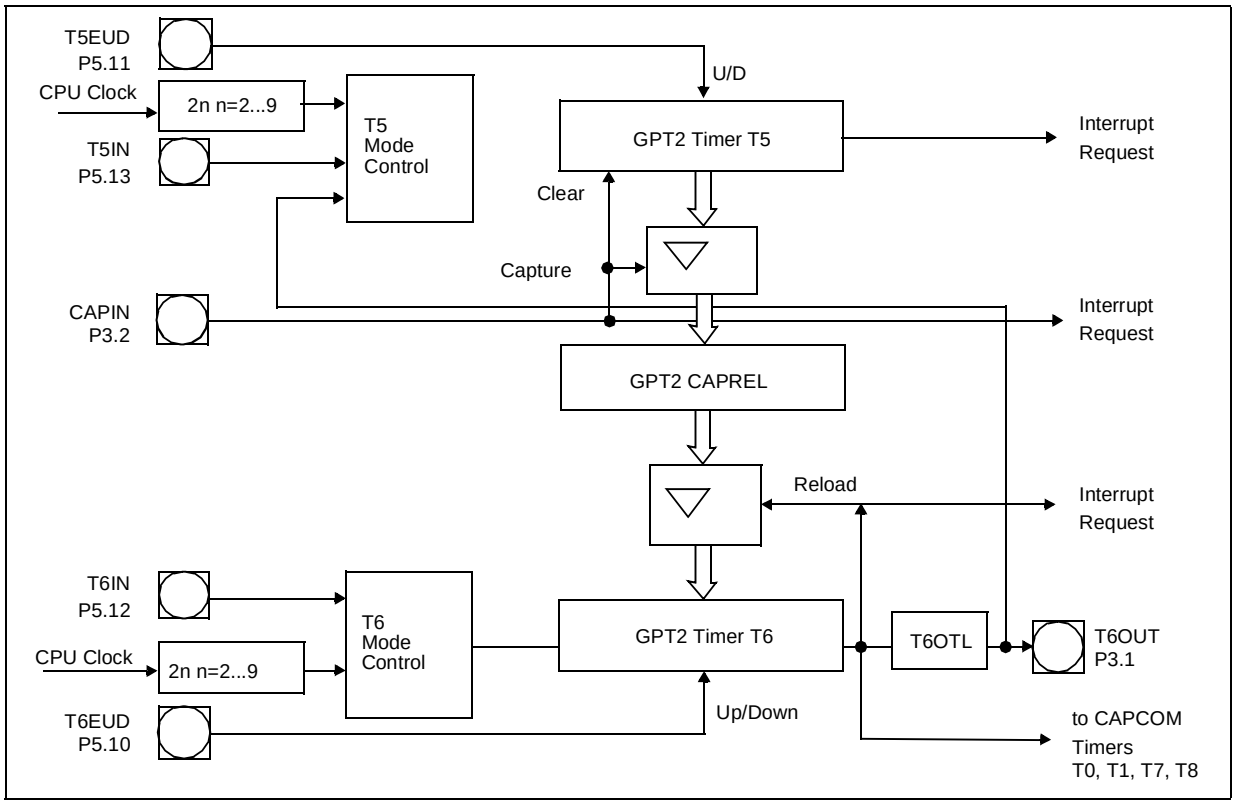


Figure 85 : GPT2 block diagram



10.2.1 - 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	-	-	-	-	**T6OTL**	T6OE	T6UDE	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.
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]

Bit	Function
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

Note 1. For the effects of bit T6UD and T6UDE refer to the direction Table 35.

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 bit 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 35. 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.

Table 35 : GPT2 core timer T6 count direction control

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

Note The direction control works the same for core timer T6 and for auxiliary timer T5. Therefore the pins and bit 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 I/O pin.

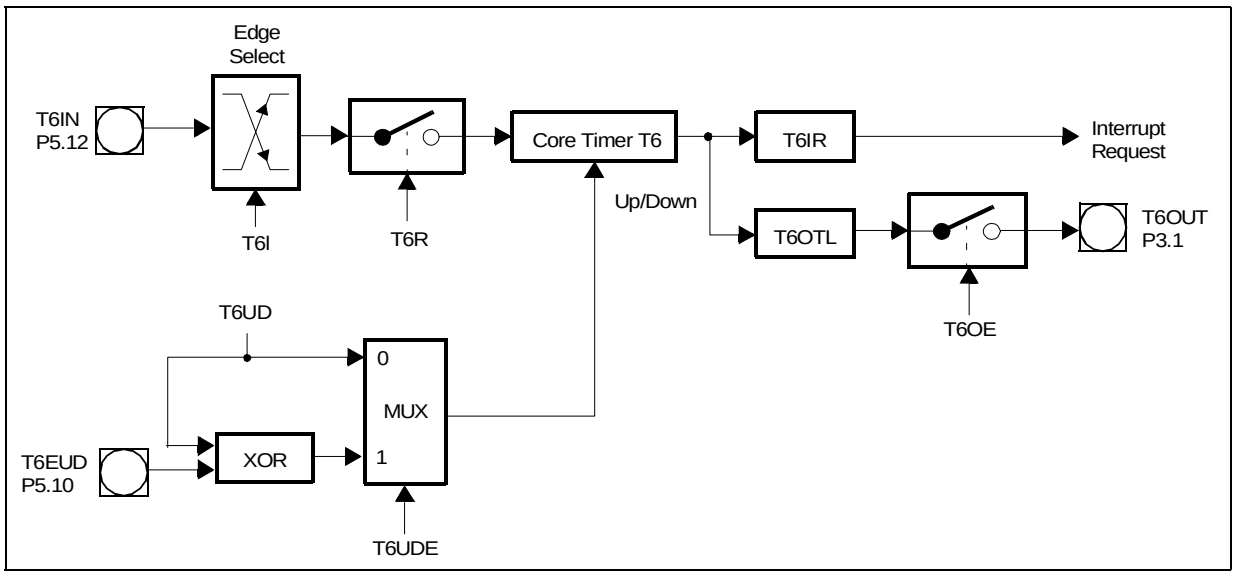
In addition, T6OTL can be used in conjunction with the timer over/underflows as an input for the counter function of the auxiliary timer T5. For this purpose, the state of T6OTL does not have to be available at pin T6OUT, because an internal connection is provided for this option.

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 '000b'. 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:

Figure 86 : Block diagram of core timer T6 in timer mode



$$f_{T6} = \frac{f_{CPU}}{4 \times 2^{(T6I)}} \qquad r_{T6}\,[\mu s] = \frac{4 \times 2^{(T6I)}}{f_{CPU}\,[MHz]}$$

The timer resolutions which result from the selected pre-scaler option are listed in the Table 36. This table also applies to the Gated Timer Mode of T6 and to the auxiliary timer T5 in timer and gated timer mode.

Table 36 : GPT2 timer resolution

	Timer Input Selection T5I / T6I							
	000b	001b	010b	011b	100b	101b	110b	111b
Pre-scaler factor	4	8	16	32	64	128	256	512
Resolution in CPU clock cycles	4	8	16	32	64	128	256	512

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 Mode

Gated timer mode for the core timer T6 is selected by setting bit-field T6M in register T6CON to '010b' or '011b'. 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 (see Figure 87). 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 '001b'. 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 37).

Figure 87 : Block diagram of core timer T6 in gated timer mode

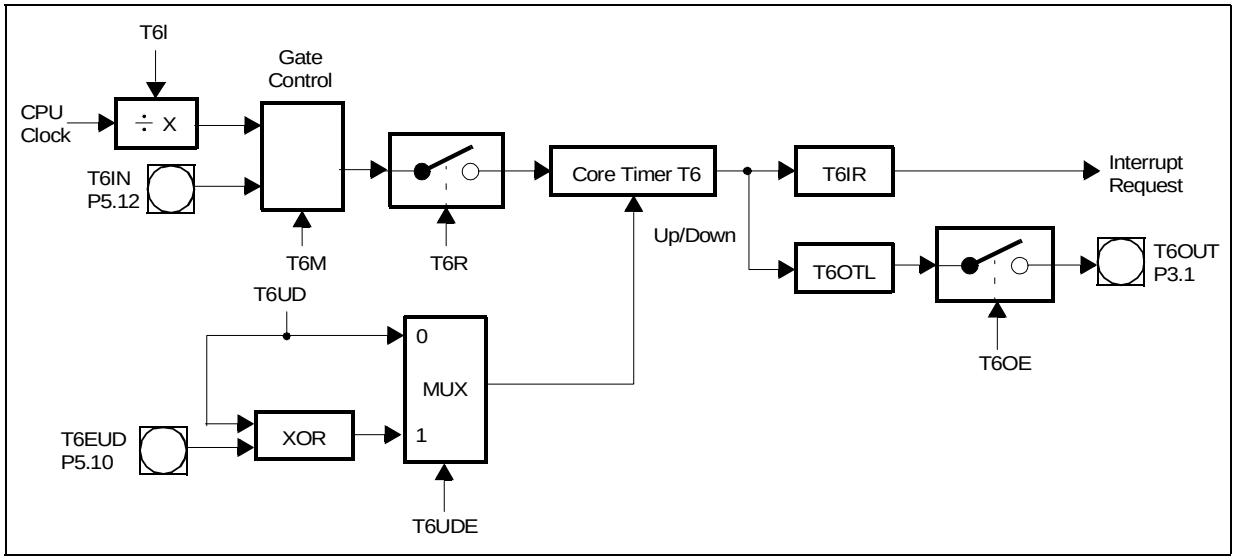


Figure 88 : Block diagram of core timer T6 in counter mode

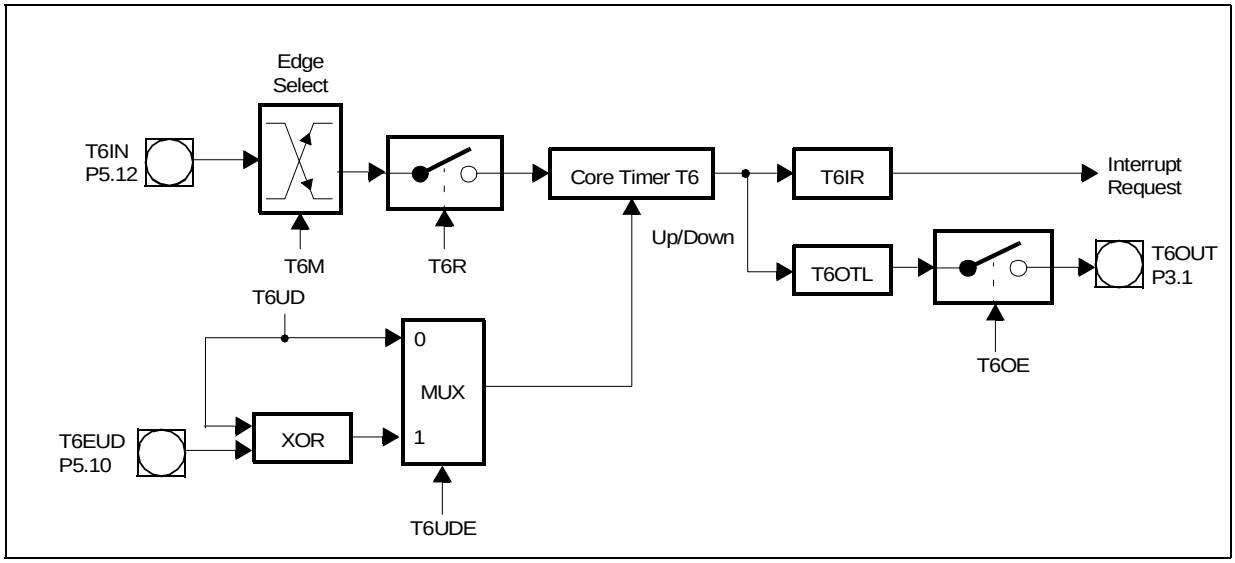


Table 37 : GPT2 core timer T6 (counter mode) input edge selection

T6I	Triggering Edge for Counter Increment / Decrement
0 0 0	None. Counter T6 is disabled
0 0 1	Positive transition (rising edge) on T6IN
0 1 0	Negative transition (falling edge) on T6IN
0 1 1	Any transition (rising or falling edge) on T6IN
1 X X	Reserved. Do not use this combination

The maximum input frequency 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.

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 bit-addressable 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	T5CLR	CI		-	-	CT3	T5UDE	T5UD	T5R	-	T5M		T5I		
RW	RW	RW				RW	RW	RW	RW		RW		RW		

Bit	Function
T5I	**Timer 5 Input Selection** Depends on the Operating Mode, see respective sections.
T5M	**Timer 5 Mode Control (Basic Operating Mode)** 0 0: Timer Mode 0 1: Counter Mode 1 0: 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]
CT3	**Capture Trigger 3** 0: Inactive 1: Capture on Timer3 events
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
T5SC	**Timer 5 Capture Mode Enable** T5SC = '0': Capture into register CAPREL Disabled T5SC = '1': Capture into register CAPREL Enabled

Note 1. For the effects of bit TxUD and TxUDE refer to the direction Table 35.

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 '001b'. 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.

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 (see Figure 89).

Bit-field T5I in control register T5CON selects the triggering transition (see Table 38).

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 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 CPU clock cycles before it changes.

Figure 89 : Block diagram of auxiliary timer T5 in counter mode

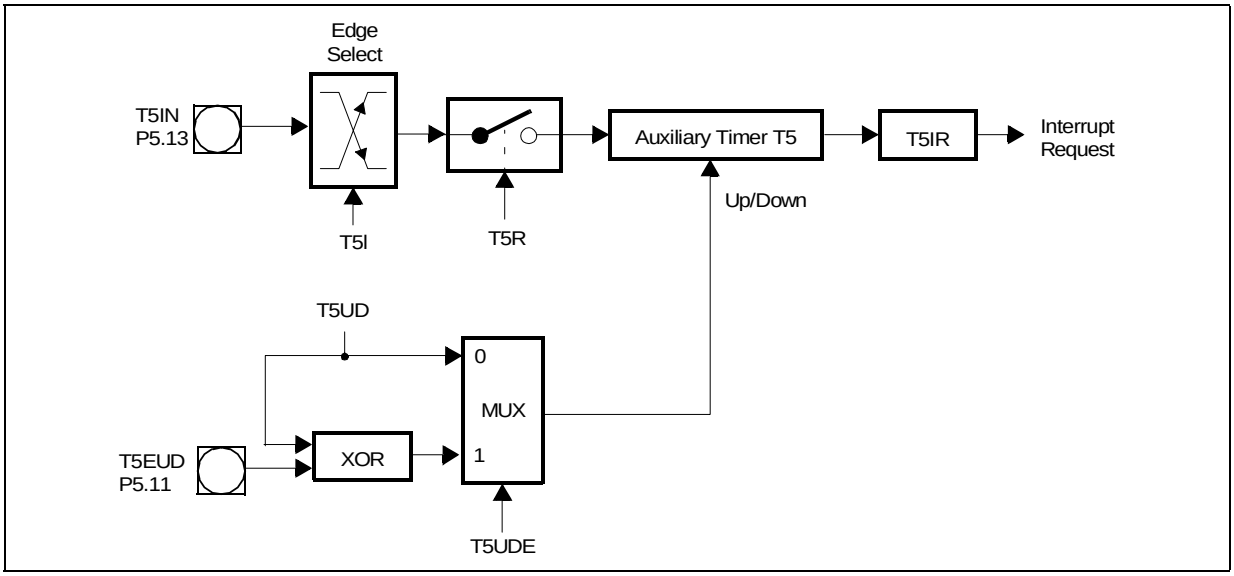


Table 38 : GPT2 auxiliary timer (counter mode) input edge selection

T5I	Triggering Edge for Counter Increment / Decrement
X 0 0	None. Counter T5 is disabled
0 0 1	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

Timer Concatenation

Using the toggle bit T6OTL as a clock source for the auxiliary timer in counter mode concatenates the core timer T6 with the auxiliary timer. Depending on which transition of T6OTL is selected to clock the auxiliary timer, this concatenation forms a 32-bit or a 33-bit timer / counter.

– 32-bit Timer/Counter: If both a positive and a negative transition of T6OTL is used to clock the auxiliary timer, this timer is clocked on every overflow/underflow of the core timer T6. Thus, the two timers form a 32-bit timer.

– 33-bit Timer/Counter: If either a positive or a negative transition of T6OTL is selected to clock the auxiliary timer, this timer is clocked on every second overflow/underflow of the core timer T6. This configuration forms a 33-bit timer (16-bit core timer+T6OTL+16-bit auxiliary timer).

The count directions of the two concatenated timers are not required to be the same. This offers a wide variety of different configurations. T6 can operate in timer, gated timer or counter mode in this case (see Figure 90).

GPT2 Capture / Reload Register CAPREL in Capture Mode

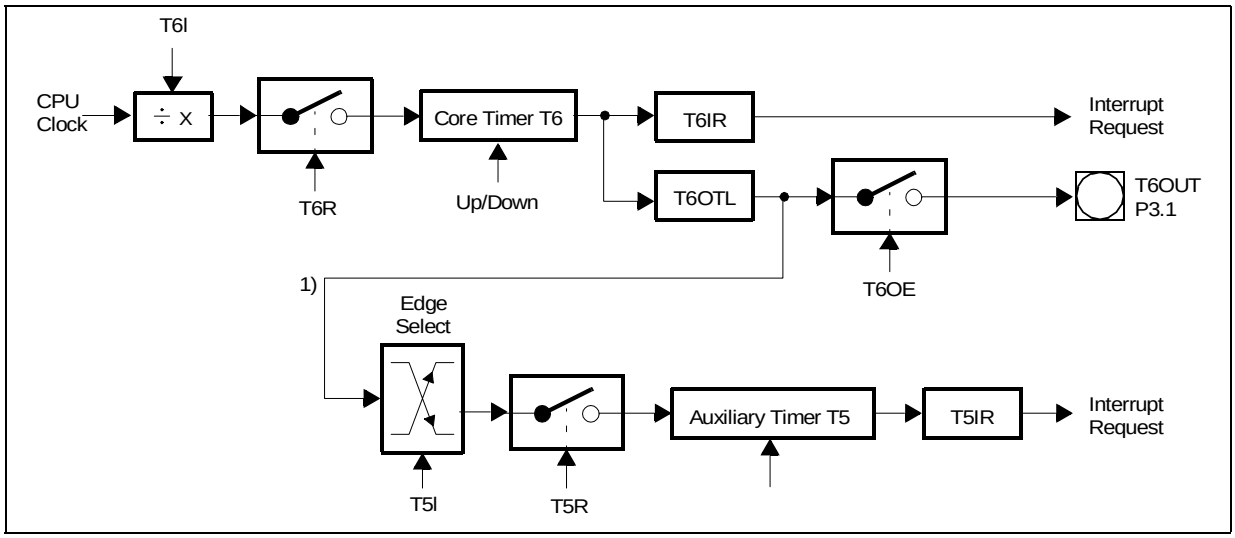
This 16-bit register can be used as a capture register for the auxiliary timer T5. This mode is selected by setting bit T5SC='1' in control register T5CON. Bit CT3 selects the external input pin CAPIN or the input pins of timer T3 as the source for a capture trigger. Either a positive, a negative, or 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 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 is latched into register CAPREL, and interrupt request flag CRIR is set. With the same event, timer T5 can be cleared to 0000h. 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.

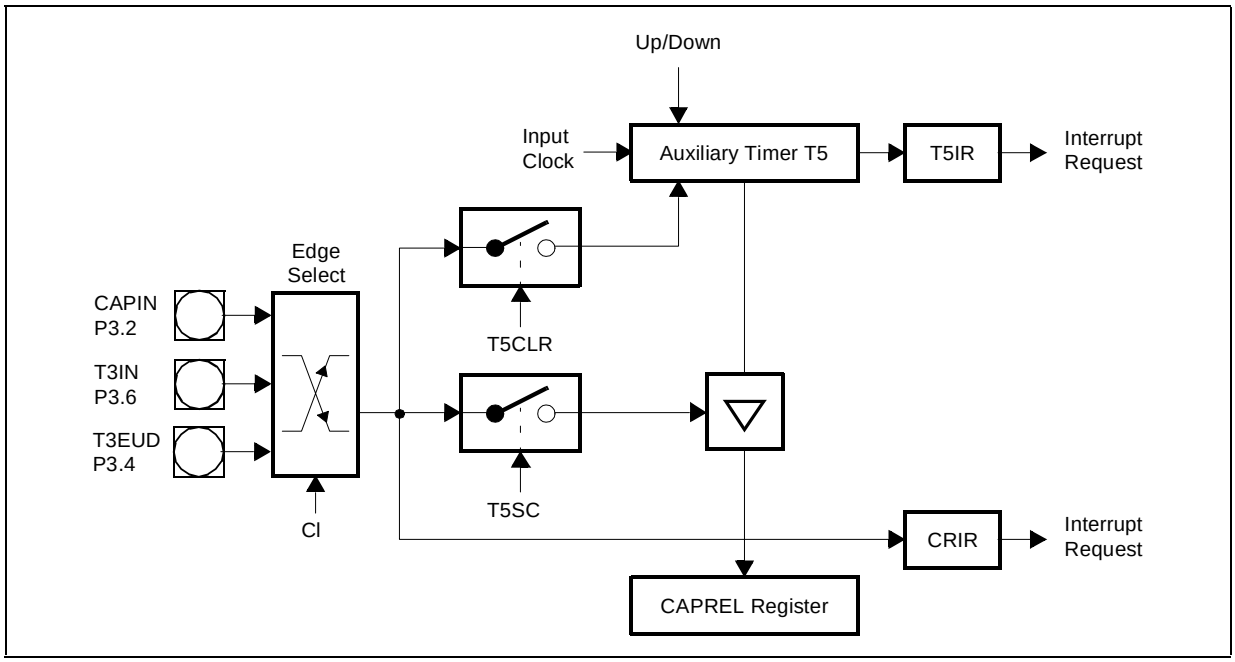
Note Bit T5SC only controls whether a capture is performed or not. If T5SC='0', the input pin CAPIN can still be used to clear timer T5 or as an external interrupt input. This interrupt is controlled by the CAPREL interrupt control register CRIC (see Figure 91).

Figure 90 : Concatenation of core timer T6 and auxiliary timer T5



Note 1. Line only affected by over/underflows of T6, but NOT by software modifications of T6OTL.

Figure 91 : GPT2 register CAPREL in capture mode

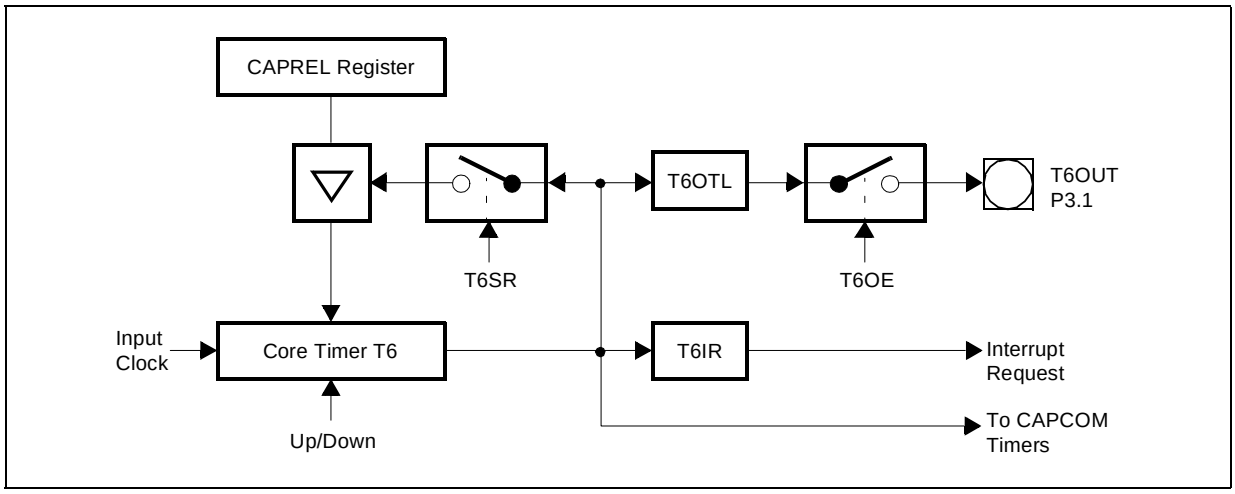


GPT2 Capture / Reload Register CAPREL in Reload Mode

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 or when it underflows from 0000h to FFFFh, 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 92 : 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 bit 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 (see Figure 93).

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.

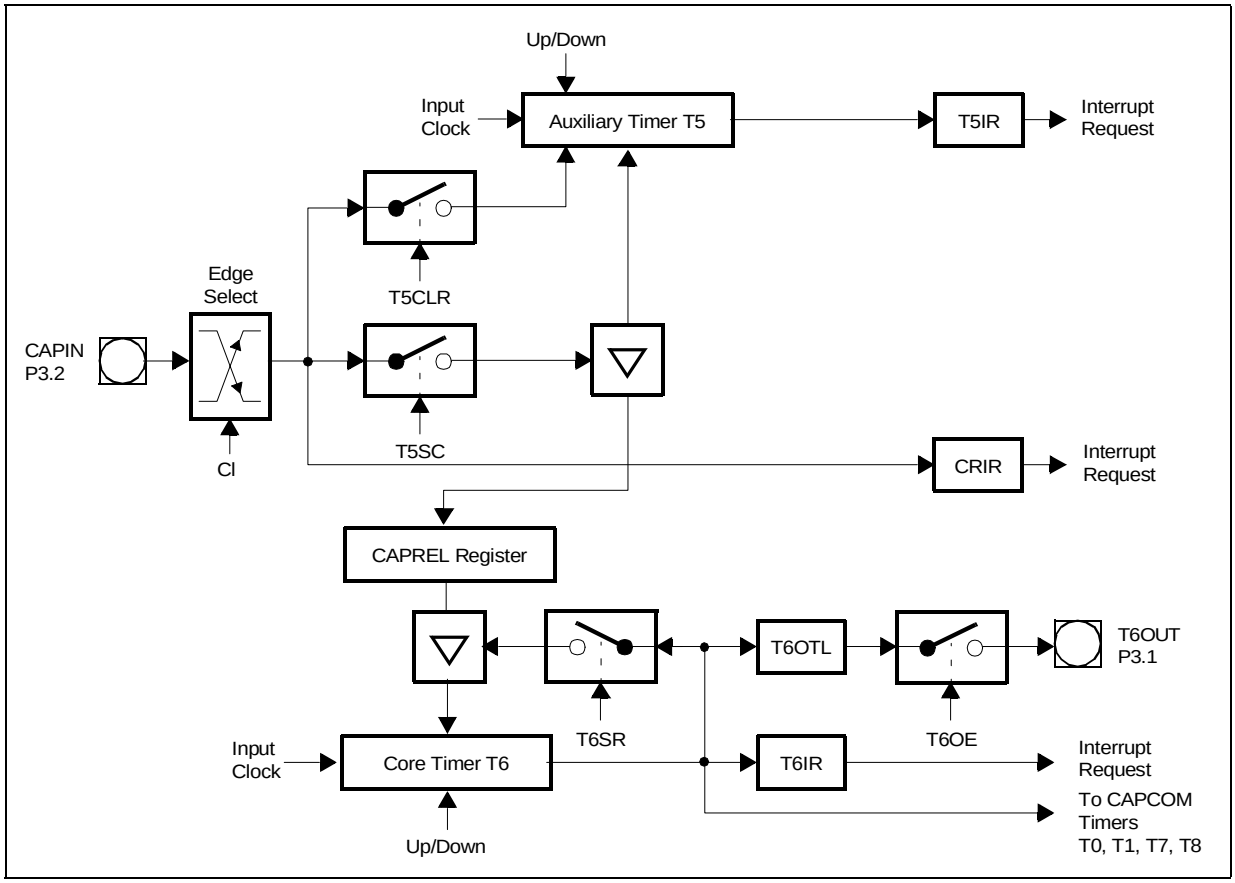
For example, Timer T5 runs in timer mode counting up with a frequency of f_{CPU} / 32. The external events are applied to pin CAPIN. When an external event occurs, the timer T5 contents are latched into register CAPREL, and timer T5 is cleared (T5CLR='1').

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 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.

Figure 93 : GPT2 register CAPREL in capture-and-reload mode



10.2.2 - Interrupt Control for GPT2 Timers and CAPREL

When a timer overflows from FFFFh to 0000h (when counting up), or when it underflows from 0000h to FFFFh (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 Please refer to Interrupt Control Registers for explanation of the control fields.

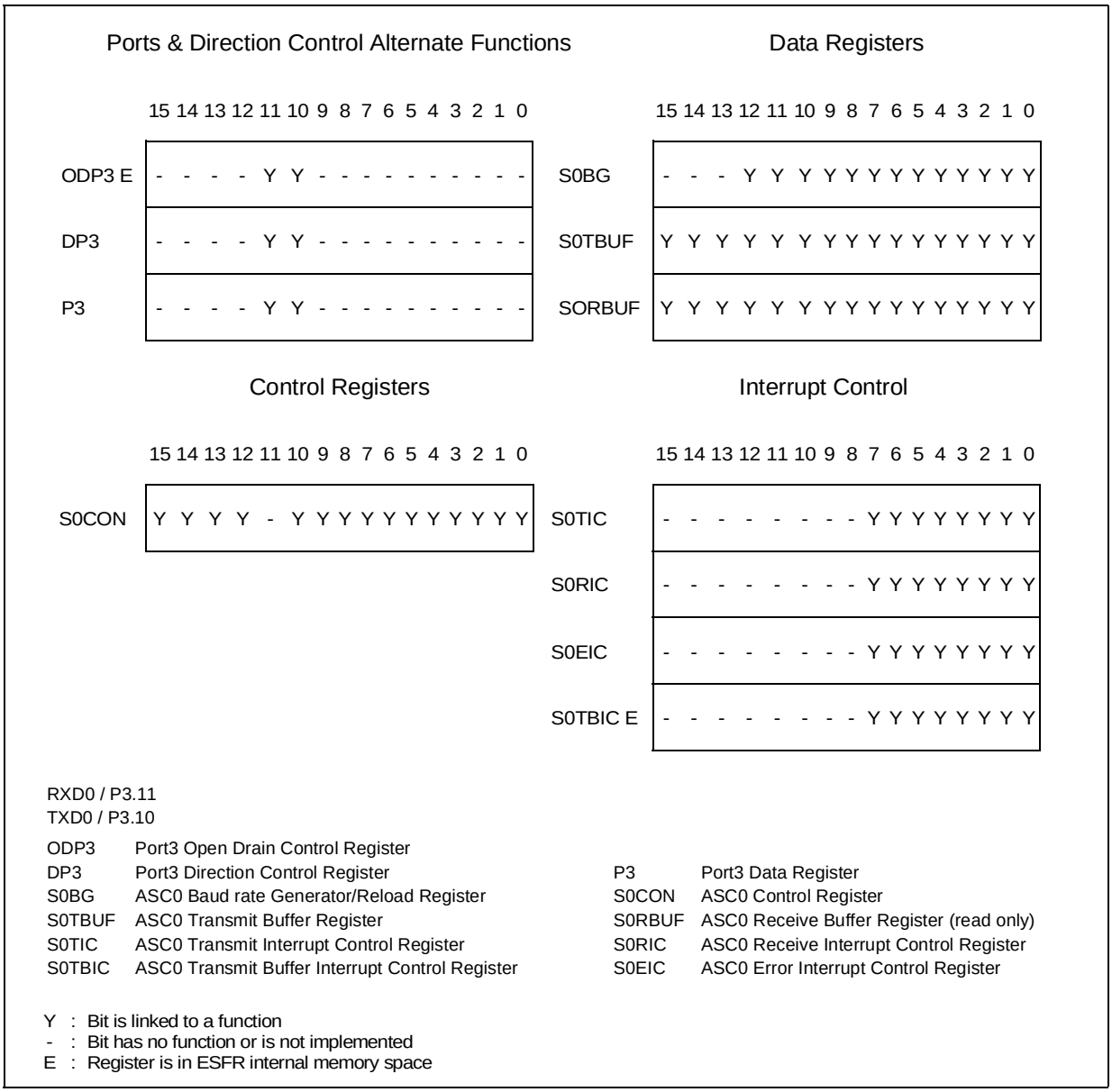
11 - ASYNCHRONOUS/SYNCHRONOUS SERIAL INTERFACE

The Asynchronous/Synchronous Serial Interface ASC0 provides serial communication between the ST10F280 and other microcontrollers, micropro- cessors or external peripherals.

In synchronous mode, data are transmitted or received synchronously to a shift clock which is generated by the ST10F280. In asynchronous mode, 8- or 9-bit data transfer, parity generation, and the number of stop bits can be selected. Parity, framing, and overrun error detection is provided to increase the reliability of data transfers. Transmission and reception of data is double-buffered.

For multiprocessor communication, a mechanism to distinguish address from data byte is included. Testing is supported by a loop-back option. A 13-bit Baud rate generator provides the ASC0 with a separate serial clock signal.

Figure 94 : SFRs and port pins associated with ASC0



The operating mode of the serial channel ASC0 is controlled by its bit-addressable control register S0CON. This register contains control bit for mode and error check selection, and status flags for error identification.

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	S0BRS	S0ODD	-	**S0OE**	**S0FE**	**S0PE**	S0OEN	S0FEN	S0PEN	**S0REN**	S0STP	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 asynchronous operation 0 1 0: Reserved. Do not use this combination 0 1 1: 7-bit data + parity asynchronous operation 1 0 0: 9-bit data asynchronous operation 1 0 1: 8-bit data + wake up bit asynchronous operation 1 1 0: Reserved. Do not use this combination 1 1 1: 8-bit data + parity asynchronous operation
S0STP	**Number of Stop bit Selection asynchronous operation** 0: One stop bit 1: Two stop bit
S0REN	**Receiver Enable bit** 0: Receiver disabled 1: Receiver enabled (Reset by hardware after reception of byte in synchronous mode)
S0PEN	**Parity Check Enable bit asynchronous operation** 0: Ignore parity 1: Check parity
S0FEN	**Framing Check Enable bit asynchronous operation** 0: Ignore framing errors 1: Check framing errors
S0OEN	**Overrun Check Enable bit** 0: Ignore overrun errors 1: Check overrun errors
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	**Baud rate 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	**Baud rate Generator Run bit** 0: Baud rate generator disabled (ASC0 inactive) 1: Baud rate generator enabled

A transmission is started by writing to the Transmit Buffer register S0TBUF (via an instruction or a PEC data transfer).

Only the number of data bit which is determined by the selected operating mode will actually be transmitted. Bit 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 the transmission of characters back-to-back without gaps.

Data reception is enabled by the Receiver Enable bit S0REN. After reception of a character has been completed, the received data and, if provided by the selected operating mode, the received parity bit can be read from the (read-only) Receive Buffer register S0RBUF.

Bit in the upper half of S0RBUF which are not valid in the selected operating mode will be read as zeros.

Data reception is double-buffered, so that reception of a second character may already begin before the previously received character has been read out of the receive buffer register.

In all modes, receive buffer overrun error detection can be selected through bit S0OEN. When enabled, the overrun error status flag S0OE and the error interrupt request flag S0EIR will be set when the receive buffer register has not been read by the time reception of a second character is complete. The previously received character in the receive buffer is overwritten.

The Loop-Back option (selected by bit S0LB) allows the data currently being transmitted to be received simultaneously in the receive buffer.

This may be used to test serial communication routines at an early stage without having to provide an external network. In loop-back mode the alternate input/output functions of the Port3 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.

S0TBUF (FEB0h / 58h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	Baud Rate												
			RW												

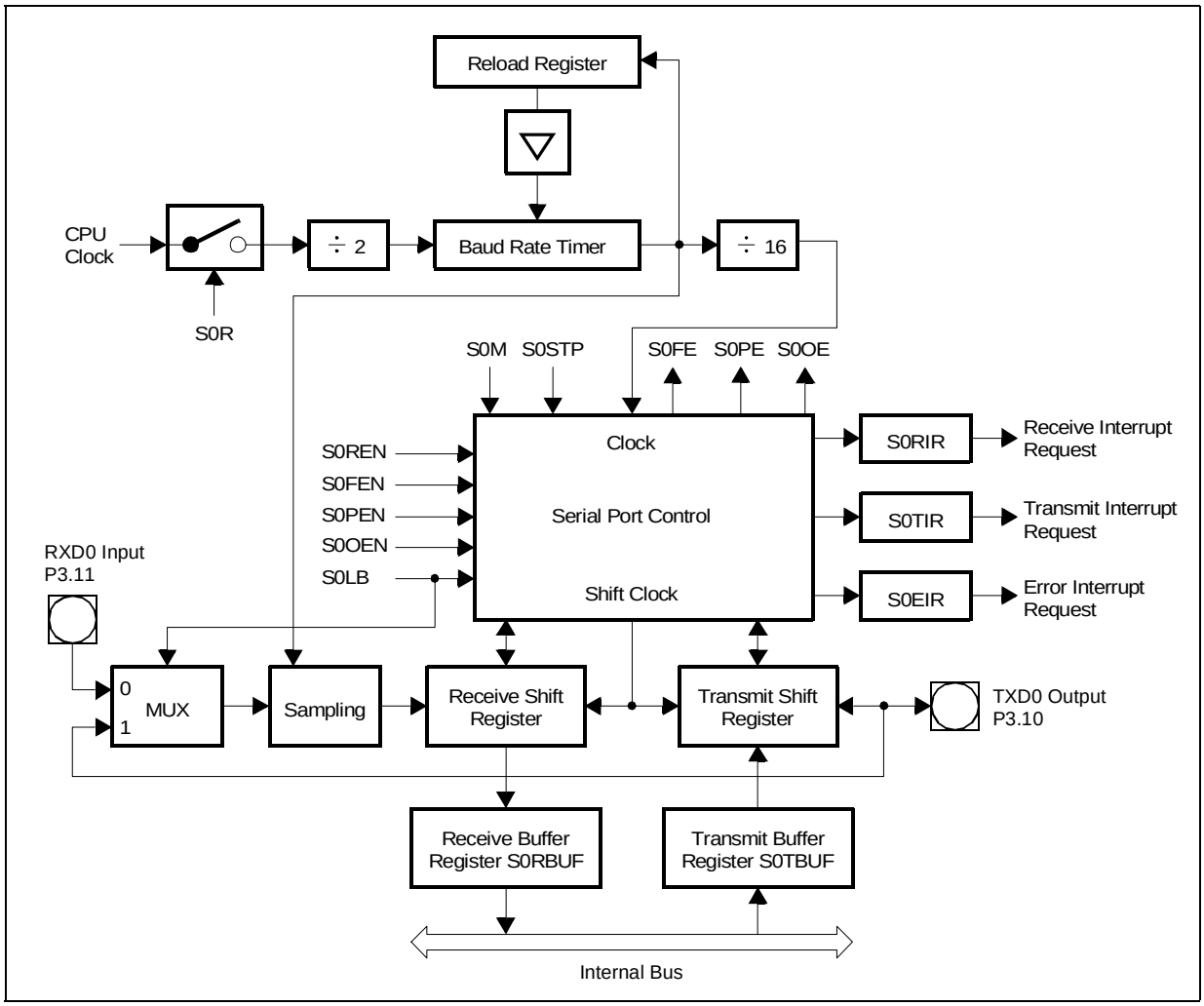
S0RBUF (FEB2h / 59h) SFR Reset Value: 00XXh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	Baud Rate												
			RW												

11.1 - Asynchronous Operation

Asynchronous mode supports full-duplex communication, where both transmitter and receiver use the same data frame format and the same Baud rate. Data is transmitted on pin TXD0/P3.10 and received on pin RXD0/P3.11. These signals are alternate functions of Port 3 pins.

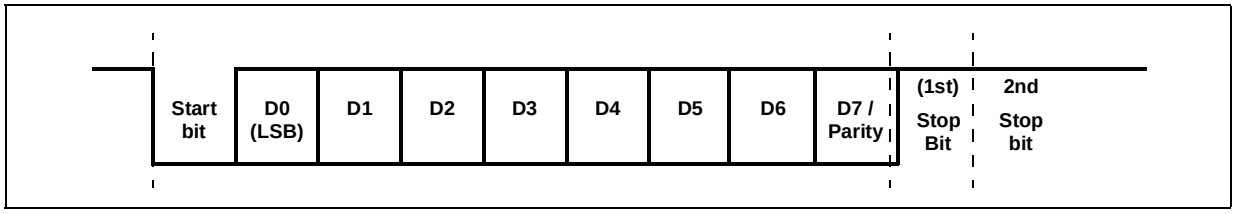
Figure 95 : Asynchronous mode of serial channel ASC0



Asynchronous Data Frames

8-bit data frames either consist of 8 data bit D7...D0 (S0M='001b'), or of 7 data bit 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 bit is '1'. An odd parity bit will be cleared in this case. Parity checking is enabled via bit S0PEN (always OFF in 8-bit data mode). The parity error flag S0PE will be set along with the error interrupt request flag, if a wrong parity bit is received. The parity bit itself will be stored in bit S0RBUF.7.

Figure 96 : Asynchronous 8-bit data frames



9-bit data frames either consist of 9 data bit D8...D0 (S0M='100b'), of 8 data bit D7...D0 plus an automatically generated parity bit (S0M='111b') or of 8 data bit 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 bit 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 8 of S0RBUF.

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 (by clearing bit S0M.0), which enables it to also receive the data byte 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 byte (see Figure 97).

Asynchronous transmission begins at the next overflow of the divide-by-16 counter (see Figure 97), 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 bit, LSB first, including a parity bit, if selected),

– the delimiter (1 or 2 stop bit).

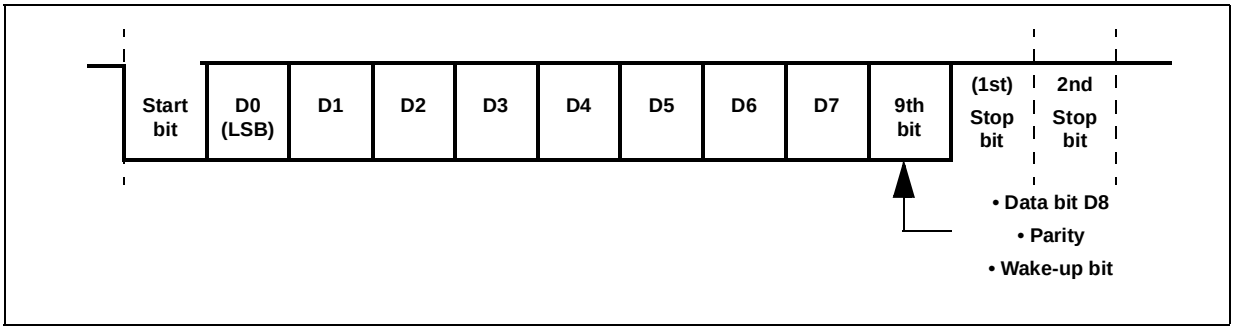
Data transmission is double buffered. When the transmitter is idle, the transmit data loaded into S0TBUF is immediately moved to the transmit shift register thus freeing S0TBUF for the next data to be sent. This is indicated by the transmit buffer interrupt request flag S0TBIR being set. S0TBUF may now be loaded with the next data, while transmission of the previous one is still going on.

The transmit interrupt request flag S0TIR will be set before the last bit of a frame is transmitted, that means 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, P3.10='1' and DP3.10='1'.

Asynchronous reception is initiated by a falling edge (1-to-0 transition) on pin RXD0, provided that bit 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.

Figure 97 : Asynchronous 9-bit data frames



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 bit 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, using direction control register 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 bit that follow this frame will not be recognized.

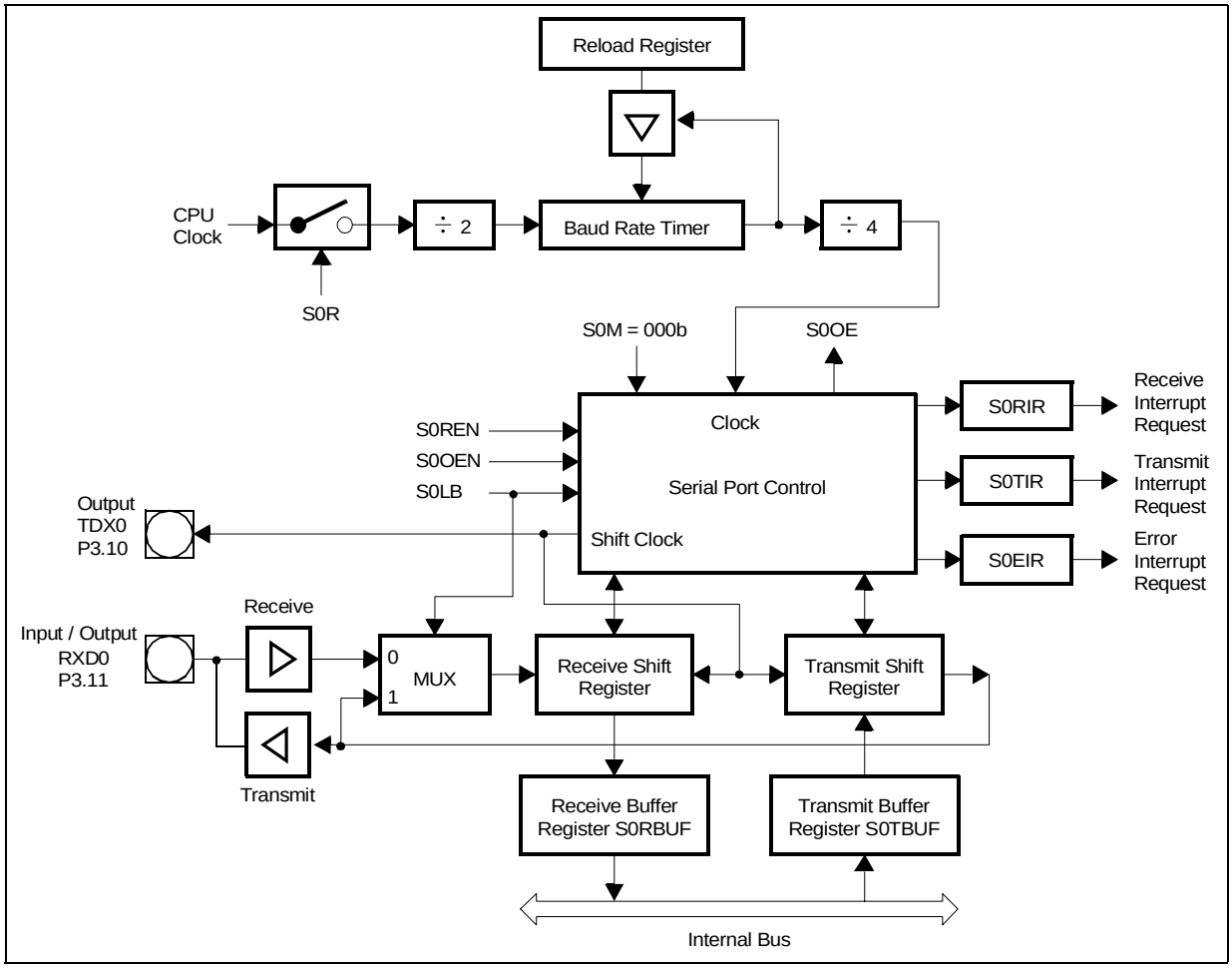
Note In wake-up mode received frames are only transferred to the receive buffer register, if the 9th bit (the wake-up bit) is '1'. If this bit is '0', no receive interrupt request will be activated and no data will be transferred.

11.2 - Synchronous Operation

Synchronous mode supports half-duplex communication, basically for simple I/O 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 Port3 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 98 : 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 synchronous with the shift clock. After the bit time for the 8th data bit, both pins TXD0 and RXD0 will go high, the transmit interrupt request flag S0TIR is set, and serial data transmission stops.

Pin TXD0/P3.10 must be configured for alternate data output, 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, 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, if the overrun check has been enabled by S0OEN.

11.3 - Hardware Error Detection

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 bit is not high, the framing error flag S0FE is set, indicating that the error interrupt request is due to a framing error (Asynchronous mode only).

– If the parity error detection enable bit S0PEN is set in parity bit receive modes, and the parity check on the received data bit proves false, the parity error flag S0PE is set, indicating that the error interrupt request is due to a parity error (Asynchronous mode only).

– If the overrun error detection enable bit S0OEN is set and the last character received was not read out of the receive buffer by software or PEC transfer at the time the reception of a new frame is complete, the overrun error flag S0OE is set indicating that the error interrupt request is due to an overrun error (Asynchronous and synchronous mode).

11.4 - ASC0 Baud Rate Generation

The serial channel ASC0 has its own dedicated 13-bit Baud rate generator with 13-bit reload capability, allowing Baud rate generation independent of the GPT timers. The Baud rate generator is clocked by f_{CPU} / 2. The timer is counting downwards and can be started or stopped through the Baud rate Generator Run bit S0R in register S0CON. Each underflow of the timer provides one clock pulse to the serial channel. The timer is reloaded with the value stored in its 13-bit reload register each time it underflows. The resulting clock is again divided according to the operating mode and controlled by the Baud rate 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 (bit 15...13 return zero), while writing to S0BG always updates the reload register (bit 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'.

S0BG (FEB4h / 5Ah) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	Baud Rate												
			RW												

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 Baud rate can be determined by the following formulas:

$$B_{Async} = \frac{f_{CPU}}{16 \times [2 + (S0BRS)] \times [(S0BRL) + 1]}$$

$$S0BRL = \left(\frac{f_{CPU}}{16 \times [2 + (S0BRS)] \times B_{Async}} \right) - 1$$

(S0BRL) represents the content of the reload register, taken as unsigned 13-bit integer,
(S0BRS) represents the value of bit S0BRS ('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.

Synchronous Mode Baud Rates

For synchronous operation, the Baud rate generator provides a clock with 4 times the rate of the established Baud rate. The Baud rate for synchronous operation of serial channel ASC0 can be determined by the following formula:

$$B_{Sync} = \frac{f_{CPU}}{4 \times [2 + (S0BRS)] \times [(S0BRL) + 1]}$$

$$S0BRL = \left(\frac{f_{CPU}}{4 \times [2 + (S0BRS)] \times B_{Sync}} \right) - 1$$

(S0BRL) represents the content of the reload register, taken as unsigned 13-bit integers,
(S0BRS) represents the value of bit S0BRS ('0' or '1'), taken as integer.

Using the above equation, the maximum Baud rate can be calculated for any given clock speed.

11.5 - ASC0 Interrupt Control

Four bit addressable interrupt control registers are provided for serial channel ASC0. Register S0TIC controls the transmit interrupt, S0TBIC controls the transmit buffer interrupt, S0RIC controls the receive interrupt and S0EIC controls the error interrupt of serial channel ASC0. Each interrupt source also has its own dedicated interrupt vector. S0TINT is the transmit interrupt vector, S0TBINT is the transmit interrupt vector, S0RINT is the receive interrupt vector, and S0EINT is the error interrupt vector.

The cause of an error interrupt request (framing, parity, overrun error) can be identified by the error status flags in control register S0CON.

Note In contrary to the error interrupt request flag S0EIR, the error status flags S0FE/S0PE/S0OE are not reset automatically upon entry into the error interrupt service routine, but must be cleared by software.

S0TIC (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 / B8) 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
-	-	-	-	-	-	-	-	**S0TBIR**	S0TBIE	ILVL				GLVL	
								RW	RW	RW				RW	

Note Please refer to Section 6.1.3 - Interrupt Control Registers for an explanation of the control fields.

Using the ASC0 Interrupts

For normal operation (besides the error interrupt) the ASC0 provides three interrupt requests to control data exchange via this serial channel:

– S0TBIRis activated when data is moved from S0TBUF to the transmit shift register.

– S0TIRis activated before the last bit of an asynchronous frame is transmitted, or after the last bit of a synchronous frame has been transmitted.

– S0RIRis activated when the received frame is moved to S0RBUF.

While the task of the receive interrupt handler is quite clear, the transmitter is serviced by two interrupt handlers. This provides advantages for the servicing software.

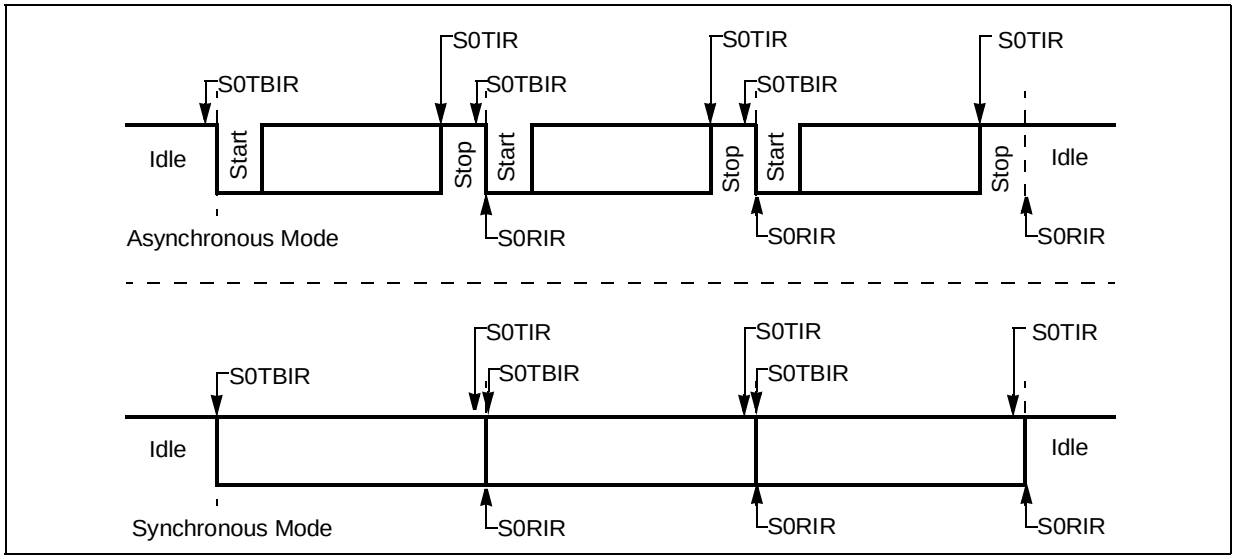
For single transfers is sufficient to use the transmitter interrupt (S0TIR), which indicates that the previously loaded data has been transmitted, except for the last bit of an asynchronous frame.

For multiple back-to-back transfers it is necessary to load the following piece of data at last until the time the last bit of the previous frame has been transmitted. In asynchronous mode this leaves just one bit-time for the handler to respond to the transmitter interrupt request, in synchronous mode it is impossible at all.

Using the transmit buffer interrupt (S0TBIR) to reload transmit data gives the time to transmit a complete frame for the service routine, as S0TBUF may be reloaded while the previous data is still being transmitted.

As shown in the Figure 99, S0TBIR is an early trigger for the reload routine, while S0TIR indicates the completed transmission. Software using handshake therefore should rely on S0TIR at the end of a data block to make sure that all data has really been transmitted.

Figure 99 : ASC0 interrupt generation



12 - HIGH-SPEED SYNCHRONOUS SERIAL INTERFACE

The High-Speed Synchronous Serial Interface SSC provides flexible high-speed serial communication between the ST10F280 and other microcontrollers, microprocessors or external peripherals.

The SSC supports full-duplex and half-duplex synchronous communication. The serial clock signal can be generated by the SSC itself (master mode) or be received from an external master (slave mode). Data width, shift direction, clock polarity and phase are programmable. This allows communication with SPI-compatible devices. Transmission and reception of data is double-buffered. A 16-bit Baud rate generator provides the SSC with a separate serial clock signal.

The high-speed synchronous serial interface can be configured in three ways, it can be used with other synchronous serial interfaces (the ASC0 in synchronous mode), or configured in like master / slave or multimaster interconnections or operate like the popular SPI interface. It can communicate with shift registers (I/O expansion), peripherals (EEPROMs etc.) or other controllers (networking). The SSC supports half-duplex and full-duplex communication. Data is transmitted or received on pins MTSR/P3.9 (Master Transmit / Slave Receive) and MRST/P3.8 (Master Receive / Slave Transmit). The clock signal is output or input on pin SCLK/P3.13. These pins are alternate functions of Port3 pins.

Figure 100 : SFRs and port pins associated with the SSC

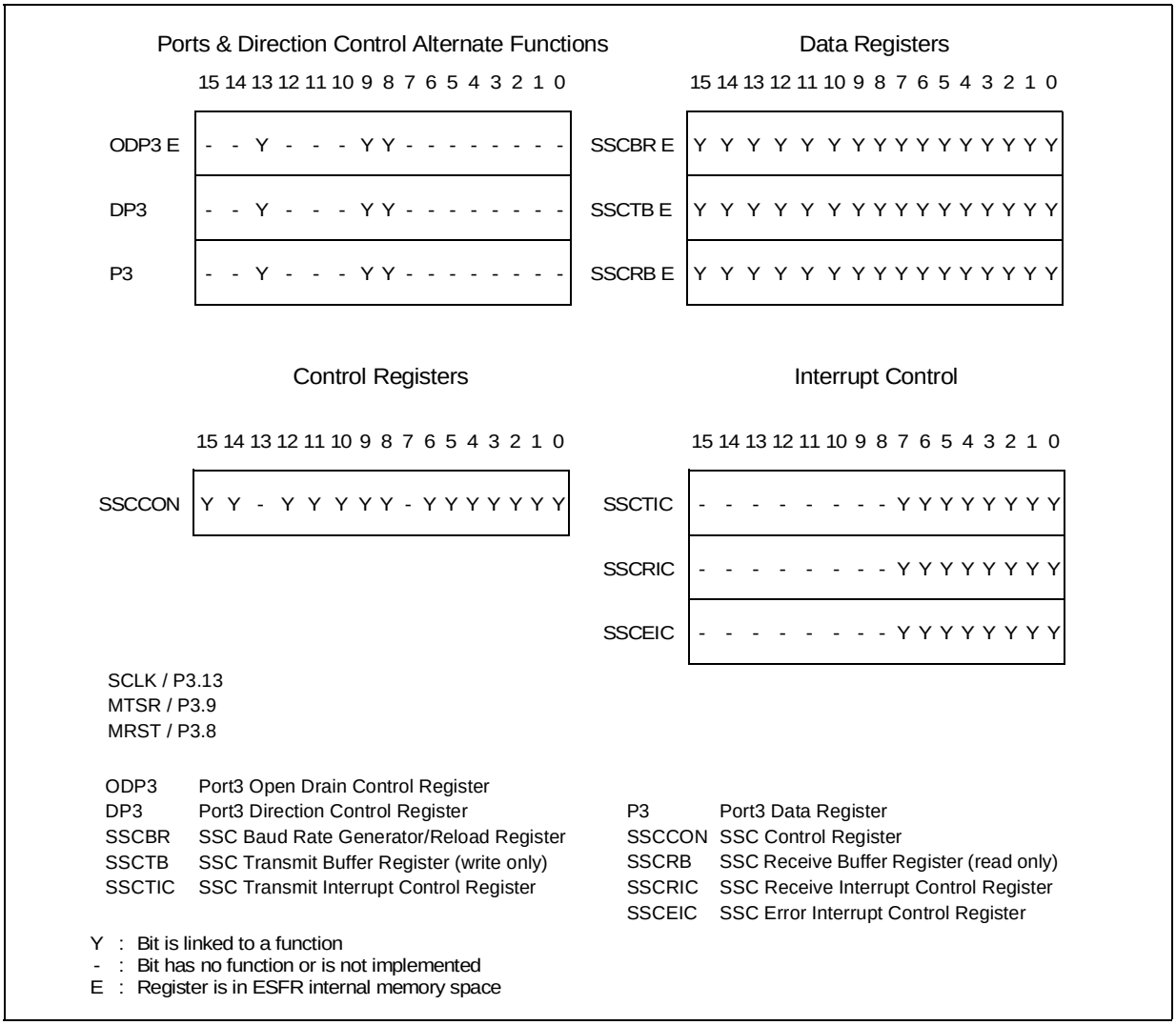
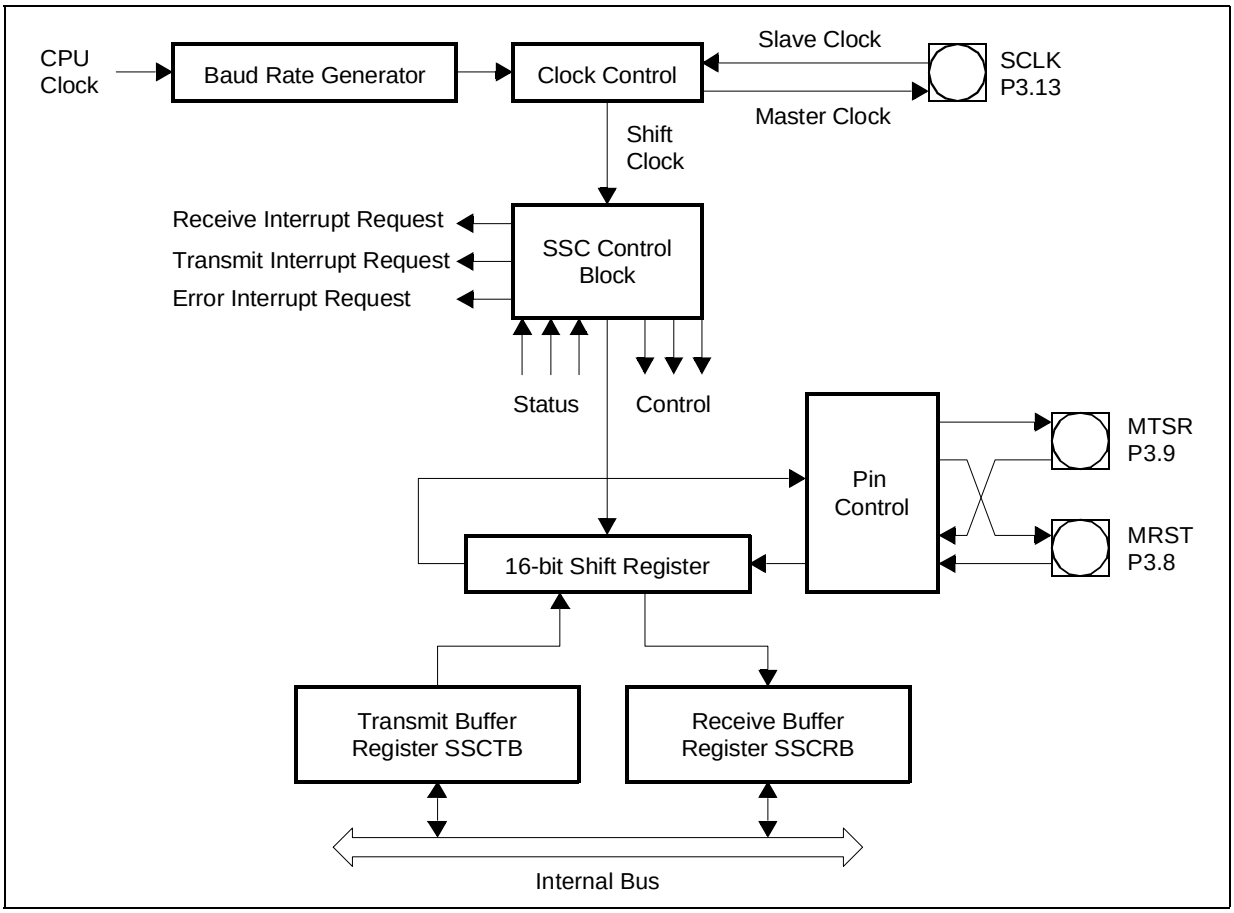


Figure 101 : Synchronous serial channel SSC block diagram



The operating mode of the serial channel SSC is controlled by its bit-addressable control register SSCCON. This register serves for two purposes:

– During programming (SSC disabled by SSCEN='0') it provides access to a set of control bit.

– During operation (SSC enabled by SSCEN='1') it provides access to a set of status flags. Register SSCCON is shown below in each of the two modes.

SSCRB (F0B2h / 59h) ESFR Reset Value: XXXXh

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

RW

SSCTB (F0B0h / 58h) ESFR Reset Value: 0000h

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

RW

SSCCON (FFB2h / D9h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SSC EN=0	SSC MS	-	SSC AREN	SSC BEN	SSC PEN	SSC REN	SSC TEN	-	SSC PO	SSC PH	SSC HB	SSCBM			
RW	RW		RW	RW	RW	RW	RW		RW	RW	RW	RW			

Bit	Function (Programming Mode, SSCEN = '0')
SSCBM	**SSC Data Width Selection** 0: Reserved. Do not use this combination. 1...15: Transfer Data Width is 2...16-bit [(SSCBM)+1]
SSCHB	**SSC Heading Control bit** 0: Transmit/Receive LSB First 1: Transmit/Receive MSB First
SSCPH	**SSC Clock Phase Control bit** 0: Shift transmit data on the leading clock edge, latch on trailing edge 1: Latch receive data on leading clock edge, shift on trailing edge
SSCPO	**SSC Clock Polarity Control bit** 0: Idle clock line is low, leading clock edge is low-to-high transition 1: Idle clock line is high, leading clock edge is high-to-low transition
SSCTEN	**SSC Transmit Error Enable bit** 0: Ignore transmit errors 1: Check transmit errors
SSCREN	**SSC Receive Error Enable bit** 0: Ignore receive errors 1: Check receive errors
SSCPEN	**SSC Phase Error Enable bit** 0: Ignore phase errors 1: Check phase errors
SSCBEN	**SSC Baudrate Error Enable bit** 0: Ignore baudrate errors 1: Check baudrate errors
SSCAREN	**SSC Automatic Reset Enable bit** 0: No additional action upon a baudrate error 1: The SSC is automatically reset upon a baudrate error
SSCMS	**SSC Master Select bit** 0: Slave Mode. Operate on shift clock received via SCLK. 1: Master Mode. Generate shift clock and output it via SCLK.
SSCEN	**SSC Enable bit = '0'** Transmission and reception disabled. Access to control bits.

SSCCON (FFB2h / D9h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SSC EN=1	SSC MS	-	SSC BSY	SSC BE	SSC PE	SSC RE	SSC TE	-	-	-	-	SSCBC			
RW	RW		RW	RW	RW	RW	RW					R			

Bit	Function (Operating Mode, SSCEN = '1')
SSCBC	**SSC bit Count Field** Shift counter is updated with every shifted bit. **Do not write to**
SSCTE	**SSC Transmit Error Flag** 1: Transfer starts with the slave's transmit buffer not being updated
SSCRE	**SSC Receive Error Flag** 1: Reception completed before the receive buffer was read
SSCPE	**SSC Phase Error Flag** 1: Received data changes around sampling clock edge
SSCBE	**SSC Baud rate Error Flag** 1: More than factor 2 or 0.5 between Slave's actual and expected Baud rate
SSCBSY	**SSC Busy Flag:** Set while a transfer is in progress. **Do not write to**
SSCMS	**SSC Master Select bit** 0: Slave Mode. Operate on shift clock received via SCLK. 1: Master Mode. Generate shift clock and output it via SCLK.
SSCEN	**SSC Enable bit = '1'** Transmission and reception enabled. Access to status flags and M/S control.

Note The target of an access to SSCCON (control bit or flags) is determined by the state of SSCEN prior to the access. Writing C057h to SSCCON in programming mode (SSCEN='0') will initialize the SSC (SSCEN was '0') and then turn it on (SSCEN='1').
When writing to SSCCON, make sure that reserved locations receive zeros.

The shift register of the SSC is connected to both the transmit pin and the receive pin via the pin control logic (see Figure 101). Transmission and reception of serial data is synchronized and takes place at the same time, so the same number of transmitted bit is also received. Transmit data is written into the Transmit Buffer SSCTB. It is moved to the shift register as soon as this is empty. An SSC-master (SSCMS='1') immediately begins transmitting, while an SSC-slave (SSCMS='0') will wait for an active shift clock. When the transfer starts, the busy flag SSCBSY is set and a transmit interrupt request (SSCTIR) will be generated to indicate that SSCTB may be reloaded again. When the programmed number of bit (2...16) has been transferred, the contents of the shift register are moved to the Receive Buffer SSCRB and a receive interrupt request (SSCRIR) will be generated. If no further transfer is to take place (SSCTB is empty), SSCBSY will be cleared at the same time. Software should not modify SSCBSY, as this flag is hardware controlled. Only one SSC can be master at a given time.

The transfer of serial data bit can be programmed in the following ways:

– The data width can be chosen from 2 bits to 16 bits.

– Transfer may start with the LSB or the MSB.

– The shift clock may be idle low or idle high.

– Data bit may be shifted with the leading or trailing edge of the clock signal.

– The Baud rate may be set for a range of values (refer to Section 12.3 - Baud Rate Generation for the formula to calculate values or to the device datasheet for specific values).

– The shift clock can be generated (master) or received (slave).

This allows the adaptation of the SSC to a wide range of applications, where serial data transfer is required.

The data width selection supports the transfer of frames of any length, from 2 bit "characters" up to 16 bit "characters". Starting with the LSB (SSCHB='0') allows communication with ASC0 devices in synchronous mode like serial interfaces. Starting with the MSB (SSCHB='1') allows operation compatible with the SPI interface.

Regardless which data width is selected and whether the MSB or the LSB is transmitted first, the transfer data is always right aligned in registers SSCTB and SSCRB, with the LSB of the transfer data in bit 0 of these registers. The data bits are rearranged for transfer by the internal shift register logic. The unselected bits of SSCTB are ignored, the unselected bits of SSCRB will be not valid and should be ignored by the receiver service routine.

The clock control allows the adaptation of transmit and receive behavior of the SSC to a variety of serial interfaces. A specific clock edge (rising or falling) is used to shift out transmit data, while the other clock edge is used to latch in receive data. Bit SSCPH selects the leading edge or the trailing edge for each function. Bit SSCPO selects the level of the clock line in the idle state. So for an idle-high clock the leading edge is a falling one, a 1-to-0 transition. The Figure 102 is a summary.

12.1 - Full-Duplex Operation

The different devices are connected through three lines. The definition of these lines is always determined by the master: The line connected to the master's data output pin MTSR is the transmit line, the receive line is connected to its data input line MRST, and the clock line is connected to pin SCLK. Only the device selected for master operation generates and outputs the serial clock on pin SCLK. All slaves receive this clock, so their pin SCLK must be switched to input mode (DP3.13='0'). The output of the master's shift register is connected to the external transmit line, which in turn is connected to the slaves' shift register input.

The output of the slaves' shift register is connected to the external receive line in order to enable the master to receive the data shifted out of the slave. The external connections are hard-wired, the function and direction of these pins is determined by the master or slave operation of the individual device.

Note The shift direction shown in the Figure 103 applies for MSB-first operation as well as for LSB-first operation.

When initializing the devices in this configuration, select one device for master operation (SSCMS='1'), all others must be programmed for slave operation (SSCMS='0'). Initialization includes the operating mode of the device's SSC and also the function of the respective port lines (see Section 12.2.1 - Port Control).

Figure 102 : Serial clock phase and polarity options

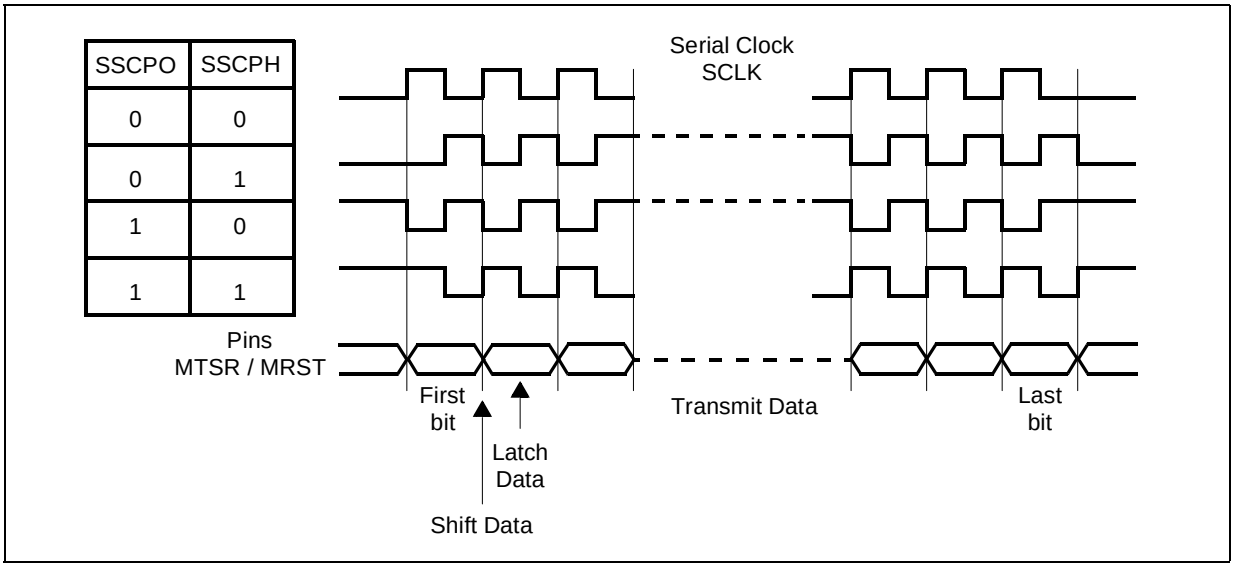
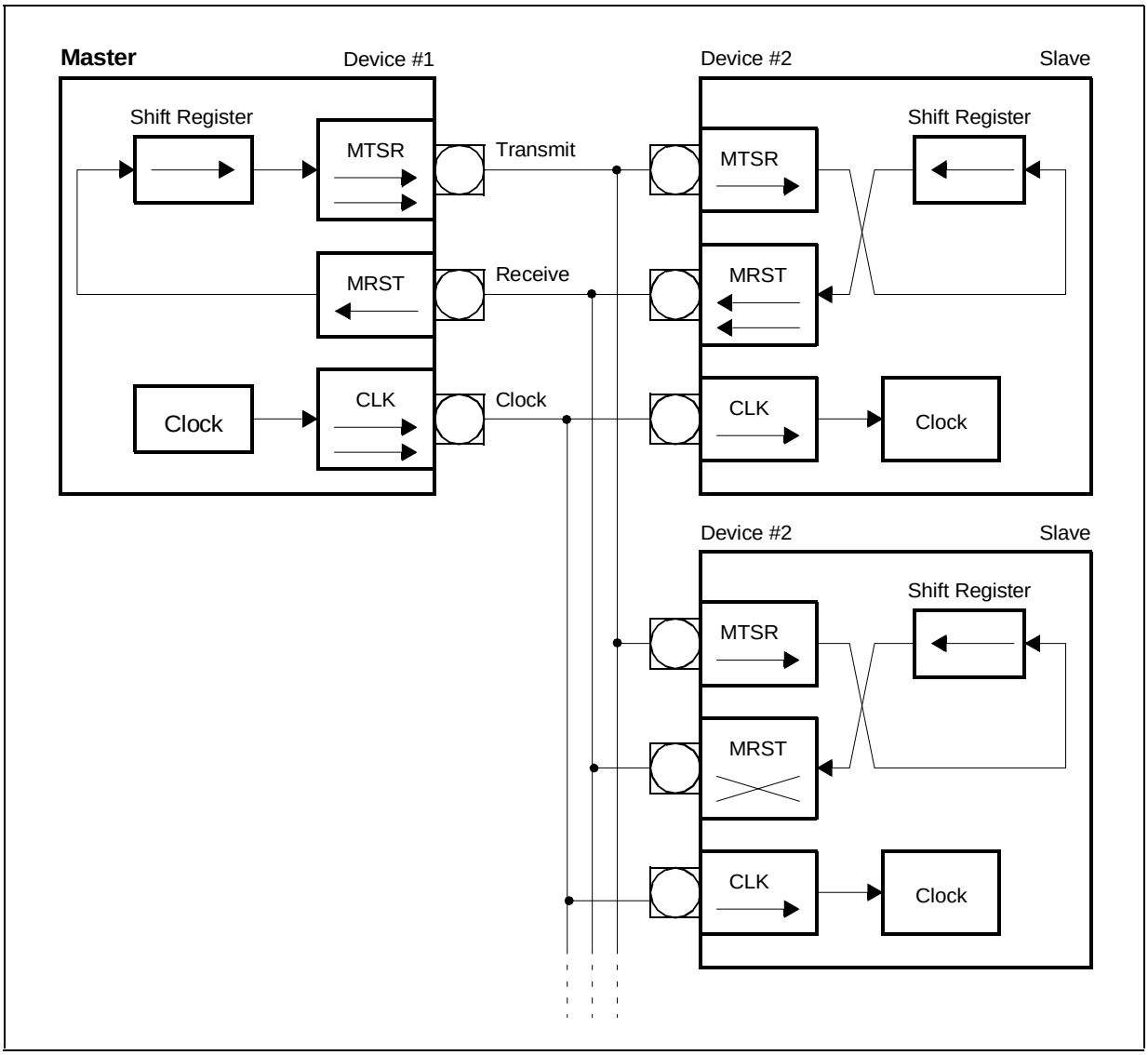


Figure 103 : SSC full duplex configuration



The data output pins MRST of all slave devices are connected together onto the one receive line in this configuration. During a transfer each slave shifts out data from its shift register. There are two ways to avoid collisions on the receive line due to different slave data:

Only one slave drives the line, it enables the driver of its MRST pin. All the other slaves have to program there MRST pins to input. So only one slave can put its data onto the master's receive line. Only receiving of data from the master is possible. The master selects the slave device from which it expects data either by separate select lines, or by sending a special command to this slave. The selected slave then switches its MRST line to output, until it gets a de-selection signal or command.

The slaves use open drain output on MRST. This forms a AND-wired connection. The receive line needs an external pull-up in this case. Corruption of the data on the receive line sent by the selected slave is avoided, when all slaves which are not selected for transmission to the master only send ones ('1'). Since this high level is not actively driven onto the line, but only held through the pull-up device, the selected slave can pull this line actively to a low level when transmitting a zero bit. The master selects the slave device from which it expects data either by separate select lines, or by sending a special command to this slave.

After performing all necessary initialization of the SSC, the serial interfaces can be enabled. For a master device, the alternate clock line will now go to its programmed polarity. The alternate data line will go to either '0' or '1', until the first transfer will start. After a transfer the alternate data line will always remain at the logic level of the last transmitted data bit.

When the serial interfaces are enabled, the master device can initiate the first data transfer by writing the transmit data into register SSCTB. This value is copied into the shift register (which is assumed to be empty at this time), and the selected first bit of the transmit data will be placed onto the MTSR line on the next clock from the Baud rate generator (transmission only starts, if SSCEN='1'). Depending on the selected clock phase, also a clock pulse will be generated on the SCLK line.

With the opposite clock edge the master at the same time latches and shifts in the data detected at its input line MRST. This "exchanges" the transmit data with the receive data. Since the clock line is connected to all slaves, their shift registers will be shifted synchronously with the master's shift register, shifting out the data contained in the registers, and shifting in the data detected at the input line. After the pre-programmed number of clock pulses (via the data width selection) the data transmitted by the master is contained in all slaves' shift registers, while the master's shift register holds the data of the selected slave. In the master and all slaves the content of the shift register is copied into the receive buffer SSCRB and the receive interrupt flag SSCRIR is set.

A slave device will immediately output the selected first bit (MSB or LSB of the transfer data) at pin MRST, when the content of the transmit buffer is copied into the slave's shift register. It will not wait for the next clock from the Baud rate generator, as the master does. The reason for this is that, depending on the selected clock phase, the first clock edge generated by the master may be already used to clock in the first data bit. So the slave's first data bit must already be valid at this time.

Note A transmission **and** a reception takes place at the same time, regardless whether valid data has been transmitted or received. This is different from asynchronous reception on ASC0.

The initialization of the SCLK pin on the master requires some attention in order to avoid undesired clock transitions, which may disturb the other receivers. The state of the internal alternate output lines is '1' as long as the SSC is disabled. This alternate output signal is ANDed with the respective port line output latch. Enabling the SSC with an idle-low clock (SSCPO='0') will drive the alternate data output and (via the AND) the port pin SCLK immediately low. To avoid this, use the following sequence:

– Select the clock idle level (SSCPO='x')

– Load the port output latch with the desired clock idle level (P3.13='x')

– Switch the pin to output (DP3.13='1')

– Enable the SSC (SSCEN='1')

– If SSCPO='0': enable alternate data output (P3.13='1')

The same mechanism as for selecting a slave for transmission (separate select lines or special commands) may also be used to move the role of the master to another device in the network. In this case the previous master and the future master (previous slave) will have to toggle their operating mode (SSCMS) and the direction of their port pins (see description above).

12.2 - Half Duplex Operation

In a half duplex configuration only one data line is necessary for both receiving **and** transmitting of data. The data exchange line is connected to both pins MTSR and MRST of each device, the clock line is connected to the SCLK pin.

The master device controls the data transfer by generating the shift clock, while the slave devices receive it. Due to the fact that all transmit and receive pins are connected to the one data exchange line, serial data may be moved between arbitrary stations.

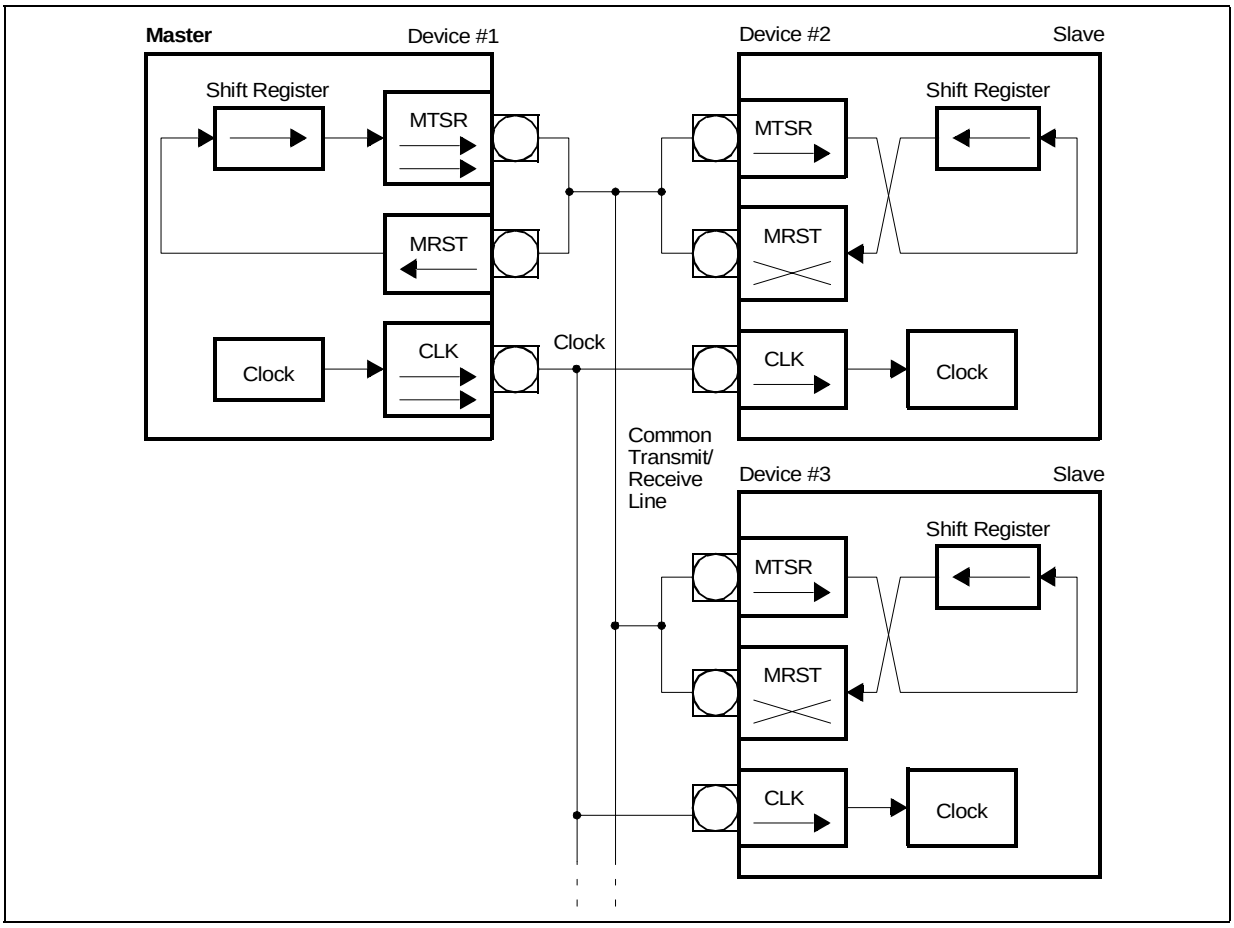
Similar to full duplex mode there are **two ways to avoid collisions** on the data exchange line:

– Only the transmitting device may enable its transmit pin driver

– The non-transmitting devices use open drain output and only send ones.

Since the data inputs and outputs are connected together, a transmitting device will clock in its own data at the input pin (MRST for a master device, MTSR for a slave). By these means any corruptions on the common data exchange line are detected, where the received data is not equal to the transmitted data.

Figure 104 : SSC half duplex configuration



Continuous Transfers

When the transmit interrupt request flag is set, it indicates that the transmit buffer SSCTB is empty and ready to be loaded with the next transmit data. If SSCTB has been reloaded by the time the current transmission is finished, the data is immediately transferred to the shift register and the next transmission will start without any additional delay. On the data line there is no gap between the two successive frames, so two bytes transfers would look the same as one word transfer. This feature can be used to interface with devices which can operate with or require more than 16 data bits per transfer. It is just a matter of software, how long a total data frame length can be. This option can also be used to interface to byte wide and word wide devices on the same serial bus.

Note Of course, this can only happen in multiples of the selected basic data width, since it would require disabling/enabling of the SSC to reprogram the basic data width on-the-fly.

12.2.1 - Port Control

The SSC uses three pins of Port3 to communicate with the external world. Pin P3.13/SCLK serves as the clock line, while pins P3.8/MRST (Master Receive / Slave Transmit) and P3.9/MTSR (Master Transmit / Slave Receive) serve as the serial data input/output lines. The operation of these pins depends on the selected operating mode (master or slave). In order to enable the alternate output functions of these pins instead of the general purpose I/O operation, the respective port latches have to be set to '1', since the port latch outputs and the alternate output lines are ANDed. When an alternate data output line is not used (function disabled), it is held at a high level, allowing I/O operations via the port latch. The direction of the port lines depends on the operating mode. The SSC will automatically use the correct alternate input or output line of the ports when switching modes. The direction of the pins, however, must be programmed by the user, as shown in the tables.

Using the open drain output feature helps to avoid bus contention problems and reduces the need for hardwired hand-shaking or slave select lines. In this case it is not always necessary to switch the direction of a port pin. The table below summarizes the required values for the different modes and pins.

Pin	Master Mode			Slave Mode		
	Function	Port Latch	Direction	Function	Port Latch	Direction
P3.13 / SCLK	Serial Clock Output	P3.13='1'	DP3.13='1'	Serial Clock Input	P3.13='x'	DP3.13='0'
P3.9 / MTSR	Serial Data Output	P3.9='1'	DP3.9='1'	Serial Data Input	P3.9='x'	DP3.9='0'
P3.8 / MRST	Serial Data Input	P3.8='x'	DP3.8='0'	Serial Data Output	P3.8='1'	DP3.8='1'

Note In the table above, an 'x' means that the actual value is irrelevant in the respective mode, however, it is recommended to set these bits to '1', so they are already in the correct state when switching between master and slave mode.

12.3 - Baud Rate Generation

The serial channel SSC has its own dedicated 16-bit Baud rate generator with 16-bit reload capability, allowing Baud rate generation independent from the timers.

The Baud rate generator is clocked by $f_{CPU}/2$. The timer is counting downwards and can be started or stopped through the global enable bit SSCEN in register SSCCON. Register SSCBR is the dual-function Baud Rate Generator/Reload register. Reading SSCBR, while the SSC is enabled, returns the content of the timer. Reading SSCBR, while the SSC is disabled, returns the programmed reload value. In this mode the desired reload value can be written to SSCBR.

Note Never write to SSCBR, while the SSC is enabled.

The formulas below calculate the resulting Baud rate for a given reload value and the required reload value for a given Baud rate:

$$\text{Baud rate}_{SSC} = \frac{f_{CPU}}{2 \times [(SSCBR) + 1]}$$

$$SSCBR = \left(\frac{f_{CPU}}{2 \times \text{Baud rate}_{SSC}} \right) - 1$$

(SSCBR) represents the content of the reload register, taken as unsigned 16 bit integer.

Refer to the device datasheet for a table of Baud rates, reload values and resulting bit times.

SSCBR (F0B4h / 5Ah) ESFR Reset Value: 0000h

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

12.4 - Error Detection Mechanisms

The SSC is able to detect four different error conditions. Receive Error and Phase Error are detected in all modes, while Transmit Error and Baud rate Error only apply to slave mode. When an error is detected, the respective error flag is set. When the corresponding Error Enable bit is set, also an error interrupt request will be generated by setting SSCEIR (see figure below). The error interrupt handler may then check the error flags to determine the cause of the error interrupt. The error flags are not reset automatically (like SSCEIR), but rather must be cleared by software after servicing. This allows servicing of some error conditions via interrupt, while the others may be polled by software.

Note The error interrupt handler must clear the associated (enabled) error flag(s) to prevent repeated interrupt requests.

A **Receive Error** (Master or Slave mode) is detected, when a new data frame is completely received, but the previous data was not read out of the receive buffer register SSCRB. This condition sets the error flag SSCRE and, when enabled via SSCREN, the error interrupt request flag SSCEIR. The old data in the receive buffer SSCRB will be overwritten with the new value and is irretrievably lost.

A **Phase Error** (Master or Slave mode) is detected, when the incoming data at pin MRST (master mode) or MTSR (slave mode), sampled with the same frequency as the CPU clock, changes between one sample before and two samples after the latching edge of the clock signal (see "Clock Control"). This condition sets the error flag SSCPE and, when enabled via SSCPEN, the error interrupt request flag SSCEIR.

A **Baud Rate Error** (Slave mode) is detected, when the incoming clock signal deviates from the programmed Baud rate by more than 100%, it either is more than double or less than half the expected Baud rate. This condition sets the error flag SSCBE and, when enabled via SSCBEN, the error interrupt request flag SSCEIR. Using this error detection capability requires that the slave's Baud rate generator is programmed to the same Baud rate as the master device.

This feature detects false additional, or missing pulses on the clock line (within a certain frame).

Note If this error condition occurs and bit SSCAREN='1', an automatic reset of the SSC will be performed in case of this error. This is done to re-initialize the SSC, if too few or too many clock pulses have been detected.

A **Transmit Error** (Slave mode) is detected, when a transfer was initiated by the master (shift clock gets active), but the transmit buffer SSCTB of the slave was not updated since the last transfer. This condition sets the error flag SSCTE and, when enabled via SSCTEN, the error interrupt request flag SSCEIR. If a transfer starts while the transmit buffer is not updated, the slave will shift out the 'old' contents of the shift register, which normally is the data received during the last transfer.
This may lead to the corruption of the data on the transmit/receive line in half-duplex mode (open drain configuration), if this slave is not selected for transmission. This mode requires that slaves not selected for transmission only shift out ones, so their transmit buffers must be loaded with 'FFFFh' prior to any transfer.

Note A slave with push-pull output drivers, which is not selected for transmission, will normally have its output drivers switched. However, in order to avoid possible conflicts or misinterpretations, it is recommended to always load the slave's transmit buffer prior to any transfer (see Figure 105).

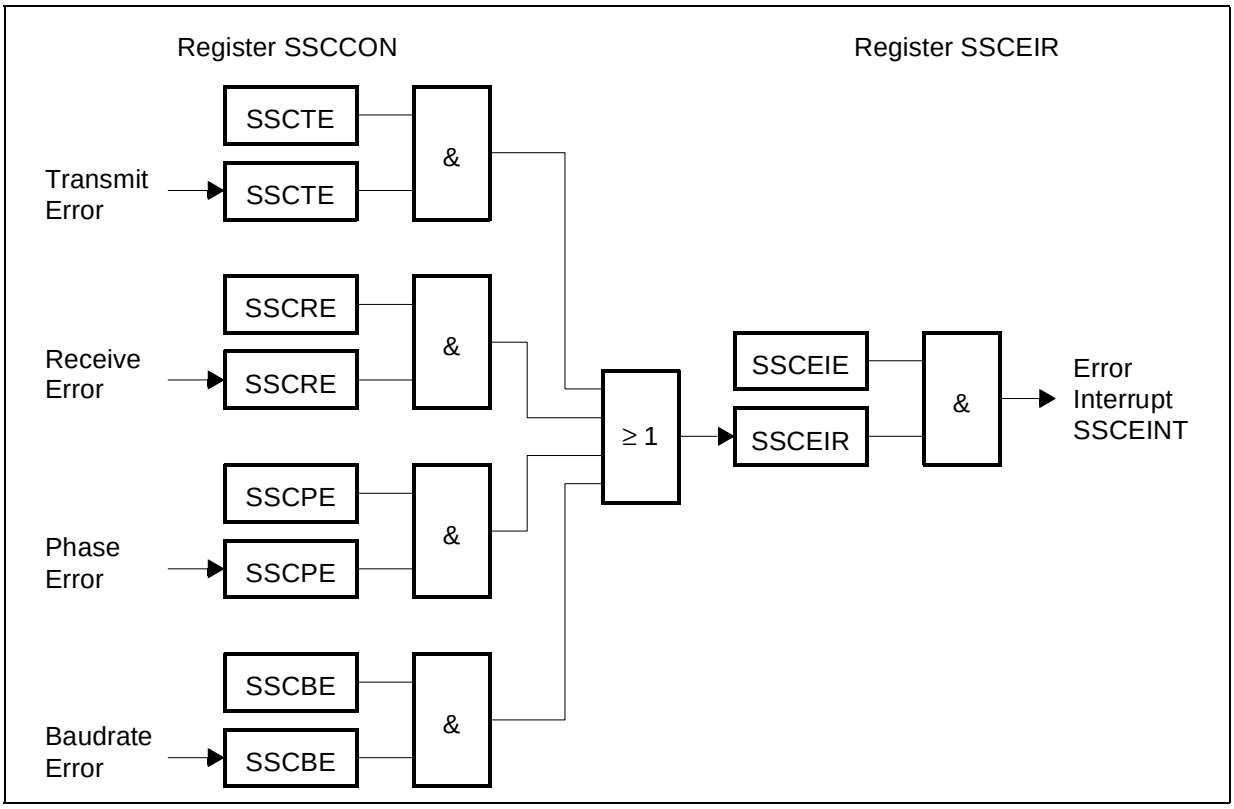
12.5 - SSC Interrupt Control

Three interrupt control registers are provided for serial channel SSC. Register SSCTIC controls the transmit interrupt, SSCRIC controls the receive interrupt and SSCEIC controls the error interrupt of serial channel SSC. Each interrupt source also has its own dedicated interrupt vector. SCTINT is the transmit interrupt vector, SCRINT is the receive interrupt vector, and SCEINT is the error interrupt vector.

The cause of an error interrupt request (receive, phase, Baud rate, transmit error) can be identified by the error status flags in control register SSCCON.

Note In contrary to the error interrupt request flag SSCEIR, the error status flags SSCxE are not reset automatically upon entry into the error interrupt service routine, but must be cleared by software.

Figure 105 : SSC error interrupt control



SSCTIC (FF72h / B9h) SFR Reset Value: - - 00h

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

SSCRIC (FF74h / BAh) SFR Reset Value: - - 00h

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

SSCEIC (FF76h / BBh) SFR Reset Value: - - 00h

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

Note Please refer to Section 6.1.3 - Interrupt Control Registers for an explanation of the control fields.

13 - WATCHDOG TIMER

The watchdog timer (WDT) provides recovery from software or hardware failure. If the software fails to service this timer before an overflow occurs, an internal reset sequence is initiated.

This internal reset will also pull the $\overline{\text{RSTOUT}}$ pin low, this resets the peripheral hardware which might have caused the malfunction. When the watchdog timer is enabled and is serviced regularly to prevent overflows, the watchdog timer supervises program execution. Overflow only occurs if the program does not progress properly.

The watchdog timer will time out, if a software error was due to hardware related failures. This prevents the controller from malfunctioning for longer than a user-specified time.

The watchdog timer provides two registers:

– Read-only timer register that contains the current count.

– Control register for initialization.

The watchdog timer is a 16-bit up counter which can be clocked with the CPU clock (f_{CPU}) either divided by 2 or divided by 128. This 16-bit timer is realized as two concatenated 8-bit timers (see Figure 107).

The upper 8 bits of the watchdog timer can be preset to a user-programmable value by a watchdog service access, in order to program the watchdog expire time. The lower 8 bits are reset on each service access.

Figure 106 : SFRs and port pins associated with the watchdog timer

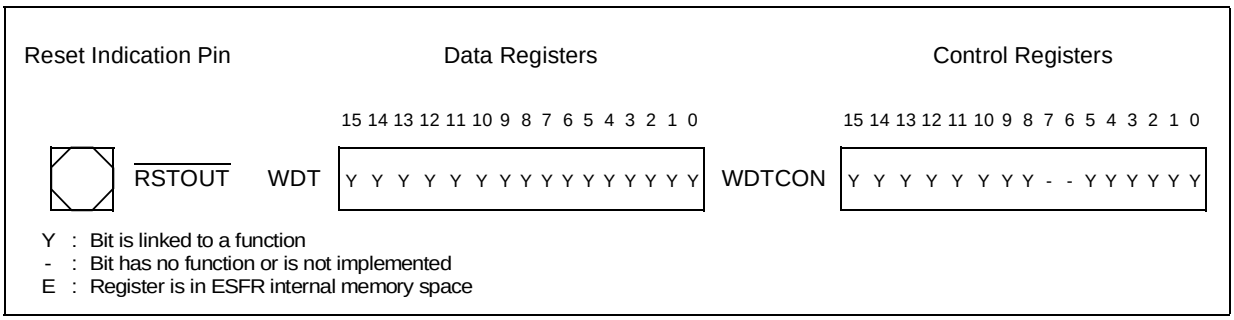
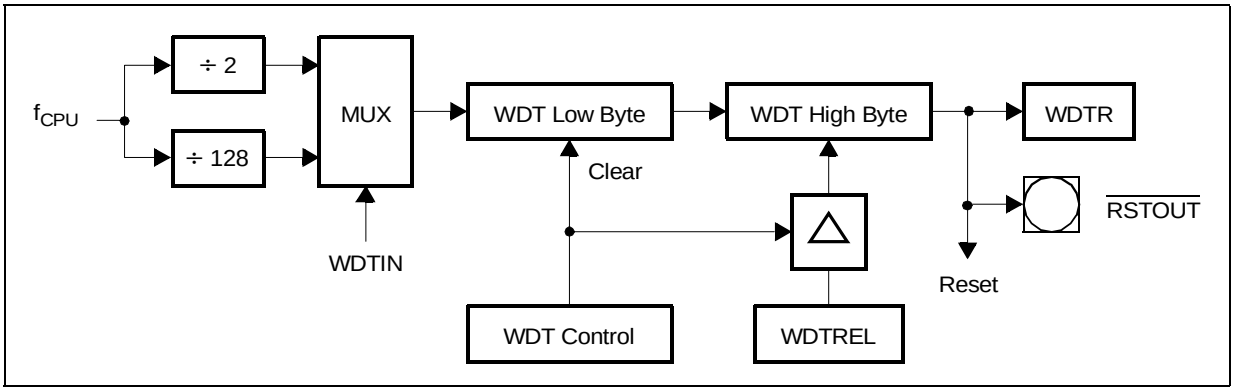


Figure 107 : Watchdog timer block diagram



13.1 - Operation of the Watchdog Timer

The current count value of the Watchdog Timer is contained in the Watchdog Timer Register WDT, which is a bit-addressable read-only register. The operation of the Watchdog Timer is controlled by its bit-addressable Watchdog Timer Control Register WDTCON. This register specifies the reload value for the high byte of the timer, selects the input clock prescaling factor and provides a flag that indicates a watchdog timer overflow.

WDTCON (FFAEh / D7h) SFR Reset Value: 00xxh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WDTREL								-	-	**PONR**	**LHWR**	**SHWR**	**SWR**	**WDTR**	WDTIN
RW										RW	RW	RW	RW	RW	RW

Bit	Function
WDTIN	**Watchdog Timer Input Frequency Selection** '0': Input Frequency is $f_{CPU}/2$. '1': Input Frequency is $f_{CPU}/128$.
WDTR[1]	**Watchdog Timer Reset Indication Flag** Set by the watchdog timer on an overflow. Cleared by a hardware reset or by the SRVWDT instruction.
SWR[1]	**Software Reset Indication Flag** Set by the SRST execution. Cleared by the EINIT instruction.
SHWR[1]	**Short Hardware Reset Indication Flag** Set by the input $\overline{\text{RSTIN}}$. Cleared by the EINIT instruction.
LHWR[1]	**Long Hardware Reset Indication Flag** Set by the input $\overline{\text{RSTIN}}$. Cleared by the EINIT instruction.
PONR[1- 2]	**Power-On (Asynchronous) Reset Indication Flag** Set by the input $\overline{\text{RSTIN}}$ if a power-on condition has been detected. Cleared by the EINIT instruction.

Notes: 1. More than one reset indication flag may be set. After EINIT, all flags are cleared.
2. Power-on is detected when a rising edge from Vcc = 0 V to Vcc > 2.0 V is recognized.

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 via 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 wait-states. 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) x 2^8]. Instruction SRVWDT has been encoded in such a way that the chance of unintentionally servicing the watchdog timer (eg. by fetching and executing a bit pattern from a wrong location) is minimized. When instruction SRVWDT does not match the format for protected instructions, the Protection Fault Trap will be entered, rather than the instruction be executed.

The PONR flag of WDTCON register is set if the output voltage of the internal 3.3V supply falls below the threshold (typically 2V) of the power-on detection circuit. This circuit is efficient to detect major failures of the external 5V supply but if the internal 3.3V supply does not drop under 2 volts, the PONR flag is not set. This could be the case on fast switch-off / switch-on of the 5V supply. The time needed for such a sequence to activate the PONR flag depends on the value of the capacitors connected to the supply and on the exact value of the internal threshold of the detection circuit.

Table 39 : WDTCON Bits Value on Different Resets

Reset Source	PONR	LHWR	SHWR	SWR	WDTR
Power On Reset	X	X	X	X	
Power on after partial supply failure	1	X	X	X	
Long Hardware Reset		X	X	X	
Short Hardware Reset			X	X	
Software Reset				X	
Watchdog Reset				X	X

Notes: 1. PONR bit may not be set for short supply failure.

2. For power-on reset and reset after supply partial failure, asynchronous reset must be used.

In case of bi-directional reset is enabled, and if the $\overline{\text{RSTIN}}$ pin is latched low after the end of the internal reset sequence, then a Short hardware reset, a software reset or a watchdog reset will trigger a Long hardware reset. Thus, Reset Indications flags will be set to indicate a Long Hardware Reset.

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

Each time it is serviced by the application software, the high byte of the watchdog timer is reloaded. For security, rewrite WDTCON each time before the watchdog timer is serviced.

The time period for an overflow of the watchdog timer is programmable in two ways:

– **The input frequency** to the watchdog timer can be selected via bit WDTIN in register WDTCON to be either f_{CPU} / 2 or f_{CPU} / 128.

– **The reload value** WDTREL for the high byte of WDT can be programmed in register WDTCON.

The period P_{WDT} between servicing the watchdog timer and the next overflow can therefore be determined by the following formula:

$$P_{WDT} = \frac{2^{[1 + (WDTIN) \times 6]} \times [2^{16} - (WDTREL) \times 2^8]}{f_{CPU}}$$

Refer to the device datasheet for a table of watchdog timer ranges. For security, you are advised to rewrite WDTCON each time before the watchdog timer is serviced.

The Table 40 shows the watchdog time range for 40 MHz CPU clock.

Table 40 : WDTREL Reload Value

Reload value in WDTREL	Prescaler for f_{CPU} = 40MHz	
	2 (WDTIN = '0')	128 (WDTIN = '1')
FFh	12.8µs	819.2ms
00h	3.276ms	209.7ms

14 - BOOTSTRAP LOADER

The built-in bootstrap loader of the ST10F280 provides a mechanism to load the startup program through the serial interface after reset. In this case, no external or internal Flash Memory is required for the initialization code starting at location 00'0000h.

The bootstrap loader moves code/data into the internal RAM, but can also transfer data via the serial interface into an external RAM using a second level loader routine. Flash Memory (internal or external) is not necessary, but it may be used to provide lookup tables or "core-code" like a set of general purpose subroutines for I/O operations, number crunching, system initialization, etc. (see Figure 108).

The bootstrap loader can be used to load the complete application software into ROMless systems, to load temporary software into complete systems for testing or calibration, or to load a programming routine for Flash devices.

The BSL mechanism can be used for standard system startup as well as for special occasions like system maintenance (firmer update) or end-of-line programming or testing.

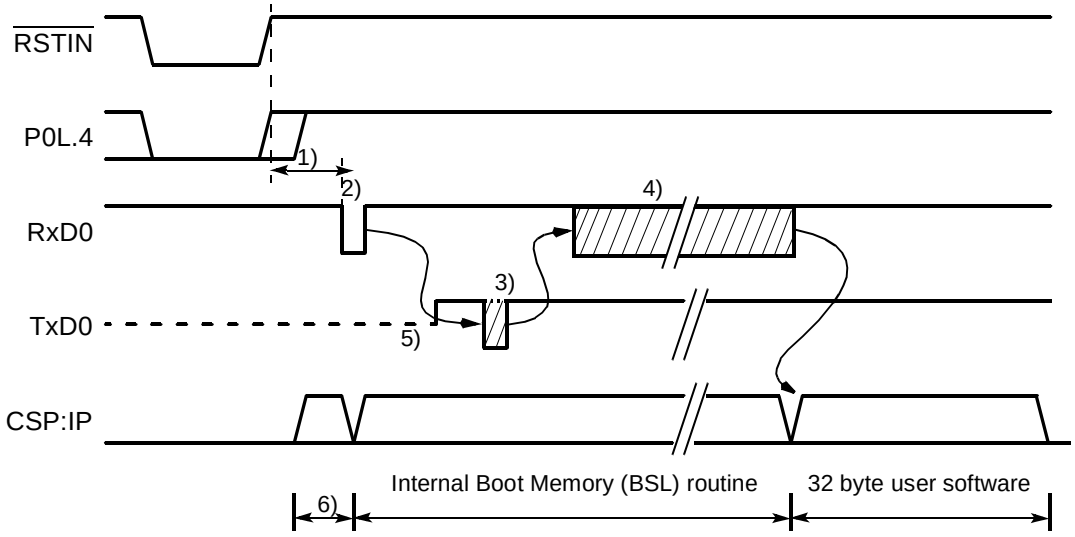
Entering the bootstrap loader

The ST10F280 enters BSL mode when pin P0L.4 is sampled low at the end of a hardware reset. In this case the built-in bootstrap loader is activated independent of the selected bus mode. The bootstrap loader code is stored in a special Boot-ROM. No part of the standard mask Memory or Flash Memory area is required for this.

After entering BSL mode and the respective initialization the ST10F280 scans the RxD0 line to receive a zero byte, one start bit, eight '0' data bits and one stop bit. From the duration of this zero byte it calculates the corresponding Baud rate factor with respect to the current CPU clock, initializes the serial interface ASC0 accordingly and switches pin TxD0 to output.

Using this Baud rate, an identification byte is returned to the host that provides the loaded data.This identification byte identifies the device to be booted. Refer to the datasheet for specific device information.

Figure 108 : Bootstrap loader sequence



1) BSL initialization time.

2) Zero byte (1 start bit, eight '0' data bits, 1 stop bit), sent by host.

3) Identification byte (D5h), sent by ST10F280.

4) 32 bytes of code / data, sent by host.

5) Caution: TxD0 is only driven a certain time after reception of the zero byte.

6) Internal Boot ROM.

When the ST10F280 has entered BSL mode, the following configuration is automatically set (values that deviate from the normal reset values, are ***marked***):

Watchdog Timer:	***Disabled***	Register SYSCON:	0E00h
Context Pointer CP:	FA00h	Register STKUN:	FA40h
Stack Pointer SP:	FA40h	Register STKOV:	FA0Ch 0<->C
Register S0CON:	***8011h***	Register BUSCON0:	acc. to startup configuration
Register S0BG:	acc. to '00' byte	P3.10 / TXD0:	'**1**'
		DP3.10:	'**1**'

In this case, the watchdog timer is disabled, so the bootstrap loading sequence is not time limited. Pin TXD0 is configured as output, so the ST10F280 can return the identification byte.

Even if the internal Flash is enabled, no code can be executed out of it.

The hardware that activates the BSL during reset may be a simple pull-down resistor on P0L.4 for systems that use this feature upon every hardware reset. A switchable solution (via jumper or an external signal) can be used for systems that only temporarily use the bootstrap loader (see Figure 109).

After sending the identification byte the ASC0 receiver is enabled and is ready to receive the initial 32 bytes from the host. A half duplex connection is therefore sufficient to feed the BSL.

Memory Configuration After Reset

The configuration (and the accessibility) of the ST10F280's memory areas after reset in Bootstrap-Loader mode differs from the standard case. Pin $\overline{EA}$ is not evaluated when BSL mode is selected, and accesses to the internal Flash area are partly redirected, while the ST10F280 is in BSL mode (see Figure 110). All code fetches are made from the special Boot-ROM, while data accesses read from the internal user Flash. Data accesses will return undefined values on ROMless devices.

The code in the Boot-ROM is not an invariant feature of the ST10F280. User software should not try to execute code from the internal Flash area while the BSL mode is still active, as these fetches will be redirected to the Boot-ROM. The Boot-ROM will also "move" to segment 1, when the internal Flash area is mapped to segment 1 (see Figure 110).

Figure 109 : Hardware provisions to activate the BSL

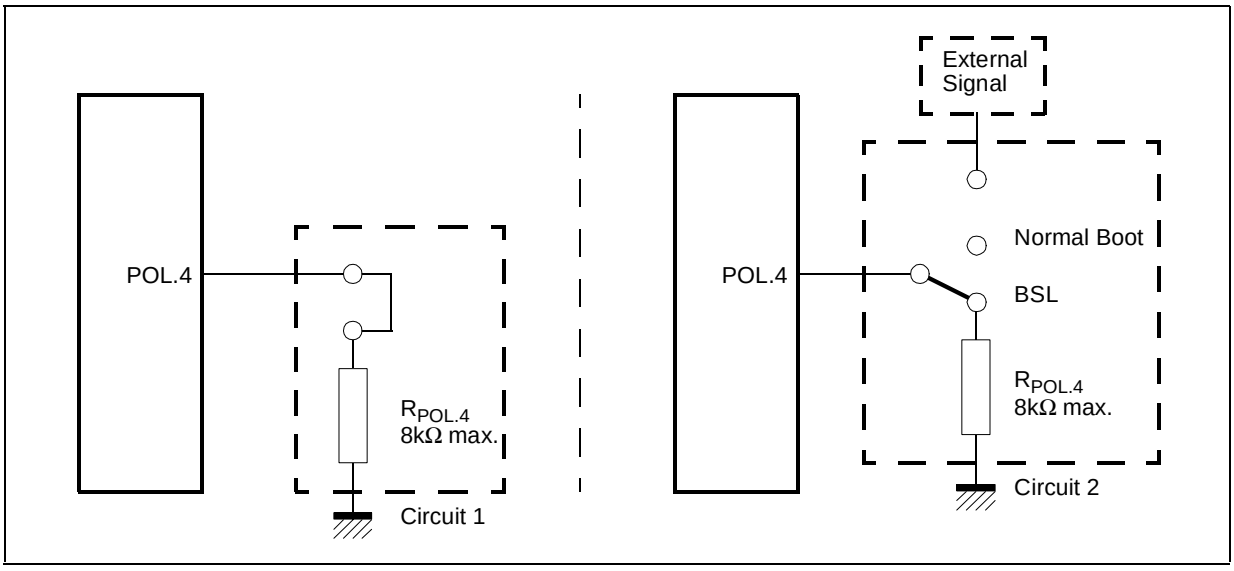
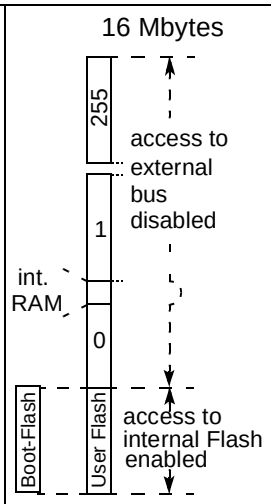
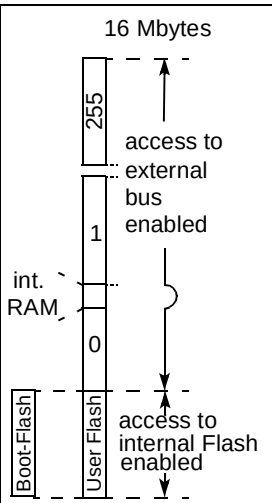
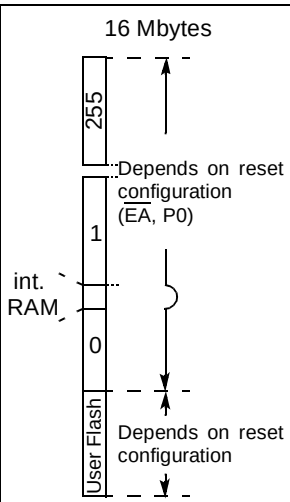


Figure 110 : Memory configuration after reset

			
BSL mode active	Yes (P0L.4='0')	Yes (P0L.4='0')	No (P0L.4='1')
$\overline{EA}$ pin	High	Low	Access to application
Code fetch from internal Flash area	Test-Flash access	Test-Flash access	User Flash access
Data fetch from internal Flash area	User Flash access	User Flash access	User Flash access

Loading the Startup Code

After sending the identification byte the BSL enters a loop to receive 32 bytes via ASC0. These bytes are stored sequentially into locations 00'FA40h through 00'FA5Fh of the internal RAM. So up to 16 instructions may be placed into the RAM area. To execute the loaded code the BSL then jumps to location 00'FA40h, which is the first loaded instruction.

The bootstrap loading sequence is now terminated, the ST10F280 remains in BSL mode, however. Most probably the initially loaded routine will load additional code or data, as an average application is likely to require substantially more than 16 instructions. This second receive loop may directly use the pre-initialized interface ASC0 to receive data and store it to arbitrary user-defined locations.

This second level of loaded code may be the final application code. It may also be another, more sophisticated, loader routine that adds a transmission protocol to enhance the integrity of the loaded code or data. It may also contain a code sequence to change the system configuration and enable the bus interface to store the received data into external memory.

This process may go through several iterations or may directly execute the final application. In all cases the ST10F280 will still run in BSL mode, that means with the watchdog timer disabled and limited access to the internal Flash area.

All code fetches from the internal Flash area (00'0000h...00'7FFFh or 01'0000h...01'7FFFh, if mapped to segment 1) are redirected to the special Boot-ROM. Data fetches access will access the internal Boot-ROM of the ST10F280, if any is available, but will return undefined data on ROMless devices.

Exiting Bootstrap Loader Mode

In order to execute a program in normal mode, the BSL mode must be terminated first. The ST10F280 exits BSL mode upon a software reset (ignores the level on P0L.4) or a hardware reset (P0L.4 must be high). After a reset the ST10F280 will start executing from location 00'0000h of the internal Flash or the external memory, as programmed via pin $\overline{EA}$.

Choosing the Baud rate for the BSL

The calculation of the serial Baud rate for ASC0 from the length of the first zero byte that is received, allows the operation of the bootstrap loader of the ST10F280 with a wide range of Baud rates. However, the upper and lower limits have to be kept, in order to insure proper data transfer.

$$B_{ST10F280} = \frac{f_{CPU}}{32 \times (S0BRL + 1)}$$

The ST10F280 uses timer T6 to measure the length of the initial zero byte. The quantization uncertainty of this measurement implies the first deviation from the real Baud rate, the next deviation is implied by the computation of the S0BRL reload value from the timer contents. The formula below shows the association:

$$S0BRL = \frac{T6 - 36}{72} \quad , \quad T6 = \frac{9}{4} \times \frac{f_{CPU}}{B_{Host}}$$

For a correct data transfer from the host to the ST10F280 the maximum deviation between the internal initialized Baud rate for ASC0 and the real Baud rate of the host should be below 2.5%. The deviation (F_B, in percent) between host Baud rate and ST10F280 Baud rate can be calculated via the formula below:

$$F_B = \left| \frac{B_{Contr} - B_{Host}}{B_{Contr}} \right| \times 100 \ \% \ ,$$

$$F_B \leq 2.5 \ \%$$

Note Function (F_B) does not consider the tolerances of oscillators and other devices supporting the serial communication.

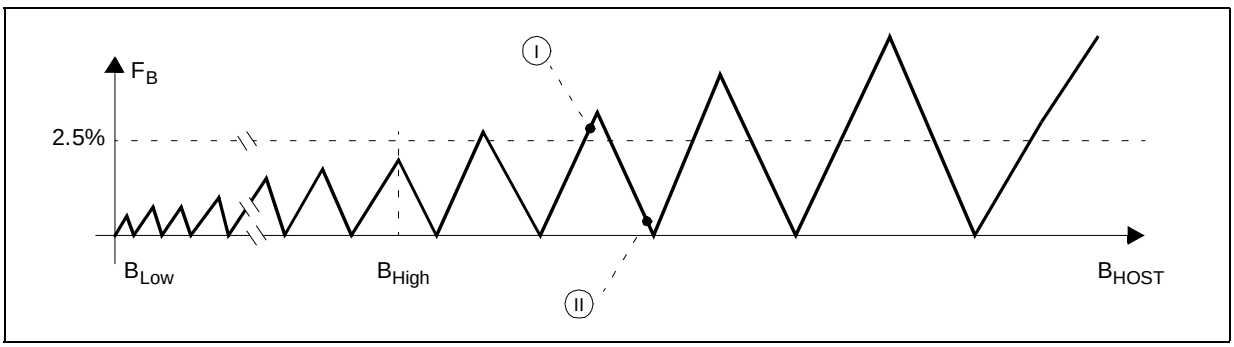
This Baud rate deviation is a nonlinear function depending on the CPU clock and the Baud rate of the host. The maxima of the function (F_B) increase with the host Baud rate due to the smaller Baud rate pre-scaler factors and the implied higher quantization error (see Figure 111).

The minimum Baud rate (B_{Low} in the Figure 111) is determined by the maximum count capacity of timer T6, when measuring the zero byte, and it depends on the CPU clock. Using the maximum T6 count 2^{16} in the formula the minimum Baud rate can be calculated. The lowest standard Baud rate in this case would be 1200 Baud. Baud rates below B_{Low} would cause T6 to overflow. In this case ASC0 cannot be initialized properly.

The maximum Baud rate (B_{High} in the Figure 111) is the highest Baud rate where the deviation still does not exceed the limit, so all Baud rates between B_{Low} and B_{High} are below the deviation limit. The maximum standard Baud rate that fulfills this requirement is 19200 Baud.

Higher Baud rates, however, may be used as long as the actual deviation does not exceed the limit. A certain Baud rate (marked 'I' in Figure 111) may violate the deviation limit, while an even higher Baud rate (marked 'II' in Figure 111) stays very well below it. This depends on the host interface.

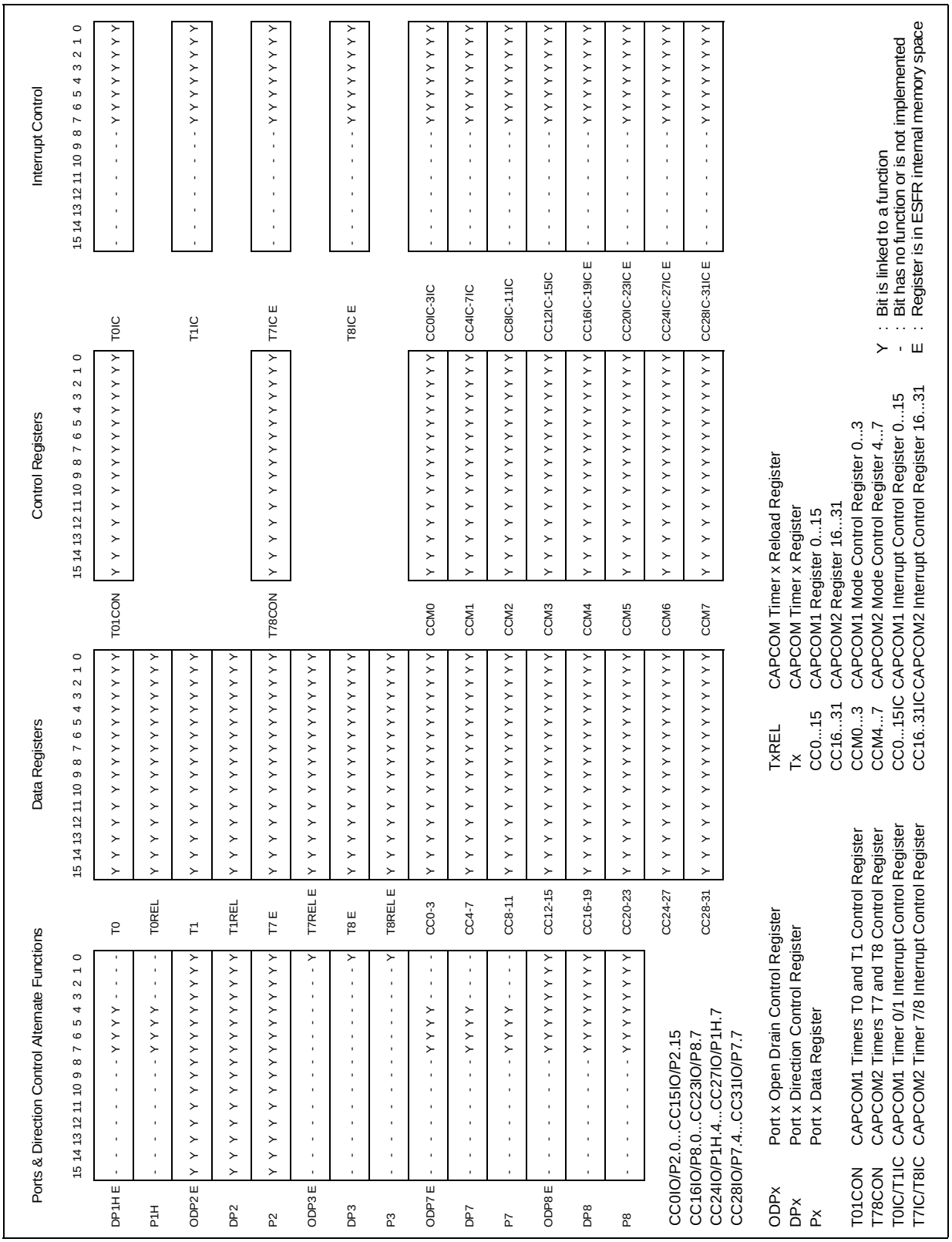
Figure 111 : Baud rate deviation between host and ST10F280



15 - THE CAPTURE / COMPARE UNITS

The ST10F280 provides two, almost identical, Capture / Compare (CAPCOM) units which differ, only in the way they are connected to the I/O pins. They provide 32 channels which interact with 4 timers. The CAPCOM units **capture** the contents of a timer on specific internal or external events, or they **compare** a timer's content with given values and modify output signals in case of a match. They support generation and control of timing sequences on up to 16 channels per unit with a minimum of software intervention. For programming, the term 'CAPCOM unit' refers to a set of SFRs associated to the peripheral, including the port pins which may be used for alternate input / output functions including their direction control bits.

Figure 112 : SFRs and Port Pins associated with the CAPCOM units



A CAPCOM unit handles high speed I/O tasks such as pulse and waveform generation, pulse width modulation, or recording of the time at which specific events occur. It also allows the implementation of up to 16 software timers. The maximum resolution of the CAPCOM units is calculated with the formula in Section 15.1 - CAPCOM Timers and is specified in the device datasheet.

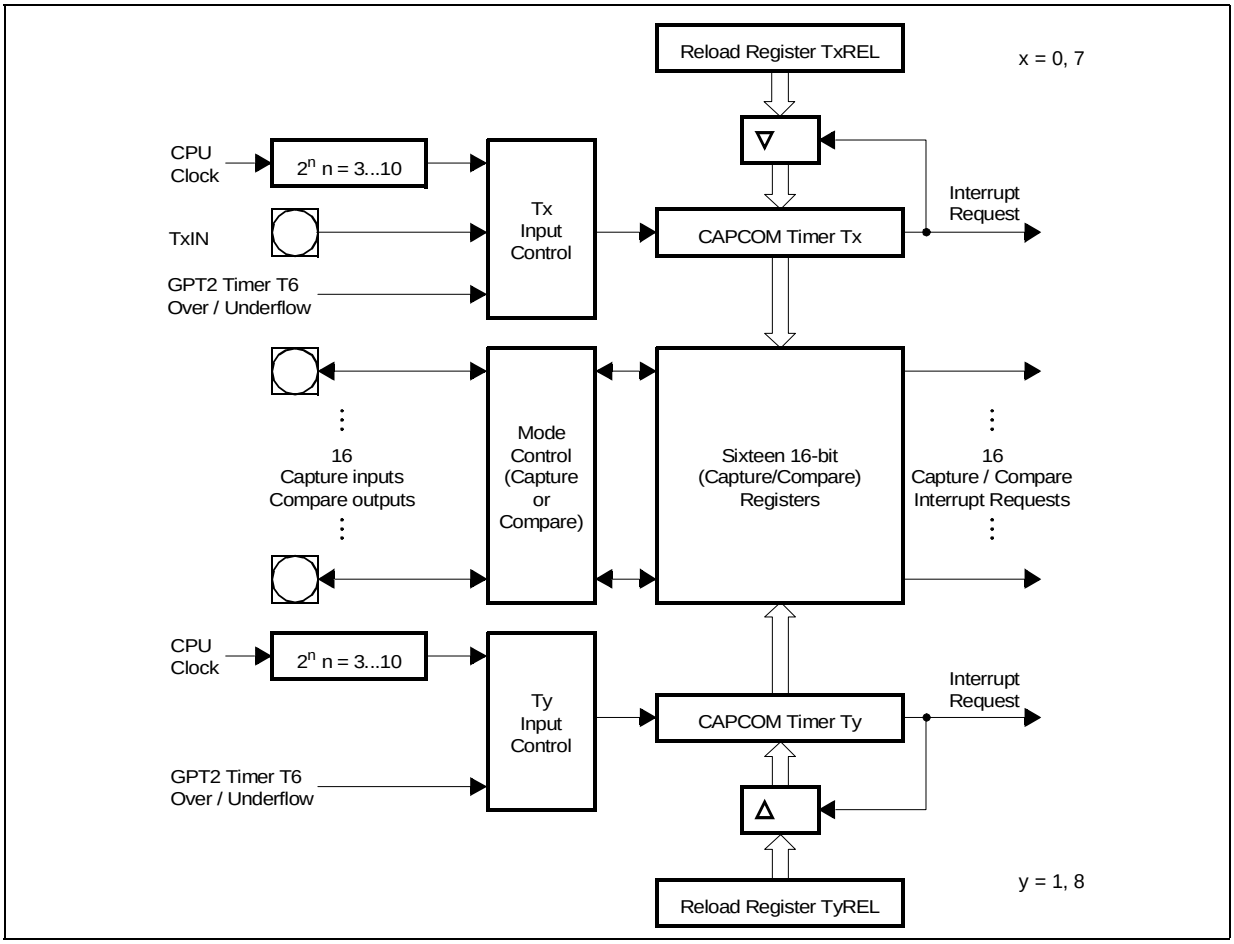
Each CAPCOM unit consists of two 16-bit timers (T0 / T1 in CAPCOM1, T7 / T8 in CAPCOM2), each with its own reload register (TxREL), and a bank of sixteen dual purpose 16-bit capture / compare registers (CC0 through CC15 in CAPCOM1, CC16 through CC31 in CAPCOM2).

The input clock for the CAPCOM timers is programmable to several pre-scaled values of the CPU clock, or it can be derived from an overflow / underflow of timer T6 in block GPT2. T0 and T7 may also operate in counter mode (from an external input) where they can be clocked by external events.

Each capture / compare register may be programmed individually for capture or compare function, and each register may be allocated to either timer of the associated unit. Each capture / compare register has one port pin associated with it which serves as an input pin for the capture function or as an output pin for the compare function (except for CC27...CC24 on P1H.7...P1H.4, which only provide the capture function). The capture function causes the current timer contents to be latched into the respective capture / compare register triggered by an event (transition) on its associated port pin. The compare function may cause an output signal transition on that port pin whose associated capture / compare register matches the current timer contents. Specific interrupt requests are generated upon each capture / compare event or upon timer overflow.

Figure 113 shows the basic structure of the two CAPCOM units.

Figure 113 : CAPCOM unit block diagram



Note The CAPCOM2 unit provides 16 capture inputs, but only 12 compare outputs.

15.1 - CAPCOM Timers

The primary use of the timers T0 / T1 and T7 / T8 is to provide two independent time bases for the capture / compare registers of each unit, but they may also be used independent of the capture / compare registers. The basic structure of the four timers is identical, while the selection of input signals is different for timers T0 / T7 and timers T1 / T8.

Figure 114 : Block diagram of CAPCOM timers T0 and T7

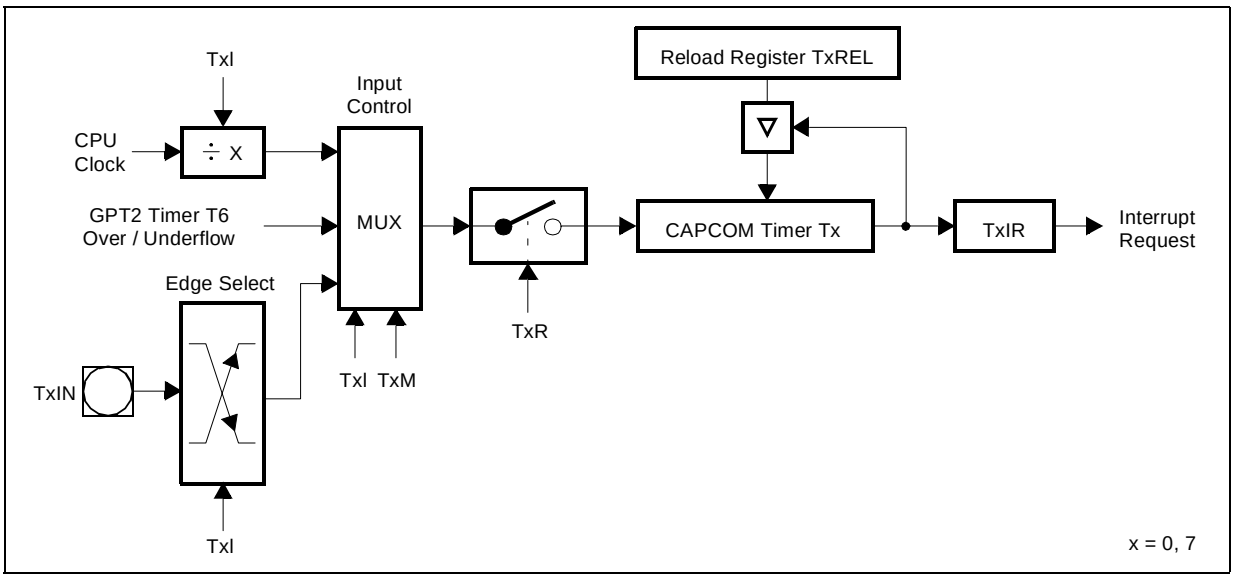
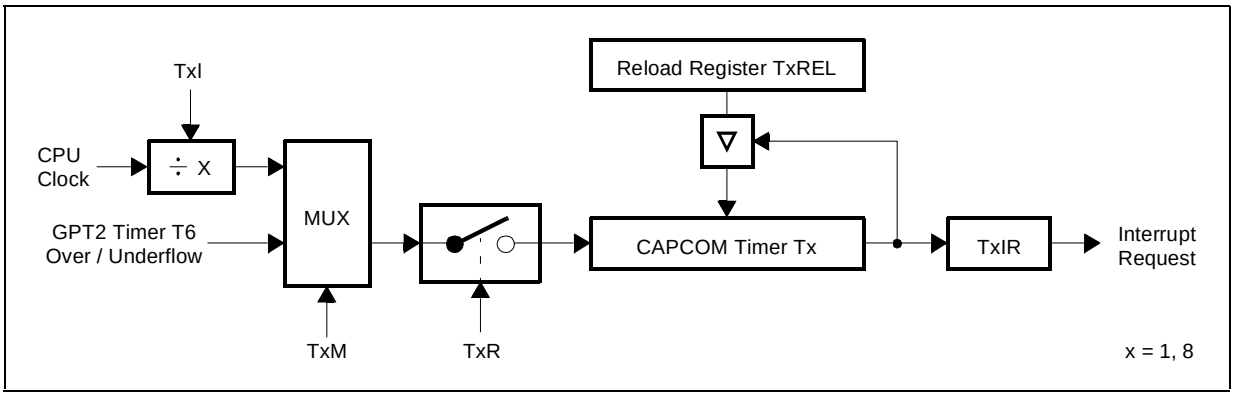


Figure 115 : Block diagram of CAPCOM timers T1 and T8



Note When an external input signal is connected to the input lines of both T0 and T7, these timers count the input signal synchronously. Thus the two timers can be regarded as one timer whose contents can be compared with 32 capture registers.

The functions of the CAPCOM timers are controlled via the bit-addressable 16-bit control registers T01CON and T78CON. The high-byte of T01CON controls T1, the low-byte of T01CON controls T0, the high-byte of T78CON controls T8, the low-byte of T78CON controls T7. The control options are identical for all four timers (except for external input).

T01CON (FF50h / A8h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	**T1R**	-	-	**T1M**	**T1I**			-	**T0R**	-	-	**T0M**	**T0I**		
	RW			RW	RW				RW			RW	RW		

T78CON (FF20h / 90h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	**T8R**	-	-	**T8M**	**T8I**			-	**T7R**	-	-	**T7M**	**T7I**		
	RW			RW	RW				RW			RW	RW		

Bit	Function
TxI	**Timer / Counter x Input Selection** Timer Mode (TxM='0')Input Frequency = $f_{CPU} / 2^{[(TxI)+3]}$ See also table below for examples. Counter Mode (TxM='1'):X00 Overflow / Underflow of GPT2 Timer 6 X01 Positive (rising) edge on pin TxIN 1) X10 Negative (falling) edge on pin TxIN 1) X11 Any edge (rising and falling) on pin TxIN 1)
TxM	**Timer / Counter x Mode Selection** '0': Timer Mode (Input derived from internal clock) '1': Counter Mode (Input from External Input or T6)
TxR	**Timer / Counter x Run Control** '0': Timer / Counter x is disabled '1': Timer / Counter x is enabled

Note 1) This selection is available for timers T0 and T7. Timers T1 and T8 will stop at this selection!

The run flags T0R, T1R, T7R and T8R enable or disable the timers. The following description of the timer modes and operation always applies to the enabled state of the timers, the respective run flag is assumed to be set to '1'.

In all modes, the timers are always counting upward. The current timer values are accessible for the CPU in the timer registers Tx, which are non bit-addressable SFRs. When the CPU writes to a register Tx in the state immediately before the respective timer increment, a reload is to be performed, the CPU write operation has priority and the increment or reload is disabled to garantee correct timer operation.

Timer Mode

The bit TxM in SFRs T01CON and T78CON selects the timer mode or the counter mode. In timer mode (TxM='0'), the input clock of a timer is derived from the internal CPU clock divided by a programmable pre-scaler.

The different options of the pre-scaler of each timer are selected separately by the bit-fields TxI.

The input frequencies f_{Tx} for Tx are determined as a function of the CPU clock as follows, where (TxI) represents the contents of the bit-field TxI:

$$f_{Tx} = \frac{f_{CPU}}{2^{[(TxI)+3]}}$$

When a timer overflows from FFFFh to 0000h it is reloaded with the value stored in its respective reload register TxREL.

The reload value determines the period P_{Tx} between two consecutive overflows of Tx as follows:

$$P_{Tx} = \frac{[2^{16} - (TxREL)] \times 2^{[(TxI)+3]}}{f_{CPU}}$$

The timer resolutions against pre-scaler option in TxI are listed in the table below.

	Timer Input Selection TxI							
	000b	**001b**	**010b**	**011b**	**100b**	**101b**	**110b**	**111b**
Pre-scaler for f_{CPU}	8	16	32	64	128	256	512	1024
Resolution in CPU clock cycles	8	16	32	64	128	256	512	1024

Refer to the device datasheet for a table of timer input frequencies, resolution and periods for each pre-scaler option in TxI.

After a timer has been started by setting its run flag (TxR) to '1', the first increment will occur within the time interval which is defined by the selected timer resolution. All further increments occur exactly after the time defined by the timer resolution.

When both timers of a CAPCOM unit are to be incremented or reloaded at the same time T0 is always serviced one CPU clock before T1, T7 before T8, respectively.

Counter Mode

The bit TxM in SFRs T01CON and T78CON select between timer or counter mode for the respective timer. In Counter mode (TxM='1') the input clock for a timer can be derived from the overflows / underflows of timer T6 in block GPT2. In addition, timers T0 and T7 can be clocked by external events. Either a positive, a negative, or both a positive and a negative transition at pin T0IN (alternate input function of port pin P3.0) or T7IN (alternate input function of port pin P2.15), respectively, can be selected to cause an increment of T0 / T7.

When T1 or T8 is programmed to run in counter mode, bit-field TxI is used to enable the overflows / underflows of timer T6 as the count source. This is the only option for T1 and T8, and it is selected by the combination TxI=X00b. When bit-field TxI is programmed to any other combination, the respective timer (T1 or T8) will stop.

When T0 or T7 is programmed to run in counter mode, bit-field TxI is used to select the count source and transition (if the source is the input pin) which should cause a count trigger (see description of TxyCON for the possible selections).

Note In order to use pin T0IN or T7IN as external count input pin, the respective port pin must be configured as input, and the corresponding direction control bit (DP3.0 or DP2.15) must be cleared ('0').
If the respective port pin is configured as output, the associated timer may be clocked by modifying the port output latches P3.0 or P2.15 via software, for example for testing purposes.

The maximum external input frequency to T0 or T7 in counter mode is f_{CPU} / 16. To ensure that a signal transition is properly recognized at the timer input, an external count input signal should be held for at least 8 CPU clock cycles before it changes its level again. The incremented count value appears in SFR T0 / T7 within 8 CPU clock cycles after the signal transition at pin TxIN.

Reload

A reload of a timer with the 16 bit value stored in its associated reload register in both modes is performed each time a timer would overflow from FFFFh to 0000h. In this case the timer does not wrap around to 0000h, but rather is reloaded with the contents of the respective reload register TxREL. The timer then resumes incrementing starting from the reloaded value.

The reload registers TxREL are not bit-addressable.

15.2 - CAPCOM Unit Timer Interrupts

Upon a timer overflow the corresponding timer interrupt request flag TxIR for the respective timer will be set. This flag can be used to generate an interrupt or trigger a PEC service request, when enabled by the respective interrupt enable bit TxIE.

Each timer has its own bit-addressable interrupt control register (TxIC) and its own interrupt vector (TxINT). The organization of the interrupt control registers TxIC is identical with the other interrupt control registers.

T0IC (FF9Ch / CEh) SFR Reset Value: - - 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	T0IR	T0IE	ILVL				GLVL	
								RW	RW	RW				RW	

T1IC (FF9Eh / CFh) SFR Reset Value: - - 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	T1IR	T1IE	ILVL				GLVL	
								RW	RW	RW				RW	

T7IC (F17Ah / BEh) ESFR Reset Value: - - 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	T7IR	T7IE	ILVL				GLVL	
								RW	RW	RW				RW	

T8IC (F17Ch / BFh) ESFR Reset Value: - - 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	T8IR	T8IE	ILVL				GLVL	
								RW	RW	RW				RW	

Note Refer to the General Interrupt Control Register description for an explanation of the control fields.

15.3 - Capture / Compare Registers

The 16-bit capture / compare registers CC0 through CC31 are used as data registers for capture or compare operations with respect to timers T0 / T1 and T7 / T8. The capture / compare registers are not bit-addressable.

Each of the registers CC0...CC31 may be individually programmed for capture mode or one of 4 different compare modes (except for CC24...CC27), and may be allocated individually to one of the two timers of the respective CAPCOM unit (T0 or T1, and T7 or T8, respectively). A special combination of compare modes additionally allows the implementation of a 'double-register' compare mode.

When capture or compare operation is disabled for one of the CCx registers, it may be used for general purpose variable storage.

The functions of the 32 capture / compare registers are controlled by the 8 mode control registers named CCM0...CCM7 which are all organized identically (see description below). These 16-bit registers are bit-addressable.

Each register contains bit for mode selection and timer allocation of four capture / compare registers.

Capture / compare mode registers for the CAPCOM1 unit (CC0...CC15)

CCM0 (FF52h / A9h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC3		CCMOD3		ACC2		CCMOD2		ACC1		CCMOD1		ACC0		CCMOD0	
RW		RW		RW		RW		RW		RW		RW		RW	

CCM1 (FF54h / AAh) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC7		CCMOD7		ACC6		CCMOD6		ACC5		CCMOD5		ACC4		CCMOD4	
RW		RW		RW		RW		RW		RW		RW		RW	

CCM2 (FF56h / ABh) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC11		CCMOD11		ACC10		CCMOD10		ACC9		CCMOD9		ACC8		CCMOD8	
RW		RW		RW		RW		RW		RW		RW		RW	

CCM3 (FF58h / ACh) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC15		CCMOD15		ACC14		CCMOD14		ACC13		CCMOD13		ACC12		CCMOD12	
RW		RW		RW		RW		RW		RW		RW		RW	

Capture / compare mode registers for the CAPCOM2 unit (CC16...CC31)

CCM4 (FF22h / 91h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC19		CCMOD19		ACC18		CCMOD18		ACC17		CCMOD17		ACC16		CCMOD16	
RW		RW		RW		RW		RW		RW		RW		RW	

CCM5 (FF24h / 92h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC23		CCMOD23		ACC22		CCMOD22		ACC21		CCMOD21		ACC20		CCMOD20	
RW		RW		RW		RW		RW		RW		RW		RW	

CCM6 (FF26h / 93h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ACC27		CCMOD27		ACC26		CCMOD26		ACC25		CCMOD25		ACC24		CCMOD24	
RW		RW		RW		RW		RW		RW		RW		RW	

CCM7 (FF28h / 94h) SFR Reset Value: 0000h

15	14 13 12	11	10 9 8	7	6 5 4	3	2 1 0
ACC31	CCMOD31	ACC30	CCMOD30	ACC29	CCMOD29	ACC28	CCMOD28
RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
CCMODx	**Mode Selection for Capture / Compare Register CCx** The available capture / compare modes are listed in the table below.
ACCx	**Allocation bit for Capture / Compare Register CCx** '0': CCx allocated to Timer T0 (CAPCOM1) / Timer T7 (CAPCOM2) '1': CCx allocated to Timer T1 (CAPCOM1) / Timer T8 (CAPCOM2)

15.3.1 - Selection of Capture Modes and Compare Modes

CCMODx	Selected Operating Mode
0 0 0	Disable Capture and Compare Modes The respective CAPCOM register may be used for general variable storage.
0 0 1	Capture on Positive Transition (Rising Edge) at Pin CCxIO
0 1 0	Capture on Negative Transition (Falling Edge) at Pin CCxIO
0 1 1	Capture on Positive and Negative Transition (Both Edges) at Pin CCxIO
1 0 0	Compare Mode 0:Interrupt Only Several interrupts per timer period. Enables double-register compare mode for registers CC8...CC15 and CC24...CC31.
1 0 1	Compare Mode 1:Toggle Output Pin on each Match Several compare events per timer period. This mode is required for double-register compare mode for registers CC0...CC7 and CC16...CC23.
1 1 0	Compare Mode 2:Interrupt Only Only one interrupt per timer period.
1 1 1	Compare Mode 3:Set Output Pin on each Match Reset output pin on each timer overflow. Only one interrupt per timer period.

The detailed discussion of the capture and compare modes is valid for all the capture / compare channels, so registers, bits and pins are only referenced by the place holder 'x'.

Note Capture / compare channels 24...27 generate an interrupt request but do not provide an output signal. The resulting exceptions are indicated in the following subsections.
A capture or compare event on channel 31 may be used to trigger a channel injection on the ST10F280's A / D converter if enabled.

15.4 - Capture Mode

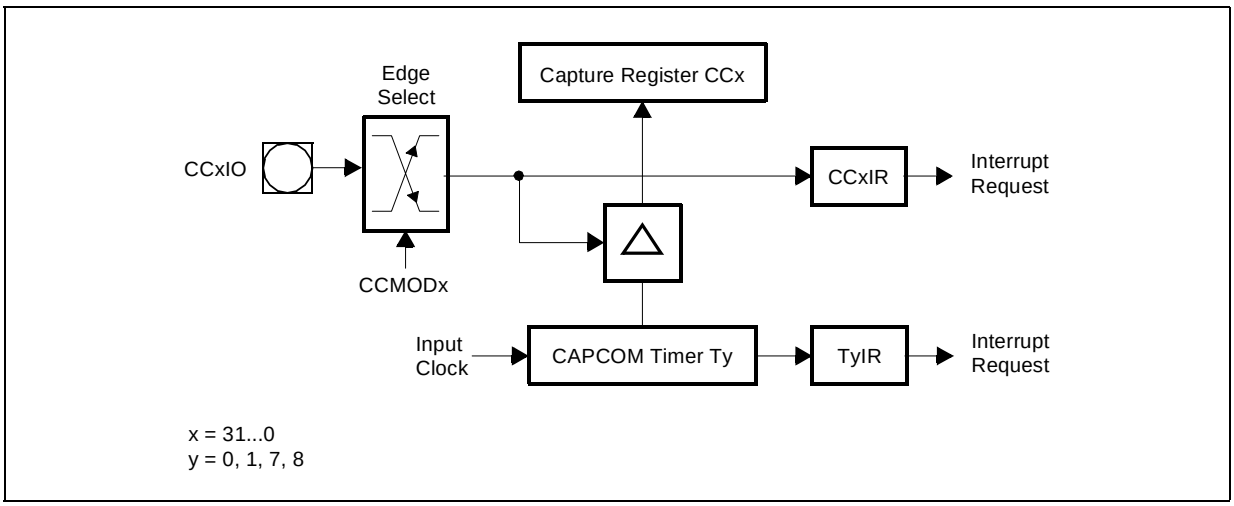
In response to an external event the content of the associated timer (T0 / T1 or T7 / T8, depending on the used CAPCOM unit and the state of the allocation control bit ACCx) is latched into the respective capture register CCx. The external event causing a capture can be programmed to be either a positive, a negative, or both a positive or a negative transition at the respective external input pin CCxIO.

The triggering transition is selected by the mode bit CCMODx in the respective CAPCOM mode control register. In any case, the event causing a capture will also set the respective interrupt request flag CCxIR, which can cause an interrupt or a PEC service request, when enabled (see Figure 116).

In order to use the respective port pin as external capture input pin CCxIO for capture register CCx, this port pin must be configured as input, the corresponding direction control bit by setting to '0'. To ensure that a signal transition is properly recognized, an external capture input signal should be held for at least 8 CPU clock cycles before it changes its level.

During these 8 CPU clock cycles the capture input signals are scanned sequentially. When a timer is modified or incremented during this process, the new timer contents will already be captured for the remaining capture registers within the current scanning sequence. If pin CCxIO is configured as output, the capture function may be triggered by modifying the corresponding port output latch via software, like for testing purposes.

Figure 116 : Capture mode block diagram



15.5 - Compare Modes

The compare modes allow triggering of events (interrupts and / or output signal transitions) with minimum software overhead.

In all compare modes, the 16 bit value stored in compare register CCx (in the following also referred to as 'compare value') is continuously compared with the contents of the allocated timer (T0 / T1 or T7 / T8). If the current timer contents match the compare value, an appropriate output signal, which is based on the selected compare mode, can be generated at the corresponding output pin CCxIO (except for CC24IO...CC27IO) and the associated interrupt request flag CCxIR is set, which can generate an interrupt request (if enabled).

As for capture mode, the compare registers are also processed sequentially during compare mode. When any two compare registers are programmed to the same compare value, their corresponding interrupt request flags will be set to '1' and the selected output signals will be generated within 8 CPU clock cycles after the allocated timer is incremented to the compare value.

Further compare events on the same compare value are disabled until the timer is incremented again or written to by software. After a reset, compare events for register CCx will only become enabled, if the allocated timer has been incremented or written to by software and one of the compare modes described in the following has been selected for this register.

The different compare modes which can be programmed for a given compare register CCx are selected by the mode control field CCMODx in the associated capture / compare mode control register. In the following, each of the compare modes, including the special 'double-register' mode, is discussed in detail.

Table 41 : Summary of compare modes

Compare Modes	Function
Mode 0	Interrupt-only compare mode; several compare interrupts per timer period are possible
Mode 1	Pin toggles on each compare match; several compare events per timer period are possible
Mode 2	Interrupt-only compare mode; only one compare interrupt per timer period is generated
Mode 3	Pin set '1' on match; pin reset '0' on compare time overflow; only one compare event per timer period is generated
Double Register Mode	Two registers operate on one pin; pin toggles on each compare match; several compare events per timer period are possible.

15.5.1 - Compare Mode 0

This is an interrupt-only mode which can be used for software timing purposes. Compare mode 0 is selected for a given compare register CCx by setting bit-field CCMODx of the corresponding mode control register to '100b'.

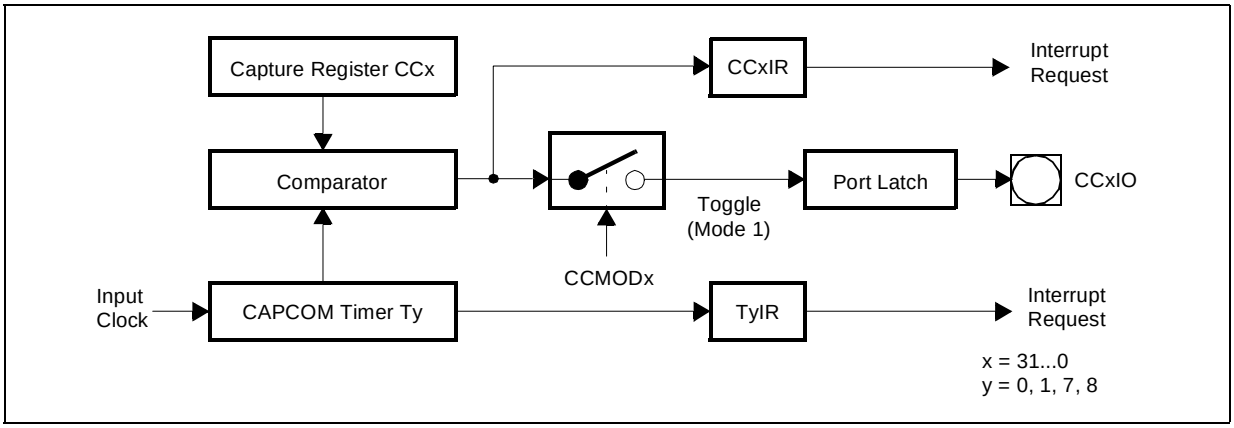
In this mode, the interrupt request flag CCxIR is set each time a match is detected between the content of compare register CCx and the allocated timer.

Several of these compare events are possible within a single timer period, when the compare value in register CCx is updated during the timer period.

The corresponding port pin CCxIO is not affected by compare events in this mode and can be used as general purpose I/O pin.

If compare mode 0 is programmed for one of the registers CC8...CC15 or CC24...CC31, the double-register compare mode becomes enabled for this register if the corresponding bank 1 register is programmed to compare mode 1 (see section "Double- Register Compare Mode").

Figure 117 : Compare mode 0 and 1 block diagram



Note The port latch and pin remain unaffected in compare mode 0.

In the example below, the compare value in register CCx is modified from cv1 to cv2 after compare events #1 and #3, and from cv2 to cv1 after events #2 and #4, etc. This results in periodic interrupt requests from timer Ty, and in interrupt requests from register CCx which occur at the time specified by the user through cv1 and cv2 (see Figure 118).

15.5.2 - Compare Mode 1

Compare mode 1 is selected for register CCx by setting bit-field CCMODx of the corresponding mode control register to '101b'.

When a match between the content of the allocated timer and the compare value in register CCx is detected in this mode, interrupt request flag CCxIR is set to '1', and in addition the corresponding output pin CCxIO (alternate port output function) is toggled. For this purpose, the state of the respective port output latch (not the pin) is read, inverted, and then written back to the output latch.

Compare mode 1 allows several compare events within a single timer period. An overflow of the allocated timer has no effect on the output pin, nor does it disable or enable further compare events.

In order to use the respective port pin as compare signal output pin CCxIO for compare register CCx in compare mode 1, this port pin must be configured as output, and the corresponding direction control bit must be set to '1'. With this configuration, the initial state of the output signal can be programmed or its state can be modified at any time by writing to the port output latch.

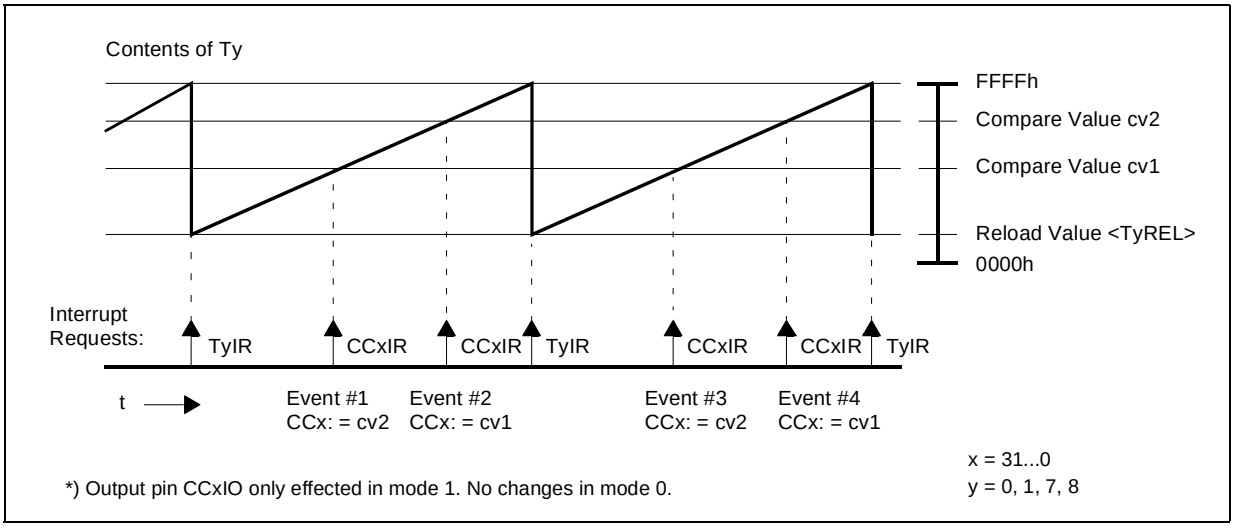
In compare mode 1 the port latch is toggled upon each compare event (see Figure 118).

Note If the port output latch is written to by software at the same time it would be altered by a compare event, the software write will have priority. In this case the hardware-triggered change will not become effective.

If compare mode 1 is programmed for one of the registers CC0...CC7 or CC16...CC23 the double-register compare mode becomes enabled for this register if the corresponding bank 1 register is programmed to compare mode 0 (see section "Double-Register Compare Mode").

Note If the port output latch is written to by software at the same time it would be altered by a compare event, the software write will have priority. In this case the hardware-triggered change will not become effective.
On channels 24...27 compare mode 1 will generate interrupt requests but no output function is provided.

Figure 118 : Timing example for compare modes 0 and 1



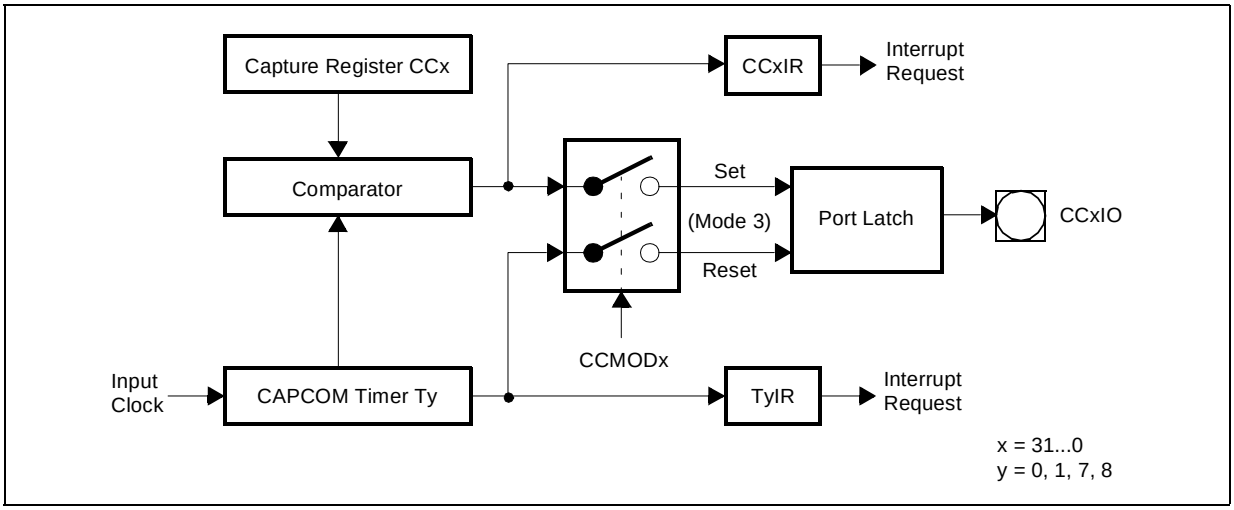
15.5.3 - Compare Mode 2

Compare mode 2 is an interrupt-only mode similar to compare mode 0, but only one interrupt request per timer period will be generated. Compare mode 2 is selected for register CCx by setting bit-field CCMODx of the corresponding mode control register to '110b'.

When a match is detected in compare mode 2 for the first time within a timer period, the interrupt request flag CCxIR is set to '1'. The corresponding Port2 pin is not affected and can be used for general purpose I/O. However, after the first match has been detected in this mode, all further compare events within the same timer period are disabled for compare register CCx until the allocated timer overflows. This means, that after the first match, even when the compare register is reloaded with a value higher than the current timer value, no compare event will occur until the next timer period.

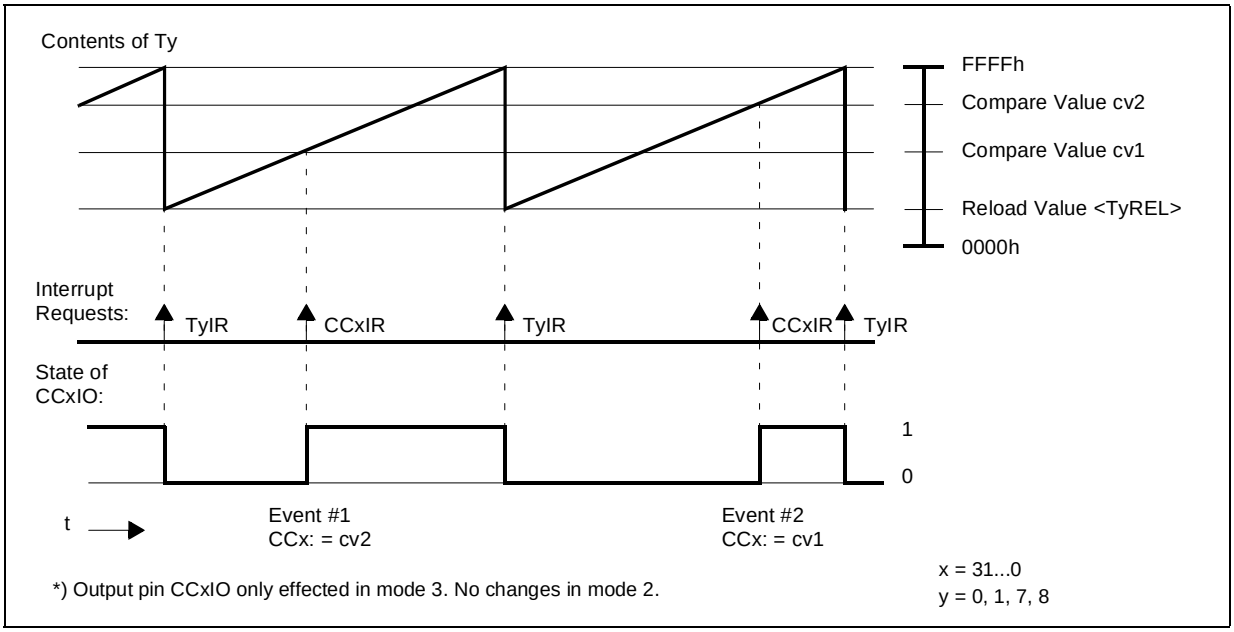
In the example below, the compare value in register CCx is modified from cv1 to cv2 after compare event #1. Compare event #2, however, will not occur until the next period of timer Ty.

Figure 119 : Compare mode 2 and 3 block diagram



Note The port latch and pin remain unaffected in compare mode 2.

Figure 120 : Timing example for compare modes 2 and 3



15.5.4 - Compare Mode 3

Compare mode 3 is selected for register CCx by setting bit-field CCMODx of the corresponding mode control register to '111b'. In compare mode 3 only one compare event will be generated per timer period.

When the first match within the timer period is detected the interrupt request flag CCxIR is set to '1' and also the output pin CCxIO (alternate port function) will be set to '1'. The pin will be reset to '0', when the allocated timer overflows.

If a match was found for register CCx in this mode, all further compare events during the current timer period are disabled for CCx until the corresponding timer overflows. If, after a match was detected, the compare register is reloaded with a new value, this value will not become effective until the next timer period.

In order to use the respective port pin as compare signal output pin CCxIO for compare register CCx in compare mode 3 this port pin must be configured as output and the corresponding direction control bit must be set to '1'. With this configuration, the initial state of the output signal can be programmed or its state can be modified at any time by writing to the port output latch.

In compare mode 3 the port latch is set upon a compare event and cleared upon a timer overflow (see Figure 120).

However, when compare value and reload value for a channel are equal the respective interrupt requests will be generated, only the output signal is not changed (set and clear would coincide in this case).

Note If the port output latch is written to by software at the same time it would be altered by a compare event, the software write will have priority. In this case the hardware-triggered change will not become effective.
On channels 24...27 compare mode 1 will generate interrupt requests but no output function is provided.

15.5.5 - Double Register Compare Mode

In double-register compare mode two compare registers work together to control one output pin. This mode is selected by a special combination of modes for these two registers.

For double-register mode the 16 capture / compare registers of each CAPCOM unit are regarded as two banks of 8 registers each. Registers CC0...CC7 and CC16...CC23 form bank 1 while registers CC8...CC15 and CC24...CC31 form bank 2 (respectively). For double-register mode a bank 1 register and a bank 2 register form a register pair. Both registers of this register pair operate on the pin associated with the bank 1 register (pins CC0IO...CC7IO and CC16IO...CC23IO).

The relationship between the bank 1 and bank 2 register of a pair and the effected output pins for double-register compare mode is listed in the Table 42.

Table 42 : Register pairs for double-register compare mode

CAPCOM1 Unit			CAPCOM2 Unit		
Register Pair		Associated Output Pin	Register Pair		Associated Output Pin
Bank 1	Bank 2		Bank 1	Bank 2	
CC0	CC8	CC0IO	CC16	CC24	CC16IO
CC1	CC9	CC1IO	CC17	CC25	CC17IO
CC2	CC10	CC2IO	CC18	CC26	CC18IO
CC3	CC11	CC3IO	CC19	CC27	CC19IO
CC4	CC12	CC4IO	CC20	CC28	CC20IO
CC5	CC13	CC5IO	CC21	CC29	CC21IO
CC6	CC14	CC6IO	CC22	CC30	CC22IO
CC7	CC15	CC7IO	CC23	CC31	CC23IO

The double-register compare mode can be programmed individually for each register pair. In order to enable double-register mode the respective bank 1 register (see Table 42) must be programmed to compare mode 1 and the corresponding bank 2 register (see Table 42) must be programmed to compare mode 0.

If the respective bank 1 compare register is disabled or programmed for a mode other than mode 1 the corresponding bank 2 register will operate in compare mode 0 (interrupt-only mode).

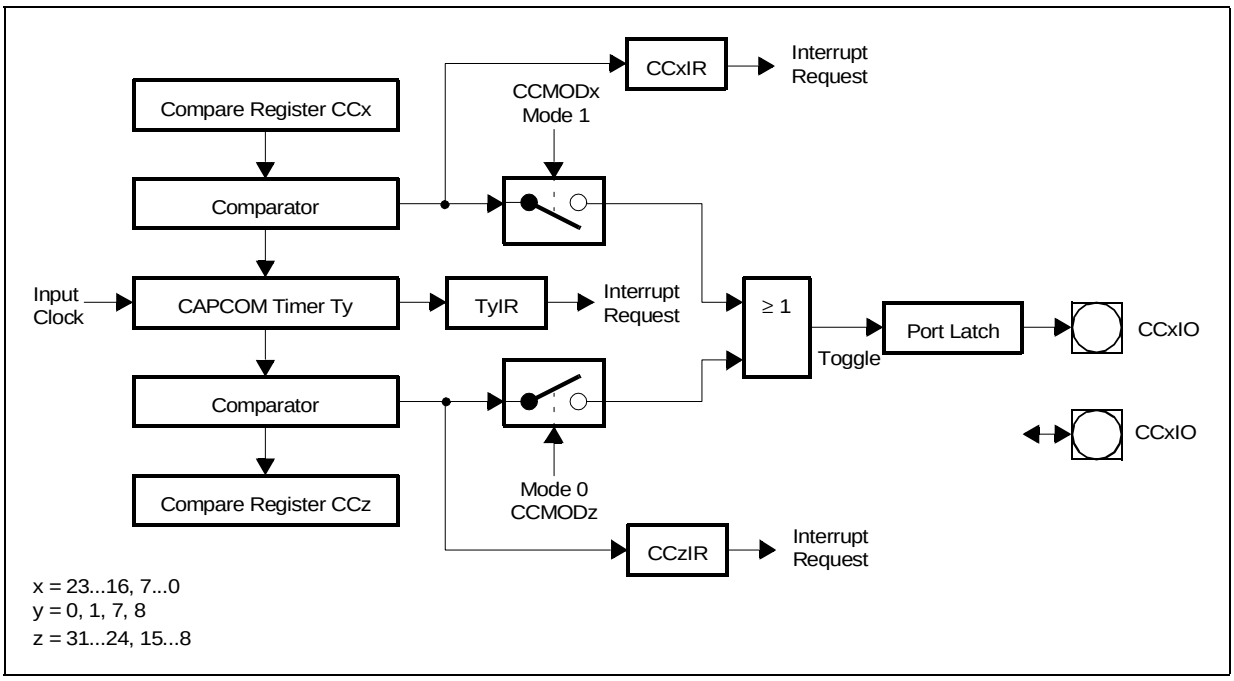
In the following, a bank 2 register (programmed to compare mode 0) will be referred to as CCz while the corresponding bank 1 register (programmed to compare mode 1) will be referred to as CCx.

When a match is detected for one of the two registers in a register pair (CCx or CCz) the associated interrupt request flag (CCxIR or CCzIR) is set to '1' and pin CCxIO corresponding to bank 1 register CCx is toggled. The generated interrupt always corresponds to the register that caused the match.

Note If a match occurs simultaneously for both register CCx and register CCz of the register pair, pin CCxIO will be toggled only once but two separate compare interrupt requests will be generated, one for vector CCxINT and one for vector CCzINT.

In order to use the respective port pin as compare signal output pin CCxIO for compare register CCx in double-register compare mode, this port pin must be configured as output, and the corresponding direction control bit must be set to '1'. With this configuration, the output pin has the same characteristics as in compare mode 1.

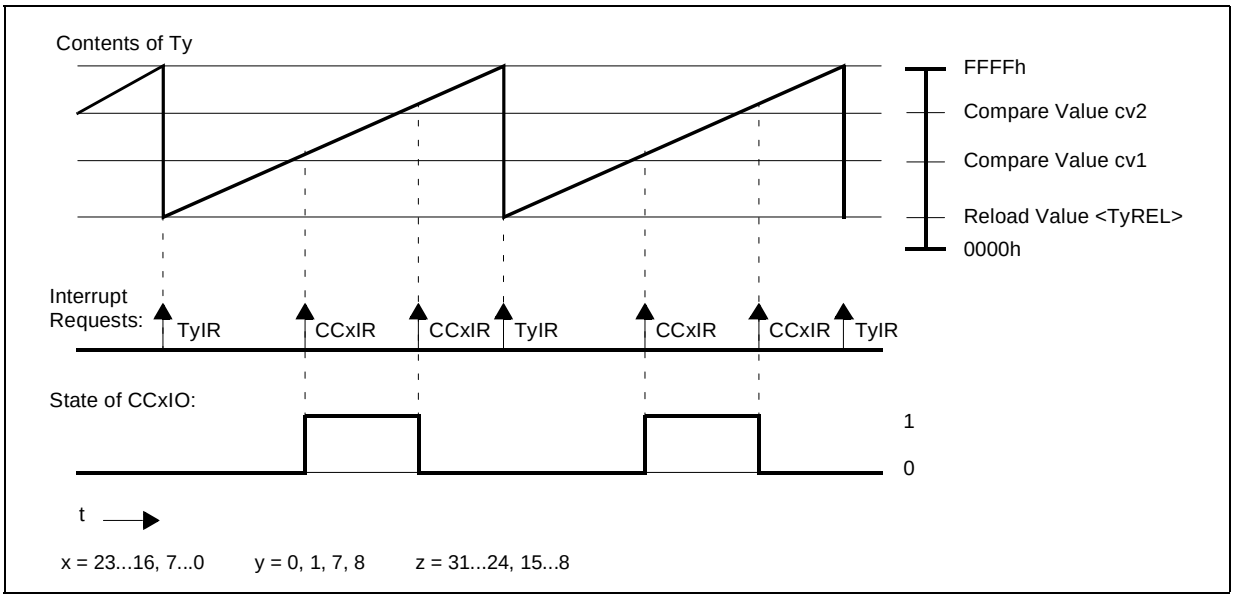
Figure 121 : Double register compare mode block diagram



In this configuration example, the same timer allocation was chosen for both compare registers, but each register may also be individually allocated to one of the two timers of the respective CAPCOM unit. In the timing example for this compare mode (below) the compare values in registers CCx and CCz are not modified.

The pins CCzIO (which are not selected for double-register compare mode) may be used for general purpose I/O.

Figure 122 : Timing example for double register compare mode



15.6 - Capture / Compare Interrupts

Upon a capture or compare event, the interrupt request flag CCxIR for the respective capture / compare register CCx is set to '1'. This flag can be used to generate an interrupt or trigger a PEC service request when enabled by the interrupt enable bit CCxIE.

Capture interrupts can be regarded as external interrupt requests with the additional feature of recording the time at which the triggering event occurred (see also section "External Interrupts").

Note Each of the 32 capture / compare registers (CC0...CC31) has its own bit-addressable interrupt control register (CC0IC...CC31IC) and its own interrupt vector (CC0INT...CC31INT). These registers are organized the same way as all other interrupt control registers. The figure below shows the basic register layout, and the table lists the associated addresses.

CCxIC (see Table 43) SFR Reset Value: - - 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	**CCxIR**	CCxIE	ILVL				GLVL	
								RW	RW	RW				RW	

Note Refer to"Interrupt control registers" chapter for more details of the control fields.

Table 43 : CAPCOM unit interrupt control register addresses

CAPCOM1 Unit			CAPCOM2 Unit		
Register	**Address**	**Register Space**	**Register**	**Address**	**Register Space**
CC0IC	FF78h / BCh	SFR	CC16IC	F160h / B0h	ESFR
CC1IC	FF7Ah / BDh	SFR	CC17IC	F162h / B1h	ESFR
CC2IC	FF7Ch / BEh	SFR	CC18IC	F164h / B2h	ESFR
CC3IC	FF7Eh / BFh	SFR	CC19IC	F166h / B3h	ESFR
CC4IC	FF80h / C0h	SFR	CC20IC	F168h / B4h	ESFR
CC5IC	FF82h / C1h	SFR	CC21IC	F16Ah / B5h	ESFR
CC6IC	FF84h / C2h	SFR	CC22IC	F16Ch / B6h	ESFR
CC7IC	FF86h / C3h	SFR	CC23IC	F16Eh / B7h	ESFR
CC8IC	FF88h / C4h	SFR	CC24IC	F170h / B8h	ESFR
CC9IC	FF8Ah / C5h	SFR	CC25IC	F172h / B9h	ESFR
CC10IC	FF8Ch / C6h	SFR	CC26IC	F174h / BAh	ESFR
CC11IC	FF8Eh / C7h	SFR	CC27IC	F176h / BBh	ESFR
CC12IC	FF90h / C8h	SFR	CC28IC	F178h / BCh	ESFR
CC13IC	FF92h / C9h	SFR	CC29IC	F184h / C2h	ESFR
CC14IC	FF94h / CAh	SFR	CC30IC	F18Ch / C6h	ESFR
CC15IC	FF96h / CBh	SFR	CC31IC	F194h / CAh	ESFR

16 - PULSE WIDTH MODULATION MODULE

The Pulse Width Modulation (PWM) Module of the ST10F280 generates up to 4 independent PWM signals. The minimum PWM signal frequency depends on the width (16 bits) and the resolution (CLK/1 or CLK/64) of the PWM timers. The maximum PWM signal frequency assumes that the PWM output signal changes with every cycle of the respective timer. In a real application, the maximum PWM frequency will depend on the required resolution of the PWM output signal.

The pulse width modulation module has 4 independent PWM channels. Each channel has a 16-bit up/down counter PTx, a 16-bit period register PPx with a shadow latch, a 16-bit pulse width register PWx with a shadow latch, two comparators, and the necessary control logic.

The operation of all four channels is controlled by two common control registers, PWMCON0 and PWMCON1, and the interrupt control and status is handled by one interrupt control register PWMIC, which is also common for all channels (see Figure 125).

New PWM Module : XPWM

The new Pulse Width Modulation (XPWM) Module of the ST10F280 is mapped on the XBUS interface (Address range 00'EC00h-00'ECFFh) and allows the generation of up to 4 independent PWM signals.The XPWM is enabled by setting XPEN bit 2 of the SYSCON register and bit 4 of the new XPERCON register.

The Pulse Width Modulation Module consists of 4 independent PWM channels. Each channel has a 16-bit up/down counter XPTx, a 16-bit period register XPPx with a shadow latch, a 16-bit pulse width register XPWx with a shadow latch, two comparators, and the necessary control logic. The operation of all four channels is controlled by two common control registers, XPWMCON0 and XPWMCON1, and the interrupt control and status is handled by one interrupt control register XP2IC, which is also common for all channels (see Figure 125).

Figure 123 : SFRs and Port Pins Associated with the XPWM Module

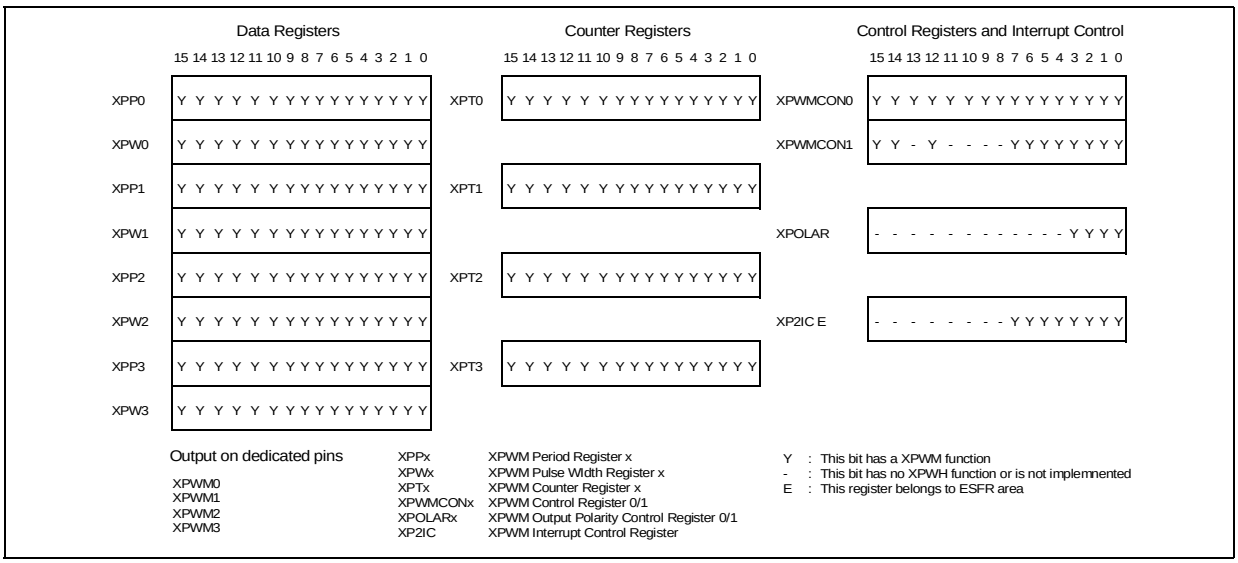


Figure 124 : SFRs and port pins associated with the PWM module

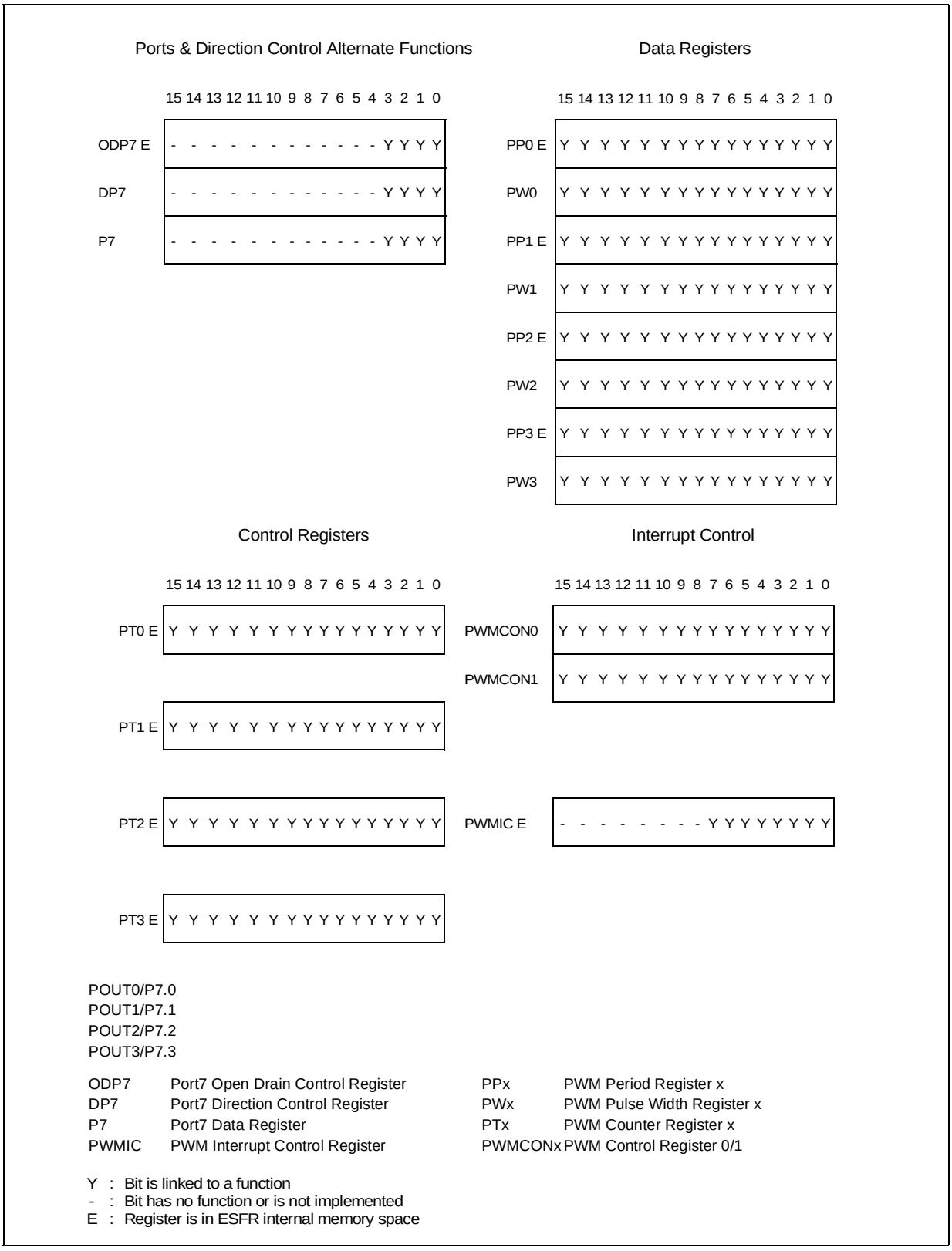
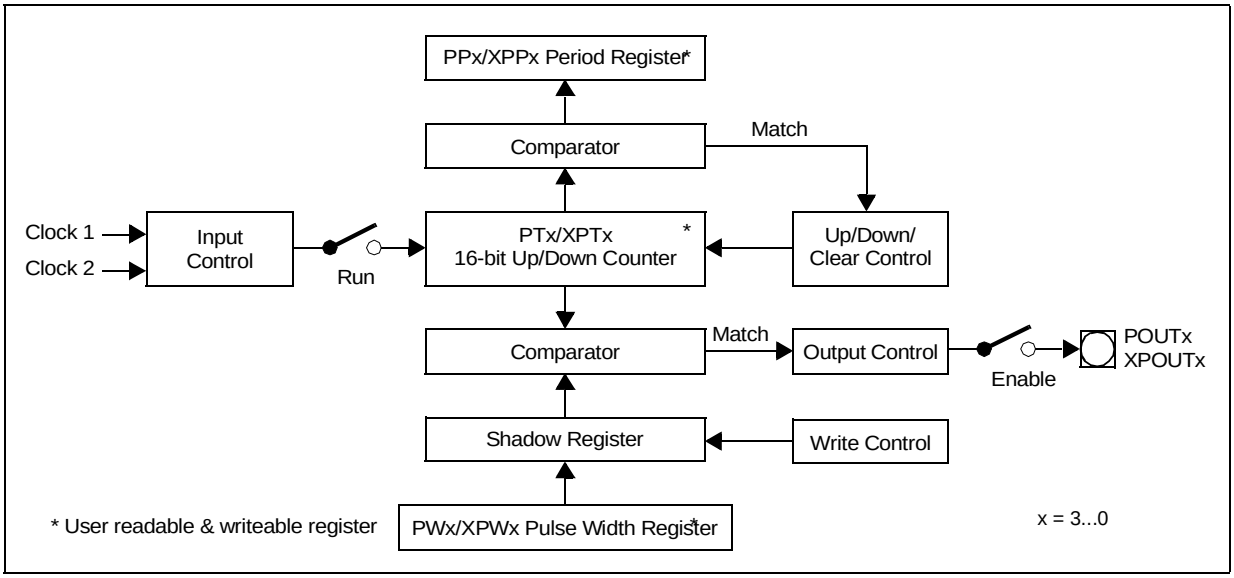


Figure 125 : PWM channel block diagram



16.1 - Operating Modes

The PWM and XPWM modules provide four different operating modes:

– **Mode 0 standard PWM** generation (edge aligned PWM) available on 4 channels

– **Mode 1 Symmetrical PWM** generation (center aligned PWM) available on all four channels

– **Burst mode** combines channels 0 and 1

– **Single shot mode** available on channels 2 and 3

Note The output signals of the PWM module are XORed with the outputs of the respective port output latches. After reset these latches are cleared, so the PWM signals are directly driven to the port pins. By setting the respective port output latch to '1' the PWM signal may be inverted (XORed with '1') before being driven to the port pin. The descriptions below refer to the standard case after reset, which is direct drive.
The output signal of the XPWM module are XORed with the outputs of the XPOLAR(3...0) register before being driven to the output pins.

16.1.1 - Mode 0: Standard PWM Generation (Edge Aligned PWM)

Mode 0 is selected by clearing the respective bit PMx in register PWMCON1/XPWMCON1 to '0'. In this mode the timer PTx, XPTx of the respective PWM channel is always counting up until it reaches the value in the associated period shadow register. Upon 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 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 low level when the respective timer is reset to 0000h, that means below the pulse width shadow register. The period of the resulting PWM signal is determined by the value of the respective PPx/XPPx shadow register plus 1, counted in units of the timer resolution.

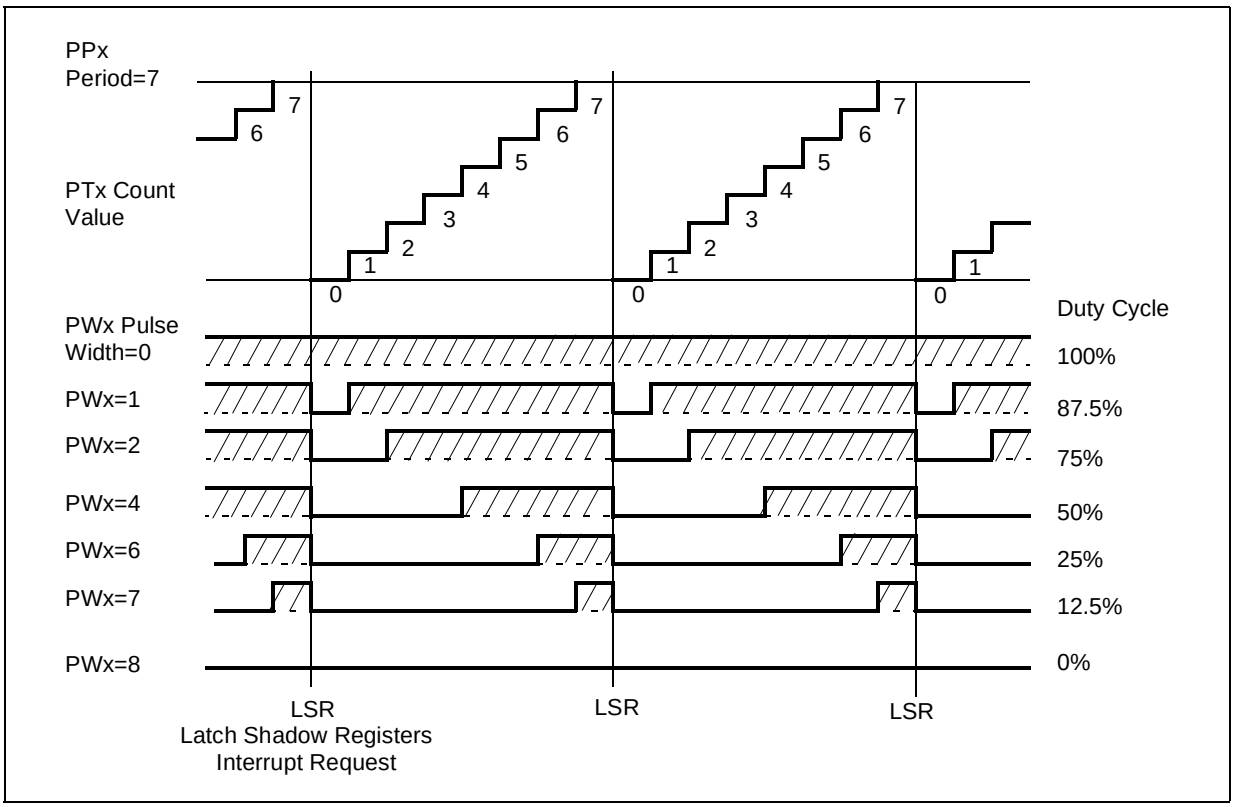
$$PWM\_Period_{Mode0} = [PPx] + 1$$

The duty cycle of the PWM output signal is controlled by the value in the respective pulse width shadow register. This mechanism allows the selection of duty cycles from 0% to 100% including the boundaries.

For a value of 0000h the output will remain at a high level, representing a duty cycle of 100%. For a value higher than the value in the period register the output will remain at a low level, which corresponds to a duty cycle of 0%.

The Figure 126 illustrates the operation and output waveforms of a PWM channel in mode 0 for different values in the pulse width register. This mode is referred to as Edge Aligned PWM, because the value in the pulse width shadow register only effects the positive edge of the output signal. The negative edge is always fixed and related to the clearing of the timer.

Figure 126 : Operation and output waveform in mode 0



16.1.2 - Mode 1: Symmetrical PWM Generation (Center Aligned PWM)

Mode 1 is selected by setting the respective bit PMx in register PWMCON1/XPWMCON1 to '1'. In this mode the timer PTx/XPTx of the respective PWM channel is counting up until it reaches the value in the associated period shadow register. Upon the next count pulse the count direction is reversed and the timer starts counting down now with subsequent count pulses until it reaches the value 0000h. Upon the next count pulse the count direction is reversed again and the count cycle is repeated with the following count pulses.

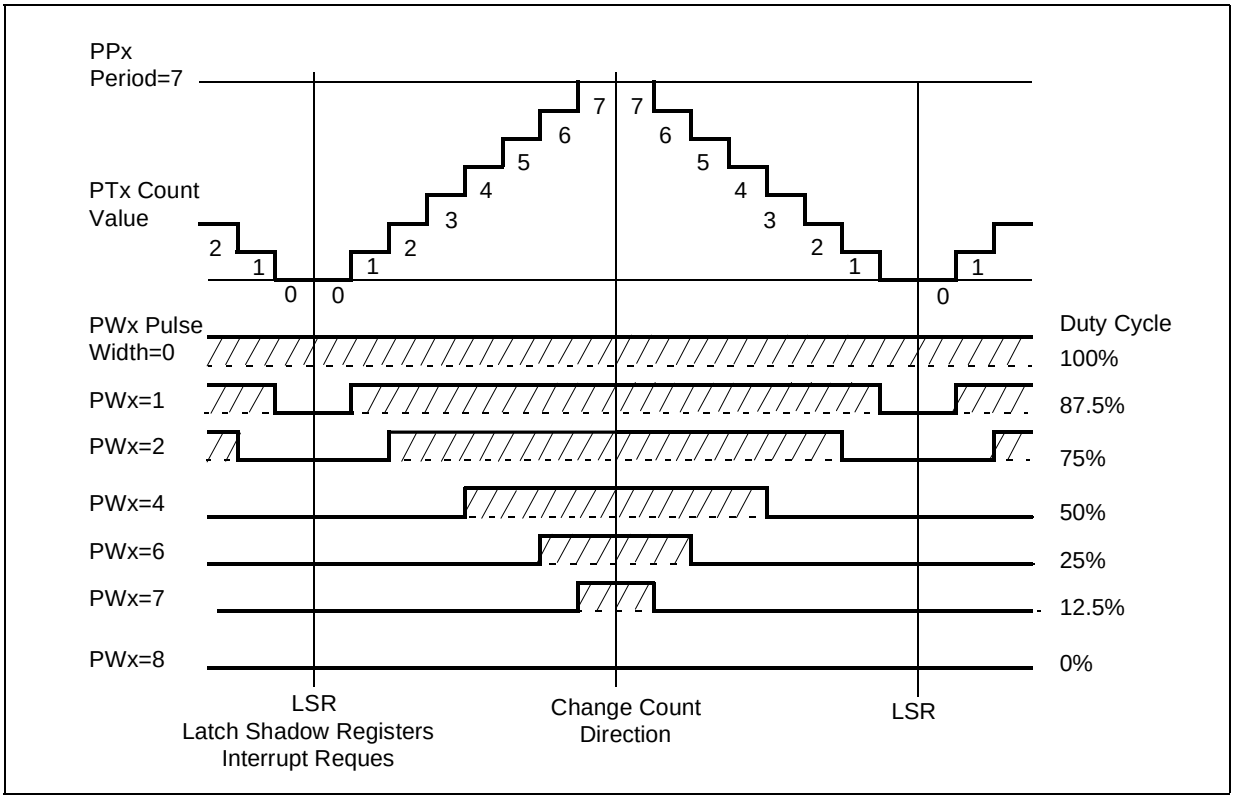
The PWM output signal is switched to a high level when the timer contents are equal to or greater than the contents of the pulse width shadow register while the timer is counting up. The signal is switched back to a low level when the respective timer has counted down to a value below the contents of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

Note that in mode 1 the period of the PWM signal is twice the period of the timer:

$$PWM\_Period_{Mode1} = 2 \times ([PPx] + 1)$$

The Figure 127 illustrates the operation and output waveforms of a PWM channel in mode 1 for different values in the pulse width register. This mode is referred to as Center Aligned PWM, because the value in the pulse width shadow register effects both edges of the output signal symmetrically.

Figure 127 : Operation and output waveform in mode 1



16.1.3 - Burst Mode

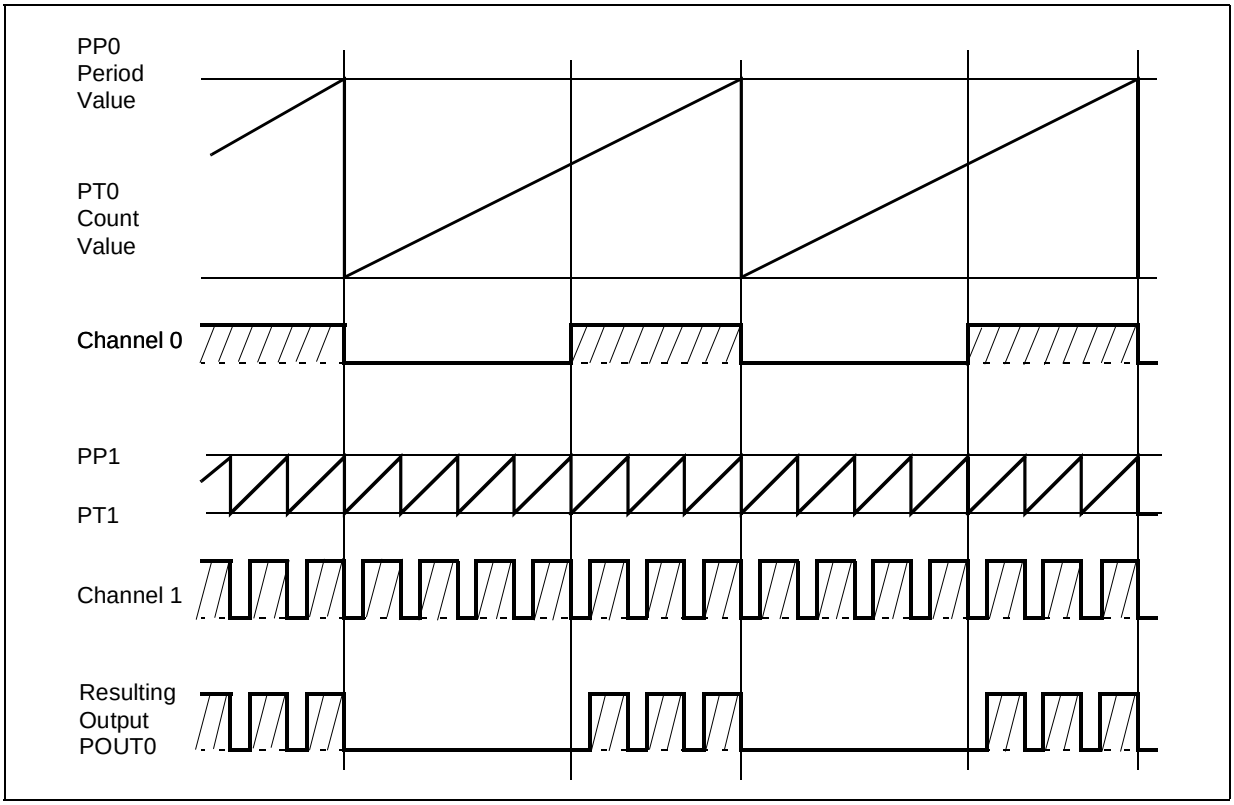
Burst mode is selected by setting bit PB01 in register PWMCON1/XPWMCON1 to '1'. This mode combines the signals from PWM channels 0 and 1 onto the port pin of channel 0.

The output of channel 0 is replaced with the logical AND of channels 0 and 1. The output of channel 1 can still be used at its associated output pin (if enabled).

Each of the two channels can either operate in mode 0 or 1.

Note It is guaranteed by design, that no spurious spikes will occur at the output pin of channel 0 in this mode. The output of the AND gate will be transferred to the output pin synchronously to internal clocks.
The XORing of the PWM/XPWM signal and the port output latch (or the XPOLAR register latch) value is done after the ANDing of channel 0 and 1 (see Figure 130 and Figure 131).

Figure 128 : Operation and output waveform in burst mode



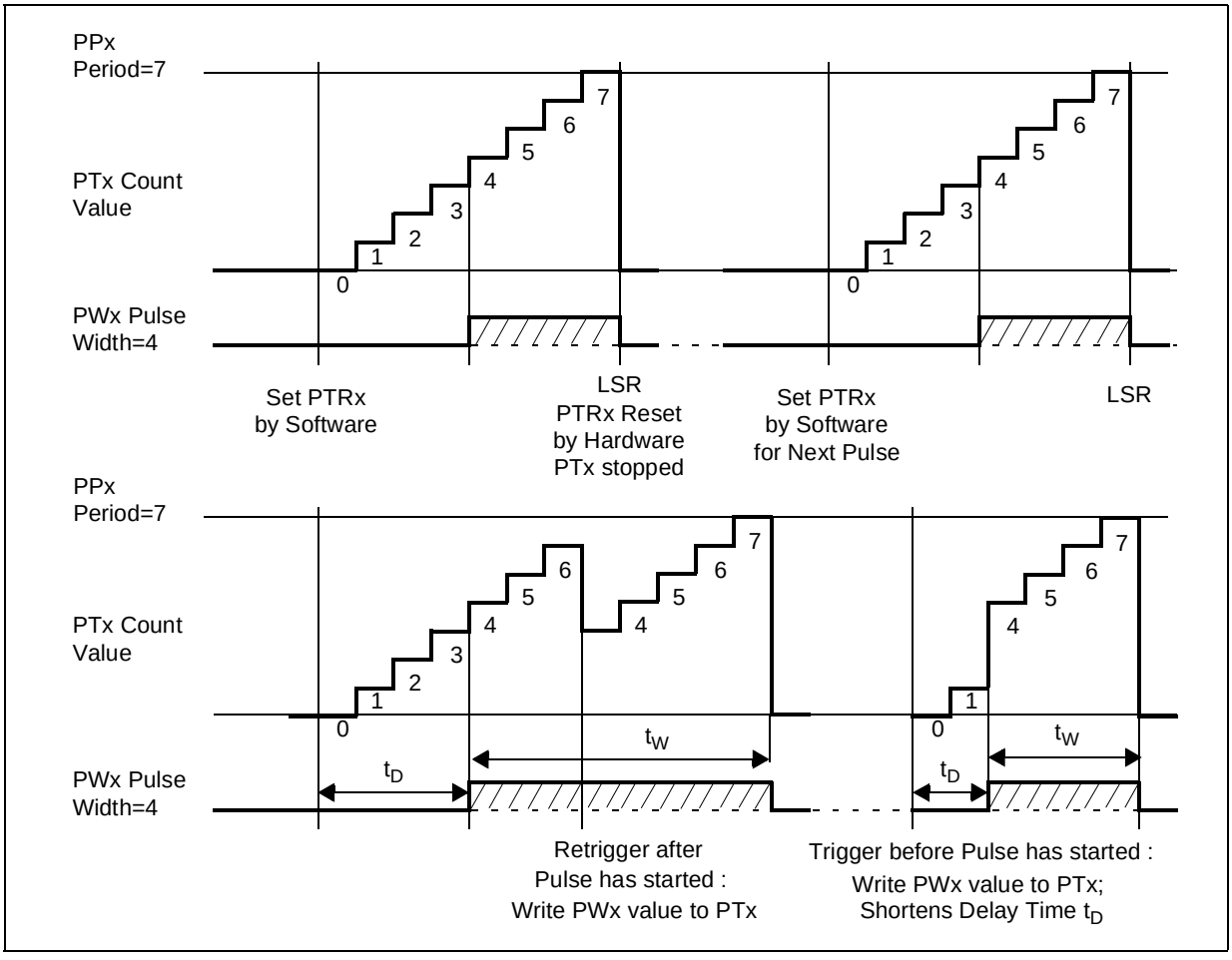
16.1.4 - Single Shot Mode

Single shot mode is selected by setting the respective bit PSx in register PWMCON1/XPWMCON1 to '1'. This mode is available for PWM channels 2 and 3.

In this mode the timer PTx/XPTx of the respective PWM channel is started via software and is counting up until it reaches the value in the associated period shadow register. Upon the next count pulse the timer is cleared to 0000h and stopped via hardware, (the respective PTRx bit is cleared). The PWM output signal is switched to 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 low level when the respective timer is cleared, because it is below the pulse width shadow register.

Thus starting a PWM timer in single shot mode produces one single pulse on the respective 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 PTRx (see Figure 129).

Figure 129 : Operation and output waveform in single shot mode



After starting the timer (with PTRx = '1') the output pulse may be modified via software. Writing to timer PTx/XPTx changes the positive and/or negative edge of the output signal, depending on whether the pulse has already started (the output is high) or not (the output is still low). This (multiple) re-triggering is always possible while the timer is running, after the pulse has started and before the timer is stopped.

Loading counter PTx/XPTx directly with the value in the respective PPx/XPPx shadow register will abort the current PWM pulse upon the next clock pulse (counter is cleared and stopped by hardware).

By setting the period (PPx/XPPx), the timer start value (PTx/XPTx) and the pulse width value (PWx/XPWx) appropriately, the pulse width (tw) and the optional pulse delay (td) may be varied in a wide range (see Figure 129).

16.2 - PWM/XPWM Module Registers

The PWM/XPWM module is controlled via two sets of registers. The waveforms are selected by the channel specific registers PTx/XPTx (timer), PPx/XPPx (period) and PWx/XPWx (pulse width). Respectively three common registers (PWMCON0/XPWMCON0, PWMCON1/XPWMCON1 and PWMIC/XP2IC) control the operating modes and the general functions of the PWM/XPWM modules as well as the interrupt behavior.

Up/down Counters PTx

Each counter PTx/XPTx of a PWM channel is clocked either directly by the CPU clock or by the CPU clock divided by 64. Bit PTIx in register PWMCON0/XPWMCON0 selects the respective clock source. A PWM counter counts up or down (controlled by hardware), while its respective run control bit PTRx is set. A timer is started (PTRx = '1') via software and is stopped (PTRx = '0') either via hardware or software, depending on its operating mode. Control bit PTRx enables or disables the clock input of counter PTx/XPTx rather than controlling the PWM output signal.

Table 44 summarizes the PWM frequencies that result from various combinations of operating mode, counter resolution (input clock) and pulse width resolution.

Period Registers PPx/XPPx

The 16-bit period register PPx/XPPx of a PWM channel determines the period of a PWM cycle and the frequency of the PWM signal. This register is buffered with a shadow register.

The shadow register is loaded from the respective PPx/XPPx register at the beginning of every new PWM cycle, or upon a write access to PPx/XPPx, while the timer is stopped. The CPU accesses the PPx/XPPx register while the hardware compares the contents of the shadow register with the contents of the associated counter PTx/XPTx.

When a match is found between counter and PPx/XPPx shadow 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. For the register locations refer to the Table 45.

Table 44 : PWM Frequencies

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	$f_{cpu}/2x64x2^{8}$	$f_{cpu}/2x64x2^{10}$	$f_{cpu}/2x64x2^{12}$	$f_{cpu}/2x64x2^{14}$	$f_{cpu}/2x64x2^{16}$

Pulse Width Registers PWx/XPWx

This 16-bit register holds the actual PWM pulse width value which corresponds to the duty cycle of the PWM signal. This register is buffered with a shadow register.

The CPU accesses the PWx/XPWx register while the hardware compares the contents of the shadow register with the contents of the associated counter PTx/XPTx. The shadow register is loaded from the respective PWx/XPWx register at the beginning of every new PWM cycle, or upon a write access to PWx/XPWx, 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 comparators may be described by the boolean formula:

PWM output signal = [PTx] $\geq$ [PWx shadow latch].

This type of comparison allows a flexible control of the PWM signal. For the register locations refer to the Table 45.

Table 45 : PWM module channel specific register addresses

Register	Address	Reg. Space	Register	Address	Reg. Space
PW0	FE30h / 18h	SFR	PT0	F030h / 18h	ESFR
PW1	FE32h / 19h	SFR	PT1	F032h / 19h	ESFR
PW2	FE34h / 1Ah	SFR	PT2	F034h / 1Ah	ESFR
PW3	FE36h / 1Bh	SFR	PT3	F036h / 1Bh	ESFR
These registers are not bit-addressable.			PP0	F038h / 1Ch	ESFR
			PP1	F03Ah / 1Dh	ESFR
			PP2	F03Ch / 1Eh	ESFR
			PP3	F03Eh / 1Fh	ESFR

Table 46 : XPWM Module Channel Specific Register Addresses

Register	Address	Register	Address
XPW0	EC30h	XPT0	EC10h
XPW1	EC32h	XPT1	EC12h
XPW2	EC34h	XPT2	EC14h
XPW3	EC36h	XPT3	EC16h
These registers are not bit-addressable.		XPP0	EC20h
		XPP1	EC22h
		XPP2	EC24h
		XPP3	EC26h

PWM/XPWM Control Register PWMCON0/XPWMCON0

Register PWMCON0/XPWMCON0 controls the function of the timers of the four PWM/XPWM channels and the channel specific interrupts respectively. Having the control bit organized in functional groups allows to start or to stop all the 4 PWM timers simultaneously with one bit-field instruction.

PWMCON0 (FF30h / 98h) SFR Reset Value: 0000h
XPWMCON0 (EC00h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PIR3	**PIR2**	**PIR1**	**PIR0**	PIE3	PIE2	PIE1	PIE0	PTI3	PTI2	PTI1	PTI0	**PTR3**	**PTR2**	PTR1	PTR0
RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
PTRx	**PWM/XPWM Timer x Run Control bit** '0': Timer PTx is disconnected from its input clock '1': Timer PTx is running
PTIx	**PWM/XPWM Timer x Input Clock Selection** '0': Timer PTx clocked with CLK_{CPU} '1': Timer PTx clocked with CLK_{CPU} / 64
PIEx	**PWM/XPWM Channel x Interrupt Enable Flag** '0': Interrupt from channel x disabled '1': Interrupt from channel x enabled
PIRx	**PWM/XPWM Channel x Interrupt Request Flag** '0': No interrupt request from channel x '1': Channel x interrupt pending (must be reset via software)

PWM/XPWM Control Register PWMCON1/XPWMCON1

Registers PWMCON1/XPWMCON1 control the operating modes and the outputs of the four PWM/XPWM channels respectively. The basic operating mode for each channel (standard=edge aligned, or symmetrical=center aligned PWM mode) is selected by the mode bit PMx. Burst mode (channels 0 and 1) and single shot mode (channel 2 or 3) are selected by separate control bit. The output signal of each PWM/XPWM channel is individually enabled by bit PENx. If the output is not enabled the respective pin can be used for general purpose I/O and the PWM channel can only be used to generate an interrupt request.

PWMCON1 (FF32h / 99h) SFR Reset Value: 0000h
XPWMCON1 (EC02h) XBUS Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PS3	PS2	-	PB01	-	-	-	-	PM3	PM2	PM1	PM0	PEN3	PEN2	PEN1	PEN0
RW	RW		RW					RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
PENx	**PWM/XPWM Channel x Output Enable bit** '0': Channel x output signal disabled, generate interrupt only '1': Channel x output signal enabled
PMx	**PWM/XPWM Channel x Mode Control bit** '0': Channel x operates in mode 0, i.e. edge aligned PWM '1': Channel x operates in mode 1, i.e. center aligned PWM
PB01	**PWM/XPWM Channel 0/1 Burst Mode Control bit** '0': Channels 0 and 1 work independently in respective standard mode '1': Outputs of channels 0 and 1 are ANDed to POUT0 in burst mode
PSx	**PWM/XPWM Channel x Single Shot Mode Control bit** '0': Channel x works in respective standard mode '1': Channel x operates in single shot mode

16.3 - Interrupt Request Generation

Each of the four channels of the PWM/XPWM module can generate an individual interrupt request. Each of these "channel interrupts" can activate the common "module interrupt", which actually interrupts the CPU. This common module interrupt is controlled by the PWM Module Interrupt Control register PWMIC/ XP2IC. The interrupt service routine can determine the active channel interrupt(s) from the channel specific interrupt request flags PIRx in register PWMCON0/XPWMCON0.

The interrupt request flag PIRx of a channel is set at the beginning of a new PWM cycle, when loading the shadow registers. This indicates that registers PPx/XPPx and PTx/XPTx are now ready to receive a new value. If a channel interrupt is enabled via its respective PIEx bit, also the common interrupt request flag PWMIR/XP2IR in register PWMIC/XP2IC is set, provided that it is enabled via the common interrupt enable bit PWMIE/XP2IE.

Note The channel interrupt request flags (PIRx in register PWMCON0/XPWMCON0) are not automatically cleared by hardware upon entry into the interrupt service routine, so they must be cleared via software. The module interrupt request flag PWMIR/XP2IR is cleared by hardware upon entry into the 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.

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	

XP2IC (F196h / CBh) ESFR Reset Value:-00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	**XP2IR**	XP2IE	ILVL				GLVL	
								RW	RW	RW				RW	

Note Refer to the general Interrupt Control Register description for an explanation of the control fields.

16.4 - PWM/XPWM Output Signals

The output signals of the four PWM channels (POUT3...POUT0) are alternate output functions on Port7 (P7.3...P7.0), while XPWM3...XPWM0 are dedicated pins. The output signal of each PWM channel is individually enabled by control bit PENx in register PWMCON1/XPWMCON1.

The PWM signals are XORed with the respective port latch outputs before being driven to the port pins, while XPWM3...XPWM0 signals are XORed with the output of the XPOLAR register before being driven to the output pins (see Figure 130 and Figure 131). The XPOLAR register is dedicated to XPWM module and described below.

16.4.1 - XPOLAR register (polarity of the XPWM channel)

XPOLAR (EC04h) XBUS Reset Value: 0000h

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

Bit	Function
XPOLAR.x	**XPOLAR Channel x polarity Bit**
	0 Polarity of Channel x is normal
	1 Polarity of Channel x is inverted

This allows driving the PWM signal directly to the port pin (XPWMx pin) or drive the inverted PWM signal (see Figure 130 and Figure 131).

Note Using the open-drain mode on Port 7 allows the combination of two or more PWM outputs through a AND-Wired configuration, with an external pull-up device. This provides sort of a burst mode for any PWM channel. This feature is not implemented in the XPWM module.

Software Control of the PWM/XPWM Outputs

In an application the PWM output signals are generally controlled by the PWM module. However, it may be necessary to influence the level of the PWM output pins via software either to initialize the system or to react on some extraordinary condition, like a system fault or an emergency.

Clearing the timer run bit PTRx stops the associated counter and leaves the respective output at its current level.

The individual PWM channel outputs are controlled by comparators according to the formula:

PWM output signal = [PTx] $\geq$ [PWx shadow latch].

So whenever software changes registers PTx/XPTx, the respective output will reflect the condition after the change. Loading timer PTx/XPTx with a value greater than or equal to the value in PWx/XPWx immediately sets the respective output, a PTx/XPTx value below the PWx/XPWx value clears the respective output.

By clearing or setting the respective Port7/XPOLAR output latch the PWM channel signal is driven directly or inverted to the port pin.

Clearing the enable bit PENx disconnects the PWM channel and switches the respective port pin to the value in the port output latch.

Note To prevent further PWM pulses from occurring after such a software intervention the respective counter must be stopped first.

Figure 130 : PWM output signal generation

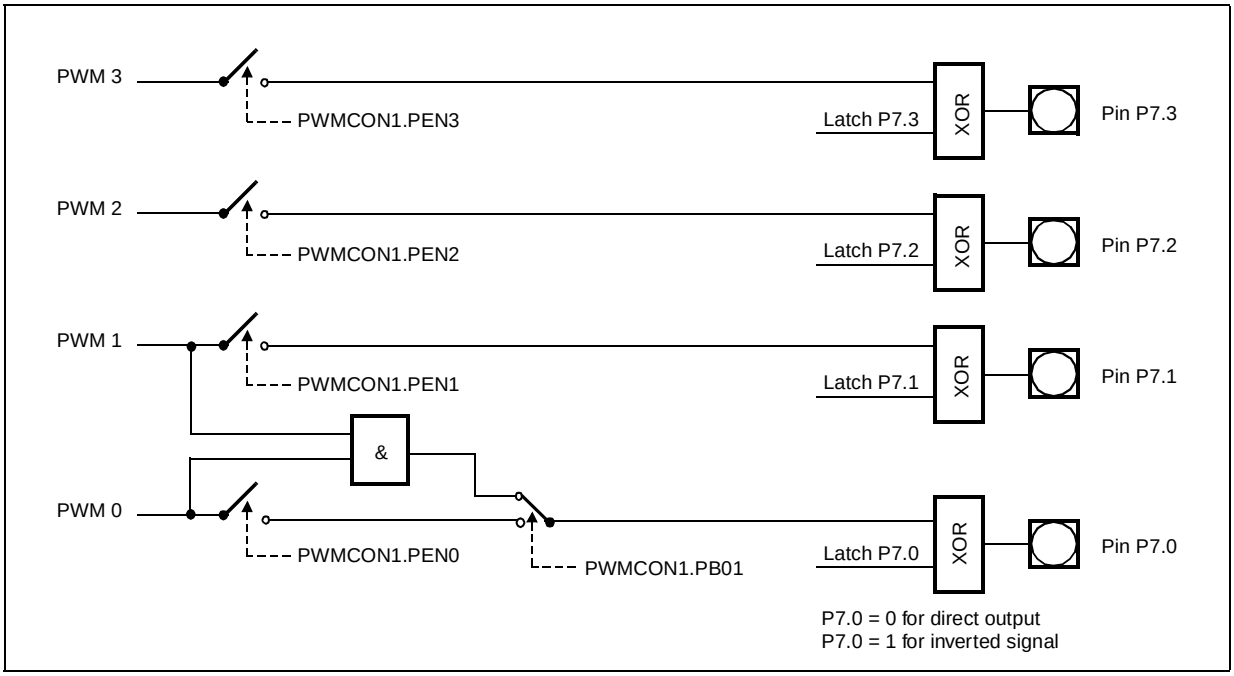
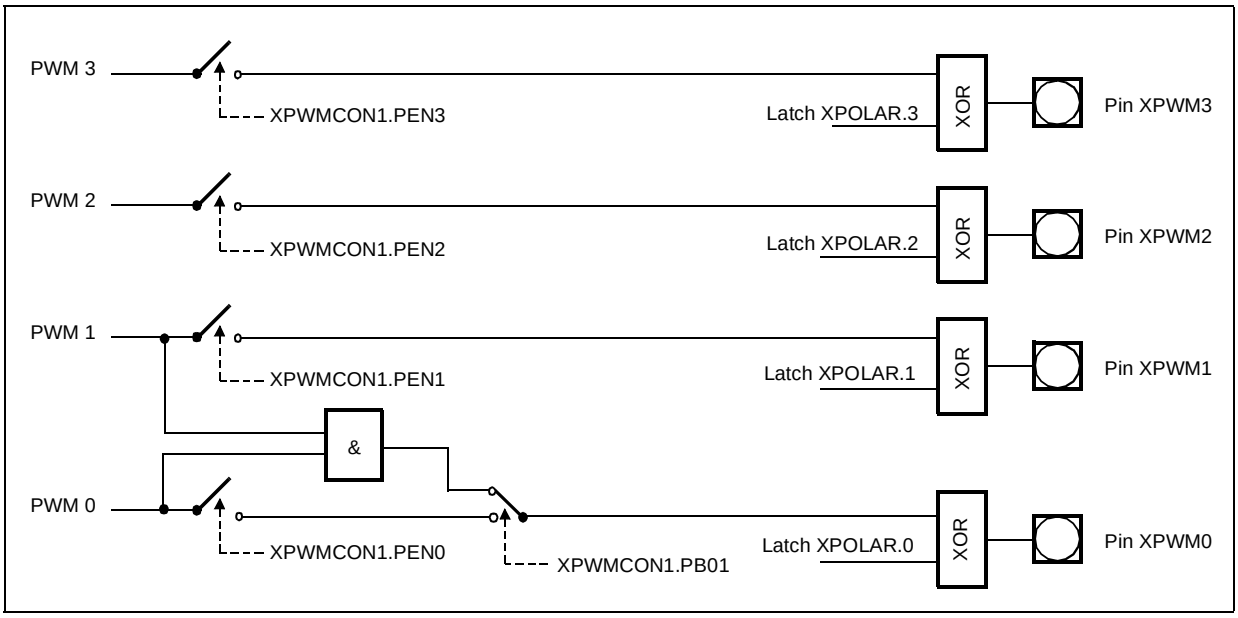


Figure 131 : XPWM Output Signal Generation



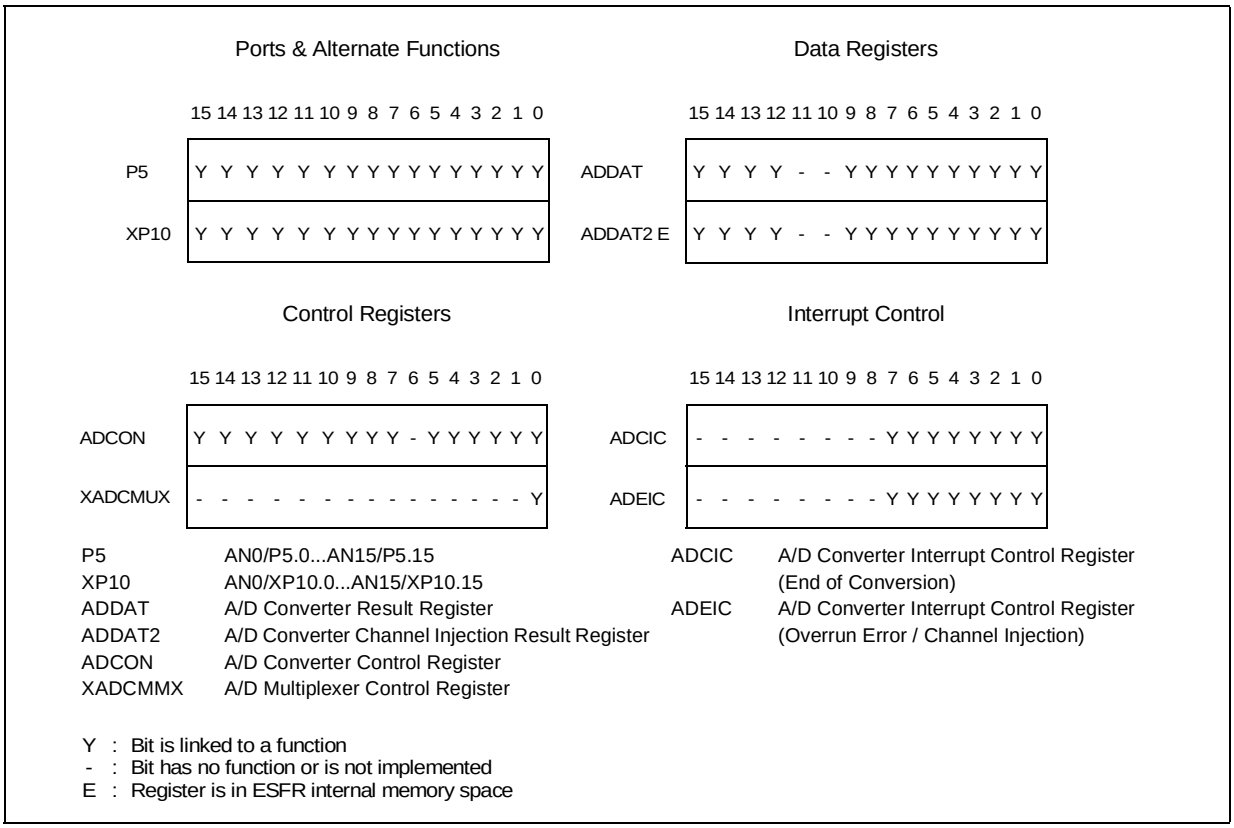
17 - ANALOG / DIGITAL CONVERTER

The ST10F280 provides an Analog / Digital Converter with 10-bit resolution and a sample & hold circuit on-chip. A multiplexer selects between up to 2 x 16 analog input channels (alternate functions of Port5 and of XPort10) either via software (fixed channel modes) or automatically (auto scan modes). The ADC supports the following conversion modes:

– **Fixed channel single conversion**
  produces just one result from the selected channel
– **Fixed channel continuous conversion**
  repeatedly converts the selected channel
– **Auto scan single conversion**
  produces one result from each of a selected group of channels
– **Auto scan continuous conversion**
  repeatedly converts the selected group of channels
– **Wait for ADDAT read mode**
  start a conversion automatically when the previous result was read
– **Channel injection mode**
  insert the conversion of a specific channel into a group conversion (auto scan)

A set of SFRs and port pins provide access to control functions and results of the ADC.

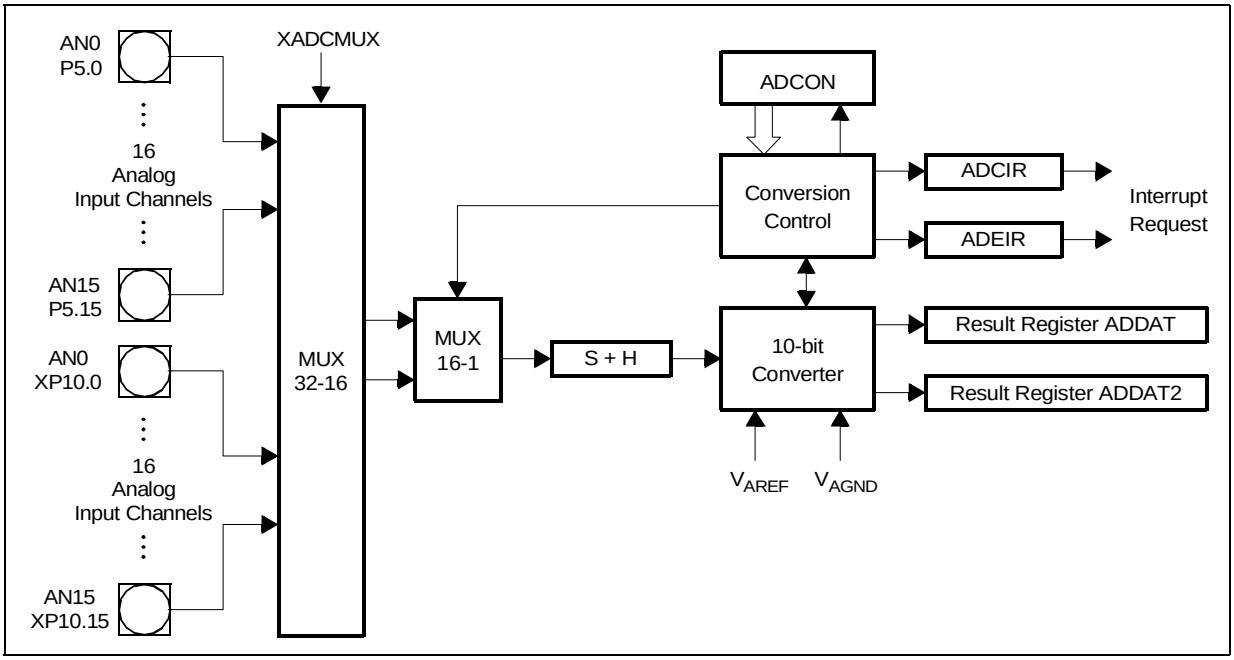
Figure 132 : SFRs and port pins associated with the A/D converter



The external analog reference voltages V_{AREF} and V_{AGND} are fixed. The separate supply for the ADC reduces the interference with other digital signals.

The sample time as well as the conversion time is programmable, so the ADC can be adjusted to the internal resistances of the analog sources and/or the analog reference voltage supply.

Figure 133 : Analog / digital converter block diagram



17.1 - Mode Selection and Operation

The analog input channels AN0...AN15 are alternate functions of Port5 which is a 16-bit input-only port. The Port5 lines may either be used as analog or digital inputs. No special action is required to configure the Port5 lines as analog inputs.

The functions of the A/D converter are controlled by the bit-addressable A/D Converter Control Register ADCON.

Its bit-fields specify the analog channel to be acted upon, the conversion mode, and also reflect the status of the converter.

ADCON (FFA0h / D0h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADCTC		ADSTC		ADCRQ	ADCIN	ADWR	ADBSY	ADST	-	ADM		ADCH			
RW		RW		RW	RW	RW	R	RW		RW		RW			

Bit	Function
ADCH	**ADC Analog Channel Input Selection**
ADM	**ADC Mode Selection** 0 0: Fixed Channel Single Conversion 0 1: Fixed Channel Continuous Conversion 1 0: Auto Scan Single Conversion 1 1: Auto Scan Continuous Conversion
ADST	**ADC Start bit**
ADBSY	**ADC Busy Flag** ADBSY = 1: a conversion is active
ADWR	**ADC Wait for Read Control**
ADCIN	**ADC Channel Injection Enable**
ADCRQ	**ADC Channel Injection Request Flag**
ADSTC	**ADC Sample Time Control** *)
ADCTC	**ADC Conversion Time Control** *)

Note *) ADSTC and ADCTC control the conversion timing. Refer to Section 17.2.

Bit-field ADCH specifies the analog input channel which is to be converted (first channel of a conversion sequence in auto scan modes). Bit-field ADM selects the operating mode of the A/D converter. A conversion (or a sequence) is then started by setting bit ADST.

Clearing ADST stops the A/D converter after a certain operation which depends on the selected operating mode.

The busy flag (read-only) ADBSY is set, as long as a conversion is in progress.

The result of a conversion is stored in the result register ADDAT, or in register ADDAT2 for an injected conversion.

Note Bit-field CHNR of register ADDAT is loaded by the ADC to indicate which channel the result refers to. Bit-field CHNR of register ADDAT2 is loaded by the CPU to select the analog channel, which is to be injected.

ADDAT (FEA0h / 50h) SFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CHNR				-	-	ADRES									
RW						RW									

ADDAT2 (F0A0h / 50h) ESFR Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CHNR				-	-	ADRES									
RW						RW									

Bit	Function
ADRES	**A/D Conversion Result (10 bits)**
CHNR	**Channel Number** (4 bits, identifies the converted analog channel)

A conversion is started by setting bit ADST='1'. The busy flag ADBSY will be set and the converter then selects and samples the input channel, which is specified by the channel selection field ADCH in register ADCON. The sampled level will then be held internally during the conversion.

When the conversion of this channel is complete, the 10-bit result together with the number of the converted channel is transferred into the result register ADDAT and the interrupt request flag ADCIR is set. If bit ADST is reset via software, while a conversion is in progress, the A/D converter will stop after the current conversion (fixed channel modes) or after the current conversion sequence (auto scan modes).

Setting bit ADST while a conversion is running, will abort this conversion and start a new conversion with the parameters specified in ADCON.

Note Stop and restart (see above) are triggered by bit ADST changing from '0' to '1', ADST must be '0' before being set.

While a conversion is in progress, the mode selection field ADM and the channel selection field ADCH may be changed. ADM will be evaluated after the current conversion. ADCH will be evaluated after the current conversion (fixed channel modes) or after the current conversion sequence (auto scan modes).

17.1.1 - Fixed Channel Conversion Modes

These modes are selected by programming the mode selection field ADM in register ADCON to '00b' (single conversion) or to '01b' (continuous conversion). After starting the converter through bit ADST, the busy flag ADBSY will be set and the channel specified in bit-field ADCH will be converted. After the conversion is complete, the interrupt request flag ADCIR will be set.

In single conversion mode the converter will automatically stop and reset bit ADBSY and ADST.

In continuous conversion mode the converter will automatically start a new conversion of the channel specified in ADCH. ADCIR will be set after each completed conversion. When bit ADST is reset by software, while a conversion is in progress, the converter will complete the current conversion and then stop and reset bit ADBSY.

17.1.2 - Auto Scan Conversion Modes

These modes are selected by programming the mode selection field ADM in register ADCON to '10_B' (single conversion) or to '11_B' (continuous conversion).

Auto Scan modes automatically convert a sequence of analog channels, beginning with the channel specified in bit-field ADCH and ending with channel 0, without requiring software to change the channel number.

After starting the converter through bit ADST, the busy flag ADBSY will be set and the channel specified in bit-field ADCH will be converted.

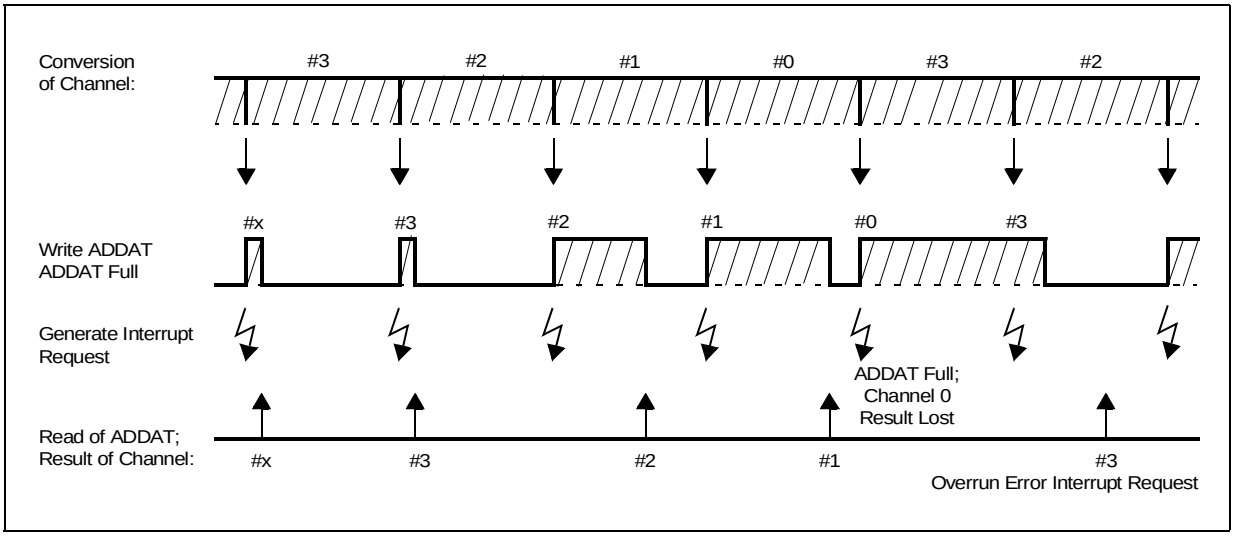
After the conversion is complete, the interrupt request flag ADCIR will be set and the converter will automatically start a new conversion of the next lower channel. ADCIR will be set after each completed conversion. After conversion of channel 0 the current sequence is complete.

In single conversion mode the converter will automatically stop and reset bits ADBSY and ADST.

In continuous conversion mode the converter will automatically start a new sequence beginning with the conversion of the channel specified in ADCH.

When bit ADST is reset by software, while a conversion is in progress, the converter will complete the current sequence (including conversion of channel 0) and then stop and reset bit ADBSY.

Figure 134 : Auto scan conversion mode example



17.1.3 - Wait for ADDAT Read Mode

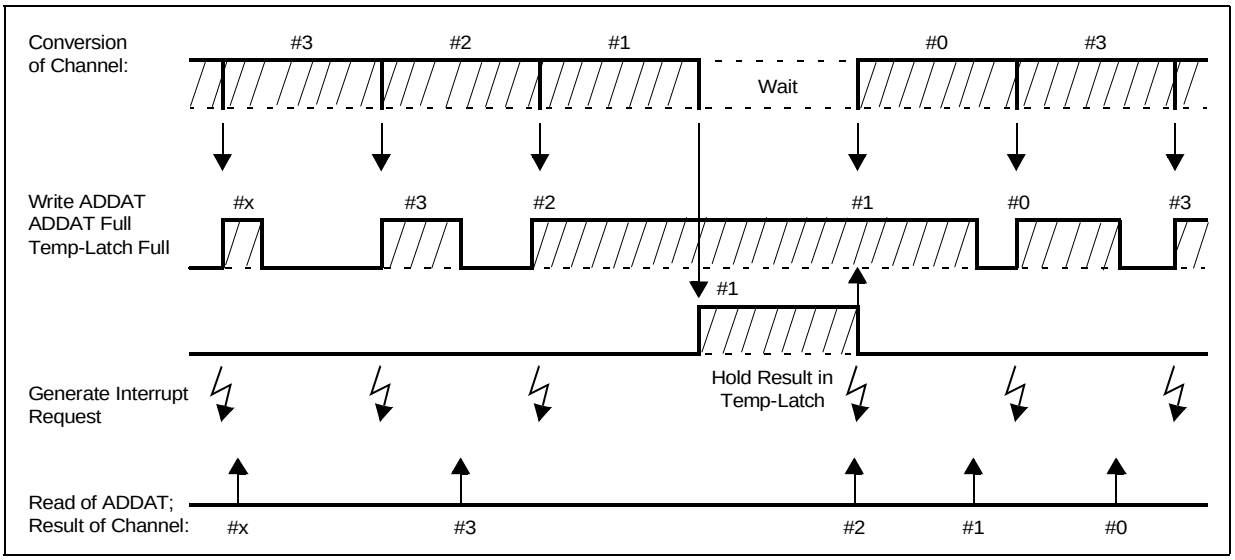
If in default mode of the ADC a previous conversion result has not been read out of register ADDAT by the time a new conversion is complete, the previous result in register ADDAT is lost because it is overwritten by the new value, and the A/D overrun error interrupt request flag ADEIR will be set.

In order to avoid error interrupts and the loss of conversion results especially when using continuous conversion modes, the ADC can be switched to "Wait for ADDAT Read Mode" by setting bit ADWR in register ADCON.

If the value in ADDAT has not been read by the time the current conversion is complete, the new result is stored in a temporary buffer and the next conversion is suspended (ADST and ADBSY will remain set in the meantime, but no end-of-conversion interrupt will be generated). After reading the previous value from ADDAT the temporary buffer is copied into ADDAT (generating an ADCIR interrupt) and the suspended conversion is started. This mechanism applies to both single and continuous conversion modes.

While in standard mode continuous conversions are executed at a fixed rate (determined by the conversion time), in "Wait for ADDAT Read Mode" there may be delays due to suspended conversions. However, this only affects the conversions, if the CPU (or PEC) cannot keep track with the conversion rate.

Figure 135 : Wait for read mode example



17.1.4 - Channel Injection Mode

Channel Injection Mode allows the conversion of a specific analog channel (also while the ADC is running in a continuous or auto scan mode) without changing the current operating mode. After the conversion of this specific channel, the ADC continues with the original operating mode.

Channel Injection mode is enabled by setting bit ADCIN in register ADCON and requires the Wait for ADDAT Read Mode (ADWR='1'). The channel to be converted in this mode is specified in bit-field CHNR of register ADDAT2.

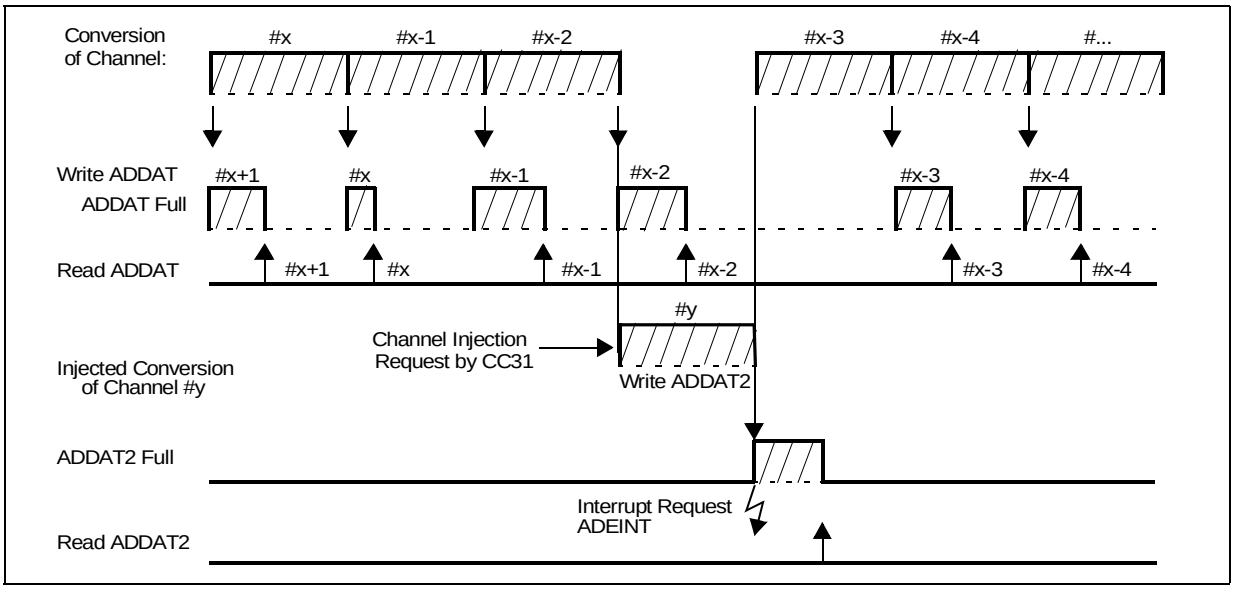
These 4 bits in ADDAT2 are not modified by the A/D converter, but only the ADRES bit-field. Since the channel number for an injected conversion is not buffered, bit-field CHNR of ADDAT2 must never be modified during the sample phase of an injected conversion, otherwise the input multiplexer will switch to the new channel. It is recommended to only change the channel number with no injected conversion running (see Figure 136). A channel injection can be triggered in two ways:

– Setting the Channel Injection Request bit ADCRQ via software a compare or a capture event of Capture/ Compare register CC31 of the CAPCOM2 Unit, which also sets bit ADCRQ.
– Triggering a channel injection at a specific time on the occurrence of a predefined count value of the CAPCOM timers or on a capture event of register CC31. This can be either the positive, negative, or both the positive and the negative edge of an external signal. In addition, this option allows recording the time of occurrence of this signal.

Note The channel injection request bit ADCRQ will be set on any interrupt request of CAPCOM2 channel CC31, regardless whether the channel injection mode is enabled or not. It is recommended to always clear bit ADCRQ before enabling the channel injection mode. While an injected conversion is in progress, no further channel injection request can be triggered. The Channel Injection Request flag ADCRQ remains set until the result of the injected conversion is written to the ADDAT2 register.
If the converter was idle before the channel injection, and during the injected conversion the converter is started by software for normal conversions, the channel injection is aborted, and the converter starts in the selected mode (as described above). This can be avoided by checking the busy bit ADBSY before starting a new operation.

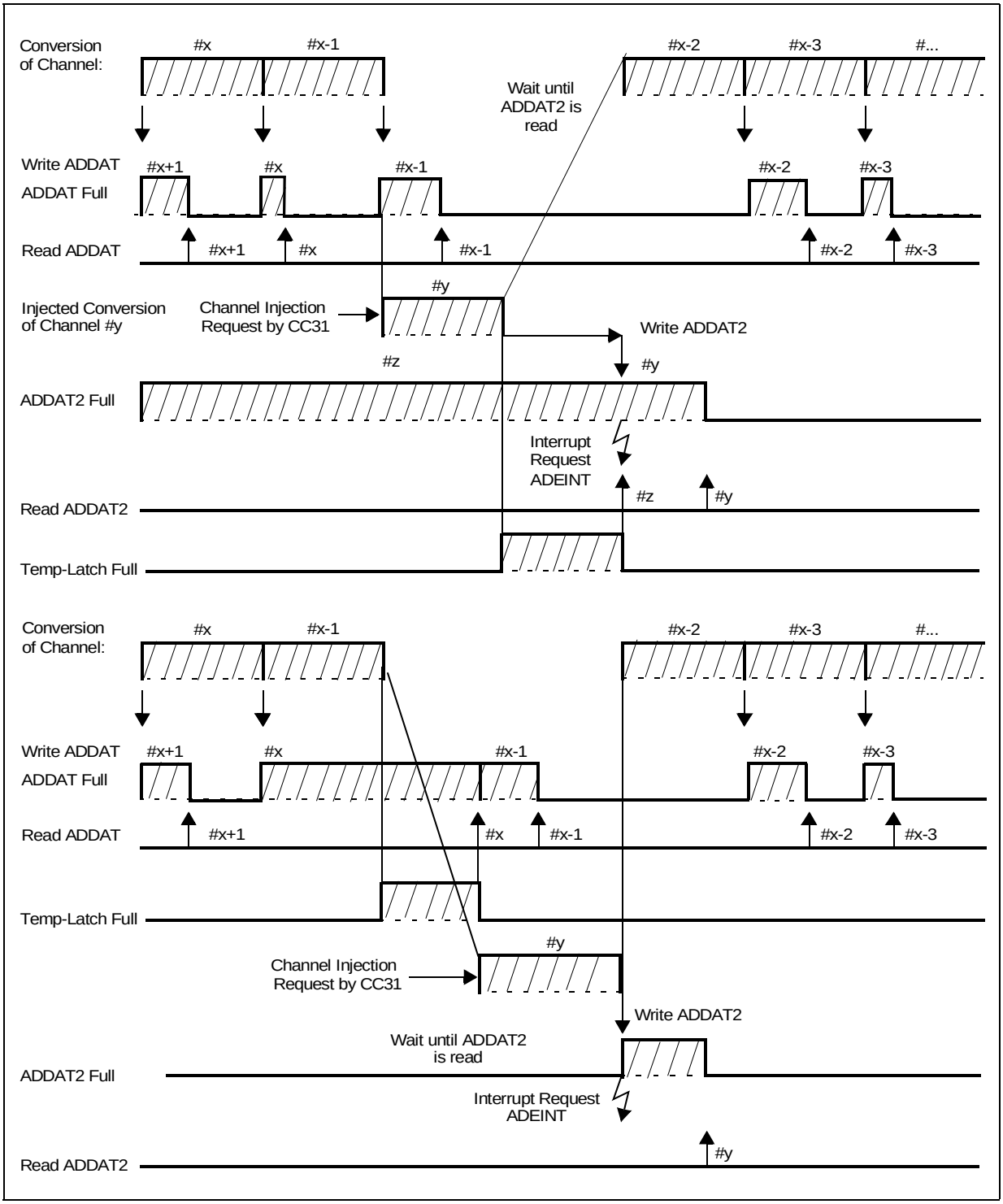
After the completion of the current conversion (if any is in progress) the converter will start (inject) the conversion of the specified channel. When the conversion of this channel is complete, the result will be placed into the alternate result register ADDAT2, and a Channel Injection Complete Interrupt request will be generated, which uses the interrupt request flag ADEIR (for this reason the Wait for ADDAT Read Mode is required).

Figure 136 : Channel injection example



If the temporary data register used in Wait for ADDAT Read Mode is full, the respective next conversion (standard or injected) will be suspended. The temporary register can hold data for ADDAT (from a standard conversion) or for ADDAT2 (from an injected conversion).

Figure 137 : Channel injection example with wait for read



17.2 - Conversion Timing Control

The time that the two different actions during conversion take (sampling, and converting) can be programmed within a certain range in the ST10F280 relative to the CPU clock. The absolute time that is consumed by the different conversion steps therefore is independent of the general speed of the controller. This allows adjusting the A/D converter of the ST10F280 to the properties of the system:

Fast conversion can be achieved by programming the respective times to their absolute possible minimum. This is preferable for scanning high frequency signals. The internal resistance of analog source and analog supply must be sufficiently low, however.

High internal resistance can be achieved by programming the respective times to a higher value, or the possible maximum.

This is preferable when using analog sources and supply with a high internal resistance in order to keep the current as low as possible. The conversion rate in this case may be considerably lower, however.

The conversion times are programmed via the upper four bit of register ADCON. bit-field ADCTC (conversion time control) selects the basic conversion clock, used for the 14 steps of converting. The sample time is a multiple of this conversion time and is selected by bit-field ADSTC (sample time control). The table below lists the possible combinations. The timings refer to the unit TCL, where f_{CPU} = 1/2TCL.

Table 47 : ADC Sampling and Conversion Timing

ADCTC	Conversion Clock t_{CC}		ADSTC	Sample Clock t_{SC}	
	TCL = 1/2 x f_{XTAL}	At f_{CPU} = 40MHz		t_S =	At f_{CPU} = 40MHz
00	TCL x 24	0.3µs	00	t_{CC} x 2	0.6µs
01	Reserved, do not use	Reserved	01	t_{CC} x 4	1.2µs
10	TCL x 96	1.2 µs	10	t_{CC} x 8	2.4µs
11	TCL x 48	0.6 µs	11	t_{CC} x 16	4.8µs

A complete conversion will take $14t_{CC}$ + t_{SC} + 4TCL (fastest convertion rate = 4.85µs at 40MHz). This time includes the conversion itself, the sample time and the time required to transfer the digital value to the result register.

Note The ST10F280 ADC does not implement any auto-calibration feature.

17.3 - A/D Converter Interrupt Control

At the end of each conversion, interrupt request flag ADCIR in interrupt control register ADCIC is set. This end-of-conversion interrupt request may cause an interrupt to vector ADCINT, or it may trigger a PEC data transfer which reads the conversion result from register ADDAT it can be stored it into a table in the internal RAM for later evaluation.

The interrupt request flag ADEIR in register ADEIC will be set, either if a conversion result overwrites a previous value in register ADDAT (error interrupt in standard mode), or if the result of an injected conversion has been stored into ADDAT2 (end-of-injected-conversion interrupt). This interrupt request may be used to cause an interrupt to vector ADEINT, or it may trigger a PEC data transfer.

ADCIC (FF98h / CCh) SFR Reset Value: - - 00h

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

ADEIC (FF9Ah / CDh) SFR Reset Value: - - 00h

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

Note Refer to Section 6.1.3 - Interrupt Control Registers for explanation of the control fields.

17.4 - XTIMER Peripheral

This new peripheral is dedicated for the Channel Injection Mode of the A/D converter. This mode injects a channel into a running sequence without disturbing this sequence. The peripheral event controller stores the conversion results in memory without entering and exiting interrupt routines for each data transfer.

A channel injection can be triggered by an event on Capture/Compare CC31 (Port P7.7) of the CAPCOM2 unit.

The dedicated output XADCINJ of the XTIMER must be connected externally on the input P7.7/CC31.

Due to the multiplexed inputs, at a time, the ADC exclusively converts the Port5 inputs or the XPort10 inputs. If one "y" channel has to be used continuously in injection mode, it must be externally hardware connected to the Port5.y and XPort10.y inputs.

The XTIMER peripheral is enabled by setting XPEN bit 2 of the SYSCON register and bit 3 of the new XPERCON register.

17.4.1 - Main Features

The XTIMER features are :

– 16 bits linear timer / 4 bits exponential prescaler

– Counting between 16 bits “start value” and 16 bits “end value”

– Counting period between 4 cycles and 2**33 cycles (100 ns and 214s using 40MHz CPU clock)

– 1 trigger ouput (XADCINJ)

– Programmable functions :

- Internal clock XCLK is derivated from the CPU clock and has the same period
- Up counting / down counting
- Reload enable
- Continue / stop modes

– 4 memory mapped registers :

- Control / prescaler
- Start value
- End value
- Current value

Table 48 : The Different Counting Modes

TLE	TCS	TCVR(n) = TEVR	TUD	TEN	TCVR(n+1)	comments
x	x	x	x	0	TCVR(n)	Timer disable
x	0	1	x	1		Stop
x	x	0	0		TCVR(n)-1	Decrement
0	1	1				Decrement (Continue)
x	x	0	1		TCVR(n)+1	Increment
0	1	1				Increment (Continue)
1	1	1	x		TSVR	Load

17.4.2 - Register Description

XTCR : Timer Control Register

XTCR (C000h) XBUS 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	TFP[3:0]				TCM	TIE	TCS	TLE	TUD	TEN
R	R	R	R	R	R	RW				RW	RW	RW	RW	RW	RW

Bit	Function
TEN	**Timer Enable** When TEN = '0', the Timer is **disabled** (reset value). To avoid glitches, it is recommended to modify TCR in 2 steps, first with new values and and second by setting TEN.
TUD	**Timer Up / Down Counting** When TUD = '0', the Timer is counting **"down"** (reset value), ie the TCVR ('current value') register content is decremented. When TUD = '1', the Timer is counting **"up"**, ie the TCVR ('current value') register content is incremented.
TLE	**Timer Load Enable** When the counter has reached its end value (TCVR = TEVR), TCVR is **(re)loaded** with TSVR ('start value') register content when TLE = '1'. When TLE = '0' (reset value), the next state of TCVR depends on TCS bit.
TCS	**Timer Continue / Stop** When TLE = '0' (no load) and when the counter has reached its end value (TCVR = TEVR), the TCVR content continues to increment / decrement according to TUD bit when TCS = '1' (**continue** mode). When TCS = '0' (**stop** mode reset value), TCVR is stopped and its content is frozen.
TIE	**Timer Output Enable** When the counter has reached its end value (TCVR = TEVR), the XADCINJ output is set when TIE = '1'. When TIE = '0' (reset value), XADCINJ output is disabled (= '0').
TCM	**Timer Clock Mode Must be Cleared** TCM = '0' (reset value), the TCVR clock is derived from **internal** XCLK clock according to TFP bits.
TFP[3:0]	**Timer Frequency Prescaler** When TCM = '0' (internal clock), the TCVR register clock is derived from the XCLK clock input by dividing XCLK by 2**(2+ TFP). The coding is as follows : - 0000 : prescaler by 2 (reset value), XCLK divided by 4 - 0001 : prescaler by 4, XCLK divided by 8 - 0010 : prescaler by 8, XCLK divided by 16 - ... - 1111 : prescaler by 2**16, XCLK divided by 2**17

XTSVR : Timer Start Value Register

XTSVR(C002h) XBUS Reset Value: 0000h

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

Bit	Function
TSVR[15:0]	**Timer Start Value** TSVR contains the data to be transferred to the TCVR 'Current Value' register when : 1) - TEN = '1' (TIM enable), - TLE = '1' (TIM Load enable), - TCVR = TEVR (count period finished), - TCS = '1' (stop mode disabled). 2) - first counting clock rising edge after the timer start (the timer starts on TEN rising edge).

XTEVR : Timer End Value Register

XTEVR(C004h) XBUS Reset Value: 0000h

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

Bit	Function
TEVR[15:0]	**Timer End Value** TEVR contains the data to be compared to the TCVR 'Current Value' register.

XTCVR : Timer Current Value Register

XTCVR(C006h) XBUS Reset Value: 0000h

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

Bit	Function
TCVR[15:0]	**Timer Current Value** TCVR contains the current counting value. When TCVR = TEVR, TCVR content is changed according to Table 48. The TCVR clock is derived from internal XCLK clock according to TFP bits when TCM = '0'.

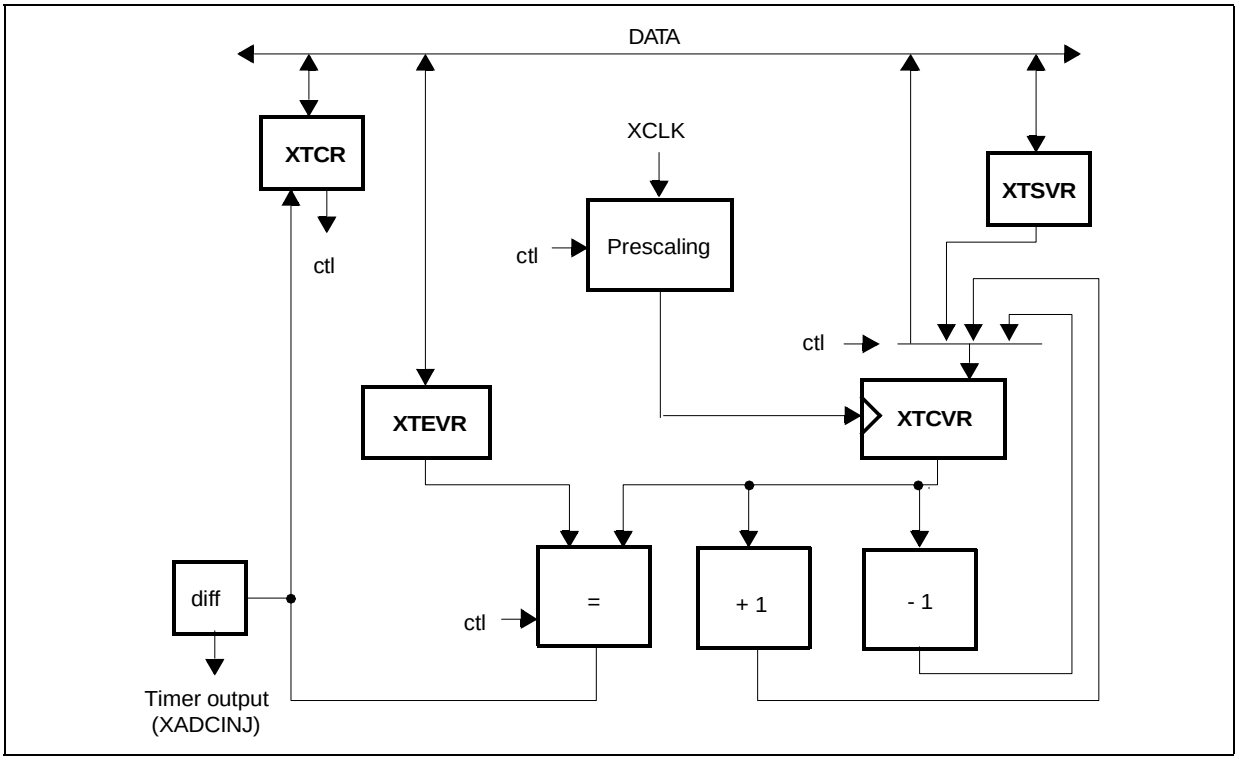
Registers Mapping

Table 49 : Timer Registers Mapping

Address (Hexa)	Register Name	Reset Value (Hexa)	Access
C000	**XTCR** : Control	0000	RW
C002	**XTSVR** : Start Value	0000	RW
C004	**XTEVR** : End Value	0000	RW
C006	**XTCVR** : Current Value	0000	R

17.4.3 - Block Diagram

Figure 138 : XTIMER Block Diagram



17.4.3.1 - Clocks

The XTCVR register clock is the prescaler output. The prescaler allows to divide the basic register frequency in order to offer a wide range of counting period, from 2**2 to 2**33 cycles (note that 1 cycle = 1 XCLK periods).

17.4.3.2 - Registers

The XTCVR register input is linked to several sources:

– XTSVR register (start value) for reload when the period is finished, or for load when the timer is starting.

– Incrementer output when the 'up' mode is selected,

– Decrementer output when the 'down' mode is selected.

– The selection between the sources is made through the XTCR control register.

When starting the timer, by setting TEN bit of TCR to '1', XTCVR will be loaded with XTSVR value on the first rising edge of the counting clock. That's to say that for counting from 0000h to 0009h for example, 10 counting clock rising edges are required.

The XTCVR register output is continuously compared to the XTEVR register to detect the end of the counting period. When the registers are equal, several actions are made depending on the XTCR control register content :

– The output XADCINJ is conditionally generated,

– XTCVR is loaded with XTSVR or stops or continues to count (see Table 48).

XTEVR, XTSVR and all TCR bits except TEN must not be modified while the timer is counting, ie while TEN bit of TCR = '1'. The timer behaviour is not guaranteed if this rule is not respected. It implies that the timer can be configured only when stopped (TEN = '0'). When programing the timer, XTEVR, XTSVR and XTCR bits except TEN can be modified, with TEN = '0'; then the timer is started by modifying only TEN bit of TCR. To stop the timer, only TEN bit should be modified, from '1' to '0'.

17.4.3.3 - Timer output (XADCINJ)

The XADCINJ output is the result of the (XTCVR = XTEVR) flag after differentiation. The duration of the output lasts two cycles (50ns at 40MHz).

Figure 139 : XADCINJ Timer Output

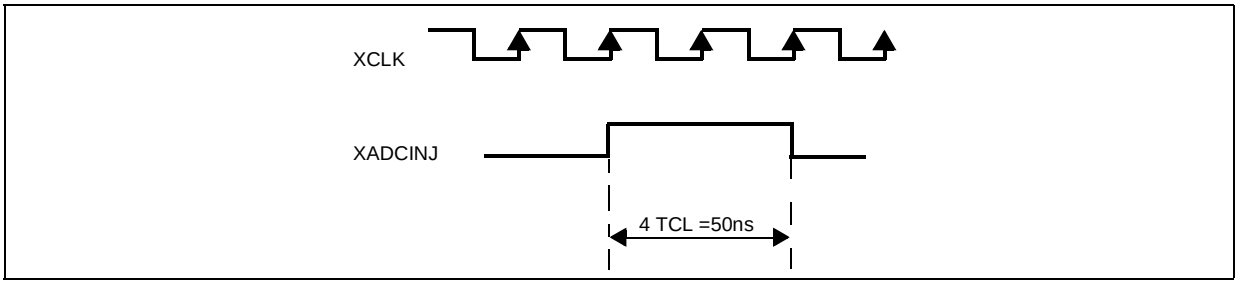
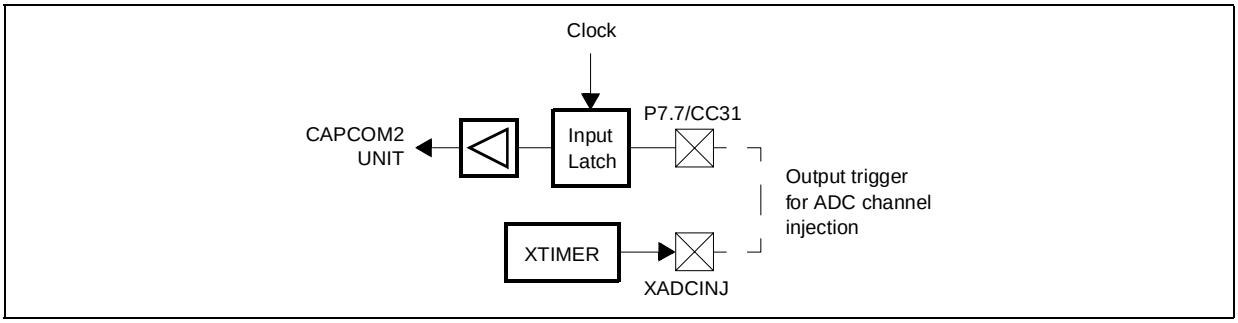


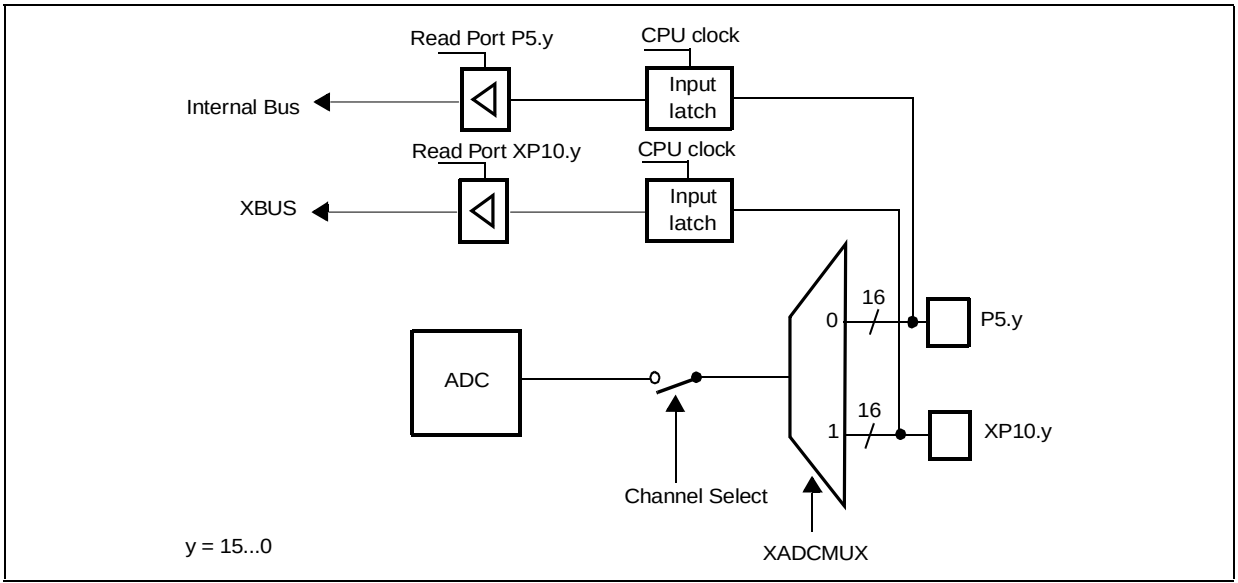
Figure 140 : External connection for ADC channel injection



17.5 - A/D Converter Input Multiplexer

The ADC can manage 16 analog inputs so to increase his capability, a new register XADCMUX is added to control the multiplexage between the first block of 16 channel (on Port5) and the second block (on XPort10). The conversion result register stay identical so only a software management can determine the block used.

Figure 141 : Block Diagram



The XADCMUX register is enabled by setting XPEN bit 2 of the SYSCON register and bit 3 of the new XPERCON register

XADCMUX (C384h) XBUS Reset Value: 0000h

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

Bit	Function	
XADCMUX.0	0	Default configuration,analog inputs on port P5.y can be converted
	1	Analog inputs on port XP10.y can be converted

18 - ON-CHIP CAN INTERFACES

The ST10F280 provides 2 CAN modules with identical features.

The integrated CAN Modules handles the autonomous transmission and reception of CAN frames in accordance with the CAN specification V2.0 part B (active), the on-chip CAN Modules can receive and transmit standard frames with 11-bit identifiers and extended frames with 29-bit identifiers.

The CANS provide **Full CAN** functionality on up to 15 full sized message objects (8 data byte each). Message object 15 may be configured for **Basic CAN** functionality with a double-buffered receive object.

Full CAN and **Basic CAN** modes both provide separate masks for acceptance filtering, accepting identifiers in Full CAN mode and disregarding identifiers in Basic CAN mode.

All message objects can be updated independently from the other objects and are equipped with buffers for the maximum message length of 8 bytes.

The bit timing is derived from the XCLK = f_{CPU} and is programmable up to a data rate of 1M Baud for CPU clock higher than 16 MHz. A CAN Module uses two pins of Port 4 to interface to a bus transceiver.

The CAN1 transmit line is connected to Port 4.6 and the receive line to Port 4.5. The CAN2 transmit line is connected to Port 4.7 and the receive line to Port 4.4.

18.1 - The CAN Controller

The CAN modules combine several functional blocks that work in parallel. These units and the functions they provide are described below.

Each of the message objects has a unique identifier and its own set of control and status bit. Each object can be configured for transmit or receive direction, except the last message which is a double receive buffer with a special mask register.

An object with its direction set as transmit can be configured to be automatically sent whenever a remote frame with a matching identifier (taking into account the respective global mask register) is received over the CAN bus.

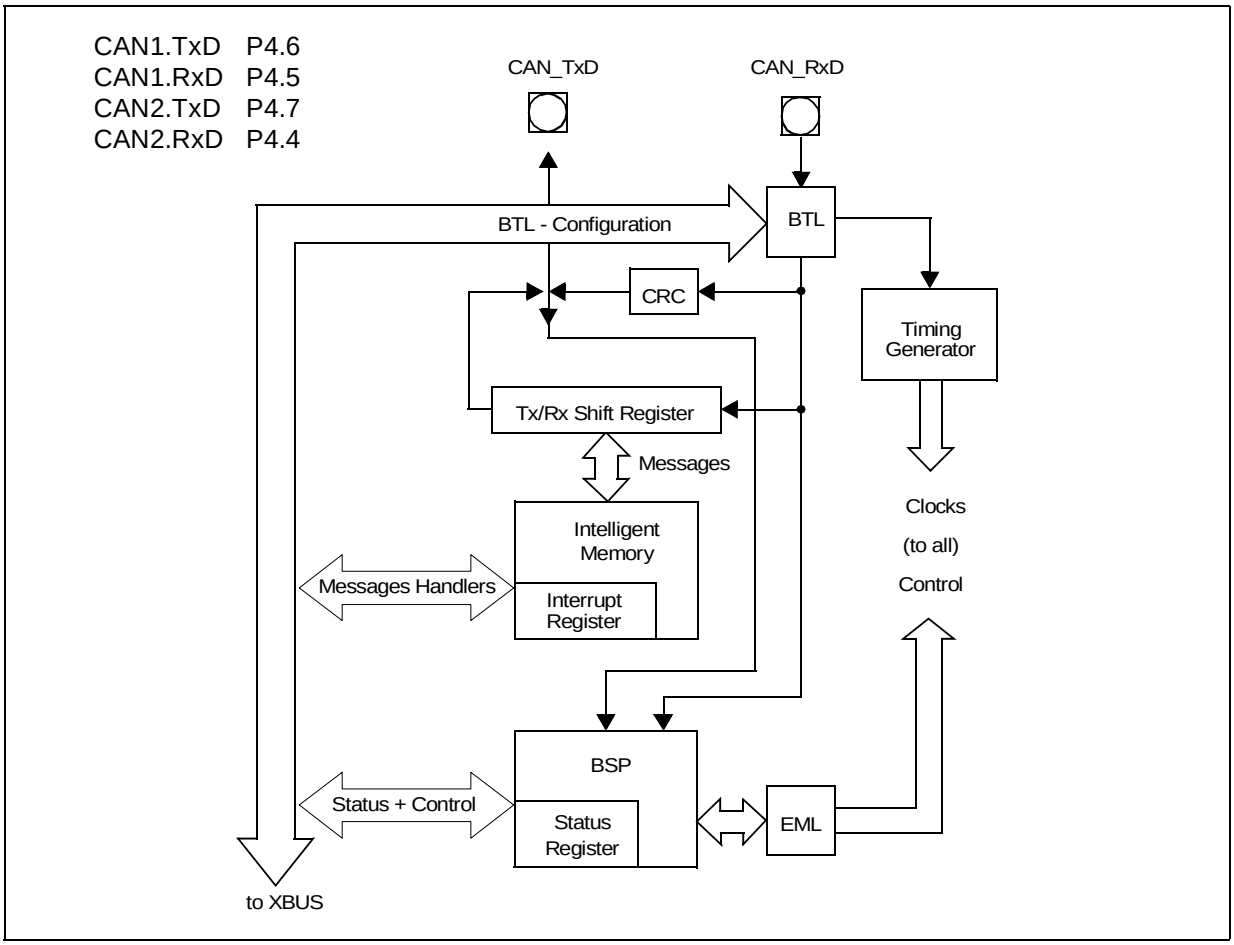
By requesting the transmission of a message with the direction set as receive, a remote frame can be sent to request that the appropriate object be sent by some other node.

Each object has separate transmit and receive interrupts and status bit, giving the CPU full flexibility in detecting when a remote/data frame has been sent or received.

Two acceptance filtering masks can be programmed for general purpose, one for identifiers of 11 bits and one for identifiers of 29 bits. However, the CPU must configure bit XTD (Normal or Extended Frame Identifier) for each valid message, to determine whether a standard or extended frame will be accepted.

The last message object has its own programmable mask for acceptance filtering, allowing a large number of infrequent objects to be handled by the system.

Figure 142 : CAN Block Diagram



Tx/Rx Shift Register

The Transmit / Receive Shift Register holds the destuffed bit stream from the bus line to give parallel access to the whole data or remote frame for the acceptance match test and the parallel transfer of the frame to and from the Intelligent Memory.

Bit Stream Processor

The bit Stream Processor (BSP) is a sequencer, controlling the sequential data stream between the Tx/Rx Shift Register, the CRC Register, and the bus line. The BSP also controls the EML and the parallel data stream between the Tx/Rx Shift Register and the Intelligent Memory such that the processes of reception, arbitration, transmission, and error signalling are performed according to the CAN protocol. Note that the automatic retransmission of messages which have been corrupted by noise or other external error conditions on the bus line is handled by the BSP.

Cyclic Redundancy Check Register

This register generates the Cyclic Redundancy Check (CRC) code to be transmitted after the data byte and checks the CRC code of incoming messages. This is done by dividing the data stream by the code generator polynomial.

Error Management Logic

The Error Management Logic (EML) is responsible for the fault confinement of the CAN device. Its counters, the Receive Error Counter and the Transmit Error Counter, are incremented and decremented by commands from the bit Stream Processor. According to the values of the error counters, the CAN controller is set into the states: error active, error passive and busoff.

The CAN controller is error active, if both error counters are below the error passive limit of 128.

It is error passive, if at least one of the error counters equals or exceeds 128.

It goes busoff, if the Transmit Error Counter equals or exceeds the busoff limit of 256. The device remains in this state until the busoff recovery sequence is finished.

Additionally, there is the bit EWRN in the Status Register, which is set, if at least one of the error counters equals or exceeds the error warning limit of 96. EWRN is reset, if both error counters are less than the error warning limit.

Bit Timing Logic

This block (BTL) monitors the bus line input CAN_RxD and handles the bus line related bit timing according to the CAN protocol.

The BTL synchronizes on a recessive to dominant bus line transition at Start of Frame (hard synchronization) and on any further recessive to dominant bus line transition, if the CAN controller itself does not transmit a dominant bit (resynchronization).

The BTL also provides programmable time segments to compensate for the propagation delay time and for phase shifts and to define the position of the Sample Point in the bit time. The programming of the BTL depends on the Baud rate and on external physical delay times.

Intelligent Memory

The Intelligent Memory (CAN/RAM Array) provides storage for up to 15 message objects of maximum 8 data byte length. Each of these objects has a unique identifier and its own set of control and status bit. After the initial configuration, the Intelligent Memory can handle the reception and transmission of data without further CPU actions.

18.2 - Register and Message Object Organization

All registers and message objects of the CAN controller are located in the special CAN address area of 256 bytes, which is mapped into segment 0. The CAN1 address range is 00'EF00h-EFFFh and the CAN2 range is 00'EE00h-EEFFh. All registers are organized as 16-bit registers, located on word addresses. However, all registers may be accessed byte wise in order to select special actions without effecting other mechanisms.

Note The address map shown in Figure 143 lists the registers which are part of the CAN controller. There are also ST10F280 specific registers that are associated with the CAN Modules. These registers, however, control the access to the CAN Modules rather than their function.

Visibility of XBUS Peripherals

In order to keep the ST10F280 compatible with the ST10C167 and with the ST10F167, the XBUS peripherals can be selected to be visible and / or accessible on the external address / data bus. CAN1EN and CAN2EN bit of XPERCON register must be set. If these bit are cleared before the global enabling with XPEN-bit in SYSCON register, the corresponding address space, port pins and interrupts are not occupied by the peripheral, thus the peripheral is not visible and not available. Refer to Chapter 21 - Register Set.

XPERCON (F024h / 12h) ESFR Reset Value: 0005h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	XPWMEN	XPERCONEN3	XRAMEN	CAN2EN	CAN1EN
											RW	RW	RW	RW	RW

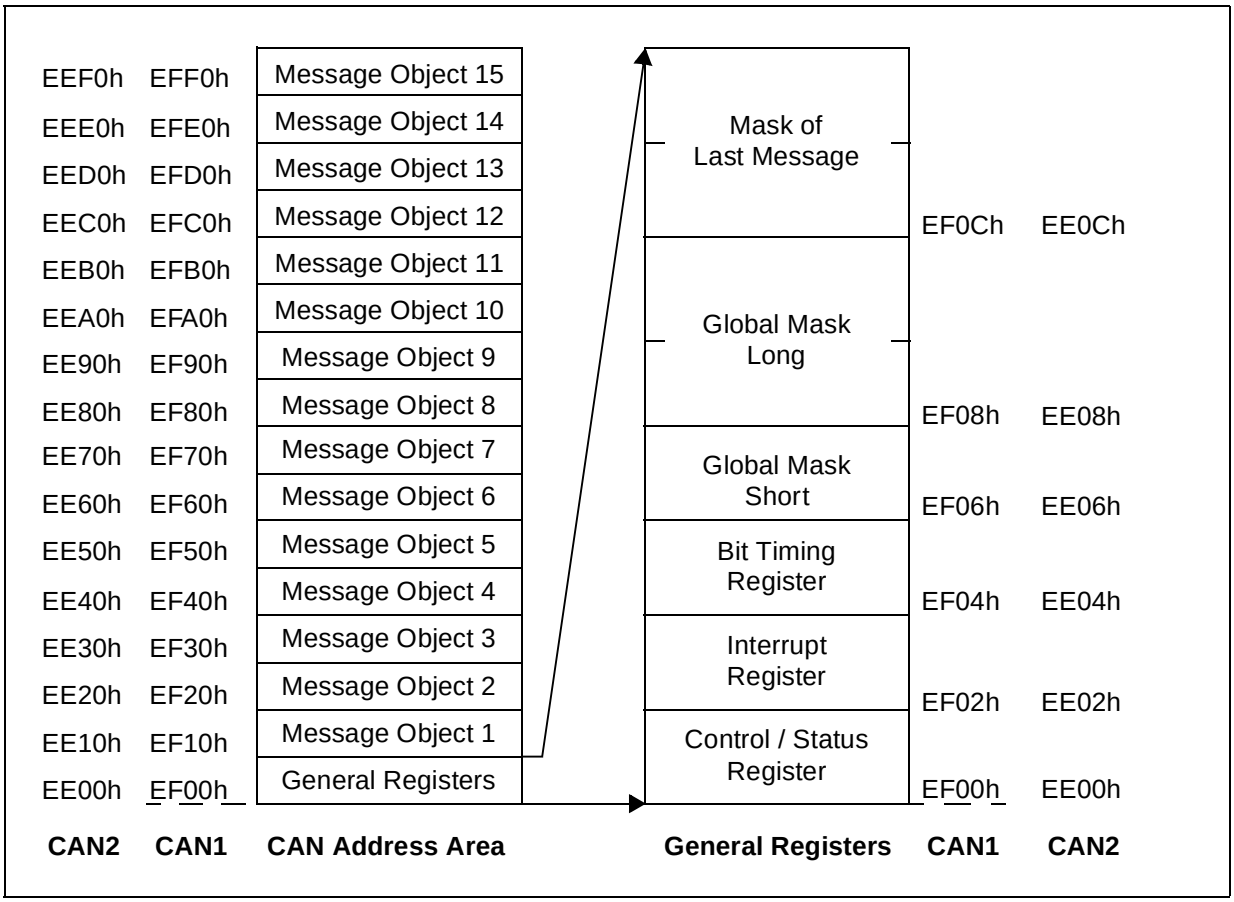
Bit	Function
CAN1EN	**CAN1 Enable Bit** 0 Accesses to the on-chip CAN1 XPeripheral and its functions are disabled. P4.5 and P4.6 pins can be used as general purpose I/Os. Address range 00'EF00h-00'EFFFh is only directed to external memory if CAN2EN and XPWM bits are cleared also. 1 The on-chip CAN1 XPeripheral is enabled and can be accessed.
CAN2EN	**CAN2 Enable Bit** 0 Accesses to the on-chip CAN2 XPeripheral and its functions are disabled. P4.4 and P4.7 pins can be used as general purpose I/Os. Address range 00'EE00h-00'EEFFh is only directed to external memory if CAN1EN and XPWM bits are cleared also. 1 The on-chip CAN2 XPeripheral is enabled and can be accessed.
XRAMEN	**XRAM Enable Bit** 0 Accesses to the on-chip 16K Byte XRAM are disabled, external access performed. 1 The on-chip 16K Byte XRAM is enabled and can be accessed.
XPERCONEN3	**XPORT9, XTIMER, XPORT10, XADCMUX Enable Bit** 0 Accesses to the XPORT9, XTIMER, XPORT10, XADCMUX peripherals are disabled, external access performed. 1 The on-chip XPORT9, XTIMER, XPORT10, XADCMUX peripherals are enabled and can be accessed.
XPWMEN	**XPWM Enable Bit** 0 Accesses to the on-chip XPWM are disabled, external access performed. Address range 00'EC00h-00'ECFFh is only directed to external memory if CAN1EN and CAN2EN are '0' also 1 The on-chip XPWM is enabled and can be accessed.

Note: - When both CAN and XPWM are disabled via XPERCON setting, then any access in the address range 00'EC00h 00'EFFFh will be directed to external memory interface, using the BUSCONx register corresponding to address matching ADDRSELx register. P4.4 and P4.7 can be used as General Purpose I/O when CAN2 is not enabled, and P4.5 and P4.6 can be used as General Purpose I/O when CAN1 is not enabled.

- The default XPER selection after Reset is : XCAN1 is enabled, XCAN2 is disabled, XRAM is enabled, XPORT9, XTIMER, XPORT10, XPWM, XADCMUX are disabled.

- Register XPERCON cannot be changed after the global enabling of XPeripherals, i.e. after setting of bit XPEN in SYSCON register.

Figure 143 : CAN module address map



Control / Status Register (EE00h-EF00h) XReg Reset Value: XX01h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BOFF	**E WRN**	-	**RXOK**	**TXOK**	**LEC**			TST	CCE	0	0	EIE	SIE	IE	**INIT**
R	R		RW	RW	RW			RW	RW	R	R	RW	RW	RW	RW

Table 50 : CAN Control/Status register

Bit	Function (Control bit)
INIT	**Initialization** 1: Software initialization of the CAN controller. While init is set, all message transfers are stopped. Setting init does not change the configuration registers and does not stop transmission or reception of a message in progress. The init bit is also set by hardware, following a busoff condition; the CPU then needs to reset init to start the bus recovery sequence. see Figure 152. 0: Disable software initialization of the CAN controller; on INI completion, the CAN waits for 11 consecutive recessive bits before taking part in bus activities.
IE	**Interrupt Enable** - Does not affect status updates. 1: Global interrupt enable from CAN module. 0: Global interrupt disable from CAN module.

Bit	Function (Control bit)
SIE	**Status Change Interrupt Enable** 1: Enables interrupt generation when a message transfer (reception or transmision is successfully completed) or CAN bus error is detected and registered in LEC is the status partition. 0: Disable status change interrupt.
EIE	**Error Interrupt Enable** 1: Enables interrupt generation on a change of bit BOFF or EWARN in the status partition. 0: Disable error interrupt.
CCE	**Configuration Change Enable** 1: Allows CPU access to the bit timing register 0: Disables CPU access to the bit timing register
TST	**Test Mode (bit 7)** Make sure that bit 7 is cleared when writing to the Control Register. Writing a 1 during normal operation may lead erroneous device behaviour.
LEC	**Last Error Code** This field holds a code which indicates the type of the last error occurred on the CAN bus. If a message has been transferred (reception or transmission) without error, this field will be cleared. Code "7" is unused and may be written by the CPU to check for updates. 0: **No Error** 1: **Stuff Error:** More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed. 2: **Form Error:** A fixed format part of a received frame has the wrong format. 3: **AckError:** The message this CAN controller transmitted was not acknowledged by another node 4: **Bit1Error:** During the transmission of a message (with the exception of the arbitration field), the device wanted to send a *recessive* level ("1"), but the monitored bus value was *dominant* 5: **Bit0Error:** During the transmission of a message (or acknowledge bit, active error flag, or overload flag), the device wanted to send a *dominant* level ("0"), but the monitored bus value was *recessive*. During *busoff* recovery this status is set each time a sequence of 11 *recessive* bits has been monitored. This enables the CPU to monitor the proceeding of the busoff recovery sequence (indicating the bus is not stuck at *dominant* or continuously disturbed). 6: **CRCError:** The CRC check sum was incorrect in the message received.
TXOK	**Transmitted Message Successfully** Indicates that a message has been transmitted successfully (error free and acknowledged by at least one other node), since this bit was last reset by the CPU (the CAN controller does not reset this bit!).
RXOK	**Received Message Successfully** Indicates that a message has been received successfully, since this bit was last reset by the CPU (the CAN controller does not reset this bit!).
EWRN	**Error Warning Status** Indicates that at least one of the error counters in the EML has reached the error warning limit of 96.
BOFF	**Busoff Status** Indicates when the CAN controller is in busoff state (see EML).

Note Reading the upper half of the Control Register (status partition) will clear the Status Change Interrupt value in the Interrupt Register, if it is pending. Use byte accesses to the lower half to avoid this.

18.3 - CAN Interrupt Handling

The CAN Modules have one interrupt output, which is connected (through a synchronization stage) to a standard interrupt node in the ST10F280 in the same manner as all other interrupts of the standard on-chip peripherals. The CAN1 control register interrupt is XP0IC (located at address F186h/C3h in the ESFR range). The associated interrupt vector is called XP0INT at location 100h (trap number 40h). The CAN2 control register interrupt is XP1IC (located at address F18Eh/C7h in the ESFR range. The associated interrupt vector is called XP1INT at location 104h (trap number 41h). With this configuration, the user has all control options available for this interrupt, such as enabling/disabling, level and group priority, and interrupt or PEC service (see note below).

XP0IC (F186h / C3h) ESFR Reset Value: - - 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	**X0IR**	X1IE	ILVL				GLVL	
								RW	RW	RW				RW	

XP1IC (F18Eh / C7h) ESFR Reset Value: - - 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	**X1IR**	X1IE	ILVL				GLVL	
								RW	RW	RW				RW	

Bit	Function
GLVL	**Group Level** Defines the internal order for simultaneous requests of the same priority. 3: Highest group priority 0: Lowest group priority
ILVL	**Interrupt Priority Level** Defines the priority level for the arbitration of requests. 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

As for all other interrupts, the interrupt request flag XPXIR in register XPXIC is cleared automatically by hardware when this interrupt is serviced (either by standard interrupt or PEC service).

Note As a rule, CAN interrupt requests can be serviced by a PEC channel. However, because PEC channels only can execute single predefined data transfers (there are no conditional PEC transfers), PEC service can only be used, if the respective request is known to be generated by one specific source, and that no other interrupt request will be generated in between. In practice this seems to be a rare case.

Since an interrupt request of the CAN Module can be generated due to different conditions, the appropriate CAN interrupt status register must be read in the service routine to determine the cause of the interrupt request. The Interrupt Identifier INTID (a number) in the Interrupt Register indicates the cause of an interrupt. When no interrupt is pending, the identifier will have the value 00h. If the value in INTID is not 00h, then there is an interrupt pending. If bit IE in the Control Register is set, also the interrupt line to the CPU is activated. The interrupt line remains active until either INTID gets 00h (after the interrupt requester has been serviced) or until IE is reset (if interrupts are disabled).

The interrupt with the lowest number has the highest priority. If a higher priority interrupt (lower number) occurs before the current interrupt is processed, INTID is updated and the new interrupt overrides the last one. The Table 51 lists the valid values for INTID and their corresponding interrupt sources.

Interrupt Register (EF02h-EE02h) XReg Reset Value: - - XXh

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

Bit	Function
INTID	**Interrupt Identifier** This number indicates the cause of the interrupt. When no interrupt is pending, the value will be "00".

Table 51 : INTID values and Corresponding Interrupt Sources

INTID	Cause of the Interrupt
00	**Interrupt Idle:** There is no interrupt request pending.
01	**Status Change Interrupt:** The CAN controller has updated (not necessarily changed) the status in the Control Register. This can refer to a change of the error status of the CAN controller (EIE is set and BOFF or EWRN change) or to a CAN transfer incident (SIE must be set), like reception or transmission of a message (RXOK or TXOK is set) or the occurrence of a CAN bus error (LEC is updated). The CPU may clear RXOK, TXOK, and LEC, however, writing to the status partition of the Control Register can never generate or reset an interrupt. To update the INTID value the status partition of the Control Register must be read.
02	**Message 15 Interrupt:** bit INTPND in the Message Control Register of message object 15 (last message) has been set. The last message object has the highest interrupt priority of all message objects. 1)
(2+N)	**Message N Interrupt:** bit INTPND in the Message Control Register of message object 'N' has been set (N = 1...14). 1) 2)

Notes 1) Bit INTPND of the corresponding message object has to be cleared to give messages with a lower priority the possibility to update INTID or to reset INTID to 00h (idle state).

2) A message interrupt code is only displayed, if there is no other interrupt request with a higher priority.

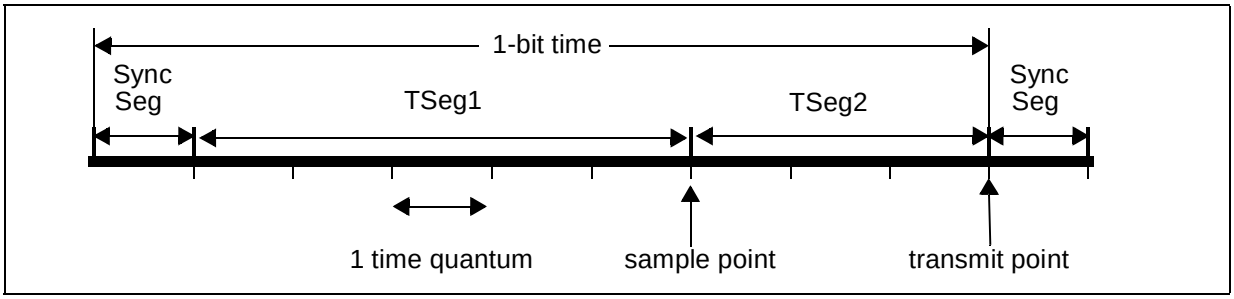
Bit Timing Configuration

According to the CAN protocol specification, a bit time is subdivided into four segments:
Sync segment, propagation time segment, phase buffer segment 1 and phase buffer segment 2.
Each segment is a multiple of the time quantum t_q
with $t_q = (BRP + 1) \times 2 \times t_{XCLK}$

The Synchronization Segment (Sync seg) is always 1 t_q long. The Propagation Time Segment and the Phase Buffer Segment1 (combined to Tseg1) defines the time before the sample point, while Phase Buffer Segment2 (Tseg2) defines the time after the sample point. The length of these segments is programmable (except Sync-Seg).

Note For exact definition of these segments please refer to the CAN Specification.

Figure 144 : Bit timing definition



Bit Timing Register (EF04h-EE04h) XReg Reset Value: UUUUh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	TSEG2			TSEG1				SJW		BRP					
R	RW			RW				RW		RW					

Bit	Function
BRP	**Baud Rate Prescaler** For generating the bit time quanta the CPU frequency is divided by 2 x (BRP+1).
SJW	**(Re)Synchronization Jump Width** Adjust the bit time by maximum (SJW+1) time quanta for re-synchronization.
TSEG1	**Time Segment before sample point** There are (TSEG1+1) time quanta before the sample point. Valid values for TSEG1 are "2...15".
TSEG2	**Time Segment after sample point** There are (TSEG2+1) time quanta after the sample point. Valid values for TSEG2 are "1...7".

Note This register can only be written, if the configuration change enable bit (CCE) is set.

Mask Registers

Messages can use standard or extended identifiers. Incoming frames are masked with their appropriate global masks. Bit IDE of the incoming message determines whether the standard 11 bit mask in Global Mask Short or the 29-bit extended mask in Global Mask Long is to be used. Bit holding a "0" mean "don't care", so do not compare the message's identifier in the respective bit position.

The last message object (15) has an additional individually programmable acceptance mask (Mask of Last Message) for the complete arbitration field. This allows classes of messages to be received in this object by masking some bits of the identifier.

Note The Mask of Last Message is ANDed with the Global Mask that corresponds to the incoming message.

Global Mask Short (EF06h-EE06h) XReg Reset Value: UFUUh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID20...18			1	1	1	1	1	ID28...21							
RW			R	R	R	R	R	RW							

Bit	Function
ID28...18	**Identifier (11-bit)** Mask to filter incoming messages with standard identifier.

Upper Global Mask Long (EF08h-EE08h) XReg Reset Value: UUUUh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID20...13								ID28...21							
RW								RW							

Lower Global Mask Long (EF0Ah-EE0Ah) XReg Reset Value: UUUUh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID4...0					0	0	0	ID12...5							
RW					R	R	R	RW							

Bit	Function
ID28...0	**Identifier (29-bit)** Mask to filter incoming messages with extended identifier.

Upper Mask of Last Message (EF0Ch-EE0Ch) XReg Reset Value: UUUUh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID20...18			ID17...13					ID28...21							
RW			RW					RW							

Lower Mask of Last Message (EF0Eh-EE0Eh) XReg Reset Value: UUUUh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID4...0					0	0	0	ID12...5							
RW					R	R	R	RW							

Bit	Function
ID28...0	**Identifier (29-bit)** Mask to filter the last incoming message (Nr. 15) with standard or extended identifier (as configured).

18.4 - The Message Object

The message object is the primary means of communication between CPU and CAN controller. Each of the 15 message objects uses 15 consecutive bytes (see Figure 145) and starts at an address that is a multiple of 16.

Note All message objects must be initialized by the CPU, even those which are not going to be used, before clearing the INIT bit..

Each element of the Message Control Register is made of two complementary bits.

This special mechanism allows the selective setting or resetting of specific elements (leaving others unchanged) without requiring read-modify-write cycles. None of these elements will be affected by reset.

The Table 52 shows how to use and to interpret these 2 bit-fields.

Figure 145 : Message object address map

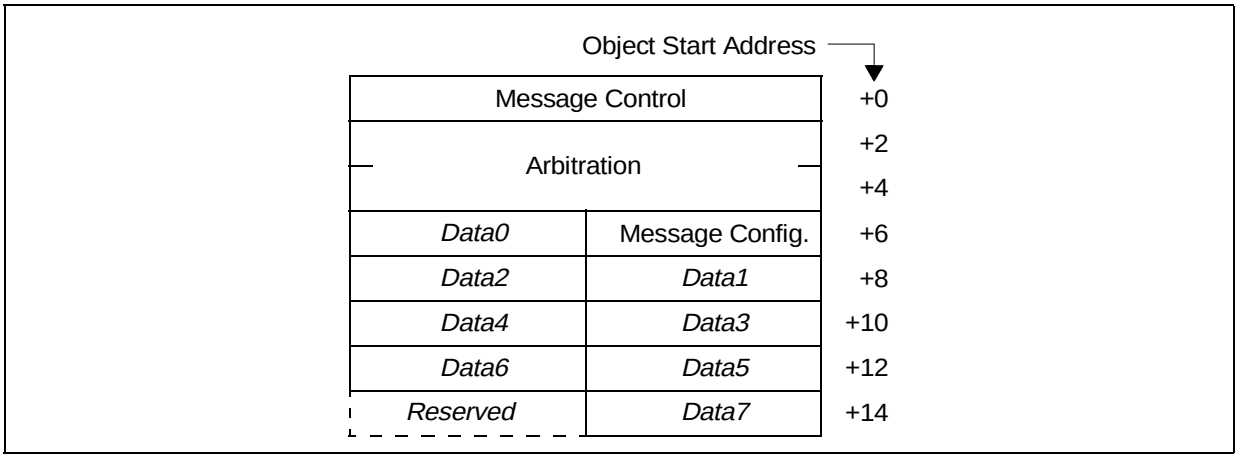


Table 52 : Functions of complementary bits of message control register

Value	Function on Write	Meaning on Read
00	Reserved	Reserved
01	Reset element	Element is reset
10	Set element	Element is set
11	Leave element unchanged	Reserved

Message Control Register (EFn0h-EEn0h) XReg Reset Value: UUUUh

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
RMTPND	**TXRQ**	**MSGLST CPUUPD**	**NEWDAT**	**MSGVAL**	TXIE	RXIE	**INTPND**
RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
INTPND	**Interrupt Pending** Indicates, if this message object has generated an interrupt request (see TXIE and RXIE), since this bit was last reset by the CPU, or not.
RXIE	**Receive Interrupt Enable** Defines, if bit INTPND is set after successful reception of a frame.
TXIE	**Transmit Interrupt Enable** Defines, if bit INTPND is set after successful transmission of a frame. [1]
MSGVAL	**Message Valid** Indicates, if the corresponding message object is valid or not. The CAN controller only operates on valid objects. Message objects can be tagged invalid, while they are changed, or if they are not used at all.
NEWDAT	**New Data** Indicates, if new data has been written into the data portion of this message object by CPU (transmit-objects) or CAN controller (receive-objects) since this bit was last reset, or not. [2]
MSGLST (Receive) CPUUPD (Transmit)	**Message Lost** (This bit applies to *receive*-objects only) Indicates that the CAN controller has stored a new message into this object, while NEWDAT was still set, i.e. the previously stored message is lost. **CPU Update** (This bit applies to *transmit*-objects only) Indicates that the corresponding message object may not be transmitted now. The CPU sets this bit in order to inhibit the transmission of a message that is currently updated, or to control the automatic response to remote requests.
TXRQ	**Transmit Request** Indicates that the transmission of this message object is requested by the CPU or via a remote frame and is not yet done. TXRQ can be disabled by CPUUPD. [1] [3]
RMTPND	**Remote Pending** (Used for transmit-objects) Indicates that the transmission of this message object has been requested by a remote node, but the data has not yet been transmitted. When RMTPND is set, the CAN controller also sets TXRQ. RMTPND and TXRQ are cleared, when the message object has been successfully transmitted.

Notes 1. In message object 15 (last message) these bits are hardwired to "0" (inactive) in order to prevent transmission of message 15.

2. When the CAN controller writes new data into the message object, unused message byte will be overwritten by non specified values. Usually the CPU will clear this bit before working on the data, and verify that the bit is still cleared once it has finished working to ensure that it has worked on a consistent set of data and not part of an old message and part of the new message.
For transmit-objects the CPU will set this bit along with clearing bit CPUUPD. This will ensure that, if the message is actually being transmitted during the time the message was being updated by the CPU, the CAN controller will not reset bit TXRQ. In this way bit TXRQ is only reset once the actual data has been transferred.

3. When the CPU requests the transmission of a receive-object, a remote frame will be sent instead of a data frame to request a remote node to send the corresponding data frame. This bit will be cleared by the CAN controller along with bit RMTPND when the message has been successfully transmitted, if bit NEWDAT has not been set. If there are several valid message objects with pending transmission request, the message with the lowest message number is transmitted first.

18.5 - Arbitration Registers

The arbitration Registers are used for acceptance filtering of incoming messages and to define the identifier of outgoing messages. A received message is stored into the valid message object with a matching identifier and DIR="0" (data frame) or DIR="1" (remote frame).

Extended frames can be stored only in message objects with XTD="1", standard frames only in message objects with XTD="0". For matching, the corresponding global mask has to be considered (in case of message object 15 also the Mask of Last Message). If a received message (data frame or remote frame) matches with more than one valid message object, it is stored into that with the lowest message number.

When the CAN controller stores a data frame, not only the data byte, but the whole identifier and the data length code are stored into the corresponding message object (standard identifiers have bit ID17...0 filled with "0"). This is implemented to keep the data byte connected with the identifier, even if arbitration mask registers are used. When the CAN controller stores a remote frame, only the data length code is stored into the corresponding message object. The identifier and the data byte remain unchanged.

There must not be more than one valid message object with a particular identifier at any time. If bit are masked by the Global Mask Registers ("don't care"), then the identifiers of the valid message objects must differ in the remaining bit which are used for acceptance filtering.

If a received data frame is stored into a message object, the identifier of this message object is updated. If some of the identifier bit are set to "don't care" by the corresponding mask register, these bit may be changed in the message object.

If a remote frame is received, the identifier in transmit-object remain unchanged, except for the last message object (which cannot start a transmission). Here, the identifier bit corresponding to the "don't care" bit of the last message object's mask may be overwritten by the incoming message.

Upper Arbitration Reg (EFn2h-EEn2h) XReg Reset Value: UUUUh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID20...18			ID17...13					ID28...21							
RW			RW					RW							

Lower Arbitration Reg (EFn4h-EEn4h) XReg Reset Value: UUUUh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID4...0					0	0	0	ID12...5							
RW					R	R	R	RW							

Bit	Function
ID28...0	**Identifier (29 bits)** Identifier of a standard message (ID28...18) or an extended message (ID28...0). For standard identifiers bit ID17...0 are "don't care".

Message Configuration and Data

The following fields hold a description of the message within this object. The data field occupies the following 8 byte positions after the Message Configuration Register.

Note There is no "don't care" option for bit XTD and DIR. So incoming frames can only match with corresponding message objects, either standard (XTD=0) or extended (XTD=1). Data frames only match with receive-objects, remote frames only match with transmit-objects.
When the CAN controller stores a data frame, it will write all the eight data byte into a message object. If the data length code was less than 8, the remaining byte of the message object will be overwritten by non specified values.

Message Configuration Register (EFn6h-EEn6h) XReg Reset Value: - - UUh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Data byte 0)								DLC				DIR	XTD	0	0
RW								RW				RW	RW	R	R

Bit	Function
XTD	**Extended Identifier** Indicates, if this message object will use an extended 29-bit identifier or a standard 11-bit identifier.
DIR	**Message Direction** DIR="1": transmit. On TXRQ, the respective message object is transmitted. On reception of a remote frame with matching identifier, the TXRQ and RMTPND bit of this message object are set. DIR="0": receive. On TXRQ, a remote frame with the identifier of this message object is transmitted. On reception of a data frame with matching identifier, that message is stored in this message object.
DLC	**Data Length Code** Valid values for the data length are 0...8.

Note The first data byte occupies the upper half of the message configuration register.

Data Area

The data area of message object n covers locations 00'EFn7h through 00'EFnEh for CAN1 and 00'EEn7h through 00'EEnEh respectively for CAN2 (locations 00'EFnFh and 00'EEnFh are reserved).

Message data for message object 15 (last message) will be written into a two-message- alternating buffer to avoid the loss of a message, if a second message has been received, before the CPU has read the first one.

Handling of Message Objects

The following diagrams summarize the necessary actions to transmit and receive data over the CAN bus. The CAN and CPU activities are described including the servicing program.

Figure 146 : CAN controller handling of message objects in transmit direction

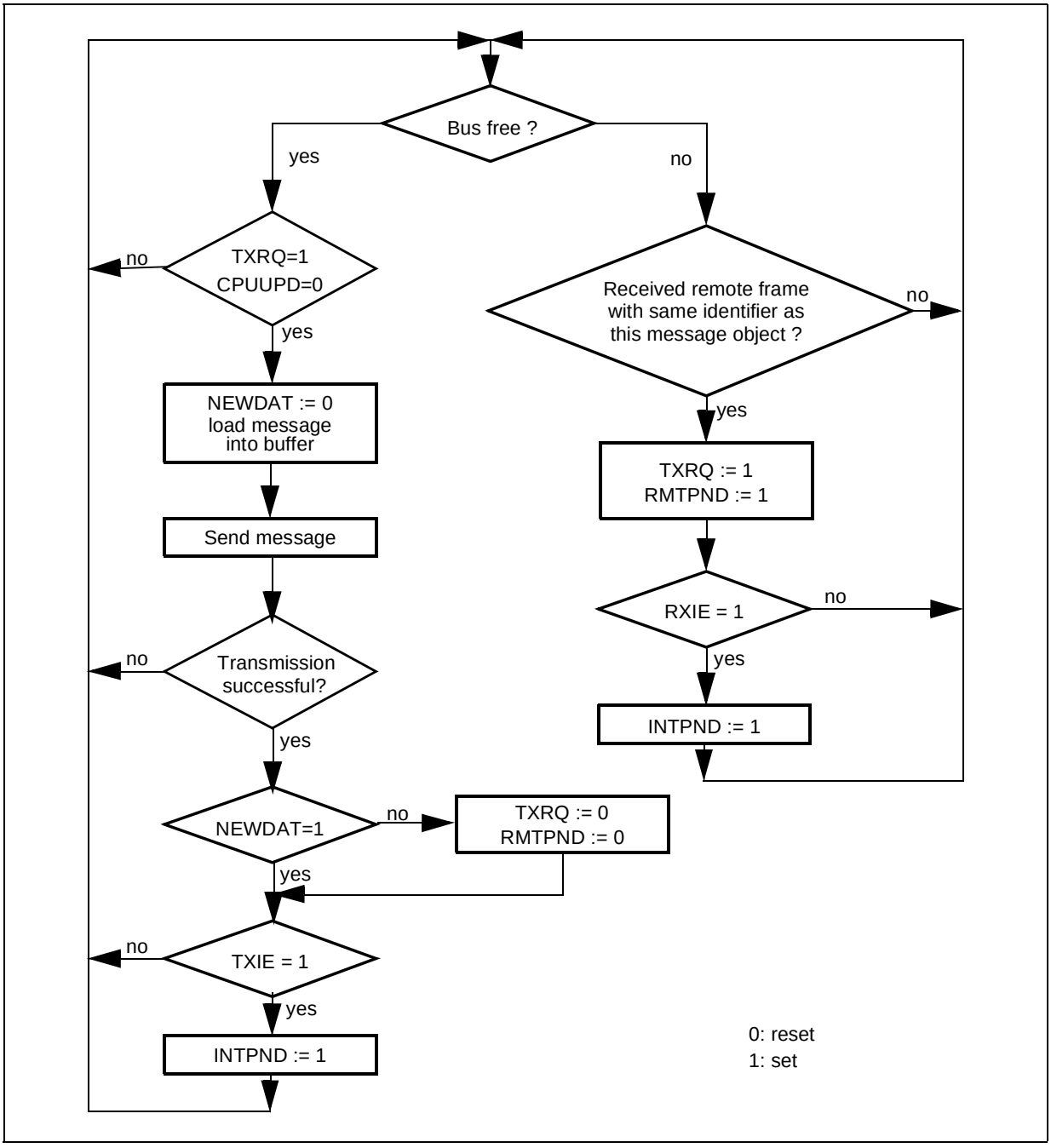


Figure 147 : CPU handling of message objects in transmit direction

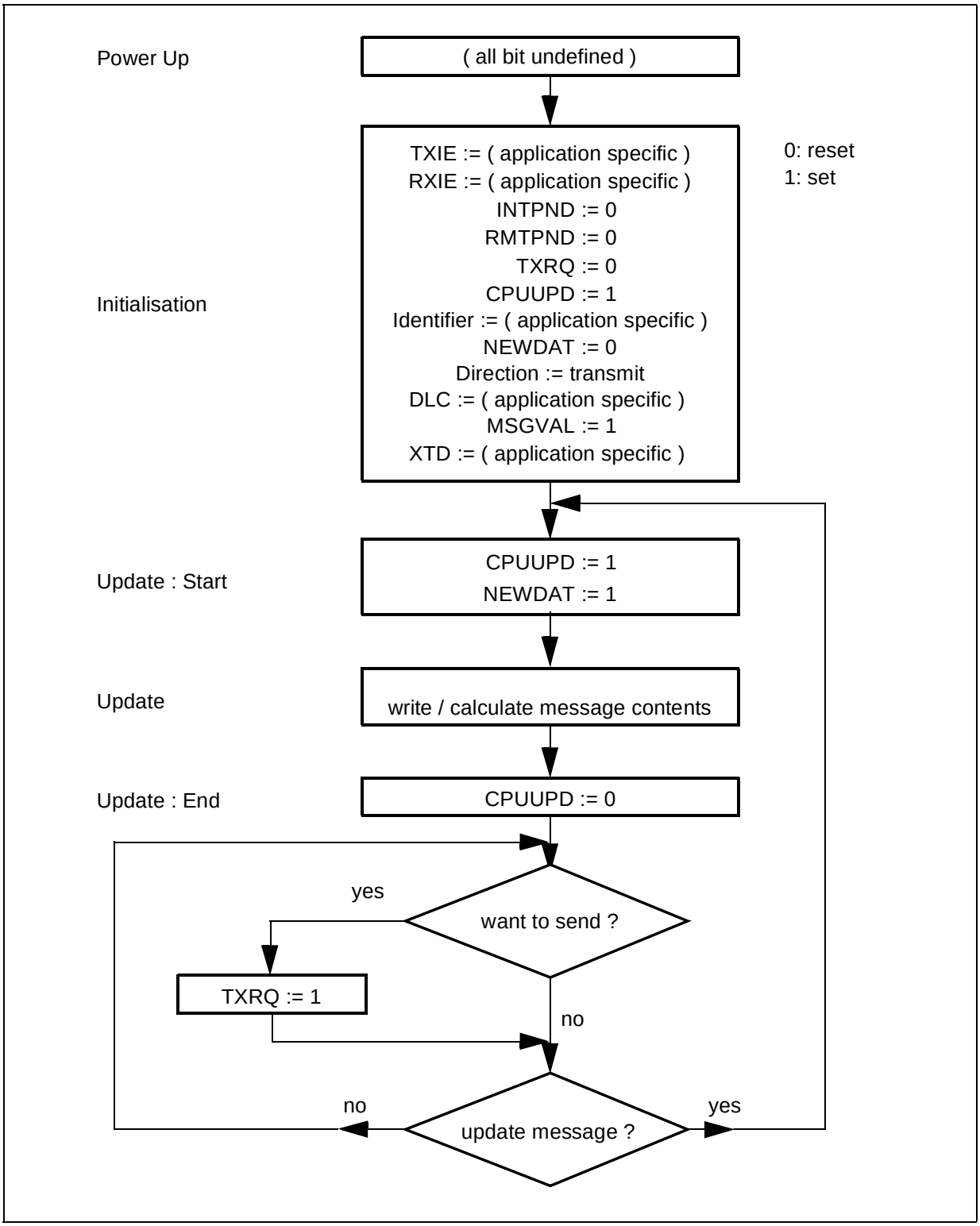


Figure 148 : CAN controller handling of message objects in receive direction

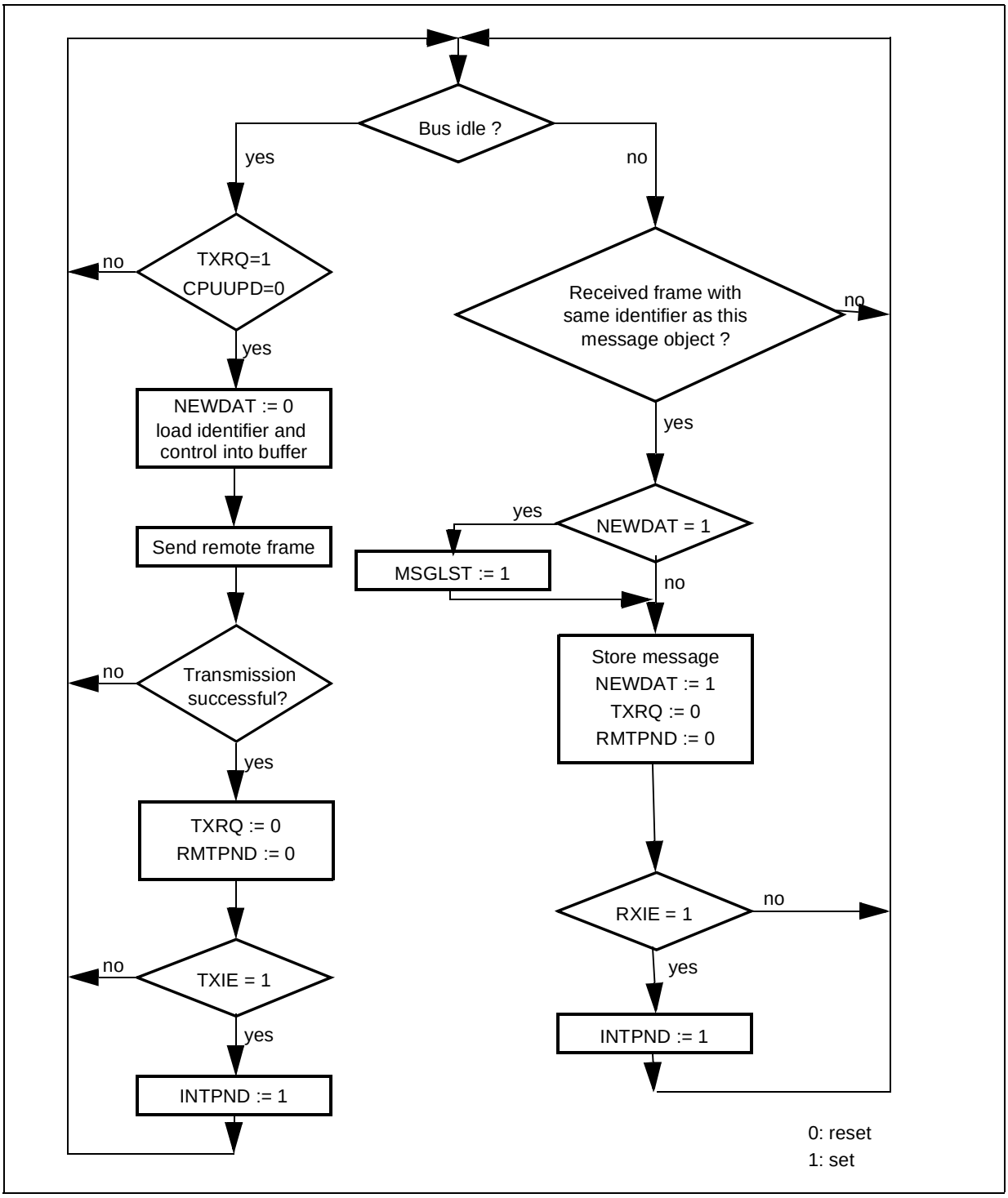


Figure 149 : CPU handling of the last message object

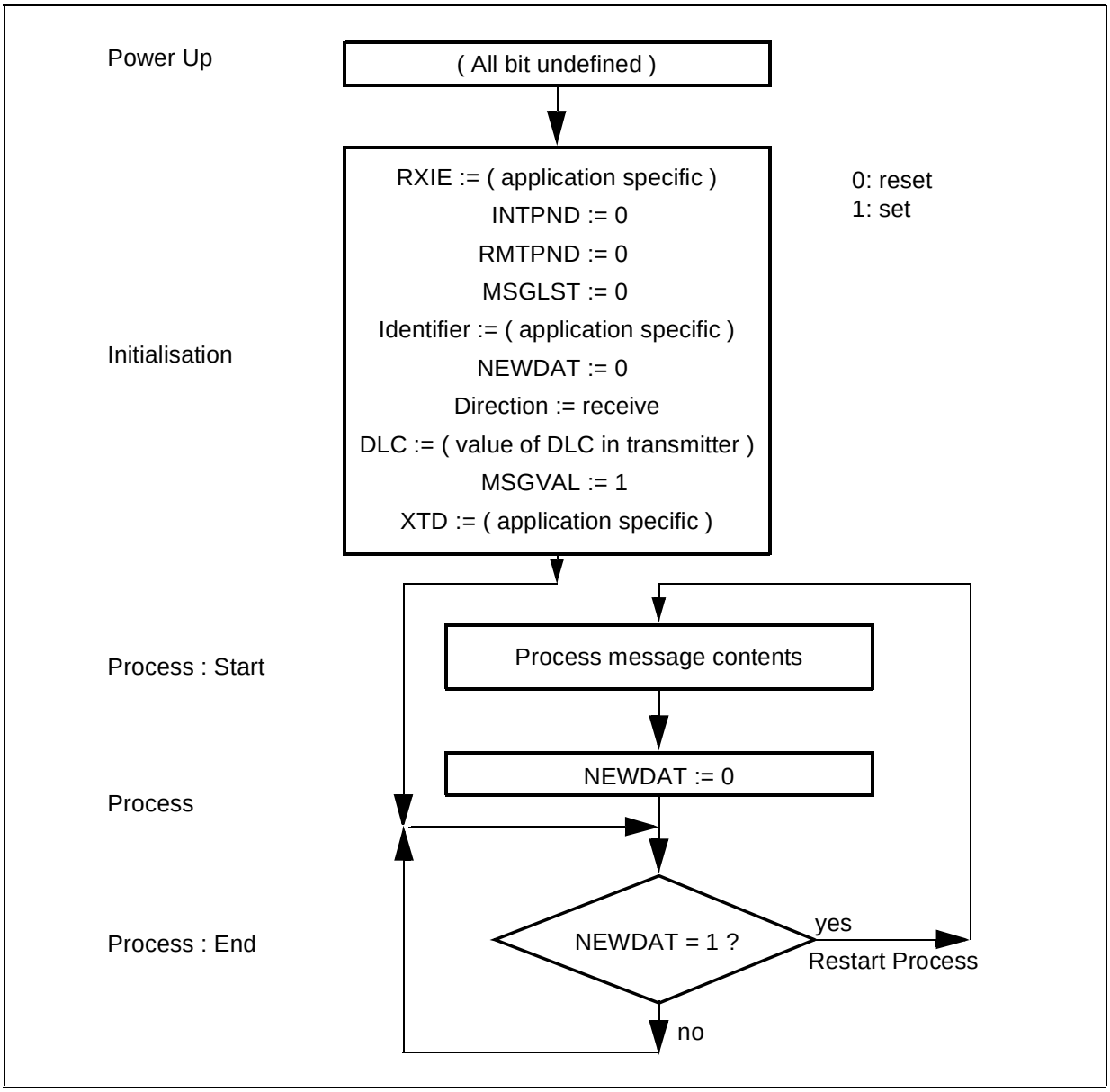


Figure 150 : CPU handling of message objects in receive direction

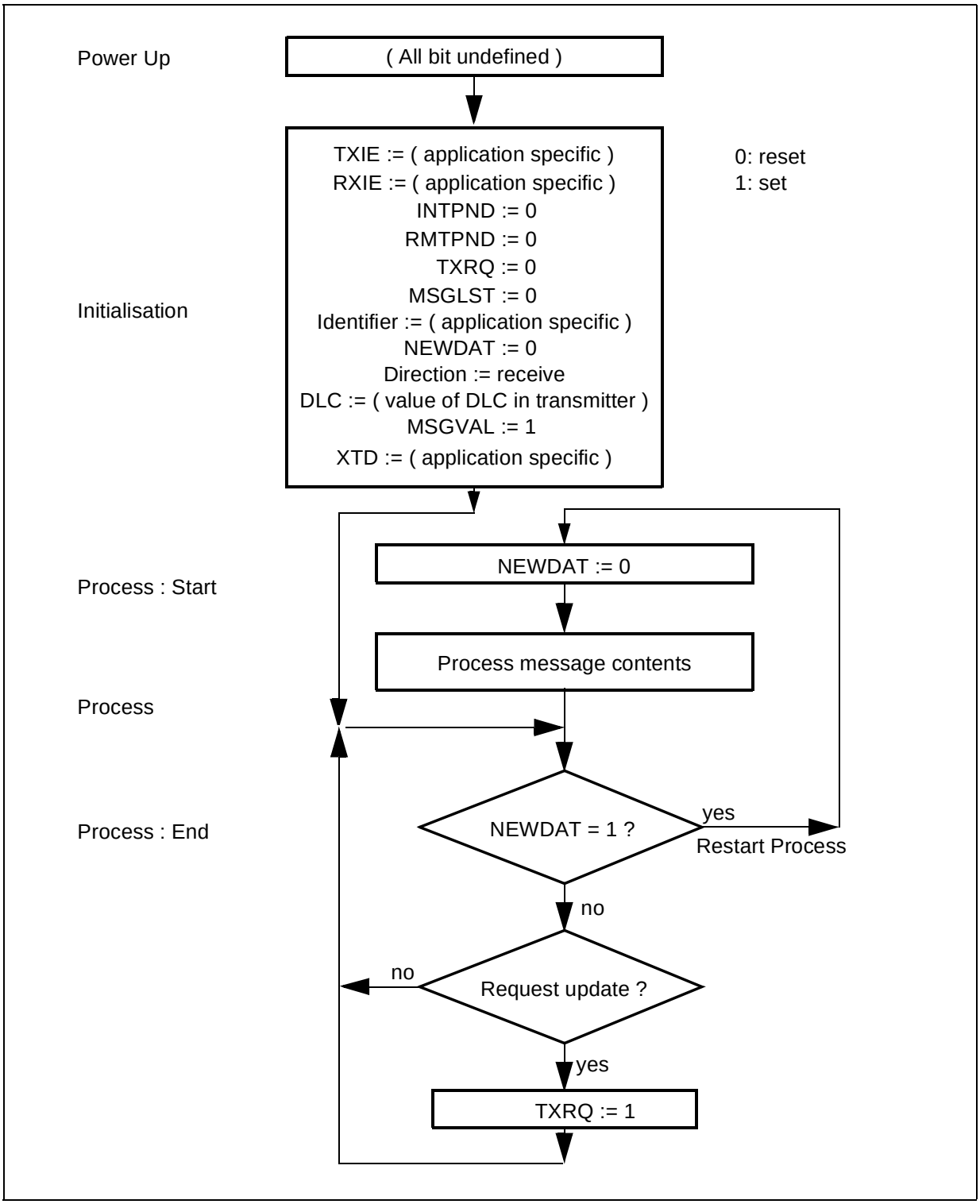


Figure 151 : Handling of the last message object's alternating buffer

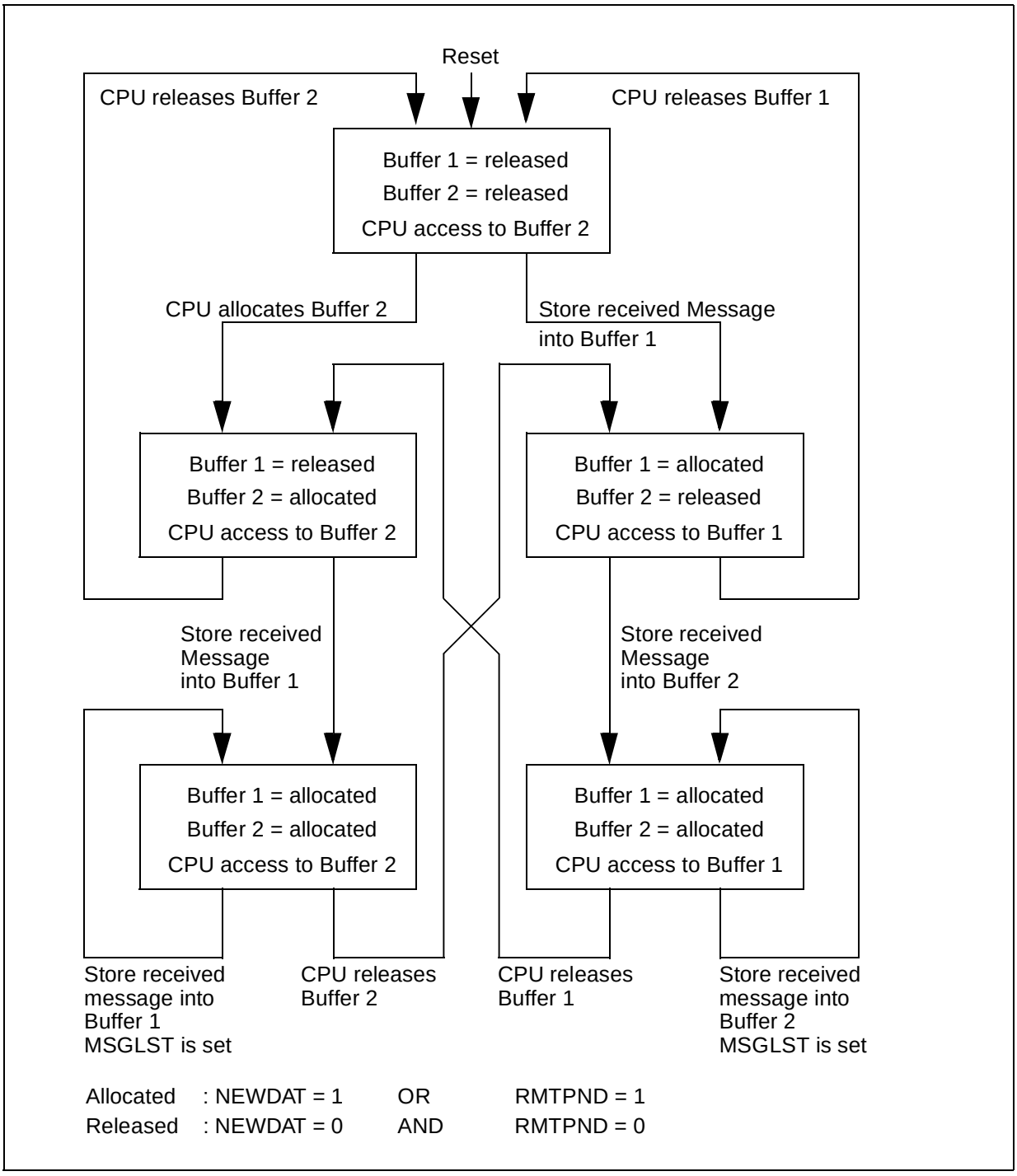


Figure 152 : CAN controller handling of bus recovery sequence

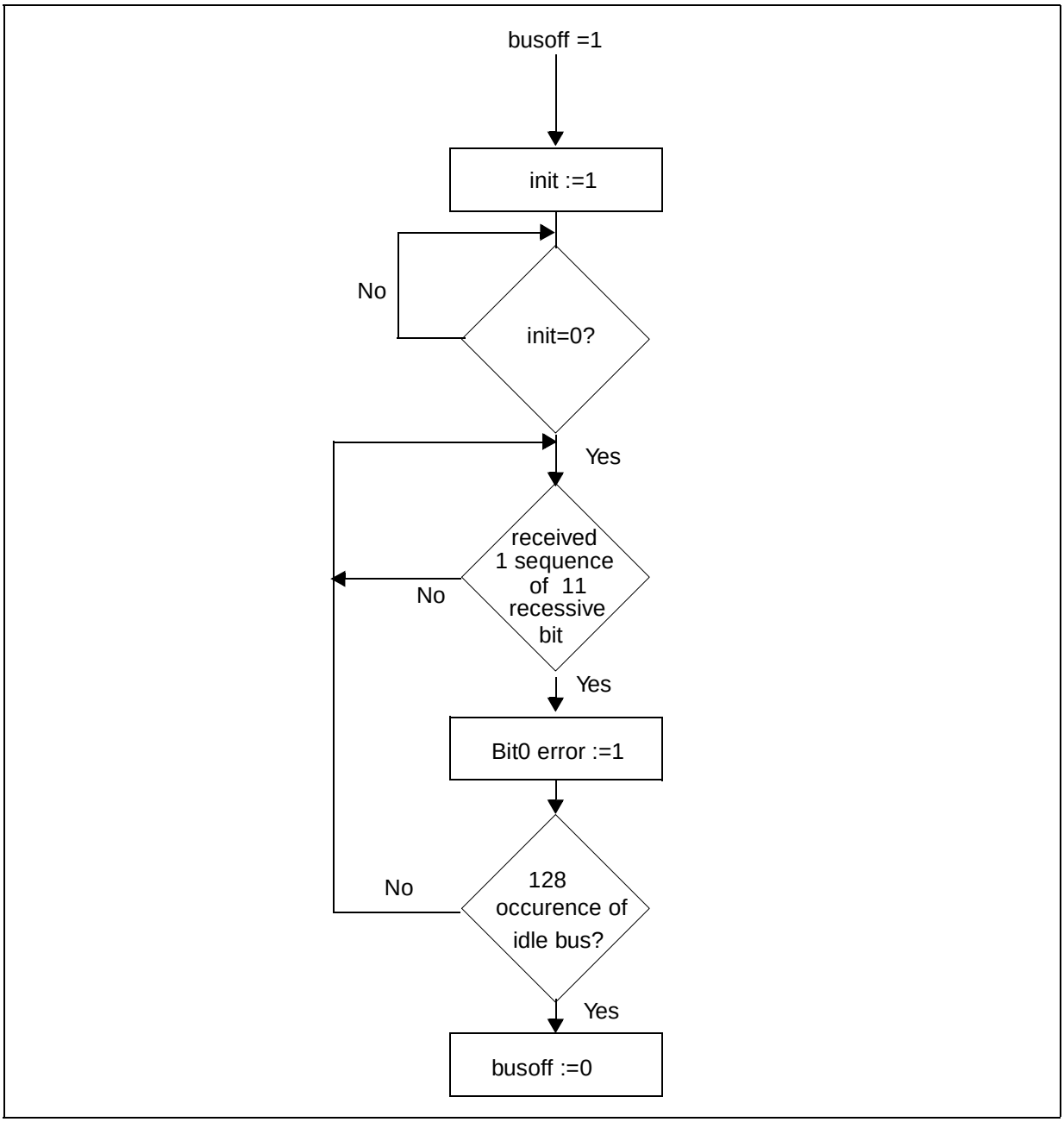
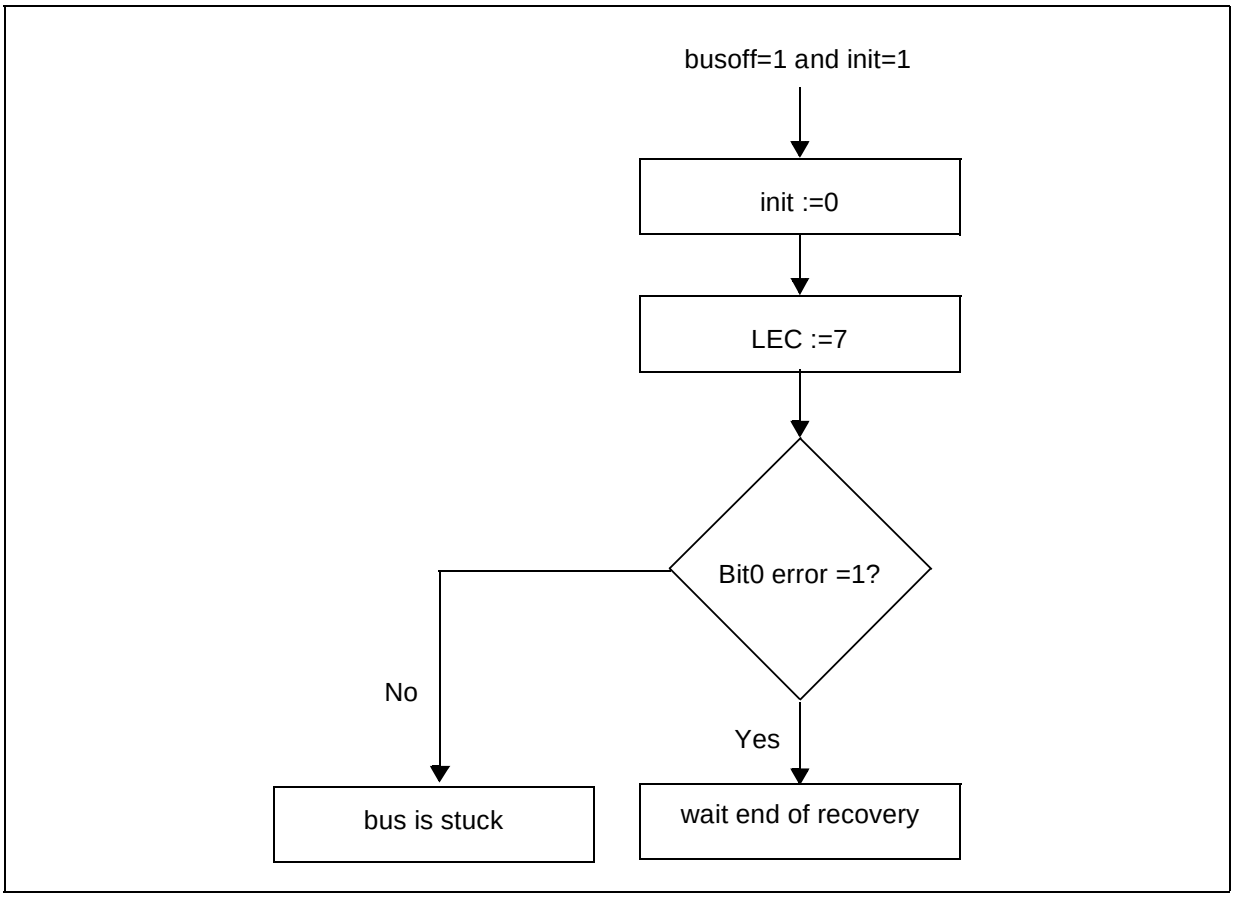


Figure 153 : CPU handling of bus recovery



18.6 - Initialization and Reset

The on-chip CAN Module is connected to the XBUS Reset signal XRESET. This signal is activated, when the ST10F280's reset input is activated, when a software reset is executed and in case of a watchdog reset. Activating the CAN Module's reset line triggers a hardware reset.

This hardware reset

– Sets the CAN_TxD output to "1" (recessive).

– Clears the error counters.

– Resets the busoff state.

– Switches the Control Register's low byte to 01h.

– Leaves the Control Register's high byte and the Interrupt Register undefined.

– Does not change the other registers including the message objects (notified as UUUU).

Note The first hardware reset after power-on leaves the unchanged registers in an undefined state. The value 01h in the Control Register's low byte prepares for software initialization.

Software Initialization

The Software Initialization is enabled by setting bit INIT in the Control Register. This can be done by the CPU via software, or automatically by the CAN controller on a hardware reset, or if the EML switches to busoff state.

While INIT is set:

– All message transfer from and to the CAN bus is stopped.

– The CAN bus output CAN_TxD is "1" (recessive).

– The control bit NEWDAT and RMTPND of the last message object are reset.

– The counters of the EML are left unchanged.

Setting bit CCE in addition, allows changing the configuration in the bit Timing Register.

To initialize the CAN Controller, the following actions are required:

– Configure the bit Timing Register (CCE required).

– Set the Global Mask Registers.

– Initialize each message object.

If a message object is not needed, it is sufficient to clear its message valid bit (MSGVAL), so it is defined as not valid. Otherwise, the whole message object has to be initialized.

After the initialization sequence has been completed, the CPU clears the INIT bit.

To change the configuration of a message object during normal operation, the CPU first clears bit MSGVAL, which defines it as not valid. When the configuration is completed, MSGVAL is set again.

Accessing the On-chip CAN Module

The CAN Modules are implemented as an X-Peripheral and are therefore accessed like an external memory or peripheral, so the registers of the CAN Module can be read and written using 16-bit or 8-bit direct or indirect MEM addressing modes. Since the XBUS, to which the CAN Module is connected, also represents the external bus, CAN accesses follow the same rules and procedures as accesses to the external bus. CAN accesses cannot be executed in parallel to external instruction fetches or data read/ writes, but are arbitrated and inserted into the external bus access stream.

Accesses to the CAN Modules use de-multiplexed addresses and a 16-bit data bus (byte accesses possible). Two wait-states give an access time of 8TCL (4 CPU clock cycles). No tristate wait-state is used.

The CANS address area starts at 00'EF00h/00'EE00h and covers 256 bytes. A dedicated hardwired XADRS/XBCON register pair selects the respective address window, so none of the programmable register pairs must be sacrificed in order to access the on-chip CAN Modules.

Locating the CAN address area to address 00'EF00h/00'EE00h in segment 0 has the advantage that the CAN Modules are accessible via data page 3, which is the 'system' data page, accessed usually through the 'system' data page pointer DPP3. In this way, the internal addresses, such like SFRs, internal RAM, and the CAN registers, are all located within the same data page and form a contiguous address space.

Power Down Mode

If the ST10F280 enters Power Down Mode, the XCLK signal will be turned off which will stop the operation of the CAN Module. Any message transfer is interrupted. In order to ensure that the CAN controller is not stopped while sending a dominant level ("0") on the CAN bus, the CPU should set bit INIT in the Control Register prior to entering Power Down Mode. The CPU can check, if a transmission is in progress by reading bit TXRQ and NEWDAT in the message objects and bit TXOK in the Control Register. After returning from Power Down Mode via hardware reset, the CAN Modules have to be reconfigured.

18.7 - CAN Application Interface

The on-chip CAN Module of the ST10F280 does not incorporate the physical layer connection to the CAN bus. This must be provided externally.

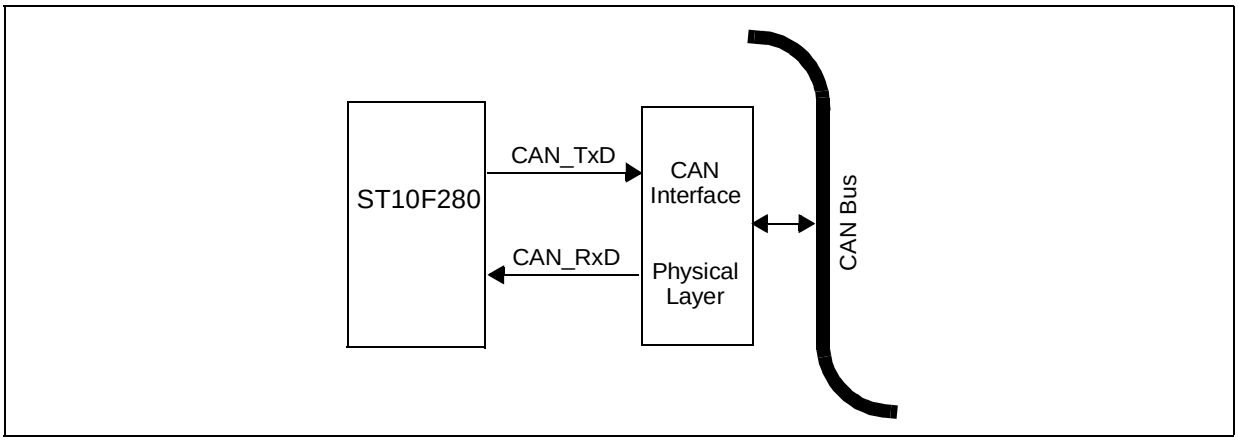
The module's CAN controller is connected to this physical layer (the CAN bus) via two signals:

CAN Signal	Port Pin	Function
CAN1 RXD	Port4.5	Receive data from the physical layer of the CAN1 bus.
CAN1 TXD	Port4.6	Transmit data to the physical layer of the CAN1 bus.
CAN2 RXD	Port4.4	Receive data from the physical layer of the CAN2 bus.
CAN2 TXD	Port4.7	Transmit data to the physical layer of the CAN2 bus.

A logic low level ("0") is interpreted as the dominant CAN bus level, a logic high level ("1") is interpreted as the recessive CAN bus level.

Note If CAN module is used, Port 4 cannot be programmed to output all the 8 segment address lines. Thus, only up to 4 or 2 segments address lines can be used, reducing the external memory space to 5 Mbytes (1 Mbyte per $\overline{CS}$ line).

Figure 154 : Connection to the CAN bus



19 - SYSTEM RESET

Table 53 : Reset event definition

Reset Source	Short-cut	Conditions
Power-on reset	PONR	Power-on
Long Hardware reset (synchronous & asynchronous)	LHWR	t_{RSTIN} > 1032 TCL
Short Hardware reset (synchronous reset)	SHWR	4 TCL < t_{RSTIN} ≤ 1032 TCL
Watchdog Timer reset	WDTR	WDT overflow
Software reset	SWR	SRST execution

System reset initializes the MCU in a predefined state. There are five ways to activate a reset state. The system start-up configuration is different for each case as shown in Table 53.

19.1 - Asynchronous Reset (Long Hardware Reset)

An asynchronous reset is triggered when RSTIN pin is pulled low while RPD pin is at low level. Then the MCU is immediately forced in reset default state. It pulls low RSTOUT pin, it cancels pending internal hold states if any, it waits for any internal access cycles to finish, it aborts external bus cycle, it switches buses (data, address and control signals) and I/O pin drivers to high-impedance, it pulls high PORT0 pins and the reset sequence starts.

Power-on reset

The asynchronous reset must be used during the power-on of the MCU. Depending on crystal frequency, the on-chip oscillator needs about 10ms to 50ms to stabilize. The logic of the MCU does not need a stabilized clock signal to detect an asynchronous reset, so it is suitable for power-on conditions. To ensure a proper reset sequence, the RSTIN pin and the RPD pin must be held at low level until the MCU clock signal is stabilized and the system configuration value on PORT0 is settled.

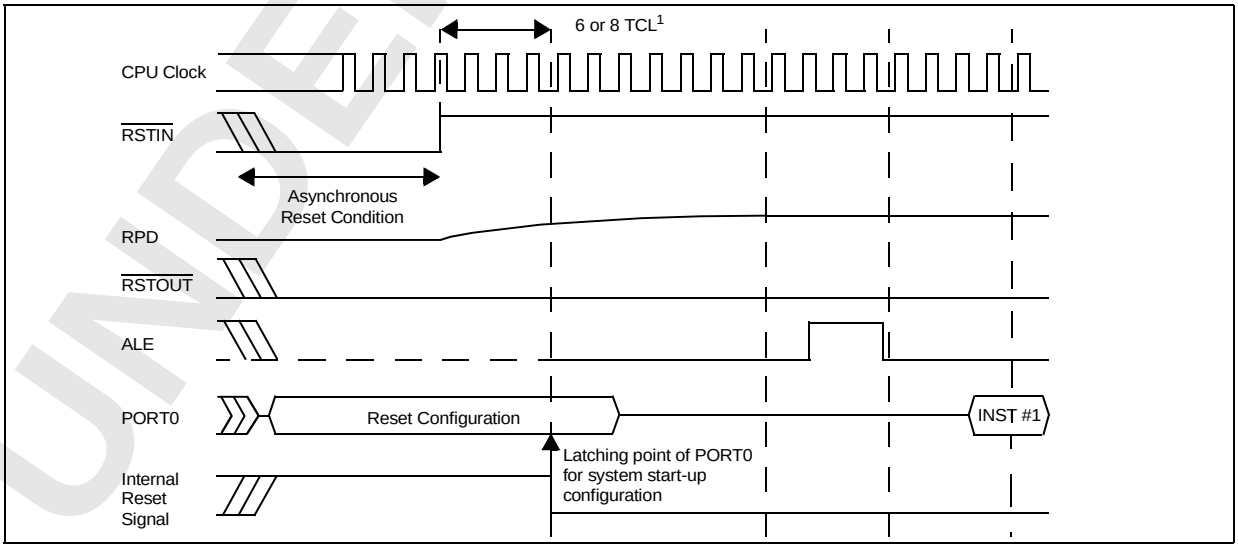
Hardware reset

The asynchronous reset must be used to recover from catastrophic situations of the application. It may be triggerred by the hardware of the application. Internal hardware logic and application circuitry are described in Reset circuitry chapter and Figures 158, 159 and 160.

Exit of asynchronous reset state

When the RSTIN pin is pulled high, the MCU restarts. The system configuration is latched from PORT0 and ALE, RD and WR/WRL pins are driven to their inactive level. The MCU starts program execution from memory location 00'0000h in code segment 0. This starting location will typically point to the general initialization routine. Timing of asynchronous reset sequence are summarized in Figure 155.

Figure 155 : Asynchronous reset timing



Note: 1. RSTIN rising edge to internal latch of PORT0 is 3CPU clock cycles (6 TCL) if the PLL is bypassed and the prescaler is on ($f_{CPU} = f_{XTAL}$ / 2), else it is 4 CPU clock cycles (8 TCL).

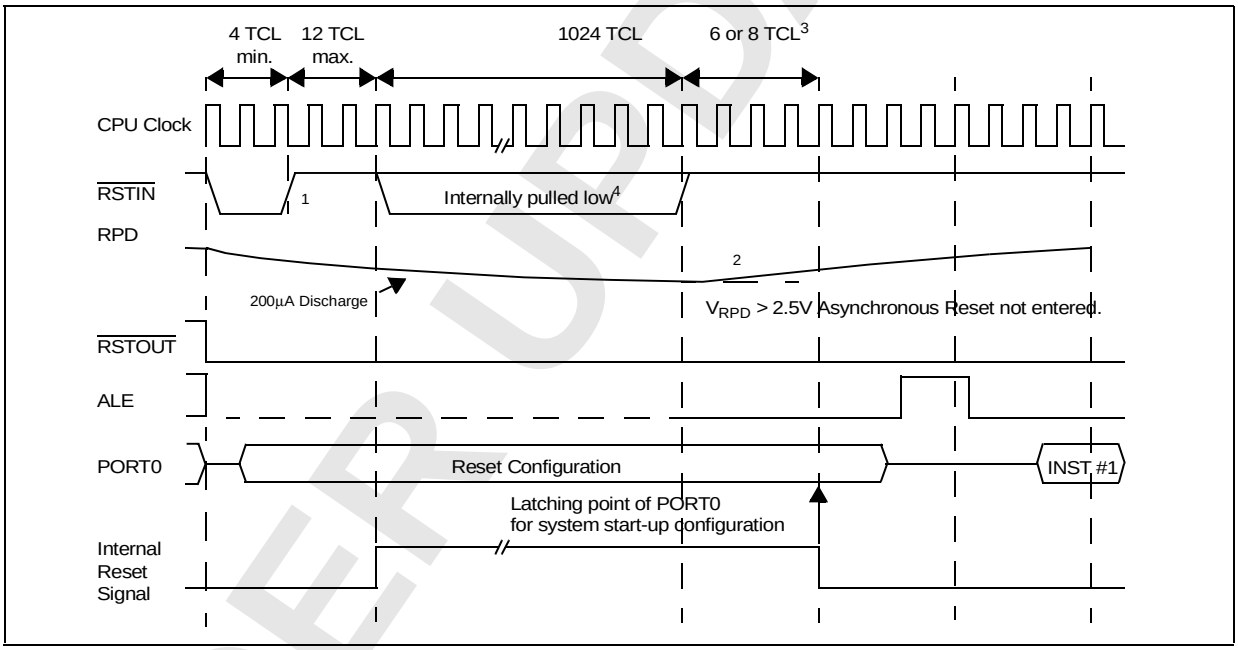
19.2 - Synchronous Reset (Warm Reset)

A synchronous reset is triggered when RSTIN pin is pulled low while RPD pin is at high level. In order to properly activate the internal reset logic of the MCU, the RSTIN pin must be held low, at least, during 4 TCL (2 periods of CPU clock). The I/O pins are set to high impedance and RSTOUT pin is driven low. After RSTIN level is detected, a short duration of 12 TCL (approximately 6 periods of CPU clock) elapes, during which pending internal hold states are cancelled and the current internal access cycle if any is completed. External bus cycle is aborted. The internal pull-down of RSTIN pin is activated if bit BDRSTEN of SYSCON register was previously set by software. This bit is always cleared on power-on or after a reset sequence.

Exit of synchronous reset state

The internal reset sequence starts for 1024 TCL (512 periods of CPU clock) and RSTIN pin level is sampled. The reset sequence is extended until RSTIN level becomes high. Then, the MCU restarts. The system configuration is latched from PORT0 and ALE, RD and WR/WRL pins are driven to their inactive level. The MCU starts program execution from memory location 00'0000h in code segment 0. This starting location will typically point to the general initialization routine. Timing of synchronous reset sequence are summarized in Figure 156 and Figure 157.

Figure 156 : Synchronous warm reset (Short low pulse on RSTIN)



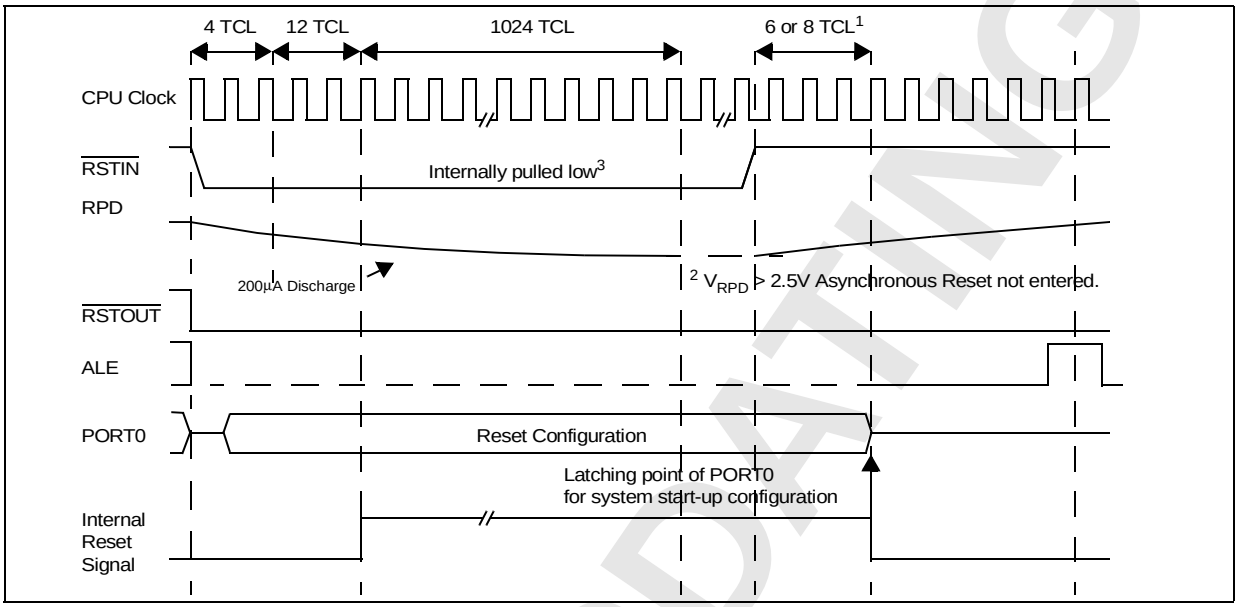
Notes: 1. RSTIN assertion can be released there.

2. If during the reset condition (RSTIN low), V_{RPD} voltage drops below the threshold voltage (about 2.5V for 5V operation), the asynchronous reset is then immediately entered.

3. RSTIN rising edge to internal latch of PORT0 is 3CPU clock cycles (6TCL) if the PLL is bypassed and the prescaler is on ($f_{CPU} = f_{XTAL} / 2$), else it is 4 CPU clock cycles (8TCL).

4. RSTIN pin is pulled low if bit BDRSTEN (bit 5 of SYSCON register) was previously set by software. Bit BDRSTEN is cleared after reset.

Figure 157 : Synchronous warm reset (Long low pulse on $\overline{\text{RSTIN}}$)



Notes: 1. $\overline{\text{RSTIN}}$ rising edge to internal latch of PORT0 is 3CPU (6TCL) clock cycles if the PLL is bypassed and the prescaler is on ($f_{CPU} = f_{XTAL} / 2$), else it is 4 CPU clock cycles (8TCL).

2. If during the reset condition ($\overline{\text{RSTIN}}$ low), V_{RPD} voltage drops below the threshold voltage (about 2.5V for 5V operation), the asynchronous reset is then immediately entered.

3. $\overline{\text{RSTIN}}$ pin is pulled low if bit BDRSTEN (bit 5 of SYSCON register) was previously set by soft-ware. Bit BDRSTEN is cleared after reset.

19.3 - Software Reset

The reset sequence can be triggered at any time using the protected instruction SRST (software reset). This instruction can be executed deliberately within a program, for example to leave bootstrap loader mode, or upon a hardware trap that reveals a system failure.

Upon execution of the SRST instruction, the internal reset sequence (1024 TCL) is started. The microcontroller behaviour is the same as for a Short Hardware reset, except that only P0.12...P0.6 bit are latched at the end of the reset sequence, while P0.5...P0.2 bit are cleared.

19.4 - Watchdog Timer Reset

When the watchdog timer is not disabled during the initialization or when it is not regularly serviced during program execution it will overflow and it will trigger the reset sequence.

Unlike hardware and software resets, the watchdog reset completes a running external bus cycle if this bus cycle either does not use $\overline{\text{READY}}$, or if $\overline{\text{READY}}$ is sampled active (low) after the programmed wait-states. When $\overline{\text{READY}}$ is sampled inactive (high) after the programmed wait-states the running external bus cycle is aborted. Then the internal reset sequence is started. At the end of the internal reset sequence (1024 TCL), only P0.12...P0.6 bit are latched, while previously latched values of P0.5...P0.2 are cleared.

19.5 -$\overline{\text{RSTOUT}}$ Pin and Bidirectional Reset

The $\overline{\text{RSTOUT}}$ pin is driven active (low level) at the beginning of any reset sequence (synchronous/ asynchronous hardware, software and watchdog timer resets). $\overline{\text{RSTOUT}}$ pin stays active low beyond the end of the initialization routine, until the protected EINIT instruction (End of Initialization) is completed.

The Bidirectional Reset function is useful when external devices require a reset signal but cannot be connected to $\overline{\text{RSTOUT}}$ pin, because $\overline{\text{RSTOUT}}$ signal lasts during initialisation. It is, for instance, the case of external memory running initialization routine before the execution of EINIT instruction.

Bidirectional reset function is enabled by setting bit 3 (BDRSTEN) in SYSCON register. It only can be enabled during the initialization routine, before EINIT instruction is completed.

When enabled, the open drain of the RSTIN pin is activated, pulling down the reset signal, for the duration of the internal reset sequence (synchronous/asynchronous hardware, software and watchdog timer resets). At the end of the internal reset sequence the pull down is released and the RSTIN pin is sampled 8 TCL periods later.

– If signal is sampled low, a hardware reset is triggered again.
– If it is sampled high, the chip exits reset state according to the running reset way (synchronous/asynchronous hardware, software and watchdog timer resets).

Note: The bidirectional reset function is disabled by any reset sequence (Bit BDRSTEN of SYSCON is cleared). To be activated again it must be enabled during the initialization routine.

19.6 - Reset Circuitry

The internal reset circuitry is described in Figure 158.

An internal pull-up resistor is implemented on RSTIN pin. (50kΩ minimum, to 250kΩ maximum). The minimum reset time must be calculated using the lowest value. In addition, a programmable pull-down (bit BDRSTEN of SYSCON register) drives the RSTIN pin according to the internal reset state as explained in Section 19.5 - -RSTOUT Pin and Bidirectional Reset.

The RSTOUT pin provides a signals to the application as described in Section 19.5.

A weak internal pull-down is connected to the RPD pin to discharge external capacitor to Vss at a rate of 100µA to 200µA. This Pull-down is turned on when RSTIN pin is low

If bit PWDCFG of SYSCON register is set, an internal pull-up resistor is activated at the end of the reset sequence. This pull-up charges the capacitor connected to RPD pin.

If Bidirectional Reset function is not used, the simplest way to reset ST10F280 is to connect external components as shown in Figure 159. It works with reset from application (hardware or manual) and with power-on. The value of C1 capacitor, connected on RSTIN pin with internal pull-up resistor (50kΩ to 250kΩ), must lead to a charging time long enought to let the internal or external oscillator and / or the on-chip PLL to stabilize.

The R0-C0 components on RPD pin are mainly implemented to provide a time delay to exit Power down mode (see Chapter 20 - Power Reduction Modes). Nervertheless, they drive RPD pin level during resets and they lead to different reset modes as explained hereafter. On power-on, C0 is totaly discharged, a low level on RPD pin forces an asynchronous hardware reset. C0 capacitor starts to charge throught R0 and at the end of reset sequence ST10F280 restarts. RPD pin threshold is typically 2.5V.

Depending on the delay of the next applied reset, the MCU can enter a synchronous reset or an asynchronous reset. If RPD pin is below 2.5V an asynchronous reset starts, if RPD pin is above 2.5V a synchronous reset starts. (see Section 19.1 - Asynchronous Reset (Long Hardware Reset) and Section 19.2 - Synchronous Reset (Warm Reset)).

Note that an internal pull-down is connected to RPD pin and can drive a 100µA to 200µA current. This Pull-down is turned on when RSTIN pin is low.

In order to properly use the Bidirectional reset features, the schematic (or equivalent) of Figure 160 must be implemented. R1-C1 only work for power-on or manual reset in the same way as explained previously. D1 diode brings a faster discharge of C1 capacitor at power-off during repetitive switch-on / switch-off sequences. D2 diode performs an OR-wired connection, it can be replaced with an open drain buffer. R2 resistor may be added to increase the pull-up current to the open drain in order to get a faster rise time on RSTIN pin when bidirectional function is activated.

The start-up configurations and some system features are selected on reset sequences as described in Table 54 and Table 55.

Table 54 describes what is the system configuration latched on PORT0 in the five different reset ways. Table 55 summarizes the bit state of PORT0 latched in RP0H, SYSCON, BUSCON0 registers.

Figure 158 : Internal (simplified) reset circuitry

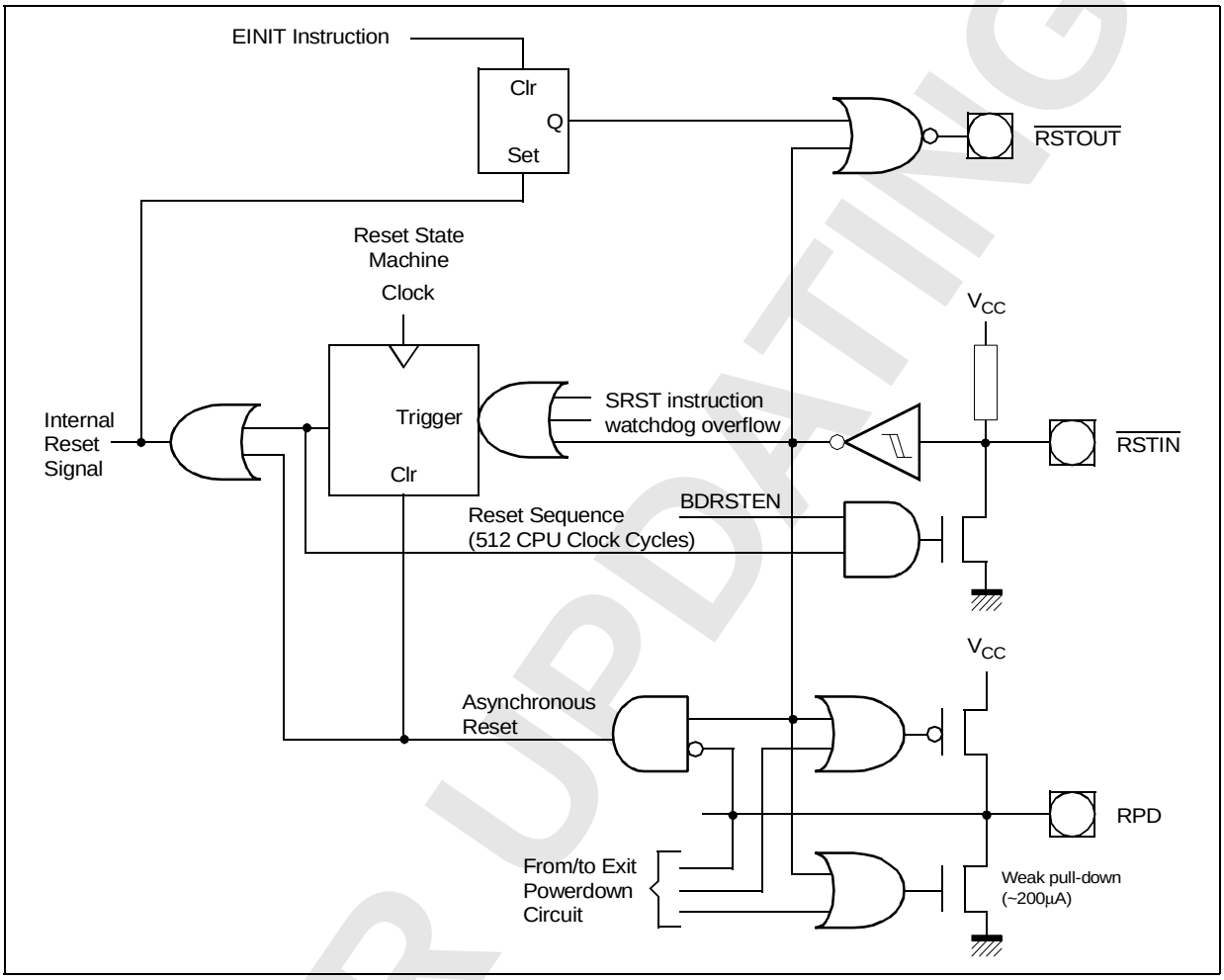


Figure 159 : Minimum external reset circuitry

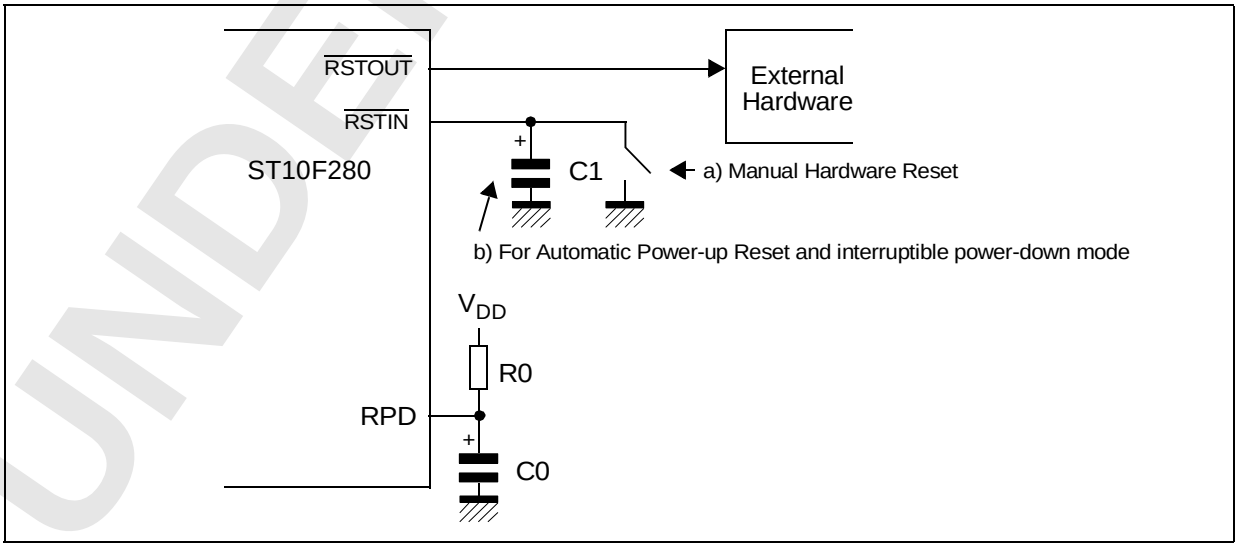
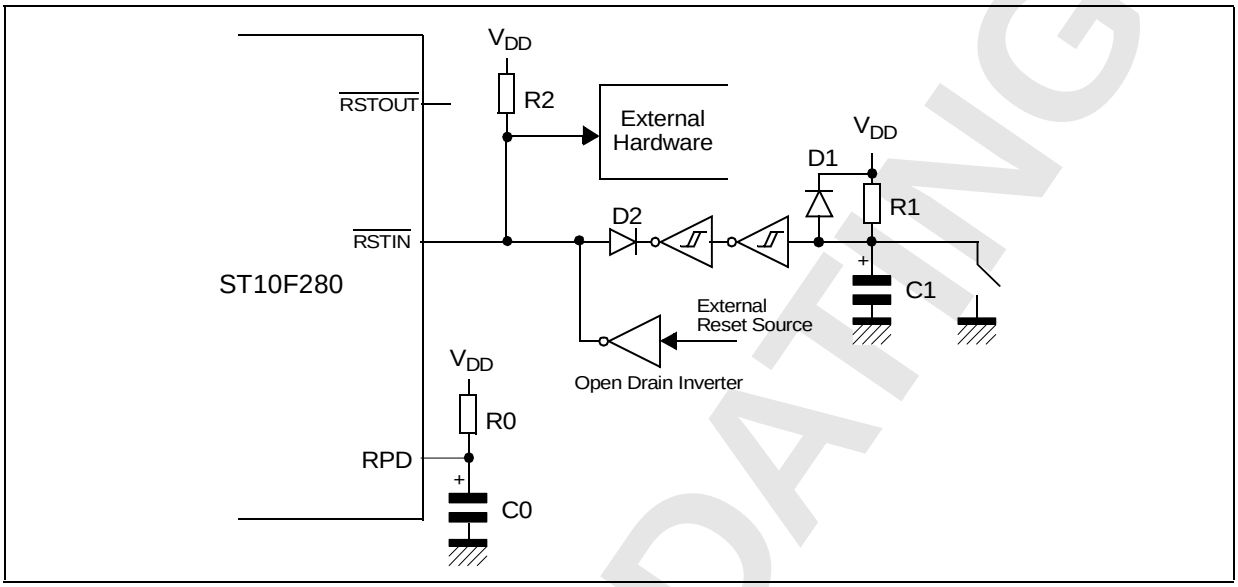


Figure 160 : External reset hardware circuitry



19.7 - Pins After Reset

After the reset sequence the different groups of pins of the ST10F280 are activated in different ways depending on their function. Bus and control signals are activated immediately after the reset sequence according to the configuration latched from PORT0, so either external accesses can take place or the external control signals are inactive.

The general purpose I/O pins remain in input mode (high impedance) until reprogrammed via software (see Figures 156 and 157). The RSTOUT pin remains active (low) until the end of the initialization routine (see Section 19.5 - -RSTOUT Pin and Bidirectional Reset).

Reset output pin

The RSTOUT pin generates a reset signal for the system components besides the controller itself. RSTOUT is driven active (low) at the beginning 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 Figures 156 and 157). This allows the complete configuration of the controller including its on-chip peripheral units before releasing the reset signal for the external peripherals of the system.

Note RSTOUT will float as long as pins P0L.0 and P0L.1 select emulation mode or adapt mode.

Watchdog timer operation after reset

The watchdog timer starts running after the internal reset has completed. It will be clocked with the internal system clock divided by 2, 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 is triggered, a watchdog timer overflow, the WDTR (Watchdog Timer Reset Indication) flag in register WDTCON is 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.

Reset values for the ST10F280 registers

During the reset sequence the registers of the ST10F280 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)

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)

The internal ram after reset

The contents of the internal RAM are not affected by a system reset. However, after a power-on reset, the contents of the internal RAM are undefined. This implies that the GPRs (R15...R0) and the PEC source and destination pointers (SRCP7...SRCP0, DSTP7...DSTP0) which are mapped into the internal RAM are also unchanged after a warm reset, software reset or watchdog reset, but are undefined after a power-on reset.

Ports and external bus configuration during reset

During the internal reset sequence all of the ST10F280'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 ST10F280 and external devices will not try to drive the same pin to different levels. Pin ALE is held low through an internal pull-down, and pins $\overline{RD}$ and $\overline{WR}$ are held high through internal pull-ups. Also the pins selected for $\overline{CS}$ output will be pulled high.

The registers SYSCON and BUSCON0 are initialized according to the configuration selected via PORT0. The reset configurations are summarized in Table 54 and Table 55.

When an external start is selected (pin $\overline{EA}$='0'):

– The Bus-Type field (BTYP) in register BUSCON0 is initialized according to P0L.7 and P0L.6.

– Bit BUSACT0 in register BUSCON0 is set to '1'.

– Bit ALECTL0 in register BUSCON0 is set to '1'.

– Bit ROMEN in register SYSCON will be cleared to '0'.

– Bit BYTDIS in register SYSCON is set according to the data bus width.

When an internal start is selected (pin EA='1'):

– Register BUSCON0 is cleared to 0000h.

– Bit ROMEN in register SYSCON will be set to '1'.

– Bit BYTDIS in register SYSCON is cleared, and. $\overline{BHE}$ is disabled.

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, Port4, Port6 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. Port4 will output the selected number of segment address lines (all zero after reset) and Port6 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 Kbytes are to be performed, segmentation may be disabled.

When the on-chip bootstrap loader is activated and sampled during reset, pin TxD0 (alternate function of P3.10) is switched to output mode after the reception of the zero byte. All other pins remain in the high-impedance state until they are changed by software or peripheral operation.

Application-specific initialization routine

After the internal reset condition is removed the ST10F280 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.

Note When the Bootstrap Loader Mode is activated and sampled during a hardware reset the ST10F280 does not fetch instructions from location 00'0000h but it waits data via serial interface ASC0.

If single chip mode is selected during reset, the first instruction is fetched from the internal Flash. Otherwise it is fetched from external memory. When internal Flash access is enabled after reset in single chip mode (Bit ROMEN='1' in register SYSCON), the software initialization routine may enable and configure the external bus interface before the execution of the EINIT instruction. When external access is enabled after reset, it may be desirable to reconfigure the external bus characteristics, because the SYSCON register is initialized during reset to the slowest possible memory configuration.

To decrease the number of instructions required to initialize the ST10F280, 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 ($\overline{NMI}$) 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 $\overline{RSTOUT}$ 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.

19.7.1 - System Start-up Configuration

Although most programmable features 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 (for example internal or external start selected via EA).

These selections are made during reset by the pins of PORT0 which are read at the end of the internal reset sequence. During reset, internal pull-up devices are active on the PORT0 lines so their input level is high, if the respective pin is left open or is low, or if the respective pin is connected to an external pull-down device. With the coding of the selections, as shown below, in many cases the default option, (high level), can be used.

The value on the upper byte of PORT0 (P0H) is latched into register RP0H upon reset, the value on the lower byte (P0L) directly influences the BUSCON0 register (bus mode) or the internal control logic of the ST10F280.

Not all Port0 bits are latched after the end of an internal reset. Depending on the reset type, different bits are latched.

When $\overline{\text{RSTIN}}$ goes active, the PORT0 configuration input pins are not transparent for the first 1024 TCL.

After that time only, the PORT0 pins are transparent and will be latched when internal reset signal becomes inactive (see Figures 155, 156 and 157). To avoid unexpected behavior, the level of the PORT0 configuration input pins should not change while PORT0 is transparent.

Table 54 : Port0 latched configuration for the different resets

X Pin is sampled - Pin is not sampled	PORT0															
	Clock Options			Segm. Addr. Lines		Chip Selects		WR config.	Bus Type		Reserved	BSL	Reserved	Reserved	Adapt Mode	Emu Mode
Sample event	**P0H.7**	**P0H.6**	**P0H.5**	**P0H.4**	**P0H.3**	**P0H.2**	**P0H.1**	**P0H.0**	**P0L.7**	**P0L.6**	**P0L.5**	**P0L.4**	**P0L.3**	**P0L.2**	**P0L.1**	**P0L.0**
Software Reset	-	-	-	X	X	X	X	X	X	X	-	-	-	-	-	-
Watchdog Reset	-	-	-	X	X	X	X	X	X	X	-	-	-	-	-	-
Short Hardware Reset	-	-	-	X	X	X	X	X	X	X	X	X	X	X	X	X
Long Hardware Reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Power-On Reset	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Table 55 : PORT0 bit latched into the different registers after reset

PORT0 bit nber	h7	h6	h5	h4	h3	h2	h1	h0	I7	I6	I5	I4	I3	I2	I1	I0
PORT0 bit Name	CLKCFG	CLKCFG	CLKCFG	SALSEL	SALSEL	CSSEL	CSSEL	WRC	BUSTYP	BUSTYP	R	BSL	R	R	ADP	EMU
RP0H [2]	X [1]	X [1]	X [1]	X [1]	X [1]	X [1]	X [1]	X [1]	CLKCFG	CLKCFG	CLKCFG	SALSEL	SALSEL	CSSEL	CSSEL	WRC
SYSCON	X [1]	X [1]	X [1]	X [1]	X [1]	X [1]	BYTDIS [3]	X [1]	WRCFG [3]	X [1]	X [1]	X [1]	X [1]	X [1]	X [1]	X [1]
BUSCON0	X [1]	X [1]	X [1]	X [1]	-	BUS ACT0 [4]	ALE CTL0 [4]	-	BTYP	BTYP	X [1]	X [1]	X [1]	X [1]	X [1]	X [1]
Internal Logic	To Clock Generator			To Port 4 Logic		To Port 6 Logic		X [1]	X [1]	X [1]	X [1]	Internal	X [1]	X [1]	Internal	Internal

Notes: 1. Not latched from PORT0.

2. Only RP0H low byte is used and the bit-fields are latched from PORT0 high byte to RP0H low byte.

3. Indirectly depend on PORT0.

4. Bits set if $\overline{EA}$ pin is 1.

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	**CLKCFG**			**SALSEL**		**CSSEL**		**WRC**
								R [1]			R [2]		R [2]		R [2]

Bit	Function
WRC	Write Configuration Control '0': Pins $\overline{WR}$ acts as $\overline{WRL}$, pin $\overline{BHE}$ acts as $\overline{WRH}$ '1': Pins $\overline{WR}$ and $\overline{BHE}$ retain their normal function
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 pull-downs)
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 pull-downs)
CLKCFG	P0H.7-5 $f_{CPU} = f_{XTAL} \times F$ — Notes 111 $f_{XTAL} \times 4$ — Default configuration 110 $f_{XTAL} \times 3$ 101 $f_{XTAL} \times 2$ 100 $f_{XTAL} \times 5$ 011 f_{XTAL} — Direct drive [4] 010 $f_{XTAL} \times 10$ 001 $f_{XTAL} \times 0.5$ — CPU clock via prescaler [3] 000 $f_{XTAL} \times 2.5$

Notes: 1. RP0H.7 to RP0H.5 bits are loaded only during a long hardware reset. As pull-up resistors are on each P0H port pins during reset, RP0H default value is "FFh".

2. These bits are set according to Port0 configuration during any reset sequence.

3. Using an external crystal with the internal oscillator of 25MHz maximum frequency (internal serial resistor of the crystal must be less than 40Ω). However, higher frequencies can be applied with an external clock source, the maximum depends on the duty cycle of the external clock signal (Refer to product data sheet).

4. The external clock input range refers to a CPU clock range of 1 to 40MHz. The PLL free-running frequency is from 2 to 10MHz. If the PLL factor or the input clock frequency is changed when Port0 os transparent, the PLL needs a synchronization time to lock (typical value is 500µs).

Pins controlling the operation of the internal logic and the reserved pins are evaluated only during a hardware triggered reset sequence.

The pins that influence the configuration of the ST10F280 are evaluated during any reset sequence, even during software and watchdog timer triggered resets.

The configuration via P0H is latched in register RP0H for subsequent evaluation by software. Register RP0H is described in chapter "The External Bus Interface".

Note The reserved pins, named R in the row "Port0 Bit Name" of Table 55, must remain high during reset in order to ensure proper operation of the ST10F280. The load on those pins must be small enough for the internal pull-up device to keep their level high, or external pull-up devices must ensure the high level.

The following describes the different selections that are offered for reset configuration.

The default modes refer to pins at high level, without external pull-down devices connected.

Emulation mode

When low during reset, pin P0L.0 (EMU) selects the Emulation Mode. This mode allows the access to integrated XBus peripherals via the external bus interface pins in application specific version of the ST10F280. In addition also the RSTOUT pin floats to tristate rather to be driven low. When the emulation mode is latched the CLKOUT output is automatically enabled. This mode is used for special emulator purposes and is not used in basic ST10F280 devices, so in this case P0L.0 should be held high.

Default: Emulation Mode is off.

Adapt mode

Pin P0L.1 (ADP) selects the Adapt Mode, when low during reset. In this mode the ST10F280 goes into a passive state, which is similar to its state during reset.

The pins of the ST10F280 float to tristate or are deactivated via internal pull-up/pull-down devices, as described for the reset state. In addition also the RSTOUT pin floats to tristate rather than be driven low, and the on-chip oscillator is switched off.

This mode allows switching a ST10F280 that is mounted to a board virtually off, so an emulator may control the board's circuitry, even though the original ST10F280 remains in its place. The original ST10F280 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).

Bootstrap loader mode

Pin P0L.4 (BSL) activates the on-chip bootstrap loader, when low during hardware reset. The bootstrap loader allows moving the start code into the internal RAM of the ST10F280 via the serial interface ASC0. The MCU will remain in bootstrap loader mode until a hardware reset with P0L.4 high or a software reset. The bootstrap loader acknowledge byte is D5h.

Default: The ST10F280 starts fetching code from location 00'0000h, the bootstrap loader is off.

External bus type

Pins P0L.7 and P0L.6 (BUSTYP) select the external bus type during reset, if an external start is selected via pin EA. This allows the configuration of the external bus interface of the ST10F280 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 de-multiplexed). This bit-field may be changed via software after reset, if required.

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

PORT0 and PORT1 are automatically switched to the selected bus mode. In multiplexed bus modes PORT0 drives both the 16-bit intra-segment address and the output data, while PORT1 remains in high impedance state as long as no de-multiplexed bus is selected via one of the BUSCON registers. In de-multiplexed bus modes PORT1 drives the 16-bit intra-segment address, while PORT0 or P0L (according to the selected data bus width) drives the output data.
For a 16-bit data bus $\overline{BHE}$ is automatically enabled, for an 8-bit data bus $\overline{BHE}$ is disabled via Bit BYTDIS in register SYSCON.

Default: 16-bit data bus with multiplexed addresses.

Note If an internal start is selected via pin $\overline{EA}$, these two pins are disregarded and bit-field BTYP of register BUSCON0 is cleared.

Write configuration

Pin P0H.0 (WRC) selects the initial operation of the control pins $\overline{WR}$ and $\overline{BHE}$ during reset. When high, this pin selects the standard function, which is $\overline{WR}$ control and $\overline{BHE}$. When low, it selects the alternate configuration, $\overline{WRH}$ and $\overline{WRL}$. Thus even the first access after a reset can go to a memory controlled via $\overline{WRH}$ and $\overline{WRL}$. This bit is latched in register RP0H and its inverted value is copied into bit WRCFG in register SYSCON.

Default: Standard function ($\overline{WR}$ control and $\overline{BHE}$).

Chip select lines

Pins P0H.2 and P0H.1 (CSSEL) define the number of active chip select signals during reset.

This allows to select which pins of Port6 drive external $\overline{CS}$ signals and which are used for general purpose I/O. The two bits are latched in register RP0H.

Default: All 5 chip select lines active ($\overline{CS4}$...$\overline{CS0}$).

CSSEL	Chip Select Lines	Note
1 1	Five: $\overline{CS4}$...$\overline{CS0}$	Default without pull-downs
1 0	None	Port6 pins free for I/O
0 1	Two: $\overline{CS1}$...$\overline{CS0}$	
0 0	Three: $\overline{CS2}$...$\overline{CS0}$	

Note The selected number of $\overline{CS}$ signals cannot be changed via software after reset.

Segment address lines

Pins P0H.4 and P0H.3 (SALSEL) define the number of active segment address lines during reset. This determines which pins of Port4 are used as address line or as I/O line. 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 Port4 are automatically switched to address output mode.

SALSEL	Segment Address Lines	Directly accessible Address Space
1 1	Two: A17...A16	256 Kbytes (Default without pull-downs)
1 0	Eight: A23...A16	16 Mbytes (Maximum)
0 1	None	64 Kbytes (Minimum)
0 0	Four: A19...A16	1 Mbyte

Even if not all segment address lines are enabled on Port4, the ST10F280 internally uses its complete 24-bit addressing mechanism.

This allows the restriction of the width of the effective address bus, while still deriving $\overline{CS}$ signals from the complete addresses.

Default: 2-bit segment address (A17...A16) allowing access to 256 Kbytes.

Note The selected number of segment address lines cannot be changed via software after reset.

Clock generation control

Pins P0H.7, P0H.6 and P0H.5 (CLKCFG) select the clock generation mode (on-chip PLL) during reset. The oscillator clock either directly feeds the CPU and peripherals (direct drive) or it is fed to the on-chip PLL which then provides the CPU clock signal (selectable multiple of the oscillator frequency). These bits are latched in register RP0H.

Table 56 : CPU frequency generation

P0H.7	P0H.6	P0H.5	CPU Frequency $f_{CPU} = f_{XTAL} \times F$	External Clock Input Range[1]	Notes
1	1	1	f_{XTAL} x 4	2.5 to 10 MHz	Default configuration
1	1	0	f_{XTAL} x 3	3.33 to 13.33 MHz	
1	0	1	f_{XTAL} x 2	5 to 20 MHz	
1	0	0	f_{XTAL} x 5	2 to 8 MHz	
0	1	1	f_{XTAL} x 1	1 to 40 MHz	Direct drive[2]
0	1	0	f_{XTAL} x 10	1 to 4 MHz	
0	0	1	f_{XTAL} x 0.5	2 to 80 MHz	CPU clock via prescaler[3]
0	0	0	f_{XTAL} x 2.5	4 to 16 MHz	

Notes: 1. The external clock input range refers to a CPU clock range of 1...40 MHz.
2. The maximum depends on the duty cycle of the external clock signal.
3. The maximum input frequency is 25 MHz when using an external crystal with the internal oscillator; providing that internal serial resistance of the crystal is less than 40 Ω. However, higher frequencies can be applied with an external clock source on XTAL1 pin, but in this case, the input clock signal must reach the defined levels V_{IL} and V_{IH2}. (Refer to product data sheet).

20 - POWER REDUCTION MODES

Two different power reduction modes have been implemented in the ST10F280.

– **Idle mode:** the CPU is stopped, while the peripherals continue their 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 is enabled but has not been activated (driven low) during the last bus access.

20.1 - Idle Mode

The power consumption of the ST10F280 microcontroller can be decreased by entering Idle mode. In this mode all peripherals, including the watchdog timer, continue to operate normally, only the CPU operation is halted.

Idle mode is entered after the IDLE instruction has been executed and the instruction before the IDLE instruction has been completed. To prevent unintentional entry into Idle mode, the IDLE instruction has been implemented as a protected 32-bit instruction.

Idle mode is terminated by interrupt request from any enabled interrupt source whose individual Interrupt Enable flag has been set before the Idle mode was entered, regardless of bit IEN.

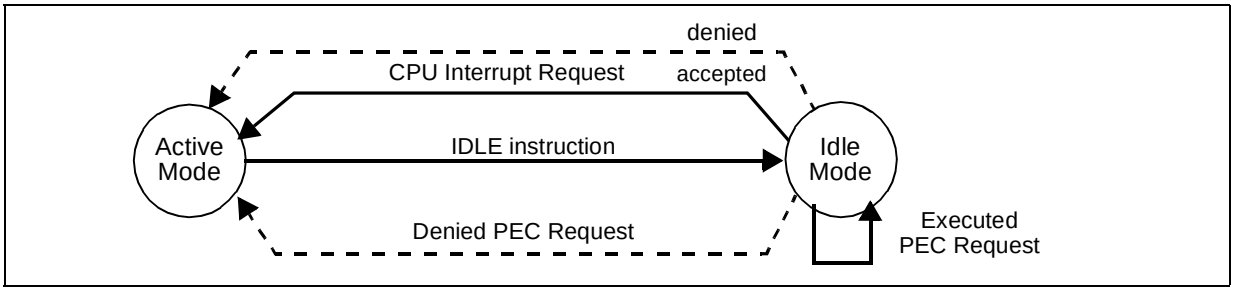
For a request selected for CPU interrupt service the associated interrupt service routine is entered if the priority level of the requesting source is higher than the current CPU priority and the interrupt system is globally enabled. After the RETI (Return from Interrupt) instruction of the interrupt service routine is executed, the CPU continues executing the program with the instruction following the IDLE instruction. Otherwise, if the interrupt request cannot be serviced because of a too low priority or a globally disabled interrupt system, the CPU immediately resumes normal program execution with the instruction following the IDLE instruction.

For a request which was programmed for PEC service, a PEC data transfer is performed if the priority level of this request is higher than the current CPU priority and if the interrupt system is globally enabled. After the PEC data transfer has been completed the CPU remains in Idle mode. Otherwise, if the PEC request cannot be serviced because of a too low priority or a globally disabled interrupt system, the CPU does not remain in Idle mode but continues program execution with the instruction following the IDLE instruction (see Figure 161).

Idle mode can also be terminated by a Non-Maskable Interrupt, with a high to low transition on the $\overline{\text{NMI}}$ pin. After Idle mode has been terminated by an interrupt or NMI request, the interrupt system performs a round of prioritization to determine the highest priority request. In the case of an NMI request, the NMI trap will always be entered.

Any interrupt request whose individual Interrupt Enable flag was set before Idle mode was entered will terminate Idle mode regardless of the current CPU priority. The CPU will **not** go back into Idle mode when a CPU interrupt request is detected, even when the interrupt was not serviced because of a higher CPU priority or a globally disabled interrupt system (IEN='0'). The CPU will **only** go back into Idle mode when the interrupt system is globally enabled (IEN='1') **and** a PEC service on a priority level higher than the current CPU level is requested and executed.

Figure 161 : Transitions between Idle mode and active mode



Note An interrupt request which is individually enabled and assigned to priority level 0 will terminate Idle mode. The associated interrupt vector will not be accessed, however.

The watchdog timer may be used to monitor the Idle mode: an internal reset will be generated if no interrupt or NMI request occurs before the watchdog timer overflows. To prevent the watchdog timer from overflowing during Idle mode it must be programmed to a reasonable duration interval before Idle mode is entered.

20.2 - Power Down Mode

To reduce power consumption, the microcontroller can be switched to Power Down mode. Clocking of all internal blocks is stopped. The contents of the internal RAM are preserved by the voltage supplied by the V_{DD} pins. The watchdog timer is stopped.

There are two different operating power down modes:

– Protected Power Down Mode

– Interruptible Power Down Mode

The power down mode is selected by bit 5 - 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	BDR STEN	XPEN	VISI BLE	XPER-SHARE
RW			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 $\overline{\text{NMI}}$ 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.

Note Register SYSCON cannot be changed after execution of the EINIT instruction.

20.2.1 - Protected Power Down Mode

This mode is selected by setting bit PWDCFG in the SYSCON register to '0'.

Entering power down mode can only be achieved if the $\overline{\text{NMI}}$ (Non Maskable Interrupt) pin is externally pulled low, while the PWRDN instruction is executed.

This feature can be used in conjunction with an external power failure signal which pulls the $\overline{\text{NMI}}$ pin low when a power failure is imminent. The microcontroller will enter the NMI trap routine, this routine can save the internal state into the RAM. After the internal state has been saved, the trap routine may set a flag or write a certain bit pattern into specific RAM locations, and then execute the PWRDN instruction. If the $\overline{\text{NMI}}$ pin is still low at this time, Power Down mode will be entered, otherwise program execution continues.

Exiting power down mode can only be achieved by external hardware reset.

The initialization routine (executed upon reset) checks the identification flag or bit pattern within the RAM to determine whether the controller was initially switched on, or whether it was properly restarted from power down mode.

20.3 - Interruptible Power Down Mode

This mode is selected by setting the bit PWDCFG in register SYSCON to '1'.

Entering power down mode can only be achieved if enabled Fast External Interrupt pins 0 to 3 (EXxIN pins, alternate functions of Port2 pins, with x = 7...0) are in their inactive level. This inactive level is configured with the EXIxES bit-field in the EXICON register, as follows:

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

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

Bit	Function
EXIxES (x=7...0)	**External Interrupt x Edge Selection Field (x=7...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' (ref as 'high' active level) 1 0: Interrupt on negative edge (falling) Enter Power Down mode if EXiIN = '1', exit if EXxIN = '0' (ref as 'low' active level) 1 1: Interrupt on any edge (rising or falling) Always enter Power Down mode, exit if EXxIN level changed.

EXISEL (F1DAh / EDh) ESFR Reset Value: 0000h

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
EXI7SS	EXI6SS	EXI5SS	EXI4SS	EXI3SS	EXI2SS	EXI1SS	EXI0SS
RW	RW	RW	RW	RW	RW	RW	RW

Bit	Function
EXIxSS	**External Interrupt x Source Selection (x=7...0)** '00': Input from associated Port 2 pin. '01': Input from "alternate source". '10': Input from Port 2 pin ORed with "alternate source". '11': Input from Port 2 pin ANDed with "alternate source".

EXIxSS	Port 2 pin	Alternate Source
0	P2.8	CAN1_RxD
1	P2.9	CAN2_RxD
2...7	P2.10...15	Not used (zero)

Exiting Power Down Mode

During power down mode, the CPU, the peripheral clocks and the oscillator and PLL clock are stopped. Power down mode can be exited by asserting, either $\overline{\text{RSTIN}}$, or one of the enabled EXxIN pins (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 (x = 3...0) Interrupt Enable bit (bit CCxIE in respective CCxIC register) need not to be set to bring the device out of power down mode.

An external RC circuit must be connected, as shown Figure 162.

To exit Power Down mode with external interrupt, an EXxIN pin has to be asserted for at least 40 ns (x = 7...0).

This signal enables the internal oscillator and PLL circuitry, and also turns on the weak pull-down (see Figure 163).

The discharging of the external capacitor provides a delay that allows the oscillator and PLL circuits to stabilize before the internal CPU and peripheral clocks are enabled.

When the RPD voltage drops below the threshold voltage (about 2.5 V), the Schmitt trigger clears Q2 flip-flop, therefore enabling 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 162 : Delay with RC on RPD pin

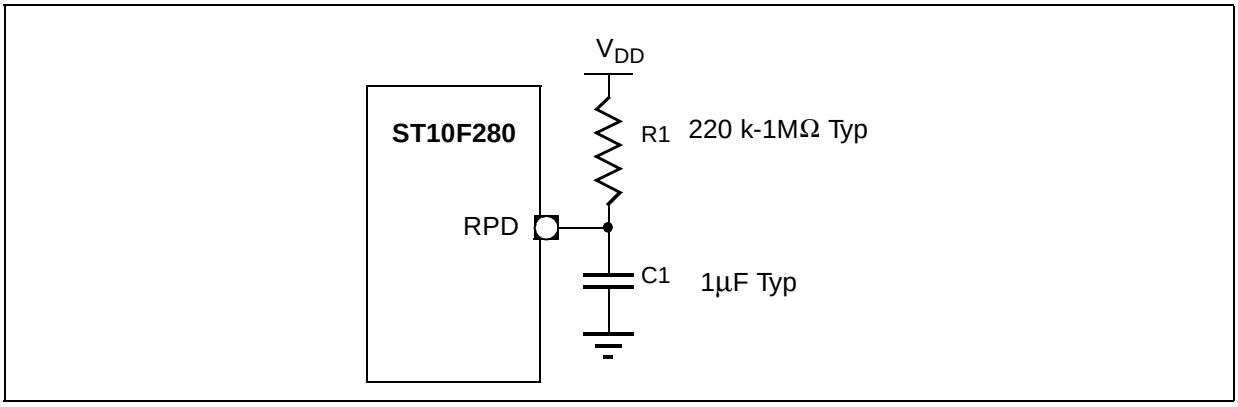


Figure 163 : Simplified powerdown exit circuitry

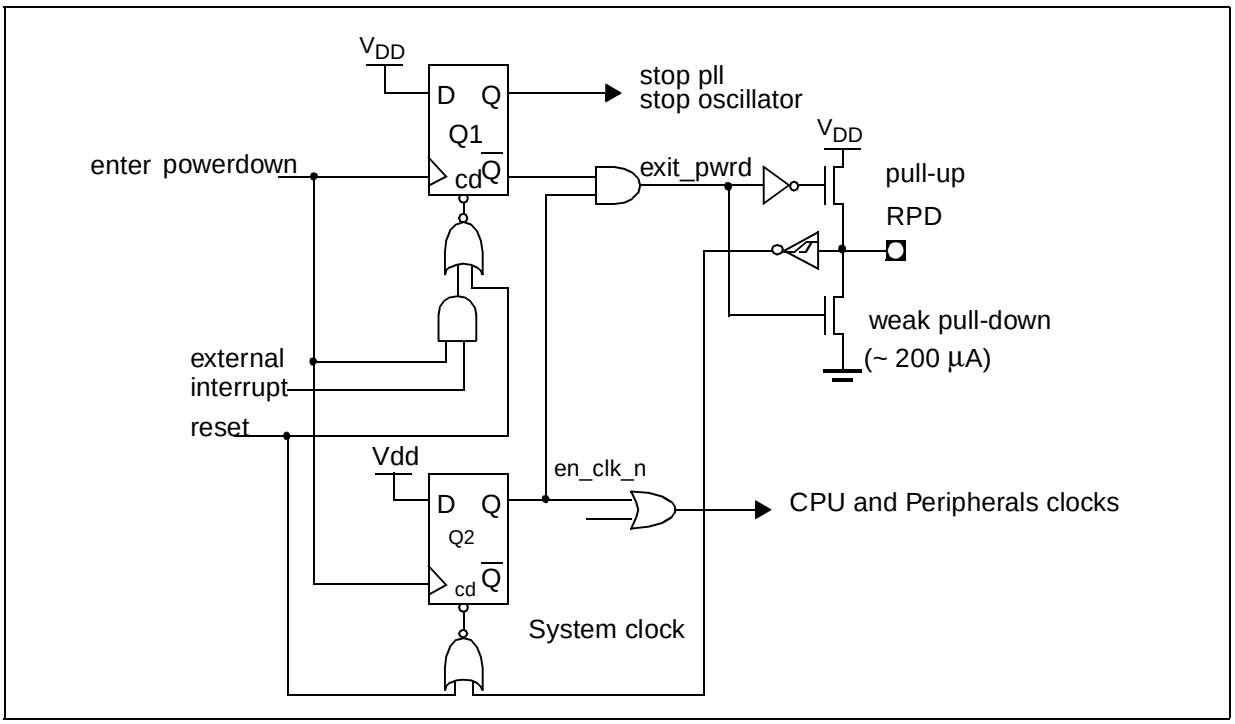
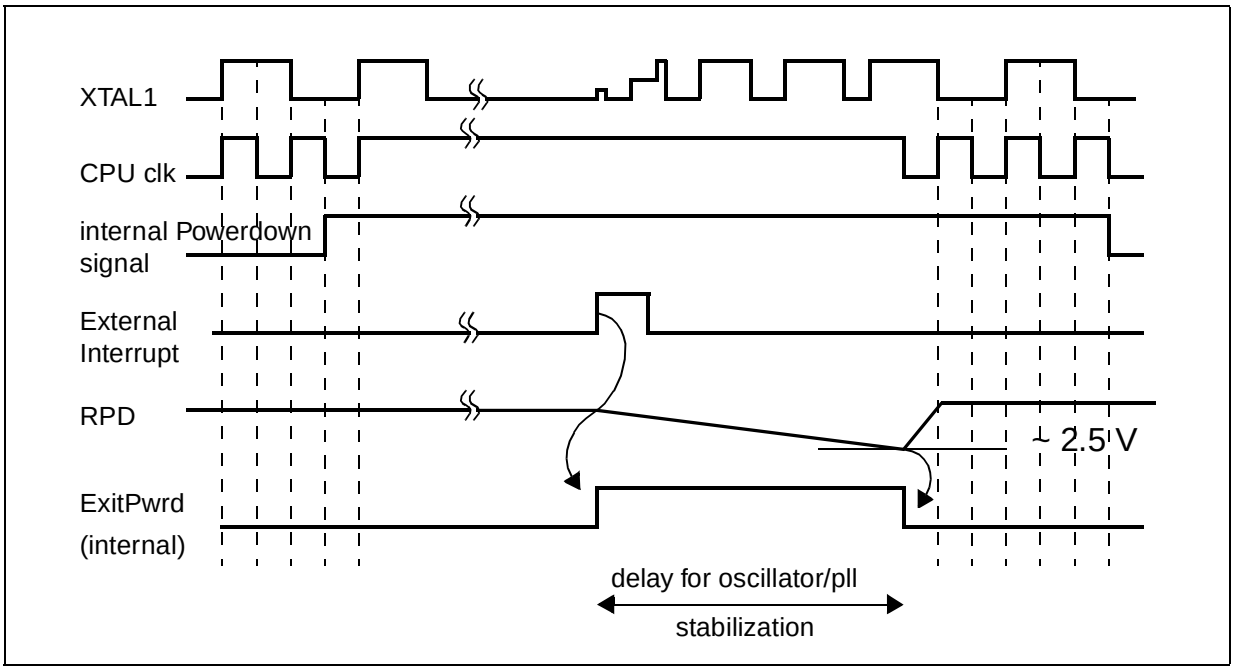


Figure 164 : Powerdown exit sequence when using an external interrupt (PLL x 2)



20.4 - Output Pin Status

During Idle mode the CPU clocks are turned off, while all peripherals continue their operation in the normal way. Therefore all ports pins, which are configured as general purpose output pins, output the last data value which was written to their port output latches. If the alternate output function of a port pin is used by a peripheral, the state of the pin is determined by the operation of the peripheral. Port pins which are used for bus control functions go into that state which represents the inactive state of the respective function ($\overline{WR}$), or to a defined state which is based on the last bus access ($\overline{BHE}$). 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 is always floating in Idle mode.

PORT1 outputs the lower 16 bits of the last address if a de-multiplexed bus mode is used, otherwise the output pins of PORT1 represent the port latch data.

PORT4 outputs the segment address for the last access on those pins that were selected during reset, otherwise the output pins of Port4 represent the port latch data.

During power down mode the oscillator and the clocks to the CPU and to the peripherals are turned off. Like in Idle mode, all port pins which are configured as general purpose output pins output the last data value which was written to their port output latches.

When the alternate output function of a port pin is used by a peripheral the state of this pin is determined by the last action of the peripheral before the clocks were switched off.

The Table 57 summarizes the state of all ST10F280 output pins during Idle and Power Down mode.

Table 57 : Output pin state during idle and powerdown modes

ST10F280 Output Pin(s)	Idle Mode		Power Down Mode	
	No external bus	External bus enabled	No external bus	External bus enabled
ALE	Low	Low	Low	Low
RD, WR	High	High	High	High
CLKOUT	Active	Active	High	High
RSTOUT	1	1	1	1
P0L	Port Latch Data	Floating	Port Latch Data	Floating
P0H	Port Latch Data	A15...A8 [2] / Float	Port Latch Data	A15...A8 [2] / Float
PORT1	Port Latch Data	Last Address [3] / Port Latch Data	Port Latch Data	Last Address [3] / Port Latch Data
Port 4	Port Latch Data	Port Latch Data/Last segment	Port Latch Data	Port Latch Data/Last segment
BHE	Port Latch Data	Last value	Port Latch Data	Last value
HLDA	Port Latch Data	Last value	Port Latch Data	Last value
BREQ	Port Latch Data	High	Port Latch Data	High
CSx	Port Latch Data	Last value [4]	Port Latch Data	Last value [4]
Other Port Output Pins	Port Latch Data / Alternate Function	Port Latch Data / Alternate Function	Port Latch Data / Alternate Function	Port Latch Data / Alternate Function

Notes 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 de-multiplexed buses.

4. The CS *signal that corresponds to the last address remains active (low), all other enabled* CS *signals remain inactive (high). By accessing an on-chip X-Periperal prior to entering a power save mode all external* CS *signals can be deactivated.*

21 - REGISTER SET

This section summarizes all registers implemented in the ST10F280, and explains the description format used in the chapters describing the function and layout of the SFRs.
For easy reference the registers (except for GPRs) are ordered in two ways:

– Ordered by address, to check which register a given address references.

– Ordered by register name, to find the location of a specific register.

21.1 - Register Description Format

In the following chapters, the function and the layout of the SFRs is described in a specific format. The example below explains this format.

A word register looks like this:

REG_NAME (A16h / A8h) SFR/ESFR/XReg 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) SFR/ESFR/XReg Reset Value: - - **h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	std bit	**hw bit**	bit field				bit field	
								RW	RW	RW				RW	

Elements:

REG_NAME	Name of this register
A16h / A8h	Long 16-bit address / Short 8-bit address
SFR**/**ESFR/XReg	Register space (SFR, ESFR or External/XBUS Register)
(* *) * *	Register contents after reset **0/1**: defined **X**': undefined (undefined ('**X**') after power up) **U**': unchanged
hwbit	bit that are set/cleared by hardware are written in **bold**

21.2 - 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.

Table 58 : General purpose registers (GPRs)

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
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

The first 8 GPRs (R7...R0) may also be accessed byte wise. Other than with SFRs, writing to a GPR byte does not affect the other byte of the respective GPR. The respective halves of the byte-accessible registers receive special names:

Table 59 : General purpose registers (GPRs) bit wise addressing

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
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) + 15	FFh	CPU General Purpose (byte) Register RH7	UUh

21.3 - Special Function Registers Ordered by Name

The following table lists all SFRs which are implemented in the ST10F280 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".

Table 60 : Special function registers listed by name

Name	Physical address	8-bit address	Description	Reset value
ADCIC b	FF98h	CCh	A/D Converter end of Conversion Interrupt Control Register	- - 00h
ADCON b	FFA0h	D0h	A/D Converter Control Register	0000h
ADDAT	FEA0h	50h	A/D Converter Result Register	0000h
ADDAT2	F0A0h E	50h	A/D Converter 2 Result Register	0000h
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
ADEIC b	FF9Ah	CDh	A/D Converter Overrun Error Interrupt Control Register	- - 00h
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
CC0	FE80h	40h	CAPCOM Register 0	0000h
CC0IC b	FF78h	BCh	CAPCOM Register 0 Interrupt Control Register	- - 00h
CC1	FE82h	41h	CAPCOM Register 1	0000h
CC1IC b	FF7Ah	BDh	CAPCOM Register 1 Interrupt Control Register	- - 00h
CC2	FE84h	42h	CAPCOM Register 2	0000h
CC2IC b	FF7Ch	BEh	CAPCOM Register 2 Interrupt Control Register	- - 00h
CC3	FE86h	43h	CAPCOM Register 3	0000h
CC3IC b	FF7Eh	BFh	CAPCOM Register 3 Interrupt Control Register	- - 00h
CC4	FE88h	44h	CAPCOM Register 4	0000h
CC4IC b	FF80h	C0h	CAPCOM Register 4 Interrupt Control Register	- - 00h
CC5	FE8Ah	45h	CAPCOM Register 5	0000h
CC5IC b	FF82h	C1h	CAPCOM Register 5 Interrupt Control Register	- - 00h
CC6	FE8Ch	46h	CAPCOM Register 6	0000h
CC6IC b	FF84h	C2h	CAPCOM Register 6 Interrupt Control Register	- - 00h
CC7	FE8Eh	47h	CAPCOM Register 7	0000h
CC7IC b	FF86h	C3h	CAPCOM Register 7 Interrupt Control Register	- - 00h
CC8	FE90h	48h	CAPCOM Register 8	0000h
CC8IC b	FF88h	C4h	CAPCOM Register 8 Interrupt Control Register	- - 00h
CC9	FE92h	49h	CAPCOM Register 9	0000h
CC9IC b	FF8Ah	C5h	CAPCOM Register 9 Interrupt Control Register	- - 00h
CC10	FE94h	4Ah	CAPCOM Register 10	0000h
CC10IC b	FF8Ch	C6h	CAPCOM Register 10 Interrupt Control Register	- - 00h
CC11	FE96h	4Bh	CAPCOM Register 11	0000h

Table 60 : Special function registers listed by name (continued)

Name	Physical address	8-bit address	Description	Reset value
CC11IC b	FF8Eh	C7h	CAPCOM Register 11 Interrupt Control Register	- - 00h
CC12	FE98h	4Ch	CAPCOM Register 12	0000h
CC12IC b	FF90h	C8h	CAPCOM Register 12 Interrupt Control Register	- - 00h
CC13	FE9Ah	4Dh	CAPCOM Register 13	0000h
CC13IC b	FF92h	C9h	CAPCOM Register 13 Interrupt Control Register	- - 00h
CC14	FE9Ch	4Eh	CAPCOM Register 14	0000h
CC14IC b	FF94h	CAh	CAPCOM Register 14 Interrupt Control Register	- - 00h
CC15	FE9Eh	4Fh	CAPCOM Register 15	0000h
CC15IC b	FF96h	CBh	CAPCOM Register 15 Interrupt Control Register	- - 00h
CC16	FE60h	30h	CAPCOM Register 16	0000h
CC16IC b	F160h E	B0h	CAPCOM Register 16 Interrupt Control Register	- - 00h
CC17	FE62h	31h	CAPCOM Register 17	0000h
CC17IC b	F162h E	B1h	CAPCOM Register 17 Interrupt Control Register	- - 00h
CC18	FE64h	32h	CAPCOM Register 18	0000h
CC18IC b	F164h E	B2h	CAPCOM Register 18 Interrupt Control Register	- - 00h
CC19	FE66h	33h	CAPCOM Register 19	0000h
CC19IC b	F166h E	B3h	CAPCOM Register 19 Interrupt Control Register	- - 00h
CC20	FE68h	34h	CAPCOM Register 20	0000h
CC20IC b	F168h E	B4h	CAPCOM Register 20 Interrupt Control Register	- - 00h
CC21	FE6Ah	35h	CAPCOM Register 21	0000h
CC21IC b	F16Ah E	B5h	CAPCOM Register 21 Interrupt Control Register	- - 00h
CC22	FE6Ch	36h	CAPCOM Register 22	0000h
CC22IC b	F16Ch E	B6h	CAPCOM Register 22 Interrupt Control Register	- - 00h
CC23	FE6Eh	37h	CAPCOM Register 23	0000h
CC23IC b	F16Eh E	B7h	CAPCOM Register 23 Interrupt Control Register	- - 00h
CC24	FE70h	38h	CAPCOM Register 24	0000h
CC24IC b	F170h E	B8h	CAPCOM Register 24 Interrupt Control Register	- - 00h
CC25	FE72h	39h	CAPCOM Register 25	0000h
CC25IC b	F172h E	B9h	CAPCOM Register 25 Interrupt Control Register	- - 00h
CC26	FE74h	3Ah	CAPCOM Register 26	0000h
CC26IC b	F174h E	BAh	CAPCOM Register 26 Interrupt Control Register	- - 00h
CC27	FE76h	3Bh	CAPCOM Register 27	0000h
CC27IC b	F176h E	BBh	CAPCOM Register 27 Interrupt Control Register	- - 00h
CC28	FE78h	3Ch	CAPCOM Register 28	0000h
CC28IC b	F178h E	BCh	CAPCOM Register 28 Interrupt Control Register	- - 00h
CC29	FE7Ah	3Dh	CAPCOM Register 29	0000h
CC29IC b	F184h E	C2h	CAPCOM Register 29 Interrupt Control Register	- - 00h
CC30	FE7Ch	3Eh	CAPCOM Register 30	0000h
CC30IC b	F18Ch E	C6h	CAPCOM Register 30 Interrupt Control Register	- - 00h
CC31	FE7Eh	3Fh	CAPCOM Register 31	0000h
CC31IC b	F194h E	CAh	CAPCOM Register 31 Interrupt Control Register	- - 00h
CCM0 b	FF52h	A9h	CAPCOM Mode Control Register 0	0000h

Table 60 : Special function registers listed by name (continued)

Name		Physical address		8-bit address	Description	Reset value
CCM1	b	FF54h		AAh	CAPCOM Mode Control Register 1	0000h
CCM2	b	FF56h		ABh	CAPCOM Mode Control Register 2	0000h
CCM3	b	FF58h		ACh	CAPCOM Mode Control Register 3	0000h
CCM4	b	FF22h		91h	CAPCOM Mode Control Register 4	0000h
CCM5	b	FF24h		92h	CAPCOM Mode Control Register 5	0000h
CCM6	b	FF26h		93h	CAPCOM Mode Control Register 6	0000h
CCM7	b	FF28h		94h	CAPCOM Mode Control Register 7	0000h
CP		FE10h		08h	CPU Context Pointer Register	FC00h
CRIC	b	FF6Ah		B5h	GPT2 CAPREL Interrupt Control Register	- - 00h
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	0000h
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	- - 00h
DP8	b	FFD6h		EBh	Port 8 Direction Control Register	- - 00h
DPP0		FE00h		00h	CPU Data Page Pointer 0 Register (10-bit)	0000h
DPP1		FE02h		01h	CPU Data Page Pointer 1 Register (10-bit)	0001h
DPP2		FE04h		02h	CPU Data Page Pointer 2 Register (10-bit)	0002h
DPP3		FE06h		03h	CPU Data Page Pointer 3 Register (10-bit)	0003h
EXICON	b	F1C0h	E	E0h	External Interrupt Control Register	0000h
EXISEL	b	F1DAh	E	EDh	External Interrupt Source Selection Register	0000h
IDCHIP		F07Ch	E	3Eh	Device Identifier Register (n is the device revision)	118nh
IDMANUF		F07Eh	E	3Fh	Manufacturer Identifier Register	0401h
IDMEM		F07Ah	E	3Dh	On-chip Memory Identifier Register	3080h
IDPROG		F078h	E	3Ch	Programming Voltage Identifier Register	0040h
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	b	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	0000h
ODP3	b	F1C6h	E	E3h	Port 3 Open Drain Control Register	0000h

Table 60 : Special function registers listed by name (continued)

Name		Physical address		8-bit address	Description	Reset value
ODP4	b	F1CAh	E	E5h	Port 4 Open Drain Control Register	- - 00h
ODP6	b	F1CEh	E	E7h	Port 6 Open Drain Control Register	- - 00h
ODP7	b	F1D2h	E	E9h	Port 7 Open Drain Control Register	- - 00h
ODP8	b	F1D6h	E	EBh	Port 8 Open Drain Control Register	- - 00h
ONES	b	FF1Eh		8Fh	Constant Value 1's Register (read only)	FFFFh
P0L	b	FF00h		80h	PORT0 Low Register (Lower half of PORT0)	- - 00h
P0H	b	FF02h		81h	PORT0 High Register (Upper half of PORT0)	- - 00h
P1L	b	FF04h		82h	PORT1 Low Register (Lower half of PORT1)	- - 00h
P1H	b	FF06h		83h	PORT1 High Register (Upper half of PORT1)	- - 00h
P2	b	FFC0h		E0h	Port 2 Register	0000h
P3	b	FFC4h		E2h	Port 3 Register	0000h
P4	b	FFC8h		E4h	Port 4 Register (8-bit)	- - 00h
P5	b	FFA2h		D1h	Port 5 Register (read only)	XXXXh
P6	b	FFCCh		E6h	Port 6 Register (8-bit)	- - 00h
P7	b	FFD0h		E8h	Port 7 Register (8-bit)	- - 00h
P8	b	FFD4h		EAh	Port 8 Register (8-bit)	- - 00h
P5DIDIS	b	FFA4h		D2h	Port 5 Digital Disable Register	0000h
POCON0L		F080h	E	40h	PORT0 Low Outpout Control Register (8-bit)	- - 00h
POCON0H		F082h	E	41h	PORT0 High Output Control Register (8-bit)	- - 00h
POCON1L		F084h	E	42h	PORT1 Low Output Control Register (8-bit)	- - 00h
POCON1H		F086h	E	43h	PORT1 High Output Control Register (8-bit)	- - 00h
POCON2		F088h	E	44h	Port2 Output Control Register	0000h
POCON3		F08Ah	E	45h	Port3 Output Control Register	0000h
POCON4		F08Ch	E	46h	Port4 Output Control Register (8-bit)	- - 00h
POCON6		F08Eh	E	47h	Port6 Output Control Register (8-bit)	- - 00h
POCON7		F090h	E	48h	Port7 Output Control Register (8-bit)	- - 00h
POCON8		F092h	E	49h	Port8 Output Control Register (8-bit)	- - 00h
POCON20		F0AAh	E	55h	ALE, $\overline{RD}$, $\overline{WR}$ Output Control Register (8-bit)	- - 00h
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
PICON	b	F1C4h	E	E2h	Port Input Threshold Control Register	- - 00h
PP0		F038h	E	1Ch	PWM Module Period Register 0	0000h
PP1		F03Ah	E	1Dh	PWM Module Period Register 1	0000h
PP2		F03Ch	E	1Eh	PWM Module Period Register 2	0000h
PP3		F03Eh	E	1Fh	PWM Module Period Register 3	0000h
PSW	b	FF10h		88h	CPU Program Status Word	0000h

Table 60 : Special function registers listed by name (continued)

Name	Physical address	8-bit address	Description	Reset value
PT0	F030h E	18h	PWM Module Up/Down Counter 0	0000h
PT1	F032h E	19h	PWM Module Up/Down Counter 1	0000h
PT2	F034h E	1Ah	PWM Module Up/Down Counter 2	0000h
PT3	F036h E	1Bh	PWM Module Up/Down Counter 3	0000h
PW0	FE30h	18h	PWM Module Pulse Width Register 0	0000h
PW1	FE32h	19h	PWM Module Pulse Width Register 1	0000h
PW2	FE34h	1Ah	PWM Module Pulse Width Register 2	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	- - 00h
QR0	F004h E	02h	MAC Unit Offset Register QR0	0000h
QR1	F006h E	03h	MAC Unit Offset Register QR1	0000h
QX0	F000h E	00h	MAC Unit Offset Register QX0	0000h
QX1	F002h E	01h	MAC Unit Offset Register QX1	0000h
RP0H b	F108h E	84h	System Start-up Configuration Register (read only)	- - XXh
S0BG	FEB4h	5Ah	Serial Channel 0 Baud Rate Generator Reload Register	0000h
S0CON b	FFB0h	D8h	Serial Channel 0 Control Register	0000h
S0EIC b	FF70h	B8h	Serial Channel 0 Error Interrupt Control Register	- - 00h
S0RBUF	FEB2h	59h	Serial Channel 0 Receive Buffer Register (read only)	- - XXh
S0RIC b	FF6Eh	B7h	Serial Channel 0 Receive Interrupt Control Register	- - 00h
S0TBIC b	F19Ch E	CEh	Serial Channel 0 Transmit Buffer Interrupt Control Register	- - 00h
S0TBUF	FEB0h	58h	Serial Channel 0 Transmit Buffer Register (write only)	0000h
S0TIC b	FF6Ch	B6h	Serial Channel 0 Transmit Interrupt Control Register	- - 00h
SP	FE12h	09h	CPU System Stack Pointer Register	FC00h
SSCBR	F0B4h E	5Ah	SSC Baud Rate Register	0000h
SSCCON b	FFB2h	D9h	SSC Control Register	0000h
SSCEIC b	FF76h	BBh	SSC Error Interrupt Control Register	- - 00h
SSCRB	F0B2h E	59h	SSC Receive Buffer (read only)	XXXXh
SSCRIC b	FF74h	BAh	SSC Receive Interrupt Control Register	- - 00h
SSCTB	F0B0h E	58h	SSC Transmit Buffer (write only)	0000h
SSCTIC b	FF72h	B9h	SSC Transmit Interrupt Control Register	- - 00h
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)]
T0	FE50h	28h	CAPCOM Timer 0 Register	0000h
T01CON b	FF50h	A8h	CAPCOM Timer 0 and Timer 1 Control Register	0000h
T0IC b	FF9Ch	CEh	CAPCOM Timer 0 Interrupt Control Register	- - 00h
T0REL	FE54h	2Ah	CAPCOM Timer 0 Reload Register	0000h
T1	FE52h	29h	CAPCOM Timer 1 Register	0000h
T1IC b	FF9Eh	CFh	CAPCOM Timer 1 Interrupt Control Register	- - 00h
T1REL	FE56h	2Bh	CAPCOM Timer 1 Reload Register	0000h

Table 60 : Special function registers listed by name (continued)

Name		Physical address		8-bit address	Description	Reset value
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	- - 00h
T3		FE42h		21h	GPT1 Timer 3 Register	0000h
T3CON	b	FF42h		A1h	GPT1 Timer 3 Control Register	0000h
T3IC	b	FF62h		B1h	GPT1 Timer 3 Interrupt Control Register	- - 00h
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	- - 00h
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	- - 00h
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	- - 00h
T7		F050h	E	28h	CAPCOM Timer 7 Register	0000h
T78CON	b	FF20h		90h	CAPCOM Timer 7 and 8 Control Register	0000h
T7IC	b	F17Ah	E	BEh	CAPCOM Timer 7 Interrupt Control Register	- - 00h
T7REL		F054h	E	2Ah	CAPCOM Timer 7 Reload Register	0000h
T8		F052h	E	29h	CAPCOM Timer 8 Register	0000h
T8IC	b	F17Ch	E	BFh	CAPCOM Timer 8 Interrupt Control Register	- - 00h
T8REL		F056h	E	2Bh	CAPCOM Timer 8 Reload Register	0000h
TFR	b	FFACh		D6h	Trap Flag Register	0000h
WDT		FEAEh		57h	Watchdog Timer Register (read only)	0000h
WDTCON	b	FFAEh		D7h	Watchdog Timer Control Register	00xxh [2]
XP0IC	b	F186h	E	C3h	CAN1 Module Interrupt Control Register	- - 00h [3]
XP1IC	b	F18Eh	E	C7h	CAN2 Module Interrupt Control Register	- - 00h [3]
XP2IC	b	F196h	E	CBh	XPWM Module Interrupt Control Register	- - 00h [3]
XP3IC	b	F19Eh	E	CFh	PLL unlock Interrupt Control Register	- - 00h [3]
XPERCON		F024h	E	12h	XPER Configuration Register	- - 05h
ZEROS	b	FF1Ch		8Eh	Constant Value 0's Register (read only)	0000h

Notes: 1. The system configuration is selected during reset.

2. Bit WDTR indicates a watchdog timer triggered reset.

3. The XPnIC Interrupt Control Registers control interrupt requests from integrated X-Bus peripherals. Some software controlled interrupt requests may be generated by setting the XPnIR bits (of XPnIC register) of the unused X-peripheral nodes.

21.4 - Special Function Registers Ordered by Address

The following table lists all SFRs which are implemented in the ST10F280 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".

Table 61 : Special Function Registers ordered by address

Name	Physical Address	8-bit Address	Description	Reset Value
QX0	F000h E	00h	MAC Unit Offset Register QX0	0000h
QX1	F002h E	01h	MAC Unit Offset Register QX1	0000h
QR0	F004h E	02h	MAC Unit Offset Register QR0	0000h
QR1	F006h E	03h	MAC Unit Offset Register QR1	0000h
XPERCON	F024h E	12h	XPER Configuration Register	- - 05h
PT0	F030h E	18h	PWM Module Up/Down Counter 0	0000h
PT1	F032h E	19h	PWM Module Up/Down Counter 1	0000h
PT2	F034h E	1Ah	PWM Module Up/Down Counter 2	0000h
PT3	F036h E	1Bh	PWM Module Up/Down Counter 3	0000h
PP0	F038h E	1Ch	PWM Module Period Register 0	0000h
PP1	F03Ah E	1Dh	PWM Module Period Register 1	0000h
PP2	F03Ch E	1Eh	PWM Module Period Register 2	0000h
PP3	F03Eh E	1Fh	PWM Module Period Register 3	0000h
T7	F050h E	28h	CAPCOM Timer 7 Register	0000h
T8	F052h E	29h	CAPCOM Timer 8 Register	0000h
T7REL	F054h E	2Ah	CAPCOM Timer 7 Reload Register	0000h
T8REL	F056h E	2Bh	CAPCOM Timer 8 Reload Register	0000h
IDPROG	F078h E	3Ch	Programming Voltage Identifier Register	0040h
IDMEM	F07Ah E	3Dh	On-chip Memory Identifier Register	3080h
IDCHIP	F07Ch E	3Eh	Device Identifier Register (n is the device revision)	118nh
IDMANUF	F07Eh E	3Fh	Manufacturer Identifier Register	0401h
POCON0L	F080h E	40h	PORT0 Low Outpout Control Register (8-bit)	- - 00h
POCON0H	F082h E	41h	PORT0 High Output Control Register (8-bit)	- - 00h
POCON1L	F084h E	42h	PORT1 Low Output Control Register (8-bit)	- - 00h
POCON1H	F086h E	43h	PORT1 High Output Control Register (8-bit)	- - 00h
POCON2	F088h E	44h	Port2 Output Control Register	0000h
POCON3	F08Ah E	45h	Port3 Output Control Register	0000h
POCON4	F08Ch E	46h	Port4 Output Control Register (8-bit)	- - 00h
POCON6	F08Eh E	47h	Port6 Output Control Register (8-bit)	- - 00h
POCON7	F090h E	48h	Port7 Output Control Register (8-bit)	- - 00h
POCON8	F092h E	49h	Port8 Output Control Register (8-bit)	- - 00h
ADDAT2	F0A0h E	50h	A/D Converter 2 Result Register	0000h
POCON20	F0AAh E	55h	ALE, RD, WR Output Control Register (8-bit)	0000h
SSCTB	F0B0h E	58h	SSC Transmit Buffer (write only)	0000h
SSCRB	F0B2h E	59h	SSC Receive Buffer (read only)	XXXXh
SSCBR	F0B4h E	5Ah	SSC Baud rate Register	0000h
DP0L b	F100h E	80h	P0L Direction Control Register	- - 00h

Table 61 : Special Function Registers ordered by address (continued)

Name		Physical Address		8-bit Address	Description	Reset Value
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
RP0H	b	F108h	E	84h	System Start-up Configuration Register (Read only)	- - XXh
CC16IC	b	F160h	E	B0h	CAPCOM Register 16 Interrupt Control Register	- - 00h
CC17IC	b	F162h	E	B1h	CAPCOM Register 17 Interrupt Control Register	- - 00h
CC18IC	b	F164h	E	B2h	CAPCOM Register 18 Interrupt Control Register	- - 00h
CC19IC	b	F166h	E	B3h	CAPCOM Register 19 Interrupt Control Register	- - 00h
CC20IC	b	F168h	E	B4h	CAPCOM Register 20 Interrupt Control Register	- - 00h
CC21IC	b	F16Ah	E	B5h	CAPCOM Register 21 Interrupt Control Register	- - 00h
CC22IC	b	F16Ch	E	B6h	CAPCOM Register 22 Interrupt Control Register	- - 00h
CC23IC	b	F16Eh	E	B7h	CAPCOM Register 23 Interrupt Control Register	- - 00h
CC24IC	b	F170h	E	B8h	CAPCOM Register 24 Interrupt Control Register	- - 00h
CC25IC	b	F172h	E	B9h	CAPCOM Register 25 Interrupt Control Register	- - 00h
CC26IC	b	F174h	E	BAh	CAPCOM Register 26 Interrupt Control Register	- - 00h
CC27IC	b	F176h	E	BBh	CAPCOM Register 27 Interrupt Control Register	- - 00h
CC28IC	b	F178h	E	BCh	CAPCOM Register 28 Interrupt Control Register	- - 00h
T7IC	b	F17Ah	E	BEh	CAPCOM Timer 7 Interrupt Control Register	- - 00h
T8IC	b	F17Ch	E	BFh	CAPCOM Timer 8 Interrupt Control Register	- - 00h
PWMIC	b	F17Eh	E	BFh	PWM Module Interrupt Control Register	- - 00h
CC29IC	b	F184h	E	C2h	CAPCOM Register 29 Interrupt Control Register	- - 00h
XP0IC	b	F186h	E	C3h	CAN1 Module Interrupt Control Register	- - 00h
CC30IC	b	F18Ch	E	C6h	CAPCOM Register 30 Interrupt Control Register	- - 00h
XP1IC	b	F18Eh	E	C7h	CAN2 Module Interrupt Control Register	- - 00h
CC31IC	b	F194h	E	CAh	CAPCOM Register 31 Interrupt Control Register	- - 00h
XP2IC	b	F196h	E	CBh	XPWM Module Interrupt Control Register	- - 00h
S0TBIC	b	F19Ch	E	CEh	Serial Channel 0 Transmit Buffer Interrupt Control Register	- - 00h
XP3IC	b	F19Eh	E	CFh	PLL unlock Interrupt Control Register	- - 00h
EXICON	b	F1C0h	E	E0h	External Interrupt Control Register	0000h
ODP2	b	F1C2h	E	E1h	Port2 Open Drain Control Register	0000h
PICON	b	F1C4h	E	E2h	Port Input Threshold Control Register	- - 00h
ODP3	b	F1C6h	E	E3h	Port3 Open Drain Control Register	0000h
ODP4	b	F1CAh	E	E5h	Port4 Open Drain Control Register	- - 00h
ODP6	b	F1CEh	E	E7h	Port6 Open Drain Control Register	- - 00h
ODP7	b	F1D2h	E	E9h	Port7 Open Drain Control Register	- - 00h
ODP8	b	F1D6h	E	EBh	Port8 Open Drain Control Register	- - 00h
EXISEL	b	F1DAh	E	EDh	External Interrupt Source Selection Register	0000h
DPP0		FE00h		00h	CPU Data Page Pointer 0 Register (10-bit)	0000h
DPP1		FE02h		01h	CPU Data Page Pointer 1 Register (10-bit)	0001h
DPP2		FE04h		02h	CPU Data Page Pointer 2 Register (10-bit)	0002h

Table 61 : Special Function Registers ordered by address (continued)

Name	Physical Address	8-bit Address	Description	Reset Value
DPP3	FE06h	03h	CPU Data Page Pointer 3 Register (10-bit)	0003h
CSP	FE08h	04h	CPU Code Segment Pointer Register (read only)	0000h
MDH	FE0Ch	06h	CPU Multiply Divide Register – High word	0000h
MDL	FE0Eh	07h	CPU Multiply Divide Register – Low word	0000h
CP	FE10h	08h	CPU Context Pointer Register	FC00h
SP	FE12h	09h	CPU System Stack Pointer Register	FC00h
STKOV	FE14h	0Ah	CPU Stack Overflow Pointer Register	FA00h
STKUN	FE16h	0Bh	CPU Stack Underflow Pointer Register	FC00h
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
PW0	FE30h	18h	PWM Module Pulse Width Register 0	0000h
PW1	FE32h	19h	PWM Module Pulse Width Register 1	0000h
PW2	FE34h	1Ah	PWM Module Pulse Width Register 2	0000h
PW3	FE36h	1Bh	PWM Module Pulse Width Register 3	0000h
T2	FE40h	20h	GPT1 Timer 2 Register	0000h
T3	FE42h	21h	GPT1 Timer 3 Register	0000h
T4	FE44h	22h	GPT1 Timer 4 Register	0000h
T5	FE46h	23h	GPT2 Timer 5 Register	0000h
T6	FE48h	24h	GPT2 Timer 6 Register	0000h
CAPREL	FE4Ah	25h	GPT2 Capture/Reload Register	0000h
T0	FE50h	28h	CAPCOM Timer 0 Register	0000h
T1	FE52h	29h	CAPCOM Timer 1 Register	0000h
T0REL	FE54h	2Ah	CAPCOM Timer 0 Reload Register	0000h
T1REL	FE56h	2Bh	CAPCOM Timer 1 Reload Register	0000h
MAL	FE5Ch	2Eh	MAC Unit Accumulator - Low Word	0000h
MAH	FE5Eh	2Fh	MAC Unit Accumulator - High Word	0000h
CC16	FE60h	30h	CAPCOM Register 16	0000h
CC17	FE62h	31h	CAPCOM Register 17	0000h
CC18	FE64h	32h	CAPCOM Register 18	0000h
CC19	FE66h	33h	CAPCOM Register 19	0000h
CC20	FE68h	34h	CAPCOM Register 20	0000h
CC21	FE6Ah	35h	CAPCOM Register 21	0000h
CC22	FE6Ch	36h	CAPCOM Register 22	0000h
CC23	FE6Eh	37h	CAPCOM Register 23	0000h
CC24	FE70h	38h	CAPCOM Register 24	0000h
CC25	FE72h	39h	CAPCOM Register 25	0000h
CC26	FE74h	3Ah	CAPCOM Register 26	0000h
CC27	FE76h	3Bh	CAPCOM Register 27	0000h

Table 61 : Special Function Registers ordered by address (continued)

Name	Physical Address	8-bit Address	Description	Reset Value
CC28	FE78h	3Ch	CAPCOM Register 28	0000h
CC29	FE7Ah	3Dh	CAPCOM Register 29	0000h
CC30	FE7Ch	3Eh	CAPCOM Register 30	0000h
CC31	FE7Eh	3Fh	CAPCOM Register 31	0000h
CC0	FE80h	40h	CAPCOM Register 0	0000h
CC1	FE82h	41h	CAPCOM Register 1	0000h
CC2	FE84h	42h	CAPCOM Register 2	0000h
CC3	FE86h	43h	CAPCOM Register 3	0000h
CC4	FE88h	44h	CAPCOM Register 4	0000h
CC5	FE8Ah	45h	CAPCOM Register 5	0000h
CC6	FE8Ch	46h	CAPCOM Register 6	0000h
CC7	FE8Eh	47h	CAPCOM Register 7	0000h
CC8	FE90h	48h	CAPCOM Register 8	0000h
CC9	FE92h	49h	CAPCOM Register 9	0000h
CC10	FE94h	4Ah	CAPCOM Register 10	0000h
CC11	FE96h	4Bh	CAPCOM Register 11	0000h
CC12	FE98h	4Ch	CAPCOM Register 12	0000h
CC13	FE9Ah	4Dh	CAPCOM Register 13	0000h
CC14	FE9Ch	4Eh	CAPCOM Register 14	0000h
CC15	FE9Eh	4Fh	CAPCOM Register 15	0000h
ADDAT	FEA0h	50h	A/D Converter Result Register	0000h
WDT	FEAEh	57h	Watchdog Timer Register (read only)	0000h
S0TBUF	FEB0h	58h	Serial Channel 0 Transmit Buffer Register(write only)	0000h
S0RBUF	FEB2h	59h	Serial Channel 0 Receive Buffer Register (read only)	- - XXh
S0BG	FEB4h	5Ah	Serial Channel 0 Baud Rate Generator Reload Reg	0000h
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
P0L b	FF00h	80h	Port0 Low Register (Lower half of PORT0)	- - 00h
P0H b	FF02h	81h	Port0 High Register (Upper half of PORT0)	- - 00h
P1L b	FF04h	82h	Port1 Low Register (Lower half of PORT1)	- - 00h
P1H b	FF06h	83h	Port1 High Register (Upper half of PORT1)	- - 00h
IDX0 b	FF08h	84h	MAC Unit Address Pointer 0	0000h
IDX1 b	FF0Ah	85h	MAC Unit Address Pointer 1	0000h
BUSCON0 b	FF0Ch	86h	Bus Configuration Register 0	0xx0h

Table 61 : Special Function Registers ordered by address (continued)

Name	Physical Address	8-bit Address	Description	Reset Value
MDC b	FF0Eh	87h	CPU Multiply Divide Control Register	0000h
PSW b	FF10h	88h	CPU Program Status word	0000h
SYSCON b	FF12h	89h	CPU System Configuration Register	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
ZEROS b	FF1Ch	8Eh	Constant Value 0's Register (read only)	0000h
ONES b	FF1Eh	8Fh	Constant Value 1's Register (read only)	FFFFh
T78CON b	FF20h	90h	CAPCOM Timer 7 and 8 Control Register	0000h
CCM4 b	FF22h	91h	CAPCOM Mode Control Register 4	0000h
CCM5 b	FF24h	92h	CAPCOM Mode Control Register 5	0000h
CCM6 b	FF26h	93h	CAPCOM Mode Control Register 6	0000h
CCM7 b	FF28h	94h	CAPCOM Mode Control Register 7	0000h
PWMCON0 b	FF30h	98h	PWM Module Control Register 0	0000h
PWMCON1 b	FF32h	99h	PWM Module Control Register 1	0000h
T2CON b	FF40h	A0h	GPT1 Timer 2 Control Register	0000h
T3CON b	FF42h	A1h	GPT1 Timer 3 Control Register	0000h
T4CON b	FF44h	A2h	GPT1 Timer 4 Control Register	0000h
T5CON b	FF46h	A3h	GPT2 Timer 5 Control Register	0000h
T6CON b	FF48h	A4h	GPT2 Timer 6 Control Register	0000h
T01CON b	FF50h	A8h	CAPCOM Timer 0 and Timer 1 Control Register	0000h
CCM0 b	FF52h	A9h	CAPCOM Mode Control Register 0	0000h
CCM1 b	FF54h	AAh	CAPCOM Mode Control Register 1	0000h
CCM2 b	FF56h	ABh	CAPCOM Mode Control Register 2	0000h
CCM3 b	FF58h	ACh	CAPCOM Mode Control Register 3	0000h
T2IC b	FF60h	B0h	GPT1 Timer 2 Interrupt Control Register	- - 00h
T3IC b	FF62h	B1h	GPT1 Timer 3 Interrupt Control Register	- - 00h
T4IC b	FF64h	B2h	GPT1 Timer 4 Interrupt Control Register	- - 00h
T5IC b	FF66h	B3h	GPT2 Timer 5 Interrupt Control Register	- - 00h
T6IC b	FF68h	B4h	GPT2 Timer 6 Interrupt Control Register	- - 00h
CRIC b	FF6Ah	B5h	GPT2 CAPREL Interrupt Control Register	- - 00h
S0TIC b	FF6Ch	B6h	Serial Channel 0 Transmit Interrupt Control Register	- - 00h
S0RIC b	FF6Eh	B7h	Serial Channel 0 Receive Interrupt Control Register	- - 00h
S0EIC b	FF70h	B8h	Serial Channel 0 Error Interrupt Control Register	- - 00h
SSCTIC b	FF72h	B9h	SSC Transmit Interrupt Control Register	- - 00h
SSCRIC b	FF74h	BAh	SSC Receive Interrupt Control Register	- - 00h
SSCEIC b	FF76h	BBh	SSC Error Interrupt Control Register	- - 00h
CC0IC b	FF78h	BCh	CAPCOM Register 0 Interrupt Control Register	- - 00h
CC1IC b	FF7Ah	BDh	CAPCOM Register 1 Interrupt Control Register	- - 00h

Table 61 : Special Function Registers ordered by address (continued)

Name		Physical Address	8-bit Address	Description	Reset Value
CC2IC	b	FF7Ch	BEh	CAPCOM Register 2 Interrupt Control Register	- - 00h
CC3IC	b	FF7Eh	BFh	CAPCOM Register 3 Interrupt Control Register	- - 00h
CC4IC	b	FF80h	C0h	CAPCOM Register 4 Interrupt Control Register	- - 00h
CC5IC	b	FF82h	C1h	CAPCOM Register 5 Interrupt Control Register	- - 00h
CC6IC	b	FF84h	C2h	CAPCOM Register 6 Interrupt Control Register	- - 00h
CC7IC	b	FF86h	C3h	CAPCOM Register 7 Interrupt Control Register	- - 00h
CC8IC	b	FF88h	C4h	CAPCOM Register 8 Interrupt Control Register	- - 00h
CC9IC	b	FF8Ah	C5h	CAPCOM Register 9 Interrupt Control Register	- - 00h
CC10IC	b	FF8Ch	C6h	CAPCOM Register 10 Interrupt Control Register	- - 00h
CC11IC	b	FF8Eh	C7h	CAPCOM Register 11 Interrupt Control Register	- - 00h
CC12IC	b	FF90h	C8h	CAPCOM Register 12 Interrupt Control Register	- - 00h
CC13IC	b	FF92h	C9h	CAPCOM Register 13 Interrupt Control Register	- - 00h
CC14IC	b	FF94h	CAh	CAPCOM Register 14 Interrupt Control Register	- - 00h
CC15IC	b	FF96h	CBh	CAPCOM Register 15 Interrupt Control Register	- - 00h
ADCIC	b	FF98h	CCh	A/D Converter End of Conversion Interrupt Control Register	- - 00h
ADEIC	b	FF9Ah	CDh	A/D Converter Overrun Error Interrupt Control Reg	- - 00h
T0IC	b	FF9Ch	CEh	CAPCOM Timer 0 Interrupt Control Register	- - 00h
T1IC	b	FF9Eh	CFh	CAPCOM Timer 1 Interrupt Control Register	- - 00h
ADCON	b	FFA0h	D0h	A/D Converter Control Register	0000h
P5	b	FFA2h	D1h	Port5 Register (read only)	XXXXh
P5DIDIS	b	FFA4h	D2h	Port 5 Digital Disable Register	0000h
TFR	b	FFACh	D6h	Trap Flag Register	0000h
WDTCON	b	FFAEh	D7h	Watchdog Timer Control Register	00xxh
S0CON	b	FFB0h	D8h	Serial Channel 0 Control Register	0000h
SSCCON	b	FFB2h	D9h	SSC Control Register	0000h
P2	b	FFC0h	E0h	Port2 Register	0000h
DP2	b	FFC2h	E1h	Port2 Direction Control Register	0000h
P3	b	FFC4h	E2h	Port3 Register	0000h
DP3	b	FFC6h	E3h	Port3 Direction Control Register	0000h
P4	b	FFC8h	E4h	Port4 Register (8-bit)	- - 00h
DP4	b	FFCAh	E5h	Port4 Direction Control Register	- - 00h
P6	b	FFCCh	E6h	Port6 Register (8-bit)	- - 00h
DP6	b	FFCEh	E7h	Port6 Direction Control Register	- - 00h
P7	b	FFD0h	E8h	Port7 Register (8-bit)	- - 00h
DP7	b	FFD2h	E9h	Port7 Direction Control Register	- - 00h
P8	b	FFD4h	EAh	Port8 Register (8-bit)	- - 00h
DP8	b	FFD6h	EBh	Port8 Direction Control Register	- - 00h
MRW	b	FFDAh	EDh	MAC Unit Repeat Word	0000h
MCW	b	FFDCh	EEh	MAC Unit Control Word	0000h
MSW	b	FFDEh	EFh	MAC Unit Status Word	0200h

21.5 - X Registers Listed by Name

Table 62 : X registers listed by name

Name	Physical address	Description	Reset value
CAN1BTR	EF04h	CAN1 Bit Timing Register	XXXXh
CAN1CSR	EF00h	CAN1 Control/Status Register	XX01h
CAN1GMS	EF06h	CAN1 Global Mask Short	XFXXh
CAN1IR	EF02h	CAN1 Interrupt Register	- - XXh
CAN1LAR1--15	EF14--EFF4h	CAN1 Lower Arbitration register 1 to 15	XXXXh
CAN1LGML	EF0Ah	CAN1 Lower Global Mask Long	XXXXh
CAN1LMLM	EF0Eh	CAN1 Lower Mask Last Message	XXXXh
CAN1MCR1--15	EF10--EFF0h	CAN1 Message Control Register 1 to 15	XXXXh
CAN1MO1--15	EF1x--EFFxh	CAN1 Message Object 1 to 15	XXXXh
CAN1UAR1--15	EF12--EFF2h	CAN1 Upper Arbitration Register 1 to 15	XXXXh
CAN1UGML	EF08h	CAN1 Upper Global Mask Long	XXXXh
CAN1UMLM	EF0Ch	CAN1 Upper Mask Last Message	XXXXh
CAN2BTR	EE04h	CAN2 Bit Timing Register	XXXXh
CAN2CSR	EE00h	CAN2 Control/Status Register	XX01h
CAN2GMS	EE06h	CAN2 Global Mask Short	XFXXh
CAN2IR	EE02h	CAN2 Interrupt Register	- - XXh
CAN2LAR1--15	EE14--EEF4h	CAN2 Lower Arbitration register 1 to 15	XXXXh
CAN2LGML	EE0Ah	CAN2 Lower Global Mask Long	XXXXh
CAN2LMLM	EE0Eh	CAN2 Lower Mask Last Message	XXXXh
CAN2MCR1--15	EE10--EEF0h	CAN2 Message Control Register 1 to 15	XXXXh
CAN2MO1--15	EE1x--EEFxh	CAN2 Message Object 1 to 15	XXXXh
CAN2UAR1--15	EE12--EEF2h	CAN2 Upper Arbitration Register 1 to 15	XXXXh
CAN2UGML	EE08h	CAN2 Upper Global Mask Long	XXXXh
CAN2UMLM	EE0Ch	CAN2 Upper Mask Last Message	XXXXh
XADCMUX	C384h	Port5 or PortX10 ADC Input Selection (Read / Write)	0000h
XDP9	C200h	Direction Register Xport9 (Read / Write)	0000h
XDP9CLR	C204h	Bit Clear Direction Register Xport9 (Write only)	0000h
XDP9SET	C202h	Bit Set Direction Register Xport9 (Write only)	0000h
XODP9	C300h	Open Drain Control Register Xport9 (Read / Write)	0000h
XODP9CLR	C304h	Bit clear Open drain Control register Xport9 (Write only)	0000h
XODP9SET	C302h	Bit Set Open Drain Control Register Xport9 (Write only)	0000h
XP10	C380h	Read only Data register Xport10 (Read only)	XXXXh
XP10DIDIS	C382h	Xport10 Schmitt Trigger Input Selection (Read / Write)	0000h
XP9	C100h	Data Register Xport9 (Read / Write)	0000h
XP9CLR	C104h	Bit Clear Data Register Xport9 (Write only)	0000h

Table 62 : X registers listed by name

XP9SET	C102h	Bit Set Data Register Xport9 (Write only)	0000h
XPOLAR	EC04h	XPWM Channel Polarity Control Register	0000h
XPP0	EC20h	XPWM Period Register 0	0000h
XPP1	EC22h	XPWM Period Register 1	0000h
XPP2	EC24h	XPWM Period Register 2	0000h
XPP3	EC26h	XPWM Period Register 3	0000h
XPT0	EC10h	XPWM Timer Counter Register 0	0000h
XPT1	EC12h	XPWM Timer Counter Register 1	0000h
XPT2	EC14h	XPWM Timer Counter Register 2	0000h
XPT3	EC16h	XPWM Timer Counter Register 3	0000h
XPW0	EC30h	XPWM Pulse Width Register 0	0000h
XPW1	EC32h	XPWM Pulse Width Register 1	0000h
XPW2	EC34h	XPWM Pulse Width Register 2	0000h
XPW3	EC36h	XPWM Pulse Width Register 3	0000h
XPWMCON0	EC00h	XPWM Control Register 0	0000h
XPWMCON1	EC02h	XPWM Control Register 1	0000h
XTCR	C000h	Xtimer Control Register (Read / Write)	0000h
XTCVR	C006h	Xtimer Current Value Register (Read / Write)	0000h
XTEVR	C004h	Xtimer End Value Register (Read / Write)	0000h
XTSVR	C002h	Xtimer Start Value Register (Read / Write)	0000h

21.6 - X Registers Ordered by Address

Table 63 : X registers ordered by address

Name	Physical address	Description	Reset value
XTCR	C000h	Xtimer Control Register (Read / Write)	0000h
XTSVR	C002h	Xtimer Start Value Register (Read / Write)	0000h
XTEVR	C004h	Xtimer End Value Register (Read / Write)	0000h
XTCVR	C006h	Xtimer Current Value Register (Read / Write)	0000h
XP9	C100h	Data Register Xport9 (Read / Write)	0000h
XP9SET	C102h	Bit Set Data Register Xport9 (Write only)	0000h
XP9CLR	C104h	Bit Clear Data Register Xport9 (Write only)	0000h
XDP9	C200h	Direction Register Xport9 (Read / Write)	0000h
XDP9SET	C202h	Bit Set Direction Register Xport9 (Write only)	0000h
XDP9CLR	C204h	Bit Clear Direction Register Xport9 (Write only)	0000h
XODP9	C300h	Open Drain Control Register Xport9 (Read / Write)	0000h
XODP9SET	C302h	Bit Set Open Drain Control Register Xport9 (Write only)	0000h
XODP9CLR	C304h	Bit clear Open drain Control register Xport9 (Write only)	0000h
XP10	C380h	Read only Data register Xport10 (Read only)	XXXXh
XP10DIDIS	C382h	Xport10 Schmitt Trigger Input Selection (Read / Write)	0000h
XADCMUX	C384h	Port5 or PortX10 ADC Input Selection (Read / Write)	0000h
XPWMCON0	EC00h	XPWM Control Register 0	0000h
XPWMCON1	EC02h	XPWM Control Register 1	0000h
XPOLAR	EC04h	XPWM Channel Polarity Control Register	0000h
XPT0	EC10h	XPWM Timer Counter Register 0	0000h
XPT1	EC12h	XPWM Timer Counter Register 1	0000h
XPT2	EC14h	XPWM Timer Counter Register 2	0000h
XPT3	EC16h	XPWM Timer Counter Register 3	0000h
XPP0	EC20h	XPWM Period Register 0	0000h
XPP1	EC22h	XPWM Period Register 1	0000h
XPP2	EC24h	XPWM Period Register 2	0000h
XPP3	EC26h	XPWM Period Register 3	0000h
XPW0	EC30h	XPWM Pulse Width Register 0	0000h
XPW1	EC32h	XPWM Pulse Width Register 1	0000h
XPW2	EC34h	XPWM Pulse Width Register 2	0000h
XPW3	EC36h	XPWM Pulse Width Register 3	0000h
CAN2CSR	EE00h	CAN2 Control/Status Register	XX01h
CAN2IR	EE02h	CAN2 Interrupt Register	- - XXh
CAN2BTR	EE04h	CAN2 Bit Timing Register	XXXXh
CAN2GMS	EE06h	CAN2 Global Mask Short	XFXXh

Table 63 : X registers ordered by address

CAN2UGML	EE08h	CAN2 Upper Global Mask Long	XXXXh
CAN2LGML	EE0Ah	CAN2 Lower Global Mask Long	XXXXh
CAN2UMLM	EE0Ch	CAN2 Upper Mask Last Message	XXXXh
CAN2LMLM	EE0Eh	CAN2 Lower Mask Last Message	XXXXh
CAN2MCR1--15	EE10--EEF0h	CAN2 Message Control Register 1 to 15	XXXXh
CAN2UAR1--15	EE12--EEF2h	CAN2 Upper Arbitration Register 1 to 15	XXXXh
CAN2LAR1--15	EE14--EEF4h	CAN2 Lower Arbitration register 1 to 15	XXXXh
CAN2MO1--15	EE1x--EEFxh	CAN2 Message Object 1 to 15	XXXXh
CAN1CSR	EF00h	CAN1 Control/Status Register	XX01h
CAN1IR	EF02h	CAN1 Interrupt Register	- - XXh
CAN1BTR	EF04h	CAN1 Bit Timing Register	XXXXh
CAN1GMS	EF06h	CAN1 Global Mask Short	XFXXh
CAN1UGML	EF08h	CAN1 Upper Global Mask Long	XXXXh
CAN1LGML	EF0Ah	CAN1 Lower Global Mask Long	XXXXh
CAN1UMLM	EF0Ch	CAN1 Upper Mask Last Message	XXXXh
CAN1LMLM	EF0Eh	CAN1 Lower Mask Last Message	XXXXh
CAN1MCR1--15	EF10--EFF0h	CAN1 Message Control Register 1 to 15	XXXXh
CAN1UAR1--15	EF12--EFF2h	CAN1 Upper Arbitration Register 1 to 15	XXXXh
CAN1LAR1--15	EF14--EFF4h	CAN1 Lower Arbitration register 1 to 15	XXXXh
CAN1MO1--15	EF1x--EFFxh	CAN1 Message Object 1 to 15	XXXXh

21.7 - Special Notes

PEC Pointer Registers

The source and destination pointers for the peripheral event controller are mapped to a special area within the internal RAM. Pointers that are not occupied by the PEC may therefore be used like normal RAM. During Power Down mode or any short reset the PEC pointers are preserved.

The PEC and its registers are described in chapter "Interrupt and Trap Functions".

GPR Access in the ESFR Area

The locations 00'F000h...00'F01Eh within the ESFR area are reserved and provide access to the current register bank via short register addressing modes. The GPRs are mirrored to the ESFR area which allows access to the current register bank even after switching register spaces (see example below).

```
MOV  R5, DP3 ;GPR access via SFR area
EXTR #1
MOV  R5, ODP3;GPR access via ESFR area
```

Writing Byte to SFRs

All special function registers may be accessed word wise or byte wise (some of them even bit wise). Reading byte from word SFRs is a non-critical operation. However, when writing byte to word SFRs the complementary byte of the respective SFR is cleared with the write operation.

21.8 - Identification Registers

The ST10F280 have four Identification registers, mapped in ESFR space. These registers contain:

– A manufacturer identifier.

– A chip identifier with its revision.

– A internal Flash and size identifier.

– Programming voltage description.

IDMANUF (F07Eh / 3Fh) ESFR Reset Value: 0401h

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

Bit	Function
MANUF	**Manufacturer Identifier** 020h: STMicroelectronics manufacturer (JTAG worldwide normalization).

IDCHIP (F07Ch / 3Eh) ESFR Reset Value: 118Xh

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

Bit	Function
IDCHIP	**Device Identifier** 118h: ST10F280.
REVID	**Device Revision Identifier** Xh: According to revision number.

IDMEM (F07Ah / 3Dh) ESFR Reset Value: 3080h

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

Bit	Function
MEMSIZE	**Internal Memory Size** Internal Memory size is 4 x (MEMSIZE) (in Kbyte) - 080h for ST10F280 (256 Kbytes)
MEMTYP	**Internal Memory Type** 3h for ST10F280 (Flash memory)

IDPROG (F078h / 3Ch) ESFR Reset Value: 0040h

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

Bit	Function
PROGVDD	**Programming V_{DD} Voltage** V_{DD} voltage when programming EPROM or FLASH devices is calculated using the following formula: V_{DD} = 20 x [PROGVDD] / 256 (volts) - 40h for ST10F280 (5V).
PROGVPP	**Programming V_{PP} Voltage** (no need of external V_{PP}) - 00h

Note 1. All identification words are read only registers.

22 - SYSTEM PROGRAMMING

Constructs for modularity, loops, and context switching have been built into the ST10F280 instruction set. Many commonly used instruction sequences have been simplified. The following programming features are available to the programmer.

Instructions Provided as Subsets of Instructions

In many cases, instructions found in other microcontrollers are provided as subsets of more powerful instructions in the ST10F280.

This provides the same functionality, while decreasing the hardware requirement and decreasing decode complexity. These instructions can be built in macros to aid assembly programming.

Directly substitutable instructions are known instructions from other microcontrollers that can be replaced by the following instructions of the ST10F280:

Substituted Instruction	ST10F280 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 byte within word

Modification of system flags is performed by using bit set or bit clear instructions (BSET, BCLR). All bit and word instructions can access the PSW register, so no instructions like CLEAR CARRY or ENABLE INTERRUPTS are required.

External memory data access does not require special instructions to load data pointers or explicitly load and store external data.

The ST10F280 provides a unified memory architecture and its on-chip hardware automatically detects accesses to internal RAM, GPRs, and SFRs.

Multiplication and Division

Multiplication and division of words and double words is provided through multiple cycle instructions implementing a Booth algorithm. Each instruction implicitly uses the 32-bit register MD (MDL = lower 16 bits, MDH = upper 16 bits).

The MDRIU flag (Multiply or Divide Register In Use) in register MDC is set whenever either half of this register is written to or when a multiply/divide instruction is started. It is cleared whenever the MDL register is read.

Because an interrupt can be acknowledged before the contents of register MD are saved, this flag is required to alert interrupt routines, which require the use of the multiply/divide hardware, so they can preserve register MD.

This register, however, only needs to be saved when an interrupt routine requires use of the MD register and a previous task has not saved the current result. This flag is easily tested by the Jump-on bit instructions.

Multiplication or division is simply performed by specifying the correct (signed or unsigned) version of the multiply or divide instruction. The result is then stored in register MD.

The overflow flag (V) is set if the result from a multiply or divide instruction is greater than 16-bit. This flag can be used to determine whether both word halves must be transferred from register MD.

The high portion of register MD (MDH) must be moved into the register file or memory first, in order to ensure that the MDRIU flag reflects the correct state.

The following instruction sequence performs an unsigned 16 by 16-bit multiplication:

```
            ...
SAVE:       JNB     MDRIU, START    ;Test if MD was in use.
            SCXT    MDC, #0010H     ;Save and clear control register, leaving
                                    MDRIU set
                                    ;(only req for interrupted multiply/divide
                                    instructions)
            BSET    SAVED           ;Indicate the save operation
            PUSH    MDH             ;Save previous MD contents...
            PUSH    MDL             ;...on system stack
START:      MULU    R1, R2          ;Multiply 16·16 unsigned, Sets MDRIU
            JMPR    cc_NV, COPYL    ;Test for only 16 Bit result
            MOV     R3, MDH         ;Move high portion of MD
COPYL:      MOV     R4, MDL         ;Move low portion of MD, Clears MDRIU
RESTORE:    JNB     SAVED, DONE     ;Test if MD registers were saved
            POP     MDL             ;Restore registers
            POP     MDH
            POP     MDC
            BCLR    SAVED           ;Multiplication is completed, program
                                    continues
DONE:       ...
```

The above save sequence and the restore sequence after COPYL are only required if the current routine could have interrupted a previous routine which contained a MUL or DIV instruction. Register MDC is also saved because it is possible that a previous routine's Multiply or Divide instruction was interrupted while in progress. In this case the information about how to restart the instruction is contained in this register. Register MDC must be cleared to be correctly initialized for a subsequent multiplication or division. The old MDC contents must be popped from the stack before the RETI instruction is executed.

For a division the user must first move the dividend into the MD register. If a 16 by 16-bit division is specified, only the low portion of register MD must be loaded.

The result is also stored into register MD. The low portion (MDL) contains the integer result of the division, while the high portion (MDH) contains the remainder.

The following instruction sequence performs a 32 by 16-bit division:

```
        MOV     MDH, R1         ;Move dividend to MD register. Sets MDRIU
        MOV     MDL, R2         ;Move low portion to MD
        DIV     R3              ;Divide 32/16 signed, R3 holds the divisor
        JMPR    cc_V, ERROR     ;Test for divide overflow
        MOV     R3, MDH         ;Move remainder to R3
        MOV     R4, MDL         ;Move integer result to R4. Clears MDRIU
```

Whenever a multiply or divide instruction is interrupted while in progress, the address of the interrupted instruction is pushed onto the stack and the MULIP flag in the PSW of the interrupting routine is set. When the interrupt routine is exited with the RETI instruction, this bit is implicitly tested before the old PSW is popped from the stack. If MULIP='1' the multiply/divide instruction is re-read from the location popped from the stack (return address) and will be completed after the RETI instruction has been executed.

Note The MULIP flag is part of the **context of the interrupted task**. When the interrupting routine does not return to the interrupted task (for example when a scheduler switches to another task) the MULIP flag must be set or cleared according to the context of the task that is switched to.

BCD Calculations

No direct support for BCD calculations is provided in the ST10F280. 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.

22.1 - Stack Operations

The ST10F280 supports two types of stacks. The system stack is used implicitly by the controller and is located in the internal RAM. The user stack provides stack access to the user in either the internal or external memory. Both stack types grow from high memory addresses to low memory addresses.

Internal System Stack

A system stack is provided to store return vectors, segment pointers, and processor status for procedures and interrupt routines. A system register, SP, points to the top of the stack. This pointer is decremented when data is pushed onto the stack, and incremented when data is popped.

The internal system stack can also be used to temporarily store data or pass it between subroutines or tasks. Instructions are provided to push or pop registers on/from the system stack. However, in most cases the register banking scheme provides the best performance for passing data between multiple tasks.

Note The system stack allows the storage of words only. Byte must either be converted to word or the respective other byte must be disregarded. Register SP can only be loaded with even byte addresses (The LSB of SP is always '0').

Detection of stack overflow/underflow is supported by two registers, STKOV (Stack Overflow Pointer) and STKUN (Stack Underflow Pointer). Specific system traps (Stack Overflow trap, Stack Underflow trap) will be entered whenever the SP reaches either boundary specified in these registers.

The contents of the stack pointer are compared to the contents of the overflow register, whenever the SP is DECREMENTED either by a CALL, PUSH or SUB instruction. An overflow trap will be entered, when the SP value is less than the value in the stack overflow register.

The contents of the stack pointer are compared to the contents of the underflow register, whenever the SP is INCREMENTED either by a RET, POP or ADD instruction. An underflow trap will be entered, when the SP value is greater than the value in the stack underflow register.

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

In many cases the user will place a software reset instruction (SRST) into the stack underflow and overflow trap service routines. This is an easy approach, which does not require special programming.

However, this approach assumes that the defined internal stack is sufficient for the current software and that exceeding its upper or lower boundary represents a fatal error.

It is also possible to use the stack underflow and stack overflow traps to cache portions of a larger external stack. Only the portion of the system stack currently being used is placed into the internal memory, thus allowing a greater portion of the internal RAM to be used for program, data or register banking. This approach assumes no error but requires a set of control routines (see below).

Circular (Virtual) Stack

This basic technique allows pushing until the overflow boundary of the internal stack is reached. At this point a portion of the stacked data must be saved into external memory to create space for further stack pushes.

This is called "stack flushing". When executing a number of return or pop instructions, the upper boundary (since the stack empties upward to higher memory locations) is reached. The entries that have been previously saved in external memory must now be restored.

This is called "stack filling". Because procedure call instructions do not continue to nest infinitely and call and return instructions alternate, flushing and filling normally occurs very infrequently. If this is not true for a given program environment, this technique should not be used because of the overhead of flushing and filling.

The basic mechanism is the transformation of the addresses of a virtual stack area, controlled via registers SP, STKOV and STKUN, to a defined physical stack area within the internal RAM via hardware. This virtual stack area covers all possible locations that SP can point to, from 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 bit-field STKSZ in register SYSCON (see below).

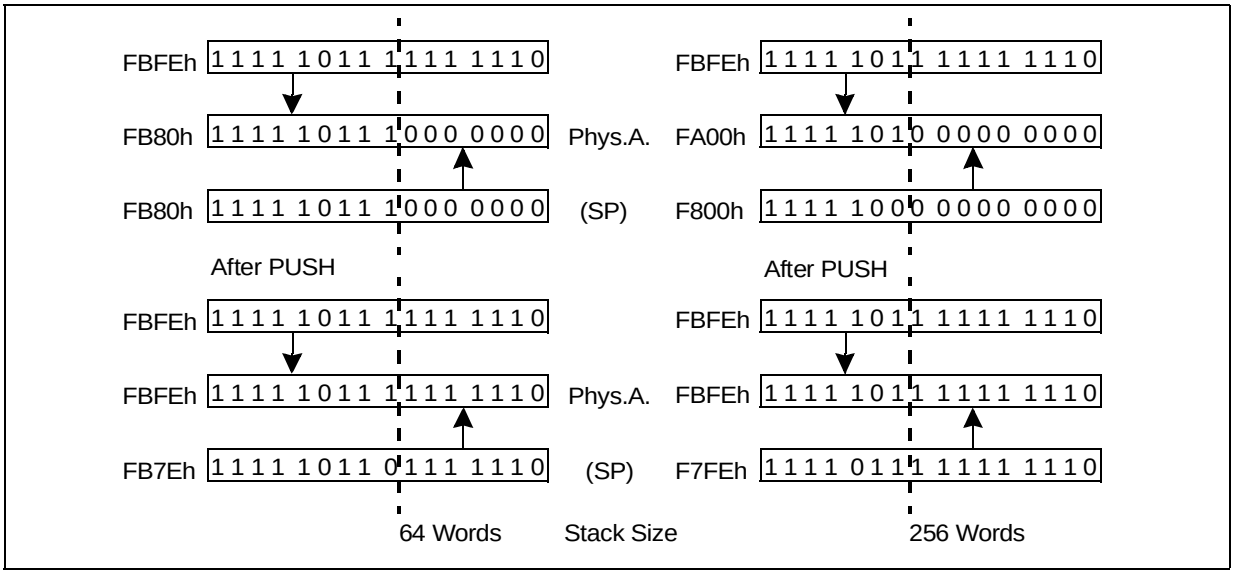
Table 64 : Stack size selection

(STKSZ)	Stack Size (words)	Internal RAM Addresses (words) of Physical Stack	Significant bit 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
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	512	00'FBFEh...00'F800h (not for 1 Kbyte IRAM)	SP.9...SP.0
1 0 1 b	---	Reserved. Do not use this combination.	---
1 1 0 b	---	Reserved. Do not use this combination.	---
1 1 1 b	1024	00'FDFEh...00'F600h (Note: No circular stack)	SP.11...SP.0

The virtual stack addresses are transformed to physical stack addresses by concatenating the significant bit of the stack pointer register SP (see table) with the complementary most significant bit of the upper limit of the physical stack area (00'FBFEh). This transformation is done via hardware (see Figure 165).

The reset values (STKOV=FA00h, STKUN=FC00h, SP=FC00h, STKSZ=000b) map the virtual stack area directly to the physical stack area and allow using the internal system stack without any changes, provided that the 256 word area is not exceeded.

Figure 165 : Physical stack address generation



The following example demonstrates the circular stack mechanism which is also an effect of this virtual stack mapping: First, register R1 is pushed onto the lowest physical stack location according to the selected maximum stack size. With the following instruction, register R2 will be pushed onto the highest physical stack location although the SP is decremented by 2 as for the previous push operation.

```
MOV SP, #0F802h   ; Set SP before last entry of physical stack of 256 Words
...               ; (SP) = F802h: Physical stack address = FA02h
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 wrap around from the end of the defined area to its beginning. When flushing and filling the internal stack, this circular stack mechanism only requires to move that portion of stack data which is really to be re-used (the upper part of the defined stack area) instead of the whole stack area. Stack data that remain in the lower part of the internal stack need not be moved by the distance of the space being flushed or filled, as the stack pointer automatically wraps around to the beginning of the freed part of the stack area.

Note This circular stack technique is applicable for stack sizes of 32 to 512 words (STKSZ = '000b' to '100b'), it does not work with option STKSZ = '111b', which uses the complete internal RAM for system stack.
In the latter case the address transformation mechanism is deactivated.

When a boundary is reached, the stack underflow or overflow trap is entered, where the user moves a predetermined portion of the internal stack to or from the external stack. The amount of data transferred is determined by the average stack space required by routines and the frequency of calls, traps, interrupts and returns. In most cases this will be approximately one quarter to one tenth the size of the internal stack. Once the transfer is complete, the boundary pointers are updated to reflect the newly allocated space on the internal stack. Thus, the user is free to write code without concern for the internal stack limits. Only the execution time required by the trap routines affects user programs.

The following procedure initializes the controller for usage of the circular stack mechanism:

– Specify the size of the physical system stack area within the internal RAM (bit-field STKSZ in register SYSCON).

– Define two pointers, which specify the upper and lower boundary of the external stack. These values are then tested in the stack underflow and overflow trap routines when moving data.
– Set the stack overflow pointer (STKOV) to the limit of the defined internal stack area plus six words (for the reserved space to store two interrupt entries).

The internal stack will now fill until the overflow pointer is reached. After entry into the overflow trap procedure, the top of the stack will be copied to the external memory. The internal pointers will then be modified to reflect the newly allocated space. After exiting from the trap procedure, the internal stack will wrap around to the top of the internal stack, and continue to grow until the new value of the stack overflow pointer is reached.

When the underflow pointer is reached while the stack is emptied the bottom of stack is reloaded from the external memory and the internal pointers are adjusted accordingly.

Linear Stack

The ST10F280 also 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 procedures to handle data transfers for a circular stack. However, this method also leaves less RAM space for variables or code. The RAM area that may effectively be consumed by the system stack is defined via the STKUN and STKOV pointers. The underflow and overflow traps in this case serve for fatal error detection only. For the linear stack option all modifiable bit 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 below the IRAM area (ESFR space and reserved area) and within address range 00'FE00h and 00'FFFEh (SFR space).
Otherwise unpredictable results will occur.

User Stacks

User stacks provide the ability to create task specific data stacks and to off-load data from the system stack. The user may push both byte 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 user stack. This mode is available to most instructions with some restrictions.

For MOV instructions, any word GPR can be used as user stack pointer.

For arithmetic, logical and compare instructions, only GPRs R0-R3 can be used.

Rb, [Rw+] or Rw, [Rw+]: Post-increment Indirect Addressing. Used to pop one byte or word from a user stack. This mode is only available for MOV instructions and can specify any GPR as the user stack pointer.

22.2 - Register Banking

Register banking provides the user with an extremely fast method to switch user context. A single instruction cycle instruction saves the old bank and enters a new register bank. Each register bank may assign up to 16 registers. Each register bank should be allocated during coding based on the needs of each task. Once the internal memory has been partitioned into a register bank space, internal stack space and a global internal memory area, each bank pointer is then assigned. Thus, upon entry into a new task, the appropriate bank pointer is used as the operand for the SCXT (switch context) instruction. Upon exit from a task a simple POP instruction to the context pointer (CP) restores the previous task's register bank.

22.3 - Procedure Call Entry and Exit

To support modular programming a procedure mechanism is provided to allow coding of frequently used portions of code into subroutines. The CALL and RET instructions store and restore the value of the instruction pointer (IP) on the system stack before and after a subroutine is executed.

Procedures may be called conditionally with instructions CALLA or CALLI, or be called unconditionally using instructions CALLR or CALLS.

Note Any data pushed onto the system stack during execution of the subroutine must be popped before the RET instruction is executed.

Passing Parameters on the System Stack

Parameters may be passed via the system stack through PUSH instructions before the subroutine is called, and POP instructions during execution of the subroutine. Base plus offset indirect addressing also permits access to parameters without popping these parameters from the stack during execution of the subroutine. Indirect addressing provides a mechanism of accessing data referenced by data pointers, which are passed to the subroutine. In addition, two instructions have been implemented to allow one parameter to be passed on the system stack without additional software overhead.

The PCALL (push and call) instruction first pushes the 'reg' operand and the IP contents onto the system stack and then passes control to the subroutine specified by the 'caddr' operand.

When exiting from the subroutine, the RETP (return and pop) instruction first pops the IP and then the 'reg' operand from the system stack and returns to the calling program.

Cross Segment Subroutine Calls

Calls to subroutines in different segments require the use of the CALLS (call inter-segment subroutine) instruction. This instruction preserves both the CSP (code segment pointer) and IP on the system stack.

Upon return from the subroutine, a RETS (return from inter-segment subroutine) instruction must be used to restore both the CSP and IP. This ensures that the next instruction after the CALLS instruction is fetched from the correct segment.

Note It is possible to use CALLS within the same segment, but still two words of the stack are used to store both the IP and CSP.

Providing Local Registers for Subroutines

For subroutines which require local storage, the following methods are provided:

Alternate bank of registers: Upon entry into a subroutine, it is possible to specify a new set of local registers by executing the SCXT (switch context) instruction. This mechanism does not provide a method to recursively call a subroutine.

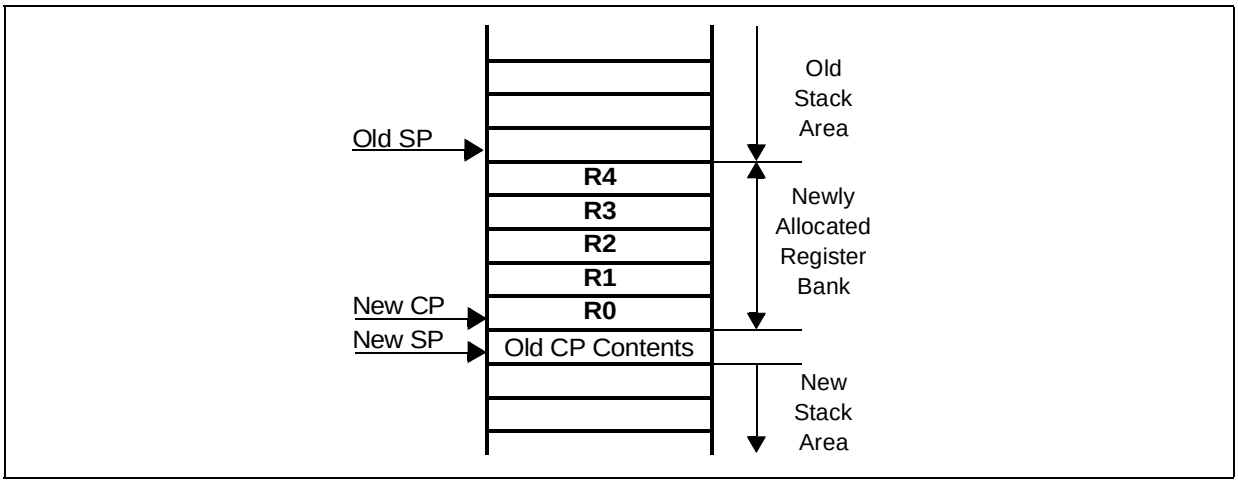
Saving and restoring of registers: To provide local registers, the contents of the registers which are required for use by the subroutine can be pushed onto the stack and the previous values be popped before returning to the calling routine. This is the most common technique used today and it does provide a mechanism to support recursive procedures. This method, however, requires two instruction cycles per register stored on the system stack (one cycle to PUSH the register, and one to POP the register).

Use of the system stack for local registers: It is possible to use the SP and CP to set up local subroutine register frames. This enables subroutines to dynamically allocate local variables as needed within two instruction 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 regist er 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. The system stack is growing downwards, while the register bank is growing upwards.

Figure 166 : Local registers



The software to provide the local register bank for the example above Figure 166 is very compact:
After entering the subroutine:

```
SUB       SP, #10D   ; Free 5 Words in the current system stack
SCXT      CP, SP     ; Set the new register bank pointer
```

Before exiting the subroutine:

```
POP       CP         ; Restore the old register bank
ADD       SP, #10D   ; Release the 5 Word of the current system stack
```

22.4 - Table Searching

A number of features have been included to decrease the execution time required to search tables. First, branch delays are eliminated by the branch target cache after the first iteration of the loop. Second, in non-sequentially searched tables, the enhanced performance of the ALU allows more complicated hash algorithms to be processed to obtain better table distribution. For sequentially searched tables, the auto-increment indirect addressing mode and the E (end of table) flag stored in the PSW decrease the number of overhead instructions executed in the loop.

The two examples below illustrate searching ordered tables and non-ordered tables, respectively:

```
          MOV      R0, #BASE      ;Move table base into R0
LOOP:     CMP      R1, [R0+]      ;Compare target to table entry
          JMPR     cc_SGT, LOOP   ;Test whether target has not been found
The last entry in the table must be greater than the largest possible target.
          MOV      R0, #BASE      ;Move table base into R0
LOOP:     CMP      R1, [R0+]      ;Compare target to table entry
          JMPR     cc_NET, LOOP   ;Test whether target is not found AND the
                                  end of table...
                                  ;...has not been reached.
```

Note The last entry in the table must be equal to the lowest signed integer (8000h).

22.5 - Peripheral Control and Interface

All communication between peripherals and the CPU is performed either by PEC transfers to and from internal memory, or by explicitly addressing the SFRs associated with the specific peripherals. After resetting the ST10F280 all peripherals (except the watchdog timer) are disabled and initialized to default values. A desired configuration of a specific peripheral is programmed using MOV instructions of either constants or memory values to specific SFRs. Specific control flags may also be altered via bit instructions.

Once in operation, the peripheral operates autonomously until an end condition is reached at which time it requests a PEC transfer or requests CPU servicing through an interrupt routine. Information may also be polled from peripherals through read accesses to SFRs or bit operations including branch tests on specific control bit in SFRs. To ensure proper allocation of peripherals among multiple tasks, a portion of the internal memory has been made bit addressable to allow user semaphores. Instructions have also been provided to lock out tasks via software by setting or clearing user specific bit and conditionally branching based on these specific bit.

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.

22.6 - Floating Point Support

All floating point operations are performed using software. Standard multiple precision instructions are used to perform calculations on data types that exceed the size of the ALU. Multiple bit rotate and logic instructions allow easy masking and extracting of portions of floating point numbers.

To decrease the time required to perform floating point operations, two hardware features have been implemented in the CPU core. First, the PRIOR instruction aids in normalizing floating point numbers by indicating the position of the first set bit in a GPR. This result can the be used to rotate the floating point result accordingly.

The second feature aids in properly rounding the result of normalized floating point numbers through the overflow (V) flag in the PSW. This flag is set when a one is shifted out of the carry bit during shift right operations. The overflow flag and the carry flag are then used to round the floating point result based on the desired rounding algorithm.

22.7 - Trap / Interrupt Entry and Exit

Interrupt routines are entered when a requesting interrupt has a priority higher than the current CPU priority level. Traps are entered regardless of the current CPU priority. When either a trap or interrupt routine is entered, the state of the machine is preserved on the system stack and a branch to the appropriate trap/interrupt vector is made.

All trap and interrupt routines require the use of the RETI (return from interrupt) instruction to exit from the called routine.

This instruction restores the system state from the system stack and then branches back to the location where the trap or interrupt occurred.

22.8 - Inseparable Instruction Sequences

The instructions of the ST10F280 are very efficient (most instructions execute in one instruction cycle) and even the multiplication and division are interruptible in order to minimize the response latency to interrupt requests (internal and external). In many microcontroller applications this is vital.

Some special occasions, however, require certain code sequences (like semaphore handling) to be non-interruptible to function properly.

This can be provided by inhibiting interrupts during the respective code sequence by disabling and enabling them before and after the sequence.

The necessary overhead may be reduced by means of the ATOMIC instruction which allows locking 1...4 instructions to an inseparable code sequence, during which the interrupt system (standard interrupts and PEC requests) **and Class A Traps** (NMI, stack overflow/underflow) are disabled. A **Class B Trap** (illegal opcode, illegal bus access, etc.), however, will interrupt the atomic sequence, since it indicates a severe hardware problem.

The interrupt inhibit caused by an ATOMIC instruction gets active immediately, and no other instruction will enter the pipeline except the one that follows the ATOMIC instruction, and no interrupt request will be serviced in between.

All instructions requiring multiple cycles or hold states are regarded as one instruction in this sense (example MUL is one instruction). Any instruction type can be used within an inseparable code sequence.

```
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
                                the ATOMIC instruction sequence
```

22.9 - Overriding the DPP Addressing Mechanism

The standard mechanism to access data locations uses one of the four data page pointers (DPPx), which selects a 16 Kbytes data page, and a 14-bit offset within this data page. The four DPPs allow immediate access to up to 64 Kbytes of data. In applications with big data arrays, especially in HLL applications using large memory models, this may require frequent reloading of the DPPs, even for single accesses.

The EXTP (extend page) instruction allows switching to an arbitrary data page for 1...4 instructions without having to change the current DPPs.

```
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 allows switching to a 64 Kbyte segment oriented data access scheme for 1...4 instructions without having to change the current DPPs. In this case all 16 bits of the operand address are used as segment offset, with the segment taken from the EXTS instruction. This greatly simplifies address calculation with continuous data like huge arrays in "C".

```
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.

Short Addressing in the Extended SFR (ESFR) Space

The short addressing modes of the ST10F280 (REG or bitOFF) implicitly access the SFR space. The additional ESFR space would have to be accessed via long addressing modes (MEM or [Rw]).

The EXTR (extend register) instruction redirects accesses in short addressing modes to the ESFR space for 1...4 instructions, so the additional registers can be accessed this way, too.

The EXTPR and EXTSR instructions combine the DPP override mechanism with the redirection to the ESFR space using a single instruction.

Note Instructions EXTR, EXTPR and EXTSR inhibit interrupts the same way as ATOMIC.
The switching to the ESFR area and data page overriding is checked by the development tools or handled automatically.

Nested Locked Sequences

Each of the described extension instruction and the ATOMIC instruction starts an internal "extension counter" counting the effected instructions. When another extension or ATOMIC instruction is contained in the current locked sequence this counter is restarted with the value of the new instruction. This allows the construction of locked sequences longer than 4 instructions.

Note Interrupt latencies may be increased when using locked code sequences.
PEC requests are not serviced during idle mode, if the IDLE instruction is part of a locked sequence.

22.10 - Handling the Internal Flash

The ST10F280 provides and controls a 256 Kbytes internal on-chip Flash memory that may store code as well as data. Access to this internal Flash area is controlled during the reset configuration and via software. The Flash area may be mapped to segment 0, to segment 1 or may be disabled at all.

Note The Flash memory always occupies an address area of 256 Kbytes.

Configuration During Reset

The default memory configuration of the ST10F280 Memory is determined by the state of the $\overline{EA}$ pin at reset. This value is stored in the Internal ROM Enable bit (named ROMEN) of the SYSCON register.

When the $\overline{EA}$ pin is high during reset (default value), the internal Flash is globally enabled and the first 32 Kbytes are mapped in segment "0". The first instruction are fetched from the internal Flash from locations 00'0000h.

When the $\overline{EA}$ pin is low during reset (ROMEN = 0), the internal Flash is disabled and external ROM is used for startup control.

Mapping the Internal Flash Area

When internal Flash is disabled on reset the first instructions are fetched from external memory locations 00'0000h. The Flash memory can later be enabled by setting the ROMEN bit of SYSCON to 1. The code performing this setting must not run from a segment of the external ROM to be replaced by a segment of the Flash memory, otherwise unexpected behaviour may occur.

For example, if external ROM code is located in the first 32 Kbytes of segment 0, the first 32 Kbytes of the Flash must then be enabled in segment 1. This is done by setting the ROMS1 bit of SYSCON to 0 before or simultaneously with setting of ROMEN bit. This must be done in the externally supplied program before the execution of the EINIT instruction.

If program execution starts from external memory, but access to the Flash memory mapped in segment 0 is later required, then the code that performs the setting of ROMEN bit must be executed either in the segment 0 but above address 00'8000h, or from the internal RAM.

Bit ROMS1 only affects the mapping of the first 32 Kbytes of the Flash memory. All other parts of the Flash memory (addresses 01'8000h - 04'FFFFh) remain unaffected.

The SGTDIS Segmentation Disable / Enable must also be set to 0 to allow the use of the full 256 Kbytes of on-chip memory in addition to the external boot memory. The correct procedure on changing the segmentation registers must also be observed to prevent an unwanted trap condition:

– Instructions that configure the internal memory must only be executed from external memory or from the internal RAM.
– An Absolute Inter-Segment Jump (JMPS) instruction must be executed after Flash enabling, to the next instruction, even if this next instruction is located in the consecutive address.
– Whenever the internal Memory is disabled, enabled or remapped, the DPPs must be explicitly (re)loaded to enable correct data accesses to the internal memory and/or external memory.

When starting from external memory, the interrupt/trap vector table, which uses locations 00'0000h through 00'01FFh, of the external memory and may therefore be modified, so the system software may now change interrupt/trap handlers according to the current condition of the system.

The internal Flash can still be used for fixed software routines like I/O drivers, math libraries, application specific invariant routines, tables, etc. This combines the advantage of an integrated non-volatile memory with the advantage of a flexible, adaptable software system.

Enabling and Disabling the Internal Flash Area After Reset

If the internal Flash does not contain an appropriate start-up code, the system may be booted from external memory, while the internal Flash is enabled afterwards to provide access to library routines, tables, etc.

If the internal Flash only contains the start-up code and/or test software, the system may be booted from internal Flash, which may then be disabled, after the software has switched to executing from external memory, in order to free the address space occupied by the internal Flash area, which is now unnecessary.

22.11 - Pits, Traps and Mines

Although handling the internal ROM or Flash provides powerful means to enhance the overall performance and flexibility of a system, extreme care must be taken in order to avoid a system crash. Instruction memory is the most crucial resource for the ST10F280 and it must be made sure that it never runs out of it.

The following precautions help to take advantage of the methods mentioned above without jeopardizing system security.

Internal Flash access after reset: When the first instructions are to be fetched from internal Flash (EA='1'), the memory must contain a valid reset vector and valid code at its destination.

Mapping the internal Flash to segment 1: Due to instruction pipelining, any new Flash mapping will at the earliest become valid for the second instruction after the instruction which has changed the Flash mapping. To enable accesses to the Flash after mapping a branch to the newly selected Flash area (JMPS) and reloading of all data page pointers is required.
This also applies to re-mapping the internal Flash to segment 0.

Enabling the internal Flash after reset: When enabling the internal Flash after having booted the system from external memory, note that the ST10F280 will then access the internal Flash using the current segment offset, rather than accessing external memory.

Disabling the internal Flash after reset: When disabling the internal Flash after having booted the system from there, note that the ST10F280 will not access external memory before a jump to segment 0 (in this case) is executed.

General Rules

When mapping the Flash no instruction or data accesses should be made to the internal Flash, otherwise unpredictable results may occur.

To avoid these problems, the instructions that configure the internal Flash should be executed from external memory or from the internal RAM.

Whenever the internal Flash is disabled, enabled or re-mapped the DPPs must be explicitly (re)loaded to enable correct data accesses to the internal Flash and/or external memory.

23 - KEY WORD INDEX

24 - INDEX OF REGISTERS

25 - REVISION HISTORY

25.1 - Revision 1.0 of 26th of February 2002

This is revision 1.0 of this document, released on 26th of February 2002.

End of file – 26th of February 2002.

25.2 - Revision 2.0 of 25th of September 2013

Updated Disclaimer

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST's terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST'S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

ST PRODUCTS ARE NOT DESIGNED OR AUTHORIZED FOR USE IN: (A) SAFETY CRITICAL APPLICATIONS SUCH AS LIFE SUPPORTING, ACTIVE IMPLANTED DEVICES OR SYSTEMS WITH PRODUCT FUNCTIONAL SAFETY REQUIREMENTS; (B) AERONAUTIC APPLICATIONS; (C) AUTOMOTIVE APPLICATIONS OR ENVIRONMENTS, AND/OR (D) AEROSPACE APPLICATIONS OR ENVIRONMENTS. WHERE ST PRODUCTS ARE NOT DESIGNED FOR SUCH USE, THE PURCHASER SHALL USE PRODUCTS AT PURCHASER'S SOLE RISK, EVEN IF ST HAS BEEN INFORMED IN WRITING OF SUCH USAGE, UNLESS A PRODUCT IS EXPRESSLY DESIGNATED BY ST AS BEING INTENDED FOR "AUTOMOTIVE, AUTOMOTIVE SAFETY OR MEDICAL" INDUSTRY DOMAINS ACCORDING TO ST PRODUCT DESIGN SPECIFICATIONS. PRODUCTS FORMALLY ESCC, QML OR JAN QUALIFIED ARE DEEMED SUITABLE FOR USE IN AEROSPACE BY THE CORRESPONDING GOVERNMENTAL AGENCY.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.
Information in this document supersedes and replaces all information previously supplied.
The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com